ปีการศึกษา 2537

เครื่องจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้

(PROGRAMMABLE LOGIC CONTROLLER SIMULATOR)

โดย

นาย พินยุ พร้อมวงษ์

นาย สมชาย ชาวเวียง

นาย ภัทร์ชนนท์ เกิดพิมพ์

อาจารย์ที่ปรึกษา

ผศ. สุพรรณ กุลพาณิชย์

ปริญญานิพนธ์ปีการศึกษา 2537

ภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องจำลองการทำงาน เครื่องควบคุมแบบโปรแกรมได้
(PROGRAMMABLE LOGIC CONTROLLER SIMULATOR)

ผู้จัดทำ

นาย พิษณุ พรัอมวงษ์ NO.35-103282

นาย สมชาย ชาวเวียงลง NO.35-103294

นาย ภัทร์ชนนท์ เกิดพิพัฒน์ NO.35-103293

.....อาจารย์ที่ปรึกษา

(ผศ. สุนรรณ กุลพาณิชย์)

เครื่องจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้

นาย พิษณุ พร้อมวงษ์
นาย สมชาย ชาวเวียง
นาย ภัทร์ชนนท์ เกิดพิพัฒน์

ผศ. สุพรรณ กุลพาณิชย์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2537

บทคัดย่อ

เครื่องควบคุมแบบโปรแกรมได้ หรือ PLC(Programmable Logic Controller)ในการเขียนโปรแกรมส่วนมากจะใช้ภาษาบูลลีนที่เขียนมาจากการเขียน แลดเดอร์ไดอะแกรม (Ladder Diagram)มาแสดงการทำงานของวงจรควบคุมในโรงงานอุตสาหกรรมต่างๆเช่นเครื่อง PLC ของบริษัท OMRON รุ่น C20 เป็นต้น

ปริญญานิพนธ์นี้ เป็นการสร้างเครื่องจำลองการทำงานของเครื่องควบคุมแบบโปรแกรมได้ซึ่งประกอบไปด้วยโปรแกรม ที่สามารถแสดงผลได้ทางจอมอนิเตอร์ของคอมพิวเตอร์ และ เครื่องควบคุมอินพุท และ ทางเอาท์พุททาง RS-232 โปรแกรมที่เขียนได้ใช้ภาษาซี และ แอสแซมบลี้ ส่วนเครื่องควบคุมอินพุท และทางเอาท์พุทได้ออกแบบและสร้างขึ้นโดยใช้วัสดุและอุปกรณ์ที่หาได้ภายในประเทศ เครื่องจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้นี้ สามารถเขียนคำสั่งภาษาบูลลีนพื้นฐานจำนวน 13 คำสั่ง และสามารถแสดงเป็น Ladder Diagram ได้ การจำลองการทำงานโดยการใช้ คีย์บอร์ด เป็น อินพุทแสดงสถานะ (Status) 8 อินพุท/เอาท์พุทและแสดงการทำงานโดยใช้ input file เป็นอินพุท ซึ่งแสดงเป็นไดอะแกรม มีทั้งหมด 30,000 step เมื่อมีการเปลี่ยนแปลง อินพุทแต่ละครั้งเครื่องควบคุมอินพุท และ เอาท์พุทจะรับคำสั่งจากโปรแกรมผ่านทาง RS-232 แล้วสามารถทำงานตามต้องการได้โดยต่ออุปกรณ์ อินพุทและเอาท์พุท ในกรณีที่ต่อใช้งานมีช่องสำหรับเสียบ(slot) จำนวนทั้งหมด 8 ช่องซึ่งเตรียมไว้สำหรับ Solid State Relay และ Output Relay

จากผลการทดสอบ เครื่องจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้ สามารถทำงานได้ตามวัตถุประสงค์ที่ตั้งไว้อย่างสมบูรณ์และยังสามารถที่จะนำไปใช้ในการศึกษาการทำงานของ PLC โดยการใช้คำสั่งพื้นฐานได้เป็นอย่างดี

PROGRAMMABLE LOGIC OCNTROLLER SIMULATOR

PISSANU PROMWONG

PATCHANON KIRDPIPAT

SOMCHAI CHAOVING

SUPHAN KULLAPANIT ADVISOR

1994

Abstract

PLC (Programmable Logic Controller),Almost be Written by using Bulean language that came from writing Ladder Diagram to monitor be worked of control circuit in industry such as PLC of OMRON model C20 etc.

This thesis was built PLC Simulator inculded program that monitor at Micro Computer, input and output controller by throughing RS-232c , using C language and Assembly language be written so input and output controller were planning and building by using material that could be found in domestic country. This PLC Simulator could be written in 13 basic Bulean language and be indicated in Ladder Diagram.

This PLC Simulator be worked by using the keyboard of PC instead of input and display output at PC ,so be written input file that show in Ladder Diagram to 30,000 steps if there were changed each input status. Input and output controller will get order from program by throughing RS-232c could be worked by connecting input and output equipment,in case of using have 8 output slot that prepared for solid state relay and output relay.

Where as to be tested PLC Simulator could be worked in perfectly objective and could be used in case of studied in PLC by using Basic Bulean Langauge.

สารบัญ

สารบัญรูปภาพ

หน้า

หน้า

หน้า

สารบัญตาราง

บทที่ 1
บทนำ

ความเป็นมาและความสำคัญของโครงงาน

ในปัจจุบันอุตสาหกรรมของประเทศไทย ได้พัฒนาไปอย่างมากมีการใช้เทคโนโลยีสมัยใหม่มาใช้ในอุตสาหกรรม เช่น งานควบคุมการทำงานของเครื่องจักร มีการใช้เครื่องควบคุมแบบโปรแกรมได้ หรือ Programable Logic Controller (PLC) ควบคุมการทำงานของเครื่องจักรการที่จะใช้ให้มีประสิทธิภาพ จึงต้องนำเอา PLC ไปทำการศึกษาและทดลองการทำงานแต่เนื่องจากตัว PLC ในสถาบันการศึกษาต่างๆ ยังมีจำนวนที่น้อยมาก

การที่จะแก้ปัญหาดังกล่าวมาแล้วจะเห็นว่า เครื่องคอมพิวเตอร์ส่วนบุคคลนั้นมีความแพร่หลายมากมีการใช้งานด้านต่างๆ จึงเห็นว่าถ้าจำลองการทำงานของ PLC เป็นโปรแกรมที่จะใช้เครื่องส่วนบุคคลได้ก็จะทำให้นักศึกษาของสถาบันการศึกษาสามารถที่จะศึกษาการเขียนโปรแกรมการทำงานของ PLC ได้ดียิ่งขึ้น

วัตถุประสงค์ของโครงงาน

สร้างโปรแกรมจำลองการทำงานของ เครื่องควบคุมแบบโปรแกรมได้ พร้อมทั้งส่วนอินเตอร์เฟสกับเครื่องคอมพิวเตอร์ ทางพอร์ตอนุกรม RS-232

ขอบเขตของโครงงาน

1.ส่วน HARDWARE

1.1 มีจำนวนช่องอินพุท 8 ช่อง เป็นแรงดันไฟฟ้ากระแสตรง 24 VDC

1.2 มีจำนวนช่องเอาท์พุท 8 ช่อง โดยแบ่งเป็น

1.2.1 ช่องสำหรับเสียบแผ่นวงจรรีเลย์/โซลิดสเตทรีเลย์ จำนวน 8 ช่อง

1.2.2 แผ่นวงจรรีเลย์จำนวน 4 แผ่น สามารถรับภาระได้ 1A ที่แรงดัน 220 VAC

1.2.3 แผ่นวงจรโซลิดสเตทรีเลย์ (Solid State Relay Card) จำนวน 4 แผ่น สามารถรับภาระได้ 450 W ที่แรงดัน 220 V_{AC}

1.3 มีแหล่งจ่ายไฟสำหรับอินพุทเป็นแรงดัน 24 VDC 1 A

1.4 ใช้ CPU Z84C11 เป็นหน่วยประมวลผลกลาง

1.5 หน่วยความจำของโปรแกรมในส่วนผู้ใช้จำนวน 4864 byte

1.6 จำนวนคำสั่งที่ใช้สูงสุด 1216 คำสั่ง

1.7 หน่วยความจำของระบบ 32 Kbyte

1.8 Scan Time Minimum 700 usec

1.9 มีการรายงานสภาวะอินพุท/เอาท์พุททุกๆ 1 วินาที

1.10 มีการรับส่งข้อมูลแบบอนุกรมตามมาตราฐาน RS -232 ความเร็วในการรับส่ง 9600 b/s

1.11 มีจำนวนตัวนับ 16 ตัว

1.12 ค่าของการนับ 0000-9999 หน่วย

1.13 มีจำนวนตัวตั้งเวลา 16 ตัว

1.14 ค่าของการตั้งเวลา 0000-9999 วินาที

2.ส่วน SOFTWARE

เป็นโปรแกรมที่ใช้งานกับเครื่องคอมพิวเตอร์ขนาด 16 บิต พี ซีเอ ที compatible จะแสดงผลแบบ VGA Color

เป็นการจำลองการทำงานของ PLC โดยใช้ฟังก์ชั่นพื้นฐานของคำสั่ง PLC OMRON รุ่น C20 ที่เป็นภาษาบูลลีน คำสั่งที่ใช้มีดังนี้คือ

LD

LD XX

LD YY

LD TR ZZ

LD CNT AA

LD TIM BB

LD-NOT

```
LD-NOT  XX
LD-NOT  YY
LD-NOT  CNT  AA
LD-NOT  TIM  BB
```

AND

```
AND  XX
AND  YY
AND  CNT  AA
AND  TIM  BB
```

AND-NOT

```
AND-NOT  XX
AND-NOT  YY
AND-NOT  CNT  AA
AND-NOT  TIM  BB
```

OR

```
OR  XX
OR  YY
OR  CNT  AA
OR  TIM  BB
```

OR-NOT

```
OR-NOT  XX
OR-NOT  YY
OR-NOT  CNT  AA
OR-NOT  TIM  BB
```

```
AND-LD

OR-LD

CNT     AA #CCCC

TIM     BB #DDDD


OUT

        OUT  YYY

        OUT  TR  ZZ

END
```

เมื่อ XX แทนหมายเลข SW

yyy แทนหมายเลข OUTPUT

zz แทนหมายเลข Tempolarly Relay

AA แทนหมายเลขตัวนับ

BB แทนหมายเลข ตัวตั้งเวลา

CCCC แทนจำนวนที่ต้องการให้นับ

DDDD แทนจำนวนเวลาที่ต้องการตั้ง

แสดงการทำงานของโปรแกรม PLC ที่ผู้ใช้เขียนขึ้นทางจอมอนิเตอร์

แสดงจำลองการทำงานด้วย Timing Diagram โดยแสดงได้ 20 ลูปต่อหนึ่งหน้า

จำนวน Loop สูงสุดที่จะจำลองการทำงานด้วย Timing Diagram 30,000 Loop

แสดง Ladder Diagram ทางจอมอนิเตอร์ได้จำนวน 20 หน้า

จำนวนเงื่อนไขที่แสดงติดต่อกันได้สูงสุด 9 เงื่อนไข

จำนวนคำสั่งที่เขียนได้สูงสุด 500 บรรทัด

มีจำนวนอินพุท 8 อินพุท (00-07)

มีจำนวนเอาท์พุท 8 เอาท์พุท (500-507)

ตัวตั้งเวลา 16 ตัว (00-15) ค่าเวลา 0000-9999 วินาที

ฐานเวลา 1 วินาที

ตัวนับ 16 ตัว (00-15) ค่าการนับ 0000-9999 หน่วย

แสดงภาวะการทำงานของชุด Controller (Hardware)

มีการรับส่งข้อมูลกับชุด Controller แบบอนุกรมตามมาตราฐาน RS-232

จัดทำคู่มือการใช้งานของ เครื่องจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้ 1 ชุด

<u>อุปรณีที่ใช้ต่อร่วมเป็น PLC SIMULATOR</u>

1. เครื่องคอมพิวเตอร์ PC/AT Compatible แสดงผลแบบ VGA COLOR
2. แผ่นเก็บข้อมูลโปรแกรม PLCSIM
3. สายรับส่งข้อมูลแบบอนุกรมตามมาตราฐาน RS-232
4. ชุด Interface กับ Computer

<u>ประโยชน์ที่คาดว่าจะได้รับ</u>

1.ใช้เป็นเครื่องมือในการศึกษาและเขียนโปรแกรมของ PLC

2.ใช้เป็นชุดทดลองการทำงานของ PLC

บทที่ 2

ความรู้เบื้องต้นของเครื่องควบคุมแบบโปรแกรมได้

(PROGRAMMABLE LOGIC CONTROLLER)

ความหมายของเครื่องควบคุมแบบโปรแกรมได้

PLC (PROGRAMMABLE LOGIC CONTROLLER)

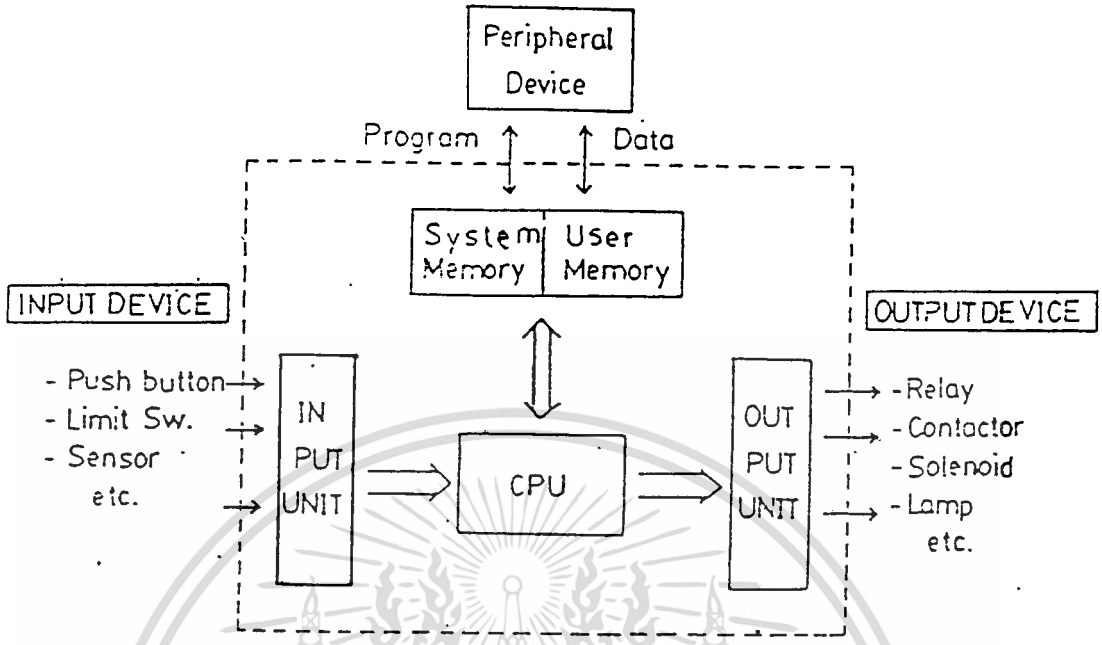
เครื่องควบคุมแบบโปรแกรมได้ หรือ PLC เป็นเครื่องมือควบคุมที่สามารถกำหนดโปรแกรมการทำงานได้ใช้ควบคุมเครื่องจักร หรือ กระบวนการใช้ทำงานตามโปรแกรมคำสั่งของผู้ใช้และข้อมูลต่างๆที่ได้รับจากอินพุท/เอาท์พุทของ PLC การทำงานของ PLC จะเป็นไปได้ทั้งการทำงานตามช่วงเวลา ตามลำดับขั้นตอนเป็นต้น

โครงสร้างและส่วนประกอบของ PLC

PLC มีส่วนประกอบที่คล้ายกับคอมพิวเตอร์หรืออาจกล่าวได้ว่า PLC เป็นคอมพิวเตอร์เฉพาะงานประเภทหนึ่ง ดังนั้น โครงสร้างโดยทั่วๆไปของ PLC จะประกอบด้วย Hardware และ Software

โครงสร้างทาง Hardware และ software ของ PLC แบ่งออกได้เป็น 4 ส่วนด้วยกันคือ

1. หน่วยประมวลผลกลาง (CPU)
2. หน่วยความจำ (Memory)
3. หน่วยอินพุท/เอาท์พุท (Input/Output Unit)
4. หน่วยติดต่อทั่วไป (Peripheral Device)



รูปที่ 2.1 โครงสร้างของ PLC

1.หน่วยประมวลผลกลาง (CPU:Central Processor Unit)

หน่วยประมวลผลกลาง หมายถึง ส่วนที่ทำหน้าที่ควบคุมการทำงานของ PLC โดยทั่วไปจะใช้ไมโครโปรเซสเซอร์ 8 บิต เป็นตัวประมวลผล ปกติหน้าที่ของ CPU คือรับข้อมูลอินพุทเข้ามาทำการประมวลผลแล้วส่งผลที่ได้ออกไปยังเอาท์พุท

2. หน่วยความจำ (Memory unit)

หน่วยความจำเป็นองค์ประกอบหนึ่งที่สำคัญของระบบ เพราะใช้เป็นที่เก็บโปรแกรมและข้อมูลขนาดของหน่วยความจำ จะเป็นตัวกำหนดความสามารถของระบบปกติมักจะมีหน่วยวัดเป็นจำนวน Step หรือ บรรทัดของโปรแกรม PLC แบ่งหน่วยความจำที่สำคัญออกเป็น 2 ส่วนคือ

2.1 หน่วยความจำของระบบ(System Memory) เก็บโปรแกรมบริหารระบบและข้อมูลของระบบ เป็นหน่วยความจำที่ผู้ใช้ไม่สามารถที่จะเปลี่ยนแปลง หรือแก้ไขข้อมูลภายในหน่วยความจำส่วนข้อมูลของระบบได้ หรือตรวจสอบสภาพ การทำงานของ PLC

2.2 หน่วยความจำผู้ใช้ (User Memory) เก็บโปรแกรมผู้ใช้ข้อมูลของหน่วยอินพุท /เอาท์พุท และอุปกรณ์ภายใน

3. หน่วยอินพุท/เอาท์พุท (Input/Output Unit)

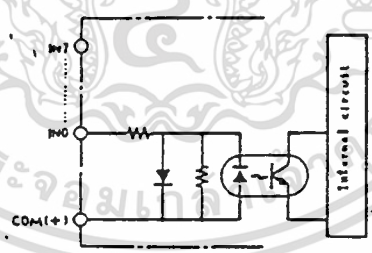
3.1 หน่วยอินพุท (Input Unit)

หน่วยอินพุท ทำหน้าที่เชื่อมต่อระหว่าง CPU กับอุปกรณ์ภายนอกโดยรับค่าสภาวะ หรือ ปริมาณทางกายภาพต่างๆจากอุปกรณ์ตรวจเช็ค (Sensor) ของเครื่องจักรหรือกระบวนการ เช่น Limit Switch ,Push Button Switch,อุณหภูมิระดับแรงดันกระแส หรือ อื่นๆส่งไปยัง CPU เพื่อประมวลผลตามโปรแกรมคำสั่งของผู้ใช้ ปกติหน้าที่ของหน่วยอินพุท คือ

1. แปลงระดับสัญญาณที่เข้าไปให้เป็นสัญญาณที่เหมาะสมกับระบบการทำงานของ CPU

2.แบ่งสัญญาณภายนอกและภายในออกจากกัน (Isolate) เพื่อเป็นการป้องกันไม่ให้หน่วยประมวลผลเสียหายเมื่อหน่วยอินพุทลัดวงจร

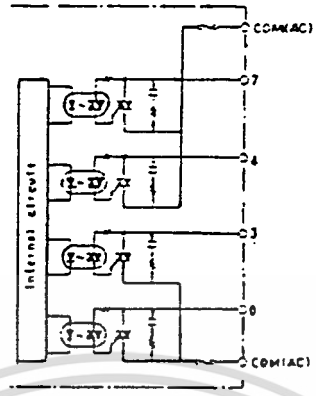
3. แก้ปัญหาการสั่นสะเทือนของหน้าสัมผัส



รูปที่ 2.2 วงจรอินพุทของ PLC

3.2 หน่วยเอาท์พุท (Output Unit)

หน่วยเอาท์พุท ทำหน้าที่รับสัญญาณที่ได้จากการประมวลผลไปขยายสัญญาณออกให้มีขนาดใหญ่พอที่จะไปขับอุปกรณ์ภายนอก เช่น วาวล์ หลอดไฟ และอื่นๆนอกจากนั้นยังทำหน้าที่ในการแยกสัญญาณภายในและภายนอกออกจากกันเพื่อป้องกันการเสียหายของเครื่อง PLC



รูปที่ 2.3 วงจรเอาท์พุทของ PLC

4. หน่วยอุปกรณ์ภายนอก (Peripheral Device)

เป็นอุปกรณ์ต่างๆที่อำนวยความสะดวกในการพัฒนาโปรแกรม สามารถที่จะใช้ PLC ร่วมกันหลายตัว

<u>หน้าที่ของอุปกรณ์ภายนอก</u>

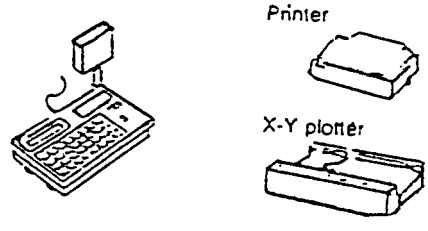
1.ใช้ป้อนโปรแกรมเข้าไปในหน่วยความจำของระบบ

2.ใช้ในการแก้ไขโปรแกรม

3.ใช้ในการเก็บรักษาโปรแกรม

4.ใช้ในการพิมพ์โปรแกรม

5.ใช้แสดงสภาวะการควบคุม



รูปที่ 2.4 อุปกรณ์ติดต่อภายนอกของ PLC

ลักษณะการทำงานตามโปรแกรม PLC

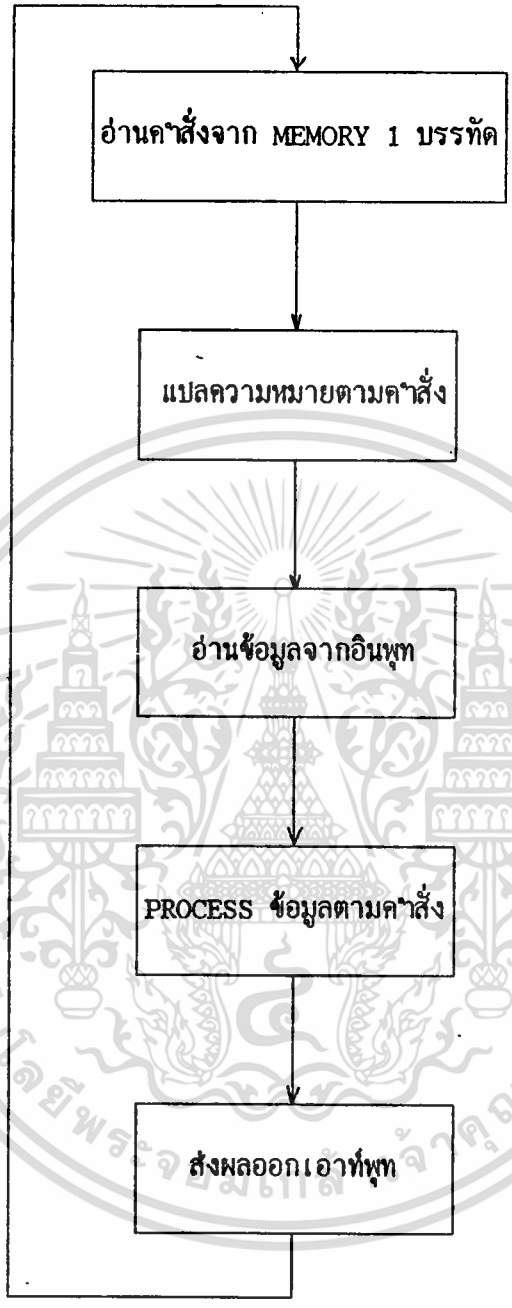
PLC เป็นอุปกรณ์ที่ใช้ในการควบคุมเครื่องจักรหรือกระบวนการผลิตต่างๆนั้นภายในมี CPU ซึ่งเป็นสมองสั่งการและมีส่วนที่เป็นอินพุทส่วนเอาท์พุทจะใช้ต่อออกไปควบคุมการทำงานของอุปกรณ์ได้ทันที เช่น ตัวตรวจวัด และ สวิทช์ต่างๆ จะต่อเข้าที่อินพุท ส่วนเอาท์พุทจะใช้ต่อออกไปควบคุมการทำงานของอุปกรณ์ หรือ เครื่องจักรที่เป็นเป้าหมาย เราสามารถสร้างวงจรหรือแบบของการควบคุมได้โดยการป้อนเป็นโปรแกรมสั่งงานเข้าไปใน PLC โปรแกรมนี้เองจะทำหน้าที่เหมือนวงจรรีเลย์ ตัวตั้งเวลา และ ตัวนับที่เคยใช้ตามปกติเมื่อกดปุ่มบังคับให้ PLC ทำงานมันจะทำงานได้เหมือนวงจรของรีเลย์ที่เราป้อนโปรแกรมเข้าไป

PLC จะสร้างอุปกรณ์ต่างๆ ภายในเองเช่น รีเลย์ ตัวตั้งเวลาและตัวนับได้โดยซอฟแวร์ ซึ่งไม่มีตัวตนในรูปของวัตถุแต่จะปรากฏในรูปของฟังก์ชั่นการทำงานที่ตรงกับของจริงนอกจากนี้การต่อสายเชื่อมโยงอุปกรณ์เหล่านี้เข้าหาด้วยกันเป็นวงจรที่ทำโดยซอฟแวร์ทั้งสิ้น เราสามารถแก้วงจรหรือเพิ่มเติมวงจรได้เพียงแต่แก้ไขโปรแกรมวงจรเท่านั้น

PLC ในปัจจุบันใช้ในเฉพาะส่วนที่เป็นวงจรควบคุมเท่านั้น ส่วนวงจรเพาเวอร์ที่ต้องใช้กระแสไหลผ่านมากๆ ยังคงเป็นเพาเวอร์รีเลย์ เพราะเอาท์พุทของ PLC จะใช้งานที่กระแส 1 A ดังนั้น PLC จะทำหน้าที่แทนวงจรควบคุมหรือแทนรีเลย์ไดอะแกรม ในการควบคุมด้วยระบบรีเลย์นั่นเอง

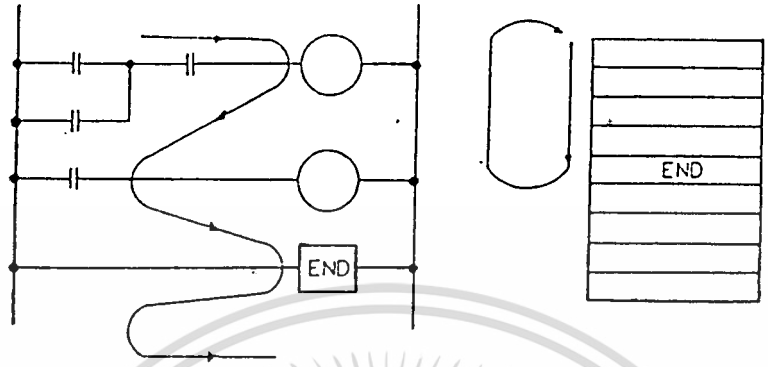
การนำ PLC มาแทนวงจรการควบคุมนั้นจะต้องเขียนโปรแกรมคำสั่งซึ่งมีลักษณะการทำงานเหมือนกับวงจรควบคุมของระบบรีเลย์ทุกประการ โดยในที่นี้อาจจะเปลี่ยนจากรีเลย์ไดอะแกรมเป็น แลดเดอร์ไดอะแกรม

หน่วยประมวลผลกลาง หรือ CPU จะทำหน้าที่ควบคุมการทำงานของ PLC โดยปกติหน้าที่ของ PLC คือรับอินพุทเข้ามาทำการประมวลผล แล้วส่งผลที่ได้ไปยังหน่วยเอาท์พุท จากนั้นก็วนกลับไปรับข้อมูลอินพุทเข้ามาอีกแล้วทำซ้ำๆ ในลักษณะนี้ไปเรื่อยๆ เรียกว่าการสแกน และการทำงานของ CPU จะอยู่ภายใต้การควบคุมของโปรแกรมที่ผู้ใช้ป้อนเข้าไป



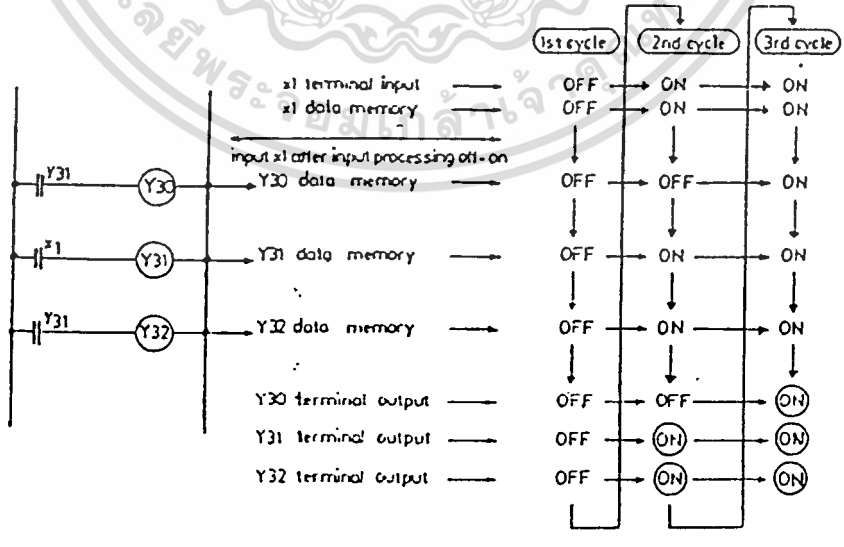
รูปที่ 2.5 แสดงการทำงานของ PLC 1 Loop

การสแกน CPU ประกอบด้วยการรับค่าสภาวะของอุปกรณ์ภายนอกจากอินพุท/เอาท์พุทมาเก็บไว้ในหน่วยความจำ หลังจากนั้นจะนำโปรแกรมควบคุมที่ผู้ใช้เขียนขึ้นมาปฏิบัติทีละคำสั่ง โดยเริ่มจากคำสั่งแรกจนสิ้นสุดโปรแกรมในหน่วยความจำ



รูปที่ 2.6 การสแกนโปรแกรมในคำสั่ง PLC

การปฏิบัติโปรแกรม ทำให้สภาวะเอาท์พุทจุดใดจุดหนึ่งเปลี่ยนแปลง ผลดังกล่าวก็จะบันทึกไว้ในหน่วยความจำก่อนเมื่อปฏิบัติตามโปรแกรมของผู้ใช้เรียบร้อยแล้วจึงนำผลการเปลี่ยนแปลงครั้งสุดท้ายส่งออกไปที่หน่วยเอาท์พุท แล้ว PLC จึงเริ่มต้นการสแกนใหม่



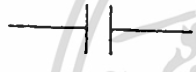
รูปที่ 2.7 ตัวอย่างการเปลี่ยนแปลงของหน่วยอินพุท/เอาท์พุท

ภาษาที่ใช้สำหรับ PLC

1.ภาษาแลดเดอร์

ภาษาแลดเดอร์ประกอบด้วยสัญลักษณ์หน้าสัมผัส มีลักษณะคล้ายวงจรรีเลย์ การเขียนโปรแกรมภาษาแลดเดอร์จากวงจรรีเลย์จึงทำได้ง่าย คำสั่งภาษาแลดเดอร์ประกอบด้วยสัญลักษณ์หน้าสัมผัสและขดลวดเพื่อแสดงเงื่อนไขการควบคุมระหว่างอุปกรณ์หน่วยอินพุท/เอาท์พุท และอุปกรณ์ภายใน. การเขียนโปรแกรมต้องระบุตำแหน่งหรือหมายเลขของอุปกรณ์เหล่านี้ให้ถูกต้องและตรงกันทุกครั้ง ซึ่งตำแหน่งหรือหมายเลขของอุปกรณ์เหล่านี้ได้มาจากข้อกำหนดของเครื่องที่ใช้อยู่ในระยะแรก

สัญลักษณ์พื้นฐานที่ใช้แทนรีเลย์ของภาษาแลดเดอร์



แทนหน้าสัมผัสแบบปกติเปิด (NO) หมายถึงการสวิทช์หรือหน้าสัมผัสของอุปกรณ์อินพุท/เอาท์พุทและอุปกรณ์ภายใน ในซึ่งปกติเปิดวงจรไม่อนุญาติให้กระแสไหลผ่าน

แทนหน้าสัมผัสแบบปกติปิด (NC) หมายถึงสวิทช์หรือหน้าสัมผัสของอุปกรณ์อินพุท/เอาท์พุท หรืออุปกรณ์ภายในซึ่งปกติปิดวงจรไฟฟ้าทำให้กระแสครบวงจร

แทนเอาท์พุทซึ่งปกติไม่ทำงาน หมายถึง อุปกรณ์เอาท์พุท เช่น หลอดไฟฟ้า และ ขดลวดรีเลย์

แทนเอาท์พุทซึ่งปกติทำงาน หมายถึงอุปกรณ์เอาท์พุท เช่น หลอดไฟฟ้า และ ขดลวดรีเลย์

ภาษามูลลีน

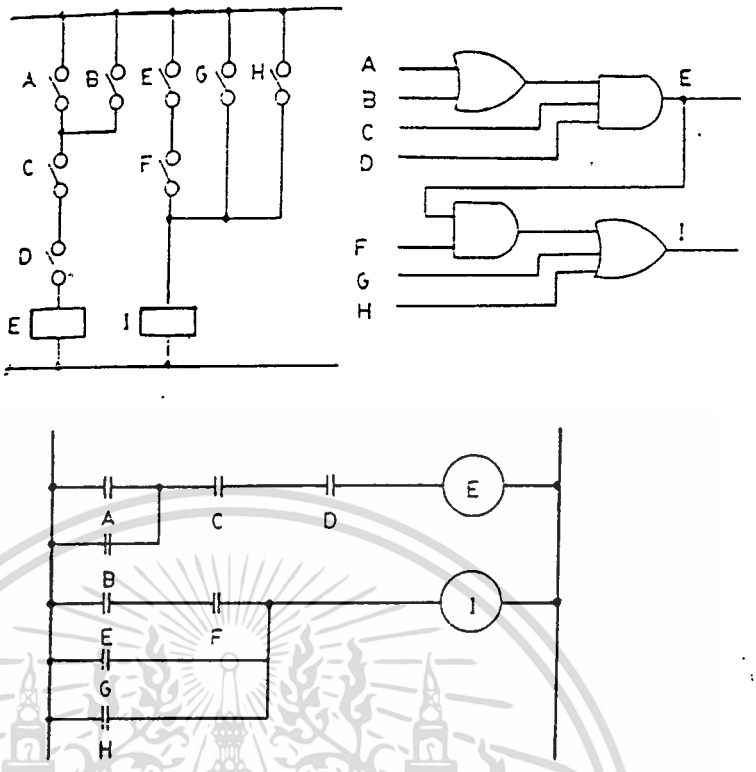
ภาษามูลลีนเป็นภาษาพื้นฐานของ PLC เช่นเดียวกับภาษาแลดเดอร์คำสั่งในภาษามูลลีนมีลักษณะคล้ายกับสัญลักษณ์ ของพีชคณิตมูลลีน ในปัจจุบันคำสั่ง ภาษามูลลีนประกอบด้วยกลุ่มคำสั่ง เช่นเดียวกับภาษาแลดเดอร์ คือวงจรรีเลย์และปฏิบัติลอจิก การหน่วงเวลา และนับจำนวน การคำนวณทางคณิตศาสตร์ และคำสั่งควบคุมโปรแกรม

ตัวอย่างการเขียนโปรแกรมคำสั่งมูลลีน

```
LD 00
AND 01
OR 02
OUT 500
END
```

หลักการเขียน LADDER DIAGRAM

LADDER DIAGRAM เป็นระบบการโปรแกรมที่นิยมส่วนมากแบบหนึ่ง ลักษณะของ Ladder Diagram แตกต่างไปจาก Logic Diagramโดยหลักของ Diagram ยังคงยึดหลักการเขียนของรีเลย์อยู่ เพื่อให้ผู้ปฏิบัติงานด้านการควบคุมในยุคของรีเลย์ ไม่ต้องเปลี่ยนแปลงแนวคิดความเข้าใจกับ Diagram ของระบบเพียงแต่เรียนรู้วิธีการโปรแกรมเพิ่มขึ้นเท่านั้น



รูปที่ 2.8 แสดงการเปรียบเทียบระหว่าง Logic,Relay และ Ladder Diagram

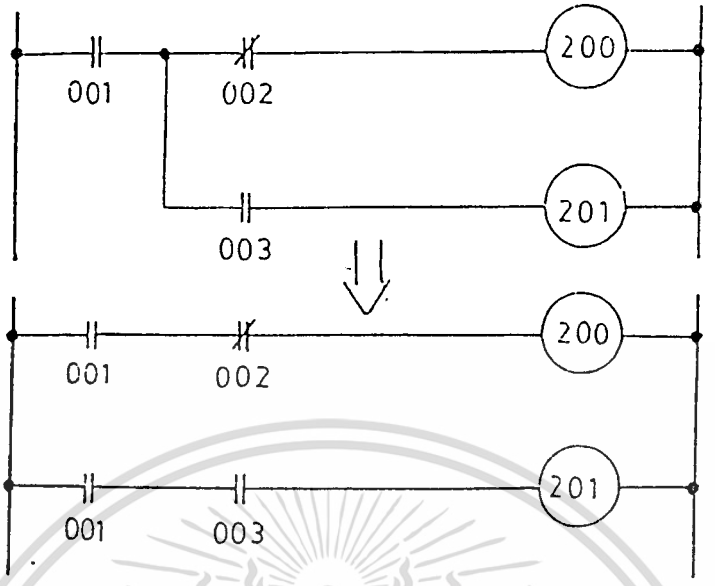
<u>หลักการในการเขียน LADDER DIAGRAM</u>

1.หมายเลขกำกับหน้าสัมผัส หรือขดลวดของอินพุท/เอาท์พุท รีเลย์ภายใน ตัวตั้งเวลา และตัวนับของ Ladder Diagram จะนำมาจากข้อกำหนดของเครื่อง PLC

Name	Allocation No.	No. of Points
Input	0-7, 10-17, 20-27, 30-37, 40-47, 50-57, 60-67, 70-77	64
Output	200-207, 210-217, 220-227, 230-237, 240-247, 250-257, 260-267, 270-277	64
Internal Relay	400-407, 410-417, 420-427, 430-437, 440-447, 450-457, 460-467, 470-477, 480-487, 490-497, 500-507, 510-517, 520-527, 530-537, 540-547, 550-557, 560-567, 570-577, 580-587, 590-597, 600-607, 610-617, 620-627, 630-637, 640-647, 650-657, 660-667, 670-677, 680-687, 690-697	240
Special Internal Relay	700-707, 710-717	16
Timer	0-79 (When using arithmetic operand: 1000-1079)	80
Counter	0-44 (When using arithmetic operand: 900-944)	45
Reversible Counter	45 (dual pulse), 46 (up/down selection) (When using arithmetic operand: 945 & 946)	1 each
Shift Register	0-127 (bidirectional)	128
Single Output	0-95	96
Data Register	800-899 (DR0-99)	100

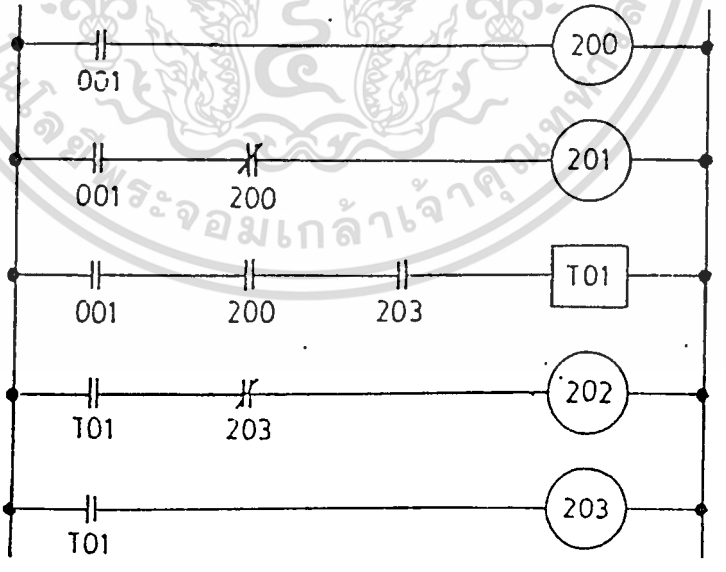
รูปที่ 2.9 แสดงหมายเลขกำกับอุปกรณ์ของ PLC เครื่องหนึ่ง

2. การเขียนรโปรแกรมควรเขียนให้อ่านง่ายและเข้าใจง่าย ไม่ควรเขียนรโปรแกรมให้สั้นแต่ซับซ้อน เพื่อประหยัดหน่วยความจำเพียงเล็กน้อย



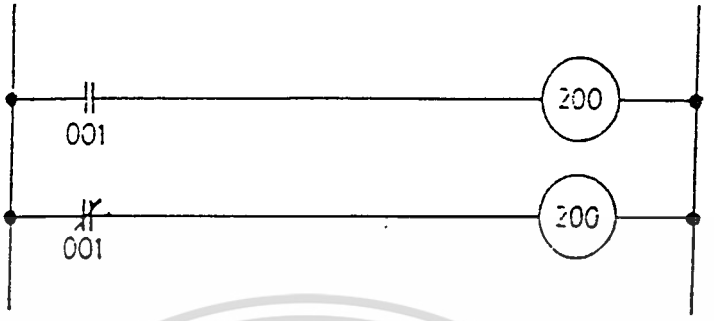
รูปที่ 2.10 แสดง LADDER DIAGRAM

3.หน้าสัมผัสของอินพุท/เอาท์พุท รีเลย์ภายใน ตัวตั้งเวลาและตัวนับหมายเลขเดียวกัน สามารถเรียกใช้ในโปรแกรมกี่ครั้งก็ได้ไม่จำกัดจำนวน



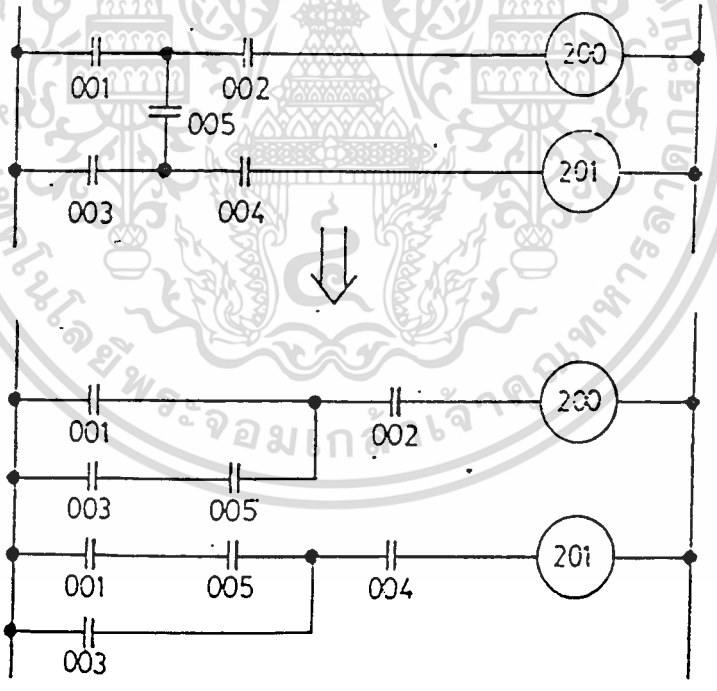
รูปที่ 2.11 LADDER DIAGRAM

4.ส่งสัญญาณควบคุมซ้ำกันมากกว่าหนึ่งครั้งไปที่ขดลวดเอาท์พุทรีเลย์ หรือ รีเลย์ภายในหมายเลขเดียวกันไม่ได้ (นอกจากมีฟังก์ชั่นพิเศษเข้ามาช่วย)



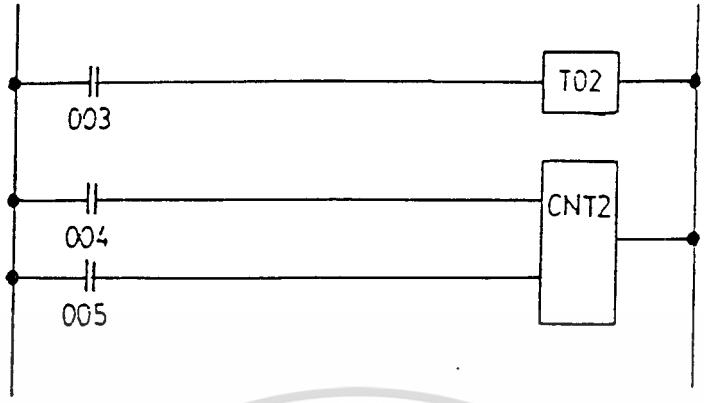
รูปที่ 2.12 แสดง LADDER DIAGRAM

5.สัญญาณควบคุมไหลจากซ้ายไปขวาทิศทางเดียวไม่ย้อนกลับ



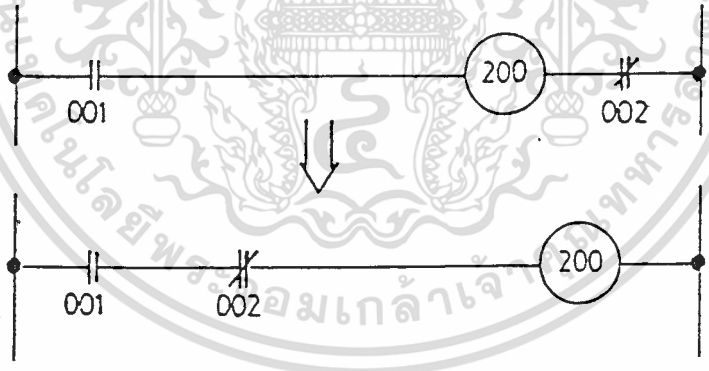
รูปที่ 2.13 แสดง LADDER DIAGRAM

6.ตัวตั้งเวลาหรือตัวนับหมายเลขเดียวกัน จะถูกเรียกใช้ในโปรแกรมมากกว่าหนึ่งครั้งไม่ได้ และ PLC บางเครื่องไม่อนุญาติให้ใช้ตัวตั้งเวลาและตัวนับหมายเลขเดียวกันในโปรแกรม



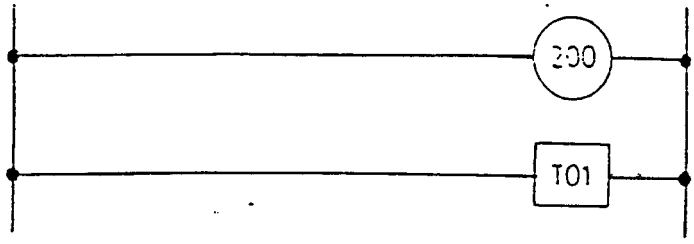
รูปที่ 2.14 แสดง LADDER DIAGRAM

7. วางตำแหน่งหน้าสัมผัสหลังขดลวดรีเลย์ไม่ได้

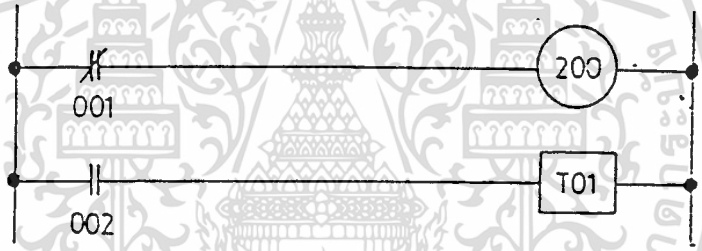


รูปที่ 2.15 แสดง LADDER DIAGRAM

8. ขดลวดของเอาท์พุทรีเลย์หรือรีเลย์ภายในต่อโดยตรงกับบัส ด้านซ้ายมือไม่ได้ (ถ้าไม่จำเป็นต้องต่อหน้าสัมผัสที่มีสภาวะ on ระหว่างบัสกับขดลวดรีเลย์โดยใช้หน้าสัมผัสของขดลวดรีเลย์ภายในที่มีสภาวะ on ตลอดเวลา)

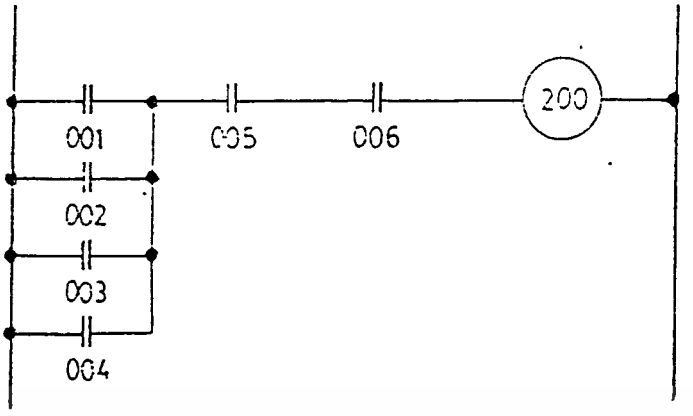


รูปที่ 2.16 แสดง LADDER DIAGRAM



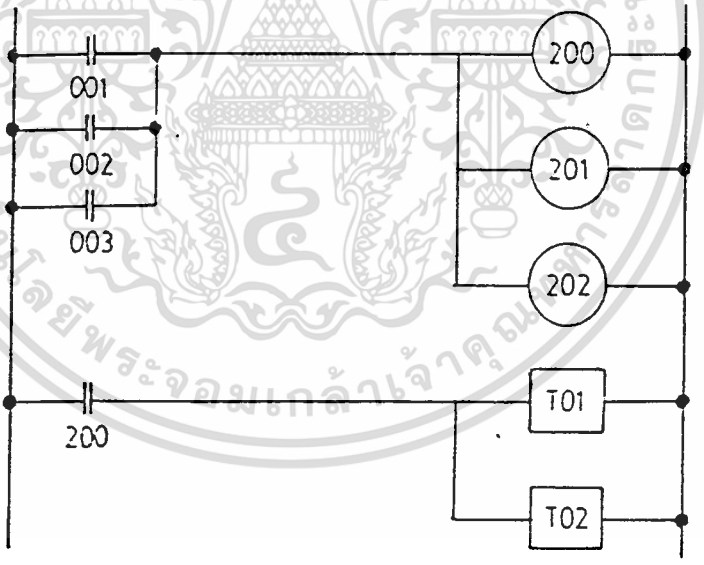
รูปที่ 2.17 แสดง LADDER DIAGRAM

9.หน้าสัมผัสของอินพุท/เอาท์พุท รีเลย์ภายใน ตัวตั้งเวลาและตัวนับ จะต่อขนานหรืออนุกรมกันจำนวนมากเท่าใดก็ได้ไม่จำกัดจำนวน



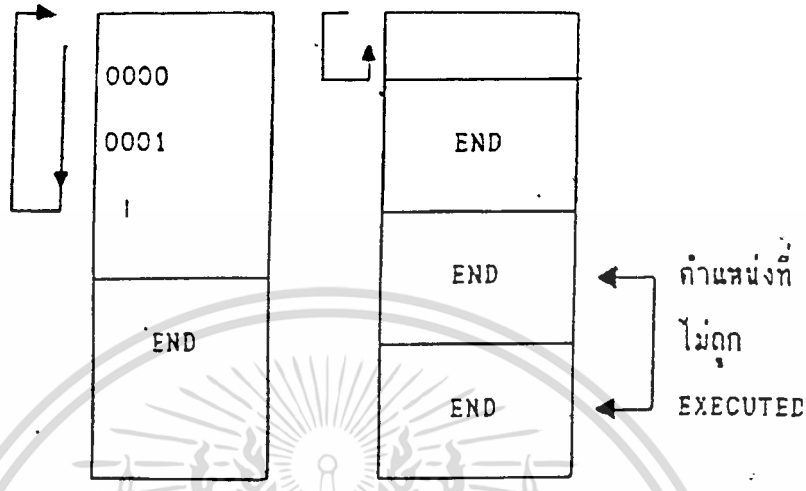
รูปที่ 2.18 แสดง LADDER DIAGRAM

10. ขดลวดของเอาท์พุทรีเลย์ รีเลย์ภายในและตัวตั้งเวลาสามารถนำมาต่อขนานกันได้



รูปที่ 2.19 แสดง LADDER DIAGRAM

11. โปรแกรมที่เขียนขึ้น CPU จะทำการ Executed จากบรรทัดแรกถึงคำสั่ง End ที่เป็นคำสั่งแรกซึ่งถูกใช้ โดยที่ End อาจมีหลายตำแหน่งก็ได้ซึ่งอาจจะต้องมีคำสั่งหรือฟังก์ชั่นพิเศษเข้ามาช่วยที่เป็นเช่นนี้เพื่อวัตถุประสงค์สำหรับการทดลองโปรแกรมทำให้ง่ายต่อการตรวจสอบและแก้ไขโปรแกรม



รูปที่ 2.20 แสดง การทำงานตามโปรแกรมของ CPU

การเขียนโปรแกรมคำสั่งบูลลีนจาก LADDER DIAGRAM

จากการที่ภาษา Ladder มีลักษณะโครงสร้างคล้ายกับการควบคุมด้วยระบบรีเลย์ จึงทำให้ง่ายต่อการทำความเข้าใจและเขียนโปรแกรมแต่เครื่อง PLC ตั้งแต่ขนาดกลางลงมาซึ่งใช้ในปัจจุบัน จะนิยมใช้คำสั่งบูลลีนในการเขียนโปรแกรมการควบคุมกระบวนการต่างๆ ซึ่งในการเขียนโปรแกรมนั้นเป็นการเทียบมาจาก Ladder Diagram นั่นเอง โครงสร้างของโปรแกรมคำสั่งบูลลีนประกอบด้วยส่วนต่างๆที่สำคัญ 3 ส่วน คือ

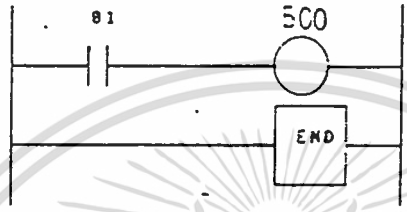
1.หมายเลขกำหนดบรรทัดของโปรแกรม

2.คำสั่ง

3.หมายเลขกำกับอุปกรณ์และหน้าสัมผัสต่างๆ

ตัวอย่างการเขียนโปรแกรมคำสั่งบูลลีนจาก Ladder Diagram

LADDER DIAGRAM



PROGRAM

```
LD 01
OUT 500
END
```

คำสั่งพื้นฐานของ PLC

คำสั่งพื้นฐานเป็นคำสั่งที่ใช้ในการควบคุม On-Off และแบบซีเควนซ์เชี่ยลเท่านั้นเพราะง่ายต่อการทำความเข้าใจและเขียนโปรแกรม ภาษาที่นิยมใช้ก็คือภาษา Ladder และภาษาบูลลีน ซึ่งจะกล่าวในลักษณะที่ควบคู่กันไป สำหรับคำสั่งที่ใช้ในเครื่อง PlC Simulater จะประกอบด้วยกลุ่มคำสั่ง วงจรรีเลย์และปฏิบัติลอจิก การหน่วงเวลาและการนับจำนวน

คำสั่ง : LOAD

สัญญลักษณ์ :

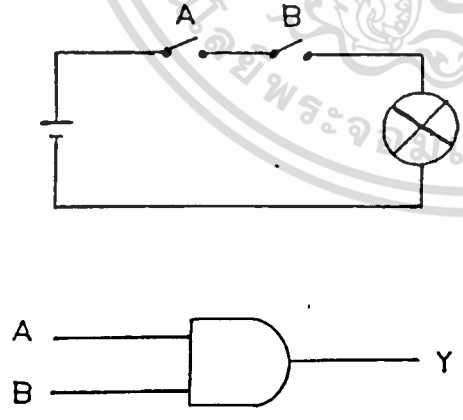
ความหมาย : เป็นการนำค่าสภาวะของอินพุท เอาท์พุท ตัวตั้งเวลา ตัวนับ หรือรีเลย์ภายในที่กำหนดเข้ามา

คำสั่ง : AND

สัญญลักษณ์ :

ความหมาย : เป็นการนำค่าสภาวะของอินพุท เอาท์พุท ตัวตั้งเวลา ตัวนับ หรือรีเลย์ภายในที่กำหนดเข้ามากระทำลอจิก AND กับค่าสภาวะปัจจุบัน

การปฏิบัติการลอจิก AND :



INPUT		OUTPUT
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

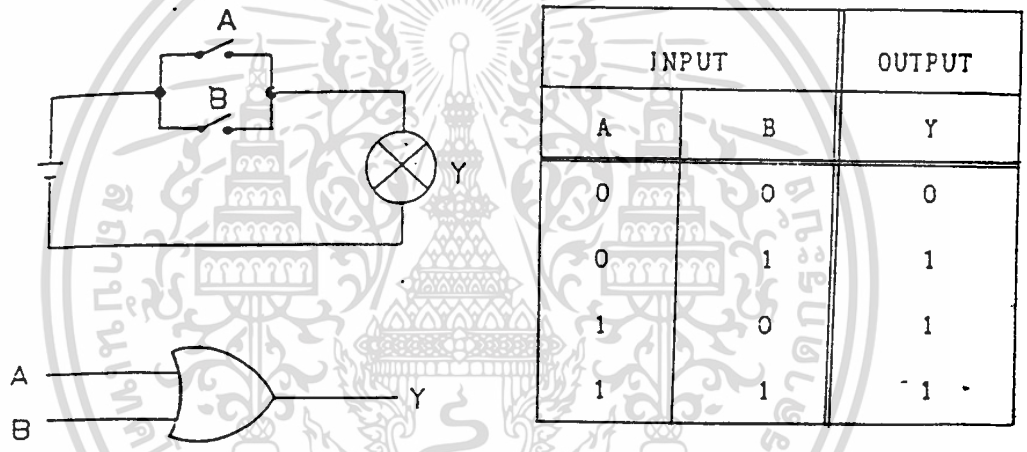
รูปที่ 2.21 แสดงการปฏิบัติการลอจิก AND

การปฏิบัติการลอจิก AND:เอาท์พุท Y จะมีสภาวะลอจิก "1" ถ้าทุกอินพุทมีสภาวะเป็นลอจิก "1" ทั้งหมด

คำสั่ง : OR

สัญญลักษณ์ :

ความหมาย : เป็นการนำสภาวะอินพุท เอาท์พุท ตัวตั้งเวลา ตัวนับ หรือ รีเลย์ภาย ในที่กำหนดเข้ามากระทำลอจิก OR กับค่าภาวะปัจจุบัน



INPUT		OUTPUT
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

รูปที่ 2.22 แสดงการปฏิบัติการลอจิก OR

การปฏิบัติการลอจิก OR:เอาท์พุท Y จะมีสภาวะลอจิก "1" ถ้าอินพุทเพียง 1 อินพุทมีสภาวะเป็นลอจิก "1"

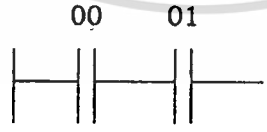
การใช้คำสั่ง LOAD,AND และ OR

คำสั่ง	หมายเลขกำหนดตำแหน่งของ
LOAD	INPUT OUTPUT STATUS INTERNAL RELAY

คำสั่ง	หมายเลขกำหนดตำแหน่งของ
AND OR	SPECIAL RELAY TIMER STATUS COUNTER STATUS

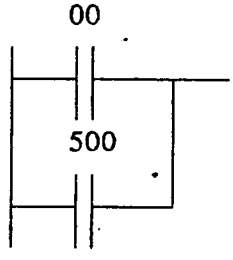
ตัวอย่าง :

LD 00



LD 00

AND 01



LD 00

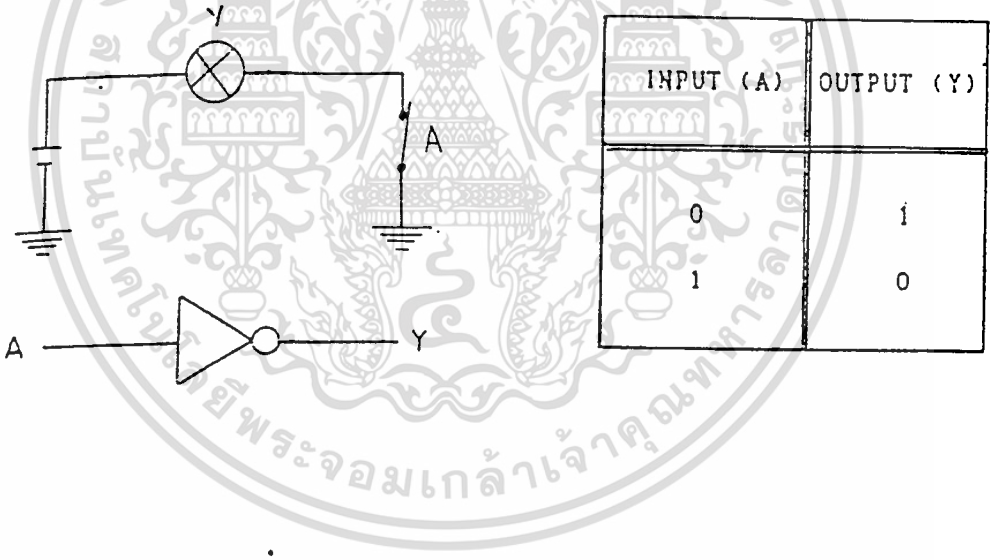
OR 500

คำสั่ง : NOT

สัญญลักษณ์ :

ความหมาย : เป็นการกระทำลอจิก NOT กับค่าสภาวะปัจจุบันโดยปกติแล้วคำสั่งนี้หมายถึงหน้าสัมผัสปกติปิดของอุปกรณ์ต่างๆ ของ PLC โดยจะใช้ร่วมกับ LOAD,AND และ OR

การปฏิบัติการลอจิก NOT:

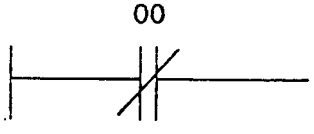


INPUT (A)	OUTPUT (Y)
0	1
1	0

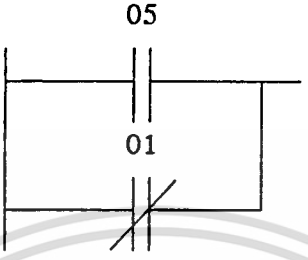
รูปที่ 2.23 การปฏิบัติการทางลอจิก NOT

ปฏิบัติการลอจิก NOT: เอาท์พุทจะมีสภาวะตรงข้ามกับสภาวะอินพุทดังผลตามตารางข้างบน

ตัวอย่าง :



LOAD-NOT 00

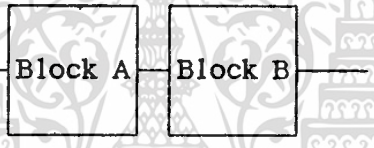


LOAD 05

OR-NOT 01

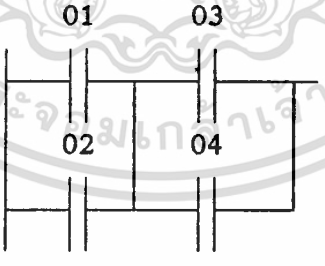
คำสั่ง : AND-LOAD

สัญญลักษณ์ :



ความหมาย : เป็นการนำสภาวะที่เก็บรักษาไว้มากระทำลอจิก AND กับค่าสภาวะปัจจุบัน

ตัวอย่าง :



LD 01

OR 02

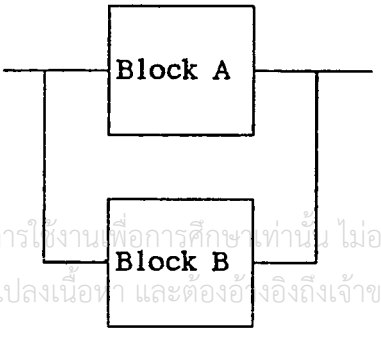
LD 03

LD 04

AND-LD

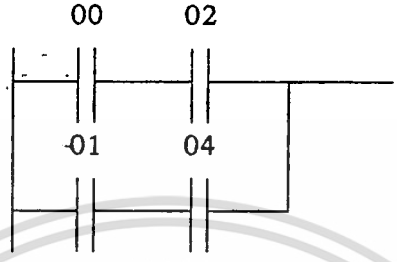
คำสั่ง : OR-LOAD

สัญญลักษณ์ :



ความหมาย : เป็นการนำค่าสภาวะที่เก็บรักษาไว้มากระทำ ลอจิก OR กับค่าสภาวะปัจจุบัน

ตัวอย่าง :



LD 00
AND 02
LD 03
AND 04
OR-LD

คำสั่ง : OUT

สัญญลักษณ์ :

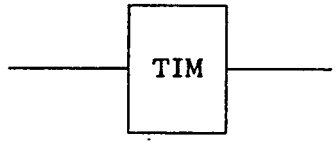
ความหมาย : เป็นการให้ค่าสภาวะแก่อุปกรณ์ทางเอาท์พุทต่างๆ โดยทั่วไปจะได้แก่ OUTPUT, INTERNAL RELAY

ตัวอย่าง :

LD 00
OUT 500

คำสั่ง : TIMER

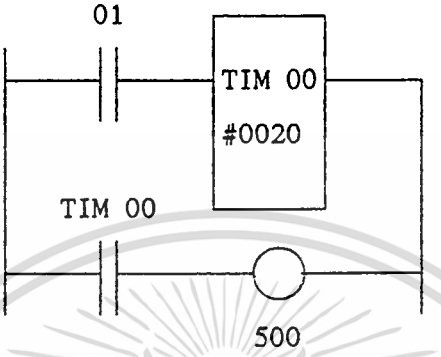
สัญญลักษณ์ :



ความหมาย : เป็นการใช้เรียกตัวตั้งเวลาซึ่งสามารถหน่วงเวลาการทำงานหรือกำหนดค่าเวลาได้ซึ่งจะขึ้นอยู่กับขีดความสามารถของ PLC จะทำหน้าที่แทนตัวตั้งเวลาของวงจรรีเลย์ใน การปิด หรือ เปิดวงจรไฟฟ้า เมื่อเวลาผ่าน

ไปช่วงเวลาหนึ่งตามที่ได้ตั้งไว้ตัวตั้งเวลาจะนับสัญญาณนาฬิกาของ CPU แล้วจะไปทำการปิดหรือเปิดรีเลย์ตามต้องการ

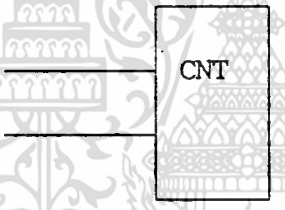
ตัวอย่าง :




```
LD   01
TIM  00
     #0020
LD   TIM 00
OUT  500
```

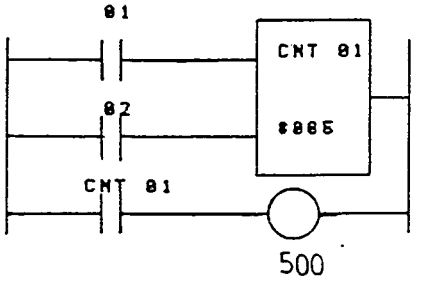
คำสั่ง : COUNTER

สัญญลักษณ์ :



ความหมาย : เป็นการเรียกใช้ตัวนับ ซึ่งสามารถทำงานแทนตัวนับของวงจรรีเลย์ในการปิดหรือเปิดวงจรไฟฟ้า เมื่อนับได้ตามจำนวนที่ต้องการ ตัวนับจะทำการตรวจนับสัญญาณพัลซ์จากอุปกรณ์อินพุท เอาท์พุท และอุปกรณ์ภายในเมื่อมีการเปลี่ยนแปลง 1 ครั้งจะนับ 1 ครั้ง เมื่อมีการใช้COUNTER จะมีการใช้สัญญาณควบคู่กัน 2 สัญญาณ คือ สัญญาณสำหรับ RESET และสัญญาณพัลซ์ที่ต้องการนับ

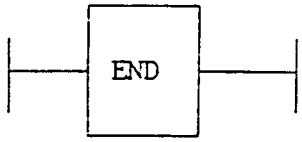
ตัวอย่าง :




```
LD   01
LD   02
CNT  01
     #0005
LD   CNT 01
OUT  500
```

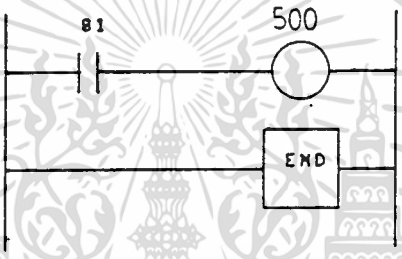
คำสั่ง : END

สัญญลักษณ์ :



ความหมาย : คำสั่งนี้จะถูกใช้เมื่อสิ้นสุดของการเขียนโปรแกรมหรืออาจกล่าวได้ว่าเป็นส่วนสุดท้ายของโปรแกรม

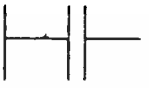
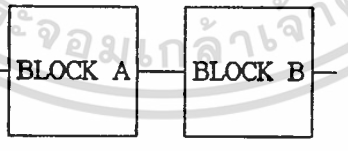
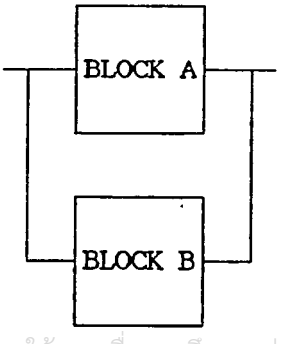
ตัวอย่าง :

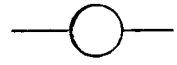
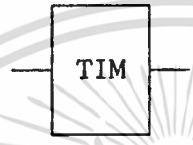
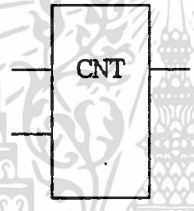
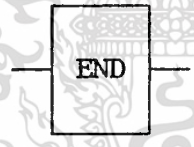



```
LD 01
OUT 500
END
```

สรุปคำสั่งพื้นฐานของ PLC SIMULATOR

ตารางที่ 1 สรุปคำสั่งพื้นฐานของ PLC SIMULATOR

INSTRUCTION	SYMBOL	NEMONIC	DATA
LOAD		LD	POINT NO.
LOAD NOT		LD-NOT	POINT NO.
AND		AND	POINT NO.
AND NOT		AND-NOT	POINT NO.
OR		OR	POINT NO.
OR NOT		OR-NOT	POINT NO.
AND LOAD		AND-LD	
OR LOAD		OR-LD	

INSTRUCTION	SYMBOL	NEMONIC	DATA
OUT		OUT	POINT NO.
OUT NOT		OUT-NOT	POINT NO.
TIM		TIM	POINT NO. SET VALUE
COUNTER		CNT	POINT NO. SET VALUE
END		END	

ทฤษฎีของสกรีนเอดิเตอร์

1.ตำแหน่งปัจจุบันสิ่งที่สำคัญเป็นเรื่องที่ต้องคำนึงมากที่สุดคือความสัมพันธ์กันระหว่างจอภาพและข้อมูลในหน่วยความจำและข้อมูลบนดิสค์ เนื่องจากโครงสร้างของข้อมูลในแต่ละแบบแตกต่างกัน โดยลักษณะข้อมูลบนจอภาพจะเป็น Array ของบรรทัดและของข้อมูลบนดิสค์เป็นสกรีนของไฟล์ซึ่งเป็นข้อมูลแบบอนุกรม ส่วนข้อมูลบนหน่วยความจำก็จะเป็นตามแบบที่โปรแกรมเมอร์เลือกใช้ข้อมูลบนจอภาพเป็นการแสดงข้อมูลเพียงส่วนหนึ่งของ File เท่านั้นขณะที่โปรแกรมจะต้องเก็บข้อมูลของ File ทั้งหมดโดยข้อมูลบางส่วนของ File อ่านเอาไว้บนหน่วยความจำขณะที่เนื้อที่ของ File ทั้งหมดยังอยู่ใน Disk

2.การแทรกและลบตัวอักษร การแทรกและลบตัวอักษรเป็นสิ่งที่ต้องคำนึงถึงในแง่ประสิทธิภาพและความเร็วในการทำงาน การแทรกตัวอักษร ในแง่ของหน่วยความจำคือการย้ายข้อมูลไป 1 ตัว อักษรจากตำแหน่งปัจจุบันแล้วนำตัวอักษรที่ต้องการแทรกไปไว้แทน ส่วนการลบตัวอักษร คือการย้ายข้อมูลในตำแหน่งถัดไป จนกระทั่งถึงท้ายข้อมูลมาข้างหน้าหากว่าข้อมูลมีขนาดใหญ่มาก การแทรก หรือการลบตัวอักษรจะทำให้โปรแกรมช้าลงมาก

3.ข้อกำหนดของการทำงาน เนื่องจากการแก้ไขข้อมูลจะต้องทำให้ Buffer ของข้อมูลตามที่โปรแกรมกำหนดไว้การพิมพ์ข้อมูลเข้าไปจนกระทั่งเกินขนาดของ Bufferของโปรแกรมย่อมทำไม่ได้

สแตค (STACK)

เป็นการนำหน่วยความจำมาทำที่เก็บข้อมูลในลักษณะ Last In-First Out หรือเรียกอีกอย่างว่า LIFO สแตคมีการใช้มากใน Software ควบคุมระบบพื้นฐานการทำงานของสแตคคือ การนำข้อมูลเข้าไปเก็บหรือเรียกว่า push และการนำข้อมูลออกมา เรียกว่า pop

ตัวอย่างการทำงาน

การกระทำ	ข้อมูลที่อยู่ในสแตค
PUSH (A)	A
PUSH (B)	B A
PUSH (C)	C B A
POP (C)	B A
PUSH (F)	F B A
POP (F)	B A
POP (B)	A
POP (A)	EMPTY

การจำลองแบบปัญหา (SIMULATION)

การจำลองแบบปัญหา เป็นวิธีการวิธีหนึ่งที่ใช้ในกระบวนการแก้ปัญหาในด้านต่างๆมาแต่โบราณ แต่ที่ได้รับความสนใจและตื่นตัวในการนำมาแก้ปัญหาในสาขาอาชีพต่างๆอย่างแพร่หลายในปัจจุบัน เป็นผลมาจากความเจริญก้าวหน้าทางเทคโนโลยีคอมพิวเตอร์ คำจำกัดความของการจำลองการแก้ปัญหาที่ Shannon ให้ไว้คือการจำลองแบบปัญหาคือ กระบวนการออกแบบแบบจำลอง (Model) ของระบบงานจริง (Real System) แล้วดำเนินการทดลองโดยใช้แบบจำลองนั้นเพื่อการเรียนรู้พฤติกรรมของระบบงานหรือเพื่อการประเมินผลการใช้นโยบายต่างๆ ในการดำเนินงานของระบบภายใต้ข้อกำหนดที่วางไว้

จากคำจำกัดความดังกล่าวจะเห็นว่า กระบวนการจำลองแบบปัญหานั้นแบ่งออกเป็น 2 ส่วน คือ การสร้างแบบจำลองส่วนหนึ่งและการนำเอาแบบจำลองส่วนนั้นไปใช้งานเชิงวิเคราะห์อีกส่วนหนึ่ง ดังจะเห็นได้ว่ากลไกของวิธีการจำลองแบบปัญหาขึ้นอยู่กับแบบจำลองและการใช้แบบจำลอง แบบจำลองที่ใช้ในการจำลองแบบปัญหานี้อาจจะเป็น หุ่น เป็นระบบ หรือเป็นแนวความคิดลักษณะหนึ่งลักษณะใดโดยไม่จำเป็นต้องเหมือนกับระบบงานจริง แต่ต้องสามารถช่วยให้เข้าใจในระบบงานจริงเพื่อประโยชน์ในการอธิบายพฤติกรรมและ เพื่อปรับปรุงการดำเนินงานของระบบงานจริง

ระบบงาน

ระบบงาน หมายถึง กลุ่มองค์ประกอบที่มีความสัมพันธ์กันโดยที่ความหมายของระบบงานบอกเฉพาะลักษณะว่าระบบงานมีลักษณะอย่างไรโดยไม่ได้บอกรูปร่างหน้าตาที่แน่ชัด ดังนั้นเมื่อจะทำการศึกษาระบบงานใดระบบหนึ่ง จึงจำเป็นที่ต้องบอกรูปร่างหน้าตาที่ชัดเจนของระบบงานที่กำลังศึกษา การบอกรูปร่างหน้าตาที่ชัดแจ้งของระบบงานมักจะบอกโดยการกำหนดขอบเขตของระบบงาน (System Boundaries) ซึ่งคือการกำหนดองค์ประกอบของระบบ การแสดงความสัมพันธ์ระหว่างองค์ประกอบ และการกำหนดองค์ประกอบอื่นๆ ที่อยู่นอกที่อยู่นอกระบบแต่มีผลกระทบต่อการทำงานของระบบ องค์ประกอบอื่นๆที่อยู่นอกระบบนี้เรียกโดยรวมว่า สิ่งแวดล้อมระบบงาน (System Enviroment) องค์ประกอบต่างๆทั้งภายในและภายนอกระบบงานมีลักษณะเฉพาะตัว (Attributes) ที่ทำให้เกิดกิจกรรม (Activities) และกิจกรรมเหล่านั้นภายใต้เงื่อนไขบางประการจะก่อให้เกิดการเปลี่ยนแปลงสถานะภาพของระบบงาน (System Status) ดังนั้นนอกจากการกำหนดขอบเขตของระบบงานแล้วยังต้องกำหนดลักษณะเฉพาะตัวขององค์ประกอบ กิจกรรมที่เกิดขึ้นจากองค์ประกอบเหล่านั้นและการเปลี่ยนแปลงของสถานะภาพของระบบงานอันเนื่องมาจากกิจกรรมขององค์ประกอบ

แบบจำลอง (MODEL)

แบบจำลอง หมายถึง ตัวแทนของวัตถุ ระบบ หรือแนวคิดลักษณะใด ลักษณะหนึ่ง แบบจำลองอาจนำไปใช้ในงานลักษณะต่างๆดังนี้

1.1 เป็นเครื่องช่วยคิด (An Aid To Thought)

1.2 เป็นเครื่องสื่อความหมาย (An Aid To Communication)

1.3 เป็นเครื่องช่วยสอนและฝึกอบรม (Purpose Of Training And Instruction) เช่น แบบจำลองเครื่องควบคุมการบิน

1.4 เป็นเครื่องมือสำหรับการทำนาย (A Tool Of Prediction)

1.5 เป็นเครื่องมือสำหรับการทดลอง (An Aid To Experimentation)

ประเภทของแบบจำลอง (CLASSIFICATION OF SIMULATION MODEL)

ประเภทของแบบจำลองสามารถที่จะจำแนกตามคุณลักษณะพิเศษดังนี้

1.แบบจำลองทางกายภาพ (Physical Models) เป็นแบบจำลองที่มีรูปร่างหน้าตาเหมือนระบบงานจริง อาจมีขนาดใหญ่กว่าหรือเล็กกว่าของจริง ตัวอย่างแบบจำลองประเภทนี้ เช่น เครื่องบินจำลองที่ใช้ทดสอบในอุโมงค์ลม

2 แบบจำลองแบบอนาลอก (Analog Model) เป็นแบบจำลองที่มีพฤติกรรมแหมือนระบบงานจริง เช่น อนาลอกคอมพิวเตอร์ ซึ่งใช้การเคลื่อนที่ของกระแสไฟฟ้าแทนการเคลื่อนที่ของวัตถุในระบบงานจริง

3.เกมส์การบริหาร (Management Games) เป็นแบบจำลองการตัดสินใจ (Decision Model) ในกิจการต่างๆ เช่น ธุรกิจ สงคราม การลงทุน

4.แบบจำลองคอมพิวเตอร์ (Computer Simulation Models) เป็นแบบจำลองที่อยู่ในรูปของคอมพิวเตอร์โปรแกรม

5.แบบจำลองทางคณิตศาสตร์ (Mathematical Models) เป็นแบบจำลองที่ใช้สัญญลักษณ์และฟังก์ชั่นทางคณิตศาสตร์แทนองค์ประกอบในระบบงานจริง

<u>กระบวนการจำลองแบบปัญหา (SIMULATION PROCESS)</u>

การจำลองแบบปัญหาในปัจจุบัน มักจะใช้กับปัญหาที่มีความยุ่งยากซับซ้อนจริงๆ จึงต้องอาศัยคอมพิวเตอร์ สำหรับช่วยในการคำนวณขั้นต้นนี้เป็นข้อเสนอแนะ สำหรับการดำเนินการจำลองแบบปัญหาที่ใช้คอมพิวเตอร์ช่วยในการคำนวณ

1.การตั้งปัญหาและการให้คำจำกัดความของระบบงาน (Problem Formulation And System Definition) ขั้นตอนนี้เป็นขั้นตอนที่สำคัญที่สุดในการจำลองแบบปัญหาขั้นตอนนี้เป็นการกำหนดวัตถุประสงค์ของการศึกษาระบบ การกำหนดขอบเขต ข้อจำกัดต่างๆ และวิธีการวัดผลของระบบงาน

2.การสร้างแบบจำลอง (Model Formulation) จากลักษณะของระบบงานที่จะต้องทำการศึกษาเขียนแบบจำลอง ที่สามารถอธิบายพฤติกรรมของระบบงานตามวัตถุประสงค์ของการศึกษา

3.การจัดเตรียมข้อมูล (Data Preartion) วิเคราะห์หาข้อมูลต่างๆที่จำเป็นสำหรับแบบจำลองและจัดเตรียมให้อยู่ในรูปแบบที่จะนำไปใช้งานกับแบบจำลองได้

4.การแปรรูปแบบจำลอง (Model Tranalation) แปลงแบบจำลองไปอยู่ในรูปแบบของโปรแกรมคอมพิวเตอร์

5.การทดสอบความถูกต้อง (Validtion) เป็นการวิเคราะห์เพื่อช่วยให้ผู้เขียนและผู้ใช้แบบจำลองมั่นใจว่าแบบจำลองที่ได้นั้น สามารภใช้แทนระบบงานจริงตามวัตถุประสงค์ของการศึกษาได้

6.การออกแบบการทดลอง (Straegic Planning)เป็นการออกแบบการทดลองที่ทำให้แบบจำลองสามารถให้ข้อมูลที่ใช้ในการวิเคราะห์หาผลลัพธ์ตามที่ต้องการ

7.การวางแผนการใช้แบบจำลอง (Tacical Planning) เป็นการวางแผนว่าจะใช้แบบจำลองในการทดลองอย่างไรจึงจะได้ข้อมูลสำหรับวิเคราะห์ผลเพียงพอ ความแตกต่างระหว่างขั้นตอนนี้ กับขั้นตอนการออกแบบการทดลองมีอยู่ว่า ในการออกแบบการทดลองเป็นแต่เพียงการบอกเงื่อนไขของการทดลองส่วนขั้นตอนนี้ เป็นการบอกว่าจะต้องดำเนินการทดลองตามเงื่อนไขดังกล่าวกี่ครั้ง จึงจะได้จำนวนข้อมูลที่เหมาะสมกล่าวคือ ได้ความมีนัยสำคัญทางสถิติที่ยอมรับได้ในราคาที่เหมาะสม

8.การดำเนินการทดลอง (Experimentation)เป็นการคำนวณหาข้อมูลต่างๆ ที่ต้องการและความไวของการเปลี่ยนแปลงข้อมูลจากแบบจำลอง

9.การตีความผลการทดลอง (Interpretation) จากผลการทดลองตีความว่าระบบงานจริงมีปัญหาอย่างไรและการแก้ปัญหาจะได้ผลอย่างไร

10.การนำไปใช้งาน (Implementation) จากผลการทดลองเลือกวิธีการที่จะแก้ปัญหาได้ดีที่สุดไปใช้กับระบบงานจริง

11.การจัดทำเป็นเอกสารการใช้งาน (Documentation) เป็นการบันทึกกิจกรรมในการจัดทำแบบจำลอง โครงสร้างของแบบจำลอง วิธีการใช้งานและผลที่ได้จากการใช้งานเพื่อประโยชน์สำหรับผู้ที่จะนำแบบจำลองไปใช้งาน และเพื่อประโยชน์ในการปรับปรุงดัดแปลงแบบจำลองเมื่อเกิดการเปลี่ยนแปลงระบบ

<u>การจำลองแบบปัญหาด้วยคอมพิวเตอร์ (COMPUTER SIMULATION)</u>

การจำลองแบบปัญหาด้วยคอมพิวเตอร์ เป็นการศึกษาปัญหาของระบบด้วยแบบจำลองซึ่งอยู่ในรูปแบบโปรแกรมคอมพิวเตอร์ โดยที่การจำลองแบบปัญหาด้วยคอมพิวเตอร์ เป็นที่นิยมใช้ที่สุดของการใช้การจำลองแบบปัญหา เพราะสามารถใช้ได้กับปัญหาของ ระบบงาน ได้มากมายหลายประเภทปัจจุบันเป็นเทคนิคการนำเอาไปใช้อย่างกว้างขวางในสหรัฐอเมริกาการจำลองแบบปัญหาเป็นวิธีการทางคณิตศาสตร์ ที่ได้รับการนำไปใช้มากที่สุด และได้นำไปใช้ในงานต่างๆ มากกว่า 70 สาขาอาชีพ และเมื่อมีผู้กล่าวถึงการจำลองแบบปัญหาทุกคนมักจะนึกถึงเข้าใจว่าเป็นการจำลองแบบ

ปัญหาด้วยคอมพิวเตอร์เสมอ หลักการที่ใช้ในการจำลองแบบปัญหาทางคอมพิวเตอร์ และเป็นหลักการแบบเดียวกับที่ใช้กับแบบจำลองปัญหาอื่นๆ ความจำเป็นที่จะสร้างแบบจำลองทางคอมพิวเตอร์หรือไม่ ขึ้นอยู่กับความยุ่งยากในการคำนวณของปัญหานั้นๆ

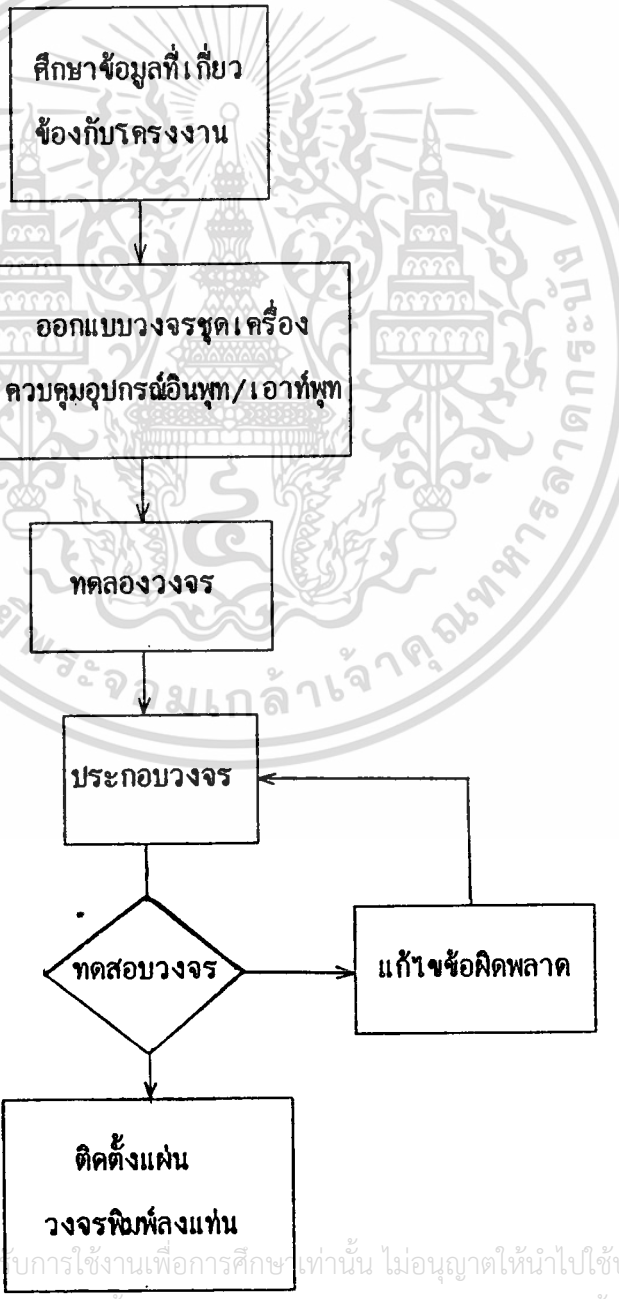
บทที่ 3

ขั้นตอนและวิธีการดำเนินงาน

โครงงาน PLC SIMULATER สามารถแบ่งขั้นตอนและวิธีดำเนินงานทออกเป็น 2 ส่วน คือ HARD WARE และ SOFT WARE

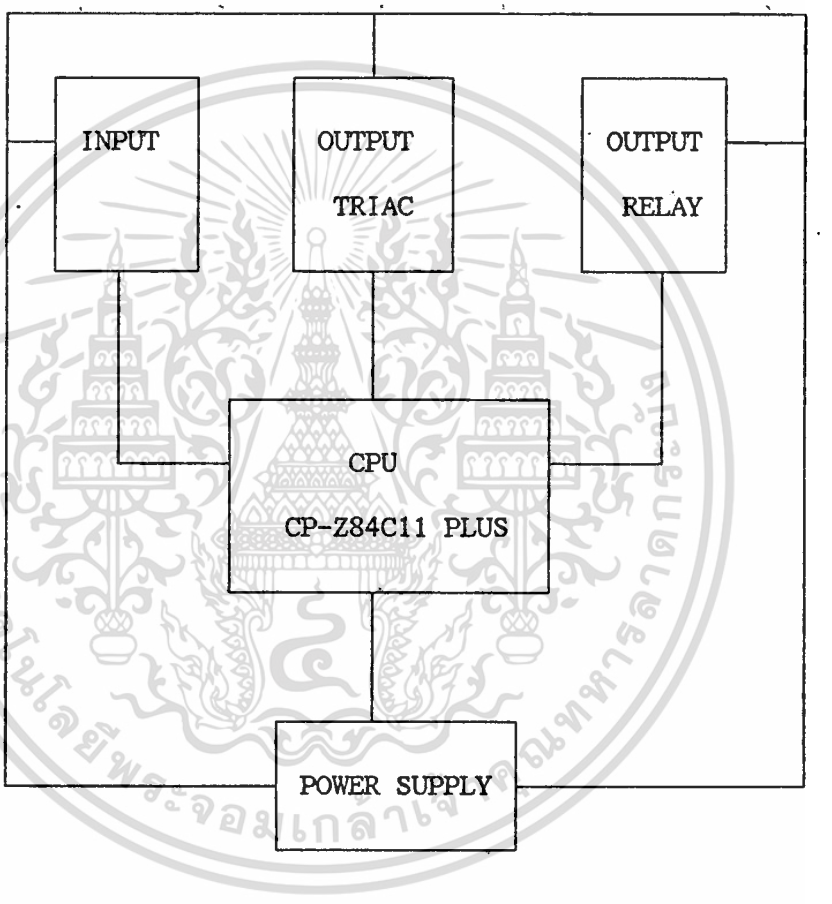
ส่วน HARD WARE

ขั้นตอนในการดำเนินงานในส่วนของ HARDWARE เราสามารถเขียนเป็นผังงานได้ดังนี้



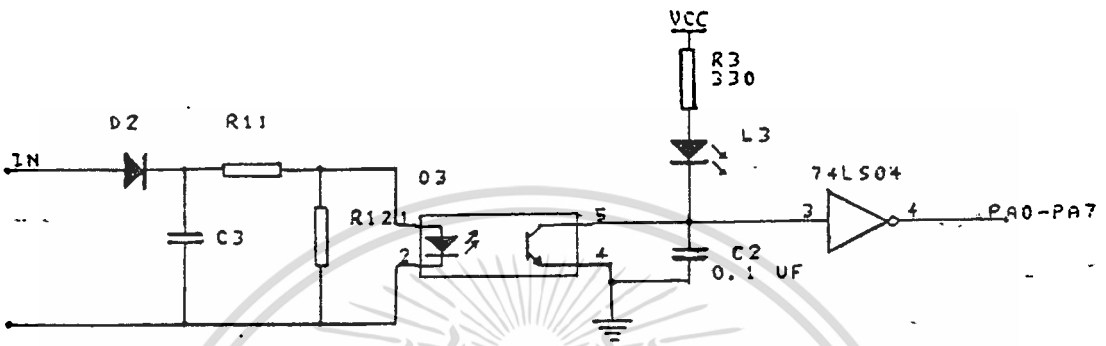
รูปที่ 3.1 ผังการดำเนินงานส่วน HARDWARE

สำหรับ HARDWARE ของเครื่องจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้สามารถเขียนเป็นบล็อคได้ดังนี้



รูปที่ 3.2 บล็อคไดอะแกรมของส่วน HARDWARE

Input circuit



รูปที่ 3.3 วงจร Input

วงจรนี้เป็นแบบ DC INPUT ชนิดแยกวงจรระหว่างส่วนภายนอก และภายในเครื่อง PLC โดยใช้ OPTO ISOLATORS เบอร์ 4N26 ปกติแล้วใช้ต่อกับอินพุทที่มีแรงดันและกระแสต่ำโดยสามารถใช้ต่อกับหน้าสัมผัสขนาดเล็กแบบต่าง ๆ หรือต่อกับ Sensor ที่เป็น Transistor Output

การทำงานของวงจร

เมื่อมีแรงดันอันเกิดจากตัว Sensor หรือ Switch ทางอินพุทที่มีค่าประมาณ 24 Volt ทางบวกก็จะสามารถผ่าน D1 ไปผ่าน C1 ก่อนจนเก็บประจุเต็มแล้วกระแสก็จะผ่านทาง R1 ไปผ่าน R2 และอินพุทของ 4N26 R1 และ R2 เป็น Voltage Divider แบ่งแรงดันจัดค่าให้เหมาะสมสำหรับ 4N26 ขณะเดียวกันทางเอาท์พุทก็จะสามารถนำกระแสได้เพราะถูกกระตุ้นด้วย LED ในตัวมันเองเมื่อTransistor นำกระแสแล้ว LED L1 จะสว่างโดยมีกระแสไหลผ่านทาง Transistor ลงกราวด์ได้และสภาวะทางอินพุทของ 74LS04 จะเป็น "0" เสมือนต่ออยู่กับกราวด์ ดังนั้น Output ของ 74LS04 จะได้สภาวะ "1" ส่งไปยัง Port A ซึ่งเป็น Input Port

<u>รายการอุปกรณ์</u>

1.DIODE 1N4001 X 8

2.CAPACITOR 0.1 UF X8

3.RESISTOR 2.2K X 8

4.RESISTOR 470 X 8

5.RESISTOR 330 X 8

6.LED X 8

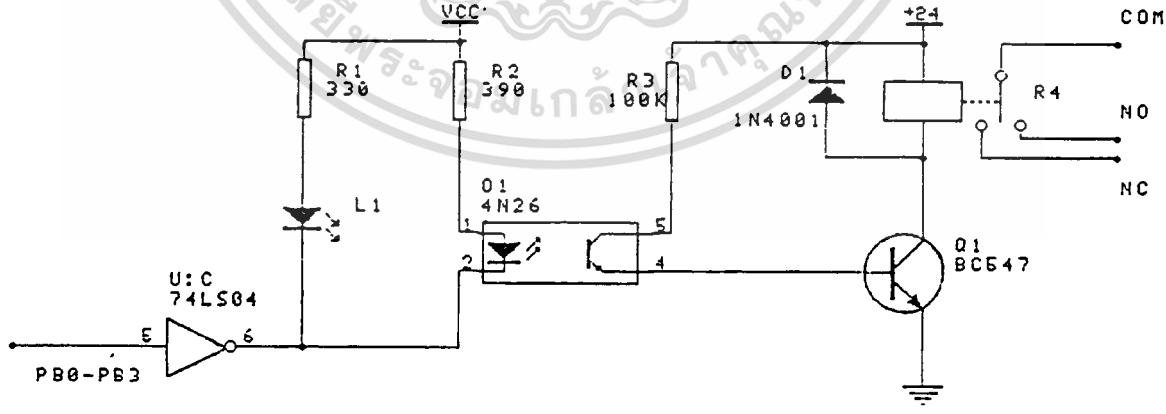
7.74LS04

8.OPTO ISOLATORS 4N26 X 8

9.TERMINAL 3 PIN X 2

10.TERMINAL 2 PIN X 2

<u>OUTPUT RELAY</u>



รูปที่ 3.4 วงจร OUTPUT RELAY

การทำงานของวงจร

เมื่ออินพุทของ 74LS04 มีสภาวะเป็น "1"ทางเอาท์พุทจะมีสภาวะเป็น "0" ดังนั้นจึงทำให้มีกระแสไหลผ่านทาง LED L1 และทาง INPUT OPTO 4N26 ดังนั้นทางขา 4,5 เสมือนต่อถึงกันเพราะ TRANSISTOR ใน PACKAGE OPTO นำกระแส ซึ่งเป็นผลทำให้มีกระแสไหลผ่าน R3 มีแรงดันไบแอสตรงให้กับ BC547 ทำให้นำกระแสจึงมีกระแสไหลผ่าน COIL RELAY ทำให้หน้า CONTACT จาก NC. ไปยัง NO. ซึ่งสามารถนำไปใช้เป็น Switch กับตัวอุปกรณ์ภายนอกได้ D1 เป็นตัวป้องกันกระแสไหลกลับขั้วเพื่อป้องกัน Coil ของ RELAY

รายการอุปกรณ์

1.DIODE 1N4001 X 4

2.RESISTOR 330 X 4

3.RESISTOR 390 X 4

4.RESISTOR 100K X 4

5.LED X 4

6.TRANSISTOR BC547 X 4

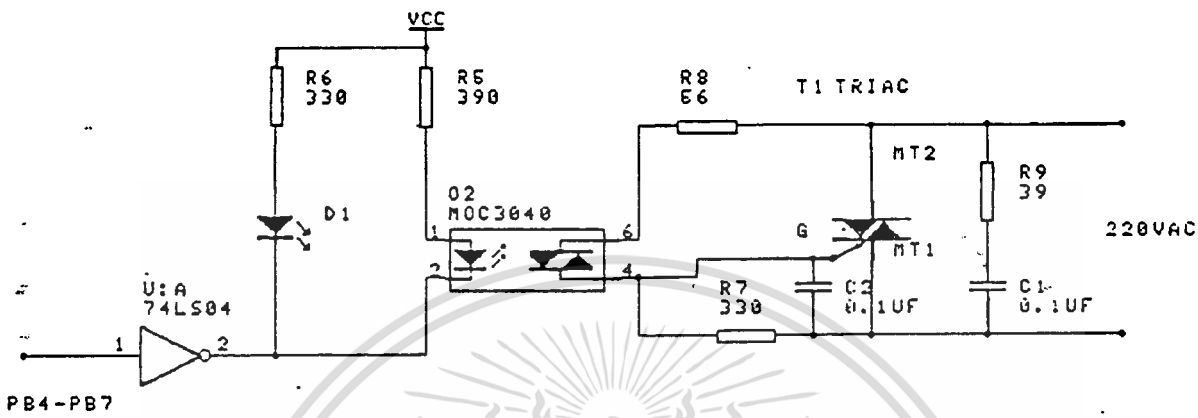
7.OPTO ISOLATORS 4N26 X 4

8.74LS04

9.RELAY 24 VDC X 1

10.TERMINAL 2PIN X 4

SOLID STATE RELAY

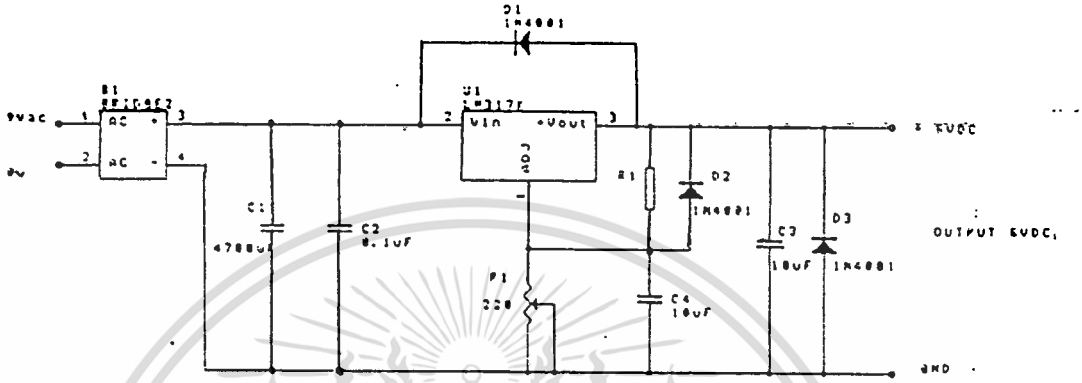


รูปที่ 3.5 วงจร SOLID STATE RELAY

วงจรนี้เป็นวงจรขับภาระที่ใช้กับแรงดัน 220 VAC ได้ไม่เกิน 450 W สำหรับการต่อใช้งานจะต้องต่อภาระกับแหล่งจ่าย 220VAC ตามรูปเส้นประก่อน วงจรนี้ก็เช่นเดียวกันกับวงจรอินพุทวงจรเอาท์พุทรีเลย์ คือ ต้องต่อแยกเป็น 2 ส่วน ระหว่างอินพุทกับเอาท์พุทของวงจร สำหรับวงจรนี้ใช้ OPTOI-SOLATOR เบอร์ MOC 3040 ซึ่งเป็นเอาท์พุทไตรแอค เหมาะสำหรับใช้งานกับแรงดัน 220VAC ซึ่งมีข้อดีคือ มีตัวตรวจจับแรงดันศูนย์ (ZERO CROSSING) อยู่ในตัว OPTO หมายความว่า โฟโตไตรแอคภายใน M1 นี้จะเริ่มนำกระแสเมื่อแรงดันรูปคลื่นซายน์ในสาย มีค่าเริ่มจากศูนย์ไปทางซีกบวกและซีกลบ

<u>การทำงานของวงจรนี้</u> เรามาพิจารณากันที่ 74LS04 ของวงจรทางขา 1 ต่ออยู่กับ PORT B เมื่อ PORT B ได้ส่งสภาวะ "1" ออกมาทำให้ที่ขา 2 ของ 74LS04 มีสภาวะเป็น "0" จากโครงสร้างทางเอาท์พุทของ 74LS04 ในสภาวะ "0" จะรับกระแสจากภายนอกไหลเข้าสู่ตัวมันได้ประมาณ 400 mA ดังนั้นจากวงจรนี้ก็จะทำให้มีกระแสไหลผ่าน L1 ซึ่งเป็น LED ทำให้เกิดสว่างขึ้นแสดงว่าตอนนี้ ในภาคเอาท์พุทไตรแอคนี้ทำงานอยู่และอีกส่วนหนึ่ง มีกระแสไหลผ่านขา 1 ของ M1 มาทางขา 2 ทำให้ LED ในตัว M1 ทำงานส่งแสงไปสั่งให้ทางโฟโตไตรแอค ภายใน M1 เองทำงาน R2 เป็นตัวต้านทานทำหน้าที่ จำกัดกระแสที่ไหลผ่านอินพุทของ M1 ในโครงงานนี้

แหล่งจ่ายแรงดัน 5 Volt 3 Amp



รูปที่ 3.6 แสดงวงจร POWER SUPPLY 5V

วงจรนี้ทำหน้าที่เป็นแหล่งจ่ายไฟ 5 VDC ให้กับชุด Control Board,Input/Output

การทำงานของวงจร

เริ่มจากมีแรงดันไฟสลับ 9 VAC ป้อนให้ BRIDGE RECTIFIER แล้วจะได้แรงดันเต็มคลื่นไปทำให้เรียบขึ้นโดย C1 ส่วน C2 ทำหน้าที่ Bypass สัญญาณรบกวนที่รบกวนซึ่งมากับ Input LM 317เป็นไอซี Regulator 3 ขา เราต่อเพื่อรักษาระดับแรงดันคงที่ 5 V C3 ป้องกันผลของ Transient State.

รายการอุปกรณ์

1. BRIDGE RECTIFIER 3 A	1
2. LM 317	1
3. VR 470	1
4. C 4700 uF	1

จะให้กระแสประมาณ 12 mA (ความต้านทาน 390 แรงดัน 5 V) เมื่อรีเลย์ไดรแอค ทำงานก็จะทำให้ มีกระแสไหลผ่านตัวมันไป ตกคร่อม C2 ซึ่งค่าแรงดันนี้จะได้จาก R3 และ R4 ซึ่งเป็นตัวแบ่งแรงดัน เมื่อแรงดันที่ C2 สูงพอที่จะทริกไดรแอคให้ทำงาน นำกระแส ก็จะทำให้ภาระที่ต่ออยู่ครบวงจรทางไฟฟ้า สามารถทำงานได้

สำหรับ R5 และ C1 เป็นตัวป้องกันทรานเซี้ยนต์อันเกิดจากการเพิ่มแรงดันของโหลดอย่างรวดเร็ว ซึ่งจะทำให้ไดรแอคเสียหายได้ นอกจากนี้ C1 ยังเป็นตัวป้องกันสัญญาณรบกวน และแรงดันกระชากในสายไฟด้วยส่วน R4 เป็นตัวต้านทานจำกัด กระแสที่ไหลผ่าน C1

<u>รายการอุปกรณ์</u>

1.RESISTOR 39 X 4

2.RESISTOR 56 X 4

3.RESISTOR 330 X 8

4.RESISTOR 390 X 4

5.CAPACITOR 0.1UF 16V X 4

6.CAPACITOR 0.1UF 250VAC X 4

7.LED X 4

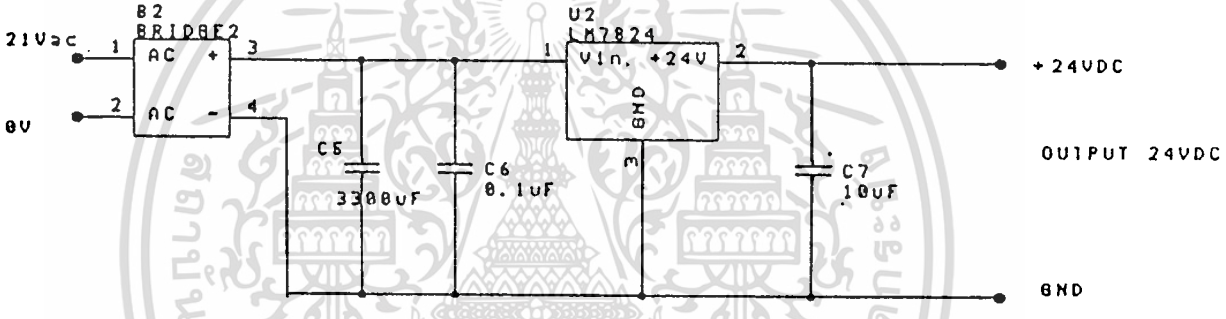
8.OPTO ISOLATORS MOC3040 X 4

9.74LS04

10.TERMINAL 2PIN X 4

5. C 10 uF	2
6. C 0.1 uF	1
7. DIODE 1N4001	3
8. R 150	1

แหล่งจ่ายแรงดันไฟตรง 24 VDC 1 A



รูปที่ 3.7 แสดงวงจร POWER SUPPLY 24 V.

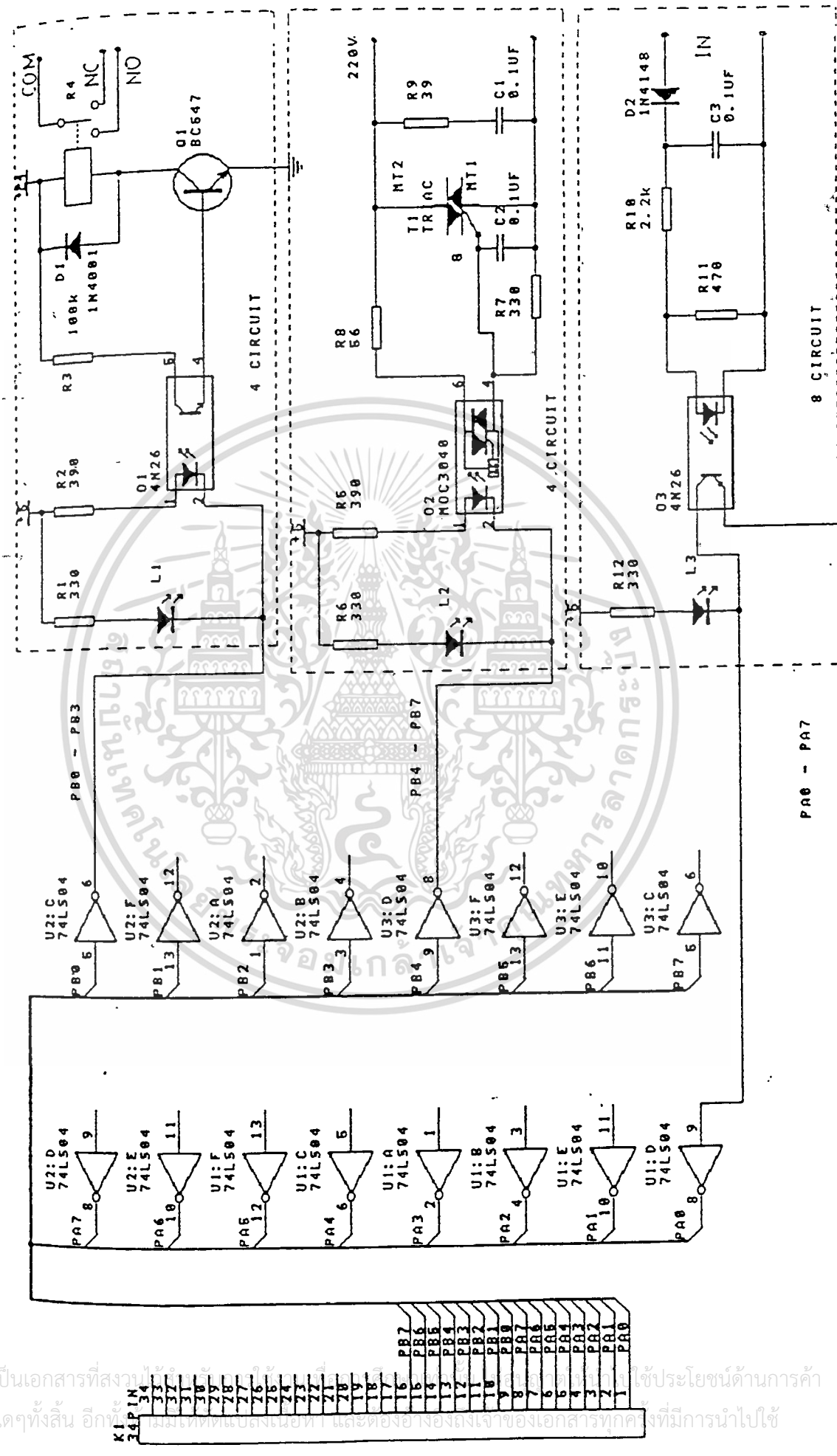
วงจรนี้ทำหน้าที่เป็นแหล่งจ่ายไฟ 24 VDC ให้กับชุด Input ส่วนการขับรีเลย์ให้ทำงาน

การทำงานของวงจร

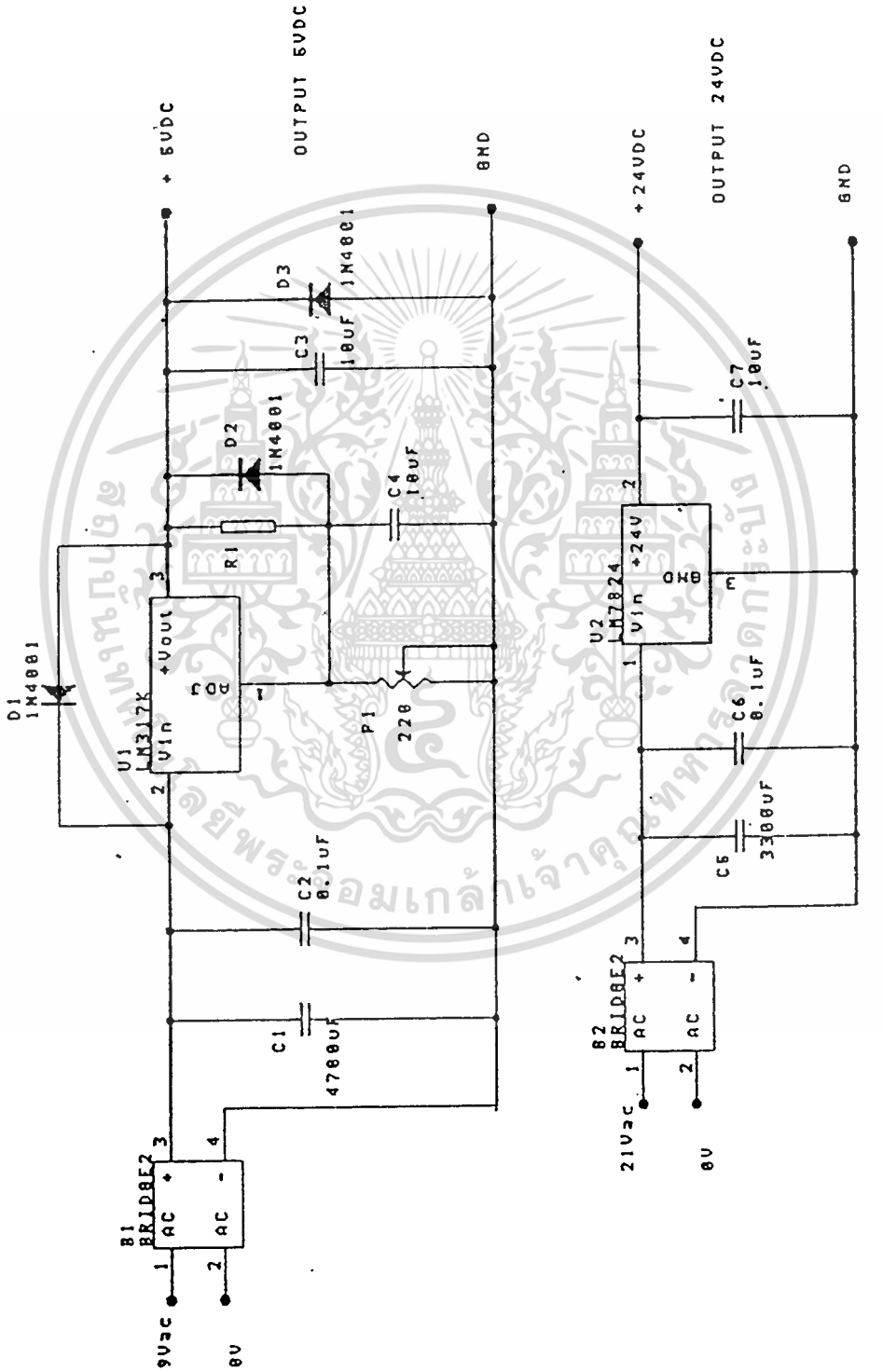
เริ่มจากมีแรงดันไฟกระแสสลับ 21 VAC ป้อนให้ BRIDGE RECTIFIER แล้วจะได้แรงดันเต็มคลื่น (Full Wave) ไปทำให้เรียบขึ้นโดย C1 ส่วน C2 ทำหน้าที่ Bypass สัญญาณรบกวนที่มากับ Input LM 7824K เป็นไอซี Regulator 3 ขา ต่อเพื่อรักษาระดับแรงดัน 24 V C3 ป้องกันผลของ Transient State.

รายการอุปกรณ์

1.BRIDGE RECTIFIER 1 A	1
2.LM 7824K	1
3.C 3300 uF	1
4.C 10 uF	1
5.C 0.1 uF	1



รูปที่ 3.8 วงจรรวมอินพุท/เอาท์พุท

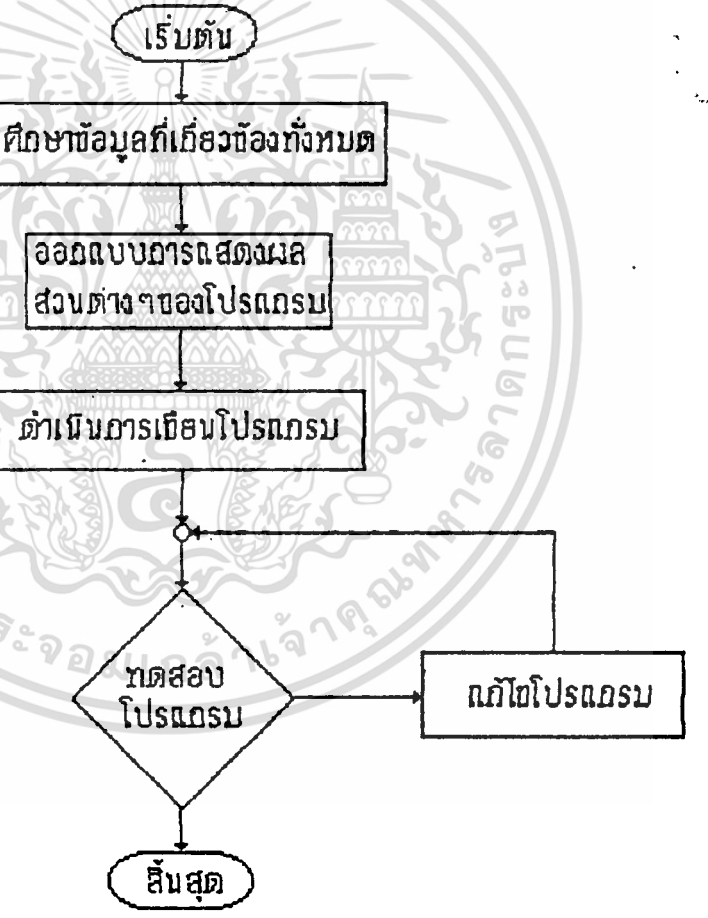


รูปที่ 3.9 วงจรรวมเพาเวอร์ซัพพลาย

ส่วน Software

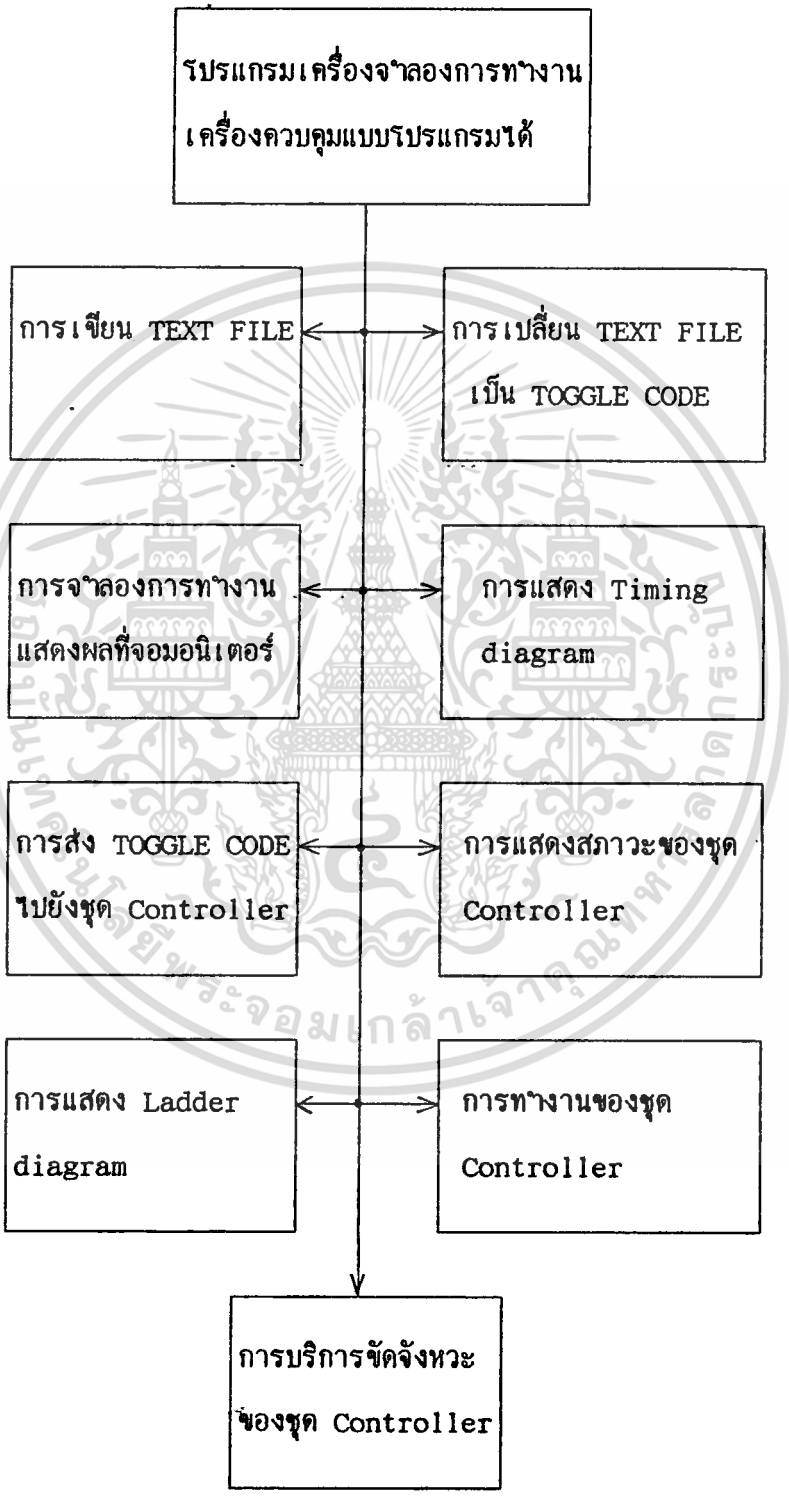
ขั้นตอนการดำเนินงาน

ขั้นตอนการดำเนินงาน เป็นการแสดงลำดับขั้นตอนการเขียนโปรแกรมจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้ดังรูป



รูปที่ 3.10 แสดง Flow Chart การดำเนินงานส่วน Software

ส่วนประกอบของโปรแกรมในส่วน Software



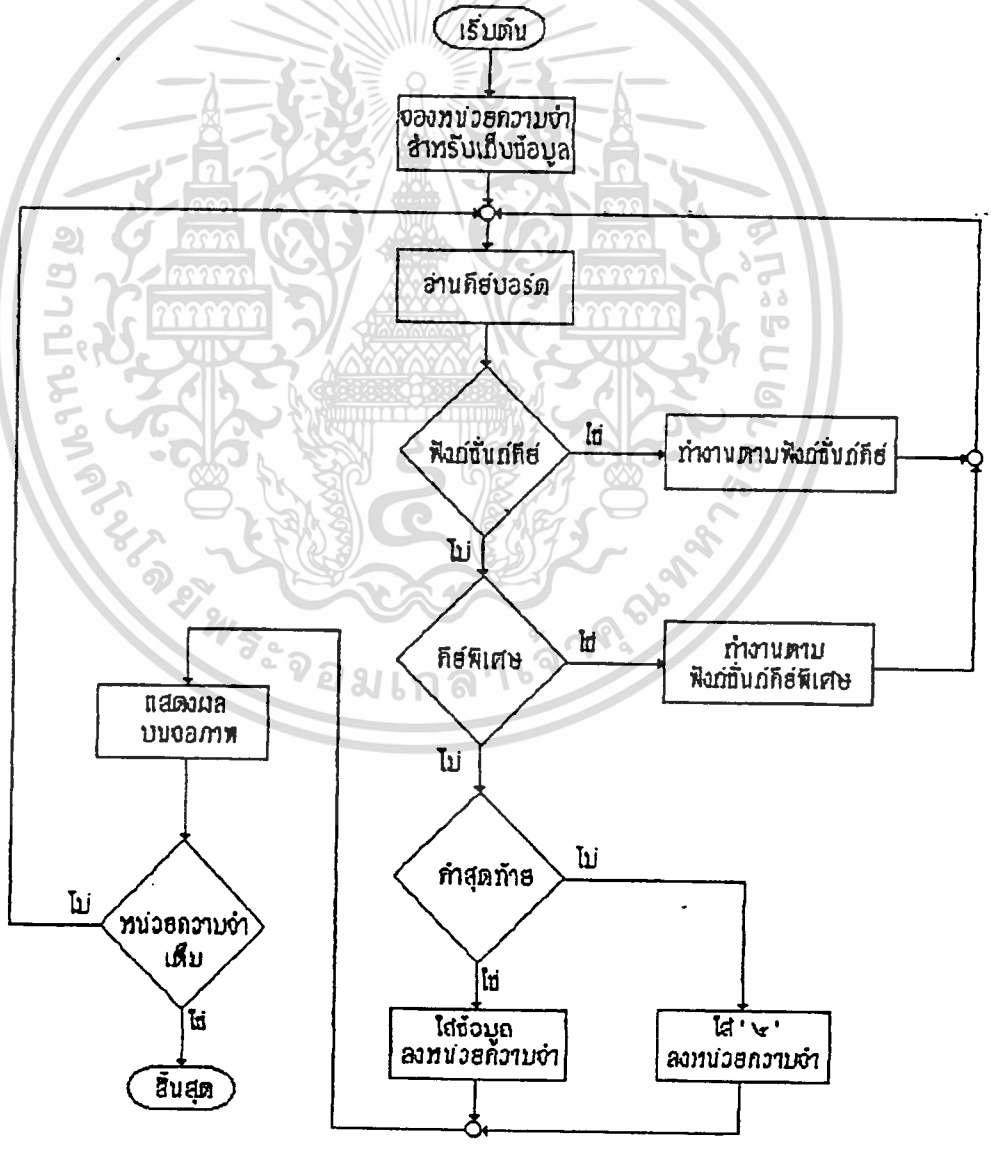
รูปที่ 3.11 แสดง Block Diagram ส่วนของ Software

การทำงาน

โปรแกรมจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้สามารถที่จะแบ่งส่วนต่างๆดังนี้คือ

1.ส่วนการสร้าง Text File

การทำงานของโปรแกรมเครื่องควบคุมโปรแกรมได้ต้องมีส่วนของการสร้าง Text File ที่ผู้ใช้เป็นผู้เขียนคำสั่งของ PLC ที่เป็นภาษาบูลลีนการทำงานของโปรแกรมส่วนการสร้าง Text File แสดงดังรูป



รูปที่ 3.12 Flow Chart ส่วนของการสร้าง Text File

การสร้าง Text File มีขั้นตอนการทำงานดังนี้

1.1 การจองหน่วยความจำไว้สำหรับเก็บข้อมูลที่เป็น Text File และกำหนดตัวชี้ตำแหน่งของข้อมูลที่จองไว้ในหน่วยความจำ

1.2 การอ่านข้อมูลจาก key board

1.3 ตรวจสอบว่าเป็นฟังก์ชั่นคีย์หรือไม่ ในกรณีที่ใช่จะทำตามรูปรแกรมของฟังก์ชั่นคีย์ที่กำหนดไว้

1.4 ตรวจสอบว่าเป็นฟังก์ชั่นคีย์พิเศษหรือไม่ ในกรณีใช่จะทำตามรูปรแกรมของฟังก์ชั่นคีย์พิเศษนั้น

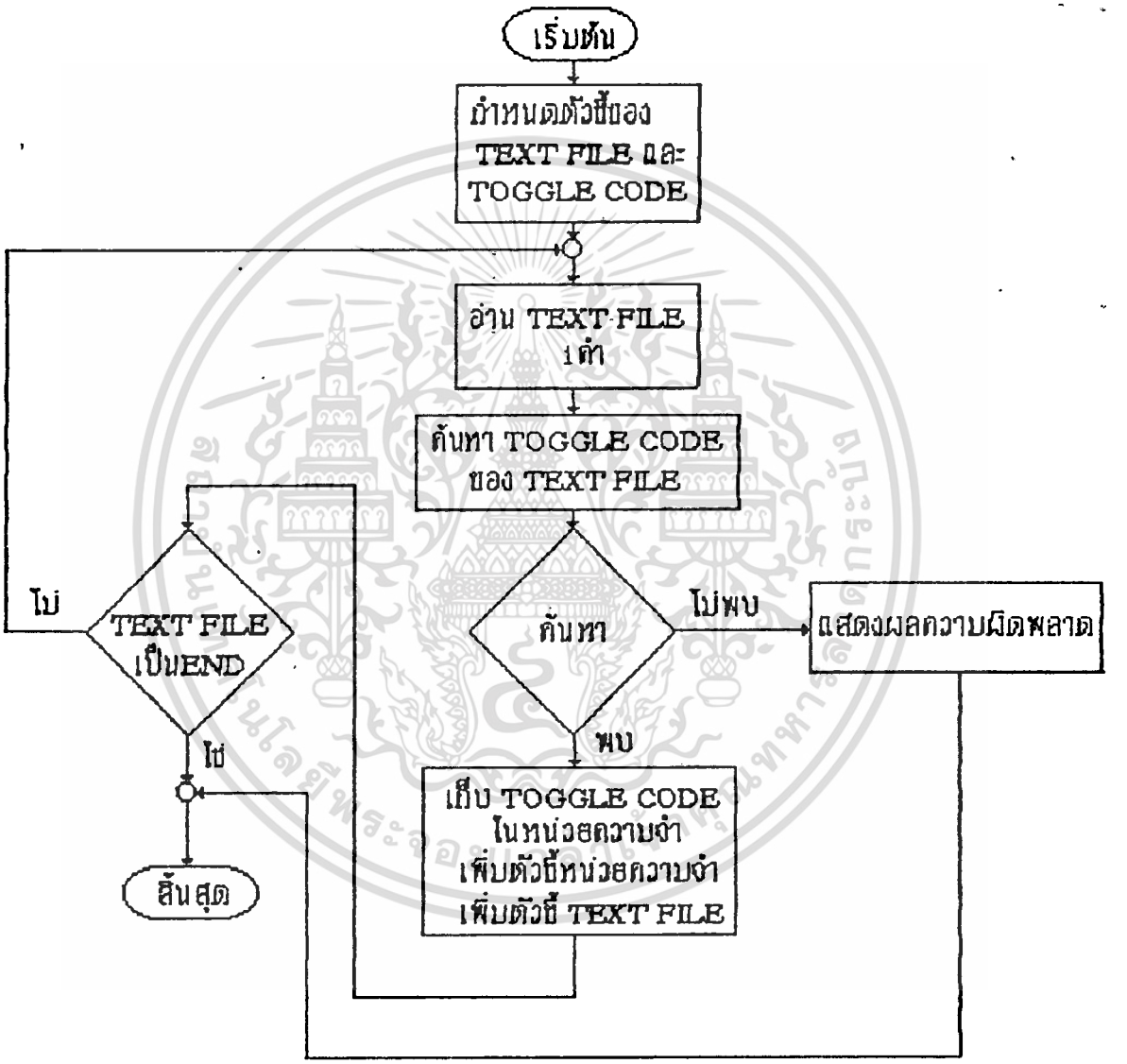
1.5 ในกรณีที่เป็นการอ่านค่าของตัวอักษรเข้ามาต้องทดสอบว่าครบ 1 บรรทัดหรือไม่ ในกรณีที่ใช่ใส่รหัสขึ้นบรรทัดใหม่ลงในหน่วยความจำ ถ้าไม่ใช่นำตัวอักษรที่อ่านได้ไปเก็บไว้ในหน่วยความจำ

1.6 แสดงผลของข้อมูลที่อยู่ในหน่วยความจำ

1.7 ตรวจหน่วยความจำเต็ม หรือยังในกรณีที่เต็มจะออกจากการเขียนรูปรแกรม ในกรณีที่หน่วยความจำไม่เต็มจะกลับไปอ่านข้อมูลใหม่

2. การเปลี่ยนรหัสที่เป็น Text File เป็นรหัส Toggle Code

การเปลี่ยนข้อมูลจากรหัสคำสั่งที่เป็น Text File เป็นรหัส Toggle Code มีขั้นตอนการทำงานดังรูป



รูปที่ 3.13 Flow chartของการเปลี่ยน Text File เป็น Toggle Code

<u>การเปลี่ยนข้อมูลจากรหัสคำสั่งที่เป็น Text File เป็นรหัส Toggle Code แบ่งขั้นตอนการทำงานเป็นดังนี้</u>

2.1 จองหน่วยความจำที่จะทำการเก็บรหัส Toggle Code และ กำหนดตัวชี้สำหรับ Text File และ Toggle Code

2.2 อ่านค่าตัวอักษรของ Text File จำนวน 1 คำ

2.3 นำค่าที่มาจาก Text File มาเปรียบเทียบกับข้อมูลของโปรแกรมเพื่อให้ได้ Toggle Code

2.4 ในกรณีที่ไม่สามารถเปลี่ยน Text File เป็น Toggle Code จะแสดงข้อผิดพลาดแล้วออกจากโปรแกรม

2.5 ในกรณีที่สามารถเปลี่ยน Text File เป็น Toggle Code ได้ทำการเก็บ Toggle code ไว้ในหน่วยความจำ

2.6 ตรวจสอบคำของ Text File ในกรณีที่เป็น" END "จะออกจากโปรแกรมแสดงว่าการเปลี่ยน Text File เป็น Toggle Code เสร็จสมบูรณ์

2.7 ในกรณีที่คำของ Text File ไม่ใช่ " END " ทำการเลื่อนตัวชี้ของ Text File และ Toggle Code ไปยังตำแหน่งถัดไปแล้วไปเริ่มกับข้อ 2.2 ใหม่

คำสั่งของโปรแกรม PLC ที่เปลี่ยนเป็น Toggle Code แสดงดังตารางที่ 2

คำสั่ง PLC	TOGGLE CODE	
	เลขฐานสิบ	เลขฐานสิบหก
END	00	00
LD XX	01	01
LD YYY	02	02
LD TR ZZ	03	03
LD CNT AA	04	04
LD TIM BB	05	05
LD NOT XX	06	06
LD NOT YYY	07	07
LD NOT CNT AA	08	08
LD NOT TIM BB	09	09
AND XX	10	10
AND YYY	11	11
AND CNT AA	12	12
AND TIM BB	13	13
AND NOT XX	14	14
AND NOT YYY	15	15
AND-NOT CNT AA	16	16
AND-NOT TIM BB	17	17
OR XX	18	18
OR YYY	19	19
OR CNT AA	20	20
OR TIM BB	21	21

คำสั่ง PLC	TOGGLE CODE	
	เลขฐานสิบ	เลขฐานสิบหก
OR NOT XX	22	22
OR NOT YYY	23	23
OR NOT CNT AA	24	24
OR NOT TIM BB	25	25
AND LD	26	26
OR LD	27	27
CNT AA #CCCC	28	28
TIM BB #DDDD	29	29
OUT YYY	30	30
OUT TR ZZ	31	31
OUT NOT YYY	32	32

เมื่อ

XX แทนหมายเลข SW

YYY แทนหมายเลข OUTPUT

ZZ แทนหมายเลข Tempolary Relay

AA แทนหมายเลข ตัวนับ

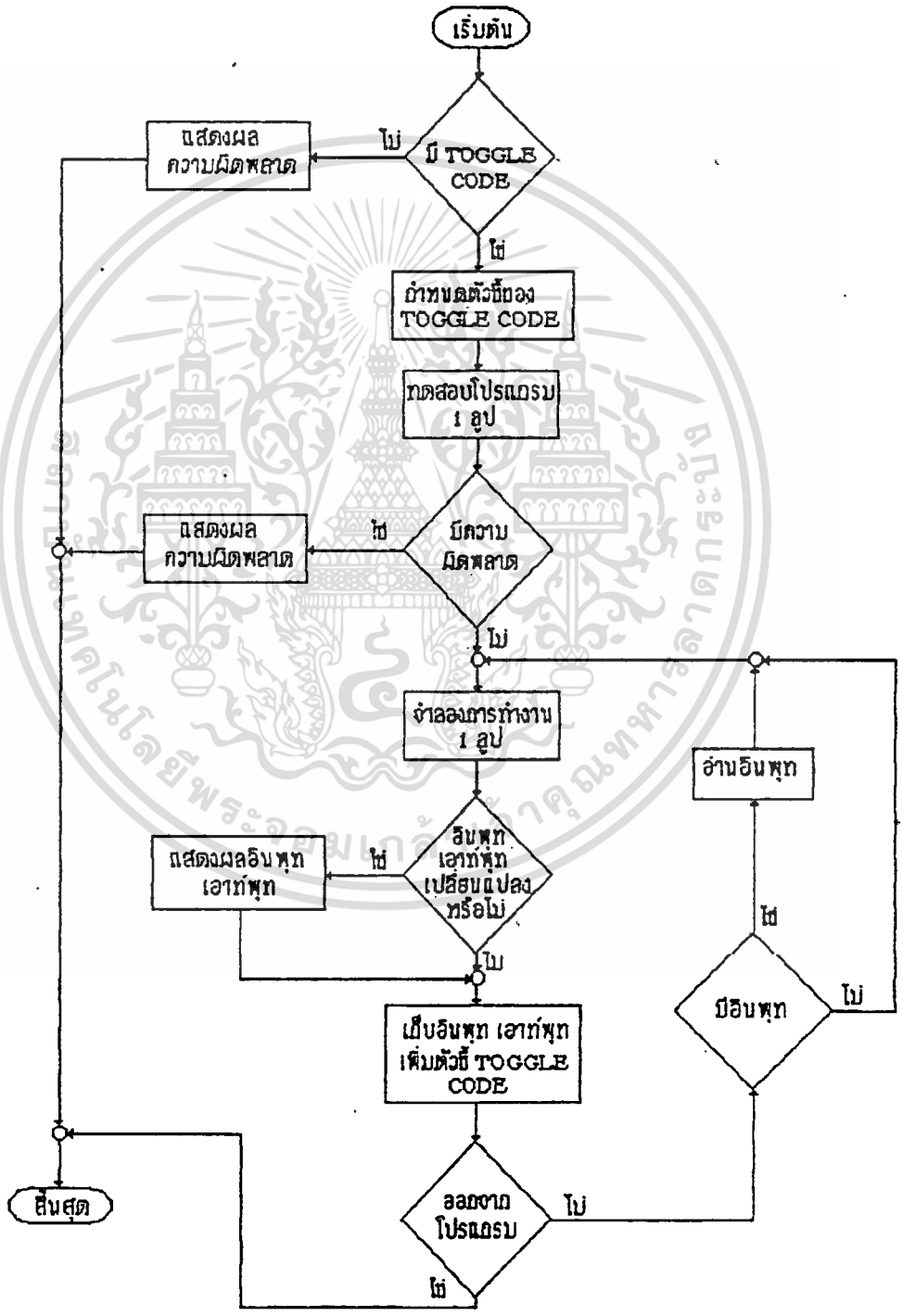
BB แทนหมายเลข ตัวตั้งเวลา

CCCC แทนจำนวนที่ต้องการให้นับ

DDDD แทนจำนวนเวลาที่ต้องการตั้ง

3. การจำลองการทำงานและการแสดงผลที่จอมอนิเตอร์

การจำลองการทำงานจาก TOGGLE CODE และแสดงผลที่จอมอนิเตอร์มีขั้นตอนการทำงานดังรูป



รูปที่ 3.14 FLOW CHART ของการจำลองการทำงานของ TOGGLE CODE

การแสดง TOGGLE CODE แบ่งขั้นตอนการทำงานดังนี้

3.1 ตรวจสอบมี TOGGLE CODE อยู่ในหน่วยความจำหรือไม่

3.2 ในกรณีที่มี TOGGLE CODE จะกำหนดตัวชี้ TOGGLE CODE ขึ้นแล้วทดสอบโปรแกรม PLC 1 Loop

3.3 ตรวจสอบข้อผิดพลาดในกรณีที่มีข้อผิดพลาดไปที่ข้อ 3.10

3.4 จำลองการทำงานจาก TOGGLE CODE จำนวน 1 Loop

3.5 อินพุทและเอาท์พุทเปลี่ยนแปลงหรือไม่

3.6 ในกรณีที่เปลี่ยนแปลง แสดงสภาวะเอาท์พุทและอินพุทที่เปลี่ยนแปลง

3.7 เก็บค่าของอินพุทและเอาท์พุท ที่เปลี่ยนแปลงและเลื่อนตำแหน่งตัวชี้ของ TOGGLE CODE

3.8 ต้องการออกจากโปรแกรมหรือไม่ในกรณีที่ต้องการออกจากโปรแกรมไปยังข้อ 3.11

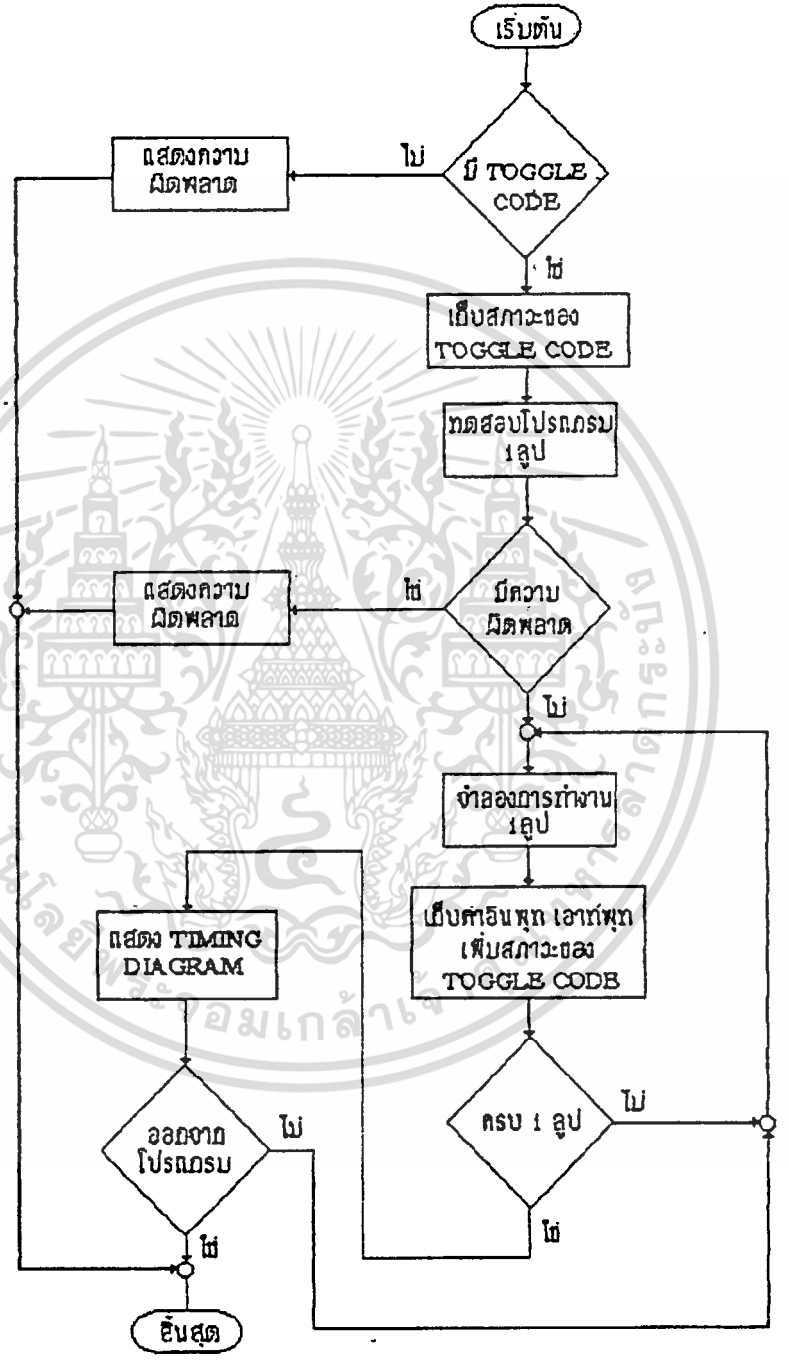
3.9 มีอินพุทเข้ามาใหม่หรือไม่ ในกรณีที่มีอินพุท อ่านค่าของอินพุทแล้วไปยังข้อ 3.4 ในกรณีที่ไม่มีอินพุทไปทำงานที่ข้อ 3.4

3.10 แสดงข้อความที่ผิดพลาด

3.11 ออกจากโปรแกรม

4.การแสดง TIMING DIAGRAM

การแสดง Timing Diagram เป็นการที่นำข้อมูลที่เก็บไว้จากการจำลองการทำงานมาแสดงดังในรูป



รูปที่ 3.15 FLOW CHART ของโปรแกรมการแสดง TIMING DIAGRAM

การแสดง TIMING DIAGRAM สามารถแบ่งขั้นตอนการทำงานได้ดังนี้

4.1 ตรวจสอบมี TOGGLE CODE อยู่ในหน่วยความจำหรือไม่

4.2 ในกรณีที่มีTOGGLE CODE จะกำหนดตัวชี้ TOGGLE CODE ขึ้นแล้วทดสอบโปรแกรม PLC 1 Loop

4.3 ตรวจสอบข้อผิดพลาดในกรณีที่มีข้อผิดพลาดไปที่ข้อ 4.9

4.4 จำลองการทำงานจาก TOGGLE CODE 1 Loop

4.5 เก็บค่าของอินพุทและเอาท์พุท ที่เปลี่ยนแปลงและเลื่อนตำแหน่งตัวชี้ของ TOGGLE CODE

4.6 จำลองการทำงานครบ 20 Loop หรือไม่ ในกรณีที่ครบไปยังข้อ 4.7 ในกรณีที่ยังไม่ครบไปยังข้อ 4.4

4.7 แสดง Timing Diagram

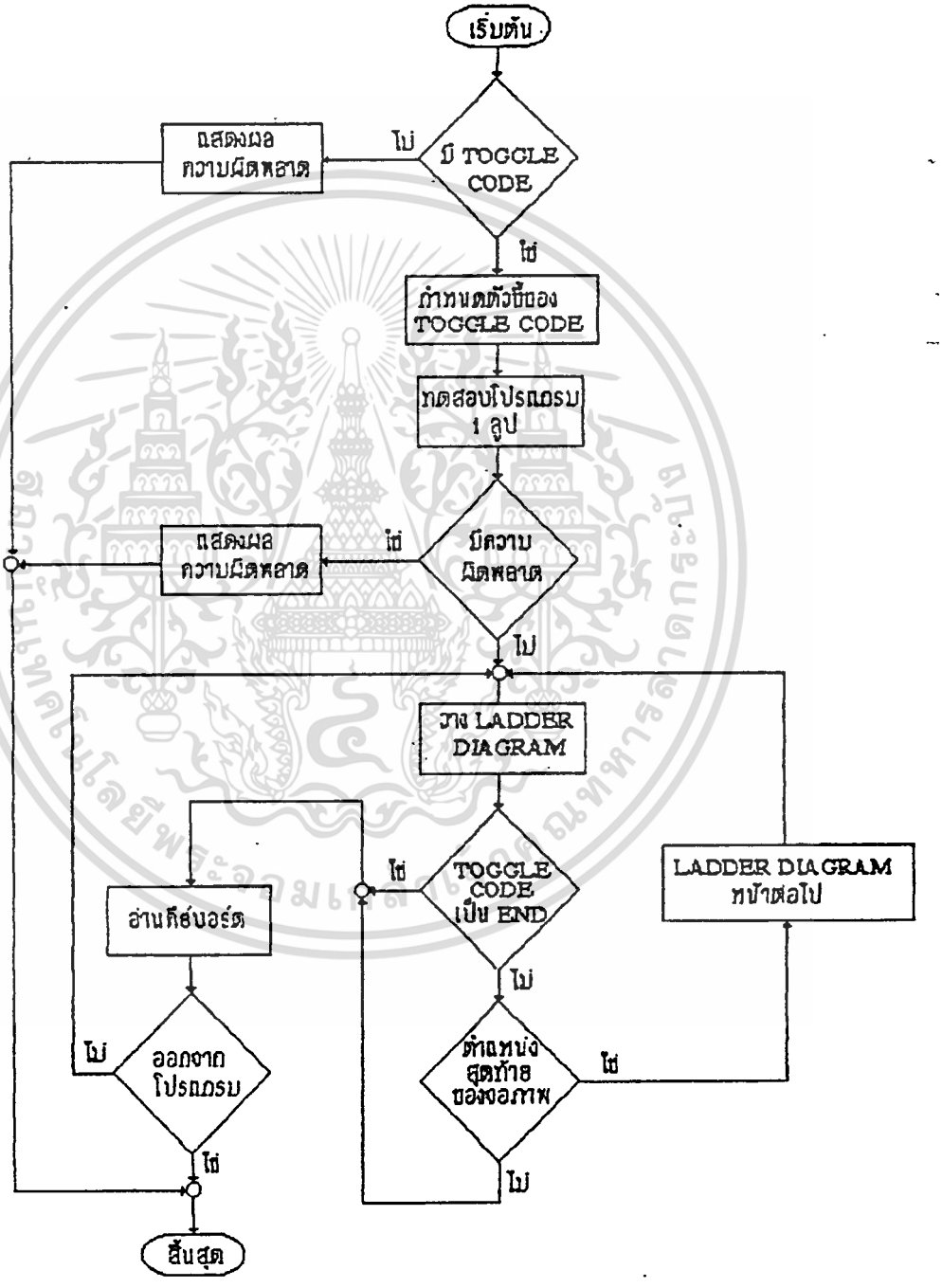
4.8 ต้องการออกจากโปรแกรมหรือไม่

4.9 แสดงข้อความที่ผิดพลาด.

4.10 ออกจากโปรแกรม

5.การแสดง LADDER DIAGRAM

การแสดง Ladder Diagram เป็นการนำค่าของ TOGGLE CODE มาหา Ladder diagram โดยการแสดง Ladder Diagram สามารถแสดงได้ตามขนาดของจอภาพ ขั้นตอนการแสดง Ladder Diagram แสดงได้ดังรูป



รูปที่ 3.16 FLOW CHART ของการแสดง LADDER DIAGRAM

การแสดง LADDER DIAGRAM เป็นการนำค่าของ TOGGLE CODE มาหา LADDER DIAGRAM มีขั้นตอนการทำงานดังนี้

5.1 ตรวจสอบมี TOGGLE CODE ในหน่วยความจำหรือไม่

5.2 ในกรณีที่มี TOGGLE CODEจะกำหนดตัวชี้ TOGGLE CODE ขึ้น แล้วทดสอบ PLC 1 Loop

5.3 ตรวจสอบข้อผิดพลาดในกรณีที่มีข้อผิดพลาดไปที่ข้อ 5.10

5.4 นำสัญลักษณ์ของ Ladder Diagram ตาม TOGGLE CODE มาแสดงที่หน้าจอภาพ

5.5 ค่า TOGGLE CODE คือ END หรือไม่ในกรณีที่ใช่ ไปยังข้อ 5.8 นกรณีที่ไม่ใช่ไปยังข้อ 5.6

5.6 การวาง Ladder Diagram อยู่ที่ตำแหน่งสุดท้ายของจอภาพหรือไม่ ในกรณีที่อยู่ตำแหน่งสุดท้ายของจอภาพ ไปยังข้อ 5.7 ในกรณีที่ไม่ใช่ตำแหน่งสุดท้ายของจอภาพไปยังข้อ 5.8

5.7 เลื่อนการแสดง Ladder Diagram ไปหน้าใหม่แล้วไปยังข้อ 5.4

5.8 อ่านค่าของคีย์บอร์ด

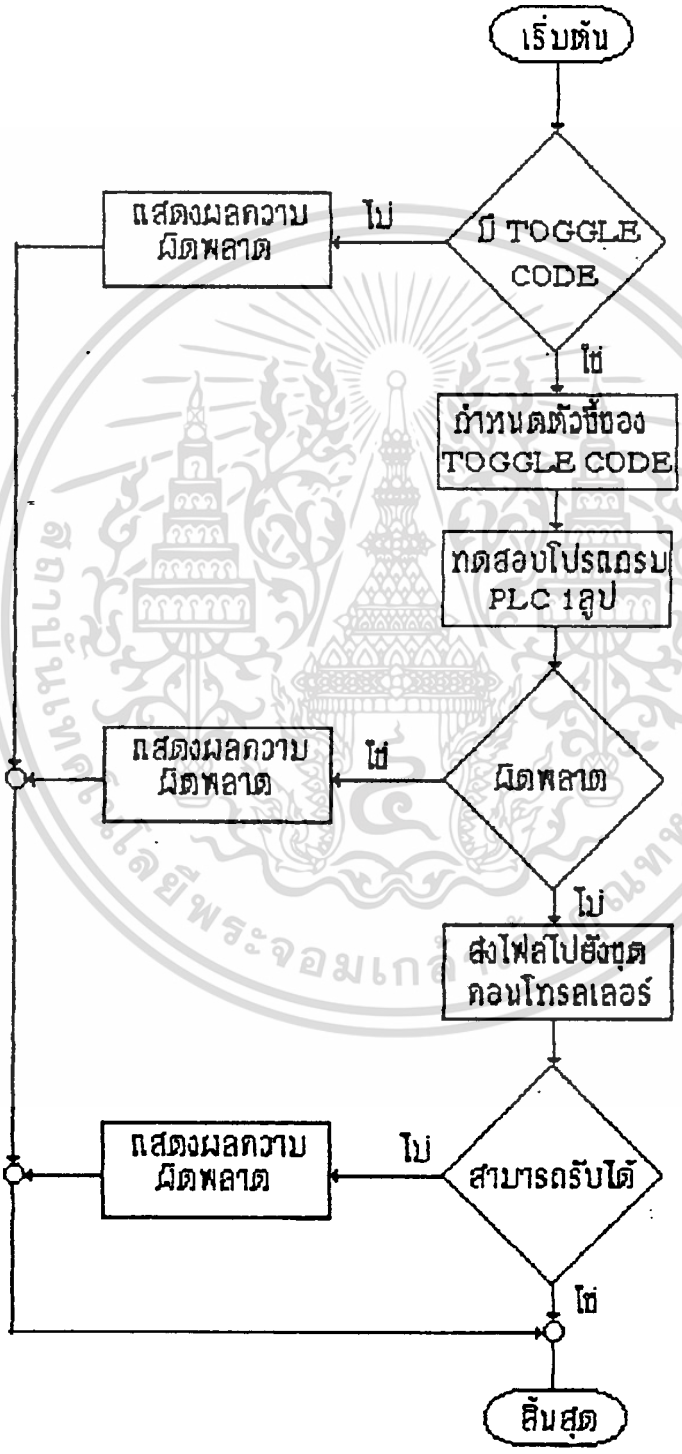
5.9 ต้อการออกจากโปรแกรมหรือไม่ในกรณีที่ต้องการออกจากโปรแกรมไปยังข้อ 5.11

5.10 แสดงข้อความที่ผิดพลาด

5.11 ออกจากโปรแกรม

6.การส่ง TOGGLE CODE ไปยังชุด CONTROLLER

การส่ง TOGGLE CODE ไปยังชุด Controller เป็นการส่งข้อมูลแบบอนุกรมโดยใช้มาตราฐาน RS-232 เมื่อส่งข้อมูลไปแล้วชุด CONTROLLER ต้องส่งสัญญาณตอบสนองกลับมาจึงจะไม่เกิดข้อผิดพลาด ขั้นตอนการส่งแสดงได้ดังรูป



รูปที่ 3.17 FLOW CHART ของโปรแกรมการส่ง TOGGLE CODE ไปยังชุด CONTROLLER

การส่ง TOGGLE CODE ไปยังชุด CONTROLLER มีขั้นตอนการทำงานดังนี้

6.1 ตรวจสอบมี TOGGLE CODE อยู่ในหน่วยความจำหรือไม่

6.2 ในกรณีที่มี TOGGLE CODE จะกำหนดตัวชี้ TOGGLE CODE ขึ้นแล้วทดสอบโปรแกรม PLC 1 Loop

6.3 ตรวจสอบข้อผิดพลาดในกรณีที่มีข้อผิดพลาดไปที่ข้อ 6.6 ในกรณีที่ไม่มีข้อผิดพลาดไปที่ข้อ 6.4

6.4 ส่ง Toggle Code ไปยังชุด Controllerโดยใช้การส่งข้อมูล แบบอนุกรมตามมาตราฐาน RS-232 ความเร็วในการส่ง 9600 b/s

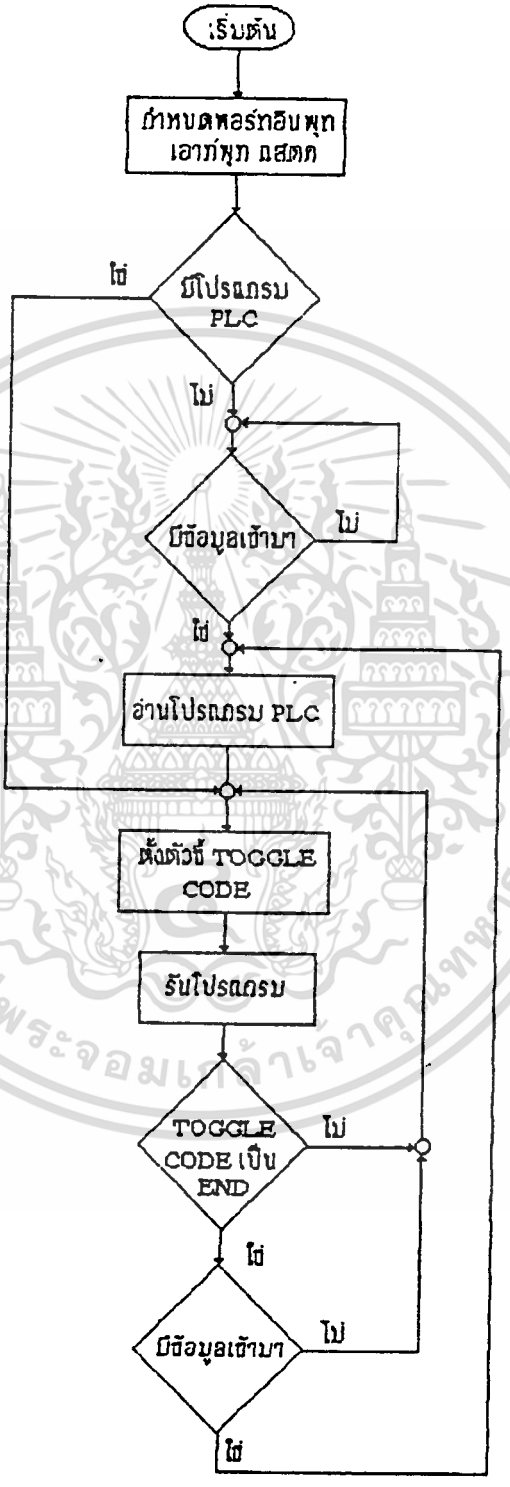
6.5 ตรวจสอบชุด Controller รับได้หรือไม่ ในกรณีที่รับได้ไปยังข้อ 6.7 ในกรณีที่รับไม่ได้ไปยังข้อ 6.6

6.6 แสดงข้อความผิดพลาดและออกจากโปรแกรม

6.7 ออกจากโปรแกรม

7. การทำงานของโปรแกรมหลักของชุด CONTROLLER

การทำงานของโปรแกรมในชุด Controller เป็นการนำ Toggle Code ที่ได้รับจากเครื่องคอมพิวเตอร์มาทำงานตามรหัสของ Toggle Code ขั้นตอนการทำงานแสดงได้ดังรูป



รูปที่ 3.18 FLOW CHART ของโปรแกรมของชุด CONTROLLER

การทำงานของโปรแกรมในชุด Controller เป็นการนำ TOGGLE CODE ซึ่งคือโปรแกรมมาทำงานขั้นการทำงานมีดังนี้

7.1 ทำการ Set CPU ให้มีการทำงานตามที่โปรแกรมต้องการและกำหนดตัวชี้สำหรับ Toggle Code, Stack

7.2 ตรวจสอบมีโปรแกรม PLC อยู่ในหน่วยความจำหรือไม่ ในกรณีที่มีโปรแกรม PLC อยู่ในหน่วยความจำ ไปยังข้อ 7.6 ในกรณีที่ไม่มีโปรแกรม PLC อยู่ในหน่วยความจำไปยังข้อ 7.3

7.3 ตรวจสอบมีข้อมูลเข้ามาหรือไม่ ในกรณีที่มีข้อมูลเข้ามาไปยังข้อ 7.4 ในกรณีที่ไม่มีข้อมูลเข้ามาไปยังข้อ 7.3

7.4 อ่านข้อมูลที่ส่งมาจากคอมพิวเตอร์

7.5 กำหนดค่าตัวชี้สำหรับ Toggle Code

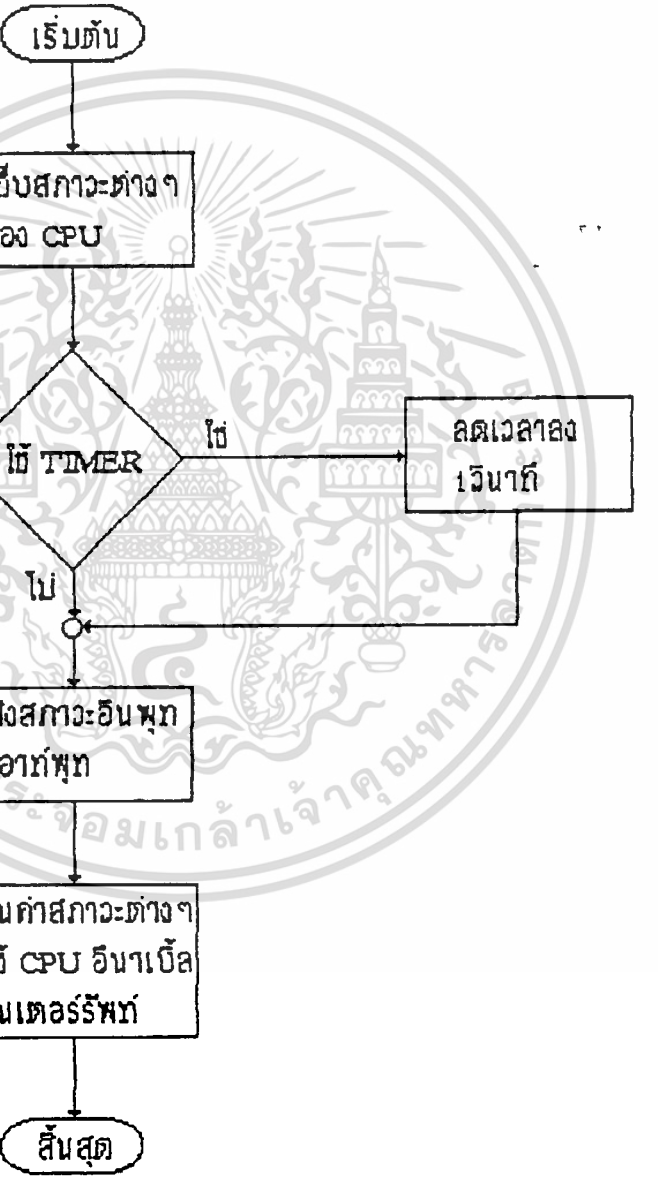
7.6 ทำงานตาม Toggle Code ที่อยู่ในหน่วยความจำ

7.7 ค่าของ Toggle Code คือ END หรือไม่ในกรณีที่ Toggle Code คือ END ไปข้อ 7.8 ในกรณีที่ Toggle Code ไม่ใช่ END ไปยังข้อ 7.5

7.8 ตรวจสอบมีการส่งข้อมูลใหม่จากคอมพิวเตอร์หรือไม่ ในกรณีที่มีการส่งข้อมูลจากเครื่องคอมพิวเตอร์ไปยังข้อ 7.4 ในกรณีที่ไม่มีการส่งข้อมูลจากเครื่องคอมพิวเตอร์ไปยังข้อ 7.5

8. การขัดจังหวะการทำงานของ CPU

การทำงานของชุด Controllerเนื่องจากมีการทำงานเกี่ยวกับเวลาจึงต้องมี การขัดจังหวะการทำงานของ CPU ซึ่งแสดงดังรูป



รูปที่ 3.19 FLOW CHART การแสดงโปรแกรมการขัดจังหวะในชุด CONTROLLER

การขัดจังหวะการทำงานของชุด CONTROLLER มีขั้นตอนการทำงานดังนี้

8.1 เก็บค่าสภาวะต่างๆของ CPU

8.2 ตรวจสอบมีการใช้ Timmer หรือไม่ ในกรณีที่มีการใช้ Timmerไปยังข้อ 8.6 ในกรณีที่ไม่มีการใช้ Timmer ไปยังข้อ 8.3

8.3 ส่งสภาวะของชุด Controller ไปยังเครื่องคอมพิวเตอร์

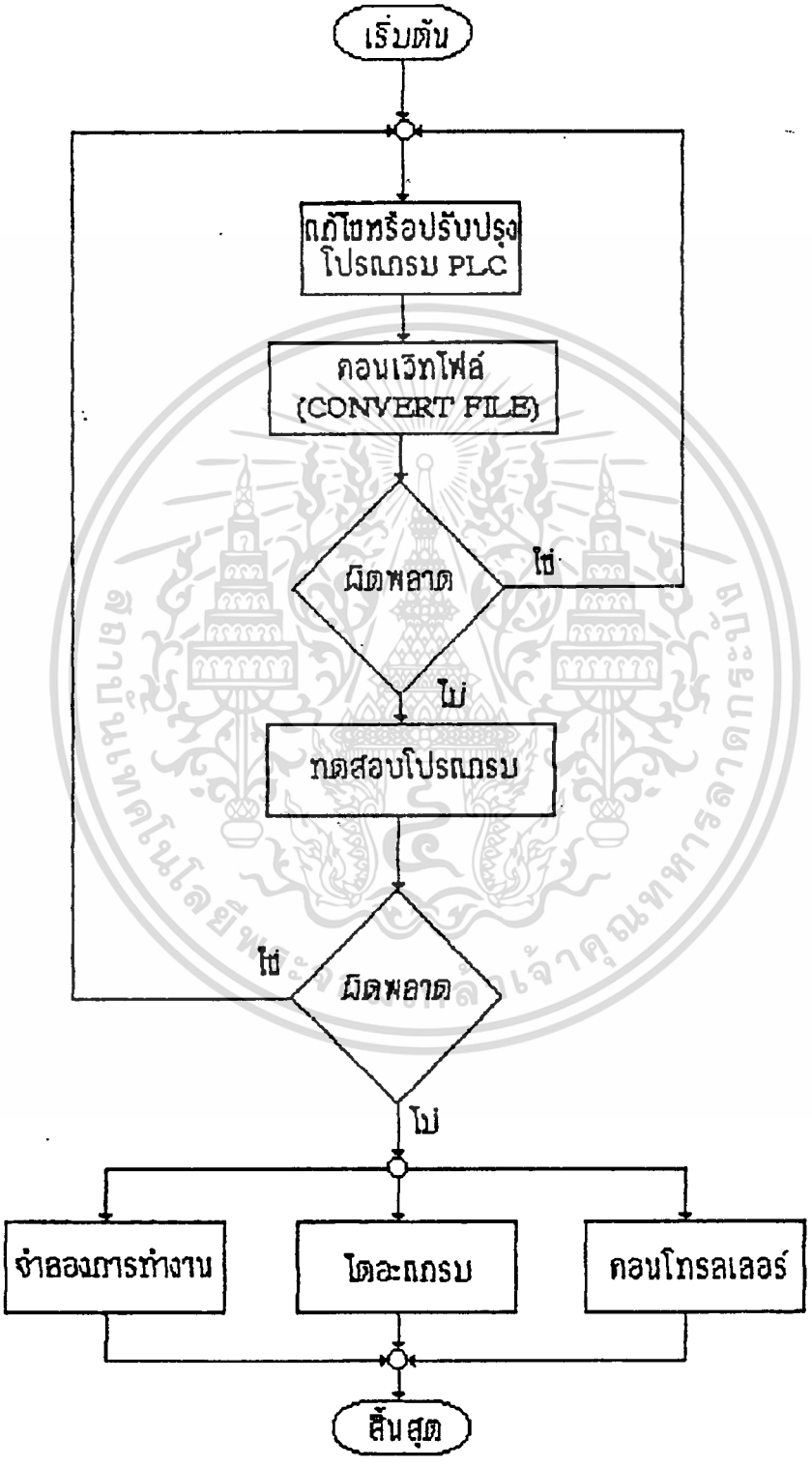
8.4 คืนค่าสภาวะต่างๆของ CPU

8.5 ออกจากโปรแกรมขัดจังหวะ

8.6 ลดค่าเวลาในหน่วยความจำลง 1 วินาที แล้วไปยังข้อ 8.3

ขั้นตอนการเขียนโปรแกรม PLC ภาษาบูลสีนโดยใช้โปรแกรมจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้

โปรแกรมจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้เป็นการเขียนโปรแกรมภาษาบูลสีนของ PLC การเขียนโปรแกรมจะเป็นลักษณะของ Text File แล้วจากนั้นจึงนำ Text File ที่ได้ไปจำลองการทำงาน ซึ่งสามารถที่จะอธิบายขั้นตอนการทำงานของโปรแกรมได้ดังรูป



รูปที่ 3.20 FLOW CHART ขั้นตอนการเขียนโปรแกรม PLC ภาษาบูลลีน

จากรูป 3.20เป็นขั้นตอนการทำงานคือในขั้นแรกต้องทำการเขียนโปรแกรม PLC ด้วยภาษามูลลีนซึ่งจะเก็บอยู่ใน TEXT FILE จากนั้นจะทำการเปลี่ยน คำสั่งของภาษามูลลีนใน TEXT FILE ให้เป็นรหัส TOGGLE CODE ที่สามารถนำไปใช้ในส่วนอื่นของโปรแกรม การที่ต้องเปลี่ยนคำสั่งของ PLC จาก TEXT FILE เป็น TOGGLE CODE เนื่องจากการนำคำสั่งของ PLC ที่เป็น ASSCII นั้นไม่สามารถที่จะนำไปใช้งานได้สะดวกเหมือน TOGGLE ที่เป็นตัวเลขโดยที่ขั้นตอนนี้จะตรวจสอบด้วยว่า คำสั่งของ PLC ที่เป็นภาษามูลลีนนั้นผู้ใช้เขียนถูกต้องหรือไม่ ในกรณีที่ไม่ถูกต้องก็จะต้องให้ผู้ใช้กลับไปแก้ไขโปรแกรมใหม่ ในกรณีที่ถูกต้องขั้นตอนต่อมาเป็นการทดลองจำลองการทำงานของโปรแกรม PLC ที่ผู้ใช้เขียนขึ้นในการทดลองการทำงานนี้จะเป็นตัวที่ทดสอบการเขียนโปรแกรมของผู้ใช้ว่าผิดหลักการเขียนโปรแกรม PLC หรือไม่ ในกรณีที่ผิดจะต้องกลับไปแก้ไขโปรแกรมใหม่ ในกรณีที่ถูกจะเป็นขั้นตอนที่ผู้ใช้จะเลือกว่าต้องการที่จำลองการทำงานของโปรแกรมหรือต้องการดู Timing Diagram หรือต้องการส่งโปรแกรมไปยังชุด Controller

ตารางที่ 3 การเคลื่อนที่ของข้อมูลคำสั่ง PLC

คำสั่ง	ก่อนกระทำตามคำสั่ง	หลังกระทำตามคำสั่ง
LD	-	PUSH ( )
LD NOT	-	PUSH ( )
AND	POP ( )	PUSH ( )
AND NOT	POP ( )	PUSH ( )
OR	POP ( )	PUSH ( )
OR NOT	POP ( )	PUSH ( )
AND LD	POP ( )	-
	POP ( )	PUSH ( )
OR LD	POP ( )	-
	POP ( )	PUSH ( )
CNT	POP ( )	PUSH ( )
	POP ( )	PUSH ( )
OUT	POP ( )	-
OUT NOT	POP ( )	-
END	-	-

บทที่ 4

ผลของโครงงาน

การทดสอบ

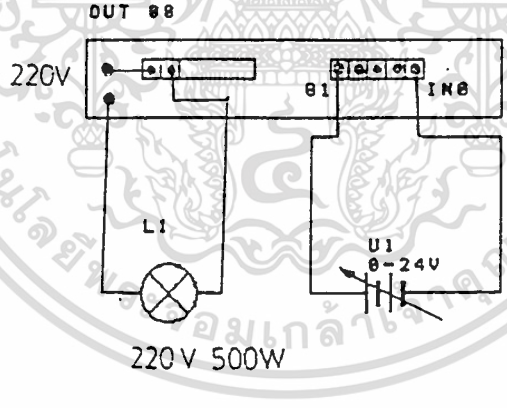
การทดสอบเครื่องจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้ แบ่งออกเป็น 2 ส่วน

ส่วน HARD WARE

1.การทดสอบภาคอินพุท

การทดสอบภาคอินพุท เป็นการทดสอบหาค่าของระดับแรงดันที่ทำให้อินพุทเป็น logic "1" และ logic "0" มีลำดับขั้นตอนดังนี้

1.1 ต่อวงจรดังรูปที่ 4.1



รูปที่ 4.1 การต่อวงจร ทดสอบภาคอินพุท

1.2 ต่อสายรับส่งข้อมูลระหว่างชุด Controller กับเครื่องคอมพิวเตอร์ดังรูป

รูปที่ 4.2 การต่อสายรับส่งข้อมูลระหว่าง COMPUTER และชุด CONTROLLER

การต่อสายรับส่งข้อมูลระหว่างชุด COMPUTER และชุด CONTROLLER

1.3 Runโปรแกรม PLCSIM

1.4 เขียนโปรแกรม PLC ดังต่อไปนี้

```
LD 00
OUT 500
END
```

1.5 ส่งโปรแกรมที่ Convert แล้วไปยังชุด Controlle

1.6 ปรับแรงดัน U1 จาก 0 V ถึง 24 V จนกระทั่ง เอาท์พุทที่ตำแหน่ง 500 "ON" ค่าแรงดันที่อ่านได้คือ 15 V

1.7 ปรับแรงดัน U1 จาก 24 V ถึง 0 V จนกระทั่งเอาท์พุทที่ตำแหน่ง 500 "OFF" ค่าแรงดันที่อ่านได้คือ 6 V

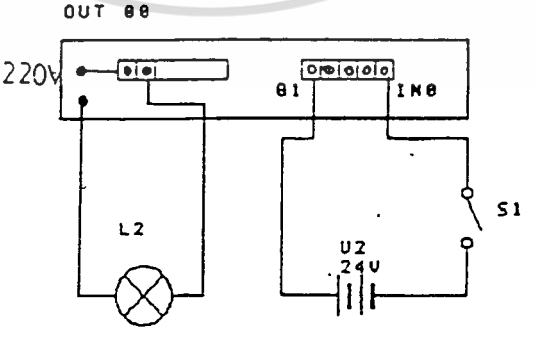
แรงดันที่ LOGIC "1" คือ 15-24 V

แรงดันที่ LOGIC "0" คือ 0-6 V

2.การทดสอบภาคเอาท์พุท

การทดสอบภาคเอาท์พุทเป็นการทดสอบปริมาณกระแสที่จ่ายให้ภาระที่แรงดัน 220 VAC

2.1 ต่อวงจรดังรูป



220V 500W

รูปที่ 4.3 แสดงการต่อวงจรทดสอบภาค เอาท์พุท

2.2 ต่อสายรับส่งข้อมูลระหว่างชุด Controller กับเครื่อง Computer ดังรูป

รูปที่ 4.4 การต่อสายรับส่งข้อมูลระหว่าง COMPUTER และชุด CONTROLLER

2.3 รันโปรแกรม PLCSIM

2.4 เขียนโปรแกรม PLC ดังต่อไปนี้

```
LD  00
OUT 500
END
```

2.5 ส่งโปรแกรมที่ Convert แล้วไปยังชุด Controller

2.6 ทำให้อินพุทที่ 00 เป็น LOGIC "1"

2.7 อ่านค่ากระแสเอาท์พุทแล้วบันทึกผล ได้ค่าเท่ากับ 2.3 A

2.8 ทำให้อินพุทที่ 00 เป็น LOGIC "0"

2.9 อ่านค่ากระแสเอาท์พุทแล้วบันทึกผล ได้เท่ากับ 0 A

ภาคเอาท์พุทสามารถจ่ายกระแสให้ Load ได้เท่ากับ 2.3 A ที่แรงดัน 220 V_{AC}

ส่วน SOFT WARE

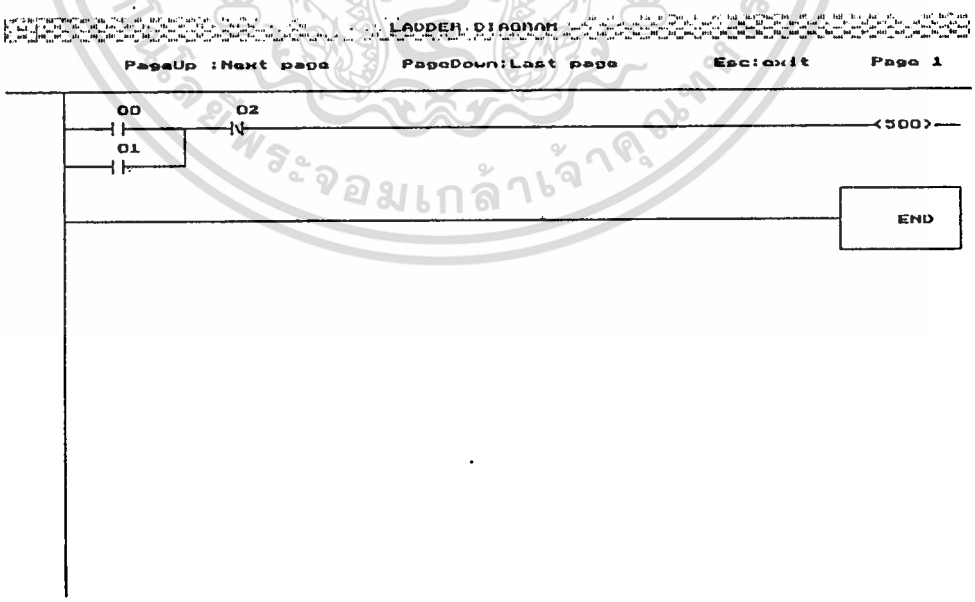
เมื่อต่อเครื่องควบคุม อินพุท/เอาท์พุท กับ คอมพิวเตอร์ ที่มีโปรแกรม PLCSIM.EXE แล้ว

การทดสอบที่ 1 ต้องการ RUN คำสั่งต่อไปนี้

```
LD   00
OR   01
AND-NOT 02
OUT  500
END
```

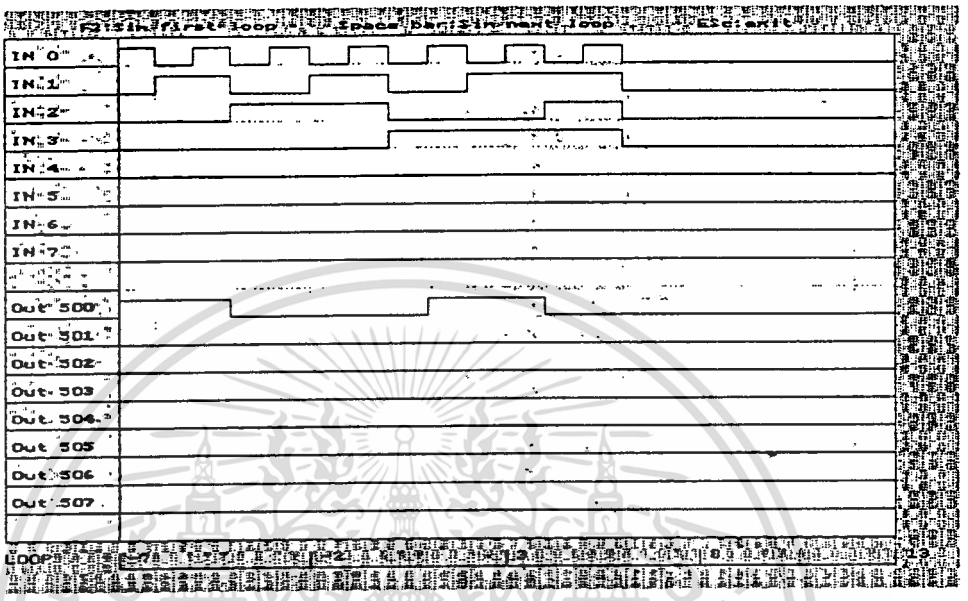
เมื่อนำคำสั่งมา RUN บน PLCSIM แล้วผลปรากฏดังนี้

LADDER DIAGRAM



รูปที่ 4.5 LADDER DIAGRAM ของโปรแกรม TEST 1

TIMING DIAGRAM



รูปที่ 4.6 แสดง TIMING DIAGRAM

ตารางที่ 4 ผลสภาวะ อินพุท/เอาท์พุทจากการจำลองการทำงาน

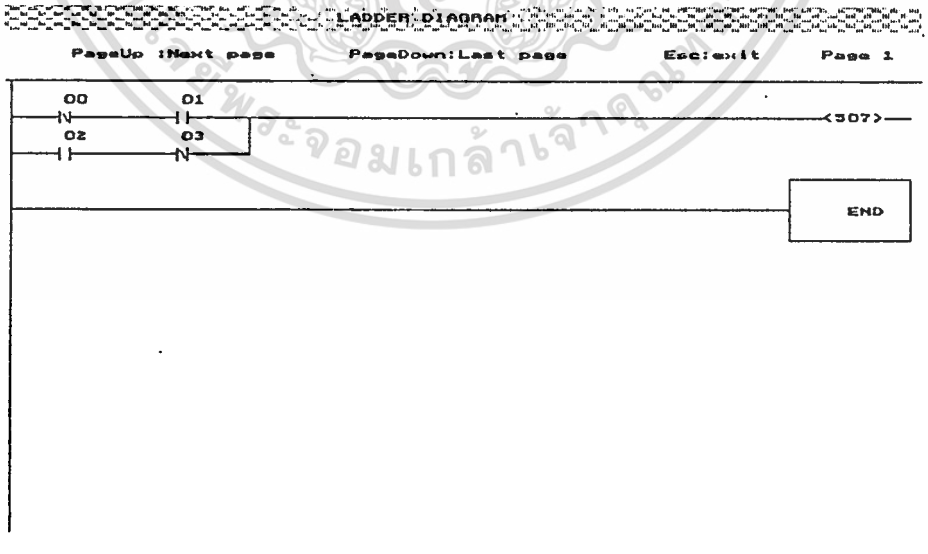
00	01	02	500
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

การทดสอบที่ 2 ต้องการ RUN คำสั่งต่อไปนี้

```
LD-NOT    00
AND       01
LD        02
AND-NOT   03
OR-LD
OUT       507
END
```

เมื่อนำคำสั่งมา RUN บน PLCSIM ผลปรากฏดังนี้

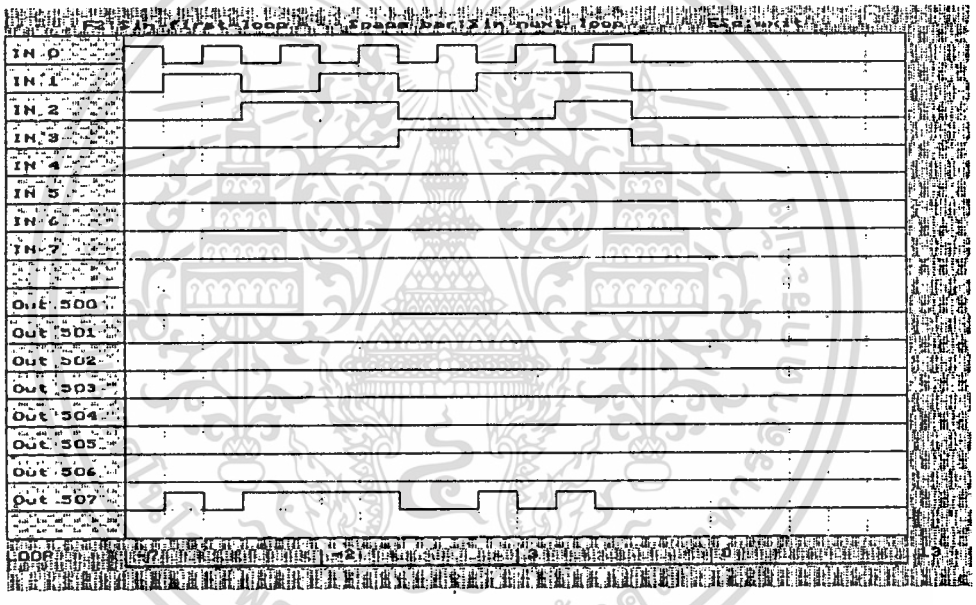
LADDER DIAGRAM



รูปที่ 4.7 LADDER DIAGRAM ของโปรแกรม TEST 2

ตารางที่ 5 ผลสภาวะอินพุท/เอาท์พุทจากการจำลองการทำงาน

00	01	02	03	507
0	0	0	0	0
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0



รูปที่ 4.8 TIMING DIAGRAM ของโปรแกรม TEST 2

ตารางที่ 6 ผลสภาวะอินพุท/เอาท์พุท จากการจำลองการทำงาน

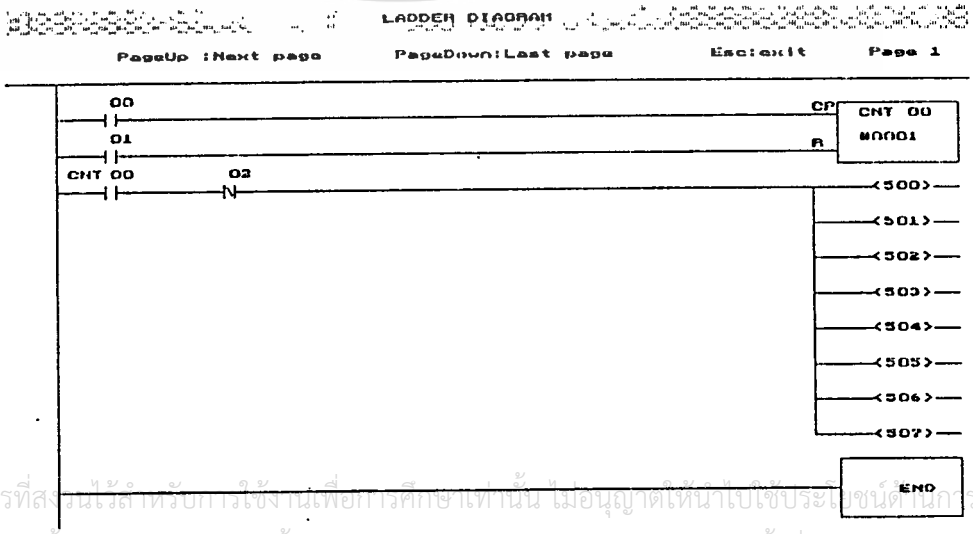
INPUT				OUTPUT	
00	TIM	01	02	TIM 00	507
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	0	0
0	1	0	1	0	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	1	1
1	0	0	1	1	1
1	0	1	0	1	0
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	0
1	1	1	1	1	1

การทดสอบที่ 4 ต้องการ RUN คำสั่งต่อไปนี้

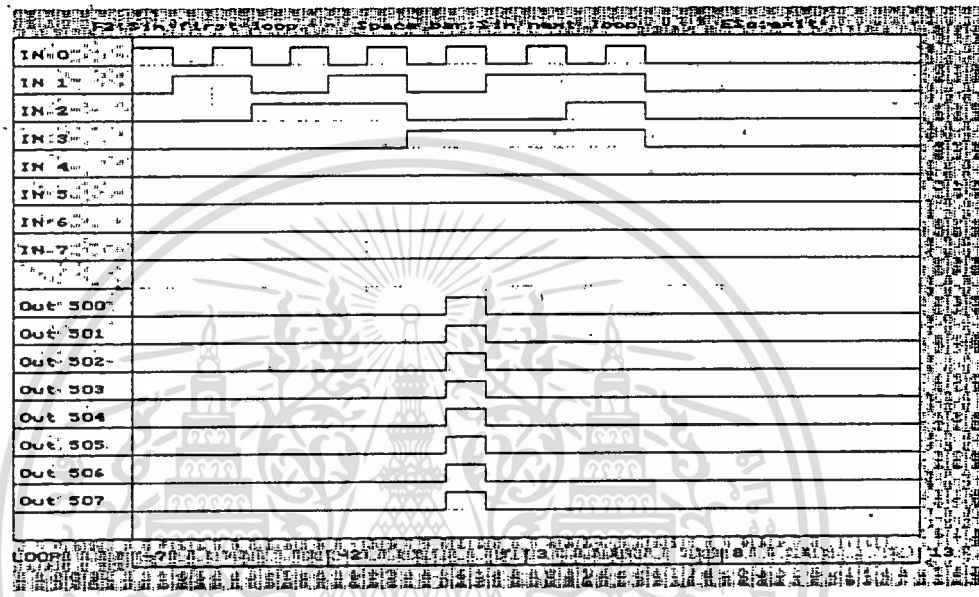
```
LD   00
LD   01
CNT 00   #0001
LD   CNT    00
AND-NOT    02
OUT 500
OUT 501
OUT 502
OUT 503
OUT 504
OUT 505
OUT 506
OUT 507
END
```

เมื่อนำมา RUN บน PLCSIM ผลปรากฏดังนี้

LADDER DIAGRAM



รูปที่ 4.10 แสดง LADDER DIAGRAM ของโปรแกรม TEST 4



รูปที่ 4.11 แสดง TIMING DIAGRAM ของโปรแกรม TEST 4

ตารางที่ 7 ผลสภาวะอินพุท/เอาท์พุท

00	01	CNT 00
0	0	0
0	1	0
1	0	1
1	1	0

ตารางที่ 8 ผลสภาวะอินพุท/เอาท์พุทจากการจำลองการทำงาน

CNT 00	02	500-507
0	0	0
0	1	0
1	0	1
1	1	0

ผลการทำงานเหมือนกับ PLC จริง

บทที่ 5

ข้อสรุปและข้อเสนอแนะ

เครื่องจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้เป็นเครื่องที่ใช้สำหรับจำลองการทำงานของเครื่องควบคุมแบบโปรแกรมได้ โครงงานนี้ประกอบด้วยส่วน HARDWARE และ SOFTWARE มีการทำงานได้ตามวัตถุประสงค์ที่ตั้งไว้ แต่เนื่องจากโครงงานนี้เป็นโครงงานใหม่ที่เริ่มพัฒนาเป็นครั้งแรก จึงมีข้อบกพร่องและข้อเสนอแนะต่างๆดังนี้

ปัญหาของโครงงาน

ส่วน HARDWARE

1.เอาท์พุทของชุด Controller ที่เป็น Solid State Relay ใช้กับภาระที่เป็นโหลดตัวต้านทานเท่านั้น

การแก้ไข

ปรับปรุงภาค Solid State Relay ให้สามารถใช้กับภาระโหลดที่เป็นชนิดอื่น

2.การรับส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์และชุด Controller ต้องมีระยะห่างไม่เกิน 50 ฟุต เพื่อการรับส่งข้อมูลจะได้ถูกต้อง

การแก้ไข

เปลี่ยนการรับส่งข้อมูลจากมาตราฐาน RS-232 เป็นมาตราฐาน RS-485

3.เมื่อทำการรับส่งข้อมูล CPU จะต้องทำงานรับส่งข้อมูลเพียงอย่างเดียวไม่สามารถที่จะทำงานอย่างอื่นได้เนื่องจากการรับส่งข้อมูลเป็นการใช้ SOFTWARE

การแก้ไข

เปลี่ยนการรับส่งข้อมูลเป็นการใช้ Hardware แทน

ส่วน SOFTWARE

1.การแสดงผลทางด้านกราฟิคในส่วนของการดูสภาวะการทำงานของชุด Controller ไม่สามารถที่จะตอบสนอง อินพุท/เอาท์พุท ที่มีการเปลี่ยนแปลงอย่างรวดเร็วได้ เนื่องจากชุด Controller จะส่งข้อมูลไปยังโปรแกรมจำลองการทำงานของเครื่องควบคุมแบบโปรแกรมได้ทุกๆ 1 วินาที

2.การแสดงผลด้วย Timing Diagramไม่สามารถที่จะจำลองการทำงานในคำสั่ง Timmer ได้เนื่องจากการทำงานของโปรแกรมจำลองการทำงานของเครื่องควบคุมแบบโปรแกรมได้ใน 1 Loop นั้นเร็วกว่าค่าเวลาของ Timer จึงไม่สามารถที่จะแสดง Timing ของ Timerได้

<u>การแก้ไข</u>

ให้โปรแกรมแสดง Timing Diagram ทำงานตามฐานเวลา

<u>ข้อเสนอแนะ</u>

1.โปรแกรมจำลองการทำงานของเครื่องควบคุมแบบโปรแกรมได้ตัวนี้ มีคำสั่งของ PLC ในจำนวน 13 คำสั่ง สามารถที่ทำการปรับปรุงเพิ่มเติมคำสั่งให้มี คำสั่งพิเศษอื่นๆได้โดยการแก้ไข SOFTWARE ส่วนของการกำหนดอินพุท/เอาท์พุทและส่วนควบคุมการแสดงผล

2.พัฒนาการแสดง Ladder Diagram ในกรณีที่โปรแกรม PLC มีความซับซ้อนมากขึ้น

3.เพิ่มอินพุท/เอาท์พุทของโปรแกรมจำลองการทำงานของเครื่องควบคุมแบบโปรแกรมได้ให้มีจำนวนมากขึ้น

4.เพิ่มจำนวนอินพุท/เอาท์พุทของชุด Controller

ภาคผนวก ก คู่มือเครื่องจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้

คู่มือโปรแกรมจำลองการทำงานเครื่องควบคุมแบบโปรแกมได้

(PLC SIMMULATOR)

โปรแกรม PLCSIM เป็นโปรแกรมจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้(PLC) โดยใช้คำสั่งพื้นฐานจำนวน 13 คำสั่งโปรแกรมนี้ เป็นส่วนหนึ่งของวิชา Special Project ของ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ชุดโปรแกรมจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้ ประกอบด้วย

1.โปรแกรมจำลองการทำงาน เครื่องควบคุมแบบโปรแกรมได้ซึ่งเป็น Software

2.ชุด Controller ซึ่งเป็น Hardware

โปรแกรมจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้

โปรแกรมจำลองการทำงานเครื่องควบคุมได้ติดตั้งในแผ่นเก็บข้อมูลขนาด 5 1/4 นิ้วใน แผ่นข้อมูลประกอบด้วยไฟล์ต่างๆ ที่สำคัญดังนี้

1. PLCSIM.EXE
2. EGAVGA.BGI
3. HERC.BGI
4. DEMO.PLC
5. LOGIC.PLC
6. README.DOC

คุณสมบัติของเครื่องคอมพิวเตอร์ที่ใช้ร่วมกับโปรแกรมจำลองการทำงานของเครื่องควบคุมแบบโปรแกรมได้

1. PC AT หรือ PC COMPATIBLE
2. DOS Version 3.3 ขึ้นไป
3. มอนิเตอร์แสดงผลแบบ VGA,EGA,VGA-MONO หรือ HERC

<u>ในกรณีที่ติดตั้งชุด Controller ต้องการอุปกรณ์เพิ่มเติมคือ</u>

1.พอร์ตการรับส่งข้อมูลแบบอนุกรมจำนวน1พอร์ต Serial Port(COM1,COM2)

2.สายสัญญาณรับส่งข้อมูลแบบอนุกรม 1 เส้น

<u>คุณสมบัติของโปรแกรมจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้</u>

1.ใช้หน่วยความจำ 136 KB

2.เป็นการจำลองการทำงานของ PLC โดยใช้ฟังก์ชั่นพื้นฐาน PLC ของ OMRON รุ่น C20 ที่เป็นภาษาบูลลีน คำสั่งที่ใช้มีดังนี้คือ

2.1 LD

LD XX

LD YYY

LD TR ZZ

LD CNT AA

LD TIM BB

2.2 LD NOT

LD-NOT XX

LD-NOT YYY

LD-NOT CNT AA

LD-NOT TIM BB

2.3 AND

AND XX

AND YYY

AND CNT AA

AND TIM BB

2.4 AND NOT

AND-NOT XX

AND-NOT YYY

AND-NOT CNT AA

AND-NOT TIM BB

2.5 OR

OR XX

OR YYY

OR CNT AA

OR TIM BB

2.6 OR NOT

OR-NOT XX

OR-NOT YYY

OR-NOT CNT AA

OR-NOT TIM BB

2.7 AND-LD

2.8 OR-LD

2.9 CNT AA #CCCC

2.10 TIM BB #DDDD

2.11 OUT

OUT YYY

OUT TR ZZ

2.12 OUT-NOT YYY

2.13 END

เมื่อ XX แทนหมายเลข SW

YYY แทนหมายเลข OUTPUT

ZZ แทนหมายเลข TEMPORY RELAY

AA แทนหมายเลข ตัวนับ

BB แทนหมายเลข ตัวตั้งเวลา

CCCC แทนจำนวนที่ต้องการให้นับ

DDDD แทนจำนวนเวลาที่ต้องการตั้ง

3.เป็นโปรแกรมที่ใช้งานกับเครื่องคอมพิวเตอร์ขนาด16บิต PC AT Compatible จอแสดงผลแบบ VGA COLOR

4.แสดงการจำลองการทำงานของโปรแกรม PLC ที่ผู้ใช้เขียนขึ้นทางจอมอนิเตอร์

5.แสดงการจำลองการทำงานด้วยTiming Diagramโดยแสดงได้ 20 Loop ต่อ 1 หน้า

6.จำนวน Loop สูงสุดที่จำลองการทำงานด้วย Timing Diagram 30,000 Loop

7.แสดง Ladder Diagram ทางจอมอนิเตอร์ได้ 20 หน้า

8.จำนวนเงื่อนไขที่แสดงติดต่อกันได้สูงสุด 9 เงื่อนไข

9.จำนวนคำสั่งที่เขียนได้สูงสุด 500 บรรทัด

10.มีจำนวนอินพุท 8 อินพุท (00-07)

11.มีเอาท์พุทจำนวน 8 เอาท์พุท (500-507)

12.ตั้งตั้งเวลา 16 ตัว (00-15) ค่าเวลา 0000-9999 วินาที

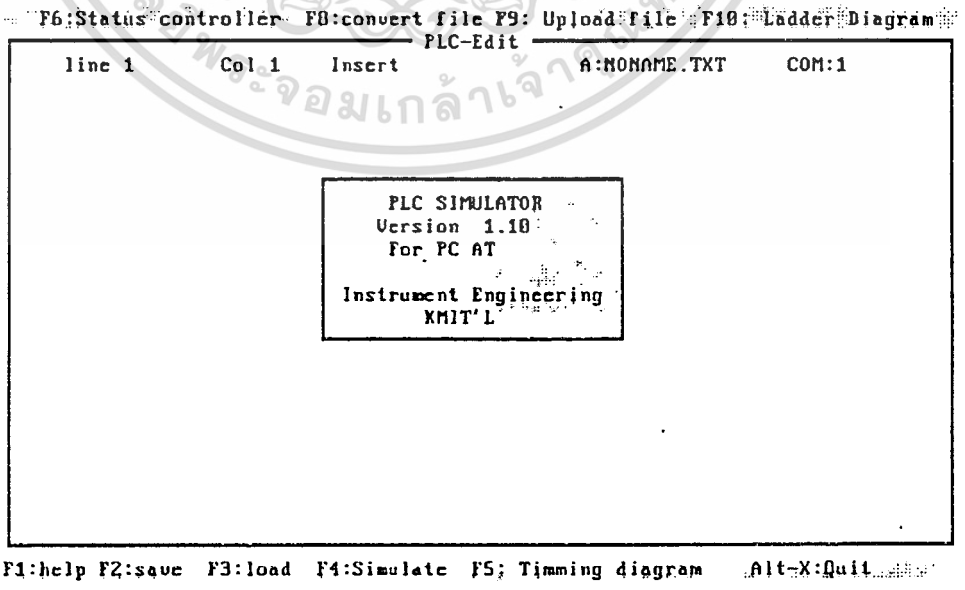
13.ฐานเวลา 1 วินาที

14.ตัวนับ 16 ตัว (00-15) ค่าการนับ 0000-9999 หน่วย

15.แสดงการสภาวะของชุด Controller (Hardware)

16.มีการรับส่งข้อมูลกับชุด Controller แบบอนุกรมตามแบบมาตราฐาน RS-232

17.จำนวน TR สูงสุดใช้ได้ 8 ตัว



รูปที่ ก.1 เมนูหลักของโปรแกรม PLCSIM

คำสั่งต่างๆของโปรแกรม PLCSIM

F1: Help

การเรียกขอความช่วยเหลือจากโปรแกรม เช่น รายละเอียดของ FUNCTION KEY ต่างๆ

F2: Save

การเก็บข้อมูลที่ผู้ใช้เขียนขึ้นไว้ในแผ่นข้อมูลโดยระบุชื่อของแฟ้มข้อมูลที่จะทำการจัดเก็บ

F3: Load

การอ่านข้อมูลจากแผ่นเก็บข้อมูล

F4: Simulate

การจำลองการทำงานของโปรแกรมที่ผู้ใช้เขียนจะแสดงสภาวะของอินพุทและเอาท์พุทในรูปของหลอดไฟ โดยอินพุทที่นำมาทำการจำลองการทำงานใช้ ดีย์บอร์ดตัวเลข 0-7

F5: Timing

การจำลองการทำงานของโปรแกรมที่ผู้ใช้เขียนขึ้นจะแสดงสภาวะของอินพุทและเอาท์พุทในรูปของ Diagram โดยที่อินพุทที่จะนำมาจำลอง การทำงานจะอยู่ในรูปของ TEXT FILE

F6: Display Controller

การแสดงสภาวะของอินพุทและเอาท์พุทของชุด Controller ทางจอมอนิเตอร์

F7: Write to File

การนำโปรแกรมที่ผู้ใช้เขียนขึ้นไปเก็บไว้ที่แผ่นข้อมูลโดยการเปลี่ยนชื่อของแฟ้มข้อมูล

F8: Convert File

การแปลงโปรแกรมที่ผู้ใช้เขียนขึ้นให้อยู่ในรูปที่โปรแกรม PLCSIM สามารถที่จะนำไปใช้งานได้การแปลงโปรแกรมนี้ผู้ใช้จะต้องกระทำก่อนทุกครั้งที่จะทำการ Simulate,Timing,Upload File

F9: Upload File

การส่งโปรแกรม PLC ไปยังชุด Controller

F10: Ldder Diagram

การแสดง Ladder Diagram จากโปรแกรม PLC ภาษามูลลีนที่ผู้ใช้เขียน

คีย์ Insert

การพิมพ์แทรก

Overwrite

การพิมพ์ทับ

Cap Lock

ใช้ตัวพิมพ์ใหญ่

C:DEMO-PLC

แสดงชื่อ File ที่กำลังเขียนโปรแกรมและ Drive

COM1:

แสดงหมายเลขของพอร์ตอนุกรมที่ใช้งานอยู่

<u>คำสั่งอื่นๆ ที่ใช้ในโปรแกรม PLCSIM</u>

Ctrl-K

ลบข้อมูลตั้งแต่ตำแหน่งเคอร์เซอร์ถึงสิ้นสุดบรรทัดโดยสามารถเรียกข้อมูลที่ลบไปคืนมาได้

Ctrl-L

เปลี่ยนช่องเก็บข้อมูล

Ctrl-P

เปลี่ยนตำแหน่งของพอร์ตอนุกรม

Ctrl-QA

ค้นหาและเปลี่ยนข้อความตั้งแต่ตำแหน่งเคอร์เซอร์จนถึงสิ้นสุดแฟ้มข้อมูล

Ctrl-QF

ค้นหาข้อความตั้งแต่ตำแหน่งเคอร์เซอร์จนถึงสิ้นสุดแฟ้มข้อมูล

Ctrl-R

เรียกข้อมูลที่ลบโดย Ctrl-K กลับคืนมา

Ctrl-Y

ลบข้อมูล ณ บรรทัดที่ตำแหน่งเคอร์เซอร์แสดงอยู่

Ctrl-Home

เลื่อนเคอร์เซอร์ไปยังตำแหน่งแรกสุดของแฟ้มข้อมูล

Ctrl-End

เลื่อนเคอร์เซอร์ไปตำแหน่งท้ายสุดของ File

PageUp

เลื่อนเคอร์เซอร์ไปยังตำแหน่งของหน้าที่ผ่านมา

PageDown

เลื่อนเคอร์เซอร์ไปยังตำแหน่งของหน้าต่อไป

Home

เลื่อนเคอร์เซอร์ไปยังตำแหน่งแรกสุดของบรรทัด

End

เลื่อนเคอร์เซอร์ไปยังตำแหน่งท้ายสุดของบรรทัด

เลื่อนเคอร์เซอร์ขึ้น 1 บรรทัด

เลื่อนเคอร์เซอร์ลง 1 บรรทัด

เลื่อนเคอร์เซอร์ไปทางขวา 1 Column

เลื่อนเคอร์เซอร์ไปทางซ้าย 1 Column

Ctrl-I

แสดงข้อมูลเกี่ยวกับโปรแกรม PLCSIM

Alt-X

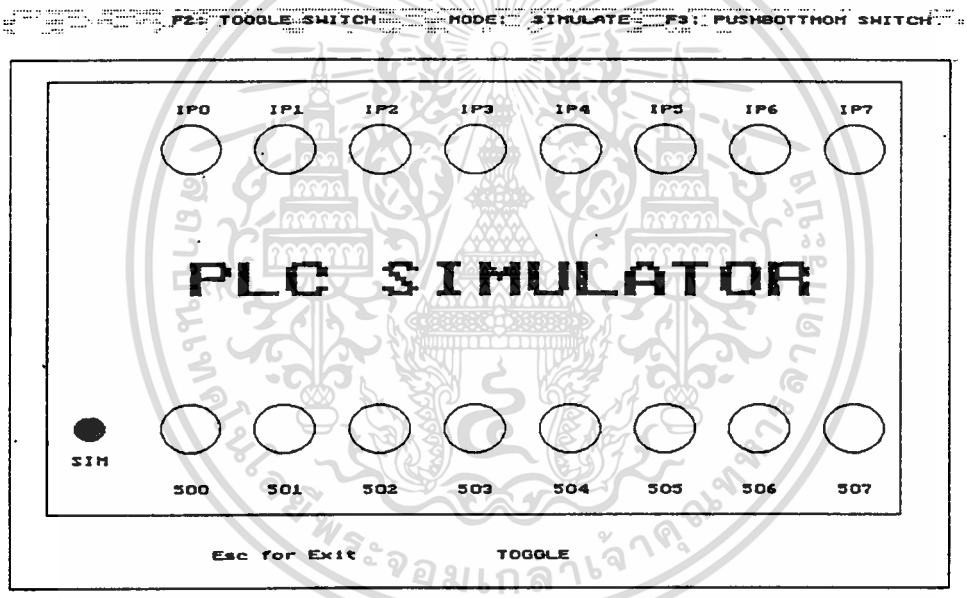
ออกจากโปรแกรม PLCSIM

Esc

ออกจากฟังก์ชั่น

<u>การแสดงผลทางฟังก์ชั่นต่างๆ</u>

การจำลองการทำงาน (SIMULATE)



รูปที่ ก.2 แสดงการจำลองการทำงานของโปรแกรม PLCSIM

1.ชื่อการทำงานของฟังก์ชั่น

2.หมายเลขอุปกรณ์อินพุท

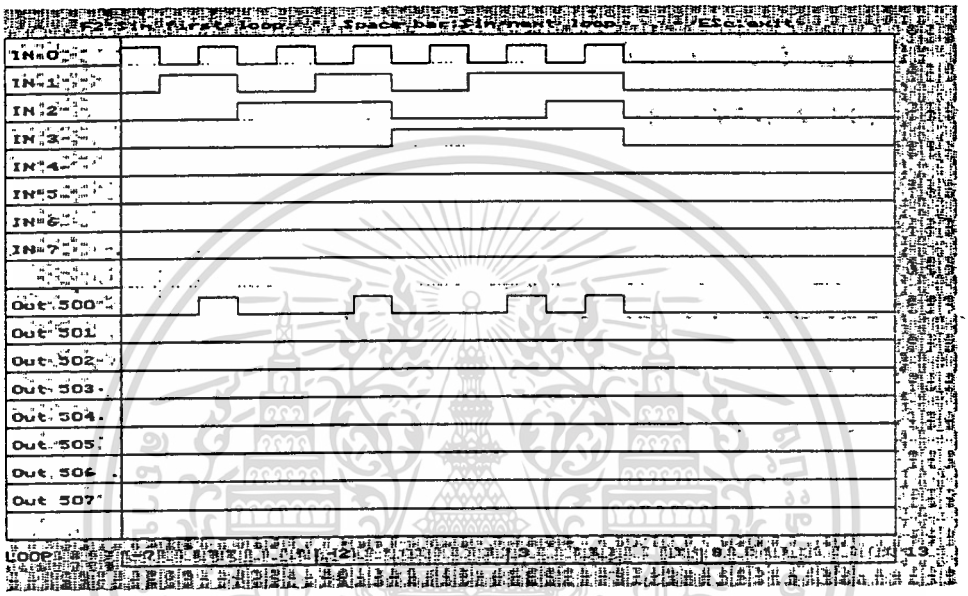
3.ตัวแสดงผลอินพุท

4.หมายเลขอุปกรณ์เอาท์พุท

5.ตัวแสดงผลเอาท์พุท

6.ชื่อการทำงานของฟังก์ชั่นจำลองการทำงานหรือสภาวะ Controller

การแสดง TIMING DIAGRAM



รูปที่ ก.3 แสดง TIMING DIAGRAM ของโปรแกรม PLCSIM

1.F2: การแสดง Timing Diagram ที่ตำแหน่งเริ่มต้น

2.Space Bar or Other Key การแสดง Timing Diagram ที่ตำแหน่งถัดไปโดยมีการเปลี่ยนแปลง 20 Loop ต่อ 1 หน้า

3.Esc ออกจากการแสดง Timing Diagram

4.หมายเลขอุปกรณ์อินพุท

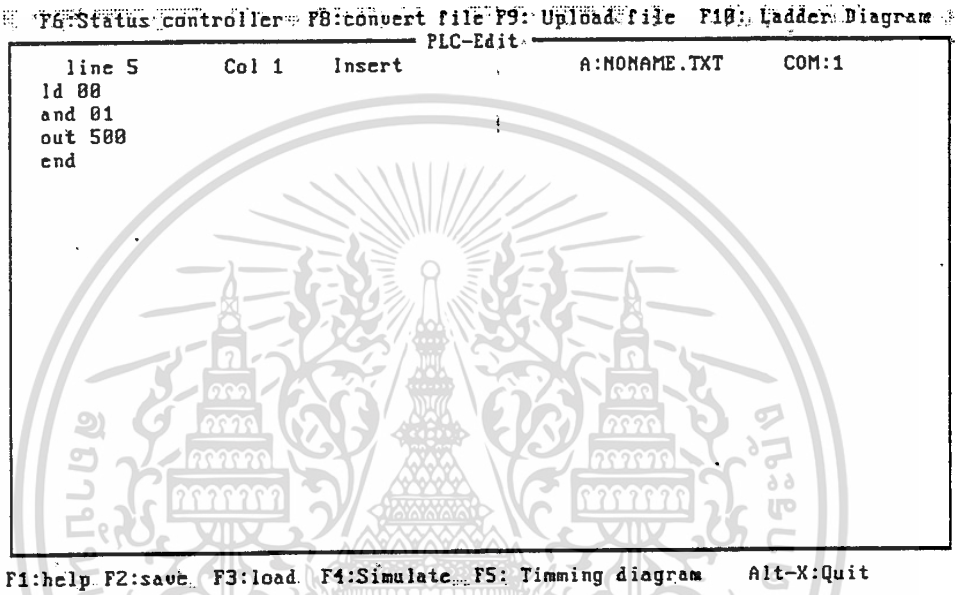
5.หมายเลขอุปกรณ์เอาท์พุท

6.จำนวน Loop ที่กระทำ

7.Timing Diagram ของอินพุท

8.Timing Diagram ของเอาท์พุท

การแสดง LADDER DIAGRAM



รูป ก.4 แสดง LADDER DIAGRAM ของโปรแกรม PLCSIM

1.ชื่อของฟังก์ชั่น

2.PageUp แสดง Ladder Diagram ที่ผ่านมา

3.PageDown แสดง Ladder Diagram หน้าต่อไป

4.Esc ออกจากฟังก์ชั่น

5.บอกจำนวนหน้า Ladder Diagram

6.ส่วนแสดง Ladder Diagram

รูปแบบของแฟ้มข้อมูลสำหรับอินพุทของ TIMING DIAGRAM

การเขียนอินพุทจะใช้ตัวเลข 0 และ 1 แทนสภาวะอินพุทคือ 0 คือลอจิกศูนย์ 1 คือลอจิก 1 โดย 1 Loop จะต้องป้อนอินพุททุกตัว เมื่อจบอินพุทจะต้องจบด้วย E เสมอ

อินพุทที่

XXXXXXXX 0 1 2 3 4 5 6 7

XXXXXXXX 0 1 2 3 4 5 6 7

E END OF INPUT

ตัวอย่าง

10100110

01000110

01111111

E

คำสั่งของโปรแกรม PLC ที่สามารถใช้ได้ใน PLCSIM

คำสั่งที่ใช้ในโปรแกรมจะเป็นคำสั่งพื้นฐาน โดยจะอิงจากคำสั่งของเครื่อง PLC ของ OMRON-TRISAK CO.,LTD รุ่น C20

คำสั่งพื้นฐาน

LD	หมายถึง	LOAD
LD-NOT	"	LOAD NOT
AND	"	AND
AND-NOT	"	AND NOT
OR	"	OR
OR-NOT	"	OR NOT
AND-LD	"	AND LOAD
OR-LD	"	OR LOAD
CNT	"	COUNTER
TIM	"	TIMER
OUT	"	OUT
OUT-NOT	"	OUT NOT
END	"	END OF PROGRAM

รูปแบบคำสั่งของ PLC ที่ใช้ในโปรแกรม PLCSIM

คำสั่ง	ตัวอย่าง
1.LOAD	
LD XX	LD 01
LD XXX	LD 500
LD TR XX	LD TR 01
LD CNT XX	LD CNT 01 #0005
LD TIM XX	LD TIM 01 #0005

2.LOAD NOT

LD-NOT XX	LD-NOT 01
LD-NOT XXX	LD-NOT 500
LD-NOT CNT XX	LD-NOT CNT 01 #0005
LD-NOT TIM XX	LD-NOT TIM 01 #0005..

3.AND

AND XX	AND 01
AND XXX	AND 500
AND CNT XX	AND CNT 01
AND TIM XX	AND TIM 01

4.AND NOT

AND-NOT XX	AND-NOT 01
AND-NOT XXX	AND-NOT 500
AND-NOT CNT XX	AND-NOT CNT 01 #0005
AND-NOT TIM XX	AND-NOT TIM 01 #0005

5.OR

OR XX	OR 01
OR XXX	OR 500
OR CNT XX	OR CNT 01
OR TIM XX	OR TIM 01

6.OR NOT

OR-NOT XX	OR-NOT 01
OR-NOT XXX	OR-NOT 500
OR-NOT CNT XX	OR-NOT CNT 01 #0005
OR-NOT TIM XX	OR-NOT TIM 01 #0005

7.AND LOAD

AND-LD	AND-LD

8.OR LOAD

OR-LD	OR-LD

9.COUNTER

CNT XX #XXXX	CNT 01 #0005

10.TIMER

TIM XX #XXXX	TIM XX #0005

11.OUT

OUT XXX	OUT 500

12.OUT TR

OUT TR XX	OUT TR 01

13.OUT NOT

OUT-NOT XXX	OUT-NOT 500

14.END

END	END

เมื่อกำหนดให้

XX แทนหมายเลขอุปกรณ์อินพุท

XX แทนหมายเลขตัวตั้งเวลาหรือตัวนับ

XXX แทนหมายเลขอุปกรณ์เอาท์พุท

XXXXX แทนค่าของ Timmer หรือ Counter

สรุปรูปแบบคำสั่งของ PLC ที่ใช้ในโปรแกรม PLCSIM

1. ——— ——— LD 01

คำสั่ง ตำแหน่งอุปกรณ์ LD 500

2. ——— ——— #——— CNT 01 #0005

คำสั่ง ตำแหน่งอุปกรณ์ ค่าที่กำหนด

3. ——— ——— ——— LD CNT 01

คำสั่ง คำสั่ง ตำแหน่งอุปกรณ์

ข้อความแสดงข้อผิดพลาดต่างๆ

Code Mismatch in PLC

หมายถึง มีการใช้คำสั่งที่แตกต่างจากคำสั่งที่ใช้ในโปรแกรม PLCSIM

Line To Long

หมายถึง จำนวนคอลัมม์ต่อบรรทัดมีค่ามากเกินกว่าที่โปรแกรมกำหนดให้ใช้

Data Point Not Assigned

หมายถึงค่าของตำแหน่งอุปกรณ์ไม่ได้กำหนดไว้ในโปรแกรมที่ผู้ใช้เขียนขึ้น

Invalid Format Of Counter Or Timer

หมายถึง การกำหนดค่าตัวแปรของคำสั่ง TIM, CNT ไม่ถูกต้อง

Not Found End Of program

หมายถึง ไม่พบจุดสิ้นสุดของโปรแกรม จะต้องมีคำสั่ง END ต่อท้ายโปรแกรมเสมอ

Syntax Error Stack Under!!!

หมายถึง การเขียนโปรแกรมไม่ถูกต้องผิดหลักการ

Sytax Error Stack Full!!!

หมายถึง การเขียนโปรแกรมไม่ถูกต้องผิดหลักการ

File Not Convert

หมายถึง ยังไม่มีการเปลี่ยนคำสั่ง PLC ที่ Text File ไปเป็นรหัสที่ใช้ในการทำงานของโปรแกรม PLCSIM

File Can Not Simulate

หมายถึง โปรแกรมที่ผู้ใช้เขียนขึ้นผิดหลักการ จึงไม่สามารถที่จะส่งไปยังชุด Controllerได้

Ladder Diagram Can Not Display

หมายถึง การเขียนโปรแกรม PLCไม่ถูกต้องจึงไม่สามารถที่จะแสดง Ladder Diagram ได้

Invalid Data Point Over

หมายถึงค่าของตำแหน่งอุปกรณ์มีค่ามากหรือน้อยกว่าที่โปรแกรมกำหนด

Fine Input Timing Not Found

หมายถึงไม่มีไฟล์ข้อมูลของ Timing

Invalid Valve Of Timer Or Counter

หมายถึงค่าของ TIM,CNT มีค่ามากกว่าที่โปรแกรมกำหนดให้ใช้ได้

<u>ตัวอย่างการใช้งานโปรแกรม PLCSIM</u>

มีโปรแกรมตัวอย่าง DEMO.PLC

<u>ภาษามูลลีน</u>

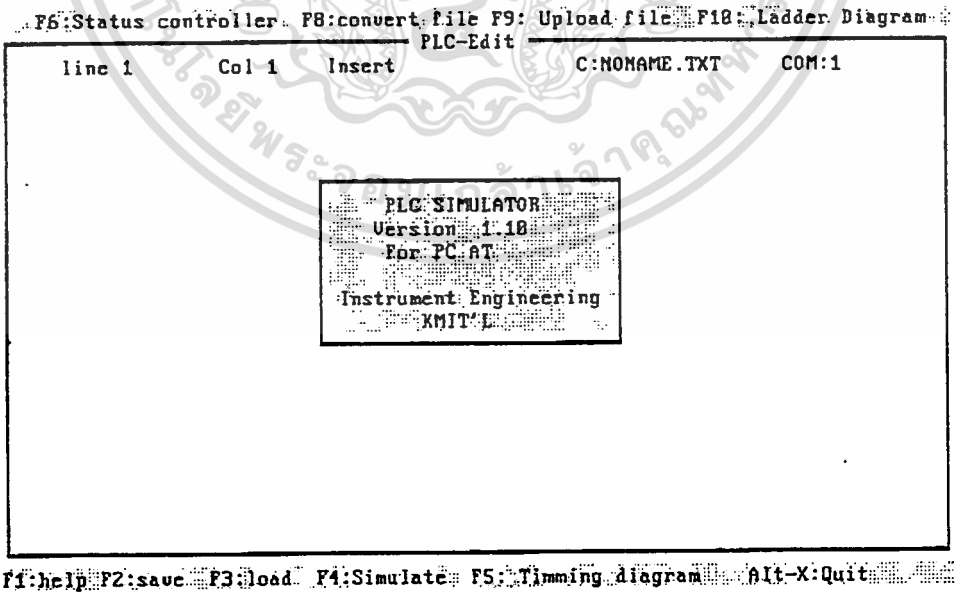
```
LD  00
AND 01
OUT 500
END
```

เราจะนำโปรแกรมนี้มาจำลองการทำงาน

1.นำแผ่นเก็บข้อมูลใส่ที่ Drive A พิมพ์

A>PLCSIM

ที่หน้าจอจะปรากฏดังรูป

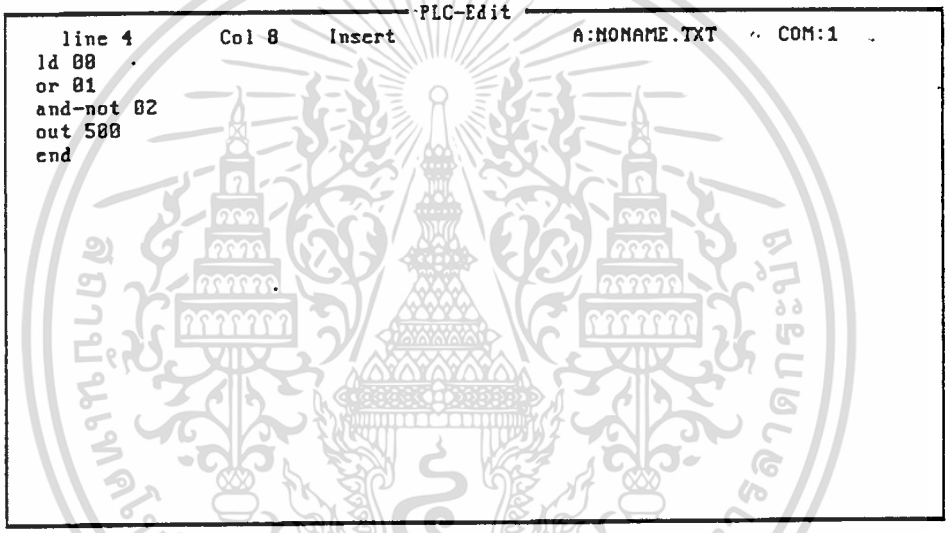


รูปที่ ก.5 แสดง MENU หลักโปรแกรม PLCSIM ของการทดลอง DEMO

2. กดคีย์ F3: Load File

ใส่ชื่อแฟ้มข้อมูล DEMO.PLC

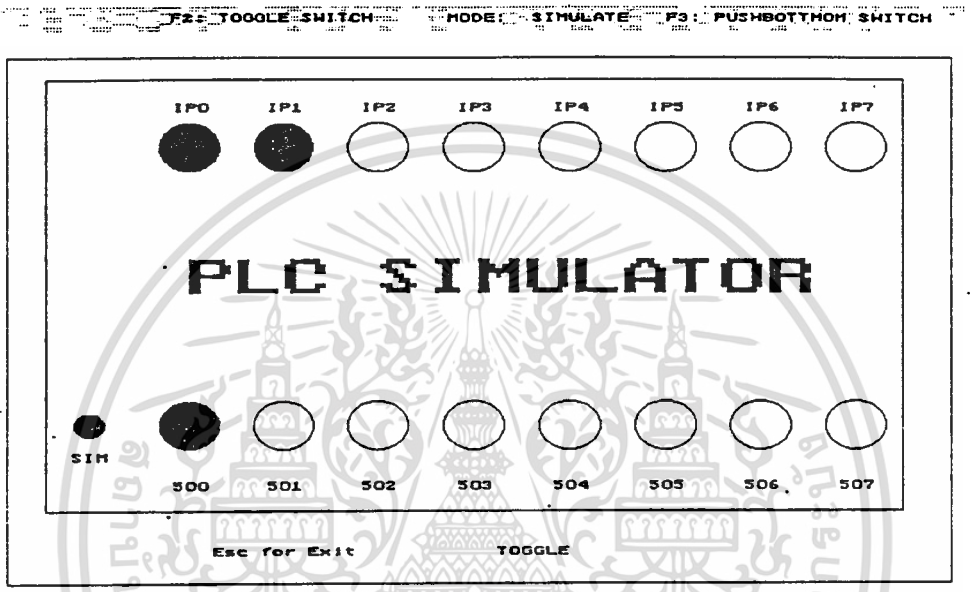
ที่หน้าจอจะปรากฏดังรูป



รูปที่ ก.6 โปรแกรม DEMO ที่เป็นภาษามูลลีน

3.กด F8: Convert File

4.กด F4: Simulate จำลองการทำงานของโปรแกรม



รูปที่ ก.7 แสดงเอาท์พุทเมื่อจำลองการทำงานของโปรแกรม DEMO

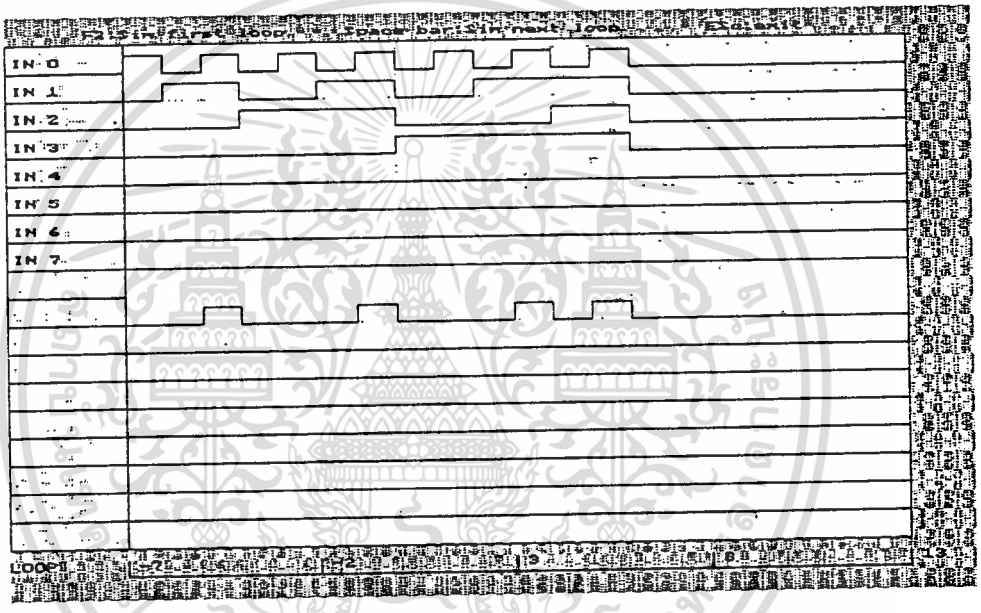
5.กดตัวเลข 0-7 เป็นอินพุท

6.กดเลข 0 เลข 1 สังเกตเอาท์พุท

7.กด ESC ออกจากการจำลองการทำงานโปรแกรมจะกลับสู่เมนูหลัก

8.กด F5: Timing Diagram ดู Timing Diagram โดยใส่อินพุท Timing Diagram ชื่อ LOGIC.PLC

9.โปรแกรมจะแสดงดังรูป

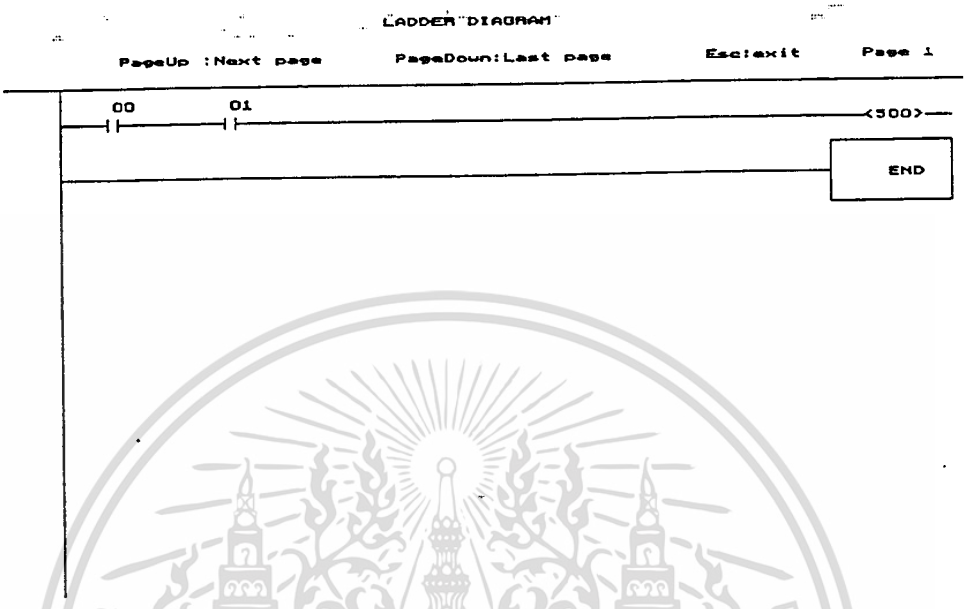


รูปที่ ก.8 แสดง TIMING DIAGRAM ของโปรแกรม DEMO

10. กด Space Bar เพื่อดู Timing Loop ต่อไป

11. กด Esc เมื่อต้องการเลิกการแสดง Timing Diagram

12. กด F10:Ladder Diagram เพื่อแสดง Ladder Diagramของโปรแกรมจะแสดงดังรูป



รูปที่ ก.9 แสดง LADDER DIAGRAM ของโปรแกรม DEMO

13.กด Esc เพื่อออกจากการแสดง Ladder Diagram กลับสู่เมนูหลัก

14.ในกรณีที่มีการติดตั้งชุด Controller กด F9: Upload File เพื่อส่งโปรแกรมไปยังชุด Controller จากนั้นทดลองป้อนอินพุทแล้วสังเกตเอาท์พุทที่ชุด Controller

15.กด F6: Display controller เพื่อแสดงสภาวะอินพุท/เอาท์พุท

16.กด Esc เมื่อเลิกการแสดงสภาวะอินพุท/เอาท์พุท

17.โปรแกรมจะกลับสู่เมนูหลัก ถ้าต้องการเลิกการทำงาน กด ALT-X

```
Help
                PLC SIMULATOR HELP

F1:  help
F2:  Save <filename>
F3:  Load <filename>
F4:  Simulate PLC program by keyboard (input)
F5:  Simulate PLC program by fileinput display output
     with timming (input)
F6:  Display status of controller unit
F7:  Write to file
F8:  Convert text file to toggle code
F9:  Transmitt program PLC to controller
F10: Display Ladder Diagram of program
Ctrl-QA   Replace  text          Ctrl-P    Change communication port
Ctrl-QF   Search   text          USE:      KEY 1 -> Com:1 KEY 2 -> Com:2
Ctrl-Y    Delete  line           Ctrl-K     Delete line can restoreo
Ctrl-R    Restore  line          Ctrl-L     Change drive
Page UP   Change page up         Ctrl-I     Information of program
Page Down Change page down
Home      First coloum           End       End of coloum

          ## PageDown to format of program ##   ## Esc for Exit ##
```

รูปที่ ก.10 แสดง HELP ของ PLCSIM หน้าที่ 1

```
Help
                Format of PLC Program
FUNCTION        FORMAT
LOAD            LD XX , LD XXX, LD TR XX, LD CNT XX
                LD TIM XX
LOAD NOT        LD-NOT XX, LD-NOT XXX, LD-NOT CNT XX
                LD-NOT TIM XX
AND             AND XX, AND XXX, AND CNT XXX, AND TIM XXX
AND-NOT         AND-NOT XX, AND-NOT XXX, AND-NOT CNT XX
                AND-NOT TIM XX
OR              OR XX, OR XXX, OR CNT XX, OR TIM XX
OR-NOT          OR-NOT XX, OR-NOT XXX, OR-NOT CNT XX
                OR-NOT TIM XX
AND LOAD        AND-LD
OR LOAD         OR-LD
COUNTER         CNT XX #XXXX
TIMMER          TIM XX #XXXX
END             END

This PLC SIMULATOR PROGRAM has timebase    1 sec
Input 8 channal (00-07) output 8 channal(500 - 507)
Timmer 16 ,counter 16 maximum value 9999  (00-15)

       ## PageUp for last page ##   ## Esc for Exit ##
```

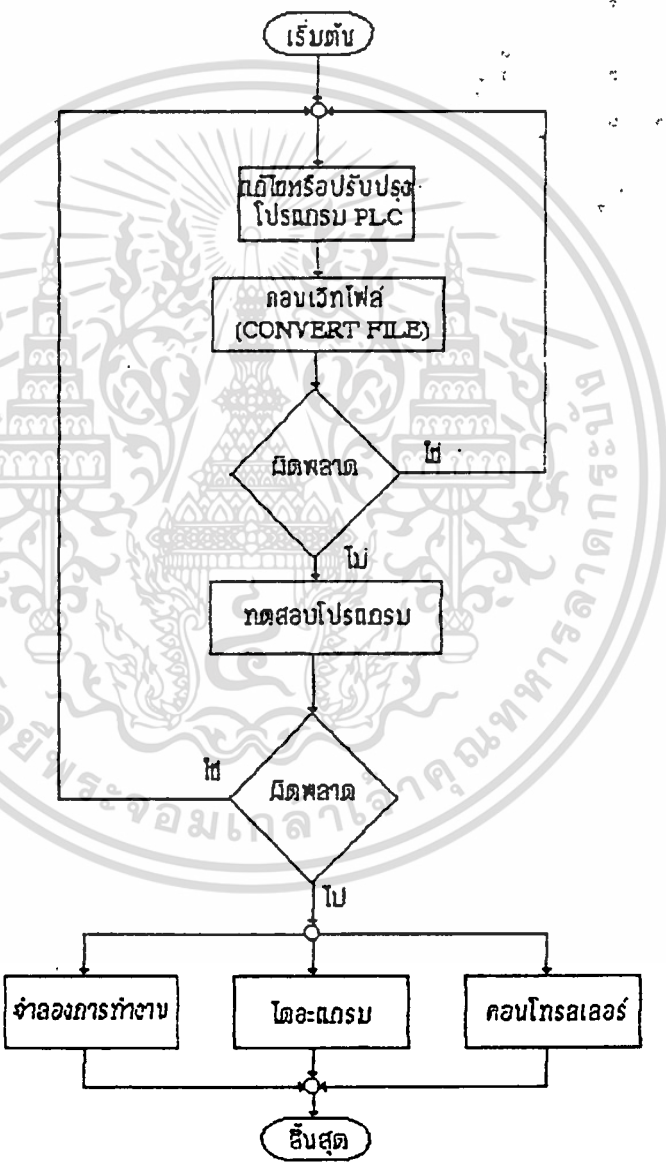

รูปที่ ก.11 แสดง HELP ของโปรแกรม PLCSIM หน้าที่ 2

ชุด CONTROLLER

คุณสมบัติของชุด CONTROLLER

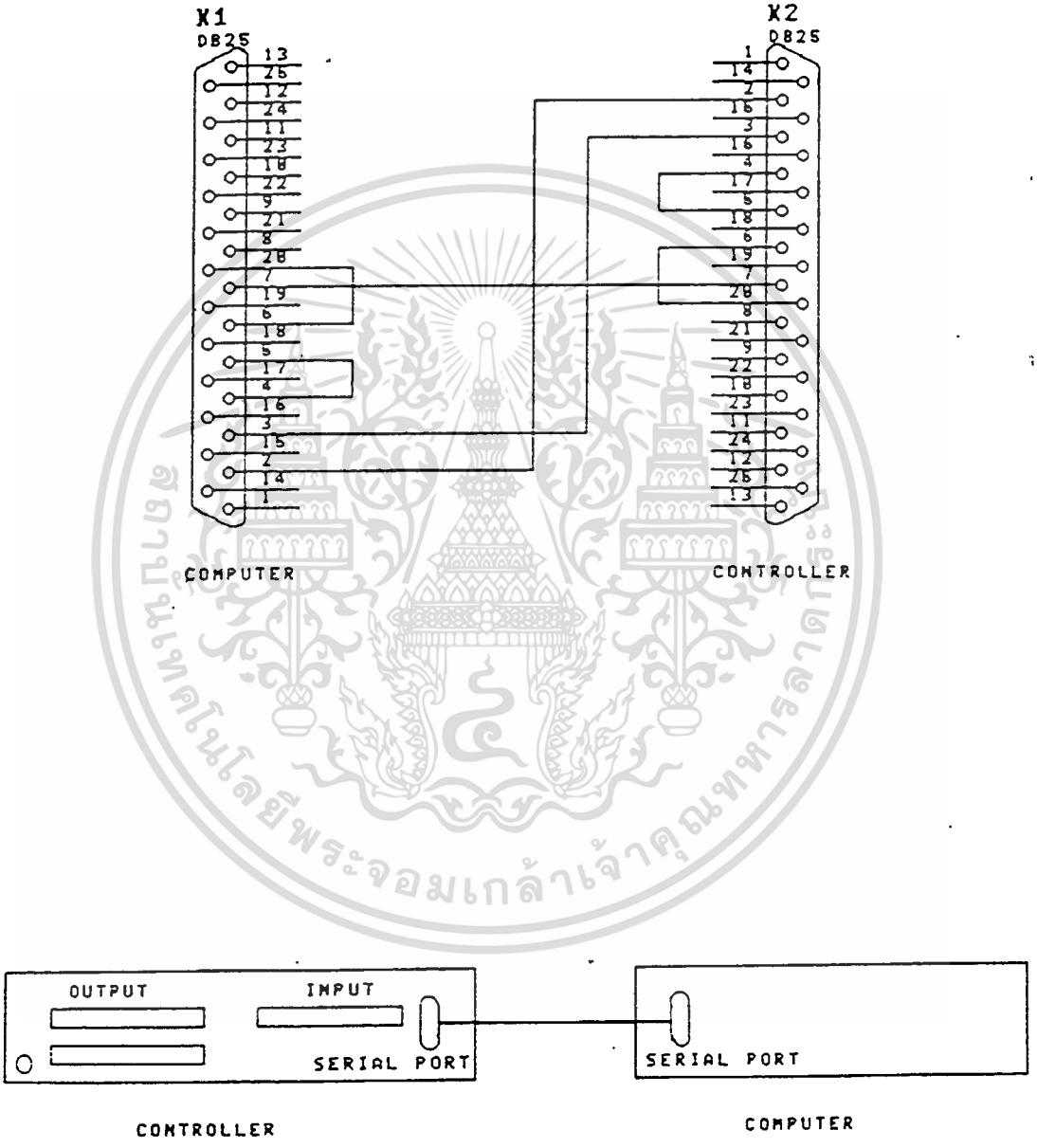
1. การรับส่งข้อมูลแบบอนุกรมใช้ความเร็วในการรับส่งข้อมูล 9600 บิต ต่อ วินาที
2. อินพุทจำนวน 8 ช่อง
3. Slot สำหรับเอาท์พุทจำนวน 8 ช่อง
4. แผ่น Card สำหรับ Output ประกอบด้วย
   4.1 ชุดรีเลย์ 220 VAC, 1 A 4 แผ่น
   4.2 ชุดไตรแอค 220 VAC, 2 A 4 แผ่น
4. แหล่งจ่ายไฟภายใน
   24 VDC,1 A
   5 VDC,3 A

ขั้นตอนการทำงานในการเขียนโปรแกรม PLC



รูปที่ ก.12 แสดง FLOW CHART ของการเขียนโปรแกรม PLC

การติดตั้งชุด Controller เข้ากับเครื่องคอมพิวเตอร์



รูปที่ ก.13 แสดงการต่อสายรับ-ส่งข้อมูลระหว่างชุด CONTROLLER กับเครื่อง COMPUTER

คู่มือการใช้งาน

SPECIFICATION

CPU	Z84C11 PLUS
EPROM	27C256
PORT A	INPUT 8 CHANNEL
PORT B	OUTPUT RELAY 8 CHANNEL
	OUTPUT TRIAC 8 CHANNEL
PORT E	COMMUNICATION
COM PORT	USE FOR RS-232
INPUT SW PORT	USE FOR PACK SW 8 CHANNEL
SUPPLY	5VDC 3 A
	24 VDC 1 A
TERMINAL	220 VAC
	24 VDC
DISPLAY	LED DISPLAY INPUT 8 CHANNEL
	LED DISPLAY OUTPUT 8 CHANNEL
CASE	LAX 19

การเตรียมเครื่องก่อนการใช้งาน

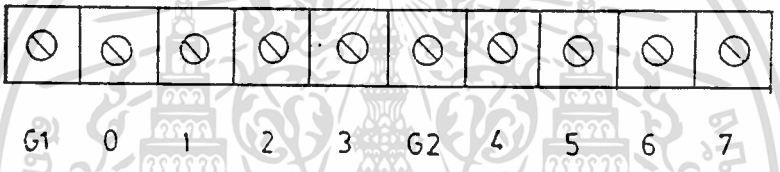
1.ต่อสาย COM PORT RS-232 ระหว่างเครื่องกับคอมพิวเตอร์ในตำแหน่ง COM1 หรือ COM2

2.ต่อสาย 220 V เปิดสวิทช์ POWER

3.เตรียมเครื่องคอมพิวเตอร์โดย RUN PLCSIM

การต่อใช้งาน

INPUT



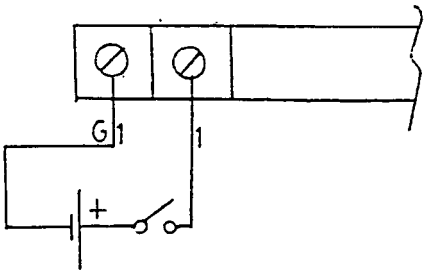
รูปที่ ก.15 แสดงขั้วต่อของ INPUT

ภาพตำแหน่งอินพุท 8 CHANNEL โดยมีระบบกราวด์แยกดังนี้

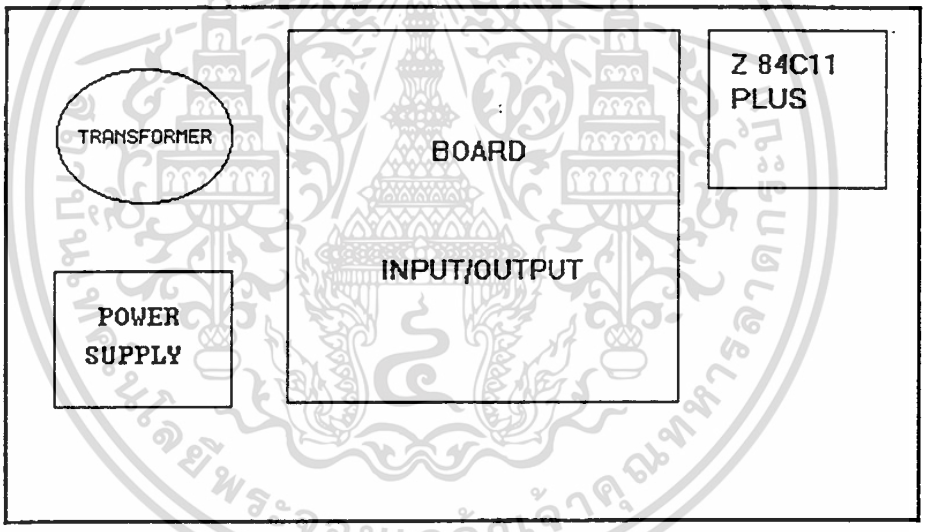
G1 เป็นกราวด์ร่วมของ CHANNEL 0-3

G2 เป็นกราวด์ร่วมของ CHANNEL 4-7

ตัวอย่างการต่อใช้งาน



รูปที่ ก.16 แสดงการต่อ INPUT กับแหล่งจ่ายไฟ

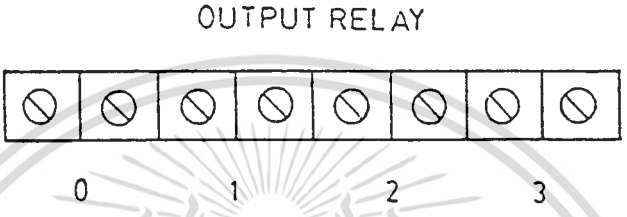


รูปที่ ก.14 แสดงการวางแผ่นวงจรพิมพ์ภาคต่างๆ ของชุด CONTROLLER

ป้อนแรงดัน 24 VDC ให้จะทำให้ได้สภาวะ "1" ให้แก่วงจร

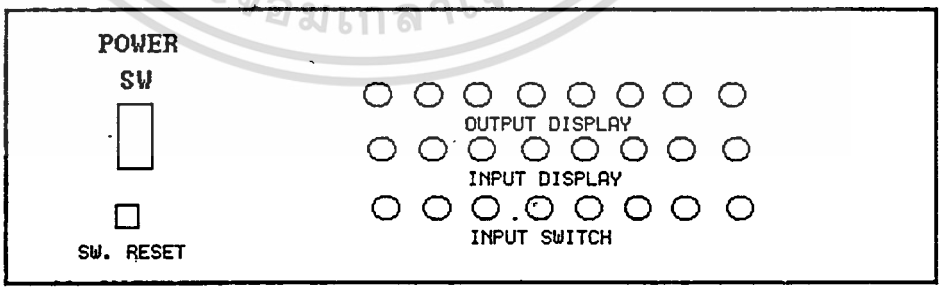
ป้อนแรงดัน 0 VDC ให้จะทำให้ได้สภาวะ "0" ให้แก่วงจร

OUTPUT RELAY

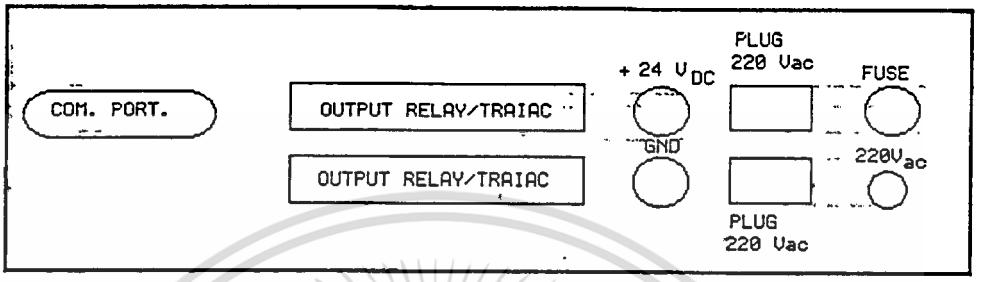


รูปที่ ก.17 แสดงขั้วต่อของ Output Relay

ภาพตำแหน่ง OUTPUT RELAY 4 CHANNEL 0-3 ที่โปรแกรม OUT ที่ 500-503 ใช้เหมือนสวิทช์ใช้กับแรงดันไฟสลับ 220 V กระแส 2 A

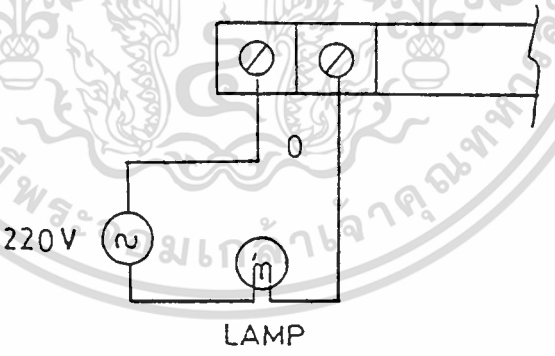


รูปที่ ก.18 แสดง Lay-Out ด้านหน้าของชุด Controller



รูปที่ ก.19 แสดง Lay-Out ด้านหลังของชุด Controller

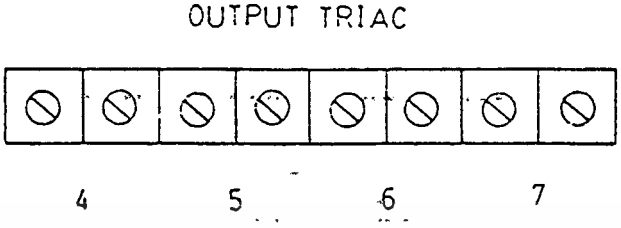
<u>ตัวอย่างการต่อใช้งาน</u>



รูปที่ ก.20 แสดงการต่อ Load ที่ Output Relay

เมื่อ OUT "1" ตำแหน่ง 500-503 ของโปรแกรม จะทำให้หน้าสัมผัสของรีเลย์ปิดวงจร

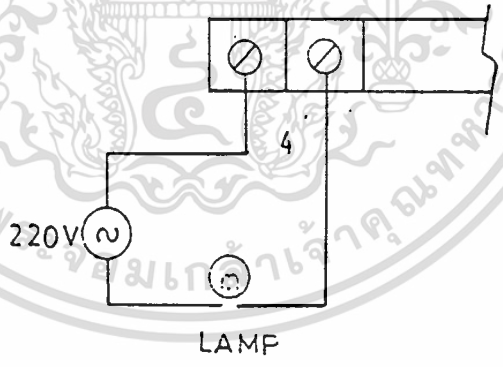
OUTPUT TRIAC



รูปที่ ก.21 ขั้วต่อ Output ของ Triac

ภาพตำแหน่ง OUTPUT TRIAC 4 CHANNEL 4-7 ที่โปรแกรม OUT ที่ 504-507 ใช้ไตรแอคเป็นสวิทช์ ทนแรงดันได้ 220 V 50 W

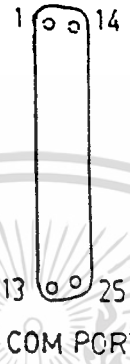
ตัวอย่าง การต่อใช้งาน



รูปที่ ก.22 แสดงการต่อ Loadที่ Output Triac

เมื่อ OUT "1" ตำแหน่ง 504-507 ของโปรแกรมจะทำให้ไตรแอคอยู่ในสภาวะนำกระแสทำให้ LOAD ได้รับแรงดัน 220 V

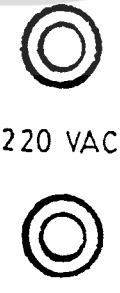
COM PORT



รูปที่ ก.23 แสดงขั้วต่อ RS-232

ใช้เป็น PORT ติดต่อสื่อสารแบบอนุกรม RS-232 ระหว่างเครื่องควบคุม อินพุท/เอาท์พุท กับไมโครคอมพิวเตอร์ ในตำแหน่ง COM1,COM2 โดย SET ในโปรแกรม PLCSIM ได้

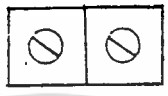
TERMINAL 220 VAC



รูปที่ ก.24 แสดงขั้วต่อ 220 VAC

ใช้เป็นแหล่งจ่ายให้เพื่อสะดวกในการต่อใช้งานร่วมกับ OUTPUT RELAY และ OUTPUT TRIAC

TERMINAL 24 VDC



รูปที่ ก.25 แสดงขั้วต่อ 24 VDC

ใช้เป็นแหล่งจ่ายให้เพื่อสะดวกในการต่อใช้งาน INPUT

ข้อควรระวัง

1.ในการใช้งาน 220 VAC ควรต่อสายให้แน่นไม่หลุดง่ายอาจทำให้เกิดอันตรายได้

2.อย่านำโหลดที่ใช้กำลังไฟฟ้าเกิน 450 W มาใช้กับ OUTPUT TRIAC นอกจากจะเปลี่ยน แผ่นระบายความร้อนขนาดใหญ่ขึ้นแทน โดยจะต้องดูกำลังไฟฟ้าสูงสุดของ TRIAC ก่อนแล้วใช้ LOAD ให้สอดคล้องกัน

ข้อแนะนำและข้อจำกัดของโปรแกรม PLCSIM และ CONTROLLER

โปรแกรมจำลองการทำงานเครื่องควบคุมแบบโปรแกรมได้ มีความสามารถที่จะทำงานได้ตามขอบเขตที่ระบุไว้ แต่ เป็นโครงงานที่ได้เริ่มพัฒนาเป็นครั้งแรกจึงมีข้อบกพร่องบางประการคือ

1.การแสดงผลทางด้านกราฟิค ในส่วนของการดูสภาวะการทำงานของชุด Controller ไม่สามารถที่จะตอบสนอง อินพุท/เอาท์พุท ที่มีการเปลี่ยนแปลงอย่างรวดเร็วได้ เนื่องจากชุด Controller จะส่งข้อมูลมายังโปรแกรมจำลองการทำงานของเครื่องควบคุมแบบโปรแกรมได้ ทุกๆ 1 วินาที

2.การแสดงผลด้วย Timing Diagram ไม่สามารถที่จะจำลองการทำงานในคำสั่ง

Timerได้ เนื่องจากการทำงานของโปรแกรมจำลองการทำงานของเครื่องควบคุมแบบโปรแกรมได้ ใน 1 Loop นั้นเร็วกว่าค่าเวลาของ Timer จึงไม่สามารถที่จะแสดง Timing ของ Timer ได้

3.การส่งข้อมูลไปยังชุด Controller ในกรณีที่ Communication Port ที่ผู้ใช้ตั้งไว้ไม่มีในคอมพิวเตอร์ โปรแกรมจำลองการทำงานของเครื่องควบคุมแบบโปรแกรมได้ จะใช้เวลานานในการที่จะกลับสู่เมนูหลัก หรือโปรแกรม จะหลุดจากการควบคุมของเครื่องคอมพิวเตอร์ เนื่องจากการตรวจสอบของ Hadrware เอง

4.การแสดง Ladder Diagram ในกรณีที่ไม่สามารถที่จะแสดงได้หมดภายใน 1 หน้าจอ ก็จะนำส่วนของ Brance ที่ไม่สามารถแสดงได้หมดไปแสดงยังหน้าถัดไปใน Ladder Diagram ที่มีความซับซ้อนการแสดง Ladder Diagram อาจแสดงได้ไม่สมบูรณ์

5.การแสดง Ladder Diagram สามารถที่แสดง Ladder Diagram ที่ติดต่อกันไม่เกิน 9 เงื่อนไข

6.โปรแกรมนี้พัฒนาบนเครื่อง PC AT ดังนั้นจึงเหมาะสมที่จะใช้งานกับเครื่อง PC AT

7.การเขียนโปรแกรม PLC ด้วยภาษาบูลลีนสามารถที่จะใช้ TEXT EDITOR ตัวอื่นๆ เขียนแล้วสามารถที่จะนำมาจำลองการทำงานที่โปรแกรม PLCSIM ได้

8. การเขียนโปรแกรม PLC จะต้องลงท้ายด้วย End เสมอ

9. การเขียนโปรแกรม PLC สามารถที่จะเขียนด้วยตัวพิมพ์เล็กหรือตัวพิมพ์ใหญ่ก็ได้

ภาคผนวก ข คู่มือบอร์ด Z84C11 PLUS

CP-Z84C11 PLUS

บทนำ

บอร์ด CP-Z84C11 PLUS เป็นบอร์ดที่ใช้ CPU Z84C11 ของบริษัท ZILOG มาเป็น CPU ประจำบอร์ด RUN ที่ SPEED 10 MHz CPU Z84C11 นี้ก็คือ CPU ที่รวบรวมเอาชิพวงจรต่างๆ ของ ZILOG เข้าด้วยกันคือ

Z84C00 เป็น CPU Z80 แบบ CMOS RUN ที่ 10 MHz

Z84C30 เป็น Z80 CTC แบบ CMOS RUN ที่ 10 MHz

CGC เป็น CLOCK GENERATOR CONTROL CIRCUIT

WDT เป็น WATCH DOG TIMER

POWER ON RESETเป็นวงจร RESET CPU เมื่อ VOLT Vcc ต่ำกว่า 2.2 V

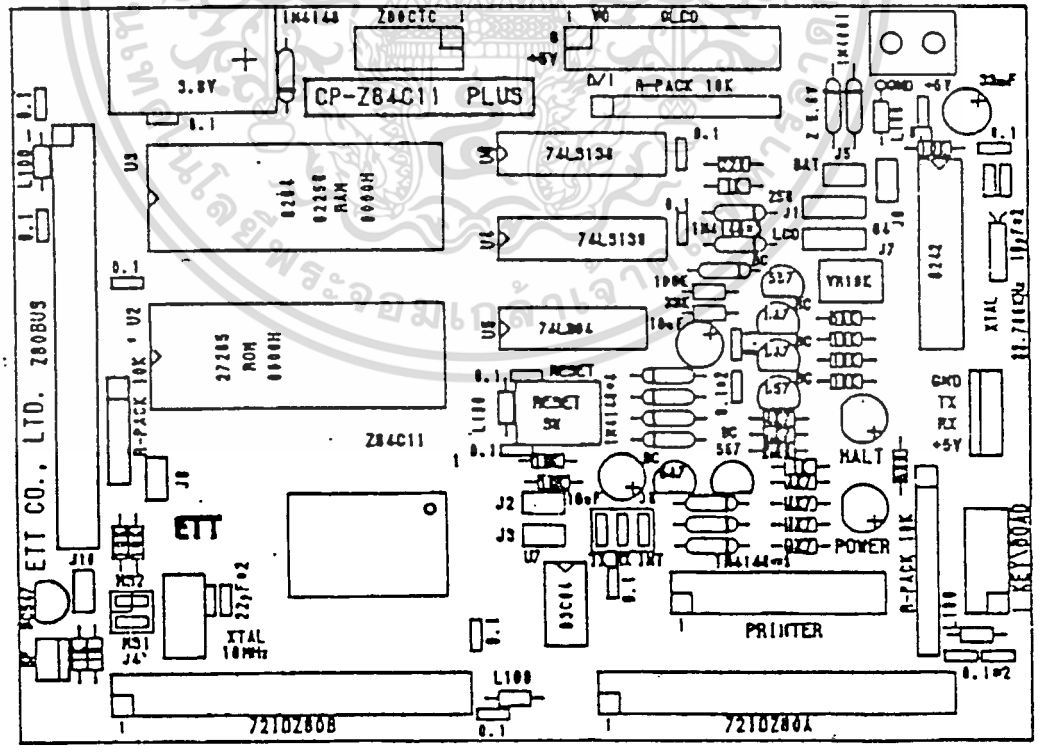
40 BIT PARALLEL PORTเป็น PORT 8 BIT จำนวน 5 PORT ใช้งานจะเห็นว่า CP-Z84C11 PLUS ก็คือเบอร์ที่มี CPU เป็น Z80 พร้อมด้วยวงจรต่างๆเพิ่มเข้ามาเหมาะสำหรับผู้ที่ใช้งาน Z80 ที่ต้องการขีดความสามารถสูงๆขึ้นอีก ประมวลผลได้รวดเร็วขึ้น พร้อมทั้งบอร์ด CP-Z84C11 นี้ยังสามารถต่อกับอุปกรณ์ต่างๆทั่วไปได้อีกเช่น LCD,PRINTER,KEYBORD,EPROM, หรือลำโพงด้วยก็เหมาะเป็นบอร์ดใช้งาน Controller เป็นอย่างยิ่ง

นอกจากนี้บอร์ด CP-Z84C11 PLUS ยังสามารถต่อร่วมกับ ET-DEBUGGER Z84C11 ทำให้สามารถพัฒนาระบบ Z84C11 นี้รวมกับเครื่อง PC ผ่านทาง RS-232 PORT ได้ โดยสามารถสั่ง RUN RUN SINGLE STEP, ดูค่า REGISTER,LOAD FILE จากเครื่อง PC หรือทำ ON LINE ASM,DASM ได้ด้วย และยังมี SYSTEM CALL ให้สามารถเรียกใช้โปรแกรมใน DEBUGGER ได้อีก

ข้อมูลของบอร์ด CP-Z84C11 PLUS

1. CPU

Z84C11 ของบริษัท ZILOG โดยเป็น HIGHT SPEED OPERATION โดยมีให้เลือก 2 รุ่น คือ RUN ที่ความถี่ 6 MHz และ 10 MHz ในบอร์ดนี้เราใช้ CPU รุ่น RUN 10 MHz CPU เบอร์นี้ของ ZILOG จะเป็นแบบ CMOS ทำให้กินกระแสต่ำมากในขณะที่เรา SET ให้อยู่ใน STOP MODE จะกินกระแสเพียง 50 uA ข้อดีอีกอย่างหนึ่งของ CPU ตัวนี้ก็คือ ถึงแม้จะ RUN 10 MHz ก็ตามเราก็สามารถ SET ให้ RUN เพียงครึ่งเดียว ก็ได้คือ 5 MHz ในกรณีที่ใช้ RAM หรือ ROM ที่มี ACESS TIME ต่ำๆ โดยเมื่อ POWER ON แล้ว CPU จะ RUN เพียง 5 MHz เท่านั้นตอนเริ่มต้น และเมื่อเราต้องการ RUN 10 MHz ก็สามารถ Set ได้ด้วยโปรแกรม รูปแบบลักษณะของ CPU จะเป็น PACKAGE แบบ 100 PIN QFP



รูปที่ ข.1 แสดงการวางตำแหน่งอุปกรณ์ของบอร์ด Z84C11 PLUS

2. MEMORY

CP-Z84C11 สามารถใส่หน่วยความจำได้สูงสุด 64 KBYTE โดยแบ่งเป็น

SOCKET 1 EPROM สามารถใส่ EPROM ขนาด 32 KBYTE ได้โดยเป็นเบอร์ 27256 มีหน่วย ความจำเริ่มจาก ADDRESS 0000H-7FFFH

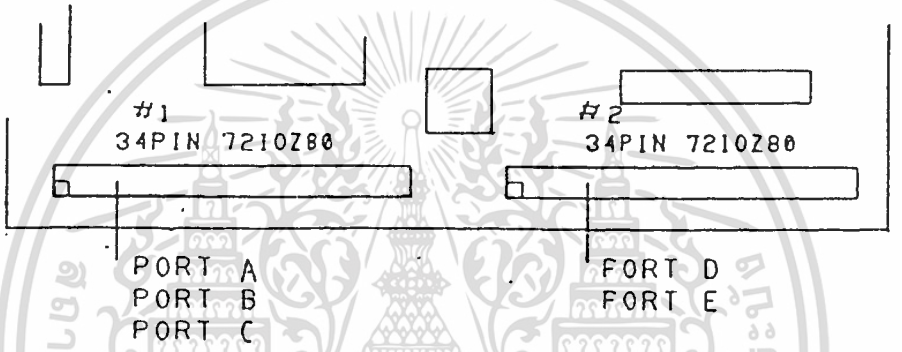
SOCKET 2 RAM สามารถใส่ RAM ขนาด 8-32 KBYTEได้โดยแบ่งเป็นเบอร์ 6264 หรือ 62256 มีหน่วยความจำเริ่มจาก ADDRESS 8000H ถึง FFFFH โดย SET ตำแหน่ง JUMPER ในการกำหนดเบอร์ไอซีดังรูป

รูปที่ ข.2 แสดงการต่อตำแหน่ง Jumper เลือก หน่วยความจำ

หน่วยความจำ RAM ส่วนนี้เราสามารถต่อ BATTERY 3.6 V ใช้ BACKUP ข้อมูลของหน่วยความจำนี้ได้ด้วย

3. PORT

CP-Z84C11 จะมี PORT ใช้งาน 40 BIT I/O หรือ 5 PORT ด้วยกันโดย 5 PORT นี้จะเป็น PORT ในตัว CPU โดยจะมีตำแหน่งต่อออกมาดังรูป



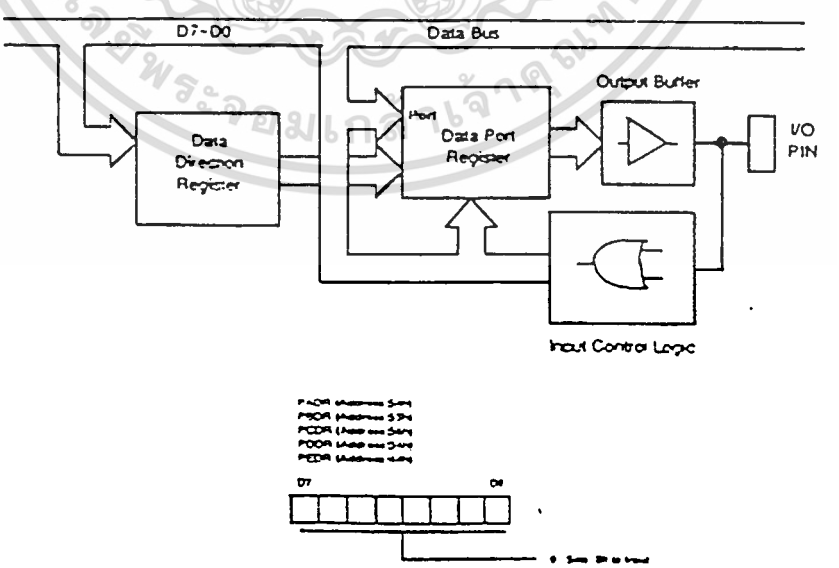
รูปที่ ข.3 แสดงขั้วต่อ PORT A,B,C,D,E ของบอร์ด Z84C11 PLUS

PORT ที่ต่อออกมานี้จะอยู่เป็น CONNECTER 34 PIN (7210Z80) สามารถต่อร่วมใช้กับอุปกรณ์ สนับสนุนต่างๆของ อีทีที ได้มากมาย เช่นชุด ET-SSRAC, ชุด ET-AD เป็นต้น

CHANNEL PORT	ADDRESS PORT
PORT A DATA PORT	50H
PORT B DATA PORT	51H
PORT C DATA PORT	52H
PORT D DATA PORT	53H
PORT E DATA PORT	54H

CHANNEL CONTROL PORT	ADDRESS PORT
PORT A DATA DIRECTION REGISTER	54H
PORT B DATA DIRECTION REGISTER	55H
PORT C DATA DIRECTION REGISTER	56H
PORT D DATA DIRECTION REGISTER	34H
PORT E DATA DIRECTION REGISTER	44H

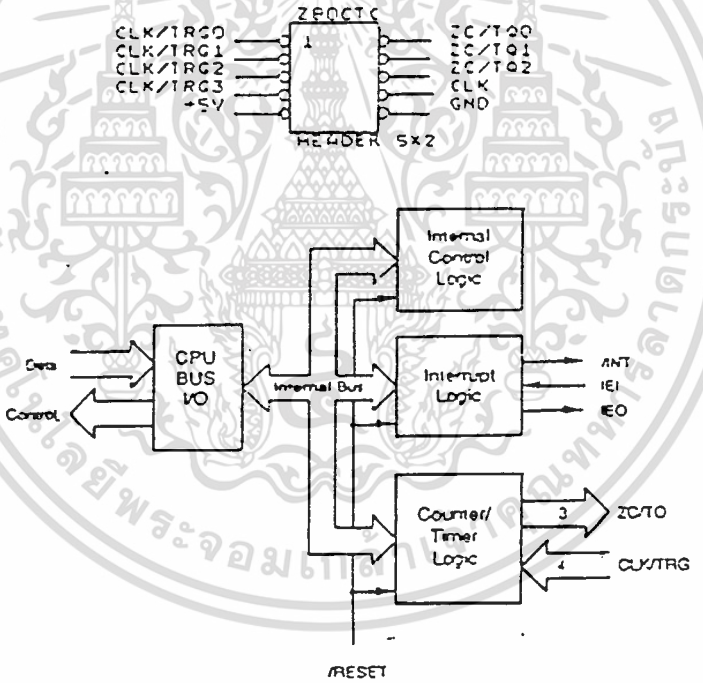
เราสามารถสั่งให้ PORT ของ Z84C11 เป็น INPUT หรือ OUTPUT ได้โดยอิสระ BIT ต่อ BIT โดยถ้าเรา SET ค่าออกที่ PORT DATA DIRECTION REGISTER ถ้าให้ BIT ใดๆ เป็น "1" ก็คือให้ PORT DATA ของ BIT นั้นๆเป็น OUTPUT PORT (ถ้า SET PORT นั้นเป็น OUTPUT: ค่าเริ่มต้นจะเป็นศูนย์) และถ้าเราต้องการให้เป็น INPUT PORT ก็ SET ค่าออกที่ PORT DATA DIRECTION เป็นค่า "0"



รูปที่ ข.4 แสดง Port Direction Register

4. CTC

CP-Z84C11 จะมีวงจร Z84C30 (Z80 CTC) ต่อร่วมกันอยู่ในตัวเรียบร้อยแล้วโดย บอร์ด CP-Z84C11 จะต่อขาใช้งาน ของ CTC ออกมาที่ CONNECTER 10 PIN แล้วดังรูป ส่วนขา INT ของ CTC นั้นจะต่อกับขา INT ของ Z80 ในตัวโดยเป็นแบบ WIRE-OR เรียบร้อยในตัวแล้ว รายละเอียดของ CTC ในบอร์ดนี้จะเหมือนกับ Z80 CTC ทั่วไป



รูปที่ ข.5 แสดง Block Diagram ของ CTC

CTC Address Port

CHANNEL	ADDRESS PORT
CH 0	10 H
CH 1	11 H
CH 2	12 H
CH 3	13 H

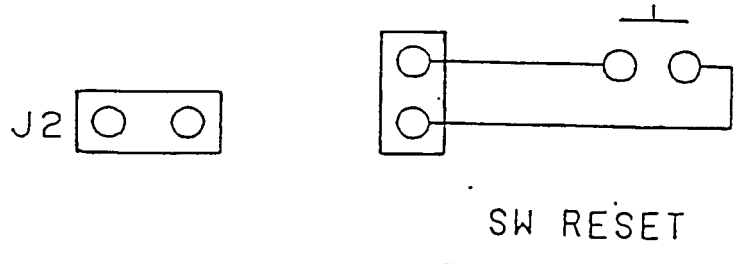
5. MS1,MS2 (J4)

MS1,MS2 จะเป็นขาอินพุท ของ Z84C11 โดยเป็นการ SET ให้ CPU อยู่ในคำสั่ง HALT แล้วจะให้อยู่ในสภาพใด (RUN,IDLE1,IDLE2,STOP)

MS1	MS2	HALT STATE
1	1	RUN MODE
0	0	IDLE1 MODE
0	1	IDLE2 MODE
1	0	STOP MODE

6. RESET (J2)

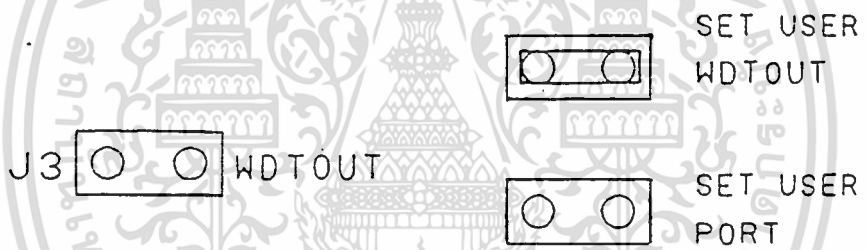
เป็น INPUT ต่อเข้ากับขา RESET CPU Z84C11 ห้ามต่อกับวงจรประเภท R,C RESET



รูปที่ ข.6. แสดงการต่อ Reset Switch

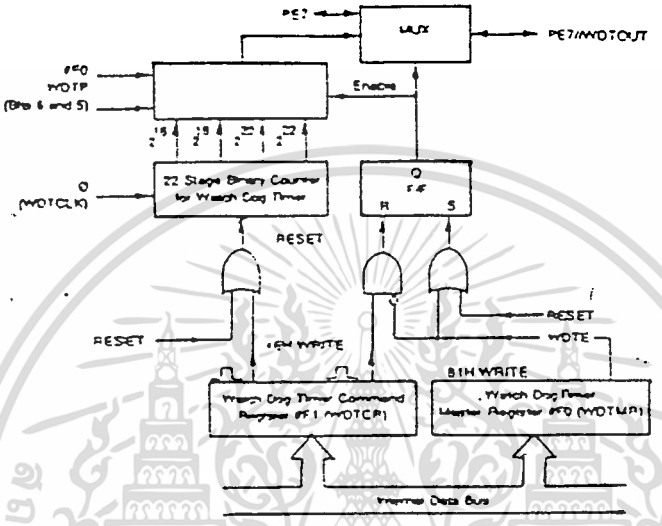
7.WDTOUT

เป็น PIN จาก PORT PE BIT 7 ซึ่งทำหน้าที่ 2 อย่างคือ เป็น DATA PORT ทั่วๆ ไป แล้วก็ยังเป็น WATGH DOG TIMER OUTPUT ด้วยโดยจะเป็น OPEN DRAIN I/O เมื่อถูก SET ให้เป็น WDTOUT โดย PIN นี้จะถูกต่อโดยตรงเข้ากับ PIN PRESET ของ CPU โดยปกติถ้าไม่มีการ SET WDTOUT แล้วให้ถอด JUMPER ออกห้ามต่อถ้าไม่มีการใช้ WDTOUT เพราะเมื่อเราจะใช้ PORT PE 7 จะทำให้ CPU ถูก RESET ตามคำสั่ง OUT PORT PE นั้นด้วย



รูปที่ ข.7 แสดงการต่อ Watch Dog

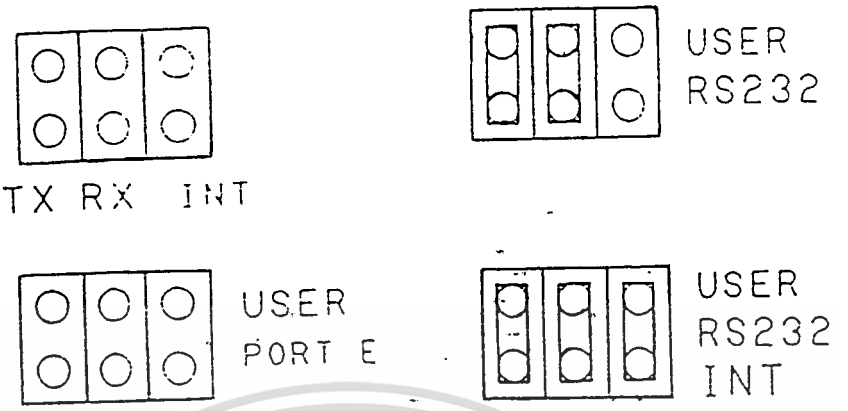
(*WATCH DOG เป็นลักษณะวงจรที่จะทำการ RESET CPU อยู่เสมอตามค่าเวลาที่เรากำหนด ซึ่งถ้าไม่ทำการ DISABLE WATCH DOG ภายในเวลาที่กำหนด CPU นั้นก็จะถูก RESET เช่นในโปรแกรมทำงานปกติเราจะ CALL DISABLE WATCH DOG อยู่เสมอ ต่ถ้า CPU กำลัง RUN อยู่นั้นเกิดมีสัญญาณรบกวนขึ้น ทำให้ไม่อาจสามารถ RUN โปรแกรมปกติที่มีการเรียกใช้โปรแกรม CALL DISABLE WATCH DOG ได้ CPU ก็จะเกิดการ RESET ขึ้นทันทีเพื่อให้กลับไปเริ่มใหม่อีก)



รูปที่ ข.8 แสดง Block diagram Watch Dog Timer

8. TX, RX, INT (J6)

เป็น CONNECTER 6 PIN ที่ต่อจากวงจร RS-232 ที่ใช้เปลี่ยนระดับสัญญาณต่อเข้า PORT PE 6,PE 5, และเข้าขา INTของ Z84C11 โดยถ้าเราไม่ใช้ PORT RS-232 ก็ให้ถอด JUMPER ออก เพื่อจะใช้ PE 6,PE 5 ได้อย่างอิสระ และอีกส่วนหนึ่งคือ JUMPER ของ INT นั้น เราสามารถ SET ให้ PORT RS-232 รับข้อมูลในรูปของขบวนการ INTERRUPT เช่นเดียวกับการ INTERRUPT จาก RS-232 PORTได้



รูปที่ ข.9 แสดงการต่อการรับ-ส่งข้อมูลของบอร์ด Z84C11 PLUS

9.BAT ON/OFF (J5)

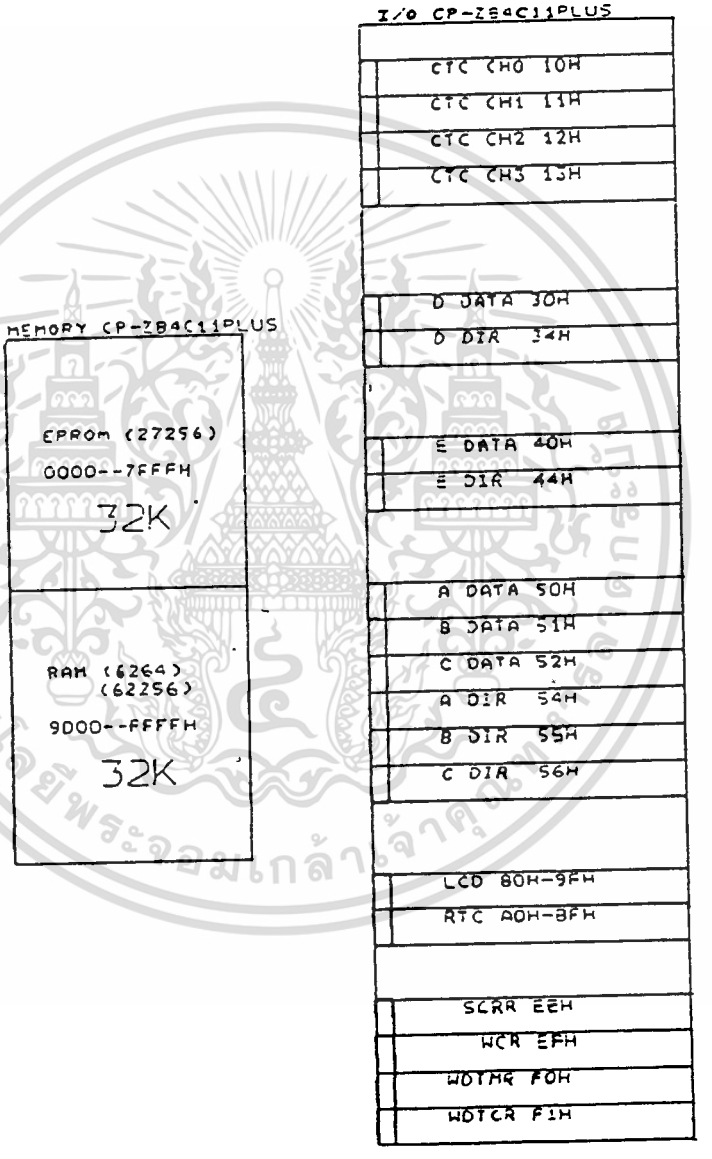
เป็น CONNECTER 2 PIN โดยเราใช้ JUMPER ในการ ปิด/เปิด ไฟ จาก BATTERY 3.6 V ถ้ามีการต่อใช้งาน BATTERY

10. 64,256 (J1)

เป็น CONNECTER 3 PIN ใช้ JUM เลือกว่าหน่วยความจำ RAM ของเราจะใช้หน่วยความจำไอซีเบอร์ 6264 (8 KBYTE) หรือ 62256 (32 KBYTE)

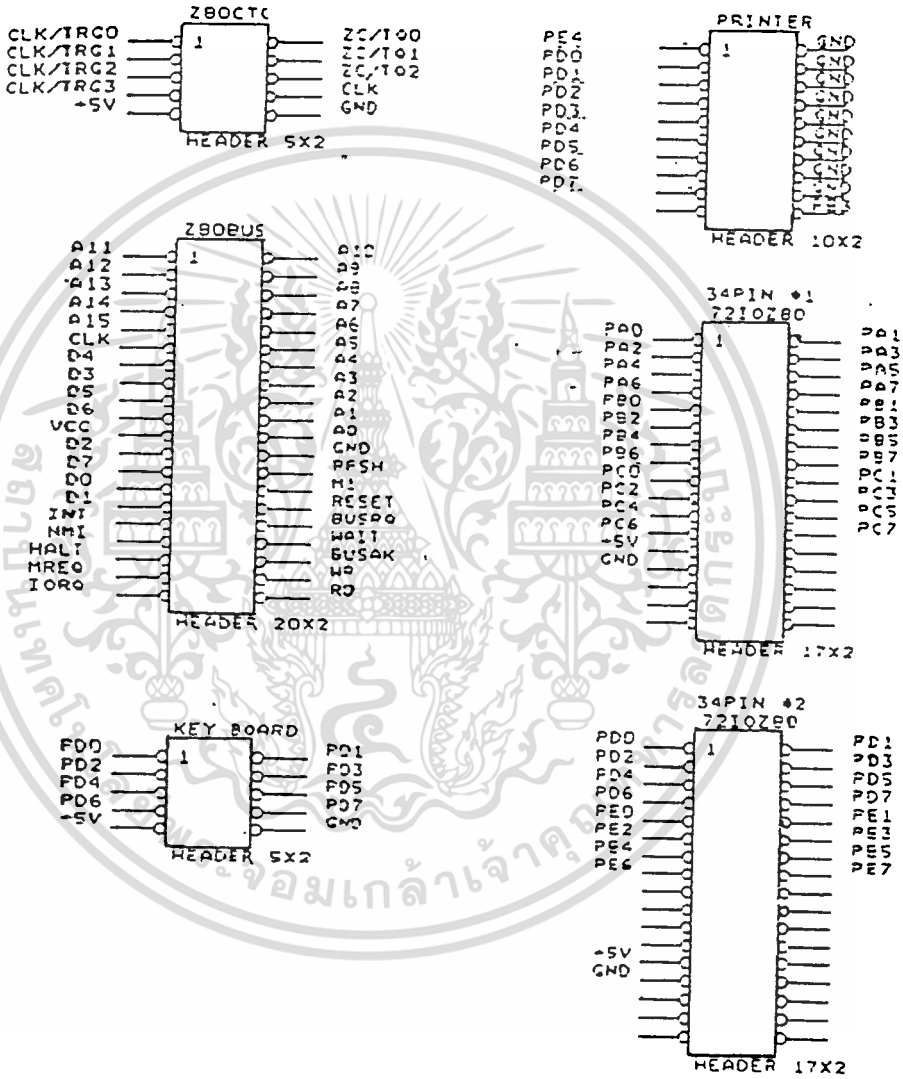
11. MEMORY MAP ADDRESS

CP-Z84C11 จัด MEMORY ออกเป็น 2 ส่วน คือ



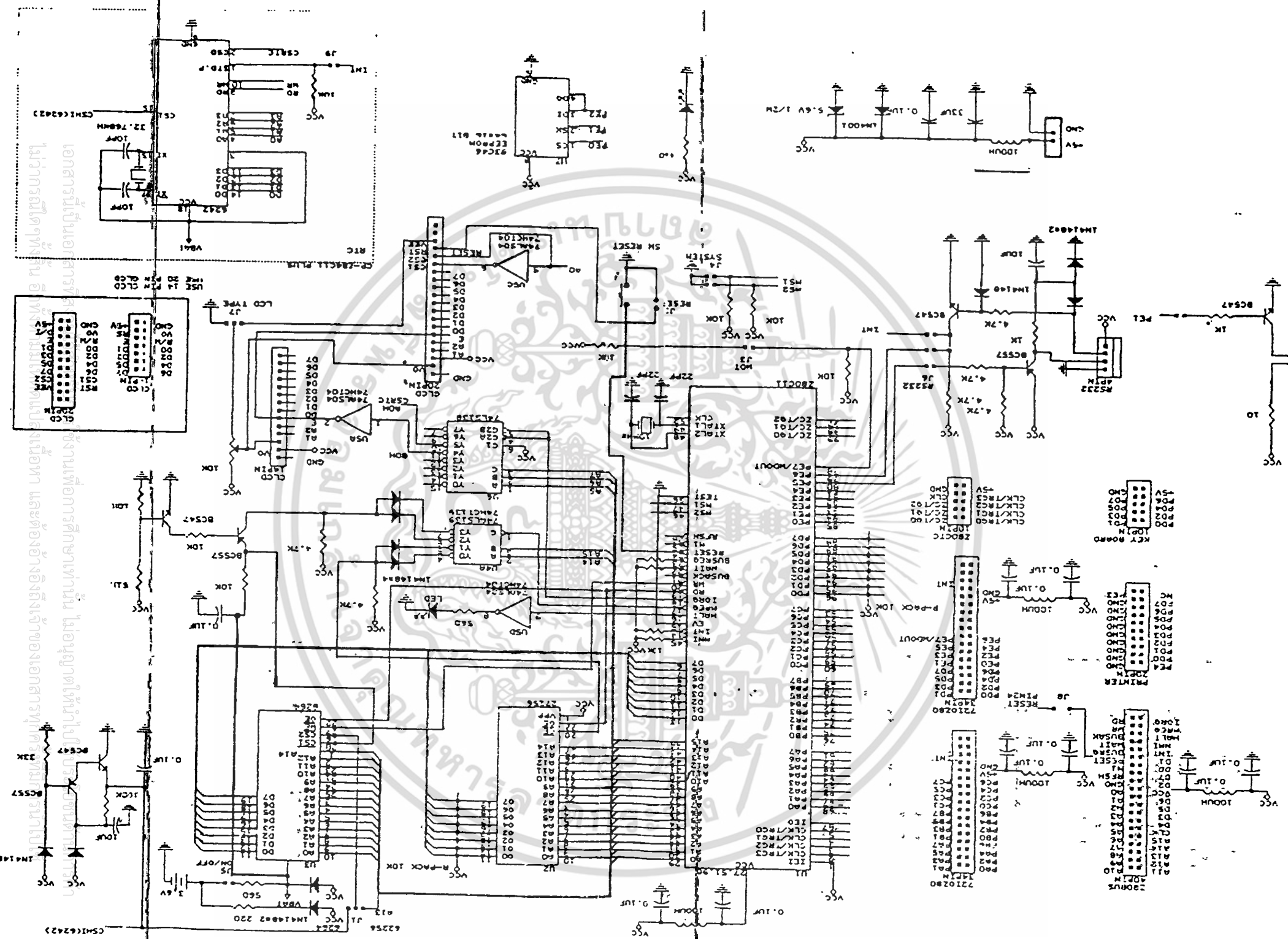
รูปที่ ข.10 แสดงการจัดหน่วยความจำของบอร์ด Z84C11 PLUS

รายละเอียดขา CONNECTER



รูปที่ ข.11 แสดงขั้วต่อต่างๆของบอร์ด Z84C11 PLUS

รูปที่ 4.12 วงจรบอร์ด Z84C11 PLUS



SPECIFICATION

CPU	:	ZILOG Z84C11-10 CMOS TYPE
MEMORY	:	27256 (32 KBYTE)
	:	6264/62256 (32 KBYTE)
RCT PORT	:	6242B
PORT	:	PA,PB,PC,PD,PE I20 PORT 40 BIT
PRINTER PORT	:	1 PORT
SERIAL PORT	:	10 CHANNEL RS-232
LCD PORT	:	1 LCD MODULE (DOT OR GRAPH)
KEYBOARD	:	1 PORT (8 BIT)
CTC PORT	:	4 CHANNEL COUNTER TIMERCONTROLLER (Z80CTC)
CLOCK RATE	:	10 MHz
POWER SUPPLY	:	CONSUMPTION 5 VDC & TERMINAL 5 VDC
CONNECTER	:	1 40PIN EXPANSION HEADER-STRIP (Z80BUS)
	:	2 34PIN EXPANSION HEADER STRIP (72IOZ80 ETT)
	:	1 20PIN EXPANSION HEADER STRIP (PRINTER PORT)
	:	1 20PIN EXPANSION HEADER STRIP (LCD PORT)
	:	1 10PIN EXPANSION HEADER STRIP (Z80CTC)
	:	1 10PIN EXPANSION HEADER STRIP (KEYBOARD)
	:	1 4 PIN EXPANSION (RS-232)
	:	1 2 PIN EXPANSION (WATCH DOG)
	:	1 2 PIN JUMPER (RESET SWITCH)
	:	1 6 PIN JUMPER (TX,RX,INT)
	:	1 4 PIN JUMPER (MS1,MS2)
	:	1 2 PIN JUMPER (ON/OFF BAT)
	:	1 3 PIN JUMPER (6264/62256)
	:	1 3 PIN JUMPER (DOG/GRAPHIC)
	:	1 2 PIN IUMPER (RESET 40 PIN Z80BUS)
	:	1 2 PIN JUMPER (SP)

LED : 1 POWER RED LED

: 1 HALT GREEN LED

PCB SIZE : 12.2 X 9 CM

รายการอุปกรณ์ CP-Z84C11 PLUS

1. CPU Z84C11	1
2. RAM 6264	1
3. TTL 74LS04	1
4. TTL 74LS138	1
5. TTL 74LS135	1
6. TR BC547	3
7. TR BC557	4
8. DIODE 1N4001	1
9. DIODE 1N4148	11
10. DIODE ZENER 5.6 V 1 W	1
11. X TAL 10 MHz	1
12. LED สีแดงกลม	1
13. LED สีเหลืองกลม	1
14. C 0.1 uF MULTILAYER	1
15. C 10 uF 16 V	2
16. C 33 uF 16 V	1
17. C 22 pF 50 V	2
18. R PACK 10K 9 PIN	2
19. R PACK 10K 9 PIN	1
20. L 100uH	4
21. MINI SW	1
22. CONNECTER 4 PIN MALE	1
23. TERMINAL 2 PIN	1
24. JUMPER 2 PIN	11
25. SOCKET 28 PIN	2
26. SOCKET 16 PIN	2
27. SOCKET 14 PIN	1
28. SOCKET 8 PIN	1

29. PCB CP-Z84C11 PLUS	1
30. MANUAL CP-Z84C11	1
31. SOCKET 20 PIN	1
32. C 10 pF	2
33. X'TAL 32.678 KHz	1
34. BAT 3.6 V	1
35. VR 10K	1

R 1/4 Watt 5%

1. 220	1
2. 560	2
3. 1K	3
4. 4.7K	6
5. 10K	7
6. 33K	1
7. 51K	1
8. 100K	1
9. 10	1

HEADER CONNECTER

1. 40 PIN	1
2. 34 PIN	2
3. 20 PIN	2
4. 10 PIN	2
5. 6 PIN	1
6. 4 PIN	1
7. 2 PIN	2

ภาคผนวก ค

OPTO ISOLATER เบอร์ 4N26

6-Pin DIP Optoisolators
Triac Driver Output

These devices consist of gallium arsenide infrared emitting diodes optically coupled to a monolithic silicon detector performing the function of a Zero Voltage Crossing bilateral triac driver.

They are designed for use with a triac in the interface of logic systems to equipment powered from 240 Vac lines, such as solid-state relays, industrial controls, motors, solenoids and consumer appliances, etc.

- Simplifies Logic Control of 240 Vac Power
- Zero Voltage Crossing
- High Breakdown Voltage: V_{DRM} = 400 V Min
- High Isolation Voltage: V_{ISO} = 7500 V Guaranteed
- Small, Economical, 6-Pin DIP Package
- dv/dt of 2000 V/μs Typ, 1000 V μs Guaranteed
- UL Recognized, File No. E54915
- VDE approved per standard 0883 6.80 (Certificate number 41853), with additional approval to DIN IEC380 VDE0806, IEC435 VDE0805, IEC65 VDE0860, VDE0110b, covering all other standards with equal or less stringent requirements, including IEC204 VDE0113, VDE0160, VDE0832, VDE0833, etc.
- Special lead form available (add suffix "T" to part number) which satisfies VDE0883 6.80 requirement for 8 mm minimum creepage distance between input and output solder pads.
- Various lead form options available. Consult "Optoisolator Lead Form Options" data sheet for details.

MAXIMUM RATINGS (T_A = 25°C unless otherwise noted)

Rating	Symbol	Value	Unit
INFRARED EMITTING DIODE			
Reverse Voltage	V_R	6	Volts
Forward Current — Continuous	I_F	60	mA
Total Power Dissipation @ T_A = 25°C Negligible Power in Output Driver Derate above 25°C	P_D	120 1.41	mW mW/°C
OUTPUT DRIVER			
Off-State Output Terminal Voltage	V_{DRM}	400	Volts
Peak Repetitive Surge Current (PW = 100 μs, 120 pps)	I_{TSM}	1	A
Total Power Dissipation @ T_A = 25°C Derate above 25°C	P_D	150 1.76	mW mW/°C
TOTAL DEVICE			
Isolation Surge Voltage (1) (Peak ac Voltage, 60 Hz, 1 Second Duration)	V_{ISO}	7500	Vac
Total Power Dissipation @ T_A = 25°C Derate above 25°C	P_D	250 2.94	mW mW/°C
Junction Temperature Range	T_J	−40 to +100	°C
Ambient Operating Temperature Range	T_A	−40 to +85	°C
Storage Temperature Range	T_{stg}	−40 to +150	°C
Soldering Temperature (10 s)	—	260	°C

(1) Isolation surge voltage, V_{ISO}, is an internal device dielectric breakdown rating. For this test, Pins 1 and 2 are common, and Pins 4, 5 and 6 are common.

MOTOROLA
SEMICONDUCTOR
TECHNICAL DATA

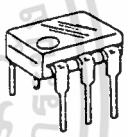
6-Pin DIP Optoisolators
Transistor Output

4N25
4N25A
4N26
4N27
4N28

6-PIN DIP
OPTOISOLATORS
TRANSISTOR OUTPUT

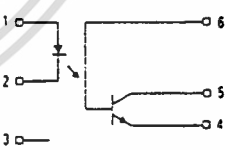
These devices consist of a gallium arsenide infrared emitting diode optically coupled to a monolithic silicon phototransistor detector.

- Convenient Plastic Dual-in-Line Package
- Most Economical Optoisolator
- High Input-Output Isolation Guaranteed — 7500 Volts Peak
- Meets or Exceeds All JEDEC Registered Specifications
- UL Recognized. File Number E54915
- VDE approved per standard 0883.6.80 (Certificate number 41853), with additional approval to DIN IEC380/VDE0806, IEC435/VDE0805, IEC55/VDE0860, VDE110b, covering all other standards with equal or less stringent requirements, including IEC204/VDE0113, VDE0160, VDE0832, VDE0833, etc.
- Special lead form available (add suffix "T" to part number) which satisfies VDE0883/6.80 requirement for 8 mm minimum creepage distance between input and output solder pads.
- Various lead form options available. Consult "Optoisolator Lead Form Options" data sheet for details.



CASE 730A-02
PLASTIC

SCHEMATIC



1. LED ANODE
2. LED CATHODE
3. N.C.
4. EMITTER
5. COLLECTOR
6. BASE

MAXIMUM RATINGS (T_A = 25°C unless otherwise noted)

Rating	Symbol	Value	Unit
INPUT LED			
Reverse Voltage	V_R	3	Volts
Forward Current — Continuous	I_F	60	mA
LED Power Dissipation @ T_A = 25°C with Negligible Power in Output Detector	P_D	120	mW
Derate above 25°C		1.41	mW/°C
OUTPUT TRANSISTOR			
Collector-Emitter Voltage	V_{CEO}	30	Volts
Emitter-Collector Voltage	V_{ECO}	7	Volts
Collector-Base Voltage	V_{CBO}	70	Volts
Collector Current — Continuous	I_C	150	mA
Detector Power Dissipation @ T_A = 25°C with Negligible Power in Input LED	P_D	150	mW
Derate above 25°C		1.76	mW/°C
TOTAL DEVICE			
Isolation Surge Voltage (1) (Peak ac Voltage, 60 Hz, 1 sec Duration)	V_{ISO}	7500	Vac
Total Device Power Dissipation @ T_A = 25°C	P_D	250	mW
Derate above 25°C		2.94	mW/°C
Ambient Operating Temperature Range	T_A	−55 to +100	°C
Storage Temperature Range	T_{stg}	−55 to +150	°C
Soldering Temperature (10 sec, 1/16" from case)	T_{sol}	260	°C

(1) Isolation surge voltage is an internal device dielectric breakdown rating. For this test, Pins 1 and 2 are common, and Pins 4, 5 and 6 are common.

4N25, 4N25A, 4N26, 4N27, 4N28

ELECTRICAL CHARACTERISTICS (T_A = 25°C unless otherwise noted)

Characteristic		Symbol	Min	Typ	Max	Unit
INPUT LED						
Forward Voltage (I_F = 10 mA)	T_A = 25°C	V_F	—	1.15	1.5	Volts
	T_A = −55°C		—	1.3	—	
	T_A = 100°C		—	1.05	—	
Reverse Leakage Current (V_R = 3 V)		I_R	—	—	100	μA
Capacitance (V = 0 V, f = 1 MHz)		C_J	—	18	—	pF
OUTPUT TRANSISTOR						
Collector-Emitter Dark Current (V_{CE} = 10 V, T_A = 25°C	4N25,25A,26,27	I_{CEO}	—	1	50	nA
	4N28		—	1	100	
(V_{CE} = 10 V, T_A = 100°C)	All Devices	I_{CEO}	—	1	—	μA
Collector-Base Dark Current (V_{CB} = 10 V)		I_{CBO}	—	0.2	—	nA
Collector-Emitter Breakdown Voltage (I_C = 1 mA)		$V_{(BR)CEO}$	30	45	—	Volts
Collector-Base Breakdown Voltage (I_C = 100 μA)		$V_{(BR)CBO}$	70	100	—	Volts
Emitter-Collector Breakdown Voltage (I_E = 100 μA)		$V_{(BR)ECO}$	7	7.8	—	Volts
DC Current Gain (I_C = 2 mA, V_{CE} = 5 V)		h_{FE}	—	500	—	—
Collector-Emitter Capacitance (f = 1 MHz, V_{CE} = 0)		C_{CE}	—	7	—	pF
Collector-Base Capacitance (f = 1 MHz, V_{CB} = 0)		C_{CB}	—	19	—	pF
Emitter-Base Capacitance (f = 1 MHz, V_{EB} = 0)		C_{EB}	—	9	—	pF
COUPLED						
Output Collector Current (I_F = 10 mA, V_{CE} = 10 V)	4N25,25A,26	I_C	2	7	—	mA
	4N27,28		1	5	—	
Collector-Emitter Saturation Voltage (I_C = 2 mA, I_F = 50 mA)		$V_{CE(sat)}$	—	0.15	0.5	Volts
Turn-On Time (I_F = 10 mA, V_{CC} = 10 V, R_L = 100 Ω)		t_{on}	—	2.8	—	μs
Turn-Off Time (I_F = 10 mA, V_{CC} = 10 V, R_L = 100 Ω)		t_{off}	—	4.5	—	μs
Rise Time (I_F = 10 mA, V_{CC} = 10 V, R_L = 100 Ω)		t_r	—	1.2	—	μs
Fall Time (I_F = 10 mA, V_{CC} = 10 V, R_L = 100 Ω)		t_f	—	1.3	—	μs
Isolation Voltage (f = 60 Hz, t = 1 sec)		V_{ISO}	7500	—	—	Vac(pk)
Isolation Resistance (V = 500 V)		R_{ISO}	10^{11}	—	—	Ω
Isolation Capacitance (V = 0 V, f = 1 MHz)		C_{ISO}	—	0.2	—	pF

TYPICAL CHARACTERISTICS

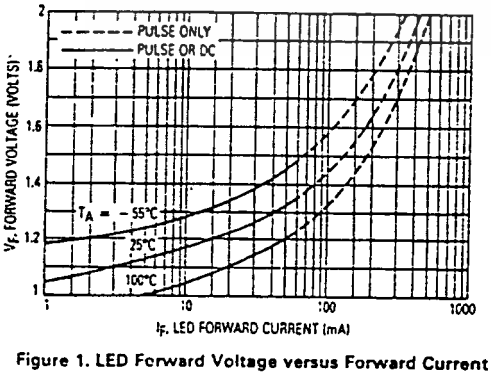


Figure 1. LED Forward Voltage versus Forward Current

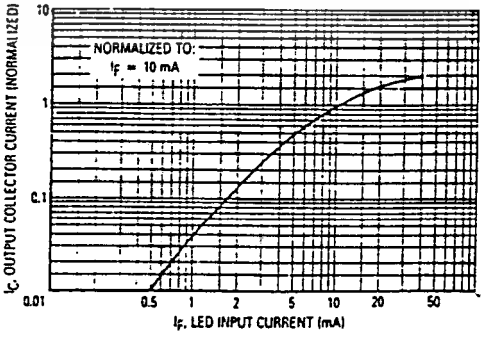


Figure 2. Output Current versus Input Current

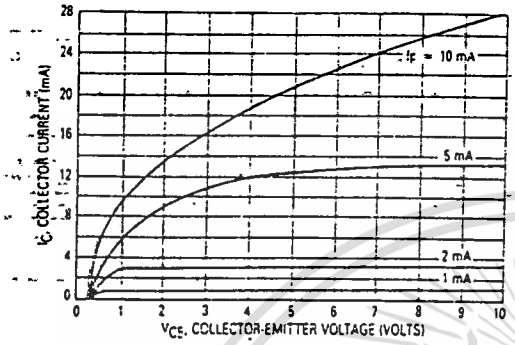


Figure 3. Collector Current versus Collector-Emitter Voltage

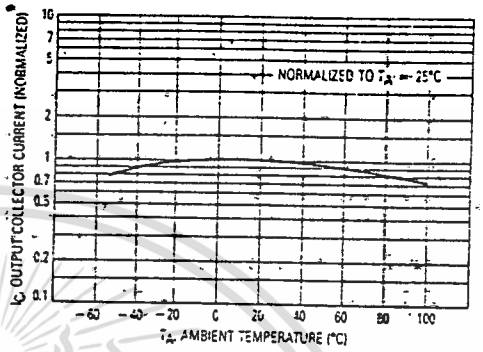


Figure 4. Output Current versus Ambient Temperature

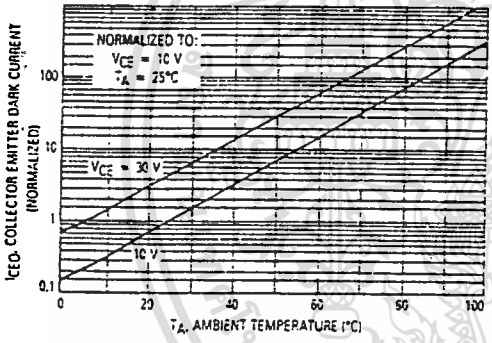


Figure 5. Dark Current versus Ambient Temperature

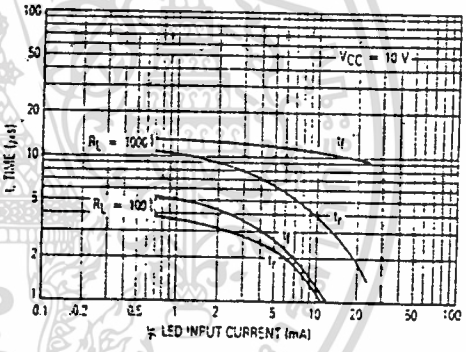


Figure 6. Rise and Fall Times

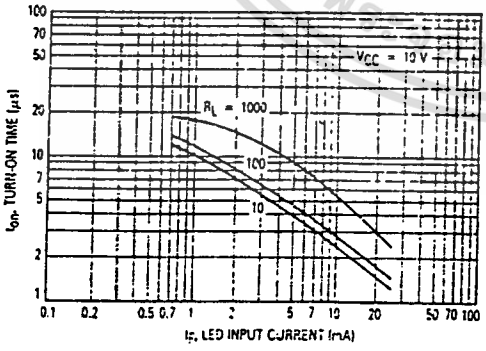


Figure 7. Turn-On Switching Times

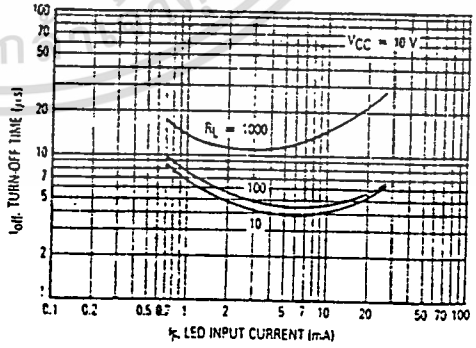


Figure 8. Turn-Off Switching Times

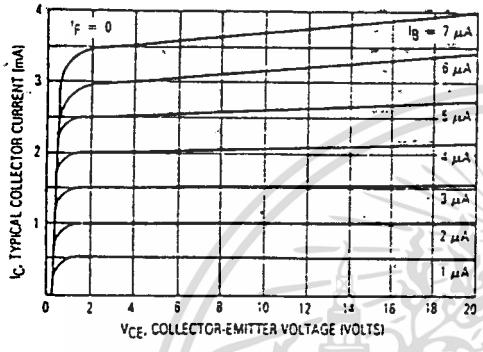


Figure 9. DC Current Gain (Detector Only)

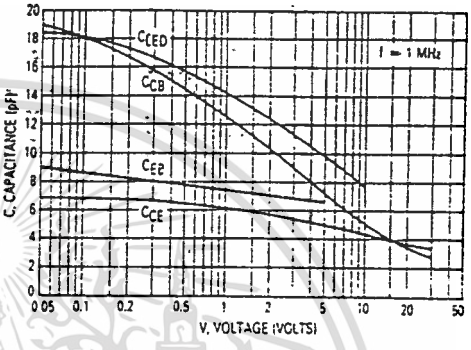
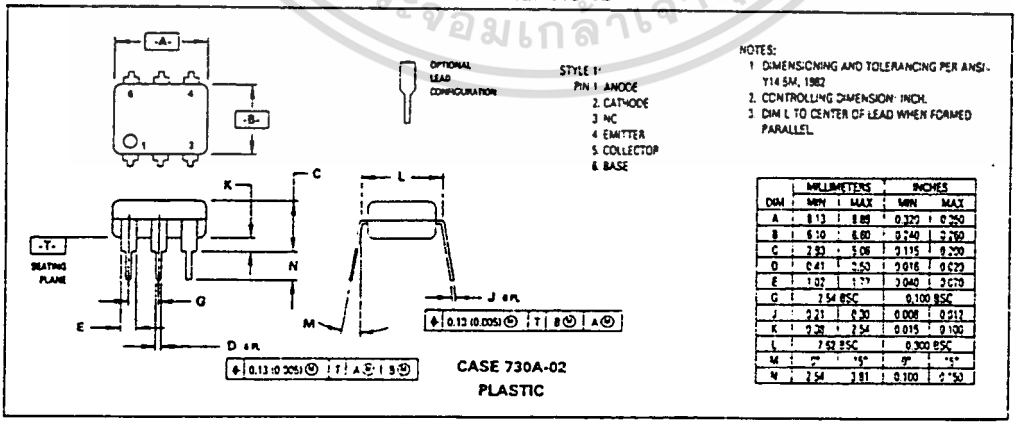


Figure 10. Capacitances versus Voltage

Figure 11. Switching Times

OUTLINE DIMENSIONS



DIM	MILLIMETERS MIN	MILLIMETERS MAX	INCHES MIN	INCHES MAX
A	8.13	8.89	0.320	0.350
B	6.10	6.60	0.240	0.260
C	2.93	5.08	0.115	0.200
D	0.41	0.55	0.016	0.022
E	1.02	1.77	0.040	0.070
G	2.54 BSC		0.100 BSC	
J	0.21	0.30	0.008	0.012
K	0.38	2.54	0.015	0.100
L	7.62 BSC		0.300 BSC	
M	0°	15°	0°	15°
N	2.54	3.81	0.100	0.150

ภาคผนวก ง

ASSEMBLY Z80 PROGRAM และการคำนวณหาค่าเวลา(TIME STATES) ของโปรแกรมย่อย(SUBRUTINE)

ASSEMBLY Z80 PROGRAM

```
OUT_STATUS      EQU     8000H
SW_STATUS       EQU     OUT_STATUS+20
TR_STATUS       EQU     SW_STATUS+20
CNT_STATUS      EQU     TR_STATUS+20
TIM_STATUS      EQU     CNT_STATUS+20
FLAG_OUT        EQU     TIM_STATUS+20
COUNT_STORE     EQU     FLAG_OUT+5
TIME_STORE      EQU     COUNT_STORE+40
FLAG_TIM        EQU     TIME_STORE+40
START_FLAG      EQU     FLAG_TIM+20
STORE_INPUT     EQU     START_FLAG+5
ADDRESS_PRO     EQU     8200H
STACK_REGA      EQU     9500H
RTC             EQU     0A0H
CREG_E          EQU     RTC+0EH
CREG_F          EQU     RTC+0FH
```

START OF PROGRAM

```
                ORG     0000H
                XOR     A
DELAY_START:    DEC     A
                NOP
                JR      NZ,DELAY_START
                JP      INITIZAL
```

```
                LD      A,(START_FLAG)
                CP      01H
                JR      Z,RUN
                LD      HL,ADDRESS_PRO
BIT6:           IN      A,(40H)
                BIT     6,A
                JR      NZ,BIT6
                CALL    RX_LINE
RUN:            IM      1
                EI
```

MAIN FUNCTION TO CALL SUBRUTINE
OF PROGRAM PLC SIMULATOR

```
BEGIN:          LD      HL,FLAG_OUT
                LD      (HL),00H
START:          LD      HL,STACK_REGA
                LD      DE,ADDRESS_PRO
START1:         LD      A,(DE)
                CALL    WATCH_DOG
                CP      21H
                JP      NC,INITIZAL1
                CP      00H
                JP      Z,CALL_END
                CP      01H
                JP      Z,CALL_LD_SW
                CP      02H
                JP      Z,CALL_LD_OUT
                CP      03H
```

INT MODE 1

```
        ORG     0038H
        DI
        CALL    DEC_TIME
        CALL    SEND_SW
        NOP
        EI
        RETI
```

MAIN PROGRAM

```
        ORG     0100H
INITIZAL:
        LD      SP,9EAOH
        LD      A,00H
        OUT     (54H),A
        LD      A,0FFH
        OUT     (55H),A
        LD      A,20H
        OUT     (44H),A
        LD      A,4EH
        OUT     (0F1H),A
        LD      A,0B1H
        OUT     (0F1H),A
        LD      A,67H
        OUT     (0F0H),A
        DI
        LD      A,01H
        OUT     (CREG_F),A
        LD      A,04H
        OUT     (CREG_F),A
        OUT     (CREG_E),A
```

```
JP      Z,CALL_LD_TR
CP      04H
JP      Z,CALL_LD_CNT
CP      05H
JP      Z,CALL_LD_TIM
CP      06H
JP      Z,CALL_LDN_SW
CP      07H
JP      Z,CALL_LDN_OUT
CP      08H
JP      Z,CALL_LDN_CNT
CP      09H
JP      Z,CALL_LDN_TIM
CP      0AH
JP      Z,CALL_AND_SW
CP      0BH
JP      Z,CALL_AND_OUT
CP      0CH
JP      Z,CALL_AND_CNT
CP      0DH
JP      Z,CALL_AND_TIM
CP      0EH
JP      Z,CALL_ANDN_SW
CP      0FH
JP      Z,CALL_ANDN_OUT
CP      10H
JP      Z,CALL_ANDN_CNT
CP      11H
JP      Z,CALL_ANDN_TIM
```

```
CP      12H
JP      Z,CALL_OR_SW
CP      13H
JP      Z,CALL_OR_OUT
CP      14H
JP      Z,CALL_OR_CNT
CP      15H
JP      Z,CALL_OR_TIM
CP      16H
JP      Z,CALL_ORN_SW
CP      17H
JP      Z,CALL_ORN_OUT
CP      18H
JP      Z,CALL_ORN_CNT
CP      19H
JP      Z,CALL_ORN_TIM
CP      1AH
JP      Z,CALL_AND_LD
CP      1BH
JP      Z,CALL_OR_LD
CP      1CH
JP      Z,CALL_CNT
CP      1DH
JP      Z,CALL_TIM
CP      1EH
JP      Z,CALL_OUT
CP      1FH
JP      Z,CALL_OUT_TR
CP      20H
```

```
                                                        T states
                    JP      CALL_OUTN
INITIZAL1:          LD      A,00H
                    LD      (START_FLAG),A
                    JP      INITIZAL

CALL_LD_SW:         LD      HL,SW_STATUS        10
                    INC     DE                  6
                    CALL    LD_SW               17
                    INC     DE                  6
                    JP      START1              10
```

เวลาที่ใช้ของโปรแกรมย่อย CALL_LD_SW: = 10T+6T+17T+6T+10T

= 49T

```
CALL_LD_OUT:        LD      HL,OUT_STATUS       10
                    INC     DE                  6
                    CALL    LD_CTO              17
                    INC     DE                  6
                    JP      START1              10
```

เวลาที่ใช้ของโปรแกรมย่อย CALL_LD_OUT: = 10T+6T+17T+6T+10T

= 49T

```
CALL_LD_TR:         LD      HL,TR_STATUS        10
                    INC     DE                  6
                    CALL    LD_CTO              17
                    INC     DE                  6
                    JP      START1              10
```

เวลาที่ใช้ของโปรแกรมย่อย CALL_LD_TR: = 10T+6T+17T+6T+10T

= 49T

```
                                                          T states
CALL_LD_CNT:        LD      HL,CNT_STATUS                   10
                    INC     DE                              6
                    CALL    LD_CTO                          17
                    INC     DE                              6
                    JP      START1                          10
```

เวลาที่ใช้ของโปรแกรมย่อย CALL_LD_CNT: = 10T+6T+17T+6T+10T

= 49T

```
CALL_LD_TIM:        LD      HL,TIM_STATUS                   10
                    INC     DE                              6
                    CALL    LD_CTO                          17
                    INC     DE                              6
                    JP      START1                          10
```

เวลาที่ใช้ของโปรแกรมย่อย CALL_LD_TIM: = 10T+6T+17T+6T+10T

= 49T

```
CALL_LDN_SW:        LD      HL,SW_STATUS                    10
                    INC     DE                              6
                    CALL    LDN_SW                          17
                    INC     DE                              6
                    JP      START1                          10
```

เวลาที่ใช้ของโปรแกรมย่อย CALL_LDN_SW: = 10T+6T+17T+6T+10T

= 49T

```
                                                        T states
CALL_LDN_OUT:       LD      HL,OUT_STATUS               10
                    INC     DE                          6
                    CALL    LDN_CTO                     17
                    INC     DE                          6
                    JP      START1                      10
```

เวลาที่ใช้ของโปรแกรมย่อย CALL_LDN_OUT: = 10T+6T+17T+6T+10T

= 49T

```
CALL_LDN_CNT:       LD      HL,CNT_STATUS               10
                    INC     DE                          6
                    CALL    LDN_CTO                     17
                    INC     DE                          6
                    JP      START1                      10
```

เวลาที่ใช้ของโปรแกรมย่อย CALL_LDN_CTO: = 10T+6T+17T+6T+10T

= 49T

```
CALL_LDN_TIM:       LD      HL,TIM_STATUS               10
                    INC     DE                          6
                    CALL    LDN_CTO                     17
                    INC     DE                          6
                    JP      START1                      10
```

เวลาที่ใช้ของโปรแกรมย่อย CALL_LDN_TIM: = 10T+6T+17T+6T+10T

= 49T

```
                                                        T states
CALL_AND_SW:    LD      HL,SW_STATUS            10
                INC     DE                      6
                CALL    AND_SW                  17
                INC     DE                      6
                JP      START1                  10
```

เวลาที่ใช้ของโปรแกรมย่อย CALL_AND_SW: = 10T+6T+17T+6T+10T

= 49T

```
CALL_AND_OUT:   LD      HL,OUT_STATUS           10
                INC     DE                      6
                CALL    AND_CTO                 17
                INC     DE                      6
                JP      START1                  10
```

เวลาที่ใช้ของโปรแกรมย่อย CALL_AND_CTO: = 10T+6T+17T+6T+10T

= 49T

```
CALL_AND_CNT:   LD      HL,CNT_STATUS           10
                INC     DE                      6
                CALL    AND_CTO                 17
                INC     DE                      6
                JP      START1                  10
```

เวลาที่ใช้ของโปรแกรมย่อย CALL_AND_CNT: = 10T+6T+17T+6T+10T

= 49T

			T states
CALL_AND_TIM:	LD	HL,TIM_STATUS	10
	INC	DE	6
	CALL	AND_CTO	17
	INC	DE	6
	JP	START1	10

เวลาที่ใช้ของโปรแกรมย่อย CALL_AND_TIM: = 10T+6T+17T+6T+10T

= 49T

CALL_ANDN_SW:	LD	HL,SW_STATUS	10
	INC	DE	6
	CALL	ANDN_SW	17
	INC	DE	6
	JP	START1	10

เวลาที่ใช้ของโปรแกรมย่อย CALL_ANDN_SW: = 10T+6T+17T+6T+10T

= 49T

CALL_ANDN_OUT:	LD	HL,OUT_STATUS	10
	INC	DE	6
	CALL	ANDN_CTO	17
	INC	DE	6
	JP	START1	10

เวลาที่ใช้ของโปรแกรมย่อย CALL_ANDN_OUT: = 10T+6T+17T+6T+10T

= 49T

			T states
CALL_ANDN_CNT:	LD	HL,CNT_STATUS	10
	INC	DE	6
	CALL	ANDN_CTO	17
	INC	DE	6
	JP	START1	10

เวลาที่ใช้ของโปรแกรมย่อย CALL_ANDN_CNT: = 10T+6T+17T+6T+10T

= 49T

CALL_ANDN_TIM:	LD	HL,TIM_STATUS	10
	INC	DE	6
	CALL	ANDN_CTO	17
	INC	DE	6
	JP	START1	10

เวลาที่ใช้ของโปรแกรมย่อย CALL_ANDN_TIM: = 10T+6T+17T+6T+10T

= 49T

CALL_OR_SW:	LD	HL,SW_STATUS	10
	INC	DE	6
	CALL	OR_SW	17
	INC	DE	6
	JP	START1	10

เวลาที่ใช้ของโปรแกรมย่อย CALL_OR_SW: = 10T+6T+17T+6T+10T

= 49T

			T states
CALL_OR_OUT:	LD	HL,OUT_STATUS	10
	INC	DE	6
	CALL	OR_CTO	17
	INC	DE	6
	JP	START1	10

เวลาที่ใช้ของโปรแกรมย่อย CALL_OR_OUT: = 10T+6T+17T+6T+10T

= 49T

CALL_OR_CNT:	LD	HL,CNT_STATUS	10
	INC	DE	6
	CALL	OR_CTO	17
	INC	DE	6
	JP	START1	10

เวลาที่ใช้ของโปรแกรมย่อย CALL_OR_CNT: = 10T+6T+17T+6T+10T

= 49T

CALL_OR_TIM:	LD	HL,TIM_STATUS	10
	INC	DE	6
	CALL	OR_CTO	17
	INC	DE	6
	JP	START1	10

เวลาที่ใช้ของโปรแกรมย่อย CALL_OR_TIM: = 10T+6T+17T+6T+10T

= 49T

			T states
CALL_ORN_SW:	LD	HL,SW_STATUS	10
	INC	DE	6
	CALL	ORN_SW	17
	INC	DE	6
	JP	START1	10

เวลาที่ใช้ของโปรแกรมย่อย CALL_ORN_SW: = 10T+6T+17T+6T+10T

= 49T

CALL_ORN_OUT:	LD	HL,OUT_STATUS	10
	INC	DE	6
	CALL	ORN_CTO	17
	INC	DE	6
	JP	START1	10

เวลาที่ใช้ของโปรแกรมย่อย CALL_ORN_OUT: = 10T+6T+17T+6T+10T

= 49T

CALL_ORN_CNT:	LD	HL,CNT_STATUS	10
	INC	DE	6
	CALL	ORN_CTO	17
	INC	DE	6
	JP	START1	10

เวลาที่ใช้ของโปรแกรมย่อย CALL_ORN_CNT: = 10T+6T+17T+6T+10T

= 49T

			T states
CALL_ORN_TIM:	LD	HL,TIM_STATUS	10
	INC	DE	6
	CALL	ORN_CTO	17
	INC	DE	6
	JP	START1	10

เวลาที่ใช้ของโปรแกรมย่อย CALL_ORN_TIM: = 10T+6T+17T+6T+10T

= 49T

CALL_AND_LD:	DEC	IY	10
	LD	A,(IY+0)	19
	LD	B,A	4
	DEC	IY	10
	LD	A,(IY+0)	19
	AND	B	0
	LD	(IY+0),A	19
	INC	IY	10
	INC	DE	6
	INC	DE	6
	JP	START1	10

เวลาที่ใช้ของโปรแกรมย่อย CALL_AND_LD: = (10+19+4+10+19+19

+10+6+6+10)T

= 113T

			T states
CALL_OR_LD:	DEC	IY	10
	LD	A,(IY+0)	19
	LD	B,A	4
	DEC	IY	10
	LD	A,(IY+0)	19
	OR	B	0
	LD	(IY+0),A	19
	INC	IY	10
	INC	DE	6
	INC	DE	6
	JP	START1	10

เวลาที่ใช้ของโปรแกรมย่อย CALL_OR_LD: = (10+19+4+10+19+19
+10+6+6+10)T
= 113T

CALL_CNT:	INC	DE	6
	CALL	COUNTER	17
	INC	DE	6
	INC	DE	6
	INC	DE	6
	JP	START1	10

เวลาที่ใช้ของโปรแกรมย่อย CALL_CNT: = (6+17+6+6+6+10)T
= 51T

```
                                                        T states
CALL_TIM:       INC     DE                              6
                CALL    TIMER                           17
                INC     DE                              6
                INC     DE                              6
                INC     DE                              6
                LD      A,(DE)                          7
                CP      1DH                             0
                JR      Z,NEXT_COM2                     7
                CP      1EH                             0
                JR      Z,NEXT_COM2                     7
                LD      HL,FLAG_OUT                     10
                LD      (HL),00H                        10
NEXT_COM2:      JP      START1                          10
```

เวลาที่ใช้ของโปรแกรมย่อย CALL_TIM: = (6+17+6+6+6+7+7+7+10+10+10)T

= 92T

```
CALL_OUT:       INC     DE                              6
                CALL    OUT                             17
                INC     DE                              6
                LD      A,(DE)                          7
                CP      1DH                             0
                JR      Z,NEXT_COM                      7
                CP      1EH                             0
                JR      Z,NEXT_COM                      7
                CP      20H                             0
                JR      Z,NEXT_COM                      7
                LD      HL,FLAG_OUT                     10
```

```
                    LD      HL,FLAG_OUT             10
                    LD      (HL),00H                10
                                                    T states
NEXT_COM:           JP      START1                  10
```

เวลาที่ใช้ของโปรแกรมย่อย CALL_OUT: = (6+17+6+7+7+7+7+10+10+10)T

= 87T

```
CALL_OUTN:          INC     DE                      6
                    CALL    OUT_NOT                 17
                    INC     DE                      6
                    LD      A,(DE)                  7
                    CP      1DH                     0
                    JR      Z,NEXT_COM1             7
                    CP      1EH                     0
                    JR      Z,NEXT_COM1             7
                    CP      20H                     0
                    JR      Z,NEXT_COM1             7
                    LD      HL,FLAG_OUT             10
                    LD      (HL),00H                10
NEXT_COM1:          JP      START1                  10
```

เวลาที่ใช้ของโปรแกรมย่อย CALL_OUTN: = (6+17+6+7+7+7+7+10+10+10)T

= 87T

```
CALL_OUT_TR:        LD      HL,TR_STATUS            10
                    INC     DE                      6
                    CALL    OUT_TR                  17
                    INC     DE                      6
                    JP      START1                  10
```

เวลาที่ใช้ของโปรแกรมย่อย CALL_OUT_TR: = (10+6+17+6+10)T

= 49T

```
                                                        T states
CALL_END:       LD      B,20H                           7
CHECK_IN:       IN      A,(40H)                         11
                BIT     6,A                             8
                JR      Z,LOAD_FILE                     7
                DJNZ    CHECK_IN                        13
                LD      DE,ADDRESS_PRO                  10
                JP      START                           10
LOAD_FILE:      LD      HL,ADDRSS_PRO                   10
                DI                                      4
                CALL    RX_LINE                         17
                CALL    CLEAR_RAM                       17
                EI                                      4
                JP      START                           10
```

เวลาที่ใช้ของโปรแกรมย่อย CALL_END: = (7+32(11+8+7+13)-5+10+10)T

= 1270T

LD SW

```
LD_SW:          PUSH    AF                              11
                PUSH    BC                              11
                LD      B,08H                           7
                LD      HL,SW_STATUS                    10
                IN      A,(50H)                         11
NEXT_SW1:       LD      C,00H                           7
                RRA                                     0
                RL      C                               0
                LD      (HL),C                          7
                INC     HL                              6
                DJNZ    NEXT_SW1                        13
```

			T states
	LD	A,(DE)	7
	LD	HL,SW_STATUS	10
	CALL	LOOK_UP	17
	LD	A,(HL)	7
	LD	(IY+0),A	19
	INC	IY	10
	CALL	WATCH_DOG	17
	POP	BC	10
	POP	AF	10
	RET		10

เวลาที่ใช้ของโปรแกรมย่อย LD_SW: = (11+11+7+10+11+8(7+7+6+13)-5
+7+10+17+7+19+10+17+10+10+10)T
= 426T

OUT PORT

OUT:	PUSH	BC	11
	PUSH	DE	11
	LD	HL,FLAG_OUT	10
	LD	A,(HL)	7
	CP	00H	0
	JR	Z,POP_REGA	12
	LD	A,(IY+0)	19
	LD	B,A	4
	JR	NEXT_OUT	12
POP_REGA:	DEC	IY	10
	LD	A,(IY+0)	19
	LD	B,A	4

			T states
NEXT_OUT:	LD	HL,OUT_STATUS	10
	LD	A,(DE)	7
	CALL	LOOK_UP	17
	LD	(HL),B	7
	LD	HL,OUT_STATUS	10
	LD	B,08H	7
RD_OUT:	LD	A,(HL)	7
	AND	01H	0
	RRA		0
	RR	C	0
	INC	HL	6
	DJNZ	RD_OUT	13
	LD	A,C	4
	OUT	(51H),A	11
	LD	HL,FLAG_OUT	10
	LD	(HL),01H	10
	CALL	WATCH_DOG	17
	POP	DE	10
	POP	BC	10
	RET		10

เวลาที่ใช้ในโปรแกรมย่อย OUT: = (11+11+10+7+12+19+4+12+10+19
+4+10+7+17+7+10+7+8(7+6+13)-5
+4+11+10+10+17+10+10+10)T
= 462T

T states

LD COUNTER TIMER OUT

```
LD_CTO:     LD      A,(DE)          7
            CALL    LOOK_UP         17
            LD      A,(HL)          7
            LD      (IY+0),A        19
            INC     IY              10
            CALL    WATCH_DOG       17
            RET                     10
```

เวลาที่ใช้ของโปรแกรมย่อย LD_CTO: = (7+17+7+19+10+17+10)T

= 87T

LD-NOT SW

```
LDN_SW:     CALL    LD_SW           17
            DEC     IY              10
            LD      A,(IY+0)        19
            CPL     A               4
            AND     01H             0
            LD      (IY+0),A        19
            INC     IY              10
            RET                     10
```

เวลาที่ใช้ของโปรแกรมย่อย LDN_SW: = (17+10+19+4+19+10+10)

= 89T

AND SW

```
AND_SW:     DEC     IY              10
            LD      A,(IY+0)        19
            LD      B,A             4
            CALL    LD_SW           17
```

			T states
	DEC	IY	10
	LD	A,(IY+0)	19
	AND	B	0
	LD	(IY+0),A	19
	INC	IY	10
	RET		10

เวลาที่ใช้ของโปรแกรมย่อย AND_SW: = (10+19+4+17+10+19+19+10+10)T

= 118T

AND-NOT SW

ANDN_SW:	DEC	IY	10
	LD	A,(IY+0)	19
	LD	B,A	4
	CALL	LDN_SW	17
	DEC	IY	10
	LD	A,(IY+0)	19
	AND	B	0
	LD	(IY+0),A	19
	INC	IY	10
	RET		10

เวลาที่ใช้ของโปรแกรมย่อย ANDN_SW: = (10+19+4+17+10+19+19+10+10)T

= 118T

OR SW

OR_SW:	DEC	IY	10
	LD	A,(IY+0)	19
	LD	B,A	4
	CALL	LD_SW	17

			T states
	DEC	IY	10
	LD	A,(IY+0)	19
	OR	B	0
	LD	(IY+0),A	19
	INC	IY	10
	RET		10

เวลาที่ใช้ของโปรแกรมย่อย OR_SW: = (10+19+4+17+10+19+19+10+10)T

= 118T

OR-NOT SW

ORN_SW:	DEC	IY	10
	LD	A,(IY+0)	19
	LD	B,A	4
	CALL	LDN_SW	17
	DEC	IY	10
	LD	A,(IY+0)	19
	OR	B	0
	LD	(IY+0),A	19
	INC	IY	10
	RET		10

เวลาที่ใช้ของโปรแกรมย่อย ORN_SW: = (10+19+4+17+10+19+19+10+10)T

= 118T

OUT TR

OUT_TR:	LD	A,(DE)	7
	CALL	LOOK_UP	17
	DEC	IY	10
	LD	A,(IY+0)	19

		T states
LD	B,(HL)	7
INC	HL	6
LD	C,(HL)	7
INC	BC	6
LD	(HL),C	7
DEC	HL	6
LD	B,(HL)	7
INC	DE	6
LD	A,(DE)	7
LD	H,A	4
INC	DE	6
LD	A,(DE)	7
LD	L,A	4
SRL	B	0
RR	C	0
LD	A,B	4
CP	H	0
JR	NZ,EXIT_CNT	7
LD	A,C	4
CP	L	0
JR	NZ,EXIT_CNT	7
POP	DE	10
LD	HL,CNT_STATUS	10
LD	A,(DE)	7
CALL	LOOK_UP	17
LD	(HL),01H	10
LD	HL,COUNT_STORE	10
LD	A,(DE)	7

			T states
	LD	(HL),A	7
	INC	IY	10
	CALL	WATCH_DOG	17
	RET		10

เวลาที่ใช้ของโปรแกรมย่อย OUT_TR: = (7+17+10+19+7+10+17+10)T

= 97T

COUNTER PLC

COUNTER:	PUSH	DE	11
	LD	HL,CNT_STATUS	10
	DEC	IY	10
	LD	A,(IY+0)	19
	DEC	IY	10
	LD	B,(IY+0)	19
	CP	01H	0
	JR	Z,RESET_CNT	7
	LD	HL,STORE_INPUT	10
	LD	A,(DE)	7
	PUSH	DE	11
	CALL	LOOK_UP	17
	LD	A,(HL)	7
	LD	(HL),B	7
	XOR	B	0
	CP	00H	0
	JR	Z,EXIT_CNT	7
	LD	HL,COUNT_STORE	10
	LD	A,(DE)	7
	CALL	LOOK_UP1	17

```
                                                    T states
                CALL    LOOK_UP1                    17
                LD      (HL),00H                    10
                INC     HL                          6
                LD      (HL),00H                    10
                INC     HL                          6
                LD      (HL),00H                    10
                CALL    WATCH_DOG                   17
                POP     DE                          10
                RET                                 10
EXIT_CNT:       CALL    WATCH_DOG                   17
                POP     DE                          10
                POP     DE                          10
                RET                                 10
RESET_CNT:      LD      HL,STORE_INPUT              10
                LD      A,(DE)                      7
                CALL    LOOK_UP                     17
                LD      (HL),01H                    10
                LD      HL,CNT_STATUS               10
                LD      A,(DE)                      7
                CALL    LOOK_UP                     17
                LD      (HL),00H                    10
                LD      HL,COUNT_STORE              10
                LD      A,(DE)                      7
                CALL    LOOK_UP1                    17
                LD      (HL),00H                    10
                INC     HL                          6
                LD      (HL),00H                    10
                CALL    WATCH_DOG                   17
```

			T states
	POP	DE	10
	RET		10

เวลาที่ใช้ของโปรแกรม COUNTER: = (11+10+10+19+10+19+7+10+7
+11+17+7+7+7+10+7+17+7+6+7
+6+7+6+7+6+7+4+6+7+4+4+7+4+7
+10+10+7+17+10+10+7+17+10+6+10
+6+10+17+10+10+17+10+10+10+10
+7+17+10+10+7+17+10+10+7+17+10
+6+10+17+10+10)T
= 680T

TIMER PLC

TIMER:	PUSH	DE	11
	LD	A,(FLAG_OUT)	13
	CP	01H	0
	JR	NZ,POP_REG	7
	LD	A,(IY+0)	19
	JR	COMPARE_A	12
POP_REG:	DEC	IY	10
	LD	A,(IY+0)	19
COMPARE_A:	CP	01H	0
	JR	NZ,TIME_0	7
	LD	HL,FLAG_TIM	10
	LD	A,(DE)	7
	CALL	LOOK_UP	17
	LD	A,(HL)	7
	CP	01H	0
	JR	Z,CHECK_TIME	7
	LD	A,01H	7

			T states
	LD	(HL),A	7
	LD	HL,TIME_STORE	10
	LD	A,(DE)	7
	LD	B,A	4
	CALL	LOOK_UP1	17
	INC	DE	6
	LD	A,(DE)	7
	LD	(HL),A	7
	INC	HL	6
	INC	DE	6
	LD	A,(DE)	7
	LD	(HL),A	7
	LD	HL,FLAG_TIM	10
	LD	A,B	4
	CALL	LOOK_UP	17
	LD	A,01H	7
	LD	(HL),A	7
	LD	HL,TIM_STATUS	10
	LD	A,B	4
	CALL	LOOK_UP	17
	LD	A,00H	7
	LD	(HL),A	7
	JR	TIME_EXIT	12
CHECK_TIME:	LD	HL,TIME_STORE	10
	LD	A,(DE)	7
	CALL	LOOK_UP1	17
	LD	A,(HL)	7
	CP	00H	0

```
                                                        T states
                    JR      NZ,TIME_EXIT                7
                    INC     HL                          6
                    LD      A,(HL)                      7
                    CP      00H                         0
                    JR      NZ,TIME_EXIT                7
                    LD      A,(DE)                      7
                    LD      HL,FLAG_TIM                 10
                    CALL    LOOK_UP                     17
                    LD      A,01H                       7
                    LD      (HL),A                      7
                    LD      HL,TIM_STATUS               10
                    LD      A,(DE)                      7
                    CALL    LOOK_UP                     17
                    LD      A,01H                       7
                    LD      (HL),A                      7
TIME_EXIT:          LD      HL,FLAG_OUT                 10
                    LD      (HL),01H                    10
                    CALL    WATCH_DOG                   17
                    POP     DE                          10
                    RET                                 10
TIME_0:             LD      HL,TIM_STATUS               10
                    LD      A,(DE)                      7
                    CALL    LOOK_UP                     17
                    LD      A,00H                       7
                    LD      (HL),A                      7
                    LD      HL,FLAG_TIM                 10
                    LD      A,(DE)                      7
                    AND     0FH                         0
```

			T states
	ADD	A,L	4
	LD	L,A	4
	LD	A,00H	7
	LD	(HL),A	7
	JR	TIME_EXIT	12

เวลาที่ใช้ของโปรแกรมย่อย TIMER: = (11+13+7+19+12+10+19+7+10+7
+17+7+7+7+7+7+10+7+4+17+6+7+7+6
+6+7+7+10+4+17+7+7+10+4+17+7+7
+12+10+7+17+7+7+6+7+7+7+10+17+7
+7+10+17+7+7+10+10+17+10+10+10
+7+17+7+7+10+7+4+4+7+7+12)T
= 666T

DEC TIME

DEC_TIME:	PUSH	HL	11
	PUSH	AF	11
	PUSH	BC	11
	PUSH	DE	11
	LD	B,00H	7
	LD	HL,FLAG_TIM	10
NEXT_TIME:	LD	A,(HL)	7
	CP	01H	0
	JR	NZ,NEXT_FLAG	7
	PUSH	HL	11
	LD	HL,TIME_STORE	10
	LD	A,B	4
	CALL	LOOK_UP1	17
	LD	D,(HL)	7

```
                                                    T states
                    INC     HL                      6
                    LD      E,(HL)                  7
                    LD      A,D                     4
                    CP      00H                     0
                    JR      NZ,GO_NEXT              7
                    LD      A,E                     4
                    CP      00H                     0
                    JR      NZ,GO_NEXT              7
                    JR      EXIT_INT                12
GO_NEXT:            DEC     DE                      6
                    LD      (HL),E                  7
                    DEC     HL                      6
                    LD      (HL),D                  7
EXIT_INT:           POP     HL                      10
NEXT_FLAG:          INC     HL                      6
                    INC     B                       4
                    LD      A,B                     4
                    CP      0FH                     0
                    JR      NZ,NEXT_TIME            7
                    CALL    WATCH_DOG               17
                    POP     DE                      10
                    POP     BC                      10
                    POP     AF                      10
                    POP     HL                      10
                    RET                             10
```

T states

เวลาที่ใช้ของโปรแกรมย่อย DEC_TIME: = (11+11+11+11+7+10+7+7+11
+10+4+17+7+6+7+4+7+4+7+12+6
+7+6+7+10+6+4+4+7+17+10+10
+10+10+10)T
= 295T

LOOK UP

LOOK_UP:	AND	0FH	0
	ADD	A,L	4
	LD	L,A	4
	RET		10

เวลาที่ใช้ของโปรแกรมย่อย LOOK_UP: = (4+4+10)T
= 18T

LOOK UP1

LOOK_UP1:	ADD	A,A	4
	ADD	A,L	4
	LD	L,A	4
	RET		10

เวลาที่ใช้ของโปรแกรมย่อย LOOK_UP1: = (4+4+4+10)T
= 22T

T states

WATCH DOG

```
WATCH_DOG:      PUSH    AF              11
                LD      A,4EH           7
                OUT     (0F1H),A        11
                LD      A,0E7H          7
                OUT     (0F0H),A        11
                POP     AF              10
                RET                     10
```

เวลาที่ใช้ของโปรแกรมย่อย WATCH_DOG: = (11+7+11+7+11+10+10)T

= 67T

สรุป เวลาที่ใช้ในการเรียกโปรแกรมย่อย

CALL_LD_SW	49T
CALL_LD_OUT	49T
CALL_LD_TR	49T
CALL_LD_CNT	49T
CALL_LD_TIM	49T
CALL_LDN_SW	49T
CALL_LDN_OUT	49T
CALL_LDN_CNO	49T
CALL_LDN_TIM	49T
CALL_AND_SW	49T
CALL_AND_CTO	49T
CALL_AND_CNT	49T
CALL_AND_TIM	49T
CALL_ANDN_SW	49T
CALL_ANDN_OUT	49T
CALL_ANDN_CNT	49T
CALL_ANDN_TIM	49T
CALL_OR_SW	49T
CALL_OR_OUT	49T
CALL_OR_CNT	49T
CALL_OR_TIM	49T
CALL_ORN_SW	49T
CALL_ORN_OUT	49T

CALL_ORN_CNT	49T
CALL_ORN_TIM	49T
CALL_AND_LD	113T
CALL_OR_LD	113T
CALL_CNT	51T
CALL_TIM	92T
CALL_OUTN	87T
CALL_OUT_TR	49T
CALL_END	1270T

<u>สรุปเวลาที่ใช้ในแต่ละโปรแกรมย่อย</u>

LD_SW	426T
OUT	462T
LD_CTO	87T
LDN_SW	89T
AND_SW	118T
ANDN_SW	118T
OR_SW	118T
ORN_SW	118T
OUT_TR	97T
COUNTER	680T
TIMER	666T
DEC_TIM	295T
LOOK_UP	18T
LOOK_UP1	22T
WATCH_DOG	67T

```
;*************************************************
;          TRANSMITT DATA BY RS-232
;          9600 B/S, 1 STOP,NON PARITY,8 BITS
;           INPUT REG -> D
;*************************************************
TX:             LD      A,80H
                OUT     (40H),A
                CALL    DELAY_TX
                LD      B,08H
NEXT_TXBIT:     XOR     A
                RRC     D
                RRA
                RRA
                RRA
                SETT    7,A
                OUT     (40H),A
                CALL    DELAY_TX
                DJNZ    NEXT_TXBIT
                LD      A,0A0H
                OUT     (40H),A
                CALL    DELAY_TX
                CALL    WATCH_DOG
                RET
DELAY_TX:       LD      A,1EH   ;DELAY VALUE
                SUB     02H
DECTX:          DEC     A
                JR      NZ,DECTX
                RET
```

```
;*****************************************************
;       SEND DATA INPUT,OUTPUT TO COMPUTER
;       FORMAT->    {{{xxxxxxxxxxxxxxxx
;*****************************************************

SEND_SW:        PUSH    AF
                PUSH    BC
                PUSH    DE
                PUSH    HL
                PUSH    IY
                LD      HL,SW_STATUS
                LD      C,08H
                LD      D,7BH   ;CHAR '{'
                CALL    TX
                LD      D,7BH
                CALL    TX
                LD      D,7BH
                CALL    TX

NEXT_SW:        LD      A,30H   ;CHAR '0'
                LD      D,(HL)
                ADD     A,D
                LD      D,A
                CALL    TX
                INC     HL
                DEC     C
                JR      NZ,NEXT_SW
                LD      HL,OUT_STATUS
                LD      C,08H
NEXT_OUT1:      LD      A,30H   ;CHAR '0'
```

```
LD      D,(HL)
ADD     A,D
LD      D,A
CALL    TX
INC     HL
DEC     C
JR      NZ,NEXT_OUT1
POP     IY
POP     HL
POP     DE
POP     BC
POP     AF
RET
```

```
;*************************************************
;       RECEIVER  DATA FORM RS-232
;       9600 B/S, 8 BITS, 1 STOP,NON PARITY
;       INPUT->     HL =>ADDRESS TO SAVE DATA
;*************************************************
RX_LINE:            LD      C,03H
START_BIT:          IN      A,(40H)
                    BIT     6,A
                    JR      NZ,START_BIT
READ_NEW:           NOP
                    CALL    DELAY
                    LD      D,00H
                    LD      B,08H
NEXT_BIT:           IN      A,(40H)
                    RLA
                    RLA
                    RR      D
                    CALL    DELAY
                    DJNZ    NEXT_BIT
                    LD      A,D
                    CP      0FFH
                    JR      Z,RX_LINE1
                    CP      0FBH
                    JR      NZ,START_BIT
RX_LINE1:           LD      C,03H
START_BIT1:         IN      A,(40H)
                    BIT     6,A
                    JR      NZ,START_BIT1
                    CALL    WATCH_DOG
READ_NEW1:          NOP
```

```
                    CALL    DELAY
                    LD      D,00H
                    LD      B,08H
NEXT_BIT1:          IN      A,(40H)
                    RLA
                    RLA
                    RR      D
                    CALL    DELAY
                    DJNZ    NEXT_BIT1
                    LD      (HL),D
                    INC     HL
                    LD      A,D
                    CP      01BH
                    JR      NZ,RX_LINE1
                    DEC     C
                    JR      NZ,START_BIT1
                    CALL    WATCH_DOG
                    LD      B,20H
                    LD      D,7BH
                    PUSH    BC
                    CALL    TX
                    POP     BC
                    LD      A,01
                    LD      (START_FLAG),A
                    RET
DELAY:              LD      A,1EH
DECA:               DEC     A
                    JR      NZ,DECA
                    RET
```

การคำนวณหาเวลาที่ใช้ในโปรแกรม

จากตัวอย่าง TEST 1 หน้า 77 สามารถหาเวลาที่ใช้ในโปรแกรมได้ดังนี้

1.คำสั่ง LD 00 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_LD_SW LD_SW WATCH_DOG LOOK_UP

2.คำสั่ง OR 01 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_OR_SW OR_SW LD_SW

3.คำสั่ง AND-NOT 02 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_ANDN_SW ANDN_SW LDN_SW

4.คำสั่ง OUT 500 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_OUT OUT WATCH_DOG LOOK_UP

5.คำสั่ง END เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_END

เวลาที่ใช้ในโปรแกรม = เวลาของผลรวมของโปรแกรมย่อยทั้งหมด

= (CALL_LD_SW+LD_SW+WATCH_DOG+LOOK_UP+CALL_OR_SW
+OR_SW+LD_SW+CALL_ANDN_SW+ANDN_SW+LDN_SW+CALL_OUT
+OUT+LOOK_UP+WATCH_DOG+CALL_END)T

= (49+426+67+18+49+118+426+49+118+89+87+462+18+67
+1270)T

= 3313T

บอร์ดคอนโทรลเลอร์ RUN โปรแกรมโดยใช้ความถี่ 5 MHz จะได้ T=1/5 MHz = 0.2 μsec

เวลาที่ใช้ในโปรแกรม TEST1 = 3313*0.2 μsec

= 662.6 μsec

จากตัวอย่าง TEST 2 หน้า 79 สามารถหาเวลาที่ใช้ในโปรแกรมได้ดังนี้

1.คำสั่ง LD-NOT 00 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_LDN_SW LDN_SW LD_SW

2.คำสั่ง AND 01 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_AND_SW AND_SW LD_SW

3.คำสั่ง LD 02 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_LD_SW LD_SW LOOK_UP WATCH_DOG

4.คำสั่ง AND-NOT 03 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_ANDN_SW ANDN_SW LDN_SW

5.คำสั่ง OR-LD เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_OR_LD

6.คำสั่ง OUT 507 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_OUT OUT LOOK_UP WATCH_DOG

7.คำสั่ง END เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_END

เวลาที่ใช้ในโปรแกรม = เวลาของผลรวมของโปรแกรมย่อยทั้งหมด

=(49+89+426+49+118+426+49+426+18+67+49+118+89+113
+49+462+18+67+1270)T

= 3952T

บอร์ดคอนโทรลเลอร์ RUN โปรแกรมโดยใช้ความถี่ 5 MHz จะได้ T=1/5 MHz = 0.2 μsec

เวลาที่ใช้ในโปรแกรม TEST2 = 0.2*3952 μsec

= 790.9 μsec

จากตัวอย่าง TEST 3 หน้า 82 สามารถหาเวลาที่ใช้ในโปรแกรมได้ดังนี้

1.คำสั่ง LD 00 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_LD_SW LD_SW LOOK_UP WATCH_DOG

2.คำสั่ง TIM 00 #0003 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_TIM TIMER LOOK_UP LOOK_UP1 LOOK_UP LOOK_UP LOOK_UP1 LOOK_UP LOOK_UP WATCH_DOG LOOK_UP

3.คำสั่ง LD TIM 00 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_LD_TIM LD_CTO LOOK_UP WATCH_DOG

4.คำสั่ง LD-NOT 01 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_LDN_SWW LDN_SW LD_SW

5.คำสั่ง OR 02 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_OR_SW OR_SW LD_SW

6.คำสั่ง AND-LD เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_AND_LD

7.คำสั่ง OUT 507 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_OUT OUT LOOK_UP WATCH_DOG

8.คำสั่ง END เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_END

เวลาที่ใช้ในโปรแกรม = เวลาของผลรวมของโปรแกรมย่อยทั้งหมด

= (49+426+18+67+92+666+18+22+18+18+22+18+18+67+18
+49+87+18+67+49+89+426+49+118+426+113+462+18+67
+1270)T

= 4845T

บอร์ดคอนโทรลเลอร์ RUN โปรแกรมโดยใช้ความถี่ 5 MHz จะได้ T=1/5 MHz = 0.2 μsec

เวลาที่ใช้ในโปรแกรม TEST3 = 4845*0.2 μsec

= 969 μsec

จากตัวอย่าง TEST 4 หน้า 84 สามารถหาเวลาที่ใช้ในโปรแกรมได้ดังนี้

1.คำสั่ง LD 00,LD 01 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_LD_SW LD_SW LOOK_UP WATCH_DOG

2.คำสั่ง CNT 00 #0001 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_CNT COUNTER LOOK_UP LOOK_UP1 LOOK_UP LOOK_UP1 WATCH_DOG LOOK_UP LOOK_UP1 WATCH_DOG

3.คำสั่ง LD CNT 00 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_LD_CNT LD_CTO LOOK_UP WATCH_DOG

4.คำสั่ง AND-NOT 02 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_ANDN_SW ANDN_SW LDN_SW

5.คำสั่ง OUT 500 - OUT 507 เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_OUT OUT LOOK_UP WATCH_DOG

6.คำสั่ง END เป็นคำสั่งที่ประกอบด้วยโปรแกรมย่อย CALL_END

เวลาที่ใช้ในโปรแกรม = เวลาของผลรวมของโปรแกรมย่อยทั้งหมด

= (2*49+2*426+14*18+14*67+51+680+3*22+49+87+8*87 +8*462+1270)T

= 8735T

บอร์ดคอนโทรลเลอร์ RUN โปรแกรมโดยใช้ความถี่ 5 MHz จะได้ T=1/5 MHz = 0.2 μsec

เวลาที่ใช้ในโปรแกรม TEST4 = 8735*0.2 μsec

= 1.747 msec

This is source code PROGRAM EDIT.H

```
/*    PROJECT  PROGRAMMABLE LOGIC CONTROLLER SIMULATED BY PERSONAL
 COMPUTER (PLC SIMULATOR)
   BY.
  1. MR. Pissanu    Promwong
  2. MR. Patchanon  Kritpipat
  3. MR. Somchai    Chaoveang
    Instrument Engineering  KMIT'L
 DATE  16  October  1994
/*       THE ENTIRE SCREEN-EDITOR SYSTEM
The screen editor is a part of program PLC SIMUTOR.
Copy form Born to code in C in Screen Editor                    */

/* A Screen Editor Subsystem */
#define TURBOC
/* If you use microsoft C, #define MICROSOFTC
   instead of TURBOC. If you have a different
   C compiler, see the text.
*/

#include      <stdio.h>
#include      <dos.h>
#include      <string.h>
#include      <bios.h>
#include      <conio.h>
#include      <graphics.h>
#include      <stdlib.h>
#include      <alloc.h>
#define   VALUE_IN          7          /* Define maximum of input */
#define   VALUE_OUT         507        /* Define maximum of output */
```

```
#define   NUM_TIM_OR_CNT   15          /* Define maximum of counter,timmer */
#define   VALUE_TIM_OR_CNT 9999        /* Define maximum of value counter,timmer */
#define   BUF_SIZE         20000       /* Define buffer size of editor */
#define   LINE_LEN         75          /* Define maximum of coloumn */
#define   MAX_LINES        22          /* Define maximum of line */
#define   BOTTOM_LINE      24          /* Define  end line */
#define   KILL_BUF_SIZE    4*LINE_LEN  /* Define buffer size of toggle code*/
#define   REVERSE          0x70        /* Define reverse attribute  */
#define   NORMAL           0x07        /* Define noraml attribute */
#define   MAX              500         /* Define  buffer size of file timming*/
#define   TOPLINE          3           /* Define editor start line */
#define   START_X          3           /* Define editor start coloumn  */
#define   RECORD           21          /* Define  buffer in record of timming */
#define   NUM_SW           8           /* Define number of input switch */
#define   DATA_READY       0x100       /* Define communication ready */
#define   SETTINGS         (0xE0|0x00 |0x03|0X00) /* Define setting baude rate
                                        9600 b/s 1 stop no parity */
#define   LOGIC_MAX        800         /* Define buffer input file timming */
#define   OUT_MAX          9           /* Define maximum output */
#define   IN_MAX           9           /* Define maximum input */
#define   TR_MAX           9           /* Define maximum tr store */
#define   CNT_MAX          17          /* Define counter */
#define   TIM_MAX          17          /* Define timmer */
#define   TIM_STORE_MAX    49          /* Define timmer store */
#define   FLAG_TIM_MAX     17          /* Define flag timmer */
#define   CNT_STORE_MAX    33          /* Define counter store */
#define   SAVE_INPUT_MAX   17          /* Define input of counter */
#define   PAGE             15          /* Define maximum of ladder*/
```



```
/* Use in sim_function */
```

```
int   reg_a;

/* Values are use in function stack   */

int   stack[MAX];
int   tos=0;        /* pointer of stack */

/* Values are use in function  Simulate */

int   code_sw[IN_MAX];
int   code_out[OUT_MAX];
int   out_store[OUT_MAX];
int   in_store[IN_MAX];
int   code_tr[TR_MAX];
int   code_cnt[CNT_MAX];
int   code_tim[TIM_MAX];
int   time[TIM_STORE_MAX];
int   flag_tim[FLAG_TIM_MAX];
int   cnt[CNT_STORE_MAX];
int   input[SAVE_INPUT_MAX];

/*   Values  are use in function timming   */
/*   Step is number loop                   */

char  logic_in[LOGIC_MAX];
char  *logic,*endlogic;
int   storeinput[NUM_SW][RECORD];
int   storeoutput[NUM_SW][RECORD];
int   step,post_step,loop_count;
int   exit_sub;
```

```
int   flag1,flag0,flag_out,stx,sty;
int   *t,*end;

/* This variable use in function save screen and restore screen */

char      far *vid_mem;      /* address of memory display in textmode */
unsigned  char *save_scr;    /* pointer to save screen */
int       startx,endx,starty,endy;   /* ponit of screen */
int       vmode;             /* color or monochrome mode */

/*   Value in error  */

int  error_set,flag_convert,ErrorCode;

/* Vaule use in graphics function */

int  graphdriver,graphmode,c;
int  maxX,maxY,status,COM;
int  setexit;
char Gray50[] = { 0xaa,0x55,0xaa,0x55,0xaa,0x55,0xaa,0x55};

/* Value in function edit */

char buf_code[4*200];
char buf[BUF_SIZE];
char *curloc,  *endloc,*code_loc,*code_data,*code_end,*old_curloc;
int  scrnx, scrny,linee ;
unsigned ins;
char killbuf[KILL_BUF_SIZE];
```

```
/*  Charector  to  gennerate toggle code
    it simple function  */

char *code1[]={ "LD","LD-NOT","AND","AND-NOT","OR","OR-NOT",
 "OR-LD","AND-LD","CNT","TIM","OUT","OUT-NOT",
 "END" };
char *code2[]={ "TR","CNT","TIM" };
char name_code[10];
/* This function is use for convert text file      */
/*    in PLC to toggle code                        */
/*    Code                    Decimal              */
/*    LD        toggle code ->   1-5               */
/*    LD-NOT    toggle code ->   6-9               */
/*    AND       toggle code ->   10-13             */
/*    AND_NOT   toggle code ->   14-17             */
/*    OR        toggle code ->   18-21             */
/*    OR_NOT    toggle code ->   22-25             */
/*    AND-LD    toggle code ->   26                */
/*    OR-LD     toggle code ->   27                */
/*    CNT       toggle code ->   28                */
/*    TIM       toggle code ->   29                */
/*    OUT       toggle code ->   30                */
/*  OUT TR    toggle code ->   31                  */
/*    OUT-NOT   toggle code ->   32                */
/*    Sub set of LD                                */
/*       xx DATA POINT INPUT                       */
/*    LD xx     input                              */
```

```
 /*   LD xx     output                                    */
 /*   LD TR xx                                           */
 /*   LD CNT xx   counter number                         */
 /*   LD TIM xx   timmer number                          */

gen_token()
{
int n, a,carry;
div_t  x;
FILE *fp;

curloc=buf;
error_set=0;
code_data=code_end=buf_code;
linee=1;
/* Space or '\r'  */
while((*curloc ==' ' || *curloc =='\r') && curloc < endloc)
{if(*curloc=='\r')linee++; curloc++; }
while(curloc < endloc && error_set ==0 ){
/* get text to buffer */
puttable();
/* compare code  text in buffer and code */
a=compare_code(1);
/* parameter '1' code first table */
switch (a)  { /* main code */
 case 0:
 while(*curloc ==' ' || *curloc =='\r') {
 if(*curloc=='\r' )linee++; curloc++;}
 if (*curloc >= '0' && *curloc <= '9' ) {
 if(*curloc == '5'){putdata(2,0); break;
```

```
}else{ putdata(a+1,1);break;}}
else {
puttable();
a=compare_code(2);
switch (a) { /* Sub code */
  case 0:
 while(*curloc ==' ' || *curloc =='\r'){
if(*curloc=='\r' )linee++;
curloc++;
}
if (*curloc >='0' && *curloc <= '9') {
putdata(3,1);
break;}
else error_code(2);
 break;
case 1:
while(*curloc ==' ' || *curloc =='\r'){
if(*curloc=='\r' )linee++;
  curloc++;}
if(*curloc >= '0' && *curloc <= '9'){
  putdata(4,2);
  break;}
 else error_code(2);
break;
case 2:
while(*curloc ==' ' || *curloc =='\r'){
if(*curloc=='\r' )linee++;
  curloc++;}
 if(*curloc >= '0' && *curloc <= '9') {
putdata(5,2);
```

```
 break;}
else error_code(2);
break;
}
}
break;
case 1:
while(*curloc ==' ' || *curloc =='\r'){
if(*curloc=='\r' )linee++;
  curloc++;}
if (*curloc >= '0' && *curloc <= '9') {
if(*curloc == '5'){
putdata(7,0);
break;
}else{
putdata(6,1);
break;}
}
else {
 puttable();
 a=compare_code(2);
 switch (a) {   /* Sub code */
case 0:
  error_code(0);
  break;
case 1:
   while(*curloc ==' ' || *curloc =='\r'){
 if(*curloc=='\r' )linee++;
curloc++;}
    if(*curloc >= '0' && *curloc <= '9'){
```

```
putdata(8,2);
 break;}
else error_code(2);
break;
case 2:
  while(*curloc ==' ' || *curloc =='\r'){
  if(*curloc=='\r' )linee++;
  curloc++;}
   if(*curloc >= '0' && *curloc <= '9') {
   putdata(9,2);
break;}
else error_code(2);
break;
    }
}
break;
 case 2:
while(*curloc ==' ' || *curloc =='\r'){
    if(*curloc=='\r' )linee++;
    curloc++;}
if (*curloc >= '0' && *curloc <= '9' ) {
  if(*curloc == '5'){
putdata(11,0);
break;
   }else{
   putdata(10,1);
   break;}
   }
else {
    puttable();
```

```
      a=compare_code(2);
      switch (a) {  /* Sub code */
   case 0:
       error_code(0);
       break;
   case 1:
       while(*curloc ==' ' || *curloc =='\r'){
  if(*curloc=='\r' )linee++;
  curloc++;}
       if(*curloc >= '0' && *curloc <= '9'){
  putdata(12,2);
  break;
       }else error_code(2);
       break;
    case 2:
       while(*curloc ==' ' || *curloc =='\r'){
  if(*curloc=='\r' )linee++;
  curloc++;}
       if(*curloc >= '0' && *curloc <= '9') {
  putdata(13,2);
  break;
       }else error_code(2);
       break;
   }
       }
  break;
       case 3:
  while(*curloc ==' ' || *curloc =='\r'){
      if(*curloc=='\r' )linee++;
      curloc++;}
```

```
if (*curloc >= '0' && *curloc <= '9' ) {
   if(*curloc == '5'){
       putdata(15,0);
       break;
   }else{
       putdata(14,1);
       break;}
}else {
puttable();
a=compare_code(2);
switch (a) { /* Sub code */
   case 0:
error_code(0);
       break;
   case 1:
       while(*curloc ==' ' || *curloc =='\r'){
 if(*curloc=='\r' )linee++;
 curloc++;}
       if(*curloc >= '0' && *curloc <= '9'){
  putdata(16,2);
  break;
       }else error_code(2);
       break;
   case 2:
while(*curloc ==' ' || *curloc =='\r'){
    if(*curloc=='\r' )linee++;
    curloc++;}
if(*curloc >= '0' && *curloc <= '9') {
    putdata(17,2);
    break;
```

```
}else error_code(2);
break;
}
   }
break;
     case 4:
while(*curloc ==' ' || *curloc =='\r'){
  if(*curloc=='\r' )linee++;
  curloc++;}
if (*curloc >= '0' && *curloc <= '9' ) {
  if(*curloc == '5'){
    putdata(19,0);
    break;
   }else{
       putdata(18,1);
       break;}
 }else {
puttable();
a=compare_code(2);
switch (a) {                        /* Sub code */
    case 0:
       error_code(0);
       break;
   case 1:
      while(*curloc ==' ' || *curloc =='\r'){
if(*curloc=='\r' )linee++;
curloc++;}
      if(*curloc >= '0' && *curloc <= '9') {
putdata(20,2);
break;
```

```
        }else error_code(2);
        break;
    case 2:
        while(*curloc ==' ' || *curloc =='\r'){
if(*curloc=='\r' )linee++;
curloc++;}
        if(*curloc >= '0' && *curloc <= '9') {
putdata(21,2);
break;
        }else error_code(2);
        break;
}
   }
break;
     case 5:
while(*curloc ==' ' || *curloc =='\r'){
   if(*curloc=='\r' )linee++;
   curloc++;}
if (*curloc >= '0' && *curloc <= '9' ) {
   if(*curloc == '5'){
        putdata(23,0);
        break;
   }else{
        putdata(22,1);
        break;}
   }else {
puttable();
a=compare_code(2);
switch (a) {                    /* Sub code */
    case 0:
```

```
        error_code(0);
        break;
      case 1:
        while(*curloc ==' ' || *curloc =='\r'){
if(*curloc=='\r' )linee++;
curloc++;}
        if(*curloc >= '0' && *curloc <= '9'){
putdata(24,2);
break;
        }else error_code(2);
        break;
      case 2:
        while(*curloc ==' ' || *curloc =='\r'){
if(*curloc=='\r' )linee++;
curloc++;}
        if(*curloc >= '0' && *curloc <= '9') {
        putdata(25,2);
        break;
        }else error_code(2);
        break;
 }
      }
break;
      case 6:
*code_data=27;
code_data++;
code_end++;
*code_data=0;
code_data++;
code_end++;
```

```
curloc++;
break;
     case 7:
*code_data=26;
code_data++;
code_end++;
*code_data=0;
code_data++;
code_end++;
curloc++;
break;
     case 8:
while(*curloc ==' ' || *curloc =='\r'){
    if(*curloc=='\r' )linee++;
    curloc++;}
if(*curloc >= '0' && *curloc <= '9') {
    putdata(28,2);
    while(*curloc ==' ')curloc++;
    if (*curloc =='#'){
    curloc++;
    n=getnum(); /* save value counter to */
    if(n >= VALUE_TIM_OR_CNT) { /*Value over*/
      error_code(13);
      return(0);
      }
    curloc++;     /* buffer */
    x=div(n,255);
    n=x.quot;
    *code_data=n;
    code_data++;
```

```
      code_end++;
      n=x.rem;
      *code_data=n;
      code_data++;
      code_end++;
  }else {
        error_code(3);
        break;
        }
    }else {error_code(0);
 break;
 }
break;
     case 9:
while(*curloc ==' ' || *curloc =='\r'){
   if(*curloc=='\r' )linee++;
  curloc++;}
if(*curloc >= '0' && *curloc <= '9') {
   putdata(29,2);
   while(*curloc ==' ')curloc++;
if (*curloc =='#'){
    curloc++;
    n=getnum(); /* save value timmer to */
    if(n >= VALUE_TIM_OR_CNT) { /*Value over*/
      error_code(13);
      return(0);
      }
    curloc++;     /* buffer */
    x=div(n,255);
    n=x.quot;
```

```
      *code_data=n;
      code_data++;
      code_end++;
      n=x.rem;
      *code_data=n;
      code_data++;
      code_end++;
  }else {
        error_code(3);
        break;
        }
    }else {error_code(0);
 break;
 }
break;
     case 10:
while(*curloc ==' ' || *curloc =='\r'){
   if(*curloc=='\r' )linee++;
   curloc++;}
if (*curloc >= '0' && *curloc <= '9' ) {
   putdata(30,0);
   break;}
 else {
puttable();
a=compare_code(2);
switch (a) {                        /* Sub code */
  case 0:
     while(*curloc ==' ' || *curloc =='\r'){
       if(*curloc=='\r' )linee++;
       curloc++;}
```

```
      if (*curloc >='0' && *curloc <= '9') {
 putdata(31,1);
 break;
       }else error_code();
       break;
   }
    }
 break;
     case 11:
 while(*curloc ==' ' || *curloc =='\r'){
    if(*curloc=='\r' )linee++;
    curloc++; }
 if (*curloc >= '0' && *curloc <= '9' ) {
    putdata(32,0);
    break;}
 else {
       puttable();
       a=compare_code(2);
       switch (a) {                    /* Sub code */
 case 0:
    error_code(0);
    break;
}
  }
 break;
     case 12:
*code_data=0;
code_data++;
code_end++;
*code_data=0;
```

```
code_data++;
code_end++;
curloc++;
return(0);
  }
   while(*curloc ==' ' || *curloc =='\r'){
      if(*curloc=='\r' )linee++;
      curloc++;}
  }
  if(error_set==1) {
      clrscr();
      while(*curloc != '\r' && curloc !=buf) curloc--;
      if(*curloc=='\r' && curloc < endloc) curloc++;
      scrnx=START_X;scrny=TOPLINE;
      display_scrn(START_X,TOPLINE,curloc);
      putch(7);
 }else { error_code(4);
 scrnx=START_X;scrny=TOPLINE;
 error_set=1;
 curloc=endloc=buf;
 display_scrn(START_X,TOPLINE,curloc);
 }
 if((fp=fopen("plc.hex","wb"))==NULL)
 return(0);

 code_data=buf_code;
 while(code_data != code_end) {
   putc(*code_data,fp);
   code_data++;
 }
```

```
  fclose(fp);
/* Read string and put they in table */

void puttable(void)
{
  code_loc=name_code;
  if(curloc<endloc){
  while(*curloc ==' '|| *curloc =='\r') curloc++;
    do{
  *code_loc=*curloc;
  curloc++;
  code_loc++;
    }while(*curloc !='\r' && *curloc != ' '&& curloc < endloc);
   *code_loc ='\0';
   }
}

/* Compare code  from table */

compare_code(int code)
{
 int len;  /* lenght of string */
 int i,a;
    i=0;
    code_loc=name_code;
    len=strlen(code_loc);
    switch (code) {
   case 1:
  while (*code1 && strncmpi(code_loc,code1[i],len)
   && i<13)  i++;
```

```
if(i== 13) error_code(0);  /* not found */
      break;
   case 2:
 while (*code2 && strncmpi(code_loc,code2[i],len)
   && i < 4) i++;
 if(i==4) error_code(0);    /* not found */
 break;

}
  return(i);
}
```

This is source code PROGRAM PLCSIM.C

```
#include  "edit.h"

/*  THIS MAIN PROGRAM OF PLC SIMULATOR  */

main(int argc, char *argv[])
{
  int i;
clrscrn();
textattr(NORMAL);
if(argc==2){
    edit(argv[1]);
    textattr(NORMAL);
    clrscrn();
    exit(0);
}
edit(argv[1]);
textattr(NORMAL);
clrscrn();
}

/*  STACK function        */
/* It use push(value);  */

void push(int i)
{
 if(tos>=MAX) {  /* stack is full */
   error_code(5);
   error_set=1;
   return;
```

```
    }
    stack[tos]=i;
    tos++;
    }

    /*  STACK function        */
    /* It use pop();  */

    pop(void)
    {
     tos--;
     if(tos < 0){   /* stack is under */
     error_code(6);
     error_set=1;
      return 0;

}
return stack[tos];
 }

   /* LD DATA */

   int ld_sw()
   {
    int i,sw;
    i=0;
    sw=*code_data;
    i=i+sw;
    reg_a=code_sw[i];
    push(reg_a);
```

```
}

int ld_out()
{
 int i,sw;
 i=0;
 sw=*code_data;
 i=i+sw;
 reg_a=code_out[i];
 push(reg_a);
}

int ld_tr()
{
 int i,sw;
 i=0;
 sw=*code_data;
 i=i+sw;
 reg_a=code_tr[i];
 push(reg_a);
}

int ld_cnt()
{
 int i,sw;
 i=0;
 sw=*code_data;
 i=i+sw;
 reg_a=code_cnt[i];
```

```
 push(reg_a);
}

int ld_tim()
{
 int i,sw;
 i=0;
 sw=*code_data;
 i=i+sw;
 reg_a=code_tim[i];
 push(reg_a);
}

/*  LD_NOT DATA */

int ld_not_sw()
{
 int i,sw;
 i=0;
 sw=*code_data;
 i=i+sw;
 reg_a=code_sw[i];
 reg_a= !reg_a;
 push(reg_a);
}

int ld_not_out()
{
 int i,sw;
 i=0;
```

```
  sw=*code_data;
  i=i+sw;
  reg_a=code_sw[i];
  reg_a=!reg_a;
  push(reg_a);
 }

 int ld_not_cnt()
 {
  int i,sw;
  i=0;
  sw=*code_data;
  i=i+sw;
  reg_a=code_cnt[i];
  reg_a=!reg_a;
  push(reg_a);
 }

 int ld_not_tim()
 {
  int i,sw;
  i=0;
  sw=*code_data;
  i=i+sw;
  reg_a=code_tim[i];
  reg_a=!reg_a;
  push(reg_a);
 }
```



```
 /* AND DATA */
 int and_sw()
```

```
 {


  int reg_b,sw,i;
  i=0;
  reg_b=pop();
  sw=*code_data;
  i=i+sw;
  reg_a=code_sw[i];
  reg_a=reg_a && reg_b;
  push(reg_a);
  }


int and_out()
{


  int reg_b,sw,i;
  i=0;


  reg_b=pop();
  sw=*code_data;
  i=i+sw;
  reg_a=code_out[i];
  reg_a=reg_a && reg_b;
  push(reg_a);
  }


int  and_cnt()
{
```



```
  int reg_b,sw,i;
```

```
 i=0;
 reg_b=pop();
 sw=*code_data;
 i=i+sw;
 reg_a=code_sw[i];
 reg_a=reg_a && reg_b;
 push(reg_a);
 }

int and_tim()
{
 int reg_b,sw,i;
 i=0;
 reg_b=pop();
 sw=*code_data;
 i=i+sw;
 reg_a=code_tim[i];
 reg_a=reg_a && reg_b;
 push(reg_a);
 }

 /* AND_NOT DATA  */

int and_not_sw()
{

 int reg_b,sw,i;
 i=0;
 reg_b=pop();
 sw=*code_data;
```

```
 j=j+sw;
 reg_a=code_sw[i];
 reg_a=! reg_a;
 reg_a=reg_a && reg_b;
 push(reg_a);
 }

int and_not_out()
{

 int reg_b,sw,i;
 i=0;
 reg_b=pop();
 sw=*code_data;
 i=i+sw;
 reg_a=code_out[i];
 reg_a= !reg_a;
 reg_a=reg_a && reg_b;
 push(reg_a);
 }

int and_not_cnt()
{

 int reg_b,sw,i;
 i=0;
 reg_b=pop();
 sw=*code_data;
 i=i+sw;
 reg_a=code_cnt[i];
```

```
 reg_a=! reg_a;
 reg_a=reg_a && reg_b;
 push(reg_a);
 }

int and_not_tim()
{

 int reg_b,sw,i;
 i=0;
 reg_b=pop();
 sw=*code_data;
 i=i+sw;
 reg_a=code_tim[i];
 reg_a=!reg_a;
 reg_a=reg_a && reg_b;
 push(reg_a);
 }
/*  OR DATA    */

int or_sw()
{

 int reg_b,sw,i;
 i=0;
 reg_b=pop();
 sw=*code_data;
 i=i+sw;
 reg_a=code_sw[i];
 reg_a=reg_a || reg_b;
```

```
 push(reg_a);
 }

int or_out()
{

 int reg_b,sw,i;
 i=0;
 reg_b=pop();
 sw=*code_data;
i=i+sw;
reg_a=code_out[i];
reg_a=reg_a || reg_b;
push(reg_a);
}

int or_cnt()
{

 int reg_b,sw,i;
 i=0;
 reg_b=pop();
 sw=*code_data;
 i=i+sw;
 reg_a=code_cnt[i];
 reg_a=reg_a || reg_b;
 push(reg_a);
}
int or_tim()
```

```
{
 int reg_b,sw,i;
 i=0;
 reg_b=pop();
 sw=*code_data;
 i=i+sw;
 reg_a=code_tim[i];
 reg_a=reg_a || reg_b;
 push(reg_a);
 }

 /* OR_NOT DATA   */

int or_not_sw()
{

 int reg_b,sw,i;
 i=0;
 reg_b=pop();
 sw=*code_data;
 i=i+sw;
 reg_a=code_sw[i];
 reg_a= !reg_a;
 reg_a=reg_a || reg_b;
 push(reg_a);
 }

int or_not_out()
{
```

```
 int reg_b,sw,i;
 i=0;
 reg_b=pop();
 sw=*code_data;
 i=i+sw;
 reg_a=code_out[i];
 reg_a=!reg_a;
 reg_a=reg_a || reg_b;
 push(reg_a);
 }

int or_not_cnt()
{

 int reg_b,sw,i;
 i=0;
 reg_b=pop();
 sw=*code_data;
 i=i+sw;
 reg_a=code_cnt[i];
 reg_a= !reg_a;
 reg_a=reg_a || reg_b;
 push(reg_a);
 }

int or_not_tim()
{

 int reg_b,sw,i;
 i=0;
```

```
 reg_b=pop();
 sw=*code_data;
 i=i+sw;
 reg_a=code_tim[i];
 reg_a =!reg_a;
 reg_a=reg_a || reg_b;
 push(reg_a);
 }

/* AND-LD     */

int and_ld()
{

 int reg_b;
 reg_b=pop();
 reg_a=pop();
 reg_a=reg_a && reg_b;
 push(reg_a);
 }

 /*  OR-LD     */

int or_ld()
{

 int reg_b;
 reg_b=pop();
 reg_a=pop();
 reg_a=reg_a || reg_b;
```

```
 push(reg_a);
 }

 /*  OUT     */

int  out()
 {
 int reg_b,out,i;
i=0;
 out=*code_data;
 i=i+out;
 if(flag_out==0) reg_a=pop();
 code_out[i]=reg_a;
 code_data++;
 if(*code_data == 29 ||*code_data== 30 ||*code_data==32)
      flag_out=1;
 else flag_out=0;
 code_data--;
 }

 /*  OUT-NOT  */

int  out_not()
 {
 int reg_b,out,i;

 i=0;
 out=*code_data;
 i=i+out;
 if(flag_out==0) reg_a=pop();
```

```
 reg_a= ! reg_a;
 code_out[i]=reg_a;
 code_data++;
 if(*code_data == 29 ||*code_data== 30||*code_data==32)
      flag_out=1;
 else flag_out=0;
 code_data--;
 }

 /* OUT-TR    */

int  out_tr()
 {
 int reg_b,out,i;

 i=0;
 out=*code_data;
 i=i+out;
 code_tr[i]=reg_a;
 }

 /* COUNTER                        */
 /* Input use rising edge pulse  */
 /* Max_counter ==9999           */

int count()
{
  int n,reg_b,reg_c;
  int i,a,b;
  div_t x;
```

```
   i=*code_data;
   reg_a=pop();
   if(reg_a==0) {

   /*  Save value counter */

reg_c=input[i];
input[i]=reg_a;
reg_a=pop();
if(code_cnt[i]==1){
  code_data++;
  code_data++;
   return;}  /* input not change */
input[i]=reg_a;

/* Save input   */

reg_b=reg_a ^ reg_c;
if(reg_b == 1) {
   n=cnt[i] +1;
   cnt[i]=n;
code_data++;

/* get  value */

a=*code_data;
code_data++;
b=*code_data;
if(b<0) b=b+256;
```

```
a=a*255 + b;
x=div(n,2);
if(a ==x.quot ){       /* counter over */
     code_cnt[i] =1;
     cnt[i] = 0;
     }else;
} else{
   code_data++;
   code_data++;}

     }else {       /* input not change */
 reg_a=pop();
 code_data++;
 code_data++;
 code_cnt[i]=0;
 cnt[i]=0;
 input[i]=1;
 }
}

/*    TIMMER           */
/*    Max_time 9999    */
int tim()
{
   div_t x;
   struct time now ;
   char *t;
   int     a,v_sec,v_min,v_hr,n_sec,n_min,
 n_hr,sec,min,hr, n,min_store,
 hr_store,i,f;
```

```
 i=*code_data * 3;
 f=*code_data;
 if(flag_out==0)reg_a=pop();
 if(reg_a==1){
 if(flag_tim[f]==0 )
{
 sec=0;min=0;hr=0;
 code_data++;
 n=*code_data;
 code_data++;
 a=*code_data;
 if(a<0){
   a=a+256;
 }else;
 n=n*255 +a;
 gettime(&now);
 x=div(n,60);
 v_sec=x.rem;
 x=div(x.quot,60);
 v_min=x.rem;
 v_hr=x.quot;
 sec=now.ti_sec+v_sec;
 min_store=0;
 if(sec>59){x=div(sec,60);
  min_store=x.quot;
  sec=x.rem;}
 min=now.ti_min+v_min+min_store;
 hr_store=0;
```

```
/* Read file of timming */
code_data=buf_code;
draw_background();
print_guide_line(13);
edit_gets(name);
if(*name) load_input(name);
if(error_set==1) return(0);   /* Can not load input file */
initzial();
/* Distance of logic 0 */
/* Distane of logic 1 */
stx=maxX/24;
sty=maxY/20;              /* distance each input */
step=loop_count=0;
post_step=0;
logic=logic_in;      /* buffer of input timming */
clear_array();
/*  Fill input first step */
i=0;
   while( i < 8 ){
   if(*logic=='\r') logic++;
   if(*logic=='E') logic=logic_in;
   a=*logic;
   switch (a) {
  case '0':  /* Logic 0 */
   code_sw[i]=0;
   break;
   case '1':  /* logic 1 */
   code_sw[i]=1;
   break;
```

```
      /* Format of file not correct */
     default:
     error_set=1;
     break;
  }
  i++;
  logic++;

  }
  do{
      do{

sim_function();
if(*code_data==0)
  { code_data=buf_code;
    if(loop_count < loop_set) show_logic(loop_count);
    step++;
    loop_count++;
    i=0;
    while( i < 8 ){
    if(*logic=='\r') logic++;
    if(*logic=='E') logic=logic_in;
    a=*logic;
    switch (a) {
   case '0':  /* Logic 0 */
    code_sw[i]=0;
    break;
    case '1':  /* logic 1 */
    code_sw[i]=1;
    break;
```

```
 /* Look up subroutine of simulate */
 /* code_data => point to call subroutine */

 sim_function()
 {
  int x;
  static int (*funcP[32]) () ={  ld_sw,ld_out,ld_tr,ld_cnt,
 ld_tim,ld_not_sw,ld_not_out,
 ld_not_cnt,ld_not_tim,and_sw,
 and_out,and_cnt,and_tim,
 and_not_sw,and_not_out,
 and_not_cnt,and_not_tim,
 or_sw,or_out,or_cnt,or_tim,
 or_not_sw,or_not_out,
 or_not_cnt,or_not_tim,
 and_ld,or_ld,count,tim,out,
 out_tr,out_not };
  x=*code_data;
  x--;
  code_data++;
  (*funcP[x]) ();      /* pointer of subroutine */
  code_data++;
}
```

```
    }while(*code_data !=0);
    initzial();
    stx=maxX/8;
    /* Make back ground */
    fillscreen(0,0);
    outtextxy(maxX/2.5,5,"LADDER DIAGRAM");
    posx=stx/2;
    posy=30;
    code_data=buf_code;
    tos=0;
    flag_andld=1;
    set_exit=0;
    lad_line=0;
    save_line=flag_start=0;
    and_line=1;
    page=0;
    code_store=code_data;
    page_num[page]=code_store;
    do{
       posx=stx/2;
       posy=30;
       line(0,60,maxX,60);
       line(stx/2-1,60,stx/2-1,maxY);
       while( posy<   maxY && *code_data ){
  if(error_set==1) return(0);
  /* MaxY is over */
  if(exit_sub==1) flag_start=1;
  else flag_start=0;
  x=*code_data;
  switch (x) {
```

```
/* Upload file to controller    */
/* Communication port COM1      */

send_file()
{
  char *p;
  int count,i;

  flag_out=0;
  code_data=buf_code;
  clear_array();
  /* Test function error ?   */
  do{
      sim_function();
      if(error_set==1) return(0);
  }while(*code_data!=0);
  tos=0;  /* set printer stack equal first */
  print_guide_line=15;
  p=buf_code;
  count=0;

  /* Set Communication port */

  bioscom(0,SETTINGS.COM);
  /* Send  to controller        */
  /* Start controller to receive  */
  while(count!
bioscom(1,0x
if (kbhit()!
count++;
```

```
/* Ladder diagram function is show ladder diagram   */
/* form  toggle code.                               */
/* Each ladder  is display at position posx,posy .  */
/* When display each ladder it save position (posx  */
/* posy).                                           */
/* And-ld ,Or-ld in this function may be can not    */
/* display of ladder .                              */
/* lad_line is line of each brance .Use in or,or-not */
/* or-ld,and-ld,out,out-not.                        */
/* and_line is value use in and,and-not,and-ld      */
/* Other value see text                             */

ladder()
{
  int x,posx,posy,lad_line,old_posy,old_posx,flag_start;
  int scrx,tr_store[16],and_line;
  int page_num[PAGE],page,posy_over,save_line,tim_f,flag_andld;
  int set_exit;
  union k{
  char ch[2];
  unsigned i;
  }key;
  char  *code_store;
  flag_out=0;
  code_data=buf_code;
  clear_array();
  /* Test for program error */
  do{
    sim_function();
    if(error_set==1) return(0);
```

```
}
   /* Format ปปป */
  bioscom(1,0xFF,COM);
  while(p<code_end){
  bioscom(1,*p,COM);
  if (kbhit())  if((c=getch())=='\x1B') return(0);
  p++;
  }
  count=0;

  /*  Receive code from  controller */

  while(count < 3 ){bioscom(1,0x1b,COM);
    count++;
    }
   while(count < 30 && c!=0x7b && exit_sub==0) {
 c=rec();
 count++;
 }
   if(c==0x7b) {
print_guide_line(16);
playsound();
;}
   else {
putch(7);
print_guide_line(17);
}
exit_sub=0;
}
```

```
/* ***************************
   Recive data from RS-232
   ***************************  */
rec()
{    int register status;
     char out;


 while (exit_sub !=1)
     {
 status = bioscom(3,0,COM);
 if (status & DATA_READY)
   if( (out = bioscom(2,0,COM) & 0x7F) != 0)
return(out);
   if(kbhit()){
if(( getch()) == '\x1B')
exit_sub = 1; }
}
 }
playsound()
{
sound(1300);
delay(20);
sound(1000);
delay(5);
nosound();
}
```

กิตติกรรมประกาศ

การทำปริญญานิพนธ์นี้ สำเร็จลุล่วงไปได้ด้วยดีก็เพราะว่าได้รับคำแนะนำข้อมูลเกี่ยวกับเครื่องควบคุมแบบโปรแกรมได้จาก อาจารย์ สุพรรณ กุลพาณิชย์ ซึ่งเป็นที่ปรึกษาปริญญานิพนธ์ รวมทั้งอาจารย์ประจำภาคเทคโนโลยีการวัดคุมทางอุตสาหกรรม และเจ้าหน้าที่ที่เกี่ยวข้องที่ได้ให้ความร่วมมือ และ ความสะดวกในการใช้อุปกรณ์ เครื่องมือต่าง ๆ รวมถึงเพื่อนๆ ที่คอยให้ความร่วมมือให้คำปรึกษา จึงขอขอบพระคุณมา ณ.ที่นี้

นาย พิษณุ พร้อมวงษ์

นาย สมชาย ชาวเวียง

นาย ภัทร์ชนนท์ เกิดพิพัฒน์

21 ตุลาคม 2537

หนังสืออ้างอิง

1. ธวัชชัย ชยาวนิช, วารสารเซมิคอนดักเตอร์อิเลคทรอนิคส์,ฉบับที่ 118,2535,หน้า 86-93.

2. วิชัย ทิพย์มณี และ พรจิต ประทุมสุวรรณ. ทฤษฎีการใช้งานเบื้องต้น PROMMABLE LOGIC CONTROLLER. กรุงเทพมหานคร: สำนักพิมพ์เรือนแก้วการพิมพ์

3. วิบูลย์ ชื่นแขก, ไมโครโปรเซสเซอร์. กรุงเทพมหานคร: สำนักพิมพ์สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ.

4. ศิริจันทร์ ทองประเสริฐ, การจำลองแบบปัญหา SIMULATION. กรุงเทพมหานคร: สำนักพิมพ์จุฬาลงกรณ์มหาวิทยาลัย

5. สุเชียร เกียรติสุนทร, หลักการทำงานและเทคนิคการประยุกต์ใช้งาน PC/PLC: กรุงเทพมหานคร: สำนักพิมพ์เอช-เอน การพิมพ์,2533.

6. สุพรรณ กุลพาณิชย์, PROMGRAMMABLE CONTROLLER เทคนิคและการใช้งานเบื้องต้น. กรุงเทพมหานคร: โรงพิมพ์ ทิพยวิสุทธิ์,2531.

7. อุดม จีนประดับ, และวรพงษ์ รัตนรภคา, การทดลองไมโครโปรเซสเซอร์ 2. กรุงเทพมหานคร: สำนักพิมพ์สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ,2533.

8. Borland International Inc,TURBO C Reference guide and User Guide Version 2 ,1987

8.Schildt M. ,Born to code in C. Berkely ,CA: Osborn McGraw-Hill,1989.