ปีการศึกษา 2530

8051 - CROSS ASSEMBLER

โดย

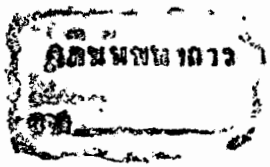
นาย ดิเรก อารีรัตน์

นาย อดิเรก สุพรรณวรรษา

อาจารย์ที่ปรึกษา

อาจารย์ วิทยา ทิพย์สุวรรณพร

อาจารย์ พูนศักดิ์ พัฒนไพรสณฑ์



ปริญญานิพนธ์ปีการศึกษา 2530

เรื่อง 8051 - CROSS ASSEMBLER

ผู้จัดทำ

1). นาย ดิเรก อารีรัตน์ เลขประจำตัว 296304

2). นาย อดิเรก สุพรรณวรรษา เลขประจำตัว 296325

ภาควิชา วัดคุมทางอุตสาหกรรม

สาขาวิชา เทคโนโลยีคอมพิวเตอร์อุตสาหกรรม

..... อาจารย์ที่ปรึกษา

(..... วิทยา ทิพย์สุวรรณพร .....)

..... อาจารย์ที่ปรึกษา

(..... พูนศักดิ์ นัฒนไพรสณฑ์ .....)

ปีการศึกษา 2530

8051 CROSS ASSEMBLER

นาย อดิเรก สุพรรณวรรษา

นาย ดิเรก อารีรัตน์

ผู้จัดทำ

อ. วิทยา ทิพย์สุวรรณพร

อ. พูนศักดิ์ พัฒนไพรสณฑ์

อาจารย์ที่ปรึกษา

บทคัดย่อ

โครงงานพัฒนาระบบไมโครคอมพิวเตอร์ชิพเดียว 8051 นี้ ได้เริ่มต้นตั้งแต่การเขียนครอสแอสเซมเบลอร์ (ตัวแปลภาษาแอสเซมบลี้) ของ 8051 จากนั้นก็นำเอาออปเจคโค๊ดที่ได้ไปดาวโหลดผ่านคาร์ดดาวโหลด เข้าสู่ชิงเกิลบอร์ดของ 8051 เพื่อนำไปใช้งานต่อไป

สำหรับโครงงานพัฒนานี้ สิ่งที่ทำคือ ตัวครอสแอสเซมเบลอร์ , คาร์ดดาวโหลด 16 กิโลไบท์ และเขียนโปรแกรมควบคุมชิงเกิลบอร์ด 8051 อุปกรณ์ที่ใช้คือ เครื่องไมโครคอมพิวเตอร์ IBM และชิงเกิลบอร์ด 8051.

YEARS 1987

8051 CROSS ASSEMBLER

MR. ADIREK SUPANWASSA

MR. DIREK AREERAT

PRODUCER

MR. WITTAYA TIPSUWANNAPORN

MR. PHUNSAK PATTANAPRISON

ADVISOR

ABSTRACT

The object of this thesis was to develop 8051 Single-Board Microcomputer system. The study was started by writing 8051's cross assembler which was compiler of assembly language , and when the assembly language's program was compiled it would get object code. Then object code would be download on card download into 8051's single-board for the following application.

Cross assembler 8051 , Card download 16 Kbyte and writing program for controlling single-board 8051 were done for this project development. The device was used an IBM computer and single-board 8051.

สารบัญ

บทนำ

ในปัจจุบันเทคโนโลยีทางด้านคอมพิวเตอร์ก้าวหน้าไปอย่างไม่หยุดยั้ง ฉะนั้นเราจะต้องติดตามวิวัฒนาการเหล่านี้ เพื่อจะได้ไม่ล้าหลัง การศึกษาวิชาการทางด้านนี้จึงมีความจำเป็นมาก โครงงานนี้ก็เป็นส่วนหนึ่งซึ่งเป็นโครงงานที่ใช้พัฒนาไมโครโปรเซสเซอร์ตระกูล 8051 ซึ่งสามารถเขียนภาษา ASSEMBLY ของ 8051 บนเครื่องไมโครคอมพิวเตอร์ IBM จากนั้นก็นำโปรแกรมที่ได้ไปผ่านการ COMPILED จากโปรแกรม CROSS ASSEMBLER ที่เขียนขึ้น ก็จะได้ OBJECT CODE ของ 8051 ออกมา จากนั้นก็นำไป DOWNLOAD ผ่าน CARD DOWNLOAD 16 Kbyte ที่ทำขึ้นเพื่อโหลดไปยังชุดพัฒนา 8051 เพื่อให้ทำงานตามโปรแกรมต่อไป

ทางผู้จัดทำหวังว่า โครงงานนี้จะเป็นแนวทางให้ผู้ที่จะศึกษาไมโครโปรเซสเซอร์ตระกูล 8051 ได้นำไปใช้ประโยชน์ได้ไม่มากก็น้อย

สถาปัตยกรรมของ MCS-51 (MCS-51 ARCHITECTURE)

ไมโครโปรเซสเซอร์ตระกูล MCS-51 เป็น Single Chip 8 บิท โดยทั่วไปจะใช้เบอร์ 8051 อ้างถึงทุกตัวในตระกูลนี้ นอกจากนี้ยังมีเบอร์อื่นที่ Compatable คือเบอร์ 8031,8032,8751,8052 เป็นต้น โดยในตระกูลนี้จะใช้เทคโนโลยีใหม่ๆ เพื่อประหยัดกำลังงานไฟฟ้า (Power Reduction)

เบอร์	เทคโนโลยีที่ใช้ผลิต	On - Chip Program Memory	On - Chip Data Memory
8051AH	HMOS II	4K - ROM	128
8031AH	HMOS II	NONE	128
8751H	HMOS I	4K - EPROM	128
80C51	CHMOS	4K - ROM	128
80C31	CHMOS	NONE	128
8052	HMOS II	8K - ROM	256
8032	HMOS II	NONE	256

ตาราง MCS-51 Family Members

การจัดการหน่วยความจำ (Memory Organization)

8051 ได้แยกหน่วยความจำที่ใช้ออกเป็น 2 ส่วนคือ

1. หน่วยความจำโปรแกรม (Program memory)
2. หน่วยความจำข้อมูล (Data memory)

หน่วยความจำโปรแกรมสามารถขยายได้ถึง 64 Kbyte ส่วนหน่วยความจำข้อมูลมี 128 byte บนชิพ พื้นที่ 128 byte นี้ใช้เป็นรีจิสเตอร์ทำหน้าที่พิเศษ (Special Function Register <SFR>) และ 64 Kbyte สำหรับหน่วยความจำข้อมูลภายนอก (External Data memory)

รายละเอียดของรีจิสเตอร์หน้าที่พิเศษ

- แอคคิวมูเลเตอร์ (Accumulator) อ้างถึงโดยใช้ ACC หรือ A
- รีจิสเตอร์ B (B register) ใช้ในการคูณหรือหาร

- รีจิสเตอร์เก็บสถานะ (Program Status Word) <PSW> บรรจุสถานะแฟลก

(บิทสูง) (บิทต่ำ)

CY	AC	FØ	RS1	RSØ	OV	-	P

สัญญลักษณ์	ตำแหน่ง	รายละเอียด
CY	PSW.7	Carry flag
AC	PSW.6	Auxiliary Carry flag ถ้ามี carry เข้าหรือออกจากบิทที่ 3 จะเซ็ท
FØ	PSW.5	Flag Ø ถูกใช้โดยผู้ใช้ (User)
RS1	PSW.4	รีจิสเตอร์เลือก BANK บิท 1
RSØ	PSW.3	รีจิสเตอร์เลือก BANK บิท Ø
OV	PSW.2	Overflow flag เมื่อเกิด overflow จะเซ็ท
-	PSW.1	reserved
P	PSW.Ø	Parity flag แสดงจำนวนบิทที่เป็น 1 ในแอคคิวมูเลเตอร์

RS1	RSØ	BANK	ตำแหน่ง
Ø	Ø	Ø	ØØh - Ø7h
Ø	1	1	Ø8h - Øfh
1	Ø	2	1Øh - 17h
1	1	3	18h - 1fh

- สแตคพอยเตอร์ (Stack Pointer) เป็นรีจิสเตอร์ 8 บิท ซึ่งจะเพิ่มค่าก่อนที่จะเก็บข้อมูลระหว่างการ PUSH และ CALL โดยสแตคสูงสุดอยู่ที่ Ø7h หลังจากการรีเซท ฉะนั้นส่วนที่เริ่มเก็บข้อมูลของสแตคคือ Ø8h

- ดาต้าพอยเตอร์ (Data Pointer) <DPTR> ประกอบด้วยไบท์สูง (DPH) และไบท์ต่ำ (DPL) โดยใช้อ้างได้ทั้ง 8 บิทแยกอิสระ และ 16 บิท

- พอร์ท Ø - 3 คือ P0,P1,P2,P3

- บัฟเฟอร์ข้อมูลอนุกรม (Serial Data Buffer) <SBUF> มี 2 รีจิสเตอร์แยกออกจากกัน คือบัฟเฟอร์ส่ง (Transmit buffer) และบัฟเฟอร์รับ (Receive buffer) เมื่อมีข้อมูลเข้ามายัง SBUF ก็จะมาเก็บที่บัฟเฟอร์ส่งซึ่งใช้

สำหรับส่งข้อมูลอนุกรม เมื่อข้อมูลถูกย้ายจาก SBUF มันจะมาจากบัฟเฟอร์รับ

- รีจิสเตอร์ควบคุม (Control Register) รีจิสเตอร์หน้าที่พิเศษ IP,IE, TMOD,TCON,T2CON,SCON และ PCON จะบรรจุบิทควบคุมและบิทสถานะสำหรับระบบอินเทอร์รัพท์ , ไทม์เมอร์/เคาน์เตอร์ , และพอร์ทอนุกรม

โครงสร้างของพอร์ทและการทำงาน

พอร์ท 4 พอร์ททั้งหมดใน 8051 เป็นพอร์ท 2 ทิศทาง (Bidirectional) คือพอร์ท P0 - P3 ขับเอาท์พุทและอินพุทบัฟเฟอร์ เอาท์พุทของพอร์ท 0 , 2 และอินพุทบัฟเฟอร์ของพอร์ท 0 จะถูกใช้ในการอ้างหน่วยความจำภายนอก ในการนำไปใช้นั้นเอาท์พุทของพอร์ท 0 จะถูกใช้เป็นไบท์ต่ำของแอดเดรสหน่วยความจำภายนอก เอาท์พุทพอร์ท 2 จะใช้เป็นไบท์สูงของแอดเดรสหน่วยความจำภายนอก

ส่วนพอร์ท 1 และ พอร์ท 3 มีหน้าที่หลายอย่างต่างกันดังนี้

ขาพอร์ท	หน้าที่การใช้งาน
P1.0	T2 (Timer/Counter 2 external input)
P1.1	T2RST (Timer/Counter 2 external reset input)
P3.0	RXD (Serial input port)
P3.1	TXD (Serial output port)
P3.2	-INT0 (External interrupt)
P3.3	-INT1 (External interrupt)
P3.4	T0 (Timer/Counter 0 external input)
P3.5	T1 (Timer/Counter 1 external input)
P3.6	-WR (External data memory write strobe)
P3.7	-RD (External data memory read strobe)

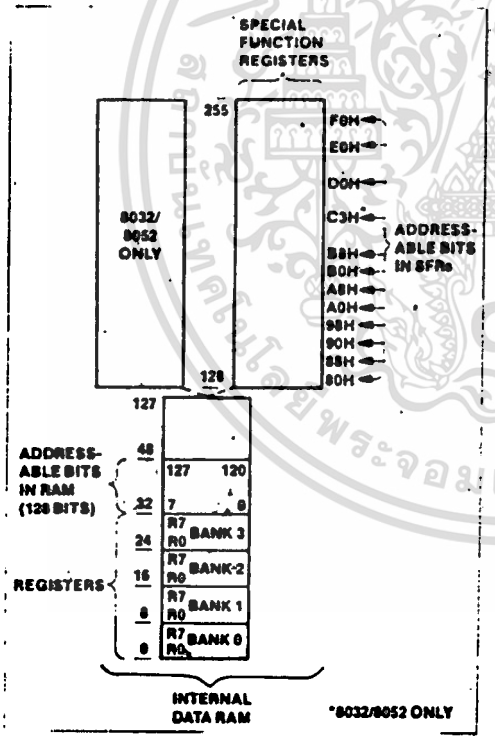
แอดเดรสหน่วยความจำโปรแกรม (Program memory address space)

หน่วยความจำโปรแกรม 64 Kbyte ประกอบด้วยหน่วยความจำภายในและหน่วยความจำภายนอก ถ้าขา -EA เป็น High 8051 จะใช้หน่วยความจำโปรแกรมภายในเว้นแต่แอดเดรสเกิน 0fffh (1fffh สำหรับ 8052) ตำแหน่ง 1000h ถึง 0fffh จะถูกเฟทจากหน่วยความจำโปรแกรมภายนอก ถ้าขา -EA เป็น Low 8051 จะเฟททุกคำสั่งจากหน่วยความจำโปรแกรมภายนอก โดย Program Counter 16 bit เป็นตัวอ้างแอดเดรส

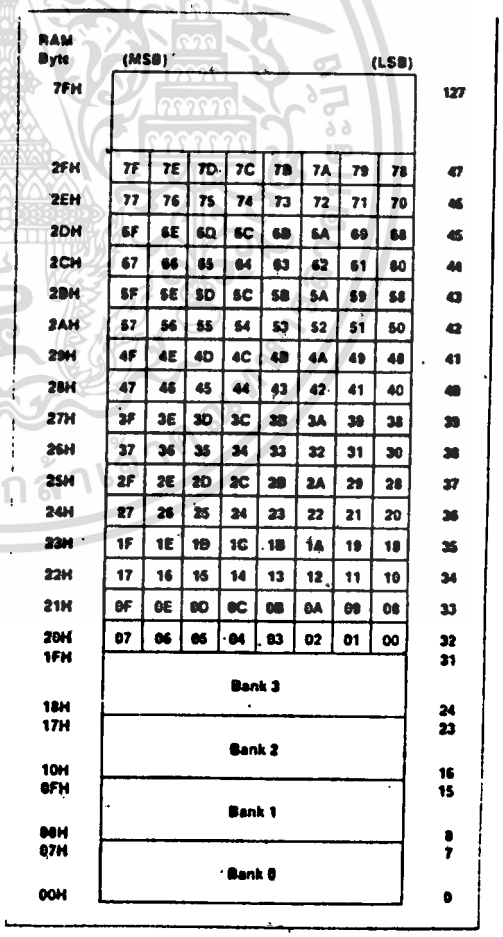
<u>แอดเดรสหน่วยความจำข้อมูล</u> (Data memory address space)

แอดเดรสหน่วยความจำข้อมูล ประกอบด้วยหน่วยความจำภายในและภายนอก หน่วยความจำภายนอกจะถูกใช้เมื่อ executed คำสั่ง MOVX

หน่วยความจำข้อมูลภายในแบ่งเป็น 3 ส่วนแยกกันและรวมกันเป็นกลุ่ม คือ : 128 ไบท์ของ RAM ; 128 ไบท์สูงของ RAM ; และ 128 ไบท์ซึ่งเป็นพื้นที่ของรีจิสเตอร์หน้าที่พิเศษ (SFR) ขณะที่พื้นที่ส่วนบนของ RAM และพื้นที่ SFR จะอยู่ในตำแหน่งแอดเดรสที่เหมือนกัน แต่การใช้จะต่างโหมดกิการอ้างแอดเดรสกัน (different addressing modes) ตามรูปข้างล่างแสดงหน่วยความจำภายใน 4 BANK (8 bit register) จะอยู่ที่ตำแหน่ง 0 - 31 ในพื้นที่ RAM ต่ำ จะมีเพียง BANK เดียว (ดูจาก 2 บิทใน PSW) ที่ใช้ได้ในเวลาหนึ่งๆ 16 ไบท์ถัดไปตำแหน่ง 32 - 47 จะบรรจุ 128 บิทตำแหน่งแอดเดรส ส่วนอีกรูปจะแสดงแอดเดรสเป็นบิทใน RAM พื้นที่ SFR จะมีตำแหน่งแอดเดรสเป็นบิท



รูป แอดเดรสหน่วยความจำข้อมูลภายในที่ว่าง



รูป ตำแหน่งแอดเดรสบิทหน้าที่พิเศษ

โหมดการอ้างแอดเดรส (Addressing Mode)

8051 มีโหมดการอ้างแอดเดรสอยู่ 5 โหมด คือ

* Register ;
* Direct ;
* Register Indirect ;
* Immediate ;
* Base - Register plus Index - Register Indirect.

ตาราง Addressing Method and Assosiated Memory Spaces

Register Addressing
- R0-R7
- ACC , B , CY (bit) ,DPTR
Direct Addressing
- Low 128 bytes ของ RAM ภายใน
- Special Function Registers
Register Indirect Addressing
- Internal RAM (@R1,@R0,SP)
- External Data Memory (@R1,@R0,@DPTR)
Immediate Addressing
- Program Memory
Base-Register plus Index-Register Indirect Addressing
- Program Memory (@DPTR+A , @PC+A)

ตาราง พื้นที่หน่วยความจำว่างซึ่งถูกเข้าถึงโดยโหมดการอ้างแอดเดรส

Source	Address
External Interrupt 0	0003h
Timer 0 Overflow	000Bh
External Interrupt 1	0013h

Source	Address
Timer 1 Overflow	001Bh
Serial Port	0023h
Timer 2 Overflow / TZEX	002Bh
Negative Transition	

การอ้างแอดเดรสโดยใช้รีจิสเตอร์ (Register Addressing)

การอ้างแอดเดรสแบบนี้ จะใช้รีจิสเตอร์ R0 - R7 ของรีจิสเตอร์ BANK ที่ถูกเลือก 3 บิทต่ำของ opcode คำสั่งจะบอกถึงรีจิสเตอร์ที่ถูกใช้งาน และ ACC, B,DPTR,CY ยังสามารถเป็นรีจิสเตอร์อ้างแอดเดรสได้

การอ้างแอดเดรสโดยตรง (Direct Addressing)

การอ้างแอดเดรสโดยตรงเป็นวิธีการอ้างถึง รีจิสเตอร์หน้าที่พิเศษ (Special function registers) 128 ไบท์ต่ำของ RAM ภายในจะถูกอ้างถึงโดยตรง

การอ้างแอดเดรสแบบ Register-Indirect Addressing

แบบนี้จะใช้ R0 หรือ R1 ใน BANK ที่ถูกเลือกเป็นตัวชี้ไปยังตำแหน่งใน 256 byte block ; 128 byte ต่ำของ RAM ภายใน ; 128 byte สูงของ RAM ภายใน ; หรือ 256 byte ต่ำของหน่วยความจำข้อมูลภายนอก (SFR ไม่สามารถเข้าถึงโดยวิธีนี้ได้) สามารถอ้างแอดเดรสของหน่วยความจำข้อมูลภายนอกได้เต็มที่ 64 Kbyte โดยใช้ 16 บิท Data Pointer

การอ้างแอดเดรสในทันทีทันใด (Immediate Addressing)

แบบนี้เป็นการอ้างแอดเดรสที่ยอมให้ค่าคงที่ (Constant) เป็นส่วนหนึ่งของ opcode คำสั่งในหน่วยความจำโปรแกรม

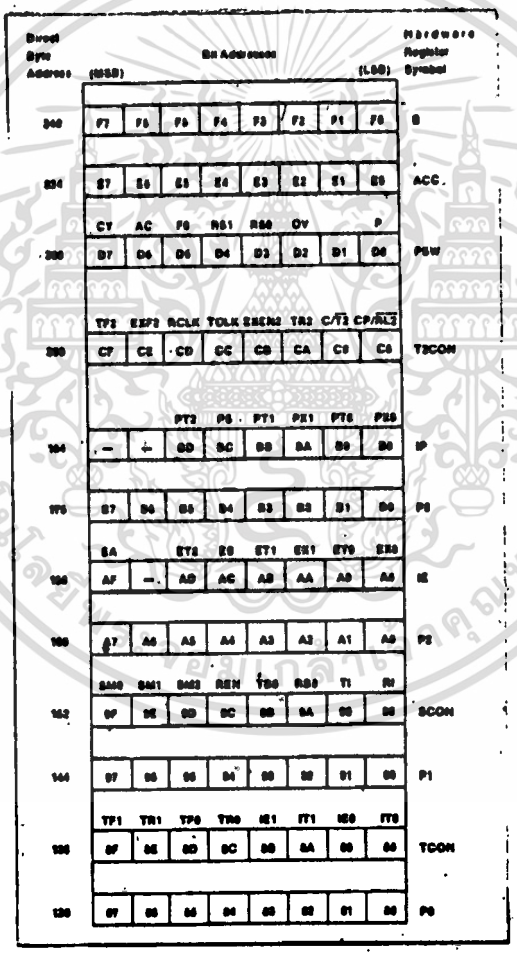
การอ้างแอดเดรสแบบ Base-Register plus Index Register-Indirect

แบบนี้จะยอมให้ 1 ไบท์ ถูกเข้าถึงจากหน่วยความจำโปรแกรม ผ่านการย้ายแบบ Indirect ซึ่งแอดเดรสคือ ผลรวมของ base register (DPTR หรือ PC) และ index register ,ACC โหมดนี้จะสะดวกในการเข้าถึงแบบเปิดตาราง (Look - up Table accesses)

บูลลีนโปรเซสเซอร์ (Boolean Processor)

เป็นการรวม bit processor ภายใน 8051 มันมีชุดคำสั่งเซ็ท ,แอคคิวมูเลเตอร์ (carry flag) และบิทแอดเดรส RAM และ I/O

คำสั่งที่จัดการข้อมูลเป็นบิทนั้นจะสามารถเซ็ทหรือเคลียร์ , เงื่อนไข, jump-if-set , jump-if-not-set , jump-if-set-then-cleared and move to/from the carry. และยังสามารถใช้บิทแอดเดรส ANDed หรือ ORed กับข้อมูลของ carry flag ผลลัพธ์จะถูกส่งกลับมายัง carry register



รูป แอดเดรสแต่ละบิทของรีจิสเตอร์หน้าที่พิเศษ (Special function register)

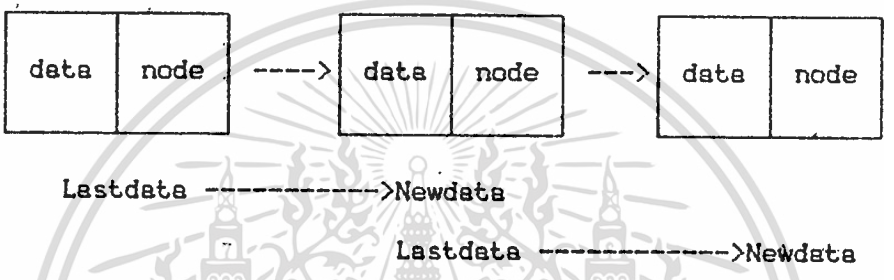
โครงสร้างและการจัดการข้อมูลต่างๆที่ใช้ในโครงงาน

ในโครงงานนี้ ส่วนของโปรแกรมที่เขียนขึ้นโครงสร้างข้อมูลสำคัญที่ใช้ คือ โครงสร้างข้อมูลแบบไดนามิคหรือพอยต์เตอร์ เพราะว่าสามารถเก็บข้อมูลได้มาก เกินกว่า 64 Kbyte สูงสุดเท่ากับหน่วยความจำของเครื่องที่ใช้

ต่อไปจะอธิบายถึงโครงสร้างข้อมูลที่ใช้

1). โครงสร้างข้อมูลแบบพอยต์เตอร์ทางเดียว(Single or One way Linked Lists)

โครงสร้างข้อมูลแบบนี้ จะมีตัวชี้ (Node) เพียงทางเดียวเพื่อชี้ข้อมูลตัวต่อไป



รูปแบบที่ใช้ในการเขียนด้วยภาษา PASCAL คือ

```
TYPE        p1    =  ^pnt1
            pnt1  =  RECORD
                     data : string[127];
                     node : p1;
                     END;
VAR         Newdata,Lastdata : p1;
```

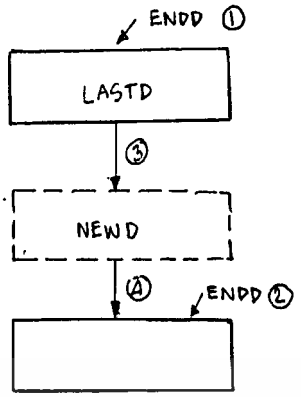
รูปแบบข้างบนเป็นการกำหนดพอยต์เตอร์ของ string ขนาด 127 ตัวอักษร โดยมี node เป็นตัวชี้ข้อมูลตัวถัดไป เมื่อเริ่มใช้งานจะต้องใช้คำสั่ง NEW() เพื่อจองเนื้อที่สำหรับตัวแปรเสียก่อน เช่น NEW(Lastdata); และในการเชื่อมโยงระหว่างพอยต์เตอร์จะใช้ node เป็นตัวเชื่อม สมมติให้พอยต์เตอร์ตัวแรกเป็น Lastdata ตัวที่ 2 เป็น Newdata ถ้าจะให้ node ของ Lastdata ชี้มาที่ Newdata ทำได้โดย

```
Lastdata^.node := Newdata;
```

จากรูปจะเห็นว่า node ตัวสุดท้ายจะชี้ตัวเองซึ่งใช้คำสั่ง

```
Newdata^.node := NIL;
```

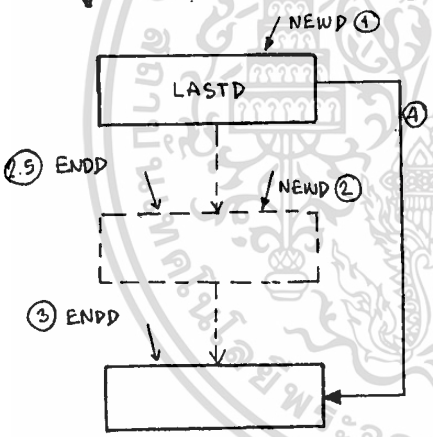
การแทรกข้อมูลของ Pointer ทางเดียว




```
VAR   Lastd,Newd,Endd : p1;

Endd := Lastd;          ...(1)
Endd := Endd^.node;     ...(2)
NEW(Newd);
Lastd^.node := Newd;    ...(3)
Newd^.node := Endd;     ...(4)
```

จากรูปเราต้องการแทรกข้อมูลระหว่างตัวที่ 2 กับ 3 ขั้นแรกให้ Lastd กับ Endd ชี้ที่ตัวที่ 2 ก่อนจากนั้น ให้ Endd ชี้ตัวที่ 3 แล้วเปิด NEW(Newd); เพื่อจองหน่วยความจำ จากนั้นก็ให้ node ของตัวที่ 2 ชี้มาที่ Newd และ ก็ให้ node ของ Newd ชี้มาที่ Endd ก็เสร็จการแทรกข้อมูล

การลบข้อมูลของ Pointer ทางเดียว




```
VAR   Lastd,Newd,Endd : p1;

Newd := Lastd;          ...(1)
Newd := Newd^.node;     ...(2)
Endd := Newd;
Endd := Endd^.node;     ...(3)
DISPOSE(Newd);
Lastd^.node := Endd;    ...(4)
```

จากรูป ให้ Newd และ Lastd ชี้ตัวที่ 2 จากนั้นให้ Newd ชี้ตัวที่ 3 และให้ Endd ชี้ที่เดียวกับ Newd จากนั้นให้ Endd ชี้ตัวที่ 4 แล้วก็ยกเลิกตัวแปรตัวที่ 3 จากนั้นโยงพอยต์เตอร์ ให้ตัวที่ 2 มาชี้ตัวที่ 4 แทน

2). โครงสร้างข้อมูลแบบพอยต์เตอร์ 2 ทาง(Doubly or Two way Linked List)

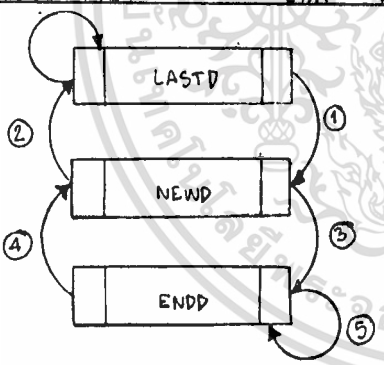
โครงสร้างข้อมูลแบบนี้จะมีตัวชี้(node) 2 ทาง เพื่อชี้ข้อมูลตัวที่ผ่านมาและตัวถัดไป ดังรูป

รูปแบบที่ใช้ในภาษา PASCAL คือ

```
TYPE      p2     =   ^pnt2;
          pnt2   =   RECORD
                     data : string[127];
                     nodeup : p2;
                     nodedn : p2;
                     END;
VAR       Lastd,Newd,Endd : p2;
```

จากรูปข้างบน เป็นการกำหนดข้อมูลพอยต์เตอร์ 2 ทาง รูปแบบข้อมูลคือ string[127] และมี nodeup เป็นตัวชี้ย้อนหลังไปหาตัวที่ผ่านมา และ nodedn เป็นตัวชี้ไปตัวถัดไป แบบนี้จะสะดวกกว่ามากเมื่อต้องการเลื่อนพอยต์เตอร์ไปยังข้อมูลย้อนหลังหรือไปข้างหน้าโดยไม่ต้องเริ่มตั้งแต่ตัวแรกเหมือน One-way linked list แต่แบบนี้มีข้อยุ่งยากในการต่อ Pointer ,ลบ ,แทรก พอสมควร โครงสร้างข้อมูลนี้ใช้ในโปรแกรม Editor ของ Cross Assembler 8051

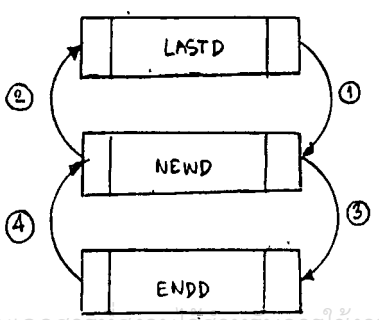
<u>การเชื่อมโยง Pointer ชนิด 2 ทาง</u>




```
Lastd^.nodeup := NIL;
Lastd^.nodedn := Newd;...(1)
Newd^.nodeup := Lastd;...(2)
Newd^.nodedn := Endd; ...(3)
Endd^.nodeup := Newd; ...(4)
Endd^.nodedn := NIL;  ...(5)
```

จากรูปเมื่อเราจองตัวแปร pointer ด้วยคำสั่ง NEW() แล้ว สมมติให้ Lastd ชี้ตัวแรก, Newd ชี้ตัวที่สอง , Endd ชี้ตัวที่สุดท้าย และโยง node ตามรูป

<u>การแทรกข้อมูล Pointer 2 ทาง</u>




```
NEW(Newd);
Lastd^.nodedn := Newd; ...(1)
Newd^.nodeup := Lastd; ...(2)
Newd^.nodedn := Endd;  ...(3)
Endd^.nodeup := Newd;  ...(4)
```

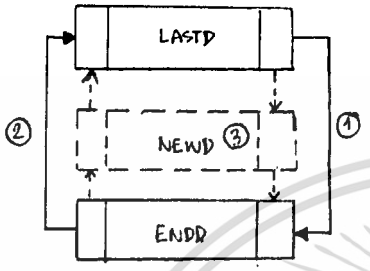
จากรูป (1) หมายถึง ให้ nodedn ของ Lastd ชี้ไปที่ Newd

(2) หมายถึง ให้ nodeup ของ Newd ชี้ไปที่ Lastd

(3) หมายถึง ให้ nodedn ของ Newd ชี้ไปที่ Endd

(4) หมายถึง ให้ nodeup ของ Endd ชี้ไปที่ Newd

การลบข้อมูลของ Pointer 2 ทาง




```
Lastd^.nodedn := Endd; ...(1)
Endd^.nodeup := Lastd; ...(2)
DISPOSE(Newd);         ...(3)
```

จากรูป (1) หมายถึง ให้ nodedn ของ Lastd ชี้ที่ Endd

(2) หมายถึง ให้ nodeup ของ Endd ชี้ที่ Lastd

(3) หมายถึง ให้ยกเลิกข้อมูลพอยต์เตอร์ที่ Newd ชี้อยู่

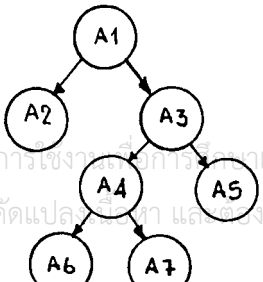
3). โครงสร้างข้อมูลแบบ Tree (Tree structure)

โครงสร้างข้อมูลแบบนี้มักใช้กับข้อมูลที่มีขนาดใหญ่ และต้องการค้นหาข้อมูลอย่างรวดเร็ว มีประสิทธิภาพสูง Tree เป็นเซ็ทของ elemant ของข้อมูลที่นับได้ตั้งแต่ 1 ชุดขึ้นไปโดยมีคุณสมบัติพื้นฐานดังนี้

- มีอีลาเมนท์หนึ่งตัวซึ่งมีลักษณะพิเศษกว่าอีลาเมนท์ตัวอื่นที่เรียกว่า root
- มีอีลาเมนท์อื่นๆ (ไม่นับ root) ถูกจัดแบ่งให้เป็นเซทเล็กๆที่เป็นอิสระต่อกัน ซึ่งเรียกเซ็ทเล็กๆเหล่านี้ว่า sub-trees ของ root

จากข้อกำหนดนี้จะเห็นว่า อีลาเมนท์ทุกตัว จะมีคุณสมบัติเป็น root ของ sub-trees ในทรีทั้งหมด และจำนวนของ sub-trees ของอีลาเมนท์หนึ่งๆจะเรียกว่า ดีกรี (degree) ของอีลาเมนท์ตัวนั้น ซึ่งถ้าอีลาเมนท์ตัวใดมีดีกรีเป็น 0 (ไม่มี sub-tree เป็นของตัวเอง) อีลาเมนท์นั้นจะเรียกว่าเป็น terminal element หรือ ลีฟ (leaf) และตัวที่ไม่เป็นลีฟหรือ non terminal element จะเรียกว่า branch element

การเขียนโครงสร้างลักษณะของทรีนั้นจะเขียนสัญญลักษณ์ได้ดังรูปข้างล่าง



และเมื่อเขียนอธิบายจะเขียนด้วยเครื่องหมายวงเล็บ (parenthetical representation) ซึ่งจากรูปข้อมูลทรีข้างบนจะเขียนแทนได้ด้วย

(A1 (A2 , A3 (A4 (A6 , A7) , A5)))

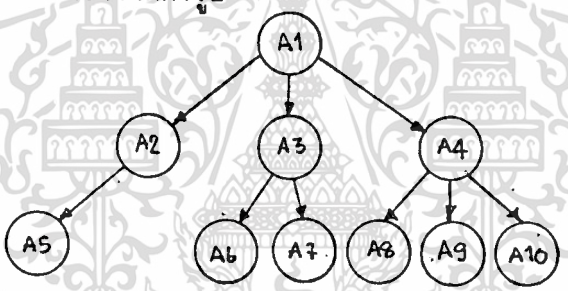
การเขียนอธิบายนี้จะอาศัยกฎเกณฑ์ 2 ข้อคือ

1). ถ้าในทรีมีอีลาเมนท์เพียงตัวเดียว อีลาเมนท์จะเขียนแทนที่ข้อมูลด้วยวงเล็บปิดกาปกติ

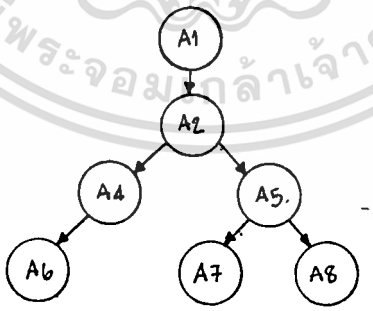
2). ถ้าในที่ตัวนั้นมี root อยู่ด้วย (ตัวเองเป็น non terminal element) ก็จะเขียนด้วยวงเล็บเปิดต่อ และเขียนตัว sub-trees และถ้ามี sub-trees มากกว่าจะแยกด้วยเครื่องหมายคอมม่า (,) แล้วจึงเขียนตามด้วยปิดกาปิด

ลักษณะของทรีนั้นเมื่อจัดแบ่งตามโครงสร้างนั้นจะแบ่งออกเป็น 2 แบบคือ

1). Ordinary tree เป็นทรีที่มีการจัดที่สามารถแบ่งหรือมี sub-trees ได้โดยไม่จำกัด ซึ่งเขียนโครงสร้างได้ดังรูป

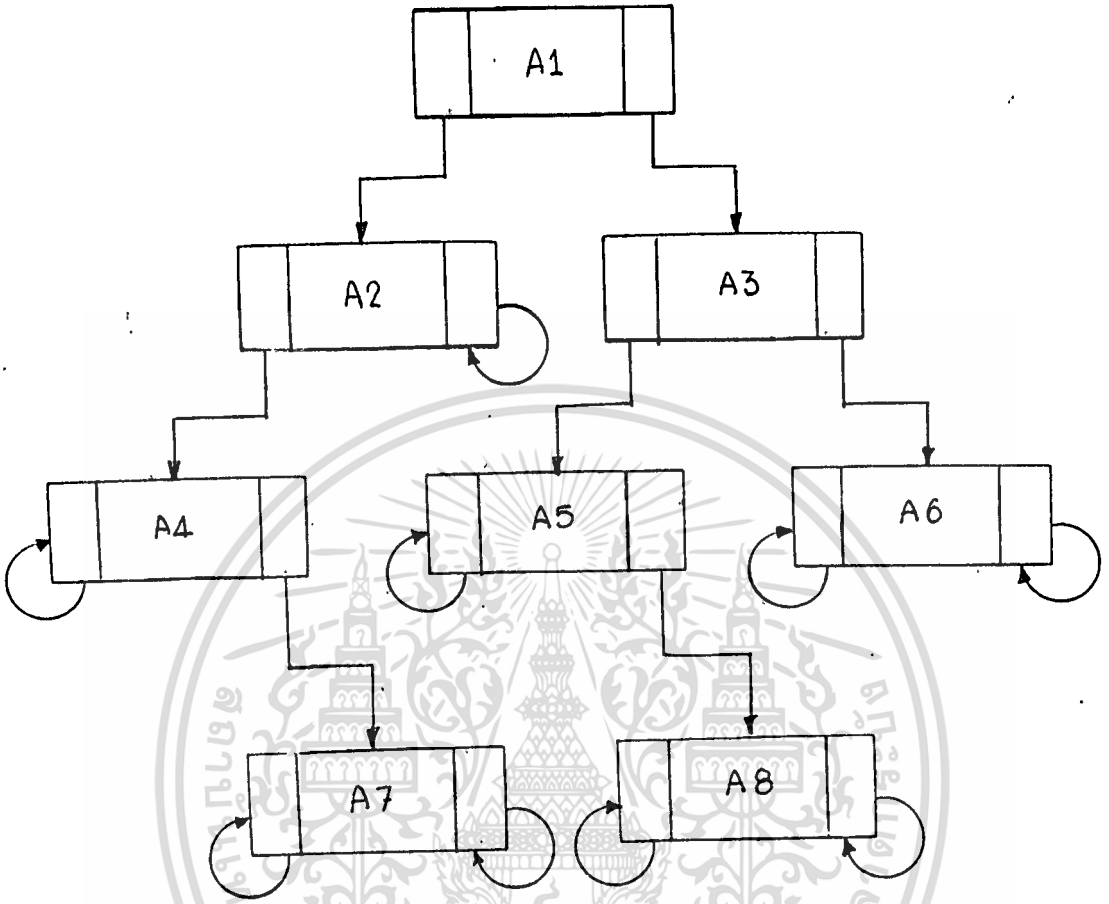


2). Binary tree เป็นทรีที่มีการจัดแบ่งหรือมี sub-trees ได้ไม่เกิน 2 ทางซึ่งเขียนได้ดังรูปข้างล่าง



จะเห็นว่า Binary tree นั้น จะจัดว่าเป็นส่วนหนึ่งของ Ordinary tree ซึ่งการจัด Binary tree นี้ทำให้เพื่อนำไปใช้พิจารณาการประมวลผลนั้น ทำได้ง่าย

การจัดโครงสร้างแบบทรีนี้ สามารถนำมาประยุกต์ใช้งานได้มากในระบบข้อมูลของคอมพิวเตอร์ โดยอาจจะนำโครงสร้างข้อมูลอันอื่นมาจัดใหม่ให้เป็นโครงสร้างแบบทรีอีกชั้นหนึ่งได้ เช่น จัดให้ข้อมูลแบบ link list เป็นโครงสร้างข้อมูลแบบทรีดังรูปข้างล่าง



รูป แสดงโครงสร้างข้อมูล ลิงค์ลิสต์ ที่เป็นแบบทรี

ทฤษฎีของ CROSS ASSEMBLER

ในการเขียนโปรแกรมของคอมพิวเตอร์ด้วยภาษาเครื่องนั้น เป็นการเขียนที่กระทำโดยตรงซึ่งสามารถทำให้คอมพิวเตอร์ สามารถเข้าใจ และปฏิบัติตาม วัตถุประสงค์ของผู้เขียนโปรแกรมได้ แต่การเขียนโปรแกรมคอมพิวเตอร์ด้วยภาษาเครื่องนั้นเป็นวิธีการที่น่าเบื่อหน่าย, ยุ่งยาก, เสียเวลา และสุดท้ายก็จะเกิดมีข้อผิดพลาดเกิดขึ้นได้ง่าย ซึ่งความยุ่งยากต่าง ๆ เหล่านี้ เกิดจากเหตุผลต่าง ๆ ดังนี้

1) คำสั่งภาษาเครื่องนั้น จะต้องเขียนในรูปของรหัสตัวเลข (NUMERICAL CODE) ซึ่งเป็นรูปเลขฐานสอง

2) ตำแหน่งหรือแอดเดรสทุก ๆ แอดเดรสที่จะอ้างอิงถึงของเครื่องนั้น จะต้องทราบและถูกกำหนดตำแหน่งให้สมบูรณ์ ดังนั้น ผู้เขียนโปรแกรมจึงต้องทราบตำแหน่งของข้อมูล ตลอดจนคำสั่งทุก ๆ คำสั่งที่ใช้ในโปรแกรม ซึ่งจะทำไปใช้ในการหาตำแหน่งของโปรแกรมอื่น หรือข้อมูลต่อไป

3) เมื่อมีการเปลี่ยนโปรแกรมหรือข้อมูลที่ใช้ หรือการจะเพิ่มหรือลดคำสั่งบางคำสั่งในโปรแกรมนั้น จะต้องมีการเปลี่ยนแปลงตำแหน่งของคำสั่งและข้อมูล ซึ่งมีผลไปถึงคำสั่งอื่น ๆ ที่เกี่ยวข้องกับข้อมูลหรือการเรียกที่อ้างอิงถึงตำแหน่งนั้น จะต้องเปลี่ยนแปลงตำแหน่งใหม่ทั้งหมด

4) ผู้เขียนโปรมแกรมภาษาเครื่องนั้น เพื่อนำโปรแกรมนั้นมาตรวจสอบ หรือตรวจสอบโดยบุคคลอื่นนั้น ทำได้ค่อนข้างยาก, ช้า และน่าเบื่อหน่าย

5) เมื่อมีโปรแกรมเก่าอยู่ และจะนำไปพัฒนาต่อไปนั้น กระทำได้ยาก

จากที่กล่าวมาข้างบนหากเราสามารถเขียนโปรแกรมด้วยสัญญาลักษณ์อื่น ๆ ที่ทำให้เราหรือบุคคลอื่น ๆ เข้าใจได้ง่าย ๆ แล้ว และยังทำให้เครื่องคอมพิวเตอร์สามารถเข้าใจและทำงานตามวัตถุประสงค์ของเราได้ ก็จะทำให้สะดวกขึ้น ทั้งในด้านการเขียนโปรแกรมและการพัฒนา แก้ไข ต่อผู้เขียน ซึ่งขบวนการ ขั้นตอนต่าง ๆ นี้จะเริ่มด้วย เมื่อเขียนโปรแกรมเป็นภาษาสัญญลักษณ์แล้ว จากนั้นผู้เขียนก็แปรเป็นภาษาเครื่องเพื่อให้เครื่องสามารถเข้าใจและปฏิบัติตามวัตถุประสงค์ของผู้เขียน แล้วป้อนภาษาเครื่อง(รูปเลขฐาน2) นี้ให้เครื่องคอมพิวเตอร์ ทำงานต่อไป

จากขบวนการพัฒนา จากภาษาสัญญลักษณ์ ให้แปลเป็นภาษาเครื่องซึ่งจะทำให้เครื่องคอมพิวเตอร์เข้าใจ เพื่อเราจัดการให้เครื่องคอมพิวเตอร์ ทำการแปลภาษาเครื่องนี้ให้เรา ก็จะทำให้เราเขียนโปรแกรมหรือพัฒนาโปรแกรมต่าง ๆ ได้สะดวกรวดเร็ว และให้ความถูกต้องมากยิ่งขึ้น ซึ่งตัวที่เราพัฒนาให้แปลภาษาสัญญลักษณ์ เป็นภาษาเครื่องนี้เป็นโปร

แกรมอันหนึ่ง ซึ่งเราเรียกว่า โปรแกรมแอซเซมเบลอร์ (ASSEMBLER PROGRAM) และเซ็ทของตัวสัญญลักษณ์ หรือภาษาสัญญลักษณ์ที่ใช้เป็นเครื่อง นั้น ซึ่งจะมีกฎเกณฑ์ต่าง ๆ ตามที่ผลเขียนโปรแกรมกำหนดใช้นั้น เรียกว่าภาษาแอซเซมเบลอร์ (ASSEMBLER LANGUAGE)

สำหรับโปรแกรมที่เขียนขึ้นโดยใช้เครื่องคอมพิวเตอร์เครื่องหนึ่ง และนำผลของภาษาเครื่องที่ได้ให้กับคอมพิวเตอร์ที่เป็นภาษาเครื่องที่ถูกต้องของตัวเอง สามารถทำงานได้ โดยตัวเครื่องคอมพิวเตอร์ที่เป็นตัวแปลภาษาเองนั้น ไม่สามารถที่จะนำภาษาเครื่องที่แปลออกมานั้นไปใช้ทำงานได้ ตามวัตถุประสงค์ของผู้เขียนโปรแกรม เราเรียกโปรแกรมแอซเซมเบลอร์แบบนี้ว่า โปรแกรมคลอสแอซเซมเบลอร์ (CROSS ASSEMBLER PROGRAM) โปรแกรมชนิดนี้จะเป็นเครื่องมืออย่างดีที่เราใช้พัฒนาภาษาแอซเซมเบลอร์ของเครื่องคอมพิวเตอร์อื่น ๆ ที่ไม่สามารถทำบนเครื่องนั้น ๆ ได้โดยตรง ทั้งนี้อาจเกิดจากเหตุผลต่าง ๆ เช่น ยังไม่มีเครื่องคอมพิวเตอร์นั้นอยู่ แต่ต้องการเขียนโปรแกรมเพื่อให้คอมพิวเตอร์นั้นสามารถทำงานได้, หรือเครื่องนั้นมีขนาดเล็ก หรือไม่มีเครื่องมือหรืออุปกรณ์ช่วยพัฒนาโปรแกรมแอซเซมเบลอร์ของตัวเองได้เป็นต้น

1. ภาษาแอซเซมเบลอร์

ก่อนอื่น เมื่อจะกล่าวถึงภาษาแอซเซมเบลอร์ที่จะใช้กับคอมพิวเตอร์นั้น ขอให้พิจารณาถึงรูปแบบภาษานี้ว่าจะเริ่มเป็นอย่างไรเพื่อจะช่วยให้เข้าใจในการเขียนโปรแกรมให้ง่ายยิ่งขึ้น ดังเช่น ตัวอย่างโปรแกรมภาษาแอซเซมบลี้ ที่บวกเลขสองจำนวนเข้าด้วยกันข้างล่างนี้ ดังโปรแกรมในรูปที่ (1) ซึ่งเมื่อเราเขียนโปรแกรมเป็นภาษาแล้ว ก็จำเป็นต้องทราบตำแหน่งที่แน่นอน ของคำสั่งและข้อมูลทุกตัว และเมื่อมีการเปลี่ยนแปลงตำแหน่งใด ๆ แล้ว ก็จะต้องมีการจัดตำแหน่งใหม่ทุกครั้ง เช่น เมื่อเราเปลี่ยนตำแหน่งของโปรแกรมที่เขียนไปเริ่มต้นที่ตำแหน่ง Ø1ØØØH นั้น ตำแหน่งคำสั่ง, ข้อมูล และภาษาที่แปลออกมานั้น จะต้องเปลี่ยนแปลงไปจากเดิม ซึ่งเป็นการลำบากมาก เมื่อเราต้องแปลภาษาเครื่องเอง ซึ่งถ้าหากเราให้โปร แกรมแปลและมีการกำหนดสัญญลักษณ์แทนตำแหน่งข้อมูลก็จะทำให้เราสามารถเปลี่ยนแปลงตำแหน่งและโปรมแกรมได้โดยสะดวกและง่ายขึ้น สัญญ–ลักษณ์ที่ใช้แทนตำแหน่งข้อมูลนี้ เราเรียกว่า ลาเบล (LABEL) และสำหรับภาษาเครื่องนี้เราจะใช้ตัวอักษรย่อเขียนแทนคำสั่งแต่ละคำสั่ง ซึ่งเมื่อโปรแกรมในตารางที่ มาเขียนใหม่จะได้ดังตารางที่ ข้างล่างนี้

แอดเดรส	ภาษาเครื่อง	ภาษาแอซเซมบลี้
0000	E5 07	MOV A,07H
0002	25 08	ADD A,08H
0004	F5 09	MOV 09H,A
0006	00	NOP
0007	10	10H
0008	20	20H
0009	30	-

รูปที่ 1

```
                MOV   A,DAT1
                ADD   A,DAT2
                MOV   DAT3,A
                NOP
DAT1 :          10H
DAT2 :          20H
DAT3 :          -
```

รูปที่ 2

จากโปรแกรมข้างบน จะเห็นว่าตำแหน่งของหน่วยความจำที่ใช้เก็บข้อมูลจะกำหนดเป็นลาเบลมี 3 ตำแหน่ง ซึ่งเมื่อมีการเรียกใช้อ้างอิงตำแหน่งข้อมูลในโปรแกรมคำสั่งก็จะอ้างด้วยชื่อลาเบลของตำแหน่งนั้น แทน ซึ่งจะเห็นว่าเป็นการและสะดวก ชื่อตำแหน่งลาเบลในโปรแกรมข้างบนนี้มี DAT1, DAT2 ใช้เป็นที่เก็บข้อมูลที่จะนำมาบวกกัน และ DAT3 ใช้เป็นที่เก็บผลลัพธ์ของข้อมูล จะสังเกตเห็นว่าชื่อลาเบลนั้นจะตามด้วยสัญญลักษณ์พิเศษเฉพาะตัวหนึ่ง (ในที่นี้ใช้โคลอน :) เพื่อใช้แยกแสดงว่าเป็นลาเบลไม่ใช่คำสั่งในโปรแกรม

1.1 <u>กระทงความภาษา</u> (LANGUATE STATEMENTS)

ในภาษาแอซเซมเบลอร์หรือภาษาอื่น ๆ ที่ใช้สำหรับเขียนโปรแกรมใด ๆ จะ

มีหน่วยพื้นฐานเริ่มต้นคือ หนึ่งบรรทัดของรหัสสัญญลักษณ์ที่เรียกว่ากระทงความ (STATEMENT) ซึ่งในภาษาใด ๆ ก็ตาม ก็จะมีการกำหนดชุดของกฎเกณฑ์ซึ่งจะแสดงให้ทราบว่า จะนำสัญญลักษณ์ต่าง ๆ มาประกอบกันเป็นกระทงความของภาษาได้อย่างไร

สำหรับในภาษาแอมเซมเบลอร์นั้น มีกฎเกณฑ์ต่าง ๆ ที่เข้าใจได้ง่ายซึ่งแบ่งลักษณะของกระทงความออกเป็น 3 ชนิดคือ

1) กระทงความคอมมานด์เครื่อง (MACHINE COMMAND STATEMENTS)

2) กระทงความคอมมานด์แอซเซมเบลอร์ (ASSEMBLER COMMAND STATEMENTS) ที่เรียกว่า PSEUDO – OPERATION

3) กระทงความข้อมูล (DATA STATEMENTS)

กระทงความคอมมานด์เครื่อง เป็นกระทงความที่แอซเซมเบลอร์จะแปลเป็นรหัสภาษาเครื่องที่สอดคล้องกันโดยตรง เช่น

```
MOV     A,DAT1
```

เมื่อใช้แอซเซมเบลอร์แปลเป็นภาษาเครื่อง จะได้

```
E5  07
```

โดยเราสมมติให้ลาเบลของ DAT1 นั้น หมายถึงหน่วยความจำตำแหน่ง 07H

กระทงความคอมมานด์แอซเซมเบลอร์ หรือ PSEUDO – OPERATION เป็นกระทงความที่แอซเซมเบลอร์ไม่ต้องออกมาเป็นภาษาเครื่อง แต่ใช้สำหรับเป็นการสร้างข่าวสารให้โปรแกรมแอซเซมเบลอร์ที่เกี่ยวข้องกับการทำงานในบางส่วนของขบวนการแปล เช่น กระทงความ END. ซึ่งต้องเป็นกระทงความสุดท้ายของโปรแกรมของแอซเซมเบลอร์ที่จะเขียน ซึ่งเมื่อแอซเซมเบลอร์แปลกระทงความนี้ออกมา ก็หมายความว่าโปรแกรมภาษาแอซเซมเบลอร์นั้น มาถึงจุดจบของโปรแกรมแล้ว และเมื่อมีโปรแกรมในบรรทัดต่อมาก็จะไม่มีการแปลอีกต่อไป

กระทงความข้อมูล เป็นกระทงความที่ใช้สำหรับกำหนดค่าข้อมูลเข้าไปยังหน่วยความจำ ณ ที่ตำแหน่ง ที่กำหนดไว้ เช่น

```
DAT1 :    08AH
```

เป็นการกำหนดค่าในหน่วยความจำตำแหน่ง ชื่อ DAT1 ด้วยค่าที่เลขฐานสอง ที่เทียบเท่ากับค่าเลขฐานสิบหก

ในการประมวลผลของภาษาแอซเซมเบลอร์นั้น จะกระทงครั้งละ 1 กระทงความ ดังนั้น เซ็ทของกฎเกณฑ์ ที่กล่าวถึง กระทงความต่าง ๆ จะต้องให้แนวทางสำหรับแอซเซมเบลอร์ ในการทำข่าวสารจากกระทงความนั้น ฉะนั้นก่อนที่จะกล่าวถึงภาษาแอซเซมเบลอร์

ก็จะอธิบายถึง เครื่องหมาย, กฎเกณฑ์ ที่เกี่ยวกับโครงสร้างของกระทงความก่อน ดังนี้

1.2 ชนิดของกระทงความ

กระทงความแต่ละกระทงความในภาษาแอซเซมเบลอร์นั้น เป็นชุดหรือแถวที่มีลำดับของตัวอักษรที่ถูกเลือกมาจากชุดของตัวอักษร ซึ่งจะแยกกระทงความด้วยชุดของตัวอักษรพิเศษ ซึ่งข่าวสารที่บรรจุในกระทงความนั้น จะจัดเป็นกลุ่มตัวอักษรซึ่งเรียกว่าเทอม (TERM) เช่น กระทงความ

MOV A,DAT1

ประกอบด้วยชุดตัวอักษร 12 ตัว โดยนับช่องว่างด้วย (SPACE) เมื่อแยกเป็นเทอมจะมี 3 เทอม คือ เทอม MOV, A และ DAT1 ซึ่งแต่ละเทอมนั้นก็ยังประกอบด้วยตัวอักษรหลายตัว เราจึงต้องหากฎเกณฑ์ในการเรียงตัวอักษรในเทอมอีกว่าจัดอย่างไร จากกระทงความข้างบนอักษรพิเศษที่ใช้ในการแยกกระทงความออกเป็นเทอมก็คือช่องว่างและเครื่องหมายคอมม่า (,)

เทอมที่ใช้เขียนแสดงแอดเดรส โอเปอร์เรชั่นโค้ด หรือสำหรับการทำงานอื่น ๆ เราเรียกสัญญลักษณ์ (SYMBOL) ซึ่งลำดับใด ๆ จะประกอบด้วยตัวอักษร (A,B,...Z) หรือตัวเลขจำนวนยาวขนาดหนึ่งตามที่ผู้เขียนโปรแกรมแอซเซมเบลอร์ กำหนดโดยตัวแรกจะต้องเป็นตัวอักษร และหลังตัวสุดท้ายจะต้องเป็นช่องว่างหรือเครื่องหมายพิเศษต่าง ๆ เช่น ; , : , + , - , = เป็นต้น เช่น "1543 " "25+ " "ØØ36 "

นอกจากที่กล่าวถึงสัญญลักษณ์และตัวเลขแล้วนั้น ยังต้องมีอักษรพิเศษต่าง ๆ แสดงถึงการทำงานที่แอซเซมเบลอร์กระทำยังสัญญลักษณ์ และตัวเลขที่มีในกระทงความ ตัวอักษรพิเศษต่าง ๆ เหล่านี้ แสดงได้ดังตัวอย่างตารางที่ ข้างล่างนี้

ตัวอักษร	ชื่อ	การใช้
	SPACE	ใช้แสดงการแบ่งขอบเขตของเทอม
	CARRIATE RETURN	ใช้แสดงการจบกระทงความ
,	COMMA	ใช้แสดงการแบ่งแยกเทอมในภาษาแอซเซมบลี้
;	SEMICOLON	ใช้แสดงว่าเทอมที่มาก่อนหน้านี้เป็นเทอมสัญญลักษณ์ของการแทนแอดเดรส
+	PLUS	ใช้แสดงการบวก
-	MINUS	ใช้แสดงการลบ
$	DONLA SIGN	ใช้แสดงการอ้างตำแหน่งแอดเดรสในขณะนั้นโดยตรง

ตารางที่ 3 แสดงตัวอย่างอักษรพิเศษ

จากที่เราได้กล่าวจากชนิดของกระทงความที่สามารถใช้ได้แล้ว ต่อไปเมื่อนำเอาเทอมต่าง ๆ นั้น มาประกอบกันเป็นกระทงความที่สมบูรณ์นั้น กระทงความ 1 บรรทัดจะประกอบด้วย

<ชื่อของเทอม> <BLANK> <COMMAND> <BLANK> <COMMENT>

ซึ่งเทอมทั้งหมด บางเทอมอาจจะไม่จำเป็นต้องมีในบรรทัดของโปรแกรมก็ได้

1.2.1 กระทงความคอมมานด์ (COMMMAND STATEMENTS)

กระทงความคอมมานด์ เป็นกระทงความที่แอซเซมเบลอร์จะต้องแปลให้เป็นภาษาเครื่อง 1 คำสั่งของแต่ละบรรทัด โครงสร้างของกระทงความนี้ขึ้นกับชนิดของการทำงานที่เราใช้แทนของภาษาแอซเซมบลี้ของเครื่องคอมพิวเตอร์หรือซีพียูแต่ละตัว สำหรับโครงสร้างทั่ว ๆ ไป กระทงความคอมมานด์มีรูปแบบดังนี้

<ลาเบล> :<เว้นว่าง><คอมมานด์><เว้นว่าง><โอเปอร์แรนด์><เว้นว่าง> ;<คอมเมนด์>

<LABEL> :<BLANK> <COMMAND><BLANK> <OPERAND> <BLANK> ;<COMMENT>

สำหรับการเขียนโปรแกรมโดยทั่ว ๆ ไป ไม่จำเป็นต้องมีการเขียนทุกส่วนให้ครบในแต่ละบรรทัด เทอมที่ใช้ในกระทงความคอมมานด์ แต่ละเทอมจะอธิบายรายละเอียดดังต่อไปนี้

เทอม <BLANK> แสดงถึงช่องว่างตามลำดับจำนวนหนึ่งที่แยกเทอมออกจากกัน

เทอม <COMMAND> แสดงถึงคำสั่งหนึ่งคำสั่งของภาษาแอซเซมบลี้ของซีพียูแต่ละเบอร์โดยประกอบขึ้นด้วยชุดตัวอักษรที่มีความหมายเฉพาะของแต่ละคำสั่งภาษาเครื่อง

เทอม <OPERAND> แสดงถึงส่วนขยายของคำสั่งที่มีต่อไปอีก ให้มีการทำงานเฉพาะเจาะจงลงไปอีกของคำสั่งซึ่งต้องตามหลังเทอมคอมมานด์ ในคำสั่งของคอมมานด์ที่ต้องการโอเปอร์แรนด์ตามมา เทอมโอเปอร์แรนด์นี้ยังแบ่งย่อยได้เป็นสองเทอม ซึ่งจะมีใช้ในบางคำสั่งของเทอมคอมมานด์ ซึ่งเทอมทั้งสองนี้จะแยกกันด้วยเครื่องหมายคอมม่า (,) เขียนแสดงได้ดังนี้

<โอเปอร์แรนด์ 1>,<โอเปอร์แรนด์ 2>

<OPERAND 1 >,<OPERAND 2 >

เทอม <LABEL>: (ตามด้วยโคลอน) ใช้แสดงเมื่อเรากำหนดลาเบลให้กับกระทงความใด ๆ และต้องการอ้างตำแหน่งถึง ในการใช้งาน จากกระทงความคอมมานด์อื่น ๆ ซึ่งเทอมของลาเบลนี้ ตามด้วยอักษรพิเศษคือ โคลอน ทั้งนี้เพื่อให้โปรแกรมแอซเซมเบอร์รู้ว่า เป็นเทอมลาเบล ที่จะใช้อ้างเป็นตำแหน่งของหน่วยความจำ ที่จะอ้างจากกระทงความคอมมานด์อื่น ๆ ซึ่งจะเห็นว่า เราใช้ชื่อลาเบลเหมือนกันในโปรแกรมที่เขียนขึ้นในโปรแกรมหนึ่ง ๆ ไม่ได้

เทอม ; <COMMENT> (นำหน้าด้วยเครื่องหมายเซมิโคลอน) ใช้เป็นเทอมที่ผู้เขียนโปรแกรมแอซเซมบลี้ต้องการใช้อธิบายการทำงาน หรือให้ความหมายอื่น ๆ เพื่อง่ายต่อการเข้าใจโปรแกรมเอง แต่เทอมนี้จะไม่มีความหมายต่อการแปลภาษาเครื่องของโปรแกรมแอซเซมเบลอร์

จากเทอมต่าง ๆ ที่กล่าวมาแล้ว ในกระทงความคอมมานด์ สิ่งที่จำเป็นที่ต้องมีในหนึ่งบรรทัดก็คือ เทอมคอมมานด์และเทอมช่องว่าง ส่วนเทอมอื่น ๆ นั้น จะขึ้นอยู่กับเทอมของคอมมานด์ (เทอมโอเปอร์แรนด์) และความต้องการของผู้เขียนโปรแกรมแอซเซมเบลอร์

1.2.2 กระทงความข้อมูล (DATA STATEMENTS)

กระทงความข้อมูลมีความใกล้ชิดกับกระทงความคอมมานด์ ใช้สำหรับในการกำหนดค่าตัวเลข ซึ่งเป็นข้อมูลที่จะกำหนดใส่เข้าไปในหน่วยความจำ ซึ่งรูปแบบกระทงความข้อมูลก็จะเหมือน ๆ กับ กระทงความคำสั่ง แต่คำสั่งที่ให้ใช้ได้เป็นการกำหนดขึ้นตามรูปแบบของผู้เขียนโปรแกรมแอซเซมเบลอร์ เช่น อาจกำหนดให้แทนใช้ใช้คำสั่ง "DEFB" สำหรับในการกำหนดค่าข้อมูลในการเขียนโปรแกรมแอซเซมบลี้ของซีพียู ก็ได้เป็นต้น รูป

แบบโดยทั่วไปของกระทงความข้อมูล แสดงได้ดังนี้

<ลาเบล> : <เว้นว่าง> <คำสั่งเทียม> <เว้นว่าง> <ข้อมูล> ; <คอมเมนต์>

ซึ่งเทอมต่าง ๆ ของกระทงความมีความหมายดังนี้

เทอม <ลาเบล> และ <คอมเมนต์> จะมีความหมายการใช้เหมือนกับในกระทงความคอมมานด์

เทอม <คำสั่งเทียม> เป็นเทอมที่ประกอบด้วยชุดตัวอักษร ที่รวมเป็นคำสั่งเฉพาะที่จะอนุญาตให้ใช้ได้ จากผู้เขียนโปรแกรมแอซเซมบลี้ ซึ่งผู้ใช้งานจะต้องปฏิบัติตามรูปแบบที่กำหนดขึ้น จึงเห็นได้ว่าเทอมคำสั่งเทียมของกระทงความนี้ จะไม่มีในภาษาแอซเซมบลี้ของซีพียูของแต่ละตัว

เทอม <ข้อมูล> เป็นเทอมที่ต้องการนำค่านี้ ไปเก็บไว้ในหน่วยความจำ ซึ่งรูปแบบของข้อมูลมีรูปแบบต่าง ๆ ซึ่งก็ขึ้นอยู่กับผู้เขียนโปรแกรมแอซเซมเบลอร์ ว่าจะอนุญาตให้ใช้ข้อมูลรูปแบบใด เช่นอาจให้ใช้ในรูปแบบเลขฐานสอง, เลขฐานสิบ หรืออาจให้ใช้รูปเลขลบหรือตัวอักษรเพิ่มเติมเข้ามา สำหรับในกรณีนี้เช่น ถ้ามีเครื่องหมายเลขลบอยู่นำหน้าเทอมข้อมูลแอซเซมเบลอร์จะเปลี่ยนเทอมข้อมูลเป็นค่าคอมพีเมนต์ของสอง ของเลขฐานสอง ที่เป็นขนาดของข้อมูลนั้น ในกรณีของข้อมูลเป็นตัวอักษร จะอยู่ในเครื่องหมายคำพูดหรือฟันหนู (",') ซึ่งข้อมูลนั้น จะให้ค่าเป็นรหัส ASCII

1.2.3 กระทงความแอซเซมเบลอร์ (ASSEMBLER COMMAND STATEMENT)

กระทงความคอมมานด์แอซเซมเบลอร์หรือ PSEUDO COMMAND เป็นกระทงความคำสั่งที่โปรแกรมแอซเซมเบลอร์อนุญาตให้ใช้ในการเขียนโปรแกรมเอง ไม่ใช่กระทงความคอมมานด์ของภาษาแอซเซมบลี้ที่ต้องแปลเป็นภาษาเครื่อง แต่จะเป็นกระทงความที่ต้องแปลเพื่อกำหนดค่าต่าง ๆ เพื่อปฏิบัติตามความหมายของคำสั่งแอซเซมเบลอร์ กระทงความแอซเซมเบลอร์ที่ใช้เช่น คอมมานด์เริ่มต้น (BEGIN) การแปลจุดเริ่มต้นของตำแหน่งหน่วยความจำที่จะใช้อ้างอิงในหน่วยความจำของโปรแกรมที่เขียนขึ้น คอมมานด์จบโปรแกรม (END) เป็นกระทงความที่จะแจ้งให้แอซเซมเบลอร์หยุดการแปลโปรแกรม ซึ่งผู้เขียนโปรแกรมภาษาแอซเซมบลี้ของซีพียูจะต้องใช้คำสั่งนี้ในตอนท้ายของโปรแก์รม

จากที่ผ่านมาจะเห็นว่าโปรแกรมแอซเซมบลี้ที่เขียนขึ้นของแต่ละเบอร์นั้น จะเขียนอยู่ในรูปของสัญญลักษณ์ของอักษรที่มีรูปแบบตามที่กำหนดจากแอซเซมเบลอร์โดยข้อมูลโปรแกรมนั้น จะเก็บไว้ในรูปของข้อมูลดิบ เรียกว่า ซอร์สโปรแกรม (SOURCE PROGRAM) เมื่อโปรแกรมแอซเซมเบลอร์แปลออกมาเป็นภาษาเครื่องแล้ว จะเรียกข้อมูลที่แปลออกมาได้นี้ว่า ออปเจคท์ โปรแกรม (OBJECT PROGRAM) เป็นขบวนการที่โปรแกรมแอซเซมเบลอร์

แปลซอร์สโปรแกรม ให้เป็นออปเจคท์โปรแกรม มีงานที่โปรแกรมต้องจัดทำดังต่อไปนี้

1) การประมวลผลกระทงความในภาษาแอซเซมเบลอร์นั้น แอซเซมเบลอร์นั้นจะต้องจำส่วนต่าง ๆ ของกระทงความได้ โดยแรกสุดเมื่อมีการกำหนดตำแหน่งเริ่มต้น ของตำแหน่งหน่วยความจำที่จะใช้เริ่มต้นแปลกระทงความอื่น ๆ ต่อไป แอซเซมเบลอร์ก็ต้องจำตำแหน่งหน่วยความจำและตำแหน่งการเปลี่ยนแปลงได้ ทั้งนี้เพื่อใช้ในการเก็บตำแหน่งของออปเจคท์โค้ด ที่แปลออกมาได้ ให้ตรงกับตำแหน่งหน่วยความจำ และต่อมาเมื่อมีการกำหนดลาเบล แอซเซมเบลอร์ก็ต้องจำชื่อของลาเบล และตำแหน่งของหน่วยความจำขณะนั้นให้ได้

2) ในการเปลี่ยนแปลงตำแหน่ง หน่วยความจำต้องเปลี่ยนแปลงให้สอดคล้องกับออปเจคท์โค้ดที่แอซเซมเบลอร์แปลออกมา เพื่อให้สามารถเก็บออปเจคท์โค้ดได้อย่างต่อเนื่องและถูกต้อง

3) การอ้างอิงตำแหน่งที่เหมาะสม ให้กับกระทงความคอมมานด์ เมื่อมีการอ้างอิงถึง และเมื่อมีการอ้างถึงลาเบลที่เป็นตำแหน่งหน่วยความจำที่แอซเซมเบลอร์นั้นยังแปลไม่ถึง จึงเป็นปัญหาที่จะต้องเก็บกระทงความนี้ไว้ก่อน และต้องนำมาแปลอีกครั้งหนึ่ง เช่น โปรแกรมข้างล่างนี้

```
        ORG     100H
        CLR     A
        MOV     RØ,DAT1
        INC     DAT2
                :
                :
DAT1:   DB      ØFFH
DAT2:   DB      Ø18H
        END.
```

จะเห็นว่ากระทงความ MOV RØ,DAT1 และ INC DAT2 นั้นจะไม่สามารถแปลออกมาเป็นภาษาเครื่องได้ จึงต้องเก็บบรรทัดข้อมูลนี้ไว้ก่อน และจะนำมาทำการแปลอีกครั้งหนึ่ง เมื่อทำการแปลซอร์สโปรแกรมทั้งหมดแล้ว

4) การพิจารณาการตรวจดัดจับ (DETECTION) และการแจ้ง (INDICATION) ที่แสดงถึงข้อผิดพลาดในซอร์สโปรแกรม ซึ่งการแจ้งและดักจับนี้ จะเป็นการแสดงให้ผู้เขียนโปรแกรมภาษาแอซเซมบลี้ของซีพียูทราบ และทำการแก้ไขซอร์สโปรแกรมนี้ใหม่

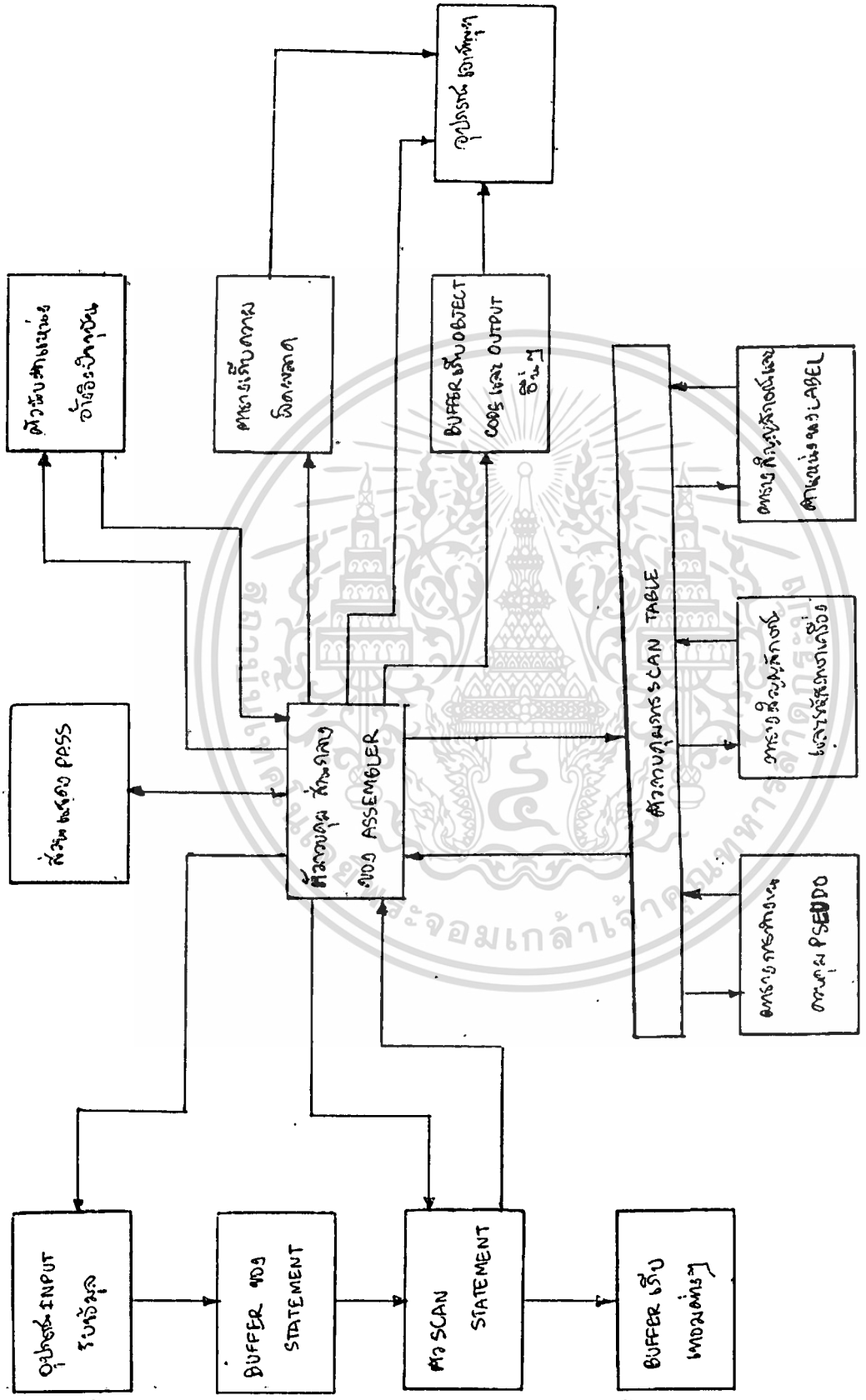
2. โครงสร้างของแอซเซมเบลอร์

ในโปรแกรมแอซเซมเบลอร์ จะต้องมีความสามารถในการจัดการกับซอร์สโปรแกรม ที่จะแปลโดยจัดการกับข้อต่าง ๆ ที่มีขึ้นทั้ง 4 ข้อที่ผ่านมาแล้วได้เป็นอย่างดี ดังนั้นก่อนที่จะพัฒนาหรือเขียนโปรแกรมแอซเซมเบลอร์ได้นั้น ต้องพิจารณาโครงสร้างการทำงานของการประมวลผลพื้นฐาน ที่โปรแกรมแอซเซมเบลอร์จะกระทำต่อซอร์สโปรแกรมก่อน ในรูปที่ แสดงบล็อกไดอะแกรม ความสัมพันธ์ ของฟังก์ชั่นในการทำงานที่ปรากฏในการประมวลผลของแอซเซมเบลอร์ โครงสร้างเริ่มแรกแอซเซมเบลอร์จะต้องมีตารางสำหรับเก็บข่าวสารข้อมูลอยู่สามชุด สำหรับขบวนการในการแอซเซมเบลอร์ (ASSEMBLY PROCESS) โดยแต่ละเทอมของขบวนความที่ผ่านอุปกรณ์อินพุทเข้ามาจะเก็บอยู่ในบัฟเฟอร์กระทงความ จะถูกตัวกวาดกระทงความทำการแยก เป็นเทอมต่าง ๆ เก็บในบัฟเฟอร์เทอม แล้วจากนั้นตัวควบคุมการกวาดตาราง ก็จะนำข้อมูลในบัฟเฟอร์เทอม (BUFFER TERM) มาทำการตรวจสอบหาความหมายโดยเปรียบเทียบกับตารางทั้งสามตารางแล้วส่งผลลัพธ์ของการเปรียบเทียบไปให้ตัวผู้คุม แอซเซมเบลอร์จัดการควบคุมผลลัพธ์ หรือจัดการด้านงานอื่น ๆ ต่อไป

ก่อนอื่นที่จะเริ่มทำการรับข้อมูลจากอินพุทเข้ามาและทำการกวาดกระทงความนั้น ต้องมีการกำหนดตำแหน่งของหน่วยความจำเริ่มต้นก่อน โดยกำหนดให้กับตัวนับตำแหน่งปัจจุบัน (CURRENT LOCATION COUNTER) ซึ่งจะทำให้ตัวนับตำแหน่งปัจจุบันแสดงถึงค่าตำแหน่งของหน่วยความจำที่จะเป็นตัวกำหนด ตำแหน่งการเก็บออปเจคท์โค้ดจากการแปลกระทงความต่อไป ค่าตำแหน่งหน่วยความจำอาจสามารถกำหนดได้ใหม่โดยมาจากซอร์สโปรแกรม เมื่อแอซเซมเบลอร์แปลโดยผ่านในหน่วยของตารางการทำงานควบคุม PSEUDO (เช่นคำสั่ง ORG 100H) ก็จะทำการปรับตำแหน่งหน่วยความจำใหม่ โดยนำค่าที่แปลได้นี้ไปเก็บในตัวนับตำแหน่งปัจจุบัน เมื่อมีการแปลรหัสคำสั่งการทำงานของซอร์สโปรแกรมแล้ว ได้ออปเจคท์โค้ดออกมาก็จะนำออปเจคท์โค้ดนี้ไปเก็บไว้ในบัฟเฟอร์ ที่เก็บออปเจคท์โค้ดที่ตำแหน่งหน่วยความจำ ที่ตรงกับตำแหน่งตัวชี้ แล้วทำการเพิ่มตำแหน่งตัวนับตำแหน่งปัจจุบันขึ้นทีละหนึ่ง โดยอัตโนมัติ เพื่อไม่ให้ตำแหน่งการเก็บข้อมูลของออปเจคท์โค้ดของโปรแกรมซ้ำกัน

ย้อนกลับมาดูตารางการทำงานทั้ง 3 ชุดอีกครั้งหนึ่ง ซึ่งตารางเหล่านี้ก็คือ ตารางการควบคุมของ PSEUDO, ตารางสัญญลักษณ์และรหัสภาษาเครื่อง และตารางสัญญลักษณ์และตำแหน่งของลาเบล โดยมีรายละเอียดดังนี้

- ตารางการทำงานและควบคุม PSEUDO ตารางนี้จะใช้เป็นที่เก็บคำสั่งควบคุมคอม



รูปที่ 3 แสดงบล็อกไดอะแกรมโครงสร้างกระบวนการของแอสเซมเบลอร์

มานด์ PSEUDO ซึ่งแบ่งเป็นสองส่วนคือ ส่วนแรกเป็นที่เก็บคำสั่งแอซเซมเบลอร์ ได้แก่ คำสั่ง ORG และ END เป็นต้น ส่วนที่สอง เป็นส่วนที่เก็บคำสั่งเทียมสำหรับใช้เป็นตัวแปลอ้างอิงได้ หรือนำไปใช้ต่อไป จากการอ้างอิงถึงภายในซอร์สโปรแกรมนั้น ๆ เช่น คำสั่งเทียม EQU, DATA เป็นต้น รูปแบบการเก็บคำสั่งเทียมไว้ในตารางนั้น จะแยกเป็นคำสั่งเทียมของแต่ละชุดตารางด้วย โดยจะเก็บส่วนที่เป็นเทอมของสัญญลักษณ์ที่สามารถใช้อ้างอิงได้ และเก็บส่วนที่เป็นภาษาเครื่อง ที่จะนำไปใช้ในการวางแทนที่ เมื่อมีการอ้างอิงถึงชื่อเทอมสัญญลักษณ์นั้น

- ตารางสัญญลักษณ์ และรหัสภาษาเครื่อง เป็นส่วนที่บรรจุชุดสัญญลักษณ์ของคำสั่งของแต่ละคำสั่ง ที่เก็บไว้แน่นอน พร้อมกับเก็บรหัสภาษาเครื่องที่ตรงกัน ซึ่งจะเห็นได้ว่าตารางนี้ต้องเก็บข้อมูลไว้ก่อน แล้วถูกนำมาเรียกใช้ตรวจสอบชุดสัญญลักษณ์เปรียบเทียบกับเทอมที่รับมาจากกระทงความภายใต้ความควบคุมของตารางการกวาด ตารางสัญญลักษณ์และรหัสภาษาเครื่องต่อไป แล้วรายงานผลนี้ไปให้ตัวควบคุมแอซเซมเบลอร์ทราบ

- ตารางเก็บสัญญลักษณ์และตำแหน่งของลาเบล จะเป็นที่เก็บชุดสัญญลักษณ์ของลาเบลและตำแหน่งหน่วยความจำที่ตรงกับสัญญลักษณ์นั้น จะเก็บลาเบลของซอร์สโปรแกรมแต่ละโปรแกรมที่นำเข้ามาทำการแอซเซมเบลอร์ เราจะเห็นว่าในการแอซเซมเบลอร์โปรแกรมแต่ละโปรแกรมนั้น เมื่อมีการอ้างอิงตำแหน่งหน่วยความจำโดยผ่านชื่อสัญญลักษณ์ลาเบลที่มีตำแหน่งอยู่ด้านล่าง ของซอร์สโปรแกรม ซึ่งเมื่อแอซเซอเบลอร์ทำการกวาดหาตัวชื่อตัวแปลในตารางลาเบลแล้ว จะยังไม่สามารถหาตำแหน่งของหน่วยความจำที่อ้างอิงถึงได้ จึงต้องทำการเก็บกระทงความของบรรทัดข้อมูลนี้ไว้ก่อน พร้อมกับจองตำแหน่งบัฟเฟอร์ไว้ให้ เหตุที่เก็บกระทงความนี้ไว้ ก็เพื่อจะได้นำมาแปลอีกครั้งหนึ่ง เมื่อมีการแปลซอร์สโปรแกรมครั้งแรกเสร็จสิ้นไปแล้ว จะเห็นได้ว่าขบวนการแอซเซมเบลอร์ต้องทำการแปลทั้งสองครั้ง ซึ่งเรียกว่า สองพาส (2-PASS) โดยการกระทำครั้งแรก หรือพาสที่หนึ่งนั้น จะมีการสร้างตารางต่าง ๆ ที่สมบูรณ์ ของซอร์สโปรแกรมนั้น ๆ เมื่อเสร็จสิ้นพาสที่หนึ่งแล้ว ก็จะมีการทำพาสที่สองต่อไป โดยในพาสที่สองนี้จะให้ค่าออปเจคท์โค้ดที่สมบูรณ์ที่สุด

นอกจากตารางทั้ง 3 ที่กล่าวมาแล้ว ยังมีอีกตารางหนึ่งที่มีความสำคัญมากในการประมวลผลของแอซเซมเบลอร์ คือตารางเก็บความผิดพลาด ตารางนี้จะเก็บข้อมูลความผิดพลาดต่าง ๆ ของซอร์สโปรแกรมที่นำมาแปล โดยความผิดพลาดนี้ จะสร้างขึ้นจากส่วนของตัวควบคุมแอซเซมเบลอร์แล้วนำข่าวสารการผิดพลาดไปเก็บไว้ในตารางความผิดพลาดต่อไป เมื่อมีการทำแอซเซมเบลอร์สองพาสแล้ว และมีข่าวสารความผิดพลาดเกิดขึ้น ก็จะนำข่าวสารความผิดพลาดนี้ออกแสดงให้ผู้ใช้ทราบต่อไป

2.1 แอซเซมเบลอร์ พาสที่ 1

ในการประมวลผลของแอซเซมเบลอร์จะกระทำหนึ่งกระทงความต่อหนึ่งบรรทัด โดยแต่ละกระทงความจะถูกอ่านเข้าไปในบัฟเฟอร์ของกระทงความ (STATEMENT BUFFER) กระทงความ (STATEMENT SCANER) จะอ่านคอนเทนท์ของเทอมของบัฟเฟอร์ของกระทงความโดยที่เทอมของคอนเทนท์ที่ได้มา จะอยู่ภายใต้การควบคุมของแอซเซมเบลอร์ เมื่อหน่วยควบคุมเรียกออกไปให้ทำการกวาดกระทงความเพื่อเลือกเทอมจากบัฟเฟอร์กระทงความ และทำการตรวจดูกรณีพิเศษที่กวาดออกมา เช่น เทอมที่มีเครื่องหมายโคลอนประกอบ (:) ซึ่งแสดงว่าเทอมที่กวาดออกมานั้นเป็นเทอมของลาเบล เทอมที่มีเครื่องหมายเซมิโคลอนประกอบ (;) ซึ่งแสดงว่าเทอมนั้นและเทอมที่ตามมาเป็นส่วนของคอมเมนต์ เทอมของตัวเลข เทอมข้อมูล หรือตรวจดูว่าบัฟเฟอร์ของกระทงความนั้นว่างเปล่า เมื่อแบ่งแยกเป็นเทอมได้แล้วก็จะนำไปเก็บในบัฟเฟอร์เทอมต่อไป

เมื่อในบัฟเฟอร์เทอมมีข้อมูล ในการประมวลผลข้อมูลขณะนั้น ก็จะนำข้อมูลในบัฟเฟอร์เทอมมาตรวจสอบอีกว่า เป็นเทอมเหมือนหรือมีลักษณะเป็นไปตามตารางใดบ้าง โดยตัวควบคุมแอซเซมเบลอร์ (ASSEMBLER CONTROL) จะทำการสั่งให้ตัวควบคุมการกวาดตาราง (TABLE SCACH CONTROL) รับทราบถึงชนิดของเทอมข้อมูล ที่อยู่ในเทอมบัฟเฟอร์ จากนั้น ตัวควบคุมกวาดตารางจะตรวจสอบว่าเทอมนั้นควรอยู่ในตารางใด ถ้าหากเป็นเทอมเลเบล และยังไม่มีอยู่ในตารางสัญญลักษณ์ของบาเบล ตัวควบคุมการกวาดจะนำเทอมนั้นไปเก็บในตารางดังกล่าว และก็นำค่าของตัวนับตำแหน่งปัจจุบันไปเก็บไว้ในตารางบาเบล ที่ตรงกับเทอมนั้นด้วย และถ้าหากว่ามีเทอมนี้อยู่ในตารางแล้ว ตัวควบคุมการกวาดตารางจะแจ้ง ความผิดพลาดที่เกิดขึ้นไปให้ตัวควบคุมแอซเซมเบลอร์ทราบ ตัวควบคุมแอซเซมเบลอร์นั้นก็จะนำข่าวสารความผิดนั้นไปเก็บในตารางความผิดพลาดต่อไป

เมื่อเสร็จสิ้นการทำงานของการจัดการกับเทอมแรกแล้ว ตัวควบคุมแอซเซมเบลอร์จะหาเทอมต่อไป ถ้าพบก็จะทำการตรวจสอบ ชนิดของเทอมนั้นต่อไปอีก เป็นวงรอบซ้ำกับที่อธิบายมาแล้ว เมื่อหมดกระทงความแรก(บรรทัดแรก) ของซอร์สโปรแกรมแล้ว ก็จะรอรับกระทงความถัดมา และทำการแปลตามลำดับที่อธิบายมาแล้ว

2.2 แอซเซมเบลอร์ พาสที่ 2

หลังจากทำการแอซเซมเบลอร์ พาสที่หนึ่งเสร็จเรียบร้อยแล้ว จะผ่านขบวนการตรวจสอบหาความผิดพลาดเบื้องต้น และการสร้างตารางต่าง ๆ เสร็จเรียบร้อยแล้ว ก็จะเข้าสู่ขบวนการแปลแอซเซมเบลอร์ พาสที่สอง ในขบวนการพาสที่สองนี้ จะทำการแปลให้ออปเจคท์โค้ดที่สมบูรณ์ซึ่งถ้าหากมีข้อใด ๆ เกิดขึ้นในพาสนี้ ก็จะเป็นความผิดพลาดจาก

การไม่กำหนดลาเบล หรือไม่กำหนดชื่อตัวแปล ในคำสั่งเทียมของซอร์สโปรแกรม ของการเขียนโปรแกรมภาษาแอซเซมบลี้

การทำงานในพาสที่สองนี้ จะคล้ายกับในส่วนของพาสที่หนึ่งโดยเมื่อจัดการกวาดกระทงความที่รับเข้ามาจากอินพุทแล้ว แยกเป็นเทอมแล้ว ก็จะทำการตรวจสอบเทอมที่ได้มาว่าเป็นเทอมพิเศษของอะไร ซึ่งถ้าเป็นเทอมของลาเบล, คำสั่งเทียม คอมเมนต์ หรือเทอมของตัวว่าง ก็จะทำการละเลยไม่ทำการตรวสอบกวาดหาในตารางต่อไป ทั้งนี้เพราะเทอมเหล่านี้จัดการทำในพาสที่หนึ่งแล้ว ดังนั้นจึงกวาดเทอมอื่น ๆ ไปเรื่อย ๆ จนพบเทอมของภาษาแอซเซมบลี้ ก็จะทำการไปกวาดหารหัสออปเจคท์ที่สัมพันธ์กัน กับเทอมคำสั่งที่รับเข้ามา และทำออปเจคท์โค้ดที่ได้นี้ไปเก็บไว้ในบัฟเฟอร์ที่เก็บออปเจคท์โค้ดต่อไป

ในกรณีที่เมื่อถอดออปเจคท์โค้ดแล้ว คำสั่งนั้นต้องมีส่วนขยายตามมาหรือโอเปอร์แรนด์ ส่วนควบคุมก็จะจัดการส่วนกวาดกระทงความจัดการหาเทอมถัดไป มาอีกเทอมหนึ่งและทำการกวาดเปรียบเทียบว่าเป็นการเอาค่าตำแหน่งหน่วยความจำ หรือตัวแปรหรือไม่ ถ้าเป็นการก็ต้องนำเทอมใหม่ถัดมานี้ไปทำการตรวจหารหัสในชุดของตารางอีกครั้งหนึ่งเพื่อให้ได้ออปเจคท์โค้ดที่สมบูรณ์ออกมา กรณีถ้าเป็นตัวเลขโดยตรง ก็จะทำการแปลให้เป็นรหัสตัวเลขโดยตรง เมื่อได้รหัสออปเจคท์โค้ดที่สมบูรณ์ก็นำไปเก็บในบัฟเฟอร์โค้ดและทำการเพิ่มค่าตัวนับปัจจุบัน โดยเพิ่มค่าเท่ากับความยาวของออปเจคท์โค้ดที่แปลออกมาได้ หลังจากนั้น ก็จะไปอ่านอินพุทรับกระทงความใหม่ของบรรทัดถัดไป มาทำการแปลต่อไป จนหมดโปรแกรม

หลังจากแอซเซมเบลอร์ทำการแปลซอร์สโปรแกรมทั้งสองพาสแล้ว ถ้าหากมีข่าวสารผิดพลาดในตารางความผิดพลาด ก็จะนำข่าวสารผิดพลาดเหล่านี้ ออกแสดงทางอุปกรณ์ทางเอาท์พุทให้ผู้ใช้โปรแกรมทราบ ก่อนการสิ้นสุดการทำงานของโปรแกรมแอซเซมเบลอร์

3. <u>งานพิเศษบางอย่างของแอซเซมเบลอร์</u>

เพื่อให้โปรแกรมแอซเซมเบลอร์มีความสามารถในการแปลซอร์สโปรแกรมมากขึ้น ก็จะทำได้โดยเพิ่มเติมการทำงานพิเศษให้มีขึ้นในโปรแกรม ทั้งนี้เพื่อให้ผู้ที่เขียนซอร์สโปรแกรมที่สะดวกขึ้น , ง่ายขึ้น และกระทัดรัดขึ้น แต่อย่างไรก็ตามเมื่อเพิ่มงานพิเศษให้กับโปรแกรมแอซเซมเบลอร์ก็จะทำให้โปรแกรมแอซเซมเบลอร์ที่จะเขียนนั้นมีความยุ่งยากมากยิ่งขึ้น และทำให้โปรแกรมทั้งหมดมีขนาดใหญ่ขึ้นและทำงานได้ช้าลง

จะเห็นว่าในการเพิ่มความสามารถพิเศษให้แก่แอซเซมเบลอร์นั้น จะต้องพิจารณาปัญหาที่เกิดขึ้น เช่น วิธีแก้ไขเปลี่ยนแปลงทำได้อย่างไร, ผลที่ได้รับเมื่อเพิ่มเข้าไปเป็นต้น การเพิ่มความสามารถพิเศษให้แก่แอซเซมเบลอร์ได้แก่ การเพิ่มความสามารถในการ

คำนวณคณิตศาสตร์พื้นฐาน ,การคำนวณขยายแอดเดรส, การเพิ่มการ การป้อนซอร์สโปรแกรมด้วยคำสั่งเทียม, การใช้เลขฐานอื่น ๆ, การให้คำนวณทางโลจิก การอนุญาตให้ใช้คำสั่งพิเศษ การมีฟังก์ชั่นพิเศษให้เป็นต้น

โปรแกรมส่วน Editor

เพื่อความสะดวกและรวดเร็วในการใช้งาน ในการเขียนโปรแกรมของ MCS-51 จึงได้สร้าง Editor ขึ้นมาใช้งานร่วมด้วย ซึ่ง Editor นี้ใช้ง่ายเหมือนเวิร์ดโปรเซส- - เซอร์ ที่ใช้งานอื่นๆทั่วไป

เอดิเตอร์ (Editor แปลว่า บรรณาธิการ) ใช้จัดการข้อมูลที่เป็นอักขระคำ ซึ่งข้อมูลเหล่านี้จะรวมกันเป็นไฟล์ข้อมูล (Text file) หรือ แฟ้มเอกสารต่างๆ จะเห็นได้ว่า เอดิเตอร์ เป็นเครื่องมือพื้นฐานที่ใช้แก้ไขหรือสร้างข้อมูลต่างๆในไฟล์

ชนิดของเอดิเตอร์แบ่งออกเป็น 2 แบบใหญ่ๆคือ

1). เอดิเตอร์แบบที่สามารถแก้ไขข้อมูลได้ทีละบรรทัด (Line oriented Editor)

เอดิเตอร์แบบนี้เหมาะสำหรับการสร้างข้อมูลที่เป็นคำสั่งสั้นๆ เวลาแก้ไขจะทำได้ทีละบรรทัด ตัวอย่างของเอดิเตอร์แบบนี้คือ โปรแกรม Edlin บนดอส

2). เอดิเตอร์แบบที่สามารถเลื่อนตำแหน่งของเคอร์เซอร์ไปยังที่ตำแหน่งใดก็ได้

ซึ่งเราสามารถต่อเติมหรือแก้ไขไฟล์เอกสารได้ทันที เราเรียก เอดิเตอร์แบบนี้ว่า " Full Screen Oriented Editor " เช่น Sidekick เป็นต้น

สำหรับเอดิเตอร์ที่สร้างขึ้นใช้ในโครงงานเป็น Full screen editor ทำให้ง่ายต่อการเปลี่ยนแปลงแก้ไขข้อมูล และยังรวดเร็ว

คุณสมบัติของเอดิเตอร์ที่สร้างใช้ในโครงงาน

1. สามารถเคลื่อนย้ายตำแหน่งเคอร์เซอร์ (Cursor) ซึ่งเป็นเครื่องหมายแสดงตำแหน่งของข้อความที่เราจะแก้ไข เพื่อเลื่อนไปยังตำแหน่งใดก็ได้บนจอภาพ
2. สามารถที่จะลบหรือแทรกข้อมูลภายในหน่วยความจำที่เอดิเตอร์จองไว้ได้ โดยอาจจะลบทีละอักขระ, ทีละคำ, ทีละบรรทัด หรือ เป็นกลุ่มก็ได้
3. สามารถเคลื่อนย้ายกลุ่มข้อมูลหรือคัดลอกกลุ่มข้อมูลไปไว้ ณ.ตำแหน่งเคอร์เซอร์ได้
4. สามารถจัดย่อหน้าของแต่ละบรรทัดได้โดยอัตโนมัติ
5. สามารถป้อนข้อมูลในแต่ละบรรทัดได้ถึง 126 ตัวอักษร/ บรรทัด
6. สามารถเก็บข้อมูลลงดิสค์หรือดึงไฟล์ข้อมูลจากดิสค์มาไว้ในหน่วยความจำเพื่อสร้าง, ต่อเติม หรือ แก้ไขได้อย่างรวดเร็ว
7. สามารถรับไฟล์ข้อมูลได้มากจนหมดหน่วยความจำใช้งานของคอมพิวเตอร์ที่จัดไว้ให้

รูปแบบและการทำงานของ EDITOR

เนื่องจากเอดิเตอร์ที่สร้างขึ้นนั้นเป็นแบบ Full screen editor ฉะนั้นจึงมีฟังก์ชั่นการทำงานของเอดิเตอร์ที่ทำงานค่อนข้างสลับซับซ้อนมาก

เริ่มต้นการสร้างเอดิเตอร์โดยใช้โครงสร้างข้อมูลแบบพอยต์เตอร์ 2 ทาง และข้อมูลแบบอาร์เรย์ 1 มิติ ของตัวอักขระ ซึ่งได้อธิบายมาแล้วในหัวข้อโครงสร้างข้อมูล สำหรับข้อมูลแบบอาร์เรย์ 1 มิติ ของตัวอักขระนั้น จะเป็นข้อมูลของอักขระ 1 บรรทัด ที่กำหนดให้มีขนาด 127 ตัวอักษร แต่เราจะสามารถป้อนข้อมูลได้เต็มที่เพียง 126 ตัว ทั้งนี้เพื่อให้เมื่อมีการจัดเก็บข้อมูลลงดิสค์จะเก็บได้ 127 ตัวพอดี (ตัวอักษรสุดท้าย-- เป็นอักษรควบคุมการขึ้นบรรทัด <Carrier return>) ส่วนข้อมูลแบบ Two-way lined lists นั้น เป็นการนำเอาข้อมูลแต่ละบรรทัดมาต่อกันเพื่อให้เกิดเป็นข้อมูลไฟล์เดียวกัน และสามารถต่อเติมได้ง่าย รวดเร็ว และเก็บข้อมูลได้มากเท่ากับขนาดของหน่วยความจำที่โปรแกรม O.S. ของระบบที่อนุญาติให้ใช้ได้ทั้งหมด

การทำงานของ EDITOR

1). เมื่อเข้าสู่โปรแกรม EDITOR จะทำงานด้วยการค้นหาชื่อไฟล์ข้อมูลที่ผู้ใช้ป้อน ถ้าค้นพบก็จะเปิดไฟล์และอ่านข้อมูลจากไฟล์นั้นลงสู่หน่วยความจำที่จัดไว้เป็นพื้นที่สำหรับการทำงานของ EDITOR และถ้าไม่พบก็จะแจ้งให้ผู้ใช้ทราบ จากนั้นก็จัดการเปิดเป็นไฟล์ข้อมูลใหม่ โดยเริ่มแรกให้เป็นข้อมูลขนาด 1 บรรทัด มีข้อมูลเป็นช่องว่าง(blank)

2). หลังจากเปิดไฟล์ข้อมูลแล้ว ก็จะนำข้อมูลนั้นออกแสดงบนจอภาพให้ผู้ใช้ได้รับรู้ โดยเริ่มแสดงข้อมูลตั้งแต่บรรทัดแรกไป 25 บรรทัด ๆ ละ 80 ตัวอักษร จากนั้นก็แสดงส่วนหัวของเอดิเตอร์ ซึ่งส่วนหัวนี้จะแสดงตำแหน่งของเคอร์เซอร์ที่ชี้อยู่บนข้อมูล และแสดงชื่อไฟล์ข้อมูลที่เปิดทำการปรับปรุงแก้ไขอยู่ จากนั้นจะให้เคอร์เซอร์ไปอยู่ที่บรรทัดที่ 1 และ คอลัมน์ที่ 1

3). รับข้อมูลและควบคุมการเปลี่ยนหรือแก้ไขจากผู้ใช้โดยผ่านทางคีย์บอร์ด เมื่อรับข้อมูลเข้ามา 1 ตัวอักษร แล้วทำการเช็คว่าอักขระที่รับเข้ามา ถ้ามากกว่าหรือเท่ากับ #20h จึงถือว่าเป็นข้อมูล แต่ถ้าอักขระที่รับเข้ามาน้อยกว่า #20h ถือว่าเป็นรหัสควบคุมพิเศษ ซึ่งจะต้องนำมาตรวจสอบว่าเป็นการสั่งให้ทำอะไร ตามรหัสที่รับเข้ามา แล้วทำงานตามคีย์คอลโทรลเหล่านั้น จากนั้นก็รอรับข้อมูลจากคีย์บอร์ดตัวต่อไป

การสร้าง CROSS ASSEMBLER 8051

โครงสร้าง

ครอสแอซเซมเบลอร์ ที่จะสร้างขึ้นนี้จะสร้างบนเครื่องไอบีเอ็มตระกูลพีซี (IBM PC/XT) ภายใต้การจัดการของเอ็มเอสดอส (MSDOS) ซึ่งมีรายละเอียดเกี่ยวกับตัวแอซเซมเบลอร์ดังต่อไปนี้

หลังจากที่ได้ศึกษาทฤษฎีอย่างง่าย ๆ ของหลักการแอซเซมเบลอร์แล้ว แนะนำมาใช้กับการสร้างแอซเซมเบลอร์ 8051 โครงสร้างของครอสแอซเซมเบลอร์ 8051 แสดงรายละเอียดได้ดังในรูปที่ โดยรายละเอียดของแต่ละบล็อกไดอะแกรมจะแสดงได้ดังต่อไปนี้

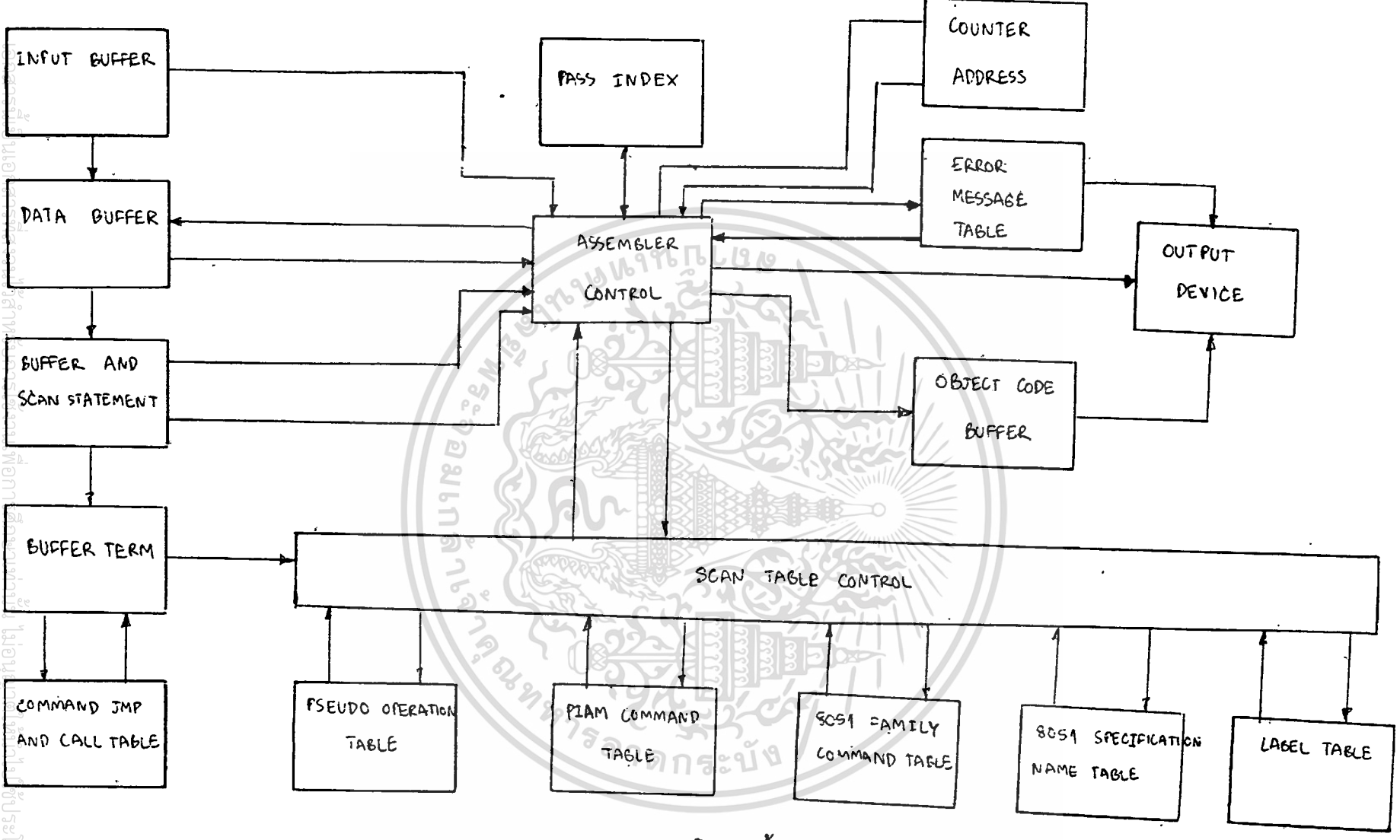
1. ส่วนอินพุทรับข้อมูล (INPUT DIVICE) จะทำการรับข้อมูลจากดิสเก็ตที่อ่านจากไฟล์ข้อมูลที่อยู่ในรูปของ TEXT FILE หรือรับจาหน่วยความจำด้วยกันเป็น SOUCE PROGRAM

2. ส่วนบัฟเฟอร์ข้อมูล (DATA BUFFER) เป็นพื้นที่หน่วยความจำที่รับข้อมูลเข้ามาเก็บไว้โดยผ่านส่วนอินพุทรับข้อมูล ซึ่งข้อมูลที่รับเข้ามาจะรับเข้ามาครั้งเดียว โดยอ่านข้อมูลหมดทั้งไฟล์ ทั้งนี้เพื่อความรวดเร็วในการทำงานของครอสแอซเซมเบลอร์ เพราะไม่ต้องรอติดต่อกับอุปกรณ์อินพุทบ่อยครั้ง

3. ตัวกวาดกระทงความ (SCAN STATEMENT) และบัฟเฟอร์กระทงความ (BUFFER STATEMENT) โดยส่วนนี้เป็นส่วนที่จัดการคัดลอกกระทงความของซอร์สโปรแกรมออกมาครั้งละหนึ่งบรรทัด และทำการตรวจสอบแบ่งแยกเป็นเทอมของกระทงความต่าง ๆ และจัดหาความหมายพิเศษของกระทงความ เพื่อแยกเป็นลักษณะต่าง ๆ ของเทอมว่าเป็นเทอมของอะไร

4. ส่วนตารางเก็บคำสั่ง เกี่ยวกับการกระโดดหรือเรียกโปรแกรมย่อยอื่น ๆ (JAMP AND CALL TABLE) เนื่องจากชุดคำสั่งของซีพียูตระกูล 8051 นั้น มีคำสั่งที่เกี่ยวกับการกระโดดและเรียกโปรแกรมย่อยอื่น ๆ มากมาย และมีรูปแบบการใช้ และการให้ออปเจคท์โค้ดที่ไม่แน่นอน อีกทั้งต้องการมีการคำนวณค่าต่าง ๆ มาก จึงจำเป็นต้องจะเก็บชุดคำสั่งเหล่านี้ไว้เป็นส่วนเฉพาะ สำหรับในการแปลออปโค้ดที่เหมาะสมอีกต่างหากแยกไปจากส่วนอื่น ๆ ของการแปลออปเจคท์โค้ด

5. ส่วนตารางคำสั่งควบคุมแอซเซมเบลอร์ (PSEUDO OPERANTION TABLE) เป็นส่วนตารางเก็บคำสั่งควบคุมการทำงานของครอสแอซเซมเบลอร์ได้แก่คำสั่ง ORG และคำสั่ง END.



รูป 4 แสดงโครงสร้างของคลอสแอสเซมเบลอร์ 8051

6. ส่วนตารางเก็บคำสั่งเทียม (PIAM COMMAND TABLE) ที่ใช้อ้างอิงเป็นตัวแปรได้ในโปรแกรม ซึ่งทำให้ผู้เขียนโปรแกรมพัฒนาซอร์สโปรแกรม ทำได้ง่ายและเข้าใจได้ง่ายด้วย ซึ่งได้แก่คำสั่ง EQU (EQUAL) , คำสั่ง BIT เป็นต้น

7. ส่วนตารางเก็บชุดคำสั่งของ 8051 ทั้งหมด (8051 FAMERY COMMAND TABLE) ส่วนนี้จะเก็บชุดคำสั่งภาษาแอซเซมบลี้ และภาษาเครื่องที่ตรงกันทั้งหมดที่สามารถอ้างอิงได้โดยตรง ในขบวนการแปลของครอสแอซเซมเบลอร์

8. ส่วนตารางเก็บชุดของตำแหน่งและชื่อที่แน่นอนและใช้ได้ทั่ว ๆ ไป (8051 SPECIFICATION CONSTANT NAME TABLE) เนื่องจากในไอซีตระกูล 8051 มีหน่วยความจำแรมภายในตัวเองมาก และยังมีชื่อเรียกใช้เฉพาะทั้งของชื่อที่เป็นไบท์และชื่อที่เป็นบิท ซึ่งชื่อเหล่านี้จะอ้างตำแหน่งของหน่วยความจำแรมได้แน่นอน เราจึงต้องนำมาเก็บไว้ในตาราง เพื่อใช้ในการอ้างอิง เมื่อหาตำแหน่งของมัน เมื่อผู้เขียนซอร์สโปรแกรมเรียกใช้ชื่อตำแหน่งหน่วยความจำเหล่านี้

9. ส่วนตารางเก็บตำแหน่งลาเบล (LABEL TABLE) ซึ่งความสำคัญของมันนั้นอธิบายผ่านมาแล้วจะไม่อธิบายอีก

10. ส่วนควบคุมตรวจหาตาราง (SCAN TABLE CONTROL) ส่วนนี้จะทำการเปรียบเทียบข้อมูลในบัฟเฟอร์เทอม กับข้อมูลต่าง ๆ ที่กำหนด การเลือกชนิดการเปรียบเทียบของตารางใด ๆ นั้น จะรับมาจากส่วนการควบคุมครอสแอซเซมเบลอร์ เมื่อทำการเปรียบเทียบแล้วได้ผลลัพธ์อย่างไรจะส่งผลลัพธ์นั้นกลับไปให้ส่วนควบคุมครอสแอซเซมเบลอร์ต่อไป

11. ส่วนนับตำแหน่งปัจจุบัน (ADDRESS COUNTER) เป็นส่วนที่แสดงค่าตำแหน่งของหน่วยความจำขณะนั้นที่กำลังแปลคำสั่งอยู่ โดยค่าในตัวนับตำแหน่งปัจจุบันนั้นจะกำหนดค่าเริ่มต้นก่อนทำงานใด ๆ มีค่าเป็น 0 และเมื่อทำการแปลคำสั่งแล้ว จะสามารถเปลี่ยนค่าของแอดเดรสได้โดยคำสั่งของส่วนตาราง PSEUDO ในคำสั่ง ORG เช่น เมื่อแปลคำสั่ง ORG 100H ค่าในตัวแปลตำแหน่งปัจจุบัน จะถูกกำหนดให้มีค่าเริ่มต้นใหม่เป็น 100 ฐาน16 ทันที (H=HEXADECIMAL) นอกจากนี้ค่าของตัวนับตำแหน่งปัจจุบันยังมีการเพิ่มค่าโดย เมื่อมีการแปลคำสั่งของซอร์สโปรแกรมแล้ว ได้ออปเจคท์โค้ดออกมา โดยผลที่เพิ่มจะเพิ่มขึ้นเท่ากับจำนวนไบท์ของตัวออปโค้ดที่แปลออกมาได้

12. ส่วนตารางความผิดพลาด (TABLE ERRORS) ส่วนนี้จะเก็บข่าวสารความผิดพลาดที่เกิดขึ้นจากการแอซเซมเบลอร์ซอร์สโปรแกรม และไม่สามารถแปลกระทงความนั้นได้

13. ส่วนอุปกรณ์เอาท์พุท (OUTPUT DIVICE) ส่วนนี้จะเป็นดิสไดร์ (DISK DRIVE) และจอภาพโดยส่วนของไฟล์เอาท์พุทซึ่งสร้างขึ้นจากการแอซเซมเบลอร์ซอร์สโปรแกรมซึ่งอาจจะเป็นออปเจคท์ไฟล์ นำไปเก็บไว้ในดิสไดร์ ส่วนของจอภาพจะเป็นการแสดงผลความผิดพลาดที่เกิดมีขึ้นจากการแอซเซมเบลอร์

14. ส่วนตารางเก็บออปเจคท์โค้ด (OBJECT CODE TABLE) เป็นส่วนของหน่วยความจำที่ใช้เก็บผลของแอซเซมเบลอร์ซอร์สโปรแกรมซึ่งเป็นออปโค้ดทั้งหมด

15. ส่วนชี้พาส (PASS INDEX) เป็นตัวควบคุมให้แอซเซมเบลอร์ว่าในขณะนี้ทำการแอซเซมเบลอร์อยู่ในพาสใด ซึ่งในครอสแอซเซมเบลอร์นี้ จะให้มีการแอซเซมเบลอร์อยู่ด้วยกันสองพาส โดยในพาสที่สองนั้นจะทำการให้ออปเจคท์โค้ดที่สมบูรณ์ทั้งหมด เท่าที่จะหามาได้ของการแอซเซมเบลอร์ซอร์สโปรแกรม

16. ส่วนควบคุมการแอซเซมเบลอร์ (ASSEMBLER CONTROL) ส่วนนี้จะทำการควบคุมส่วนต่าง ๆ ทั้งหมดที่กล่าวมาแล้ว เพื่อให้เกิดการทำงานที่สัมพันธ์กันของแต่ละส่วน และทำให้ขบวนการแอซเซมเบลอร์ได้สมบูรณ์

ขอบเขตความสามารถของครอสแอซเซมเบลอร์ 8051

ครอสแอซเซมเบลอร์ที่สร้างขึ้นนี้จะอาศัยคุณสมบัติโครงสร้างทางพิกสิกส์ และความง่ายสะดวกของผู้ใช้มาก ตลอดจนมีความรวดเร็วในการแปลภาษาเครื่องเป็นพื้นฐาน จึงได้มีการออกแบบสร้างโปรแกรมที่ค่อนข้างกระทัดรัด โดยมีรายละเอียดข้อมูลดังต่อไปนี้

1) กำหนดให้มีความสามารถในการแปลภาษาเครื่องหรือออปโค้ด ได้สูงถึง 64 กิโลไบท์ (KBYTE) ซึ่งค่าขณะนี้เป็นค่าที่ไอซีชิงเกิ้ลชิฟ 8051 สามารถรับทำงานได้สูงสุด เพราะโครงสร้างของพิกสิกส์ตระกูลนี้มีขาแอดเดรสต่อสู่ภายนอกได้เพียง 16 ขา ซึ่งก็สามารถต่อหน่วยความจำได้ 64 KBYTE อยู่แล้ว

2) สามารถอ้างชื่อเฉพาะภายในของหน่วยความจำแรมที่มีอยู่ภายในไอซีตระกูลนี้ได้โดยตรงโดยไม่ต้องผ่านวิธีอื่น ในการเขียนซอร์สโปรแกรม ซึ่งชื่อเฉพาะที่อ้างอิงนี้เมื่อจัดการผ่านการแอซเซมเบลอร์แล้ว ครอสแอซเซมเบลอร์จะจัดการหาตำแหน่งหน่วยความจำนั้นได้โดยอโนมัติ ชื่อตำแหน่งเฉพาะเหล่านี้แบ่งได้เป็นสองชุดคือ

2.1 ชื่อตำแหน่งเฉพาะของไบท์ เช่น P0 (PORT 0) มีค่าแอดเดรสเป็น 080H,PSW (PROGRAM STATUS WORD) มีค่าแอดเดรสเป็น 0D0H. และ TCON มีค่าแอดเดรสเป็น 088H เป็นต้น เมื่อผู้เขียนซอร์สโปรแกรมเพียงแต่อ้างชื่อนี้โดยตรง ตัวครอสแอซเซมเบลอร์ ก็จะให้ออปโค้ดออกมาได้ทันที เช่น

```
74 80          MOV     A,P0
45 88          ORL     A,TCON
74 00          MOV     A,PSW
```

จะเห็นว่าออปโค้ดไบท์หลังของการแปลตัวครอสแอซเซมเบลอร์จะจัดหาให้เอง

2.2 ชื่อตำแหน่งเฉพาะของบิท เช่น P0.2 (บิทที่ 2 ของ พอร์ท P0) มีค่าเป็น 082H, RS0 (บิทที่ 3 ของ PSW มีค่าเป็น 0D3H, IE1 (บิทที่ 3 ของ TCON) มีค่าเป็น 08DH เป็นต้น ซึ่งในการเรียกใช้ก็จะคล้าย ๆ กับ ในหัวข้อของชื่อตำแหน่งไบท์ แต่จะถูกเรียกใช้เฉพาะในชุดคำสั่งการกระทำของบิทเท่านั้น เช่น

```
72 8B          ORL     C,IE1
D3 D3          SETB    RS0
C2 82          CLR     P0.2
```

ในบางกรณีเมื่อผู้เขียนซอร์สโปรแกรมต้องการอ้างบิทของชื่อไบท์ เฉพาะแต่ไม่รู้ชื่อของบิทผู้เขียนก็อาจใช้ชื่อของไบทเฉพาะแทนและตามด้วย จุด (.) และลำดับบิทที่ของไบท์นั้นได้ (เริ่มเรียงจาก 0 ถึง 7) เมื่อไม่รู้ชื่อของบิทที่ 3 ของ PSW (ชื่อ RS0) ก็อาจจะอ้างอิงได้โดยใช้ชื่อ PSW.3 แทน เช่นคำสั่ง

```
CLR     PSW.3
```

ก็จะให้ออปโค้ดออกมาเป็น C2 D3 ได้ ซึ่งค่าของ D3 ก็คือตำแหน่งบิทของ RS0 นั่นเอง

3) ในคำสั่งที่เกี่ยวกับการควบคุมแอซเซมเบลอร์ อนุญาตให้ใช้ได้สองคำสั่ง คือ คำสั่ง ORG และคำสั่ง END. โดยคำสั่ง ORG จะเป็นการกำหนดค่าตำแหน่งของหน่วยความจำที่จะใช้เริ่มต้นเป็นที่เก็บตำแหน่งของออปโค้ด ซึ่งในโปรแกรมหนึ่ง ๆ นั้น ผู้เขียนซอร์สโปรแกรมสามารถกำหนดกี่ครั้งก็ได้ แต่ต้องพึงระวัง ถ้าหากมีการกำหนดแอดเดรสในการแปลตำแหน่งใหม่ โดยที่กำหนดให้มีค่าน้อยกว่า ค่าแอดเดรสตำแหน่งปัจจุบันครั้งหลังสุดที่กำลังแปลอยู่ เพราะจะทำให้พื้นที่ แอดเดรสที่เก็บออปโค้ดนั้นทับกัน ซึ่งยังผลให้ออปโค้ดเก่าที่แปลออกมาและเก็บไว้แล้วจะหายไป ซึ่งจะทำให้โปรแกรมที่เขียนนั้นไม่ถูกต้อง ส่วนคำสั่ง END. เป็นการแจ้งให้ครอสแอซเซมเบลอร์หยุดและสิ้นสุดขบวนการแปลซอร์สโปรแกรมซึ่งจะยังผลให้ซอร์สโปรแกรมที่ตามหลังมานั้น ไม่ได้รับการแปลอีกต่อไป

4) การใช้ข้อมูลที่เป็นเลขฐาน จะให้เขียนได้ 3 ฐานด้วยกันคือ เลขฐาน 2, เลขฐาน 10, และเลขฐาน 16 โดยมีรายละเอียดของแต่ละเลขฐานดังนี้

- เลขฐาน 2 การเขียนจะเขียนด้วยเลข 0 และ 1 เท่านั้น จะเขียนเลขอื่นปนอยู่ไม่ได้ และตัวสุดท้ายของเลขฐานนี้ ต้องตามด้วยการบอกเลขฐาน โดยตามด้วยตัวอักษร b หรือ B (BINARY) ซึ่งถ้าไม่ตามด้วยอักษรนี้ จะถือว่าไม่ใช่เลขฐาน 2 และความยาวของเลขฐาน 2 นั้นจะยาวได้ไม่เกิน 16 ตัวเท่านั้น ตัวอย่างเช่น 0110b , 10001110B เป็นต้น

- เลขฐาน 10 เขียนด้วยตัวเลข 0 - 9 ประกอบกัน โดยไม่มีตัวอักษรอื่นๆ ปนอยู่เลย

- เลขฐาน 16 เขียนด้วยตัวเลข 0 - 9 และอักษร A - F หรือ a - f โดยที่ตัวอักษรตัวสุดท้ายที่ตามมาเมื่อเขียนเสร็จแล้วต้องเป็น H หรือ h (HEXADECIMAL) และกรณีการเขียนที่หลักแรกจะเป็นตัวอักษร A - F หรือ a - f ไม่ได้ ซึ่งถ้าต้องการเขียน ก็ต้องเขียนนำหน้าด้วยเลข 0 ก่อนเสมอ เช่น 19Ah, 0FFH, 012H, 1Ah เป็นต้น

5. การจองข้อมูลในหน่วยความจำ กำหนดให้ใช้คำสั่งใหม่ได้ 2 คำสั่ง คือ คำสั่ง DB (DEFINE BYTE) และ DW (DEFINE WORD) โดยข้อมูลที่ตามมาหรือต้องการใส่เข้าไปนั้น จะเป็นตัวเลขหรือตัวอักษรก็ได้ โดยถ้าเป็นตัวอักษรจะต้องอยู่ในเครื่องหมาย ' ' หรือ " " และเมื่อมีข้อมูลหลาย ๆ ชุดให้เขียนแยกกันด้วยเครื่องหมาย , สำหรับคำสั่งทั้ง 2 ที่กำหนดให้ใช้นั้นมีรายละเอียดดังนี้

- คำสั่ง DB กำหนดให้มีการใส่ข้อมูลได้เป็นไบต์เท่านั้น โดยเข้าไปทีละ 1 ไบต์ เช่น

```
DB      00011111b,0faH,'A',"B",'ABC'
```

เมื่อผ่านการแอซเซมเบลอร์แล้ว จะได้ออปโค้ดเป็น

```
1F FA 41 42 41 42 43
```

- คำสั่ง DW กำหนดให้มีการใส่ข้อมูลได้เป็นเวิร์ดเท่านั้น ซึ่งถ้าเขียนข้อเขียนข้อมูลมาเป็นเพียงไบต์ ก็จะปรับให้เป็นเวิร์ดโดยอัตโนมัติ เช่น

```
DW      00110011b,0FAAh,'A',"BA",'ABC'
```

เมื่อผ่านการแอซเซมเบลอร์แล้ว จะได้ออปโค้ดเป็น

```
3300 AA0F 4100 4200 4100 4100 4200 4300
```

6. กำหนดการใช้เครื่องหมายพิเศษดังนี้

เครื่องหมายโคลอน (:) (COLON) ใช้สำหรับแจ้งแสดงว่าเทอมที่รับเข้ามานั้นเป็น ชื่อตัวตัวแปรของลาเบล โดยที่เครื่องหมายนี้จะต้องประกอบเป็นตัวสุดท้ายของเทอม เช่น LABEL1:

เป็นต้น

เครื่องหมายเซมิโคลอน (;) (SEMICOLON) ใช้สำหรับแจ้งแสดงว่าเทอมที่รับเข้ามานั้น และเทอมต่อไปของกระทงความนั้นเป็นส่วนของคอมเมนท์ (COMMENT) โดยที่เครื่องหมายนี้จะอยู่ที่อักขระแรกของเทอม เช่น " ; This is comment. "

เครื่องหมายดอลล่าร์ ($) (Dollar sign) เป็นการแสดงตำแหน่งของแอดแดรสขณะนั้น ในช่วงระหว่างคลอสแอสเซมเบลอร์แปรกระทงความนั้นอยู่ เช่น STMP $

เครื่องหมายบวก ลบ (+,-) เป็นการแสดง ใช้ประกอบกับตัวเลข ซึ่งถ้าเป็นเครื่องหมายบวกจะมีการจัดการแบบปกติคือไม่คิดเครื่องหมาย แต่ถ้าเป็นเครื่องหมายลบ จะมีการคิดเป็นแบบเลข Two's complement (2's) เช่น LABEL DB -2AH เมื่อมีการเก็บข้อมูลจะจัดเก็บเป็น C6

เครื่องหมาย @ ใช้ในส่วนของโอเปอร์แรนท์ซึ่งอยู่เป็นอักขระแรกของเทอม เป็นตัวกำหนดว่าเทอมที่รับเข้ามาเป็นเทอมของการอ้างแอดแดรสชิ่งโหมด

เครื่องหมาย # ใช้ในส่วนของโอเปอร์แรนท์ซึ่งอยู่เป็นอักขระแรกของเทอม เป็นตัวกำหนดว่าเทอมที่รับเข้ามาเป็นเทอมของข้อมูล

เครื่องหมาย , เป็นตัวใช้แยกโอเปอร์แรนท์ของคำสั่งออกเป็น 2 ส่วน เพื่อใช้ในการพิจารณาแปลออปเจ็คโค้ด

เครื่องหมาย . เป็นตัวกำหนดว่าเทอมที่รับเข้ามามีการแยกย่อยได้อีก ในการอ้างคำสั่งที่เกี่ยวกับการกระทำของบิทแต่อ้างในเทอมของไบท์ เช่น SETB P3.Ø

7. กำหนดให้ใช้คำสั่งเทียมได้ 3 คำสั่ง (คำสั่งสำหรับสร้างตัวแปร) ซึ่งใช้เพื่อความง่ายต่อการเข้าใจและการเขียน Source Program ทั้ง 3 คำสั่งนี้คือ คำสั่ง EQU (Equal) , DATA , BIT ซึ่งแต่ละคำสั่งมีรายละเอียดข้อกำหนดและการใช้งานดังต่อไปนี้

- คำสั่งเทียม EQU ใช้สำหรับกำหนดชื่อตัวแปลทั่วๆไป ซึ่งอาจกำหนดให้มีขนาด 16 บิท หรือ 8 บิท ก็ได้โดยในการเรียกใช้นั้นจะเรียกได้กับคำสั่งทั่วๆไป ตัวอย่างเช่น

```
1).   ADDRESS    EQU    Ø100H
      ORG        ADDRESS
```

```
2).     PAGE        EQU     02
                    MOV     A , #PAGE
```

ในตัวอย่างที่ 1 จะเห็นว่ากำหนดให้ตัวแปลชื่อ ADDRESS มีค่าเท่ากับ 100h และเมื่อเรียกใช้โดยคำสั่ง ORG ก็จะเรียกใช้ชื่อตัวแปลได้โดยตรง

ในตัวอย่างที่ 2 กำหนดให้ PAGE มีค่าเป็น 02 และเมื่อเรียกใช้โดยคำสั่ง MOV ก็จะเรียกใช้ชื่อตัวแปลได้โดยตรง ทั้ง 2 ตัวอย่างนี้ เมื่อผ่านกระบวนการคลอสแอสเซมเบลอร์มันจะจัดการหาโค็ดได้โดยตรง

- คำสั่งเทียม DATA ใช้สำหรับการกำหนดค่าให้แก่ตัวแปรเช่นเดียวกับคำสั่งเทียม EQU แต่สามารถกำหนดให้ตัวแปรได้ขนาดเพียง 8 บิทเท่านั้น และในการเรียกใช้นั้นจะเรียกใช้ได้เฉพาะในคำสั่งที่เกี่ยวกับการอ้างแอดแดรสขนาด 8 บิท และ คำสั่งที่เกี่ยวกับการย้ายข้อมูลแบบไบท์เท่านั้น ตัวอย่างเช่น

```
DATA1       DATA        08H
DATA2       DATA        0FFH
ADDRESS_8   DATA        054H
            ORG         00H
            MOV         A,#DATA1
            MOV         R0,#DATA2
            MOV         @R0,ADDRESS_8
```

- คำสั่งเทียม BIT ใช้สำหรับการกำหนดให้ค่าแก่ตัวแปรเหล่านี้ จะนำไปใช้ได้เฉพาะคำสั่งที่เกี่ยวกับการกระทำเกี่ยวกับคำสั่งบิทเท่านั้น ไม่สามารถนำไปใช้เกี่ยวกับคำสั่งอื่นๆได้ ตัวอย่างเช่น

```
BIT1        BIT         23H
BIT2        BIT         7FH
            ORG         100H
            MOV         C,BIT1
            SETB        BIT2
```

จะเห็นว่าชื่อบิทที่กำหนดตั้งขึ้นและเรียกใช้นั้นจะเป็นการนำตำแหน่งบิทที่มีอยู่ในหน่วยความจำ RAM ภายในตัวชิงเกิลชิฟต์เอง

8. กำหนดให้มีการใช้คำสั่งพิเศษ LOW() และ HIGH() สำหรับใช้ในการอ้างอิงเพียงเฉพาะไบท์ใดไบท์หนึ่งของหน่วยความจำได้ โดยค่าที่อยู่ในวงเล็บนั้นจะเป็นชื่อ

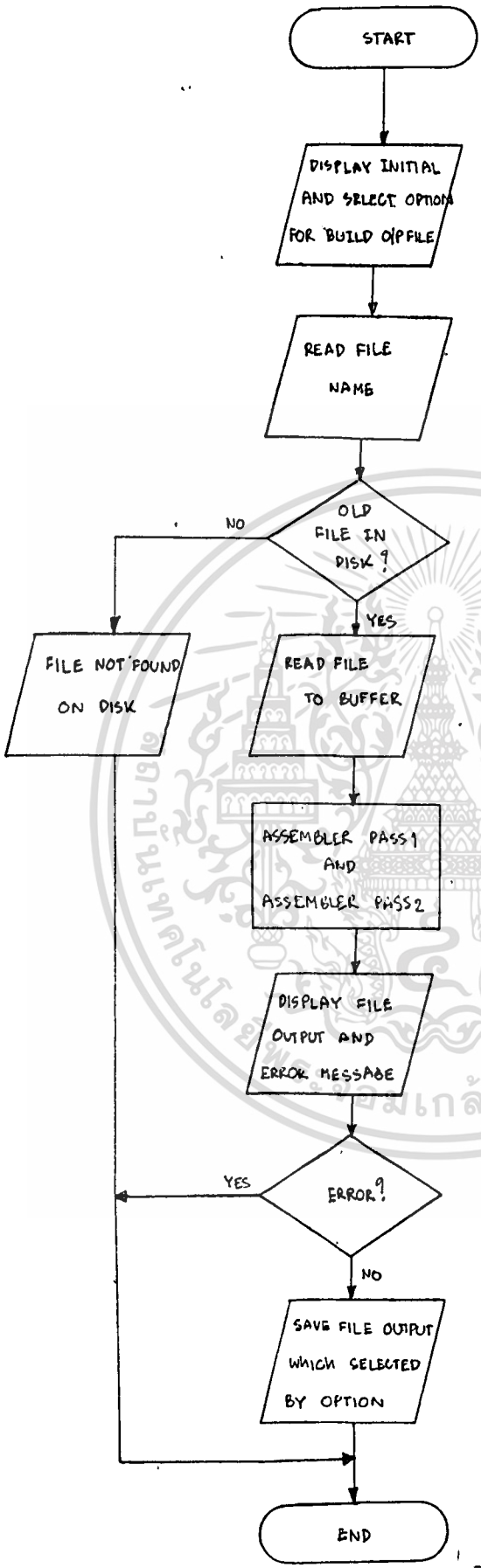
ตัวแปลของลาเบลเท่านั้น โดยคำสั่ง LOW() เป็นการนำเอาไบท์ต่ำของแอดแดรสหน่วยความจำออกมา ส่วนคำสั่ง HIGH() เป็นการนำเอาไบท์สูงของแอดแดรสหน่วยความจำออกมา ตัวอย่างเช่น

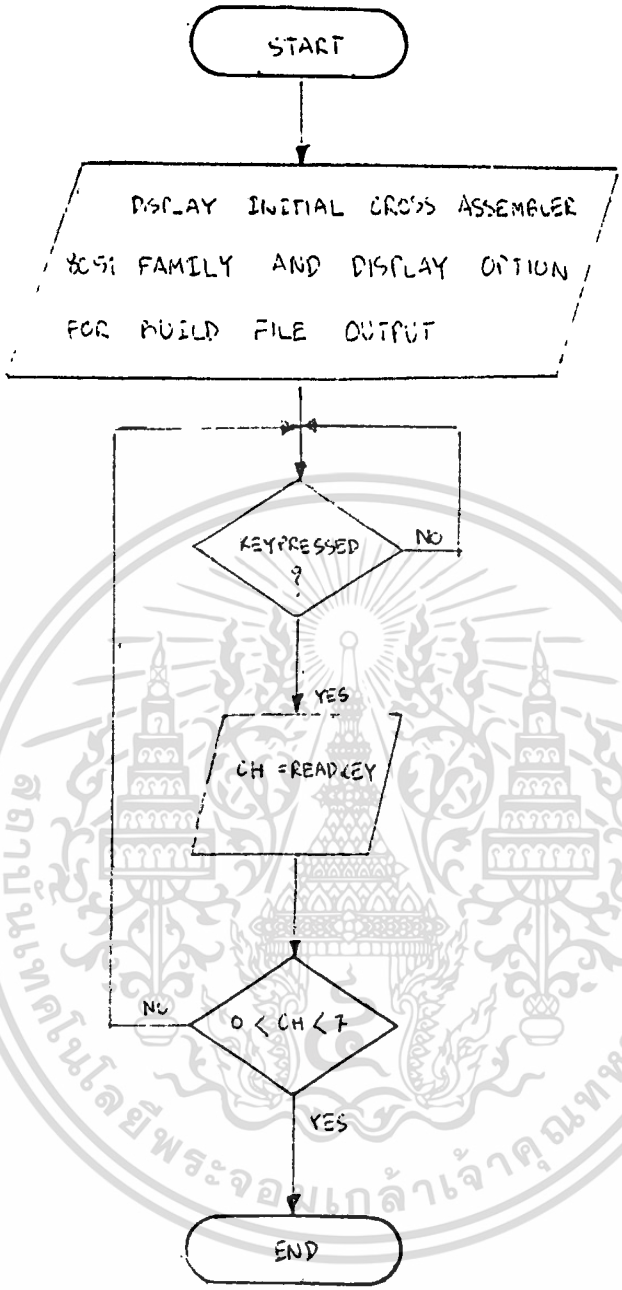
```
        ORG     00H
        MOV     A,INDEX
        ADD     A,LOW(BASE)
        MOV     DPL,A
        MOV     A,#HIGH(BASE)
        MOV     DPH,A
        ..............
BASE:   DB      0,1,2,3
        DB      012,1AH,00110011B
        ..............
```

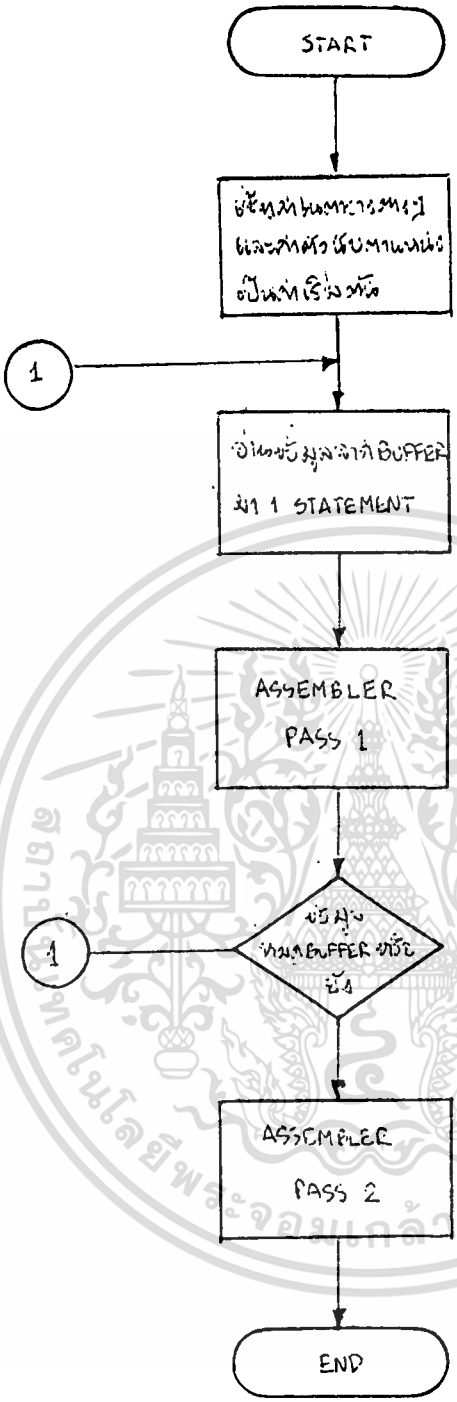
9. กำหนดให้มีการใช้ชุดตัวอักษรประกอบเป็นชื่อตัวแปรสำหรับชื่อในคำสั่งเทียมชื่อลาเบล ซึ่งประกอบด้วยชุดตัวอักษร A..Z , a..z , 0..9 และ _ เท่านั้น โดยตัวอักษรตัวแรก จะขึ้นต้นด้วยกลุ่มของตัวอักษร 0 - 9 ไม่ได้ และความยาวสำหรับชื่อตัวแปรนั้นจะใช้ชื่อตั้งได้ยาวไม่เกิน 17 ตัวอักษรเท่านั้น ซึ่งก็ยาวเพียงพอ สำหรับการตั้งชื่อใช้งานโดยไม่ซ้ำกัน

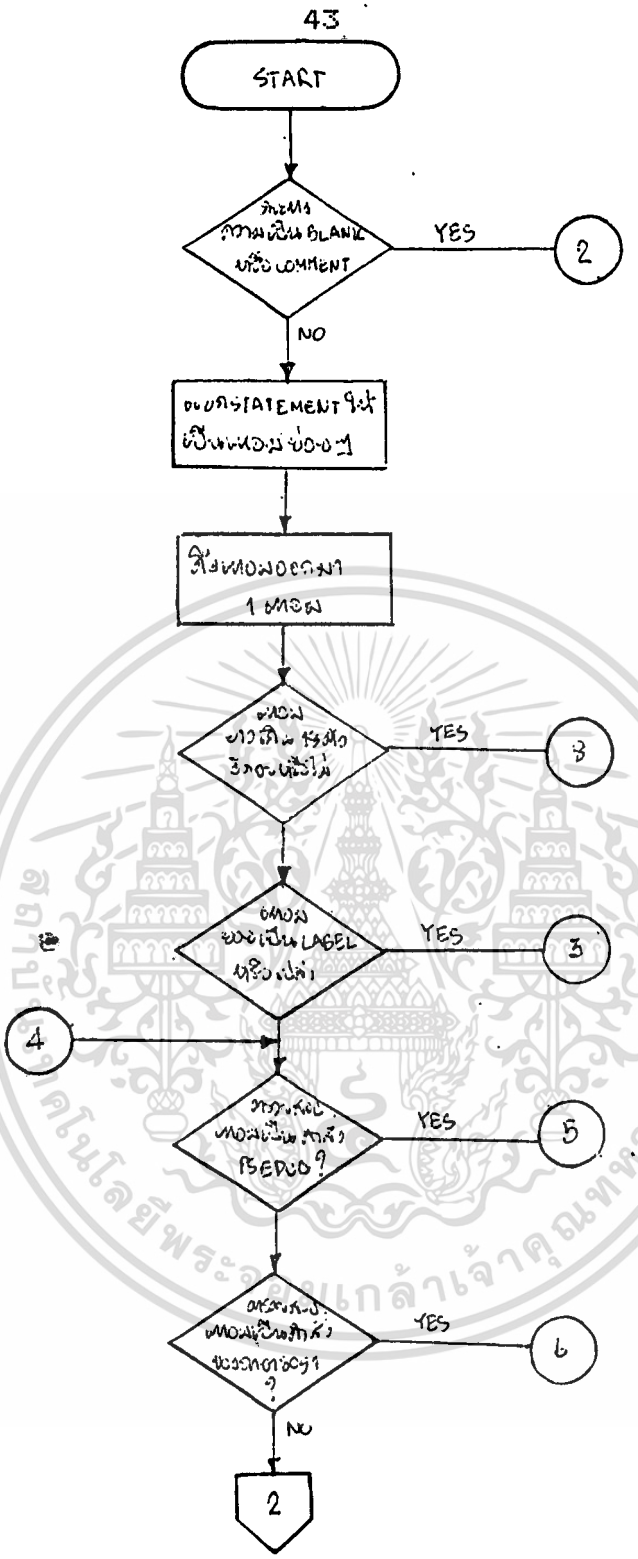
<u>การทำงานของ CROSS ASSEMBLER 8051</u>

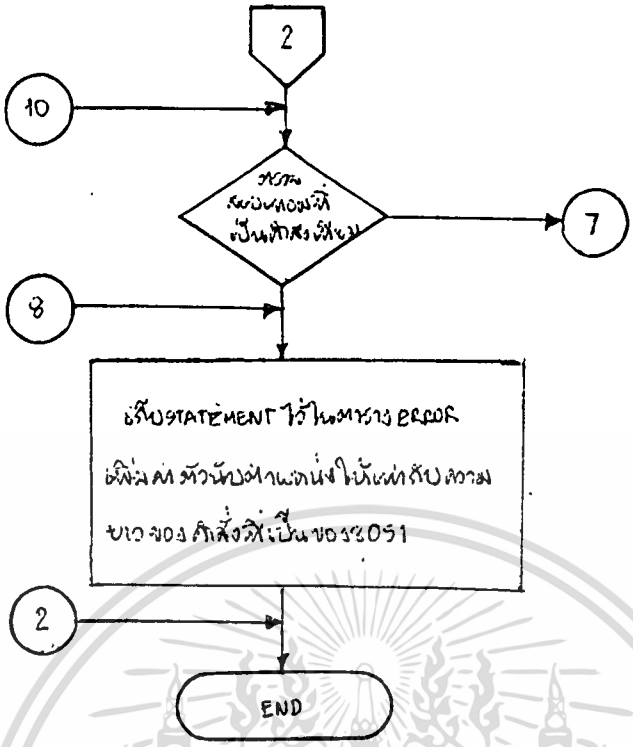
เพื่อความง่ายต่อการทำความเข้าใจ จะแสดงขบวนการแอสแซมของคลอสแอสแซมเบลอร์ 8051 ออกเป็นส่วนๆ โดยเริ่มต้นแบ่งเป็นส่วนใหญ่ก่อน จากนั้นจึงแบ่งแยกอธิบายแต่ละส่วนออกเป็นส่วนย่อยๆอีกครั้งหนึ่งภายหลังโดยเขียนอธิบายในรูปของ FLOW CHART ทั้งหมดโดยมีรายละเอียดดังต่อไปนี้

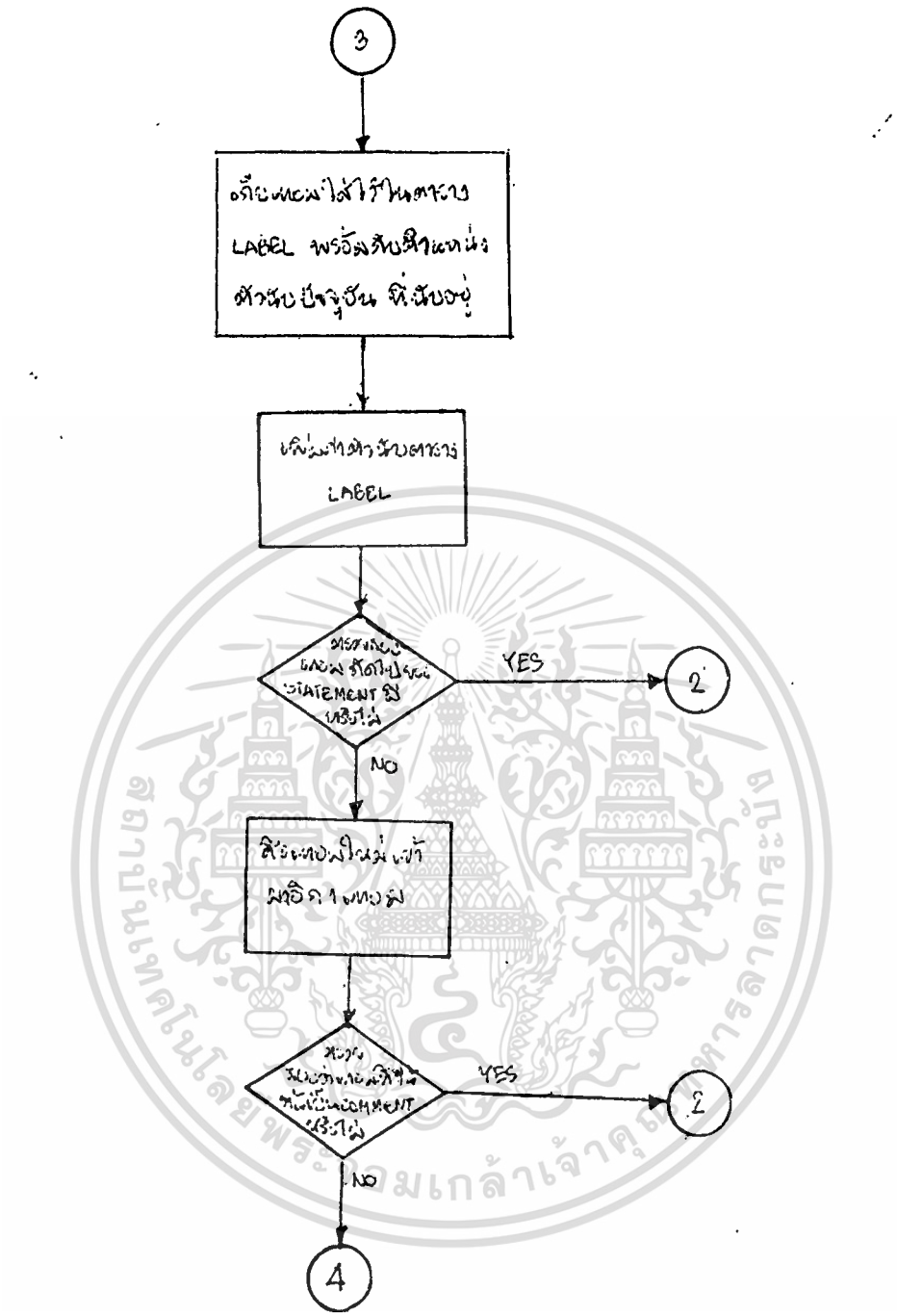


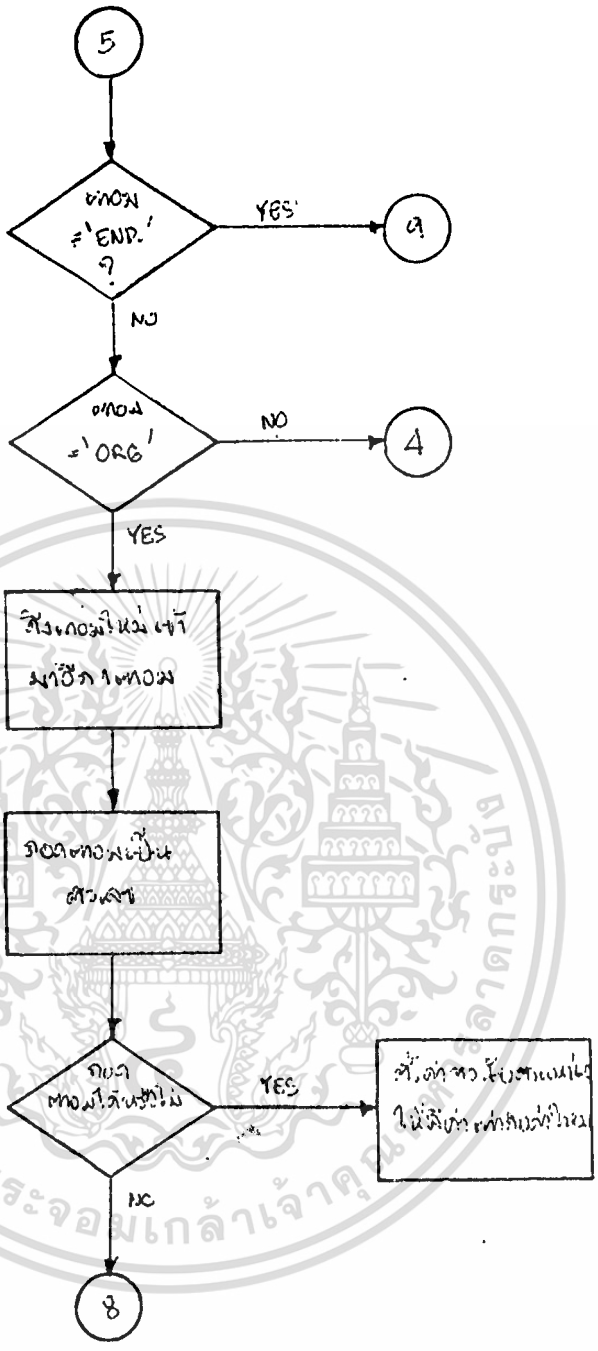


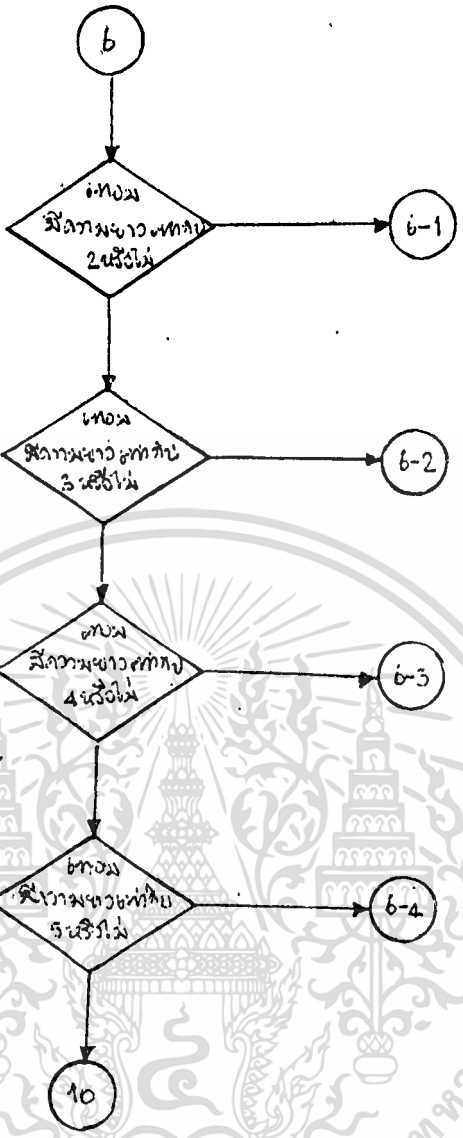


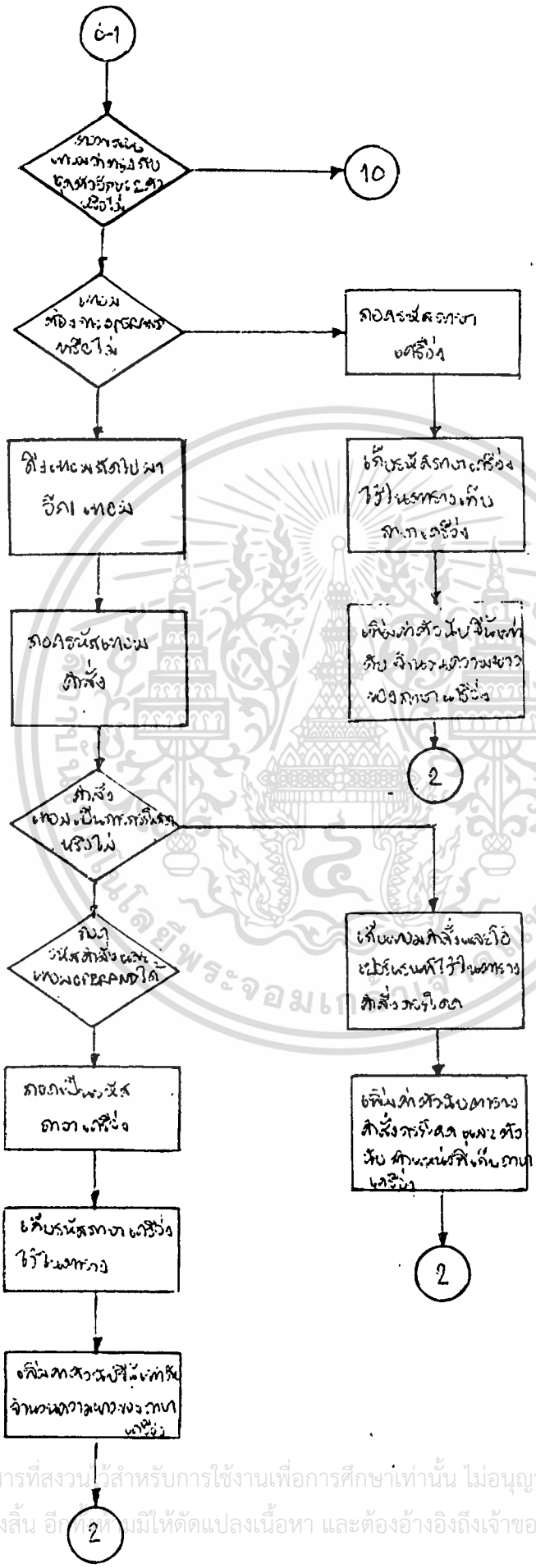


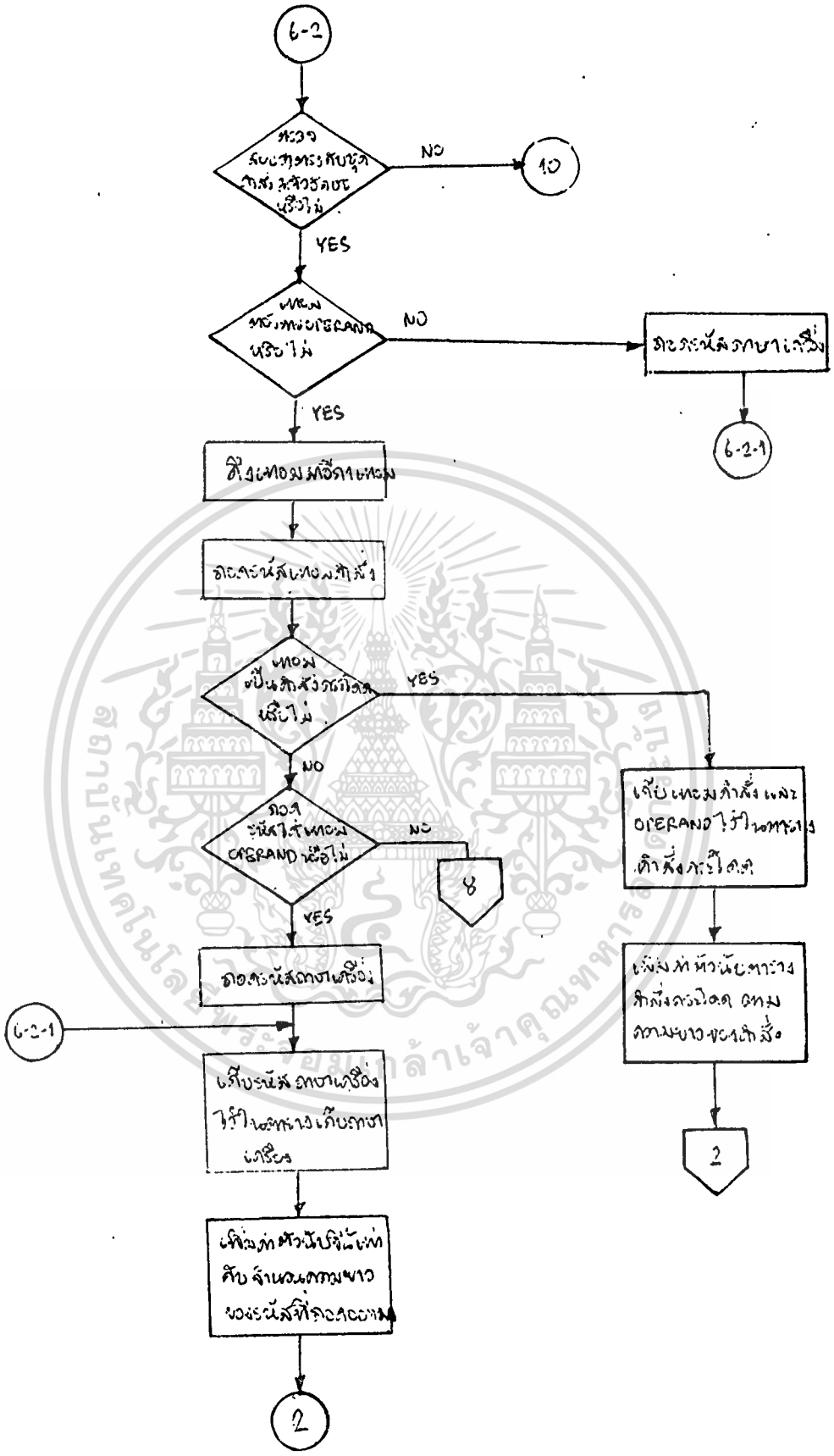


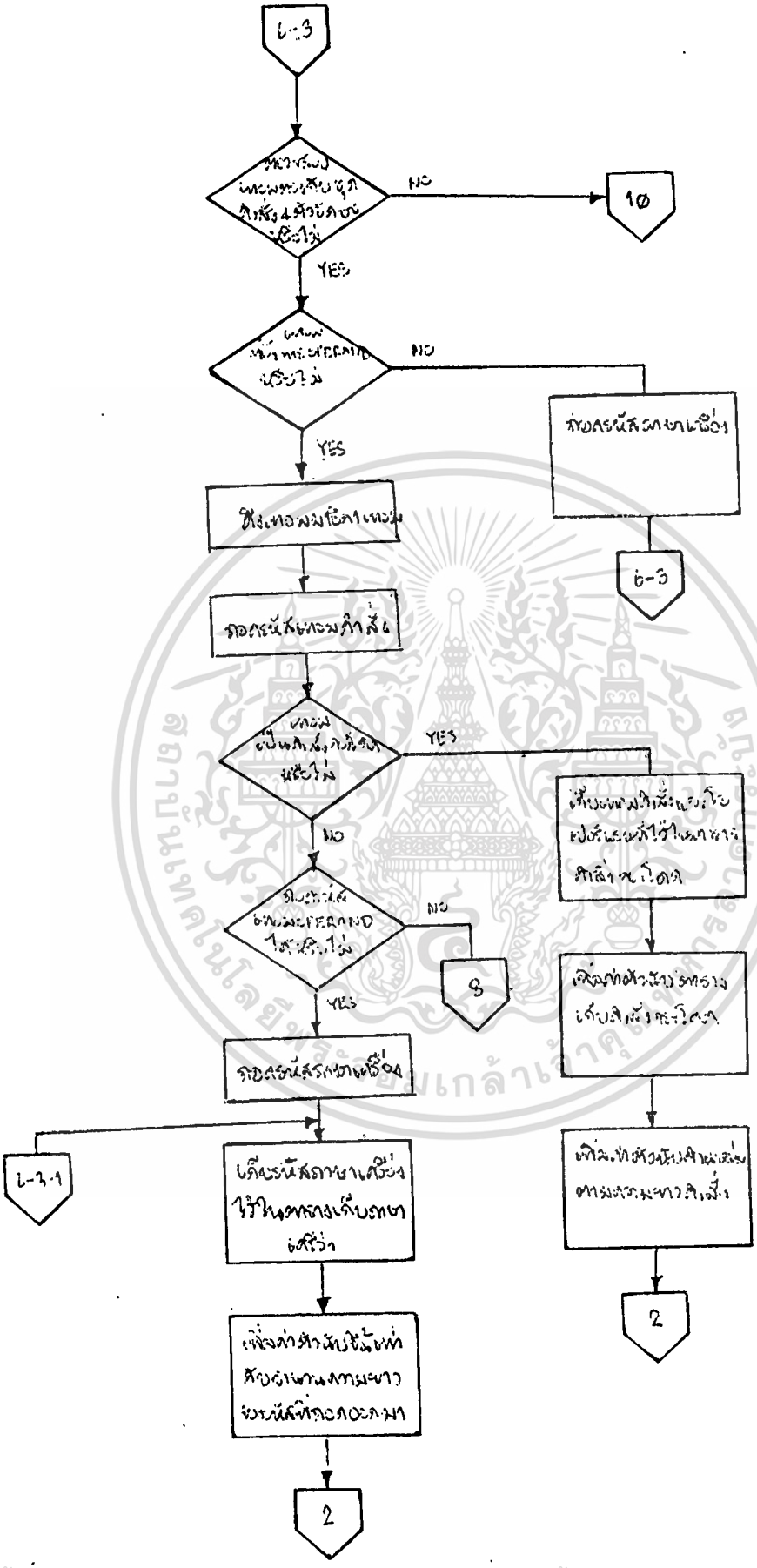


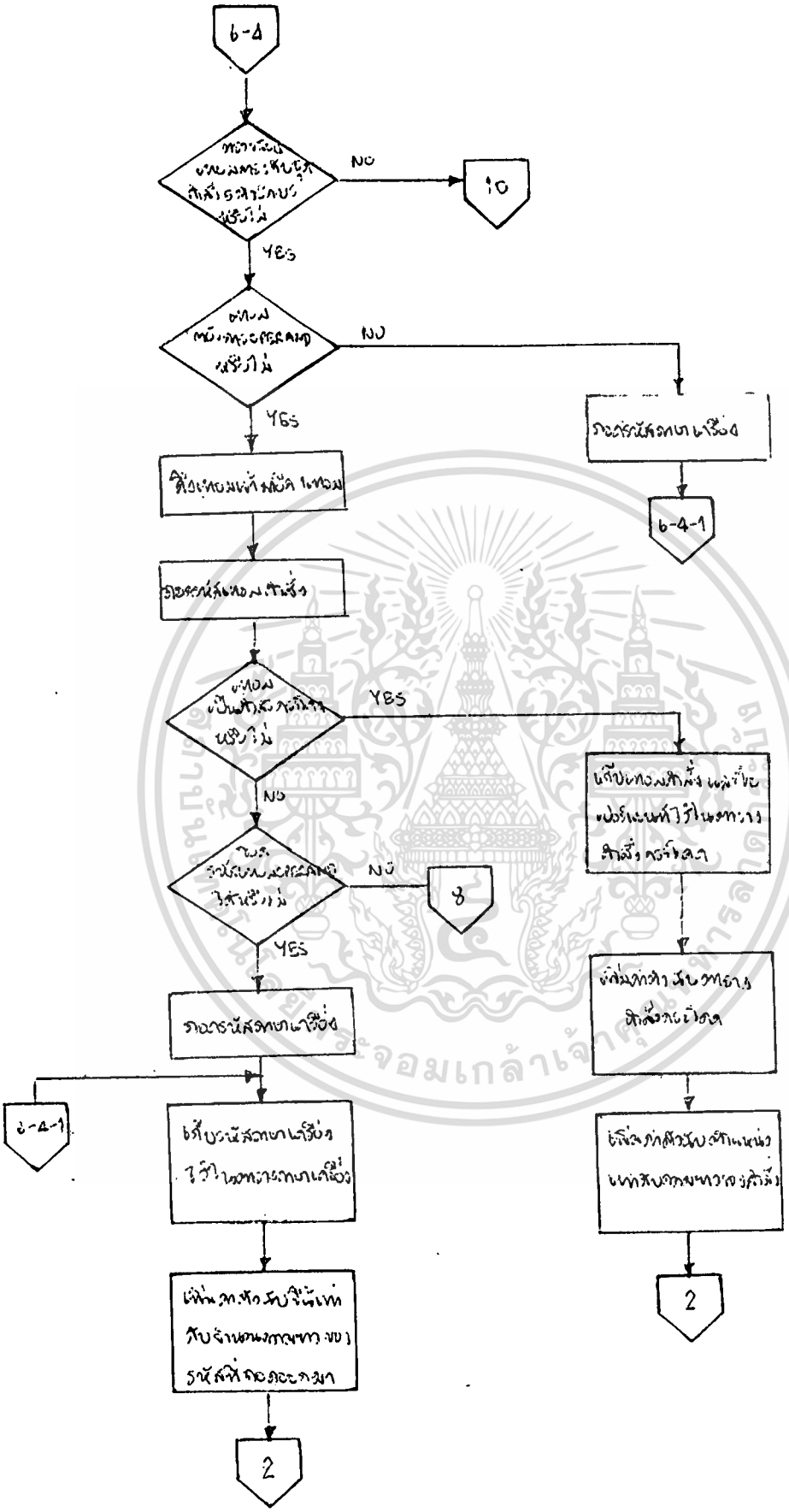


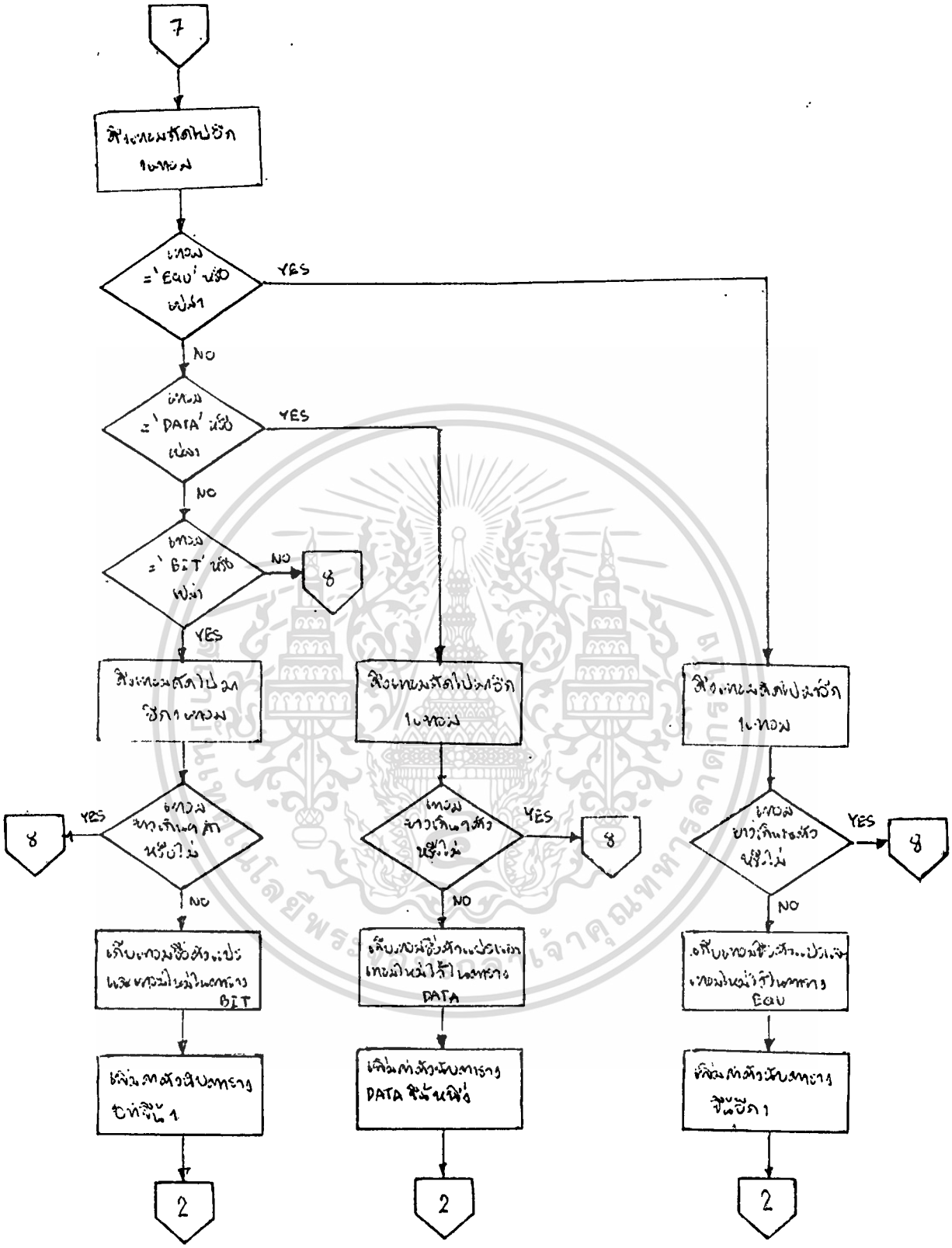


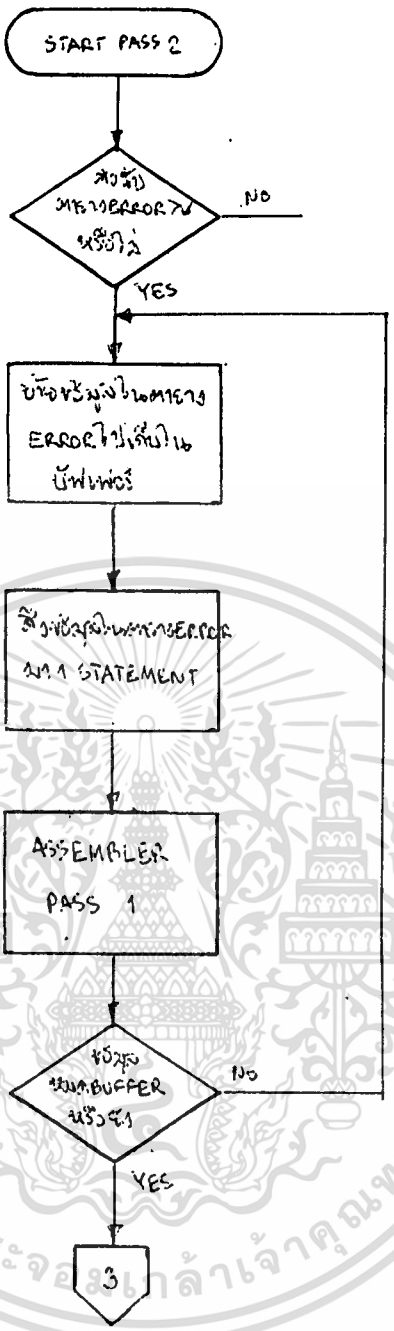


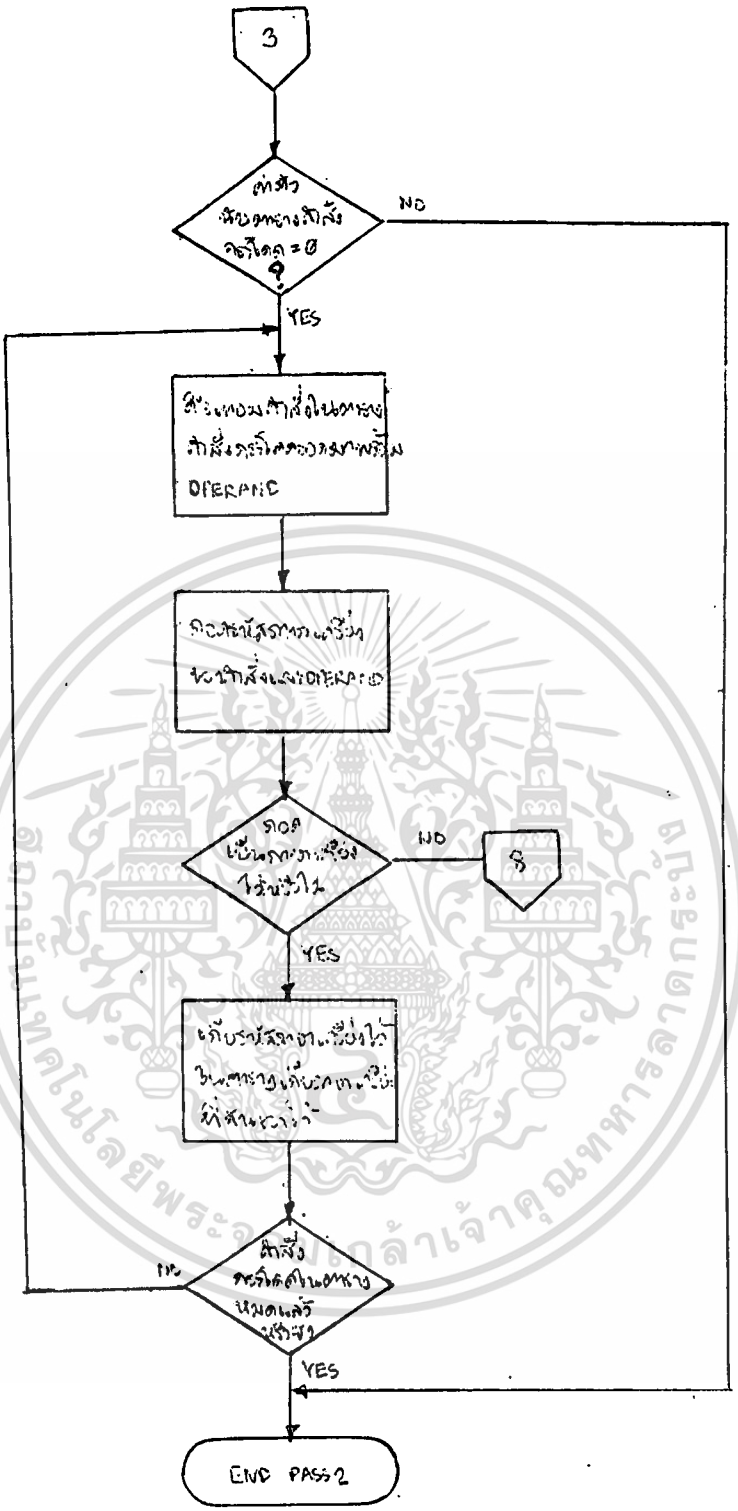


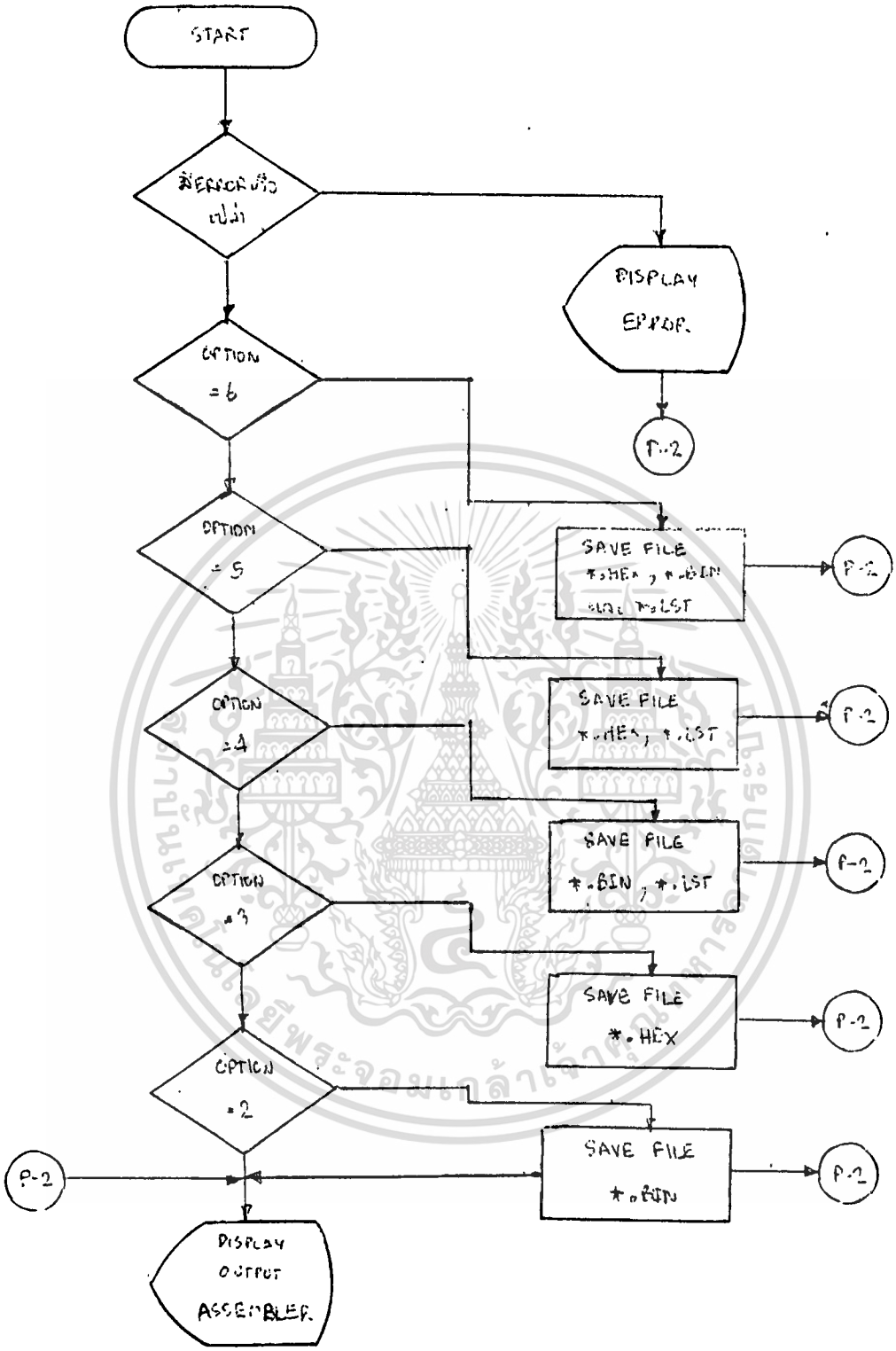












หลักการสร้างคอลโทรลคีย์ต่างๆของ EDITOR

การทำงานของการรับข้อมูลจากคีย์บอร์ดนี้ จะทำงานเป็นวงรอบ (Loop) ร่วมกับการจัดการกับไฟล์ข้อมูลและการแสดงผลของไฟล์ข้อมูลให้ผู้ใช้ทราบ ดังนี้

- อักขระข้อมูลที่มีรหัส ASCII มากกว่า #31 (ฐานสิบ) จะถือว่าเป็นข้อมูลของผู้ใช้ที่ต้องการป้อนเก็บไว้ในไฟล์ ซึ่งโปรแกรมจะทำการตรวจสอบโหมดของการต่อเติมไฟล์ ถ้าเป็นโหมดแทรกข้อมูล (Insert Indent) ก็จะทำการเลื่อนข้อมูลไปทางขวา 1 อักขระ โดยเริ่มเลื่อน ณ.ตำแหน่งที่เคอร์เซอร์อยู่ แล้วนำอักขระที่รับมาจากคีย์บอร์ดใส่ไว้ในหน่วยความจำตามตำแหน่งนั้นๆ การทำงานในโหมดนี้ ก่อนที่จะแทรกข้อมูลลงไปจะทำการตรวจสอบข้อมูล ณ.คอลัมน์ที่ 126 ของบรรทัดที่เคอร์เซอร์อยู่ก่อน ถ้าที่ตำแหน่งนี้ไม่มีข้อมูลจึงจะทำการแทรกข้อมูลได้ แต่ถ้ามีข้อมูลอยู่จะแทรกข้อมูลในบรรทัดนี้ไม่ได้ ถ้าเป็นการทำงานในโหมดเขียนทับ (Rewrite) โปรแกรมจะนำข้อมูลที่รับเข้ามาจากคีย์บอร์ดไปใส่แทนที่ข้อมูลเดิมที่เก็บไว้ที่ตำแหน่งตัวชี้ไฟล์ข้มูลชี้อยู่ หลังจากนั้นจะเลื่อนตัวชี้ไฟล์ข้อมูลไปทางขวาอีก 1 คอลัมน์ จากนั้นก็จะแสดงผลไฟล์ข้อมูลใหม่บนจอภาพพร้อมกับเลื่อนตำแหน่งเคอร์เซอร์ไปทางขวาอีก 1 คอลัมน์ ซึ่งจะเห็นว่าการเลื่อนตัวชี้ในไฟล์ข้อมูลและตำแหน่งเคอร์เซอร์ที่แสดงบนจอภาพของไฟล์ข้อมูลนั้นจะต้องสัมพันธ์กันจึงจะทำให้การเปลี่ยนแปลงไฟล์ข้อมูลและการแสดงผลของการเปลี่ยนแปลงนั้นถูกต้องตรงกัน

กลุ่มคีย์ควบคุมการเคลื่อนย้ายตำแหน่ง

- Ctrl A (^A) เป็นฟังก์ชั่นที่จะทำการเลื่อนตัวชี้ไปทางซ้าย 1 คำ โดยจะเลื่อนไปชี้ที่อักขระแรกของคำที่อยู่ทางซ้ายของคำที่ตัวชี้ฯคำปัจจุบันอยู่ ถ้าในกรณีคำทางซ้ายในบรรทัดนั้นไม่มี ก็จะเลื่อนไปชี้คำสุดท้ายของบรรทัดก่อนหน้านี้ และถ้าบรรทัดปัจจุบันเป็นบรรทัดแรก ตัวชี้ตำแหน่งข้อมูลจะเลื่อนไปชี้อักขระตัวแรกของคำแรก
- Ctrl S (^S) หรือ (<-) เป็นการเลื่อนตัวชี้ตำแหน่งไฟล์ข้อมูลไปทางซ้าย 1 อักขระ ถ้าตัวชี้อยู่ที่คอลัมน์แรกของบรรทัด ก็จะไม่มีการเลื่อนตัวชี้ จากนั้นก็เลื่อนตำแหน่ง เคอร์เซอร์ให้ตรงกับตำแหน่งของตัวชี้ไฟล์ข้อมูลซึ่งแสดงบนจอภาพ
- Ctrl D (^D) หรือ (->) เป็นการเลื่อนตัวชี้ตำแหน่งไฟล์ข้อมูลไปทางขวา 1 อักขระ พร้อมกันนั้นก็เลื่อนเคอร์เซอร์ไปทางขวาด้วย ในกรณีที่เคอร์เซอร์เลื่อนมาอยู่ที่คอลัมน์ 80 แล้วกดต่อไปอีกข้อมูลบนจอภาพทั้งหน้าจะเลื่อนไปทางซ้าย 1 ตำแหน่ง ตำแหน่งเคอร์เซอร์ก็ยังอยู่ที่คอลัมน์ 80 แต่เลขแสดงคอลัมน์จะเพิ่มขึ้นอีก 1 จนถึงคอลัมน์ที่ 126 จึงจะไม่มีการเลื่อนของข้อมูลอีก

- Ctrl F (^F) เป็นการเลื่อนตัวชี้ข้อมูลและเคอร์เซอร์ไปชี้ที่อักขระตัวแรกของคำที่อยู่ถัดไปทางขวามือ ซึ่งถ้าไม่มีคำทางขวาอยู่ในบรรทัดนั้นก็จะเลื่อนมาชี้ที่คำแรกของบรรทัดสุดท้ายแล้วก็จะไม่มีการเลื่อนตำแหน่งของตัวชี้และเคอร์เซอร์ สำหรับการแสดงผลบนจอภาพนั้น ถ้าตำแหน่งที่ชี้เกินกว่า 80 คอลัมน์หรือยังไม่มีบรรทัดข้อมูลของการชี้แสดงอยู่ ซึ่งไม่มีการแสดงตำแหน่งของข้อมูลให้เห็นบนจอภาพ ก็จะจัดการแสดงผลข้อมูลพร้อมกับตำแหน่งของเคอร์เซอร์
- Ctrl E (^E) หรือ คีย์ลูกศรขึ้นข้างบน เป็นการเลื่อนตำแหน่งตัวชี้ข้อมูลและเคอร์เซอร์ขึ้นบน 1 บรรทัด ที่คอลัมน์เดิม ซึ่งถ้าข้อมูลเดิมเป็นบรรทัดแรกอยู่แล้วจะไม่มีการเปลี่ยนแปลง ถ้าไม่ใช่บรรทัดที่ 1 ก็จะเลื่อนข้อมูลบนจอภาพลง 1 บรรทัด แล้วพิมพ์ข้อมูลบรรทัดก่อนหน้านั้น พร้อมทั้งเลื่อนตัวชี้ข้อมูลให้ตรงกันด้วย
- Ctrl X (^X) หรือ คีย์ลูกศรลงข้างล่าง เป็นการเคลื่อนย้ายตำแหน่งตัวชี้ข้อมูลลงมาบรรทัดล่าง 1 บรรทัด พร้อมกับเลื่อนตำแหน่งเคอร์เซอร์ลงมาบรรทัดล่าง 1 บรรทัดด้วย ถ้าข้อมูลที่ชี้อยู่เป็นบรรทัดสุดท้ายจะไม่มีการเปลี่ยนแปลง แต่ถ้าข้อมูลไม่ใช่บรรทัดสุดท้ายแต่เคอร์เซอร์อยู่บรรทัดสุดท้ายก็จะเลื่อนข้อมูลบนจอภาพขึ้นบน 1 บรรทัด แล้วพิมพ์ข้อมูลบรรทัดล่างหลังจากนี้ 1 บรรทัดที่บรรทัดล่างสุด
- Ctrl W (^W) เป็นการเลื่อนการแสดงผลข้อมูลบนจอภาพเลื่อนลงมาข้างล่าง 1 บรรทัด ถ้าตำแหน่งของเคอร์เซอร์ไม่อยู่ที่บรรทัดสุดท้ายตำแหน่งของเคอร์เซอร์ก็จะเลื่อนตามลงมา 1 บรรทัดด้วยเพื่อให้ชี้ตรงตามบรรทัดข้อมูลปกติ แต่ถ้าตำแหน่งอยู่บรรทัดสุดท้ายอยู่แล้วตำแหน่งของเคอร์เซอร์จะไม่ชี้เลื่อนแต่ตัวชี้ข้อมูลจะถูกเลื่อนขึ้น 1 บรรทัด ถ้าการแสดงผลข้อมูลบรรทัดบนสุดเป็นการแสดงบรรทัดแรกของไฟล์ข้อมูลอยู่แล้ว จะไม่เกิดการเลื่อนข้อมูล
- Ctrl Z (^Z) เลื่อนการแสดงผลข้อมูลขึ้นบน 1 บรรทัด โดยถ้าการแสดงผลบรรทัดล่างสุดของข้อมูลยังไม่ใช่บรรทัดล่างสุดของไฟล์ข้อมูลก็จะเกิดการเลื่อนข้อมูล และถ้าข้อมูลบรรทัดล่างของการแสดงผลเป็นบรรทัดล่างสุดของไฟล์ข้อมูล ก็จะไม่เกิดการเลื่อนบรรทัดการแสดงข้อมูล กรณีของการเลื่อนการแสดงผลข้อมูล ถ้าตำแหน่งของเคอร์เซอร์อยู่บรรทัดแรกของการแสดงเพื่อกด Ctrl-w แล้ว ตำแหน่งของเคอร์เซอร์จะอยู่คงที่ไม่เลื่อนไปแต่ตัวชี้ข้อมูลในไฟล์จะชี้เลื่อนลงมาบรรทัดล่างอีก 1 บรรทัดแทน
- Ctrl R (^R) หรือ PgDn เลื่อนการแสดงผลข้อมูลและตัวชี้ข้อมูลไป 1 หน้าจอ (ในที่นี้เลื่อนไป 23 บรรทัด)ลงด้านล่างซึ่งการเลื่อนนี้ จะตรวจสอบตำแหน่งของ

ตัวชี้ไฟล์ข้อมูลก่อน ถ้าตัวชี้ไม่ได้ชี้บรรทัดสุดท้ายของไฟล์ข้อมูลก็จะเกิดการเลื่อนข้อมูล และถ้าชี้อยู่บรรทัดสุดท้ายอยู่แล้วจะไม่เกิดการเลื่อนตำแหน่งตัวชี้และการแสดงผลจะไม่เปลี่ยนแปลง ในกรณีที่มีการเปลี่ยนแปลงตัวชี้ข้อมูลและเลื่อนตัวชี้ไป 23 บรรทัด โดยเลื่อนลงข้างล่าง ซึ่งถ้าเลื่อนลงมาถึงบรรทัดไฟล์ข้อมูลก่อนก็จะไม่มีการเลื่อนอีกต่อไป จากนั้นจะนำข้อมูลที่ตำแหน่งตัวชี้และตำแหน่งเคอร์เซอร์เดิมให้ตรงกัน และแสดงข้อมูลอื่นของบรรทัดที่แสดงบนจอภาพ

- Ctrl C (^C) หรือ PgUp เลื่อนการแสดงผลข้อมูลและตัวชี้ข้อมูล ขึ้นมาข้างบน 1 หน้าจอ (เลื่อนไป 23 บรรทัด) การเลื่อนนี้จะทำการตรวจสอบตัวชี้ไฟล์ข้อมูลก่อน ว่าชี้ที่บรรทัดแรกของข้อมูลหรือไม่ ถ้าไม่ใช่บรรทัดแรกก็จะเลื่อนตัวชี้ไฟล์ข้อมูลขึ้นมาบรรทัดบนไปจนครบ 23 บรรทัดหรือถึงบรรทัดแรกของไฟล์ข้อมูล ถ้าอยู่บรรทัดแรกแล้ว ก็จะไม่เกิดการเปลี่ยนแปลงการชี้ และการแสดงข้อมูลบนจอภาพ ในกรณีที่มีการเลื่อนบรรทัดของตัวชี้ข้อมูลแล้ว การแสดงผลข้อมูลจะเลื่อนตำแหน่งเคอร์เซอร์ ให้ตรงกับตำแหน่งของตัวชี้ไฟล์ข้อมูล แล้วแสดงข้อมูลที่อยู่ในช่วง 23 บรรทัดแสดงออกทางจอภาพ

-Ctrl Q+S (^Q^S) หรือ Home เลื่อนตำแหน่งตัวชี้ข้อมูลและตำแหน่งเคอร์เซอร์ไปที่ตำแหน่งคอลัมล์แรกของบรรทัดเดิม ซึ่งถ้าตัวชี้เดิมไม่ได้ชี้เกิน 80 คอลัมภ์แล้ว การแสดงข้อมูลบนหน้าจอจะไม่เปลี่ยนแปลง แต่ถ้าเกิน 80 คอลัมภ์แล้ว การแสดงผลจะเลื่อนการแสดงผลใหม่ โดยคอลัมภ์แรกของไพล์ข้อมูลจะแสดงตรงกับคอลัมภ์แรกของการแสดงผลบนจอแล้วแสดงไป 74 คอลัมภ์

- Ctrl Q+E (^Q^E) หรือ Ctrl Home เลื่อนตำแหน่งตัวชี้ข้อมูลและเคอร์เซอร์ไปที่คอลัมภ์สุดท้ายของไฟล์ข้อมูลของบรรทัดนั้น โดยการแสดงผลนั้นถ้าคอลัมภ์สุดท้ายของข้อมูลแสดงบนจอภาพแล้ว การแสดงผลของข้อมูลจะไม่เปลี่ยนแปลง แต่ถ้ายังไม่แสดงจะเกิดการเลื่อนการแสดงของหน้าจอข้อมูลโดยคอลัมภ์สุดท้ายที่ตัวชี้ข้อมูลชี้อยู่จะถูกนำออกแสดงที่คอลัมภ์ที่ 78 และข้อมูลอื่นๆจะเลื่อนแสดงทางด้านซ้ายถัดกันมาเรื่อยๆจนถึงคอลัมภ์ที่ 1 ของจอภาพ

- Ctrl Q+E หรือ Ctrl-Home เป็นการเลื่อนตำแหน่งของเคอร์เซอร์ไปที่บรรทัดแรกของการแสดงผลข้อมูลโดยคอลัมภ์คงที่พร้อมกับเลื่อนตำแหน่งของตัวชี้ตามไปด้วย

- Ctrl Q+X หรือ Ctrl End เป็นการเลื่อนตัวชี้ตำแหน่งเคอร์เซอร์ไปที่บรรทัดสุดท้ายของการแสดงผลข้อมูลพร้อมกับเลื่อนตัวชี้ข้อมูลของไฟล์ตามไปดวัย โดยที่ตำ

สุดท้ายของการแสดงผลข้อมูลพร้อมกับเลื่อนตัวชี้ข้อมูลของไฟล์ตามไปดวัย โดยที่ตำแหน่งคอลัมภ์ยังคงที่เดิม

- Ctrl Q+C หรือ Ctrl PgDn เป็นการเลื่อนตัวชี้ข้อมูลของไฟล์ไปที่บรรทัดสุดท้ายและคอลัมภ์สุดท้ายของไฟล์ดวัย และแสดงข้อมูลออกจอภาพโดยตำแหน่งเคอร์เซอร์จะอยู่ตรงกับตัวชี้ตำแหน่งของไฟล์ข้อมูล
- Ctrl Q+R หรือ Ctrl PgUp เลื่อนตัวชี้ข้อมูลของไฟล์ไปที่ตำแหน่งคอลัมภ์แรกบรรทัดแรกของไฟล์ข้อมูล และตำแหน่งเดอร์เซอร์อยู่ที่คอลัมภ์แรกและบรรทัด และแสดงข้อมูลออกจอภาพ ตั้งแต่บรรทัดแรกไป 23 บรรทัด ก่อนคีย์ควบคุมการแทรกและลบ
- Ctrl V หรือ Ins เป็นการเลือก MODE ของการรับข้อมูลที่ปอ้นเข้ามาโดยคีย์ที่กด จะแสดงให้ทราบข้างบนของเอดิเตอร์ ซึ่งจะแสดงใน 2 Mode คือ Insert Indent และ Over write Indent ซึ่งจะแสดงสลับกันจากการกดคีย์เป็นแบบ Toggle ในโหมด Insert Indent จะทำงานในการแทรกข้อมูลที่รับเข้ามา ส่วนโหมด Overwrite Indent จะเป็นการเขียนทับข้อมูลคำเก่า จากข้อมูลตัวเก่าที่รับเข้ามา
- Ctrl N แทรกบรรทัดข้อมูลที่อยู่ถัดลงมาข้างล่างของตัวชี้ข้อมูล 1 บรรทัดพร้อมกับนำข้อมูลที่อยู่หลังตัวชี้ข้อมูลของไฟล์ของบรรทัดที่ชี้อยู่ ไปใส่ไว้ในที่ตำปหน่งคอลัมภ์แรกของบรรทัดที่แสดงขึ้นมา แล้วเลื่อนตำแหน่งตัวชี้ไปยังคอลัมภ์แรกของบรรทัดที่แทรกขึ้นมา การแสดงผลบนหน้าจอจะแสดงบรรทัดที่มีเดอร์เซอร์อยู่ใหม่ และการแทรกบรรทัดแรกของข้อมูลที่แสดงออกมา พรอ้มกับเลื่อนตำแหน่งเดอร์เซอร์ไปคอลัมภ์แรกของบรรทัดถัดมา (บรรทัดที่แทรก)
- Ctrl-Y ลบบรรทัดที่ตำแหน่งตัวชี้ข้อมูลออกจากไฟล์ข้อมูลแล้วเลื่อนตัวชี้ข้อมูลของไฟล์ไปยังบรรทัดบนถัดไปของไฟล์ข้อมูลที่ตำแหน่งคอลัมภ์แรก การแสดงผลบนจอภาพนั้น จะลบบรรทัดที่มีเดอร์เซอร์ออก แล้วนำบรรทัดข้างล่างต่อๆมาแสดงแทน จากนั้นจะเลื่อนตำแหน่งเดอร์เซอร์ไปยังบรรทัดบนและคอลัมภ์แรกของบรรทัดนั้น ในกรณีที่ตำแหน่งของเดอร์เซอร์อยู่ที่บรรทัดแรกของการแสดงผล และเลื่อนเดอร์เซอร์ไปที่คอลัมภ์แรกของข้อมูล บรรทัดที่ตัวชี้ชี้อยู่ และถ้าเติมตัวชี้บรรทัดข้อมูลนั้นอยู่บรรทัดของไฟล์ข้อมูลแล้ว เมื่อลบบรรทัดนี้ออกแล้ว ตัวชี้ข้อมูลจะไปชี้ที่บรรทัดล่างถัดลงมาและจัดให้เป็นบรรทัดแรกของไฟล์ข้อมูลต่อไป
- Ctrl G หรือ Del ข้อมูล 1 ตัวที่ตำแหน่งที่ตัวชี้อยู่ของไฟล์ข้อมูล โดยเลื่อนอัก

ขระข้อมูลที่อยู่ทางขวาของตัวชี้ตำแหน่งข้อมูลของบรรทัดที่ชี้อยู่ เลื่อนเข้ามา 1 ตัวอักษร ในการแสดงผลนั้น จะแสดงข้อมูลที่ตำแหน่งที่มีเดอร์เซอร์อยู่ใหม่ตามข้อมูลในไฟล์ข้อมูล

- Ctrl T ลบข้อมูลที่อยู่ทางขวาของตำแหน่งตัวชี้ออก 1 คำโดยถ้าตัวชี้อยู่ระหว่างคำนั้นอักขระที่ตั้งอยู่ทางขวาของอักขระคำนี้จะถูกลบออกจากไฟล์ข้อมูลหรือถ้าไม่มีการลบคำทางขวาของบรรทัดนั้น ก็จะดึงข้อมูลของบรรทัดถัดลงมาข้างล่าง มาลบข้อมูลออกมา 1 คำ โดยดึงข้อมูลบรรทัดถัดมา มาใส่ต่อกับบรรทัดที่มีตัวชี้อยู่ แล้วบรรทัดข้อมูลข้างหลังที่ตามมาก็จะถูกลบออกจากไฟล์ข้อมูล การลบคำนี้จะถูกกระทำโดยทำข้อมูลที่ตำแหน่งคำถัดไปมาใส่ ณ.ที่ตำแหน่งที่อยู่ถัดมาจากตำแหน่งที่ตัวชี้ ชี้อยู่ทางขวา การแสดงผลบนจอภาพจะแสดงผลของข้อมูลบรรทัดที่ที่ตำแหน่งเดอร์เซอร์อยู่ใหม่ การนำบรรทัดข้อมูงข้างล่างมาลบต่อดวัยก็จะแสดงข้อมูลบรรทัดที่อยู่ถัดมาใหม่

<u>คีย์เกี่ยวกับกลุ่มข้อมูล</u>

- Ctrl K+B ๆ เป็นการกำหนดจุดเริ่มต้นของการกำหนดกลุ่มข้อมูลโดยปกติเมื่อเริ่มต้นเข้าสู่เอดิเตอร์แล้ว จุดเริ่มต้นของการกำหนดกลุ่มข้อมูล จะถูกกำหนดไว้ที่บรรทัดแรกและคอลัมภ์แรกของไฟล์ข้อมูล การกำหนดจุดเริ่มต้นใหม่นี้ จะถูกกำหนด ณ.ที่ตัวชี้ตำแหน่งข้อมูลของไฟล์ชี้อยู่ เมื่อกำหนดจุดเริ่มต้นของกลุ่มข้อมูลแล้ว ก็จะตรวจสอบการกำหนดจุดสุดท้ายที่กำหนดของกลุ่มข้อมูล ถ้าจุดสุดท้ายของกลุ่มข้อมูลเกิดขึ้นที่ตำแหน่งหลังจุดกำหนดเริ่มต้นกลุ่มข้อมูล ก็จะเกิดเป็นการกำหนดกลุ่มข้อมูลขึ้น โดยบนจอภาพจะแสดงการเปลี่ยนสีของข้อมูลที่แสดงหรือความเข้มของแสงในจอแสดงผลแบบโมโนโครม พร้อมกับแสดง "Block" บนส่วนหัวของเอดิเตอร์

-Ctrl K+K หรือ F เป็นการกำหนดจุดสุดท้ายของกลุ่มข้อมูล โดยกำหนดไว้ ณ.ที่จุดที่ตำแหน่งตัวชี้ตำแหน่งของไฟล์ข้อมูลในปัจจุบัน ปกติเมื่อเข้าสู่เอดิเตอร์ตอนแรกของไฟล์ข้อมูล จะกำหนดกลุ่มสุดท้ายของกลุ่มข้อมูลไปไว้ที่ตำแหน่งคอลัมภ์แรกบรรทัดแรกของไฟล์ข้อมูล เมื่อกำหนดจุดสุดท้ายของกลุ่มข้อมูลแล้ว จะตรวจสอบตำแหน่งของตนเองว่าถูกกำหนดให้อยู่มากกว่าที่ตำแหน่งกำหนดจุดเริ่มต้นของกลุ่มข้อมูลหรือไม่ ถ้าถูกกำหนดให้มากกว่าก็จะเป็นการกำหนดกลุ่มข้อมูลที่สมบูรณ์ซึ่งบนจอภาพ จะแสดงโดยการเปลี่ยนของการแสดงข้อมูล (ปรับ Attribute) ในจอสีหรือเปลี่ยนความเข้มของแสงในการแสดงผลบนจอโมโนโครม และแสดง

"Block"บนส่วนหัวของเอดิเตอร์ ถ้าการกำหนดกลุ่มข้อมูลไม่สมบูรณ์จะไม่เกิดการแสดงเปลี่ยนแปลงเพิ่มขึ้น

-Ctrl K+Y เป็นการลบกลุ่มข้อมูลที่เกิดขึ้นโดยถ้ามีการกำหนดกลุ่มข้อมูลที่สมบูรณ์เกิดขึ้นแล้ว เมื่อกดคีย์ควบคุมนี้ กลุ่มข้อมูลที่ถูกกำหนดไว้ ก็จะถูกลบออกจากไฟล์ข้อมูลซึ่งถ้าไม่มีการกำหนดกลุ่มข้อมูลขึ้น ก็จะไม่เกิดการเปลี่ยนแปลงขึ้นในการแสดงผลเมื่อมีการกำหนดกลุ่มข้อมูลนั้น ถ้ามีกลุ่มข้อมูลที่ถูกกำหนดแสดงบนจอภาพขึ้น ก็จะลบข้อมูลกลุ่มนี้ออกจากจอภาพ และนำข้อมูลบรรทัดถัดมาจากกลุ่มข้อมูลมาแทน

- Ctrl K+C เป็นการคัดลอกกลุ่มข้อมูลที่กำหนดไว้ ไปไว้ ณ.ที่ตำแหน่งตัวชี้ข้อมูลชี้อยู่ ซึ่งก่อนเกิดการคัดลอกข้อมูลจะมีการตรวจสอบว่ามีการกำหนดกลุ่มข้อมูลแล้วหรือยังและตำแหน่งตัวชี้นั้นชี้อยู่นอกกลุ่มข้อมูลนั้นหรือไม่ ถ้ามีการกำหนดกลุ่มข้อมูล และตำแหน่งตัวชี้อยู่นอกกลุ่มข้อมูลที่กำหนด ก็จะเกิดการคัดลอกกลุ่มข้อมูลไปไว้ ณ.ที่ตำแหน่งตัวชี้ชี้อยู่ปัจจุบัน ในการแสดงผลนั้น จะเริ่มแสดงผลข้อมูลที่อยู่ถัดมาจากตำแหน่งเดอร์เซอร์ลงมาข้างล่างใหม่ โดยจะแสดงข้อมูลที่คัดลอกเข้ามาแสดงแทน พร้อมกับแสดงว่าเป็นตำแหน่งของกลุ่มข้อมูลใหม่ที่กำหนด (แสดงโดย Bit attribute ของการแสดงผล)

- Ctrl K+V เป็นการย้ายกลุ่มข้อมูลที่กำหนดไปไว้ ณ.ที่ตำแหน่งตัวชี้ ข้อมูลชี้อยู่ในปัจจุบัน ซึ่งการย้ายนั้นจะตรวจสอบว่ามีการกำหนดกลุ่มข้อมูลแล้วหรือยังและตำแหน่งตัวชี้ปัจจุบันนั้นจะอยู่นอกกลุ่มข้อมูลหรือไม่ ถ้ามีการกำหนดกลุ่มข้อมูล และตัวชี้ข้อมูลอยู่ข้างนอกกลุ่มข้อมูล ก็จะเกิดการย้ายกลุ่มข้อมูลขึ้น โดยในการแสดงผลนั้นจัดการแทรกการแสดงผลที่อยู่ถัดจากเดอร์เซอร์ข้างล่างใหม่โดยการแสดงข้อมูลใหม่ที่ย้ายมา และถ้ามีกลุ่มข้อมูลที่กำหนด แสดงบนจอภาพด้วยก็จะลบข้อมูลที่กำหนดส่วนที่กำหนดแสดงบนจอภาพด้วยก็จะลบข้อมูลที่แสดงบนจอภาพออก

-Ctrl K+R เป็นการอ่านข้อมูลไฟล์อื่นๆจากดิสเกตต์มาแทรกในไฟล์ข้อมูล ของเอดิเตอร์ ณ.ที่ตำแหน่งตัวชี้ลงมาข้างล่าง พร้อมกับกำหนดเป็นกลุ่มข้อมูลใหม่ ในการแสดงผลนั้นจะเริ่มแสดงผลของไฟล์ข้อมูลใหม่ในตำแหน่งหลังจากเดอร์เซอร์ลงมา

-Ctrl K+W เป็นการเขียนกลุ่มข้อมูลลงสู่ดิสเก็ตต์ โดยก่อนการเขียนนั้น จะทำการตรวจสอบว่ามีการกำหนดกลุ่มข้อมูลไว้ก่อนหรือไม่ ที่มีการกำหนดก็จะเกิดการเขียนขึ้นถ้าไม่มีก็จะไม่เกิดอะไรขึ้น สำหรับการเขียนนั้นจะทำการตรวจสอบชื่อไฟล์ที่มีอยู่ในดิสเก็ตต์ ว่ามีชื่อกำหนดอยู่หรือไม่ ถ้าไม่มีก็เขียนลงไป ถ้ามีจะถามผู้ใช้ก่อนว่าจะให้เขียนทับลงไปหรือไม

สำหรับคีย์ Ctrl K+R และ Ctrl K+W นั้นสามารถหยุดการทำงานได้ในช่องที่เอดิเตอร์ถามชื่อไฟล์ โดยผู้ที่กด Key Ctrl U

- Ctrl K+D หรือ F10 เป็นการออกจากการใช้งานเอดิเตอร์

<u>คีย์พิเศษอื่นๆ</u>

- TAB เป็นการเลื่อนข้อมูลตั้งแต่ตำแหน่งที่ตัวชี้ชี้ข้อมูลอยู่ ไปทางขวาโดยตำแหน่งที่เลื่อนข้อมูลไปจะเลื่อนไปให้ตรงกับตำแหน่งคอลัมภ์แรกของคำของข้อมูลบรรทัดบนของคำถัดไป ในกรณีที่ไม่มีบรรทัดบนหรือไม่มีคำถัดไปของข้อมูลบรรทัดบน การเลื่อนข้อมูลก็จะเลื่อนไปทางขวาทีละ 8 คอลัมภ์ ซึ่งการทำงานทั้งหมดก่อนจะเลื่อนข้อมูล จะตรวจสอบพื้นที่ที่เหลือของแต่ละบรรทัดว่าเมื่อเลื่อนไปแล้วจะมีพื้นที่พอที่จะเลื่อนข้อมูลออกไปได้หมด หรือไม่ ซึ่งถ้าพอก็จะเกิดการเลื่อน ถ้าไม่พอก็จะไม่เกิดการกระทำใดๆเกิดขึ้น

- SHIFT + TAB เป็นการเลื่อนตัวชี้ข้อมูลและข้อมูลที่ไม่อยู่ทางขวาของตัวชี้ ให้ชี้เลื่อนมาทางด้านซ้าย โดยให้ตรงกับคอลัมภ์แรกของคำแรกทางซ้ายของข้อมูลบรรทัดบน ซึ่งถ้าไม่มีข้อมูลบรรทัดบน หรือไม่มีคำทางซ้ายมือของบรรทัดบน การเลื่อนข้อมูลก็จะเลื่อนไปทางด้านซ้าย 8 คอลัมภ์แทน และก่อนการเลื่อนจะมีการตรวจสอบก่อนว่าจะเลื่อนไปทางด้านซ้าย 8 คอลัมภ์นั้น พื้นที่ทางด้านซ้ายมีพอหรือมากกว่า 8 หรือไม่ ถ้าพอก็จะเกิดการเลื่อน ถ้าไม่พอก็จะไม่เกิดการเลื่อน

- Back space เป็นการเลื่อนข้อมูลและตัวชี้ ไปทางซ้าย 1 คอลัมภ์ ซึ่งถ้าตัวชี้อยู่ที่คอลัมภ์ที่ 1 จะไม่เกิดการเลื่อนขึ้น การเลื่อนข้อมูลไปทางซ้ายนี้ ถ้าตัวที่อยู่ทางด้านซ้ายของตัวชี้ที่ติดกันก็จะถูกเขียนทับลงไปด้วยข้อมูล ณ.ตำแหน่งตัวชี้เดิม.

โปรแกรมส่วนพิเศษอื่นๆ

โปรแกรมพิเศษที่เพิ่มเข้ามาในโครงงานเพื่อความสะดวกในการใช้งานมีดังนี้

* ฟังก์ชั่นดอส (Dos function) ในโครงงานฟังก์ชั่นดอสทำเฉพาะฟังก์ชั่นดอสที่ใช้งานบ่อยๆ ที่เขียนในโรงงานคือ
  - Copy file ฟังก์ชั่นนี้จะทำการก็อปปี้ข้อมูลจากไฟล์หนึ่งไปยังอีกไฟล์หนึ่ง
  - Log drive ฟังก์ชั่นนี้จะทำการเปลี่ยนการติดต่อไปยังไดร์ฟที่กำหนด
  - Rename ฟังก์ชั่นนี้จะทำการเปลี่ยนชื่อไฟล์จากชื่อเดิมเป็นชื่อใหม่
  - Erase ฟังก์ชั่นนี้จะทำการลบไฟล์บนดิสค์
  - Directory ฟังก์ชั่นนี้จะทำการแสดงชื่อไฟล์ที่มีอยู่บนดิสค์ และเนื้อที่ว่าง
* ฟังก์ชั่นหาข้อผิดพลาด (Find error) ฟังก์ชั่นนี้จะนำไปสู่บรรทัดของข้อมูลที่ผิดพลาดเนื่องจากการแอสเซมเบลอร์ โดยเข้าสู่อีดิเตอร์
* ฟังก์ชั่นคาร์ดดาวโหลด (Gard Download) ฟังก์ชั่นนี้มีฟังก์ชั่นย่อยๆดังนี้
  - Help Use Card เป็นรายละเอียดเกี่ยวกับคาร์ดและฟังก์ชั่นอื่นๆที่ใช้
  - Load Data file ฟังก์ชั่นนี้จะเอาข้อมูลจากไฟล์มาเก็บในหน่วยความจำเพื่อเตรียมโหลดลงคาร์ดดาวโหลด
  - Move file to Card ฟังก์ชั่นนี้จะนำข้อมูลโหลดลงคาร์ดดาวโหลด
  - Dump memory Card ฟังก์ชั่นนี้จะแสดงข้อมูลที่มีอยู่บนคาร์ดดาวโหลด
  - Show Data file ฟังก์ชั่นนี้จะแสดงไฟล์ข้อมูลที่มีอยู่ในหน่วยความจำ
  - Test Memory Card ฟังก์ชั่นนี้จะใช้ในการพัฒนาหรือทดสอบคาร์ดดาวโหลด

โปรแกรมส่วนพิเศษต่างๆ เหล่านี้เขียนด้วยภาษา PASCAL ทั้งหมด

การนำไปใช้งานและการประยุกต์ใช้งาน

การใช้งานของ CROSS ASSEMBLER

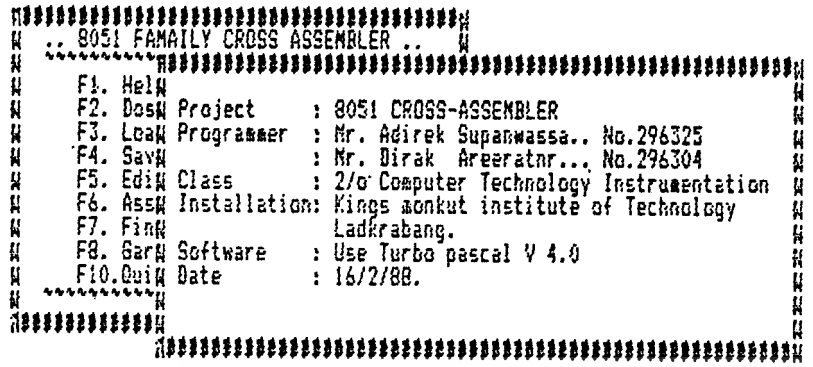
ในโครงงานนี้ได้สร้างโปรแกรมสำหรับใช้งาน 2 ชุดคือ ไฟล์ชื่อ C51.EXE และ MC51.EXE โดยโปรแกรมที่เขียนขึ้นทั้งหมดนี้ใช้ภาษา PASCAL เขียนทั้งหมด โดยใช้ TURBO PASCAL ของบริษัท BORLAND VERSION 4.0 ซึ่งเป็น VERSION ล่าสุดในขณะนี้

รายละเอียดของไฟล์มีดังต่อไปนี้

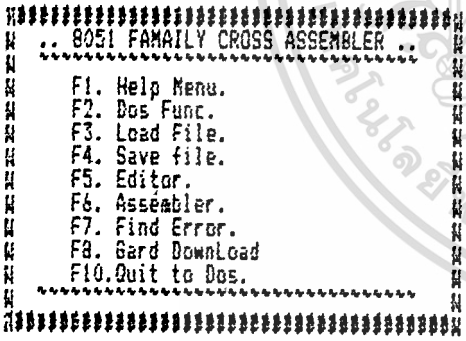
1. ไฟล์ C51.EXE มีฟังก์ชั่นการใช้งานต่างๆดังต่อไปนี้
   - Help เป็นส่วนอธิบายหน้าที่ของฟังก์ชั่นคีย์ต่างๆ
   - Function Dos เป็นการทำงานที่เกี่ยวกับคำสั่งในดอส
   - Load file เป็นการอ่านข้อมูลซึ่งเป็น Text file ลงสู่หน่วยความจำในกรณีที่มีไฟล์อยู่ ถ้าไม่มีไฟล์จะแจ้งว่าเป็นไฟล์ใหม่
   - Save file เป็นการบันทึกข้อมูลที่มีอยู่ของ Text file ลงสู่ฟลอปปี้ดิสค์ โดยจะสำรองไฟล์ข้อมูลให้ด้วย (ชื่อไฟล์.BAK)
   - Editor เป็นการเข้าสู่โหมดการทำงานของอีดิเตอร์เพื่อแก้ไขข้อมูล
   - Assembler จะทำการแปลข้อมูลภาษา Assembly เป็น object code
   - Find Error เป็นการค้นหาตำแหน่งข้อผิดพลาดเนื่องจากข้อผิดพลาดของภาษา ทำให้แปลเป็น object code ไม่ได้
   - Gard Download เป็นการโหลด object code ไปยัง Card
   - Quit to dos ออกจากการทำงานของโปรแกรมสู่ดอส

   สำหรับรายละเอียดของแต่ละฟังก์ชั่นได้แสดงไว้ดังรูป หลังหัวข้อนี้

2. ไฟล์ MC51.EXE เป็นไฟล์ที่มีการทำงานเฉพาะส่วนของ Assembler เพียงอย่างเดียว ทั้งนี้เพื่อความสะดวกของผู้ใช้ในกรณีที่มีไฟล์ภาษา Assembly อยู่แล้วบนดิสค์



รูป MENU แรกของไฟล์ C51.EXE



รูป แสดง MAIN MENU

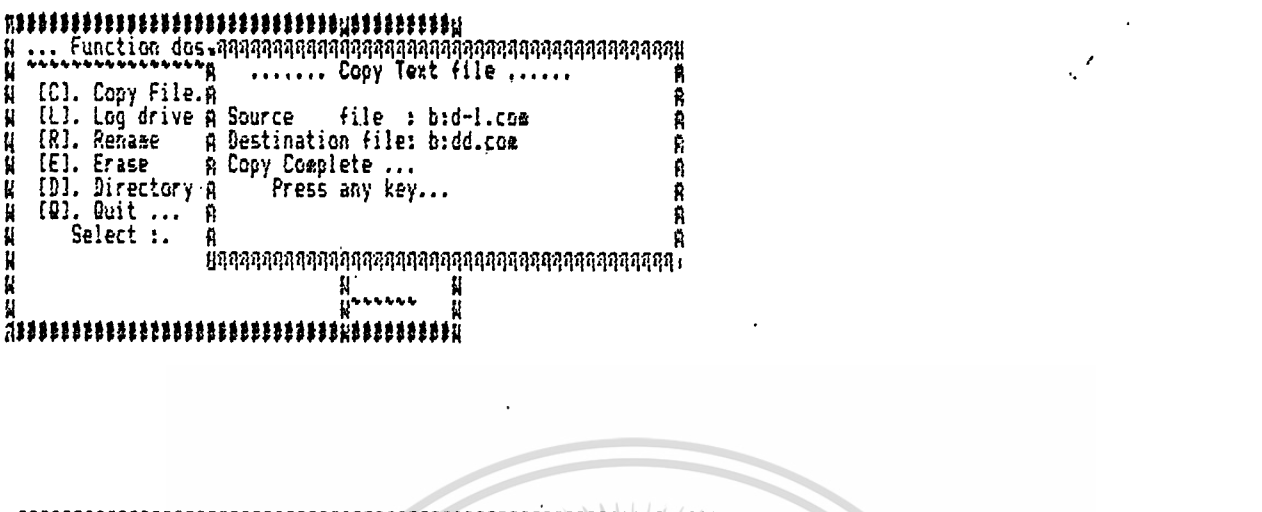
```
  .. 80
  *****    -----*** DESCRIPTION OF CROSS ASSEMBLER 8051 ***-----
     F1       ------***** FUNCTION [F1] *****-------
     F2
     F3         This is program cross-assembler 8051 .It can be
     F4  compiled program assembly language 8051 to be object code
     F5  of 8051. This object code will loaded to single-board 8051
     F6  and run program on single-board 8051.
     F7
     F8 [F2]  This key is used command of dos such as [C]opy etc.
     F1 [F3]  This key is used load file from disk to memory buffer.
  ***** [F4]  This key is used save file on memory buffer to disk.
        [F5]  This key is used initial into editor for update data.
        [F6]  This key is used compile assembly 8051 .This output is
              selected option (*.lst),(*.hex),(*.bin).
        [F7]  This key is used find error instruction by into editor.

        [F8]  This key is used download object code to single-board.
        [F10] This key is used quit out from program.

                           Press key ESC to quit...
- Mov                                                        n key
```

รูป เมื่อกด F1 หรือ H จะแสดง HELP

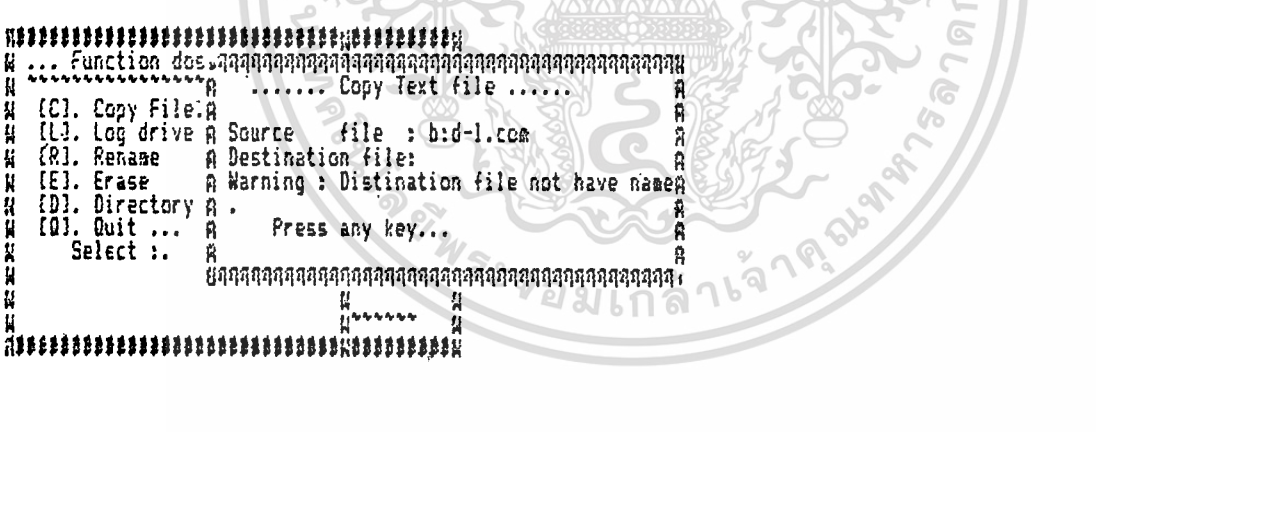
```
 ... Function dos ...          LER ..
 ********************          ******

  [C]. Copy File.
  [L]. Log drive
  [R]. Rename
  [E]. Erase
  [D]. Directory
  [Q]. Quit ...
     Select :.

                               ******
```

```
- Move bar. Select by Pressing a Higthlighed Leter , <- , or Function key
```

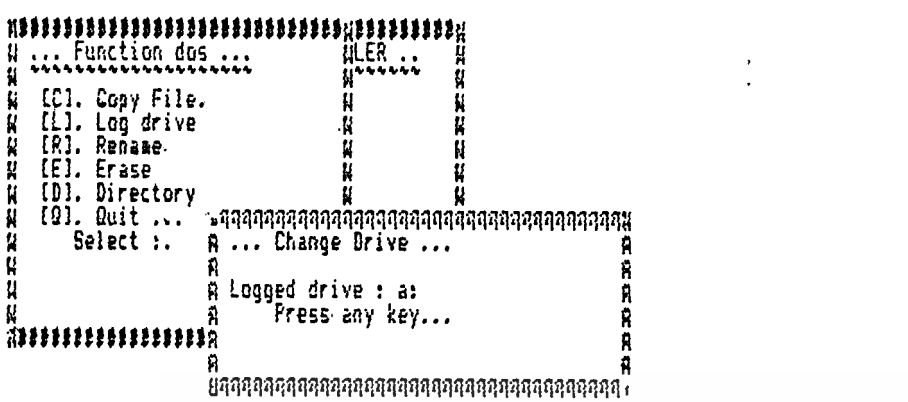
รูป เมื่อกด F2 หรือ D แสดง MENU ของ Function dos



รูป เมื่อกด C จะเป็นฟังก์ชั่น Copy file



รูป เมื่อเกิดข้อผิดพลาดใน Function copy file



รูป เมื่อกด L เพื่อเปลี่ยน Logged drive

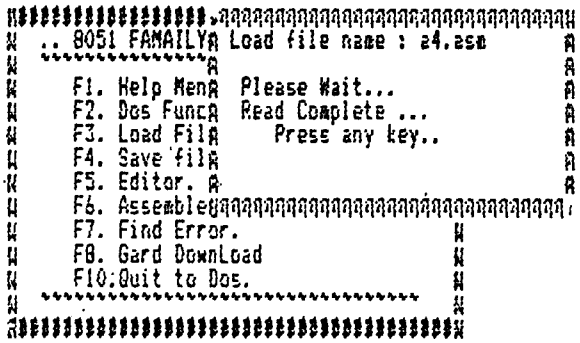
รูป เมื่อกด R เพื่อเปลี่ยนชื่อไฟล์

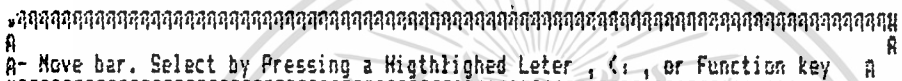
```
IMMMMMMMMMMMMMMMMMMMMMMMMMMMMM;MMMMMMMM;
: ... Function dos ...        :LER ..   :
: ^^^^^^^^^^^^^^^^^^^^        :^^^^^^   :
:  [C]. Copy File.            :         :
:  [L]. Log drive             :         :
:  [R]. Rename                :         :
:  [E]. Erase                 :         :
:  [D]. Directory             :         :
:  [Q]. Quit ...              :         :
:     Select :.               :         :
:                             :         :
:                             :         :
:                 ZDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD?
HMMMMMMMMMMMMMMMMM3    ... ERASE FILE ...              3
                  3                                    3
                  3     Want Delete file : AA.A        3
                  3... Erase file [ AA.A ] complete ...  3
                  3     Press any key...               3
                  3                                    3
                  3                                    3
                  @DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDY
ZDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD?
3                                                                             3
- Move bar. Select by Pressing a Higthlighed Leter , <Y , or Function key    3
@DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDY
```

รูป เมื่อกด E จะลบไฟล์บนดิสค์

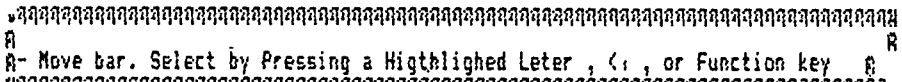
รูป เมื่อกด D จะแสดง Directory

```
IMMMMMMMMMMMMMMMMMMMMMMMMMMMMM;MMMMMMMM;
: ... Function dos ...        :LER ..   :
: ^^^^^^^^^^^^^^^^^^^^        :^^^^^^   :
:  [C]. Copy File.            :         :
:  [L]. Log drive             :         :
:  [R]. Rename                :         :
:  [E]. Erase                 :         :
:  [D]. Directory             :         :
:  [Q]. Quit ...              :         :
:     Select :.               :         :
ZDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD?
3            ... DIRECTORY FILE ...                                          3
3                                                                            3
3  Directory mask : B:\                                                      3
3   DLOAD   .TPU    FUNDOS  .TPU    C51     .EXE    EDITER1 .TPU             3
3   FUNDOS  .PAS    MC51    .EXE    C51     .PAS    WINCRT  .PAS             3
3   WINCRT  .TPU    VARA    .TPU    A1      .BIN    MCS51   .TPU             3
3                                                                            3
3                                                                            3
3        Total file = 12 file(s).        1024 bytes free.                    3
3                                                                            3
3                                                                            3
3                                                                            3
3                                                 Press any key...           3
@DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDY
```

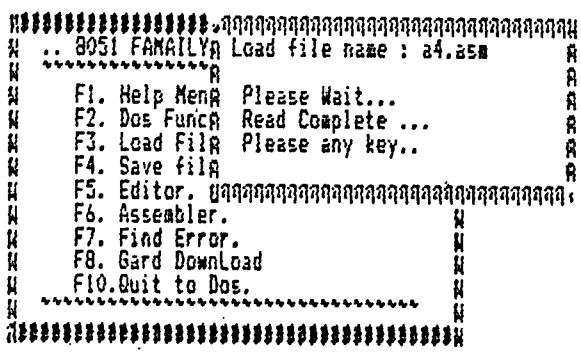




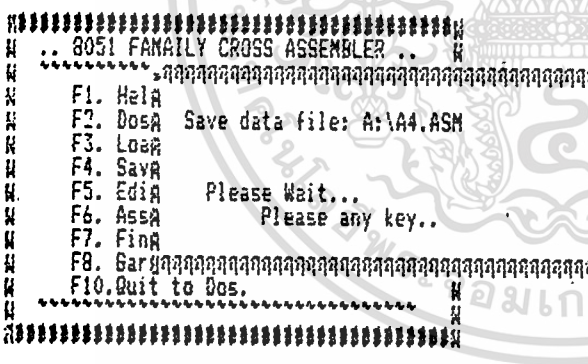
รูป เมื่อกด F3 หรือ L จะโหลดไฟล์มาเก็บไว้

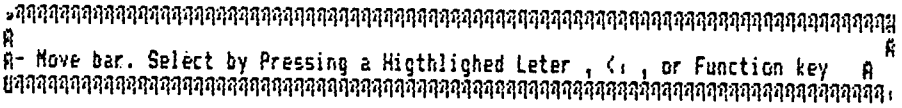


รูป แสดงว่าเป็นไฟล์ใหม่



รูป เมื่อโหลดไฟล์มาเก็บไว้เรียบร้อยแล้ว





รูป เมื่อกด F4 หรือ S จะ Save file ลงดิสค์

```
   Line     1  Col    1  INSERT    Indent  Drive    File A:\A4.ASM
;PROGRAM 8031 display  8 0 3 1 * * * ( * = ..... ) for initial
;And read scan key board input for user press
;display contunue (Not delay time )
              org      0100h
;Set data initial for display
              mov      r4,#080h      ; buffer initial digit 0 display '8'
              mov      r5,#0c0h      ; buffer initial digit 0 display '0'
              mov      r6,#0b0h      ; buffer initial digit 0 display '3'
              mov      r7,#0f9h      ; buffer initial digit 0 display '1'
   start:     setb     p3.3          ; conect io/m of 8155
                                     ; select  io port
              mov dptr,#0ff00h       ; no use in address of down-load card
              mov      a,#01h        ; Pa:o/p Pb,Pc:i/p timer:stop
              clr      p3.1          ; conect /cs of 8155
              movx     @dptr,a
              inc      dptr
   loop:      mov      a,r4          ;digit 0
              movx     @dptr,a
              anl      p1,#0f0h
              acall    Readkey0
              setb     p1.0          ;digit 1
              mov      a,r5
              movx     @dptr,a
              acall    Readkey1
```

รูป เมื่อกด F5 หรือ E จะเข้าสู่ Editor

```
#################################################################################
#       -------- ****** DISCRIPTION OF KEY IN EDITTER ****** --------           #
#  F1.     : HELP.                          F2.    : LINE UP FOR ERROR          #
#  F3.     : LINE DOWN FOR ERROR            F4,^K^W. : WRITE BLOCK FILE         #
#F5,^K^R. : READ BLOCK FILE                F6,^K^B. : MARK START BLOCK          #
#F7,^K^K. : MARK END BLOCK FILE            F8,^K^C. : COPY BLOCK DATA TO CUSSOR #
#F9,^K^V. : MOVE BLOCK DATA TO CUSSOR      F10,^K^D.: END EDITER.               #
#          --------- ****** FUNCTION MOVE CUSSOR IN EDITER ****** --------      #
#^S,LArr.  : CHARACTER LEFT                 ^D,RArr.  : CHARACTER RIGHT         #
#^E,UArr.  : LINE UP                        ^X,DArr.  : LINE DOWN               #
#^A.       : WORD LEFT                      ^F.       : WORD RIGHT              #
#^W.       : SCROLL UP                      ^Z.       : SCROLL DOWN             #
#^R,PgUp.  : PAGE UP                        ^C,PgDn.  : PAGE DOWN               #
#^Q^S,Home.: LEFT ON LINE                   ^Q^D,End. : RIGHT ON LINE           #
#^Q^E,^Home: TOP OF PAG                     ^Q^X,^End.: BOTTOM OF PAGE          #
#^Q^R,^PgUp: TOP OF FILE                    ^Q^C,^PgDn: END OF FILE             #
#          ---------- ****** FUNCTION INSERT & DELETE ****** ----------         #
#^N        : INSERT LINE                    ^Y        : DELETE LINE             #
#^T        : DELETE RIGHT                   ^G,Del    : DELETE CHARACTER UNDER CURSO#
#^V,INS    : INSERT MODE ON/OF              ^Q^Y      : DELETE MARK BLOCK       #
#^I,Tab    : TAB OR 8 CHARS RIGHT           Shift Tab : TAB OR 8 CHARS LEFT     #
#                                                                               #
#                                                    Please key ESC             #
#################################################################################
              acall    Readkey1
```

รูป เมื่อกด F1 ใน Editor จะแสดง Help คีย์ต่างๆ

```
.. 8051
    CROSS ASSAMBLER MCS-51 FAMAILY
F1
F2      OPTION USE BUILD OUTPUT FILE.
F3      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
F4  1.) ASSEMBLER ONLY NO HAVE OUTPUT FILE.
F5  2.) BUILD FILE   *.BIN ONLY.
F6  3.) BUILD FILE   *.HEX ONLY.
F7  4.) BUILD FILE   *.BIN AND  *.LST.
F8  5.) BUILD FILE   *.HEX AND  *.LST.
F1  6.) BUILD FILE   *.HEX , *.BIN  AND *.LST.
~~~~~~
          Option :.
```

```
* Move bar. Select by Pressing a Higthlighed Leter , <; , or Function key  *
```

รูป เมื่อกด F6 หรือ A จะเข้าสู่การ ASSEMBLER

```
.. 8051 FAMAILY CROSS ASSEMBLER ..
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   F1. Help Menu.
   F2. Dos Func.
   F3. Load File.
   F4. Save file.    8051 ASSAMBLER FILE : B:\A4.A31
   F5. Editor.
   F6. Assembler.          PASS 1. ASSAMBLING
   F7. Find Error.
   F8. Gard DownLoad             Line   134
   F10.Quit to Dos.
~~~~~~~~~~~~~~~~~~~~
```

```
* Move bar. Select by Pressing a Higthlighed Leter , <; , or Function key  *
```

รูป แสดงการ ASSEMBLER PASS 1

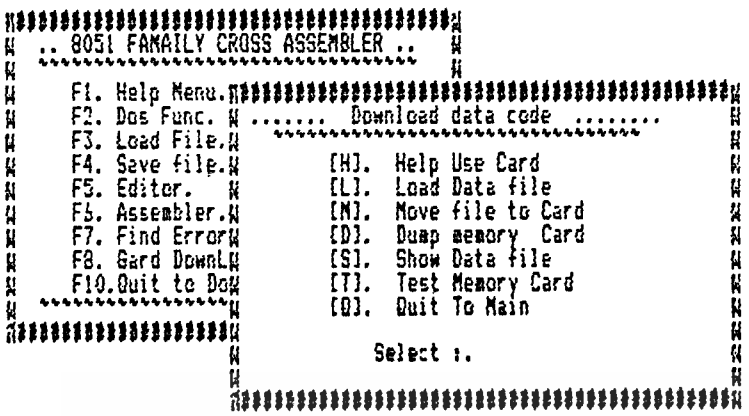
```
.. 8051 FAMAILY CROSS ASSEMBLER ..
   F1. Help Menu.
   F2. Dos Func.
   F3. Load File.
   F4. Save file.
   F5. Editor.
   F6. Assembler.
   F7. Find Error.
   F8. Gard DownLoad
   F10.Quit to Dos.

        8051 ASSAMBLER FILE : B:\A4.A31

              PASS 2. ASSAMBLING

                   Line    95
```

```
A- Move bar. Select by Pressing a Higthlighed Leter , <↵ , or Function key   A
```

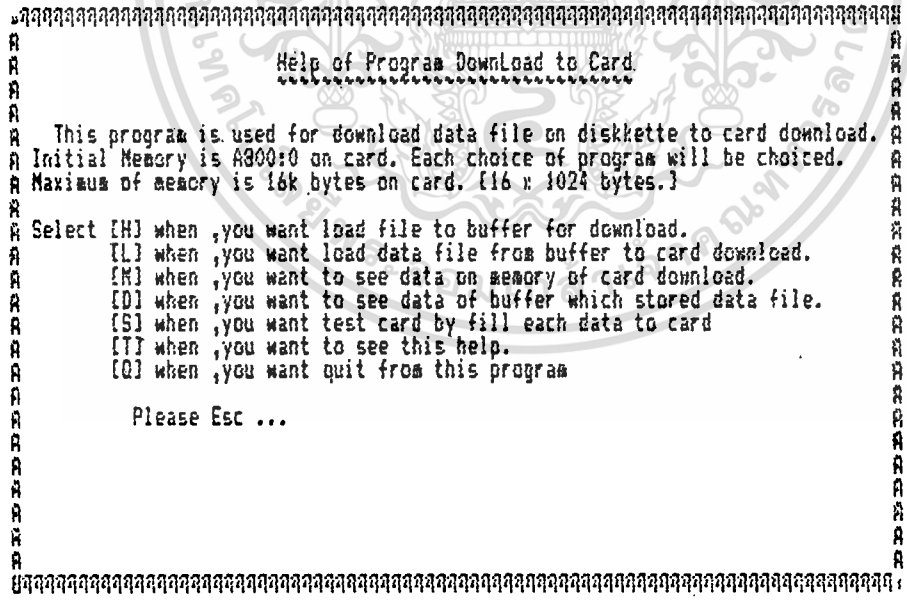
รูป แสดงการ ASSEMBLER PASS 2

```
A  140  020A  78 FF          delay1:   mov     r0,#0ffh      Press any key...
A  141  020C  D8 FE                    djnz    r0,$     A
A  142  020E  D9 FA                    djnz    r1,delay
A  143  0210  22             exit:     ret
A  144  0211  EE           shift_buff: mov     a,r6
A  145  0212  FF                       mov     r7,a
A  146  0213  ED                       mov     a,r5
A  147  0214  FE                       mov     r6,a
A  148  0215  EC                       mov     a,r4
A  149  0216  FD                       mov     r5,a
A  150  0217  E3                       mov     a,r0
A  151  0218  FC                       mov     r4,a
A  152  0219  22                       ret
A  153                                 end.
A
A
Instruction error  0 line
```

รูป แสดงผลหลัง ASSEMBLER เสร็จ



รูป เมื่อกด F8 หรือ G จะแสดงฟังก์ชั่น Download



รูป เมื่อกด H ใน Function Card download

```
IHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH;
:  .. 8051 FAMAILY CROSS ASSEMBLER ..    :
:  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^    :
:     F1.  Help MenuIHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH;
:     F2.  Dos Func.: .......  Download data code  ........      :
:     F3.  Load File:   ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^        :
:     F4.  Save file:       [H].  Help Use Card                  :
:     F5.  Editor.  :       [L].  Load Data file                 :
:     F6.  Assembler:       [M].  Move file to Card              :
:     F7.  Find Erro:       [D].  Dump memory  Card              :
:     F8.  Gard Down:       [S].  Show Data file                 :
:     F10. Quit to D:       [T].  Test Memory Card               :
:                   :       [Q].  Quit To Main                   :
HHHHHHHHHHHHHHHHHHH:                                             :
                    :             Select :.                      :
                    :                                            :
                    HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH<

                  ZDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD?
                  3 ... Load File For DownLoad To Card ...  3
                  3    Load File : a1.bin                   3
ZDDDDDDDDDDDDDDDDD3            Wait...                      3DDDDDDDDDDDDDDDDDD?
3                 3     ... Load file Complete ...          3                  3
- Move bar. Sel3            Press any key...               3or Function key   3
@DDDDDDDDDDDDDDDD@DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDYDDDDDDDDDDDDDDDDDDY
```

รูป เมื่อกด L ใน Function Card download

รูป เมื่อกด S ใน Function Card download

```
ZDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD?
3  ... Data Read Form File ...                                                3
3                                                                             3
3       Page : 1                                                              3
3     E0 BE CE BC 99 90 00 7B 01 CE BC B0 AD 7B 00 00  :  .......t.....t..    3
3     58 99 90 12 39 E0 99 12 39 CE 99 E0 9A 12 39 E0  :  S...6...6.....6.    3
3     99 12 39 58 99 90 E0 9B 12 39 E0 99 12 39 CE 99  :  ..6S.....6...6..    3
3     E0 9A 12 39 B8 E5 81 80 82 03 80 80 E6 EE E8 EA  :  ...6..y.z.x.....    3
3     E7 E6 24 00 00 00 00 00 00 00 00 00 00 00 00 00  :  ..".............    3
3     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  :  ................    3
3     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  :  ................    3
3     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  :  ................    3
3     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  :  ................    3
3     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  :  ................    3
3     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  :  ................    3
3     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  :  ................    3
3     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  :  ................    3
3     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  :  ................    3
3     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  :  ................    3
3     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  :  ................    3
3 Press any key...                                                            3
3                                                                             3
3                                                                             3
3                                                                             3
@DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDY
```

```
... Dump Data on Memory Card ..
Dump Memory On Card begin Address :100
A800:0100h 46 46 46 46 46 46 46 46 46 46 46 46 46 46 FE 46 | FFFFFFFFFFFFF.F
A800:0110h 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 | FFFFFFFFFFFFFFFF
A800:0120h 46 46 46 46 46 46 46 FE 46 46 46 46 46 46 46 46 | FFFFFFF.FFFFFFF
A800:0130h 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 | FFFFFFFFFFFFFFFF
A800:0140h 46 FE 46 46 46 46 46 46 46 46 46 46 46 46 46 46 | F.FFFFFFFFFFFFF
A800:0150h 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 | FFFFFFFFFFFFFFF
A800:0160h 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 | FFFFFFFFFFFFFFF
A800:0170h 46 FE 46 46 46 46 46 46 46 46 FE 46 46 46 46 46 | F.FFFFFFF.FFFF
A800:0180h 46 46 46 46 46 46 46 46 46 46 46 FE 46 46 46 46 | FFFFFFFFF.FFFF
A800:0190h 46 46 46 46 FE 46 46 46 46 46 46 46 46 46 46 46 | FFFF.FFFFFFFFF
A800:01A0h 46 FE 46 46 46 FE 46 46 46 46 46 46 46 46 46 46 | F.FFF.FFFFFFFFF
A800:01B0h 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 | FFFFFFFFFFFFFFF
A800:01C0h 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 | FFFFFFFFFFFFFFF
A800:01D0h 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 46 | FFFFFFFFFFFFFFF
A800:01E0h 46 46 46 46 46 46 46 46 FE 46 46 46 46 46 46 46 | FFFFFFF.FFFFFFF
A800:01F0h 46 46 46 46 46 46 46 46 FE 46 46 46 46 46 46 46 | FFFFFFF.FFFFFFF
Press any key (Esc) for break.
```

รูป เมื่อกด D ใน Function Card download

```
.. 8051 FAMAILY CROSS ASSEMBLER ..
F1. Help Menu.
F2. Dos Func.
F3. Load File.
F4. Save file.
F5. Editor.
F6. Assembler.
F7. Find Error
F8. Gard Down
F10.Quit to Do
                    ..... Download data cod
                    [H].  Help Use Card
                    [L].  Load Data fil
                    [M].  Move file to
                    [D].  Dump memory
                    [S].  Show Data fil
                    [T].  Test Memory Ca
                    [Q].  Quit To Main
                        Select :.
.... Test Card Download ....
Stat Address fill:100
A800:0100h    88.34
A800:0101h    88.54
A800:0102h    88.aa
A800:0103h    FF.78
A800:0104h    DF.ba
A800:0105h    88.35
A800:0106h    88.90
A800:0107h    88.fe
A800:0108h    88.35
A800:0109h    FF.12
Data write error.
88 88 88 88 88 FF 88 88 89 88
... continue (y/n) ?
- Move bar. Select by Pressing a Higthlighed
```

รูป เมื่อกด T ใน Function Card download

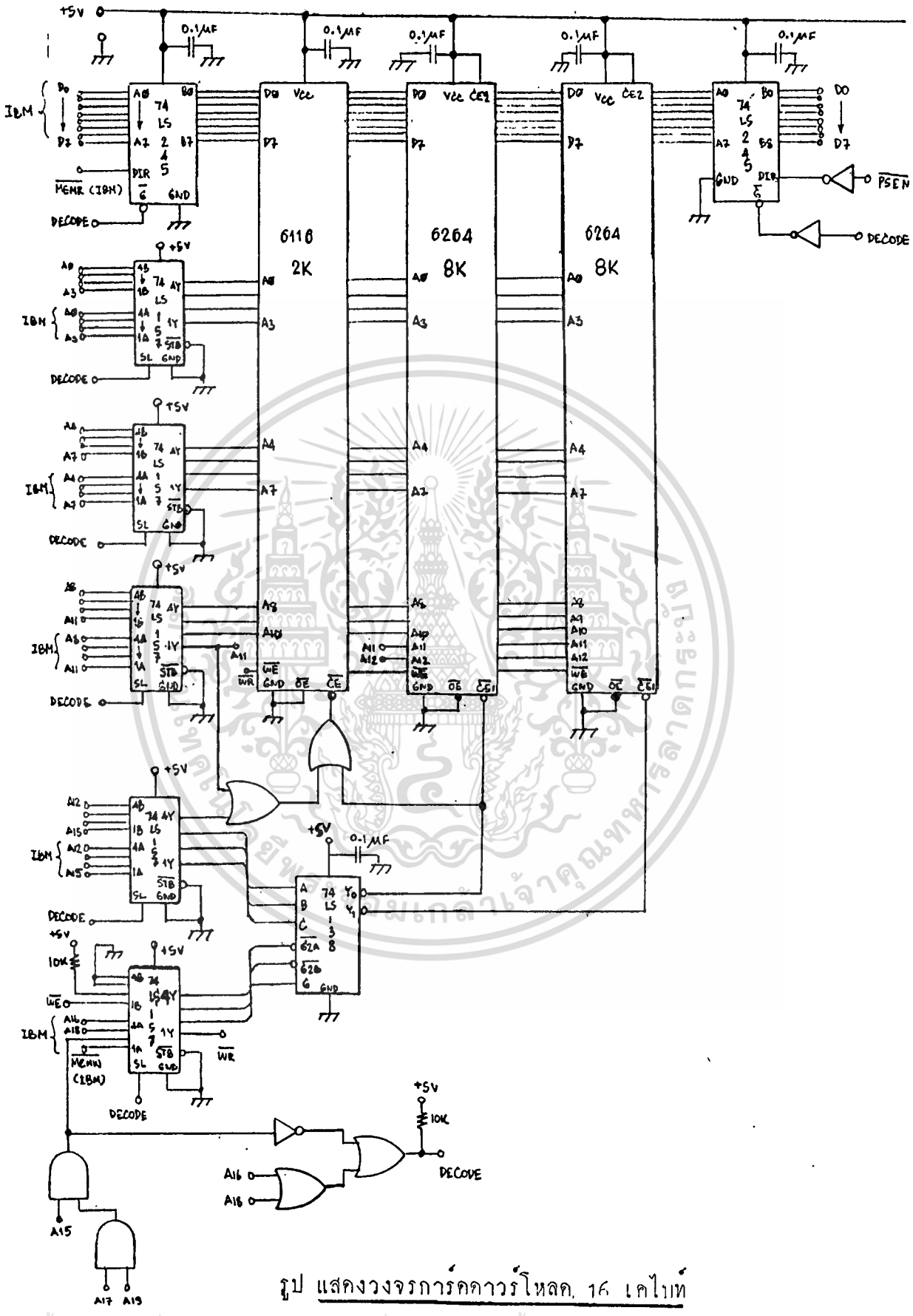
การประยุกต์ใช้งานร่วมกับ Hardware

Hardware นี้เป็นส่วนที่นำไปใช้ร่วมกับ software ที่สร้างขึ้น เพื่อให้เห็นประโยชน์ของ software นี้สามารถนำไปใช้งานได้จริง และมีประสิทธิภาพเต็มที่ โดยส่วนของ Hardware แบ่งเป็น 2 ส่วนคือ

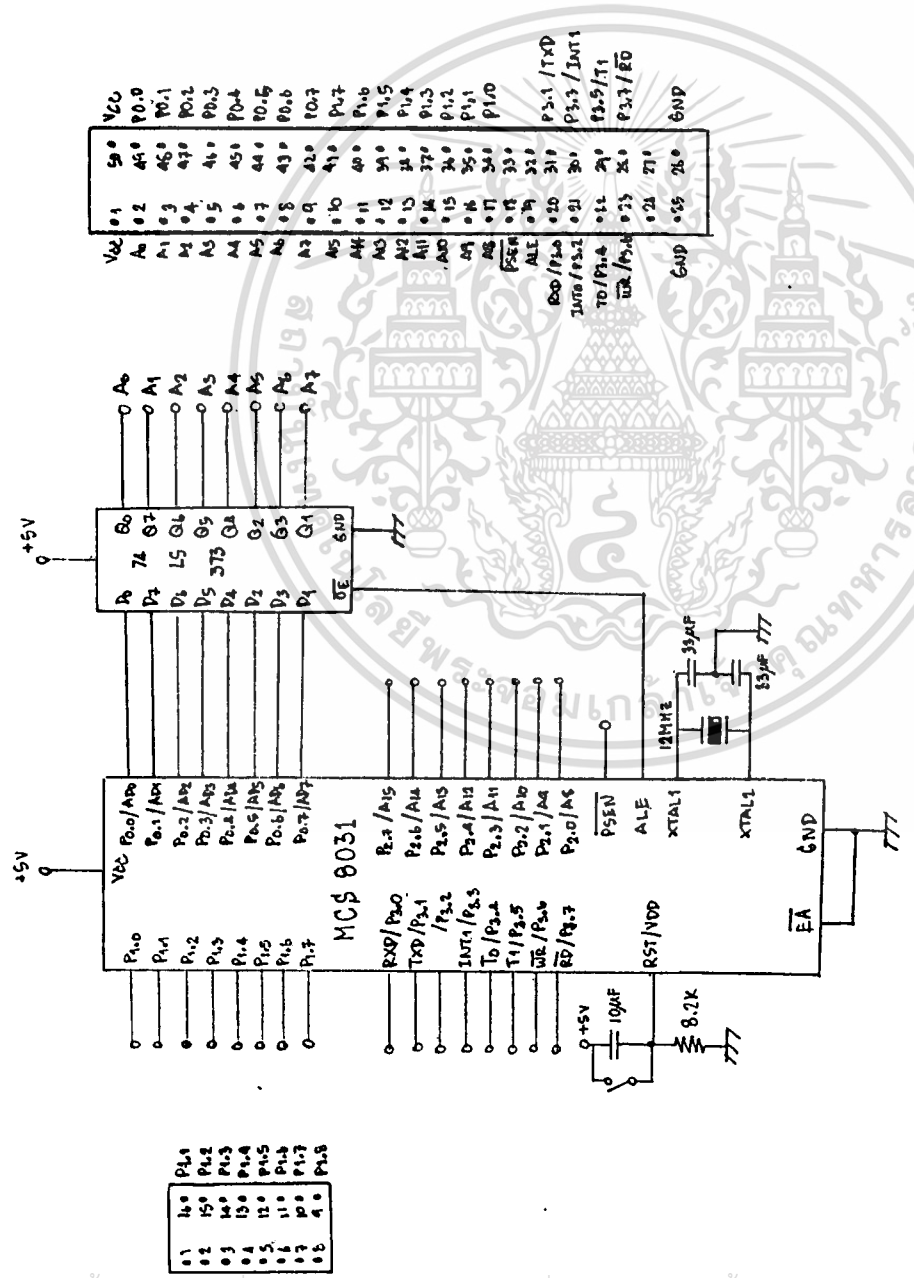
1). ส่วน Card Download เป็นส่วนที่รับข้อมูลจากเครื่อง IBM เข้าสู่ Single-board 8031 เพื่อให้ตัวมันทำงานตามโปรแกรมที่เก็บอยู่บน Card Download ขนาดของหน่วยความจำบน Card มี 16 Kbyte และขยายได้สูงสุด 64 Kbyte สำหรับรายละเอียดวงจรดูได้จากรูปด้านหลัง

2). ส่วน Single-board 8031 ส่วนนี้จะติดต่อกับ Card Download ซึ่งจะนำโปรแกรมจาก Card มา RUN และแสดงผลตามโปรแกรมที่อยู่บน Card ส่วน Hardware นี้เป็นส่วนที่นำมาประกอบกับส่วน Software เพื่อให้ผู้ใช้สามารถเห็นประโยชน์ของ Software ว่าสามารถนำไปใช้ได้จริง

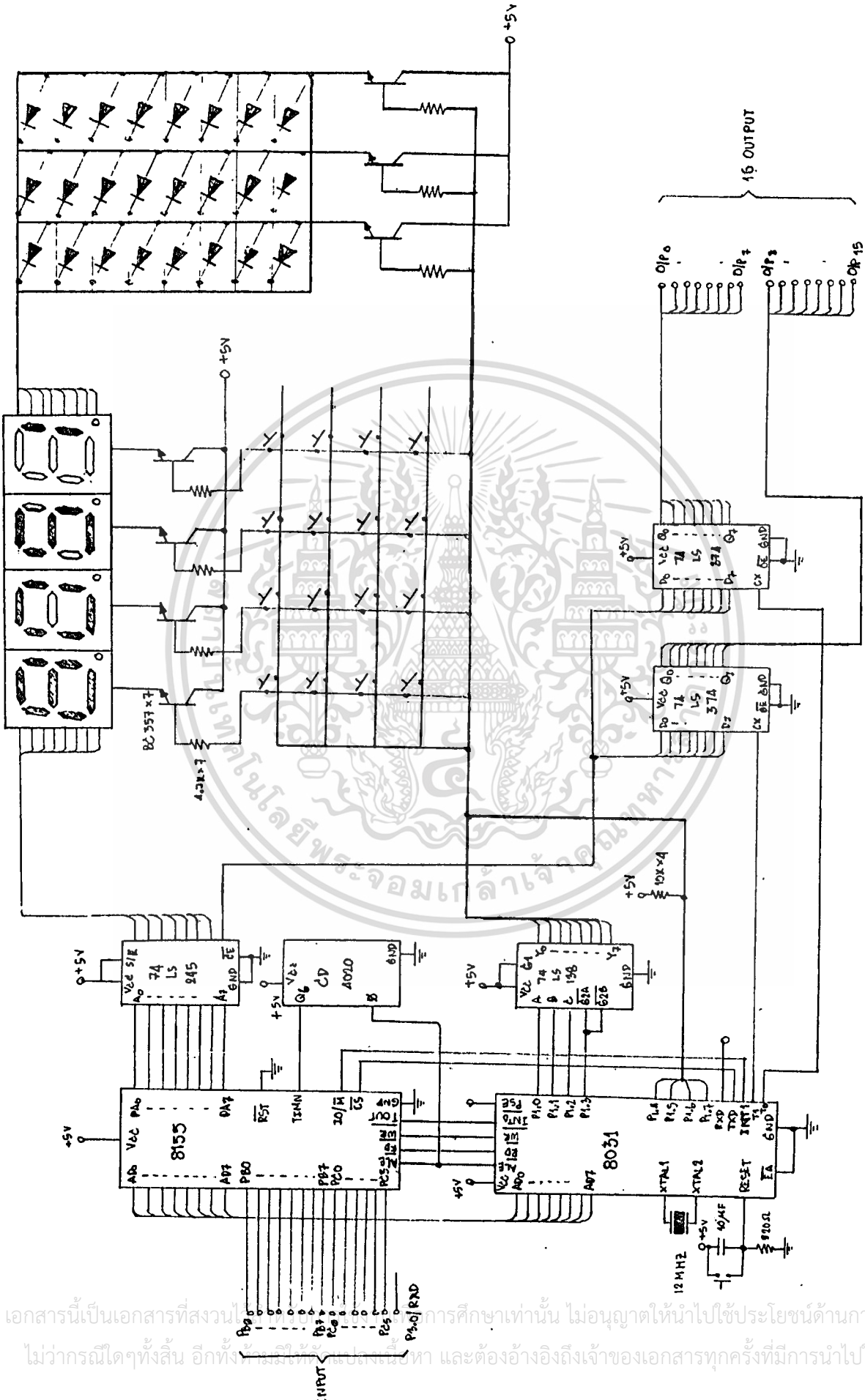
ในโครงงานที่ทำนี้ ผู้เขียนได้แสดงตัวอย่างการเขียนโปรแกรมที่ควบคุมชุดพัฒนา 8031 โดยควบคุมให้สามารถแสดงผล , ตรวจสอบ และติดต่อกับผู้ใช้ผ่านทาง Keyboard ของชุดพัฒนาเองได้ โดยรายละเอียดของโปรแกรมได้แสดงไว้ข้างล่าง



รูป แสดงวงจรการถอดรหัส 16 เคไบท์



รูป แสดงการจัดสัญญาณต่างๆของตัว 8031



รูป แสดงวงจรส่วนแสดงผลและส่วนทดลอง

บทวิจารณ์และสรุปผลโครงงานที่ทำ

ในโครงงานนี้ ได้เริ่มโดยเขียนโปรแกรม CROSS ASSEMBLER เพื่อแปลโปรแกรมภาษาแอสเซมบลี้ 8051 เป็น object code จากนั้นก็เริ่มทำ Card download ซึ่งมีขนาดหน่วยความจำ 16 กิโลไบท์ การ์ดนี้ใช้เสียบบน Slot ของ IBM และสายอีกด้านหนึ่งจะต่อกับ Single-board 8051 โดยใช้หน่วยความจำร่วมกัน จากนั้นก็เขียนโปรแกรมแอสเซมบลี้ 8051 บนเครื่อง IBM ใช้ CROSS ASSEMBLER คอมไพล์ แล้วส่ง object code ไปยัง Card download เพื่อควบคุม Single-board 8051 ให้ทำงานตามโปรแกรมที่กำหนด

ปัญหาที่เกิดก็คือ สัญญาณรบกวน (Noise) ทำให้ข้อมูลที่โหลดเข้าไปเสียหายได้

จะเห็นว่าโครงงานที่ทำนี้ ให้ความสะดวกแก่ผู้ที่พัฒนาระบบไมโครโปรเซสเซอร์ในตระกูล MCS-51 เป็นอย่างมาก ในการแก้ไขข้อมูลโปรแกรมตอนช่วงพัฒนาตอนแรก

ภาคผนวก

ภาคผนวกแสดงโปรแกรม MONITOR ส่วนที่ CONTROL HARDWARE ของ SINGLE - BOARD 8051 ซึ่งผ่านการ COMPILED โดย CROSS ASSEMBLER

```
                                        ;PROGRAM 8031 display  8 0 3 1 * * * ( * = ..... ) for initial
                    ;And read scan key board input for user press
                    ;display contunue (Not delay time )
                              org       0000h
                    ;Set data initial for display
                    ;---------**********************-------------------
                       buff0    equ     030h
                       buff1    equ     031h
                       buff2    equ     032h
                       buff3    equ     033h
                    ;---------**********************-------------------
0000  D2 B3            start:   setb    p3.3           ; conect ic/a of 8155
                                                       ; select  io port
0002  90 FF 00                 mov     dptr,#0ff00h   ; no use in address of down-load card
0005  74 01                    mov     a,#01h         ; Pa:o/p Pb,Pc:i/p timer:stop
0007  C2 B1                    clr     p3.1           ; conect /cs of 8155
0009  F0                       movx    @dptr,a
000A  A3                       inc     dptr
000B  11 50                    acall   set_buff
000D  78 80                    mov     r0,#80h
000F  11 47                    acall   dis_int
0011  78 C0                    mov     r0,#0c0h
0013  11 47                    acall   dis_int
0015  78 B0                    mov     r0,#0b0h
0017  11 47                    acall   dis_int
0019  78 F9                    mov     r0,#0f9h  ; display initial '8031'
001B  11 47                    acall   dis_int
001D  11 50                    acall   set_buff
001F  11 49                    acall   dis_int1
0021  78 89                    mov     r0,#089h
0023  11 47                    acall   dis_int
0025  78 86                    mov     r0,#086h
0027  11 47                    acall   dis_int
0029  78 C7                    mov     r0,#0c7h
002B  11 47                    acall   dis_int
002D  78 C0                    mov     r0,#0c0h  ; display initial 'HELO'
002F  11 47                    acall   dis_int
0031  11 50                    acall   set_buff
0033  11 49                    acall   dis_int1
0035  78 BF                    mov     r0,#0bfh
0037  11 47                    acall   dis_int
0039  78 C0                    mov     r0,#0c0h
003B  11 47                    acall   dis_int
003D  78 89                    mov     r0,#089h
003F  11 47                    acall   dis_int
0041  78 BF                    mov     r0,#0bfh  ; display initial '-OH-'
0043  11 47                    acall   dis_int
0045  80 36                    sjmp    loop
                    ;---------**********************-------------------
0047  31 56            dis_int: acall   shift_buff1
0049  AF 09            dis_int1: mov    r7,09h
004B  11 5D            loopdis:  acall  loop_int
004D  DF FC                      djnz   r7,loopdis
004F  22                         ret
                    ;---------**********************-------------------
0050  75 30 FF          set_buff: mov    buff0,#0ffh     ; buffer initial digit 0
0053  75 31 FF                    mov    buff1,#0ffh     ; buffer initial digit 1
0056  75 32 FF                    mov    buff2,#0ffh     ; buffer initial digit 2
0059  75 33 FF                    mov    buff3,#0ffh     ; buffer initial digit 3
005C  22                          ret
                    ;---------**********************-------------------
005D  E5 30            loop_int: mov    a,buff0         ;digit 0
005F  F0                         movx   @dptr,a
0060  53 90 F0                   anl    p1,#0f0h
0063  31 3A                      acall  delay
0065  D2 90                      setb   p1.0            ;digit 1
0067  E5 31                      mov    a,buff1
0069  F0                         movx   @dptr,a
006A  31 3A                      acall  delay
006C  C2 90                      clr    p1.0            ;digit 2
006E  D2 91                      setb   p1.1
0070  E5 32                      mov    a,buff2
```

```
0072  F0                    movx    @dptr,a
0073  31 3A                 acall   delay
0075  D2 90                 setb    p1.0            ;digit 3
0077  E5 33                 mov     a,buff3
0079  F0                    movx    @dptr,a
007A  31 3A                 acall   delay
007C  22                    ret
                  ;---------**********----------
007D  E5 30         loop:   mov     a,buff0         ;digit 0
007F  F0                    movx    @dptr,a
0080  53 90 F0              anl     p1,#0f0h
0083  11 B2                 acall   Readkey0
0085  D2 90                 setb    p1.0            ;digit 1
0087  E5 31                 mov     a,buff1
0089  F0                    movx    @dptr,a
008A  11 D4                 acall   Readkey1
008C  C2 90                 clr     p1.0            ;digit 2
008E  D2 91                 setb    p1.1
0090  E5 32                 mov     a,buff2
0092  F0                    movx    @dptr,a
0093  11 F6                 acall   Readkey2
0095  D2 90                 setb    p1.0            ;digit 3
0097  E5 33                 mov     a,buff3
0099  F0                    movx    @dptr,a
009A  31 18                 acall   Readkey3
009C  53 90 F0              anl     p1,#0f0h        ;digit 4
009F  D2 92                 setb    p1.2
00A1  74 00                 mov     a,#00h          ;diplay .......
00A3  F0                    movx    @dptr,a
00A4  31 3A                 acall   delay
00A6  D2 90                 setb    p1.0            ;digit 5
00A8  31 3A                 acall   delay
00AA  C2 90                 clr     p1.0            ;digit 6
00AC  D2 91                 setb    p1.1
00AE  31 3A                 acall   delay
00B0  80 CB                 sjmp    loop
                  ;---------**********----------
00B2  E5 90         Readkey0: mov     a,p1
00B4  B4 E0 05              cjne    a,#11100000b,key4  ;compare check key #00 press
00B7  78 C0                 mov     r0,#0c0h
00B9  31 43                 acall   shift_buff
00BB  22                    ret
00BC  B4 D0 05      key4:   cjne    a,#11010000b,key8  ;compare check key #04 press
00BF  78 99                 mov     r0,#99h
00C1  31 43                 acall   shift_buff
00C3  22                    ret
00C4  B4 B0 05      key8:   cjne    a,#10110000b,keyc  ;compare check key #08 press
00C7  78 80                 mov     r0,#80h
00C9  31 43                 acall   shift_buff
00CB  22                    ret
00CC  B4 70 68      keyc:   cjne    a,#01110000b,delay  ;compare check key #0c press
00CF  78 C6                 mov     r0,#0c6h
00D1  31 43                 acall   shift_buff
00D3  22                    ret
                  ;---------**********----------
00D4  E5 90         Readkey1: mov     a,p1
00D6  B4 E1 05              cjne    a,#11100001b,key5  ;compare check key #01 press
00D9  78 F9                 mov     r0,#0f9h
00DB  31 43                 acall   shift_buff
00DD  22                    ret
00DE  B4 D1 05      key5:   cjne    a,#11010001b,key9  ;compare check key #05 press
00E1  78 92                 mov     r0,#92h
00E3  31 43                 acall   shift_buff
00E5  22                    ret
00E6  B4 B1 05      key9:   cjne    a,#10110001b,keyd  ;compare check key #09 press
00E9  78 90                 mov     r0,#90h
00EB  31 43                 acall   shift_buff
00ED  22                    ret
00EE  B4 71 49      keyd:   cjne    a,#01110001b,delay  ;compare check key #0d press
```

```
00F1  78 A1                   mov     r0,#0a1h
00F3  31 43                   acall   shift_buff
00F5  22                      ret
                    ;---------**********----------
00F6  E5 90           Readkey2: mov     a,p1
00F8  B4 E2 05                cjne    a,#11100010b,key6  ;compare check key #02 press
00FB  78 A4                   mov     r0,#0a4h
00FD  31 43                   acall   shift_buff
00FF  22                      ret
0100  B4 D2 05        key6:   cjne    a,#11010010b,keya  ;compare check key #06 press
0103  78 82                   mov     r0,#82h
0105  31 43                   acall   shift_buff
0107  22                      ret
0108  B4 B2 05        keya:   cjne    a,#10110010b,keye   ;compare check key #0a press
010B  78 88                   mov     r0,#88h
010D  31 43                   acall   shift_buff
010F  22                      ret
0110  B4 72 27        keye:   cjne    a,#01110010b,delay  ;compare check key #0e press
0113  78 86                   mov     r0,#086h
0115  31 43                   acall   shift_buff
0117  22                      ret
                    ;---------**********----------
0118  E5 90           Readkey3: mov     a,p1
011A  B4 E3 05                cjne    a,#11100011b,key7  ;compare check key #03 press
011D  78 B0                   mov     r0,#0b0h
011F  31 43                   acall   shift_buff
0121  22                      ret
0122  B4 D3 05        key7:   cjne    a,#11010011b,keyb  ;compare check key #07 press
0125  78 F8                   mov     r0,#0f8h
0127  31 43                   acall   shift_buff
0129  22                      ret
012A  B4 B3 05        keyb:   cjne    a,#10110011b,keyf   ;compare check key #0b press
012D  78 83                   mov     r0,#83h
012F  31 43                   acall   shift_buff
0131  22                      ret
0132  B4 73 05        keyf:   cjne    a,#01110011b,delay  ;compare check key #0f press
0135  78 8E                   mov     r0,#08eh
0137  31 43                   acall   shift_buff
0139  22                      ret
                    ;---------**********----------
013A  7A 02           delay:  mov     r2,#02h
013C  7B FF           delay1: mov     r3,#0ffh
013E  DB FE                   djnz    r3,$
0140  DA FA                   djnz    r2,delay1
0142  22              exit:   ret
                    ;---------**********----------
0143  C0 E0           shift_buff: push    acc
0145  31 56                   acall   shift_buff1
0147  74 FF                   mov     a,#0ffh
0149  F0                      movx    @dptr,a
014A  D0 F0                   pop     b
014C  31 3A           repeat: acall   delay
014E  E5 90                   mov     a,p1
0150  B5 F0 EF                cjne    a,b,exit
0153  80 F7                   sjmp    repeat
0155  22              exit:   ret
                    ;---------**********----------
0156  E5 31          shift_buff1: mov     a,buff1
0158  F5 30                   mov     buff0,a
015A  E5 32                   mov     a,buff2
015C  F5 31                   mov     buff1,a
015E  E5 33                   mov     a,buff3
0160  F5 32                   mov     buff2,a
0162  E8                      mov     a,r0
0163  F5 33                   mov     buff3,a
0165  22                      ret
                    ;---------**********----------
                              end.
```

Interrupt Response Time: To finish execution of current instruction, respond to the interrupt request, push the PC and to vector to the first instruction of the interrupt service program requires 38 to 81 oscillator periods (3 to 7μs @ 12 MHz).

INSTRUCTIONS THAT AFFECT FLAG SETTINGS[1]

INSTRUCTION	C	OV	AC	INSTRUCTION	C	OV	AC
ADD	X	X	X	CLR C	0		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	0	X		ANL C,/bit	X		
DIV	0	X		ORL C,bit	X		
DA	X			ORL C,/bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

[1]Note that operations on SFR byte address 208 or bit addresses 209-215 (i.e., the PSW or bits in the PSW) will also affect flag settings.

Notes on instruction set and addressing modes:

- **Rn** —Register R7-R0 of the currently selected Register Bank.
- **direct** —8-bit internal data location's address. This could be an Internal Data RAM location (0-127) or a SFR [i.e., I/O port, control register, status register, etc. (128-255)].
- **@Ri** —8-bit internal data RAM location (0-255) addressed indirectly through register R1 or R0.
- **#data** —8-bit constant included in instruction.
- **#data 16** —16-bit constant included in instruction
- **addr 16** —16-bit destination address. Used by LCALL & LJMP. A branch can be anywhere within the 64K-byte Program Memory address space.
- **addr 11** —11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.
- **rel** —Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is -128 to +127 bytes relative to first byte of the following instruction.
- **bit** —Direct Addressed bit in Internal Data RAM or Special Function Register.
- **•** —New operation not provided by 8048AH/8049AH.

ARITHMETIC OPERATIONS

Mnemonic		Description	Byte	Oscillator Period
ADD	A,Rn	Add register to Accumulator	1	12
ADD	A,direct	Add direct byte to Accumulator	2	12
ADD	A,@Ri	Add indirect RAM to Accumulator	1	12
ADD	A,#data	Add immediate data to Accumulator	2	12
ADDC	A,Rn	Add register to Accumulator with Carry	1	12
ADDC	A,direct	Add direct byte to Accumulator with Carry	2	12
ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDC	A,#data	Add immediate data to Acc with Carry	2	12
SUBB	A,Rn	Subtract register from Acc with borrow	1	12
SUBB	A,direct	Subtract direct byte from Acc with borrow	2	12

ARITHMETIC OPERATIONS Cont.

Mnemonic		Description	Byte	Oscillator Period
SUBB	A,@Ri	Subtract indirect RAM from Acc with borrow	1	12
SUBB	A,#data	Subtract immediate data from Acc with borrow	2	12
INC	A	Increment Accumulator	1	12
INC	Rn	Increment register	1	12
INC	direct	Increment direct byte	2	12
INC	@Ri	Increment indirect RAM	1	12
DEC	A	Decrement Accumulator	1	12
DEC	Rn	Decrement Register	1	12
DEC	direct	Decrement direct byte	2	12
DEC	@Ri	Decrement indirect RAM	1	12
INC	DPTR	Increment Data Pointer	1	24
MUL	AB	Multiply A & B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12

ตาราง สรุปชุดคำสั่ง 8051

LOGICAL OPERATIONS

Mnemonic		Description	Byte	Oscillator Period
ANL	A,Rn	AND register Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND indirect RAM to Accumulator	1	12
ANL	A,#data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,#data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12
ORL	A,#data	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,#data	OR immediate data to direct byte	3	24
XRL	A,Rn	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12

LOGICAL OPERATIONS Cont.

Mnemonic		Description	Byte	Oscillator Period
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement Accumulator	1	12
RL	A	Rotate Accumulator Left	1	12
RCL	A	Rotate Accumulator Left through the Carry	1	12
RR	A	Rotate Accumulator Right	1	12
RRC	A	Rotate Accumulator Right through the Carry	1	12
SWAP	A	Swap nibbles within the Accumulator	1	12

ตาราง สรุปชุดคำสั่ง 8051 (ต่อ)

DATA TRANSFER

Mnemonic		Description	Byte	Oscillator Period
MOV	A,Rn	Move register to Accumulator	1	12
MOV	A,direct	Move direct byte to Accumulator	2	12
MOV	A,@Ri	Move indirect RAM to Accumulator	1	12
MOV	A,#data	Move immediate data to Accumulator	2	12
MOV	Rn,A	Move Accumulator to register	1	12
MOV	Rn,direct	Move direct byte to register	2	24
MOV	Rn,#data	Move immediate data to register	2	12
MOV	direct,A	Move Accumulator to direct byte	2	12
MOV	direct,Rn	Move register to direct byte	2	24
MOV	direct,direct	Move direct byte to direct	3	24
MOV	direct,@Ri	Move indirect RAM to direct byte	2	24
MOV	direct,#data	Move immediate data to direct byte	3	24
MOV	@Ri,A	Move Accumulator to indirect RAM	1	12
MOV	@Ri,direct	Move direct byte to indirect RAM	2	24
MOV	@Ri,#data	Move immediate data to indirect RAM	2	12

DATA TRANSFER Cont.

Mnemonic		Description	Byte	Oscillator Period
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	24
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to Acc	1	24
MOVC	A,@A+PC	Move Code byte relative to PC and Acc	1	24
MOVX	A,@Ri	Move External RAM (8-bit addr) to Acc	1	24
MOVX	A,@DPTR	Move External RAM (16-bit addr) to Acc	1	24
MOVX	@Ri,A	Move Acc to External RAM (8-bit addr)	1	24
MOVX	@DPTR,A	Move Acc to External RAM (16-bit addr)	1	24
PUSH	direct	Push direct byte onto stack	2	24
POP	direct	Pop direct byte from stack	2	24
XCH	A,Rn	Exchange register with Accumulator	1	12
XCH	A,direct	Exchange direct byte with Accumulator	2	12
XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	12
XCHD	A,@Ri	Exchange low-order Digit indirect RAM with Acc	1	12

ตาราง สรุปชุดคำสั่ง 8051 (ต่อ)

BOOLEAN VARIABLE MANIPULATION

Mnemonic		Description	Byte	Oscillator Period
CLR	C	Clear Carry	1	12
CLR	bit	Clear direct bit	2	12
SETB	C	Set Carry	1	12
SETB	bit	Set direct bit	2	12
CPL	C	Complement Carry	1	12
CPL	bit	Complement direct bit	2	12
ANL	C,bit	AND direct bit to Carry	2	24
ANL	C,/bit	AND complement of direct bit to Carry	2	24
ORL	C,bit	OR direct bit to Carry	2	24
ORL	C,/bit	OR complement of direct bit to Carry	2	24
MOV	C,bit	Move direct bit to Carry	2	12
MOV	bit,C	Move Carry to direct bit	2	24
JC	rel	Jump if Carry is set	2	24
JNC	rel	Jump if Carry not set	2	24
JB	bit,rel	Jump if direct Bit is set	3	24
JNB	bit,rel	Jump if direct Bit is Not set	3	24
JBC	bit,rel	Jump if direct Bit is set & clear bit	3	24

PROGRAM BRANCHING

Mnemonic		Description	Byte	Oscillator Period
ACALL	addr11	Absolute Subroutine Call	2	24
LCALL	addr16	Long Subroutine Call	3	24
RET		Return from Subroutine	1	24

PROGRAM BRANCHING Cont.

Mnemonic		Description	Byte	Oscillator Period
RETI		Return from interrupt	1	24
AJMP	addr11	Absolute Jump	2	24
LJMP	addr16	Long Jump	3	24
SJMP	rel	Short Jump (relative addr)	2	24
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	24
JZ	rel	Jump if Accumulator is Zero	2	24
JNZ	rel	Jump if Accumulator is Not Zero	2	24
CJNE	A,direct,rel	Compare direct byte to Acc and Jump if Not Equal	3	24
CJNE	A,#data,rel	Compare immediate to Acc and Jump if Not Equal	3	24
CJNE	Rn,#data,rel	Compare immediate to register and Jump If Not Equal	3	24
CJNE	@Ri,#data,rel	Compare immediate to indirect and Jump if Not Equal	3	24
DJNZ	Rn,rel	Decrement register and Jump if Not Zero	3	24
DJNZ	direct,rel	Decrement direct byte and Jump if Not Zero	3	24
NOP		No Operation	1	12

ตาราง สรุปชุดคำสั่ง 8051 (ต่อ)

กิตติกรรมประกาศ

ในโครงงาน 8051 CROSS ASSEMBLER นี้ ทางคณะผู้จัดทำขอขอบคุณ อาจารย์ วิทยา ทิพย์สุวรรณพร ที่ช่วยจัดหาอุปกรณ์ต่างๆที่ใช้ในโครงงาน ขอบคุณ คุณ น้ำผึ้ง วัฒนชัยกุลวัฒน์ (ช่างศิลป์ 33) และคุณ กัญญา จักรปั่น (กสท.) ที่ช่วยพิมพ์ปริญญานิพนธ์นี้จนสำเร็จ

บรรณานุกรม

1. Ray Duncan , **"Advanced Msdos"** , Microsoft Corporation 1986.

2. Intel ," **Microcontroller Handbook** ", Intel Corporation 1983.

3. Borland , " **Turbo Pascal version 3.0 Reference Manual** " , Borland International , 1985.

4. Borland , " **Turbo Pascal Owner's Handbook version 4.0** " , Borland International U.S.A ,1987.

5. IBM , " **Technical Refrence** " , IBM Corp. 1983.

6. Lewis C. Eggebrecht , " **Interfacing to the IBM Personal Computer** ", Howard W. Sams & Co.,Inc. U.S.A 1983.

7. รศ.ยืน ภู่วรวรรณ & ดร.ชัยยงค์ วงค์ชัยสุวัฒน์ & น.ท.ดร.ไพศาล สงวนหมู่ , **" เทคโนโลยีไมโครคอมพิวเตอร์ 16 บิต "** , บริษัท ซีเอ็ดยูเคชั่น จำกัด ,2530.

8. ดร.สุชาย ธนวเสถียร & วิชัย จิวังกูร ," **โครงสร้างข้อมูลเพื่อการออกแบบโปรแกรมคอมพิวเตอร์** ", บริษัท ซีเอ็ดยูเคชั่น จำกัด ,2529.