



**ภาควิชาครุศาสตร์วิศวกรรม**

**คณะครุศาสตร์อุตสาหกรรม**

**สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง**

**ใบรับรองปริญญานิพนธ์**

**ชื่อหัวข้อ** ชุดสาธิตการทำงาน PLC ควบคุมด้วยไมโครโปรเซสเซอร์

PLC Lab Control with Microprocessor

**ชื่อนักศึกษา**

| | | | |
|---|---|---|---|
| 1. นายทองดี | ศรีเจริญ | รหัสประจำตัว | 41031312 |
| 2. นายพงศกร | พรหมเจริญ | รหัสประจำตัว | 41031320 |
| 3. นายวิชิต | เวียงคำ | รหัสประจำตัว | 41031329 |

**หลักสูตร** ครุศาสตร์อุตสาหกรรมบัณฑิต **สาขาวิชา** อิเล็กทรอนิกส์และคอมพิวเตอร์

**อาจารย์ที่ปรึกษา** อาจารย์อมรชัย ชัยชนะ

**อาจารย์ที่ปรึกษาร่วม** อาจารย์สุรพงษ์ สิริพงศ์ดี

| คณะกรรมการสอบปริญญานิพนธ์ | | ลายมือชื่อ |
|---|---|---|
| 1. อาจารย์อมรชัย | ชัยชนะ | ..... |
| 2. อาจารย์กิติพงศ์ | มะโน | ..... |
| 3. อาจารย์สุชิน | อาจหาญ | ..... |
| 4. อาจารย์อำพล | ทองระอา | ..... |
| 5. อาจารย์ไพบูลย์ | พวงวงศ์ตระกูล | ..... |

**วัน/เดือน/ปีที่สอบ** วันเสาร์ที่ 13 พฤษภาคม พ.ศ. 2543 เวลา 20.30 น.

**สถานที่สอบ** ห้อง ค.301 คณะครุศาสตร์อุตสาหกรรม สจล.

ภาควิชารับรองแล้ว

ลงนาม.....

(ผศ.วิสุทธิ์ อธิพรธรรม)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

วันที่ 10 เดือน พ.ค. พ.ศ. 2543

# ปริญญานิพนธ์

## ชุดสาธิตการทำงาน PLC ควบคุมด้วยไมโครโปรเซสเซอร์

## PLC LAB CONTROL WITH MICROPROCESSOR

นายทองดี ศรีเจริญ
นายพงศกร พรหมเจริญ
นายวิชิต เวียงคำ

ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาทางหลักสูตรครุศาสตร์อุตสาหกรรมบัณฑิต
สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์
ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2542

# ปริญญานิพนธ์

**เรื่อง** ชุดสาธิตการทำงาน PLC ควบคุมด้วยไมโครโปรเซสเซอร์
PLC LAB CONTROL WITH MICROPROCESSOR

**วัตถุประสงค์**

1. เพื่อศึกษาการใช้งานจริงของไมโครคอนโทรลเลอร์ไปประยุกต์ใช้งานจริง
2. เพื่อศึกษาการทำงานของมอเตอร์กระแสตรง และมอเตอร์กระแสสลับ
3. เพื่อศึกษาการเขียนโปรแกรม PLC
4. เพื่อออกแบบและสร้างชุดทดลองสำหรับศึกษาในวิชาเรียน
5. เพื่อนำชุดทดลองไปทดลองสำหรับศึกษาในชั้นเรียน
6. เพื่อนำความรู้ที่ได้จากโครงงานนี้ไปประยุกต์ใช้งาน

**ประโยชน์ที่คาดว่าจะได้รับ**

1. เข้าใจทฤษฎีและหลักการทำงานของไมโครคอนโทลเลอร์มาประยุกต์ใช้งานจริงได้
2. เข้าใจการทำงานของมอเตอร์กระแสตรง และ มอเตอร์กระแสสลับ
3. เข้าใจการเขียนโปรแกรมควบคุมผ่านทาง PC ได้
4. สามารถเขียนโปรแกรม PLC แบบบูลลีน และ แบบแลดเดอร์ได้
5. สามารถออกแบบและสร้างชุดเชื่อมต่อกับชุดทดลอง PLC ได้
6. สามารถนำโครงงานนี้ไปใช้เป็นสื่อในการเรียนการสอนได้จริง

| | |
|---|---|
| **ชื่อหัวข้อ** | ชุดสาธิตการทำงาน PLC ควบคุมด้วยไมโครโปรเซสเซอร์ |
| **นักศึกษา** | นายทองดี ศรีเจริญ |
| | นายพงศกร พรหมเจริญ |
| | นายวิชิต เวียงคำ |
| **อาจารย์ที่ปรึกษา** | อาจารย์อมรชัย ชัยชนะ |
| **อาจารย์ที่ปรึกษาร่วม** | อาจารย์สุรพงษ์ สิริพงศ์ดี |
| **หลักสูตร** | ครุศาสตร์อุตสาหกรรมบัณฑิต |
| **สาขาวิชา** | อิเล็กทรอนิกส์และคอมพิวเตอร์ |
| **ปีการศึกษา** | 2542 |


## บทคัดย่อ

ชุดสาธิตการทำงาน PLC ได้ออกแบบขึ้นมาเพื่อใช้ในการศึกษา และทดลอง การใช้ PLC โดยออกแบบให้มีลักษณะเหมือน Single Board ซึ่งผู้ใช้สามารถ ทำความเข้าใจ และพัฒนาโปรแกรมได้สะดวกและชัดเจน

โดยชุดสาธิตการทำงาน PLC จะประกอบด้วย ตัวชุดสาธิตการทำงาน PLC และตัวอินเตอร์เฟส ซึ่ง สามารถใช้เป็นชุดทดลองให้นักศึกษาเรียนรู้ เกี่ยวกับการใช้งาน PLC มาควบคุมไฟวิ่ง , ควบคุม DC Motor , ควบคุม Stepping Motor , และไฟจราจร

| | |
|---|---|
| **Thesis Title** | PLC LAB Control with Microprocessor |
| **Students** | Mr.Thongdi Srichareon |
| | Mr.Pongsakorn Promcharoen |
| | Mr.Wichit Wiangkhum |
| **Advisor** | Mr.Amornchai Chaichana |
| **Co-Advisor** | Mr.Surapong Siriponmgdee |
| **Education Level** | Bachelor of Science in Industrial Education |
| **Program in** | Electronics and Computer |
| **Academic Year** | 1999 |


## ABSTRACT


PLC Lab Set is designed for studying and experimenting. This thesis has been designed like a single board set that users can study , understand and develop it conveniently And elearly.

The content of the thesis is about PLC Demonstrating Set and Interface which can be used as Training Set for the students study in using PLC controls Running Light, DC Motor,Stepping Motor and Traffic Light.

# กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ลุล่วงไปได้ด้วยดี เนื่องจากความร่วมมือของสมาชิกภายในกลุ่มทุกท่าน ขอขอบคุณอาจารย์อมรชัย ชัยชนะ และท่านอาจารย์สุรพงษ์ สิริพงศ์ดี และ คณาจารย์ภาควิชาครุศาสตร์วิศวกรรมทุกท่านที่ให้ความอนุเคราะห์เครื่องมือ และอุปกรณ์ รวมทั้งให้คำแนะนำแนวความคิด ความรู้ต่างๆ แนวทางแก้ไขปัญหาในการทำปริญญานิพนธ์ ขอขอบตุณห้องสมุดคณะครุศาสตร์อุตสาหกรรม ที่ช่วยอำนวยความสะดวกและเอื้อเพื้อ สถานที่ในการค้นคว้าข้อมูล สุดท้ายนี้ขอขอบคุณ บิดา มารดาที่เป็นผู้ให้ความสนับสนุนด้านการศึกษา และเป็นผู้ให้กำลังใจมาด้วยดีมาตลอด ตั้งแต่อดีตจนถึงปัจจุบัน

# สารบัญ


| เรื่อง | หน้า |
|---|---|
| บทคัดย่อภาษาไทย | I |
| บทคัดย่อภาษาอังกฤษ | II |
| กิตติกรรมประกาศ | III |
| สารบัญ | IV |
| สารบัญตาราง | VIII |
| สารบัญรูป | IX |
| บทที่ 1บทนำ | 1 |
| 1.1 ความเป็นมาและความสำคัญของปริญญานิพนธ์ | 1 |
| 1.2 ขีดความสามารถของโครงงาน | 2 |
| 1.3 เนื้อหาโดยสังเขป | 2 |
| บทที่ 2 ทฤษฎีและหลักการ | 2 |
| 2.1 กล่าวนำ | 4 |
| 2.2 ไมโครโปรเซสเซอร์ Z80180 | 4 |
| 2.2.1 ขาที่ใช้งาน | 5 |
| 2.2.2 Internal I/O Register | 8 |
| 2.2.3 Operation Modes | 12 |
| 2.2.4 Timing | 15 |
| 2.2.5 Wait State Generator | 16 |
| 2.2.6 Halt และ Low Power Mode | 18 |
| 2.2.7 Memory Management Unit (MMU) | 18 |
| 2.2.8 Interrupt | 24 |
| 2.2.9 Dynamic RAM Refresh Control | 26 |
| 2.2.10 DMA Mode Register (DMOD Address I/O H) | 29 |
| 2.2.11 Clock Serial I/O Port (CSI/O) | 39 |
| 2.2.12 Programmable Reload Timer (PRT) | 40 |
| 2.2.13 Secondary Bus Interface | 42 |
| 2.2.14 Free Running Counter | 42 |

# สารบัญ ( ต่อ )


| เรื่อง | หน้า |
|---|---|
| 2.2.15 คำสั่งเพิ่มเติม 9 คำสั่ง | 43 |
| 2.3 Real Time Clock | 46 |
| 2.3.1 การจัดขาและหน้าที่ของขาต่างๆ | 46 |
| 2.3.2 รีจิสเตอร์ต่างๆ | 47 |
| 2.4 Liquid Crystal Display | 52 |
| 2.4.1 โครงสร้างของ LCD | 52 |
| 2.4.2 LCD แบบ Nemetic ชนิดเกลียว | 53 |
| 2.4.3 การแสดงผลแบบต่างๆ ของ LCD | 55 |
| 2.4.4 สีสันของ LCD แบบ TN | 56 |
| 2.4.5 คุณสมบัติทางแสง | 57 |
| 2.4.6 มุมมอง | 57 |
| 2.4.7 ความเร็วของการแสดงผล | 58 |
| 2.4.8 การขับ LCD | 58 |
| 2.5 มาตรฐาน RS 232C | 59 |
| 2.5.1 ลักษณะของสัญญาณ RS 232C | 60 |
| 2.5.2 การกำหนดจุดต่อของ RS 232C | 61 |
| 2.5.3 มาตรฐาน RS 232C กับ V.24 | 64 |
| 2.6 สเต็ปเปอร์มอเตอร์ | 66 |
| 2.6.1 ชนิดของสเต็ปเปอร์มอเตอร์ | 66 |
| 2.6.2 การพันขดลวดแบบสเต็ปเปอร์มอเตอร์ | 67 |
| 2.6.3 การกระตุ้นและการควบคุมการหมุนของสเต็ปเปอร์มอเตอร์ | 68 |
| บทที่ 3 การออกแบบ การสร้าง และการทำงาน | 71 |
| 3.1 กล่าวนำ | 71 |
| 3.2 การออกแบบวงจร | 71 |
| 3.2.1 วงจรควบคุม | 71 |
| 3.2.2 หน่วยความจำ | 72 |
| 3.2.3 ส่วนของวงจรรับข้อมูล | 73 |

# สารบัญ ( ต่อ )


| เรื่อง | หน้า |
|---|---|
| 3.2.4 ส่วนของการแสดงผล | 74 |
| 3.2.5 ส่วนการ Interface | 76 |
| 3.2.6 ส่วนการสื่อสารข้อมูลแบบอนุกรม | 77 |
| บทที่ 4 การทดลองและผลการทดลองผลการทดลองและทดสอบ | 78 |
| 4.1 การทดลองส่วนของการแสดงผลของเอาต์พุทของชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ | 78 |
| 4.1.1 ลำดับการทดลอง | 78 |
| 4.1.2 ผลการทดลอง | 79 |
| 4.2 การทดลองชุดรีเลย์และสวิทช์ | 80 |
| 4.2.1 ลำดับการทดลอง | 80 |
| 4.2.2 ผลการทดลอง | 82 |
| 4.3 การทดลองส่วนของชุด DC Motor | 82 |
| 4.3.1 ลำดับการทดลอง | 82 |
| 4.3.2 ผลการทดลอง | 84 |
| 4.4 การทดลองส่วนของชุด Stepping Motdr | 84 |
| 4.4.1 ลำดับการทดลอง | 84 |
| 4.4.2 ผลการทดลอง | 85 |
| 4.5 การทดลองส่วนของชุดการควบคุมไฟจราจรทางมาลาย | 86 |
| 4.5.1 ลำดับการทดลอง | 86 |
| 4.5.2 ผลการทดลอง | 89 |
| 4.6 การทดลองส่วนของการ รับส่งข้อมูลระหว่างคอมพิวเตอร์กับชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ | 89 |
| 4.6.1 ลำดับการทดลอง | 89 |
| 4.6.2 ผลการทดลอง | 90 |
| บทที่ 5 สรุปปัญหาแนวทางแก้ไขและการพัฒนาโครงงาน | 92 |
| 5.1 บทสรุป | 92 |
| 5.2 ปัญหาและแนวทางแก้ปัญหา | 92 |

# สารบัญ ( ต่อ )


| เรื่อง | หน้า |
|---|---|
| 5.3 แนวทางการพัฒนาโครงงาน | 93 |
| ภาคผนวก ก เครื่องต้นแบบ | 94 |
| ภาคผนวก ข วงจร และแผ่นวงจรพิมพ์ | 99 |
| ภาคผนวก ค ผังงานและโปรแกรมการทำงาน | 107 |
| ภาคผนวก ง ใบงานการทดลอง | 230 |
| ภาคผนวก จ รายละเอียดและคุณสมบัติของอุปกรณ์ | 288 |
| ภาคผนวก ฉ คู่มือการใช้งาน | 299 |
| บรรณานุกรม | 322 |
| ประวัติผู้แต่ง | 328 |

# สารบัญตาราง


| ตาราง | | หน้า |
|---|---|---|
| ตารางที่ 2.1 | สถานะการทำงานของ CPU | 7 |
| ตารางที่ 2.2 | Internal I/O Register | 8 |
| ตารางที่ 2.3 | แสดงช่วง Address I/O ของภาค Control IOA7 & IOA6 | 11 |
| ตารางที่ 2.4 | การ Set ค่า MWI1, MWI0 | 17 |
| ตารางที่ 2.5 | การ Set ค่า IWI1, IWI0 | 17 |
| ตารางที่ 2.6 | Combination of Transfer Mode | 30 |
| ตารางที่ 2.7 | DMA Request Sense | 31 |
| ตารางที่ 2.8 | การทำ DMA Channel 0 กับ ASCI | 32 |
| ตารางที่ 2.9 | MPBR/EFR Multiprocessor Bit Receive/error | 37 |
| ตารางที่ 2.10 | แสดง Baud Rate เมื่อทำการเลือก Speed Select | 40 |
| ตารางที่ 2.11 | Timer Output Control<TOC1, TOC0> | 41 |
| ตารางที่ 2.12 | แสดงข้อมูลของรีจิสเตอร์ | 48 |
| ตารางที่ 2.13 | แสดงคาบเวลาที่อินพุต คือ T1 และ T0 | 50 |
| ตารางที่ 2.14 | แสดงการกำหนดมาตรฐานแรงดันไฟฟ้า | 61 |
| ตารางที่ 2.15 | คุณลักษณะโดยย่อของสัญญาณ RS 232C | 65 |
| ตารางที่ 2.16 | การป้อนกระแสแบบเวฟ | 68 |
| ตารางที่ 2.17 | การป้อนกระแสแบบ 2 เฟส | 69 |
| ตารางที่ 2.18 | การป้อนกระแสแบบครึ่งสเต็ป | 70 |

# สารบัญรูป


| รูป | | หน้า |
|---|---|---|
| รูปที่ 2.1 | บล็อคไดอะแกรม Z80180 | 4 |
| รูปที่ 2.2 | โครงสร้างของชิพ Z80180 | 4 |
| รูปที่ 2.3 | Register I/O Tc2 Address 3EH | 11 |
| รูปที่ 2.4 | UMCR I/O Address 3EH | 12 |
| รูปที่ 2.5 | Operation of RETI Instruction | 12 |
| รูปที่ 2.6 | Up-Code Fetch Timing | 13 |
| รูปที่ 2.7 | Writing 0 to LIRTE เมื่อ LIRE = 0 | 13 |
| รูปที่ 2.8 | I/O Read and Write Cycles with IOC = 1 | 14 |
| รูปที่ 2.9 | I/O Read and Write Cycles with IOC = 0 | 14 |
| รูปที่ 2.10 | CPU Z80180 Machine Cycle | 15 |
| รูปที่ 2.11 | CPU Z80 Machine Cycle | 16 |
| รูปที่ 2.12 | DCNTL Memory Wait Insertion 32 H | 16 |
| รูปที่ 2.13 | Logical Address | 19 |
| รูปที่ 2.14 | การกำหนดตำแหน่งของ Logical Address | 20 |
| รูปที่ 2.15 | Logical Memory Organization | 21 |
| รูปที่ 2.16 | การ Set 0000H = Blank Area = Common Area | 23 |
| รูปที่ 2.17 | Interrupt Vector Low Register < IL I/O Address 35H > | 25 |
| รูปที่ 2.18 | Int/Trap Control Register (ITC Address I/O 34H) | 25 |
| รูปที่ 2.19 | Refresh Control Register (RCR Address I/O 36H) | 26 |
| รูปที่ 2.20 | DMA Status Register (DSTAT I/O Address 30H) | 28 |
| รูปที่ 2.21 | DMA Mode Register (DMOD Address I/O 31H) | 29 |
| รูปที่ 2.22 | DMA/Wait Control Register (DCNTL I/O Address 32H) | 31 |
| รูปที่ 2.23 | Figure 35.ASCI Block Diagram | 33 |
| รูปที่ 2.24 | Stat 0 (I/O Address 04H) | 34 |
| รูปที่ 2.25 | Stat 1 (I/O Address 05H) | 34 |
| รูปที่ 2.26 | CNTLA 0 (I/O address 01H) | 36 |
| รูปที่ 2.27 | CNTLA 1 (I/O address 01H) | 36 |

# สารบัญรูป(ต่อ)


| รูป | | หน้า |
|---|---|---|
| รูปที่ 2.28 | CNTLB 0, 1 I/O Address 02H, 03H | 38 |
| รูปที่ 2.29 | CSI I/o Control Register Address OAH | 39 |
| รูปที่ 2.30 | TCR Address 10H | 41 |
| รูปที่ 2.31 | แสดงการจัดวางของ MSM 6242B | 45 |
| รูปที่ 2.32 | โครงสร้างของ LCD | 52 |
| รูปที่ 2.33 | แสดงโครงสร้างของโมเลกุลของผลึกเหลวทั้ง 3 แบบ | 53 |
| รูปที่ 2.34 | การทำงานพื้นฐานของตัวแสดงผลแบบนีเมติกชนิดเกลียว | 54 |
| รูปที่ 2.35 | การแสดงผลเชิงบวก | 55 |
| รูปที่ 2.36 | การแสดงผล 7 ส่วน แบบมีจุดทศนิยม | 56 |
| รูปที่ 2.37 | แบบต่างๆ ของการแสดงผลของ LCD | 56 |
| รูปที่ 2.38 | ความสัมพันธ์ระหว่างความเข้มของแสงกับแรงดัน | 58 |
| รูปที่ 2.39 | วงจรสมมูล ของ LCD | 59 |
| รูปที่ 2.40 | การใช้ RS 232C เชื่อมต่ออุปกรณ์ | 60 |
| รูปที่ 2.41 | ย่านของแรงดันไฟฟ้าที่ใช้ในสัญญาณ RS 232C | 61 |
| รูปที่ 2.42 | การกำหนดขั้วต่อ RS-232 แบบ DB-25 | 62 |
| รูปที่ 2.43 | แสดงขั้วต่อ RS-232 แบบ DB-9 | 63 |
| รูปที่ 2.44 | สเต็ปเปอร์มอเตอร์ 4 เฟส แบบยูนิโพลาร์เพอร์มาแนนท์แม็กเนต | 66 |
| รูปที่ 2.45 | การพันขดลวดบนสเต็ปเปอร์มอเตอร์ | 67 |
| รูปที่ 2.46 | การขับแบบเวฟ | 69 |
| รูปที่ 2.47 | การขับแบบ 2 เฟส | 69 |
| รูปที่ 2.48 | การขับแบบครึ่งสเต็ป | 70 |
| รูปที่ 3.1 | วงจรควบคุม | 72 |
| รูปที่ 3.2 | การจัด Memory Map CPU Z80180 แบบ Logical | 72 |
| รูปที่ 3.3 | การจัด Memory Map CPU Z80180 แบบ Physical | 73 |
| รูปที่ 3.4 | Matrix Switch ของวงจรควบคุม | 74 |
| รูปที่ 3.5 | ภาคแสดงผลโดยใช้ LCD | 74 |
| รูปที่ 3.6 | ภาคแสดงผลโดยใช้ LED | 75 |

# สารบัญรูป (ต่อ)


| รูป | | หน้า |
|---|---|---|
| รูปที่ 3.7 | ภาคแสดงผลโดยใช้ลำโพง | 75 |
| รูปที่ 3.8 | การ Interface ระหว่าง 8255 กับ Matrix Switch | 76 |
| รูปที่ 3.9 | การ Interface ระหว่าง 8255 กับอุปกรณ์ภายนอก | 76 |
| รูปที่ 3.10 | วงจรสื่อสารข้อมูลแบบอนุกรม RS-232 | 77 |
| รูปที่ 3.11 | วงจรสื่อสารข้อมูลแบบอนุกรม RS-485 | 78 |
| รูปที่ 4.1 | แสดงผลการรันโปรแกรม | 79 |
| รูปที่ 4.2 | แสดงการต่อวงจรทดลองชุดรีเลย์และสวิทช์ | 80 |
| รูปที่ 4.3 | แสดงการต่อทดลองจริง | 81 |
| รูปที่ 4.4 | แสดงการต่อวงจร DC Motor | 82 |
| รูปที่ 4.5 | แสดงการต่อทดลองจริง | 83 |
| รูปที่ 4.6 | แสดงการต่อวงจร Stepping Motor | 85 |
| รปที่ 4.7 | แสดงการต่อทดลองจริง | 86 |
| รูปที่ 4.8 | แสดงการต่อวงจรไฟจราจร | 86 |
| รูปที่ 4.9 | แสดงการต่อทดลองจริง | 87 |
| รูปที่ 4.10 | แสดงการเชื่อมต่อระหว่างชุดสาธิต PLC กับ คอมพิวเตอร์ | 90 |
| รูปที่ 4.11 | แสดงการต่อทดลองจริง | 91 |
| รูปที่ ก.1 | ด้านหน้าตัวเครื่องชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ | 95 |
| รูปที่ ก.2 | ด้านข้างตัวเครื่องชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ | 95 |
| รูปที่ ก.3 | ภายในตัวเครื่องชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ | 96 |
| รูปที่ ก.4 | ด้านหน้าตัวเครื่องชุดอินเตอร์เฟส | 96 |
| รูปที่ ก.5 | ด้านข้างตัวเครื่องชุดอินเตอร์เฟส | 97 |
| รูปที่ ก.6 | ด้านในตัวเครื่องชุดอินเตอร์เฟส | 97 |
| รูปที่ ก.7 | สายเชื่อมต่อชุดสาธิต PLC เข้ากับคอมพิวเตอร์ | 98 |
| รูปที่ ข.1 | วงจรชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ | 100 |
| รูปที่ ข.2 | วงจรสเตปปิ้งมอเตอร์ | 102 |
| รูปที่ ข.3 | แสดงลายวงจรด้านบน ชุดสาธิตการทำงาน PLC | 103 |

# สารบัญรูป (ต่อ)


| รูป | | หน้า |
|---|---|---|
| รูปที่ ข.4 | แสดงลายวงจรด้านล่าง ชุดสาธิตการทำงาน PLC | 104 |
| รูปที่ ข.5 | แสดงการวางอุปกรณ์ด้านบน ชุดสาธิตการทำงาน PLC | 105 |
| รูปที่ ข.6 | แสดงการวางอุปกรณ์ด้านล่าง ชุดสาธิตการทำงาน PLC | 106 |
| รูปที่ ค.1 | ผังงานโปรแกรมหลัก | 108 |
| รูปที่ ค.2 | ผังงานโปรแกรมย่อย LD_IN | 109 |
| รูปที่ ค.3 | ผังงานโปรแกรมย่อย LD_OUT | 110 |
| รูปที่ ค.4 | ผังงานโปรแกรมย่อย LD_AUX | 111 |
| รูปที่ ค.5 | ผังงานโปรแกรมย่อย LD_TIM | 112 |
| รูปที่ ค.6 | ผังงานโปรแกรมย่อย LD_CNT | 113 |
| รูปที่ ค.7 | ผังงานโปรแกรมย่อย LD_NOT | 114 |
| รูปที่ ค.8 | ผังงานโปรแกรมย่อย AND | 115 |
| รูปที่ ค.9 | ผังงานโปรแกรมย่อย OR | 116 |
| รูปที่ ค.10 | ผังงานโปรแกรมย่อย SET | 117 |
| รูปที่ ค.11 | ผังงานโปรแกรมย่อย RES | 118 |
| รูปที่ ค.12 | ผังงานโปรแกรมย่อย CNT | 119 |
| รูปที่ ค.13 | ผังงานโปรแกรมย่อย TIMER | 120 |

# บทที่1
# บทนำ

## 1.1 ความเป็นมา และความสำคัญของปริญญานิพนธ์

ในสถานการณ์ปัจจุบัน โรงงานอุตสาหกรรมของประเทศไทยกำลังเจริญก้าวหน้า โรงงานต่างๆ เหล่านี้จะทำงานเป็นระบบ ซึ่งระบบของโรงงานจะมีการควบคุมอุปกรณ์หรือเครื่องจักรต่างๆ ภายในโรงงานก็จะควบคุมอุปกรณ์เครื่องจักรเหล่านั้นด้วยระบบอัตโนมัตินั้นก็คือ การนำเอา PLC มาใช้ในโรงงาน

PLC (Programmable Logic Controller) เป็นอุปกรณ์ชนิดหนึ่ง ที่นำมาแทนการควบคุมที่ใช้รีเลย์ ทำให้สะดวกขึ้น เพราะเป็นระบบอิเล็กทรอนิกส์ซึ่งใช้การเขียนโปรแกรมควบคุม ทำนองเดียวกับคอมพิวเตอร์แทนการเดินสายไฟฟ้า มีหน่วย Input/Output แบบลอจิก (On / Off) และแบบแอนาล็อก (Analog) จึงทำให้สามารถควบคุมเครื่องจักรได้ทุกชนิด นอกจากนั้น ยังมีหน่วย Input/Output จำนวนมาก อีกทั้งมีขนาดเล็กและราคาถูก เมื่อเสียหายก็ทำได้โดยเปลี่ยนโมดูลเท่านั้น และ PLC สามารถตรวจสอบสภาวะ "On " หรือ "Off" ของอุปกรณ์ภายนอกตามโปรแกรมได้ ทำให้สามารถตรวจหาข้อบกพร่องได้อย่างรวดเร็ว

PLC มีข้อดีหลายข้อดีหลายประการ ดังต่อไปนี้

1) การแก้ไขโปรแกรมคอนโทรลได้ง่าย
2) เนื้อที่ติดตั้งใช้เนื้อที่น้อย
3) มีความน่าเชื่อถือ
4) บำรุงรักษาง่าย
5) ลดการเดินสายไฟฟ้าควบคุม
6) มีประสิทธิภาพสูง

ดังนั้นจากประโยชน์ของ PLC ดังที่ได้กล่าวมาแล้ว ทางคณะผู้จัดทำจึงได้เลือกที่จะศึกษาออกแบบ รวมถึงการเขียนโปรแกรมควบคุม โปรแกรมประยุกต์ใช้งานต่างๆ เพื่อที่จะเป็นแนวทางในการพัฒนาในรุ่นต่อๆไปได้

## 1.2 ขีดความสามารถของโครงงาน

โครงงานนี้มีขีดความสามารถดังนี้

1) หน่วยอินพุต 16 อินพุต และหน่วยเอาต์พุต 8 เอาต์พุต พร้อม LED แสดงสภาวะการทำงานและจุดต่อเสียบ
2) ชุดเชื่อมต่อมอเตอร์ขนาด 24 VDC และ Stepping Motor ; Traffic Light
3) แป้นพิมพ์ ขนาด 32 ปุ่ม สำหรับป้อนคำสั่ง ข้อมูลที่เป็นตัวเลข สำหรบแก้ไขโปรแกรม และปุ่มสั่งงาน
4) หน่วยแสดงผล เป็น LCD ขนาด 2 x 16 ตัวอักษร
5) สั่งงานโปรแกรมผ่านทางเครื่อง PC ด้วยโปรแกรม PL7-07
6) ความจุโปรแกรม ป้อนโปรแกรมได้สูงสุด 1000 บรรทัด
7) ส่วนเชื่อมต่อคอมพิวเตอร์ พอร์ต RS-232C
8) แสดงสภาวะการทำงานของชุดทดลองด้วยสัญญาณ
9) ใบงานประกอบการทดลอง 10 ใบงาน

## 1.3 เนื้อหาโดยสังเขป

ปริญญานิพนธ์ฉบับนี้มีเนื้อหาทั้งหมด 5 บท ดังต่อไปนี้

บทที่ 1 บทนำ ซึ่งเป็นเนื้อหาเกี่ยวกับ ความเป็นมา และความสำคัญของปัญหาที่ต้องทำให้เกิดโครงงานนี้ขึ้น อีกทั้งยังกล่าวถึงวัตถุประสงค์ ขอบเขตและประโยชน์ของการทำปริญญานิพนธ์ในครั้งนี้

บทที่ 2 ทฤษฎีและหลักการ จะกล่าวถึงเนื้อหาที่นำมาอ้างอิงและใช้เป็นแนวทางในการออกแบบและสร้างชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์

บทที่ 3 การออกแบบและการสร้างจะเป็นเนื้อหาโดยละเอียดตั้งแต่ขั้นตอนในการออกแบบวงจรในส่วนต่างๆ การนำส่วนต่างๆ มาอินเตอร์เฟสกัน เพื่อให้สามารถทำงานร่วมกันได้อย่างมีประสิทธิภาพ

บทที่ 4 การทดลองและผลการทดลอง ในบทนี้เป็นการนำเสนอการทดลองและผลการทดลอง โดยแบ่งการทดลองออกเป็นส่วนๆ ตามการออกแบบและการสร้าง พร้อมบันทึกผลการทดลองในแต่ละส่วน

บทที่ 5 บทสรุป ปัญหาแนวทางแก้ไขและพัฒนา ซึ่งเป็นการสรุปผลเกี่ยวกับความสามารถประสิทธิภาพการทำงานของชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ และกล่าวถึงปัญหาที่เกิดขึ้นนับตั้งแต่การเริ่มสร้างโครงงานจนกระทั่งโครงงานเสร็จสมบูรณ์ ตลอดจนแนวทางแก้ไข

ปัญหาที่เกิดขึ้น พร้อมทั้งเสนอแนวทางการพัฒนาชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซส เซอร์ ให้สามารถนำไปประยุกต์ใช้งานได้อย่างกว้างขวาง และปรับปรุงให้มีประสิทธิภาพดียิ่งขึ้น

ภาคผนวก ก เครื่องต้นแบบ

ภาคผนวก ข วงจร และแผ่นวงจรพิมพ์

ภาคผนวก ค ผังงานและโปรแกรมการทำงาน

ภาคผนวก ง ใบงานการทดลอง

ภาคผนวก จ รายละเอียดและคุณสมบัติของอุปกรณ์

ภาคผนวก ฉ คู่มือการใช้งาน

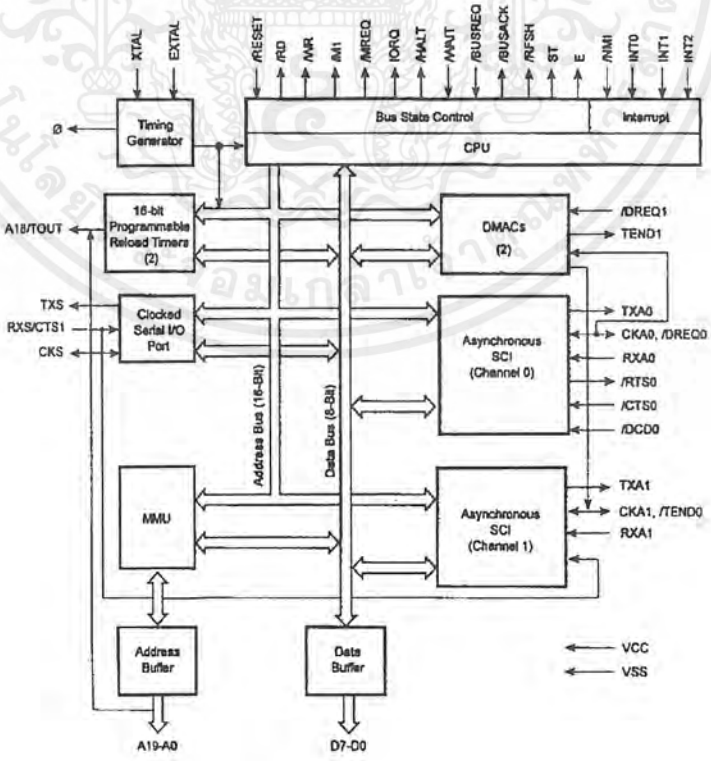
# บทที่ 2
# ทฤษฎีและหลักการ

## 2.1 กล่าวนำ

โดยทั่วๆไป หลักการทำงานของ PLC นั้น จะมีส่วนประกอบที่สำคัญ คือ CPU การใช้คำสั่ง การติดต่อกับอุปกรณ์ภายนอก ซึ่งในแต่ละลักษณะจะมีรายละเอียดดังนี้

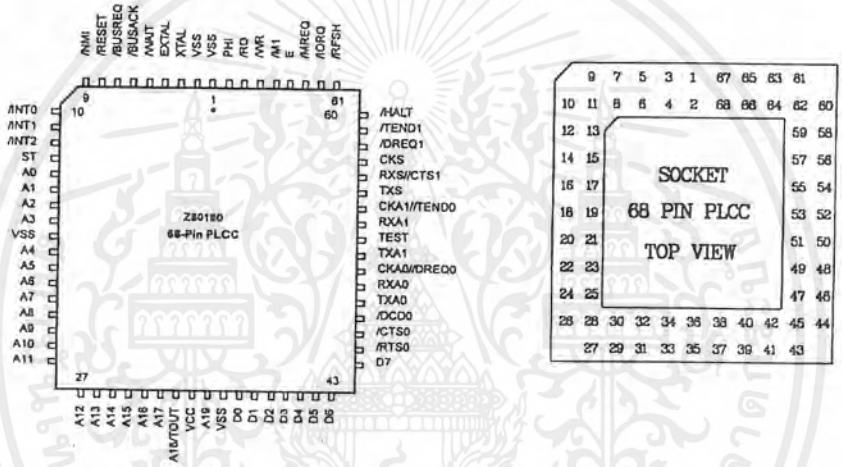
## 2.2 ไมโครโปรเซสเซอร์ Z80180

Z80180 เป็น CPU ที่มีความสามารถสูงที่ได้รวมชิพสำคัญอื่นๆ ไว้ใน CPU ชิพเดี่ยวจึงทำให้มีลักษณะคล้ายกับ CPU ที่ใช้งานควบคุมในจำพวก"ชิพเดี่ยว"แต่เนื่องจากชิพเดี่ยวมีข้อดี คือ เป็นระบบเล็กราคาถูก แต่ข้อเสียคือการโปรแกรมควบคุมค่อนข้างยากในตอนเริ่มต้นและกับระบบงานที่ใหญ่ขึ้นแต่ Z80180 ทางด้านโปรแกรมจะสะดวกอย่างมากเพราะคำสั่งที่ใช้มีมาก และตรงไปตรงมาทั้งคู่มือภาษาไทยและตัวอย่างการใช้งานอย่างมากมายเพราะ Z80180 นี้เป็น SUPPER COMPAT Z80 คือคำสั่งทั้งหมดยังเป็น Z80 และได้เพิ่มชุดคำสั่งขึ้นมาเพื่อเพิ่มความสะดวกในการใช้งานขึ้นอีก ดังรูป2.1




**รูปที่ 2.1** ผังงานของ Z80180

เมื่อมองดูระบบไมโครคอนโทรลเลอร์ชิพเดี่ยวแล้ว Z80180 จะด้อยกว่าตรงที่ไม่มี ROM RAM และ Port แต่ถ้าเป็นในระดับงานอุตสาหกรรมแล้วระบบของ Z80180 กับชิพไมโครคอนโทรลเลอร์แล้วจะไม่ต่างกันเลยเพราะความต้องการเนื้อที่ในการเก็บข้อมูลมากและ PORT มากตามจึงทำให้ต้องต่อเพิ่มภายนอกขึ้นจึงทำให้ Z80180 ในระดับงานคอนโทรลอุตสาหกรรมคล่องตัวกว่ามากเพราะภายใน Z80180 ประกอบด้วย เป็น CMOS ,Oscilator ในตัวใช้งานที่ 10 MHz, MMU ชิพ อ้าง หน่วยความจำได้ 1 MB , DMA 2 Channel , Port , สื่อสาร UART 2 Channel , Clock Serial I/O , 16 Bit Timer Counter และเกี่ยวกับ Port สื่อสารสามารถทำ Multi Processor Communication ซึ่งโครงสร้างของชิพนี้จะเป็นดังรูปที่ 2.2




รูปที่ 2.2 โครงสร้างของชิพ Z80180

### 2.2.1 ขาที่ใช้งาน

**A0-A19** Address Bus 20 เส้น ระหว่าง Reset จะเป็น High Impedance

**BUSAK** Bus Acknowledge เป็นขา Output Active Low ทำงานก็ต่อเมื่อ Z80180 ตอบสนองต่อการ ขอ BUS ของ BUSRQ และจะทำให้ BUS ข้อมูล BUS Address และสัญญาณ Control บางเส้นเป็น High Impedance

**BUSRQ** BUS Request เป็นขา Input Active Low ซึ่งจะมีความสำคัญสูงกว่า NMI โดยจะมีการตรวจสอบสัญญาณนี้ทุกๆการสิ้นสุดของ Machine Cycle

**CXA0,CXA1** Asynchronous Clock 0 และ1 เป็นสัญญาณ Clock แบบ 2 ทิศทาง คือจะใช้เป็นขา อินพุตหรือเอาต์พุตก็ได้

| | |
|---|---|
| **CKS** | Serial Clock เป็นขา Clock 2 ทิศทาง CSI/0 |
| **CLOCK** | เป็นขา Output โดยจะเป็นครึ่งหนึ่งของคริสตอล หรือ Clock Out เช่น คริสตอล 12 MHz Z80180 จะทำงานที่ 6 MHz |
| **CTS0-CTS1** | Clear to Send 0 และ 1 เป็นขา Input Active Low ใช้ในการควบคุม โมเด็ม |
| **D0-D7** | Data BUS เป็นแบบ 2 ทิศทาง |
| **DCD0** | Data Carrier Detect 0 เป็นขา Input Active Low ใช้ควบคุมในการติดต่อกับโมเด็มของ ASCI Channel 0 |
| **DREQ0-DREQ1** | DMA Request 0 และ 1 เป็นขา Input Active Low ใช้ในการขอ DMA และ ขานี้จะโปรแกรมได้ว่าให้ตรวจสัญญาณที่ขอบ หรือระดับได้ |
| **E** | Enable Clock เป็นขา Output Active High ซึ่งใช้ซิงค์การทำงานกับอุปกรณ์ภายนอกระหว่างการทำงานเกี่ยวกับ Bus และใช้เชื่อมต่อกับอุปกรณ์ภายในตระกูล 68XX และ 80XX |
| **HALT** | เป็นขา Output Active Low จะทำงานเมื่อทำคำสั่ง Halt หรือ SLP |
| **INT0** | Maskable Interrupt 0 เป็นขา Input Active Low สัญญาณ ที่ขานี้จะถูกตรวจทุก ๆ การสิ้นสุดคำสั่ง |
| **INT1,INT2** | เช่น เดียวกับ INT0 แต่มีระดับความสำคัญรองลงมาตามลำดับ |
| **IORQ** | เป็นขา Output เพื่อบอกว่ากำลังติดต่อกับ I/O หรือขา IOE ใน 64180 |
| **M1** | Machine Cycle 1 เป็นขา Output Active Low จะทำงานเมื่อ Fetch Op-Code หรือเป็นขา LIR ของ 64180 |
| **NMI** | Non Maskable Interrupt เป็นขา Input Active Low ขานี้จะตอบรับการอินเตอร์รัพท์เสมอโดยไม่สามารถหยุดด้วยซอฟท์แวร์ |
| **RD** | เป็นขาที่ใช้ทำการอ่านข้อมูลจากหน่วยความจำหรืออุปกรณ์ Input/Output |
| **RFSH** | เป็นขาที่ให้ Address Low (A0-A7) ไป Refresh Dynamic RAM หรือ ขา REF ของ 64180 |

**RTS0** Request to Send เป็นขา Output Active Low ขานี้ใช้โปรแกรมสัญญาณควบคุมโมเด็มของ ASCI Channel 0

**RXA0,RXA1** Receive Data 0 และ 1 เป็นขารับสัญญาณจาก Serial ของ ASCI

**RXS** Clock Serial Receive Data เป็นขารับสัญญาณ Serial ของ CSIO

**ST** Status เป็นขา Output Active High ใช้แสดงสถานะการทำงานของ CPU โดยรวมกับ M1 และ HALT ดังตารางที่ 2.1

**ตารางที่ 2.1** สถานะการทำงานของ CPU

| St | Halt | M1 | Operation |
|---|---|---|---|
| 0 | 1 | 0 | CPU operation (1 st op-code fetch) |
| 1 | 1 | 0 | CPU operation (2 nd op-code and 3 rd op-code fetch) |
| 1 | 1 | 1 | CPU operation (MC except for op-code fetch) |
| 0 | X | 1 | DMA operstion |
| 0 | 0 | 0 | HALT mode |
| 1 | 0 | 1 | SLEEP mode (Including System Stop Mode) |

Note X: Don't care

MC:Machine cycle

**TEND0-TEND1** Transfer End 0 และ 1 เป็นขา Output Active Low ใช้แสดงถึงว่า ทำ DMA สิ้นสุดลงแล้ว

**TOUT** Timer Out ใช้กำเนิดพัลส์จาก PRT Channel 1

**TXA0,TXA1** Transmit Data 0 และ 1 เป็นขาส่งข้อมูล Serial ของ ASCI

**TXS** Clock Serial Transmit เป็นขาส่งข้อมูล Serial ของ CSIO

**WAIT** ขา Input Active Low จะถูกตรวจที่ขอบขาลงของ Clock ลูกที่ 2 ของทุกๆ Machine เพื่อเป็นการรออุปกรณ์ภายนอกทำงานให้ทันกับการทำงานของ CPU

**WR** ใช้สำหรับการส่งข้อมูลไปยัง I/O หรือ Memory

**X-TAL** เป็นขาที่ใช้ต่อกับ X-TAL

**ขาที่ MULTIPLEX**

**A18/TOUT** ระหว่าง Reset จะเป็น A18 แต่ถ้ามีการเลือก Set Bit TOC1 หรือ TOC0 ใน Timer Control Registor (TCR) ก็จะทำหน้าที่เป็น TOUT

**CKA0/DREQ0** ระหว่าง Reset ขานี้จะเป็น CKA0 แต่ถ้า DM1 หรือ SM1 ใน DMA Mode Registor (DMODE) ถูก Set เป็น 1 จะเป็นขา DREQ0

**CAK1/TEND0** ระหว่าง Reset จะเป็นขา CKA1 แต่ถ้า BIT CKA1D ใน ASCI ถูก Set จะเป็นขา TEND0

**RXS/CTS1** ระหว่าง Reset ขานี้จะเป็นขา RXS ถ้า BIT CTS1E ใน ASCI ถูก Set จะเป็นขา CTS1

### 2.2.2 Internal I/O Registor

Internal I/O Registor ซึ่งมีด้วยกัน 64 I/O Address ดังแสดงตารางที่ 2.2

**ตารางที่ 2.2** Internal I/O Registor

| | Register | Mnemonic | Address | |
|---|---|---|---|---|
| | | | Binary | Hexadecimal |
| ASCI | ASCI Control Register A Ch 0 | CNTLA0 | XX000000 | 00H |
| | ASCI Control Register A Ch 1 | CNTLA 1 | XX000001 | 01H |
| | ASCI Control Register B Ch 0 | CNTLB0 | XX000010 | 02H |
| | ASCI Control Register B Ch 1 | CNTLB1 | XX000011 | 03H |
| | ASCI Status Register Ch 0 | STATO | XX000100 | 04H |
| | ASCI Status Register Ch 1 | STAT1 | XX000101 | 05H |
| | ASCI Transmit Data Register Ch 0 | TDRO | XX000110 | 06H |
| | ASCI Transmit Data Register Ch 1 | TDR 1 | XX000111 | 07H |
| | ASCI Receive Data Register Ch 0 | RDRO | XX001000 | 08H |
| | ASCI Receive Data Register Ch 1 | RDR1 | XX001001 | 09H |
| CSI/O | CSI/O Control Register | CNTR | XX001010 | 0AH |
| | CSI/O Transmit/Receive Data Register | TRDR | XX001011 | 0BH |

**ตารางที่ 2.2** (ต่อ) Internal I/O Registor

| | Register | Mnemonic | Address | |
|---|---|---|---|---|
| | | | Binary | Hexadecimal |
| Timer | Timer Data Register Ch OL | TMDROL | XX001100 | 0CH |
| | Timar Data Register Ch OH | TMDROH | XX001101 | 0DH |
| | Reload Register Ch OL | RLDROL | XX001110 | 0EH |
| | Reload Register Ch OH | RLDROH | XX001111 | 0FH |
| | Timer Control Rdgister | TCR | XX010000 | 10H |
| | Reserved | | XX010001 | 11H |
| | | | XX010011 | 13H |
| | Timer Data Register Ch 1L | TMDR1L | XX010100 | 14H |
| | Timer Data Register Ch 1H | TMDR1H | XX010101 | 15H |
| | Reload Register Ch 1L | RLDR1L | XX010110 | 16H |
| | Reload Register Ch 1H | RLDR1H | XX010111 | 17H |
| Othere | Free Running Counter | FRC | XX011000 | 18H |
| | Reserved | | XX011001 | 19H |
| | | | XX011111 | 1FH |
| DMA | DMA Source Address Ragister Ch 0L | SAROL | XX100000 | 20H |
| | DMA Source Address Register Ch OH | SAROH | XX100001 | 21H |
| | DMA Source Address Register Ch OB | SAROB | XX100010 | 22H |
| | DMA Destination Address Register Ch OL | DAROL | XX100011 | 23H |
| | DMA Destination Address Register Ch OH | DAROH | XX100100 | 24H |
| | DMA Destination Address Register Ch OB | DAROB | XX100101 | 25H |
| | DMA Byte Count Register Ch OL | BCROL | XX100110 | 26H |
| | DMA Byte Count Register Ch OH | BCROH | XX100111 | 27H |
| | DMA Memory Address Register Ch 1L | MAR1L | XX101000 | 28H |
| | DMA Memory Address Register Ch 1H | Mar1H | XX101001 | 29H |
| | DMA Memory Address Register Ch 1B | MAR1B | XX101010 | 2AH |
| | DMA 1/O Address Register Ch 1L | LAR1L | XX101011 | 2BH |

**ตารางที่ 2.2** (ต่อ) Internal I/O Registor

| | Register | Mnemonic | Address | |
|---|---|---|---|---|
| | | | Binary | Hexadecimal |
| DMA | DMA 1/O Address Register Ch 1H | LAR1H | XX101100 | 2CH |
| | Reserved | | XX10110 1 | 2DH |
| | DMA Byte Count Register Ch 1L | BCR1L | XX101110 | 2EH |
| | DMA Byte Count Register Ch 1H | BCR1H | XX101111 | 2FH |
| | DMA Status Register | DSTAT | XX110000 | 30H |
| | DMA Mode Register | DMODE | XX110001 | 31H |
| | DMA/WAIT Control Register | DCNTL | XX110010 | 32H |
| INT | IL Register (Interrupt Vector Low Register) | IL | XX110011 | 33H |
| | INT/TRAP Control Register | ITC | XX110100 | 34H |
| | Resarved | | XX110101 | 35H |
| Refresh | Refresh Control Register | RCR | XX110110 | 36H |
| | Reserved | | XX110111 | 37H |
| MMU | MMU Common Base Register | CBR | XX111000 | 38H |
| | MMU Bank Base Register | BBR | XX111001 | 39H |
| | MMU Common/Bank Area Register | CBAR | XX111010 | 3AH |
| I/O | Reserved | | XX111011 | 3BH |
| | | | XX111101 | 3DH |
| | Operation Mode Control Register | OMCR | XX111110 | 3EH |
| | I/O Control Register | ICR | XX111111 | 3FH |

จาก MAP I/O ภายในจะเห็นว่าโปรแกรม Z80 เก่าที่เรามีอยู่อาจจะมีการส่ง Port ซ้ำกับI/O ภายใน ทำให้โปรแกรมเดิมทำงานไม่ได้ สามารถแก้ไขโดยการโปรแกรมย้าย MAP I/O ภายใน โดย

การ Control Bit ใน Registor I/O ICR Address 3FH ซึ่งสามารถย้ายไป ที่ใดก็ได้ภายใน 256 ตำแหน่ง ดังรูปที่ 2.3 ส่วนการโปรแกรมแสดงดังตารางที่ 2.3

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | I0A7 | I0A6 | IOSTP | - | - | - | - | - |

**รูปที่ 2.3** Registor I/O Ica Address 3 FH

**ตารางที่ 2.3** แสดงช่วง Address I/O ของภาค Control I0A7&I0A6

| I0A7 | I0A6 | ช่วง Address I/O |
|---|---|---|
| 0 | 0 | 0000-003FH |
| 0 | 1 | 0040-007FH |
| 1 | 0 | 0080-00BFH |
| 1 | 1 | 00C0-00FFH |

เช่น ต้องการย้าย I/O ภายในไป I/O Address 80H เป็นต้นไป จะโปรแกรมได้เป็น

```
LD      A,80H
OUT0    (3FH),A
```

ส่วน IOSTP:IOSTOP Mode Bit 5 เป็น 1 จะทำให้ I/O ภายในหยุดทำงาน เมื่อ Reset Bit นี้ จะเป็น 0

Z80180 จะมอง I/O ภายในเป็นลักษณะ 16 Bit ซึ่งอยู่ในขอบเขต 0000H-00FFH ในการใช้คำสั่ง IN/OUT ของ Z80 ปกติจะมีค่า Address Byte High (A8-A15) ซึ่งไม่ใช้เป็น 00H ดังนั้นการติดต่อกับ I/O ภายในควรใช้คำสั่งเกี่ยวกับ I/O ที่เพิ่มขึ้นมาคือ IN0,OUT0,OTIM,OTIMR,OTDM ,PTDMRและTSTIO ซึ่ง คำสั่งเหล่านี้จะทำให้A8-A15=00H

### 2.2.3 Operation Modes

Z80180 สามารถกำหนดการทำงานให้เหมือน 64180 ได้โดยการ Set Bit Control Mode Control Register (OMCR I/O Address 3EH) แสดงดังรูปที่ 2.4

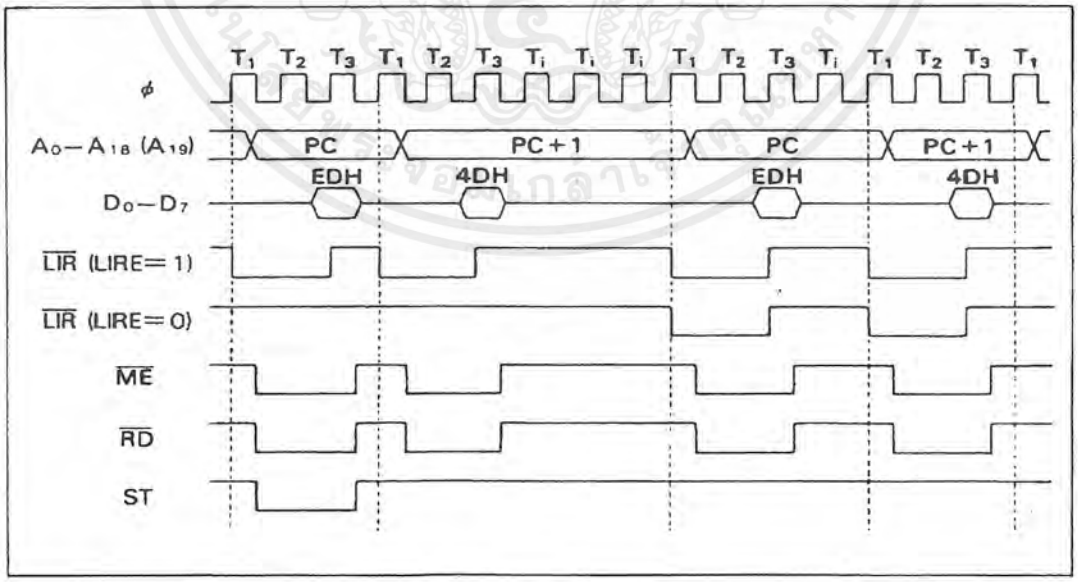
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | M1E(R/W) | MITE(R/W) | IOC(R/W) | - | - | - | - | - |

**รูปที่ 2.4** OMCR I/O Address 3EH

**M1E (M1 Enable)** เป็น Bit ที่ใช้ควบคุมขาสัญญาณ M1 ซึ่งในระหว่าง REST MTE =1

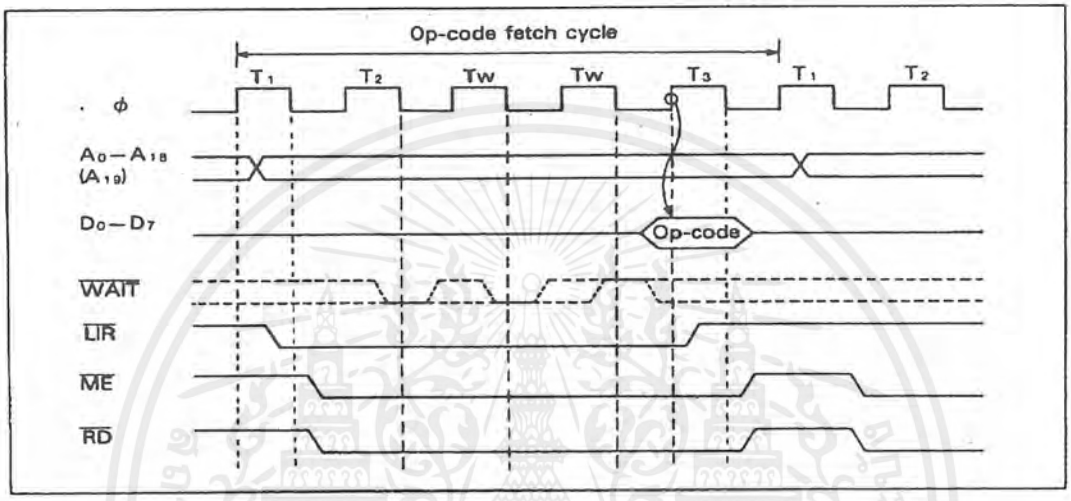
**M1E=1** จะอนุญาตให้ขา M1 ทำงาน Active 0 (เช่นเดียวกับ Z80 ) เมื่อปฏิบัติ Cycles คำสั่ง รับรู้ Cycles ของการ Interrupt INT 0 Machine Cycles แรกของการรับรู้สัญญาณ NMI

**M1E=0** ขาสัญญาณ M1 ปกติจะไม่ทำงาน (LOGIC 1 ) จะ Active Low ต่อเมื่อมีการปฏิบัติคำสั่งครั้งที่ 2 ของคำสั่ง RET1การรับรู้ Cycles ของการ Interrupt INT 0



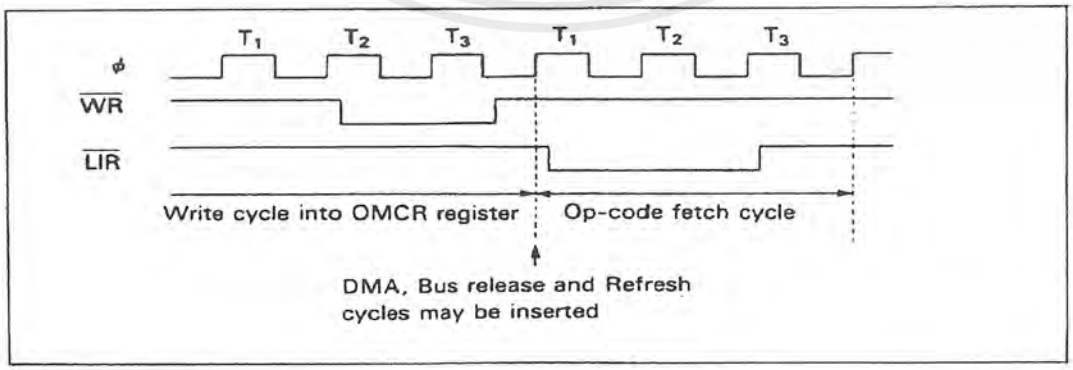

**รูปที่ 2.5** Operation of RETI Instruction

จากรูป 2.5 ของการทำคำสั่ง RET1 สังเกตที่สัญญาณขา M1 จะเห็นว่าถ้า BIT M1E=1 ในการต่อ Interface กับ Z80 'S Peripheral ขา M1 จะนำมาตรวจสอบร่วมด้วย โดยถ้ามีการ Interrupt แบบ Daisy Chain อาจจะทำให้มีการผิดพลาดขึ้นสามารถสรุปเป็นตารางในการ Set Bit ใน Operation Mode Control Registor



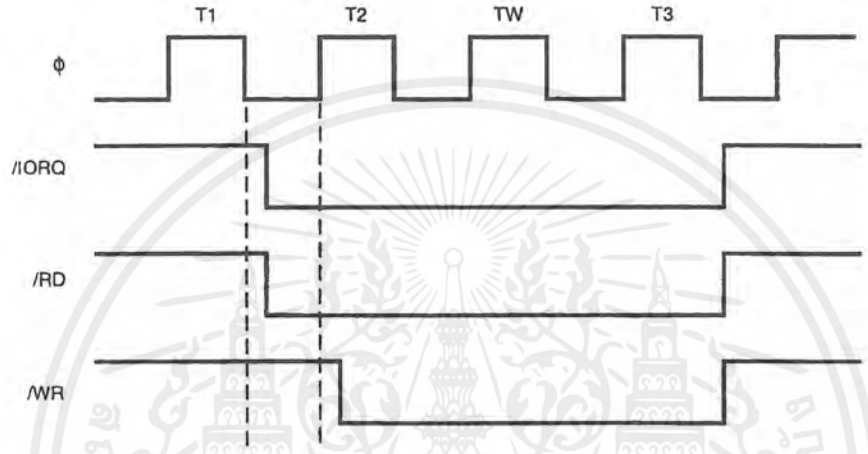

รูปที่ 2.6 Op-Code Fetch Timing

**M1TE (M1 Tempopary Enable )** เป็น Bit ทำให้เกิดสัญญาณ M1 สำรองขึ้นถ้าให้ M1TE=1 ขาสัญญาณ M1 จะเป็นต่อการให้ค่าใน Bit MTE เท่านั้น แต่ถ้า M1TE=0 และ M1E=0 และ M1E=0 ไปแล้วดังรูปที่ 2.7

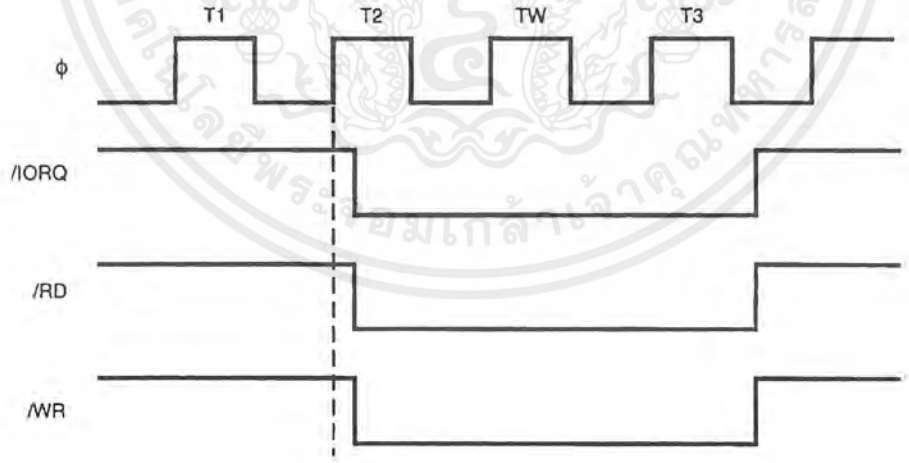



รูปที่ 2.7 Writing 0 to LIRTE เมื่อ LIRE = 0

IOC เป็น Bit ใช้ควบคุม Timing ของ IORQ และ RD ให้เหมือน Z80 หรือ 64180 โดยถ้า BIT นี้ถูก Set เป็น 1 Timing จะเป็นของ 64180 คือ IORQ และ RD จะ Active ที่ขอบขาลงของ T1 แต่ถ้า Bit นี้เป็น 0 Timing จะเป็นของ Z80 คือ จะ Activeที่ขอบขาขึ้นของ T2 เพื่อให้ใช้อุปกรณ์สนับสนุนของ Z80 ได้ ระหว่าง Reset bit นี้จะเป็น 1 ดังรูป




**รูปที่ 2.8** I/O Read and Write Cycles With /IOC = 1

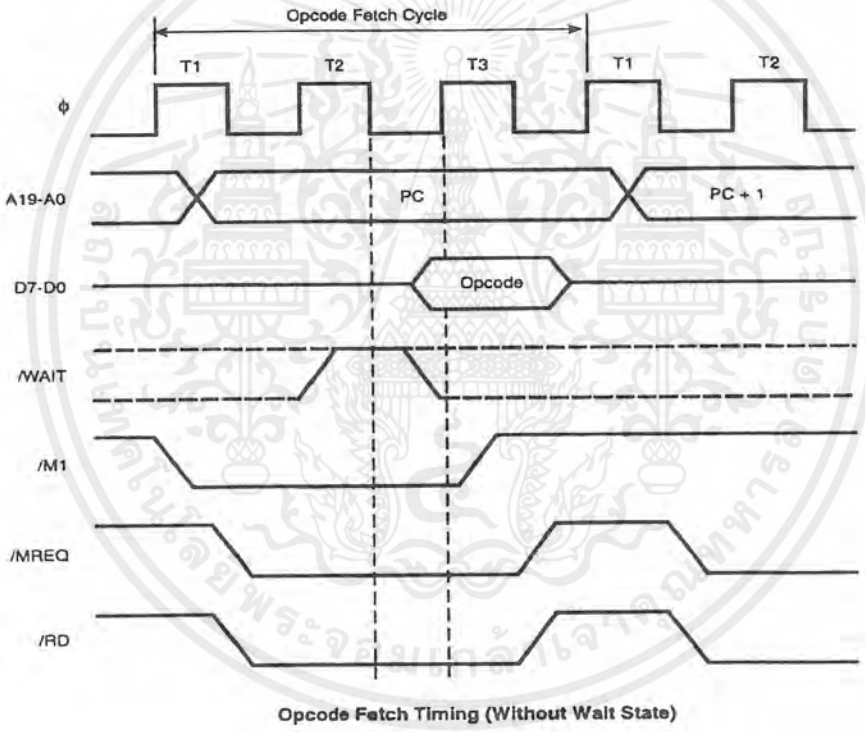



**รูปที่ 2.9** I/O Read and Write Cycles With /IOC = 0

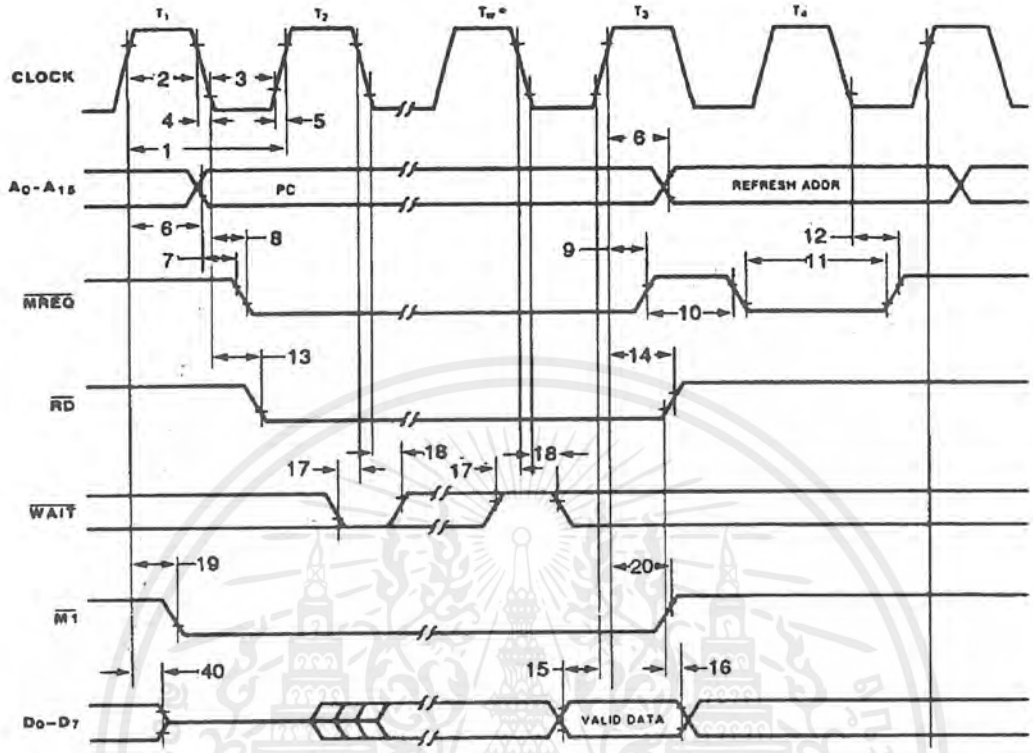
หมายเหตุ การ Set ค่าใน OMCR Register นี้จะต้องโปรแกรมให้เรียบร้อยก่อน ก่อนที่จะมีการปฏิบัติคำสั่ง I/O อื่น

### 2.2.4 เกี่ยวกับ Timing

Z80180 ใช้เวลาในการ ทำคำสั่งใน 1 Machine Cycle น้อยกว่า Z80 อยู่ 1 T State คือ ใช้เวลาใน 1 Machine Cycle เพียง 3 T State ในขณะที่ Z80 ใช้ 4 T State จะเห็นได้ว่าในขณะที่ให้ Z80180 RUN ความถี่เดียวกันกับ Z80 CPU Z80180 ก็ยังให้ความเร็วกว่า Z80 ถึง 25 % แต่ในขณะเดียวกัน Z80180 ยังสามารถต่อ Clock สูงกว่า Z80 ได้มากกว่า 1 เท่า จึงทำให้ความเร็วในการทำงานของ Z80180 ดีกว่ามาก ดูรูปเปรียบเทียบ T State ของ Z80 กับ Z80180




**รูปที่ 2.10** CPU Z80180 Machine Cycle




**รูปที่ 2.11** CPU Z80 Machine Cycle

### 2.2.5 Wait State Generator

Z80180 ทำงานด้วยถี่ที่สูงขึ้นจึงอาจทำให้ Memory หรือ I/O ทำงานไม่ทันจึงต้อง มีสัญญาณมาเป็นตัวช่วยกำหนดความพร้อมระหว่าง CPU กับอุปกรณ์ภายนอกนั้นก็คือสัญญาณ Wait ซึ่ง Z80 นั้นต้องให้อุปกรณ์ภายนอกส่งสัญญาณนี้มาให้แต่ Z80180 ยังสามารถให้โปรแกรมจำนวน Wait State เพื่อเพิ่มเข้าไปในขณะที่ CPU ปฏิบัติคำสั่งหรือทำ DMA ด้วยการโปรแกรมจะใช้ 4 BIT ของ DMA/Wait Control Registor

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | MWI1 | MWI0 | MWI1 | MWI0 | - | - | - | - |

**รูปที่ 2.12** DCNTL Memory Wait Insertion32H

จะทำการเพิ่มจาก 0-3 Wait State ของการเข้าถึง Memory โดยการโปรแกรมดังตารางที่ 2.4

**ตารางที่ 2.4** การ Set ค่า MWI1, MWI0

| MWI1 | MWI0 | จำนวน Wait State |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 2 |
| 1 | 1 | 3 |

BIT 5, 4 IWI1 , [ I/O Wait Insertion ]

จะทำการเพิ่ม Wait State ให้กับ I/O ภายนอกจาก 1-6 ดังตาราง

**ตารางที่ 2.5** การ Set ค่า IWI1, IWI0

| IWI1 | IWI0 | I/O ภายนอก | INTO |
|---|---|---|---|
| 0 | 0 | 1 | 2 |
| 0 | 1 | 2 | 4 |
| 1 | 0 | 3 | 5 |
| 1 | 1 | 4 | 6 |

จะเห็นว่า Wait State ของ I/O มากกว่า Memory อยู่ 1 T State เพราะขณะเข้าถึง I/O ปกติ Wait State จะถูกเพิ่มขึ้น 1 อยู่แล้ว ดังนั้นเมื่อเพิ่ม Wait State เข้าไปก็รวมกับที่ มีอยู่ปกติ และส่วน INT0 ก็เช่นเดียวกัน ขณะเกิด INT0 ปกติจะมี Wait State อยู่ 2 Wait State อยู่แล้ว และขณะที่ Reset Bit Control Wait State ทั้ง 4 จะเป็น 1 ทั้งหมด คือ อยู่ใน Mode ของ Max Wait State

**ตัวอย่าง** เราต้องการเพิ่ม Wait State ในการเข้าถึง Memory 2 Wait State จะโปรแกรมดังนี้

```
IN0 A,(32H)     ; IN ค่าใน Registor DMA/Wait
AND 0BFH        ; Fill เฉพาะ Bit 7 และ 6 เท่านั้น
OUT0(32H),A     ; ที่ใช้ In แล้ว And ก็เพราะว่า Registor นี้
                ; มีการกำหนดเกี่ยวกับ DMA ดังนั้นเราจึง Fill
                ; เฉพาะ Bit ที่จะต้องการโปรแกรม
```

**2.2.6 Halt และ Low Power Mode** มีด้วยกัน 4 Mode คือ

**Halt Mode** โดยทำคำสั่ง 76H จะทำให้ CPU หยุดทำคำสั่ง แต่การทำงานต่างๆของ CPU ยังทำปกติ การออกจาก Halt โดย Reset หรือ Interrupt

**Sleep Mode** โดยการทำคำสั่ง SLP ซึ่ง CPU จะหยุด Clock ภายใน ทำให้ Address เป็น High , Data Bus เป็น Tristale, DRAM Refresh, Internal DMAC หยุด ทำงานการออกจาก Sleep Mode โดย Reset หรือ Interrupt

**IOStop Mode** ใช้หยุดการทำงานของ Chip ภายในคือ ASCI,CSI/O และ PRT โดยการ Set bit ใน I/O Control Register (ICR I/O Address 3FH) เป็น 1 และจะให้ทำงานต่อก็ Resetหรือโปรแกรมให้ Bit ICR เป็น 0

**System Stop Mode** เป็นการรวมกันของ IoStop กับ Sleep Mode โดย การ Set Bit ใน ICR แล้ว ตามด้วยคำสั่ง SLP จะทำให้ IO ภายในหยุดการทำงานและ CPU หยุดทำงานเพื่อเป็นการประหยัดพลังงานซึ่งใน Mode นี้ CPU จะกินกระแส เพียง 7.5 mA ในขณะปกติจะกินกระแสประมาณ 35 mA เมื่อจะออกจาก System Stop Mode ก็โดยการ Reset หรือ Interrupt จากภายนอก

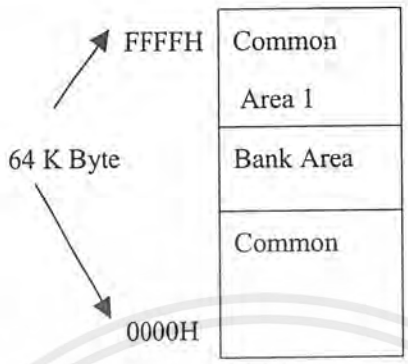
**2.2.7 Memory Management Unit (MMU)**

เนื่องจาก Z80180 สามารถอ้างหน่วยความจำได้ 1024 K Byte แต่ในชุดคำสั่ง Z80 นั้น ไม่มีคำสั่งใดที่จะอ้างข้ามเกิน 64 K Byte ได้และด้วยเป็นการที่ไม่ให้กระเทือนต่อผู้เคยใช้ Z80 อยู่แล้ว การเขียนโปรแกรมและการอ้างถึงหน่วยความจำก็ยังคงสภาพเดิมใน 64 K Byte แต่ในการปฏิบัติงานของ CPU Z80180 จริงๆ ที่จะกระทำกับหน่วยความจำทั้ง 1024 K Byte จะมี MMU มาเป็นผู้จัดการในการเข้าถึงตำแหน่งอันแท้จริง 00000H-FFFFFH ซึ่งในการปฏิบัติงาน จะถูกแยกเป็น 2 แบบ คือ

1) ส่วนของ User เรียกว่า Logical Address (0000H-FFFFH) เป็นส่วนที่ถูกเรียกใช้ในโปรแกรมที่ผู้ใช้เขียนขึ้นมีขอบเขต 64 K Byte

2) ส่วนของ CPU เรียกว่า Physical Address (00000H-FFFFFH) เป็นตำแหน่งที่ CPU ใช้ในการปฏิบัติงานจริง 1024 K Byte

Logical Address 64 K Byte จะถูกแยกเป็น 3 ส่วน ดังนี้




**รูปที่ 2.13** Logical Address

**Common AREA0,1** ส่วนนี้เมื่อเรียกตำแหน่งทาง Logical ที่กำหนดไว้ไม่ว่าการทำงานของ CPU จะอยู่ใน Physical จริงที่ใดก็ตาม จะ กลับมายัง ตำแหน่งที่กำหนดเป็น Common นั้นก็คือ Common จะตามด้วย CPU ไปทุกๆ ตำแหน่งที่กำลังปฏิบัติงานอยู่

**Bank Area** จะมีลักษณะการทำงานเป็น Page เมื่อย้ายตำแหน่งที่เกิน Page ที่กำหนดไปยัง Page อื่น ก็จะไม่สามารถติดต่อ Page ก่อนหน้านี้ได้

## การกำหนดตำแหน่งในการใช้งานของ Logical Address

จะมี Register ชื่อ Common/Bank Area Register (Bar I/O Address-3AH) ซึ่งใช้กำหนดตำแหน่งการเรียกใช้ในทางโปรแกรม (Logical )ขนาด 1 Byte โดยแบ่งออกเป็น 2 Nibbles คือ

High Nibble ใช้กำหนดค่าของ CA (Common Area1 (D7-D4))

Low Nibble ใช้กำหนดค่าของ BA(Bank Area) (D3-D0)

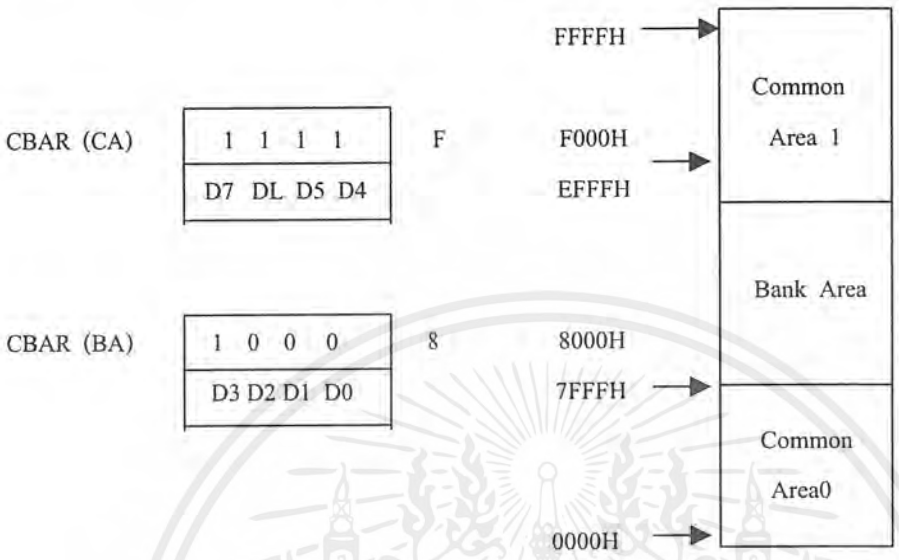
การมองค่าใน CBAR ค่าแต่ละ Nibble จะ X000H จาก Logical ที่ถูกแยกเป็น 3 ส่วนจะได้การกำหนดค่าดังนี้

ตำแหน่ง 0000H →Bank Area (ค่าที่กำหนดใน BA-1)

ตำแหน่งใน BA →Common Area (ค่าที่กำหนดใน CA-1)

ตำแหน่งใน CA →ตำแหน่ง FFFFH

ตัวอย่าง กำหนดให้ CBAR-F8 ดังนั้น CA-F และ BA-8 ตำแหน่ง Logical จะเป็นดังนี้




**รูปที่ 2.14** การกำหนดตำแหน่งของ Logical Address

**Common Area 0** เมื่อถูกเรียกใช้ทางโปรแกรมอยู่ในขอบเขต 32 K Byte จาก Address 0000H-7FFFH

**Bank Area** จะมีขอบเขตในการเรียกใช้ทาง Logical 28K Byte จาก Address 8000H-EFFFH

**Common Area 1** จะมีขอบเขตในการเรียกใช้ทาง Logical 4K BYTE จาก Addressf F000H-FFFFH

ในส่วนของ Bank Area และ Common Area 1 สามารถกำหนดตำแหน่งการใช้งานจริงว่าจะให้อยู่ส่วนใดของหน่วยความจำขนาด 1024 KByte ได้จากค่า Register ขนาด 8 Bit คือ Bank Base Register (BBR: Address 39H) และ Common Base Register (CBR:Address 38H) ตามลำดับ

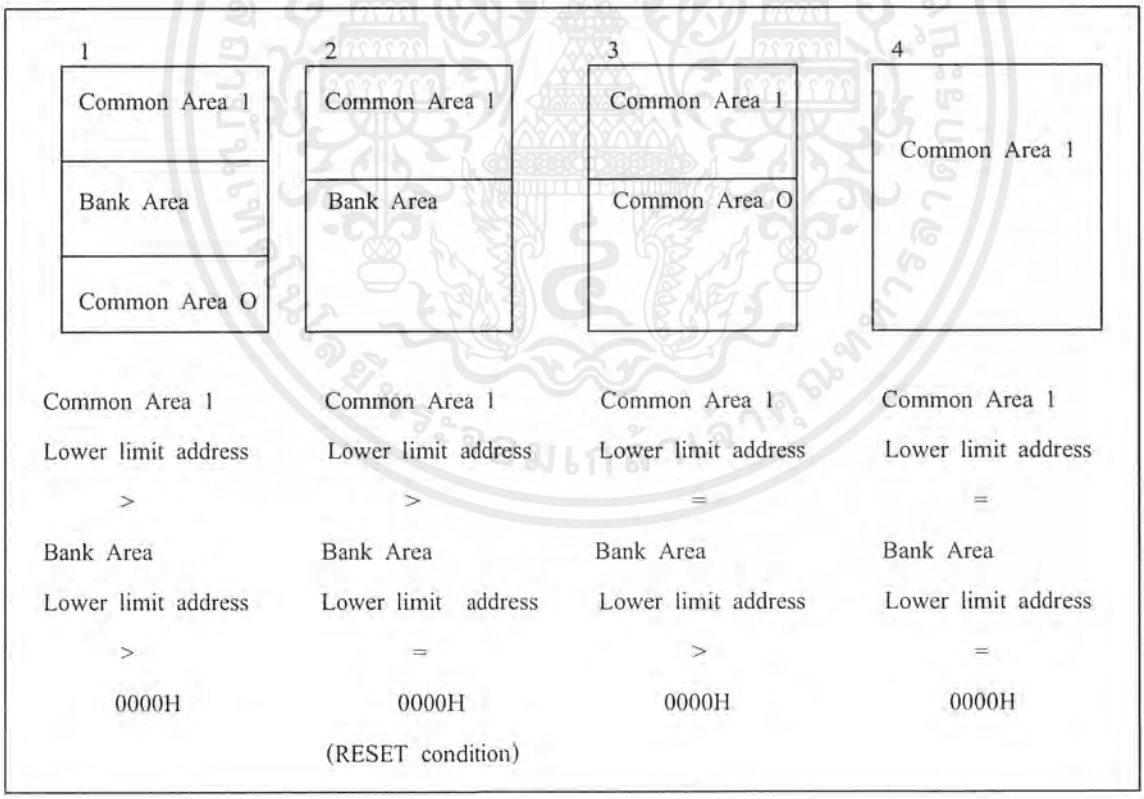
ตำแหน่ง 1024 K Byte (Physical Address)=Bank Base หรือ Common Area1+(BBRหรือ CBR*1000H) จาก Logical ที่กำหนด ต้องการให้ Bank Area อยู่ที่ตำแหน่ง 18000 และ Common Area 1 อยู่ที่ตำแหน่ง 3000H ดังนั้น Physical Address เราทราบแล้วสิ่งที่ต้องการหาคือ ค่าใน BBR

และ CBR จากสูตรด้านบนเปลี่ยนใหม่เป็น

$$BRR = \frac{18000\,H - 8000\,H}{1000\,H} = 40H$$

$$CBR = \frac{30000\,H - F000\,H}{1000\,H} = 21H$$

เช่นเมื่อมีการอ้างถึงตำแหน่ง 8000H ในโปรแกรม ตำแหน่ง 18000H Bank Area จะทำงาน และถ้าอ้างถึงตำแหน่ง F007H จากโปรแกรม ตำแหน่ง 30007H จะทำงาน และถ้ามีการเปลี่ยนค่า BBR จาก 10H เป็น CBR ตำแหน่ง Physical ของ Bank Area หรือ Common Area 1 จะเปลี่ยนไป ในขณะการอ้างถึงทาง Logical ยังคงเดิม เช่นเปลี่ยนค่า BBR จาก 10H เป็น 08H เมื่อมีการอ้างถึง ตำแหน่ง 8000H ในโปรแกรม Physical ของ Bank Area จะถูกทำงานที่ตำแหน่ง 10000H แทน Logical Memory สามารถจัดได้ใน 4 ลักษณะ




**รูปที่ 2.15** Logical Memory Organization

แบบที่ 1 0000H น้อยกว่า CA ทำให้มี Area ใช้งานแยกเป็น 3 ส่วน ดังเช่นการให้ CBAR = F8H

| | | | | | |
|---|---|---|---|---|---|
| 0000H | → BA-1 | BA | → CA-1 | CA | → FFFFH |
| 0000H-7FFFH | | 8000H-EFFFH | | F000H-FFFFH | |
| (Common 0) | | (Bank) | | (Common 1) | |

แบบที่ 2 0000H=BA,BA น้อยกว่า CA ทำให้มี Area ใช้งานเป็นส่วน ดังเช่นการให้ BAR=F0H (กรณีเกิดการ Reset )

| | | | |
|---|---|---|---|
| 0000H = BA | → CA-1 | CA | → FFFFH |
| 0000H-EFFFH(Bank) | | F000H-FFFFH (Common 1) | |

จะเห็นว่า Common Area 0 ไม่มีเนื่องจากการคิดจะเริ่มจากซ้ายไปขวา นั้นก็คือสิ่งที่อยู่ทางขวาจะถือเป็นค่าที่ใหม่กว่า เมื่อตำแหน่ง 0000H = BA แล้วก็จะเป็นค่าของ BA แทน ในกรณีนี้เมื่อ Reset CPU จะเริ่มทำงานที่ Address 0000H ซึ่งเป็น Monitor โปรแกรม ถ้ามีคำสั่งย้าย Bank จะทำให้โปรแกรมทำงานไม่ได้เพราะ Common Area 0 ไม่มีแต่ถ้าต้องการย้าย Bank จำเป็นต้องกระทำการย้ายโปรแกรมไป Run ในส่วน Common 1 ก่อนแล้วจึงทำการย้าย Bank

แบบที่ 3 _0000H น้อยกว่า BA,BA = CA Area ใช้งานก็จะมี 2 ส่วน ดังเช่นการให้ CBAR = 88H

| | | | |
|---|---|---|---|
| 0000H | → BA-1 | BA=CA | → FFFFH |
| 0000H-7FFFH(Common 0) | | 8000-FFFFH(Common 1) | |

ประโยชน์สามารถย้าย Common Area 1 เป็น Block จะ 32 K ทำให้ง่ายต่อการคำนวณตำแหน่งย้าย Block และการเขียนโปรแกรมแต่ในส่วน Common Area 0 จะต้องมีที่เก็บของ Stack ด้วย

แบบที่ 4 0000H=BA=CA Area ใช้งานมีเพียงส่วนดียว ดังนั้น Cbar=00H

0000H=CA=CA → FFFFH

การ SET แบบนี้การใช้พื้นที่ส่วนเดียว 64 K ซึ่งจะทำการย้าย Common Area 1 ก็ไม่ได้ ซึ่งเป็นการทำงานแบบ Z-80 นั้นเอง คือ Physical จะเริ่มจาก 00000H-0FFFFH

**ตารางที่ 2.7** การ Set 0000H = Blank Area = Common Area

| Register | Minemonics | Address | Remarks |
|---|---|---|---|
| MMU Common Base Register | : CBR | 38 | bit / during RESET / R/W: CB7 CB6 CB5 CB4 CB3 CB2 CB1 CB0 / 0 0 0 0 0 0 0 0 / R/W R/W R/W R/W R/W R/W R/W R/W — MMU Common Base Register |
| MMU Bank Base Register | :BBR | 39 | bit / during RESET / R/W: BB7 BB6 BB5 BB4 BB3 BB2 BB1 BB0 / 0 0 0 0 0 0 0 0 / R/W R/W R/W R/W R/W R/W R/W R/W — MMU Bank Base Register |
| MMU Common/Bank Area Register | : CBAR | 3A | bit / during RESET / R/W: CA3 CA2 CA1 CA0 BA3 BA2 BA1 BA0 / 0 0 0 0 0 0 0 0 / R/W R/W R/W R/W R/W R/W R/W R/W — MMU Common Area Register; MMU Bank Area Register |

ตัวอย่าง ต้องการให้

| | | |
|---|---|---|
| Monitor Program | อยู่ที่ 0000H - 7FFFH | Address จริง 0000H – 7FFFH |
| Ram ใช้งาน | อยู่ที่ 8000H - EFFFH | Address จริง 8000H –EFFFH |
| Ram ที่เก็บ Stack | อยู่ที่ F000H - FFFFH | Address จริง F000H – FFFFH |

จะเขียนโปรแกรมดังนี้

```
LD A 0F8H          Common Area 0
OUT0(CBAR),A       Monitor Address 0000-7FFFH Not Remove
LD A,08H
OUT0(BBR),A        Bank Area 10000H Can Remove
OUT0(CBR),A        Common Area 1 10000H Not Remove
```

สรุป

1)เมื่อ Reset ค่าใน CBAR = F0H

2)จะต้องกำหนด Logical Address ใน Register CBAR (34H)ก่อน

3)Phyaical = logical + (BBR (39H)) หรือ CBR (38H) *1000H)

$$BBR, CBR = \frac{Physical - Logical}{1000\,H}$$

4)ในทางการกลับกันรู้ Physical ต้องการหาค่าของ BBR หรือ CBR

5)เมื่อต้องการย้าย Bank หรือ Common โปรแกรมในขณะ RUN นั้น จะต้องอยู่ในส่วนของ Common

### 2.2.8 Interrupt

มีด้วยกัน 12 Interrupt แบ่งเป็น 4 Interrupt ภายนอก และ 8 Interrupt ภายใน โดยมีลำดับความสำคัญจากมากไปน้อย ดังนี้ Trap ,(ภายใน),(ภายนอก) NMI,INT0,INT1,INT2, (ภายใน)Timer 0,Timer 1,DMA Chanel 0, DMA Chanel 1 Clock Serial ,ASCI Chanel 0 และ ASCI Chanel 1

**Registor และ Flag ที่ใช้ควบคุมการ Interrupt**

Inerrrupt Vector Low (IL), Interrupt High (1), Interrupt Trap Control(ITC) และ Flag IEF1,IEF2 โดยที่ Flag IEF1 จะใช้ในการ Enable Interrupt ภายในทั้งหมดยกเว้น Trap

**Interrupt Vector Low Registor (IL I/O Addess 33H)**

ใช้เป็น Vector Table Byte ต่ำ ของ Interrupt ภายนอก INT 1,INT 2 และ Interrupt ภายในทั้งหมดยกเว้น "Trap" โดย 3 Bit สูงของ IL สามารถโปรแกรมได้ แต่ 5 Bit หลังจะถูก Fix ดังรูป

**ตารางที่ 2.8** Interrupt Vector Low Registor <IL I/O Address 35H>

| Interrupt Source | Priority | IL | | | Fixed Code | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| /INT1 | Highest | • | • | • | 0 | 0 | 0 | 0 | 0 |
| /INT2 | • | • | • | • | 0 | 0 | 0 | 0 | 0 |
| PRT Channel 0 | • | • | • | • | 0 | 0 | 0 | 0 | 0 |
| PRT Channel 1 | • | • | • | • | 0 | 0 | 0 | 0 | 0 |
| DMA Channel 0 | • | • | • | • | 0 | 1 | 0 | 0 | 0 |
| DMA Channel 1 | • | • | • | • | 0 | 1 | 0 | 1 | 0 |
| CSI/O | • | • | • | • | 0 | 1 | 1 | 0 | 0 |
| ASCI Channel 0 | • | • | • | • | 0 | 1 | 1 | 1 | 0 |
| ASCI Channel 1 | Lowest | • | • | • | 1 | 0 | 0 | 0 | 0 |

ด้านการ Interrupt ส่วนใหญ่จะเป็น Mode 2 คือ นำค่าใน I และ IL หรือจากอุปกรณ์ที่ ขอ Inerrrupt ในกรณี INT0 มาประกอบกันเป็น Address ที่จะเก็บข้อมูลที่กระโดดไปเช่น I=10H และ IL=40H และใน Address 1040H มีข้อมูล 00H,60H ตามลำดับ เมื่อเกิด Interrupt ขึ้นก็จะกระโดดไปทำโปรแกรมที่ตำแหน่ง 6000H นั้นเอง

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TRAP | UFO | - | - | - | - | - | - |

**รูปที่ 2.15** Int / Trap Control Register (ITC Address I/O 34H)

**E2,1,0** Interrupt Enable 2,1,0 ใช้ Enable และ Disable Interrupt ภายนอก ถ้าเป็น 0 จะ Disable แต่ Bit นี้ จะไม่ทำให้เกิด Interrupt ขึ้นทันทีจนกว่าจะทำคำสั่ง EI ดังนั้น INT 0 จะต่างกับ Z80 ตรงที่มีส่วนนี้ แต่เมื่อเกิด Reset IET0 จะถูก Set เป็น 1 โดยอัตโนมัติเพื่อให้ขึ้นกับคำสั่ง EI หรือ DI อย่างเดียว เช่น Z80 แต่ ITE1 ITE2 จะเป็น 0

**TRAP** จะเป็น 1 เมื่อทำคำสั่งที่ไม่มีใน Z80180 Trap สามารถ Reset ภายใต้โปรแกรมควบคุมได้ แต่ไม่สามารถเขียน 1 เข้าไปได้ระหว่าง Reset จะถูก Clear

**UFO** Undefind Fetch Object เมื่อ Trap เกิดขึ้น UFO จะให้ค่าของตำแหน่งที่ผิดในคำสั่งนั้นไว้ใน Stack เนื่องจาก Trap อาจเกิดขึ้นจาก Opcode 2 หรือ 3 Byte UFO จะปรับค่า PC ให้คือ ถ้าเป็นคำสั่ง Opcode 2 Byte UFO จะเป็น 0 และ จะทำให้ PC ของคำสั่งถัดไป จากคำสั่งที่ไม่ใช่ของ Z80180 ถูกลดลงหนึ่ง แต่ถ้า UFO=1 คำสั่งที่ผิด จะมี Opcode 3 Byte และ PC จะถูกลดลง 2 ตำแหน่ง และค่า PC จะถูกเก็บไว้ใน Stack เช่น

2000 ED 99

2000 PC ชี้คำสั่งถัดไป

เมื่อ CPU Run มาพบข้อมูลที่ตำแหน่ง 2000 ก็จะเกิด Interrupt Trap ขึ้น และรู้ด้วยว่าเป็นคำสั่ง 2 BYTE และ PC ก็ชี้คำสั่งถัดไปคือ Address 2002 แต่ Flag UFO จะถูกทำให้เป็น 0 เพื่อปรับค่า PC นั้นด้วยการลดลงหนึ่งเช่น Address 2001 ซึ่งก็คือ ตำแหน่งข้อมูลที่ผิดนั้นเอง

**Trap Interrupt** เป็นเหมือน NMI คือ ไม่สามารถหยุดได้เมื่อเกิดกระทำคำสั่งผิดขึ้น ซึ่งเป็น ตัวช่วยให้เกิดความหน้าเชื่อถือทางด้าน Software และ อาจใช้เพิ่มคำสั่งได้อีกด้วย Bit Trap ใน ITC จะถูก Set เป็น 1 และ UFO จะ Set หรือไม่ Set ขึ้นอยู่กับว่าเป็นคำสั่ง 2 หรือ 3 Byte และ Flag UFOนี้ก็จะไปปรับ PC ให้ถูกต้อง และเก็บไว้ใน Stack แล้ว กระโดดไป Run ที่ Address 0000H

### 2.2.9 Dynamic RAM Refresh Control

Z80180 ให้ Address A0-A7 สำหรับ Dynamic RAM และยังสามารถโปรแกรมเวลาในการ Refresh โดยการโปรแกรมที่ RCR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | REFE | REFW | - | - | - | - | - | - |

**รูปที่ 2.16** Refresh Control Register (RCR Address I/O 36 H )

**REFE** Refresh Enable เมื่อเป็น 0 จะ Disable การทำงานของ CPU จะเร็วขึ้นด้วย แต่ ถ้าเป็น 1 จะให้สัญญาณ Refresh ระหว่าง Reset จะเป็น 1

**REFW** Refresh Wait เป็น 0 จะให้สัญญาณ Refresh ทุกๆ 2 Clock จะเพิ่ม Refresh Wait เข้าอีกระหว่าง Reset จะเป็น 1

**CYC1,CYC0** Cycle Interval ใช้กำหนดช่วงเวลาในการ Refresh เช่นกรณี Dynamic RAM จะต้อง Refresh 128 ครั้ง ทุกๆ 2 ms ( หรือ 256 ครั้งทุกๆ 4 ms) เพราะฉะนั้นสัญญาณ Refresh แต่ละครั้งจะต้องไม่น้อยกว่า 15.625 usจาก ตาราง ค่าที่ ขีดเส้นใต้เป็นค่าโปรแกรม

**DMA Controller (DMAC)**

มีด้วยกัน 2 Chanel เพื่อเป็นการเพิ่มความเร็วในการ Transfer ข้อมูล โดยการกระทำ ไม่ต้องผ่าน CPU โดยมีความสามารถดังนี้

**Memory Address Space** โดยสามารถกำหนดตำแหน่ง Source และ Destination ที่ใดก็ได้ใน 1024 K Byte Transfer Length ใช้เป็น Counter ในการ Transfer ได้เป็นบล็อค ๆ ละ 64 K Byte

**DREQ** เป็นขา Input จะตรวจจับที่ระดับหรือขอบของสัญญาณ

**IEND** เป็นขา Output เพื่อบอกกับอุปกรณ์ภายนอกว่าทำ DMA หมด Block แล้ว

**Transfer Rate** การ Transfer แต่ละครั้งจะเกิดทุกๆ 6 Clock และ Wait State สามารถเพิ่มเข้าไปใน DMA ได้ สำหรับ Memory หรือ I/O ที่ ทำงานช้าที่ระบบ System CL(0) = 6 MHZ Transfer จะสูงถึง 1 M Byteใน 1วินาที(ไม่มี Wait State)

**ความสามารถของแต่ละ Chanel**

**Chanel 0** สามารถ Transfer Memory Memory, Memory I/O, Memory Memory I/O Map และสามารถให้ Address ใน การ Transfer เพิ่ม,ลดหรือให้คงก็ได้ การ Transfer จะให้เป็นแบบ Cycle Steal (ขโมยเวลาเป็นช่วง) คือ เมื่อ Transfer ครบ 1 หรือ 2 Byte ก็จะคืน Bus ให้ UP จนอุปกรณ์ พร้อมก็จะ Transfer ข้อมูลต่อทำ อย่างนี้ สลับกันไปจนหมด Block ใช้อุปกรณที่ทำงานช้าและ

Burst (Transfer แบบต่อเนื่อง) คือ ทำจนจบ Block จึงจะคืน Bus ให้ Up

**Chanel 1** จะใช้กับ Memory I/O โดย Memory Address เพิ่มหรือลดได้

**DMAC Registor**

**Chanel 0** มี SARO (I/O Address 20H-22H) เป็นที่กำหนด Source Address ได้ถึง 1 M Byte หรือ 4K Byte สำหรับ I/O DARO (I / O Address 23H-25H) ใช้กำหนด Destination 0

**Chanel 1** มี MAR 1 (I/O Address 28H-2AH) ใช้กำหนด Physical Address ได้ถึง 1 M Byte โดย อาจจะให้เป็น Source หรือ Desination ก็ได้ IAR 1 (I/O Address 2BH - 2CH) ใช้กำหนด Address ของ I/O โดยอาจจะให้เป็น Source Destination ก็ได้ SCR1(I/O Address 2EH-2FH ) เช่นเดียวกับ BCR0

**Registor ที่เป็น Common**

DMA Status Registor (DSTAT I/O Address 30H ) ประกอบด้วย

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | DE1 | DE0 | DWE1 | DWE0 | DIE1 | DIE0 | - | DME |
| | R/W | R/W | W | W | R/W | R/W | | R |

**รูปที่ 2.17** DMA Status Registor (DSTAT I/O Address 30 H)

**DE1** DMA Enable Chanel 1 เมื่อ DE1 =1 จะทำให้ DME = 1 DMA Chanel 1 จะถูก Enable และเมื่อการ Transfer สิ้นสุดลง(BCR1=0 ) เมื่อนั้น DE1 จะถูก Clear เป็น 0 และถ้า DMA Interrupt ถูก Enable( DIE1=1)CPU จะถูก Interrupt ระหว่าง Reset DE1 จะถูก Clear

**DE0** DMA Enable Chanel 0 มีการทำงานลักษณะเดียว กับ Chanel 1

**DWE1** DE1 Bit Write Enable เมื่อมีการเขียนข้อมูลเข้าไปที่ DE1ในขณะเดียวกันต้องเขียน DWE1 ด้วย 0 และค่านี้ จะไม่คงอยู่ ตลอดไป และถ้า อ่านจะเป็น 1 เสมอ

**DWE0** DE0 Bit Write Enable เช่นเดียวกับ DWE1

**DIE1** DMA Interrupt Enable Chanel 1 เมื่อ DIE1 ถูก Set เป็น 1 และเมื่อการทำ DMA สิ้นสุดลง(เมื่อ DE1=0)ก็จะเกิด Interrupt ขึ้น แต่ถ้า BIT นี้เป็น 0 จะเป็นการ Disable ระหว่าง Reset Bit นี้จะเป็น 0

**DIE0** DMA Interrupt Enable Chanel 0 ลักษณะเช่นเดียวกับ DIE1

**DME** DMA Main Enable เป็น Bit ที่ใช้บอกการ Enable DMA เมื่อ DE Bit (DE0, DE1) ถูก Set เมื่อนั้น DMA Bit จะถูก Set เป็น 1 ซึ่ง Bit นี้ใช้อ่านอย่างเดียว

### 2.2.10 DMA Mode Registor (DMOD Address I/O 31H)

ใช้กำหนด Address ในการ Transfer ของ Chanel 0 ซึ่งประกอบด้วย

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | - | - | DM1 | DM0 | SM1 | SM0 | MMOD | - |

**รูปที่ 2.18** DMA Mode Registor (DMOD Address I/O 31H)

โดย Bit ของ DM1, DM0 เป็นตัวกำหนดเงื่อนไข Destination ส่วน SM1,SM0 ใช้กำหนดเงื่อนไขของ Source ในระหว่าง Reset 4 Bit นี้จะเป็น 0 การโปรแกรมสามารถจัดเป็น ตารางได้ดังนี้

**ตารางที่ 2.9** Combination of Transfer Mode

| DM1 | DM0 | SM1 | SM0 | Transfer Mode | Address Increment/Decrement |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Memory to Memory | SARO + 1, DARO + 1 |
| 0 | 0 | 0 | 1 | Memory to Memory | SARO – 1, DARO + 1 |
| 0 | 0 | 1 | 0 | Memory* to Memory | SARO fixed, DARO + 1 |
| 0 | 0 | 1 | 1 | I/O to Memory | SARO fixee, DARO + 1 |
| 0 | 1 | 0 | 0 | Memory to Memory | SARO + 1, DARO - 1 |
| 0 | 1 | 0 | 1 | Memory to Memory | SARO – 1, DARO - 1 |
| 0 | 1 | 1 | 0 | Memory to Memory | SARO fixed, DARO - 1 |
| 0 | 1 | 1 | 1 | I/O to Memory | SARO fixed, DARO - 1 |
| 1 | 0 | 0 | 0 | Memory to Memory | SARO + 1, DARO fixed |
| 1 | 0 | 0 | 1 | Memory to Memory | SARO – 1, DARO fixed |
| 1 | 0 | 1 | 0 | reserved | |
| 1 | 0 | 1 | 0 | reserved | |
| 1 | 1 | 0 | 0 | Memory to I/O | SARO + 1, DARO fixed |
| 1 | 1 | 0 | 1 | Memory to I/O | SARO – 1, DARO fixed |
| 1 | 1 | 1 | 0 | reserved | |
| 1 | 1 | 1 | 1 | reserved | |

**MMOD** Memory Mode Chanel 0 เมื่อ Chanel 0 ถูกกำหนดการทำงานเป็น Memory ขา Input ภายนอก DREQ0 จะไม่ถูกใช้ แต่จะถูกแทน ด้วย การ Transfer อัตโนมัติ 2 แบบด้วยกันคือ Burst Mode (MMOD = 1)คือ DMAC จะทำการ Transfer ข้อมูลอย่างต่อเนื่องจนจบ Block ที่กำหนดและ Cycle Steal Mode (MMOD=0) คือ การ Transfer แบบขโมยเวลาของ CPU ซึ่งจะทำ DMA 1 Byte สลับกับ CPU ทำ1 Cycle (1 Machine) ไปเรื่อยจนกว่าจะจบ Block

สำหรับ DMS กับ I/O ที่เป็น Source หรือ Destination ขา Input DREQ0 จะถูกนำมาใช้ และ MMODE จะไม่มีความหมายระหว่าง Reset MMOD จะถูก Clear

**DMA/Wait Control Registor (DCNTL I/O Address 32H)** ประกอบด้วย

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | MWI1 | MWI0 | IWI1 | IWI0 | DMS1 | DMS0 | DIM1 | DIM0 |

**รูปที่ 2.19** DMA/Wait Control Registor (DCNTL I/O Address 32H)

**MWI1, MWI0** Memory Wait State Insertion ใช้กำหนดจำนวน Wait State ของ CPU หรือ DMAC ดู หัวข้อ Wait State Generator

**IWI1, IWI0** I/O Wait Insertion กำหนด Wait State I/O หรือ CPU ดูหัวข้อ Wait State Generator

**DMS1, DMS0** DMA Request Sense ใช้กำหนดการรับรู้ของ ขา Input DREQ0, DREQ1 เมื่อเป็น 0 จะตรวจที่ Level และถ้าเป็น 1 จะตรง ที่ Edge ระหว่าง Reset 2 Bitนี้ จะเป็น 0

**DIM1, DIM0** DMA chanel 1 I/O และ Memory Mode ใช้กำหนด Source และ Destintion ของ Chanel 1 เมื่อ Reset 2 Bit นี้เป็น 0 สามารถ Set ได้ดังตาราง

**ตารางที่ 2.10** DMA Request Sense

| DIM1 | DIN0 | Transfer Mode | Address Increment/Decrement |
|---|---|---|---|
| 0 | 0 | Memory ⟶ I/O | Mar1+1 , Iar 1 Fixed |
| 0 | 1 | Memory ⟶ I/O | Mar1-1 , Iar 1 Fixd |
| 1 | 0 | I/O ⟶ Memory | Iar 1 Fix . Mar1+1 |
| 1 | 1 | I/O ⟶ Memory | Iar 1 Fix . Mar1-1 |

ความสามารถพิเศษของ Chanel 0 คือ DMA กับ ASCI ได้ทั้ง 2 Chanel ในกรณีนี้ DREQ0 จะไม่ถูกใช้ แต่จะใช้ Status Bit ของ ASCI จะเป็นตัวกำเนิด DREQ0 ภายในขึ้นโดย Bit TDRE และ RDRF สำหรับการส่งและรับ โดยให้กำหนด Source และ Destination ที่ SAR0 และ DAR0 ให้เป็น Address ของ I/O ของการส่งและรับ (ASCI Address 6H-9H) และ Bit A8-A15 ให้เป็น 0 และ Bit A17-A16 ต้องโปรแกรมดัง ตาราง

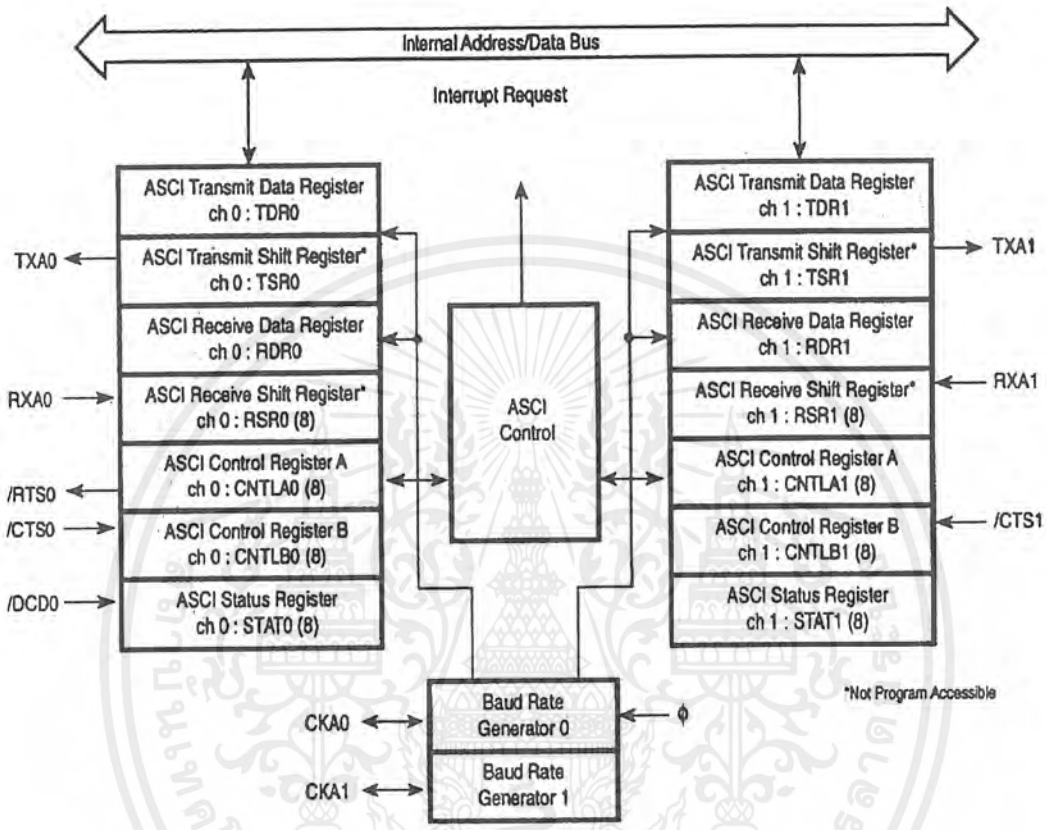
**ตารางที่ 2.11** การทำ DMA Chanel 0 กับ ASCI

| SAR18 | SAR17 | SAR16 | DMA Transfer Request |
|---|---|---|---|
| X | 0 | 0 | DREQ0 |
| X | 0 | 1 | RDRF (ASCI Chanel 0) |
| X | 1 | 0 | RDRF (ASCI Chanel 1) |
| X | 1 | 1 | Reserved |
| DAR18 | DAR17 | DAR16 | DMA Transfer Request |
| X | 0 | 0 | DREQ0 |
| X | 0 | 1 | TDRE (ASCI Chanel 0) |
| X | 1 | 0 | TDRE (ASCI Chanel 1) |
| X | 1 | 1 | Reserved |

จากนั้นกำหนด Mode เป็น "Edge Sense" ส่วน ASCI ตอนเริ่มต้นต้องถูกกำหนดดังนี้ เวลารับต้อง "EMPTY" คือ RDRF = 0 ส่วนการส่งต้อง "FULL" คือ TDRE =(คือ Byte แรกจะถูกกระทำโดย ASCI ) ส่วน Byte ถัดไป DMAC จะเป็นตัวจัดการ ขณะทำ DMA สามารถทำ Interrupt NMI ได้โดย

**Asyncronous Serial Communication Interface (ASCI)**

มีด้วยกัน 2 Chanel Block Diagram ดังรูป




**รูปที่ 2.20** Figure 35 . ASCI Block Diagram

**TSR0,1** เป็น Shift Registor ที่รับข้อมูลจาก Transmit Data Registor (TDR) แล้วนำข้อมูลนั้น SHIFT ออกที่ขา TXA

**TDR 0 , 1** (I/O Address 06H , 07H) เป็น Registor ที่ใช้ส่ง Data ออกไปที่ขา TXA โดยการนำข้อมูลใน TDR ส่งไปที่ TSR เมื่อ TSR ว่างลง และสามารถที่จะเขียนข้อมูลเข้าไปที่ ได้อีกในขณะที่ TSR กำลัง SHIFT ข้อมูลออกไปที่ขา TXA

**RSR 0 , 1** เป็น Registor ที่รับข้อมูลจาก RXA PIN เมื่อรับเต็ม Buffer แล้วก็ SHIFT ไปที่ RDR ถ้า RSR ไม่ว่าง เมื่อมีการรับข้อมูล Byte ต่อไปเข้าอีก จะเกิดข้อมูลทับซ้อนกันขึ้น จะทำให้การผิดพลาด

และผลของการผิดพลาดจะแสดงที่ Registor สถานะ Registor สามารถโปรแกรมได้

**RDR 0 , 1** (I/O Address 08H , 09H) คือ Registor ที่ใช้เก็บข้อมูลที่รับเข้ามาจาก RXA PIN และในขณะที่กำลังบรรจุข้อมูลที่รับเข้ามาจาก RSR ข้อมูล BYTE ถัดไปสามารถรับเข้ามาต่อได้

**STAT 0 , 1** (I/O Address 04H , 05H) แต่ละ Chanel จะมี Registor ใช้สำหรับตรวจสอบการสื่อสารเกี่ยวกับการผิดพลาด และสถานะสัญญาณ Control Modem การ Enable และ Disable ASCI ดังรูป

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RDRF | OVRN | PE | FE | RIE | DCD0 | TDRE | TIE |

**รูปที่ 2.21** STAT 0 (I/O Address 04H)

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | RDRF | OVRN | PE | FE | RIE | CTSIE | TDRE | TIE |

**รูปที่ 2.22** STAT 1 (I/O Address 05H)

**RDRF** Receive Data Registor Full จะถูก Set เป็น 1 เมื่อ ข้อมูลที่รับเข้ามาถูกส่งเข้ามาที่ RDR เรียบร้อยแล้ว (ครบ Byte) แต่ถ้าการรับเกิด error ขึ้น RDRF ก็จะถูก Set ค้างและข้อมูลที่ผิดนั้นก็จะถูกส่งมาที่ RDR และคงอยู่ ดังนั้นจะต้องทำการ Clear Flag Error RDRF จะถูก Clear เป็น 0 , RDR, DCDO เป็น High สำหรับ Chanel 0, IOSTOP และการ Reset

**OVRN** Overun Error จะเป็น 1 เมื่อ RDR เต็มและ RSR เต็ม แล้วยังมีการรับข้อมูล เข้ามาอีก จะถูก Clear ได้ เมื่อ EFR Bitใน CNTLA เป็น 0 , DCD0 High IOSTOP และ Reset

| | |
|---|---|
| **PE** | Parity Error เป็น 1 เมื่อข้อมูลที่รับเข้ามา Parity ผิด และ Clear ได้เช่นเดียวกับ Overun |
| **FE** | Framing Error เมื่อข้อมูลที่รับเข้ามารูปแบบผิดไปจากที่กำหนด Bit FE จะถูก Set เป็น 1 และการ Clear เช่นเดียวกับ Overun |
| **RIE** | Receive Interrupt Enable เมื่อเป็น 1 จะอนุญาติให้ ASCI ทำการขอ Interrupt ได้ เมื่อ RDRF , OVRN , PE หรือ FE ถูก Set เป็น 1 ด้วย เมื่อนั้น ASCI ก็จะให้สัญญาณ Interrupt สำหรับ ASCI Chanel 0 Interrupt สามารถเกิดขึ้นโดยการเปลี่ยนแปลงที่ขารับสัญญาณ Input ภายนอกที่ขา DCD0 จาก Low เป็น High และ RIE จะถูก Clear เป็น 0 ระหว่าง Reset |
| **DCD0** | Data Carrier Detect Bit นี้จะถูก Set เป็น 1 เมื่อขา Input DCD0 เป็น High และจะถูก Clear เป็น 0 จากการอ่าน STATO ครั้งแรก นั้นขา Input DCD0 จะถูกเปลี่ยนจาก High เป็น Low และระหว่าง Reset เมื่อ DCD0 เป็น 1 ส่วนของภาครับจะไม่ทำงาน |
| **CTSIE** | Chanel 1 CTS Enable ที่ Chanel 1 มีขา Input CTS1 ภายนอก ซึ่ง Multiplex กับ RXS เมื่อ Set Bit นี้เป็น 1 จะถูกเลือกเป็นขา CTS1 |
| **TDRE** | Transmit Data Registor Empty เป็นตัวบอกว่าข้อมูลพร้อมที่จะส่งได้หรือไม่ ถ้าเป็น 1 คือ พร้อมที่จะส่งข้อมูลแล้วให้เขียนข้อมูลเข้าไปที่ TDR ได้ และเมื่อมีการเขียนข้อมูลเข้าไปที่ TDR ก็จะทำให้ TDRE เป็น 0 และข้อมูลใน TDR ก็จะถูกส่งให้ TSR จน TDR ว่างลงอีก TDRE ก็จะกลับเป็น 1 อีกครั้ง |
| **TIE** | Transmit Interrupt Enable เมื่อเป็น 1 จะอนุญาตให้ ASCI ใช้การ ส่งแบบ Interrupt ได้ โดยที่ TDRE ต้องเป็น 1 ด้วย TIE จะถูก Clear เป็น 0 ระหว่าง Reset |
| **CNTLA 0,1** | (I/O Address 00H-01H) เป็น Registor กำหนดการทำงานประกอบด้วย |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | MPE | RE | TE | RTS0 | MPBR/EFR | MOD2 | MOD1 | MOD0 |

**รูปที่ 2.23** CNTLA 0 (I/O Address 00H)

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | MPE | RE | TE | CKAID | MPBR/EFR | MOD2 | MOD1 | MOD0 |

**รูปที่ 2.24** CNTLA 1 (I/O Address 01H)

**MPE** Multiprocessor Mode Enable ใช้ Enable ในการสื่อสารแบบไมโครโปรเซสเซอร์ร่วมจากเมื่อมีการเลือก Mode การสื่อสารแล้ว (MP = 1 ใน CNTLB) ในการสื่อสารแบบนี้ Format ของการสื่อสารจะมี Bit พิเศษเพิ่มเข้ามาเรียกว่า MPB Bit ซึ่ง Bit นี้จะถูกใช้ในการตรวจสอบหรือใช้งานเมื่อ Enable MPE ให้เป็น 1 และถ้า MPB = 1 เมื่อนั้นภาครับของ Multiprocessor จะทำงาน คือ RDRF และ Error Flag จะทำงานแต่ถ้า MPB = 0 ASCI จะไม่สนใจข้อมูล Byte นั้น ๆ ถ้า MPE = 0 จะไม่สามารถทำการสื่อสารแบบไมโครโปรเซสเซอร์ร่วมได้ แม้จะเป็น Set MP เป็น 1 แล้วก็ตาม

**RE** Receiver Enable เป็น 1 จะ Enable การรับของ ASCI แต่ถ้าเป็น 0 จะ Disable การรับ แต่ RDRF และ Error Flag จะไม่ถูก Reset ตาม

**TE** Transmit Enable เป็น 1 จะ Enable การส่ง ถ้าเป็น 0 จะ Disable แต่ TDRE Flag ถูก Reset ตาม

**RTS0** Request to Send Chanel 0 เป็น Bit ที่ให้ผลเช่นเดียวกับขา Output RTS0 คือ ถ้า Bit นี้เป็น 1 ขา Output RTS0 ก็จะเป็น 1 ถ้า Bit นี้เป็น 0 ขา Output ก็เป็น 0 RTS0 Bit นี้จะถูก Set เป็น 1 ระหว่าง Reset

**CKA1D** CKA1 Clock Disable ซึ่งขา CKA1 จะ Multiplex กับ TEND0 เมื่อ Bit นี้เป็น 1 จะเลือกเป็นขา TEND0 แต่ถ้าเป็น 0 ก็จะเป็นขา Clock ของ ASCI Chanel 1 Bit นี้จะเป็น 0 ระหว่าง Reset

MPBR/EFR Multiprocessor Bit Receive/Error Flag เมื่อ Multiproces Mode ถูกเลือก (MP ใน CNTLB = 1 ) เมื่ออ่าน Bit นี้จะเป็นค่าใน Bit ของ MPB แต่ถ้าเขียน 0 ให้ Bit นี้จะเป็นการ Reset Error Flag ในการรับ

MOD 2,1,0 ; ASCI Data Format Mode 2,1,0 โดย

MOD 2 = 0 7 Bit , 1 = 8 Bit MOD1 = 0 Noparity, 1 = Parity Enable

MOD 0 = 0 1 Stop Bit ,1=2 Stop Bit สรุปได้ดังตาราง

**ตารางที่ 2.12** MPBR / EFR Multiprocessor Bit Receive / error

| MOD 2 | MOD1 | MOD0 | Data Format |
|---|---|---|---|
| 0 | 0 | 0 | Start + 7 Bit Data +1 Stop |
| 0 | 0 | 1 | Start + 7 Bit Data +2 Stop |
| 0 | 1 | 0 | Start + 7 Bit Data + Parity + 1 Stop |
| 0 | 1 | 1 | Start + 7 Bit Data + Parity + 2 Stop |
| 1 | 0 | 0 | Start + 8 Bit Data +1 Stop |
| 1 | 0 | 1 | Start + 8 Bit Data +2 Stop |
| 1 | 1 | 0 | Start + 8 Bit Data + Parity + 1 Stop |
| 1 | 1 | 1 | Start+ 8 Bit Data + Parity + 2 Stop |

**ASCI Control Registor B0 , 1 (CNTLB 0, 1 I/O Address 02H, 03H) ประกอบด้วย**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | MPBT | MP | CTS/PS | PE0 | DR | SS2 | SS1 | SS0 |

**รูปที่ 2.25** CNTLB 0 ,1 I/O Address 02H , 03H

**MPBT** Multiprocessor Bit Transmit ใช้ส่ง MPB Bit โดย ถ้า MPBT = 0 เมื่อ นั้น MPB Bit = 1 และ MPBT = 0 MPB ก็ = 0 ด้วย ระหว่าง Reset ไม่สามารถกำหนดได้

**MP** Multiprocessor Mode ถ้าเป็น 1 จะเป็นการ Set การติดต่อแบบ ไมโครโปรเซสเซอร์ร่วมโดยใช้ FORMAT ของ MOD 2 กับ MOD 0 โดยยกเว้น MOD 1 ดังนี้ Start BIT + 7 หรือ 8 Data Bit + MPB Bit + 1 หรือ 2 Stop Bit ระหว่าง Reset MP จะเป็น 0

**CTS/PS** Clear to Send / Prescale เมื่ออ่าน Bit นี้จะใช้แสดงสถานะของขา Input CTS ภายนอก ถ้าขา CTS เป็น High ภาคส่งของ ASCI จะไม่ทำงานแต่ถ้าเขียนเข้าไปที่ Bit นี้จะเป็นการกำหนด Baud Rate Bit นี้เป็น 0 ระหว่าง Reset

**PEO** Parity Even Odd Bit นี้จะไม่มีผลต่อการ Enable หรือ Disable ของ Parity (MOD1 ใน CNTLA) แต่จะใช้เลือกว่าเมื่อ มีการ Enable Parity ใน MOD 1 จะให้ Parity คู่หรือคี่ ถ้า PEO = 0 คือคู่ แต่ถ้า = 1 คือคี่

**DR** Divide Ratio ใช้กำหนด Band Rate Bit นี้จะเป็น 0 Reset

**SS2, 1, 0** Source/Speed Select 2, 1, 0 ใช้กำหนด Clock ว่าจะให้เป็นภายในหรือภายนอก (โดยภายนอกคือ ขา Clock CKA) และเป็นตัวกำหนด Baud Rate ด้วย ระหว่าง Reset ทั้ง 3 Bit นี้จะเป็น 1 คือ เป็นการใช้ Clock จากภายนอกนั่นเองซึ่งจากที่กล่าวมาในการกำหนด Baud Rate จึงมีด้วยกันหลายตัว

### 2.1.11 Clock Serial I/O Port (CSI/O)

มี 1 Chanel ซึ่งเป็น Synchronous Serial I/O Port โดยใช้ได้เฉพาะเป็น Half-Duplex เท่านั้น และ Data ถูกกำหนดเป็น 8 Bit โดย Clock ที่ใช้ในการซิงค์เลือกได้ว่าจะใช้จาก System Clock หรือ Clock ภายนอกที่ขา (CKS) ก็ได้ ซึ่ง CSI/O ประกอบไปด้วย 2 Registor คือ

**1) CSI/O Transmit / Receive Data Registor (TRDR I/O Address 0BH)**

ใช้ในการส่งและรับข้อมูล โดยระบบต้องเป็น Half-Duplex (คือ การส่งและรับจะเกิดพร้อมกันไม่ได้)

**2.) CSI/O Control /Status Registor (CNTR I/O Address 0AH)**

เป็นตัวบอกสถานะและ Control CSI/O ประกอบด้วย

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | EF | EIE | RE | TE | - | SS2 | SS1 | SS0 |

**รูปที่ 2.26** CSI I/O Control Register Address 0AH

**EF** End Flag เป็น 1 เมื่อ CSI/O รับหรือส่งข้อมูลครบ 8 BIT แล้ว ซึ่งถ้า EIE ถูก Set เป็น 1 ไว้ก็จะทำให้ CSI/O ขอ Interrupt ได้ ระหว่าง Reset Bit นี้จะเป็น 0 และใน OSTOP Mode ด้วย

**EIE** End Interrupt Enable เมื่อเป็น 1 จะเป็นการ Enable Interrupt และจะ Interrupt เมื่อ EF = 1 ด้วย ระหว่าง Reset EIE = 0

**RE** Receive Enable ภาครับจะทำงานเมื่อ RE = 1 โดยข้อมูลจากขา RXS จะถูก Shift เข้ามาที่ TRDR โดยข้อมูลที่เข้าทาง RXS จะซิงค์กับสัญญาณ Clock ซึ่งจะเป็นภายในหรือภายนอก โดยการเลือก ถ้าเป็นภายในสัญญาณ Clock ที่ทำการหารแล้วก็จะออกที่ขา CKS ด้วย หรือถ้าเป็นภายนอก ขา CKS ก็จะเป็นตัวรับสัญญาณ Clock เพื่อใช้ซิงค์นั่นเอง หลังจากที่ CSI/O รับข้อมูลครบ 8 Bit แล้วก็จะ Clear RE โดยอัตโนมัติ และ EF จะถูก Set เป็น 1 ถ้าเกิด Set EIE ไว้ก็จะเกิด Interrupt ขึ้นได้ และ RE กับ TE จะต้องไม่เป็น 1 พร้อมกัน

**TE** Transmit Enable ลักษณะการทำงานเช่นเดียวกับ RE แต่ TE นี้จะใช้ในการส่ง

**SS2, 1, 0** Speed Select2, 1, 0 ใช้เลือก Clock ในการรับส่ง

**ตารางที่ 2.13** แสดง Baud Rate เมื่อทำการเลือก Speed Select

| SS2 | SS1 | SS0 | Divide Ratio | Baud Rate |
|---|---|---|---|---|
| 0 | 0 | 0 | -20 | (200000) |
| 0 | 0 | 1 | -40 | (100000) |
| 0 | 1 | 0 | -80 | (50000) |
| 0 | 1 | 1 | -160 | (25000) |
| 1 | 0 | 0 | -320 | (12500) |
| 1 | 0 | 1 | -640 | (6250) |
| 1 | 1 | 0 | -1280 | (3125) |
| 1 | 1 | 1 | Extenal Clock Input ( Less Than – 20 ) | |

### 2.2.12 Programmable Relode Timer (PRT)

มีด้วยกัน 2 Chanel เป็น 16 Bit Programmable Relode Timer และสำหรับ Chanel 1 มีขา Output สามารถให้สัญญาณได้ทั้ง 2 Chanel ประกอบด้วย

**1) Timer Data Registor (TMDR : I/O Address-CH0 ; 0DH, 0CH, CH1; 15H, 14H)**

เป็น Registor 16 Bit ใช้กำหนด Timer โดย Address I/O สูงเก็บค่า Timer ค่าสูงระหว่าง Reset TMDR0 และ TMDR1 จะเป็น 0FFFFH โดย TMDR จะนับลง 1 ครั้งทุก ๆ 20 Clock System เมื่อ TMDR นับลงเป็น 0 ค่าใน Reload จะถูก Load มาให้ TMDR โดยอัตโนมัติ การอ่านค่าใน TMDR อ่านได้เลยโดยไม่ต้องหยุด PRT แต่ถ้าเป็นการเขียนต้องหยุด PRT ก่อน

**2) Timer Reload Registor (RLDR : I/O Address-CH0 ; 0FH, 0EH , CH1 ;17H, 16H)**

ใช้ Load ค่าที่อยู่ใน RLDR ไปให้ TMDR เมื่อ TMDR ลดลงเป็น 0 Timer Control Registor (TCR : I/O Address 10H) เป็น Registor ใช้แสดงสถานะและ Control

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| | TIF1 | TIF0 | TIE1 | TIE0 | TOC1 | TOC0 | TDE1 | TDE0 |

**รูปที่ 2.27** TCR Address 10H

**TIF1** Timer Interrupt Flag 1 เมื่อ TMDR 1 ลดลงเป็น 0 TIF จะถูก Set เป็น 1 และถ้า TIE 1 = 1 ก็จะทำให้เกิด Interrupt ขึ้นได้ TIF จะถูก Clear เป็น 0 ก็ต่อเมื่อทำการอ่านค่าใน TCR กับอ่านค่าใน Byte High หรือ Low ของ TMDR1 ระหว่าง Reset TIF = 0

**TIF 0** Timer Interrupt Flag หลักการเช่นเดียวกับ TIF1

**TIF 1, 0** Timer Interrupt Enable 0, 1 เมื่อ SET เป็น 1 จะอนุญาตให้ Interrupt ได้ระหว่าง Reset 2 Bit นี้จะเป็น 0

**TOC 1, 0** Timer Output Control 2 Bit นี้ใช้ควบคุมขา Output ของ PRT 1 โดยถ้าทั้ง 2 Bit นี้เป็น 0 จะเป็นการใช้งาน A 18 นอกจากนั้นจะเป็นการกำหนดให้ TOUT เป็น High, Low หรือ Toggle ดังตาราง ระหว่าง Reset 2 Bit นี้จะเป็น 0

**ตารางที่ 2.14** Timer Output Control <TOC1,TOC0>

| TOC1 | TOC0 | Output |
|---|---|---|
| 0 | 0 | Address A18 |
| 0 | 1 | Toggle |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**TDE 1, 0** Timer Down Count Enable เมื่อ Set เป็น 1 ก็คือให้เริ่มทำการนับ TMDR ได้ แต่ถ้าเป็น 0 การนับจะหยุดทำงานระหว่าง Reset Bit ทั้งคู่นี้จะเป็น 0

การคำนวณเวลาเราทราบแล้วว่า Timer จะนับลงทุก ๆ 20 Clock ดังนั้นถ้า X'TAL ที่ใช้งาน คือ 12 MHz ความถี่ RUN บน BOARD=6 MHz หาค่า Timer คือ

T = 1/F = 0.1666 US ต่อ 1 Clock

20 Clock = 0.1666 x 20 = 3.333 US

นั่นคือ Timer นับลง 1 ครั้ง ทุก ๆ 3.333 US ที่ X'TAL 12 MHz เช่นให้ TMDR มีค่า =1 และ Set Flag Interrrupt ไว้ก็จะทำให้ PRT เกิดการ Interrupt ทุก ๆ 6.666 US เพราะการนับลงจะนับจากตัวเองลงก่อนคือ 1 แล้วก็ 0 จึงเท่ากับ 2 ครั้งนั่นเอง และถ้าให้กำเนิดสัญญาณสแควที่ TOUT ก็จะทำให้เกิดการ Toggle กันทุก ๆ จำนวนที่ให้นับ เช่น จากตัวอย่างข้างบนก็จะเป็น High 6.666 US และ LOW 6.666 US ดังนั้น 1 ลูก สัญญาณจะประมาณ 13 US หรือความถี่จะต่ำลงเท่าหนึ่งของค่าเวลาที่คิดจากจำนวนครั้งในการนับค่าก็ได้

### 2.2.13 Secondary Bus Interface

E Clock Output Timing เป็นสัญญาณ Bus ที่ 2 เพื่อใช้เชื่อมต่อ Interface เป็นไปได้ง่ายกับอุปกรณ์ Periheral ในตระกูลอื่น ๆ เช่น 68XX และ 80XX และเป็นสัญญาณที่ทำให้ระบบเกิดความน่าเชื่อถือในการใช้งาน เพราะจะติดต่อกับอุปกรณ์ก็ต่อเมื่อมีสัญญาณที่ขานี้ จากรูปขณะที่ MREQ หรือ IORQ จะเกิดขึ้นช่วงที่ T State แรกซึ่งยังไม่ใช่ช่วงของ Data ที่อ่านหรือเขียน จึงทำให้อาจเกิดข้อมูลผิดพลาดกับอุปกรณ์ภายนอก แต่สัญญาณจะให้สัญญาณ Active High เมื่อมีการจ่ายหรือรับ Data เท่านั้น

### 2.2.14 Free Running Counter (18H)

เป็น Register I/O ที่ใช้อ่านได้อย่างเดียวใช้สำหรับการ Refresh Dynamic RAM ซึ่งเป็น Counter นับลง 8 Bit (A0-A7) แบบอิสระโดยจะนับลง 1 ครั้งทุก ๆ 10 Clock และถ้าเกิดมีการเขียนข้อมูลไปที่ Registor นี้ จะทำให้ช่วงเวลาของการ Refresh Dynamic RAM, Baud Rate ของ ASCI และ CSI/O ไม่แน่นอน (คือจะไม่ถูกรับประกันว่าตรงตามที่กำหนดในคู่มือ) ถึงแม้อยู่ใน IO Stop Mode ก็ตาม Free Running Counter นี้ก็ยังนับอยู่อย่างต่อเนื่องซึ่งในขณะ Reset จะมีค่าเป็น 0FFH

### 2.2.15 คำสั่งเพิ่มเติม 9 คำสั่ง

**SLP** เมื่อใช้คำสั่งนี้ CPU จะหยุดทำงานบางอย่าง ทำให้ใช้กำลังงานต่ำ

**MLT** Multiply ใช้สำหรับคูณเลข 8 Bit 2 จำนวน โดยผลลัพธ์จะเป็น 16 Bit โดย Registor ที่ใช้ในการคูณอาจจะเป็น BC, DE, HL หรือ SP โดยผลลัพธ์จะได้ที่ Registor คู่นั้น

**OTIM, OTIMR, OTDM, OTDMR-Block I/O**

เป็นคำสั่ง Out Port เป็น Block ของ Port Address ต่ำ A0-A7 เท่านั้น คือ จะทำการ Out ข้อมูลเป็น Block โดยที่ Port เพิ่มหรือลดตามจำนวนข้อมูล โดยใช้ HL เป็นตัวชี้ข้อมูลที่จะ Out ออกไป และ C เป็น Number Port ในคำสั่ง OTIM และ OTDM ก็ คือจะเพิ่ม ค่า HL ที่ชี้ขึ้นเป็นหนึ่งหรือลดลง 1 ตาม ด้วย Port เพิ่มขึ้นหรือลดลงด้วยและค่า B จะลดลง 1 ซึ่ง B จะเป็น Counter ในการส่ง Data ส่วน OTIMR และ OTDMR จะมีลักษณะเช่นเดียวกับ OTIM และ OTDM เพียงแต่จะทำการส่งข้อมูลเพิ่มขึ้นหรือลดและ Port Number เพิ่มขึ้นหรือลดลงตามค่า จนกระทั่ง B = 0

**TSTIO m** ใช้สำหรับ Test I/O Port คือ จะทำการอ่านค่า Port ที่กำหนดโดย Registor c เข้ามาแล้วทำการ And กับ Data 8 Bit ที่ต้องการ โดยที่ค่าข้อมูลที่ IN เข้ามานั้นไม่เปลี่ยนแปลง แต่จะให้ผลที่ Flag และ Port ที่ IN เข้ามาจะเป็นเฉพาะ Address ต่ำ A0-A7 เท่านั้น

**TST g-Test Registor**

โดยค่าที่กำหนดใน Registor จะ AND กับ Accumulator ซึ่งจะทำให้มีผลต่อ Flag ตามคำสั่ง AND แต่ค่าใน Accumulator และ Registor ไม่เปลี่ยนแปลง

**TST m-Test Immediate**

เช่นเดียวกับ Registor เพียงแต่ข้อมูลเป็น Data โดยตรงที่ AND กับ Accumulator

**TST (HL) - Test Memory**

คือ จะนำค่าใน Memory ที่ถูกชี้โดย HLAND กับ Accumulator โดยค่าทั้ง 2 ไม่เปลี่ยนแปลงแต่ให้ผลการกระทำที่ Flag

**IN0 g(m) Input, Immediate I/O**

In ค่าจาก Port 8 Bit (A0 – A7 ) มายัง Registor ใดๆ ก็ได้

A,BC,DE,HL

**OUTO (m),g – Output , Immediate I/O**

Out ค่าจาก Registor ใดๆ ไปยัง Port 8 Bit (A0 – A7)

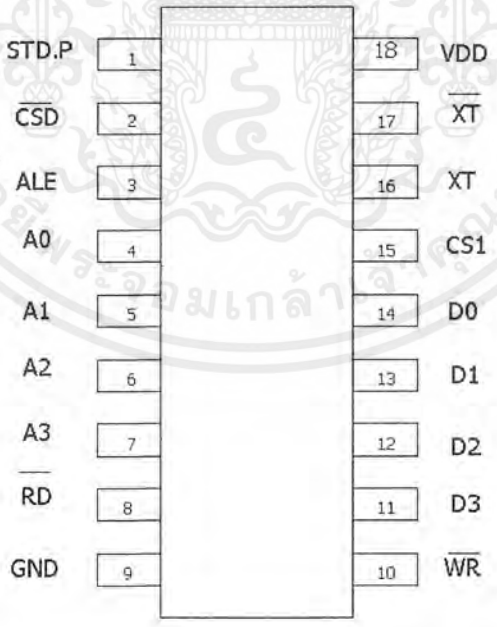
Registor ก็มี A,BC,DE,HL

## 2.3 Real Time Clock

ในการทำไมโครโปรเซสเซอร์ไปใช้ในงานที่มีความเกี่ยวข้องกับเรื่องของเวลานั้น RTC เหมาะที่จะใช้กับงานลักษณะนี้เพราะสามารถบอกได้ทั้ง วัน เดือน ปี วันในรอบสัปดาห์ ชั่วโมง นาที วินาที ในการติดต่อกับ RTC นั้นเปรียบเสมือนการติดต่อกับพอร์ทอินพุท/เอาท์พุท คือเราสามารถเขียนข้อมูลเกี่ยวกับเวลาไปที่เบอร์พอร์ทของ RTC ตัวนับเวลาภายในก็จะเดินตามเวลาที่เราตั้งให้ และเราก็สามารถอ่านข้อมูลจาก RTC ได้เช่นกัน

MSM 6242B สามารถที่จะปรับวันที่ให้ถูกต้องกับเดือนได้ไม่ว่าจะเป็นเดือนที่ลงท้ายด้วย "คม" ลงท้ายด้วย "ยน" หรือแม้กระทั่งเดือนกุมภาพันธ์ซึ่งปกติจะมี 28 วัน แต่ในปีอธิกสุรทินเดือนกุมภาพันธ์จะมี 29 วัน MSM 6242B ก็สามารถปรับวันที่ได้อย่างถูกต้อง

MSM 6462B เป็นไอซี Real Time Clock / Calendar ชนิด CMOS ใช้ต่อกับบัสของไมโครโปรเซสเซอร์ / ไมโครคอมพิวเตอร์ ได้โดยตรงมี Address Bus และ Data Bus ขนาด 4 บิทมี Control Rrgister ขนาด 4 บิท 3 ตัว คือ CD, CE, CF MSM 6242B โดยปกติจะทำงานที่ 5 V +10% ที่ -30 ถึง 25 C มี PACKAGE 3 แบบคือ 18 Pin Plastic Dip , 24 Pin Astic Flat Package และแบบ 12 Pin Plccpackage การจัดขาต่าง ๆ ดังแสดงในรูปที่ 2.31




**รูปที่ 2.28** แสดงการจัดวางของ MSM 6242B

**18 Pin Plastic dip Package**

| | |
|---|---|
| **A0 - A3** | : Address input |
| **D0 -D3** | : Data input/output |
| **CS0 , CS1** | : CHIP SELECT 0,1 |
| **RD** | : READ enable |
| **WR** | : WRITE enable |
| **ALE** | : Address latch enable |
| **STD.P** | : Standard pulse output |
| **XT,XT** | : XTAL oscillator input / output |
| **VDD** | : +5V supply |
| **GND** | : Ground |

### 2.3.1 การจัดขาและหน้าที่ของขาต่าง ๆ

MSM 6242B ได้ถูกออกแบบให้มาอินเตอร์เฟสเข้ากับ CPU ในตระกูล 8085 , MCS 48 , Z80 และ Z80180 ได้ด้วย สำหรับหน้าที่ของขาต่าง ๆ มีดังนี้

**D0 - D3 (Data Bus)** เป็นบัสข้อมูลอินพุท / เอาท์พุทสามารถต่อเข้ากับบัสของไมโครคอนโทรลเลอร์ได้โดยตรงใช้ในการอ่าน และเขียนข้อมูลของรีจิสเตอร์ภายในที่เป็นนาฬิกาปฏิทินและรีจิสเตอร์ควบคุมโดย D0 = LSB และ D3 = MSB

**A0 - A3 (Address Bus)** เป็นบัสแอดเดรสสำหรับติดต่อกับรีจิสเตอร์ภายในของ RTC เพื่อที่จะเขียนหรืออ่านข้อมูลในตำแหน่งนั้น ตำแหน่งของรีจิสเตอร์ต่าง ๆ ดูได้ในตารางที่ 2.10 A0 - A3 จะใช้ร่วมกับ ALE สำหรับการอ้างตำแหน่งรีจิสเตอร์

**ALE (Address Latch Enable)** เมื่อ CS0 = 0 และ ALE เปลี่ยนจากลอจิก "1" ไปเป็นลอจิก "0" แอดเดรสจะถูกแลทซ์เอาไว้ในตัวของ RTC Microcontroller / Microprocessors ที่มีขา ALE เป็นเอาท์พุทควรจะต่อเข้ากับขา ALE ของ RTC ด้วย แต่ถ้าไม่มีขา ALE ให้ต่อขา ALE ของ MSM 6242B เข้ากับ VDD

**WR (Write Enable)** ใช้เขียนข้อมูลเข้าไปในรีจิสเตอร์ของ RTC แอคตีฟที่ลอจิก "0" โดยที่ CS1 = 1 และ CS0 = 0

**RD (Read Enable)** ใช้อ่านข้อมูลจากรีจิสเตอร์ของ RTC แอคตีฟที่ลอจิก "0" โดยที่ CS1 = 1 และ CS0 = 0 และ RD กับ WR จะต้องไม่แอคตีฟพร้อมกัน

**CS0,CS1 (Chip select 0,1)** เป็น Chip Select ทำหน้าที่ Enable/Disable การทำงานของ ALE, RD และ WR โดยที่ CS0 และ ALE จะทำงานร่วมกัน ส่วน CS1 กับ ALE จะทำงานแยกกัน

**STD.P (Standard Pulse Output)** เป็นขาเอาท์พุทชนิด N - CH Open Drain ใช้ต่อกับขาอินเตอร์รัพท์ของ CPU

**XT,XT (XTAL Oscillator Input/Output)** ต่อเข้ากับตัวคริสตอล 32.768 KHz ถ้าต้องการป้อนความถี่จากภายนอก 32.768 KHz ทำได้โดยป้อนความถี่เข้าที่ขา XT ถ้าความถี่มาจากเอาท์พุทของไอซี TTL ควรต่อ R Pull Up ไว้ด้วย ส่วนขา XT ควรปล่อยลอยไว้

**VDD** เป็นขา Power Supply +2 ~ +6v

**GND** ขา Ground

### 2.3.2 รีจิสเตอร์ต่าง ๆ

S1 , S10 , MI1 , MI10 , H1 , H10 , D1 , D10 , MO1 , MO10 , Y1 , Y10 , W

กลุ่มอักษรเหล่านี้เป็นชื่อย่อของรีจิสเตอร์ตามลำดับคือ Second1 , Second10 , Minute1 , Minute10 , Hour1 , Hour10 , Day1 , Day1 , Month1 , Month10 , Year1 ,Year10 และ Week ในการกำหนดค่ารีจิสเตอร์เหล่านี้จะต้องให้เป็นรหัส BCD ตัวอย่างเช่น รีจิสเตอร์ S1 (S8, S4, S2, S1) = 1001 ซึ่งจะหมายถึง 9 วินาที

PM/AM, h20, h10 ในโหมด 24 ชม. บิท PM/AM จะไม่ใช้และจะอ่านบิทนี้ได้เป็น "0" ตลอดในขณะที่อยู่ในโหมด 12 ชม. บิท h20 จะถูกเซทในการอ่าน ถ้าบิท h0 ถูกเขียนด้วย "0" บิทนี้จะอ่านค่าได้เป็น "0" ตลอด ถ้าไม่มีการเขียน "1" เข้าไปที่บิทนี้

MSM 6242B ได้ถูกออกแบบมาสำหรับปีคริสต์ศักราช และยังสามารถจัดการเกี่ยวกับปีอธิกสุรทิน (Leap Year) ได้อย่างอัตโนมัติ

ส่วนรีจิสเตอร์ W สามารถมีข้อมูลได้ตั้งแต่ 0-6 ดูได้จากตาราง 2.15

**ตารางที่ 2.15** แสดงข้อมูลของรีจิสเตอร์

| W4 | W2 | W1 | Day of Week |
|---|---|---|---|
| 0 | 0 | 0 | Sunday |
| 0 | 0 | 1 | Monday |
| 0 | 1 | 0 | Tuesday |
| 0 | 1 | 1 | Wednesday |
| 1 | 0 | 0 | Thursday |
| 1 | 0 | 1 | Friday |
| 1 | 1 | 0 | Saturday |

**CD Register (Control d Register)**

**Hold (DO)** เมื่อเซ็ทบิทนี้เป็น "1" สัญญาณ Clock 1 Hz ที่จะเข้ามาที่ S1 จะถูกหยุดไว้ ในเวลานี้เองบิท D1 (Busy) ซึ่งเป็นบิทสถานะจะสามารถอ่านได้ เมื่อ Busy เท่ากับ "0" รีจิสเตอร์ S1 ~ W สามารถอ่านหรือเขียนได้ ถ้าในช่วงเวลานี้มีตัวทดเกิดขึ้นที่วงจรนับของ S1 จะมีผลทำให้หลักหน่วยของวินาทีมีค่าเพิ่มขึ้นอีก 1 วินาที หลังจากที่ Hold = 0 เมื่อ CS1 = 0 จะทำให้ Hold = 0 โดยไม่สนใจสภาวะการณ์อื่น ๆ

**Busy (D1)** เป็นบิทที่แสดงสถานะของการอินเตอร์เฟสกับไมโครคอนโทรลเลอร์/ไมโครโปรเซสเซอร์บิทนี้สามารถอ่านได้อย่างเดียวเท่านั้น เมื่อบิท Busy = 0 หมายถึงพร้อมที่จะให้อ่านหรือเขียนกับรีจิสเตอร์ S1 W (Address 0 C)

**IRQ Flag (D2)** บิทสถานะนี้จะสัมพันธ์กับระดับสัญญาณของขา STD.P = 0 แล้ว IRQ Flag จะเท่ากับ "0" IRQ Flag จะเป็นตัวบอกไมโครคอมพิวเตอร์ว่า การอินเตอร์รัพท์เกิดจาก MSM 6242B (เมื่อ IRQ Flag = 1) บิทสถานะ IRQ Flag จะทำงานร่วมกับรีจิสเตอร์ตัวอื่นอีกดังต่อไปนี้

รีจิสเตอร์ CE บิท D0 (Mask) เมื่อ D0 (Mask) = 0 STD.P จะเปิดในทางตรงกันข้าม ถ้า D0 (Mask) = 0 STD.P เปลี่ยนสถานะตามเวลาที่กำหนดโดย D3 (T1) และ D2 (T0) ของรีจิสเตอร์ E

เมื่อบิท D1 (Intrpt/Stnd) ของรีจิสเตอร์ E = 1 (อินเตอร์รัพท์โหมด) เมื่อเกิดการอินเตอร์รัพท์เกิดขึ้น STD.P จะเป็น Low จนกว่าบิท IRQ Flag จะถูกเขียนด้วย "0" และถ้า IRQ Flag เป็น "1" อยู่ และเกิดการอินเตอร์รัพท์ใหม่ขึ้นจะไม่มีผลต่อ STD.P (เป็น LOW เหมือนเดิม)

เมื่อบิท D1 (Intrpt/Stnd) ของรีจิสเตอร์ E = 0 (Standard Pulse output Mode) เมื่อเกิดการอินเตอร์รัพท์ STD.P จะยังคงเป็น Low จนกระทั่งบิท IRQ Flag ถูกเขียนด้วย "0" หรือเมื่อเวลาผ่านไป 7.8125 ms STD.P จะกลับเป็น HI โดยอัตโนมัติ

เมื่อมีการเขียนไปที่บิท Hold หรือ 30 SEC Adjust ของรีจิสเตอร์ D จำเป็นจะต้องเขียนไปที่บิท IRQ Flag ด้วย

**30 ADJ (D3)** ถ้าเซ็ทบิทนี้ให้เป็น "1" ในขณะที่เราเซ็ทเวลานั้น ถ้าหลักวินาทีนับไปได้น้อยกว่า 30 วินาทีจะมีผลทำให้หลักวินาทีถูกเซ็ทเป็น "00" วินาที แต่ถ้าเกิดว่าหลักวินาทีนับไปได้มากกว่าหรือเท่ากับ 30 วินาทีจะมีผลทำให้หลักนาทีเพิ่มค่าขึ้นอีก 1 นาที แล้วหลักวินาทีจะถูกเซ็ทให้เป็น "00" วินาที ในขณะที่เซ็ทบินนี้เป็น "1" นั้นไม่ควรที่จะอ่านหรือเขียนในเวลา 125 us หลังจากนั้นบิทนี้จะถูกเซ็ทเป็น "1" มันจะเปลี่ยนกลับมาเป็น "0" อย่างอัตโนมัติหลังจากนั้นก็สามารถอ่านหรือเขียนข้อมูลในรีจิสเตอร์ได้

**CE Register (Control E Register)**

**Mask (D0)** เป็นบิทที่ใช้ควบคุมเอาท์พุทของ STD.P เมื่อ MASK = 1 จะมีผลทำให้ STD.P = 1 (OPEN) คือ ไม่สามารถใช้บิทอื่นมาเปลี่ยนสภาวะของ STD.P ได้ และเมื่อให้ Mask = 0 ก็จะทำให้ STD.P = Output Mode นั่นคือบิทอื่น ๆ สามารถ ควบคุมเอาท์พุทของ STD.P ได้ตามต้องการ ความสัมพันธ์ระหว่าง Mask บิทกับเอาท์พุทของ STD.P ดังแสดงในรูปที่ 4

**INTRPT/STND (D1)** ใช้เป็นตัวเลือกสัญญาณเอาท์พุทของ STD.P ได้ 2 โหมด คือ อินเตอร์รัพท์กับ Standard Timing Waveforms (ผลิตพัลซ์ออกมาด้วยคาบเวลาที่แน่นอน)

ถ้า INTRPT/STND = 1 และ MASK = 0 เมื่อเกิดการอินเตอร์รัพท์จาก RTC เอาท์พุทของ STD.P จะให้ลอจิก Low จนกว่าจะเขียน "0" ไปที่ IRQ FLAG ในรีจิสเตอร์ C

ถ้า INTRPT/STND = 0 และ MASK = 0 จะส่งพัลซ์ออกไปที่ขาเอาท์พุทของ STD.P โดยมี T1 และ T0 เป็นตัวกำหนดคาบเวลาในการอินเตอร์รัพท์ และมี LOW-Level Pulse Width ออกมาที่ขา STD.P ถ้าไม่มีการเขียนลอจิก "0" ไปที่ IRQ FLAG ความกว้างของพัลซ์จะเท่ากับ 7.8125 ms

**T0 (D2) , T3 (D3)** บิททั้ง 2 นี้จะเป็นตัวกำหนดคาบเวลาของสัญญาณเอาท์พุท ของ STD.P ของทั้ง 2 โหมด คือ Interrupt และ Fixed Timing Waveform ตารางข้างล่างจะแสดงถึงคาบเวลาซึ่งมี t0 และ t1 เป็นบิทอินพุทซึ่งมีส่วน
เกี่ยวข้องกับ STD.P กับ INTRPT/STND

**ตารางที่ 2.16** แสดงคาบเวลาที่อินพุทคือ T1 และ T0

| T1 | T0 | Periop | Duty Cycle of "0" Level When INTRPT/STND Bit is "0" |
|---|---|---|---|
| 0 | 0 | 1/64 Secound | 1/2 |
| 0 | 1 | Second | 1/128 |
| 1 | 0 | Minute | 1/7680 |
| 1 | 1 | Hour | 1/460800 |

ถ้าเราให้บิท INTRPT/STND เป็น "1" ความสัมพันธ์ระหว่าง STD.P กับ Mask Bit จะเป็นดังนี้ คือ ถ้า STD.P เป็น "0" อยู่ก่อนแล้วเราเขียนให้ Mask Bit เป็น "1" ก็จะมีผลทำให้ STD.P เปลี่ยนเป็น "1" และในขณะที่ STD.P เป็น "1" อยู่นั้น (โดยการ Set ให้ IRQ Flag เป็น "0") แล้วเราให้ Mask Bit เป็น "1" ก่อนที่ STD.P จะเปลี่ยนมาเป็น "0" นั้นก็จะมีผลทำให้ STD.P เป็น "1"

ถ้าเราให้บิท INTRPT/STND เป็น "0" ความสัมพันธ์ระหว่าง STD.P กับ Mask Bit จะเป็นดังนี้คือ ถ้า STD.P เป็น "0" อยู่ก่อนแล้วเราเขียนให้ Mask Bit เป็น "1" ก็จะไม่มีผลอะไรเกิดขึ้นและในขณะที่ STD.P เป็น "1" อยู่นั้นแล้วเราให้ Mask Bit เป็น "1" ก่อนที่ STD.P จะเปลี่ยนมาเป็น 0

**CF Register (Control F Register)**

**RESET (D0)** บิทนี้จะใช้ในการเคลียร์ Clock ภายในที่ใช้ในการนับ/หารของวินาที เมื่อ Rest = 1 จะทำให้ STD.P = 1 และวงจรนับภายในจะถูกรีเซ็ทและเมื่อต้องการให้วงจรนับ

ภายในทำงานต่อ (ออกจากการ Reset) จำเป็นจะต้องให้ Rest = 1 ถ้า CS1 = 1 ดังนั้น Rest = 1 อย่างอัตโนมัติ

**STOP (D1)** จะใช้ในการหยุดตัวทดที่จะเข้าไปในวงจรหารความถี่ 8192 Hz และจะมีการหน่วงเวลาไป 122 us ก่อนที่เวลาจะทำการเดินหรือหยุดเดิน หลังจากที่มีการเปลี่ยนสภาวะของ Flag นี้ เป็น "1" = Stop / เป็น "0" = RUN ในขณะที่จะเซ็ทเวลาให้ RTC นั้นควรให้บิทนี้เป็น "1" เพื่อไม่ให้ตัวทดเข้ามาที่หลักวินาที หลังจากเซ็ทเวลาให้ RTC เสร็จแล้วจึงให้บิทนี้เป็น "0" เพราะว่าในขณะที่เราเซ็ทเวลานั้นเกิดมีตัวทดเข้ามาจะทำให้หลักวินาทีเพิ่มค่าขึ้นอีก 1 วินาที หลังจากที่เราเซ็ทเวลาเสร็จแล้ว

**24/12 (D2)** บิทนี้จะเป็นการเลือกว่าจะให้เวลาเดินแบบ 24 ชม. หรือ 12 ชม. (มี AM/PM) ถ้าเลือกโหมด 1 - 24 ชม. บิท PM/AM จะไม่ถูกนำมาใช้ (มีค่าเป็น "0") แต่ถ้าเลือกโหมด 0 - 12 ชม. บิท PM/AM จะมีการเปลี่ยนสภาวะไปด้วยสำหรับการเซ็ทบิทมี ขั้นตอนดังนี

1) ต้องให้ Reset Bit = 1
2) 24/12 Hour Bit = 0 หรือ 1 ถ้าเป็น "0" หมายถึง โหมด 24 ชม. ถ้าเป็น "1" หมายถึงโหมด 12 ชม.
3) Reset Bit = 0

**หมายเหตุ** RE0T จะต้องเป็น "1" ถึงจะเขียนบิท 24/12 ได้

**TEST (D3)** เมื่อบิทนี้เป็น "1" อินพุทของวงจรนับในหลักวินาทีจะมาจากวงจรนับ/หารแทนที่จะมาจากภาคหาร 15

ดังนั้นวงจรนับหลักวินาทีจะนับความถี่ที่ 5.4163 KHz แทน (ปกติจะนับที่ความถี่ 1 Hz) เมื่อ Test = 1 (Test Mode) บิท Stop และบิท Reset จะต้องไม่ถูกเซ็ท ในขณะที่อยู่ใน Teat Mode (Test = 1) ถ้า Hold = 1 วงจรนับภายในจะถูกหยุดไว้ แต่เมื่อ HOLD กลับมาเป็น "0" จะไม่รับรองว่าเวลาที่ได้จะถูกต้อง

## 2.4 Liquid Crystal Display

LCD เป็นชื่อย่อของ Liquid Crystal Display หรือที่เรียกกันอย่างไทย ๆ ว่าตัวแสดงผลแบบผลึกเหลว

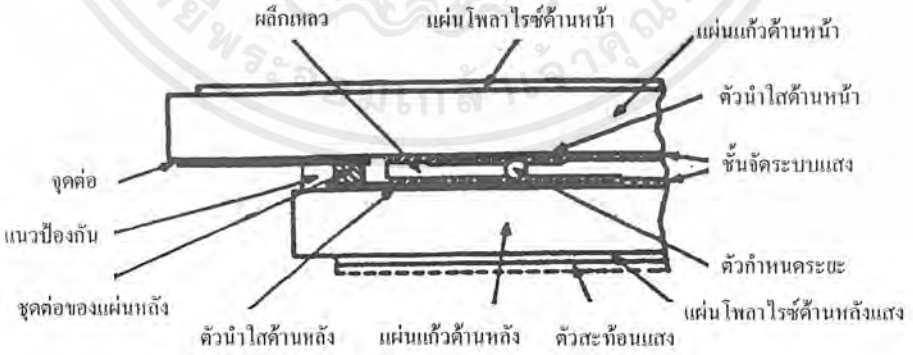
ผลึกเหลวเป็นสารที่รวมกันอย่างได้สัดส่วนระหว่างของเหลวกับผลึก จุดหลอมเหลวของสารชนิดนี้อยู่ในช่วงอุณหภูมิที่เรียกว่า เมโซเฟส (Mesophase) ซึ่งโมเลกุลของสารสามารถเคลื่อนที่ได้เหมือนของเหลว แต่สารชนิดนี้ถูกจัดอยู่ประเภทเดียวกับผลึกที่เป็นของแข็งทั่ว ๆ ไป

ในปี พ.ศ. 2513 ได้มีการค้นพบว่าแผนสารผลึกเหลวสามารถเปลี่ยนตัวเองจากใสกลายเป็นทึบแสง หรือจากทึบแสงกลายเป็นใสได้โดยการป้อนแรงดันเข้าไป คุณสมบัตินี้ก็คือหลักการพื้นฐานของ LCD ในปัจจุบันนั่นเอง

### 2.4.1 โครงสร้างของ LCD

ประกอบด้วยแผ่นแก้วสองแผ่นประกบกัน โดยเว้นช่องกลางไว้ประมาณ 6 ถึง 10 ไมโครเมตร ผิวด้านในของแผ่นแก้วเคลือบด้วยตัวนำไฟฟ้าชนิดใสที่ไว้แสดงตัวอักษร สัญลักษณ์หรือเครื่องหมายต่าง ๆ มักทำมาจากสารอินเดียมทินออกไซด์ (Indium/Tin Oxide : ITO) ระหว่างตัวนำไฟฟ้าชนิดใสกับผลึกเหลวจะมีชั้นสารที่ทำให้โมเลกุลของผลึกรวมตัวกันในทิศทางของแสงที่ส่องมาชั้นสารนี้จึงเป็นที่รู้จักกันในนามของชั้นที่หักเหเข้าหาแสงหรือชั้นจัดระบบรับแสง (Alignment Layer) ระยะห่างระหว่างแผ่นทั้งสองถูกกำหนดโดยตัวจัดระยะ (สังเกตดูรูปที่ 2.32 จะเห็นเป็นวงกลมอยู่ระหว่างชั้น)

ชนิดของผลึกเหลวที่ใช้ โดยทั่วไปคือแบบนีเมติก (Nematic) แสดงดังรูปที่ 2.30 ก โมเลกุลของผลึกเหลวแบบนิเมติกจะวางขนานกันไปเป็นแนวตรงคล้ายเส้นลวดยาว ถ้าหากวางกลับทิศจะทำให้คุณสมบัติของมันเปลี่ยนไป คริสตอลเหลวที่สามารถแสดงเครื่องหมายแสดงต่าง ๆ ได้คือ แบบโคเรสเทอริก (Choresteric) รูปที่ 2.30 ข. และแบบสเมกติก (Smectic) รูปที่ 2.30 ข

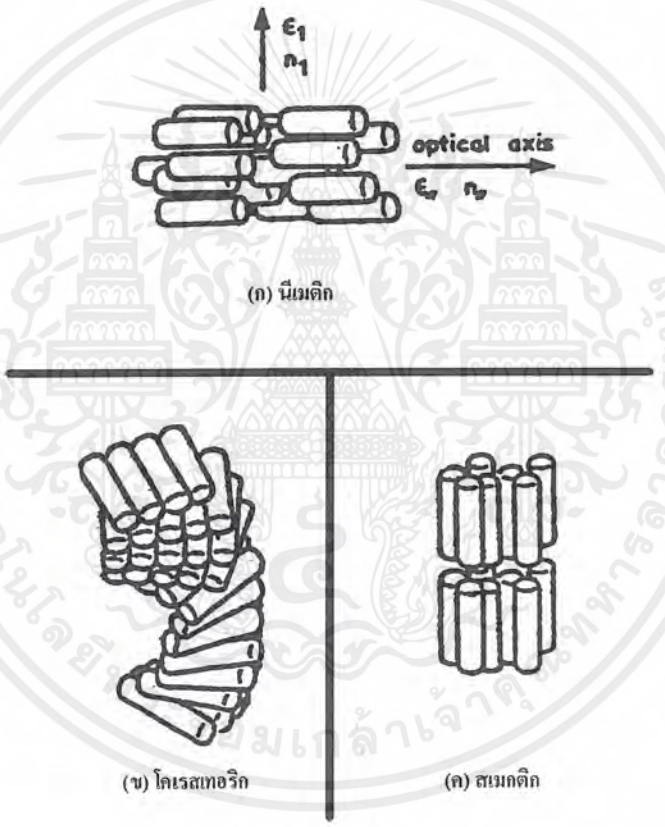



**รูปที่ 2.29** โครงสร้างของ LCD

### 2.4.2 LCD แบบ Nemetic ชนิดเกลียว

การทำงานเบื้องต้นของ LCD แบบ Nemetic ชนิดเกลียว (Twisted Nemetic : TN) แสดงในรูปที่ 2.31 โมเลกุลของผลึกเหลวนีเมติกจะถูกจัดไว้ในตำแหน่งโดยชั้นจัดระบบปรับแสง

ตำแหน่งของการรับแสงระหว่างแผ่นล่างกับแผ่นบนจะต่างกัน 90 องศา โมเลกุลของผลึกเหลวจะบิดเป็นเกลียวต่างกัน 90 องศา เมื่อแสงจากด้านล่างผ่านโมเลกุลของผลึกเหลวที่บิดเป็นเกลียว




**รูปที่ 2.30** แสดงโครงสร้างของโมเลกุลของผลึกเหลวทั้ง 3 แบบ

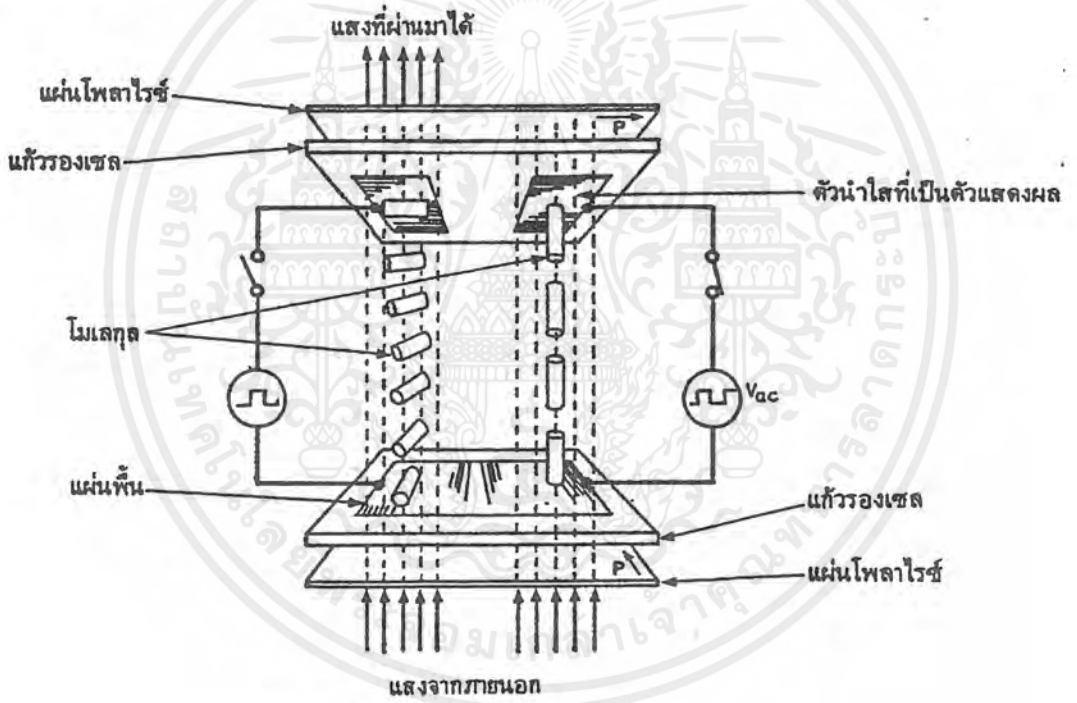
ขึ้นไปยังแผ่นบนทำให้ตำแหน่งของแสงหมุนไป 90 องศา ด้วยปรากฏการณ์นี้เกิดจากคุณสมบัติแอนนิโซโทรฟี ทางแสงของโมเลกุลทำให้แสงลอดผ่านแผ่นโพลาไลซ์ออกไปได้ แสดงดังรูปที่ 2.31 ก.

**แอนโซโทรฟี (Anisotropy)** หมายถึงค่าคงตัวของไดอิเล็กตริก (Permittivity : E) ของผลึกเหลวเปลี่ยนค่าไปตามตำแหน่งต่าง ๆ)

ถ้ามีแรงดันป้อนให้ที่ตัวนำไฟฟ้าชนิดใส คุณสบัติแอนนิโซโทฟีทางไฟฟ้าของโมเลกุลจะทำให้การวางตัวของโมเลกุลเปลี่ยนไป (ดูรูปที่ 2.31 ข.) แสงจะผ่านผลึกเหลวด้วยทิศทางคงเดิม จึงไม่สามารถผ่านแผ่นโพลาไลซ์ด้านบนออกไปได้

เมื่อเราป้อนแรงดัน โมเลกุลก็จะเรียงตัวดังเดิมอีกครั้งหนึ่ง (รูปที่ 2.31 ก.) แสงก็สามารถทะลุผ่านไปได้อีก การทำงานของ LCD แบบนี้เรียกว่า การแสดงผลเชิงบวก (positive image display) แสดงดังรูปที่ 2.32

หากเราวางแผ่นโพลาไลซ์ให้หมุนไปมากกว่า 90 องศา ผลก็จะเกิดตรงกันข้ามคือ LCD จะมืดทึบเมื่อไม่มีแรงดันป้อน แต่จะเป็นแผงใสเมื่อป้อนแรงดัน เช่นนี้เรียกว่า การแสดงผลเชิงลบ (negative image display)




(ก) ขณะไม่ทำงาน (ข) ขณะทำงาน

**รูปที่ 2.31** ทำงานพื้นฐานของตัวแสดงผลแบบนีเมติกชนิดเกลียว

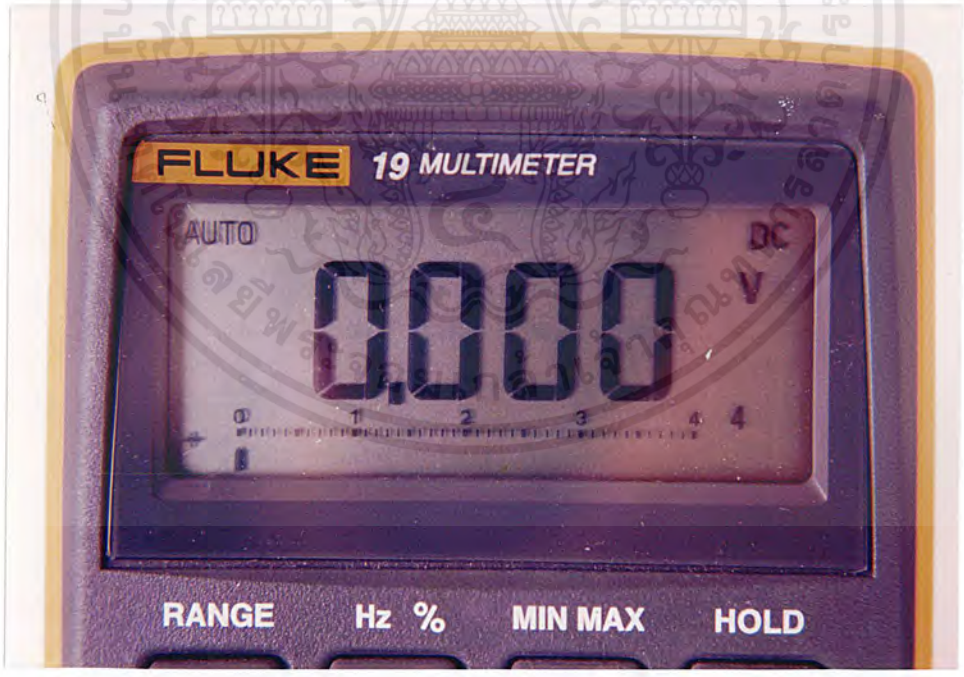
### 2.4.3 การแสดงผลแบบต่าง ๆ ของ LCD

LCD สามารถแสดงผลให้เราเห็นได้โดยมีหลักการ 3 แบบ จะเลือกใช้แบบใดก็ขึ้นอยู่กับแสงสว่างโดยรอบ

**1) แบบสะท้อน (reflective mode)** จะมีสารประเภทโลหะเคลือบอยู่ที่แผ่นหลังของ LCD เช่น เคลือบด้วยอลูมิเนียมฟอล์ยจะทำการสะท้อนแสงจากภายนอกผ่านตัวแสดงผลไปยังตาของเรา (ดูรูปที่ 2.34 ข.) แบบนี้จะเหมาะกับที่ที่มีแสงสว่างเพียงพอ ข้อดีคือ ไม่ต้องการแหล่งจ่ายแรงดันป้อนให้กับหลอดไฟใด ๆ อีก

**2) แบบส่งผ่าน (transmissive mode)** มักใช้กับ LCD ที่มีการแสดงผล (ซึ่งสภาวะการทำงานของตัวแสดงผลจะโปร่งใสในขณะที่พื้นเป็นสีทึบ) แบบนี้จะวางหลอดไฟไว้ด้านหลังทำให้อ่านค่าแสดงผลได้ชัดเจน แสดงดังรูปที่ 2.34 ก.

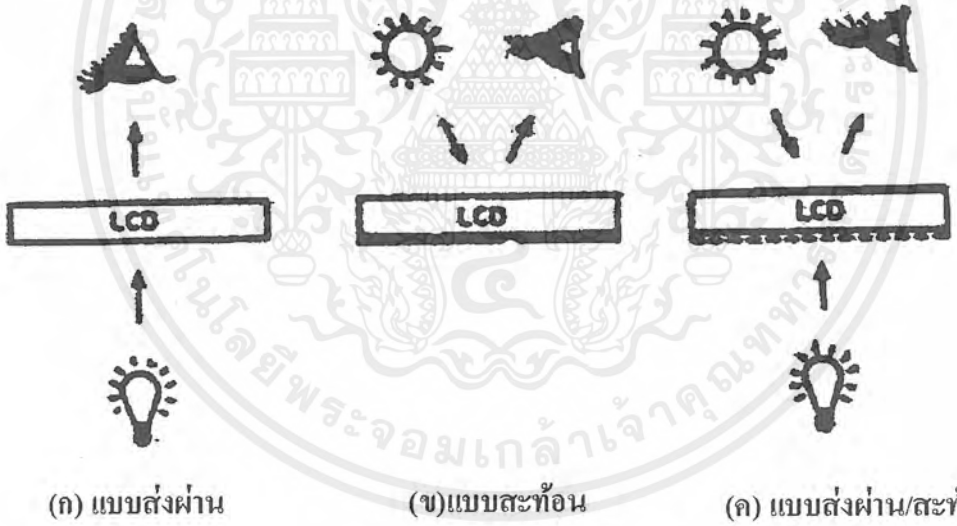
**3) แบบส่งผ่าน/สะท้อน (transflective mode)** เป็นการรวมระหว่าง 2 แบบที่กล่าวมาแล้ว ตัวแสดงผลอ่านได้จากการสะท้อนของแสงจากภายนอก และมีแสงส่องสว่างจากด้านหลังเมื่อต้องการอ่านค่าต่าง ๆ ในที่มืด แสดงดังรูปที่ 2.34 ค.




**รูปที่ 2.32** การแสดงผลเชิงบวก

(ก) ส่วนแสดงผล (ข) ส่วนจุดร่วม

รูปที่ 2.33 ภาคแสดงผล 7 ส่วน




(ก) แบบส่งผ่าน (ข)แบบสะท้อน (ค) แบบส่งผ่าน/สะท้อน

รูปที่ 2.34 แบบต่าง ๆ ของการแสดงผลของ LCD

### 2.4.4 สีสันของ LCD แบบ TN

สีของ LCD แบบนีเมติกชนิดเกลียว (TN) ได้มาจาก 3 สิ่ง คือ ตัวกระจายสี, ตัวกรองสี และแสงสีจากด้านหลัง ตัวกระจายสีจะให้ส่วนแสดงผลเป็นสีต่าง ๆ บนแผ่นพื้นไร้สีหรือตัวอักษรไร้สีบนแผ่นพื้นสีก็ได้ โดยการใช้แผ่นโพลาไลซ์ 2 สี เช่น สีแดงและเขียว อาจให้ตัวแสดงผลสีแดงอยู่บนพื้นสีเขียวหรือตัวแสดงผลสีเขียวอยู่บนพื้นสีแดงก็ได้ ตัวกรองสีจะแผ่นฟอล์

ยอยู่ด้านหลังหรือเป็นสีขุ่น ๆ บนตัวแสดงผลก็ได้ มักใช้ใน LCD แบบส่งผ่าน เช่น ตัวอักษรสีแดงบนพื้นทึบแสง แสงสีจากด้านหลังทำให้ตัวแสดงผลเป็นสีดำบนแผ่นพื้นสีต่าง ๆ หรือตัวแสดงผลสีต่าง ๆ บนแผ่นพื้นสีดำ หากจะใช้ในแบบส่งผ่าน/สะท้อนควรคำนึงถึงด้วยว่าแสงจากด้านหลังจะถูกลดทอนลงอย่างมากโดยแสงจากภายนอก

### 2.4.5 คุณสมบัติทางแสง

ความชัดเจนในการอ่านขึ้นอยู่กับตัวแปรสำคัญคือ ความสว่าง (brightness) และความเข้มของแสง (contrast) ของ LCD ด้วย ความเข้มของแสงหาได้จากความสว่างของบริเวรไร้สีหารด้วยความสว่างของบริเวณที่มีสีทึบ ใน LCD แบบ TN อัตราส่วนความเข้มของแสงจะอยู่ในย่าน 5 ถึง 50 แต่อัตราส่วนความเข้มสูงสุดที่ตามนุษย์สามารถจับได้มีค่าประมาณ 10 และความชัดเจนอย่างน้อยที่สุดประมาณ 2

ในตัวแสดงผลแบบที่มีแสงส่องจากด้านหลังต้องมีอัตราส่วนความเข้มของแสงสูงเป็นพิเศษเพราะเมื่อมองผ่านแสงที่สวนขึ้นมาตาของเราจะจับความเข้มได้น้อยลง ทั้งความสว่างและความเข้มขึ้นอยู่กับชนิดของแผ่นโพลาไลซ์ สำหรับตัวแสดงผลแบบสะท้อนเชิงบวกแผ่นโพลาไลซ์ประสิทธิภาพต่ำจะทำให้ความสว่างมากแต่ความเข้มน้อย ส่วนแผ่นโพลาไลซ์ประสิทธิภาพสูงจะให้ความเข้มสูงแต่ความสว่างลดลง

### 2.4.6 มุมมอง

รูปที่ 2.35 แสดงความสัมพันธ์ระหว่างความเข้มของแสงกับแรงดันของมุมมอง 3 มุมที่แรงดันต่ำมาก ๆ LCD ไม่สามารถแสดงผลให้เห็นชัดเจนได้ เมื่อ $\varnothing$ คือมุมในแนวระดับของ LCD และ $\alpha$ คือมุมที่อ้างอิงถึงกับเส้นตั้งฉาก

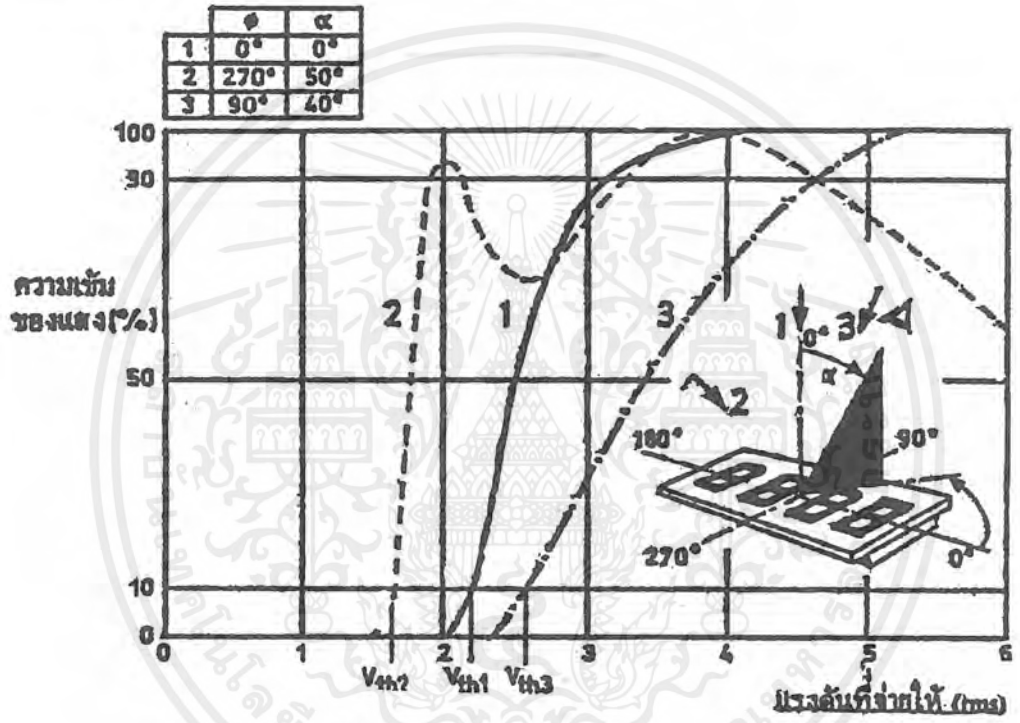
เมื่อแรงดันเพิ่มขึ้นมุมที่มองเห็นก็จะสูงตาม (มุมเงยต่ำ) ดังเส้นกราฟเส้นที่สอง หากแรงดันสูงขึ้นมุมเงยก็เพิ่มขึ้นตามลำดับ

แรงดันที่น้อยที่สุดที่พอจะทำให้มองเห็นได้ชัดเจน (10 เปอร์เซนต์ของแรงดันสูงสุด) เรียกว่า แรงดันแทรชโฮล (Threshold vottage : $V_{th}$) แรงดันที่มีค่าถึง 90 เปอร์เซนต์ของแรงดันสูงสุด เรียกว่า แรงดันอิ่มตัว (Saturtion voltage : $V_{sat}$)

ความสัมพันธ์ระหว่างแรงดันกับความเข้มของแสงจะเปลี่ยนไปตามส่วนผสมที่แตกต่างกันของผลึกเหลว อัตราส่วนผสมส่วนใหญ่มักมีสัมประสิทธิ์ของอุณหภูมิเป็นลบ ซึ่ง $V_{th}$ จะลดลงเมื่ออุณหภูมิสูงขึ้น

### 2.4.7 ความเร็วของการแสดงผล

เวลาตอบสนองการทำงานอยู่ในช่วง 50 ถึง 100 มิลลิวินาที (ในอุณหภูมิห้อง 25 องศา) สิ่งสำคัญที่มีผลต่อช่วงเวลาตอบสนองคือความเหนียวของผลึกเหลวซึ่งความเหนียว (Viscosity) ของมันจะเพิ่มขึ้นเมื่ออุณหภูมิลดลง เพราะโมเลกุลมีอิสระในการเคลื่อนที่น้อยลงทำให้การตอบสนองช้าลง เวลาตอบสนองยังขึ้นอยู่กับขนาดของแรงดันที่จ่ายให้วิธีป้อนแรงดันและความหนาของแผ่นผลึกเหลวด้วย



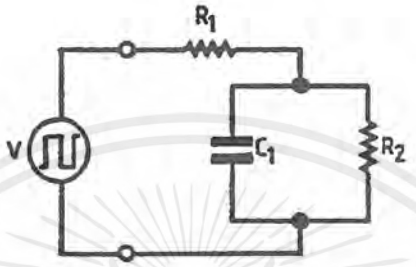

รูปที่ 2.35 ความสัมพันธ์ระหว่างความเข้มของแสงกับแรงดัน

### 2.4.8 การขับ LCD

LCD แต่ละส่วน (เซกเมนต์) สามารถเขียนวงจรสมมูลย์ได้ดังรูปที่ 2.39 ประกอบด้วยตัวต้านทาน $R_1$ ค่าต่ำต่ออนุกรมกับตัวเก็บประจุ $C_1$ ซึ่งต่อขนานกับตัวต้านทาน $R_2$ (มีค่าสูงกว่า $R_1$) ค่าของตัวเก็บประจุจะขึ้นอยู่กับแรงดันที่ปรากฏคร่อม LCD ความถี่ของสัญญาณที่ป้อนเข้าไปต้องไม่ต่ำกว่า 30 เฮิร์ตซ์ เพื่อป้องกันการกระพริบของตัวแสดงผล กระแสที่ใช้กันทั่วไปคือ 1.5 ไมโครแอมป์ต่อตารางเซนติเมตรของแผ่น LCD คือจำนวนจุดร่วมของแผ่นหลังหรือจำนวนของส่วนแสดงผลในแต่ละกลุ่ม

## 2.5 มาตรฐาน RS 232C

โดยปกติไมโครคอมพิวเตอร์จะมีพอร์ตที่เป็นอนุกรม เรียกชื่อกันว่า RS 232C อยู่ในตัวเองอยู่แล้ว หลายเครื่องไม่มีมากับเครื่องอย่างเช่น IBM PC จำเป็นจะต้องมีการ์ดที่เรียกว่าอะซิงโครนัสอะแดปเตอร์ (Asynchronous Communication Adapter) มาเสียบใส่พอร์ต RS 232C




**รูปที่ 2.36** วงจรสมมูลย์ของ LCD

ทำหน้าที่รับและส่งข้อมูลในแบบอนุกรมเรียกว่า Universal Asynchronous Adapter เหตุที่มีชื่อเรียกว่า RS 232C ก็เนื่องจากสมาคมผู้ผลิตอุปกรณ์อิเล็กทรอนิกส์ของอเมริกาหรือ EAI ได้กำหนดมาตรฐานของอุปกรณ์ การสื่อสารแบบอนุกรมเอาไว้ภายใต้ชื่อว่า RS 232C ความจริงมาตรฐานของการส่งข้อมูลแบบอนุกรมมีหลายมาตรฐานแต่ที่นิยมกันมากที่สุดสำหรับ ไมโครคอมพิวเตอร์ก็คือ RS 232C

**หน้าที่สำคัญของการสื่อสารแบบอะซิงโครนัสคือ**

ตัวรับสัญญาณ

1. เปลี่ยนสัญญาณเข้ามาแบบอนุกรมให้เป็นแบบขนาน
2. ตรวจสอบความผิดพลาดของสัญญาณที่รับ
3. ตัดสต๊อปบิตและพาริตี้บิตออก
4. ส่งสัญญาณให้ซีพียูรู้ว่ารับสัญญาณไว้แล้ว

ตัวส่งสัญญาณ

1. เปลี่ยนสัญญาณแบบขนานจากซีพียูค่อยทยอยส่งออกเป็นแบบอนุกรม
2. เพิ่มสต๊อปบิตและพาริตี้
3. เพิ่มสัญญาณควบคุมโมเด็มที่ต่อเชื่อม (ถ้ามี)

มาตรฐาน RS 232C ได้จัดพิมพ์ขึ้นเมื่อ ปี ค.ศ. 1969 โดยสมาคมผู้ผลิตอุปกรณ์อิเล็กทรอนิกส์แห่งสหรัฐอเมริกา RS ย่อมาจาก Recommened Standard ส่วน 232 เป็นหลายเลขบ่งบอกของมาตรฐานตัวนี้ C เป็นหมายเลขของฉบับท้ายสุดของมาตรฐานตัวนี้ จุดประสงค์

ของมาตรฐานตัวนี้ก็เพื่อบรรยายคุณลักษณะของการการเชื่อมต่ออุปกรณ์รับส่งข้อมูลปลายทาง (Data Terminal Equipment DTE) กับอุปกรณ์สื่อสารข้อมูล

(Data communication Equipment DCE) สำหรับผู้ใช้ไมโครคอมพิวเตอร์ DTE ก็หมายถึงตัวไมโครคอมพิวเตอร์และ DCE ก็หมายถึง โมเด็ม อุปกรณ์อื่น ๆ เช่น เครื่องพิมพ์ที่รับสัญญาณแบบอนุกรมอาจจะเป็นได้ทั้ง DTE และ DCE


**รูปที่ 2.37** การใช้ RS 232C เชื่อมต่ออุปกรณ์

ขึ้นอยู่กับผู้ผลิต ข้อแตกต่างของ DTE และ DCE จะเห็นได้จาก รูปที่ 2.15 จากรูปนี้เราจะเห็นได้ว่า RS 232C มีส่วนสำคัญอย่างใหญ่หลวงสำหรับการสื่อสารข้อมูลระหว่างไมโครคอมพิวเตอร์ ความจริงอีกประการหนึ่งของ RS 232C ก็คือ ความเร็วและระยะทางของการเชื่อมต่อ RS 232C สามารถเชื่อมต่อการถ่ายโอนข้อมูลได้จาก 0-20,000 บิตต่อวินาที ซึ่งเพียงพอสำหรับไมโครคอมพิวเตอร์ที่มีขนาดอัตราบอด 110 ถึง 9600 บอด ความยาวของสายเชื่อมต่อโดยสัญญาณตามมาตรฐานของ RS 232 จำกัดอยู่แค่ 50 ฟุตซึ่งเพียงพอสำหรับการสื่อสารไมโครคอมพิวเตอร์กับอุปกรณ์รอบนอก

### 2.5.1 ลักษณะของสัญญาณ RS 232C

เพื่อเป็นหลักประกันว่าข้อมูลถูกส่งออกไปอย่างถูกต้องและอุปกรณ์ถูก ควบคุมอย่างถูกต้อง จำเป็นจะต้องมีข้อตกลงกันในเรื่องของสัญญาณที่ใช้ มาตรฐาน RS 232C กำหนดย่านของแรงดันไฟฟ้าในสัญญาณเพื่อสนองจุดประสงค์ข้างบน ดังแสดงในตารางที่ 2.14 และรูป 2.47

สำหรับเครื่องไมโครคอมพิวเตอร์บางเครื่อง ใช้แต่สัญญาณลอจิกออกมาเป็นสัญญาณของ RS 232C เลย อย่างเช่น อะซิงโครนัสอะแดปเตอร์ของ IBM PC ในกรณีเช่นนี้ระยะทางของสายที่เชื่อมต่ออาจจะไปได้สั้นกว่า 50 ฟุต ดังที่กล่าวเอาไว้เนื่องจากระดับของกราวนด์เปลี่ยนแปลงไป อันเนื่องจากการสูญเสียไปในความต้านทานของสาย

**ตารางที่ 2.17** แสดงการกำหนดมาตรฐานแรงดันไฟฟ้า

| แรงดันไฟฟ้า | สถานภาพลอจิก | สถานภาพของสัญญาณ | ฟังก์ชั่นในการควบคุม |
|---|---|---|---|
| บวก | 0 | สเปซ | ออน |
| ลบ | 1 | มาร์ค | ออฟ |

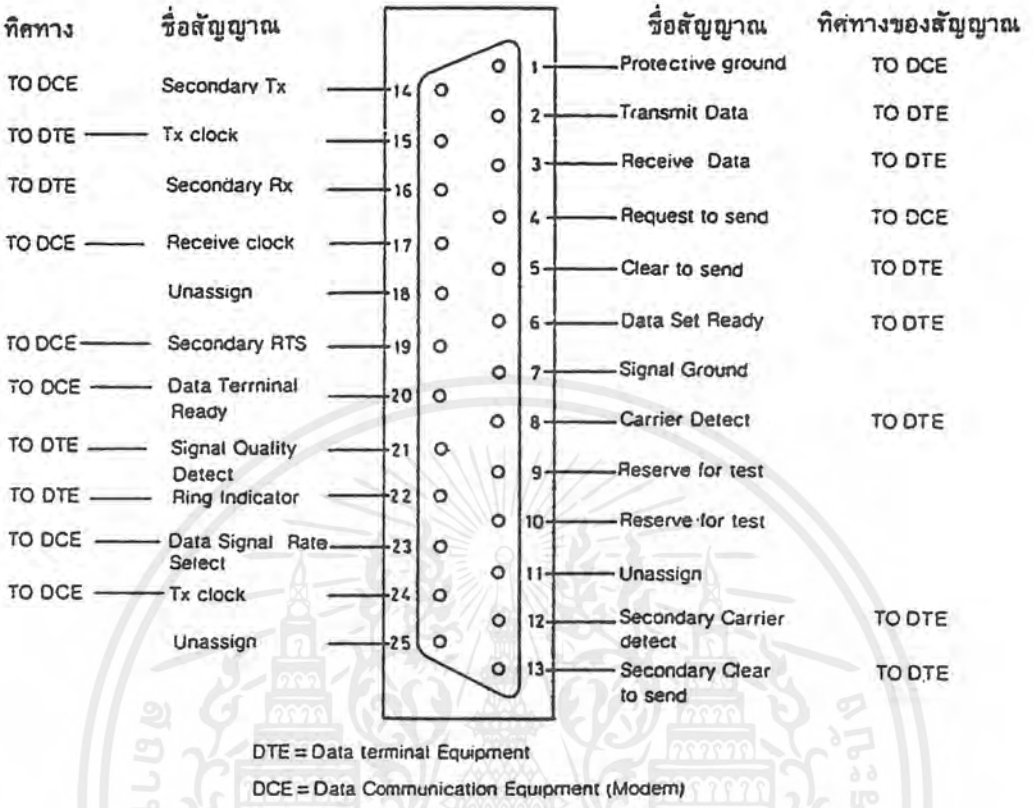
ผู้ที่เคยใช้ IBM PC อาจจะเคยประสบปัญหานี้มาแล้วว่าทำไมต่อสัญญาณ RS 232C เกินกว่า 10 ฟุต แล้วใช้งานไม่ได้ แต่อย่างไรก็ตาม RS 232C ของ IBM PC ยังมีโอกาสให้เลือกใช้ 20 มิลลิแอมแปร์ กระแสกลับแรงดันไฟฟ้า


**รูปที่ 2.38** ย่านของแรงดันไฟฟ้าที่ใช้ในสัญญาณ RS 232C

### 2.5.2 การกำหนดจุดต่อของ RS 232C

ในทางฟิสิกส์แล้ว มาตรฐานของ RS 232C กำหนดข้อต่อแบบ DB-25 แต่ละขาของขั้วต่อกำหนดไว้ดังรูปที่ 2.38

อาจจะใช้ข้อต่อชนิดอื่นที่นอกเหนือไปจาก DB-25 ยกตัวอย่างเช่น Fujitsu F-8 IBM AT, IBM Jr เป็นต้นตัวเมียของขั้วต่อควรจะอยู่ที่ตัวโมเด็ม ขณะที่ตัวผู้ควรจะอยู่ที่ Asynchronous Communication Adapter หรือที่ตัวไมโครคอมพิวเตอร์เอง อย่างไรก็ตาม ผู้ผลิตหลายรายไม่ได้ทำตามกฎเกณฑ์ที่ว่านี้




รูปที่ 2.39 การกำหนดของขั้วต่อ RS-232 แบบ DB-25

**การทำงานของขั้วต่อ RS-232 ดังนี้**

**1) Transmit Data (TD ขาที่ 2)**

เป็นสัญญาณที่ส่งออกจาก DTE (หรือตัวไมโครคอมพิวเตอร์) ไปยังโมเด็มหรือต่อเข้าโดยตรงกับไมโครคอมพิวเตอร์ตัวอื่น หรือเครื่องพิมพ์ เมื่อไม่มีสัญญาณส่งออกสถานภาพของลอจิกที่ขานี้จะมีค่าเท่ากับ "1" หรือเทียบเท่ากับสต๊อปบิต

**2) Receive Data (RD ขาที่ 3)**

เป็นทางของสัญญาณเข้าไปยัง DTE หรือไมโครคอมพิวเตอร์เมื่อไม่มีสัญญาณรับเข้ามา ขานี้จะมีสถานภาพทางลอจิกเป็น "1"

**3) Request To Send (RTS ขาที่ 4)**

ใช้สำหรับรับส่งสัญญาณไปยังโมเด็มหรือเครื่องพิมพ์เป็นการเรียกร้องที่จะส่งสัญญาณมาทางขา 2

สัญญาณนี้ใช้คู่กับ CTS หรือ Clear to send อุปกรณ์รับหากได้รับสัญญาณ RTS จะตรวจสอบตัวเองว่า พร้อมจะรับสัญญาณได้หรือยัง หากพร้อมที่จะรับก็ส่งสัญญาณออกไปที่สาย CTS

4) **Clear To Send (CTS ขาที่ 5)**

ดังอธิบายไว้ใน RTS เมื่อสัญญาณนี้อยู่ในสถานะออฟ (negative voltage หรือลอจิก "1") หมายความว่า อุปกรณ์รับกำลังบอกว่าพร้อมที่จะรับข้อมูลแล้ว

5) **Data Set Ready (DSR ขาที่ 6)**

เมื่อสัญญาณสายนี้อยู่ในสถานะออน (หรือลอจิก 0) เป็นการบอกไมโครคอมพิวเตอร์หรือฝ่ายส่งว่า โมเด็มต่อเข้ากับสายโทรศัพท์เรียบร้อยแล้วและพร้อมที่จะส่งได้แล้ว

6) **Signal Ground (SG ขาที่ 7)**

SG ทำหน้าที่เป็นระดับแรงดันอ้างอิงสำหรับทุก ๆ สายของสัญญาณ จะมีแรงดันเป็น "0" เมื่อเทียบกับสัญญาณตัวอื่น

7) **Carrier Detect (CD ขาที่ 8)**

โมเด็มจะส่งสัญญาณที่อยู่ในสถานะออน (ลอจิก "0") ไปบอกไมโครคอมพิวเตอร์ เมื่อได้รับสัญญาณจากโมเด็มของอีกฝ่ายหนึ่ง สัญญาณนี้จะนำไปจุด LED บอกว่าได้รับสัญญาณจากโมเด็มอีกฝ่ายหนึ่งแล้วไฟ LED จะอยู่บนหน้าปัดของโมเด็มเอง

8) **Data Terminal Ready (DTR ขาที่ 20)**

คอมพิวเตอร์เปิดสัญญาณสายนี้ให้ออน (ลอจิก "0") เมื่อพร้อมที่จะติดต่อกับโมเด็ม โมเด็มส่วนมากจะไม่รายงานสถานภาพของตัวเอง (CD, DSR และ CTS) ให้คอมพิวเตอร์รู้ หากคอมพิวเตอร์ไม่เปิดสัญญาณ DTR

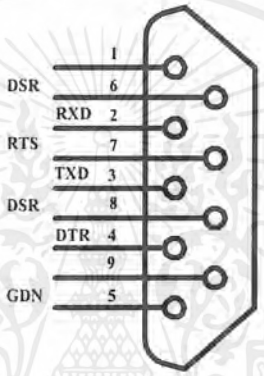
9) **Ring Indicator (RI ขาที่ 22)**

สัญญาณนี้ใช้ในโมเด็มที่เป็นระบบตอบได้อัตโนมัติ(Auto-answer) สัญญาณนี้จะออนเมื่อมีสัญญาณกระดิ่งมาและออฟระหว่างเสียงของกระดิ่ง

บางทีเราอาจจะสับสนระหว่างสถานะภาพของลอจิกกับสถานะภาพของสัญญาณ โดยปกติเราจะคุ้นเคยอยู่กับความรู้สึกที่ว่า เมื่อแรงดันเป็นบวก หรือสัญญาณออน ลอจิกเป็น "0" หรือขณะที่ไม่มีอะไรส่งควรจะมีสัญญาณทางไฟฟ้าครบวงจรอยู่ตลอดเวลา จะได้รู้ว่าวงจรไม่ขาดระหว่างทางตรงไหน ควรจะรู้ว่าวงจรครบอยู่ตลอดเวลาก็โดยการให้ค่าแรงดันที่ฝ่ายส่ง ดังนั้นจึงถือกันว่าสัญญาณไฟบวกใช้เป็นลอจิก "0"

### 2.5.3 มาตรฐาน RS 232C กับ V.24

ได้กล่าวถึงสัญญาณตามสมาคมผู้ผลิตอุตสาหกรรมอิเล็กทรอนิกส์ ของสหรัฐอเมริกา หรือ RS 232C ไปแล้ว สหประชาชาติและกลุ่มของ CCITT (Comite Consultatif International Telephonique Telegraphique) ได้ออกมาตรฐานมาเหมือนกัน และก็หลายฉบับตั้งแต่การประชุมครั้งที่สอง ที่กรุงนิวเดลฮี ปี ค.ศ. 1960 ออกมาเป็นสมุดปกแดง ครั้งที่สาม ปี ค.ศ. 1964 ที่กรุงเจนีวา ออกมาเป็นสมุดปกสีน้ำเงิน จนกระทั่งครั้งที่ 6 เมื่อ ปี ค.ศ. 1977 ที่กรุงเจนีวา อีกเหมือนกัน ออกมาเป็นสมุดปกสีส้ม ได้เป็นมาตรฐานออกมา 3 รูปแบบ




**รูปที่ 2.40** แสดงขั้วต่อ RS-232 แบบ DB-9

V.24 บรรยายถึงการเชื่อมต่ออุปกรณ์รับส่งข้อมูล (DTE) กับอุปกรณ์รับส่งข้อมูลปลายทาง

V.28 บรรยายถึงลักษณะทางไฟฟ้าสำหรับการใช้ Unbalance Double Current Interchange Circuit

คุณลักษณะโดยย่อของสัญญาณ RS 232C

| | |
|---|---|
| Driver output logic levels with | 15V>$0_h$>5v |
| 3k to 7k load | -5>$0_l$>-15โวลต์ |
| Driver output voltage when open Circuit | Vo <25 โวลต์ |
| Driver output impedance with Power off | $R_o$>300 0hms |
| Output short circuit current | $I_o$ < 0.5 A |
| Driver slew rate dv/dt <30 V/s | |
| Receiver input impedance | 7k > $R_{in}$ > 3k |
| Receiver input voltage | +15 compatible with |
| Driver | |
| Receiver output with open | Mark |
| Circuit input | |
| Receiver output with +3V input | Space |
| Receiver output with -3V input | Mark |
| +15 | Logic 0 = Space = |
| +5 | Control on |
| +5 | Noise Margin |
| +3 | |
| +3 | Transition Region |
| -3 | |
| -3 | Noise Margin |
| -5 | |
| -5 | Logic 1 = Mark = |
| -15 | Control off |

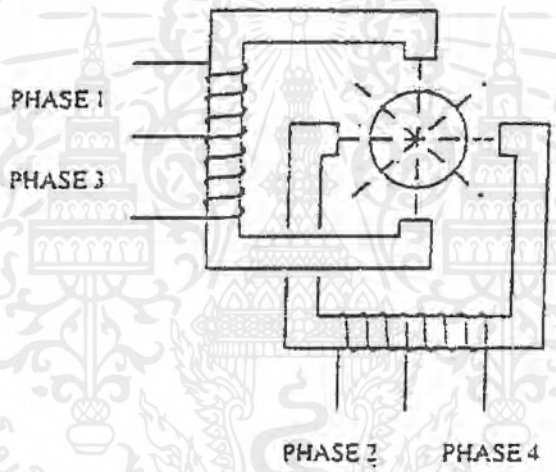
## 2.6 สเต็ปเปอร์มอเตอร์

สเต็ปเปอร์มอเตอร์ ถือว่าเป็นอุปกรณ์เอาต์พุตอย่างหนึ่ง ซึ่งสามารถควบคุมได้ด้วยไมโครคอนโทรลเลอร์ ลักษณะการทำงานของสเต็ปเปอร์มอเตอร์ จะเคลื่อนที่เป็นสเต็ป (Step) ซึ่งอาจเป็นสเต็ปละ 1.8, 5, 7.5 องศา ก็แล้วแต่ชนิดของมอเตอร์ ส่วนใหญ่สเต็ปเปอร์มอเตอร์จะใช้ในงานควบคุมระบบดิจิเตอร์ เช่น Printer, X - Y Plotter, Disk drive ตลอดจนอุปกรณ์ในงานอิเล็กทรอนิกส์อุตสาหกรรม หรือเครื่องมือวัดและระบบควบคุมอื่น ๆ

สเต็ปเปอร์มอเตอร์จะประกอบด้วยส่วนที่สำคัญ 2 ส่วน คือ

**1. โรเตอร์ (Rotor)** เป็นส่วนที่หมุนได้ จะเป็นแม่เหล็กถาวรและอื่น ๆ

**2. สเตเตอร์ (Stator)** เป็นส่วนที่อยู่กับที่ จะเป็นขดลวดหลาย ๆ ขด




**รูปที่ 2.41** สเต็ปเปอร์มอเตอร์ 4 เฟส แบบยูนิโพลาร์เพอร์มาเนนแม็กเนต

### 2.6.1 ชนิดของสเต็ปเปอร์มอเตอร์

เราสามารถแบ่งสเต็ปเปอร์มอร์เตอร์ตามพื้นฐานได้ 4 ชนิด คือ

**1) ชนิดวาริเอเบิลรีลักแตนซ์ (Variable reluctance หรือ VR)** สเต็ปเปอร์ชนิดนี้มีข้อเสียคือ เมื่อมีสเต็ปในการหมุนสูง จะทำให้ความถูกต้องของตำแหน่งและทำงานได้ไม่ดี เราสามารถทดสอบเพื่อให้ทราบว่าเป็นสเต็ปเปอร์มอเตอร์ชนิดนี้ได้ง่ายมากโดยใช้มือหมุนที่เพลาของมอเตอร์ ซึ่งจะไม่เกิดปรากฏการณ์ทางแม่เหล็ก (Magnetism) จะทำให้หมุนได้โดยไม่ติดขัด แตกต่างจากชนิดอื่น คือ เมื่อทำการหมุนจะรู้สึกขัด ๆ เหมือนเป็นฟันเฟือง

**2) ชนิดเพอร์มาเนนต์แม็กเนต (Permanent magnet หรือ PM)** สเต็ปเปอร์มอเตอร์ชนิดนี้ มีข้อดีคือ มีความถูกต้องข้อตำแหน่งเมื่อเปรียบเทียบกับชนิดอื่น

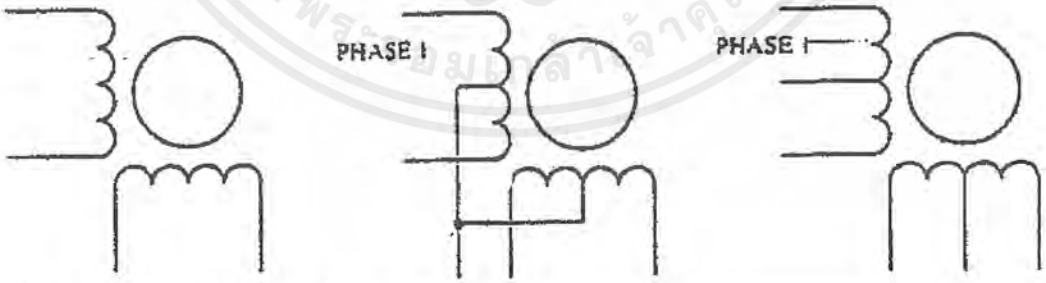
**3) ชนิดไฮบริด (Hybrid)** เป็นชนิดที่นิยมใช้มากที่สุดในเครื่องคอมพิวเตอร์ สเต็ปเปอร์มอเตอร์ชนิดนี้ มีโครงสร้างภายในคือสเตเตอร์เป็นชนิดวาริเอเบิลรีลักแตนซ์ ส่วนโรเตอร์เป็นชนิดเพอร์มาเนนต์แม็กเนตนำมาประกอบเข้าด้วยกัน ทำให้เป็นมอเตอร์ชนิดที่มีแรงยึดหน่วงสูง มีแรงบิดดึงและผลักดี และยังคงทำงานได้ดีแม้ว่าจะมีจำนวนสเต็ปต่อรอบในการหมุนสูง

**4) ชนิดแรเอิร์ธเพอร์มาเนนต์แม็กเนต (Rare earth permanent magnet)** หรือที่เรียกกันว่า ชนิดดิสก์แม็กเนตสเต็ปเปอร์มอเตอร์ (Disk magnet steppers) การทำงานจะเป็นแบบเดิม แต่โครงสร้างเป็นแบบใหม่จะทำให้เกิดความเฉื่อยต่ำมาก, มีอัตราเร่งสูง มอเตอร์ชนิดนี้จึงจัดเป็นมอเตอร์ที่มีประสิทธิภาพสูงทั้งในด้าน แรงบิดดึง, กำลังทางกลที่ได้ของมอเตอร์, ความถูกต้องของตำแหน่งสูงมากและความเร็วในการเริ่มหมุนและหยุดสูง อีกทั้งมีการสูญเสียของกำลังงานต่ำ

## 2.6.2 การพันขดลวดบนสเต็ปเปอร์มอเตอร์มีอยู่ 2 วิธี คือ

**1) แบบไบโพลาร์ (Bipolar)** สเต็ปเปอร์แบบไบโพลาร์จะมีการพันขดลวด 1 ขดบนแต่ละขั้วแม่เหล็กของสเตเตอร์ ขั้วแม่เหล็กที่เกิดขึ้นบนสเตเตอร์ จะถูกกำหนด โดยทิศทางของกระแสไฟฟ้า และทำให้เกิดขั้วแม่เหล็กในทิศทางตรงกันข้ามได้โดยการกลับทิศทางการไหลของกระแสไฟฟ้า ซึ่งการกำหนดทิศทางการไหลและการกลับทิศทางของกระแสไฟฟ้าทำได้โดยการใช้วงจรสวิตชิ่งกลับขั้วไฟฟ้า ดังแสดงในรูปที่ 2.42 (ก)

**2) แบบยูนิโพลาร์ (Unipolar)** จะมีการพันขดลวด 2 ขดบนแต่ละขั้วแม่เหล็กของสเตเตอร์ ซึ่งแต่ละขดจะทำให้เกิดขั้วแม่เหล็กเปลี่ยนไปมาได้โดยการใช้สวิตชิ่งกระแสไฟฟ้าจากขดลวดขั้วหนึ่งไปยังอีกขั้วหนึ่งเท่านั้น โดยปกติขดลวดทั้งสองจะมีจุดร่วมเพื่อลดจำนวนของสายไฟที่ต่อจากมอเตอร์ ดังแสดงในรูปที่ 2.42 (ข)




**(ก) ไบโพลาร์ แบบ 4 สาย 2 เฟส** **(ข) ยูนิโพลาร์ แบบ 5 สาย 4 เฟส และแบบ 6 สาย 4 เฟส**

**รูปที่ 2.42** การพันขดลวดบนสเต็ปเปอร์มอเตอร์แบบไบโพลาร์และแบบยูนิโพลาร์

การพันลวดแบบยูนิโพลาร์จะมีข้อเสียที่การพันแบบนี้จะทำให้เกิดแรงบิดน้อย กว่า แบบไบโพลาร์ เพราะในระยะเวลาหนึ่งจะมีเพียงครึ่งหนึ่งของขดลวดเท่านั้นที่ถูกกระตุ้น ให้ทำงาน

ส่วนการพิจารณาว่า สเต็ปเปอร์มอเตอร์ตัวใดมีการพันขดลวดแบบใด จะสังเกตได้โดยถ้าเป็นแบบไบโพลาร์จะมีสายไฟต่อออกมาจากมอเตอร์เพียง 4 สาย และถ้าเป็นแบบยูนิโพลาร์จะมี 5 หรือ 6 สาย หรืออาจอ่านได้จากป้าย (name plate) ที่ติดอยู่กับมอเตอร์

### 2.6.3 การกระตุ้นและการควบคุมการหมุนของสเต็ปเปอร์มอเตอร์

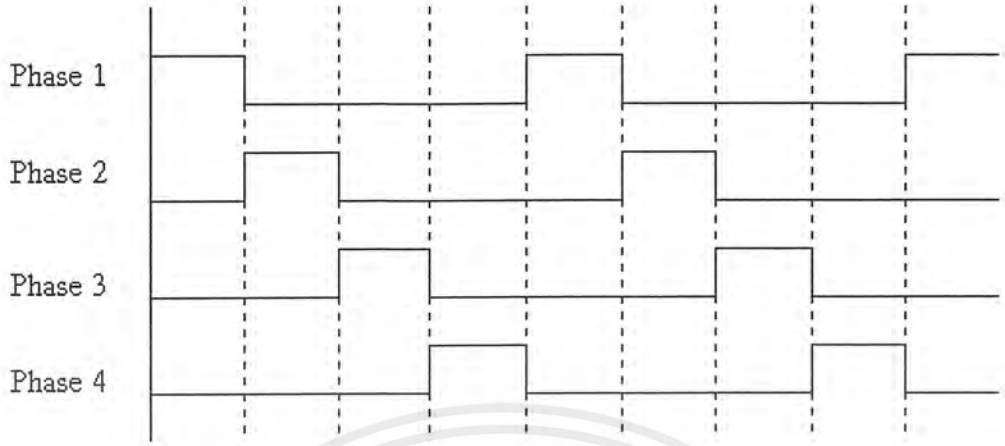
การทำให้สเต็ปเปอร์มอเตอร์เคลื่อนไปทีละสเต็ป ทำได้โดยการจ่ายกำลังไฟฟ้าไปยังขดลวดแต่ละขดบนสเตเตอร์ ซึ่งจะต้องป้อนเป็นแบบซีเควนเชียลในรูปแบบที่ถูกต้อง การป้อนพัลซ์กระตุ้นสเต็ปเปอร์มอเตอร์สามารถทำได้ 3 รูปแบบคือ

**1) แบบเวฟ (Wave)** เป็นการป้อนกระแสให้กับขดลวดแต่ละขดของสเต็ปเปอร์มอเตอร์ทีละขดเรียงลำดับกันได้ ลักษณะการขับแบบนี้จะทำให้แรงบิดน้อย ดังแสดงในรูปที่ 2.43

**2) แบบ 2 เฟส (Two phase)** มีลักษณะคล้ายกับแบบเวฟ แต่การกระตุ้นแบบนี้จะทำการกระตุ้นโดย จ่ายกำลังไฟฟ้าไปที่ขดลวด 2 ขด ที่อยู่ใกล้กันในเวลาเดียวกัน เรียงถัดกันไปเช่นเดียวกับแบบเวฟขึ้นอยู่กับทิศทางของการหมุน การเพิ่มจำนวนขดของขดลวดที่ถูกกระตุ้นจะทำให้เพิ่มแรงบิดได้มากกว่าแบบเวฟ โรเตอร์จะเคลื่อนที่ด้วยแรงดึงอย่างเต็มที่ด้วยแรงดึงจาก 2 ขดลวดที่ถูกกระตุ้นพร้อมกัน ข้อเสียของการกระตุ้นแบบนี้ คือ การกระตุ้นแบบนี้ต้องจ่ายกำลังไฟฟ้ามากขึ้น การทำงานต่าง ๆ จะแสดงในรูปที่ 2.44

**ตารางที่ 2.18** การป้อนกระแสแบบเวฟ

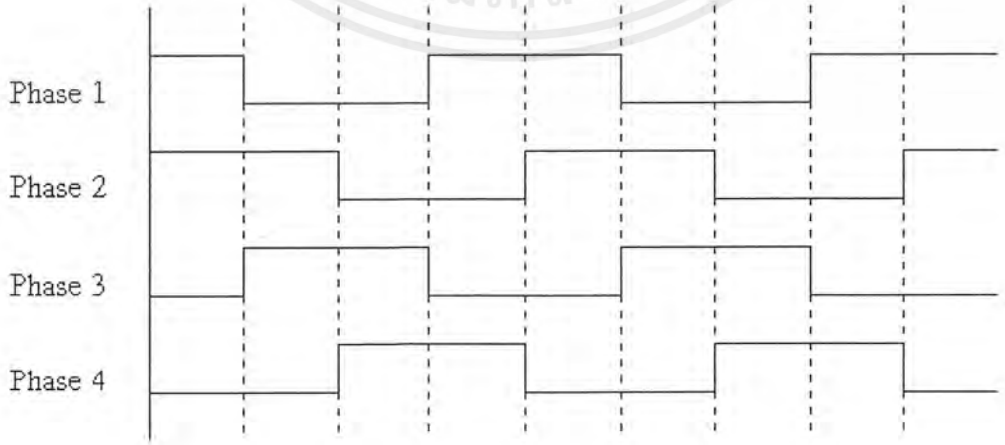
| Step | Phase | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| 1 | 1 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 1 |




**รูปที่ 2.43** แสดงสภาวะการทำงานมอเตอร์แบบเวฟ

**ตารางที่ 2.19** การป้อนกระแสแบบ 2 เฟส

| Step | Phase | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| 1 | 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 1 | 0 |
| 3 | 0 | 0 | 1 | 1 |
| 4 | 1 | 0 | 0 | 1 |

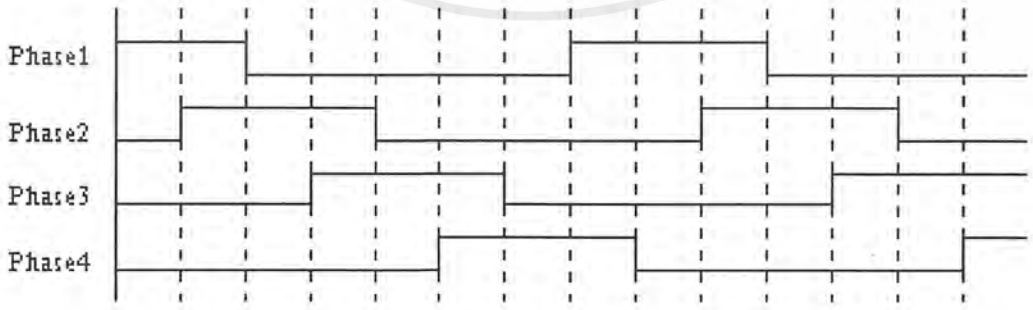



**รูปที่ 2.44** แสดงสภาวะการทำงานมอเตอร์แบบ 2 เฟส

**3) แบบครึ่งสเต็ป (Half step)** เป็นแบบที่ได้จากการผสมระหว่างการกระตุ้นแบบเวฟและแบบ 2 เฟส ดังแสดงในรูปที่ 2.45 เพื่อเพิ่มจำนวนสเต็ปต่อรอบอีกหนึ่งเท่าตัว แรงบิดที่ได้จากการกระตุ้นแบบนี้จะเพิ่มมากขึ้นอีก เพราะช่วงสเต็ปมีระยะสั้นลง และแต่ละสเต็ปเกิดจากแรงดึงของขดลวด 2 ขด ที่ถูกกระตุ้นพร้อมกัน ความถูกต้องของตำแหน่งจึงมีเพิ่มมากขึ้น ที่สำคัญการกระตุ้นแบบนี้จะต้องทำการหมุน 2 สเต็ปจึงเท่ากับ 1 สเต็ปของ 2 แบบแรก ส่วนแหล่งจ่ายกำลังไฟฟ้าต้องใช้เหมือนกับแบบ 2 เฟส

**ตารางที่ 2.20** การป้อนกระแสแบบ ครึ่งสเต็ป

| Step | Phase | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| 1 | 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 |
| 4 | 0 | 0 | 1 | 0 |
| 5 | 0 | 0 | 1 | 1 |
| 6 | 0 | 0 | 0 | 1 |
| 7 | 1 | 0 | 0 | 1 |
| 8 | 1 | 0 | 0 | 0 |




**รูปที่ 2.45** แสดงสภาวะการทำงานมอเตอร์แบบครึ่งสเต็ป

# บทที่ 3
# การออกแบบ การสร้าง และการทำงาน

## 3.1 กล่าวนำ

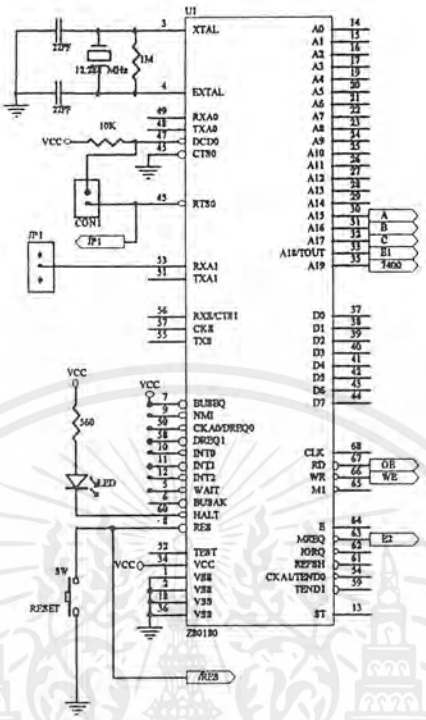
ในการออกแบบชุดฝึก PLC โปรแกรมด้วยคอมพิวเตอร์ จะประกอบด้วย 2 ส่วนสำคัญ คือ ส่วนของวงจรควบคุม ซึ่งได้แก่ ส่วนที่เป็น Hardware และส่วนที่เป็น Software ในส่วนของ Hardware แบ่งส่วนประกอบย่อยๆ ได้กล่าวคือ วงจรควบคุมซึ่งมีไมโครโปรเซสเซอร์ Z80180 เป็นหัวใจสำคัญ ส่วนหน่วยความจำซึ่งได้แก่ ROM และ RAM ทำหน้าที่ในการเก็บ Source Code และโปรแกรมย่อยต่างๆ ที่ใช้ทั้งหมดในวงจร ส่วนรับข้อมูลหรือ Input ทำหน้าที่ในการรับข้อมูลเพื่อทำการส่งข้อมูลให้แก่ไมโครโปรเซสเซอร์ เพื่อนำไปประมวลผลต่อไป ในส่วนนี้ได้แก่ Matrix Switch และ Input Switch ต่างๆ ส่วนของการแสดงผลหรือ ภาค Output ใช้ LCD และ LED ในการแสดงผล ส่วน Interface ระหว่างไมโครโปรเซสเซอร์ กับ Port ติดต่อภายนอก โดยใช้ Port ของ 8255 ในการ Interface

## 3.2 การออกแบบวงจร

ดังที่ได้กล่าวมาแล้วชุดฝึก PLC โปรแกรมด้วยคอมพิวเตอร์ หัวใจของชุดฝึกนี้อยู่ที่ ไมโครโปรเซสเซอร์ Z80180 ซึ่งขอได้กล่าวเป็นประเด็นย่อยๆ ดังนี้

### 3.2.1 วงจรควบคุม

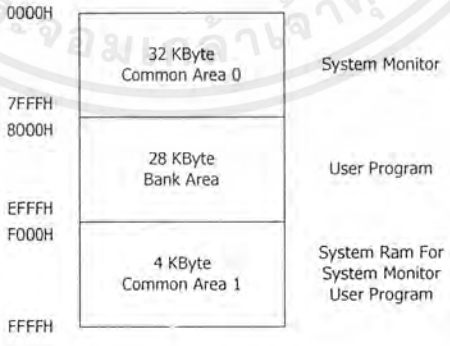
จะทำหน้าที่ควบคุมการรับส่งข้อมูล รวมทั้งการประมวลผลต่างๆ ภายในไมโครโปรเซสเซอร์ ซึ่งในวงจรนี้ใช้ไมโครโปรเซสเซอร์ เบอร์ Z80180 เป็นตัวประมวลผลข้อมูลที่รับเข้ามา และใช้คริสตัล 12.288 MHz เป็นตัวกำเนิดฐานเวลา



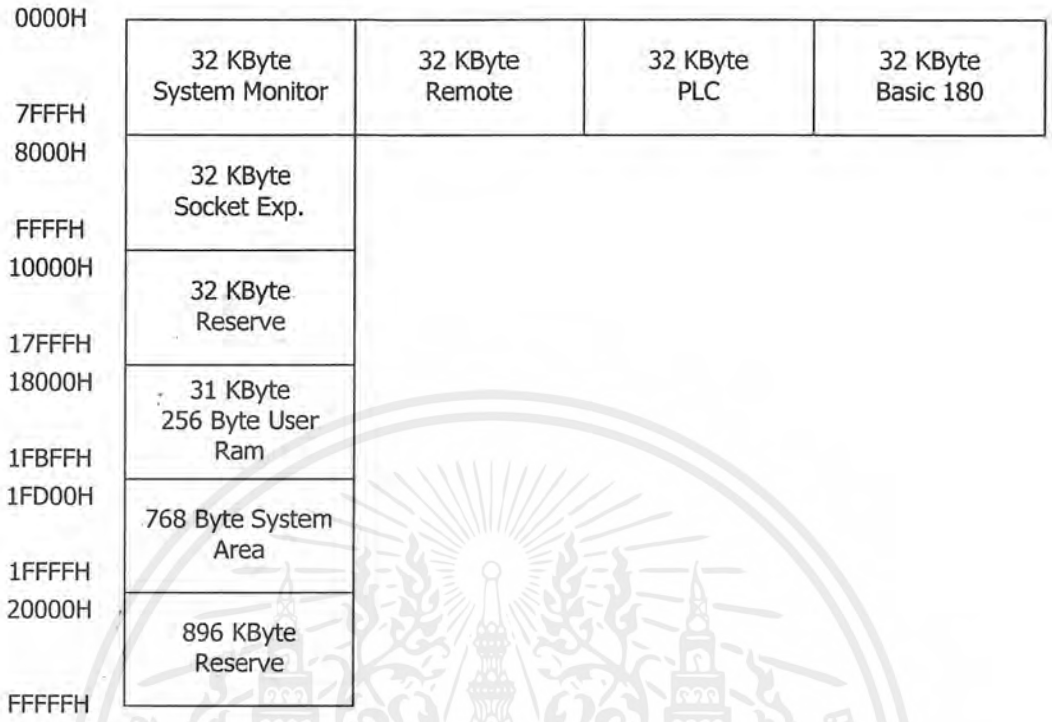

รูปที่ 3.1 วงจรควบคุม

### 3.2.2 หน่วยความจำ

การจัดการหน่วยความจำ ROM และ RAM ในที่นี้ขอให้ดูจากการ Memory Map จะทำให้เข้าใจได้ดียิ่งขึ้น




รูปที่ 3.2 การจัด Memory Map CPU Z80180 แบบ Logical

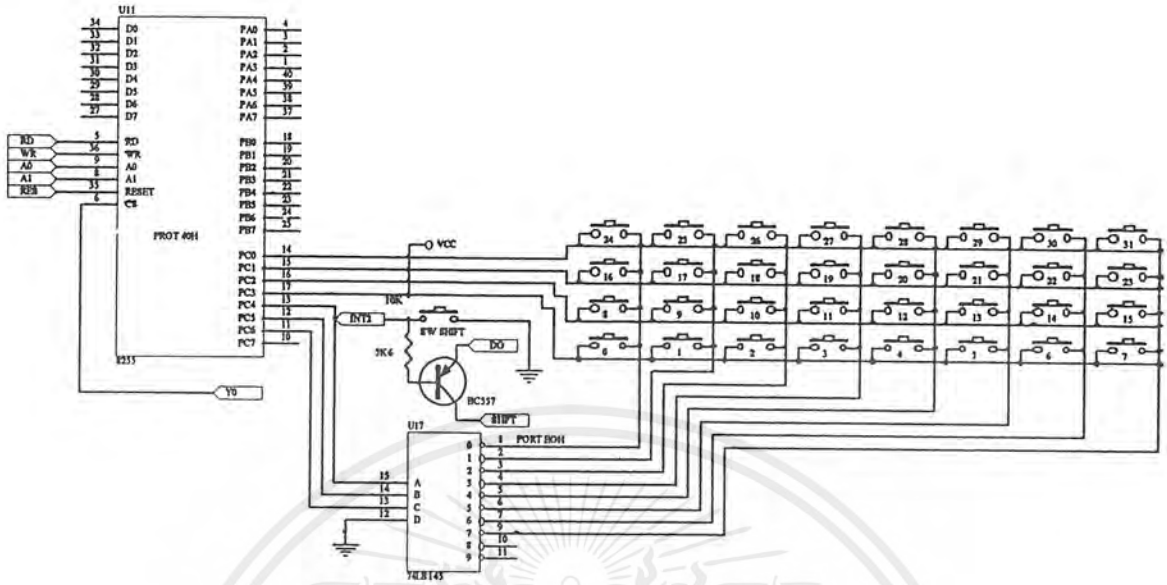



**รูปที่ 3.3** การจัด Memory Map CPU Z80180 แบบ Physical

หลังจากที่ได้ทำการ Memory Map แล้วจึงทำการเลือก EPROM โดยกำหนดให้พื้นที่ 0000H – 17FFFH ใช้ EPROM เบอร์ 27C1001 พื้นที่ 18000H – 1FFFFH ใช้ RAM เบอร์ 62256

### 3.2.3 ส่วนของวงจรรับข้อมูล

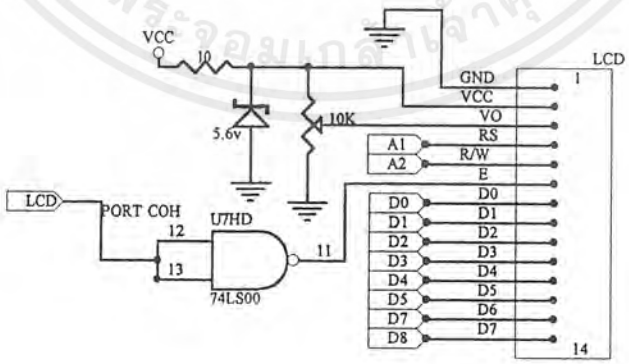
ในส่วนของวงจรรับข้อมูลนั้นใช้ Matrix Switch ในการรับสัญญาณข้อมูลจากการกดรหัสของผู้ที่ใช้งาน Matrix Switch มี 32 ปุ่มกด โดยแบ่งการเดินสายออกเป็นทางแนวนอน 8 เส้น และทางแนวตั้ง 4 เส้น ดังนั้นจึงทำให้ได้ปุ่มกด 32 ปุ่มตามที่ต้องการ




รูปที่ 3.4 Matrix Switch ของวงจรควบคุม

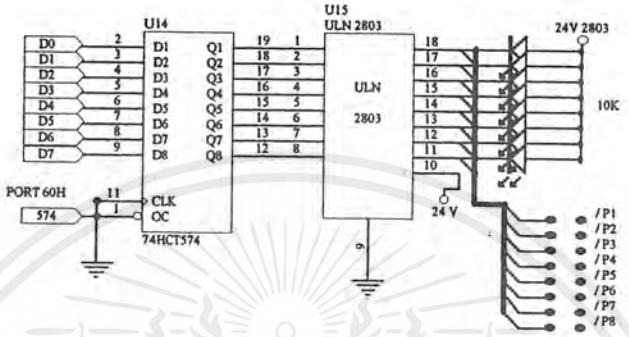
### 3.2.4 ส่วนของการแสดงผล

ในส่วนของการแสดงผลนั้น จะใช้ LCD และ LED ในการแสดงผล การทำงานของวงจรแสดงผล LCD เริ่มจากการส่งข้อมูลกำหนดการอ่านหรือเขียน แล้วจึงกำหนดการอ่านหรือเขียนนั้นเป็นข้อมูลหรือคำสั่ง ส่งข้อมูลออกไปทางขา Data (D0 – D7) เป็นการทำงานแบบ 8 บิต การแสดงผลบรรทัดที่ 1 และ 2 ต้องกำหนดบัสแอดเดรส ซึ่งตำแหน่งในการแสดงผลบรรทัดที่ 1 จะอยู่ที่แอดเดรส 80H บรรทัดที่ 2 จะอยู่ที่แอดเดรส 0C0H



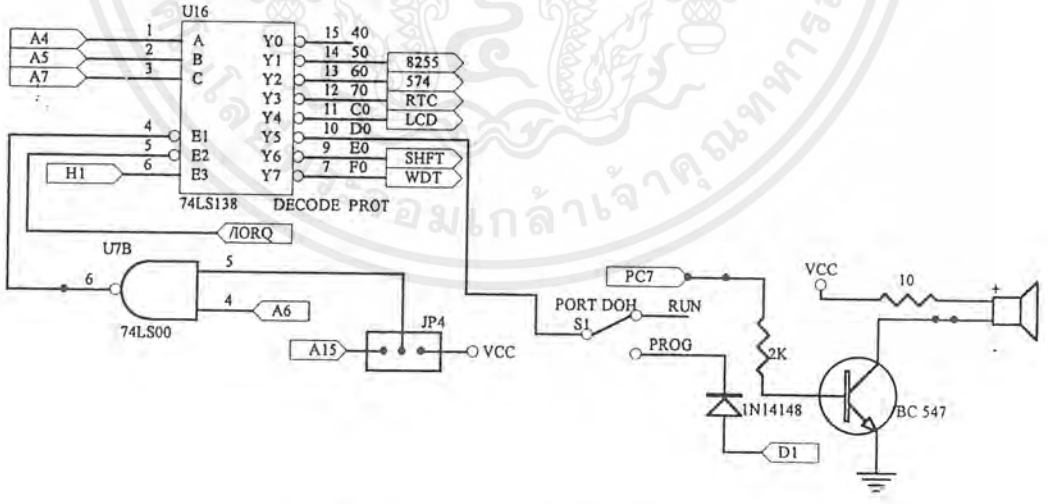

รูปที่ 3.5 ภาคแสดงผลโดยใช้ LCD

ส่วนวงจรแสดงผลโดยใช้ LED นั้นเป็นการส่งข้อมูลให้กับ LED โดยต้องทำการส่งลอจิก "0" ออกไปทางขา Data (D0 – D7) ซึ่งเป็นการทำงานแบบ 8 บิต เช่นเดียวกับการแสดงผลที่ใช้ LCD เช่นกัน




รูปที่ 3.6 ภาคแสดงผลโดยใช้ LED

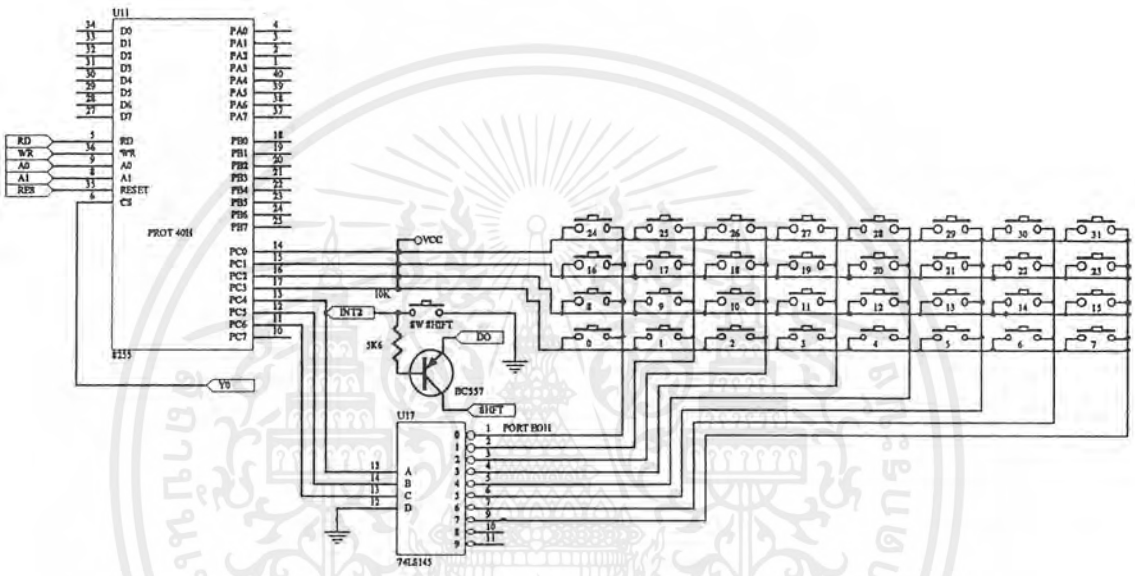
ส่วนวงจรแสดงผลโดยใช้ลำโพงนั้นเป็นการส่งข้อมูลให้กับ Transistor โดยต้องทำการส่งข้อมูลไปที่ Decode Port ที่แอดเดรส "D0" ออกไปทางขา Y5 ของ 74LS138 ซึ่งจะทำให้เกิดสัญญาณเสียงดังออกที่ลำโพง ระยะเวลาการดังหรือความถี่ขึ้นอยู่กับผู้เขียนโปรแกรมควบคุม



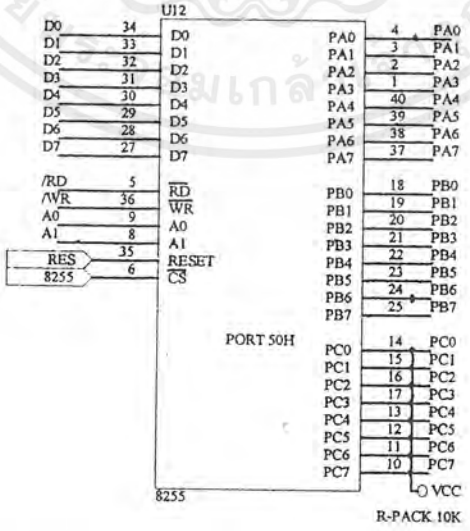

รูปที่ 3.7 ภาคแสดงผลโดยใช้ลำโพง

### 3.2.5 ส่วนการ Interface

การติดต่อสื่อสารระหว่างหน่วยอินพุตกับหน่วยประมวลผล หรือหน่วยประมวลผลกับหน่วยแสดงผลนั้น จำเป็นต้องมีตัวกลางที่คอยทำหน้าที่เป็นถนนหรือทางเดินให้กับอุปกรณ์ซึ่งในส่วนนี้ เราเรียกว่าหน่วย Interface การ Interface เกือบทั้งหมดในวงจรนี้เชื่อมต่อโดยใช้ 8255 เกือบทั้งสิ้น



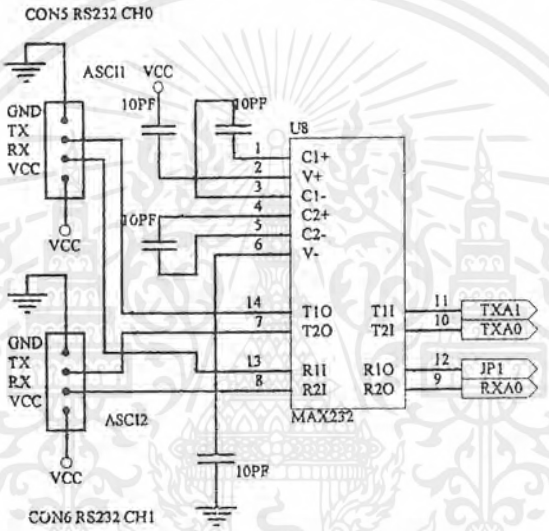

รูปที่ 3.8 การ Interface ระหว่าง 8255 กับ Matrix Switch



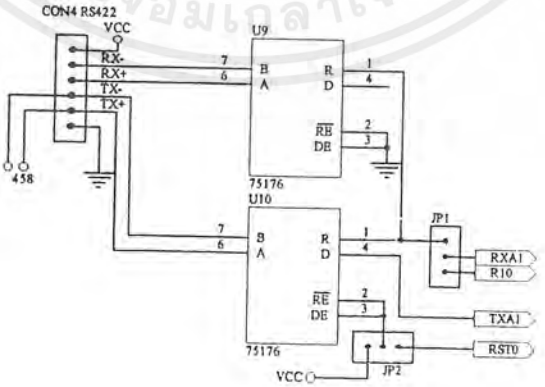

รูปที่ 3.9 การ Interface ระหว่าง 8255 กับอุปกรณ์ภายนอก

### 3.2.6 ส่วนการสื่อสารข้อมูลแบบอนุกรม

ในที่นี้เราได้ทำการจัดแบ่งประเภทการสื่อสารแบบอนุกรมในวงจรนี้ออกเป็น 2 ประเภท ด้วยกันคือ การสื่อสารข้อมูลแบบอนุกรมRS-232 และการสื่อสารข้อมูลแบบอนุกรม RS-485 การสื่อสารข้อมูลแบบอนุกรม RS-232 เราได้จัดวงจรในลักษณะใช้ไอซีสำเร็จรูป MAX232 ในการติดต่อกับPort อนุกรมภายนอก ซึ่งการส่งข้อมูลในระบบการติดต่อสื่อสารแบบนี้เราสามารถติดต่อข้อมูลได้ทีละบิต แต่เป็นการช่วยประหยัดสายสัญญาณที่จะนำมาใช้งานได้เป็นอย่างดี ดังรูป




**รูปที่ 3.10** วงจรสื่อสารข้อมูลแบบอนุกรม RS-232




**รูปที่ 3.11** วงจรสื่อสารข้อมูลแบบอนุกรม RS-485

# บทที่ 4

# การทดลองและผลการทดลอง

ในบทนี้เป็นการทดลอง วงจรส่วนต่างๆของชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ และชุดอินเตอร์เฟส เพื่อให้ง่ายแก่การทดลองและการตรวจสอบการทำงานของระบบจึงแบ่งการทดลองออกเป็น 7 ส่วน คือ ส่วนที่หนึ่งเป็นส่วนของการแสดงผลเอาต์พุตของชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ ส่วนที่สองเป็นส่วนของการทำลองชุดรีเลย์ ส่วนที่สามเป็นส่วนของการทดลอง ชุด DC Motor ส่วนที่สี่เป็นส่วนของการทดลองชุด Stepping Motor ส่วนที่ห้าเป็นส่วนของการทดลองชุด Switch ส่วนที่หกเป็นส่วนของการทดลองชุด Rraffic Light ส่วนที่เจ็ดเป็นส่วนของการทดลองการรับ-ส่ง ข้อมูล ระหว่าง คอมพิวเตอร์และชุดคอมพิวเตอร์และชุดสาธิตการทำงาน PLC ควบคุมด้วยไมโครโปรเซสเซอร์

## 4.1 การทดลองส่วนของการแสดงผลของเอาต์พุตของชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์

### 4.1.1 ลำดับการทดลอง

1) นำชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ มาทำการเปลี่ยน Jumper ของการกำหนด แรงดันของชุดแสดงผลเอาต์พุตให้เป็น 10 V ซึ่งได้มาจาก แรงดันภายในชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ดังรูปที่ 4.1

2) ทำการ CLS โปรแกรม ที่อยู่ในหน่วยความจำ โดยการกด

FUNC 7 8 CLR MONT CLR

3) ทำการเขียนโปรแกรมไฟวิ่งเพื่อทดสอบการทำงานของเอาต์พุต ตามโปรแกรมข้างล่างนี้

| | |
|---|---|
| LD NOT | 0900 |
| OUT | 0901 |
| LD NOT TIM | 0000 |
| OUT | 0902 |
| LD | 0902 |
| TIM | 00 |
| TIM DATA # | 0010 |

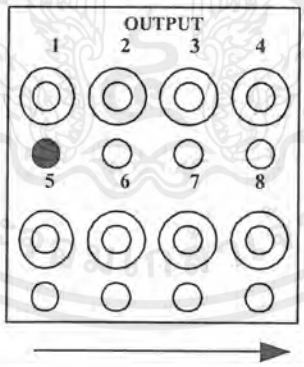
```
LD              0707
OR              0901
LD  NOT  TIM    0000
LD              0000
SFTREG          0007
LD   NOT        0903
OUT             0900
END (01)
```

4) รันโปรแกรม

5) สังเกตการแสดงผลของ LED

### 4.1.2 ผลการทดลอง

จากการทดลองส่วนของการแสดงผลเอาต์พุตของชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ ผลที่ได้เมื่อทำการรันโปรแกรมไฟวิ่งจากซ้ายไปขวา แล้ว ไฟ LED ที่เป็นส่วนของเอาต์พุตนั้นจะติดทางด้านซ้ายไปทางด้านขวามือจะติดดับที่ละดวง ซึ่งแสดงว่าส่วนของ เอาต์พุตสามารถทำงานได้



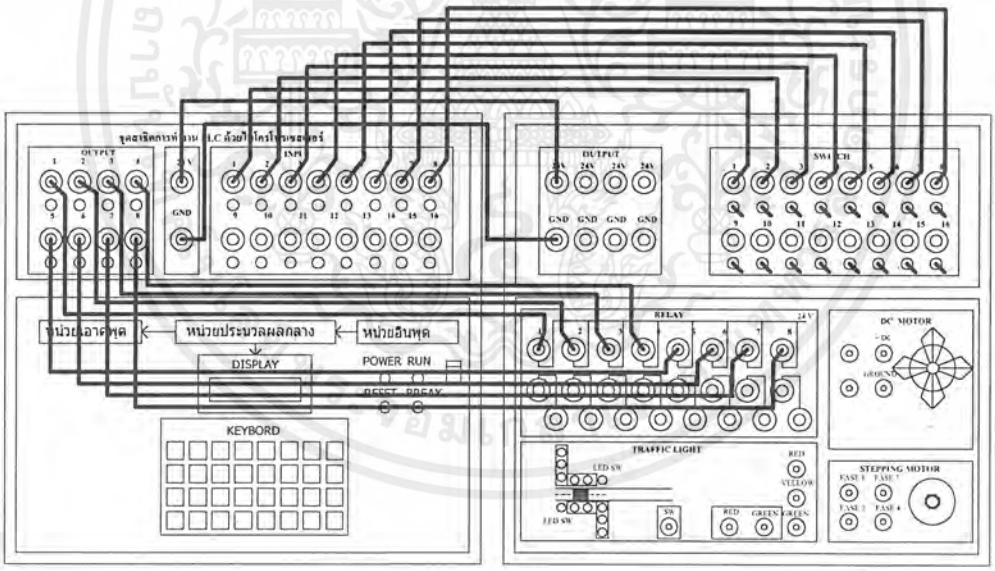

**รูปที่ 4.1** แสดงผลการรันโปรแกรม

## 4.2 การทดลองส่วนของการทดลองชุดรีเลย์และ สวิทช์

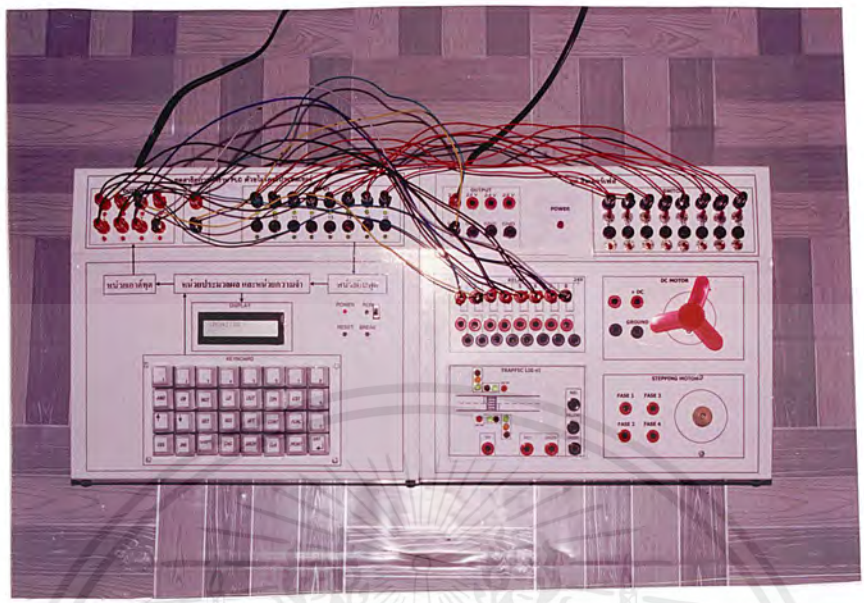
### 4.2.1 ลำดับขั้นการทดลอง

1) ต่อไฟ 24 V ที่ได้จากชุดอินเตอร์เฟส เข้ากับชุดสาธิตการทำงาน PLC

2) ทำการต่อวงจรเพื่อทดสอบชุดรีเลย์ ในการต่อวงจรนี้จะต้องใช้สายต่อที่มีแจ๊ค ตัวผู้ 2 ตัวต่ออยู่กับขั้วสายและปลายสาย จะใช้ทั้งหมดจำนวน 18 เส้น นำสายต่อ 2 เส้น ขั้วสายทั้ง 2 เส้น ต่อ 24 V และ GND ที่ชุดอินเตอร์เฟส และปลายสายต่อ 24 V และ GND ที่ชุดสาธิต PLC เพื่อทำการต่อไฟแล้วจึงทำการเชื่อมต่อสวิทช์ จะใช้สวิทช์ทั้งหมด 8 ตัว ที่ชุดอินเตอร์เฟสนำสายต่อขั้วของสายต่อสวิทช์ 1 ถึง 8 ในชุดอินเตอร์เฟส และอีกปลายสายต่อเข้ากับ Input ชองชุดสาธิต PLC เมื่อต่อสวิทช์แล้วต่อไฟเข้าแล้ว ก็จะมาเชื่อมต่อรีเลย์กันบ้าง นำสายเชื่อมต่อ 8 เส้นขั้วของสายต่อเข้ากับรีเลย์ 1 ถึง 8 ที่ชุดอินเตอร์เฟส ปลายสายต่อที่ Output 1 ถึง 8 ที่ชุดสาธิต PLC ดังรูปที่ 4.2




**รูปที่ 4.2** แสดงการต่อทดสอบชุดรีเลย์และสวิทช์



**รูปที่ 4.3 รูปการต่อทดลองจริง**

3) ทำการ CLS โปรแกรมที่อยู่ในหน่วยความจำ โดยการกด

FUNC 7 8 CLR MONT CLR

4) เมื่อทำการ CLS แล้ว ทำการป้อนโปรแกรมการทดสอบตามนี้

| | |
|---|---|
| LD | 0000 |
| OUT | 0700 |
| LD | 0001 |
| OUT | 0701 |
| LD | 0002 |
| OUT | 0702 |
| LD | 0003 |
| OUT | 0703 |
| LD | 0004 |
| OUT | 0704 |
| LD | 0005 |
| OUT | 0705 |
| LD | 0006 |
| OUT | 0706 |

LD 0007

OUT 0707

END (01)

5) รันโปรแกรม

### 4.2.2 ผลการทดลอง

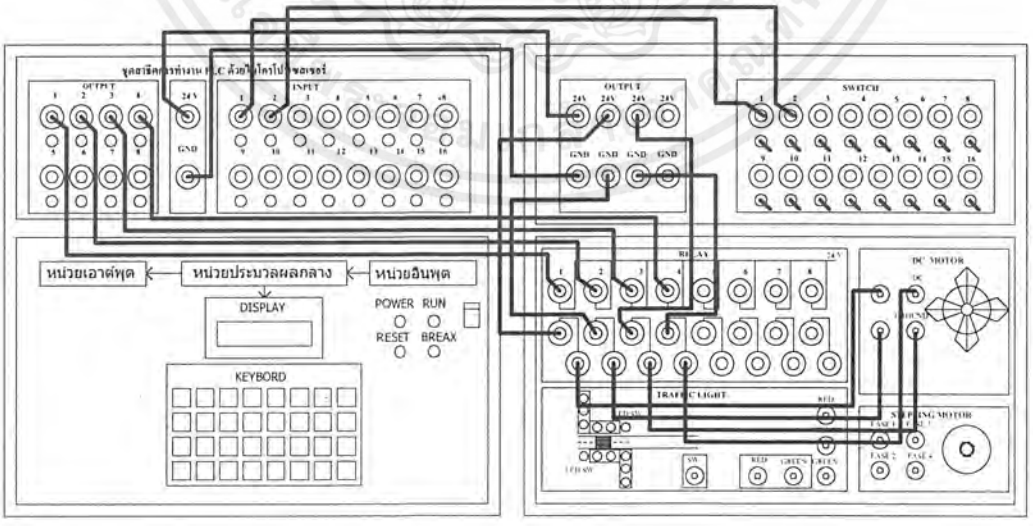
จากการทดลอง ป้อนโปรแกรม และทำการรันโปรแกรม สวิทช์ทั้ง 8 ตัว คือตัวที่ 1 ถึง 8 นั้น แต่ละตัวจะควบคุมรีเลย์ 1 ตัว สวิทช์ตัวที่ 1 ควบคุมรีเลย์ตัวที่ 1 สวิทช์ตัวที่ 2 ควบคุมรีเลย์ตัวที่ 2 จะเรียงตามลำดับเช่นนี้จนครบ 8 ตัว สังเกตจากใช้มิเตอร์วัดที่หน้าสัมผัสตัวที่ 1 เพื่อทำการโยกสวิทช์ค่าความต้านทานจะขึ้น เพราะหน้าคอนแทครีเลย์จะถึงกัน เมื่อโยกสวิทช์ตัวเดิมกลับไปที่ตำแหน่งเดิมหน้าสัมผัสจะเปิดทำให้ไม่มีความต้านทานขึ้น ทดสอบทั้ง 8 ตัวผลการทดสอบสวิทช์สามารถควบคุมรีเลย์ได้ สรุปว่า สวิทช์ สามารถควบคุม รีเลย์ได้ แสดงว่า ชุดสาธิต PLC ทางด้าน Input ใช้ได้

## 4.3 การทดลองส่วนของชุด DC Motor

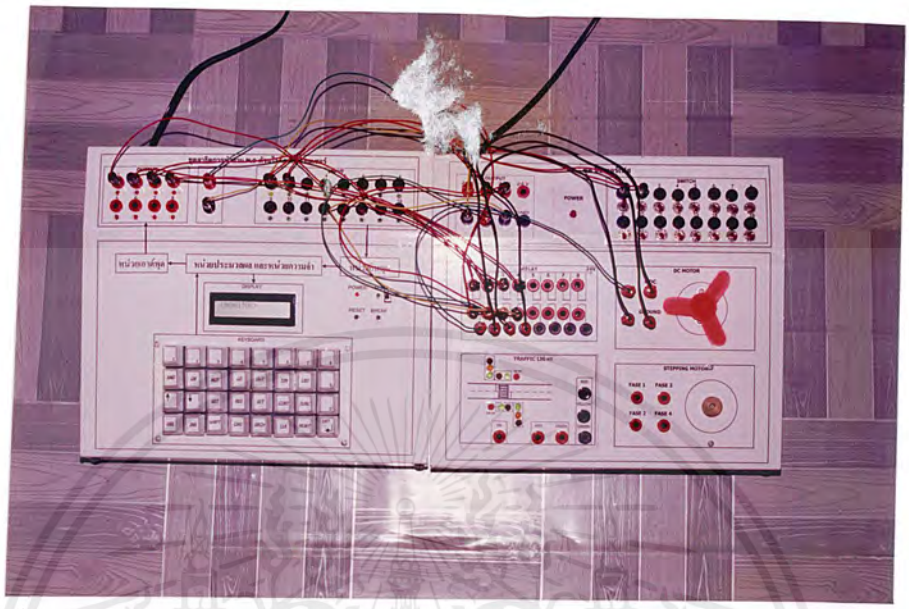
### 4.3.1 ลำดับขั้นตอนการทดลอง

1) นำชุดสาธิตการทำงาน PLC ด้วยไมโคร ต่อไฟ 24 V GND เข้ากับชุดอินเตอร์เฟส

2) ทำการต่อวงจรเพื่อทดสอบชุด DC Motor ดังรูป




รูปที่ 4.4 การต่อวงจร DC Motor เพื่อทดลอง



**รูปที่ 4.5** แสดงการต่อและทดลองจริง

3) เมื่อต่อวงจรเสร็จแล้ว ทำการป้อนโปรแกรมดังนี้

| | |
|---|---|
| LD | 0000 |
| LD NOT TIM | 01 |
| TIM | 00 |
| TIM DATA | 0020 |
| LD TIM | 00 |
| OUT | 700 |
| OUT | 701 |
| LD NOT TIM | 03 |
| TIM | 01 |
| TIM DATA | 0020 |
| LD TIM | 01 |
| AND NOT TIM | 03 |
| TIM | 02 |
| TIM DATA | 0020 |
| LD TIM | 02 |

| | |
|---|---|
| OUT | 702 |
| OUT | 703 |
| LD NOT TIM | 00 |
| TIM | 03 |
| TIM DATA | 0030 |
| END (01) | |

4) ทำการรันโปรแกรม

5) สังเกตการทำงานของ DC Motor

### 4.3.2 สรุปผลการทดลอง

จากการทดลองส่วนของชุด DC Motor เมื่อรันโปรแกรมแล้วสังเกตว่า DC Motor หมุนตามที่ โปรแกรมสั่งงาน คือ หมุนตามเข็มนาฬิกา แล้วหยุดหมุน แล้วก็หมุนทวนเข็มนาฬิกาเป็นอย่างนี้ไปตลอดแสดงว่าชุด DC Motor ทำงานได้

## 4.4 การทดลองส่วนของชุด Stepping Motor

### 4.4.1 ลำดับการทดลอง

1) ทำการต่อไฟ 24 V จากชุดอินเตอร์เฟส เข้ากับชุดสาธิตการทำงาน PLC

2) ทำการต่อวงจรเพื่อทดสอบชุด Stepping Motor ในการต่อวงจร Stepping นี้จะใช้ Output ที่ 1 ถึง 4 ของชุดสาธิตการทำงาน PLC ต่อเข้ากับชุด Stepping Motor โดยใช้สายเชื่อมต่อจำนวน 4 เส้น ขั้วสายทั้ง 4 เส้นต่อที่ Output 1 ถึง 4 ของชุดสาธิต PLC ปลายสายต่อกับ Stepping Motor 1 ต่อ กับ 1 , 2 ต่อกับ 2 , 3 ต่อกับ 3 , 4 ต่อกับ 4

3) ทำการ CLR โปรแกรมที่อยู่ในหน่วยความจำที่ไม่ได้ใช้ออกให้หมดด้วยการ กด

FUNC 7 8 CLR MONT CLR

4) ทำการป้อนโปรแกรม ดังนี้

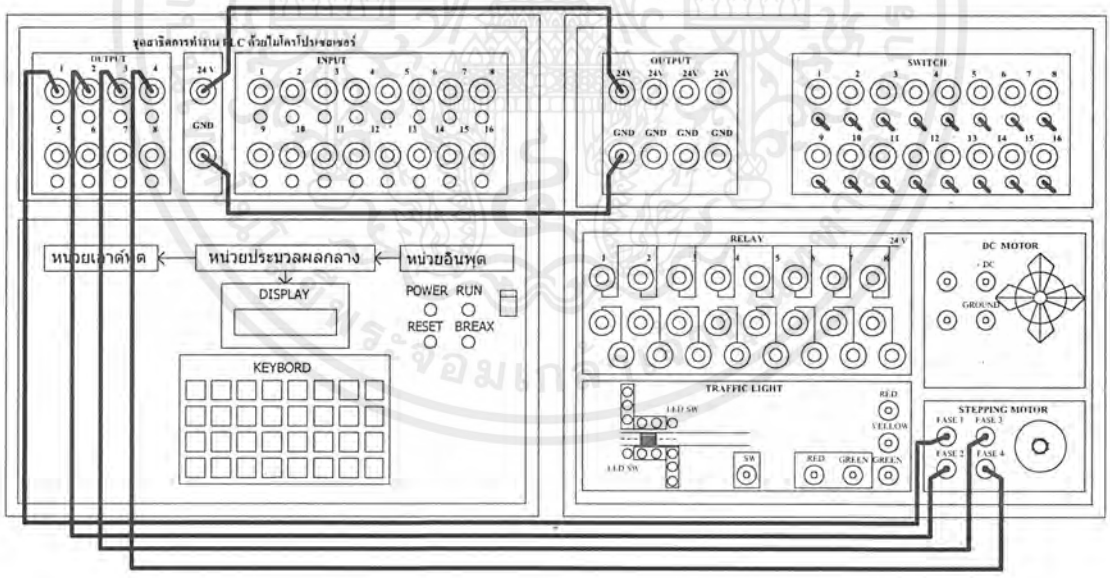
| | |
|---|---|
| LD NOT | 0900 |
| OUT | 0901 |
| LD NOT TIM | 0000 |
| OUT | 0902 |
| LD | 0902 |
| TIM | 00 |

| | |
|---|---|
| TIM DATA # | 0001 |
| LD | 0703 |
| OR | 0901 |
| LD NOT TIM | 0000 |
| LD | 0000 |
| SFTREG | 0007 |
| LD NOT | 0903 |
| OUT | 0900 |
| END(01) | |

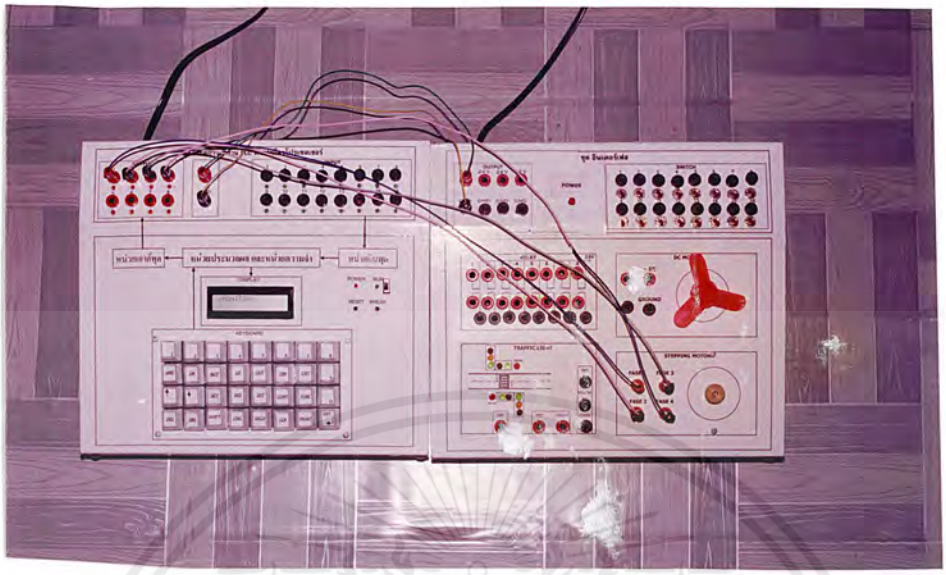
5) ทำการรันโปรแกรม

### 4.4.2 ผลการทดลอง

จากการทดลองรันโปรแกรมสังเกตที่ Stepping Motor มีผลตอบสนองคือ Stepping Motor มีการหมุนจากซ้ายไปขวา หมุนทีละ step step ละ1 วินาที แสดงว่าชุด สเต็ปปิ้งทำงานได้




**รูปที่ 4.6** แสดงการต่อวงจร Stepping Motor



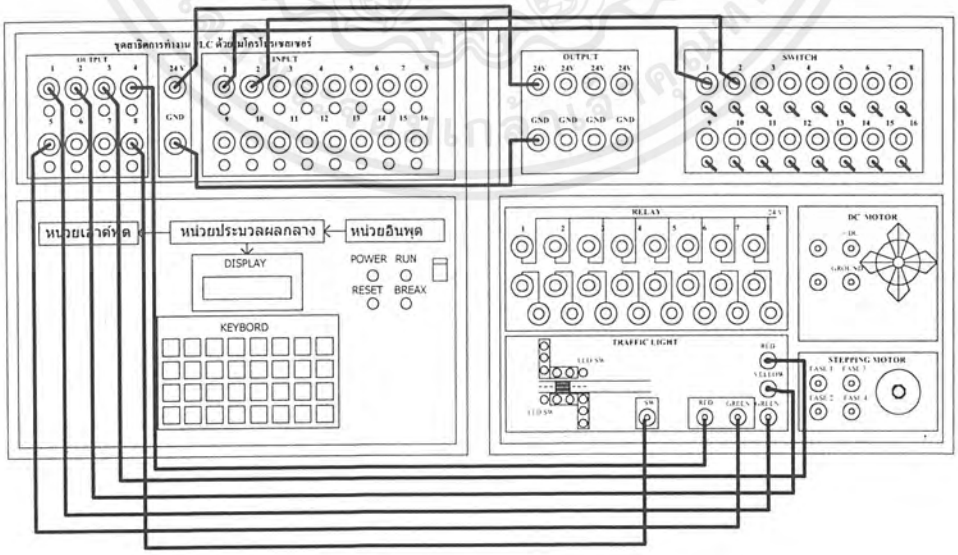
**รูปที่ 4.7** แสดงการต่อและทดลองจริง

## 4.5 การทดลองส่วนของชุด การควบคุมไฟจราจร

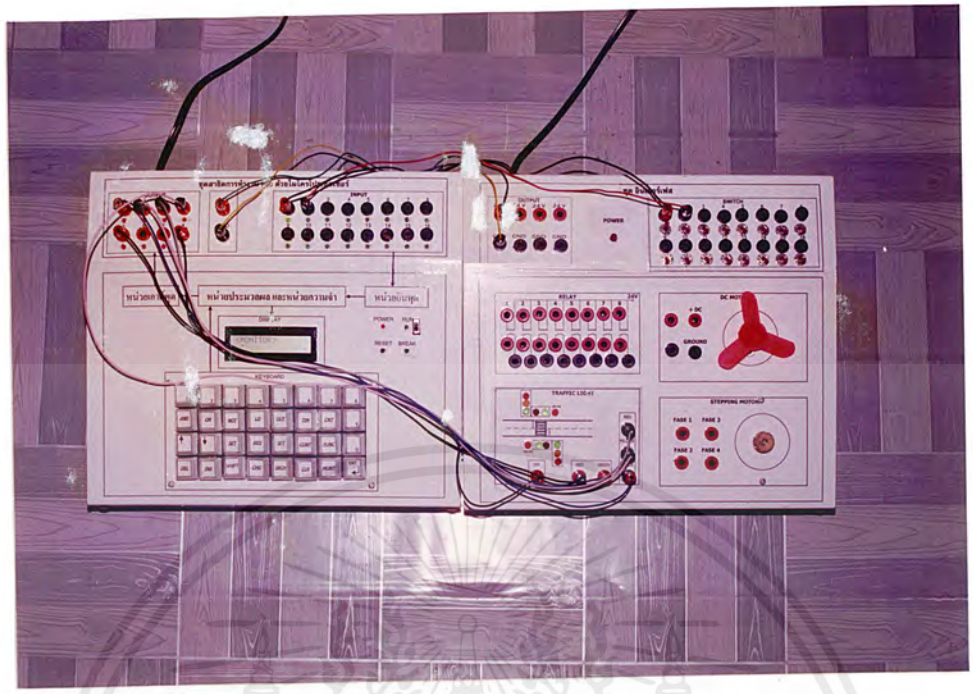
### 4.5.1 ลำดับขั้นการทดลอง

1) ทำการต่อไฟ 24 V และ Gnd จากชุดอินเตอร์เฟส เข้ากับชุดสาธิตการทำงาน PLC

2) ทำการต่อวงจรควบคุมไฟจราจร ดังรูป




**รูปที่ 4.8** การต่อวงจรไฟจราจร



**รูปที่ 4.9** แสดงการต่อและทดลองจริง

3) ทำการ CLR โปรแกรมที่อยู่ในหน่วยความจำ โดยการกด

FUNC 9 8 CLR MONT CLR

4) ป้อนโปรแกรมดังนี้

| | |
|---|---|
| LD | 0000 |
| OR | 0001 |
| OR | 0900 |
| AND NOT TIM | 0006 |
| OUT | 0900 |
| OUT | 0707 |
| LD | 0900 |
| TIM | 00 |
| TIM DATA | #0100 |
| LD TIM | 0000 |
| TIM | 01 |
| TIM DATA | #0050 |
| LD TIM | 0001 |

```
TIM              02
TIM DATA         #0020
LD  TIM          0002
TIM              03
TIM DATA         #0060
LD  TIM          0003
AND NOT TIM      0005
LD  CNT          0000
AND              0900
OR  LD           0000
TIM              04
TIM  DAT A       #0005
LD   TIM         0004
TIM              05
TIM  DATA        #0005
LD  TIM          0004
LD  NOT          0900
CNT              00
CNT  DATA        #0006
LD   CNT         0000
TIM              06
TIM  DATA        # 0020
LD               0900
AND  NOT  TIM    0000
OR   NOT         0900
OUT              0700
LD               0900
AND  TIM         0000
AND  NOT  TIM    0001
OUT              0701
```

| | |
|---|---|
| LD | 0900 |
| AND NOT | 0700 |
| AND NOT | 0701 |
| OUT | 0702 |
| LD NOT TIM | 0002 |
| OR CNT | 0000 |
| OUT | 0703 |
| LD TIM | 0002 |
| AND NOT TIM | 0003 |
| OR TIM | 0004 |
| AND NOT CNT | 0000 |
| OUT | 0704 |

5) รันโปรแกรมและสังเกตการทดลอง

### 4.5.2 ผลการทดลอง

เมื่อ Switch กดอยู่ทั้ง 2 ด้าน โดยยังไม่กดปุ่มทางด้านใด ไฟเขียวทางด้านรถวิ่งจะติด และไฟแดงทางด้านคนข้ามจะติดค้าง เปิดสวิทช์ทางด้านใดถูกกดไฟ สถานะการกดจะติดค้าง ต่อจากนั้น 10 วินาที ไฟเขียวทางด้านรถวิ่งจะเปลี่ยนเป็นเหลืองนานเป็นเวลา 5 วินาที จากนั้นไฟแดงจะติดขึ้น เมื่อไฟแดงติด 2 วินาที ทางด้านคนข้ามจะเปลี่ยนจากแดงเป็นเขียวค้างอยู่นาน 6 วินาที จากนั้นก็จะกระพริบทุก 1 วินาที จำนวน 5 ครั้ง และเปลี่ยนเป็นแดงจนผ่านไป 2 วินาที ไฟแดงทางด้านรถวิ่งจะเปลี่ยนเป็นเขียว และไฟแสดงการกดสวิทช์ดับ

## 4.6 การทดลองของส่วนของการรับ-ส่ง ข้อมูลระหว่าง คอมพิวเตอร์ กับ ชุด สาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์

### 4.6.1 ลำดับขั้นการทดลอง

1) ต่อ Port RS-232C ระหว่าง ระหว่าง คอมพิวเตอร์ กับ ชุด สาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์

2) Set การรับส่งข้อมูลของ Port Com1 โดยให้เป็นมาตรฐานเดียวกับชุด สาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ คือ 8 Bit Data, Non Parity และ 1 Stop Bit โดยมีอัตราการรับ-ส่ง 9600 บิต ต่อ วินาที โดยใช้คำสั่ง Mode Com1:96,n,8,1

3) ทดลองการรับข้อมูล (ข้อมูลเป็น format.hex) จาก ชุด สาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ มาเก็บไว้ที่ คอมพิวเตอร์ โดยการใช้ คำสั่ง Copy Com1 Test1.Cod ที่คอมพิวเตอร์ และ คำสั่ง FUNC 98 ที่ ชุด สาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์

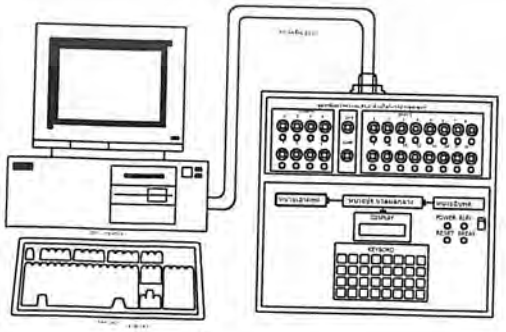
4) ทดลองการส่งข้อมูล (ข้อมูลเป็น format.hex) จาก คอมพิวเตอร์ มาเก็บไว้ที่ ชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ โดยการใช้ คำสั่ง Copy Test1.Cod Com1 ที่คอมพิวเตอร์ และ คำสั่ง Func 97 ที่ ชุด สาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์

5) ทดลองการส่งข้อมูล(ข้อมูลเป็น format ของภาษา Boolean) จาก ชุด สาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ มาเก็บไว้ที่ คอมพิวเตอร์ โดยการใช้ คำสั่ง Copy Com1 Test1.Lad ที่คอมพิวเตอร์ และ คำสั่ง Func 95 ที่ ชุด สาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์

6) ตรวจสอบข้อมูลโดยใช้โปรแกรม EDIT

### 4.6.2 ผลการทดลอง

ในการทดลองนี้ จะเป็นการส่งข้อมูล จาก PLC ไปยังคอมพิวเตอร์ และจาก คอมไปยัง PLC ในการส่งข้อมูล จาก PLC ไปยังคอมพิวเตอร์นั้น การส่งจะส่ง ได้ 2 แบบคือ เป็น File จุด HEX คือ เป็นเลขฐาน 16 กับ File จุด Lad เป็น File ที่เก็บ เป็น โปรแกรม Ladder เลย ผลการทดลอง สามารถทำได้ ส่วนการส่ง File ที่มีโปรแกรมไป RUN ที่ PLC นั้น ทำการเลือกโปรแกรมที่จะส่ง จะต้อง Fave File เป็นจุด HEX หรือ จุด Lad เท่านั้น เมื่อทำการส่ง Fave แล้วจะสังเกตเห็นว่า ที่ PLC นั้นจะมีโปรแกรม Fave นั้นอยู่ สรุปว่าการติดต่อข้อมูลจาก PLC ไป คอมพิวเตอร์ และ จาก คอมพิวเตอร์ ไปยัง PLC เป็นสำเร็จ



**รูปที่ 4.10** แสดงการเชื่อมต่อระหว่างชุดสาธิต PLC กับ คอมพิวเตอร์

**รูปที่ 4.11** แสดงการต่อใช้งานจริง

# บทที่ 5
# บทสรุป ปัญหา แนวทางแก้ไขและการพัฒนา

## 5.1 สรุป

การสร้างชุดฝึก PLC โปรแกรมด้วยคอมพิวเตอร์ของปริญญานิพนธ์นี้จะได้ประโยชน์ในด้าน

1) ไปใช้ในการเรียน การสอนวิชาที่เกี่ยวข้องกับไมโครโปรเซสเซอร์หรือ PLC ได้

2) ไปเป็นเครื่องตัวอย่างในการประยุกต์ใช้งานหรือพัฒนา

3) สามารถพัฒนาโปรแกรมบนเครื่อง PC แล้วทำการส่งผ่านไปยังชุดฝึก PLC ซึ่งเพิ่มความสะดวกในการทำงานมากยิ่งขึ้น

การใช้งานชุดฝึกนี้เป็นการฝึกการเขียนและทำความเข้าใจโปรแกรมภาษา Ladder เพื่อให้ไมโครโปรเซสเซอร์ไปทำงานควบคุมระบบตามที่ต้องการได้อย่างง่ายและสะดวกในการทำงาน เนื่องจากมีอุปกรณ์สนับสนุนในการอำนวยความสะดวกในการเขียนโปรแกรมพอสมควร

ชุดฝึกนี้คณะผู้จัดทำตั้งใจทำขึ้นเพื่อใช้ประกอบการเรียนการสอนในการเขียนโปรแกรมภาษา Ladder หรือไมโครโปรเซสเซอร์ ตระกูล Z80 เพื่อเสริมสร้างความรู้ ความเข้าใจและความชำนาญในการเขียนโปรแกรมจากการศึกษาภาคทฤษฎีแล้ว ให้มีความเข้าใจดียิ่งขึ้น อีกทั้งยังมีใบงานประกอบการการทดลอง เพื่อให้ผู้ที่สนใจได้ฝึกหัดและเป็นแนวทางในการเขียนโปรแกรมควบคุมในงานอุตสาหกรรมต่อไปได้

## 5.2 ปัญหาและแนวทางในการแก้ปัญหา

จากการทดลองชุดฝึก PLC โปรแกรมด้วยคอมพิวเตอร์ ซึ่งได้ทดลองต่ออุปกรณ์ที่ละส่วนแล้วนำทุกส่วนมาประกอบรวมกันเป็นวงจรสมบูรณ์ ในการทดลองแต่ละส่วนนั้น สามารถสรุปปัญหา แนวทางในการแก้ปัญหา ในการทดลองเป็นข้อๆได้ดังนี้

1) ในส่วนของการทดลองของภาคแสดงผลโดยใช้ LED นั้น LED ไม่สว่างเท่าที่ควร แก้ไขโดยการสลับขั้ว LED โดยที่ขั้ว Cathode ต่อเข้ากับ Data Port และป้อน VCC เข้าขา Anode โดยใช้ R-Pack 10 KΩ เป็นตัวจำกัดกระแส

2) ราคาของ EPROM เบอร์ 27C1001 ราคาแพง เนื่องจากเป็น ROM ที่มีความจุสูง ทำให้ต้องระมัดระวังในการใช้งาน

3) ในการเขียนและพัฒนาโปรแกรมต้องใช้ EPROM Emulator แทน EPROM จริง ที่ใช้ในแผ่นวงจรพิมพ์ ทำให้ในการเคลื่อนย้ายต้องถอด EPROM Emulator ออกไปมา เป็นผลให้ Socket ของ Emulator ชำรุด (2 ครั้ง)

4) การทดลองในส่วนของอุปกรณ์แสดงผล LCD คู่มือการใช้งานที่ทางบริษัทฯ ให้มานั้นมีรายละเอียดไม่เพียงพอและยากต่อการทำความเข้าใจ และนักศึกษาในกลุ่มก็ไม่เคยมีใครได้เคยทดลองเลย ทำให้ต้องใช้เวลาในการศึกษาค้นคว้าและทดลองนาน

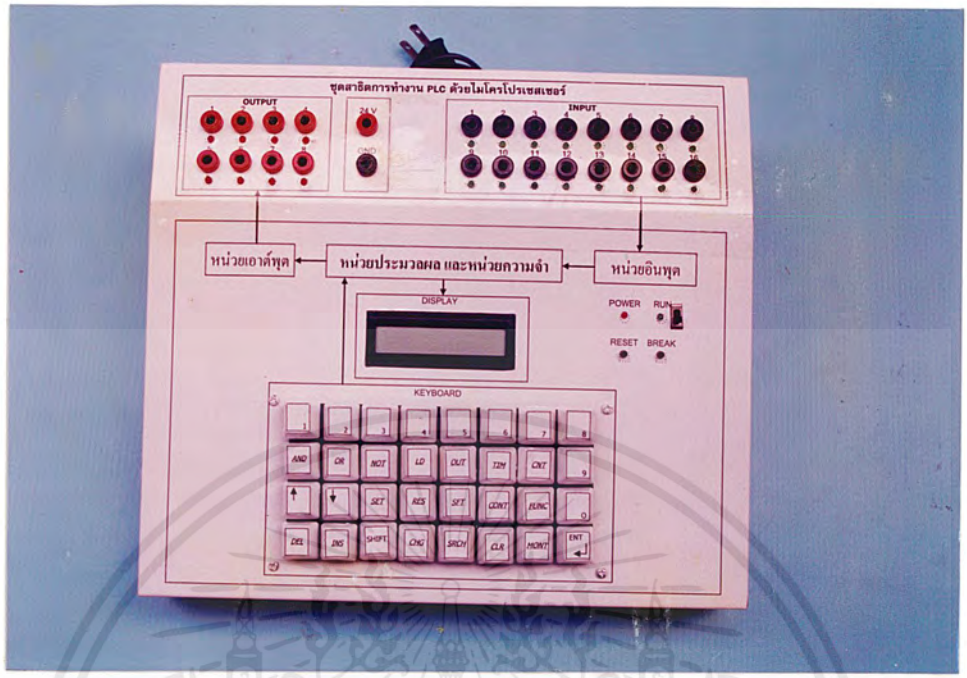
5) หลังจากทำการออกแบบลายวงจรพิมพ์เรียบร้อยแล้ว เมื่อทำการตรวจเช็คขั้นสุดท้าย ปรากฏว่า มีจุดผิดพลาดอีก 3-4 จุด ทำให้ต้องแก้ไขเป็นเวลานาน เนื่องจากไม่สามารถเดินลายวงจรใหม่ได้แต่แก้ไขโดยการใช้รู Hole 2-3 รูต่อจุดผิดพลาด 1 จุด

## 5.3 แนวทางการแก้ไขและการพัฒนาโครงงาน

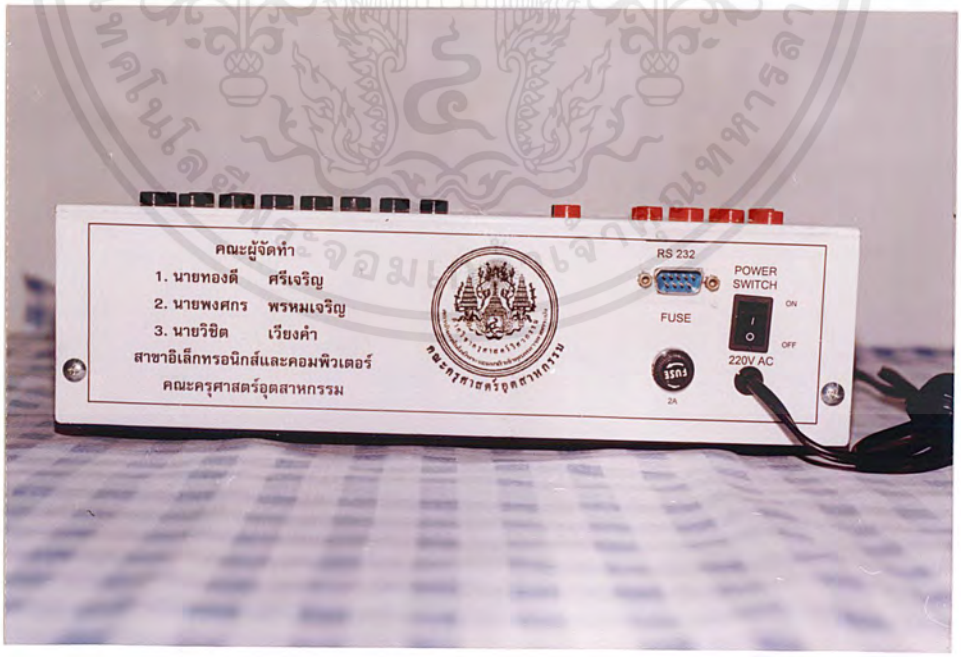
1) ชุดควบคุมของโครงงานนี้และชุด PLC แยกกันอยู่ถ้าเป็นไปได้ในการพัฒนาขั้นต่อไปควรให้อยู่ในแผ่นวงจรพิมพ์เดียวกัน

2) โครงงานนี้ยังถือว่าเป็นชุดฝึก PLC อยู่ ดังนั้นถ้าผู้พัฒนาสามารถดัดแปลง แก้ไข หรือปรับปรุงก็สามารถนำไปใช้งานจริงในอุตสาหกรรมได้

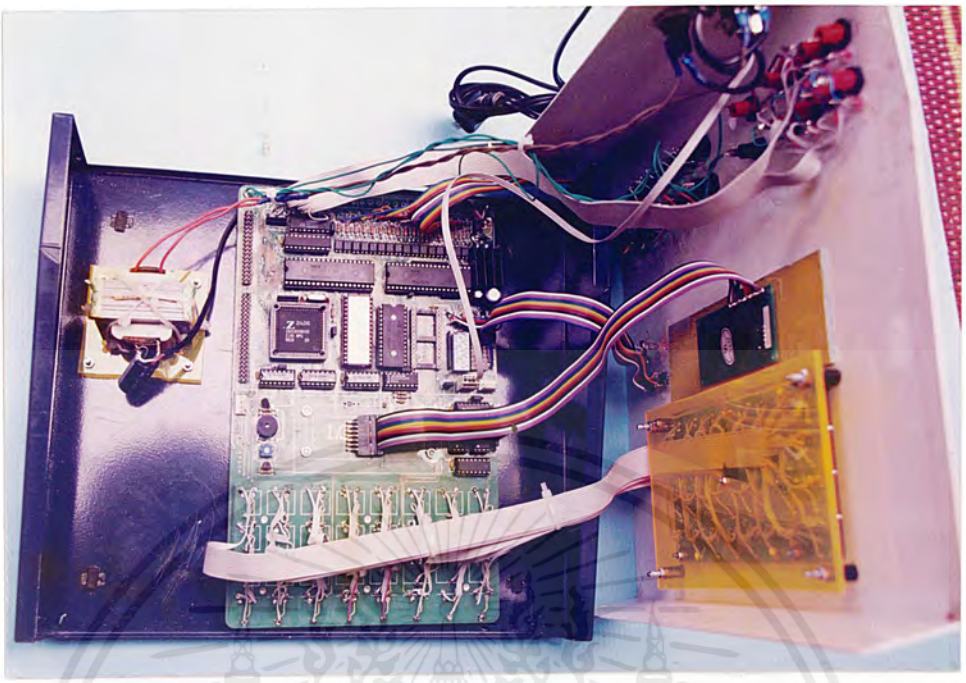
# ภาคผนวก ก
# เครื่องต้นแบบ



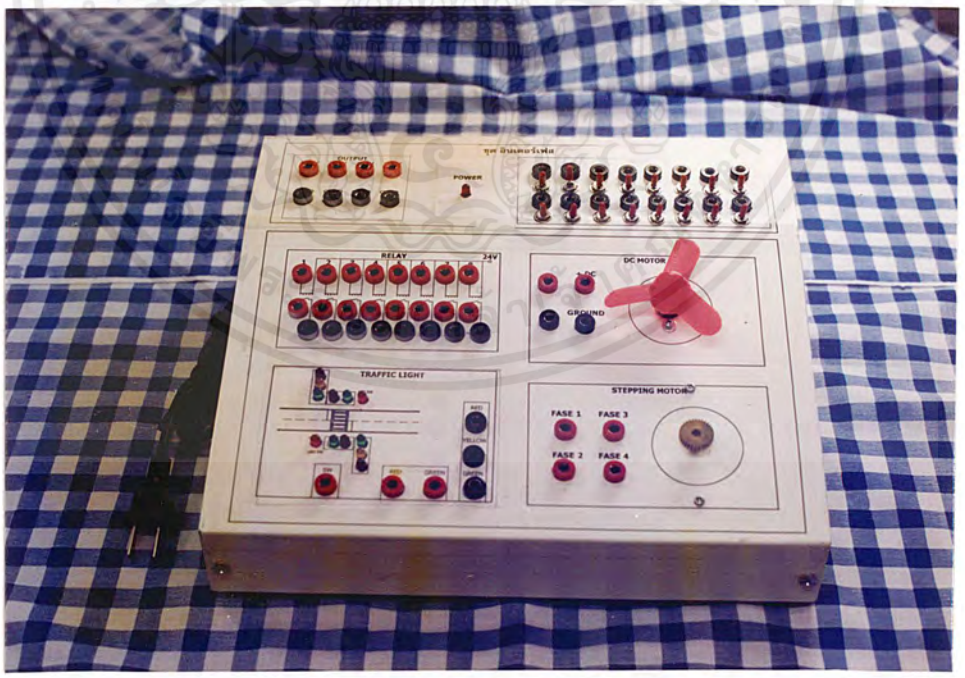

**รูปที่ ก.1** ด้านหน้าตัวเครื่องชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์



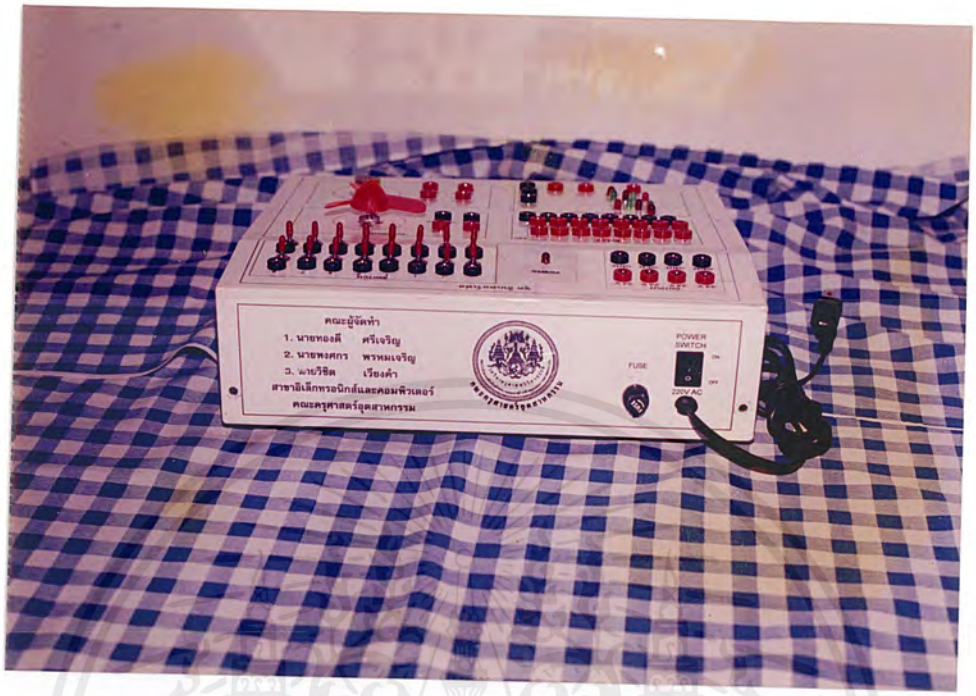

**รูปที่ ก.2** ด้านข้างตัวเครื่องชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์



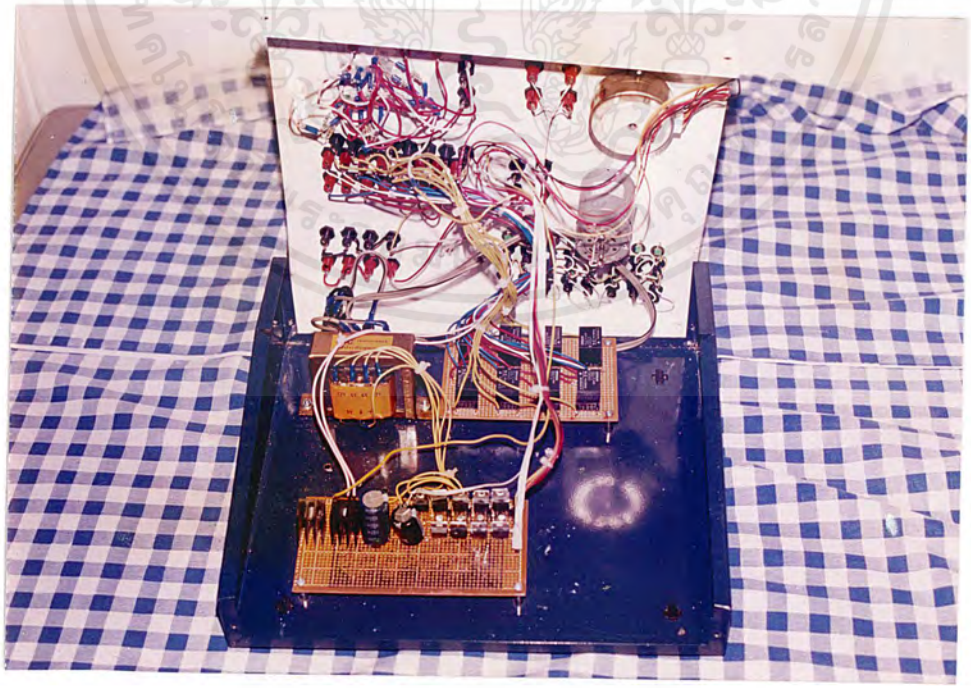
รูปที่ ก.3 ภายในตัวเครื่องชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์



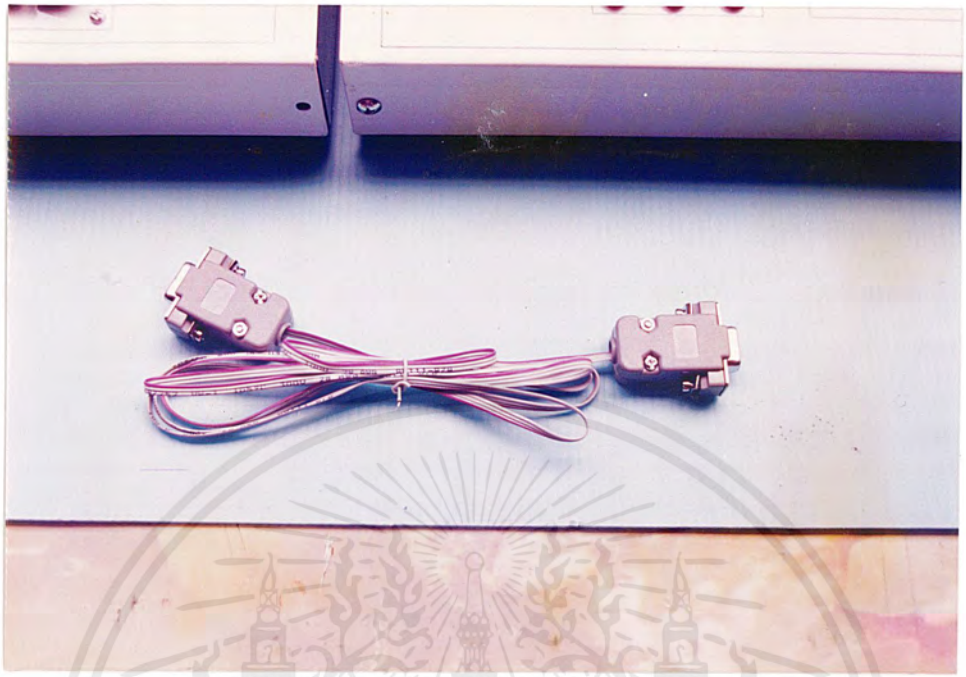

รูปที่ ก.4 ด้านหน้าตัวเครื่องชุดอินเตอร์เฟส




**รูปที่ ก.5** ด้านข้างตัวเครื่องชุดอินเตอร์เฟส

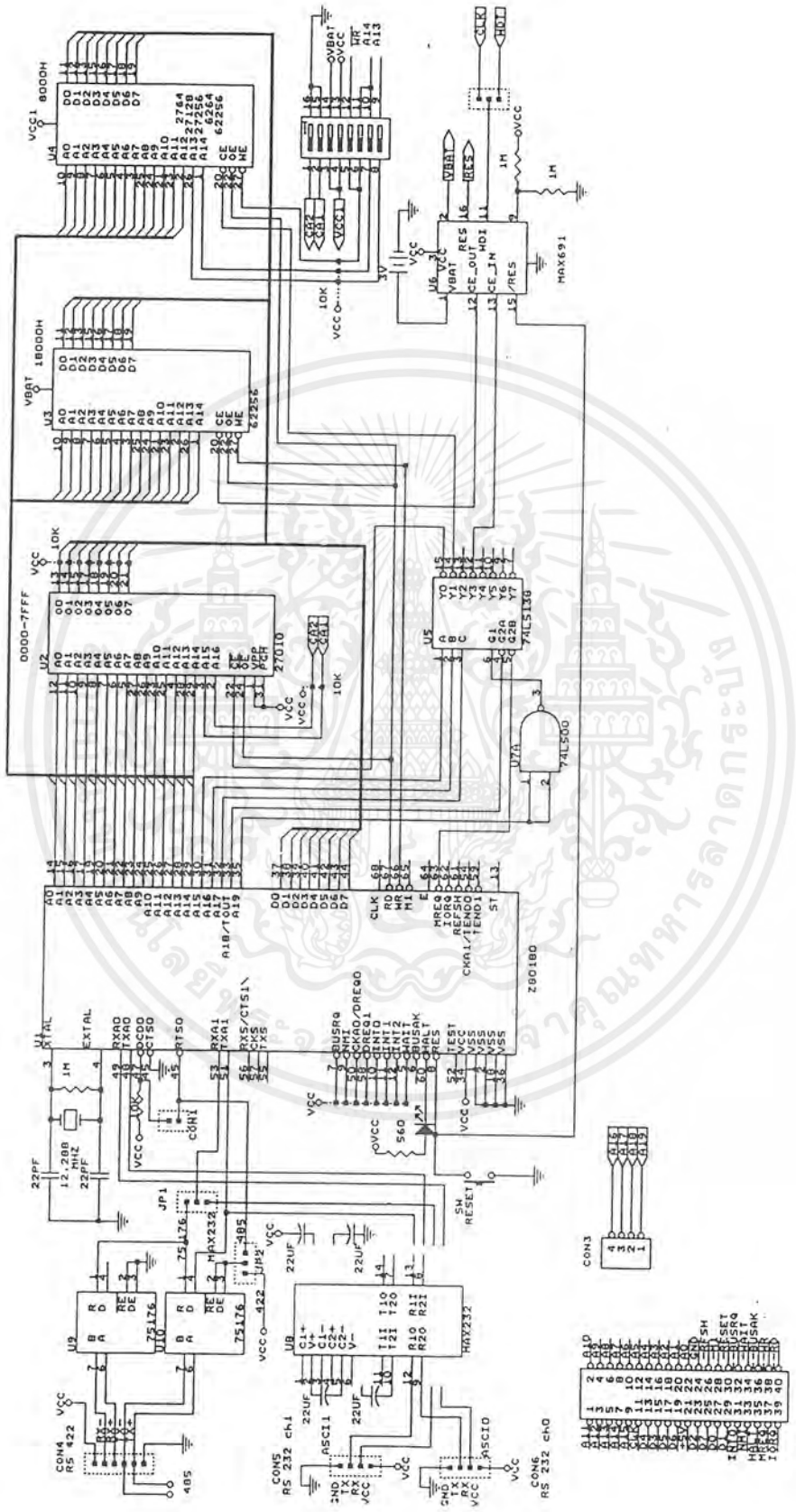


**รูปที่ ก.6** ด้านในตัวเครื่องชุดอินเตอร์เฟส

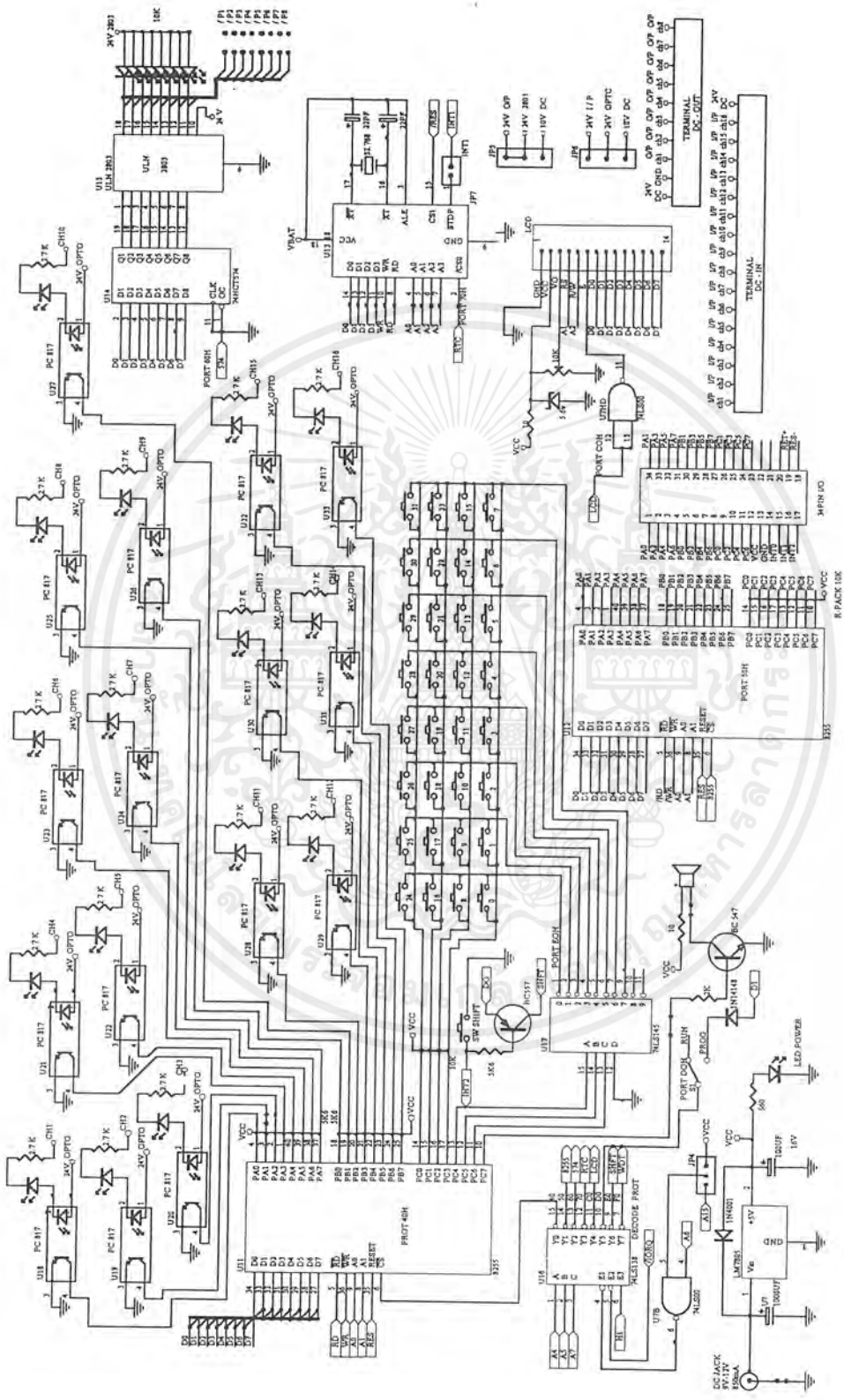


**รูปที่ ก.7** สายเชื่อมต่อชุดสาธิต PLC เข้ากับ คอมพิวเตอร์

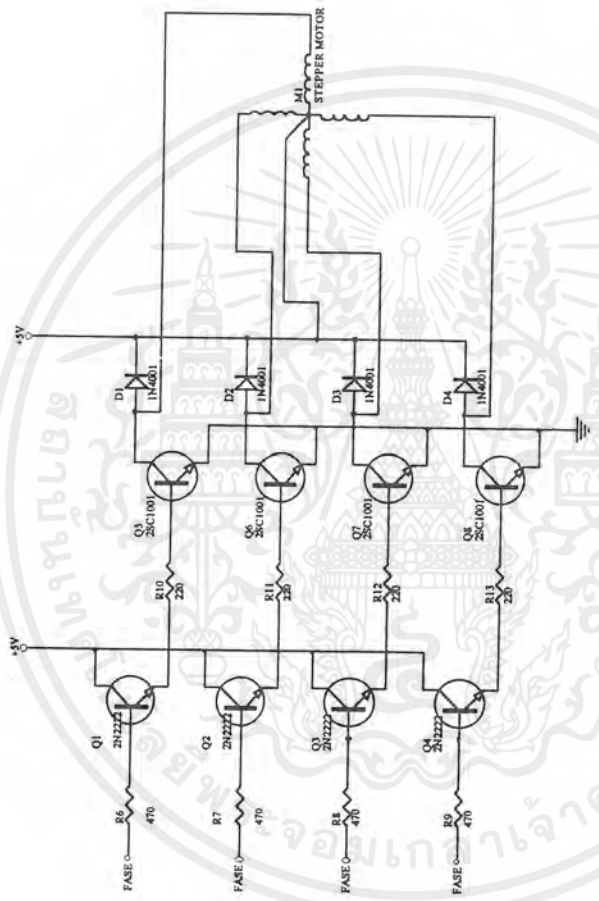
# ภาคผนวก ข
# วงจรและแผ่นวงจรพิมพ์



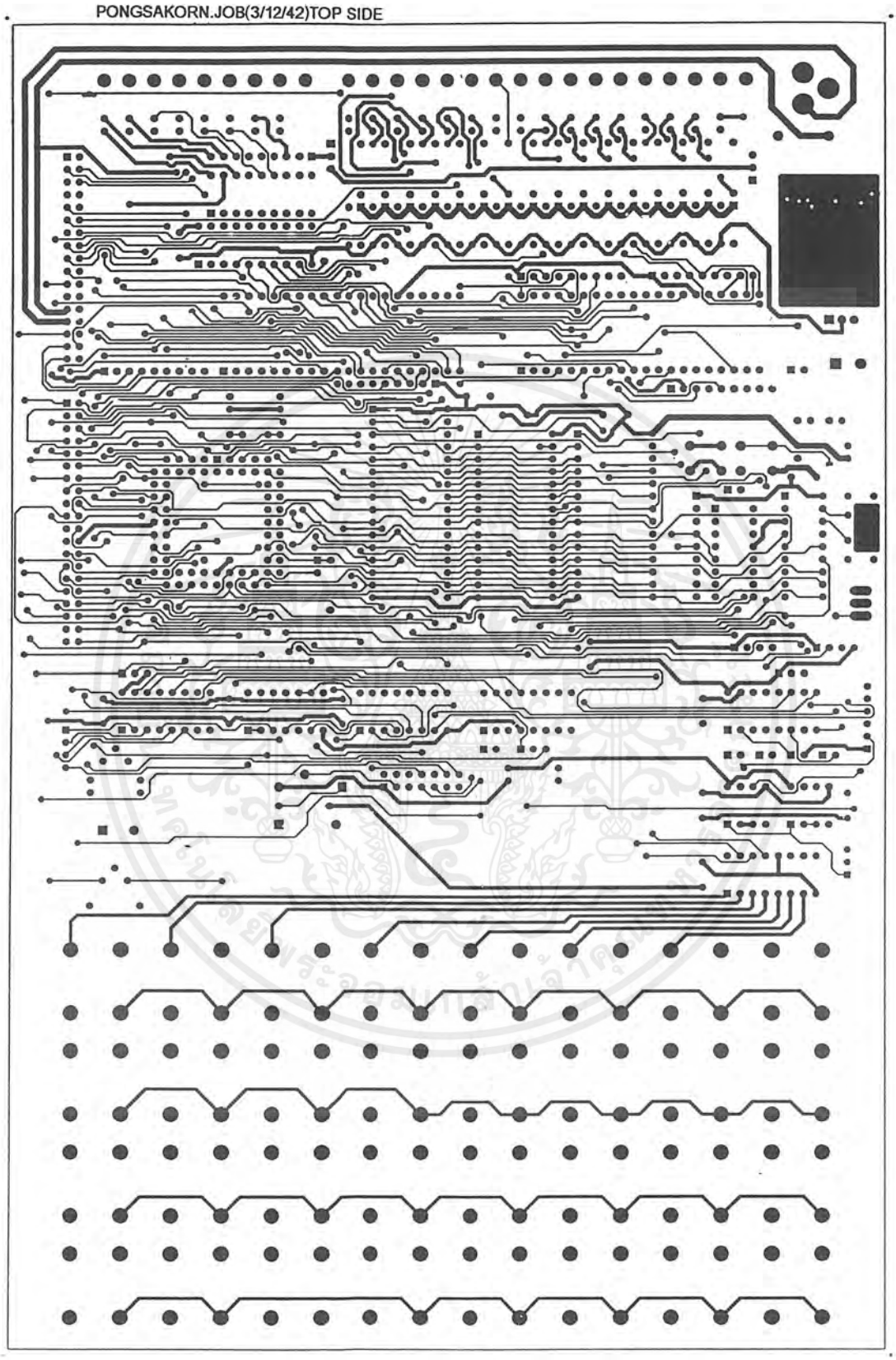
รูปที่ ข.1 วงจรชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์



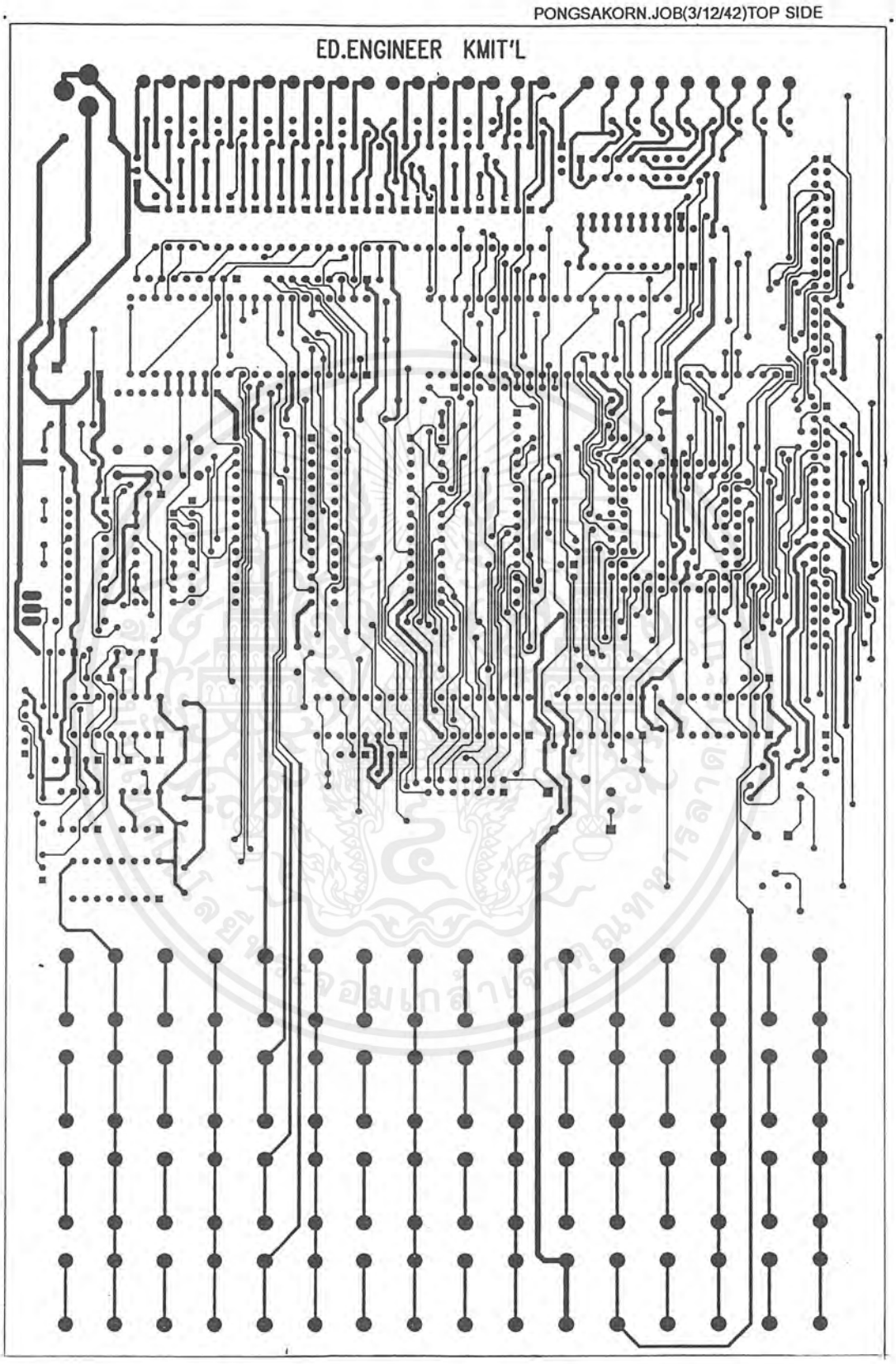
**รูปที่ ข.1** วงจรชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ (ต่อ)



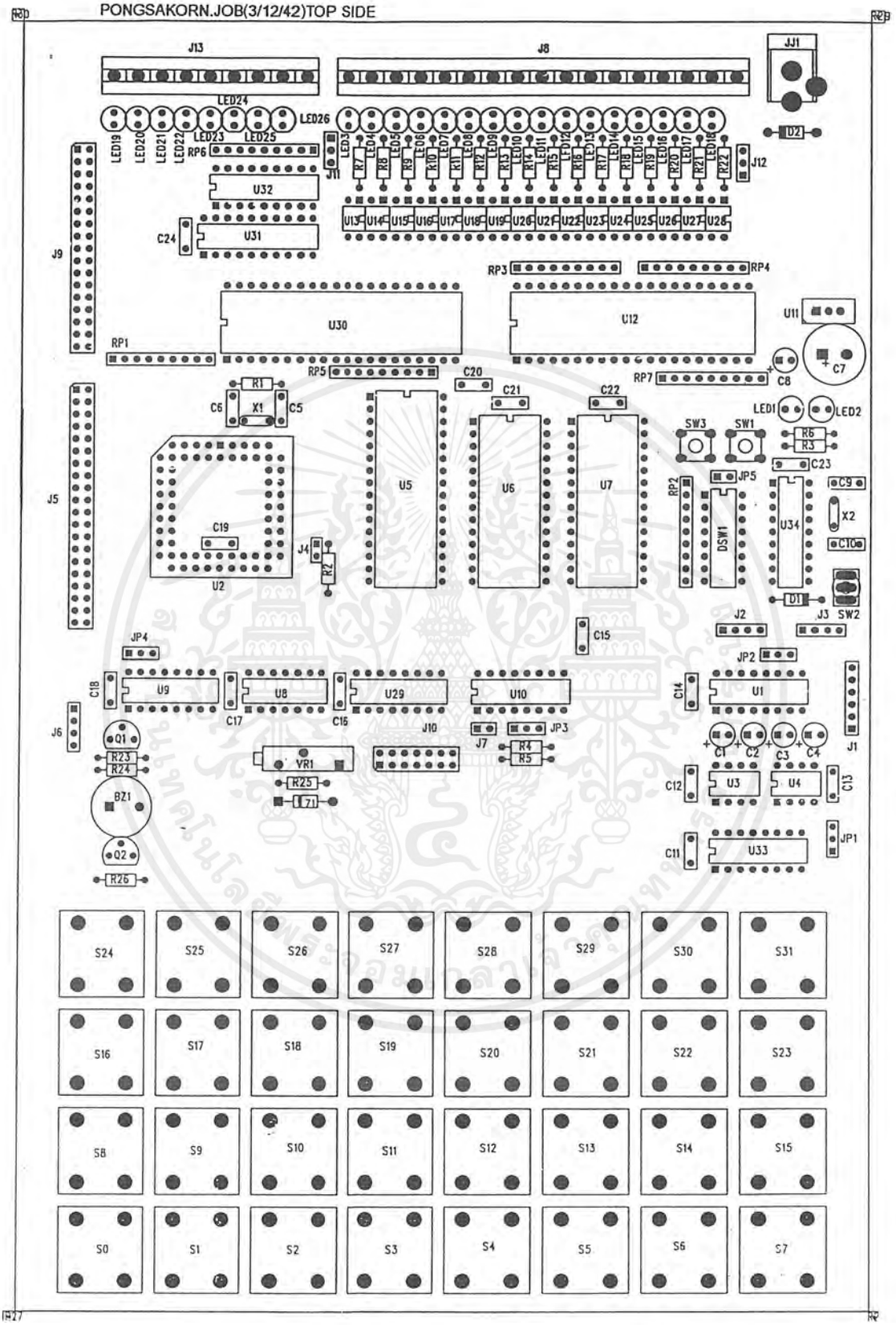

รูปที่ ข.2 วงจร สเตปปิ้งมอเตอร์



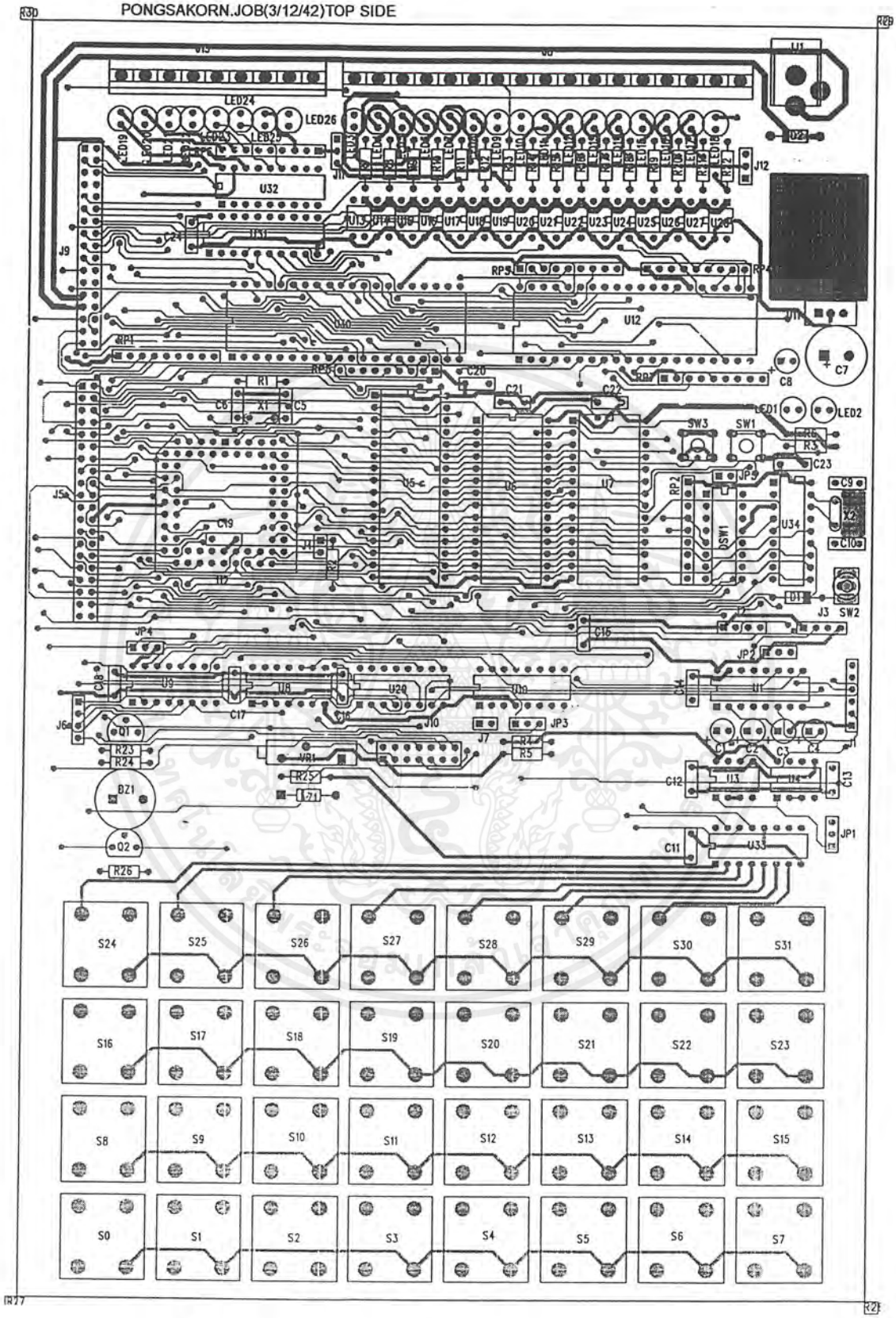

**รูปที่ ข.3** แสดงลายวงจรด้านบน ชุดสาธิตการทำงาน PLC




รูปที่ ข.4 แสดงลายวงจรด้านล่าง ชุดสาธิตการทำงาน PLC

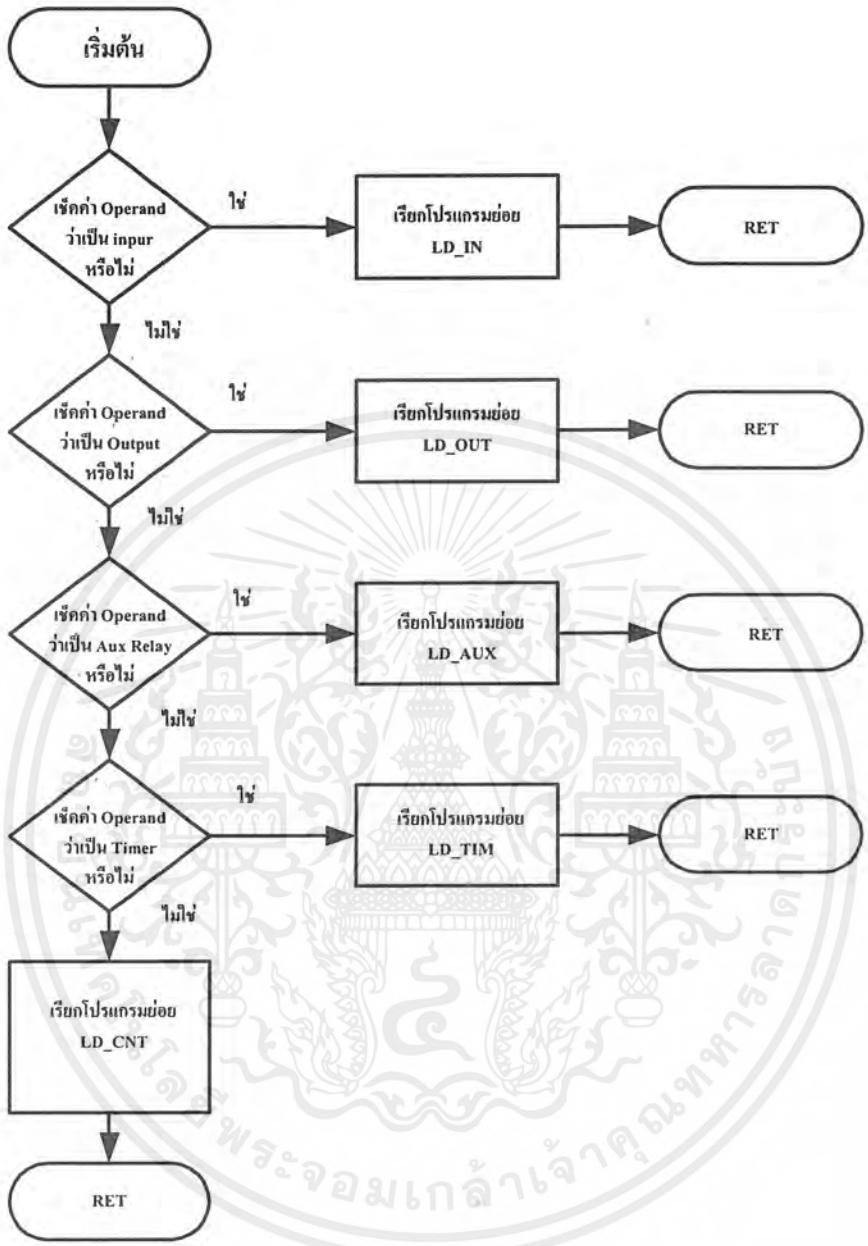



รูปที่ ข.5 แสดงการวางอุปกรณ์ด้านบนชุดสาธิตการทำงาน PLC

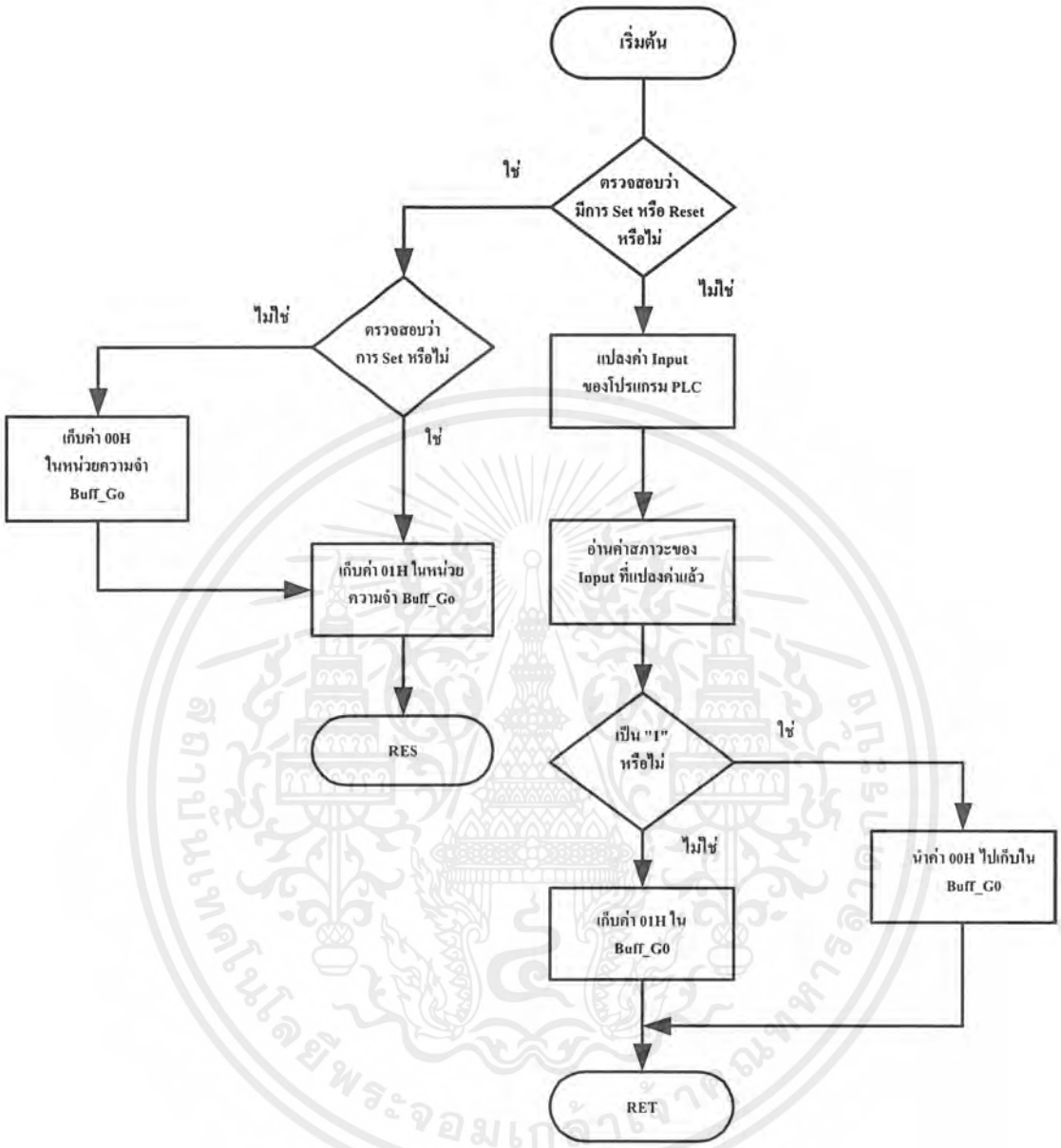



รูปที่ ข.6 แสดงการวางอุปกรณ์ด้านล่างชุดสาธิตการทำงาน PLC

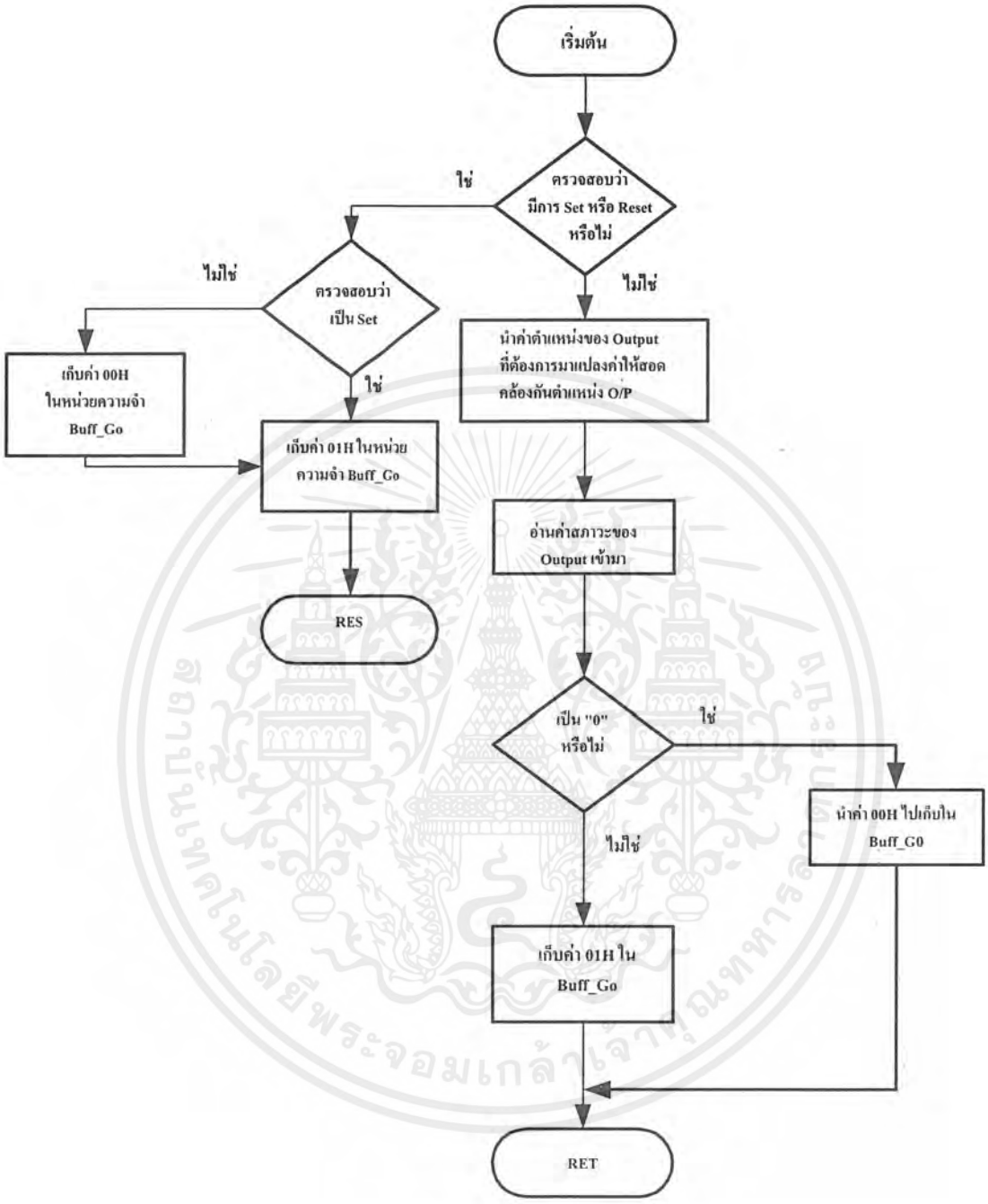
# ภาคผนวก ค
# ผังงานและโปรแกรมการทำงาน



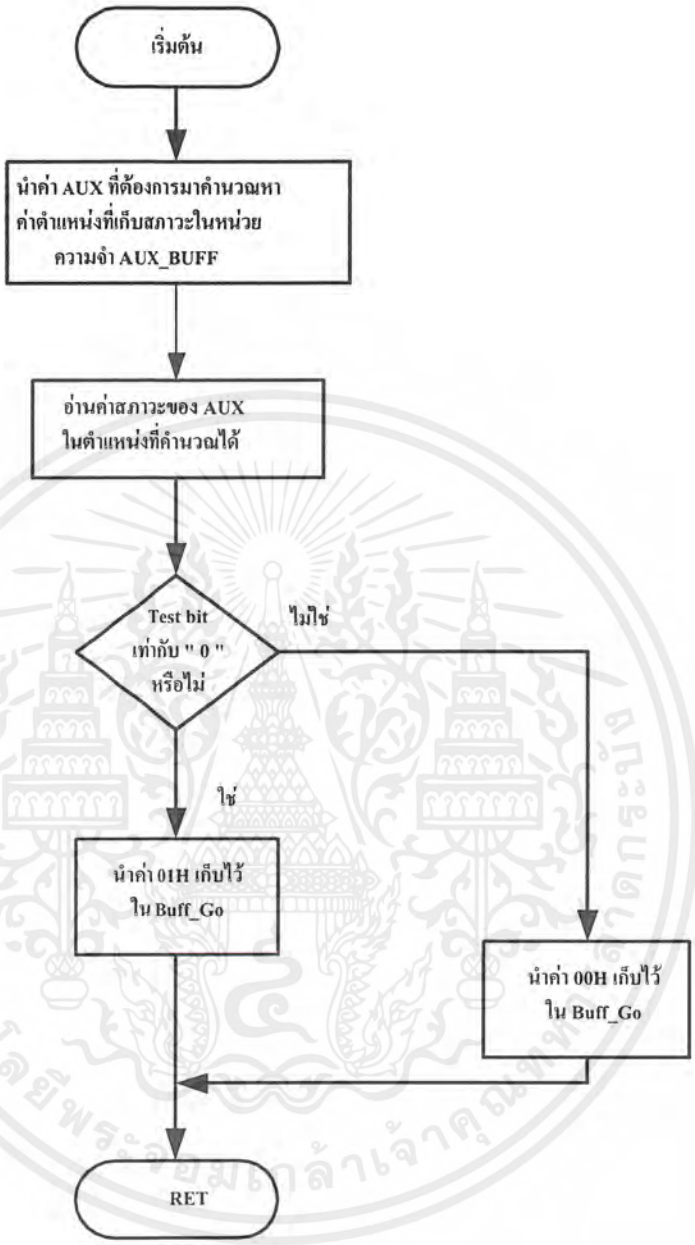

รูปที่ ค.1 ผังงานโปรแกรมหลัก




รูปที่ ค.2 ผังงานโปรแกรมย่อย LD-IN

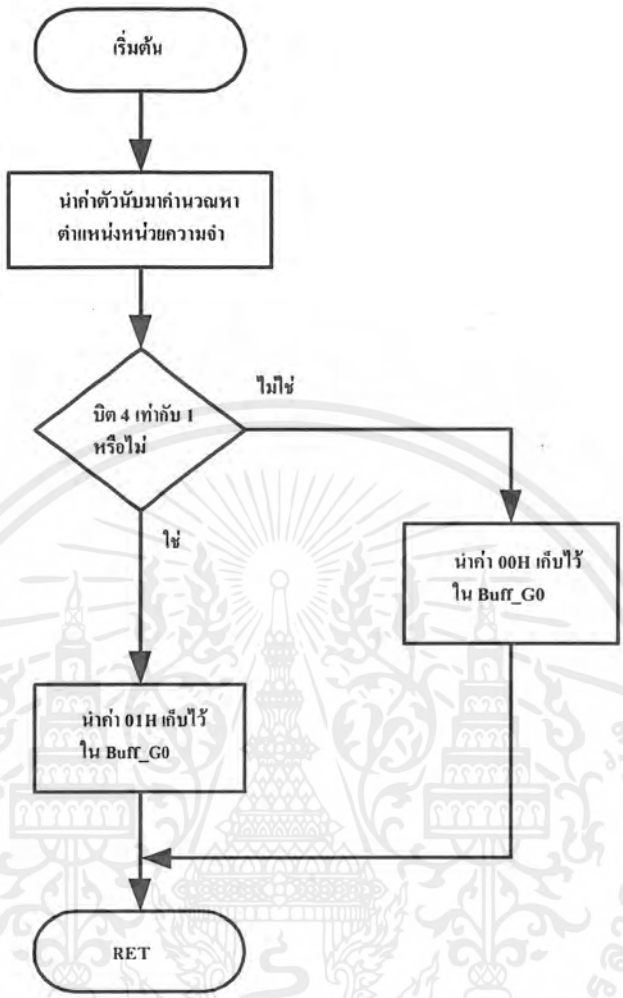



**รูปที่ ค.3** ผังงานโปรแกรมย่อย LD-OUT



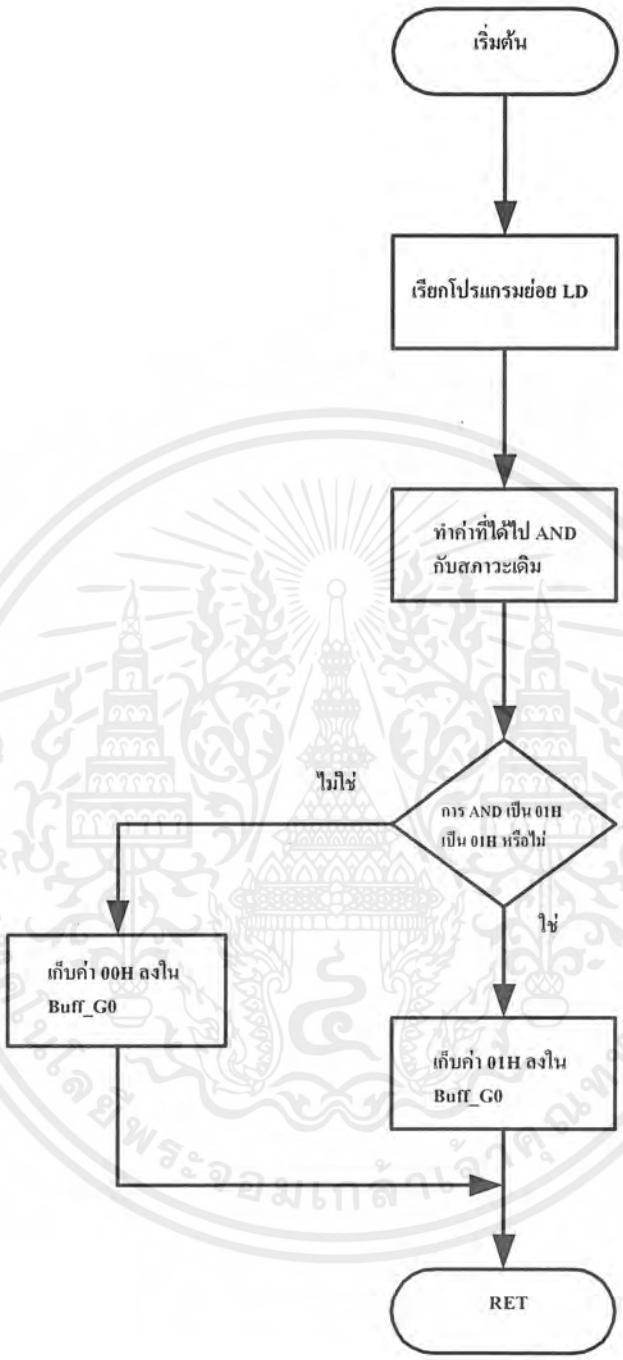

รูปที่ ค.4 ผังงานโปรแกรมย่อย LD-AUX


รูปที่ ค.5 ผังงานโปรแกรมย่อย LD-TIM

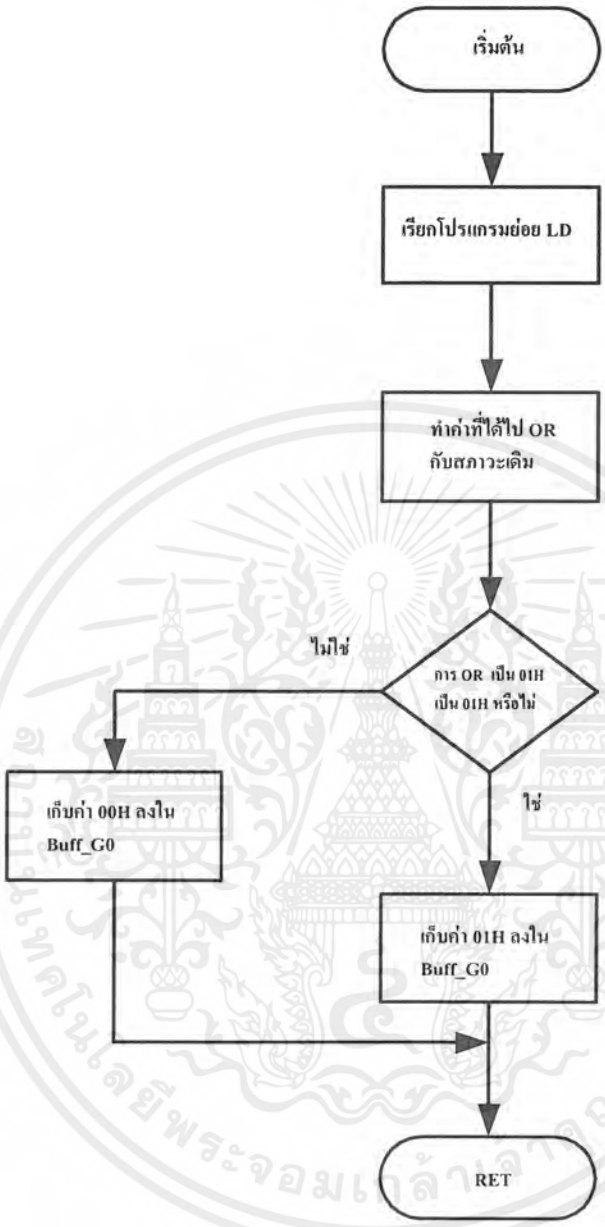



**รูปที่ ค.6** ผังงานโปรแกรมย่อย LD-CNT


**รูปที่ ค.7** ผังงานโปรแกรมย่อย LD-NOT




**รูปที่ ค.8** ผังงานโปรแกรมย่อย AND




**รูปที่ ค.9** ผังงานโปรแกรมย่อย OR


**รูปที่ ค.10** ผังงานโปรแกรมย่อย SET


รูปที่ ค.11 ผังงานโปรแกรมย่อย RES


**รูปที่ ค.12** ผังงานโปรแกรมย่อย CNT


**รูปที่ ค.13** ผังงานโปรแกรมย่อย TIMER

```
        CPU     "Z180.TBL"
        HOF     "INT8"
;*********************************************************
;CNTLA0 :       EQU     00H     ; ASCII CONTROL REG A       CH 0
CNTLA1 :        EQU     01H     ; ASCII CONTROL REG A       CH 1
CNTLB0 :        EQU     02H     ; ASCII CONTROL REG B       CH 0
CNTLB1 :        EQU     03H     ; ASCII CONTROL REG B       CH 1
STAT0 : EQU     04H     ; ASCII STATUS REG    CH 0
STAT1 : EQU     05H     ; ASCII STATUS REG    CH 1
TDR0 :          EQU     06H     ; ASCII XMIT DATA REG       CH 0
TDR1 :          EQU     07H     ; ASCII SMIT DATA REG       CH 1
RDR0 :          EQU     08H     ; ASCII RECV DATA REG       CH 0
RDR1 :          EQU     09H     ; ASCII RECV DATA REG       CH 1

CNTR :          EQU     0AH     ;CSI / O CONTROL REGISTER
TRDR :          EQU     0BH     ;CSI /O XMIT / RECV DATA R EG

TMDR0L :        EQU     0CH     ;TIMER DATA REG CH 0L
TMDR0H :        EQU     0DH     ;TIMER DATA REG CH 0H
RLDR0L :        EQU     0EH     ;RELOAD REGISTER CH 0L
RLDR0H :        EQU     0FH     ;RELOAD REGISTER CH 0H
TCR :           EQU     10H     ;TIMER CONTROL REGISTER

TMDR1L :        EQU     14H     ;TIMER    DATA REG    CH 1L
TMDR1H :        EQU     15H     ;TIMER    DATA REG    CH 1H
RLDR1L :        EQU     16H     ;RELOAD   REGISTER    CH 1L
RLDR1H :        EQU     17H     ;RELOAD REGISTER      CH 1H

SAR0L:          EQU     20H     ;DMA SOURCE ADDRESS REG CH 0L
SAR0H:          EQU     21H     ;DMA SOURCE ADDRESS REG CH 0H
SAR0B:          EQU     22H     ;DMA SOURCE ADDRESS REG CH 0B
DAR0L:          EQU     23H     ;DMA DSTNTN  ADDRESS REG CH 0L
DAR0H:          EQU     24H     ;DMA DSTNTN  ADDRESS REG CH 0H
DAR0B:          EQU     25H     ;DMA DSTNTN  ADDRESS REG CH 0B
```

```
BCR0L :     EQU   26H    ;DMA BYTE COUNT REGISTER CH 0L
BCR0H ;     EQU   27H    ;DMA BYTE COUNT REGISTER CH 0H
MAR1L :     EQU   28H    ;DMA MEMORY ADDRESS REG CH 1L
MAR1H :     EQU   29H    ;DMA MEMORY ADDRESS REG CH 1H
MAR1B       EQU   2AH    ;DMA MEMORY ADDRESS REG CH 1B
IAR1L :     EQU   2BH    ;DMA DSTNTN I/O          REG CH 1L
IAR1H :     EQU   2CH    ;DMA DSTNTN I/O          REG CH 1H
BCR1L :     EQU   2EH    ;DMA BYTE COUNT REGISTER CH 1L
BCR1H :     EQU   2FH    ;DMA BYTE CONUT REGISTER CH 1H
DSTAT :     EQU   30H    ;DMA  STATUS REGISTER
DMODE :     EQU   31H    ;DMA MODE REGISTER
DCNTL :     EQU   32H    ;DMA/ WAIT CONTROL REGISTER

IL :        EQU   33H    ;IL REGISTER (INT VECTOR REG)
ITC :       EQU   34H    ;INT/ TRAP CONTROL REGISTER

RCR :       EQU   36H    ;REFRESH CONTROL REGISTER

;       NO DISPLAY
;             (KEYIN) = 0FFH        ;key not active
;       O/P = A (KEYIN)
;       REG = A
SCANK :     CALL  SCANN
            CP    0FFH
            JR    Z, SCANK1
            CALL  K_NEW
            JR    C , SCANK
            CALL  K_CODE
SCANK1 ;    RET

;*********************************
;       SCANK KEY CODE   *
;*********************************
; ADDRESS = 0103H
```

```
; Scan key 1 cycle KEY code = code
; NO display
;         (KEYIN) = 0FFH ; key not active
; I/P    = -
; O/P    = A, (KEYIN)
; REG    = A

SCANC :     CALL    SCANN
            CP      0FFH
            JR      NZ , SCANC1
            IN      A , (P_SHIFT)
            BIT     0 , A
            JR      NZ , SCANC
;
SCANC0 :    IN      A , (P_SHIFT)
            BIT     0 , A
            JR      Z , SCANC0
            LD      A , 2
            JR      SCANC2

SCANC1 :    CALL    K_NEWO
            JR      C , SCANC
SCANC2 :    LD      (KEYIN) , A
            RET

;*********************************
;         DISPLAY LCDBUF        *
;*********************************
; ADDRESS = 0104H
; I/P    = LCDBUF
; O/P = HL (LCDBUF ADDRESS AT (DATA = 00))
; REG    = HL

DSP_BUF :       PUSH    AF
```

```
                PUSH    BC
                PUSH    DE

                CALL CLR_LCD
                LD      LH , LCD_BUF
;
                LD      B , 16
DSP_B0 :        LD A , (HL)
                CP      0
                JR      Z , DSP_B2
                LD      D , (HL)
                CALL    WRLCD
                INC     HL
                DJNZ    DSP_B0
;
                LD      D , 0C0H
                CALL    LCDCMD
                LD      B , 16
DSP_B1 :        LD      A , (HL)
                CP      0
                JR      Z , DSP_B2
                LD      D , (HL)
                CALL    WRLCD
                INC     HL
                DJNZ DSP_B1
;
DSP_B2 :        POP     DE
                POP     BC
                POP     AF
                RET

;*********************************
;       CLEAR LCD BUF       *
;*********************************
```

```
; ADDRESS = 0105H
; REG = -

CLEAR :PUSH   AF
              PUSH   BC
              PUSH   HL

              LD     HL , LCD_BUF
              LD     B , 32
CLR_L1 :      LD     (HL) , 0
              INC    HL
              DJNZ   CLR_L1
              CALL   DSP_BUF

              POP    HL
              POP    BC
              POP    AF
              RET
;
;**********************************
;         GET HEX WORD          *
;**********************************
; ADDRESS = 0106H
; I / P     = HL
; O / P     = DE , CF
; REG       = AFDEHL

GETHEXW :     CALL GET_WORD
              CALL OVR_HEX
              RET
;
;**********************************
;         GET HEX BYTE *
;**********************************
```

```
; ADDRESS = 0107H
; I/P   = HL
; O/P   = E, CF
; REG   = AFDEHL

GETHEXB :       CALL GET_BYTE
                CALL OVR_HEX
                RET
;
;**********************************
;           BLAK SKIP           *
;**********************************
; ADDRESS = 0108H
; I/P   = HL
; O/P   = HL
; REG   = AHL
;
BLKSKP :        LD      A,(HL)
                CP      " "
                RET     NZ
                INC     HL
                JR      BLKSKP
;
;**********************************
;       CHECK COMMAND           *
;**********************************
; ADDRESS       = 0109H
; I/P =     HL (TABLE COMMAND)
;           DE (USER COMMAND)
;           IX (TABLE FOR JUMP SUB.)
;           B (BYTE COUNT OF COMMAND)
;           C (NUMBER COMMAND IN TABLE)
; O/P = IX (TABLE OF SUB.), CF
; REG   = AFBCHLIX
```

```
CHK_COMM :
                PUSH    BC
CHK_CM0 :       CALL    CK_CHAR
                JR      Z , CHK_CM1
                DEC     C
                JR      NZ , CHK_CM0
                SCF
                POP     BC
                RET
;
; / * Command found * /

CHK_CM1 :       LD      A , C
                POP     BC
                PUSH    DE

                PUSH    DE
                POP     HL
                LD      E , B
                LD      D , 0
                ADD     HL , DE
                LD      B , C
                LD      C , A
                CALL    CHK_END
                POP     DE
                RET
;
;**********************************
;           INITIAL LCD             *
;**********************************
; ADDRESS = 010AH
; REG = -
```

```
INIT_LCD :
                PUSH    DE
                LD      D , 38H         ;Reset code
                CALL    LCDCMD          ;Out command
                LD      D , 0DH         Display on & blink
                CALL    LCDCMD
                LD      D , 6           ;Cursor move - >
                CALL    LCDCMD
                CALL    CLR_LCD         ;HOME
                POP     DE
                RET
;
;*********************************
;   OUT LCD COMMAND         *
;*********************************
; ADDRESS = 010BH
; I / P   = D
; O / P   = -
; REG   = -

LCDCMD :        PUSH    AF              ;Save reg
                CALL  READY             ;Wait for LCD
                LD      A , D
                OUT     (CMD_POTW) , A
                CALL  READY
                POP AF
                RET
;
;*********************************
;       WAIT FOR LCD            *
;*********************************
; ADDRESS = 010CH
; REG = -
```

```
READY :     PUSH   AF                  ;Save reg
RDY1 :      IN     A , (CMD_POTR)      ;Check for wait
            BIT    7 , A
            JR     NZ , RDY1           ;Singflag = negate loop
            POP    AF
            RET
;
;**********************************
;   GOTO LCD ADDRESS        *
;**********************************
; ADDRESS = 010DH
; I / P     = D
; O / P     = -
; REG       = D

GOTO_ADR
            PUSH   AF
            SET    7 , D
            CALL   LCDCMD
            POP    AF
            RET
;**********************************
;      WRITE LCD SUB. *
;**********************************
; ADDRESS = 010EH
; / * Write character to LCD one byte
; I / P   = D (Ascii)
; O / P   = -
; REG     = -

WRLCD :     PUSH   AF
            PUSH   DE
            CALL   READY            ;Wait
            LD     A , D
```

```
                OUT     (DAT_POTW), A;Write character to LCD
                CALL    READY
WRLCD2:         POP     DE
                POP     AF
                RET
;
;*********************************
;       READ LCD CHAR *
;*********************************
; ADDRESS       = 010FH
; I/P   = -
; O/P   = D (character)
; REG   = D

RDLCD: PUSH     AF
                CALL    READY
                IN      A, (DAT_POTR)
                LD      D, A
                CALL    READY
                POP     AF
                RET
;*********************************
;       BACKSPACE LCD *
;*********************************
; ADDRESS = 0110H
; REG = -

BS_LCD:         PUSH    DE
                CALL    LEFT_LCD
                LD      D, " "
                CALL    WRLCD
                CALL    LEFT_LCD
                POP     DE
                RET
```

```
;
;**********************************
;       LEFT LCD SUB.   *
;**********************************
; ADDRESS = 0111H
; REG = -
LEFT_LCD :
                PUSH    DE
                LD      D , 10H  ;Cursor move left
                CALL    LCDCMD
                POP     DE
                RET
;
;**********************************
;       RIGHT LCD SUB  *
;**********************************
; ADDRESS = 0112H
; REG = -

RIGHT_LCD :
                PUSH    DE
                LD      D , 14H
                CALL    LCDCMD
                POP     DE
                RET
;**********************************
;       READ CURSOR    *
;**********************************
; ADDRESS = 0113H
; I / P      = -
; O / P      = A (address of cursor 0 - 15)
; REG        = AF
;
```

```
RDCUR :     IN      A , (CMD_POTR)
            BIT     7 , A
            JR      NZ , RDCUR
            RET

;**********************************
;    WRITE HEX TO LCD        *
;**********************************
; ADDRESS = 0114H
; I / P  = D (HEX)
; REG = -

LCDHEX :    PUSH    DE
            PUSH    AF
            LD      A , D
            CALL    HEXASC
            CALL    WRLCD
            LD      D , E
            CALL    WRLCD
            POP     AF
            POP     DE
            RET
;~~~~~~~~~~~~~~~~~~~~~~~~~~
; WRITE HEX IN DE TO LCD  *
;**************************************
; ADDRESS = 0115H
; I / P = DE (HEX)
; REG = -

DELCD :     PUSH    AF
            PUSH    DE
            CALL    LCDHEX
            LD      D , E
            CALL    LCDHEX
```

```
                POP     DE
                POP     AF
                RET
;
;**********************************
;     WRITE BLOCK LCD          *
;**********************************
; ADDRESS = 0116H
; I/P = HL table of text
; REG = HL

WR_LCD_MSG :
                PUSH    AF
                PUSH    DE
WR_MSG0 :       LD      A , (HL)
                OR      A
                JR      Z , WR_MSG1
                LD      D , A
                CALL    WRLCD
                INC     HL
                JR      WR_MSG0
WR_MSG1 :       POP     DE
                POP     AF
                RET
;
;**********************************
;          CLEAR LCD    *
;**********************************
; ADDRESS = 0117H
; REG = -

CLR_LCD :
                PUSH    DE
                LD      D , 1
```

```
                CALL    LCDCMD
                POP     DE
                RET
;
;*********************************
;       CURSOR ON     *
;*********************************
; ADDRESS = 0118H
; REG = -

CUR_ON :        PUSH    DE
                LD      D , 00001101B
                CALL    LCDCMD
                POP     DE
                RET
;*********************************
;       CURSOR OFF    *
;*********************************
; ADDRESS = 0119H
; REG = -

CUR_OFF :
                PUSH    DE
                LD      D , 00001100B
                CALL    LCDCMD
                POP     DE
                RET
;
;*********************************
;      WRITE CG RAM  *
;*********************************
; ADDRESS = 011AH
; I / P         = HL
;               = D (0 - 7)
```

```
; REG       = AHL

WR_CGRAM :
                PUSH    BC
                PUSH    DE
                CALL    RDCUR
                PUSH    AF
                SLA     D
                SLA     D
                SLA     D
                SET     6 , D           ;ADDR CGRAM
                CALL    LCDCMD
;
                LD      B , 8
WR_CG1 :        LD      D , (HL)
                CALL    WRLCD
                INC     HL
                DJNZ    WR CG1

                POP     AF
                LD      D , A
                CALL    GOTO_ADR
                POP     DE
                POP     BC
                RET
;
;*********************************
;   ASCII BYTE TO HEX*
;*********************************
; ADDRESS = 011BH
; I / P   = E (ASCII)
; O / P = A (HEX) , CF ( E - 42H -> A = 0BH )
; REG  = AF
ASC1_HEX :
```

```
                LD      A , E
                CALL    HEX
                RET
;
;**********************************
;   ASCII WORD TO HEX          *
;**********************************
; ADDRESS = 011CH
; I / P   = DE (ASCII)
; O / P = A (HEX) DE = 31H, 32H A =12H
; REG  = AFD
ASC2_HEX :
                LD      A, D
                CALL    HEX
                RET     C
                RLCA
                RLCA
                RLCA
                RLCA
                LD      D , A
                CALL    ASC1_HEX
                OR      D
                RET
;
;**********************************
;       HEX TO ASCII            *
;**********************************
; ADDRESS = 011DH
; I / P   = C       (HEX)
; O / P = DE      (ASCII)
; REG   = DE
;
HTOA:           PUSH    AF
                LD      A , C
```

```
                CALL    HEXASC
                POP     AF
                RET
;
;**********************************
;   CHECK ASCII UPPER*
;**********************************
; ADDRESS = 011EH
; I/P  = D
; O/P = A (big character)
; REG  = AF
;
UPPER :         LD      A , D
                CALL    UP
                RET
;
;**********************************
;   CHECK ASCII LOWER       *
;**********************************
; ADDRESS = 011FH
; I/P  = D
; O/P = A
; REG  = AF
LOWER ;         LD      A , D
                CP      "A"
                RET     C
                CP      "Z" + 1
                RET     NC
                ADD     A , 32          ;YES BIG
                RET
;
;**********************************
;         HEX -> DEC    *
;**********************************
```

```
; ADDRESS = 0120H
; I/P  = DE
; O/P = ABC
; REG  = AFBC
HEXDEC :        LD      (CBUF) , DE
                CALL    HTOD
                LD      BC, (CBUF+2)
                LD      A, (CBUF+4)
                RET
;
;**********************************
;         DEC -> HEX              *
;**********************************
; ADDRESS = 0121H
; I/P  = ABC
; O/P = DE
; REG  = AFDE
DECHEX :        PUSH    BC
                LD      (CBUF+2) , BC
                LD      (CBUF+4) , A
                CALL    DTOH
                LD      DE , (CBUF)
                POP     BC
                RET
;
;**********************************
;         TABLE BYTE    *
;**********************************
; ADDRESS = 0122H
; I/P  = B NO. (0 - FF)
;           HL start table
; O/P = A data from table
; REG  = AF
TABLEB :        PUSH    HL
```

```
                PUSH    BC
                LD      C , B
                LD      B , 0
                ADD     HL , BC
                LD      A , (HL)
                POP     BC
                POP     HL
                RET
;
;**********************************
;          TABLE SUB              *
;**********************************
; ADDRESS = 0123H
; Look - up  2  Byte  table
; I / P   =  B  NO.  (0 - FF)
;            HL  start  table
; O / P  =  HL  DATA    from  table
; REG   =  HL
;
TABLEW :        PUSH    AF
                PUSH    BC
                PUSH    DE
                LD      A , B
                ADD     A , A
                LD      B , 0
                LD      C , A
                ADD     HL , BC
                LD      E , (HL)
                INC     HL
                LD      D , (HL)
                EX      DE , HL
                POP     DE
                POP     BE
                POP     AF
```

```
                RET
;
;**********************************
;       TABLE SUB           *
;**********************************
; ADDRESS = 0124H
; Look - up TABLE (any length)
; I/P  = B 0 - 255
;         C TABLE LENGTH
;         HL = START TABLE
; O/P = HL FIRST DATA IN TABLE
; REG = BHL
;
TABLES :        PUSH    AF
                PUSH    DE
                PUSH    HL
                INC     B
                LD      D , 0
                LD      E , C
                LD      HL , 0
TABLES1 :       DEC     B
                JR      Z , TABLES2     ;FOUND
                ADD     HL , DE
                JR      TABLES1
TABLES2 :       EX      DE , HL
                POP     HL
                ADD     HL , DE
                POP     DE
                POP     AF
                RET
;;
;**********************************
;       CHECK SUM   *
;**********************************
```

```
; ADDRESS = 0125H
; I/P  = HL start address
;          BC length
; O/P = A , D
; REG  = AFBCDHL
;
CHKSUM :        LD      D , 0
CHKSUM1 :       LD      A , D
                XOR     (HL)
                LD      D , A
                INC     HL
                DEC     BC
                LD      A , B
                OR      C
                JR      NZ , CHKSUM1
                LD      A , D
                CPL
                LD      D , A
                RET
;
;*********************************
;    DISPLAY LCDBUF  *
;*********************************
; ADDRESS = 0126H
; I/P  = DE
; REG  = DE
PRINT :         PUSH AF
                PUSH   BC
                PUSH HL
                LD      B , 32
                LD      HL , LCD_BUF
PRINT0 :        LD      A , (DE)
                LD      (HL) , A
                CP      0
```

```
                JR      Z , PRINT1
                INC     HL
                INC     DE
                DJNZ    PRINT0
;
PRINT1 :        CALL DSP_BUF
                POP     HL
                POP     BC
                POP     AF
                RET
;
;**************************
;   SWAP REG. A         *
;**************************
; ADDRESS       = 0127H
; I / P         = A
; O / P         = A
; REG           = A
SWAPA :         PUSH    BC
                PUSH    AF
                RLCA
                RLCA
                RLCA
                RLCA
                LD      C , A
                POP     AF
                LD      A , C
                POP     BC
                RET
;**************************
;   SAVE ALL REG        *
;**************************
; ADDRESS       = 0128H
; REG           = -
```

```
PUSHR :     LD      (B_CALL) , HL
            POP     HL
            LD      (B_CALL+2) , HL
            PUSH    AF
            PUSH    BC
            PUSH    DE
            PUSH    IX
            PUSH    IY
            LD      HL , (B_CALL)
            PUSH    HL
            LD      HL , (B_CALL+2)
            EX      (SP) , HL
            RET
;
;*****************************
;  RESTORE  ALL  REG. *
;*****************************
; ADDRESS     = 0129H
; REG     = -
POPR :      POP     HL
            LD      (B_CALL+2) , HL
            POP     HL
            POP     IY
            POP     IX
            POP     HL
            POP     DE
            POP     BC
            POP     AF
            PUSH    HL
            LD      HL , (B_CALL+2)
            EX      (SP) , HL
            RET
;
;*************************
```

```
;WATCH DOG SUB       *
;*************************
; ADDRESS       = 012AH
; REG     = -
WDT :           OUT     P_WDT), A
                RET
;
;*************************
;  COMPARE 16 BIT     *
;*************************
; ADDRESS       = 012BH
; COMPARE IX - BC
; I / P       = IX , BC
; O / P       = FLAG
; REG     = -
;
COMP16 :        PUSH    HL
                PUSH    IX
                POP     HL
                OR      A
                SBC     HL , BC
                POP     HL
                RET
;*************************
;    SORTING            *
;*************************
; ADDRESS       = 012CH
; I / P       = IY ADDR AT SORTING
;               C NUMBER SORTING
; O / P       = (IY)
; REG     = -
;
SORT :          PUSH    IX
                PUSH    HL
```

```
                PUSH    DE
                PUSH    BC
                PUSH    AF
SORTAGI :       PUSH    IY          ;store address sorting
                POP     IX          ;Get addr sort to ix
                LD      A , 0       ;A = 0 NO CHANGE
                LD      B , C       ;Number sorting
                DEC     B
SORTNEX :       LD      L , (IX+0)
                LD      H , (IX+1)
                LD      E , (IX+2)
                LD      D , (IX+3)
                OR      A           ;Clear carry
                SBC     HL , DE
                JR      C , SORTNO
                JR      Z , SORTNO
;
SORTCHG :       LD      A , 1       ;SET A=CHANGE
                LD      L , (IX+0)  ;NO carry change
                LD      H , (IX+1)
                LD      (IX+0) , E
                LD      (IX+1) , D
                LD      (IX+2) , L
                LD      (IX+3) , H
SORTNO :        INC     IX
                INC     IX
                DJNZ    SORTNEX
                CP      1
                JR      Z , SORTAGI
                POP     AF
                POP     BC
                POP     DE
                POP     HL
                POP     IX
```

```
                RET
;
;**************************
;  DELAY 10 USEC      *
;**************************
; ADDRESS      = 012DH
; I/P      = B
; REG      = B
DUSEC :         PUSH    HL              ;11
                POP     HL              ;9
                PUSH    HL
                POP     HL
                NOP                     ;12
                NOP
                NOP
                NOP
                DJNZ    DUSEC           ;9
                RET
;**************************
;  DELAY 1 MSEC       *
;**************************
; ADDRESS      = 012EH
; I/P      = B
; REG      = B
DMSEC :         PUSH HL
                PUSH    DE
                PUSH    AF
DMS0 :          LD      HL , 99         ;9
DMS1 :          LD      DE , 0          ;9
                PUSH    IX              ;14
                POP     IX              ;12
                NOP                     ;6
                NOP
                DEC     HL              ;4
```

```
                LD      A , H           ;4
                OR      L               ;4
                JR      NZ , DMS1       ;8
;
                PUSH    IX              ;14
                POP     IX              ;12
                LD      DE , 0          ;9
                LD      A , H           ;4
                DJNZ    DMS0            ;13
                POP     AF
                POP     DE
                POP     HL
                RET
;
;**************************
;     DELAY 1 SEC       *
;**************************
; ADDRESS     = 012FH
; I / P       = B
; REG         = B
DSEC :          PUSH    HL
                PUSH    AF
DSEC1 :         LD      HL , 35712      ;9
DSEC2 :         PUSH    IX              ;14
                LD      IX , (0000H)    ;126
                LD      IX , (0000H)
                LD      IX , (0000H)
                LD      IX , (0000H)
                LD      IX , (0000H)
                LD      IX , (0000H)
                LD      IX , (0000H)
                POP     IX              ;12
                DEC     HL              ;4
                LD      A , H           ;4
```

```
                OR      L               ;4
                JR      NZ , DSEC2      ;8
                LD      A , 1           ;24
                LD      A , 1
                LD      A , 1
                LD      A , 1
                DJNZ DSEC1              ;9
                POP     AF
                POP     HL
                RET

;*****************************----------(LOCAL USE)
; HEX TO DEC SUB.      *
;*****************************
; IN        = (CBUF)
; OUT       = (CBUF+2)
; REG       = ABC
HTOD            PUSH    HL
                XOR     A
                LD      HL , CBUF+5
                LD      B , 3
HTOD1 :         DEC     HL
                LD      (HL) , A
                DJNZ    HTOD1
                LD      C , 16
HTOD2 :         LD      HL , CBUF
                RL      (HL)
                INC     HL
                RL      (HL)
                INC     HL
                LD      B , 3
HTOD3 :         LD      A , (HL)
                ADC     A , A
                DAA
```

```
                LD      (HL) , A
                INC     HL
                DJNZ    HTOD3
                DEC     C
                JR      NZ , HTOD2
                POP     HL
                RET
;*************************----------------------(LOCAL USE)
;  DEC TO HEX SUB. *
;*************************
; IN       = (CBUF+2)
; OUT      = (CBUF)
; REG      = ABC
DTOH :          PUSH    HL
                LD      C , 16
DTOH1 :         LD      B , 3
                XOR     A
                LD      HL , CBUF+4
DTOH2 :         LD      A , (HL)
                RRA
                PUSH    AF
                BIT     7 ,A
                JR      Z , DTOH3
                SUB     30H
DTOH3 :         BIT     3 , A
                JR      Z , DTOH4
                SUB     3
DTOH4 :         LD      (HL) , A
                DEC     HL
                POP     AF
                DJNZ    DTOH2
                RR      (HL)
                DEC     HL
                RR      (HL)
```

```
                DEC     C
                JR      NZ , DTOH1
;
                POP     HL
                RET
;
;*****************************----------------------(LOCAL USE)
; CHECK ASCII UPPER*
;*****************************
; input     A
; output    A (big charecter)
; reg       a
;
UP :            CP      "A"
                RET     C
                CP      "Z"+1
                RET     NC
                SUB     "a" - "A";yes small
; /*-----CALCULATOR 32 BIT SUB. ----- */
; /* HCAL GROUP PROGRAM */
;-------------------------------------------------------
; MON3 0CL.ASM (INCLUDE-FILE)
; STACK CALCULATION SUBROUTINE
; SOFTWARE ENGINEER :KRIANGSAK B.
J UPDATE 11/06/88
; CONCEPT OF STACK CALCULATION . . .
;
; -----| X |-----  PUSH    POP  eg.      LD      HL , 2A30H
;    | Y |     (ENTER)    (X)           CALL    CLE16     X=2A30H
;    | Z |       |                ^     LD      HL , 4F21H
;    | T |       |                |     CALL    CLE16     Y=2A30H X=4F21H
;    |------|     v                |     CALL    CLAD      X=X+Y
;                                        CALL    CLX16     HL      =7951H
;    |------|
```

```
;   |MEM|    32 BIT DATA eg. | 01 | 45 | F1 | 2A | =0145F12A
;   |-------|
; PLEASE SET WORKING AREA ON YOUR MAIN PROGRAM
; CLSTK DS 16 4 LEVEL STACK
; CLBUF DS 8  BUFFER FOR CALCULATION
; CLMEM DS 4 MEMORY DATA
; CLEXX DS 28 EXCHANGE AREA (FOR INTERRUPT PROCESS)
;                INPUT.OUTPUT        ZF  CF    ACTION
; CLE            -       -           -   -     PUSH X
;CLE8            D       -           -   -     PUSH   X & X=D
; CLE16          HL      -           -   -     PUSH   X & X=HL
; CLE32          (IX+0)  -           -   -     PUSH   X & X=(IX+0) -> (IX+3)
; CLX            -       -           -   -     POP    X
; CLX8           -       D           -   -     POP    X & D=X
; CLX16          -       HL          -   -     POP    X & HL=X
; CLX32          -       (IX+0)      -   -     POP    X & (IX+0) -> (IX+3) =X

; CLMP           -       -           Y   Y     X=X*Y
; CLDV           -       -           Y   Y     X=X/Y CLBUF=REMAINDER
; CLAD           -       -           Y   Y     X=X+Y
; CLSB           -       -           Y   Y     X=X-Y
; CLHD           -       -           Y   Y     CHANGE X FROM HEX TO DEC
; CLDH           -       -           -   -     CHANGE X FROM DEC TO HEX
; CLSW           -       -           -   -     X<->Y
; CLEX           -       -           -   -     INTERRUPT PROCESS
; CLMR           -       -           -   -     PUSH X & X=MEM
; CLMC           -       -           -   -     CLEAR MEM
; CLMAD          -       -           -   -     MEM=MEM+X
; CLMSB          -       -           -   -     MEM=MEM-X
; CLDAD          -       -           Y   Y     X=X+Y (BCD)
; CLDSB          -       -           Y   Y     X=X-Y (BCD)
; CLTAD          -       -           Y   Y     X=X+Y TIME (HHHH,MM,SS)
; CLTSB          -       -           Y   Y     X=X-Y TIME
; ********** CLE SUB **********
```

```
; ADDRESS = 0130H
; REG = BCDEHL
CLE :           LD      HL , CLSTK+11
                LD      DE , CLSTK+15
                LD      BC , 12
                LDDR
                RET
; ********** CLE8 SUB **********
; ADDRESS = 0131H
; IN  = D
; REG = ABCDEHL
CLE8 :          PUSH    DE
                CALL    CLE
                CALL    CLCLR
                POP     DE
                LD      A, D
                LD      (CLSTK+3) , A
                RET
; ********** CLE16 SUB **********
; ADDRESS = 0132H
; IN  = HL
; REG = ABCDEHL
CLE16 :         PUSH    HL
                CALL    CLE
                CALL    CLCLR
                POP     HL
                LD      A , H
                LD      H , L
                LD      L , A
                LD      (CLSTK+2) , HL
                RET
; ********** CLX SUB **********
; ADDRESS = 0133H
; REG = BCDEHL
```

```
CLX :       LD      HL , CLSTK+4
            LD      DE , CLSTK
            LD      BC , 12
            LDIR
            LD      HL , CLSTK+15
            XOR     A
            LD      B , 4
CLX1 :      LD      (HL) , A
            DEC     HL
            DJNZ    CLX1
            RET
; ********** CLX8 SUB **********
; ADDRESS = 0134H
; OUT = D
; REG = ABCDHL
CLX8        LD      A , (CLSTK+3)
            LD      D , A
            PUSH    DE
            CALL    CLX
            POP     DE
            RET
; ********** CLE32 SUB **********
; ADDRESS = 0135H
; IN  = (IX+0)
; REG = BCDEHL
CLE32 :     CALL    CLE
            PUSH    IX
            POP     HL
            LD      DE , CLSTK
            LD      BC , 4
            LDIR
            RET
; ********** CLX16 SUB **********
; ADDRESS = 0136H
```

```
OUT = HL
; REG = ABCDEHL
CLX16 :         LD      HL , (CLSTK+2)
                LD      A , H
                LD      H, L
                LD      L , A
                PUSH    HL
                CALL    CLX
                POP     HL
                RET
; ********** CLX32 SUB **********
; ADDRESS = 0137H
; OUT = (IX+0)
; REG = ABCDEHL
CLX32 :         LD      HL , CLSTK
                PUSH    IX
                POP     DE
                LD      BC , 4
                LDIR
                CALL    CLX
                RET
; ********** CLMP SUB **********
; ADDRESS = 0138H
; REG = ABCDEHL
CLMP :          CALL CLCLB
                LD      B , 32
CLMP1 :         PUSH    BC
                CALL    CLMPS
                CALL    CLMPR
                JR      NC , CLMP2
                CALL    CLMPA
CLMP2 :         POP     BC
                DJNZ    CLMP1
                CALL    CLEND
```

```
                RET
CLMPS :         XOR     A               ;SHIFT RESULT (CLBUF)
                LD      HL , CLBUF+7
                LD      B , 8
CLMPS1 :        RL      (HL)
                DEC     HL
                DJNZ    CLMPS1
                RET
CLMPR :         XOR     A               ;ROTATE MULTIPLICAND (CLSTK+4)
                LD      HL , CLSTK+7
                LD      B , 4
CLMPR1 :        RL      (HL)
                DEC     HL
                DJNZ    CLMPR1
                RET     NC
                LD      HL , CLSTK+7
                SET     0 , (HL)
                RET
CLMPA :         XOR     A               ;ADD CLSTK TO CLBUF
                LD      HL , CLSTK+3
                LD      DE , CLBUF+7
                LD      B , 4
CLMPA1 :        LD      A , (DE)
                ADC     A , (HL)
                LD      (DE) , A
                DEC     HL
                DEC     DE
                DJNZ    CLMPAI
                RET     NC
                LD      BC , 400H
CLMPA2 :        LD      A , (DE)
                ADC     A , C
                LD      (DE) , A
                DEC     DE
```

```
                DJNZ    CLMPA2
                RET
; ********** CLDV SUB **********
; ADDRESS = 0139H
; REG = ABCDEHL
CLDV :          CALL    CLCLB
                LD      C , 32
CLDV1 :         CALL    CLDVX
                CALL    CLDVZ
                CALL    CLDVS
                JR      NC , CLDV2
                CALL    CLDVA
CLDV2 :         CCF
                CALL    CLDVZ4
                DEC     C
                JR      NZ , CLDV1
                CALL    CLEND
                RET
CLDVX :         XOR     A               ;ROTATE CLSTK4
                LD      HL , CLSTK+7
                LD      B , 4
CLDVX1 :        RL      (HL)
                DEC     HL
                DJNZ    CLDVX1
                RET     NC
                LD      HL , CLSTK + 7
                SET     0 , (HL)
                RET
CLDVZ :         LD      HL , CLBUF+3    ;ROTATE CLBUF
                LD      B , 4
                JR      CLDVZ41
CLDVZ4 :        LD      HL , CLBUF+7    ;ROTATE CLBUF4
                LD      B , 4
CLDVZ41 :       RL      (HL)
```

```
                DEC     HL
                DJNZ    CLDVZ41
                RET
CLDVS :         CL      HL , CLSTK+3    ;CLBUF=CLBUF-CLSTK
                LD      DE , CLBUF+3
                LD      B , 4
CLDVS1 :        LD      A , (DE)
                SBC     A , (HL)
                LD      (DE) , A
                DEC     HL
                DEC     DE
                DJNZ    CLDVS1
                RET
CLDVA :         XOR     A               ;CLBUF=CLBUF+CLSTK
                LD      HL , CLSTK+3
                LD      DE , CLBUF+3
                LD      B , 4
CLDVA1 :        LD      A , (DE)
                ADC     A , (HL)
                LD      (DE) , A
                DEC     HL
                DEC     DE
                DJNZ    CLDVA1
                RET
; ********** CLAD SUB **********
; ADDRESS = 013AH
; REG = ABCDEHL
CLAD:           PUSH    IX
                CALL    CLCLB
                LD      HL , CLSTK+7
                LD      DE , CLSTK+3
                LD      IX , CLBUF+7
                LD      B , 4
CLAD1 :         LD      A , (DE)
```

```
                ADC     A , (HL)
                LD      (IX+0) , A
                DEC     HL
                DEC     DE
                DEC     IX
                DJNZ    CLAD1
                LD      A , 0
                ADC     A , A
                LD      (IX+0) , A
                POP     IX
                ALL     CLEND
                RET
; ********** CLSB SUB **********
; ADDRESS = 013BH
; REG = ABCDEHL
CLSB :          PUSH    IX
                CALL    CLCLB
                LD      HL , CLSTK+7
                LD      DE , CLSTK+3
                LD      IX , CLBUF+7
                LD      B , 4
CLSB1 :         LD      A , (DE)
                SBC     A , (HL)
                LD      (IX+0) , A
                DEC     HL
                DEC     DE
                DEC     IX
                DJNZ    CLSB1
                LD      A , 0
                ADC     A , A
                LD      (IX+0) , A
                POP     IX
                CALL    CLEND
                RET
```

```
; ********** CLHD SUB **********
; ADDRESS = 013CH
; REG = ABCDEHL
CLHD :      CALL   CLCLB
            LD     C , 32
CLHD1 :     LD     HL , CLSTK+3
            LD     B , 4
CLHD2 :     RL     (HL)
            DEC    HL
            DJNZ   CLHD2
            LD     HL , CLBUF+7
            LD     B , 8
CLHD3 :     LD     A , (HL)
            ADC    A , A
            DAA
            LD     (HL) , A
            DEC    HL
            DJNZ   CLHD3
            DEC    C
            JR     NZ , CLHD1
            CALL   CLBTX
            CALL   CLFAG
            RET
; ********** CLDH SUB **********
; ADDRESS = 013DH
; REG = ABCDEHL
CLDH :      LD     C , 32
CLDH1 :     LD     B , 4
            XOR    A
            LD     HL , CLSTK
CLDH2 :     LD     A , (HL)
            RRA
            PUSH   AF
            BIT    7 , A
```

```
                JR      Z , CLDH3
                SUB     30H
CLDH3 :         BIT     3 , A
                JR      Z , CLDH4
                SUB     3
CLDH4 :         LD      (HL) , A
                INC     HL
                POP     AF
                DJNZ    CLDH2
                LD      HL , CLBUF+4
                LD      B , 4
CLDH5 :         RR      (HL)
                INC     HL
                DJNZ    CLDH5
                DEC     C
                JR      NZ , CLDH1
                CALL    CLBTX
                CALL    CLFAG
                RET
; ********** CLSW SUB **********
; ADDRESS = 013EH
; REG = BCDEHL
CLSW :          LD      HL , CLSTK+4
                LD      DE , CLBUF+4
                LD      BC , 4
                LDIR
                LD      HL , CLSTK
                LD      DE , CLSTK+4
                LD      BC , 4
                LDIR
                CALL    CLBTX
                RET
; ********** CLEX SUB **********
; ADDRESS = 013FH
```

```
; REG = ABCDEHL
CLEX :          LD      HL , CLSTX
                LD      DE , CLEXX
                LD      B , 28
CLEX1 :         LD      C , (HL)
                LD      A , (DE)
                LD      (HL) , A
                LD      A , C
                LD      (DE) , A
                INC     HL
                INC     DE
                DJNZ    CLEX1
                RET
; ********** CLMR SUB **********
; ADDRESS = 0140H
; REG = BCDEHL
CLMR :          CALL    CLE
                LD      HL , CLMEM
                LD      DE , CLSTK
                LD      BC , 4
                LDIR
                RET
; ********** CLMC SUB **********
; ADDRESS = 0141H
; REG = ABHL
CLMC :          LD      HL , CLMEM
                LD      B , 4
                XOR     A
CLMC1 :         LD      (HL) , A
                INC     HL
                DJNZ    CLMC1
                RET
; ********** CLMAD SUB **********
; ADDRESS = 0142H
```

```
; REG = ABCDEHL
CLMAD :     LD      HL , CLMEM+3
            LD      DE , CLSTK+3
            LD      B , 4
            XOR     A
CLMAD1 :    LD      A , (DE)
            ADC     A , (HL)
            LD      (HL) , A
            DEC     HL
            DEC     DE
            DJNZ    CLMAD1
            RET
; ********** CLMSB SUB **********
; ADDRESS = 0143H
; REG = ABCDEHL
CLMSB :     LD      HL , CLSTK+3
            LD      DE , CLMEM+3
            LD      B , 4
            XOR     A
CLMSB1 :    LD      A , (DE)
            SBC     A , (HL)
            LD      (DE) , A
            DEC     HL
            DEC     DE
            DJNZ    CLMSB1
            RET
; ********** CLDAD SUB **********
; ADDRESS = 0144H
; REG = ABCDEHL
CLDAD :     PUSH    IX
            CALL    CLCLB
            LD      HL , CLSTK+7
            LD      DE , CLSTK+3
            LD      IX , CLBUF+7
```

```
                LD      B , 4
CLDAD1 :        LD      A , (DE)
                ADC     A , (HL)
                DAA
                LD      (IX+0) , A
                DEC     HL
                DEC     DE
                DEC     IX
                DJNZ    CLDAD1
                LD      A , 0
                ADC     A , A
                LD      (IX+0) , A
                POP     IX
                CALL    CLEND
                RET
; ********** CLDSB SUB **********
; ADDRESS = 0145H
; REG = ABCDEHL
 CLDSB :        PUSH    IX
                CALL    CLCLB
                LD      HL , CLSTK+7
                LD      DE , CLSTK+3
                LD      IX , CLBUF+7
                LD      B , 4
CLDSB1 :        LD      A , (DE)
                SBC     A , (HL)
                DAA
                LD      (IX+0) , A
                DEC     HL
                DEC     DE
                DEC     IX
                DJNZ    CLDSB1
                LD      A , 0
                ADC     A , A
```

```
                LD      (IX+0) , A
                POP     IX
                CALL    CLEND
                RET
; ********** CLTAD SUB **********
; ADDRESS = 0146H
; REG = ABCDEHL
CLTAD :         PUSH    IX
                CALL    CLCLB
                LD      HL , CLSTK+7
                LD      DE , CLSTK+3
                LD      IX , CLBUF+7
                LD      B , 2
CLTAD1 :        CALL    CLTADS
                LD      (IX+0) , A
                DEC     HL
                DEC     DE
                DEC     IX
                DJNZ    CLTAD1
                LD      B , 2
CLTAD2 :        LD      A , (DE)
                ADC     A , (HL)
                DAA
                LD      (IX+0) , A
                DEC     HL
                DEC     DE
                DEC     IX
                DJNZ    CLTAD2
                LD      A , 0
                ADC     A , A
                LD      (IX+0) , A
                POP     IX
                CALL    CLEND
                RET
```

```
CLTADS :    LD    A , (DE)          ;A= (DE) + (HL)  IN  TIME
            ADC   A , (HL)
            DAA
            JR    NC , CLTADS1      ;<100
            ADD   A , 40H
            DAA
            SCF
            RET
CLTADS1 :   CP    60H
            JR    C , CLTADS2       ;<60
            SUB   60H
            DAA
            SCF
            RET
CLTADS2 :   SCF
            CCF
            RET
; ********** CLTSB SUB **********
; ADDRESS = 0147H
; REG = ABCDEHL
CLTSB :     PUSH  IX
            CALL  CLCLB
            LD    HL , CLSTK+7
            LD    DE , CLSTK+3
            LD    IX , CLBUF+7
            LD    B , 2
CLTSB1 :    CALL  CLTSBS
            LD    (IX+0) , A
            DEC   HL
            DEC   DE
            DEC   IX
            DJNZ  CLTSB1
            LD    B , 2
CLTSB2 :    LD    A , (DE)
```

```
                SBC     A , (HL)
                DAA
                LD      (IX+0) , A
                DEC     HL
                DEC     DE
                DEC     IX
                DJNZ    CLTSB2
                LD      A , 0
                ADC     A , A
                LD      (IX+0) , A
                POP     IX
                CALL    CLEND
                RET
CLTSBS :        LD      A , (DE)        ;A= (DE) - (HL) IN TIME
                SBC     A , (HL)
                DAA
                RET     NC
                SUB     40H
                DAA
                SCF
                RET
; ********** CLEND SUB **********  (LOCAL USE)
; REG = ABCDEHL
CLEND :         CALL    CLX
                CALL    CLBTX
                CALL    CLFAG
                RET
; ********** CLBTX SUB **********  (LOCAL USE)
; LOAD (CLBUF+4) TO (CLSTK)
; REG = BCDEHL
CLBTX :         LD      HL , CLBUF+4
                LD      DE , CLSTK
                LD      BC , 4
                LDIR
```

```
                RET
; ********** CLCLR & CLCLB SUB **********  (LOCAL USE)
; CLEAR (CLSTK) & (CLBUF)
; REG = ABCDEHL
CLCLB :         LD      HL , CLBUF
                LD      B , 8
                JR      CLCLR1
CLCLR :         LD      HL , CLSTK
                LD      B , 4
CLCLR1 :        XOR     A
CLCLR2 :        LD      (HL) , A
                INC     HL
                DJNZ    CLCLR2
                RET
; ********** CLFAG SUB **********  (LOCAL USE)
; REG = ABCHL
; SET CARRY & ZERO FLAG
CLFAG :         LD      C , 0
                LD      HL , CLBUF
                LD      B , 4
CLFAG1 :        LD      A , (HL)
                CP      0
                JR      Z , CLFAG2
                SET     0 , C
                JR      CLFAG3
CLFAG2 :        INC     HL
                DJNZ    CLFAG1
CLFAG3 :        LD      HL , CLSTK
                LD      B , 4
CLFAG4 :        LD      A , (HL)
                CP      0
                JR      NZ , CLFAG5
                INC     HL
                DJNZ    CLFAG4
```

```
                SET     7 , C
CLFAG5 :        PUSH    BC
                POP     AF
                RET
;***************************
;       SOUND SUB     *
;***************************
; ADDRESS       = 0148H
; I / P     = C   frequency
;             HL  length
; REG       = -
SOUND :         PUSH    HL
                PUSH    DE
                PUSH    BC
                PUSH    AF
SOUND1 :        LD      D , 0
                LD      A , 00001111B
                CALL    SOUND2
                LD      A , 00001110B
                CALL    SOUND2
                DEC     D
                JR      NZ , SOUND1
                POP     AF
                POP     BC
                POP     DE
                POP     HL
                RET
SOUND2 :        OUT     (ANALOG) , A
                LD      E , C
                OR      A
SOUND3 :        DEC     HL
                LD      A , H
                OR      L
                JR      NZ , SOUND4
```

```
                INC     D
SOUND4 :        LD      B , 4
                DJNZ    $
                DEC     E
                JR      NZ , SOUND3
                RET
;


;*************************
;   L&H BEEP SUB      *
;*************************
; ADDRESS      = 0149H
; REG      = -
HBEEP :         LD      C , HBPFRE
                JR      BEEPS
LBEEP :         LD      C , LBPFRE
                JR      BEEPS
BEEPS :         PUSH    HL
                LD      HL , 2000H
                CALL SOUND
                POP     HL
                RET
;
; /* RTC GROUP PROGRAM */
;-------------------------------------------------------
;*************************
;     SET RTC           *
;*************************
; ADDRESS      = 014BH
; I / P      = B , C , D (HOUR , MINUTE , SECOND)
; REG      = -
;
WR_TIME :       PUSH    AF
```

```
                PUSH    BC
                PUSH    HL
                LD      A , 00000111B    ;24 , STOP , RES
                OUT     (CREG_F) , A
;
                LD      L , C
                LD      C , HOUR1        ;HOUR
                CALL WR_RTC
;
                LD      C , MIN1;MINUTE
                LD      B , L
                CALL    WR_RTC
;
                LD      C , SEC1;SECOND
                LD      B , D
                CALL    WR_RTC
                LD      A , 00000100B    ;RUN
                OUT     (CREG_F) , A
                POP     HL
                POP     BC
                POP     AF
                RET
;*************************
;    WRITE DATE         *
;*************************
; ADDRESS      = 014CH
; I/P      = B , C , D (DAY , MONTH , YEAR)
; REG      = -
WR_DATE :       PUSH    AF
                PUSH    BC
                PUSH    HL
                LD      A , 00000111B    ;24 , STOP , RES
                OUT     (CREG_F) , A
;
```

```
                LD      L , C
                LD      C , DATE1       ;DATE
                CALL    WR_RTC
;
                LD      C , MONT1       ;MINUTE
                LD      B , L
                CALL    WR_RTC
;
                LD      C , YEAR1       ;SECOND
                LD      B , D
                CALL    WR_RTC
;
                LD      A, 00000100B    ;RUN
                OUT     (CREG_F) ,A
                POP     HL
                POP     BC
                POP     AF
                RET
;**************************
; WRITE WEEK RTC    *
;**************************
; ADDRESS       = 014DH
; I / P         = B (WEEK)
; REG           = -
WR_WEEK :               PUSH    AF
                LD      A , B
                OUT     (WEEK) , A
                POP     AF
                RET
;**************************
;    READ TIME*
;**************************
; ADDRESS       = 014EH
; O / P         = B , C , D (HOUR , MINUTE , SECOND)
```

```
; REG     = B , C , D
RD_TIME :       PUSH    AF
                PUSH    HL
                LD      C , HOUR1
                CALL    RD_RTC
                LD      H , A           ;HOUR
;
                LD      C , MIN1
                CALL    RD_RTC
                CL      L , A           ;MINUTE
;
                LD      C , SEC1
                CALL    RD_RTC
                LD      D , A           ;SEC
                LD      C , L           ;MINUTE
                LD      B , H           ;HOUR
;
                POP     HL
                POP     AF
                RET
;


;**************************
;    READ DATE          *
;**************************
; ADDRESS       = 014FH
; O / P     = B , C , D (DAY , MONTH , YEAR)
; REG       = B , C , D
RD_DATE :
                PUSH    AF
                PUSH    HL
                LD      C , DATE1
                CALL    RD_RTC
```

```
                LD      H , A           ;DATE
;
                LD      C , MONT1
                CALL    RD_RTC
                LD      L , A           ;MONTH
;
                LD      C , YEAR1
                CALL    RD_TRC
                LD      D , A           ;YEAR
                LD      C , L           ;MINUTE
                LD      B , H           ;HOUR
;
                POP     HL
                POP     AF
                RET


;**************************
; READ WEEK RTC    *
;**************************
; ADDRESS      = 0150H
; O / P     = B
; REG     = B
RD_WEEK :       PUSH    AF
                IN      A , (WEEK)
                AND     0FH
                LD      B , A
                POP     AF
                RET
;
;**************************----------(LOCAL USE)
; WRITE PORT RTC    *
;**************************
```

```
; I/P      = C (PORT LOW OF RTC UNIT)
; REG      = -
WR_RTC :        PUSH    AF
                PUSH    BC
                CALL    HEX_BCD
                OUT     (C) , B         ;LOW
                INC     C
                OUT     (C) , A         ;HIGH
                POP     BC
                POP     AF
                RET


;*************************---------------------------(LOCAL USE)
;  READ PORT RTC      *
;*************************
; I/P      = C (PORT LOW OF RTC UNIT)
; O/P  = A
; REG  = A
RD_RTC :        PUSH    BC
                INC     C               ;HIGH
                IN      B , (C)         ;
                LD      A , B
                AND     0FH
                LD      B , A
                DEC     C               ;LOW
                IN      A , (C)         ;
                AND     0FH
                LD      C , A
                CALL    BCD_HEX
                POP     BC
                RET
;
; /* SERIAL GROUP PROGRAM */
```

```
;-------------------------------------------------------------
;*********************************
;  GET BAUD RATE CH1          *
;*********************************
; ADDRESS = 0151H
; I/P      = C (BAUD RATE)
; O/P  = CF
; REG   = AF
GBAUD1 :        PUSH    DE
                LD      D , CNTLB1
                CALL    GBAUD
                POP     DE
                RET
;*********************************
;    SEND ONE BYTE SUB        *
;*********************************
; ADDRESS = 0152H
; I/P      = D
; REG   = -
TX_BYT1 :       PUSH    AF
TX_B10 :        IN0     A , (STAT1)
                BIT     1 , A
                JR      Z , TX_B10
                OUT0    (TDR1) , D
                POP     AF
                RET
;
;*********************************
;    READ SERIAL 1 BYTE         *
;*********************************
; ADDRESS = 0153H
; O/P  = D
; REG   = -
RX_BYT1 :       PUSH AF
```

```
RX_B10:     IN0     D,(STAT1)
            BIT     7,D
            JRZ,RX_B10
            IN0     D,(RDR1)
            POP     AF
            RET
;
;**********************************
;   DISPLAY HEX IN C *
;**********************************
; ADDRESS = 0154H
; I/P     = C
; REG   = -
;
DSPC:       PUSH    AF
            PUSH    DE
            PUSH    HL
            CALL    HTOA
            CALL    TX_BYTI
            LD      D,E
            CALL    TX_BYTI
            POP     HL
            POP     DE
            POP     AF
            RET
;
;**********************************
;   DISPLAY HEX IN DE       *
;**********************************
; ADDRESS = 0155H
; I/P     = DE
; REG   = -
DSPDE;      PUSH    AF
            PUSH    BC
```

```
                PUSH    DE
                LD      C , D           ;High byte
                CALL    DSPC
                LD      C , E           ;low byte
                CALL    DSPC
                POP     DE
                POP     BC
                POP     AF
                RET


;**********************************
;     DISPLAY STRING   *
;**********************************
; ADDRESS = 0156H
; I / P       = HL address string
; REG   = HL
;
TXBLOCK :       PUSH    AF
                PUSH    DE
TX_BL0 :        LD      A , (HL)
                CP      0
                JR      Z , TX_BL1
                LD      D , A
                CALL    TX_BYT1
                INC     HL
                JR      TX_BL0
TX_BL1 :        POP     DE
                POP     AF
                RET
;
;**********************************
;    DISPLAY CR & LF            *
;**********************************
```

```
; ADDRESS = 0157H
; REG  = -
CRLF :          PUSH   AF              ;Carrier & Linefeed
                PUSH   DE
                LD     D , CR
                CALL   TX_BYT1
                LD     D , LF
                CALL   TX_BYT1
                POP    DE
                POP    AF
                RET
;**********************************
;   SERIAL SEND BS    *
;**********************************
; ADDRESS = 0158H
; REG  = -
CON BS :        PUSH   AF
                PUSH   DE
                LD     D , BS
                CALL   TX_BYT1
                LD     D , " "
                CALL   TX_BYT1
                LD     D , BS
                CALL   TX_BYT1
                POP    DE
                POP    AF
                RET
;**********************************
;       RECEIVE TEXT    *
;**********************************
; ADDRESS = 0159H
; I / P      = HL
; REG  = AFHL
;
```

```
RTEXT :     PUSH   DE
            CALL   RX_BYT1
            LD     (HL), D
            INC    HL
            POP    DE
RTEXT0 :    PUSH   BC
RTEXT1 :    LD     BC, 0
RTEXT2 :    IN0    E, (STAT1)
            DEC    BC
            LD     A, B
            OR     C
            JR     Z, RTEXT3
            BIT    7, E
            JR     Z, RTEXT2
            IN0    A, (RDR1)
            LD     (HL), A
            OUT    (P_REG), A
INC         HL
            JR     RTEXT1
;
RTEXT3 :    POP    BC
            LD     A, 1AH
            LD     (HL), A
            LD     A, 0
            OUT    (P_REG), A
            RET
;
;**********************************
;   GET BAUD RATE CH0       *
;**********************************
; ADDRESS = 015AH
; I/P     = C
; O/P  = CF
; REG  = -
```

```
GBAUD0 :        PUSH    DE
                LD      D , CNTLB0
                CALL    GBAUD
                POP     DE
                RET
;***********************************
;   SEND ONE BYTE SUB.      *
;***********************************
; ADDRESS = 015BH
; IN   = D
; REG  = -
TX_BYT0 :       PUSH    AF
TX_B00 :        IN0     A , (STAT0)
                BIT     1 , A
                JR      Z , TX_B00
                OUT0    (TDR0) , D
                POP     AF
                RET
;
;***********************************
;   READ SERIAL 1 BYTE      *
;***********************************
; ADDRESS = 015CH
; O / P  = D
; REG  = -
RX_BYT0 :       PUSH    AF
RX_B00 :        IN0     D , (STAT0)
                BIT     7 , D
                JR      Z , RX_B00
                IN0     D , (RDR0)
                POP     AF
                RET
;*******************************----------(LOCAL USE)
; GET BAUD RATE SUB. *
```

```
;*******************************
; I/P  = C (12 , 23 , 48 , 96)
; O/P = Carry flag (Set BAUD rate err)
; REG = AF
GBAUD :         PUSH    BC
                PUSH    HL
                LD      HL , T_BAUD
                LD      B , 1
GBAUD1 :        LD      A , (HL)
                INC     HL
                OR      A
                SCF
                JR      Z , GBAUD2
                INC     B
                CP      C
                JR      NZ , GBAUD1
                LD      C , D
                LD      D , B
                LD      B , 0
                OR      A
                OUT     (C) , D
;
GBAUD2 :        POP     HL
                POP     BC
                RET
;
T BAUD :        DFB     96H , 48H , 24H , 12H
;
;*****************************************
;     READ DATA OF PHYCICAL*
;*****************************************
; ADDRESS = 015DH
; I/P      = CDE (PHYCICAL ADDR)
; O/P  = A
```

```
; REG  = -
RD_PHY_ADR :
                PUSH    HL
                PUH     AF
                PUSH    BC
                IN0     B , (BBR)
                CALL    GO_ADR
                LD      A , (HL)
                OUT0    (BBR) , B
                POP     BC
                LD      L , A
                POP     AF
                LD      A , L
                POP     HL
                RET
;************************************
; WRITE DATA OF PHYCICAL *
;************************************
; ADDRESS = 015EH
; I / P      = CDE (PHYCICAL ADDR)
; O / P  = A DATA
; REG  = -
WR_PHY_ADR :
                PUSH    AF
                PUSH    HL
                PUSH    BC
                IN0     B , (BBR)
                CALL    GO_ADR
                LD      (HL) , A
                OUT0    (BBR) , B
                POP     BC
;
                POP     HL
                POP     AF
```

```
                RET
;*********************************
;  INCRESS ADDRESS CDE      *
;*********************************
; ADDRESS = 015FH
; I/P      = CDE
; O/P  = CDE, ZF
; REG  = -
;
INC_CDE:        PUSH    HL
                LD      HL, 1
                ADD     HL, DE
                EX      DE, HL
                LD      L, A
                LD      A, C
                ADC     A, 0
                AND     0FH
                LD      C, A
                OR      D
                OR      E
                LD      A, L
                POP     HL
                RET
;*********************************
;  DECRESS ADDRESS CDE      *
;*********************************
; ADDRESS = 0160H
; I/P      = CDE
; O/P  = CDE, ZF
; REG  = -
;

DEC_CDE:        PUSH    HL
                EX      DE, HL
                LD      DE, 1
```

```
                SBC     HL , DE
                EX      DE , HL
                LD      L , A
                LD      A , C
                SBC     A , 0
                AND     0FH
                LD      C , A
                OR      D
                OR      E
                LD      A , L
                POP     HL
                RET
;**********************************
;   CALL PHYCICAL ADDR      *
;**********************************
; ADDRESS = 0161H
; I / P     = CDE (ADDRESS CALL)
; REG  = -
CALL_PHY :
                PUSH    AF
                IN0     A , (BBR)
                LD      (MEM+2) , A
                PUSH    BC
                LD      B , 8
                MLT     BC
                LD      A , C
                SUB     8
                LD      C , A
                LD      A , D
                SRL     A
                SRL     A
                SRL     A
                SRL     A
                CP      0FH
```

```
                JR      C , CALL_PH1
                LD      A , 7
                OR      C
                LD      C , A
                LD      A , D
                AND     8FH
                LD      D , A
CALL_PH1 :
                OUT0 (BBR) , C
                POP     BC
                LD      A , 0C3H
                LD      (CALNMC) , A
                LD (CALADR) , DE
                POP     AF
                LD      DE , CALL_PH2
                PUSH    DE
                JP      CALNMC
CALL_PH2 :
                PUSH    AF
                LD      A , (MEM+2)
                OUT0    (BBR) , A
                POP     AF
                RET
;
;*********************************---------------------------(LOCAL USE)
; GET ADDRESS PHYCICAL     *
;*********************************
; I / P      = CDE
; O / P  = HL (ADDRESS FOR CALL)
; REG   = -
GO_ADR :        PUSH    AF
                PUSH    DE
                LD      A , D
                SRL     A
```

```
                SRL     A
                SRL     A
                SRL     A
                SLA     C
                SLA     C
                SLA     C
                SLA     C
                OR      C
                SUB     8
                OUT0    (BBR), A
                LD      A, D
                AND     0FH
                LD      D, A
                LD      HL, USRAM
                ADD     HL, DE
                POP     DE
                POP     AF
                RET
; /* ----------SUBROUTINE BIT ACCESS ---------- */
;**********************************
;    SET I/O NUMBER *
;**********************************
; ADDRESS : 0162H
; I/P     = HL (Address of number port)
; REG  = -
I_O_MAP:        LD      (ADRP_USR), HL
                RET
;**********************************
;   OUT BLOCD I/O BIT          *
;**********************************
; ADDRESS = 0163H
; I/P     = HL (No. port), D (data 00, ff)
; REG  = HL
BLOCK_IO:       PUSH    AF
```

```
                PUSH    BC
                PUSH    DE
                LD      E , 0
BL_IO1 :        LD      C , (HL)
                INC     HL
                LD      A , (HL)
                DEC     HL
                OR      C
                JR      Z , BL_I011
                INC     E
                LD      C , (HL)
                LD      B , 0
                OUT     (C) , D
                INC     HL
                JR      BL_I01
;
BL_I011 :       LD      HL , MEM_PORT
BL_I02 :        LD      (HL) , D
                INC     HL
                DEC     E
                JR      NZ , BL_IO2
                POP     DE
                POP     BC
                POP     AF
                RET

;***********************************
;    CLEAR MEM. BIT   *
;***********************************
; ADDRESS = 0164H
; REG   = -
CLR_MEM :       PUSH    AF
                PUSH    HL
                LD      A , 32
```

```
                LD      HL , MEM_BIT
CLR_M1 :        LD      (HL) , 0
                INC     HL
                DEC     A
                JR      NZ , CLR_M1
                POP     HL
                POP     AF
                RET
;*********************************
;   OUT BYTE BIT I/O *
;*********************************
; ADDRESS = 0165H
; I/P       = B (DATA) , C (PORT)
; REG   = -
OUT_BYT_IO :
                PUSH    AF
                PUSH    BC
                PUSH    DE
                PUSH    HL
                LD      HL , (ADRP_USR)
                LD      A , C
                LD      D , 0
;
OUT_BYT0 :      CP      (HL)
                JR      Z , OUT_BYT1
                INC     D
                INC     HL
                JR      OUT_BYT0
;
OUT_BYT1 :      LD      C , D
                LD      D , B
                LD      B , 0
                LD      HL , MEM_PORT
                ADD     HL , BC
```

```
                LD      (HL), D
                LD      C, A
                OUT     (C), D
                POP     HL
                POP     DE
                POP     BC
                POP     AF
                RET
;**********************************
;   CLEAR BIT n. .nn (port)       *
;**********************************
; ADDRESS = 0166H
; I/P      = B (bit of port)
; REG  = -
CLR_PBIT:       PUSH    HL
                LD      HL, FIND_P
                CALL    CLR_B
                POP     HL
                RET
;**********************************
;   SET BIT n. . nn (port) *
;**********************************
; ADDRESS = 0167H
; I/P      = B (bit of port)
; REG  = -
;
SET_PBIT:       PUSH    HL
                LD      HL, FIND_P
                CALL    SET_B
                POP     HL
                RET
;**************************************
;COMPLEMENT BIT n. .nn (port) *
;**************************************
```

```
; ADDRESS = 0168H
; I/P     = B (bit of port)
; REG   = -
;
CPL_PBIT :      PUSH    HL
                LD      HL , FIND_P
                CALL    CPL_B
                POP     HL
                RET


;*********************************
; LD CY -> BIT n. .nn (port)      *
;*********************************
; ADDRESS = 0169H
; I/P     = B (bit of port)
; REG   = -
;
LD_PBIT_C :     PUSH    HL
                LD      HL , FIND_P
                CALL    LD_B_CY
                POP     HL
                RET
;*********************************
; LD CY <- BIT n. .nn (port)      *
;*********************************
; ADDRESS = 016AH
; I/P     = B (bit of port)
; O/P     = CY
; REG   = -
;
LD_C_PBIT :     PUSH    HL
                LD      HL , FIND_P
                CALL    LD_CY_B
```

```
                POP     HL
                RET
;**********************************
;  OR CY BIT n. .nn (port)        *
;**********************************
; ADDRESS = 016BH
; I/P     = B (bit of port)
; O/P     = CY
; REG   = -
OR_PBIT :       PUSH    HL
                LD      HL , FIND_P
                CALL    OR_B
                POP     HL
                RET
;
;*****************************************
;OR CY NOT BIT n. .nn (port)     *
;*****************************************
; ADDRESS = 016CH
; I/P     = B (bit of port)
; O/P     = CY
; REG   = -
ORN_PBIT :      PUSH    HL
                LD      HL , FIND_P
                CALL    OR NOTB
                POP     HL
                RET
;
;**********************************
; AND CY BIT n. .nn (port)       *
;**********************************
; ADDRESS = 016DH
; I/P     = B (bit of port)
; O/P     = CY
```

```
; REG  = -
AND_PBIT :      PUSH   HL
                LD     HL , FIND_P
                CALL   AND_B
                POP    HL
                RET
;
;**************************************
; AND CY NOT BIT n. .nn (port) *
;**************************************
; ADDRESS = 016EH
; I / P     = B (bit of port)
; O / P     = CY
; REG  = -
ANDN_PBIT ;     PUSH   HL
                LD     HL , FIND_P
                CALL   AND_NOTB
                POP    HL
                RET
;***********************************
; XOR CY BIT n. .nn (port)       *
;***********************************
; ADDRESS = 016FH
; I / P     = B (bit of port)
; O / P     = CY
; REG  = -
XOR_PBIT :      PUSH   HL
                LD     HL , FIND_P
                CALL   XOR_B
                POP    HL
                RET
;
;**************************************
; XOR CY NOT BIT n. .nn (port) *
```

```
;*************************************
; ADDRESS = 0170H
; I/P     = B (bit of port)
; O/P     = CY
; REG  = -
XORN_PBIT:    PUSH   HL
              LD     HL,FIND_P
              CALL   XOR_NOTB
              POP    HL
              RET
;***********************************
;        CLEAR BIT (PORT)      *
;***********************************
; ADDRESS = 0171H
; I/P     = B (bit 0-7),c (port)
; REG  = -
CLR_PORT:     PUSH   HL
              LD     HL,OUT_BIT
              CALL   CLR_B
              POP    HL
              RET
;***********************************
;        SET BIT (PORT)*
;***********************************
; ADDRESS = 0172H
; I/P     = B (bit 0-7),c (port)
; REG  = -
;
SET_PORT:     PUSH   HL
              LD     HL,OUT_BIT
              CALL   SET_B
              POP    HL
              RET
```

```
;**********************************
; COMPLEMENT BIT (PORT)     *
;**********************************
; ADDRESS = 0173H
; I/P     = B (bit 0 - 7), c (port)
; REG  = -
;
CPL_PORT :      PUSH   HL
                LD     HL , OUT_BIT
                CALL   CPL_B
                POP    HL
                RET
;**********************************
;       OUT CY -> (PORT)      *
;**********************************
; ADDRESS = 0174H
; I/P     = B (bit 0 - 7), c (port)
; REG  = -
;
OUT_CY :        PUSH   HL
                LD     HL , OUT_BIT
                CALL   LD_B_CY
                POP    HL
                RET
;**********************************
;       IN CY <- (PORT)       *
;**********************************
; ADDRESS = 0175H
; I/P     = B (bit 0 - 7), c (port)
; O/P     = CY
; REG  = -
;
IN_CY :         PUSH   HL
                LD     HL , OUT_BIT
```

```
                CALL    LD_CY_B
                POP     HL
                RET
;***********************************
;         OR CY (PORT)  *
;***********************************
; ADDRESS = 0176H
; I/P       = B (bit 0 - 7), c (port)
; O/P       =CY
; REG   = -
;
OR_PORT :       PUSH    HL
                LD      HL, OUT_BIT
                CALL    OR_B
                POP     HL
                RET
;
;***********************************
;         OR CY NOT (PORT)      *
;***********************************
; ADDRESS = 0177H
; I/P       = B (bit 0 - 7), c (port)
; O/P       = CY
; REG   = -
OR_NPORT :      PUSH    HL
                LD      HL, OUT_BIT
                CALL    OR_NOTB
                POD     HL
                RET
;
;***********************************
;         AND CY (PORT)*
;***********************************
; ADDRESS = 0178H
```

```
; I/P      = B (bit 0 - 7), c (port)
; O/P      = CY
; REG  = -
AND_PORT:    PUSH   HL
             LD     HL, OUT_BIT
             CALL   AND_B
             POP    HL
             RET


;
;**********************************
;   AND CY NOT (PORT)        *
;**********************************
; ADDRESS = 0179H
; I/P      = B (bit 0 - 7), c (port)
; O/P     =CY
; REG  = -
AND_NPORT:   PUSH   HL
             LD     HL, OUT_BIT
             CALL   AND_NOTB
             POP    HL
             RET
;
;**********************************
;         XOR CY (PORT)*
;**********************************
; ADDRESS = 017AH
; I/P      = B (bit 0 - 7), c (port)
; O/P      = CY
; REG  = -
XOR_PORT:    PUSH   HL
             LD     HL, OUT_BIT
             CALL   XOR_B
```

```
                POP     HL
                RET
;
;**********************************
;  XOR CY NOT (PORT)*
;**********************************
; ADDRESS = 017BH
; I/P     = B (bit 0 - 7), c (port)
; O/P     = CY
; REG   = -
XOR_NPORT :   PUSH    HL
                LD      HL, OUT_BIT
                CALL    XOR_NOTB
                POP     HL
                RET
;**********************************
;  CLEAR BIT n..nn (MEM)      *
;**********************************
; ADDRESS = 017CH
; I/P     = B (bit of MEM)
; REG   = -
CLR_BIT :       PUSH    HL
                LD      HL, FIND_B
                CALL    CLR_B
                POP     HL
                RET
;**********************************
;  SET BIT n..nn (MEM)*
;**********************************
; ADDRESS = 017DH
; I/P     = B (bit of MEM)
; REG   = -
SET_BIT :       PUSH    HL
                LD      HL, FIND_B
```

```
                CALL    SET_B
                POP     HL
                RET
;****************************************
;COMPLEMENT BIT n..nn (MEM) *
;****************************************
; ADDRESS = 017EH
; I/P     = B (bit of MEM)
; REG   = -
;
CPL_BIT :       PUSH    HL
                LD      HL , FIND_B
                CALL    CPL_B
                POP     HL
                RET
;****************************************
;LD CY -> BIT n..nn (MEM)     *
;****************************************
; ADDRESS = 017FH
; I/P     = B (bit of MEM)
; REG   = -
;
LD_BIT_C :      PUSH    HL
                LD      HL , FIND_B
                CALL    LD_B_CY
                POP     HL
                RET
;****************************************
;LD CY <- BIT n..nn (MEM)     *
;****************************************
; ADDRESS = 0180H
; I/P     = B (bit of MEM)
; O/P     = CY
; REG     = -
```

```
;  AND CY BIT n . .nn (MEM)     *
;*****************************************
; ADDRESS = 0183H
; I / P      = B (bit of MEM)
; O / P      = CY
; REG        = -
AND_BIT :       PUSH    HL
                LD      HL , FIND_B
                CALL    AND_B
                POP     HL
                RET
;
;************************************************
;  AND CY NOT BIT n . .nn (MEM)     *
;************************************************
; ADDRESS = 0184H
; I / P      = B (bit of MEM)
; O / P      = CY
; REG        = -
ANDN_BIT :      PUSH    HL
                LD      HL , FIND_B
                CALL    AND_NOTB
                POP     HL
                RET
;
;*****************************************
;  XOR CY BIT n . .nn (MEM)     *
;*****************************************
; ADDRESS = 0185H
; I / P      = B (bit of MEM)
; O / P      = CY
```

```
; REG     = -
XOR_BIT :       PUSH    HL
                LD      HL , FIND_B
                CALL    XOR_B
                POP     HL
                RET
;
;***********************************************
;  XOR CY NOT BIT n . .nn (MEM)     *
;***********************************************
; ADDRESS = 0186H
; I / P     = B (bit of MEM)
; O / P     = CY
; REG     = -
XORN_BIT :      PUSH    HL
                LD      HL , FIND_B
                CALL    XOR_NOTB
                POP     HL
                RET
;
;*****************************----------------------(COCAL USE)
;  CLEAR BIT PORT     *
;*****************************
; I / P = B (BIT 00 - FF) , HL = SUBROUTINE
; REG = -
CLR_B :         PUSH    AF
                PUSH    BC
                PUSH    DE
                PUSH    HL
                LD      A , 11111110B           ;MASK OFF
                LD      DE , CLR_RET
                PUSH    DE
                PUSH    HL
                RET                             ;GOTO SUBROUTINE IN  HL
```

```
CLR_RET :       AND     B
                LD      (HL), A         ;SAVE TEMP
                POP     HL
                OR      A
                LD      DE, FIND_B
                SBC     HL, DE
                JR      Z, CLR_B1
                LD      B, 0
                OUT     (C) , A         ;OUT REAL PORT
CLR_B1 :        POP     DE
                POP     BC
                POP     AF
                RET
;
;*****************************----------(COCAL USE)
; SET BIT PORT    *
;*****************************
; I/P = B (BIT 00 - FF), HL = SUBROUTINE
; REG = -
SET_B :         PUSH    AF
                PUSH    BC
                PUSH    DE
                PUSH    HL
                LD      A, 00000001B            ;MASK ON
                LD      DE, SET_RET
                PUSH    DE
                PUSH    HL
                RET                             ;GOTO SUBROUTINE
SET_RET :       OR      B
                LD      (HL), A
                POP     HL
                OR      A
                LD      DE, FIND_B
                SBC     HL, DE
```

```
                JR      Z , SET_B1
                LD      B , 0
                OUT     (C) , A
SET_B1 :        POP     DE
                POP     BC
                POP     AF
                RET
;
;**************************************----------(COCAL USE)
; COMPLEMENT BIT PORT     *
;**************************************
; I / P = B (BIT 00 - FF) , HL = SUBROUTINE
; REG = -
CPL_B :         PUSH    AF
                PUSH    BC
                PUSH    DE
                PUSH    HL
                LD      A, 00000001B            ;
                LD      DE , CPL_RET
                PUSH    DE
                PUSH    HL
                RET                             ;GOTO SUBROUTINE
CPL_RET :       TST     B
                JR      Z , CPL_B1
                CPL
                AND     B
                JR      CPL_B2
;
CPL_B1 :        OR      B
CPL_B2 :        LD      (HL) , A
                POP     HL
                OR      A
                LD      DE , FIND_B
                SBC     HL , DE
```

```
                JR      Z , CPL_B3
                LD      B , 0
                OUT     (C) , A
CPL_B3 :        POP     DE
                POP     BC
                POP     AF
                RET
;
;*****************************----------------------------(COCAL USE)
; LOAD PBIT <- CARRY *
;*****************************
; I / P = B (BIT 00 - FF) , HL = SUBROUTINE
; REG = -
LD_B_CY :       PUSH    AF
                PUSH    BC
                PUSH    DE
                PUSH    HL
                LD      A,00000001B             ;MASK ON
                PUSH    AF
                LD      DE , LDB_RET
                PUSH    DE
                PUSH    HL
                RET                             ;GOTO SUBROUTINE
LDB_RET :       LD      D , A
                POP     AF
                JR      NC, LD_B_C1
                LD      A , D
                OR      B
                JR      LD_B_C2
;
LD_B_C1 :       LD      A , D
                CPL
                AND     B
LD_B_C2 :       LD      (HL) , A
```

```
                POP     HL
                OR      A
                LD      DE , FIND_B
                SBC     HL , DE
                JR      Z , LD_B_C3
                LD      B , 0
                OUT     (C) , A
LD_B_C3 :       POP     DE
                POP     BC
                POP     AF
                RET
;
;*****************************----------(LOCAL  USE)
-------------------------------------------------------------------------------------------------------------
;           LOAD CARRY <- PBIT      *
;****************************************
; I/P   = B       (BIT 00 – FF) , HL = SUBROUTINE
; O/P  = CY
; REG = -
LD_CY_B :       PUSH    DE
                PUSH    BC
                PUSH    AF
                PUSH    HL
                LD      A, 00000001B
                LD      DE, LDC_RET
                PUSH    DE
                PUSH    HL
                RET                                     ;GOTO SUBROUTINE

LDC_RET :       POP     HL
                OR      A
                LD      DE ,FIND_B
                SBC     HL,DE
                LD      D , B
```

```
                JR      Z, LD_CY_B1
                LD      B , 0
                IN      D, (C)

LD_CY_B1 :      TST     D
                SCF
                JR      NZ , LD_CY_B2
                CCF
LD_CY_B2 :      POP     BC
                LD      A , B
                POP     BC
                POP     DE
                RET

;
;*************************************----------(LOCAL USE)
;       OR CY <- BIT I/O        *
;*************************************
; I/P    = B      (00 – FF)
; O/P    = CY
; REG = -
OR_B:           PUSH    BC
                PUSH    AF

                PUSH    AF
                CALL    LD_CY_B
                LD      B, 0
                RL      B
                POP     AF
                LD      A, 0
                RL      A
                OR      B
                RRCA
                POP     BC
```

```
                RET
;
;*************************************----------(LOCAL USE)
;       OR  CY <- BIT NOT I/O *
;*************************************
; I/P    = B  (00 – FF)
; O/P    = CY
; REG    = -

OR_NOTB :       PUSH    BC
                PUSH    AF

                PUSH    AF
                CALL    LD_CY_B
                CCF
                LD      B, 0
                RL      B
                POP     AF
                LD      A, 0
                RL      A
                OR      B
                RRCA

                POP     BC
                LD      A, B
                POP     BC
                RET
;
;*************************************----------(LOCAL USE)
;       AND  CY <- BIT I/O     *
;*************************************
; I/P    = B     (00 – FF)
; O/P    = CY
; REG    = -
```

```
AND_B :     PUSH   BC
            PUSH   AF

            PUSH   AF
            CALL   LD_CY_B
            LD     B, 0
            RL     B
            POP    AF
            LD     A, 0
            RL     A
            AND    B
            RRCA

            POP    BC
            LD     A, B
            POP    BC
            RET
;
;*************************************----------------(LOCAL USE)
;       AND CY <- BIT NOT I/O     *
;*************************************
; I/P   = B    (00 – FF)
; O/P   = CY
; REG   = -

AND_NOTB :  PUSH   BC
            PUSH   AF

            PUSH   AF
            CALL   LD_CY_B
            CCF
```

```
                LD      B, 0
                RL      B
                POP     AF
                LD      A, 0
                RL      A
                AND     B
                RRCA

                POP     BC
                LD      A, B
                POP     BC
                RET
;
;************************************ ---------------(LOCAL USE)
;       XOR CY <- BIT I/O     *
;************************************
; I/P    = B     (00 – FF)
; O/P    = CY
; REG    = -

XOR_B :         PUSH    BC
                PUSH    AF

                PUSH    AF
                CALL    LD_CY_B
                LD      B, 0
                RL      B
                POP     AF
                LD      A, 0
                RL      A
                XOR     B
                RRCA

                POP     BC
```

```
                LD      A, B
                POP     BC
                RET
;


;*************************************----------(LOCAL USE)
;   XOR CY <- BIT NOT I/O     *
;*************************************
; I/P    = B     (00 – FF)
; O/P    = CY
; REG    = -

XOR_NOTB :      PUSH    BC
                PUSH    AF

                PUSH    AF
                CALL    LD_CY_B
                CCF
                LD      B, 0
                RL      B
                POP     AF
                LD      A, 0
                RL      A
                XOR     B
                RRCA

                POP     BC
                LD      A, B
                POP     BC
                RET
;
;*****************************************----------(LOCAL USE)
;       XOR CY <- BIT NOT I/O     *
```

```
;*****************************************
; I/P    = B     (00 – FF)
; O/P    = CY
; REG    = -

XOR_NOTB :   PUSH   BC
             PUSH   AF

             PUSH   AF
             CALL   LD_CY_B
             CCF
             LD     B, 0
             RL     B
             POP    AF
             LD     A, 0
             RL     A
             XOR    B
             RRCA

             POP    BC
             LD     A, B
             POP    BC
             RET
;
;*****************************************----------------(LOCAL USE)
;        FIND BIT I/O     *
;*****************************************
; I/P    =       B       (00 – FF), A (MASK)
; O/P    =       C       (PORT), HL (TEMP PORT), A (MASK)

FIND_P :     PUSH   DE

             PUSH   AF                  ;SAVE BIT MARK
             LD     A,B
```

```
                AND     00000111B
                LD      C, A                    ;SAVE BIT 0-3
                LD      A,B
                AND     11111000B
                RRCA
                RRCA
                RRCA                            ;ROTATE = 00011111B (32 BYTE)
                LD      E,A
                LD      D,0

                LD      HL,(ADRP_USR) ;ADDRESS USER PORT
                ADD     HL,DE                   ;GET MEMORY
                LD      B, (HL)         ;B= NUMBER PORT
;
                LD      HL,MEM_PORT  ;TEMPORARY MEM PORT
                ADD     HL,DE
                POP     AF                      ;RESTORE BIT MARK
                INC     C


FIND_P1:        DEC     C
                JR      Z_FIND_P2
                RLCA
                JR      FIND_P1
FIND_P2:        LD      C, B                    ;PORT
                LD      B, (HL)         ;BIT

                POP     DE
                RET

;************************************ ---------- ----- -----( LOCAL USE)
;       FIND BIT MEM            *
;************************************
;
```

```
' I/P    =      B (00-FF) , A (MASK)
; O/P    =      B (DATA) , A (MASK)
;REG     =      AFBC

FIND_B :        PUSH   DE

                PUSH   AF              ;SAVE BIT MARK
                LD     A,B
                AND    00000111B
                LD     C, A            ;SAVE BIT 0-3
                LD     A,B
                AND    11111000B
                RRCA
                RRCA
                RRCA                   ;ROTATE = 00011111B (32 BYTE)
                LD     E,A
                LD     D,0

                LD     HL,MEM_BIT      ;ADDRESS MEMORY BIT
                ADD    HL,DE           ;GET MEMORY
                LD     B, (HL) ;B= NUMBER PORT
;
                POP    AF              ;RESTORE BIT MARK
                INC    C
FIND_B1:        DEC    C
                JR     Z,FIND_B2
                RLCA
                JR     Z,FIND_B1
;
FIND_B2:        POP    DE
                RET
;
;*************************************----------- ----- -----( LOCAL USE)
;       FIND OUT BIT            *
```

```
;*************************************
' I/P    =      A (MASK), C (PORT), B (BIT)
; O/P    =      A (MASK),  C (PORT), B (BIT), HL =MEM PORT
;REG     =      AFBCHL

OUT_BIT :       PUSH    DE
                PUSH    AF
                LD      HL, (ADRP_USR)
                LD      A,C
                LD      D,0
;
OUT_BIT1:       CP      (HL)
                JR      Z, OUT_BIT2
                INC     D
                INC     HL
                JR      OUT_BIT1

OUT_BIT 2 :
                LD      C,D
                LD      D,B
                LD      B,0

                LD      HL, HL, MEM_PORT
                ADD     HL, BC

                LD      C,A
                LD      A,D
                AND     00000111B
                LD      D,A

                POP     AF
                INC     D
OUT_BIT3:       DEC     D
                JR      Z, OUT_BIT4
```

```
                RLCA
                JR OUT_BIT3
OUT_BIT4:       LD      B, (HL)
                POP     DE
                RET
;
;
;***********************************************
;               SCAN CODE SUB       *
;***********************************************

SCAN:           CALL    SCAN
                CP      0FFH
                JR      Z, SCAN
                LD      D, REP2
                CALL    K_NEW
                JR      C, SCAN
                CALL    K_CODE
                RET

; -------------------------------
;               SCAN SUB          -
;-------------------------------- -

; IN    =       (DISPY)
;       =       B4 OF (SYSFAG) 0=CHECK – INSTANT – KEY 1 =DON'T – CARE
; OUT   =       A
; REG   =       A
SCANN:           PUSH   BC
                LD      B,8             ;LOOP COUNT
                LD      C,0             ;TABLE COUNT

SCAN1:          CALL    SCANS
                IN      A, (RDKEY)      ;IN KEY
```

```
                AND     0FH
                CP      0FH
                JR      NZ, SCAN12

SCAN11:         LD      A, 10H          ;INC COL
                ADD     A,C
                LD      C,A
                DJNZ    SCAN1



                LD      A, (SYSFAG)
                RES     0,A                     ;IS RELEASE RESET PRESS FLAG
                LD      (SYSFAG),A
                LD      A,REP1
                LD      (REPDLY) ,A
                LD      A, 0FFH         ;Clear buffer key
                LD      (KEYIN) , A
SCAN12:         LD      B, 0
SCAN13:         NOP
                NOP
                NOP
                DJNZ    SCAN13


                CP      0FFH
                CALL    NZ, K_PRESS
                POP     BC
                RET
;
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
;               KEY PRESS SUB.          -
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
' O/P    =      reg A
; REG    =      ABC
KEY_PRESS :     PUSH    HL
```

```
                OR      C                       ;OR WITH TABLE COUNT
                LD      BC,K_ASCII-KEYTAB
                LD      HL,KEYTAB
                CPIR
                LD      BC , KEYTAB + 1
                SBC     HL,BC
                LD      A,L
                POP     HL
                RET


;---------- -------- --- -------
;               NEW KEY  SUB.       -
;----- -------------------------
; REG   =       none

K_NEW           PUSH    HL
                PUSH    BC
                LD      L,A
                LD      A, (SYSFAG)
                BIT     0,A
                JR      NZ, K_NEW1
                SET     0,A
                LD      (SYSFAG) , A
                CALL    HBEEP
                OR      A
                JR      K_EXT
;
;_____ yes to repeat___
K_NEW1:         LD      A, (REPDLY)
                DEC     A
                SCF
                JR      NZ, H_NEW2
                CCF
```

```
                LD      A,REP2
K_NEW2:         LD      (REPDLY),A
K_EXT           LD      A,L
                POP     BC
                POP     HL
                RET
;
;**************************************
;       KEY NEW NO REP          *
;**************************************
;
K_NEWO:         PUSH    HL
                LD      HL, SYSFAG
                BIT     0, (HL)
                SCF
                JR      NZ , K   NEW01
                SET     0 , (HL)
                CCF
K_NEWO1:        POP     HL
                RET
;-------------------------------
;       NEW  KEY  PRESS  SUB.
;----- --------------------------
;  I/P   =      A key  code

;  O/P   =      A  (KEYIN)
;  REG   =      A
K_CODE :        PUSH    HL
                PUSH    BC
                LD      C,A
                LD      B,0
                IN      A,  (P_SHIFT)
                BIT     0,A
                LD      HL,K_ASCII
```

```
                JR      NZ,K_CODE1
                LD      HL,K_ASCS
K_CODE1 :       ADD     HL,BC
                LD      A,C_LOCK
                CP      (HL)
                LD      A,(SYSFAG)
                JR      NZ,K_CODE2
-------------------------------------------------------------------
                XOR     01000000B
                LD      (SYSFAG) , A
K_CODE2 :       BIT     6,A
                LD      A,(HL)
                JR      Z,K_CODE3
                CP      "A"
                JR      C,K_CODE3
                CP      "Z"+1
                JR      NC,K_CODE3
                ADD     A,20H
;
K_CODE3 :       LD      (KEYIN) ,A
                POP     BC
                POP     HL
                RET                     ;EXIT
;
;---------------------------
;       SCAN COL SUB  -
; ------------------------- -
;       I/P     C
;       reg     AF
                                        :
SCANS :         LD      A,00001000B     ;BIT 4 OFF
                BIT     4, C
                JR      Z,SCANS1
                INC     A               ;BIT 4 ON
SCANS1:         OUT     (SYSCTRL) ,A
```

```
                                        :
                LD      A, 00001010B    ;BIT 5 OFF
                BIT     5, C
                JR      Z,SCANS2
                INC     A               ;BIT 5 ON
SCANS2:         OUT     (SYSCTRL) ,A
                                        :                       :
                LD      A, 00001100B    ;BIT 6 OFF
                BIT     6, C
                JR      Z,SCANS3
                INC     A               ;BIT 6 ON
SCANS3:         OUT     (SYSCTRL) ,A
                RET
;
;*****************************************
;           CHANGE ASCII            *
;*****************************************
; Ascii (HL)      , Binary DE
;
GET_BYTE :
                PUSH    BC
                LD      B,2
                CALL    CONV_ASC
                POP     BC
                RET
;
GET_WORD :
                PUSH    BC
                LD      B,4
                CALL    CONV_ASC
                POP     BC
                RET
; / * Convert ascii  to  hex * /
CONV_ASC :
```

```
                LD      DE , 0
                CALL    BLKSKP
                CALL    HEX
                RET     C
;
CONV_A1 :       LD      A, (HL)
                CALL    HEX
                RET     C
                EX      DE,HL
                ADD     HL,HL
                ADD     HL,HL
                ADD     HL,HL
                ADD     HL,HL
                OR      L
                LD      L,A
                EX      DE,HL
                INC     HL
                DJNZ    CONV_A1
                RET
;
; / * Check  hex  over      * /
OVR_HEX :       LD      A, (HL)
                CP      0
                JR      Z,OVR_H1
                CP      " , "
                JR      Z,OVR_H1
                CP      " "
                JR      Z,OVR_H1
                SCF
                JR      OVR_H2
OVR_H1 :        OR      A
OVR_H2 :        RET
;
;************************************
```

```
;  CHECK END OF COMMAND  *
;************************************
;
CHK_END :       LD      A, (HL)
                OR      A
                JR      Z , CHK_EN1
                CP      " "
                JR      Z , CHK_EN1
                SCF
                RET
;
;/* Command ok */
CHK_EN1 :       LD      A,B     ; LEGNTH
                SUB     C
                LD      C,A
                CALL    MSG_JMP
                OR      A
                RET
;
;****************************************
;       CHECK  CHAR             *
;****************************************
; I / P   HL      =       SOURCE
;         DE      =       DESTINATION
;         B       =       NUMBER  CHAR
; O / P   Zero flag
; REG   AFC
CK_CHAR :       PUSH    DE
                PUSH    BC
                PUSH    HL

                LD      C , 0
CK_C1 : LD      A, (DE)
                CP      (HL)
```

```
                LD      A,0
                JR      NZ,CK_C2
                INC     C
                INC     HL
                INC     DE
                DJNZ    CK_C1
                LD      A,C

;
CK_C2 : POP     HL
                POP     BC
                LD      E,B
                LD      D,0
                ADD     HL,DE
CK_C3 :POP      DE
                CP      B
                RET
;**************************************
;         MASSAGE JUMP                *
;**************************************
; I / P   =      C
; O /P    =      IX (Address jump)
; REG     =      BCIX

MSG_JMP :       LD      B , 2
                MLT     BC
                ADD     IX, BC
                LD      C, (IX+0)
                LD      B, (IX+1)
                PUSH    BC
                POP     IX
                RET
;
;**************************************
```

```
;          DATA TABLE              *
;*****************************************

KEYTAB :        DFB     07H,17H,27H,37H,47H,57H,67H,77H
                DFB     0BH,1BH,2BH,3BH,4BH,5BH,6BH,7BH
                DFB     0DH,1DH,2DH,3DH,4DH,5DH,6DH,7DH
                DFB     0EH,1EH,2EH,3EH,4EH,5EH,6EH,7EH
K_ASCII :       DFB     "ZC" ,0, "BN, " , CR
              DFB       "ASDFHJL0"
              DFB       "ERTUIOP9"
              DFB       "12345678"
K_ASCS :        DFB     "YX" ,0, " # ." , 1,22H,CR
              DFB       2,3,"'G@KM/"
              DFB       4,5 "VW: =Q*"
              DFB       ESC,6,7"( )+ -" ,BS
;
;*****************************************
;          HEX TO BCD             *
;*****************************************
; I/P    =        B
; O/P    =        AB      A=HIGHB=LOW
; REG    =        AB
;
HEX_BCD :       LD      A,B
                PUSH    AF
              AND       0FH
                LD      B , A           ;Low byte
               POP      AF
               SRL      A
               SRL      A
               SRL      A
               SRL      A               ;High byte
                RET
;*****************************************
```

```
;          BCD TO HEX              *
;*****************************************
; I/P   =        BC      B=HIGH C=LOW
; O/P   =        A
; REG   =        A

BCD_HEX :      LD      A,B
               SLA     A
               SLA     A
               SLA     A
            SLA        A
               OR      C
               RET
;*********************************************
;* SUB SYSTEM FOR SYSTEM CALL *
;*********************************************
;** check hex 0-F **
; A ( hex low nibble), flag c=1 (error)

; reg          A
HEX            CP      "0"
               JR      C,ERRHEX
               CP      "9"+1
               JR      C,HEX0
               CP      "A"
               JR      C , ERRHEX
               CP      "F"+1
               JR      NC, ERRHEX
               SUB     7
HEX0 :         SUB     "0"
               OR      A
               RET
;
ERRHEX :       SCF             Error
```

```
                RET
;
;*****************************************
;       BINARY ACC ASCII     *
;*****************************************
; input  =      A
; output =      DE (ascii)
; reg    =      ADE
;
HEXASC :        PUSH   AF
                RRCA
                RRCA
                RRCA

;
CBUF :          DFS    5
REPDLY :        DFS    1
SYSFAG :        DFS    1
;
CALNMC :        DFS    1
CALADR :        DFS    2
;
CLSTK : DFS     16     ;CALCULATE BUFFER
CLBUF : DFS     8
CLMEM :         DFS    4
CLEXX           DFS    28

HCALOPR :       DFS    1
B_CALL :        DFS    4
;
                END

                RRCA
                CALL   HEXA1
```

```
                LD      D,A
                POP     AF
;
HEXA1 :AND      0FH
                ADD     A, 90H
                DAA
                ADC     A,40H
                DAA
                LD      E,A
                RET


;*****************************************
;RAM  AREA  FOR  SUBROUTIN *
;*****************************************
;
SYSTK : EQU     0FD60H
                ORG     SYSTK
MEM_PORT :      DFS     32
MEM_BIT :       DFS     32
;
COMBUF :        DFS     32
LCD_BUF :       DFS     34
ADRP_USR :      DFS     2
MEM :           DFS     4
KEYIN : DFS     1

REPDLY :        DFS     1
SYSFAG :        DFS     1
;
CALNMC :        DFS     1
CALADR :        DFS     2
;
CLSTK : DFS     16      ;CALCULATE  BUFFER
```

```
CLBUF : DFS     8
CLMEM :         DFS     4
CLEXX           DFS     28

HCALOPR :       DFS     1
B_CALL :        DFS     4
;
                END
                :
                RRCA
                CALL    HEXA1
                LD      D,A
                POP     AF
;
HEXA1 : AND     0FH
                ADD     A, 90H
                DAA
                ADC     A,40H
                DAA
                LD      E,A
                RET


;*****************************************
;RAM  AREA  FOR  SUBROUTIN *
;*****************************************
;
SYSTK : EQU     0FD60H
                ORG     SYSTK
MEM_PORT :      DFS     32
MEM_BIT :       DFS     32
COMBUF :        DFS     32
LCD_BUF :       DFS     34
ADRP_USR :      DFS     2
```

```
MEM :       DFS   4
KEYIN :     DFS   1
```

**รูปที่ ค.14** โปรแกรมควบคุมการทำงานชุดสาธิตการทำงาน PLC

# ภาคผนวก ง
# ใบงานการทดลอง

# ใบงานที่ 1
# การใช้คำสั่งเบื้องต้น

## วัตถุประสงค์

1.เพื่อให้เข้าใจคำสั่งพื้นฐานของ PLC MODE

2.เพื่อให้สามารถสร้าง ladder diagram ได้

3.เพื่อเขียนคำสั่งในรูป boolean ได้

## เครื่องมือและอุปกรณ์

1.ชุดฝึก PLC 1 ชุด

## ทฤษฎีเบื้องต้น

คำสั่งพื้นฐานที่ใช้ใน PLC MODE การทดลองในบทนี้จะประกอบด้วย 8 คำสั่งคือ

1.LD

2.LD NOT

3.AND

4.AND NOT

5.OR

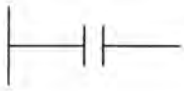
6.OR NOT

7.OUT

8.OUT NOT

**1. LD**

ชุดคำสั่ง LD เป็นชุดคำสั่งที่ใช้สำหรับอ่านหรือนำค่าสภาวะต่าง ๆ จริง (1) หรือเท็จ (0) ของอุปกรณ์ที่กำหนด เช่น อินพุท , เอาต์พุท , รีเลย์ภายใน , ตัวตั้งเวลา , หรือตัวนับ โดยคำสั่งนี้จะใช้ในการเริ่มต้นคำสั่งของ ladder diagram หรือเริ่มต้นบล็อกการทำงาน เนื่องจากการมีสภาวะทางเอาต์พุทได้ จะต้องมีส่วนอินพุท หรือค่าสภาวะที่ให้มาก่อน ซึ่งโครงสร้างของคำสั่ง LD แสดงได้ในรูปที่ 1.1

| INSTRUCTION | LADDER | BOOLEAN |
|---|---|---|
| LOAD |  | LD |

**รูปที่ 1.1** แสดงโครงสร้างของคำสั่ง LD

**2. LD NOT**

ชุดคำสั่ง LD NOT ชุดคำสั่งที่นำค่าสภาวะที่ได้กลับให้เป็นสภาวะตรงข้าม โดยรายละเอียดของคำสั่งสามารถแสดงได้ในรูปที่ 1.2

| INSTRUCTION | LADDER | BOOLEAN |
|---|---|---|
| LOAD NOT |  | LD NOT |

**รูปที่ 1.2** แสดงโครงสร้างของคำสั่ง LD NOT

**3. AND**

ชุดคำสั่ง AND เป็นชุดคำสั่งที่นำค่าสภาวะของอุปกรณ์ต่าง ๆ ที่กำหนดตั้งแต่ 2 อุปกรณ์ขึ้นไป มากระทำลอจิก AND ซึ่งสามารถทราบรายละเอียดของคำสั่งได้ในรูปที่ 1.3

| INSTRUCTION | LADDER | BOOLEAN |
|---|---|---|
| AND |  | AND |

**รูปที่ 1.3** แสดงโครงสร้างของคำสั่ง AND

**4. AND NOT**

ชุดคำสั่ง AND NOT เป็นชุดคำสั่งที่นำค่าสภาวะตรงข้ามของอุปกรณ์ที่กำหนดมากระทำลอจิก AND กับค่าสภาวะของอุปกรณ์อีกตัว ซึ่งลักษณะการเขียนคำสั่งดังแสดงในรูปที่ 1.4

| INSTRUCTION | LADDER | BOOLEAN |
|---|---|---|
| AND NOT |  | AND NOT |

**รูปที่ 1.4** แสดงโครงสร้างของคำสั่ง AND NOT

**5. OR**

ชุดคำสั่ง OR เป็นชุดคำสั่งที่นำค่าสภาวะของอุปกรณ์ต่าง ๆ ที่กำหนดตั้งแต่ 2 อุปกรณ์ขึ้นไปมากระทำลอจิก OR โดยโครงสร้างของคำสั่ง OR ดังแสดงในรูปที่ 1.5

| INSTRUCTION | LADDER | BOOLEAN |
|---|---|---|
| OR |  | OR |

**รูปที่ 1.5** แสดงโครงสร้างของคำสั่ง OR

**6. OR NOT**

ชุดคำสั่ง OR NOT เป็นชุดคำสั่งที่นำค่าสภาวะตรงข้ามของอุปกรณ์ที่กำหนดมากระทำลอจิก OR กับค่าสภาวะของอุปกรณ์อีกตัว ซึ่งลักษณะการเขียนคำสั่งดังแสดงในรูปที่ 1.6

| INSTRUCTION | LADDER | BOOLEAN |
|---|---|---|
| OR NOT |  | OR NOT |

**รูปที่ 1.6** แสดงโครงสร้างของคำสั่ง OR NOT

**7. OUT**

ชุดคำสั่ง OUT เป็นชุดคำสั่งที่นำค่าสภาวะที่ได้ส่งไปยังอุปกรณ์ที่กำหนด โดยสามารถทราบรายละเอียดของคำสั่งได้ในรูปที่ 1.7

| INSTRUCTION | LADDER | BOOLEAN |
|---|---|---|
| OUT | —( ) | OUT |

**รูปที่ 1.7** แสดงโครงสร้างของคำสั่ง OUT

**8. OUT NOT**

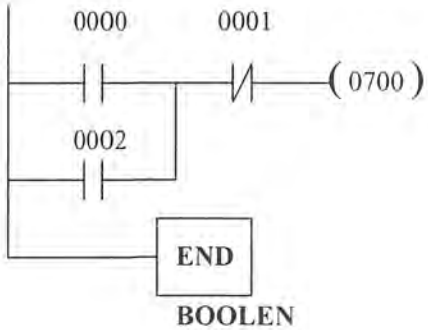
ชุดคำสั่ง OUT NOT เป็นชุดคำสั่งที่นำค่าสภาวะที่จะส่งไปให้อุปกรณ์กลับให้เป็นตรงข้ามแล้วจึงส่งไปยังอุปกรณ์ดังโครงสร้างที่แสดงในรูปที่ 1.8

| INSTRUCTION | LADDER | BOOLEAN |
|---|---|---|
| OUT NOT | —(/) | OUT NOT |

**รูปที่ 1.8** แสดงโครงสร้างของคำสั่ง OUT NOT

ตัวอย่างการนำคำสั่งพื้นฐานไปใช้งาน ดังในรูปที่ 1.9 โดยจะเป็นการแสดงในส่วนของ ladder และ boolean

**LADDER DIAGRAM**




**BOOLEN**

LD 0000

OR 0002

AND NOT 0001

OUT 0700

END

**รูปที่ 1.9** แสดง ladder diagram และ boolean ของตัวอย่างการนำคำสั่งพื้นฐานไปใช้งาน

จากตัวอย่างข้างต้นแสดงถึงการใช้คำสั่งพื้นฐานโดยมีความหมายถึงการนำค่าสภาวะของ relay 0000 มากระทำลอจิก OR กับค่าสภาวะของ relay 0002 แล้วจึงนำค่าสภาวะที่ได้ไปกระทำการ AND กับสภาวะตรงข้ามของ relay 0001 แล้ว จึงส่งค่าที่ได้ออกไปที่ relay 0700 ที่สภาวะของ relay ต่าง ๆ ในรูปที่ 1.9 แสดงได้ในตารางที่ 1.1

**ตารางที่ 1.1** แสดงผลลัพธ์ของตัวอย่าง

| 0000 | 0001 | 0002 | 0700 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 |

## ลำดับขั้นการทดลอง

**1.จงเขียน INSTRUCTION ต่อไปนี้ให้อยู่ในรูป ladder diagram และ boolean**

1.1) LOAD ค่าสภาวะของอินพุทตำแหน่ง 0000

1.2) LOAD NOT ค่าสภาวะของอินพุทตำแหน่ง 0001

1.3) AND ค่าสภาวะของอินพุทตำแหน่ง 0002 กับอินพุทตำแหน่งใด ๆ

1.4) AND NOT ค่าสภาวะของอินพุทตำแหน่ง 0003 กับอินพุทตำแหน่งใด ๆ

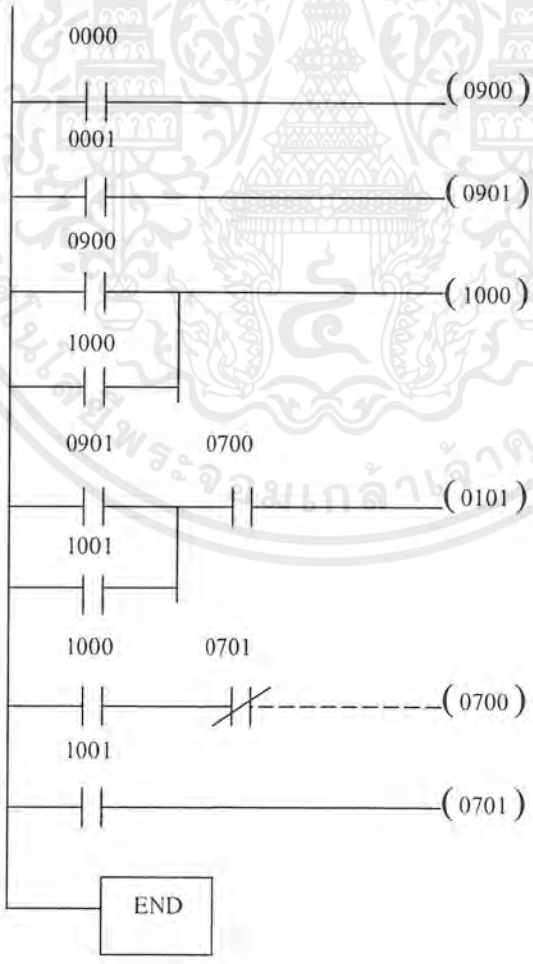
1.5) OR ค่าสภาวะของอินพุทตำแหน่ง 0004 กับอินพุทตำแหน่งใด ๆ

1.6) OR NOT ค่าสภาวะของอินพุทตำแหน่ง 0005 กับอินพุทตำแหน่งใด ๆ

1.7) OUT ค่าสภาวะออกเอาต์พุทตำแหน่ง 0700

1.8) OUT NOT ค่าสภาวะออกเอาต์พุทตำแหน่ง 0701

**2.จงแปลง ladder giagram ต่อไปนี้ให้เป็น boolean**



**3. จงแปลง boolean ต่อไปนี้ให้เป็น ladder diagram**

LD 0000

OUT 0900

LD 0900

OR 1000

AND NOT 0901

OUT 1000

LD 0001

OUT 0901

LD 0901

AND NOT 1000

OR 0900

OUT 0700

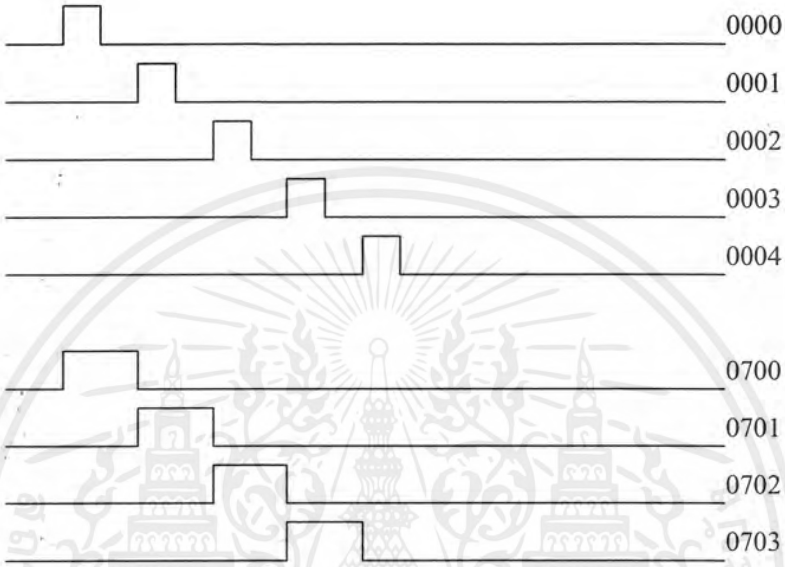
LD NOT 0701

AND 0901

OUT 0701

END

4. จงเขียนโปรแกรมตาม ไดอะแกรมเวลา ต่อไปนี้ โดยป้อนสภาวะของรีเลย์ทาง อินพุทตามไดอะแกรมเวลา เพื่อให้ได้สภาวะของรีเลย์ทางเอาต์พุทตามเวลาที่กำหนดด้วย พร้อมสรุปผลการทดลอง




5. สรุปผลการใช้คำสั่งเบื้องต้น

# ใบงานที่ 2
# การกระทำคำสั่งทางบล็อก

## วัตถุประสงค์

1.เพื่อให้เข้าใจและสามารถนำคำสั่งทางบล็อกไปใช้งานได้

2.สามารถสร้างคำสั่งทางบล็อก ในรูปของ ladder diagram ได้

3.สามารถเขียนคำสั่งทางบล็อก ในรูป boolean ได้

## เครื่องมือและอุปกรณ์

1.ชุดฝึก PLC 1 ชุด

## ทฤษฎีเบื้องต้น

คำสั่งทางบล็อก ของ PLC Mode จะประกอบด้วย 2 คำสั่งคือ

1.AND - LD

2. OR - LD

**1. AND LD**

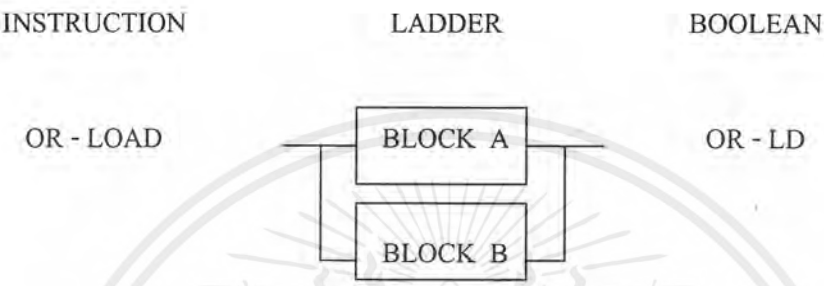
ชุดคำสั่ง AND - LD เป็นชุดคำสั่งที่นำหน้าสัมผัสของอุปกรณ์ที่ขนานกันตั้งแต่ 2 หน้าสัมผัสขึ้นไปมาทำการ AND เข้ากับชุดหน้าสัมผัสที่มีการขนานกับอีกชุดหนึ่ง ซึ่งจะเป็นการขนานหน้าสัมผัสที่ 1 อนุกรมกับการขนานหน้าสัมผัสชุดที่ 2 โดยโครงสร้างของคำสั่งสามารถที่แสดงได้ในรูปที่ 2.1

| INSTRUCTION | LADDER | BOOLEAN |
|---|---|---|
| AND - LOAD | BLOCK A — BLOCK B | AND - LD |

**รูปที่ 2.1** แสดงโครงสร้างของคำสั่ง AND - LOAD

**2. OR LD**

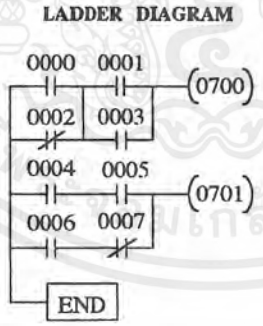
ชุดคำสั่ง OR-LD เป็นชุดคำสั่งที่นำหน้าสัมผัสของอุปกรณ์ที่อนุกรมกันตั้งแต่ 2 หน้าสัมผัสขึ้นไปมากระทำการ OR เข้ากับชุดหน้าสัมผัสที่อนุกรมกับอีกชุดหนึ่ง คือ หน้าสัมผัส ที่ 1 OR เข้ากับหน้าสัมผัส ที่ 2 ซึ่งการใช้งานของคำสั่งดังแสดงในรูปที่ 2.2




**รูปที่ 2.2** แสดงโครงสร้างของคำสั่ง OR - LOAD

**ตัวอย่าง**

ตัวอย่างต่อไปนี้เป็นการแสดงให้เห็นถึงการนำคำสั่ง AND - LOAD และ OR - LOAD ไปใช้งานเบื้องต้น ซึ่งในตัวอย่างได้มีการแสดงการใช้งานทั้ง การเขียน ladder diagram และการเขียนคำสั่งในรูปของ boolean รวมกันไว้ในรูปที่ 2.




**BOOLEAN**

LD 0000

OR NOT 0002

LD 0001

OR 0003

AND - LD

OUT 0700

LD 0004

LD 0006

AND NOT 0007

OR - LD

OUT 0701

END

**รูปที่ 2.3** แสดงตัวอย่าง ladder diagram และ boolean ของคำสั่ง AND - LOAD และ OR – LOAD

จากตัวอย่างข้างต้น ชุดรีเลย์ที่ควบคุม เอาต์พุท 0700 คือรีเลย์ 0000 , 0001 , 0002 , 0003 ประกอบร่วมกัน เรียกการกระทำของรีเลย์ชุดนี้ว่า การกระทำ AND - LD ส่วนการกระทำของรีเลย์ อีกชุดหนึ่งคือ รีเลย์ 0004 , 0005 , 0006 , 0007 ประกอบร่วมกันคือการกระทำที่เรียกว่า OR - LD ซึ่งจะได้เอาต์พุทออกที่รีเลย์ 0701 ซึ่งคำตอบที่ได้จากตัวอย่างแสดงในตารางที่ 2.1

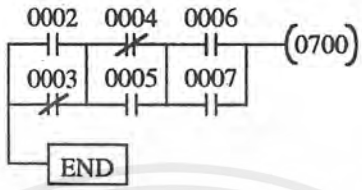
**ตารางที่ 2.1** แสดงผลลัพธ์ของตัวอย่าง

| 0000 | 0001 | 0002 | 0003 | 0004 | 0005 | 0006 | 0007 | 0700 | 0701 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

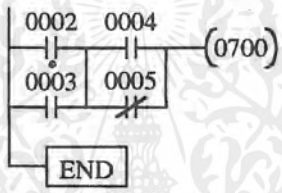
# ลำดับขั้นการทดลอง

**1.จงเขียน ladder diagram ต่อไปนี้เป็น boolean**

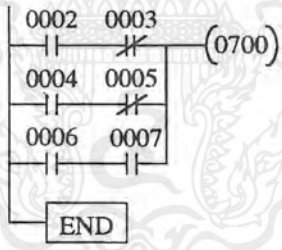
1.1



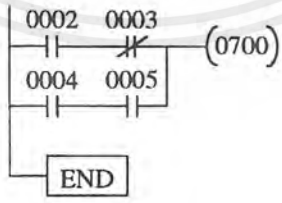

1.2




1.3




1.4



**2. จงเขียน boolean ต่อไปนี้ ในรูปของ ladder diagram**

2.1

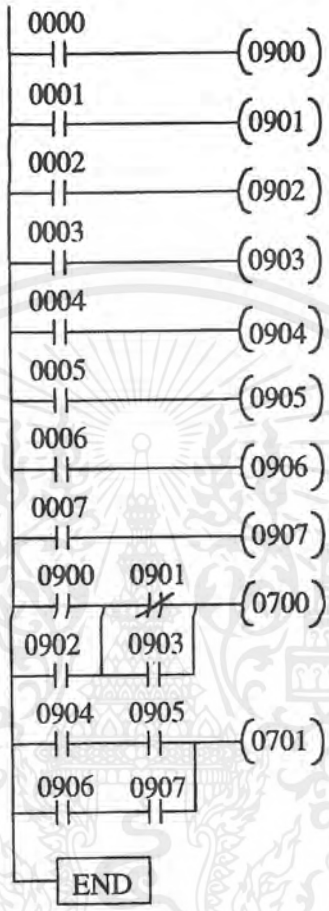
```
LD 0000
OR 0001
LD 0002
OR NOT 0003
LD NOT 0004
OR 0005
AND - LD
AND - LD
OUT 0701
END
```

2.2

```
LD 0000
AND NOT 0001
LD NOT 0002
AND 0003
OR - LD
LD 0004
AND NOT 0005
OR - LD
OUT 0700
END
```

**3.จงทดสอบโปรแกรมต่อไปนี้**

**LADDER DIAGRAM**




**BOOLEAN**

LD 0000

OUT 0900

LD 0001

OUT 0901

LD 0002

OUT 0902

LD 0003

OUT 0903

LD 0004

```
OUT 0904
LD 0005
OUT 0905
LD 0006
OUT 0906
LD 0007
OUT 0907
LD 0900
OR 0902
LD NOT 0901
OR 0903
AND - LD
OUT 0700
LD 0904
AND 0905
LD 0906
AND 0907
OR - LD
OUT 0701
END
```

ทดสอบเปลี่ยนสภาวะลอจิกดังตารางที่ 2.2 แล้วบันทึกสภาวะของรีเลย์ 0700 และ 0701 ลงตารางที่ 2.2 พร้อมสรุปผลการทดลอง

**ตารางที่ 2.2** ตารางบันทึกผลการทดลอง

| 0000 | 0001 | 0002 | 0003 | 0004 | 0005 | 0006 | 0007 | 0700 | 0701 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | | |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | | |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | | |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | | |

**4. สรุปผลการใช้คำสั่งทางบล็อก**

# ใบงานที่ 3
# การใช้คำสั่ง DIFU/DIFD

## วัตถุประสงค์

1.เพื่อให้เข้าใจการทำงานของคำสั่ง DIFU/DIFD

2.เพื่อประยุกต์ใช้คำสั่ง DIFU/DIFD ได้

3.สามารถหาขอบขาขึ้นและขอบขาลงของพัลส์ได้

## เครื่องมือและอุปกรณ์

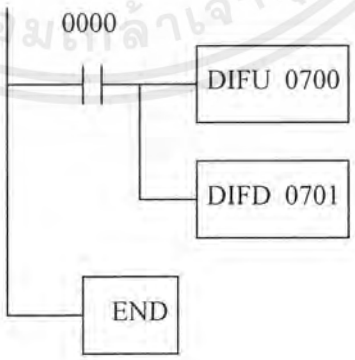
1.ชุดฝึก PLC 1 ชุด

## ทฤษฎีเบื้องต้น

คำสั่ง DIFFERENTION - UP (DIFU) และ DIFFERENTION - DOWN (DIFD) จะมีผลต่อตำแหน่งรีเลย์ที่ถูกระบุด้วยเวลาเพียง SCANTIME เดียวเท่านั้น ขณะที่ได้รับเงื่อนไข "ON" จาก CONTRAC ที่อยู่ด้านหน้า

**ตัวอย่าง**

**LADDER DIAGRAM**



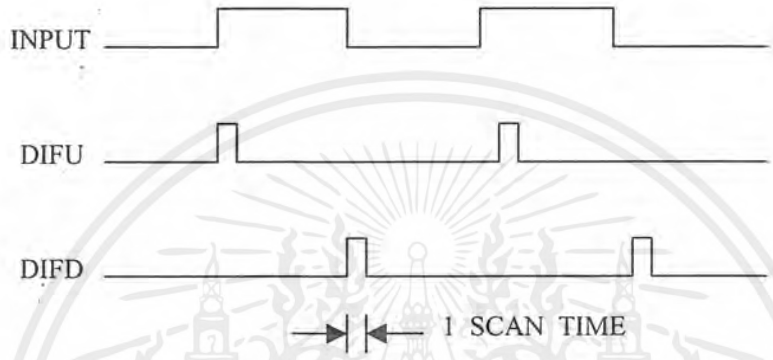
**BOOLEAN**

LD 0000

DIFU 0700

DIFD 0701

END

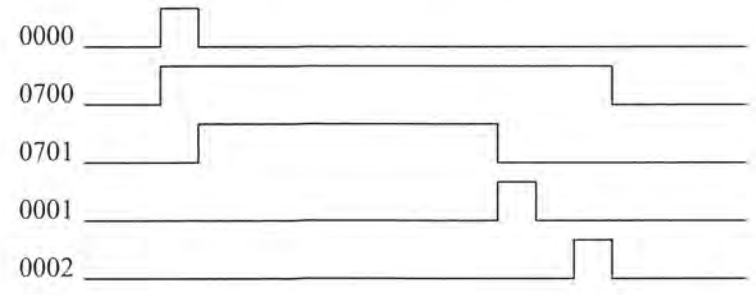



**รูปที่ 3.1** แสดงการใช้คำสั่ง DIFU/DIFD

จากตัวอย่างจะพบว่าการใช้คำสั่ง DIFU และ DIFD จะทำให้รีเลย์ 0700 และรีเลย์ 0701 มีสภาวะเป็น 1 ที่ช่วงเวลาต่างกัน คือ DIFU จะให้รีเลย์ 0700 มีสภาวะเป็น 1 ช่วง 1 SCANTIME ในช่วงเวลาที่ขอบขาขึ้นของการเป็นสภาวะ 1 ของรีเลย์ 0000 และ DIFD จะให้รีเลย์ 0701 มีสภาวะเป็น 1 SCANTIME ในช่วงที่ขอบขาลงของการเป็นสภาวะ 1 ของรีเลย์ 0000

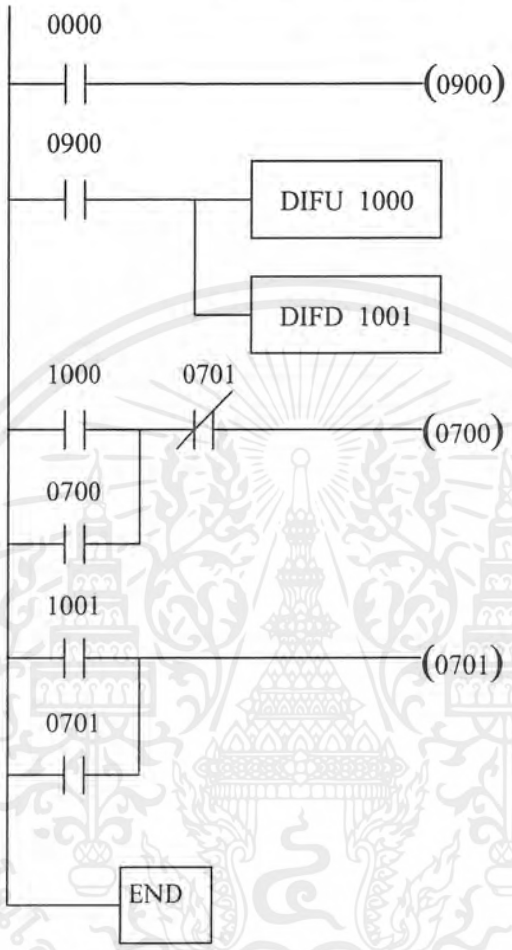
## ขั้นตอนการทดลอง

**1.จงเขียนโปรแกรมตามไดอะแกรมเวลาต่อไปนี้โดยใช้คำสั่ง DIFU/DIFD พร้อมกับสรุปผลการทดลอง**



**2.จงทดสอบโปรแกรมต่อไปนี้**

**LADDER DIAGRAM**




**BOOLEAN**

LD 0000

OUT 0900

LD 0900

DIFU 1000

DIFD 1001

LD 1000

OR 0700

AND NOT 0701

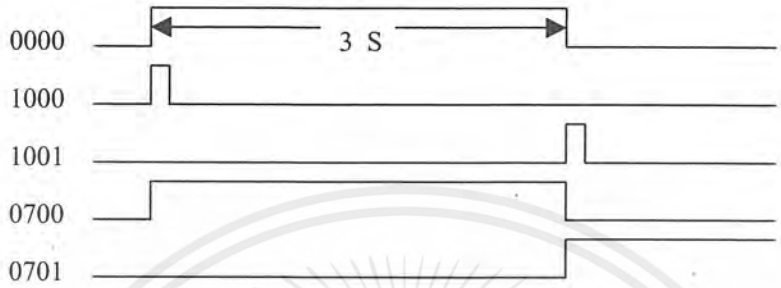
OUT 0700

LD 1001

OR 0701

OUT 0701

END




การทดลองให้เป็นสภาวะ "ON" กับรีเลย์ 0000 ช่างเวลาหนึ่งตาม ไดอะแกรมเวลา (ประมาณ 3 วินาที หรือมากกว่า) สังเกตผลทางรีเลย์ 0700 และ 0701 หลังจากนั้นให้ "OFF" รีเลย์ 0000 สังเกตรีเลย์ 0700 และ 0701 พร้อมสรุปผลการทดลอง

**3.จงอธิบายหลักการทำงานของคำสั่ง DIFU/DIFD พร้อมยกตัวอย่างการนำไปใช้งาน**

**4. จงอธิบายถึงประโยชน์ของการหาขอบขาขึ้นและการหาขอบขาลงของพัลส์**

# ใบงานที่ 4
# ใช้คำสั่ง TIMER

## วัตถุประสงค์

1.เพื่อให้เข้าใจการใช้คำสั่ง TIM

2. เพื่อนำคำสั่ง TIM ไปประยุกต์ใช้งานได้

## เครื่องมือและอุปกรณ์

1.ชุดฝึก PLC 1 เครื่อง

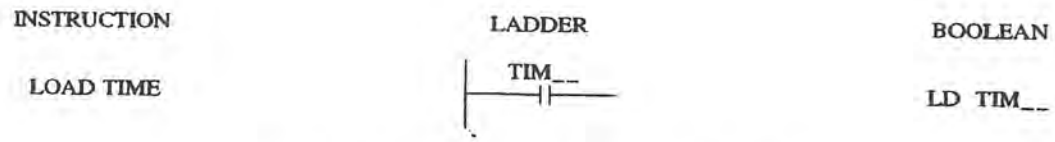
## ทฤษฎีเบื้องต้น

คำสั่ง TIM เป็นคำสั่งที่ใช้เป็นตัวตั้งเวลามีด้วยกัน 48 ตำแหน่ง ใน ET - BOARD v5 PLC MODE โดยเริ่มจาก TIM00 ถึง TIM 47 ซึ่งมีหน่วยการนับเวลาใน 1 หน่วยเท่ากับ 100 ms โดยสามารถกำหนดด้วยตัวเลข 0000 ถึง 9999 ดังนั้นจะเท่ากับ 000.0 ถึง 999.9 วินาที TIM จะทำงานก็ต่อเมื่อสภาวะที่ให้กับ TIM มีค่าสภาวะ "ON" ก็จะเริ่มนับโดยการนับลงจากค่าที่กำหนด จนค่านั้นมีค่าเป็นศูนย์ TIM ก็จะมีสภาวะ "ON" แต่ถ้ายังนับเวลาไม่ครบสภาวะอินพุทที่ให้กับ TIM มีสภาวะ "OFF" ก็จะเป็นการ RESET TIM ทำให้ค่าที่นับอยู่กลับมาสู่ค่าที่กำหนดไว้ในครั้งแรก และสภาวะจะไม่ "ON" ถ้าผลเวลาที่นับไม่เท่ากับศูนย์

## คำสั่งที่ใช้ใน TIMER

**1. LD TIM__**

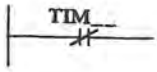
เป็นคำสั่งที่นำค่าสภาวะของตัวตั้งเวลาเข้ามา




**รูปที่ 4.1** แสดงโครงสร้างของคำสั่ง LD TIM

2. **LD NOT TIM _**

เป็นคำสั่งที่นำค่าสภาวะของตัวตั้งเวลาแล้วกลับสภาวะนั้นเป็นตรงข้าม

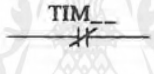
| INSTRUCTION | LADDER | BOOLEAN |
|---|---|---|
| LOAD NOT TIME |  | LD NOT TIM__ |



**รูปที่ 4.2** แสดงโครงสร้างของคำสั่ง LD NOT TIM__

3. **AND TIM –**

เป็นคำสั่งที่นำค่าสภาวะของอุปกรณ์ AND เข้ากับสภาวะของตัวตั้งเวลา

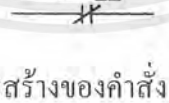
| INSTRUCTION | LADDER | BOOLEAN |
|---|---|---|
| AND NOT TIM |  | AND NOT TIM__ |



**รูปที่ 4.3** แสดงโครงสร้างของคำสั่ง AND TIM

4. **AND NOT TIM –**

เป็นคำสั่งที่นำค่าสภาวะตรงข้ามของตัวตั้งเวลากระทำลอจิก AND กับสภาวะของอุปกรณ์

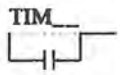
| INSTRUCTION | LADDER | BOOLEAN |
|---|---|---|
| AND NOT TIM |  | AND NOT TIM__ |



**รูปที่ 4.4** แสดงโครงสร้างของคำสั่ง AND NOT TIM

**5. OR TIM_**

เป็นการนำค่าสภาวะของอุปกรณ์ OR เข้ากับสภาวะของตัวตั้งเวลา

| INSTRUCTION | LADDER | BOOLEAN |
|---|---|---|
| OR TIM |  | OR TIM__ |



**รูปที่ 4.5** แสดงโครงสร้างของคำสั่ง OR TIM

**6. OR NOT TIM**

เป็นการนำค่าสภาวะตรงข้ามของตัวตั้งเวลา OR เข้ากับสภาวะของอุปกรณ์

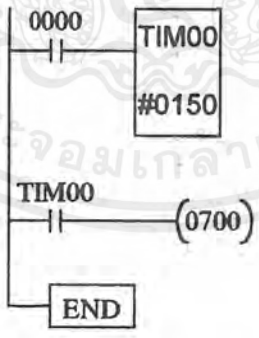
| INSTRUCTION | LADDER | BOOLEAN |
|---|---|---|
| OR NOT TIM |  | OR NOT TIM__ |



**รูปที่ 4.6** แสดงโครงสร้างของคำสั่ง OR NOT TIM

ตัวอย่าง

**LADDER DIAGRAM**




**รูปที่ 4.7** แสดงตัวอย่างการใช้คำสั่ง TIM

**BOOLEAN**

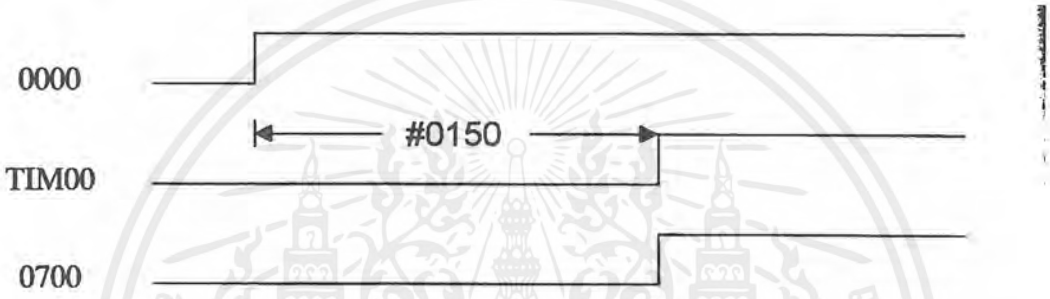
LD 0000

TIM 00

#0150

```
LD TIM 00
OUT 0700
END
```

ตัวอย่างเป็นการแสดงการใช้คำสั่ง TIM _ โดยเมื่อรีเลย์ 0000 มีสภาวะเป็น 1 จะทำให้ TIM00 เริ่มนับจนครบ 150 ค่า แล้วทำให้ TIM00 มีสภาวะเป็น 1 ค้างไว้ ทำให้รีเลย์ 0700 มีสภาวะเป็น 1 ด้วย



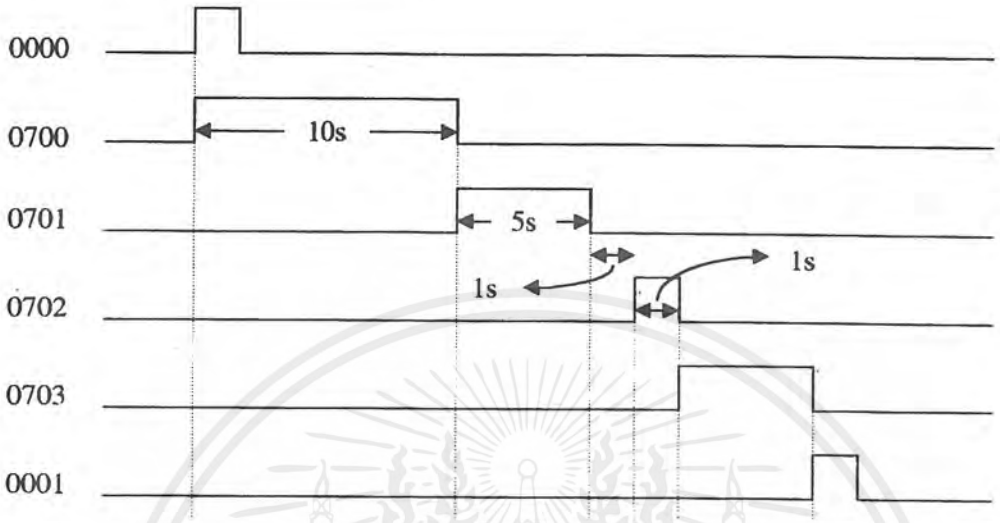

**รูปที่ 4.8** แสดงไดอะแกรมเวลาของตัวอย่างข้างต้น

## ลำดับขั้นการทดลอง

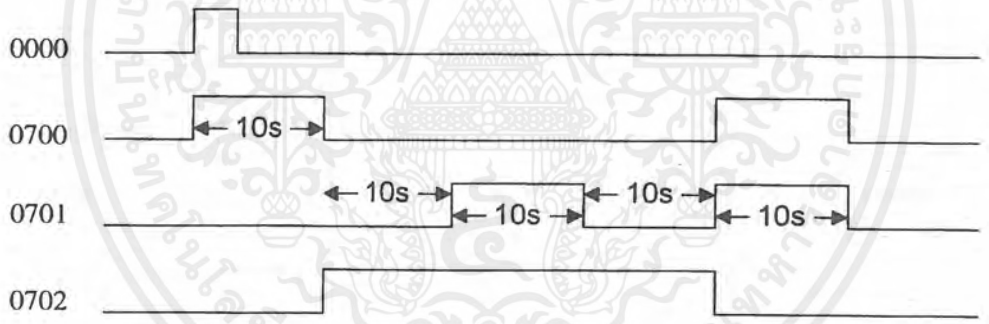
1. การคำนวณหาค่าของ DATA ที่จะกำหนดใน TIMER ตามค่าเวลาต่อไปนี้

| | |
|---|---|
| 1.1 | 1 s |
| 1.2 | 10 s |
| 1.3 | 50 s |
| 1.4 | 100 s |
| 1.5 | 500 s |

2. จงเขียนโปรแกรมตามไดอะแกรมเวลาต่อไปนี้ พร้อมสรุปผลการทดลอง




3. จงเขียนโปรแกรมตามไดอะแกรมเวลาต่อไปนี้ พร้อมสรุปผลการทดลอง




4. จงอธิบายหลักการทำงานของคำสั่ง TIM

5. จงยกตัวอย่างการประยุกต์ใช้งานคำสั่ง TIM

# ใบงานที่ 5
# การใช้คำสั่ง CNT/CNTR

## วัตถุประสงค์

1.เพื่อให้เข้าใจการทำงานของคำสั่ง CNT/CNTR

2.เพื่อประยุกต์ใช้งานคำสั่ง CNT/CNTR ได้

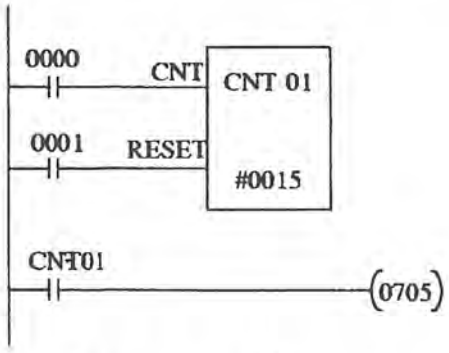
## เครื่องมืออุปกรณ์

1.ชุดฝึก PLC 1 ชุด

## ทฤษฎีเบื้องต้น

คำสั่ง CNT เป็นตัวนับแบบนับลง (COUNT DOWN) โดยจะมีอินพุทในการรับสัญญาณอยู่ 2 ขา คือ ขา COUNT และขา RESET ซึ่งจะนับค่าได้จาก 0000 ถึง 9999 โดยถ้ามีสัญญาณที่ขา CNT (COUNT) เปลี่ยนจากสภาวะ "OFF" มาเป็นสภาวะ "ON" , COUNTER จะทำการนับค่าลง 1 ค่า และถ้าค่าการนับมีค่าเป็นศูนย์ จะทำให้ CNT มีสภาวะ "ON" และถ้าขา RESET มีค่าสภาวะ "ON" จะทำให้ CNT โหลดค่าการนับที่กำหนดไว้คืนมา ซึ่งมีด้วยกัน 48 ตำแหน่ง จาก CNT00 ถึง CNT47 แต่ในส่วนของ CNT นี้ยังรวมเข้ากับตัวนับแบบนับขึ้นนับลงด้วย มีชื่อเรียกว่า CNTR ซึ่งเป็นตำแหน่งเดียวกันกับ CNT00 ถึง CNT47

ตัวอย่าง 1

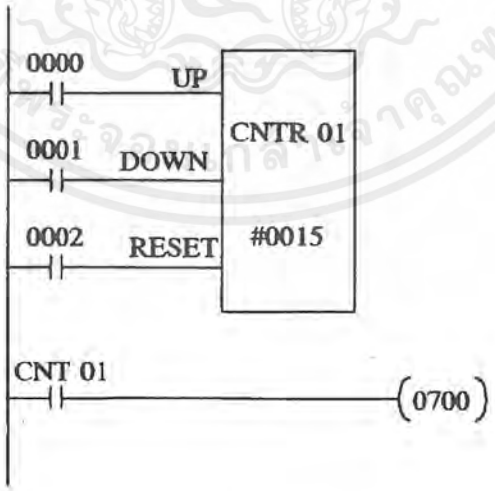



รูปที่ 5.1 แสดงการใช้คำสั่ง CNT

จากตัวอย่างข้างต้นจะเป็นการแสดงให้เห็นถึงการนำคำสั่ง CNT มาใช้คือ relay 0000 จะทำหน้าที่กำเนิด clock โดยเมื่อ relay เปลี่ยนสภาวะจาก "OFF" มาเป็น "ON" จะทำให้ CNT นับลง 1 ค่า จาก 15 ที่ตั้งไว้ ลงมาที่ 14 ซึ่งเมื่อค่าการนับลงมาที่ 0 จะทำให้ contrac ของ CNT01 มีสภาวะ "ON" ซึ่งส่งผลให้ relay 0705 มีสภาวะ "ON" ตามไปด้วย

คำสั่ง CNTR เป็นตัวนับแบบนับขึ้นและนับลงมีขาในการใช้งานด้วยกัน 3 สัญญาณ UP ใช้ในการนับขึ้น, ขาสัญญาณ DOWN ใช้ในการนับลง และขาสัญญาณ RESET ใช้ในการ CLEAR ค่าการนับให้เป็นศูนย์ และค่าสภาวะของ COUN เป็น "OFF" ด้วย มีด้วยกัน 48 ตำแหน่ง เป็นที่ ๆ เดียวกับ CNT คือ CNTR00 ถึง CNTR47 และในการเรียกสภาวะ COUNTER นี้มาใช้งานจะเรียกผ่านโดย CNT00 ถึง CNT47 ถ้ามีสัญญาณที่ขา UP ก็จะเป็นการนับขึ้นส่วนถ้าเป็นขา DOWN ก็จะเป็นการนับลงลักษณะของสัญญาณของ UP และ DOWN สัญญาณจะเกิดการนับก็ต่อเมื่อสัญญาณที่เข้ามาเปลี่ยนจาก "OFF" เป็น "ON" ในการนับจะเป็นลักษณะวงแหวนต่อเนื่องกันไปขา UP จะเริ่มนับจาก 0 จนถึงค่าเป้าหมายที่กำหนด และถ้ามีสัญญาณที่ขา UP อีกก็จะเปลี่ยนจากค่าเป้าหมายมามีค่าศูนย์ ซึ่งทำให้ COUNTER มีสภาวะ "ON" ในการเรียกค่าสภาวะของ CNTR มาใช้งานจะเรียกโดยใช้คำสั่ง LD CNT แทน และถ้าขาสัญญาณที่ขา UP และ DOWN เกิดมีสภาวะ "ON" พร้อมกับค่าสภาวะต่าง ๆ ของ CNTR จะไม่มีการเปลี่ยนแปลง

ตัวอย่าง 2

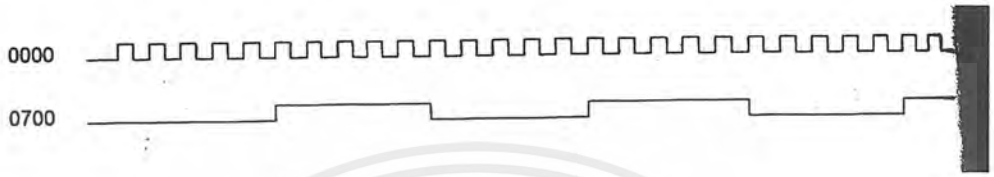



รูปที่ 5.2 แสดงการใช้คำสั่ง CNTR

## ลำดับขั้นการทดลอง

**1. จงเขียนโปรแกรมตามไดอะแกรมเวลาต่อไปนี้**

1.1 ป้อน clock ที่ 0000 และ 0001 เป็น reset โดยพิจารณา relay output ตามไดอะแกรมเวลา




**2. จงเขียนโปรแกรมนับค่า 0 ถึง 40 โดย**

2.1 clock input ทาง relay 0000

2.2 ให้ค่า relay output 0700 ถึง 0707 แสดงสภาวะ "ON" เมื่อ clock input นับค่าถึง 5, 10, 15, 20, 25, 30, 35, 40 โดยเรียงลำดับจาก 0700 ถึง 0707

**3. จงอธิบายการทำงานของคำสั่ง CNT/CNTR**

4. สรุปผลการใช้คำสั่ง CNT/CNTR พร้อมยกตัวอย่างการนำไปใช้งาน

# ใบงานที่ 6
# การใช้คำสั่ง SFT

## วัตถุประสงค์

1.เพื่อให้เข้าใจการทำงานของคำสั่ง SFT

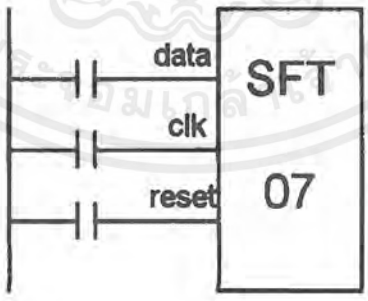
2.เพื่อประยุกต์ใช้งานได้

## เครื่องมือและอุปกรณ์

1.ชุดฝึก PLC 1 ชุด

## ทฤษฎีเบื้องต้น

คำสั่ง SFT จะใช้สำหรับการเลื่อนข้อมูลของเอาต์พุทหรือรีเลย์ภายในซึ่งจะกระทำในลักษณะไบต์ (8 bit) โดยการนำข้อมูลที่ขา data เลื่อนเข้าไปที่ bit 0 จากนั้นบิท 0 จะถูกเลื่อนจากบิท 0 ไปหาบิท 7 โดยการเลื่อนแต่ละครั้งจะถูกควบคุมด้วยขาสัญญาณ clk เมื่อขานี้ถูกเปลี่ยนจากสภาวะ "OFF" เป็นสภาวะ "ON" และขา RESET จะใช้ในการ CLEAR ค่าเอาต์พุททั้ง 8 bit ให้มีสภาวะ "OFF" ตำแหน่งที่ใช้ในการ SHIFT จะอ้างได้ตั้งแต่ 07-27




**รูปที่ 6.1** แสดงโครงสร้าง SFT

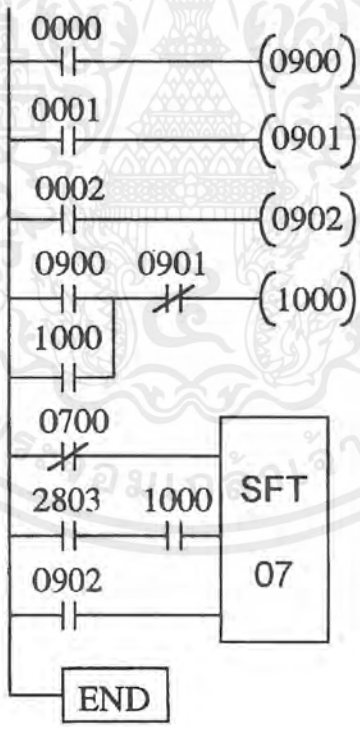
**ตัวอย่าง**

ตัวอย่างต่อไปนี้จะเป็นการนำคำสั่ง SFT มาใช้ทำไฟกระพริบ โดยลักษณะการกระพริบจะแสดงได้ดังรูปที่ 6.2 โดยโปรแกรมสามารถแสดงได้ดังรูปที่ 6.3 ซึ่งดวงไฟจะเลื่อนด้วย

ความเร็ว 1 วินาที ซึ่งประเด็นสำคัญจะอยู่ที่ การนำ relay output มาใช้เป็นค่า data และ relay ที่กำเนิด clk คือ relay 2803


รูปที่ 6.2 แสดงการกระพริบของหลอดไฟ

**LADDER DIAGRAM**




**BOOLEAN**

| | |
|---|---|
| 0000 start | LD 0000 |
| 0001 stop | OUT 0900 |
| 0002 reset | LD 0001 |

```
OUT 0901
LD 0002
OUT 0902
LD 0900
OR 1000
AND NOT 0901
OUT 1000
LD NOT 0700
LD 2803
AND 1000
LD 0902
SFT 07
END
```

รูปที่ 6.3 แสดงโปรแกรมการใช้คำสั่ง SFT มาทำเป็นไฟกระพริบ

## ลำดับขั้นการทดลอง

จงเขียนโปรแกรมไฟกระพริบโดยให้มีการเลื่อนดังรูป และเลื่อนด้วยความเร็ว 1 วินาที โดยเงื่อนไขกำหนดว่า

**start** ไฟเริ่มกระพริบ

**stop** ค้างสภาวะไว้

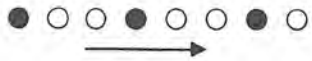
**reset** ไฟทุกดวงดับ

**1.กำหนดให้** 0000 start

0001 stop

0002 reset

0700-0707 output



● ON
○ OFF

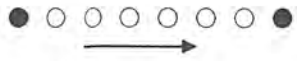
2. กำหนดให้ 0000 start
0001 stop
0002 reset
0700-0707 output

● ON
○ OFF

3. กำหนดให้ 0000 start
0001 stop
0002 reset
0700-0707 output

● ON
○ OFF

4. กำหนดให้ 0000 start
0001 stop
0002 reset
0700-0707 output



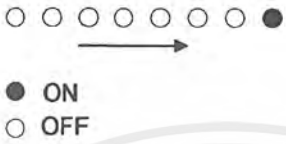
● ON
○ OFF

**5. กำหนดให้** 0000 start

0001 stop

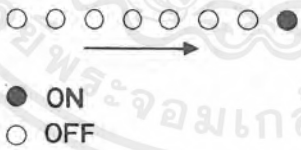
0002 reset

0700-0707 output




**6. กำหนดให้ สามารถเลือกความเร็วของการกระพริบได้**

| | |
|---|---|
| 0000 | 10 ms |
| 0001 | 10 ms |
| 0002 | 500 ms |
| 0003 | 1 s |
| 0004 | start |
| 0005 | stop |
| 0006 | reset |
| 700-707 | output |




**7. สรุปการใช้คำสั่ง SFT**

# ใบงานที่ 7
# การใช้คำสั่ง JMP/JME

## วัตถุประสงค์

1.เพื่อให้เข้าใจการทำงานของคำสั่ง JMP/JME

2.เพื่อประยุกต์ใช้คำสั่ง JMP/JME

## เครื่องมือและอุปกรณ์

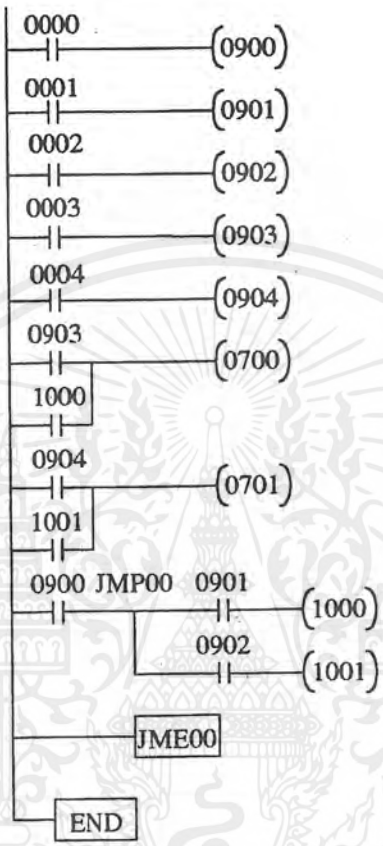
1.ชุดฝึก PLC 1 ชุด

## ทฤษฎีเบื้องต้น

คำสั่ง JMP/JME เป็นคำสั่งสำหรับข้ามการทำงานของบล็อก ซึ่งจะใช้คู่กับคำสั่ง JME ในกรณีชุด CONTRAC ตรงส่วนหน้าของ JMP มีสภาวะเป็น "ON" จะทำให้โปรแกรมที่อยู่ระหว่าง JMP กับ JME ถูกทำงานเป็นลำดับตามปรกติ แต่ถ้า CONTRAC ตรงส่วนหน้าของ JMP มีสภาวะเป็น "OFF" จะทำให้ลำดับการทำงานของโปรแกรม ข้ามชุดคำสั่งที่อยู่ระหว่าง JMP และ JME นั่นก็คือสภาวะใด ๆ ที่อยู่ในส่วนนี้ก็คงสภาพไม่มีการเปลี่ยนแปลง โดยจะไปทำคำสั่งใน LINE ถัดจากคำสั่ง JME ซึ่งคำสั่งนี้มีด้วยกัน 8 JMP คือ JMP00 ถึง JMP07 โดยแต่ละ JMP จะเรียกใช้กี่ครั้งก็ได้แต่ต้องจบด้วย JME ของ JMP นั้น ๆ เสมอ

ตัวอย่าง

**LADDER DIAGRAM**




**BOOLEAN**

LD 0000

OUT 0900

LD 0001

OUT 0901

LD 0002

OUT 0902

LD 0003

OUT 0903

LD 0004

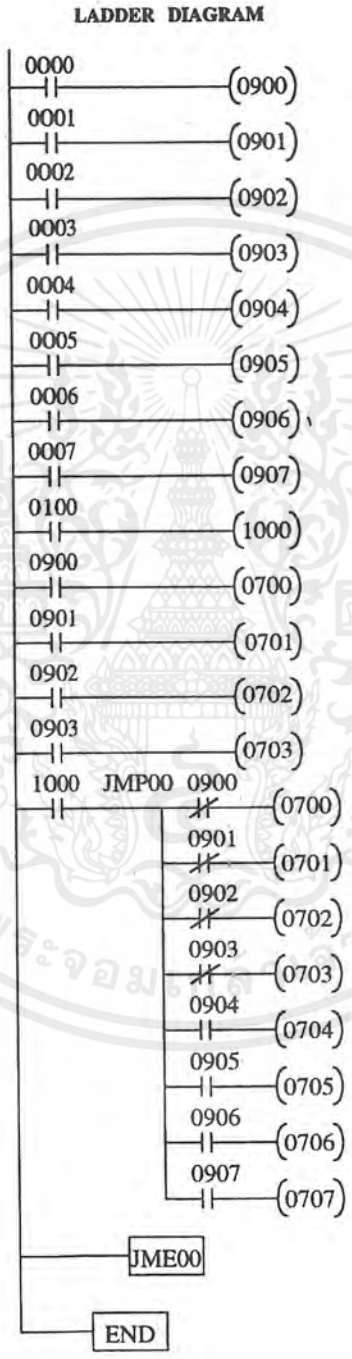
OUT 0904

```
LD 0903
OR 1000
OUT 0700
LD 0904
OR 1001
OUT 0701
LD 0900
JMP00
LD 0901
OUT 1000
LD 0902
OUT 1001
JME00
END
```

จากตัวอย่างข้างต้น เป็นการแสดงให้เห็นถึงการทำงานของคำสั่ง JMP/JME โดยคำสั่ง JMP/JME ถูกควบคุมการทำงานด้วย relay 0900 โดยถ้า relay 0900 มีสภาวะ "ON" จะทำให้ชุด relay ภายในบล็อกของคำสั่ง JMP/JME เกิดการทำงานตามสภาวะปรกติ คือ เมื่อ relay 0901 "ON" จะทำให้ relay 0700 "ON" ตามไปด้วย หรือเมื่อ relay 0902 "ON" ก็ทำให้ relay 0701 "ON" ตามไปด้วยเช่นกัน หรือในทางตรงข้าม ถ้า relay 0901 หรือ 0902 เกิด "OFF" จะทำให้ relay 0700 และ 0701 เกิด "OFF" ตามลำดับ แต่ถ้า contrac 0900 มีสภาวะ "OFF" ก็จะทำให้ชุด contrac ที่อยู่ภายในบล็อกของคำสั่ง JMP/JME ยังคงอยู่ในสภาวะเดิม คือไม่ว่า relay 0901 หรือ 0902 จะเปลี่ยนแปลงอย่างไร ก็ไม่ทำให้ relay 0700 และ 0701 เกิดการเปลี่ยนแปลง

## ลำดับขั้นการทดลอง

**1.จงทดสอบโปรแกรมต่อไปนี้**



**BOOLEAN**

LD 0000

OUT 0900

LD 0001

OUT 0901

LD 0002

OUT 0902

LD 0003

OUT 0903

LD 0004

OUT 0904

LD 0005

OUT 0905

LD 0006

OUT 0906

LD 0007

OUT 0907

LD 0100

OUT 1000

LD 0900

OUT 0700

LD 0901

OUT 0701

LD 0902

OUT 0702

LD 0903

OUT 0703

LD 1000

JMP00

LD NOT 0900

OUT 0700

LD NOT 0901

OUT 0701

LD NOT 0902

OUT 0702

LD NOT 0903

OUT 0703

LD 0904

OUT 0704

LD 0905

OUT 0705

LD 0906

OUT 0706

LD 0907

OUT 0707

JME

END

1.1 ทำการป้อนสภาวะของ input 0000 ถึง 0700 แล้ว run program โดยเปลี่ยนสภาวะของ relay 0100 เป็น "OFF" พิจารณาเอาต์พุท 0700 ถึง 0707 และเปลี่ยนสภาวะของ relay 0100 เป็น "ON" พิจารณาเอาต์พุท 0700 ถึง 0707

**2. จงเขียนโปรแกรมกำหนดสภาวะของ relay โดยใช้คำสั่ง JMP/JME โดยเมื่อ 0000 "ON" ให้สภาวะเอาต์พุทถูกควบคุมด้วย 0001 โดยใช้ 0001 เป็นอุปกรณ์กำเนิด clock เมื่อเริ่ม clock ลูกที่ 1 ให้แสดงสภาวะ "ON" ที่ 0700 และ clock ลูกต่อไปให้แสดงสภาวะ "ON" ที่เอาต์พุทถัดไป แต่เอาต์พุทที่แสดงก่อนหน้าจะต้อง "OFF" และกำหนดให้ relay 0002 มีหน้าที่เป็น reset และทำให้สภาวะของ relay 0000 "OFF" โดยถ้า relay 0000 "OFF" หรือมีการ reset ให้สภาวะเอาต์พุทค้างไว้**

**3.อธิบายการทำงานของคำสั่ง JMP/JME อย่างละเอียด**

**4. สรุปผลการใช้งานคำสั่ง JMP/JME พร้อมยกตัวอย่างการนำไปประยุกต์ใช้งาน**

# ใบงานที่ 8
# การใช้คำสั่ง MCS/MCR

## วัตถุประสงค์

1.เพื่อให้เข้าใจการทำงานของคำสั่ง MCS/MCR

2.เพื่อประยุกต์ใช้งานคำสั่ง MCS/MCR ได้

## เครื่องมือและอุปกรณ์

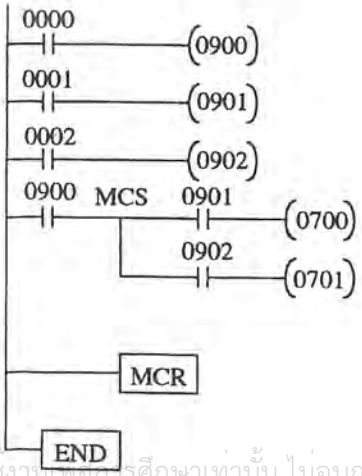
1.ชุดฝึก PLC 1 ชุด

## ทฤษฎีเบื้องต้น

MCS (MASTER CONTROL SFT) เป็นคำสั่งในการควบคุมรีเลย์ หรือส่วนของเอาต์พุทในบล็อกที่กำหนด และการใช้ MCS จะต้องจบด้วย MCR ซึ่งทั้ง 2 คำสั่งนี้จะต้องใช้ร่วมกัน โดยลักษณะของการทำงานจะทำการตรวจสอบ CONTRAC ตรงส่วนของ MCS ถ้ามีสภาวะ "ON" จะทำให้โปรแกรมที่อยู่ระหว่าง MCS ทำงานให้สภาวะเอาท์พุทเป็นไปตามปรกติแต่ถ้า CONTRAC ตำแหน่งดังกล่าวมีสภาวะ "OFF" ผลการทำงานในบล็อกก็ยังทำงานเช่นเดิมแต่สภาวะเอาต์พุทของส่วนเอาต์พุทหรือรีเลย์ภายในจะมีสภาวะ "OFF" ถึงแม้คำสั่งเงื่อนไขต่าง ๆ ในบล็อกจะมีสภาวะ "ON" ก็ตาม

<u>ตัวอย่าง</u>

**LADDER DIAGRAM**



**BOOLEAN**

LD 0000

OUT 0900

LD 0001

OUT 0901

LD 0002

OUT 0902

LD 0900

MCS

LD 0901

OUT 0700

LD 0902

OUT 0701

MCR

END

จากตัวอย่างข้างต้น จะเป็นการแสดงให้เห็นถึงการทำงานของคำสั่ง MCS/MCR โดยในการทำงานของคำสั่งคือ เมื่อรีเลย์ 0900 มีสภาวะ "ON" จะทำให้รีเลย์ 0700 ถูกควบคุมด้วยรีเลย์ 0901 และรีเลย์ 0701 ถูกควบคุมด้วยรีเลย์ 0902 แต่เมื่อรีเลย์ 0900 อยู่ในสภาวะ "OFF" จะทำให้รีเลย์ 0700 และ 0701 มีสภาวะ "OFF" ตลอดเวลาไม่ว่ารีเลย์ 0901 และรีเลย์ 0902 จะมีสภาวะใด ๆ ก็ตาม สามารถทราบผลการทำงานของตัวอย่างได้จากตารางที่ 8.1

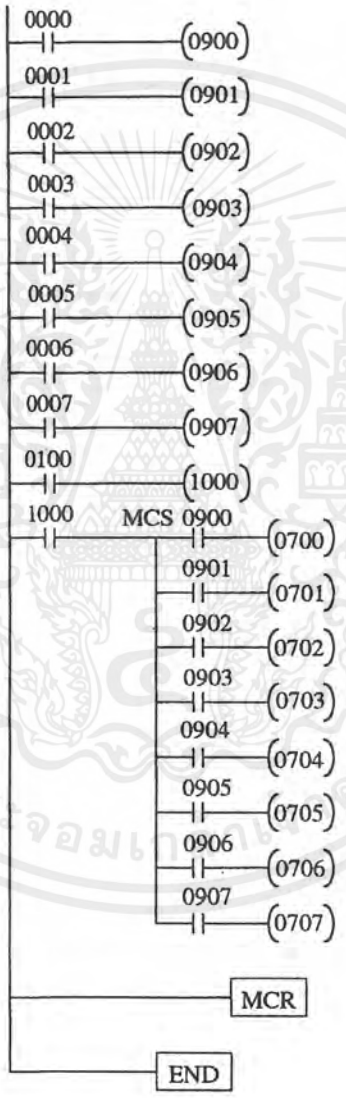
| 0000 | 0001 | 0002 | 0900 | 0901 | 0902 | 0700 | 0701 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |

**ตารางที่ 8.1** แสดงผลของตัวอย่าง

## ลำดับการทดลอง

1. จงทดลองโปรแกรมต่อไปนี้

**LADDER DIAGRAM**



**BOOLEAN**

LD 0000

OUT 0900

LD 0001

OUT 0901

LD 0002

OUT 0902

LD 0003

OUT 0903

LD 0004

OUT 0904

LD 0005

OUT 0905

LD 0006

OUT 0906

LD 0007

OUT 0907

LD 0100

OUT 1000

LD 1000

MCS

LD 0900

OUT 0700

LD 0901

OUT 0701

LD 0902

OUT 0702

LD 0903

OUT 0703

LD 0904

OUT 0704

LD 0705

OUT 0705

LD 0906

OUT 0706

LD 0907

OUT 0707

MCR

END

ป้อนสภาวะทางรีเลย์อินพุท 0000 ถึง 0707 และ 0100 ตามตาราง และสังเกตค่ารีเลย์เอาต์พุท 0700-0707 พร้อมบันทึกผลการทดลองลงในตาราง

| 0100 | 0000 | 0001 | 0002 | 0003 | 0004 | 0005 | 0006 | 0007 | 0700 | 0701 | 0702 | 0703 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | | |

| 0100 | 0000 | 0001 | 0002 | 0003 | 0004 | 0005 | 0006 | 0007 | 0704 | 0705 | 0706 | 0707 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | | | |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | | |

สรุปผลการทดลอง

2. ทดลองเขียนโปรแกรมรับค่า relay 0000 โดยถ้า relay 0000 "ON" ให้ relay output 0704-0707 ถูกควบคุมสภาวะด้วย relay 0004-0007 ตามลำดับ แต่ถ้า relay 0000 "OFF" ให้ relay 0704-0707 อยู่ในสภาวะ "OFF" ทั้งหมด

4. อธิบายการทำงานของคำสั่ง MCS/MCR อย่างละเอียด

5. สรุปผลการใช้งานคำสั่ง MCS/MCR

# ใบงานที่ 9

# RELAY DIAGRAM TO LADDER DIAGRAM

## วัตถุประสงค์

1.เพื่อให้ทราบความหมายของ relay diagram

2.เพื่อให้สามารถแปลง relay diagram ไปเป็น ladder diagram

## เครื่องมือและอุปกรณ์

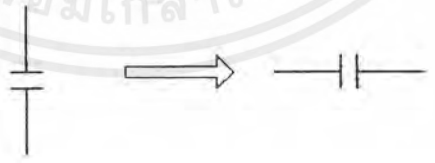
1.ชุดฝึก PLC 1 ชุด

## ทฤษฎีเบื้องต้น

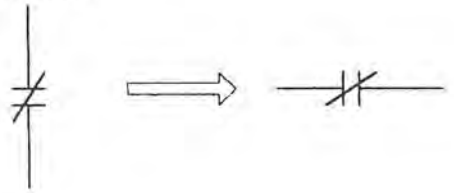
ในระบบการควบคุมแบบเก่าจะใช้แผง relay หรือแผง contractor เป็นแผงที่ใช้ในการควบคุม ซึ่งในปัจจุบันได้มีการเปลี่ยนแปลงการควบคุมมาใช้ PLC เกือบทั้งหมด แต่โดยพื้นฐานของ ladder diagram ของ PLC ก็มาจาก relay diagram ของการควบคุมในระบบไฟฟ้า เพราะฉะนั้นในการควบคุมครั้งนี้จะเป็นการศึกษาถึงความหมายของ สัญลักษณ์ใน relay diagram ที่ตรงกับ ladder diagram

RELAY DIAGRAM LADDER DIAGRAM

NORMAL OPEN

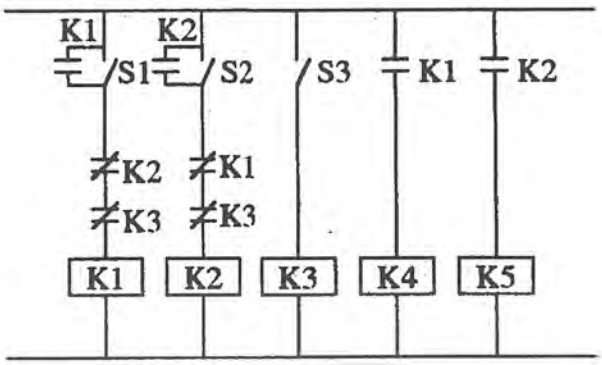


NORMAL CLOSE

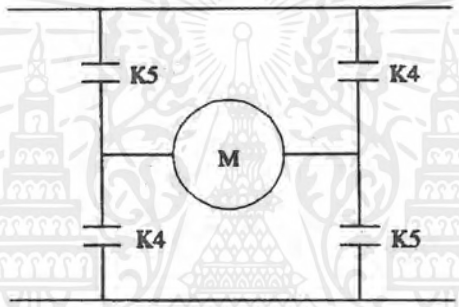



COIL

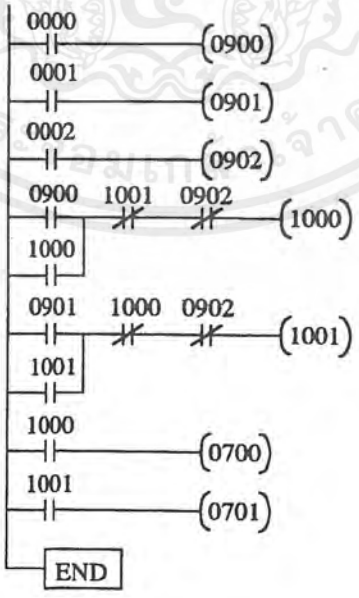
ตัวอย่าง



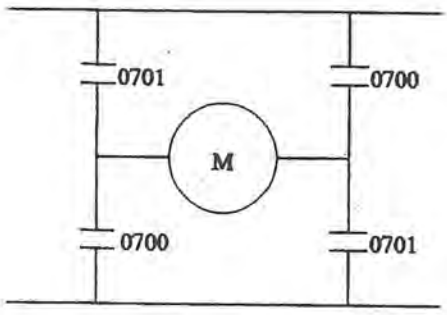

รูปที่ 9.2 (ก) แสดง relay control diagram




รูปที่ 9.2 (ข) แสดง relay power diagram




รูปที่ 9.3 (ก) แสดง ladder diagram




รูปที่ 9.3 (ข) แสดง power diagram

จาก relay diagram และ ladder diagram ในตัวอย่างข้างต้นซึ่งจะพบว่าสามารถที่จะแทนความหมายของการควบคุมจาก relay diagram เป็น ladder diagram โดยในวงจรทั้งหมดของรูปที่ 9.2 (ก) จะเป็นการแสดงแผงวงจรควบคุมของ relay diagram และในรูปที่ 9.2 (ข) จะเป็น power diagram ของการควบคุม จากแผง relay diagram ในรูป 9.2 (ก) โดยในตัวอย่างเป็นการควบคุม dc motor ให้สามารถหมุนซ้ายหมุนขวาและหยุดได้ ด้วยการกด switch คือ s1 จะควบคุมการหมุนขวา , s2 ควบคุมการหมุนซ้าย , s3 ควบคุมการหยุด โดยแผงควบคุมในรูปที่ 9.2 (ก) เมื่อกด s1 จะทำให้กระแสไหลผ่าน coil ของ relay K1 ทำให้ coil ของ relay K1 มีกระแสไฟฟ้าซึ่งทำ motor ที่ต่อในรูปที่ 9.2 (ข) หมุนไปทางขวา แต่ถ้ากด s2 ciol ของ relay K2 ก็จะมีกระแสไหลซึ่งส่งผลให้ contrac K2 close (ON) จึงทำให้ coil ชอง relay K5 มีกระแสไหล ทำให้ contrac K5 close (ON) ซึ่งก็ทำให้ motor หมุนซ้าย แต่ถ้ามีการกด s3 ทำให้ contrac K3 close ซึ่งก็จะทำให้ทั้ง K1 และ K2 ไม่สามารถ close (ON) ได้ เนื่องจาก contrac ของ K3 ทำการตัดวงจรทำให้ dc motor หยุดหมุน ซึ่งในวงจรจะมีการ interlock โดยเมื่อมีการหมุนซ้ายจะไม่สามารถหมุนขวาในขณะเดียวกันได้เนื่องจาก contrac ของ K1 และ K2 ทำการตัดวงจรซึ่งกันและกันส่วน ladder diagram ของรูปที่ 9.3 (ก) จะเป็นการแทนความหมายของ relay diagram ในรูปที่ 9.2 (ก) โดยสามารถแทน

s1 ได้เท่ากับ switch ที่ต่ออยู่กับ 0000

s2 ได้เท่ากับ switch ที่ต่ออยู่กับ 0001

s3 ได้เท่ากับ switch ที่ต่ออยู่กับ 0002

และ K1 มีความหมายเหมือนกับ 1000

K2 มีความหมายเหมือนกับ 1001

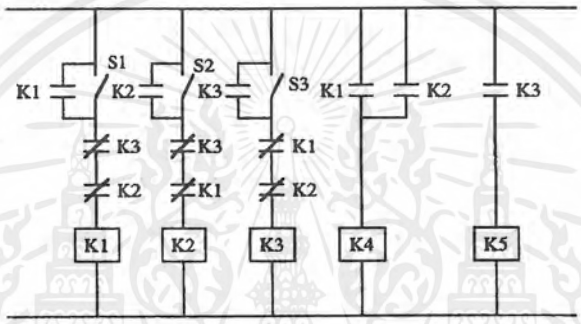
K3 มีความหมายเหมือนกับ 0902

K4 มีความหมายเหมือนกับ 0700

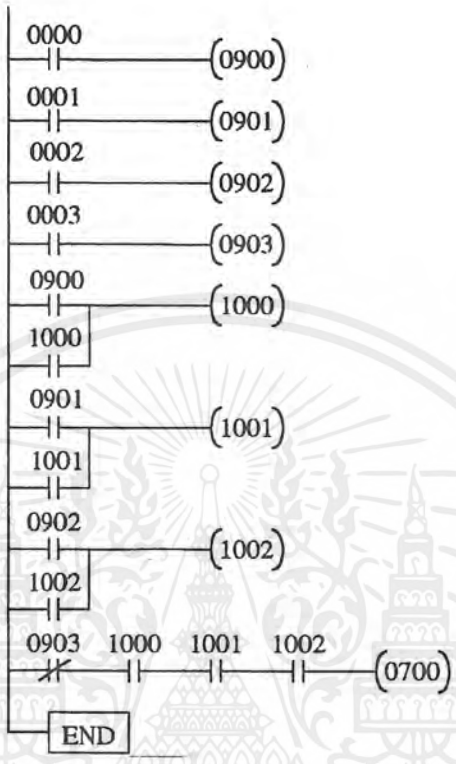
K5 มีความหมายเหมือนกับ 0701

## ลำดับขั้นการทดลอง

**1. จงเปลี่ยน relay diagram ต่อไปนี้ในรูป ladder diagram**



**2. จงแปลง ladder diagram ต่อไปนี้ในรูป relay diagram**




**3. จงออกแบบควบคุม motor 2 ตัว ให้หมุนซ้ายและหมุนขวาได้พร้อม stop โดยมีเงื่อนไขดังนี้**

3.1 ขณะที่ motor 1 หมุนซ้าย motor 2 จะต้องหมุนขวา

3.2 ขณะที่ motor 1 หมุนขวา motor 2 จะต้องหมุนซ้าย

3.3 กด stop จะต้องหยุดพร้อมกัน

โดยออกแบบทั้ง relay diagram และ ladder diagram

# ใบงานที่ 10

# CAR COUNTER

## วัตถุประสงค์

1.เพื่อสามารถนำ PLC ประยุกต์ใช้งานได้

2.สามารถประยุกต์ใช้ PLC ในการนับรถยนต์ได้

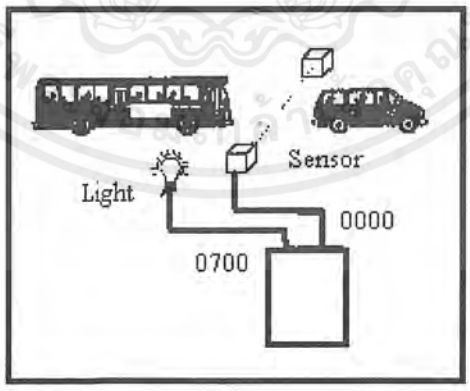
3.สามารถนำคำสั่ง counter มาประยุกต์ใช้งานได้

## เครื่องและอุปกรณ์

1.ชุดฝึก PLC 1 ชุด

## ทฤษฎีเบื้องต้น

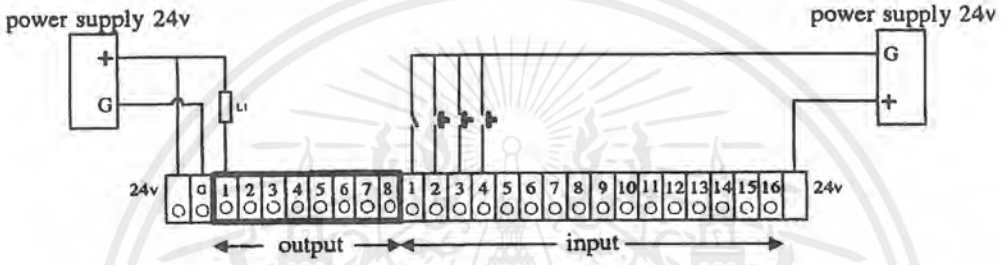
ในการทดลองนี้จะเป็นการประยุกต์ใช้ PLC ในการนับรถยนต์ที่จะเข้าไปจอดในลานจอดรถ โดยสมมุติว่าเป็นการนับรถทางประตูเข้าทางเดียว ซึ่งจะไม่มีรถออกทางประตูนี้ โดยระบบการทำงานจะแสดงได้ในรูปที่ 10.1 ซึ่งมีคุณสมบัติในการนับคือ เมื่อนับถึงค่าที่กำหนดก็จะให้หยุดนับ และมีไฟแสดงออกมาว่านับถึงค่าที่กำหนดแล้ว




**รูปที่ 10.1** แสดงการใช้ PLC ในการนับรถยนต์

กำหนดให้มีอุปกรณ์ภายนอกต่อร่วมกับ relay input และ relay output เพื่อทำหน้าที่ต่าง ๆ ดังต่อไปนี้

| | | |
|---|---|---|
| relay input | 0000 | sensor |
| | 0001 | switch start |
| | 0002 | switch reset |
| | 0003 | switch stop |
| relay output | 0700 | light |

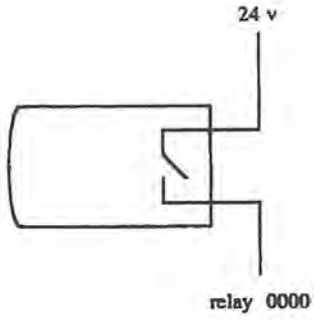



**รูปที่ 10.2** แสดงการต่อ I/O ของ PLC

**คุณสมบัติของอุปกรณ์ประกอบร่วม**

**sensor**

เป็นอุปกรณ์ตรวจจับแบบ on-off ซึ่งการทำงานจะประกอบด้วยส่วนของตัวส่งสัญญาณและส่วนของ ตัวรับสัญญาณโดย ที่ส่วนของตัวรับสัญญาณจะมี relay ประกอบร่วมอยู่ซึ่งโดยปรกติ contrac ของ relay นี้จะ open (off) แต่เมื่อมีวัตถุมาตัดผ่านทางเดินของสัญญาณที่ส่งออกมาจากตัวส่งสัญญาณออกมายังตัวรับสัญญาณของ sensor จะทำให้ relay ที่ sensor ของตัวรับ close (on) โดยในการทดลองนี้สมมุติให้นำ contrac ของ relay ของ sensor ทางด้านตัวรับสัญญาณ มาต่อเข้ากับทางอินพุท 0000 ของ PLC ดังรูปที่ 10.2 ซึ่งโครงสร้างของ contrac ภายในตัว sensor ดังแสดงในรูปที่ 10.3




**รูปที่ 10.3** แสดง contrac ของ sensor

**switch start**

กำหนดให้เป็น switch แบบ กดติด ปล่อยดับ โดยเมื่อกดจะทำให้ระบบเริ่มทำการนับ

**switch reset**

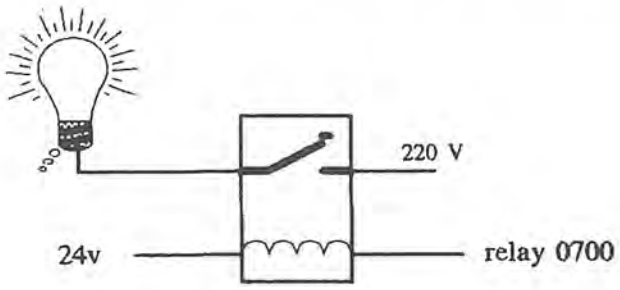
กำหนดให้เป็น switch แบบ กดติด ปล่อยดับ โดยเมื่อมีการกด switch reset จะทำให้ค่าที่นับไว้มีค่าเป็น 0 และรอการกด switch อื่นต่อไป

**switch stop**

กำหนดให้เป็น switch แบบกดติด ปล่อยดับ โดยเมื่อกด switch นี้จะให้ค่าที่นับค้างค่าเดิมไว้และระบบหยุดนับเพื่อรอการกด switch อื่นต่อไป

**light**

จะมีหน้าที่ในการแสดงสภาวะการนับ โดยเมื่อการนับมีการนับถึงค่าที่กำหนดหลอดไฟจะติดสว่างแต่ถ้าการนับยังไม่ถึงค่าที่กำหนดหรือมีการกด switch ใด ๆ หลอดไฟจะอยู่ในสภาวะดับ โดยหลอดไฟจะรับแรงดันไฟฟ้า 22 v ซึ่งการควบคุมการจ่ายไฟให้ หลอดไฟ จะต้องใช้ contrac ของ relay มาควบคุมการตัดต่อ กระแสไฟฟ้า โดยในส่วนของ coil ของ relay ที่จะใช้ควบคุมการตัดต่อกระแสไฟฟ้านั้น จะถูกนำไปต่อกับ relay output 0700 ของ PLC เพื่อให้ relay ควบคุมการทำงานของ coil ของ relay ซึ่งลักษณะการต่อ contrac เพื่อควบคุม หลอดไฟแสดงดังรูปที่ 10.4




**รูปที่ 10.4** แสดงการควบคุมหลอดไฟ

## ลำดับขั้นการทดลอง

**จงออกแบบระบบนับรถยนต์ โดยให้มีการนับรถจำนวน 20 คัน ถ้ามีรถมากกว่านั้นให้หลอดไฟติดสว่าง โดยใช้เงื่อนไขของอุปกรณ์จากทฤษฎีข้างต้น**

**1. จงเขียนไดอะแกรม เวลาของ relay input และ relay output**

0000.....

0001.....

0002.....

0003.....

0700.....

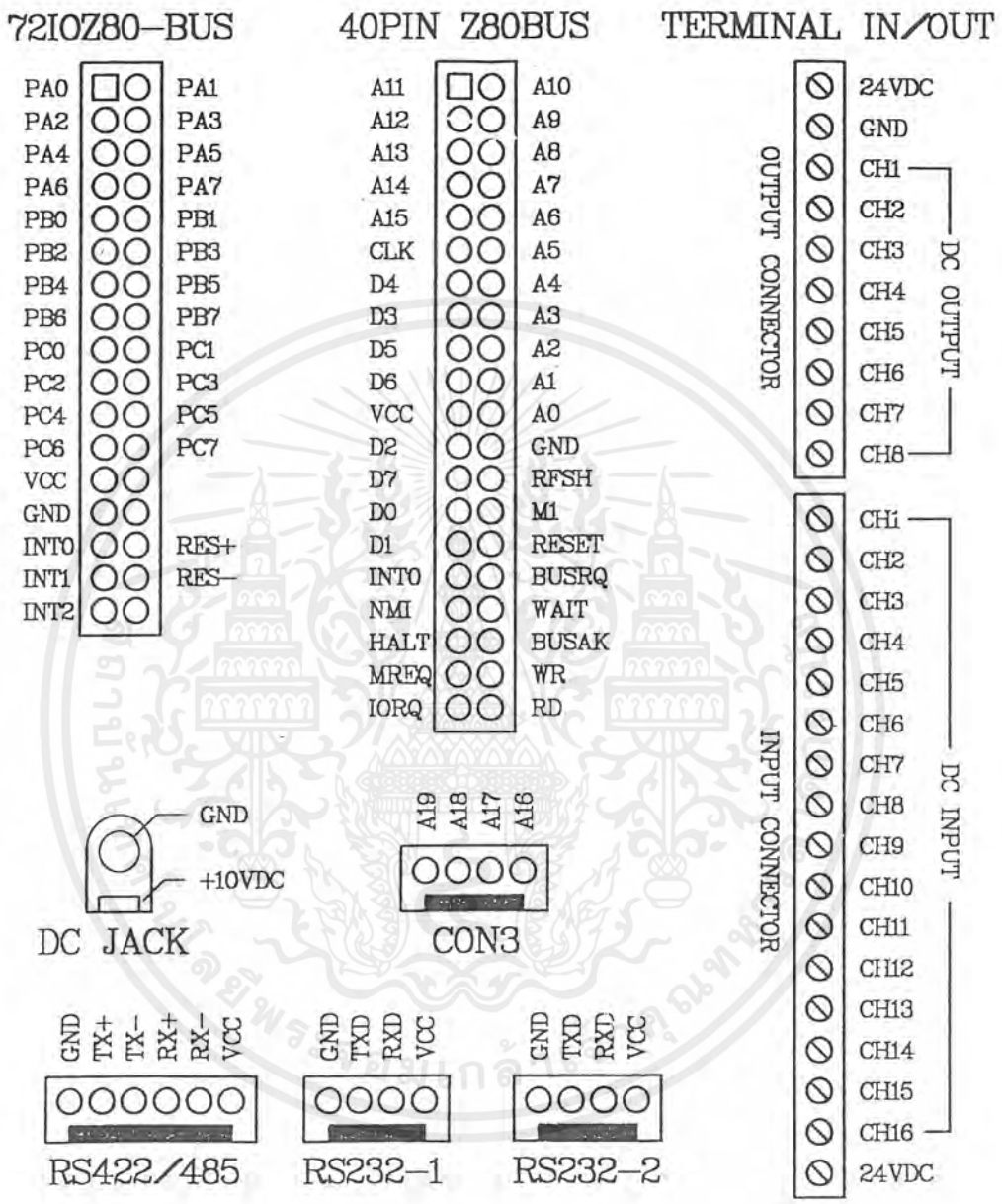
**2.จงเขียน ladder diagram ของการใช้งาน**

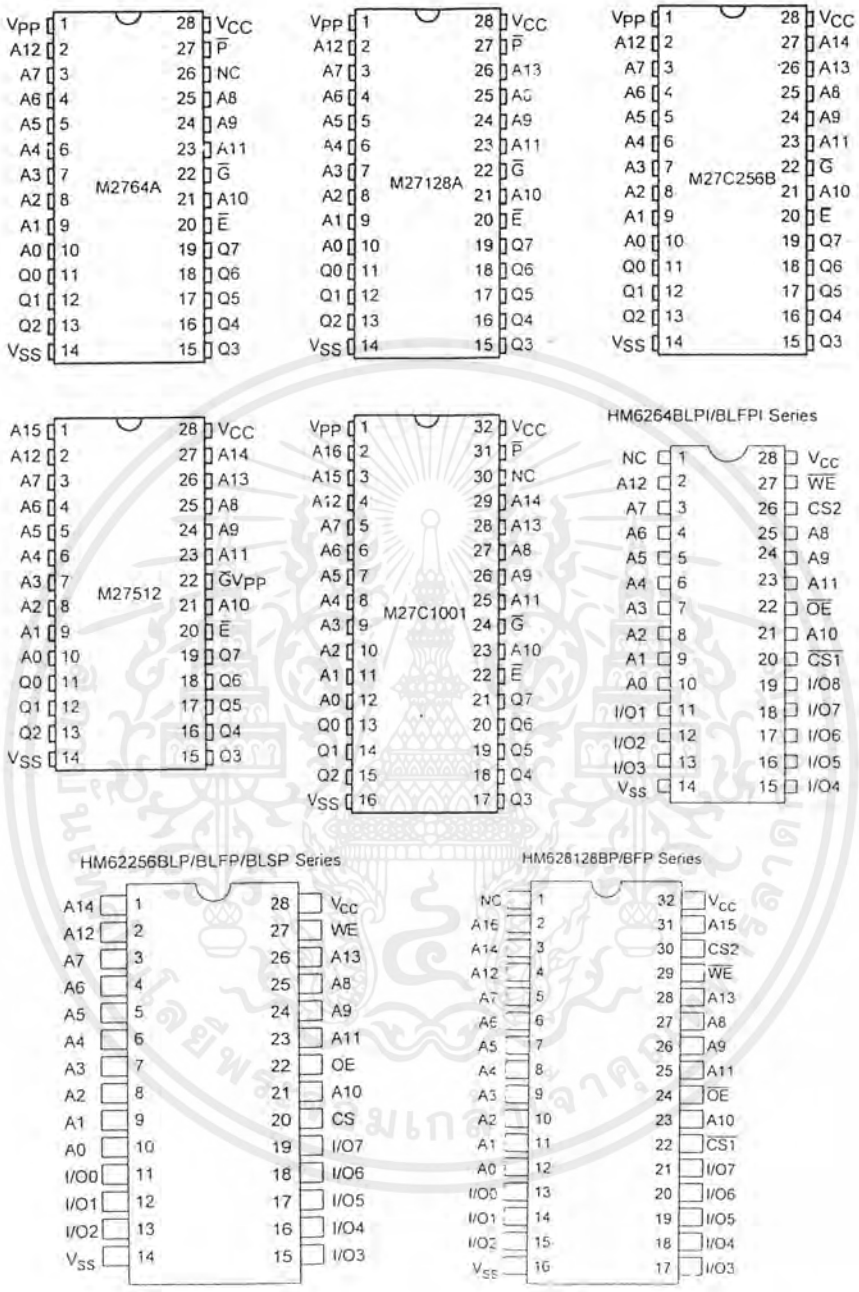
**3.จงเขียน boolean จาก ladder diagram**

**4. ทดลอง program กับเครื่อง PLC โดยทดลองป้อนสภาวะของ relay input ตาม ไดอะแกรมเวลาที่กำหนดขึ้นในข้อ 1 และดูค่าสภาวะ relay output ว่าเป็นไปตามไดอะแกรมเวลาของ relay output ในข้อ 1 หรือไม่**

**5.สรุปผลการทดลอง**

# ภาคผนวก จ
# รายละเอียดและคุณสมบัติของอุปกรณ์





HARRIS SEMICONDUCTOR

# 82C55A

## CMOS Programmable Peripheral Interface

August 1996

### Features

- Pin Compatible with NMOS 8255A
- 24 Programmable I/O Pins
- Fully TTL Compatible
- High Speed, No "Wait State" Operation with 5MHz and 8MHz 80C86 and 80C88
- Direct Bit Set/Reset Capability
- Enhanced Control Word Read Capability
- Scaled SAJI IV CMOS Process
- 2.5mA Drive Capability on All I/O Ports
- Low Standby Power (ICCSB) . . . . . 10µA

### Ordering Information

| PACKAGE | TEMPERATURE RANGE | 5MHz | 8MHz | PKG. NO. |
|---|---|---|---|---|
| Plastic DIP | 0°C to +70°C | CP82C55A-5 | CP82C55A | E40.6 |
| | -40°C to +85°C | IP82C55A-5 | IP82C55A | E40.6 |
| PLCC | 0°C to +70°C | CS82C55A-5 | CS82C55A | N44.65 |
| | -40°C to +85°C | IS82C55A-5 | IS82C55A | N44.65 |
| CERDIP | 0°C to +70°C | CD82C55A-5 | CD82C55A | F40.6 |
| | -40°C to +85°C | ID82C55A-5 | ID82C55A | F40.6 |
| | -55°C to +125°C | MD82C55A-5/B | MD82C55A/B | F40.6 |
| SMD# | | 8406601QA | 8406602QA | F40.6 |
| CLCC | -55°C to +125°C | MR82C55A-5/B | MR82C55A/B | J44.A |
| SMD# | | 8406601XA | 8406602XA | J44.A |

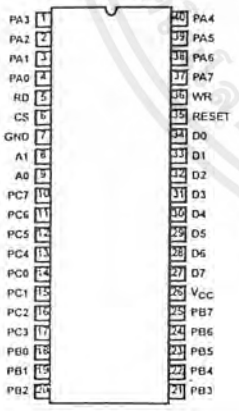
### Description

The Harris 82C55A is a high performance CMOS version of the industry standard 8255A and is manufactured using a self-aligned silicon gate CMOS process (Scaled SAJI IV). It is a general purpose programmable I/O device which may be used with many different microprocessors. There are 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. The high performance and industry standard configuration of the 82C55A make it compatible with the 80C86, 80C88 and other microprocessors.

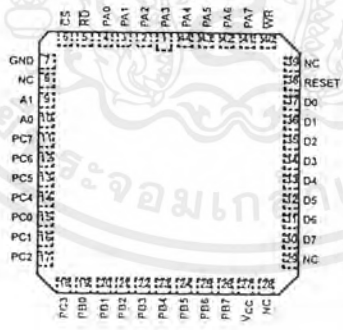
Static CMOS circuit design insures low operating power. TTL compatibility over the full military temperature range and bus hold circuitry eliminate the need for pull-up resistors. The Harris advanced SAJI process results in performance equal to or greater than existing functionally equivalent products at a fraction of the power

### Pinouts

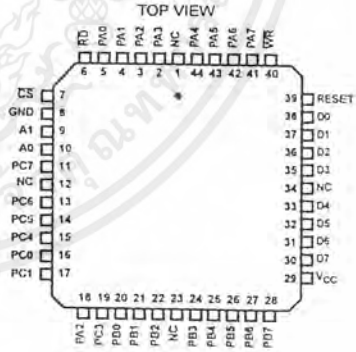
40 LEAD DIP
TOP VIEW




44 LEAD CLCC
TOP VIEW




44 LEAD PLCC
TOP VIEW



HARRIS SEMICONDUCTOR

# ICL232

## +5V Powered Dual RS-232 Transmitter/Receiver

December 1993

### Features

- Meets All RS-232C Specifications
- Requires Only Single +5V Power Supply
- Onboard Voltage Doubler/Inverter
- Low Power Consumption
- 2 Drivers
  - ±9V Output Swing for +5V Input
  - 300Ω Power-off Source Impedance
  - Output Current Limiting
  - TTL/CMOS Compatible
  - 30V/μs Maximum Slew Rate
- 2 Receivers
  - ±30V Input Voltage Range
  - 3kΩ to 7kΩ Input Impedance
  - 0.5V Hysteresis to Improve Noise Rejection
- All Critical Parameters are Guaranteed Over the Entire Commercial, Industrial and Military Temperature Ranges

### Applications

- Any System Requiring RS-232 Communications Port
  - Computer - Portable and Mainframe
  - Peripheral - Printers and Terminals
  - Portable Instrumentation
  - Modems
  - Dataloggers

### Description

The ICL232 is a dual RS-232 transmitter/receiver interface circuit that meets all EIA RS-232C specifications. It requires a single +5V power supply, and features two onboard charge pump voltage converters which generate +10V and -10V supplies from the 5V supply.

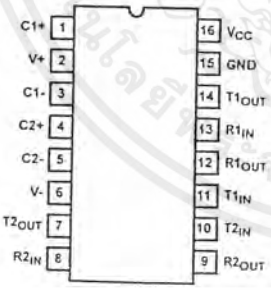
The drivers feature true TTL/CMOS input compatibility, slew-rate-limited output, and 300Ω power-off source impedance. The receivers can handle up to +30V, and have a 3kΩ to 7kΩ input impedance. The receivers also have hysteresis to improve noise rejection.

### Ordering Information

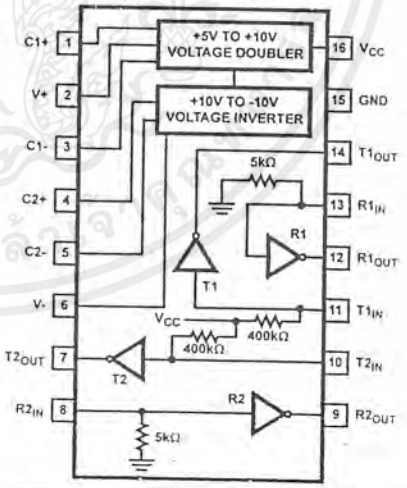
| PART NUMBER | TEMPERATURE RANGE | PACKAGE |
|---|---|---|
| ICL232CPE | 0°C to +70°C | 16 Lead Plastic DIP |
| ICL232CJE | 0°C to +70°C | 16 Lead Ceramic DIP |
| ICL232CBE | 0°C to +70°C | 16 Lead SOIC (W) |
| ICL232IPE | -40°C to +85°C | 16 Lead Plastic DIP |
| ICL232IJE | -40°C to +85°C | 16 Lead Ceramic DIP |
| ICL232IBE | -40°C to +85°C | 16 Lead SOIC (W) |
| ICL232MJE | -55°C to + 125°C | 16 Lead Ceramic DIP |

### Pinouts

ICL232
(PDIP, CDIP, SOIC)
TOP VIEW



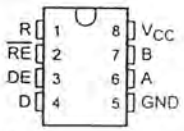

### Functional Diagram



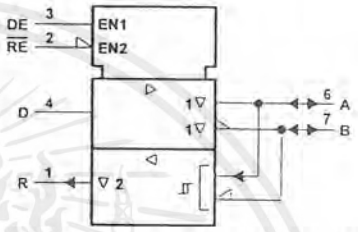
SLLS101A – JULY 1985 – REVISED MAY 1995

- Bidirectional Transceivers
- Meet or Exceed the Requirements of ANSI Standards EIA/TIA-422-B and RS-485 and ITU Recommendations V.11 and X.27
- Designed for Multipoint Transmission on Long Bus Lines in Noisy Environments
- 3-State Driver and Receiver Outputs
- Individual Driver and Receiver Enables
- Wide Positive and Negative Input/Output Bus Voltage Ranges
- Driver Output Capability . . . ±60 mA Max
- Thermal Shutdown Protection
- Driver Positive and Negative Current Limiting
- Receiver Input Impedance . . . 12 kΩ Min
- Receiver Input Sensitivity . . . ±200 mV
- Receiver Input Hysteresis . . . 50 mV Typ
- Operate From Single 5-V Supply

D OR P PACKAGE
(TOP VIEW)




logic symbol†

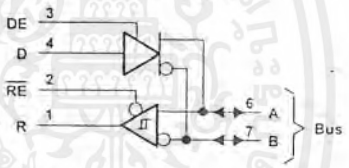



† This symbol is in accordance with ANSI/IEEE Std 91-1984 and IEC Publication 617-12

## description

The SN65176B and SN75176B differential bus transceivers are monolithic integrated circuits designed for bidirectional data communication on multipoint bus transmission lines. They are designed for balanced transmission lines ano meet ANSI Standards EIA/TIA-422-B and RS-485 and ITU Recommendations V.11 and X.27.

logic diagram (positive logic)




The SN65176B and SN75176B combine a 3-state differential line driver and a differential input line receiver, both of which operate from a single 5-V power supply. The driver and receiver have active-high and active-low enables, respectively, that can be externally connected together to function as a direction control. The driver differential outputs and the receiver differential inputs are connected internally to form differential input/output (I/O) bus ports that are designed to offer minimum loading to the bus whenever the driver is disabled or $V_{CC}$ = 0. These ports feature wide positive and negative common-mode voltage ranges making the device suitable for party-line applications.

**Function Tables**

DRIVER

| INPUT D | ENABLE DE | OUTPUTS A | OUTPUTS B |
|---|---|---|---|
| H | H | H | L |
| L | H | L | H |
| X | L | Z | Z |

RECEIVER

| DIFFERENTIAL INPUTS A – B | ENABLE $\overline{RE}$ | OUTPUT R |
|---|---|---|
| $V_{ID} \geq 0.2$ V | L | H |
| $-0.2$ V < $V_{ID}$ < 0.2 V | L | ? |
| $V_{ID} \leq -0.2$ V | L | L |
| X | H | Z |
| Open | L | H |

H = high level, L = low level, ? = indeterminate, X = irrelevant, Z = high impedance (off)

19-0094; Rev 6; 7/95

MAXIM

# Microprocessor Supervisory Circuits



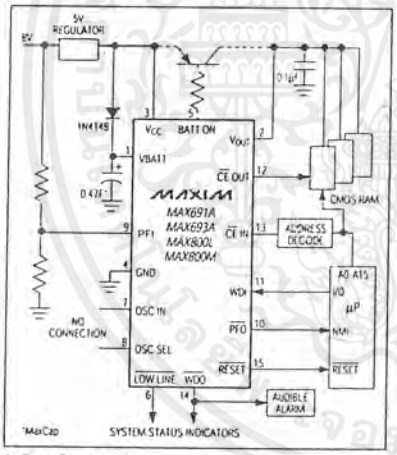
## General Description

The MAX691A/MAX693A/MAX800L/MAX800M microprocessor (µP) supervisory circuits are pin-compatible upgrades to the MAX691, MAX693, and MAX695. They improve performance with 30µA supply current, 200ms typ reset active delay on power-up, and 6ns chip-enable propagation delay. Features include write protection of CMOS RAM or EEPROM, separate watchdog outputs, backup-battery switchover, and a $\overline{RESET}$ output that is valid with $V_{CC}$ down to 1V. The MAX691A/MAX800L have a 4.65V typical reset-threshold voltage, and the MAX693A/MAX800M's reset threshold is 4.4V typical. The MAX800L/MAX800M guarantee power-fail accuracies to ±2%.

## Applications

Computers
Controllers
Intelligent Instruments
Automotive Systems
Critical µP Power Monitoring

## Typical Operating Circuit




## Features

- 200ms Power-OK/Reset Timeout Period
- 1µA Standby Current, 30µA Operating Current
- On-Board Gating of Chip-Enable Signals, 10ns Max Delay
- MaxCap™ or SuperCap™ Compatible
- Guaranteed $\overline{RESET}$ Assertion to $V_{CC}$ = 1V
- Voltage Monitor for Power-Fail or Low-Battery Warning
- Power-Fail Accuracy Guaranteed to ±2% (MAX800L/M)
- Available in 16-Pin Narrow SO and Plastic DIP Packages

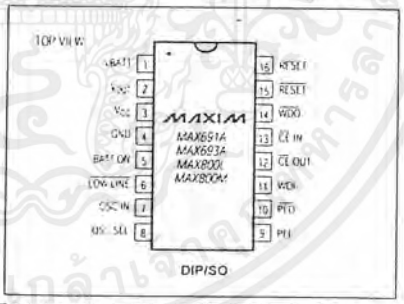
## Ordering Information

| PART | TEMP. RANGE | PIN-PACKAGE |
|---|---|---|
| MAX691ACPE | 0°C to +70°C | 16 Plastic DIP |
| MAX691ACSE | 0°C to +70°C | 16 Narrow SO |
| MAX691ACWE | 0°C to +70°C | 16 Wide SO |
| MAX691AC/D | 0°C to +70°C | Dice* |
| MAX691AEPE | -40°C to +85°C | 16 Plastic SO |
| MAX691AESE | -40°C to +85°C | 16 Narrow SO |
| MAX691AEWE | -40°C to +85°C | 16 Wide SO |
| MAX691AEJE | -40°C to +85°C | 16 CERDIP |
| MAX691AMJE | -55°C to +125°C | 16 CERDIP |

*Ordering Information continued on last page.*

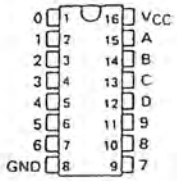
** Dice are specified at $T_A$ = +25°C*

## Pin Configuration




*™ SuperCap is a registered trademark of Baknor Industries. ™ MaxCap is a registered trademark of The Carborundum Corp.*

SN54145, SN54LS145 . . . J OR W PACKAGE
SN74145 . . . N PACKAGE
SN74LS145 . . . D OR N PACKAGE
(TOP VIEW)




FUNCTION TABLE

| NO. | INPUTS | | | | OUTPUTS | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | D | C | B | A | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 0 | L | L | L | L | L | H | H | H | H | H | H | H | H | H |
| 1 | L | L | L | H | H | L | H | H | H | H | H | H | H | H |
| 2 | L | L | H | L | H | H | L | H | H | H | H | H | H | H |
| 3 | L | L | H | H | H | H | H | L | H | H | H | H | H | H |
| 4 | L | H | L | L | H | H | H | H | L | H | H | H | H | H |
| 5 | L | H | L | H | H | H | H | H | H | L | H | H | H | H |
| 6 | L | H | H | L | H | H | H | H | H | H | L | H | H | H |
| 7 | L | H | H | H | H | H | H | H | H | H | H | L | H | H |
| 8 | H | L | L | L | H | H | H | H | H | H | H | H | L | H |
| 9 | H | L | L | H | H | H | H | H | H | H | H | H | H | L |
| INVALID | H | L | H | L | H | H | H | H | H | H | H | H | H | H |
| | H | L | H | H | H | H | H | H | H | H | H | H | H | H |
| | H | H | L | L | H | H | H | H | H | H | H | H | H | H |
| | H | H | L | H | H | H | H | H | H | H | H | H | H | H |
| | H | H | H | L | H | H | H | H | H | H | H | H | H | H |
| | H | H | H | H | H | H | H | H | H | H | H | H | H | H |

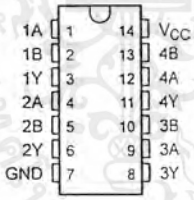
H = high level (off), L = low level (on)

(TOP VIEW)


FUNCTION TABLE
(each flip-flop)

| INPUTS | | | OUTPUT |
|---|---|---|---|
| $\overline{OE}$ | CLK | D | Q |
| L | ↑ | H | H |
| L | ↑ | L | L |
| L | L | X | $Q_0$ |
| H | X | X | Z |

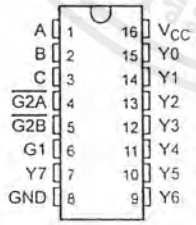
SN54ALS00A, SN54AS00 . . . J PACKAGE
SN74ALS00A, SN74AS00 . . . D OR N PACKAGE
(TOP VIEW)



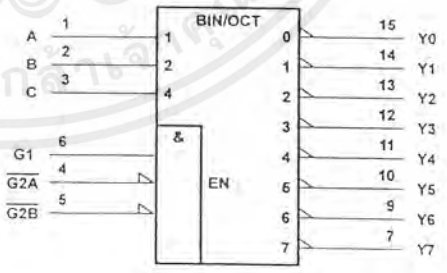

FUNCTION TABLE
(each gate)

| INPUTS | | OUTPUT |
|---|---|---|
| A | B | Y |
| H | H | L |
| L | X | H |
| X | L | H |

SN54ALS138A, SN54AS138 . . . J PACKAGE
SN74ALS138A, SN74AS138 . . . D OR N PACKAGE
(TOP VIEW)




logic symbols (alternatives)†

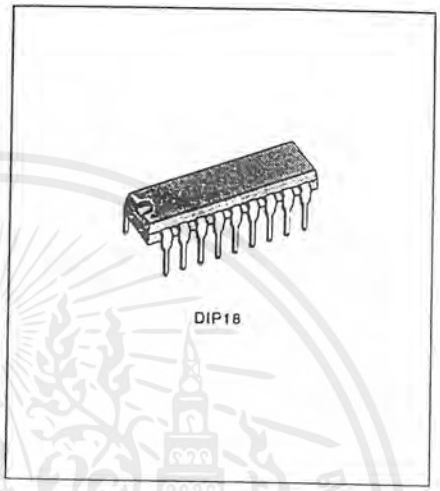


SGS-THOMSON MICROELECTRONICS

**ULN2801A**
**ULN2802A - ULN2803A**
**ULN2804A - ULN2805A**

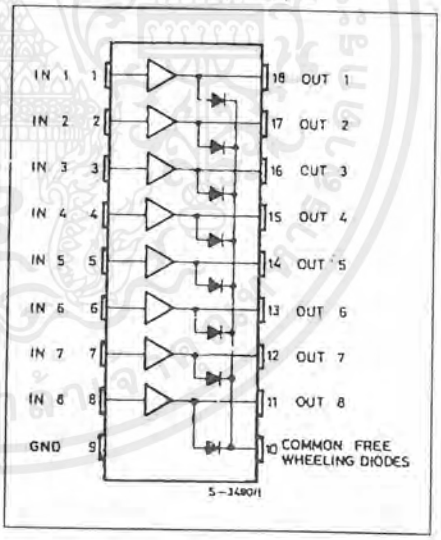
# EIGHT DARLINGTON ARRAYS

- EIGHT DARLINGTONS WITH COMMON EMITTERS
- OUTPUT CURRENT TO 500 mA
- OUTPUT VOLTAGE TO 50 V
- INTEGRAL SUPPRESSION DIODES
- VERSIONS FOR ALL POPULAR LOGIC FAMILIES
- OUTPUT CAN BE PARALLELED
- INPUTS PINNED OPPOSITE OUTPUTS TO SIMPLIFY BOARD LAYOUT



DIP18

PIN CONNECTION (top view)

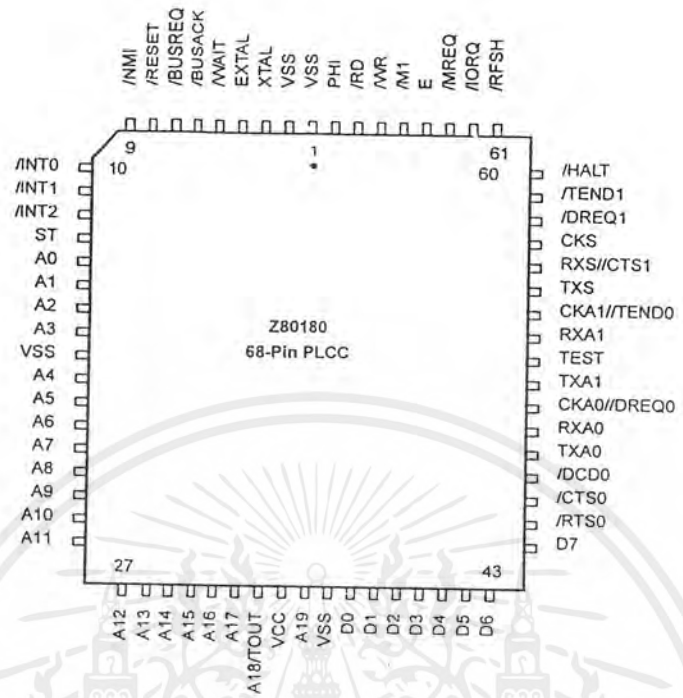



## DESCRIPTION

The ULN2801A-ULN2805A each contain eight darlington transistors with common emitters and integral suppression diodes for inductive loads. Each darlington features a peak load current rating of 600mA (500mA continuous) and can withstand at least 50V in the off state. Outputs may be paralleled for higher current capability.

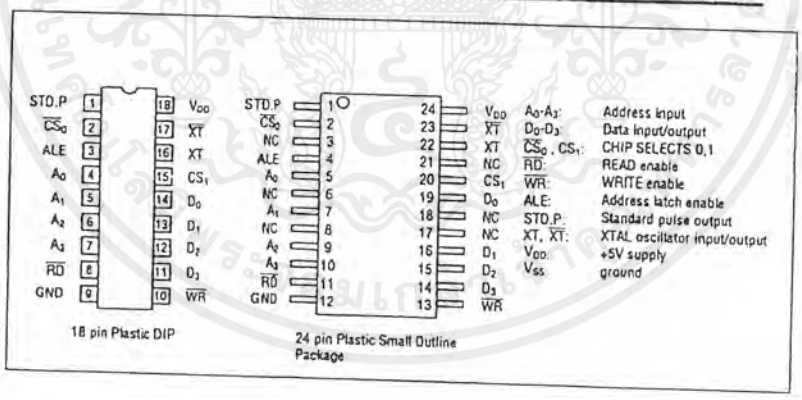
Five versions are available to simplify interfacing to standard logic families : the ULN2801A is designed for general purpose applications with a current limit resistor ; the ULN2802A has a 10.5kΩ input resistor and zener for 14-25V PMOS ; the ULN2803A has a 2.7kΩ input resistor for 5V TTL and CMOS ; the ULN2804A has a 10.5kΩ input resistor for 6-15V CMOS and the ULN2805A is designed to sink a minimum of 350mA for standard and Schottky TTL where higher output current is required.

All types are supplied in a 18-lead plastic DIP with a copper lead frame and feature the convenient input-opposite-output pinout to simplify board layout.



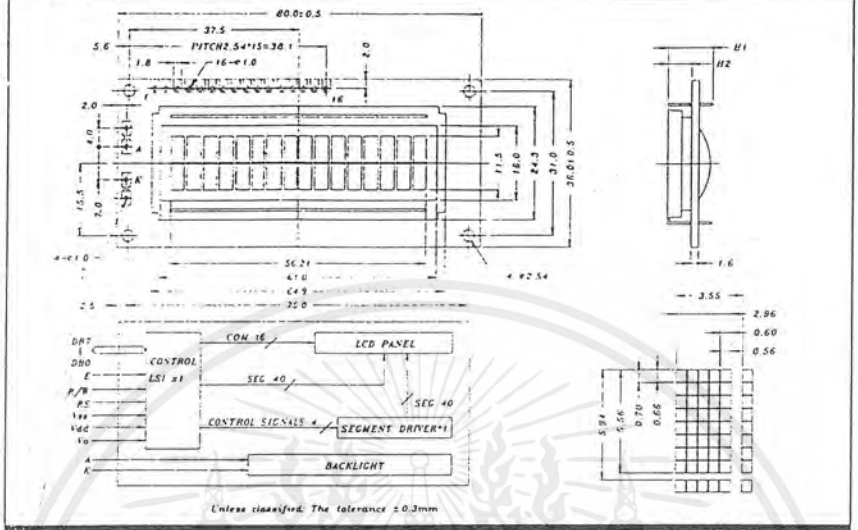

# MSM6242B

## DIRECT BUS CONNECTED CMOS REAL TIME CLOCK/CALENDAR



# PC 1602-C Series

## OUTLINE DIMENSION & BLOCK DIAGRAM




Unless classified. The tolerance ±0.3mm

## FEATURE

| | | | |
|---|---|---|---|
| Number of character | 16 x 2 | | |
| Character font | 5 x 7 dots-cursor | | |
| Duty-Bias | 1/16 - 1/4 | | |
| Type | Positive / Negative | | |

## MECHANICAL SPECIFICATION

| | | | |
|---|---|---|---|
| Overall size | 80.0 x 36.0 | Module | H2 / H1 |
| View area | 61.0 x 16.0 | N/O B/L | 5.1 / 9.7±0.5 |
| Dots size | 0.56 x 0.66 | EL B/L | 5.1 / 9.7±0.5 |
| Character size | 2.96 x 5.56 | LED B/L | 9.1 / 13.7±0.5 |

## PIN ASSIGNMENT

| Pin no. | Symbol | Function |
|---|---|---|
| 1 | Vss/K | Power supply (GND) |
| 2 | Vdd/A | Power supply (+5V) |
| 3 | Vo | Power supply for LCD driver |
| 4 | RS | Register select signal |
| 5 | R/W | Data read & write |
| 6 | E | Enable signal |
| 7 | DB0 | Data bus line |
| 8 | DB1 | Data bus line |
| 9 | DB2 | Data bus line |
| 10 | DB3 | Data bus line |
| 11 | DB4 | Data bus line |
| 12 | DB5 | Data bus line |
| 13 | DB6 | Data bus line |
| 14 | DB7 | Data bus line |
| 15 | NC/A | No connection/Power supply for B/L |
| 16 | NC/K | No connection/Power supply for B/L |

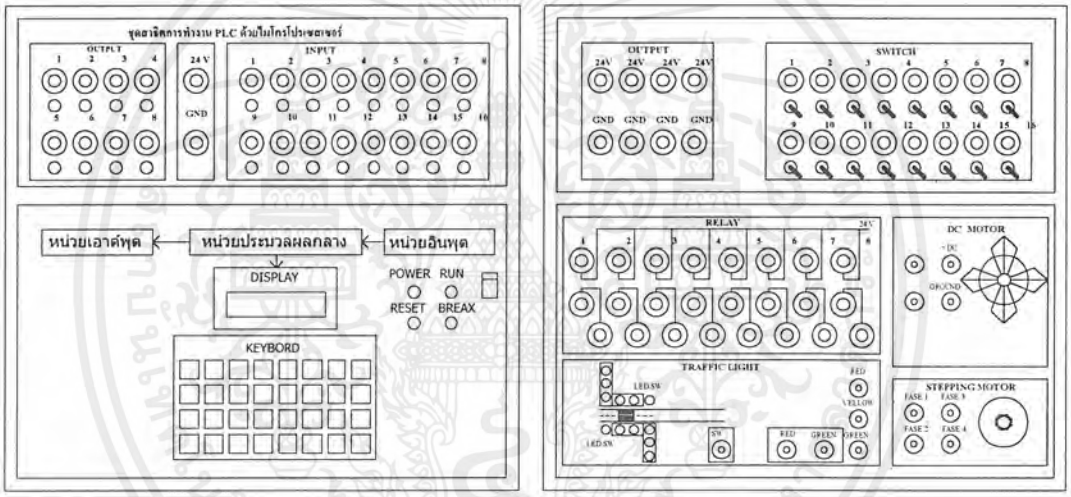
## ABSOLUTE MAXIMUN RATING

| Item | Symbol | Condition | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| Supply for LCD driver<br>Note:<br>N:Nor. Temp.<br>W:Wide Temp. | Vdd-Vo | Ta=-20°C | N | - | - | - | V |
| | | Ta=0°C | N | 4.5 | 4.8 | 5.1 | V |
| | | Ta=25°C | N | 4.1 | 4.4 | 4.7 | V |
| | | Ta=50°C | N | 3.8 | 4.1 | 4.4 | V |
| | | Ta=70°C | W | - | - | - | V |
| Supply for logic | Vdd-Vss | Ta=25°C | | 4.5 | 5.0 | 5.5 | V |
| Operation Temp. | Top | - | N | 0 | 25 | 50 | °C |
| | | | W | -20 | 25 | 70 | |
| Storage Temp. | Tstg | - | N | -20 | 25 | 60 | °C |
| | | | W | -30 | 25 | 80 | |
| LED current consumption | If | Ta=25°C | | - | 90 | 180 | mA |
| LED B/L brightness | Iv | Ta=25°C If= 90mA | | - | 105 | - | cd/cm² |

# ภาคผนวก ฉ
# คู่มือการใช้งาน

# คู่มือการใช้งาน

## ชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์




สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์

ภาควิชาครุศาสตร์วิศวกรรม

คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

# สารบัญ


| เรื่อง | หน้า |
|---|---|
| 1.เครื่องมือและอุปกรณ์ | 302 |
| 2. ส่วนประกอบของชุดสาธิตการทำงาน PLC และ ชุดอินเตอร์เฟส | 302 |
| 2.1 เครื่องชุดสาธิตการทำงาน PLC | 302 |
| 2.2 ชุดอินเตอร์เฟส | 308 |
| 3. ขั้นตอนการใช้งาน | 311 |
| 3.1 การเคลียร์โปรกรมและการรันโปรแกรม | 311 |
| 3.2 ตัวอย่างการใช้ PLC ควบคุมไฟวิ่ง จากซ้ายไปขวา | 311 |
| 3.3 ตัวอย่างการใช้งานชุดรีเลย์และสวิทช์ | 312 |
| 3.4 ตัวอย่างการใช้งานชุด DC Motor | 314 |
| 3.5 ตัวอย่างการใช้งานชุด Stepping Motor | 315 |
| 3.6 ตัวอย่างการใช้งานส่วนของชุด การควบคุมไฟจราจร | 317 |
| 3.7 ตัวอย่างการใช้งานส่วนของการรับส่งข้อมูลระหว่างคอมพิวเตอร์กับชุดสาธิตการทำงาน PLC | 320 |
| 4. ข้อควรระวังในการใช้งาน | 321 |

คู่มือการใช้ชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ คู่มือในเล่มนี้ จะกล่าวถึงการใช้งานของระบบได้แก่

1. เครื่องมือและอุปกรณ์
2. ส่วนประกอบของชุดสาธิตการทำงาน PLC กับชุดอินเตอร์เฟส
3. ขั้นตอนการใช้งาน
4. ข้อควรระวัง

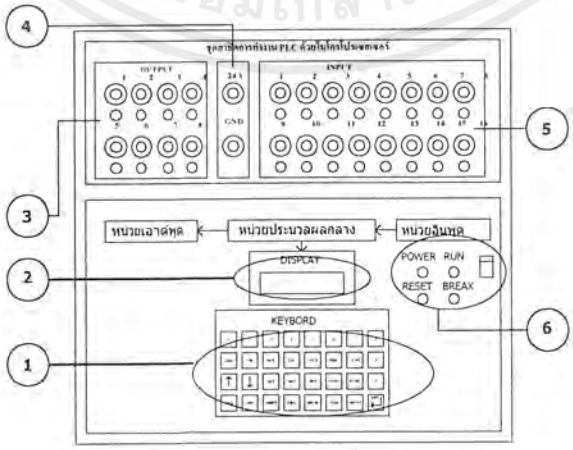
**รายละเอียดมีดังนี้**

## 1.เครื่องมือและอุปกรณ์

| | | |
|---|---|---|
| 1.1 ชุดสาธิต PLC ด้วยไมโครโปรเซสเซอร์ | 1 | เครื่อง |
| 1.2 ชุดอินเตอร์เฟส | 1 | เครื่อง |
| 1.3 เครื่องคอมพิวเตอร์ จะต้องมี สเปคดังนี้ | 1 | เครื่อง |
| 1.3.1 คอมพิวเตอร์ของ IBM เพนเทียม 133 เมกะเฮิรตซ์ ขึ้นไป | | |
| 1.3.2 หน่วยความจำแรมอย่างน้อย 16 เมกะไบต์ | | |
| 1.3.3 ระบบปฏิบัติการไมโครซอฟต์วินโดว์ 98 | | |
| 1.3.4 จอภาพมีความละเอียด 800*600 พิเซล ขึ้นไป | | |
| 1.4 สายเชื่อมต่อ RS-232 | 1 | เส้น |
| 1.5 สายต่อเพื่อทดลอง | 20 | เส้น |

## 2. ส่วนประกอบของชุดสาธิตการทำงาน PLC กับชุดอินเตอร์เฟส

2.1 ส่วนประกอบของชุดสาธิตการทำงาน PLC




**รูปที่ 2.1** ชุดสาธิตการทำงาน PLC

**ในส่วนที่ 1** จะเป็นคีย์ที่ใช้งาน จะมีอยู่ 4 แถว จะทำการอธิบายไปทีละแถว

1) แถวแรกจากซ้ายไปขวา


คีย์ที่ 1 – 8 เป็นคีย์ตัวเลข

2) แถวที่สองจากซ้ายไปขวา

| | |
|---|---|
| AND | เป็นการนำค่าสภาวะของอุปกรณ์ต่างๆที่กำหนด ตั้งแต่ 2 อุปกรณ์ขึ้นไปมากระทำลอจิก AND |
| OR | เป็นการนำค่าสภาวะของอุปกรณ์ต่างๆที่กำหนดตั้งแต่ 2 อุปกรณ์ขึ้นไปมากระทำการ OR |
| NOT | เป็นการนำค่าสภาวะที่ได้ทำการกลับเป็นสภาวะตรงกันข้าม |
| LD | ใช้สำหรับอ่านหรือนำค่าสภาวะต่างๆ จริง เท็จ ของอุปกรณ์ โดยคำสั่งนี้จะใช้ในการเริ่มต้นของ LADDER DIAGRAM |
| OUT | เป็นการนำค่าสภาวะที่ได้ส่งไปยังอุปกรณ์ที่กำหนด |
| TIM | เป็นคำสั่งการตั้งเวลาให้กับไทม์เมอร์ |
| CNT | เป็นคำสั่งที่ใช้ตัวนับ ซึ่งมี 48 ตำแหน่ง คือ 00-47 โดยจะให้มีการนับได้ในช่วง 0000 – 9999 ครั้ง |
| 9 | เป็นคีย์เลข 9 |

3) แถวที่สามจากซ้ายไปขวา

| | |
|---|---|
| ↑ | เป็นคีย์ลูกศรเลื่อนขึ้น |
| ↓ | เป็นคีย์ลูกศรเลื่อนลง |
| SET | ใช้รวมกับคำสั่ง CONT ใช้บังคับรีเลย์ภายในให้สภาวะเป็น ON |
| RES | ใช้รวมกับคำสั่ง CONT ใช้บังคับรีเลย์ภายในให้สภาวะเป็น OFF |
| SFT | สำหรับใช้ในการเลื่อนข้อมูล |
| CONT | ใช้บังคับสถานะรีเลย์ ใช้ร่วมกับคำสั่ง SET ,RES |
| FUNC | เป็นคีย์ที่ใช้ในการเรียกคำสั่งที่นอกเหนือจากคำสั่งพื้นฐานมาใช้งาน |
| 0 | เป็นคีย์หมายเลข 0 |

4) แถวที่สี่จากซ้ายไปขวา

| | |
|---|---|
| DEL | ใช้ในการลบ |
| INS | ใช้ในการแทรกคำสั่ง |
| SHIFT | ใช้เมื่อต้องการเปลี่ยนแปลงค่าของตัวตั้งเวลา |
| CHG | เป็นคีย์ที่ใช้ค้นหาคำสั่ง |

SRCH — กดคีย์ 1 ครั้ง จะเป็นการ CLEAR หน้าจอแสดงผล และใช้ยกเลิกสภาวะคำสั่งกดคีย์ 2 ครั้ง ตำแหน่งโปรแกรมจะชี้ไปที่ตำแหน่งแรก

MONT — ใช้ในสภาวะโหมดควบคุมโดยทำงานคู่กับการใช้ คีย์ CONT คือ ถ้ามีการใช้คีย์ CONT บังคับสถานะ OUTPUT ของรีเลย์ภายในเมื่อมีการกด MONTในตำแหน่งที่มีคำสั่งเกี่ยวกับ OUTPUT หน้าจอ LCD บรรทัดที่ แสดงสถานะ “ON” หรือ “OFF” จะหายไป

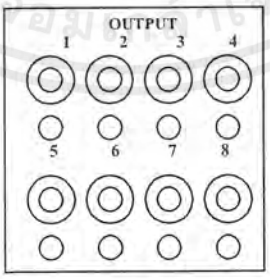
ENT ← — เป็นคีย์คำสั่งที่ใช้ในการปิดท้ายคำสั่งทุกคำสั่ง

**ในส่วนที่ 2**

จะเป็นส่วนของ LCD ซึ่งสามารถแสดงผลได้ถึง 16X2 ตัวอักษรดังรูปที่ 2


**รูปที่ 2.2** ส่วนของ DISPLAY

**ในส่วนที่ 3**




**รูปที่ 2.3** แสดง OUTPUT ชุดสาธิต PLC

เป็นส่วนของ OUTPUT มีทั้งหมด 8 OUTPUT ในแต่ละ OUTPUT นั้น จะมีการอ้างตำแหน่งแตกต่างกัน คือ

OUTPUT 1 จะต้องอ้างตำแหน่งที่ 0700

OUTPUT 2 จะต้องอ้างตำแหน่งที่ 0701

OUTPUT 3 จะต้องอ้างตำแหน่งที่ 0702

OUTPUT 4 จะต้องอ้างตำแหน่งที่ 0703

OUTPUT 5 จะต้องอ้างตำแหน่งที่ 0704

OUTPUT 6 จะต้องอ้างตำแหน่งที่ 0705

OUTPUT 7 จะต้องอ้างตำแหน่งที่ 0706

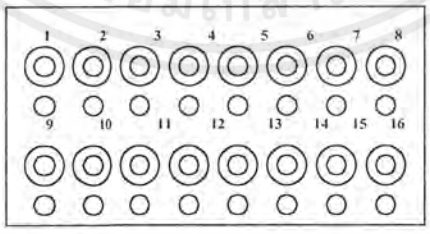
OUTPUT 8 จะต้องอ้างตำแหน่งที่ 0707

**ในส่วนที่ 4**


**รูปที่ 2.4** แสดง จุดต่อไฟ 24 V ให้กับชุดสาธิต PLC

ในส่วนนี้จะเป็นส่วนที่ เอาไว้เพื่อใช้ในการป้อนไฟ 24 V ให้กับชุดสาธิต PLC เอาไว้จ่ายให้กับโหลด

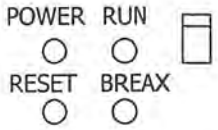
**ในส่วนที่ 5**




**รูปที่ 2.5** แสดงส่วนของ INPUT ชุดสาธิต PLC

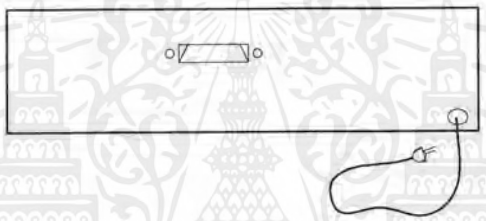
ส่วนนี้จะเป็นส่วน INPUT เอาไว้เพื่อใช้การทดลองในบ้างโปรแกรมจะต้องมีการใช้สวิทช์ในการควบคุม จะต้องต่อสวิทช์ทางช่อง INPUT นี้ จะสามารถต่อสวิทช์ได้ทั้งหมด 16 ตัว

**ส่วนที่ 6**




ในส่วนนี้จะมีสวิทช์ 3 ตัวคือ ตัวที่ 1 จะใช้ทำหน้าที่ในการ RESET ตัวที่ 2 จะเป็นการหยุดโปรแกรมชั่วคล่าว สวิทช์ตัวที่ 4 จะเป็นสวิทช์ที่ใช้รันโปรแกรม และจะมี LED แสดงสถานะ POWER และ การ รันโปรแกรม

**ในส่วนที่ 7**

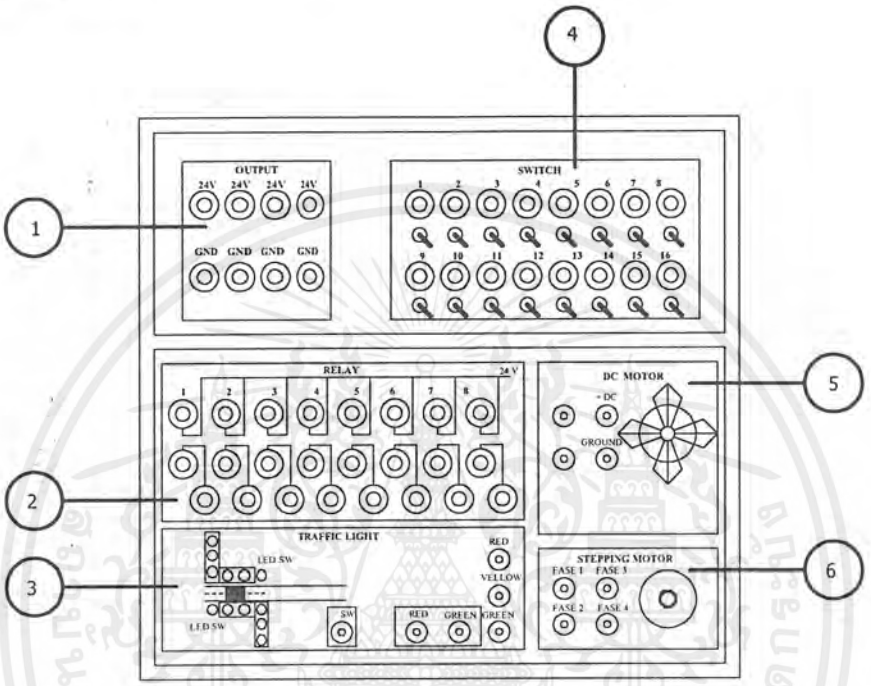


**รูปที่ 2.6** แสดง PORT RS 232

ในรูปจะเป็นด้านข้างของชุดสาธิต PLC ในส่วนนี้จะประกอบไปด้วยสายไฟ 220Vและ PORT RS – 232 เพื่อใช้ในการเชื่อมต่อระหว่าง PLC กับเครื่องคอมพิวเตอร์

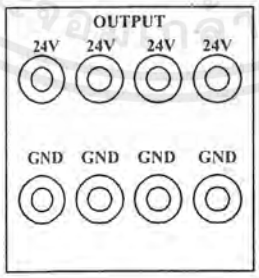
**2.2 ส่วนประกอบชุดอินเตอร์เฟส**

ในส่วนของชุดอินเตอร์เฟสนั้นจะแบ่งได้เป็น 6 ส่วนด้วยกันคือ 1. แหล่งจ่ายไฟ 2 .ส่วนของรีเลย์ 3.ส่วนของ ไฟจราจร 4.ส่วนของสวิทช์ 5.ส่วนของ DC MOTOR 6.ส่วนของ STEPPING MOTOR




**รูปที่ 2.7** ตัวเครื่องชุดอินเตอร์เฟส

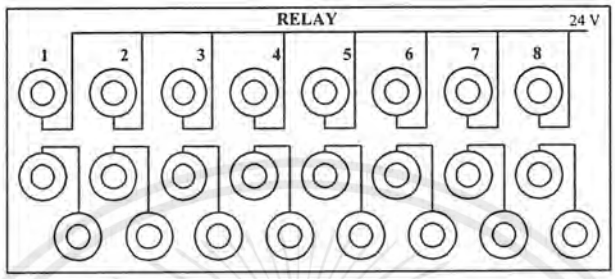
**ส่วนที่ 1** จะเป็นส่วนแหล่งจ่ายไฟ




**รูปที่ 2.8** แสดงส่วน OUTPUT ของชุดอินเตอร์เฟส

จะเป็นส่วนของแหล่งจ่ายไฟ 24 V เพื่อใช้จ่ายให้กับ PLC และเป็นแหล่งจ่ายให้กับ DC MOTOR และ STEPPING MOTOR

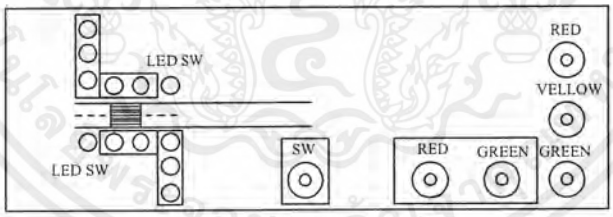
**ส่วนที่ 2** จะเป็นส่วนของรีเลย์




**รูปที่ 2.9** ส่วนของรีเลย์

ในส่วนนี้จะมีรีเลย์ด้วยกันถึง 8 ตัว เพื่อที่จะใช้ในการทดลอง รีเลย์จักเป็นสวิทช์แต่สวิทช์นี้สามารถควบคุมได้โดยใช้กระแสไฟเป็นตัวควบคุม

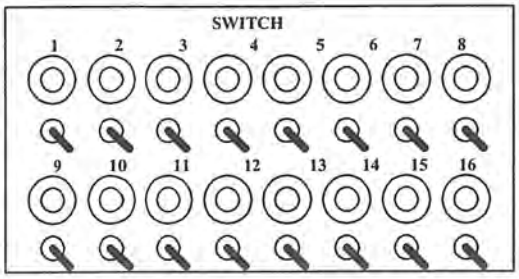
**ส่วนที่ 3** จะเป็นส่วนของไฟจราจร




**รูปที่ 2.10** ส่วนของไฟจราจร

ในส่วนนี้จะเป็นส่วนที่จะใช้เป็นการทดลองและฝึกทักษะการเขียนโปรแกรมควบคุมไฟจราจร

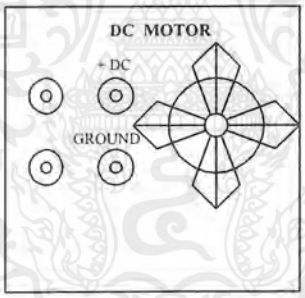
**ส่วนที่ 4** เป็นส่วนของสวิทช์




**รูปที่ 2.11** แสดงส่วนของสวิทช์

ในส่วนที่ 4 นั้นจะมีสวิทช์ที่สามารถนำไปใช้งานได้ทั้งหมด 16 ตัว เพื่อใช้ในการทดลองร่วมกับชุดสาธิต PLC

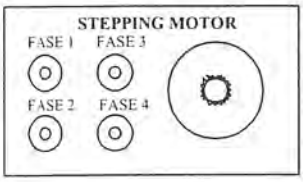
**ส่วนที่ 5** เป็นส่วนของ DC MOTOR




**รูปที่ 2.12** เป็นส่วน DC MOTOR

**ส่วนที่ 6** เป็นส่วน STEPPING MOTOR

ในชุดอินเตอร์เฟสนั้นจะมีชุดการทดลอง STEPPING MOTOR และ ชุดนี้เป็น STEPPING MOTOR แบบ 4 FASE ในการควบคุมจะต้องใช้ทั้ง 4 FASE ถึงจะสามารถควบคุมการทำงานของมันได้




**รูปที่ 2.13** เป็นส่วน STEPPING MOTOR

## 3. ขั้นตอนการใช้งาน

### 3.1 การ เคลียร์ โปรแกรม และการ รันโปรแกรม

1) ในการเขียนโปรแกรมหรือการที่จะป้อนโปรแกรมทุกครั้งจะต้องมีการเคลียร์ โปรแกรมก่อน ขั้นตอนในการเคลียร์โปรแกรมคุณก็ทำการกด FUNE 7 8 CLR MONT CLR เมื่อกดแล้วจะสังเกตได้ว่าโปรแกรมจะหายไปและจะแสดงที่ตำแหนงแอดเดรสที่ 0000 คือทำการเริ่มต้นการป้อนโปรแกรมใหม่

2) การรันโปรแกรมง่ายมาก เมื่อคุณทำการป้อนโปรแกรมเสร็จแล้วก็ทำการกด FUNC 01 เป็นการจบโปรแกรม และคุณก็ทำการโยก สวิทช์ RUN เท่านั้นเครื่องก็จะทำการรันโปรแกรมให้ทันที

### 3.2 ตัวอย่างการใช้งาน PLC ควบคุมไฟวิ่ง จากซ้ายไปขวา

1) นำชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ มาทำการเปลี่ยน Jumper ของการกำหนด แรงดันของชุดแสดงผลเอาต์พุตให้เป็น 10 V ซึ่งได้มาจาก แรงดันภายในชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์

2) ทำการ CLS โปรแกรม ที่อยู่ในหน่วยความจำ โดยการกด

FUNC 7 8 CLR MONT CLR

3) ทำการเขียนโปรแกรมไฟวิ่งเพื่อทดสอบการทำงานของเอาต์พุต ตามโปรแกรมข้างล่างนี้

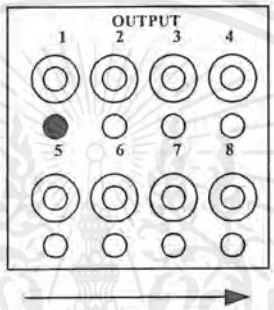
```
LD NOT          0900
OUT             0901
LD NOT TIM      0000
OUT             0902
LD              0902
TIM              00
TIM   DATA #    0010
TIM              00
TIM   DATA #    0010
LD              0707
OR              0901
LD NOT TIM      0000
LD              0000
```

| | |
|---|---|
| SFTREG | 0007 |
| LD NOT | 0903 |
| OUT | 0900 |
| END (01) | |

4) รันโปรแกรม

5) สังเกตการแสดงผลของ LED

6) การรันโปรแกรม



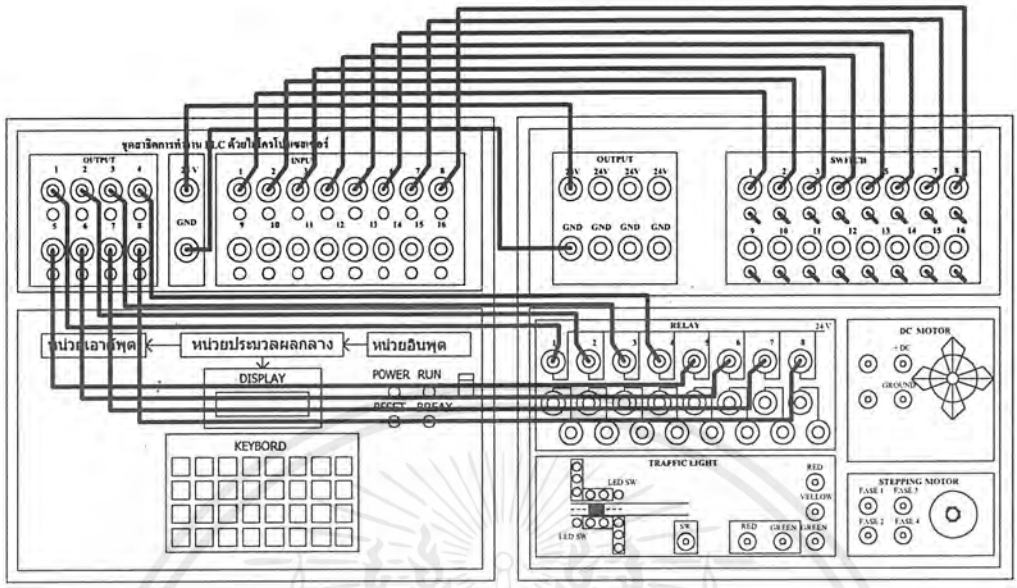

**รูปที่ 3.1** แสดงการรัน

เมื่อทำการรันโปรแกรมแล้ว สังเกตที่ OUTPUT นั้น LED จะติดจากซ้ายแล้วดับติดที่ LED 2 และดับ ติดที่ LED 3 และดับ จะติดดับ จากซ้ายไปขวาไปจนกว่าเราจะ RESET โปรแกรม

**3.3 ตัวอย่างการใช้งานชุดรีเลย์และสวิทช์**

1) ต่อไฟ 24 V ที่ได้จากชุดอินเตอร์เฟส เข้ากับชุดสาธิตการทำงาน PLC

2) ทำการต่อวงจรเพื่อทดสอบชุดรีเลย์ ในการต่อวงจรนี้จะต้องใช้สายต่อที่มีแจ๊ค ตัวผู้ 2 ตัวต่ออยู่กับขั้วสายและปลายสาย จะใช้ทั้งหมดจำนวน 18 เส้น นำสายต่อ 2 เส้น ขั้วสายทั้ง 2 เส้น ต่อ 24 V และ GND ที่ชุดอินเตอร์เฟส และปลายสายต่อ 24 V และ GND ที่ชุดสาธิต PLC เพื่อทำการต่อไฟแล้วจึงทำการเชื่อมต่อสวิทช์ จะใช้สวิทช์ทั้งหมด 8 ตัว ที่ชุดอินเตอร์เฟสนำสายต่อขั้วของสายต่อสวิทช์ 1 ถึง 8 ในชุดอินเตอร์เฟส และอีกปลายสายต่อเข้ากับ Input ของชุดสาธิต PLC เมื่อต่อสวิทช์แล้วต่อไฟเข้าแล้ว ก็จะมาเชื่อมต่อรีเลย์กันบ้าง นำสายเชื่อมต่อ 8 เส้นขั้วของสายต่อเข้ากับรีเลย์ 1 ถึง 8 ที่ชุดอินเตอร์เฟส ปลายสายต่อที่ Output 1 ถึง 8 ที่ชุดสาธิต PLC ดังรูปที่ 2.2




**รูปที่ 3.2** แสดงต่อใช้งานชุดรีเลย์และสวิทช์

3) ทำการ CLS โปรแกรมที่อยู่ในหน่วยความจำ โดยการกด

FUNC 7 8 CLR MONT CLR

4) เมื่อทำการ CLS แล้ว ทำการป้อนโปรแกรมการทดสอบตามนี้

| | |
|---|---|
| LD | 0000 |
| OUT | 0700 |
| LD | 0001 |
| OUT | 0701 |
| LD | 0002 |
| OUT | 0702 |
| LD | 0003 |
| OUT | 0703 |
| LD | 0004 |
| OUT | 0704 |
| LD | 0005 |
| OUT | 0705 |
| LD | 0006 |
| OUT | 0706 |

LD 0007

OUT 0707

END (01)

5) รันโปรแกรม

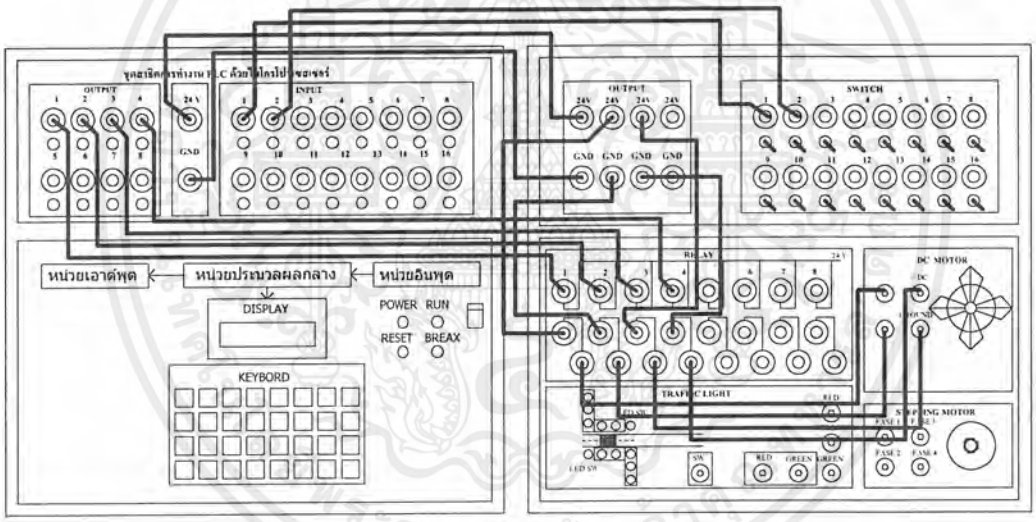
6) ผลการรันโปรแกรม

จะสังเกตเห็นว่าสวิทช์นั้นสามารถที่จะคอนโทรลการทำงานของ รีเลย์ได้เมื่อเรา ทำการโยกสวิทช์ตัวที่ 1 ริเลย์ตัวที่ 1 หน้าสัมผัสของรีเลย์ตัวนี้จะต่อทำให้สามารถวัดค่าความต้าน ทานได้

**3.4 ตัวอย่างการใช้งาน DC Motor**

1) นำชุดสาธิตการทำงาน PLC ด้วยไมโคร ต่อไฟ 24 V GND เข้ากับชุดอินเตอร์เฟส

2) ทำการต่อวงจรเพื่อทดสอบชุด DC Motor ดังรูป




**รูปที่ 3.3** การต่อวงจร DC Motor

3) เมื่อต่อวงจรเสร็จแล้ว ทำการป้อนโปรแกรมดังนี้

LD 0000

LD NOT TIM 01

TIM 00

TIM DATA 0020

LD TIM 00

OUT 700

| | |
|---|---|
| OUT | 701 |
| LD NOT TIM | 03 |
| TIM | 01 |
| TIM DATA | 0020 |
| LD TIM | 01 |
| AND NOT TIM | 03 |
| TIM | 02 |
| TIM DATA | 0020 |
| LD TIM | 02 |
| OUT | 702 |
| OUT | 703 |
| LD NOT TIM | 00 |
| TIM | 03 |
| TIM DATA | 0030 |
| END (01) | |

4) ทำการรันโปรแกรม

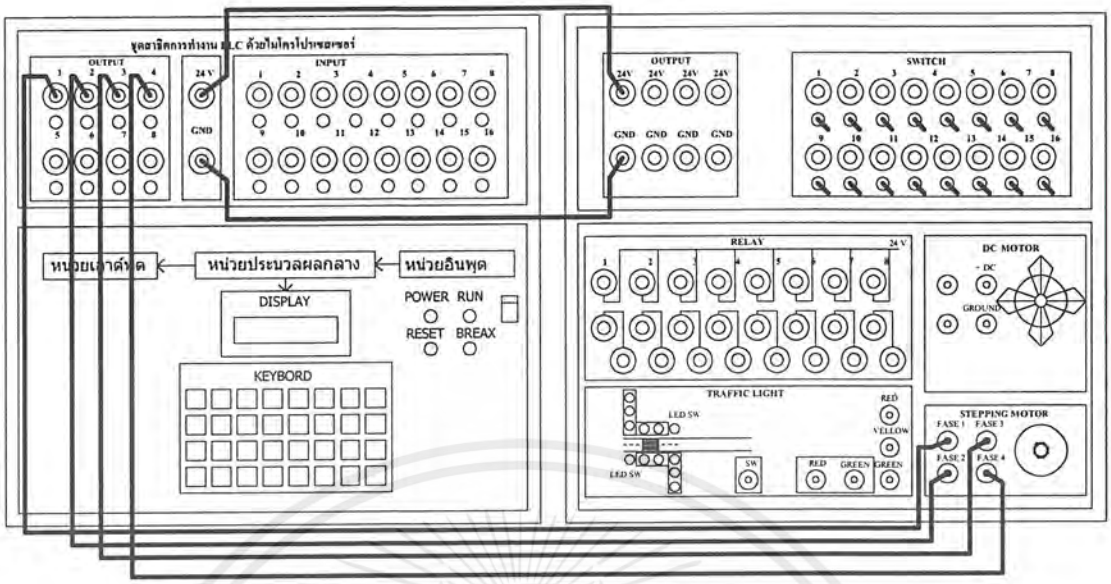
5) สังเกตการทำงานของ DC Motor

จากการเมื่อรันโปรแกรมแล้วสังเกตว่า DC Motor หมุนตามที่ โปรแกรมสั่งงาน คือ หมุนตามเข็มนาฬิกา แล้วหยุดหมุน แล้วก็หมุนทวนเข็มนาฬิกาเป็นอย่างนี้ไปตลอด

**3.5 ตัวอย่างการใช้งานชุด Stepping Motor**

1) ทำการต่อไฟ 24 V จากชุดอินเตอร์เฟส เข้ากับชุดสาธิตการทำงาน PLC

2) ทำการต่อวงจรเพื่อทดสอบชุด Stepping Motor ในการต่อวงจร Stepping นี้จะใช้ Output ที่ 1 ถึง 4 ของชุดสาธิตการทำงาน PLC ต่อเข้ากับชุด Stepping Motor โดยใช้สายเชื่อมต่อ จำนวน 4 เส้น ขั้วสายทั้ง 4 เส้นต่อที่ Output 1 ถึง 4 ของชุดสาธิต PLC ปลายสายต่อกับ Stepping Motor 1 ต่อ กับ 1 , 2 ต่อกับ 2 , 3 ต่อกับ 3 , 4 ต่อกับ 4 ดังรูปที่ 2.4




**รูปที่ 3.4** แสดงการต่อ Stepping Motor

3) ทำการ CLR โปรแกรมที่อยู่ในหน่วยความจำที่ไม่ได้ใช้ออกให้หมดด้วยการ กด

FUNC 7 8 CLR MONT CLR

4) ทำการป้อนโปรแกรม ดังนี้

| | |
|---|---|
| LD NOT | 0900 |
| OUT | 0901 |
| LD NOT TIM | 0000 |
| OUT | 0902 |
| LD | 0902 |
| TIM | 00 |
| TIM DATA # | 0001 |
| LD | 0703 |
| OR | 0901 |
| LD NOT TIM | 0000 |
| LD | 0000 |
| SFTREG | 0007 |
| LD NOT | 0903 |

OUT 0900

END(01)

5) ทำการรันโปรแกรม

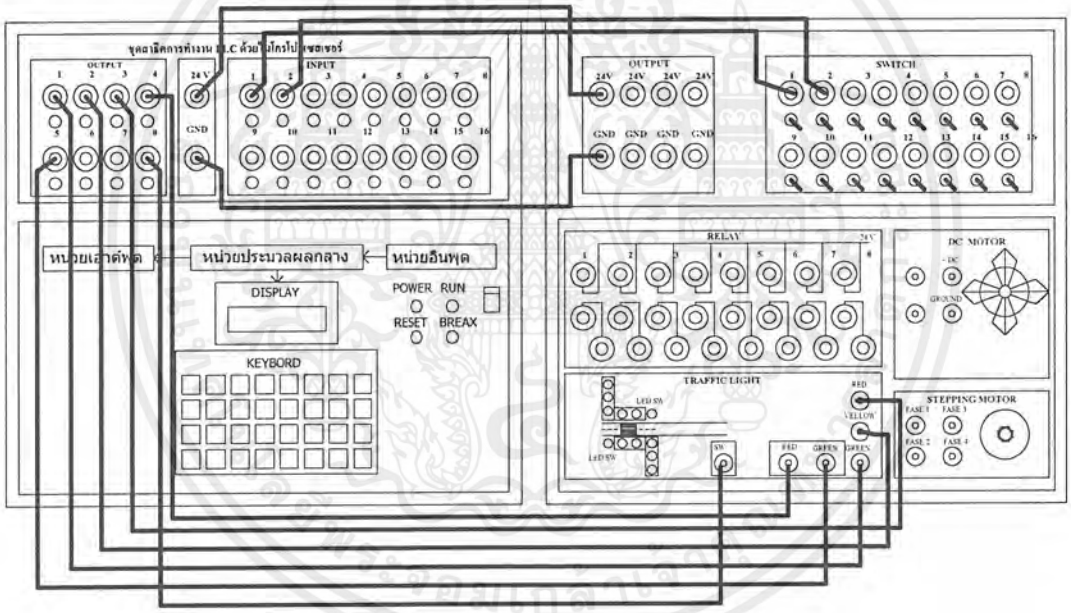
6) สังเกตการทำงาน ของ Stepping Motor

คุณจะเห็นว่าโปรแกรมก็คือไฟกระพริบนั้นเองเมื่อเราทำการรันโปรแกรมแล้วจะมีกระแสไหลไปที่ FASE ที่ 1 ก่อนและหยุดจ่าย มาจ่ายให้กับ ที่ FASE ที่ 2 ทำแบบนี้จนครบ 4 FASE และเริ่มใหม่ เมื่อ Stepping Motor ได้กระแสแบบนี้ในแต่ละ FASE ตัวมันก็จะหมุนจากซ้ายไปขวา

**3.6 ตัวอย่างการใช้งานส่วนของชุด การควบคุมไฟจราจร**

1) ทำการต่อไฟ 24 V และ Gnd จากชุดอินเตอร์เฟส เข้ากับชุดสาธิตการทำงาน PLC

2) ทำการต่อวงจรควบคุมไฟจราจร ดังรูป




รูปที่ 3.5 การต่อวงจรไฟจราจร

3) ทำการ CLR โปรแกรมที่อยู่ในหน่วยความจำ โดยการกด

FUNC 9 8 CLR MONT CLR

4) ป้อนโปรแกรมดังนี้

LD 0000

OR 0001

OR 0900

| | |
|---|---|
| AND NOT TIM | 0006 |
| OUT | 0900 |
| OUT | 0707 |
| LD | 0900 |
| TIM | 00 |
| TIM DATA | #0100 |
| LD TIM | 0000 |
| TIM | 01 |
| TIM DATA | #0050 |
| LD TIM | 0001 |
| TIM | 02 |
| TIM DATA | #0020 |
| LD TIM | 0002 |
| TIM | 03 |
| TIM DATA | #0060 |
| LD TIM | 0003 |
| AND NOT TIM | 0005 |
| LD CNT | 0000 |
| AND | 0900 |
| OR LD | 0000 |
| TIM | 04 |
| TIM DAT A | #0005 |
| LD TIM | 0004 |
| TIM | 05 |
| TIM DATA | #0005 |
| LD TIM | 0004 |
| LD NOT | 0900 |
| CNT | 00 |
| CNT DATA | #0006 |
| LD CNT | 0000 |

```
TIM          06
TIM DATA     #0020
LD           0900
AND NOT TIM  0000
OR  NOT      0900
OUT          0700
LD           0900
AND  TIM     0000
AND  NOT TIM 0001
OUT          0701
LD           0900
AND NOT      0700
AND NOT      0701
OUT          0702
LD NOT TIM   0002
OR CNT       0000
OUT          0703
LD  TIM      0002
AND NOT TIM  0003
OR  TIM      0004
AND NOT CNT  0000
OUT          0704
```

5) รันโปรแกรม

เมื่อ Switch กดอยู่ทั้ง 2 ด้าน โดยยังไม่กดปุ่มทางด้านใด ไฟเขียวทางด้านรถวิ่งจะติดและไฟแดงทางด้านคนข้ามจะติดค้าง เปิดสวิทช์ทางด้านใดถูกกดไฟ สถานะการกดจะติดค้าง ต่อจากนั้น 10 วินาที ไฟเขียวทางด้านรถวิ่งจะเปลี่ยนเป็นเหลืองนานเป็นเวลา 5 วินาที จากนั้นไฟแดงจะติดขึ้น เมื่อไฟแดงติด 2 วินาที ทางด้านคนข้ามจะเปลี่ยนจากแดงเป็นเขียวค้างอยู่นาน 6 วินาที จากนั้นก็จะกระพริบทุก 1 วินาที จำนวน 5 ครั้ง และเปลี่ยนเป็นแดงจนผ่านไป 2 วินาที ไฟแดงทางด้านรถวิ่งจะเปลี่ยนเป็นเขียว และไฟแสดงการกดสวิทช์ดับ

**3.7 ตัวอย่างการใช้ส่วนของการรับ-ส่ง ข้อมูลระหว่าง คอมพิวเตอร์ กับ ชุดสาธิต การทำงาน PLC ด้วยไมโครโปรเซสเซอร์**

1) ต่อ Port RS-232C ระหว่าง คอมพิวเตอร์ กับ ชุด สาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์

2) Set การรับส่งข้อมูลของ Port Com1 โดยให้เป็นมาตรฐานเดียวกับชุด สาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ คือ 8 Bit Data, Non Parity และ 1 Stop Bit โดยมีอัตราการรับ-ส่ง 9600 บิต ต่อ วินาที โดยใช้คำสั่ง Mode Com1:96,n,8,1

3) ทดลองการรับข้อมูล (ข้อมูลเป็น format.hex) จาก ชุด สาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ มาเก็บไว้ที่ คอมพิวเตอร์ โดยการใช้ คำสั่ง Copy Com1 Test1.Cod ที่คอมพิวเตอร์ และ คำสั่ง FUNC 98 ที่ ชุด สาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์

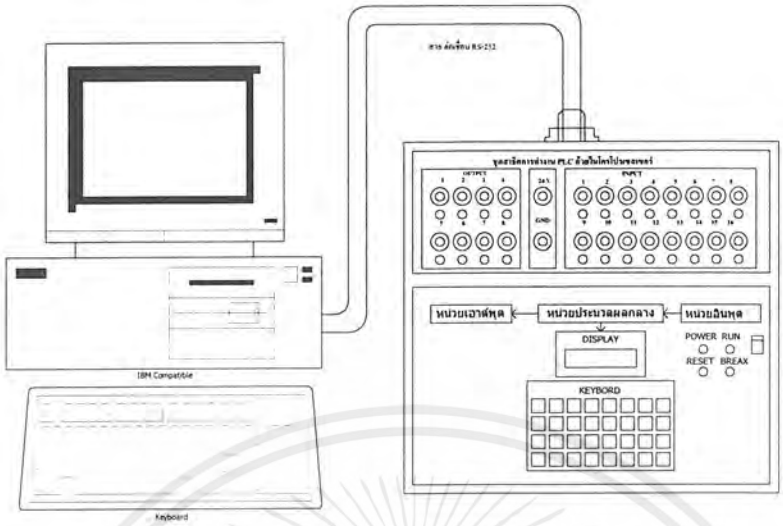
4) ทดลองการส่งข้อมูล (ข้อมูลเป็น format.hex) จาก คอมพิวเตอร์ มาเก็บไว้ที่ ชุดสาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ โดยการใช้ คำสั่ง Copy Test1.Cod Com1 ที่คอมพิวเตอร์ และ คำสั่ง Func 97 ที่ ชุด สาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์

5) ทดลองการส่งข้อมูล(ข้อมูลเป็น format ของภาษา Boolean) จาก ชุด สาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์ มาเก็บไว้ที่ คอมพิวเตอร์ โดยการใช้ คำสั่ง Copy Com1 Test1.Lad ที่คอมพิวเตอร์ และ คำสั่ง Func 95 ที่ ชุด สาธิตการทำงาน PLC ด้วยไมโครโปรเซสเซอร์

6) ตรวจสอบข้อมูลโดยใช้โปรแกรม EDIT

7) สังเกตผลจะได้ว่า

ตัวอย่างนี้จะเป็นการส่งข้อมูล จาก PLC ไปยังคอมพิวเตอร์ และจาก คอมไปยัง PLC ในการส่งข้อมูล จาก PLC ไปยังคอมพิวเตอร์นั้น การส่งจะส่ง ได้ 2 แบบคือ เป็น File จุด HEX คือ เป็นเลขฐาน 16 กับ File จุด Lad เป็น File ที่เก็บ เป็น โปรแกรม Ladder เลย ผลการทดลองสามารถทำได้ ส่วนการส่ง File ที่มีโปรแกรมไป RUN ที่ PLC นั้น ทำการเลือกโปรแกรมที่จะส่ง จะต้อง Fave File เป็นจุด HEX หรือ จุด Lad เท่านั้น เมื่อทำการส่ง Fave แล้วจะสังเกตเห็นว่า ที่ PLC นั้นจะมีโปรแกรม Fave นั้นอยู่




**รูปที่ 3.6** แสดงการเชื่อมต่อระหว่างชุดสาธิต PLC กับ คอมพิวเตอร์

## 4.ข้อควรระวังในการใช้งาน

4.1 ในการต่อไฟ 24 V เข้ากับชุดสาธิต PLC และชุดอินเตอร์เฟสจะต้องระวังการต่อสายกลับขั้ว

4.2 ในการเชื่อมต่อสาย RS-232 เข้ากับคอมพิวเตอร์ ควรจะเสียบให้แน่นถ้าเสียบไม่แน่นการส่งข้อมูลระหว่าง PLC กับ คอมพิวเตอร์อาจจะผิดพลาดได้

# บรรณานุกรม

ศูนย์ภาษาคอมพิวเตอร์, "การใช้งาน Z-80180 ",กรุงเทพมหานคร,หจก. สำนักพิมพ์ฟิสิกส์เซ็นเตอร์, 2534

บริษัทอีทีที จำกัด, "คู่มือ CPU Z80180 ",กรุงเทพมหานคร

บริษัทอีทีที จำกัด, "คู่มือ การใช้งาน ET-BOARD V4.0 ",กรุงเทพมหานคร

ธานินทร์ ถาวรศาสนวงศ์ และ ทินกร ดุ๊ก, "การอินเตอร์เฟส IBM PC ", กรุงเทพมหานคร,หจก. สำนักพิมพ์ฟิสิกส์เซ็นเตอร์, 2534

# ประวัติผู้แต่ง

| | |
|---|---|
| ชื่อผู้ทำปริญญานิพนธ์ | นายทองดี ศรีเจริญ |
| วันเดือนปีเกิด | 20 สิงหาคม 2520 |
| สถานที่เกิด | จังหวัดจันทบุรี |
| ภูมิลำเนาเดิม | 18/2 หมู่ 2 ตำบลพลอยแหวน อำเภอท่าใหม่ จังหวัดจันทบุรี 22120 |
| ที่อยู่ปัจจุบัน | 384 หมู่ 1 ซอยจินดานิเวศน์ แขวงลำปลาทิว เขตลาดกระบัง กรุงเทพมหานคร 10520 |
| โทรศัพท์ | 039-339407,02-7391224 |

## ประวัติการศึกษา

| | |
|---|---|
| ประถมศึกษา | โรงเรียนวัดบูรพาพิทยาราม |
| มัธยมศึกษาตอนต้น | โรงเรียนพูลสวัสดิราษฎร์นุกูล |
| ประกาศนียบัตรวิชาชีพ (ปวช.) | วิทยาลัยเทคนิคจันทบุรี |
| ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.) | วิทยาลัยเทคนิคจันทบุรี |
| ปริญญาตรี | สาขาวิชาอิเล็กทรอนิคส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม |
| คติพจน์ | จงทำวันนี้ให้ดีที่สุด เพื่ออนาคตในวันหน้า |

# ประวัติผู้แต่ง

| | |
|---|---|
| ชื่อผู้ทำปริญญานิพนธ์ | นายพงศกร พรหมเจริญ |
| วันเดือนปีเกิด | 7 ตุลาคม 2519 |
| สถานที่เกิด | จังหวัดชลบุรี |
| ภูมิลำเนาเดิม | 148/18 หมู่ 5 ตำบลห้วยกะปิ อำเภอเมือง จังหวัดชลบุรี 20130 |
| ที่อยู่ปัจจุบัน | 148/18 หมู่ 5 ตำบลห้วยกะปิ อำเภอเมือง จังหวัดชลบุรี 20130 |
| โทรศัพท์ | 038-383319 |

## ประวัติการศึกษา

| | |
|---|---|
| ประถมศึกษา | โรงเรียนประภัสสรวิทยา |
| มัธยมศึกษาตอนต้น | โรงเรียนชลราษฎรอำรุง |
| ประกาศนียบัตรวิชาชีพ (ปวช.) | วิทยาลัยเทคนิคชลบุรี |
| ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.) | วิทยาลัยเทคนิคชลบุรี |
| ปริญญาตรี | สาขาวิชาอิเล็กทรอนิคส์และคอมพิวเตอร์<br>ภาควิชาครุศาสตร์วิศวกรรม<br>คณะครุศาสตร์อุตสาหกรรม |
| คติพจน์ | รู้เขา รู้เรา ชัยชนะจะไปไหน |

# ประวัติผู้แต่ง

| | |
|---|---|
| ชื่อผู้ทำปริญญานิพนธ์ | นายวิชิต เวียงคำ |
| วันเดือนปีเกิด | 27 กุมภาพันธ์ 2519 |
| สถานที่เกิด | จังหวัดศรีสะเกษ |
| ภูมิลำเนาเดิม | 57 หมู่ 8 บ้านเทิน ต.บัวน้อย อ.กันทรารมย์ 33130 |
| ที่อยู่ปัจจุบัน | 384 หมู่ 1 ซ.จินดานิเวศน์ แขวงลำปลาทิว เขตลาดกระบัง กทม.10520 |
| โทรศัพท์ | 039-339407,02-7391224 |

## ประวัติการศึกษา

| | |
|---|---|
| ประถมศึกษา | โรงเรียนบ้านเทิน |
| มัธยมศึกษาตอนต้น | โรงเรียนบัวน้อยวิทยาลัย |
| ประกาศนียบัตรวิชาชีพ (ปวช.) | วิทยาลัยเทคโนโลยีอุบลราชชานี |
| ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.) | วิทยาลัยเทคนิคศรีสะเกษ |
| ปริญญาตรี | สาขาวิชาอิเล็กทรอนิคส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม |
| คติพจน์ | ศรัทธามุ่งมั่น,เดินทางสายกลาง |