เครื่องควบคุมที่โปรแกรมได้

PROGRAMMABLE CONTROLLER

วิริยะ กองรัตน์

VIRIYA KONGRATANA

วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2531

เครื่องควบคุมที่โปรแกรมได้

PROGRAMMABLE CONTROLLER

วิริยะ กองรัตน์

VIRIYA KONGRATANA

อาจารย์ที่ปรึกษา

รองศาสตราจารย์ กิตติ ตีรเศรษฐ

ADVISOR

Assc.Prof.KITTI TIRASESTH

วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิศวกรรมไฟฟ้า

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2531

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

แบบฟอร์มการให้คะแนนการสอบวิทยานิพนธ์

สำหรับนักศึกษาระดับมหาบัณฑิต

ชื่อนักศึกษา นายวิริยะ กองรัตน์ เลขประจำตัว 29.0021

ชื่อเรื่องวิทยานิพนธ์ เครื่องควบคุมที่โปรแกรมได้

(Programmable Controller)

ชื่ออาจารย์ผู้ควบคุมการสอบ		ลายมือชื่อ	ผลการสอบ
รศ.กิตติ	ตีระเศรษฐ์		ดี
รศ.ดร.โยธิน	เปรมประพีรัชต์		ดี
ศ.ดร.ไพรัช	ธัชยพงษ์		ดี
ผศ.ดร.จงกล	งามวิวิทย์		ดี

วันเดือนปี ที่สอบ 31 ตุลาคม 2531 เวลา 13.00 น. สถานที่ ห้อง A-305

(นายสิทธิชัย [illegible])

คณบดีบัณฑิตวิทยาลัย

สารบัญ

หัวข้อวิทยานิพนธ์ เครื่องควบคุมที่โปรแกรมได้
นักศึกษา นายวิริยะ กองรัตน์ 29.0021
อาจารย์ที่ปรึกษา รศ. กิตติ ตีรเศรษฐ
ระดับการศึกษา วิศวกรรมศาสตรมหาบัณฑิตทางวิศวกรรมไฟฟ้า
ปีการศึกษา 2531

บทคัดย่อ

การใช้อุปกรณ์ไฟฟ้าเชิงกล ในการควบคุมกระบวนการทางอุตสาหกรรมจะสิ้นเปลืองกำลังสูง มีขนาดใหญ่ และความยืดหยุ่นของระบบต่ำ วิทยานิพนธ์ฉบับนี้ขอเสนอการพัฒนาออกแบบสร้างเครื่องควบคุมที่โปรแกรมได้ โดยใช้ไมโครโปรเซสเซอร์เป็นตัวจัดการ ซึ่งจะทำให้เครื่องควบคุมมีประสิทธิภาพสูง สิ้นเปลืองกำลังงานต่ำ และใช้เครื่องคอมพิวเตอร์เป็นส่วนป้อนโปรแกรมคำสั่งแบบแลดเดอร์ไดอะแกรมและโปรแกรมคำสั่งแบบบูลลีน โดยมีการติดต่อส่งผ่านข้อมูลทางพอร์ทอนุกรมแบบ RS-232C สามารถโปรแกรมเปลี่ยนแปลงเงื่อนไขการทำงาน ตรวจสอบสภาวะการทำงานของกระบวนการได้อย่างตลอดเวลาบนจอภาพแสดงผลของเครื่องคอมพิวเตอร์ ข้อดีอีกอย่างหนึ่งก็คือ เมื่อเสร็จสิ้นการโปรแกรมการทำงานหรือไม่ต้องการแสดงผลค่าสภาวะของกระบวนการแล้ว เครื่องคอมพิวเตอร์ดังกล่าวก็สามารถนำไปใช้ในงานอื่น ๆ ได้

Thesis Title PROGRAMMABLE CONTROLLER
Name VIRIYA KONGRATANA
Thesis Advisor Assc. Prof. KITTI TIRASESTH
Level of Study MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING
Academic Year 1988

ABSTRACT

The electromachanical devices used in process control usually consume much power,larger size of control system and less flexibility to change the operation. This thesis presents a method of design and develop a programmable controller using microprocessor as a processing unit. The instruction program of programmable controller in ladder diagram and/or boolean instruction will be programmed through the computer by means of RS-232C. The designed and developed programmable controller consume low power,high efficiency as well as high flexibility when the change of operation condition of the process control is required,and can be used with the another process also. Apart from this,the another remarkably advantage of this programmable controller is that all of process variable can be monitored through the CRT of the computer. Moreover, the computer also can be used for the other purposed when instruction has been done or no process variables monitoriong is required.

บทที่ 1
บทนำ

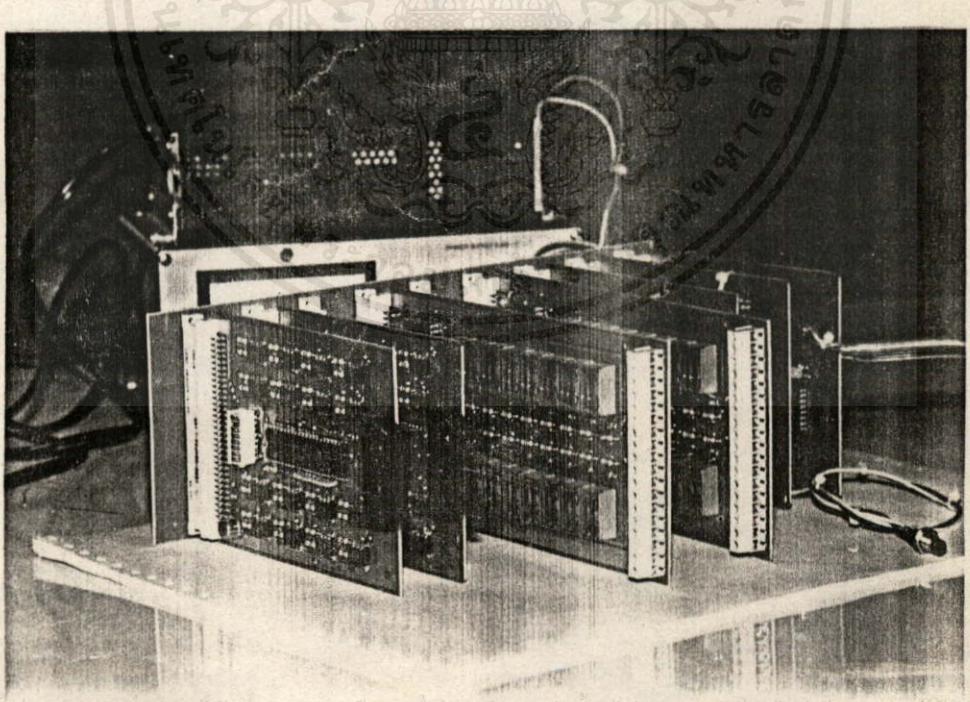
การควบคุมกระบวนการทางอุตสาหกรรมส่วนใหญ่จะเป็นการนำอุปกรณ์ไฟฟ้าเชิงกล เช่น รีเลย์ (RELAY) ตัวนับ (COUNTER) ตัวตั้งเวลา (TIMER) มาต่อร่วมกันเป็นระบบควบคุม แต่การใช้วิธีนี้จะมีข้อเสียมาก เนื่องจากอุปกรณ์ไฟฟ้าเชิงกลจะสิ้นเปลืองกำลังงานสูง และมีอายุการใช้งานค่อนข้างสั้น นอกจากนั้นเมื่อต้องการเปลี่ยนแปลงลำดับการทำงานของกระบวนการก็จะต้องแก้ไขระบบควบคุมใหม่ทำให้ยุ่งยากและเสียเวลามาก หรือในกรณีที่มีการเปลี่ยนแปลงมาก ๆ อาจจะต้องออกแบบระบบควบคุมใหม่ทั้งหมด

การประยุกต์ใช้ไมโครโปรเซสเซอร์และไมโครคอมพิวเตอร์มาใช้ในการควบคุมกระบวนการเป็นวิธีหนึ่งที่มีประสิทธิภาพสูง เนื่องจากสามารถเปลี่ยนแปลงลำดับขั้นการทำงานหรือการควบคุมได้โดยการเปลี่ยนแปลงหรือแก้ไขโปรแกรม ดังนั้นในปัจจุบันจึงมีการใช้ไมโครโปรเซสเซอร์หรือไมโครคอมพิวเตอร์ ในการควบคุมกระบวนการกันอย่างแพร่หลาย อย่างไรก็ดีการแก้ไขหรือเปลี่ยนแปลงโปรแกรมนั้น จะต้องกระทำโดยผู้ที่เข้าใจโครงสร้างทางฮาร์ดแวร์ (HARDWARE) ของระบบควบคุมหรือกระบวนการเป็นอย่างดี ดังนั้นผู้ปฏิบัติงานหรือผู้ใช้มักจะไม่สามารถแก้ไขหรือเปลี่ยนแปลงโปรแกรมดังกล่าวได้เอง ดังนั้นจึงได้มีการพัฒนาเครื่องควบคุมที่โปรแกรมได้ (PROGRAMMABLE CONTROLLER) ขึ้น ซึ่งผู้ปฏิบัติงานหรือผู้ใช้สามารถที่จะเปลี่ยนแปลงหรือแก้ไขโปรแกรมการควบคุมได้เองโดยใช้ภาษาที่กำหนด เครื่องควบคุมที่โปรแกรมได้ส่วนใหญ่จะใช้ภาษาสำหรับทำการโปรแกรม เช่นภาษาโปรแกรมแบบแลดเดอร์ไดอะแกรม โปรแกรมแบบบูลลีน หรือภาษาชั้นสูงต่างๆ

วิทยานิพนธ์ฉบับนี้ได้นำเสนอเครื่องควบคุมที่โปรแกรมได้ ที่สามารถโปรแกรมได้ทั้งแบบแลดเดอร์ไดอะแกรมและแบบบูลลีน การโปรแกรมแบบแลดเดอร์ไดอะแกรมและการโปรแกรมแบบบูลลีน จะใช้เครื่องคอมพิวเตอร์ในการสร้างและแก้ไขการโปรแกรมจากนั้นแปลให้เป็นชุดคำสั่งของเครื่องควบคุมที่โปรแกรมได้ สำหรับการโปรแกรมแบบบูลลีนนั้นจะใช้วิธีสร้างภาพของปุ่มกด ที่มีลักษณะคล้ายกับแผงโปรแกรม (PROGRAMMING CONSOLE) ขึ้นมาบนจอภาพแล้วใช้ปากกาแสง (LIGHT PEN) ในการควบคุมการโปรแกรมการใช้วิธีนี้ทำให้ผู้ใช้หรือผู้ปฏิบัติงานสามารถใช้งานหรือสร้างโปรแกรมได้อย่างรวดเร็ว แต่ในขณะเดียวกันก็สามารถใช้แป้นพิมพ์ในการสร้างหรือแก้ไขโปรแกรมได้เช่นเดียวกัน จากนั้นชุด

คำสั่งหรือข้อมูลที่ได้จากการสร้างและแก้ไขแล้ว จะถูกส่งให้กับ เครื่องควบคุมทางพอร์ทอนุกรมแบบ RS-232C มีการแสดงผลและติดตามการ เปลี่ยนแปลงค่าสภาวะ เมื่อกระทำการควบคุมบนจอภาพแสดงผล ที่เป็นทั้งตารางแสดงค่าสภาวะและแบบแลดเดอร์ไดอะแกรม

สำหรับภายในวิทยานิพนธ์ในบทที่ 2 จะได้กล่าวถึงทฤษฎีและหลักการทั่ว ๆ ไปของเครื่องควบคุมที่โปรแกรมได้ บทที่ 3 กล่าวถึงโครงสร้างของ เครื่องควบคุมที่ประกอบขึ้นเป็นระบบคอมพิวเตอร์ บทที่ 4 กล่าวถึงระบบการจัดการของ เครื่องควบคุมที่จะต้องนำประกอบร่วมกับโครงสร้างในบทที่ 3 บทที่ 5 เป็นการประยุกต์ใช้ เครื่องคอมพิวเตอร์กับเครื่องควบคุม เพื่อทำให้ เครื่องควบคุมสามารถที่จะทำการโปรแกรม เงื่อนไขการทำงานโดยใช้ภาษาสำหรับโปรแกรมแบบบูลลีนและแบบแลดเดอร์ไดอะแกรม บทที่ 6 กล่าวถึงการแปลความหมายโปรแกรมแบบแลดเดอร์ไดอะแกรมให้เป็นรหัสคำสั่ง เพื่อส่งให้ เครื่องควบคุมรับไปปฏิบัติ บทที่ 7 กล่าวถึงการตรวจสอบและติดตามการทำงานของ เครื่องควบคุม เพื่อแสดงค่าสภาวะการทำงานของ เครื่องควบคุมขณะทำงาน บทที่ 8 เป็นตัวอย่างและผลการทดลอง ส่วนบทที่ 9 จะ เป็นบทสรุป



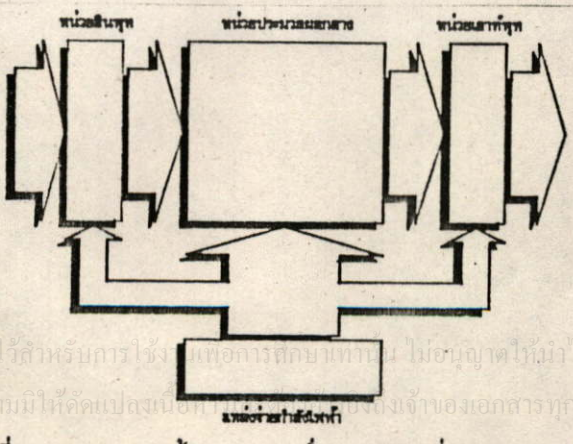
รูปที่ 1.1 แสดง เครื่องควบคุมที่ทำการทดลอง

บทที่ 2
ทฤษฎีทั่วไป เกี่ยวกับ เครื่องควบคุมที่โปรแกรมได้

เครื่องควบคุมที่โปรแกรมได้ เป็นการประยุกต์ใช้ เทคโนโลยีทางด้านไมโครโปร-เซส เซอร์ทั้งทางด้านฮาร์ดแวร์ (HARDWARE) และซอฟท์แวร์ (SOFTWARE) มาใช้ในการควบคุม เครื่องจักรอัตโนมัติ หรือกระบวนการทางอุตสาหกรรม ทำให้การควบคุมกระบวนการหรือ เครื่องจักรต่าง ๆ เหล่านั้น เป็นไปอย่างมีประสิทธิภาพ นอกจากนั้นยังสามารถเปลี่ยนแปลงรูปแบบของการควบคุมและสามารถพัฒนาขีดความสามารถให้สูงขึ้นได้

2.1 หลักการทำงานของ เครื่องควบคุมที่โปรแกรมได้

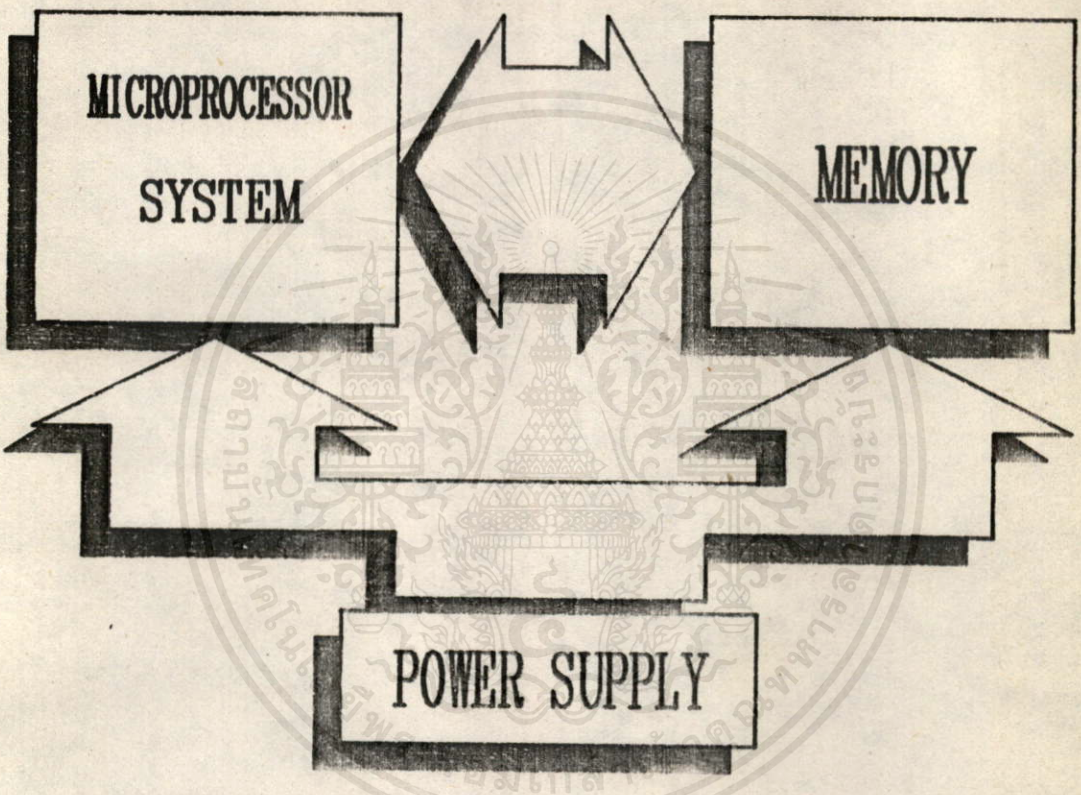
เครื่องควบคุมที่โปรแกรมได้ เป็นระบบคอมพิว เตอร์ที่อินพุทและ เอาท์พุทได้ถูกออกแบบให้มีสภาวะ เหมาะสมกับกระบวนการที่ต้องการควบคุม และมีหน่วยประมวลผลกลางที่ทำหน้าที่ควบคุมการทำงานของระบบทั้งหมด หน่วยประมวลผลกลางหมายถึงระบบไมโครโปร เซส เซอร์ที่ทำหน้าที่ควบคุมการทำงานทั้งหมด โปรแกรมที่ใช้ควบคุมการทำงานของหน่วยประมวลผลกลางดังกล่าวจะถูกแบ่งออก เป็น 2 ส่วน ส่วนหนึ่งคือโปรแกรมที่ใช้ในการกำหนดลำดับการทำงานของการควบคุม ซึ่งผู้ใช้สามารถที่จะ เปลี่ยนแปลงหรือแก้ไขโปร-แกรมในส่วนนี้ได้ ซึ่งโปรแกรมในส่วนนี้ก็คือโปรแกรมที่บ่งบอก เงื่อนไขต่าง ๆ ที่ใช้สำหรับการควบคุม โปรแกรมอีกส่วนหนึ่งคือโปรแกรมที่ใช้ในการควบคุมระบบ โปรแกรมส่วนนี้จะทำหน้าที่ควบคุมการรับค่าสภาวะอินพุท ซึ่งได้มาจากกระบวนการแล้วนำมาประมวลผลทาง ลอจิก คำนวณทางคณิตศาสตร์ เพื่อให้ข้อมูลอยู่ในรูปแบบที่ เหมาะสมและส่งค่าสภาวะออกทาง เอาท์พุท ตามลำดับขั้นตอนของโปรแกรมในส่วนแรก โครงสร้างของ เครื่องควบคุม แสดงดังรูปที่ 2.1



รูปที่ 2.1 โครงสร้างของ เครื่องควบคุมที่โปรแกรมได้

2.1.1 หน่วยประมวลผลกลาง (CENTRAL PROCESSING UNIT : CPU)

หน่วยประมวลผลกลางประกอบด้วย ระบบไมโครโปรเซสเซอร์ซึ่งทำหน้าที่ในการประมวลผลข้อมูล หน่วยความจำสำหรับเก็บโปรแกรมของผู้ใช้ และหน่วยจ่ายกำลังไฟฟ้า ดังรูปที่ 2.2



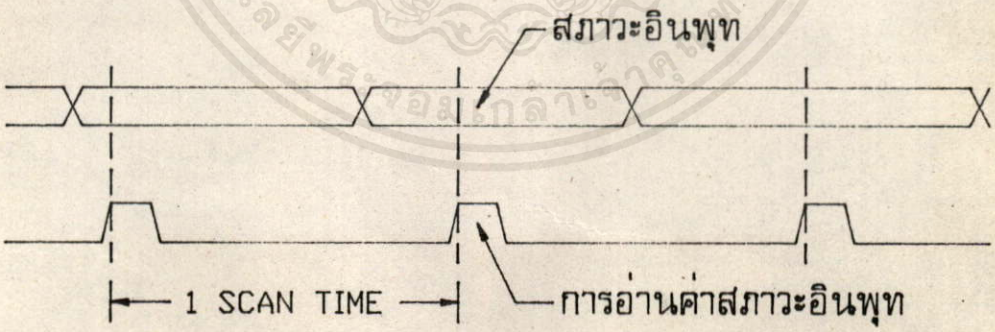
(CPU.DWG)

รูปที่ 2.2 โครงสร้างของหน่วยประมวลผลกลาง

ในเครื่องควบคุมบางแบบอาจจะรวมหน่วยประมวลผลกลาง หน่วยความจำและหน่วยจ่ายกำลังไฟฟ้าเข้าไว้ด้วยกัน เพื่อให้มีขนาดเล็กกระทัดรัด แต่บางแบบก็อาจจะแยกกันออกเป็นส่วน ๆ เพื่อสะดวกในการบำรุงรักษา ระบบไมโครโปรเซสเซอร์ซึ่งทำหน้าที่ในการประมวลผลเป็นส่วนประกอบที่สำคัญของเครื่องควบคุมมีหน้าที่ในการประมวลผลข้อมูล การคำนวณทางคณิตศาสตร์และทางลอจิก รวมทั้งควบคุมการทำงานของส่วนต่าง ๆ ทั้งหมด

ในปัจจุบันเทคโนโลยีทางด้านไมโครโปรเซสเซอร์ได้พัฒนาขึ้นมาก จึงได้มีการนำเอาไมโครโปรเซสเซอร์หลาย ๆ ตัวมาทำงานร่วมกัน (MULTI-PROCESSOR) เพื่อให้มีการประมวลผลได้รวดเร็วและมีประสิทธิภาพยิ่งขึ้น โดยแยกการทำงานของไมโครโปร-เซสเซอร์แต่ละตัวเป็นอิสระต่อกันแต่มีการแลกเปลี่ยนข้อมูลกันอยู่ตลอดเวลา นอกจากนี้ในหน่วยอินพุทหรือเอาท์พุทบางชนิดก็จะมีไมโครโปรเซสเซอร์แยกอิสระจากเครื่องควบคุมอีกทีหนึ่ง เช่น หน่วยอินพุทเอาท์พุทแบบ PID ซึ่งสามารถทำงานได้ด้วยตัวเอง

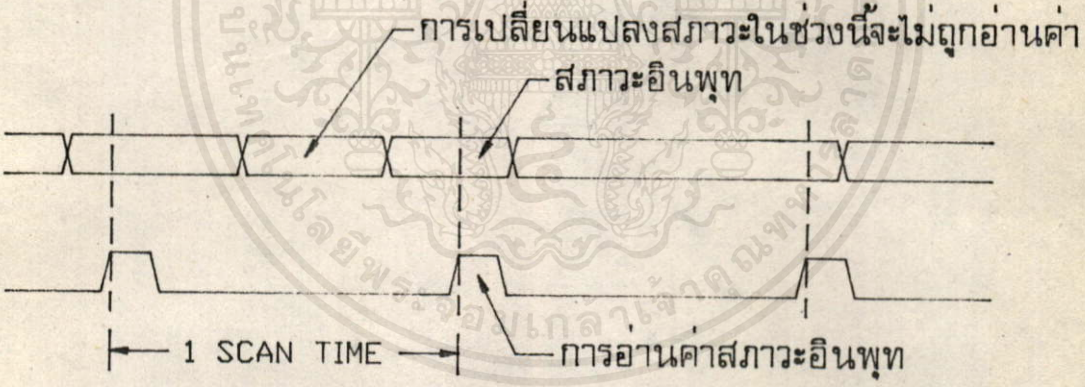
ค่าสภาวะต่าง ๆ ถูกนำมาจากกระบวนการโดยผ่านทางหน่วยอินพุท ผ่านการประ-มวลผลจากนั้นผลที่ได้จะถูกส่งออกทางเอาท์พุท เพื่อไปควบคุมการทำงานของกระบวนการตามต้องการ และจะย้อนกลับมารับค่าสภาวะจากกระบวนการเพื่อนำไปประมวลผลและส่งผลออกทางเอาท์พุทอีก จะมีการทำอย่างนี้ตลอดไป การรับค่าสภาวะจากภายนอกมาทำการประมวลผลและส่งค่าออกทางเอาท์พุทในแต่ละครั้งเรียกว่าการสแกน (SCANNING) หรือหนึ่งรอบของการทำงาน ระยะเวลาในการสแกนหนึ่งรอบ (1 SCAN TIME) จะมีค่าเท่ากับระยะเวลาของการอ่านค่าสภาวะอินพุทครั้งแรกจนกระทั่งมีการอ่านค่าสภาวะอินพุทครั้งต่อไป ดังรูปที่ 2.3



(SCANTIME.DWG)

รูปที่ 2.3 ระยะเวลาการสแกนหนึ่งรอบ

ระยะ เวลาในการสแกนจะขึ้นอยู่กับความยาวของโปรแกรมที่ผู้ใช้งานโปรแกรม เงื่อนไขการทำงาน เข้าไป เพื่อควบคุม ซึ่งผู้ผลิตมักจะกำหนดไว้ว่า เครื่องควบคุมนั้น ๆ สามารถทำโปรแกรมได้สูงสุด เพียงใด และการใช้ชุดคำสั่งในการโปรแกรมนั้นแต่ละคำสั่งใช้ เวลาในการประมวลผล เท่าใด นอกจากนั้นก็จะขึ้นอยู่กับสมรรถนะของ เครื่องควบคุม เอง รวมทั้งอุปกรณ์ภายนอกที่ติดต่ออยู่กับ เครื่องควบคุม ดังนั้นระยะ เวลาในการสแกนหนึ่งรอบจึง เป็นตัวประกอบส่วนหนึ่งที่จะใช้ในการตรวจสอบประสิทธิภาพของ เครื่องควบคุม เนื่องจากการควบคุมกระบวนการจะต้องกระทำได้อย่างรวด เร็วและแม่นยำ อย่างไรก็ตามถ้า เครื่องควบคุมมีค่าระยะ เวลาการสแกนที่มาก เกินไป อาจจะทำให้การควบคุม เกิดการผิดพลาดขึ้นได้ เพราะว่าค่าสภาวะบางค่าอาจจะมีการ เปลี่ยนแปลงที่ เร็วมาก ๆ ซึ่งถ้าการ เปลี่ยนแปลงนี้มีค่า เวลาที่สั้นกว่าค่าระยะ เวลาของการสแกนแล้วจะทำให้ข้อมูลที่ได้รับ เพื่อใช้ในการประมวลผลผิดพลาดได้ ดังรูปที่ 2.4



(MISSING.DWG)

รูปที่ 2.4 ความผิดพลาดที่ เกิดขึ้น เมื่อค่าสภาวะอินพุท
มีระยะ เวลาการ เปลี่ยนแปลงสั้นกว่าระยะ เวลาการสแกน

แต่ในทางปฏิบัติแล้ว การเปลี่ยนแปลงค่าสภาวะอินพุทของกระบวนการมักจะมีระยะเวลาที่ยาวกว่าระยะเวลาการสแกนของเครื่องควบคุม เนื่องจากอุปกรณ์ที่เป็นอินพุทส่วนใหญ่จะเป็นอุปกรณ์ไฟฟ้าเชิงกล หรือสวิทช์ชนิดต่าง ๆ ซึ่งมีค่าระยะเวลาการเปลี่ยนแปลงที่ค่อนข้างมาก นอกจากจะเป็นค่าสภาวะที่ได้รับมาจากอุปกรณ์อิเลคทรอนิคส์ซึ่งมีความเร็วในการสวิทช์สูง ในการใช้งานลักษณะนี้เครื่องควบคุมจำเป็นจะต้องมีอุปกรณ์พิเศษเพื่อช่วยให้การควบคุมเป็นไปอย่างถูกต้อง ซึ่งบางแบบอาจจะใช้ฮาร์ดแวร์เข้าช่วยในการรับค่าสภาวะมาทำการประมวลผลขั้นหนึ่งก่อน บางแบบอาจจะใช้วิธีอินเตอร์รัพท์ (INTERRUPT) เพื่อให้เครื่องควบคุมมาอ่านข้อมูลไปประมวลผลทันที หรือนำข้อมูลส่งออกไปเอาท์พุทในทันที เป็นต้น

หน่วยความจำในหน่วยประมวลผลกลางจะถูกแบ่งออกเป็น 2 ส่วนใหญ่ ๆ คือ ส่วนแรกใช้ในการเก็บโปรแกรมและข้อมูลสำหรับการควบคุมระบบ และอีกส่วนหนึ่งจะใช้ในการเก็บโปรแกรมของผู้ใช้ และข้อมูลสภาวะอินพุทเอาท์พุท ในเครื่องควบคุมจะมีหน่วยความจำอยู่จำกัดและไม่สามารถขยายได้เหมือนกับคอมพิวเตอร์ เครื่องควบคุมขนาดใหญ่จะมีหน่วยความจำที่มีความจุสูง แต่จะมีราคาแพงและเครื่องควบคุมขนาดเล็กก็จะมีความจุของหน่วยความจำต่ำซึ่งอาจจะไม่เพียงพอต่อการใช้งาน ดังนั้นการเลือกใช้เครื่องควบคุมจะต้องคำนึงถึงสิ่งเหล่านี้ด้วย

การจัดหน่วยพื้นที่ความจำของหน่วยประมวลผลกลางจะแบ่งออกเป็น 4 ส่วน ดังนี้

1. โปรแกรมบริหารงานระบบ (OPERATING SYSTEM)
2. พื้นที่ข้อมูลสำหรับประมวลผล (PROCESSOR WORKING AREA)
3. ตารางข้อมูล (DATA TABLE)
4. โปรแกรมผู้ใช้ (USER PROGRAM)

พื้นที่หน่วยความจำในส่วนที่ 1 และ 2 จะใช้วางโปรแกรมและข้อมูลสำหรับการควบคุมระบบและเป็นพื้นที่ใช้งานของไมโครโปรเซสเซอร์ พื้นที่ในส่วนที่ 3 และ 4 เป็นส่วนที่ผู้ใช้สามารถเข้าถึงได้ทั้งการอ่านและการเขียน สำหรับส่วนที่ 3 เป็นตารางข้อมูลซึ่งเก็บค่าอินพุทเอาท์พุท รีเลย์ภายใน (INTERNAL RELAY) และค่ารีจิสเตอร์ (REGISTER) และส่วนที่ 4 เป็นโปรแกรมผู้ใช้ซึ่งเป็นชุดคำสั่งสำหรับการควบคุม

การเก็บข้อมูลอินพุทจะเก็บอยู่ในลักษณะเป็นตาราง (TABLE) มีขนาดเท่ากับขนาดของหน่วยอินพุท เช่น เครื่องควบคุมที่มีอินพุทสูงสุด 64 อินพุทก็จะมีตารางของอินพุทที่มีขนาด

64 ช่องและค่าในตารางนี้ก็จะมีค่าเช่นเดียวกับค่าสภาวะที่เครื่องควบคุมอ่านเข้ามาได้ ตารางเอาท์พุทก็มีลักษณะเช่นเดียวกัน สำหรับตารางรีเลย์ภายในนั้นเป็นเอาท์พุทแต่ไม่มีการเชื่อมต่อกับอุปกรณ์ภายนอก เนื่องจากเป็นตำแหน่งในหน่วยความจำเท่านั้นซึ่งจะใช้ในการเก็บค่าสภาวะต่าง ๆ ชั่วคราว ส่วนตารางรีจิสเตอร์นั้นใช้ในการเก็บค่าและค่าสภาวะของตัวตั้งเวลา ตัวนับ และฟังก์ชั่นพิเศษต่าง ๆ

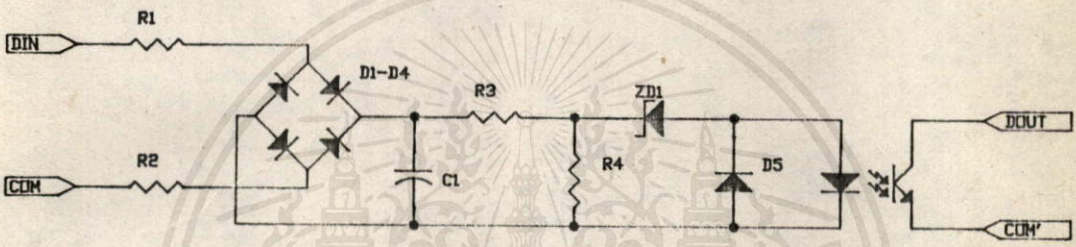
โปรแกรมผู้ใช้เป็นโปรแกรมที่ผู้ใช้เขียนขึ้น เพื่อให้เครื่องควบคุมมีการทำงานตามที่กำหนดโดยมีเงื่อนไขตามสภาวะของอินพุท โปรแกรมนี้จะสามารถเปลี่ยนแปลงได้โดยผู้ใช้เป็นผู้กำหนด สำหรับในกรณีที่ไม่ต้องการเปลี่ยนแปลงโปรแกรมบ่อย ๆ ก็อาจจะใช้วิธีเก็บโปรแกรมดังกล่าวลงในรอม (ROM : READ ONLY MEMORY) ก็ได้

สำหรับแหล่งจ่ายกำลังไฟฟ้าทำหน้าที่จ่ายกระแสไฟฟ้าให้กับส่วนต่าง ๆ ของเครื่องควบคุม แหล่งจ่ายกำลังไฟฟ้าจะต้องถูกออกแบบเป็นพิเศษเพื่อให้จ่ายแรงดันได้คงที่ตลอดช่วงการทำงานแม้ว่าแรงดันทางด้านอินพุทจะมีการเปลี่ยนแปลงก็ตาม และต้องสามารถที่จะป้องกัน สัญญาณรบกวนที่เข้ามาได้เป็นอย่างดี เนื่องจากเครื่องควบคุมส่วนใหญ่จะนำไปใช้กับกระบวนการทางอุตสาหกรรมซึ่งมีสัญญาณรบกวนในสายส่งกำลังเป็นอันมาก และในกรณีที่มีสัญญาณรบกวนในสายส่งกำลังสูงหรือแรงดันไม่คงที่ หน่วยจ่ายกำลังไฟฟ้าจะไม่สามารถทำงานได้ตามปกติ จึงจำเป็นจะต้องมีการปรับสภาพของแรงดันไฟฟ้าให้คงที่เสียก่อน ส่วนอุปกรณ์ที่ใช้ในการปรับแรงดันไฟฟ้า หรือลดสัญญาณรบกวนในลักษณะนี้มีอยู่ด้วยกันหลายชนิด บางชนิดสามารถลดหรือกำจัดสัญญาณรบกวนได้เพียงอย่างเดียว บางชนิดก็สามารถปรับแรงดันไฟฟ้าให้คงที่ได้ด้วย หรือบางชนิดอาจจะมีคุณสมบัติพิเศษในการกำจัดฮาร์โมนิค (HARMONIC) ได้เป็นต้น

2.1.2 หน่วยอินพุท เอาท์พุท (INPUT / OUTPUT UNIT)

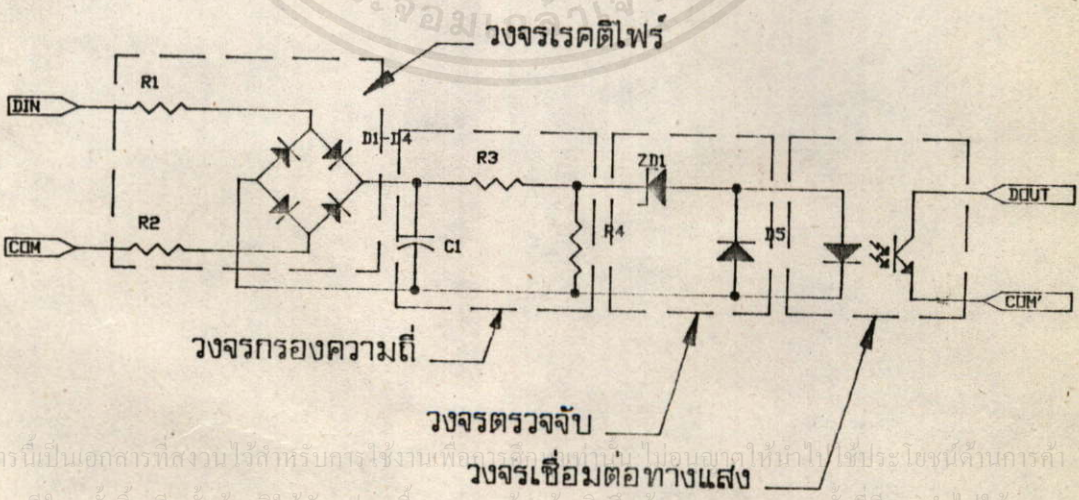
ส่วนนี้จะทำหน้าที่ติดต่อระหว่างเครื่องควบคุมกับกระบวนการ ซึ่งหน่วยอินพุทจะทำหน้าที่รับค่าสภาวะมาจากกระบวนการ เช่น สวิทช์ (SWITCH) ชนิดต่าง ๆ อุปกรณ์ตรวจจับ (SENSOR) และอุปกรณ์วัด ซึ่งค่าสภาวะต่าง ๆ เหล่านี้แบ่งออกเป็นแบบดิจิตอล (DIGITAL) และแบบอนาลอก (ANALOG) ทั้งสองชนิดก็มีการออกแบบพิเศษสำหรับงานเฉพาะอย่างอีกมาก ซึ่งการเลือกใช้หน่วยอินพุทหรือเอาท์พุทก็จะต้องพิจารณาจากรูปแบบของข้อมูลหรือสภาวะที่ต้องการ การเลือกใช้หน่วยอินพุทเอาท์พุทที่เหมาะสมจะทำให้การควบคุมเป็นไปอย่างมีประสิทธิภาพ

หน่วยอินพุท เอาท์พุทแบบดิจิตอล ทำหน้าที่ติดต่อกับอุปกรณ์ภายนอกที่มีการ เปลี่ยนแปลงสภาวะ เพียง 2 สภาวะ เท่านั้น คือ 0 และ 1 อุปกรณ์ที่มีค่าสภาวะ เหล่านี้ เช่น สวิทช์ชนิดต่าง ๆ อุปกรณ์สวิทชิ่งทางอิเลคทรอนิคส์ หรือในกรณีที่ เป็น เอาท์พุทก็จะ เป็นพวก มอเตอร์ โซลินอยด์วาล์ว (SOLINOID VALVE) หลอดไฟฟ้า เป็นต้น สำหรับระดับสัญญาณ ของหน่วยอินพุท เอาท์พุทแบบดิจิตอลก็มักจะอยู่ในช่วง 0 - 24 โวลท์ หรือ 0 - 220 โวลท์ หรืออาจจะ เป็นระดับแรงดันทีทีแอล (TTL) คือ 0 - 5 โวลท์ก็ได้ ซึ่งการกำหนด ระดับการ เปลี่ยนแปลงของลอจิก 0 และ 1 ก็ เป็นไปตามการออกแบบของหน่วยอินพุท เอาท์พุทนั้น ๆ ดังตัวอย่างวงจรในรูปที่ 2.5



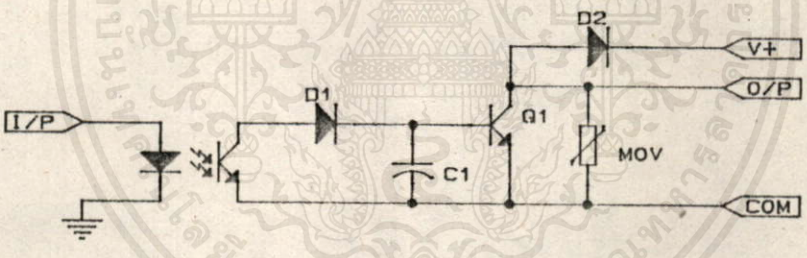
รูปที่ 2.5 วงจรอินพุทแบบดิจิตอล

จากวงจรในรูปที่ 2.5 เป็นวงจรพื้นฐานที่ใช้กันโดยทั่วไป วงจรดังกล่าวสามารถรับอินพุทได้ทั้งไฟสลับและไฟตรงส่วนขนาดแรงดันสูงสุดขึ้นอยู่กับการออกแบบ หลักการทำงานโดยทั่วไปอธิบายได้ตามผังภาพ (BLOCK DIAGRAM) ในรูปที่ 2.6

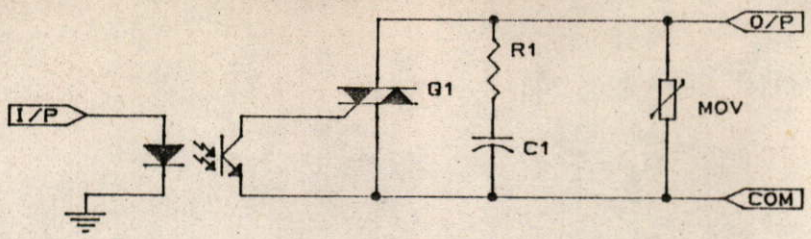


รูปที่ 2.6 ผังภาพของวงจรในรูปที่ 2.5

วงจรเรคติไฟร์ (RECTIFIER) D_1 - D_4 ทำหน้าที่เปลี่ยนสัญญาณไฟสลับให้เป็นไฟตรงหรือในกรณีที่เป็นไฟตรงก็สามารถผ่านได้ทันที ความต้านทาน R_1 และ R_2 ทำหน้าที่ในการลดแรงดันค่าของความต้านทานขึ้นอยู่กับแรงดันอินพุทสูงสุด ส่วนวงจรกรองความถี่ทำหน้าที่กรองสัญญาณรบกวนซึ่งอาจจะทำให้วงจรทำงานผิดพลาดได้ออก วงจรตรวจจับทำหน้าที่เปรียบเทียบระดับแรงดันเพื่อกำหนดสภาวะในการเป็น 0 หรือ 1 ค่าสภาวะดังกล่าวจะเป็นสภาวะในแบบดิจิตอลแล้ว จากนั้นก็จะส่งค่าสภาวะนี้ไปยังเครื่องควบคุมแต่จะใช้การเชื่อมต่อกันทางแสง (OPTICAL ISOLATOR) เพื่อป้องกันไม่ให้เครื่องควบคุมเสียหายอันเนื่องมาจากอันตรายจากแรงดันอินพุทสูงเกินกำหนดหรือเกิดการลัดวงจรขึ้นทางอินพุท นอกจากนี้หน่วยอินพุทแบบดิจิตอลอีกหลายชนิดซึ่งได้ออกแบบมาให้เหมาะสมกับกระบวนการทางอุตสาหกรรมบางชนิด เช่นหน่วยอินพุทแบบดิจิตอลที่มีระดับแรงดันเป็นทีทีแอลเป็นต้น สำหรับหน่วยเอาท์พุทแบบดิจิตอลนั้นชนิดที่ใช้กันมากคือ เอาท์พุทที่เป็นคอนแทครีเลย์ (CONTACT RELAY) ส่วนเอาท์พุทชนิดอื่น ๆ เช่นชนิดเอาท์พุทเป็นทรานซิสเตอร์ (TRANSISTOR) หรือเป็นหน่วยเอาท์พุทสำหรับไฟสลับ ดังรูปที่ 2.7 และ 2.8



รูปที่ 2.7 วงจรเอาท์พุทแบบดิจิตอลชนิดทรานซิสเตอร์



รูปที่ 2.8 วงจรเอาท์พุทแบบดิจิตอลชนิดไฟสลับ

นอกจากที่ได้กล่าวมาทั้งหมดแล้วหน่วยอินพุท เอาท์พุทแบบดิจิตอลอีกชนิดหนึ่ง เป็นการรับส่งค่าสภาวะ เป็นกลุ่มของข้อมูล เช่นรหัส BCD หรือเป็น BINARY เป็นต้น หน่วยอินพุท เอาท์พุท เหล่านี้ได้ถูกออกแบบมาสำหรับการประมวลผลเชิงตัวเลข เพื่อการตั้งค่าเป็นตัวเลข หรือให้เครื่องควบคุมสามารถส่งข้อมูลออกมาในรูปของรหัสที่ต้องการ เป็นต้น

หน่วยอินพุท เอาท์พุทแบบอนาลอก ทำหน้าที่ติดต่อกับอุปกรณ์ภายนอกที่มีการส่งหรือรับค่าสภาวะซึ่งค่าสภาวะ เหล่านี้จะเป็นค่าปริมาณทางไฟฟ้า เช่น อุณหภูมิ ความดัน ความชื้น อัตราการไหล หรือแรงดันไฟฟ้า เป็นต้น สำหรับทาง เอาท์พุทก็มักจะเป็นระบบเซอร์โว (SERVO) หรือวาล์วควบคุม (CONTROL VALVE) การรับส่งค่าสภาวะมักจะเป็นลูปกระแส (CURRENT LOOP) 4 - 20 มิลลิแอมแปร์ หรือแรงดัน 1 - 5 โวลท์

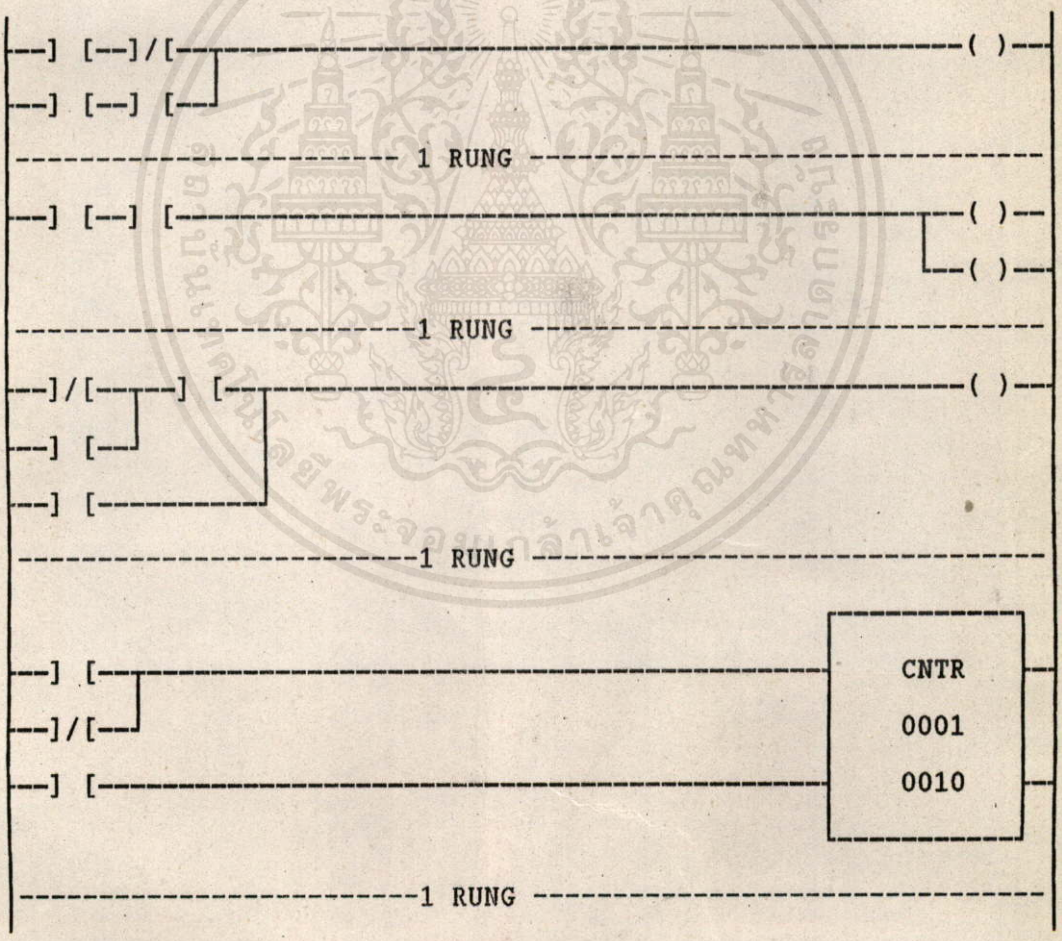
นอกจากในหัวข้อ 2.2.1 และ 2.2.2 แล้วหน่วยอินพุท เอาท์พุทยังมีอีกหลายชนิด เช่น หน่วยอินพุทชนิดความเร็วสูงสำหรับอุปกรณ์อินพุทที่มีความเร็วสูงจนหน่วยอินพุทของเครื่องควบคุมไม่สามารถตรวจจับได้ ตัวนับชนิดความเร็วสูงสำหรับการนับค่าจากอินพุทที่มีความเร็วสูงไม่สามารถใช้ตัวนับในเครื่องควบคุมได้ หรือหน่วยเชื่อมต่อกับคอมพิวเตอร์ (COMPUTER LINK) เพื่อใช้ในการรับส่งข้อมูลหรือโปรแกรมกับเครื่องคอมพิวเตอร์ด้วยพอร์ทมาตรฐาน เช่น RS-422 RS-232 เป็นต้น อุปกรณ์เหล่านี้มักจะมีไมโครโปรเซสเซอร์เพื่อการประมวลผลเป็นอิสระมิฉะนั้นเครื่องควบคุมอาจจะทำงานผิดพลาดได้ และจะทำให้การควบคุมมีประสิทธิภาพสูงขึ้นด้วย

2.2 ภาษาที่ใช้ในการโปรแกรม

ภาษาที่ใช้ในการโปรแกรมเครื่องควบคุมที่โปรแกรมได้มีอยู่ด้วยกันหลายชนิด เช่น โปรแกรมภาษาคำสั่งแบบบูลลีน (BOOLEAN INSTRUCTION PROGRAMMING) แบบแลดเดอร์ไดอะแกรม (LADDER DIAGRAM PROGRAMMING) และอื่น ๆ ในเครื่องควบคุมส่วนใหญ่จะโปรแกรมด้วยภาษาคำสั่งแบบบูลลีน และแบบแลดเดอร์ไดอะแกรม

2.2.1 โปรแกรมภาษาคำสั่งแบบแลดเดอร์ไดอะแกรม

เป็นการโปรแกรมที่มีประสิทธิภาพมาก เนื่องจากมีลักษณะคล้ายกับรีเลย์ไดอะแกรม (RELAY DIAGRAM) ดังนั้นการเขียนโปรแกรมจากรีเลย์ไดอะแกรมจึงทำได้ง่าย แลดเดอร์ไดอะแกรมจะแบ่งออกเป็นส่วน ๆ แต่ละส่วนคือ เอาท์พุท 1 จุดหรือเอาท์พุทมากกว่า 1 จุดแต่เอาท์พุทเหล่านี้จะมีลอจิกเหมือนกัน แต่ละส่วนเหล่านี้เรียกว่า "รังค์ (RUNG)"



รูปที่ 2.9 การแบ่งแลดเดอร์ไดอะแกรมออกเป็นรังค์

การโปรแกรมด้วยแลดเดอร์ไดอะแกรมจะสะดวกและมีประสิทธิภาพมากแต่ข้อเสียของการโปรแกรมแบบนี้คือ อุปกรณ์ที่ใช้ในการเขียนโปรแกรมจะต้องมีหน่วยแสดงผลที่สามารถแสดงอักขระหรือไดอะแกรมได้ครั้งละหลาย ๆ บรรทัด ดังนั้นส่วนใหญ่แล้วอุปกรณ์ที่ใช้ในการโปรแกรมแบบนี้มักจะใช้จอซีอาร์ที (CRT : CATHODE RAY TUBE) เป็นหน่วยแสดงผล ทำให้มีขนาดใหญ่ไม่สะดวกในการทำงาน แต่ในปัจจุบันได้มีการออกแบบและพัฒนาหน่วยแสดงผลที่มีขนาดเล็กเพื่อความสะดวก แต่ก็ยังมีราคาแพงและขนาดของจอภาพก็ยังมีขนาดเล็กไม่สามารถแสลงแลดเดอร์ไดอะแกรมครั้งละหลาย ๆ บรรทัด บางครั้งก็อาจจะใช้คอมพิวเตอร์หรือไมโครคอมพิวเตอร์ ทำการเชื่อมต่อกับเครื่องควบคุมเพื่อใช้ในการโปรแกรมด้วยแลดเดอร์ไดอะแกรม โดยคอมพิวเตอร์ทำหน้าที่ในการสร้างและแก้ไขแลดเดอร์ไดอะแกรมแล้วทำการแปลเป็นชุดคำสั่งของเครื่องควบคุมก่อน จากนั้นก็จะส่งชุดคำสั่งดังกล่าวมายังเครื่องควบคุมเพื่อให้ทำงานตามที่ต้องการต่อไป นอกจากนั้นแล้วขณะที่เครื่องควบคุมทำงานก็สามารถตรวจสอบการทำงานได้โดยการแสดงผล ซึ่งจะแสดงเป็นแลดเดอร์ไดอะแกรมและเมื่ออินพุทหรือเอาท์พุทตัวใดมีการเปลี่ยนแปลงสภาวะ ก็จะมีการเปลี่ยนสีตามสภาวะของการเปลี่ยนแปลงของอินพุทหรือเอาท์พุทตัวนั้น ๆ

2.2.2 โปรแกรมภาษาคำสั่งแบบบูลลีน

การโปรแกรมชนิดนี้มีลักษณะคล้ายกับสัญญลักษณ์ของพีชคณิตบูลลีน เช่น

```
LD          0010
OR          0011
AND NOT     0012
OUT         1000
```

นอกจากจะเป็นสัญญลักษณ์ของพีชคณิตบูลลีนแล้วก็ยังมีการใช้ฟังก์ชั่นพิเศษต่าง ๆ อีกมากเช่น ตัวนับ ตัวตั้งเวลา เป็นต้น

นอกจากการโปรแกรมทั้งสองแบบที่กล่าวมาแล้วก็ยังมีอีกวิธีหนึ่งที่มีใช้กันอยู่พอสมควรคือการใช้ภาษาชั้นสูง (HIGH LEVEL LANGUAGE) คือการใช้ภาษาในการโปรแกรมคอมพิวเตอร์ เช่น ภาษาปาสคาล (PASCAL) ภาษาฟอร์แทรน (FORTRAN) เป็นต้น แต่การใช้วิธีนี้มีข้อจำกัดอยู่ว่าจะทำงานได้ช้า และวิธีนี้ก็เหมาะสำหรับการโปรแกรมที่มีการทำงานซับซ้อนมากหรือมีการคำนวณทางวิทยาศาสตร์ ดังนั้นวิธีนี้มักจะใช้กับเครื่องควบคุมหรือระบบควบคุมกระบวนการขนาดใหญ่ วิธีที่มีประสิทธิภาพและใช้กันอย่างแพร่หลายโดยทั่วไปก็

คือการใช้แลดเดอร์ไดอะแกรม เนื่องจากแลดเดอร์ไดอะแกรมใกล้เคียงกับรีเลย์ไดอะแกรมมากที่สุด ตลอดจนการตรวจสอบการทำงานของโปรแกรมหรือการควบคุมก็ทำได้โดยสะดวก ส่วนฟังก์ชั่นทางคณิตศาสตร์นั้นสามารถนำมาใช้ร่วมกับแลดเดอร์ไดอะแกรมได้โดยง่าย ซึ่งรายละเอียดจะได้กล่าวต่อไป

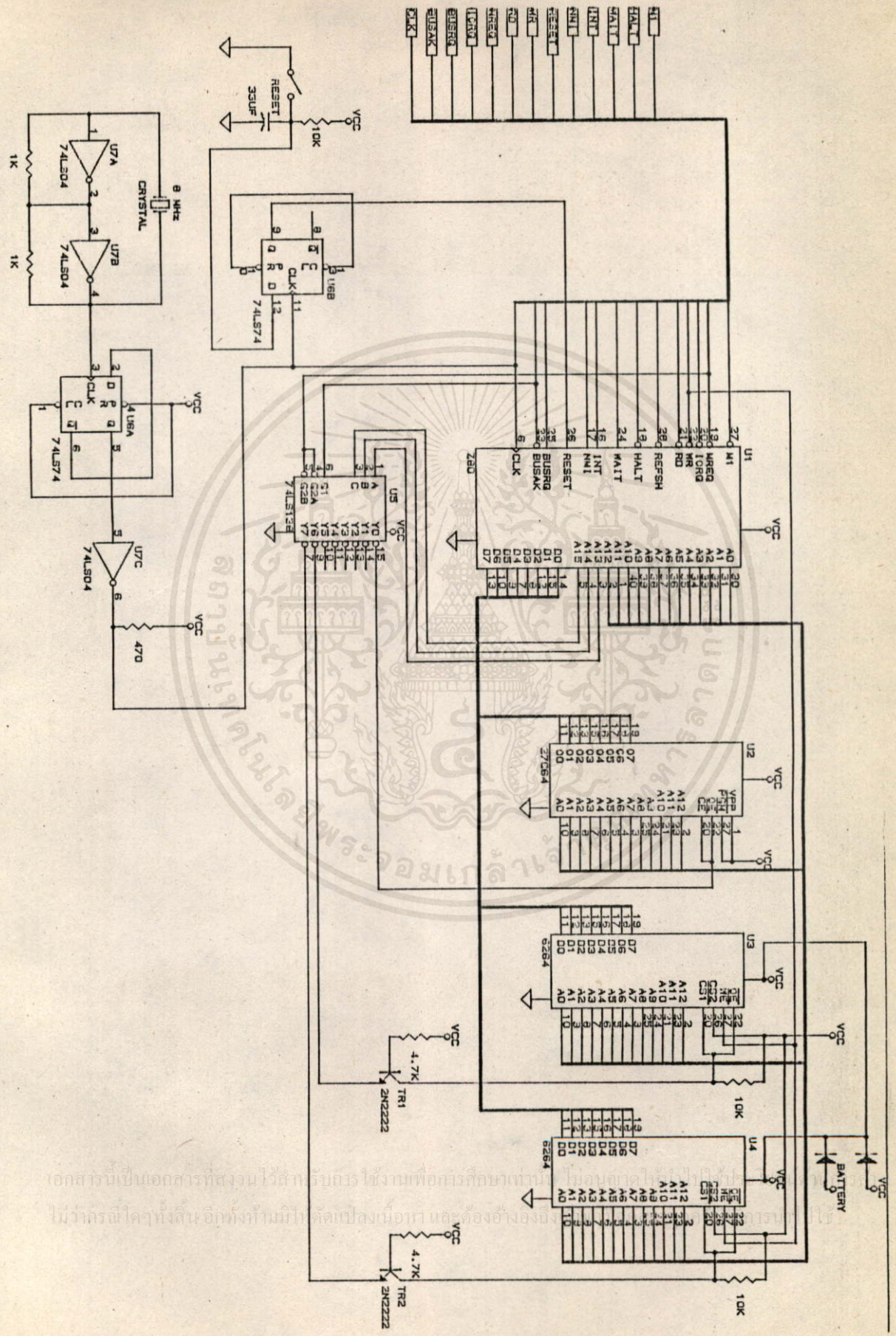
บทที่ 3
โครงสร้างของเครื่องควบคุมที่โปรแกรมได้

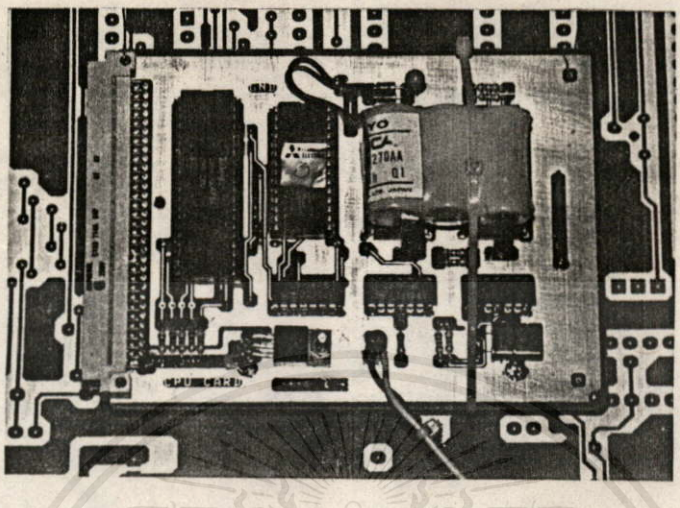
บทนี้จะกล่าวถึงคุณลักษณะของโครงสร้างทางฮาร์ดแวร์ของเครื่องควบคุมที่โปรแกรมได้ ที่นำเอาระบบไมโครโปรเซสเซอร์มาใช้เป็นหน่วยประมวลผลกลางต่อร่วมกับหน่วยอินพุทเอาท์พุทแบบต่าง ๆ หน่วยเชื่อมต่อกับคอมพิวเตอร์เพื่อใช้สำหรับทำการรับส่งข้อมูลหรือโปรแกรมหรือแสดงค่าสภาวะขณะทำการควบคุม หน่วยสร้างฐานเวลาให้กับเครื่องควบคุม โดยแบ่งอธิบายเป็นหัวข้อดังนี้

- หน่วยประมวลผลกลาง
- หน่วยอินพุทเอาท์พุท
- หน่วยเชื่อมต่อกับเครื่องไมโครคอมพิวเตอร์
- หน่วยสร้างฐานเวลา

3.1 หน่วยประมวลผลกลาง (CENTRAL PROCESSING UNIT : CPU)

หน่วยประมวลผลกลางเป็นระบบไมโครโปรเซสเซอร์ซึ่งได้ออกแบบมาเพื่อทำการควบคุมกระบวนการโดยรับค่าสภาวะมาทางหน่วยอินพุทและทำการประมวลผลข้อมูลอินพุทตามโปรแกรมที่ผู้ใช้กำหนดและส่งข้อมูลไปยังหน่วยเอาท์พุทโดยมีโปรแกรมควบคุมระบบเป็นตัวจัดการ ข้อมูลอินพุทจะได้รับมาจากค่าสภาวะของกระบวนการและข้อมูลเอาท์พุทก็คือข้อมูลที่จะส่งออกไปควบคุมกระบวนการ ส่วนโปรแกรมที่ผู้ใช้เขียนขึ้นนั้นเป็นชุดคำสั่งการประมวลผลทั้งทางด้านลอจิก คณิตศาสตร์ และการเคลื่อนย้ายข้อมูล วิทยานิพนธ์ฉบับนี้ได้ออกแบบให้หน่วยประมวลผลกลางเป็นระบบไมโครโปรเซสเซอร์ขนาด 8 บิท โดยใช้ไมโครโปรเซสเซอร์ของบริษัทไซลอก (ZILOG) คือ Z-80 ซึ่งเป็นไมโครโปรเซสเซอร์ที่มีสมรรถนะสูงและมีชุดคำสั่งที่ง่ายต่อการใช้งาน โครงสร้างทางฮาร์ดแวร์ของระบบจะมีลักษณะเป็นระบบไมโครโปรเซสเซอร์พื้นฐานดังรูปที่ 3.1 และรูปที่ 3.2





รูปที่ 3.2 หน่วยประมวลผลที่ใช้ในการทดลอง

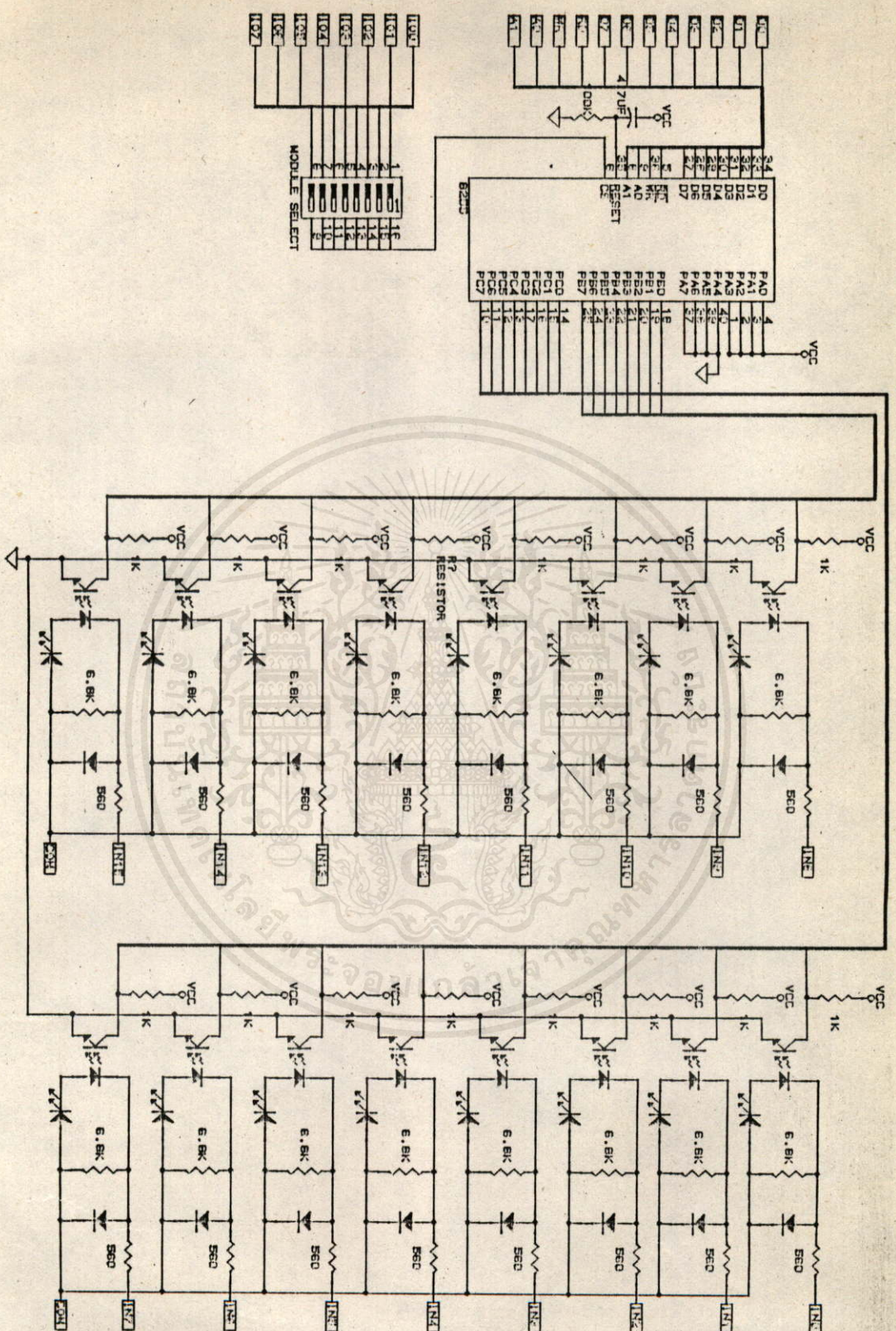
จากรูปที่ 3.1 พิจารณาวงจร U1 คือไมโครโปรเซสเซอร์ Z80 ทำงานที่ความถี่ 4 MHz ซึ่งได้จากการหารสองความถี่ของผลึกแร่ (CRYSTAL) ด้วย U6B ผ่านให้ U7C เพื่อขับกระแสให้สูงขึ้น แอสเดรสบัส (ADDRESS BUS) และบัสข้อมูล (DATA BUS) ถูกต่ออยู่กับหน่วยความจำแบบอ่านได้อย่างเดียว (ROM : READ ONLY MEMORY) U2 ขนาดความจุ 8 กิโลไบท์ซึ่งใช้สำหรับโปรแกรมควบคุมระบบ U3 และ U4 เป็นหน่วยความจำแบบอ่านได้เขียนได้ (RAM : RANDOM ACCESS MEMORY) ขนาดความจุ 8 กิโลไบท์เช่นเดียวกันโดยมี U5 ทำการถอดรหัส (DECODE) เพื่อเลือกตำแหน่งแอสเดรสของหน่วยความจำ และในหน่วยความจำแบบอ่านได้เขียนได้ก็จะมีแบตเตอรี่สำรอง (BACK-UP BATTERY) สำหรับป้องกันการเสียหายของข้อมูลใน U3 U4 อันเนื่องมาจากไฟฟ้าดับ โดยมี TR1 และ TR2 สำหรับตัดวงจรเพื่อไม่ต้องการให้มีกระแสไหลจากแบตเตอรี่เข้าไปในวงจรอื่นนอกจาก U3 U4 ส่วนการเริ่มการทำงานของ U1 มี U6B ร่วมกับ R1 C1 ทำการควบคุมจังหวะการเริ่มทำงานเมื่อเริ่มมีการจ่ายแรงดันให้กับวงจร

3.2 หน่วยอินพุทเอาท์พุท (INPUT OUTPUT UNIT)

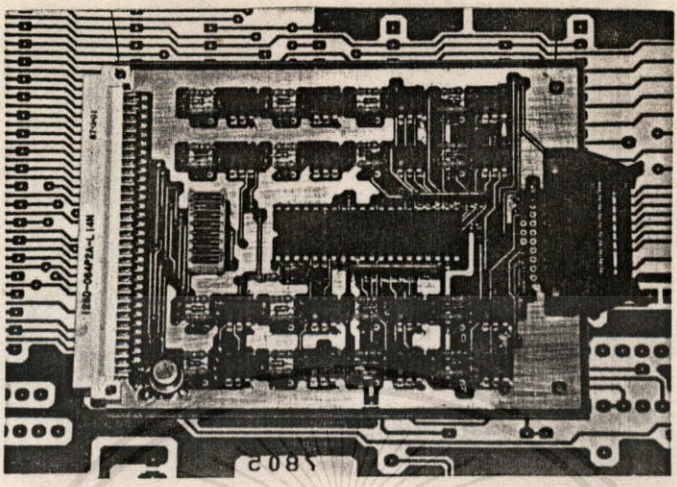
หน่วยอินพุทเอาท์พุท มีหน้าที่เชื่อมต่อกับอุปกรณ์ภายนอก ทั้งการรับค่าสภาวะหรือค่าที่วัดจากอุปกรณ์ภายนอก เช่นการเปิดปิดของสวิทช์ต่างๆ ระดับของเหลว อุณหภูมิ ความดัน ระดับแรงดันและกระแสไฟฟ้า หรือส่งค่าสภาวะออกไปยังอุปกรณ์ควบคุม หรืออุปกรณ์ที่ต้องการควบคุมเช่น รีเลย์ มอเตอร์ไฟฟ้า ปั๊ม วาล์ว เป็นต้น แบ่งออกดังนี้

3.2.1 หน่วยอินพุทแบบสภาวะลอจิก ทำหน้าที่รับค่าสภาวะของอุปกรณ์ภายนอกที่มีการเปลี่ยนแปลงสภาวะเป็นแบบลอจิกที่มีแรงดันไฟฟ้าขนาด 12 โวลท์ โดยเปลี่ยนให้เป็นสภาวะทางลอจิกจากนั้นหน่วยประมวลผลจะทำการอ่านและนำเก็บสภาวะลอจิกดังกล่าวไว้ในตารางอินพุทเอาท์พุท ดังแสดงในรูปที่ 3.3

จากรูปที่ 3.3 แรงดันไฟฟ้าที่จุด IN0 - IN15 จะต้องมีขนาดแรงดัน 5 - 12 โวลท์เมื่อเทียบกับจุด COM เพื่อที่จะทำให้ทรานซิสเตอร์ (TRANSISTOR) ที่อยู่ภายในอุปกรณ์เชื่อมต่อด้วยแสงหรือออปโตไอโซเรเตอร์ (OPTO ISOLATOR) สามารถนำกระแส (CONDUCT) ได้ ออปโตไอโซเรเตอร์ทำหน้าที่เชื่อมต่อการส่งสัญญาณไฟฟ้าด้วยแสงเพื่อแยกออกจากกันอย่างเด็ดขาดทางไฟฟ้าระหว่างอุปกรณ์ภายนอกกับเครื่องควบคุม เพื่อป้องกันความเสียหายซึ่งอาจจะเกิดขึ้นเนื่องมาจากการลัดวงจรของอุปกรณ์ภายนอก การนำกระแสของทรานซิสเตอร์ก็คือสภาวะลอจิกที่มีแรงดันตามมาตรฐานทีทีแอล (TTL) ซึ่งสภาวะลอจิกดังกล่าวก็คือข้อมูลสภาวะอินพุทนั่นเอง ข้อมูลนี้จะถูกส่งไปยัง U1 (8255 PPI) U1 จะทำหน้าที่รับข้อมูลเหล่านี้ส่งผ่านให้หน่วยประมวลผลกลาง เมื่อหน่วยประมวลผลกลางต้องการอ่านค่าสภาวะ พอร์ท U1 ที่ PA0 - PA3 ถูกทำให้มีระดับแรงดันเท่ากับแหล่งจ่าย และ PA4 - PA7 มีแรงดัน 0 โวลท์เพื่อให้หน่วยประมวลผลรับรู้ว่าเป็นหน่วยอินพุทเนื่องจากว่ามีการใช้พื้นที่หน่วยความจำในตารางอินพุทเอาท์พุทร่วมกัน

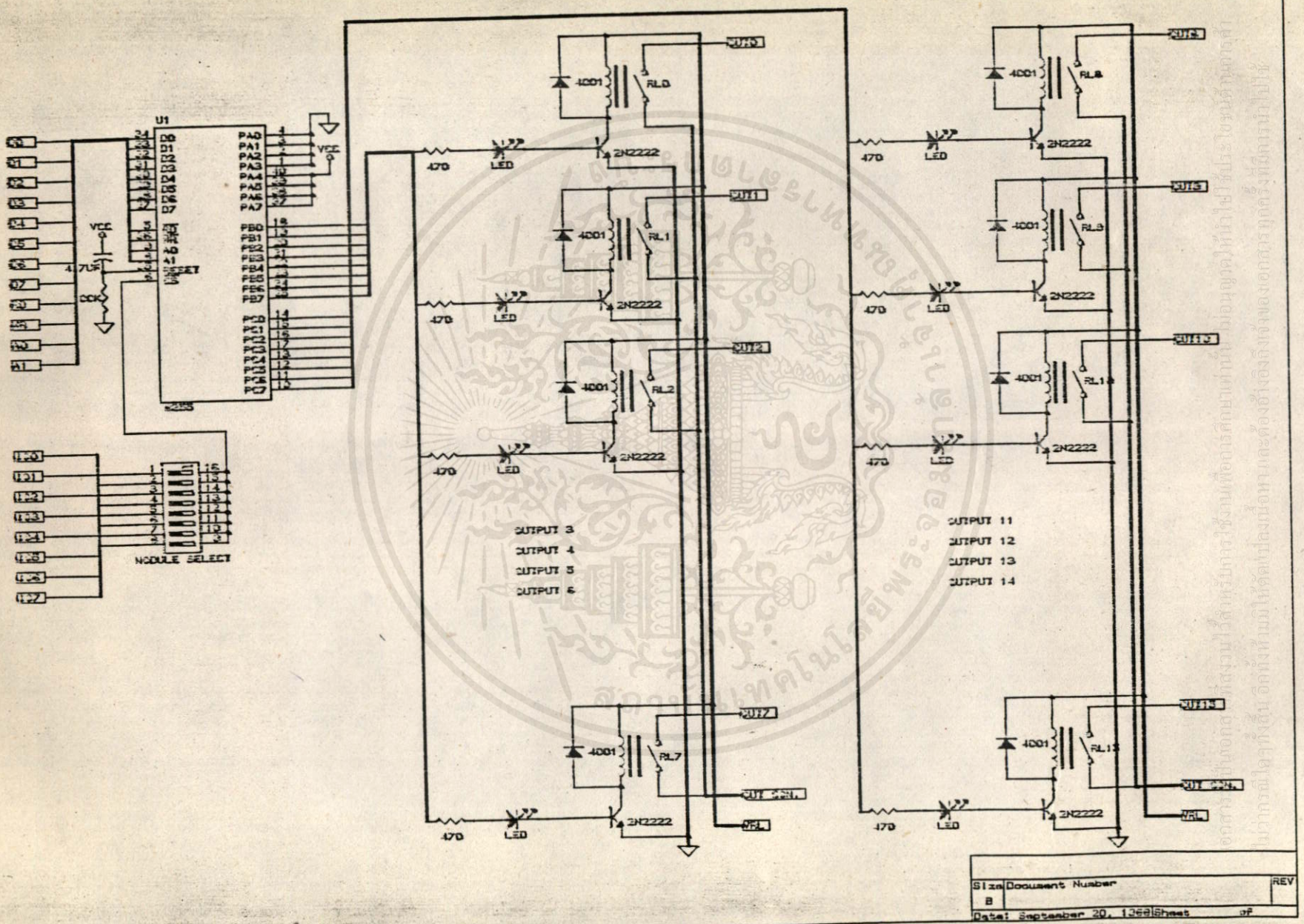


รูปที่ 3.3 วงจรหน่วยอินพุทแบบลอจิก

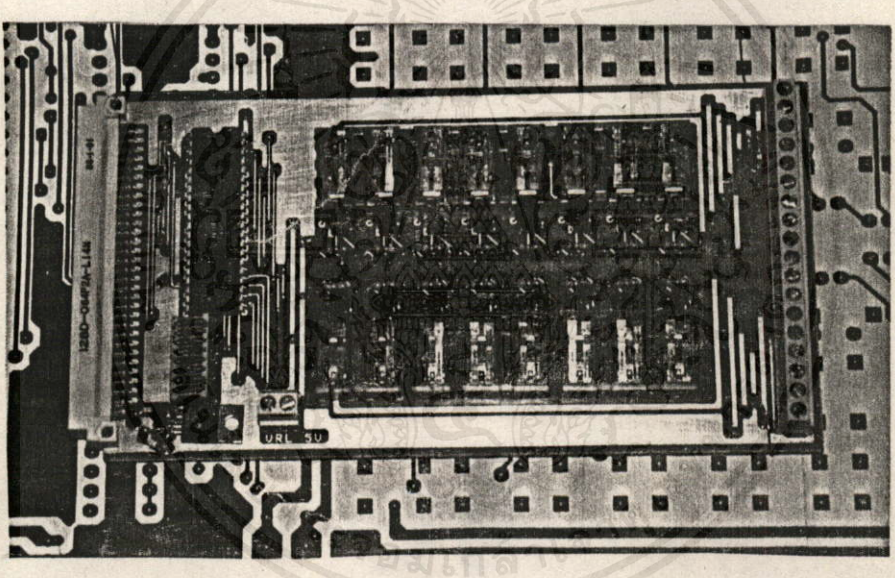


รูปที่ 3.4 หน่วยอินพุทแบบลอจิกที่ใช้ในการทดลอง

3.2.2 หน่วย เอาท์พุทแบบหน้าสัมผัส ทำหน้าที่นำข้อมูลจากหน่วยประมวลผลกลาง ส่งออกไปยังอุปกรณ์ควบคุมซึ่ง เชื่อมต่ออยู่ให้มีสภาวะตามต้องการ โดยข้อมูลดังกล่าวได้จาก ตาราง เอาท์อินพุท เอาท์พุท แล้วใช้หน้าสัมผัส เป็นตัวควบคุม เปิดปิดวงจรไฟฟ้า สามารถควบคุมอุปกรณ์ไฟฟ้าทั้งชนิดกระแสตรงหรือกระแสสลับได้ ดังแสดงในรูปที่ 3.5


U1
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
VCC
RESET
MODULE SELECT
OUT0
OUT1
OUT2
OUT7
OUT8
OUT9
OUT13
OUT14
OUT15
4001
RL0
RL1
RL2
RL7
RL8
RL9
RL13
2N2222
LED
470
OUTPUT 3
OUTPUT 4
OUTPUT 5
OUTPUT 6
OUTPUT 11
OUTPUT 12
OUTPUT 13
OUTPUT 14
Size
B
Document Number
REV
Date: September 20, 1988
Sheet
of

พอร์ท PA0 - PA3 ถูกทำให้มีแรงดัน 0 โวลท์หรือเป็นลอจิก 0 และ PA4 - PA7 ถูกทำให้มีแรงดันเท่ากับแรงดันแหล่งจ่ายหรือให้มีลอจิกเป็น 1 ซึ่งจะตรงข้ามกับหน่วยอินพุท เพื่อให้หน่วยประมวลผลรับรู้การเป็นหน่วยเอาท์พุท หน่วยประมวลผลจะส่งสภาวะลอจิกให้กับพอร์ท U1 ทำให้มีการเปลี่ยนแปลงของระดับแรงดันไฟฟ้า เป็นผลให้ทรานซิสเตอร์เกิดการนำกระแสไหลผ่านขดลวดของรีเลย์และทำให้หน้าสัมผัสทำงาน ดังนั้นอุปกรณ์ควบคุมที่ต่ออยู่กับเอาท์พุทตำแหน่งดังกล่าวก็จะทำงานตามสภาวะที่ต้องการ

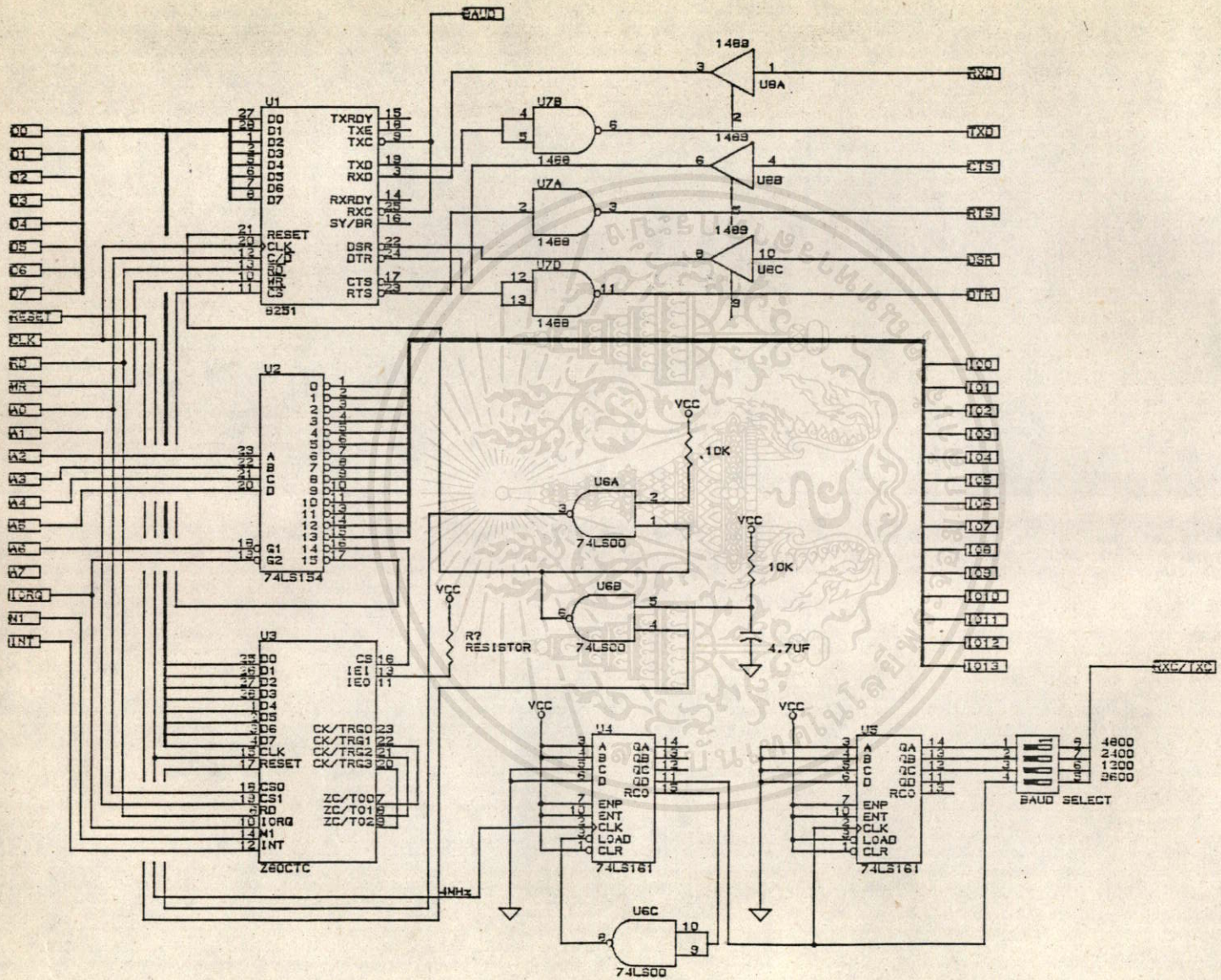


รูปที่ 3.6 วงจรหน่วยเอาท์พุทแบบรีเลย์ที่ใช้ในการทดลอง

3.3 หน่วยเชื่อมต่อกับคอมพิวเตอร์

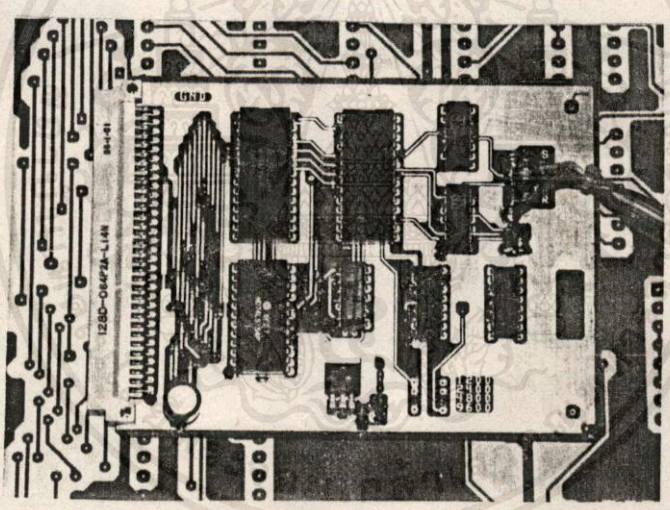
หน่วยเชื่อมต่อกับคอมพิวเตอร์ทำหน้าที่ส่งชุดคำสั่งของโปรแกรมผู้ใช้ซึ่งจะเป็นโปรแกรมที่กำหนดลำดับการทำงานและการประมวลผลของการควบคุม ไปยังเครื่องควบคุมและรับค่าสภาวะอินพุทเอาท์พุทจากเครื่องควบคุมมายังคอมพิวเตอร์ เพื่อใช้ในการตรวจสอบและติดตามการเปลี่ยนแปลงสภาวะของกระบวนการ การเชื่อมต่อกับคอมพิวเตอร์กระทำผ่านพอร์ทอนุกรมมาตรฐาน RS-232C ดังแสดงในรูปที่ 3.7

U1 (UART 8251) เป็นอุปกรณ์ที่ใช้สำหรับการรับและส่งข้อมูลแบบอนุกรมโดยมีบัสข้อมูลต่อกับไมโครโปรเซสเซอร์ เพื่อรับส่งข้อมูลระหว่างคอมพิวเตอร์กับเครื่องควบคุม การติดต่อแบบอนุกรมนี้จะทำให้ระยะทางในการสื่อสารข้อมูลได้ระยะทางไกลและค่าใช้จ่ายต่ำ เนื่องจากระดับแรงดันของสัญญาณถูกทำให้มีขนาดสูงถึง -12 และ +12 โวลท์ เมื่อข้อมูลเป็น 0 และ 1 ตามลำดับ ข้อมูลที่มีระดับแรงดันทีทีแอลจะถูกเปลี่ยนให้มีระดับแรงดันดังกล่าวด้วย U7 และ U8 ส่วนความเร็วในการรับส่งขึ้นอยู่กับความถี่ของสัญญาณนาฬิกาที่ได้จาก U4 U5 U6C โดยปกติแล้วจะใช้ความเร็วในการรับส่งขนาด 9600 บิทต่อวินาที


BAUD
U1
8251
RESET
CLK
RD
WR
CS
TXRDY
TXE
TXC
TXD
RXD
RXRDY
RXC
SY/BR
DSR
DTR
CTS
RTS
U7B
U7A
U7D
1488
U8A
U8B
U8C
1489
TXD
RXD
CTS
RTS
DSR
DTR
U2
74LS154
G1
G2
U3
Z80CTC
CS
IEI
IEO
CK/TRG0
CK/TRG1
CK/TRG2
CK/TRG3
ZC/TO0
ZC/TO1
ZC/TO2
IORQ
M1
INT
4MHz
VCC
10K
R?
RESISTOR
4.7UF
U6A
U6B
U6C
74LS00
U4
U5
74LS161
ENP
ENT
CLK
LOAD
CLR
RCO
BAUD SELECT
4800
2400
1200
9600
RXC/TXC

3.4 หน่วยสร้างฐานเวลา

หน่วยสร้างฐานเวลาทำหน้าที่สร้างฐานเวลาให้กับเครื่องควบคุม เนื่องจากฟังก์ชั่นการควบคุมบางฟังก์ชั่นจะมีการนำค่าเวลามาประมวลผลด้วย การสร้างฐานเวลากระทำโดย U3 (Z80 CTC) ซึ่งเป็นอุปกรณ์ที่ใช้สำหรับทำการหารความถี่จากสัญญาณนาฬิกาของไมโครโปรเซสเซอร์ให้มีความถี่ลดลงจนใกล้เคียงกับสัญญาณนาฬิกาที่ใช้เป็นฐานเวลา การกำหนดค่าในการหารนั้นทำได้โดยการโปรแกรมค่าการหารผ่านทางบัสข้อมูลที่ต่อร่วมกับไมโครโปรเซสเซอร์ หลังจากทำการโปรแกรมแล้ว U3 ก็จะทำหน้าที่ในการนับและหารค่าเวลา ไมโครโปรเซสเซอร์ก็จะอ่านค่าฐานเวลาได้จาก U3 แล้วนำค่าฐานเวลาดังกล่าวไปประมวลผลต่อไป



รูปที่ 3.8 วงจรหน่วยเชื่อมต่อกับเครื่องคอมพิวเตอร์และหน่วยสร้างฐานเวลา

บทที่ 4
ระบบจัดการของเครื่องควบคุมที่โปรแกรมได้

ระบบจัดการของเครื่องควบคุมหมายถึงโปรแกรมคำสั่งที่ใช้ในการควบคุมการทำงาน การจัดการหน่วยความจำและโครงสร้างข้อมูล การติดต่อกับหน่วยอื่น ๆ ทำการแปลคำสั่งควบคุมให้เป็นรหัสที่หน่วยประมวลผลกลางสามารถปฏิบัติได้ และการทำงานของฟังก์ชั่นควบคุม โดยจะอธิบายเป็นลักษณะของอัลกอลิทัม แบ่งเป็นหัวข้อดังนี้

- การแบ่งพื้นที่ใช้งานของหน่วยความจำ
- การจัดข้อมูลของโปรแกรมผู้ใช้
- การแปลชุดคำสั่งจากโปรแกรมผู้ใช้เป็นภาษาเครื่อง
- การทำงานของฟังก์ชั่นควบคุม

4.1 การแบ่งพื้นที่ใช้งานของหน่วยความจำ

จากบทที่ 3 ในโครงสร้างทางฮาร์ดแวร์ของหน่วยประมวลผลกลางมีหน่วยความจำที่ใช้ในการประมวลผลขนาด 24 กิโลไบท์ที่ถูกกำหนดขึ้นสำหรับเครื่องควบคุม แบ่งเป็นพื้นที่ใช้งานดังรูปที่ 4.1

รูปที่ 4.1 แสดงการแบ่งพื้นที่ใช้งานของหน่วยความจำ

4.1.1 โปรแกรมบริหารงานระบบ ใช้พื้นที่หน่วยความจำร่วมกับข้อมูลระบบ มีขนาด 8 กิโลไบท์ ถูกใช้เป็นโปรแกรมจัดวางระบบข้อมูลโปรแกรมผู้ใช้ แปลชุดคำสั่งของโปรแกรมผู้ใช้เป็นภาษาเครื่อง ติดต่อสื่อสารกับหน่วยป้อนโปรแกรมและอุปกรณ์ร่วมอื่น ๆ พื้นที่ส่วนนี้ผู้ใช้ไม่สามารถที่จะเปลี่ยนแปลงแก้ไขข้อมูลในหน่วยความจำได้

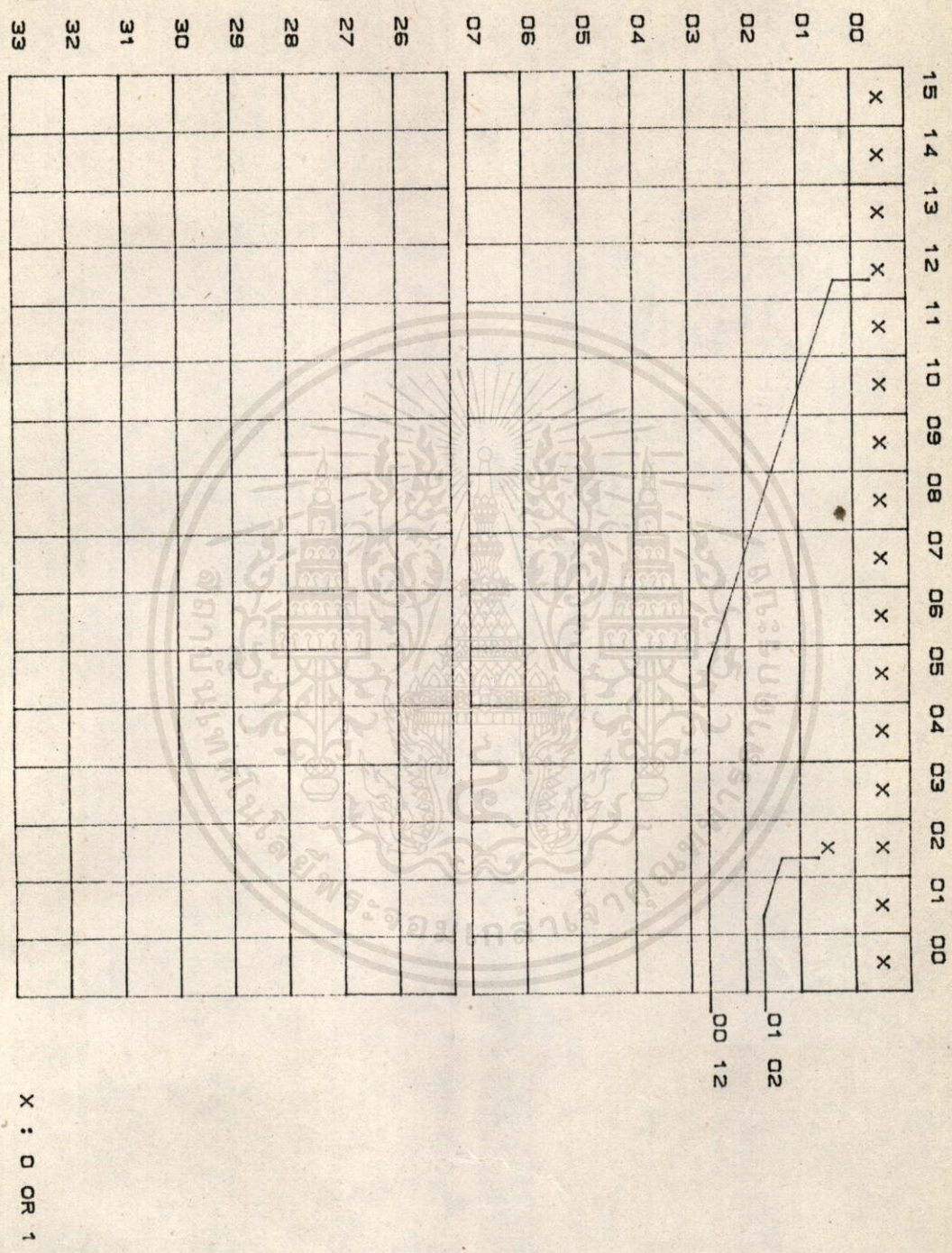
4.1.2 โปรแกรมควบคุม เป็นพื้นที่หน่วยความจำของโปรแกรมในการประมวลผลฟังก์ชั่นควบคุมที่ผู้ใช้ทำการโปรแกรม เข้ามาในหน่วยความจำของพื้นที่ส่วนโปรแกรมผู้ใช้

4.1.3 ตารางข้อมูล เป็นพื้นที่หน่วยความจำที่ผู้ใช้สามารถเปลี่ยนแปลงแก้ไขได้ ภายในตารางข้อมูล จะถูกเก็บข้อมูลที่มีลักษณะ "1" หรือ "0" ใช้แทนสภาวะ "ON" และ "OFF" ทางไฟฟ้า และข้อมูลแทนค่าตัวเลขที่ใช้ในการคำนวณทางคณิตศาสตร์ ข้อมูลของการตั้งค่าเวลาและค่าการนับ ซึ่งตารางข้อมูลได้ถูกแบ่งออกเป็น 3 ส่วนดังรูปที่ 4.2

รูปที่ 4.2 แสดงการแบ่งพื้นที่ใช้งานในตารางข้อมูล

ตารางอินพุท เอาท์พุทและตารางรีเลย์ภายในถูกกำหนดขึ้นเป็นจำนวน 34 ชุด เป็นชุดที่ 0 ถึงชุดที่ 33 ส่วนข้อมูลชุดที่ 0 ถึง 11 เป็นข้อมูลสำหรับตารางอินพุท เอาท์พุทและข้อมูลชุดที่ 12 ถึง 33 เป็นข้อมูลสำหรับตารางรีเลย์ภายใน แต่ละชุดของข้อมูลภายในตารางจะเป็นข้อมูลขนาด 8 บิทจำนวน 2 ไบท์ ในรูปแบบเลขจำนวนเต็มแบบเวิร์ด (WORD) เพื่อให้สามารถกำหนดสภาวะอินพุทหรือเอาท์พุทได้ถึง 16 สภาวะ และเก็บค่าในการคำนวณทางคณิตศาสตร์ได้โดยมีความละเอียดถึง 16 บิทหรือ 0 ถึง 65535

ตารางรีจีสเตอร์กำหนดให้มีขนาด 34 ชุด ซึ่งเป็นข้อมูลในรูปแบบเลขจำนวนเต็มฐานสิบขนาด 4 หลัก (BCD INTEGER) ใช้สำหรับเก็บข้อมูลค่าการตั้งเวลาและการนับอย่างละ 17 ชุดข้อมูลโดยมีความละเอียด 0 ถึง 9999 ค่า



รูปที่ 4.3 แสดงตารางอิบพุท เอาท์พุทและตารางรีเลย์ภายใน

พิจารณาชุดข้อมูลที่ 00 ซึ่งเป็นชุดข้อมูลหนึ่งในตารางอินพุท เอาท์พุท ในการเรียกหรืออ้างถึงค่าสภาวะ จะทำการอ้างถึงตำแหน่งชุดข้อมูลก่อนและตามด้วยตำแหน่งบิทของข้อมูลชุดนั้น เช่น

คำสั่ง LD 0012 หมายความว่า นำค่าสภาวะจากตำแหน่งข้อมูลชุดที่ 00 บิทที่ 12

คำสั่ง OR 0102 หมายความว่า นำมาทำลอจิก OR กับค่าสภาวะในชุดข้อมูลที่ 01 บิทที่ 2

4.1.4 โปรแกรมผู้ใช้ หน่วยความจำส่วนนี้เก็บโปรแกรมควบคุมที่ผู้ใช้เป็นเขียนขึ้นโดยใช้หน่วยป้อนโปรแกรมซึ่งจะถูกเปลี่ยนให้เป็นรหัสของชุดคำสั่งนั้น ๆ ข้อมูลในหน่วยความจำของโปรแกรมผู้ใช้ จะเป็นรหัสของชุดคำสั่งซึ่งเป็นคำสั่งทางลอจิก คำสั่งทางคณิตศาสตร์ คำสั่งการเคลื่อนย้ายข้อมูล ควบคู่กับตำแหน่งของอินพุทหรือเอาท์พุท เช่น

LD	0010	คำสั่งคือ LD ตำแหน่งอินพุท เอาท์พุท 0010
OUT NOT	0100	คำสั่งคือ OUT NOT ตำแหน่งอินพุท เอาท์พุท 0100

จากนั้นโปรแกรมบริหารงานระบบจะทำการแปลความหมายรหัสเหล่านี้ให้เป็นชุดคำสั่งของไมโครโปรเซสเซอร์ที่ใช้เป็นหน่วยประมวลผลกลาง

4.2 การจัดข้อมูลของโปรแกรมผู้ใช้

ข้อมูลของโปรแกรมผู้ใช้จะมีลักษณะเป็นชุดคำสั่งแต่ละชุดคำสั่งถูกจัดเป็นเรคคอร์ด (RECORD) ภายในประกอบด้วยฟิลด์ (FIELD) 7 ฟิลด์ แต่ละเรคคอร์ดมีขนาด 10 ไบท์ โดยมีโครงสร้างของข้อมูลดังนี้

```
RECORD
  CODE : BYTE;
  TYPE1 : BYTE;
  DATA1 : WORD;
  TYPE2 : BYTE;
  DATA2 : WORD;
  TYPE3 : BYTE;
  DATA3 : WORD;
END;
```

ฟิลด์	ชนิด	ความหมาย
CODE	ไบท์	กำหนดรหัสของคำสั่ง
TYPE1	ไบท์	ชนิดของข้อมูล DATA1 เป็นเลขฐานสิบหรือฐานสิบหก
DATA1	เวิร์ด	ข้อมูลตำแหน่งตารางข้อมูลที่ 1
TYPE2	ไบท์	บอกชนิดข้อมูล DATA2 เป็นเลขฐานสิบหรือฐานสิบหก
DATA2	เวิร์ด	ข้อมูลตำแหน่งตารางข้อมูลที่ 2
TYPE3	ไบท์	บอกชนิดข้อมูล DATA3 เป็นเลขฐานสิบหรือฐานสิบหก
DATA3	เวิร์ด	ข้อมูลตำแหน่งตารางข้อมูลที่ 3

การเก็บข้อมูลของเรคคอร์ดดังกล่าวในหน่วยความจำจะเก็บเรียงกันไปในหน่วยความจำดังรูปที่ 4.4

	BYTE	BYTE	WORD	BYTE	WORD	BYTE	WORD
REC 1	CODE	TYPE1	DATA1	TYPE2	DATA2	TYPE3	DATA3
REC 2	CODE	TYPE1	DATA1	TYPE2	DATA2	TYPE3	DATA3
REC 3	CODE	TYPE1	DATA1	TYPE2	DATA2	TYPE3	DATA3
REC N	CODE	TYPE1	DATA1	TYPE2	DATA2	TYPE3	DATA3

รูปที่ 4.4 แสดงการเก็บข้อมูลของโปรแกรมผู้ใช้ในหน่วยความจำ

เช่น	LD	0001	ข้อมูลตำแหน่งตารางข้อมูลที่ 1 เป็น 0001
			ข้อมูลตำแหน่งตารางข้อมูลที่ 2 เป็น 0000
			ข้อมูลตำแหน่งตารางข้อมูลที่ 3 เป็น 0000
	ADD	05	ข้อมูลตำแหน่งตารางข้อมูลที่ 1 เป็น 05
		06	ข้อมูลตำแหน่งตารางข้อมูลที่ 2 เป็น 06
		40	ข้อมูลตำแหน่งตารางข้อมูลที่ 3 เป็น 40

เรคคอร์ดของชุดคำสั่งจะต้องทำการ เก็บรหัสชุดคำสั่งและข้อมูลตำแหน่งตารางข้อมูล อีก 3 ตำแหน่งข้อมูล พร้อมทั้งบอกชนิดของข้อมูล เพื่อให้ เป็นการง่ายในการแปลความหมายให้ เป็นชุดคำสั่งของหน่วยประมวลผลกลาง ซึ่งจะได้กล่าวต่อไปในหัวข้อของการแปลความหมายโปรแกรมผู้ใช้ เป็นภาษา เครื่อง

ฟิล์ด CODE ใช้กำหนดรหัสของคำสั่งที่ผู้ใช้ทำการโปรแกรมขึ้น เพื่อให้ เครื่องควบคุมทำงานตามฟังก์ชั่นการควบคุม คำสั่งที่ได้สร้างขึ้นมีทั้งหมด 58 คำสั่ง ดังรายละ เอียดต่อไปนี้

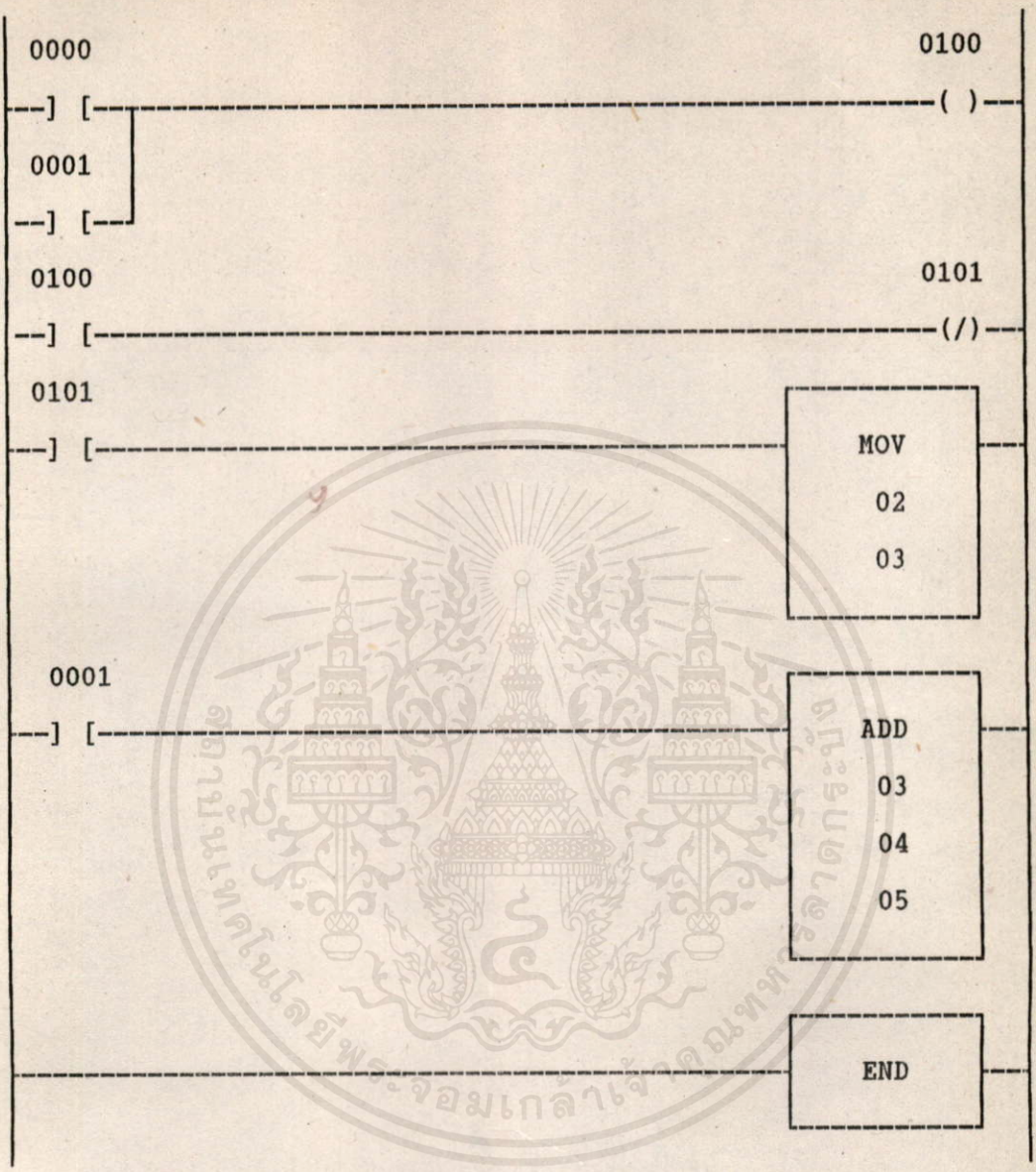
CODE	คำสั่ง	ความหมาย
00	LD	นำค่าสภาวะจากตารางข้อมูลมา เป็นผลลัพธ์
01	LD NOT	นำค่าสภาวะที่ตรงข้ามจากตารางข้อมูลมา เป็นผลลัพธ์
02	AND	กระทำลอจิก AND กับสภาวะที่นำจากตารางข้อมูล
03	AND NOT	กระทำลอจิก AND กับสภาวะตรงข้ามที่นำจากตารางข้อมูล
04	OR	กระทำลอจิก OR กับสภาวะที่นำจากตารางข้อมูล
05	OR NOT	กระทำลอจิก OR กับสภาวะตรงข้ามที่นำจากตารางข้อมูล
06	AND LD	กระทำลอจิก AND กับสภาวะที่นำจาก STACK
07	OR LD	กระทำลอจิก OR กับสภาวะที่นำจาก STACK
08	OUT	นำผลลัพธ์ทางลอจิกไว้ในตารางข้อมูล
09	OUT NOT	นำผลลัพธ์ทางลอจิกแล้วทำตรงข้ามไว้ในตารางข้อมูล
10	TIM	เรียกใช้ตัวตั้งเวลา
11	CNT	เรียกใช้ตัวนับ
12	LD TIM	นำค่าสภาวะจากตัวตั้งเวลามา เป็นผลลัพธ์

13	LD NOT TIM	นำค่าสภาวะที่ตรงข้ามจากตัวตั้งเวลามาเป็นผลลัพธ์
14	LD CNT	นำค่าสภาวะจากตัวนับมาเป็นผลลัพธ์
15	LD NOT CNT	นำค่าสภาวะที่ตรงข้ามจากตัวนับมาเป็นผลลัพธ์
16	AND TIM	กระทำลอจิก AND กับสภาวะที่นำจากตัวตั้งเวลา
17	AND NOT TIM	กระทำลอจิก AND กับสภาวะที่ตรงข้ามนำจากตัวตั้งเวลา
18	AND CNT	กระทำลอจิก AND กับสภาวะที่นำจากตัวนับ
19	AND NOT CNT	กระทำลอจิก AND กับสภาวะตรงข้ามที่นำจากตัวนับ
20	OR TIM	กระทำลอจิก OR กับสภาวะที่นำจากตัวตั้งเวลา
21	OR NOT TIM	กระทำลอจิก OR กับสภาวะตรงข้ามที่นำจากตัวตั้งเวลา
22	OR CNT	กระทำลอจิก OR กับสภาวะที่นำจากตัวนับ
23	OR NOT CNT	กระทำลอจิก OR กับสภาวะตรงข้ามที่นำจากตัวนับ
128	NOP	ไม่มีการทำงาน
129	END	สิ้นสุดคำสั่งให้เริ่มทำรอบใหม่
130	IL	เริ่มให้มีการควบคุมหลัก
131	ILC	สิ้นสุดการควบคุมหลัก
132	JMP	กระโดดไปข้ามไปที่คำสั่ง JME โดยไม่ทำงาน
133	JME	เริ่มทำงานหลังจากคำสั่ง JME
138	SFT	ทำการเลื่อนข้อมูลในตารางข้อมูลทางขวาหนึ่งบิท
139	KEEP	คงค่าสภาวะเมื่อมีการ SET และคลายค่าเมื่อ RESET
140	CNTR	เรียกใช้ตัวนับแบบนับขึ้นลง
141	DIFU	คงค่าสภาวะหนึ่งรอบการทำงานเมื่อรับสัญญาณขาขึ้น
142	DIFD	คงค่าสภาวะหนึ่งรอบการทำงานเมื่อรับสัญญาณขาลง
143	TIMH	เรียกใช้ตัวตั้งเวลาความเร็วสูง
148	CMP	เปรียบเทียบค่าข้อมูลในตารางข้อมูล
149	MOV	ย้ายข้อมูลไปอีกที่หนึ่งในตารางข้อมูล
150	MVN	ย้ายข้อมูลและทำตรงข้ามไปอีกที่หนึ่งในตารางข้อมูล
151	BIN	เปลี่ยนค่าข้อมูลเป็นเลขฐานสองในตารางข้อมูล
152	BCD	เปลี่ยนค่าข้อมูลเป็นเลขฐานสิบในตารางข้อมูล
157	COM	เปลี่ยนค่าข้อมูลเป็นตรงข้ามในตารางข้อมูล
158	ADD	บวกค่าของข้อมูลและเก็บไว้ในตารางข้อมูล

159	SUB	ลบค่าของข้อมูลและเก็บไว้ในตารางข้อมูล
160	MUL	คูณค่าของข้อมูลและเก็บไว้ในตารางข้อมูล
161	DIV	หารค่าของข้อมูลและเก็บไว้ในตารางข้อมูล
162	ANDW	กระทำลอจิก AND กับข้อมูลในตารางข้อมูลทั้ง เวิร์ด
163	ORW	กระทำลอจิก OR กับข้อมูลในตารางข้อมูลทั้ง เวิร์ด
164	XORW	กระทำลอจิก XOR กับข้อมูลในตารางข้อมูลทั้ง เวิร์ด
165	XNRW	กระทำลอจิก XOR และทำตรงข้ามกับข้อมูลในตารางข้อมูล
166	INC	เพิ่มค่าข้อมูลในตารางข้อมูลหนึ่งค่า
167	DEC	ลดค่าข้อมูลในตารางข้อมูลหนึ่งค่า

TYPE1 ใช้แสดงชนิดของข้อมูลใน DATA1 เป็นเลขฐานสิบหรือฐานสิบหก ในกรณีที่เป็นเลขฐานสิบ TYPE1 จะมีค่าข้อมูลเป็น 00 ส่วนเลขฐานสิบหก TYPE1 จะแสดงค่าข้อมูลเป็น 01 ในทำนองเดียวกัน TYPE2 TYPE3 ก็จะแสดงชนิดของข้อมูลใน DATA2 DATA3 ตามลำดับ เช่นเดียวกับ TYPE1 และ DATA1

DATA2 ใช้เป็นข้อมูลบอกตำแหน่งของตารางข้อมูล ข้อมูลเหล่านี้มักเป็นเลขฐานสิบ ส่วนข้อมูลที่เป็นเลขฐานสิบหกจะถูกวางให้อยู่ในตารางรีจีสเตอร์ ได้แก่ ค่าพารามิเตอร์ของการตั้งเวลาและการนับ ตัวอย่างแลดเดอร์ไดอะแกรมและชุดคำสั่งแบบบูลลีน รวมทั้งความหมายของคำสั่งต่าง ๆ ตลอดจนข้อมูลจากผู้ใช้ที่ทำการโปรแกรมผ่านหน่วยป้อนโปรแกรมเข้ามาที่หน่วยความจำส่วนโปรแกรมผู้ใช้ ดังในรูปที่ 4.5



รูปที่ 4.5 (ก) แลดเดอร์ไดอะแกรมที่ใช้ยกตัวอย่าง

ชุดคำสั่งที่	คำสั่ง	พารามิเตอร์
0000	LD	0000
0001	OR	0001
0002	OUT	0100
0003	LD	0100
0004	OUT NOT	0101
0005	LD	0101
0006	MOV	02 03
0007	LD	0001
0008	ADD	03 04 05
0009	END	

รูปที่ 4.5 (ข) ชุดคำสั่งที่ได้จากแลดเดอร์ไดอะแกรมในรูปที่ 4.5 (ก)

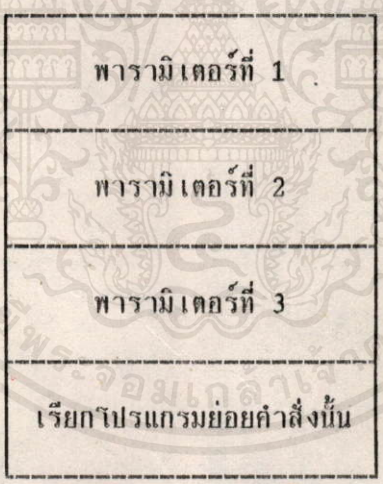
เรคคอร์ดที่	CODE	TYPE1	DATA1	TYPE2	DATA2	TYPE3	DATA3
0001	00	00	0000	00	0000	00	0000
0002	04	00	0001	00	0000	00	0000
0003	08	00	0100	00	0000	00	0000
0004	00	00	0100	00	0000	00	0000
0005	09	00	0101	00	0000	00	0000
0006	149	00	0002	00	0003	00	0000
0007	00	00	0001	00	0000	00	0000
0008	158	00	0003	00	0004	00	0005
0009	129	00	0000	00	0000	00	0000

รูปที่ 4.5 (ค) เป็นรหัสคำสั่งที่ถูกจัดเตรียมขึ้น

4.3 การแปลความหมายโปรแกรมผู้ใช้เป็นภาษาเครื่อง

หน่วยความจำตรงพื้นที่โปรแกรมผู้ใช้มีลักษณะของข้อมูลเป็นแบบรหัสชุดคำสั่ง ซึ่งหน่วยประมวลผลไม่สามารถที่จะนำไปปฏิบัติได้โดยตรง จำเป็นจะต้องทำการแปลเพื่อให้หน่วยประมวลผลสามารถรับรู้ถึงคำสั่งที่ผู้ใช้ได้ทำการโปรแกรมไว้ โดยหลักการแปลจะใช้หลักการอ่านข้อมูลในแต่ละเรคคอร์ดและทำการเปรียบเทียบรหัสในฟิลด์ CODE เมื่อเปรียบเทียบค่าเท่ากันแล้วทำให้ได้รหัสใหม่ ซึ่งรหัสที่ได้ใหม่นี้จะเป็นตัวชี้ตำแหน่งของโปรแกรมย่อยของคำสั่งนั้น ๆ ส่วนตำแหน่งตารางข้อมูลที่อยู่ในเรคคอร์ดเดียวกันนั้นจะใช้เป็นพารามิเตอร์ในการส่งผ่านให้กับโปรแกรมย่อยดังกล่าว สำหรับจำนวนพารามิเตอร์ที่จะต้องทำการผ่านค่าไปให้นั้นขึ้นอยู่กับคำสั่งที่ทำการเปรียบเทียบได้ ส่วนการเปรียบเทียบจะสิ้นสุดเมื่อเป็นคำสั่ง END หรือรหัสในฟิลด์ CODE มีค่าเท่ากับ 129

ภายหลังการแปลเสร็จสิ้นจะได้รหัสใหม่ซึ่งเป็นชุดคำสั่งของหน่วยประมวลผลซึ่งรหัสที่ได้ใหม่นี้ถูกวางให้อยู่ในตำแหน่งพื้นที่ของโปรแกรมผู้ใช้เช่นเดิมโดยมีลักษณะดังนี้



จากที่ได้กล่าวมาแล้วว่าหน่วยประมวลผลกลางใช้ Z-80 เป็นไมโครโปรเซสเซอร์ ดังนั้นการผ่านค่าพารามิเตอร์ให้กับโปรแกรมย่อยจะใช้หลักการผ่านค่าไปกับรีจีสเตอร์ BC DE HL ตามลำดับซึ่งเป็นรีจีสเตอร์ของหน่วยประมวลผลที่ใช้ในการทดลอง ในบางคำสั่งที่มีการกำหนดตารางข้อมูลสูงสุดจะต้องใช้รีจีสเตอร์ถึง 3 รีจีสเตอร์ ในการผ่านค่าพารามิเตอร์เหล่านี้ เช่นคำสั่งทางคณิตศาสตร์ได้แก่ ADD SUB MUL DIV เป็นต้น

ตัวอย่างการสร้างรหัสเพื่อการผ่านค่าพารามิเตอร์ไปกับรีจีสเตอร์และรหัสเพื่อเรียกโปรแกรมย่อยของคำสั่ง ซึ่งรหัสที่ได้นั้นถูกแสดงเป็นรหัสภาษาแอสแซมบลี้ (ASSEMBLY)

โดยพิจารณาคำสั่งดังต่อไปนี้

```
LD      0500        คำสั่ง LD   พารามิเตอร์ที่ 1 เท่ากับ 0500
รหัสที่ได้          LD BC,0500      ;ตารางข้อมูลถูกใส่ค่าลงรีจีสเตอร์ BC
                    CALL LD_SUB     ;เรียกใช้โปรแกรมย่อย LD_SUB
ADD       01        คำสั่ง ADD  พารามิเตอร์ที่ 1 เท่ากับ 0001
          02                    พารามิเตอร์ที่ 2 เท่ากับ 0002
          06                    พารามิเตอร์ที่ 3 เท่ากับ 0006
รหัสที่ได้          LD BC,0001      ;ตารางข้อมูลถูกใส่ค่าลงรีจีสเตอร์ BC
                    LD DE,0002      ;ตารางข้อมูลถูกใส่ค่าลงรีจีสเตอร์ DE
                    LD HL,0006      ;ตารางข้อมูลถูกใส่ค่าลงรีจีสเตอร์ HL
                    CALL ADD_SUB    ;เรียกใช้โปรแกรมย่อย ADD_SUB
```

4.4 การทำงานของฟังก์ชั่นควบคุม

ฟังก์ชั่นควบคุมของเครื่องควบคุมที่โปรแกรมได้ ที่ใช้ในการควบคุมกระบวนการทางอุตสาหกรรมนั้น จะมีลักษณะของการทำงานคล้ายกับการทำงานของหน่วยประมวลผลทั่วไปที่มีคำสั่งการควบคุมแบบลอจิกเช่น AND OR การคำนวณทางคณิตศาสตร์เช่น ADD SUB MUL DIV การเคลื่อนย้ายข้อมูลเช่น MOV MVN เป็นต้น ด้วยเหตุนี้เองไมโครโปรเซส-เซอร์ จึงถูกเลือกเพื่อนำมาใช้เป็นหน่วยประมวลผลของเครื่องควบคุมที่โปรแกรมได้ ส่วนโปรแกรมที่สามารถทำให้ไมโครโปรเซสเซอร์ มีการทำงานเป็นอุปกรณ์ควบคุมหรือเครื่องควบคุมที่โปรแกรมได้ นั้นเป็นเรื่องที่ค่อนข้างจะซับซ้อน เนื่องจากว่าการทำงานมีลักษณะการซ้อนกันของโปรแกรมผู้ใช้กับโปรแกรมที่จะให้มีการปฏิบัติงานได้โดยตรง อีกทั้งฟังก์ชั่นการควบคุมกับคำสั่งที่ใช้ควบคุมการทำงานของไมโครโปรเซสเซอร์ก็มีความคล้ายคลึงกัน ซึ่งต่อไปจะเป็นการอธิบายการทำงานของฟังก์ชั่นควบคุมของเครื่องควบคุมที่โปรแกรมได้ในแต่ละฟังก์ชั่น ที่เป็นโครงสร้างทางซอฟร์แวร์ใช้ควบคุมไมโครโปรเซสเซอร์ เพื่อให้ไมโคร-โปรเซสเซอร์สามารถทำงานในคำสั่งต่าง ๆ โปรแกรมที่เขียนสำหรับฟังก์ชั่นควบคุมจะเป็นภาษาแอสแซมบลี้ของ Z80 (ดูภาคผนวก 2)

คำสั่ง LD

```
PROCEDURE LD_SUB (IOASSIGN)
BEGIN
    RESULT := DATATABLE [IOASSIGN];
    IF RESULT > 0 THEN RESULT := $FF ELSE RESULT := $00;
    PUSH RESULT TO STACK;
END;
```

คำสั่ง LD NOT

```
PROCEDURE LDNOT_SUB (IOASSIGN)
BEGIN
    LD_SUB (IOASSIGN);
    POP RESULT FROM STACK;
    COMPLEMENT RESULT;
    PUSH RESULT TO STACK;
END;
```

คำสั่ง AND

```
PROCEDURE AND_SUB (IOASSIGN);
BEGIN
    POP TEMPRESULT FROM STACK;
    LD_SUB (IOASSIGN);
    POP RESULT FROM STACK;
    RESULT := RESULT AND TEMPRESULT;
    PUSH RESULT TO STACK;
END;
```

คำสั่ง AND NOT

```
PROCEDURE ANDNOT_SUB (IOASSIGN);
BEGIN
    POP TEMPRESULT FROM STACK;
    LDNOT_SUB (IOASSIGN);
    POP RESULT FROM STACK;
    RESULT := RESULT AND TEMPRESULT;
    PUSH RESULT TO STACK;
END;
```

คำสั่ง OR

```
PROCEDURE OR_SUB (IOASSIGN);
BEGIN
    POP TEMPRESULT FROM STACK;
    LD_SUB (IOASSIGN);
    POP RESULT FROM STACK;
    RESULT := RESULT OR TEMPRESULT;
    PUSH RESULT TO STACK;
END;
```

คำสั่ง OR NOT

```
PROCEDURE ORNOT_SUB (IOASSIGN);
BEGIN
    POP TEMPRESULT FROM STACK;
    LDNOT_SUB (IOASSIGN);
    POP RESULT FROM STACK;
    RESULT := RESULT OR TEMPRESULT;
    PUSH RESULT TO STACK;
END;
```

คำสั่ง ANDLD

```
PROCEDURE ANDLD_SUB;
BEGIN
    POP TEMPRESULT FROM STACK;
    POP RESULT FROM STACK;
    RESULT := RESULT AND TEMPRESULT;
    PUSH RESULT TO STACK;
END;
```

คำสั่ง ORLD

```
PROCEDURE ORLD_SUB;
BEGIN
    POP TEMPRESULT FROM STACK;
    POP RESULT FROM STACK;
    RESULT := RESULT OR TEMPRESULT;
    PUSH RESULT TO STACK;
END;
```

คำสั่ง OUT

```
PROCEDURE OUT_SUB (IOASSIGN);
BEGIN
    POP RESULT FROM STACK;
    DATATABLE [IOASSIGN] := RESULT;
    PUSH RESULT TO STACK;
END;
```

คำสั่ง OUT NOT

```
PROCEDURE OUTNOT_SUB (IOASSIGN);
BEGIN
    POP RESULT FROM STACK;
    DATATABLE [IOASSIGN] := COMPLEMENT RESULT;
    PUSH RESULT TO STACK;
END;
```

คำสั่ง TIM

```
PROCEDURE TIM_SUB (TIMCH);
BEGIN
    POP RESULT FROM STACK;
    IF RESULT = $FF THEN
    BEGIN
        CHECK BIT14 OF REGISTERTABLE [TIMCH];
        IF BIT14 OF REGISTERTABLE [TIMCH] = SET THEN
            DECREMENT REGISTERTABLE [TIMCH];
        IF  BIT0 TO BIT 13 OF REGISTERTABLE  [TIMCH]  = 0 THEN
            SET  BIT15  OF REGISTERTABLE [TIMCH];
        PUSH RESULT TO STACK
    END
    ELSE
    BEGIN
        RESET BIT14,BIT15;
        REGISTERTABLE [TIMCH] := TIMEREGISTER;
        PUSH RESULT TO STACK;
    END;
END;
```

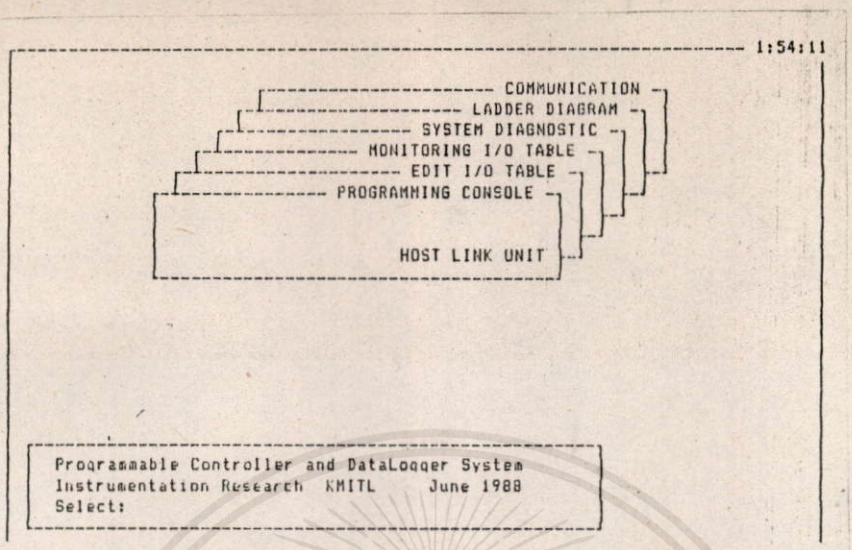
คำสั่ง CNT

```
PROCEDURE CNT_SUB (CNTCH);
BEGIN
    POP RESULT FROM STACK;
    IF RESULT = $FF THEN
    BEGIN
        CHECK BIT14 OF REGISTERTABLE [TIMCH];
        IF BIT14 OF REGISTERTABLE [TIMCH] = SET THEN
              DECREMENT REGISTERTABLE [TIMCH];
        IF  BIT0 TO BIT 13 OF REGISTERTABLE  [TIMCH]  = 0 THEN
              SET  BIT15  OF REGISTERTABLE [TIMCH];
        PUSH RESULT TO STACK
    END
    ELSE
    BEGIN
        RESET BIT14,BIT15;
        REGISTERTABLE [TIMCH] := TIMEREGISTER;
        PUSH RESULT TO STACK;
    END;
END;
```

บทที่ 5
การประยุกต์ใช้คอมพิวเตอร์กับเครื่องควบคุมที่โปรแกรมได้

การนำเครื่องควบคุมที่โปรแกรมได้ไปใช้ควบคุมกระบวนการนั้น ลักษณะของการควบคุมขึ้นอยู่กับโปรแกรมที่ผู้ใช้เขียนขึ้น การโปรแกรมเครื่องควบคุมนี้กระทำได้หลายวิธี เช่น โปรแกรมแบบบูลลีน (BOOLEAN INSTRUNCTION) การโปรแกรมแบบแลดเดอร์ไดอะแกรม (LADDER DIAGRAM) และอื่น ๆ การใช้แลดเดอร์ไดอะแกรมในการโปรแกรมเป็นการโปรแกรมที่มีประสิทธิภาพในการใช้งานมากที่สุด เนื่องจากมีความสะดวกรวดเร็วเห็นภาพพจน์ในการทำงาน ตลอดจนเห็นขั้นตอนของกระบวนการได้ดีกว่าการโปรแกรมแบบอื่น ๆ และผู้ใช้สามารถสังเกตุการแปรค่าหรือการเปลี่ยนแปลงค่าสภาวะต่างๆของกระบวนการนั้นได้ หรือในกรณีที่มีการผิดพลาดก็สามารถหาข้อผิดพลาดได้อย่างรวดเร็ว การโปรแกรมแบบแลดเดอร์ไดอะแกรมนี้ จะต้องมีส่วนแสดงผลที่สามารถแสดงแลดเดอร์ไดอะแกรม ได้ครั้งละหลาย ๆ บรรทัด ดังนั้นส่วนใหญ่จึงมักจะใช้จอภาพ (CRT) เป็นส่วนแสดงผลและใช้เครื่องคอมพิวเตอร์ในการประมวลผล นอกจากนั้นจะต้องสามารถเชื่อมต่อกับเครื่องควบคุมได้ด้วยมาตรฐานแบบต่างๆ เช่น RS-232C หรือ IEEE-488 เป็นต้น

สำหรับวิทยานิพนธ์ฉบับนี้ได้ประยุกต์ใช้คอมพิวเตอร์มาใช้ในการสร้างและแก้ไข รวมทั้งการแปลแลดเดอร์ไดอะแกรมให้เป็นชุดคำสั่งแบบบูลลีนหรือสร้างและแก้ไขชุดคำสั่งแบบบูลลีนให้เป็นชุดคำสั่งสำหรับเครื่องควบคุม โดยใช้เครื่องคอมพิวเตอร์ IBM PC เป็นตัวประมวลผลและเชื่อมต่อกับเครื่องควบคุมที่โปรแกรมได้ ทางพอร์ทแบบอนุกรม RS-232C ส่วนโปรแกรมที่ได้พัฒนาขึ้นสำหรับใช้ในการสร้างแก้ไข และประมวลผลต่าง ๆ บน IBM PC นี้ มีชื่อว่า "PCLINK" (Programmable Controller Linker) ซึ่งทำหน้าที่ในการสร้างและแก้ไขแลดเดอร์ไดอะแกรม ส่วนการโปรแกรมแบบบูลลีนนั้นได้ออกแบบให้ผู้ใช้ได้สามารถสร้างและแก้ไขได้จากจอภาพด้วยปากกาแสง (LIGHT PEN) หรือแป้นพิมพ์ โดยภาพที่แสดงบนจอจะมีลักษณะคล้ายกับแผงโปรแกรม (PROGRAMMING CONSOLE) ของเครื่องควบคุมที่โปรแกรมทั่ว ๆ ไป เพื่อให้ผู้ใช้ที่คุ้นเคยการโปรแกรมแบบบูลลีนกับเครื่องควบคุมที่โปรแกรมได้ทั่วไป สามารถทำการเขียนหรือพัฒนาโปรแกรมขึ้นได้โดยสะดวก



รูปที่ 5.1 แสดงการเลือกโหมดการใช้งานของโปรแกรม PCLINK

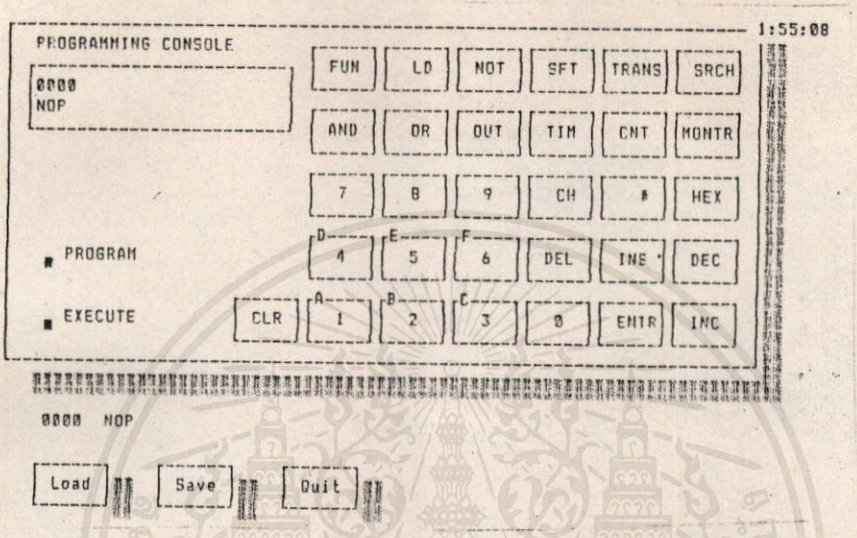
ในการออกแบบและพัฒนาโปรแกรม **PCLINK** ได้แบ่งการทำงานของโปรแกรมออกเป็นส่วนใหญ่ๆได้ 6 ส่วน ดังนี้

- **โปรแกรมสร้างและแก้ไขโปรแกรมแบบบูลลีน**
- **โปรแกรมสร้างและแก้ไขแลดเดอร์ไดอะแกรม**
- **โปรแกรมแปลแลดเดอร์ไดอะแกรมเป็นชุดคำสั่งแบบบูลลีนและชุดคำสั่ง**
- **โปรแกรมตรวจสอบและติดตามการทำงานหรือการเปลี่ยนแปลงสภาวะ**
- **โปรแกรมควบคุมการติดต่อสื่อสารระหว่างเครื่องไมโครคอมพิวเตอร์กับเครื่องควบคุมที่โปรแกรมได้**
- **โปรแกรมสนับสนุนอื่นๆ เช่น โปรแกรมจัดการการพิมพ์แลดเดอร์ไดอะแกรม**

ในบทที่ 5 จะกล่าวถึงเฉพาะหัวข้อของการสร้างแก้ไขโปรแกรมแบบบูลลีนและการสร้างแก้ไขโปรแกรมแบบแลดเดอร์ไดอะแกรม ส่วนการแปลความหมายของแลดเดอร์ไดอะแกรม การตรวจสอบการทำงานหรือติดตามการเปลี่ยนแปลงสภาวะและหัวข้อที่เหลือจะขอกล่าวในบทต่อไป

5.1 โปรแกรมสร้างและแก้ไขโปรแกรมแบบบูลลีน

โปรแกรมสร้างและแก้ไขโปรแกรมแบบบูลลีนมีหน้าที่สร้างรูปลักษณะของแป้นพิมพ์ขึ้นบนจอแสดงผลดังรูปที่ 5.2



รูปที่ 5.2 แสดงลักษณะของแป้นพิมพ์ที่ถูกสร้างขึ้นบนจอแสดงผล

หลังจากนั้นจะทำการรอรับคำสั่งการโปรแกรมจากผู้ใช้ทางแป้นพิมพ์หรือปากกาแสง ทำการถอดรหัสที่ได้มาพิจารณาว่าเป็นคำสั่งใด ซึ่งแต่ละคำสั่งมีหน้าที่หรือคุณสมบัติในการควบคุมการโปรแกรมอย่างย่อ ๆ ดังนี้

FUN : คำสั่งเลือกฟังก์ชั่นพิเศษ เช่น ฟังก์ชั่นทางคณิตศาสตร์ เป็นต้น

LD : คำสั่งเพื่อเรียกค่าสภาวะจากจากตำแหน่งที่ต้องการเข้ามา

NOT : คำสั่งเพื่อการทำให้ผลลัพธ์เป็นตรงข้าม

SFT : คำสั่งเพื่อการเลื่อนบิท (BIT) ของเอาท์พุท

TRANS : คำสั่งเพื่อการรับส่งข้อมูลระหว่างคอมพิวเตอร์กับเครื่องควบคุม

SRCH : คำสั่งเพื่อการค้นหาชุดคำสั่งที่ต้องการ

AND : คำสั่งเพื่อเรียกค่าสภาวะที่ต้องการมากระทำลอจิก AND กับผลลัพธ์ก่อน

OR : คำสั่งเพื่อเรียกค่าสภาวะที่ต้องการมากระทำลอจิก OR กับผลลัพธ์ก่อน

OUT : คำสั่งเพื่อส่งค่าผลลัพธ์หลังการกระทำทางลอจิกออกทางเอาท์พุทที่ต้องการ

TIM : คำสั่งเพื่อกำหนดค่าเวลาของตัวตั้งเวลาที่ต้องการ

CNT : คำสั่งเพื่อกำหนดค่าการนับของตัวนับที่ต้องการ

MONTR : คำสั่งเพื่อการเรียกดูค่าสภาวะขณะควบคุม

DEL : คำสั่งเพื่อการลบชุดคำสั่งออกจากหน่วยความจำ

INS : คำสั่งเพื่อการแทรกชุดข้อมูลเข้าในหน่วยความจำ

DEC : คำสั่งเพื่อการเรียกดูชุดคำสั่งตำแหน่งก่อนหน้า

CLR : คำสั่งเพื่อการลบชุดคำสั่ง

ENTR : คำสั่งเพื่อการใส่ชุดคำสั่งลงหน่วยความจำ

INC : คำสั่งเพื่อการเรียกดูชุดคำสั่งตำแหน่งถัดมา

ชุดคำสั่งแต่ละชุดที่ถูกโปรแกรมเหล่านี้จะถูกส่งให้เครื่องควบคุม นำไปแปลความหมาย เพื่อใช้ในการควบคุมเครื่องจักรหรือกระบวนการอีกทีหนึ่ง มีการจัดวางข้อมูลก่อนที่จะทำการส่งให้เครื่องควบคุม ซึ่งชุดข้อมูลดังกล่าวมีลักษณะตามโครงสร้างดังนี้

```
CommandPack     =  Record
                       InsAddress    : Word;
                       Opcode        : Byte;
                       DataType1     : Byte;
                       IOAssign1     : Word;
                       DataType2     : Byte;
                       IOAssign2     : Word;
                       DataType3     : Byte;
                       IOAssign3     : Word;
                   End;

PackCommand     =  Array [1..2000] of CommandPack
```

CommandPack เป็นตัวแปรแบบเรคคอร์ด (Record) ภายในเรคคอร์ดประกอบด้วยฟิลด์ (Field) ขนาด 8 ฟิลด์ดังนี้

5.1.1 ฟิลด์ InsAddress เป็นข้อมูลแบบเวิร์ด (WORD) สำหรับวางค่าลำดับของชุดคำสั่ง

5.1.2 ฟิลด์ Opcode เป็นข้อมูลเลขจำนวนเต็มแบบไบต์ (Byte) สำหรับวางรหัสของชุดคำสั่ง

5.1.3 ฟิลด์ DataType1 เป็นข้อมูลเลขจำนวนเต็มแบบไบต์ สำหรับแสดงชนิดข้อมูลของฟิลด์ IOAssign1 ว่าเป็นแบบเลขฐานสิบหก (Hexadecimal) หรือเลขฐานสิบ (Decimal)

5.1.4 ฟิลด์ IOAssign1 เป็นข้อมูลแบบเวิร์ด สำหรับแสดงตำแหน่งของอินพุทเอาท์พุทในตารางข้อมูล ข้อมูลที่ 1

5.1.5 ฟิลด์ DataType2 เป็นข้อมูลเลขจำนวนเต็มแบบไบต์ สำหรับแสดงชนิดข้อมูลของฟิลด์ IOAssign2 ว่าเป็นแบบเลขฐานสิบหก หรือเลขฐานสิบ

5.1.6 ฟิลด์ IOAssign2 เป็นข้อมูลแบบเวิร์ด สำหรับแสดงตำแหน่งของอินพุทเอาท์พุทในตารางข้อมูล ข้อมูลที่ 2

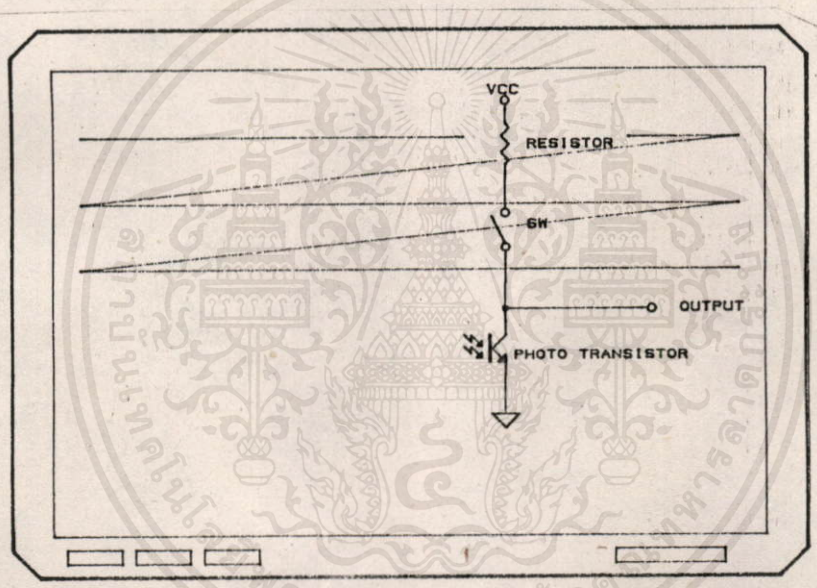
5.1.7 ฟิลด์ DataType3 เป็นข้อมูลเลขจำนวนเต็มแบบไบต์ สำหรับแสดงชนิดข้อมูลของฟิลด์ IOAssign3 ว่าเป็นแบบเลขฐานสิบหก หรือเลขฐานสิบ

5.1.8 ฟิลด์ IOAssign3 เป็นข้อมูลแบบเวิร์ด สำหรับแสดงตำแหน่งของอินพุทเอาท์พุทในตารางข้อมูล ข้อมูลที่ 3

PackCommand เป็นตัวแปรแบบอาเรย์ซึ่งใช้สำหรับเก็บข้อมูลแบบ CommandPack ดังที่กล่าวมาแล้วทั้งหมดได้จำนวน 2000 เรคคอร์ด ซึ่งหมายความว่าชุดคำสั่งจะถูกโปรแกรมได้ 2000 ชุดคำสั่ง หลังจากทำการโปรแกรมเรียบร้อยแล้วข้อมูลทั้งหมดสามารถนำเก็บในแผ่นจานแม่เหล็ก ในทำนองเดียวกันก็สามารถที่จะเรียกโปรแกรมในแผ่นจานแม่เหล็กมาใช้งานได้ หรือเมื่อต้องการส่งข้อมูลเหล่านี้ให้กับเครื่องควบคุมทำได้โดยการใช้คำสั่ง TRANS ข้อมูลดังกล่าวก็จะถูกส่ง ไปในขณะเดียวกันเครื่องควบคุมก็จะทำการรับข้อมูลและนำเก็บไว้ในหน่วยความจำพื้นที่ของโปรแกรมผู้ใช้ต่อไป ส่วนการใช้ปากกาแสงในการควบคุมการโปรแกรมเป็นการสะดวกมากกว่า เนื่องจากสามารถนำปากกาแสงไปรับแสงที่จอภาพแสดงผลได้โดยตรง เมื่อมีการกวาดของแสงมาที่ตำแหน่งบนจอภาพที่มีปากกาแสงรับแสงอยู่อุปกรณ์ภายในปากกาแสงก็จะมีการรับรู้และเปลี่ยนแปลง

ภายในปากกาแสงประกอบด้วยโฟโต้ทรานซิสเตอร์ (PHOTO TRANSISTOR) และสวิทช์ เมื่อถูกแสงในขณะที่สวิทช์ปิดวงจรทำให้มีกระแสไฟฟ้าผ่านจากปากกาแสงเข้ามาที่วงจรตรวจจับและสามารถรับรู้ว่ามีแสงเข้ามาที่ปากกา โปรแกรมก็ทำการอ่านตำแหน่งจากจอภาพว่าเป็นตำแหน่งใดและนำมาเข้ารหัส เพื่อให้ได้คำสั่งที่ปากกาแสงกำลังรับ

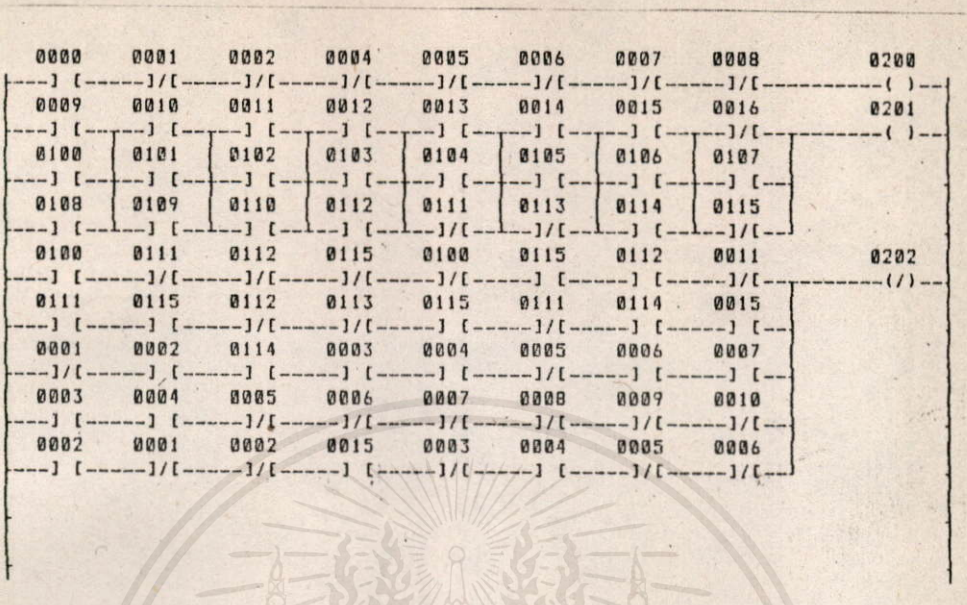
แสงอยู่ โดยปกติเครื่องคอมพิวเตอร์ที่มีการแสดงผลบนจอภาพแสดงผลจะมีวงจรควบคุมการแสดงผล หรือ CRT CONTROLLER และวงจรควบคุมการแสดงผลก็จะมีวงจรตรวจจับการรับสัญญาณจากปากกาแสงติดตั้งอยู่แล้ว ในส่วนของการทำงานนั้นโปรแกรมย่อยจัดการอินพุทเอาท์พุท (BIOS) จะถูกจัดเตรียมไว้เพื่อให้ภายนอกเรียกโดยให้ค่าตำแหน่งจากจอภาพที่ตรวจจับได้ออกมาเป็นเอาท์พุทของโปรแกรมย่อยดังกล่าว ซึ่งขอบเขตของจอภาพแสดงผลมีขนาด 80 อักขระต่อบรรทัด เป็นจำนวน 25 บรรทัด



รูปที่ 5.3 แสดงหลักการทำงานของปากกาแสง

5.2 การสร้างและแก้ไขแลดเดอร์ไดอะแกรม

โครงสร้างของข้อมูลแลดเดอร์ไดอะแกรมจะถูกสร้างขึ้นด้วยอักขระ (CHARACTER) โดยการจัดวางอักขระที่มีรูปร่างคล้ายกับจุดต่อ เส้นตรง หน้าสัมผัสของรีเลย์ บล๊อคฟังก์ชั่น เอาท์พุท เป็นต้น นำมาประกอบกันขึ้นให้มีลักษณะเป็น แลดเดอร์ไดอะแกรม ดังรูปที่ 5.4



รูปที่ 5.4 แสดงอักขระที่มีการจัดวางเป็นแลดเดอร์ไดอะแกรม

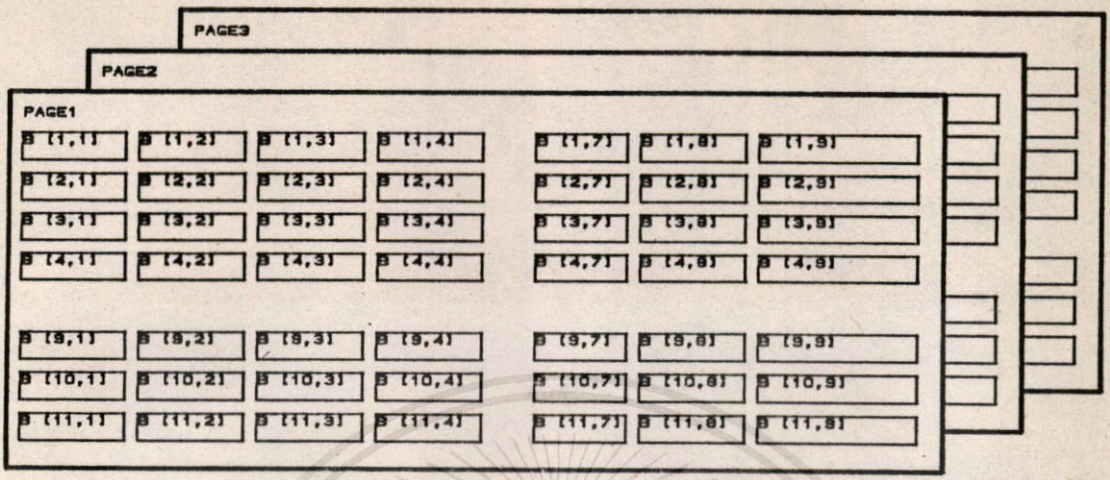
ตำแหน่งและการจัดวาง หน้าสัมผัสรีเลย์ ทั้งแบบปกติเปิดและแบบปกติปิด จุดต่อเอาท์พุท ตลอดจนบล๊อคฟังก์ชั่น ที่แสดงผลบนจอแสดงผลนั้นถูกจัดวางให้อยู่ในตำแหน่งที่โปรแกรมจัดการสามารถรับรู้ได้ เนื่องจากตำแหน่งต่างๆ มีผลต่อโปรแกรมการแปลแลดเดอร์ไดอะแกรม ซึ่งจะกล่าวต่อไปในบทที่ 6 (โปรแกรมแปลความหมายแลดเดอร์ไดอะแกรม)

ข้อมูลแลดเดอร์ไดอะแกรมที่ถูกโปรแกรมจากผู้ใช้ทั้งหมด แบ่งออกเป็นหน้าแต่ละหน้าจะมีขนาดเท่ากับจอแสดงผลขนาด 80 อักขระต่อบรรทัด จำนวน 24 บรรทัด ที่ถูกจัดแบ่งย่อยเป็นบล๊อคขนาด 99 บล๊อค วางเป็นแมตริก (MATRIX) ขนาด 11 * 9 โดยเริ่มจาก

B[1,1] B[1,2] B[1,3] ..... B[1,9]

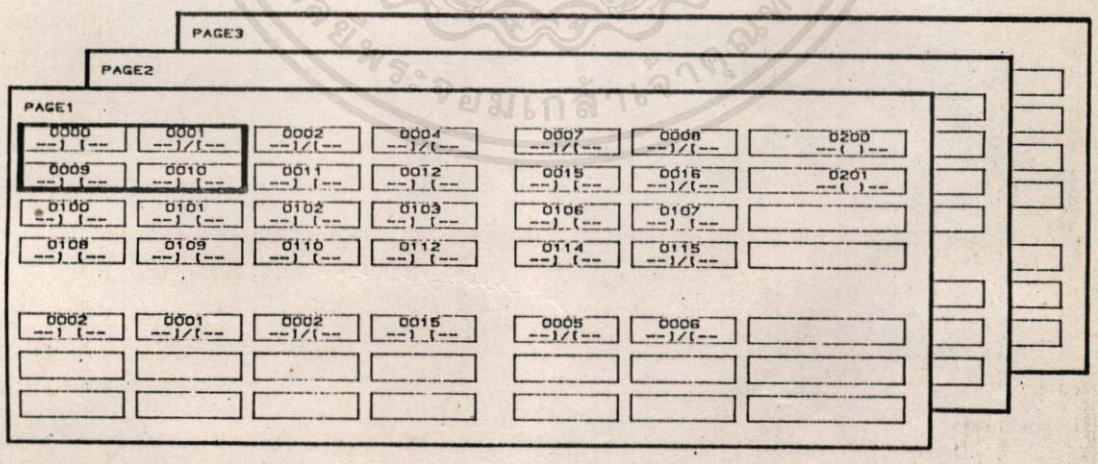
B[2,1] B[2,2] B[2,3] ..... B[2,9]

B[11,1] B[11,2] B[11,3] ..... B[11,9]



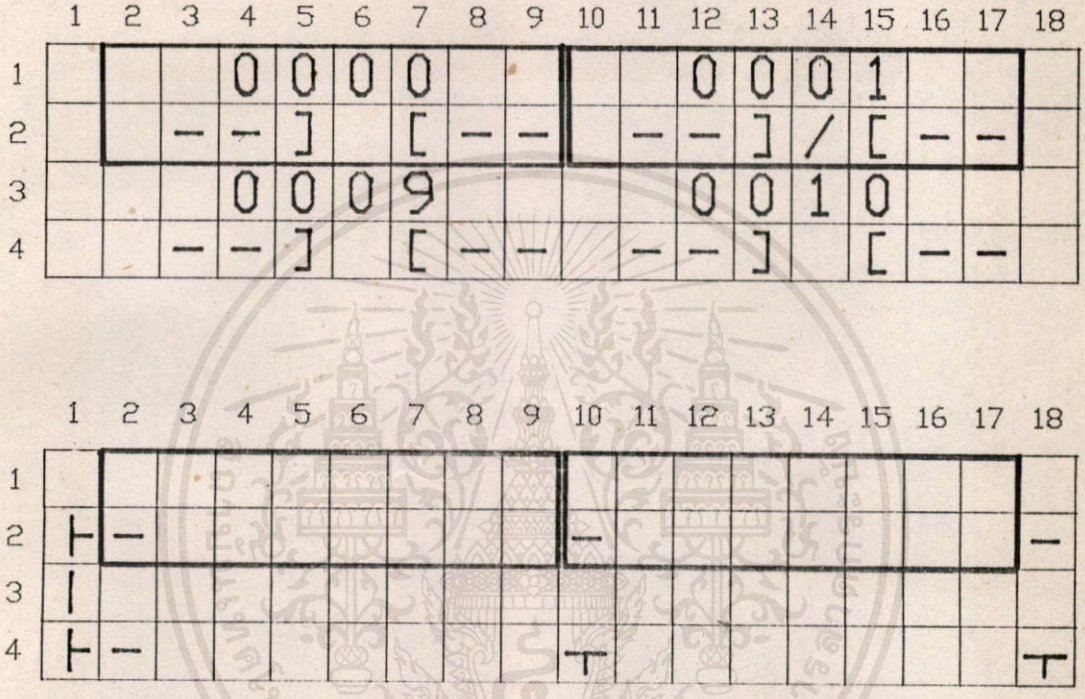
รูปที่ 5.5 แสดงการจัดข้อมูลในแต่ละหน้าออกเป็นบล๊อค

เพื่อจัดวางหน้าสัมผัสรีเลย์ เอาท์พุท บล๊อคฟังก์ชั่น หมายเลขกำหนดตำแหน่ง ให้อยู่ใน บล๊อคได้ ดังแสดงในรูปที่ 5.6



รูปที่ 5.6 แสดงการจัดวางข้อมูลแลดเดอร์ไดอะแกรม

พิจารณาการจัดวางอักขระในแต่ละบล๊อคโดยนำบล๊อคที่ [1,1] [1,2] [2,1] และ [2,2] ของรูปที่ 5.6 มาแสดงโดยแต่ละบล๊อคถูกแบ่งให้มีขนาด 8 อักขระต่อบรรทัด จำนวน 2 บรรทัด ดังรูปที่ 5.7 และทำการแยกเพื่อให้เห็นชัดระหว่างอักขระของหน้าสัมผัสกับอักขระของจุดเชื่อมต่อซึ่งปกติจะอยู่ภายในบล๊อคเดียวกัน



รูปที่ 5.7 แสดงข้อมูลหน้าสัมผัสแบบปกติเปิดและหมายเลขกำหนด ที่ถูกวางในบล๊อค [1,1]

ซึ่งอักขระในบล๊อคที่แสดงในรูปที่ 5.7 ประกอบด้วย ชุดอักขระของหมายเลขกำหนดตำแหน่งที่วางอยู่เหนือชุดอักขระของหน้าสัมผัสแบบปกติเปิด มีรายละเอียดดังนี้

ตำแหน่ง	อักขระ	ตำแหน่ง	อักขระ
1,2	SPACE [$20]	2,2	- [$C4]
1,3	SPACE [$20]	2,3	- [$C4]
1,4	0 [$30]	2,4	- [$C4]

1,5	0 [$30]	2,5	] [$5D]
1,6	0 [$30]	2,6	SPACE [$20]
1,7	0 [$30]	2,7	[[$5B]
1,8	SPACE [$20]	2,8	- [$C4]
1,9	SPACE [$20]	2,9	- [$C4]

การสร้างแลดเดอร์ไดอะแกรม ขึ้นบนจอภาพแสดงผลจะใช้แป้นพิมพ์ของเครื่องคอมพิวเตอร์ในการควบคุมตำแหน่งการวางหน้าสัมผัสรีเลย์ เอาท์พุท บล๊อคฟังก์ชั่นต่าง ๆ ตลอดจนการลากเส้นเชื่อมต่อทั้งแนวนอนและแนวตั้ง การเปลี่ยนหน้าเพื่อสร้างแลดเดอร์ไดอะแกรม การเก็บข้อมูลแลดเดอร์ไดอะแกรมลงในแผ่นจานแม่เหล็กและนำข้อมูลแลดเดอร์ไดอะแกรมจากแผ่นจานแม่เหล็กมาใช้ การแก้ไข การลบ การแทรก สามารถทำได้โดยสะดวกและมองเห็นภาพพจน์ได้ดีกว่าการโปรแกรมแบบบูลลีนในหัวข้อ 5.1

การใช้แป้นพิมพ์เพื่อสร้างแลดเดอร์ไดอะแกรมจะใช้ฟังก์ชั่นคีย์ F1 ถึง F10 ดังนี้

F1 ใช้กำหนดหน้าสัมผัสรีเลย์แบบปกติเปิดในตำแหน่งที่ตัวชี้ (CURSOR)

F2 ใช้กำหนดหน้าสัมผัสรีเลย์แบบปกติปิดในตำแหน่งที่ตัวชี้

F3 ใช้กำหนดเอาท์พุทแบบปกติไม่ทำงานที่ตำแหน่งเอาท์พุท

F4 ใช้กำหนดเอาท์พุทแบบปกติทำงานที่ตำแหน่งเอาท์พุท

F5 ใช้กำหนดตัวตั้งเวลาที่ตำแหน่งเอาท์พุท

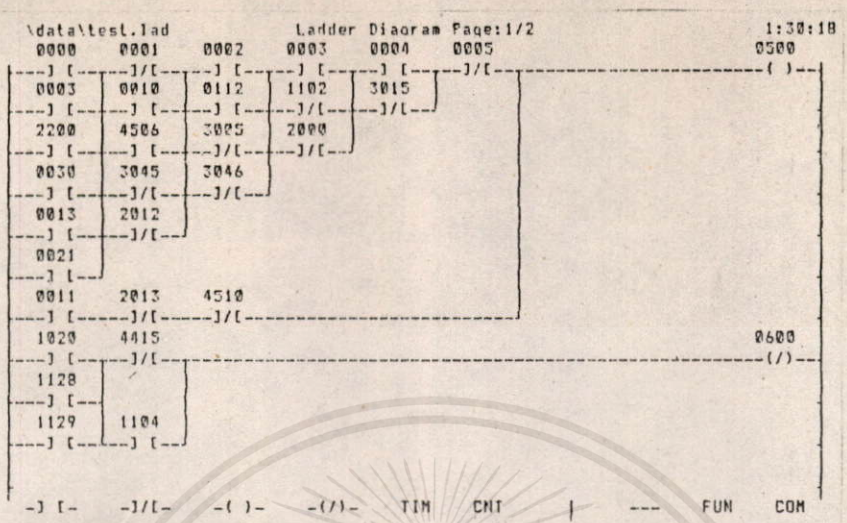
F6 ใช้กำหนดตัวนับที่ตำแหน่งเอาท์พุท

F7 ใช้ลากเส้นแนวตั้งเพื่อเป็นจุดต่อที่ตำแหน่งตัวชี้

F8 ใช้ลากเส้นแนวนอนที่ตำแหน่งตัวชี้

F9 ใช้เรียกฟังชั่นการควบคุมต่าง ๆ ที่ตำแหน่งเอาท์พุท

F10 ใช้เรียกคำสั่งย่อยเพื่อการทำงานในโหมดต่าง ๆ



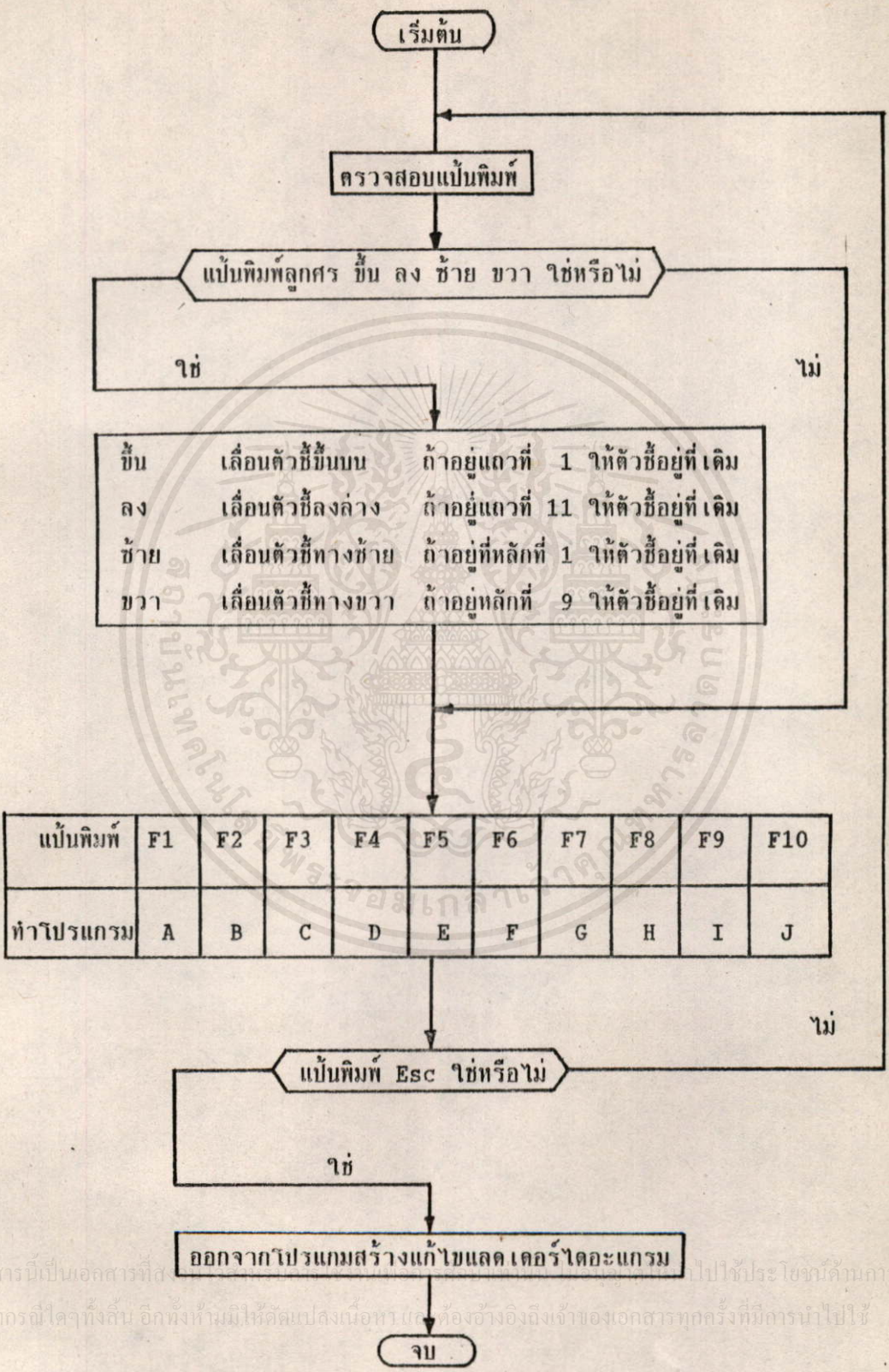
รูปที่ 5.8 แสดงตัวอย่างแลดเดอร์ไดอะแกรมที่ทำการสร้างขึ้น

จากที่กล่าวมาแล้วการสร้างและแก้ไขโปรแกรมแบบแลดเดอร์ไดอะแกรม จะใช้หลักการนำเอาอักขระที่มีลักษณะคล้ายกับหน้าสัมผัสรีเลย์ เอาท์พุท จุดเชื่อมต่อต่าง ๆ นำมาประกอบกันขึ้นเป็นแลดเดอร์ไดอะแกรม ซึ่งอักขระดังกล่าวได้ถูกจัดเตรียมไว้เป็นกลุ่ม เช่น

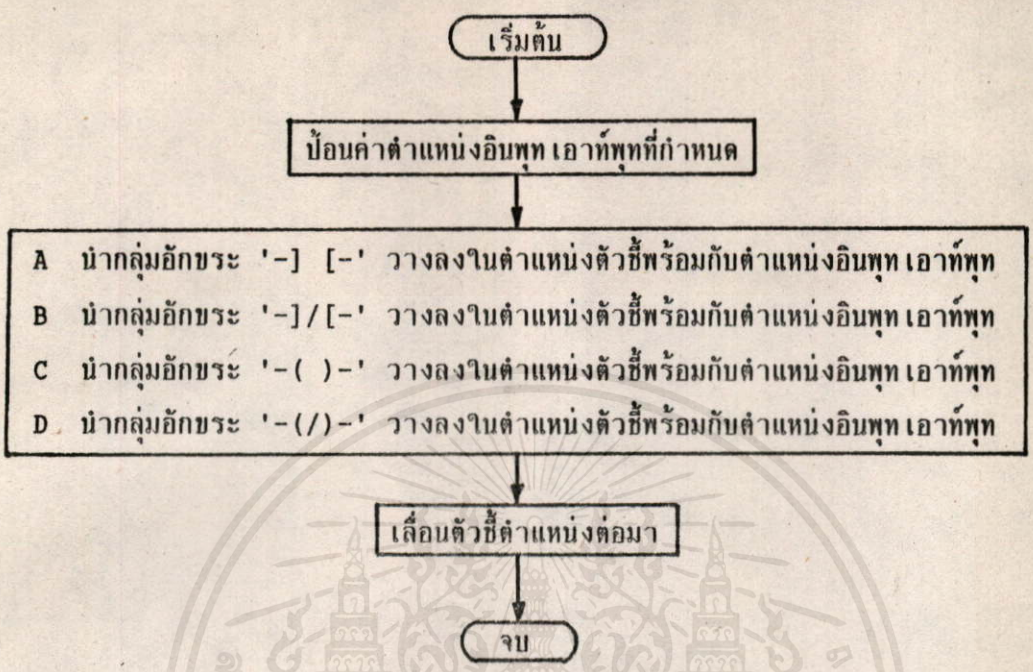
'-] [-' และ '-]/[-' แสดงกลุ่มอักขระของหน้าสัมผัสแบบปกติ เปิดและปิด
'-( )-' และ '-(/)-' แสดงกลุ่มอักขระของเอาท์พุทแบบปกติและตรงข้าม

และพร้อมที่จะถูกนำมาวางในตำแหน่งตัวชี้ (CURSOR) ที่ปรากฏอยู่บนจอภาพแสดงผล โดยผู้ใช้หรือผู้ปฏิบัติงานเป็นผู้กำหนดด้วยการใช้แป้นพิมพ์ตัวลูกศรเลื่อนตำแหน่งตัวชี้ ซ้าย ขวา ขึ้น ลง เมื่อได้ตามที่ต้องการแล้วจึงเรียกกลุ่มอักขระด้วยการใช้ฟังก์ชั่นคีย์ พร้อมทั้งกำหนดตำแหน่งอินพุทเอาท์พุท และนำวางลงบนจอภาพแสดงผลต่อไป การเคลื่อนที่ของตัวชี้จะมีการเคลื่อนที่ในตำแหน่งของบล็อคที่ถูกกำหนด (รูปที่ 5.5) การสร้างและแก้ไขแลดเดอร์ไดอะแกรมมีการทำงานตามผังงานดังนี้

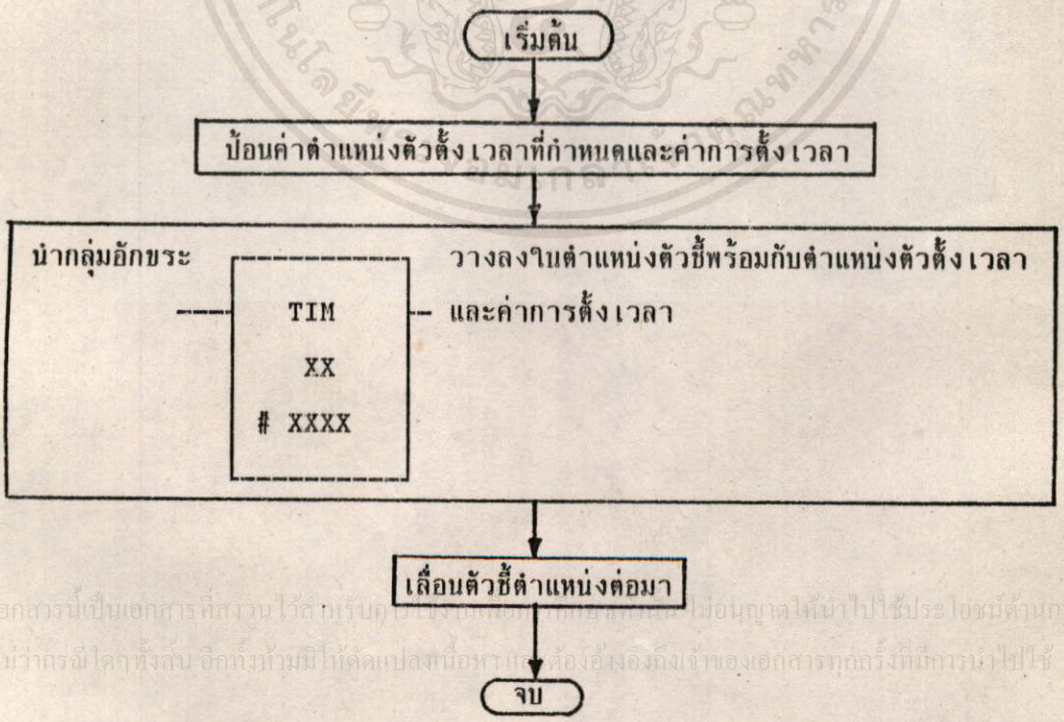
โปรแกรมหลักการสร้างแก้ไขแลดเดอร์ไดอะแกรม



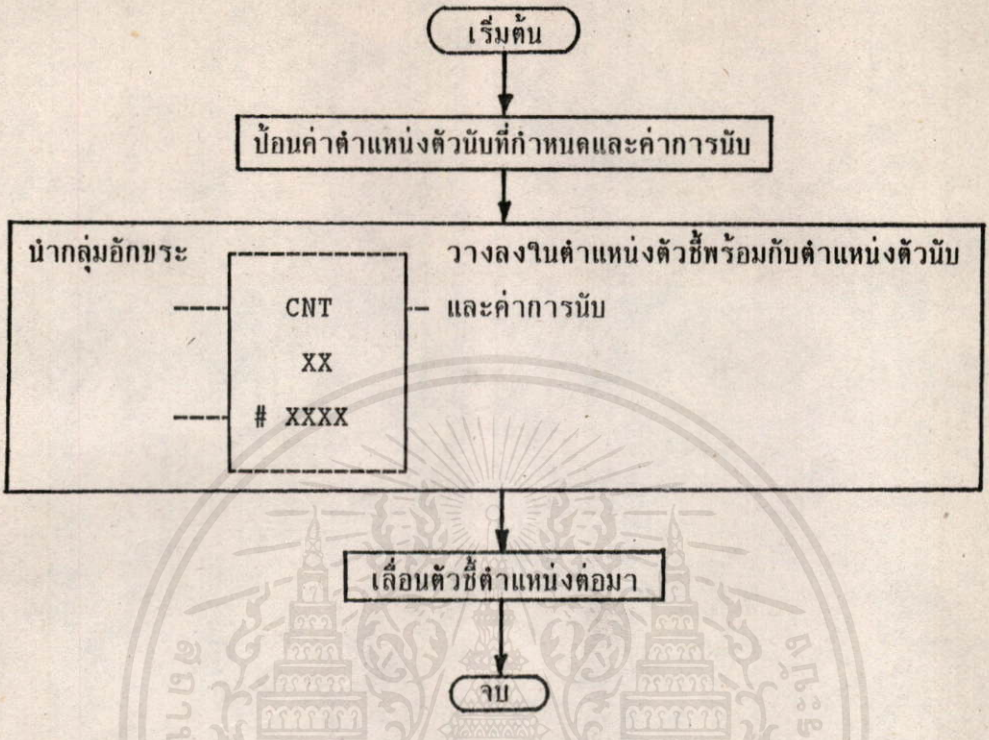
โปรแกรม A,B,C,D นำกลุ่มอักขระมาวางที่ตำแหน่งตัวชี้



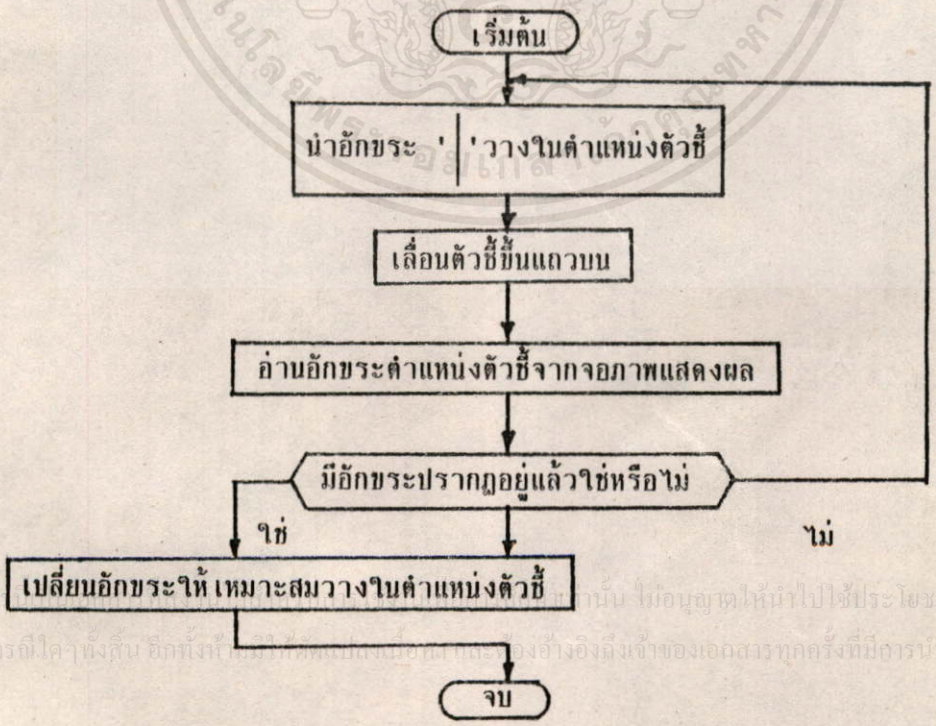
โปรแกรม E นำกลุ่มอักขระของตัวตั้งเวลามาวางที่ตำแหน่งตัวชี้



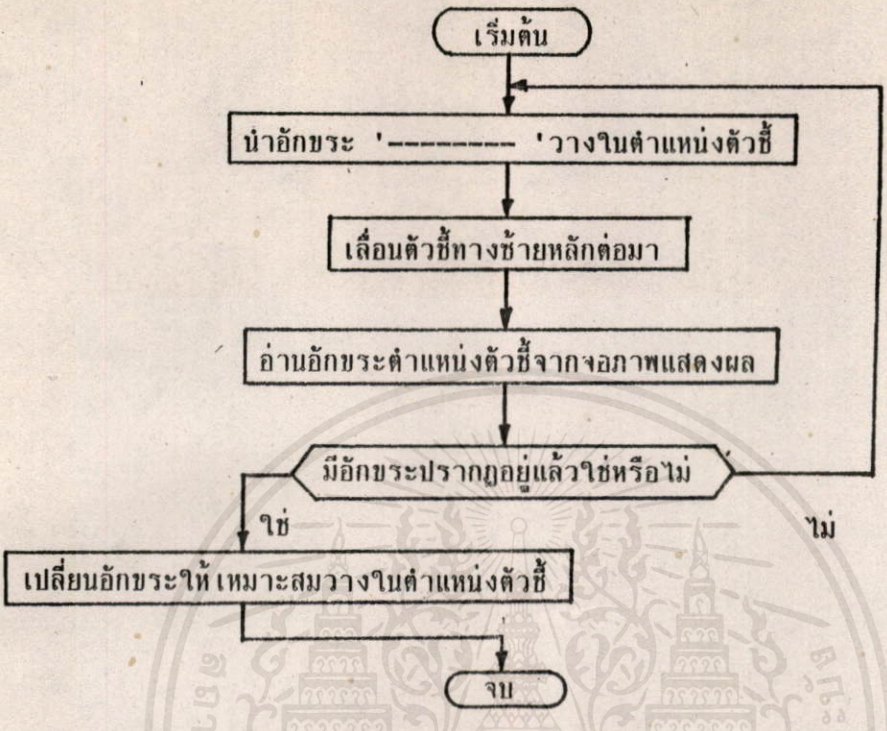
โปรแกรม F นำกลุ่มอักขระของตัวนับมาวางที่ตำแหน่งตัวชี้



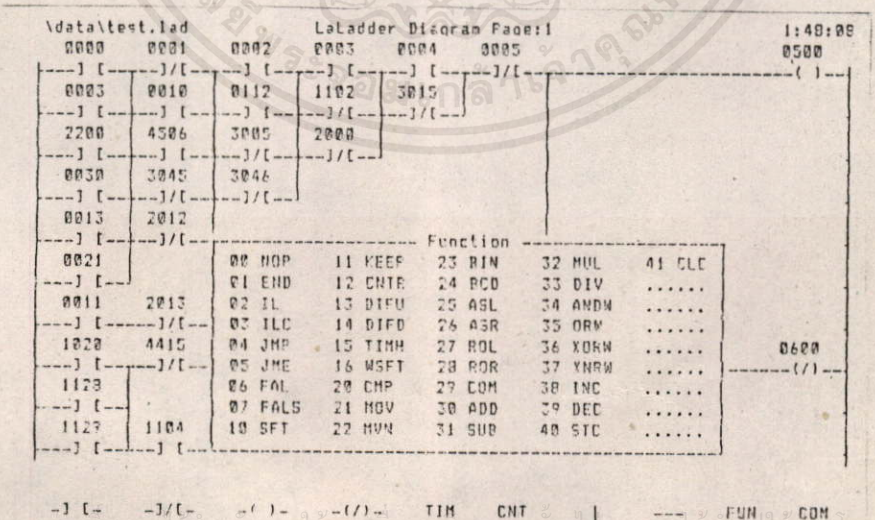
โปรแกรม G ทำการลากเส้นทางแนวตั้ง



โปรแกรม H ทำการลากเส้นทางแนวนอน

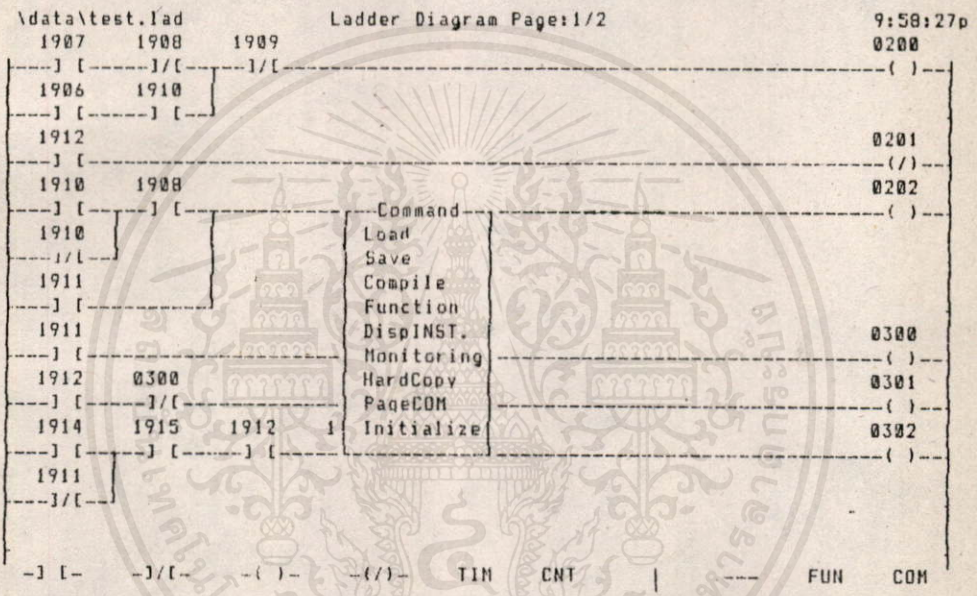


โปรแกรม I ทำการนำฟังก์ชั่นการควบคุมต่าง ๆ ที่มีอยู่ออกมาแสดงผลแบบหน้าต่าง (WINDOW DISPLAY) ซ้อนข้อมูลในจอภาพแสดงผลที่มีอยู่เดิม โดยการเก็บข้อมูลที่มีอยู่บนจอภาพไว้หลักจากเสร็จสิ้นจึงนำเข้ามาตามเดิม



รูปที่ 5.9 แสดงการเรียกฟังชั่นการควบคุมขึ้นมาแสดงผล

โปรแกรม J แสดงผลเมนูเพื่อเลือกคำสั่งการทำงานร่วมกับการสร้างและแก้ไขแลดเดอร์ไดอะแกรม ซึ่งเป็นคำสั่งที่ใช้ในการเรียกหรือเก็บบันทึกระหว่างเครื่องคอมพิวเตอร์กับหน่วยขับแผ่นจานแม่เหล็ก (DISK DRIVE UNIT) คำสั่งสำหรับการแปลแลดเดอร์ไดอะแกรมให้เป็นชุดข้อมูลคำสั่งแบบบูลลีน เป็นต้น ซึ่งคำสั่งเหล่านี้ได้อธิบายหลักการทำงานไว้ในบทที่ 6 (การแปลความหมายโปรแกรมแบบแลดเดอร์ไดอะแกรม) และบทที่ 7 (การแสดงผลและติดตามค่าสภาวะการทำงาน) ส่วนความหมายของคำสั่งต่าง ๆ มีดังนี้



รูปที่ 5.9 แสดงคำสั่งย่อยเพื่อการทำงานในโหมดต่าง ๆ

Load สำหรับนำข้อมูลแลดเดอร์ไดอะแกรมจากแผ่นจานแม่เหล็กมาตรวจสอบหรือใช้งาน โดยการกำหนดชื่อแฟ้มข้อมูลตามไป

Save สำหรับนำข้อมูลแลดเดอร์ไดอะแกรม เก็บลงในแผ่นจานแม่เหล็กด้วยชื่อแฟ้มข้อมูลที่กำหนด

Compile สำหรับทำการแปลความหมายของข้อมูลแลดเดอร์ไดอะแกรมทั้งหมดให้เป็นโปรแกรมแบบบูลลีนก่อนการส่งให้เครื่องควบคุม

Function สำหรับเลือกดูฟังก์ชั่นการควบคุมที่ต้องการ

DispINST สำหรับตรวจสอบความถูกต้องของโปรแกรมแบบบูลลีนที่ได้หลังจากการแปลความหมาย

Monitoring สำหรับตรวจสอบการทำงานของ เครื่องควบคุมโดยแสดงผลการ เปลี่ยนแปลงแบบแลด เดอร์ไดอะแกรม

HardCopy สำหรับสั่งการพิมพ์ข้อมูลแลด เดอร์ไดอะแกรมด้วยแฟ้มข้อมูลที่กำหนดพร้อมกับโปรแกรมบูลลีนทาง เครื่องพิมพ์

PageCom สำหรับควบคุมการจัดหน้าสำหรับการสร้างแก้ไขแลด เดอร์ไดอะแกรม

Transfer สำหรับส่งข้อมูลแบบบูลลีนที่ทำการแปลความหมายจากแลด เดอร์ไดอะแกรมไปให้กับ เครื่องควบคุม

บทที่ 6
การแปลความหมายโปรแกรมแบบแลดเดอร์ไดอะแกรม

บทนี้จะกล่าวถึง หลักการตรวจสอบข้อมูลแบบแลดเดอร์ไดอะแกรม การเก็บค่าพารามิเตอร์ที่จำเป็นของข้อมูลเพื่อนำมาทำการแปล และการเข้าคำสั่ง โดยจะอธิบายเป็นหัวข้อดังนี้

- การตรวจสอบข้อมูลแลดเดอร์ไดอะแกรม
- การแปลความหมายหลังจากตรวจสอบ

6.1 การตรวจสอบข้อมูลแบบแลดเดอร์ไดอะแกรม

การตรวจสอบข้อมูลแบบแลดเดอร์ไดอะแกรม จะใช้หลักการสแกนข้อมูลในแต่ละหน้าของแลดเดอร์ที่กำลังแสดงผลบนจอภาพแสดงผลในขณะนั้น โดยการอ่านเฉพาะอักขระข้อมูลในตำแหน่งที่จอภาพแสดงผล ทีละบรรทัดจนครบในหน้านั้น ๆ ทำให้สามารถรับรู้ได้ว่ามีชนิดของหน้าสัมผัส เอาท์พุท เป็นแบบปกติปิดหรือแบบปกติเปิด มีจุดต่อ (NODE) ของหน้าสัมผัสและการกำหนดตำแหน่งอินพุทเอาท์พุทเป็นอย่างไร และทำการเก็บข้อมูลที่จำเป็นนั้นไว้ในตัวแปรแบบอาเรย์ขนาดสองมิติ แบ่งเป็นหัวข้อดังนี้

6.1.1 การอ่านอักขระที่จอภาพแสดงผล จะใช้หลักการเรียกโปรแกรมย่อยในระบบจัดการอินพุทเอาท์พุท (BIOS) ของเครื่องไมโครคอมพิวเตอร์ มีวิธีการดังนี้

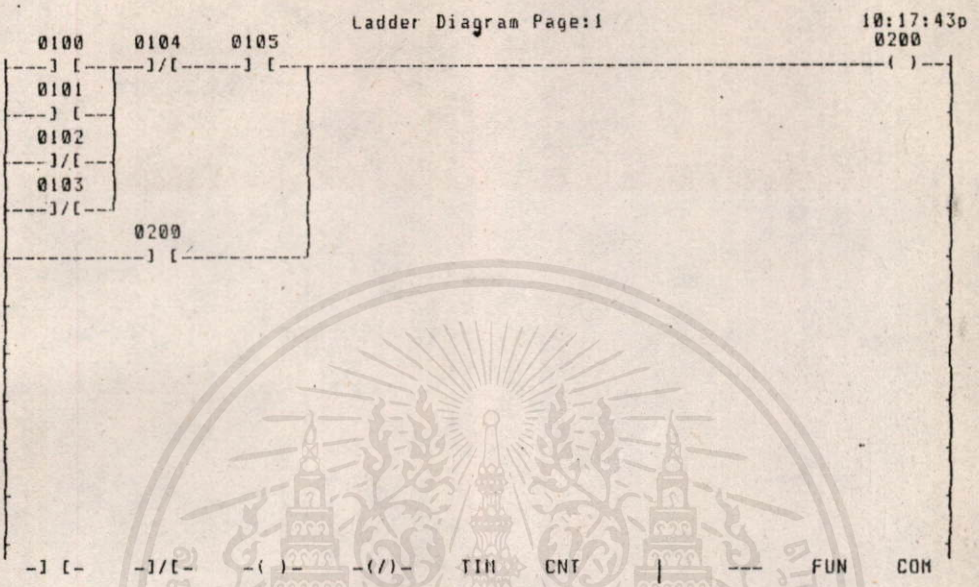
```
Type RegPack  =
        Record Case Integer of
            1   :   (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags  : Integer);
            2   :   (AL,AH,BL,BH,CL,CH,DL,DH  : Byte);
        End;
Var  Regs        :  RegPack;
     Cursor_CH   :  Byte;
Procedure GetChar (Var Cursor_ch : Byte);
begin
```

```
    Regs.BH   :=  $00     { Active Screen Number }
    Intr ($10,Dos.Registers(Regs));
    Cursor_CH :=  Regs.AL;
End;
```

โดยการกำหนดชนิดของตัวแปรและตัวแปรที่ใช้ก่อน และทำการเรียกใช้โปรแกรม GetChar ซึ่งเป็นโปรแกรมที่จะทำการอ่านอักขระบนจอภาพแสดงผลในตำแหน่งที่ตัวชี้วางอยู่ โปรแกรมดังกล่าวจะทำการใส่ค่าพารามิเตอร์ที่แสดงอันดับที่ของจอภาพลงใน Regs.BH จากนั้นโปรแกรมจะทำการเรียกโปรแกรมย่อยในระบบจัดการอินพุทเอาท์พุท และค่าอักขระที่ทำการอ่านก็ถูกผ่านค่ากลับมาใน Regs.AL ใส่เข้าที่ตัวแปร Cursor_CH ซึ่งเป็นการอ่านค่าอักขระจากจอภาพแสดงผลที่ง่าย เนื่องจากโปรแกรมจัดการอินพุทเอาท์พุทของเครื่องไมโครคอมพิวเตอร์ถูกทำหนดให้รู้ว่าอักขระใดแสดงผลอยู่บนจอภาพแสดงผลบ้าง

6.1.2 หลักการสแกน โดยใช้หลักการสแกนอ่านอักขระข้อมูลภายในแต่ละบล็อค (ที่แสดงในรูป 5.6) ทีละบรรทัดจนครบหน้า โดยจากบล็อคที่ 1 ถึง 8 จะทำการอ่านในตำแหน่งที่แสดงลักษณะเฉพาะบ่งบอกชนิดของหน้าสัมผัส เช่น อักขระ " / " และลักษณะของจุดเชื่อมต่อของหน้าสัมผัส เมื่อสแกนจนถึงบล็อคที่ 9 ก็จะทำการอ่านในลักษณะเดียวกันแต่ความหมายจะเป็นเอาท์พุทเท่านั้น ในการอ่านลักษณะเฉพาะของหน้าสัมผัสและเอาท์พุทในแต่ละบล็อคและจุดเชื่อมต่อหน้าสัมผัสนั้น จะมีการอ่านอักขระที่กำหนดอินพุทเอาท์พุทไปด้วย โดยใช้วิธีการอ่านและนำรวมกันเป็นข้อมูลแบบสตริง (STRING) การสแกนอ่านอักขระในแต่ละบล็อคจะได้ข้อมูลออกมา 3 ชุด

6.1.3 การเก็บข้อมูลของหน้าสัมผัสและเอาท์พุท ข้อมูลของหน้าสัมผัสหรือเอาท์พุทและข้อมูลตำแหน่งอินพุทเอาท์พุท จะถูกเก็บลงในอาเรย์ขนาดสองมิติ เช่นจากแลดเดอร์-ไดอะแกรมดังรูป 6.1 เมื่อทำการสแกนผ่านไปแล้วจะได้ข้อมูลที่ถูกเก็บในอาเรย์ดังนี้



0000	LD	0100
0001	OR	0101
0002	OR NOT	0102
0003	OR NOT	0103
0004	AND NOT	0104
0005	AND	0105
0006	OR	0200
0007	OUT	0200

รูปที่ 6.1 แสดงแลดเดอร์ไดอะแกรมที่ใช้ยกตัวอย่าง

	01	02	03	04	05	06	07	08	09
01	O	C	O	.	.	.	.	.	O
02	O								
03	C								
04	C								
05	.	O							
06									
07									
08									
09									
10									
11									

รูปที่ 6.2 แสดงอาเรย์ในการเก็บข้อมูลลักษณะของหน้าสัมผัสรีเลย์และเอาท์พุท

แสดงอาเรย์สองมิติขนาด 11 * 9 สำหรับเก็บลักษณะของหน้าสัมผัส เมื่อหน้าสัมผัสเป็นแบบปกติเปิดจะใช้อักขระ "O" เป็นแบบปกติปิดใช้อักขระ "C" เป็นข้อมูลใส่ลงในอาเรย์ หลังจากทำการสแกนอ่านข้อมูลจนได้ข้อมูลของหน้าสัมผัสแล้วต่อไปจะทำการเก็บลักษณะของจุดต่อเชื่อมของหน้าสัมผัสลงในอาเรย์ที่มีขนาด 11 * 9 ดังรูปที่ 6.3

	01	02	03	04	05	06	07	08	09
01	├	┬	─	┬	─	─	─	─	─
02	├	┤		│					
03	├	┤		│					
04	├	┘		│					
05	├	─	─	┘					
06									
07									
08									
09									
10									
11									

รูปที่ 6.3 แสดงอาเรย์ในการเก็บข้อมูลจุดเชื่อมต่อของหน้าสัมผัส

ส่วนอาเรย์ชุดที่สามเป็นอาเรย์ของข้อมูลแบบสตริงใช้เก็บข้อมูลที่บ่งบอกตำแหน่งของอินพุทเอาท์พุทมีขนาด 11 * 9 เช่นเดียวกัน

	01	02	03	04	05	06	07	08	09
01	0100	0104	0105						0200
02	0101								
03	0102								
04	0103								
05		0200							
06									
07									
08									
09									
10									
11									

รูปที่ 6.4 แสดงอาเรย์ในการเก็บข้อมูลของตำแหน่งอินพุท เอาท์พุท

6.2 การแปลความหมายหลังการตรวจสอบ

การแปลความหมายของแลดเดอร์ไดอะแกรมให้เป็นคำสั่งแบบบูลลีนทำได้โดยนำข้อมูลที่ได้จากอาเรย์ทั้ง 3 ดังที่กล่าวมาแล้วในหัวข้อ 6.1 มาทำการตรวจสอบโดยใช้วิธีการอ่านข้อมูล จากอาเรย์ข้อมูลจุดเชื่อมต่อหน้าสัมผัส จากนั้นนำมาแยกว่าเป็นจุดเชื่อมต่อแบบใด ซึ่งจุดเชื่อมต่อต่าง ๆ นั้นจะมีลักษณะเฉพาะแสดงให้เห็นถึงเงื่อนไขการกระทำกันทางลอจิกของหน้าสัมผัสที่ประกอบอยู่ระหว่างจุดเชื่อมต่อ การอ่านข้อมูลในอาเรย์ข้อมูลจุดเชื่อมต่อจะทำการอ่านแบบค้นหาโดยมีลักษณะดังนี้

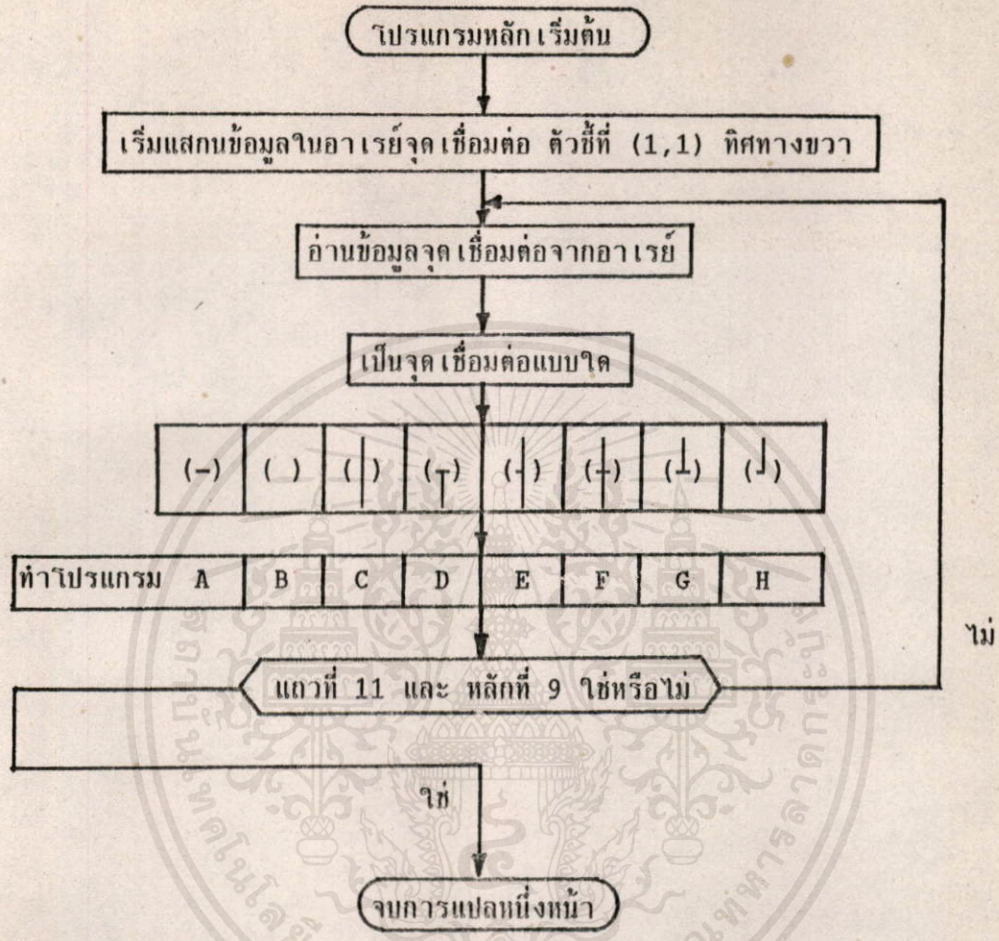
จากข้อมูล 1,1 ไป 1,2 เรียกการเคลื่อนที่ค้นหาทางขวา

จากข้อมูล 1,3 ไป 1,2 เรียกการเคลื่อนที่ค้นหาทางซ้าย

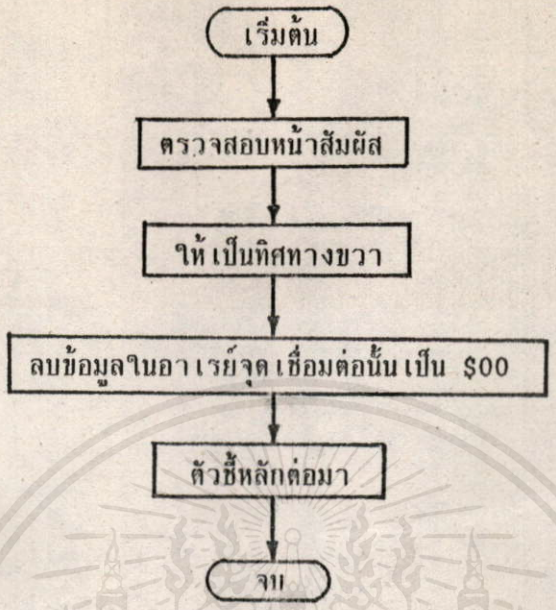
จากข้อมูล 1,2 ไป 2,2 เรียกการเคลื่อนที่ค้นหาทางลง

จากข้อมูล 3,2 ไป 2,2 เรียกการเคลื่อนที่ค้นหาทางขึ้น

หลังจากที่ทำการค้นหาข้อมูลในอาเรย์ดังกล่าว จะต้องทำการเปลี่ยนแปลงข้อมูลหรือทำการลบมูลในอาเรย์ เพื่อการค้นหาต่อไปจะไม่เกิดการสับสน สามารถอธิบายตามหลักการดังนี้

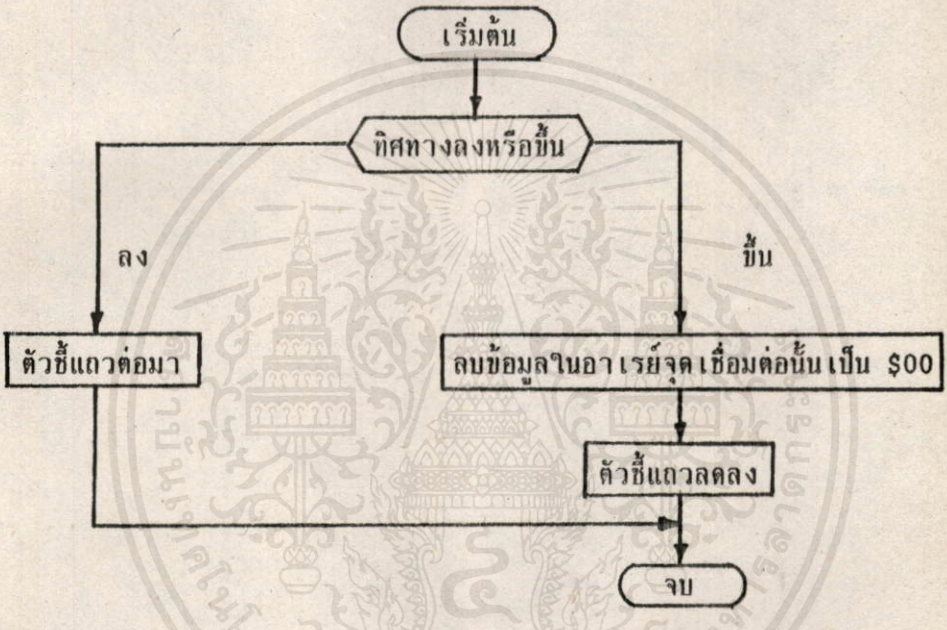


โปรแกรม A (-) หรือ $C4

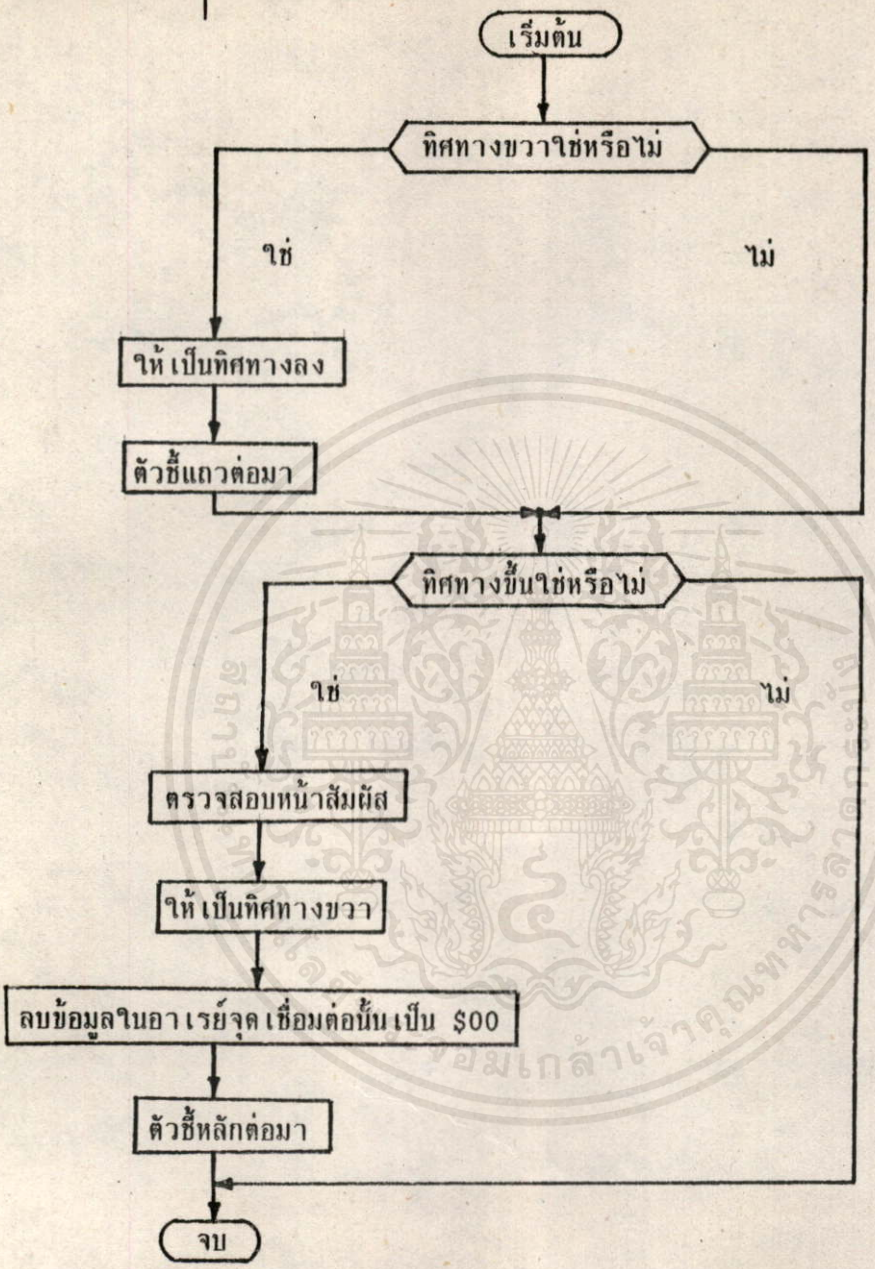


โปรแกรม B ( ) $20,$00

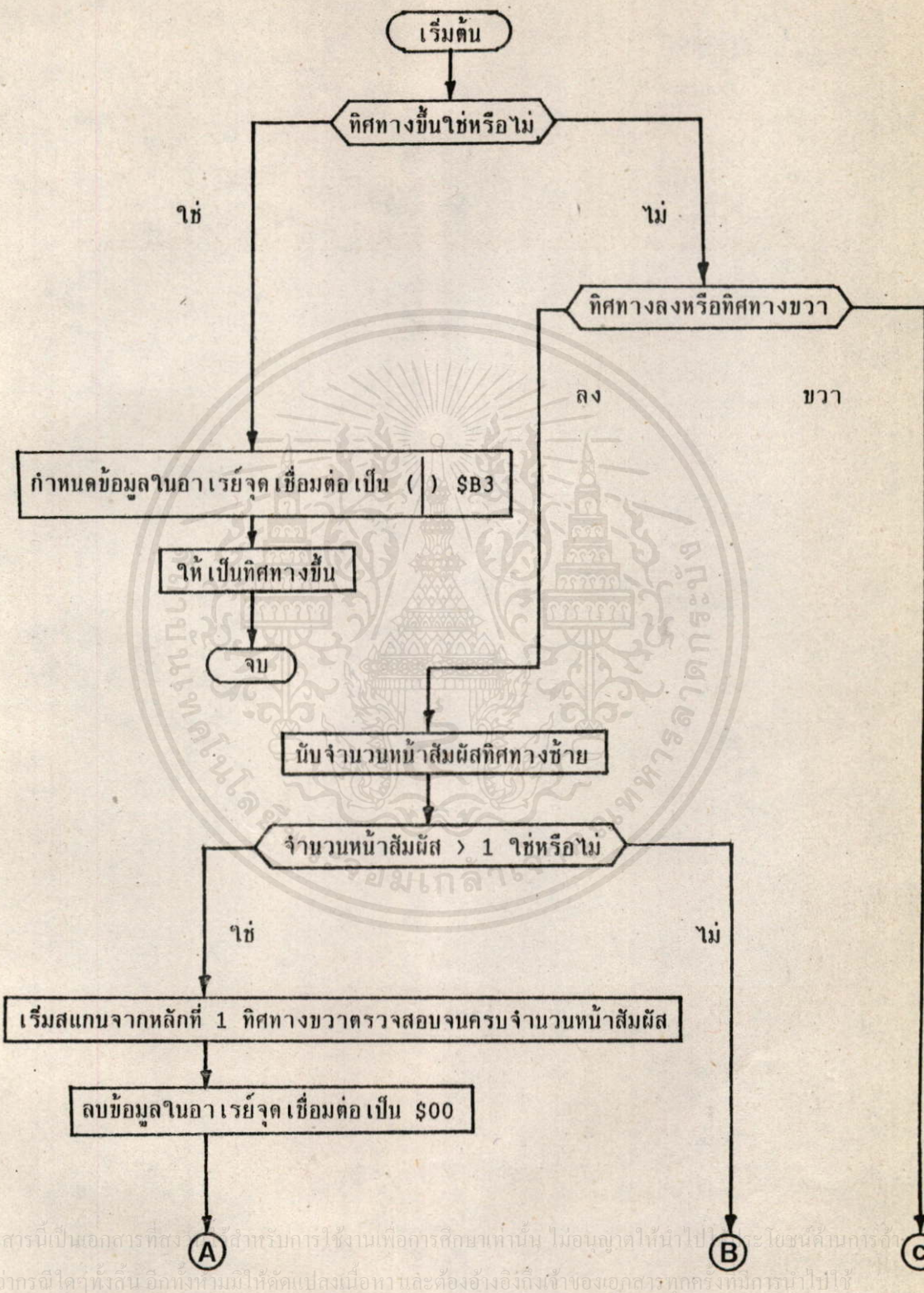
โปรแกรม C (|) $B3

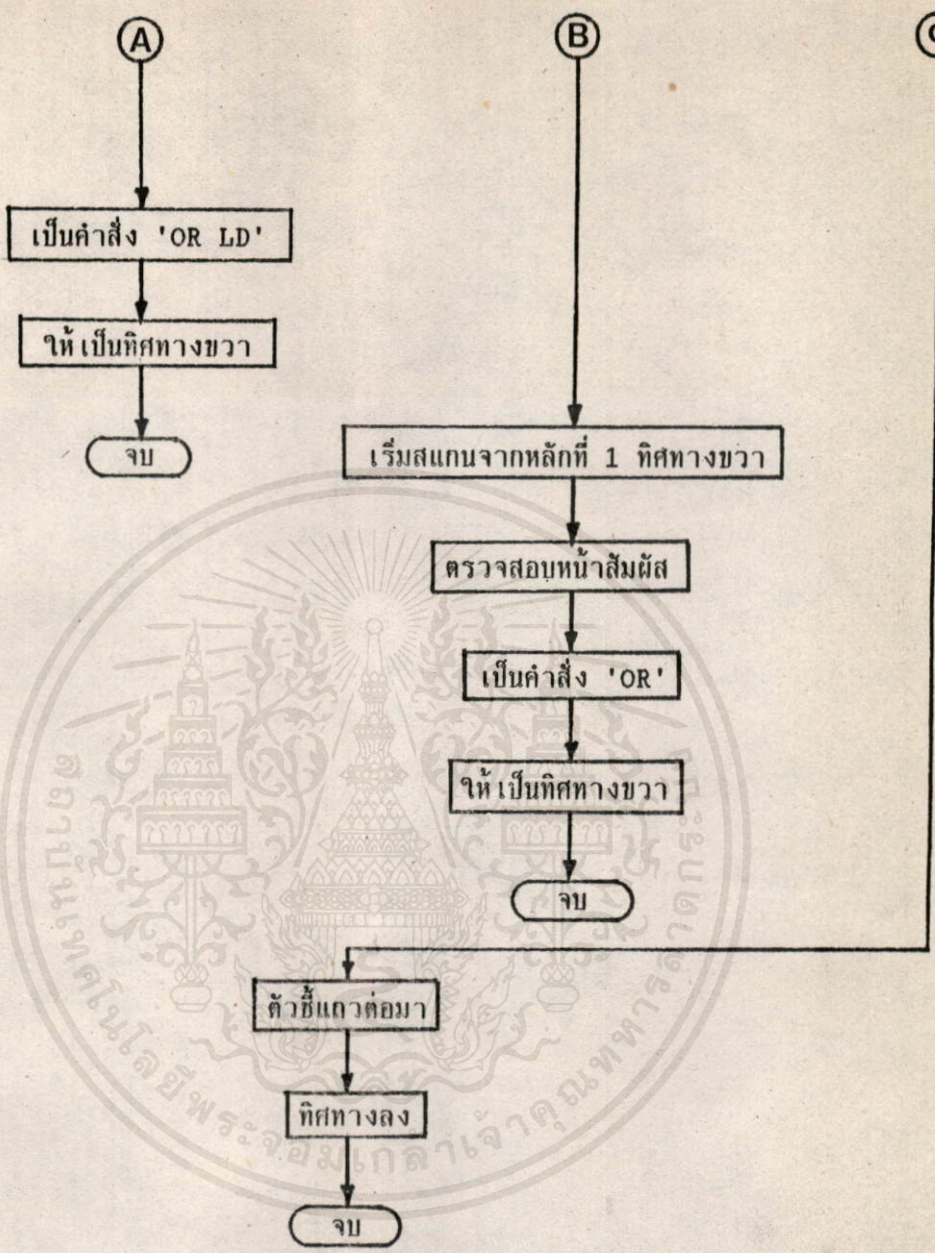


โปรแกรม D (┬) $C2

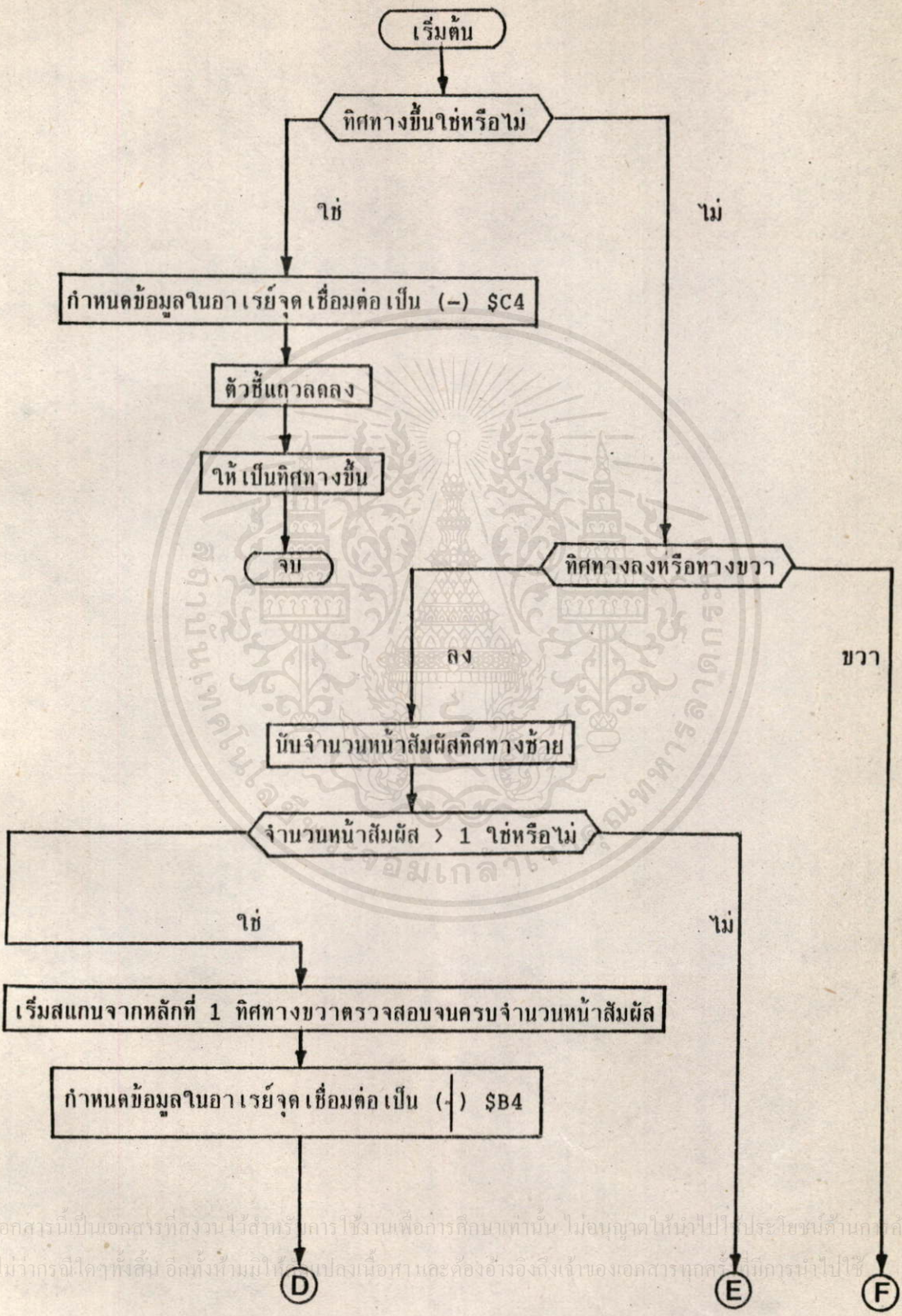


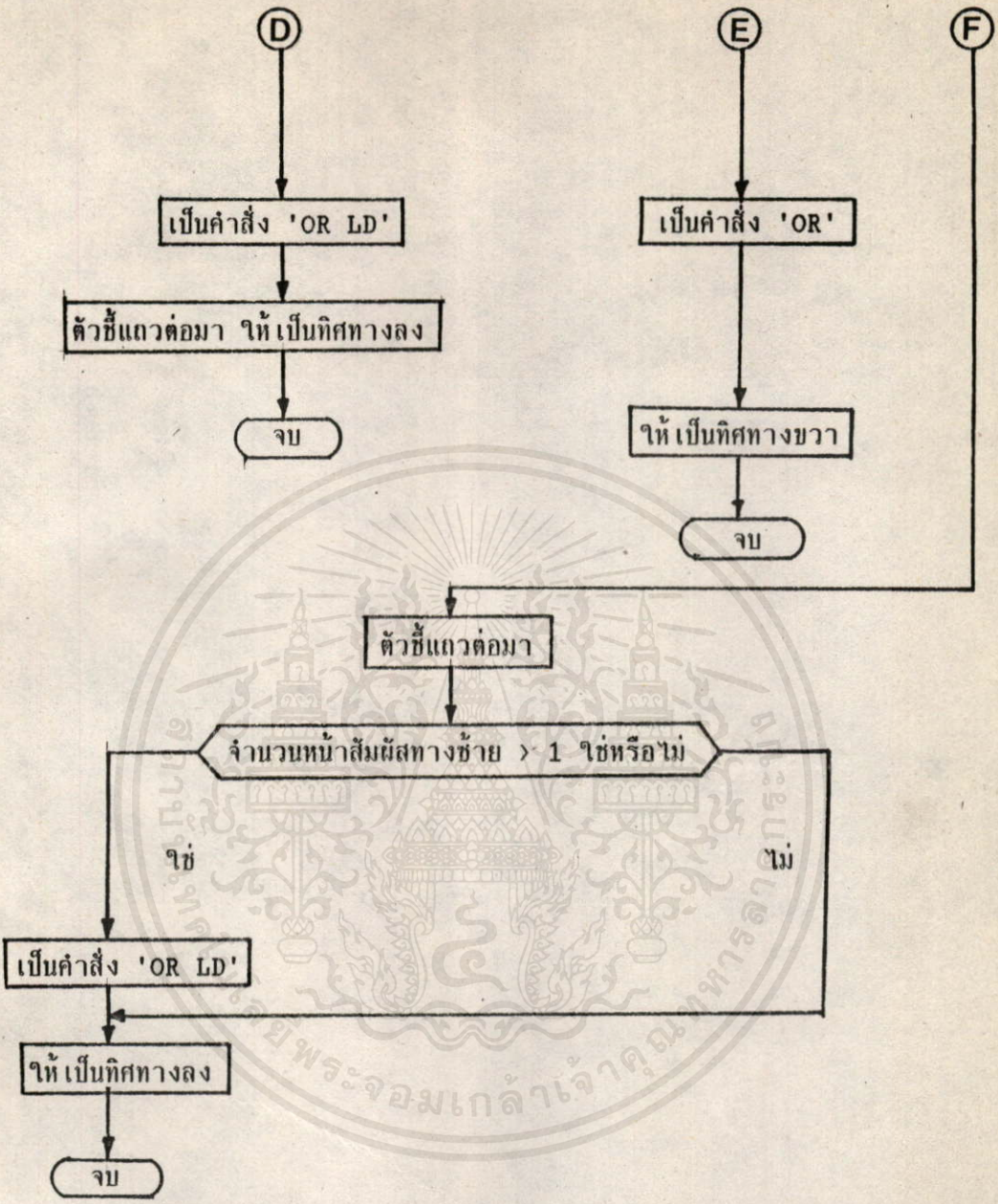
โปรแกรม E (┤) $B4



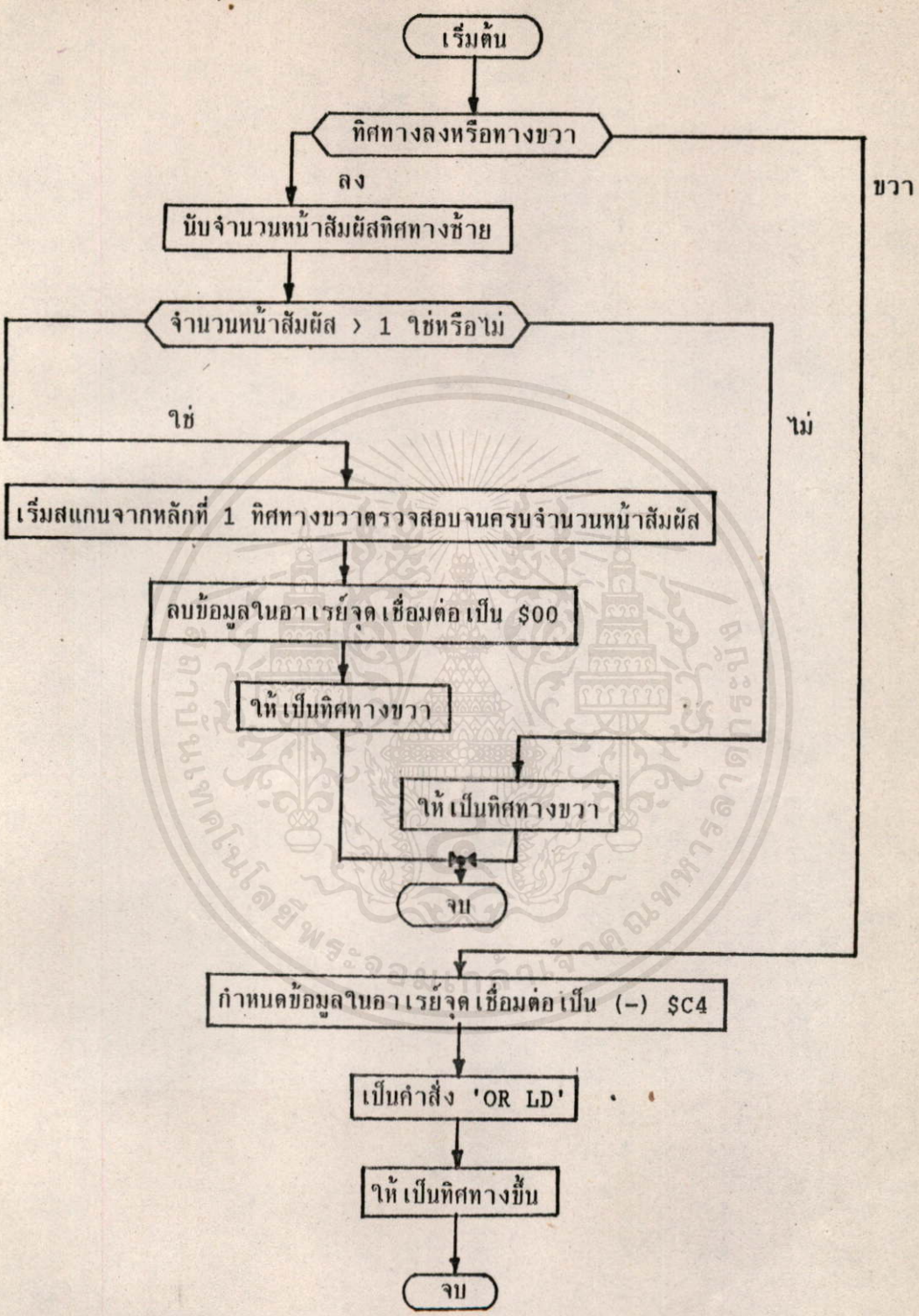


โปรแกรม F (+) $C5

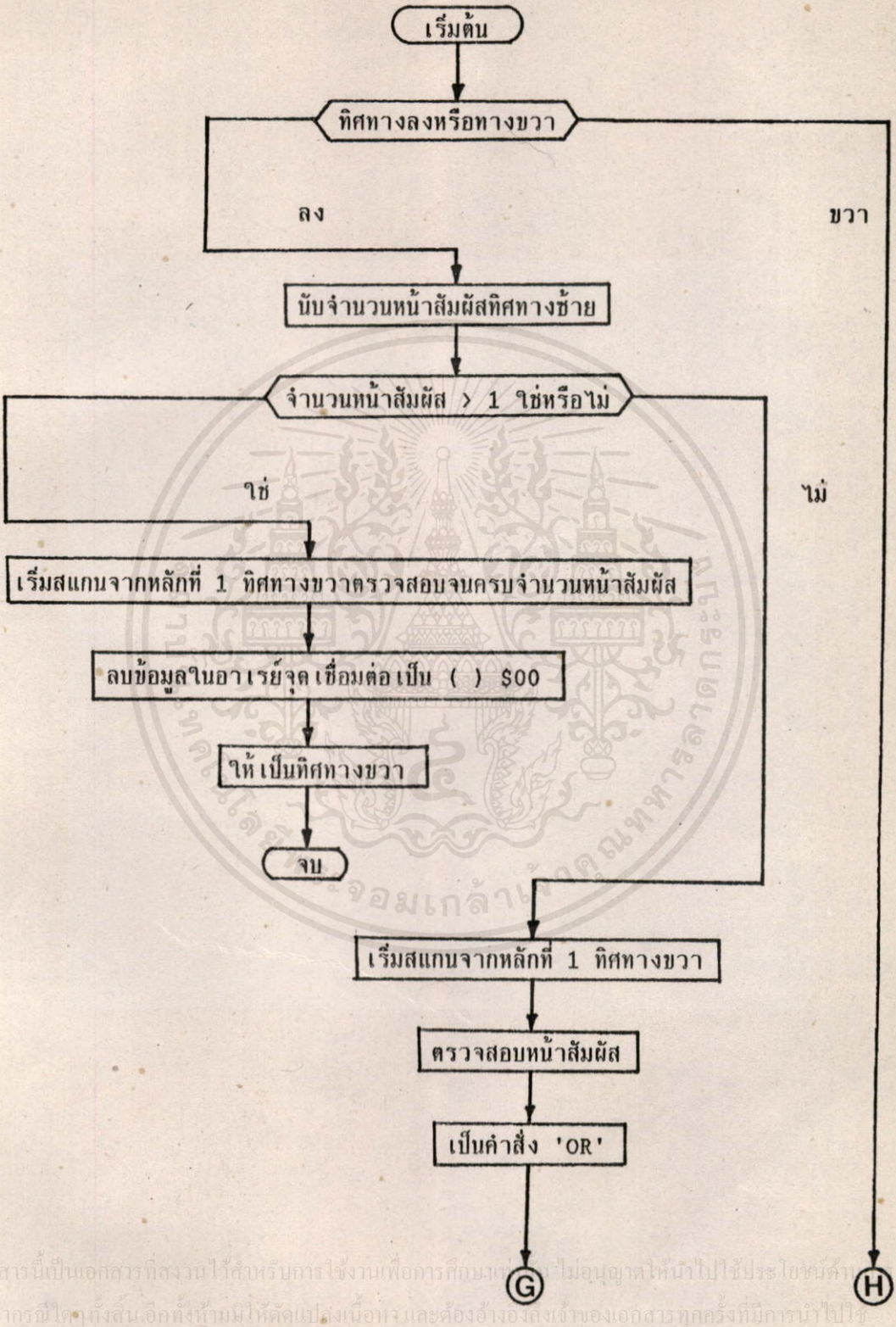


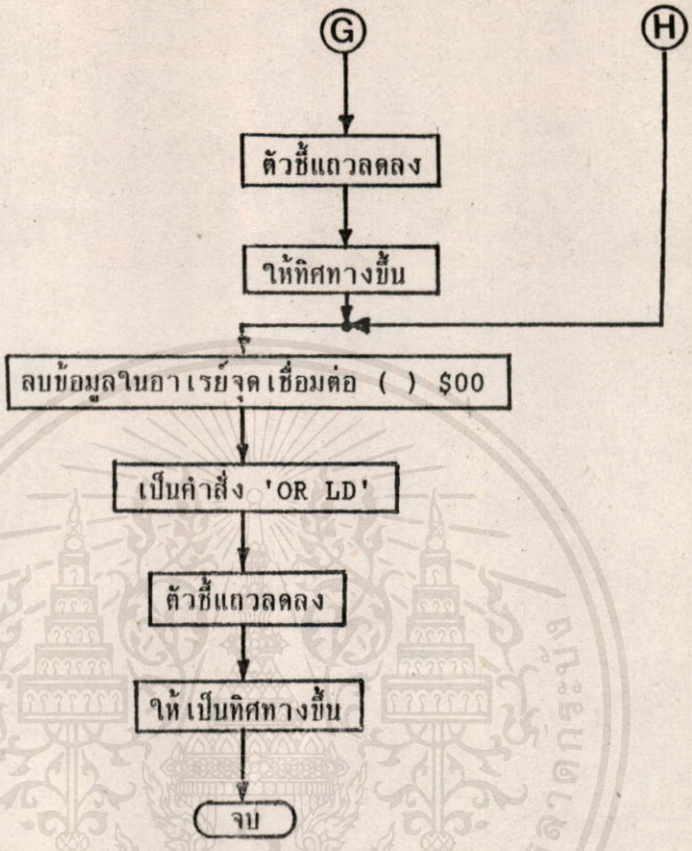


โปรแกรม G (⊥) $C1



โปรแกรม H (┘) $D9





บทที่ 7
การตรวจสอบการทำงานของเครื่องควบคุมและการติดต่อกับอุปกรณ์ร่วม

การตรวจสอบการทำงาน หมายถึงการรับส่งค่าสภาวะระหว่างเครื่องควบคุมกับเครื่องคอมพิวเตอร์ โดยเครื่องควบคุมจะนำข้อมูลที่ได้จากการประมวลผลในแต่ละรอบของการทำงาน (ซึ่งข้อมูลเหล่านั้นก็คือข้อมูลในตารางข้อมูล) เพื่อแสดงหรือตรวจสอบสภาวะการทำงาน เช่น อินพุท เอาท์พุท ค่าผลลัพธ์จากการคำนวน ค่าการนับ ค่าการตั้งเวลา เป็นต้น นำมาแสดงผลในขณะที่เครื่องควบคุมมีการทำงานตามขั้นตอนของผู้ใช้ โดยเครื่องควบคุมจะทำการส่งค่าสภาวะหรือข้อมูลภายในตารางข้อมูลออกสู่ภายนอกทุก ๆ รอบของการทำงาน

ในบทนี้จะกล่าวถึงโปรแกรม **PCLINK** ที่ใช้เป็นระบบการติดต่อสื่อสารข้อมูล การแสดงผลหรือติดตามค่าสภาวะ ต่อจากบทที่ 6 แบ่งอธิบายเป็นหัวข้อดังนี้

- โปรแกรมตรวจสอบและติดตามการทำงานของเครื่องควบคุมที่โปรแกรมได้
- โปรแกรมสนับสนุนอื่น ๆ

7.1 โปรแกรมตรวจสอบและติดตามการทำงานของเครื่องควบคุมที่โปรแกรมได้

การตรวจสอบการทำงาน จะกระทำได้โดยการใช้คำสั่งที่มีลักษณะเป็นที่ตกลงกันไว้ระหว่างเครื่องคอมพิวเตอร์กับเครื่องควบคุม หรือโปรโตคอล (PROTOCOL) ส่งให้เครื่องควบคุม ข้อมูลที่เครื่องคอมพิวเตอร์ส่งให้กับเครื่องควบคุมมีลักษณะที่เป็นเลขฐานสิบหกมีค่าเท่ากับตำแหน่งของข้อมูลในตารางข้อมูลที่ต้องการในขณะนั้น เมื่อเครื่องควบคุมได้รับข้อมูลดังกล่าวแล้วก็จะนำไปประมวลผลเพื่อหาตำแหน่งในตารางข้อมูล จากนั้นก็ทำการส่งข้อมูลที่เครื่องคอมพิวเตอร์ต้องการออกมา (การส่งผ่านข้อมูลทั้งหมดกระทำบนการสื่อสารข้อมูลแบบ RS-232C) และเครื่องคอมพิวเตอร์จะนำข้อมูลที่ได้นั้นไปทำการแยกออกเป็นบิทเพื่อแสดงผลเป็นเลขฐานสองหรือแสดงผลเป็นลักษณะของการเปิดปิดของหน้าสัมผัส ซึ่งจะได้อธิบายต่อไป ในหัวข้อการแสดงผลค่าสภาวะแบบตารางและแบบแลดเดอร์ไดอะแกรม

MONITORING TABLE 01 — 1:56:09

CH00	0110000100000001	[6101]	CH17	0000011000000110	[0606]
CH01	1101111010000110	[DD06]	CH18	0000000010010111	[0097]
CH02	1000000001011110	[805E]	CH19	0110011100100000	[6720]
CH03	1010000100100000	[A120]	CH20	0111010000000001	[7A01]
CH04	0110011000000001	[6601]	CH21	1000110100000110	[8D06]
CH05	0111001000000110	[7206]	CH22	0000000010010111	[0097]
CH06	1000000001011111	[805F]	CH23	0001000100100000	[1120]
CH07	0110100000100000	[6820]	CH24	0111111100000001	[7F01]
CH08	0110101110000001	[6B01]	CH25	1000110100000110	[8D06]
CH09	0101010100000110	[5506]	CH26	0000000010010111	[0057]
CH10	0000000011000111	[00C7]	CH27	1001001100100000	[9320]
CH11	0110100100000000	[6900]	CH28	1000010000000001	[8401]
CH12	0111000000000001	[7001]	CH29	0000011000000110	[0606]
CH13	1010010100000110	[A506]	CH30	0000000010010111	[0097]
CH14	0000000011000111	[00C7]	CH31	1001101100100000	[9B20]
CH15	1001001000000000	[9200]	CH32	1000100100000001	[8901]
CH16	0111010100000001	[7501]	CH33	0111001000000110	[7206]

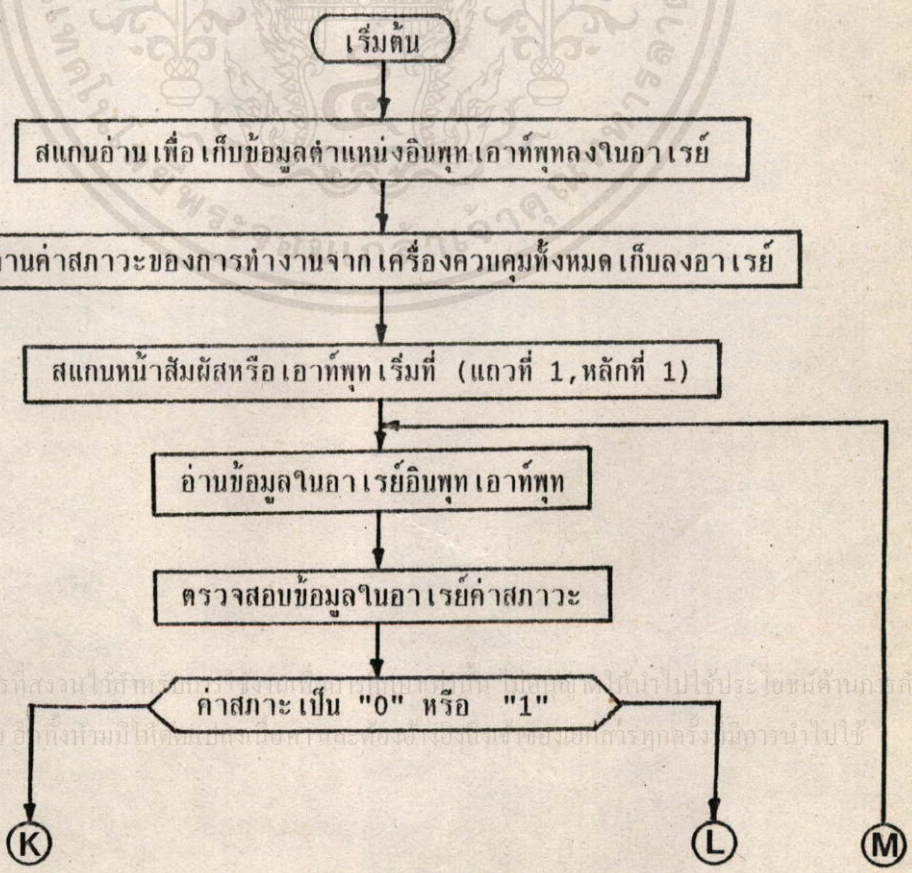
Programmable Controller and DataLogger System
Instrumentation Research KMITL June 1988
I/O Monitoring

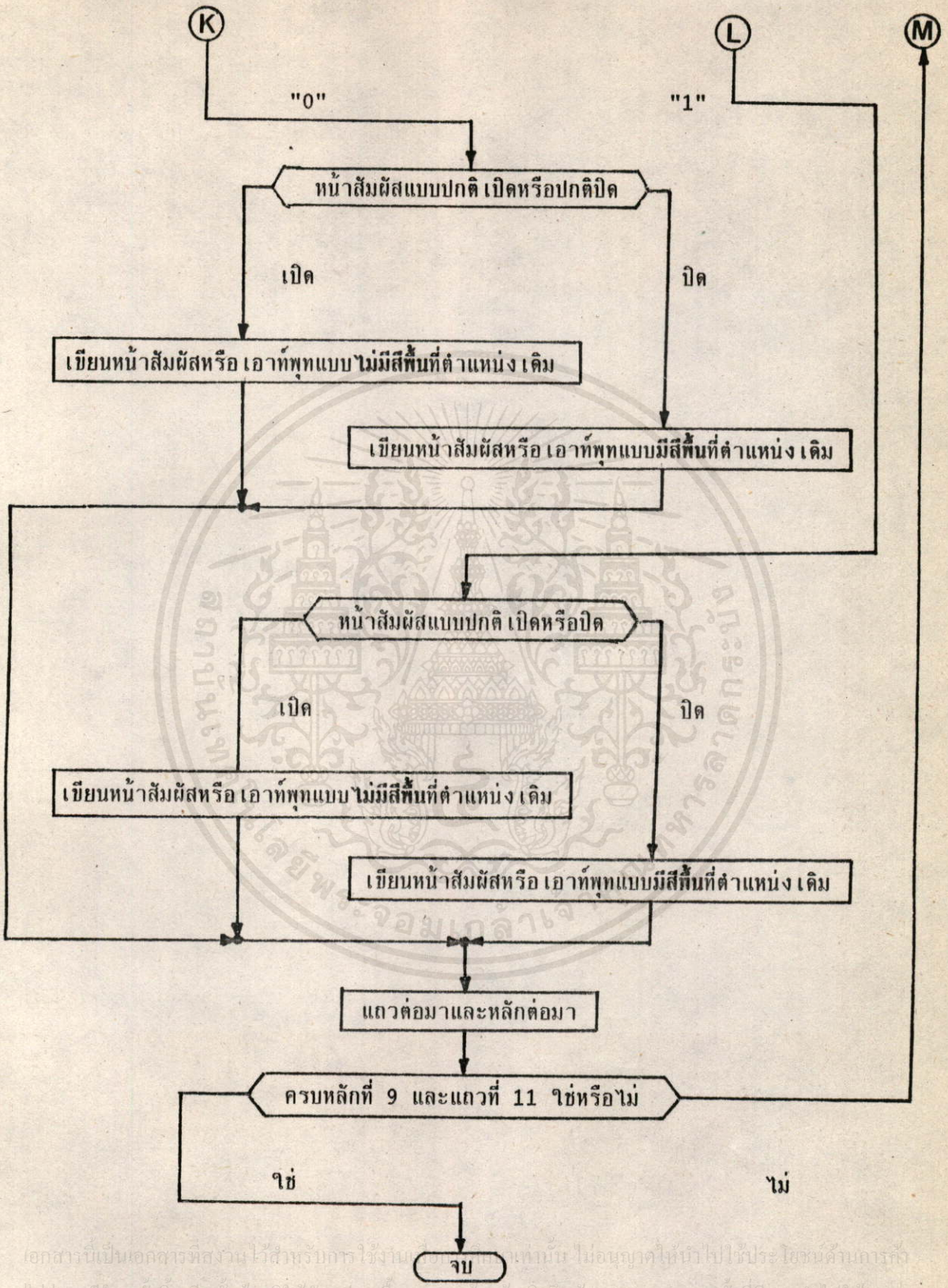
รูปที่ 7.1 แสดงตารางค่าสภาวะของเครื่องควบคุมที่เครื่องคอมพิวเตอร์รับค่าได้

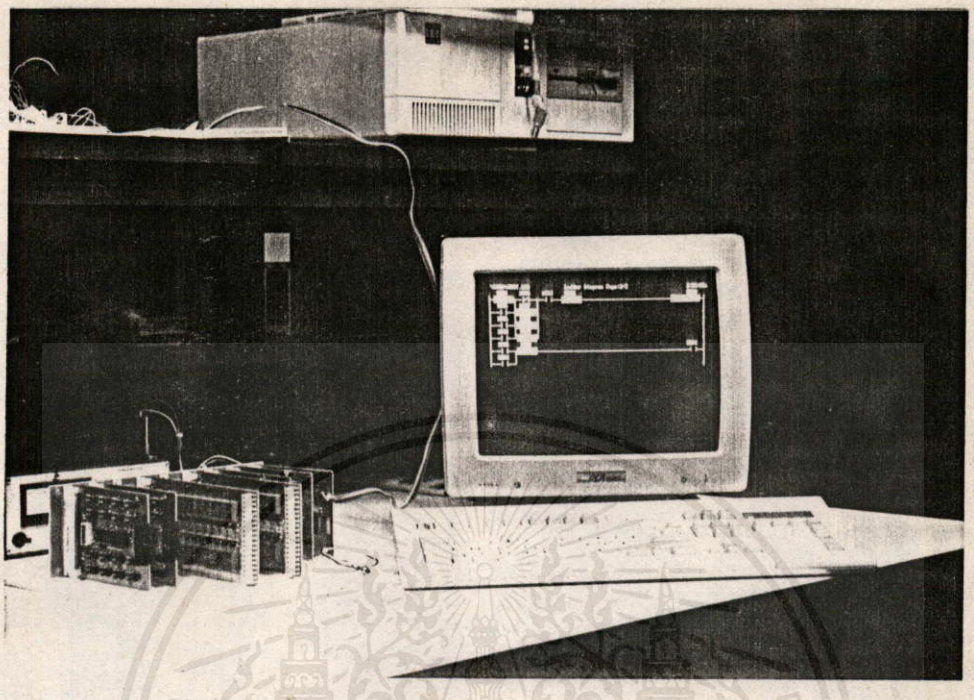
การแสดงผลค่าสภาวะแบบตาราง เป็นการแสดงผลเฉพาะข้อมูลเลขฐานสิบหกและเลขฐานสองของข้อมูลที่รับได้จากเครื่องคอมพิวเตอร์ แสดงการ "ON" หรือ "OFF" ของอินพุทเอาท์พุทด้วยบิทของข้อมูล "1" หรือ "0" ตามลำดับ การแสดงผลดังกล่าวนี้สามารสังเกตุการเปลี่ยนแปลงสภาวะได้เกือบทั้งหมด ซึ่งเป็นข้อดีในการที่จะนำไปประยุกต์ใช้งานเกี่ยวกับการแสดงผลของกระบวนการ (PROCESS MONITORING) การบันทึกการเปลี่ยนแปลงค่าสภาวะของกระบวนการในช่วงเวลาที่ต้องการได้ แต่ก็มีข้อเสียคือการตรวจสอบเงื่อนไขของกระบวนการทำได้ยาก เนื่องจากไม่สามารถเห็นภาพพจน์ของความสัมพันธ์กันระหว่างอินพุทกับเอาท์พุทได้

การแสดงค่าสภาวะแบบแลดเดอร์ไดอะแกรม เป็นการแสดงผลที่มีประสิทธิภาพมากวิธีหนึ่ง เนื่องจากสามารถมองเห็นภาพพจน์การเปลี่ยนแปลงค่าสภาวะตลอดจนเงื่อนไขระหว่างอินพุทเอาท์พุทที่มีความสัมพันธ์กันในกระบวนการที่กำลังทำการควบคุม ได้บนจอภาพแสดงผล ทำให้การตรวจสอบการทำงานเป็นไปได้โดยสะดวก

หลักการทำงานจะใช้วิธีการสแกนอ่านเพื่อเก็บค่าตำแหน่งอินพุทเอาท์พุทไว้ในอาเรย์ (หัวข้อการตรวจสอบข้อมูลแลดเดอร์ไดอะแกรมในบทที่ 6) เมื่อได้ตำแหน่งอินพุทเอาท์พุทจากการตรวจสอบข้อมูลแบบแลดเดอร์ไดอะแกรมแล้ว จึงทำการอ่านค่าสภาวะในตำแหน่งของอินพุทเอาท์พุทดังกล่าวจากเครื่องควบคุมและเก็บลงในอาเรย์ที่มีขนาด [0..33] ชนิดของข้อมูลแบบเวิร์ด และจัดการให้หน้าสัมผัสแสดงค่าสภาวะการเปิดปิดให้สอดคล้องกับค่าสภาวะนั้น ๆ เช่นกรณีหน้าสัมผัสหรือเอาท์พุทแบบปกติปิด เมื่อตรวจสอบค่าสภาวะแล้วปรากฏว่าเป็น "0" จะต้องทำการเขียนหน้าสัมผัสหรือเอาท์พุทดังกล่าว**ให้มีพื้นสี (BACKGROUND)** ที่ตำแหน่งนั้น และในทางตรงข้ามเมื่อตรวจสอบว่าสภาวะมีค่าเป็น "1" จะต้องทำการเขียนหน้าสัมผัสหรือเอาท์พุทโดย**ไม่มีพื้นสี**ที่ตำแหน่งนั้น ซึ่งจะต้องมีการปรับให้สอดคล้องกับการเปลี่ยนแปลงของค่าสภาวะอยู่ตลอดเวลา โดยมีวิธีการดังผังการทำงานต่อไปนี้





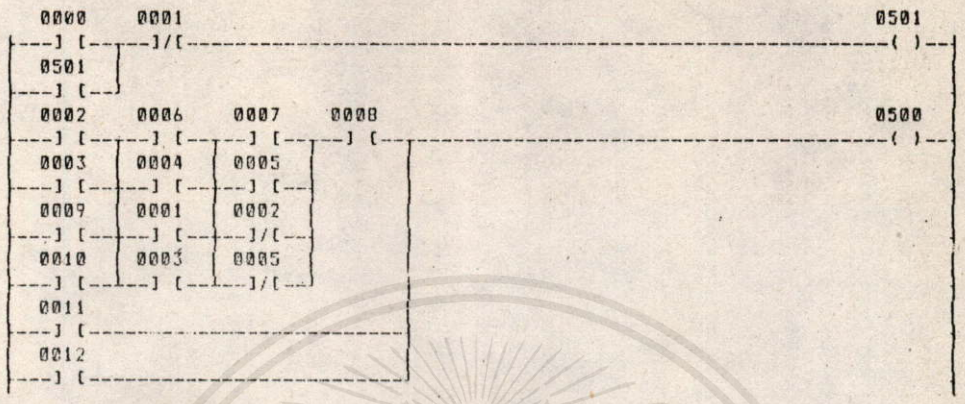


รูปที่ 7.3 การแสดงค่าสภาวะแบบแลดเดอร์ไดอะแกรมบนจอภาพแสดงผล

7.2 โปรแกรมสนับสนุนอื่น ๆ

การบันทึกโปรแกรมแบบแลดเดอร์ โปรแกรมที่ได้สร้างไว้ นอกจากนำเก็บลงแผ่นจานแม่เหล็กแล้วยังสามารถที่จะนำออกแสดงผลทางเครื่องพิมพ์ได้ จากที่กล่าวมาแล้วในหัวข้อ 5.2 การสร้างและแก้ไขแลดเดอร์ไดอะแกรม ซึ่งข้อมูลของแลดเดอร์ไดอะแกรมจะประกอบขึ้นจากอักขระหรือกลุ่มของอักขระที่มีรูปลักษณะคล้ายกับแลดเดอร์ไดอะแกรม การพิมพ์ข้อมูลแลดเดอร์ก็คล้ายกับการพิมพ์อักขระโดยทั่วไป ก่อนทำการพิมพ์โปรแกรมจะทำการแปลความหมายของแลดเดอร์ไดอะแกรมทีหนึ่งก่อน เพื่อให้ได้โปรแกรมคำสั่งแบบบูลลีน หลังจากนั้นก็ทำการส่งข้อมูลทั้งสองที่ได้ออกทางเครื่องพิมพ์

```
Programmable Controller and DataLogger System
Instrumentation Research KMITL        June 1988
Ladder Diagram Listing
```

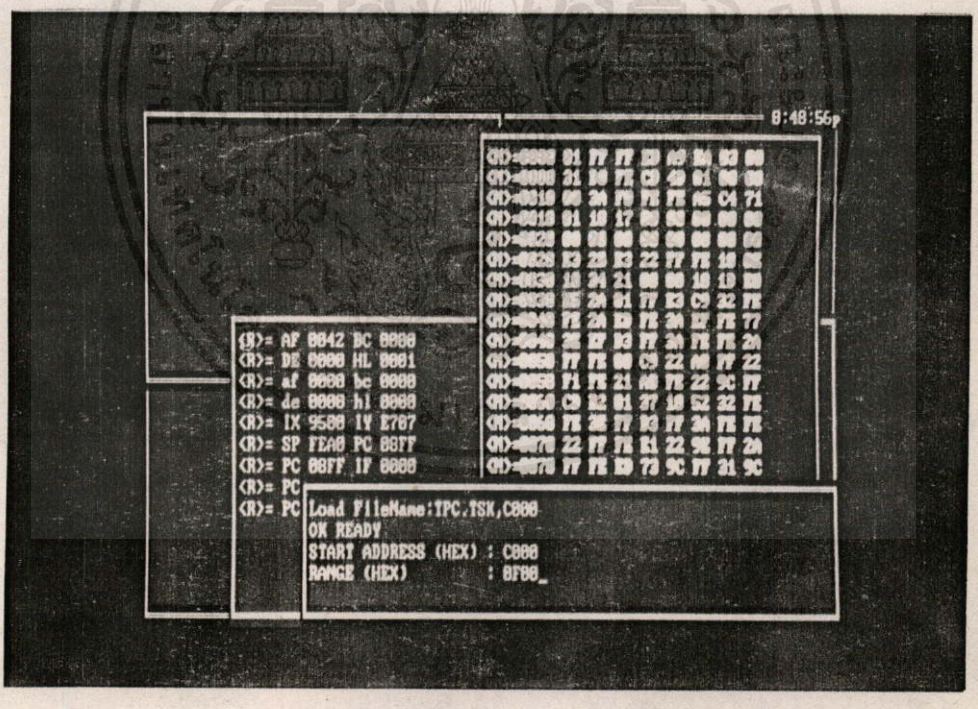


Instruction Listing

0000	LD	0000
0001	OR	0501
0002	AND NOT	0001
0003	OUT	0501
0004	LD	0002
0005	OR	0003
0006	OR	0009
0007	OR	0010
0008	LD	0006
0009	LD	0004
0010	OR LD	
0011	LD	0001
0012	OR LD	
0013	LD	0003
0014	OR LD	
0015	LD	0007
0016	LD	0005
0017	OR LD	
0018	LD NOT	0002
0019	OR LD	
0020	LD NOT	0005
0021	OR LD	
0022	AND LD	
0023	AND LD	
0024	AND	0008
0025	OR	0011
0026	OR	0012
0027	OUT	0500

รูปที่ 7.4 แสดงรายงานโปรแกรมแบบแลดเดอร์ไดอะแกรมจากเครื่องพิมพ์

การพัฒนาทางด้านฮาร์ดแวร์ ที่เกี่ยวกับระบบไมโครโปรเซสเซอร์ของเครื่องควบคุมที่โปรแกรมได้นั้นจำเป็นจะต้องมีระบบที่มีความสมบูรณ์มากกว่า มาช่วยให้การทำงานมีความสะดวกรวดเร็วและสามารถทำการตรวจสอบระบบได้ในระหว่างการพัฒนา ระบบเพื่อการตรวจสอบการทำงานหรือหาข้อบกพร่องที่เกิดขึ้น เรียกระบบดังกล่าวนี้ว่า **" ระบบช่วยพัฒนาไมโครโปรเซสเซอร์ "** ซึ่งเป็นโปรแกรมหนึ่งที่ผู้เขียนวิทยานิพนธ์ได้ทำการวิจัยสำหรับทำการพัฒนาระบบไมโครโปรเซสเซอร์ Z80 และได้นำมาใช้ในการพัฒนาเครื่องควบคุมที่โปรแกรมได้ (เอกสารรายงานวิจัย **" ระบบช่วยพัฒนาไมโครโปรเซสเซอร์ "** การประชุมสัมนาวิชาการทางวิศวกรรมไฟฟ้า 8 สถาบัน ครั้งที่ 10 จุฬาลงกรณ์มหาวิทยาลัย) ซึ่งการทำงานของระบบช่วยพัฒนาไมโครโปรเซสเซอร์นี้ สามารถที่จะตรวจสอบการทำงานของโปรแกรมที่กำลังพัฒนา โดยตรวจสอบค่าการเปลี่ยนแปลงของรีจีสเตอร์หรือค่าข้อมูลในหน่วยความจำได้ภายหลังการทำงาน นำโปรแกรมที่ทำการพัฒนาส่งผ่านไปในหน่วยความจำในตำแหน่งใด ๆ ที่กำหนด (DOWNLOAD)



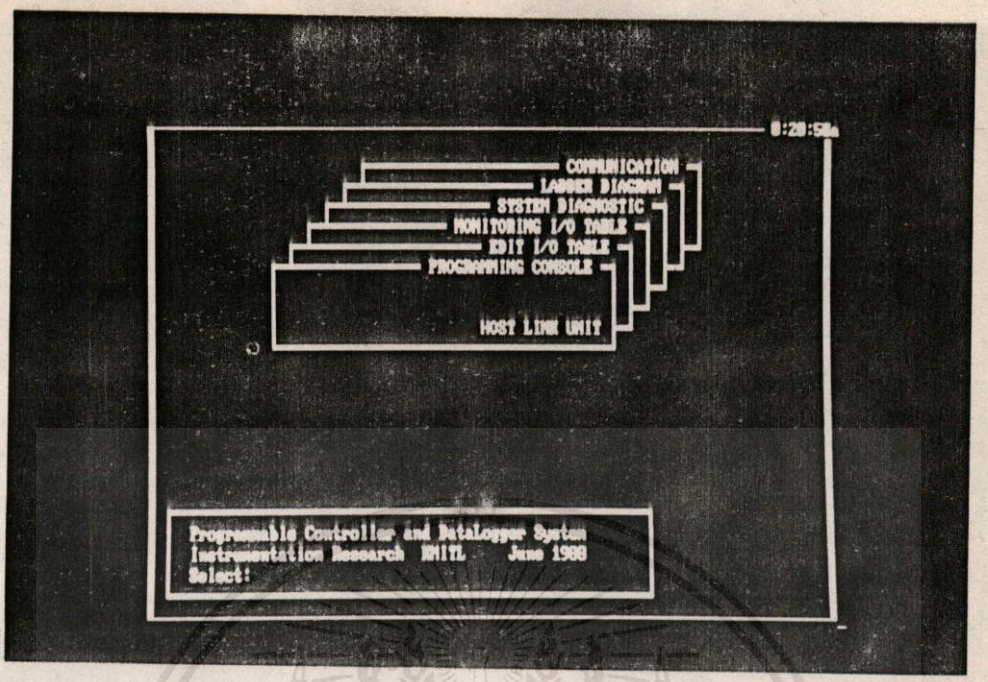
รูปที่ 7.4 แสดงค่าของข้อมูลในหน่วยความจำ ค่ารีจีสเตอร์
และการส่งผ่านข้อมูลไปในหน่วยความจำ

บทที่ 8
ตัวอย่างและผลการทดลอง

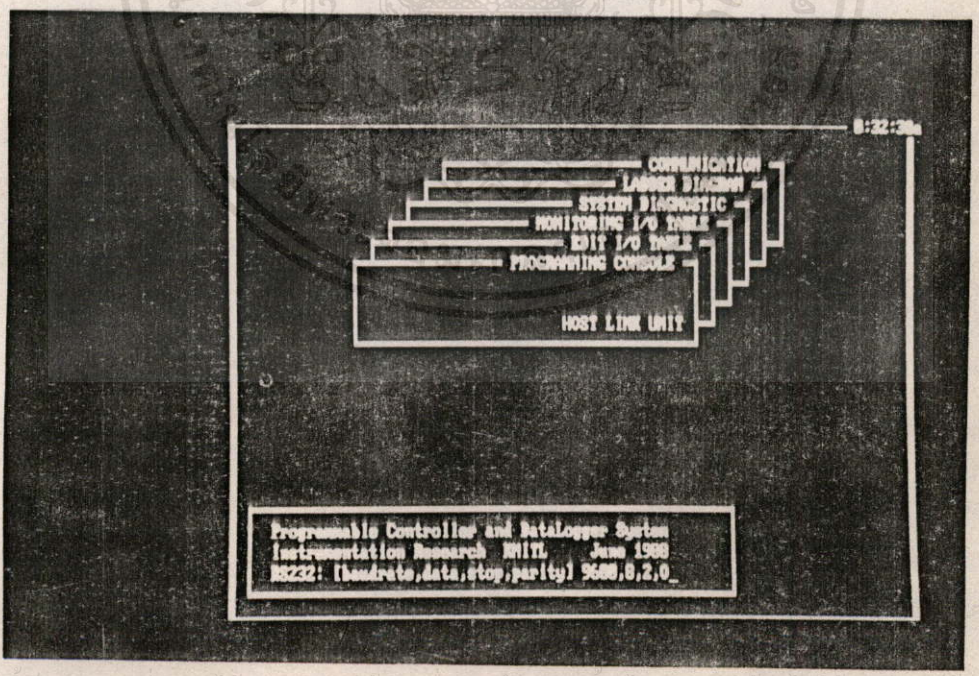
จากการทดลองในส่วนเครื่องควบคุมพบว่าฟังก์ชั่นการควบคุมทางลอจิกไม่มีปัญหาแต่อย่างไร สามารถที่จะใช้ควบคุมได้ ในกรณีที่ต้องส่งค่าการแสดงผลของสภาวะจะทำให้ความเร็วหรือเวลาในการสแกนหนึ่งรอบนั้นมีเวลามากขึ้น แต่อย่างไรก็ตามเวลาที่ใช้ในการประมวลผลเพื่อการควบคุมรวมกับเวลาในการส่งผ่านค่าสภาวะแล้วจะน้อยกว่าการเปลี่ยนแปลงสภาวะของอุปกรณ์ไฟฟ้าเชิงกล คือจะใช้เวลาในการประมวลผลประมาณ 130 คำสั่งต่อ 6 มิลลิวินาที

อีกอย่างหนึ่งคือไมโครโปรเซสเซอร์ ที่ใช้ควบคุมการทำงานของเครื่องควบคุมประมวลผลได้ครั้งละ 8 บิทแต่ฟังก์ชั่นการควบคุมต่าง ๆ ที่ได้สร้างขึ้นนั้นมีการประมวลผลแบบ 16 บิท ทำให้การพัฒนาโปรแกรมควบคุมการทำงานเป็นไปได้ยาก และทำให้ระยะเวลาการสแกนในแต่ละรอบสูงขึ้นมากในบางฟังก์ชั่น ปัญหานี้สามารถแก้ได้ด้วยการใช้ไมโครโปรเซสเซอร์ที่มีการประมวลผลแบบ 16 บิท ซึ่งจะทำในการพัฒนาโปรแกรมควบคุมเป็นไปได้ง่าย เนื่องจากมีคำสั่งในการทำงานที่ใกล้เคียงกันมาก อย่างไรก็ตามวิทยานิพนธ์ฉบับนี้เป็นเพียงแนวทางในการพัฒนาระบบควบคุมที่มีการโปรแกรมได้เท่านั้น ซึ่งในปัจจุบันยังเป็นเทคโนโลยีที่ปิดบังของผู้ผลิตอยู่ การออกแบบระบบไมโครโปรเซสเซอร์ที่ใช้ในการควบคุมทางอุตสาหกรรมมีความละเอียดอ่อน และใช้เทคโนโลยีที่สูงมากในการผลิตเส้นลายวงจรไฟฟ้าตลอดจนอุปกรณ์ทางอิเลคทรอนิคส์ ทั้งนี้เนื่องจากจะไม่ยอมให้ระบบมีการทำงานที่ผิดพลาดได้และต้องมีความทนทาน ส่วนการโปรแกรมแบบแลดเดอร์ไดอะแกรมเป็นการนำเอาเทคโนโลยีทางด้านซอฟท์แวร์ เข้ามาใช้ร่วมหรือเป็นส่วนเสริมเพื่อให้การควบคุมมีความสะดวกมากยิ่งขึ้น สำหรับการทดลองการโปรแกรมแบบแลดเดอร์ไดอะแกรมมีข้อจำกัดอยู่เล็กน้อย กล่าวคือการสร้างโปรแกรมจะต้องกระทำทีละหน้าโดยไม่จำกัดรังค์แต่จะต้องเสร็จสิ้นภายในหน้านั้น ๆ ไม่สามารถที่จะเลื่อนภาพขึ้นลง (SCROLL) ของแลดเดอร์ไดอะแกรมบนจอภาพแสดงผลได้

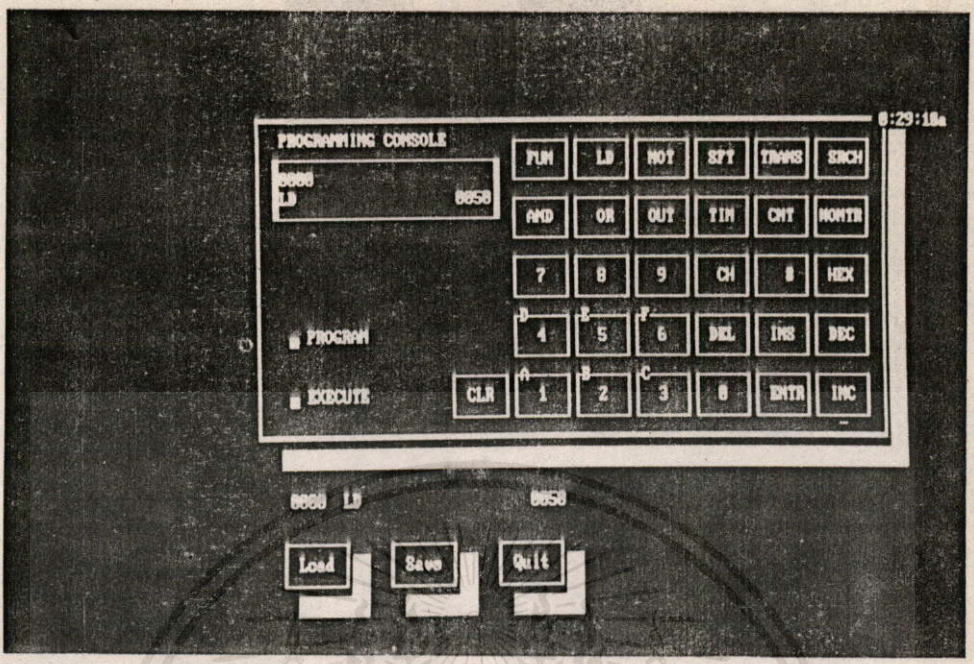
จากภาพต่อไปนี้ แสดงการจำลองแบบการทำงานของเครื่องควบคุมกับเครื่องคอมพิวเตอร์ที่ใช้ในการทดลอง การสร้างแก้ไขข้อมูลทั้งแบบบูลลีนและแลดเดอร์ไดอะแกรม การแสดงผลค่าสภาวะแบบแสดงในตารางและแลดเดอร์ไดอะแกรม พร้อมทั้งตัวอย่างการบันทึกข้อมูลแบบแลดเดอร์ไดอะแกรมและข้อมูลแบบบูลลีน



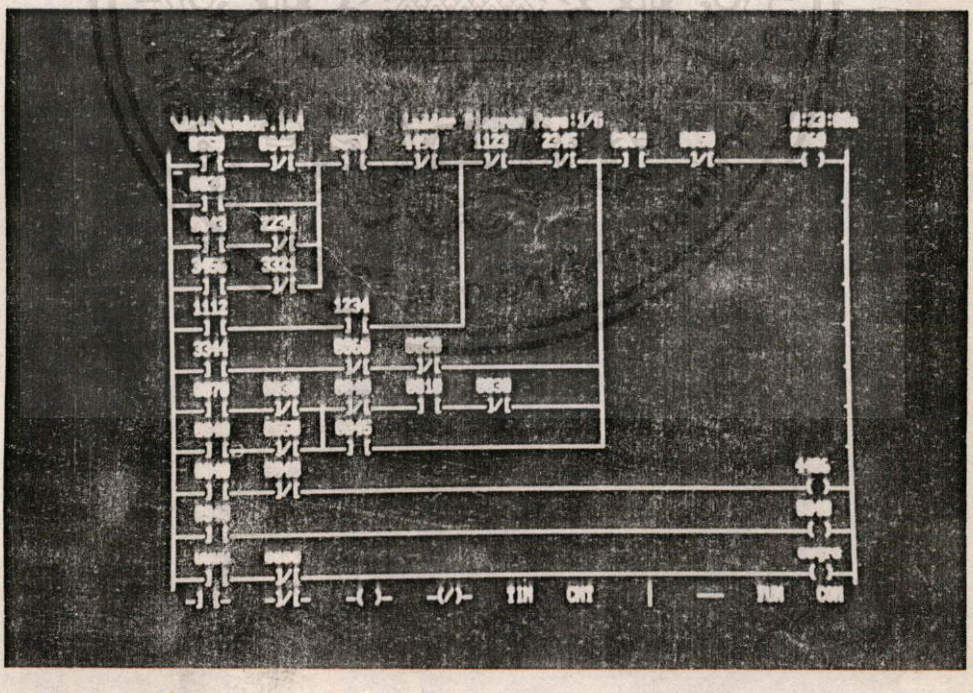
รูปที่ 8.1 แสดงเมนูหลักเพื่อเลือกโหมดการทำงาน



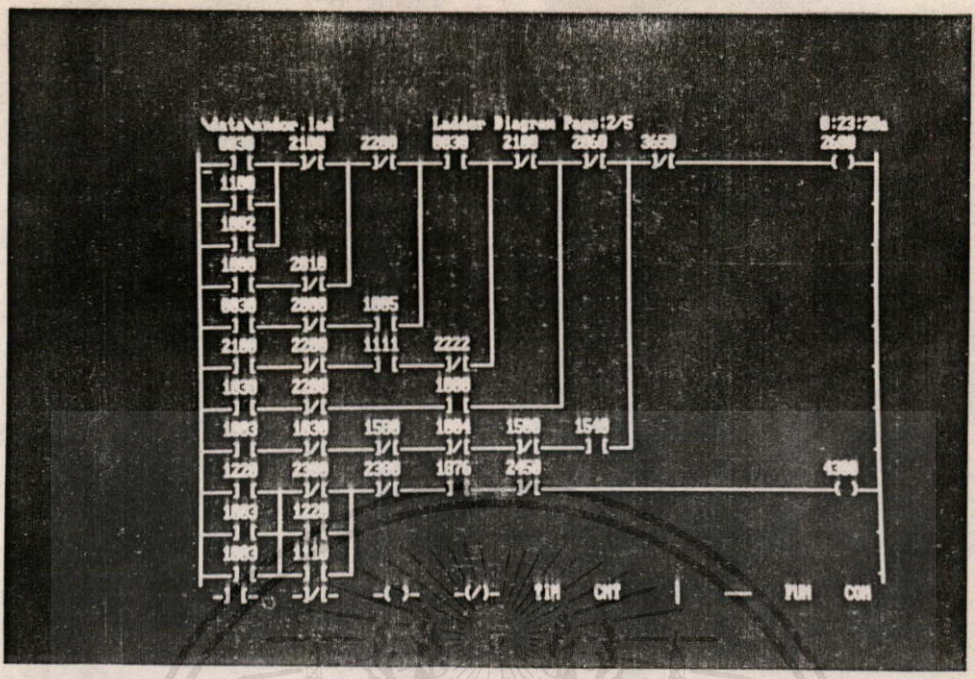
รูปที่ 8.2 แสดงการโปรแกรมให้กับอุปกรณ์การสื่อสารข้อมูล



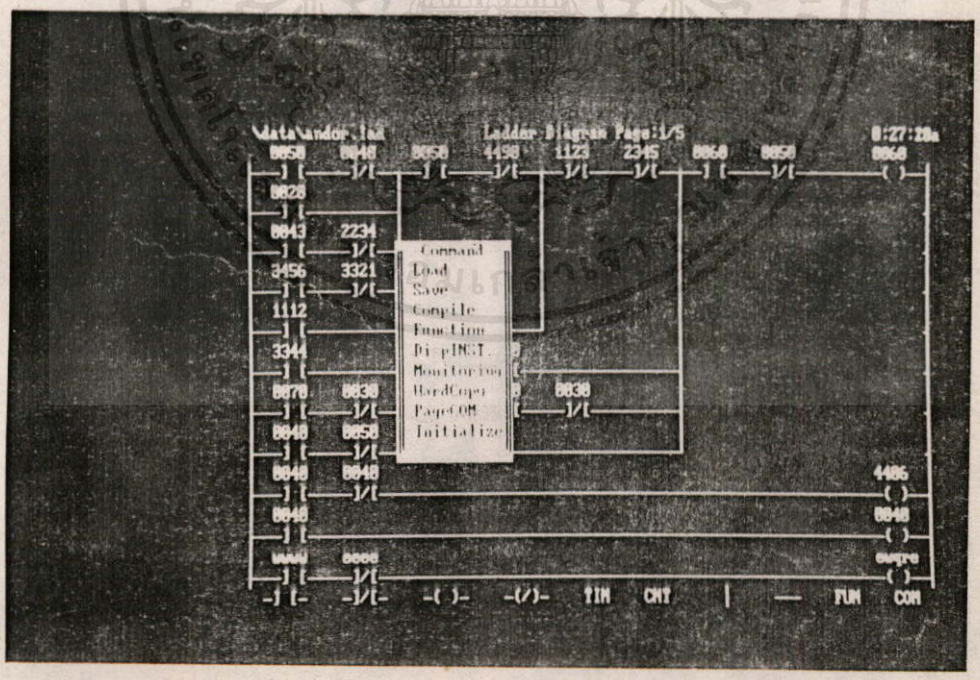
รูปที่ 8.3 แสดงหน่วยป้อนโปรแกรมที่ใช้สำหรับการโปรแกรมแบบบูลลีน



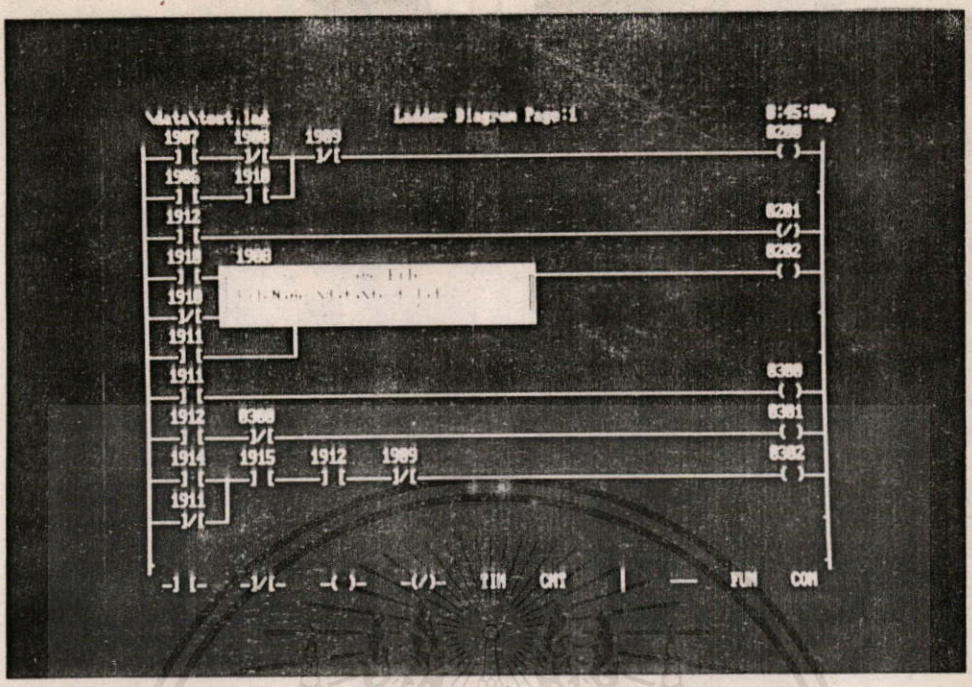
รูปที่ 8.4 แสดงแลดเดอร์ไดอะแกรมที่ได้สร้างขึ้น



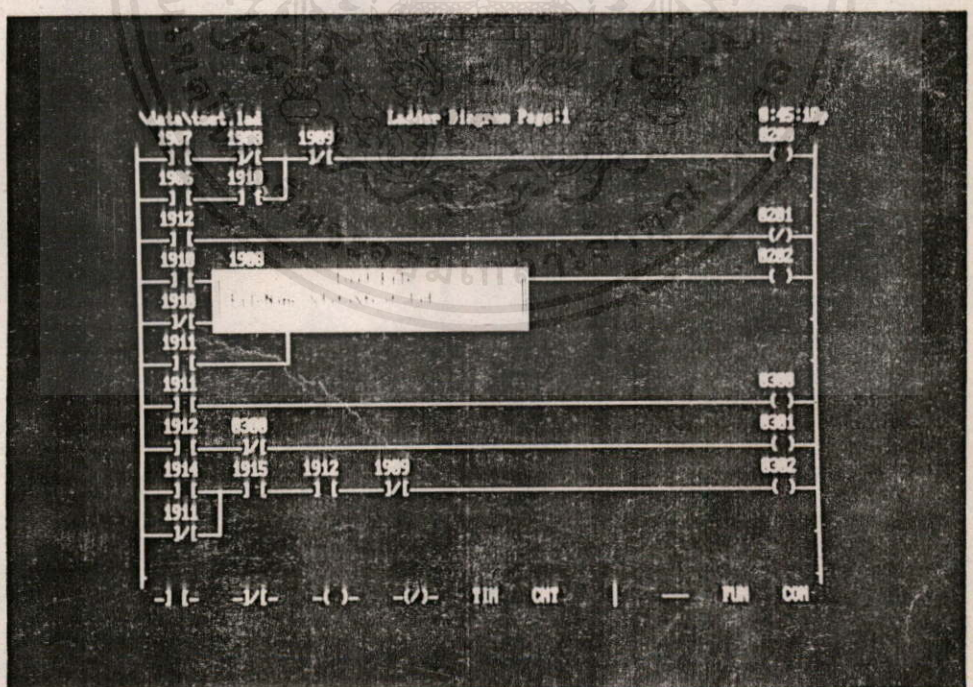
รูปที่ 8.5 แสดงแลดเดอร์ไดอะแกรมที่ได้สร้างขึ้น



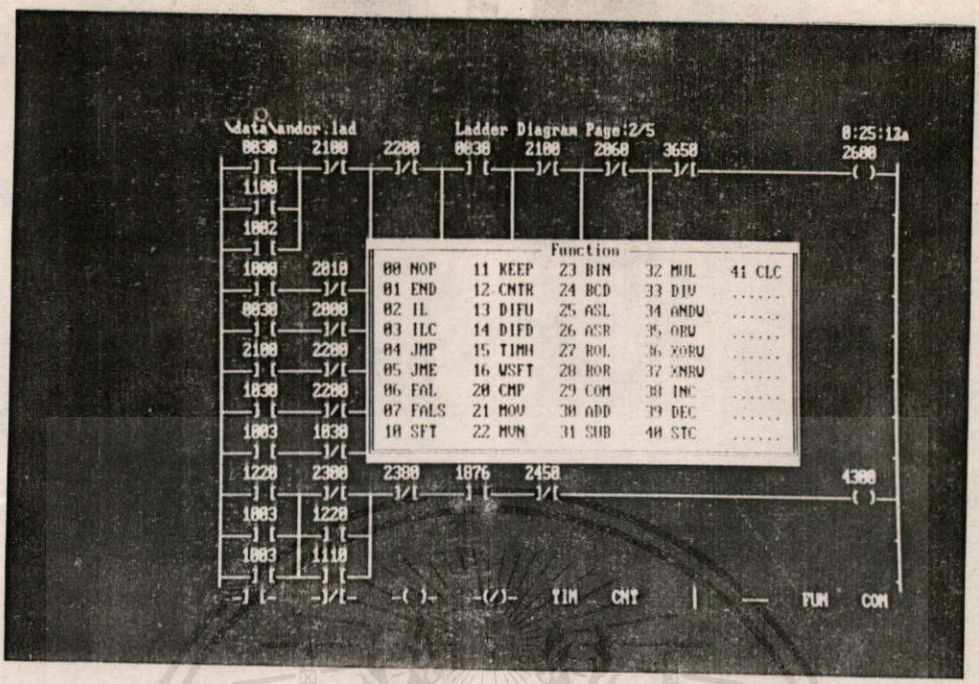
รูปที่ 8.6 แสดงคำสั่งการทำงานในโหมดต่าง ๆ



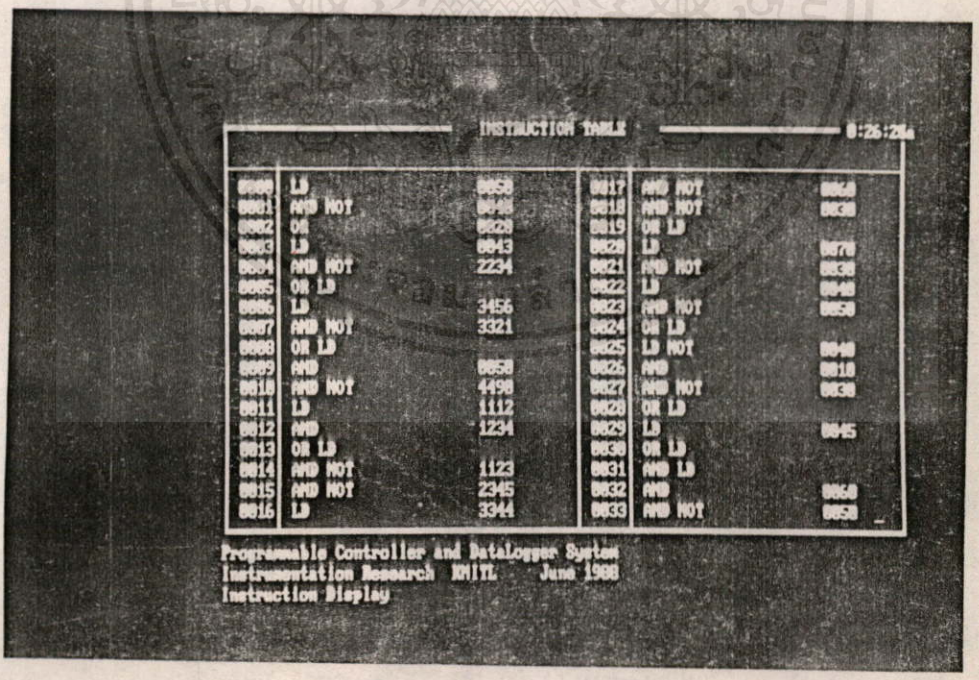
รูปที่ 8.7 แสดงการเก็บข้อมูลแลดเดอร์ไดอะแกรมลงแผ่นจานแม่เหล็ก



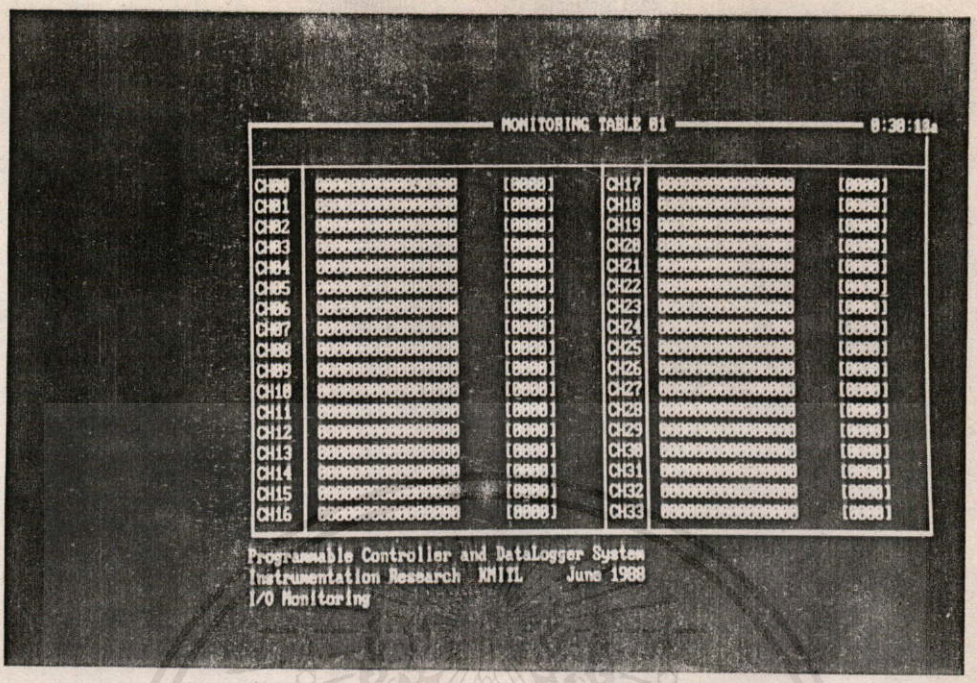
รูปที่ 8.8 แสดงการนำข้อมูลแลดเดอร์ไดอะแกรมจากแผ่นจานแม่เหล็ก



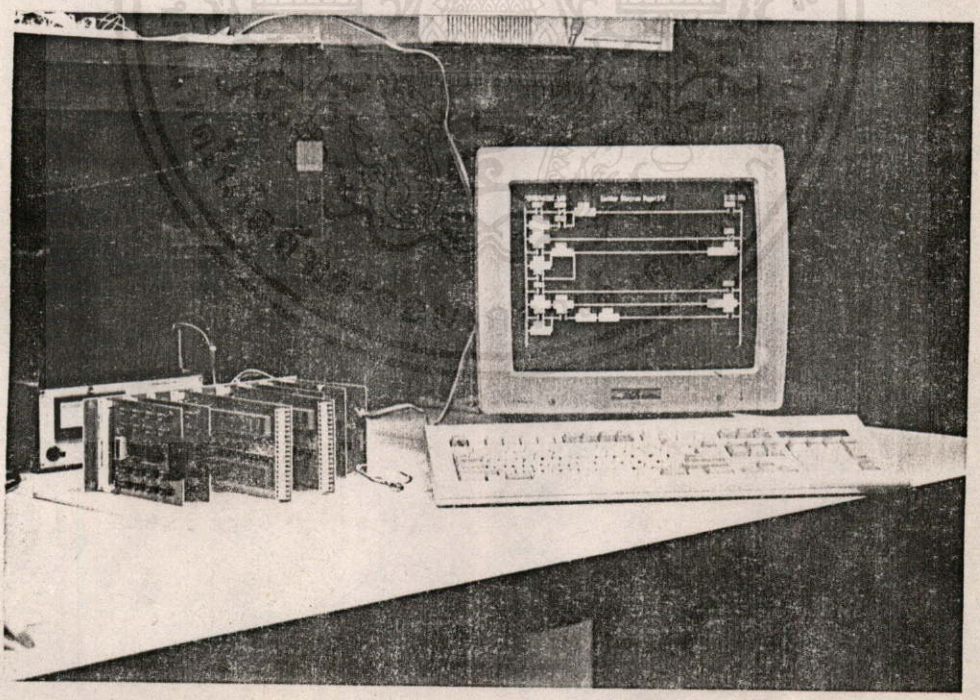
รูปที่ 8.9 แสดงฟังก์ชั่นควบคุม



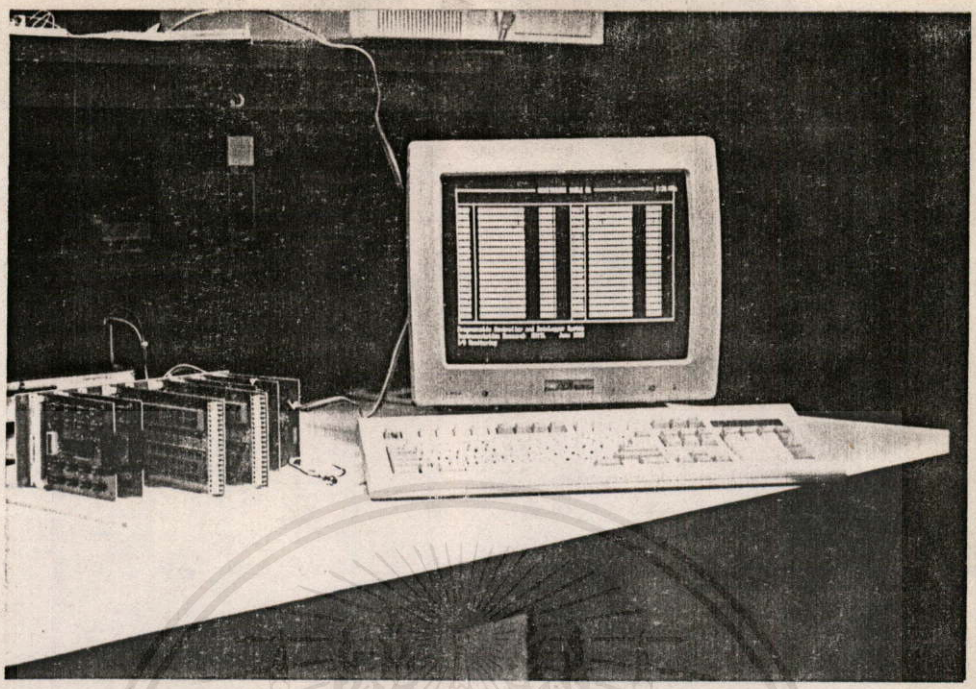
รูปที่ 8.10 แสดงตารางของชุดข้อมูลแบบบูลลีนหลังจาก
การแปลความหมายจากแลดเดอร์ไดอะแกรมแล้ว



รูปที่ 8.11 แสดงตารางการแสดงผลค่าสภาวะของเครื่องควบคุม



รูปที่ 8.12 แสดงการจำลองแบบการทำงานของ
เครื่องควบคุมต่อร่วมกับเครื่องไมโครคอมพิวเตอร์แสดงผลสภาวะแบบแลดเดอร์ไดอะแกรม



รูปที่ 8.13 แสดงการจำลองแบบการทำงานของเครื่องควบคุม
ต่อร่วมกับเครื่องไมโครคอมพิวเตอร์แสดงผลแบบตารางแสดงสภาวะ

บทที่ 9
บทสรุป

เครื่องควบคุมที่โปรแกรมได้เป็นการประยุกต์ใช้ไมโครโปรเซสเซอร์ในการควบคุมกระบวนการที่มีประสิทธิภาพวิธีหนึ่ง จึงได้มีการนำเครื่องควบคุมไปใช้กับกระบวนการทางอุตสาหกรรมกันอย่างแพร่หลาย เนื่องจากสามารถเปลี่ยนแปลงลำดับการทำงานหรือการควบคุมได้ทันที ดังนั้นการออกแบบและการซ่อมบำรุงจึงทำได้อย่างง่าย นอกจากนั้นยังสามารถเชื่อมต่อกับคอมพิวเตอร์ เพื่อนำผลหรือค่าสภาวะการควบคุมมาแสดงผลหรือประมวลผลสำหรับการควบคุมคุณภาพหรือควบคุมการผลิตได้สะดวก

วิทยานิพนธ์ฉบับนี้ได้เสนอการออกแบบและสร้างเครื่องควบคุมที่โปรแกรมได้ ซึ่งได้ออกแบบให้มีราคาถูกและใช้งานได้ง่าย ดังนั้นจึงได้พัฒนาโปรแกรมเพื่อใช้ในการโปรแกรมแบบแลดเดอร์ไดอะแกรมขึ้น เนื่องจากการโปรแกรมแบบแลดเดอร์เป็นวิธีการโปรแกรมที่มีประสิทธิภาพมาก โปรแกรมดังกล่าวได้พัฒนาอยู่บนไมโครคอมพิวเตอร์ IBM PC ซึ่งแพร่หลายกันอยู่ทั่วไปทำให้ระบบมีราคาถูกลง นอกจากการโปรแกรมแบบแลดเดอร์แล้วก็ยังสามารถโปรแกรมแบบบูลลีนได้อีกด้วย การโปรแกรมแบบนี้ถึงแม้ว่าจะมีประสิทธิภาพด้อยกว่าการโปรแกรมแบบแลดเดอร์อยู่บ้างแต่ในบางกรณี การใช้โปรแกรมแบบบูลลีนจะสะดวกและรวดเร็วกว่า ในการออกแบบได้กำหนดให้จอภาพแสดงภาพของหน่วยป้อนโปรแกรม (PROGRAM LOADER) ของเครื่องควบคุมที่มีจำหน่ายอยู่ในท้องตลาด การป้อนโปรแกรมกระทำได้ทั้งการเลื่อนปุ่มลูกศร (ARROW KEY) ไปยังตำแหน่งที่ต้องการหรือการใช้ปากกาแสง (LIGHT PEN) ในการป้อนโปรแกรม การใช้ปากกาแสงนี้เองจะทำให้การโปรแกรมสะดวกขึ้นอีกมาก แต่การใช้ปากกาแสงเองก็มีข้อเสีย ซึ่งเป็นคุณสมบัติเฉพาะตัวของมันเอง เพราะว่าปากกาแสงต้องการแสงในการทำงานค่อนข้างสูง ดังนั้นเมื่อจะใช้ปากกาแสงก็ควรจะปรับจอภาพให้แสงสว่างเพิ่มขึ้นเล็กน้อย และจะต้องใช้ปากกาแสงตรงตำแหน่งที่จอภาพมีตัวอักษรอยู่ เป็นต้น นอกจากที่ได้กล่าวมาแล้วข้างต้น เมื่อผู้ใช้เขียนโปรแกรมแล้วส่งชุดคำสั่งให้กับเครื่องควบคุม ขั้นตอนต่อไปก็คือการตรวจสอบการทำงานของโปรแกรม ซึ่งได้ออกแบบให้มีฟังก์ชั่นนี้อยู่ด้วย นั่นคือฟังก์ชั่นการติดตามและตรวจสอบค่าการเปลี่ยนแปลงสภาวะของกระบวนการจึงทำให้การตรวจสอบโปรแกรมทำได้โดยสะดวก

อย่างไรก็ดีการออกแบบก็ยังมีข้อบกพร่องอยู่บ้าง เช่น การสร้างฟังก์ชั่นพิเศษต่าง ๆ นั้นยังสร้างไว้ไม่มากนัก เนื่องจากการพัฒนาได้เลือกเฉพาะฟังก์ชั่นที่จำเป็นก่อน และแนวทางในการพัฒนาอีกทางหนึ่งคือการออกแบบและพัฒนาหน่วยอินพุทเอาท์พุทแบบต่าง ๆ เช่น

หน่วยอินพุทความเร็วสูง หน่วยประมวลผลแบบ PID เป็นต้น และส่วนของการติดต่อสื่อสารข้อมูลสำหรับการโปรแกรม หรือการรับส่งค่าสภาวะ เพื่อการแสดงผลควรจะมีไมโครโปรเซสเซอร์เข้ามาร่วมในการทำงานเกี่ยวกับการสื่อสารข้อมูลดังกล่าวนี้โดยเฉพาะ ซึ่งจะทำให้การทำงานของเครื่องควบคุมมีประสิทธิภาพมากยิ่งขึ้น และเป็นผลให้ความผิดพลาดอันเนื่องมาจากการสื่อสารลดน้อยลง อีกทั้งยังเป็นแนวทางในการที่จะสร้างหรือพัฒนาระบบให้สามารถอยู่ในรูปแบบของการทำงานในลักษณะโครงข่ายได้ (NETWORK)

กิตติกรรมประกาศ
(ACKNOWLEDGEMENT)

วิทยานิพนธ์ฉบับนี้สำเร็จลงได้ก็ด้วยความร่วมมือจากหลายฝ่ายด้วยกัน ดังนั้นจึงขอขอบพระคุณ รองศาสตราจารย์ กิตติ ตีรเศรษฐ ซึ่งเป็นอาจารย์ที่ปรึกษาและอาจารย์ท่านอื่น ๆ รวมทั้งเพื่อนร่วมงานทุกคน ที่ได้ให้ความช่วยเหลือจนทำให้วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปด้วยดี

เอกสารอ้างอิง
REFERENCES

1. เอกสารประกอบการอบรมเชิงปฏิบัติการ PROGRAMMABLE LOGIC CONTROLLER & PROGRAMMABLE CONTROLLER ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง

2. สุเธียร เกียรติสุนทร หลักการทำงานและเทคนิคการประยุกต์ใช้งาน PC/PLC ภาควิชาวิศวกรรมระบบควบคุม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง

3. THOMAS E. KISSELL, UNDERSTANDING AND USING PROGRAMMABLE CONTROLLERS, Prentice - Hall International,Inc., 1986

4. Jones Clarence T., Bryan Luis A., PROGRAMMABLE CONTROLLER CONCEPTS AND APPLICATION, International Programmable Controls, 1983

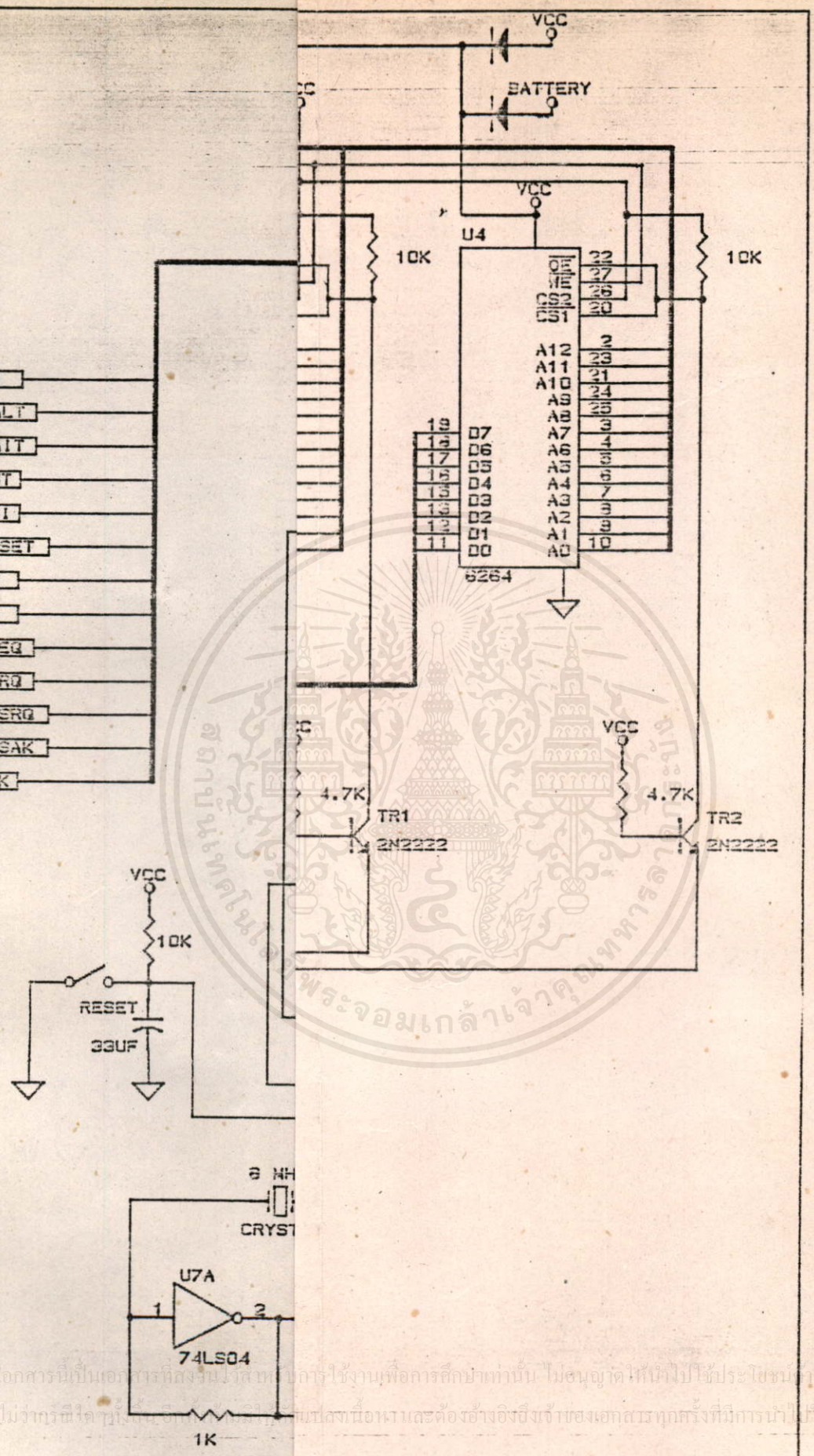
5. TURBO PASCAL VERSION 4.0 REFERANCE MANUAL, Borland International Inc., 1988

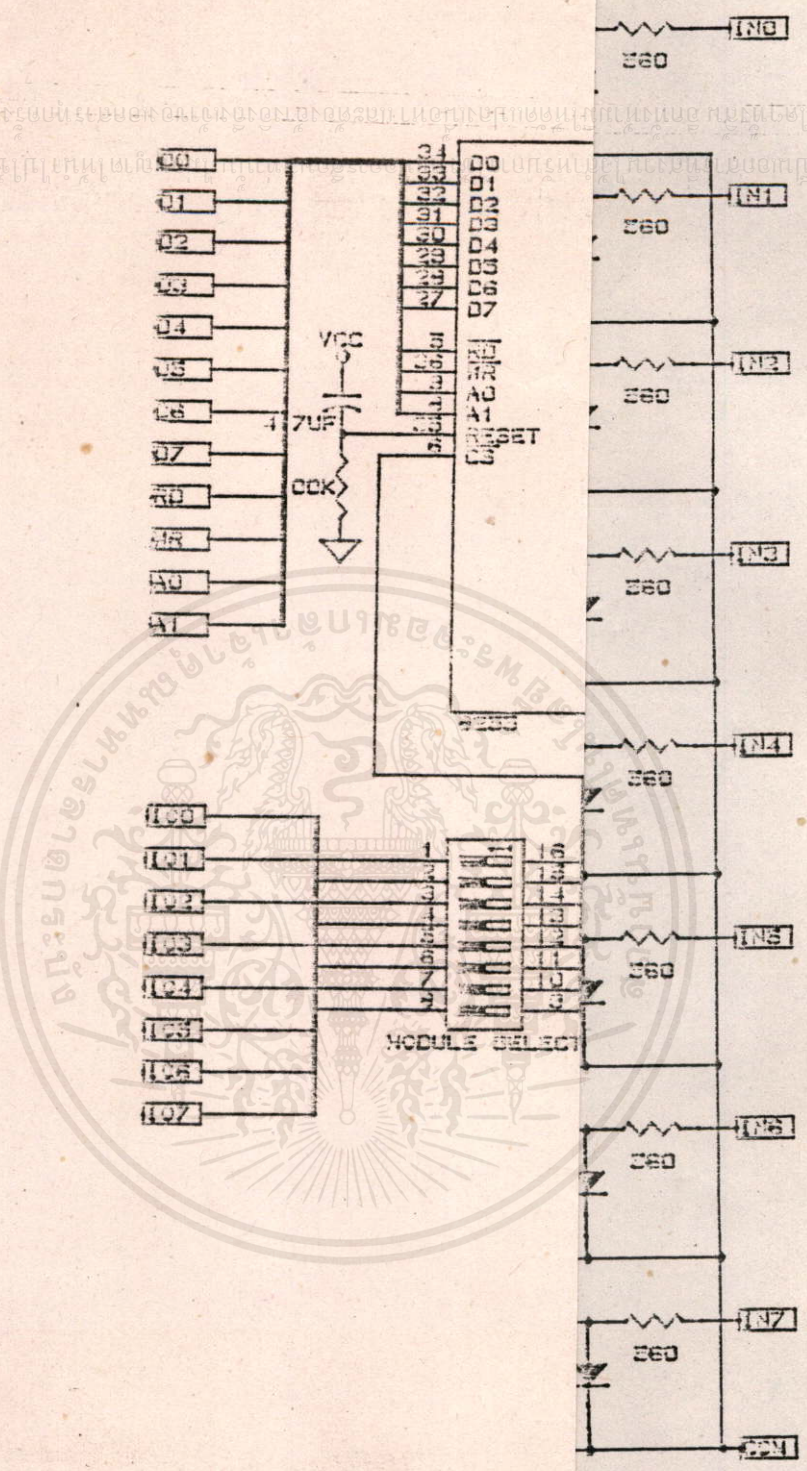
6. Lance A. Levenhal, Winthrop Sarille, Z80 Assembly Language Subroutine

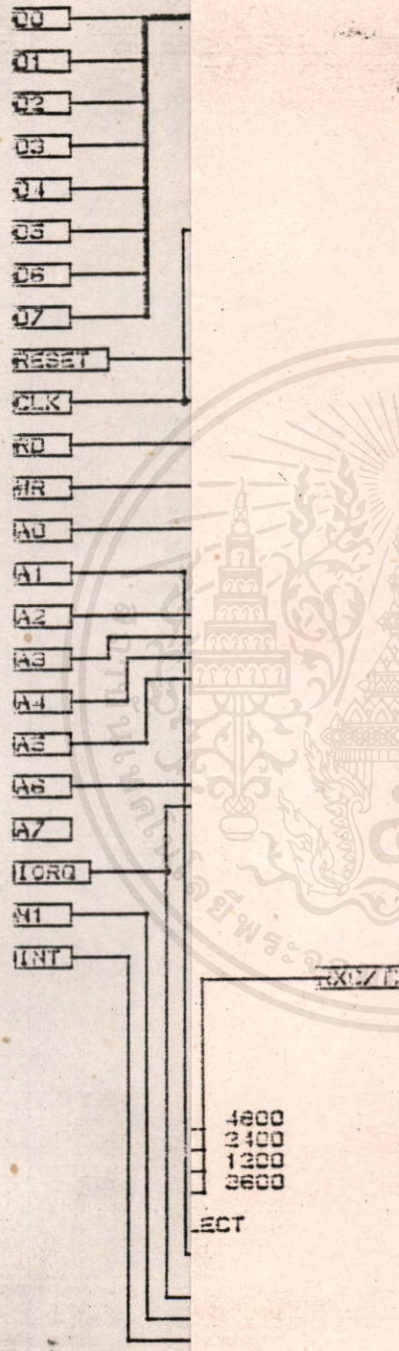
7. Microsystem Component Handbook, Volume II,INTEL

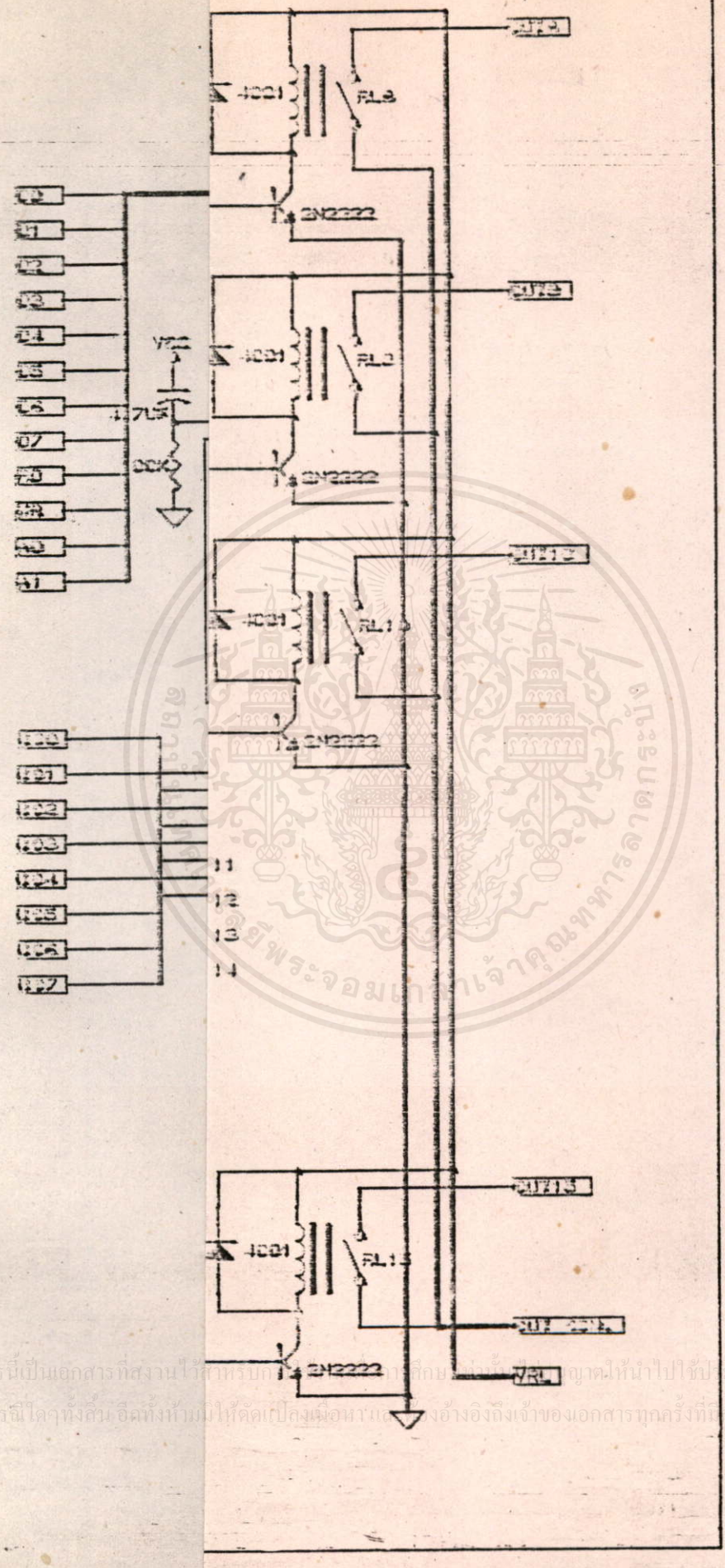
8. Low Power Schottky TTL ICs, Data Book, SGS Technology and Service, June 1985

ภาคผนวก 1




D0
D1
D2
D3
D4
D5
D6
D7
VCC
7UF
00K
RD
WR
A0
A1
RESET
CS
MODULE SELECT
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7
OPM
360





ภาคผนวก 2

```
Text File List Processing Program V4.00C Page : 1
File : D:\X80\PCSYS.ASM
Current Date : Monday November 7, 1988
Current Time : 2:13 PM

Program : System Controller
Programmer : Viriya Kongratana

;PROGRAM SYSTEM CONTROLLER UNIT
;------------------CONSTANT------------------------------
INPCODE         EQU 9BH
OUTCODE         EQU 90H
CTC_CH0         EQU 38H
CTC_CH1         EQU 39H
CTC_CH2         EQU 3AH
CTC_CH3         EQU 3BH
UART_DTA        EQU 3CH
UART_CNT        EQU 3DH
CNT_IO0         EQU 03H         ;PORT SCAN I/O
PRTYP0          EQU 00H         ;PORT TYPE IN OR OUT
POWERCD         EQU 55H         ;POWERUP CODE
LENEXEC         EQU 2000
LENEDIT         EQU 2000
_LD             EQU 00
LDNOT           EQU 01
_AND            EQU 02
ANDNOT          EQU 03
_OR             EQU 04
ORNOT           EQU 05
ANDLD           EQU 06
ORLD            EQU 07
_OUT            EQU 08
OUTNOT          EQU 09
TIM             EQU 10
CNT             EQU 11
LDTIM           EQU 12
LDNTIM          EQU 13
LDCNT           EQU 14
LDNCNT          EQU 15
ANDTIM          EQU 16
ANDNTIM         EQU 17
ANDCNT          EQU 18
ANDNCNT         EQU 19
ORTIM           EQU 20
ORNTIM          EQU 21
ORCNT           EQU 22
ORNCNT          EQU 23
;(FUN Command bit 7 is Set)
_NOP            EQU 128     ;(FUN 00)
END             EQU 129     ;(FUN 01)
IL              EQU 130
ILC             EQU 131
JMP             EQU 132
```

```
JME             EQU 133
SFT             EQU 138     ;(FUN 10)
KEEP            EQU 139
CNTR            EQU 140
DIFU            EQU 141
DIFD            EQU 142
TIMH            EQU 143
WSFT            EQU 144
CMP             EQU 148
MOV             EQU 149
MVN             EQU 150
BIN             EQU 151
BCD             EQU 152
ASL             EQU 153
ASR             EQU 154
ROL             EQU 155
ROR             EQU 156
COM             EQU 157
ADD             EQU 158
SUB             EQU 159
MUL             EQU 160
DIV             EQU 161
ANDW            EQU 162
ORW             EQU 163
XORW            EQU 164
XNRW            EQU 165
INC             EQU 166
DEC             EQU 167
STC             EQU 168
CLC             EQU 169
FUN70           EQU 198
FUN71           EQU 199
FUN72           EQU 200
FUN73           EQU 201
FUN74           EQU 202
FUN75           EQU 203
FUN76           EQU 204
FUN77           EQU 205
FUN78           EQU 206
FUN79           EQU 207
FUN80           EQU 208
FUN81           EQU 209
FUN82           EQU 210
FUN83           EQU 211
FUN84           EQU 212
FUN85           EQU 213
FUN94           EQU 222
FUN95           EQU 227

;------------------VARIABLE-----------------------------
SYS_STACK       EQU 0F7FFH    ;SYSTEM STACK
BIN_BUFFER      EQU 0F800H    ;BYTE
BCD_BUFFER      EQU 0F808H    ;BYTE
```

```
MODULE0         EQU 0F810H    ;BYTE
MODULE1         EQU 0F811H    ;BYTE
MODULE2         EQU 0F812H    ;BYTE
MODULE3         EQU 0F813H    ;BYTE
MODULE4         EQU 0F814H    ;BYTE
MODULE5         EQU 0F815H    ;BYTE   I/O ASSIGN FROM USER
MODULE6         EQU 0F816H    ;BYTE
MODULE7         EQU 0F817H    ;BYTE
MODULE8         EQU 0F818H    ;BYTE
MODULE9         EQU 0F819H    ;BYTE
MODULE10        EQU 0F81AH    ;BYTE
MODULE11        EQU 0F81BH    ;BYTE   DIGITAL TO ANALOG [OUTPUT]
MODULE12        EQU 0F81CH    ;BYTE   ANALOG TO DIGITAL [INPUT]
MODULE13        EQU 0F81DH    ;BYTE   BUS_REQUEST;

IOCH0           EQU 0F820H    ;WORD
IOCH1           EQU 0F822H    ;WORD
IOCH2           EQU 0F824H    ;WORD
IOCH3           EQU 0F826H    ;WORD
IOCH4           EQU 0F828H    ;WORD
IOCH5           EQU 0F82AH    ;WORD
IOCH6           EQU 0F82CH    ;WORD
IOCH7           EQU 0F82EH    ;WORD
IOCH8           EQU 0F830H    ;WORD
IOCH9           EQU 0F832H    ;WORD
IOCH10          EQU 0F834H    ;WORD
IOCH11          EQU 0F836H    ;WORD
IOCH19          EQU 0F846H    ;WORD TIMBASE CH. [INTERNAL CLOCK]
                              ;ARRAY [0..33] OF WORD INTERNAL RELAY
TIMCH0          EQU 0F864H    ;ARRAY [0..16] OF WORD
CNTCH0          EQU 0F886H    ;ARRAY [0..16] OF WORD
TIMSTA0         EQU 0F8A8H    ;ARRAY [0..16] OF BYTE
CNTSTA0         EQU 0F8B9H    ;ARRAY [0..16] OF BYTE
CORDSTA0        EQU 0F8CAH    ;ARRAY [0..16] OF BYTE

EDITPOI         EQU 0FA00H    ;WORD
EXECPOI         EQU 0FA02H    ;WORD
ADDRTEMP        EQU 0FA04H    ;WORD
PARAMTR1        EQU 0FA06H    ;WORD
PARAMTR2        EQU 0FA08H    ;WORD
PARAMTR3        EQU 0FA0AH    ;WORD
PARAMNO         EQU 0FA0CH    ;BYTE
ADDLOOP         EQU 0FA0DH    ;BYTE
POWERUP         EQU 0FA10H    ;BYTE
TYPEDTA1        EQU 0FA11H    ;BYTE
TYPEDTA2        EQU 0FA12H    ;BYTE
TYPEDTA3        EQU 0FA13H    ;BYTE
RETADDR         EQU 0FA14H    ;WORD
RETADDR1        EQU 0FA16H    ;WORD
LASTTIM         EQU 0FA18H    ;BYTE
EDTRANGE        EQU 0FA19H    ;WORD
LASTCNT0        EQU 0FA21H    ;ARRAY [1..48] OF BYTE
IOPORTCH0       EQU 0FA50H    ;ARRAY [0..10] OF WORD
```

```
LSTSTATEO       EQU OFA70H    ;ARRAY [0..20] OF BYTE
DIFSTA          EQU OFA90H    ;ARRAY [0..20] OF BYTE
LSTSTAPOI       EQU OFAB1H    ;BYTE
DIFPOI          EQU OFAB2H    ;BYTE
ILFLAG          EQU OFAB3H    ;BYTE

EDITMEM         EQU OE000H    ;EDIT MEMORY LOCATION DEFM 4096 BYTE
EXECMEM         EQU OE800H    ;EXEC MEMORY LOCATION DEFM 4096 BYTE
;_____________________________________________________________
START:          LD BC,OFFFFH
RE_START:       CPD
                JP PE,RE_START
                JP MAIN

INIT_TIMBASE:   LD A,37H
                OUT (CTC_CH0),A
                LD A,80
                OUT (CTC_CH0),A
                LD A,57H
                OUT (CTC_CH1),A
                LD A,0FFH
                OUT (CTC_CH1),A
                LD A,57H
                OUT (CTC_CH2),A
                LD A,0FFH
                OUT (CTC_CH2),A
                RET
;_____________________________________________________________
RECEIVE:        IN A,(UART_CNT)
                BIT 1,A
                JR Z,RECEIVE
                IN A,(UART_DTA)
                RET

TRANSMIT:       PUSH AF
TRANS1:         IN A,(UART_CNT)
                BIT 0,A
                JR Z,TRANS1
                POP AF
                OUT (UART_DTA),A
                CALL DELAY
                RET

DELAY:          PUSH BC
                PUSH AF
                LD BC,200H
DEL1:           DEC BC
                LD A,B
                OR C
                JR NZ,DEL1
                POP AF
                POP BC
                RET
;_____________________________________________________________
```

```
DWLOAD:     CALL RECEIVE
            LD (EDTRANGE),A
            CALL RECEIVE
            LD (EDTRANGE+1),A  ;HI BYTE
            LD HL,EDITMEM
            LD BC,(EDTRANGE)
DWLOAD1:    CALL RECEIVE
            LD (HL),A
            INC HL
            DEC BC
            LD A,B
            OR C
            JP NZ,DWLOAD1
            RET

SCNPROG:    LD D,0BH            ;SCAN 11 MODULE
            LD E,04H
            LD C,CNT_IOO        ;START MODULE 0
SCNPROG1:   LD A,INPCODE        ;SCAN PROG.CONTROL WORD
            OUT (C),A
            LD A,C
            ADD A,E
            LD C,A
            DEC D
            JP NZ,SCNPROG1
            RET

GETPORT:    LD D,0BH
            LD E,04H
            LD C,PRTYPO      ;PORT TYPE START MODULE 0
            LD HL,MODULE0
GETPORT1:   IN A,(C)
            LD (HL),A           ;STOREAGE IN TABLE
            LD A,C
            ADD A,E
            LD C,A
            INC HL
            DEC D
            JP NZ,GETPORT1
            RET

SETIO:      LD D,0BH
            LD E,04H
            LD C,CNT_IOO
            LD HL,MODULE0
SETIO2:     LD A,(HL)
            CP 0F0H             ;CMP OUTPUT MODULE
            JP Z,SETOUT
            CP 0FFH             ;NO MODULE CARD I/O
            JP Z,SETIO1
SETINPUT:   LD A,INPCODE
            OUT (C),A
SETIO1:     LD A,C
            ADD A,E
```

```
                LD C,A
                INC HL
                DEC D
                JP NZ,SETIO2
                RET

SETOUT:         LD A,OUTCODE
                OUT (C),A
                JP SETIO1

CHECKIO:        CALL SCNPROG        ;SCAN PROG.PORT 8255
                CALL GETPORT        ;GET PORT TYPE
                CALL SETIO          ;SEPARATE INPUT OUTPUT MODULE
                RET
;_____________________________________________________________
DISOUT:         LD D,0BH
                LD E,04H
                LD C,01H            ;POINT I/O PORT 00 MODULE 0
                LD HL,MODULE0
DIS3:           LD A,(HL)
                CP 0F0H             ;CMP OUTPUT MODULE
                JP Z,DIS1
DIS2:           LD A,C
                ADD A,E
                LD C,A
                INC HL
                DEC D
                JP NZ,DIS3
                RET

DIS1:           LD A,00
                OUT (C),A
                INC C
                OUT (C),A
                DEC C
                JP DIS2
;_____________________________________________________________
BCD2BN:         PUSH AF
                LD A,B
                LD B,A              ;INPUT  B  BCD
                AND 0F0H            ;EXIT:  B  BIN
                RRCA
                LD C,A
                RRCA
                RRCA
                ADD A,C
                LD C,A
                LD A,B
                AND 0FH
                ADD A,C
                LD B,A
                POP AF
                RET
```

```
BCDBIN:     LD C,32
DBLP:       LD B,5
            XOR A
            LD HL,BCD_BUFFER
CORO:       LD A,(HL)
            RRA
            PUSH AF
            BIT 7,A
            JR Z,COR1
            SUB 30H
COR1:       BIT 3,A
            JR Z,COR2
            SUB 3
COR2:       LD (HL),A
            DEC HL
            POP AF
            DJNZ CORO
            LD B,4
SHR4        RR (HL)
            DEC HL
            DJNZ SHR4
            DEC C
            JR NZ,DBLP
            RET

BINBCD:     XOR A
            LD B,E
            LD HL,BCD_BUFFER
CLR:        LD (HL),A
            INC HL
            DJNZ CLR
            LD A,D
            ADD A,A
            ADD A,A
            ADD A,A
            LD C,A
LOOP:       LD L,0
            LD B,D
SHLB:       RL (HL)
            INC HL
            DJNZ SHLB
            LD L,8
            LD B,E
BCDADJ:     LD A,(HL)
            ADC A,A
            DAA
            LD (HL),A
            INC HL
            DJNZ BCDADJ
            DEC C
            JR NZ,LOOP
            RET

CONVERT:    PUSH AF
```

```
                PUSH BC
                PUSH DE
                PUSH HL
                PUSH IY

                LD B,0AH
                LD HL,BIN_BUFFER
CLRBUFFER:      LD (HL),00
                INC HL
                DJNZ CLRBUFFER

                LD E,(IX+00)
                INC IX
                LD D,(IX+00)
                DEC IX
                LD (BIN_BUFFER),DE
                LD DE,0203H
                CALL BINBCD
                LD DE,BCD_BUFFER
                LD A,(DE)
                LD (IX+00),A
                INC IX
                INC DE
                LD A,(DE)
                LD (IX+00),A
                DEC IX
                POP IY
                POP HL
                POP DE
                POP BC
                POP AF
                RET
;_____________________________________________________________________
PUTPARAM:       LD IX,PARAMTR1          ;PARAMMETER ADDR TEMP
                LD IY,TYPEDTA1          ;TYPE OF DATA INTEGER OR BCD,HEX
                LD BC,(EDITPOI)
                INC BC
                LD (EDITPOI),BC         ;INC POINTER

PUTPAM1:        LD HL,EDITMEM
                LD BC,(EDITPOI)
                ADD HL,BC
                LD C,(HL)
                LD (IY+00),C        ;SAVE TYPE_DATA
                INC HL              ;INC EDITMEM
                INC IY              ;INC TYPE STORAGE
                LD C,(HL)
                INC HL
                LD B,(HL)
                LD (IX+00),C
                INC IX
                LD (IX+00),B
                INC IX
```

```
                LD BC,(EDITPOI)
                INC BC
                INC BC
                INC BC
                LD (EDITPOI),BC          ;INC POINTER

                DEC A
                JP NZ,PUTPAM1

                LD HL,EXECMEM
                LD DE,(EXECPOI)
                ADD HL,DE
                LD (HL),21H              ;CODE LD HL,XXXX
                INC HL
                LD IX,PARAMTR1
                LD IY,TYPEDTA1
                LD A,(IY+00)
                CP 00
                JP NZ,PUTDTA1
                CALL CONVERT             ;CONVERT TO BCD
                LD B,(IX+00)
                CALL BCD2BN
PUTDTA4:        LD (HL),B
                INC IX
                INC HL
                LD B,(IX+00)
                CALL BCD2BN
                LD (HL),B
                INC DE
                INC DE
                INC DE
                LD (EXECPOI),DE
                LD A,(PARAMNO)
                DEC A
                LD (PARAMNO),A
                RET Z

                LD HL,EXECMEM
                LD DE,(EXECPOI)
                ADD HL,DE
                LD (HL),11H              ;CODE LD DE,XXXX
                INC HL
                LD IX,PARAMTR2
                LD IY,TYPEDTA2
                LD A,(IY+00)
                CP 00
                JP NZ,PUTDTA2
                CALL CONVERT
                LD B,(IX+00)
                CALL BCD2BN
PUTDTA5:        LD (HL),B
                INC IX
                INC HL
                LD B,(IX+00)
```

```
                CALL BCD2BN
                LD (HL),B
                INC DE
                INC DE
                INC DE
                LD (EXECPOI),DE
                LD A,(PARAMNO)
                DEC A
                LD (PARAMNO),A
                RET Z

                LD HL,EXECMEM
                LD DE,(EXECPOI)
                ADD HL,DE
                LD (HL),01H             ;CODE LD BC,XXXX
                INC HL
                LD IX,PARAMTR3
                LD IY,TYPEDTA3
                LD A,(IY+00)
                CP 00
                JP NZ,PUTDTA3
                CALL CONVERT
                LD B,(IX+00)
                CALL BCD2BN
PUTDTA6:        LD (HL),B
                INC IX
                INC HL
                LD B,(IX+00)
                CALL BCD2BN
                LD (HL),B
                INC DE
                INC DE
                INC DE
                LD (EXECPOI),DE
                LD A,(PARAMNO)
                DEC A
                LD (PARAMNO),A
                RET Z
                RET

PUTDTA1:        LD B,(IX+00)
                JP PUTDTA4
PUTDTA2:        LD B,(IX+00)
                JP PUTDTA5
PUTDTA3:        LD B,(IX+00)
                JP PUTDTA6

CODEEXE:        LD HL,EXECMEM
                LD DE,(EXECPOI)
                ADD HL,DE
                LD (HL),0CDH            ;CODE CALL XXXX
                LD IX,ADDRTEMP
                INC HL
                INC DE                  ;INC EXEC POINTER
```

```
                LD B,(IX+00)
                LD (HL),B
                INC IX
                INC HL
                INC DE
                LD B,(IX+00)
                LD (HL),B
                INC IX
                INC HL
                INC DE
                LD (EXECPOI),DE
                RET

PUTCODE:        LD (ADDRTEMP),BC        ;INST ADDR TEMP
                LD (PARAMNO),A          ;NO OF PARAMETER
                CP 00                   ;CHECK NO PARAM
                JP Z,PUTCDRET
                CALL PUTPARAM
PUTCODE1:       CALL CODEEXE
                RET

PUTCDRET:       LD BC,(EDITPOI)
                INC BC
                LD (EDITPOI),BC
                JR PUTCODE1
;_____________________________________________________________________
CDINST:         LD HL,0000H
                LD (EDITPOI),HL     ;CLR POINTER
                LD (EXECPOI),HL
CDINST1:        LD HL,EDITMEM
                LD DE,(EDITPOI)
                ADD HL,DE
                LD A,(HL)
                CP END
                JP Z,ENDCOM         ;IF INS = 'END' THEN ENDCOMPILE
                CALL INSASGN
                JP CDINST1

ENDCOM:         LD HL,EXECMEM
                LD DE,(EXECPOI)
                ADD HL,DE
                LD (HL),0C3H        ;CODE JP XXXX
                INC HL
                LD IY,IOSCAN        ;ADDR I/O SERVICE
                LD (ADDLOOP),IY
                LD A,(ADDLOOP)
                LD (HL),A           ;ADDR LOOP SCANTIME LOW
                LD A,(ADDLOOP+1)
                INC HL
                LD (HL),A           ;ADDR LOOP SCANTIME HI
                INC DE
                INC DE
                INC DE
                LD (EXECPOI),DE
```

```
                LD DE,(EDITPOI)
                INC DE
                LD (EDITPOI),DE
                RET

COMPILE:        CALL CDINST
                RET
;_____________________________________________________________
CLREXE:         LD BC,LENEXEC
                LD HL,EXECMEM
CLREXE1:        LD (HL),00H         ;NOP INSTRUCTION
                INC HL
                DEC BC
                LD A,C
                OR B
                JR NZ,CLREXE1
                RET

CLREDIT:        LD BC,LENEDIT
                LD HL,EDITMEM
CLREDT1:        LD (HL),END         ;END INSTRUCTION
                INC HL
                DEC BC
                LD A,C
                OR B
                JR NZ,CLREDT1
                RET

CLRMEM:         CALL CLREXE           ;CLR EXECMEM
                CALL CLREDIT          ;CLR EDITMEM
                CALL CLRTCIO          ;ALL CLR
                LD A,POWERCD          ;SAVE POWER UP
                LD (POWERUP),A
                RET

MOVEIO:         LD BC,22        ;11 PORT ADDRESS
                LD DE,IOPORTCHO
                LD HL,IOCHO
                LDIR
                RET
;_____________________________________________________________
MONITOR:        IN A,(UART_DTA) ;START CODE 1 ADDR 0
                CP 0FEH         ;BREAK EXECUTE FOR CONTROLLER CODE
                JP Z,MAIN       ;MAIN
                CP 0FFH
                RET Z
                LD D,00H
                LD E,A
                LD HL,IOCHO
                ADD HL,DE
                LD A,(HL)
                CALL TRANSMIT
                INC HL
                LD A,(HL)
```

```
                CALL TRANSMIT
                RET
;_____________________________________________________________
CLRIO:          LD HL,IOPORTCH0
                LD B,22
CLRIO1:         LD (HL),00
                INC HL
                DJNZ CLRIO1
                RET

CLRDTACH:       LD HL,IOCH0
                LD B,68
CLRDTA1:        LD (HL),00
                INC HL
                DJNZ CLRDTA1
                RET

CLRTIM:         LD HL,TIMCH0
                LD B,34
CLRTIM1:        LD (HL),00
                INC HL
                DJNZ CLRTIM1
                RET

CLRCNT:         LD HL,CNTCH0
                LD B,34
CLRCNT1:        LD (HL),00
                INC HL
                DJNZ CLRCNT1
                RET

CLRINT:         LD HL,IOCH0
                LD B,68     ;CLR INTERNAL RELAY 48 CH [0..47]
CLRINT1:        LD (HL),00
                INC HL
                DJNZ CLRINT1
                RET

CLRLST:         LD HL,IOCH0
                LD B,68     ;CLR DATA MEM LAST STATE
CLRLST1:        LD (HL),00
                INC HL
                DJNZ CLRLST1
                RET

CLRTCIO:        CALL CLRIO
                CALL CLRDTACH
                CALL CLRTIM
                CALL CLRCNT
                RET
;_____________________________________________________________
MAIN:           LD SP,SYS_STACK
                CALL INIT_TIMBASE
                CALL CHECKIO        ;I/O INSTALLATION
```

```
                CALL DISOUT             ;DISABLE OUTPUT
                LD A,(POWERUP)
                CP POWERCD
                CALL NZ,CLRMEM          ;CLEARMEM
                CALL CLRIO
SCAN:           CALL RECEIVE
                CP 'L'
                CALL Z,DWLOAD
                CP 'C'
                CALL Z,COMPILE          ;COMPILE CODE TO CONTROLLER
                CP 'R'
                JP Z,EXECUTE            ;EXECUTE
                JP SCAN

EXECUTE:        CALL CLRINT
                CALL CLRLST
                CALL CLRIO
                JP EXECMEM
;_____________________________________________________________________
IOSCAN:         LD SP,SYS_STACK
                CALL WATCHDOG   ;REFRESH TIMEBASE
                CALL MONITOR
                LD D,0BH
                LD E,04H        ;OFFSET MODULE
                LD C,01H        ;POINT MODULE 0
                LD IX,IOCH0
                LD HL,MODULE0
IOSCAN3:        LD A,(HL)
                CP 0FFH
                JP Z,IOSCAN4
                CP 0F0H         ;OUTPUT MODULE
                JP Z,OUTSCAN
                CP 0FH
                JP Z,INSCAN
                JP IOSCAN4
INSCAN:         IN A,(C)
                CPL
                LD (IX+00),A
                INC C
                INC IX
                IN A,(C)
                CPL
                LD (IX+00),A
                INC IX
                DEC C
IOSCAN1:        LD A,C
                ADD A,E
                LD C,A
                INC HL
                DEC D
                JP NZ,IOSCAN3
                CALL CLRIO              ;REFRESH I/O
                LD A,00
                LD (LSTSTAPOI),A        ;RESET POINT LAST STATE
```

```
                JP EXECMEM          ;1 SCANTIME IN/OUT

OUTSCAN:        LD A,(IX+00)
                OUT (C),A
                INC C
                INC IX
                LD A,(IX+00)
                INC IX
                OUT (C),A
                DEC C
                JP IOSCAN1

IOSCAN4:        INC IX
                INC IX
                JP IOSCAN1

WATCHDOG:       LD HL,IOCH19
                IN A,(CTC_CH1)
                LD (HL),A
                INC HL
                IN A,(CTC_CH2)
                LD (HL),A
                IN A,(CTC_CH1)
                AND 01H
                JR Z,SET0
SET1:           LD A,0FFH
                LD HL,LASTTIM
                LD B,(HL)
                LD (HL),A
                JR CHKTIM1
SET0:           LD A,00H
                LD HL,LASTTIM
                LD B,(HL)
                LD (HL),A
                JR CHKTIM1
;_____________________________________________
CHKTIM1:        XOR B
                EX AF,AF'
                LD A,B
                CPL
                LD B,A
                EX AF,AF'
                AND B
                RET Z
DECTIM:         LD B,00H
DECTIM1:        LD E,B
                LD D,00H
                LD HL,TIMSTA0
                ADD HL,DE
                LD A,(HL)
                CP 0FFH
                JR Z,NOTDEC
                LD A,B
                ADD A,A
```

```
                LD E,A
                LD D,00H
                LD HL,TIMCH0
                ADD HL,DE
                INC HL
                BIT 7,(HL)
                JR Z,NOTDEC
                LD A,7FH
                AND (HL)
                LD D,A
                DEC HL
                LD E,(HL)
                DEC DE      ;COUNTDOWN
                LD (HL),E
                INC HL
                LD (HL),D
NOTDEC:         INC B
                LD A,17       ;17 CH
                XOR B
                JR NZ,DECTIM1
                RET
;_______________________________________________________________
                END
```

File : D:\X80\PCSUB.ASM
Current Date : Monday November 7, 1988
Current Time : 2:17 PM

Program : PC Subroutine
Programmer : Viriya Kongratana

```
;PROGRAMMABLE CONTROLLER SUBROUTINE

INSASGN:    LD A,(HL)           ;GET INST FROM EDITMEM POINT BY EDITPOI
            CP _LD
            JP Z,LDINS
            CP LDNOT
            JP Z,LDNINS
            CP _AND
            JP Z,ANDINS
            CP ANDNOT
            JP Z,ANDNINS
            CP _OR
            JP Z,ORINS
            CP ORNOT
            JP Z,ORNINS
            CP ANDLD
            JP Z,ANDLDINS
            CP ORLD
            JP Z,ORLDINS
            CP _OUT
            JP Z,OUTINS
            CP OUTNOT
            JP Z,OUTNINS
            CP TIM
            JP Z,TIMINS
            CP CNT
            JP Z,CNTINS
            CP LDTIM
            JP Z,LDTIMINS
            CP LDNTIM
            JP Z,LDNTIMINS
            CP LDCNT
            JP Z,LDCNTINS
            CP LDNCNT
            JP Z,LDNCNTINS
            CP ANDTIM
            JP Z,ANDTIMINS
            CP ANDNTIM
            JP Z,ANDNTIMINS
            CP ANDCNT
            JP Z,ANDCNTINS
            CP ANDNCNT
            JP Z,ANDNCNTINS
            CP ORTIM
            JP Z,ORTIMINS
```

```
        CP ORNTIM
        JP Z,ORNTIMINS
        CP ORCNT
        JP Z,ORCNTINS
        CP ORNCNT
        JP Z,ORNCNTINS
        CP _NOP          ;{FUN Command bit 7 is Set}
        JP Z,NOPINS
        CP END
        JP Z,ENDINS
        CP IL
        JP Z,ILINS
        CP ILC
        JP Z,ILCINS
        CP JMP
        JP Z,JMPINS
        CP JME
        JP Z,JMEINS
        CP SFT
        JP Z,SFTINS
        CP KEEP
        JP Z,KEEPINS
        CP CNTR
        JP Z,CNTRINS
        CP DIFU
        JP Z,DIFUINS
        CP DIFD
        JP Z,DIFDINS
        CP TIMH
        JP Z,TIMHINS
        CP WSFT
        JP Z,WSFTINS
        CP CMP
        JP Z,CMPINS
        CP MOV
        JP Z,MOVINS
        CP MVN
        JP Z,MVNINS
        CP BIN
        JP Z,BININS
        CP BCD
        JP Z,BCDINS
        CP ASL
        JP Z,ASLINS
        CP ASR
        JP Z,ASRINS
        CP ROL
        JP Z,ROLINS
        CP ROR
        JP Z,RORINS
        CP COM
        JP Z,COMINS
        CP ADD
        JP Z,ADDINS
```

```
        CP SUB
        JP Z,SUBINS
        CP MUL
        JP Z,MULINS
        CP DIV
        JP Z,DIVINS
        CP ANDW
        JP Z,ANDWINS
        CP ORW
        JP Z,ORWINS
        CP XORW
        JP Z,XORWINS
        CP XNRW
        JP Z,XNRWINS
        CP INC
        JP Z,INCINS
        CP DEC
        JP Z,DECINS
        CP STC
        JP Z,STCINS
        CP CLC
        JP Z,CLCINS
        CP FUN70
        JP Z,FUN70INS
        CP FUN71
        JP Z,FUN71INS
        CP FUN72
        JP Z,FUN72INS
        CP FUN73
        JP Z,FUN73INS
        CP FUN74
        JP Z,FUN74INS
        CP FUN75
        JP Z,FUN75INS
        CP FUN76
        JP Z,FUN76INS
        CP FUN77
        JP Z,FUN77INS
        CP FUN78
        JP Z,FUN78INS
        CP FUN79
        JP Z,FUN79INS
        CP FUN80
        JP Z,FUN80INS
        CP FUN81
        JP Z,FUN81INS
        CP FUN82
        JP Z,FUN82INS
        CP FUN83
        JP Z,FUN83INS
        CP FUN84
        JP Z,FUN84INS
        CP FUN85
        JP Z,FUN85INS
```

```
                CP FUN94
                JP Z,FUN94INS
                CP FUN95
                JP Z,FUN95INS

ILLEGAL:        LD DE,(EDITMEM)
                INC DE
                LD (EDITMEM),DE
                RET

LDINS:          LD BC,LDADD
                LD A,01
                CALL PUTCODE
                RET

LDNINS:         LD BC,LDNADD
                LD A,01
                CALL PUTCODE
                RET

ANDINS:         LD BC,ANDADD
                LD A,01
                CALL PUTCODE
                RET

ANDNINS:        LD BC,ANDNADD
                LD A,01
                CALL PUTCODE
                RET

ORINS:          LD BC,ORADD
                LD A,01
                CALL PUTCODE
                RET

ORNINS:         LD BC,ORNADD
                LD A,01
                CALL PUTCODE
                RET

ANDLDINS:       LD BC,ANDLDADD
                LD A,01
                CALL PUTCODE
                RET

ORLDINS:        LD BC,ORLDADD
                LD A,01
                CALL PUTCODE
                RET

OUTINS:         LD BC,OUTADD
                LD A,01
                CALL PUTCODE
                RET
```

```
OUTNINS:    LD BC,OUTNADD
            LD A,01
            CALL PUTCODE
            RET

TIMINS:     LD BC,TIMADD
            LD A,02
            CALL PUTCODE
            RET

CNTINS:     LD BC,CNTADD
            LD A,02
            CALL PUTCODE
            RET

LDTIMINS:   LD BC,LDTIMADD
            LD A,01
            CALL PUTCODE
            RET

LDNTIMINS:  LD BC,LDNTIMADD
            LD A,01
            CALL PUTCODE
            RET

LDCNTINS:   LD BC,LDCNTADD
            LD A,01
            CALL PUTCODE
            RET

LDNCNTINS:  LD BC,LDNCNTADD
            LD A,01
            CALL PUTCODE
            RET

ANDTIMINS:  LD BC,ANDTIMADD
            LD A,01
            CALL PUTCODE
            RET

ANDNTIMINS: LD BC,ANDNTIMADD
            LD A,01
            CALL PUTCODE
            RET

ANDCNTINS:  LD BC,ANDCNTADD
            LD A,01
            CALL PUTCODE
            RET

ANDNCNTINS: LD BC,ANDNCNTADD
            LD A,01
            CALL PUTCODE
```

```
                RET

ORTIMINS:       LD BC,ORTIMADD
                LD A,01
                CALL PUTCODE
                RET

ORNTIMINS:      LD BC,ORNTIMADD
                LD A,01
                CALL PUTCODE
                RET

ORCNTINS:       LD BC,ORCNTADD
                LD A,01
                CALL PUTCODE
                RET

ORNCNTINS:      LD BC,ORNCNTADD
                LD A,01
                CALL PUTCODE
                RET

NOPINS:         LD BC,NOPADD
                LD A,00
                CALL PUTCODE
                RET

ENDINS:         LD BC,ENDADD
                LD A,00
                CALL PUTCODE
                RET

ILINS:          LD BC,ILADD
                LD A,00
                CALL PUTCODE
                RET

ILCINS:         LD BC,ILCADD
                LD A,00
                CALL PUTCODE
                RET

JMPINS:         LD BC,JMPADD
                LD A,00
                CALL PUTCODE
                RET

JMEINS:         LD BC,JMEADD
                LD A,00
                CALL PUTCODE
                RET

SFTINS:         LD BC,SFTADD
                LD A,02
```

```
                CALL PUTCODE
                RET

KEEPINS:        LD BC,KEEPADD
                LD A,01
                CALL PUTCODE
                RET

CNTRINS:        LD BC,CNTRADD
                LD A,01
                CALL PUTCODE
                RET

DIFUINS:        LD BC,DIFUADD
                LD A,01
                CALL PUTCODE
                RET

DIFDINS:        LD BC,DIFDADD
                LD A,01
                CALL PUTCODE
                RET

TIMHINS:        LD BC,TIMHADD
                LD A,02
                CALL PUTCODE
                RET

WSFTINS:        LD BC,WSFTADD
                LD A,02
                CALL PUTCODE
                RET

CMPINS:         LD BC,CMPADD
                LD A,02
                CALL PUTCODE
                RET

MOVINS:         LD BC,MOVADD
                LD A,02
                CALL PUTCODE
                RET

MVNINS:         LD BC,MVNADD
                LD A,02
                CALL PUTCODE
                RET

BININS:         LD BC,BINADD
                LD A,02
                CALL PUTCODE
                RET

BCDINS:         LD BC,BCDADD
```

```
                LD A,02
                CALL PUTCODE
                RET

ASLINS:         LD BC,ASLADD
                LD A,01
                CALL PUTCODE
                RET

ASRINS:         LD BC,ASRADD
                LD A,01
                CALL PUTCODE
                RET

ROLINS:         LD BC,ROLADD
                LD A,01
                CALL PUTCODE
                RET

RORINS:         LD BC,RORADD
                LD A,01
                CALL PUTCODE
                RET

COMINS:         LD BC,COMADD
                LD A,01
                CALL PUTCODE
                RET

ADDINS:         LD BC,ADDADD
                LD A,03
                CALL PUTCODE
                RET

SUBINS:         LD BC,SUBADD
                LD A,03
                CALL PUTCODE
                RET

MULINS:         LD BC,MULADD
                LD A,03
                CALL PUTCODE
                RET

DIVINS:         LD BC,DIVADD
                LD A,03
                CALL PUTCODE
                RET

ANDWINS:        LD BC,ANDWADD
                LD A,03
                CALL PUTCODE
                RET
```

```
ORWINS:     LD BC,ORWADD
            LD A,03
            CALL PUTCODE
            RET

XORWINS:    LD BC,XORWADD
            LD A,03
            CALL PUTCODE
            RET

XNRWINS:    LD BC,XNRWADD
            LD A,03
            CALL PUTCODE
            RET

INCINS:     LD BC,INCADD
            LD A,01
            CALL PUTCODE
            RET

DECINS:     LD BC,DECADD
            LD A,01
            CALL PUTCODE
            RET

STCINS:     LD BC,STCADD
            LD A,00
            CALL PUTCODE
            RET

CLCINS:     LD BC,CLCADD
            LD A,00
            CALL PUTCODE
            RET

FUN70INS:   LD BC,FUN70ADD
            LD A,03
            CALL PUTCODE
            RET

FUN71INS:   LD BC,FUN71ADD
            LD A,03
            CALL PUTCODE
            RET

FUN72INS:   LD BC,FUN72ADD
            LD A,03
            CALL PUTCODE
            RET

FUN73INS:   LD BC,FUN73ADD
            LD A,03
            CALL PUTCODE
            RET
```

```
FUN74INS:       LD BC,FUN74ADD
                LD A,03
                CALL PUTCODE
                RET

FUN75INS:       LD BC,FUN75ADD
                LD A,03
                CALL PUTCODE
                RET

FUN76INS:       LD BC,FUN76ADD
                LD A,03
                CALL PUTCODE
                RET

FUN77INS:       LD BC,FUN77ADD
                LD A,03
                CALL PUTCODE
                RET

FUN78INS:       LD BC,FUN78ADD
                LD A,03
                CALL PUTCODE
                RET

FUN79INS:       LD BC,FUN79ADD
                LD A,03
                CALL PUTCODE
                RET

FUN80INS:       LD BC,FUN80ADD
                LD A,03
                CALL PUTCODE
                RET

FUN81INS:       LD BC,FUN81ADD
                LD A,03
                CALL PUTCODE
                RET

FUN82INS:       LD BC,FUN82ADD
                LD A,03
                CALL PUTCODE
                RET

FUN83INS:       LD BC,FUN83ADD
                LD A,03
                CALL PUTCODE
```

```
                RET


FUN84INS:       LD BC,FUN84ADD
                LD A,03
                CALL PUTCODE
                RET


FUN85INS:       LD BC,FUN85ADD
                LD A,03
                CALL PUTCODE
                RET


FUN94INS:       LD BC,FUN94ADD
                LD A,03
                CALL PUTCODE
                RET


FUN95INS:       LD BC,FUN95ADD
                LD A,03
                CALL PUTCODE
                RET
;_____________________________________________________________
LDADD:          EX DE,HL        ;SAVE I/O ASSIGN HL
                EX (SP),IY
                POP IX
                LD HL,IOCH0
                LD A,D
                ADD A,A         ;CAL OFFSET MUL 2
                LD C,A
                LD B,00H
                ADD HL,BC
                LD A,E          ;BIT IN CH
                CP 07H
                JP M,LDLSB
                JR Z,LDLSB      ;IF EQUAL LESS 7 THEN LOW BYTE
LDMSB:          INC HL
                SUB 08H
LDLSB:          EX DE,HL        ;DE POINT IOCH ACTIVE
                LD HL,MASKDTA
                LD B,00H
                LD C,A
                ADD HL,BC
                LD A,(HL)
                EX DE,HL
                AND (HL)
                JR Z,RESET
SET:            LD A,0FFH
                PUSH AF         ;PUSH RESULT
                PUSH IY         ;RET ADDR
                RET
```

```
RESET:      LD A,00H
            PUSH AF
            PUSH IY
            RET

MASKDTA:    DEFB 01H
            DEFB 02H
            DEFB 04H
            DEFB 08H
            DEFB 10H
            DEFB 20H
            DEFB 40H
            DEFB 80H
;___________________________________
LDNADD:     EX (SP),IY
            POP IX
            LD (RETADDR),IY
            CALL LDADD
            POP AF
            CPL
            PUSH AF
            LD IY,(RETADDR)
            PUSH IY
            RET
;___________________________________
ANDADD:     EX (SP),IY
            POP IX
            LD (RETADDR),IY
            CALL LDADD
            POP AF
            POP BC
            AND B
            PUSH AF
            LD IY,(RETADDR)
            PUSH IY
            RET
;___________________________________
ANDNADD:    EX (SP),IY
            POP IX
            LD (RETADDR1),IY
            CALL LDNADD
            POP AF
            POP BC
            AND B
            PUSH AF
            LD IY,(RETADDR1)
            PUSH IY
            RET
;___________________________________
ORADD:      EX (SP),IY
            POP IX
            LD (RETADDR),IY
            CALL LDADD
```

```
                POP AF
                POP BC
                OR B
                PUSH AF
                LD IY,(RETADDR)
                PUSH IY
                RET
;_______________________________________
ORNADD:         EX (SP),IY
                POP IX
                LD (RETADDR1),IY
                CALL LDNADD
                POP AF
                POP BC
                OR B
                PUSH AF
                LD IY,(RETADDR1)
                PUSH IY
                RET
;_______________________________________
ANDLDADD:       EX (SP),IY
                POP IX
                POP AF
                POP BC
                AND B
                PUSH AF
                PUSH IY
                RET
;_______________________________________
ORLDADD:        EX (SP),IY
                POP IX
                POP AF
                POP BC
                OR B
                PUSH AF
                PUSH IY
                RET
;_______________________________________
OUTADD:         EX (SP),IY
                POP IX
                EX DE,HL    ;IOCH IN DE
                POP AF
                PUSH AF     ;STATUS BEFORE OUTINST.
                AND OFFH
                JP Z,OUTO
OUT1:           LD HL,IOCHO
                LD A,D
                ADD A,A
                LD C,A
                LD B,00H
                ADD HL,BC
                LD A,E      ;BIT IN CH
                CP 07H
                JR Z,OUTLSB
```

```
                JP M,OUTLSB
OUTMSB:         SUB 08H
                INC HL
OUTLSB:         EX DE,HL
                LD HL,MASKDTA
                LD C,A
                ADD HL,BC
                LD A,(HL)
                EX DE,HL
                OR (HL)
                LD (HL),A
                PUSH IY
                RET

OUTO:           LD HL,IOCHO
                LD A,D
                ADD A,A
                LD C,A
                LD B,00H
                ADD HL,BC
                LD A,E          ;BIT IN CH
                CP 07H
                JR Z,OUTLSBO
                JP M,OUTLSBO

OUTMSBO:        SUB 08H
                INC HL
OUTLSBO:        EX DE,HL
                LD HL,MASKDTA
                LD C,A
                ADD HL,BC
                LD A,(HL)
                CPL
                EX DE,HL
                AND (HL)
                LD (HL),A
                PUSH IY
                RET
;_______________________________________
OUTNADD:        EX (SP),IY
                POP IX
                EX DE,HL        ;IOCH IN DE
                POP AF
                AND OFFH
                JP Z,OUT1       ;IF 0 THEN OUT 1
                JP NZ,OUTO      ;IF 1 THEN OUT 0
;_______________________________________
TIMADD:         EX (SP),IY
                POP IX
                LD (RETADDR),IY
                POP AF
                AND OFFH
                JP Z,TIMRST             ;SET TIME START
                LD IX,TIMCHO
```

```
                LD C,L
                LD A,C
                ADD A,A
                LD C,A
                LD B,00H
                ADD IX,BC
                LD A,(IX+00)
                LD B,(IX+01)
                OR B
                JR Z,TIMSET
                LD A,80H            ;START TIME
                OR (IX+01)
                LD (IX+01),A
EXITIM:         LD IY,(RETADDR)
                PUSH IY
                RET
TIMSET:         LD IX,TIMSTAO
                EX DE,HL
                ADD IX,DE
                EX DE,HL
                LD (IX+00),0FFH
                JR EXITIM
TIMRST:         LD A,L
                LD H,00
                LD IX,TIMSTAO
                EX DE,HL
                ADD IX,DE
                EX DE,HL
                LD (IX+00),00
                ADD A,A
                LD L,A
                LD IX,TIMCHO
                EX DE,HL
                ADD IX,DE
                EX DE,HL
                LD (IX+00),E  ;TIMORD IN DE REG.
                LD (IX+01),D
                JR EXITIM
;_____________________________________________
CNTORDER:       LD (IY+00),0FFH
                LD IY,CNTCHO
                LD C,L
                LD A,C
                ADD A,A
                LD C,A
                LD B,00H
                ADD IX,BC
                LD (IX+00),E
                LD (IX+01),D
                RET

CNTADD:         EX (SP),IY    ;HL [POINT CH].....DE [#DATA]
                POP IX
                LD (RETADDR),IY
```

```
                LD IY,CORDSTAO
                EX DE,HL
                ADD IX,DE
                EX DE,HL
                LD A,(IX+00)
                AND 0FFH
                CALL Z,CNTORDER
                POP AF          ;RESET COUNTER SIGNAL
                AND 0FFH
                JR NZ,RSCNT
                POP AF          ;COUNT SIGNAL
                LD IX,LASTCNTO
                EX DE,HL
                ADD IX,DE
                EX DE,HL
                LD B,(IX+00)
                LD (IX+00),A
                XOR B
                EX AF,AF'
                LD A,B
                CPL
                LD B,A
                EX AF,AF'
                AND B
                CP 00H
                JR Z,EXITCNT
                LD IX,CNTSTAO
                EX DE,HL
                ADD IX,DE
                EX DE,HL
                LD A,(IX+00)
                CP 00H
                JR NZ,EXITCNT   ;IF COUNT = ZERO THEN NOT COUNT
COUNT:          LD IX,CNTCHO
                LD C,L
                LD A,C
                ADD A,A
                LD C,A
                LD B,00H
                ADD IX,BC
                LD C,(IX+00)
                LD B,(IX+01)
                DEC BC          ;COUNT DOWN
                LD (IX+00),C
                LD (IX+01),B
                LD A,B
                OR C
                JR Z,SCNT       ;SET STATUS COUNT 'ON'
EXITCNT:        LD IY,(RETADDR)
                PUSH IY
                RET
SCNT:           LD IX,CNTSTAO
                EX DE,HL
```

```
                ADD IX,DE
                EX DE,HL
                LD (IX+00),0FFH
                JR EXITCNT

RSCNT:          LD IX,CNTSTA0
                EX DE,HL
                ADD IX,DE
                LD (IX+00),00H          ;RESET STATUS
                LD IX,CNTCH0
                LD A,E
                ADD A,A
                LD E,A
                ADD IX,DE
                EX DE,HL
                LD (IX+00),E            ;UP DATE DATA IN DE
                LD (IX+01),D
                JR EXITCNT
;_____________________________________
LDTIMADD:       EX (SP),IY
                POP IX
                EX DE,HL
                LD HL,TIMSTA0
                ADD HL,DE
                LD A,(HL)
                PUSH AF
                PUSH IY
                RET
;_____________________________________
LDNTIMADD:      EX (SP),IY
                POP IX
                LD (RETADDR),IY
                CALL LDTIMADD
                POP AF
                CPL
                PUSH AF
                LD IY,(RETADDR)
                PUSH IY
                RET
;_____________________________________
LDCNTADD:       EX (SP),IY
                POP IX
                EX DE,HL
                LD HL,CNTSTA0
                ADD HL,DE
                LD A,(HL)
                PUSH AF
                PUSH IY
                RET
;_____________________________________
LDNCNTADD:      EX (SP),IY
                POP IX
                LD (RETADDR),IY
                CALL LDCNTADD
```

```
                POP AF
                CPL
                PUSH AF
                LD IY,(RETADDR)
                PUSH IY
                RET
;_______________________________________
ANDTIMADD:      EX (SP),IY
                POP IX
                LD (RETADDR),IY
                CALL LDTIMADD
                POP AF
                POP BC
                AND B
                PUSH AF
                LD IY,(RETADDR)
                PUSH IY
                RET
;_______________________________________
ANDNTIMADD:     EX (SP),IY
                POP IX
                LD (RETADDR1),IY
                CALL LDNTIMADD
                POP AF
                POP BC
                AND B
                PUSH AF
                LD IY,(RETADDR1)
                PUSH IY
                RET
;_______________________________________
ANDCNTADD:      EX (SP),IY
                POP IX
                LD (RETADDR),IY
                CALL LDCNTADD
                POP AF
                POP BC
                AND B
                PUSH AF
                LD IY,(RETADDR)
                PUSH IY
                RET
;_______________________________________
ANDNCNTADD:     EX (SP),IY
                POP IX
                LD (RETADDR1),IY
                CALL LDNCNTADD
                POP AF
                POP BC
                AND B
                PUSH AF
                LD IY,(RETADDR1)
                PUSH IY
                RET
```

```
;_____________________________________
ORTIMADD:       EX (SP),IY
                POP IX
                LD (RETADDR),IY
                CALL LDTIMADD
                POP AF
                POP BC
                OR B
                PUSH AF
                LD IY,(RETADDR)
                PUSH IY
                RET
;_____________________________________
ORNTIMADD:      EX (SP),IY
                POP IX
                LD (RETADDR1),IY
                CALL LDNTIMADD
                POP AF
                POP BC
                OR B
                PUSH AF
                LD IY,(RETADDR1)
                PUSH IY
                RET
;_____________________________________
ORCNTADD:       EX (SP),IY
                POP IX
                LD (RETADDR),IY
                CALL LDCNTADD
                POP AF
                POP BC
                OR B
                PUSH AF
                LD IY,(RETADDR)
                PUSH IY
                RET
;_____________________________________
ORNCNTADD:      EX (SP),IY
                POP IX
                LD (RETADDR1),IY
                CALL LDNCNTADD
                POP AF
                POP BC
                OR B
                PUSH AF
                LD IY,(RETADDR1)
                PUSH IY
                RET
;_____________________________________
NOPADD:         RET
ENDADD:         RET
;_____________________________________
ILADD:          RET
;_____________________________________
```

```
ILCADD:         RET
;_____________________________________
JMPADD:         NOP
JMEADD:         NOP
;_____________________________________
SFTADD:         RET
;_____________________________________
KEEPADD:        EX (SP),IY
                POP IX
                EX DE,HL    ;IOCH IN DE
                POP AF
                AND OFFH
                JR NZ,RSTKEEP
                POP AF
                AND OFFH
                JR Z,RETKEEP

SETKEEP:        LD HL,IOCHO
                LD A,D
                ADD A,A
                LD C,A
                LD B,OOH
                ADD HL,BC
                LD A,E      ;BIT IN CH
                CP O7H
                JR Z,KEEPLSB
                JP M,KEEPLSB
KEEPMSB:        SUB O8H
                INC HL
KEEPLSB:        EX DE,HL
                LD HL,MASKDTA
                LD C,A
                ADD HL,BC
                LD A,(HL)
                EX DE,HL
                OR (HL)
                LD (HL),A
RETKEEP:        PUSH IY
                RET

RSTKEEP:        POP AF              ;POP KEEP SET
                LD HL,IOCHO
                LD A,D
                ADD A,A
                LD C,A
                LD B,OOH
                ADD HL,BC
                LD A,E      ;BIT IN CH
                CP O7H
                JR Z,KEEPLSBO
                JP M,KEEPLSBO
KEEPMSBO:       SUB O8H
                INC HL
KEEPLSBO:       EX DE,HL
```

```
                LD HL,MASKDTA
                LD C,A
                ADD HL,BC
                LD A,(HL)
                CPL
                EX DE,HL
                AND (HL)
                LD (HL),A
                PUSH IY
                RET
;_____________________________________
CNTRADD:        RET
;_____________________________________
DIFUADD:        EX (SP),IY
                POP IX
                EX DE,HL            ;CH. DIFFUP
                LD A,(LSTSTAPOI)
                LD HL,LSTSTATEO
                LD C,A
                INC A
                LD (LSTSTAPOI),A    ;INC POINTER
                LD B,00
                ADD HL,BC
                LD B,(HL)           ;OLD STATUS FROM MEM
                POP AF              ;PRESENT STATE
                PUSH AF             ;LAST RESULT
                LD (HL),A           ;NEW STATUS TO MEM
                XOR B
                EX AF,AF'
                LD A,B
                CPL
                LD B,A
                EX AF,AF'
                AND B
                CP 00H
                JR Z,RETDIFU        ;IF Z NOT DIFFERENTIATION UP

DIFFUP:         LD A,(DIFPOI)       ;DIFF POINTER
                LD HL,DIFSTA        ;DIFF STATUS
                LD C,A
                INC A
                LD (DIFPOI),A
                LD B,00
                ADD HL,BC
                LD A,(HL)
                CP 00H
                JR Z,SETDIFF
                JR RETDIFU

RETDIFU:        LD HL,IOCH0         ;OFF AFTER 1 SCANTIME LATER
                LD A,D
                ADD A,A
                LD C,A
                LD B,00H
```

```
                ADD HL,BC
                LD A,E      ;BIT IN CH
                CP 07H
                JR Z,DIFLSB0
                JP M,DIFLSB0
DIFMSB0:        SUB 08H
                INC HL
DIFLSB0:        EX DE,HL
                LD HL,MASKDTA
                LD C,A
                ADD HL,BC
                LD A,(HL)
                CPL
                EX DE,HL
                AND (HL)
                LD (HL),A
                PUSH IY
                RET

SETDIFF:        LD (HL),00     ;RST DIFF STA
                LD HL,IOCH0
                LD A,D
                ADD A,A
                LD C,A
                LD B,00H
                ADD HL,BC
                LD A,E      ;BIT IN CH
                CP 07H
                JR Z,DIFLSB1
                JP M,DIFLSB1
DIFMSB1:        SUB 08H
                INC HL
DIFLSB1:        EX DE,HL
                LD HL,MASKDTA
                LD C,A
                ADD HL,BC
                LD A,(HL)
                EX DE,HL
                OR (HL)
                LD (HL),A
                PUSH IY
                RET
;_____________________________________
DIFDADD:        EX (SP),IY
                POP IX
                EX DE,HL          ;CH. DIFFUP
                LD A,(LSTSTAPOI)
                LD HL,LSTSTATE0
                LD C,A
                INC A
                LD (LSTSTAPOI),A  ;INC POINTER
                LD B,00
                ADD HL,BC
                LD B,(HL)         ;OLD STATUS FROM MEM
```

```
                POP AF              ;PRESENT STATE
                PUSH AF             ;LAST RESULT
                LD (HL),A           ;NEW STATUS TO MEM
                XOR B
                LD B,A
                LD A,(HL)
                CPL
                AND B
                CP 00H
                JP Z,RETDIFU        ;IF Z NOT DIFFERENTIATION DOWN
                JP DIFFUP
;_____________________________________
TIMHADD:        RET
WSFTADD:        RET
CMPADD:         RET
;_____________________________________
MOVADD:         EX (SP),IY
                POP IX
                POP AF
                PUSH AF
                CP 0FFH
                JR NZ,RETMOV
                LD A,L              ;USE LOWBYTE CH00XX
                ADD A,A
                LD C,A
                LD B,00H
                LD HL,IOCH0
                ADD HL,BC
                EX DE,HL
                LD A,L      ;USE LOWBYTE CH00XX
                ADD A,A
                LD C,A
                LD B,00H
                LD HL,IOCH0
                ADD HL,BC
                EX DE,HL
                LDI
                LDI
RETMOV:         PUSH IY
                RET
;_____________________________________
MVNADD:         EX (SP),IY
                POP IX
                POP AF
                PUSH AF
                CP 0FFH
                JR NZ,RETMOVN
                LD A,L      ;USE LOWBYTE CH00XX
                ADD A,A
                LD C,A
                LD B,00H
                LD HL,IOCH0
                ADD HL,BC
                EX DE,HL
```

```
                LD A,L      ;USE LOWBYTE CHOOXX
                ADD A,A
                LD C,A
                LD B,OOH
                LD HL,IOCHO
                ADD HL,BC
                EX DE,HL
                LD A,(HL)
                CPL
                LD (DE),A
                INC HL
                INC DE
                LD A,(HL)
                CPL
                LD (DE),A
RETMOVN:        PUSH IY
                RET
;_____________________________________
BINADD:         NOP
BCDADD:         NOP
ASLADD:         NOP
ASRADD:         NOP
ROLADD:         NOP
RORADD:         NOP
;_____________________________________
COMADD:         EX (SP),IY
                POP IX
                POP AF
                PUSH AF
                CP OFFH
                JR NZ,RETCOM
                LD A,L
                ADD A,A
                LD HL,IOCHO
                LD C,A
                LD B,OO
                ADD HL,BC
                LD A,(HL)
                CPL
                LD (HL),A
                INC HL
                LD A,(HL)
                CPL
                LD (HL),A
RETCOM:         PUSH IY
                RET
;_____________________________________
ADDADD:         EX (SP),IY
                POP IX
                POP AF
                PUSH AF
                LD (RETADDR),SP
                CP OFFH
                JR NZ,RETADD
```

```
                PUSH BC     ;PUSH RESULT ADDR
                LD A,L
                ADD A,A
                LD C,A
                LD B,00
                LD HL,IOCH0
                ADD HL,BC
                EX DE,HL
                LD A,L
                ADD A,A
                LD C,A
                LD B,00
                LD HL,IOCH0
                ADD HL,BC
                EX DE,HL
                POP BC      ;POP RESULT ADDR
                LD IX,IOCH0
                LD A,C
                ADD A,A
                LD C,A
                LD B,00
                ADD IX,BC
                LD C,(HL)
                INC HL
                LD B,(HL)
                PUSH BC
                EX DE,HL
                LD C,(HL)
                INC HL
                LD B,(HL)
                POP HL
                ADD HL,BC
                LD SP,IX
                EX (SP),HL
                LD SP,(RETADDR)
RETADD:         PUSH IY
                RET
;_____________________________________
SUBADD:         EX (SP),IY
                POP IX
                POP AF
                PUSH AF
                LD (RETADDR),SP
                CP 0FFH
                JR NZ,RETSUB
                PUSH BC     ;PUSH RESULT ADDR
                LD A,L
                ADD A,A
                LD C,A
                LD B,00
                LD HL,IOCH0
                ADD HL,BC
                EX DE,HL
                LD A,L
```

```
                ADD A,A
                LD C,A
                LD B,00
                LD HL,IOCH0
                ADD HL,BC
                EX DE,HL
                POP BC      ;POP RESULT ADDR
                LD IX,IOCH0
                LD A,C
                ADD A,A
                LD C,A
                LD B,00
                ADD IX,BC
                LD C,(HL)
                INC HL
                LD B,(HL)
                PUSH BC
                EX DE,HL
                LD C,(HL)
                INC HL
                LD B,(HL)
                POP HL
                ADD HL,BC
                LD SP,IX
                EX (SP),HL
                LD SP,(RETADDR)
RETSUB:         PUSH IY
                RET
;_______________________________________

MULADD:         NOP
DIVADD:         NOP
;_______________________________________
ANDWADD:        EX (SP),IY
                POP IX
                POP AF
                PUSH AF
                CP 0FFH
                JR NZ,RETAND
                PUSH BC     ;PUSH RESULT ADDR
                LD A,L
                ADD A,A
                LD C,A
                LD B,00
                LD HL,IOCH0
                ADD HL,BC
                EX DE,HL
                LD A,L
                ADD A,A
                LD C,A
                LD B,00
                LD HL,IOCH0
                ADD HL,BC
                EX DE,HL
```

```
                POP BC      ;POP RESULT ADDR
                LD IX,IOCHO
                LD A,C
                ADD A,A
                LD C,A
                LD B,00
                ADD IX,BC
                LD A,(HL)
                LD B,A
                LD A,(DE)
                AND B
                LD (IX+00),A
                INC HL
                INC DE
                INC IX
                LD A,(HL)
                LD B,A
                LD A,(DE)
                AND B
                LD (IX+00),A
RETAND:         PUSH IY
                RET
;_____________________________________
ORWADD:         EX (SP),IY
                POP IX
                POP AF
                PUSH AF
                CP 0FFH
                JR NZ,RETOR
                PUSH BC     ;PUSH RESULT ADDR
                LD A,L
                ADD A,A
                LD C,A
                LD B,00
                LD HL,IOCHO
                ADD HL,BC
                EX DE,HL
                LD A,L
                ADD A,A
                LD C,A
                LD B,00
                LD HL,IOCHO
                ADD HL,BC
                EX DE,HL
                POP BC      ;POP RESULT ADDR
                LD IX,IOCHO
                LD A,C
                ADD A,A
                LD C,A
                LD B,00
                ADD IX,BC
                LD A,(HL)
                LD B,A
                LD A,(DE)
```

```
                OR B
                LD (IX+00),A
                INC HL
                INC DE
                INC IX
                LD A,(HL)
                LD B,A
                LD A,(DE)
                OR B
                LD (IX+00),A
RETOR:          PUSH IY
                RET
;_____________________________________
XORWADD:        EX (SP),IY
                POP IX
                POP AF
                PUSH AF
                CP 0FFH
                JR NZ,RETXOR
                PUSH BC     ;PUSH RESULT ADDR
                LD A,L
                ADD A,A
                LD C,A
                LD B,00
                LD HL,IOCH0
                ADD HL,BC
                EX DE,HL
                LD A,L
                ADD A,A
                LD C,A
                LD B,00
                LD HL,IOCH0
                ADD HL,BC
                EX DE,HL
                POP BC      ;POP RESULT ADDR
                LD IX,IOCH0
                LD A,C
                ADD A,A
                LD C,A
                LD B,00
                ADD IX,BC
                LD A,(HL)
                LD B,A
                LD A,(DE)
                XOR B
                LD (IX+00),A
                INC HL
                INC DE
                INC IX
                LD A,(HL)
                LD B,A
                LD A,(DE)
                XOR B
                LD (IX+00),A
```

```
RETXOR:     PUSH IY
            RET
;_____________________________________
XNRWADD:    EX (SP),IY
            POP IX
            POP AF
            PUSH AF
            CP OFFH
            JR NZ,RETXORN
            PUSH BC     ;PUSH RESULT ADDR
            LD A,L
            ADD A,A
            LD C,A
            LD B,00
            LD HL,IOCH0
            ADD HL,BC
            EX DE,HL
            LD A,L
            ADD A,A
            LD C,A
            LD B,00
            LD HL,IOCH0
            ADD HL,BC
            EX DE,HL
            POP BC      ;POP RESULT ADDR
            LD IX,IOCH0
            LD A,C
            ADD A,A
            LD C,A
            LD B,00
            ADD IX,BC
            LD A,(HL)
            LD B,A
            LD A,(DE)
            XOR B
            CPL
            LD (IX+00),A
            INC HL
            INC DE
            INC IX
            LD A,(HL)
            LD B,A
            LD A,(DE)
            XOR B
            CPL
            LD (IX+00),A
RETXORN:    PUSH IY
            RET
;_____________________________________
INCADD:     EX (SP),IY
            POP IX
            POP AF
            PUSH AF
            LD (RETADDR),SP
```

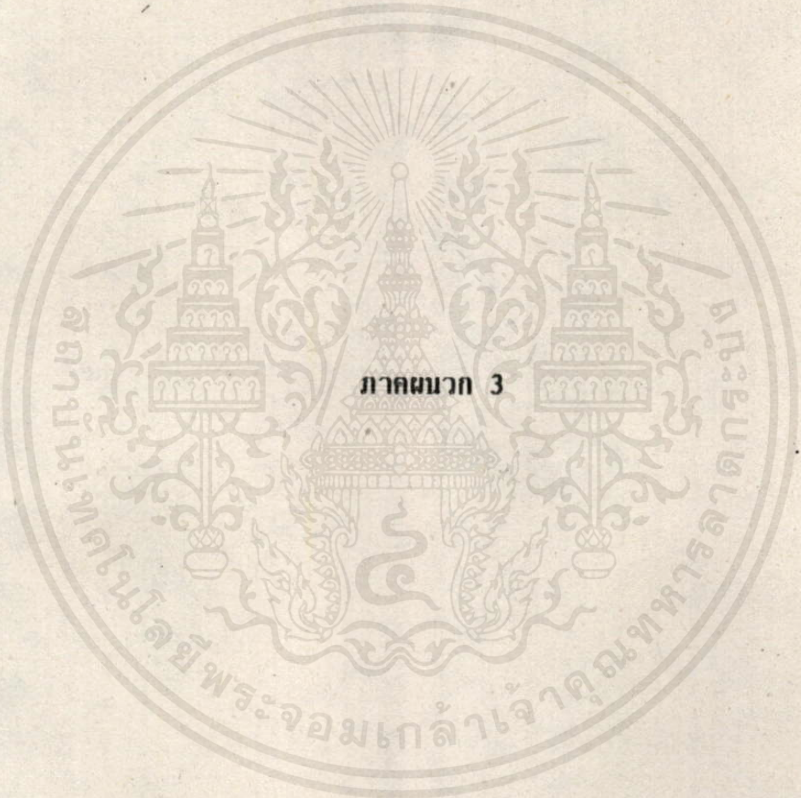
```
                CP OFFH
                JR NZ,RETINC
                LD A,L          ;USE LOW BYTE OOXX [CHXX]
                ADD A,A
                LD C,A
                LD B,OO
                LD HL,IOCHO
                ADD HL,BC
                LD SP,HL
                EX (SP),HL
                INC HL
                EX (SP),HL
                LD SP,(RETADDR)
RETINC:         PUSH IY
                RET
;_____________________________________
DECADD:         EX (SP),IY
                POP IX
                POP AF
                PUSH AF
                LD (RETADDR),SP
                CP OFFH
                JR NZ,RETDEC
                LD A,L
                ADD A,A
                LD C,A
                LD B,OO
                LD HL,IOCHO
                ADD HL,BC
                LD SP,HL
                EX (SP),HL
                DEC HL
                EX (SP),HL
                LD SP,(RETADDR)
RETDEC:         PUSH IY
                RET
;_____________________________________
STCADD:         NOP
CLCADD:         NOP
FUN70ADD:       NOP
FUN71ADD:       NOP
FUN72ADD:       NOP
FUN73ADD:       NOP
FUN74ADD:       NOP
FUN75ADD:       NOP
FUN76ADD:       NOP
FUN77ADD:       NOP
FUN78ADD:       NOP
FUN79ADD:       NOP
FUN80ADD:       NOP
FUN81ADD:       NOP
FUN82ADD:       NOP
FUN83ADD:       NOP
FUN84ADD:       NOP
```

```
FUN85ADD:       NOP
FUN94ADD:       NOP
FUN95ADD:       NOP
                RET  ;PROTECT
                END
```

File : D:\X80\PCMAIN.ASM
Current Date : Monday November 7, 1988
Current Time : 2:26 PM

Program : PCMAIN
Programmer : Viriya Kongratana

```
            ORG 0C000H
;           INCLUDE D:DOWNLOAD.ASM
            INCLUDE D:PCSYS.ASM
            INCLUDE D:PCSUB.ASM
            END
```



ภาคผนวก 3

Text File List Processing Program V4.00C Page : 1
File : A:PCM.PAS
Current Date : Tuesday September 6, 1988
Current Time : 2:23 PM

```
Program : Link System Main Program
Programmer : Viriya Kongratana

Program MainSystemController;

uses Crt,Dos,LadMain;


{$I D:\PASCAL4\DECLARE.PAS}
{$I D:\PASCAL4\OPENFILE.PAS}
{$I D:\PASCAL4\PCC.PAS}          {PC Communication}
{$I D:\PASCAL4\PCE.PAS}          {PC ConsoleEditor}
{$I D:\PASCAL4\PCD.PAS}          {PC Diagnostic}
{$I D:\PASCAL4\PCS.PAS}          {PC MenuSelect}
{$I D:\PASCAL4\PCMON.PAS}        {PC Monitoring}

begin
  Clrscr;
  Repeat
     MenuSelect (Choice);
     Choice := UpCase(Choice);
     Case Choice of
     'C' : InitCommunication;
     'P' : EditConSoleKey;
     'S' : Diagnostic;
     'M' : Monitoring;
     'L' : MainEditor;
     end;
  Until Choice = 'Q'
end.
```

Program : Variable Declaration
Programmer : Viriya Kongratana

```
Const   Esc             = #27;
        UP              = #72;
        DOWN            = #80;
        LEFT            = #75;
        RIGHT           = #77;
        F1              = #59;
        F2              = #60;
        F3              = #61;
        F4              = #62;
        F5              = #63;
        F6              = #64;
        F7              = #65;
        F8              = #66;
        CR              = #13;
        ON              = $FF;
        OFF             = $00;
        Maxstep         = 2000;
        MaxData         = 10000;
        _IOType         = 00;
        _DtaType        = 01;
        _LD             = 00;
        _LD_NOT         = 01;
        _AND            = 02;
        _AND_NOT        = 03;
        _OR             = 04;
        _OR_NOT         = 05;
        _AND_LD         = 06;
        _OR_LD          = 07;
        _OUT            = 08;
        _OUT_NOT        = 09;
        _TIM            = 10;
        _CNT            = 11;
        _LD_TIM         = 12;
        _LD_NOT_TIM     = 13;
        _LD_CNT         = 14;
        _LD_NOT_CNT     = 15;
        _AND_TIM        = 16;
        _AND_NOT_TIM    = 17;
        _AND_CNT        = 18;
        _AND_NOT_CNT    = 19;
        _OR_TIM         = 20;
        _OR_NOT_TIM     = 21;
        _OR_CNT         = 22;
        _OR_NOT_CNT     = 23;
                                {FUN Command bit 7 is Set}
```

```
_NOP            = 128;      {FUN 00}
_END            = 129;      {FUN 01}
_IL             = 130;
_ILC            = 131;
_JMP            = 132;
_JME            = 133;
_SFT            = 138;      {FUN 10}
_KEEP           = 139;
_CNTR           = 140;
_DIFU           = 141;
_DIFD           = 142;
_TIMH           = 143;
_WSFT           = 144;
_CMP            = 148;
_MOV            = 149;
_MVN            = 150;
_BIN            = 151;
_BCD            = 152;
_ASL            = 153;
_ASR            = 154;
_ROL            = 155;
_ROR            = 156;
_COM            = 157;
_ADD            = 158;
_SUB            = 159;
_MUL            = 160;
_DIV            = 161;
_ANDW           = 162;
_ORW            = 163;
_XORW           = 164;
_XNRW           = 165;
_INC            = 166;
_DEC            = 167;
_STC            = 168;
_CLC            = 169;
_FUN70          = 198;
_FUN71          = 199;
_FUN72          = 200;
_FUN73          = 201;
_FUN74          = 202;
_FUN75          = 203;
_FUN76          = 204;
_FUN77          = 205;
_FUN78          = 206;
_FUN79          = 207;
_FUN80          = 208;
_FUN81          = 209;
_FUN82          = 210;
_FUN83          = 211;
_FUN84          = 212;
_FUN85          = 213;
_FUN94          = 222;
_FUN95          = 227;
```

```
        _FUN            = 255;

type    RegPack         = Record case Integer of
                          1 : (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : Integer);
{! 1. Instead use the Registers type from the Turbo 4.0 DOS unit.          ^ }
                          2 : (AL,AH,BL,BH,CL,CH,DL,DH : Byte);
                          end;

        CommandPack     = Record
                          InsAddress : Integer;
                          Opcode     : Byte;
                          datatype1  : Byte;
                          Assignment1: Integer;
                          datatype2  : Byte;
                          Assignment2: Integer;
                          datatype3  : Byte;
                          Assignment3: Integer;
                          end;

        PackCommand     = Array [0..Maxstep] of CommandPack;
        String4         = String[4];
        FileType        = File of PackCommand;
        FileDest        = File of PackCommand;

Var     Instruction     : PackCommand;
        FileVar         : FileType;
        FileData        : File of Byte;
        DataTransfer    : Array [0..MaxData] of byte;
        Regs            : RegPack;
        Ch              : Char;
        I,J,K           : Integer;
        XCursor         : Byte;
        YCursor         : Byte;
        TempXCursor     : Byte;
        TempYCursor     : Byte;
        RowKey          : Byte;
        ColKey          : Byte;
        KeyCode         : Byte;
        LastCommand     : Byte;
        Attribute       : Byte;
        Cursor_Ch       : Byte;
        ClrFlag         : Byte;
        CommandFlag     : Byte;
        FunctionFlag    : Byte;
        FunctionParam   : Byte;
        DataTypeFlag    : Byte;
        HexFlag         : Byte;
        TypeData        : Byte;
        TypeData1       : Byte;
        TypeData2       : Byte;
        TypeData3       : Byte;
        ParamCount      : Byte;
        ParamFlag       : Byte;
        ParamFlag1      : Byte;
```

```
    TransferFlag      : Byte;
    FunctionDecode    : Byte;
    Transfer          : Char;
    Error             : Boolean;
    Command           : Integer;
    CommandSearch     : Integer;
    Step_Address      : Integer;
    Step_End          : Integer;
    Step_Count        : Integer;
    StepDelete        : Integer;
    StepInsert        : Integer;
    AssignIO          : Integer;
    AssignIO1         : Integer;
    AssignIO2         : Integer;
    AssignIO3         : Integer;
    AssignIOSearch    : Integer;
    Code              : Integer;
    StepTemp          : Integer;
    CommandTemp       : Integer;
    AssignIOTemp      : Integer;
    CommandLine       : String[20];
    FileName          : String[16];
    IOAssign          : String[4];
    StepString        : String4;
    AssignStr1        : String4;
    AssignStr2        : String4;
    AssignStr3        : String4;
    Address           : String4;
    FunctionString    : String[2];
    Choice            : char;
```

Program : File Handling Utility
Programmer : Viriya Kongratana

```
Function OpenReadFile (Title : string; var Filevar : FileType; FileName,Extension : string) : boolean;
var
  Buf : string;
  Ok       : boolean;
  FilenameBuffer : String[30];
  Ch : char;
  Exist    : boolean;
  ChCount : byte;
begin { OpenReadFile }
  repeat
    FilenameBuffer := '';
    Write (Title,' [',Extension,'] : ');
    readln (Filename);
    if Pos('.',Filename) = 0 then
      Filename := concat (Filename,Extension);
    for ChCount := 1 to Length(Filename) do
    begin
      Buf := Copy(Filename,ChCount,1);
      Ch := Upcase(Buf[1]);
      FilenameBuffer := FilenameBuffer + Ch;
    end;
    Filename := FilenameBuffer;
    Assign (Filevar,Filename);
    {$I-} Reset (Filevar); {$I+}
    Ok := (IOresult = 0);
    if not Ok then
    begin
      GotoXY(40,WhereY-1);
      ClrEol;
      Write (^G,'File [',Filename,'] not found');
      Ch := ReadKey;
      GotoXY(40,WhereY);
      ClrEol;
      OpenReadFile := False;
    end
    else
      OpenReadFile := True;
  until (Ok or (Ch = Esc));
end { OpenReadFile };

function OpenWriteFile (Title : string; var Filevar : FileType; FileName,Extension : String) : boolean;
var
  Buf : string;
  Ok       : boolean;
  FilenameBuffer : String[30];
```

```
  Ch : char;
  Exist    : boolean;
  ChCount : byte;
begin { OpenWriteFile }
  repeat
    FilenameBuffer := '';
    Write (Title,' [',Extension,'] : ');
    readln (Filename);
    if Pos('.',Filename) = 0 then
      Filename := concat (Filename,Extension);
    for ChCount := 1 to Length(Filename) do
    begin
      Buf := Copy(Filename,ChCount,1);
      Ch := Upcase(Buf[1]);
      FilenameBuffer := FilenameBuffer + Ch;
    end;
    Filename := FilenameBuffer;
    Assign (Filevar,Filename);
    {$I-} Reset (Filevar); {$I+}
    Ok := (IOresult = 0);
    if Ok then
    begin
      GotoXY (40,WhereY-1);
      ClrEol;
      Write (^G,'File [',Filename,'] exist');
      Ch := ReadKey;
      GotoXY(40,WhereY); ClrEol;
      if Ch = Esc then
      begin
        Write('Open file process terminated.');
        OpenWriteFile := false;
      end;
    end;
    if (not OK) or (Ch = ' ') then
    begin
      GotoXY(40,WhereY);
      if Ch = ' ' then Write('Overwriting to ',FileName);
      OpenWriteFile := true;
      ReWrite (Filevar);
    end;
  until (Ch = Esc) or (Ch = ' ') or (not OK);
end { OpenWriteFile };
```

```
Program : Communication Management
Programmer : Viriya Kongratana

Var  Baudrate_string  : String[4];
     Setformat        : String[10];
     Databit_string   : String[1];
     Stopbit_string   : String[1];
     Parity_string    : String[1];
     Result           : Integer;
     Baudrate         : Integer;
     Databit          : Integer;
     StopBit          : Integer;
     Count            : Integer;
     RxCheck          : Integer;
     TXRX             : Integer;
     Character        : Char;
     Chtr             : Char;
     Parity           : Char;
     Command_Word     : Byte;
     Bit01            : Byte;
     Bit2             : Byte;
     Bit3             : Byte;
     Bit4             : Byte;
     Bit567           : Byte;
     Data             : Byte;
     CommPort         : Byte;
     ChStr            : string;


Procedure Baudrate_value;
begin
  Chtr  := #00;
  count := 1;
  Baudrate_string := '';
  While Chtr <> ',' do
    begin
      ChStr  := copy(setformat,count,1);
      Chtr := ChStr[1];
      Baudrate_string := Concat (Baudrate_string,Chtr);
      Count := Count+1;
    end;
    Val (Baudrate_string,Baudrate,Result);
    Delete (Setformat,1,Count-1);
end;

Procedure Databit_value;
begin
  ChStr := Copy(Setformat,1,1);
```

```
  Chtr := ChStr[1];
  Val (Chtr,Databit,Result);
  Delete (Setformat,1,2);
end;

Procedure Stopbit_value;
begin
  ChStr := Copy(Setformat,1,1);
  Chtr := ChStr[1];
  Val (Chtr,Stopbit,Result);
  Delete (Setformat,1,2);
end;

Procedure Spit_value;
begin
 Baudrate_value;
 Databit_value;
 Stopbit_value;
 ChStr := Copy (Setformat,1,1);
 Chtr := ChStr[1];
 Parity := Chtr;
end;

Procedure Initial_8250;
var LSB , MSB : byte;
begin
  case BaudRate of
    300 : begin
            MSB := $01;
            LSB := $80;
          end;
    600 : begin
            MSB := $00;
            LSB := $C0;
          end;
   1200 : begin
            MSB := $00;
            LSB := $60;
          end;
   2400 : begin
            MSB := $00;
            LSB := $30;
          end;
   4800 : begin
            MSB := $00;
            LSB := $18;
          end;
   9600 : begin
            MSB := $00;
            LSB := $0C;
          end;
  end; {case}

  case CommPort of
```

```
  1: begin
       Port [$3FB] := $80;  { Buad Rate Divisor Register }
       Port [$3F8] := LSB;  { LSB Value for BuadRate }
       Port [$3F9] := MSB;  { MSB Value for BuadRate }
       TXRX        := $3F8; {TxRx Port}
       RxCheck     := $3FD; {Receive Check}
     end;

  2: begin
       Port [$2FB] := $80;  { Buad Rate Divisor Register }
       Port [$2F8] := LSB;  { LSB Value for BuadRate }
       Port [$2F9] := MSB;  { MSB Value for BuadRate }
       TXRX        := $2F8; {TxRx Port}
       RxCheck     := $2FD; {Receive Check}
     end;
  end;{case}
  case databit of
     5 : Bit01 := $00;
     6 : Bit01 := $01;
     7 : Bit01 := $02;
     8 : Bit01 := $03;
  end;

  case stopbit of
    1 : Bit2 := $00;
    2 : Bit2 := $04;
  end;

  parity := upcase(parity);
  case parity of
  'N' : Bit3 := $00;
  'O' : begin
          Bit3 := $08;
          Bit4 := $00;
        end;
  'E' : begin
          Bit3 := $08;
          Bit4 := $10;
        end;
  end;
  Bit567 := $00;
  Command_word := Bit01 OR Bit2 OR Bit3 OR Bit4 OR Bit567;
  If CommPort = 2 then Port [$2FB] := Command_word else
  Port [$3FB] := Command_Word;
end;

Procedure Disable_Interrupt;
begin
  If CommPort = 2 then Port [$2F9] := $00 else Port[$3F9] := $00;
end;

Procedure Init_Interface;
begin
  Initial_8250 ;
```

```
  Disable_Interrupt;
end; { Init_Interface }

Procedure Transmit;
begin { Teansmit }
  Port [TXRX] := Data;
end; {Transmit}

Procedure Receive;
begin { Receive }
  Character := Chr(Port[TXRX]);
end { Receive };

Procedure receive_massage;
begin
  if port[RxCheck] AND $01 = $01 then
    begin
      receive;
      write (Character);
    end;
end;

Procedure Transmit_massage;
begin
   Data := Ord(character);
   Transmit;
end;

Procedure InitCommunication;
begin
   gotoxy (6,23);
   Write ('COMMUNICATION PORT [1],[2] '); Readln (CommPort);
   gotoxy (6,23);
   Write ('RS232: [baudrate,data,stop,parity] '); readln (Setformat);
   Spit_value;
   Init_interface;
end;
```

```
Program : Console Editor
Programmer : Viriya Kongratana

{Program ProgrammingConsoleEditor;}

Procedure GetChar (var Cursor_ch : byte);
{ Procedure to get Charactor at current cursor Position  }
begin
  Regs.AH := $08; { Read Attribute Function }
  Regs.BH := $00; { Active Screen Number }
  Intr ($10,Dos.Registers(Regs));
  Attribute := Regs.AH;  { Attribute byte in AH }
  Cursor_ch := Regs.AL;
end;

Const MaxBox     =  36;
Type  BoxOffT    =  array [1..MaxBox] of byte;
Const BoxOffsetX :  BoxOffT =(30,37,44,51,58,65,
                              30,37,44,51,58,65,
                              30,37,44,51,58,65,
                              30,37,44,51,58,65,
                              30,37,44,51,58,65,23,7,7,4,16,28);
      BoxOffsetY :  BoxOffT =( 2, 2, 2, 2, 2, 2,
                               5, 5, 5, 5, 5, 5,
                               8, 8, 8, 8, 8, 8,
                              11,11,11,11,11,11,
                              14,14,14,14,14,14,14,12,15,22,22,22);

Type  XArray      = Array [0..6] of Byte;
Var   _XPos,_YPos : Byte;
      KeyArray    : Array [0..2] of XArray;
      Ok          : boolean;
    Xlp,Ylp,OX,OY : byte;


Procedure CreatConsole;
begin
   Clrscr;
   TextBackGround (Black);
   TextColor (LightGray);
   Writeln ('+-------------------------------------------------------------------------------+');
   Writeln ('|  PROGRAMMING CONSOLE       +----++----++----++----++----++----+ |mm');
   Writeln ('| +--------------------------+ | FUN || LD || NOT || SFT ||TRANS|| SRCH| |mm');
   Writeln ('| |                          | +----++----++----++----++----++----+ |mm');
   Writeln ('| |                          | +----++----++----++----++----++----+ |mm');
   Writeln ('| +--------------------------+ | AND || OR || OUT || TIM || CNT ||MONTR| |mm');
   Writeln ('|                            +----++----++----++----++----++----+ |mm');
   Writeln ('|                            +----++----++----++----++----++----+ |mm');
```

```
   Writeln ('|                                  | 7 || 8 || 9 || CH ||   # || HEX | |');
   Writeln ('|                                  +---++---++---++----++----++-----+ |');
   Writeln ('|                                  +D--++E--++F--++----++----++-----+ |');
   Writeln ('|   * PROGRAM                      | 4 || 5 || 6 || DEL || INS || DEC | |');
   Writeln ('|                                  +---++---++---++----++----++-----+ |');
   Writeln ('|                          +-------++A--++B--++C--++----++----++-----+ |');
   Writeln ('|   * EXECUTE              | CLR || 1 || 2 || 3 || 0 || ENTR|| INC | |');
   Writeln ('|                          +-------++---++---++---++----++----++-----+ |');
   Writeln ('+-------------------------------------------------------------------------+');
   Writeln ('   ');
   gotoxy (4,22); Write ('+F1----+    +F2----+    +F3----+ ');
   gotoxy (4,23); Write ('| Load |  | Save |  | Exit |  ');
   gotoxy (4,24); Write ('+------+  +------+  +------+  ');
   gotoxy (4,25); Write ('          ');
end;

Procedure Execute;
begin
     Character := 'C';
     Transmit_Massage;
     Delay (1000);
     Character := 'R';
     Transmit_Massage;
end;

Procedure IntegerString (_Integer : Integer ; Var _String : String4);
begin
   Str (_Integer,_String);
   i := Length (_String);
   Case i of
   1 : _String := '000' + _String;
   2 : _String := '00'  + _String;
   3 : _String := '0'   + _String;
   end; {case}
end;

Procedure WriteAddress (Step : Integer);
begin
   IntegerString (Step,StepString);
   gotoxy (4,4);  Write (StepString);
   gotoxy (5,20); Write (StepString);
end;

Procedure WriteIOAssign (Step : Integer);
begin
   IntegerString (Step,StepString);
   gotoxy (24,5);  Write (StepString);
   gotoxy (32,20); Write (StepString);
end;

Procedure SaveCursor;
begin
  TempXCursor  := XCursor;
  TempYCursor  := YCursor;
```

```
end;

Procedure LoadCursor;
begin
  XCursor       := TempXCursor;
  YCursor       := TempYCursor;
end;

Procedure CursorHome;
begin
   gotoxy (61,16);
end;

Procedure GetPositionCursor;
begin
   XCursor := WhereX;
   YCursor := WhereY;
end;

Procedure CalKeyPosition;
begin
   RowKey  := (YCursor - 1) DIV 3;
   RowKey  := RowKey - 1;
   ColKey  := (XCursor - 26) DIV 7;
   ColKey  := ColKey - 1;
   KeyCode := (RowKey * 6) + ColKey;
   If (KeyCode = 23) and (RowKey = 4) then KeyCode := 30;
end;

Procedure SaveFile;
begin
  gotoxy (40,23); ClrEol;
  if OpenWriteFile('Save FileName',FileVar,FileName,'.MIC') then
  begin
    Write (FileVar,Instruction);
    Close (FileVar);
  end;
  gotoxy (40,23); ClrEol;
end;

Procedure LoadFile;
begin
  gotoxy (40,23); ClrEol;
  if OpenReadFile('Load FileName',FileVar,FileName,'.MIC') then
  begin
    Read (FileVar,Instruction);
    Close (FileVar);
  end;
  gotoxy (40,23); ClrEol;
end;

Procedure GetCommandTable (Command : Byte);
begin
  Case Command of
```

```
 00 : CommandLine := 'LD                          ';
 01 : CommandLine := 'LD      NOT                 ';
 02 : CommandLine := 'AND                         ';
 03 : CommandLine := 'AND     NOT                 ';
 04 : CommandLine := 'OR                          ';
 05 : CommandLine := 'OR      NOT                 ';
 06 : CommandLine := 'AND     LD                  ';
 07 : CommandLine := 'OR      LD                  ';
 08 : CommandLine := 'OUT                         ';
 09 : CommandLine := 'OUT     NOT                 ';
 10 : CommandLine := 'TIM                         ';
 11 : CommandLine := 'CNT                         ';
 12 : CommandLine := 'LD          TIM             ';
 13 : CommandLine := 'LD      NOT TIM             ';
 14 : CommandLine := 'LD          CNT             ';
 15 : CommandLine := 'LD      NOT CNT             ';
 16 : CommandLine := 'AND         TIM             ';
 17 : CommandLine := 'AND     NOT TIM             ';
 18 : CommandLine := 'AND         CNT             ';
 19 : CommandLine := 'AND     NOT CNT             ';
 20 : CommandLine := 'OR          TIM             ';
 21 : CommandLine := 'OR      NOT TIM             ';
 22 : CommandLine := 'OR          CNT             ';
 23 : CommandLine := 'OR      NOT CNT             ';
128 : CommandLine := 'NOP                         ';
129 : CommandLine := 'END                         ';
130 : CommandLine := 'IL                          ';
131 : CommandLine := 'ILC                         ';
132 : CommandLine := 'JMP                         ';
133 : CommandLine := 'JME                         ';
138 : CommandLine := 'SFT                    0000';
139 : CommandLine := 'KEEP[11]               0000';
140 : CommandLine := 'CNTR[12]               0000';
141 : CommandLine := 'DIFU[13]               0000';
142 : CommandLine := 'DIFD[14]               0000';
143 : CommandLine := 'TIMH[15]               0000';
144 : CommandLine := 'WSFT[16]               0000';
148 : CommandLine := 'CMP[20]                0000';
149 : CommandLine := 'MOV[21]                0000';
150 : CommandLine := 'MVN[22]                0000';
151 : CommandLine := 'BIN[23]                0000';
152 : CommandLine := 'BCD[24]                0000';
153 : CommandLine := 'ASL[25]                0000';
154 : CommandLine := 'ASR[26]                0000';
155 : CommandLine := 'ROL[27]                0000';
156 : CommandLine := 'ROR[28]                0000';
157 : CommandLine := 'COM[29]                0000';
158 : CommandLine := 'ADD[30]                0000';
159 : CommandLine := 'SUB[31]                0000';
160 : CommandLine := 'MUL[32]                0000';
161 : CommandLine := 'DIV[33]                0000';
162 : CommandLine := 'ANDW[34]               0000';
163 : CommandLine := 'ORW[35]                0000';
164 : CommandLine := 'XORW[36]               0000';
```

```
 165 : CommandLine := 'XNRW[37]              0000';
 166 : CommandLine := 'INC[38]               0000';
 167 : CommandLine := 'DEC[39]               0000';
 168 : CommandLine := 'STC[40]                  ';
 169 : CommandLine := 'CLC[41]                  ';
 198 : CommandLine := 'MOVE BLOCK[70]        0000';
 199 : CommandLine := 'BLOCK SET[71]         0000';
 200 : CommandLine := 'SQR ROOT[72]          0000';
 201 : CommandLine := 'DATA EXCHANGE[73]     0000';
 202 : CommandLine := 'DIGIT SHL[74]         0000';
 203 : CommandLine := 'DIGIT SHR[75]         0000';
 204 : CommandLine := '4TO16 DECODER[75]     0000';
 205 : CommandLine := '16TO4 ENCODER[76]     0000';
 206 : CommandLine := '7 SEGMENT DECODER     0000';
 207 : CommandLine := 'FLOTING DIVIDE        0000';
 208 : CommandLine := 'DATA DIST[80]         0000';
 209 : CommandLine := 'DATA EXTRACTION[81]   0000';
 210 : CommandLine := 'BIT TRANSFER[82]      0000';
 211 : CommandLine := 'DIGIT TRANSFER[83]    0000';
 212 : CommandLine := 'LEFT/RIGHT SFT[84]    0000';
 213 : CommandLine := 'TABLE COMPARE[85]     0000';
 214 : CommandLine := 'WATCHDOG TIMSET[94]   0000';
 215 : CommandLine := 'RUN STOP[95]          0000';
 end;
end;

Procedure Clrdisplay1;
begin
   gotoxy (4,4); Write ('                                   ');
end;

Procedure ClrDisplay2;
begin
   gotoxy (4,5); Write ('                                   ');
end;

Procedure GetIOAssign;
begin
  for i := 3 downto 1 do
  begin
     gotoxy (24+i,5); GetChar (Cursor_Ch);
     Insert (Chr(Cursor_Ch),IOAssign,1);
  end;
end;

Procedure GetAddress;
begin
  for i := 3 downto 1 do
  begin
     gotoxy (4+i,4); GetChar (Cursor_Ch);
     Insert (Chr(Cursor_Ch),Address,1);
  end;
end;
```

```
Procedure GetFunction;
begin
  for i := 1 downto 1 do
  begin
     gotoxy (14+i,4); GetChar (Cursor_Ch);
     Insert (Chr(Cursor_Ch),FunctionString,1);
  end;
end;

Procedure NumericAssign (Number : Byte);
begin
   If FunctionFlag = ON then
   begin
      GetFunction;
      Case Number of
      12 : Ch :=  '7';
      13 : Ch :=  '8';
      14 : Ch :=  '9';
      18 : Ch :=  '4';
      19 : Ch :=  '5';
      20 : Ch :=  '6';
      24 : Ch :=  '1';
      25 : Ch :=  '2';
      26 : Ch :=  '3';
      27 : Ch :=  '0';
      end;
      Insert (Ch,FunctionString,2);
      gotoxy (14,4); Write (FunctionString);
      ChStr := Copy (FunctionString,1,1);
      Ch := ChStr[1];
      If (Ch >=#48) and (Ch <=#57) then FunctionDecode := ON;
   end
   else
   If CommandFlag = ON then
   begin
      GetIOAssign;
      Case Number of
      12 : Ch :=  '7';
      13 : Ch :=  '8';
      14 : Ch :=  '9';
      18 : Ch :=  '4';
      19 : Ch :=  '5';
      20 : Ch :=  '6';
      24 : Ch :=  '1';
      25 : Ch :=  '2';
      26 : Ch :=  '3';
      27 : Ch :=  '0';
      end;
      Insert (Ch,IOAssign,4);
      gotoxy (24,5); Write (IOAssign);
   end;
end;

Procedure SaveCommand;
```

```
begin
  Instruction[Step_Address].InsAddress := Step_Address;
  Instruction[Step_Address].Opcode := Command;
  If Command IN[_NOP,_END,_IL,_ILC,_JMP,_JME,_STC,_CLC,_AND_LD,_OR_LD] then
  Instruction[Step_Address].Assignment1 := 0
  else
  begin
     Val (IOAssign,AssignIO,Code);
     Instruction[Step_Address].Assignment1 := AssignIO;
     Instruction[Step_Address].DataType1   := TypeData;
     Instruction[Step_Address].Assignment2 := 0;
     Instruction[Step_Address].Assignment3 := 0;
  end;
end;

Procedure WriteParameter;
begin
   AssignIO1     :=Instruction[Step_Address].Assignment1;
   AssignIO2     :=Instruction[Step_Address].Assignment2;
   AssignIO3     :=Instruction[Step_Address].Assignment3;
   TypeData1     :=Instruction[Step_Address].DataType1;
   TypeData2     :=Instruction[Step_Address].DataType2;
   TypeData3     :=Instruction[Step_Address].DataType3;
   IntegerString (AssignIO1,AssignStr1);
   IntegerString (AssignIO2,AssignStr2);
   IntegerString (AssignIO3,AssignStr3);
   gotoxy (5,21);
   If AssignStr1 = '0000' then Write ('Data[A]: ','    ')else Write ('Data[A]: ',AssignStr1);
   gotoxy (20,21);
   If AssignStr2 = '0000' then Write ('Data[B]: ','    ')else Write ('Data[B]: ',AssignStr2);
   gotoxy (35,21);
   If AssignStr3 = '0000' then Write ('Data[C]: ','    ')else Write ('Data[C]: ',AssignStr3);
   gotoxy (60,21); ClrEol;
   gotoxy (60,21); Write (TypeData1,' ',TypeData2,' ',TypeData3);
end;

Procedure DisplayInstruction;
begin
   gotoxy (5,20); ClrEol;
   WriteAddress (Step_Address);
   GetCommandTable (Command);
   gotoxy (4,5);  Write (CommandLine);
   gotoxy (11,20); Write (CommandLine);
   If Command IN[_NOP,_END,_IL,_ILC,_JMP,_JME,_STC,_CLC,_AND_LD,_OR_LD] then
   begin
      gotoxy (24,5); Write ('    ');
   end
   else
   WriteIOAssign (AssignIO);
end;

Procedure ParameterFunction;
begin
   If (Command < 128) AND (NOT (Command IN[10,11])) then FunctionParam := 1
```

```
   else
   begin
       case Command of
       _CLC,_STC,_NOP,_END,_IL,_ILC,_JMP,_JME                : FunctionParam := 0;

       _KEEP,_DIFU,_DIFD,_TIMH,_ASL,_ASR,
       _ROL,_ROR,_COM,_INC,_DEC                              : FunctionParam := 1;

       _TIM,_CNT,_SFT,_CNTR,_WSFT,_CMP,_MOV,_MVN,_BIN,_BCD   : FunctionParam := 2;
       else FunctionParam := 3;
   end;{case}
   If FunctionParam  > 1 then  ParamFlag := ON
   end; {If}
end;

Procedure FunctionParameter;
begin
  If (ParamCount > FunctionParam) AND (ParamFlag = ON) then
  begin
     ParamFlag := OFF;
     ParamFlag1:= OFF;
     ParamCount:= 1;
     Step_Address := Step_Address + 1;
     Command      :=Instruction[Step_Address].Opcode;
     AssignIO     :=Instruction[Step_Address].Assignment1;
     DisplayInstruction;
  end;
end;

Procedure BoxCommandCheck(X,Y : byte;var Boxnum : byte);
var i : byte;
    X0,Y0,X1,Y1 : byte;

function CheckInBox : boolean;
begin
  CheckInBox := false;
  if (X0 <= X) and (X <= X1) then
  if (Y0 <= Y) and (Y <= Y1) then
    CheckInBox := true;
end;

begin
    for i := 1 to MaxBox do
    begin
      X0 := BoxOffsetX[i];
      Y0 := BoxOffsetY[i];
      X1 := BoxOffsetX[i]+7;
      Y1 := BoxOffsetY[i]+3;
      if CheckInBox then
      begin
        Boxnum := i;
      end;
    end;
end;
```

```
Procedure KeyDisp(Mode:boolean; Boxnum:byte);
var X0,Y0,X1,Y1:byte ;
var Buf : byte;
    i,j : byte;
    TxtColor,GndColor : byte;
begin
  X0 := BoxOffsetX[Boxnum];
  Y0 := BoxOffsetY[Boxnum];
  if Boxnum in [34,35,36] then
  begin
    X1 := BoxOffsetX[Boxnum]+8;
    Y1 := BoxOffsetY[Boxnum]+3;
  end
  else
  begin
    if Boxnum in [32,33] then
    begin
      X1 := BoxOffsetX[Boxnum]+7;
      Y1 := BoxOffsetY[Boxnum];
    end
    else
    begin
      X1 := BoxOffsetX[Boxnum]+7;
      Y1 := BoxOffsetY[Boxnum]+3;
    end;
  end;
  if Mode = true then
  begin
    TxtColor := Black;
    GndColor := LightGray;
  end
  else
  begin
    TxtColor := LightGray;
    GndColor := Black;
  end;
  if X0 = X1 then X1 := X0+1;
  if Y0 = Y1 then Y1 := Y0+1;
  for j := Y0 to Y1-1 do
  for i := X0 to X1-1 do
  begin
    gotoxy (i,j);
    GetChar(Buf);
    TextBackGround(GndColor);
    TextColor(TxtColor);
    Write(Chr(Buf));
  end;
end;

procedure KeyBeep;
begin
  Sound(4000);delay(100);Nosound;
end;
```

```
Procedure CheckScreenCommand;
var Boxnum : byte;
begin
  BoxCommandCheck(Xlp,Ylp,Boxnum);
  if Boxnum in [1..MaxBox] then
  begin
    KeyDisp(True,Boxnum);
    KeyBeep;
    Delay(400);
    KeyDisp(false,Boxnum);
  end;
  if Boxnum > 0 then
    KeyCode := Boxnum-1;
end;

function ReadLpOk(var X,Y : byte) : boolean;
begin
    ReadLpOk := false;
    Regs.AH := $04;
    Intr ($10,Dos.Registers(Regs));
    Xlp := Regs.DL + 1;
    Ylp := Regs.DH + 1;
    if Regs.AH = 1 then
    begin
      repeat
        Regs.AH := $04;
        Intr ($10,Dos.Registers(Regs));
      until Regs.AH <> 1;
      ReadLpOk := true;
    end;
    if Xlp = 0 then Xlp := 1;
    if Ylp = 0 then Ylp := 1;
end;

Procedure LoadFileInstruction;
begin
   SaveCursor;
   LoadFile;
   Step_address := 0;
   Command      :=Instruction[Step_Address].Opcode;
   AssignIO     :=Instruction[Step_Address].Assignment1;
   DisplayInstruction;
   LoadCursor;
   gotoxy (Xcursor,Ycursor);
end;

Procedure SaveFileInstruction;
begin
   SaveCursor;
   SaveFile;
   Step_address := 0;
   Command      :=Instruction[Step_Address].Opcode;
   AssignIO     :=Instruction[Step_Address].Assignment1;
```

```
   DisplayInstruction;
   LoadCursor;
   gotoxy (Xcursor,Ycursor);
end;

Procedure GetKeyCommand;
begin
   Ok := false;
       SaveCursor;
       Repeat
          If ReadLpOK (Xlp,Ylp) then
          begin
              OK := True;
              CheckScreenCommand;
          end;
       Until KeyPressed OR OK;
       LoadCursor;
       If NOT Ok then
       begin
           OK := false;
           Repeat
              Ch := ReadKey;
              If (Ch = #0) and KeyPressed then
              begin
                 GetPositionCursor;
                 Ch := ReadKey;
                 Case Ch of
              F1   : LoadFileInstruction;
              F2   : SaveFileInstruction;

              Up   : begin
                        YCursor := YCursor - 3;
                        If Ycursor = 1 then YCursor := 16;
                        gotoxy (XCursor,YCursor);
                     end;
              Down : begin
                        YCursor := YCursor + 3;
                        If YCursor = 19 then YCursor := 4;
                        gotoxy (XCursor,YCursor);
                     end;
              Left : begin
                        XCursor := XCursor - 7;
                        If YCursor = 16 then
                        begin
                            If XCursor = 19 then XCursor := 68;
                        end
                        else
                        If XCursor = 26 then XCursor := 68;
                        gotoxy (XCursor,YCursor);
                     end;

              Right : begin
                        XCursor := XCursor + 7;
                        If YCursor = 16 then
```

```
                        begin
                            If XCursor = 75 then XCursor := 26;
                        end
                        else
                        If XCursor = 75 then XCursor := 33;
                        gotoxy (XCursor,YCursor);
                    end;

                end; {Case}
              end; {If}
            Until (Ch = CR) OR (Ch = F3);
            GetPositionCursor;
            CalKeyPosition;
        end;
end;

Procedure PushData1;
begin
   TypeData1  := Instruction[Step_Address].DataType1;
   DataTransfer[Step_Count] := TypeData1; Inc (Step_Count);
   AssignIO1  := Instruction[Step_Address].Assignment1;
   J          := AssignIO1 AND $00FF;
   DataTransfer[Step_Count] := J; Inc (Step_Count);
   AssignIO1  := AssignIO1 SHR 8;
   K          := AssignIO1 AND $00FF;
   DataTransfer[Step_Count] := K; Inc (Step_Count);
end;

Procedure PushData2;
begin
   TypeData2  := Instruction[Step_Address].DataType2;
   DataTransfer[Step_Count] := TypeData2; Inc (Step_Count);
   AssignIO2  := Instruction[Step_Address].Assignment2;
   J          := AssignIO2 AND $00FF;
   DataTransfer[Step_Count] := J; Inc (Step_Count);
   AssignIO2  := AssignIO2 SHR 8;
   K          := AssignIO2 AND $00FF;
   DataTransfer[Step_Count] := K; Inc (Step_Count);
end;

Procedure PushData3;
begin
   TypeData3  := Instruction[Step_Address].DataType3;
   DataTransfer[Step_Count] := TypeData3; Inc (Step_Count);
   AssignIO3 := Instruction[Step_Address].Assignment3;
   j          := AssignIO3 AND $00FF;
   DataTransfer[Step_Count] := J; Inc (Step_Count);
   AssignIO3  := AssignIO3 SHR 8;
   K          := AssignIO3 AND $00FF;
   DataTransfer[Step_Count] := K; Inc (Step_Count);
end;

Procedure ErrorNoENDIns;
begin
```

```
   ClrDisPlay2;
   Write (Chr($07));
   gotoxy (4,5); Write ('NO END INS');
   TransferFlag := OFF;
end;

Procedure FindEndStep;
begin
   Error := True;
   Step_Address := 0;
   Command := Instruction[Step_Address].Opcode;
   While (Command <> 129) AND (Step_Address < 2000) do
   begin
      Command := Instruction[Step_Address].Opcode;
      Inc (Step_address);
   end;
   If Step_Address >= 2000 then ErrorNoENDIns else
   begin
      Step_End := Step_Address - 1;
      Error := False;
   end;
end;

Procedure UpLoad;
begin
end;

Procedure DownLoad;
begin
  If NOT Error then
  begin
     ClrDisplay2;
     gotoxy (4,5);
     Character := 'L';
     Transmit_massage;Delay (200);
     Delay (1000);
     Step_Address := Step_Address + 1;
     Data := Step_Address AND $00FF;    {Range Low Byte IN BC Reg}
     Transmit; Delay (500); Write (data);
     Data :=  (Step_Address AND $FF00) SHR 4;
     Transmit; Delay (500); Write (data);

     gotoxy (4,5); Write ('TRANSFERING');
     for Step_Count := 0 to Step_Address do
     begin
        Data := DataTransfer[Step_Count];
        Transmit; Delay (100);
     end;

     gotoxy (4,5); Write ('TRANSFER OK');
  end;
end;

Procedure ConvertCode;
```

```
begin
  FindEndStep;
  If NOT Error then
  begin
     Step_Count := 0;
     for Step_Address := 0 to Step_End do
     begin
        Command := Instruction[Step_Address].Opcode;
        DataTransfer[Step_Count] := Command; Inc (Step_Count);
        ParameterFunction;
        Case FunctionParam of
        1 : PushData1;

        2 : begin
              PushData1;
              PushData2;
            end;

        3 : begin
              PushData1;
              PushData2;
              PushData3;
            end;
         end; {case}
     end;
  end;
  Step_Address := Step_Count - 1;
end;

Procedure DisConvertCode;
begin
end;

Procedure ExecCommandKey;
begin
  If (KeyCode IN[0,1,2,3,4,5,6,7,8,9,10,11,15,21,22,23,29]) AND (ParamFlag1 = ON) then
  begin
    Write (Chr($07));
  end
  else
begin
  Case KeyCode of
  0 :  begin
          ClrDisplay2;
          gotoxy (10,4); Write ('FUN ??');
          AssignIO := 0;
          CommandFlag := ON;
          FunctionFlag := ON;
          ClrFlag := OFF;
       end;
  1 :  begin
         If LastCommand IN [_AND,_OR] then
         begin
           gotoxy (9,5); Write ('LD                ');
```

```
            Case LastCommand of
            _AND : Command := _AND_LD;
            _OR  : Command := _OR_LD;
            end;
            CommandFlag := ON;
            ClrFlag := OFF;
         end
         else
         begin
            ClrDisplay1;
            WriteAddress (Step_Address);
            ClrDisplay2;
            gotoxy (4,5); Write ('LD                    0000');
            IOAssign := '0000';
            Command := _LD;
            CommandFlag := ON;
            ClrFlag := OFF;
         end;
         FunctionFlag := OFF;
      end;
 2 :  begin
         If CommandFlag = ON then
         begin
            If LastCommand IN[_FUN,_SFT,_TIM,_CNT] then
            begin
               Write (Chr($07));
               Command := LastCommand;
               CommandFlag := ON;
               ClrFlag := OFF;
            end
            else
            begin
               ClrDisplay1;
               WriteAddress (Step_Address);
               gotoxy (9,5); Write ('NOT');
               Case LastCommand of
               _LD       : Command := _LD_NOT;
               _AND      : Command := _AND_NOT;
               _OR       : Command := _OR_NOT;
               _OUT      : Command := _OUT_NOT;
               _LD_TIM   : Command := _LD_NOT_TIM;
               _LD_CNT   : Command := _LD_NOT_CNT;
               _AND_TIM  : Command := _AND_NOT_TIM;
               _AND_CNT  : Command := _AND_NOT_CNT;
               _OR_TIM   : Command := _OR_NOT_TIM;
               end; {Case}
               CommandFlag := ON;
               ClrFlag := OFF;
            end;
         end;
         FunctionFlag := OFF;
      end;

 3 :  begin
```

```
        CommandFlag := ON;
        Command := _SFT;
        AssignIO := 0;
        DisplayInstruction;
        ParamFlag := ON;
        FunctionParam := 2;
        ClrFlag := OFF;
      end;

 4:   begin
        ClrDisplay1;
        ClrDisplay2;
        If TransferFlag = OFF then
        begin
          gotoxy(4,5); Write ('HOST TO CONTROLLER');
          Write (Chr($07));
          TransferFlag := ON;
          Transfer := 'D';  {HOST TO COMPUTER}
        end
        else
        begin
          gotoxy(4,5); Write ('CONTROLLER TO HOST');
          Write (Chr($07));
          TransferFlag := OFF;
          Transfer := 'U'; {COMPUTER TO HOST}
        end;
      end;

 5 :  begin
        Val (IOAssign,AssignIO,Code);
        Step_Address := Step_Address + 1;
        Repeat
          CommandSearch  :=Instruction[Step_Address].Opcode;
          AssignIOSearch :=Instruction[Step_Address].Assignment1;
          Step_Address := Step_Address + 1;
        Until (Command = CommandSearch) and (AssignIO = AssignIOSearch) or (Step_Address >= Maxstep);
        Step_Address := Step_Address - 1;
        If Step_Address >= (Maxstep-1) then
        begin
          ClrDisplay1;
          ClrDisplay2;
          gotoxy(4,5); Write ('NOT FOUND');
        end
        else
        begin
          ClrDisplay1;
          Command   :=Instruction[Step_Address].Opcode;
          AssignIO  :=Instruction[Step_Address].Assignment1;
          DisplayInstruction;
        end;
      end;

 6 :  begin
        ClrDisplay1;
```

```
        WriteAddress (Step_Address);
        ClrDisplay2;
        gotoxy (4,5); Write ('AND                0000');
        IOAssign := '0000';
        Command := _AND;
        CommandFlag := ON;
        ClrFlag := OFF;
        FunctionFlag := OFF;
      end;

  7 : begin
        ClrDisplay1;
        WriteAddress (Step_Address);
        ClrDisplay2;
        gotoxy (4,5); Write ('OR                 0000');
        IOAssign := '0000';
        Command := _OR;
        CommandFlag := ON;
        ClrFlag := OFF;
        FunctionFlag := OFF;
      end;

  8 : begin
        ClrDisplay1;
        WriteAddress (Step_Address);
        ClrDisplay2;
        gotoxy (4,5); Write ('OUT                0000');
        IOAssign := '0000';
        Command := _OUT;
        CommandFlag := ON;
        ClrFlag := OFF;
        FunctionFlag := OFF;
      end;

  9 : begin
        If LastCommand IN[_LD,_AND,_OR,_LD_NOT,_AND_NOT,_OR_NOT] then
        begin
          gotoxy (15,5); Write ('TIM       000');
          IOAssign := '000';
          Case LastCommand of
          _LD       : Command := _LD_TIM;
          _AND      : Command := _AND_TIM;
          _OR       : Command := _OR_TIM;
          _LD_NOT   : Command := _LD_NOT_TIM;
          _AND_NOT  : Command := _AND_NOT_TIM;
          _OR_NOT   : Command := _OR_NOT_TIM;
          end;
          CommandFlag := ON;
          ClrFlag := OFF;
        end
        else
        begin
          ClrDisplay1;
          WriteAddress (Step_Address);
```

```
            ClrDisplay2;
            IOAssign := '0000';
            gotoxy (4,5); Write ('TIM                  000');
            Command := _TIM;
            CommandFlag := ON;
            ParamFlag := ON;
            FunctionParam := 2;
         end;
         FunctionFlag := OFF;
      end;

10 :  begin
         If LastCommand IN[_LD,_AND,_OR,_LD_NOT,_AND_NOT,_OR_NOT] then
         begin
            gotoxy (15,5); Write ('CNT      000');
            IOAssign := '000';
            Case LastCommand of
            _LD       : Command := _LD_CNT;
            _AND      : Command := _AND_CNT;
            _OR       : Command := _OR_CNT;
            _LD_NOT   : Command := _LD_NOT_CNT;
            _AND_NOT  : Command := _AND_NOT_CNT;
            _OR_NOT   : Command := _OR_NOT_CNT;
            end;
            CommandFlag := ON;
            ClrFlag := OFF;
         end
         else
         begin
            ClrDisplay1;
            WriteAddress (Step_Address);
            ClrDisplay2;
            IOAssign := '0000';
            gotoxy (4,5); Write ('CNT                  000');
            Command := _CNT;
            CommandFlag := ON;
            ClrFlag := OFF;
            ParamFlag := ON;
            FunctionParam := 2;
         end;
         FunctionFlag := OFF;
      end;

12,13,14,18,19,20,24,25,26,27 : begin
                                   If LastCommand IN[_AND_LD,_OR_LD] then Write (Chr($07)) else
                                   NumericAssign (KeyCode);
                                end;

16 :  begin
         If ParamFlag = ON then
         begin
         If DataTypeFlag = OFF then
         begin
            DataTypeFlag := ON;
```

```
            TypeData     := _DtaType;
            gotoxy (23,5); Write ('#');
        end
        else
        begin
            DataTypeFlag := OFF;
            TypeData     := _IOType;
            gotoxy (23,5); Write (' ');
        end; {If}
        end; {If}
    end;

17 :  begin
        HexFlag := ON;
      end;

21 :  begin
        StepDelete := Step_Address;
        repeat
          Command      :=Instruction[Step_Address+1].Opcode;
          AssignIO1    :=Instruction[Step_Address+1].Assignment1;
          TypeData1    :=Instruction[Step_address+1].DataType1;
          AssignIO2    :=Instruction[Step_Address+1].Assignment2;
          TypeData2    :=Instruction[Step_address+1].DataType2;
          AssignIO3    :=Instruction[Step_Address+1].Assignment3;
          TypeData3    :=Instruction[Step_address+1].DataType3;

          Instruction[Step_Address].Opcode := Command;
          Instruction[Step_Address].Assignment1 := AssignIO1;
          Instruction[Step_Address].DataType1   := TypeData1;
          Instruction[Step_Address].Assignment2 := AssignIO2;
          Instruction[Step_Address].DataType2   := TypeData2;
          Instruction[Step_Address].Assignment3 := AssignIO3;
          Instruction[Step_Address].DataType3   := TypeData3;

          Step_Address := Step_Address + 1;
        Until Step_Address = Maxstep;
        Step_Address := StepDelete;
        Command      :=Instruction[Step_Address].Opcode;
        AssignIO     :=Instruction[Step_Address].Assignment1;
        DisplayInstruction;
        gotoxy (10,4); Write ('DELETE READY'); Write (Chr($07));
        gotoxy (10,4); Write ('             ');
      end;

22 :  begin
        StepInsert   := Step_Address;
        StepTemp     := MaxStep;
        CommandTemp  := Command;
        AssignIOTemp := AssignIO;
        repeat
          Command    :=Instruction[StepTemp-1].Opcode;
          AssignIO1  :=Instruction[StepTemp-1].Assignment1;
          AssignIO2  :=Instruction[StepTemp-1].Assignment2;
```

```
            AssignIO3     :=Instruction[StepTemp-1].Assignment3;
            TypeData1     :=Instruction[StepTemp-1].DataType1;
            TypeData2     :=Instruction[StepTemp-1].DataType2;
            TypeData3     :=Instruction[StepTemp-1].DataType3;

            Instruction[StepTemp].Opcode := Command;
            Instruction[StepTemp].Assignment1 := AssignIO1;
            Instruction[StepTemp].Assignment2 := AssignIO2;
            Instruction[StepTemp].Assignment3 := AssignIO3;
            Instruction[StepTemp].DataType1   := TypeData1;
            Instruction[StepTemp].DataType2   := TypeData2;
            Instruction[StepTemp].DataType3   := TypeData3;

            StepTemp := StepTemp - 1;
          Until StepTemp = stepInsert;
          Step_Address := StepInsert;
          Command  := CommandTemp;
          AssignIO := AssignIOTemp;
          SaveCommand;
          DisplayInstruction;
          gotoxy (10,4); Write ('INSERT READY'); Write (Chr($07));
          gotoxy (10,4); Write ('                ');
        end;

28 :  begin
        If ParamFlag = ON then
        begin
          ClrDisplay1;
          ClrDisplay2;
          ParamFlag1 := ON;
          Instruction[Step_Address].InsAddress := Step_Address;
          Instruction[Step_Address].Opcode := Command;
          Val (IOAssign,AssignIO,Code);
          case ParamCount of
          1 : begin
                Instruction[Step_Address].Assignment1 := AssignIO;
                Instruction[Step_Address].DataType1   := TypeData;
              end;
          2 : begin
                Instruction[Step_Address].Assignment2 := AssignIO;
                Instruction[Step_Address].DataType2   := TypeData;
              end;
          3 : begin
                Instruction[Step_Address].Assignment3 := AssignIO;
                Instruction[Step_Address].DataType3   := TypeData;
              end;
          end;{case}
          case ParamCount of
          1 : begin
                gotoxy (4,5);
                Write ('Data B');
              end;
          2 : begin
                gotoxy (4,5);
```

```
                Write ('Data C');
              end;
          end; {case}
          ParamCount := ParamCount + 1;
        end
        else
        If TransferFlag = ON then
        begin
          case Transfer of
          'D' : begin
                  ConvertCode;
                  DownLoad;
                  TransferFlag := OFF;
                end;
          'U' : begin
                  UpLoad;
                  DisConvertCode;
                  TransferFlag := OFF;
                end;
          end; {case}
        end
        else
        If CommandFlag = ON then
        begin
          ClrDisplay1;
          ClrDisplay2;
          SaveCommand;
          Step_Address := Step_Address + 1;
          Command      :=Instruction[Step_Address].Opcode;
          AssignIO     :=Instruction[Step_Address].Assignment1;
          DisplayInstruction;
          CommandFlag := OFF;
          FunctionFlag:= OFF;
          Command := _NOP;
          ClrFlag := OFF;
        end
        else
        Write (Chr($07));
        DataTypeFlag := OFF;
        TypeData     := _IOType;
      end;

23  :  begin
         Step_Address := Step_Address - 1;
         If Step_Address < 0 then Step_Address := 0;
         Command      :=Instruction[Step_Address].Opcode;
         AssignIO     :=Instruction[Step_Address].Assignment1;
         DisplayInstruction;
         WriteParameter;
         Clrflag := OFF;
       end;

29  :  begin
```

```
            Step_Address := Step_Address + 1;
            If Step_Address > MaxStep then Step_Address := MaxStep;
            Command      :=Instruction[Step_Address].Opcode;
            AssignIO     :=Instruction[Step_Address].Assignment1;
            DisplayInstruction;
            WriteParameter;
            Clrflag := OFF;
          end;

 30  :  begin
            If ClrFlag = ON then
            begin
               ClrDisPlay1;
               ClrDisPlay2;
               Step_Address := 0;
               WriteAddress (Step_Address);
               ClrFlag := OFF;
               Command := _NOP;
               CommandFlag := OFF;
            end
            else
            begin
               ClrDisPlay2;
               ClrFlag := ON;
               CommandFlag := OFF;
               Command := _NOP;
            end;
         end;

 32  :  Execute;

 33  :  LoadFileInstruction;
 34  :  SaveFileInstruction;
 35  :  Ch := F3;


  end;{Case}
  If FunctionDecode = ON then
  begin
     FunctionDecode := OFF;
     FunctionFlag   := OFF;
     Val(FunctionString,Command,Result);
     Command := Command + 128;
     DisplayInstruction;
     ParameterFunction;
  end
  else
  FunctionParameter;
  LastCommand := Command;
  gotoxy (XCursor,YCursor);
end;
end;

Procedure ClearCommand;
```

```
begin
   for Step_Address := 0 to Maxstep do
   begin
      Instruction[Step_Address].InsAddress := Step_Address;
      Instruction[Step_Address].Opcode := _NOP;

      Instruction[Step_Address].Assignment1 := 0;
      Instruction[Step_Address].Assignment2 := 0;
      Instruction[Step_Address].Assignment3 := 0;

      Instruction[Step_Address].DataType1   := 0;
      Instruction[Step_Address].DataType2   := 0;
      Instruction[Step_Address].DataType3   := 0;

   end;
end;

Procedure InitialParameter;
begin
   Ch                   := 'S';
   LastCommand          := _FUN[28];
   ClrFlag              := OFF;
   CommandFlag          := OFF;
   TransferFlag         := OFF;
   FunctionFlag         := OFF;
   FunctionDecode       := OFF;
   ParamFlag            := OFF;
   ParamFlag1           := OFF;
   DataTypeFlag         := OFF;
   IOAssign             := '0000';
   Address              := '0000';
   FunctionParam        := 0;
   TypeData             := _IOType;
   ParamCount           := 1;
   ClearCommand;
   SaveCursor;
   Step_address         := 0;
   Command      :=Instruction[Step_Address].Opcode;
   AssignIO     :=Instruction[Step_Address].Assignment1;
   DisplayInstruction;
   LoadCursor;
end;

Procedure ConsoleEdit;
begin
   Repeat
      GetKeyCommand;
      ExecCommandKey;
   Until Ch = F3;
end;

Procedure EditConSoleKey;
begin
   Clrscr;
```

```
   CreatConsole;
   InitialParameter;
   CursorHome;
   ConsoleEdit;
end;
```

```
Program : Diagnostic Routine
Programmer : Viriya Kongratana

Type  RangeStr     = String[4];
      FileDown     = File of Byte;
Var   Filenameaddr : String[20];
      Rangeaddr    : RangeStr;
      AddressDiag  : String[4];
      FileVarDiag  : FileDown;
      Side         : Integer;


{procedure Transmit;
begin
  Port [$2F8] := Data;
end;

procedure Receive;
begin
  Character := Chr(Port[$2F8]);
end

Procedure receive_massage;
begin
  if port[$2FD] AND $01 = $01 then
    begin
      receive;
      write (Character);
    end;
end;

procedure Transmit_massage;
begin
   Data := Ord(character);
   Transmit;
end;}

Procedure converse_hex (var number : integer; var rangeaddr : rangestr);
type character = string[1];
var hex : array [1..4] of character;
    num : array [1..4] of byte;
    i : byte;
    number1,number2 : integer;
procedure check_byte (number:byte);
begin
   num[1] := number mod 16;
   num[2] := number div 16;
end;
```

```
procedure check_integer(number:integer);
begin
   number1 := number div 256;
   number2 := number mod 256;
   check_byte (number2);
   num[3] := number1 mod 16;
   num[4] := number1 div 16;
end;

procedure converse_hexchar;
var i : byte;
begin
   for i := 1 to 4 do
   begin
     if num[i] >= 10 then
     case num[i] of
     10 : hex[i] := 'A';
     11 : hex[i] := 'B';
     12 : hex[i] := 'C';
     13 : hex[i] := 'D';
     14 : hex[i] := 'E';
     15 : hex[i] := 'F';
     end
     else
     str (num[i],hex[i]);
   end;
end;

begin
   num[1] := 0; num[2] := 0; num[3] := 0; num[4] := 0;
   if number <= 255 then check_byte (number) else check_integer (number);
   converse_hexchar;
   rangeaddr := concat (hex[4],hex[3],hex[2],hex[1]);
end;

function check_IO (var fi : filedown) : boolean;
begin
 {$I-} reset (fi); {$I+}
 check_IO := IOresult = 0;
end;

Procedure transfer_file;
begin
  write ('Load FileName:');readln (filenameaddr);
  filename := copy(filenameaddr,1,pos(',',filenameaddr)-1);
  delete (filenameaddr,1,length(filename)+1);
  addressDiag := filenameaddr;
  assign (filevarDiag,filename);
  if check_IO (filevarDiag) then
  begin
  reset (filevarDiag);
  Side := Filesize(FilevarDiag);
  converse_hex (Side,Rangeaddr);
```

```
    Transmit_massage;
    repeat
      receive_massage;
    until ord(character) = $0A;
    repeat
      receive_massage;
    until ord(character) = $3A;
    receive_massage;
    character := chr($20); transmit_massage; delay(100); receive_massage;
    for i := 1 to 4 do
      begin
        ChStr := copy(AddressDiag,i,1);
        character := ChStr[1];
        transmit_massage; delay(100); receive_massage;
      end;
    character := chr($0D); transmit_massage;
    repeat
      receive_massage;
    until ord(character) = $3A;
    receive_massage;
    character := chr($20); transmit_massage; delay(100); receive_massage;
    for i := 1 to 4 do
      begin
        ChStr := copy(rangeaddr,i,1);
        character := ChStr[1];
        transmit_massage; delay(100); receive_massage;
      end;
    character := chr($0D); transmit_massage;
    for count := 1 to side do
      begin
        read (filevarDiag,data);
        transmit;delay(20);
      end;
    close (filevarDiag);
    writeln;
    end
    else writeln ('File not exist, or bad parameter');
end;

Procedure comunication;
begin
    repeat
        repeat
            receive_massage;
        until keypressed;
        Character := ReadKey;
        Character := upcase(Character);
      if Upcase(Character) = 'L' then Transfer_file else Transmit_massage;
    Until Upcase(Character) = 'Q';
end;

Procedure Diagnostic;
begin
  Clrscr;
```

```
  TextColor (LightGray);
  WriteLn('PC & DATALOGGER DIAGNOSTIC SYSTEM');
  WriteLn('Instrumentation Research KMITL  1988');
  WriteLn;
  repeat
    Comunication;
  until Upcase(Character) = 'Q';
end;
```

```
Program : Menu Management
Programmer : Viriya Kongratana

Procedure MenuItem;
begin
   WriteLn('+-------------------------------------------------------------------------+');
   WriteLn('|                                                                         |');
   WriteLn('|                        +-------------------- COMMUNICATION -+           |');
   WriteLn('|                      +-+------------------- LADDER DIAGRAM -+ |           |');
   WriteLn('|                    +-+--------------------- SYSTEM DIAGNOSTIC -+ | |     |');
   WriteLn('|                  +-+----------------------- MONITORING I/O TABLE -+ | | |');
   WriteLn('|                +-+------------------------- EDIT I/O TABLE -+ | | +-+   |');
   WriteLn('|              +-+--------------------------- PROGRAMMING CONSOLE -+ | | +-+ |');
   WriteLn('|              |                                                  | | +-+   |');
   WriteLn('|              |                                                  | +-+     |');
   WriteLn('|              |                                  HOST LINK UNIT +-+       |');
   WriteLn('|              +-----------------------------------------------+             |');
   WriteLn('|                                                                         |');
   WriteLn('|                                                                         |');
   WriteLn('|                                                                         |');
   WriteLn('|                                                                         |');
   WriteLn('|                                                                         |');
   WriteLn('|                                                                         |');
   WriteLn('|                                                                         |');
   WriteLn('| +-------------------------------------------------------+               |');
   WriteLn('| |  Programmable Controller and DataLogger System        |               |');
   WriteLn('| |  Instrumentation Research  KMITL     June 1988        |               |');
   WriteLn('| |  Select:                                              |               |');
   Writeln('| +-------------------------------------------------------+               |');
   Write  ('+-------------------------------------------------------------------------+');
end;

Procedure MenuSelect (Var Ch : Char);
begin
  Clrscr;
  TextColor (LightGray);
  MenuItem;
  Ch := ReadKey;
end;
```

```
Program : Table Monitoring
Programmer : Viriya Kongratana

Type  IOCH   = Array [0..33] of Integer;
      TIMCH  = Array [0..16] of Integer;
      CNTCH  = Array [0..16] of Integer;
      TIMSTA = Array [0..16] of Byte;
      CNTSTA = Array [0..16] of Byte;

Var   IOCHDisplay  : IOCH;
      TIMCHDisplay : TIMCH;
      CNTCHDisplay : CNTCH;
      TIMStatus    : TIMSTA;
      CNTStatus    : CNTSTA;
      IOCHWord     : Integer;
      WordIO       : Integer;
      IOCHWordTemp : Integer;
      IOCHBIN      : Array [0..3] of String[4];
      IOCHBINSTR   : String[16];
      IOCHHEXSTR   : String[4];
      IOCHHEX      : Array [0..3] of Char;
      DataLO,DataHI: Byte;
      DataByte     : Byte;
      L,M,N        : Byte;

Procedure Cursor (CursorType : Boolean);
begin
   if Cursortype = true then Regs.CX := $0707 else
   Regs.CH := $20;
   Regs.AH := $01;
   Intr ($10,Dos.Registers(Regs));
end;

Procedure CreatTable1;
begin
   Clrscr;
   WriteLn('+------------------------- MONITORING TABLE 01 -------------------------+');
   WriteLn('|                                                                       |');
   WriteLn('+------+-----------------------------+------+-----------------------------+');
   WriteLn('|CH00 | 0000000000000000     [0000]     |CH17| 0000000000000000     [0000]     |');
   WriteLn('|CH01 | 0000000000000000     [0000]     |CH18| 0000000000000000     [0000]     |');
   WriteLn('|CH02 | 0000000000000000     [0000]     |CH19| 0000000000000000     [0000]     |');
   WriteLn('|CH03 | 0000000000000000     [0000]     |CH20| 0000000000000000     [0000]     |');
   WriteLn('|CH04 | 0000000000000000     [0000]     |CH21| 0000000000000000     [0000]     |');
   WriteLn('|CH05 | 0000000000000000     [0000]     |CH22| 0000000000000000     [0000]     |');
   WriteLn('|CH06 | 0000000000000000     [0000]     |CH23| 0000000000000000     [0000]     |');
   WriteLn('|CH07 | 0000000000000000     [0000]     |CH24| 0000000000000000     [0000]     |');
   WriteLn('|CH08 | 0000000000000000     [0000]     |CH25| 0000000000000000     [0000]     |');
```

```
   WriteLn('|CH09 | 0000000000000000     [0000]     |CH26| 0000000000000000     [0000]     |');
   WriteLn('|CH10 | 0000000000000000     [0000]     |CH27| 0000000000000000     [0000]     |');
   WriteLn('|CH11 | 0000000000000000     [0000]     |CH28| 0000000000000000     [0000]     |');
   WriteLn('|CH12 | 0000000000000000     [0000]     |CH29| 0000000000000000     [0000]     |');
   WriteLn('|CH13 | 0000000000000000     [0000]     |CH30| 0000000000000000     [0000]     |');
   WriteLn('|CH14 | 0000000000000000     [0000]     |CH31| 0000000000000000     [0000]     |');
   WriteLn('|CH15 | 0000000000000000     [0000]     |CH32| 0000000000000000     [0000]     |');
   WriteLn('|CH16 | 0000000000000000     [0000]     |CH33| 0000000000000000     [0000]     |');
   WriteLn('+------+-----------------------------------+----+-----------------------------------+');
   WriteLn ('Programmable Controller and DataLogger System');
   WriteLn ('Instrumentation Research  KMITL     June 1988');
   Write ('I/O Monitoring');
end;

Procedure CreatTable2;
begin
   Clrscr;
   WriteLn('+------------------------ MONITORING TABLE 02 ------------------------+');
   WriteLn('|                                                                     |');
   WriteLn('+--------+-----------------------------+----+--------+----------------+');
   WriteLn('|TIMER  00 | 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|TIMER  01 | 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|TIMER  02 | 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|TIMER  03 | 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|TIMER  04 | 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|TIMER  05 | 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|TIMER  06 | 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|TIMER  07 | 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|TIMER  08 | 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|TIMER  09 | 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|TIMER  10 | 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|TIMER  11 | 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|TIMER  12 | 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|TIMER  13 | 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|TIMER  14 | 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|TIMER  15 | 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|TIMER  16 | 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('+--------+-----------------------------+----+--------+----------------+');
   WriteLn ('Programmable Controller and DataLogger System');
   WriteLn ('Instrumentation Research  KMITL     June 1988');
   Write ('I/O Monitoring');
end;

Procedure CreatTable3;
begin
   Clrscr;
   WriteLn('+------------------------ MONITORING TABLE 03 ------------------------+');
   WriteLn('|                                                                     |');
   WriteLn('+---------+----------------------------+----+--------+----------------+');
   WriteLn('|COUNTER 00| 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|COUNTER 01| 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|COUNTER 02| 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|COUNTER 03| 0000000000000000  [0000]  | OFF |  [0000]  |              |');
   WriteLn('|COUNTER 04| 0000000000000000  [0000]  | OFF |  [0000]  |              |');
```

```
    WriteLn('|COUNTER 05| 0000000000000000  [0000]  | OFF |  [0000]  |                    |');
    WriteLn('|COUNTER 06| 0000000000000000  [0000]  | OFF |  [0000]  |                    |');
    WriteLn('|COUNTER 07| 0000000000000000  [0000]  | OFF |  [0000]  |                    |');
    WriteLn('|COUNTER 08| 0000000000000000  [0000]  | OFF |  [0000]  |                    |');
    WriteLn('|COUNTER 09| 0000000000000000  [0000]  | OFF |  [0000]  |                    |');
    WriteLn('|COUNTER 10| 0000000000000000  [0000]  | OFF |  [0000]  |                    |');
    WriteLn('|COUNTER 11| 0000000000000000  [0000]  | OFF |  [0000]  |                    |');
    WriteLn('|COUNTER 12| 0000000000000000  [0000]  | OFF |  [0000]  |                    |');
    WriteLn('|COUNTER 13| 0000000000000000  [0000]  | OFF |  [0000]  |                    |');
    WriteLn('|COUNTER 14| 0000000000000000  [0000]  | OFF |  [0000]  |                    |');
    WriteLn('|COUNTER 15| 0000000000000000  [0000]  | OFF |  [0000]  |                    |');
    WriteLn('|COUNTER 16| 0000000000000000  [0000]  | OFF |  [0000]  |                    |');
    WriteLn('+----------+----------------------------+-----+----------+--------------------+');
    WriteLn ('Programmable Controller and DataLogger System');
    WriteLn ('Instrumentation Research  KMITL     June 1988');
    Write ('I/O Monitoring');
end;

Procedure WriteTable1;
begin
    IOCHBINSTR := IOCHBIN[3]+IOCHBIN[2]+IOCHBIN[1]+IOCHBIN[0];
    gotoxy (9,4+i); Write (IOCHBINSTR);
    IOCHHEXSTR := IOCHHEX[3] + IOCHHEX[2] +IOCHHEX[1] +IOCHHEX[0];
    gotoxy (31,4+i); Write (IOCHHEXStr);
end;

Procedure WriteTable2;
begin
    IOCHBINSTR := IOCHBIN[3]+IOCHBIN[2]+IOCHBIN[1]+IOCHBIN[0];
    gotoxy (48,4+i-17); Write (IOCHBINSTR);
    IOCHHEXSTR := IOCHHEX[3] + IOCHHEX[2] +IOCHHEX[1] +IOCHHEX[0];
    gotoxy (70,4+i-17); Write (IOCHHEXStr);
end;

Procedure WriteTIMTable;
begin
    IOCHBINSTR := IOCHBIN[3]+IOCHBIN[2]+IOCHBIN[1]+IOCHBIN[0];
    gotoxy (14,4+L); Write (IOCHBINSTR);
    IOCHHEXSTR := IOCHHEX[3] + IOCHHEX[2] +IOCHHEX[1] +IOCHHEX[0];
    gotoxy (33,4+L); Write (IOCHHEXStr);
end;

Procedure StatusTC (Status : Boolean);
begin
    gotoxy (42,4+I);
    If Status = true then Write ('ON ') else Write ('OFF');
end;

Procedure HEXBIN (IOCHWord : Integer);
begin
    for J := 0 to 3 do
    begin
        IOCHWordTemp :=  IOCHWord SHR (J * 4);
        IOCHWordTemp :=  IOCHWordTemp AND $000F;
```

```
      Case IOCHWordTemp of
      0 : IOCHBIN[j] := '0000';
      1 : IOCHBIN[j] := '0001';
      2 : IOCHBIN[j] := '0010';
      3 : IOCHBIN[j] := '0011';
      4 : IOCHBIN[j] := '0100';
      5 : IOCHBIN[j] := '0101';
      6 : IOCHBIN[j] := '0110';
      7 : IOCHBIN[j] := '0111';
      8 : IOCHBIN[j] := '1000';
      9 : IOCHBIN[j] := '1001';
     10 : IOCHBIN[j] := '1010';
     11 : IOCHBIN[j] := '1011';
     12 : IOCHBIN[j] := '1100';
     13 : IOCHBIN[j] := '1101';
     14 : IOCHBIN[j] := '1110';
     15 : IOCHBIN[j] := '1111';
      end;
      If IOCHWordTemp <= 9 then IOCHHEX[j] := Chr (IOCHWordTemp + $30) else
      IOCHHEX[j] := Chr (IOCHWordTemp + 55);
   end;
end;

Procedure ReceivePort;
begin
   repeat
   Until (Port[RXCheck] AND $01 = $01) OR KeyPressed;
   Data := Port[TXRX];
end;

Procedure Receive_Data (Address : Byte);
begin
   i := Address;
   j := i * 2;
   Data := j;
   Transmit;
   ReceivePort;
   DataLO := Data;
   Data := j + 1;
   Transmit;
   ReceivePort;
   DataHI := Data;
   Data := $FF;
   Transmit;
end;

Procedure Receive_Byte (ByteAddr : Byte);
begin
   Data := ByteAddr;
   Transmit;
   ReceivePort;
   DataByte := Data;
   Data := $FF;
   Transmit;
```

```
end;

Procedure TIMMonitoring (StartCH,EndCH : Byte);
begin
  L := StartCH;
  Repeat
      Receive_Data (34+L);
      i := L;
      TIMCHDisplay [i] := DataHI AND $7F;
      TIMCHDisplay [i] := TIMCHDisplay [i] SHL 8;
      TIMCHDisplay [i] := TIMCHDisplay [i] OR DataLO;
      WordIO := TIMCHDisplay[i];
      HEXBIN (WordIO);
      WriteTIMTable;
{      Receive_Byte ((68*2)+L);
       TIMStatus[i] := DataByte;
       If TIMStatus[i] = $00 then StatusTC (False) else StatusTC (true);
}     Inc (L);
   Until (L > EndCH) OR Keypressed;
   Data := $FF;
   Transmit;
end;

Procedure CNTMonitoring (StartCH,EndCH : Byte);
begin
   L := StartCH;
   Repeat
      Receive_Data (51+L);
      i := L;
      CNTCHDisplay [i] := DataHI;
      CNTCHDisplay [i] := CNTCHDisplay [i] SHL 8;
      CNTCHDisplay [i] := CNTCHDisplay [i] OR DataLO;
      WordIO := CNTCHDisplay[i];
      HEXBIN (WordIO);
      WriteTIMTable;
      Inc (L);
   Until (L > EndCH) OR KeyPressed;
   Data := $FF;
   Transmit;
end;

Procedure IOMonitoring (StartCH,EndCH : Byte);
begin
     L := StartCH;
     repeat
         Receive_data (L);
         IOCHDisplay [i] := DataHI;
         IOCHDisplay [i] := IOCHDisplay [i] SHL 8;
         IOCHDisplay [i] := IOCHDisplay [i] OR DataLO;
         WordIO := IOCHDisplay[i];
         HEXBIN (WordIO);
         If i > 16 then WriteTable2 else WriteTable1;
         Inc(L);
     until (L > EndCH) or KeyPressed;
```

```
     Data := $FF;
     Transmit;
end;

Procedure MonitoringTable1;
begin
    CreatTable1;
    repeat
       IOMonitoring(0,33);
    until KeyPressed;
    Ch := ReadKey;
end;

Procedure MonitoringTable2;
begin
    CreatTable2;
    repeat
       TIMMonitoring(00,16);
    until KeyPressed;
    Ch := ReadKey;
end;

Procedure MonitoringTable3;
begin
    CreatTable3;
    repeat
       CNTMonitoring(00,16);
    until KeyPressed;
    Ch := ReadKey;
end;

Procedure Monitoring;
Var Table  : Byte;
begin
  Cursor (False);
  Table := 1; Ch := '0';
  While Upcase(Ch) <> 'Q' do
  begin
     Case Table of
     1: MonitoringTable1;
     2: MonitoringTable2;
     3: MonitoringTable3;
     end; {Case}
     Inc (Table);
     If Table = 4 then Table := 1;
  end;
  Cursor (True);
  Data := $FF;
  Transmit;
end;
```

```
Program : Ladder Main Program
Programmer : Viriya Kongratana

unit LadMain;

interface

Uses Crt,Dos,Printer;

procedure MainEditor;

implementation

Const Out_Pattern          = '-( )-:';
      Out_Not_Pattern      = '-(/)-:';
      Relay_Pattern        = '-] [-:';
      Relay_Not_Pattern    = '-]/[-:';
      Timer_Pattern        = 'TIM:';
      Counter_Pattern      = 'CNT:';
      Esc                  = #27;
      F1                   = #59;
      F2                   = #60;
      F3                   = #61;
      F4                   = #62;
      F5                   = #63;
      F6                   = #64;
      F7                   = #65;
      F8                   = #66;
      F9                   = #67;
      F10                  = #68;
      ClearChr             = #00;
      Return               = #13;
      EndKey               = #79;
      InsKey               = #82;
      DelKey               = #83;
      Up                   = #72;
      Down                 = #80;
      Left                 = #75;
      Right                = #77;
      PgUp                 = #73;
      PgDw                 = #81;
      Home                 = #71;
      InstPage             = #79;
      Blue                 = 1;
      White                = 15;
      MaxChr               = 700;
      PageChar             = 3680;
      MaxDiagramPage       = 20; {80}
```

```
      OFF                   = False;
      ON                    = True;

Type  String6               =  String[6];
      String3               =  String[3];
      String4               =  String[4];
      PageLadder            = ^PageLadderPointer;
      PageLadderPointer = record
                            LadderPage : array [1..PageChar]   of Byte;
                          end;
      DiagramLadder     = array [1..MaxDiagramPage] of PageLadder;
      RegPack           =  Record case Integer of
                             1 : (AX,BX,CX,DX,BP,SI,DI,DS,ES,Flags : Integer);
                             2 : (AL,AH,BL,BH,CL,CH,DL,DH : Byte);
                           end;

      FileType          = File of PageLadderPointer;
Var   Regs              : RegPack;
      Ch                : Char;
      Count             : integer;
      Code              : integer;
      Cursor_Ch         : Byte;
      Cursor_Ch1        : Byte;
      XCor              : Byte;
      YCor              : Byte;
      XCor_Temp         : Byte;
      XCor_Temp1        : Byte;
      YCor_Temp         : Byte;
      YCor_Temp1        : Byte;
      Dash              : Byte;
      IOassign_len      : Byte;
      i,j,k             : Byte;
      Page              : Byte;
      Relay_Scan        : Byte;
      PageSwitch        : Byte;
      InstConfig        : Byte;
      ScreenMem         : Integer Absolute $B800:$0000;

      BlockLine         : array [1..MaxChr] of Byte;
      BlockChar         : array [1..MaxChr] of Byte;
      LadderDiagram     : DiagramLadder;
      Dir               : Char;
      DiagramPage       : Byte;
      SizeBlock         : Byte;
      LineBlock         : Byte;
      HighBlock         : Byte;
      DelFlag           : Byte;
      PCFunction        : Byte;
      IOassign          : String6;
      TimCnt_dta        : String6;
      Block_Name        : String4;
      Data1             : String6;
      Data2             : String6;
      Data3             : String6;
```

```
      Data4           : String6;
      FileName        : String[25];
      FileNameDest    : String[25];
      FileVar         : FileType;

procedure PageUp; forward;

procedure PageDown; forward;

procedure PushWindow (Size,Line : Byte); forward;

Procedure CancelDelBlock_Block (X,Y,Block_Size,j : Byte); forward;

{$I d:Scrnfn.pas}
{$I d:OpenFile.pas}
{$I d:LadMenu.pas}
{$I d:EditFun.pas}
{$I d:comp1.pas}
{$I d:Laddisp.pas}
{$I d:LadPrn.pas}
{$I d:LadTx.pas}
{$I d:ladMon.pas}
{$I d:Editor.pas}
{$I d:EditMain.pas}

var
  InstNo : word;
begin
  for InstNo := 1 to MaxDiagramPage do
  New(LadderDiagram[InstNo]);
  for InstNo := 0 to MaxInst do
  New(InstructionPack[InstNo]);
end.
```

```
Program : Screen Management
Programmer : Viriya Kongratana


Type  String80   = String[80];
var   BuffScreen : array [0..$0FFF] of byte;
      Scrn       : array [1..4,0..$0FFF] of byte absolute $B000:$8000;
      Attribute  : byte;
      BufStr : string;

const ColorSeg   = $B000;
      ColorOfs   = $8000;
      BuffSeg    = $B000;
      BuffOfs    = $9000;

procedure GetColor (var Cursor_ch : byte);

{ Procedure to get color at current cursor Position  }

begin { GetColor }
  Regs.AH := $08; { Read Attribute Function }
  Regs.BH := $00; { Active Screen Number }
  Intr ($10,Dos.Registers(Regs));
  Attribute := Regs.AH;  { Attribute byte in AH }
  Cursor_ch := Regs.AL;
end { GetColor };

procedure SetColor (Foreground,Background : integer);
begin
  case Foreground of
    Black        : Attribute := $00;
    Blue         : Attribute := $01;
    Green        : Attribute := $02;
    Cyan         : Attribute := $03;
    Red          : Attribute := $04;
    Magenta      : Attribute := $05;
    Brown        : Attribute := $06;
    LightGray    : Attribute := $07;
    DarkGray     : Attribute := $08;
    LightBlue    : Attribute := $09;
    LightGreen   : Attribute := $0A;
    LightCyan    : Attribute := $0B;
    LightRed     : Attribute := $0C;
    LightMagenta : Attribute := $0D;
    Yellow       : Attribute := $0E;
    White        : Attribute := $0F;
  end { case };
  case Background of
```

```
    Black          : Attribute := Attribute or $00;
    Blue           : Attribute := Attribute or $10;
    Green          : Attribute := Attribute or $20;
    Cyan           : Attribute := Attribute or $30;
    Red            : Attribute := Attribute or $40;
    Magenta        : Attribute := Attribute or $50;
    Brown          : Attribute := Attribute or $60;
    LightGray      : Attribute := Attribute or $70;
    DarkGray       : Attribute := Attribute or $80;
    LightBlue      : Attribute := Attribute or $90;
    LightGreen     : Attribute := Attribute or $A0;
    LightCyan      : Attribute := Attribute or $B0;
    LightRed       : Attribute := Attribute or $C0;
    LightMagenta   : Attribute := Attribute or $D0;
    Yellow         : Attribute := Attribute or $E0;
    White          : Attribute := Attribute or $F0;
  end { case };
end { SetColor };

procedure DirectWrite (S,X,Y : byte; str : string80);
var
  I,J,Color  : Integer;

procedure WriteScreen(S,C,R : Integer; Str : String80);
                    {R=row C=column S=Screen Str=String}
var
  I,J,len,count : Integer;
begin
  J :=((R-1)*160) + ((C-1)*2);       { compute starting location }
  len := Length(Str);
  for count := 1 to len do
  begin
    if S = 0 then
    begin
      BufStr := Copy(Str,count,1);
      Mem[Seg(BuffScreen):Ofs(BuffScreen) + J]    := Ord(BufStr[1]);
      Mem[Seg(BuffScreen):Ofs(BuffScreen)+J+1]    := Attribute;
    end
    else
    begin
      BufStr := Copy(Str,count,1);
      Mem[Seg(Scrn[S]):Ofs(Scrn[S]) + J]    := Ord(BufStr[1]);
      Mem[Seg(Scrn[S]):Ofs(Scrn[S])+J+1]    := Attribute;
    end;
    J := J+2;
  end;
end;

begin
  WriteScreen (S,X,Y,str);
end { DirectWrite };
Procedure SwapScreen (Source,Dest : byte);
begin
```

```
    Move (Scrn[Source],Scrn[4],$1000);
    Move (Scrn[Dest],Scrn[Source],$1000);
    Move (Scrn[4],Scrn[Dest],$1000);
end;

procedure SwitchScreen (Source,Dest:byte);
begin { SwitchScreen }
  if Source = 0 then
    Move(BuffScreen,Scrn[Dest],$1000);
  if Dest = 0 then
    Move(Scrn[Source],BuffScreen,$1000);
  if (Source > 0) and (Dest > 0) then
    Move(Scrn[Source],Scrn[Dest],$1000);
end { SwitchScreen };

procedure Cursor (Cursortype : boolean);
begin { Cursor }
  if Cursortype = true then
    Regs.CX := $0707  { cursor on }
  else
    Regs.CH := $20; { cursor off }
  Regs.AH := $01;
  Intr($10,Dos.Registers(Regs));
end { Cursor };

procedure ClearScreen (ScreenNumber : byte);
var LineCount : byte;
const SpaceString : String[80] =
'                                                                              ';
begin { ClearScreen }
  for LineCount := 1 to 25 do
    DirectWrite (ScreenNumber,1,LineCount,SpaceString);
end { ClearScreen };

procedure ActiveScreen (ScreenNumber : byte);
begin { ActiveScreen }
  if ScreenNumber in [1..4] then
  begin
    Regs.AH := $05;
    Regs.AL := ScreenNumber - 1;
    Intr ($10,Dos.Registers(Regs));
  end;
end { ActiveScreen };
```

Program : Ladder Menu Management
Programmer : Viriya Kongratana

```
Procedure  MainMenu;
begin
  Clrscr;
  Cursor (Off);
  TextBackGround (DarkGray);
  TextColor (White);
  writeln ('+-------------------------------+');
  writeln ('| Programmable Controller       |');
  writeln ('| Data Logger System            |');
  writeln ('|                               |');
  writeln ('|                               |');
  writeln ('|                               |');
  writeln ('| (C)CopyRight 1988   KMITL.    |');
  writeln ('+-------------------------------+');
  DiagramPage := 1;
  GotoXY(50,25);
  write ('Press any key to continue....');
  Repeat
  Until Keypressed; Ch := ReadKey;
  Cursor (On);
end;
```

Program : Function Editor
Programmer : Viriya Kongratana

```
{////////////////////////////////////////////////////////////////////////////}
{//                                                                        //}
{//                          EDITOR FUNCTION                               //}
{//                                                                        //}
{////////////////////////////////////////////////////////////////////////////}
Procedure Read_Cursor;
begin
   XCor := WhereX;
   YCor := WhereY;
end;

Procedure PushCursor;
begin
  Read_Cursor;
  XCor_Temp := XCor;
  YCor_Temp := YCor;
end;

Procedure PopCursor;
begin
  XCor := XCor_Temp;
  YCor := YCor_Temp;
  gotoxy (XCor,YCor);
end;

Procedure PushCursor1;
begin
  Read_Cursor;
  XCor_Temp1 := XCor;
  YCor_Temp1 := YCor;
end;

Procedure PopCursor1;
begin
  XCor := XCor_Temp1;
  YCor := YCor_Temp1;
  gotoxy (XCor,YCor);
end;

Procedure LinkLeft;
begin
  gotoxy (XCor,YCor);
  GetColor (Cursor_ch);
  while Cursor_ch = $20 do
```

```
  begin
    write ('-');XCor := XCor - 1; gotoxy (XCor,YCor); GetColor (Cursor_ch);
  end;
end;

Procedure LinkLeft_Check;  {Check Left Direction}
begin
  Read_Cursor;
  PushCursor;
  LinkLeft;
  PopCursor;
  gotoxy (XCor,YCor);
  GetColor (Cursor_Ch);
  Case Cursor_Ch of
  $C3 : gotoxy (XCor,YCor);
  $C4 : gotoxy (XCor,YCor);
  $D9 : write ('+');
  $B4 : write ('ล');
  $C2 : write ('+');
  $C1 : write ('+');
  $B3 : write ('+');
  end; {case}
end;

Procedure Link (Link_line : Byte);
begin
  for Dash := 1 to Link_line do
  begin
    Read_Cursor;
    gotoxy (XCor,YCor);
    write ('-');
  end;
end;

Procedure Linkup_check;
begin
  Repeat
      Read_Cursor;
      XCor := XCor - 1;
      YCor := YCor - 1;
      gotoxy (XCor,YCor);
      GetColor (Cursor_ch);
      Case Cursor_ch of
        $00 : write ('|');
        $20 : write ('|');
        $C4 : write ('+');
        $C1 : write ('ล');
        $D9 : write ('+');
        $C2 : write ('+');
        $B3 : begin
                write ('|');
                Cursor_Ch := $20;
              end;
      end; {case}
```

```
  Until (Cursor_ch > $20) OR (YCor = 3);
end;

Procedure Link_Up;
begin
  Read_Cursor;
  If YCor <> 3 then
  begin
      GetColor (Cursor_Ch);
      Case Cursor_Ch of
      $00 : write ('+');
      $20 : write ('+');
      $C4 : write ('+');
      $C2 : write ('ã');
      $C1 : write ('+');
      $D9 : write ('+');
      $B4 : write ('+');
      $C5 : write ('ã');
      end;
      Read_Cursor;
      XCor_Temp := XCor - 1;
      YCor_Temp := YCor;
      Linkup_check;
      gotoxy (XCor_Temp,YCor_Temp);
  end;
end;

Procedure Link_End (End_Cursor : Byte);
begin
  Dash := End_Cursor - XCor;
  gotoxy (XCor,YCor);
  Link (Dash);
end;

Procedure Dash_Line;
begin
    Read_Cursor;
    If XCor <= 79 then
    begin
      LinkLeft_Check;
      Read_Cursor;
      gotoxy (XCor,YCor);
      GetColor (Cursor_ch);
      Case Cursor_ch of
      $B3 : write ('+');
      $B4 : write ('+');
      $C0 : write ('+');
      $C3 : write ('ã');
      end; {case}
      for i := 1 to 8 do
      write ('-');
    end;
end;
```

```
\////////////////////////////////////////////////////////////////////////////
//                                                                          //
//                          END EDITOR FUNCTION                             //
//                                                                          //
////////////////////////////////////////////////////////////////////////////\
```

Program : Ladder Compilation
Programmer : Viriya Kongratana

```
{//////////////////////////////////////////////////////////////////////////
/                                                                        /
/              Ladder Diagram Compilation  20-3-88                       /
/                                                                        /
//////////////////////////////////////////////////////////////////////////}
Const NC                        = 'C';
      NO                        = 'O';
      Junction                  = '.';
      NodeBlock                 = 'B';
      _Up                       = 00;
      _Down                     = 01;
      _Left                     = 02;
      _Right                    = 03;
      Ld_Inst                   = 00;
      And_Inst                  = 01;
      Or_inst                   = 02;
      _Set                      = 01;
      _Reset                    = 00;
      MaxInst                   = 2000;
      _LD                       = 00;
      _LD_NOT                   = 01;
      _AND                      = 02;
      _AND_NOT                  = 03;
      _OR                       = 04;
      _OR_NOT                   = 05;
      _AND_LD                   = 06;
      _OR_LD                    = 07;
      _OUT                      = 08;
      _OUT_NOT                  = 09;
      _TIM                      = 10;
      _CNT                      = 11;
      _LD_TIM                   = 12;
      _LD_NOT_TIM               = 13;
      _LD_CNT                   = 14;
      _LD_NOT_CNT               = 15;
      _AND_TIM                  = 16;
      _AND_NOT_TIM              = 17;
      _AND_CNT                  = 18;
      _AND_NOT_CNT              = 19;
      _OR_TIM                   = 20;
      _OR_NOT_TIM               = 21;
      _OR_CNT                   = 22;
      _OR_NOT_CNT               = 23;
      {FUN Command bit 7 is Set}
      _NOP                      = 128;      {FUN 00}
```

```
    _END                = 129;      {FUN 01}
    _IL                 = 130;
    _ILC                = 131;
    _JMP                = 132;
    _JME                = 133;
    _SFT                = 138;      {FUN 10}
    _KEEP               = 139;
    _CNTR               = 140;
    _DIFU               = 141;
    _DIFD               = 142;
    _TIMH               = 143;
    _WSFT               = 144;
    _CMP                = 148;
    _MOV                = 149;
    _MVN                = 150;
    _BIN                = 151;
    _BCD                = 152;
    _ASL                = 153;
    _ASR                = 154;
    _ROL                = 155;
    _ROR                = 156;
    _COM                = 157;
    _ADD                = 158;
    _SUB                = 159;
    _MUL                = 160;
    _DIV                = 161;
    _ANDW               = 162;
    _ORW                = 163;
    _XORW               = 164;
    _XNRW               = 165;
    _INC                = 166;
    _DEC                = 167;
    _STC                = 168;
    _CLC                = 169;
    _FUN70              = 198;
    _FUN71              = 199;
    _FUN72              = 200;
    _FUN73              = 201;
    _FUN74              = 202;
    _FUN75              = 203;
    _FUN76              = 204;
    _FUN77              = 205;
    _FUN78              = 206;
    _FUN79              = 207;
    _FUN80              = 208;
    _FUN81              = 209;
    _FUN82              = 210;
    _FUN83              = 211;
    _FUN84              = 212;
    _FUN85              = 213;
    _FUN94              = 222;
    _FUN95              = 227;
    _FUN                = 255;
```

```
Type NodeRow              = array [1..11] of byte;
     NodeDiagram          = array [1..11] of NodeRow;
     ComponentRow         = array [1..11] of Char;
     ComponentDiagram     = array [1..11] of ComponentRow;
     AssignRow            = array [1..11] of String[6];
     AssignDiagram        = array [1..11] of AssignRow;
     String7              = String[7];

     Stack_ORLD           = array [0..10] of record
                                               CountNode_Stack : byte;
                                               CountJunc_Stack : byte;
                                               end;

     Stack_Node           = array [0..10] of record
                                             NodePosRow  : byte;
                                             NodePosLine : byte;
                                             end;

     PackInstructionP     =  Record
                                             Address         : Integer;
                                             Instruction     : String[7];
                                             OpcodeInst      : Byte;
                                             AssignComponent : String[6];
                                             IOAssignInst    : Integer;
                                             end;

     PackInstruction      = ^PackInstructionP;

     PackInst             = array [0..MaxInst] of PackInstruction;

     PackCommand          = Record
                              InsAddress  : Integer;
                              Opcode      : Byte;
                              DataType1   : Byte;
                              Assignment1 : Integer;
                              DataType2   : Byte;
                              Assignment2 : Integer;
                              DataType3   : Byte;
                              Assignment3 : Integer;
                            end;


Var  DiagramNode          : NodeDiagram;
     DiagramComponent     : ComponentDiagram;
     DiagramAssign        : AssignDiagram;
     InstructionPack      : PackInst;
     CommandPack          : PackCommand;
     FileDest             : File of PackCommand;
     ORLDStack            : Stack_ORLD;
     NodeStack            : Stack_Node;
     AssignComponent      : String[6];
     AssignTemp           : String[4];
     Instruction          : String[7];
     AssignFunction       : String[2];
```

```
    Result                  : Integer;
    Function_Block          : Integer;
    Address                 : Integer;
    _IOAssign               : Integer;
    _Step                   : Integer;
    Opcode                  : byte;
    Node                    : byte;
    ORLDStack_Point         : byte;
    Component               : Char;
    RowScan,LineScan        : byte;
    X_Pos,Y_Pos             : byte;
    CountNode               : byte;
    CountJunc               : byte;
    CountLeft               : byte;
    Direction               : byte;
    FlagANDLD               : byte;
    FlagORLD                : byte;
    FlagT_ANDLD             : byte;
    FlagX_ANDLD             : byte;
    ANDLD_Flag              : byte;
    FlagLD                  : byte;
    FlagOR                  : byte;


Procedure _WriteLn (_Instruction : String7; Opcode:byte; _AssignIO : String6);
begin
   InstructionPack[Address]^.Address := Address;
   InstructionPack[Address]^.Instruction := _Instruction;
   InstructionPack[Address]^.OpcodeInst := Opcode;
   InstructionPack[Address]^.AssignComponent := _AssignIO;
   AssignTemp := Copy (_AssignIO,1,4);
   Val (AssignTemp,_IOAssign,Result);
   InstructionPack[Address]^.IOAssignInst := _IOAssign;
   Inc (Address);
end;

Procedure CheckLinePosition;
 begin
  If RowScan = 10 then
  begin
    RowScan := 1;
    LineScan := LineScan + 1;
  end;
end;

Procedure Parameter (Var Row,Line : byte);
begin
    Node      := DiagramNode [Row,Line];
    Component := DiagramComponent [Row,Line];
    AssignComponent := DiagramAssign [Row,Line];
end;

Procedure ScanRight;
begin
```

```
    If (RowScan = 1) AND (Component = 'O') then
                                              begin
                                                  Instruction := 'LD';
                                                  Opcode := _LD;
                                                  FlagLD := _Reset;
                                              end;

    If (RowScan = 1) AND (Component = 'C') then
                                             begin
                                                 Instruction := 'LD NOT';
                                                 Opcode := _LD_NOT;
                                                 FlagLD := _Reset;
                                             end;

    If (RowScan = 9) AND (Component = 'O') then
    begin
       Instruction := 'OUT';
       Opcode := _OUT;
       Direction := _Down;
       FlagLD := _Set;
       ANDLD_Flag := 0;
    end;

    If (RowScan = 9) AND (Component = 'C') then
    begin
       Instruction := 'OUT NOT';
       Opcode := _OUT_NOT;
       Direction := _Down;
       FlagLD := _Set;
       ANDLD_Flag := 0;
    end;

    If (RowScan = 9) AND (Component = 'B') then
    begin
       Instruction := 'BLOCK';
       Direction := _Down;
       FlagLD := _Set;
       ANDLD_Flag := 0;
    end;

    If (RowScan <> 1) AND (RowScan <> 9) AND (Component = 'O') then
        If FlagLD = _Set then
                          begin
                            Instruction := 'LD';
                            Opcode := _LD;
                            FlagLD := _Reset;
                          end
                          else
                          begin
                            Instruction := 'AND';
                            Opcode := _AND;
                          end;

    If (RowScan <> 1) AND (RowScan <> 9) AND (Component = 'C') then
```

```
        If FlagLD = _Set then
                              begin
                                Instruction := 'LD NOT';
                                Opcode := _LD_NOT;
                                FlagLD := _Reset;
                              end
                              else
                              begin
                                 Instruction := 'AND NOT';
                                 Opcode := _AND_NOT;
                              end;

    If Component = 'B'  then _Writeln (AssignComponent,Opcode,Instruction) else
    If Component <> '.' then _Writeln (Instruction,Opcode,AssignComponent);
end;

Procedure ScanLeft;
begin
    If  Component = 'O' then
    begin
       Instruction := 'OR';
       Opcode := _OR;
    end;
    If  Component = 'C' then
    begin
       Instruction := 'OR NOT';
       Opcode := _OR_NOT;
    end;
    If  Component <> '.' then _Writeln (Instruction,Opcode,AssignComponent);
end;

Procedure ScanLeftOR (Count : byte);
begin
  for CountLeft := 1 to Count do
  begin
      Parameter (RowScan,LineScan);
      ScanLeft;
      DiagramNode [RowScan,LineScan] := $00;
      RowScan := RowScan + 1;
  end;
      RowScan := RowScan - 1;
end;

Procedure PushORLDStack;
begin
   ORLDStack [ORLDStack_Point].CountNode_Stack := CountNode;
   ORLDStack [ORLDStack_Point].CountJunc_Stack := CountJunc;
   ORLDStack_Point :=  ORLDStack_Point + 1;
end;

Procedure PopORLDStack;
begin
   ORLDStack_Point := ORLDStack_Point - 1;
   CountNode :=  ORLDStack [ORLDStack_Point].CountNode_Stack;
```

```
   CountJunc :=  ORLDStack [ORLDStack_Point].CountJunc_Stack;
end;

Procedure Instruction_Compile;
begin
   LineScan  := 1;
   RowScan   := 1;
   Direction := _Right;
   FlagLD    := _set;
   FlagANDLD := _Reset;
   FlagORLD  := _Reset;
   FlagT_ANDLD := _Reset;
   FlagX_ANDLD := _Reset;
   ANDLD_Flag:= 0;
   ORLDStack_Point := 0;

   Repeat
       Parameter (RowScan,LineScan);
       Case Node of

{-}    $C4    : begin
                    ScanRight; Direction := _Right;
                    DiagramNode [RowScan,LineScan] := $00;
                    RowScan := RowScan + 1;
                    CheckLinePosition;
                  end;

{ }    $20,$00    : begin
                    RowScan := RowScan + 1;
                    CheckLinePosition;
                    Direction := _Right;
                 end;

{|}    $B3    : begin
                   If Direction = _Down then LineScan := LineScan + 1
                   else
                   begin
                      DiagramNode [RowScan,LineScan] := $00;
                      LineScan := LineScan - 1;
                   end;
                end;


{+}    $C2    : begin
                    If (ANDLD_Flag = 0) AND (FlagANDLD = _Reset) then
                    begin
                        FlagX_ANDLD := _Reset;
                        FlagT_ANDLD := _Reset;
                        ANDLD_Flag := 0;
                    end;

                    If (ANDLD_Flag > 0) AND (Direction = _Up)
                    AND (FlagLD = _Reset) then
                    begin
```

```
                        Instruction := 'AND LD';
                        OpCode := _AND_LD;
                        AssignComponent := '     ';
                        _IOAssign := 0;
                        _Writeln (Instruction,Opcode,AssignComponent);
                        ANDLD_Flag := ANDLD_Flag - 1;
                        FlagANDLD := _Reset;
                        Direction := _Up;
                     end
                     else
                     begin {########}
                        If Direction =_Right  then
                        begin
                           Direction :=_Down;
                           LineScan := LineScan + 1;
                        end;
                        If Direction = _Up then
                        begin
                            ScanRight;  Direction := _Right;
                            DiagramNode [RowScan,LineScan] := $00;
                            RowScan := RowScan + 1;
                            CheckLinePosition;
                            If (FlagT_ANDLD = _Set)
                            OR (FlagX_ANDLD =_Set) then ANDLD_Flag := ANDLD_Flag + 1;
                        end;
                     end;{########}
                  end;

{+}     $B4    : begin
                     If Direction = _Up then
                     begin
                            DiagramNode [RowScan,LineScan] := $B3;
                            FlagLD := _Reset;
                     end
                     else
                     begin {#####}
                     If Direction = _Down then
                     begin
                         CountNode := 0;
                         CountJunc := 0;
                         Repeat
                             RowScan := RowScan - 1;
                             Parameter (RowScan,LineScan);
                             If Component <> '.' then CountNode := CountNode + 1;
                             If Component  = '.' then CountJunc := CountJunc + 1;
                         Until (Node = $00) OR (RowScan = 1);

                         PushORLDStack;

                         If (CountNode = 1) AND (CountJunc > 0) then
                         begin
                             CountJunc := CountJunc + 1;
                             ScanLeftOR (CountJunc);
                             CountJunc := CountJunc + 1;
```

```
                        Direction := _Right;
                    end
                    else

                    begin {####}
                    If CountNode > 1 then
                    begin
                        Direction := _Right;
                        FlagLD    := _Set;
                    end
                    else
                    begin
                         ScanLeft;
                         DiagramNode [RowScan,LineScan] := $00;
                         RowScan := RowScan + 1;
                         If CountJunc = 0 then
                         begin
                             LineScan := LineScan + 1;
                             Direction := _Down;
                             PopORLDStack;
                         end;
                    end;
                    end;{####}
                end
                else
                begin
                    If Direction = _Right then
                    begin
                        LineScan := LineScan + 1;
                        PopORLDStack;
                        If CountNode > 1 then
                        begin
                            Instruction := 'OR LD';
                            Opcode := _OR_LD;
                            AssignComponent := '     ';
                            _IOAssign := 0;
                            _Writeln (Instruction,Opcode,AssignComponent);
                        end;
                        Direction := _Down;
                        end;
                    end;
                end; {#####}
            end;


{a}     $C5     : begin
                    If Direction = _Up then
                    begin
                            DiagramNode [RowScan,LineScan] := $C4;
                            FlagLD := _set;
                            FlagX_ANDLD := _Set;
                            FlagANDLD  := _Set;
                            LineScan := LineScan - 1;
```

```
                end
                else
                begin {#####}
                If Direction = _Down then
                begin
                    CountNode := 0;
                    CountJunc := 0;
                    Repeat
                        RowScan := RowScan - 1;
                        Parameter (RowScan,LineScan);
                        If Component <> '.' then CountNode := CountNode + 1;
                        If Component  = '.' then CountJunc := CountJunc + 1;
                    Until (Node = $00) OR (RowScan = 1);

                    PushORLDStack;

                    If (CountNode = 1) AND (CountJunc > 0) then
                    begin
                        CountJunc := CountJunc + 1;
                        ScanLeftOR (CountJunc);
                        CountJunc := CountJunc + 1;
                        Direction := _Right;
                    end
                    else

                    begin {####}
                    If CountNode > 1 then
                    begin
                        Direction := _Right;
                        FlagLD    := _Set;
                    end
                    else
                    begin
                         ScanLeft;
                         DiagramNode [RowScan,LineScan] := $00;
                         RowScan := RowScan + 1;
                         If CountJunc = 0 then
                         begin
                             LineScan := LineScan + 1;
                             Direction := _Down;
                             PopORLDStack;
                         end;
                    end;
                    end;{####}
                end
                else
                begin
                    If Direction = _Right then
                    begin
                        LineScan := LineScan + 1;
                        PopORLDStack;
                        If CountNode > 1 then
                        begin
                            Instruction := 'OR LD';
```

```
                                Opcode := _OR_LD;
                                AssignComponent := '     ';
                                _IOAssign := 0;
                                _Writeln (Instruction,Opcode,AssignComponent);
                            end;
                            Direction := _Down;
                            end;
                        end;
                    end; {#####}
                end;

{+}     $C1     : begin
                    If Direction = _Down then
                    begin
                        CountNode := 0;
                        CountJunc := 0;
                        Repeat
                            RowScan := RowScan - 1;
                            Parameter (RowScan,LineScan);
                            If Component <> '.' then CountNode := CountNode + 1
                            else CountJunc := CountJunc + 1;
                        Until (Node = $00) OR (RowScan = 1);

                        PushORLDStack;

                        If (CountNode = 1) AND (CountJunc > 0) then
                        begin
                            CountJunc := CountJunc+1 ;
                            ScanLeftOR (CountJunc);
                            CountJunc := CountJunc-1 ;
                            Direction := _Right;
                        end
                        else
                        begin   {###}
                            If CountNode > 1 then
                            begin
                               Direction := _Right;
                               FlagLD := _Set;
                            end
                            else
                            begin
                                ScanLeft;
                                DiagramNode [RowScan,LineScan] := $00;
                                RowScan := RowScan + 1;
                                If CountJunc = 0 then
                                begin
                                    FlagLD := _Set;
                                    FlagT_ANDLD := _Set;
                                    FlagANDLD := _Set;
                                    If FlagT_ANDLD = _Set then DiagramNode[RowScan,LineScan]
                                    else
                                    DiagramNode[RowScan,LineScan] := $00;
                                    LineScan := LineScan - 1;
                                    Direction := _Up;
```

```
                                    PopORLDStack;
                                end else Direction := _Right;
                            end;
                        end; {###}
                    end
                    else
                    begin
                        If Direction = _Right then
                        begin
                            FlagT_ANDLD := _Set;
                            FlagLD := _Set;
                            FlagANDLD := _Set;
                            DiagramNode[RowScan,LineScan] := $C4;
                            LineScan := LineScan - 1;
                            PopORLDStack;
                            If CountNode > 1 then
                                begin
                                    Instruction := 'OR LD';
                                    Opcode := _OR_LD;
                                    AssignComponent := '     ';
                                    _IOAssign := 0;
                                    _Writeln (Instruction,Opcode,AssignComponent);
                                end;
                            end;
                            Direction := _Up;
                        end;
                    end;

{+}     $D9     : begin
                    If Direction = _Down then
                    begin
                        CountNode := 0;
                        CountJunc := 0;
                        Repeat
                            RowScan := RowScan - 1;
                            Parameter (RowScan,LineScan);
                            If Component <> '.' then CountNode := CountNode + 1
                            else CountJunc := CountJunc + 1;
                            If (Node = $C2) then FlagORLD := _Set;
                        Until (Node = $00) OR (RowScan = 1);

                        PushORLDStack;

                        If (CountNode = 1) AND (FlagORLD = _Set) then
                        begin
                            Direction := _Right;
                            FlagLD := _Set;
                            CountJunc := 0;
                            FlagORLD := _Reset;
                        end
                        else
                        If (CountNode = 1) AND (CountJunc > 0) then
                        begin
```

```
                        CountJunc := CountJunc+1 ;
                        ScanLeftOR (CountJunc);
                        CountJunc := CountJunc-1 ;
                        Direction := _Right;
                    end
                    else
                    begin   {###}
                    If (CountNode > 1) then
                    begin
                       Direction := _Right;
                       FlagLD := _Set;
                    end
                    else
                    begin
                         ScanLeft;
                         DiagramNode [RowScan,LineScan] := $00;
                         RowScan := RowScan + 1;
                         If CountJunc = 0 then
                         begin
                            If FlagANDLD = _Set then DiagramNode[RowScan,LineScan] := $C4
                            else
                            DiagramNode[RowScan,LineScan] := $00;
                            LineScan := LineScan - 1;
                            Direction := _Up;
                            PopORLDStack;
                         end
                         else
                            Direction := _Right;
                    end;
                    end; {###}
                end
                else
                begin
                    If Direction = _Right then
                    begin
                         DiagramNode[RowScan,LineScan] := $00;
                         LineScan := LineScan - 1;
                         PopORLDStack;
                         If CountNode > 1 then
                         begin
                            Instruction := 'OR LD';
                            Opcode := _OR_LD;
                            AssignComponent := '     ';
                            _IOAssign := 0;
                            _Writeln (Instruction,Opcode,AssignComponent);
                         end;
                    end;
                    Direction := _Up;
            end;
        end;

    end; {Case}

Until (LineScan = 12) OR Keypressed;
```

```
end;

Procedure StoreComponent (XPos,YPos : byte);
begin
  DiagramNode [XPos,YPos] := Cursor_Ch;
  If XPos < 9 then
  begin
     If Cursor_Ch IN [$B3,$B4,$D9] then
     begin
     end
     else
     begin
        gotoxy (XCor+4,YCor);
        GetColor (Cursor_Ch);
        case Cursor_Ch of
          $20 : Component := NO;
          $2F : Component := NC;
          $C4 : Component := Junction;
        end; {Case}
        DiagramComponent [XPos,YPos] := Component;
        for i:= 1 to 6 do
        begin
           gotoxy ((XCor+1+i),YCor-1);
           GetColor (Cursor_Ch);
           Insert (Chr(Cursor_Ch),AssignComponent,i);
        end;
        DiagramAssign [XPos,YPos] := AssignComponent;
     end;
  end;
  If XPos = 9 then
  begin
       gotoxy (XCor+2,YCor);
       GetColor (Cursor_Ch);
       If Cursor_Ch = $B4 then
       begin
          Component := NodeBlock;
          DiagramComponent [XPos,YPos] := Component;
          AssignFunction := '';
          for i:= 1 to 6 do
          begin
              gotoxy ((XCor+4+i),YCor);
              GetColor (Cursor_Ch);
              If (Cursor_Ch >= $30) AND (Cursor_Ch <= $39) then
              begin
                  AssignFunction := AssignFunction + Chr(Cursor_Ch);
                  Val (AssignFunction,Function_Block,Result);
              end;
              Insert (Chr(Cursor_Ch),AssignComponent,i);
              DiagramAssign [XPos,YPos] := AssignComponent;
          end;
       end
       else
       begin
           gotoxy (XCor+9,YCor);
```

```
            GetColor (Cursor_Ch);
            case Cursor_Ch of
            $20 : Component := NO;
            $2F : Component := NC;
            end; {Case}
            DiagramComponent [XPos,YPos] := Component;
            for i:= 1 to 6 do
            begin
               gotoxy ((XCor+6+i),YCor-1);
               GetColor (Cursor_Ch);
               Insert (Chr(Cursor_Ch),AssignComponent,i);
            end;
            DiagramAssign [XPos,YPos] := AssignComponent;
        end;
  end; {If}
end;

Procedure WriteStoreNode;
begin
  for Y_Pos := 1 to 11 do
    begin
       for X_Pos := 1 to 9 do
       write (Chr(DiagramNode[X_Pos,Y_Pos]),DiagramComponent[X_Pos,Y_Pos]);
       writeln;
    end;
  for Y_Pos := 1 to 11 do
    begin
       for X_Pos := 1 to 9 do
       write (DiagramAssign [X_Pos,Y_Pos]);
       writeln;
    end;
end;

Procedure ClearArray;
begin
  for Y_Pos := 1 to 11 do
     for X_Pos := 1 to 11 do
     begin
       DiagramNode [X_Pos,Y_Pos] := $20;
       DiagramComponent [X_Pos,Y_Pos] := ' ';
       DiagramAssign [X_Pos,Y_Pos] := '      ';
     end;
end;

Procedure ScanRung;
begin
   Address := 0;
   for Page := 1 to  DiagramPage do
   begin
      ClearArray;
      Move (LadderDiagram[Page]^,ScreenMem,PageChar);
      YCor := 3;
      While YCor <= 22 do
      begin
```

```
          XCor := 2; gotoxy (XCor,YCor);
          GetColor (Cursor_Ch);
          If (Cursor_Ch IN[$20,$00]) AND (XCor = 2) then YCor := YCor + 1;
          for Relay_Scan := 1 to 9 do
          begin
               gotoxy (Xcor,YCor);
               GetColor (Cursor_Ch);
               PushCursor;
               RowScan := YCor DIV 2;
               If Cursor_Ch >= $B3 then StoreComponent (Relay_Scan,RowScan);
               PopCursor;
               XCor := ((XCor DIV 8)*8)+2;
               XCor := XCor + 8;
          end;
          YCor := YCor + 1;
       end;
       Instruction_Compile;
    end;
end;

Procedure SaveDataLoader;
begin
    for _Step := 0 to MaxInst do
    begin
      CommandPack.InsAddress := InstructionPack[_Step]^.Address;
      CommandPack.Opcode     := InstructionPack[_Step]^.OpcodeInst;
      CommandPack.DataType1  := 00;
      CommandPack.Assignment1:= InstructionPack[_Step]^.IOAssignInst;
      CommandPack.DataType2  := 00;
      CommandPack.Assignment2:= 00;
      CommandPack.DataType3  := 00;
      CommandPack.Assignment3:= 00;
      Write (FileDest,CommandPack);
    end;
    Close (FileDest);
end;

Procedure ClearInstruction;
begin
  for _Step := 0 to MaxInst do
  begin
       InstructionPack[_Step]^.Instruction := 'NOP    ';
       InstructionPack[_Step]^.OpcodeInst := _NOP;
       InstructionPack[_Step]^.AssignComponent := '      ';
  end;
end;

Procedure Compile ;
begin
    PageUp;
    ClearInstruction;
    ScanRung;
    Write (Chr($07));
    TextColor (LightGray + Blink);
```

```
   gotoxy (50,24); Write ('Instruction Created');
   Assign (FileDest,FileNameDest);
   Rewrite (FileDest);
   SaveDataLoader;
   repeat
   until KeyPressed;
   TextColor (LightGray);
   gotoxy (50,24); Write ('                    ');
end;
```

```
Text File List Processing Program V4.00C Page : 75
File : A:LADPRN.PAS
Current Date : Tuesday September 6, 1988
Current Time : 3:11 PM

Program : Ladder Hardcopy
Programmer : Viriya kongratana

Const Space      = #32;
Var   PrintChar  : Integer;
      PrintLine  : byte;
      Character  : Char;

Procedure SeparatePrint;
begin
    PrintChar := 159;
    PrintLine := 80;
    While PrintChar <= PageChar do
    begin
        If PrintLine = 80 then
        begin
           Character := Chr(LadderDiagram[i]^.LadderPage[PrintChar]);
           If ORD(Character) = $00 then Character := Space;
           Writeln (LST);
           PrintLine := 1;
           PrintChar := PrintChar + 1;
           PrintChar := PrintChar + 1;
        end;
        Character := Chr(LadderDiagram[i]^.LadderPage[PrintChar]);
        If ORD(Character) = $00 then Character := Space;
        write (LST,Character);
        PrintChar := PrintChar + 1;
        PrintChar := PrintChar + 1;
        PrintLine := PrintLine + 1;
    end;
end;

Procedure PrintWindow;
begin
  PushCursor1;
  TextBackGround (LightRed);
  TextColor (LightCyan);
  gotoxy (XCor,YCor);    write ('+------- Print Ladder File--------+');
  gotoxy (XCor,YCor+1);  write ('|                                 |');
  gotoxy (XCor,YCor+2);  write ('+---------------------------------+');
  PopCursor1; gotoxy (XCor,YCor);
end;

Procedure InstructionListing;
Var StepPrn : Integer;
begin
   StepPrn := 0;
   While InstructionPack[StepPrn]^.OpcodeInst <> _NOP  do
```

```
    begin
        IntegerString (StepPrn,StepString);
        Write   (LST,StepString,'      ');
        Write   (LST,InstructionPack[StepPrn]^.Instruction);
        i := Length (InstructionPack[StepPrn]^.Instruction);
        WriteLn (LST,InstructionPack[StepPrn]^.AssignComponent:40-i);
        INC (StepPrn);
    end;
end;

Procedure PrintLadderDiagram;
begin
    SizeBlock := 36; LineBlock := 3;
    PushWindow (SizeBlock,LineBlock);
    PrintWindow;
    gotoxy (XCor+2,YCor+1);
    write ('FileName:',FileName);
    Repeat
    Until Keypressed;
    Ch := ReadKey;
    If Ch <> Return then
    begin
       gotoxy (XCor+2,YCor+1);
       write ('FileName:','                    ');
       gotoxy (XCor+2,YCor+1);
       write ('FileName:');
       read (FileName);
       Cursor (Off);
    end;
    assign (FileVar,FileName);
    reset (FileVar);

    WriteLn (LST,'Programmable Controller and DataLogger System');
    WriteLn (LST,'Instrumentation Research KMITL      June 1988');
    WriteLn (LST,'Ladder Diagram Listing');
    WriteLn (LST);
    WriteLn (LST);
    for i := 1 to DiagramPage do
    begin
        Read (FileVar,LadderDiagram[i]^);
        SeparatePrint;
    end;
    close (FileVar);
    writeln (Lst);
    gotoxy (XCor,YCor);
    SizeBlock := 36; LineBlock := 3;
    CancelDelBlock_Block (XCor,(YCor+1),SizeBlock,(LineBlock));
    Cursor (On);
    Compile;
    WriteLn (LST,'Instruction Listing');
    WriteLn (LST);
    InstructionListing;
end;
```

File : A:LADTX.PAS
Current Date : Tuesday September 6, 1988
Current Time : 3:12 PM

Program : Ladder Instruction Transfer
Programmer : Viriya Kongratana

```
Procedure Transmitinst;
begin
end;
```

```
Program : Instruction Listing
Programmer : Viriya Kongratana

Var  StepString : String4;
Procedure IntegerString (_Integer : Integer ; Var _String : String4);
Var i : Integer;
begin
   Str (_Integer,_String);
   i := Length (_String);
   Case i of
   1 : _String := '000' + _String;
   2 : _String := '00'  + _String;
   3 : _String := '0'   + _String;
   end; {case}
end;

Procedure GetInstruction (Start_Step,End_Step : Integer);
begin
    for i := 0 to 33 do
    begin
      If i <= 16 then
      begin
          IntegerString (Start_Step + i,StepString);
          gotoxy (3,4+i);  Write (StepString);
          gotoxy (9,4+i);  Write (InstructionPack[Start_Step+i]^.Instruction,'          ');
          gotoxy (30,4+i); Write (InstructionPack[Start_Step+i]^.AssignComponent);
      end
      else
      begin
          k := i - 17;
          IntegerString (Start_Step + i,StepString);
          gotoxy (43,4+k);  Write (StepString);
          gotoxy (49,4+k);  Write (InstructionPack[Start_Step+i]^.Instruction,'          ');
          gotoxy (70,4+k);  Write (InstructionPack[Start_Step+i]^.AssignComponent);
      end;
    end;
end;

Procedure CreatInstTable;
begin
   Clrscr;
   WriteLn('+------------------------------  INSTRUCTION TABLE  ------------------------------+');
   WriteLn('|                                                                                |');
   WriteLn('+------+-------------------------------------------+------+-------------------------+');
   WriteLn('|      |                                           |      |                         |');
   WriteLn('|      |                                           |      |                         |');
   WriteLn('|      |                                           |      |                         |');
   WriteLn('|      |                                           |      |                         |');
```

```
    WriteLn('|     |                                        |     |
    WriteLn('|     |                                        |     |
    WriteLn('|     |                                        |     |
    WriteLn('|     |                                        |     |
    WriteLn('|     |                                        |     |
    WriteLn('|     |                                        |     |
    WriteLn('|     |                                        |     |
    WriteLn('|     |                                        |     |
    WriteLn('|     |                                        |     |
    WriteLn('|     |                                        |     |
    WriteLn('|     |                                        |     |
    WriteLn('|     |                                        |     |
    WriteLn('|     |                                        |     |
    WriteLn('+-----+----------------------------------------+-----+-------------------------------------------------------------------
   WriteLn ('Programmable Controller and DataLogger System');
   WriteLn ('Instrumentation Research KMITL     June 1988');
   Write   ('Instruction Display');
end;

Procedure DisplayInstruction;
Var  StepVar    : Integer;
begin
    StepVar := 0;
    CreatInstTable;
    GetInstruction (0,33);
    Repeat
       Repeat
       until Keypressed;
       Ch := ReadKey;
       If (Ch = #0) AND Keypressed then
       begin
          Ch := ReadKey;
          Case ch of

          Down : begin
                    StepVar := StepVar + 1;
                    If StepVar >= (MaxInst - 33) then StepVar := MaxInst - 33;
                    GetInstruction (0+StepVar,33+StepVar);
                 end;
          Up   :  begin
                     StepVar := StepVar - 1;
                     If StepVar <= 0 then StepVar := 0;
                     GetInstruction (0+StepVar,33+StepVar);
                  end;
          Left  : begin
                     StepVar := StepVar - 17;
                     If StepVar < 0 then StepVar := 0;
                     GetInstruction (0+StepVar,33+StepVar);
                  end;
          Right : begin
                     StepVar := StepVar + 17;
                     If StepVar >= (MaxInst - 33) then StepVar := MaxInst - 33;
                     GetInstruction (0+StepVar,33+StepVar);
                  end;
```

```
          end; {case}
        end;
      until Ch = Esc;
end;

Procedure DispInstruction;
begin
  DisplayInstruction;
end;
```

Program : Ladder Editor
Programmer : Viriya Kongratana

```
{////////////////////////////////////////////////////////////////////////////////
//                                                                              //
//                              PLC FUNCTION                                    //
//                                                                              //
////////////////////////////////////////////////////////////////////////////////}

Procedure Begin_Rung;
begin
   Read_Cursor;
   If (YCor Mod 2) = 0 then YCor := YCor + 1; gotoxy (XCor,YCor);
   write('+');
   Read_Cursor;
   XCor_Temp := XCor;
   YCor_Temp := YCor;
   gotoxy (79,YCor);
   writeln ('+');
   gotoxy (XCor_Temp,YCor_Temp);
end;

Procedure Begin_Rung_Block;
begin
  write('|');
  Read_Cursor;
  XCor_Temp := XCor;
  YCor_Temp := YCor;
  gotoxy (79,YCor);
  writeln ('|');
  gotoxy (XCor_Temp,YCor_Temp);
end;

Procedure New_Line_Block;
begin
   writeln;
   GetColor (Cursor_ch);
   If Cursor_ch <> $C3 then write ('|') else write ('+');
   Read_Cursor;
   gotoxy (79,YCor);
   GetColor (Cursor_ch);
   If Cursor_ch <> $B4 then writeln ('|') else writeln ('+');
   Begin_Rung_Block;
end;

Procedure New_line;
begin
  Read_Cursor;
```

```
  If YCor <= 21 then
  begin
     writeln;
     write ('|');
     Read_Cursor;
     gotoxy (79,YCor);
     writeln ('|');
     Begin_Rung;
  end;
end;

Procedure Out (Col,Line : Byte; IOassign : String6);
begin
  Col := 74;
  gotoxy (Col-1,Line-1);
  write(IOassign);
  gotoxy (Col,Line);
  write('( )—+');
end;

Procedure Out_Not (Col,Line : Byte; IOassign : String6);
begin
  Col := 74;
  gotoxy (Col-1,Line-1);
  write(IOassign);
  gotoxy (Col,Line);
  write('(/)—+');
end;

Procedure Relay (Col,line : byte; IOassign : string6);
begin
  gotoxy (Col+1,Line-1);
  write(IOassign);
  gotoxy (Col,Line);
  write ('—] [—');
end;

procedure Relay_Not (Col,Line : byte; IOassign : string6);
begin
  gotoxy (Col+1,Line-1);
  write(IOassign);
  gotoxy (Col,line);
  write ('—]/[—');
end;

Procedure Timer (Col,Line : byte; TimCnt_dta,IOassign : string6);
begin
  YCor := YCor - 1;
  gotoxy (68,YCor);   write ('+————————+');                    New_Line_Block;
  gotoxy (68,YCor-1); write ('+  TIMER  +');                    New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');                    New_Line_Block;
  gotoxy (68,YCor-1); write ('|  Ch',IOassign); gotoxy (78,YCor - 1); write('|');  New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');                    New_Line_Block;
  gotoxy (68,YCor-1); write ('|  #',TimCnt_dta);gotoxy (78,YCor - 1); write('|');  New_Line_Block;
```

```
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('+---------+');              New_Line_Block;
end;

Procedure TimerH (Col,Line : byte; TimCnt_dta,IOassign : string6);
begin
  YCor := YCor - 1;
  gotoxy (68,YCor);   write ('+---------+');              New_Line_Block;
  gotoxy (68,YCor-1); write ('+  TIMH15 +');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|  Ch',IOassign); gotoxy (78,YCor - 1); write('|');  New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|  #',TimCnt_dta);gotoxy (78,YCor - 1); write('|');  New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('+---------+');              New_Line_Block;
end;

Procedure Counter (Col,Line : byte; TimCnt_dta,IOassign : string6);
begin
  YCor := YCor - 1;
  gotoxy (68,YCor);   write ('+---------+');              New_Line_Block;
  gotoxy (68,YCor-1); write ('+  COUNT  +');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|  Ch',IOassign); gotoxy (78,YCor - 1); write('|');  New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('+  #',TimCnt_dta);gotoxy (78,YCor - 1); write('|');  New_Line_Block;
  gotoxy (68,YCor-1); write ('+---------+');              New_Line_Block;
end;

Procedure CounterReverse (Col,Line : byte; data1,Data2 : string6);
begin
  YCor := YCor - 1;
  gotoxy (68,YCor);   write ('+---------+');              New_Line_Block;
  gotoxy (68,YCor-1); write ('+  CNTR12 +');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|  Ch',Data1); gotoxy (78,YCor - 1); write('|');  New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('+  #',Data2); gotoxy (78,YCor - 1); write('|');  New_Line_Block;
  gotoxy (68,YCor-1); write ('+---------+');              New_Line_Block;
end;

Procedure SFTBlock (Col,Line : byte; Data1,Data2 : string6);
begin
  YCor := YCor - 1;
  gotoxy (68,YCor);   write ('+---------+');              New_Line_Block;
  gotoxy (68,YCor-1); write ('+  SFT',PCFunction); gotoxy (78,YCor - 1); write ('+'); New_Line_Block;
```

```
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('+ ',Data1); gotoxy (78,YCor - 1); write('|');  New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('+ ',Data2);gotoxy (78,YCor - 1); write('|');  New_Line_Block;
  gotoxy (68,YCor-1); write ('+---------+');              New_Line_Block;
end;

Procedure LatchKeep (Col,Line : byte; Data1 : string6);
begin
  YCor := YCor - 1;
  gotoxy (68,YCor);   write ('+---------+');              New_Line_Block;
  gotoxy (68,YCor-1); write ('+ KEEP11 +');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('+ ',Data1);gotoxy (78,YCor - 1); write('|');  New_Line_Block;
  gotoxy (68,YCor-1); write ('|         |');              New_Line_Block;
  gotoxy (68,YCor-1); write ('+---------+');              New_Line_Block;
end;

Procedure Push_IO (IO_Pattern : String6);
begin
  IOassign_len := 0;
  Repeat
  begin
    gotoxy (3,24); clreol;
    gotoxy (3,24); write(IO_Pattern);Readln(IOassign);
    IOassign_len := Length(IOassign);
  end;
  Until IOassign_len <= 6;
  gotoxy(3,24); clreol;
end;

Procedure Push_Data (Fun_Data : String6);
begin
  gotoxy (3,24);write (Fun_Data);Readln(TimCnT_Dta);
  gotoxy (3,24); clreol;
end;

{////////////////////////////////////////////////////////////////////////////}
{//                                                                        //}
{//                          END PLC FUNCTION                              //}
{//                                                                        //}
{////////////////////////////////////////////////////////////////////////////}

{////////////////////////////////////////////////////////////////////////////}
{//                                                                        //}
{//                          PC - FUNCTION                                 //}
{//                                                                        //}
{////////////////////////////////////////////////////////////////////////////}
```

```
Procedure Block (Col,Line,BlockLine : Byte; Block_Name,Data1,Data2,Data3 : String6);
begin
  PushCursor1;
  YCor := YCor - 1;
  Case BlockLine of
  1 : begin
        gotoxy (68,YCor);   write ('+--------+');                New_Line_Block;
        gotoxy (68,YCor-1); write ('+  ',Block_Name,PCFunction); gotoxy (78,YCor - 1); write('+');  New
        gotoxy (68,YCor-1); write ('+--------+');                New_Line_Block;
        PopCursor1;
        LinkLeft_check;
        Link_End (68);
        New_Line;
        New_Line;
      end;
  2 : begin
        gotoxy (68,YCor);   write ('+--------+');                New_Line_Block;
        gotoxy (68,YCor-1); write ('+  ',Block_Name,PCFunction); gotoxy (78,YCor - 1); write('+');  Ne..
        gotoxy (68,YCor-1); write ('+--------+');                New_Line_Block;
        gotoxy (68,YCor-1); write ('|  ',Data1); gotoxy (78,YCor - 1); write('|');  New_Line_Block;
        gotoxy (68,YCor-1); write ('+--------+');                New_Line_Block;
        PopCursor1;
        LinkLeft_check;
        Link_End (68);
        New_Line;
      end;
  3 : begin
        gotoxy (68,YCor);   write ('+--------+');                New_Line_Block;
        gotoxy (68,YCor-1); write ('+  ',Block_Name,PCFunction); gotoxy (78,YCor - 1); write('+');  New
        gotoxy (68,YCor-1); write ('+--------+');                New_Line_Block;
        gotoxy (68,YCor-1); write ('|  ',Data1); gotoxy (78,YCor - 1); write('|');  New_Line_Block;
        gotoxy (68,YCor-1); write ('+--------+');                New_Line_Block;
        gotoxy (68,YCor-1); write ('|  ',Data2); gotoxy (78,YCor - 1); write('|');  New_Line_Block;
        gotoxy (68,YCor-1); write ('+--------+');                New_Line_Block;
        PopCursor1;
        LinkLeft_check;
        Link_End (68);
        New_Line;
      end;

  4 : begin
        gotoxy (68,YCor);   write ('+--------+');                New_Line_Block;
        gotoxy (68,YCor-1); write ('+  ',Block_Name,PCFunction); gotoxy (78,YCor - 1); write('+');  New_
        gotoxy (68,YCor-1); write ('+--------+');                New_Line_Block;
        gotoxy (68,YCor-1); write ('|  ',Data1); gotoxy (78,YCor - 1); write('|');  New_Line_Block;
        gotoxy (68,YCor-1); write ('+--------+');                New_Line_Block;
        gotoxy (68,YCor-1); write ('|  ',Data2); gotoxy (78,YCor - 1); write('|');  New_Line_Block;
        gotoxy (68,YCor-1); write ('+--------+');                New_Line_Block;
        gotoxy (68,YCor-1); write ('|  ',Data3); gotoxy (78,YCor - 1); write('|');  New_Line_Block;
        gotoxy (68,YCor-1); write ('+--------+');                New_Line_Block;
        PopCursor1;
        LinkLeft_check;
        Link_End (68);
```

```
        New_Line;
      end;

  end; {case}
end;

Procedure PC_Function;
begin
  PushCursor;
  gotoxy (3,24); clreol;
  gotoxy (3,24); write ('Function:'); Readln (PCFunction);
  PopCursor;
  Case PCFunction of
  00 : Block (XCor,YCor,1,'NOP',Data1,Data2,Data3);
  01 : Block (XCor,YCor,1,'END',Data1,Data2,Data3);
  02 : Block (XCor,YCor,1,'IL',Data1,Data2,Data3);
  03 : Block (XCor,YCor,1,'ILC',Data1,Data2,Data3);
  04 : Block (XCor,YCor,1,'JUM',Data1,Data2,Data3);
  05 : Block (XCor,YCor,1,'JME',Data1,Data2,Data3);
  40 : Block (XCor,YCor,1,'STC',Data1,Data2,Data3);
  41 : Block (XCor,YCor,1,'CTC',Data1,Data2,Data3);


  10 : begin
               Read_Cursor;
               XCor_Temp1 := XCor;
               YCor_Temp1 := YCor;
               gotoxy (3,24); clreol;
               gotoxy (3,24); write ('[SFT10] StartCh:');Readln (Data1);
               gotoxy (3,24); clreol;
               gotoxy (3,24); write ('[SFT10] EndCh:');Readln (Data2);
               SFTBlock (XCor,YCor,Data1,Data2);
               XCor := XCor_Temp1;
               YCor := YCor_Temp1;
               gotoxy (XCor,YCor);
               LinkLeft_check;
               Link_End (68);
               New_Line;
       end;

  11 : begin
               Read_Cursor;
               XCor_Temp1 := XCor;
               YCor_Temp1 := YCor;
               gotoxy (3,24); clreol;
               gotoxy (3,24); write ('[KEEP10] Relay:');Readln (Data1);
               LatchKeep (XCor,YCor,Data1);
               XCor := XCor_Temp1;
               YCor := YCor_Temp1;
               gotoxy (XCor,YCor);
               LinkLeft_check;
               Link_End (68);
               New_Line;
       end;
```

```
 12 : begin
              Read_Cursor;
              XCor_Temp1 := XCor;
              YCor_Temp1 := YCor;
              gotoxy (3,24); clreol;
              gotoxy (3,24); write ('[CNTR12] Ch:');Readln (Data1);
              gotoxy (3,24); clreol;
              gotoxy (3,24); write ('[CNTR12] #/Ch:');Readln (Data2);
              CounterReverse (XCor,YCor,Data1,Data2);
              XCor := XCor_Temp1;
              YCor := YCor_Temp1;
              gotoxy (XCor,YCor);
              LinkLeft_check;
              Link_End (68);
              New_Line;
      end;

 13:  begin
       PushCursor;
       gotoxy (3,24); clreol;
       gotoxy (3,24); write ('[DIFU13] Relay:');Readln (Data1);
       PopCursor;
       Block (XCor,YCor,2,'DIFU',Data1,Data2,Data3);
      end;

 14 : begin
       PushCursor;
       gotoxy (3,24); clreol;
       gotoxy (3,24); write ('[DIFD14] Relay:');Readln (Data1);
       PopCursor;
       Block (XCor,YCor,2,'DIFD',Data1,Data2,Data3);
      end;

 15 : begin
              Read_Cursor;
              XCor_Temp1 := XCor;
              YCor_Temp1 := YCor;
              gotoxy (3,24); clreol;
              gotoxy (3,24); write ('[TIMH15] Ch:'); Readln(IOassign);
              gotoxy (3,24); clreol;
              gotoxy (3,24); write ('[TIMH15] #Data:'); Readln (TimCnt_Dta);
              TimerH (XCor,YCor,TimCnt_Dta,IOassign);
              XCor := XCor_Temp1;
              YCor := YCor_Temp1;
              gotoxy (XCor,YCor);
              LinkLeft_check;
              Link_End (68);
              New_Line;
      end;

 25 : begin
       PushCursor;
       gotoxy (3,24); clreol;
```

```
        gotoxy (3,24); write ('[ASL25] Ch:');Readln (Data1);
        PopCursor;
        Block (XCor,YCor,2,'ASL',Data1,Data2,Data3);
      end;

  26 : begin
        PushCursor;
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[ASR26] Ch:');Readln (Data1);
        PopCursor;
        Block (XCor,YCor,2,'ASR',Data1,Data2,Data3);
      end;

  27 : begin
        PushCursor;
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[ROL27] Ch:');Readln (Data1);
        PopCursor;
        Block (XCor,YCor,2,'ROL',Data1,Data2,Data3);
      end;

  28 : begin
        PushCursor;
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[ROR28] Ch:');Readln (Data1);
        PopCursor;
        Block (XCor,YCor,2,'ROR',Data1,Data2,Data3);
      end;

  29 : begin
        PushCursor;
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[COM29] Ch:');Readln (Data1);
        PopCursor;
        Block (XCor,YCor,2,'COM',Data1,Data2,Data3);
      end;

  16 : begin
        PushCursor;
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[WSFT16] D1:');Readln (Data1);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[WSFT16] D2:');Readln (Data2);
        PopCursor;
        Block (XCor,YCor,3,'WSFT',Data1,Data2,Data3);
      end;

  20 : begin
        PushCursor;
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[CMP20] S1:');Readln (Data1);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[CMP20] S2:');Readln (Data2);
        PopCursor;
```

```
        Block (XCor,YCor,3,'CMP',Data1,Data2,Data3);
      end;

 21 : begin
        PushCursor;
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[MOV21] SourceCh:');Readln (Data1);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[MOV21] DestCh:');Readln (Data2);
        PopCursor;
        Block (XCor,YCor,3,'MOV',Data1,Data2,Data3);
      end;

 22 : begin
        PushCursor;
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[MOVN22] SourceCh:');Readln (Data1);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[MOVN22] DestCh:');Readln (Data2);
        PopCursor;
        Block (XCor,YCor,3,'MOVN',Data1,Data2,Data3);
      end;

 23 : begin
        PushCursor;
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[BIN23] SourceCh/Data#:');Readln (Data1);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[BIN23] DestCh:');Readln (Data2);
        PopCursor;
        Block (XCor,YCor,3,'BIN',Data1,Data2,Data3);
      end;

 24 : begin
        PushCursor;
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[BCD24] SourceCh/Data#:');Readln (Data1);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[BCD24] DestCh:');Readln (Data2);
        PopCursor;
        Block (XCor,YCor,3,'BCD',Data1,Data2,Data3);
      end;

 30 : begin
        PushCursor;
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[ADD30] S1:');Readln (Data1);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[ADD30] S2:');Readln (Data2);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[ADD30] DestCh:');Readln (Data3);
        PopCursor;
        Block (XCor,YCor,4,'ADD',Data1,Data2,Data3);
      end;
```

```
 31 : begin
        PushCursor;
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[SUB31] S1:');Readln (Data1);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[SUB31] S2:');Readln (Data2);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[SUB31] DestCh:');Readln (Data3);
        PopCursor;
        Block (XCor,YCor,4,'SUB',Data1,Data2,Data3);
      end;

 32 : begin
        PushCursor;
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[MUL32] S1:');Readln (Data1);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[MUL32] S2:');Readln (Data2);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[MUL32] DestCh:');Readln (Data3);
        PopCursor;
        Block (XCor,YCor,4,'MUL',Data1,Data2,Data3);
      end;

 33 : begin
        PushCursor;
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[DIV33] S1:');Readln (Data1);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[DIV33] S2:');Readln (Data2);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[DIV33] DestCh:');Readln (Data3);
        PopCursor;
        Block (XCor,YCor,4,'DIV',Data1,Data2,Data3);
      end;

 34 : begin
        PushCursor;
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[ANDW34] S1:');Readln (Data1);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[ANDW34] S2:');Readln (Data2);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[ANDW34] DestCh:');Readln (Data3);
        PopCursor;
        Block (XCor,YCor,4,'ANDW',Data1,Data2,Data3);
      end;

 35 : begin
        PushCursor;
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[ORW35] S1:');Readln (Data1);
        gotoxy (3,24); clreol;
```

```
        gotoxy (3,24); write ('[ORW35] S2:');Readln (Data2);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[ORW35] DestCh:');Readln (Data3);
        PopCursor;
        Block (XCor,YCor,4,'ORW',Data1,Data2,Data3);
      end;

  36 : begin
        PushCursor;
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[XORW36] S1:');Readln (Data1);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[XORW36] S2:');Readln (Data2);
        gotoxy (3,24); clreol;
        gotoxy (3,24); write ('[XORW36] DestCh:');Readln (Data3);
        PopCursor;
        Block (XCor,YCor,4,'XORW',Data1,Data2,Data3);
      end;

  end; {case}
end;

{////////////////////////////////////////////////////////////////////////////
//                                                                          //
//                        END PC - FUNCTION                                 //
//                                                                          //
////////////////////////////////////////////////////////////////////////////}

{////////////////////////////////////////////////////////////////////////////
//                                                                          //
//                          DELETE MODE                                     //
//                                                                          //
////////////////////////////////////////////////////////////////////////////}


Procedure DeleteBlock (X,Y,I : Byte);
begin
 PushCursor;
 TextBackGround (Black);
 TextColor (LightGray);
 for j := 1 to i do
 begin
   gotoxy ((X+j-1),Y); write (Chr($20));
   gotoxy ((X+j-1),(Y-1)); write (Chr($20));
 end;
 FillChar (BlockLine,MaxChr,$20);
 FillChar (BlockChar,MaxChr,$20);
 PopCursor;
end;

Procedure Cancel_DelBlock (X,Y,I : Byte);
begin
 PushCursor;
 TextBackGround (Black);
```

```
 TextColor (LightGray);
 GetColor (Cursor_Ch);
 If Cursor_Ch > $20 then
 for j := 1 to i do
 begin
   gotoxy ((X+j-1),Y); write (Chr(BlockLine[j]));
   gotoxy ((X+j-1),(Y-1)); write (Chr(BlockChar[j]));
 end else
 for j := 1 to i do
 begin
  gotoxy ((X+j-1),Y); write (Chr($20));
  gotoxy ((X+j-1),(Y-1)); write (Chr($20));
 end;
 FillChar (BlockLine,MaxChr,$20);
 FillChar (BlockChar,MaxChr,$20);
 PopCursor;
end;

Procedure Mark (Var X,Y,I : Byte);
begin
 PushCursor;
 TextBackGround (LightGray);
 TextColor (Black);
 for j := 1 to i do
 begin
   gotoxy ((X+j-1),Y); write (Chr(BlockLine[j]));
   gotoxy ((X+j-1),(Y-1)); write (Chr(BlockChar[j]));
 end;
 PopCursor;
end;

Procedure MarkBlock_Hor (SizeBlock : Byte);
begin
  PushCursor;
  for i := 1 to SizeBlock do
  begin
    GetColor (Cursor_Ch);
    BlockLine[i] := Cursor_Ch;
    gotoxy (XCor,(YCor - 1));
    GetColor (Cursor_Ch);
    BlockChar[i] := Cursor_Ch;
    XCor := XCor + 1;
    gotoxy (XCor,YCor);
  end;
PopCursor;
i := SizeBlock;
Mark (XCor,YCor,I);
end;

Procedure DeleteBlock_Ver (X,Y,I : Byte);
begin
 PushCursor;
 TextBackGround (Black);
```

```
 TextColor (LightGray);
 for j := 1 to i do
 begin
   gotoxy (X,(Y-j+1)); write (Chr($20));
 end;
 FillChar (BlockLine,MaxChr,$20);
 PopCursor;
end;

Procedure Cancel_DelBlockVer (X,Y,I : Byte);
begin
 PushCursor;
 TextBackGround (Black);
 TextColor (LightGray);
 GetColor (Cursor_Ch);
 If Cursor_Ch > $20 then
 for j := 1 to i do
 begin
   gotoxy (X,(Y-j+1)); write (Chr(BlockLine[j]));
 end else
 for j := 1 to i do
 begin
  gotoxy (X,(Y-j+1)); write (Chr($20));
 end;
 fillChar (BlockLine,MaxChr,$20);
 PopCursor;
end;

Procedure Mark_Ver (Var X,Y,I : Byte);
begin
 PushCursor;
 TextBackGround (LightGray);
 TextColor (Black);
 for j := 1 to i do
 begin
   gotoxy (X,(Y-j+1)); write (Chr(BlockLine[j]));
 end;
 PopCursor;
end;

Procedure Find_High;
begin
  PushCursor1; HighBlock := 0;
  gotoxy (XCor,YCor);
  repeat
    GetColor (Cursor_Ch);
    HighBlock := HighBlock + 1; gotoxy (XCor,(YCor - HighBlock));
  Until (Cursor_Ch = $C2) OR ((YCor - HighBlock) = 2);
  HighBlock := HighBlock - 1;
  PopCursor1;
end;

Procedure MarkBlock_Ver;
begin
```

```
  PushCursor;
  Find_High;
  for i := 1 to HighBlock do
  begin
    GetColor (Cursor_Ch);
    BlockLine[i] := Cursor_Ch;
    gotoxy (XCor,(YCor - 1));
  end;
PopCursor;
i := HighBlock;
Mark_Ver (XCor,YCor,I);
end;

{////////////////////////////////////////////////////////////////////////////////
//                                                                              //
//                              DELETE Block                                    //
//                                                                              //
////////////////////////////////////////////////////////////////////////////////}

Procedure DeleteBlock_Block (X,Y,Block_Size,J : Byte);
begin
 PushCursor;
 TextBackGround (Black);
 TextColor (LightGray);
 for k := 0 to j do
 for i := 1 to Block_Size do
 begin
   gotoxy ((X+i-1),(Y-1+k)); write (Chr($20));
 end;
 fillchar (BlockChar,MaxChr,$20);
 PopCursor;
end;

Procedure CancelDelBlock_Block (X,Y,Block_Size,j : Byte);
begin
  PushCursor;
  TextBackGround (Black);
  TextColor (LightGray);
  GetColor (Cursor_Ch);
  If Cursor_Ch > $20 then
  for k := 0 to j do
    for i := 1 to Block_Size do
      begin
        gotoxy ((X+i-1),(Y-1+k)); write (Chr(BlockChar[(Block_Size * k)+i]));
      end else
    for k := 0 to j do
      for i := 1 to Block_Size do
        begin
          gotoxy ((X+i-1),(Y-1+k)); write (Chr($20));
        end;
  PopCursor;
  FillChar (BlockChar,MaxChr,$20);
end;
```

```
Procedure Find_SizeBlock (Var k : Byte) ;
begin
  PushCursor1; XCor := 68; LineBlock := 0;
  gotoxy (XCor,YCor);
  repeat
    GetColor (Cursor_Ch);
    k := k + 1; gotoxy (XCor,(YCor + k));
  Until (Cursor_Ch = $C0) OR ((YCor + k) >= 23);
  PopCursor1;
end;

Procedure Mark_Block (X,Y,Block_Size : Byte; Var J : Byte);
begin
 PushCursor;
 TextBackGround (LightGray);
 TextColor (Black);
 for k := 0 to j do
 for i := 1 to Block_Size do
 begin
   gotoxy ((X+i-1),(Y-1+k)); write (Chr(BlockChar[((Block_Size*k)+i)]));
 end;
 PopCursor;
end;

Procedure MarkBlock_Block;
begin
  PushCursor;
  SizeBlock := 13; k := 0;
  Find_SizeBlock (k);
  for j := 0 to k do
  for i := 1 to SizeBlock do
  begin
    gotoxy ((XCor+i-1),(YCor - 1 + j));
    GetColor (Cursor_Ch);
    BlockChar[((SizeBlock*j)+i)] := Cursor_Ch;
  end;
  PopCursor;
  I := 13; LineBlock := k;
  Mark_Block (XCor,YCor,I,LineBlock);
end;


Procedure MarkBlock;
begin
  GetColor (Cursor_Ch);
  Case Cursor_Ch of
  $B3,$B4              : begin
                           DelFlag := 01;
                           MarkBlock_Ver;
                         end;
  $C4,$C1,$C2,$C5,$D9 : begin
                           DelFlag := 02;
                           If XCor = 66 then
                           begin
```

```
                        gotoxy (68,YCor); GetColor (Cursor_ch);
                        gotoxy (68,YCor -1); GetColor (Cursor_Ch1);
                        If (Cursor_ch = $B4) AND (Cursor_Ch1 = $DA) then
                        begin
                          gotoxy (XCor,YCor);
                          DelFlag := 03;
                          MarkBlock_Block;
                        end else  gotoxy (XCor,YCor);MarkBlock_Hor (13);
                      end else  MarkBlock_Hor (8);
                     end;
  end; {Case}
end;

procedure Delete;
begin
  PushCursor;
  gotoxy (3,24); clreol;
  write ('Ins/Delete Mode');
  PopCursor;
  If XCor > 8 then
    XCor := ((XCor DIV 8)*8)+2 else XCor := 2;
  gotoxy (XCor,YCor);
  DelFlag := 00;
  Repeat
   Repeat
   until Keypressed;
   Ch := ReadKey;
   If (Ch = #0) AND Keypressed then
   begin
     Ch := ReadKey;
     If DelFlag <> 00 then
     begin
       If (DelFlag = 01) AND (Ch = DelKey) then DeleteBlock_Ver (XCor,YCor,I);
       If (DelFlag = 01) AND (Ch IN [Up,Down,Left,Right]) then Cancel_DelBlockVer (XCor,YCor,I);
       If (DelFlag = 02) AND (Ch = DelKey) then DeleteBlock (XCor,YCor,I);
       If (DelFlag = 02) AND (Ch IN [Up,Down,Left,Right]) then Cancel_DelBlock (XCor,YCor,I);
       If (DelFlag = 03) AND (Ch = DelKey) then DeleteBlock_Block (XCor,YCor,I,LineBlock);
       If (DelFlag = 03) AND (Ch IN [Up,Down,Left,Right]) then CancelDelBlock_Block (XCor,YCor,I,LineBlock);
     end;
     DelFlag := 00;
     Case ch of

     Up    : begin
               Read_Cursor;
               YCor := YCor - 2;
               gotoxy (XCor,YCor);
               GetColor (Cursor_ch);
               If Cursor_ch >= $B3 then MarkBlock;
             end;

     Down  : begin
               Read_Cursor;
               YCor := YCor + 2;
               gotoxy (XCor,YCor);
```

```
              GetColor (Cursor_ch);
              If Cursor_ch >= $B3 then MarkBlock;
            end;

     Left  : begin
              Read_Cursor;
              XCor := ((XCor DIV 8)*8)+2;
              XCor := XCor - 8;
              gotoxy (XCor,YCor);
              GetColor (Cursor_ch);
              If Cursor_ch >= $B3 then MarkBlock;
            end;

     Right : begin
              Read_Cursor;
              XCor := ((XCor DIV 8)*8)+2;
              XCor := XCor + 8;
              If XCor > 66 then XCor := 66;
              gotoxy (XCor,YCor);
              GetColor (Cursor_ch);
              If Cursor_Ch >= $B3 then MarkBlock;
            end;

     end;{case}
     end; {If}
  until Ch = Esc;
  Ch := ClearChr; {Clear SubCommand};
  TextBackGround (Black);
  TextColor (LightGray);
  CanCel_DelBlock (XCor,YCor,I);
end;

{////////////////////////////////////////////////////////////////////////////
//                                                                          //
//                          End DELETE Block                                //
//                                                                          //
////////////////////////////////////////////////////////////////////////////}

{////////////////////////////////////////////////////////////////////////////
//                                                                          //
//                       EDITOR OPERATING SYSTEM                            //
//                                                                          //
////////////////////////////////////////////////////////////////////////////}

Procedure PushWindow (Size,Line : Byte);
begin
  PushCursor1;
  for j := 0 to Line do
  for i := 1 to Size do
  begin
    gotoxy ((XCor+i-1),(YCor + j));
    GetColor (Cursor_Ch);
    BlockChar[((Size*j)+i)] := Cursor_Ch;
  end;
```

```
  PopCursor1;
end;

Procedure CommandWindow;
begin
  PushCursor1;
  TextBackGround (LightGray);
  TextColor (Black);
  gotoxy (XCor,YCor);    write ('+--Command--+');
  gotoxy (XCor,YCor+1);  write ('| Load      |');
  gotoxy (XCor,YCor+2);  write ('| Save      |');
  gotoxy (XCor,YCor+3);  write ('| Compile   |');
  gotoxy (XCor,YCor+4);  write ('| Function  |');
  gotoxy (XCor,YCor+5);  write ('| DispINST. |');
  gotoxy (XCor,YCor+6);  write ('| Monitoring|');
  gotoxy (XCor,YCor+7);  write ('| HardCopy  |');
  gotoxy (XCor,YCor+8);  write ('| PageCOM   |');
  gotoxy (XCor,YCor+9);  write ('| Initialize|');
  gotoxy (XCor,YCor+10); write ('+-----------+');
  PopCursor1; gotoxy (XCor,YCor);
end;

Procedure PageWindow;
begin
  PushCursor1;
  TextBackGround (LightGray);
  gotoxy (XCor,YCor);    write ('+-------------------+');
  gotoxy (XCor,YCor+1);  write ('|                   |');
  gotoxy (XCor,YCor+2);  write ('|                   |');
  gotoxy (XCor,YCor+3);  write ('+-------------------+');
  TextColor (Black);
  gotoxy (XCor,YCor);    write ('+--PageCOM---------+');
  gotoxy (XCor,YCor+1);  write ('| Clear             |');
  gotoxy (XCor,YCor+2);  write ('| PageNum[80]..',DiagramPage); gotoxy (Xcor+19,YCor+2);write ('|');
  gotoxy (XCor,YCor+3);  write ('+-------------------+');
  PopCursor1; gotoxy (XCor,YCor);
end;


Procedure LoadWindow;
begin
  PushCursor1;
  TextBackGround (LightGray);
  TextColor (Black);
  gotoxy (XCor,YCor);    write ('+---------- Load File----------+');
  gotoxy (XCor,YCor+1);  write ('|                               |');
  gotoxy (XCor,YCor+2);  write ('+-------------------------------+');
  PopCursor1; gotoxy (XCor,YCor);
end;

Procedure SaveWindow;
begin
  PushCursor1;
  TextBackGround (LightGray);
```

```
  TextColor (Black);
  gotoxy (XCor,YCor);    write ('+---------- Save File----------+');
  gotoxy (XCor,YCor+1);  write ('|                              |');
  gotoxy (XCor,YCor+2);  write ('+------------------------------+');
  PopCursor1; gotoxy (XCor,YCor);
end;

Procedure CompileWindow;
begin
  PushCursor1;
  TextBackGround (LightGray);
  TextColor (Black);
  gotoxy (XCor,YCor);    write ('+---------- Compilation ----------+');
  gotoxy (XCor,YCor+1);  write ('|                                 |');
  gotoxy (XCor,YCor+2);  write ('+---------------------------------+');
  PopCursor1; gotoxy (XCor,YCor);
end;

Procedure FunctionWindow;
begin
  PushCursor1;
  TextBackGround (LightGray);
  TextColor (Black);
  gotoxy (XCor,YCor);    write ('+---------- Function ----------+');
  gotoxy (XCor,YCor+1);  write ('| 00 NOP   11 KEEP  23 BIN   32 MUL   41 CLC |');
  gotoxy (XCor,YCor+2);  write ('| 01 END   12 CNTR  24 BCD   33 DIV   ...... |');
  gotoxy (XCor,YCor+3);  write ('| 02 IL    13 DIFU  25 ASL   34 ANDW  ...... |');
  gotoxy (XCor,YCor+4);  write ('| 03 ILC   14 DIFD  26 ASR   35 ORW   ...... |');
  gotoxy (XCor,YCor+5);  write ('| 04 JMP   15 TIMH  27 ROL   36 XORW  ...... |');
  gotoxy (XCor,YCor+6);  write ('| 05 JME   16 WSFT  28 ROR   37 XNRW  ...... |');
  gotoxy (XCor,YCor+7);  write ('| 06 FAL   20 CMP   29 COM   38 INC   ...... |');
  gotoxy (XCor,YCor+8);  write ('| 07 FALS  21 MOV   30 ADD   39 DEC   ...... |');
  gotoxy (XCor,YCor+9);  write ('| 10 SFT   22 MVN   31 SUB   40 STC   ...... |');
  gotoxy (XCor,YCor+10); write ('+-------------------------------------------+');
  PopCursor1; gotoxy (XCor,YCor);
end;

Procedure FunctionWindow1;
begin
  PushCursor1;
  TextBackGround (LightGray);
  TextColor (Black);
  gotoxy (XCor,YCor);    write ('+---------- Function ----------+');
  gotoxy (XCor,YCor+1);  write ('| 70 Move Block           79 Floating Devide  |');
  gotoxy (XCor,YCor+2);  write ('| 71 Block Set            80 Data Distribution|');
  gotoxy (XCor,YCor+3);  write ('| 72 Square Root          81 Data Extraction  |');
  gotoxy (XCor,YCor+4);  write ('| 73 Data Exchange        82 Bit Transfer     |');
  gotoxy (XCor,YCor+5);  write ('| 74 One Digit Shift Left 83 Digit Transfer   |');
  gotoxy (XCor,YCor+6);  write ('| 75 One Digit Shift Right ................. |');
  gotoxy (XCor,YCor+7);  write ('| 76  4 to 16 Decoder      ................. |');
  gotoxy (XCor,YCor+8);  write ('| 77 16 to  4 Decoder      ................. |');
  gotoxy (XCor,YCor+9);  write ('| 78 Seven Segment Decoder ................. |');
  gotoxy (XCor,YCor+10); write ('+-------------------------------------------+');
  PopCursor1; gotoxy (XCor,YCor);
```

```
end;

Procedure FunctionWindow2;
begin
  PushCursor1;
  TextBackGround (LightGray);
  TextColor (Black);
  gotoxy (XCor,YCor);    write ('+---------- Special Relay ----------+');
  gotoxy (XCor,YCor+1);  write ('|                                   |');
  gotoxy (XCor,YCor+2);  write ('|                                   |');
  gotoxy (XCor,YCor+3);  write ('|                                   |');
  gotoxy (XCor,YCor+4);  write ('|                                   |');
  gotoxy (XCor,YCor+5);  write ('|                                   |');
  gotoxy (XCor,YCor+6);  write ('|                                   |');
  gotoxy (XCor,YCor+7);  write ('|                                   |');
  gotoxy (XCor,YCor+8);  write ('|                                   |');
  gotoxy (XCor,YCor+9);  write ('|                                   |');
  gotoxy (XCor,YCor+10); write ('+-----------------------------------+');
  PopCursor1; gotoxy (XCor,YCor);
end;

Procedure PageClear (PageClr : byte);
begin
  Count := 1;
  While Count <= PageChar do
  begin
    LadderDiagram[PageClr]^.LadderPage[Count] := $20;
    Count := Count + 1;
    LadderDiagram[PageClr]^.LadderPage[Count] := $07;
    Count := Count + 1;
  end;
  Move (LadderDiagram[PageClr]^.LadderPage[1],ScreenMem,PageChar);
  gotoxy (30,1); Clreol; writeln ('Ladder Diagram Page:',Page);
  writeln;
  Begin_Rung;
  Read_Cursor;
  XCor_Temp := XCor; YCor_Temp := YCor;
end;

Procedure NumPage;
begin
  PushCursor;
  gotoxy (XCor+15,YCor+2);write ('  '); gotoxy (XCor+15,YCor+2);
  Readln (DiagramPage);
  PopCursor;
end;

Procedure PushPage (CurPage : Byte);
begin
  Move (ScreenMem,LadderDiagram[CurPage]^,PageChar);
end;

Procedure PopPage (CurPage : Byte);
begin
```

```
    Move (LadderDiagram[CurPage]^,ScreenMem,PageChar);
  end;

  Procedure PageUp;
  begin
    PushPage (Page);
    Page := Page - 1;
    If Page < 1 then Page := DiagramPage;
    clrscr;
    PopPage (Page);
    gotoxy (2,1);write (FileName);
    gotoxy (28,1); Clreol; writeln ('Ladder Diagram Page:',Page,'/',DiagramPage);
    writeln;
    Begin_Rung;
    Read_Cursor;
    XCor_Temp := XCor; YCor_Temp := YCor;
  end;

  Procedure PageDown;
  begin
    PushPage (Page);
    Page := Page +  1;
    If Page > DiagramPage then Page := 1;
    clrscr;
    PopPage (Page);
    gotoxy (2,1);write (FileName);
    gotoxy (28,1); Clreol; writeln ('Ladder Diagram Page:',Page,'/',DiagramPage);
    writeln;
    Begin_Rung;
    Read_Cursor;
    XCor_Temp := XCor; YCor_Temp := YCor;
  end;

  Procedure ClearLadderMem;
  begin
    i := 1;
    While i <= MaxDiagramPage do
    begin
      Count := 1;
      While Count <= PageChar do
      begin
        LadderDiagram[i]^.LadderPage[Count] := $20;
        Count := Count + 1;
        LadderDiagram[i]^.LadderPage[Count] := $07;
        Count := Count + 1;
      end;
      Move (LadderDiagram[i]^.LadderPage[1],LadderDiagram[i]^,PageChar);
      i := i + 1;
    end;
  end;

  Procedure Command;
  begin
    If XCor > 29 then XCor := 29;
```

```
  If YCor > 13 then YCor := 13;
  gotoxy (XCor,YCor);
  SizeBlock := 13;LineBlock := 10;
  PushWindow (SizeBlock,LineBlock);
  CommandWindow;
  Ch := ReadKey;
    If Ch <> Esc then
    begin
      CancelDelBlock_Block (XCor,(YCor+1),SizeBlock,(LineBlock));
      Ch := UpCase (Ch);
      Case Ch of
      'L' : begin
              SizeBlock := 36; LineBlock := 3;
              PushWindow (SizeBlock,LineBlock);
              LoadWindow;
              gotoxy (XCor+2,YCor+1);
              write ('FileName:',FileName);
              Repeat
              Until Keypressed;
              Ch := ReadKey;
              If Ch <> Return then
              begin
                  gotoxy (XCor+2,YCor+1);
                  write ('FileName:','                ');
                  gotoxy (XCor+2,YCor+1);
                  write ('FileName:');
                  ReadLn (FileName);
              end;
              Assign (FileVar,FileName);
              Reset (FileVar);
              DiagramPage := FileSize(FileVar);
              for i := 1 to DiagramPage do
                  Read (FileVar,LadderDiagram[i]^);
              close (FileVar);
              gotoxy (XCor,YCor);
              SizeBlock := 36; LineBlock := 3;
              CancelDelBlock_Block (XCor,(YCor+1),SizeBlock,(LineBlock));
              Clrscr;
              Move (LadderDiagram[Page]^,ScreenMem,PageChar);
              gotoxy (2,1);write (FileName);
              gotoxy (30,1); Clreol; writeln ('Ladder Diagram Page:',Page);
              gotoxy (XCor,YCor);
            end;

      'S' : begin
              SizeBlock := 36; LineBlock := 3;
              PushWindow (SizeBlock,LineBlock);
              SaveWindow;
              gotoxy (XCor+2,YCor+1);
              write ('FileName:',FileName);
              Repeat
              Until Keypressed;
              Ch := ReadKey;
              If Ch <> Return then
```

```
            begin
                gotoxy (XCor+2,YCor+1);
                write ('FileName:','                ');
                gotoxy (XCor+2,YCor+1);
                write ('FileName:');
                ReadLn (FileName);
            end;
               Assign (FileVar,FileName);
               Rewrite (FileVar);
               for i := 1 to DiagramPage do
                   write (FileVar,LadderDiagram[i]^);
               close (FileVar);
               gotoxy (XCor,YCor);
               SizeBlock := 36; LineBlock := 3;
               CancelDelBlock_Block (XCor,(YCor+1),SizeBlock,(LineBlock));
               gotoxy (XCor,YCor);
          end;

     'F' : begin
             SizeBlock := 50; LineBlock := 10;
             PushWindow (SizeBlock,LineBlock);
             FunctionWindow;
             Repeat
             Until Keypressed; Ch := ReadKey;
             FunctionWindow1;
             Repeat
             Until Keypressed; Ch := ReadKey;
             FunctionWindow2;
             Repeat
             Until KeyPressed; Ch := ReadKey;
             SizeBlock := 50; LineBlock := 10;
             CancelDelBlock_Block (XCor,(YCor+1),SizeBlock,(LineBlock));
           end;

     'P' : begin
             SizeBlock := 21;LineBlock := 3;
             PushWindow (SizeBlock,LineBlock);
             PageWindow;
             Repeat
             Ch := ReadKey;
             Ch := Upcase (Ch);
                Ch := UpCase (Ch);
                Case Ch of
                'C' : begin
                          CancelDelBlock_Block (XCor,(YCor+1),SizeBlock,LineBlock);
                          PageClear (Page);
                      end;
                'P' : begin
                          NumPage;
                          CancelDelBlock_Block (XCor,(YCor+1),SizeBlock,LineBlock);
                      end;
             end;
              until Ch IN ['C','P'];
          end;
```

```
      'C' :  begin
                SizeBlock := 70; LineBlock := 3;
                PushWindow (SizeBlock,LineBlock);
                ComPileWindow;
                gotoxy (XCor+2,YCor+1);
                write ('Destination FileName: ');
                Readln (FileNameDest);
                gotoxy (XCor,YCor);
                SizeBlock := 70; LineBlock := 3;
                CancelDelBlock_Block (XCor,(YCor+1),SizeBlock,(LineBlock));
                gotoxy (XCor,YCor);
                Compile ;
             end;

      'I' : begin
              ClearLadderMem;
              PageClear (Page);
            end;

      'H' : PrintLadderDiagram;
      'M' : LadderMonitor;
      'D' : begin
               SwitchScreen (1,3);
               DispInstruction;
               SwitchScreen (3,1);
               gotoxy (2,3);
            end;

      end; {case}
      Ch := ReadKey;
    end;
  TextBackGround (Black);
  TextColor (LightGray);
end;
```

Program : Ladder Editor Main Program
Programmer : Viriya Kongratana

```
Procedure  MainEditor;
begin
 MainMenu;
 ClearLadderMem;
 ClearInstruction;
 FileName := '';
 TextColor(LightGray);
 clrscr;
 PageSwitch := _Reset;
 InstConfig := _Set;
 Page := 1;
 gotoxy (2,1);write (FileName);
 gotoxy (28,1);writeln ('Ladder Diagram Page:',Page,'/',DiagramPage);
 writeln;
 Begin_Rung;
 Read_Cursor;
 XCor_Temp := XCor; YCor_Temp := YCor;
 Repeat
 begin
   Cursor (Off);
   gotoxy (3,24);write ('-] [-    -]/[-    -( )-    -(/)-    TIM    CNT        |        ---      FUN
   Cursor (On);
   XCor := XCor_Temp; YCor := YCor_Temp;
   gotoxy (XCor,YCor);
   Repeat
   until Keypressed;
   Ch := ReadKey;
   If Ch = Return then New_Line;
   If (Ch = #0) AND Keypressed then
   begin
     Ch := ReadKey;
     Case ch of

    Up     : begin
               Read_Cursor;
               YCor := YCor - 2;
               gotoxy (XCor,YCor);
             end;

    Down   : begin
               Read_Cursor;
               YCor := YCor + 2;
               gotoxy (XCor,YCor);
             end;
```

```
Left  : begin
          Read_Cursor;
          XCor := ((XCor DIV 8)*8)+2;
          XCor := XCor - 8;
          gotoxy (XCor,YCor);
        end;

Right : begin
          Read_Cursor;
          XCor := ((XCor DIV 8)*8)+2;
          XCor := XCor + 8;
          gotoxy (XCor,YCor);
        end;

F3    : begin
          Read_Cursor;
          XCor_Temp := XCor;
          YCor_Temp := YCor;
          Push_IO (Out_Pattern);
          Out (XCor,YCor,IOassign);
          XCor := XCor_Temp;
          YCor := YCor_Temp;
          gotoxy (XCor,YCor);
          LinkLeft_check;
          Link_End (74);
          New_Line;
        end;

F4    : begin
          Read_Cursor;
          XCor_Temp := XCor;
          YCor_Temp := YCor;
          Push_IO (Out_Not_Pattern);
          Out_Not (XCor,YCor,IOassign);
          XCor := XCor_Temp;
          YCor := YCor_Temp;
          gotoxy (XCor,YCor);
          LinkLeft_check;
          Link_End (74);
          New_Line;
        end;

F5    : begin
          Read_Cursor;
          XCor_Temp1 := XCor;
          YCor_Temp1 := YCor;
          Push_IO (Timer_Pattern);
          Push_Data ('Data#:');
          Timer (XCor,YCor,TimCnt_Dta,IOassign);
          XCor := XCor_Temp1;
          YCor := YCor_Temp1;
          gotoxy (XCor,YCor);
          LinkLeft_check;
```

```
            Link_End (68);
            New_Line;
          end;

    F6    : begin
            Read_Cursor;
            XCor_Temp1 := XCor;
            YCor_Temp1 := YCor;
            Push_IO (Counter_Pattern);
            Push_Data ('Data#:');
            Counter (XCor,YCor,TimCnt_Dta,IOassign);
            XCor := XCor_Temp1;
            YCor := YCor_Temp1;
            gotoxy (XCor,YCor);
            LinkLeft_check;
            Link_End (68);
            New_Line;
          end;

    F1    : begin
            Read_Cursor;
            If Xcor = 2 then Link (2) else
            begin
              LinkLeft_check;
              Read_Cursor;
              gotoxy (XCor - 1,YCor);
              GetColor (Cursor_Ch);
              If Cursor_Ch IN [$C4,$00,$20] then
              begin
                gotoxy (XCor,YCor);
                Link (2);
              end;
            end;
            Push_IO (Relay_Pattern);
            Relay (XCor,YCor,IOassign);
          end;

    F2    : begin
            Read_Cursor;
            If Xcor = 2 then Link (2) else
            begin
              LinkLeft_check;
              Read_Cursor;
              gotoxy (XCor - 1,YCor);
              GetColor (Cursor_Ch);
              If Cursor_Ch IN [$C4,$00,$20] then
              begin
                gotoxy (XCor,YCor);
                Link (2);
              end;
            end;
            Push_IO (Relay_Not_Pattern);
            Relay_Not (XCor,YCor,IOassign);
          end;
```

```
      F7    : begin
                Read_Cursor;
                Link_Up;
              end;

      F8    : begin
                If XCor = 66 then
                begin
                  Link (2);
                  XCor := 66;
                  gotoxy (XCor,YCor);
                end else
                Dash_Line;
              end;

      F10   : Command;

      F9    : PC_Function;

      InsKey,DelKey: Delete;

      PgUp  : PageUp;
      PgDw  : PageDown;

      end; {Case}
    end;
    Read_Cursor;
    XCor_Temp := XCor; YCor_Temp := YCor;
end;
until Ch = Esc;
clrscr;
end;
```