# รายการอ้างอิง


1. Ramesh, B., S., Jamadagni., H., S., and Marco Oligiati. Microprocessor Laboratory Primer. Bangalore, India: CEDT, 1987.

2. ก้าวสู่โลกคอมพิวเตอร์ด้วยซิงเกอร์บอร์ด ET-3.5 เครื่องมืออันทรงประสิทธิภาพ. เซมิคอนดัคเตอร์อิเลคทรอนิคส์ ฉบับที่ 107 (มิถุนายน 2534): 16-19.

3. Advanced Electronic Systems. SDA-85 System User's Manual. Bangalore, India.

4. King Instrument Electronics Co., Ltd. Computer Interface Control Lab CIC -100 User Guide. Taiwan: King Instrument Electronics Co., Ltd., 1994.

5. Electro Systems Associate Pvt Ltd. Interface Mannuals. Bangalore, India: Electro Systems Associates Pvt Ltd.

6. King Instrument Electronics Co.,Ltd. Computer Interface Control Lab CIC -100 Applications Module Experiment Manual. Taiwan: King Instrument Electronics Co., Ltd., 1994.

7. Gates, S., C., and Becker, J. Laboratory Automation Using the IBM PC. U.S.A.: Prentice - Hall International, 1989.

8. Michael Tischer. PC Intern. U.S.A.: ABACUS, 1992.

9. ธานินทร์ ถาวรศาสนวงศ์ และ ทินกร ดุ๊ก. การอินเทอร์เฟส IBM PC. กรุงเทพมหานคร: ฟิสิกส์เซ็นเตอร์การพิมพ์.

10. IBM Corporation. Technical Reference Personal Computer XT. U.S.A.: IBM Corp., 1983.

11. Sandige, R., S. Modern Digital Design. Singapore: McGraw - Hill, 1970.

12. คู่มือทรานซิสเตอร์. กรุงเทพมหานคร: บริษัทซีเอ็ด ยูเคชั่น จำกัด, 2531.

13. คู่มือไอซีซีพียูพอร์ตและหน่วยความจำ. กรุงเทพมหานคร: บริษัทซีเอ็ด ยูเคชั่น จำกัด, 2529.

14. Lawrence, P., D., and Konrad Mauch. Real Time Microcomputer System Design. Singapore: McGraw - Hill, 1988.

15. Motorola Incorporation. Motorola CMOS Intergrated Circuits. U.S.A.: Motorola Inc., 1978.

16. National Semiconductor Corporation. Data Acquisition Handbook. U.S.A.: National Semiconductor Corporation.

17. ชูชัย ธนสารตั้งเจริญ และ ทินกร ดุ๊ก. การสื่อสารข้อมูล. กรุงเทพมหานคร: ฟิสิกส์เซ็นเตอร์การพิมพ์, 2533.

18. ยืน ภู่วรวรรณ, ชัยณรงค์ วงศ์ชัยสุวัฒน์ และ ไพศาล สงวนหมู่. เทคโนโลยี ไมโครคอมพิวเตอร์ 16 บิต. กรุงเทพมหานคร: บริษัทซีเอ็ด ยูเคชั่น จำกัด, 2530.

19. Borland International Incorporation. TURBO C User's Guide. U.S.A.: Borland International, Inc., 1987.

20. Borland International Incorporation. TURBO C Reference Guide. U.S.A.: Borland International, Inc., 1987.

21. วันชัย พักอินทร์. 74HC ซีมอสตัวใหม่. เซมิคอนดัคเตอร์ อิเลคทรอนิคส์ ฉบับที่ 71

(มิถุนายน - กรกฎาคม 2529): 150-153.

22. Hans - Peter Messmer. The Indispensable PC Hardware Book. 2 nd ed. U.S.A.: Addison - Wesley, 1975.

23. ชูเกียรติ วัฒนากูล. ไอซีลอจิกตระกูลต่าง ๆ. เซมิคอนดัคเตอร์ อิเลคทรอนิคส์ ฉบับที่ 91 (มีนาคม - เมษายน 2529): 261-267.

24. Miller, L., H., and Quilici, A., E. Joy of C. U.S.A.: John Wisley & Sons, 1993.

25. Kernighan, B., W., and Ritchie, D., M. The C Programming Language. 2 nd ed. Prentice - Hall Software Series. U.S.A.: Prentice Hall, 1988.

26. ดวงแก้ว สวามิภักดิ์. การโปรแกรมภาษา C. กรุงเทพมหานคร: บริษัท เอซ.เอ็น.กรุ๊ปจำกัด, 2537.

27. เฮอร์เบิร์ต ชิลด์. การประยุกต์การใช้งานภาษาซี. แปลโดย ศิวัฒน์ ศิวะบวร, พรชัย จักรธำรงค์ และ จิรศักดิ์ ชัยวิริยะกุล. กรุงเทพมหานคร: บริษัท เอซ.เอ็น.กรุ๊ป จำกัด, 2536.

28. ยืน ภู่วรวรรณ. มุมนักทดลองดิจิตอล พอร์ตสื่อสาร 8250. เซมิคอนดัคเตอร์ อิเลคทรอนิคส์ ฉบับที่ 94 (สิงหาคม 2532): 296-301.

29. ยืน ภู่วรวรรณ. มุมนักทดลองดิจิตอล พอร์ตสื่อสาร 8250. เซมิคอนดัคเตอร์ อิเลคทรอนิคส์ ฉบับที่ 95 (กันยายน 2532): 260-264.

30. ไกรวุฒิ โรจน์ประเสริฐสุด. บอร์ดควบคุมสเต็ปเปอร์มอเตอร์ด้วยเครื่องพีซี. รวมโครงงานอิเล็กทรอนิกส์ไมโครคอมพิวเตอร์. กรุงเทพมหานคร: บริษัท ซีเอ็ด ยูเคชั่น จำกัด, 2539.

31. สุวลัย กลั่นความดี และ วราภรณ์ เชาว์วิศิษฐ. คู่มือปฏิบัติการระบบควบคุม. กรุงเทพมหานคร: ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย, 2534.

32. บริษัท แอนาดิจิท จำกัด. เซมิคอนดัคเตอร์ อิเลคทรอนิคส์ ฉบับที่ 92 (พฤษภาคม - มิถุนายน 2532): 112.

# ภาคผนวก

# ภาคผนวก ก

## ใบงานการทดลอง

การทดลองที่ 1 : Familiarization with the Training Set

1. วัตถุประสงค์

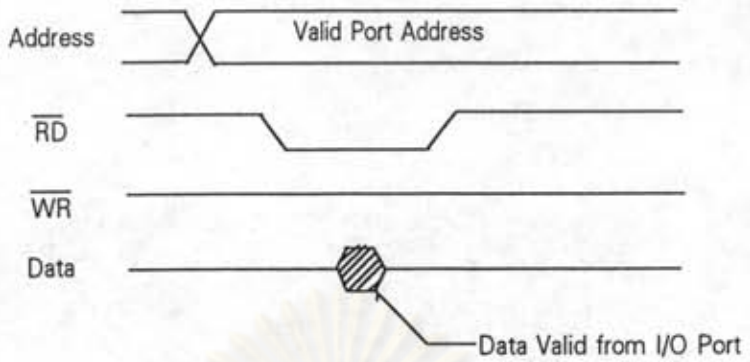
1. เพื่อเรียนรู้ส่วนประกอบต่าง ๆ ของชุดทดลองอย่างกว้าง ๆ
2. เพื่อเรียนรู้วิธีการใช้งานชุดทดลอง

2. ทฤษฎี

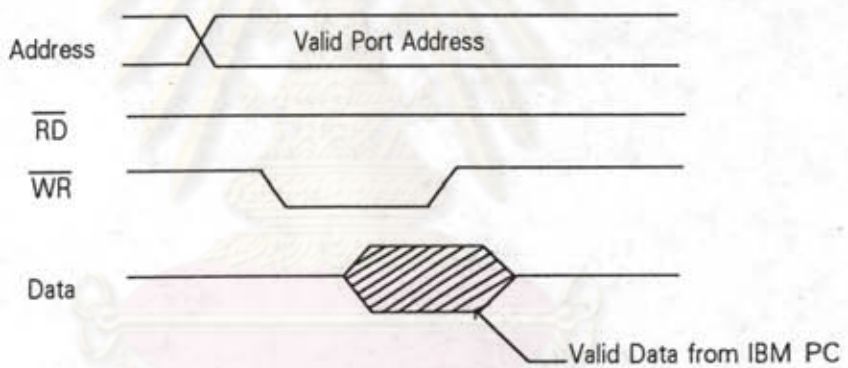
เมนบอร์ดของชุดฝึกทดลองจะทำหน้าที่ในการสร้างสัญญาณควบคุมระบบไมโครโปรเซสเซอร์ เป็นสัญญาณที่มีลักษณะเหมือนกับสัญญาณการเขียนอ่าน ข้อมูลระหว่าง CPU และอุปกรณ์ต่างๆ หรือที่เราเรียกว่า ระบบบัส เมนบอร์ดทำหน้าที่สร้างระบบบัสจำลองเพื่อนำไปใช้ในการอินเตอร์เฟซกับอุปกรณ์ต่าง ๆ โดยที่เราไม่จำเป็นต้องนำสัญญาณจากสล็อตของ IBM PC โดยตรง ระบบบัสจำลองถูกสร้างขึ้นโดยใช้พอร์ตขนาน (บางครั้งเรียก พอร์ตเครื่องพิมพ์) ของ IBM PC ระบบบัสประกอบด้วยสัญญาณต่างๆดังนี้

1. แอดเดรสบัส (Address Bus)
2. บัสข้อมูล (Data Bus)
3. สัญญาณควบคุมการเขียนอ่านข้อมูล (READ , WRITE Signal)
4. สัญญาณการร้องขออินเตอร์รัปต์ CPU (IRQ)

รูปที่ ก.1.1 และ ก.1.2 แสดง Timing Diagram ของระบบบัส ในการเขียนและอ่านข้อมูลระหว่าง CPU และอุปกรณ์ต่าง ๆ




รูปที่ ก.1.1 แสดง Timing Diagram ของ READ Cycle




รูปที่ ก.1.2 แสดงลักษณะของ WRITE Cycle

ขั้นตอนการอ่านข้อมูล

1. เครื่อง IBM PC ส่งสัญญาณมายัง บัสแอดเดรส เพื่อเลือกอุปกรณ์ที่จะทำการติดต่อด้วย
2. สัญญาณการอ่านข้อมูล ($\overline{RD}$) เปลี่ยนสถานะ เป็น low
3. อุปกรณ์ ส่งข้อมูลมายังบัสข้อมูล
4. เครื่อง IBM PC อ่านข้อมูลจากบัสข้อมูล
5. สัญญาณ $\overline{RD}$ เปลี่ยนสถานะ เป็น High
6. อุปกรณ์ปล่อยการยึดเกาะบัสข้อมูล บัสข้อมูลเปลี่ยนสถานะกลับเป็น High Impedance(Hi-Z)

ขั้นตอนการเขียนข้อมูล

1. เครื่อง IBM PC ส่งสัญญาณ มายังบัสแอดเดรส เพื่อเลือกอุปกรณ์ที่จะทำการติดต่อด้วย
2. สัญญาณ $\overline{WR}$ เปลี่ยนสถานะ เป็น Low
3. เครื่อง IBM PC ส่งข้อมูลมายังบัสข้อมูล
4. สัญญาณ $\overline{WR}$ เปลี่ยนสถานะ เป็น High
5. อุปกรณ์อ่านข้อมูลจาก บัสข้อมูล ในช่วงขอบขาขึ้นของสัญญาณ $\overline{WR}$
6. เครื่อง IBM PC ปล่อยการยึดเกาะบัสข้อมูล เปลี่ยนสถานะกลับเป็น Hi-Z

**คำสั่งที่ใช้ในการเขียนอ่านข้อมูล**

จะใช้คำสั่ง IN ในการอ่านข้อมูลจากอุปกรณ์มายัง CPU และคำสั่ง OUT ในการเขียนข้อมูลจาก CPU ไปยังอุปกรณ์ ทั้ง 2 คำสั่งเป็นฟังก์ชั่นที่สร้างขึ้นใช้สำหรับชุดฝึกทดลองโดยเฉพาะ ซึ่งจะอยู่ในไฟล์ edl.h ดังนั้นในโปรแกรมภาษาซีที่ต้องการใช้คำสั่งทั้งสองจะต้องมีประโยค

```
#include <edl.h>
```

โปรแกรมที่ใช้ในการทดสอบเมนบอร์ดมีดังนี้

```
#include<edl.h>
main()
{
        while(1)
        {
                in(0x00);
                in(0xff);
        }
}
```

รูปที่ ก.1.3 โปรแกรมที่ใช้ในการทดสอบความเร็วในการอ่านข้อมูล

```
#include<edl.h>
main()
{
        while(1)
        {
                out(0x00,0xff);
                out(0xff,0x00);
        }
}
```

รูปที่ ก.1.4 โปรแกรมที่ใช้ในการทดสอบความเร็วในการเขียนข้อมูล

## 3. อุปกรณ์การทดลอง

1. โปรแกรม TURBO C Version 2.0
2. เครื่องคอมพิวเตอร์ IBM PC
3. ชุดทดลอง
4. ออสซิลโลสโคป

## 4. ขั้นตอนการทดลอง

1. อ่านคู่มือการติดตั้งชุดทดลองและทำการติดตั้งชุดฝึกทดลองตามคู่มือ
2. ทดลองหา timing diagram ของสัญญาณบนระบบบัสของเมนบอร์ด โดยใช้โปรแกรมทดสอบที่ให้มา ใช้ CRO จับสัญญาณเพื่อดูช่วงเวลาต่างๆของสัญญาณในขณะทำคำสั่ง IN และ OUT
3. วาดรูป timing diagram ของสัญญาณเหล่านั้น
4. ต่อมอดูลสวิตช์เข้ากับเมนบอร์ด
5. ทดลองรันโปรแกรมตัวอย่าง EX1.EXE ในแผ่นดิสค์

การทดลองที่ 2 : การเขียนโปรแกรมด้วยภาษาซี

1. วัตถุประสงค์

1. เพื่อเรียนรู้โครงสร้างและคำสั่งที่สำคัญๆของภาษาซี
2. เพื่อเรียนรู้การเขียนโปรแกรมภาษาซีและการทดสอบแก้ไขโปรแกรม

2. ทฤษฎี

อ่านคู่มือการเขียนโปรแกรมภาษาซี และคู่มือการใช้เทอร์โบซี

3. อุปกรณ์การทดลอง

1. เครื่องคอมพิวเตอร์ IBM PC
2. โปรแกรม Turbo C Version 2.0

4. หนังสืออ่านประกอบ

1. Turbo C User's Guide[19]
2. Turbo C Reference Guide[20]
3. การโปรแกรมภาษาซี[26]

5. ขั้นตอนการทดลอง

1. ทดลองเขียนโปรแกรมตัวอย่าง ทำการคอมไพล์และรันดูว่าโปรแกรมทำหน้าที่อะไร
2. เขียนโปรแกรมรับค่าอุณหภูมิหน่วยเป็น องศา C ทางคีย์บอร์ดคอมพิวเตอร์ และทำการเปลี่ยนหน่วยให้เป็น องศา F แล้วแสดงผลบนหน้าจอคอมพิวเตอร์ โดยมีรูปแบบ ดังนี้

| องศา C | องศา F |
|---|---|
| 27 | ..... |
| 29 | ..... |
| 40 | ..... |

↑——ค่าที่ป้อนเข้าไป

3. เขียนฟังก์ชัน หาค่า $X^Y$ โดยมีรูปแบบ

POWER( X,Y ) , X = ฐาน , Y = กำลัง , X,Y เป็นจำนวนเต็ม

ทดลองหาค่าของ 2 ยกกำลัง 16

4 เขียนโปรแกรมวาดแกน X ,Y ในโมดกราฟฟิก และเขียนกราฟของสมการ

$Y = X^2$ ด้วยการวาดทีละจุดโดยใช้ฟังก์ชัน putpixel

- ทดลองเปลี่ยนสเกลของแกน X และ Y

5. เขียนฟังก์ชันที่ทำการเรียงค่าในตัวแปรชนิดจำนวนเต็ม จำนวน 4 ตัว โดยเรียงจากน้อยไปหามาก

รูปแบบ : Sort( a , b , c , d )

ตัวอย่าง : ก่อนเรียกฟังก์ชัน

a = 7 , b = 6 , c = 9 , d = 2

หลังจากเรียกฟังก์ชัน

a = 2 , b = 6 , c = 7 , d = 9

6. เขียนโปรแกรมสร้างรูปเลข 0 ขนาดใหญ่กลางจอภาพ โดยนำตัวอักษรเลข O มาเรียงต่อกัน หลังจากนั้นเขียนโปรแกรมให้ทำการเลื่อนเลขศูนย์ไปทางซ้ายมือของจอภาพ โดยใช้วิธีเขียนอ่านกับ VIDEO RAM โดยตรง

โปรแกรมตัวอย่าง

```
/*********************************
*      FIRST C PROGRAM          *
*    C EXAMPLE PROGRAM          *
*********************************/
#include <stdio.h>

#define IN  1
#define OUT 0

/* count lines , words , and characters in input */
main( )
{
    int c, nl, nw, nc, state;

    state = OUT;
    nl = nw = nc = 0;
    while ((c = getchar( )) != '!') {
        ++nc;
        if (c == '\n')
            ++nl;
        if (c == ' ' || c == '\n' || c == '\t')
            state = OUT;
        else if (state == OUT) {
            state = IN;
            ++nw;
        }
    }
    printf("%d %d %d\n", nl, nw, nc);
}
```

การทดลองที่ 3: Digital I/O

1. วัตถุประสงค์

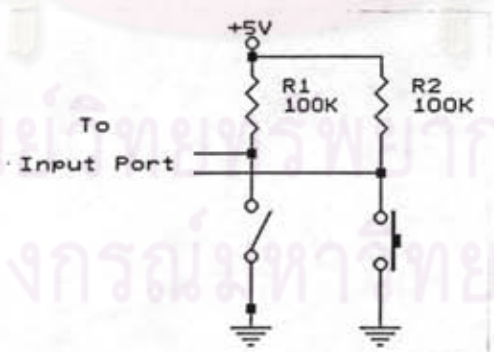
1. เพื่อเรียนรู้การรับส่งสัญญาณกับ Digital I/O ของคอมพิวเตอร์
2. เพื่อเรียนรู้การเขียนโปรแกรมตัวควบคุมแบบ Combinational
3. เพื่อเรียนรู้การเขียนโปรแกรมตัวควบคุมแบบ Sequential

2. ทฤษฎี

อุปกรณ์ประเภทดิจิทัลซึ่งรวมถึงระบบไมโครโปรเซสเซอร์ จะทำงานที่ระดับแรงดัน 2 สถานะ คือ สถานะ high คือระดับแรงดันที่มากกว่า 2 V (สำหรับTTL) และสถานะ low คือ ระดับแรงดันที่ต่ำกว่า 0.8 V ในการต่ออุปกรณ์ภายนอกเข้ากับระบบไมโครโปรเซสเซอร์โดยตรง อุปกรณ์ภายนอกจะต้องสามารถป้อนและรับระดับแรงดันดังกล่าวซึ่งเรียกว่าสัญญาณดิจิทัล

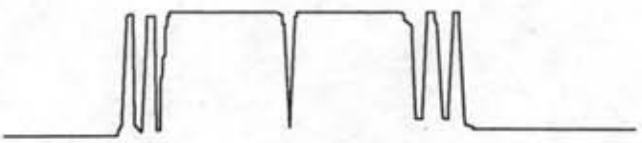
วงจรสวิตช์

ชุดฝึกทดลองมีวงจรสวิตช์เพื่อป้อนสัญญาณดิจิทัล ดังแสดงในรูป ก.2.1 R 100K ต่อไว้เป็น pull-up resistor ในขณะที่สวิตช์เปิดวงจร สัญญาณที่ป้อนให้แก่ input port จะมีระดับแรงดันเป็น +5 V หรือมีสภาพเป็น high และเมื่อสวิตช์ปิดวงจร สัญญาณจะมีระดับแรงดันเป็น 0 V มีสถานะเป็น low




รูปที่ ก.2.1 วงจรสวิตช์

ปัญหาในการใช้งานสวิตช์ คือ การกระเด้งของสัญญาณ ถ้าเรานำสัญญาณที่ได้ในขณะสวิตช์เปลี่ยนตำแหน่งจาก low เป็น high และ high เป็น low มาแสดงจะมีลักษณะดังรูปที่ ก.2.2



รูปที่ ก.2.2 แสดงการกระเด้งของสัญญาณจากสวิตช์

ช่วงเวลาของการกระเด้งของสัญญาณมีค่าประมาณ 15 ms[14] ซึ่งถ้าเรานำสัญญาณดังกล่าวป้อนเข้าระบบไมโครโปรเซสเซอร์ เนื่องจากระบบไมโครโปรเซสเซอร์ทำงานได้เร็วมาก อาจจะทำให้ตรวจสอบผิดว่า มีการกดสวิตช์ เข้า-ออก หลายๆครั้ง จึงต้องทำการ debounce ซึ่งสามารถทำได้ 2 วิธี คือ debounce โดยฮาร์ดแวร์ และ debounce โดยซอฟต์แวร์ ในที่นี้จะขอกล่าวถึงเฉพาะวิธีหลัง

วิธีการ debounce ทำได้ดังแสดงในรูป ก.2.3


รูปที่ ก.2.3 แสดงวิธีการ debounce สวิตช์

ขั้นตอนการ debounce สัญญาณจากสวิตช์

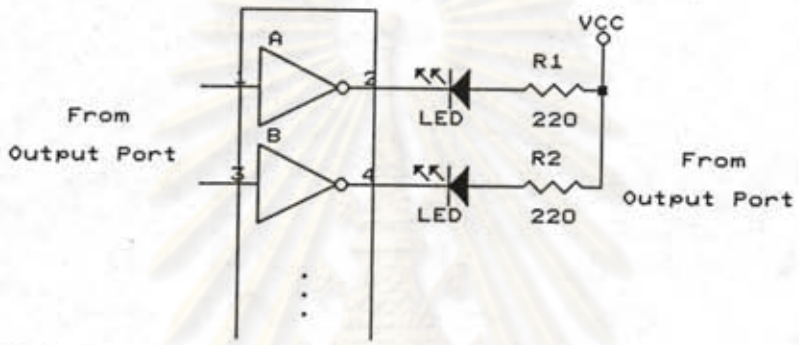
1 ตรวจสอบได้ว่ามีการเปลี่ยนสถานะของสัญญาณ
2 หน่วงเวลา
3 อ่านสถานะของสัญญาณซ้ำ สถานะที่อ่านได้จะเป็นตัวกำหนดสถานะของการกดสวิตช์จริง กล่าวคือ ถ้าสถานะคงเดิม(กับข้อ 1) แสดงว่ามีการเปลี่ยนตำแหน่งสวิตช์จริง ถ้าสถานะเปลี่ยนไป แสดงว่าการเปลี่ยนสถานะในข้อ 1 เป็น noise ที่เข้ามา
4 ตรวจสอบได้ว่ามีการเปลี่ยนสถานะของสัญญาณ
5 หน่วงเวลา

6 อ่านสถานะของสัญญาณซ้ำ ถ้าสถานะคงเดิม แสดงว่าการเปลี่ยนสถานะในข้อ 4 เป็น noise ที่เข้ามา แต่ถ้าสถานะเปลี่ยนไป แสดงว่ามีการเปลี่ยนตำแหน่งสวิตช์จริง

7-9 ลักษณะเช่นเดียวกับขั้นตอน 1-3

**วงจร LED**

เอาต์พุตของไอซีดิจิทัลโดยทั่วไปไม่สามารถขับกระแสสูงๆได้ จึงต้องมีวงจรบัฟเฟอร์ช่วย สำหรับชุดฝึกทดลองมีวงจรขับ LED ดังแสดงในรูปที่ ก.2.4

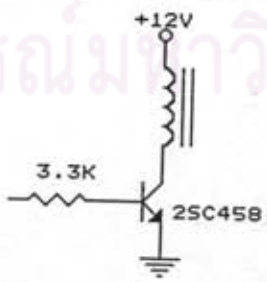



รูปที่ ก.2.4 วงจรขับ LED

จากวงจรในรูปที่ ก.2.4 เมื่อสัญญาณจาก output port มีสถานะเป็น high จะทำให้ LED สว่าง และเมื่อมีสถานะเป็น low จะทำให้ LED ดับ LED จึงใช้บอกสถานะของ output port ได้

**วงจรขับรีเลย์**

วงจรขับรีเลย์ของชุดฝึกทดลอง แสดงดังรูปที่ ก.2.5




รูปที่ ก.2.5 วงจรขับรีเลย์

เมื่อสัญญาณจาก output port มีสถานะเป็น high จะทำให้ 2SC485 นำกระแส ทำให้รีเลย์มีสถานะ ON และเมื่อ output port มีสถานะเป็น low จะทำให้ 2SC458 หยุดนำกระแส รีเลย์มีสถานะ OFF หน้าสัมผัสของรีเลย์สามารถนำไปต่อเพื่อควบคุมอุปกรณ์ไฟฟ้าต่าง ๆ

**วงจรขับทรานซิสเตอร์**

แสดงดังรูปที่ ก.2.6


รูปที่ ก.2.6 วงจรขับทรานซิสเตอร์

วงจรนี้สามารถนำไปขับวงจรที่กินกระแสสูงๆได้ โดย 2SC1061 สามารถทนกระแสได้ 3A

**การทำ Logic Operation**

คือ การหาค่าตัวแปรจากสมการ โดยใช้ตัวดำเนินการทางลอจิก และแทนค่าของตัวแปรขาเข้าด้วยค่าทางลอจิกโดยตรง

ตัวอย่างเช่น เราจะหาค่าของ d ตามสมการ

$d = a*b+c$

โดยที่ $a=0$ , $b=1$ , $c=0$

โดยการแทนค่า a,b,c ลงในสมการ จะได้ว่า

$d = 0*1+0$

$= 0+0 = 0$

**การใช้ Look-up Table**

คือ การหาค่าตัวแปรโดยการเปิดตาราง ซึ่งได้จากการคำนวณในทุกๆเงื่อนไขของตัวแปรขาเข้าไว้ก่อนแล้ว

ตัวอย่างเช่น การหาค่า d เช่นเดียวกับ หัวข้อที่ 5 ให้สร้างตารางดังแสดง นำค่า $a = 0$ , $b = 1$ , $c = 0$ ไปเปิดตารางจะได้ว่า $d = 0$

เมื่อใช้คอมพิวเตอร์ดำเนินการทั้ง 2 วิธี จะมีความเร็วและความยากง่ายที่แตกต่างกัน ทั้งนี้ขึ้นกับจำนวนตัวแปร และความซับซ้อนของสมการ

ตารางที่ ก.2.1 การสร้าง Look-up Table

| Input | | | Output |
|---|---|---|---|
| a | b | c | d |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

## 3. อุปกรณ์การทดลอง

1. IBM PC
2. โปรแกรม Turbo C Version 2.0
3. เมนบอร์ด
4. มอดูลสวิตช์
5. IC 74LS00
6. หลอดไฟ 6V
7. แหล่งจ่ายไฟ

## 4. หนังสืออ่านประกอบ

1. Turbo C Reference Guide[20]
2. การโปรแกรมภาษาซี[26]
3. คู่มือการเขียนโปรแกรมภาษาซี

5. ขั้นตอนการทดลอง

1. ต่อมอดูลสวิตช์เข้ากับเมนบอร์ด
2. เขียนโปรแกรมตัวอย่าง และ ทดลองรันดู
3. Combinational Controller

กำหนดให้ SW1-SW4 บนมอดูลสวิตช์เป็นตัวแปร X1-X4 ตามลำดับ และ LED1-LED4 เป็นตัวแปร Y1-Y4 ตามลำดับ

3.1 เขียนโปรแกรมที่ประมวลผลตามสมการบูลลีนดังนี้

Y1 = / X1

Y2 = ( X1*/ X2 ) + ( X1*X2 )

Y3 = X2 + X3*/ X4

Y4 = X1*X2*X3

3.2 โปรแกรมสามารถทำได้ 2 วิธี

ก) โดยใช้ Logical Operation ทำทีละบิท

ข) ใช้ Look-up Table โดยใช้ค่าอินพุตเป็น offset ในการชี้ Table เพื่ออ่านค่าเอาต์พุต

4. Sequential Controller

ต่อวงจรควบคุมการปิดเปิดหลอดไฟด้วยรหัสโดยใช้รีเลย์บนชุดทดลอง รีเลย์จะทำงานเมื่อกดรหัสที่ถูกต้องทาง SW1-SW4 กำหนดให้สวิตช์แต่ละตัวมีรหัสตรงกับหมายเลขของตัวสวิตช์เอง เช่น SW1 มีรหัสเป็น 1 SW2 มีรหัสเป็น 2 เป็นต้น เมื่อจะทำการปิด ให้กดรหัสใดก็ได้ ให้รหัสดังกล่าวเป็น 2-3-4

5. ต่อวงจรและเขียนโปรแกรมเพื่อทดสอบ IC 74LS00

โปรแกรมตัวอย่าง

```
/*****************************************************
* FILE_NAME : EXAMPLE.C                              *
* FUNCTION  : RUNNING LIGHT DISPLAYING               *
*****************************************************/

#include <edl.h>
#define LED 0xc0

delaya(long speed)
{  long count;
   for (count=0;count<=speed;count++);
}
main( )
{
int i = 1;
  while (c=getchar( )!=27)
  {
        out(LED,i);
        delaya(100000);
        i *= 2;
        if (i>=256) i=1;
  }
}
```

การทดลองที่ 4 : D/A Converter

วัตถุประสงค์

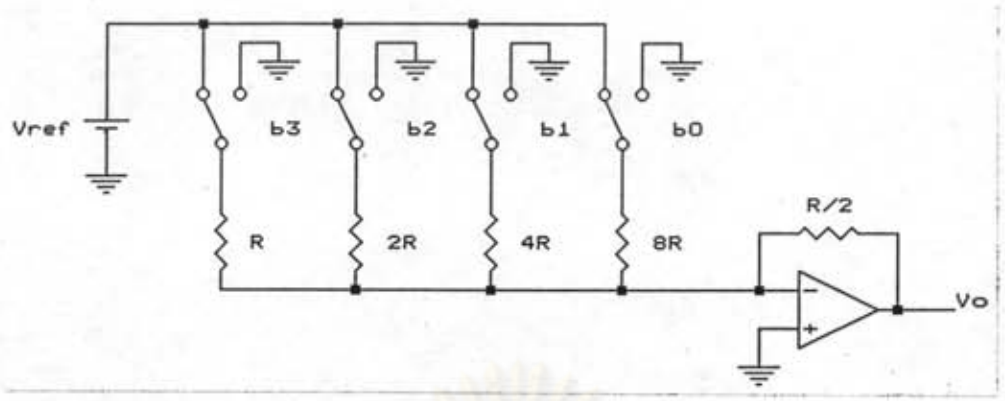
1. เพื่อพิจารณาคุณสมบัติต่างๆของวงจร D/A
2. เพื่อทดลองสร้างรูปสัญญาณด้วย D/A
3. เพื่อทดลองใช้คอมพิวเตอร์ในการควบคุมระดับแรงดัน

2. ทฤษฎี

ไมโครคอมพิวเตอร์สามารถส่งสัญญาณขาออกได้แต่เฉพาะที่เป็นสัญญาณดิจิทัลเท่านั้น แต่อุปกรณ์ไฟฟ้าส่วนใหญ่ที่ใช้ในชีวิตประจำวันรับสัญญาณที่เป็นสัญญาณแอนะลอก จึงไม่สามารถที่จะเชื่อมต่อกันได้โดยตรง จะต้องอาศัยอุปกรณ์ที่ทำหน้าที่แปลงสัญญาณดิจิทัลให้เป็นสัญญาณแอนะลอก เพื่อให้ไมโครคอมพิวเตอร์สามารถส่งสัญญาณไปควบคุมอุปกรณ์ต่างๆได้ อุปกรณ์ดังกล่าวเรียกว่า D/A (Digital to Analog Converter)

การแปลงสัญญาณจากดิจิทัลเป็นแอนะลอก

สัญญาณดิจิทัลซึ่งมีลักษณะเป็นตัวเลขจะถูกเปลี่ยนไปเป็นแรงดัน หรือกระแส ซึ่งมีขนาดเป็นสัดส่วนโดยตรงกับค่าของตัวเลข ตัวอย่างเช่น ถ้าค่าดิจิทัลที่มีค่าระหว่าง 0-255 ถูกเปลี่ยนไปเป็นแรงดันซึ่งมีช่วง 0-5.1 V ตัวเลข 0 จะแทนแรงดัน 0.00 V และตัวเลข 255 แทนแรงดัน 5.10 V ดังนั้นตัวเลข n จะแทนแรงดัน = 0.00+(n/255)*(5.10-0.00) V ถ้า n=1 จะแทนแรงดัน =0.02 V ค่า 0.02 V นี้เป็นค่าแรงดันที่น้อยที่สุดที่จะเกิดการเปลี่ยนแปลงได้ เราเรียกว่า ค่าความละเอียดของการแปลงสัญญาณจากดิจิทัลเป็นแอนะลอก(resolution) วงจร D/A สามารถแบ่งได้เป็น 2 ชนิดหลักๆ คือ แบบ weighted resistor network และแบบ R-2R ladder network แบบ weighted resistor network มีวงจรแสดงดังในรูปที่ ก.4.1




รูปที่ ก.4.1 D/A แบบ weighted resistor network

ซึ่งเราจะได้ว่า

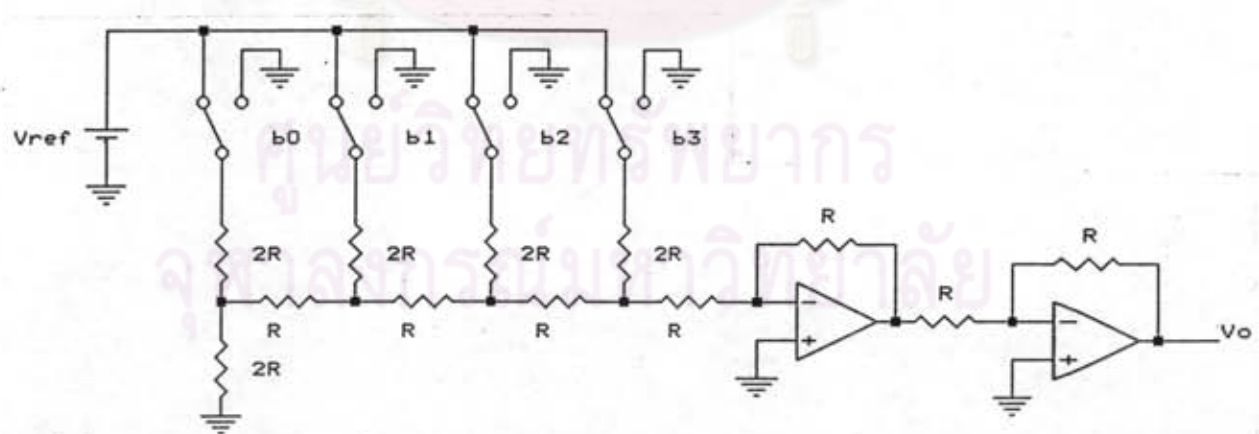
$$V_0 = V_{ref}( b_3/2 + b_2/4 + b_1/8 + b_0/16 )$$

$$= V_{ref}( b_3*2^{-1} + b_2*2^{-2} + b_1*2^{-3} + b_0*2^{-4} )$$

โดยที่ $b_0$ ถึง $b_3$ มีค่า 1 เมื่อสวิตช์ต่อกับ $V_{ref}$

มีค่า 0 เมื่อสวิตช์ต่อลงกราวนด์

จะเห็นว่าช่วงของ R ที่ใช้มีค่าตั้งแต่ $1\text{-}2^{n-1}$ เท่า โดย n เป็นจำนวนบิตของ D/A ในทางปฏิบัติมักจะใช้ D/A ขนาด 8 บิตหรือ 12 บิต นั่นคือ R มีค่าตั้งแต่ 1-128 เท่า(สำหรับ 8 บิต) หรือ 1-2048(สำหรับ 12 บิต) ซึ่ง R ดังกล่าวมีช่วงกว้างมาก ทำให้การผลิตให้ได้ความเที่ยงตรงสูงตลอดช่วง ทำได้ยาก วงจร D/A แบบ R-2R ladder network จะช่วยแก้ปัญหานี้ได้




รูปที่ ก.4.2 D/A แบบ R-2R Ladder

วงจรนี้ให้ค่า $V_0$ ดังสมการของวงจร weighted resistor network

โดยทั่วไปช่วงของแรงดันแอนะลอกขาออก(analog voltage range) มักมีค่าเป็น 5,10,5.12,10.24 V ตัวเลข 2 ค่าหลัง คือ 5.12 และ 10.24 V นั้นมีเพื่อทำให้ค่าเพิ่มของแรงดันมีค่าลงตัว

**การพิจารณา D/A ในการใช้งาน[14]**

สิ่งที่ต้องพิจารณามีดังนี้

1. Resolution คือ ความละเอียดในการแปลงสัญญาณ มักบอกเป็นจำนวนบิตของ D/A เช่น D/A ขนาด 8 บิต,12 บิต เป็นต้น

2. Slew rate คือ ค่าความชันของแรงดันขาออก เมื่อเปลี่ยนจากแรงดันต่ำสุดไปเป็นแรงดันเต็มพิกัด หน่วยเป็น V/μs

3. Settling time คือ เวลาที่ใช้ในการเปลี่ยนแรงดันขาออกจากจุดหนึ่งไปเป็นเศษส่วนค่าหนึ่งของแรงดันสุดท้าย หน่วยเป็น ns

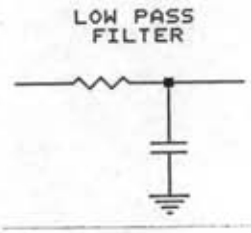
4. Linearity คือ ค่าเบี่ยงเบนออกจากเส้นตรงที่มากที่สุดของแรงดันขาออกเมื่อเขียนกราฟระหว่างค่าดิจิทัลและแรงดันขาออก

**วงจร D/A ในทางปฏิบัติ**

ในทางปฏิบัติ สัญญาณขาออกของ D/A จะนำไปเข้าวงจร signal conditioning เพื่อปรับสัญญาณให้อยู่ในช่วงแรงดันที่ต้องการ เช่น แรงดันขาออกของ D/A อยู่ในช่วง 0-5 V แต่เราต้องการเปลี่ยนช่วงเป็น (-5)-5 V ซึ่งสามารถทำโดยการขยายสัญญาณเป็น 2 เท่า และปรับ offset โดยการบอกแรงดัน offset -5 V เข้าไป วงจร signal conditioning จึงประกอบด้วย 2 ส่วน คือ ส่วนของการขยายสัญญาณ ซึ่งปรับอัตราขยายได้ และส่วนของการปรับ offset สำหรับวงจร D/A ของชุดฝึกทดลองสามารถปรับช่วงของแรงดันขาออกได้ประมาณ 0V ถึง10.2V ตัวปรับ offset มีช่วง 0 ถึง 12 V วงจร D/A อยู่ที่พอร์ตหมายเลข C6h ของชุดฝึกทดลอง

**การนำ D/A มาสร้างรูปคลื่นสัญญาณ**

ทำโดยการป้อนข้อมูลดิจิทัลให้แก่ D/A ด้วยลำดับและความเร็วที่เหมาะสมตามรูปคลื่นสัญญาณที่ต้องการ ความเร็วจะมีผลต่อความถี่ของสัญญาณที่ได้ รูปคลื่นสัญญาณที่ได้จะมีลักษณะเป็นขั้นบันได ถ้าต้องการปรับให้สัญญาณเรียบขึ้น ทำได้โดยการนำสัญญาณไปผ่านวงจรกรอง(Filter) ดังแสดงในรูปที่ ก.4.3




รูปที่ ก.4.3 การปรับสัญญาณให้เรียบ

## 3. อุปกรณ์การทดลอง

1. IBM PC
2. โปรแกรม Turbo C Version 2.0
3. เมนบอร์ด
4. มอดูล ADAC
5. ดิจิตอลโวลต์มิเตอร์
6. ออสซิลโลสโคป
7. แหล่งจ่ายไฟ

## 4. หนังสืออ่านประกอบ

1. Turbo C Reference Guide[20]

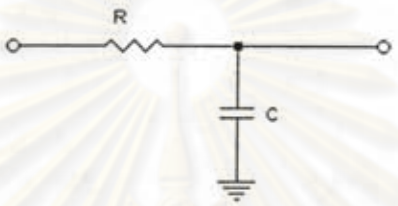
## 5. ขั้นตอนการทดลอง

1. ต่อมอดูล ADAC เข้ากับเมนบอร์ด
2. เขียนโปรแกรมทดลองเพื่อส่งค่าให้ D/A เพื่อผลิตแรงดันขาออก หลังจากนั้นทดลองปรับ Gain และ offset ของวงจรเพื่อให้ D/A กำเนิดแรงดันในช่วง 0.00-5.10V
3. ทดลองหาค่าแรงดันขาออกของค่าดิจิตอล 20 ค่า นำมาเขียนกราฟระหว่างค่าทั้งสองเพื่อหา Linearity ของ D/A
4. เขียนโปรแกรมเพื่อสร้างสัญญาณรูปซายน์ โดยให้มีจำนวนตัวอย่าง 20 ตัวอย่างใน 1 คาบ กำหนดให้มีความถี่เท่ากับ 1 kHz และมีขนาด 5 $V_{pp}$ ใช้ CRO จับดูรูปคลื่นของสัญญาณ

(แนะนำ: คำนวณค่าของฟังก์ชันซายน์ก่อน แล้วนำไปเก็บไว้ในตัวแปรอาร์เรย์จนครบ แล้วจึงนำค่าในตัวแปรอาร์เรย์มาใช้)

5. ทดลองปรับสัญญาณให้เรียบขึ้น โดยนำสัญญาณไปผ่านวงจรกรองผ่านต่ำ (Low-pass filter) ที่มีวงจรดังแสดงในรูป ทดลองเปลี่ยนค่า R,C เพื่อให้ได้รูปสัญญาณที่ดีขึ้น

- วงจรกรองผ่านต่ำมีความถี่ตัดที่ความถี่เท่าไร




รูปที่ ก.4.4 วงจรกรองผ่านต่ำ

การทดลองที่ 5 : A/D Converter

1. วัตถุประสงค์

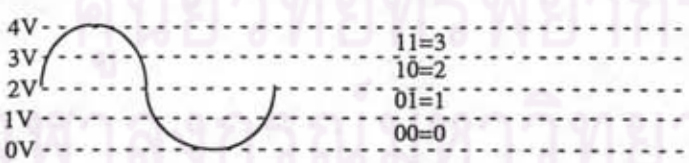
1. เพื่อพิจารณาคุณสมบัติของ A/D
2. เพื่อทดลองนำ A/D ไปใช้ในการวัดสัญญาณเสียง แสง อุณหภูมิ
3. เพื่อทดลองนำ A/D ไปใช้ในการบันทึกเสียง

2. ทฤษฎี

A/D (Analog to Digital Converter) เป็นอุปกรณ์ที่ทำหน้าที่แปลงสัญญาณแอนะลอก ซึ่งเป็นสัญญาณเอาต์พุตจากอุปกรณ์ส่วนใหญ่ให้เป็นสัญญาณดิจิทัลที่เครื่องคอมพิวเตอร์สามารถรับและเข้าใจได้ เพื่อทำการประมวลผลข้อมูลและทำการควบคุมต่อไป ด้วยวิธีนี้จะทำให้เครื่องคอมพิวเตอร์สามารถติดต่อกับโลกภายนอกได้มากขึ้น เช่น การวัดอุณหภูมิ การวัดแรงดัน การวัดแสง เป็นต้น

**การแปลงสัญญาณจากแอนะลอกเป็นดิจิทัล**

สัญญาณแอนะลอกซึ่งมีลักษณะเป็นค่าต่อเนื่อง จะถูกนำมาแบ่งช่วงๆ และแต่ละช่วงจะถูกแทนตัวเลข 1 ค่า โดยเรียงจากน้อยไปหามาก วิธีนี้เราเรียกว่า การ quantization โดยวิธีนี้ทำให้เราสามารถแทนระดับของสัญญาณแอนะลอกได้ด้วยตัวเลข ซึ่งก็คือสัญญาณดิจิทัลที่สามารถส่งให้แก่เครื่องคอมพิวเตอร์นั่นเอง มีความผิดพลาดที่เกิดขึ้นจากการแปลงสัญญาณที่เราเรียกว่า quantization error อันเนื่องมาจากการต้องแทนสัญญาณแอนะลอกที่มีค่าต่างกันด้วยตัวเลขตัว




รูปที่ ก.5.1 แสดงการ quantization ของสัญญาณแอนะลอก

เดียวกัน ยกตัวอย่างเช่นในรูปที่ ก.5.1 สัญญาณแอนะลอกที่มีค่า 2.2 V กับ 2.8 V จะถูกแทนด้วยเลขฐานสอง 10 เหมือนกัน ทำให้เราไม่สามารถแยกความแตกต่างของระดับสัญญาณทั้งสองได้ อย่างไรก็ตามเมื่อเราแบ่งช่วงของการ quantization ให้ละเอียดขึ้น(จำนวนขั้นก็มากขึ้น

ด้วย) ก็จะทำให้เราสามารถแยกความแตกต่างของระดับสัญญาณได้ละเอียดขึ้น ซึ่งเราเรียกว่าความละเอียด(resolution) ของ A/D นั่นเอง

**วงจร A/D**

วงจร A/D สามารถแบ่งได้เป็นหลายประเภทตามลักษณะวิธีการแปลงสัญญาณ[6] ดังนี้

1. Dual slope A/D
2. Voltage-Frequency A/D
3. Flash A/D
4. Successive Approximation A/D

ในที่นี้จะไม่ขอกล่าวถึงรายละเอียดของแต่ละประเภท ผู้สนใจสามารถศึกษาได้จากหนังสือทั่วไปที่กล่าวถึง A/D เช่น [6,7,14]

**การพิจารณา A/D ในการใช้งาน**

มีสิ่งที่ต้องพิจารณาดังนี้

1. Conversion Time

คือ เวลาที่ A/D ใช้ในการแปลงสัญญาณโดยนับตั้งแต่เมื่อป้อนสัญญาณแอนะลอกทางอินพุตจนได้ค่าดิจิทัลทางเอาต์พุตที่ถูกต้อง conversion time นี้มีค่าแตกต่างกันขึ้นอยู่กับชนิดของ A/D และยังเป็นตัวกำหนดอัตราในการสุ่มสัญญาณสูงสุดของระบบด้วยตัวหนึ่ง

2. ความละเอียด (Resolution)

คือ ความละเอียดในการแปลงสัญญาณ ยิ่งมีจำนวนบิตมากก็ยิ่งละเอียดมาก จึงมักกำหนดความละเอียดเป็นจำนวนบิตของ A/D เช่น A/D ขนาด 8 บิต,12 บิต เป็นต้น

3. Linearity

คือ ความเป็นเชิงเส้นของ A/D

**วงจร A/D ในทางปฏิบัติ**

ไอซี A/D มักจะสามารถรับช่วงของสัญญาณขาเข้าได้ในขาวงจำกัดช่วงหนึ่ง เช่น 0.5 V แต่เนื่องจากสัญญาณที่เราต้องการป้อนเข้าอาจไม่อยู่ในช่วงดังกล่าว เช่น อยู่ในช่วง 2-8 V จึงต้องทำการปรับแต่งสัญญาณขาเข้าก่อนที่จะป้อนให้แก่ไอซี A/D หรือในกรณีที่ช่วงของสัญญาณที่ป้อนเข้าอยู่ในช่วงที่ไอซี A/D สามารถรับได้ เช่น 0-2 V แต่ช่วงดังกล่าวไม่เต็มพิกัดของไอซี A/D

ก็จะทำให้ได้ความละเอียดในการใช้งานต่ำลง วงจรปรับแต่งสัญญาณขาเข้า(Signal Conditioning Circuit)ก็จะทำหน้าที่แก้ปัญหาตรงนี้ด้วย วงจรดังกล่าวประกอบด้วย 2 ส่วน คือ ส่วนของการขยายสัญญาณซึ่งปรับอัตราขยายได้และส่วนของการปรับ offset สำหรับชุดฝึกทดลองตัวปรับอัตราขยายสามารถปรับได้ในช่วงตั้งแต่ 0.5 เท่าจนถึง 5.5 เท่า และแรงดัน offset สามารถปรับได้ตั้งแต่ -5V จนถึง +5V

**การทำงานของไอซี ADC0804**

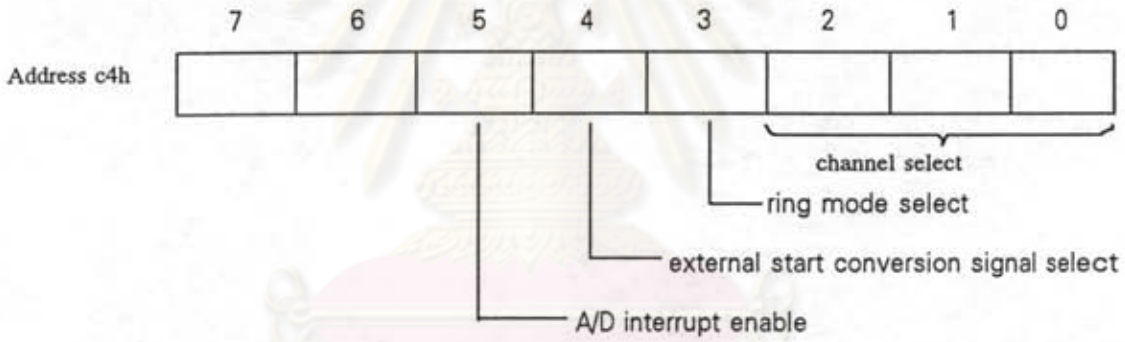
ไอซี A/D ที่ใช้ในชุดฝึกทดลอง คือ ADC0804 ซึ่งสามารถอินเตอร์เฟซกับไมโครคอมพิวเตอร์ และใช้ไมโครคอมพิวเตอร์ในการควบคุมได้โดยตรง มีวงจรสร้างสัญญาณนาฬิกาภายในเพื่อใช้ในระบบการแปลงสัญญาณ ซึ่งสามารถกำหนดความถี่ได้ด้วยการต่อ R,C ภายนอก ในชุดฝึกทดลองได้กำหนดความถี่ไว้ที่ประมาณ 820 KHz สามารถปรับแรงดันอ้างอิง(Vref) ได้โดยการป้อนแรงดันที่มีค่าเป็นครึ่งหนึ่งของแรงดันอ้างอิงที่ต้องการเข้าที่ขา Vref ขา $\overline{CS}$ เป็นสัญญาณ chip select ขา $\overline{RD}$ และ $\overline{WR}$ เป็นสัญญาณการเขียนอ่านข้อมูลจากไมโครคอมพิวเตอร์ ขา INTR เป็นสัญญาณบอกการสิ้นสุดการแปลงสัญญาณ การแปลงสัญญาณจะเริ่มต้นจากการส่งสัญญาณ $\overline{CS}$ มาเพื่อให้ชิปพร้อมที่จะทำงาน(ในที่นี้ต่อลงกราวด์ เพื่อให้ชิปพร้อมทำงานตลอดเวลา) หลังจากนั้นจะส่งสัญญาณเริ่มต้นการแปลง(start conversion) ซึ่งมีสถานะเป็น low เข้าที่ขา $\overline{WR}$ การแปลงสัญญาณจะเริ่มต้นที่ขอบขาขึ้นของสัญญาณ $\overline{WR}$ หลังจากนั้นจะใช้เวลาในการแปลงสัญญาณครู่หนึ่ง(คือ conversion time ของ A/D)จนเสร็จ ก็จะส่งสัญญาณสิ้นสุดการแปลงมาที่ขา INTR โดยจะมีสถานะเป็น low และค้างไว้จนกว่าจะมีการอ่านข้อมูลไปจาก ADC0804 เป็นอันจบการแปลงสัญญาณใน 1 รอบ การอ่านข้อมูลออกไปใช้งานทำโดยการส่งสัญญาณที่มีสถานะ low มาที่ขา $\overline{RD}$ ซึ่งจะทำให้ ADC0804 ส่งข้อมูลออกมาที่เดต้าบัส และขา INTR จะกลับไปมีสถานะ high ขา INTR นี้สามารถใช้เป็นสัญญาณเพื่ออินเตอร์รัปต์ไมโครคอมพิวเตอร์ได้ เราสามารถทำให้การแปลงสัญญาณมีลักษณะเป็นวงรอบได้ คือ เมื่อสิ้นสุดการแปลงสัญญาณใน 1 รอบ ก็จะส่งสัญญาณไปเริ่มต้นการแปลงในรอบถัดไปโดยอัตโนมัติ และวนซ้ำเช่นนี้ไปเรื่อยโดยไม่ต้องส่งสัญญาณมาทำการเริ่มต้นการแปลงอีก ซึ่งสามารถทำได้โดยการต่อขา INTR เข้ากับขา $\overline{WR}$ เมื่อสิ้นสุดการแปลงแต่ละรอบ INTR จะมีสถานะเป็น low และเมื่อมีการอ่านข้อมูลไปจาก A/D ก็จะทำให้ INTR เปลี่ยนสถานะจาก low เป็น high เป็นการส่งสัญญาณขอบขาขึ้นไปกระตุ้น $\overline{WR}$ เพื่อเริ่มต้นการแปลงในรอบถัดไป เราเรียกการทำงานแบบนี้ว่า ring mode อย่างไรก็ตามในรอบแรกของการแปลงสัญญาณ อาจไม่มีสัญญาณเพื่อไปเริ่มต้น

การแปลง ในชุดฝึกทดลองจึงได้ต่อสวิตช์ S1 ไว้เพื่อสร้างสัญญาณเริ่มต้นการแปลงสัญญาณในรอบแรกโดยการกดสวิตช์ S1

**การโปรแกรม A/D**

ในชุดฝึกทดลอง มอดูล A/D จะอยู่ที่พอร์ตหมายเลข C4h-C5h โดยพอร์ตหมายเลข C4h เป็น control register ใช้ในการเลือกช่องสัญญาณการแปลง โมดการแปลง สัญญาณการแปลงจากภายนอก และการเอนาเบิล-ดิสเอเบิลสัญญาณอินเตอร์รัปต์ ซึ่งส่งไปยัง $\overline{IRQ7}$ ของชุดฝึกทดลอง control register มีลักษณะดังแสดงในรูปที่ ก.5.2 การโปรแกรม control register ทำได้โดยการเขียนข้อมูลมาที่พอร์ตหมายเลข C4h นี้ โดยใช้คำสั่ง OUT เช่น

```
OUT(0XC4,0X08);
```




รูปที่ ก.5.2 แสดง control register ของ A/D

เป็นการโปรแกรมให้ A/D เลือกแชนแนล 0 ทำงานใน ring mode และดิสเอเบิลการอินเตอร์รัปต์ ส่วนพอร์ตหมายเลข C5h เป็น data register ของ A/D การอ่านข้อมูลไปจาก A/D ทำโดยใช้คำสั่ง IN เพื่ออ่านข้อมูลไปจากพอร์ต C5h ดังตัวอย่าง

```
data = IN(0XC5) ;
```

**3. อุปกรณ์การทดลอง**

1. เครื่องคอมพิวเตอร์ IBM PC
2. โปรแกรม Turbo C Version 2.0

3. เมนบอร์ด
4. มอดูล ADAC
5. มอดูลขยายเสียง
6. แหล่งจ่ายไฟ

## 4. หนังสืออ่านประกอบ

1. Turbo C Reference Guide[20]
2. Roger L. Tokheim , Digital Electronics

## 5. ขั้นตอนการทดลอง

1. ต่อมอดูล ADAC เข้ากับเมนบอร์ด
2. โปรแกรม Control register ของ A/D ให้ทำงานใน ring mode
3. ป้อนแรงดันช่วง 0.00-5.00 V เข้าทาง A_IN2 ของ A/D ทดลองปรับ $V_{ref}$ ที่ R13 จนอ่านค่าดิจิตอลได้ 0-255 (ฐานสิบ)
4. เขียนโปรแกรมอ่านค่าจาก A/D เพื่อรับสัญญาณอินพุตจาก VR , Microphone , LM 335 , Photo TR , แล้วนำมาแสดงเป็นรูปคลื่นบนจอภาพโดย ใช้โมดกราฟฟิก (แสดงครั้งละ 1 สัญญาณ)
5. การบันทึกสัญญาณเสียงและเล่นกลับ
   - 5.1 ต่อสัญญาณ เอาต์พุตของ D/A เข้ากับอินพุตของ มอดูลขยายเสียง
   - 5.2 เขียนโปรแกรมเพื่อสุ่มสัญญาณจากไมโครโฟนเข้าไปเก็บไว้ในตัวแปร จำนวน 30,000 ตัวอย่าง
   - 5.3 ทำการเล่นกลับโดยใช้ D/A ทดลองฟังคุณภาพของเสียงที่ได้
   - 5.4 ทดลองเพิ่ม Delay time ในโปรแกรมเพื่อให้สามารถบันทึกเสียงและเล่นกลับได้นานขึ้น
   - 5.5 แก้ไขโปรแกรมให้สามารถบันทึกค่าตัวแปรที่สุ่มมาได้ลงในไฟล์และทำการอ่านขึ้นมาใช้งานได้

การทดลองที่ 6 : Timer/Counter

1. วัตถุประสงค์

1. เพื่อเรียนรู้วิธีการใช้งาน Timer/Counter
2. เพื่อเรียนรู้การประยุกต์ใช้ Timer/Counter ในการวัดคาบของสัญญาณ กำเนิดสัญญาณที่มีคาบเวลาที่ต้องการ การวัดความถี่ของสัญญาณ และอื่นๆ

2. ทฤษฎี

Timer/Counter หรือที่เรียกย่อๆว่า T/C สามารถนำมาประยุกต์ใช้งานได้มากมาย เช่น การนับจำนวนเหตุการณ์ การสร้างฐานเวลาให้แก่ไมโครคอมพิวเตอร์ การวัดคาบสัญญาณ หรือการสร้างสัญญาณพัลส์ T/C ช่วยลดภาระของ CPU ในการที่จะต้องมาทำงานต่างๆเหล่านี้ ปัจจุบันเครื่องคอมพิวเตอร์ IBM PC มีรุ่นต่างๆมากมาย และมีความเร็วในการทำงานแตกต่างกัน เราสามารถประยุกต์ใช้ T/C ช่วยในการเขียนโปรแกรมทำให้โปรแกรมมีความเร็วในการทำงานคงที่ โดยไม่ขึ้นกับความเร็วของเครื่อง IBM PC ทำให้สามารถนำโปรแกรมไปใช้กับเครื่องอื่นๆได้โดยไม่ต้องแก้ไขโปรแกรมมอดูล T/C ของชุดฝึกทดลองประกอบด้วย 2 ส่วนหลักๆ คือ

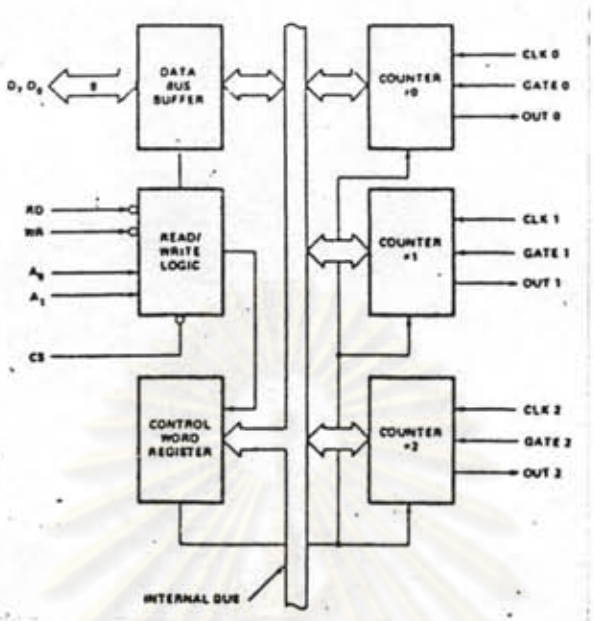
1. ส่วนของชิป T/C ใช้ชิปเบอร์ 8253 เป็นส่วนของ counter ทำหน้าที่ในการนับจำนวนพัลส์ที่เข้ามา มีโมดการนับที่สามารถโปรแกรมได้ โดยปกติ counter ของ T/C จะนับแบบถอยหลัง โดยค่าเริ่มต้นในการนับสามารถโปรแกรมได้ ในส่วนนี้จะมีสัญญาณเอาต์พุตซึ่งจะส่งสัญญาณตามเงื่อนไขของเหตุการณ์ต่างๆ ขึ้นอยู่กับโมดของการนับ

2. ส่วนกำเนิดสัญญาณนาฬิกา ทำหน้าที่สร้างสัญญาณนาฬิกาป้อนให้แก่ counter เพื่อใช้เป็นฐานเวลาในการนับ ทำให้เราสามารถทราบช่วงเวลาของเหตุการณ์ต่างๆได้ สัญญาณนาฬิกาที่สร้างขึ้นมักจะมีความถี่สูงเพื่อให้เกิดความละเอียดในการวัดช่วงเวลา อย่างไรก็ตาม สำหรับ T/C บางตัวสามารถโปรแกรมความถี่ของสัญญาณนาฬิกานี้ได้

โดยปกติ T/C จะมี counter หลายๆตัว เพื่อประกอบกันในการใช้งาน และสามารถอินเตอร์เฟซกับไมโครคอมพิวเตอร์ได้โดยตรง

ชิป 8253[9,13]

8253 เป็น programable interval timer(PIT) ประกอบด้วย counter 16 บิต 3 ตัวอิสระจากกัน(3 แชนแนล) สามารถนับพัลส์ที่มีความถี่สูงสุด 2.5 MHz บล็อกไดอะแกรมของ 8253 แสดงดังในรูปที่ ก.6.1




รูปที่ ก.6.1 บล็อกไดอะแกรมของ 8253

ขา CLK ทำหน้าที่รับสัญญาณนาฬิกาจากภายนอกเพื่อนำไปลดค่าของข้อมูลในเคาน์เตอร์ ขา GATE เป็นขาควบคุมการผ่านเข้าของสัญญาณที่ขา CLK ถ้าขา GATE มีสถานะเป็น low จะทำให้สัญญาณจากขา CLK ไม่สามารถผ่านเข้าไปยังเคาน์เตอร์เพื่อทำการลดค่าได้

**โมดการทำงานของ 8253**

ประกอบด้วย 6 โมด ดังนี้

1. โมด 0 : Interrupt on Terminal Count
2. โมด 1 : Programmable One Shot
3. โมด 2 : Rate Generator
4. โมด 3 : Square Wave Rate Generator
5. โมด 4 : Software Triggered Strobe
6. โมด 5 : Hardware Triggered Strobe

สำหรับรายละเอียดการทำงานของแต่ละโมด สามารถดูได้จาก Data Sheet ของ 8253 หรือ หนังสือทั่วไป เช่น [9,13]

การโปรแกรม 8253

การใช้งาน 8253 ทำโดยการโปรแกรมรีจิสเตอร์ภายใน ซึ่งประกอบด้วยรีจิสเตอร์ 4 ตัว ดังแสดงในตาราง ก.6.1

ตาราง ก.6.1 แสดงรีจิสเตอร์ภายใน 8253

| รีจิสเตอร์ | หมายเลขพอร์ตบนชุดฝึกทดลอง(HEX) |
|---|---|
| รีจิสเตอร์เคาน์เตอร์ แชนแนล 0 | F4 |
| รีจิสเตอร์เคาน์เตอร์ แชนแนล 1 | F5 |
| รีจิสเตอร์เคาน์เตอร์ แชนแนล 2 | F6 |
| รีจิสเตอร์ Mode Control | F7 |

การโปรแกรมจะเริ่มต้นจากการโปรแกรมรีจิสเตอร์ Mode Control ก่อน และตามด้วยรีจิสเตอร์เคาน์เตอร์ หน้าที่และความหมายของบิตต่างๆ ในรีจิสเตอร์ Mode Control แสดงดังในตาราง ก.6.2

ตาราง ก.6.2 แสดงหน้าที่ของบิตต่างๆในรีจิสเตอร์ Mode Control

| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|
| SC1 | SC0 | RL1 | RL0 | M2 | M1 | M0 | BCD |

| SC1 | SC2 | แชนแนลที่เลือก |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 2 |
| 1 | 1 | - |

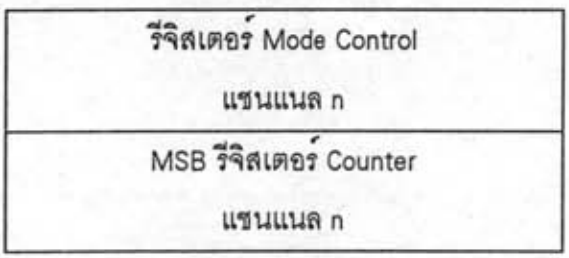
| RL1 | RL0 | ความหมาย |
|---|---|---|
| 0 | 0 | ทำการแลตช์ค่าในรีจิสเตอร์เคาน์เตอร์ |
| 0 | 1 | อ่าน/เขียนเฉพาะข้อมูลใน 8 บิตล่าง(Least-significant Bit) |
| 1 | 0 | อ่าน/เขียนเฉพาะข้อมูลใน 8 บิตบน(Most-significant Bit) |
| 1 | 1 | อ่าน/เขียนข้อมูลทั้ง 16 บิต โดยเริ่มจาก 8 บิตล่างก่อน จากนั้นจึงอ่าน/เขียนข้อมูลใน 8 บิตบน |

| M2 | M1 | M0 | โมดการทำงาน |
|---|---|---|---|
| 0 | 0 | 0 | โมด 0 : interrupt on Terminal Count |
| 0 | 0 | 1 | โมด 1 : Programmable One-shot |
| 0 | 1 | 0 | โมด 2 : Rate Generator |
| 0 | 1 | 1 | โมด 3 : Square Wave Generator |
| 1 | 0 | 0 | โมด 4 : Software Triggered Strobe |
| 1 | 0 | 1 | โมด 5 : Hardware Triggered Strobe |

| BCD | ความหมาย |
|---|---|
| 0 | ทำการลดค่าของข้อมูลในรีจิสเตอร์เคาน์เตอร์แบบ Binary |
| 1 | ทำการลดค่าของข้อมูลในรีจิสเตอร์เคาน์เตอร์แบบ BCD(Binary Coded Decimal) |

หลังจากโปรแกรมรีจิสเตอร์ Mode Control เรียบร้อยแล้ว ก็จะทำการโปรแกรมรีจิสเตอร์เคาน์เตอร์ จำนวนไบต์ของข้อมูลที่ส่งให้แก่รีจิสเตอร์เคาน์เตอร์อาจเป็น 1 ไบต์หรือ 2 ไบต์ ทั้งนี้ขึ้นอยู่กับค่าของ RL0,RL1 ที่ส่งให้แก่รีจิสเตอร์ Mode Control ในตอนแรกเราสามารถสรุปลำดับในการโปรแกรมรีจิสเตอร์ภายใน 8253 ได้ดังรูป ก.6.2


RL1=1,RL0=1




RL1=1,RL0=0

| รีจิสเตอร์ Mode Control<br>แชนแนล n |
| --- |
| LSB รีจิสเตอร์ Counter<br>แชนแนล n |

RL1=0,RL0=1

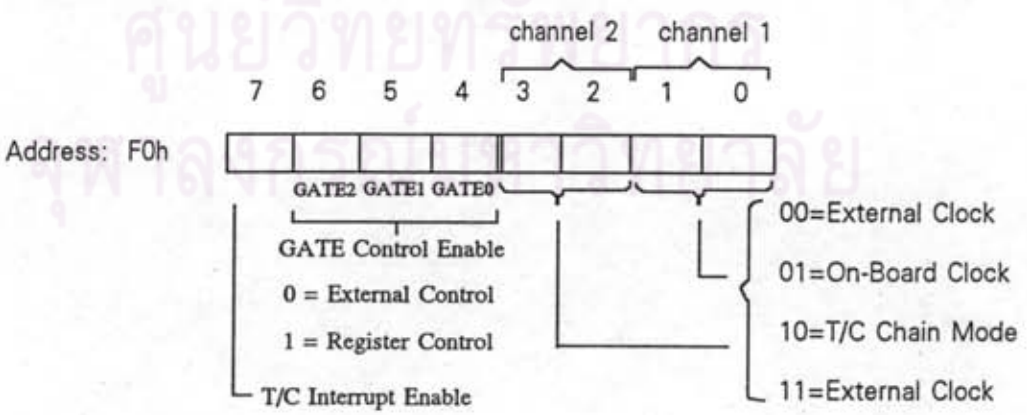
รูปที่ ก.6.2 ลำดับในการโปรแกรมรีจิสเตอร์ภายใน 8253

ในกรณีของ RL1=0,RL0=0 จะเป็นแลตช์ค่าในรีจิสเตอร์เคาน์เตอร์ จึงไม่ต้องทำการโปรแกรมรีจิสเตอร์เคาน์เตอร์ต่อจากรีจิสเตอร์ Mode Control เราสามารถเลือกโปรแกรมรีจิสเตอร์ของแชนแนลใดก่อนก็ได้ และหลังจากที่โปรแกรมรีจิสเตอร์ Mode Control ของแชนแนลใดแชนแนลหนึ่งแล้ว อาจทำการโปรแกรมรีจิสเตอร์ Mode Control ของแชนแนลอื่น ๆ ก่อน แล้วจึงกลับมาโปรแกรมรีจิสเตอร์ counter ของแชนแนลเดิม ทั้งนี้เราสามารถเลือกโปรแกรมรีจิสเตอร์ counter ของแชนแนลใดก่อนก็ได้เช่นกัน โดยไม่ขึ้นกับลำดับการโปรแกรมรีจิสเตอร์ Mode Control

## การโปรแกรมรีจิสเตอร์บนมอดูล ADAC

รีจิสเตอร์บนมอดูล ADAC ใช้ในการควบคุมการทำงานของ T/C และอ่านสถานะของขา OUT ของแต่ละแชนแนล รายละเอียดของรีจิสเตอร์แต่ละตัวมีดังนี้

1. T/C Control Register1



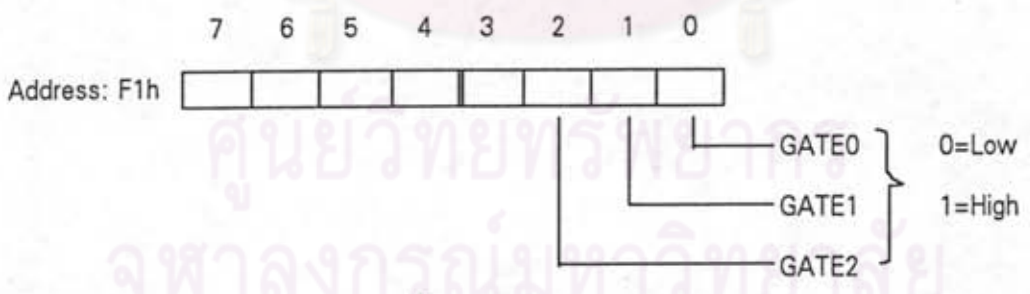

รูปที่ ก.6.3 T/C Control Register1

บิต 0-3 ใช้ในการเลือกแหล่งสัญญาณที่ป้อนให้แก่ขา CLK1-CLK2 ของ 8253 แบ่งออกเป็น 2 แชนแนล แชนแนลละ 2 บิต ถ้า 2 บิตนี้มีค่าเป็น 00 หรือ 11 จะเป็นการเลือกแหล่งสัญญาณจากภายนอกโดยป้อนเข้าที่ขั้ว CLK1 และ CLK2 บนมอดูล ถ้ามีค่าเป็น 01 จะเป็นการเลือกสัญญาณนาฬิกาที่สร้างขึ้นบนมอดูล T/C ซึ่งมีความถี่ 1.19318 MHz ถ้ามีค่าเป็น 11 จะเป็นการเลือกสัญญาณจากขา OUT ของแชนแนลถัดลงไป เช่น ถ้าบิต 1-0 มีค่าเป็น 10 (ฐานสอง) จะเป็นการต่อขา CLK1 เข้ากับขา OUT0 ถ้าบิต 3-2 มีค่าเป็น 10 (ฐานสอง) จะเป็นการต่อขา CLK2 เข้ากับขา OUT2

บิต 4-6 ใช้ในการเลือกแหล่งสัญญาณที่ป้อนให้แก่ขา GATE0-GATE2 ของ 8253 ถ้ามีค่าเป็น 0 จะเป็นการเลือกแหล่งสัญญาณจากภายนอกโดยป้อนเข้าที่ขั้ว GATE0-GATE2 บนมอดูล ถ้ามีค่าเป็น 1 จะเป็นการเลือกแหล่งสัญญาณจาก T/C Control Register2 ซึ่งทำให้สามารถควบคุมสถานะของขา GATE0-GATE2 ของ 8253 ได้ด้วยซอฟต์แวร์ บิต 7 เป็นบิตที่ใช้ในเอนาเบิลสัญญาณจากขา OUT2 ที่จะไปอินเตอร์รัพท์เครื่องคอมพิวเตอร์ ( ต่อเข้ากับ $\overline{IRQ7}$ )

2. T/C Control Register2

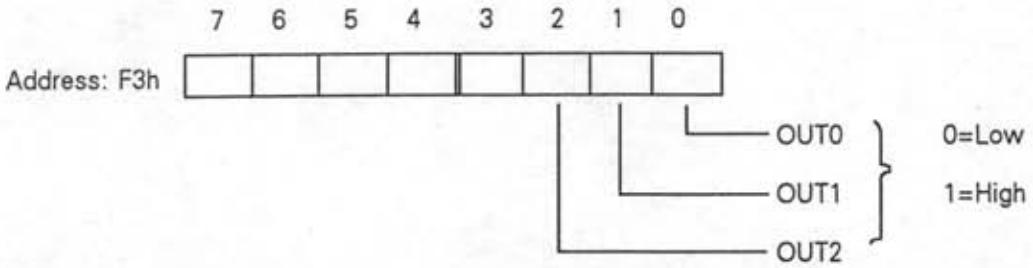
เป็นรีจิสเตอร์ที่ทำหน้าที่ป้อนสัญญาณให้แก่ขา GATE0-GATE2 ของ 8253 โดยจะต้องทำการเอนาเบิลที่ T/C Control Register1 ก่อน ( บิต 4-6 )




รูปที่ ก.6.4 T/C Control Register2

3. T/C Status Register

ใช้อ่านสถานะของขา OUT0-OUT2 ถ้ามีค่าเป็น 0 แสดงว่าอยู่ในสถานะ low ถ้ามีค่าเป็น 1 แสดงว่าอยู่ในสถานะ high




รูปที่ ก.6.5 T/C Status Register

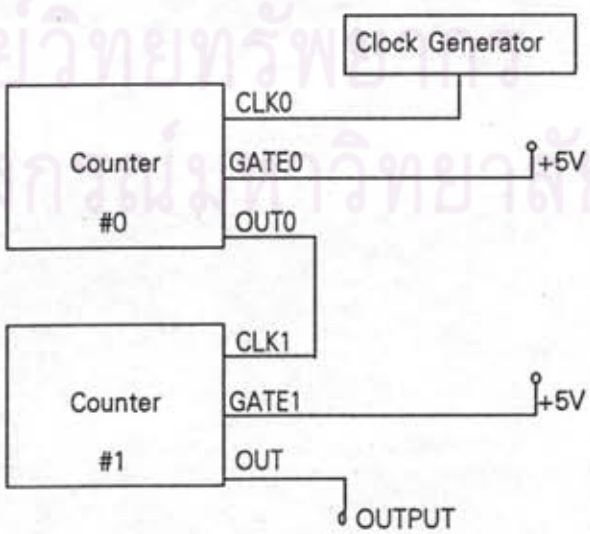
นอกจากนี้ยังมีพอร์ตหมายเลข F2h ซึ่งเป็นพอร์ตสัญญาณออกที่ทำหน้าที่เคลียร์สัญญาณอินเตอร์รัปต์ที่ส่งจากมอดูล T/C นี้ไปยังเครื่องคอมพิวเตอร์ ( ทำให้มีสถานะเป็น high ) การเคลียร์สัญญาณอินเตอร์รัปต์ทำได้โดยการเขียนข้อมูลที่มีค่าเท่าไรก็ได้ไปยังพอร์ตหมายเลขดังกล่าว

Address : F2h - clear interrupt signal

**ตัวอย่างการประยุกต์ใช้งาน T/C**

1. การกำเนิดสัญญาณนาฬิกาที่สามารถโปรแกรมความถี่ได้

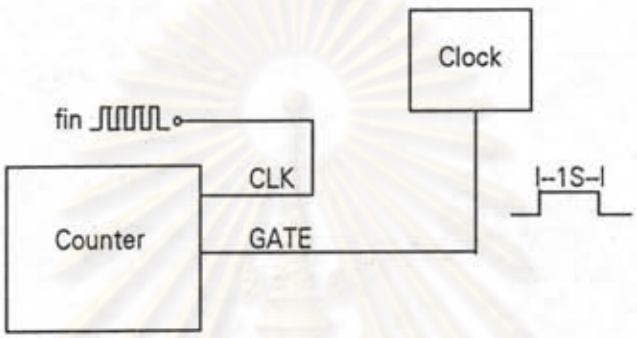
ทำโดยป้อนสัญญาณนาฬิกาเข้าที่ขา CLK ของ T/C และโปรแกรมให้ T/C ทำงานในโมด 3 เป็นการหารความถี่ของสัญญาณนาฬิกาขาเข้าด้วยค่าในรีจิสเตอร์เคาน์เตอร์ ซึ่งสามารถหารได้ตั้งแต่ 2-65535(ฐานสิบ) ในกรณีที่ต้องการตัวหารมากขึ้นทำโดยใช้หลายๆแซนแนลต่อแบบ Chain ดังแสดงในรูป




รูปที่ ก.6.6 การใช้ T/C ในการหารสัญญาณนาฬิกา

2. การวัดความถี่ของสัญญาณ

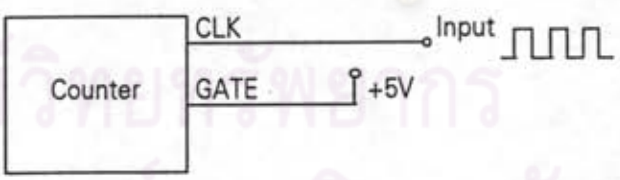
ความถี่ของสัญญาณหาได้จากจำนวนพัลส์ที่เข้ามาภายในช่วงเวลา 1 วินาที เราจะใช้เคาน์เตอร์ตัวหนึ่งเป็นตัวสร้างฐานเวลา 1 วินาที(อาจต้องใช้ 2 ตัวต่อ Chain กันในกรณีตัวหารความถี่ไม่เพียงพอ) และนำไปควบคุมการเปิด-ปิดเกตของเคาน์เตอร์ เพื่อผ่านสัญญาณเข้าไปลดค่าในเคาน์เตอร์ ค่าสุดท้ายในเคาน์เตอร์เมื่อนำไปหักลบกับค่าเริ่มต้น ก็จะทำให้ทราบจำนวนพัลส์ที่เข้ามาภายในเวลา 1 วินาที




รูปที่ ก.6.7 การวัดความถี่ของสัญญาณ

3. การนับจำนวนพัลส์

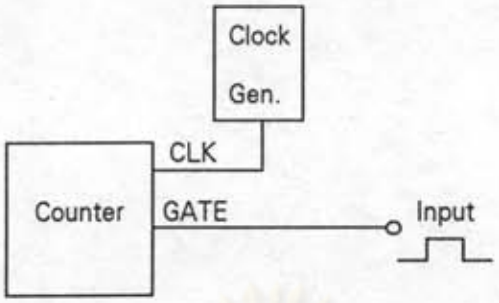
โดยการผ่านสัญญาณพัลส์เข้าไปที่ขา CLK ของ 8253 เพื่อไปลดค่าในเคาน์เตอร์ ค่าสุดท้ายในเคาน์เตอร์เมื่อหักลบกับค่าเริ่มต้น ก็จะทำให้ทราบจำนวนพัลส์ที่เข้ามา




รูปที่ ก.6.8 การนับจำนวนพัลส์

4. การวัดความกว้างของพัลส์

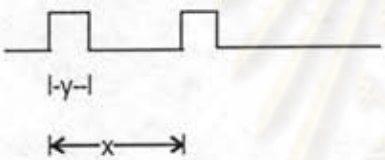
โดยการนำพัลส์ไปควบคุมการเปิดเกต และป้อนสัญญาณนาฬิกาที่ทราบความถี่เข้าที่ขา CLK เมื่อทราบค่าที่ลดลงของเคาน์เตอร์ก็จะสามารถคำนวณหาช่วงเวลาได้




รูปที่ ก.6.9 การวัดความกว้างของพัลส์

5. การกำเนิดพัลส์

สมมุติว่าต้องการสร้างพัลส์ที่มีลักษณะดังรูป




สามารถทำได้โดยใช้เคาน์เตอร์ 2 แชนแนล แชนแนลแรกสร้างสัญญาณนาฬิกาที่มีคาบเวลาเท่ากับ x แล้วนำสัญญาณไปกระตุ้นเคาน์เตอร์ตัวที่สองเพื่อสร้างพัลส์ที่มีความกว้างเท่ากับ x-y (ใช้โมด 1 ของ 8253)

## 3. อุปกรณ์การทดลอง

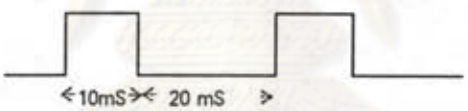
1. IBM PC
2. โปรแกรม Turbo C Version 2.0
3. เมนบอร์ด
4. มอดูล T/C
5. ออสซิลโลสโคป
6. IC 555 , R 10k , VR 10 k , C 0.1μ
7. ฟังก์ชันเจเนเรเตอร์
8. แหล่งจ่ายไฟ

## 4.หนังสืออ่านประกอบ

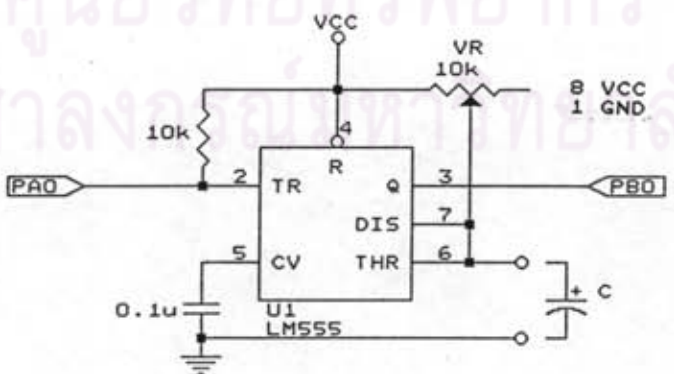
1. Turbo C Reference Guide[20]
2. การอินเทอร์เฟส IBM PC[9]

## 5. ขั้นตอนการทดลอง

1. ต่อมอดูล T/C เข้ากับเมนบอร์ด
2. ทดลองโปรแกรม 8253 ให้กำเนิดสัญญาณนาฬิกาที่มีความถี่ 1 kHz ทดลองใช้ CRO วัดดู
3. เตรียมสัญญาณ square wave จาก Function Gen. ขนาด 5Vp-p dc (สัญญาณดิจิตอล) ความถี่ระหว่าง 100 Hz - 10 kHz ต่อวงจรและเขียนโปรแกรมเพื่อวัดคาบสัญญาณดังกล่าว เปรียบเทียบค่าที่วัดได้กับค่าที่อ่านได้จาก CRO และทดลองเปลี่ยนความถี่ของสัญญาณขาเข้า
4. สร้างสัญญาณพัลส์ที่มีความกว้าง 10 ms ตามรูป ทดลองใช้ CRO วัดดู




5. สร้างเครื่องวัดค่า C โดยต่อวงจรดังรูป



การทำงานของวงจร

5.1 เริ่มต้นที่ให้ PA0 เป็นลอจิก 0 (พัลล์สั้นๆ) เพื่อกระตุ้น IC555 ซึ่งต่อแบบ mono stable multivibrator ซึ่งให้เอาต์พุตเป็นลอจิก 1 ที่ขา 3

5.2 ช่วงเวลาที่ขา 3 เป็นลอจิก 1 จะขึ้นกับค่า C ที่ต่ออยู่ หลังจากนั้นจะกลับเป็นลอจิก 0 ตามสมการ

$$t = 1.1*R*C$$

เราจะใช้ Timer/Counter ทำหน้าที่วัดช่วงเวลาดังกล่าว เมื่อทราบช่วงเวลาก็สามารถคำนวณหาค่าของ C ที่ต่ออยู่ได้

5.3 ทดลองนำ C ที่ทราบค่ามาต่อ และทดลองปรับค่า VR เพื่อให้อ่านค่าได้ถูกต้อง

## การทดลองที่ 7 : การอินเตอร์รัปต์ (Interrupt)

### 1. วัตถุประสงค์

1. เพื่อเรียนรู้การใช้งาน การอินเตอร์รัปต์ บน IBM PC
2. เพื่อเรียนรู้การประยุกต์ใช้งานการอินเตอร์รัปต์ในการควบคุม

### 2. ทฤษฎี

ลักษณะหนึ่งของการทำงานของเครื่องคอมพิวเตอร์ คือ ความสามารถตอบสนองงานที่ไม่ได้คาดเดาล่วงหน้าว่าจะเกิดขึ้นเมื่อไร ลักษณะของความสามารถนี้เรียกว่า การอินเตอร์รัปต์ ในขณะที่เครื่องคอมพิวเตอร์กำลังทำงานตามโปรแกรมอยู่ เมื่อเกิดการอินเตอร์รัปต์ขึ้น จะทำให้คอมพิวเตอร์หยุดพักงานที่กำลังทำอยู่ชั่วคราว และสับเปลี่ยนไปทำงานอื่นอันเป็นการตอบสนองการอินเตอร์รัปต์ งานนั้นเราเรียกว่า ตัวจัดการอินเตอร์รัปต์(Interrupt Handler) หรือโปรแกรมตอบสนองการอินเตอร์รัปต์(Interrupt Service Routine) ที่เรียกย่อๆว่า ISR ตัวอย่างของการอินเตอร์รัปต์ เช่น การกดปุ่มคีย์บอร์ด การอัปเดตเวลาภายในเครื่องคอมพิวเตอร์ การรับข้อมูลจากพอร์ตอนุกรม เป็นต้น งานที่ใช้ในการอินเตอร์รัปต์มักเป็นงานที่ไม่สามารถคาดเดาได้ว่าจะเกิดขึ้นเมื่อไรดังที่ได้กล่าวข้างต้น ในกรณีเช่นนี้ ถ้าไม่ใช้การอินเตอร์รัปต์เข้าช่วย ก็จะทำให้คอมพิวเตอร์ต้องคอยตรวจสอบอยู่ตลอดเวลา(polling) ซึ่งจะทำให้คอมพิวเตอร์เสียเวลาการทำงานไปแทนที่จะไปทำงานอื่นๆ ได้อย่างเต็มที่ เช่น จะต้องตรวจสอบคีย์บอร์ดอยู่ตลอดเวลาว่ามีการกดหรือไม่ ทำให้ต้องสูญเสียเวลาการทำงานส่วนหนึ่งไปโดยเปล่าประโยชน์

การอินเตอร์รัปต์สามารถแบ่งได้เป็น 2 ประเภท คือ

1. ฮาร์ดแวร์อินเตอร์รัปต์(Hardware Interrupt)
   คือ การอินเตอร์รัปต์ที่เกิดจากสัญญาณที่ส่งมาจากวงจรส่วนอื่นๆของคอมพิวเตอร์

2. ซอฟต์แวร์อินเตอร์รัปต์(Software Interrupt)
   คือ การอินเตอร์รัปต์ที่ถูกสร้างขึ้นจากโปรแกรม ตัวอย่างในเครื่อง IBM PC จะใช้คำสั่งภาษาแอสเซมบลี INT เพื่อเรียกการอินเตอร์รัปต์ชนิดนี้ ความจริงอินเตอร์รัปต์ชนิดนี้ไม่ใช่ลักษณะของอินเตอร์รัปต์ที่เกิดจากเหตุการณ์ที่ไม่ได้คาดเดาล่วงหน้าเหมือนกับฮาร์ดแวร์อินเตอร์รัปต์

## การอินเตอร์รัปต์บน IBM PC[9]

ภายใน IBM PC มีหลายส่วนของวงจรที่จำเป็นต้องใช้การอินเตอร์รัปต์ จึงได้จัดอินเตอร์รัปต์ไว้เป็นระดับต่างๆไว้ 9 ระดับ โดยมีลำดับความสำคัญ(Priority)ต่างๆกัน เหตุที่ต้องจัดลำดับความสำคัญของการอินเตอร์รัปต์ทั้ง 9 ระดับไว้ไม่เท่ากัน ก็เพื่อป้องกันกรณีที่มีการร้องขออินเตอร์รัปต์มากกว่า 1 แชนแนลในเวลาเดียวกัน เมื่อเกิดเหตุการณ์ในลักษณะเช่นนี้ แชนแนลที่มีลำดับความสำคัญสูงกว่าจะได้รับการตอบสนองก่อนโดยอัตโนมัติ การอินเตอร์รัปต์ทั้ง 9 ระดับสามารถแบ่งได้เป็น 2 ประเภท คือ นอน-มาสค์เคเบิล(Non-Maskable Interrupt : NMI) และ มาสค์เคเบิล(Maskable Interrupt : INT)

### 1. Non-Maskable Interrupt

เป็นอินเตอร์รัปต์ชนิดที่ไม่สามารถทำการดิสเอเบิลได้ และมีลำดับความสำสูลสุด เมื่อเกิดอินเตอร์รัปต์ชนิดนี้ขึ้น CPU จะหยุดพักจากงานต่างๆทั้งหมดมาทำงานในตัวจัดการอินเตอร์รัปต์ สัญญาณการร้องขออินเตอร์รัปต์ชนิดนี้จะถูกส่งเข้าที่ขา NMI ของ CPU อย่างไรก็ตาม เราสามารถจัดให้สัญญาณการร้องขออินเตอร์รัปต์ไม่ถูกส่งไปยังขา NMI ได้โดยใช้บิตมาสค์เข้าช่วย(เป็นวงจรภายนอก CPU) ซึ่งก็เท่ากับเป็นการดิสเอเบิลอินเตอร์รัปต์ชนิดนี้นั่นเอง บิตมาสค์นี้จะอยู่ที่บิต 7 ของพอร์ต 0A0h กรณีต้องการดิสเอเบิลการอินเตอร์รัปต์ชนิดนี้ทำโดยการเซ็ตบิตดังกล่าว การอินเตอร์รัปต์ชนิดนี้ใช้กับเหตุการณ์ที่มีความสำคัญมากๆที่ต้องการให้ CPU รีบไปจัดการ เช่น เมื่อเกิดความผิดพลาดในการทำงานของเครื่อง กรณีนี้ไม่สามารถปล่อยให้เครื่องสามารถทำงานต่อไปได้ หรือเมื่อสัญญาณ $\overline{\text{I/O CHCK}}$ เข้ามาที่สล็อตของ IBM PC(ขา A1) ของสก็อต เป็นต้น

### 2. Maskable Interrupt

เป็นการอินเตอร์รัปต์ชนิดที่สามารถดิสเอเบิลใช้ได้(Mask) สัญญาณการร้องขออินเตอร์รัปต์ชนิดนี้จะถูกส่งเข้าที่ขา INTR ของ CPU ใน IBM PC ชิปเบอร์ 8259 เข้าช่วยในการจัดอินเตอร์รัปต์ชนิดนี้ ทำให้สามารถแบ่งระดับการอินเตอร์รัปต์แบบ Maskable นี้ได้เป็น 8 ระดับ ในบางระดับถูกใช้งานโดย IBM PC อยู่แล้ว ตารางข้างล่างแสดงถึงการใช้งานอินเตอร์รัปต์ในระดับต่างๆของ IBM PC

| Interrupt Level | | Usage |
|---|---|---|
| Highest Level | NMI | Baseboard RAM parity, I/O channel check, numeric processor. |
| | IRQ 0 | System timer output 8253-5 Channel 0. |
| | IRQ 1 | Keyboard scan code interrupt. |
| Available in System Bus | IRQ 2 | Not used at present. |
| | IRQ 3 | Not used at present. |
| | IRQ 4 | RS-232-C serial port. |
| | IRQ 5 | Not used at present. |
| | IRQ 6 | Diskette DRV status. |
| | IRQ 7 | Parallel PRT port (not used in BIOS). |

ตารางที่ ก.7.1การใช้งานอินเตอร์รัปต์ของ IBM PC

การอินเตอร์รัปต์ชนิดนี้สามารถดิสเอเบิลได้โดยการรีเซ็ทแฟล็ก I (Interrupt Flag) ของ CPU ซึ่งจะเป็นการดิสเอเบิลการอินเตอร์รัปต์ชนิดนี้ทั้งหมด(ทั้ง 8 ระดับ) หรือดิสเอเบิลด้วยการโปรแกรมที่ 8259 ซึ่งสามารถเลือกได้ว่าจะดิสเอเบิลแชนแนลใด 8259 เป็น programmable interrupt controller ทำหน้าที่ในการจัดการสัญญาณการร้องขออินเตอร์รัปต์ทั้ง 8 ระดับก่อนที่จะส่งสัญญาณการร้องขออินเตอร์รัปต์ไปยัง CPU เนื่องจาก CPU มีขาที่ใช้รับสัญญาณแบบ maskable นี้เพียงขาเดียว คือ ขา INTR มีโมดการทำงานอยู่ 4 โมด คือ

1. โมด Fully Nested
2. โมด Rotating Priority
3. โมด Special Mask
4. โมด Polled

สำหรับรายละเอียดของการทำงาน และวิธีการโปรแกรม 8259 ให้ทำงานในโมดต่างๆ สามารถอ่านได้จากหนังสือ การอินเตอร์เฟส IBM PC [ 9 ] ในกรณีเครื่อง IBM PC 8259 จะถูกโปรแกรมให้ทำงานในโมด Fully Nested เป็นโมดที่จัดให้การร้องขออินเตอร์รัปต์ที่ผ่านเข้ามาทางแชนแนล0 มีลำดับความสำคัญสูงสุด และ แชนแนล1 ลดลำดับความสำคัญลง จนถึง แชนแนล 7 มีลำดับความสำคัญต่ำสุด ในกรณีที่มีการร้องขออินเตอร์รัปต์มากกว่า 1 แชนแนลในเวลาเดียวกัน แชนแนลที่มีลำดับความสำคัญสูงกว่าจะได้รับการตอบสนองก่อน

**ขั้นตอนการทำงานเมื่อเกิดอินเตอร์รัปต์**

1. วงจรอินเตอร์เฟซร้องขอการอินเตอร์รัปต์ โดยส่งสัญญาณมาที่ขา IR0-IR7 ของ 8259 (ขาใดขาหนึ่ง) สัญญาณการร้องขออินเตอร์รัปต์สามารถเลือกได้ว่าจะเป็นแบบ Level-Triggered หรือ Edge-Triggered โดยการโปรแกรมที่ 8259

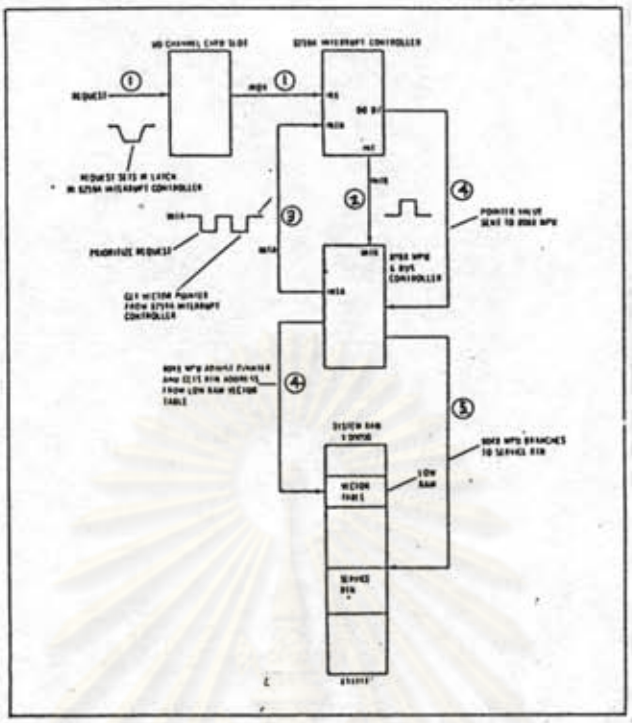
2. 8259 ทำการเซ็ตบิต IRR(Interrupt Request Register) ที่สัมพันธ์กับสัญญาณที่ร้องขอการอินเตอร์รัปต์ เช่น ถ้ามีการร้องขออินเตอร์รัปต์ที่ขา IR2 Interrupt Request 2 บิต IRR2 ก็จะถูกเซ็ต การเซ็ตบิต IRR ก็เพื่อเป็นการบอกว่ามีสัญญาณอินเตอร์รัปต์ผ่านเข้ามาที่ขา IR Interrupt Request หลังจากนั้น 8259 จะส่งสัญญาณร้องขออินเตอร์รัปต์ไปยัง CPU ที่ขา INTR

ในกรณีที่มีการร้องขออินเตอร์รัปต์ในขณะที่ CPU อยู่ในระหว่างการทำงานในโปรแกรมตอบสนองต่อการอินเตอร์รัปต์ที่เกิดขึ้นก่อนหน้านี้ 8259 จะทำการเปรียบเทียบลำดับความสำคัญของแชนแนลของสัญญาณการร้องขออินเตอร์รัปต์ที่เกิดขึ้นใหม่นี้กับแชนแนลของสัญญาณการร้องขออินเตอร์รัปต์เดิมที่ CPU กำลังตอบสนองอยู่ ถ้าพบว่าแชนแนลของของสัญญาณร้องขออินเตอร์รัปต์ที่เกิดขึ้นใหม่มีลำดับความสำคัญสูงกว่า 8259 ก็จะส่งสัญญาณไปอินเตอร์รัปต์ CPU แต่ถ้าแชนแนลของสัญญาณดังกล่าวมีลำดับความสำคัญต่ำกว่าหรือเท่ากับ 8259 จะรอจนกว่า CPU จะเสร็จสิ้นการทำงานในโปรแกรมตอบสนองการอินเตอร์รัปต์ จึงค่อยส่งสัญญาณไปอินเตอร์รัปต์ CPU

3. CPU ส่งสัญญาณ $\overline{\text{INTA}}$(Interrupt Acknowledge) มายัง 8259 โดยส่งเป็นพัลส์ 2 ลูก พัลส์ลูกแรกจะทำให้บิต IS(In-Service Register) ที่ตรงกับแชนแนลที่ 8259 จะทำการตอบสนองการอินเตอร์รัปต์ถูกเซ็ต เช่น ถ้า 8259 จะตอบสนองการอินเตอร์รัปต์ของแชนแนล 2 บิต IS2 ก็จะถูกเซ็ต การเซ็ตบิต IS ก็เพื่อเป็นการบอกว่าขณะนี้แชนแนลใดของการอินเตอร์รัปต์กำลังได้รับการตอบสนองอยู่ ในช่วงเวลานี้ 8259 ก็จะรีเซ็ตบิต IRR ที่ตรงกับบิต IS ที่ถูกเซ็ตด้วย

4. CPU ส่งพัลส์ $\overline{\text{INTA}}$ ลูกที่สองมายัง 8259 เพื่อให้ 8259 ทำกาส่งอินเตอร์รัปต์เวคเตอร์ออกมาบนบัสข้อมูล เพื่อให้ CPU นำไปเปิดตารางอินเตอร์รัปต์เวคเตอร์เพื่อหาตำแหน่งแอดเดรสเริ่มต้นของโปรแกรมการตอบสนองอินเตอร์รัปต์ เวคเตอร์ที่ส่งออกมาจะมีค่าแตกต่างกันไปตามแชนแนลที่ร้องขออินเตอร์รัปต์

5. CPU จะคำนวณค่าแอดเดรสเริ่มต้นของโปรแกรมตอบสนองการอินเตอร์รัปต์ แล้วทำการเก็บค่าของรีจิสเตอร์ CS,IP และ แฟล็ก ลงบนสแต็ก จากนั้นจึงทำการโหลดค่าแอดเดรสที่คำนวณได้ลงในรีจิสเตอร์ CS และ IP แล้วจึงทำงานต่อ ซึ่งเป็นผลทำให้ CPU กระโดดไปทำงานในโปรแกรมตอบสนองอินเตอร์รัปต์ บล็อกไดอะแกรมของขั้นตอนทั้ง 5 แสดงดังในรูป ก.7.2



รูปที่ ก.7.2 ขั้นตอนต่างๆในกระบวนการอินเตอร์รัปต์

จากขั้นตอนทั้งห้าที่กล่าวถึงนั้น เป็นขั้นตอนทั่วไปที่เกิดขึ้นในขบวนการอินเตอร์รัปต์ของระบบ อย่างไรก็ตามมีขั้นตอนบางประการที่ผู้อ่านควรจะทราบไว้ด้วย คือ

ในช่วงที่บิต IS ถูกเซ็ตในขั้นตอนที่ 3 นั้น ถ้ามีการร้องขออินเตอร์รัปต์ในช่วงดังกล่าว จะทำให้ 8259 ไม่ตอบสนองต่อการร้องขออินเตอร์รัปต์ในแชนแนลที่มีลำดับความสำคัญเท่ากับหรือต่ำกว่าแชนแนลที่บิต IS ถูกเซ็ต ไม่ว่า CPU จะเสร็จจากการทำงานในโปรแกรมตอบสนองต่อการอินเตอร์รัปต์แล้วหรือไม่ เช่น ในกรณีที่มีการร้องขออินเตอร์รัปต์ในแชนแนล 3 และ 8259 ทำการตอบสนองโดยการร้องขออินเตอร์รัปต์ไปยัง CPU แล้ว เมื่อ CPU พร้อมก็จะส่งพัลส์ลูกแรกของสัญญาณ $\overline{\text{INTA}}$ ออกมาให้กับ 8259 ในช่วงนี้บิต ISR ของแชนแนล 3 จะถูกเซ็ต เป็นผลทำให้การร้องขออินเตอร์รัปต์ในแชนแนลที่มีลำดับความสำคัญเท่ากับหรือต่ำกว่า(คือ แชนแนลที่ 3-7) ที่เข้ามาในช่วงนี้จะถูกเพิกเฉยไป เหมือนไม่มีการร้องขอเข้ามา จนกว่าบิต IS จะถูกรีเซ็ตเสียก่อน ซึ่งการรีเซ็ตบิต IS สามารถทำได้ 2 วิธี คือ

1.1 โปรแกรมให้ 8259 ทำงานในโมด AEOI(Automatic End of Interrupt)

จะทำให้ 8259 ทำการรีเซ็ตบิต IS โดยอัตโนมัติ เมื่อได้รับพัลส์ $\overline{\text{INTA}}$ลูกที่สองจาก CPU

1.2 ส่งคำสั่ง EOI(End of Interrupt)ให้กับ 8259

ลักษณะนี้จะทำให้เราสามารถกำหนดได้เองว่าจะรีเซ็ตบิต IS ในช่วงเวลาใด โดยทั่วไปแล้วมักจะทำการส่งคำสั่ง EOI ให้กับ 8259 ในช่วงท้ายของโปรแกรมตอบสนองการอินเตอร์รัปต์ การส่งคำสั่ง EOI ให้กับ 8259 ทำโดยการส่งค่า 20h ให้กับพอร์ตหมายเลข 20h

**ข้อควรระวัง:**

1. ในวิธีที่ 1.1 นั้น บิต IS จะถูกรีเซ็ตเมื่อ 8259 ได้รับพัลส์ $\overline{\text{INTA}}$ ลูกที่สอง ดังนั้นจะทำให้ 8259 ทำการตอบสนองการร้องขออินเตอร์รัปต์ซ้อนของแชนแนลที่มีลำดับความสำคัญเท่ากับหรือต่ำกว่าแชนแนลที่ร้องขออินเตอร์รัปต์อยู่ ถึงแม้ว่า CPU จะยังไม่เสร็จสิ้นการทำงานในโปรแกรมตอบสนองการอินเตอร์รัปต์ก็ตาม

2. ในขั้นตอนที่ 2 เมื่อ 8259 ส่งสัญญาณ INTR ให้แก่ CPU แล้ว ในขั้นตอนที่ 3 CPU จะส่งพัลส์ $\overline{\text{INTA}}$ ออกมาก็ต่อเมื่อ แฟล็ก I ถูกเซ็ตอยู่(เป็นการเอนาเบิลอินเตอร์รัปต์)

3. ในขั้นตอนที่ 1 เมื่อวงจรอินเตอร์เฟซส่งสัญญาณการร้องขออินเตอร์รัปต์ออกมาที่ขา IR ของ 8259 สัญญาณดังกล่าวจะต้องคงสภาพลอจิกไว้จนกว่า 8259 จะได้รับพัลส์ $\overline{\text{INTA}}$ ลูกแรก มิฉะนั้น 8259 จะถือว่าการร้องขออินเตอร์รัปต์นั้นเป็นการร้องขอผ่านทางแชนแนล 7 โดยไม่สนใจว่าการร้องขออินเตอร์รัปต์นั้นเกิดขึ้นที่แชนแนลใด

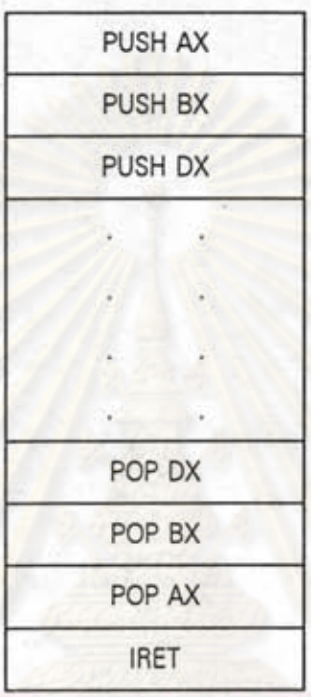
**โปรแกรมตอบสนองการอินเตอร์รัปต์**

เป็นส่วนของโปรแกรมที่ประกอบด้วยคำสั่งที่ทำหน้าที่ที่ต้องการเมื่อเกิดการอินเตอร์รัปต์ ในส่วนนี้จำเป็นจะต้องเพิ่มโปรแกรมบางส่วน เพื่อป้องกันความผิดพลาดที่เกิดขึ้นเมื่อ CPU กลับสู่การทำงานตามปกติ ส่วนของโปรแกรมที่เพิ่มเข้าไปมีดังนี้

1. ส่วนของการเก็บค่ารีจิสเตอร์ หรือตัวแปรต่างๆลงสแต็ก

เนื่องจากเราไม่สามารถคาดเดาได้ว่าการอินเตอร์รัปต์จะเกิดขึ้นเมื่อใด เมื่อ CPU กระโดดไปทำงานในโปรแกรมตอบสนองการอินเตอร์รัปต์อาจมีการเปลี่ยนแปลงค่าในรีจิสเตอร์ หรือตัวแปรต่างๆ และเมื่อกลับสู่การทำงานในโปรแกรมเดิมเนื่องจากค่าในรีจิสเตอร์ หรือตัวแปรถูกเปลี่ยนแปลงไป จึงอาจทำให้โปรแกรมทำงานผิดพลาดได้ ดังนั้นในส่วนของโปรแกรมตอบสนองการอินเตอร์รัปต์ จึงต้องมีการเก็บค่ารีจิสเตอร์ หรือตัวแปรต่างๆลงสแต็กก่อนที่จะมีการเปลี่ยนแปลงค่ารีจิสเตอร์เหล่านั้น และเมื่อทำงานเสร็จก็จะคืนค่าเหล่านั้นจากสแต็กให้แก่รีจิสเตอร์ดังเดิม ในส่วนของการเก็บค่าลงสแต็กมักจะอยู่ที่ในส่วนต้นของโปรแกรม และในส่วนของการคืนค่าจากสแต็กมักจะอยู่ส่วนท้ายของโปรแกรม ดังแสดงในรูป ก.7.3 อย่างไรก็ตามสำหรับการเขียนโปรแกรมด้วยภาษาสูง อย่างเช่น ภาษา C การผ่านค่าตัวแปรระหว่างฟังก์ชันเป็นการผ่านแบบผ่านค่า(pass by value) ทำให้การเปลี่ยนแปลงค่าตัวแปรในโปรแกรม ISR ไม่มีผลต่อการ

เปลี่ยนแปลงค่าตัวแปรในโปรแกรมที่ทำงานอยู่ และหากใช้ฟังก์ชันชนิด interrupt ค่ารีจิสเตอร์ต่างๆจะถูกเก็บลงสแต็กโดยอัตโนมัติ และจะคืนค่าโดยอัตโนมัติเมื่อเสร็จสิ้นการทำงานใน ISR ในกรณีที่เราต้องการให้มีการเปลี่ยนแปลงค่าตัวแปรโดยโปรแกรม ISR ทำโดยการประกาศชนิดตัวแปรเป็นแบบ global




รูปที่ ก.7.3 ส่วนประกอบของโปรแกรมตอบสนองการอินเตอร์รัปต์

2. ส่วนของการเอนาเบิล-ดิสเอเบิลอินเตอร์รัปต์

ในขณะที่ CPU กระโดดไปทำงานใน ISR แฟล็ก I ของ CPU จะถูกรีเซ็ตทำให้ CPU ไม่ตอบสนองต่อการอินเตอร์รัปต์ทุกชนิด แม้การอินเตอร์รัปต์ที่เข้ามาจะมีลำดับความสำคัญสูงกว่ายกเว้นชนิด NMI ดังนั้น ถ้าเราต้องการให้เกิดการอินเตอร์รัปต์ซ้อนขึ้น จะต้องทำการเซ็ตแฟล็ก I ในส่วนของโปรแกรม ISR ภาษาซีมีฟังก์ชันที่ทำหน้าที่เซ็ตแฟล็ก I คือ enable( ) และฟังก์ชันที่หน้าที่รีเซ็ตแฟล็ก I คือ disable( )

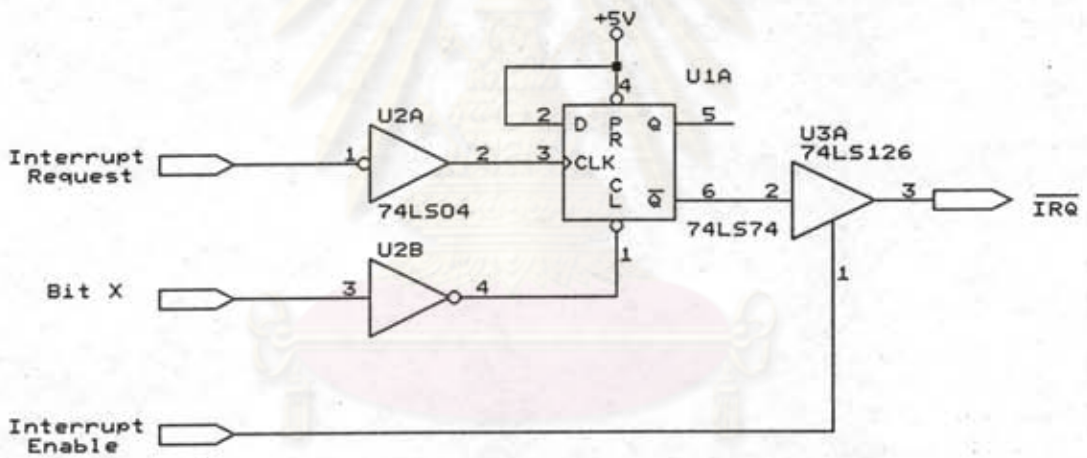
3. ส่วนของการส่ง EOI ให้กับ 8259

ดังที่ได้กล่าวมาแล้วในกรณีของ Hardware Interrupt เมื่อมีการอินเตอร์รัปต์เกิดขึ้นจะทำให้บิต IS ของ 8259 ถูกเซ็ต และทำให้ 8259 ไม่รับการร้องขออินเตอร์รัปต์ของแชนแนลที่มีลำดับความสำคัญต่ำกว่าหรือเท่ากับแชนแนลที่ขออินเตอร์รัปต์อยู่ ดังนั้นจะทำให้ไม่สามารถเกิดอิน

เตอร์รัปต์ในรอบต่อไปได้ จึงต้องทำการรีเซ็ตบิต IS ก่อน โดยส่งคำสั่ง EOI ไปให้ 8259(ส่งค่า20h ไปให้แก่พอร์ตหมายเลข 20h) ซึ่งมักจะอยู่ในส่วนท้ายของโปรแกรม ISR

4. ส่วนของการเคลียร์สัญญาณการร้องขออินเตอร์รัปต์

สัญญาณการร้องขออินเตอร์รัปต์ที่วงจรอินเตอร์เฟซสร้างขึ้นจะต้องมีความยาวนานพอที่จะทำให้ 8259 รับรู้อย่างถูกต้อง และควรจะต้องไม่ยาวจนเกินไปจนทำให้ CPU นึกว่ามีการร้องขอการอินเตอร์รัปต์อีกครั้งหนึ่ง อย่างไรก็ตามโดยทั่วไปเป็นการยากที่เราจะสร้างวงจรอินเตอร์เฟซให้มีคุณสมบัติดังกล่าวได้ ดังนั้นจึงได้ทำให้วงจรอินเตอร์เฟซสร้างสัญญาณร้องขออินเตอร์รัปต์ที่มีความยาวนานพอที่จะทำให้ 8259 รับรู้สัญญาณดังกล่าว และสร้างวงจรเพื่อช่วยเคลียร์สัญญาณการร้องขออินเตอร์รัปต์ไม่ให้มีความยาวนานเกินไป วงจรดังกล่าวแสดงดังในรูปที่ ก.7.4 การเคลียร์สัญญาณการร้องขออินเตอร์รัปต์จะกระทำในช่วงก่อนที่จะส่งคำสั่ง EOI ไปให้ 8259 โดยการส่งสัญญาณไปรีเซ็ตฟลิปฟลอปที่บิต X




รูปที่ ก.7.4 วงจรเคลียร์สัญญาณการอินเตอร์รัปต์

**ตารางการอินเตอร์รัปต์เวคเตอร์**

IBM PC ใช้หน่วยความจำ 1024 ไบต์ล่าง คือ จาก 00000h จนถึง 003ffh สำหรับเก็บแอดเดรสของโปรแกรมย่อยที่ตอบสนองต่อการอินเตอร์รัปต์ ซึ่งมีการอินเตอร์รัปต์ทั้งสิ้น 256 ชนิด และเนื่องจากการอ้างแอดเดรสในหน่วยความจำจะต้องกำหนดค่าให้กับรีจิสเตอร์ CS และ IP ดังนั้นหน่วยความจำที่ใช้เก็บค่าแอดเดรสของการอินเตอร์รัปต์แต่ละชนิดจะมีจำนวน 4 ไบต์ อินเตอร์รัปต์หมายเลข 0 จะถูกเก็บที่ตำแหน่ง 00000h หมายเลข 1 ถูกเก็บที่ตำแหน่ง 0004h และหมายเลข 2 ถูกเก็บที่ตำแหน่ง 0008h เรียงต่อไปเช่นนี้ จะเก็บค่าของไบต์ต่ำก่อน และเก็บค่าของ IP ก่อน จึงตามด้วยค่าของ CS

สำหรับฮาร์ดแวร์อินเตอร์รัปต์ จะมีหมายเลขอินเตอร์รัปต์ของการอินเตอร์รัปต์(Interrupt Number)ตั้งแต่หมายเลข 8 เป็นต้นไป โดยหมายเลข 8 จะเป็นของแชนแนล0 หมายเลข 9 เป็นของแชนแนล1 หมายเลข 10 เป็นของแชนแนล 2 เรียงต่อไปเช่นนี้ รูปที่ ก.7.5 แสดงการใช้ตารางอินเตอร์รัปต์เวคเตอร์ของ 8259 บน IBM PC

| HEX ADDRESS | | |
|---|---|---|
| 0003C | IRQ7 | TYPE 15 PARALLEL PRINTER PORT CARD |
| 00038 | IRQ6 | TYPE 14 DISKETTE ADAPTER CARD |
| 00034 | IRQ5 | TYPE 13 NOT USED |
| 00030 | IRQ4 | TYPE 12 SERIAL PORT CARD |
| 0002C | IRQ3 | TYPE 11 NOT USED |
| 00028 | IRQ2 | TYPE 10 NOT USED |
| 00024 | IRQ1 | TYPE 9 KEYBOARD |
| 00020 | IRQ0 | TYPE 8 TIMER COUNTER CHANNEL 0 |

รูปที่ ก.7.5 ตารางอินเตอร์รัปต์เวคเตอร์

**ขั้นตอนการเตรียมการอินเตอร์รัปต์**

สามารถสรุปได้ดังนี้

1. เขียนโปรแกรมตอบสนองการอินเตอร์รัปต์
2. จัดเตรียมเนื้อที่ในหน่วยความจำที่จะใช้เป็นสแต็ก
3. เปลี่ยนแปลงค่าในตารางอินเตอร์รัปต์ให้ชี้ไปยังโปรแกรมตอบสนองการอินเตอร์รัปต์ที่ต้องการ
4. ทำการโปรแกรม 8259
5. ทำการเอนาเบิลการอินเตอร์รัปต์ของ CPU โดยเซ็ตแฟล็ก I ของรีจิสเตอร์แฟล็ก

ในกรณีที่ใช้ภาษาสูง ขั้นตอนบางขั้นตอนอาจไม่จำเป็น เนื่องจากภาษาได้เตรียมการไว้ให้แล้ว

**3. อุปกรณ์การทดลอง**

1. เครื่องคอมพิวเตอร์ IBM PC
2. โปรแกรม Turbo C Version 2.0

3. เมนบอร์ด
4. แหล่งจ่ายไฟ
5. มอดูล A/D
6. มอดูล T/C

## 4. หนังสืออ่านประกอบ

1. คู่มือการใช้งานชุดฝึกทดลอง

## 5. ขั้นตอนการทดลอง

1. ทดลองเขียนโปรแกรมตัวอย่าง ทำการคอมไพล์และรันดูว่าโปรแกรมทำหน้าที่อะไร
2. ทดลองแก้ไขโปรแกรมตัวอย่าง เพื่อทำการเปลี่ยนแปลงค่า แอดเดรสในตารางอินเตอร์รัพต์เวคเตอร์ของเครื่องคอมพิวเตอร์ IBM PC ให้ชี้ไปยังฟังก์ชันที่เขียนขึ้น โดยทดลองกับอินเตอร์รัปต์หมายเลข 09h ซึ่งเป็นอินเตอร์รัปต์ของคีย์บอร์ด และให้มีการเก็บค่าแอดเดรสเก่าของอินเตอร์รัปต์ของรูทีนเดิมลงในตัวแปร และเมื่อออกจากโปรแกรม ให้ทำการคืนค่าแอดเดรสเก่าลงในตาราง ฟังก์ชันที่เขียนขึ้น ให้ทำหน้าที่นับจำนวนครั้งของการอินเตอร์รัปต์ที่เกิดขึ้น หลังจากนั้น ให้คืนการควบคุมไปยังอินเตอร์รัปต์รูทีนเดิมของคีย์บอร์ดโปรแกรมหลักให้ทำหน้าที่ในการพิมพ์จำนวนครั้งของการอินเตอร์รัปต์ที่เกิดขึ้นบนหน้าจอ
3. ต่อมอดูล A/D และมอดูล T/C เข้ากับเมนบอร์ด
4. โปรแกรมรีจิสเตอร์ควบคุมของมอดูล ADAC เพื่อให้ ADAC อยู่ในโมดการอินเตอร์รัปต์ รับสัญญาณ start conversion จากภายนอกและรับสัญญาณแอนะล็อกจากตัววัดอุณหภูมิ หลังจากต่อ ขา out ของแชนแนลใดแชนแนลหนึ่งของมอดูล T/C เข้ากับขา strat conversion ของ A/D
5. โปรแกรมรีจิสเตอร์ของ T/C เพื่อให้ T/C สร้างสัญญาณไปกระตุ้น A/D ให้เริ่มต้นการแปลงสัญญาณ ในทุกๆ 5 วินาที
6. เขียนโปรแกรมให้ทำการสุ่มสัญญาณผ่านทาง A/D ด้วยคาบเวลาที่แน่นอน คือทุกๆ 5 วินาที สัญญาณที่สุ่ม คือ สัญญาณอุณหภูมิ และนำไปเก็บไว้ในตัวแปรทำการเฉลี่ยค่าอุณหภูมิที่วัดได้ใน 1 นาที และแสดงผลลัพธ์บนหน้าจอ

โปรแกรมตัวอย่าง

```
#include<stdio.h>
#include<dos.h>

void interrupt (*oldvect)( );
void interrupt prn( );
int i=0;

void interrupt prn( )
{
        i++;
        _AH = 0x0a;
        _AL = i;
        _BH = 0;
        _CX = 10;
        geninterrupt(0x10);             /* call BIOS interrupt */
        outportb(0x20,0x20);            /* send EOI to 8259 */
        (*oldvect)( );                  /* call old ISR */
}

main( )
{
        int c;
        oldvect = getvect(8);           /* save old vector */
        setvect(8,prn);                 /* store new vector point to prn fuction */
        while(c=getche( )!=27);         /* loop until press ESC key */
        setvect(8,oldvect);             /* restore old vector */
        printf("%u \n",i);              /* print number of interrupt happen */
}
```

การทดลองที่ 8 : การสื่อสารข้อมูล (Data Communication)

1. วัตถุประสงค์

1. เพื่อเรียนรู้การสื่อสารข้อมูลแบบอนุกรม และแบบขนาน
2. เพื่อทดลองเขียนโปรแกรมเพื่อสื่อสารข้อมูลระหว่างเครื่องคอมพิวเตอร์ 2 เครื่อง ด้วยวิธีอนุกรมและขนาน
3. เพื่อทดลองการควบคุมผ่านทางพอร์ตอนุกรมและพอร์ตขนานของเครื่องคอมพิวเตอร์

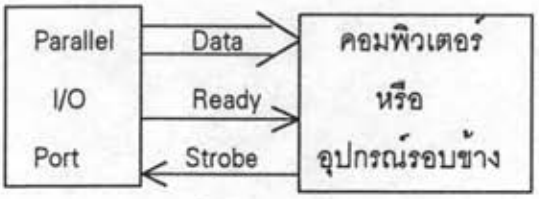
2. ทฤษฎี

การติดต่อส่งผ่านข้อมูลกันระหว่างอุปกรณ์ภายในเครื่องคอมพิวเตอร์อาศัยบัสของระบบ ซึ่งสามารถส่งข้อมูลได้เร็ว แต่มีระยะทางที่สั้น การจะทำให้คอมพิวเตอร์ตั้งแต่ 2 เครื่องขึ้นไป สามารถส่งข้อมูลถึงกันได้ หรือให้คอมพิวเตอร์รับส่งข้อมูลกับอุปกรณ์ภายนอก ถ้าใช้ระบบบัสจะทำให้ต้องเสียเวลาในการออกแบบการเชื่อมต่ออยู่เสมอ จึงได้มีการคิดมาตรฐานในการสื่อสารข้อมูลขึ้น โดยที่เพียงคอมพิวเตอร์ 2 ตัวที่ใช้มาตรฐานการสื่อสารข้อมูลเดียวกัน ก็สามารถรับส่งข้อมูลกันได้ การสื่อสารข้อมูลสามารถแบ่งได้เป็น 2 ชนิด คือ การสื่อสารข้อมูลแบบขนาน และการสื่อสารข้อมูลแบบอนุกรม การสื่อสารข้อมูลแบบขนานเป็นวิธีการรับส่งข้อมูลโดยอาศัยการส่งข้อมูลพร้อมกันทีละหลายๆบิต ซึ่งจะต้องอาศัยสายส่งจำนวนหลายเส้น ส่วนการสื่อสารข้อมูลแบบอนุกรมเป็นการส่งข้อมูลทีละบิต อาศัยสายส่งเพียง 2 เส้น(สายสัญญาณและกราวด์) ถ้าเปรียบเทียบลักษณะของการสื่อสารทั้ง 2 แบบ จะได้ดังนี้

| แบบอนุกรม | แบบขนาน |
|---|---|
| 1. ใช้สายส่งเพียง 2 เส้น | 1. ใช้สายส่งหลายเส้น(ขึ้นกับจำนวนบิต) |
| 2. ส่งทีละบิต | 2. ส่งทีละหลายบิตพร้อมกัน |
| 3. อัตราการส่งข้อมูลช้า | 3. อัตราการส่งข้อมูลเร็ว |
| 4. สะดวกในการเดินสายส่งระยะไกล | 4. สิ้นเปลืองสายส่งในกรณีส่งระยะไกลๆ |

การสื่อสารข้อมูลแบบขนาน[17]

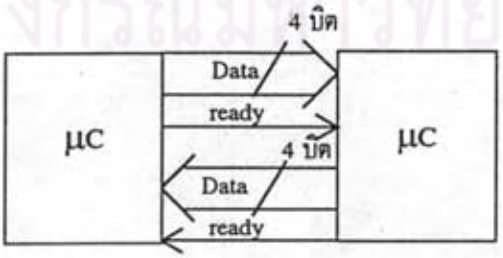
รูปที่ ก.8.1 แสดงหลักการส่งข้อมูลแบบขนานพื้นฐาน[17] มีหลักการที่ง่ายๆ ขั้นตอนน้อย ข้อมูลจะถูกส่งออกไปพร้อมกันทีละหลายๆบิตผ่านสาย Data และจะมีสายสัญญาณแฮนด์เชคอีกเส้นหนึ่งหรือมากกว่า




รูปที่ ก.8.1 การส่งข้อมูลแบบขนาน

คำว่า แฮนด์เชคในทางการสื่อสารข้อมูล อาจจะหมายความได้ว่า คือ การซิงโครไนซ์กันระหว่างอุปกรณ์ทางด้านส่งและด้านรับนั่นเอง จากรูป ก.8.1 สายสัญญาณ Ready จะเป็นตัวบอกให้อุปกรณ์ด้านรับทราบว่า ข้อมูลนั้นได้ถูกส่งออกมาจากทางด้านส่งเรียบร้อยแล้ว สายสัญญาณสโตรปจะเป็นสัญญาณที่ส่งมาจากทางด้านรับเพื่อบอกทางด้านส่งให้ทราบว่า ทางด้านรับได้รับข้อมูลไปแล้ว และพร้อมที่จะรับข้อมูลตัวต่อไป

ตัวอย่างมาตรฐานการสื่อสารข้อมูลแบบขนาน เช่น มาตรฐาน IEEE-488(GPIB)[7] เป็นมาตรฐานที่นิยมใช้ในการรับส่งข้อมูลระหว่างอุปกรณ์ต่างๆในระยะสั้นภายในห้องทดลองหรือโรงงานอุตสาหกรรม สามารถส่งข้อมูลได้สูงสุด 1 เมกกะไบต์ต่อวินาที สามารถเชื่อมต่ออุปกรณ์ต่างๆเข้าด้วยกันสูงสุดจำนวน 15 ตัว สายส่งระหว่างอุปกรณ์แต่ละตัวจะต้องยาวไม่เกิน 2 เมตร มีจำนวนเส้น 24 เส้น[7] นอกจากนี้ยังมีการสื่อสารข้อมูลแบบขนานที่นิยมในปัจจุบันบนเครื่อง IBM PC โดยการใช้สาย LapLink® ทำการเชื่อมต่อพอร์ตขนานของ IBM PC 2 เครื่องเข้าด้วยกัน เพื่อรับข้อมูล หรือโอนย้ายไฟล์กัน โปรโตคอลที่ใช้ในการส่งข้อมูลผ่านสาย LapLink อาจจะมีแบบต่างๆมากมาย ในการทดลองเรื่องการสื่อสารข้อมูลแบบขนานนี้ จะเป็นการทดลองส่งข้อมูลโดยใช้สาย LapLinkนี้ และได้เสนอตัวอย่างของโปรโตคอลที่ใช้ในการส่งข้อมูล ซึ่งเป็นโปรโตคอลที่มีการตรวจสอบความถูกต้องของข้อมูลที่ส่ง[8] รูปที่ ก.8.2 แสดงการรับส่งข้อมูลโดยใช้สาย LapLink

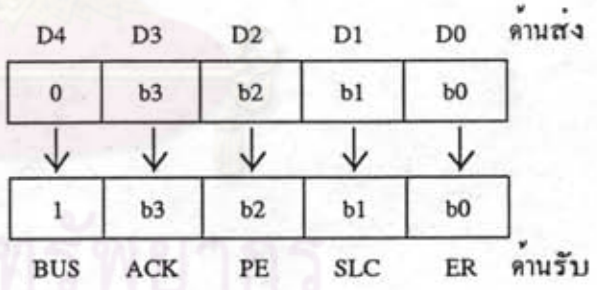



รูปที่ ก.8.2 การรับส่งข้อมูลโดยใช้สาย LapLink

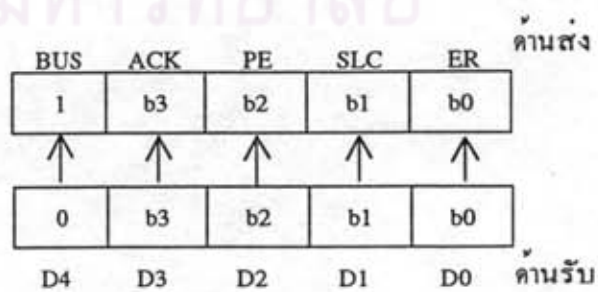
คอมพิวเตอร์ทั้ง 2 เครื่อง สามารถทำหน้าที่เป็นทั้งตัวรับและตัวส่ง การส่งข้อมูลจะเริ่มต้นจากทางด้านส่ง โดยทางด้านส่งจะส่งข้อมูลออกมาบนสาย Data และส่งสัญญาณ Ready เพื่อบอกทางด้านรับว่า ข้อมูลได้ถูกส่งออกมาแล้ว ทางด้านรับจะอ่านข้อมูลเข้าไป และจะส่งข้อมูลที่อ่านเข้าไปกลับออกมาทางสาย Data เพื่อส่งให้แก่ทางด้านส่งนำไปตรวจสอบว่า ทางด้านรับนั้นรับข้อมูลได้ถูกต้องหรือไม่

สัญญาณที่นำมาใช้จะใช้สัญญาณจากพอร์ตขนาน บิต D0-D4 ซึ่งเป็นพอร์ตสัญญาณขาออก ทำหน้าที่ในการส่งข้อมูลออก และใช้บิต BUSY , -ACK , PE , SLCT , ERROR ซึ่งเป็นพอร์ตสัญญาณขาเข้า ทำหน้าที่ในการรับข้อมูล รูปที่ ก.8.3 แสดงการเชื่อมต่อบิตต่างๆของทางด้านส่ง และด้านรับเข้าด้วยกัน ซึ่งมีจำนวนทั้งหมด 10 บิต และแสดงโปรโตคอลที่ใช้ในการส่ง การส่งข้อมูลจะกระทำทีละ 4 บิต( 1 nibble) ฉะนั้นจึงต้องทำการส่ง 2 ครั้ง เพื่อให้ได้ข้อมูล 1 ไบต์

1. ทางด้านส่งเริ่มส่งข้อมูลโดยส่ง low nibble โดยเขียนข้อมูลลงบนบิต D0-D3 และทำการรีเซ็ตบิต D4 ให้เป็น 0 ซึ่งทางด้านรับจะรับค่าได้เป็น 1 ทางบิต BUSY

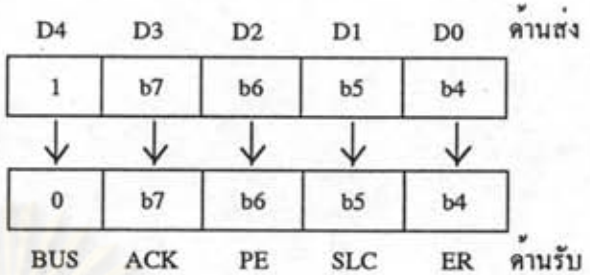



2. ทางด้านรับซึ่งกำลังรอบิต BUSY ให้เปลี่ยนเป็น 1 เมื่อได้รับ BUSY เป็น 1 ก็จะทำการอ่านข้อมูลเข้ามาและทำการเขียนข้อมูลที่อ่านได้กลับไปบนบิต D0-D3 เพื่อส่งให้แก่ด้านส่ง โดยจะทำการรีเซ็ตบิต D4 ให้เป็น 0 ด้วย ทางด้านส่งก็จะไปปรากฏเป็น 1 ที่บิต BUSY



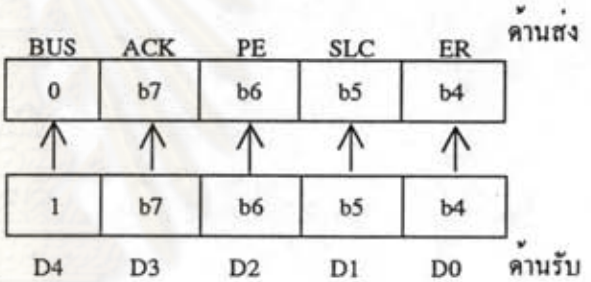
3.

ทางด้านส่งซึ่งกำลังรอบิต BUSY
ให้เปลี่ยนเป็น 1 เมื่อได้รับ BUSY
เป็น 1 ก็จะอ่านข้อมูลเข้ามาเพื่อ
ทำการตรวจสอบกับข้อมูลที่ได้ส่งออกไป
ในภายหลังหลังจากนั้นจะเขียน
ข้อมูล high nibbleลงบิต D0-D3 และ
เซ็ตบิต D4 ให้เป็น 1 ซึ่งทางด้านรับจะรับค่าได้เป็น 0
เพื่อแสดงว่าเป็นการส่งข้อมูลส่วน high nibble




4.

ในขณะนี้ทางด้านรับกำลังรอ
บิต BUSY ให้เปลี่ยนเป็น 0
เมื่อได้รับก็จะอ่านข้อมูลเข้ามา
และทำการเขียนข้อมูลที่อ่านได้
กลับไปยังตัวส่ง โดยเซ็ตบิต D4
ให้เป็น 1 ซึ่งทางด้านส่งจะรับค่าได้เป็น 0
ทางบิต BUSY




5.

เมื่อบิต BUSY ของด้านส่ง เปลี่ยนเป็น 0
ด้านส่งก็จะอ่านข้อมูลเข้ามาเพื่อนำไป
ตรวจสอบความถูกต้อง เป็นอันจบขั้นตอน
การสื่อสารข้อมูล ทางด้านรับ ก็จะนำ nibble
2 ค่าที่ได้รับมา นำมาสร้างเป็นข้อมูล 1 ไบต์
และทางด้านส่ง ก็จะนำข้อมูลที่รับเข้ามา
ตรวจสอบกับข้อมูลที่ส่งไป ว่ามีความผิดพลาด
เกิดขึ้นหรือไม่

รูปที่ ก.8.3 โปรโตคอลการส่งข้อมูลผ่านสาย LapLink

**การสื่อสารข้อมูลแบบอนุกรม[17]**

การสื่อสารข้อมูลแบบอนุกรมอาศัยการแปลงรูปแบบข้อมูลจากแบบขนานให้กลายเป็นแบบอนุกรม และทำการส่งโดยอาศัยสายจำนวนน้อย ทางด้านรับจะทำการแปลงข้อมูลจากแบบอนุกรมให้กลับมาเป็นแบบขนาน อย่างไรก็ตามการส่งข้อมูลแบบอนุกรมนั้นจะต้องมีการทำแฮนด์เชคระหว่างด้านรับและด้านส่ง มิฉะนั้นอาจทำให้เกิดความผิดพลาดต่างๆดังนี้

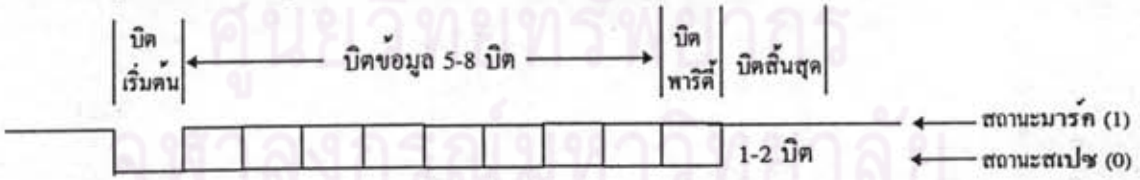
1. ด้านส่งทำการส่งข้อมูลทั้งๆที่ด้านรับยังไม่พร้อมจะรับข้อมูล(overrun error)
2. ด้านส่งอาจไม่ทำการส่งข้อมูลแม้ว่าด้านรับพร้อมที่จะรับข้อมูลแล้ว
3. ด้านรับสุ่มข้อมูลด้วยจังหวะเวลาที่ผิด ทำให้รับข้อมูลคลาดเคลื่อน(framing error)

การทำแฮนด์เชคสามารถทำได้ 2 วิธีคือการทำฮาร์ดแวร์แฮนด์เชค และ การทำซอฟต์แวร์แฮนด์เชค การทำฮาร์ดแวร์แฮนด์เชคจะอาศัยการเพิ่มสายสัญญาณแฮนด์เชคเข้าไป ส่วนการทำซอฟต์แวร์แฮนด์เชค จำนวนสายส่งมีจำนวน 2 เส้นเท่าเดิม(กรณี Simplex) แต่จะอาศัยการส่งข้อมูลพิเศษ เพื่อทำแฮนด์เชคระหว่างด้านรับและด้านส่ง ตัวอย่างของโปรโตคอลที่ใช้ซอฟต์แวร์แฮนด์เชค เช่น

- XON/XOFF
- XMODEM
- YMODEM

**รูปแบบการส่งข้อมูลอนุกรมแบบอะซิงค์โครนัส**

การส่งข้อมูลแบบอนุกรมจะทำการส่งออกไปทีละบิต โดยจะมีการเพิ่มบิตบางบิตเข้าไปในข้อมูลเพื่อบอกจุดเริ่มต้นของการส่ง บอกจุดสิ้นสุด หรือเพื่อตรวจสอบความถูกต้องของข้อมูลที่ส่งออกไป รูปแบบของข้อมูลแบบที่นิยมใช้แสดงดังรูปที่ ก.8.4




รูป ก.8.4 รูปแบบข้อมูลของการส่งแบบอนุกรม

- บิตเริ่มต้น

ในโปรโตคอลของการส่งข้อมูลแบบอะซิงค์โครนัสจะกำหนดให้สถานะมาร์ค(Mark State) เป็นสัญญาณลอจิก 1 และสถานะสเปซ(Space State) เป็นสัญญาณลอจิก0 เมื่อทางด้านส่งจะ

ทำการส่งข้อมูล จะบอกให้ทางด้านรับนั้นทราบโดยการส่งสัญญาณลอจิก 0 จำนวน 1 บิตออกไปก่อน

- บิตข้อมูล

เป็นส่วนของเนื้อหาข้อมูลที่จะทำการส่ง สามารถมีจำนวนบิตได้ตั้งแต่ 5 บิตถึง 8 บิต โดยทั่วไปนิยมส่งขนาด 8 บิต แต่ถ้าเป็นการส่งรหัส ASCII ซึ่งเป็นรหัสขนาด 7 บิต ก็จะส่งด้วยจำนวนบิต 7 บิต สำหรับการส่งแบบ 8 บิต เช่น การส่งรหัส EBCDIC เป็นต้น

- บิตพาริตี้

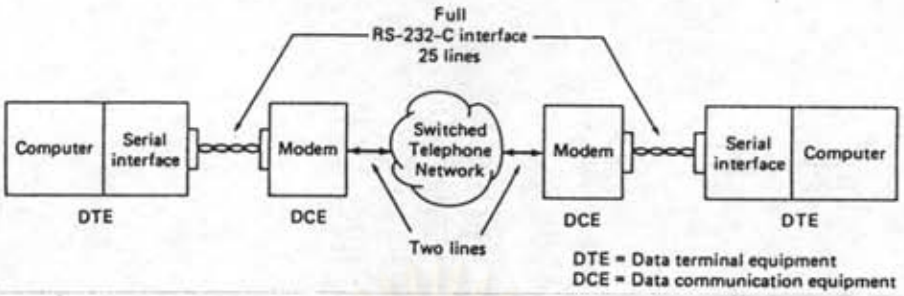
เป็นบิตที่ทำหน้าที่ในการตรวจสอบข้อมูลที่รับเข้ามาว่าถูกต้องหรือไม่ สามารถเลือกชนิดได้ว่าจะส่งแบบพาริตี้คู่ หรือพาริตี้คี่ ในกรณีเลือกแบบพาริตี้คู่ ถ้าจำนวนบิตที่เป็น 1 ของบิตข้อมูลรวมกับบิตพาริตี้เป็นจำนวนคู่ จะทำการส่งบิตพาริตี้เป็นลอจิก 0 ถ้าเป็นจำนวนคี่ จะทำการส่งบิตพาริตี้เป็นลอจิก 1 (คือ ทำให้จำนวนบิตที่เป็น 1 ของบิตข้อมูลรวมกับบิตพาริตี้เป็นจำนวนคู่นั่นเอง) และในกรณีเลือกแบบพาริตี้คี่ บิตพาริตี้จะถูกกำหนดให้เป็น 0 หรือ 1 เพื่อทำให้จำนวนบิตที่เป็น 1 ของบิตข้อมูลเมื่อรวมกับบิตพาริตี้เป็นจำนวนคี่ บิตพาริตี้นี้สามารถเลือกส่งหรือไม่ก็ได้

- บิตสิ้นสุด

บิตนี้ใช้ในการตรวจสอบจุดสิ้นสุดของข้อมูลสามารถมีจำนวนบิตเป็น 1,1.5หรือ 2 บิตก็ได้

**มาตรฐาน RS-232**

มาตรฐานที่นิยมใช้ในการสื่อสารข้อมูลแบบอนุกรม คือ มาตรฐาน RS-232 ซึ่งได้บรรยายวิธีการสื่อสารข้อมูลระหว่างเครื่องคอมพิวเตอร์หรืออุปกรณ์ผ่านเครือข่ายโทรศัพท์ (ดูรูปที่ ก.8.5) อุปกรณ์ที่ใช้ในการสื่อสารตามมาตรฐาน RS-232 สามารถแบ่งได้เป็น 2 ชนิด คือ DTE(Data Terminal Equipment) และ DCE(Data Communication Equipment) DTE เป็นอุปกรณ์ที่ทำหน้าที่เป็นต้นทาง คือ ให้กำเนิดข้อมูลในการส่ง หรือ เป็นปลายทาง คือ เป็นตัวรับข้อมูล ซึ่งมักจะเป็นเครื่องคอมพิวเตอร์ ส่วน DCE เป็นอุปกรณ์ที่ทำหน้าที่ในการเชื่อมต่อ คือ เป็นตัวเปลี่ยนสัญญาณให้สามารถส่งผ่านสื่อได้ (ผ่านเครือข่ายโทรศัพท์) และเปลี่ยนสัญญาณกลับมาเป็นสัญญาณที่ DTE สามารถรับได้ โดยทั่วไป DCE ก็คือ MODEM (modulation-demodulation) ซึ่งทำหน้าที่ในการเปลี่ยนสัญญาณลอจิก 0 , 1 ให้กลายเป็นความถี่ในย่านความถี่เสียง เพื่อสามารถส่งผ่านเครือข่ายโทรศัพท์ได้ ความจริงมาตรฐาน RS-232 ถูกสร้างขึ้นเพื่อบรรยายการเชื่อมต่อระหว่าง DTE และ DCE แต่ก็ได้มีการนำไปใช้ในการเชื่อมต่ออุปกรณ์ดิจิทัลต่างๆมากมาย โดยไม่ได้ยึดตามมาตรฐาน เช่น นำไปใช้เชื่อมต่อระหว่าง DTE กับ DTE หรือ DCE กับ DCE ซึ่งไม่มีมาตรฐานที่แน่นอนในการเชื่อมต่อ[17]




รูปที่ ก.8.5 การสื่อสารข้อมูลผ่านเครือข่ายโทรศัพท์

**พอร์ตสื่อสารของ IBM PC[18,28,29]**

พอร์ตสื่อสาร หรือมีชื่อเรียกอีกอย่างหนึ่งว่า COM Port เป็นพอร์ตประจำเครื่อง IBM PC ซึ่งผู้ออกแบบได้ออกแบบให้เป็นไปตามมาตรฐาน RS-232 มีโครงสร้างที่สามารถโปรแกรมได้ด้วยการส่งรหัสคำสั่งให้กับชิปหลัก สามารถเลือกได้ว่าจะสื่อสารแบบกระแสวงรอบ(Current Loop) หรือ แบบแรงดัน โดยการเซ็ตที่จัมเปอร์ การส่งข้อมูลเป็นไปตามมาตรฐานการส่งข้อมูลอนุกรมแบบอะซิงค์โครนัส สามารถโปรแกรมสตาร์ตบิต สต๊อปบิต พาริตี้บิต และอัตราการส่ง(baud rate) ได้ ซึ่งสามารถกำหนดได้ตั้งแต่ 50 baud ถึง 9600 baud ถ้าเป็นกรณีที่ใช้ชิปหลัก คือ 8250 ปัจจุบันเครื่อง IBM PC มักจะเปลี่ยนไปใช้ชิปหลัก คือ 16550 ซึ่งสามารถส่งด้วยอัตราที่สูงกว่า 9600 baud ชิป 16550 มีความเข้ากันได้(Compatibility) กับ 8250 ซึ่งในที่นี้จะกล่าวถึงเฉพาะ 8250 ซึ่งสามารถนำไปใช้กับ 16550 ได้ทุกประการ พอร์ตสื่อสารบน IBM PC สามารถต่อได้สูงสุด 7 พอร์ต แต่เครื่อง IBM PC โดยทั่วไปมักจะมีอยู่ 2 พอร์ต คือ COM1 และ COM2 ซึ่งทำงานแยกกันเป็นอิสระ COM1 จะมีหมายเลขพอร์ตอยู่ที่ 3F8h-3FEh และ COM2 จะมีหมายเลขพอร์ต 2F8h-2FEh สำหรับขีดความสามารถของพอร์ตสื่อสาร ได้แก่

- มีบัฟเฟอร์ในการรับส่งข้อมูล
- สามารถดีเทคความผิดพลาดในการรับส่งข้อมูลได้
- มีสัญญาณการควบคุมโมเด็ม คือ RTS, CTS, DTR, DSR, DCD, RI
- มีสัญญาณไปอินเตอร์รัปต์ CPU
- มีสัญญาณนาฬิกาในตัวที่ใช้กำหนดอัตราการส่ง และสามารถโปรแกรมอัตราการส่งได้

ตาราง ก.8.1 แสดงหมายเลขพอร์ต และระดับของการอินเตอร์รัปต์ของพอร์ตสื่อสาร

| พอร์ตสื่อสาร | หมายเลขพอร์ต | ระดับของการอินเตอร์รัปต์ |
|---|---|---|
| COM1 | 3F8-3FE | IRQ4 |
| COM2 | 2F8-2FE | IRQ3 |

ตาราง ก.8.2 แสดงสัญญาณต่างๆที่พอร์ตสื่อสารของ IBM PC

| ขา | สัญญาณ |
|---|---|
| ขา 18 | ขา + ของข้อมูลที่จะรับเข้ามา |
| ขา 25 | ขา - ของข้อมูลที่จะรับเข้ามา |
| ขา 9 | ขา + ของข้อมูลที่จะส่ง |
| ขา 11 | ขา - ของข้อมูลที่จะส่ง |
| ขา 2 | ขาข้อมูลส่งออก |
| ขา 3 | ขารับข้อมูล |
| ขา 4 | ขา RTS |
| ขา 5 | ขา CTS |
| ขา 6 | ขา DSR |
| ขา 7 | กราวด์ |
| ขา 8 | ขาดีเทคสัญญาณพาหะ |
| ขา 20 | ขา DTR |
| ขา 22 | ขาดีเทคสัญญาณกระดิ่ง |

**การโปรแกรมพอร์ตสื่อสาร**

การควบคุมพอร์ตสื่อสารทำได้โดยการโปรแกรมรีจิสเตอร์ต่างๆของ 8250 ซึ่งประกอบด้วยรีจิสเตอร์ต่างๆดังแสดงในตารางที่ ก.8.3

ตารางที่ ก.8.3 รีจิสเตอร์บน 8250

| อินพุตเอาต์พุตพอร์ต | | เลือกรีจิสเตอร์ | สถานะ DLAB |
|---|---|---|---|
| พอร์ต COM1 | พอร์ต COM2 | | |
| 3F8 | 3FA | บัฟเฟอร์ Tx | DLAB=0(เขียน) |
| 2F8 | 2FA | บัฟเฟอร์ Rx | DLAB=0(อ่าน) |
| 3F8 | 3FB | แลตช์ตัวหาร LSB | DLAB=1 |
| 2F8 | 2FB | แลตช์ตัวหาร MSB | DLAB=1 |
| 3F8 | 3FC | รีจิสเตอร์อินเตอร์รัปต์เอนาเบิล | |
| 2F8 | 2FC | รีจิสเตอร์เลือกอินเตอร์รัปต์ | |
| 3F9 | 3FD | รีจิสเตอร์ควบคุมสายสื่อสาร | |
| 2F9 | 2FD | รีจิสเตอร์ควบคุมโมเด็ม | |
| 3F9 | 3FE | รีจิสเตอร์แสดงสถานะสายสื่อสาร | |
| 2F9 | 2FE | รีจิสเตอร์แสดงสถานะโมเด็ม | |

สำหรับรายละเอียดการใช้งานรีจิสเตอร์แต่ละตัวสามารถศึกษาได้จากหนังสือ IBM PC Technical Reference[10] นอกจากการโปรแกรม 8250 โดยตรง เราสามารถใช้ฟังก์ชันของ ROM BIOS ซึ่งอยู่ที่อินเตอร์รัปต์หมายเลข 20 (14h) เป็นฟังก์ชันจัดการการสื่อสารซีเรียล ผู้อ่านสามารถดูรายละเอียดได้จากโปรแกรม Expert Help ( EH )

## 3. อุปกรณ์การทดลอง

1. เครื่องคอมพิวเตอร์ IBM PC 2 เครื่อง
2. โปรแกรม Turbo C Version 2.0
3. เมนบอร์ด
4. มอดูลการสื่อสารแบบขนาน
5. มอดูลการสื่อสารแบบอนุกรม
6. หลอดไฟ 6V

## 4. หนังสืออ่านประกอบ

1. การสื่อสารข้อมูล[17]
2. เทคโนโลยีไมโครคอมพิวเตอร์[18]

5. ขั้นตอนการทดลอง

1. Parallel Communication

1.1 ต่อ มอดูล การสื่อสารแบบขนาน เข้ากับเครื่องคอมพิวเตอร์ทั้ง 2 เครื่อง ทางพอร์ตขนาน

1.2 ทดลองเขียนโปรแกรม เพื่อส่งสัญญาณไปยังพอร์ตขนาน สังเกตดู LED ว่าติด-ดับ ถูกต้องหรือไม่

1.3 ทดลองเขียนโปรแกรมภาษาซี เพื่อส่งข้อมูลสำหรับเครื่องคอมพิวเตอร์ตัวหนึ่ง และโปรแกรมรับข้อมูลสำหรับเครื่องคอมพิวเตอร์อีกตัวหนึ่ง โดยใช้โปรโตคอลตามที่เขียนในทฤษฎี

1.4 ทดลองรับ-ส่งข้อมูลระหว่างคอมพิวเตอร์ทั้งสอง สังเกตสถานะของ LED ปรับปรุงโปรแกรมให้สามารถ รับ-ส่ง ข้อมูล ระหว่างเครื่องคอมพิวเตอร์ทั้งสอง ได้ในเวลาเดียวกัน โดยโปรแกรมจะทำงานในลักษณะเป็นโปรแกรมคุยโต้ตอบอิเล็คทรอนิคส์ (Talk) นำตัวอักษรที่พิมพ์เข้าไป ส่งไปยังฝ่ายตรงข้าม และนำตัวอักษรที่ฝ่ายตรงข้ามพิมพ์เข้ามา แสดงผลบนจอภาพ

2. Serial Communication

2.1 ต่อ มอดูล การสื่อสารแบบอนุกรม เข้ากับเครื่องคอมพิวเตอร์ 2 เครื่อง ทางพอร์ตอนุกรม

2.2 ทดลอง โปรแกรม ชิปเบอร์ 8250 (หรือ 16550) บน เครื่องคอมพิวเตอร์ทั้ง 2 เครื่อง ให้มีลักษณะดังนี้

1) Baud rate = 9600 bps
2) Data bit = 8 bit
3) Parity bit = none
4) Stop bit = 1 bit

2.3 ทดลองเขียนโปรแกรม เพื่อรับ - ส่งข้อมูล 1 ตัวอักษรโดยใช้วิธี polling ให้สังเกตว่า สามารถ รับ-ส่งข้อมูลได้ถูกต้องหรือไม่

2.4 ต่อชุดฝึกทดลองเข้ากับเครื่องคอมพิวเตอร์ IBM PC

2.5 เขียนโปรแกรม เพื่อส่งตัวอักษรผ่านทางพอร์ตอนุกรม เพื่อไปควบคุมการปิด-เปิด รีเลย์ ที่อยู่บนชุดฝึกทดลองที่ต่อเข้ากับเครื่องคอมพิวเตอร์อีกด้านหนึ่ง โดยให้เมื่อมีการส่งตัวอักษรที่มีรหัสตามที่กำหนดไป 1 ตัว ให้ทำการสลับสถานะเปิด-ปิดของรีเลย์ ทดลองต่อรีเลย์เพื่อควบคุมการเปิด-ปิดหลอดไฟ

2.6 ทดลองโปรแกรมรีจิสเตอร์ควบคุมโมเด็ม สังเกตการติด - ดับของ LED บนมอดูล

2.7 เขียนโปรแกรมเพื่อรับ-ส่งข้อมูลแบบอนุกรม อย่างต่อเนื่องขนาด 10 ไบต์ โดยมีการทำแฮนด์เชค (ฮาร์ดแวร์แฮนด์เชค) โดยใช้สัญญาณควบคุมโมเด็มโปรโตคอลที่ใช้ให้ทำการออกแบบขึ้นมาเอง โดยให้สามารถรับส่งข้อมูลได้โดยไม่เกิด overrun error

## การทดลองที่ 9 : การประยุกต์ใช้งาน 1
## การควบคุมสเตปปิงมอเตอร์

### 1. วัตถุประสงค์

1. เพื่อทดลองประยุกต์ใช้งานการควบคุม โดยใช้ไมโครคอมพิวเตอร์ในงานต่าง ๆ
2. เพื่อทดลองเขียนโปรแกรมควบคุม Stepping Motor

### 2. ทฤษฎี

ในหุ่นยนต์หรือเครื่องกลมักมีองค์ประกอบหลักคือ มอเตอร์ ประกอบอยู่ด้วย และเครื่องที่ควบคุมด้วยคอมพิวเตอร์ส่วนใหญ่มักมีสเต็ปปิงมอเตอร์เป็นตัวขับเคลื่อน ถ้าจะมองหาตัวอย่างก็สามารถพบได้ในเครื่องคอมพิวเตอร์ นั่นก็คือ ฟลอปปี้และฮาร์ดดิสค์ไดรฟ์ซึ่งใช้สเต็ปเปอร์มอเตอร์ขนาดเล็กในการขับเคลื่อนหัวอ่าน/เขียนข้อมูลไปบนพื้นผิวของแผ่นดิสค์ เพื่อให้ตรงกับตำแหน่งของแทร็คที่ถูกต้อง นอกจากนี้สเต็ปปิงมอเตอร์ยังใช้ในงานต่าง ๆ มากมาย เช่น ควบคุมทิศทางการหมุนของสายอากาศ สร้างเป็นพล็อตเตอร์ ระบบเซอร์โว ตลอดจนถึงเครื่องจักรกลต่าง ๆ เป็นต้น

### การทำงานของสเต็ปปิงมอเตอร์

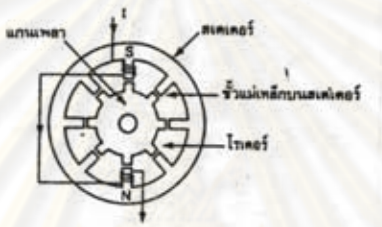
สเต็ปปิงมอเตอร์มีความแตกต่างจากมอเตอร์ทั่วๆไป โดยเมื่อป้อนกำลังไฟฟ้าให้กับมันมันจะหมุนเพียงเล็กน้อยตามเส้นรอบวงและหยุด ซึ่งแตกต่างจากมอเตอร์ทั่วๆไปซึ่งจะหมุนทันทีและตลอดเวลา สเต็ปปิงมอเตอร์สามารถกำหนดตำแหน่งของการหมุนด้วยตัวเลขได้อย่างละเอียด โดยการใช้คอมพิวเตอร์เป็นตัวกำหนดและจัดเก็บตัวเลขเหล่านั้นไว้

สเต็ปปิงมอเตอร์สามารถใช้งานในระบบเปิด(open loop system) นั่นก็คือ มันทำงานได้โดยไม่ต้องการการป้อนกลับ(feedback)

เช่นเดียวกับมอเตอร์ทั่วไป การที่จะทำให้เกิดการหมุนของโรเตอร์(rotor)ได้ ต้องมีการกระทำของสนามแม่เหล็กที่เกิดขึ้นระหว่างโรเตอร์และสเตเตอร์(stator) ซึ่งขึ้นอยู่กับการจัดวางขั้วแม่เหล็ก(pole) การหมุนทำได้ทั้งแบบต่อเนื่องและกลับทิศทางไปมา โดยกระบวนการทางไฟสลับหรือการจัดวางแปรงถ่าน และการจัดแยกคอมมิวเตเตอร์ และทำการสวิตชิ่งกำลังไฟฟ้าให้เกิดแรงดึงดูดของแม่เหล็ก(magntic attraction) ที่ขั้วแม่เหล็กสร้างและหยุดสลับกัน ผลก็คือเกิดสนามแม่เหล็กหมุนขึ้นบนสเตเตอร์โดยการจ่ายกำลังไฟฟ้าทีละคู่ของขั้วแม่เหล็กในทิศทางตรงกันข้ามไปตลอดเวลา และเมื่อต้องการหยุดหมุนทำได้โดยหยุดการเกิดขั้วแม่เหล็กที่จุดหนึ่งโดยหยุด

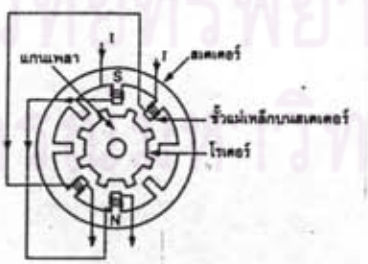
การ สวิตชิ่งในลำดับต่อไปเสีย การหมุนกลับทิศทางก็ทำได้เช่นเดียวกับที่กล่าวมาแล้วเพียงแต่ทำการ สวิตชิ่งกำลังไฟฟ้าให้เกิดสนามแม่เหล็กหมุนในทิศทางกลับกัน หรือกลับลำดับการสวิตช์ของมัน

โครงสร้างของขั้วแม่เหล็กบนสเตเตอร์ประกอบขึ้นจากแผ่นเหล็กวงแหวนที่มีขี่ยื่นออกมา แต่ละขี่เหล่านั้นจะมีคอยล์พันสวมอยู่ ดังนั้นเมื่อป้อนกระแสไฟฟ้าผ่านคอยล์ทำให้เกิดสนามแม่เหล็กไฟฟ้าขึ้น ด้านตรงข้ามของแต่ละขั้วแม่เหล็กจะได้รับกระแสไฟฟ้าในขณะเดียวกัน แต่ว่าจะไหลวนในทิศทางตรงกันข้ามทำให้เกิดสนามแม่เหล็กไฟฟ้าในทิศตรงข้ามขึ้น ดังแสดงในรูปที่ ก.9.1 ดังนั้นถ้าเพิ่มจำนวนของขั้วแม่เหล็กมากขึ้นจะเพิ่มจำนวนของสเต็ปต่อวงรอบมากขึ้นตามไปด้วย




รูปที่ ก.9.1 การกระตุ้นให้เกิดขั้วแม่เหล็ก 1 ขั้ว

อย่างไรก็ตามผู้ใช้งานสามารถเพิ่มจำนวนของสเต็ปได้อีกวิธีหนึ่งโดยไม่ต้องปรับเปลี่ยนโครงสร้างภายใน โดยทำการจ่ายกำลังไฟฟ้าไปยังขั้วแม่เหล็ก 2 ขั้วที่อยู่ใกล้กันในเวลาเดียวกัน ซึ่งจะทำให้โรเตอร์หยุดหมุนอยู่ระหว่างกลางของ 2 ขั้วแม่เหล็กนั้น หรือเคลื่อนที่ไปครึ่งสเต็ปเท่านั้น และวิธีการนี้ยังช่วยให้เกิดแรงบิด(torque)มากขึ้นด้วย ดังแสดงในรูปที่ ก.9.2




รูปที่ ก.9.2 การกระตุ้นให้เกิดขั้วแม่เหล็ก 2 ขั้ว

สเต็ปปิงมอเตอร์โดยทั่วไปมีจำนวนของขั้วแม่เหล็กหรือจำนวนสเต็ปต่อรอบเป็นจำนวนมาก ปกติอยู่ที่ประมาณ 100-400 สเต็ปต่อรอบ การมีจำนวนสเต็ปมากๆนี้ไม่ได้เพิ่มที่จำนวนขั้วแม่เหล็กไฟฟ้าที่สเตเตอร์ แต่ทำได้โดยเพิ่มจำนวนขี่ขั้วแม่เหล็กที่โรเตอร์ จำนวนสเต็ปต่อรอบทั้ง

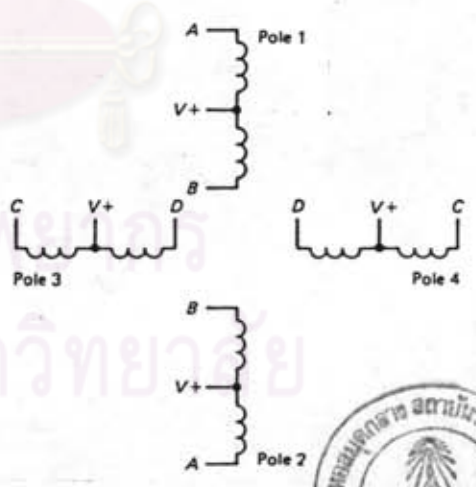
หมดจะได้จากการคูณจำนวนขั้วแม่เหล็กบนสเตเตอร์และจำนวนซี่ที่โรเตอร์ ดังเช่น ถ้ามีขั้วแม่เหล็ก 3 ขั้วบนสเตเตอร์ และ 8 ซี่ขั้วแม่เหล็กบนโรเตอร์ สเต็ปปิงมอเตอร์ตัวนี้จะทำงาน 24 สเต็ปต่อรอบ หรือหมุนเป็นมุม 15 องศาต่อสเต็ป

การใช้วงจรดิจิทัลคอนโทรลเลอร์กำหนดการจ่ายกำลังไฟฟ้าเข้าสู่ขดลวดบนสเตเตอร์แบบซีเควนเชียลทำให้สามารถควบคุมการเคลื่อนที่ทุกสเต็ปได้ เช่นเดียวกับการควบคุมในวงจรดีซีเซอร์โว(DC servo) แต่การควบคุมด้วยดิจิทัลไม่จำเป็นต้องมีการป้อนกลับ การเคลื่อนที่ทุกสเต็ปได้จากการคำนวณจำนวนรอบ หรือมุมในการหมุนที่ต้องการ แล้วจึงส่งข้อมูลที่ได้ไปควบคุมการหมุนของมอเตอร์ พิกัดในการทำงาน เช่น ความเร็ว, มุมในการเคลื่อนที่, ตำแหน่งของเพลาถูกกำหนดจากข้อมูลที่ส่งมาควบคุม

## การกระตุ้นและควบคุมการหมุนของสเต็ปปิงมอเตอร์

การกระตุ้นและการควบคุมการหมุนของมอเตอร์ให้เคลื่อนที่ไปแต่ละสเต็ปทำได้โดยจ่ายกำลังไฟฟ้าไปยังขดลวดแต่ละขดบนสเตเตอร์ ซึ่งต้องป้อนเป็นแบบลำดับในรูปแบบที่ถูกต้องด้วย แบ่งออกได้เป็น 3 รูปแบบคือ แบบ 1 เฟส(signal phase), แบบ 2 เฟส(two phase) และแบบครึ่งสเต็ป(half step) ทั้ง 3 แบบต่างก็มีข้อดีและข้อเสียแตกต่างกันออกไป รูปที่ ก.9.3 แสดงลำดับการกระตุ้นขดลวดของทั้ง 3 แบบ

| Rotor position | Winding *A* | Winding *B* | Winding *C* | Winding *D* |
|---|---|---|---|---|
| **Single-phase excitation** | | | | |
| 0 | Off | On | Off | Off |
| $\theta$ | Off | Off | On | Off |
| $2\theta$ | On | Off | Off | Off |
| $3\theta$ | Off | Off | Off | On |
| **Two-phase excitation** | | | | |
| $\theta/2$ | Off | On | On | Off |
| $3/2\,\theta$ | On | Off | On | Off |
| $5/2\,\theta$ | On | Off | Off | On |
| $7/2\,\theta$ | Off | On | Off | On |
| **Half-step mode** | | | | |
| 0 | Off | On | Off | Off |
| $\theta/2$ | Off | On | On | Off |
| $\theta$ | Off | Off | On | Off |
| $3/2\,\theta$ | On | Off | On | Off |
| $2\,\theta$ | On | Off | Off | Off |
| $5/2\,\theta$ | On | Off | Off | On |
| $3\,\theta$ | Off | Off | Off | On |
| $7/2\,\theta$ | Off | On | Off | On |




รูปที่ ก.9.3 แสดงการกระตุ้นขดลวดของสเต็ปปิงมอเตอร์ในแบบต่าง ๆ

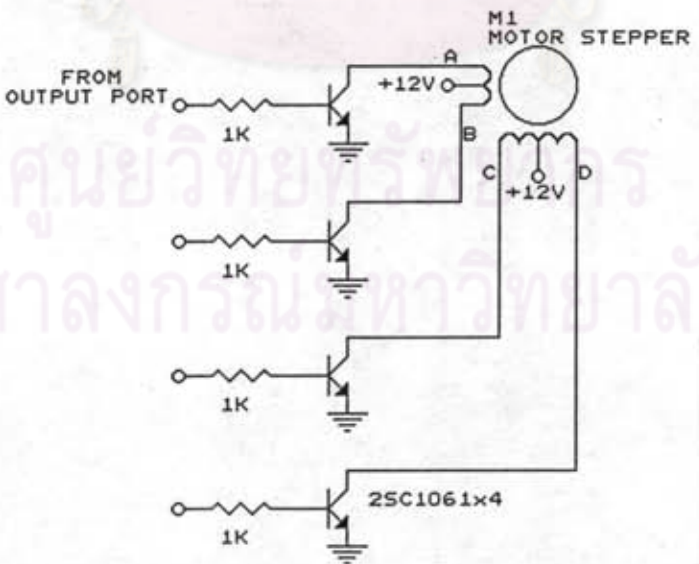
แบบ 1 เฟสเป็นการกระตุ้นรูปแบบที่ง่ายที่สุด โดยทำการกระตุ้นขดลวดทีละขดในเวลาหนึ่ง และเรียงถัดกันไป วงจรกระตุ้นแบบนี้จึงมีราคาถูกและง่าย

แบบ 2 เฟส เป็นการกระตุ้นอีกรูปแบบหนึ่งซึ่งคล้ายกับแบบ 1 เฟส แต่การกระตุ้นแบบนี้จะทำการกระตุ้นโดยจ่ายกำลังไฟฟ้าไปที่ขดลวด 2 ขดที่อยู่ใกล้กันในเวลาเดียวกัน และเรียงถัดกันไปเช่นเดียวกับแบบ 1 เฟส การเพิ่มจำนวนของขดลวดที่ถูกกระตุ้นนี้ทำให้เพิ่มแรงบิดได้มากกว่าแบบ 1 เฟส โรเตอร์จะเคลื่อนที่ด้วยแรงดึงอย่างเต็มแรงจาก 2 ขดลวดที่ถูกกระตุ้นพร้อมกัน และต่อไปด้วยแรงดึงจากอีก 2 ขดลวดถัดไป สำหรับข้อเสียก็คือ การกระตุ้นแบบนี้ต้องใช้แหล่งจ่ายกำลังไฟฟ้ามากขึ้น

แบบครึ่งสเต็ปเป็นรูปแบบที่เกิดจากการผสมผสานระหว่างการกระตุ้นแบบ 1 เฟสและแบบ 2 เฟส เพื่อเพิ่มจำนวนของสเต็ปต่อรอบอีกเท่าตัว แรงบิดที่ได้จากการกระตุ้นแบบนี้จะเพิ่มมากกว่าแบบ 1 เฟส เพราะช่วงสเต็ปมีระยะสั้นลงและแต่ละสเต็ปเกิดแรงดึงจากขดลวด 2 ขดที่ถูกกระตุ้นพร้อมกันในบางจังหวะ ความละเอียดของตำแหน่งมีเพิ่มมากขึ้นหนึ่งเท่าตัว แหล่งจ่ายกำลังไฟฟ้าต้องใช้เทียบเท่ากับแบบ 2 เฟสจึงจะเพียงพอ

**วงจรควบคุมสเต็ปปิงมอเตอร์**

แสดงดังในรูป ก.9.4 โดยใช้ทรานซิสเตอร์เบอร์ 2SC1061 เป็นตัวขับคอยล์แต่ละชุด การควบคุมให้มอเตอร์หมุนทำโดยการกระตุ้นคอยล์แต่ละเฟสด้วยลำดับที่เหมาะตามรูปที่ ก.9.3 และถ้าต้องการให้มอเตอร์หมุนกลับทาง ทำโดยการกระตุ้นคอยล์ด้วยลำดับที่สวนทาง




รูปที่ ก.9.4 วงจรควบคุมสเต็ปปิงมอเตอร์

## 3. อุปกรณ์การทดลอง

1. เครื่องคอมพิวเตอร์ IBM PC
2. โปรแกรม Turbo C Version 2.0
3. เมนบอร์ด
4. มอดูล สเตปปิงมอเตอร์
5. มอดูลสวิตช์
6. แหล่งจ่ายไฟ

## 4. หนังสืออ่านประกอบ

1. Real Time Microcomputer System Design : An Introduction[14]

## 5. ขั้นตอนการทดลอง

1. Stepping Motor Control

1.1 ต่อชุดทดลองเข้ากับ เครื่องคอมพิวเตอร์ IBM PC โดยใช้มอดูลสวิตช์ และ ต่อมอดูลสเตปปิงมอเตอร์ เข้ากับมอดูลสวิตช์

1.2 เขียนโปรแกรม เพื่อควบคุมสเตปปิงมอเตอร์ให้หมุน โดยใช้การกระตุ้นแบบ 1 เฟส 2 เฟส และแบบครึ่งสเตป สังเกตมุมที่มอเตอร์หมุนไปแต่ละสเตปของการกระตุ้นแต่ละแบบและทำการเปรียบเทียบ

1.3 เขียนโปรแกรม เพื่อควบคุมมอเตอร์ให้ หมุนไป-กลับ โดยการสั่งงานจากคีย์บอร์ด

1.4 เขียนโปรแกรม รับค่า องศา ที่ต้องการให้มอเตอร์หมุนไป และทำการหมุนมอเตอร์ไปตามองศานั้น ๆ สังเกตว่า มอเตอร์หยุดได้ถูกตำแหน่งหรือไม่

**การทดลองที่ 10 : การประยุกต์ใช้งาน 2 :**
**การควบคุมการแสดงผลแบบเมตริกซ์ LED**

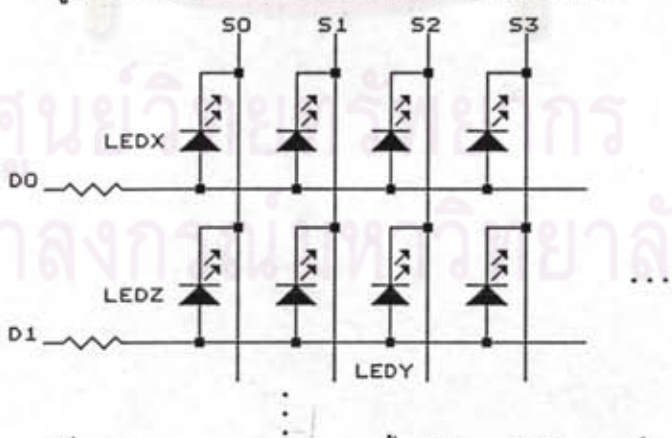
**1. วัตถุประสงค์**

1. เพื่อทดลองเขียนโปรแกรมควบคุมตัวแสดงผลแบบเมตริกซ์ LED

**2. ทฤษฎี**

ถ้าสมมุติเรามีแผงเมตริกซ์ LED ขนาด 3 x 4 ดังแสดงในรูปที่ ก.10.1 การจะสร้างตัวอักษรหรือรูปร่างต่างๆทำได้โดยการให้ LED บางดวงติด และบางดวงดับ ในกรณีนี้เนื่องจากมี LED จำนวน 12 ดวง จึงต้องใช้สายควบคุมทั้งสิ้น 13 เส้น (กราวนด์ 1 เส้น) ในกรณีที่แผง LED มีขนาดใหญ่ขึ้น จำนวนสายควบคุมก็มากขึ้นตามไปด้วย เช่น การณีแผงเมตริกซ์ LED ขนาด 8 x 8 จะต้องใช้สายควบคุมทั้งสิ้น 65 เส้น ซึ่งคงเป็นเรื่องยากที่จะเดินสายขนาด 65 เส้น ไปที่แผงแสดงผล อีกทั้งวงจรควบคุมก็จะมีความยุ่งยาก ซับซ้อนมาก จึงได้มีการคิดค้นวิธีที่จะลดลงสายดังกล่าว วิธีหนึ่งก็คือ วิธีการมัลติเพลกซ์

รูปที่ ก.10.1 การแสดงผลด้วย LED แบบเมตริกซ์




รูปที่ ก.10.2 แสดงการแสดงผลด้วยวิธีการมัลติเพลกซ์

จากรูปที่ ก.10.2 ถ้าเราต้องการให้ LEDX ติด สามารถทำได้ โดยการให้ สาย D0 มีสถานะลอจิกเป็น 1 และสาย S0 มีสถานะลอจิกเป็น 0 หรือถ้าต้องการให้ LEDY ติดก็ทำโดยให้ D1

มีสถานะลอจิกเป็น 1 และ S2 มีสถานะลอจิกเป็น 0 ด้วยวิธีการเช่นนี้จะทำให้เราสามารถควบคุมการติดดับของ LED ได้ทุกดวง ( หรือถ้าเราต้องการให้ LEDX และ LEDZ ติดพร้อมกัน ทำโดยให้ D0 , D1 มีลอจิกเป็น 1 และ S0 มีลอจิกเป็น 0) อย่างไรก็ตาม เราไม่สามารถควบคุมให้ LED ติดพร้อมกันหมดได้ การติดพร้อมกันจะทำได้เฉพาะ LED ในแนวตั้งเท่านั้น แต่ถ้าเราควบคุมการติดดับของ LED แต่ละดวง ด้วยความเร็วที่สูงขึ้น จนตาไม่สามารถมองการกระพริบของ LED ได้ทัน เราก็จะเห็นรูปร่างที่ต้องการปรากฏขึ้น ซึ่งการควบคุมนี้มักจะทำโดย ไมโครคอมพิวเตอร์ นอกจากนั้นเรายังสามารถสร้างเป็นภาพหรือตัวอักษรที่เลื่อนได้ โดยใช้เทคนิคการเขียนโปรแกรม

## 3. อุปกรณ์การทดลอง

1. เครื่องคอมพิวเตอร์ IBM PC
2. โปรแกรม TURBO C Version 2.0
3. เมนบอร์ด
4. มอดูลพอร์ต I/O
5. มอดูลตัวแสดงผลแบบเมตริกซ์ LED
6. แหล่งจ่ายไฟ

## 4. หนังสืออ่านประกอบ

1. IBM Technical Reference [10]

## 5. ขั้นตอนการทดลอง

1. ต่อมอดูลตัวแสดงผลแบบเมตริกซ์แอลอีดี เข้ากับชุดทดลอง

2. เขียนโปรแกรม เพื่อแสดงตัวอักษร A บนตัวแสดงผล

3. ทดลองเปลี่ยนความถี่ในการสแกน สังเกตความสว่างของ LED ว่าเปลี่ยนไปหรือไม่

4. เขียนโปรแกรมสร้างตัวอักษรเลื่อนคำว่า "EDL - 100" ให้เลื่อนไปทางซ้าย โดยใช้แบบตัวอักษรจาก Character Generator Table ของ ROM BIOS ซึ่งเริ่มต้นที่หน่วยความจำแอดเดรส F000:FA6E

การทดลองที่ 11

Digital Control System

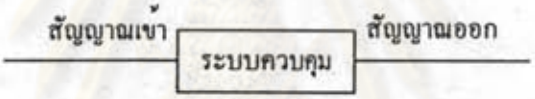
1. วัตถุประสงค์

1. เพื่อเรียนรู้วิธีการนำไมโครคอมพิวเตอร์มาใช้ในระบบควบคุมแบบวงปิด
2. เพื่อทดลองนำไมโครคอมพิวเตอร์มาใช้ในการควบคุมความเร็วของดีซีมอเตอร์ขนาดเล็ก

2. ทฤษฎี

**ระบบควบคุมแบบวงเปิดและวงปิด [31]**

ระบบควบคุมวงเปิด คือ ระบบซึ่งสัญญาณออกไม่ได้นำบล็อกไดอะแกรม ดังแสดงในรูปที่ ก.11.1



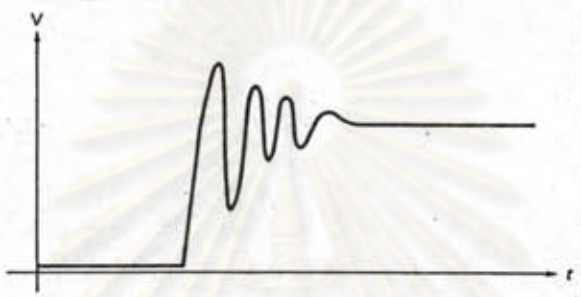

รูปที่ ก.11.1 แผนภาพกรอบระบบควบคุมวงเปิด

ความละเอียดถูกต้องของระบบควบคุมวงเปิด ขึ้นกับการปรับตั้งค่าของชิ้นส่วนต่างๆให้ถูกต้องตามที่ออกแบบ ในบางระบบที่ใช้คนควบคุม ความละเอียดถูกต้องก็ขึ้นกับการทำงานของคนควบคุม ข้อเสียของระบบควบคุมวงเปิดคือ ถ้ามีสิ่งรบกวนเข้าไปในระบบ จะทำให้ได้สัญญาณออกผิดไปทันที

ระบบควบคุมวงปิด คือ ระบบที่นำเอาสัญญาณออกป้อนกลับมาเปรียบเทียบกับสัญญาณเข้า แล้วนำผลที่แตกต่างนี้ไปควบคุมการทำงาน บล็อกไดอะแกรมแสดงในรูปที่ ก.11.2


รูปที่ ก.11.2 แผนภาพกรอบระบบควบคุมวงปิด

การนำเอาสัญญาณออกป้อนกลับเข้าไปช่วยควบคุมการทำงานนี้ ทำให้ระบบทำงานได้ดี แม้ว่าจะมีสิ่งรบกวนเข้าไป และอุปกรณ์ต่างๆที่ใช้ในระบบก็ไม่จำเป็นต้องเป็นชิ้นส่วนที่มีความละเอียดถูกต้องมาก ข้อเสียของระบบควบคุมวงปิดก็คือ อาจเกิดปัญหาเรื่องเสถียรภาพของระบบ หรือเกิดการแกว่ง (oscilate) ได้ดังแสดงในรูปที่ ก.11.3




รูปที่ ก.11.3 ผลตอบของระบบ

**ระบบควบคุมความเร็ว**

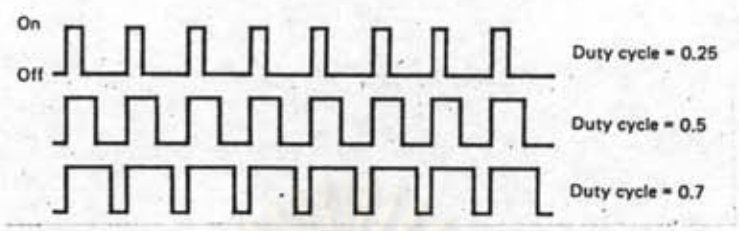
ระบบควบคุมความเร็วที่ใช้ในการทดลองนี้มีแผนภาพ ดังรูป ก.11.4


รูปที่ ก.11.4 แสดงบล็อกไดอะแกรมระบบควบคุมที่ใช้ในการทดลอง

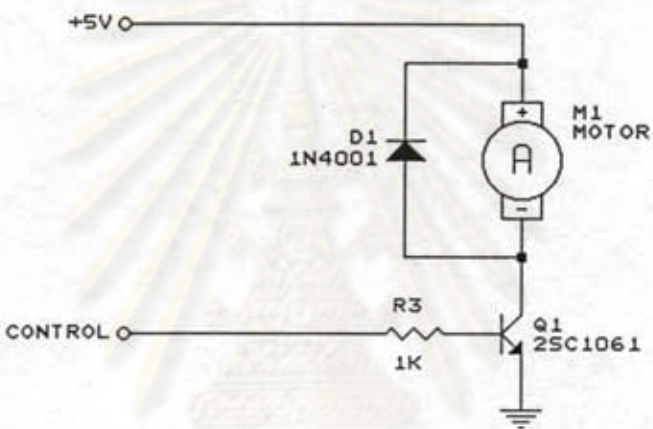
**การควบคุมด้วย Pulse-Width Modulation (PWM) [14]**

ในการทดลองนี้จะควบคุมมอเตอร์ด้วยวิธีแบบ Pulse-Width Modulation (PWM) เพื่อควบคุมระดับแรงดันเฉลี่ยที่ผ่านมอเตอร์ แรงดันที่ผ่านมอเตอร์จะมีลักษณะดังรูปที่ ก.11.5 ค่าเฉลี่ยของแรงดันจะเป็นตัวกำหนดความเร็วของมอเตอร์ ซึ่งค่าเฉลี่ยนี้ขึ้นอยู่กับ duty cycle ของแรงดัน

duty cycle คือสัดส่วนของช่วงเวลาสัญญาณ ON ใน 1 คาบต่อคาบเวลาของสัญญาณ การควบคุม duty cycle จะทำโดยการสั่งงานด้วยซอฟต์แวร์ผ่านวงจร ดังรูปที่ ก.11.6



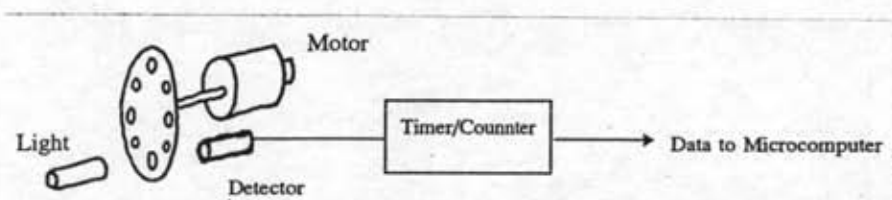

รูปที่ ก.11.5 แรงดันที่จ่ายให้แก่มอเตอร์ด้วยวิธี PWM




รูปที่ ก.11.6 แสดงวงจรที่ใช้ควบคุมมอเตอร์แบบ PWM

**การวัดความเร็ว**

อาศัยทรานส์ดิวเซอร์ที่ทำหน้าที่เปลี่ยน ความเร็วเป็นความถี่ของสัญญาณพัลส์ และสัญญาณพัลส์ที่ได้จะนำไปเข้า Timer / Counter เพื่อทำการวัดคาบเวลาของสัญญาณ ซึ่งทำให้ทราบความถี่ของพัลส์และสามารถหาความเร็วของมอเตอร์ได้ รูปที่ ก.11.7 แสดงวิธีการวัดความเร็วของมอเตอร์




รูปที่ ก.11.7 วิธีการวัดความเร็วของมอเตอร์

โปรแกรมควบคุม

โปรแกรมควบคุมจะประกอบด้วย ส่วนของการอ่านความเร็วของมอเตอร์เข้ามา จาก Timer / Counter และนำไปเปรียบเทียบกับค่า Set Point หรือความเร็วที่ต้องการ แล้วส่งข้อมูลไปยังส่วนของการกำเนิดสัญญาณ PWM (ควบคุมด้วยโปรแกรม) เพื่อทำการปรับแก้ให้ได้ความเร็วตามต้องการ ซึ่งอาจจะใช้ตัวควบคุมแบบต่างๆเช่น P , PI , PID

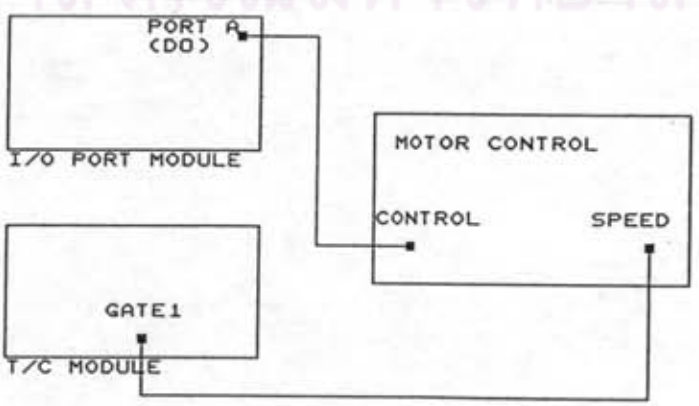
ข้อควรระวัง - ในกรณีใช้ตัวควบคุมแบบ P จะต้องตั้งอัตราขยายให้เหมาะสม มิฉะนั้นอาจทำระบบเกิดการแกว่ง (Oscillation) ได้ในกรณีที่อัตราขยายมากเกินไป หรือ ในกรณีที่อัตราขยายมีค่าน้อยเกินไป จะทำให้เกิด Stead state eroor มาก (ทั้งหมดนี้ต้องควบคุมด้วยซอฟต์แวร์)

## 3. อุปกรณ์การทดลอง

1. เครื่องคอมพิวเตอร์ IBM PC
2. โปรแกรม TURBO C Version 2.0
3. เมนบอร์ด
4. มอดูล T/C
5. มอดูล I/O Port
6. มอดูล Motor Control
7. แหล่งจ่ายไฟ
8. ออสซิลโลสโคป

## 4. ขั้นตอนการทดลอง

1. ต่อมอดูล Motor Control ตามรูป



การวัดความเร็ว

2. เขียนโปรแกรมเพื่อเซตบิต 0 ของพอร์ต A ให้เป็น 1 เพื่อให้มอเตอร์หมุน

3. ทดลองใช้ T/C วัดความถี่ของสัญญาณพัลส์ที่ขั้ว Speed ของมอดูล Motor Control และทดลองใช้ CRO วัดคาบของสัญญาณดังกล่าวเพื่อคำนวณหาความถี่ของสัญญาณ เปรียบเทียบค่าที่วัดได้จาก T/C กับค่าที่คำนวณได้จาก CRO

การควบคุมความเร็ว

4. เขียนโปรแกรมเพื่อควบคุมความเร็วของมอเตอร์ให้คงที่ โดยใช้การควบคุมแบบวงรอบปิด โดยมีสัญญาณป้อนกลับคือ ความเร็วของมอเตอร์ และสัญญาณที่ส่งไปควบคุมมอเตอร์ใช้วิธีการแบบ PWM (pulse - width modulation)

การทดลองที่ 12 : Project

1. วัตถุประสงค์

เพื่อทดลองนำความรู้ที่ได้เรียนมา มาประยุกต์ใช้งานในการควบคุม โดยใช้ไมโครคอมพิวเตอร์

2. ขั้นตอนการทดลอง

ทดลองสร้างชิ้นงานที่อาศัยการควบคุมจากไมโครคอมพิวเตอร์โดยสามารถใช้ชุดทดลองเป็นอุปกรณ์ประกอบได้ ใช้เวลาในการทำไม่เกิน 3 ครั้งการทดลอง

# ภาคผนวก ข

# คู่มือการเขียนโปรแกรมภาษา C

# บทที่ 1
# โปรแกรมภาษา C (C Programming Language)

[24,25,26]

<u>ลักษณะของภาษา/แนะนำภาษา</u>

โปรแกรมภาษา C ได้ถูกพัฒนาขึ้นในช่วงทศวรรษที่ 1970 เพื่อใช้เป็นภาษาพัฒนาระบบยูนิกซ์ มีจุดเด่นตรงที่มีการรวมจุดเด่นของภาษาระดับสูงและภาษาระดับต่ำเข้าด้วยกัน ดังนี้

| ภาษาระดับสูง | ภาษาระดับต่ำ |
|---|---|
| 1. มีลักษณะเป็นโครงสร้างบล็อก<br>2. ไม่ขึ้นกับเครื่องคอมพิวเตอร์ที่ใช้<br>3. ง่ายในการโปรแกรมงานโปรแกรมที่ซับซ้อนได้ง่าย | 1. มีความใกล้ชิดเครื่อง<br>2. ทำงานได้เร็ว |

ปัจจุบัน ภาษา C ได้ถูกพัฒนาไปเป็น C++ ซึ่งมีลักษณะเป็น object-oriented สามารถใช้ในการเขียนโปรแกรมประยุกต์ต่าง ๆ หรือ คอมไพเลอร์

ณ ที่นี้ จะพูดเน้นถึงการใช้ ภาษา C ในงานของการควบคุมด้วยไมโครคอมพิวเตอร์ โดยอิงกับโปรแกรม TURBO C version 2.0 ซึ่งจะพูดถึงโครงสร้างของภาษา C และฟังก์ชันต่าง ๆ ที่เป็นประโยชน์ และการใช้โปรแกรม TURBO C ในการเขียนโปรแกรม โดยจะสมมุติว่า ผู้อ่านมีความรู้ทางด้านการเขียนโปรแกรมมาบ้างแล้ว

# บทที่ 2
# โครงสร้างของโปรแกรมภาษา C

## 2.1 ตัวอย่างโปรแกรม

```
/*********************************************************
*              THIS IS THE COMMENT PART                 *
* FILE_NAME: EXAMPLE.C                                  *
* FUNCTION  : RUNNING LIGHT DISPLAYING                  *
* DATE      : 1995 DEC 18                               *
*********************************************************/

#include <edl.h>                    /* file inclusion */

#define LED 0xc0                    /* macro define */
#define SW  0xc1

int i;                              /* variable & function declaration */
delaya(long speed);

delaya(long speed)                  /* function definition */
{
        long count;
        for (count=0;count<=speed;count++);
}

main( )                             /* main function */
{
        i = 1;
        while (1){
                out(LED,i);
                delaya(100000);
                i *= 2;
                if (i>=256) i=1;
        }
}
```

## 2.2 ส่วนอธิบายโปรแกรม (Program Comment)

ส่วนอธิบายโปรแกรม คือ ส่วนที่ผู้เขียนโปรแกรมสามารถใช้เป็นส่วนอธิบาย การทำงานของโปรแกรม เครื่องคอมพิวเตอร์จะไม่ทำการแปลส่วนดังกล่าว ส่วนอธิบายโปรแกรมนี้ จะไม่มีผลต่อความเร็วในการทำงานของโปรแกรม

ตัวอย่างของส่วนอธิบายโปรแกรม

```
/* This is a comment part */
```

- วิธีใช้:

ขึ้นต้น /* และจบลงด้วย */ และในระหว่างกลาง ให้ใส่ข้อความที่ต้องการ โดยจะต้องไม่ใช่ /* หรือ */

## 2.3 ส่วนของ C Preprocessor

ส่วนของ C Preprocessor เป็นส่วนที่บอกคอมไพเลอร์ ให้รู้ว่าจะนำเอาอะไรมาร่วมในการคอมไพล์โปรแกรมด้วย ประกอบด้วย

### 2.3.1 ส่วน Include ไฟล์

ตัวอย่าง

```
#include <stdio.h>
```

- เป็นการบอกคอมไพเลอร์ให้นำเอาไฟล์ ชื่อ stdio.h มาคอมไพล์ร่วมด้วย และจะเขียนโดยการใช้ #include <stdio.h> หรือ #include "stdio.h" ก็ได้
- มีข้อแตกต่างระหว่างการใช้ < > และ " " คือ ในกรณีที่ใช้ " " คอมไพเลอร์จะทำการค้นหาไฟล์โดยเริ่มจากไดเรกทอรีที่อยู่ในปัจจุบัน แต่ถ้าใช้ < > คอมไพเลอร์จะทำการค้นหาไฟล์โดยเริ่มจากไดเรกทอรี .\INCLUDE
- ชื่อไฟล์จะอยู่ในเครื่องหมาย < > หรือ " "

### 2.3.2 ส่วนของมาโคร

- ตัวอย่าง

```
#define FOREVER for (; ;)      /* infinite loop */
#define PORTA 0X3E
```

- เป็นส่วนที่จะให้คอมไพเลอร์แทนค่าที่อยู่ทางซ้ายมือ ด้วยค่าที่อยู่ทางขวามือในขณะที่ทำการคอมไพล์

### 2.3.3 ส่วน Conditional Inclusion

- ตัวอย่างการใช้

```
# if !defined (HDR)
# define HDR
```

- มีคำต่าง ๆ ที่ใช้ตรงนี้ คือ

```
# if
# endif
# elif (=else if)
# else
```

- ความหมาย คือ ถ้าประโยคที่ตามหลัง if มีค่าความจริงไม่เป็น 0 จะทำคำสั่งที่อยู่ถัดจาก if ไป จนกว่าจะไปเจอคำ endif, elif, else
- ส่วนของ Conditional inclusion นี้ ใช้เพื่อป้องกันไม่ให้มีการนิยามซ้ำซ้อนมากกว่า 1 ครั้ง

## 2.4 ส่วนของการประกาศชนิดของตัวแปร

โดยปกติ ภาษา C จะไม่รู้ตัวแปรใด ๆ จนกว่าจะมีการประกาศไว้ล่วงหน้าก่อน โดยจะต้องมีการระบุชนิดของตัวแปรนั้น ๆ

ส่วนของการประกาศชนิดของตัวแปร มีลักษณะการใช้ เช่น

```
int i, j ;
```

คือ การประกาศตัวแปร i, j โดยให้เป็นตัวแปรชนิดจำนวนเต็ม (int = integer)

การประกาศตัวแปรหลาย ๆ ตัว ที่เป็นชนิดเดียวกัน สามารถใช้ประโยคเดียวกัน และคั่นระหว่างตัวแปรด้วย , และประโยคต้องจบด้วย ; โดยในส่วนนี้ อาจจะไปอยู่ในที่ต่าง ๆ ของโปรแกรมได้ ซึ่งจะอธิบายในรายละเอียดต่อไป

## 2.5 ส่วนของการประกาศชนิดของฟังก์ชัน

ภาษา C จะไม่รู้จักฟังก์ชันใด ๆ จนกว่าจะทำการประกาศชนิดของฟังก์ชันก่อน เช่นเดียวกับ การประกาศชนิดของตัวแปร

ส่วนของการประกาศชนิดของฟังก์ชัน มีลักษณะการใช้ เช่น

```
int getline(x , y) ;
void readport( a ) ;
```

คือ การประกาศให้ฟังก์ชัน getline ส่งค่ากลับชนิดจำนวนเต็ม และ การใช้ void หมายถึงฟังก์ชันนั้น ๆ ไม่มีการส่งค่ากลับ

ภาษา C จะมองโปรแกรมย่อยทุกตัวเป็นฟังก์ชัน หรือ แม้แต่โปรแกรมหลัก ก็เป็นฟังก์ชัน โดยให้ชื่อว่า "ฟังก์ชัน main( )" การทำงานของภาษา C จะเริ่มต้นทำงานที่โปรแกรมหลักก่อน และ ส่วนของการประกาศชนิดของฟังก์ชันนี้ จะต้องจบด้วยเครื่องหมาย ;

## 2.6 ส่วนของการนิยามฟังก์ชัน

ส่วนของการนิยามฟังก์ชัน เป็นส่วนที่บอกว่า ฟังก์ชันนั้น ๆ ประกอบคำสั่ง ขั้นตอนในการทำงานอย่างไร และ การนิยามฟังก์ชัน เราสามารถนิยามสลับกันอย่างไรก็ได้ โดยการนิยามก่อนหลัง จะไม่มีผลต่อการทำงาน

ส่วนของการนิยามฟังก์ชันนี้ ไม่ต้องมีเครื่องหมาย ; ตอนจบประโยค

ตัวอย่าง:

```
int power (int base , int n)
{
        int p ;
        for (p = 1 ; n > 0 ; --n)
                p = p * base ;
        return p ;
```

```
}
```

## 2.7 ส่วนของโปรแกรมหลัก

```
main( )
{
        ส่วนประกอบของโปรแกรมหลัก
}
```

โปรแกรมหลัก ก็คือ ฟังก์ชัน ฟังก์ชันหนึ่ง แต่เป็นฟังก์ชันที่ภาษา C จะมาทำงานที่ฟังก์ชันนี้เป็นฟังก์ชันแรก แล้วภายในฟังก์ชัน จะเรียกใช้งานฟังก์ชันอื่น ๆ ต่อไป และเมื่อทำงานจบโปรแกรมหลัก ก็เป็นอันจบการทำงาน แม้จะมีฟังก์ชันอื่น ๆ เขียนต่อหรือไม่ก็ตาม

อย่างไรก็ตาม ฟังก์ชัน main( ) นี้ อาจอยู่ที่ใดในโปรแกรมก็ได้

# บทที่ 3
# โครงสร้างของภาษา

## 3.1 Control Flow

### 3.1.1 ข้อความ และ บล็อก (Statement and Block)

"ข้อความ" คือ ประโยคที่สมบูรณ์เพียงพอที่บอกให้โปรแกรมรู้ว่าจะทำงานอะไร

ตัวอย่าง:

```
X = 0 ;          เป็นข้อความ
ADDRESS ;        ไม่เป็นข้อความ
printf( x ) ;    เป็นข้อความ
While( i = 0 ) ; ไม่เป็นข้อความ
```

ข้อความในภาษา C จะจบข้อความด้วย ;

"บล็อก" คือ กลุ่มของข้อความ มากกว่า 1 ข้อความ ที่มารวมกัน

ตัวอย่าง:

```
{       X = 0 ;
        i++ ;
        printf( i ) ;
}
```

1) ข้อความที่มารวมกัน จะใช้เครื่องหมาย { } ในการรวมกลุ่มข้อความ (group) ให้เป็นบล็อก ทำให้มีลักษณะเป็นเหมือนข้อความเดียว ฉะนั้น บล็อก ก็เป็นข้อความชนิดหนึ่ง

2) อย่างไรก็ตาม การจบบล็อก ไม่ต้องมีเครื่องหมาย ;

### 3.1.2 ประโยค If-else

การใช้งาน:

```
if (นิพจน์)
        ข้อความ 1
else
        ข้อความ 2
```

ตัวอย่าง:

```
if ( n > 0 )
   X = X+2 ;
else
   X = 0 ;
```

ลักษณะการทำงาน:

1) โปรแกรมจะหาค่าของนิพจน์ออกมา ถ้านิพจน์มีค่าไม่เท่ากับ 0 โปรแกรมจะทำงานในข้อความต่อจากประโยค if ( ) ในที่นี้คือ ข้อความ 1 ตรงกันข้าม ถ้า นิพจน์มีค่าเท่ากับ 0 โปรแกรมจะไปทำงานในข้อความ 2

2) เราสามารถ ตัด else ออกได้ คงเหลือ

```
if (นิพจน์)
   ข้อความ
```

3) เราสามารถใช้ if-else ซ้อนใน if-else ได้ เช่น

```
if ( n = 0 )
        if ( X = 0 )
                X = X+5 ;
        else    X = 0 ;
else
        X = 8 ;
```

4) นอกจากนี้ เราสามารถละข้อความ 1, ข้อความ 2 ได้ ถ้าไม่ต้องการกระทำการใดๆ ถ้าละข้อความ 2 จะต้องตัด else ทิ้งไป

### 3.1.3 ประโยค While

การใช้:

```
While (นิพจน์)
        ข้อความ
```

ตัวอย่าง:

```
While (State)
        X = X+1 ;
```

หรือ

```
While ( X = 0 )
{       y++ ;
        z *= 2 ;
}
```

ลักษณะการทำงาน:

โปรแกรมจะหาค่าของนิพจน์ ถ้านิพจน์ มีค่าไม่เท่ากับ 0 ข้อความจะถูกกระทำ แต่ถ้าเท่ากับ 0 จะจบประโยค While ไปทำคำสั่งต่อจากข้อความ หลังจากนั้น จะหาค่าของนิพจน์ใหม่ และทำเช่นเดิมจนกว่านิพจน์จะมีค่าเท่ากับ 0 และ โปรแกรมจะจบประโยค While ไปกระทำในคำสั่งถัดไป

### 3.1.4 ประโยค Do-while

การใช้:

```
Do      ข้อความ
while (นิพจน์) ;
```

ตัวอย่าง:

```
Do
{       x = inportb(0x3e7)&0x01;
        i ++;
} while( x )
```

ลักษณะการทำงาน:

1) โปรแกรมจะกระทำข้อความ หลังจากนั้นจะทำการหาค่านิพจน์ ถ้านิพจน์มีค่าไม่เท่ากับ 0 จะวนกลับไปกระทำข้อความ แล้ว ทำการหาค่านิพจน์ใหม่ และจะทำซ้ำเช่นเดิม จนกว่านิพจน์จะมีค่าเท่ากับ 0 ก็จะจบประโยค ไปทำคำสั่งถัดไป

2) มีข้อสังเกตว่า ประโยค Do-while จะต้องกระทำ ข้อความอย่างน้อย 1 ครั้ง ผิดกับประโยค While ที่อาจไม่กระทำข้อความเลย ในกรณีที่นิพจน์มีค่าเป็น 0 ตั้งแต่แรก

### 3.1.5 ประโยค For

การใช้:

```
For (ข้อความ 1 ; นิพจน์ ; ข้อความ 2)
        ข้อความ 3
```

ตัวอย่าง:

```
for (i=1; i<40 ; i++)
        a[ i ] = sin( i*2*PI/40 );
```

ลักษณะการทำงาน:

โปรแกรมจะกระทำข้อความ 1 หลังจากนั้น หาค่าของนิพจน์ ถ้านิพจน์มีค่าไม่เท่ากับ 0 จะกระทำข้อความ 3 และตามด้วยข้อความ 2 หลังจากนั้น จะทำการหาค่านิพจน์ใหม่ และทำซ้ำเช่นเดิม จนกว่านิพจน์มีค่าเท่ากับ 0 ก็จะจบประโยค For ไปทำงานในคำสั่งถัดไป แต่ถ้านิพจน์มีค่าเท่ากับ 0 ตั้งแต่แรก ก็จะจบประโยค For ไปทำงานในคำสั่งถัดไป

ข้อสังเกต:

1) ทั้งประโยค If-else , While และ Do-while เราสามารถละส่วนของข้อความได้ ถ้าไม่ต้องการให้โปรแกรมกระทำสิ่งใดในสภาพนั้น ๆ แต่จะต้องมีเครื่องหมาย ;

เช่น
```
if ( X == 0 ) ;
while ( X == 0 ) ;                  /* รอจนกว่า X จะมีค่าไม่เท่ากับ 0 */
For ( ; X == 0 ; ) ;
```

2) ในกรณีของประโยค for เราสามารถละส่วนของนิพจน์ได้ ซึ่งหมายถึง นิพจน์มีค่าเป็นจริงตลอด (ไม่เท่ากับ 0 )

เช่น
```
for ( ; ; )
```

คือ การวนรอบนิรันดร์

### 3.1.6 ประโยค Switch-case

การใช้:

```
Switch (นิพจน์)
{       case นิพจน์ ค่าคงที่ 1 : ข้อความ 1
        case นิพจน์ ค่าคงที่ 2 : ข้อความ 2
                :
                :
        default : ข้อความ n
}
```

ตัวอย่าง:

```
x = getchar( );
switch( x )
{       case 0 : outportb(0xc3,0);
        case 1 : outportb(0xc3,1);
        case 2 : outportb(0xc3,2);
}
```

ลักษณะการทำงาน:

โปรแกรมจะทำการหาค่าของนิพจน์ และ ทำการเปรียบเทียบกับนิพจน์ค่าคงที่ โดยเริ่มเปรียบเทียบกับนิพจน์ ค่าคงที่ 1 ถ้ามีค่าเท่ากัน จะกระทำข้อความ 1 หลังจากนั้น มาเปรียบเทียบนิพจน์กับนิพจน์ค่าคงที่ 2 ถ้ามีค่าเท่ากัน จะกระทำข้อความ 2 ทำเช่นนี้ไปเรื่อย ๆ ถ้านิพจน์มีค่าไม่เท่ากับกรณีใด ๆ เลย โปรแกรมก็จะทำข้อความ n ที่อยู่ที่บรรทัด default บรรทัดนี้ เราสามารถละได้

### 3.1.7 คำสั่ง Break

คำสั่ง Break คือ คำสั่งที่ควบคุมการทำงานของโปรแกรมให้ออกจากบล็อก while , for , do-while ที่คำสั่ง break วางอยู่ ทันทีที่ทำงานมาถึงคำสั่งนี้ โดยไม่มีเงื่อนไข

ตัวอย่าง:

```
while ( X > 0 )
{       X = X x 2 ;
        break ;
}
```

เมื่อโปรแกรมทำงานมาถึงคำสั่ง break โปรแกรมก็จะกระโดดออกไปทำงานคำสั่งถัดจาก while ทันที ไม่ว่า นิพจน์ X > 0 จะมีค่าเป็นอย่างไร เรามักใช้คำสั่ง break กับประโยค Switch เช่น

```
Switch ( C )
{
        case 0 : X[C]++ ;
        break ;
        case 1 : X[C]- - ;
        break ;
        case 2 : X[C] = 0 ;
}
```

### 3.1.8 คำสั่ง Goto

การใช้:

```
goto   label  ;
          :
          :
label:    :
          :
```

เป็นคำสั่งที่ทำให้โปรแกรมกระโดดไปทำงานในบรรทัดที่ชื่อตรงกับชื่อที่อยู่หลังคำว่า goto ชื่อในบรรทัดที่กระโดดไป จะต้องตามหลังด้วยเครื่องหมาย :

### 3.1.9 คำสั่ง Continue

จะทำงานตรงกันข้ามกับคำสั่ง break คือ จะทำให้โปรแกรมย้อนกลับไป ทำงานยังต้นบล็อกของประโยค while, for หรือ do-while โดยทำในรอบการทำงานถัดไป

เช่น:

```
while ((C = getchar( )) ! = -1)
{       if ( C >= '0' && C <= '9' )
                continue ;
        else   putchar(C) ;
}
```

โปรแกรมจะทำการอ่านตัวอักษรเข้ามาและพิมพ์ออก ถ้าเป็นตัวเลข ก็จะอ่านตัวอักษรใหม่ทันที โดยไม่พิมพ์ออก

## 3.2 ชนิดของตัวแปร

ภาษา C จะยังไม่รู้จักตัวแปรใด ๆ จนกว่าจะมีการประกาศไว้ก่อนล่วงหน้า ตัวแปรในภาษา C สามารถแบ่งออกได้เป็นชนิดต่าง ๆ คือ

| | |
|---|---|
| char | เป็นตัวแปรขนาด 1 ไบต์ ใช้เก็บตัวอักขระขนาด 1 ตัว |
| int | เป็นตัวแปรจำนวนเต็ม |
| float | เป็นตัวแปรทศนิยม (จำนวนจริง) |
| double | เป็นตัวแปรทศนิยม ขนาดความละเอียด 2 เท่า |

ตัวแปรแต่ละชนิด จะมีขนาดกี่ไบต์ ขึ้นอยู่กับเครื่องคอมพิวเตอร์ แต่สำหรับ TURBO C บนไอบีเอ็มพีซี นั้น จะมีขนาดดังแสดงในตาราง ซึ่งได้แสดงช่วงของค่าที่สามารถเก็บได้ไว้ด้วย

ยังมีคำที่ใช้เติมนำหน้าชนิดของตัวแปร คือ signed และ unsigned ซึ่งใช้เติมนำหน้าตัวแปรได้ทุกชนิด โดยตัวแปรยังมีขนาดเท่าเดิม กรณีไม่ระบุจะหมายถึง ชนิด signed

| | | |
|---|---|---|
| signed | คือ | ตัวแปรชนิดที่มีเครื่องหมาย |
| unsigned | คือ | ตัวแปรชนิดที่ไม่มีเครื่องหมาย(ค่าบวกอย่างเดียว) |

นอกจากนี้ ยังมีคำ short และ long ที่ใช้เติมนำหน้าตัวแปรชนิด int ทำให้ตัวแปรมีขนาดแตกต่างกัน ถ้าใช้คำทั้งสอง เราสามารถละคำว่า int ได้ กรณีเขียนเพียง int จะหมายถึง short int

เราสามารถสรุปชนิดของตัวแปรได้ดังตาราง[19]

Turbo C Data Types , Sizes , and Ranges

| Type | Size (bits) | Ranges |
|---|---|---|
| unsigned char | 8 | 0 - 255 |
| char | 8 | (-128) - 127 |
| enum | 16 | (-32768) - 32767 |
| unsigned short | 16 | 0 - 65535 |
| short | 16 | (-32768) - 32767 |
| unsigned int | 16 | 0 - 65535 |
| int | 16 | (-32768) - 32767 |
| unsigned long | 32 | 0 - 4294967295 |
| long | 32 | (-2147483648) - 2147483647 |
| float | 32 | 3.4E-38 - 3.4E+38 |
| double | 64 | 1.7E-308 - 1.7E+308 |
| long double | 64 | 1.7E-308 - 1.7E+308 |
| pointer | 16 | (near,_cs,_ds,_es,_ss pointers) |
| pointer | 32 | (far,huge pointers) |

สำหรับตัวแปรชนิด pointer จะได้กล่าวถึงในรายละเอียดต่อไป

ตัวแปรชนิด long double มีความหมายเช่นเดียวกับ double

## 3.3 ชนิดของค่าคงที่

มีลักษณะเช่นเดียวกับชนิดของตัวแปร โดยเราสามารถเติมอักษร L หรือ U ต่อท้ายสำหรับค่าคงที่ชนิด int และสามารถใช้ทั้ง L และ U พร้อมกันได้ สำหรับค่าคงที่ 1 ตัว

| | | |
|---|---|---|
| L | หมายถึง | long |
| U | หมายถึง | unsigned |
| F | หมายถึง | float |

ตัวอย่าง:

1234L จะหมายถึงค่าคงที่ 1234 ที่เป็นชนิด long แทนที่จะเป็นชนิด int

5345F หมายถึงค่าคงที่ 5345 ที่เป็นชนิด float

สำหรับค่าคงที่ทศนิยม โดยปกติจะหมายถึง ค่าคงที่ชนิด double อย่างไรก็ตาม เราสามารถทำให้เป็นชนิด float ได้ โดยการใช้ตัวอักษร F ต่อท้าย เช่น 2.34F

ข้อควรระวัง:

กรณีที่ระบุค่าให้แก่ ค่าคงที่ หรือ ตัวแปร มากกว่าที่สามารถรับได้ จะเกิด overflow โดยไม่มีการเตือน ผลลัพธ์ของค่าที่ได้จะเป็น บิตต่ำของผลลัพธ์ที่เป็นจริง

## 3.4 การเปลี่ยนชนิด

### การใช้ CAST

ในภาษา C เราสามารถทำการเปลี่ยนชนิดของค่าคงที่หรือตัวแปรได้ทันที โดยการเขียนชื่อชนิดไว้ในวงเล็บ นำหน้าค่าคงที่หรือตัวแปรที่ต้องการเปลี่ยนชนิด

```
(type) expression;
```

ตัวอย่าง:

```
(int) x ;
(float) (x+0.5) ;
average = (double) (sum/n);
ptr  =  (char *) 0xb8000000 ;     /* ptr is a pointer variable */
```

## 3.5 ชื่อของตัวแปร (Identifier)

1) สามารถประกอบด้วย ตัวอักษร A-Z, a-z, ตัวเลข 0-9, ตัว underscore ( _ ) และสามารถใช้เครื่องหมาย $

2) ชื่อตัวแปรจะต้องขึ้นด้วยตัวอักษร หรือ underscore

3) ตัวแปรสามารถตั้งชื่อยาว ๆ ได้ แต่ จะถือตัวอักษร 32 ตัวแรกเท่านั้น ที่มีนัยสำคัญ อย่างไรก็ตาม เราสามารถกำหนดได้ว่า จะให้จำนวนตัวอักษรที่มีนัยสำคัญเป็นเท่าไร

## 3.6 ตัวดำเนินการ (Operators)

### 3.6.1 ตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operators)

ตัวดำเนินการทางคณิตศาสตร์ ประกอบด้วย

| | | |
|---|---|---|
| + | ตัวดำเนินการ | บวก |
| - | " | ลบ |
| * | " | คูณ |
| / | " | หาร |
| % | " | modulus (ให้ค่าเศษที่เหลือจากการหาร) |

### 3.6.2 ตัวดำเนินการทางตรรก (Logical Operators)

ตัวดำเนินการทางตรรก ประกอบด้วย

| | | |
|---|---|---|
| > | ตัวดำเนินการ | มากกว่า |
| < | " | น้อยกว่า |
| >= | " | มากกว่า หรือ เท่ากับ |
| <= | " | น้อยกว่า หรือ เท่ากับ |
| == | ตัวดำเนินการ | เท่ากับ |
| != | " | ไม่เท่ากับ |
| && | " | และ |
| \|\| | " | หรือ |

### 3.6.3 ตัวดำเนินการ การเพิ่มค่า และ ลดค่า

ตัวดำเนินการการเพิ่มค่าและลดค่า จะใช้เครื่องหมาย

++ หมายถึง เพิ่มค่าขึ้น 1

-- หมายถึง ลดค่าลง 1

ตัวอย่าง:

```
++n ;
```

เราสามารถใช้เครื่องหมายทั้งสอง นำหน้า หรือ ตามหลัง (Prefix or Postfix)

เช่น:

```
++n ;   หรือ    n++ ;
```

มีข้อแตกต่าง คือ สำหรับ ++n ค่าของ n จะถูกเพิ่มค่าก่อนที่จะนำค่า n ไปใช้ ส่วน n++ ค่าของ n จะถูกนำไปใช้ก่อน หลังจากนั้นจึงค่อยเพิ่มค่า n

ตัวอย่าง:

ถ้า n มีค่าเท่ากับ 5

```
x = n++ ;
```

x จะมีค่าเท่ากับ 5

```
x = ++n ;
```

x จะมีค่าเท่ากับ 6

### 3.6.4 ตัวดำเนินการบนบิต (Bitwise Operators)

ตัวดำเนินการบนบิต ประกอบด้วย

| | | |
|---|---|---|
| & | ตัวดำเนินการ | AND |
| \| | " | OR |
| ^ | ตัวดำเนินการ | Exclusive OR |
| << | shift left | |
| >> | shift right | |
| ~ | 1's complement | |

เรามักใช้ &(AND) ในการมาสก์บิต หรือ |(OR) ในการเซ็ตบิต และ ^(EX-OR) ในการ toggle บิต

### 3.6.5 ตัวดำเนินการกำหนดค่า (Assignment Operators)

ตัวดำเนินการกำหนดค่า ใช้เครื่องหมาย = เช่น

```
y = x+5;
```

หรือใช้เครื่องหมาย op =

โดย op เป็นตัวดำเนินการ ทางคณิตศาสตร์ หรือ ตัวดำเนินการบนบิต

| | |
|---|---|
| ตัวอย่าง: | i+ = 5 ; |
| โดยถ้า | x op = y |
| จะมีความหมายเท่ากับ | x = x op y |
| เช่น | i* = 5 ; |
| มีความหมายเหมือนกับ | i = i*5 ; |

การใช้ comma operator ( , )

ตัวอย่าง:

```
if( x=0 , y=3 , z<0) ...
```

นิพจน์จะถูกดำเนินการจากซ้ายไปขวา และค่าของนิพจน์สุดท้ายที่ดำเนินการจะเป็นค่าของทั้งวงเล็บ เช่น ตัวอย่างข้างต้น ประโยค if จะทำการตรวจสอบนิพจน์ z < 0

### 3.6.6 ตัวดำเนินการเงื่อนไข

รูปแบบ: expr1 ? expr2 : expr3

ให้ความหมายเช่นเดียวกับ

```
if (expr1)
        expr2
else
        expr3
```

ตัวอย่างเช่น

```
Y = (X<1)?3:5;
```

Y จะมีค่าเป็น 3 ถ้า X มีค่าน้อยกว่า 1 มิฉะนั้นจะมีค่าเป็น 5

## 3.7 นิพจน์ (Expression)

นิพจน์ คือ ประโยคที่สามารถหาค่าความจริงได้ ในภาษา C จะถือว่า ค่าใดก็ตามที่มีค่าไม่เท่ากับ 0 จะมีค่าความจริงเป็นจริง และถ้าค่าเท่ากับ 0 จะมีค่าความจริงเป็นเท็จ ดังนั้น ตัวแปรที่กำหนดค่าแล้ว หรือ ค่าคงที่ ก็ถือเป็นนิพจน์ได้ เช่น

ถ้า x มีค่าเท่ากับ 5

x เป็น นิพจน์

นิพจน์หลาย ๆ นิพจน์ ที่มาดำเนินการด้วยตัวดำเนินการ ก็ยังเป็นนิพจน์ เช่น

x = x+1 เป็น นิพจน์

## 3.8 ลำดับของการดำเนินการและความสำคัญ

ตารางข้างล่าง แสดงลำดับของการดำเนินการ และความสำคัญของตัวดำเนินการ ตัวดำเนินการที่อยู่สูงกว่าในตาราง จะมีความสำคัญมากกว่าและจะถูกดำเนินการหลังตัวดำเนินการที่มีความสำคัญต่ำกว่า ตัวดำเนินการที่มีความสำคัญมากกว่าจะถูกดำเนินการก่อน และในบรรทัดเดียวกันของตาราง มีความสำคัญเท่ากัน แต่จะดำเนินการก่อน-หลังตามลำดับการเขียนตัวดำเนินการโดยเรียงจากซ้ายไปขวา หรือ ขวาไปซ้าย ตามที่เขียนในช่องลำดับการดำเนินการ

ตารางแสดงความสำคัญของตัวดำเนินการและลำดับการดำเนินการ [25]

| ตัวดำเนินการ | ลำดับการดำเนินการ |
|---|---|
| () [] -> . | ซ้ายไปขวา |
| ! ~ ++ -- + - * & (type) sizeof | ขวาไปซ้าย |
| * / % | ซ้ายไปขวา |
| + - | ซ้ายไปขวา |
| << >> | ซ้ายไปขวา |
| < <= > >= | ซ้ายไปขวา |
| == != | ซ้ายไปขวา |
| & | ซ้ายไปขวา |

ตารางแสดงความสำคัญของตัวดำเนินการและลำดับการดำเนินการ (ต่อ)

| ตัวดำเนินการ | ลำดับการดำเนินการ |
|---|---|
| ^ | ซ้ายไปขวา |
| \| | ซ้ายไปขวา |
| && | ซ้ายไปขวา |
| \|\| | ซ้ายไปขวา |
| ?: | ขวาไปซ้าย |
| = += -= *= /= %= &= ^= \|= <<= >>= | ขวาไปซ้าย |
| , | ซ้ายไปขวา |

# บทที่ 4
# ฟังก์ชันในภาษา C และ โครงสร้างของโปรแกรม

## 4.1 ประโยชน์ของฟังก์ชัน

1) ทำให้สามารถแบ่งงานใหญ่ ๆ ลงเป็นงานย่อย ๆ ได้
2) ทำให้งานที่มีลักษณะคล้าย ๆ กัน สามารถเขียนทีเดียวได้ และเรียกใช้ซ้ำ โดย
ส่วน ที่แตกต่างกัน ก็คือ ค่าพารามิเตอร์ที่ส่งให้แก่ฟังก์ชันนั่นเอง
3) ทำให้สามารถสร้างงานบนฐานที่ผู้อื่นได้ทำมา ไม่ต้องเริ่มต้นใหม่ทั้งหมด

## 4.2 การใช้งานฟังก์ชัน

ตัวอย่าง:

```
int  sum(a, b, c);
int  sum(a , b , c)
{
        return(a+b+c);
}
main( )
{
        int x = 1, y = 2, z = 3;
        printf("%d ", sum(x, y, z));
}
```

1) การนิยามฟังก์ชัน จะมีรูปแบบ ดังนี้

```
ชนิดของค่าที่ส่งกลับ  ชื่อฟังก์ชัน(การประกาศตัวแปรอาร์กิวเมนต์)
        {
                ส่วนของการนิยาม หรือ ข้อความ
        }
```

2) การใช้งาน ฟังก์ชัน จะต้องทำการประกาศชนิดของฟังก์ชันก่อน (ดูหัวข้อ 2.5) หลังจากนั้น จึงเริ่มต้นนิยามฟังก์ชัน (หัวข้อ 2.6)

3) ค่าที่ส่งให้แก่ฟังก์ชัน จะอยู่ในเครื่องหมาย ( ) เราเรียกว่า "ค่าพารามิเตอร์" อาจเป็นตัวแปร หรือ ค่าคงที่ ก็ได้ และฟังก์ชันจะรับค่าผ่านตัวแปร ซึ่งเรียกว่า "อาร์กิว-เมนต์" ค่าพารามิเตอร์และอาร์กิวเมนต์แต่ละตัว คั่นด้วยเครื่องหมาย , ตัวอย่างเช่น

```
power (10, z) ;          /* ส่วนเรียกใช้งานฟังก์ชัน */
Float   power (x, y)     /* ส่วนนิยามฟังก์ชัน */
```

คือ การส่งค่า 10 ให้แก่ตัวแปร x และค่า z ให้แก่ y ค่า 10 และ z เป็น พารามิเตอร์ x, y เป็น อาร์กิวเมนต์

4) การเรียกใช้งานฟังก์ชัน ถือเป็นข้อความ ๆ หนึ่ง จะต้องตามหลังด้วยเครื่องหมาย ;

5) ส่วนของการประกาศชนิดของฟังก์ชัน จะต้องจบด้วยเครื่องหมาย ; แต่ส่วนของการนิยามฟังก์ชัน ไม่มีเครื่องหมาย ;

6) การส่งผ่านค่าพารามิเตอร์ ให้แก่ฟังก์ชัน จะส่งผ่านแบบที่เรียกว่า ส่งผ่านแบบผ่านค่า (passed by value) คือ ค่าที่ส่งมาให้แก่อาร์กิวเมนต์ของฟังก์ชัน จะถูกเก็บในตัวแปรที่เป็นคนละตำแหน่งกับค่าพารามิเตอร์เดิม นั่นคือ ถ้ามีการเปลี่ยนแปลงใด ๆ ของอาร์กิวเมนต์ในฟังก์ชัน จะไม่มีผลต่อค่าตัวแปรเดิมที่ส่งค่ามาให้

7) ฟังก์ชันที่ไม่มีการส่งค่าพารามิเตอร์ให้ ก็ยังต้องมีเครื่องหมาย ( ) ตามหลังชื่อฟังก์ชัน เช่น

```
clrscr( ) ;          /* ส่วนเรียกใช้งานฟังก์ชัน */
void clrscr( ) ;     /* ส่วนนิยามฟังก์ชัน */
```

8) ส่วนของการประกาศชนิดของฟังก์ชัน มีรูปแบบ คือ

ชนิดของค่าที่ส่งกลับ ชื่อฟังก์ชัน (การประกาศตัวแปรอาร์กิวเมนต์) ;

เช่น:

```
double atof( char s[ ] ) ;
```

9) การส่งค่ากลับจากฟังก์ชัน

จะใช้คำสั่ง return ตามด้วยค่าที่ต้องการส่งกลับ

เช่น return(result);

ค่าที่ส่งกลับจะถูกเปลี่ยนให้เป็นชนิดเดียวกับฟังก์ชันก่อน แล้วจึงทำการส่งค่า

## 4.3 ชนิดของฟังก์ชัน

ฟังก์ชัน มี 3 ชนิด คือ ฟังก์ชันแบบไม่มีการส่งค่ากลับ ฟังก์ชันแบบส่งค่ากลับ ฟังก์ชันแบบอินเตอร์รัปต์รูทีน มีรายละเอียดดังต่อไปนี้

### 4.3.1 แบบไม่มีการส่งค่ากลับ

จะใช้คำสั่ง void นำหน้าชื่อ ของฟังก์ชัน ในการประกาศชนิดของฟังก์ชัน อย่างไรก็ตาม เราสามารถละคำ void นี้ได้ เช่น

```
void  out(address , data);
```

หรือ

```
out(addrsss , data);
```

### 4.3.2 แบบส่งค่ากลับ

จะใช้ ชนิดของค่าที่ส่งกลับ นำหน้าชื่อฟังก์ชัน เช่น

```
int     input(ADDRESS) ;
float   calc(double x , double y);
```

### 4.3.3 แบบอินเตอร์รัปต์รูทีน[19]

แบบอินเตอร์รัปต์รูทีน คือ ฟังก์ชันที่ถูกเรียกใช้งาน เมื่อเกิดการอินเตอร์รัปต์ขึ้น อันที่จริงฟังก์ชันชนิดนี้ก็จัดอยู่ในชนิดของฟังก์ชันสองชนิดบน แต่ในที่นี้ได้กล่าวแยกออกมาเพื่อให้เห็นชัดเจนมากยิ่งขึ้น

ฟังก์ชันชนิดนี้ไม่ได้เป็นมาตรฐานของภาษา C การใช้งานจะใช้ คำว่า "interrupt" นำหน้าชื่อ ฟังก์ชัน และถ้าฟังก์ชันมีการส่งค่ากลับด้วย ก็ให้ บอกชนิดของค่าที่ส่งกลับ นำหน้า ถ้าไม่มีการส่งค่ากลับ ให้ใช้ void เช่น

```
int interrupt getsignal( ) ;
void interrupt output(address) ;
```

## 4.4 ขอบเขตของตัวแปร

ตัวแปรในภาษา C ถ้าแบ่งตามขอบเขตของตัวแปร จะมี 4 ชนิด คือ

1. External Variables
2. Automatic Variables
3. Static Variables
4. Register Variables

### 4.4.1 ตัวแปรชนิด External

ตัวแปรชนิด External คือ ตัวแปรที่ทุก ๆ ส่วนในโปรแกรมสามารถเรียกใช้ได้ เป็นตัวแปรชนิดเดียวกับตัวแปรชนิด global ในภาษาปาสคาล หรือ ตัวแปรชนิด common ในภาษาฟอร์แทรน ขอบเขตของการใช้งานของตัวแปร จะเริ่มต้นที่จุดที่ประกาศชนิดตัวแปรเป็นต้นไป ตัวแปรชนิดนี้สามารถถูกเรียกใช้งานจากไฟล์อื่น ๆ ที่เชื่อม (link) เข้าด้วยกันได้ด้วย

วิธีการประกาศตัวแปรชนิดนี้ ทำโดยการประกาศตัวแปรที่ด้านนอกของฟังก์ชันทุก ๆ ฟังก์ชัน โดยอาจใช้คำว่า "extern" ในการระบุ หรือไม่ใช้ก็ได้ เช่น

```
main( )
{       i = i+2 ;
        j = j-1 ;
}
int k ;         /* k มีขอบเขตการใช้งานตั้งแต่จุดนี้เป็นต้นไป */
```

```
atof(x) ;
{       :
        :
}
```

k เป็นตัวแปรชนิด External อาจระบุได้ด้วย extern int k ;

### 4.4.2 ตัวแปรชนิด Automatic (หรือ ตัวแปรชนิด Internal)

ตัวแปรชนิด Automatic หรือ ตัวแปรชนิด Internal คือ ตัวแปรที่ถูกสร้างขึ้นใช้งานภายในบล็อกหนึ่ง ๆ และจะสิ้นสุดการใช้งานลงเมื่อจบบล็อก บล็อกนั้นอาจเป็นฟังก์ชัน หรือ เป็นบล็อกธรรมดาก็ได้ ตัวแปรชนิดนี้ไม่สามารถถูกเรียกใช้งานได้จากบล็อกอื่น ๆ (เป็นตัวแปรแบบ local) ขอบเขตการใช้งานของตัวแปร จะเริ่มต้นที่จุดที่เริ่มประกาศตัวแปรไปจนจบบล็อก

วิธีการประกาศตัวแปรชนิดนี้ ทำโดยการประกาศตัวแปรภายในบล็อกหนึ่ง ๆ ซึ่งมักจะไว้ที่ต้นบล็อก ตัวแปรชนิดนี้ ถ้าผู้เขียนไม่ได้กำหนดค่าเริ่มต้นไว้ ตัวแปรจะมีค่าไม่แน่นอน ขึ้นอยู่กับว่า เครื่องจะไปจัดสรรเนื้อที่ในหน่วยความจำบริเวณใด

ในกรณีที่มีการระบุตัวแปรชนิดนี้ โดยใช้ชื่อซ้ำกับตัวแปรชนิด External เมื่อมีการเรียกใช้งานตัวแปร ภายในบล็อกที่อยู่ในขอบเขตการใช้งานของตัวแปรชนิด automatic ตัวแปรที่ถูกใช้งาน จะคือ ตัวแปรชนิด automatic และไม่มีผลต่อตัวแปรชนิด external

- การระบุค่าเริ่มต้นของตัวแปร

เช่น:

```
int  i = 10 ;
Void  Fn( )
{       int i ;
        i = 1 ;
        Printf( "i = %d" , i ) ;
}
main( )
{       Fn( ) ;
        Printf( "i = %d" , i ) ;
}
```

จะให้ผลลัพธ์ คือ การพิมพ์ข้อความ

i = 1

i = 10

เมื่อสิ้นสุดการทำงานของบล็อก เครื่องจะทำการปลดปล่อยหน่วยความจำบริเวณที่เก็บตัวแปรทิ้งไป และเมื่อมีการกลับมาเรียกใช้งานตัวแปรชนิดนี้อีก (กลับเข้ามาทำงานในบล็อกใหม่) เครื่องจะทำการจัดสรรหน่วยความจำใหม่ ฉะนั้น ค่าที่ได้อาจไม่ใช่ค่าเดิมกับค่าในขณะออกจากบล็อกไป

### 4.4.3 ตัวแปรชนิด Static

ในการใช้งานตัวแปรชนิด Automatic ดังที่กล่าวไปแล้วว่า เมื่อมีการกลับมาใช้งานตัวแปรเดิมอีก ค่าที่ได้จะมีค่าไม่แน่นอน เราสามารถทำให้ค่าเดิมยังคงค้างอยู่ได้ โดยการระบุตัวแปรชนิด Static ส่วนลักษณะอื่น ๆ จะเหมือนกับตัวแปรชนิด automatic

รูปแบบการใช้งาน:

```
static  int  i ;
```

เราสามารถใช้ตัวแปรชนิด Static นี้ ในลักษณะของตัวแปร external ได้ด้วย แต่มีข้อแตกต่าง คือ ขอบเขตการใช้งานของตัวแปรจะอยู่แต่ภายในไฟล์ของโปรแกรมที่เขียนเท่านั้น โปรแกรมที่อยู่ในไฟล์อื่น ๆ ไม่สามารถเรียกใช้งานได้

### 4.4.4 ตัวแปรชนิด Register

ตัวแปรชนิด Register มีลักษณะการใช้งานเช่นเดียวกับ ตัวแปรชนิด automatic ต่างกันตรงที่ เครื่องจะทำการจัดสรรเนื้อที่ให้แก่ตัวแปร โดยใช้รีจิสเตอร์ของเครื่อง มีประโยชน์ในการเพิ่มความเร็วในการทำงาน การระบุตัวแปรชนิดนี้ มีขีดจำกัด ขึ้นกับเครื่องแต่ละเครื่อง เช่น บางเครื่อง ระบุได้ไม่เกิน 3 ตัว และชนิดของตัวแปรต้องเป็นแบบ int หรือ char หรือ pointer เท่านั้น กรณีที่รีจิสเตอร์มีไม่พอกับความต้องการ เครื่องจะทำการเก็บตัวแปรดังกล่าวไว้ในหน่วยความจำให้โดยอัตโนมัติ

ตัวอย่างการใช้งาน:

```
register int x ;
register char c ;
```

## 4.5 การกำหนดค่าเริ่มต้นให้แก่ตัวแปร

ภาษา C จะกำหนดค่าเริ่มต้นให้กับตัวแปรชนิด External และ Static ให้มีค่าเป็นศูนย์โดยอัตโนมัติ ส่วนตัวแปรแบบ Automatic และ Register ไม่ถูกกำหนด (undefined) มีค่าไม่แน่นอน

เราสามารถกำหนดค่าเริ่มต้นของตัวแปรไปพร้อม ๆ กับ การประกาศชนิดของตัวแปร โดยมีลักษณะ ดังนี้

```
int  i = 1 ;
char  st = '$' ;
int  x = 50*4 ;
int  a[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0} ;
char  st[ ] = "TEST" ;    มีความหมายเทียบเท่ากับ
char  st[ ] = {'T', 'E', 'S', 'T', '\0'} ;
```

# บทที่ 5
# พอยน์เตอร์ในภาษา C

การใช้พอยน์เตอร์มีบทบาทสำคัญมากในภาษา C และเป็นจุดเด่นประการหนึ่งของภาษาการใช้พอยน์เตอร์ทำให้เราสามารถทำงานได้ในหลายๆ ลักษณะที่ไม่สามารถใช้วิธีการธรรมดาได้ พอยน์เตอร์ยังช่วยลดขนาดของโปรแกรม และได้ Code ของโปรแกรมที่มีขนาดเล็กลง แต่มีประสิทธิภาพสูงขึ้น ฟังก์ชันมากมายในภาษา C ถูกเรียกใช้ในลักษณะการใช้พอยน์เตอร์ ฉะนั้นความเข้าใจเกี่ยวกับการใช้พอยน์เตอร์จึงเป็นสิ่งสำคัญ

ตัวแปรทั่วไปจะทำการเก็บค่าลงในหน่วยความจำโดยเก็บค่าที่ใช้งานของตัวแปร เช่น X มีค่าเท่ากับ 5 ณ ตำแหน่งหน่วยความจำของตัวแปร X จะเก็บค่า 5 ไว้ หรือ Y มีค่าเท่ากับ "$" ณ ตำแหน่งหน่วยความจำของตัวแปร Y ก็จะเก็บค่ารหัส "$" ไว้ แต่สำหรับตัวแปรพอยน์เตอร์ เป็นตัวแปรที่ไม่ได้เก็บค่าใช้งานไว้โดยตรง แต่จะเก็บค่าแอดเดรสของหน่วยความจำที่ใช้เก็บค่าใช้งานของตัวแปร เช่น Pt เป็นตัวแปรพอยน์เตอร์ เก็บค่าแอดเดรสของตัวแปร X ที่มีค่าเท่ากับ 5 ในลักษณะนี้เราจะเรียกว่า Pt เป็นตัวแปรพอยน์เตอร์ที่ชี้ไปยังตัวแปร X

## 5.1 ตัวแปรชนิดพอยน์เตอร์

ตัวแปรชนิดพอยน์เตอร์ ก็มีลักษณะเช่นเดียวกับตัวแปรแบบอื่นๆ คือ จะต้องมีการประกาศตัวแปรก่อน โดยชนิดของตัวแปรจะขึ้นกับว่าตัวแปรพอยน์เตอร์นั้นชี้ไปยังค่าชนิดใด ตัวอย่าง สมมุติให้ Pi ชี้ไปยังค่าที่เป็นจำนวนเต็ม จะประกาศตัวแปรได้โดย

```
int *Pi ;
```

ให้ PR ชี้ไปยังจำนวนทศนิยม จะประกาศได้ดังนี้

```
float *PR ;
```

และ เมื่อต้องการนำค่าที่ตัวแปรพอยน์เตอร์ชี้อยู่ ออกมาใช้งาน จะเขียนได้ว่า

```
X = *Pi
Y = *PS
```

จะทำให้ X มีค่าเท่ากับค่าที่ Pi ชี้อยู่ และ Y เท่ากับค่าที่ PS ชี้อยู่

การกำหนดให้พอยน์เตอร์ชี้ไปยังตัวแปรใด สามารถเขียนได้ดังนี้

```
PS = &Y          ( PS เป็นพอยน์เตอร์ )
```

จะทำให้ PS ชี้ไปยังตัวแปร Y หรือความหมายคือ PS เก็บค่าแอดเดรสของตัวแปร Y

มีข้อควรระวัง คือ เราไม่สามารถกำหนดให้พอยน์เตอร์ชี้ไปยังจำนวนที่ไม่ใช่ตัวแปรได้ เช่น

```
PS = &3          /* ใช้ไม่ได้ */
```

และไม่สามารถใช้พอยน์เตอร์ชี้ไปยังตัวแปรแบบรีจิสตอร์ได้ เราสามารถนำพอยน์เตอร์มาประกอบเป็นนิพจน์ได้ เช่น

```
a = *Pi + 3 ;
```

จะกำหนดให้ a มีค่าเท่ากับค่าที่ Pi ชี้อยู่ บวกด้วย 3

และยังสามารถเขียนค่าที่พอยน์เตอร์ชี้ไปไว้ทางด้านซ้ายของประโยคกำหนดค่าได้ เช่น จากตัวอย่างเดิมที่ให้ PS ชี้ไปยัง Y

```
*PS = 0 ;
```

ซึ่งจะให้ความหมายเหมือนกับ

```
Y = 0;
```

หรือประโยค

```
*PS += 1 ;
```

มีความหมายเหมือนกับ

```
Y = Y + 1 ;
```

## 5.2 พอยน์เตอร์กับการส่งค่าให้กับฟังก์ชัน

ในภาษา C การส่งผ่านค่าให้แก่ฟังก์ชันจะเป็นจะเป็นแบบ "การส่งผ่านแบบผ่านค่า" (passed by value) ( ซึ่งมีอีกลักษณะหนึ่ง คือ "การส่งผ่านแบบตัวแปร" (passed by reference)) ลักษณะ คือ ค่าที่ส่งให้แก่ฟังก์ชันจะถูกส่งให้แก่ตัวแปรคนละตัวกับตัวแปรที่ส่งค่าให้ ดังนั้นเมื่อมีการเปลี่ยนแปลงค่าตัวแปรในขั้นตอนการทำงานของฟังก์ชัน จะไม่มีผลต่อตัวแปรเดิมที่ส่งค่าให้ ลักษณะนี้เป็นลักษณะที่พึงระวังในการใช้งาน เนื่องจากจะทำให้โปรแกรมทำงานผิดจากที่ตั้งใจไว้ได้

ตัวอย่างเช่น โปรแกรมการสลับค่าตัวแปร X , Y

```
main( )
{       int X = 3 , Y = 5 ;
        swap( X , Y ) ;
}
```

```
void swap( a , b )
{           int  a , b , c  ;
             c  =  a  ;
             a  =  b  ;
}            b  =  c  ;
```

โปรแกรมนี้หวังว่าผลลัพธ์สุดท้าย จะได้ค่า X = 5 , Y = 3 แต่ผลจริงๆ คือ ค่า X , ค่า Y ยังมีค่าเหมือนเดิม ไม่เปลี่ยนแปลง เนื่องจากเหตุผลดังที่กล่าวไปข้างต้น ทางแก้ปัญหา คือ การใช้พอยน์เตอร์เข้ามาช่วย โดยแทนที่ค่า X และค่า Y จะถูกส่งให้แก่ฟังก์ชัน swap เราจะส่งค่าตำแหน่งของหน่วยความจำที่เก็บค่า X และค่า Y ไปแทน และฟังก์ชัน swap จะทำการเปลี่ยนแปลงค่าในตำแหน่งนั้นๆ ทำให้ค่า X และค่า Y มีการเปลี่ยนแปลงได้ ลักษณะนี้ถูกใช้อย่างมากในการใช้งานฟังก์ชันในภาษา C โปรแกรมสลับค่า X , Y แบบใหม่ที่ทำงานอย่างถูกต้อง สามารถเขียนได้ดังนี้

```
main( )
{           int  X = 3 , Y = 5  ;
            swap( &X , &Y )  ;
}
void   swap( PX , PY )  ;
{           int  *PX , *PY , C  ;
             C  =  *PX  ;
            *PX  =  *PY  ;
            *PY  =  C  ;
}
```

การส่งค่าโดยใช้พอยน์เตอร์นั้นมีผลดีที่ทำให้โปรแกรมประหยัดเนื้อที่ในการทำงาน และทำงานได้เร็วกว่า เนื่องจากไม่ต้องเสียเวลาในการคัดลอกข้อมูล โดยเฉพาะเมื่อข้อมูลที่ส่งให้มีขนาดใหญ่

## 5.3 พอยน์เตอร์กับตัวแปรอาร์เรย์

### ตัวแปรอาร์เรย์

เช่นเดียวกับภาษาอื่นๆ ภาษา C มีการใช้ตัวแปรชนิดอาร์เรย์ ซึ่งมีรูปแบบการใช้งานดังนี้

```
int  a[ 10 ]  ;
```

คือ การกำหนดให้ a เป็นตัวแปรอาร์เรย์ มีสมาชิก 10 ตัว คือ a[ 0 ] ถึง a[ 9 ] โดยเป็นแบบจำนวนเต็ม และเรายังสามารถใช้งานหลายมิติได้ ดังนี้

```
int  b[ 10 ][ 20 ]  ;
```

คือ การกำหนดให้ b เป็นตัวแปร 2 มิติ เก็บค่าที่เป็นจำนวนเต็มมีจำนวนสมาชิก 10*20 = 200 ตัว เมื่อเขียนว่า b[ 3 ][ 4 ] จะหมายถึง สมาชิกของ b ในแถวที่ 3 คอลัมน์ที่ 4 เราสามารถกำหนดค่าเริ่มต้นให้แก่ตัวแปรแบบอาร์เรย์ได้ เช่น

```
int  a[ 10 ] = { 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 }
```

คือ การกำหนดค่าให้ a[ 0 ] = 0 , a[ 1 ] = 1 , ... , a[ 9 ] = 9

```
int  c[ 2 ][ 3 ] = { { 1 , 2 , 3 } ,
                     { 4 , 5 , 6 } }
```

มีความหมายเหมือนกับ

```
int c[ 2 ][ 3 ]  ;
    c[ 0 ][ 0 ] = 1  ;
    c[ 0 ][ 1 ] = 2  ;
    c[ 0 ][ 2 ] = 3  ;
    c[ 1 ][ 0 ] = 4  ;
    c[ 1 ][ 1 ] = 5  ;
    c[ 1 ][ 2 ] = 6  ;
```

เราสามารถประกาศตัวแปรแบบอาร์เรย์นี้ได้โดยไม่ระบุจำนวนสมาชิก เช่น

```
int  a[ ]  ;
int  b[ ][ 3 ]  ;
```



**<u>พอยน์เตอร์กับตัวแปรอาร์เรย์</u>**

เมื่อเรากำหนดตัวแปรอาร์เรย์

```
int a[ 10 ] ;
```

ในความเป็นจริงก็คือ การกำหนดให้พอยน์เตอร์ a ชี้ไปยังสมาชิกของอาร์เรย์ นั่นคือ ชื่อของตัวแปรอาร์เรย์ แท้ที่จริงก็คือ ตัวแปรพอยน์เตอร์นั่นเอง โดยที่เมื่อทำการประกาศตัวแปรอาร์เรย์ พอยน์เตอร์จะชี้ไปยังสมาชิกตัวแรกของอาร์เรย์ ในที่นี้คือ a[ 0 ] นั่นคือพอยน์เตอร์ a จะชี้ไปยัง a[ 0 ] หรือ

```
a = &a[ 0 ]
```

และจะเห็นว่า *( a + 1 ) ก็หมายถึง a[ 1 ] นั่นเอง หรือ

```
*( a + 1 ) = a[ 1 ]
```

จะเห็นได้ว่าตัวแปรอาร์เรย์มีความเกี่ยวข้องกับพอยน์เตอร์อย่างใกล้ชิด ทำให้เราสามารถประยุกต์ใช้งานได้ต่างๆ ต่อไปจะยกตัวอย่างความสัมพันธ์กันระหว่างตัวแปรอาร์เรย์ และพอยน์เตอร์

```
char s[ ] ;          สมมูลกับ  char *s  ;
f( &a[ 2 ] ) ;       สมมูลกับ  f( a+2 ) ;    /* f เป็นฟังก์ชัน */
f( int arr[ ] ) { ... } ; สมมูลกับ  f( int *arr) { ... }
```

นอกจากนี้เรายังสามารถกำหนดอาร์เรย์ของพอยน์เตอร์ได้ ซึ่งก็คือ พอยน์เตอร์ที่ชี้ไปยังพอยน์เตอร์ หรืออาร์เรย์แบบหลายมิตินั่นเอง ซึ่งจะไม่ขอกล่าวถึงในที่นี้ ผู้ที่สนใจค้นคว้าเพิ่มเติม สามารถค้นคว้าได้จากหนังสือ [19,24,25,26] จะแสดงตัวอย่างการประกาศอาร์เรย์ของพอยน์เตอร์ซึ่งมีความใกล้เคียงกับอาร์เรย์หลายมิติมาก

```
int ( *a )[ 3 ] ;
int **a  ;
```

## 5.4 Pointer Arithmetic

ตัวแปรชนิดพอยน์เตอร์มีการดำเนินการทางคณิตศาสตร์ที่แตกต่างจากตัวแปรธรรมดาบางประการ ซึ่งจะได้รวบรวมไว้ในที่นี้

### 5.4.1 พอยน์เตอร์และเครื่องหมายกำหนดค่า

ถ้าให้ p เป็นพอยน์เตอร์ที่เป็นตัวชี้ของอาร์เรย์ เมื่อเขียนว่า

```
p++ ;
```

จะหมายถึง ให้ p ชี้ไปยังสมาชิกตัวถัดไปของอาร์เรย์ เช่น เดิม p ชี้ที่ p[ 0 ] จะกลายเป็นชี้ไปที่ p[ 1 ] และถ้าเขียนว่า

```
p += i ;
```

ก็หมายถึงให้ p ชี้ไปยังสมาชิกตัวที่ i ถัดจากที่ชี้อยู่ปัจจุบัน เช่น ถ้าเดิม p ชี้ที่ p[ 0 ] จะกลายเป็นชี้ไปที่ p[ i ]

การกำหนดค่าเริ่มต้นให้แก่พอยน์เตอร์ สามารถทำได้ในขณะที่ประกาศตัวแปร เช่น

```
char *p = a ;
char *pa = &b[ 0 ] ;
```

จะทำให้พอยน์เตอร์ p ชี้ไปยังตัวแปร a และพอยน์เตอร์ pa ชี้ไปยังสมาชิกตัวแรกของอาร์เรย์ b

### 5.4.2 การเปรียบเทียบพอยน์เตอร์

พอยน์เตอร์สามารถนำมาเปรียบเทียบกันได้ ภายใต้สภาพบางสภาพ

ถ้า p , q เป็นพอยน์เตอร์ที่ชี้ไปยังอาร์เรย์เดียวกัน เราสามารถใช้เครื่องหมาย == , != , < , > , = และอื่นๆ ในการเปรียบเทียบได้ เช่น

```
p<q
```

ซึ่งกรณีนี้จะเป็นจริงเมื่อ p ชี้ไปยังสมาชิกของอาร์เรย์ตัวที่อยู่ก่อนตัวที่ q ชี้อยู่ (นับจากน้อยไปหามาก) นอกจากนี้เรายังสามารถเปรียบเทียบพอยน์เตอร์กับค่าศูนย์ได้ ซึ่งมักใช้ในการตรวจสอบการส่งค่ากลับของฟังก์ชัน

ถ้า p , q ชี้ไปยังอาร์เรย์คนละอาร์เรย์ การเปรียบเทียบ p , q ดังกล่าวจะไม่มีผลที่คาดเดาได้ (unidefined)

### 5.4.3 การบวกพอยน์เตอร์กับจำนวนเต็ม

เช่นเดียวกับการใช้พอยน์เตอร์ กับการใช้เครื่องหมายกำหนดค่า เราสามารถบวกจำนวนเต็มเข้ากับพอยน์เตอร์ได้ เช่น p+n ผลลัพธ์ที่ได้ คือ พอยน์เตอร์ที่ชี้ไปยังตัวแปรตัวที่ n ถัดจากตัวแปรที่พอยน์เตอร์กำลังชี้อยู่ ในที่นี้พอยน์เตอร์อาจชี้ไปยังตัวแปรชนิดใดก็ได้ไม่จำเป็นต้องเป็นตัวแปรอาร์เรย์ ซึ่งชนิดของตัวแปรจะกำหนดระยะในการกระโดดของพอยน์เตอร์

### 5.4.4 การลบพอยน์เตอร์

ภายใต้เงื่อนไข p , q ชี้ไปยังอาร์เรย์เดียวกัน และ p<q

q-p+1

จะให้ค่าเป็นจำนวนของสมาชิก ที่ p ชี้อยู่ จนถึงสมาชิกที่ q ชี้อยู่(ให้สังเกตค่า 1 ที่บวกเข้า)

การดำเนินการทางเลขคณิตนอกเหนือจากที่กล่าวมา ไม่สามารถทำได้ เช่น การบวกพอยน์เตอร์ 2 ตัวเข้าด้วยกัน การคูณ หาร ชิฟท์ การปฏิบัติการบนบิต หรือการบวกจำนวนทศนิยมเข้ากับพอยน์เตอร์ หรือแม้แต่การกำหนดค่าพอยน์เตอร์ชนิดหนึ่งให้กับพอยน์เตอร์อีกชนิดหนึ่งโดยไม่ใช้ cast ยกเว้นกรณีใช้พอยน์เตอร์ที่ชี้ไปยัง void (Generic Pointer)

เช่น
```
int *px ;
float *py ;
px = py ;          /* ใช้ไม่ได้ */
```
จะต้องใช้ cast
```
px = (int *) py ;
```
แต่ถ้าเป็นกรณีพอยน์เตอร์ที่ชี้ไปยัง void จะสามารถเกิด automatic conversion ได้ เช่น
```
void *pv ;
px = pv ;
```
พอยน์เตอร์ที่ชี้ไปยังอาร์เรย์นั้น คือเป็นพอยน์เตอร์แบบค่าคงที่ ฉะนั้นเราจึงไม่สามารถใช้การกำหนดค่าให้แก่พอยน์เตอร์ชนิดนี้ได้ เช่น
```
int a[ 0 ], *b ;
    a = b ;          /* ใช้ไม่ได้ */
```
หลายครั้งเราต้องการคัดลอกอาร์เรย์ 2 ชุด เช่น a[ ] และ b[ ] เราไม่สามารถใช้การกำหนดค่าให้พอยน์เตอร์ได้ เช่น
```
int a[ 10 ], b[ 10 ] ;
```

```
a = b ;          /* ผิด */
```

การคัดลอกจะต้องทำโดยการคัดลอกสมาชิกในอาร์เรย์ทีละตัว โดยใช้โปรแกรมช่วย ซึ่งในภาษา C ก็ได้เตรียมฟังก์ชันประเภทนี้ไว้ให้แล้ว ตัวอย่างเช่น ฟังก์ชัน memcpy( ) ซึ่งอยู่ใน Library string.h

## 5.5 พอยน์เตอร์ที่ชี้ไปยังฟังก์ชัน [24]

ในภาษา C เราสามารถจะส่งฟังก์ชันไปยังฟังก์ชันได้ เหมือนกับการส่งค่าไปยังฟังก์ชัน และฟังก์ชันก็สามารถส่งค่ากลับที่เป็นฟังก์ชันได้ ( ฟังก์ชันเป็นเหมือนพารามิเตอร์ ) ซึ่งทำได้โดยการใช้พอยน์เตอร์ที่ชี้ไปยังฟังก์ชัน พอยน์เตอร์นี้เองถูกใช้ในการส่งผ่านค่า

การประกาศพอยน์เตอร์ที่ชี้ไปยังฟังก์ชัน ใช้รูปแบบ ดังนี้

```
type ( *funcptr ) ( parameter-list );
```

จะเป็นการประกาศให้ funcptr เป็นพอยน์เตอร์ชี้ไปยังฟังก์ชันที่มีพารามิเตอร์อยู่ใน ( parameter-list ) และส่งค่ากลับเป็นชนิด type จะต้องมีวงเล็บคร่อม *funcptr เสมอ มิฉะนั้นจะให้ความหมายที่ผิดออกไป

เช่น type *funcptr ( parameter-list );

จะกลายเป็น funcptr เป็นฟังก์ชันที่ส่งค่ากลับเป็นพอยน์เตอร์ที่ชี้ไปยังชนิด type

การกำหนดให้พอยน์เตอร์ชี้ไปยังฟังก์ชันที่ต้องการ ทำได้โดยการกำหนดชื่อฟังก์ชันให้แก่พอยน์เตอร์โดยไม่ต้องใช้ตัวดำเนินการ & และไม่มี parameter-list เช่น

```
funcptr = pow ;
funcptr = &pow ;      /* ผิด */
funcptr = pow( x , y ) ;    /* ผิด */
```

จะทำให้ funcptr ชี้ไปยังฟังก์ชัน pow โดย pow เป็นชื่อฟังก์ชัน

การเรียกใช้ฟังก์ชันที่พอยน์เตอร์ชี้อยู่ ทำได้ในลักษณะเดียวกับพอยน์เตอร์ชนิดอื่นๆ เช่น

```
( *funcptr )( 2 , 3 ) ;
```

สมมูลกับ `pow( 2 , 3 ) ;`

## 5.6 ชนิดของตัวแปรพอยน์เตอร์

ตัวแปรพอยน์เตอร์ถ้าแบ่งตามระยะความไกลในการชี้ไปยังตัวแปร จะสามารถแบ่งได้เป็น 3 ชนิด[19] คือ

1. Near Pointer

เป็นพอยน์เตอร์ขนาด 16 บิต ใช้เป็นออฟเซตในการชี้บนรีจิสเตอร์เซกเมนต์ตัวใดตัวหนึ่ง เช่น รีจิสเตอร์ cs , ds , es , ss ซึ่งมีขอบเขตการชี้อยู่ภายในหน่วยความจำ 64 K การประกาศใช้ทำได้โดยใช้คำสั่ง near ดังตัวอย่าง

```
char near *s  ;
int near *px  ;
```

2. Far Pointer

เป็นพอยน์เตอร์ขนาด 32 บิต เก็บทั้งค่าออฟเซต และค่าเซกเมนต์ ทำให้สามารถชี้หน่วยความจำในขอบเขตเกินกว่า 64 K มีข้อควรระวังในการใช้งานคือ เมื่อดำเนินการทางคณิตศาสตร์ หรือ ลอจิกกับพอยน์เตอร์ชนิดนี้จะมีผลกับเฉพาะในส่วนของออฟเซต ไม่มีผลต่อส่วนของเซกเมนต์ เช่น ถ้าเรามีพอยน์เตอร์ที่ชี้ไปยังตำแหน่งค่าเดียวกัน 3 ตัว ดังนี้

a = 0000 : 0120 , b = 0010 : 0020 , c = 0011 : 0010

ทั้ง 3 ตัวจะหมายถึง แอดเดรสตำแหน่งเดียวกัน แต่เมื่อเราเขียนประโยคข้างล่าง จะให้ค่าความจริงเป็นเท็จทั้งหมด

```
if( a==b ) ...
if( b==c ) ...
if( a==c ) ...
```

หรือถ้าทำการเปรียบเทียบพอยน์เตอร์โดยใช้เครื่องหมาย < , > , <= , >= จะทำการเปรียบเทียบโดยนำค่าออฟเซตมาคิดเท่านั้น หรือถ้าทำการบวกค่าจำนวนเต็ม เช่น บวก 1 เข้ากับ 5031:FFFF ค่าที่ได้จะเป็น 5031:0001 ไม่ใช่ 6031:0000 เป็นต้น

การประกาศใช้พอยน์เตอร์ชนิดนี้ทำโดยใช้คำสั่ง far ดังตัวอย่าง

```
char far *fp  ;
```

3. Huge Pointer

มีขนาด 32 บิต มีลักษณะเช่นเดียวกับ Far Point แต่ต่างกันตรงที่พอยน์เตอร์ชนิดนี้จะผ่านการ normalized แล้ว จึงทำให้แอดเดรสหนึ่งๆ จะมีค่าของพอยน์เตอร์ที่ชี้มาได้เพียง 1 ค่าทำให้แก้ปัญหาเกี่ยวกับการดำเนินการทางคณิตศาสตร์ดังเช่นของ Far Pointer การประกาศใช้ทำโดยใช้คำสั่ง Huge เช่น

```
int huge *l;
```

# บทที่ 6
# ชนิดของตัวแปรที่สร้างขึ้น

## 6.1 Structure

ชนิดของตัวแปรที่ผ่านมา เราสามารถแยกได้เป็น 2 พวก พวกแรกคือ ตัวแปรชนิดที่เก็บได้ทีละ 1 ค่า และพวกที่สอง คือ ชนิดที่เก็บได้เป็นกลุ่ม อันได้แก่ อาร์เรย์ แต่แม้อาร์เรย์จะเก็บค่าได้หลายๆ ค่า ในแต่ละค่าก็จะต้องเป็นตัวแปรชนิดเดียวกัน เช่น integer , float เป็นต้น ในกรณีที่เราต้องการรวมกลุ่มข้อมูลต่างชนิดกันเข้าด้วยกัน โดยใช้ชื่อตัวแปรตัวเดียวกัน ทำได้โดยใช้โครงสร้างข้อมูลแบบ structure โครงสร้างแบบ structure นี้ก็คือ record ในภาษาปาสคาลนั่นเอง

รูปแบบการใช้งาน

```
structure record
{
        int i ;
        char a[20] ;
} ;
```

เป็นส่วนของการนิยามชนิดของโครงสร้าง structure โดยในที่นี้เป็นชนิดโครงสร้างที่มีชื่อว่า record ประกอบด้วยตัวแปร 2 ตัวแปร คือ i ซึ่งเป็นชนิดจำนวนเต็ม และ a เป็นชนิดอาร์เรย์ของตัวอักษรโดยมีขนาดความยาว 20 ตัวอักษร การประกาศตัวแปรชนิด struture ทำได้โดย

```
struct ชื่อชนิด ตัวแปรแบบ structure ;
```

เช่น `struct record p ;`

หรือเราอาจจะประกาศตัวแปรโดยใช้ชื่อตัวแปรต่อท้ายการนิยามชนิดได้ทันที เช่น

```
struct record
{       int i ;
        char a[20] ;
}       p ;
```

การอ้างถึงค่าตัวแปรใน structure กระทำได้โดยการใส่ชื่อตัวแปรตามด้วยเครื่องหมาย . และชื่อของตัวแปรภายใน structure เช่น

```
p.i = 3 ;
p.a[1] = '$' ;
```

การกำหนดค่าเริ่มต้นของตัวแปร struct สามารถกระทำได้ในเวลาที่ประกาศตัวแปร โดยมีข้อบังคับ คือ ชนิดของตัวแปรที่จะทำการกำหนดค่าเริ่มต้นต้องเป็นแบบ static หรือ external เท่านั้น เช่น

```
static struct record p =
{ 3 , "Chulalongkorn" } ;
```

เราสามารถสร้างตัวแปร structure ซ้อนอยู่ในตัวแปร structure ได้ เช่น

```
struct date
{       int month ;
        int day ;
        int year ;
} ;
struct record
{       int i ;
        char a[20] ;
        struct date birthday ;
} p ;
```

และในการอ้างอิงตัวแปรจะทำได้ในลักษณะเดิม เช่น

```
p.birthday.month = 3 ;
p.birthday.year = 1980 ;
```



## การส่ง structure ให้แก่ฟังก์ชัน

เราสามารถส่ง structure ให้แก่ฟังก์ชันได้โดยตรง ในลักษณะเช่นเดียวกับตัวแปรชนิดอื่นๆ โดยการส่งด้วยชื่อของตัวแปร structure ซึ่งจะเป็นการส่งแบบผ่านค่า (passed by value) ดังเช่นตัวแปรชนิดอื่นๆ

ตัวอย่าง ให้ p เป็นตัวแปร struct ที่ได้ประกาศตัวแปรไว้แล้ว

```
Func(p)  ;
```

อย่างไรก็ตามเราสามารถส่งพอยน์เตอร์ของ struct ไปยังฟังก์ชันได้เช่นกัน

```
Func(&p)  ;
```

## 6.2 ยูเนียน

ยูเนียน คือ ชนิดของตัวแปรที่สามารถเก็บค่าได้หลาย ๆ ชนิด(คนละเวลากัน) เช่น สามารถเก็บจำนวนเต็ม จำนวนทศนิยม หรือ ตัวอักษร ถ้า mix เป็นชนิดของตัวแปรดังกล่าว สามารถระบุได้โดย

```
union   mix
{
        int i  ;
        float f  ;
        char c  ;
}  ;
```

และการประกาศตัวแปร x ให้เป็นชนิด mix ทำได้โดย ระบุ

```
union   mix     x  ;
```

หรืออาจวาง x ไว้ตอนท้ายของการนิยาม mix ได้เช่นเดียวกับ struct

การกำหนดค่าหรือการอ้างอิงตัวแปรชนิด union นี้ ทำโดยระบุชื่อตัวแปร ตามด้วย . และตามด้วยชื่อตัวแปรในฟิลด์ที่เป็นชนิดที่ต้องการกำหนดค่า หรืออ้างอิง เช่น

```
x.i  =  100  ;
x.f  =  32.5  ;
```

ตัวแปรยูเนียนสามารถเก็บค่าได้ทีละค่าเท่านั้น เช่น ตัวอย่างการกำหนดค่า x ข้างบน ค่าของ x จะเป็นค่าที่ถูกกำหนดไว้หลังสุด คือ 32.5 และเป็นชนิด float

## 6.3 บิตฟิลด์

ในตัวแปรชนิด struct ในฟิลด์ที่เป็นจำนวนเต็ม บางครั้งเราอาจใช้จำนวนเต็มที่มีขนาดไม่ใหญ่ เช่น มีขนาดเล็กกว่า 16 บิต แต่เนื่องจากการกำหนดตัวแปรชนิดจำนวนเต็มจะให้ขนาดอย่างน้อย 16 บิต จึงทำให้สูญเสียเนื้อที่ในการเก็บไป ภาษา C อนุญาตให้เราสามารถแก้ไขเพื่อประหยัดเนื้อที่ได้ โดยสามารถกำหนดจำนวนบิตที่ต้องการใช้ในฟิลด์จำนวนเต็ม หลังจากนั้นเครื่องจะทำการรวบรวมฟิลด์ต่างๆ ให้เป็นคำ(word) เพื่อประหยัดเนื้อที่ในการเก็บ ฟิลด์ที่เรากำหนดจำนวนบิตที่ต้องการใช้นี้ เราเรียกว่า บิตฟิลด์ การกำหนดบิตฟิลด์สามารถทำได้ในลักษณะเดียวกับตัวแปร struct เพียงแต่เพิ่มเครื่องหมาย : (colon) หลังชื่อฟิลด์ และตามด้วยจำนวนบิตที่ต้องการ เช่น

```
struct data
{
        unsigned int year  : 7 ;
        unsigned int day   : 5 ;
        unsigned int month : 4 ;
} x ;
```

การอ้างอิงถึงฟิลด์ที่ต้องการ ก็ทำในลักษณะเดียวกับตัวแปร struct ดังตัวอย่าง

```
x.month = 10 ;
y = x.year ;
```

มีข้อควรระวังในการใช้งาน คือ คอมไพเลอร์ส่วนมากจะไม่ยอมให้เรากำหนดบิตฟิลด์ที่มีขนาดใหญ่กว่าขนาดที่ใช้เก็บคำ(word) และ เราไม่สามารถสร้างพอยน์เตอร์ให้ชี้ไปยังบิตฟิลด์ได้

## 6.4 Enumerated Types

ในบางครั้งตัวแปรที่เราใช้งานอาจจะทำการเก็บค่าเพียงไม่กี่ค่าที่เป็นไปได้ เช่น เก็บค่าเพียง 3 ค่าที่เป็นไปได้ คือ integer , real , string ในกรณีนี้เราสามารถใช้ตัวแปร enumerated type ดังนี้

```
enum itype { integer , real , string } ;          /* กำหนดชนิด */
```

ในลักษณะการทำงานภายในเครื่อง จะกำหนดให้ integer มีค่า 0 real มีค่า 1 string มีค่า 2 จริงๆ ลักษณะการกำหนดเช่นนี้สามารถใช้ # defined หรือ const แทนได้ แต่การใช้ enum จะสะดวกกว่า การประกาศตัวแปรทำได้ดังตัวอย่าง

```
enum itype x ;
```

ประโยชน์ของการใช้ enum คือ เราสามารถใช้ในการกำหนดค่าคงที่ที่เป็นจำนวนเต็มที่มีจำนวนมากได้ เช่น ตัวอย่างข้างบนคือ การกำหนดให้ integer มีค่า 0 real มีค่า 1 string มีค่า 2 หรือ

```
enum day {Monday, Tuesday, Wednesday, Thursday, Friday, Saturday,
          Sunday};
```

เป็นการกำหนดให้ Monday = 0 , Tuesday = 1 , ... , Sunday = 6

# บทที่ 7
# ฟังก์ชันที่ใช้ในการทดลอง

ในบทที่ 1-6 ได้กล่าวถึงภาษา C ส่วนที่เป็นตัวโครงสร้างของภาษา (C Element) ซึ่งมีลักษณะเป็นมาตรฐานที่สามารถใช้ได้กับ C Compiler ทุกตัว อย่างไรก็ตามลักษณะของภาษาซีที่เด่นมากประการหนึ่ง คือ การมีฟังก์ชันที่จัดเตรียมไว้ให้มากมายที่สามารถเรียกใช้งานได้ ทำให้สามารถเขียนโปรแกรมได้ง่ายขึ้น และได้โปรแกรมที่มีขนาดเล็กลงมาก อย่างไรก็ตามในส่วนของฟังก์ชันนี้อาจมีลักษณะที่แตกต่างกันไปสำหรับ C Compiler แต่ละตัว ในที่นี้จะกล่าวถึงฟังก์ชันของคอมไพเลอร์ TURBO C version 2.0 ของบริษัท Borland โดยจะรวบรวมฟังก์ชันต่าง ๆ ที่ถูกใช้ในการทดลองและยกตัวอย่างการใช้ฟังก์ชันบางฟังก์ชันที่ผู้ใช้อาจเกิดปัญหา เพื่อให้ผู้ใช้สามารถใช้เข้าใจการใช้งานฟังก์ชันได้อย่างรวดเร็ว จะแบ่งเป็นหัวข้อ ดังนี้

1. ฟังก์ชันเกี่ยวกับการรับและการแสดงผลข้อมูล

   จะกล่าวถึงการรับข้อมูลจากคีย์บอร์ด และการแสดงผลบนจอภาพ

2. ฟังก์ชันเกี่ยวกับการเขียน/อ่านไฟล์

3. ฟังก์ชันเกี่ยวกับการติดต่อกับอุปกรณ์ฮาร์ดแวร์และหน่วยความจำ

   จะกล่าวถึงวิธีการเขียน/อ่านข้อมูลกับพอร์ต I/O และหน่วยความจำ

4. ฟังก์ชันเกี่ยวกับการเรียกใช้บริการจากไบออสหรือดอส

   จะกล่าวถึงการเรียกใช้ซอฟท์แวร์อินเตอร์รัปต์ของไบออสและดอส ซึ่งมีฟังก์ชันที่ เตรียมไว้ให้มากมาย โดยจะกล่าวถึงเฉพาะบางตัวที่ถูกใช้งานบ่อยๆ และจะได้กล่าวถึงการเข้าถึงรีจิสเตอร์ต่างๆ ของ CPU ในส่วนนี้จะทำให้เราสามารถทำงานได้อย่างใกล้ชิดกับเครื่อง ซึ่งเป็นจุดเด่นของภาษาระดับต่ำ ที่ภาษา C ได้รวบรวมมาเป็นจุดเด่นของตัวเองด้วย

5. ฟังก์ชันอื่นๆ

   จะกล่าวถึงฟังก์ชันอื่นๆ บางตัวที่ไม่ได้อยู่ในหัวข้อ 1-4 เช่น ฟังก์ชันเกี่ยวกับการจัดการหน่วยความจำ ฟังก์ชันทางคณิตศาสตร์ เป็นต้น

TURBO C ยังมีฟังก์ชันต่างๆ อีกมากที่ไม่ได้กล่าวถึงในที่นี้ ผู้ใช้สามารถศึกษาเพิ่มเติมได้จากหนังสือต่างๆ เช่น TURBO C Reference Guide ของบริษัท Borland [20] และหนังสือการเรียนรู้ภาษา C อื่นๆ ซึ่งมักจะกล่าวถึงฟังก์ชันของภาษา C ในลักษณะการประยุกต์ใช้งาน

ตามลักษณะเป้าหมายของหนังสือนั้น ๆ เป็นต้น การจะหาหนังสือภาษา C ที่กล่าวถึงการใช้ฟังก์ชันอย่างค่อนข้างสมบูรณ์นั้นทำได้ยาก หนังสือดังกล่าวมักจะมาจากบริษัทผู้ผลิตคอมไพเลอร์นั้นๆอย่างไรก็ตามผู้ใช้สามารถดูวิธีการใช้งานของฟังก์ชันได้จาก HELP ของ TURBO C เอง สำหรับรูปแบบการอธิบายการใช้งานฟังก์ชันที่จะกล่าวต่อไป มีรูปแบบเช่น

```
void putpixel(int x , int y , int pixelcolor);
```

ความหมาย คือ ฟังก์ชัน putpixel รับค่าพารามิเตอร์ 3 ตัว คือ x , y , pixelcolor ซึ่งเป็นชนิดจำนวนเต็ม ฟังก์ชันไม่มีการส่งค่ากลับ(ชนิด void) ตัวอย่างการเรียกใช้ฟังก์ชันดังกล่าว

```
putpixel(100 , 100 , 7);
```

1. ฟังก์ชันการรับและแสดงผลข้อมูล

ภาษา C เองไม่มีคำสั่งที่ใช้ในการรับและแสดงผลข้อมูลโดยตรงเหมือนกับภาษาอื่น ๆ การรับและแสดงผลข้อมูลจะต้องทำโดยการเรียกใช้ฟังก์ชันที่เกี่ยวข้อง ตัวอย่างเช่น

| | | | |
|---|---|---|---|
| printf | getc | gotoxy | putpixel |
| scanf | getch | clrscr | line |
| putc | getchar | detectgraph | cleardevice |
| putch | getche | initgraph | clearviewport |
| putchar | kbhit | closegraph | |

ตัวอย่างการเรียกใช้โหมดกราฟฟิกของระบบ

```
int gdriver,gmode;
detectgraph(&gdriver,&gmode);
initgraph(&gdriver,&gmode, "C:\\TC");
```

ในที่นี้ graphic driver อยู่ในไดเรกเตอรี C:\TC

2. ฟังก์ชันการเขียน/อ่านไฟล์

ประกอบด้วย

| | | |
|---|---|---|
| fopen | ftell | fwrite |
| fclose | fseek | feof |
| rewind | fread | |

การใช้งานฟังก์ชันเหล่านี้ จำเป็นจะต้องรู้จักกับ ไฟล์พอยน์เตอร์(file pointer) ซึ่งเป็นพอยน์เตอร์ที่ชี้ไปยังข้อมูลที่เป็นชนิดไฟล์ ซึ่งจะพบว่าภาษา C มาตรฐานไม่มีข้อมูลชนิดดังกล่าว ข้อมูลชนิดไฟล์นี้ คือ ข้อมูลที่ถูกนิยามไว้ใน Library stdio.h โดยมีชื่อว่า FILE ในที่นี้จะแสดงตัวอย่างการใช้งานฟังก์ชันเหล่านี้บางฟังก์ชัน

```
#include <stdio.h>
/* return the number of bytes in file stream */
long filesize(FILE *stream)
{
        long curpos,length;
        curpos = ftell(stream);
        fseek(stream, 0L, SEEK_END);
        length = ftell(stream);
        fseek(stream, curpos, SEEK_SET);
        return(length);
}
main( )
{
        FILE *stream;
        stream = fopen("MYFILE.TXT","r");
        printf("filesize of MYFILE.TXT is %ld bytes\n", filesize(stream));
        fclose(stream);
}
```

3. ฟังก์ชันการติดต่อกับอุปกรณ์ฮาร์ดแวร์และหน่วยความจำ ประกอบด้วย

| | | | |
|---|---|---|---|
| inp | outp | poke | peekb |
| inport | outport | pokeb | |
| inportb | outportb | peek | |

นอกจากนี้ยังมีฟังก์ชันที่เขียนขึ้น ซึ่งใช้กับชุดฝึกทดลองโดยเฉพาะ 2 ฟังก์ชัน คือ

in - คือ ฟังก์ชันการอ่านข้อมูลจากระบบบัสของชุดทดลอง

การใช้งาน
```
#include<edl.h>
char in(address);
```
รายละเอียด อ่านข้อมูลขนาด 1 ไบต์(byte) จากพอร์ตของชุดฝึกทดลองที่ระบุด้วย address

out - คือ ฟังก์ชันการเขียนข้อมูลไปยังระบบบัสของชุดทดลอง

การใช้งาน
```
#include<edl.h>
void out(char address,char byte);
```
รายละเอียด เขียนข้อมูลขนาด 1 ไบต์(byte)ไปยังพอร์ตของชุดฝึกทดลองที่ระบุด้วย address

4. ฟังก์ชันการเรียกใช้บริการจากไบออส หรือดอส

| | |
|---|---|
| geninterrupt | intdos |
| intr | int86x |
| int86 | intdosx |

ในการเรียกฟังก์ชันเหล่านี้ จะต้องทำการส่งค่าให้แก่รีจิสเตอร์ของ CPU วิธีการส่งค่าให้รีจิสเตอร์ของ CPU สามารถทำได้ดังนี้

1. ส่งค่าโดย Pseudo Register[19]

ตัวอย่างการส่งค่าให้แก่รีจิสเตอร์ AX , CX

```
_AX = 0x0a42;
_CX = 0x10;
geninterrupt(0x10);
```

ชื่อต่าง ๆ ก็คือ ชื่อของรีจิสเตอร์ของ CPU นั่นเอง ซึ่งจะต้องเขียนโดยใช้เครื่องหมาย underscore( _ ) และตามด้วยชื่อรีจิสเตอร์ โดยจะต้องเป็นอักษรตัวใหญ่(capital letter)

2. ส่งค่าโดยตัวแปร

โดยการกำหนดค่าที่ต้องการส่งให้รีจิสเตอร์ให้แก่ตัวแปร ซึ่งได้นิยามไว้ใน header file "dos.h" ฟังก์ชันจะทำการคัดลอกค่าต่าง ๆ ในตัวแปรลงในรีจิสเตอร์ของ CPU เพื่อจะปฏิบัติการต่อไป ตัวแปรเหล่านี้ประกอบด้วย

ตัวแปร REGS

เป็นตัวแปรชนิด union มีรายละเอียดที่ได้ถูกนิยามไว้ดังแสดง[27]

```
struct WORDREGS {
        unsigned int as,bx,cx,dx,si,di,cflag,flags;
        };
struct BYTEREGS {
        unsigned char al,ah,bl,bh,cl,ch,dl,dh;
        };
union   REGS    {
        struct WORDREGS x;
        struct BYTEREGS   h;
        };
```

ตัวอย่างการใช้งาน

```
        #include<dos.h>
        union  REGS   r;
        r.h.ah  =  2;
        r.h.al  =  0x41;
        int86(0x10,&r,&r);              /*  เรียกใช้ฟังก์ชัน int86 */
```

ตัวแปร SREGS

เป็นตัวแปรชนิด struct ที่ทำหน้าที่ส่งค่าให้แก่เซกเมนต์รีจิสเตอร์ ซึ่งใช้ในกรณีที่โปรแกรมเมอร์ต้องการใช้ far pointer หรือใช้ large data memory model เพื่อใช้กำหนดเซกเมนต์ที่ต้องการ ตัวแปร SREGS มีรายละเอียดดังแสดง

```
struct   SREGS {
         unsigned int es;
         unsigned int cs;
         unsigned int ss;
         unsigned int ds;
};
```

ตัวอย่างการใช้งาน

```
#include<dos.h>
struct SREGS S ; union REGS r;
s.ds = 0xF000;
r.h.ah = 0x02;
r.h.al = 0x61;
intdosx(&r,&r,&s);
```

ตัวแปร REGPACK

มีรูปแบบดังนี้

```
struct   REGPACK {
         unsigned r_ax, r_bx, r_cx, r_dx;
         unsigned r_bp, r_si, r_di, r_ds, r_es, r_flags;
};
```

สามารถใช้ในการส่งค่าให้แก่รีจิสเตอร์ของ CPU ได้ ตัวอย่างฟังก์ชันที่ใช้ตัวแปรชนิด REGPACK คือ ฟังก์ชัน intr เช่น

```
#include<dos.h>
struct REGPACK rp;
r_ax = 0x0a41;
r_cx = 10;
intr(0x20,&rp);
```

การส่งค่ากลับจากฟังก์ชัน(return value) จะทำในลักษณะเดียวกันกับการส่งค่าไป เช่น

```
int86(0x10,&r,&r1);
```

คือ การส่งค่ากลับให้แก่ r1 ซึ่ง r1 อาจใช้ตัวแปรซ้ำกับการส่งค่าไปก็ได้ เช่น

```
int86(0x10,&r,&r);
```

5. ฟังก์ชันอื่นๆ

<u>ฟังก์ชันการควบคุม CPU</u>

enable disable

<u>ฟังก์ชันการจัดการหน่วยความจำ</u>

malloc realloc

calloc free

นอกจากนี้ยังมีคำว่า sizeof ซึ่งไม่ใช่ฟังก์ชัน แต่เป็น operator ซึ่งมักจะใช้ควบคู่กับฟังก์ชันการจัดการหน่วยความจำเหล่านี้ ใช้เป็นตัวช่วยคำนวณหาขนาดของข้อมูล ให้ผลลัพธ์เป็นจำนวนไบต์ที่ใช้เก็บข้อมูลนั้นๆ มีรูปแบบ คือ

```
sizeof(arg)
```

โดย arg จะเป็นชื่อของตัวแปร หรือชื่ออาร์เรย์ หรือชนิดของข้อมูลก็ได้ เช่น

sizeof(x) ใช้ในการหาขนาดเนื้อที่หน่วยความจำที่ใช้เก็บตัวแปร x

sizeof(int) ใช้ในการหาขนาดเนื้อที่หน่วยความจำที่ใช้เก็บตัวแปรชนิดจำนวนเต็ม

sizeof(struct data) ใช้ในการหาขนาดเนื้อที่หน่วยความจำที่ใช้เก็บตัวแปรชนิด data ที่เป็นแบบ struct ซึ่งได้นิยามไว้แล้ว

<u>ฟังก์ชันทางคณิตศาสตร์</u>

| | | |
|---|---|---|
| sin | abs | fmod |
| cos | pow | exp |
| tan | sqrt | log |

<u>ฟังก์ชันอื่นๆ เช่น</u>

getvect delay setvect

# ภาคผนวก ค

# คู่มือการเขียนโปรแกรมภาษา C สำหรับงานควบคุม

# คู่มือการเขียนโปรแกรมภาษา C สำหรับงานควบคุม

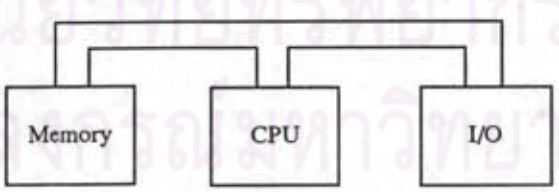
## 1. บทนำ

คู่มือการเขียนโปรแกรมภาษา C สำหรับงานควบคุมนี้ กล่าวถึง การใช้ภาษา C ในการควบคุมฮาร์ดแวร์ต่าง ๆ เช่น การเขียนอ่านข้อมูลกับ I/O การโปรแกรมรีจิสเตอร์บนชิปต่าง ๆ วิธีการเข้าถึงหน่วยความจำโดยตรง การจัดการการอินเตอร์รัปต์ เป็นต้น โดยจะใช้เครื่องคอมพิวเตอร์ IBM PC ในการควบคุม และใช้โปรแกรม TURBO C ของบริษัท Borland ซึ่งทำงานบนระบบปฏิบัติการ DOS ตั้งแต่เวอร์ชั่น 2.0 เป็นต้นไปในการเขียนโปรแกรมภาษา C อย่างไรก็ตาม ผู้อ่านสามารถนำเนื้อหาของคู่มือนี้ไปประยุกต์ใช้กับระบบอื่น ๆ ได้ ผู้อ่านที่ต้องการข้อมูลเพิ่มเติมในการใช้งานโปรแกรม TURBO C สามารถหาข้อมูลได้จากหนังสือ ต่อไปนี้

- TURBO C Reference Guide, Borland International Inc., USA, 1987[20]
- TURBO C User's Guide, Borland International Inc., USA, 1987[19]

## 2. การเขียนอ่านข้อมูลกับ I/O

I/O คือ อุปกรณ์รอบข้างที่ต่อเข้ากับระบบคอมพิวเตอร์ เพื่อทำหน้าที่ต่าง ๆ เช่น คีย์บอร์ด ทำหน้าที่รับข้อมูลจากผู้ใช้ จอภาพทำหน้าที่ในการแสดงผล ฟลอปปี้ดิสค์ ทำหน้าที่ในการบันทึกและอ่านข้อมูลบนจานแม่เหล็ก เป็นต้น ระบบคอมพิวเตอร์พื้นฐานแสดงดังในรูปที่ ค.1 [14]




รูปที่ ค.1 ระบบคอมพิวเตอร์พื้นฐาน

จากรูปที่ ค.1 จะเห็นว่า อุปกรณ์ต่าง ๆ นอกเหนือจาก CPU และ Memory จัดเป็น I/O ทั้งสิ้น สำหรับเครื่องคอมพิวเตอร์ IBM PC I/O อาจจะเป็นอุปกรณ์ที่อยู่บนเมนบอร์ด หรืออุปกรณ์ที่ต่อภายนอกก็ได้ ตัวอย่าง I/O ที่อยู่บนเมนบอร์ด เช่น ชิป 8259, 8253, 8255 ตัวอย่าง I/O ที่ต่อภายนอก ได้แก่ คีย์บอร์ด จอภาพ เครื่องพิมพ์ เป็นต้น I/O เป็นช่องทางที่จะต่ออุปกรณ์

ภายนอกเข้ากับเครื่องคอมพิวเตอร์เพื่อทำการควบคุม ในที่นี้จะกล่าวถึงวิธีการเขียนอ่านข้อมูลกับ I/O โดยแบ่งเป็นหัวข้อต่าง ๆ ดังนี้

- การเชื่อมต่อ I/O เข้ากับเครื่อง IBM PC
- การใช้ In-Line Assembly
- การใช้ฟังก์ชันของภาษา C
- การเขียนอ่านข้อมูลกับ I/O ชนิด Memory mapped I/O
- การเขียนอ่านข้อมูลกับ I/O ของชุดทดลอง

## 2.1 การเชื่อมต่อ I/O เข้ากับเครื่อง IBM PC

เราสามารถเชื่อมต่อ I/O เข้ากับ IBM PC โดยใช้สัญญาณต่าง ๆ ดังนี้

A0-A9 (address bus)

เป็นสัญญาณการอ้างแอดเดรสของ I/O มีจำนวน 10 สาย จึงสามารถอ้างแอดเดรสของ I/O ได้จำนวน 1K (1024 I/O) ค่าแอดเดรสที่ใช้งานจะต้องอยู่ในช่วง 0000h-03FFh ซึ่งแอดเดรสดังกล่าวบางส่วนได้ถูกใช้งานไปแล้ว ดังนั้น ในการเลือกแอดเดรสของ I/O ที่เราสร้างขึ้น จะต้องหลีกเลี่ยงแอดเดรสดังกล่าว

D0-D7 (data bus)

เป็นบัสข้อมูลของระบบ ในบัสไซเคิลของการเขียนข้อมูลกับ I/O ข้อมูลจะถูกส่งออกมาบนบัสก่อนที่สัญญาณ $\overline{IOW}$ จะเปลี่ยนจากลอจิก 0 เป็น 1 ซึ่งโดยทั่วไปขอบขาขึ้นของสัญญาณ $\overline{IOW}$ จะถูกใช้เพื่อสั่งให้ I/O ที่มีแอดเดรสตรงกับค่าแอดเดรสบนบัสแอดเดรสรับข้อมูลเข้าไป ในบัสไซเคิลของการอ่านข้อมูลกับ I/O I/O จะต้องส่งข้อมูลออกมาบนบัสข้อมูล ก่อนที่สัญญาณ $\overline{IOR}$ จะเปลี่ยนจากลอจิก 0 เป็นลอจิก 1 ประมาณ 30 ns

$\overline{IOR}$ (I/O read)

เป็นสัญญาณที่สั่งให้ I/O ที่ถูกอ้างอิงถึง ส่งข้อมูลออกมาบนบัสข้อมูล

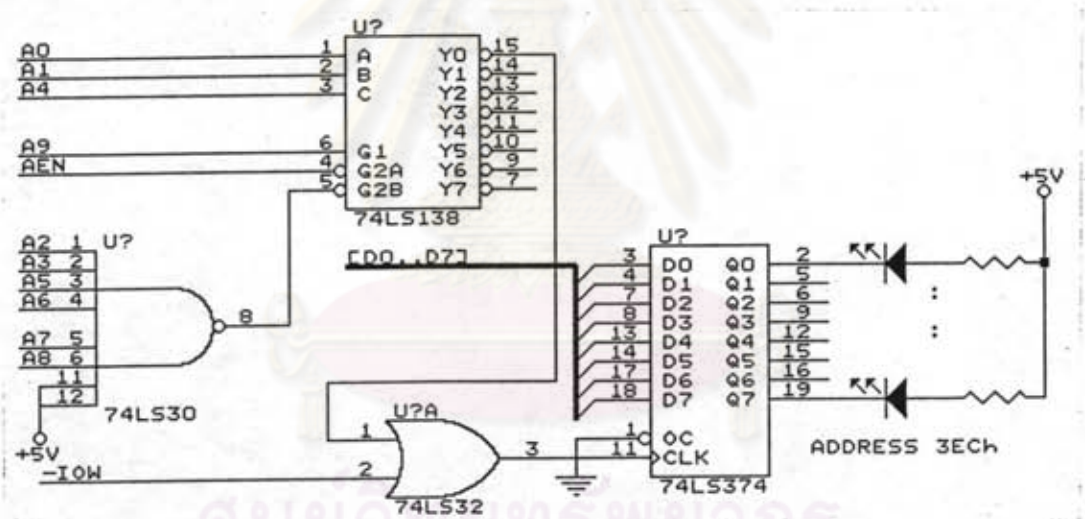
$\overline{IOW}$ (I/O write)

เป็นสัญญาณที่สั่งให้ I/O ที่ถูกอ้างอิงถึง รับข้อมูลที่ CPU ส่งออกมาบนบัสข้อมูลไปเก็บไว้

AEN (address enable)

เป็นสัญญาณออกที่ใช้แสดงว่าบัสไซเคิลที่เกิดขึ้นในช่วงที่สัญญาณ AEN แอคทีฟ (ลอจิก 1) เป็นบัสไซเคิลของขบวนการ DMA ดังนั้น ในการใช้งาน I/O อื่น ๆ จะต้องใช้สัญญาณ AEN นี้ในการเอนาเบิล หรือดิสเอเบิล I/O ด้วย

สัญญาณดังกล่าวถูกต่อออกมาที่สล็อตของ IBM PC รูปที่ ค.2 แสดงตัวอย่างการเชื่อมต่อ I/O เข้ากับ IBM PC ซึ่งเป็นพอร์ตสัญญาณออกที่ใช้ขับ LED




รูปที่ ค.2 ตัวอย่างการเชื่อมต่อ I/O เข้ากับ IBM PC

## 2.2 การใช้ฟังก์ชันของภาษา C

ภาษา C ไม่มีคำสั่งการเขียนอ่าน I/O โดยตรง จะต้องอาศัยฟังก์ชันที่สร้างขึ้นเฉพาะสำหรับเครื่องคอมพิวเตอร์แต่ละแบบ จึงมีลักษณะต่าง ๆ กัน ขึ้นกับคอมไพเลอร์แต่ละตัว สำหรับ TURBO C ฟังก์ชันดังกล่าว บรรจุอยู่ในไลบรารีของ TURBO C ซึ่งได้ประกาศฟังก์ชันไว้ใน header file (นามสกุล .h บางครั้งเรียก include file) โดยอยู่ใน header file ที่ชื่อ "dos.h" ในการเรียกใช้ฟังก์ชันของ TURBO C จะต้องกำหนด include file ในส่วนของ C

preprocessor โดยระบุชื่อของ header file ที่ได้ทำการประกาศฟังก์ชันที่ต้องการใช้ไว้ ดังนั้น ในการใช้งานฟังก์ชันการเขียนอ่าน I/O ในส่วนต้นโปรแกรม จะต้องมีประโยค

```
#include <dos.h>
```

ฟังก์ชันดังกล่าว ประกอบด้วย

- inportb
- outportb
- inport
- outport
- inp
- outp

รายละเอียดการใช้งานของแต่ละฟังก์ชัน ผู้อ่านสามารถดูได้จาก Help ของ TURBO C ในที่นี้จะกล่าวถึงเฉพาะฟังก์ชัน inportb และ outportb

inportb : เป็นฟังก์ชันการอ่านข้อมูลจาก I/O โดยข้อมูลดังกล่าวมีขนาด 1 ไบต์ ทำหน้าที่เหมือนคำสั่ง IN ในภาษาแอสเซมบลี

รูปแบบการใช้ : inportb(หมายเลขแอดเดรสของ I/O)

รายละเอียด : หมายเลขแอดเดรสของ I/O เป็นชนิดจำนวนเต็ม (int) ค่าที่ส่งกลับของฟังก์ชัน เป็นชนิด char ซึ่งมีขนาด 1 ไบต์

ตัวอย่าง : การอ่านค่าจาก I/O หมายเลขแอดเดรส C1h (h ย่อมาจาก hexadecimal) มาเก็บไว้ที่ตัวแปร X สามารถเขียนได้ดังนี้

```
X = inportb(0XC1) ;
```

ในภาษา C เราสามารถเขียนเลขฐานสิบหกได้โดยการนำหน้าตัวเลขด้วย 0X เช่น 0XF8, 0X11 เป็นต้น และถ้าเขียนตัวเลขโดยไม่มีการกำหนดสัญลักษณ์ใด ๆ จะหมายถึง เลขฐานสิบ ตัวอย่างเช่น ถ้าเขียนว่า inport(97) จะหมายถึง การอ่านค่าจาก I/O ที่มีแอดเดรสหมายเลข 97 ฐานสิบ (หรือเท่ากับ 41 ฐานสิบหก) จำนวนสายที่ใช้ในการอ้างแอดเดรสของ I/O บนเครื่อง IBM PC มีจำนวนทั้งหมด 10 เส้น คือ A0-A9 ซึ่งทำให้สามารถอ้าง I/O ได้จำนวนทั้งหมด $2^{10}=1024$ ซึ่งค่าดังกล่าวสามารถอ้างได้ครบโดยใช้จำนวนชนิดจำนวนเต็ม จากข้อกำหนดของฟังก์ชัน หมายเลขแอดเดรสของ I/O เป็นจำนวนชนิดจำนวนเต็ม ฟังก์ชันจึงสามารถอ้างแอดเดรสของ I/O ได้อย่างครบถ้วนทุก I/O

เราสามารถอ่านข้อมูลจาก I/O โดยไม่นำค่าที่ส่งกลับของฟังก์ชันมาใช้งานใด ๆ ซึ่งเป็นเทคนิคการสร้างบัสไซเคิลของการอ่านข้อมูลจาก I/O บนระบบบัส เพื่อควบคุมสัญญาณต่าง ๆ บนระบบบัสและนำมาใช้งานอื่น ๆ ต่อไป เช่น

```
inportb(0XC5) ;
```

outportb : เป็นฟังก์ชันการเขียนข้อมูลขนาด 1 ไบต์ ไปยังพอร์ต I/O ทำหน้าที่เหมือนคำสั่ง OUT ในภาษาแอสเซมบลี

รูปแบบการใช้ : outportb(หมายเลขแอดเดรสของ I/O , ข้อมูลขนาด 1 ไบต์)

รายละเอียด : หมายเลขแอดเดรสของ I/O เป็นชนิดจำนวนเต็ม(int) ข้อมูลเป็นชนิด char ฟังก์ชันไม่มีการส่งค่ากลับ

ตัวอย่าง : การเขียนข้อมูล 80h ไปยังพอร์ต I/O หมายเลขแอดเดรส 3E8h สามารถเขียนได้ว่า

```
outportb(0X3E8 , 0X80) ;
```

### 2.3 การใช้ In-Line Assembly

เราสามารถเขียนคำสั่งแอสเซมบลีแทรกเข้าไปในโปรแกรมภาษา C ได้ ซึ่งเรียกว่า In-line assembly และ คำสั่งในการเขียนอ่านข้อมูลกับ I/O ของภาษาแอสเซมบลี คือ คำสั่ง IN และ OUT ซึ่งมีรูปแบบการใช้ดังนี้

IN เป็นคำสั่งที่ใช้ในการอ่านข้อมูลจาก I/O ขนาด 1 ไบต์ หรือ 1 เวิร์ด (2 ไบต์) ไปเก็บไว้ใน accumulator

รูปแบบการใช้ : IN accumulator , immed.8
IN accumulator , DX

รายละเอียด : immed.8 คือ หมายเลขแอดเดรสของ I/O และ DX เก็บค่าหมายเลขแอดเดรสของ I/O ในกรณีที่อ้างแอดเดรสของ I/O โดยใช้ immed.8 จะอ้างจำนวน I/O ได้ 256 I/O (หมายเลข 0-255) ในกรณีใช้ DX จะอ้างได้ทุก I/O

ตัวอย่าง :

```
IN   AL , 45          ; Read 1 byte data
```

```
IN   AX , DX          ; Read 2 bytes data
IN   AL , DX
```

OUT เป็นคำสั่งที่ใช้ในการเขียนข้อมูลขนาด 1 ไบต์ หรือ 1 เวิร์ด จาก accumulator ไปยัง I/O

รูปแบบการใช้ : OUT immed.8 , accumulator

OUT DX , accumulator

รายละเอียด : เช่นเดียวกับคำสั่ง IN

ตัวอย่าง :

```
OUT   254 , AX        ; Write 2 bytes data
OUT   DX , AL         ; Write 1 byte data
```

การใช้ In-line assembly จะมีรูปแบบการใช้ดังนี้

```
asm  <OPCODE>  <OPERAND>  <; or  newline>
```

ซึ่งสามารถแทรกในส่วนใดของโปรแกรมก็ได้ ในส่วนของ OPCODE ก็คือ คำสั่งภาษาแอสเซมบลี และในส่วนของ OPERAND เราสามารถใช้มาโครที่ได้นิยามขึ้นในโปรแกรมภาษา C แทนได้ ตัวอย่างเช่น

```
#define  PORTA  0XC8
         :
asm  OUT  PORTA , AL
         :
```

ในกรณีต้องการเขียนคำสั่งแอสเซมบลีหลาย ๆ คำสั่งติดกัน คำสั่งแอสเซมบลีแต่ละคำสั่งจะต้องมี asm นำหน้าเสมอ การจบคำสั่ง ทำโดยการใส่เครื่องหมาย ; หรือ หากไม่ใช้เครื่องหมาย ; ก็สามารถใช้การขึ้นบรรทัดใหม่ ดังนั้นคำสั่ง In-line assembly อาจไม่ได้จบประโยคด้วยเครื่องหมาย ; เช่น

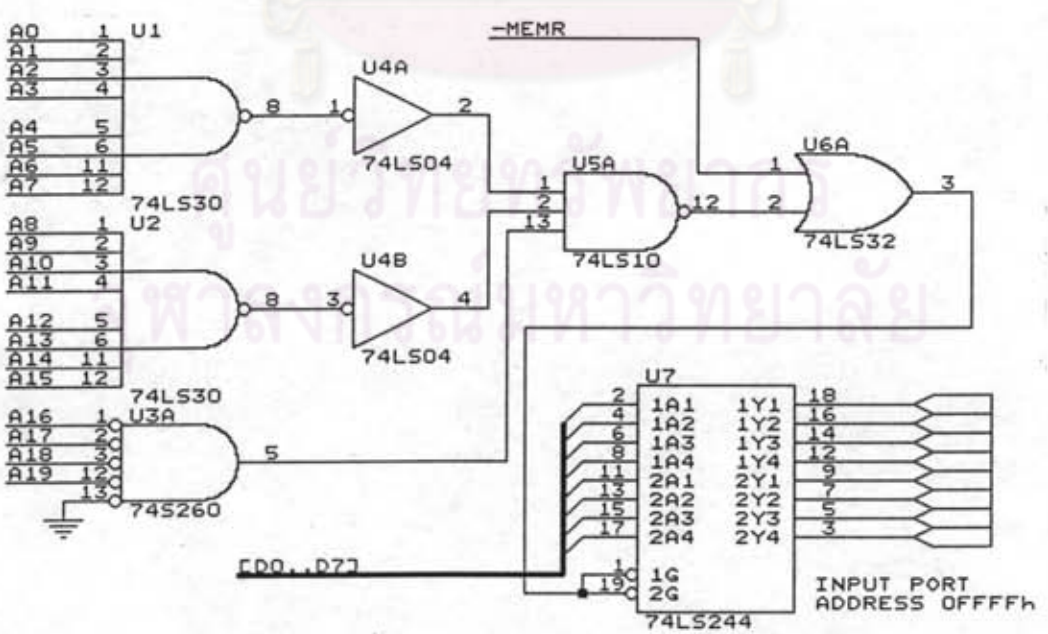
```
asm mov DX, 0x378   /* This line is In-line assembly */
asm out  DX, 21
i = i + 1;          /* This line is C statement */
```

มีข้อควรระวังในการใช้ In-line assembly คือ คำสั่งภาษาแอสเซมบลีมักจะมีการใช้งานรีจิสเตอร์ของ CPU ซึ่งในขณะนั้นโปรแกรมภาษา C อาจมีการใช้งานรีจิสเตอร์ดังกล่าวอยู่ โดยเฉพาะอย่างยิ่งในกรณีใช้ตัวแปรแบบรีจิสเตอร์ จึงอาจทำให้เกิดการเปลี่ยนแปลงค่าในรีจิส

เตอร์อย่างไม่สามารถคาดเดาผลได้ ซึ่งทำให้โปรแกรมทำงานผิดพลาดได้ จึงเป็นข้อที่ควรตระหนักในการใช้งาน อย่างไรก็ตามรีจิสเตอร์ที่ควรระวังคือเซกเมนต์รีจิสเตอร์และ BI

## 2.4 การเขียนอ่านข้อมูลกับ I/O ชนิด Memory mapped I/O

เครื่องคอมพิวเตอร์บางรุ่น เช่น เครื่องคอมพิวเตอร์ที่ใช้ไมโครโปรเซสเซอร์ตระกูล Motorola มีการจัด I/O โดยการมอง I/O เป็นเสมือนหน่วยความจำ ในระบบจะไม่มีสัญญาณ $\overline{IOR}$ , $\overline{IOW}$ แยกกับสัญญาณ $\overline{MEMR}$ , $\overline{MEMW}$ จะมีแต่เพียง $\overline{MEMR}$ , $\overline{MEMW}$ ในลักษณะนี้จะมีข้อเสียคือ เนื้อที่หน่วยความจำจะไม่สามารถใช้ได้อย่างเต็มที่ เนื่องจากจะต้องแบ่งแอดเดรสบางส่วนเพื่อใช้กับ I/O และในการถอดรหัสแอดเดรสจะต้องทำการถอดรหัสบัสแอดเดรสจำนวนมาก ทำให้วงจรถอดรหัสมีความยุ่งยาก แต่ก็มีข้อดีคือ ช่วยลดจำนวนสายสัญญาณลง และทำให้ไม่ต้องมีคำสั่งในการเขียนอ่าน I/O แต่จะใช้คำสั่งการเขียนอ่านหน่วยความจำ ซึ่งทำให้การจัดการข้อมูลกับ I/O มีความยืดหยุ่นและประสิทธิภาพสูงกว่ามาก มีการอ้างแอดเดรสแบบต่าง ๆ มากกว่าซึ่งต่างจากคำสั่งการเขียนอ่าน I/O ปกติที่มีการอ้างแอดเดรสเพียง 1-2 แบบ รูปที่ ค.3 แสดงตัวอย่างการเชื่อมต่อ I/O แบบ memeory mapped จะสังเกตเห็นว่าวงจรถอดรหัสแอดเดรสแบบหน่วยความจำจะไม่ใช้สัญญาณ AEN มาช่วยในการถอดรหัสด้วย




รูปที่ ค.3 ตัวอย่างการเชื่อมต่อ I/O แบบ memory mapped

การเขียนอ่านข้อมูลกับ I/O ชนิดนี้ จะใช้วิธีเช่นเดียวกับการเขียนอ่านหน่วยความจำ ซึ่งในภาษา C เราสามารถใช้พอยน์เตอร์ในการทำหน้าที่นี้ได้ นอกจากนี้เราสามารถใช้ฟังก์ชัน pokeb , peekb รายละเอียดของการเขียนอ่านหน่วยความจำสามารถดูได้จากหัวข้อที่ 5 ตัวอย่างข้างล่างนี้จะแสดงการเขียนอ่านข้อมูลกับ I/O โดยใช้พอยน์เตอร์ ซึ่งจะเห็นได้ว่ามีความยืดหยุ่นสูงกว่า I/O ชนิดธรรมดามาก

```
#define PORT 0X0FFFF
#define LED 0X0FFFE
main( )
{
        int x;
        char far *portptr  =  (char far *)PORT;        /* declare portptr point to PORT */
        char far *ledprt  =  (char far *)LED;          /* declare ledptr point to LED port */
        x = (*portptr)+2;       /* read data from PORT add with 2 */
        *ledptr = x;            /* write data to LED port */
}
```

## 2.5 การเขียนอ่านข้อมูลกับ I/O บนชุดทดลอง

เนื่องจากชุดทดลองถูกสร้างขึ้นเพื่อต่อเข้ากับเครื่อง IBM PC โดยผ่านทางพอร์ตขนาน ไม่ได้ใช้ระบบบัสของ IBM PC โดยตรง ซึ่งจะเห็นว่าพอร์ตขนานเมื่อมองจาก IBM PC จะเป็น I/O ที่ใช้แอดเดรสจำนวนเพียง 3 แอดเดรส แต่การอ้างแอดเดรสของ I/O บนชุดทดลองสามารถทำได้จำนวนทั้งหมด 256 แอดเดรส ทั้งนี้เป็นเพราะ I/O ที่อยู่บนชุดทดลองมีลักษณะเป็น I/O จำลองที่ ไม่เหมือนกับ I/O ที่ต่อกับ IBM PC โดยตรง ดังนั้นการเขียนอ่านข้อมูลกับ I/O บนชุดทดลองจึงไม่เหมือนกับการเขียนอ่านข้อมูลกับ I/O ปกติ การเขียนอ่านข้อมูลกับ I/O บนชุดทดลองจะกระทำโดยใช้ชุดคำสั่งของการเขียนอ่านข้อมูลกับ I/O ปกติหลาย ๆ คำสั่งมาประกอบกันรวมทั้งคำสั่งอื่น ๆ ซึ่งถูกสร้างไว้ในลักษณะของฟังก์ชัน ประกอบไปด้วยฟังก์ชัน IN , OUT ฟังก์ชัน IN เป็นฟังก์ชันในการอ่านข้อมูลจาก I/O (ของชุดทดลอง) ฟังก์ชัน OUT เป็น

ฟังก์ชันในการเขียนข้อมูลไปยัง I/O ฟังก์ชันทั้งสองมีลักษณะการใช้งานเช่นเดียวกับฟังก์ชัน inportb และ outportb มีความแตกต่างกันตรงที่แอดเดรสของฟังก์ชัน IN , OUT จะอ้างได้ในช่วง 0-255 (ฐานสิบ) คือ มีขนาด 1 ไบต์ แต่แอดเดรสของฟังก์ชัน inportb , outportb จะอ้างแอดเดรสได้ตลอดช่วงแอดเดรสของ I/O บน IBM PC ฟังกชัน IN , OUT ทั้งสองได้ถูกประกาศไว้ใน header file ชื่อ "edl.h" ฉะนั้นในการใช้งานฟังก์ชันทั้งสองจะต้องมีประโยค

```
#include <edl.h>
```

บรรจุอยู่ในส่วน C preprocessor ของโปรแกรม (โดยปกติอยู่ที่ส่วนต้นโปรแกรม) อย่างไรก็ตาม จะต้องทำการเพิ่มมอดูลฟังก์ชันทั้งสองเข้าไปในไลบรารีของ TURBO C ก่อน จึงจะสามารถเรียกใช้ฟังก์ชันทั้งสองได้ โดยการใช้โปรแกรม Install ของชุดทดลอง (ดูรายละเอียดจากคู่มือการใช้งานชุดทดลอง) รูปที่ ค.4 แสดงตัวอย่างโปรแกรมซึ่งเรียกใช้งานฟังก์ชันทั้งสอง

```
#include<edl.h>
#define LED 0XC0
#define SW 0XC1
main( ){    out(0XC0 , in(0XC1));
}
```

รูปที่ ค.4 โปรแกรมตัวอย่างการเขียนอ่าน I/O บนชุดทดลอง

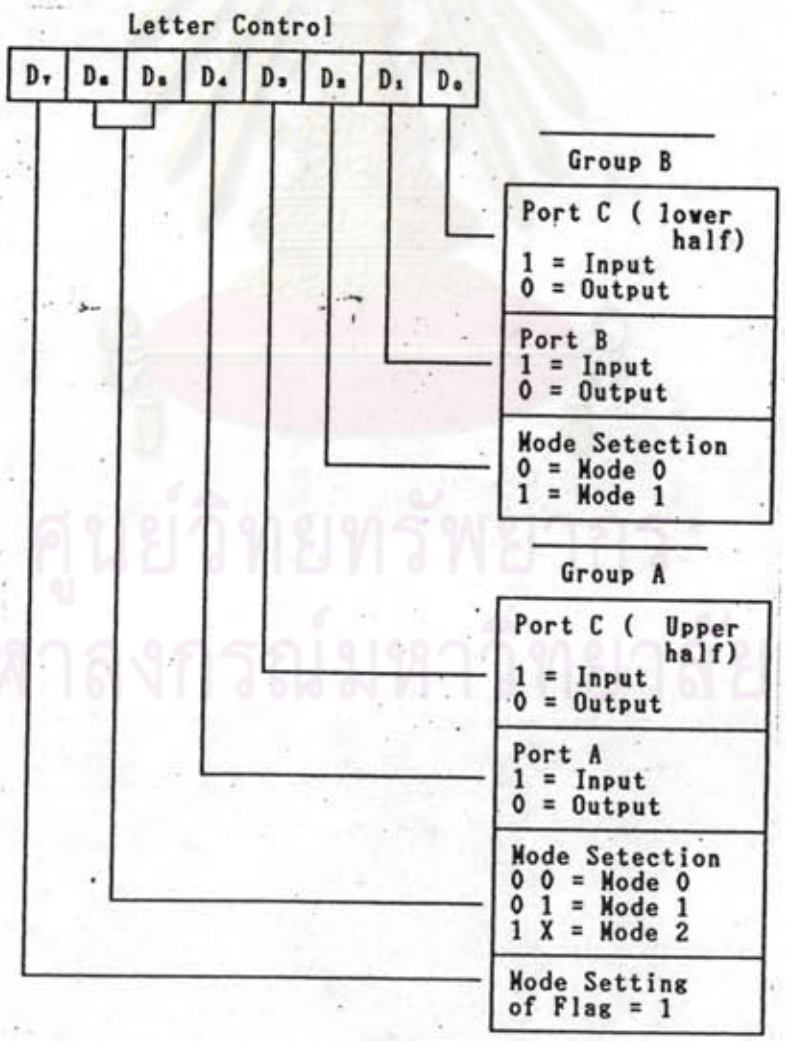
## 3. การโปรแกรมชิปซัพพอร์ต

ในส่วนนี้จะกล่าวถึง วิธีการโปรแกรมชิปต่าง ๆ ที่มักใช้ในงานควบคุม เช่น 8253 ตัวตั้งเวลา/ตัวนับที่โปรแกรมได้ 8255-parallel peripheral interface และการควบคุมไอซี A/D, D/A การโปรแกรมรีจิสเตอร์ที่ทำหน้าที่ควบคุมการทำงานบนชุดทดลอง การโปรแกรมชิปต่าง ๆ มีหลักการที่เหมือนกัน คือ อาศัยวิธีการเขียนอ่านข้อมูลกับ I/O ดังที่ได้กล่าวไว้ในหัวข้อที่ 2 รีจิสเตอร์ที่อยู่บนชิปอาจมีจำนวน 1 ตัว หรือมากกว่า และจะมีหมายเลขแอดเดรสประจำแต่ละรีจิสเตอร์ โดยในการโปรแกรม จะมีรายละเอียดซึ่งขึ้นกับชิปแต่ละตัว

### 3.1 การโปรแกรมชิป 8255

8255 ทำหน้าที่เป็นพอร์ตสัญญาณเข้า/ออก ที่สามารถโปรแกรมได้ แบ่งออกได้เป็น 3 พอร์ต คือ พอร์ต A, พอร์ต B และพอร์ต C แต่ละพอร์ตมีขนาด 8 บิต จากสามพอร์ตนี้สามารถแบ่งออกเป็น 2 กลุ่ม คือ กลุ่ม A อันประกอบด้วยพอร์ต A และพอร์ต C ครึ่งบน (PC7-PC4) และกลุ่ม B ประกอบด้วยพอร์ต B และพอร์ต C ครึ่งล่าง (PC3-PC0) 8255 มีโหมดการทำงาน 3 โหมด คือ โหมด 0, 1 และ 2 โหมด 0 เป็นโหมดพอร์ตสัญญาณเข้า/ออกพื้นฐานโดยไม่มีสัญญาณ handshake โหมด 1 เป็นโหมดพอร์ตสัญญาณเข้า/ออก โดยมีพอร์ต C ทำหน้าที่เป็นสัญญาณ handshake และโหมด 2 ซึ่งเป็นโหมดที่พอร์ต A ทำหน้าที่เป็นพอร์ตแบบ bi-directional ซึ่งหมายถึงข้อมูลสามารถส่งเข้าหรือออกจากพอร์ตเดียวกัน โดยใช้สายเส้นเดียวกันได้ และมีพอร์ต C ทำหน้าที่เป็นสัญญาณ handshake

รีจิสเตอร์ควบคุมภายใน 8255 แสดงดังรูปข้างล่าง




รูปที่ ค.5 8255 control word

สมมติว่า 8255 ถูกติดตั้งไว้ที่พอร์ตหมายเลข 70h-73h โดยมีรายละเอียดดังนี้

PORT A 70h

PORT B 71h

PORT C 72h

CONTROL 73h

ถ้าเราต้องการโปรแกรม 8255 เพื่อให้ PORT A เป็น output port PORT B เป็น input port และ PORT C เป็น output port และทำงานในโหมด 0 ทั้งหมด เราสามารถเขียน control word ได้เป็น

| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
|---|---|---|---|---|---|---|---|

= 82h

ตัวอย่างโปรแกรมข้างล่างนี้ แสดงวิธีการโปรแกรม 8255 ให้ทำงานในโหมดดังกล่าว พร้อมกับเขียนค่า AAh ไปที่พอร์ต A และค่า FFh ไปที่พอร์ต C

```
#define  PORTA  0X70
#define  PORTB  0X71
#define  PORTC  0X72
#define  CONTROL  0X73
main( )
{
        outportb(CONTROL , 0X82);       /*  OUT  CONTROL  WORD   */
        outportb(PORTA , 0XAA);         /*  OUT DATA AAh TO PORTA  */
        outportb(PORTC , 0XFF);         /*  OUT DATA FFh TO PORTC  */
}
```

### 3.2 การโปรแกรมชิป 8253

ไอซี 8253 ประกอบด้วยรีจิสเตอร์ 4 ตัว คือ

counter register # 0

counter register # 1

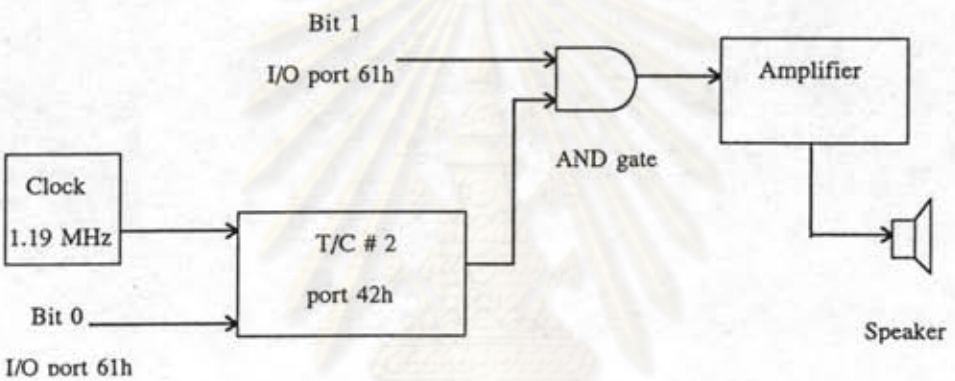
counter register # 2

Control Word register

การโปรแกรมจะเริ่มจากการโปรแกรม control word register ซึ่งมีขนาด 1 ไบต์ แล้วตามด้วย counter register ซึ่งอาจโปรแกรม 1 ไบต์ หรือ 2 ไบต์ ขึ้นอยู่กับการโปรแกรม control word register ในตอนแรก

การสร้างเสียงด้วยตัวตั้งเวลา/ตัวนับ

สำหรับเครื่อง IBM PC จะมีชิป 8253 จะอยู่ที่พอร์ตแอดเดรสหมายเลข 40-43h งตัวนับแซนแนล 2 จะต่อกับวงจรควบคุมลำโพงเพื่อทำหน้าที่สร้างเสียง รูปที่ ค.6 แสดงองค์ประกอบทางฮาร์ดแวร์ที่ใช้กำเนิดเสียง




รูปที่ ค.6 องค์ประกอบทางฮาร์ดแวร์ที่ใช้กำเนิดเสียงบน IBM PC

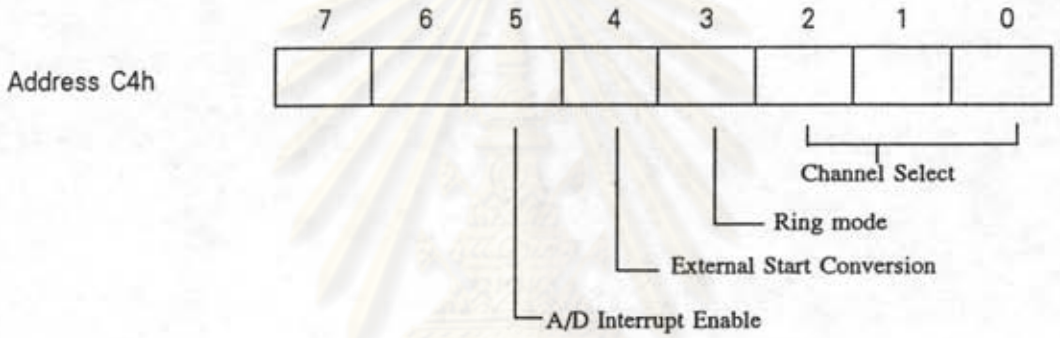
โปรแกรมตัวอย่างแสดงการกำเนิดเสียงบนเครื่อง IBM PC

```
#include <dos.h>
main( )
{
        outportb(0X43 , 0XB4);          /* OUT DATA TO CONTROL WORD */
                                        /*  SET 8253 TO MODE 2      */
        outportb(0X42 , 0XA6);          /* LOAD counter register with ...*/
        outportb(0X42 , 0x04);          /*  FOR Frequency 1,000 Hz  */
        outportb(0X61 , inportb(0X61)|0X03); /* OPEN 8253 Gate  */
                                        /*      enable data being sent to speaker */
}
```

### 3.3 การโปรแกรมรีจิสเตอร์บนชุดทดลอง

รีจิสเตอร์บนชุดทดลองประกอบด้วย รีจิสเตอร์บนมอดูล ADAC จำนวน 1 ตัว (ไม่รวม data register) และรีจิสเตอร์บนมอดูล T/C จำนวน 3 ตัว ในที่นี้จะกล่าวถึงตัวอย่างการโปรแกรมรีจิสเตอร์บนมอดูล ADAC

มอดูล ADAC อยู่ที่ตำแหน่งแอดเดรสหมายเลข C4h-C6h โดยตำแหน่ง C4h เป็นรีจิสเตอร์ควบคุมของวงจร A/D ตำแหน่ง C5h เป็นรีจิสเตอร์ข้อมูลของ A/D ตำแหน่ง C6h เป็นรีจิสเตอร์ข้อมูลของ D/A การใช้งาน A/D จะเริ่มต้นด้วยการโปรแกรมรีจิสเตอร์ควบคุม ซึ่งมีลักษณะดังรูป




รูปที่ ค.7 รีจิสเตอร์ควบคุมของ A/D

บิต 2-0 ทำหน้าที่ในการเลือกแชนแนลของการแปลงสัญญาณ เมื่อต้องการเลือกแชนแนลหมายเลขใด ก็ให้เขียนหมายเลขนั้นในลักษณะของเลขไบนารีลงใน 3 บิตดังกล่าว ตัวอย่างเช่น ถ้าต้องการเลือกแชนแนล 0 ก็ให้เขียนค่า 000b (b=binary) ลงในบิต 2-0 ถ้าต้องการเลือกแชนแนล 4 ทำโดยเขียนค่า 100b เป็นต้น แต่ละแชนแนลของ A/D จะเป็นตัวรับสัญญาณแอนะลอกจากแหล่งต่าง ๆ ดังแสดงในตาราง

| Channel | Analog Source |
|---|---|
| 0 | potentiometer |
| 1 | microphone |
| 2 | LM335 |
| 3 | photo TR |
| 4 | AUX |
| 5 | A_IN 2 |
| 6 | A_IN 3 |
| 7 | A_IN 4 |

บิต 4-3 ทำหน้าที่ในการเลือกวิธีเริ่มต้นการแปลงสัญญาณ A/D เมื่อทำการเซ็ตบิต 3 เป็น 1 จะเป็นการแปลงสัญญาณแบบวงรอบ (ring mode) คือ เมื่อสิ้นสุดการแปลงสัญญาณในรอบหนึ่ง ๆ และได้ทำการอ่านค่าไปจาก A/D แล้ว ก็จะเกิดการแปลงสัญญาณในรอบถัดไปโดยอัตโนมัติ การแปลงสัญญาณแบบนี้เป็นไปอย่างต่อเนื่อง ผู้ใช้ไม่จำเป็นจะต้องทำการควบคุมสัญญาณเริ่มต้นการแปลง สามารถอ่านข้อมูลไปจาก A/D ได้โดยตรง เมื่อเซ็ตบิต 4 เป็น 1 จะเป็นการแปลงสัญญาณโดยอาศัยการกระตุ้นจากสัญญาณภายนอกเพื่อเริ่มต้นการแปลง ซึ่งสัญญาณภายนอกจะอยู่ที่ขั้ว EXT_ST CONV บนมอดูล ในการใช้งานบิต 4 และ 3 ไม่ควรจะถูกเซ็ตพร้อมกัน แต่ถ้าถูกเซ็ตพร้อมกันจะเท่ากับไม่เลือกทั้งสองวิธี

บิต 5 ทำหน้าที่เอนาเบิลสัญญาณร้องขอการอินเตอร์รัปต์ที่ส่งไปยัง CPU ซึ่งเกิดจากขา INTR ของชิป ADC0804 ที่ทำหน้าที่ในการแปลงสัญญาณ A/D โดยจะส่งสัญญาณไปยังอินเตอร์รัปต์ CPU ทุกครั้งที่สิ้นสุดการแปลงสัญญาณในแต่ละรอบ

การโปรแกรมรีจิสเตอร์ดังกล่าว และการอ่านค่าจาก A/D แสดงดังตัวอย่าง โดยโปรแกรมให้ A/D ทำงานใน ring mode เลือกแชนแนลที่ 2 และดิสเอเบิลการอินเตอร์รัปต์

```
#include<edl.h>
#include<dos.h>
#include<stdio.h>
#define  A_CTRL  0xC4
#define  A_DATA  0xC5
main( )
{
        out(A_CTRL,0X0A);              /* program A/D control register */
        for( ; ; ){
                delay(1);              /* delay 1 ms */
                printf("%u " , in(A_DATA));   /* read data from A/D then*/
        }                              /*      output to screen   */
}
```

จากโปรแกรม จะสังเกตว่ามีประโยค delay(1) ทำหน้าที่ในการหน่วงเวลาประมาณ 1 mS เพื่อให้แน่ใจว่า A/D ได้แปลงสัญญาณเสร็จเรียบร้อยแล้ว ซึ่ง A/D ใช้เวลาในการแปลง

สัญญาณประมาณ 100 µs ฟังก์ชัน delay ถูกนิยามไว้ในไฟล์ "dos.h" อย่างไรก็ตาม การหน่วงเวลา 1 ms ซึ่งยังไม่รวมเวลาของคำสั่ง printf และ for อาจจะเป็นเวลาที่นานเกินไปเมื่อเทียบกับ 100 µs ในกรณีที่ผู้ใช้ต้องการความเร็วที่สูงขึ้นในการอ่านข้อมูลก็สามารถเขียนฟังก์ชันหน่วงเวลาขึ้นเองได้

## 4. การเรียกใช้อินเตอร์รัปต์ของ BIOS หรือ DOS

BIOS และ DOS ของ IBM PC ได้จัดเตรียมรูทีนต่าง ๆ ที่เป็นประโยชน์ที่ผู้ใช้สามารถเรียกใช้งานได้มากมาย รูทีนเหล่านี้มักจะเกี่ยวข้องกับการติดต่อควบคุมอุปกรณ์ฮาร์ดแวร์ต่างๆ เช่น การควบคุมจอภาพ คีย์บอร์ด เครื่องพิมพ์ ดิสค์ไดรฟ์ พอร์ตอนุกรม เป็นต้น ถ้าผู้ใช้สามารถเรียกใช้งานรูทีนเหล่านี้ ก็จะทำให้ลดเวลาในการเขียนรูทีนเพื่อควบคุมอุปกรณ์ต่างๆ ลงไปมาก ทำให้สามารถพัฒนาโครงสร้างโปรแกรม อัลกอริทึมในการแก้ปัญหาได้อย่างเต็มที่ รูทีนเหล่านี้ถูกใช้อย่างแพร่หลายและสามารถถูกเรียกใช้จากโปรแกรมภาษาระดับสูงอย่างเช่นภาษา C ได้

การเรียกใช้งานรูทีนเหล่านี้จะกระทำผ่านซอฟต์แวร์อินเตอร์รัปต์ โดยใช้คำสั่งภาษาแอสเซมบลี

```
INT  intr_num
```

intr_num เป็นหมายเลขของการอินเตอร์รัปต์ อินเตอร์รัปต์แต่ละหมายเลขจะประกอบด้วยฟังก์ชันย่อย ซึ่งเป็นรูทีนที่ผู้ใช้จะติดต่อด้วย การเลือกฟังก์ชันทำโดยการกำหนดหมายเลขของฟังก์ชันลงในรีจิสเตอร์ตัวที่กำหนด ซึ่งโดยทั่วไปหมายเลขฟังก์ชันจะถูกกำหนดลงในรีจิสเตอร์ AH แล้วจึงทำการเรียกอินเตอร์รัปต์หมายเลขนั้น ๆ ตัวอย่างเช่น การเรียกฟังก์ชัน 0 ของอินเตอร์รัปต์หมายเลข 14h (ให้บริการเกี่ยวกับการสื่อสารผ่าน RS-232) กระทำได้โดยคำสั่ง

```
MOV  AH,0
INT  14                ;call function 0 of int 14h
```

การส่งผ่านค่าพารามิเตอร์ต่าง ๆ ให้แก่ฟังก์ชัน หรือการส่งค่ากลับจากฟังก์ชันกระทำผ่านรีจิสเตอร์ต่าง ๆ สำหรับอินเตอร์รัปต์หมายเลข 21h ประกอบด้วยฟังก์ชันย่อยต่าง ๆ มากมาย ฟังก์ชันเหล่านี้เป็นของ DOS จึงถูกเรียกว่า ฟังก์ชันดอส (DOS functions) ฉะนั้นเมื่อกล่าวถึงฟังก์ชันดอสหมายเลข 01h จะเป็นที่เข้าใจว่าหมายถึงฟังก์ชันหมายเลข 01h ของอินเตอร์รัปต์หมายเลข 21h (แท้ที่จริงยังมีฟังก์ชันอื่น ๆ อีกที่เป็นของ DOS) รายละเอียดการใช้งาน

อินเตอร์รัปต์หมายเลขต่างๆและฟังก์ชันผู้ใช้สามารถอ่านได้จากหนังสือเทคโนโลยีไมโครคอมพิวเตอร์ 16 บิต [18] หรือเรียกดูจากโปรแกรม Expert Help (EH)

การเรียกฟังก์ชันของ BIOS หรือ DOS จากภาษา C จะกระทำผ่านฟังก์ชันซึ่งถูกสร้างไว้แล้วในไลบรารี ฟังก์ชันเหล่านี้ประกอบด้วย

| | | |
|---|---|---|
| int86 | intdos | intr |
| int86x | intdosx | geninterrupt |

รายละเอียดการใช้งานของแต่ละฟังก์ชัน ผู้ใช้สามารถอ่านได้จากหนังสือ TURBO C Reference Guide ในที่นี้จะยกตัวอย่างการใช้ฟังก์ชัน intr86 และ geninterrupt

### 4.1 ฟังก์ชัน int86

รูปแบบการใช้ : #include<dos.h>

```
int int86(int intr_num , union REGS *inregs ,
          union REGS *outregs);
```

intr_num คือ หมายเลขของการอินเตอร์รัปต์ การส่งค่าไปกลับกับฟังก์ชันจะกระทำผ่านตัวแปรชนิด REGS ซึ่งนิยามไว้ใน dos.h มีรายละเอียดดังนี้

```
struct WORDREGS {
        unsigned int as,bx,cx,dx,si,di,cflag,flags;
        };
struct BYTEREGS {
        unsigned char al,ah,bl,bh,cl,ch,dl,dh;
        };
union   REGS    {
        struct WORDREGS x;
        struct BYTEREGS  h;
        };
```

ตัวอย่างการเรียกใช้ฟังก์ชันดังกล่าว แสดงได้ดังนี้

```
union REGS r;
r.h.ah = 2;              /* send character to display function */
r.h.al = 0X41;           /* character to send out */
int86(0X21 , &r ,&r);    /* call BIOS INT 21h */
```

รูปที่ ค.8 การเรียกใช้ฟังก์ชันดอส

4.2 **ฟังก์ชัน** geninterrupt

รูปแบบการใช้ : include<dos.h>

```
void geninterrupt(int intr_num);
```

ฟังก์ชันนี้ไม่มีการส่งผ่านค่าให้แก่รีจิสเตอร์ ผู้ใช้จะต้องกำหนดค่าลงในรีจิสเตอร์เองก่อนเรียกฟังก์ชัน TURBO C ได้เตรียม pseudo variable ไว้เพื่อให้ผู้ใช้สามารถกำหนดค่าให้แก่รีจิสเตอร์ได้โดยตรง ตัวอย่างเช่น

```
_AX = 10 ;
_DL = 5  ;
```

เป็นการกำหนดค่าให้รีจิสเตอร์ ax มีค่าเป็น 10 และ รีจิสเตอร์ dl มีค่าเป็น 5 ฉะนั้นจากตัวอย่างรูปที่ ค.8 เราสามารถใช้ฟังก์ชัน geninterrupt แทนได้ดังนี้

```
_AH = 2;
_DL = 0X41;              /* character 'A' */
geninterrupt(0X21);
```

สำหรับรายละเอียดของการใช้ pseudo variable ผู้ใช้สามารถอ่านได้จากหนังสือ Turbo C User's Guide

5. การเข้าถึงหน่วยความจำโดยตรง

ในโปรแกรมภาษาต่าง ๆ เมื่อเราอ้างถึงตัวแปรที่ใช้งาน เช่น a=3 เราไม่สามารถทราบได้ว่าตัวแปร a ถูกเก็บ ณ ตำแหน่งแอดเดรสใดของหน่วยความจำ จะเป็นหน้าที่ของคอมไพเลอร์ในการจัดสรรเนื้อที่ในหน่วยความจำให้แก่ตัวแปร แต่ในงานบางอย่างเราอาจมีความจำเป็นที่จะต้องเข้าถึงหน่วยความจำโดยตรง ตัวอย่างเช่น การเขียนอ่านข้อมูลกับ I/O ที่เป็นชนิด memory mapped I/O การจัดการกับจอภาพโดยใช้วิธีเขียนไปยัง VIDEO RAM โดยตรง เป็นต้น การอ้างแอดเดรสของหน่วยความจำบนเครื่อง IBM PC จะใช้รีจิสเตอร์ 2 ตัว คือ เซกเมนต์รีจิสเตอร์ และ ออฟเซตรีจิสเตอร์ แต่ละตัวมีขนาด 16 บิต เพื่อจะอ้างอิงหน่วยความจำได้ในช่วง 1 Mbyte ซึ่งทำให้สามารถแบ่งการอ้างแอดเดรสของหน่วยความจำได้เป็น 2 ประเภท

1. การอ้างแอดเดรสหน่วยความจำที่อยู่ภายในขอบเขต 64 Kbyte

ในการอ้างแอดเดรสชนิดนี้จะไม่มีการเปลี่ยนแปลงค่าในเซกเมนต์รีจิสเตอร์ (มีค่าคงที่) จะเปลี่ยนแปลงเฉพาะออฟเซตรีจิสเตอร์ จึงทำให้อ้างแอดเดรสหน่วยความจำได้ในช่วง 64 Kbyte พอยน์เตอร์ที่ใช้ชี้หน่วยความจำจะใช้ชนิด near pointer

2. การอ้างแอดเดรสหน่วยความจำที่อยู่เกินขอบเขต 64 Kbyte

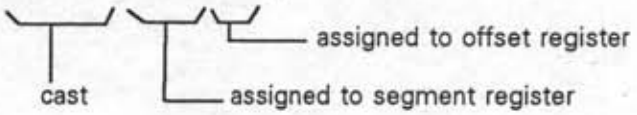
การอ้างแอดเดรสแบบนี้จะมีการเปลี่ยนแปลงค่าในเซกเมนต์รีจิสเตอร์ด้วย เพื่อให้สามารถอ้างแอดเดรสได้กว้างขึ้น คือ มีขนาด 1 Mbyte พอยน์เตอร์ที่ใช้จะต้องเป็นชนิด far pointer

การเข้าถึงหน่วยความจำอาจทำได้ 2 วิธี ดังนี้

### 5.1 การใช้พอยน์เตอร์

โดยการกำหนดให้พอยน์เตอร์ชี้ไปยังตำแหน่งแอดเดรสของหน่วยความจำที่ต้องการ เมื่อทำการกำหนดค่าให้แก่ตำแหน่งที่พอยน์เตอร์ชี้อยู่ ก็จะเป็นการเขียนข้อมูลลงในหน่วยความจำตำแหน่งดังกล่าว หรือเมื่อทำการอ่านค่าจากตำแหน่งที่พอยน์เตอร์ชี้อยู่ ก็จะเป็นการอ่านข้อมูลจากหน่วยความจำ พอยน์เตอร์ที่ใช้มักใช้ชนิด far pointer เนื่องจากในขณะที่โปรแกรมทำงานเราไม่สามารถทราบได้ว่า เซกเมนต์รีจิสเตอร์มีค่าเป็นเท่าไร จึงต้องทำการกำหนดค่าให้แก่เซกเมนต์รีจิสเตอร์ โดยใช้พอยน์เตอร์ชนิดดังกล่าว การประกาศตัวแปรพอยน์เตอร์ชนิด far pointer และการกำหนดค่าเริ่มต้นให้แก่ตัวแปรให้ชี้ไปยังแอดเดรสของหน่วยความจำที่ต้องการสามารถทำได้ดังตัวอย่าง

```
char far *memptr  =  (char far *)0XB8000000 ;
```



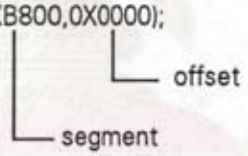

เป็นการประกาศให้ memptr เป็นตัวแปรชนิด far และให้ชี้ไปยังหน่วยความจำตำแหน่ง B800:0000 (ฐานสิบหก) จะเห็นได้ว่า เราไม่สามารถเขียนดังนี้

```
char far *memptr  =  0XB8000000;          /* wrong */
```

เนื่องจากค่า 0XB8000000 ไม่ได้เป็นค่าชนิดพอยน์เตอร์ จึงต้องใช้ cast เพื่อกำหนดให้ค่าดังกล่าวเป็นชนิดพอยน์เตอร์ จึงจะสามารถกำหนดค่าให้แก่ memptr ได้

เราอาจใช้มาโคร MK_FP (ย่อมาจาก make far pointer) ในการสร้างค่าชนิด far ponter จากเซกเมนต์รีจิสเตอร์และออฟเซตรีจิสเตอร์ได้ ตัวอย่างเช่น

```
char far *memptr  =  MK_FP(0XB800,0X0000);
```



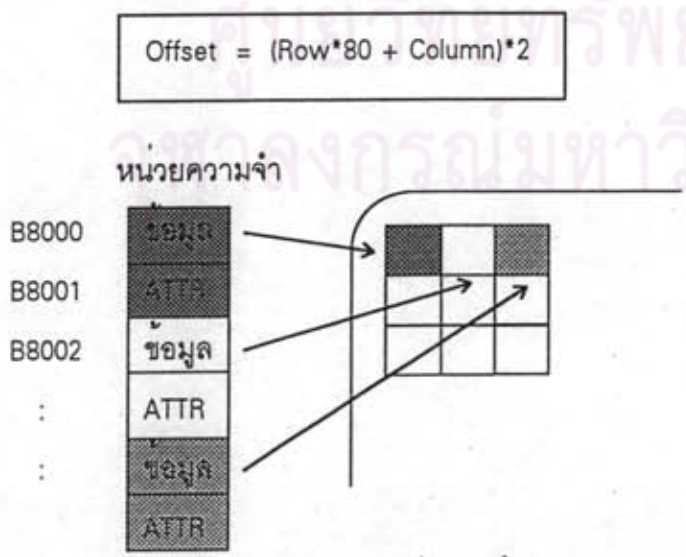

จากตัวอย่างข้างต้น เราได้กำหนดให้ตัวแปร memptr ชี้ไปยัง char (มีขนาด 1 ไบต์) เนื่องจากเราต้องการเขียนอ่านข้อมูลทีละ 1 ไบต์ อย่างไรก็ตาม เราสามารถกำหนดให้ memptr ชี้ไปยังค่าชนิดอื่น ๆ ได้ ซึ่งจะทำให้เราสามารถเขียนอ่านข้อมูลที่มีขนาดต่าง ๆ เช่น int(จำนวนเต็ม) มีขนาด 2 ไบต์ เป็นต้น การเขียนอ่านข้อมูลสามารถทำได้ดังแสดง

```
char far *memptr  =  (char far *)0xB8000000; /* assign memptr points to B8000*/
char x;
*memptr  =  0XFF;     /* write FFh to memory address B8000 */
*(memptr+1) = 0XFE;   /* write FEh to address B8001 */
memptr[2] = 0xFD;     /* write FDh to address B8002 */
x = memptr[3];        /* read data from B8003 assign to x variable */
/*———— 2 bytes(1 word) data ————————*/
int far *mptr = (int far *)0XA0000000;
```

```
int y;
mptr[0] = 0XFF80;      /* assign 80h to address A0000 */
                       /*        FFh to address A0001 */
y = mptr[2];           /* read data from address A0002 and A0003 */
                       /*          then assign to y */
```

### 5.1.1 การจัดการกับจอภาพโดยวิธีเขียนอ่านโดยตรงกับ VIDEO RAM

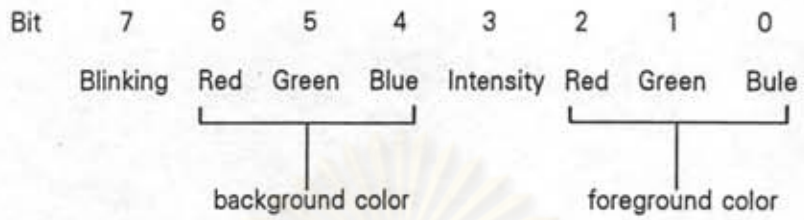
ในที่นี้จะแสดงตัวอย่างการประยุกต์ใช้พอยน์เตอร์ในการเข้าถึงหน่วยความจำโดยตรง โดยการทดลองควบคุมการแสดงผลบนจอภาพ วิธีนี้มักใช้กับงานที่ต้องการความเร็วในการแสดงผล เช่น การทำรูปเคลื่อนไหว การเลื่อนจอภาพ (การจัดการทั้งจอภาพ) เป็นต้น ซึ่งทำโดยการเขียนอ่านข้อมูลโดยตรงกับ VIDEO RAM ซึ่งเป็นหน่วยความจำที่ใช้เก็บข้อมูลที่นำไปแสดงผลบนจอภาพ พื้นฐานการแสดงผลบนจอภาพแสดงดังในรูปที่ ค.9 ชนิดการแสดงผลในที่นี้เป็นชนิด VGA ซึ่งมีตำแหน่งแอดเดรสของ VIDEO RAM ในโหมด Text เริ่มต้นที่ B8000(ฐานสิบหก) จอภาพจะถูกแบ่งออกเป็น 25 แถว 80 คอลัมน์ การคำนวณหาตำแหน่งที่อยู่ในหน่วยความจำ (ค่าออฟเซตนับจากแอดเดรสเริ่มต้น) ของแต่ละตำแหน่งบนจอภาพสามารถกระทำโดยใช้สูตรง่าย ๆ โดยการแทนค่าแถว และ คอลัมน์ลงในสูตร โดยกำหนดให้แถวที่ 1 และคอลัมน์ 1 เริ่มต้นด้วยค่า 0




ATTR = attribute

รูปที่ ค.9 พื้นฐานการแสดงผลในโหมด Text

สำหรับ attribute คือ รหัสควบคุมว่าตัวอักษรจะถูกแสดงบนจอภาพออกมาอย่างไร เช่น สีอะไร ตัวกะพริบ ตัวเข้ม เป็นต้น ข้อมูลทั้ง 8 บิตของ attribute ถูกแบ่งออกเป็นดังนี้




โปรแกรมต่อไปนี้ แสดงตัวอย่างการเปลี่ยนจอภาพที่แสดงผลอยู่ให้แสดงตัวอักษรเดิม แต่เปลี่ยนให้เป็นลักษณะตัวเข้ม และตัวกะพริบ

```
#define VGA 0XB8000000                    /* VGA VIDEO RAM address */
#define ATTR 0x88                         /* blinking , intensive attribute */
main( )
{
        char far *memptr = (char far *)VGA;          /* pointer initialization */
        int row , column;
        for(row = 0 ; row<25 ; row++)     /* loop for 25 rows 80 columns */
                for(column = 0 ; column<80 ; column++)
                        memptr[(row*80+column)*2+1]
                        = memptr[(row*80+column)*2+1]|ATTR;
}
```

## 5.2 การใช้ฟังก์ชันของภาษา C

TURBO C มีฟังก์ชันการเขียนอ่านหน่วยความจำโดยตรง คือ

pokeb

peekb

poke

peek

ฟังก์ชัน poke , peek เป็นฟังก์ชันการเขียน และอ่านข้อมูลกับหน่วยความจำตามลำดับ โดยกระทำทีละ 1 เวิร์ด (2 ไบต์) ส่วนฟังก์ชัน pokeb และ peekb เป็นฟังก์ชันการเขียนอ่านหน่วยความจำโดยกระทำทีละ 1 ไบต์ ในที่นี้จะกล่าวถึงเฉพาะฟังก์ชัน pokeb และ peekb ซึ่งมีรูปแบบการใช้งานดังนี้

pokeb เขียนข้อมูลขนาด 1 ไบต์ไปยังหน่วยความจำตำแหน่งที่ต้องการ

รูปแบบการใช้ : pokeb( segment , offset , value ) ;

รายละเอียด : segment และ offset เป็นค่า หรือตัวแปรชนิด int value เป็นชนิด char ฟังก์ชันนี้ไม่มีการส่งค่ากลับ

ตัวอย่าง : pokeb( 0XB800 , 0X0000 , 0XFF ) ;

peekb อ่านข้อมูลขนาด 1 ไบต์ จากหน่วยความจำ

รูปแบบการใช้ : peekb( segment , offset ) ;

รายละเอียด : segment และ offset เป็นค่า หรือ ตัวแปรชนิด int ค่าที่ส่งกลับของฟังก์ชันเป็นชนิด char

char X ;

ตัวอย่าง : X = peekb( 0XB800 , 0X00FF ) ;

จะเห็นว่าการใช้พอยน์เตอร์ในการเขียนอ่านหน่วยความจำ มีความสะดวกมากกว่าการใช้ฟังก์ชันทั้งสอง แต่ถ้าเป็นการเขียนอ่านข้อมูลเพียงไม่กี่ไบต์ เช่น ในการโปรแกรมค่าเริ่มต้นให้แก่ชิปซัพพอร์ต การใช้งานฟังก์ชันทั้งสองจะสะดวกกว่า เนื่องจากไม่ต้องทำการนิยามตัวแปรชนิดพอยน์เตอร์ ฟังก์ชันทั้งสองถูกนิยามใน header file "dos.h"

## 6. การจัดการการอินเตอร์รัปต์

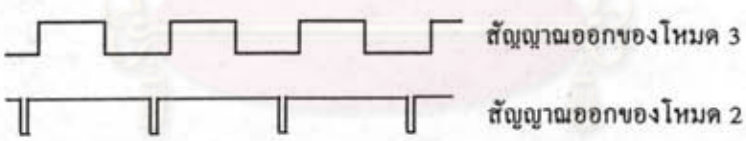
ในการใช้งานอินเตอร์รัปต์บนเครื่องไมโครคอมพิวเตอร์มักจะมีความยุ่งยากมากกว่าบนบอร์ดไมโครโปรเซสเซอร์แบบบอร์ดเดี่ยว (single board microprocessor) เนื่องจากมีการใช้ชิป Interrupt Controller และต้องคำนึงถึงตำแหน่งที่อยู่ของโปรแกรมตอบสนองการอินเตอร์รัปต์ภายในหน่วยความจำ ผลของการอินเตอร์รัปต์ที่อาจส่งผลกระทบต่อการอินเตอร์รัปต์ลำดับความสำคัญอื่น ๆ ซึ่งอาจส่งผลต่อการทำงานของโปรแกรมระบบของไมโครคอมพิวเตอร์ เนื่องจากการทำงานของไมโครคอมพิวเตอร์จำเป็นต้องอาศัยโปรแกรมระบบ (operating system) อย่างไรก็

ตาม ถ้าเราใช้งานภาษาสูง ดังเช่น ภาษา C ในการเขียนโปรแกรม ก็จะทำได้ง่ายขึ้น เนื่องจากโปรแกรมภาษาได้เตรียมการสิ่งต่าง ๆ ไว้ให้แล้ว การอินเตอร์รัปต์ถือเป็นพื้นฐานการควบคุมชนิดหนึ่งที่สำคัญ ซึ่งผู้ศึกษาด้านการควบคุมด้วยไมโครคอมพิวเตอร์ควรจะเรียนรู้

ในการดำเนินการอินเตอร์รัปต์ จะต้องเตรียมการใน 3 ส่วนด้วยกัน คือ

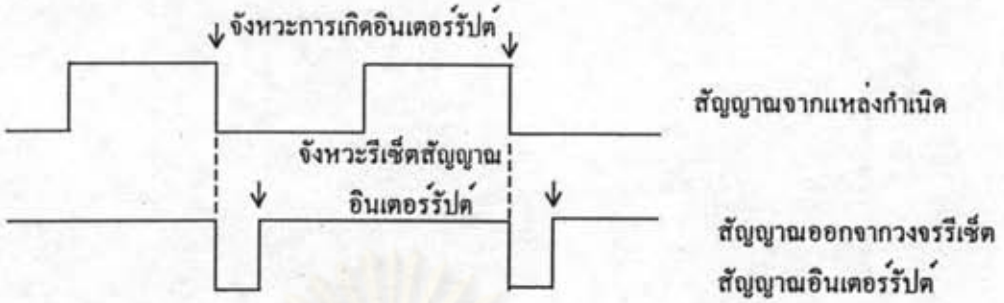
### 6.1 การจัดการทางด้าน Hardware

ถ้าเป็นกรณีของ hardware interrupt จะต้องเตรียมแหล่งกำเนิดสัญญาณอินเตอร์รัปต์ซึ่งอาจเป็น สวิตช์กด ตัวตั้งเวลา คีย์บอร์ด ฯลฯ แหล่งกำเนิดสัญญาณการอินเตอร์รัปต์ควรกำเนิดสัญญาณอินเตอร์รัปต์ที่ชัดเจน ไม่มี noise หรือ การกระเด้ง (bouncing) ของสัญญาณ ทั้งนี้เพื่อไม่ให้เกิดการอินเตอร์รัปต์โดยไม่ตั้งใจ ในกรณีที่เป็นการอินเตอร์รัปต์แบบ level trig. สัญญาณอินเตอร์รัปต์ควรมีช่วงเวลาการอินเตอร์รัปต์ที่สั้น เพื่อป้องกันการเกิดอินเตอร์รัปต์ซ้อนขึ้น ตัวอย่างวิธีการกำเนิดสัญญาณ เช่น สัญญาณอินเตอร์รัปต์ที่เกิดจากวงจรตัวตั้งเวลา / ตัวนับ (Timer / Counter) ใช้ชิปเบอร์ 8253 เราควรโปรแกรมให้ทำงานในโหมด 2 (Rate generator) ซึ่งให้รูปสัญญาณที่มีลักษณะข้างต้น แทนการใช้โหมด 3 (square wave generator) ซึ่งให้รูปสัญญาณดังแสดง

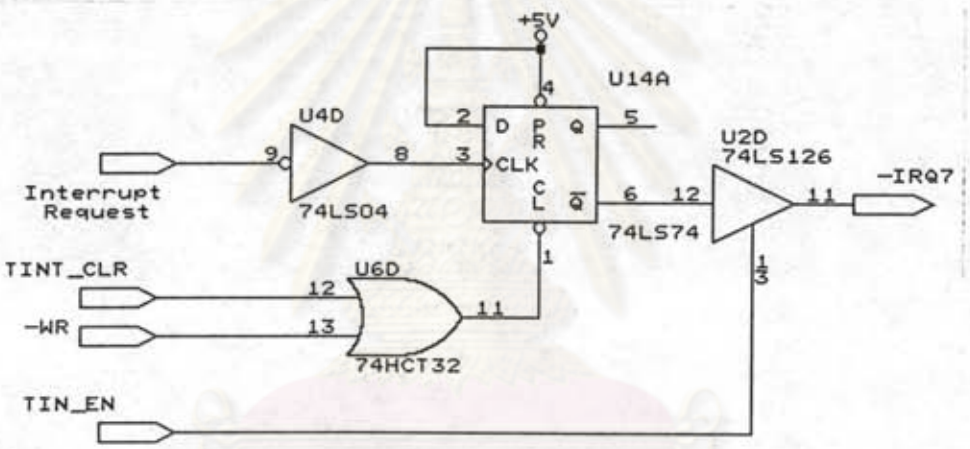



รูปที่ ค.10 เปรียบเทียบสัญญาณออกที่ได้จากวงจรตัวตั้งเวลา/ตัวนับในโหมด 2 และ 3

อย่างไรก็ตาม ในบางกรณีเราอาจไม่สามารถควบคุมความกว้างของสัญญาณที่แหล่งกำเนิดสัญญาณอินเตอร์รัปต์ได้ เช่น กรณีของสวิตช์กด เราอาจใช้วงจร monostable ในการควบคุมความกว้างของสัญญาณ หรืออาจใช้วงจรรีเซ็ตสัญญาณอินเตอร์รัปต์ ซึ่งมักจะทำการรีเซ็ตสัญญาณในช่วงของการทำงานในโปรแกรมตอบสนองการอินเตอร์รัปต์ ซึ่งวงจรดังกล่าวจะทำหน้าที่สร้างสัญญาณดังแสดงในรูปที่ ค.11 และวงจรดังกล่าวแสดงดังในรูปที่ ค.12




รูปที่ ค.11 สัญญาณที่ได้จากวงจรรีเซ็ตสัญญาณอินเตอร์รัปต์




รูปที่ ค.12 วงจรรีเซ็ตสัญญาณอินเตอร์รัปต์

6.2 โปรแกรมตอบสนองการอินเตอร์รัปต์ (Interrupt Service Routine)

ในส่วนนี้จะประกอบด้วย

a. ส่วนของการเอนาเบิลอินเตอร์รัปต์

เมื่อเกิดการอินเตอร์รัปต์ขึ้น โปรแกรมจะเก็บค่าของ IP และ CPU Flag ลงสแต็ก และทำการดิสเอเบิลอินเตอร์รัปต์แฟล็กบน CPU แล้วจึงกระโดดไปทำงานในส่วนของโปรแกรมตอบสนองการอินเตอร์รัปต์ ในช่วงนี้ถ้ามีการอินเตอร์รัปต์เข้ามาจะทำให้ CPU ไม่รับการอินเตอร์รัปต์ใด ๆ ยกเว้นการอินเตอร์รัปต์ชนิด NMI (non-maskable interrupt) จนกว่าจะเสร็จสิ้นจากการทำงานในโปรแกรมตอบสนองการอินเตอร์รัปต์ ซึ่ง CPU จะทำการ pop ค่า IP จากสแต็ก เป็นการเอนาเบิลอิน

เตอร์รัปต์แฟล็กโดยอัตโนมัติ ในช่วงนี้ถ้ามีอินเตอร์รัปต์ที่มี priority สูงและเป็นงานที่มีความสำคัญเข้ามาก็จะไม่ถูกตอบสนองไปด้วย เช่น การอินเตอร์รัปต์ของ Timer เพื่อรีเฟรชหน่วยความจำชนิดไดนามิกแรม หรือการแก้ไขเวลาของเครื่อง จึงอาจทำให้เครื่องทำงานผิดพลาดได้ ดังนั้นจึงต้องทำการเอนาเบิลอินเตอร์รัปต์แฟล็ก เพื่อให้ CPU สามารถรับการอินเตอร์รัปต์ที่เกิดขึ้นได้ คำสั่งในภาษาแอสเซมบลีที่ทำหน้าที่ดังกล่าว คือ STI (set interrupt flag) สำหรับภาษา C ได้เตรียมฟังก์ชันที่ทำหน้าที่ดังกล่าวโดยสามารถเขียนเป็นประโยค ดังนี้

```
enable( );
```

อย่างไรก็ตามถ้าเป็นอินเตอร์รัปต์ที่มี priority ต่ำกว่าหรือเท่ากับเกิดขึ้นในช่วงดังกล่าว สัญญาณอินเตอร์รัปต์จะไม่ถูกส่งมาที่ CPU โดยอัตโนมัติ (ควบคุมโดยชิป 8259) โดยปกติการเอนาเบิลอินเตอร์รัปต์แฟล็กจะต้องถูกดำเนินการเร็วที่สุดเท่าที่จะเป็นไปได้ คำสั่งการเอนาเบิลอินเตอร์รัปต์แฟล็กจึงมักจะอยู่ที่ต้นโปรแกรมตอบสนองการอินเตอร์รัปต์ สำหรับคำสั่งที่ตรงข้ามกับคำสั่งการเซตอินเตอร์รัปต์แฟล็ก คือ คำสั่ง CLI (clear interrupt flag) หรือคำสั่ง disable( ) ในภาษา C

b. ส่วนของการเก็บค่ารีจิสเตอร์ต่าง ๆ ลงสแต็ก

เมื่อเกิดการอินเตอร์รัปต์ขึ้น และโปรแกรมกระโดดไปทำงานในส่วนของโปรแกรมตอบสนองการอินเตอร์รัปต์ ในช่วงการทำงานในโปรแกรมตอบสนองการอินเตอร์รัปต์ ค่ารีจิสเตอร์ต่าง ๆ อาจถูกใช้งาน และมีค่าเปลี่ยนแปลงไป ทำให้เมื่อเสร็จสิ้นจากโปรแกรมตอบสนองการอินเตอร์รัปต์ และโปรแกรมกระโดดไปทำงานในคำสั่งต่อจากคำสั่งที่ทำงานสุดท้ายก่อนกระโดดไปยังโปรแกรมตอบสนองการอินเตอร์รัปต์ อาจทำให้เกิดความผิดพลาดได้ เนื่องจากค่าของรีจิสเตอร์ถูกเปลี่ยนแปลงไป ซึ่งเป็นการเปลี่ยนแปลงที่เราไม่สามารถคาดเดาได้ว่าจะเกิดขึ้นเมื่อไร ดังนั้นในโปรแกรมตอบสนองการอินเตอร์รัปต์จะต้องทำการเก็บค่ารีจิสเตอร์ต่าง ๆ ไว้ ก่อนจะมีการเปลี่ยนแปลงค่าในรีจิสเตอร์เหล่านี้ โดยทั่วไปการทำงานดังกล่าวจะอยู่ในส่วนต้นของโปรแกรมตอบสนองการอินเตอร์รัปต์ ในภาษาแอสเซมบลีจะทำการเก็บค่ารีจิสเตอร์ลงสแต็กโดยการใช้คำสั่ง PUSH เช่น

```
PUSH DX
PUSH AX
PUSH BX
```

สำหรับโปรแกรมที่เขียนด้วย TURBO C ผู้ใช้ไม่จำเป็นต้องจัดการกับหน้าที่ดังกล่าว เนื่องจากโปรแกรมได้จัดเตรียมฟังก์ชันชนิด Interrupt ไว้ให้ ซึ่งทำหน้าที่ดังกล่าวให้โดยอัตโนมัติ ซึ่งจะได้กล่าวถึงรายละเอียดต่อไป

c. ส่วนของการควบคุมสัญญาณอินเตอร์รัปต์

เป็นส่วนที่ทำหน้าที่รีเซ็ตสัญญาณอินเตอร์รัปต์ เพื่อทำให้ไม่เกิดการอินเตอร์รัปต์ซ้อนในกรณีที่สัญญาณอินเตอร์รัปต์ที่เข้ามามีความยาวนาน ดังได้กล่าวไปแล้วข้างต้น(หัวข้อ 6.1) วงจรรีเซ็ตสัญญาณอินเตอร์รัปต์แสดงดังในรูปที่ ค.12 การรีเซ็ตสัญญาณทำโดยเขียนข้อมูลอะไรก็ได้ไปยังพอร์ตหมายเลข F2h โดยใช้ประโยค

```
outportb(0XF2 , 0X00) ;
                  └─ ข้อมูลอะไรก็ได้
```

d. ส่วนของการควบคุม Interrupt Controller (8259)

สัญญาณอินเตอร์รัปต์ที่ผ่านเข้ามายัง CPU จะถูกควบคุมโดยชิป 8259 โดย 8259 จะทำการจัดลำดับความสำคัญ (priority) ของสัญญาณอินเตอร์รัปต์ที่เข้ามา ถ้าสัญญาณอินเตอร์รัปต์เข้ามาพร้อม ๆ กัน จะทำการตอบสนองต่ออินเตอร์รัปต์ที่มีลำดับความสำคัญสูงกว่า และเมื่อเสร็จสิ้นการตอบสนองการอินเตอร์รัปต์ ก็จะตอบสนองต่ออินเตอร์รัปต์อื่น ๆ ต่อไป อย่างไรก็ตาม 8259 จะตอบสนองต่อการอินเตอร์รัปต์อื่น ๆ ต่อไปได้ ก็ต่อเมื่อผู้ใช้ส่งคำสั่ง EOI (End of Interrupt) ไปยัง 8259 แล้ว ดังนั้น ในโปรแกรมตอบสนองการอินเตอร์รัปต์จะต้องมีส่วนของการส่งคำสั่งดังกล่าว สำหรับเครื่อง IBM PC ทำได้โดยการส่งข้อมูล 20h ไปยังพอร์ต I/O หมายเลขแอดเดรส 20h ซึ่งเขียนเป็นประโยคภาษา C ได้ดังนี้

```
outportb(0X20 , 0X20) ;
```

e. ส่วนของการคืนค่าให้แก่รีจิสเตอร์ต่าง ๆ

เป็นส่วนที่ทำการคืนค่าจากสแต็กให้แก่รีจิสเตอร์ต่าง ๆ ซึ่งได้ทำการเก็บไว้ในขั้นตอน b. ในภาษาแอสเซมบลีก็จะใช้คำสั่ง POP เช่น

```
POP BX
POP AX
POP DX
```

สำหรับผู้ใช้โปรแกรม TURBO C ผู้ใช้ไม่จำเป็นต้องจัดการกับส่วนนี้เช่นเดียวกับข้อ b.

นอกจากส่วนประกอบดังที่ได้กล่าวมาแล้ว โปรแกรมตอบสนองการอินเตอร์รัปต์ยังต้องสามารถคงอยู่ในหน่วยความจำได้ หลังจากการทำงานจบโปรแกรมแล้วโดยไม่ถูกเขียนทับ ลักษณะโปรแกรมเช่นนี้เรียกว่า โปรแกรม TSR (Terminate and Stay Resident)

สำหรับ TURBO C ได้มีการเตรียมฟังก์ชันชนิดที่เรียกว่า Interrupt Function ซึ่งจะทำหน้าที่ในข้อ b. , e. และการทำให้เป็นโปรแกรม TSR โดยอัตโนมัติ ผู้ใช้เพียงประกาศโปรแกรมในส่วนของการตอบสนองการอินเตอร์รัปต์ให้เป็นฟังก์ชันชนิดดังกล่าว ซึ่งมีรูปแบบการใช้งานดังนี้

```
                    return value type
                        function type = interrupt type
                            function name
void interrupt ISR(parameter list)
{
        :
        function content
        :
}
```

โดยทั่วไปฟังก์ชันชนิด interrupt นี้จะกำหนดให้ไม่มีการส่งค่ากลับ (ชนิด void) เนื่องจากเราไม่สามารถคาดการณ์ได้ว่าจะเกิดการอินเตอร์รัปต์ขึ้นในช่วงจังหวะใดของโปรแกรมการทำงาน การใช้งานฟังก์ชันชนิดนี้จะมีลักษณะเช่นเดียวกับฟังก์ชันทั่ว ๆ ไป แต่มีข้อแตกต่าง คือ

ฟังก์ชันจะทำการเก็บค่า IP (Installation Pointer) , CPU Flag และรีจิสเตอร์ต่าง ๆ ลงสแต็กโดยอัตโนมัติเมื่อเข้าสู่ฟังก์ชัน และเมื่อออกจากฟังก์ชันก็จะทำการ pop ค่าต่าง ๆ กลับคืน ซึ่งเหมือนกับมีคำสั่ง IRET ในภาษาแอสเซมบลีอยู่ในส่วนท้ายของฟังก์ชัน โดยผู้ใช้ไม่จำเป็นต้องเขียนคำสั่งดังกล่าว เราสามารถสรุปขั้นตอนภายในโปรแกรมตอบสนองการอินเตอร์รัปต์ได้ ดังนี้

```
Enable interrupt        ;optional
Save registers
Interrupt procedures
Reset interrupt signal
Send EOI code to 8259
Restore registers
```

## 6.3 โปรแกรมเตรียมการการอินเตอร์รัปต์

เป็นส่วนที่ทำหน้าที่ในการติดตั้งโปรแกรมการตอบสนองการอินเตอร์รัปต์ลงในหน่วยความจำ และทำการแก้ไขตารางอินเตอร์รัปต์เวคเตอร์ให้ชี้ไปยังโปรแกรมตอบสนองการอินเตอร์รัปต์ หลังจากที่โปรแกรมในส่วนนี้ทำงานเสร็จ เครื่องอาจจะไปทำงานในโปรแกรมส่วนอื่นๆ ต่อไป และรอการอินเตอร์รัปต์จากภายนอก ในส่วนโปรแกรมเตรียมการการอินเตอร์รัปต์จะประกอบไปด้วย

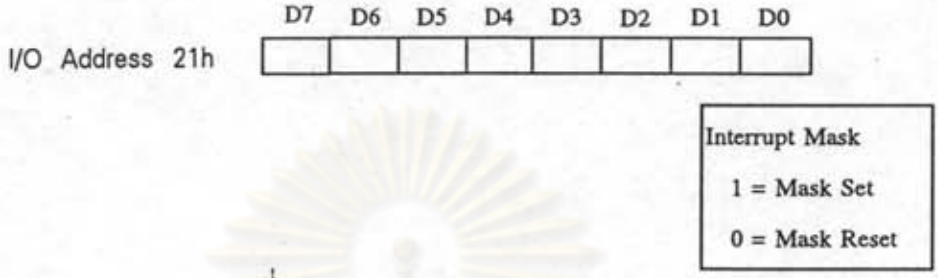
a. ส่วนของการดิสเอเบิลอินเตอร์รัปต์
ส่วนนี้จะใช้ประโยค ดังนี้

```
disable( ) ;
```

เป็นส่วนที่ทำหน้าที่รีเซ็ตอินเตอร์รัปต์แฟล็กบน CPU เพื่อห้ามการอินเตอร์รัปต์ใด ๆ ที่เข้ามาในระหว่างที่ทำการเปลี่ยนแปลงในตาราง Interrupt Vector เพื่อป้องกันความผิดพลาดในการทำงาน

b. ส่วนของการเอนาเบิลการอินเตอร์รัปต์บน Interrupt Controller

ชิป 8259 ประกอบด้วยรีจิสเตอร์ IM (Interrupt Mask Register) ที่ทำหน้าที่ในการดิสเอเบิลการอินเตอร์รัปต์ ดังรูป




รูปที่ ค.13 Interrupt Mask Register

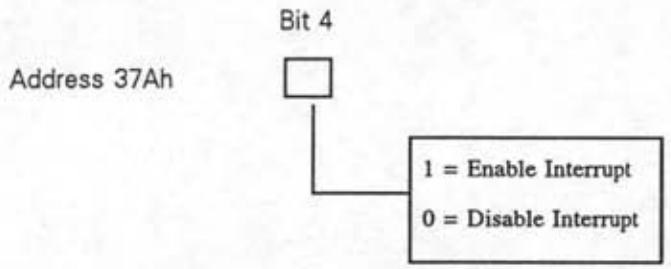
สำหรับเครื่อง IBM PC รีจิสเตอร์ดังกล่าวอยู่ที่แอดเดรส 21h D0 จะตรงกับ IRQ0 D1 จะตรงกับ IRQ1 ฯลฯ ฉะนั้นถ้าเราต้องการเอนาเบิล IRQ0 , IRQ1และ IRQ7 จะทำโดยการส่ง 7ch ไปยัง I/O หมายเลขแอดเดรส 21h ซึ่งเขียนเป็นประโยคได้ ดังนี้

```
outportb(0X21 , 0X7C) ;
```

ในทางปฏิบัติ เนื่องจากระบบไมโครคอมพิวเตอร์ที่เรากำลังใช้งานอยู่อาจจะใช้อินเตอร์รัปต์หมายเลขอื่น ๆ อยู่ด้วย ซึ่งถ้าเราส่งรหัสให้แก่ IM อาจจะเป็นการ mask อินเตอร์รัปต์ที่กำลังใช้งานอยู่ ซึ่งอาจทำให้ระบบทำงานผิดพลาดได้ ดังนั้นเราจึงควรอ่านค่า IM เข้ามาก่อน แล้วใช้เทคนิคการรีเซ็ตบิตโดยใช้ AND เพื่อเอนาเบิลอินเตอร์รัปต์หมายเลขที่ต้องการ ตัวอย่างเช่น ถ้าต้องการเอนาเบิลอินเตอร์รัปต์หมายเลข 7 ทำได้ดังนี้

```
outportb(0X21 , inportb(0X21) & 0X7F) ;
```

สำหรับชุดทดลอง สัญญาณ $\overline{IRQ7}$ จะผ่านเข้ามาทางพอร์ตขนาน ซึ่งบนพอร์ตขนานจะมีบิตที่ทำหน้าที่ เอนาเบิล/ดิสเอเบิล อินเตอร์รัปต์ดังกล่าวอยู่ ดังนั้นนอกจากการเอนาเบิลการอินเตอร์รัปต์บน 8259 แล้ว จะต้องทำการเอนาเบิลบนพอร์ตขนานด้วย บิตดังกล่าวอยู่ที่บิต 4 ของพอร์ตหมายเลขแอดเดรส 37Ah




ซึ่งเราสามารถทำการเอนาเบิลได้ด้วยคำสั่ง

```
outportb(0X37A,inportb(0X37A)|0X10);
```

c. ส่วนของการเปลี่ยนแปลงค่าในตาราง Interrupt Vector

เป็นส่วนที่ทำการแก้ไขค่าในตารางอินเตอร์รัปต์เวคเตอร์ให้ชี้ไปยังฟังก์ชันตอบสนองการอินเตอร์รัปต์ที่สร้างขึ้น ซึ่งการกระทำดังกล่าว จะต้องรู้จักกับพอยน์เตอร์ที่ชี้ไปยังฟังก์ชัน ฟังก์ชันที่ทำหน้าที่ดังกล่าวมีรูปแบบดังต่อไปนี้

```
setvect(intr_num,void interrupt (*isr)( ));
```

หมายเลขของอินเตอร์รัปต์ที่ต้องการแก้ไขค่าในตารางเวคเตอร์

พอยน์เตอร์ที่ชี้ไปยังฟังก์ชันตอบสนองการอินเตอร์รัปต์

int_num คือ หมายเลขของอินเตอร์รัปต์ในตารางอินเตอร์รัปต์เวคเตอร์ ( ประกอบด้วย ฮาร์ดแวร์ และ ซอฟต์แวร์ อินเตอร์รัปต์) ซึ่งการหาหมายเลขอินเตอร์รัปต์ในตารางของฮาร์ดแวร์อินเตอร์รัปต์ ทำได้โดยการบวก 8 เข้ากับหมายเลขของฮาร์ดแวร์อินเตอร์รัปต์ ตัวอย่างเช่น ฮาร์ดแวร์อินเตอร์รัปต์หมายเลข 0 (IRQ0) จะตรงกับอินเตอร์รัปต์หมายเลข 0+8=8 IRQ1 ตรงกับอินเตอร์รัปต์หมายเลข 1+8 = 9 ฯลฯ

สำหรับพอยน์เตอร์ที่ชี้ไปยังฟังก์ชัน มีวิธีการประกาศตัวแปร ดังนี้

```
type ( *fnptr )( parameter-list ) ;
```

เป็นการประกาศให้ fnptr เป็นพอยน์เตอร์ที่ชี้ไปยังฟังก์ชันซึ่งมีชนิดของพารามิเตอร์ดังใน parameter-list type คือ ชนิดของค่าที่ส่งกลับของฟังก์ชันที่ fnptr ชี้อยู่ ตัวอย่างการประกาศ เช่น

```
double ( *funcptr )( double , double )
```

เนื่องจากภาษา C จะรู้จักชื่อของฟังก์ชันในฐานะที่ไม่ใช่ตัวแปร และเราสามารถใช้กำหนดค่าให้แก่พอยน์เตอร์ชนิดชี้ไปยังฟังก์ชันได้โดยตรง เช่น

```
        funcptr = pow ;
ไม่ใช่  funcptr = &pow ;   /* wrong */
```

เป็นการกำหนดให้ funcptr ชี้ไปยังฟังก์ชัน pow ซึ่งอยู่ในไลบรารี math ซึ่งได้นิยามไว้ดังนี้

```
double  pow( double x , double y ) ;
```

การเรียกใช้ฟังก์ชันโดยพอยน์เตอร์ สามารถกระทำได้ดังตัวอย่าง

```
        ( *funcptr )( 14.5 , 0.5 ) ;
ซึ่งสมมูลกับ  pow(14.5 , 0.5)
```

จะสังเกตได้ว่า ชื่อของฟังก์ชัน (pow ในตัวอย่าง) มีลักษณะเป็นพอยน์เตอร์ที่ชี้ไปยังฟังก์ชันอยู่ในตัว (สังเกตได้จาก funcptr = pow) ดังนั้น เราจึงสามารถกำหนดชื่อของฟังก์ชันลงในฟังก์ชัน Setvect ได้โดยตรง เช่น ถ้าต้องการให้อินเตอร์รัปต์หมายเลข 10 (ฐานสิบ) ชี้ไปยังฟังก์ชัน pow สามารถเขียนได้ดังนี้

```
        setvect( 10 , pow ) ;
สมมูลกับ  setvect( 10 , funcptr ) ;
```

อย่างไรก็ตาม ในการแก้ไขค่าในตารางอินเตอร์รัปต์เวคเตอร์ เมื่อเสร็จจากการทำงานเกี่ยวกับอินเตอร์รัปต์ทั้งหมดแล้ว ควรจะทำการแก้ไขค่าในตารางเวคเตอร์กลับไปเป็นอย่างเดิม เพื่อป้องกันระบบการทำงานผิดพลาด เราจึงต้องทำการเก็บค่าเดิมไว้ โดยการอ่านค่าเข้ามาเก็บในตัวแปร ซึ่งทำได้โดยใช้ฟังก์ชัน getvect ซึ่งมีรูปแบบดังนี้

```
getvect( intr_num ) ;
```

ชนิดของค่าที่ส่งกลับ คือ พอยน์เตอร์ที่ชี้ไปยังฟังก์ชันชนิด interrupt ตัวอย่างการใช้งาน เช่น

```
void  interrupt( *oldvect )( ) ;
oldvect  =  getvect( 10 ) ;
```

และเมื่อต้องการเปลี่ยนแปลงค่าในตารางเวคเตอร์ให้กลับเป็นอย่างเดิม สามารถทำได้ดังนี้

```
setvect( 10 , oldvect ) ;
```

d. ส่วนของการเอนาเบิลอินเตอร์รัปต์

หลังจากที่ทำการแก้ไขค่าแอดเดรสในตารางอินเตอร์รัปต์เวคเตอร์เสร็จ โปรแกรมก็พร้อมที่ตอบสนองต่อการอินเตอร์รัปต์ โดยจะต้องทำการเอนาเบิลอินเตอร์รัปต์แฟล็กของ CPU จากที่ดิสเอเบิลไว้ในข้อ a. โดยใช้คำสั่ง

```
enable( ) ;
```

ตัวอย่างโปรแกรมจัดการการอินเตอร์รัปต์แสดงดังรูปที่ ค.14

```
/* ---------- Count number of keyboard sroke using interrupt ----------*/
#include<stdio.h>
#include<dos.h>

void interrupt (*oldvect)( );
void interrupt count( );
int i=0;

void interrupt count( )
{
        enable( );                      /* enable interrupt */
        i++;
        outportb(0x20,0x20);            /* send EOI to 8259 */
        (*oldvect)( );                  /* call old interrupt routine */
}

main( )
{
        int c;
        clrscr( );                      /* clear screen */
        oldvect = getvect(9);           /* save old vector */
        setvect(9,count);               /* store new interrupt no.9(keyboard int) vector*/
        while(c=getche( ),c!=27){
                gotoxy(10,10);
                printf("Number of keyboard stroke = %u \n",i/2-1);
        }
        setvect(9,oldvect);             /* restore old vector */
}
```

รูปที่ ค.14 ตัวอย่างโปรแกรมจัดการการอินเตอร์รัปต์

## 7. เทคนิคอื่น ๆ

### 7.1 การใช้บิตฟิลด์ในการดำเนินการระดับบิต

ในงานควบคุม เรามักจะต้องเผชิญกับการดำเนินการในระดับบิต ยกตัวอย่างเช่น การเซ็ต/รีเซ็ตบิต การ toggle บิต หรือการทดสอบบิต เป็นต้น ซึ่งเรามักจะใช้ตัวดำเนินการ เช่น AND , OR , XOR เข้าช่วยโดยการดำเนินการจะต้องทำทั้งไบต์ ไม่สามารถเข้าไปแก้ทีละบิตได้โดยตรง ยกตัวอย่างเช่น ถ้าเรามีตัวแปร X ที่ต้องการจะเซ็ตบิต 7 เราจะทำโดยใช้ตัวดำเนินการ OR ดังนี้

```
X = X | 0X80 ;  /* X OR 80h , set bit 7 of X */
```

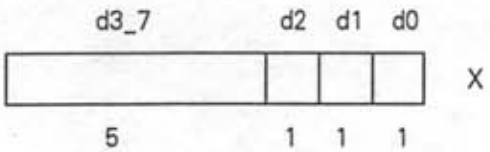
หรือถ้าต้องการ toggle บิต 3 ของ X ก็ทำได้ดังนี้

```
X = X^0X08 ;  /* X XOR 08h , toggle bit 3 of X */
```

อย่างไรก็ตาม ในภาษา C เราสามารถประยุกต์บิตฟิลด์มาใช้ในการดำเนินการระดับบิตได้ โดยจะทำให้สามารถเข้าไปแก้ไขข้อมูลทีละบิตได้โดยตรง ซึ่งทำให้เกิดความสะดวกขึ้นมาก รูปแบบการใช้บิตฟิลด์แสดงดังตัวอย่างต่อไปนี้

```
struct newtype {
        unsigned d0 : 1;
        unsigned d1 : 1;
        unsigned d2 : 1;
        unsigned d3_7 : 5;
} x ;
```

เป็นการกำหนดตัวแปร X ที่ประกอบด้วย 4 ฟิลด์ คือ d0 มีขนาด 1 บิต d1 มีขนาด 1 บิต d2 มีขนาด 1 บิต d3_7 มีขนาด 5 บิต โดยบิต d0 จะอยู่ทางขวามือสุด ดังแสดง




การกำหนดค่าให้แต่ละฟิลด์ทำโดยระบุชื่อตัวแปร ตามด้วยจุด แล้วตามด้วยชื่อฟิลด์ ตัวอย่างเช่น

```
x.d0 = 1;
x.d2 = 0;
x.d3_7 = 16;
```

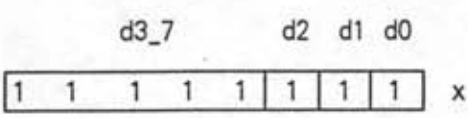
ในบางกรณีเราอาจจำเป็นต้องกำหนดค่า X พร้อมกันทั้ง 8 บิต ตัวอย่างเช่น กรณีอ่านข้อมูลจากพอร์ตในภาษา C เราไม่สามารถกำหนดค่า X ได้โดยตรง ตัวอย่างเช่น

```
x = inportb(0XC1);      /* wrong */
x = 0XFF;               /* wrong */
```

ในกรณีนี้ เราสามารถแก้ไขได้โดยการใช้พอยน์เตอร์เข้าช่วย โดยกำหนดให้มีพอยน์เตอร์ที่ชี้ไปยังตัวแปร X และเมื่อต้องการเปลี่ยนแปลงค่า X ก็ทำโดยการเปลี่ยนแปลงค่าที่พอยน์เตอร์ชี้อยู่ เช่น

```
char *px;               /* declare pointer */
px = &x;                /* point to x */
*px = 0XFF;             /* assign FFh to x */
```

ซึ่งในกรณีนี้เมื่อเราตรวจดูค่าใน X จะได้ดังนี้




หรือถ้าต้องการอ่านค่าจากพอร์ต I/O เข้ามาเก็บในตัวแปร X ก็ทำได้โดย

```
*px = inportb(0XC1);
```

หลังจากนั้นเราก็สามารถดำเนินการระดับบิตกับ X ได้ เช่น

```
x.d2 = 0;              /* reset bit 2 of x */
```

เป็นการรีเซ็ตบิต 2 ของ X

```
x.d1 = ~x.d1;          /* toggle bit 1 of x */
```

เป็นการ toggle บิต 1 ของ X

นอกจากการใช้พอยน์เตอร์ดังกล่าวแล้ว เรายังสามารถใช้ union มาแก้ไขปัญหาดังกล่าวได้ โดยการประกาศตัวแปร X ขึ้นใหม่ดังนี้

```
union  type1  {
            char  all;
            struct  newtype  {
                        unsigned d0 : 1;
                        unsigned d1 : 1;
                        unsigned d2 : 1;
                        unsigned d3_7 : 5;
                    } y;
    } x;
```

การดำเนินการระดับบิตก็สามารถทำได้ดังตัวอย่าง

```
x.y.d0 = 1;
x.y.d1 = 0;
```

การกำหนดค่าให้แก่ X พร้อมกันทุกบิต ทำได้ดังตัวอย่าง

```
x.all = 0XFF;
```

## 7.2 การสร้าง Look-up table

Look-up table มักถูกใช้ในการแปลงค่าหรือรหัสจากรหัสหนึ่งไปเป็นรหัสหนึ่ง เช่น แปลงรหัสจาก binary ไปเป็น BCD จาก binary ไปเป็น gray code เป็นต้น หรือใช้ในการหาค่าจากสมการทางคณิตศาสตร์หาค่าของฟังก์ชันตรีโกณมิติ เช่น การแปลงค่าอุณหภูมิจากองศาเซลเซียสไปเป็นฟาเรนไฮต์ ซึ่งเราสามารถใช้ Look-up table แทนสมการการแปลง C/5 = (F-32)/ 9 หรือการหาค่าของฟังก์ชัน sin , cos , tan , asin เป็นต้น การใช้ Look-up table มีข้อดีตรงที่สามารถทำงานได้เร็ว ช่วยลดภาระแก่เครื่องคอมพิวเตอร์ในการคำนวณค่าจากสมการที่ซับซ้อน หรือการปฏิบัติขั้นตอนที่ยุ่งยากทำให้การเขียนโปรแกรมทำได้ง่าย ไม่ยุ่งยากซับซ้อน แต่ก็มีข้อจำกัดตรงที่ ตัวแปรขาเข้าจะต้องเป็นเซ็ตจำกัด หรือไม่ก็ใช้การประมาณค่าเข้าช่วย ซึ่งทำให้ได้ความละเอียดลดลง และ การสิ้นเปลืองหน่วยความจำในการเก็บข้อมูลในตาราง Look-up table จะต้องครอบคลุมค่าอินพุตที่เป็นไปได้ทั้งหมด หรือจะต้องทราบว่าจะจัดการกับอินพุตแต่ละค่าอย่างไร เพื่อหลีกเลี่ยงความผิดพลาดในการทำงาน หรือเกิดผลที่คาดเดาไม่ได้ในกรณีที่ค่าอินพุตเป็นค่าที่ไม่ได้กำหนดไว้ในตาราง

ในภาษา C เราจะใช้ตัวแปรชนิดอาร์เรย์ในการเก็บข้อมูลในตาราง ตัวอย่างเช่น ถ้าเรามีตารางดังตารางที่ 1 เราสามารถเก็บค่าลงในตัวแปรอาร์เรย์ได้ดังนี้

```
char gray[16] = { 0 , 1 , 3 , 2 ,
                  6 , 7 , 5 , 4 ,
                 12 ,13 ,15 ,14 ,
                 10 ,11 , 9 , 8 };
```

ตารางที่ 1 ตารางการเปลี่ยนรหัส binary เป็น gray code

| Binary number | | Gray code | |
|---|---|---|---|
| Code | Decimal | Code | Decimal |
| 0000 | 0 | 0000 | 0 |
| 0001 | 1 | 0001 | 1 |
| 0010 | 2 | 0011 | 3 |
| 0011 | 3 | 0010 | 2 |
| 0100 | 4 | 0110 | 6 |
| 0101 | 5 | 0111 | 7 |

| Binary number | | Gray code | |
|---|---|---|---|
| Code | Decimal | Code | Decimal |
| 0110 | 6 | 0101 | 5 |
| 0111 | 7 | 0100 | 4 |
| 1000 | 8 | 1100 | 12 |
| 1001 | 9 | 1101 | 13 |
| 1010 | 10 | 1111 | 15 |
| 1011 | 11 | 1110 | 14 |
| 1100 | 12 | 1010 | 10 |
| 1101 | 13 | 1011 | 11 |
| 1110 | 14 | 1001 | 9 |
| 1111 | 15 | 1000 | 8 |

ในการหาค่าจากตาราง เราจะใช้อินพุตเป็นตัวชี้ค่าในตาราง ดังตัวอย่าง

```
binary = 10;
printf("%d ",gray[binary]);
```

จากตัวอย่างเป็นการหาค่า gray code ของ binary = 10 (ฐานสิบ) ซึ่งจะให้ผลลัพธ์เป็น 15 (ฐานสิบ) เราสามารถประยุกต์ใช้อาร์เรย์หลายมิติในการสร้าง Look-up table ที่มีตัวแปรอินพุตหลาย ๆ ตัว ดังตัวอย่างต่อไปนี้

Z =

| x \ y | 0 | 1 | 5 | 2 |
|---|---|---|---|---|
| 0 | 0 | 1 | 5 | 2 |
| 1 | 3 | 0 | 4 | 0 |

```
char z[2][4] = {{0,1,5,2},{3,0,4,0}};
char x,y;
x = 1;
y = 2;
```

```
printf("z = %d ",z[x][y]);
```

ตัวอย่างข้างต้นเป็นตัวอย่างการหาค่า Z จากตาราง เมื่อ x = 1 , y = 2 ซึ่งจะให้ผลลัพธ์ค่า z เป็น 4

## 7.3 การเขียนโปรแกรมภาษา C ที่เรียกมอดูลภาษาแอสเซมบลี

ปัจจุบันเรามักนิยมใช้ภาษาระดับสูงในการพัฒนาโปรแกรมต่าง ๆ แทนภาษาระดับต่ำ เช่น การเขียนคอมไพเลอร์ การเขียนโปรแกรมประยุกต์ต่าง ๆ หรือแม้แต่โปรแกรมที่จัดการในระดับฮาร์ดแวร์ จะสังเกตได้ว่าไมโครคอนโทรลเลอร์บางตัวถูกออกแบบมาให้ใช้ภาษาระดับสูง ทั้งนี้เนื่องจากการเขียนโปรแกรมด้วยภาษาระดับสูงมีความซับซ้อนน้อยกว่ามาก และยังช่วยลดเวลาและค่าใช้จ่ายในการพัฒนาโปรแกรมลง อีกทั้งปัจจุบันเครื่องคอมพิวเตอร์ถูกพัฒนาให้ทำงานเร็วขึ้นมาก มีราคาไม่แพง ดังนั้นแทนที่จะเสียค่าใช้จ่ายไปในการพัฒนาโปรแกรมที่ทำงานด้วยความเร็วสูงโดยใช้ภาษาระดับต่ำ การหันมาใช้ภาษาระดับสูงโดยใช้เครื่องคอมพิวเตอร์ที่ทำงานได้เร็วขึ้นอาจจะคุ้มค่ากว่าและการช่วยลดเวลาในการพัฒนาโปรแกรม ทำให้เครื่องต้นแบบถูกผลิตออกมาได้อย่างรวดเร็ว อย่างไรก็ตามในประเด็นนี้เป็นสิ่งที่จะต้องพิจารณาหาจุดเหมาะสมสำหรับแต่ละงานไป หลักโดยทั่วไปในการพัฒนาโปรแกรมสำหรับงานควบคุม คือ พยายามเขียนโปรแกรมด้วยภาษาระดับสูงให้มากที่สุด เว้นแต่ในส่วนที่ต้องการความเร็วสูงเป็นจุดวิกฤตทางด้านเวลา หรือไม่สามารถเขียนด้วยภาษาระดับสูงได้ จึงใช้ภาษาระดับต่ำ ฉะนั้นการเรียนรู้วิธีการเชื่อมต่อโปรแกรมภาษาระดับสูงเข้ากับโปรแกรมภาษาระดับต่ำจึงเป็นสิ่งสำคัญ ในที่นี้เราจะได้เรียนรู้การเขียนโปรแกรมภาษา C ที่เรียกมอดูลภาษาแอสเซมบลีอย่างง่าย ๆ และทดลองนำมาประยุกต์ใช้งาน

### วิธีการ

ขั้นตอนง่าย ๆ ในการที่จะเข้าใจวิธีการอย่างรวดเร็ว กระทำได้โดยการนำโปรแกรมที่เขียนขึ้นด้วยภาษา C ไปคอมไพล์ให้เป็นภาษาแอสเซมบลี ซึ่งสามารถกระทำโดยใช้โปรแกรม TCC ตัวเลือก -S แสดงดังรูปที่ ค.15 - ค.16

```
/* C PROGRAM */
#include<stdio.h>
int a = 1;              /* global initialized variable */
int b;                  /* global uninitialized varible */
int shl(int c);         /* declare function shl which return an int value */
main( )
{
        b = shl(a);
}
int shl(int c)          /* shl function definition */
{
        int d;          /* local variable */
        d = c<<1;
        return(d);
}
```

รูปที่ ค.15 โปรแกรมภาษา C

```
ifndef  ??version
?debug  macro
        endm
        endif
        ?debug  S "temp.c"
_TEXT   segment byte public 'CODE'
DGROUP  group   _DATA,_BSS
        assume  cs:_TEXT,ds:DGROUP,ss:DGROUP
_TEXT   ends
_DATA   segment word public 'DATA'
d@      label   byte
d@w     label   word
_DATA   ends
_BSS    segment word public 'BSS'
```

```
b@          label       byte
b@w         label       word
            ?debug      C E95965BE200674656D702E63
_BSS        ends
_DATA       segment word public 'DATA'
_a          label       word
            dw          1
_DATA       ends
_TEXT       segment     byte public 'CODE'
;           ?debug      L 8
_main       proc        near
;           ?debug      L 10
            push        word ptr DGROUP:_a
            call        near ptr _shl
            pop         cx
            mov         word ptr DGROUP:_b,ax
@1:
;           ?debug      L 11
            ret
_main       endp
;           ?debug      L 12
_shl        proc        near
            push        bp
            mov         bp,sp
            push        si
;           ?debug      L 15
            mov         si,word ptr [bp+4]
            shl         si,1
;           ?debug      L 16
            mov         ax,si
            jmp         short @2
@2:
;           ?debug      L 17
            pop         si
            pop         bp
```

```
            ret
_shl        endp
_TEXT       ends
_BSS        segment word public 'BSS'
_b          label       word
            db          2 dup (?)
_BSS        ends
            ?debug      C E9
_DATA       segment word public 'DATA'
s@          label       byte
_DATA       ends
_TEXT       segment     byte public 'CODE'
_TEXT       ends
            public      _main
            public      _shl
            public      _b
            public      _a
            end
```

□

รูปที่ ค.16 โปรแกรมแอสเซมบลีที่ได้จากการคอมไพล์โปรแกรมภาษา C ในรูปที่ ค.15

จากโปรแกรมในรูปที่ ค.16 ถ้าเรานำมาเขียนใหม่โดยให้เรียกฟังก์ชัน shl ที่เขียนขึ้นเป็นมอดูลในภาษาแอสเซมบลี จะสามารถเขียนโปรแกรมได้ดังนี้

```
/* C PROGRAM CALLING ASSEMBLY MODULE */
int a = 1;
int b;
int extern shl(int c);
main( )
{
        b = shl(a);
}
```

รูปที่ ค.17(a) ส่วนของโปรแกรมภาษา C

```
_TEXT     segment    byte public 'CODE'
DGROUP  group   _DATA
          assume     cs:_TEXT,ds:DGROUP,ss:DGROUP
_TEXT     ends

_DATA   segment word public 'DATA'
_a        label      word
          dw         1
_DATA     ends

_TEXT   segment byte public 'CODE'
_shl      proc       near
          push       bp
          mov        bp,sp
          push       si
          mov        si,word ptr [bp+4]
          shl        si,1
          mov        ax,si
          jmp        short @2
@2:
          pop        si
          pop        bp
          ret
_shl      endp
_TEXT     ends
          public     _shl
          end
```

□

รูปที่ ค.17(b) มอดูลภาษาแอสเซมบลี

เราสามารถสรุปวิธีการได้ดังนี้

1. โปรแกรมภาษา C ถูกเขียนขึ้นตามปกติ แต่ในส่วนของฟังก์ชันที่เป็นมอดูลภาษาแอสเซมบลี ให้ประกาศฟังก์ชันโดยมีคำว่า extern เพื่อระบุว่าฟังก์ชันดังกล่าวเป็นมอดูลภายนอก ดังตัวอย่าง

```
int extern shl(int c);
```

2. มอดูลภาษาแอสเซมบลี

2.1 โครงสร้างของมอดูลจะเป็นดังนี้

```
_TEXT     SEGMENT BYTE PUBLIC 'CODE'
DGROUP    GROUP   _DATA , _BSS
          ASSUME  CS:_TEXT , DS:DGROUP , SS:DGROUP
_TEXT     ENDS
_DATA     SEGMENT WORD PUBLIC 'DATA'
          :
<---------- INITIALIZED DATA SEGMENT ---------->
          :
_DATA     ENDS
_TEXT     SEGMENT BYTE PUBLIC 'CODE'
          :
<---------- CODE SEGMENT ---------->
          :
_TEXT     ENDS
_BSS      SEGMENT WORD PUBLIC 'BSS'
<---------- UNINITIALIZED DATA SEGMENT ---->
_BSS      ENDS
          END
```

ส่วนของ _TEXT Segment เป็นส่วนที่ทำหน้าที่เก็บคำสั่งการดำเนินการของมอดูล (program code) ส่วนของ _DATA Segment เป็นส่วนที่เก็บข้อมูลที่ต้องการกำหนดค่าเริ่มต้น _BSS Segment เป็นส่วนที่เก็ยข้อมูลที่ไม่มีการกำหนดค่าเริ่มต้นเจาะจง (สังเกตการใช้เครื่องหมาย ?) ส่วน _BSS Segment เป็นส่วนที่สามารถละได้หากไม่ต้องการใช้งาน สังเกตได้ว่าตัวแปร a ถูกเก็บไว้ในส่วน _DATA และตัวแปร b ถูกเก็บไว้ในส่วน _BSS

2.2 ชื่อของตัวระบุชื่อ (ตัวแปร หรือ ฟังก์ชัน) ภายนอก (ในโปรแกรมภาษา C) ที่อ้างถึง จะต้องขึ้นต้นด้วยเครื่องหมาย underscore ( _ ) ตัวอย่างเช่น

_printf , _a

ตัวแปรภายนอกที่อ้างอิงถึงจะต้องถูกประกาศในส่วนของ _DATA Segment หรือ _BSS Segment ด้วย

2.3 ชื่อของตัวระบุชื่อในภาษาแอสเซมบลีที่ถูกอ้างโดยภาษา C จะต้องขึ้นต้นด้วยเครื่องหมาย underscore เช่นกัน เช่น ชื่อของฟังก์ชัน shl จะเขียนเป็น

_shl

และชื่อเหล่านี้จะต้องถูกประกาศให้เป็น public เพื่อให้ภายนอกสามารถมองเห็นได้ โดยถ้าเป็นชื่อของฟังก์ชัน จะถูกประกาศไว้ใน code segment แต่ถ้าเป็นชื่อของตัวแปร จะประกาศไว้ใน data segment ตัวอย่างเช่น

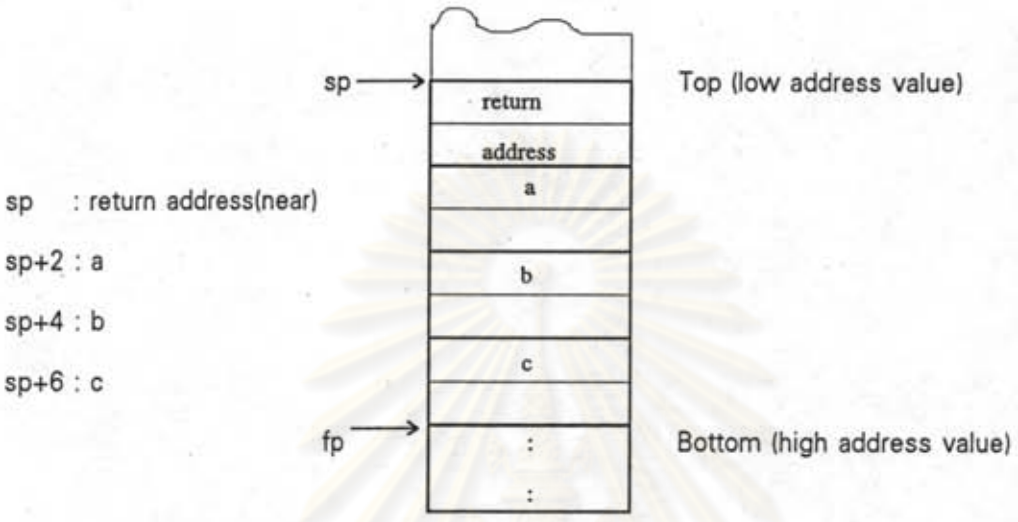
```
public _shl          ; function name declared in code segment
public _var          ; variable name declared in data segment
```

3. การส่งผ่านค่าพารามิเตอร์ไปยังฟังก์ชัน

ภาษา C ส่งผ่านค่าไปยังฟังก์ชันโดยใช้สแต็ก โดยการคัดลอกค่าพารามิเตอร์ลงสแต็กแทนการคัดลอกค่าพอยน์เตอร์ของพารามิเตอร์ ภาษา C จึงเป็นภาษาที่ส่งผ่านค่าไปยังฟังก์ชันแบบ passed by value ในกรณีที่มีค่าพารามิเตอร์มากกว่า 1 ตัว จะทำการเก็บค่าลงสแต็กจากขวาไปซ้าย ตัวอย่างเช่น

```
int a,b,c;
func(a , b , c);
```

ค่า c จะถูกเก็บลงสแต็กก่อนแล้วตามด้วย b และ a ตามลำดับ สแต็กจะมีลักษณะดังนี้




รูปที่ ค.18 การส่งผ่านค่าพารามิเตอร์ผ่านสแต็กในภาษา C

fp เป็นตำแหน่งที่ stack pointer ชี้อยู่ก่อนเรียกฟังก์ชัน sp เป็นตำแหน่งที่ stack pointer ชี้เมื่อทำการเรียกฟังก์ชัน จะสังเกตว่ามีการเก็บค่า return address ลงสแต็กโดยเก็บเป็นค่าสุดท้าย return address นี้อาจมีขนาด 2 ไบต์ หรือ 4 ไบต์ ขึ้นกับชนิดของฟังก์ชันที่ถูกเรียกว่าเป็นชนิด near หรือ far การนำค่าพารามิเตอร์ออกมาใช้จะต้องไม่มีผลกระทบต่อสแต็กก่อนจะกลับไปยังรูทีนเดิม เราจึงมักใช้ base pointer(bp) แทน stack pointer(sp) โดยการคัดลอกค่า sp ลง bp อย่างไรก็ตามค่า bp จะต้องไม่เปลี่ยนแปลงด้วย จึงทำการเก็บค่า bp ลงสแต็กก่อน ดังตัวอย่างต่อไปนี้แสดงการนำค่าพารามิเตอร์ a ออกมาใช้งาน จะสังเกตว่าจะใช้ค่าที่ตำแหน่ง bp+4 สำหรับค่า a แทนที่จะเป็น bp+2 เนื่องจากได้เก็บ bp ลงสแต็กไว้ด้วย

```
push  bp                    ; save bp
mov  bp , sp                ; move sp into bp
mov  ax , word ptr [bp+4]   ; move a into ax
   :
pop  bp                     ; restore bp
ret                         ; return to calling routine
```

4. การส่งค่ากลับ

ภาษา C ส่งผ่านค่าพารามิเตอร์ไปยังฟังก์ชันโดยใช้สแต็ก แต่ในการส่งค่ากลับจะใช้รีจิสเตอร์ โดยถ้าค่าที่ส่งกลับมีขนาด 16 บิต ( char , int , enum , near pointer ) จะใช้รีจิสเตอร์ AX แต่ถ้ามีขนาด 32 บิต ( long int , float , far , huge , pointer ) จะใช้รีจิสเตอร์ DX ร่วมด้วย โดย AX เก็บ low order และ DX เก็บ high order ฉะนั้นในมอดุลฟังก์ชันจะทำการคัดลอกค่าที่ส่งกลับ (คำตอบ) ลงในรีจิสเตอร์ดังกล่าว ดังตัวอย่าง

```
mov  ax , si          ; copy return value to ax register
pop  si
pop  sp
ret                   ; return to calling routine
```

5. คอมไพล์โปรแกรมภาษา C ให้เป็น object code ซึ่งทำโดยใช้คำสั่ง compile ที่อยู่บนเมนูของ เอดิเตอร์ หรือใช้โปรแกรม TCC ( -c ) จะได้ไฟล์นามสกุล .obj และ คอมไพล์มอดูลแอสเซมบลีให้เป็น object code โดยใช้โปรแกรม Masm (Macro Assembler) หรือ Tasm (Turbo Assembler)

6. ทำการลิงค์โปรแกรมภาษา C กับมอดูลภาษาแอสเซมบลีเข้าด้วยกันโดยใช้โปรแกรม TLINK ซึ่งมีรูปแบบการใช้ดังนี้

```
        tlink  object1  object2  ... , output name
เช่น    tlink  c.obj  mod.obj  , c.exe
```

ผลลัพธ์ที่ได้จะเป็น execute file ในกรณีที่มอดูลที่จะทำการลิงค์เข้าด้วยกันมีนามสกุลเป็น .OBJ อยู่แล้ว ก็สามารถละนามสกุลได้ และในกรณีต้องการ output file เป็น .EXE ก็ไม่จำเป็นจะต้องระบุเช่นกันดังตัวอย่าง

```
tlink  c  mod  ,  c
```

ภาคผนวก ง

# คู่มือการใช้งานชุดฝึกทดลอง

# คู่มือการใช้งานชุดฝึกทดลอง

## 1. ข้อกำหนดรายละเอียดทางด้านฮาร์ดแวร์

### 1.1 เมนบอร์ด

- บัสสัญญาณการเขียน/อ่านข้อมูล แบ่งเป็น ดาต้าบัสขนาด 8 บิต แอดเดรสบัส ขนาด 8 บิต สัญญาณ $\overline{RD}$ (สัญญาณการอ่านข้อมูล) สัญญาณ $\overline{WR}$ (สัญญาณการอ่านข้อมูล) กราวด์
- สัญญาณขาเข้า อินเตอร์รัปต์ ระดับ 7
- ใช้ภาษา C ในการควบคุม
- ไม่ต้องใช้ไฟเลี้ยงภายนอก(เลือกได้)
- ความเร็วของพอร์ตสัญญาณเข้า 28 kHz
- ความเร็วของพอร์ตสัญญาณออก 30 kHz

(ทดสอบกับเครื่อง PC 386-40MHz)

- ขนาดของบอร์ด 8ซม.x12ซม.

### 1.2 มอดูล

มอดูล A ถึง D และ I แต่ละมอดูลมีขนาด 8ซม.x 12ซม.

A. มอดูลตัวแปลงสัญญาณ แอนะล็อก เป็นดิจิทัล และดิจิทัลเป็นแอนะล็อก (มอดูล ADAC)

1. ตัวแปลงสัญญาณ แอนะล็อกเป็นดิจิทัล(A/D)
   - ความละเอียดขนาด 8 บิต
   - เวลาในการแปลงสัญญาณ 100 µS
   - วงจรขยายสัญญาณขาเข้าที่สามารถปรับอัตราขยาย และแรงดันเบี่ยงเบน
   - สัญญาณแอนะล็อกขาเข้าไมโครโฟน ตัววัดอุณหภูมิ โฟโต้ทรานซิสเตอร์ โพเทนชิโอมิเตอร์
   - ควบคุมด้วย IBM PC มีสัญญาณเพื่ออินเตอร์รัปต์ IBM PC
   - ใช้ไฟ +5V, +12V, -12V
2. ส่วนของตัวแปลงสัญญาณดิจิทัลเป็น แอนะล็อก

- ความละเอียดขนาด 8 บิต settling time 150 ns

- วงจรขยายสัญญาณขาออก ที่สามารถปรับอัตราขยาย และแรงดันเบี่ยงเบน แรงดันขาออกอยู่ในช่วง -10 V ถึง +10 V

- ใช้ไฟ +5V, +12V, -12V

B. มอดูลตัวจับเวลา/ตัวนับ (มอดูล T/C)

- ใช้ไอซีเบอร์ 8253

- วงจรกำเนิดสัญญาณนาฬิกาขนาด 1.19 MHz

- ใช้ไฟ +5V

C. มอดูลพอร์ตสัญญาณเข้าออก (มอดูลพอร์ตI/O)

- พอร์ตสัญญาณขาเข้า 2 พอร์ต

- พอร์ตสัญญาณขาออก 2 พอร์ตมีแลตช์ฟังก์ชัน

- ใช้ไฟ +5V

D. มอดูลสวิตช์

- รีเลย์ 4 ตัว

- ตัวขับทรานซิสเตอร์ 4 ตัว

- ตัวแสดงผลแบบแอลอีดี จำนวน 8 ดวง

- สวิตช์ป้อนสัญญาณขาเข้า แบบกด 4 ตัว แบบโยก 4 ตัว

- ใช้ไฟ +5V,+12V

E. มอดูลสื่อสารข้อมูลอนุกรม

- ตัวแสดงผลแบบแอลอีดี 8 ดวงแสดงสถานะของสัญญาณ Tx, Rx, RTS, CTS, DTR, DSR, DCD, RI

F. มอดูลสื่อสารข้อมูลแบบขนาน

- ตัวแสดงผลแบบแอลอีดี 10 ดวงแสดงสถานะของสัญญาณ

G. มอดูลสเตปปิงมอเตอร์

- ความละเอียด 1.8°/สเตป จำนวน 200 สเตป

- ใช้ไฟ +12V

- ขนาด 4 เฟส

H. มอดูลควบคุมมอเตอร์

- สัญญาณพัลส์วัดความเร็ว

- ใช้ไฟ +5V

I. มอดูลแสดงผลแอลอีดีแบบเมตริกซ์ (มอดูล Display)

- ขนาดการแสดงผล 8x12 จุด
- สัญญาณข้อมูลขนาด 8 บิต และ สัญญาณมัลติเพลกซ์ ขนาด 4 บิต
- ใช้ไฟ +5V

J. มอดูลวงจรขยายเสียง

- ขนาด 0.5W
- ใช้ไฟ +5V
- สามารถปรับขนาดสัญญาณเข้าได้

K. โปรโตบอร์ด

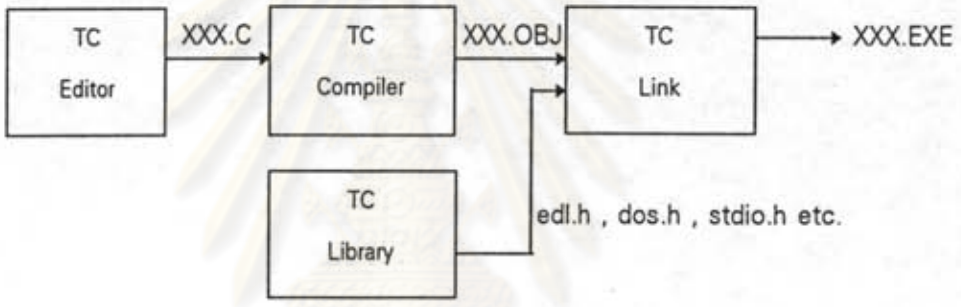
- ขนาด 800 ขา พร้อมแถบไฟ 2 แถบ
- ขั้วไฟเลี้ยงจำนวน 5 ขั้ว

1.3 <u>อื่นๆ</u>

- สายต่อต่างๆ
  - สายพรินเตอร์
  - สายจ่ายไฟ
  - สายคอนเนกเตอร์แบบแบน
  - สายปากคีบ
- คู่มือทฤษฎีการทดลองและใบงานการทดลอง
- คู่มือการเขียนโปรแกรมภาษา C
- คู่มือการเขียนโปรแกรมภาษา C สำหรับงานควบคุม
- แผ่นดิสค์โปรแกรมติดตั้ง และโปรแกรมเฉลย

2. ข้อกำหนดรายละเอียดทางด้านซอฟต์แวร์

การเขียนโปรแกรมควบคุมภาษา C ใช้คอมไพเลอร์ Turbo C ของบริษัท Borland ซึ่งทำงานบน DOS(Disk Operating System)ตั้งแต่เวอร์ชัน 2.0 เป็นต้นไป และต้องการหน่วยความจำอย่างน้อย 384K สามารถใช้งานได้โดยใช้ดิสค์ไดร์ฟเพียงไดร์ฟเดียว แต่ในที่นี้ขอแนะนำให้ใช้ฮาร์ดดิสค์ การควบคุมชุดฝึกทดลองจะใช้ฟังก์ชันที่เขียนขึ้นสำหรับชุดทดลองเอง คือ ฟังก์ชัน IN, OUT ซึ่งไม่มีใน Library ของ Turbo C จะต้องทำการติดตั้ง รูปที่ ง.1 แสดงขั้นตอนการเขียนโปรแกรม ซึ่งขั้นตอนต่างๆ Turbo C สามารถทำได้โดยอัตโนมัติด้วย IDE(Integrated Development Environment)




รูปที่ ง.1 ขั้นตอนการเขียนโปรแกรม

3. ลักษณะการใช้งาน

A. ต่อเข้ากับเครื่อง IBM PC หรือ เครื่องพีซีทั่วไป
B. ต่อเข้ากับเครื่อง IBM PC ผ่านทางพอร์ตขนาน
C. ควบคุมการทำงานด้วยภาษา C สามารถดัดแปลงให้ใช้ภาษาอื่นๆ ได้
D. ประกอบด้วยการทดลองรวม 12 การทดลอง
E. ควบคุมพอร์ตสัญญาณเข้า-ออกโดยการเรียกใช้งานฟังก์ชันที่ให้มากับชุดฝึกทดลองแทนคำสั่งปกติ

4. การทดลอง

การทดลอง ประกอบด้วย

A. Familiarization with the Training Set

B. การเขียนโปรแกรมด้วยภาษา C

C. Digital I/O

D. D/A Converter

E. A/D Converter

F. Timer/Counter

G. อินเตอร์รัปต์

H. Data Communication

I. Applications 1: Stepping Motor Control

J. Applications 2: Display Control

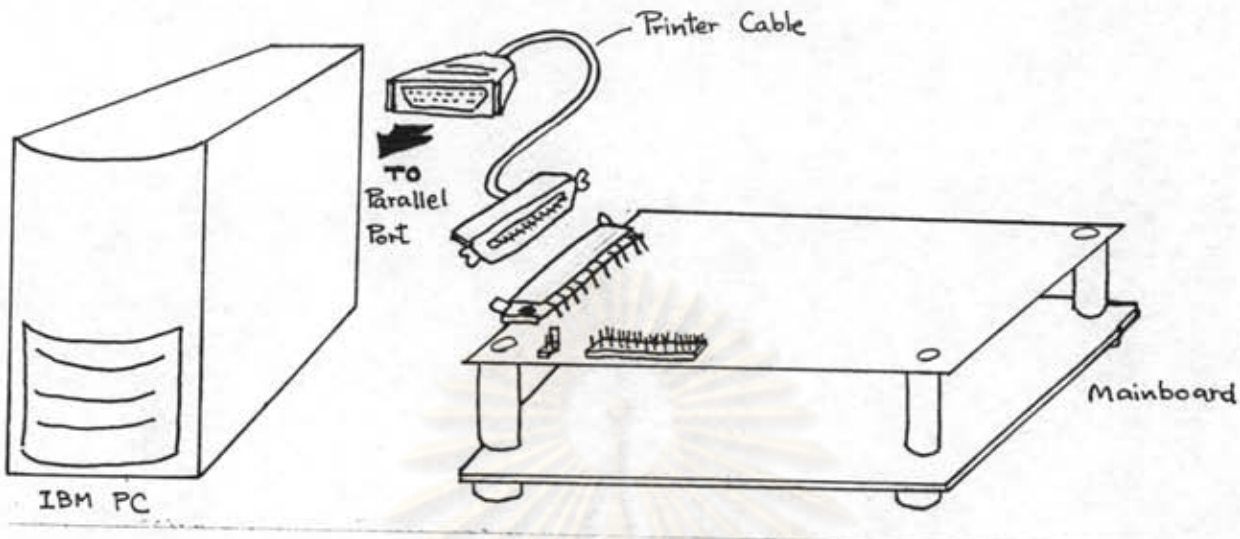
K. Digital Control System

L. Project

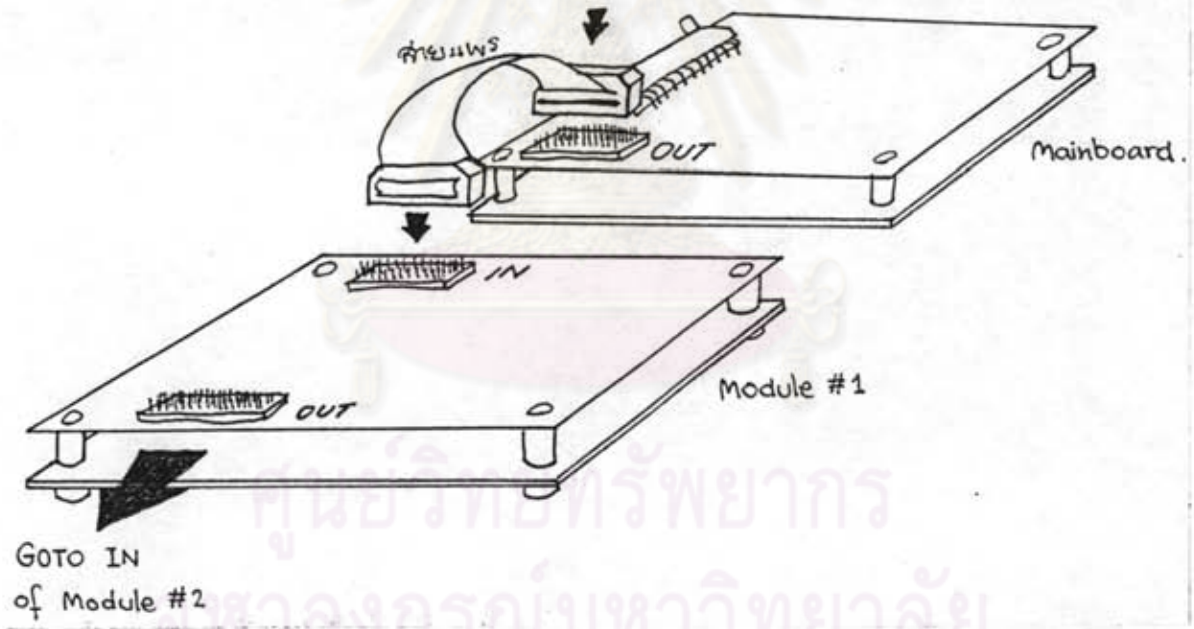
## 5. การติดตั้ง

### 5.1 ฮาร์ดแวร์

เมนบอร์ดต่อเข้ากับ IBM PC ทางพอร์ตขนาน โดยใช้สายพรินเตอร์ที่มีลักษณะเหมือนสายพรินเตอร์ที่ใช้งานอยู่โดยทั่วไป ดังรูปที่ ง.2 การต่อมอดูลเข้ากับเมนบอร์ดใช้สายแพรที่ต่อกับคอนเนกเตอร์ขนาด 20 ขา ขั้ว OUT ของเมนบอร์ดจะต่อเข้ากับขั้ว IN ของมอดูล ในกรณีที่ต้องการต่อมอดูลอื่นๆเพิ่มเติมเข้าไป ให้ใช้วิธีต่อแบบ cascade โดยขั้ว OUT ของมอดูลที่ 1 จะต่อเข้ากับขั้ว IN ของมอดูลที่ 2 และต่อในลักษณะเช่นนี้ไปเรื่อยๆ




รูปที่ ง.2 แสดงการติดตั้งชุดทดลอง




รูปที่ ง.3 แสดงการต่อมอดูลเข้ากับเมนบอร์ด

5.2 **ซอฟต์แวร์**

ผู้ใช้ควรจะมีฮาร์ดดิสค์ และโปรแกรมระบบปฏิบัติการ DOS (Disk Operation System) การติดตั้งซอฟต์แวร์เริ่มต้นโดยการเข้าสู่ระบบปฏิบัติการ

C>

ใส่แผ่นดิสค์ลงในช่องไดรฟ์ A และเปลี่ยนการทำงานไปที่ช่องไดรฟ์ เช่น ไดรฟ์ A พิมพ์คำว่า INSTALL และตามด้วยพอร์ตขนานที่ใช้ LPT1 หรือ LPT2 เช่น ถ้าต้องการติดตั้งที่ LPT1

A> INSTALL LPT1 <ENTER>

เปลี่ยนการทำงานไปที่ไดรฟ์ C ลองพิมพ์คำสั่ง

C> DIR

จะพบ subdirectory TC ซึ่งเป็นไดเรคทอรีของโปรแกรม Turbo C version 2.0 เป็นอันเสร็จการติดตั้งซอฟต์แวร์

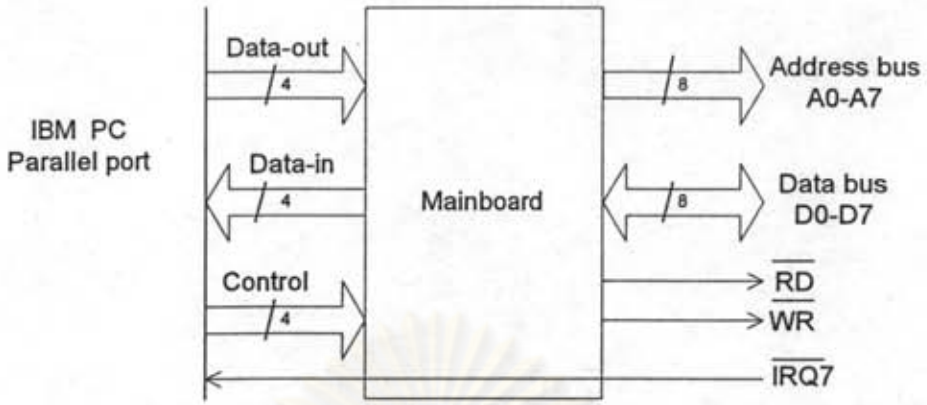
## 6. อุปกรณ์ที่ใช้ในการทดลอง

1. ชุดทดลอง
2. ออสซิลโลสโคป 2 แชนแนล
3. แหล่งจ่ายไฟ +5V ขนาด 200 mA, +12V 1.5A, -12V 15mA
4. ดิจิทัลโวลต์มิเตอร์
5. ลอจิกโพรบ
6. คู่มือการทดลอง ประกอบด้วย
   - ทฤษฎีการทดลองและใบงานการทดลอง
   - คู่มือการเขียนโปรแกรมภาษา C
   - คู่มือการเขียนโปรแกรมภาษา C สำหรับงานควบคุม
7. หนังสือ TURBO C Reference Guide ของบริษัท Borland
8. เครื่องคอมพิวเตอร์ IBM PC หรือ PC Compatible รุ่นใดก็ได้ที่มีพอร์ตขนาน และพอร์ตอนุกรม (พอร์ตอนุกรมใช้เฉพาะการทดลองเรื่องการสื่อสารข้อมูลอนุกรม)
9. โปรแกรม TURBO C ของ บริษัท Borland version 2.0

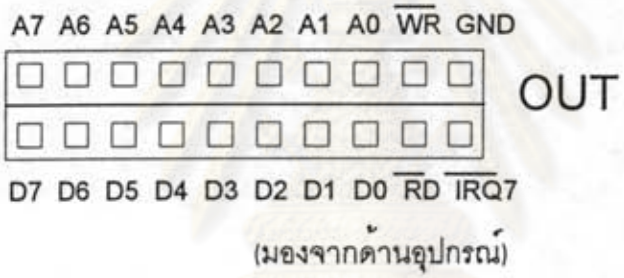
## 7. การทดสอบ

### 7.1 เมนบอร์ด

ทำหน้าที่ขยายพอร์ตเครื่องพิมพ์ให้เป็นพอร์ตขนาด 8 บิต ข้อมูล จำนวน 256 พอร์ต (8 บิตแอดเดรส) เพื่อใช้ในการอินเตอร์เฟซ แสดงดังในไดอะแกรม สัญญาณที่ได้เป็นสัญญาณพื้นฐานในการอินเตอร์เฟซของระบบไมโครโปรเซสเซอร์




การจัดตำแหน่งสัญญาณที่ขั้ว OUT ของเมนบอร์ด ซึ่งเป็นลักษณะของขั้ว OUT ของมอดูลอื่นๆ ด้วย แสดงดังรูป




(มองจากด้านอุปกรณ์)

A0 - A7 คือ บัสแอดเดรส D0 - D7 คือ บัสข้อมูล $\overline{WR}$ คือ สัญญาณการเขียนข้อมูล $\overline{RD}$ คือ สัญญาณการอ่านข้อมูล $\overline{IRQ7}$ คือ สัญญาณร้องขอการอินเตอร์รัปต์ระดับ 7 ในการอินเตอร์เฟซกับวงจรภายนอก จะต้องออกแบบให้วงจรภายนอกรับข้อมูลไปจากบัสข้อมูลในช่วงจังหวะเขียนข้อมูลในจังหวะขอบขาขึ้นของสัญญาณ $\overline{WR}$ และให้อุปกรณ์ภายนอกส่งข้อมูลมาที่บัสข้อมูลในช่วงจังหวะของการอ่านข้อมูลภายในช่วงเวลาไม่เกิน 3 μS หลังจากที่สัญญาณ $\overline{RD}$ เปลี่ยนสถานะเป็น อย่างไรก็ตาม ที่กล่าวมาข้างต้นเป็นหลักในการอินเตอร์เฟซกับเมนบอร์ดอย่างง่ายๆ โดยรับรองว่าจะใช้งานอย่างได้ผล เราสามารถยืดหยุ่นได้โดยอาศัยการดูจาก timing diagram ของสัญญาณเหล่านี้โดยตรง

การเขียนอ่านข้อมูลกับพอร์ตจะใช้คำสั่ง IN, OUT ซึ่งมีรูปแบบ ดังนี้

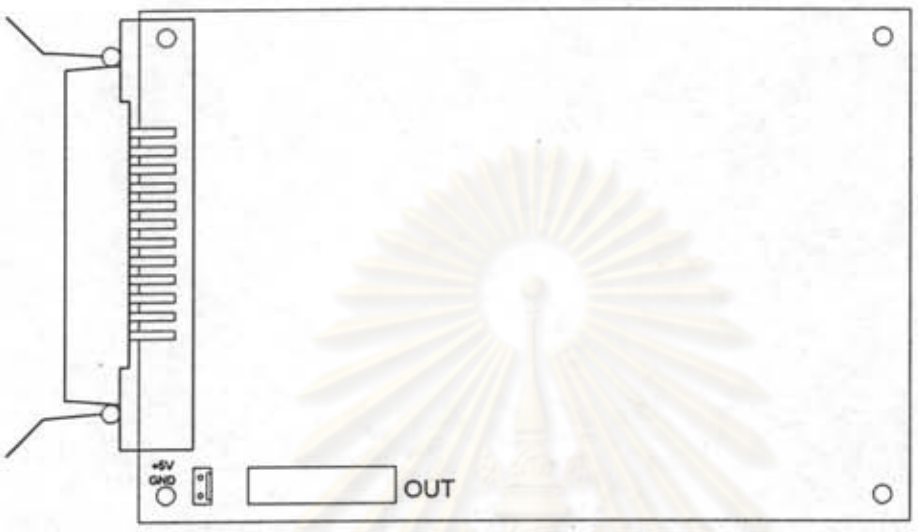
IN(port address) - อ่านข้อมูลจากพอร์ตหมายเลข port address

เช่น X = IN(OXC1) ;

OUT(port address, data) - เขียนข้อมูล data ไปยังพอร์ตหมายเลข port address

เช่น OUT (OXCO, OXFF) ;

โดยจะต้องมีประโยค #include<edl.h> อยู่ภายในโปรแกรม รูปของมอดูลเมนบอร์ดแสดงดังข้างล่าง




เมนบอร์ด

ขั้นตอนการทดสอบ

1. ต่อเมนบอร์ดเข้ากับเครื่อง IBM PC โดยไม่ต้องป้อนไปเลี้ยงภายนอก
2. เข้าสู่โปรแกรม TURBO C เขียนโปรแกรมตามโปรแกรมที่ 1
3. ทดลองรันโปรแกรม
4. ใช้ลอจิกโพรบจับดูที่ขา A0 - A7 ขา D0 - D7 ขา $\overline{WR}$ และขา $\overline{RD}$ สังเกตสถานะของสัญญาณเหล่านี้

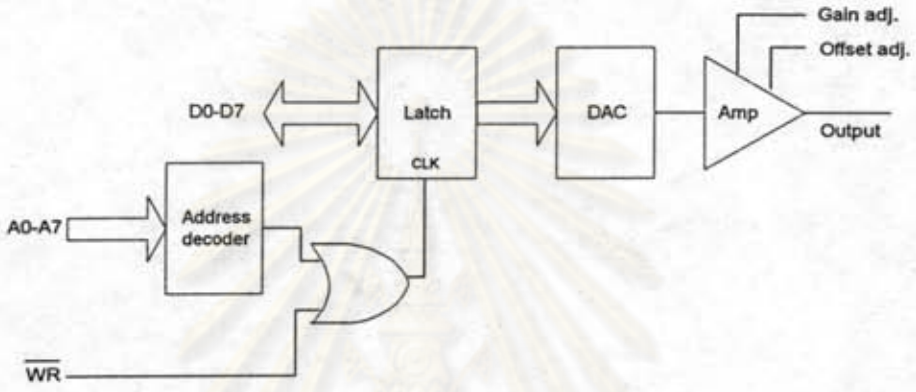
```
/*---------------- Program 1  :  Test Mainboard -----------------*/
#include<edl.h>

main()
{
      for(;;){                    /* Loop forever */
            out(0XAA,0X55);       /* Write data to address AAh */
            in(0XAA);             /* Read data from address AAh */
      }
}
```

7.2 มอดูล ADAC

ทำหน้าที่ในการแปลงสัญญาณจากดิจิทัลเป็นแอนะล็อก (D/A) และจากแอนะล็อกเป็นดิจิทัล (A/D)

7.2.1 D/A

ใช้ไอซีเบอร์ DAC0808 ทำหน้าที่ในการแปลงสัญญาณจากดิจิทัลเป็นแอนะล็อก ซึ่งมีความละเอียดขนาด 8 บิต ในส่วนของเอาต์พุต เป็นวงจร signal conditioning ทำหน้าที่ปรับแรงดันขาออกให้อยู่ในช่วงที่ต้องการ บล็อกไดอะแกรมของ D/A แสดงดังในรูป วงจรถอดรหัสแอดเดรสจะทำการถอดรหัสหมายเลข C6h ให้แก่ D/A



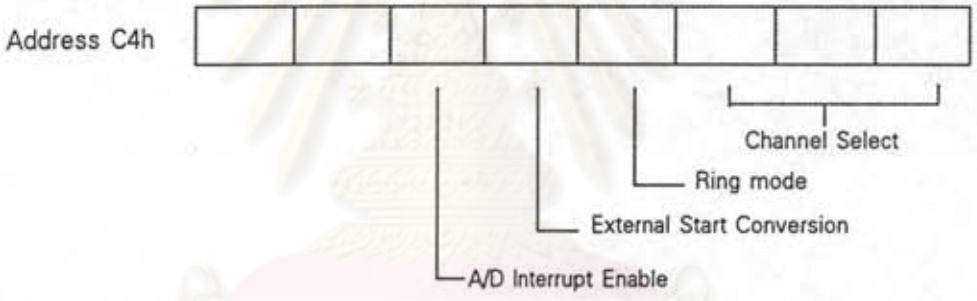

ขั้นตอนการทดสอบ

1. ต่อมอดูลเข้ากับเมนบอร์ด
2. ต่อไฟเลี้ยงให้แก่มอดูลโดยใช้สายที่เตรียมไว้ให้ ระวังอย่าต่อสลับขั้ว
3. เขียนโปรแกรมตามโปรแกรมที่ 2
4. ทดลองรันโปรแกรมและทำตามคำสั่งของโปรแกรม
5. นำโวลต์มิเตอร์มาวัดแรงดันขาออก สังเกตค่าที่อ่านได้
6. ทดลองปรับ offset และ gain สังเกตค่าที่อ่านได้จากโวลต์มิเตอร์

```
/*---------- Program 2 : Test D/A ----------*/
#include<edl.h>
#include<stdio.h>
#define DAC 0XC6                    /*D/A port*/

main()
{
      int i;
      do{
            scanf("Enter digital value %d \n",&i);    /*Data input*/
            out(DAC,i);                 /*Out data to D/A*/
      } while((i>=0)&&(i<=255));        /*Loop until i is out of range*/

}
```

7.2.2 A/D

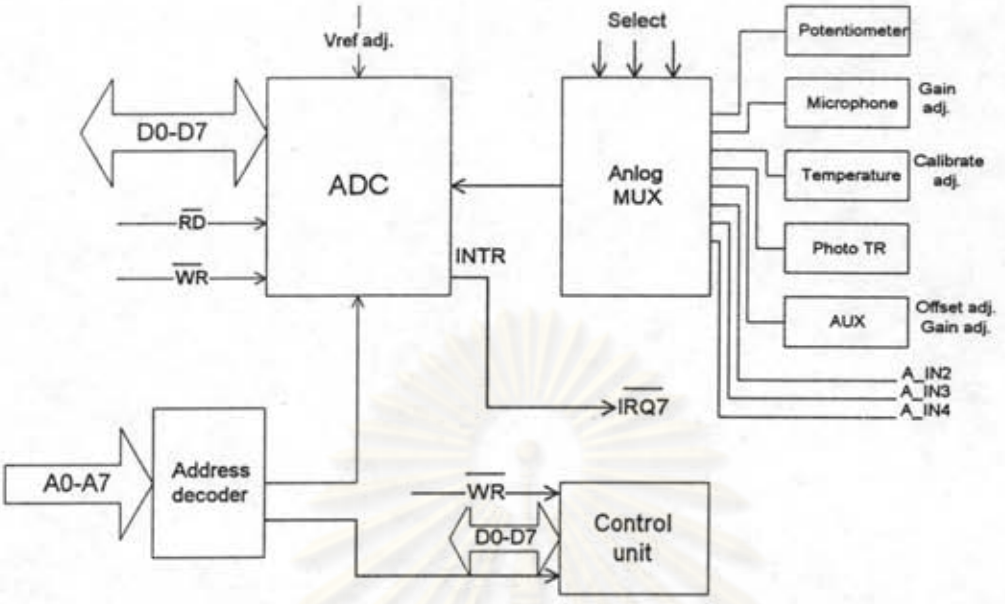
ใช้ไอซีเบอร์ ADC0804 ในการแปลงสัญญาณจากแอนะล็อกเป็นดิจิทัล มีความละเอียด 8 บิต และมีความเร็วในการแปลงสัญญาณ 100 μs เป็นไอซีชนิด microprocessor interface มีสัญญาณ $\overline{RD}$, $\overline{WR}$, INTR ที่สามารถเชื่อมต่อกับระบบไมโครโปรเซสเซอร์ และทำการควบคุมได้โดยตรง ในส่วนของ A/D นี้มีวงจรรับสัญญาณแอนะล็อกขาเข้าจากเซนเซอร์ต่างๆ และสัญญาณแอนะล็อกจากภายนอกซึ่งมีวงจร signal conditioning ไว้คอยปรับแต่งสัญญาณให้อยู่ที่ช่วงที่เหมาะสม รวมทั้งหมด 8 แชนแนล ทำการเลือกแชนแนลด้วยซอฟต์แวร์ โดยการโปรแกรมรีจิสเตอร์ควบคุม รีจิสเตอร์ควบคุมยังทำหน้าที่ควบคุมโหมดการทำงานและการอินเตอร์รัปต์ บล็อกไดอะแกรมของ A/D และรีจิสเตอร์ควบคุม แสดงดังในรูป รีจิสเตอร์ควบคุมอยู่ที่พอร์ตหมายเลข C4h และ A/D data อยู่ที่พอร์ตหมายเลข C5h




| บิต 2-0 | แชนแนล | สัญญาณแอนะล็อก |
|---|---|---|
| 000 | 0 | Potentiometer |
| 001 | 1 | Microphone |
| 010 | 2 | Temperature |
| 011 | 3 | Photo transistor |
| 100 | 4 | AUX |
| 101 | 5 | A_IN2 |
| 110 | 6 | A_IN3 |
| 111 | 7 | A_IN4 |

| โหมด | การทำงาน |
|---|---|
| Ring | แปลงสัญญาณแบบวนรอบ |
| External start conversion | แปลงสัญญาณโดยการกระตุ้นจากสัญญาณภายนอก |

บล็อกไดอะแกรมของวงจร A/D แสดงดังรูปข้างล่างนี้

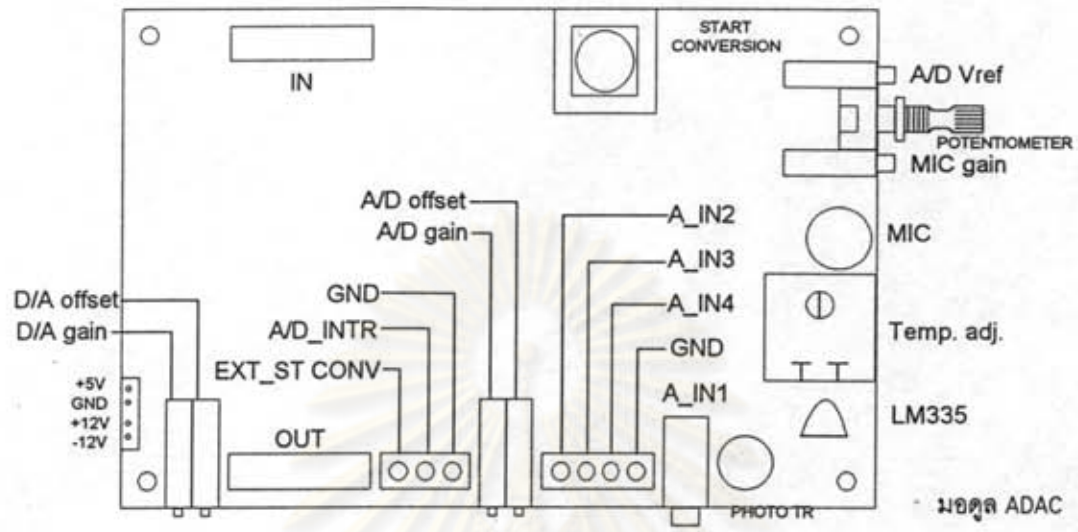



ขั้นตอนการทดสอบ

1. ทำเช่นเดียวกับข้อ 1-2 ของ D/A
2. เขียนโปรแกรมตามโปรแกรมที่ 3
3. ทดลองรันโปรแกรม
4. ทดลองปรับ Vref ของ ADC, Potentiometer, Mic gain, Temp.adj. และเปลี่ยนแปลงตัวแปรทางฟิสิกส์ เช่น แสง เสียง อุณหภูมิ แล้วสังเกตดูค่าที่เปลี่ยนไป
5. ในกรณีที่ไม่ทำงานให้ทดลองกดปุ่ม start conversion

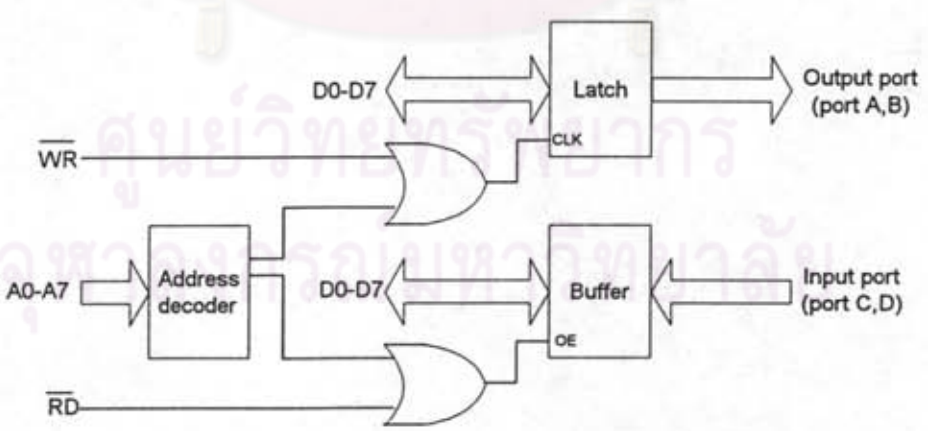
```
/*------ Program 3 : Test A/D ----------*/
#include<edl.h>
#include<stdio.h>
#define ACTRL 0XC4              /*Control register*/
#define ADC   0XC5              /*Data register*/
#define CTRLDATA 0X08           /*Control data*/

main()
{
      unsigned char channel;
      scanf("Which channel? %u \n",channel); /*Input channel select*/
      if (channel>7) exit(0);       /*If out of range -> exit*/
      out(ACTRL,CTRLDATA+channel);  /*Out control data*/
      while(!kbhit())               /*Loop until key pressed*/
        printf("Reading data is %u \n",in(ADC)); /*Out reading data*/
}
```

รูปของมอดูล ADAC แสดงดังรูปข้างล่าง



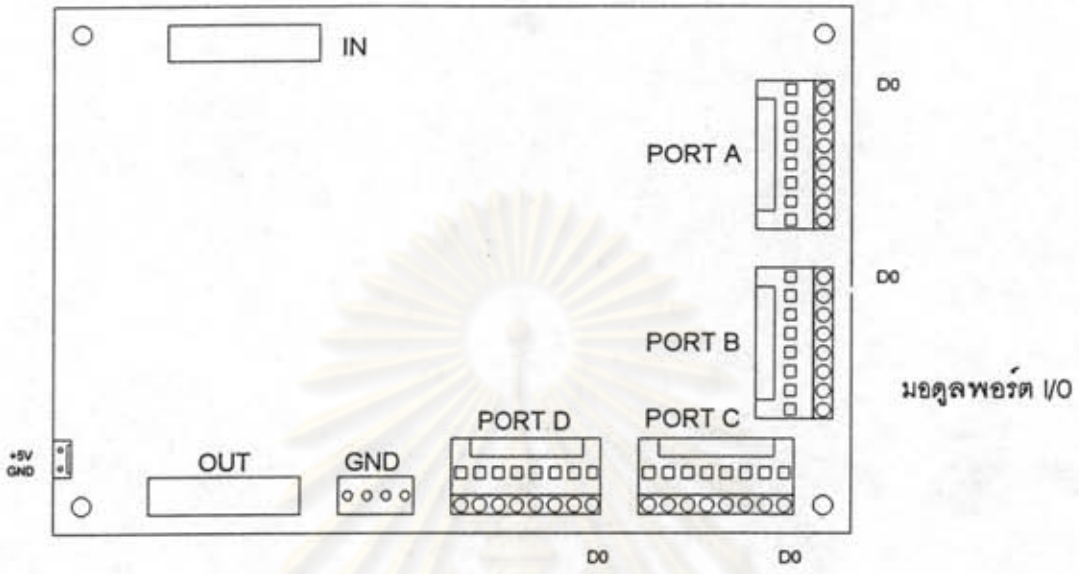

7.3 <u>มอดูลพอร์ต I/O</u>

เป็นพอร์ตสัญญาณดิจิทัล ซึ่งประกอบด้วยพอร์ตสัญญาณขาเข้า 2 พอร์ต คือ พอร์ต C และ D พอร์ตสัญญาณขาออก 2 พอร์ต คือ พอร์ต A และ B มีแลตช์ฟังก์ชัน บล็อกไดอะแกรมของมอดูลแสดงดังในรูป การใช้งานสามารถเขียนอ่านข้อมูลไปที่พอร์ตโดยตรงได้ทันที โดยไม่ต้องโปรแกรมรีจิสเตอร์ใดๆ




| Port | Address (Hex) |
|---|---|
| A | C8 |
| B | C9 |
| C | CA |
| D | CB |

รูปของมอดูล ADAC แสดงดังรูปข้างล่าง




ขั้นตอนการทดสอบ

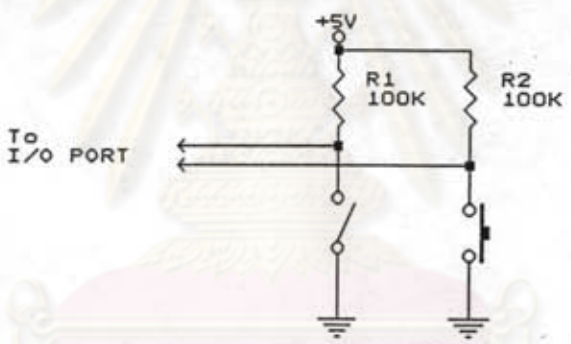
1. ต่อมอดูลเข้ากับเมนบอร์ด
2. ป้อนไฟเลี้ยงให้แก่มอดูลโดยใช้สายที่ให้มากับชุดฝึกทดลอง
3. เขียนโปรแกรมตามโปรแกรมที่ 4
4. ทดลองรันโปรแกรม
5. ทดสอบพอร์ตสัญญาณขาออกโดยใช้ลอจิกโพรบจับดูสัญญาณต่างๆ ของพอร์ต A,B สังเกตความถี่ในการกะพริบของแต่ละบิต บิต MSB จะกะพริบถี่น้อยที่สุด ส่วนบิต LSB จะกะพริบถี่มากที่สุด
6. ทดสอบพอร์ตสัญญาณขาเข้าโดยป้อนสัญญาณสถานะ high และ low เข้าทีละบิต และสังเกตผลลัพธ์ที่หน้าจอคอมพิวเตอร์

```
/*----- Program 4 : Test port I/O module -----*/
#include<edl.h>
#include<stdio.h>
#define PORT 0XC8;

main()
{
      unsigned char i=0,j;

      /*-----Test output port-------*/
      printf("Now testing output port..., press anykey to continue \n
");
      while(!kbhit()){
```

```
            i = i+1;
            out(PORT,i);        /*Test port A&B by writing data to port*/
            out(PORT+1,i);
            delay(100);
        }
    /*-----Test input port--------*/
        for(j=0;j<2;j++)        /*Test port C&D , C=#0 D=#1*/
        for(i=0;i<8;i++)        /*Test each bit*/
              for(;;){
                    printf("Now testing input port#%u bit d%u ,",j,i);
                          printf("logic is %u \n",in(PORT+2+j)&(1<<i));
                    if (kbhit()) break;
              }
}
```

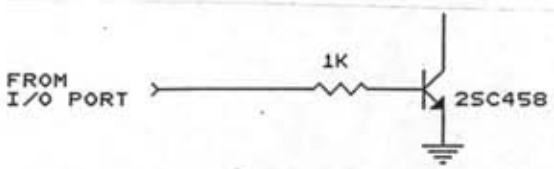
### 7.4 มอดูลสวิตช์

ประกอบด้วยวงจรรับสัญญาณจากสวิตช์ วงจรขับ LED วงจรขับรีเลย์ วงจรขับทรานซิสเตอร์ วงจรสวิตช์มีลักษณะเช่นเดียวกับพอร์ตสัญญาณเข้าของมอดูลพอร์ต I/O แต่เพิ่มในส่วนของสวิตช์เพื่อป้อนสัญญาณดิจิทัลเข้าไปดังรูป

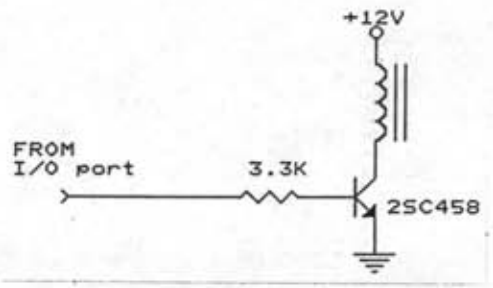



วงจรขับ LED วงจรขับรีเลย์ วงจรขับทรานซิสเตอร์ มีลักษณะเช่นเดียวกับพอร์ตสัญญาณออกของมอดูลพอร์ต I/O แต่เพิ่มเติมในบางส่วน ดังนี้


วงจรขับ LED




วงจรขับทรานซิสเตอร์

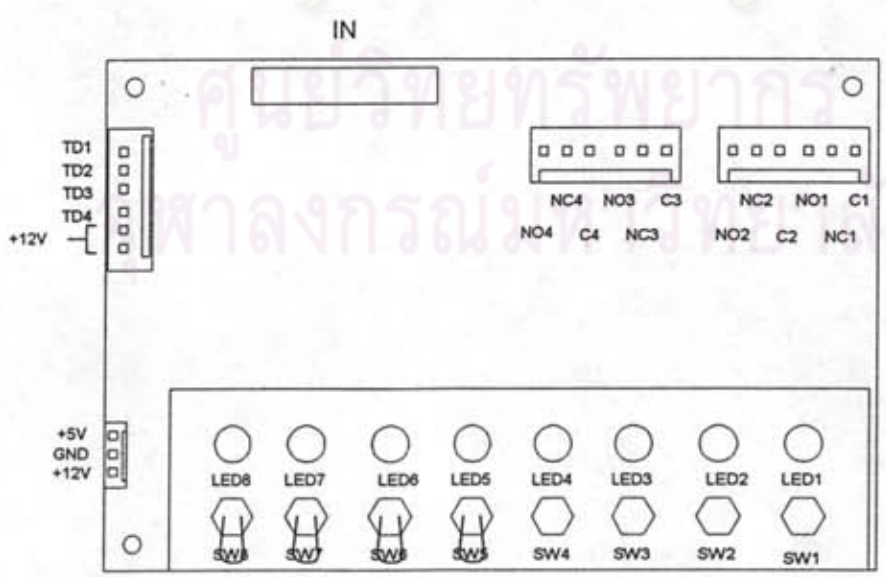



วงจรขับรีเลย์

วงจรขับทรานซิสเตอร์สามารถนำไปใช้ขับมอดูลสเตปปิงมอเตอร์ได้โดยตรง และวงจรขับรีเลย์สามารถนำไปใช้ควบคุมการเปิด-ปิด อุปกรณ์ไฟฟ้า หมายเลขแอดเดรสของพอร์ตต่างๆ แสดงดังในตาราง

| Port | Adress (Hex) |
|---|---|
| LED | C0 |
| Switch | C1 |
| Relay | C2 (bit0-3) |
| Transistor | C2 (bit4-7) |

รูปของมอดูลสวิตช์แสดงดังรูปข้างล่างนี้




มอดูลสวิตช์

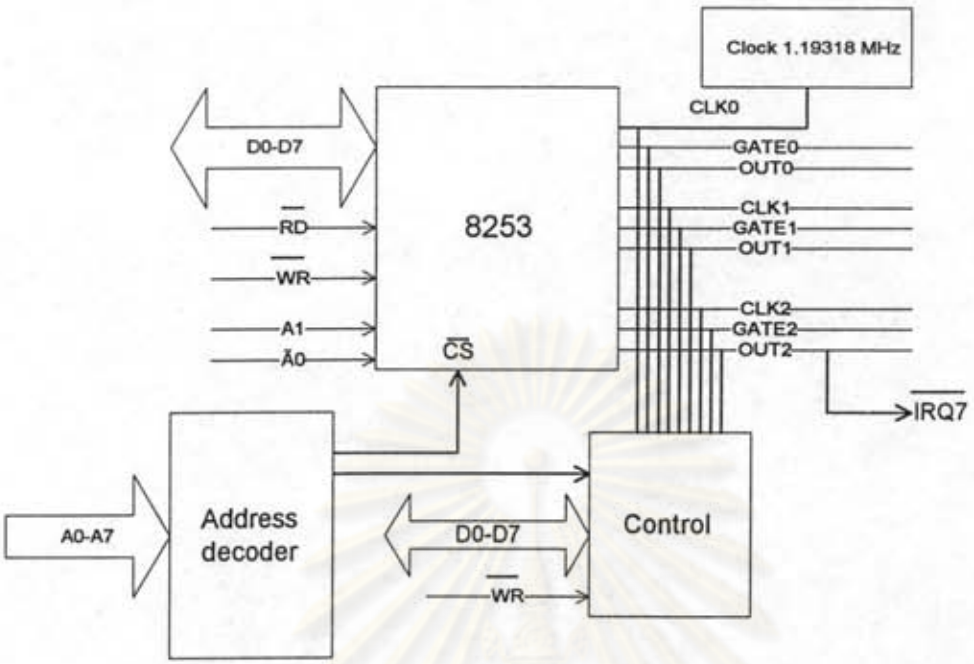
ขั้นตอนการทดสอบ

1. ต่อมอดูลเข้ากับเมนบอร์ด
2. ป้อนไฟเลี้ยงให้แก่มอดูลโดยใช้สายที่ให้มากับชุดฝึกทดลอง
3. เขียนโปรแกรมตามโปรแกรมที่ 5
4. รันโปรแกรมและทดลองกดสวิตช์ปุ่มใดๆ สังเกตการติด-ดับ ของ LED
5. กดคีย์ใดๆ บนคีย์บอร์ดเพื่อทำงานต่อ ในการทดสอบรีเลย์ และวงจรขับทรานซิสเตอร์ สังเกตเสียงการสับเปลี่ยนหน้าสัมผัสของรีเลย์
6. ใช้ลอจิกโพรบวัดดูที่ขา TD1-TD4 สังเกตการเปลี่ยนแปลง

```
/*-----Program 5 : Test SW module------*/
#include<edl.h>
#include<stdio.h>
#define LED 0XC0
#define SW  0XC1
#define DRV 0XC2

main()
{
      int i=1;
      prinrf("Test switch and LED....<press any switch>\n");
      while(!kbhit())
            out(LED,in(SW));          /*out LED by SW data*/
      printf("Relay & TR driver testing....\n");
      while(!kbhit()){
            out(DRV,i);
            i*=2;
            if(i>128) i=1;
            delay(500);
      }
}
```

7.5 มอดูล T/C

เป็นวงจร Timer/Counter ใช้ชิป 8253 เชื่อมต่อเข้ากับระบบบัสของชุดฝึกทดลอง มีวงจรกำเนิดสัญญาณซึ่งมีความถี่ 1.19318 MHz เพื่อใช้สร้างฐานเวลาให้แก่ 8253 และได้เพิ่มเติมวงจรควบคุมการป้อนสัญญาณให้แก่ขาต่างๆ ของ 8253 ทำให้สามารถควบคุมการทำงานได้ด้วยซอฟต์แวร์ บล็อกไดอะแกรมของมอดูลแสดงได้ดังรูป




<u>บล็อกควบคุมทำหน้าที่</u> ดังนี้

1. ควบคุมการป้อนสัญญาณให้แก่ขา CLK

ขา CLK จะต่อเข้ากับ clock 1.19 MHz ตลอดเวลา ส่วนขา CLK1-CLK2 เราสามารถเลือกได้ว่า จะป้อนสัญญาณให้จากแหล่งใด ดังนี้

1. clock 1.19 MHz

2. ขา OUT0 หรือ OUT1 โดย OUT0 สามารถเลือกต่อเข้ากับ CLK1 ส่วน OUT1 สามารถเลือกต่อเข้ากับ CLK2

3. แหล่งสัญญาณภายนอก คือ จากขั้ว CLK1, CLK2

2. ควบคุมการป้อนสัญญาณให้แก่ขา Gate

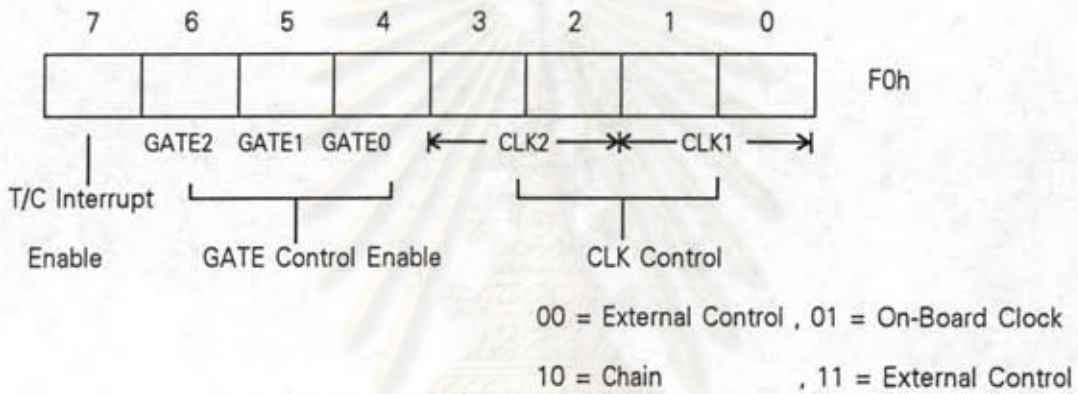
สามารถเลือกแหล่งสัญญาณ ได้ดังนี้

1. จากพอร์ตสัญญาณออกที่อยู่บนมอดูล ทำให้สามารถควบคุมสถานะของขา Gate ได้โดยการส่งข้อมูลไปที่พอร์ตสัญญาณออก

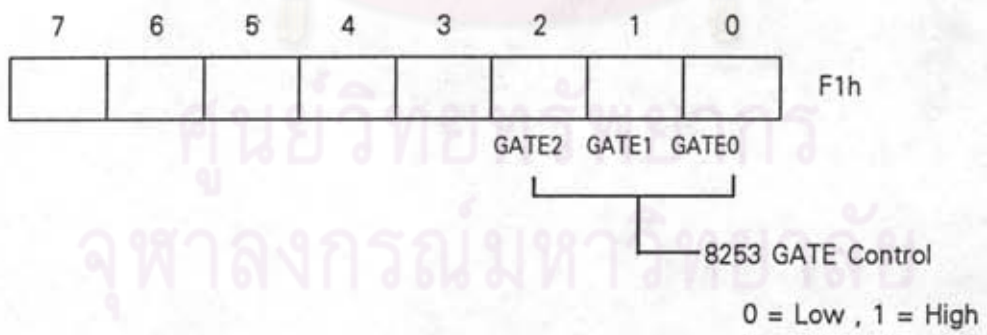
2. แหล่งสัญญาณภายนอก คือ จากขั้ว Gate0, Gate1, Gate2 นอกจากนี้ ขา OUT2 จะถูกนำไปอินเตอร์รัปต์ CPU ซึ่งสามารถทำการเอนาเบิล / ดิสเอเบิลได้

| Address (Hex) | I/O |
|---|---|
| F0 | T/C CTRL1 register |
| F1 | T/C CTRL2 register |
| F2 | Clear interrupt signal |
| F3 | T/C status register |
| F4 | Counter register0 |
| F5 | Counter register1 |
| F6 | Counter register2 |
| F7 | 8253 Control word |

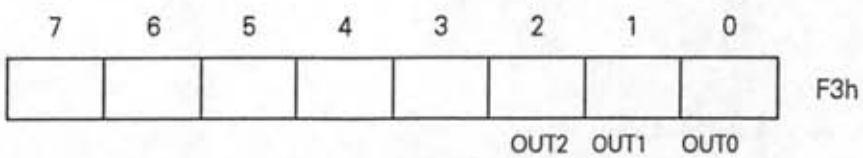
T/C CTRL1 Register

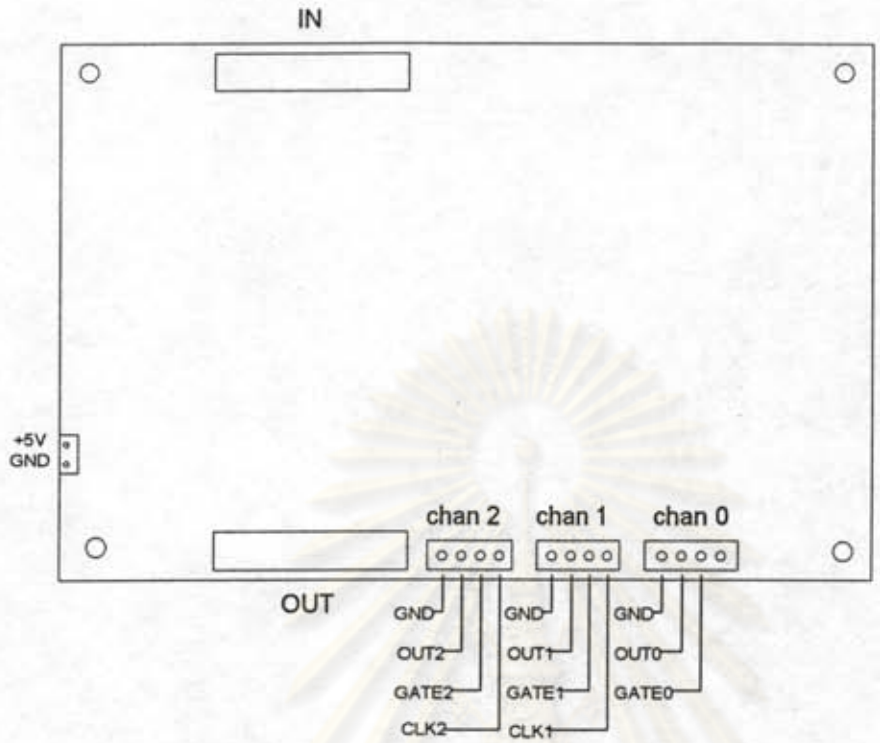



T/C CTRL2 Register




T/C Status Register






มอดูล T/C

ขั้นตอนการทดสอบ

1. ต่อมอดูล T/C เข้ากับเมนบอร์ด และต่อแหล่งจ่ายไฟเข้ากับมอดูล
2. เขียนโปรแกรมตามโปรแกรมที่ 6
3. ทดลองรันโปรแกรม
4. ใช้ออสซิลโลสโคปจับสัญญาณที่ขา OUT1 จะได้สัญญาณรูปคลื่นสี่เหลี่ยมที่มีความถี่ 1 kHz

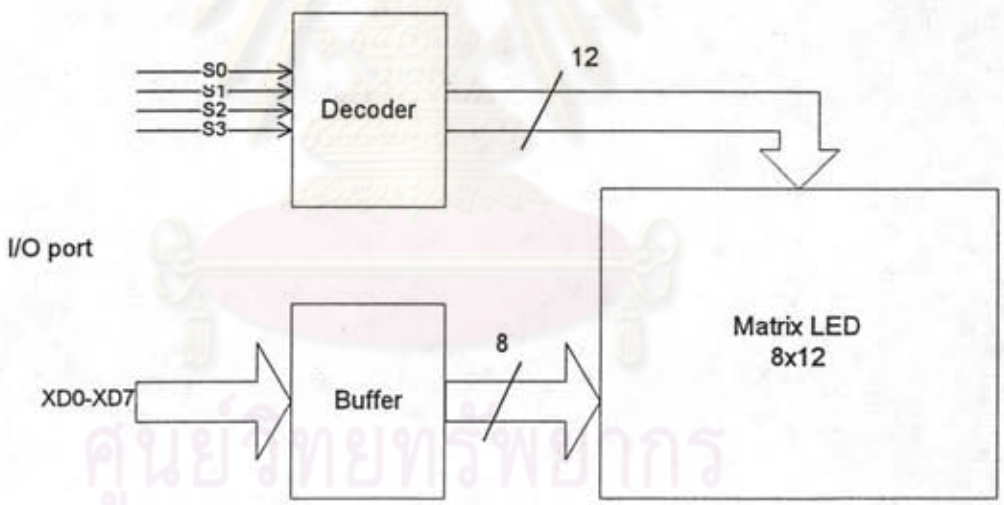


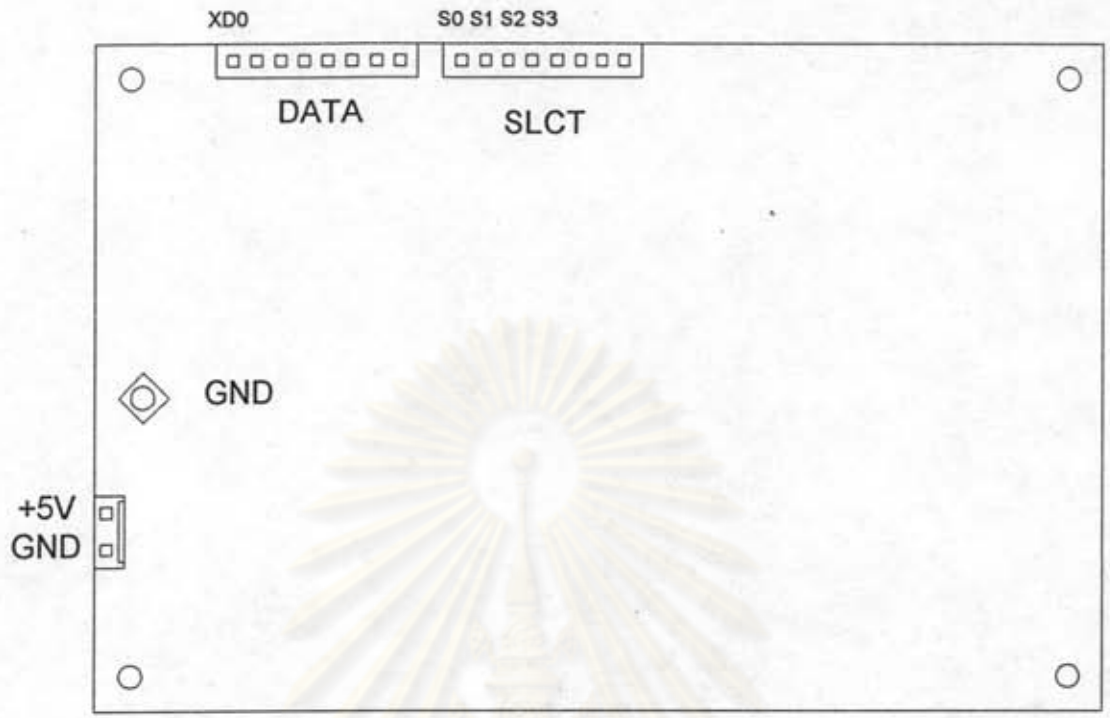
```
/*--------Program 6 : Test T/C module--------*/
#include<edl.h>
#include<stdio.h>
#define TC_CTRL1 0XF0
#define TC_CTRL2 0XF1
#define COUNTER1 0XF5
#define TC_WORD  0XF7

main()
{
      printf("Program T/C CTRL1 register \n");
      getch();
      out(TC_CTRL1,0x21);
      prinrf("Program T/C CTRL2 register \n");
      getch();
      out(TC_CTRL2,0X02);
      printf("Program 8253 control word \n");
      getch();
      out(TC_WORD,0X76)                          /*Mode 3 selected*/
```

```
        printf("Program counter register#1 \n");
        getch();
        out(COUNTER1,0XA9);
        out(COUNTER1,0X04);
        printf("Finish.....");
}
```

7.6 มอดูล Display

เป็นมอดูลที่ใช้ศึกษาเรื่องการแสดงผลด้วยเมทริกซ์ LED ซึ่งมีขนาด 8 x 12 จุด บล็อกไดอะแกรมของมอดูลแสดงดังในรูป การแสดงผลควบคุมโดยการส่งข้อมูลขนาด 8 บิตมาที่ XD0-XD7 หลังจากนั้น ส่งข้อมูลมาที่ S0-S3 ซึ่งจะผ่านวงจร decoder เพื่อทำการเลือกคอลัมน์ที่ต้องการแสดงผล LED ในคอลัมน์ที่เลือกก็จะติด-ดับตามรูปแบบข้อมูลที่ส่งมา เมื่อทำการส่งข้อมูลและมัลติเพลกซ์สัญญาณเลือกคอลัมน์ให้มีความถี่สูงขึ้น จนตาไม่สามารถมองเห็นการกะพริบได้ ก็จะเห็นภาพปรากฏขึ้นบนเมทริกซ์ LED






มอดูล DISPLAY

ขั้นตอนการทดสอบ

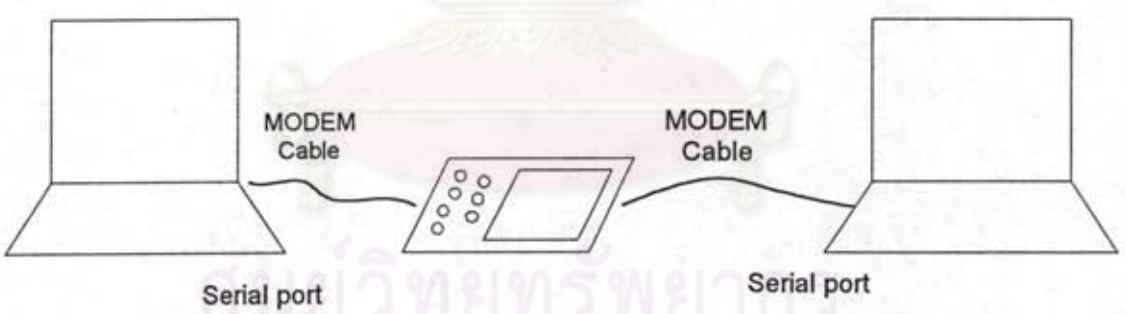
1. ต่อมอดูลพอร์ต I/O เข้ากับเมนบอร์ด แล้วต่อมอดูล Display เข้ากับมอดูลพอร์ต I/O โดยให้พอร์ต DATA ต่อเข้ากับพอร์ต A และพอร์ต SLCT ต่อเข้ากับพอร์ต B
2. ต่อแหล่งจ่ายไฟเข้ากับมอดูลทั้งสอง โดยใช้ขั้วกราวด์ร่วมกัน
3. เขียนโปรแกรมตามโปรแกรมที่ 7 และทดลองรันโปรแกรม

```
/*------Program 7 : Display-------*/
#include<edl.h>
#include<conio.h>
#define porta 0xc8
#define portb 0xc9

main()
{
      char i,c,a[12]={  0xfe,0x92,0x92,
                        0x00,0xfe,0x82,
                        0x44,0x38,0x00,
                        0xfe,0x80,0x80  };

      for(;;)
      {
            for(i=0;i<12;i++){
                  out(portb,i);           /* out scan line */
                  out(porta,a[i]);        /* out data */
```

```
                delay(1);
                out(porta,0);
            }
        if(c=kbhit()) if (c=getch()==27) break;
                                    /*Loop until press ESC key*/
        }
        out(portb,12);
}
```

### 7.7 มอดูล Serial Communication

เป็นมอดูลที่นำสัญญาณจากพอร์ตอนุกรมของเครื่อง IBM PC 2 เครื่องมาไว้บน proto board เพื่อทำการ jump สัญญาณต่างๆ เข้าด้วยกัน เพื่อทำการรับส่งข้อมูล ดังแสดงในรูป สัญญาณต่างๆ สามารถแบ่งได้เป็น สัญญาณข้อมูล (Tx,Rx) และสัญญาณแฮนด์เชค แต่ละสัญญาณจะมี LED แสดงสถานะของสัญญาณ ซึ่งมีไว้เพื่อสังเกตการทำงานในขณะรับส่งข้อมูล นอกจากการรับส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์ 2 เครื่อง ในกรณีที่มีเครื่องคอมพิวเตอร์เพียงเครื่องเดียว ก็สามารถทำ Loop - back คือ การส่งข้อมูลย้อนกลับเข้าเครื่อง เพื่อทำการศึกษาเรื่องการสื่อสารข้อมูลได้

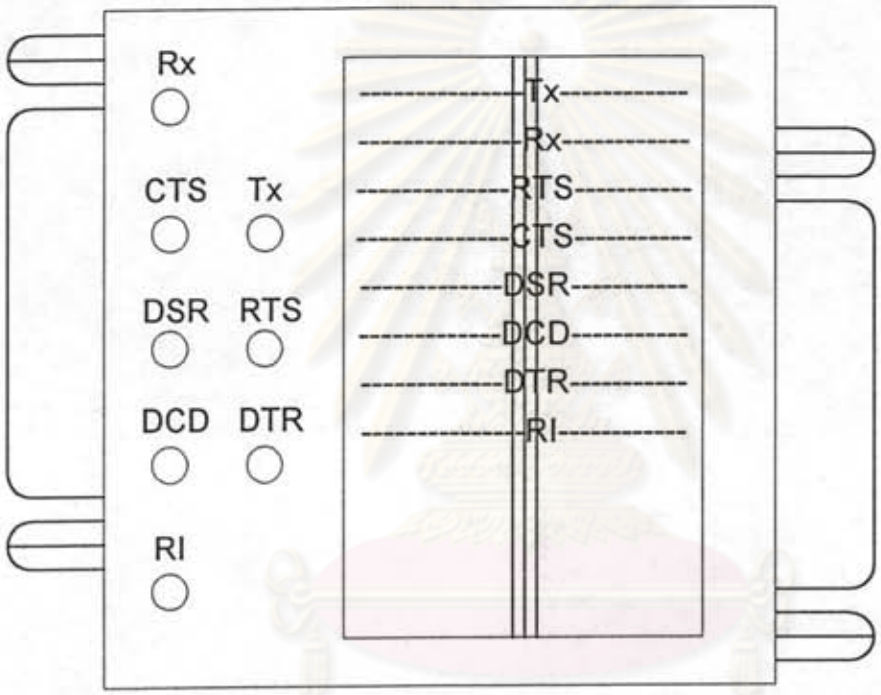





สัญญาณที่นำออกมาจากพอร์ตอนุกรม แสดงได้ดังในตาราง สายที่ใช้ต่อคือสายที่ใช้เชื่อมต่อ MODEM เข้ากับ IBM PC ธรรมดา ซึ่งสามารถหาซื้อได้ทั่วไป (ไม่มีการไขว้สายภายใน)

| ชื่อย่อ | ชื่อสัญญาณ |
|---|---|
| TX | Transmitted data |
| Rx | Received data |
| RTS | Request to send |

| ชื่อย่อ | ชื่อสัญญาณ |
|---|---|
| CTS | Clear to send |
| DSR | Data set ready |
| DCD | Data carier detect |
| DTR | Data terminal ready |
| RI | Ring indicator |




มอดูล

SERIAL COMMUNICATION



ขั้นตอนการทดสอบ

1. ต่อมอดูลเข้ากับเครื่อง IBM PC เพียง 1 เครื่อง โดยใช้ด้านซ้ายของมอดูล และใช้สาย MODEM ในการเชื่อมต่อ

2. ทดลองเขียนและรันโปรแกรมต่อไปนี้ สังเกต LED แสดงสถานะของสัญญาณ RTS, DTR หลังจากนั้นทดลองเปลี่ยนข้อมูลที่ส่งไปยังพอร์ตเป็น 0X01, 0X02, 0XFF

```
#include<dos.h>
#define SERIAL 0x3f8          /*Replace 0x3f8 with 0x2f8 for COM2*/

main()
{
```

```
        outportb(SERIAL+4,0x03);
}
```

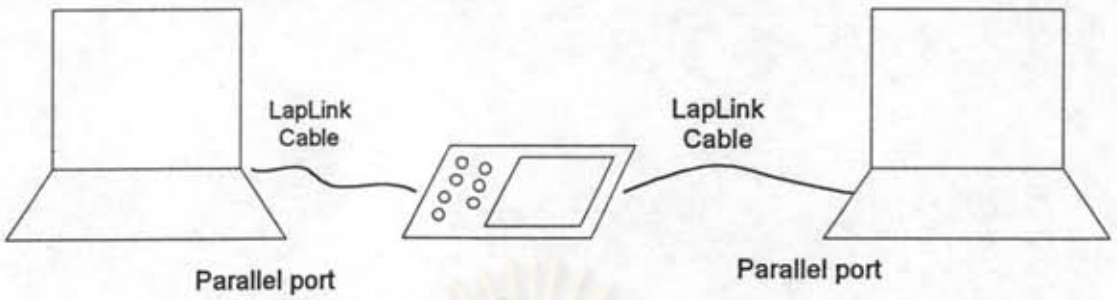
3. ทำ Loop - back โดย jump สัญญาณ Tx เข้ากับ Rx บน proto board

4. เขียนโปรแกรมรับส่งข้อมูลจำนวน 1 ไบต์ ดังต่อไปนี้ และทดลองรันโปรแกรม

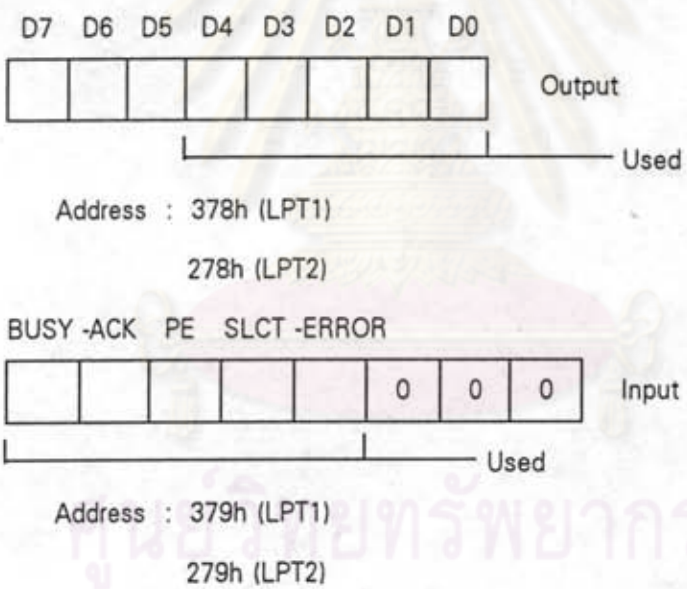
```
/*-------Program 8 : Serial communication--------*/
#include<stdio.h>
#include<dos.h>
#include<conio.h>
#define SERIAL 0x3f8

main()
{
        char empty;

        outportb(SERIAL+3,0x80);        /*Control register-set up for
                                          300 baud */
        outportb(SERIAL,0x80);          /*Low order of divisor*/
        outportb(SERIAL+1,0x01);        /*High order of divisor*/
        outportb(SERIAL+3,0x03);        /*No parity, 8 bit character*/

/*---------send a character---------*/
        do{
           empty = inportb(SERIAL+5)&0x20; /*Test Tx holding register*/
        }while(!empty);
        outportb(SERIAL,'A');                /*character to send*/

/*-------receive a character-------*/
        do{
           empty = inportb(SERIAL+5)&1;     /*Test Rx buffer*/
        }while(empty);
        printf("Received character is ");
        putch(inportb(SERIAL));              /*receive*/
}
```

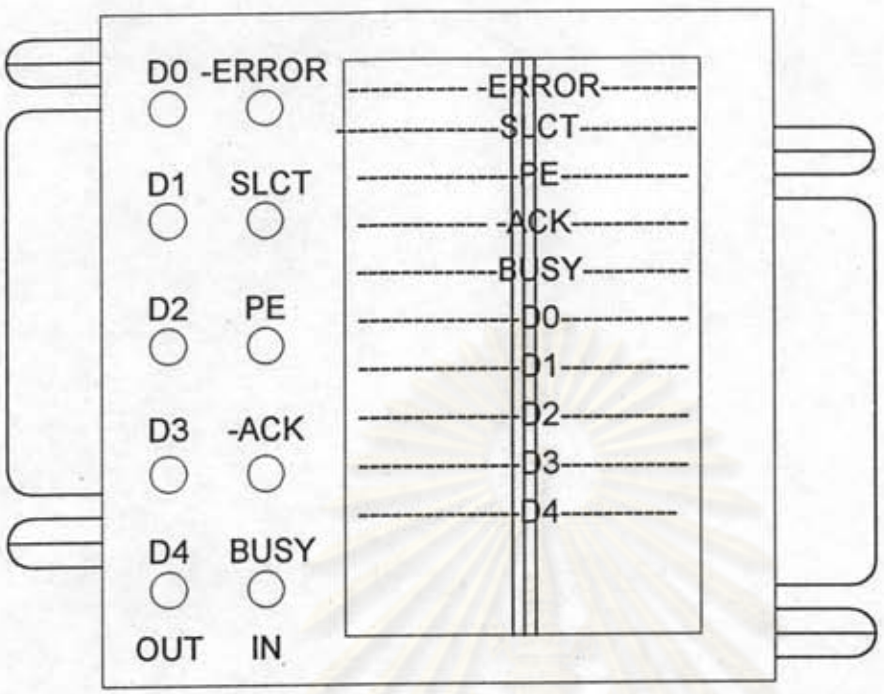
7.8 <u>มอดูล Parallel Communication</u>

เป็นมอดูลที่นำสัญญาณจากพอร์ตขนานของเครื่อง IBM PC 2 เครื่องมาไว้บน proto board เพื่อทำการ jump สัญญาณต่างๆ เข้าด้วยกัน เพื่อรับส่งข้อมูลแบบขนาน ดังแสดงในรูป สัญญาณแบ่งออกเป็น สัญญาณออกจำนวน 5 บิต และสัญญาณเข้าจำนวน 5 บิต โดยทำการส่งแบบ Simplex ในสายส่ง สัญญาณทั้ง 5 บิต แบ่งเป็นสัญญาณข้อมูลจำนวน 4 บิต และสัญญาณแฮนด์เชคจำนวน 1 บิต แต่ละสัญญาณมี LED แสดงสถานะของสัญญาณ นอกจากการรับส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์ 2 เครื่องแล้ว เราอาจทำ Loop - back เพื่อส่งข้อมูลย้อนกลับเข้าเครื่องได้ โดยการ jump สายบน proto board




สัญญาณที่นำออกมาจากพอร์ตขนานมีดังแสดงในรูปของมอดูล สายที่ใช้เชื่อมต่อคือสายเคเบิลแบบ LapLink ซึ่งสามารถหาซื้อได้ทั่วไป ตำแหน่งของบิตและหมายเลขพอร์ตของสัญญาณต่างแสดงดังในรูปต่อไปนี้






มอดูล

PARALLEL COMMUNICATION

ขั้นตอนการทดลอง

1. ต่อมอดูลเข้ากับเครื่อง IBM PC 1 เครื่อง โดยใช้ด้านซ้ายของมอดูล สายที่ใช้เชื่อมต่อ คือสาย LapLink

2. ทดลองเขียนและรันโปรแกรมต่อไปนี้ สังเกตสถานะของ LED แล้วทดลองเปลี่ยนค่าของข้อมูลที่ส่งออกไปเป็นค่าต่างๆ

```
#include<dos.h>
#define LPT 0x378
main()
{
        outportb(LPT,0xff);
}
```

3. ทำ Loop - back โดย jump สายสัญญาณดังต่อไปนี้

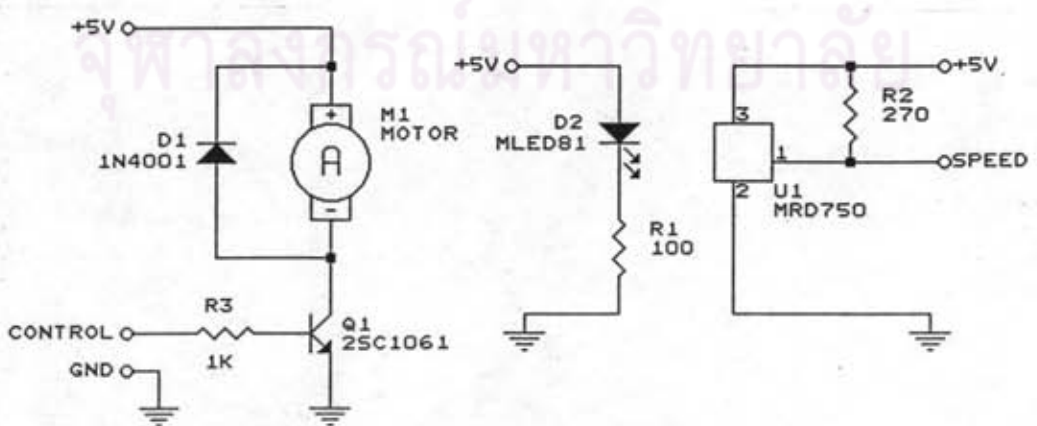
| | | |
|---|---|---|
| -ERROR | ต่อเข้ากับ | D0 |
| SLCT | ต่อเข้ากับ | D1 |
| PE | ต่อเข้ากับ | D2 |
| -ACK | ต่อเข้ากับ | D3 |
| BUSY | ต่อเข้ากับ | D4 |

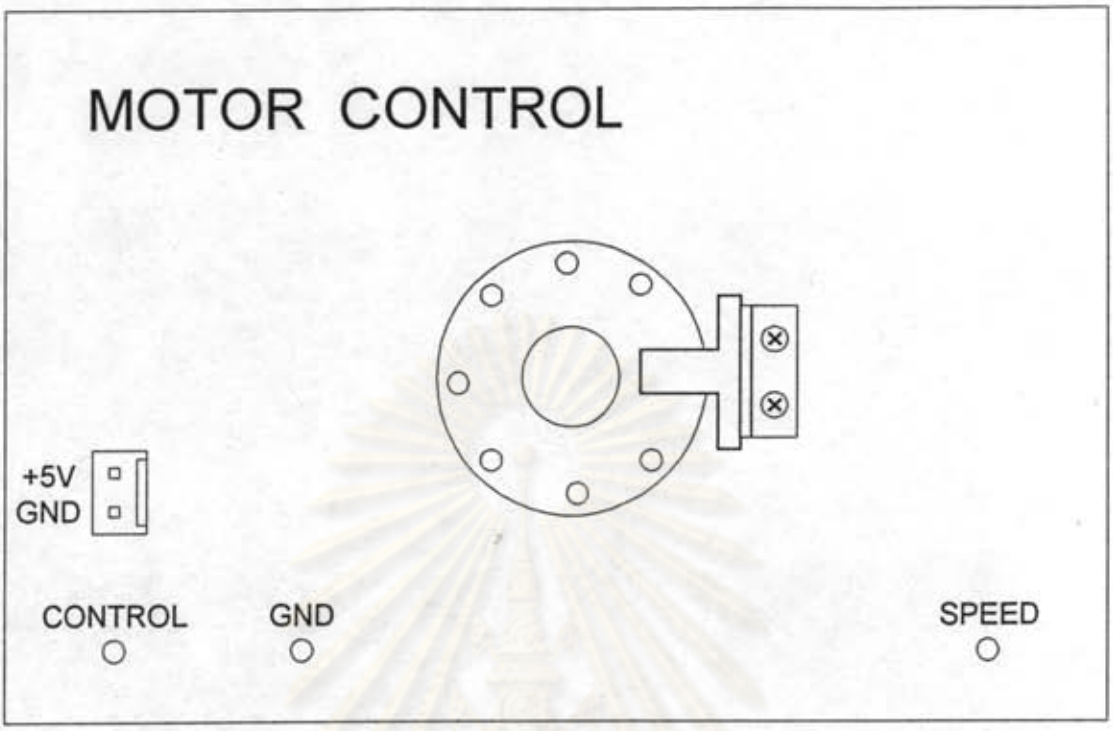
4. เขียนและรันโปรแกรมต่อไปนี้ เพื่อทดลองรับส่งข้อมูลจำนวน 1 ไบต์

```
/*--------Program 9 : Parallel communication-----*/
#include<dos.h>
#define LPT 0x378       /*replace with 0x278 for LPT2*/

main()
{
      char lrc,hrc;

/*----send & receive low nibble(4 bit)-----*/
      outportb(LPT,'A'&0x0f);              /*send*/
      lrc = (inportb(LPT+1)>>3)&0x0f;      /*receive*/

/*----send & receive high nibble-----------*/
      outportb(LPT,('A'>>4)&0x0f);         /*send*/
      hrc = (inportb(LPT+1)<<1)&0xf0       /*receive*/

/*----print character received-------------*/
      printf("Received character is ");
      putchar(hrc|lrc);
}
```

## 7.9 มอดูล Motor Control

มอดูลนี้ใช้ในการทดลองเรื่องการวัดความเร็วและการควบคุมความเร็วของมอเตอร์ มอดูลมีวงจรดังแสดงในรูป อุปกรณ์ MLED81 คือ ตัวส่งแสงอินฟราเรด และ MRD750 คือตัวรับแสงในย่านอินฟราเรด การวัดความเร็วของมอเตอร์ทำได้โดยการวัดความถี่ของสัญญาณพัลส์ ที่ขั้ว SPEED แล้วนำไปคำนวณหาความเร็วต่อไป การควบคุมความเร็วทำได้โดยการส่งพัลส์มาที่ขั้ว CONTROL เพื่อกระตุ้นมอเตอร์ให้เปิด-ปิดเป็นจังหวะ ซึ่งก็คือวิธีการแบบ PWM (pulse-width modulation)

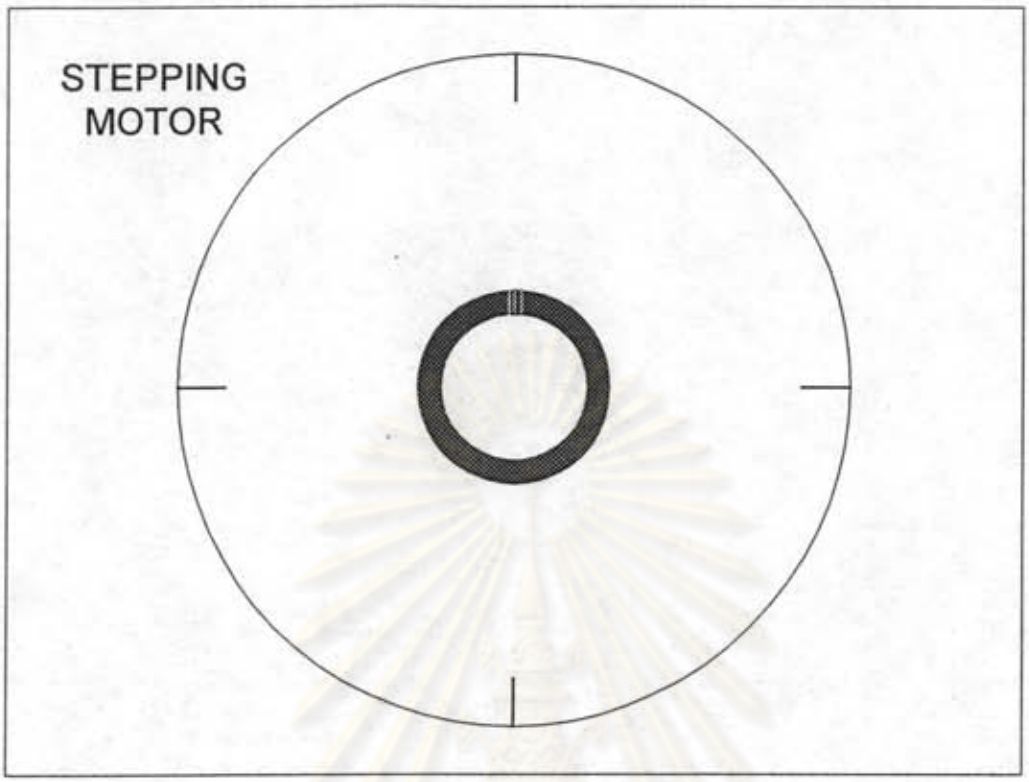





มอดูล
MOTOR CONTROL

7.10 <u>มอดูลสเตปปิงมอเตอร์</u>

ใช้ในการทดลองเรื่องการควบคุมสเตปปิงมอเตอร์ สเตปปิงมอเตอร์ที่ใช้มีขนาด 4 เฟส 12 โวลต์ มีค่าความต้านทานกระแสตรงของแต่ละเฟสเท่ากับ 33 โอห์ม จำนวนสเตป 200 สเตป (1.8 องศา/สเตป)




มอดูล STEPPING MOTOR

ขั้นตอนการทดสอบ

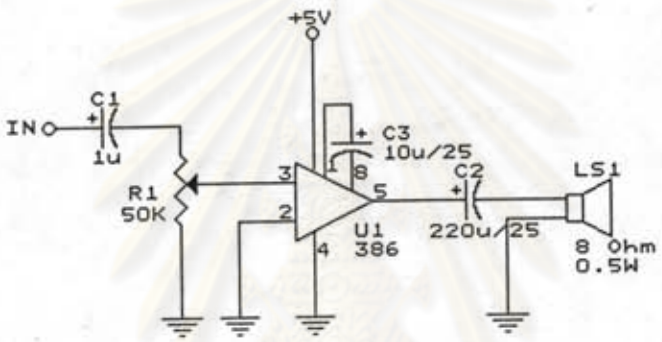
1. ต่อมอดูลสวิตช์เข้ากับเมนบอร์ด
2. ต่อมอดูลสเตปปิงมอเตอร์เข้ากับมอดูลสวิตช์ ที่ขั้ว TR Driver
3. ทดลองเขียนและรันโปรแกรมต่อไปนี้

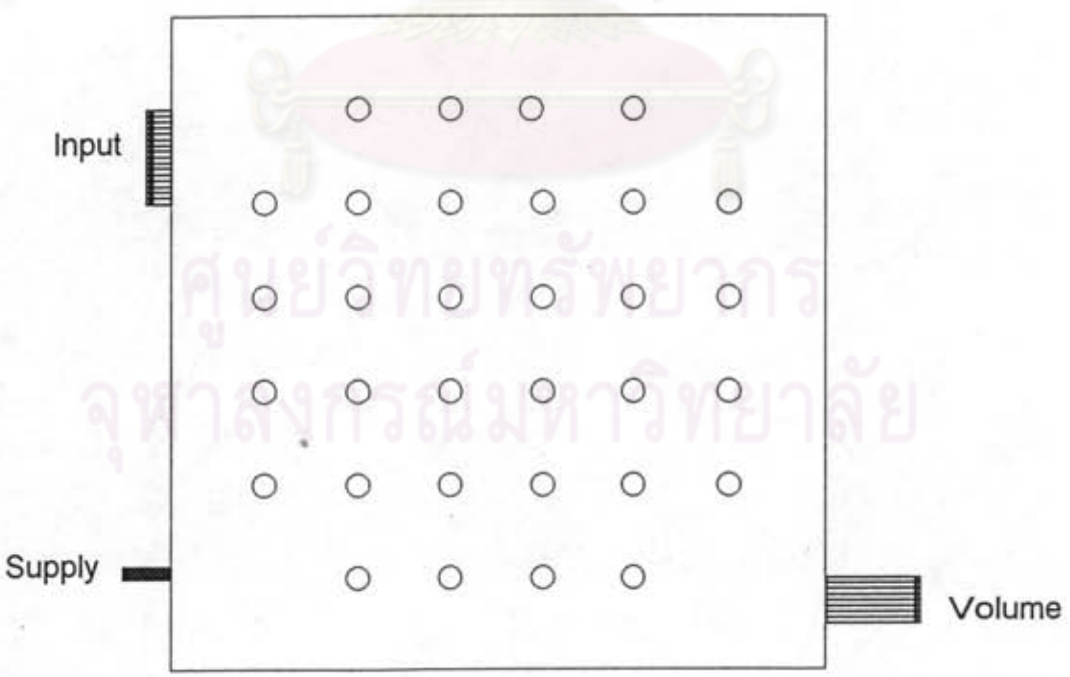
```
/*----------Program 10 : Stepping motor control-------*/
#include<edl.h>
#include<conio.h>
#define DRV 0xc2

main()
{
      int c;

      for(;;)
      {
            out(DRV,0x60);          /*2 phase excitation*/
            delay(100);
            out(DRV,0x50);
            delay(100);
            out(DRV,0x90);
            delay(100);
            out(DRV,0xa0);
```

```
            delay(100);
            if (kbhit()) if (c = getch() == 27) break;
        }
}
```

7.11 <u>มอดูล Audio Amplifier</u>

ใช้ในการทดสอบสัญญาณในย่านความถี่เสียงโดยใช้วิธีการฟัง ซึ่งจะใช้ในการทดลองเรื่องการสุ่มสัญญาณเสียงและเล่นกลับ และสามารถนำไปประยุกต์ใช้งานอื่นๆ เช่น การควบคุมลำโพง เป็นต้น



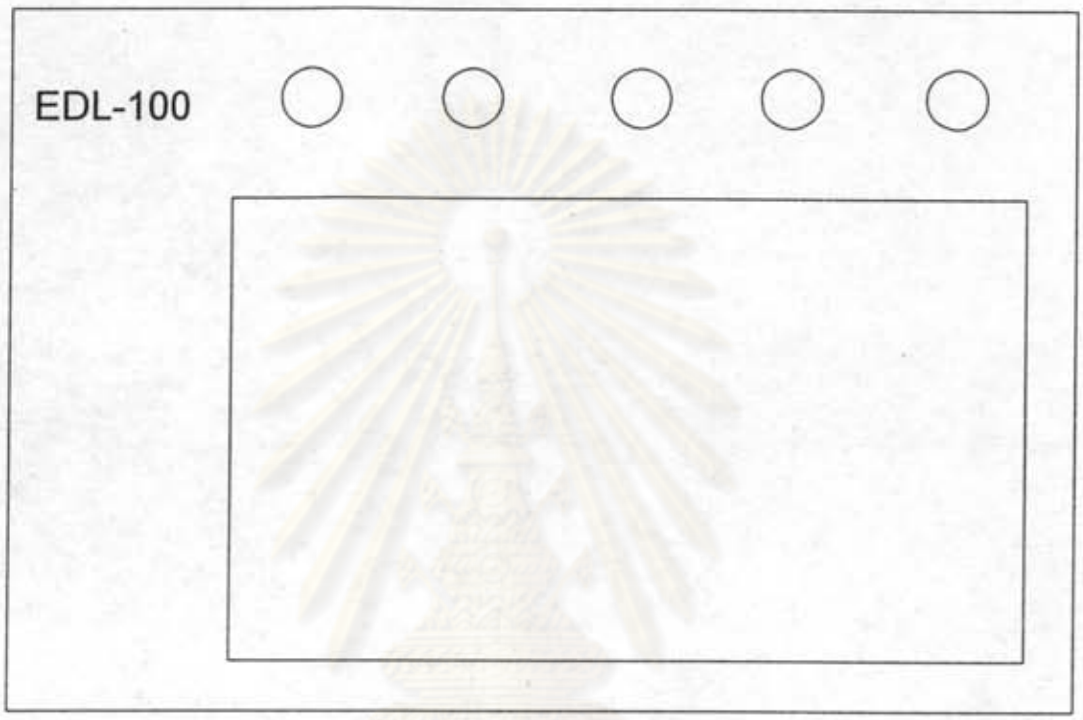




มอดูล

AUDIO AMPLIFIER

7.12 <u>โปรโตบอร์ด</u>

ใช้ในการต่อวงจรภายนอกเพิ่มเติมสำหรับแต่ละการทดลอง (ถ้ามี)




โปรโตบอร์ด

## 8. การใช้ TURBO C

### 8.1 การเรียกโปรแกรม TURBO C

1) เข้าสู่ไดเร็กทอรี TC

2) พิมพ์คำสั่ง

C : \TC > TC

### 8.2 การเรียกเมนู

การเรียกเมนู สามารถเรียกได้ โดยการกด F10 และใช้ลูกศรเลื่อนไปยังเมนูที่ต้องการ ซึ่งจะเป็นแบบ pull-down menu การเลือกเมนูทำโดยการเลื่อน Highlight ไปยังเมนูที่ต้องการแล้วกด Enter

| | |
|---|---|
| File | จัดการเกี่ยวกับไฟล์ เช่น การเรียกไฟล์ การบันทึก การสร้างไฟล์ใหม่ การเรียกดูชื่อไฟล์ การเปลี่ยนไดเร็กทอรี การออกไปยัง DOS การออกจากโปรแกรม |
| Edit | สร้างและแก้ไขเนื้อความของไฟล์(เรียกใช้เอดิเตอร์) |
| Run | ทำการคอมไพล์ ลิงค์ และรันโปรแกรมแบบอัตโนมัติ |
| Compile | คอมไพล์โปรแกรมให้อยู่ในรูป object file(.OBJ) และ executable file (.EXE) |
| Project | ใช้สร้างโปรแกรมที่ประกอบด้วยไฟล์หลายๆไฟล์ |
| Options | เลือกวิธีการคอมไพล์ เช่น memory model, compile-time, diagnostics, linker การกำหนดมาโคร การเลือกไดเร็กทอรีที่เก็บของ Include file, Output file, Library file และเรียกและบันทึกวิธีการคอมไพล์จาก Configuration file |
| Debug | การเรียกดู error และการจัดการกับ error message |
| Break/watch | การตั้ง breakpoint และการตรวจสอบค่าในตัวแปร |

### 8.3 คำสั่งการใช้แป้นพิมพ์

เอดิเตอร์ของ TURBO C มีความคล้ายคลึงกับ เอดิเตอร์ของ Turbo Pascal และ Sidekick โดยมีคำสั่งการใช้งานที่เกือบเหมือนกัน เอดิเตอร์เป็นอุปกรณ์ที่ใช้ในการเขียน Source Program ที่เป็น Text file

8.4 การคอมไพล์และรันโปรแกรม

กด ALT- C - คอมไพล์โปรแกรม ซึ่งจะได้ object file ที่มีนามสกุล .OBJ เมื่อทำการคอมไพล์และเกิด error ขึ้น ผู้ใช้สามารถเข้าไปแก้ไขโปรแกรมได้โดยการกด F6

กด F6 - แก้ไขโปรแกรม

กด F9 - คอมไพล์โปรแกรม และสร้าง executable file จะได้ไฟล์ที่มีนามสกุล .OBJ และ .EXE ซึ่งสามารถเรียกใช้งานได้จาก DOS Prompt

กด CTRL-F9 - รันโปรแกรม โปรแกรมจะทำการคอมไพล์ สร้าง executable file และรันโดยอัตโนมัติ

8.5 การทดสอบและแก้ไขโปรแกรม

TURBO C ได้เตรียมอรรถประโยชน์เพื่อใช้ในการทดสอบและแก้ไขโปรแกรม คือ การทำ Single Step การตั้ง breakjoint และการตรวจสอบค่าในตัวแปร

8.5.1 การทำ Single Step

กด F7 - ทำทีละคำสั่ง

กด F8 - ทำทีละฟังก์ชัน คือ ในกรณีที่เป็นฟังก์ชัน โปรแกรมจะกระโดดไปทำงานในฟังก์ชันจนจบฟังก์ชัน แล้วจึงหยุดทำงาน

กด CTRL-F2 - รีเซ็ตโปรแกรม ทำให้โปรแกรมกระโดดกลับไปเริ่มทำงานยังจุดเริ่มต้นของโปรแกรม

8.5.2 การตั้ง breakpoint

โดยการเลื่อน CURSOR ไปยังบรรทัดที่ต้องการให้โปรแกรมหยุดการทำงาน และกดปุ่มCTRL-F8 ซึ่งสามารถตั้งได้หลายๆ จุด การนำ breakpoint ออก ทำโดยการเลื่อน CURSOR ไปยังจุดที่ต้องการแล้วกด CTRL-F8 เช่นเดิม

กด CTRL-F8 - Toggle breakpoint

8.5.3 การตรวจสอบค่าในตัวแปร

ขณะอยู่ในเอดิเตอร์ ทำโดยการเลื่อน CURSOR ไปยังชื่อตัวแปรที่ต้องการตรวจสอบค่า และกด CTRL-F7 ถ้าไม่ต้องการอีก ให้ลบทิ้งโดยใช้ DEL

กด CTRL-F7 - Add Watch

8.6 การเรียก HELP

กด F1 - จะขึ้น HELP ของการใช้เอดิเตอร์ กด F1 อีกครั้งหนึ่ง จะขึ้น HELP INDEX แล้วจึงทำการเลือกหัวข้อที่ต้องการ

กด ALT-F1 - ย้อนกลับไปหน้าจอ HELP หน้าที่แล้ว

เราสามารถเรียกดูลักษณะการใช้งานของฟังก์ชันต่างๆ หรือ ค้นความหมายของคำต่างๆ ที่ใช้ในภาษา C เพื่อช่วยในการเขียนโปรแกรมได้ โดยการเรียกใช้ HELP ซึ่งมีวิธีการเรียกใช้ใน 2 วิธี คือ

1. การเลือกฟังก์ชันหรือคำจากเมนูชื่อ

กดปุ่ม CTRL-F1 จะขึ้นเมนูให้เลือก คือ Keyword และ Header Files เลือก Keyword เพื่อดูคำต่างๆ ที่เป็นคำสงวนของภาษา C การตั้งชื่อตัวแปรต่างๆในโปรแกรมจะต้องไม่ซ้ำกับคำเหล่านี้ เลือก Header Files จะปรากฏชื่อของ Header File ต่างๆ เลือกชื่อของ Header File ที่เก็บฟังก์ชันที่ต้องการจะค้นหา หลังจากนั้นเลือกฟังก์ชันนั้น ๆ ซึ่ง HELP จะให้รายละเอียดต่างๆในการใช้งาน วิธีนี้ผู้ใช้จะต้องทราบว่าฟังก์ชันที่ต้องการค้าหานั้น อยู่ใน Header File อะไร

2. การเลือกฟังก์ชันหรือคำโดยตรง

เมื่อรู้คำหรือชื่อฟังก์ชันที่ต้องการจะค้นหา ให้พิมพ์คำนั้นๆ ณ ตำแหน่งใดก็ได้ของเอดิเตอร์ โดยจะต้องไม่ติดกับชื่ออื่นๆ หลังจากนั้นเลื่อนเคอร์เซอร์ให้ไปอยู่ ณ ตำแหน่งของคำนั้นๆ แล้วกดปุ่ม CTRL-F1 TURBO C ก็จะแสดงรายละเอียดต่างๆของคำหรือฟังก์ชันนั้นๆ ถ้าชื่อนั้นไม่มีในไลบรารีฟังก์ชันของ TURBO C ก็จะถูกรายงานผลให้ทราบ

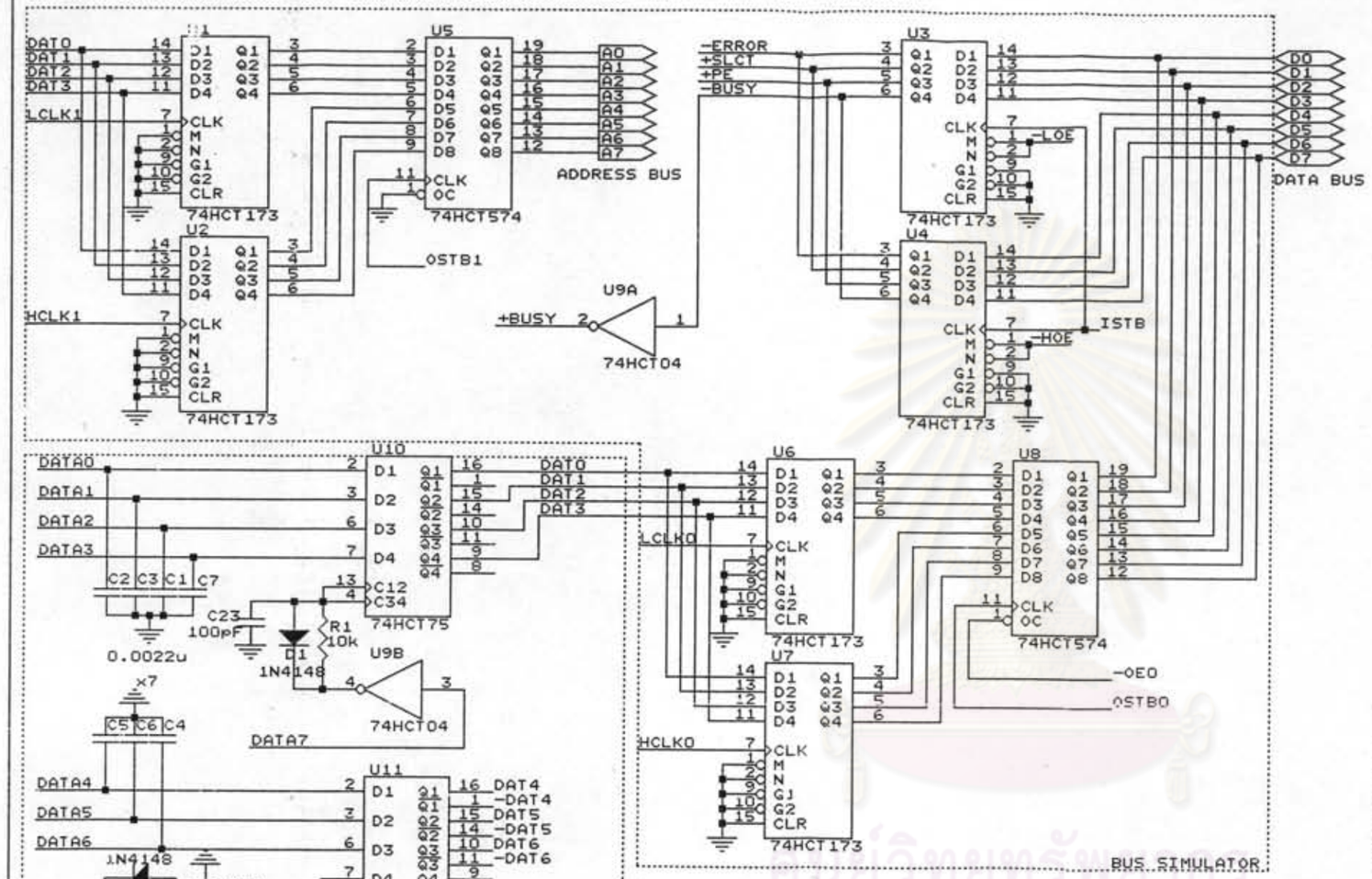
รายละเอียดการใช้ TURBO C ยังมีอีกมาก ผู้ใช้สามารถอ่านเพิ่มเติมได้จากหนังสือ TURBO C Reference Guide [20] และ TURBO C User's Guide[19]

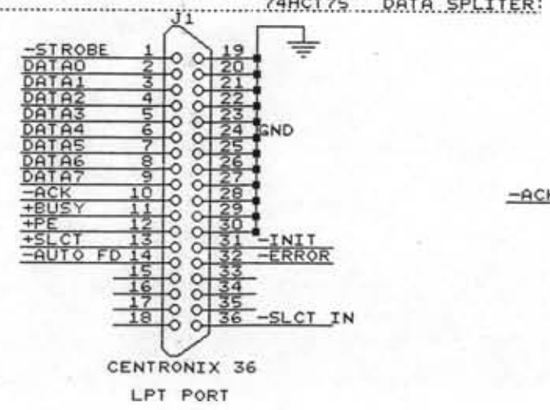
9. SYSTEM I/O MAP

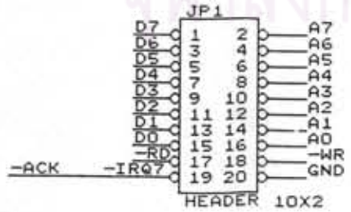
| ADDRESS | TYPE | DEVICE | ADDRESS | TYPE | DEVICE |
|---|---|---|---|---|---|
| C0h | output | LED × 8 | F0h | output | T/C Control Register 1 |
| C1h | input | Switch × 8 | F1h | output | T/C Control Register 2 |
| C2h | output | Relay & TR Driver | F2h | output | Clear T/C interrupt |
| C4h | output | A/D Control Register | F3h | input | T/C Status Register |
| C5h | input | A/D Data | F4h | input/ output | 8253 Counter Register 0 |
| C6h | output | D/A Data | F5h | input / output | 8253 Counter Register 1 |
| C8h | output | PortA | F6h | input / output | 8253 Counter Register 2 |
| C9h | output | PortB | F7h | output | 8253 Control Word |
| CAh | input | PortC | | | |
| CBh | input | PortD | | | |

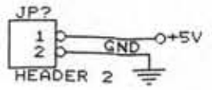
# ภาคผนวก จ

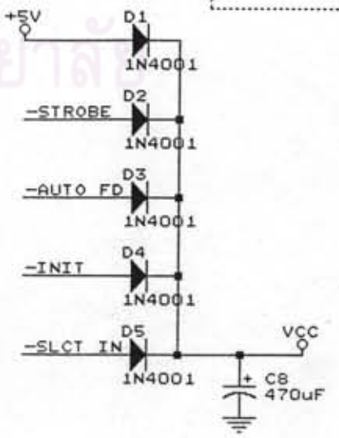
## แผนภาพวงจรของชุดฝึกทดลอง

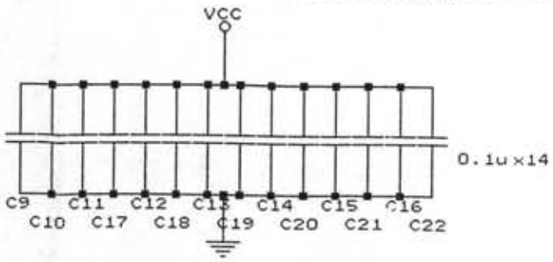


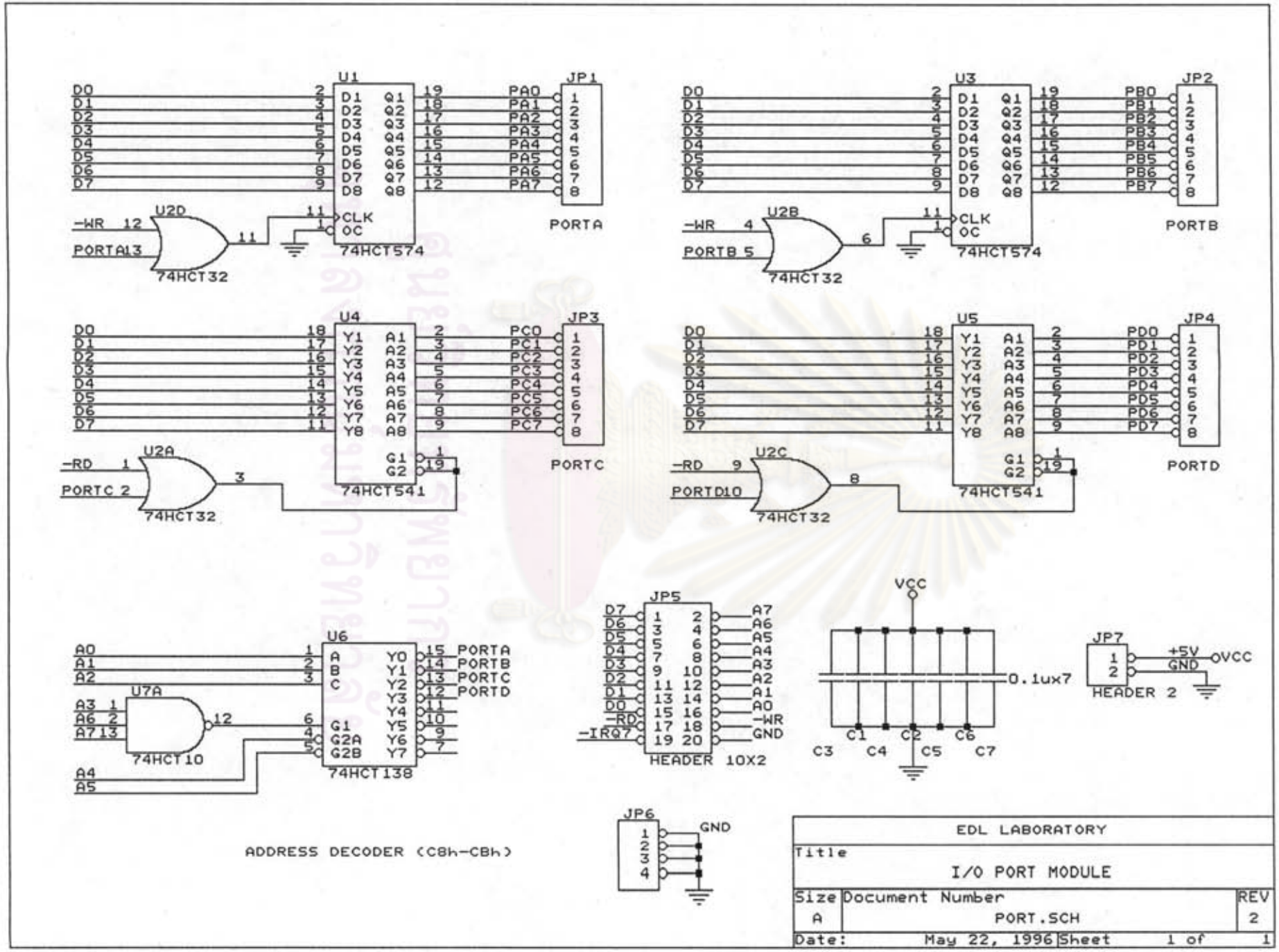



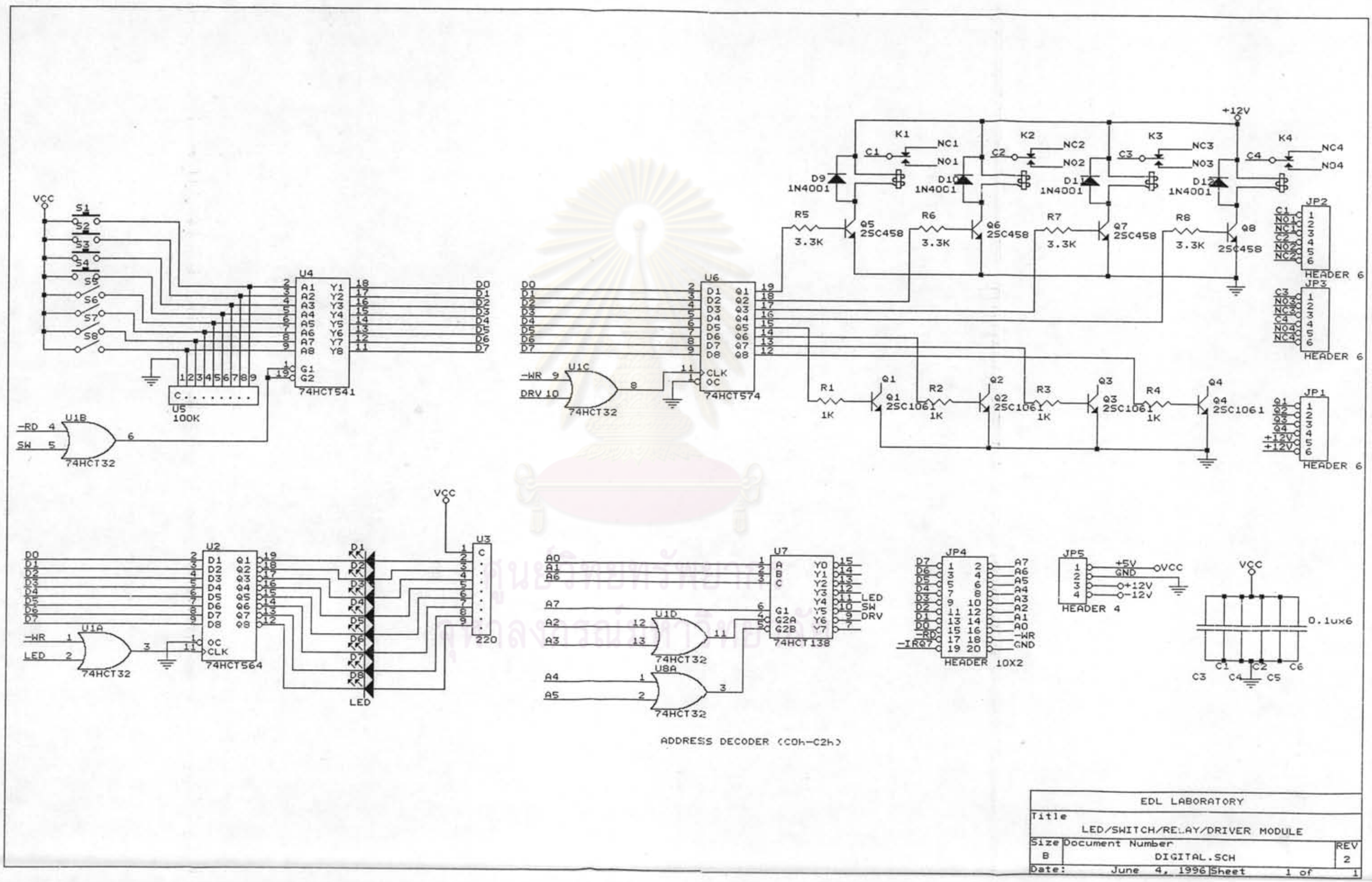



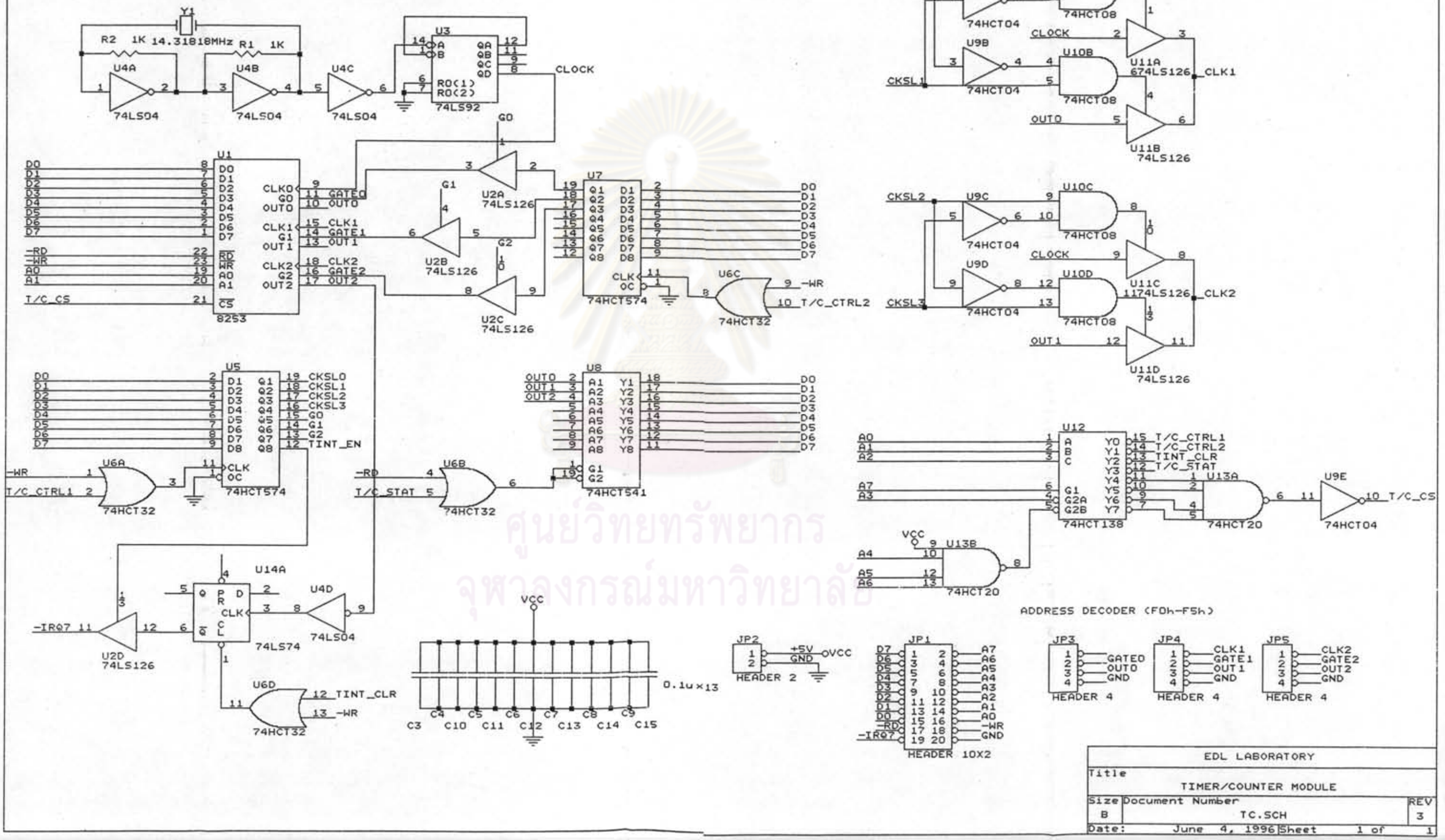

| | | |
|---|---|---|
| EDL LABORATORY | | |
| Title: MAINBOARD | | |
| Size: B | Document Number: MAIN.SCH | REV: 3 |
| Date: May 22, 1996 | Sheet 1 of 1 | |

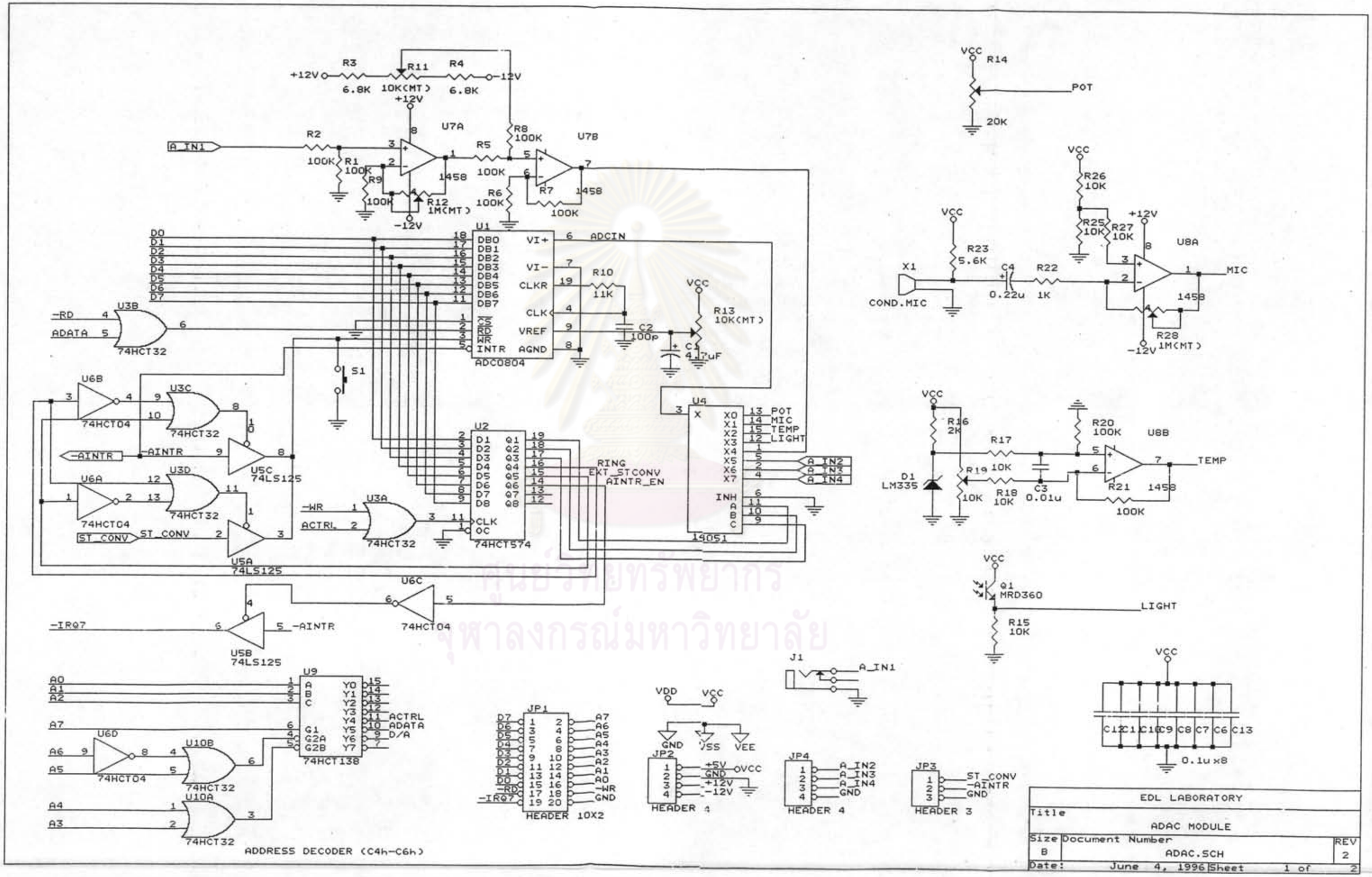

U1
JP1
74HCT574
74HCT32
PORTA
U3
JP2
PORTB
U4
JP3
74HCT541
PORTC
U5
JP4
PORTD
U6
74HCT138
U7A
74HCT10
JP5
HEADER 10X2
VCC
0.1ux7
JP7
HEADER 2
+5V
GND
JP6
ADDRESS DECODER (C8h-CBh)
EDL LABORATORY
Title
I/O PORT MODULE
Size A
Document Number
PORT.SCH
REV 2
Date: May 22, 1996
Sheet 1 of 1

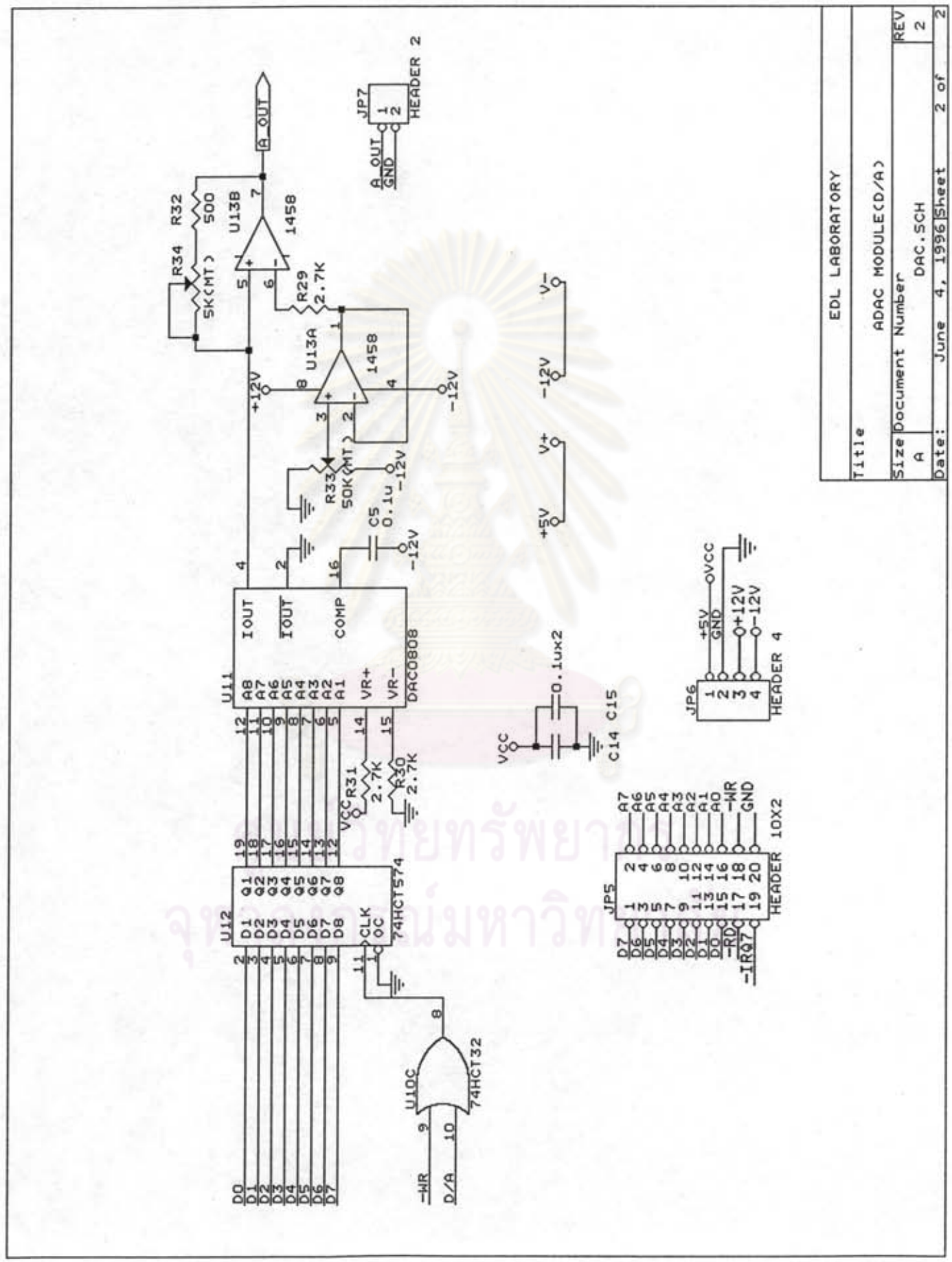


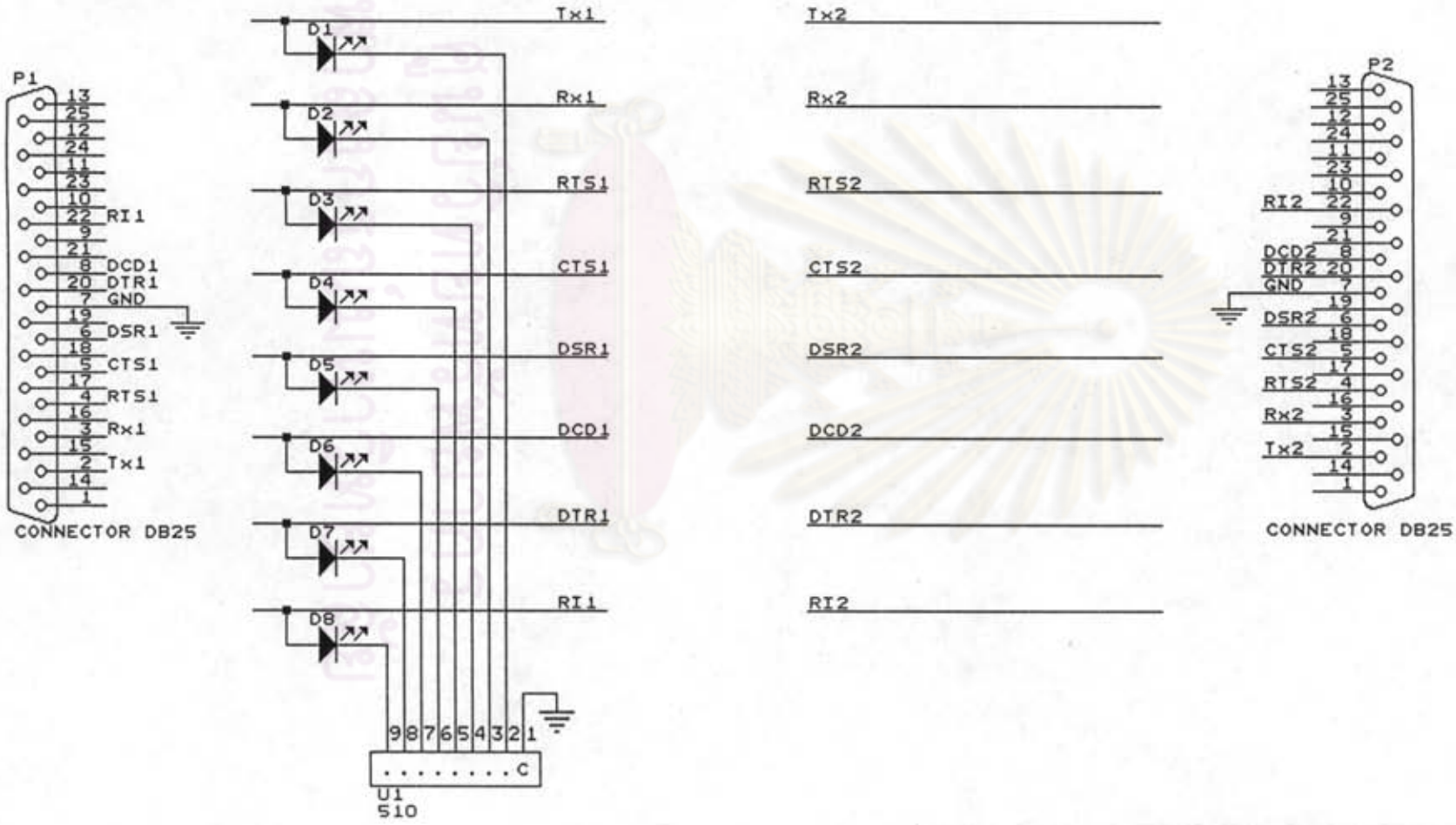

ADDRESS DECODER (C0h-C2h)

| | | |
|---|---|---|
| EDL LABORATORY | | |
| Title: LED/SWITCH/RELAY/DRIVER MODULE | | |
| Size: B | Document Number: DIGITAL.SCH | REV: 2 |
| Date: June 4, 1996 | Sheet 1 of 1 | |

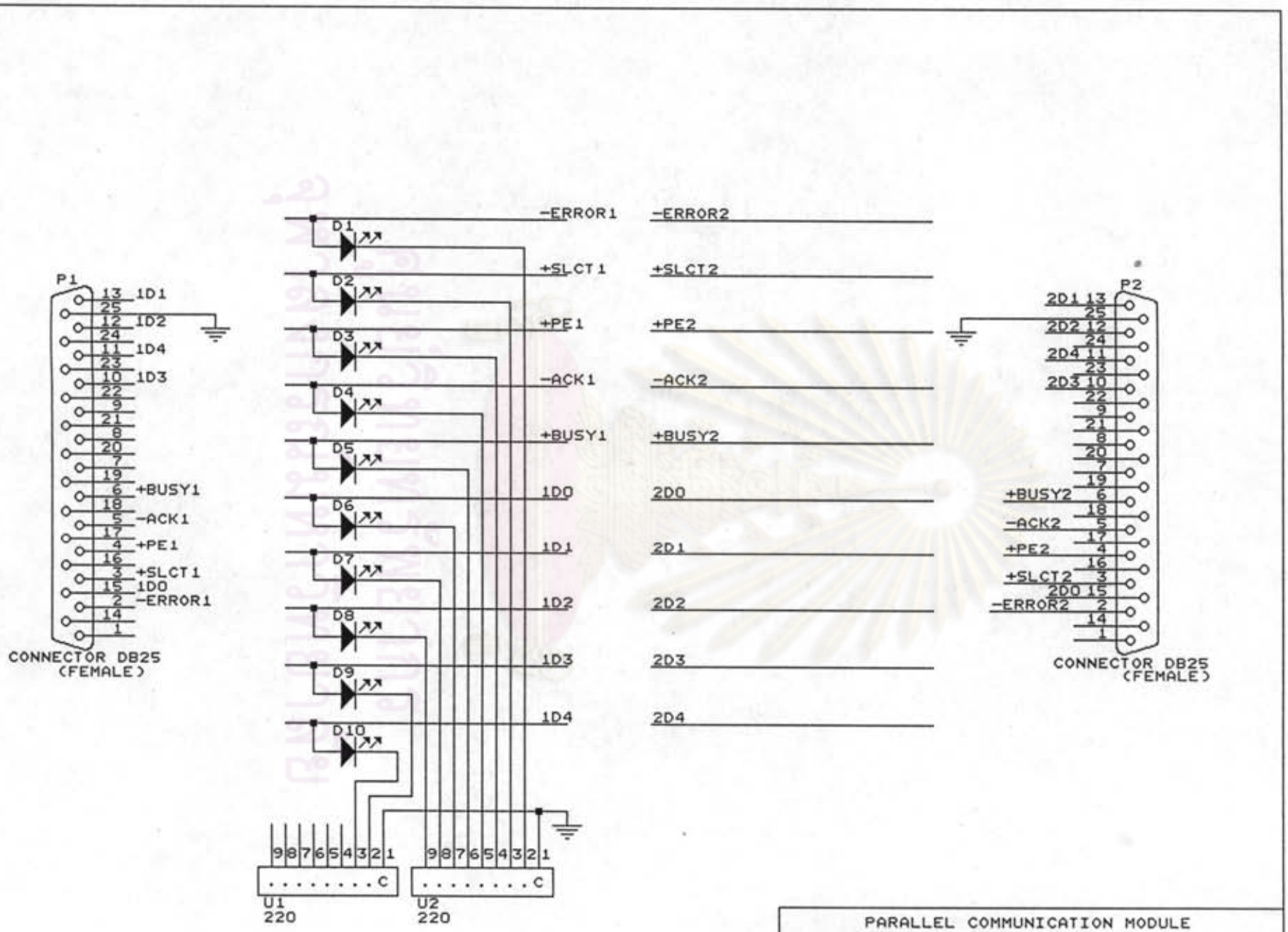


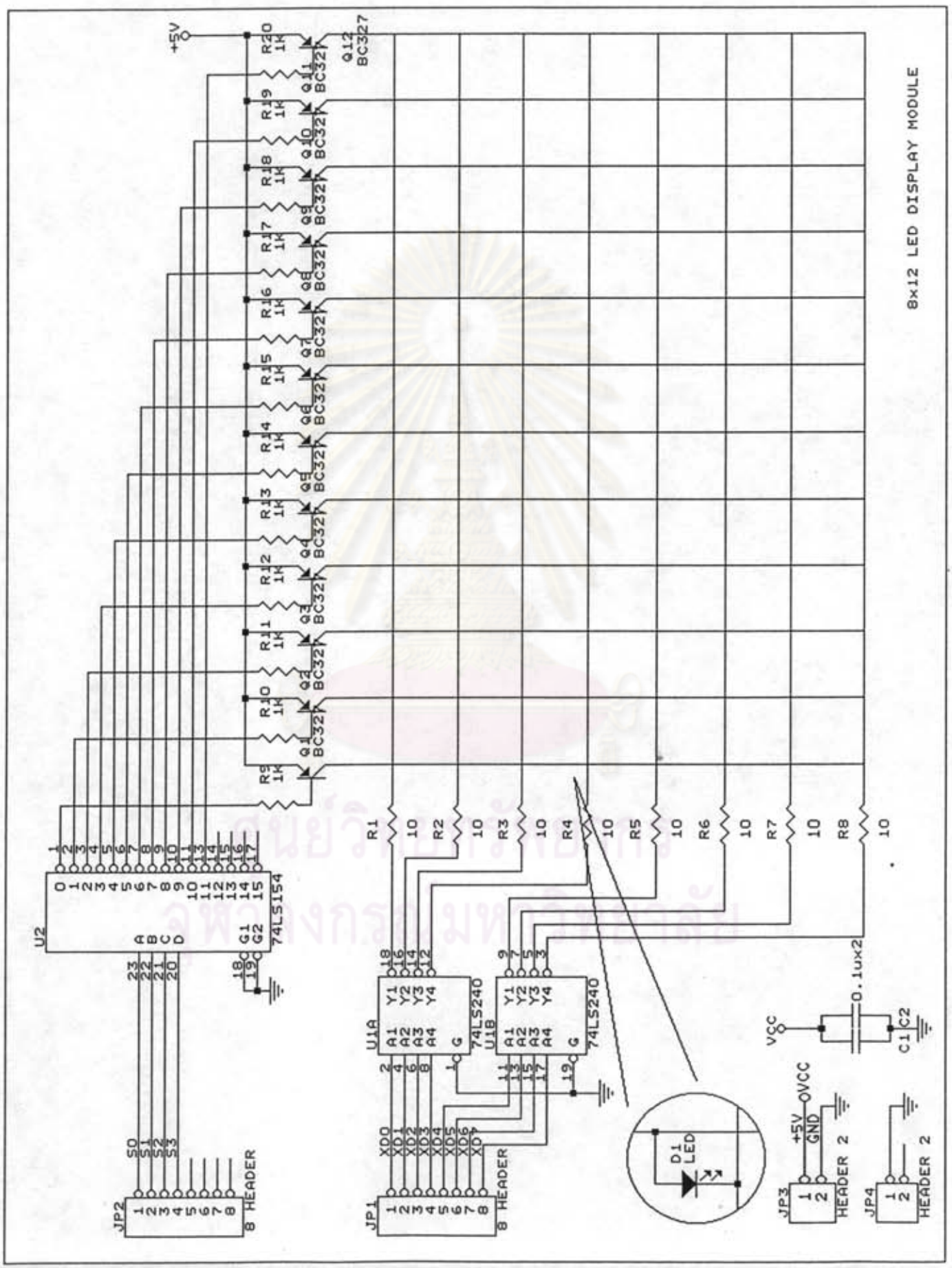


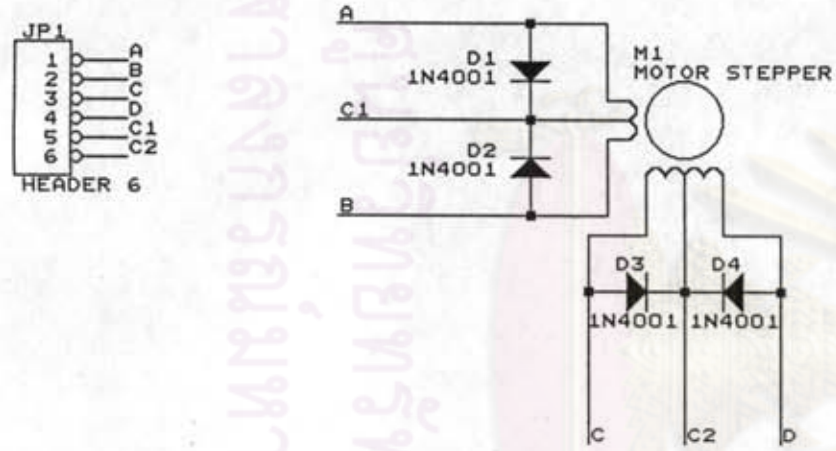

EDL LABORATORY
Title
ADAC MODULE(D/A)
Size
A
Document Number
DAC.SCH
REV
2
Date:
June 4, 1996
Sheet
2 of 2
A OUT
HEADER 2
JP7
A OUT
GND
U13B
1458
R32
500
R34
5K(MT)
R29
2.7K
U13A
1458
R33
50K(MT)
C5
0.1u
+12V
-12V
+5V
V+
V-
U11
DAC0808
IOUT
IOUT
COMP
VR+
VR-
R31
2.7K
R30
2.7K
U12
74HCT574
CLK
OC
U10C
74HCT32
-WR
D/A
D0
D1
D2
D3
D4
D5
D6
D7
C14 C15
0.1ux2
VCC
JP6
HEADER 4
JP5
HEADER 10X2
A7
A6
A5
A4
A3
A2
A1
A0
-WR
GND
-RD
-IRQ7



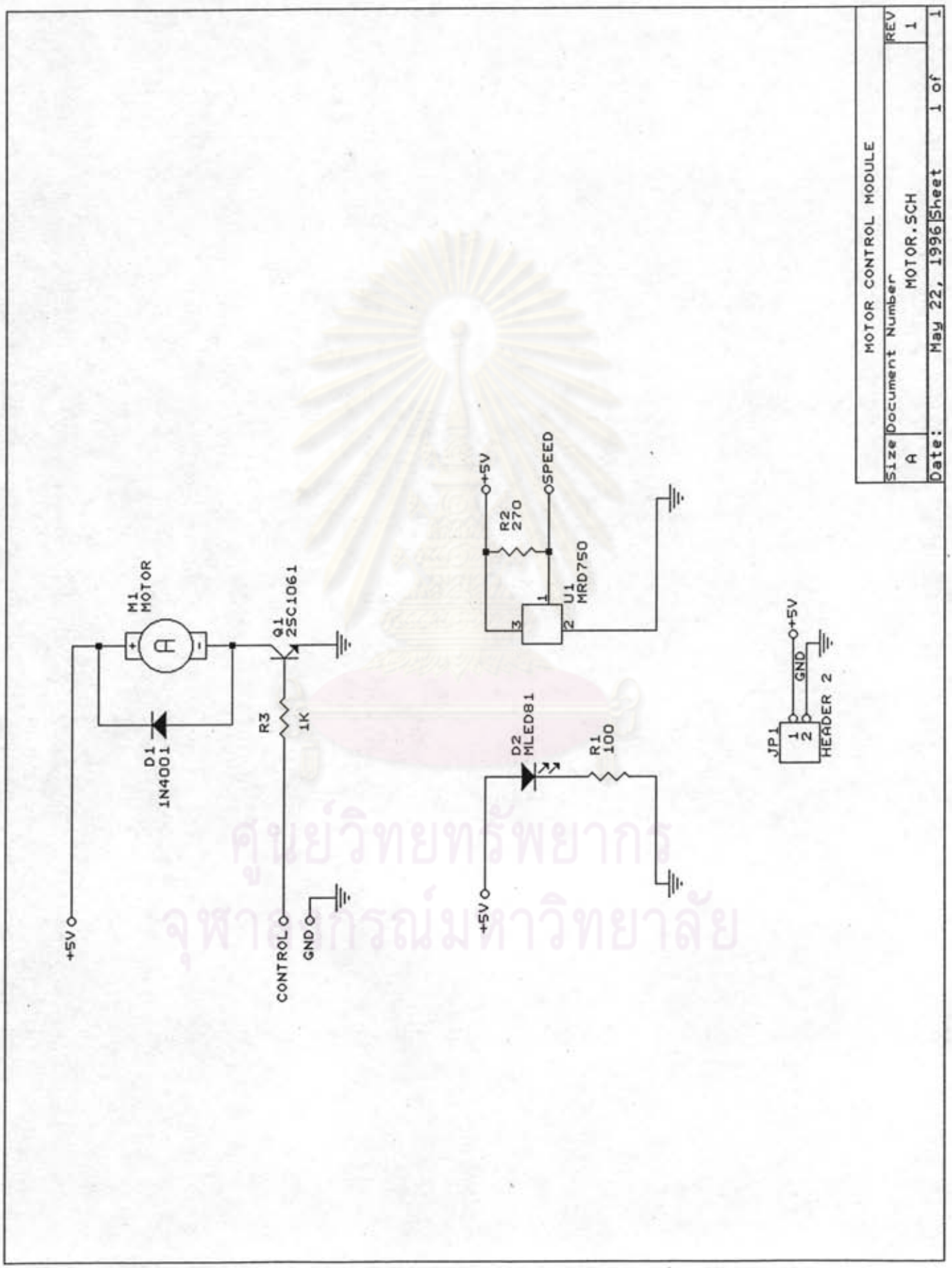

| SERIAL COMMUNICATION MODULE | | |
|---|---|---|
| Size A | Document Number SERIAL.SCH | REV 2 |
| Date: May 22, 1996 | Sheet 1 of 1 | |

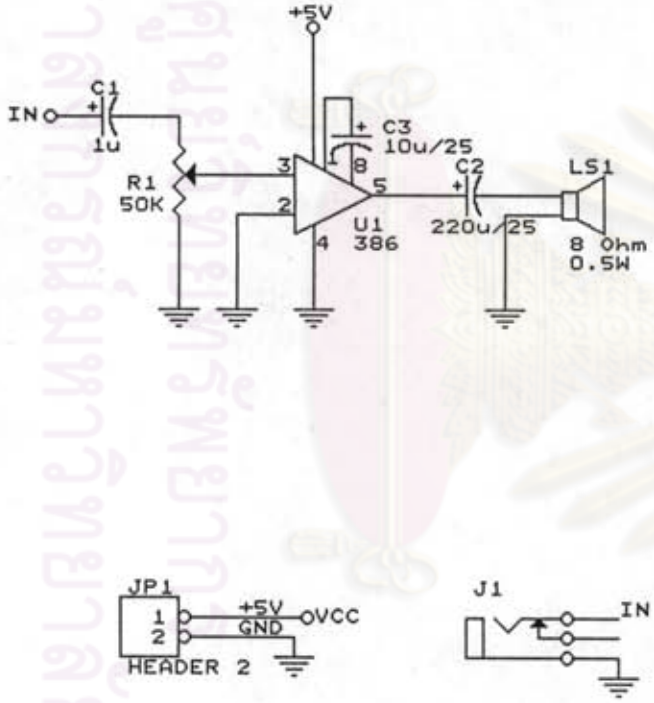

P1
13 1D1
25
12 1D2
24
11 1D4
23
10 1D3
22
9
21
8
20
7
19
6 +BUSY1
18
5 -ACK1
17
4 +PE1
16
3 +SLCT1
15 1D0
2 -ERROR1
14
1
CONNECTOR DB25
(FEMALE)
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
-ERROR1
+SLCT1
+PE1
-ACK1
+BUSY1
1D0
1D1
1D2
1D3
1D4
9 8 7 6 5 4 3 2 1
C
U1
220
9 8 7 6 5 4 3 2 1
C
U2
220
-ERROR2
+SLCT2
+PE2
-ACK2
+BUSY2
2D0
2D1
2D2
2D3
2D4
P2
2D1 13
25
2D2 12
24
2D4 11
23
2D3 10
22
9
21
8
20
7
19
+BUSY2 6
18
-ACK2 5
17
+PE2 4
16
+SLCT2 3
2D0 15
-ERROR2 2
14
1
CONNECTOR DB25
(FEMALE)
PARALLEL COMMUNICATION MODULE
Size A
Document Number
PARALLEL.SCH
REV 2
Date: June 4, 1996
Sheet 1 of 1


8x12 LED DISPLAY MODULE
+5V
U2
74LS154
JP1
JP2
JP3
JP4
8 HEADER
HEADER 2
U1A
U1B
74LS240
D1
LED
VCC
GND
0.1ux2
C1
C2




| STEPPING MOTOR MODULE | | |
|---|---|---|
| Size: A | Document Number: STEPPER.SCH | REV: 1 |
| Date: May 22, 1996 | Sheet 1 of 1 | |






| EDL LABORATORY | | |
|---|---|---|
| Title: AUDIO AMPLIFIER MODULE | | |
| Size: A | Document Number: AMP.SCH | REV: 1 |
| Date: May 22, 1996 | Sheet 1 of 1 | |

# ประวัติผู้เขียน

นายทวีชัย เจริญเศรษฐศิลป์ เกิดวันที่ 7 เมษายน 2514 ณ อำเภอบางรัก กรุงเทพมหานคร สำเร็จการศึกษาปริญญาวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมไฟฟ้า ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ปีการศึกษา 2534 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาและภาควิชาเดียวกัน ของคณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อ พ.ศ. 2535