โครงงาน

เครื่องควบคุมที่โปรแกรมได้

PROGRAMMABLE LOGIC CONTROLLER

โดย

นาย ณวงค์ จิรารัตนกูลชัย 30.1070

นาย ธวัช เชิญสิริสิน 30.1090

อาจารย์ที่ปรึกษา

อาจารย์ วัชระ ฉัตรวิริยะ

คณะวิศวกรรมศาสตร์ ภาควิชา คอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2533

โครงงานปีการศึกษา 2533

สาขา คอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง .... เครื่องควบคุมที่โปรแกรมได้

ผู้จัดทำ .....

1. นาย ณรงค์ จิรารัตน์กุลชัย 30.1070

2. นาย ธวัช เชิญสิริสิน 30.1090

(_____)

อาจารย์ วัชระ ฉัตรวิริยะ อาจารย์ที่ปรึกษา

โครงงาน

เครื่องควบคุมที่สามารถโปรแกรมได้

(Programmable Logic Controller)

วัตถุประสงค์โดยทั่วไป

1.สามารถนำไปใช้แทน PLCของต่างประเทศที่มีราคาแพงในอนาคตได้ โดยลดต้นทุนในการผลิตให้ต่ำ แต่ความสามารถและคุณภาพใกล้เคียงของต่างประเทศ

2.สามารถพัฒนาเทคโนโลยีขึ้นมาใช้เอง ทำให้พัฒนาความสามารถทางวิชาการ

ขอบข่ายของ โครงงาน

1. หน่วยป้อนโปรแกรม และแสดงผล
   - 1.1 คีย์บอร์ด
     - numberic keys 16 keys
     - instruction keys 8 keys
     - operation keys 6 keys
     - three mode operation keys
   - 1.2 ส่วนแสดงผล ด้วย LCD
2. หน่วยอินพุต (INPUT MODULE)
3. หน่วยเอาต์พุต (OUTPUT MODULE)
4. หน่วยประมวลผล (CPU MODULE)

หัวข้อโครงงาน	เครื่องควบคุมที่โปรแกรมได้		
นักศึกษา			
	นาย ณรงค์	จิรารัตนกุลชัย	30.1070
	นาย อวัช	เชิญวิริสิน	30.1090
อาจารย์ที่ปรึกษา	อาจารย์ วิชระ	ฉัตรวิริยะ	
ระดับการศึกษา	วิศวกรรมศาสตร์		
ปีการศึกษา	2533		

บทคัดย่อ

เครื่องควบคุมที่โปรแกรมได้ ที่ใช้ในการควบคุมกระบวนการทางอุตสาหกรรม เพื่อให้เกิดประสิทธิภาพทั้งในด้านการควบคุม ประหยัดพลังงาน และการซ่อมบำรุงรักษา โดยมีส่วนควบคุม และหน่วยย่อนโปรแกรมที่แยกจากกันได้ ทั้งยังสามารถตรวจสอบสภาวะการทำงาน และผลของกระบวนการได้ด้วย

PROJECT Title	Programmable Logic Controller	
Name		
	NARONG GIRARATANAKUNCHAI	30.1070
	THAWAT CHANSIRISIN	30.1090
PROJECT Advisor	WATCHARA CHARTVIRIYA	
Level Of Study	BACHELOR'S DEGREE IN COMPUTER	
Academic Year	1990	

ABSTRACT

PROGRAMMABLE LOGIC CONTROLLER TO CONTROL INDUSTRAIL PROCESSING IN ORDER TO BE HIGH EFFICIENCY CONTROL, SAVE ENERGY AND EASY TO REPAIR. THERE ARE MANY PARTS TO SEPARATE THAT CONTROLLER MODULE AND PROGRAMMER MODULE. IT CAN TO TEST PROCESS AS WELL AS TO DETECT ITS RESOUCE.

สารบัญ

บทที่ 1 บทนำ

PLCs (Programmable logic Controllers)

หรือเรียกอีกอย่างหนึ่งว่า เครื่องควบคุมที่สามารถโปรแกรมได้ เรียกทั่ว ๆ ไปว่า PLCs เป็นอุปกรณ์ที่ถูกผลิตขึ้นปลายปี ค.ศ. 1960 เพื่อนำมาใช้งานแทนระบบควบคุมแบบเก่า โดยในตอนแรกเริ่มแรกนั้น PLCs ถูกประดิษฐ์เพื่อใช้งานอย่างง่าย ๆ โดยการโปรแกรม ส่วนอินพุตและเอาต์พุตของ PLCs จะมีลักษณะเป็นแบบดิจิตอล คือ มี ON กับ OFF สัญญาณ ON และ OFF เหล่านี้จะถูกนำไปใช้ควบคุมอุปกรณ์ภายนอก เช่น รีเลย์ หรือหน่วยแสดงผลอื่น ๆ ดังนั้น คำว่า "Programmable logic Controllers" จึงเป็นที่รู้จักอย่างกว้างขวาง

ปัจจุบันนี้ PLCs ได้ถูกพัฒนาขึ้นมาอย่างรวดเร็ว โดยมีความสามารถสูงขึ้น สามารถทำให้งานควบคุมทั้งทางด้านอนาล็อก และดิจิตอล โดยที่ขนาดของ PLCs มิได้ใหญ่โตขึ้น อุปกรณ์ส่วนใหญ่ของ PLCs เป็นอุปกรณ์พวกไมโครอิเล็กทรอนิกส์ต่าง ๆ โดยมี ส่วนประมวลผลกลาง (CPU)เป็นไมโครโปรเซสเซอร์ ควบคุมการทำงานภายในอาจจะเป็น IC ตระกูล'51 หรือของ Motorola ตระกูล 68 เป็นต้น

นอกจากนี้ยังมีการนำเอา PLCs มาเชื่อมต่อกับเครื่องคอมพิวเตอร์ ตลอดจนอุปกรณ์ควบคุมอื่น ๆ ทำให้ PLCs มีความสามารถในการใช้งานมากขึ้น (โดยสามารถทำ Self Diagnestic Self test ตลอดจน Communication Ling ฯลฯ) ดังนั้น PLCs จึงกลายเป็นหัวใจสำคัญสำหรับงานอุตสาหกรรมสมัยใหม่ ด้วยเหตุผลหลัก ๆ คือ

ประการแรก – การใช้งาน สามารถทำได้โดยการเขียนโปรแกรมเป็นภาษาง่าย ๆ โดยที่ตัวโปรแกรมสามารถดัดแปลงแก้ไขได้ และถูกเก็บเอาไว้ใน PLCs

ประการที่สอง – รูปร่างลักษณะภายนอก ได้ถูกออกแบบมาให้ใช้งานได้อย่างสะดวกเหมาะสมกับสภาพของไวงงานอุตสาหกรรมในปัจจุบัน

อย่างไรก็ตาม แม้ว่าในแถบยุโรป และอเมริกา PLCs จะเป็นที่นิยมอย่างสูงสุด แต่ประเทศไทย ค่อนข้างจะเป็นของใหม่ (ที่มีขายอยู่ก็ราคาแพง) การใช้งานก็ยังไม่กว้างขวางนัก

การควบคุมกระบวนการทางอุตสาหกรรมส่วนใหญ่จะเป็นการนำอุปกรณ์ไฟฟ้าเชิงกล เช่น รีเลย์ (relay) ตัวนับ (counter) ตัวตั้งเวลา (timer) มาต่อร่วมกันเป็นระบบควบคุม แต่การใช้วิธีการดังกล่าวนี้ จะมีข้อเสียมากเนื่องจากอุปกรณ์ไฟฟ้าเชิงกลจะสิ้นเปลืองกำลังงานสูง และมีอายุการใช้งานค่อนข้างสั้น นอกจากนั้นเมื่อต้องการเปลี่ยนแปลงลำดับการทำงานของกระบวนการก็จะต้องแก้ไขระบบควบคุมใหม่ทำให้ยุ่งยากและเสียเวลามาก หรือในกรณีที่มีการเปลี่ยน

แปลงมาก ๆ อาจจะต้องออกแบบระบบควบคุมใหม่ทั้งหมด

การประยุกต์ใช้ไมโครโปรเซสเซอร์มาใช้ในการควบคุมกระบวนการ เป็นวิธีหนึ่งที่มี ประสิทธิภาพสูง เนื่องจากสามารถเปลี่ยนแปลงลำดับขั้นการทำงานหรือการควบคุมได้โดยการเปลี่ยนแปลงหรือแก้ไขโปรแกรม ดังนั้นในปัจจุบันจึงมีการใช้ไมโครโปรเซสเซอร์ หรือไมโครคอมพิวเตอร์ ในการควบคุมกระบวนการกันอย่างแพร่หลายอย่างไรก็ดีการแก้ไข หรือเปลี่ยนแปลงโปรแกรมนั้น จะต้องกระทำโดยผู้ที่เข้าใจโครงสร้างทางฮาร์ดแวร์ (HARDWARE) ของระบบควบคุม หรือกระบวนการเป็นอย่างดี ดังนั้นผู้ปฏิบัติงาน หรือผู้ใช้มักจะไม่สามารถแก้ไข หรือเปลี่ยนแปลงโปรแกรมดังกล่าวได้เอง ดังนั้นจึงได้มีการพัฒนาเครื่องควบคุมที่โปรแกรมได้ (PROGRAMM ABLE LOGIC CONTROLLER) ขึ้น ซึ่งผู้ปฏิบัติงาน หรือผู้ใช้สามารถที่จะเปลี่ยนแปลง หรือแก้ไขโปรแกรมการควบคุมได้เองโดยใช้ภาษาที่กำหนด เครื่องควบคุมที่โปรแกรมได้ส่วนใหญ่จะใช้ภาษาสำหรับทำการโปรแกรม เช่น ภาษาโปรแกรมแบบแลดเดอร์ไดอะแกรม โปรแกรมแบบบูลลีน หรือ ภาษาชั้นสูงต่าง ๆ

โครงงานฉบับนี้ ได้นำเสนอเครื่องควบคุมที่โปรแกรมได้ ซึ่งออกแบบเทคโนโลยีขึ้น จากความที่สามารถทางวิชาการไมโครโปรเซสเซอร์ โดยวางแนวทางให้มีความสามารถ และ คุณสมบัติใกล้เคียงกับ PLC จากต่างประเทศ สามารถป้อนโปรแกรมแบบบูลลีน โดยการแก้ไข เปลี่ยนแปลงโปรแกรมทางคีย์บอร์ด ทั้งยังแสดงผลของกระบวนการทาง LCD และ LED ให้ทราบได้

ส่วนรายละเอียดของแต่ละบท ครอบคลุมเนื้อหา ดังนี้ ในบทที่ 2 กล่าวถึง ความรู้ทั่วไปที่เกี่ยวกับ PLC เช่น หลักการทำงาน โครงสร้าง องค์ประกอบ บทที่ 3 กล่าวถึง การโปรแกรมของ PLC เช่น Logic diagram Relay diagram ในบทที่ 4 ขั้นตอนการออกแบบ ทั้ง ฮาร์ดแวร์ และ ซอฟท์แวร์ บทที่ 5 กล่าวถึงการประยุกต์ใช้งาน โดยมีกรณีศึกษายกมาประกอบ และ บทที่ 6 เป็นคู่มือการใช้เครื่อง ส่วน รายละเอียดด้าน ฮาร์ดแวร์ และซอฟท์แวร์ รวมทั้ง ไฟลชาร์ทของระบบ ที่กล่าวถึง อยู่ในส่วนของภาคผนวก

บทที่ 2 ความรู้ทั่วไปเกี่ยวกับเครื่องควบคุมที่โปรแกรมได้

เครื่องควบคุมที่โปรแกรมได้ เป็นการประยุกต์ใช้เทคโนโลยีทางด้านไมโครโปรเซสเซอร์ทั้งทางด้านฮาร์ดแวร์ (Hardware) และ ซอฟท์แวร์ (Software) มาใช้ในการควบคุมเครื่องจักรอัตโนมัติ หรือกระบวนการทางอุตสาหกรรม ทำให้การควบคุมกระบวนการ หรือเครื่องจักรต่างๆ เหล่านั้นเป็นไปอย่างมีประสิทธิภาพ นอกจากนั้นยังสามารถเปลี่ยนแปลงรูปแบบของการควบคุมและสามารถพัฒนาขีดความสามารถให้สูงขึ้นได้

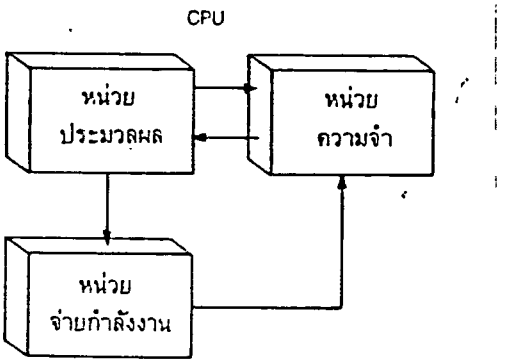
2.1 หลักการทำงานของเครื่องควบคุมที่โปรแกรมได้

เครื่องควบคุมที่โปรแกรมได้ เป็นระบบคอมพิวเตอร์ที่อินพุทและเอาท์พุทได้ถูกออกแบบให้มีสภาวะเหมาะสมกับกระบวนการที่ต้องการควบคุม และมีหน่วยประมวลผลกลางที่ทำหน้าที่ควบคุมการทำงานของระบบทั้งหมด หน่วยประมวลผลกลางหมายถึงระบบไมโครโปรเซสเซอร์ที่ทำหน้าที่ควบคุมการทำงานทั้งหมด โปรแกรมที่ใช้ควบคุมการทำงานของหน่วยประมวลกลางดังกล่าวจะถูกแบ่งออกเป็น 2 ส่วน ส่วนหนึ่งคือโปรแกรมที่ใช้ในการกำหนดลำดับการทำงานของการควบคุม ซึ่งผู้ใช้สามารถที่จะเปลี่ยนแปลงหรือแก้ไขโปรแกรมในส่วนนี้ได้ ซึ่งโปรแกรมในส่วนนี้ก็คือโปรแกรมที่บ่งบอกเงื่อนไขต่างๆ ที่ใช้สำหรับการควบคุม โปรแกรมอีกส่วนหนึ่งคือโปรแกรมที่ใช้ในการควบคุมระบบ โปรแกรมส่วนนี้จะทำหน้าที่ควบคุมการรับค่าสภาวะอินพุท ซึ่งได้มาจากกระบวนการ แล้วนำมาประมวลผลทางเอาท์พุท ตามลำดับขั้นตอนของโปรแกรมในส่วนแรก โครงสร้างของเครื่องควบคุมแสดงดังรูปที่ 2.1

รูปที่ 2.1 โครงสร้างของเครื่องควบคุมที่โปรแกรมได้

หน่วยประมวลผลกลาง (Central Processing Unit : CPU)

หน่วยประมวลผลกลางประกอบด้วย ระบบไมโครโปรเซสเซอร์ซึ่งทำหน้าที่ในการประมวลผลข้อมูล หน่วยความจำสำหรับเก็บโปรแกรมของผู้ใช้ และหน่วยจ่ายกำลังไฟฟ้าดังรูปที่ 2.2



รูปที่. 2.2 โครงสร้างของหน่วยประมวลผลกลาง

ในเครื่องควบคุมบางแบบอาจจะรวมหน่วยประมวลผลกลาง หน่วยความจำและหน่วยจ่ายกำลังไฟฟ้าเข้าไว้ด้วยกันเพื่อให้มีขนาดเล็กกระทัดรัด แต่บางแบบก็อาจจะแยกกันออกเป็นส่วนๆ เพื่อสะดวกในการบำรุงรักษา ระบบไมโครโปรเซสเซอร์ ซึ่งทำหน้าที่ในการประมวลผลเป็นส่วนประกอบที่สำคัญของเครื่องควบคุม มีหน้าที่ในการประมวลผลข้อมูล การคำนวณทางคณิตศาสตร์ และทางลอจิก รวมทั้งควบคุมการทำงานของส่วนต่างๆทั้งหมด

ในปัจจุบันเทคโนโลยีทางด้านไมโครโปรเซสเซอร์ได้พัฒนาขึ้นมาก จึงได้มีการนำเอาไมโครโปรเซสเซอร์หลายๆตัวมาทำงานร่วมกัน (Multi-Processor) เพื่อให้มีการประมวลผลได้รวดเร็วและมีประสิทธิภาพยิ่งขึ้น โดยแยกการทำงานของไมโครโปรเซสเซอร์แต่ละตัวเป็นอิสระต่อกันแต่มีการแลกเปลี่ยนข้อมูลกันอยู่ตลอดเวลา นอกจากนี้ในหน่วยอินพุทหรือเอาพ์พุทบางชนิดก็จะมีไมโครโปรเซสเซอร์แยกอิสระจากเครื่องควบคุมอีกที่หนึ่ง เช่น หน่วยอินพุทเอาพ์พุทแบบ PID ซึ่งสามารถทำงานได้ด้วยตัวเอง

ค่าสภาวะต่างๆ ถูกนำมาจากกระบวนการโดยผ่านทางหน่วยอินพุท ผ่านการประมวลผล จากนั้น ผลที่ได้จะถูกส่งออกทางเอาพ์พุท เพื่อไปควบคุมการทำงานของกระบวนการ ตามต้องการ และ จะย้อนกลับมารับค่าสภาวะจากกระบวนการ เพื่อนำไปประมวลผล และส่งผลออกทางเอาพ์พุทอีก จะมีการทำอย่างนี้ตลอดไป การรับค่าสภาวะจากภายนอก มาทำการประมวลผลและส่งค่าออกทางเอาพ์พุทในแต่ละครั้งเรียกว่าการสแกน (Scanning) หรือหนึ่งรอบของการทำงาน ระยะเวลาในการสแกนหนึ่งรอบ (1 Scan Time) จะมีค่าเท่ากับระยะเวลาของการอ่านค่าสภาวะอินพุทครั้งแรกจนกระทั่งมีการอ่านค่าสภาวะอินพุทครั้งต่อไป ดังรูปที่. 2.3 ส่วนรายละเอียดของแต่ละหน่วยได้อธิบายตามลำดับ ในเนื้อหาต่อไป

รูปที่ . 2.3 ระยะเวลาการสแกนหนึ่งรอบ

ระยะเวลาในการสแกนจะขึ้นอยู่กับความยาวของโปรแกรมที่ผู้ใช้งานโปรแกรมเงื่อนไขการทำงานเข้าไปเพื่อควบคุม ซึ่งผู้ผลิตมักจะกำหนดไว้ว่าเครื่องควบคุมนั้นๆ สามารถทำโปรแกรมได้สูงสุดเพียงใด และการใช้ชุดคำสั่งในการโปรแกรมนั้นแต่ละคำสั่งใช้เวลาในการประมวลผลเท่าใด นอกจากนั้นก็จะขึ้นอยู่กับสมรรถนะของเครื่องควบคุมเอง รวมทั้งอุปกรณ์ภายนอกที่ติดต่ออยู่กับเครื่องควบคุม เนื่องจากการควบคุมกระบวนการ จะต้องกระทำได้อย่างรวดเร็ว และ แม่นยำ อย่างไร ก็ตามถ้าเครื่องควบคุมมีค่าระยะเวลาการสแกนที่มากเกินไป อาจจะทำให้การควบคุมเกิดการผิดพลาดขึ้นได้เพราะว่าค่าสภาวะบางค่าอาจจะมีการเปลี่ยนแปลงที่เร็วมากๆ ซึ่งถ้าการเปลี่ยนแปลงนี้มีค่าเวลาที่สั้นกว่าค่าระยะเวลาของการสแกนแล้วจะทำให้ข้อมูลที่ได้รับ เพื่อใช้ในการประมวลผลผิดพลาดได้

แต่ในทางปฏิบัติแล้ว การเปลี่ยนแปลงค่าสภาวะอินพุทของกระบวนการ มักจะมีระยะเวลาที่ยาวกว่าเวลาการสแกนของเครื่องควบคุม เนื่องจากอุปกรณ์ที่เป็นอินพุทส่วนใหญ่จะเป็นอุปกรณ์ไฟฟ้าเชิงกล หรือสวิทช์ชนิดต่างๆ ซึ่งมีค่าระยะเวลาการเปลี่ยนแปลงที่ช้ามาก นอกจากจะเป็นค่าสภาวะที่ได้รับมาจากอุปกรณ์อิเลคทรอนิคส์ ซึ่งมีความเร็วในการสวิทช์สูง ในการใช้งานลักษณะนี้ เครื่องควบคุมจำเป็นจะต้องมีอุปกรณ์พิเศษ เพื่อช่วยในการรับค่าสภาวะมาทำการประมวลผลขั้นหนึ่งก่อน บางแบบอาจจะใช้วิธีอินเตอร์รัพท์ (Interrupt) เพื่อให้เครื่องควบคุมมาอ่านข้อมูลไปประมวลผลทันที หรือนำข้อมูลส่งออกไปเอาพ์พุทในทันที เป็นต้น

หน่วยความจำในหน่วยประมวลผลกลางจะถูกแบ่งออกเป็น 2 ส่วนใหญ่ๆ คือ ส่วนแรกใช้ในการเก็บโปรแกรมและข้อมูลสำหรับการควบคุมระบบ และอีกส่วนหนึ่งจะใช้ในการเก็บโปรแกรมของผู้ใช้ และข้อมูลสภาวะอินพุทเอาพ์พุท ในเครื่องควบคุมจะมีหน่วยความจำอยู่จำกัดและไม่สามารถขยายได้เหมือนกับคอมพิวเตอร์ เครื่องควบคุมขนาดใหญ่จะมีหน่วยความจุสูง แต่จะมีราคาแพง และเครื่องควบคุมขนาดเล็ก ก็จะมีความจุของหน่วยความจำต่ำ ซึ่งอาจจะไม่เพียงพอต่อการใช้งาน ดังนั้นการเลือกใช้เครื่องควบคุมจะต้องคำนึงถึงสิ่งเหล่านี้ด้วย

2.2 โครงสร้างของ PLC

PLCs มีลักษณะเหมือนกับไมโครคอมพิวเตอร์ ซึ่งอุปกรณ์ส่วนใหญ่ได้แก่ CPU, Memory, I/P, O/P และหน่วยจ่ายไฟตรง ข้อแตกต่างหลักระหว่าง PLCs กับไมโครคอมพิวเตอร์ คือ

- เทคนิคการโปรแกรม

ดังนั้นการผลิต และ การค้นคว้า PLCs จึงได้ถูกพัฒนาให้ใช้ควบคู่ไปกับไมโครคอมพิวเตอร์ เป็นการขยายขีดความสามารถของไมโครคอมพิวเตอร์ให้สูงขึ้นอีก ส่วน CPU คือ หัวใจของ PLCs จะทำการ Execute คำสั่งต่าง ๆ จากหน่วยความจำ (memory) หน่วยความจำที่ใช้อยู่ในปัจจุบันมีทั้งแบบ EPROM, EEPROM และ RAM ซึ่งแบบ RAM ต้องการใช้แบตเตอรี่ที่เป็นแหล่งจ่ายไฟสำรอง สำหรับสัญญาณที่ใช้อยู่ ก็มีระดับตั้งแต่ 5VAC-230VAC ในการแสดงผลของ I/P และ O/P จะถูกเปลี่ยนแปลงหรือไม่ ขึ้นอยู่กับโปรแกรมควบคุมระบบนั้น ๆ โดยทั่วไปแล้ว I/P จะถูกอ่านเข้ามาเมื่อเริ่ม Scanning ส่วน O/P จะถูกกำหนดในช่วงท้ายของการ Scanning ในส่วนที่ซับซ้อนของ PLCs คือ เป็นการทำงานที่ต้องมีการคำนวนเลขคณิต ส่วนการควบคุมระบบอุปกรณ์ I/O ของ PLCs นั้นจะมีการติดต่อกับอุปกรณ์ในการใช้งานทั่วไป เช่น Proximity Switch, Limit Switch, LED, สเตรนเกจ เทอร์โมคับเปิล และ อุปกรณ์ interface ต่าง ๆ เช่น Printer, Barcode, Recorder

ใน I/O modules ที่ผลิตขึ้นมาใหม่ ๆ จะมี Microprocessors Programmed ซึ่งผู้ผลิตเครื่อง เป็นผู้ทำขึ้นโดยสามารถ Process ข้อมูลก่อน และ หลังการทำงานของ PLCs Processor ซึ่งทำให้เกิดการทำงานที่มีประสิทธิภาพมากขึ้น ตัวอย่างที่เห็นกันง่าย ๆ เช่น การควบคุม Steppier meter ที่มีความเร็วสูงเท่ากับ 25,000 steps/second ในส่วนของ I/O ที่มี Processor จัดการควบคุมทางด้านนี้แทนโดยที่ PLCs ก็ควบคุมการทำงานทั้งหมดของระบบโดยเฉพาะอย่างยิ่งงาน PID Controller นั้น การ Process Signal จะทำที่ Micro Process ของ PLCs แทนที่จะใช้การสุ่มตัวอย่าง (Sanpling-Speed) ซึ่งทำให้การทำงานมีความเที่ยงตรงและแน่นอน

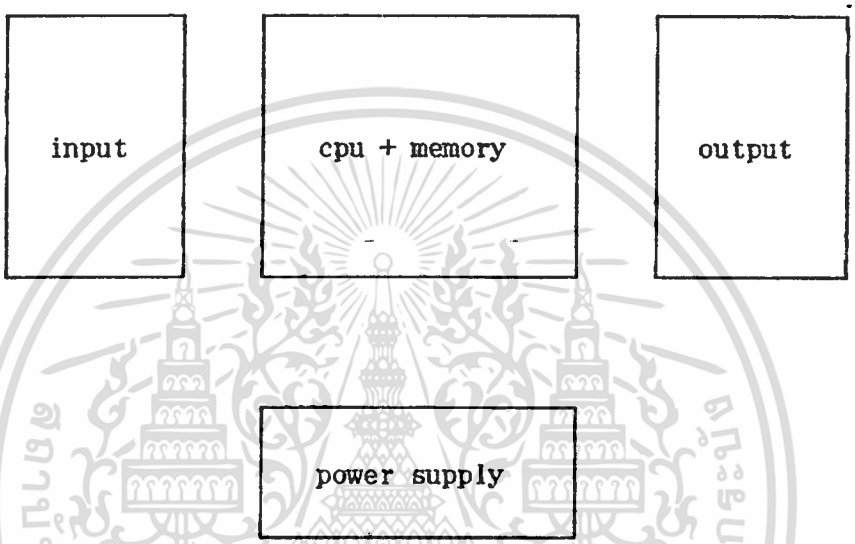
<u>การสแกน</u> (Scanning)

การทำงาน โดยการรับค่าสภาวะต่าง ๆ ของเครื่องจักร หรือกระบวนการผลิตผ่านทางหน่วยอินพุต ประมวลผลตามโปรแกรมของผู้ใช้ที่เก็บไว้ในหน่วยความจำ หลังจากนั้น จึงนำผลลัพธ์ที่ได้ส่งไปควบคุมเครื่องจักรทางหน่วยเอาต์พุต

หน่วยป้อนโปรแกรม (Programming Unit)

ทำหน้าที่ติดต่อระหว่าง PC กับผู้ใช้ รับโปรแกรมที่เขียนขึ้นเก็บไว้ในหน่วยความจำ ปกติหน่วยป้อนโปรแกรม จะต่อเชื่อมกับ PLCs เมื่อผู้ใช้ต้องการป้อน ตรวจสอบ หรือแก้ไขโปรแกรมเท่านั้น

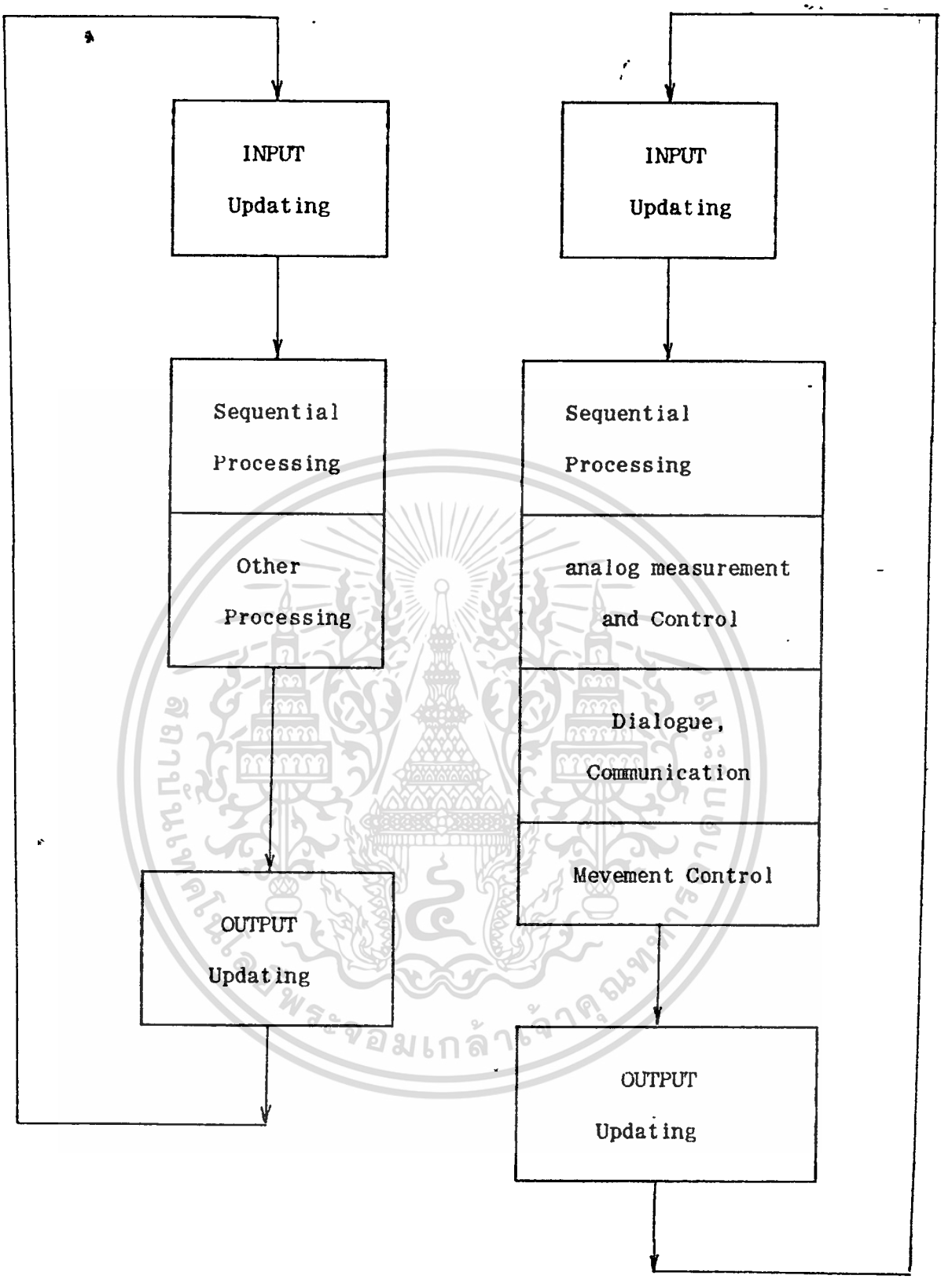
และ PLCs สามารถทำงานได้โดยไม่ต้องพึ่งหน่วยป้อนโปรแกรม ดังนั้นหน่วยป้อนโปรแกรม จึงไม่ได้ถูกจัดเป็นส่วนประกอบของ PLCs



รูปที่ 2.4 ส่วนประกอบหลักของ PLC

แบ่งได้ดังนี้

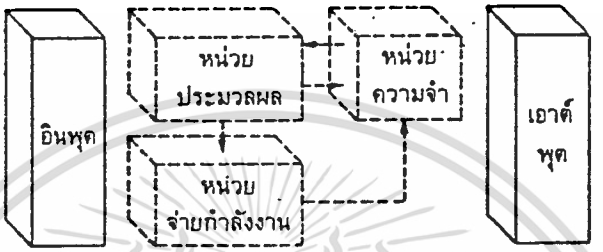
- cpu+memory module
- input module (ส่วนคีย์บอร์ดและแสดงผลอยู่ในส่วนนี้ด้วย)
- output module



รูปที่ 2.5 ขั้นตอนการทำงานของ PLCS

2.3 หน่วยอินพุท/หน่วยเอาต์พุต

หน่วยอินพุต/เอาต์พุต ทำหน้าที่ติดต่อระหว่าง PC กับอุปกรณ์ภายนอก หน่วยอินพุตทำหน้าที่รับสภาวะและค่าวัดจากอุปกรณ์ภายนอกที่มีการเปลี่ยนแปลง เช่น การ ON/OFF ของสวิตช์ตำแหน่งเครื่องจักร ระดับของเหลว อุณหภูมิ ความดัน ระดับแรงดันและกระแสไฟฟ้าส่งต่อให้ PC CPU จะใช้ค่าหรือสภาวะจากหน่วยอินพุต/เอาต์พุต เป็นข้อมูลในการประมวลผลตามโปรแกรมผู้ใช้ และส่งผลที่ได้ไปที่หน่วยเอาต์พุต เพื่อควบคุมอุปกรณ์ภายนอก เช่น รีเลย์ มอเตอร์ไฟฟ้า ปั๊ม และ วาล์ว



รูปที่. 2.6 หน่วยอินพุต/เอาต์พุต

ระยะแรก PC มีหน่วยอินพุต/เอาต์พุตแบบสภาวะลอจิก (discrete input/output) เพียงชนิดเดียว ทำให้ PC ถูกใช้ในการควบคุมแบบ ON/OFF ซึ่งเป็นส่วนหนึ่งของระบบควบคุมทั้งหมดเท่านั้น ปัจจุบันหน่วยอินพุต/เอาต์พุตแบบตัวเลข (numerical data input/output) ทำให้ขอบเขตการใช้ PC ในการควบคุมกว้างขึ้น ทั้งการควบคุมเครื่องจักรและกระบวนการอุตสาหกรรม ดังรายละเอียดต่อไปนี้

2.3.1.หน่วยอินพุต/เอาพุทแบบสภาวะลอจิก

หน่วยอินพุท/เอาท์พุตชนิดนี้ ทำหน้าที่ติดต่อกับอุปกรณ์ภายนอกที่มีการเปลี่ยนแปลงเพียง 2 สภาวะ เช่น การ ON/OFF ของสวิตช์ไฟฟ้า หรือหน้าสัมผัสของรีเลย์

2.3.1.1 สัญญาณมาตรฐานของอุปราณ์อินพุต/เอาต์พุตแบบสภาวะลอจิก

สัญญาณไฟฟ้าที่อุปกรณ์อินพุต/เอาต์พุตแบบสภาวะลอจิกใช้มีหลายค่า ทั้งไฟฟ้ากระแสตรงและกระแสสลับ สัญญาณมาตรฐานที่หน่วยอินพุต/เอาต์พุตของ PC ใช้เชื่อมต่อกับอุปกรณ์ภายนอกได้แสดงในตารางที่ 2.1

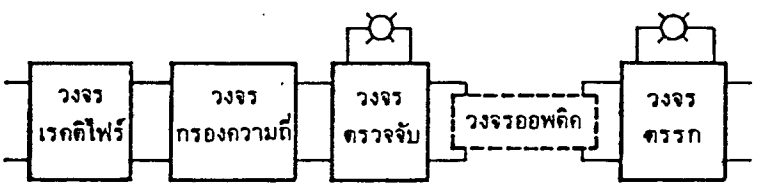
ตาราง 2.1 สัญญาณมาตรฐานของอุปกรณ์อินพุต/เอาต์พุตแบบสภาวะลอจิก

อุปกรณ์อินพุต	อุปกรณ์เอาต์พุต
24 VAC/DC	12-48 VAC/DC
48 VAC/DC	120 VAC/DC
120 VAC/DC	220 VAC/DC
220 VAC/DC	หน้าสัมผัสรีเลย์
ระดับ TTL	ระดับ TTL

หน่วยอินพุตแลลสภาวะลอจิก ทำหน้าที่รับค่าทางไฟฟ้าของอุปกรณ์ภายนอกและเปลี่นยเป็นสภาวะทางลอจิกเพื่อเก็บไว้ในหน่วยความจำ ส่วนตารางอินพุตของ CPU บิตที่มีค่า "1" หมายถึง การ ON หรือปิดวงจรไฟฟ้า และบิตที่มีค่า "0" หมายถึง การ OFF หรือเปิดวงจรไฟฟ้าของอุปกรณ์อินพุต หน่วยเอาต์พุตจะรับสภาวะลอจิกจากตารางเอาต์พุตของ CPU และเปลี่ยนเป็นค่าทางไฟฟ้าเพื่อควบคุมอุปกรณ์ภายนอกอีกทีหนึ่ง บิตที่มีค่า "1" หมายถึงการ ON หรือการต่อวงจรไฟฟ้า และบิตที่มีค่า "0" หมายถึงการ OFF หรือการตัดวงจรไฟฟ้า การเลือกใช้หน่วยอินพุต/เอาต์พุตชนิดที่ถูกต้องและเหมาะสม จำเป็นต้องเข้าใจคุณลักษณะ และการทำงานของหน่วยอินพุต/เอาต์พุตแต่ละชนิด

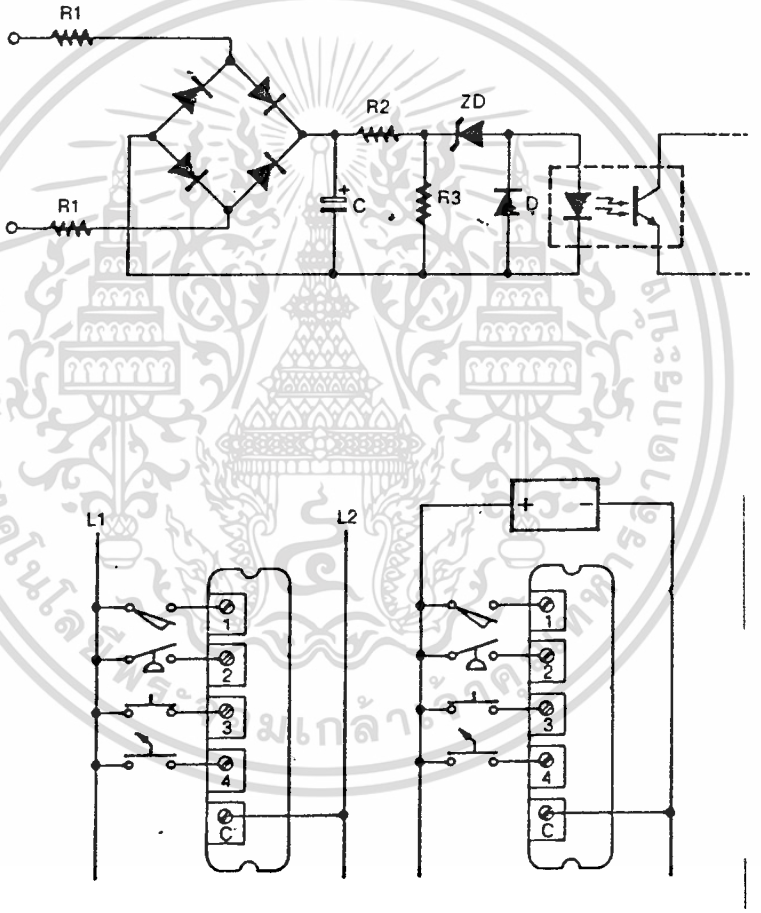
2.3.1.2 หน่วยอินพุตแบบ AC/DC

หน่วยอินพุตแบบ AC/DC ทำหน้าที่ รับสัญญาณไฟฟ้าจากอุปกรณ์อินพุต เปรียบเทียบระดับแรงดันไฟฟ้าและเปลี่ยนเป็นสภาวะทางลอจิกส่งไปยัง CPU หน่วยอินพุตแบบ AC/DC ประกอบด้วย วงจรเรคติไฟร์ (rectifier) วงจรกรองความถี่ (filter) วงจรตรวจจับ (detector) และวงจรตรรกที่ทำหน้าที่ส่งข้อมูล



รูปที่ 2.7 หน่วยอินพุตแบบ AC/DC

วงจรเรคติไฟร์ ทำหน้าที่เปลี่ยนสัญญาณ AC หรือ DC เป็นระดับสัญญาณ DC วงจรกรองความถี่ ทำหน้าที่กำจัดสัญญาณรบกวน วงจรตรวจจับ ทำหน้าที่เปรียบเทียบ ระดับสัญญาณเพื่อตรวจสอบสภาวะของอุปกรณ์อินพุตและวงจรตรรกทำหน้าที่ติดต่อกับ CPUระหว่างวงจรตรรกกับวงจรส่วนอื่นจะใช้การเชื่อมต่อแบบออพติค (optical coupling) เพื่อป้องกัน CPU จากการรบกวนของอุปกรณ์ภายนอก ดังในรูป และ แสดงโครงสร้าง และวงจรของหน่วยอินพุตแบบ AC/DC หน่วยอินพุตชนิดนี้ ต้องใช้ เวลาตรวจสอบสภาวะของอุปกรณ์อินพุตประมาณ 9 ถึง 25 ms. เนื่องจากการทำงานของวงจรเรคติ ไฟร์และวงจรกรองความถี่ หน่วยอินพุตชนิดนี้จะมีสัญญาณไฟแจ้งสภาวะของอุปกรณ์อินพุตให้ผู้ใช้ทราบ รูปที่ 2.8 ได้แสดงการเชื่อมต่อหน่วยอินพุตแบบ AC/DC กับอุปกรณ์ภายนอก

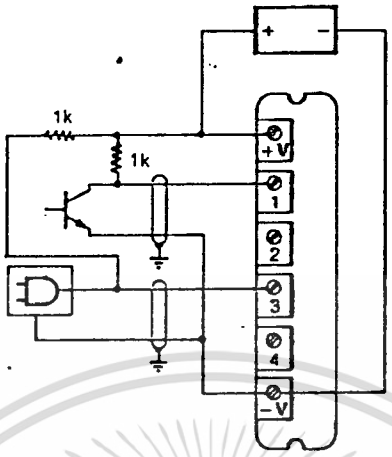


รูปที่. 2.8 การเชื่อมต่อหน่วยอินพุตแบบ AC/DC กับอุปกรณ์ภายนอก

2.3.1.3 หน่วยอินพุตแบบ TTL

หน่วยอินพุตแบบ TTL ทำหน้าที่ รับสัญญาณอินพุตจากอุปกรณ์อิเล็กทรอนิกส์ชนิด TTL (Transister Transister Logic) หรือ อุปกรณ์ที่ใช้สัญญาณไฟฟ้า 5 VDC หน่วยอิน

พุดชนิดนี้ไม่จำเป็นต้องมีวงจรเรคติไฟร์และวงจรกรองความถี่ จึงทำงานได้เร็วกว่าหน่วยอินพุตแบบ AC/DC คือ ใช้เวลาเพียง 1 ถึง 3 ms. รูปที่ 2.9 แสดงการเชื่อมต่อหน่วยอินพุตแบบ TTL



รูปที่ 2.9 แสดงการเชื่อมต่อหน่วยอินพุตแบบ TTL

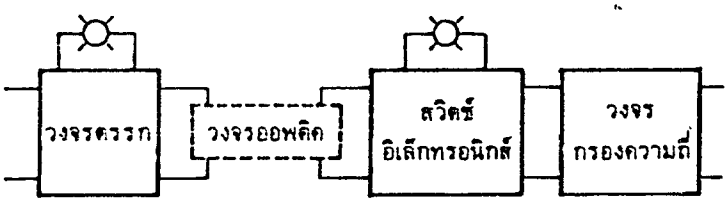
2.3.1.4 หน่วยอินพุตแบบหน้าสัมผัส (Contact input)

หน่วยอินพุตแบบหน้าสัมผัส ทำหน้าที่ ตรวจสอบสภาวะของอุปกรณ์อินพุตที่ไม่มีแรงดันไฟฟ้าในตัว เช่นสวิตช์ไฟฟ้า และหน้าสัมผัสของรีเลย์ โดยหน่วยอินพุตชนิดนี้จะใช้แรงดันไฟฟ้า 12 หรือ 24 VDC จ่ายให้อุปกรณ์ภายนอก

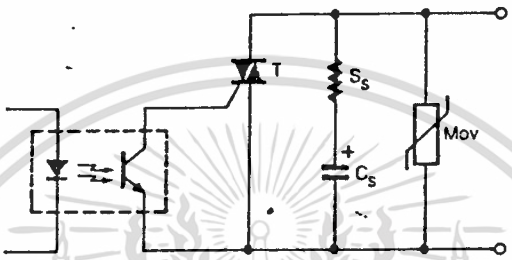
2.3.1.5 หน่วยเอาต์พุตแบบ AC

หน่วยเอาต์พุตแบบ AC ทำหน้าที่รับสภาวะการควบคุมจาก CPU เปลี่ยนเป็นระดับแรงดันไฟฟ้า กระแสสลับเพื่อควบคุมการทำงานของอุปกรณ์เอาต์พุต หน่วยเอาต์พุตแบบ AC ประกอบด้วย วงจรตรรก ทำหน้าที่ติดต่อกับ CPU วงจรเชื่อมต่อแบบออพติค สวิตช์อินเล็กทรอนิกส์ (electronics switch) และวงจรกรองความถี่ รูปที่ 2.10 แสดงส่วนประกอบ และวงจรของหน่วยเอาต์พุตแบบ AC

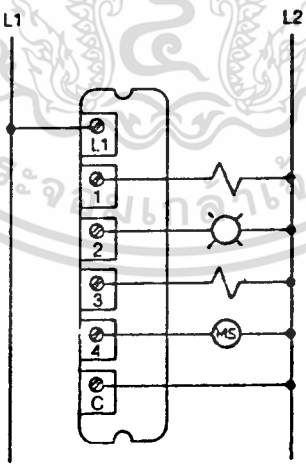
วงจรตรรกรับสัญญาณควบคุมจาก CPU และควบคุมการทำงานของสวิตช์อิเล็กทรอนิกส์ โดยใช้การเชื่อมต่อแบบออพติค และวงจรกรองความถี่ทำหน้าที่ปรับแรงดันเอาต์พุตให้สม่ำเสมอ สวิตช์อิเล็กทรอนิกส์ในหน่วยเอาต์พุตชนิดนี้มักใช้ ไทร์แอค (triac) หรือ SCR ซึ่งเปิดและปิดวงจรไฟฟ้าโดยไม่เกิดประกายไฟเช่น หน้าสัมผัสของรีเลย์ จึงไม่รบกวนการทำงานของอุปกรณ์ภายนอกและยืดอายุการใช้งาน หน่วยเอาต์พุตชนิดนี้มักมีสัญญาณไฟแจ้งสภาวะการ ON หรือ OFF ให้ผู้ใช้ทราบ รูปที่ 2.11 แสดงการเชื่อมต่อหน่วยเอาต์พุตแบบ AC กับอุปกรณ์ภายนอก



รูปที่ 2.10 แสดงส่วนประกอบ



วงจรของ หน่วยเอาต์พุตแบบ AC

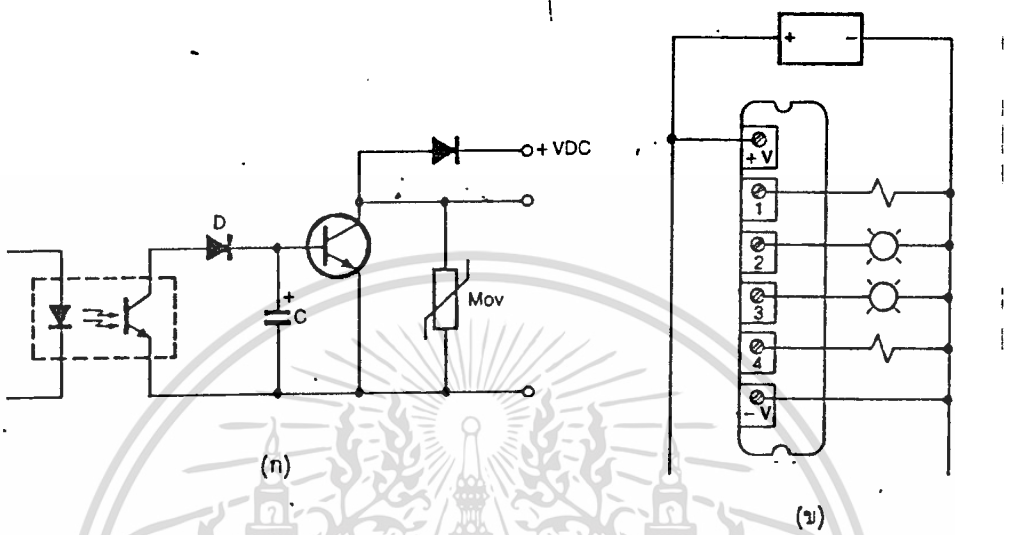


รูปที่ 2.11 แสดงการเชื่อมต่อหน่วยเอาต์พุตแบบ AC กับอุปกรณ์ภายนอก

2.3.1.6 หน่วยเอาต์พุตแบบ DC

หน่วยเอาต์พุตแบบ DC ทำหน้าที่ ควบคุมอุปกรณ์เอาต์พุตที่ทำงานด้วยสัญญาณไฟฟ้ากระ

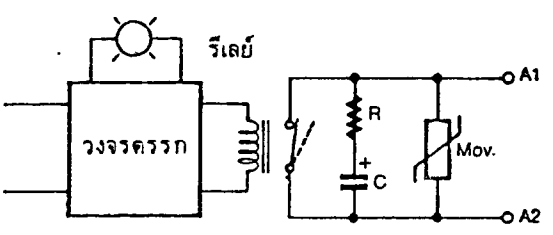
แสดงลักษณะและการทำงานของวงจรคล้ายกับหน่วยเอาต์พุตแบบ AC แต่ใช้ทรานซิสเตอร์เปิดและปิดวงจรไฟฟ้าแทนไทร์แอค และ SCR



รูปที่ 2.12 วงจรหน่วยเอาต์พุตแบบ DC และ การเชื่อมต่อกับอุปกรณ์ภายนอก

2.3.1.7 หน่วยเอาต์พุตแบบหน้าสัมผัส (Contact output)

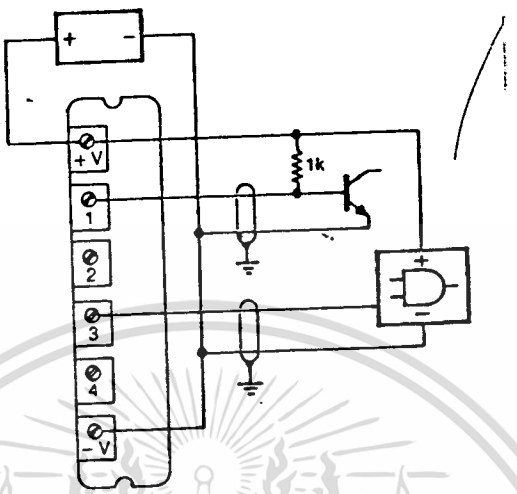
หน่วยเอาต์พุตแบบหน้าสัมผัส ทำหน้าที่แทนหน้าสัมผัสของรีเลย์ในการควบคุมอุปกรณ์ ที่มีแรงดันไฟฟ้าของตนเอง ลักษณะและการทำงานของวงจรคล้ายกับหน่วยเอาต์พุตแบบ AC และ DC แต่ใช้หน้าสัมผัสและรีเลย์เปิดและปิดวงจรไฟฟ้าแทนอุปกรณ์อิเล็กทรอนิกส์ หน่วยเอาต์พุตชนิดนี้ใช้กับอุปกรณ์ไฟฟ้ากระแสตรงหรือกระแสสลับก็ได้



รูปที่ 2.13 วงจรหน่วยเอาต์พุตแบบหน้าสัมผัส

2.3.1.8 หน่วยเอาต์พุตแบบ TTL

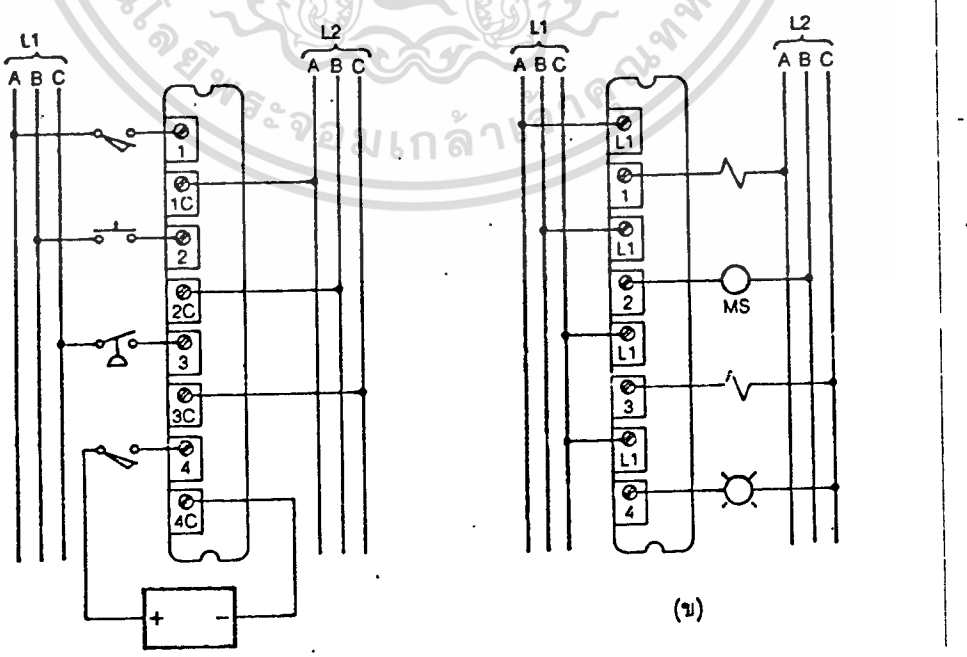
หน่วยอินพุตแบบTTL ทำหน้าที่ควบคุมการทำงานของอุปกรณ์อิเล็กทรอนิกส์ที่เป็น TTL หรืออุปกรณ์ที่ใช้สัญญาณไฟฟ้า 5 VDC หน่วยเอาต์พุตชนิดนี้มักต้องการหน่วยจ่ายกำลังงานขนาด 5 VDC จากภายนอก



รูปที่ 2.14 หน่วยเอาต์พุตแบบ TTL

2.3.1.9 หน่วยอินพุต/เอาต์พุตแบบอิสระ (Isolated input/output)

หน่วยอินพุต/เอาต์พุตแบบอิสระ ทำหน้าที่คล้ายกับหน่วยอินพุต/เอาต์พุตแบบหน้าสัมผัส แต่แยกวงจรไฟฟ้าของอุปกรณ์ทั้งหมดออกจากกัน ทำให้อุปกรณ์ทุกชิ้นมีวงจรไฟฟ้าและหน่วยจ่ายกำลังงานของตนเอง



รูปที่ 2.15 หน่วยอินพุต/เอาต์พุตแบบอิสระ

2.3.2. หน่วยอินพุต/เอาต์พุตแบบตัวเลข

ไมโครโปรเซสเซอร์ทำให้ PC สามารถ คำนวณทางคณิตศาสตร์ ตรวจสอบและเคลื่อนย้ายข้อมูลที่เป็นตัวเลข หน่วยอินพุต/เอาต์พุตแบบจำนวนเลขทำให้ PC สามารถรับข้อมูลจากอุปกรณ์วัด และควบคุมเครื่องจักรหรือกระบวนการอุตสาหกรรม ระบบควบคุมที่ใช้ PC จึงครบวงจรทั้งการควบคุมแบบ ON/OFF และอะนาลอก

หน่วยอินพุต/เอาต์พุตแบบตัวเลขมี 2 ชนิด คือ แบบรหัสเลขฐานสอง และ แบบอะนาลอก หน่วยอินพุต/เอาต์พุตแบบรหัสเลขฐานสองทำงานคล้ายกับแบบสภาวะลอจิก แต่การรับและส่งข้อมูลทำได้พร้อมกันครั้งละหลายบิต ข้อมูลอาจอยู่ในรูป รหัส ASCII รหัส BCD หรือรหัส GRAY หน่วยอินพุต/เอาต์พุตแบบอะนาลอก ทำหน้าที่รับ และส่งข้อมูลที่เป็นสัญญาณไฟฟ้า ทำให้ PC สามารถติดต่อกับอุปกรณ์วัดและควบคุมในกระบวนการอุตสาหกรรม โดยตารางที่ ๘๗๕๒ ได้แสดงตัวอย่างอุปกรณ์อินพุต/ เอาต์พุตแบบจำนวนเลขทั้ง 2 ชนิด

2.3.2.1 สัญญาณมาตรฐานของอุปกรณ์อินพุต/เอาต์พุตแบบตัวเลข

สัญญาณไฟฟ้าของอุปกรณ์อินพุต/เอาต์พุตแบบอะนาลอกอาจมีทั้งค่าบวกและลบหรือค่าบวกหรือลบอย่างใดอย่างหนึ่ง ซึ่งผู้ใช้สามารถปรับเลือกขนาดและชนิดของสัญญาณได้ โดยใช้วิธีทางซอฟต์แวร์หรือ ฮาร์ดแวร์ตามที่บริษัทผู้ผลิตกำหนด ตารางที่ 2.2 สำหรับหน่วยอินพุต/เอาต์พุตแบบรหัสเลขฐานสองมักจะใช้สัญญาณชนิดเดียวกับอุปกรณ์อินพุต/เอาต์พุตแลลสภาวะลอจิก

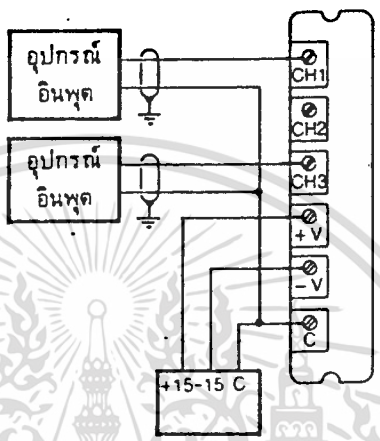
อุปกรณ์อินพุต	อุปกรณ์เอาต์พุต
4 – 20 mADC	4 – 20 mADC
0 – 1 VDC	10 – 50 mADC
0 – 5 VDC	0 – 5 VDC
0 – 10 VDC	0 – 10 VDC
1 – 5 VDC	±25 VDC
±5 VDC	±5 VDC
±10 VDC	±10 VDC

ตารางที่ 2.2 สัญญาณมาตรฐานของอุปกรณ์อินพุต/เอาต์พุตแบบอะนาลอก

2.3.2.2 หน่วยอินพุตแบบอะนาลอก

หน่วยอินพุตแบบอะนาลอก ทำหน้าที่รับสัญญาณอินพุตจากอุปกรณ์ภายนอก ปรับระดับให้เหมาะสม และใช้วงจรแปลงสัญญาณอะนาลอก/ดิจิตอล หรือ ADC (Analog to Degital

Converter) เปลี่ยนสัญญาณอะนาลอกเป็นค่าทางดิจิตอลส่งให้ CPU วงจรส่วนแรกของหน่วยอินพุตคือ วงจรกันชน และวงจรกรองความถี่เพื่อลดสัญญาณรบกวนจากภายนอก วงจรกันชนทำหน้าที่เพิ่มค่าอิมพิแดนซ์ด้านอินพุต (input impedance) ให้สูงขึ้นเพื่อเชื่อมต่อกับอุปกรณ์อินพุตที่มีอิมพิแดนซ์สูง สายส่งที่เชื่อมต่อระหว่างหน่วยอินพุตและอุปกรณ์ภายนอกควรเป็นสายโคแอคเชียลเพื่อป้องกันการรบกวนจากสนามไฟฟ้าและรักษาค่าอิมพิแดนซ์ให้คงที่ การเชื่อมต่อหน่วยอินพุตแบบอะนาลอกกับอุปกรณ์ภายนอกได้แสดงในรูปที่ 2.16



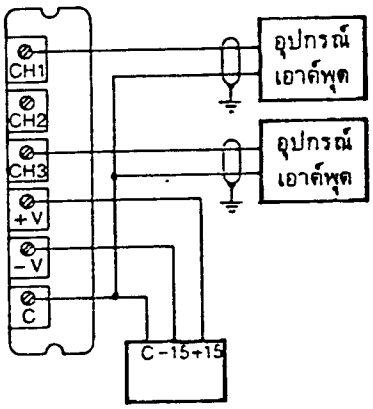
รูปที่ 2.16 การเชื่อมต่อหน่วยอินพุตแบบอะนาลอกกับอุปกรณ์ภายนอก

2.3.2.3 หน่วยเอาต์พุตแบบอะนาลอก

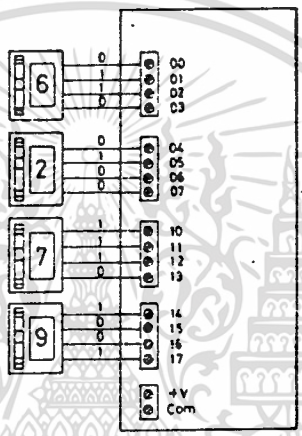
หน่วยเอาต์พุตแบบอะนาลอกทำหน้าที่รับข้อจาก CPU เปลี่ยนเป็นระดับสัญญาณไฟฟ้า โดยใช้วงจรแปลงสัญญาณดิจิตอล/อะนาลอก หรือ DAC (Digital to Analog Converter) เพื่อควบคุมอุปกรณ์ภายนอกหน่วยเอาต์พุตชนิดนี้ใช้การเชื่อมต่อแบบออพติค เพื่อป้องกันการรบกวน CPU จากอุปกรณ์ภายนอก การเชื่อมต่ออุปกรณ์เอาต์พุตแบบอะนาลอกกับ PC ในรูปที่ 2.17 ต้องใช้หน่วยจ่ายกำลังงานภายนอกเพื่อให้วงจร DAC ทำงาน

2.3.2.4 หน่วยอินพุตแบบรีจิสเตอร์ (Register input)

หน่วยอินพุตแบบรีจิสเตอร์ ทำหน้าที่รับข้อมูลรหัสเลขฐานสอง เช่น รหัส BCD และรหัส GRAY จากอุปกรณ์ภายนอก เช่น สวิตช์รหัส ส่งให้ CPU อุปกรณ์อินพุตมักใช้สัญญาณไฟฟ้าขนาด 5 VDC หรือ ระดับ TTL โดยสัญญาณไฟฟ้า 0 VDC หมายถึงสภาวะลอจิก "0" และ 5 VDC หมายถึงสภาวะลอจิก "1" การรับข้อมูลของหน่วยอินพุตมีจำนวนครั้งละ 16 หรือ 32 บิต รูปที่ 2.18 แสดงการเชื่อมต่อหน่วยอินพุตแบบรีจิสเตอร์



รูปที่ 2.17 การเชื่อมต่ออุปกรณ์เอาต์พุตแบบอะนาลอก



รูปที่ 2.18 การเชื่อมต่อหน่วยอินพุตแบบรีจิสเตอร์

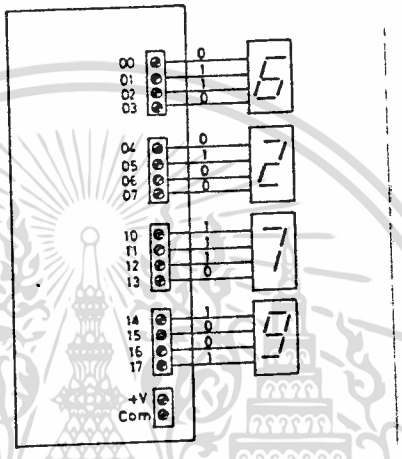
2.3.2.5 หน่วยเอาต์พุตแบบรีจิสเตอร์ (Register output)

หน่วยเอาต์พุตแบบรีจิสเตอร์ ทำหน้าที่ส่งข้อมูลรหัสเลขฐานสอง เช่น รหัส BCD รหัส GRAY จาก CPU ส่งให้อุปกรณ์เอาต์พุต เช่น ตัวแสดงผลเจ็ดส่วน และจอภาพ สัญญาณไฟฟ้าของหน่วยเอาต์พุต มีขนาด 5 ถึง 30 VDC และจ่ายกระแสไฟฟ้า ได้ ประมาณ 0.5 แอมแปร์ รูปที่ 2.19 แสดงการเชื่อมต่อหน่วยเอาต์พุตแบบรีจิสเตอร์

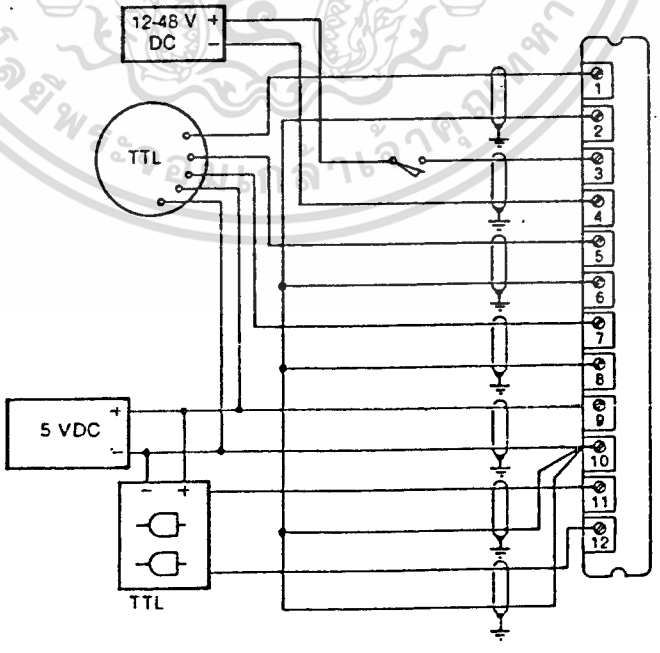
2.3.2.6 หน่วยเชื่อมต่อแบบวงจรนับและเข้ารหัส

หน่วยเชื่อมต่อแบบวงจรนับและเข้ารหัสทำหน้าที่เป็นวงจรนับความเร็วสูงการทำงานอิสระจาก CPU วงจรนับจะทำการนับทันทีที่ได้รับสัญญาณพัลส์จากภายนอก หน่วยเชื่อมต่อชนิดนี้ใช้ตรวจนับจำนวนสินค้าและตรวจสอบตำแหน่งเครื่องจักร วงจรนับจะติดต่อรับคำสั่งเริ่มต้น หยุด และยกเลิกการทำงาน ค่าเริ่มต้นและค่าสุดท้ายในการนับจาก CPU ขณะที่วงจรนับทำงานจะส่งผลการนับ

ให้วงจรเข้ารหัสเปลี่ยนค่าเป็น รหัส BCD หรือ รหัส GRAY ส่งไปยัง CPU ทุกช่วงการสแกน เมื่อนับได้จำนวนที่ต้องการ หน่วยเชื่อมต่อจะส่งสัญญาณควบคุมไปยังอุปกรณ์ภายนอก และแจ้งให้ CPU ทราบ ความเร็วในการนับตั้งแต่ 100 Hz ถึง 50 kHz รูปที่ 2.20 แสดงการเชื่อมต่อวงจรนับและเข้ารหัส หน่วยเชื่อมต่อ ต้องใช้หน่วยจ่ายกำลังงานจากภายนอก หน่วยเชื่อมต่อรับสัญญาณอินพุตขนาด 48 VDC และ ใช้สัญญาณเอาต์พุตขนาด 24 VDC หรือ ระดับ TTL ถ้าการเชื่อมต่อมีระยะเกินกว่า 50 ฟุต ต้องใช้สายส่งแบบ โคแอคเชียลเพื่อป้องกันการรบกวนจากสนามไฟฟ้า



รูปที่ 2.19 การเชื่อมต่อหน่วยเอาต์พุตแบบรีจิสเตอร์



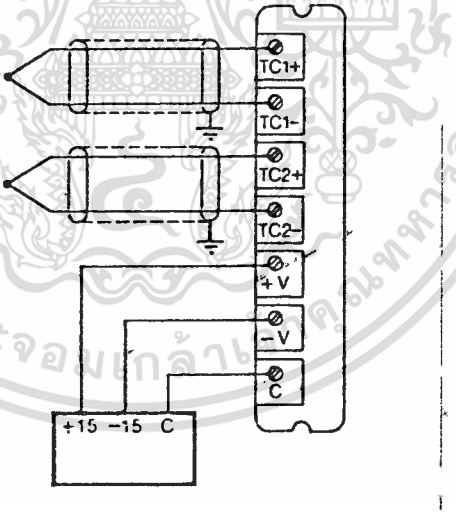
รูปที่ 2.20 แสดงการเชื่อมต่อวงจรนับและเข้ารหัส

2.3.3. หน่วยเชื่อมต่อแบบพิเศษ

PC ส่วนใหญ่ใช้หน่วยอินพุต/เอาต์พุตแบบสภาวะลอจิกและแบบตัวเลข ในการตรวจสอบและควบคุมอุปกรณ์ภายนอก ทั้งเครื่องจักรและกระบวนอุตสาหกรรม นอกจากหน่วยอินพุต/เอาต์พุต ทั้ง 2 ชนิด การควบคุมบางอย่างอาจต้องใช้หน่วยเชื่อมต่อพิเศษ เช่น หน่วยอินพุตที่ใช้กับสัญญาณระดับต่ำ หรือมีความเร็วสูงมาก ๆ หน่วยอินพุต/เอาต์พุตที่ใช้ไมโครโปรเซสเซอร์สำหรับงานควบคุมที่ต้องใช้การตัดสินใจอิสระจาก PC และทำงานร่วมกับ CPU ในลักษณะของระบบมัลติโปรเซสเซอร์ หน่วนเชื่อมต่อแบบพิเศษนี้มีใช้ใน PC ขนาดกลาง และขนาดใหญ่เท่านั้น

2.3.3.1 หน่วยอินพุตแบบเทอร์โมคัปเปิล (Thermocouple input)

หน่วยอินพุตแบบเทอร์โมคัปเปิลทำหน้าที่ติดต่อกับ CPU เพื่อรับสัญญาณอินพุตจาก เทอร์โมคัปเปิง ที่มีระดับสัญญาณต่ำมาก เพียงประมาณ 50 mVDC เท่านั้น หน่วยอินพุตประกอบด้วยวงจรชดเชยอุณหภูมิ วงจรปรับและขยายสัญญาณให้เหมาะสม และวงจร ADC การใช้หน่วยอินพุตชนิดนี้ต้องทราบชนิดของเทอร์โมคัปเปิลที่ใช้ก่อนเสมอ เพราะเทอร์โมคัปเปิลแต่ละชนิดมีคุณลักษณะต่างกัน



รูปที่ 2.21 การเชื่อมต่ออินพุตแบบเทอร์โมคัปเปิล

2.3.3.2 หน่วยอินพุตความเร็วสูง (High speed input)

หน่วยอินพุตความเร็วสูง ทำหน้าที่ตรวจจับสัญญาณพัลส์ที่มีความเร็วสูง และช่วงกว้างสั้นมากจนหน่วยอินพุตแบบสภาวะลอจิกตรวจสอบไม่ได้ หน่วยอินพุตชนิดนี้ จะสอดแทรกการทำงานของ CPU ทันทีที่ตรวจพบสัญญาณพัลส์จากภายนอก สัญญาณพัลส์ที่มีขนาด 10 ถึง 24 VDC และมี

ความกว้างประมาณ 50 ถึง 100 us. ซึ่งเร็วกว่าการสแกนตามปกติของ PC มาก จะสามารถกระตุ้นให้หน่วยอินพุตชนิดนี้ทำงาน

2.3.3.3 หน่วย ASCII

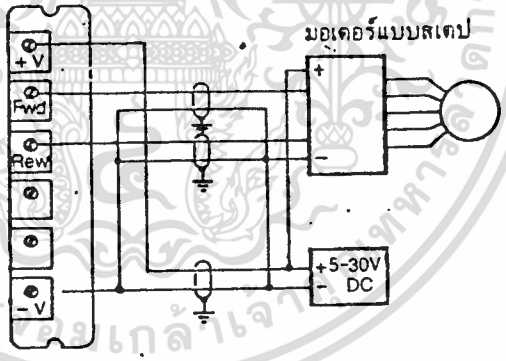
หน่วย ASCII ทำหน้าที่ ติดต่อ รับส่งข้อมูลที่เป็นรหัส ASCII ระหว่าง PC กับอุปกรณ์ร่วมภายนอก เช่น จอภาพ เครื่องพิมพ์ การรับและส่งข้อมูลมีหลายวิธี เช่น RS-232 RS-422 และลูปกระแส (current loop) 20 mADC

2.3.3.4 หน่วยอินพุตแบบสเตรนเกจ (Strain gage input)

หน่วยอินพุตแบบสเตรนเกจ ทำหน้าที่วัดค่าความเครียด(strain) จากสเตรนเกจ

2.3.3.5 หน่วยเชื่อมต่อมอเตอร์แบบสเตป (Stepper motor interface)

ทำหน้าที่ควบคุมมอเตอร์แบบสเตป ใช้จำนวนสัญญาณพัลส์ กำหนดระยะการหมุน ความถี่กำหนดความเร็ว และการเปลี่ยนแปลงความถี่กำหนดความเร่งและความหน่วง



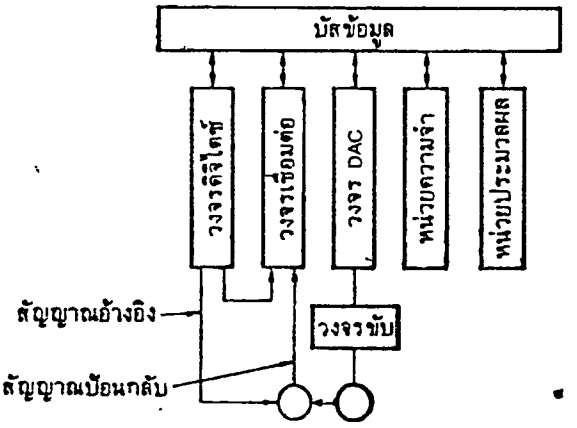
รูปที่ 2.22 หน่วยเชื่อมต่อมอเตอร์แบบสเตป

2.3.3.6 หน่วยเชื่อมต่อเซอร์โว (Servo interface)

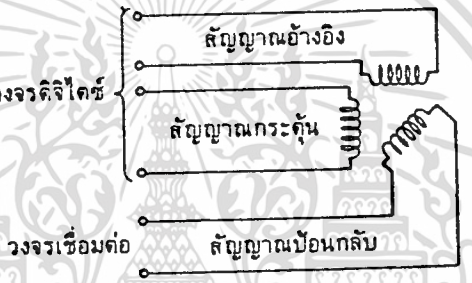
ใช้ในการควบคุมระบบคลัตช์และเกียร์ หรือตำแหน่งเครื่องจักร แทนคอมพิวเตอร์ควบคุมเครื่องจักรเชิงเลข (CNC :Computer Numerical Control)

ประกอบด้วยการทำงาน 2 ส่วน คือ

- สัญญาณอ้างอิงที่ถูกส่งออกจากหน่วยเชื่อมต่อ
- สัญญาณที่ถูกป้อนกลับจากอุปกรณ์ภายนอก



รูปที่ 2.23 ตัวอย่างของการเชื่อมต่อเซอร์โว

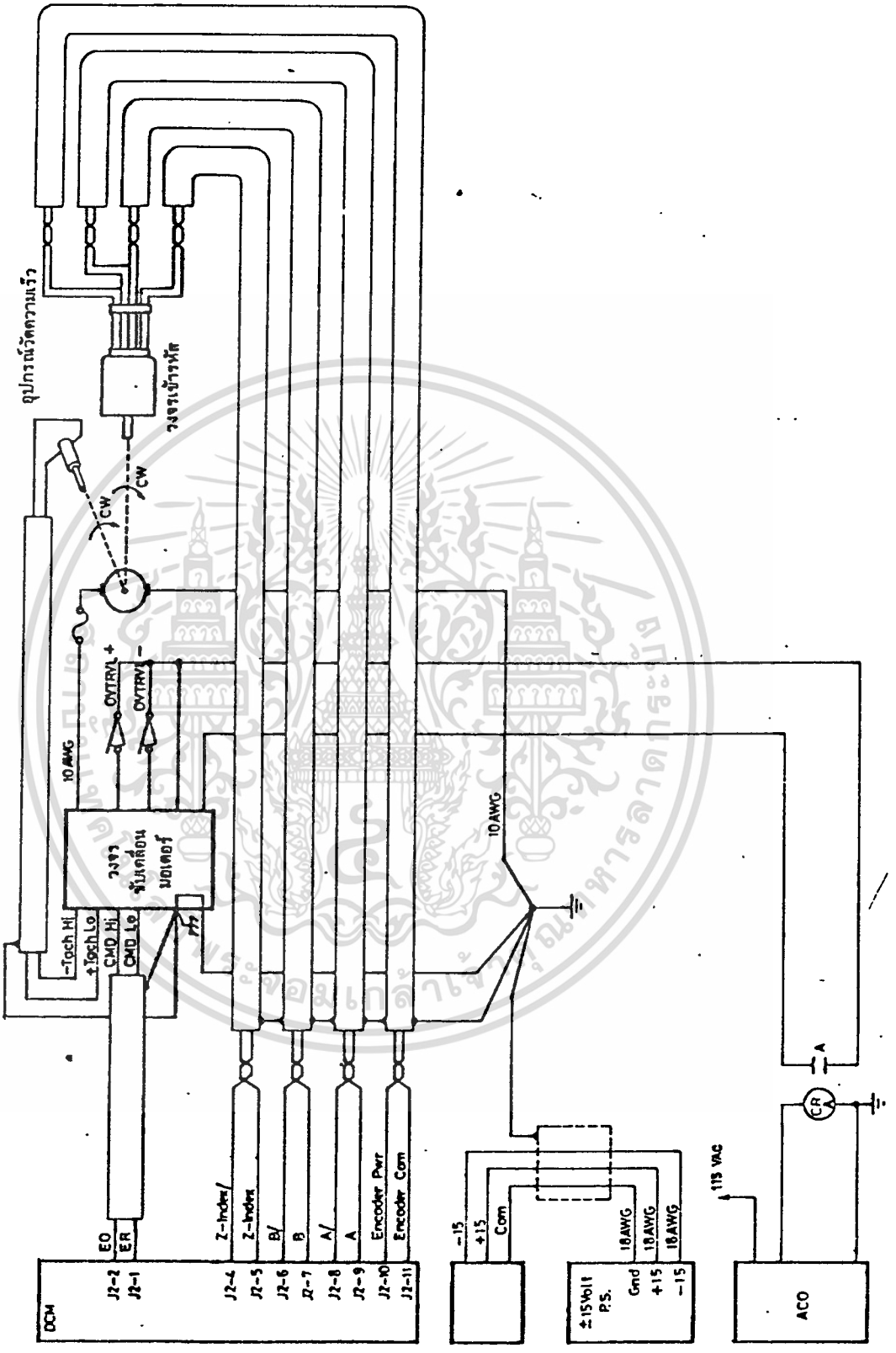


รูปที่ 2.24 วงจรดิจิไตซ์

รูปที่ 2.25 สัญญาณที่สร้างขึ้นโดยวงจรดิจิไตซ์

2.3.3.7 หน่วยเชื่อมต่อ FDM (Feedrate Motor) และ DCM (DC Motor)

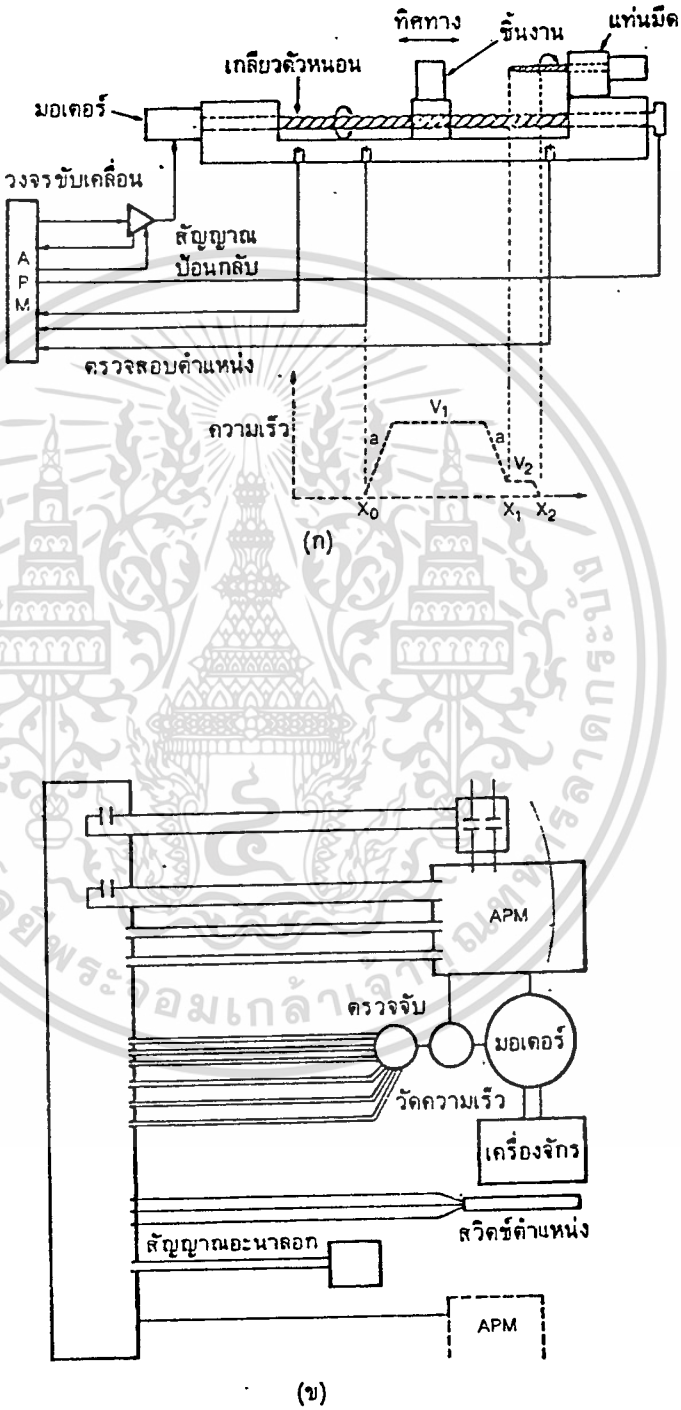
ควบคุมระบบเซอร์โว หน่วยFDM ส่งสัญญาณพัลส์แจ้งตำแหน่งเซอร์โวให้หน่วย DCM หน่วย DCM จะเปรียบเทียบตำแหน่งของเซอร์โวกับจุดที่ต้องการ และส่งสัญญาณควบคุมผ่านวงจร DAC แสดงในรูปที่ 2.26



รูปที่ 2.26 การเชื่อมต่อหน่วย FDM และ DCM

2.3.3.8 หน่วย APM (Axis Positioning Module)

หน่วย APM ทำหน้าที่ ควบคุมระบบเซอร์โวให้ทำงานเป็นลำดับขั้นตอนที่ต้องการโดยใช้ไมโครโปรเซสเซอร์ควบคุม หน่วย APM ทำงานอิสระจาก CPU แสดงในรูปที่ 2.27



รูปที่ 2.27 ตัวอย่างการใช้หน่วย APM (ก) และการเชื่อมต่อ (ข)

2.3.3.9 หน่วย PID

หน่วย PID ทำหน้าที่เป็นเครื่องควบคุมแบบ PID ในกระบวนการอุตสาหกรรม เช่น การควบคุมเตาเผา ระดับและความดันของหม้อน้ำ อัตราการไหลของก๊าซ โดยหน่วย PID จะตรวจสอบค่าตัวแปรกระบวนการ หรือ PV (Process Variable) ที่ต้องการควบคุม ผ่านวงจร ADC เปรียบเทียบกับค่าเป้าหมาย หรือ SP (Set Point) นำผลการเปรียบเทียบมาประมวลผลหาสัญญาณควบคุม หรือ CV (Control Variable) เพื่อควบคุมกระบวนการ โดยใช้วงจร DAC ผลการทำงานของหน่วย PID จะทำให้ตัวแปรกระบวนการมีค่าเท่ากับ เป้าหมายที่ต้องการ หน่วย PID ทำงานโดยใช้หลักการ แสดงในรูปที่ 2.28

2.3.3.10 หน่วยประมวลข้อมูล (Data processing module)

คำนวณทางคณิตศาสตร์ ตรวจสอบและ เคลื่อนย้ายข้อมูล

2.3.3.11 หน่วยเชื่อมต่อโครงข่าย (Network interface module)

เชื่อมต่อโครงข่ายของ PC เข้ากับ ระบบโครงข่าย ทำให้ PC สามารถติดต่อแลกเปลี่ยนข้อมูลกับ PC ต่างระบบ คอมพิวเตอร์ ระบบควบคุมอื่น และอุปกรณ์ร่วมภายนอก เช่น จอภาพ และเครื่องพิมพ์ การรับส่งข้อมูลระหว่าง PC โดยใช้หน่วยเชื่อมต่อ โครงข่ายจะมีความเร็วสูงมาก

2.3.4. หน่วยอินพุต/เอาต์พุตแบบรีโมต

มีหน่วยประมวลผลและหน่วยจ่ายกำลังงานของตนเอง การทำงานอิสระจาก CPU การเชื่อมต่อระหว่าง CPU และหน่วยอินพุต/เอาต์พุต แบบรีโมตใช้สายส่งคู่เดียว ทำให้ค่าใช้จ่ายในการเดินสายระหว่าง PC กับหน่วยอินพุต/เอาต์พุตลดลง

การเชื่อมต่อมี 2 วิธี คือ

- การเชื่อมต่อระบบลูกโซ่ (daisy chain)
- การเชื่อมต่อระบบแบบกระจาย (star)

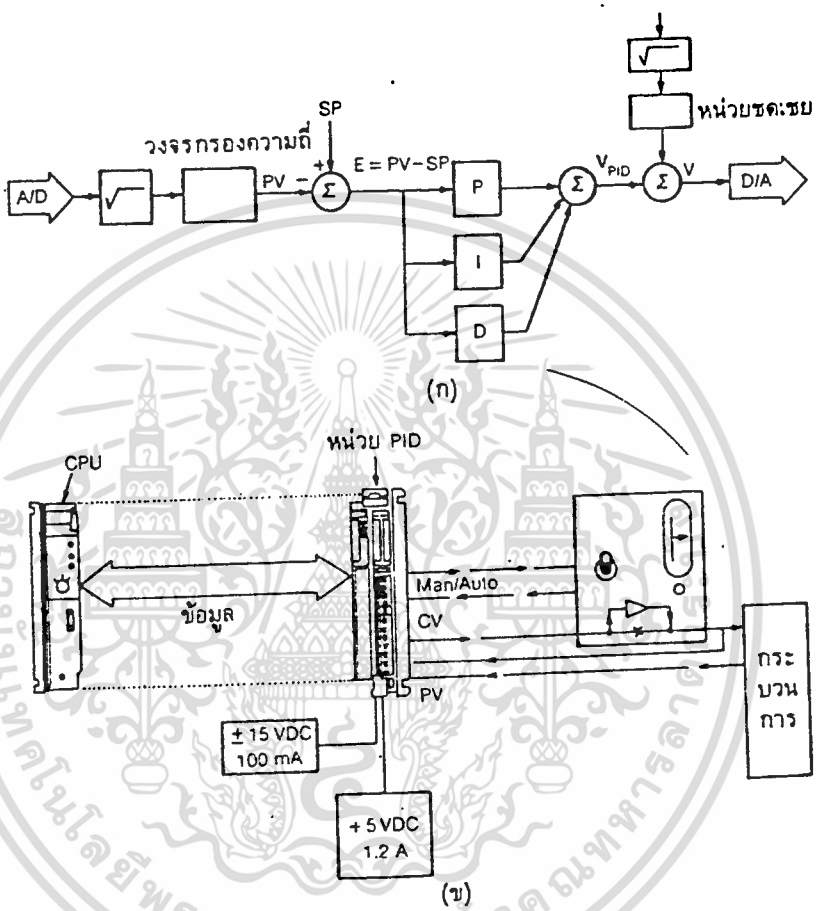
$$CV = K_P E + K_i \int E.dt + K_d .dE/dt$$

$$E = PV - SP$$

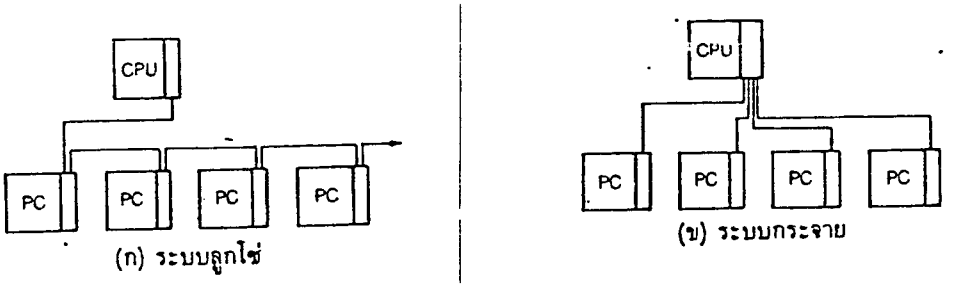
โดย K_P คือ อัตราขยายของเครื่องควบคุม (proportional gain)

K_i คือ ค่าคงที่ในการอินทิเกรต (integral gain) และ

K_d คือ ค่าคงที่ในการหาอนุพันธ์ (derivative gain)



รูปที่ 2.28 โครงสร้างของหน่วย PID (ก) และการเชื่อมต่อ (ข)



รูปที่ 2.29 การเชื่อมต่อหน่วยอินพุต/เอาต์พุตแบบรีโมต

2.4 · จุดที่จะเลือก PLC ให้เหมาะสม (Specification)

1. ประเภทของ I/O

- I/O มีลักษณะอย่างไร (AC, DC, etc)
- I/O มีจำนวนเท่าไร
- มีการ Communicate หรือไม่
  - IEEE 488
  - RS-232, 485,
  - ใกล้, ไกล
- ในอนาคตมีโอกาสจะขยายได้หรือเพิ่มเติมหรือไม่
- มีระบบแยกส่วนแรงดันไหม่ (sclater)

2. การโปรแกรม

- การโปรแกรมมีความยากง่ายเพียงไร
- หน่วยความจำของโปรแกรมมากน้อยเพียงไร มี Battery สำรองไหม
- Scanning time ประมาณเท่าไหร่
  - การทำงานเร็วมาก
  - กรณีต้องมีการประมวลผล

3. มีส่วนตรวจสอบกระแสรั่วไหม? และป้องกันไฟช็อต

2.5 การแบ่งขนาดของ PC

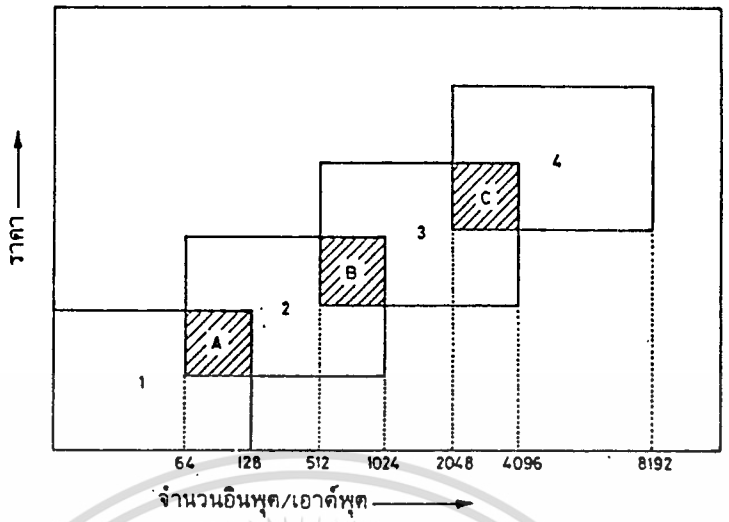
โดยปกติ PC จะมี 2 ขนาด

- PC ขนาดเล็ก หรือ PLC I/O จำกัด
- PC ขนาดใหญ่

ปัจจุบัน PC แบ่งเป็น 4 ขนาด

แบ่งตามขนาดของอินพุต/เอาต์พุต คือ

1. PC ขนาดเล็ก หรือ PLC	จำนวน I/O ไม่เกิน	128 จุด
2. PC ขนาดกลาง	"----------"	1024 จุด
3. PC ขนาดใหญ่	"----------"	4096 จุด
4. PC ขนาดใหญ่มาก	"----------"	ประมาณ 8192 จุด



พื้นที่ 1,2,3,4 หมายถึง ขนาดของ PC ที่เหมาะสมกับงานตามขนาด 10 ที่ต้องการ

พื้นที่ A,B,C หมายถึง ช่วงต่อขนาดของ PC ซึ่งขนาดของอินพุต/เอาต์พุต ไม่ใช่ปัจจัยหลักในการเลือกอีกต่อไป แต่การเลือกจะขึ้นอยู่กับ ข้อดีข้อเสีย และความสามารถพิเศษอื่น ๆ ของ PC แต่ละรุ่น

นอกจากนี้ยังมี PLC ขนาดเล็ก จุดเพียง 32 จุด เรียกว่า Micro-PLC

2.6 ข้อดีของ PLC

1. ฮาร์ดแวร์ และซอฟต์แวร์ แบ่งออกเป็น ส่วนประกอบย่อย ๆ เรียกว่า โมดูล (module) ทำงานร่วมกัน แต่ละโมดูลจะทำหน้าที่ของตนเอง แต่สามารถสลับเปลี่ยนโมดูลที่ทำหน้าที่เดียวกันได้ เพื่อความเหมาะสมกับลักษณะของงาน ทำให้การเปลี่ยนแปลงแก้ไข หรือขยายขอบเขตการใช้งานของ PLC ทั้งในแง่ของซอฟท์แวร์และฮาร์ดแวร์ เช่นเปลี่ยนแปลงขนาด I/O หน่วยความจำ ทำได้ง่าย

2. การควบคุมที่มีความคล่องตัว

ระบบควบคุม PLC ทำงานด้วยโปรแกรม ภายในหน่วยความจำซึ่งต่างจากระบบรีเลย์ที่ใช้การเดินสาย จะเปลี่ยนรูปแบบการทำงานใหม่ ก็ต้องเดินสายใหม่ PLC มีความสามารถป้อนโปรแกรมใหม่เปลี่ยนรูปแบบการทำงานได้เลย ไม่ต้องเสียเวลาในการเดินสายใหม่ นอกจากนี้ รูปแบบการทำงานได้เลย ไม่ต้องเสียเวลาในการเดินสายใหม่ นอกจากนี้ PLC ยังมีความสามารถ ตัดสินใจได้ นอกเหนือจากที่ควบคุมอุปกรณ์ให้เป็นที่ต้องการแล้ว

3. การติดตั้งทำได้ง่าย ใช้เนื้อที่น้อย เมื่อเทียบกับระบบของรีเลย์

4. บำรุงรักษาง่าย เนื่องจาก PLC ประกอบขึ้นกันเป็นลักษณะของโมดูล มีการตรวจสอบสภาพการทำงานของตนเอง ง่ายในการค้นหาจุดเสีย การซ่อมแซมเพียงแต่สับเปลี่ยนโมดูลที่เสียออก ขณะเดียวกัน PLC สามารถตรวจสอบสภาวะ ON หรือ OFF ของอุปกรณ์ภายนอก ทุกขั้นตอนการทำงานของโปรแกรม ทำให้ค้นหาจุดเสียหายในกระบวนการที่ควบคุมอยู่ทำได้ง่าย

ตารางแสดงลักษณะเปรียบเทียบข้อดีของ PLCs

ลักษณะของ PLC	ข้อดีของ PLC
- อุปกรณ์ไมโครอิเล็กทรอนิกส์ - ควบคุมด้วยโปรแกรมภายในหน่วยความจำขนาดเล็ก - Micro Processor - จับเวลา, นับเวลาด้วยซอฟต์แวร์ - ระบบประกอบด้วยโมดูล - I/O หลายชนิด - I/O แบบ Remote - ระบบตรวจสอบตัวเอง - สภาวะของตัวแปรภายใน ระบบถูกเก็บไว้ในหน่วยความความจำ	- ความน่าเชื่อถือสูง - แก้ไขง่าย คล่องตัว - เปลืองเนื้อที่น้อย - ติดต่อกับระบบอื่น ๆ ได้ง่าย - ประสิทธิภาพ การทำงานสูง - การผลิตที่ได้มาตรฐานสูง - สามารถทำงานที่ซับซ้อน - หลีกเลี่ยงฮาร์ดแวร์ที่ยุ่งยาก - เปลี่ยนแปลงแก้ไขง่าย - ติดตั้งง่าย - คล่องตัวในการใช้งาน - บำรุงรักษาง่าย - ขอบเขตการควบคุมกว้าง - ป้องกันการผูกขาดจากผู้ผลิต - ลดการเดินสาย - ลดการบำรุงรักษา - ซ่อมแซมง่าย - ตรวจสอบรวมข้อมูล และจัดทำรายงานได้ง่าย

PLCs ส่วนใหญ่จะใช้ภาษา ladder Diagram ซึ่งภาษานี้ใช้สัญญลักษณ์ของการควบคุม ซึ่งทำให้ง่ายต่อการทำงานของผู้ใช้ไม่ว่าจะเป็นช่างเทคนิคและวิศวกร โดยปกติถ้าเป็นชนิดที่มีจอ CRT Ladder Diagram จะถูกแสดงผลบนจอมอนิเตอร์ และผู้ใช้ทำการโปรแกรมดัดแปลงแก้ไขได้โดยง่าย และในการทำงานแต่ละขั้นตอน สถานะของอุปกรณ์จะถูกแสดงขั้นที่ Ladder ในแบบ Real time เพื่อสะดวกแก่ผู้ใช้ในการตรวจซ่อมโปรแกรม และหาข้อผิดพลาด (Fault) ของระบบ โดยการทำงานของ "Power Flow'

รูปแบบของคำสั่งต่าง ๆ ในการกำหนดตัวแปร เพื่อใช้กับสัญญาณต่าง ๆ ไม่ว่าจะเป็นดิจิตอล หรืออนาล็อค และการทำงานของ Data Register ก็ใช้คำสั่งในรูปแบบที่ง่ายต่อความเข้าใจ ตัวอย่างเช่น

Normally Open Contact (ปกติเปิด)

Normally Close Contact (ปกติปิด)

(out) Register Output

Timer ตั้งเวลา เป็นต้น

Ladder language =

Grefect language ใช้ Flow chart ของระบบเข้ามาช่วย เพื่อง่ายต่อความเข้าใจในระบบ และค้นหาสิ่งผิดปกติภายในโปรแกรม

Literal language เป็นคำสั่งในรูปแบบของภาษา Basic ซึ่งใช้อยู่ในคอมพิวเตอร์ เหมาะสำหรับการ Link ระบบ PLC_S เข้ากับระบบต่าง ๆ รวมทั้งการทำ FA (Factory Automation) ด้วย

Logic Language เป็นคำสั่งเฉพาะแต่ละบริษัท โดยมีความคล้ายคลึงกันส่วนใหญ่ ผู้ใช้ต้องมีความรู้ ความสามารถแปลงจาก Flowchart หรือ Ladder มาเป็น logic ได้

สำหรับ PLCs ที่ใช้งานในระบบใหญ่ อาจมีคำสั่งทางด้านการติดต่อ (interface)

กับคอมพิวเตอร์ต่าง ๆ ซึ่งใน PLCs ที่สลับซับซ้อน สามารถนำรูปแบบของ Program ต่าง ๆ ผสมเข้าด้วยกัน ภายในโปรแกรมหนึ่ง ๆ ได้ เพื่อความเหมาะสมในการใช้งาน เช่น Operator interface

3.1 การจัดการโดยอินเตอร์เฟรส (Operator Interface)

ในการควบคุมงานของ PLCs จะทำที่แผงควบคุม ซึ่งจะมีปุ่มควบคุมต่าง ๆ เช่น Auto/Manual ปุ่ม Reset Alarm และ Push botton ต่าง ๆ ในกรณีที่ระบบมีการแสดงผลโดยใช้จอมอนิเตอร์แสดงสถานะการทำงานของระบบ จะแสดงรูปกราฟของระบบในรูปแบบการควบคุมนั้น ๆ หรือ Ladder diagram พร้อมสถานะการทำงานในแต่ละช่วงของ Scanning time

สำหรับระบบเล็กลงมาที่ไม่มีจอมอนิเตอร์ ก็อาจจะใช้หลอดไฟแสดงสถานะของ I/O ที่สำคัญ ๆ ตลอดจน การแสดงการเกิดข้อผิดพลาด (Fault) ต่าง ๆ ได้เช่นเดียวกัน

ในปัจจุบัน รูปแบบการทำงานโดยใช้ PLCs ความสามารถได้ขยายออกไปอย่างกว้างขวาง เช่น

- สามารถแสดงผล ณ จุดต่าง ๆ เมื่อเกิดข้อผิดพลาดในรูปแบบของตัวหนังสือ
- สามารถสร้างรูปของขบวนการผลิตขึ้นแสดงผลบนจอภาพ
- สามารถเลื่อนภาพได้เอง ขณะที่เกิด Fault และ สามารถพิมพ์ข้อมูลต่าง ๆ ออกมา printer ได้
- สามารถเก็บข้อมูลต่าง ๆ เข้ามาประมวลผลได้

3.2 พื้นฐานของการโปรแกรม PLC ทางด้าน Logic (BASICS OF PLC PROGRAMMING)

3.2.1 วงจรตรรก (logic)

หมายถึง วงจรไฟฟ้าที่ประกอบด้วย อุปกรณ์อิเล็กทรอนิกส์ หรือรีเลย์ที่มีสัญญาณเพียง 2 ระดับ คือ 2 สภาวะ PLC จะใช้สัญญาณไฟฟ้า 2 ระดับ แทน 2 เหตุการณ์ที่แตกต่างกัน เช่น การปิดและเปิดวงจรไฟฟ้า ของสวิตช์ หรือหน้าสัมผัสของรีเลย์ การสว่างและดับของหลอดไฟ การเปิดและปิดวาล์ว

วงจรตรรกมี 2 ชนิด คือ positive logic และ negative logic

positive logic

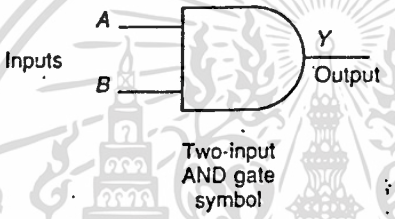
- ไฟสูง แทน ลอจิก "1"
- ไฟต่ำ แทน ลอจิก "0"

negative logic

- ไฟสูง แทน ลอจิก "0"
- ไฟต่ำ แทน ลอจิก "1"

– การปฏิบัติการ AND

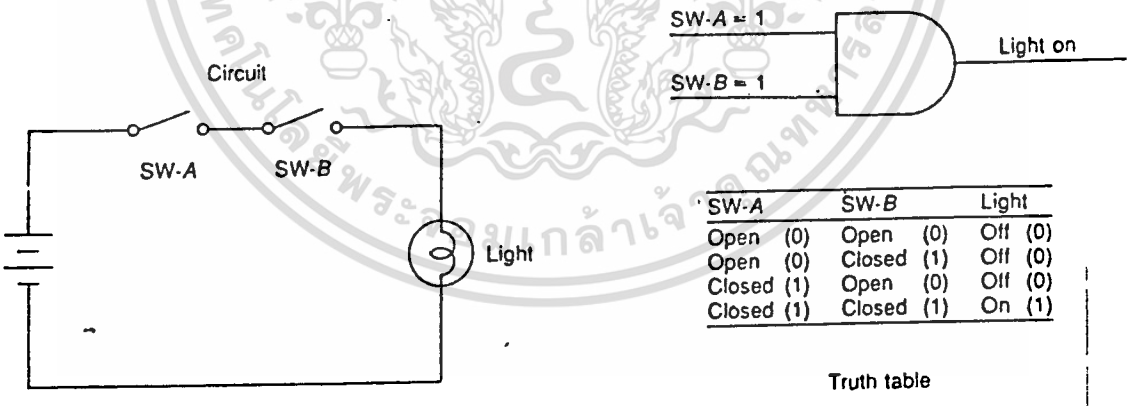
การปฏิบัติลอจิก AND เอาต์พุต Y ถ้าทุกอินพุตมีสภาวะลอจิก "1" หมด



Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

AND truth table

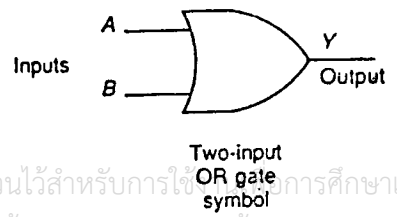
แทนได้ดังวงจร



SW-A		SW-B		Light	
Open	(0)	Open	(0)	Off	(0)
Open	(0)	Closed	(1)	Off	(0)
Closed	(1)	Open	(0)	Off	(0)
Closed	(1)	Closed	(1)	On	(1)

Truth table

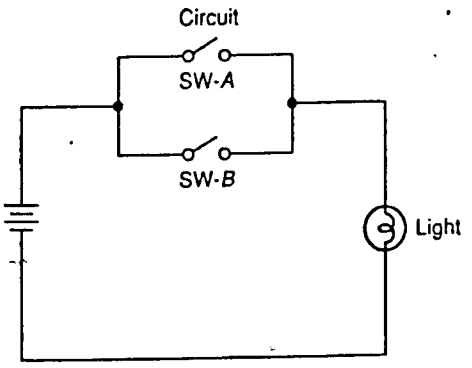
– การปฏิบัติการ OR

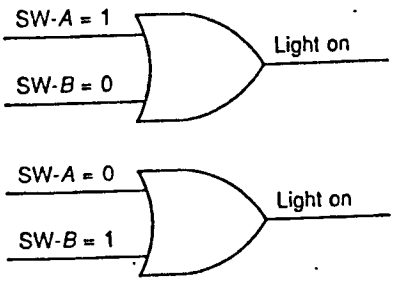


Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

OR truth table

แทนได้ดังวงจร





SW-A		SW-B		Light	
Open	(0)	Open	(0)	Off	(0)
Open	(0)	Closed	(1)	On	(1)
Closed	(1)	Open	(0)	On	(1)
Closed	(1)	Closed	(1)	On	(1)

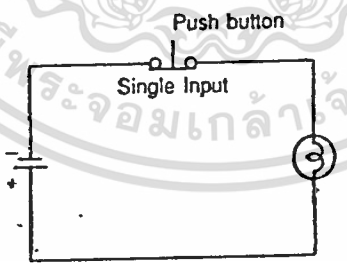
Truth table

– การปฏิบัติการ NOT

เอาพุต จะมีสภาวะตรงข้ามกับสภาวะอินพุต

A	NOT A
0	1
1	0

NOT truth table

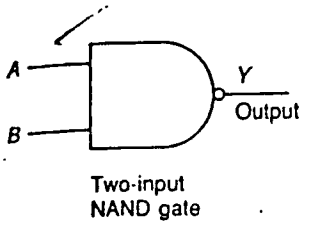


Push button		Light	
Not pressed	(0)	On	(1)
Pressed	(1)	Off	(0)

Truth table

Light will be on if the push button is not pressed.

– ปฏิบัติการ NAND

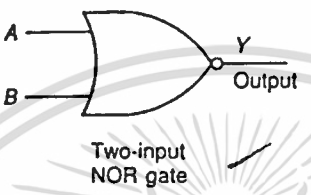


Two-input
NAND gate

Inputs		Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NAND truth table

– การปฏิบัติการ NOR



Two-input
NOR gate

Inputs		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

NOR truth table

การปฏิบัติการ XOR (exclusive–OR)

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Truth table

<u>วงจรลอจิก</u> หมายถึง การเดินสายเชื่อมต่ออุปกรณ์ต่าง ๆ เช่น สวิตช์ รีเลย์ ตัวตั้งเวลา และตัวนับให้ทำงานร่วมกันในระบบควบคุมแบบเก่า ข้อเสียทำให้การเปลี่ยนแปลงแก้ไขและขยายระบบไม่สะดวก ระบบควบคุม PLC ใช้โปรแกรมลอจิก หรือการโปรแกรมกำหนดเงื่อนไข การควบคุมแทนการเชื่อมต่ออุปกรณ์ต่าง ๆ ด้วยวิธีการเดินสาย ทำให้เกิดความสะดวกและง่ายขึ้น

PLC แทนวงจรรีเลย์ด้วยขั้นตอน การปฏิบัติลอจิก AND OR และ NOT ซึ่งถูกกำหนดขึ้นตามเงื่อนไข การควบคุมที่ต้องการ โดยใช้คำสั่งหรือภาษาของ PLC ภาษาพื้นฐานที่ควบคุมแบบ

ON และ OFF คือ ภาษา ladder และ ภาษาบูลลีน (Boolean language)

ภาษาแลดเดอร์ใช้สัญลักษณ์แทนหน้าสัมผัสของ Relay ในการเขียนโปรแกรม การเปลี่ยนแปลงวงจร Relay ก็คือการเปลี่ยนแปลงโปรแกรมใน PLC

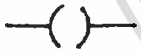
ชุดคำสั่งของ PLC ที่ประกอบขึ้นเพื่อควบคุมเอาต์พุต 1 จุดเรียกว่า รังก์ (Rung) ในบางครั้ง รังก์อาจมีเอาต์พุตมากกว่า 1 จุด แต่เอาต์พุตต้องมีจุดต่อร่วมกันเสมอ

โปรแกรมของ PLC จะประกอบด้วยกลุ่มรังก์หลายรังก์ ทำหน้าที่ร่วมกัน

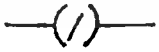
<u>สัญลักษณ์ หน้าสัมผัส</u>

Normally-Opened Contact = หมายถึง สภาพตอนแรกสวิตช์จะอยู่ในลักษณะเปิด เราจะให้สภาวะเปิดแทน logic "0" และหน้าสัมผัสปิดแทน logic "1"

Normally-close Contact หมายถึง สภาพตอนแรก สวิตช์จะอยู่ในลักษณะปิด เราจะให้สภาวะปิดแทน logic "0" และหน้าสัมผัส<u>เปิด</u> แทน logic "1"



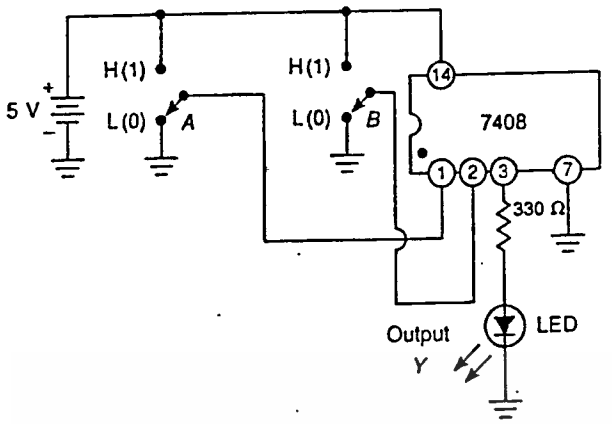
เอาต์พุตแบบปกติเปิด (energized output) หมายถึงอุปกรณ์เอาต์พุต เช่นหลอดไฟฟ้า และขดลวดรีเลย์ <u>จะทำงาน เมื่อมีสภาวะลอจิก</u> "1"

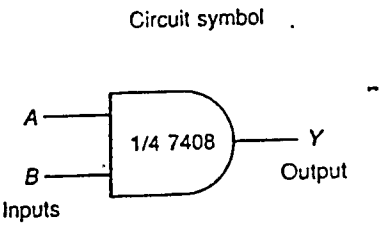


เอาต์พุตแบบปกติปิด (de-energized output) หมายถึง อุปกรณ์เอาต์พุต เช่นหลอดไฟฟ้า และขดลวดรีเลย์ ซึ่งจะทำงานเมื่อมีสภาวะลอจิก "0"

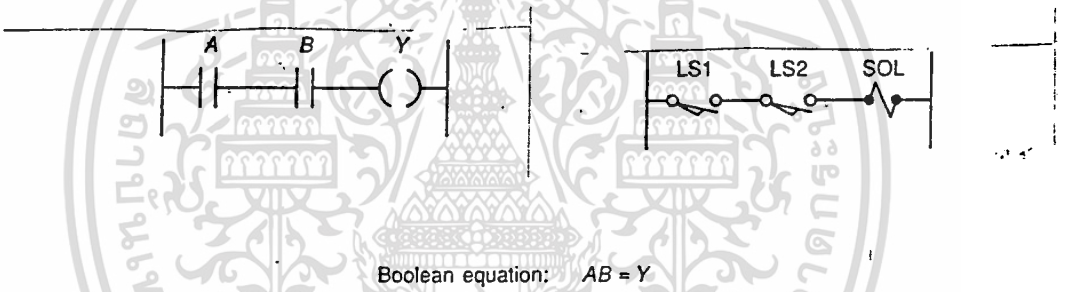
<u>การแทนวงจร</u> Relay ให้เป็น <u>logic ladder diagram</u>

ตัวอย่างที่ 1 ถ้าเราต้องการใช้ limit switch เพื่อจะควบคุม solenoid valve



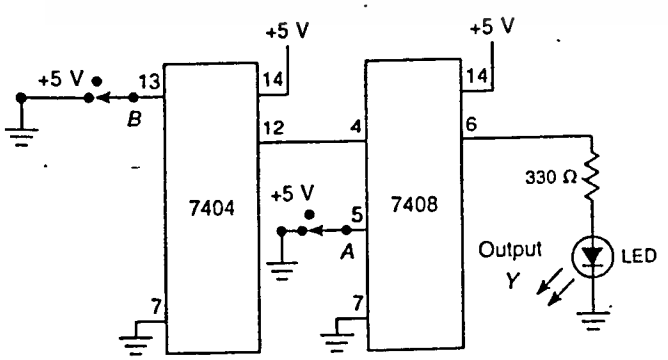


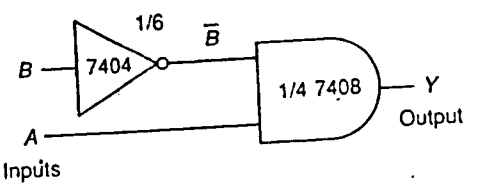
Logic gate Schematic Circuit Symbol



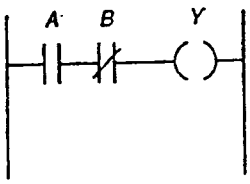
Logic Ladder diagram program Relay ladder diagram

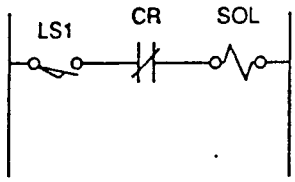
ตัวอย่างที่ 2 มี 1 Limit Switch ต่ออนุกรมกับ Normally close contact (NC) เพื่อที่จะควบคุม Solenoid Valve





Logic gate Schematic Circuit Symbol





Logic Ladder diagram program Relay ladder diagram

3.2.2 การใช้ PLCs ทดแทนวงจรรีเลย์ เมื่อมีความต้องการดังต่อไปนี้

1. ระบบควบคุมที่แก้ไขดัดแปลงได้ง่าย
2. ความน่าเชื่อถือสูง
3. เนื้อที่จำกัด ในขอบเขตที่กำหนด
4. การขยายจำนวน I/P, O/P ในอนาคต
5. เก็บรวบรวมข้อมูลการผลิต และของกระบวนการ
6. แก้ไขลักษณะการควบคุมบ่อยครั้ง และใช้เวลาน้อยที่สุด
7. การควบคุมลักษณะคล้ายกัน ถูกใช้กับเครื่องจักรหลายเครื่องพร้อมกัน
8. ระบบควบคุมมีการขยายในอนาคต
9. จัดหา และฝึกอบรมบุคลากรที่ทำหน้าที่ดูแลรักษา PLCs ได้
10. มีระบบการตรวจสอบตัวเอง
11. บำรุงรักษาง่าย

3.2.3 ส่วน Timer และ Counter

การใช้ Timer เพื่อจุดประสงค์เพื่อจะ delay การปิดและเปิด ของหน้าสัมผัส (Contact) หรือจุดต่อของ output เพื่อใช้ในการควบคุมกระบวนการใดกระบวนการหนึ่ง

Machanical Timing Relay

แบ่งออกเป็น 2 แบบ

- On delay จะทำให้เกิดการ delay เมื่อถูก energize)
- Off delay จะถูกทำให้เกิดการ delay time

เมื่อ Relay ถูก de-energized

<u>สัญญลักษณ์</u> ของ On-delay Symbols

สภาวะเริ่มแรก Relay ยังไม่ได้รับพลังงาน (de energized) เมื่อ Relay ได้รับพลังงาน (energized) จะ delay time ไปสักพัก หลังจากนั้นหน้าสัมผัสค่อยปิดลง

กรณีเป็นแบบ Normally open, timed closed contact (NOTC)

กรณีที่เป็นแบบ Normally-closed, timed open contact (NCTO)

สภาวะเริ่มแรกหน้าสัมผัสจะปิดอยู่ (close) ขณะที่ยังไม่ได้รับพลังงาน (de-energized) เมื่อ relay ได้รับพลังงาน (energizeed) จะเกิดการหน่วงเวลา (delay-time) ไปสักครู่ หลังจากนั้นหน้าสัมผัสค่อยเปิด (open)

<u>สัญญลักษณ์</u> ของ Off-delay Symblos

กรณีเป็นแบบ Normally Open, timed Open contacts (NOTO)

- โดยที่ Contact ปกติจะเปิด(open) ขณะที่ relay ยังไม่ได้รับพลังงาน (denergized)

- เมื่อเราให้พลังงานให้แก่ Relay coil หน้าสัมผัสจะปิดทันที (Closes instantly)
- หลังจากนั้นก็เอาพลังงานออกจาก Relay coil จะเกิดการหน่วงเวลาสักครู่ ก่อนที่หน้าสัมผัสจะเปิด (open) เหมือนสภาวะปกติหรือสภาวะเริ่มแรก

กรณีเป็นแบบ Normally closed, timed closed contact (NCTC)

- โดยปกติ contact หรือหน้าสัมผัสจะปิดอยู่เสมอ (closed) ในขณะที่ยังไม่ ให้พลังงานแก่ relay coid (de-energized)

- เมื่อให้พลังงานแก่ relay coil หน้าสัมผัสจะเปิด (open) ออกทันที
- เมื่อเอาพลังงานออกจาก Relay coil (deenergized) จะเกิดการหน่วงเวลาสักครู่ ก่อนที่หน้าสัมผัสจะปิด (closed) เข้าสู่สภาวะปกติอีกครั้ง

(Timer Instructions) คำสั่ง Timer

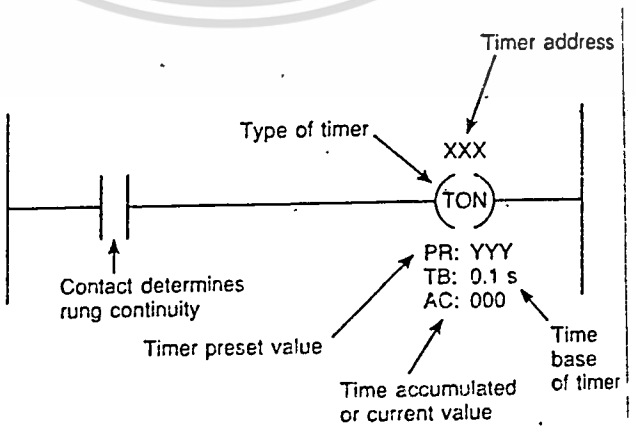
PLC Timer เป็นคำสั่ง Output ทำงานเสมือนกับ mechanical Relays ซึ่งมันจะ activate และ deactivate อุปกรณ์ภายในช่วงเวลาหนึ่ง

คำสั่ง timer และ counter ในตอนเริ่มแรกยังไม่มีใน PLC แต่ได้มีใช้ในชุดคำสั่งของ PLC ในเวลาต่อมา มี 2 วิธีที่จะแสดง timer ในรูปแบบ logic ladder program ของ PLC ดังรูปที่ 3.1

<u>ตัวอย่างที่ 1</u>

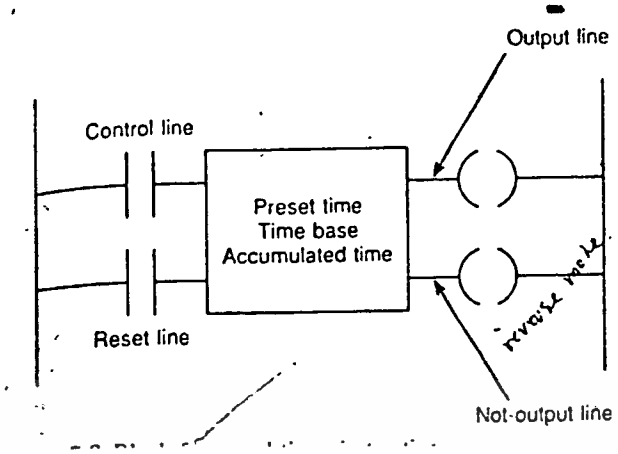
ส่วนของ time's preset value หรือ time delay period เมื่อ timer rung ได้ต่อเนื่องกันทางลอจิกแล้ว (logic contiuity) timer ก็จะนับทีละช่วง time-based นับจนกระทั่งค่าของ current value มีค่าเท่ากับ Time preset value เมื่อค่า current value = ค่าที่อยู่ใน preset Value Timer จะ energized

ส่วน Output ทำให้ output contact ของ Timer ทำงาน (ถ้าปกติเปิด ถ้าได้รับ energize output contact ของ Timer ก็จะปิด) การกำหนด Contact ให้อยู่ในรูปแบบใด ขึ้นอยู่กับการโปรแกรม



รูปที่ 3.1 coil-formatted timer instruction

ตัวอย่างที่ 2 รูปแบบจะแสดงเป็น block format ดังรูปที่ 3.2



รูปที่ 3.2 Block-formatted timer instruction

timer block จะมี 2 input (contition) ที่ต่อเข้าไปยัง block input แรกชื่อ Control input ที่สอง reset ซึ่งส่วน Control line จะควบคุมการดำเนินการเกี่ยวกับเวลา ของ Timer (timing Operation) เมื่อใดก็ตามที่ input นี้ TRUE ก็เหมือนกับการให้ power supplied ให้แก่ input นี้ จะทำให้ Timer เริ่มจับเวลา ถ้านำ power ออกจาก control time timer ก็จะหยุดทำงาน

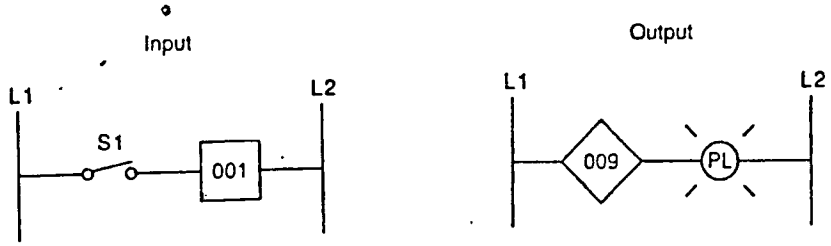
reset line จะ Reset ส่วน Accumulated value ของ timer ให้เป็นศูนย์ ในบางงานต้องการทั้งส่วนของ control line และ reset line ที่ TRUE เพื่อควบคุมเวลาต่อเวลาบางงานต้องการ Reset ด้วย บางงานก็ไม่ต้องการ และจะกระทำคำสั่ง Reset เมื่อ Reset line มีค่าเป็น TRUE

ใน Timer Instruction block จะมีตัวที่กระทำการจับเวลาต่าง ๆ ดังนี้ preset time, time base of the timer, the current or accumulate time

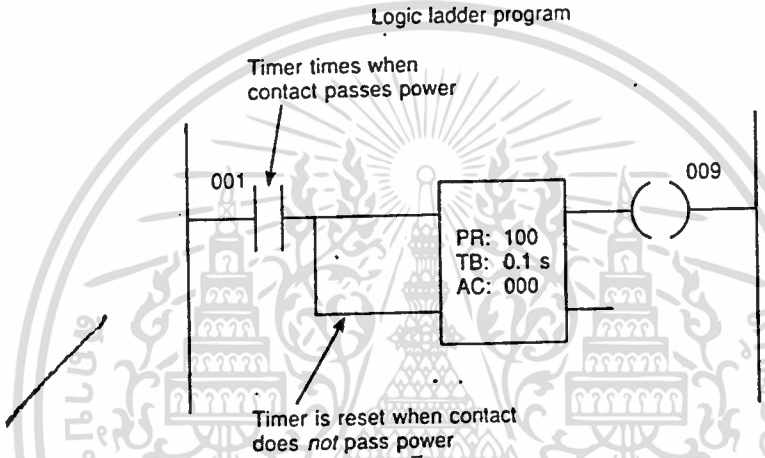
ทุก ๆ block-formatted timers ต้องมีอย่างน้อย 1 เอาต์พุต ที่ออกจาก timer กรณีที่มี 2 เอาต์พุต เอ๊าต์พุตที่สองจะทำงานใน reverse mode เมื่อใดก็ตามที่เวลายังไม่ถึงภาวะที่กำหนด เอาต์พุตแรกจะ off และเอาต์พุตที่สองจะ ON เมื่อใดก็ตาม timer ทำงานสิ้นสุดที่เรากำหนดเอาไว้ เอาต์พุตแรกก็จะ ON และเอาต์พุตที่สองก็จะ Off

ON-DELAY TIMER INSTRUCTION (non-retentive timer)

คำสั่ง ON-DELAY timer เป็นคำสั่งจำลองการทำงานของ pneumatic electromechanical On-delay time ดังแสดงในรูป



Logic Ladder program



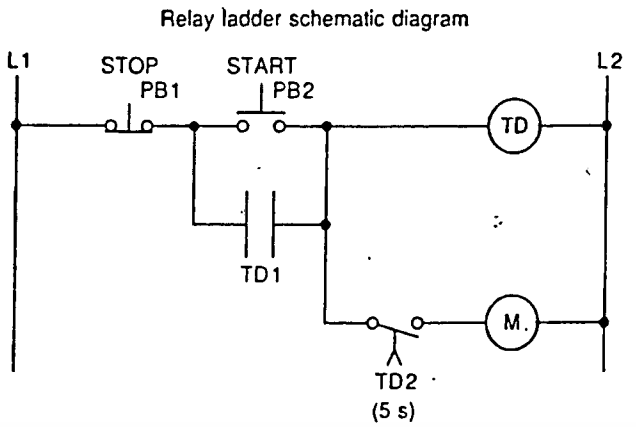
โดยการโปรแกรม ON-DELAY timer โดยใช้คำสั่ง block-fermatted Timer จะทำงานเมื่อปิดสวิตท์ input หมายเลข 001 โดยเราตั้ง timer ให้นับได้ 10 second ซึ่งเมื่อครบตามเวลาที่โปรแกรมใช้แล้วก็จะส่ง power timer ไปออก output หมายเลข 009

ซึ่งเมื่อ input 001 สวิตช์ปิด จะเริ่มนับทันทีนับตั้งแต่ AC=000 จนกระทั่ง AC=100 (ครบ 100 step โดยจะนับทุก ๆ 0.1 วินาที 100 ครั้ง ได้ 0.1x100 = 10 วินาที) หลังจากนั้นจะส่ง power ไปที่ output 009 ให้สวิตท์ปิด

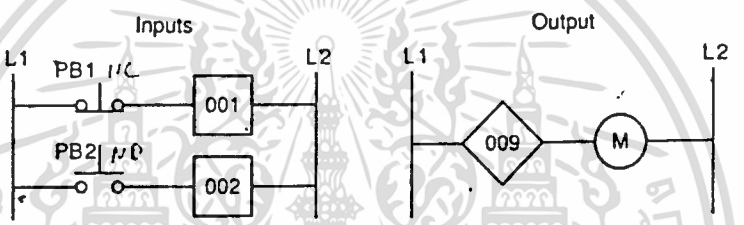
ถ้า switch input เปิดก่อนที่ timer จะนับเสร็จ accumulated-timer จะ Reset โดยอัตโนมัติ ลักษณะการทำงานอย่างนี้เรียกอีกอย่างว่า "NON-retentive timer" ก็ได้

อธิบายได้อีกอย่างคือ เมื่อเกิดการสูญเสีย power flew จะให้ timer Reset ตัวมันเองทันทีโดยอัตโนมัติ

เมื่อ SW ของหลอดไฟ (output 009) หลังจากที่ timer นับไปเอง SW ของหลอดไฟจากที่เคยเปิด (OFF) ก็จะปิด (ON) ทันที รูปที่ 3.3 แสดงการนำไปใช้งาน

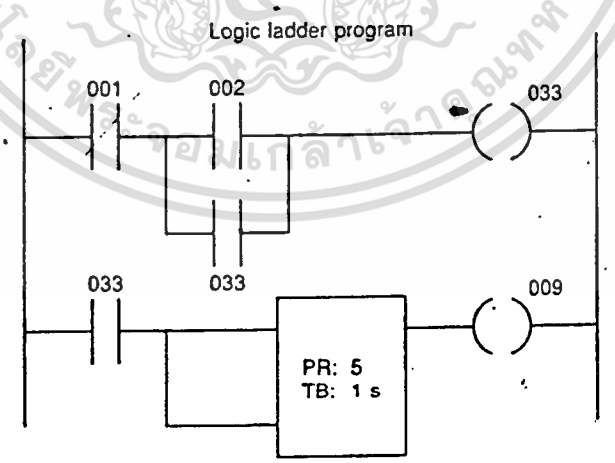


Relay ladder schematic diagram



Logic ladder diagram

จะได้ Logic Ladder diagram ดังนี้

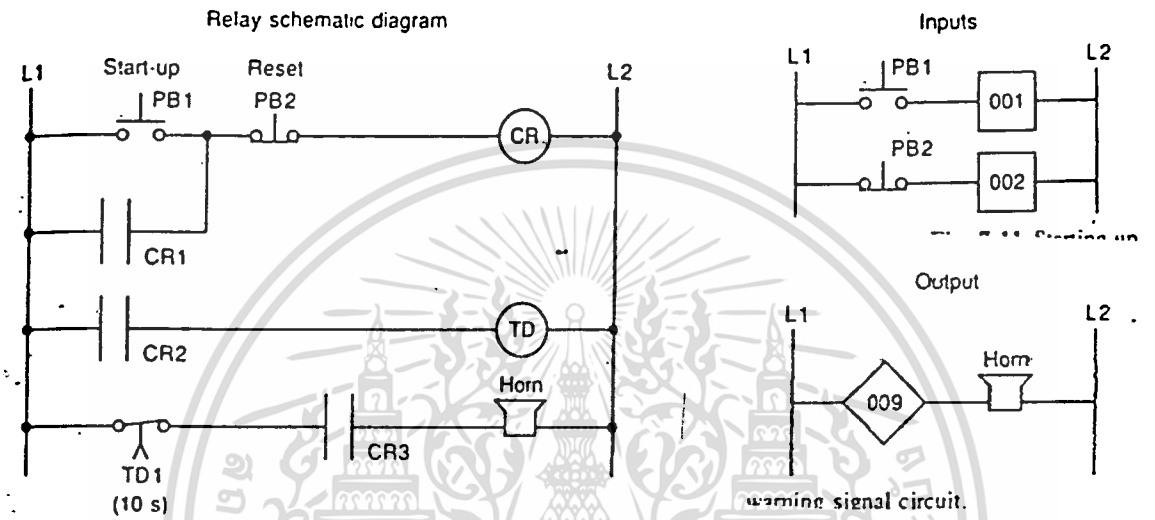


ตามรูป Coil M จะได้รับพลังงานหลังจากที่กดปุ่ม START ไปแล้ว 5 วินาที (5s) Contact ID1 คือ instantaneos contact จะปิดคอนแทค เมื่อ Coil TD รับพลังงานทันที ส่วน TD2 คือ timed contact จะปิดคอนแทคเมื่อ Coil TD ได้รับพลังงานไปแล้วช่วง

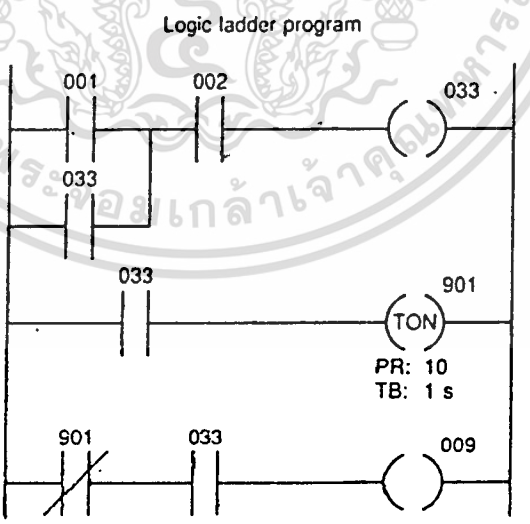
เวลาหนึ่ง ในที่นี้คือ 5 วินาที

ซึ่งใน Ladder program ที่แสดงให้เห็นนั้นเป็นคำสั่งควบคุมคอนแทค (contact) โดยใช้ Relay ภายใน (internal Relay) ที่ใช้จัดการเกี่ยวกับ timer การนี้ โดย TD1 (instantaneous Contact) จะอ้างอิงถึง internal Relay coil ในขณะที่ TD2 (Time-delay contact) จะอ้างถึง timer output coil

รูปที่ 3.4 starting-up warning signal circuit



จะได้ Logic Ladder diagram

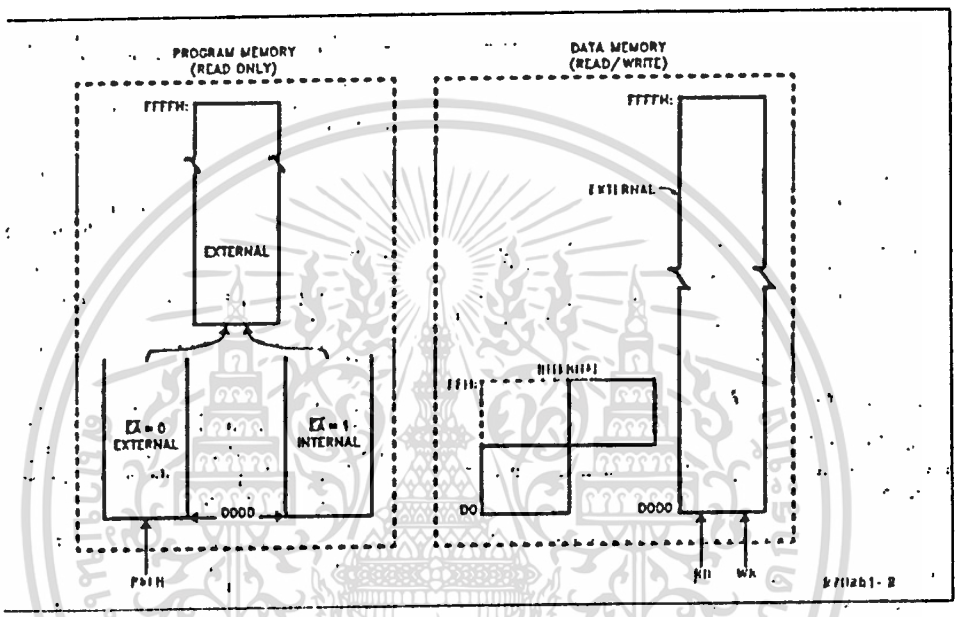


ดังรูปที่ 3.4 แสดงการใช้งาน On-delay timer ที่ใช้ NCTO contactในวงจรนี้ เป็นวงจรส่งสัญญาณเตือนกรณีมีการโยกย้ายสิ่งของเกิดขึ้น

4.1 โครงสร้างของ 8031

ในส่วนนี้จะกล่าวถึง รายละเอียดบางประการของไมโครโปรเซสเซอร์ 8031 เพื่อเป็นแนวทางในการออกแบบ และพัฒนา ดังจะเสนอต่อไปนี้

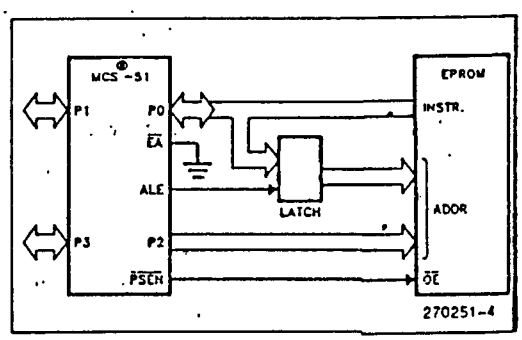
4.1.1. ส่วนฮาร์ดแวร์ของ 8031



Reset address 0000H

External Interrupt0	0003H	Time0	000BH
External interrupt1	0013H	Time1	B001BH

รูปที่ 1



รูปที่ 2

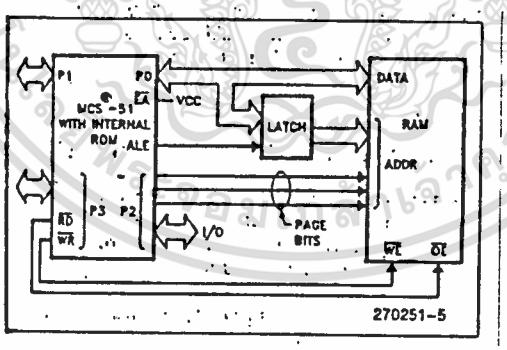
จากรูปที่ 2 แสดงถึงการต่อทางด้าน hardware โดยใช้ external program execution

จะเห็นได้ว่า 16I/O line (Porto และ Port2) ใช้สำหรับทำการเกี่ยวกับบัส (Bus) ระหว่างการ fetch คำสั่ง จาก external program memory โดยที่ Port0 จะ multiplex เอาสัญญาณ address และ data bus ออกมา ถ้ากระทำตัวแบบ address bus ก็จะปล่อยส่วนของ program Counter (PCL) ที่ไบต์ต่ำออกมา แล้วก็จะเกิดสถานะลอย (float state) เพื่อรอโค๊ดคำสั่ง [Code byte] จากส่วนของ program memory

และในระหว่างที่ไบต์ต่ำของส่วนโปรแกรมเคาน์เตอร์ (PC) คำสั่งมีสถานะชัดเจน (Valid) จะมีสัญญาณ ALE (Address latch enable) ไป demultiplex เอา address latch เอาไว้ ในขณะที่พอร์ต 2 (P2) จะปล่อย address ไบต์สูงของโปรแกรมเคาน์เตอร์ออกมา (PCH) และเมื่อมีสัญญาณจาก PSEN (program store enable) ไปกระตุ้น EPROM ให้ปล่อยโค๊ดคำสั่งออกมาให้ไมโครคอนโทรลเลอร์

แอดเดรส ของ program memory จะเป็นแบบ 16 บิตเสมอถึงแม้จะถูกตัวต่ำกว่า 16 Kbyte ก็ตาม

โดยที่การกระทำคำสั่งกับ external program พอร์ต 8 บิต 2 พอร์ต จะถูกนำมาใช้เสมอในการอ้างแอดเดรส คือ พอร์ต 0 และ พอร์ต 1

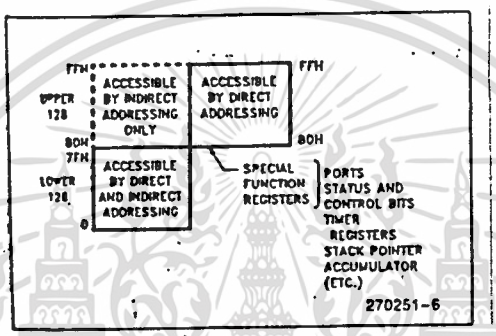


รูปที่ 3

รูปที่ 3 ภายใน 8051 นั้น มีหน่วยความจำถาวร ROM อยู่ภายในแล้ว ฉนั้นจะ fetchs คำสั่ง ภายในตัวมันเองเลย โดยจะเริ่มจาก 0000H ภายใน Rom ที่อยู่ในตัวมันเอง สัญญาณที่จะบอกว่า fetch ภายในหรือภายนอกก็คือ EA (External Access) ถ้า EA=0 ให้ fetch คำสั่งทั้งหมด external program memory ถ้า EA=1 ให้ fetch คำสั่งเริ่มแรกที่ภายในตัวมันเอง ส่วนภายนอกจะต่อกับหน่วยความจำชั่วคราว (RAM) ขนาด 2 Kbyte โดยใช้

แอดเดรสไบต์สูงเลือก Page เลือกได้ 8 page page ละ 256 bype ได้ขนาด 256 x 8 = 2048 bytes =2 Kbyte นั่นเอง ส่วนบิตที่เหลือของ P2 ก็สามารถใช้เป็นอินพุต และเอาท์พุตได้ยิกด้วย สัญญาณที่ใช้ควบคุมในการอ่านหน่วยความจำชั่วคราว คือสัญญาณ RD และ WR อันเป็นบิตหนึ่งของพอร์ต P3 ส่วนพอร์ตที่เหลือก็สามารถนำไปใช้เป็น I/O ได้อีก

เราสามารถขยายหน่วยความจำ ได้สูงถึง 64 Kbyte การอ้างแอดเดรสของ external data memory สามารถแยกเป็นแบบ 1 byte และ 2 byte ได้ กรณี 1 ไบต์ ใช้กรณีแบ่งเป็นเพจ (page)ให้ RAM memory กรณี 2 ไบต์ คือ ต้องการปล่อย address byte สูงออกไปยัง port P2



รูปที่ 4 internal data memory

จากรูปที่ 4 ภายในของ Internal data memory จะถูกแบ่งออกเป็น 3 บล๊อก (3 block)

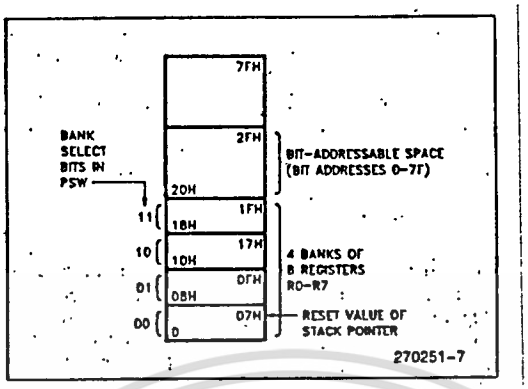
1. Lower 128 (บล๊อกต่ำ)
2. Upper 128 (บล๊อกสูง)
3. SFR Space

จำนวนไบต์ที่อ้าง Internal data memory หมายถึง แอดเดรส ใช้เพียง 1 ไบต์ก็พอเพียง ซึ่งอ้างได้ 256 ไบต์เท่านั้น แต่อย่างไรก็ตามในความเป็นจริงสามารถอ้างได้ถึง 256 + 128 = 384 ไบต์ โดยใช้การทริค (Simple trick)

จากรูปที่ 4 จะเห็นว่า 80-FFH จะมี 2 blocks ส่วนแรกอ้างแอดเดรสไม่โดยตรงกับอีกส่วนอ้างแบบโดยตรง ถึงแม้จะบอกว่า Upper128 และ SFR จะอยู่ที่แอดเดรสเดียวกันก็ตาม แต่จริง ๆ ทางกายภาพแยกกันโดยเด็ดขาด (physically separate entitics)

ในรูปที่ 5 32 ไบต์ต่ำสุดจะถูกแบ่งเป็นกลุ่มได้ 4 กลุ่ม (4 banks) แต่ละกลุ่มจะ

มี 8 ไบต์ แต่ละไบต์เราจะมองเหมือน Register 1 ตัว ฉนั้น แต่ละกลุ่มหรือแบงค์ จะมี Register 8 ตัว (R0-R7) โดยที่โปรแกรมจะนำรีจีสเตอร์เหล่านี้ไปใช้



รูปที่ 5 Lower 128 bytes of internal RAM

เราสามารถเลือกใช้รีจีสเตอร์แต่ละแบงค์ได้ โดยไป Set บิตของ PSW (program status word) บิตที่ใช้เลือกแบ๊งค์ คือ บิตที่ 3 และบิตที่ 4 โดยที่ PSW จะเป็น Register ที่อยู่ในพื้นที่ของ FFR space แอดเดรสที่ 0D0H

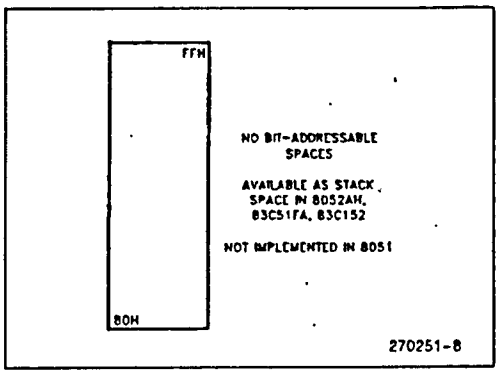
CY	AC	F0	RS1	RS0	OV		P

bit1 bit0

Select bank

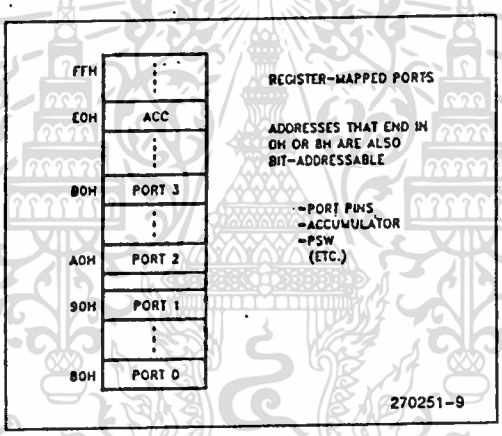
สำหรับ 16 ไบต์ เหนือขึ้นไปจากรีจีสเตอร์แบ๊งค์ จะเป็นรูปแบบของการอ้างแบบบิตได้ (bit-addressable memory space) ในเซ็ตคำสั่งของ MCS-51 จะรวมคำสั่งที่กระทำแบบบิตได้ เอาไว้ด้วย 16 ไบต์ จะมีทั้งหมด 16x8 = 128 บิต สามารถอ้างได้โดยตรงแบบบิต โดยที่ตำแหน่งแอดเดรสของบิตเริ่มที่ 00H จนถึง 7FH

ในรูปที่ 5 Lower 128 สามารถอ้างทั้งที่เป็นแบบ direct และ indirect ได้



รูปที่ 6 The Upper 128 byte of internal Ram

รูปที่ 6 upper 128 ไม่มีอยู่ภายในของ 8031 (MCS-51) สำหรับ upper 128 สามารถ Access แบบ direct เท่านั้น



รูปที่ 7 เป็นการมองส่วน SFR Space รวมทั้ง พอร์ตและชาร์, ไทม์เมอร์ ฯลฯ รีจิสเตอร์นี้ (SFR) ตัวแอดเดรสได้แบบ direct เท่านั้น

ตารางที่ 1 แสดงตระกูลของ MCS-51 Microcontrollers

Device Name	ROMless Version	EPROM Version	ROM Bytes	RAM Bytes	16-bit Timers	Clk Type
8051	8031	(8751)	4K	128	2	HMOS
8051AH	8031AH	8751H	4K	128	2	HMOS
8052AH	8032AH	8752BH	8K	256	3	HMOS
80C51BH	80C31BH	87C51	4K	128	2	CHMOS

การอธิบายส่วน hardware ในช่วงนี้จะอธิบายถึง

- Port drivers การทำงานของ Port, สำหรับ Port0 และ Port2 ในการกระทำแบบ bus operation
- ส่วนของ Timer/Counters
- Serial Interface
- Interrupt System
- Reset
- The Reduced Power Modes in the CHMOS devices
- The EPROM versions of the 8051 AH, 8052A#, และ 80C51BH

จากตารางที่ 1 บอกว่า เบอร์ของไมโครคอนโทรลเวอร์ นั้นที่แสดงในตารางอยู่ในตระกูล MCS-51 ทั้งหมด แต่ละเบอร์จะมีข้อปลีกย่อยต่างกันไบต์เบอร์ 8051_s โดยทั่วไปก็เช่น 8051, 8051AH, 80C51BH คำสั่งที่ใช้กับไมโครคอนโทรลเลอร์เหมือนกันแต่จะแตกต่างกันบ้างเล็กน้อยที่สถาปัตยกรรม ภายใน เช่น 8051 มี ROM ภายใน 4K RAM 12K ไบต์ มี Timer แบบ 16 bit ให้ใช้ได้ 2 ตัว แต่ถ้าเป็น 8031 จะไม่มี Rom ภายในตัวมัน ต้องต่อออกมาใช้ที่ภายนอกเท่านั้น

ACC คือ Accumulator register เป็น mnemonics สำหรับ Accumulator คำสั่งเฉพาะ เขียนง่าย ๆ ว่า "A"

B register

รีจิสเตอร์ B จะใช้สำหรับกระทำการคูณ และหาร และใช้ได้ในกรณีทั่วไปเหมือนรีจิสเตอร์ตัวอื่น ๆ

Program Status Word

เขียนย่อ ๆ เอาไว้ PSW PSW รีจิสเตอร์จะเก็บข้อมูลสถานะของโปรแกรม

STACK POINTER

สแต็ค พอยเตอร์ รีจิสเตอร์ (Stack Pointer Register) มีความกว้าง 8 บิต สแต็คพอยเตอร์จะเพิ่มค่าก่อนที่จะเก็บข้อมูลลงไป ในเมื่อใช้คำสั่ง PUSH แล้วถึงค่อย CALL

executions ซึ่ง Stack เราสามารถกำหนดให้อยู่บนส่วนใด ๆ ของ RAM ก็ได้ แต่ค่าเริ่มต้นของ Stack Printer จะอยู่ที่ 07H หลังจากที่เกิดการ Reset เป็นสาเหตุให้ Stack จะเริ่มที่แอดเดรส 08H

DATA POINTER

เขียนย่อ ๆ DTPR จะประกอบด้วย ไบต์สูง (DPH) และไบต์ต่ำ (DPL) ดาต้าพอยเตอร์นี้จะเก็บแอดเดรส 16 บิต เอาไว้ สามารถกำหนดเป็นได้ทั้งรีจิสเตอร์แบบ 16 บิต และรีจิสเตอร์ 8 บิต แยกออกจากกันก็ได้

Ports 0 ถึง 3

พอร์ต 0,1,2,3 จะอยู่ในส่วนของ SFR_S แบบ latch ของพอร์ต P0,P1,P2,P3 ตามลำดับ

Serial data Buffer (SBUF)

ซีเรียล ดาต้า บัฟเฟอร์ จะประกอบด้วยรีจิสเตอร์ 2 ตัวแยกจากกัน ส่วนหนึ่งจะทำหน้าที่เป็น transmit buffer Register และอีกส่วนหนึ่งทำหน้าที่เป็น receive buffer register

เมื่อเรา move ข้อมูลไปที่ SBUF ข้อมูลจะไปอยู่ที่ transmit-buffer ซึ่งจะเป็นส่วน Serial transmission (Moving a byte to SBUF is What initiatcs the transmission)

เมื่อ data ถูก mov มาจาก SBUF ข้อมูลจะมาเก็บไว้ที่ receive butter

Timer Registers

เป็นรีจิสเตอร์คู่ (register pairs) (TH0, TL0), (TH1,TL1) และ (TH2,TL2) เป็นรีเตอร์นับ 16 บิต สำหรับ Timer/Counter 0,1 และ 2 ตามลำดับ

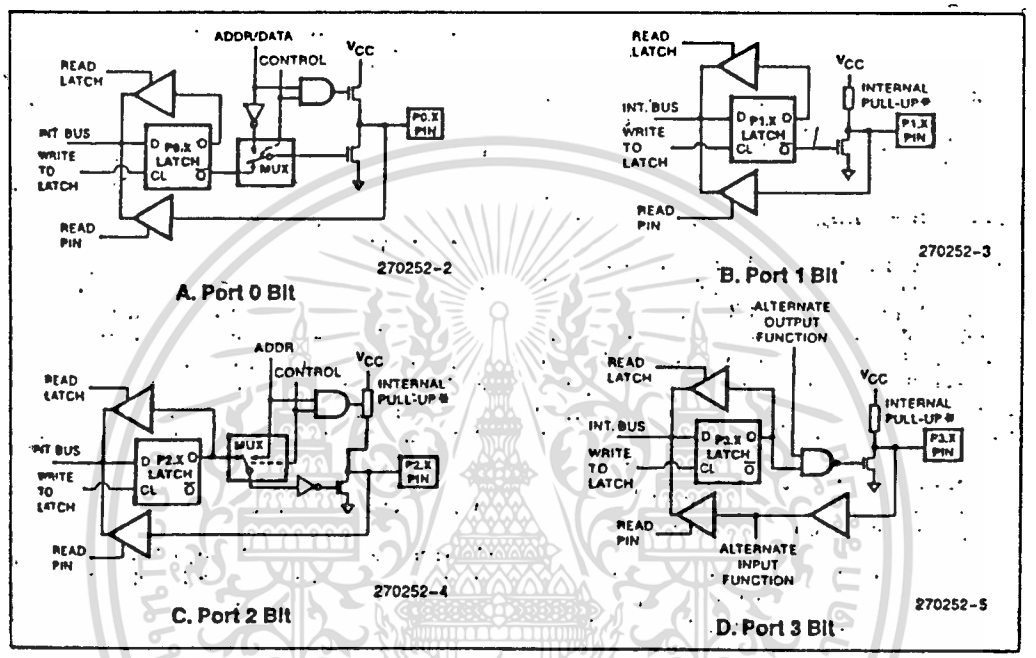
CAPTURE Registers

เป็นรีจิสเตอร์คู่ (RCAP2H, RCAP2L) เป็น Capture รีจิสเตอร์ สำหรับ Time2 ใน "Capture Mode" ใน Mode นี้ จะมีผลใน $8052_{,S}$ ที่ขา T2EX, โดยที่ TH2 และ TL2 จะถูกก๊อปปี้ไว้ใน RCAP2H และ RCAP2L ซึ่ง Timer2 จะมี 16 บิต auto-reload mode

โดยที่ RCAP2H และ RCAP2L จะเก็บค่า reload เอาไว้ จะอธิบายละเอียดหัวข้อถัดไป

CONTROL REGISTERS

เป็น Special Function Registers IP,IE,TMOD,TOON T2CON, SCON และ PCON ประกอบด้วยคอนโทลบิตและสถานะบิตเอาไว้ สำหรับ interrupt System, Timer/Counters, และ serial port



(8051 Port bit latches and I/O Buffers)

Port Structure) and Operation

ทั้ง 4 พอร์ต ใน 8051 เป็นแบบ bidirectional คือ ทิศทางของข้อมูลไหลได้ ทั้ง 2 ทิศทาง ส่งและรับได้ และสามารถเก็บเก็บข้อมูล (latch) ไว้ได้ (Special Function Registers P0-P3), มี Output driver, และมี input buffer

Output driver ของ พอร์ต P0 และ P2 และ input buffer ของพอร์ต P0 ใช้สำหรับอ้างถึงส่วน external memory ในการนำไปประยุกต์ใช้งาน

P0 Output เป็นแอดเดรสไบต์ต่ำ ของ external memory และใช้ Time-multiplexed กับข้อมูลที่จะอ่านหรือเขียนด้วย

P2 Output เป็นแอดเดรสไบต์สูงของส่วน external memory

เมื่อมีการอ้างแอดเดรสแบบ 16 บิต นอกเหนือจากติดต่อกับ external memory

แล้ว P2 จะทำหน้าที่เป็นพอร์ตธรรมดา นั่นคือจะเก็บข้อมูลส่งออกมา ของ Port2, (ที่เก็บอยู่ใน SFR content)

(Port 3) P3 ทุกขา เป็นแบบ multifuntional คือ นอกจากจะทำหน้าที่เป็นแบบพอร์ตธรรมดาแล้ว ยังมีหน้าที่พิเศษอย่างอื่น ๆ อีก (ซึ่งรวมทั้ง Port 1.0 และ Port 1.1 ด้วย) ดังนี้ (สำหรับ P1.0, P1.1 8052 เท่านั้น)

Port Pin	Alternate Function
P1.0	T2 (Timer/Counter 2 external input)
P1.1	T2EX (Timer/Counter2 CAPTURE/Reload-triggler)

(* P.1.0, P1.1 สำหรับ 8052 เท่านั้น)

Port Pin	Alternate Function
P3.0	RXD (serial input Port)
P3.1	TXD (serial output Port)
P3.2	INT0 (external interrupt)
P3.3	INT1 (external interrupt)
P3.4	T0 (Timer/Counter0 external input)
P3.5	T1 (Timer/Counter1 external input)
P3.6	WR (external data Memory write strobe)
P3.7	RD (external data Memory Read strobe)

I/O configurations

จากรูปที่ 1 แสดง functional diagram ของ bit latch และ I/O buffer ใน แต่ละพอร์ต, บิตแลทช์ (บิตหนึ่งในหนึ่งพอร์ตของ SFR) แสดงในรูปแบบของ D-FLIP FLOP

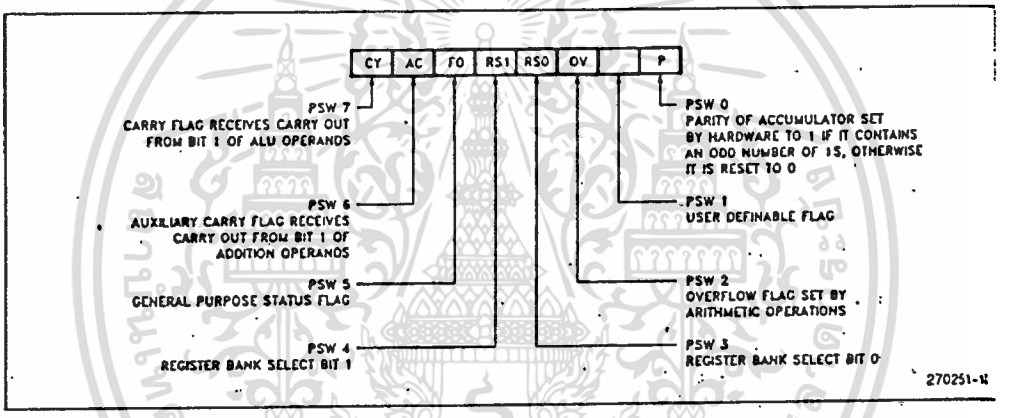
2. ส่วนการจัดการด้านโปรแกรมของ 8031(THE MCS-51 INSTRUCTION SET)

ไมโครโปรเซสเซอร์ตระกูล MCS-51 ทุกเบอร์จะใช้ชุดคำสั่งเหมือนกันหมด เนื่อง

จากชุดคำสั่งได้ถูกจัดให้มีความเหมาะสม (optimize) สำหรับ 8 บิต คอนโทรล (8-bit control application) มีการอ้างข้อมูลได้อย่างรวดเร็ว เนื่องด้วยมีหน่วยความจำชั่วคราวอยู่ภายใน (internal RAM) สำหรับ 8051 มี 256 byte สำหรับกรณี 8031 (Romless Version) มีเพียง 128 ไบต์ แต่ก็เพียงพอสำหรับข้อมูลขนาดเล็ก ๆ ได้ นอกจากนี้ยังมีคำสั่งสนับสนุนการอ้างพอร์ตภายนอกแบบ 1 บิตด้วย ซึ่งถูกออกแบบมาสำหรับงานควบคุม ซึ่งใช้หลักการทางลอจิก หรือเรียกอีกแบบว่า Bolean processing ก็ได้

Program Status Word

Program Status word (PSW) จะเก็บสถานะบิตที่ตอบสนองกับความเป็นไปในขณะนั้นของ CPU ดังรูปที่ 8



รูปที่ 8

ซึ่ง PSW จะอยู่ในส่วนของ SFR (Special Function Reg.) RSW ซึ่งประกอบไปด้วยCarry bit, Auxilary Carry (for BCD operation and two user-definable status flags

Carry bit นอกจากจะใช้กระทำเกี่ยวกับการกระทำทางคณิตศาสตร์แล้ว (Carithematic function) ยังทำหน้าที่เป็น เสมือน Accumulater" สำหรับการกระทำทางลอจิก อีกด้วย (Boolean operations)

RS0 และ RS1 เป็นตัวเลือก bank ของ register (R0-R7) เลือกได้ทั้งหมด 4 bank นั่น คือ มี R0-R7 อยู่ 4 ชุด แยกออกจากกันได้ 4 bank โดยการ Set

RSO และ RS1 ดูรูปที่ 5

Parity bit จะมีผลต่อลอจิก 1 (15) ใน Accumulator ถ้าใน Accumulator มีลอจิก 1 เป็นคี่ P=1, มีลอจิก 1 เป็นจำนวนครบคู่ที่ให้ P=0 อาจกล่าวได้ว่า จำนวนลอจิกใน Accumulator ที่มีลอจิก 1 เมื่อรวมกับ Parity bit แล้วจะต้องเป็นเลขคู่เสมอ และอีก 2 bit ของ PSW ถูกใช้สำหรับกรณีทั่วไป

Addressing Modes

ลักษณะการอ้างแอดเดรส โดยการใช้ชุดคำสั่งของ MCS-51 มีดังนี้

- แบบ Direct Addressing (แบบอ้างโดยตรง)

การอ้างแบบโดยตรง ถูกกำหนดแอดเดรสในช่วง 8 bit เท่านั้น คือ สามารถอ้างได้ภายใน internal RAM และในส่วน FSR_S เท่านั้น โดยการอ้างแบบโดยตรงเท่านั้น

- แบบ Indirect-Addressing

การออกแบบโดยอ้อม โดยเราจะใช้ Register กับแอดเดรสของข้อมูลอีกทีหนึ่ง ก่อนที่เราจะอ้างถึงซึ่งส่วนที่เราสามารถอ้างแบบอ้อมได้ ก็มีส่วน Internal RAM และExternal (memory) ทั้งหมดโดยเราจะใช้ RO หรือ R1 ที่มาจากการเลือกแบงค์แล้ว สำหรับ กรณีอ้างแอดเดรส 8 บิต, หรือ Stack Pointer ด้วย

แต่กรณีจะอ้างแอดเดรส แบบ 16 bit สามารถใช้ Register 16 บิต data pointer", DPTR ได้เท่านั้น

คำสั่ง Register (Register Instructions)

ใน Register bank, จะประกอบไปด้วย RO-R7 การอ้างถึงคำสั่งโดยผ่านทางรีจีสเตอร์ นั้น ทำให้การทำงานมีประสิทธิภาพขึ้น ตัวปัญหา เรื่อง addressbyte เมื่อมีการ execute คำสั่ง 1 ใน 8 ของรีจีสเตอร์ในแบงค์ที่เราเลือกใช้ จะถูกอ้างถึง เราสามารถเลือกแบงค์ได้โดย Set bank Select ของ bank ที่ PSW

Register-Specific instructions

ในบางคำสั่งจะถูกกำหนดให้กระทำทางรีจีสเตอร์ที่แน่นอนไปเลย เช่น ในบางคำสั่งจะให้กระทำเฉพาะใน Accumulator เท่านั้น ในบางคำสั่งใช้เฉพาะ data pointer เป็นต้นใน

คำสั่งจะอ้างถึง Accumulater ด้วย สัญญลักษณ์ "A" ใน assembler ซึ่งเป็น accumulater -specific apcodes

IMMEDIATE CONSTANTS

จัดการข้อมูลที่เป็นค่าคงที่ MOV A, #100 คือ ให้ค่าในรีจีสเตอร์ A ด้วยค่าคงที่ 100 (ฐาน 10) หรือเท่ากับ 64H ในฐาน 16

INDEXED Addressing

เฉพาะในส่วนของ Program memory เท่านั้น ที่สามารถอ้างแอดเดรสแบบ indexed ได้ (ตัวแบบตัวชี้) และอ่านได้อย่างเดียวเท่านั้น

โดยการใช้อ่านค่าจากตารางข้อมูล โดยจะกำหนดให้ 16-bit base register

มีอยู่ 2 ตัว คือ

- DPTR
- Program Counter

จะถูกกำหนดให้เป็นฐานของตาราง (base table) แล้วจะกำหนด รีจีสเตอร์ A ถูก Set ให้เลื่อนไปเลื่อนมา เพื่อข้อมูลในตาราง

เพื่อข้อมูลในตาราง

หลักการ : การอ่านข้อมูล (อ่าน) ใน Program memory โดยกำหนด

DDTR, Program Counter เป็นฐานของตาราง และบอกค่ากับรีจีสเตอร์เข้าไปกับ base table ก็จะได้ address ของตารางใน table ในส่วนของ Program memory ออกมา

ยังมีอีกกรณีในการตัวแบบ ตัวชี้ คือ กรณี "case Jump" ในชุดคำสั่ง โดยที่ค่าที่ปลายทางจะถูกต้องโดยการบอกค่าในรีจีสเตอร์ A กับ base pointer (จะได้ address นั้น)

ชุดคำสั่ง คณิตศาสตร์ (Arithmetic Intructions)

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
ADD A,<byte>	A = A + <byte>	X	X	X	X	1
ADDC A,<byte>	A = A + <byte> + C	X	X	X	X	1
SUBB A,<byte>	A = A − <byte> − C	X	X	X	X	1
INC A	A = A + 1	Accumulator only				1
INC <byte>	<byte> = <byte> + 1	X	X	X		1
INC DPTR	DPTR = DPTR + 1	Data Pointer only				2
DEC A	A = A − 1	Accumulator only				1
DEC <byte>	<byte> = <byte> − 1	X	X	X		1
MUL AB	B:A = B x A	ACC and B only				4
DIV AB	A = Int [A/B] B = Mod [A/B]	ACC and B only				4
DA A	Decimal Adjust	Accumulator only				1

ชุดในคำสั่งดังในตารางที่ 1 จะบอกแอดเดรส mode ต่าง ๆ ดังในตัวอย่าง

ตัวอย่าง

```
ADD  A,7FH      (A)   (A) + (7F)      (direct addressing)
ADD  A,@R0      (A)   (A) + ((R0))    (indirect addressing)
ADD  A,R7       (A)   (A) + (R7)      (register addressing)
ADD  A,#127     (A)   (A) + 127       (immediate constant)
```

ในตารางที่ 1 สมมติให้ในวงจร MCS-51 ใช้ clock ความถี่ 12 MH_Z แล้ว จะได้ว่าทุก ๆ คำสั่ง ในตาราง จะใช้เวลาในการกระทำคำสั่งนั้น ๆ เป็นเวลา 1 MS ยกเว้นคำสั่ง INC DPTR ใช้เวลา 2usec และคำสั่งคูณและหาร ใช้เวลา 4 usec

Note : ที่ไบต์ใด ๆ ในส่วนของ Internal data memory space เราสามารถที่จะเพิ่มหรือลดค่าภายในตัวมันเองโดยไม่ต้องผ่าน Accumulater

คำสั่ง INC บน 16 bit data pointer, DPTR ใช้สำหรับ การอ้างแอดเดรสแบบ 16 bit ใช้กรณี, external memory ฉนั้นการอ้างเพิ่มแบบเป็นการอ้างที่ให้ความสะดวก

คำสั่ง MUL AB เป็นการหาผลคูณระหว่าง ข้อมูลใน Accumulater กับ รีจีสเตอร์ B จะได้ผลลัพธ์เป็นแบบ 16 บิต ไว้ที่รีจีสเตอร์ B และ A (Accumulator)

คำสั่ง DIV AB จะหาร Accumulater ด้วยข้อมูลในรีจีสเตอร์ B จะเก็บผลที่ได้ (QUOTIENT) ในAccumulater ส่วนที่เหลือจากผลหาร (remainder) เก็บในรีจีสเตอร์ B

การหารอาจทำได้อีกแบบ โดย อาศัยหลักการ Strift ไปทางขวาจะหารด้วย 2 ไป

เรื่อย n ครั้ง ก็เท่ากับตัวหาร 2^n คำสั่ง Dir AB จะใช้เวลา 4 โดยที่ bit ที่ถูก stift ไปทางขวา จะเก็บอยู่ที่ register B (remainder นั่นเอง)

คำสั่ง DAA สำหรับ BCD arithmetic operation

ในการกระทำทางคณิตศาสตร์แบบ BCD นั้น ถ้ามีการใช้ ADD หรือคำสั่ง ADDC มักมีคำสั่ง DAA ตามมาด้วยเสมอ

NOTE : DAA ไม่ได้ convert จาก binary number ไปเป็น BCD แต่ DAA มีประโยชน์หลังจากที่เกิดการบวกกันของ BCD 2 จำนวนก่อน (First Step)

คำสั่งทางลอจิก LOGICAL instructions

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
ANL A,<byte>	A = A .AND. <byte>	X	X	X	X	1
ANL <byte>,A	<byte> = <byte> .AND. A	X				1
ANL <byte>,#data	<byte> = <byte> .AND. #data	X				2
ORL A,<byte>	A = A .OR. <byte>	X	X	X	X	1
ORL <byte>,A	<byte> = <byte> .OR. A	X				1
ORL <byte>,#data	<byte> = <byte> .OR. #data	X				2
XRL A,<byte>	A = A .XOR. <byte>	X	X	X	X	1
XRL <byte>,A	<byte> = <byte> .XOR. A	X				1
XRL <byte>,#data	<byte> = <byte> .XOR. #data	X				2
CRL A	A = 00H	Accumulator only				1
CPL A	A = .NOT. A	Accumulator only				1
RL A	Rotate ACC Left 1 bit	Accumulator only				1
RLC A	Rotate Left through Carry	Accumulator only				1
RR A	Rotate ACC Right 1 bit	Accumulator only				1
RRC A	Rotate Right through Carry	Accumulator only				1
SWAP A	Swap Nibbles in A	Accumulator only				1

จากตารางที่ 2 แสดงคำสั่งทางลอจิกของ MCS-51 เป็นการกระทำบูลลีน (Boolean Operations) AND, OR, Exceccusive OR, NOT

ตัวอย่าง ถ้า (A) = (00110101 B)

<byte> = (01010011 B)

ANL A, <byte>

ผลที่ได้ เมื่อดูผลในรีจิสเตอร์ A ได้ผลลัพท์ดังนี้ (A) = (00010001 B)

ANK A, <byte>

การใช้คำสั่งนั้น สามารถเข้าออก <byte> ได้หลายแบบเช่น

```
ANL A, 7FH      (direct addressing)
ANL A, @R1      (indirect addressing)
ANL A, R6       (Register addressing)
ANL A, #53H     (immediate constant)
```

NOTE : การกระทำแบบบูลลีน (Boolean operations) สามารถกระทำ บนไบต์ใด ๆ ใน Internal Data memory โดยไม่ต้องผ่านรีจีสเตอร์ A ก็ได้ เช่น WRL <byte>,#data ทำให้รวดเร็ว

```
XRL P1, #0FFH
```

• ถ้า operation นั้น ตอบสนองตาม interupt, อย่าใช้ Accumubtor เก็บ (Save time) ควรเก็บค่าต่าง ๆ ลงใน Stack สำหรับการบริการโปรแกรมย่อย

คำสั่ง Rotate RLA, RLC A เป็นการเลื่อน Accumulator ไป 1 บิต ทางซ้าย (หรือขวา) กรณี left บิตสูงจะเลื่อนวนไปทางบิตต่ำ ส่วนบิตต่ำ เลื่อนไปแทนบิตสูง

คำสั่ง SWAP A เป็นคำสั่งทำหน้าที่สลับ 4 บิตสูง กับ 4 บิตต่ำ ของ Accumulator มีประโยชน์กรณีการกระทำแบบ BCD

ตัวอย่าง

ถ้า Accumulator เก็บค่า binary number ซึ่งมีค่าน้อยกว่า 100 เราสามารถแปลงเป็น BCD ได้ ดังนี้

```
MOV     B, #10
DIV     AB
```

```
SWAP    A
ADD     A,B
```

การหารด้วย 10 จะให้ผลอยู่ที่ 4 บิตล่างของ Accumulator ขณะที่เศษที่เหลืออยู่ที่ register B หลังจากนั้นก็สลับ 4 bit สูงกับ 4 บิตต่ำของ Accumulator แล้วก็นำผลจาก Register B มาบวก จะได้ รูปแบบของ (binary) HEX BCD

เช่น ใน register A (A) = (20H) Hex

(A) = (32) decimale

การโอนย้ายข้อมูล Data Transfers

การโอนย้ายข้อมูล ภายใน Internal Ram

ตารางที่ 3

A list of the MCS-51 data transfer Instructions that Access internal data memory Space

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
MOV A,<src>	A = <src>.	X	X	X	X	1
MOV <dest>,A	<dest> = A	X	X	X		1
MOV <desi>, <src>	<dest> = <src>	X	X	X	X	2
MOV DPTR,#data16	DPTR = 16-bit immediate constant.				X	2
PUSH <src>	INC SP : MOV "@SP",<src>	X				2
POP <dest>	MOV <dest>, "@SP" : DEC SP	X				2
XCH A,<byte>	ACC and <byte> exchange data	X	X	X		1
XCHD A,@Ri	ACC and @Ri exchange low nibbles		X			1

จากตารางที่ 3 แสดงชุดคำสั่งที่ทำหน้าที่โอนย้ายข้อมูลภายใน internal memory

คำสั่ง MOV <dest>, (src> อนุญาติให้มีการดอนย้ายข้อมูลระหว่าง internal RAM หรือ SFR โดยไม่ผ่านรีจีสเตอร์ A ได้

• ข้อควรคำนึงไว้เสมอว่า 128 ไบต์บนของ data Ram อ้างถึงเฉพาะแบบ indirect เท่านั้น และในส่วน SFR on ได้แบบ direct เท่านั้น

NOTE : ในทุก ๆ เบอร์ตระกูล MCS-51 ไมโครโปรเซสเซอร์นั้น สแตค (stack) จะอยู่ภายในชิพของหน่วยความจำภายในอยู่แล้ว RAM ทุกครั้งที่ใช้จะมีการเพิ่มค่า (grows upwards)

- คำสั่ง PUSH ทุกครั้งที่มีการใช้คำสั่งนี้ มีการกระทำดังนี้เกิดขึ้น

    1. เพิ่มค่า Stack Pointer (SP)

    2. เอาข้อมูลจาก (Src) ไปใช้ใน Stack

- คำสั่ง POP

    1. เอาข้อมูลจาก Stack ไปเก็บไว้ที่ <lest>

    2. ลดค่าใน Stack Pointer (SP)

คำสั่ง PUSH, POP ใช้กรณีแบบ direct เท่านั้น เพื่อแสดงความแน่ใจว่าเป็นข้อมูลที่เรา save หรือ restord เอาไว้ แต่ตัว SP เอง สามารถอ้างแอดเดรสแบบ indirect ได้โดยใช้ SP register เราสามารถกำหนดให้ใช้ 128 ไบต์บนได้ ด้วยการเขียนคำสั่ง แต่ต้องไม่ใช่ ในส่วนของ SFR space

128 ไบต์บนไม่สามารถถูกใช้ได้บน 8051, 8051AH, หรือ 80C51BH รวมทั้ง Romless หรือ EPROM Version

ถ้าใช้ SP ชี้ไปที่ 128 ไบต์บน ถ้าใช้ PUSH ข้อมูลจะหาย ถ้า POP ก็จะได้ข้อมูลที่ไม่ต้องการออกมา

การโอนย้ายข้อมูลรวม 16 บิต addr. Mov เอาไว้ด้วย โดยเริ่มต้น DPTR ให้ชี้ไปที่ส่วนของ TABLE ของ Program Memory หรือ 16 บิต ในส่วน external Data Memory

คำสั่ง XCH A, <byte> ทำให้เกิดการเปลี่ยนข้อมูลระหว่างกัน คือระหว่างรีจีสเตอร์ A กับ <byte> ส่วน XCHD A, CRi ก็คล้าย ๆ กัน แต่จะเปลี่ยนข้อมูลระหว่างกัน แค่ 4 บิตล่างเท่านั้น

อย่างแรก ปัญหาการชิพ (เลื่อน) 8 bit BCD (ตัวเลข 2 ตัว) ไปทางขวาตามรูปที่ 9 ซึ่งจะเปรียบเทียบกับการใช้คำสั่ง MOV_S และ XCH_S ซึ่งผลสุดท้ายจะได้บรรทัดสุดท้าย ในการเลื่อนเหมือนกัน แต่การใช้ XCH จะประหยัดจำนวนไบต์คำสั่ง และใช้เวลาน้อยกว่า ถ้าใช้คำสั่ง MOV จะเสียเวลา 9 uSec ถ้า XCH เสียเวลา 5uSec เมื่อเทียบกับสัญญาณนาฬิกา 12 MH_Z

ประโยชน์ของ XCH กับ XCHD ดังในรูปที่ 9

		2A	2B	2C	2D	2E	ACC
MOV	A,2EH	00	12	34	56	78	78
MOV	2EH,2DH	00	12	34	56	56	78
MOV	2DH,2CH	00	12	34	34	56	78
MOV	2CH,2BH	00	12	12	34	56	78
MOV	2BH,#0	00	00	12	34	56	78

(a) Using direct MOVs: 14 bytes, 9 μs

		2A	2B	2C	2D	2E	ACC
CLR	A	00	12	34	56	78	00
XCH	A,2BH	00	00	34	56	78	12
XCH	A,2CH	00	00	12	56	78	34
XCH	A,2DH	00	00	12	34	78	56
XCH	A,2EH	00	00	12	34	56	78

(b) Using XCHs: 9 bytes, 5 μs

รูปที่ 9 Shifting = BCD Number Two Ligits to the right

<u>ตัวอย่าง</u> การใช้คำสั่งเลื่อนตัวเลข BCD number ไปทางขวา 1 ตัวเลข โดยใช้คำสั่ง XCHD ดังรูปที่ 10

		2A	2B	2C	2D	2E	ACC
	MOV R1,#2EH	00	12	34	56	78	XX
	MOV R0,#2DH	00	12	34	56	78	XX
	loop for R1 = 2EH:						
LOOP:	MOV A,@R1	00	12	34	56	78	78
	XCHD A,@R0	00	12	34	58	78	76
	SWAP A	00	12	34	58	78	67
	MOV @R1,A	00	12	34	58	67	67
	DEC R1	00	12	34	58	67	67
	DEC R0	00	12	34	58	67	67
	CJNE R1,#2AH,LOOP						
	loop for R1 = 2DH:	00	12	38	45	67	45
	loop for R1 = 2CH:	00	18	23	45	67	23
	loop for R1 = 2BH:	08	01	23	45	67	01
	CLR A	08	01	23	45	67	00
	XCH A,2AH	00	01	23	45	67	08

จากรูปที่ 10 พ้อยเตอร์ (pointer) R1, และ R0 จะถูกเซ็ตให้ชี้ไปที่ 2 ไบต์ที่เก็บตัวเลขแบบ BCD แล้ว รูปแบบการเลื่อน จะให้ข้อมูลตัวเลขเลื่อนไปทางขวา 1 ตัว และให้ทางซ้ายสุดมีศูนย์มาแทนที่

Address Width	Mnemonic	Operation	Execution Time (μs)
8 bits	MOVX A,@Ri	Read external RAM @Ri	2
8 bits	MOVX @Ri,A	Write external RAM @Ri	2
16 bits	MOVX A,@DPTR	Read external RAM @DPTR	2
16 bits	MOVX @DPTR,A	Write external RAM @DPTR	2

ตารางที่ 4

[A List of the MCS-51 Data Transfer Instructions that]

Access External Data Memory Space

External RAM

จากตารางที่ 4 แสดงชุดคำสั่งการโอนย้ายข้อมูลเฉพาะ external data memory RAM

• เป็นการอ้างแอดเดรสแบบ indirect เท่านั้น ต้องการอ้างแบบ 8 บิตก็ใช้ Ri อาจจะเป็น R0 หรือไม่ก็ R1 ตัวใดตัวหนึ่ง สามารถเลือก Bank ที่เราต้องการได้

ถ้าเป็นการอ้างแบบ 16 บิต (2 ไบต์) , @DPTR

ข้อเสียของการอ้าง address แบบ 16 บิต คือต้องใช้ Port 2 ทั้งหมด

ถ้าใช้ RAM ภายนอกเพียงไม่กี่กิโลไบต์ (K byte) แต่ถ้าให้อ้างแบบ 8 บิต เหมือนกรณีรูปที่ 3 ทำให้ P2 เหลืออยู่ทำให้นำไปใช้งานอื่น ๆ ได้อีก

NOTE : ในการอ้าง external DATA RAM access, Accumulator จะถูกกำหนดให้ปลายทาง (destination) หรือไม่ก็ต้นทาง (source) ของข้อมูลเสมอ

สัญญาณ strobe ทั้งขา RD และขา WR จะตอบสนองและส่งสัญญาณ RD หรือ WR ออกไปยัง external RAM จะมีก็ต่อเมื่อใช้คำสั่ง MOVX

• ซึ่งในกรณีที่ไม่มีการใช้สัญญาณทั้งสัญญาณนี้เลย ขาเหล่านี้จะถูกกำหนดเป็น I/O line ได้ (หน้า 5-11)

LOOKUP TABLES

Mnemonic	Operation	Execution Time (μs)
MOVC A,@A+DPTR	Read Pgm Memory at (A+DPTR)	2
MOVC A,@A+PC	Read Pgm Memory at (A+PC)	2

ตารางที่ 5 Lookup table Read Instructions

จากตารางที่ 5 จะเห็นว่า มีคำสั่ง 2 คำสั่งที่จะสามารถอ่านค่าจากตารางในส่วน Program Memory ได้ เพียงถูกออกแบบมาให้อ้างถึงได้เฉพาะส่วน Program Memory ตาราง สามารถอ่านได้เพียงอย่างเดียวและไม่สามารถแก้ไขได้ สัญญลักษณ์ "MOVC" ก็คือ "MOVC CONSTANT"

ถ้ามีการอ้างถึง external program memory สัญญาณ Read strede คือ PSEN จะ active

คำสั่ง MOVE คำสั่งแรกจากตาราง สามารถอ้างข้อมูลอยู่ในช่วง 0-255 ได้ทั้งหมด 256 ตำแหน่ง DPTR จะเก็บค่าของ data pointer เก็บค่าชี้ที่จุดเริ่มต้นของตาราง และ จำนวนของข้อมูลที่จะอ้างในตารางก็เก็บเอาไว้ในรีจิสเตอร์ A ดังนี้

```
MOVE A, @A+DPTR
```

ข้อมูลที่ได้มาจะเก็บค่าไว้ในรีจิสเตอร์ A

ส่วนคำสั่ง MOVE คำสั่งที่สอง การทำงานเหมือนคำสั่งแรก ยกเว้น

- ยกเว้น Program Counter (PC) จะถูกใช้เป็นฐานของตาราง และตารางจะถูกอ้าง ผ่านทางโปรแกรมย่อย (Subroutine)

ค่าแรกที่ต้องการ (ข้อมูล) จะถูกนำมาเก็บไว้ใน Accumulator ดังนี้

```
MOV  A, ENTRY_NUMBER
CALL TABLE
```

โปรแกรมย่อย TABLE คือ

```
TABLE : MOVC  A,@A+PC
        RET
```

ซึ่งเห็นว่า TABLE จะจบด้วยคำสั่ง RET (return) ใน Program memory

การอ้างข้อมูลแบบนี้อ้างข้อมูลทั้งหมด (entrics) ได้ 255 entrics 1-255 entory ที่ 0 ใช้ไม่ได้ เพราะว่า เวลาที่คำสั่ง MOVE กำลังทำงาน PC จะเก็บตำแหน่งของคำสั่ง RET ฉนั้น entry numbers จะเป็นคำสั่ง RET จากตัวมันเอง

คำสั่ง บูลลีน Boolean Operations

MCS-51 ไมโครโปรเซสเซอร์ จะมีส่วนทำหน้าที่ Boolean (Single-bit) processor โดยที่ ภายใน Internal Ram จะอ้างข้อมูลได้ 128 address-bit และในส่วน SFR สามารถทำได้อีก 128 แอดเดรสบิตและทุกพอร์ต (0-3) สามารถอ้างข้อมูลได้แบบบิตแอดเดรส แยกออกจากกัน คำสั่งการอ้างแอดเดรสบิตไม่ใช่สำหรับการกระโดดแบบเงื่อนไขเท่านั้น แต่ยังมีคำสั่งต่าง ๆ อีกเช่นคำสั่ง move, set, clear, complement, OR, และ AND

ชุดคำสั่งสำหรับบูลลีนโปรเซสเซอร์ แสดงในตารางที่ 6 ทุกบิตมีการอ้างถึงแบบ direct แอดเดรส บิตแอดเดรสตั้งแต่ 00H ถึง 7FH ในส่วนของ 128 ไบต์แรกของ Internal RAM และบิตแอดเดรสตั้งแต่ 80H-FFH ในส่วนของ SFR Space จากตัวอย่างเรา สามารถนำค่า flag ภายในย้ายออกไปที่พอร์ต pin ได้ดังนี้

```
MOV   C, FLAG
MOV   P1.0,C
```

อธิบายตัวอย่าง : FLAG เป็นชื่อของบิตแอดเดรสใด ๆ ในส่วน 128 ไบต์แรกของ RAM ภายใน หรือ SFR space, ในส่วน I/O (LSB ของ พอร์ต 1 ในตัวอย่าง) จะถูก set หรือ cleared ขึ้นอยู่กับบิตของแฟล็ค ว่าจะเป็นลอจิก 1 หรือ 0

ตารางที่ 6 A List of the MCS-51 Boolean Instructions

Mnemonic		Operation	Execution Time (μs)
ANL	C,bit	C = C .AND. bit	2
ANL	C,/bit	C = C .AND. .NOT. bit	2
ORL	C,bit	C = C .OR. bit	2
ORL	C,/bit	C = C .OR. .NOT. bit	2
MOV	C,bit	C = bit	1
MOV	bit,C	bit = C	2
CLR	C	C = 0	1
CLR	bit	bit = 0	1
SETB	C	C = 1	1
SETB	bit	bit = 1	1
CPL	C	C = .NOT. C	1
CPL	bit	bit = .NOT. bit	1
JC	rel	Jump if C = 1	2
JNC	rel	Jump if C = 0	2
JB	bit,rel	Jump if bit = 1	2
JNB	bit,rel	Jump if bit = 0	2
JBC	bit,rel	Jump if bit = 1; CLR bit	2

ส่วน Carry bit ใน PSW จะถูกใช้เป็น Accumulator 1 บิต สำหรับการกระทำแบบบูลลีน (Boolean) ของส่วน Boolean processor

• คำสั่งเกี่ยวกับบิตนั้นจะอ้างถึง Carry bit ด้วยตัว C เป็น Carry-Specific instructions (CLRC, etc). Carry bit ตัวแบบโดยตรงได้ด้วย เนื่องด้วยมันอยู่ภายใน PSW register ซึ่งมีการตัวแบบ bit addressable

NOTE : คำสั่งบูลลีน จะรวมเอา ANL และ ORL แต่ไม่นำ XRL มาใช้ซึ่ง XRL สามารถเขียนได้ด้วยโปรแกรม

ตัวอย่างถ้าเราต้องการรูปแบบการ Exclusive OR สำหรับ บิต 2 บิต

C = bit1. XRL. bit2

สามารถแทนด้วย ซอฟท์แวร์ดังนี้

```
        MOV     C, bit1
        JNB     bit2, OVER
        CPL     C
OVER :  (Continued)
```

อธิบายตัวอย่าง : ขั้นแรก เอาบิต 1 ไปเก็บไว้ใน Carry ถ้าบิต 2=0 แล้ว C จะได้ค่าที่ถูกต้อง นั่นคือ bit1.XRL.bit2=bit1 ถ้า bit2=0 หรืออีกกรณีหนึ่งคือ ถ้า bit2=1, แล้ว C จะต้องคอมพลีเมนต์ก่อน จึงจะได้ค่าที่ถูกต้อง

RELATIVE OFFSET

การกำหนดปลายทางในการกระโดดไปนั้น ตัวแอสเซมเบลอร์ จะกำหนดได้เป็น 2 แบบ ลาเบล (LABEL) หรือไม่ก็เป็น address จริง ๆ กันเลย อย่างไรก็ตาม การแปลงแอดเดรสปลายทางจะถูกแปลงเป็น Relative offset byte This is a signed (two's complement) offset byte which is added to the PC in two's complement arithmetic if the jump is executed

ช่วงของการกระโดดอยู่ในช่วง -128 ถึง +127 ใน Program memory โดยจะสัมพัทธ์กับแอดเดรสของคำสั่งนั้น

(-128 to +127 program memory bytes relative to the first byte following the instruction

Jump Instructions คำสั่งการอ้างกระโดด

Mnemonic	Operation	Execution Time (μs)
JMP addr	Jump to addr	2
JMP @A+DPTR	Jump to A+DPTR	2
CALL addr	Call subroutine at addr	2
RET	Return from subroutine	2
RETI	Return from interrupt	2
NOP	No operation	1

ตารางที่ 7 Unconditional Jumps in MCS-51 Devices

ในตารางจะแสดง IMP คำสั่งเดียวเท่านั้น แต่ความจริงแล้วมีความสั่งอยู่ 3 คำสั่ง คือ SJMP, LJMP, AJMP มีรูปแบบ (Format) ในการใช้กระโดดไปที่ปลายทางต่างกัน

คำสั่ง JMP เป็น Memonic ทั่วไป (เป็นตัวแทน) กลุ่มคำสั่ง JMP ซึ่งสามารถใช้ได้ถ้าโปรแกรมเมอร์ ไม่สนใจว่าจะกระโดดไปทางทิศทางไหน ช่วงไหน

คำสั่ง SJMP กำหนดช่วงปลายทางของแอดเดรสในช่วง relative offset ความ

ยาวของ คำสั่ง 2 ไบต์ ประกอบด้วย opcode และ relative offset byte ช่วงการกระโดด -128 to +127 bytes Relative ที่ตามหลังคำสั่ง SJMP คำสั่ง LJMP กำหนดช่วงแอดเดรส 16 bit ความยาวของคำสั่ง 3 ไบต์ คือ opcode 1 ไบต์ และ 2 ไบต์ สำหรับแอดเดรส กระโดดได้ในช่วง 64 K ที่ใดก็ได้ในช่วง 64 Kbytes นี้ ใน Program memory Space

คำสั่ง AJMP กำหนดช่วงปลายทางด้วย 11 บิต แอดเดรส ความยาวคำสั่ง 2 ไบต์ ประกอบไปด้วย opcode ซึ่งจะเก็บ 3 บิต บนของแอดเดรสไว้ด้วย และอีก 1 ไบต์จะเก็บ 8 บิต ไบต์ต่ำของแอดเดรสที่จะก้าวกระโดดไป เมื่อคำสั่งถูก executed 11 บิต แอดเดรส ที่เก็บเอาไว้นี้จะเอาไปแทนที่ใน 11 บิต ใน PC 5 บิต สูงของ PC ยังคงเหมือนเดิม ซึ่งคำสั่ง AJMP จะใช้ก้าวกระโดดภายในช่วง 2K เท่านั้น

ในทุกกรณี (The all case) โปรแกรมเมอร์กำหนดแอดเดรสปลายทางในแอสเซมเบอร์วิธีที่เหมือนกันได้เลย คืออาจจะเป็น Lable หรือไม่ก็ 16-bit constant แอสเซมเบอร์ก็จะใส่ค่าแอดเดรสปลายทางให้ในรูปแบบ (Format) ที่ถูกต้องให้เลย แต่ถ้าเรากำหนด Format ไม่ถูกต้องในการกระโดดไปที่แอดเดรสปลายทาง

ข้อความ "Destination out of range" จะถูกแต่เอาไว้ที่ List file

คำสั่ง JMP @A+DPTR มีไว้สนับสนุนการกระโดดแบบมีเงื่อนไข หรือกรณี (case) แอดเดรสปลายทางจะถูกคำนวนเมื่อมีการ excuted คำสั่ง โดยแอดเดรสปลายทางมาจากผลบวกของ 16 บิต DPTR รีจิสเตอร์ และแอคคิวมูเลเตอร์ โดยทั่วไป DPTR จะถูกใช้เป็นตัวบอกแอดเดรสของ TABLE ส่วนแอคคิวมูเลเตอร์ จะกำหนดเป็นตัวชี้ตาราง

<u>ตัวอย่าง</u>

```
            MOV     DPTR, # JUMP_TABLE
            MOV     A, INDEX_NUMBER
            RL      A
            JMP     @A+DPTR
JUMP_TABLE
            AJMP    CASE_0
            AJMP    CASE_1
            AJMP    CASE-2
            AJMP    CASE-3
            AJMP    CASE-4
```

จากตัวอย่างเป็นการกระโดด 5m (5-way branch) ค่า 0-4 ซึ่งจะถูก load เข้าไปเก็บไว้ในแอคคิวมูเลเตอร์

คำสั่ง RLA จะ Convert ตัว index-number (0-4) โดยจะ Rotate จำนวนคู่ 0-ถึง 8 เพราะว่า แต่ละตำแหน่งในตารางห่างกัน 2 ไบต์ พอดี

จากตารางที่ 7 คำสั่ง CALL addr หมายถึง LCALL และ ACALL CALL in a generic mnemonic which can be used if the programmer does not care which way the address is encoded

คำสั่ง LCALL ใช้ 16 บิตแอดเดรส สามารถเรียกโปรแกรมย่อยได้ในช่วง 64K ไบต์ ที่ใด ๆ ใน Program memory Space

คำสั่ง ACALL ใช้ 16 แอดเดรส สามารถเรียกโปรแกรมย่อยได้ในช่วง 2K ไบต์ ที่ใด ๆ ใน Program memory Space คำสั่ง ACALL ใช้ 11 บิต แอดเดรส สามารถเรียกโปรแกรมย่อยได้ในชวง 2K ไบต์ (block)

เราสามารถกำหนดแอดเดรสของโปรแกรมย่อย ได้ด้วย LABEL หรือค่าคงที่ 16 บิตแอสเซมเบลอร์ จะใส่แอดเดรสในรูปแบบที่ถูกต้องให้เลย การใช้โปรแกรมย่อยจะต้องลบด้วย คำสั่ง RET ซึ่งจะ Return คำสั่งมายังคำสั่งต่อไปที่ถัดมาจากคำสั่ง CALL

คำสั่ง RETI จะถูกใช้กรณีการกระโดดกลับจาก interrupt service routine

ข้อแตกต่างระหว่าง RET และ RETI คือ

• RETI tells the interrupt control System that the interrupt in progress is done. if there is no interrupt in progress at the time RETI is executed, THEN the RETI is functionally identical to RET

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
JZ rel	Jump if A = 0	Accumulator only				2
JNZ rel	Jump if A ≠ 0	Accumulator only				2
DJNZ <byte>,rel	Decrement and jump if not zero	X		X		2
CJNE A,<byte>,rel	Jump if A ≠ <byte>	X			X	2
CJNE <byte>,#data,rel	Jump if <byte> ≠ #data		X	X		2

ตารางที่ 8 Conditional Jumps in MCS-51 Devices

จากตารางที่ 8 แสดงการกระโดดแบบมีเงื่อนไข ทุกคำสั่งการกระโดดแบบมีเงื่อนไขแอดเดรสปลายทางเป็นแบบ relative offset ช่องการก้าวกระโดด -128 to+127 ไบต์จากคำสั่งที่ตามหลังคำสั่ง Jump Condition Important to note : however, the user specifics to the actual destiration address the same way as the other Jumps : as a label or a 16-bit constant

คำสั่ง DJNZ ลดค่า และก้าวกระโดดกรณีที่ยังมีค่าไม่เท่ากับศูนย์เป็นคำสั่งสำหรับ loop control

กรณีที่จะส่งให้กระทำลูป N ครั้ง โดยการโหลดค่าที่ต้องการลงไป เช่น

ตัวอย่าง

N = 10 จะได้ว่า

```
        MOV     COUNTER, #10
LOOP :  (begin loop)
        .
        .
        (end loop)
        DJNZ  COUNTER, LOOP
        (continue)
```

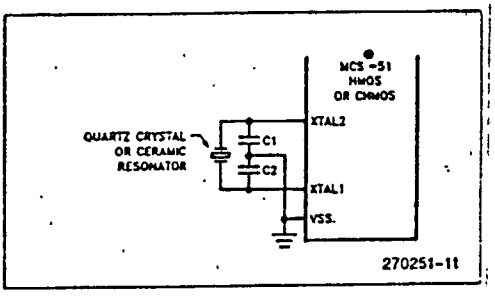
คำสั่ง CJNE เป็นคำสั่งเปรียบเทียบค่า และจะก้าวกระโดดเมื่อทั้ง 2 ค่า ที่มาเปรียบเทียบมีค่าไม่เท่ากัน มีประโยชน์สำหรับ loop control เช่นกัน ตัวอย่างในรูปที่ 10 ต้องมี operand 2 ตัว และจะก้าวกระโดดเมื่อ operand ทั้งสองตัวมีค่าไม่เท่ากัน operand ตัวแรกคือ ข้อมูลที่เก็บไว้ใน R1 ค่าเริ่มต้นคือ 2EH และ operand ตัวที่ 2 คือ 2AH ทุกครั้งที่กระทำ loop R1 จะถูกลดค่า ลูปจะถูกกระทำจนกระทั่งข้อมูลใน R1 มีค่าเท่ากับ 2AH

การนำไปใช้อย่างอื่น เช่น เปรียบเทียบมากกว่าหรือน้อยกว่า โดยเปรียบจำนวน 2 จำนวน เป็น จำนวนเต็มที่ไม่คิดเครื่องหมาย (unsinged integer) ถ้าจำนวนแรกน้อยกว่าจำนวนที่ 2 Carry บิต จะถูกเซ็ตให้เป็น "1" ถ้าจำนวนแรกมีค่ามากกว่าหรือเท่ากับ Carry bit จะถูกเคลียร์ Clear ให้เป็นศูนย์

CPU Timing

MCS-51 ทุกตัว จะมีวงจรกำเนิดสัญญาณ (oscillator) ภายในตัว IC อยู่แล้ว

เราเพียงต่อตัว Resonater เช่น Crytal หรือไม่ก็ Ceramic resanater ระหว่างขา XTAL1 และ ขา XTAL2 เท่านั้น และก็ต่อตัวเก็บประจุดังรูปที่ 11



รูปที่ 11

กรณีที่เราต้องการใช้ oscillator จากภายนอก เราสามารถต่อได้ดังรูปที่ 12 ตัวกำเนิดสัญญาณนาฬิกา ภายใน MCS-51 ทำให้เกิดขบวนของ Machin cycle ดังนี้

Machine Cycle

ใน 1 machine Cycle ประกอบไปด้วย sequence 6 states

คือ สเตรท S1 ถึง S6 แต่ละเสตรทประกอบด้วย 2 คาบเวลา

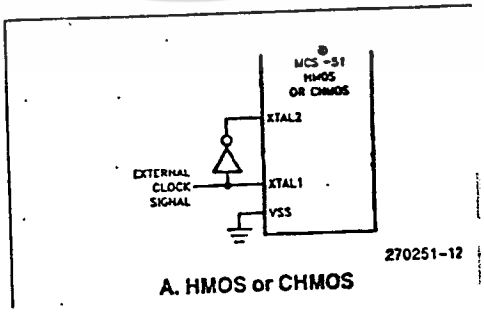
(2 period)

1 machin cycle = 6 state (S1-S6)

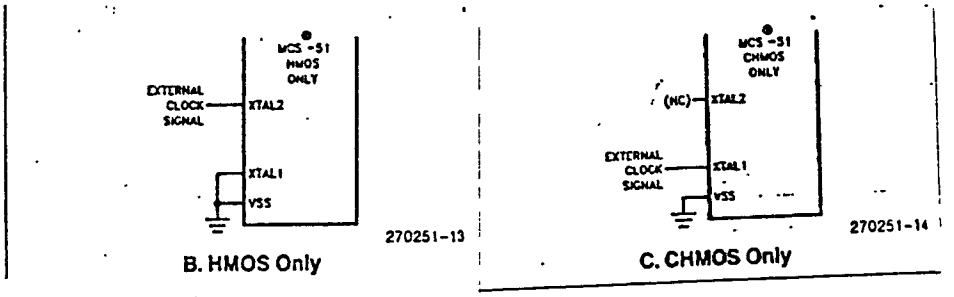
1 State = 2 period

จะได้ว่า 1 machine Cycle จะมี 12 period กรณีที่ใช้ความถี่ 12 MH_Z 1 machine Cycle นี่ก็คือ 1uSec นั่นเอง ดังรูปที่ 13 (5-15)

โดยทั่วไปจะมีการ fetch คำสั่งเสตรทที่ 1 และ 4 (S1 และ S4) และเมื่อ State ถึง S6 การ execute จะเสร็จสิ้นด้วย ใน Machine Cycle เดียวกัน



A. HMOS or CHMOS



รูปที่ 12 Using an External clock

3. คู่มือนำทางสำหรับนักโปรแกรมและชุดคำสั่งของ 8051

(MCS-51 Programmer's Guide and Instruction Set)

Memory organization

Program memory

8051 (รวมทั้ง 8031 ด้วย) ได้แยกส่วนแอดเดรสเป็นสองส่วนคือ

- ส่วน Program memory
- ส่วน data memory

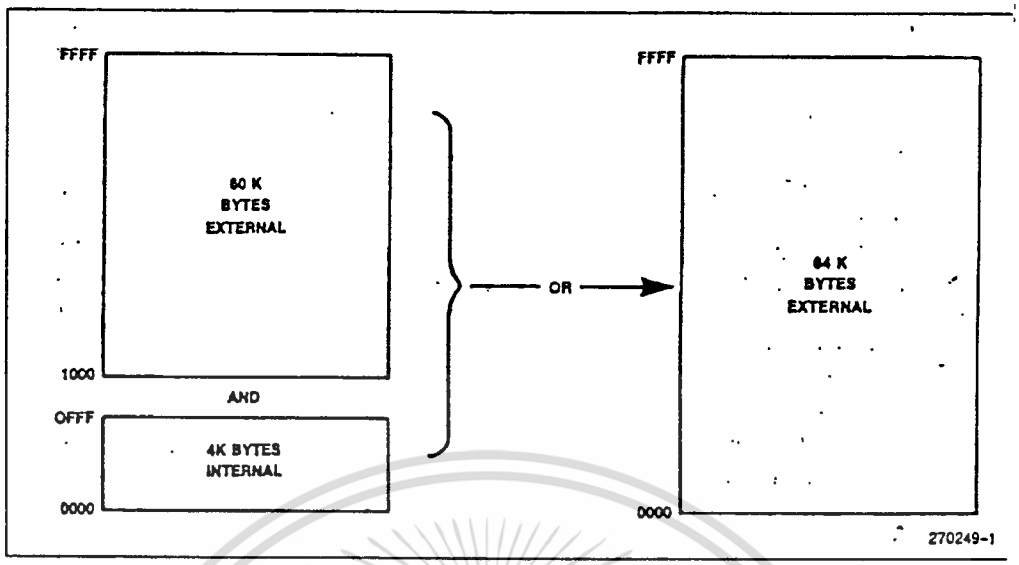
ส่วนของ Program memory อ้างได้ถึง 64 K

ในเบอร์ 8051 มีอยู่ภายในตัวมันแล้ว 4K (8052 มี 8K) ส่วน 8031 ไม่มี Program memory ในตัว ต้องต่อใช้ด้านนอก

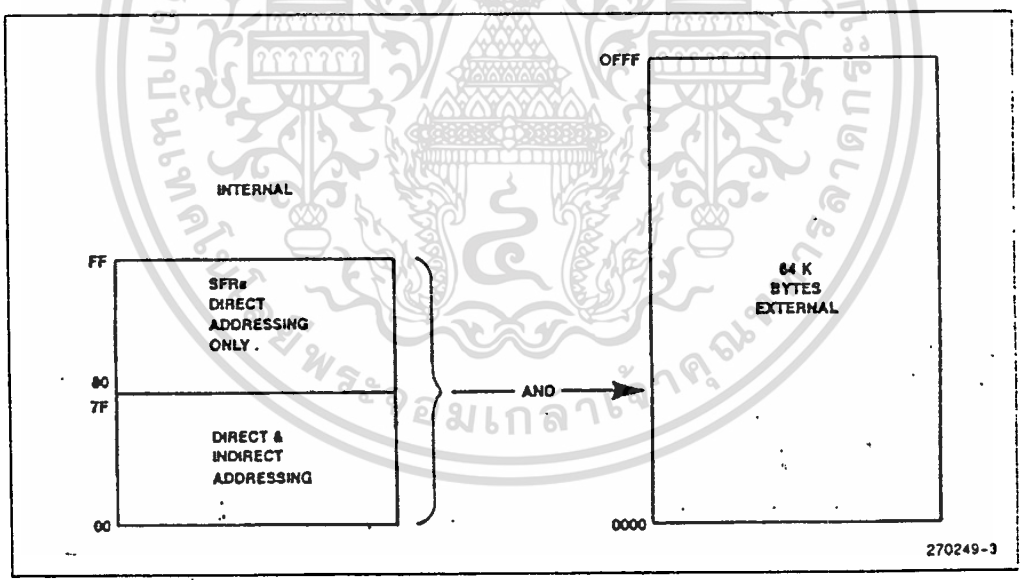
Data memory

ใน 8051 ส่วน data memory สามารถ อ้างได้ถึง 64K bytes ใน 8051 มี data memory ภายในตัว 128 bytes (RAM) (ส่วน 8052 มี 256 bytes) เบอร์ 8031 มี data memory ภายในตัว 128 byte เช่นกัน และยังมีส่วน Special Function Registers (SFR_S)

โดยที่ 128 byte ต่ำเป็นส่วนของ RAM และ 128 ไบต์ บนใช้กรณี SFR_S การใช้คำสั่ง "MOVX" เป็นการอ้างถึงแอดเดรสนี่เป็นส่วน external data memory 128 byte ต่ำ (RAM) สามารถอ้างได้ทั้งแบบ direct (MOV data addv) หรือไม่ก็ indirect (MOV @Ri) ก็ได้ ส่วน SFR ตัวได้แบบ direct เท่านั้น



รูปที่ 13 The 8051 Program Memory

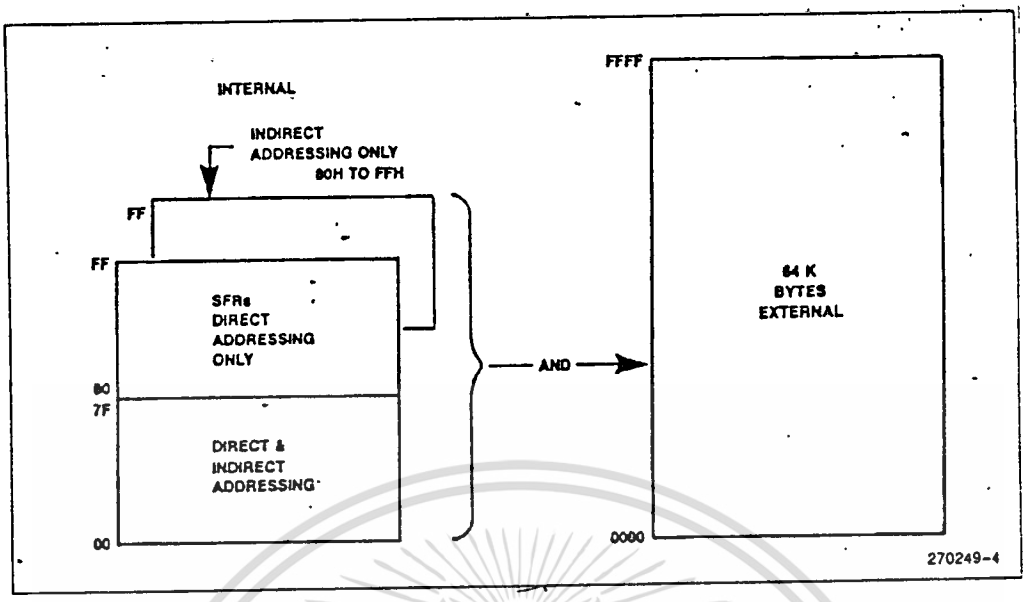


รูปที่ 14 แสดงส่วน data memory ของ 8051 (8031)

Indirect ADDRESS AREA

จะเห็นว่า ส่วนที่เป็น SFR_s และส่วน indirect addressing จะอยู่บนแอดเดรสเดียวกัน 80-FFH มองเหมือนว่าซ้อนทับกันอยู่ แต่อย่างไรก็ตาม เราสามารถแยกพื้นที่ออกจากกันได้โดยเด็ดขาด ขึ้นอยู่กับการใช้คำสั่ง สำหรับกรณี

กรณี 8052 ส่วนของ data memory ดังในรูปที่ 15



รูปที่ 15 8052 data memory

จากตัวอย่าง

```
MOV    80H, # 0AAH
```

จะเห็นว่า นั่นคือการเอาค่า 0AAH ไปใช้ที่ address 80H (หรือจากตารางแอดเดรส ของส่วน SFR_S ที่ต้องแบบ direct เท่านั้น คือ Port0 นั่นเอง) นั่นคือตอนนี้ที่ Port0 มีข้อมูล 0AAH อยู่

และจากตัวอย่าง

```
MOV    R0, #80H
MOV    @R0, #0BBH
```

นั่นคือใส่ข้อมูลไปที่ address 80H เช่นกัน ในส่วนของ data RAM ถ้าเรา execute คำสั่ง ทั้งสองชุด คือ 1 และ 2 เราจะได้ Port0 มี 0AAH ส่วนแอดเดรส 80H ของ RAM มี 0BBH (ทั้งที่ Port0 ก็คือ แอดเดรส 80H เช่นกัน) เราจะมองสองส่วนนี้แยกออกจากกัน สำหรับ 128 bytes บน กรณีนี้มีปัญหาเฉพาะของ 2 เท่านั้น กรณีของ 8051 (รวมทั้ง 8031) จะมีเฉพาะส่วน SFR_S เท่านั้น นั่นคือ 128 bytes บน สามารถอ้างได้แบบ direct เท่านั้น

ส่วนของ DIRECT AND INDIRECT ADDRESS AREA

128 bytes ล่าง สามารถอ้างได้ทั้ง direct และ indirect แบ่งเป็น 3 segments คือ

1. Register Banks 0-3 :

อยู่ตั้งแต่แอดเดรส 0-ถึง 1FH (32 ไบต์) หลังที่ผ่านการ Reset จะมีค่า default ไว้ที่ bank0 การที่จะเลือก bank ของ Register ผู้ใช้สามารถเลือกได้จากซอฟท์แวร์ และแต่ละรีจีสเตอร์แบงค์ จะมีรีจีสเตอร์ 8 ตัว (ตัวละไบต์) Register 0 ถึง 7 พอ Reset เริ่มต้น Stack Pointer จะอยู่ที่ 07H และจะเริ่มเพิ่มค่าที่ address 08H ซึ่งตรงกับตำแหน่ง R0 ของ Register bank1 ฉนั้น กรณีที่เราต้องการใช้ Register bank มากกว่า 1 bank เราต้องไปกำหนด SP register ให้ชี้ไปยัง address RAM ที่ไม่ใช้เก็บข้อมูล

2. Bit addressable AREA :

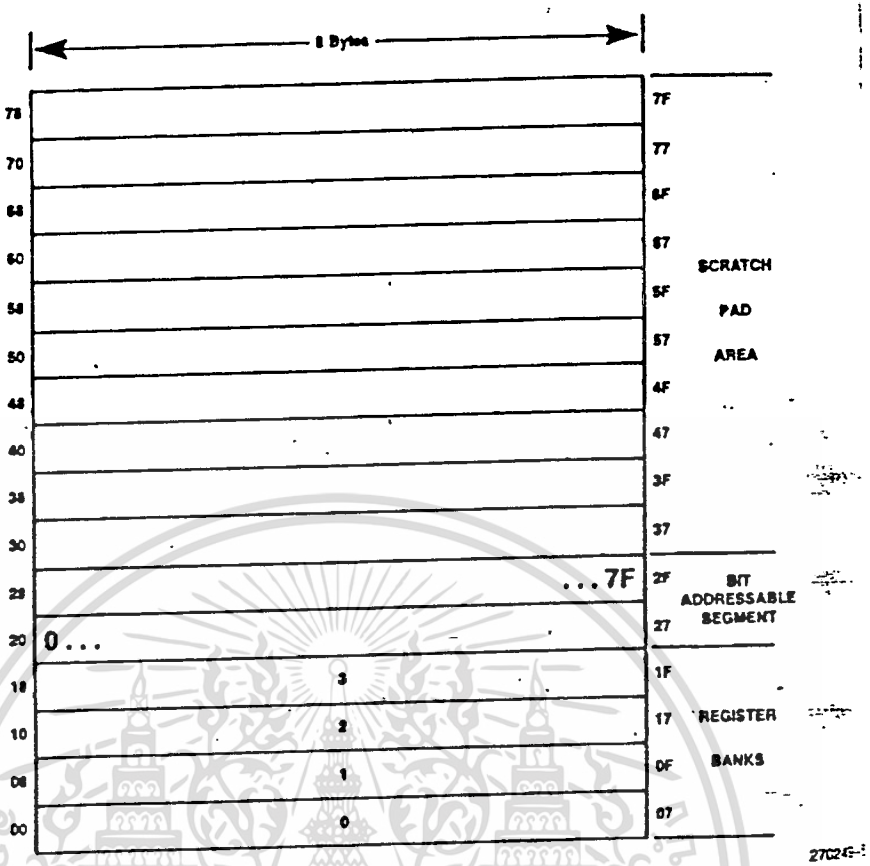
มีทั้งหมด 16 byte ตั้งแต่ แอดเดรส 20H-2FH หรือ 128 บิต โดยตัวบิตแอดเดรสได้ตั้งแต่ 0-7FH ได้โดยตรงเลย

การเข้าถึงแบบบิตอาจทำได้ 2 วิธี วิธีแรกอ้างถึงแอดเดรสบิต 0-7FH อีกวิธีหนึ่งอ้างแอดเดรส 20H-2FH, บิต 0-7 โดยอ้างดังนี้

20.0 -20.7, บิตที่ 8-FH ก็คือ 21.0-21.7 ใน 16 byte นี้ (หมายถึงบิตแอดเดรสเอเบิลแอเรีย) สามารถอ้างแบบไบต์ก็ได้

3. Scratch Pad Area

คือ บริเวณแอดเดรสไบต์ที่ 30H ถึง 7FH เป็นส่วนที่สามารถนำมาใช้งานได้



รูปที่ 16 128 bytes OF RAM direct and indirect addressable

Symbol	Name	Address
*ACC	Accumulator	0E0H
*B	B Register	0F0H
*PSW	Program Status Word	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer 2 Bytes	
DPL	Low Byte	82H
DPH	High Byte	83H
*P0	Port 0	80H
*P1	Port 1	90H
*P2	Port 2	0A0H
*P3	Port 3	0B0H
*IP	Interrupt Priority Control	0B8H
*IE	Interrupt Enable Control	0A8H
TMOD	Timer/Counter Mode Control	89H
*TCON	Timer/Counter Control	88H
*+T2CON	Timer/Counter 2 Control	0C8H
TH0	Timer/Counter 0 High Byte	8CH
TL0	Timer/Counter 0 Low Byte	8AH
TH1	Timer/Counter 1 High Byte	8DH
TL1	Timer/Counter 1 Low Byte	8BH
+TH2	Timer/Counter 2 High Byte	0CDH
+TL2	Timer/Counter 2 Low Byte	0CCH
+RCAP2H	T/C 2 Capture Reg. High Byte	0CBH
+RCAP2L	T/C 2 Capture Reg. Low Byte	0CAH
*SCON	Serial Control	98H
SBUF	Serial Data Buffer	99H
PCON	Power Control	87H

ตารางที่ 1 แสดงส่วนของ SFRs และ แอดเดรสทั้งหมดของ 8051

Register	Value in Binary
*ACC	00000000
*B	00000000
*PSW	00000000
SP	00000111
DPTR	
DPH	00000000
DPL	00000000
*P0	11111111
*P1	11111111
*P2	11111111
*P3	11111111
*IP	8051 XXX00000, 8052 XX000000
*IE	8051 0XX00000, 8052 0X000000
TMOD	00000000
*TCON	00000000
*+T2CON	00000000
TH0	00000000
TL0	00000000
TH1	00000000
TL1	00000000
+TH2	00000000
+TL2	00000000
+RCAP2H	00000000
+RCAP2L	00000000
*SCON	00000000
SBUF	Indeterminate
PCON	HMOS 0XXXXXXX CHMOS 0XXX0000

X = Undefined
* = Bit Addressable
+ = 8052 only

SFR หลังจาก Power-On หรือ hardware Reset

8 Bytes

F8								FF
F0	B							F7
E8								EF
E0	ACC							E7
D8								DF
D0	PSW							D7
C8	T2CON		RCAP2L	RCAP2H	TL2	TH2		CF
C0								C7
B8	IP							BF
B0	P3							B7
A8	IE							AF
A0	P2							A7
98	SCON	SBUF						9F
90	P1							97
88	TCON	TMOD	TL0	TL1	TH0	TH1		8F
80	P0	SP	DPL	DPH				PCON 87

↑ Bit Addressable

Figure 5

SFR MEMORY MAP

คำสั่ง ACALL address 11

Function : absolute Call การเรียกโปรแกรมย่อยแบบแอบโซลูต คำสั่ง ACALL เป็นคำสั่งที่ใช้เรียกโปรแกรมย่อย อย่างไม่มีเงื่อนไข (unecdition) เมื่อเราใช้คำสั่งนี้ PC จะเพิ่มค่า 2 ครั้ง จะได้ค่าคำสั่งถัดไปจากคำสั่ง ACALL แล้วเก็บค่าของ PC ที่เพิ่มค่าแล้ว ลงไปใน Stack โดยจะเก็บ byte แรก ก่อนแล้วค่อยใส่ PC ไบต์สูงตามไป นั่นคือค่าของสแต็กพอยเตอร์จะเพิ่ม 2 ครั้ง เช่นกัน การเรียกโปรแกรมย่อย จะอยู่ในบล็อค 2K ของ Program memory เท่านั้น เมื่อเริ่มนับจากคำสั่ง ที่ถัดมาจากคำสั่ง ACALL คำสั่งนี้ไม่มีผลต่อ flag

จำนวนไบต์ ที่เก็บคำสั่ง : 2 ไบต์

จำนวนรอบที่ใช้ 2 ไซเคิล (Machine cycle)

ลักษณะของการเก็บรหัสคำสั่งและแอดเดรส

ENCEDING : a10 a9 a8 1 0001 a7 a6 a5 a4 a3 a2 a1 a0

opcode

address ช่วง 2K block

ลำดับขั้นการดำเนินการ (operation) : ACALL

(PC) (PC) +2

(SP) (SP) +1

((SP)) (PC_{7-0})

(SP) (SP) +1

((SP)) (PC_{15-8})

(PC_{10-0}) page address

ตัวอย่างการใช้งาน :

เริ่มแรกใน SP มีค่าเท่ากับ 07H (SP) = 07H

ลาเบล "SURTN" ใน Program memory อยู่ที่ 0345H

หลังจากที่ execute คำสั่ง :

ADDRESS 0123H : ACALL SUBRTN

(next instruction)

ที่แอดเดรส 0123H จะเริ่มต้นเรียกโปรแกรมย่อย หลังจากที่เรียกโปรแกรมย่อยใน SP จะมีค่า 09H และที่แอดเดรส 08H และ 09H จะใส่ค่า 25H และ 01H (นั่นคือแอดเดรสของคำสั่งถัดไปต่อจาก SCALL SUBRTN) และ PC จะมีค่า 0345H ซึ่งเป็นแอดเดรสของโปรแกรมย่อยอยู่ หลังจากนั้นก็จะกระโดดไปใช้บริการของโปรแกรมย่อย

คำสั่ง ADD A, <sec-byte>

Function : ADD การบวกกันของ operand 2 จำนวน

4.2 ข้อเปรียบเทียบระหว่างคอมพิวเตอร์ และ PLC

PLCs เป็นคอมพิวเตอร์เฉพาะงานประเภทหนึ่ง PLCs แตกต่างกับไมโครคอมพิวเตอร์ดังนี้

1. ถูกออกแบบและสร้างขึ้นให้ทนต่อสภาพแวดล้อม ในโรงงานอุตสาหกรรม โดยเฉพาะ เช่น อุณหภูมิสูงและต่ำมาก ๆ ความชื้นสูง ระบบไฟฟ้าที่มีคลื่นรบกวน และไม่สม่ำเสมอ การกระทบกระแทกอย่างรุนแรง
2. ตั้งแต่ช่วงติดตั้ง จนถึงช่วงใช้งาน ทำให้การบำรุงรักษาทำได้ง่าย
3. ถูกพัฒนาให้มีความสามารถตัดสินใจสูงขึ้นเรื่อย ๆ ทำให้การใช้งานสะดวกขึ้น ขณะที่วิธีการใช้คอมพิวเตอร์ยุ่งยากและซับซ้อนขึ้นเรื่อย ๆ เพื่อให้สามารถปฏิบัติตามโปรแกรมของผู้ใช้ได้หลายโปรแกรม ขณะที่ PLCs สามารถปฏิบัติตามโปรแกรมของผู้ใช้ได้ เพียงโปรแกรมเดียว

4.3 รูปแบบภายในระบบ PLC ในโครงการนี้

ในระบบของ PLC จะแบ่งแต่ละส่วนของ PLC เป็น Module หลักๆ ได้แก่

- CPU + Memory Module
- input module
- output module

โดยที่แต่ละ module จะมี System bus เชื่อมต่อระหว่างกัน ทำให้ CPU + Memory

ติดต่อกับ input module และ output module ได้

ข้อดี การแบ่งระบบ PLC เป็น โมดูล(module) คือ

1. เป็นมาตรฐาน โดยจะยึดหลักการตามระบบบัสข้อมูลมาตรฐาน(STANDARD BUS) เช่นกรณีนี้ คือ STD BUS
2. สามารถตรวจซ่อมได้ง่าย โดย module ส่วนอื่น ๆ ไม่กระทบกระเทือน
3. สามารถจะขยายระบบได้ในอนาคต โดยที่ ไม่กระทบกระเทือนต่อระบบทั้งหมด หรือ ออกแบบระบบใหม่

รูปแบบการจัดเป็น Module ที่เชื่อมต่อด้วย STANDARD BUS แสดงดังรูปที่ 4.1 ก่อนที่ สายบัสต่าง ๆ ใน CPU + Mem module จะติดต่อทาง bus จะต้องมีการทำให้สัญญาณต่าง ๆ ที่จะส่งไป มีความสัมพันธ์เดียวกันกับมาตรฐานที่เราใช้ เพราะ Micro Processor Timing Diagram ยังไม่ตรงตามมาตรฐานของ STD จะต้องเพิ่มส่วนปรับ ให้เข้ากับมาตรฐานเสียก่อน โดยใช้buffer ช่วยขับอีกทีหนึ่ง

4.3.1 การออกแบบทาง HARDWARE

PLC (Program Logic Controller)

เนื่องจากโครงงานนี้ได้วางหลักการ (ทาง hardware) เอาไว้ดังนี้

PLC ประกอบไปด้วย 4 ส่วนหลัก

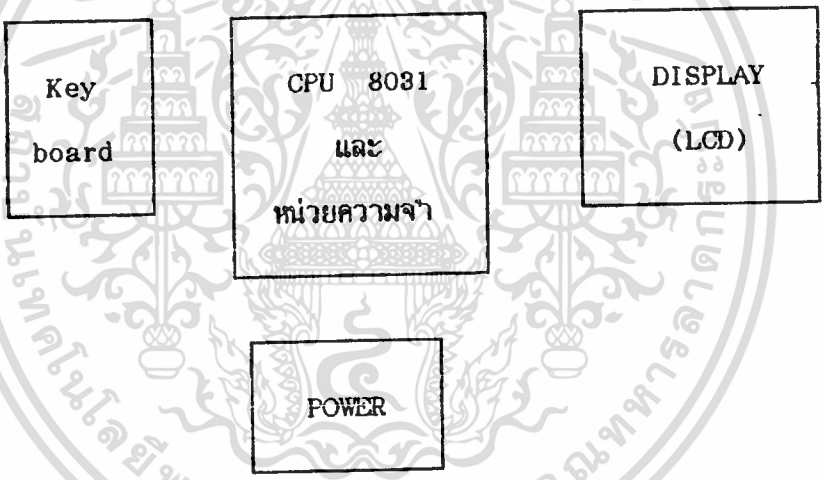
1. ส่วน CPU
2. ส่วน I/P
3. ส่วน O/P
4. Power supply

ซึ่งดูตามรูป block แล้ว 4 ส่วนหลักนี้มีความจำเป็นต่อ PLC ถ้าขาดส่วนใดส่วนหนึ่ง PLC ไม่สามารถทำงานให้บรรลุเป้าหมายได้

- ส่วน CPU ประกอบด้วย ตัวประมวลผลกลาง (และหน่วยความจำ) ควบคุมการทำงานของทั้งระบบ ตั้งแต่ตรวจสอบข้อผิดพลาดของระบบ, รับค่า Input มา ประมวลผล ตามที่โปรแกรมรูปแบบการทำงานเอาไว้แล้ว เมื่อประมวลผลก็ส่งผลการคำนวนออกไปควบคุมอุปกรณ์รอบข้างอีกทีหนึ่ง เมื่อส่งออกไปก็ต้องมีการตอบ

สนองค่า input กลับมาอีกทีนำมาประมวลผลแล้วส่งออกไปยัง Output อีก เป็นวัฏจักร (cycle) ซึ่งการนำค่า input มาประมวลผลแล้วส่งผลการควบคุม ไปออก Output 1 รอบ เราเรียกว่า การ 1 รอบกำร Scan ยิ่งเวลาที่ Scan น้อยเท่าใด ประสิทธิภาพของ PLC ก็ยิ่งสูง ราคาของ PLC ที่มีขายอยู่นี้นอยู่กับเวลาของการ Scan ด้วย

- ส่วนของ I/P (Input) นอกจากจะเป็น port input ที่รับค่าจาก อุปกรณ์ที่เราต้องการควบคุมแล้ว เรายังรวมส่วนของการรับค่าจาก Keyboard อีกด้วย ส่วนของการรับค่าจาก Keyboard นี้ได้รับการออกแบบให้มี ไมโครโปรเซสเซอร์ตระกูล MO51 เบอร์ 8031 (Romless Version ของ '51) เป็นตัวควบคุมการสแกนคีย์บอร์ดต่างหาก (เพื่อเพิ่มความเร็วในการรับข้อมูล) ส่วน Scan Keyboard ส่วนประกอบดังนี้

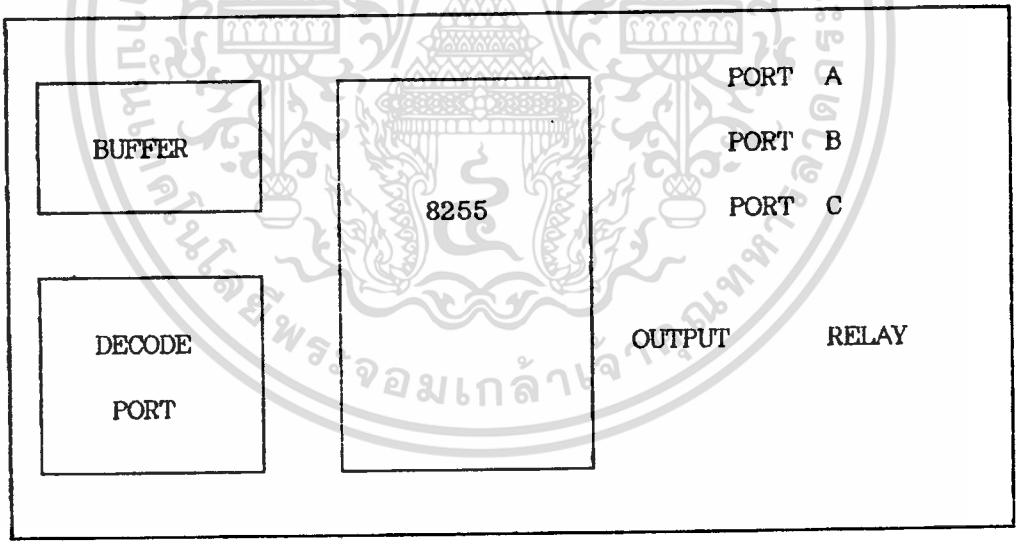


รูปที่ 17 แสดงส่วนของ Scan Keyboard

จะสังเกตุเห็นว่า ส่วนของ Scan Keyboard ส่วนหน่วยแสดงผลด้วย LCD แต่ไม่ใช่ส่วน O/P ของ PLC แต่ Scan Keyboard จะเป็นส่วนหนึ่งของ Input ของ PLC เท่านั้น การติดต่อของส่วน Scan Keyboard หลังจากที่เราโปรแกรมการทำงานแล้ว แปลง Code การทำงานเสร็จแล้วจะถูกเก็บใช้ที่หน่วยความจำชั่วคราวของ Scan Keyboard อยู่ก่อน หลังจากนั้นค่อยส่ง Code โปรแกรมการทำงานผ่านทาง Serial ของ 8031 ไปยังบอร์ด 8251 ของหน่วย I/P ของ PLC อีกทีหนึ่ง

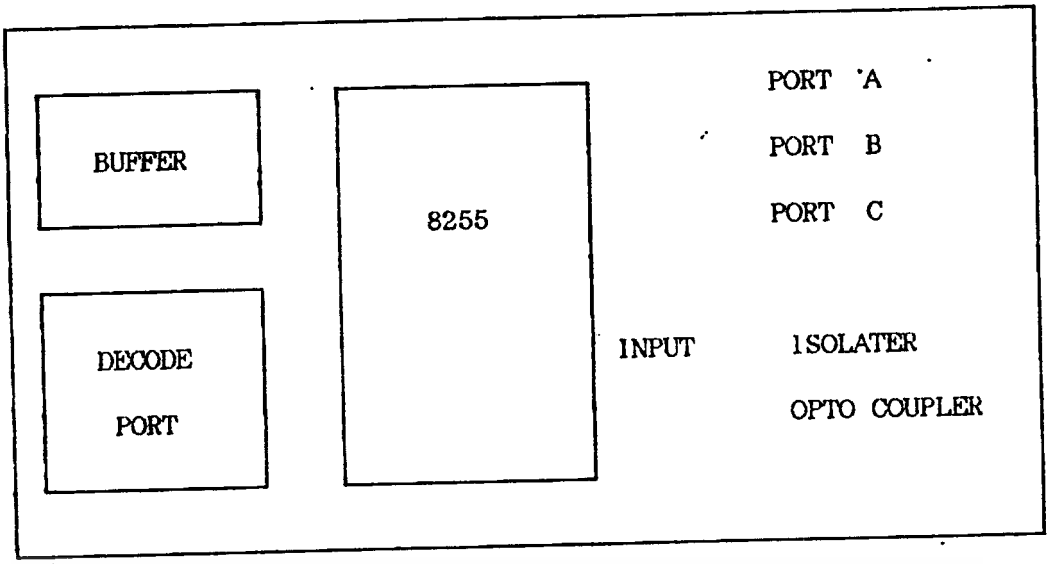
เนื่องจากการออกแบบระบบควบคุมใด ๆ ในปัจจุบันนอกจากจะต้องการอุปกรณ์ที่มีความสามารถมีความยืดหยุ่นดีแล้ว ยังมีความต้องการที่จะตรวจสอบระบบซ่อมแซมระบบที่มีความสะดวกไม่ยุ่งยากอีกด้วย ทำให้เกิดความคิดที่ว่าถ้าระบบควบคุมทั้งหมดอยู่บนบอร์ดเดียวกัน ถ้าควบคุมงานที่สำคัญมาก ถ้าอุปกรณ์ตัวใดตัวหนึ่งเสียแล้ว ระบบไม่สามารถทำงานต่อไปได้ ก็ต้องเอามาซ่อมทั้ง Board ใหม่ ซึ่งนอกจากจะตรวจสอบยากแล้วยังทำให้งานต้องชะงักไปอีกด้วย ทำให้เกิดความเสียหายในเชิงธุรกิจอีกด้วย จึงได้มีการเสนอการออกแบบ Hardware ให้มีลักษณะแยกเป็นชุด ๆ ไป หรือเรียกว่าเป็น Module (โมดูล) ส่วนที่ทำหน้าที่ส่งผลการคำนวน เราก็เรียกว่า "input Module" ส่วนที่ทำหน้าที่ส่งผลการคำนวน เราก็เรียกว่า "Output Module" ส่วนที่ทำหน้าที่ประมวลผลและจัดการเกี่ยวกับหน่วยความจำต่าง ๆ ก็เรียกว่า CPU Memory Module เมื่อเรามีส่วนที่ให้กำลังงาน (Power) แก่ระบบทั้งหมดเราเรียกว่า "Power Supply Module"

ซึ่งโมดูลต่าง ๆ ที่กล่าวมานี้ เมื่อมีการแยกโมดูลออกจากกัน โดยเราสามารถนำไปตรวจสอบแยกกันได้ เราก็ต้องมีการนำโมดูลต่าง ๆ มาต่อเชื่อมกันได้ด้วยเช่นกันซึ่งการต่อเชื่อมกันเรา



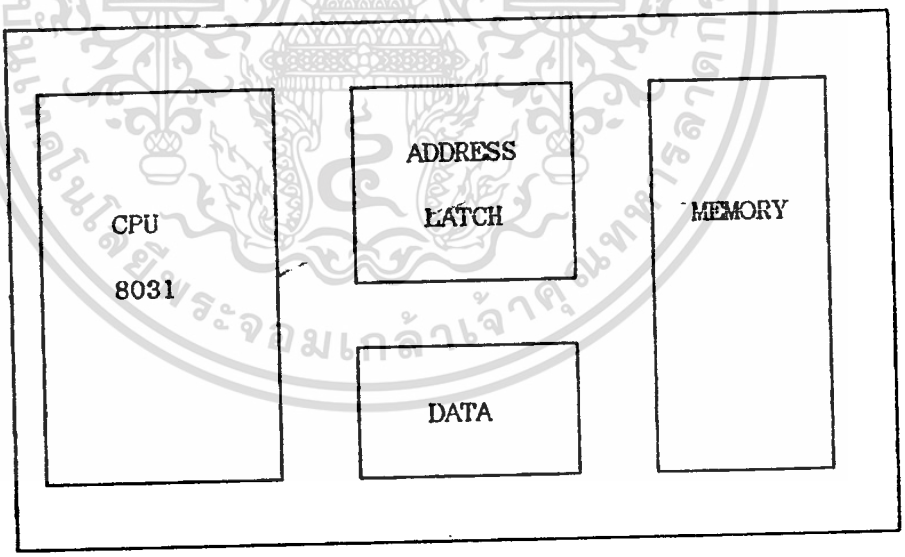
SYSTEM BUS (STD)

หน่วยเอาต์พุต (OUTPUT MODULE)



SYSTEM BUS (STD)

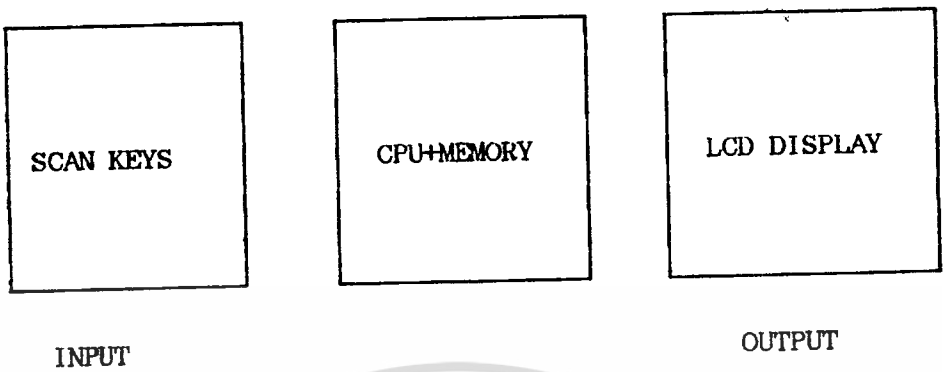
หน่วยอินพุต (INPUT MODULE)



SYSTEM BUS (STD)

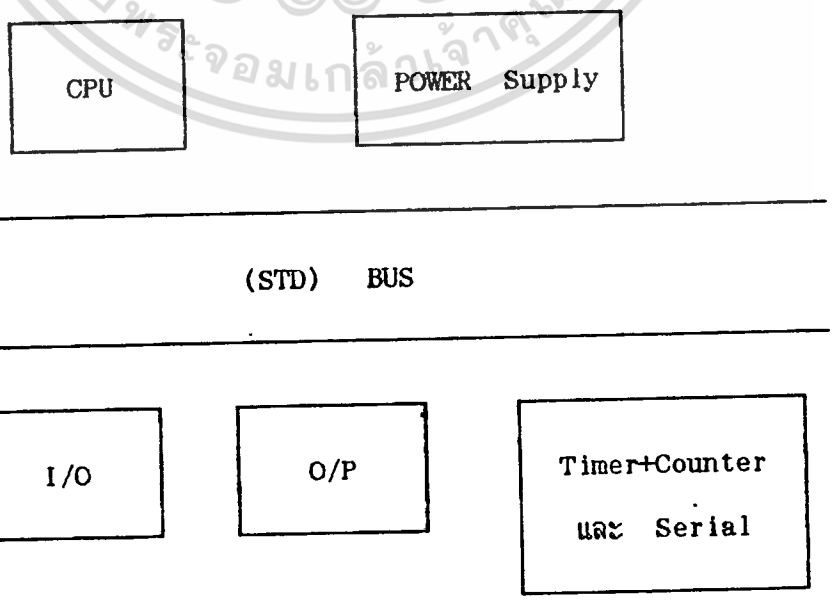
หน่วย CPU + MEMORY MODULE

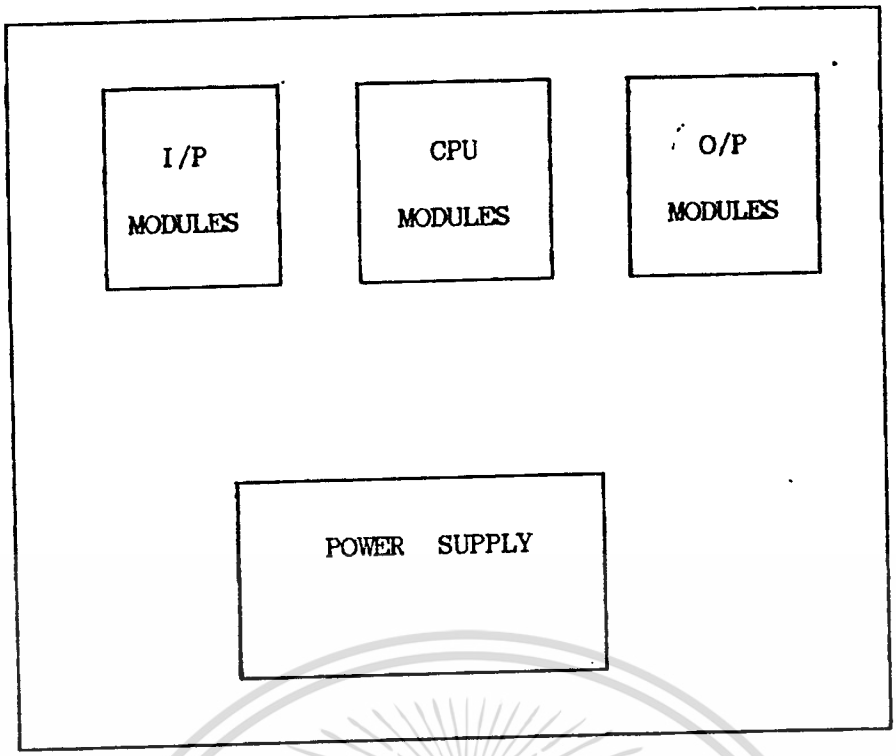
ส่วนสแกนคีย์บอร์ดและหน่วยแสดงผลทาง LCD เป็นการทำงานหนึ่งที่แยกออก จากส่วนหลักของ plc



แสดงส่วนต่าง ๆ ของส่วนรับคีบอร์ด และหน่วยแสดงผลด้วยแอลซีดี

การสื่อสารภายใน PLCs จะติดต่อโดยใช้ bus ร่วมกัน กรณีที่มีการติดต่อกับส่วนนอก ถ้าเป็นอุปกรณ์ควบคุมจะติดต่อผ่านส่วน I/O module และกรณีติดต่อทาง RS-232 จะติดต่อผ่านทาง Serial Module (กรณีรวมเข้ากับ Timer-Counter)





CPU + MEMORY MODULE

จะอาศัย ระบบบัสเข้าช่วย ซึ่งระบบบัส (BUS SYSTEM) ก็คือระบบทางเดินของสัญญาณต่าง ๆ ร่วมกันของแต่ละโมดูล นั่นเอง ในระบบบัสต่าง ๆ จะประกอบด้วยสัญญาณ 4 ชนิดคือ

1. Control bus
2. Data bus
3. address bus
4. Power bus

ในปัจจุบันมีระบบบัสที่เป็นมาตรฐานอยู่มากมาย ในวงการอุตสาหกรรม ที่เป็นที่ยอมรับทั่วไป เช่น PC bus, AT bus, VME bus, S-100 bus, STD bus เป็นต้น ซึ่งในโครงงานนี้เราจะใช้ระบบบัสมาตรฐาน ที่มีชื่อว่า STD บัส

<u>หน้าที่หลักของส่วน Scan Keyboard (และ LCD DISPLAY)</u>

- เป็นส่วนที่ติดต่อกับผู้ใช้ ก่อนที่จะทำการติดต่อกับส่วนหลักของ PLC อีกทีหนึ่ง
- เป็นหน่วยป้อนโปรแกรม และแก้ไขโปรแกรมให้ถูกต้องก่อนและเก็บ CODE โปรแกรมเอาไว้ก่อนที่ จะส่งไป RUN บน PLC จริง ๆ
- เป็นหน่วยแสดงผลได้เมื่อมีข้อความต้องการที่ต้องการเตือน

หน้าที่หลักของ PLC หลัก

- นำโปรแกรม CODE ของโปรแกรมที่ USER ป้อนเอาไว้เรียบร้อยแล้วเข้ามา RUN โดยเอา CODE มาถอดความทีละคำสั่ง และกระทำตามคำสั่ง จนกว่าจะหมดแล้วก็ Scan รอบคำสั่งต่อไปเรื่อย ๆ

SCAN KEYBOARD

การจัดการ Memory ของส่วน Scan Keyboard (8031)

ROM 0000H-1FFFH (8K) ส่วนของ program memory

RAM 2000H-3FFFH (8K) ส่วนของ External data memory

8255 4000H-4003H ส่วนติดต่อกับ Ship support 8255 ซึ่งทำหน้าที่การ Scan Keyboard

LCD 6000H-6003H ส่วนที่ติดต่อกับหน่วยแสดงผลแบบ LCD

Keyboard เป็นแบบ 40 key Membrane Switch Kit

เนื่องจากเป็นคีย์บอร์ดขนาด 40 Key แต่จำนวนสายสัญญาณที่ใช้มีเพียง -13 สาย (8x5 = 40 key) การ Scan เราควบคุมทาง Software ไม่มีการอินเทอร์รัพท์) ฉนั้น คีย์บอร์ดที่แสดงที่แสดงให้เห็นภายนอกเป็นแบบ 4x10 แต่ระบบสัญญาณภายในเป็นแบบ 8x5 การ map ค่าของสัญญาณต่าง ๆ จึงต้องเอาค่าที่ได้จากการ Scan ไป map ให้เข้ากับค่าที่เรามองเห็นบนคีย์บอร์ดเสียก่อน รายละเอียดดูที่ภาคผนวก

หน่วยแสดงผล LCD

ที่ใช้ในโครงการนี้ เป็น DOT MATRIX LCD MODULE ซึ่งโดยทั่วไปแล้ว DOT MATRIX LCD MODULE ยังแบ่งออกเป็น 3 ชนิด

1. Character LCD module
2. Graphic LCD module
3. Segment Displey type LCD module

ซึ่ง DOT Matrix ที่เราใช้เป็นแบบที่ 1 คือ character LCD

ภายใน block ของ LCD จะมีส่วนประกอบใหญ่ 3 ส่วนประกอบคือ

1. DOT MATRIX LCD เป็นตัวแสดงผลให้เรามองเห็นในลักษณะการปิดและเปิดตัวเองกับแสง

2. DRIVER เป็นตัวรับสัญญาณจากตัวควบคุมมาขับ LCD ตัว Driver ที่นิยมใช้

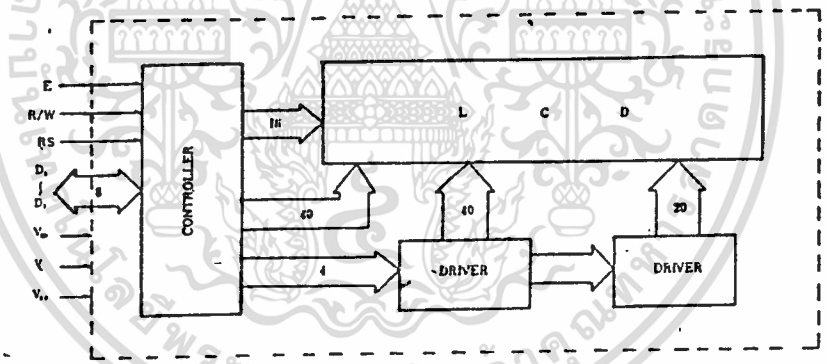
เช่นเบอร์ HD44100H, MSM5259

3. Controller เป็นตัวรับข้อมูลจากอุปกรณ์ภายนอกมาและจัดการควบคุม LCD MODULE ให้ทำงานแสดงผลต่าง ๆ เช่น การลบจอภาพ, แสดงตัวอักษร ไอซีที่นิยมใช้คือ HD44780 ซึ่งใช้ในแบบ CHARACTER LCD MODULE IC HD61830 ใช้ในแบบ Graphic LCD MODULE

จากการศึกษาการควบคุมการทำงานของ LCD ทำได้ไม่ยุ่งยากนัก เนื่องจากภายใน LCD MODULE จะมีส่วน Driver และ Controller ควบคุมการทำงานของ LCD อยู่แล้ว เพียงแต่เราทำความเข้าใจกับส่วนของ LCD Controller เท่านั้นก็เพียงพอแล้ว

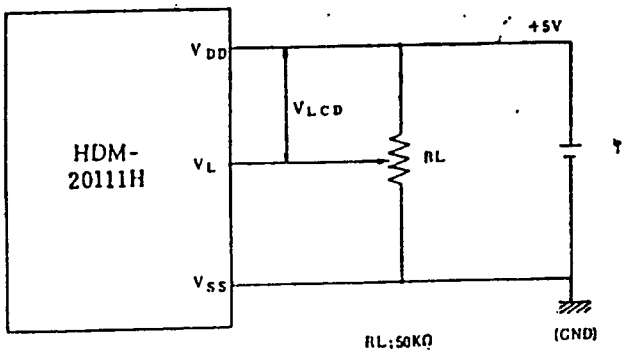
ตัวอย่าง IC Controller LCD เบอร์ HD44780 เป็นตัวควบคุมให้แสดงผลในรูปตัวอักษรหรือสัญลักษณ์ต่าง ๆ ตัวมันเองสามารถต่อใช้งานแบบ 4 bit หรือ 8 bit ก็ได้ โดยถ้าเราต่อแบบ 4 bit จะต่อใช้งานที่ DB7-DB4 เท่านั้น โดยข้อมูลที่ส่งออกไปครั้งแรกนั้น HD44780 ถือเป็นข้อมูล 4 bit บน และข้อมูลที่ส่งต่อมาเป็นข้อมูล 4 bit ล่าง

block Diagram ของ LCD แสดงดังรูปที่ 18 (รายละเอียดย่อยดูที่ภาคผนวก)



รูปที่ 18 แสดง BLOCK DIAGRAM MODULE ของ LCD

การเขียนโปรแกรมควบคุม กรณีทั้งเขียนข้อมูลให้แสดงบน LCD ควรจะมีการ delay เวลาไม่ต่ำกว่า 20 ms ในแต่ละช่วงคำสั่งที่ติดต่อกัน เนื่องจากว่าการแสดงผลบนจอ LCD ทำงานโดยใช้ขั้วไฟฟ้าสลับทำให้เกิดการบิดตัวของผลึกของเหลว ทำให้การหักเหของแสงเปลี่ยนไปจากใสเป็นมืด เป็นรูปตัวอักษรสีดำได้ ทำให้เราเห็นตัวอักษรได้บนจอ เนื่องจากความถี่ที่ใช้กับไมโครโปรเซสเซอร์ มีความถี่สูงมากเกินไป เมื่อมีคำสั่งให้แสดงผลบนจอ LCD ตัว LCD จะรับคำสั่งและไปแสดงผลไม่ทัน



รูปที่ 19 แสดงการต่อกับ Power Supply

PIN CONNECTIONS

Pin No.	Symbol	Level	Function	
1	VSS	–	0 V	Power supply
2	VDD	–	5 V	
3	VL	–	–	
4	RS	H/L	H: Data input L: Instruction data input	
5	R/W	H/L	H: Data read L: Data write	
6	E	H,H→L	Enable signal	
7	D0	H/L	Data bus line*	
8	D1	H/L		
9	D2	H/L		
10	D3	H/L		
11	D4	H/L		
12	D5	H/L		
13	D6	H/L		
14	D7	H/L		

* In case of 4 bits instruction, data is transferred by twice using only 4 buses of D4-D7, and D0-D3 are not used; first operation is higher order 4 bits and second is lower 4 bits of 8 bits, but in case of 8 bits instruction, data is transferred by data bus of D0-D7.

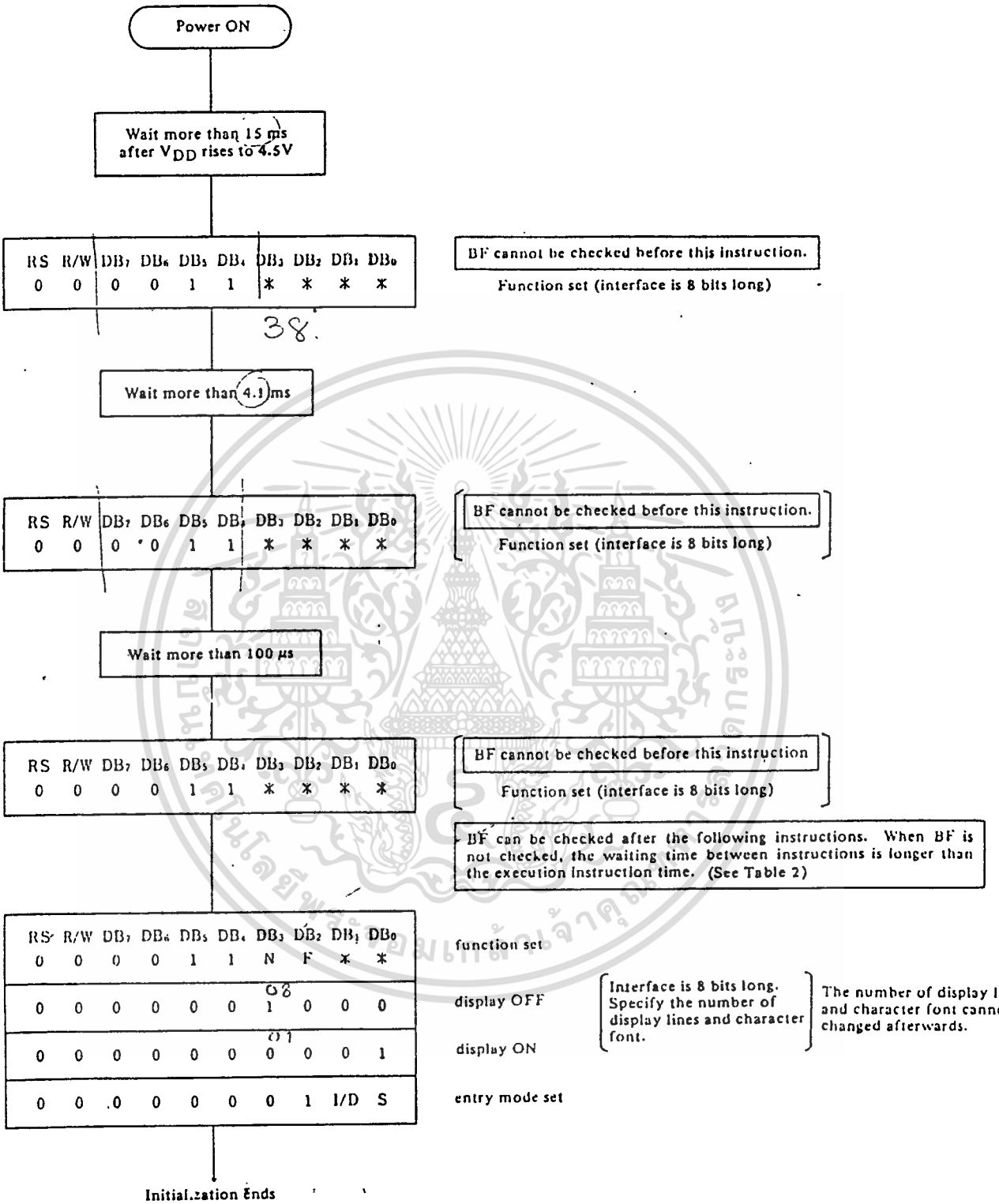
รูปที่ 20 แสดงการต่อสัญญาณของขาสัญญาณใน LCD

ฉนั้นทุกครั้งที่มีการเขียนข้อมูลไปแสดงที่จอ LCD ทุกครั้ง ติดต่อกันหลายคำสั่ง ควรมี การ delay time โดยเรียกใช้ Subroutine delay-time

(รายละเอียดคำสั่งดูได้จากภาคผนวก)

ขาสัญญาณต่าง ๆ ในการต่อใช้งาน (HD 44780)

1. RS (REGISTER SELECTION) จะเป็นขาเลือก Register ภายใน ซึ่งมีอยู่ 2 ตัว คือ instruction Register (IR) และ DATA Register (DR) โดยถ้าเป็น "1" จะเป็นการเลือก DATA และถ้าเป็น "0" จะเป็นการเลือก Instruction



รูปที่ 21 แสดงการเริ่มต้นของ LCD

2. R/W (Read/Write) เป็นตัวเลือกว่าจะเขียนหรือจะอ่านข้อมูล จากตัว IC

ถ้าเป็น "1" จะเป็นการอ่านข้อมูล

"0" จะเป็นการเขียนข้อมูล

3. E (Enable Signal) เป็นขากำหนดสภาพการรับ, หรืออ่านข้อมูล ความสัมพันธ์ระหว่างสัญญาณ RS R/W และ E ดังตาราง

4. DB0-DB7 เป็นบัสรับและส่งข้อมูล

5. VDD ไฟเลี้ยววงจร

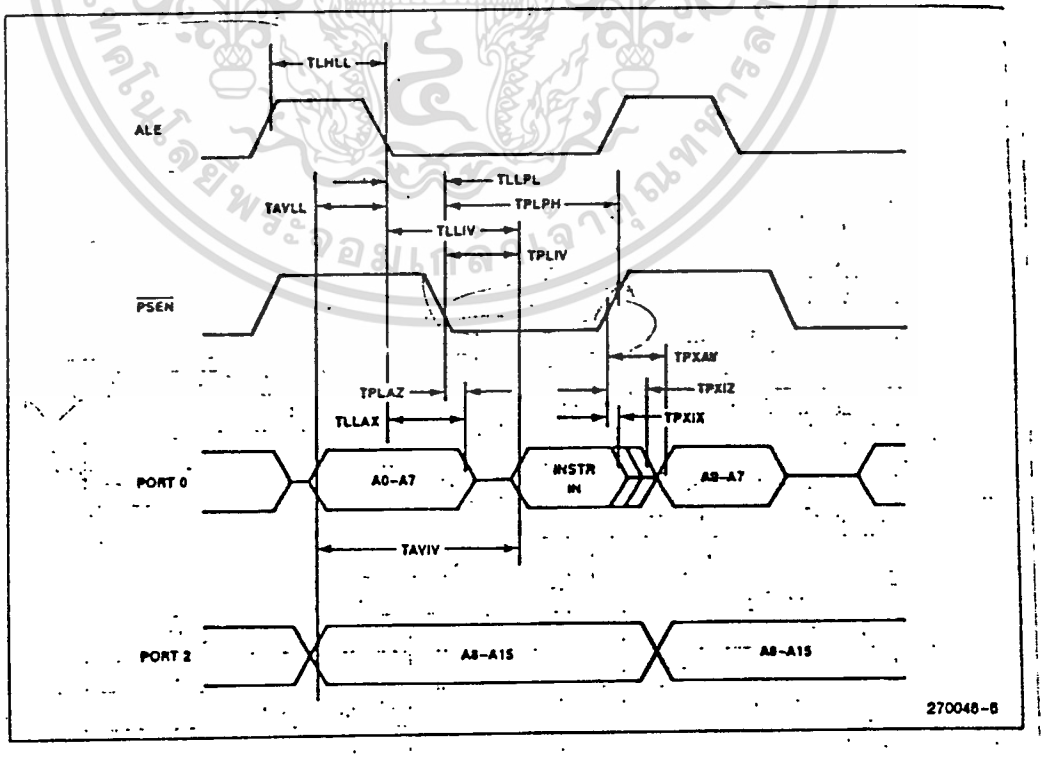
6. V_{ss} ขากราวด์

7. V_l เป็นขารับระดับ Voltage สามารถปรับให้มืดหรือสว่างได้

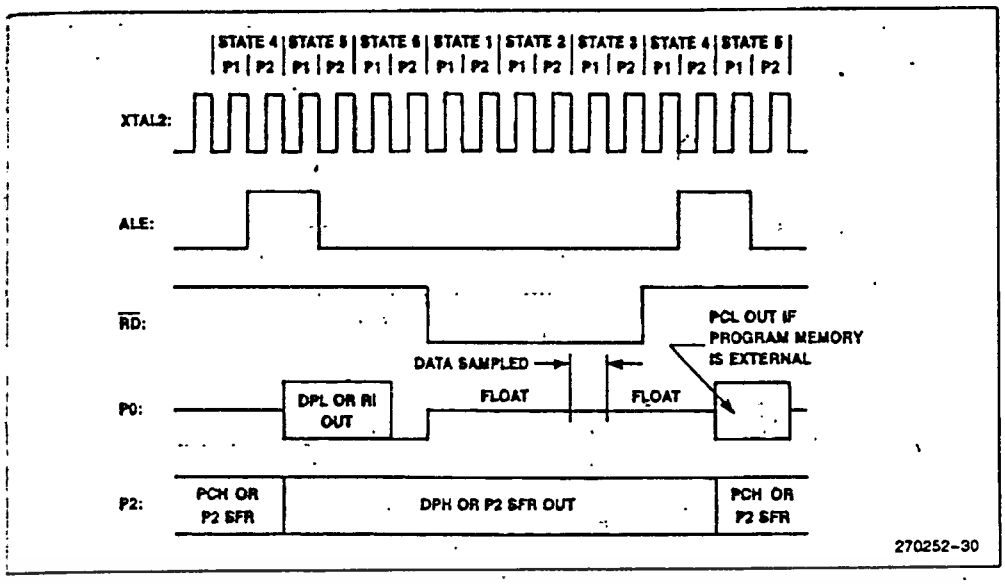
ฮาร์ดแวร์ของส่วน Keyboard

ส่วนสแกนคีย์บอร์ดและหน่วยแสดงผลทาง LCD เป็นการทำงานหนึ่งที่แยกออกจากส่วนหลักของ PLC (ส่วนประกอบหลักใช้ไมโครโปรเซสเซอร์ 8088)

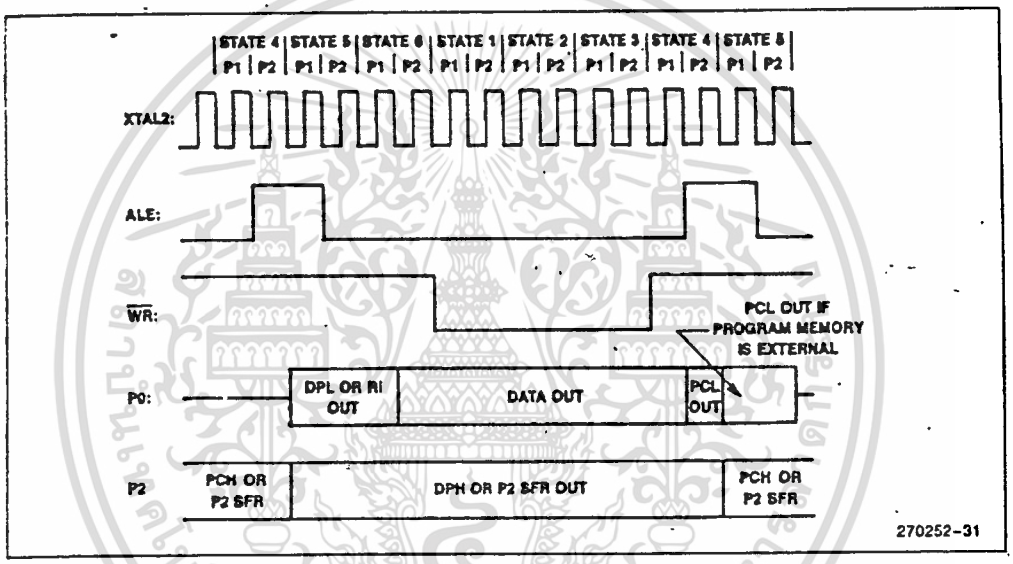
ส่วนสแกนคีย์บอร์ดและหน่วยแสดงผลทาง LCD ใช้ไมโครโปรเซสเซอร์เบอร์ 8031 ซึ่งอยู่ในตระกูล 51 (mcs-51) ของบริษัท intel มีคุณสมบัติเหมือนกับ 8051 เว้นแต่ว่าไม่มี EPROM ภายในตัวเหมือนกับ 8051 (8031 เป็น romless version ของเบอร์ 8051 ซึ่งอยู่ในตระกูล 51 เช่นเดียวกับเบอร์ 8031) เมื่อต้องการนำมาใช้งานก็ต้องต่อกับ EPROM ภายนอก



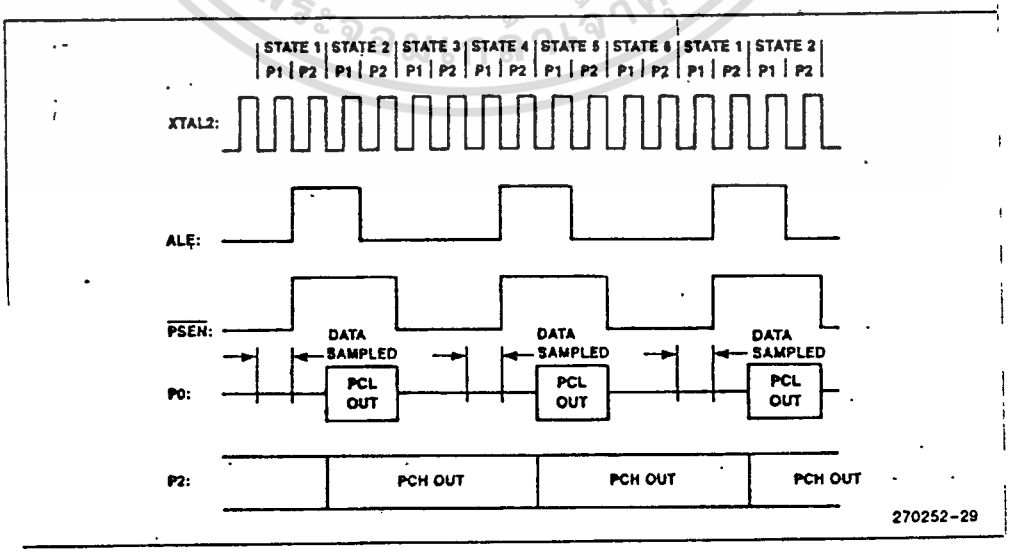
รูปที่ 25 EXTERNAL PROGRAM MEMORY READ CYCLE



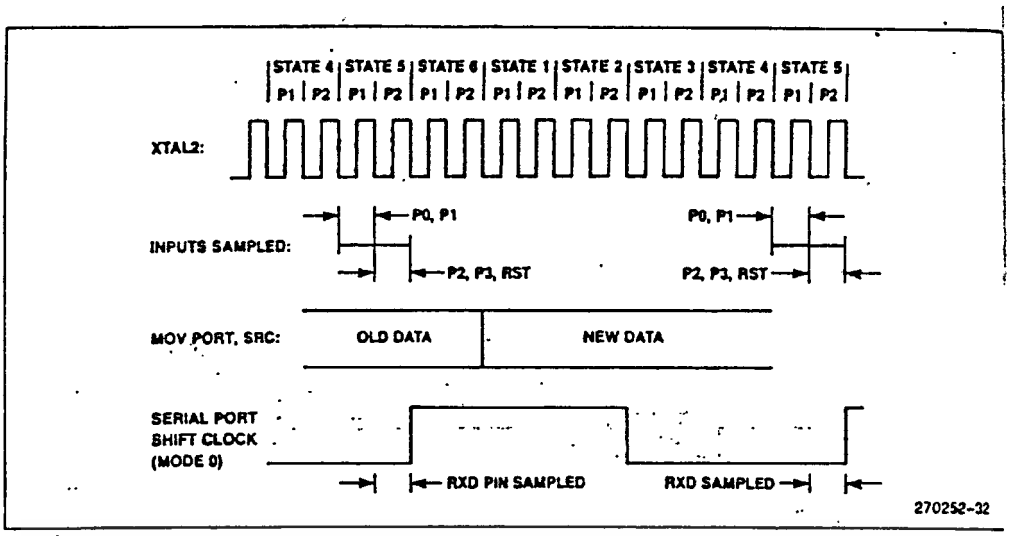
รูปที่ 26 External Data Memory Read Cycle



รูปที่ 27 External Data Memory Write Cycle



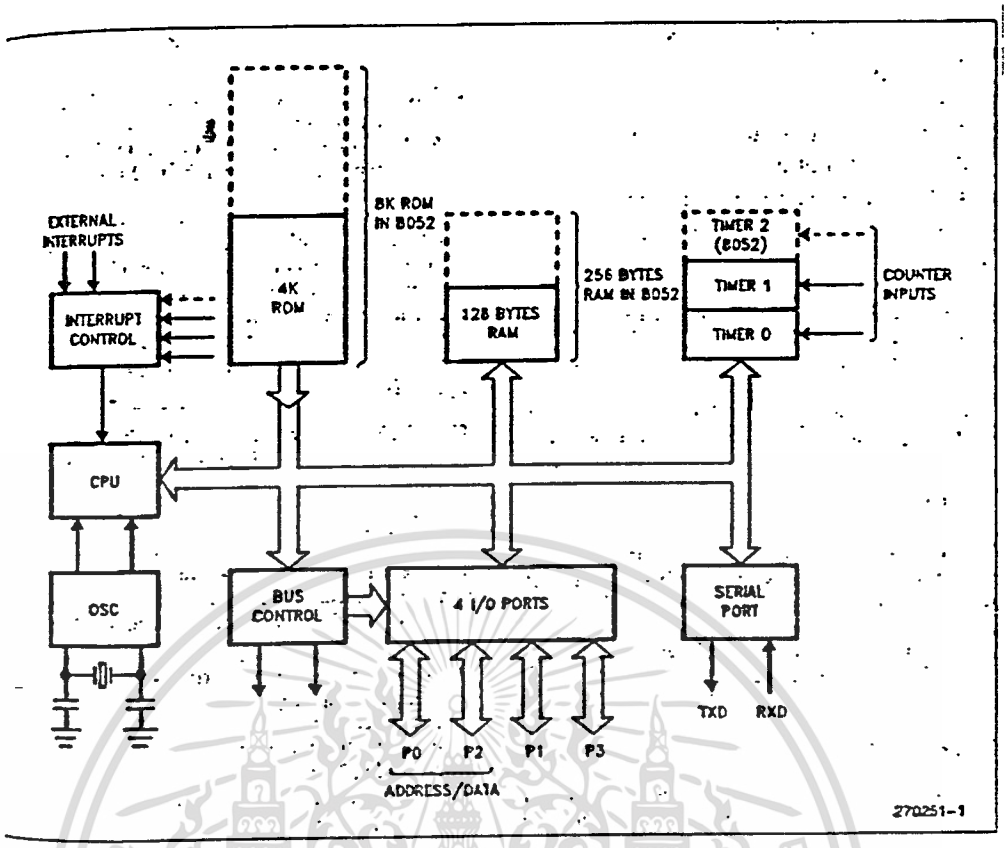
รูปที่ 28 External Data Memory Fetches



รูปที่ 29 Port Operation

ลักษณะทั่วไปของ mcs-51 (feature of mcs 51)

- 8 bit cpu optimized for control application
- extensive boolean processing (single bit logic) capability
- 32 bidirectional and individually addressable I/O lines (four bit port)
- full duplex UART
- 5 source interrupt structure with 2 priority levels
- 64 Kbyte maximum on-board program size
- 256 byte maximum on-board ram size(128 for 8031)
- 64 Kbyte maximum off-board ram size
- 2 timers/counters ขนาด 16 bit
- bit-addressable ram
- 111 instructions
- มีวงจร oscillator ภายในตัว



รูปที่ 30 Block Diagram ของ 8031

STD BUS

STD (Simple To Designed) BUS เป็นที่รู้จักครั้งแรกในปี 1978 จากความร่วมมือของ 2 บริษัทอเมริกัน คือ PRO-LOG และ MOSTEK ครั้งแรกเป็นระบบบัสที่รองรับให้เป็นบัสมาตรฐานของไมโครโปรเซสเซอร์ ขนาด 8 บิต เท่านั้น อันได้แก่ 8085A, Z-80, 6800 เป็นต้น

หลังจากนั้นไม่นาน STD BUS ได้ถูกเผยแพร่สู่สาธารณะ ทำให้มีผู้ผลิตอุปกรณ์ควบคุมรองรับมาตรฐาน STD บัส ในปลายปี 1985 ได้มีการรวมกลุ่มผู้ผลิตอุปกรณ์รองรับมาตรฐาน STD BUS เรียกว่า STD manufacturers

- Group (STDMG)

ซึ่งตามมาตรฐานของ STD BUS ได้มีการกำหนดขนาดของ CARD ที่แน่นอน ตลอดจนส่วนประกอบอื่น ๆ ที่จำเป็น และที่ขาดไม่ได้คือ การจัดตำแหน่งของสัญญาณต่าง ๆ

และ timing Diagram ให้ตรวจตามมาตรฐานอีกด้วย

ในระบบบัสสากล (IEEE : the Institute of Electrical and Electronic Enginerrs) มีมาตรฐานที่ใช้กันหลายแบบ ดังนี้

- VME (Single width Eurocard)
- Multibus (IEEE p796)
- IBM PC (Maximum size)
- STD (IEEE P961)
- VME (Double-sidth Eurocard)
- S-100 (ieee S696)

ตารางการเปรียบ มาตรฐาน STD BUS กับมาตรฐานบัสอื่น ๆ

	ขนาดแผ่นการ์ด	คอนเน็กเตอร์	จำนวนขั้วสัมผัส
PC & PC/AT	13.2" x 4"	EDGE	62/96
STD	6.5" x 4.5"	EDGE	56
VME	9.187" x 6.3" (6.3" x 3.94")	DIN	96
MULTIBUS	12" x 6.75"	EDGE	86

จะเห็นว่า STD BUS มีจำนวน 56 สัญญาณ เท่านั้น ที่ใช้ติดต่อร่วมกัน เหมาะสำหรับระบบขนาดกลางและขนาดเล็ก แต่ข้อได้เปรียบ ก็คือ เป็นระบบที่ไม่ซับซ้อน ทำให้ประสิทธิภาพในการทำงานสูง ซึ่งตามมาตรฐาน STD BUS CARD ที่ใช้จะมีการ์ดแร็ก (Card Rack) เป็นส่วนจับยึดการ์ด เอาไว้ ไม่ให้สั่นสะเทือน หรือบิดงอ ซึ่งทนต่อสภาวะของการสั่นสะเทือนและกันการกระแทกได้ดี นอกจากนี้ ยังมีส่วนของ ejector เป็นตัวดีดเพื่อถอดการ์ดออกจาก card Rack อย่างง่ายดายอีกด้วย

ในเวลาต่อมา ได้มีการนำเอาไมโครโปรเซสเซอร์ 16 bit มาใช้กับ STD บัส ได้แก่ 6809,68008, 8088 และ 80188 ซึ่งภายนอก data bus เป็นแบบ 8 บิต แต่สถาปัตยกรรมภายในเป็นแบบ 16 บิต ซึ่งได้รับความนิยมมากเนื่องจากการประมวลผล ภายในจะเร็ว

ขึ้น ซึ่งบริษัทWIN SYSTEMS และกลุ่มผู้ผลิต STD BUS ได้เสนอ STD BUS สำหรับไมโครโปรเซสเซอร์ ขนาด16 บิต ขึ้นมาและเป็นที่ยอมรับ จากสมาคม STDMG โดยไม่ต้องเปลี่ยนแปลงแก้ไขส่วนโครงสร้างของการ์ดเลย

STD BUS กับเทคโนโลยีของอุปกรณ์ซีมอส

ในช่วงที่อุปกรณ์อิเล็กทรอนิกส์ ที่เป็นแบบ CMOS กำลังได้รับความนิยมเป็นช่วงที่นำ STD มาใช้ ได้มีการใช้อุปกรณ์ CMOS กับมาตรฐานนี้

ข้อดีของ STD BUS คือ

- สามารถป้องกันสัญญาณรบกวนได้ดี
- ทนอุณหภูมิการใช้งานได้สูง ภายใต้อุณหภูมิช่วง 0-65 C ทำให้ไม่จำเป็นต้องติดพัดลมระบายความร้อน และใช้ในโรงงานที่มีสัญญาณรบกวนได้ดี

ข้อมูลทาง technic ของ BUS (STD BUS)

กลุ่มสัญญาณเราแบ่งออกเป็น 4 กลุ่มคือ

1. กลุ่มบัสข้อมูล (data bus)
2. กลุ่มแอดเดรสบัส (address bus)
3. กลุ่มบัสควบคุม (Control bus)
4. กลุ่มไฟเลี้ยงระบบ (Power Supply)

ดูได้จากตารางแสดงการจัดตำแหน่งของสัญญาณของ STD BUS

<u>กรณีใช้ CPU 8088</u>

เราจะเปลี่ยนสัญญาณ STATUS 1 เป็น DT/R* (PIN 39)

MCSYNC* "-" ALE* (PIN 38)

เป็นสัญญาณที่ใช้ demultiplex แอดเดรส (8 byte ต่ำออกมา)

STATUS0* เป็น SSO* (PIN40) นอกนั้นคงเดิม

ค่า Power Supply ที่ยอมรับได้		
	+ 5V	0.25V
	- 5V	0.25V
	+12V	0.5V
	-12V	0.5V

เมื่อเทียบกับ Ground

ในระบบของโครงงานนี้ ได้พัฒนาใช้กับระบบ STD BUS มีองค์ประกอบดังนี้

1. มี 56 conductors :

- 22 ขาควบคุม (control lines) ทำหน้าที่ control data ระหว่าง CPU กับ MEMORY หรือ I/O registors
- 6 ขาไฟเลี้ยง (logic power lines) : 5V, 12V
- 8 ขาข้อมูล (data lines) : แบบ สองทิศทาง(bidirections) โดยมี การสื่อสารแบบซิงโครนัส (synchronous)
- 16 ขาแอดเดรส (address lines)
- 4 ขาช่วยไฟเลี้ยง (auxiliary power lines)

2. มีหน่วยความจำที่ส่งแบบโดยตรง (DMA : direct memory access) ถึง 64K และ แบบหน่วยความจำธรรมดา ขยายได้ถึง 1 Mbyte

3. ใช้กับชิพไมโครโปรเซสเซอร์ ที่มีขาข้อมูล ภายนอกแบบ 8 ขา โดยอาจจะมีการจัดการข้อมูลภายในแบบ 8, 16 หรือ 32 บิทก็ได้ เช่น microprocessor ของ INTEL 8088, MOTOROLA 68008, TI 9995 เป็นต้น

ในเชิงอุตสาหกรรมมีการใช้ STD BUS กันมาก โดยสามารถประยุกต์ใช้ในระบบคอมพิวเตอร์ได้อีกด้วย โดยจะมีองค์ประกอบของอุปกรณ์ แบบ CMOS เพราะมีขอบเขตของอุณหภูมิกว้าง และต้านทานต่อ สัญญาณรบกวน (noise) ได้ดี ทั้งยังมีขนาดเล็กกระทัดรัด ซึ่งง่ายต่อการลดอุณหภูมิ OPTO COUPLER

ข้อดีของการใช้ Photo coupler

1. i/p, o/p แยกจากกันทางไฟฟ้า
2. ใช้แหล่งจ่ายไฟคนละแหล่ง
3. GND แยกจากกัน ไม่เกิด Ground loop
4. มีความปลอดภัยต่อระบบ
5. กำจัดสัญญาณรบกวนได้ดี

ข้อพิจารณา ในการออกแบบวงจรพิมพ์

1. เส้นวงจรแต่ละเส้นควรมีระยะห่างช่วงละเท่า ๆกัน เพื่อไม่ให้เกิดความหนาแน่นมากจนเกินไป
2. เรียงลำดับการทำงานแบบต่อเนื่อง

3. ลายเส้นควรปันข้ามน้อยที่สุด (วิธีแก้ ใช้ jumper จั๊มสายข้าม) ป้องกัน GND Loop กรณีหน้าเดียว กรณี 2 หน้า ก็ใช้ pth (plated through holes)

4. มุมหักไม่ควรเป็นมุมฉาก หรือ เป็นรูปตัว V เพราะเหตุว่ามีปัญหา ตอนกัดลายทองแดงไม่ออก ทำให้แผ่นพิมพ์ลายลอกได้ง่าย เช่นได้รับความร้อนจากการบักกรีได้ง่าย หรือกรณีมีกระแสไหลผ่านตัว V มาก ๆ ควรให้ลายเป็นเส้นโค้ง

5. ขนาดลายวงจรใหญ่ หรือ เล็กขึ้นอยู่กับขนาดของกระแส

6. ในระบบดิจิตอล ปัญหาเรื่องกราวด์ลูบ จะทำให้เกิด double pulse วิธีแก้ GND loop ทำได้ โดยร้างลายเส้น ในอยู่ในรูปแบบกราวด์จุดเดียว

การสะท้อนของสัญญาณ (Bus termination)

Bus termination network เป็นการกำจัดสัญญาณสะท้อนในบัส ซึ่งอาจเกิดจากการสวิตช์ของทรานซิสเตอร์ที่ความเร็วสูง การเกิด ringing จากคุณสมบัติความจุไฟฟ้าและความเหนี่ยวไฟฟ้าทำให้การอ่านข้อมูลไปไม่แน่นอน เกิดความผิดพลาดขึ้นได้

การรีเซทขณะเปิดเครื่อง (POWER-ON RESET)

ขณะเปิดเครื่องใหม่ ๆ จะเกิด RESET โดยอัตโนมัติ โดยตัวของมันเองให้อยู่ในสภาวะเริ่มต้น

เราต้องมี POWER ON RESET เนื่องจากก่อนที่เราจะส่ง Control word Register ใน IC อยู่ในสภาวะที่มีข้อมูลไม่แน่นอน การทำ POWER ON RESET ทำให้ IC อยู่ในสภาวะเริ่มต้น พร้อมที่จะรับ control word และ command word ได้

Relay ขนาดเล็ก MATSUSHITA

มองจากด้านบน

INPUT	5VDC
OUTPUT VOLT	12V
CURRENT	1A.

อธิบายการทำงาน ของRelay ชนิดนี้

กรณีป้อนไฟบวกที่ ขา 1 และไฟ GND ที่ขา 16

- Switch ของหน้าสัมผัสจะทำงาน คือ ตอนแรก 4 short กับ 6 ก็จะเปิดออก เป็น open และ ขา 8 กับ 6 ก็จะ close และขา 9 11 13 ก็เช่นเดียวกัน กรณีเอาไฟออกจาก ขา 1 และขา 16
- contact Relay จะค้างอยู่ เนื่องจาก machanics ภายในของมันเป็นเช่นนั้น กรณีป้อนไฟบวกที่ขา 2 และ GND ขา 15
- contact ข้างในจึงจะทำงานอีกครั้ง คือ ต้องป้อนคู่ 1 และ 16 กับ 2 และ 15 สลับ Relay ถึงจะทำงานเปิดและปิดเหมือนเดิม หรืออีกกรณี คือ ป้อนไฟสลับขั้ว โดยตอนแรก ป้อน ขา 1 (+) กับขา 16 (-) เมื่อเอาไฟออก Relay จะค้าง เราก็ป้อน ขา 1(-) กับขา 16 (+) จึงจะกลับเหมือนสภาพเดิม แต่ยากและลำบาก จึงควรใช้วิธีแรกจะดีกว่า

4.3.2 การออกแบบทาง SOFTWARE

ในการออกแบบระบบทางซอฟท์แวร์ โดยแบ่งออกเป็น 2 ส่วน คือ

1. หน่วยป้อนโปรแกรม

2. หน่วยประมวลผล

ซึ่งทั้งสองส่วนนี้ มีการเชื่อมถึงกันโดยพอร์ทอนุกรม

โปรแกรมที่อยู่บนหน่วยป้อนโปรแกรม

การอธิบาย ROUTINE แต่ละ ROUTINE ในหน่วยป้อนโปรแกรม เป็นหน่วยย่อย ๆ ดังนี้ (โดยแต่ละ ROUTINE จะมีการอ้าง ROUTINE ย่อย ๆ ตามรูป FLOW CHART ในภาคผนวก จึงขออธิบาย ผลที่จะเกิดขึ้น ของแต่ละ ROUTINE)

POWER_ON :

เป็นการเริ่มต้นของระบบ โดยเซ็ท การแสดงผลของ LCD, control word ไปที่ 8255 มีการ test RAM ว่าใช้ได้หรือไม่ ตรวจการใช้ว่าใส่รหัสผ่านหรือไม่ เมื่อผ่านจากรหัส จะมีการเลือกรูปแบบว่า ต้องการเข้าใน mode ใด

โดยมีการเรียกใช้รูทีน TEST_RAM, PASSWORD, SELMOD

SEL_MOD :

เป็นการเลือกโหมดการแสดงว่า จะเป็นโหมดใด(monitor, program, run)

มีการเรียกใช้รูทีน MOD_MON, MOD_PRO, MOD_RUN

MOD_MON :

มีการแสดงค่าในหน่วยความจำของผู้ใช้ว่าได้ป้อนโปรแกรมอะไรไปแล้ว แล้วหน่วยประมวลผลได้กระทำคำสั่งอะไรไปบ้าง ตามลำดับ

มีการเรียกรูทีน DISPLAY, F3

MOD_PRO :

มีการเข้าการโปรแกรม โดยเมื่อผู้ใช้เลือกในโหมดนี้ จะมีการแสดงบรรทัดเริ่มต้น จะโปรแกรม แล้วรอรับการกดคีย์ว่ากดคีย์ใด

มีการเรียกรูทีน DOWN_1, F3

MOD_RUN :

เหมือนในโหมด MONITOR แต่จะไม่แสดงค่าออกมาให้ทราบว่ากระทำไปถึงไหน แต่จะกระทำไปตามลำดับไปเรื่อยๆ

DOWN_1 :

เป็นรูทีนที่มีการเช็คค่าคีย์ว่า เป็นการกดคีย์ใด(LD, OUT, AND, OR, TIM, CNT, FUN) เพื่อจะได้เข้าไปในรูทีนย่อยนั้นๆ

TO_LD :

มีการแสดงเครื่องหมาย LD บน LCD และ เรียกรูทีน KEY_LD

TO_OUT :

มีการแสดงเครื่องหมาย OUT บน LCD และ เรียกรูทีน KEY_OUT

TO_AND :

มีการแสดงเครื่องหมาย AND บน LCD และ เรียกรูทีน KEY_AND

TO_OR :

มีการแสดงเครื่องหมาย OR บน LCD และ เรียกรูทีน KEY_OR

TO_TIM :

มีการแสดงเครื่องหมาย TIM บน LCD และ เรียกรูทีน KEY_TIM

TO_CNT :

มีการแสดงเครื่องหมาย CNT บน LCD และ เรียกรูทีน KEY_CNT

TO_FUN :

มีการแสดงเครื่องหมาย FUN บน LCD และ เรียกรูทีน KEY_FUN

NUM_1 :

เป็นการเขียน ตัวเลข บน LCD ในลักษณะรูปแบบ channal : port

โดย channal มี การแสดง 2 หลัก

port มี การแสดง 2 หลัก

มี เครื่องหมาย " : " คั่นระหว่าง channal กับ port

โดยมีการเรียกรูทีน NUMBER, READKEY, CH_PORT

WRITETWOCHAR :

เป็นรูทีนย่อย ๆ ของ รูทีน READKEY (ค่าสแกนคีย์)ในการรีเทินค่ากลับออกมาเป็น ค่า R7

NUM_2 :

มีการรับค่าตัวเลข เพื่อแสดงค่าตัวเลข โดยแสดงออกเป็น 4 หลัก

มีการเรียกใช้รูทีน NUMBER, CH_PORT, READKEY

NUM_3 :

มีการรับค่าตัวเลข เพื่อแสดงค่าตัวเลข โดยแสดงออกเป็น 2 หลัก

มีการเรียกใช้รูทีน NUMBER, CH_PORT, READKEY

NUMBER :

เป็นการจากัดค่า R7 ให้มีค่าเป็นเพียงตัวเลขเท่านั้น

NO_CURSOR :

เป็นเซ็ท command word ไปที่ LCD ด้วยค่า 0C ไม่ให้ปรากฏเครื่องหมายพร้อม

SECOND_LINE :

เป็นเซ็ท command word ไปที่ LCD ด้วยค่า C0 เป็นการสั่งให้เขียน ไปที่ DD RAM หรือ ขึ้นบรรทัดที่ สอง

40H

CUR_ON :

เป็นเซ็ท command word ไปที่ LCD ด้วยค่า 0E ให้ ปรากฏ เครื่องหมายพร้อม

CLEAR :

เป็นเซ็ท command word ไปที่ LCD ด้วยค่า 01H

RETURN_HOME :

เป็นเซ็ท command word ไปที่ LCD ด้วยค่า 02H

KEY_OUT :

เป็นการรับค่าคีย์ว่ามีการกด NOT หรือไม่ ถ้ามีก็เขียน NOT บน LCD แล้วตามด้วยการเรียกรูทีน NUM_1

KEY_TIM :

มีการเรียก รูทีน NUM_2 แล้วเว้นช่องว่าง 2 ช่อง แล้วเรียก NUM_2 อีกครั้ง

KEY_CNT :

มีการเรียก รูทีน NUM_2 แล้วเว้นช่องว่าง 2 ช่อง แล้วเรียก NUM_2 อีกครั้ง

KEY_FUN :

มีการเรียก รูทีน NUM_3 เพื่อรับค่า และแสดงค่า ตัวเลข 2 หลัก

PASSWORD :

เป็นการเซ็ทไว้ว่าจะมีการกดคีย์ถูกต้องตามที่ต้องการเซ็ทไว้หรือไม่ สามารถเซ็ทได้

DISPLAY :

เป็นการแสดงอักษรบนจอ LCD ตามที่ต้องการโดยโหลดลงใน DD RAM ของ LCD

PASS_LET :

เป็นการแสดงว่าคีย์ที่กดถูกต้อง

WR_DATA :

เป็นการข้อมูลจาก RAM ลงแสดงบน LCD ในรูปเลขฐานสิบหก 2 หลัก

KEY_LD :

เป็นการรับค่าคีย์ว่ากดคีย์ใด ก็เข้าไปเรียกรูทีนนั้นๆ อีกที โดยมีการตรวจคีย์ดังนี้ ว่ากดคีย์ NOT, TIM, CNT หรือ ตัวเลข

KEY_AND :

รับค่าคีย์ว่ามีการกด LD หรือไม่ ถ้ามี ก็แสดงค่า LD บน จอ LCD ถ้าไม่มี ก็เรียกรูทีน CHECK_FN

KEY_OR :

รับค่าคีย์ว่ามีการกด LD หรือไม่ ถ้ามี ก็แสดงค่า LD บน จอ LCD ถ้าไม่มี ก็เรียกรูทีน CHECK_FN

WR_DIGIT :

เป็นการแสดงเลขที่ของบรรทัด ที่แสดงอยู่เทียบกับค่าใน RAM ที่ 2000H

CHECK_FN :

มีการรอรับค่าคีย์ ว่าเป็นการกดปุ่มใด (NOT, TIM, CNT,

CH_PORT :

มีการแสดงค่าตัวเลข channal และ port

LD_TIM :

เป็นการแสดง TIM บน LCD และ รอรับค่าคีย์ตัวเลข 4 หลัก

มีการเรียกใช้รูทีน NUM_2, DISPLAY

LD_CNT :

เป็นการแสดง CNT บน LCD และ รอรับค่าคีย์ตัวเลข 4 หลัก

มีการเรียกใช้รูทีน NUM_2, DISPLAY

LD_NOT :

เป็นการแสดง NOT บน LCD และรอรับค่าจากคีย์ว่าจะกดปุ่มใด(TIM, CNT, หรือ ตัวเลข) เมื่อกดปุ่มใด ก็เข้าไปเรียกในรูทีนย่อย นั้นๆ

มีการเรียกใช้รูทีน READKEY, LD_TIM, LD_CNT, NUM_1, DISPLAY

F3 :

เป็นการแสดงบรรทัดของโปรแกรม โดยมีการเพิ่มค่าทีละ 1 บรรทัดทุกครั้งที่ถูกเรียก

F4 :

เป็นการแสดงบรรทัดของโปรแกรม โดยมีการลดค่าทีละ 1 บรรทัดทุกครั้งที่ถูกเรียก

PLC.ASM เป็นโปรแกรมหลักที่อยู่บนส่วน PLC เมนบอร์ด

ประกอบไปด้วย 3โหมด ที่เหมือนกันกับโปรแกรมที่อยู่บนส่วนรับข้อมูล(lcd momule)เพียงแต่ว่าส่วนรับข้อมูลมีหน้าที่หลักคือจัดการเกี่ยวกับการรับข้อมูลจากผู้ใช้(USER)แล้วเก็บข้อมูลเอาไว้ในหน่วยชั่วคราวเมื่อผู้ใช้ป้อนข้อมูลหรือโปรแกรมเสร็จแล้วก็สั่งให้ส่วนรับข้อมูลส่งโปรแกรมที่ผู้ใช้แล้วไปประมวลผลคำสั่งอีกทีที่ส่วนโปรแกรมหลักวิธีการติดต่อผ่านทางพอร์ตอนุกรม(serial)นั่นคือโปรแกรมหลักมีหน้าที่นำชุดคำสั่งที่ป้อนเสร็จเรียบร้อยแล้วจากหน่วยรับข้อมูลไปประมวลผล ตามคำสั่งนั้นๆที่ละคำสั่งตั้งแต่คำสั่งแรกจนถึงคำสั่งสุดท้ายเมื่อครบคำสั่งสุดท้ายแล้วก็จะเริ่มกระโดดไปทำคำสั่งแรกใหม่จนกระทั่งคำสั่งสุดท้ายอีกเป็นอย่างนี้ไปเรื่อยๆ จนกว่าจะมีคำสั่งหยุดหรือคำสั่งเปลี่ยนโหมด ช่วงเวลาในการนำคำสั่งแรกมาประมวลผลจนถึงคำสั่งสุดท้ายก่อนที่จะกระโดดมากระทำคำสั่งแรกใหม่ในรอบต่อไปเราเรียกว่า 1 รอบสแกน ช่วงเวลาในการสแกนจะมากหรือน้อยขึ้นอยู่กับความยาวของโปรแกรมที่ผู้ใช้งานป้อนเข้ามา แต่ข้อกำหนดช่วงเวลาในการสแกนไม่ควรเกิน 100 msec เพื่อความสามารถในการติดตามการเปลี่ยนแปลงระดับสัญญาณอินพุตและตอบสนองต่อเอาท์พุตให้ทันเวลา

PLC.ASM ประกอบด้วย 3 โหมด

1.โหมดโปรแกรม(PROGRAM MODE)

2.โหมดรัน(RUN MODE)

3.โหมดมอนิเตอร์(MONITOR MODE)

แต่ละโหมดต้องมีการติดต่อกับ serial subroutine อยู่เสมอเพื่อตรวจสอบโหมดการทำงานเพื่อให้รู้ว่าคำสั่งอยู่ในโหมดใด และกระโดดออกไปทำงานยังโหมดอื่นเมื่อต้องการเปลี่ยนโหมด ซึ่งครั้งแรกที่เกิด power on reset หรืออาจจะ reset โดย สวิตช์ก็ได้ซึ่งนั่นคือการเริ่มต้นใหม่ โปรแกรมหลักก็เข้าสู่โปรแกรมหลัก CHECK_MOD จะทำการติดต่อกับส่วนของ lcd module ที่ทำหน้าที่รับข้อมูลว่าจะเลือกเข้าสู่โหมดใด โดยจะติดต่อผ่านทางพอร์ตอนุกรม แอดเดรสที่โหลดชุดคำสั่งมาเก็บไว้ก่อนที่จะมีการนำคำสั่งเหล่านี้ไปประมวลผลคือแอดเดรสที่ 2000H ซึ่งจะถูกชี้โดยดาต้าพอยท์เตอร์รีจิสเตอร์(DATA POINTER, DPTR)เป็นรีจิสเตอร์ขนาด 16 บิต

กรณีที่เข้าสู่ส่วนของโหมดโปรแกรมก็จะกระโดดไป PROGRAM MODE ROUTINE ในส่วนนี้มีหน้าที่ติดต่อกับส่วน lcd module หรือส่วนรับข้อมูล ว่าจะมีการส่งชุดคำสั่งเข้าหรือไม่ และยัง

มีการตรวจสอบว่าจะมีการเปลี่ยนโหมดการทำงานหรือไม่ ถ้ามีการเปลี่ยนโหมดก็จะออกจากโหมดโปรแกรมกลับไปยังเมนโปรแกรม check mode ก่อนเพื่อตรวจสอบว่าต้องการโหมดใดที่ไม่ใช่โหมดที่กระทำอยู่ หลังจากนั้นก็จะกระโดดไปยังโหมดที่ต้องการ

กรณีที่เข้าสู่โปรแกรมมอนิเตอร์ในโหมดนี้มีหน้าที่คือตรวจสอบสภาวะของอินพุตและเอาพุตต์ สามารถกำหนดโดย set และ reset relayได้ด้วย (FORCED SET/RESET) และจะตรวจสอบว่าจะมีการเปลี่ยนโหมดการทำงานหรือไม่ ถ้ามีคำสั่งจาก lcd module ก็จะออกจากมอนิเตอร์โหมดเข้าสู่ check mode เพื่อตรวจสอบการเปลี่ยนโหมดอีกครั้งก่อนที่จะกระโดดไปโหมดโปรแกรมอื่นต่อไป

กรณีเข้าสู่โหมดรัน ซึ่งเป็นส่วนที่สำคัญมากสำหรับส่วนโมดูลนี้ เราสามารถจะทำให้โครงงานนี้มีประโยชน์ก็ต้องสามารถนำโปรแกรมที่ user ป้อนนั้นมาประมวลผลตามคำสั่งตามลักษณะของงานควบคุมที่ต้องการได้ และจะตรวจสอบการเปลี่ยนโหมดด้วยลักษณะการตรวจที่คล้ายกับโหมดอื่นๆ

อธิบายรายละเอียดโปรแกรมต่างๆ

RUN_MODE ROUTINE สำคับการทำงานมีดังนี้

เป็นโปรแกรมที่อยู่ในโหมดรันเมื่อเริ่มเข้ามาสู่โหมดนี้มันจะเริ่มการประมวลผลคำสั่ง ทีละคำสั่งที่แอดเดรส 2000H (หลังจากที่มีการ โหลดชุดคำสั่งจากส่วนรับข้อมูลมาเก็บไว้แล้วในโปรแกรมโหมด แล้ว) เป็นคำสั่งแรกซึ่งคำสั่งแรกจะต้องเป็นคำสั่ง LD เท่านั้นเนื่องจากว่าในกระบวนการควบคุมใดๆต้องมีการควบคุมจากอินพุตเสมอ ถ้าคำสั่งแรกไม่เป็นคำสั่ง LD จะแสดงข้อผิดพลาดออกมา เมื่อผ่านคำสั่งแรกมาแล้วก็จะตรวจสอบบรรทัดคำสั่งต่อไปว่าเป็นคำสั่งใดก็จะกระโดดไปยังโปรแกรมย่อยของคำสั่งนั้นและจะทำคำสั่งนั้นให้เสร็จที่ละคำสั่ง ก่อนที่จะกระทำคำสั่งที่บรรทัดคำสั่งถัดไป(ความหมายของบรรทัดคำสั่งถัดไป ไม่เหมือนกับแอดเดรสถัดไป ซึ่งแอดเดรสถัดไปเราสามารถทราบได้เลยว่าห่างกันหนึ่งไบต์ แต่บรรทัดคำสั่งนั้นเมื่อคิดเป็นแบบแอดเดรสแล้วเราไม่สามารถกำหนดความห่างว่ากี่ไบต์แน่ เนื่องจากว่าแต่ละคำสั่งจะใช้หน่วยความจำเก็บไม่เท่ากัน)และก็จะทำการตรวจสอบว่าพบคำสั่ง END หรือยังถ้ายังให้ทำคำสั่งต่อไป แต่ถ้าพบคำสั่ง END ถือว่าจบโปรแกรมที่บรรทัดคำสั่งนั้น เมื่อพบคำสั่งจบโปรแกรมมันกระโดดไปทำ watch dog ซึ่งในโปรแกรมตัวนี้จะเป็นตัว reset watch dog flag เสมอเมื่อพบคำสั่ง END

เพื่อตรวจสอบสภาวะการทำงานของรันโหมดว่ามีการวนลูปอนันต์หรือไม่ หรือมีการหลุดออกจากคำสั่งหรือไม่ ถ้าไม่มีข้อผิดพลาดหรือเวลาการสแกนไม่นานมากจนเกินไปแล้ว watch dog flag จะถูก reset ทุกครั้งเมื่อพบคำสั่ง END เพื่อให้ timer 0 อินเท่อร์รัพต์รูทีนตรวจสอบเมื่อครบ 100 msec ว่า watch dog flag อยู่ในสภาวะ set หรือ reset ถ้าอยู่ในสภาวะ reset ก็จะออกจากโปรแกรมย่อย watch dog กรณีที่ watch dog flag อยู่ในสภาวะ set มันจะกระโดดไปยังแอดเดรส 0000h (power on reset address) เพื่อเริ่มต้นทุกอย่างใหม่หมด ซึ่งอาจจะเป็นที่สงสัยได้ว่าทุกครั้งที่กระทำคำสั่งในหนึ่งรอบการสแกนก็มีการ reset flag ทุกครั้งแล้วทำไมจึงเกิดสภาวะ set ได้คำตอบก็คือ ตัว timer0 interrupt จะทำการ set flag ทุกๆ 1 msec จนครบ 100 ครั้ง(100 msec)จึงค่อยตรวจสอบสภาวะว่า setหรือreset หลังจากออกจาก watch dog routine ก็จะตรวจสอบว่ามีคำสั่งเปลี่ยนโหมดหรือไม่ถ้าไม่มีการเปลี่ยนโหมดก็กระโดดไปยังแอดเดรส 2000h เป็นบรรทัดคำสั่งแรก วนอย่างนี้ไปเรื่อยๆ

routine run mode จะทำการเรียกใช้โปรแกรมย่อยดังนี้

- RUN_LD
  - LD_
    - -GET_RL_BIT
    - -SAV_IN_IP
  - LD_NOT_
    - -GET_RL_BIT
    - -SAV_IN_BIT
  - LD_TIM_
    - -GET_TIM_BIT_RL
  - LD_NOT_TIM_
    - -GETTIM_BIT_RL
  - LD_CNT_
    - -GETCNT_BIT_RL
  - LD_NOT_CNT_
    - -GETCNT_BIT_RL
- RUN_AND

- AND_
    - -GET_RL_BIT
    - -SAV_IN_IP
- AND_NOT_
    - -GET_RL_BIT
    - -SAV_IN_IP
- AND_TIM_
    - -GETTIM_BIT_RL
- AND_NOT_TIM_
    - -GETTIM_BIT_RL
- AND_CNT_
    - -GETCNT_BIT_RL
- AND_NOT_CNT_
    - -GETCNT_BIT_RL
- AND_LD_

- RUN_OR
    - OR_
        - -GET_RL_BIT
        - -SAV_IN_IP
    - OR_NOT_
        - -GET_RL_BIT
        - -SAV_IN_IP
    - OR_TIM_
        - -GETTIM_BIT_RL
    - OR_NOT_TIM_
        - -GETTIM_BIT_RL
    - OR_CNT_
        - -GETCNT_BIT_RL
    - OR_NOT_CNT_

-GETCNT_BIT_RL

- OR_LD_

- RUN_OUT

- OUT_

-OUTP_BIT_INRL

-OUTP_BIT_PORT

-OUT_INTER_BIT

- OUT_NOT_

-OUTP_BIT_INRL

-OUTP_BIT_PORT

-OUT_INTER_BIT

- RUN_TIM

- RUN_CNT

โปรแกรมย่อยระดับต่ำสุดที่ถูกเรียกใช้โดยโปรย่อยอื่นๆดังนี้

โปรแกรมย่อย GET_RL_BIT(GET RELAY NUMBER XX:YY)

XX => CHANNEL .YY =>RELAY BIT

เป็นโปรแกรมทำหน้าที่รับค่าอินพุตบิตที่ต้องการเข้ามาค่าอินพุตมี 2 ลักษณะคือ รับอินพุตจากอุปกรณ์ภายนอกหรือรับค่าจากรีเลย์ภายในหรือรีเลย์ช่วย(internal auxilary relay)โดยจะทำการตรวจสอบ แชนแนล(channel number) และบิตที่เท่าใดของแชนแนลนั้นๆหรือพอร์ตที่เท่าใด (พอร์ตในที่นี้มีความหมายเดียวกับจุด(point)) ถ้าเป็นแชนแนล 00 คือรับค่าจากอุปกรณ์ภายนอก แชนแนล 09 คือรับค่าจากรีเลย์ภายในซึ่งเป็นบิตรีจิตเตอร์นั่นเอง (relay number ประกอบไปด้วย channel และ relay number)เมื่อรับอินพุตรีเลย์แล้วจะส่งผ่านค่าทาง carry flag ซึ่งเป็น bit register การอ้างสามารถอ้างได้โดยตรง ซึ่งจะให้รูททีนอื่นๆที่เรียกใช้นำผลที่ได้จาก carry flag ไปใช้ต่อไป

โปรแกรมย่อย SAV_IN_IP (SAVE INTERNAL INPUT RELAY NUMBER XX:YY)

เป็นโปรแกรมที่ใช้คู่กับ GET_RL_BIT โดยจะนำค่าของอินพุตภายนอกที่ได้ไปเก็บไว้ใน

หน่วยความจำชั่วคราวในตารางของ internal input relay เพื่อใช้ประโยชน์ในการตรวจสอบสภาวะของอินพุตได้โดยไปตรวจสอบค่าในตารางที่กล่าวมานี้ โดยที่ต้องการ channel number ,relay bit number และ temp(เป็นรีจิสเตอร์ 1 บิต ชั่วคราว โดยจะรับค่าผ่านทาง carry flag อีกทีหนึ่ง

โปรแกรมย่อย GETTIM_BIT_RL(GET TIMER BIT RELAY NUMBER XX)

จะนำค่าจากบิตรีจิสเตอร์ของ timer00 ถึง07 บิตใดบิตหนึ่งออกมาใช้โดยส่งค่าผ่านทาง carry flag

โปรแกรมย่อย GETCNT_BIT_RL(GET COUNTER BIT RELAY NUMBER XX)

จะนำค่าจากบิตรีจิสเตอร์ของ counter08-0F บิตใดบิตหนึ่งออกมาโดยผ่านค่าทางcarry flag register

โปรแกรมย่อย OUTP_BIT_INRL(OUTPUT TO INTERNAL OF OUTPUT BIT RELAY)

เป็นโปรแกรมที่เก็บสภาวะของเอาพุตไว้ที่หน่วยความความจำชั่วคราวภายใน ก่อนที่จะส่งออกไปควบคุมเอาพุตภายนอก เพื่อเอาไว้ตรวจสอบสภาวะในมอนิเตอร์โหมด

โปรแกรมย่อย OUTP_BIT_PORT(OUTPUT TO SPECIFIED BIT PORT)

เป็นโปรแกรมส่งค่าใน result register ออกไปควบคุมข้างนอกเครื่องควบคุม ซึ่งออกทางบิตพอร์ตที่เราต้องการ

โปรแกรมย่อย OUT_INTER_BIT(OUTPUT RESULT TO INTERNAL AUXILARY REGISTER) ส่งค่าของ result register ไปเก็บยังรีเลย์ช่วยภายในซึ่งเมื่อใช้คำสั่ง OUT ไปยังพอร์ตเอาท์พุตแต่ไม่จำเป็นที่ต้องส่งออกไปยังพอร์ตจริงๆ แต่อาจจะเป็นหน่วยความจำที่มีผลให้เสมือนกับว่าเป็นเอาท์พุตรีเลย์คอยล์(output relay coil)เราสามารถนำค่าที่เก็บไว้นั้นมาใช้เป็นค่าอินพุตเพื่อการควบคุมกระบวนบางอย่างง่ายและละดวกยิ่งขึ้น

การดำเนินการตามคำสั่งต่างๆ(OPERATION INSTRUCTION) ซึ่งสามารถดูได้จากโฟว์ชาร์ตและโปรแกรมส่วนเมนบอร์ดซึ่งรูททีนแต่ละตัวยังมีการเรียกใช้รูททีนย่อยๆลงไปอีกจนกระทั่ง

ถึงรูทที่นระดับต่ำสุดที่ไม่สามารถเรียกได้อีกต่อไป รายละเอียดแสดงเอาไว้ข้างล่างนี้

("--> " = store)

LD INSTRUCTION

- result register --> stack register

-content of specified relay number xx:yy --> result register

โดยที่ specified relay อาจจะเป็น input relay xx:yy หรือ internal auxilary relay ก็ได้

LD NOT INSTRUCTION

-result register --> stack register

-content of specified relay number xx:yy --> temp (temporary register)

- not temp

-temp -->result register

LD TIMER INSTRUCTION

-result register --> stack register

-content timer relay (internal relay)number xx -->result register

LD NOT TIMER INSTRUCTION

-result register --> stack register

-content timer relay number xx --> temp

-not temp(/ temp)

-temp -->result register

LD COUNTER INSTRUCTION

-result register --> stack register

-content counter relay number xx --> result register

LD NOT COUNTER INSTRUCTION

-result register -->stack register

-content counter relay number xx --> temp

-not temp(/temp)

-temp --> result register

AND INSTRUCTION

-content of specified relay number xx:yy -->temp

-result register.AND.temp--> result register

AND NOT INSTRUCTION

-content of specified relay number xx:yy -->temp

-result register.AND./temp--> result register

AND LD INSTRUCTION

-stack register--> temp

-result register.AND.temp-->result register

AND TIM INSTRUCTION

-content of timer (internal)relay number xx:yy -->temp

-result register.AND.temp-->result register

AND NOT TIMER INSTRUCTION

-content of timer (internal)relay number xx:yy -->temp

-result register.AND./temp-->result register

AND COUNTER INSTRUCTION

-content of counter (internal)relay number xx:yy -->temp

-result register.AND.temp-->result register

AND NOT COUNTER INSTRUCTION

-content of counter (internal)relay number xx:yy -->temp

-result register.AND./temp-->result register

OR INSTRUCTION

-content of specified relay number xx:yy -->temp

-result register.OR.temp--> result register

OR NOT INSTRUCTION

-content of specified relay number xx:yy -->temp

-result register.OR./temp--> result register

OR LD INSTRUCTION

-stack register--> temp

-result register.OR.temp-->result register

OR TIM INSTRUCTION

-content of timer (internal)relay number xx:yy -->temp

-result register.OR.temp-->result register

OR NOT TIMER INSTRUCTION

-content of timer (internal)relay number xx:yy -->temp

-result register.OR./temp-->result register

OR COUNTER INSTRUCTION

-content of counter (internal)relay number xx:yy -->temp

-result register.OR.temp-->result register

OR NOT COUNTER INSTRUCTION

-content of counter (internal)relay number xx:yy -->temp

-result register.OR./temp-->result register

OUT INSTRUCTIN

-result register --> specified output relay number xx:yy

OUT NOT INSTRUCTIN

-result register -->temp

-/temp (not temp)-->specified output relay number xx:yy

การทำงานของโปรแกรม timer/counter

TIMER

การกำหนดเบอร์ของ timer เบอร์ 00-07 ซึ่งการทำงานของtimer เป็นไปตามค่าของ result register(RR) และ stack register(SR) ดังต่อไปนี้

RR = RESET TIMER

SR = START TIMER

RR	SR	การทำงาน
0	0	reset timer,load preset value
0	1	เริ่มจับเวลา=preset value,set value
1	0	reset timer,load preset value
1	1	reset timer,load preset value

จากตารางข้างบนจะเห็นว่าเป็นการทำงานของ ON-DELAY TIMER เท่านั้น การเริ่มจับเวลาจะจับเวลาจนมีค่าเท่ากับค่า preset value ที่ผู้ใช้ป้อนเอาไว้และจะไป set bit ของ timer เบอร์ที่กำหนดเอาไว้ และจะค้างไว้จนกว่าจะมีการรีเซ็ตใหม่และเริ่มต้นโหลดค่า preset value ใหม่

COUNTER

counter number 08 -0F

จะตรวจสอบค่าใน result register(RR) และ stack register(SR)

RR = RESET

SR = START COUNTER

RR	SR	การทำงาน
0	0->1	ค้างค่านับเอาไว้
0	1->0	counter:=counter-1
1	0->1	reset counter,load couter preset-value
1	1->0	reset counter.load counter preset value

การทำงานของ counter(หลังจากที่ทำการ โหลดค่า preset counter)จะทำการลดค่าของ preset value ทุกครั้งที่มีสัญญาณอินพุตจากภายนอกในลักษณะขอบขาลง(1->0) จนกระทั่งค่าใน preset value เป็นศูนย์ก็จะทำการเซ็ตบิตของ counter number (internal relay)

ชุดคำสั่งของ PLC ในโครงงานนี้

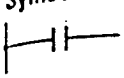
1. LD (channel) : (port)
2. LD NOT (channel) : (port)
3. LD TIM (No:)
5. LD CNT (No:)
6. LD NOT TIM (No:)
7. LD NOT CNT (No:)
8. OUT (channel) : (port)
9. OUT NOT (channel) : (port)
10. AND (channel) : (port)
11. AND NOT (channel) : (port)
12. AND TIM (No:)
13. AND CNT (No:)
14. AND NOT TIM (No:)
15. AND NOT CNT (No:)
16. OR (channel) : (port)
17. OR NOT (channel) : (port)
18. OR TIM (No:)
19. OR CNT (No:)
20. OR NOT TIM (No:)
21. OR NOT CNT (No:)
22. AND LD
23. OR LD
24. TIM (No:) (set value)
25. CNT (No:) (set value)
26. NOP
27. END

ซึ่งแต่ละคำสั่งมีลักษณะ ดังนี้

LOAD (LD)

The contents of specified relay no. — R — S

Symbol



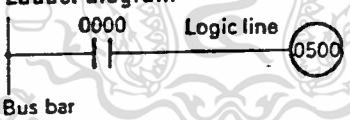
Contents of data	LD	LD-NOT AND, AND-NOT OR, OR-NOT	OUT
I/O relay, internal auxiliary relay	0000 to 1907		0500 to 1807
Holding relay	HR 000 to 915		
Timer/counter	TIM/CNT00 to 47		
Temporary memory relay	TR 0 to 7	–	TR 0 to 7

OUTPUT (OUT)

Symbol

R — Specified relay no.

Ladder diagram

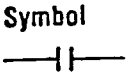


Coding chart

Address	Instruction	Data
0000	LD	0000
0001	OUT	0500

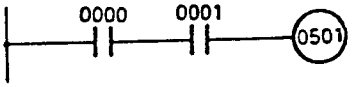
AND

Symbol



The contents of specified relay no. — AND — R

Ladder diagram

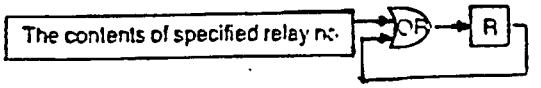


Coding chart

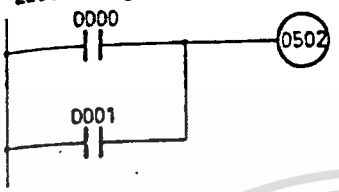
Address	Instruction	Data
0003	LD	0000
0004	AND	0001
0005	OUT	0501

OR

Symbol



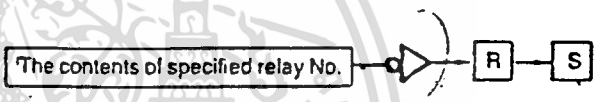
Ladder diagram



Coding chart

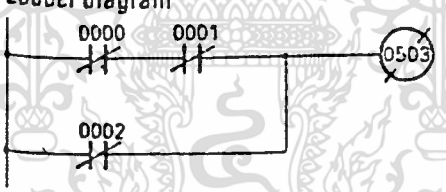
Address	Instruction	Data
0006	LD	0000
0007	OR	0001
0008	OUT	0502

LOAD-NOT (LD-NOT)



Symbol

Ladder diagram

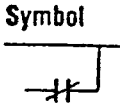


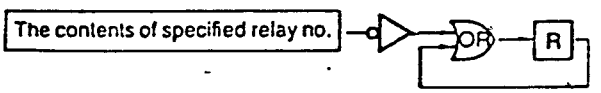
Coding chart

Address	Instruction	Data
0009	LD-NOT	0000
0010	AND-NOT	0001
0011	OR-NOT	0002
0012	OUT-NOT	0503

OR-NOT

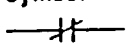
Symbol

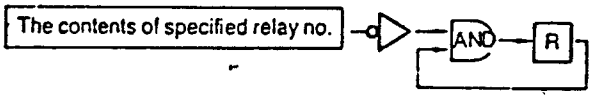




AND-NOT

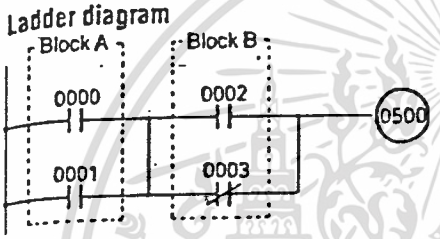
Symbol

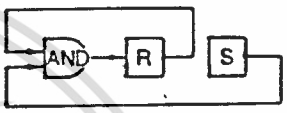




AND-LOAD (AND-LD)

Ladder diagram



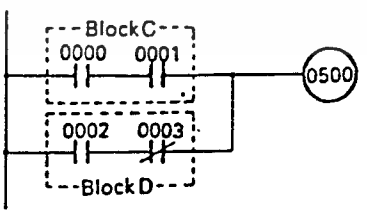


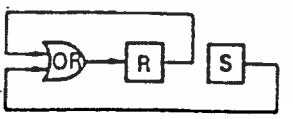
Coding chart

Address	Instruction	Data
0200	LD	0000
0201	OR	0001
0202	LD	0002
0203	OR-NOT	0003
0204	AND-LD	–
0205	OUT	0500

OR-LOAD (OR-LD)

Ladder diagram





Coding chart

Address	Instruction	Data
0200	LD	0000
0201	AND	0001
0202	LD	0002
0203	AND-NOT	0003
0204	OR-LD	–
0205	OUT	0500

TIMER (TIM)

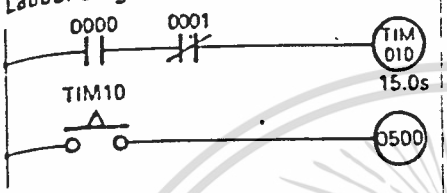
Symbol

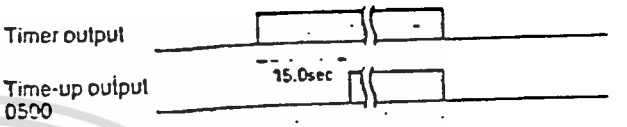
Coding chart

Address	Instruction	Data
0200	LD	0000
0201	AND-NOT	0001
0202	TIM	10*
		# 0150**
0203	LD	TIM 10
0204	OUT	0500

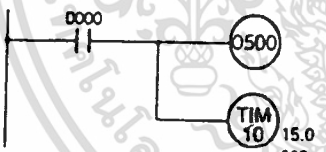
*Timer number
**Set time value

Ladder diagram





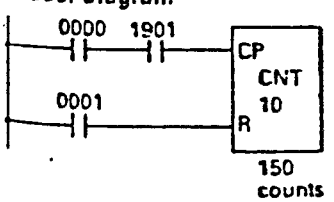
Ladder diagram



Coding chart

Instruction	Data
LD	0000
OUT	0500
TIM	10
	# 0150

Ladder diagram

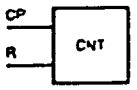


Coding chart

Instruction	Data
LD	0000
AND	1901
LD	0001
CNT	10
	# 0150

COUNTER (CNT)

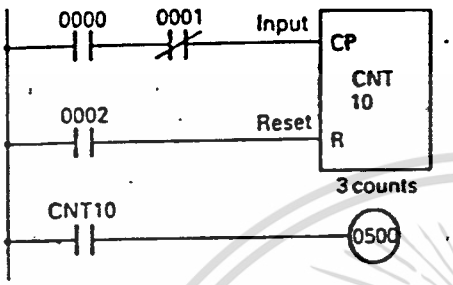
Symbol



Coding chart

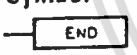
Address	Instruction	Data
0200	LD	0000
0201	AND-NOT	0001
0202	LD	0002
0203	CNT	10
		# 0003
0204	LD	CNT 10
0205	OUT	0500

Ladder diagram

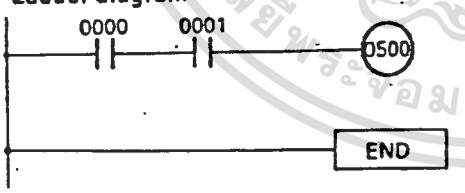


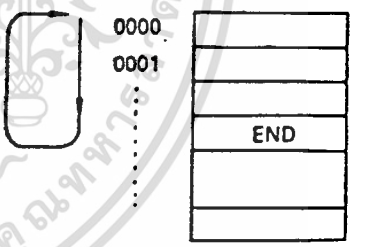
END

Symbol



Ladder diagram





เมื่อต้องการป้อนโปรแกรมด้วยคำสั่งใด ก็กดตามคำสั่งที่ต้องการนั้นๆ เช่น ต้องการกด คำสั่ง

LD NOT TIM (No:)

เริ่มด้วย การทำตามขั้นตอน ตามลำดับ ดังนี้

1. กดปุ่ม LD
2. กดปุ่ม NOT
3. กดปุ่ม TIM
4. กดปุ่ม ตัวเลขที่ต้องการ
   - ตัว (No:) มีการกำหนด
     โดยโปรแกรมให้มีการรับค่า เพียงตัวเลข 4 หลัก
   - ตัว (channel) : (port) มีการกำหนด
     โดยโปรแกรมให้มีการรับค่า เพียงตัวเลข 4 หลัก
   - ตัว (set value) มีการกำหนด
     โดยโปรแกรมให้มีการรับค่า เพียงตัวเลข 4 หลัก
5. เมื่อกดคำสั่งตามที่ต้องการแล้ว ก็กด ปุ่ม WR ปิดท้าย เช่น เมื่อกดตามลำดับจากข้อ 1 ถึง 4 แล้ว ก็กดปุ่ม WR ต่อ จะมีการแสดงหน้าจอว่า < O.K. > แล้วจะขึ้นบรรทัดใหม่ เพื่อรอการโปรแกรม

ส่วนการเปลี่ยนโหมด การแสดงมีให้ เลือก 3 แบบ คือ

- PROGRAM MODE
- MONITOR MODE
- RUN MODE

ที่ได้กล่าวไว้ข้างต้น เป็นการเลือกใน PROGRAM MODE ส่วน 2 แบบที่เหลือ เป็นการส่งค่าใน PROGRAM MODE ไปให้หน่วยประมวลผลนำค่าไปกระทำ ส่วนข้อแตกต่างของทั้ง 2 แบบ ก็คือ ใน MONITOR MODE จะเป็นการแสดงสภาวะของการกระทำ ณ ขณะนั้น ๆ เป็นขั้น ๆ ไป แต่ใน RUN MODE จะไม่มีการแสดงสภาวะออกมาให้ทราบ

ลักษณะ KEY BOARD มี 40 keys ซึ่งแบ่งออกเป็นส่วน ๆ ได้ 3 ส่วน คือ

1. NUMERIC KEYS
2. INSTRUCTION KEYS
3. OPERATION KEYS

ลักษณะ KEY BOARD ทั้ง 40 KEY มีดังนี้

C	D	E	F	FUN	NOT	F1	F2	DEL	JMP
8	9	A	B	OR	AND	F3	F4	INS	INC
4	5	6	7	LD	OUT	F5	F6	WR	DEC
0	1	2	3	TIM	CNT	cont	set	reset	

NUMERIC KEYS

C	D	E	F
8	9	A	B
4	5	6	7
0	1	2	3

เป็นปุ่มกด ตัวเลข 0 - F (เลขฐานสิบหก)

INSTRUCTION KEYS

FUN	NOT	F1	F2
OR	AND	F3	F4
LD	OUT	F5	F6
TIM	CNT		

ปุ่ม FUN, NOT, OR, AND, LD, OUT, TIM, CNT เป็นไปตามลักษณะที่กล่าวข้างต้น

ส่วนปุ่ม F1, F2, F3, F4, F5, F6 เป็นปุ่มที่มีการเซ็ทไว้ให้ทำหน้าที่ที่ตั้งขึ้นเฉพาะของผู้ใช้ เช่น F1 เป็นปุ่มเลือก mode

OPERATION KEYS

DEL	JMP
INS	INC
WR	DEC

ปุ่มพวกนี้ เป็นปุ่มช่วยในการ โปรแกรม เช่น

JMP เป็นการกระโดดไปยัง บรรทัดที่ต้องการ

INC เป็นการเพิ่มค่าบรรทัดที่แสดงอยู่ขึ้นอีก 1 บรรทัด

DEC เป็นการลด ค่าบรรทัดที่แสดงอยู่ลงอีก 1 บรรทัด

DEL เป็นการลบค่าที่แสดงอยู่

INS เป็นการแทรกค่าที่แสดงอยู่

WR เป็นการเขียนค่าที่แสดงอยู่ใน บรรทัดนั้น ๆ ลงในหน่วยความจำ

ในการประยุกต์การใช้งาน ในบทนี้ มีตัวอย่างประกอบด้วยกัน 2 ตัวอย่าง คือในรูปที่ 6.1 และ รูปที่ 6.2 แต่จะยกตัวอย่างในรูปที่ 6.1 เพื่อประกอบการอธิบาย พร้อมหลักในการป้อนโปรแกรมที่หน่วยป้อนโปรแกรม

ตัวอย่างที่ 1 ในรูปที่ 6.1 เป็นการจำลอง ร้านบริการล้างรถที่ควบคุมกลไกด้วย PLC จากรูป

มีการกำหนด หน่วยอินพุต ออกเป็น 3 หน่วย คือ

1. ปุ่มกดเดินเครื่อง
2. เซนเซอร์ตรวจจับวัตถุ (ในที่นี้ คือ รถยนต์)ว่า ได้มีการเคลื่อนตัวจากจุดไหน ถึงจุดไหน
3. ปุ่มกดเพื่อการหยุดเครื่อง

ส่วน หน่วยเอาต์พุต มี 3 ลักษณะด้วยกัน คือ

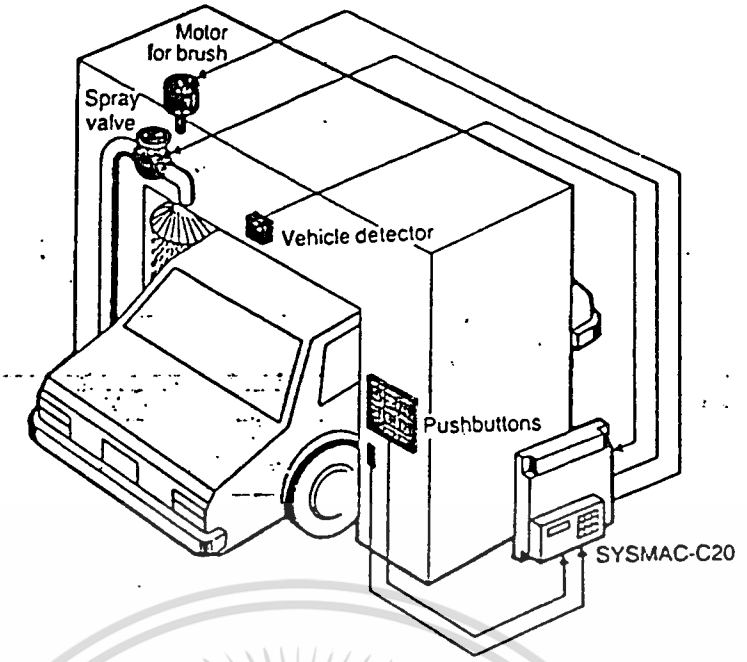
1. วาล์วสเปรย์เพื่อฉีดสาร
2. มอเตอร์ใช้ในการล้าง
3. มอเตอร์ใช้ในการเคลื่อนที่ของเครื่องล้าง

เมื่อกำหนดได้แล้ว เราจะต้องมาใช้ในการกำหนดในโปรแกรม PLC ว่า ได้กำหนดให้

- หน่วยรีเลย์อินพุต มีค่าแอสเดรส จาก 0300 - 030FH
- หน่วยรีเลย์เอาต์พุต มีค่าแอสเดรส จาก 0400 - 040FH
- หน่วยรีเลย์ออกซิลารี่ มีค่าแอสเดรส จาก 0900 - 090FH

เมื่อได้แล้ว สมมุติเรากำหนด หน่วยอินพุต และหน่วยเอาต์พุต ได้ตามตารางในรูป จากนั้นจึงมาเขียนการทำงานเป็นแบบ Ladder diagram แล้วใช้ Ladder diagram มาเขียนโปรแกรมด้วย ภาษาบูลลีน ดังในรูป แล้วป้อนโปรแกรมตามภาษาบูลลีน ผ่านทางหน่วยป้อนโปรแกรมตามลำดับ

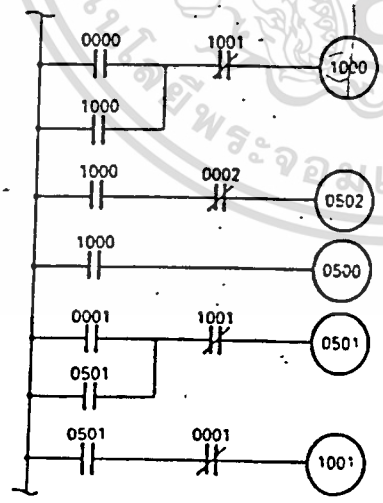
ส่วนการประยุกต์ใช้งานในด้านอื่น ๆ ก็เป็นการแทนวงจรรีเลย์ หรือ การ ON/OFF สวิทช์ที่สามารถนำมาใช้งานแทนวงจรควบคุมเหล่านั้นได้ด้วย เช่นในรูปที่ 6.2 ส่วนการจะเพิ่มเติมระบบ หรือ ขยายขีดความสามารถของระบบ ก็โดยการ design หน่วยต่าง ๆ ขึ้นในขอบเขตที่เป็นไปได้ ดังที่ได้แสดงหน่วยอินพุต/เอาต์พุต ในบทที่ 2 นำมาทำเป็นโมดูล ซึ่งเชื่อมต่อด้วยระบบบัส



I/O assignment

Input	Relay
Start button Vehicle detector Condition at which washing machine stops	0000 0001 0002
Output	
Spray valve Brush motor Movement of washing machine	0500 0501 0502

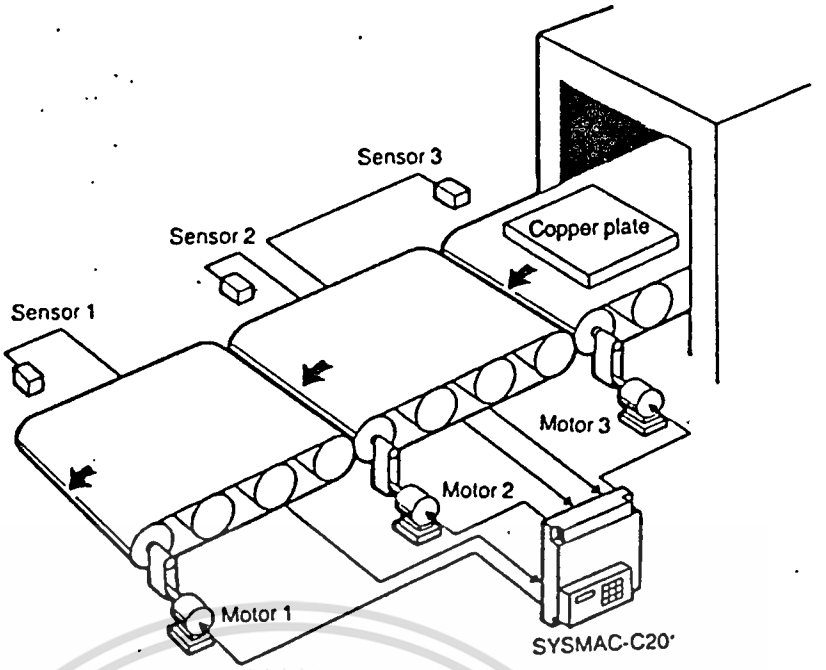
Ladder diagram



Coding Chart

Address	Instruction	Data
0000	LD	0000
0001	OR	1000
0002	AND-NOT	1001
0003	OUT	1000
0004	LD	1000
0005	AND-NOT	0002
0006	OUT	0502
0007	LD	1000
0008	OUT	0500
0009	LD	0001
0010	OR	0501
0011	AND-NOT	1001
0012	OUT	0501
0013	LD	0501
0014	AND-NOT	0001
0015	OUT	1001

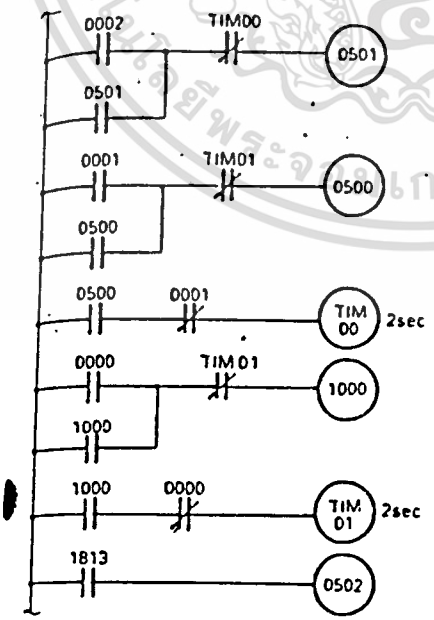
รูปที่ 6.1



I/O assignment

Input	Relay
Sensor 1 Sensor 2 Sensor 3	0000 0001 0002
Output	
Motor 1 Motor 2 Motor 3	0500 0501 0502

Ladder diagram



Coding chart

Address	Instruction	Data
0200	LD	0002
0201	OR	0501
0202	AND-NOT	TIM00
0203	OUT	0501
0204	LD	0001
0205	OR	0500
0206	AND-NOT	TIM01
0207	OUT	0500
0208	LD	0500
0209	AND-NOT	0001
0210	TIM	00
		#0020
0211	LD	0000
0212	OR	1000
0213	AND-NOT	TIM01
0214	OUT	1000
0215	LD	1000
0216	AND-NOT	0000
0217	TIM	01
		#0020
0218	LD	1813
0219	OUT	0502

รูปที่ 6.2

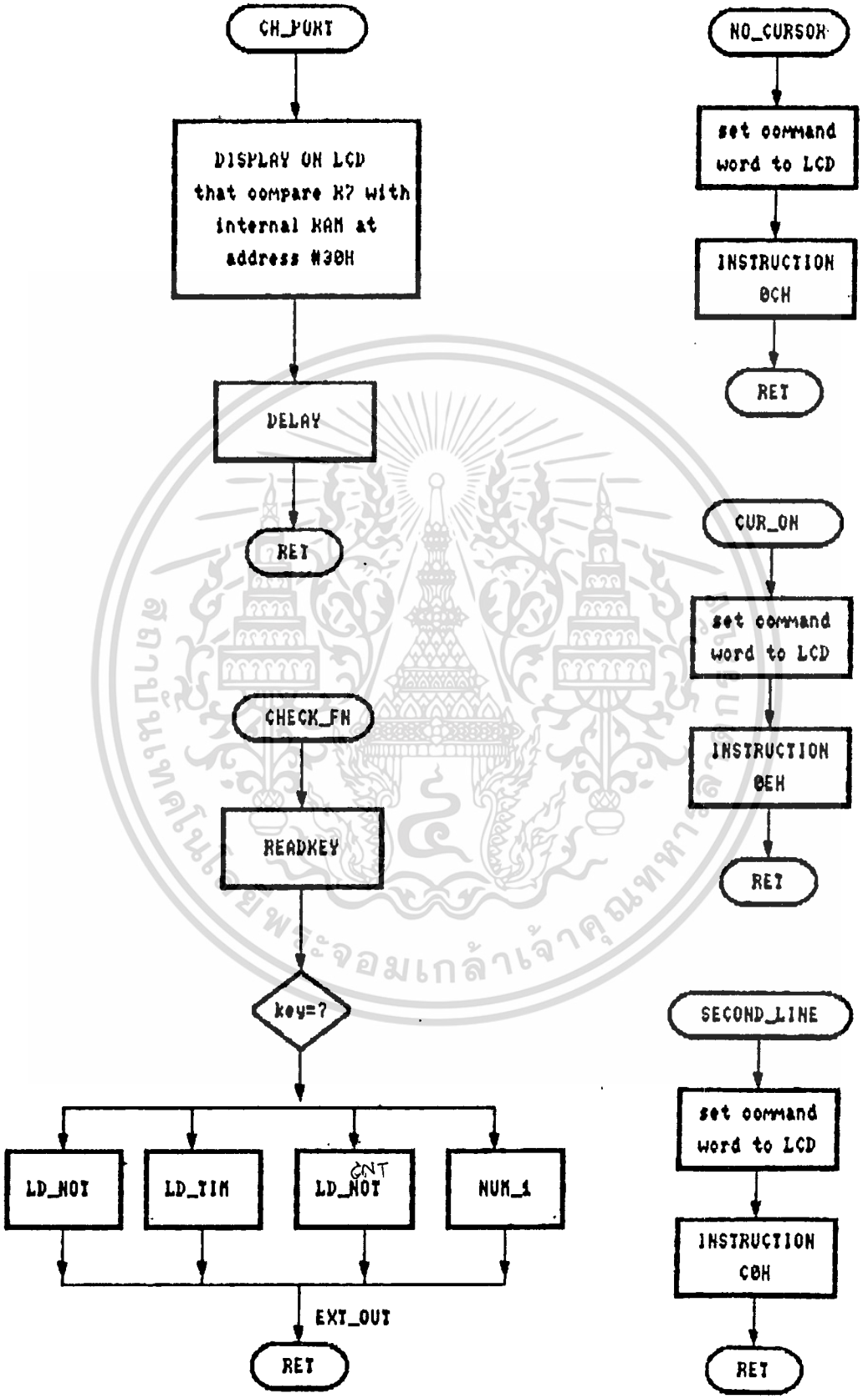
ปัญหาที่เกิดขึ้น

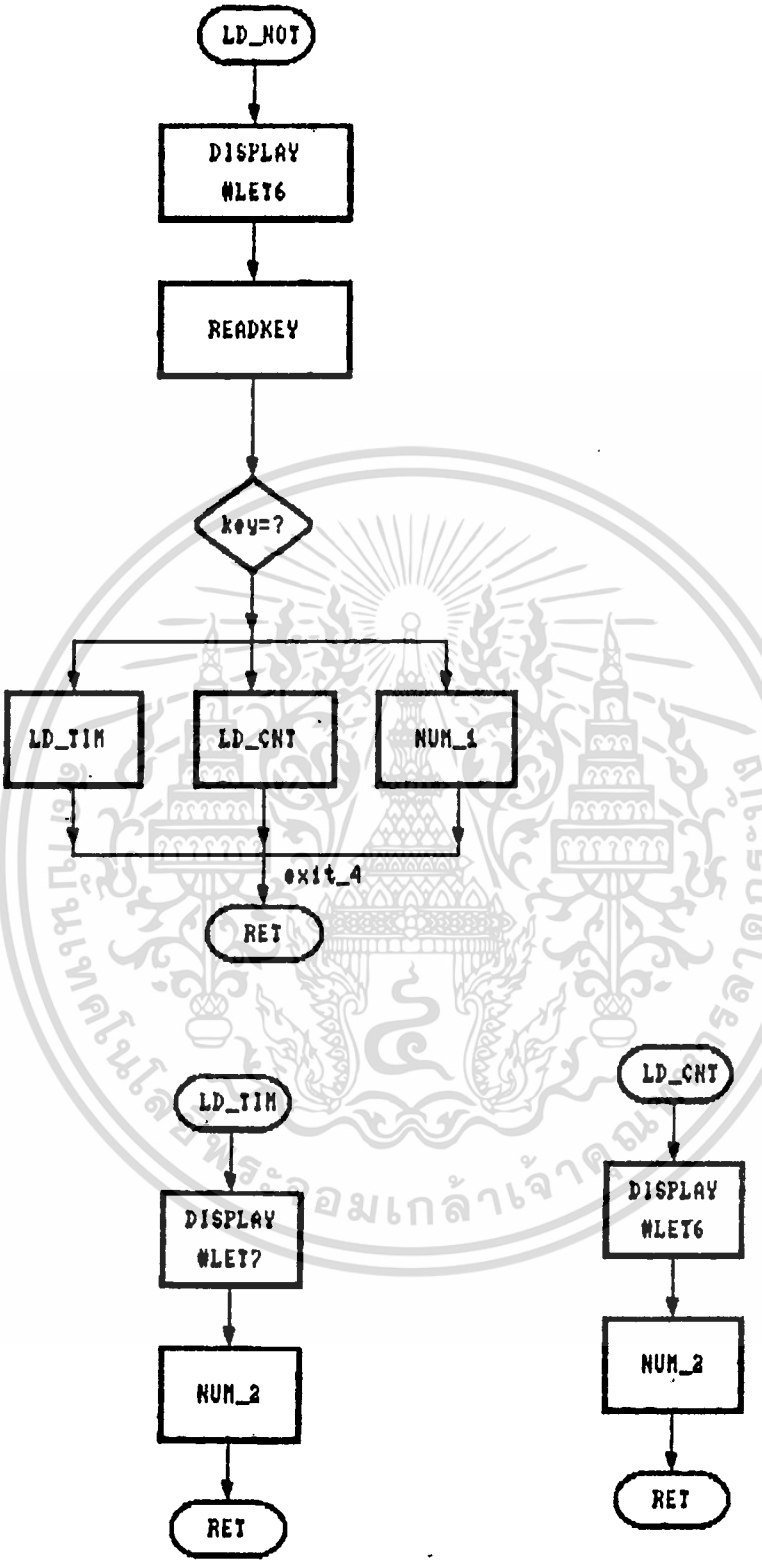
1. มีลายไฟขาดในลายสัญญาณต่างๆได้ ต้องระมัดระวัง
2. แผ่น prote board คุณภาพไม่ดี ทำให้ ระบบ Ground ไม่เสถียร (ซึ่งอาจเกิด Ground leep ได้)
3. การประสานงานระหว่างฝ่าย hardware และ software ขาดการต่อเนื่อง
4. เครื่องมือที่ช่วยอำนวยความสะดวก ยังไม่สามารถนำมาประยุกต์ใช้งานของตนได้
5. เครื่องมือ เช่น Scope มีข้อผิดพลาดมากโดยเฉพาะสายโพรบ เมื่อจับวัดสัญญาณด้วยสายโพรบเองอาจทำให้สัญญาณเกิดความเพี้ยนขึ้นได้ แต่อย่างไรก็ตามสัญญาณที่จับได้ ก็พอดูได้
6. เครื่องมือด้านตรวจจับสัญญาณของระบบบัส การชิงของสัญญาณ ในการออกแบบระบบ ซึ่งทำได้ยาก เพราะไม่มีเครื่องมือที่มีคุณภาพพอในงาน

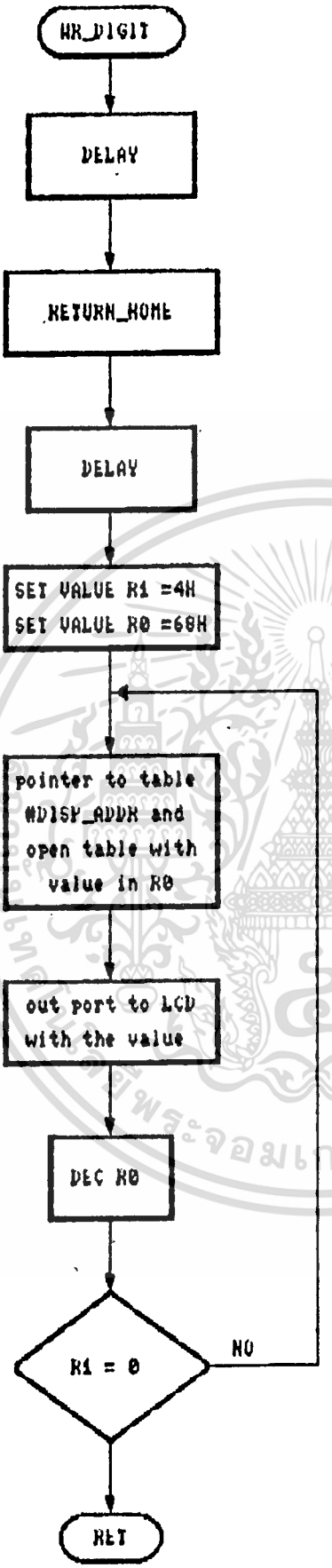
สรุป สำหรับโครงงานนี้ เป็นเพียงต้นแบบ ที่ต้องมีการพัฒนาในอีกหลายด้าน เช่น ในส่วนโปรแกรมนั้น มีขนาดใหญ่กว่าหน่วยความจำที่ออกแบบไว้ ทำให้ต้องลดขนาดของโปรแกรมลงซึ่งจะยุ่งยากมาก เพราะฉนั้นต้องออกแบบหน่วยความจำให้มากเผื่อไว้ ส่วนหน่วยอินพุต/เอาต์พุตที่ จะออกแบบเพิ่มเติมให้ดูรายละเอียดจากบทที่ 2 เป็นแนวทางได้ เพราะระบบ PLC ที่ดี จะมีการออกแบบตั้งแต่จาก การป้องกันสัญญาณรบกวนในระบบ และจากสภาวะแวดล้อม เช่น การลดคาร์ทมอเตอร์ในวงงาน จะมีสัญญาณความถี่สูงมารบกวนทำให้ระบบรวนได้ หรือ การรับค่าอินพุตที่ไม่ใช่สัญญาณทางลอจิก หรือมีความเร็วของสัญญาณสูงมาก ต้องมีหน่วยอินพุตเฉพาะ ขึ้น หรือ การต่อเป็นระบบเน็ตเวอร์ค ซึ่งในโครงงานนี้ได้ออกแบบ แยกออกเป็นโมดูล แต่ละส่วนเชื่อมต่อด้วยระบบบัส ทำให้ระบบอ่อนตัวดี

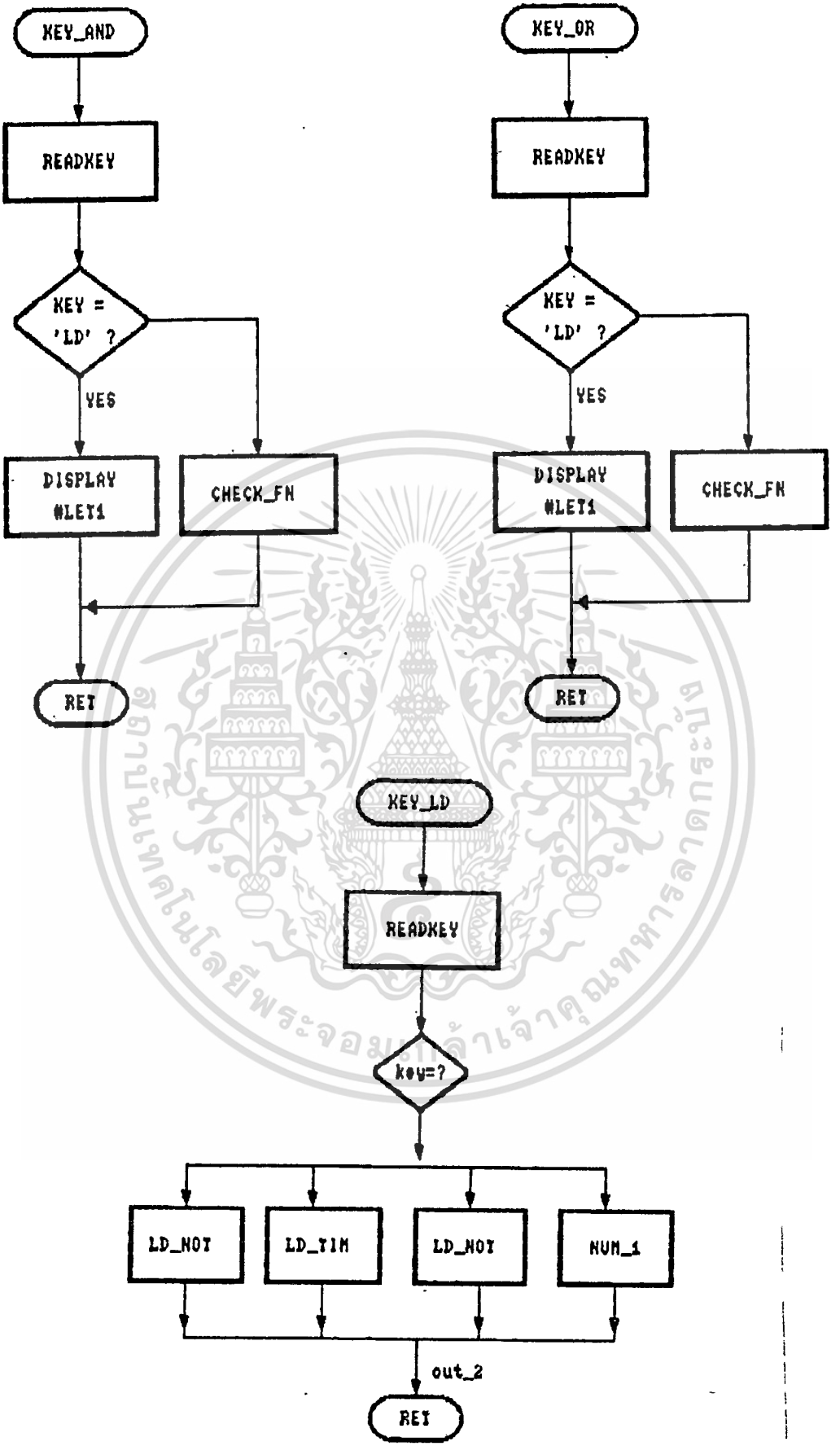
ภาคผนวก

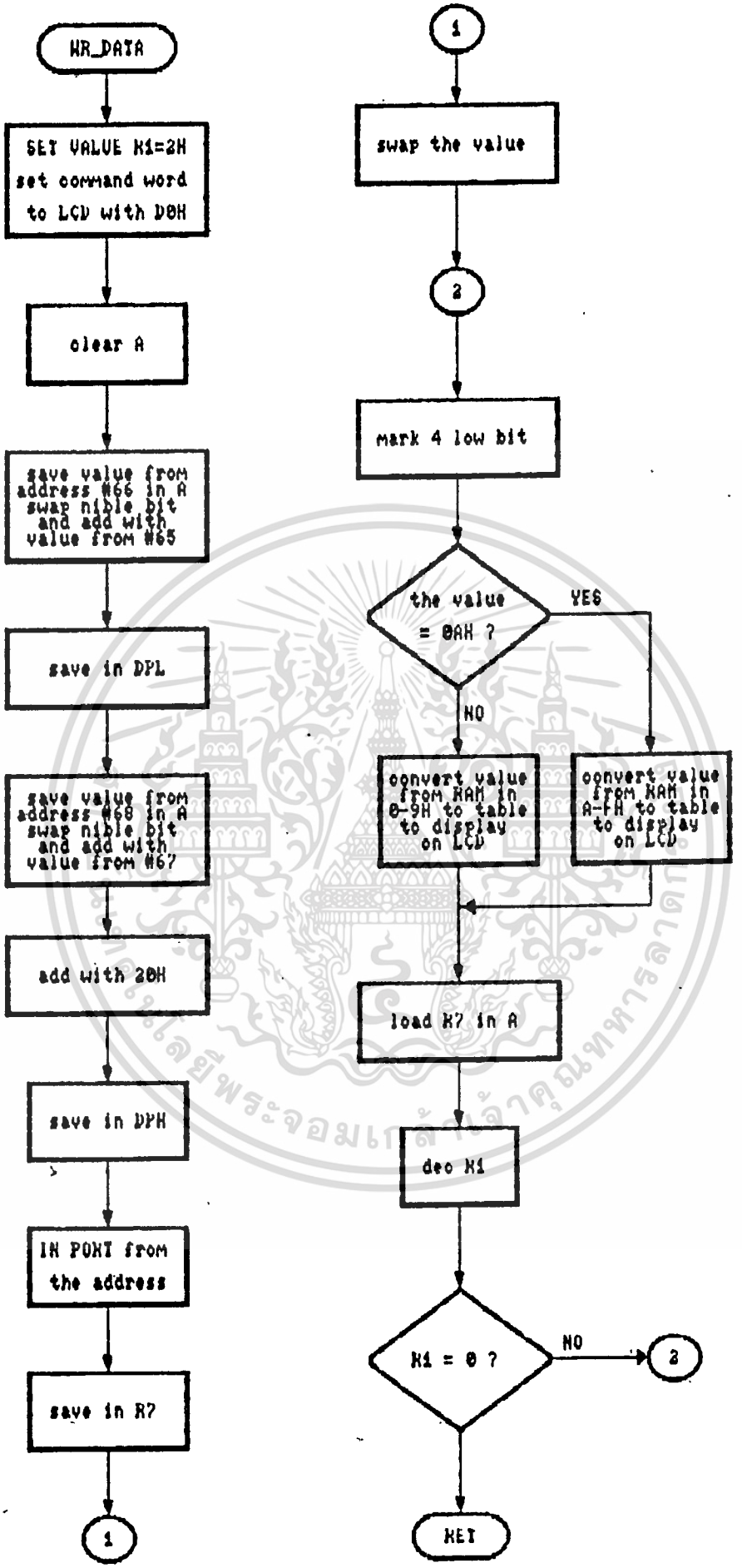
ภาคผนวก ก.
ฟลว์ชาร์ทของหน่วยย่อนโปรแกรม

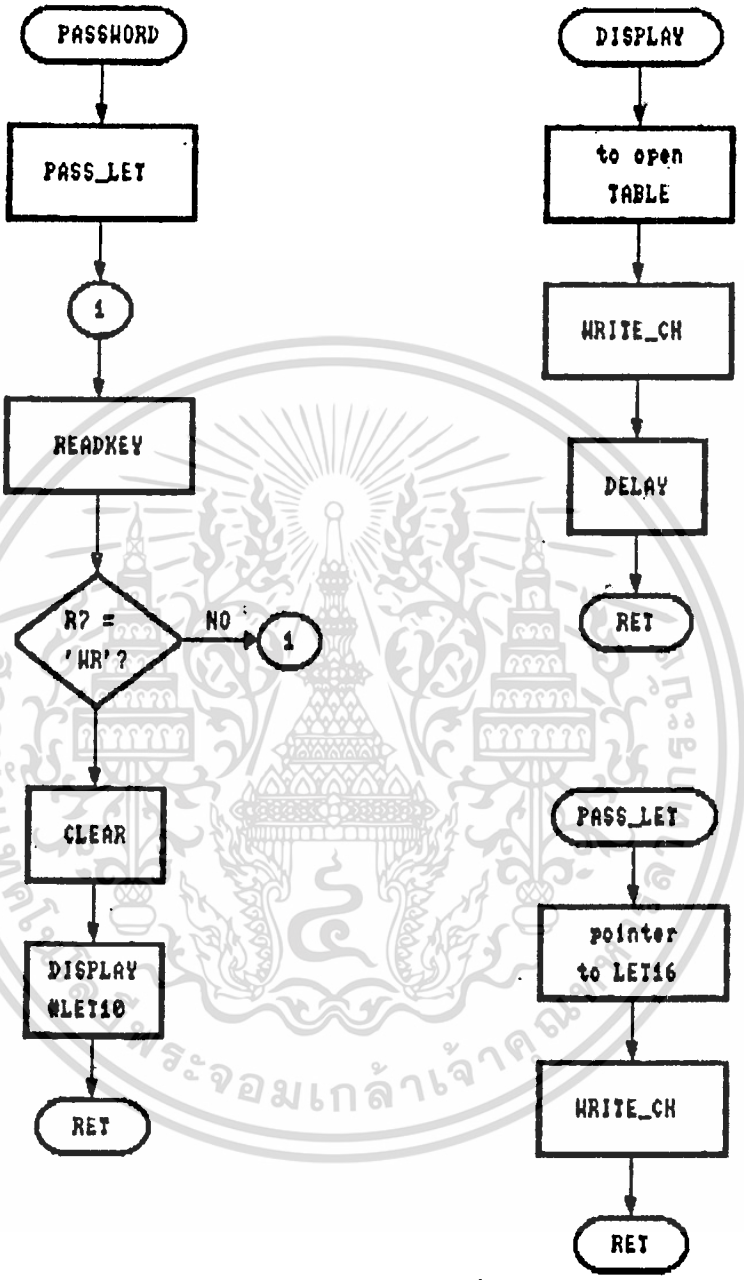


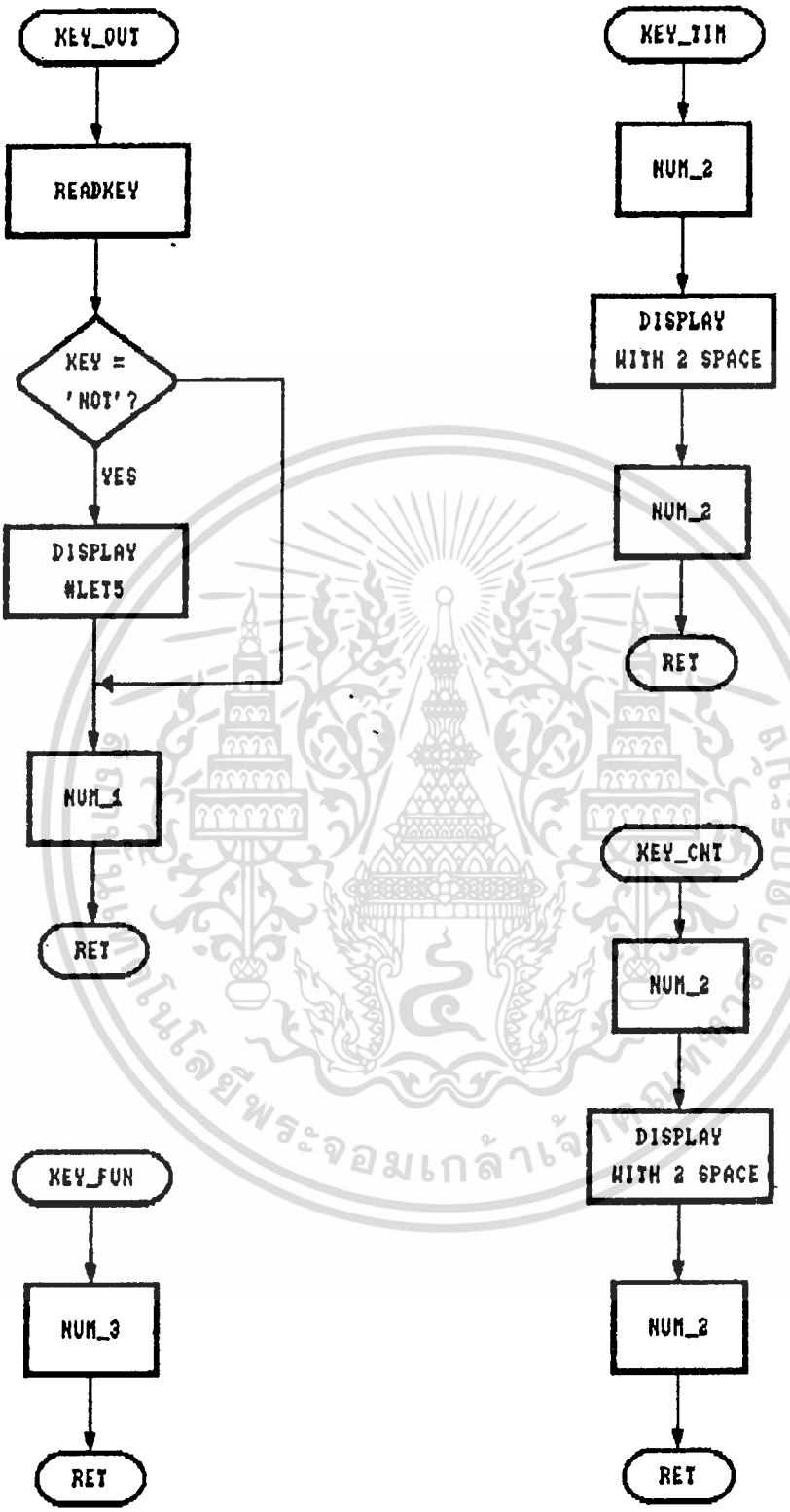


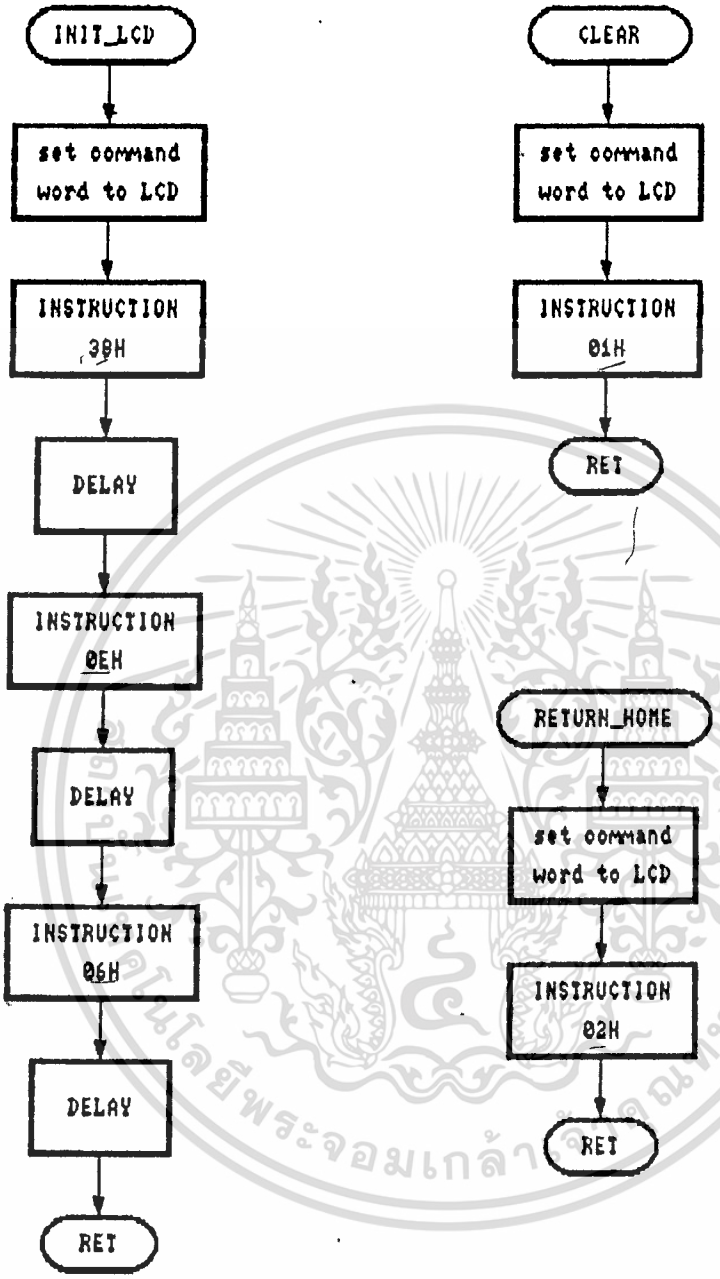


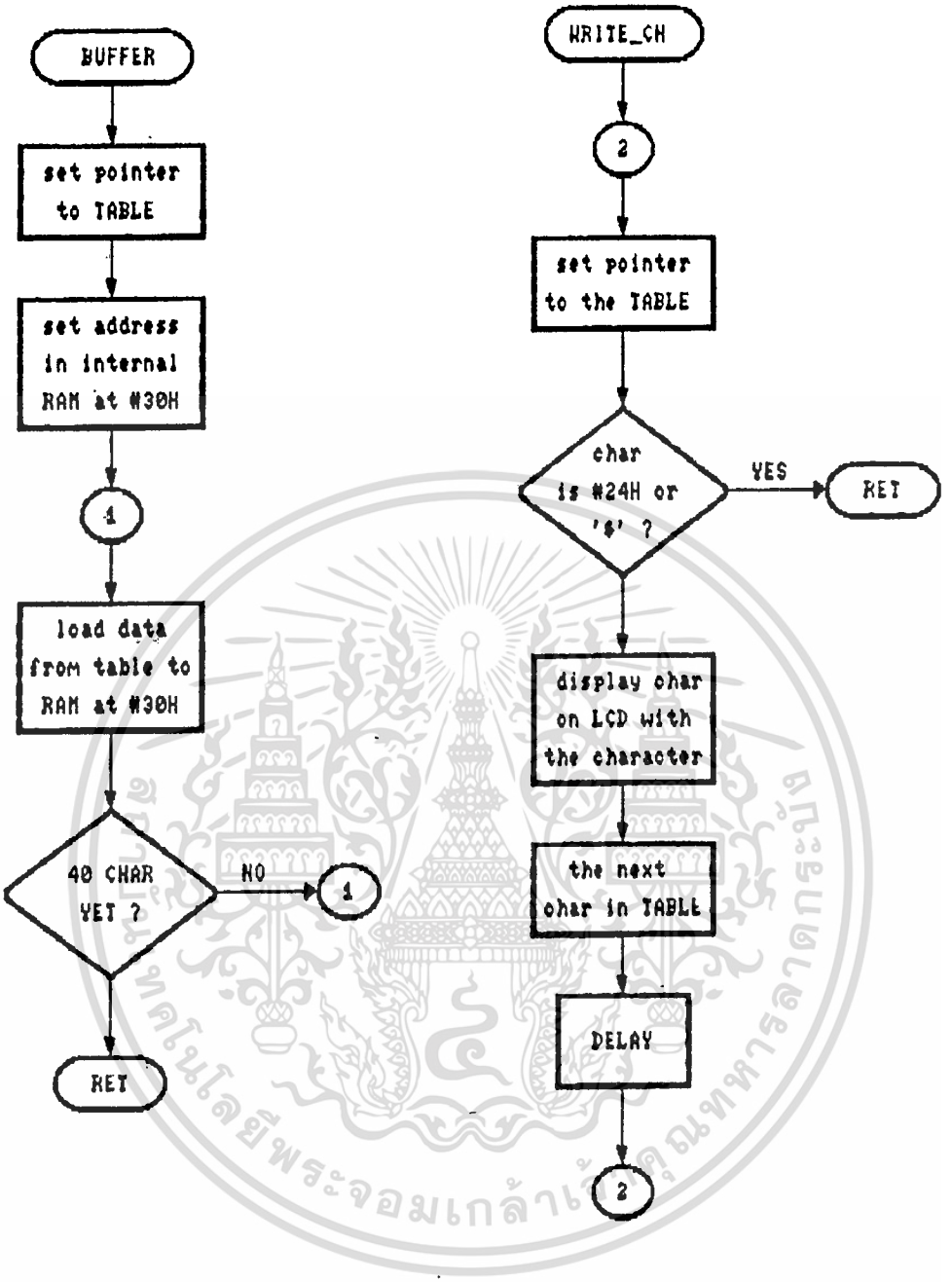


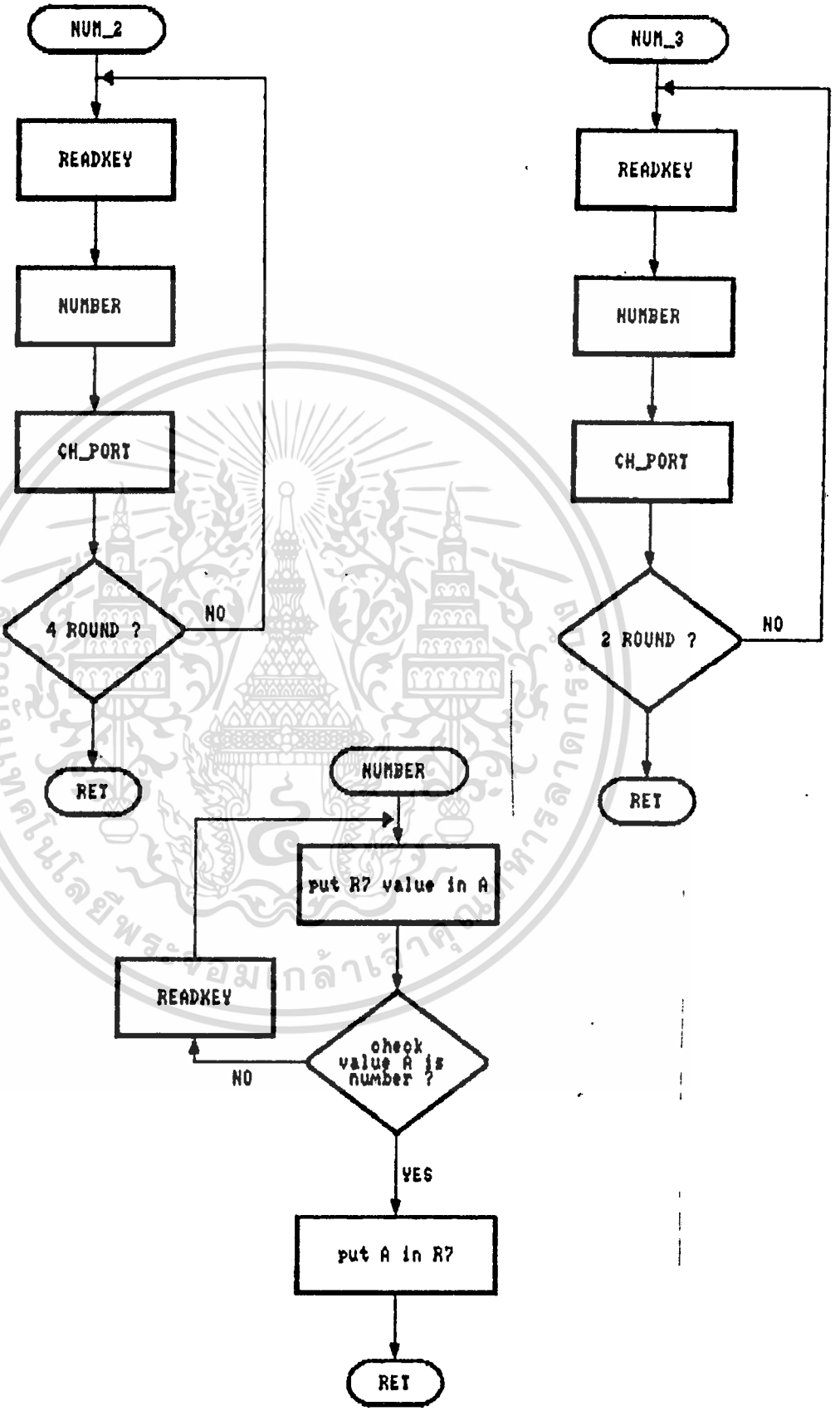


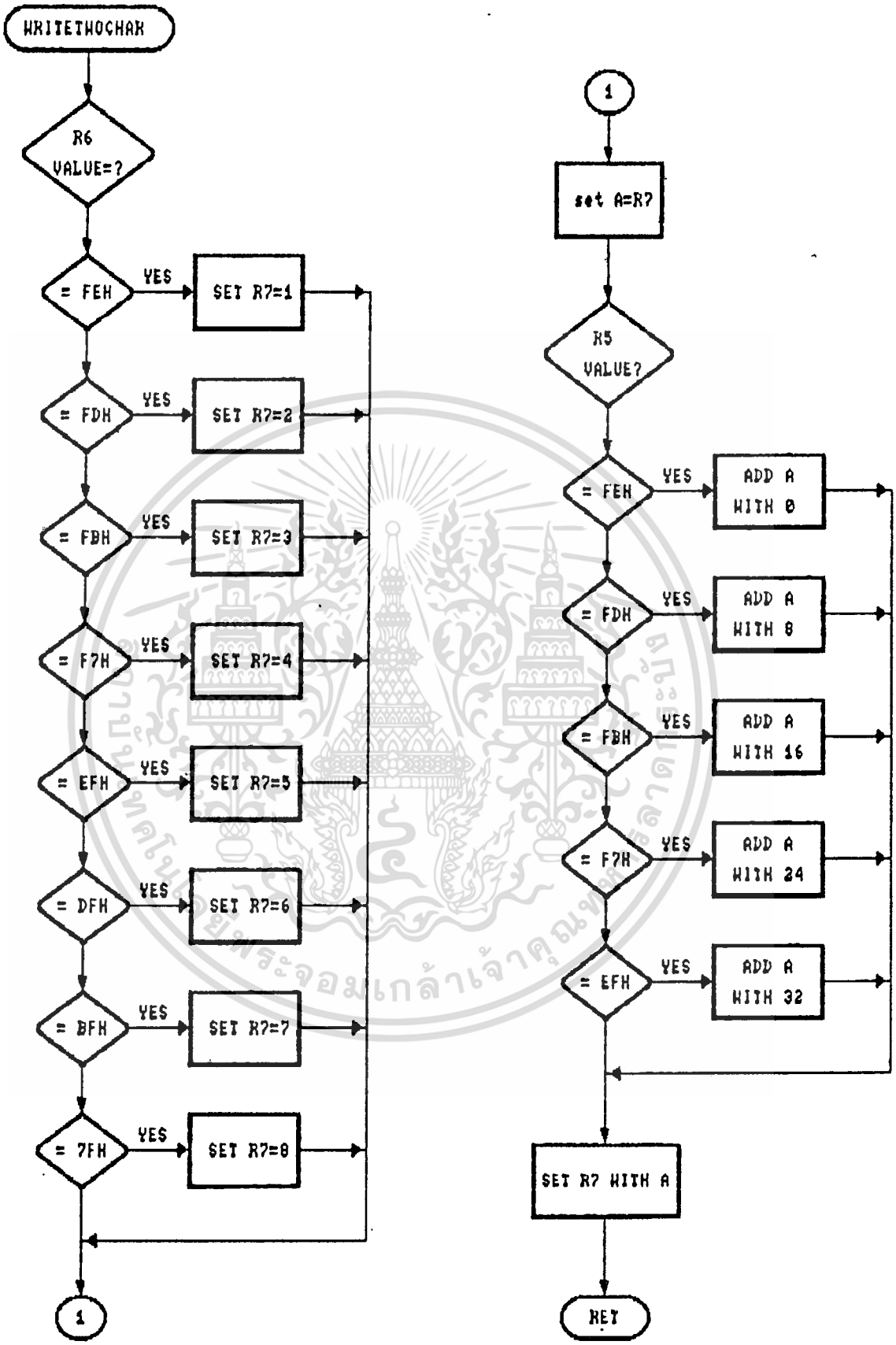


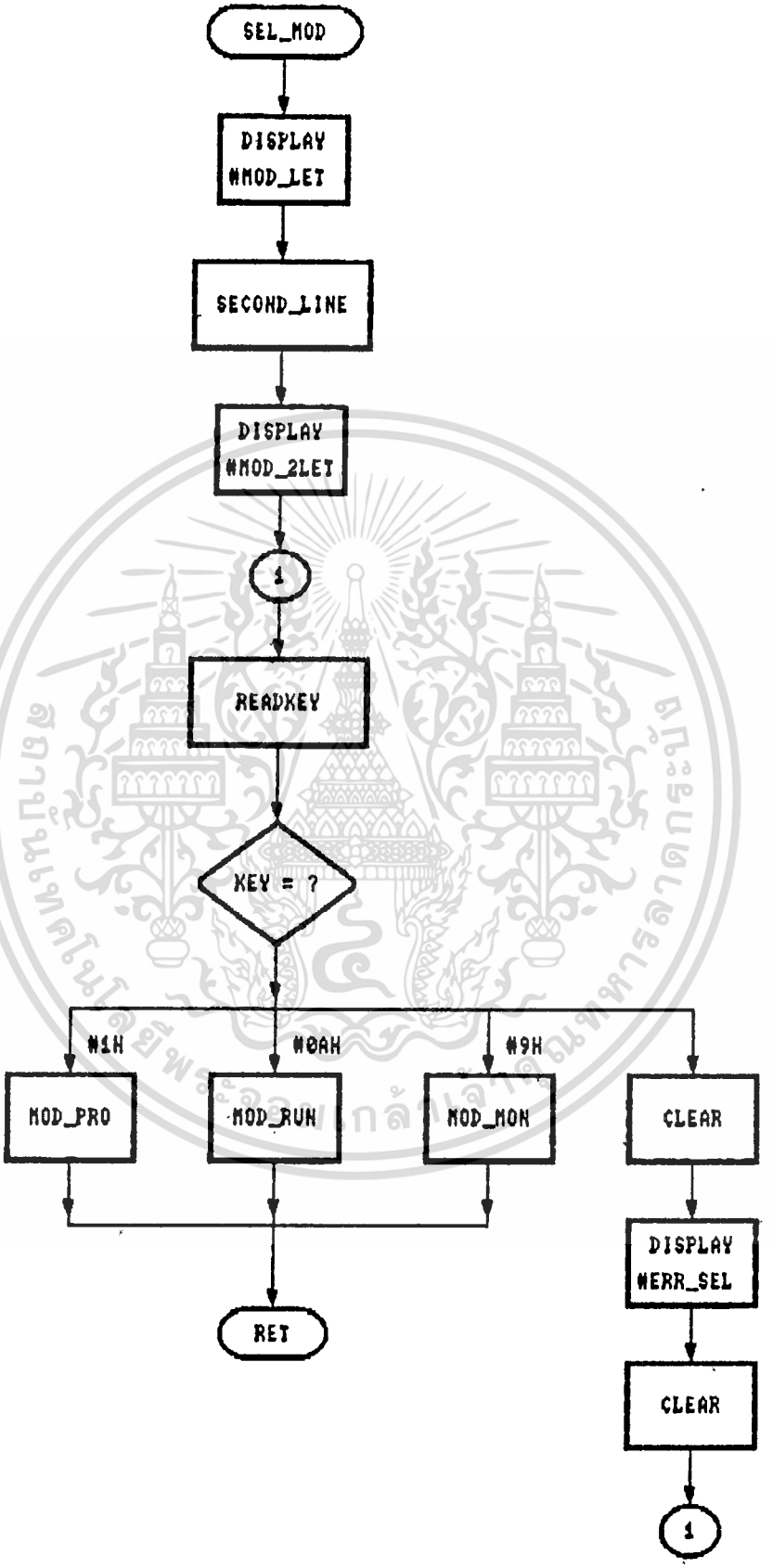


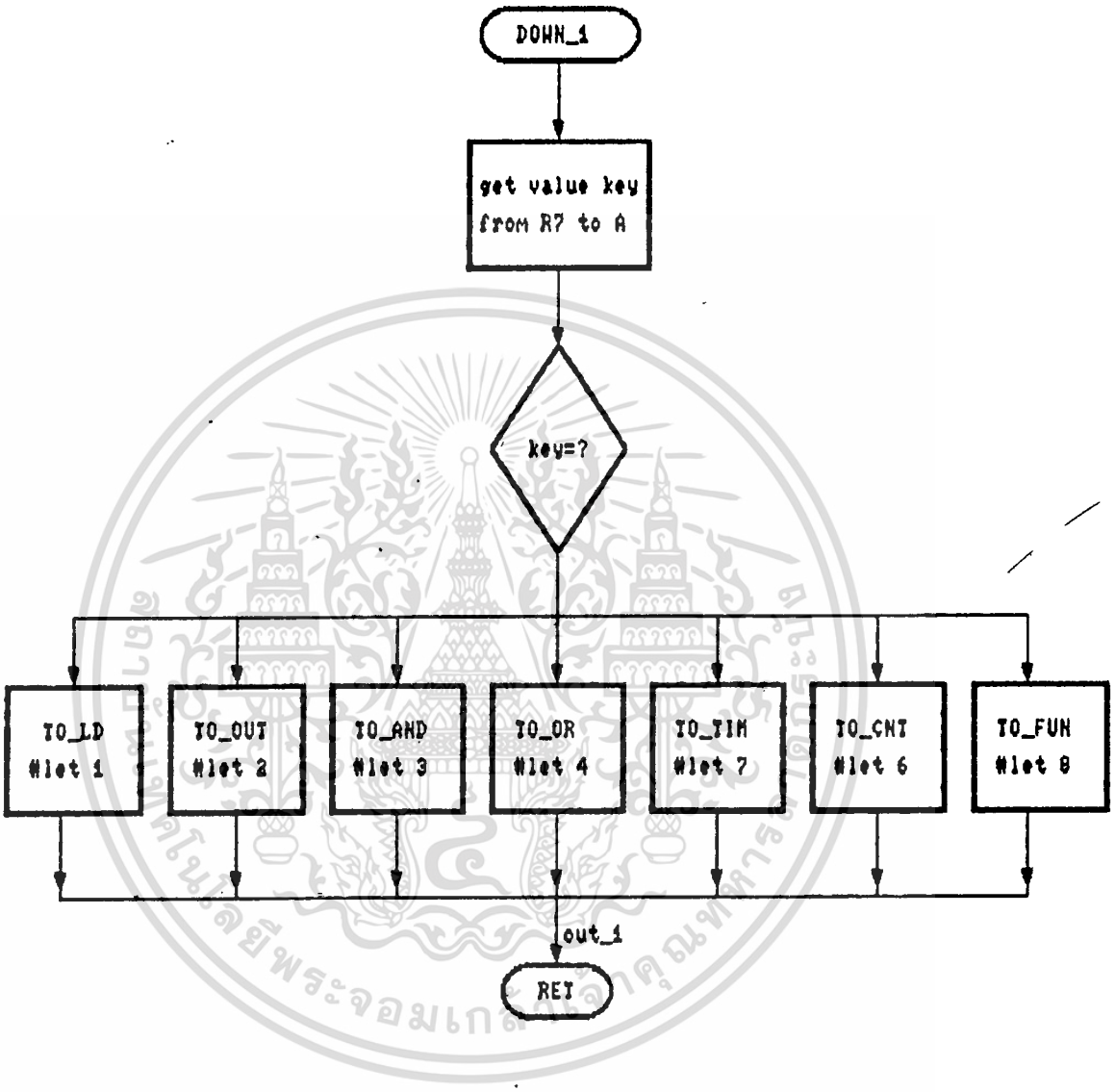


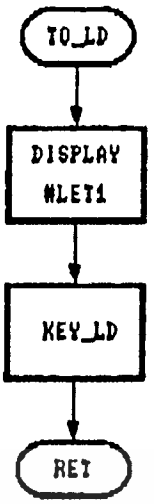


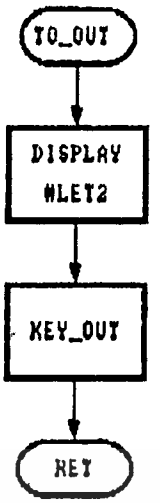


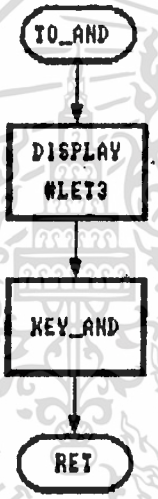


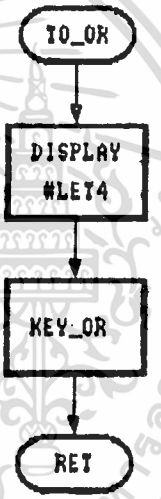


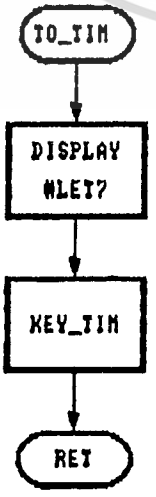


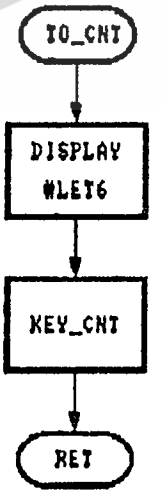


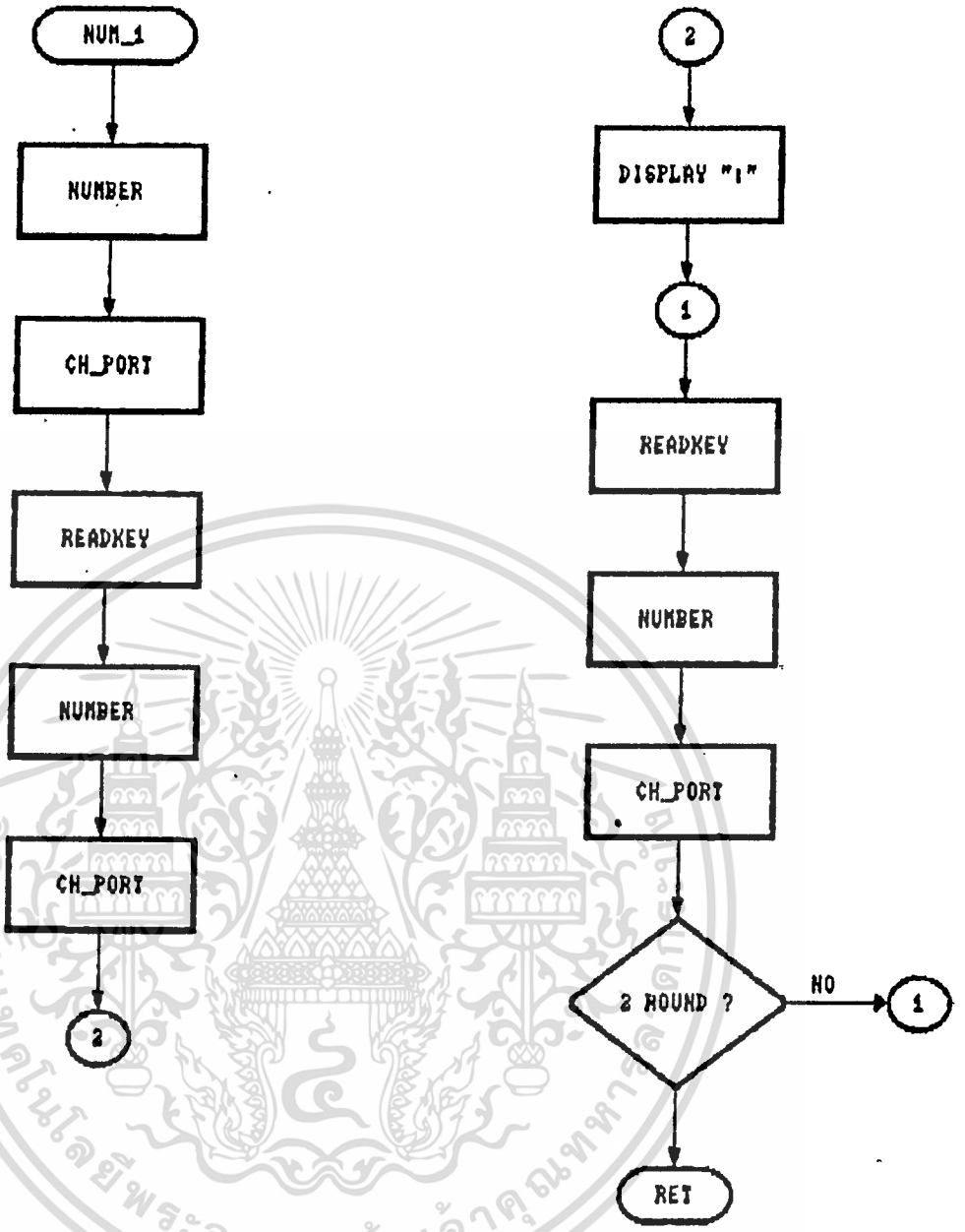








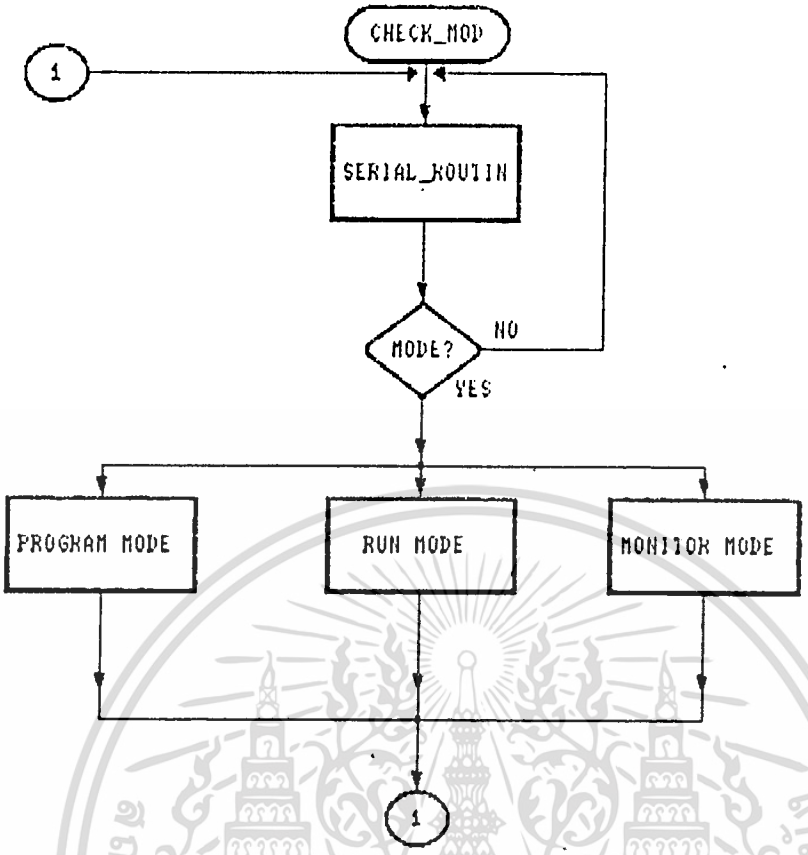


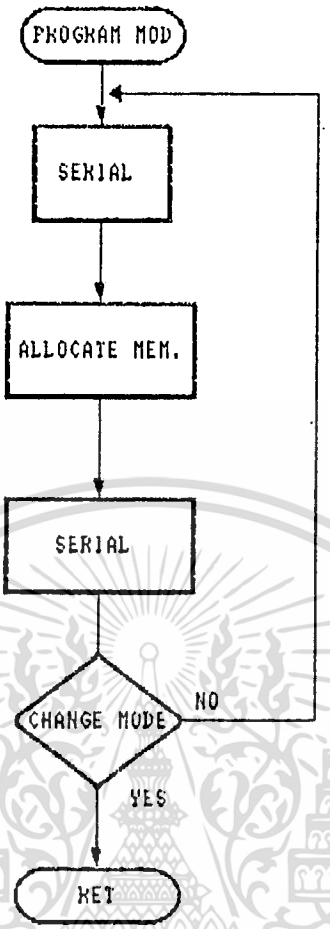


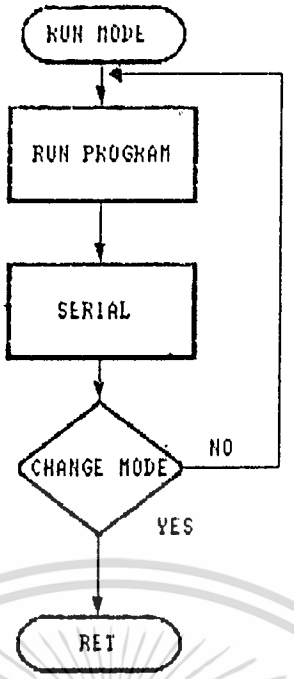
3

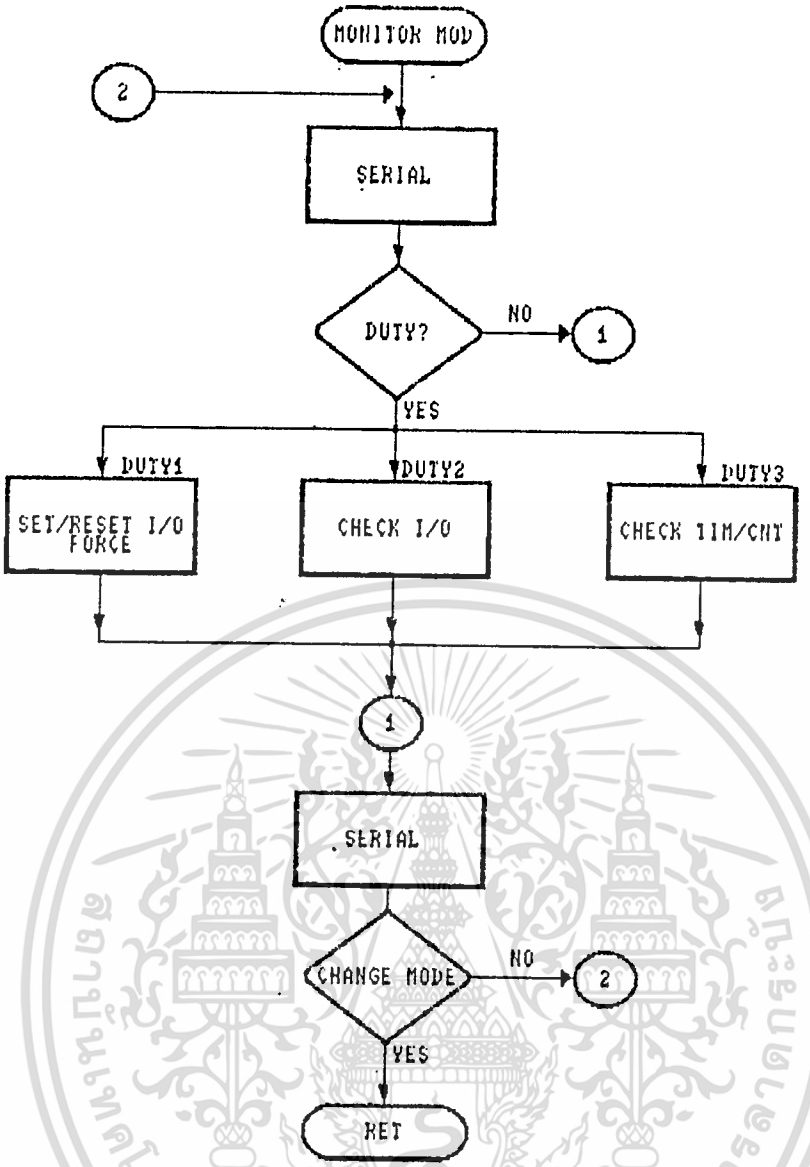
ภาคผนวก ข.

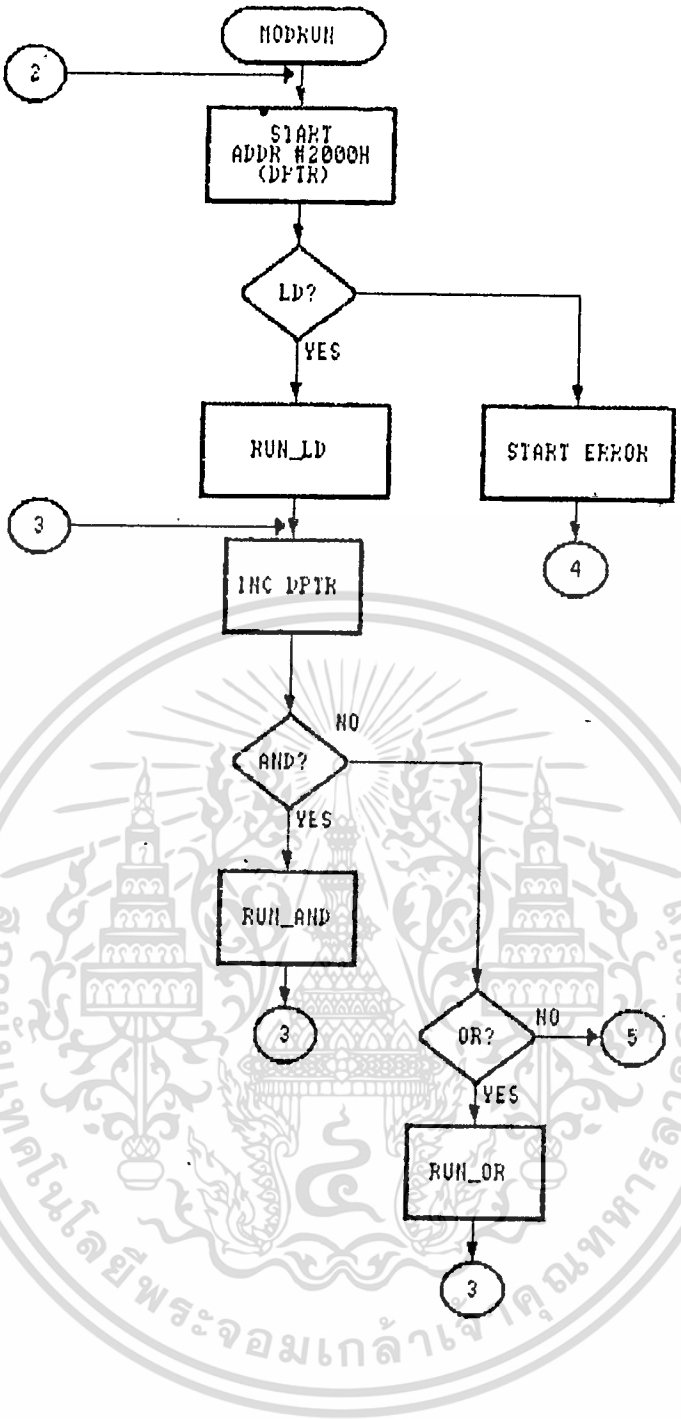
โฟล์วชาร์ทของหน่วยประมวลผลกลาง

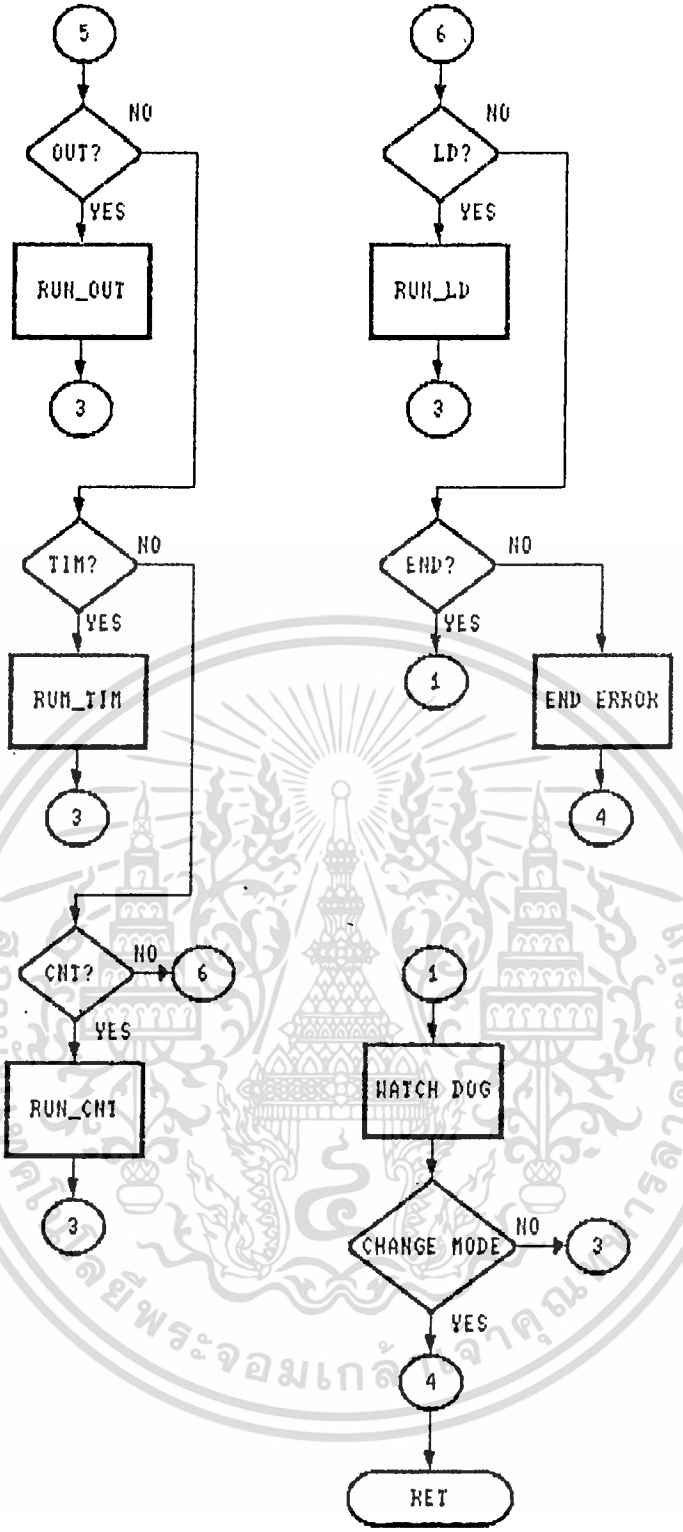


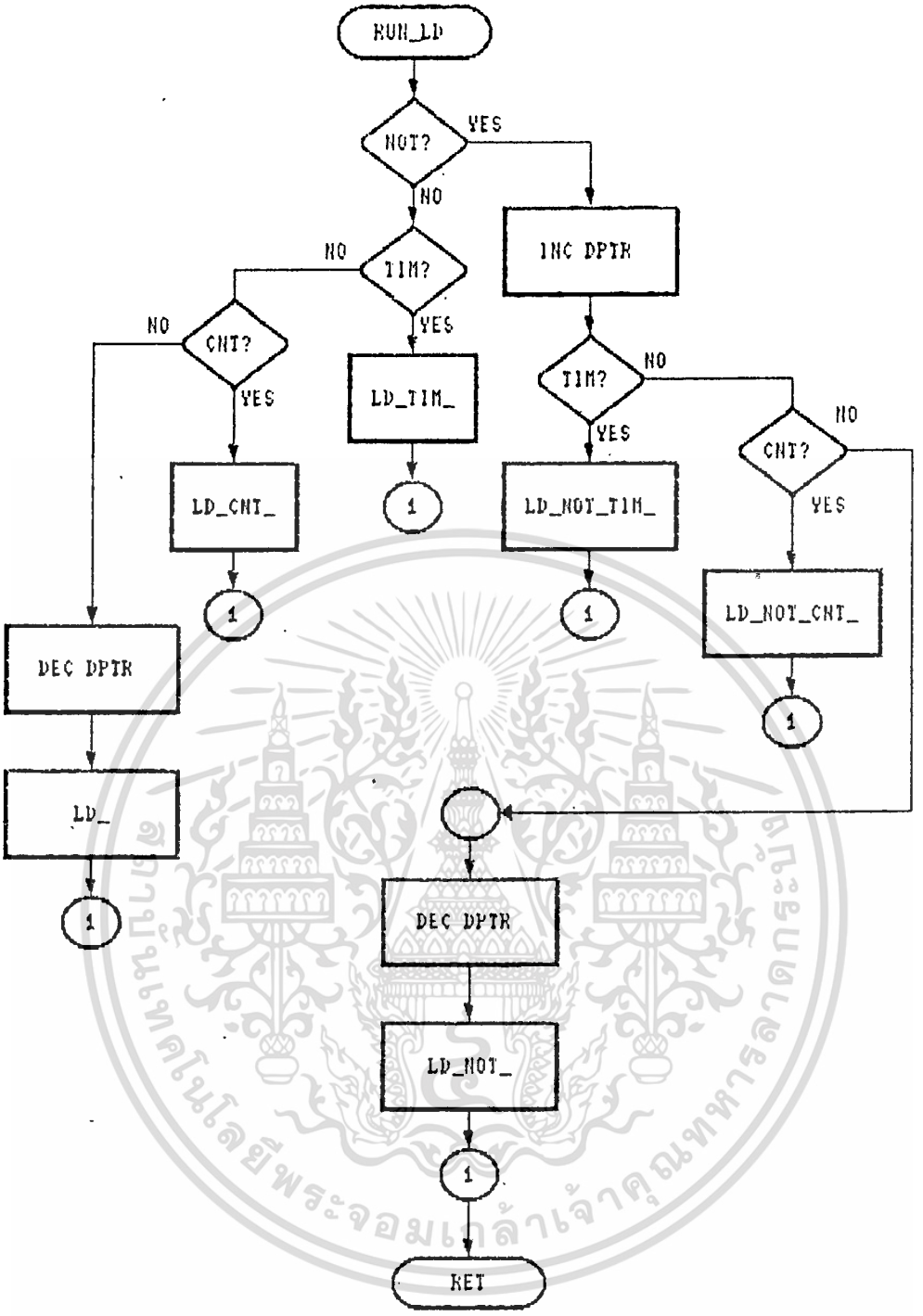


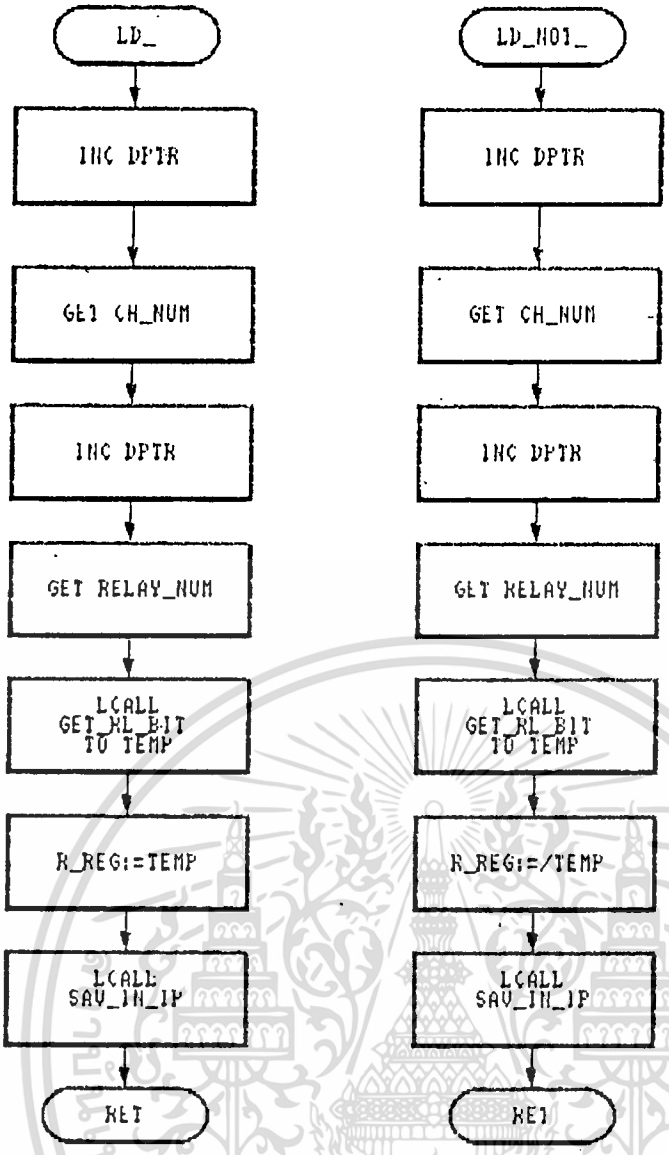


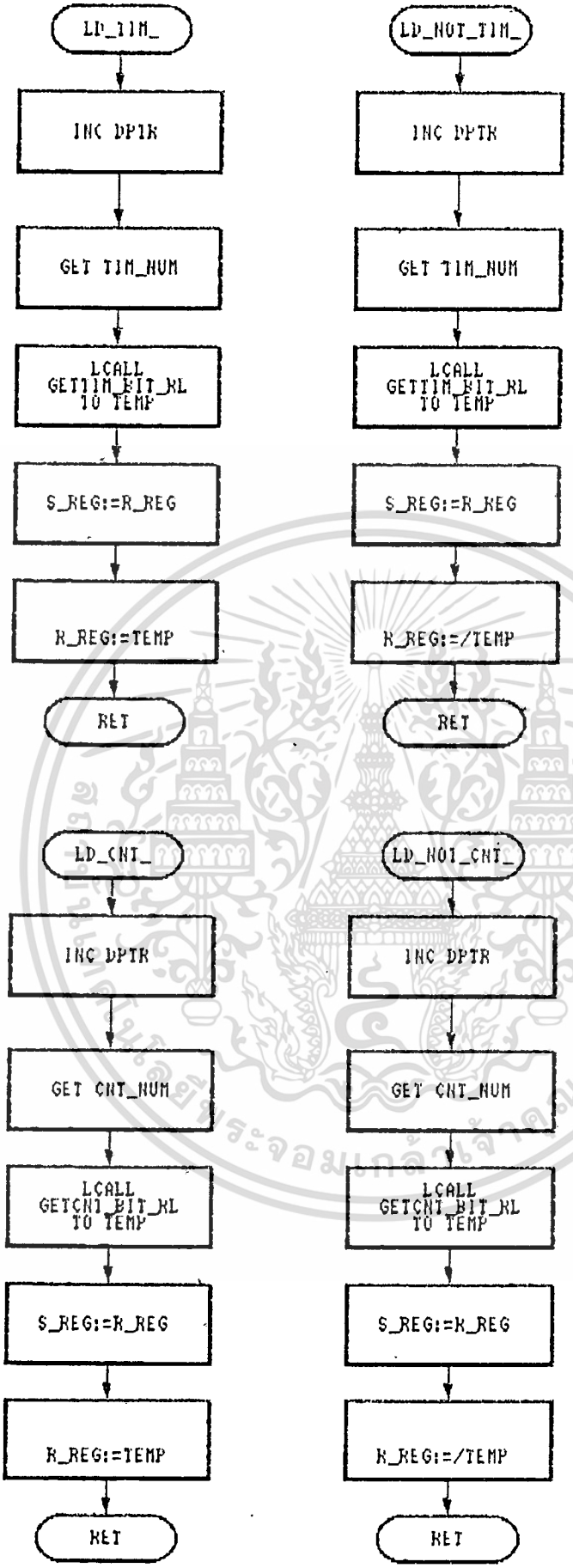


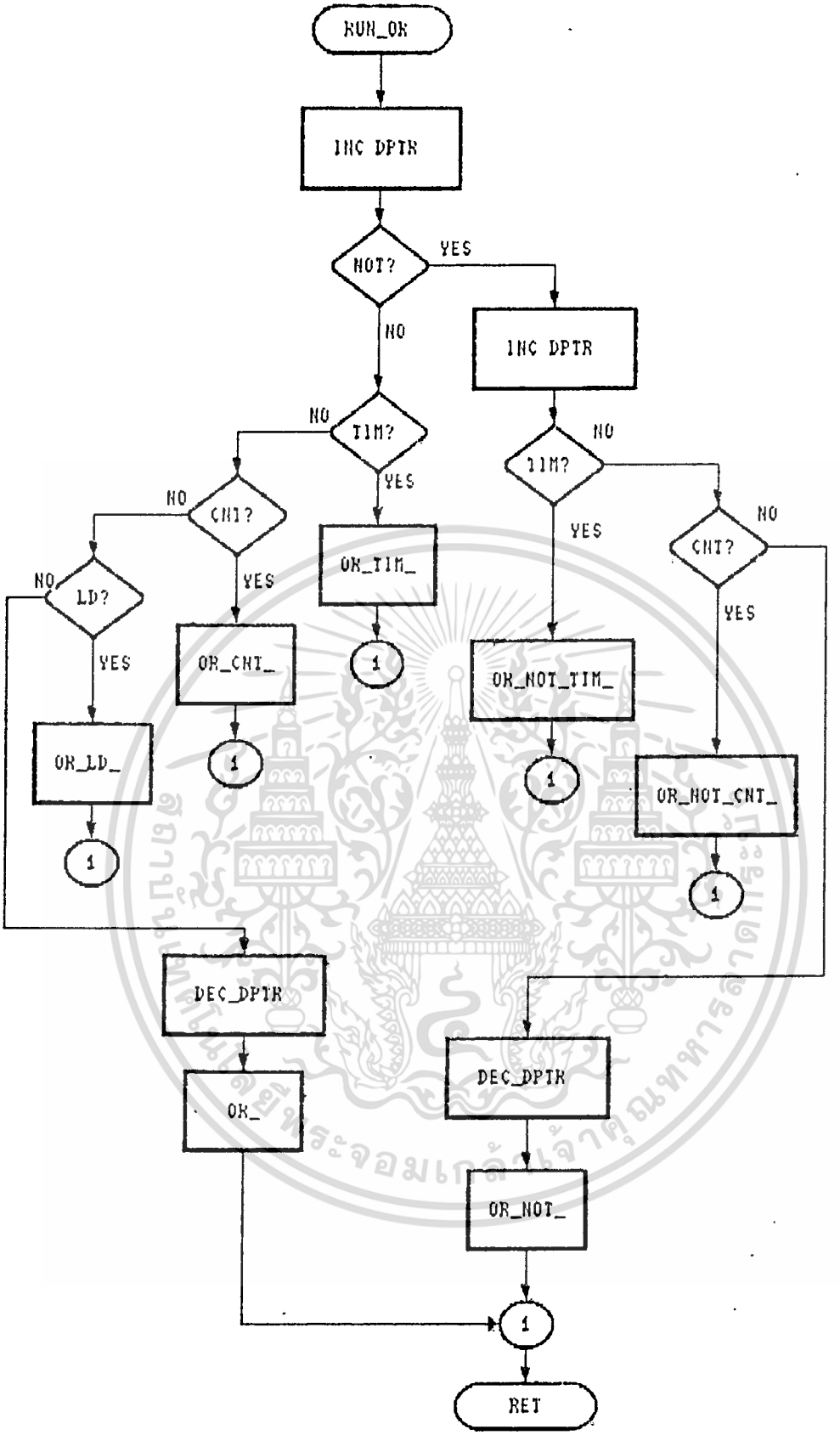


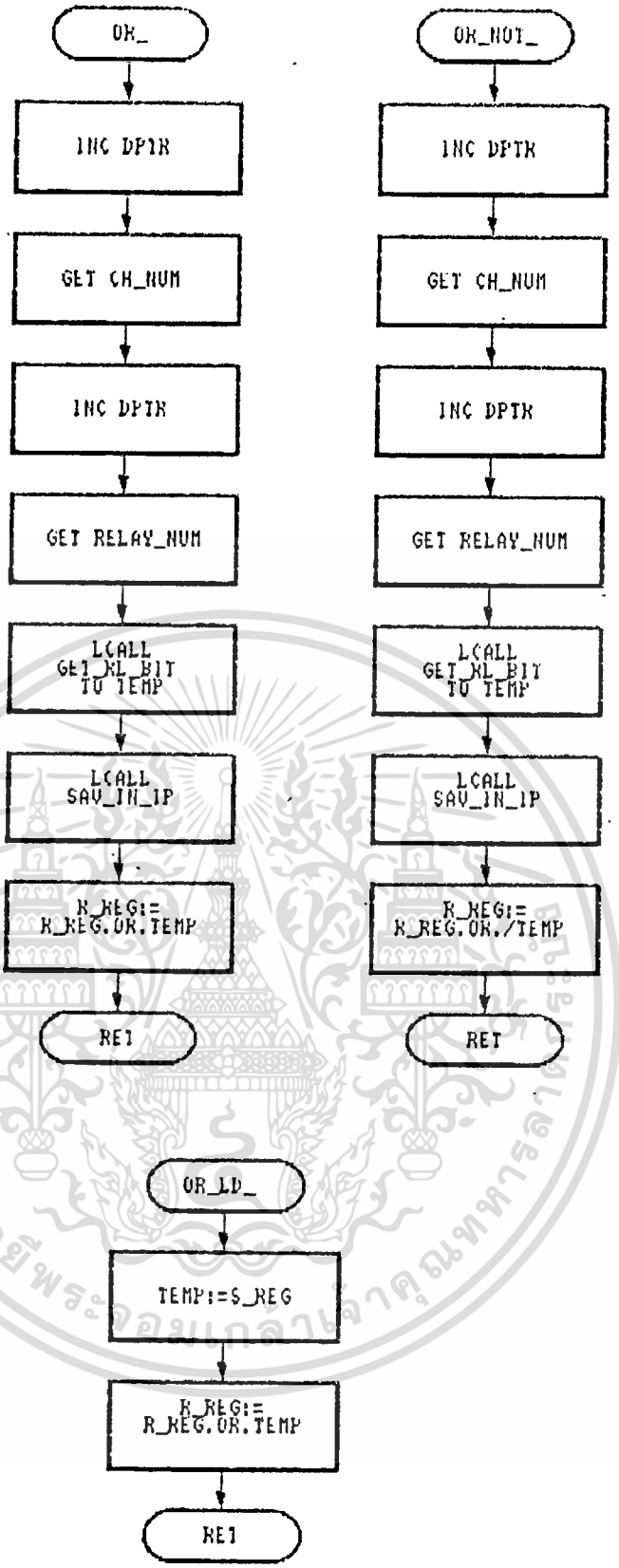


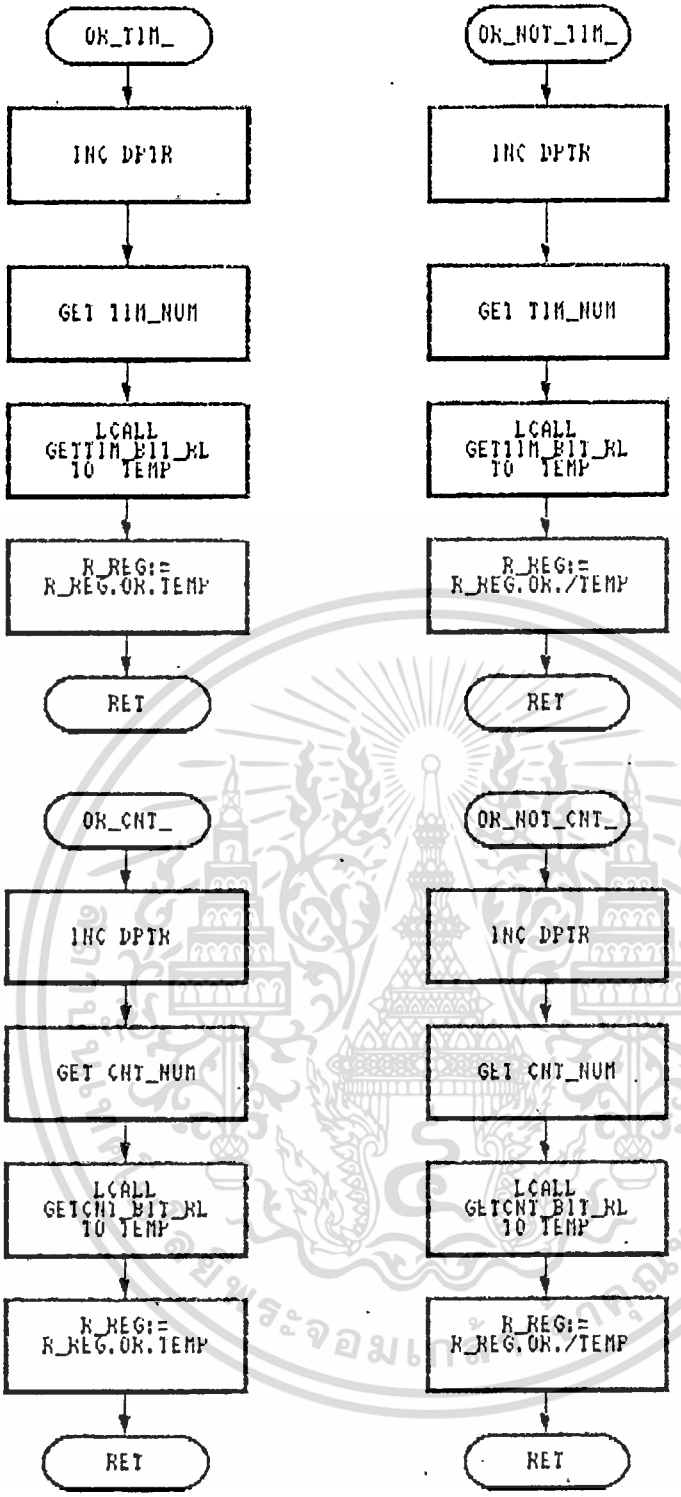


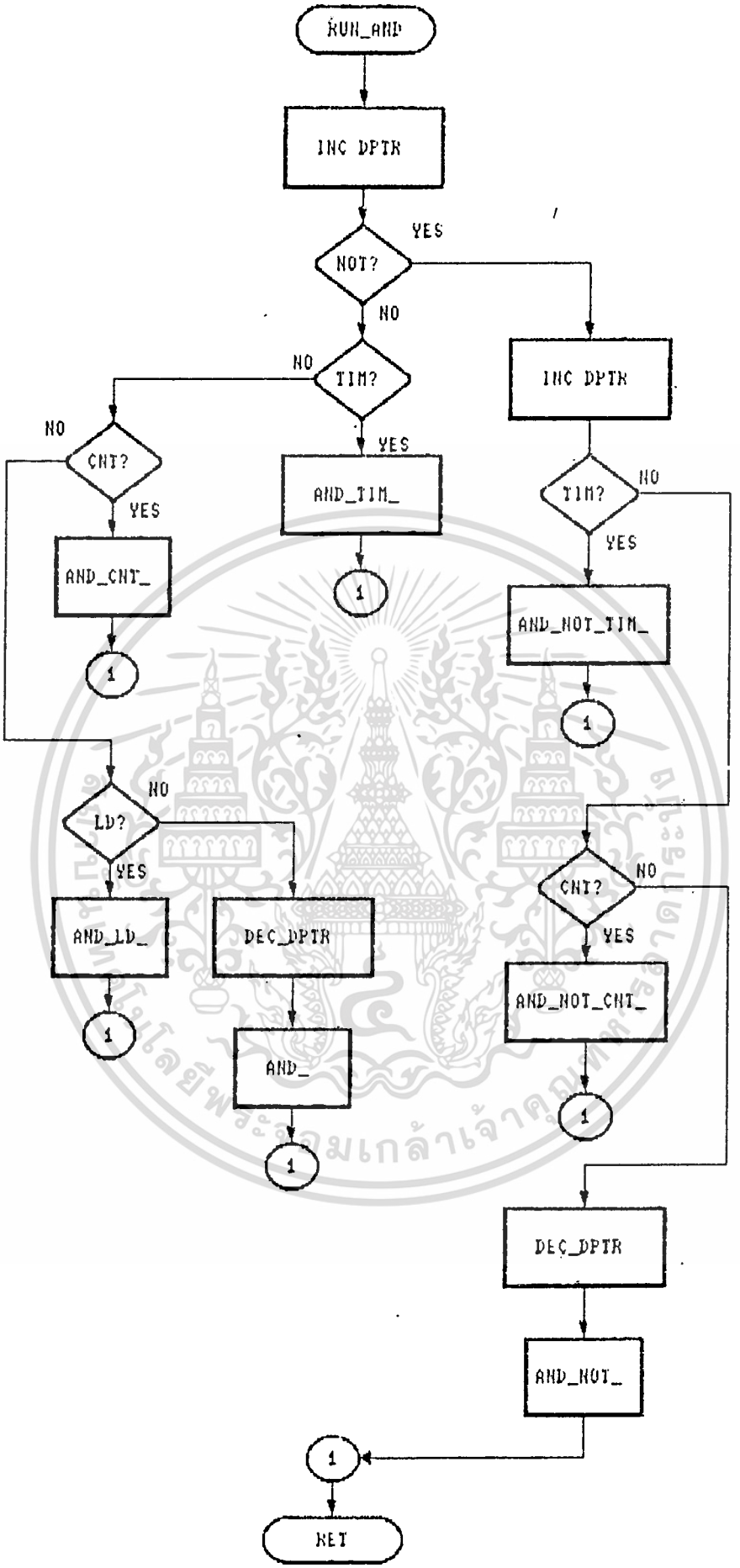


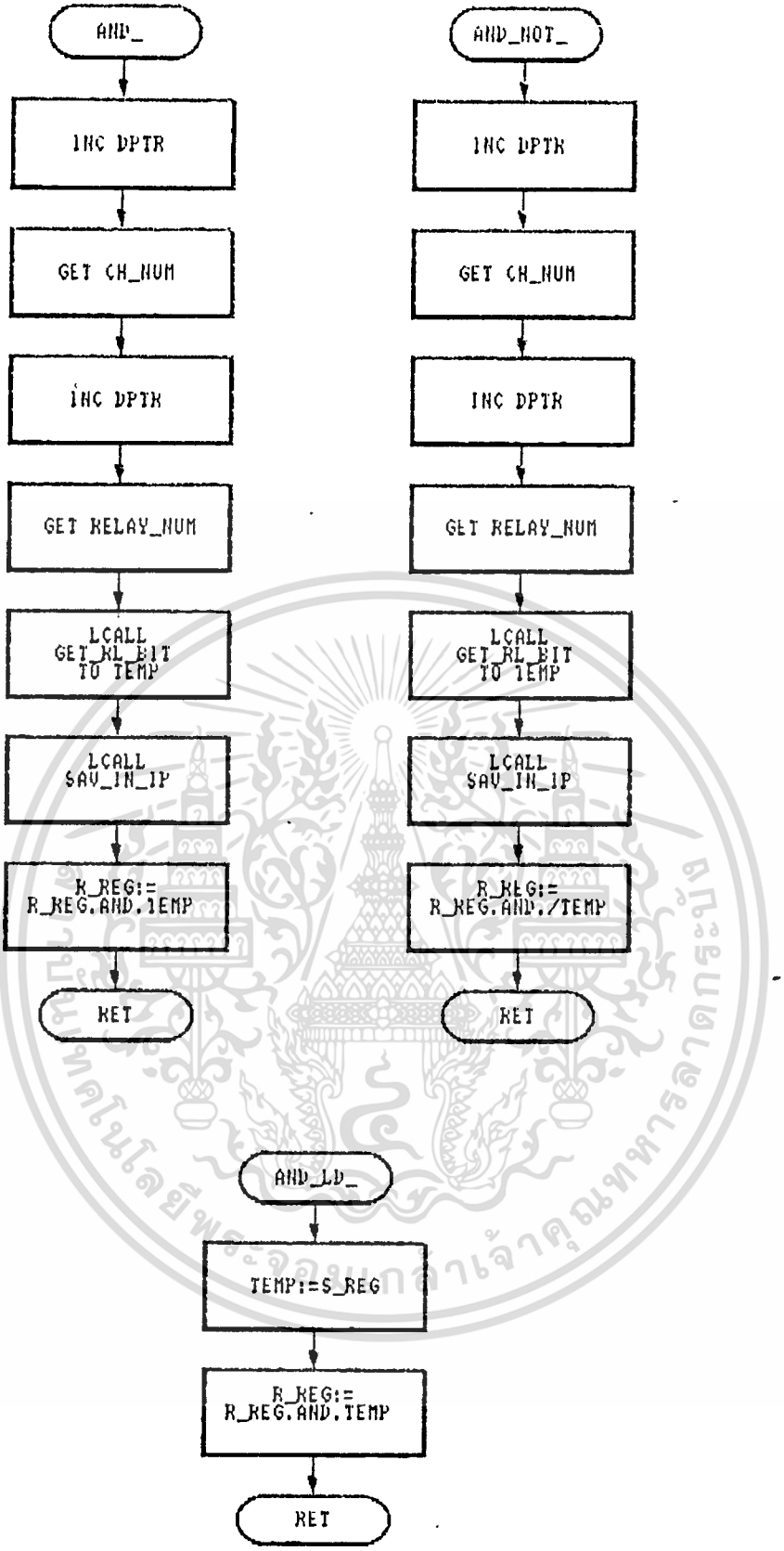


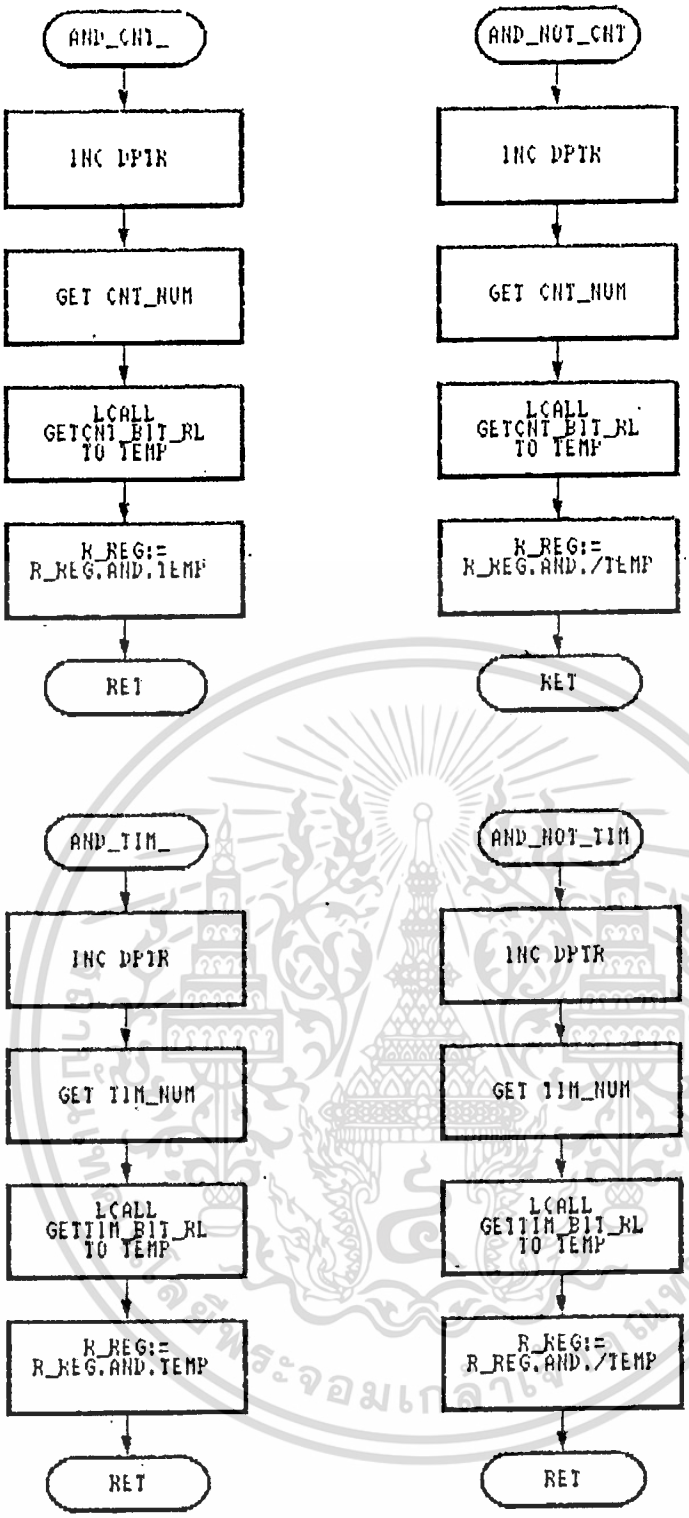


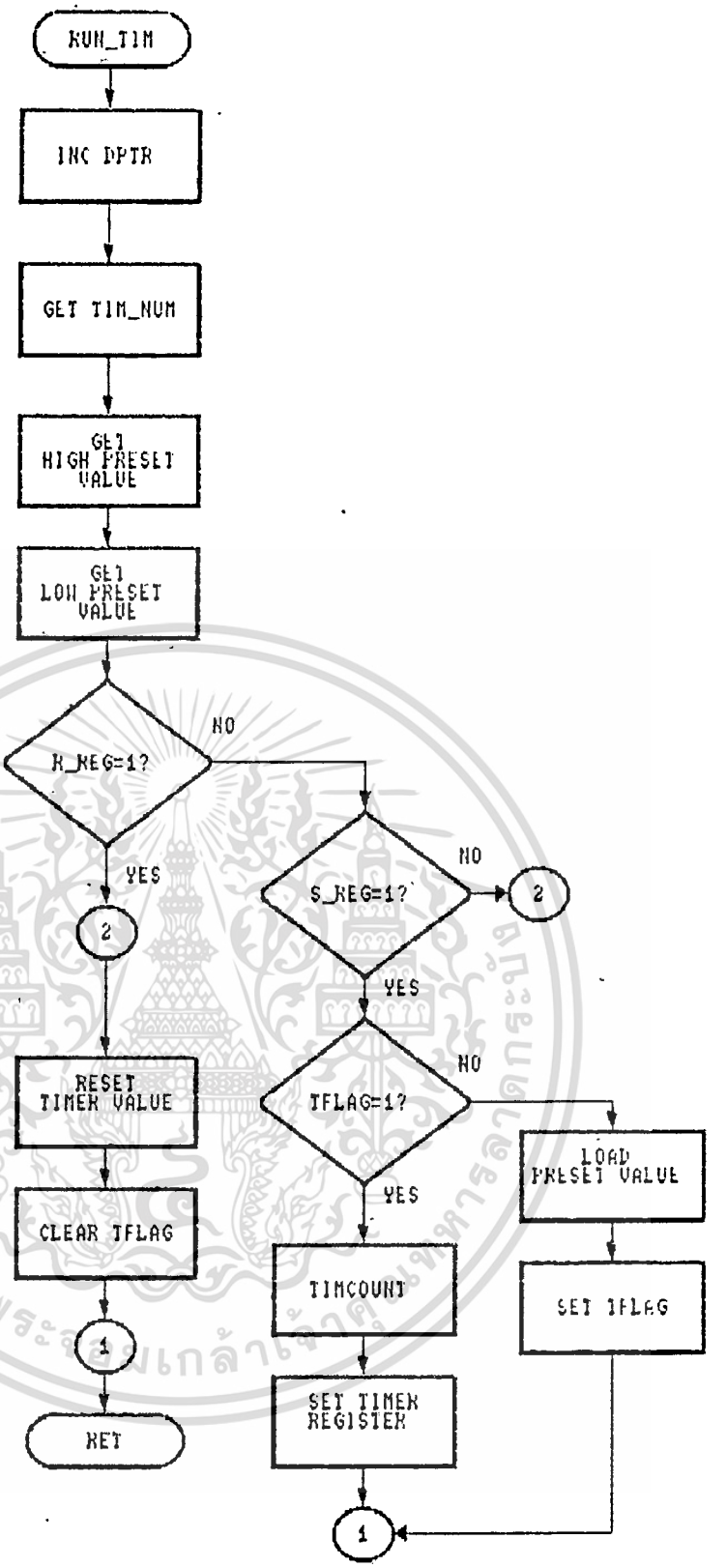


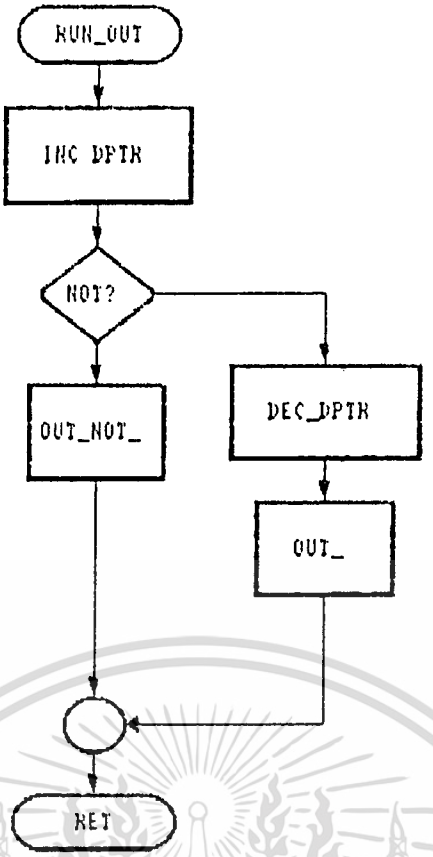


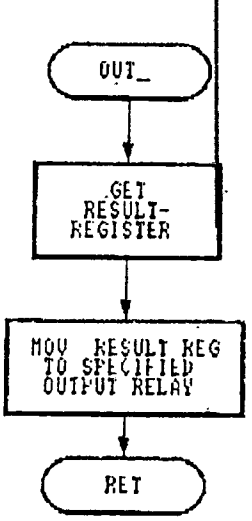


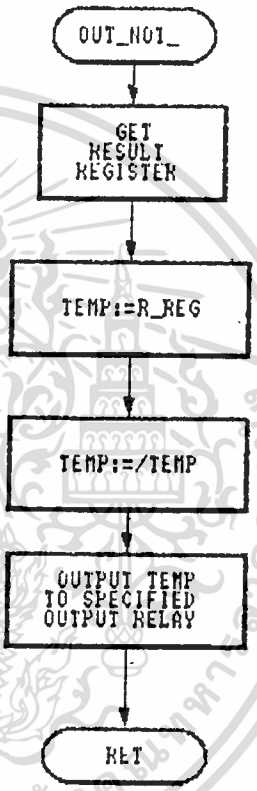












ภาคผนวก ค.
โปรแกรมของส่วนหน่วยป้อนโปรแกรม

```
        CPU     "8051.TBL"
        HOF     "INT8"

;*****************************************************************
;       MCS-51 INTERNAL REGISTERS
;-----------------------------------------------------------------

B:      EQU     0F0H    ;B REGISTER
ACC:    EQU     0E0H    ;ACCUMULATOR
PSW:    EQU     0D0H    ;PROGRAM STATUS WORD
IPC:    EQU     0B8H    ;INTERRUPT PRIORITY
P3:     EQU     0B0H    ;PORT 3
IEC:    EQU     0A8H    ;INTERRUPT ENABLE
P2:     EQU     0A0H    ;PORT 2
SBUF:   EQU     99H     ;SEND BUFFER
SCON:   EQU     98H     ;SERIAL CONTROL
P1:     EQU     90H     ;PORT 1
PCON:   EQU     87H     ;POWER CONTROL REGISTER
DPH:    EQU     83H     ;DATA POINTER HIGH
DPL:    EQU     82H     ;DATA POINTER LOW
SP:     EQU     81H     ;STACK POINTER
P0:     EQU     80H     ;PORT 0
;
;*****************************************************************
;           INTERNAL RAM USED
; ADDRESS
;           60H-64H  FOR INSTRUCTION PROGRAM
;           65H -LOW DIGIT DISPLAY
;           66H  2ND DIGIT
;           67H  3RD DIGIT
;           68H  4TH DIGIT
;           REGISTER USED
;
;           60H   NUMBER OF BYTE
;           61H   address pointer  line  high
;           62H     "       "      "   low
;           63H     "       "    instruction  high
;           64H     "       "      "          low
;
```

```
;       R7  USED IN READKEY ROUTINE AND RETURN KEY VALUE(1-40)
;-------------------------------------------------------------

        ORG  0000H

;*****************   MAIN  PROGRAM  HAS WORKED ******************************

 START:      LCALL   DELAY           ;initailize all value
        LCALL   INIT_LCD
        LCALL   DELAY_1
        LCALL   CLEAR
        LCALL   DELAY_1
        LCALL   BUFFER
        LCALL   TEST_RAM
        LCALL   LOAD_LINE
        LCALL   LOAD_INST
        MOV     A,#82H          ;for key board
        MOV     DPTR,#4003H     ;initail 8255
        Movx    @DPTR,A
        MOV     60H,#0H
        MOV     61H,#20H
        MOV     62H,#00H
        MOV     63H,#2FH
        MOV     64H,#0FFH
        LCALL   PASSWORD
sel_1:        LCALL  SEL_MOD        ;SELECT MODE PROGRAM and jump to
        Ljmp    sel_1           ;PROGRAM MODE,RUN MODE,MONITOR MODE

;*******************   DELAY TIME   ***************************

DELAY:        PUSH   00
        PUSH   01
        PUSH   02
        mov    r0,#01h
   delay1:  mov    r1,#50h
   delay2:  mov    r2,#84h
   delay3:  DJNZ   r2,delay3
        DJNZ   r1,delay2
        DJNZ   r0,delay1
```

```
        POP     02
        POP     01
        POP     00
        RET

DELAY_1:    PUSH    00                  ; 30 msec
        PUSH    01
        PUSH    02
        mov     r0,#1h
   delay01: mov     r1,#25h
   delay02: mov     r2,#64h
   delay03: DJNZ    r2,delay03
        DJNZ    r1,delay02
        DJNZ    r0,delay01
        POP     02
        POP     01
        POP     00
        RET

D_LAY_2:    PUSH    00                  ;DELAY TIME ---> 2s
        PUSH    01
        PUSH    02
        MOV     R0,#25H
 D_LAY1:    MOV     R1,#50H
 D_LAY2:    MOV     R2,#64H
 D_LAY3:    DJNZ    R2,D_LAY3
        DJNZ    R1,D_LAY2
        DJNZ    R0,D_LAY1
        POP     02
        POP     01
        POP     00
        RET

;*****************   INITIALIZE LCD DISPLAY    *********************

INIT_LCD:   MOV     DPTR,#6000H     ;WRITE COMMAND ADDRESS
        MOV     A,#38H          ;FUNCTION SET  DL=1 8 BIT
                    ;N=1 1/16 DUTY,F=0 DOTMATRIX 5X7
        MOVX    @DPTR,A
```

```
        LCALL  DELAY_1
        MOV    A,#0EH          ;DISPLAY ON/OFF D=1 ON,C=1 CURSOR ON
                               ;B=0 NO BLINKING
        MOVX   @DPTR,A
        LCALL  DELAY_1
        MOV    A,#06H          ;ENTRY MODE SET I/D=1 INCREMENT DIRECTION
                               ;S=0 NOT SHIFT THE DISPLAY
        MOVX   @DPTR,A
        LCALL  DELAY_1
        RET                    ;END INITIALIZE LCD

;********************  SUBROUTINE  ****************************

CLEAR:      MOV    DPTR,#8000H
        MOV    A,#01H          ;CLEAR DISPLAY and return to-
        MOVX   @DPTR,A         ;the first line
        RET

RETURN_HOME:  PUSH   DPH
        PUSH   DPL
        MOV    DPTR,#8000H
        MOV    A,#02H          ;RETURN HOME POSITION
        MOVX   @DPTR,A
        POP    DPL
        POP    DPH
        RET

NO_CURSOR:    MOV    DPTR,#8000H     ;NOT SHOW CURSOR
        MOV    A,#0CH
        MOVX   @DPTR,A
        RET

CUR_ON:     MOV    DPTR,#8000H     ;SHOW CURSOR
        MOV    A,#0EH
        MOVX   @DPTR,A
        RET

SECOND_LINE:  PUSH   DPH
        PUSH   DPL
```

```
        MOV     DPTR,#0C000H    ;CURSOR TO THE SECCOND LINE POSITION
        MOV     A,#0C0H
        MOVX    @DPTR,A
        POP     DPL
        POP     DPH
        RET

BUFFER:     MOV     R0,#30H             ;BUFFER ADDRESS
        MOV     R4,#28H             ;FOR 40 CHARACTER
        MOV     DPTR,#TABLE
        MOV     A,#00H              ;AT 0DBH  IN RAM
        MOV     R1,A
  LOOP1:    MOVC    A,@A+DPTR
        MOV     @R0,A               ;DATA FROM PROGRAM MEM
        INC     R0
        MOV     A,R1
        INC     A
        MOV     R1,A
        DJNZ    R4,LOOP1
        RET

;*****************************************************************
;THIS'S PROGRAM FOR GETTING PASSWORD FROM  USER
;-----------------------------------------------------------------

PASSWORD:     LCALL  PASS_LET
   GET_PW:  LCALL  READKEY
        MOV     A,R7                ;GET PASSWORD ID
        CJNE    A,#24H,GET_PW
        LCALL   CLEAR
        MOV     DPTR,#LET10
        LCALL   WRITE_CH
        LCALL   D_LAY_2
        LCALL   CLEAR
        RET

PASS_LET:     MOV     DPTR,#LET16           ;OUTPUT THE "PASSWORD : "
        LCALL   WRITE_CH
        RET
```

```
;*********************************************************************
;this routine is for get character form table to lcd display
;FORMAT
;               MOV    DPTR,#LET7
;               LCALL  WRITE_CH
;*********************************************************************

WRITE_CH:     MOV   A,#00H
   L_VEL:   MOV   R1,A
        MOVC  A,@A+DPTR
        CJNE  A,#24H,NOT_EQ   ;CHECK THE LAST CHARACTER IS "$" Y/N ?
        LJMP  O_U_T           ;IF "$" --> YES
  NOT_EQ:   PUSH  DPH
        PUSH  DPL
        MOV   DPTR,#6002H
        MOVX  @DPTR,A
        LCALL DELAY_1
        MOV   A,R1
        INC   A
        POP   DPL
        POP   DPH
        LJMP  L_VEL
    o_u_t:
        RET

;=====================================================================
;        READKEY ROUTINE
;  THIS PROGRAM SCAN KEY BOARD AND RETURN KEY VALUE
;        R7 = 1-40
;---------------------------------------------------------------------

READKEY:      MOV    R6,#0FEH
        AJMP   REPEAT1
  REPEAT0:   MOV    A,R6
        RL     A
        MOV    R6,A
  REPEAT1:   MOV    DPTR,#4000H
        MOV    A,R6
```

```
        MOVX    @DPTR,A
        INC     DPTR
        MOVX    A,@DPTR
        CJNE    A,#0FFH,RLOOP0
        AJMP    REPEAT0
RLOOP0:
RLOOP1:  LCALL  DELAY
        MOV     DPTR,#4000H
        MOV     A,R6
        MOVX    @DPTR,A
        INC     DPTR
        MOVX    A,@DPTR
        CJNE    A,#0FFH,CONT111
        AJMP    REPEAT0
;
;  NOW  WE CAN DETECT KEYPRESSED
;
CONT111:  MOV    R5,A
REPEAT2:  MOVX   A,@DPTR
        CJNE    A,#0FFH,REPEAT2
        PUSH    05
        LCALL   DELAY
RLOOP2:
        POP     05
        MOVX    A,@DPTR
        CJNE    A,#0FFH,REPEAT2
        LCALL   WRITETWOCHAR
        RET

WRITETWOCHAR: CJNE   R6,#0FEH,WSKIP1
        MOV     R7,#1
WSKIP1:   CJNE   R6,#0FDH,WSKIP2
        MOV     R7,#2
WSKIP2:   CJNE   R6,#0FBH,WSKIP3
        MOV     R7,#3
WSKIP3:   CJNE   R6,#0F7H,WSKIP4
        MOV     R7,#4
WSKIP4:   CJNE   R6,#0EFH,WSKIP5
        MOV     R7,#5
```

```
WSKIP5:  CJNE   R6,#0DFH,WSKIP6
         MOV    R7,#6
WSKIP6:  CJNE   R6,#0BFH,WSKIP7
         MOV    R7,#7
WSKIP7:  CJNE   R6,#07FH,WSKIP8
         MOV    R7,#8
WSKIP8:  MOV    A,R7
         CJNE   R5,#0FEH,WSKIP11
         ADD    A,#0
WSKIP11: CJNE   R5,#0FDH,WSKIP12
         ADD    A,#8
WSKIP12: CJNE   R5,#0FBH,WSKIP13
         ADD    A,#16
WSKIP13: CJNE   R5,#0F7H,WSKIP14
         ADD    A,#24
WSKIP14: CJNE   R5,#0EFH,WSKIP15
         ADD    A,#32
WSKIP15: MOV    R7,A                    ; return key value -->R7
         RET

;*****************************************************************

KEY_WR:
  KEY_WR_1:  LCALL   READKEY
         CJNE   R7,#36,KEY_WR_1
         mov    dptr,#6000h
         mov    a,#8Bh
         movx   @dptr,a
         LCALL  DELAY
         mov    dptr,#let17
         lcall  write_ch
         lcall  delay
         MOV    DPTR,#6000H
         MOV    A,#8AH
         MOVX   @DPTR,A
         LCALL  DELAY
         MOV    DPTR,#LET18
         LCALL  WRITE_CH
         LCALL  DELAY
```

```
        MOV     DPTR,#8000H
        MOV     A,#89H
        MOVX    @DPTR,A
        LCALL   DELAY
        MOV     DPTR,#LET19
        LCALL   WRITE_CH
        LCALL   DELAY
        LCALL   CLEAR
        LCALL   RETURN_HOME
        LCALL   POINT_8
        RET

DOWN_1:         PUSH    00
        PUSH    02
    down1:  MOV     A,R7
        CJNE    A,#11H,NOT_CNT1
        LCALL   TO_CNT
        LJMP    OUT_1
    NOT_CNT1: CJNE  A,#12H,NOT_TIM1
        LCALL   TO_TIM
        LJMP    OUT_1
    NOT_TIM1: CJNE  A,#14H,NOT_LD1
        LCALL   TO_LD
        LJMP    OUT_1
    NOT_LD1:  CJNE  A,#15H,NOT_AND1
        LCALL   TO_AND
        LJMP    OUT_1
    NOT_AND1: CJNE  A,#16H,NOT_FUN1
        LCALL   TO_FUN
        LJMP    OUT_1
    NOT_FUN1: CJNE  A,#18H,NOT_OR1
        LCALL   TO_OR
        LJMP    OUT_1
    NOT_OR1:  CJNE  A,#13H,NOT_OUT1
        LCALL   TO_OUT
        LJMP    OUT_1
    NOT_OUT1: lcall readkey
        sjmp    down1
    OUT_1:  POP     02
```

```
            POP     00
            RET

;=============================================================
;
; THIS ROUTINE IS SUB_ROUTINE IN DOWN_1 ROUTINE :
;  USER TO CHOOSE PRESS  COMMAND KEY
;       1. TO_LD
;       2. TO_OUT
;       3. TO_AND
;       4. TO_OR
;       5. TO_FUN
;       6. TO_TIM
;       7. TO_CNT
;-------------------------------------------------------------

TO_LD:      LCALL   SPACE_2
        MOV     DPTR,#LET1
        LCALL   WRITE_CH
        lcall   point_1
        LCALL   KEY_LD
        RET

TO_OUT:     LCALL   SPACE_2
        MOV     DPTR,#LET2
        LCALL   WRITE_CH
        lcall   point_1
        LCALL   KEY_OUT
        RET

TO_AND:     LCALL   SPACE_2
        MOV     DPTR,#LET3
        LCALL   WRITE_CH
        lcall   point_1
        LCALL   KEY_AND
        RET

TO_OR:      LCALL   SPACE_2
        MOV     DPTR,#LET4
```

```
        LCALL   WRITE_CH
        lcall   point_1
        LCALL   KEY_OK
        RET

TO_FUN:     LCALL   SPACE_2
        MOV     DPTR,#LET8
        LCALL   WRITE_CH
        lcall   point_1
        LCALL   KEY_FUN
        RET

TO_CMT:     LCALL   SPACE_2
        MOV     DPTR,#LET6
        LCALL   WRITE_CH
        lcall   point_1
        LCALL   KEY_CMT
        RET

TO_TIM:     LCALL   SPACE_2
        MOV     DPTR,#LET7
        LCALL   WRITE_CH
        lcall   point_1
        LCALL   KEY_TIM
        RET

;=============================================================
;  ROUTINE KEY_LD IN TO_LD ROUTINE
;-------------------------------------------------------------

  KEY_LD:       PUSH    00
            PUSH    01
            PUSH    02
    KEY_LD_1:   LCALL   READKEY
            MOV     A,R7
            CJNE    A,#17H,NOT_LD_NOT
            LCALL   LD_NOT
            LJMP    OUT_2
    NOT_LD_NOT: CJNE    A,#12H,NOT_LD_TIM
```

```
        LCALL   LD_TIM
        LJMP    OUT_2
NOT_LD_TIM: CJNE    A,#11H,NOT_LD_CNT
        LCALL   LD_CNT
        LJMP    OUT_2
NOT_LD_CNT: LCALL   NUM_1
OUT_2:      POP     02
        POP     01
        POP     00
        RET

LD_CNT:  MOV     DPTR,#LET6
        LCALL   WRITE_CH
        lcall   point_1
        LCALL   NUM_2
        RET

LD_TIM:  MOV     DPTR,#LET7
        LCALL   WRITE_CH
        lcall   point_1
        LCALL   NUM_2
        RET

LD_NOT:  MOV     DPTR,#LET5
        LCALL   WRITE_CH
        lcall   point_1
        LCALL   READKEY
        CJNE    R7,#11H,LINE_1
        LCALL   LD_CNT
        SJMP    EXIT_4
LINE_1: CJNE    R7,#12H,LINE_2
        LCALL   LD_TIM
        SJMP    EXIT_4
LINE_2: LJMP    NUM_1
EXIT_4:
        RET

KEY_FUN:        LCALL   READKEY
        LCALL   NUMBER
```

```
          lcall    point_1
          LCALL    NUM_3
          RET

KEY_OR:         LCALL    READKEY
          CJNE     R7,#14H,LINE_3
          MOV      DPTR,#LET1
          LCALL    WRITE_CH
          lcall    point_1
          LJMP     LINE_4
     LINE_3: LCALL    CHECK_FN
     LINE_4:
          RET

KEY_AND:        LCALL    READKEY
          CJNE     R7,#14H,LINE_5
          MOV      DPTR,#LET1
          LCALL    WRITE_CH
          lcall    point_1
          LJMP     LINE_6
     LINE_5: LCALL    CHECK_FN
     LINE_6:
          RET

KEY_TIM:        LCALL    NUM_2
          LCALL    NUM_2
          RET

KEY_CNT:        LCALL    NUM_2
          LCALL    NUM_2
          RET

KEY_OUT:        LCALL    READKEY
          CJNE     R7,#17H,LINE_7
          MOV      DPTR,#LET5
          LCALL    WRITE_CH
          lcall    point_1
   LINE_7:    LCALL    NUM_1
          RET
```

```
;=============================================================
; This routine is used by AND-routine, OR-routine to select
;  which  follow with LD or not?  If it's not with LD ,they
;  like LD-routine to select call call-function
;-------------------------------------------------------------

CHECK_FN:          PUSH     00
                   PUSH     01
                   PUSH     02
     EXT_1:
                   CJNE     r7,#17H,EXT_2
                   LCALL    LD_NOT
                   LJMP     EXT_OUT
     EXT_2:        CJNE     A,#12H,EXT_3
                   LCALL    LD_TIM
                   LJMP     EXT_OUT
     EXT_3:        CJNE     A,#11H,EXT_4
                   LCALL    LD_CNT
                   LJMP     EXT_OUT
     EXT_4:        LCALL    NUM_1
     EXT_OUT:
                   POP      02
                   POP      01
                   POP      00
                   RET

;=============================================================
; NUM_1 ROUTINE IS TO DISPLAY xx:yy   AS
;       xx  =    display 2 numbers of CHANNEL
;       yy  =    display 2 numbers of   PORT
;             which between them be mark " : "
;
; NUM_2 ROUTINE  IS TO DISPLAY  zzzz   AS
;       zzzz  =  display 4 numbers fo  NUMBER OF HEX-NUMBER
;-------------------------------------------------------------

NUM_1:             push     00
          push     01
```

```
        push    02
        LCALL   SPACE_2
NUMB_1:   LCALL   NUMBER
        LCALL   CH_PORT
        lcall   point_1
        LCALL   DELAY
        LCALL   READKEY
        LCALL   NUMBER
        LCALL   CH_PORT
        lcall   point_1
        MOV     DPTR,8002H
        MOV     A,83AH          ; WRITE " : " ON LCD
        MOVX    @DPTR,A
        mov     r7,a
        lcall   point_1
        LCALL   DELAY
        MOV     R1,02H
NUMB_2:     LCALL   READKEY
        LCALL   NUMBER
        LCALL   CH_PORT
        lcall   point_1
        LCALL   DELAY
        LCALL   DELAY
        DJNZ    R1,NUMB_2
        pop     02
        pop     01
        pop     00
        RET


NUM_2:          PUSH    00
        PUSH    01
        PUSH    02
        LCALL   SPACE_2
        MOV     R1,04H
    NUMB_4: LCALL   READKEY
        LCALL   NUMBER
        LCALL   CH_PORT
        lcall   point_1
```

```
        DJNZ    R1,NUMB_4
        POP     02
        POP     01
        POP     00
        RET


NUM_3:          PUSH    00
        PUSH    01
        PUSH    02
        LCALL   SPACE_2
        LCALL   CH_PORT
        LCALL   READKEY
        LCALL   NUMBER
        lcall    point_1
        LCALL   CH_PORT
        POP     02
        POP     01
        POP     00
        RET

;===================================================================
;
;   ROUTINE  IN  KEY_LD
;     1. CH_PORT is  to play number on LCD  that channel and port
;                or  to play numbers on LCD follow loop command
;     2. NUMBER  is  to check pressed-key be a number?  return value
;                as  a number from 0 to F to R7
;
;     THESE ROUTINES ARE USED IN NUM_1, NUM_2 ROUTINE
;-------------------------------------------------------------------


CH_PORT:        push   00
        push   01
        MOV    A,r7             ; STROE R7
        MOV    DPTR,#8002H       ; WRITE  COMMAND
        MOV    R0,#2FH           ; SET ADDRESS IN RAM = 30H
        ADD    A,R0
        MOV    R0,A
```

```
        MOV     A,@R0
        MOVX    @DPTR,A
        LCALL   DELAY
        pop     01
        pop     00
        RET

NUMBER:
  NOT_A:    MOV     A,R7
        CJNE    A,#1H,NOT_1
        LJMP    NUMBER1
  NOT_1:    CJNE    A,#2H,NOT_0
        LJMP    NUMBER1
  NOT_0:    CJNE    A,#3H,NOT_5
        LJMP    NUMBER1
  NOT_5:    CJNE    A,#4H,NOT_4
        LJMP    NUMBER1
  NOT_4:    CJNE    A,#5H,NOT_9
        LJMP    NUMBER1
  NOT_9:    CJNE    A,#6H,NOT_C
        LJMP    NUMBER1
  NOT_C:    CJNE    A,#7H,NOT_D
        LJMP    NUMBER1
  NOT_D:    CJNE    A,#8H,NOT_8
        LJMP    NUMBER1
  NOT_8:    CJNE    A,#9H,NOT_3
        LJMP    NUMBER1
  NOT_3:    CJNE    A,#0AH,NOT_2
        LJMP    NUMBER1
  NOT_2:    CJNE    A,#0BH,NOT_7
        LJMP    NUMBER1
  NOT_7:    CJNE    A,#0CH,NOT_6
        LJMP    NUMBER1
  NOT_6:    CJNE    A,#0DH,NOT_B
        LJMP    NUMBER1
  NOT_B:    CJNE    A,#0EH,NOT_E
        LJMP    NUMBER1
  NOT_E:    CJNE    A,#0FH,NOT_F
        LJMP    NUMBER1
```

```
NOT_F:      CJNE   A,#10H,NOT_A
NUMBER1:    MOV    R7,A
            SJMP   EXIT_3
NOT_A:      LCALL  READKEY
            LJMP   NOT__A
EXIT_3:
            RET

SPACE_3:          mov      dptr,#6002h
            mov      a,#20h
            movx     @dptr,a
            lcall    delay
            movx     @dptr,a
            lcall    delay
            movx     @dptr,a
            lcall    delay
            RET

SPACE_2:          mov      dptr,#6002h
            mov      a,#20h
            movx     @dptr,a
            lcall    delay
            movx     @dptr,a
            lcall    delay
            RET

SPACE_1:          mov      dptr,#6002h
            mov      a,#20h
            movx     @dptr,a
            lcall    delay
            RET

;=================================================================
;              THIS IS ROUTINE FOR SELECT MODE                   =
;       SECLECT  1 -> MODE PROGRAM                               =
;                2 -> MODE RUN                                   =
;                3 -> MODE MONITOR                               =
;-----------------------------------------------------------------
```

```
SEL_MOD:    MOV     DPTR,#MOD_LET   ;DISPLAY FIRST LINE
            LCALL   WRITE_CH        ;" SECLECT MODE 1 2 OR 3"
            LCALL   DELAY
            LCALL   SECOND_LINE     ;SECOND LINE DISPLAY
            LCALL   DELAY
            MOV     DPTR,#MOD_2LET  ;"PROG(1) RUN(2) MON(3)"
            LCALL   WRITE_CH
  SMOD:     LCALL   READKEY
            CJNE    R7,#1H,SKIP
            LCALL   MOD_PRO
            LJMP    SKIP_3
   SKIP:    CJNE    R7,#0AH,SKIP_1
            LCALL   MOD_RUN
            LJMP    SKIP_3
  SKIP_1:   CJNE    R7,#9H,SKIP_2
            LCALL   MOD_MON
            LCALL   CLEAR
            mov     dptr,#let15
            lcall   write_ch
            lcall   d_lay_2
            LCALL   CLEAR
            LJMP    SKIP_3
  SKIP_2:   LCALL   CLEAR
            MOV     DPTR,#ERR_SEL
            LCALL   WRITE_CH
            LCALL   D_LAY_2
            LCALL   D_LAY_2
            LCALL   CLEAR
  SKIP_3:
            RET

;=================================================================
;           PROGRAM MODE                                        .
;utility routine                                                 .
;    use F3 routine for up cursor                                .
;         F4     down cursor                                     .
;         F1   - (return )                                       .
;         F2   - (clear)                                         .
;         F5   - (shift character right)                         .
```

```
;       F6   - (shift character left)                          .
;       R_INS  insert character                                .
;       R_INC  increment cursor forward                        .
;       R_DEC  decrement cursor backward                       .
;       R_DEL  delete character                                .
;       R_JMP  jump to address that want to see display        .
;       R_WRITE when fill data already then save it's data and .
;              goto the next address display                   .
;       R_SPACE backward to delete data that address           .
; instruction program routine of PLC                           .
;       R_FUN  for append another uses and this is used to     .
;              assign the of program (FUN 01 = END.)           .
;       R_LOAD for load  data from relay number                .
;       R_OUT  output data to relay number                     .
;       R_OR   get data from the result register OR with input .
;              data and store result to result register        .
;       R_AND  get data from result register AND with input    .
;              data and store result to result register        .
;       R_NOT  get data and NOT it                             .
;       R_TIM  for timer                                       .
;       R_CNT  for counter                                     .
;-------------------------------------------------------------

MOD_PRO:      lcall    delay
       LCALL     CLEAR        ;clear lcd display
       lcall     delay
       MOV       65H,#0FFH    ;initial first digit address display
       MOV       66H,#00H     ;        2nd  digit
       MOV       67H,#0H      ;        3rd  digit
       MOV       68H,#0H      ;        4th  digit
       lcall     delay
 mod_pro_1: lcall     f3
       lcall     readkey
        CJNE     R7,#23h,MOD_PRO_2_1
        LCALL    R_DEC
        LJMP     MOD_PRO_2
mod_pro_2_1:  CJNE     R7,#25h,MOD_PRO_2_2
       LCALL     R_INC
       LJMP      MOD_PRO_2
```

```
mod_pro_2_2:   cjne    r7,#30,mod_pro_2
               lcall   clear
               ljmp    sel_mod
  mod_pro_2:   lcall   second_line
               lcall   down_1
               lcall   key_wr
               ljmp    mod_pro_1
               RET

MOD_RUN:       LCALL  CLEAR
               MOV    DPTR,#LET12
               LCALL  WRITE_CH
               LCALL  D_LAY_2          ; RUN MODE HERE!
               LCALL  CLEAR
               RET

MOD_MON:       lcall  no_cursor
               LCALL  DELAY
               LCALL  CLEAR
               MOV    DPTR,#LET13
               LCALL  WRITE_CH
               LCALL  D_LAY_2          ; MONITOR MODE HERE !
               LCALL  CLEAR
               lcall  delay
               mov    dptr,#START_1
               lcall  write_ch
               lcall  d_lay_2
               LCALL  CLEAR
               LCALL  DISPLAY_1
               RET

;=========================================================

LOAD_LINE:     MOV   61H,#0H            ; TO LOAD NUMBER OF LINE
               MOV   62H,#0H            ; IN RAM  FOR INITIAL
               MOV   DPTR,#2000H
               MOV   A,61H
               MOVX  @DPTR,A
               MOV   A,62H
```

```
        INC   DPTR
        MOVX  @DPTR,A
LOAD_L1: INC   62H
        INC   DPTR
        MOV   A,61H
        MOVX  @DPTR,A
        INC   DPTR
        MOV   A,62H
        MOVX  @DPTR,A
        CJNE  A,#0FFH,LOAD_L1
        INC   61H
        MOV   A,61H
        CJNE  A,#3H,LOAD_L1
        RET

LOAD_INST:    MOV   A,#0H              ; TO INITIAL RAM WITH 0H
        MOV   63H,#2FH
        MOV   64H,#0FFH
 L_INST_1:  MOV   DPH,63H
        MOV   DPL,64H
        MOVX  @DPTR,A
        DEC   64H
        MOV   A,64H
        CJNE  A,#0FFH,L_INST_1
        DEC   63H
        MOV   A,63H
        CJNE  A,#28H,L_INST_1
        RET

POINT_1:       push   00
         push   01
         MOV    A,R7              ; POINTER OF DATA FROM #2FFFH
         MOV    R0,63H            ; DEC 1
         MOV    R1,64H
         CJNE   R1,#0H,POINT_P1
         DEC    R0
  POINT_P1: DEC    R1
         MOV    63H,R0
         MOV    64H,R1
```

```
        MOV     DPH,63H
        MOV     DPL,64H
        MOVX    @DPTR,A
        INC     60H
        pop     01
        pop     00
        RET

POINT_3:        push    00
        push    01
        MOV     A,60H
        MOV     DPH,63H
        MOV     DPL,64H
   POINT_P3:  INC     DPTR
        DJNZ    60H,POINT_P3
        MOVX    @DPTR,A
        mov     60h,80h
        mov     a,64h
        cjne    a,80h,point_p4
        dec     63h
   point_p4:  dec     64h
        pop     01
        pop     00
        RET

;-------------------------------------------------
;       MOVE UP CURSOR ROUTINE(F3)
;=================================================
;  USE INTERNAL RAM ADDRESS  FOR PASS VALUE
;  DISPLAY DATA ADDRESS
;           (68H)(67H)(66H)(65H) = 0000 - 9999 DECIMAL
;           65H FOR STORE LOW DIGIT
;           66H             2ND DIGIT
;           67H             3RD DIGIT
;           68H             HIGH DIGIT DISPLAY
;      USE WR_DIGIT ROUTINE
;INITAIL VALUE (65H)=0FFH  (66H),(67H),(68H)=0H FOR FIRST-> 0000
;-------------------------------------------------
```

```
F8:         INC    65H
            MOV    A,65H
            CJNE   A,#0Fh,F_SKIP   ; IF (65H)=/= #0F THEN SKIP
            LJMP   F_OUT           ; IF (65H)= 0F
F_SKIP:     JC     F_OUT           ; IF (65H)< 0F THEN SKIP
            mov    65h,#0          ; IF (65H)> 0F
INC65:      INC    66H
            MOV    A,66H           ; TEST (66H)=<0F ?
            CJNE   A,#0FH,F_SKIP1  ; IF YES THEN F_OUT
            LJMP   F_OUT           ; IF NOT THEN CONTINUE
F_SKIP1:    JC     F_OUT
            mov    66h,#0
INC66:      INC    67H
            MOV    A,67H           ; TEST (67H)=<0F ?
            CJNE   A,#0FH,F_SKIP2  ; IF YES THEN F_OUT
            LJMP   F_OUT           ; IF NOT THEN CONTINUE
F_SKIP2:    JC     F_OUT
            mov    67h,#0
INC67:      INC    68H
            MOV    A,68H           ; TEST (68H)=<0F ?
            CJNE   A,#0FH,F_SKIP3  ; IF YES THEN F_OUT
            LJMP   F_OUT           ; IF NOT THEN CONTINUE
F_SKIP3:    JC     F_OUT
            MOV    68H,#0
F_OUT:      LCALL  WR_DIGIT
            RET

R_PHAT:     MOVX   A,@DPTR
            INC    DPTR
            PUSH   DPH
            PUSH   DPL
            MOV    DPTR,#8002H
            MOVX   @DPTR,A
            POP    DPL
            POP    DPH
            LCALL  DELAY_1
            DJNZ   R1,R_PHAT
            RET
```

```
R_DEL:      MOV   DPTR,#6002H
            MOV   A,#20H
            MOVX  @DPTR,A
            LCALL DELAY
            MOV   DPTR,#6000H
            MOV   A,#10H
            MOVX  @DPTR,A
            RET

R_DEC:      LCALL F4  ;CURSOR UP
            RET

R_INC:      LCALL F8  ;CURSOR DOWN
            RET
;=================================================================
;  JMP ROUTINE FOR JUMP TO TARGET ADDRESS                        .
;    IF JUMP KEY ACTIVE THEN                                     .
;      DISPLAY    " JUMP TO :    "                              .
;  AND FILL 4 DIGIT RIGHT TO LEFT (HIGH TO LOW DIGIT ADDRESS)    .
;  AND DISPLAY THE ADDRESS JUMP AND SHOW DATA                    .
;      65H,66H,67H,68H STORE ADDRESS JUMP                        .
;      69H,6AH,6BH,6CH STORE ADDRESS BEFORE JUMP                 .
;-----------------------------------------------------------------
R_JMP:      PUSH  00
            PUSH  01
            LCALL DELAY
            LCALL CLEAR
            MOV   DPTR,#JMP_LET     ;" JMP TO :    "
            LCALL WRITE_CH
            MOV   DPTR,#6000H
            MOV   A,#8EH            ;CURSOR AT FIRST LINE COLUMN 15
            LCALL DELAY
            MOVX  @DPTR,A
            MOV   6CH,68H           ;SAVE ADDRESS BEFORE JUMP
            MOV   6BH,67H           ;(6CH)(6BH)(6AH)(69H)
            MOV   6AH,66H           ; 6CH HIGH DIGIT ADDRESS
            MOV   69H,65H           ; 69H LOW  DIGIT ADDRESS
            MOV   R1,#4H
            MOV   R0,#68H
```

```
GET_AGAIN:   LCALL  READKEY          ;READ FIRST DIGIT
             LCALL  NUMBER           ;NUMBER IN (62H)
             MOV    @R0,62H          ;GET ADDRESS FOR JUMP
             MOV    A,@R0
             CJNE   A,#0AH,GET_SKIP
             SJMP   GET_SKIP2
GET_SKIP:    JC     GET_SKIP3        ;A >9    ->A,B,C,D,E,F
GET_SKIP2:   CLR    C
             ANL    A,#0FH
             SUBB   A,#0AH
             ADD    A,#41H
             SJMP   LCD_DISP
GET_SKIP3:   ADD    A,#30H           ;A=<9    0,1,2,3,4,5,6,7,8,9
LCD_DISP:    LCALL  DELAY
             MOV    DPTR,#8002H
             MOVX   @DPTR,A
             DEC    R0
             DJNZ   R1,GET_AGAIN     ;GET 4TH DIGIT Y/N ?
                                     ;IF YES..CONTINUE
WAIT_WR:     LCALL  READKEY
             CJNE   R7,#24H,WAIT_WR  ;CHECK "WR" KEY IS PRESSED Y/N?
                                     ;IF    YES ..CONTINUE
             LCALL  DELAY
             LCALL  CLEAR
             LCALL  WR_DIGIT
             LCALL  WR_DATA          ;SHOW DATA AT ADDRESS JUMP
             POP    01
             POP    00
             RET

;-------------------------------------------------------------
;THIS PROGRAM ROUTINE FOR USE
; READ DATA FROM RAM ADDRESS 2000H-2FFFH(4K)
; TO LCD (SECCOND LINE )
; PASS VALUE  FROM INTERNAL RAM ADDRESS 65H 66H 67H 68H
;-------------------------------------------------------------

WR_DATA:     PUSH   DPH
             PUSH   DPL
```

```
      PUSH    01
      MOV     R1,#2H
      MOV     DPTR,#6000H
      MOV     A,#0D0H     ;SET CURSOR SECCOND LINE COLUMN 17
      MOVX    @DPTR,A
      LCALL   DELAY
      CLR     A           ;CLEAR
      MOV     A,66H       ;LOAD 2ND DIGIT ADDRESS DISPLAY
      SWAP    A           ;MOV 2ND DIGIT TO HIGH NIBBLE
      ADD     A,65H       ;GET 1ST DIGIT AT LOW NIBBLE OF BYTE
      MOV     DPL,A       ;GET LOW ADDRESS TO LOW BYTE OF DATA POINTER
      CLR     A
      MOV     A,68H
      SWAP    A
      ADD     A,67H
      ADD     A,#20H      ;START ADDRESS 2000H
      MOV     DPH,A       ;GET HIGH ADDRESS TO HIGH BYTE OF DATA POINTER
      MOVX    A,@DPTR     ;GET DATA RAM
      MOV     DPTR,#6002H
      MOV     R7,A
      SWAP    A
WR_SKIP3:  ANL     A,#0FH              ; SELECT 4 HIGH BIT TO LCD
      CJNE    A,#0AH,WR_SKIP
      SJMP    WR_SKIP1
WR_SKIP:   JC      WR_SKIP2            ; A <9  THEN 0,1,2,3,4,5,6,7,8,9
WR_SKIP1:  CLR     C                   ; A >=9  THEN A,B,C,D,E,F
      SUBB    A,#0AH
      ADD     A,#41H
      LCALL   DELAY
      MOVX    @DPTR,A
      SJMP    WR_SKIP4
WR_SKIP2:  ADD     A,#30H
      LCALL   DELAY
      MOVX    @DPTR,A
WR_SKIP4:  MOV     A,R7                ; SELECT 4 LOW BIT TO LCD
      DJNZ    R1,WR_SKIP3
      POP     01
      POP     DPL
      POP     DPH
```

```
        RET
;-----------------------------------------------------------------
;THIS ROUTIME FOR DOWN CURSOR DISPLAY THE LOWER ADDRESS DISPLAY
;         (68H)(67H)(66H)(65H)=9999-0000 DECIMAL DECREMENT
;            65H 1ST DIGIT (LOW)
;            66H 2ND
;            67H 3RD
;            68H 4TH DIGIT(HIGH)
;          USE WR_DIGIT ROUTINE
; INITAIL VALUE (65H)=0AH   (66H),(67H),(68H)=09H
;-----------------------------------------------------------------

F4:           DEC     65H
        MOV     A,65H
        CJNE    A,#0FFH,F4_OUT      ; LOW DIGIT >= 0? IF YES THEN SKIP
        MOV     65H,#0FH            ; IF EQUAL 0 THEN INITAIL
  DEC66:     DEC     66H
        MOV     A,66H
        CJNE    A,#0FFH,F4_OUT      ; 2ND DIGIT >=0 ? IF YES ->SKIP
        MOV     66H,#0FH            ; IF NOT THEN INITAIL
  DEC67:     DEC     67H
        MOV     A,67H
        CJNE    A,#0FFH,F4_OUT      ; 3RD DIGIT >=0? IF YES THEN SKIP
        MOV     67H,#0FH            ; IF NOT THEN INITIAL
  DEC68:     DEC     68H
        MOV     A,67H
        CJNE    A,#0FFH,F4_OUT      ; 4TH DIGIT >=0? IF YES THEN SKIP
        MOV     68H,#0FH            ; IF NOT THEN INITAIL
 F4_OUT:    LCALL   DELAY
        LCALL   RETURN_HOME
        LCALL   WR_DIGIT            ; 4 digit to lcd display
        RET


;\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
;             WR_DIGIT ROUTINE
;
; USED BY F3 (CURSOR UP) AND F4 CURSOR DOWN
```

```
;  WRITE 4 DIGIT TO LCD
;  USE TABLE NAME --> DISP_ADDR
;                     INTERNAL REG. ADDRESS
;                     65H,66H,67H,68H FOR PASS VALUE
;-------------------------------------------------

WR_DIGIT:      PUSH    01
        PUSH    00
        LCALL   DELAY
        LCALL   RETURN_HOME
        LCALL   DELAY
        MOV     R1,#4H
        MOV     R0,#68H          ;WRITE ADDRESS DISPLAY 4 DIGIT
  F_REP:    MOV     A,@R0              ;TO LCD
        MOV     DPTR,#DISP_ADDR  ;BASE TABLE
        MOVC    A,@A+DPTR        ;POINT TO TABLE AND GET DATA
        MOV     DPTR,#6002H      ;
        lcall   delay
        MOVX    @DPTR,A          ;TO LCD DISPLAY
        DEC     R0
        DJNZ    R1,F_REP         ; 4 DIGIT DISPLAY YET ?
                        ; IF YES CONTINUE
        POP     00
        POP     01
        RET

;-------------------------------------------------------------

DISPLAY_1:          PUSH     00
           PUSH     01
           PUSH     02
           MOV      61H,#20H
           MOV      62H,#00H
           MOV      DPTR,#2FFFH
           MOVX     A,@DPTR
    disp_4:     jnz      disp_2
           ljmp     disp_5
     disp_2:    mov      60h,a
           MOV      63H,DPH
```

```
      MOV     64H,DPL
      dec     64h
      LCALL   CLEAR
      LCALL   LINE_LL1
      LCALL   SECOND_LINE
      LCALL   SPACE_2
      MOV     A,60H
      JZ      DISP_5
disp_1:  LCALL   DOWN_2
      mov     a,60h
      jnz     disp_6
      ljmp    disp_7
 disp_6: lcall   space_2
      LCALL   NUMBER_1
 disp_7:  LCALL   D_LAY_2
      lcall   d_lay_2
      LCALL   CLEAR
      LCALL   DeLAY
      mov     dph,63h
      mov     dpl,64h
      movx    a,@dptr
      Ljmp    disp_4
disp_5:  LCALL   D_LAY_2
      lcall   d_lay_2
      lcall   CLEAR
      LCALL   DELAY
      LCALL   SECOND_LINE
      LCALL   DELAY
      MOV     DPTR,#6000H
      MOV     A,#0C8H
      MOVX    @DPTR,A
      LCALL   DELAY
      mov     dptr,#end_0
      lcall   write_ch
      LCALL   DELAY
      MOV     DPTR,#6000H
      MOV     A,#0C7H
      MOVX    @DPTR,A
      LCALL   DELAY
```

```
        mov     dptr,#end_1
        lcall   write_ch
        LCALL   DELAY
        MOV     DPTR,#8000H
        MOV     A,#0C8H
        MOVX    @DPTR,A
        LCALL   DELAY
        mov     dptr,#end_2
        lcall   write_ch
        LCALL   D_LAY_2
        LCALL   CLEAR
        LCALL   DELAY
disp_3: POP     02
        POP     01
        POP     00
        ret


LINE_LL1:       PUSH    00
        PUSH    01
        PUSH    02
        MOV     R1,#2H
LINE_L1:    MOV     DPH,61H
        MOV     DPL,62H
        MOVX    A,@DPTR
        MOV     R0,A
        SWAP    A
        ANL     A,#0FH
        MOV     DPTR,#DISP_ADDR
        MOVC    A,@A+DPTR
        LCALL   DELAY
        MOV     DPTR,#8002H
        MOVX    @DPTR,A
        lcall   dElay
        MOV     A,R0
        ANL     A,#0FH
        MOV     DPTR,#DISP_ADDR
        MOVC    A,@A+DPTR
        MOV     DPTR,#8002H
```

```
        MOVX    @DPTR,A
        LCALL   DELAY
        inc     62h
        MOV     A,62H
        cjne    A,#0h,line_l2
        inc     61h
line_l2: DJNZ   R1,LINE_L1
        POP     02
        POP     01
        POP     00
        RET

DOWN_2:         PUSH    00
        PUSH    01
        PUSH    02
down_n2:    MOV     DPH,63H
        MOV     DPL,64H
        MOVX    A,@DPTR
        CJNE    A,#11H,NOT_CNT2
        LCALL   TO_CNT2
        LJMP    OUT__1
NOT_CNT2: CJNE  A,#12H,NOT_TIM2
        LCALL   TO_TIM2
        LJMP    OUT__1
NOT_TIM2: CJNE  A,#14H,NOT_LD2
        LCALL   TO_LD2
        LJMP    OUT__1
NOT_LD2:  CJNE  A,#15H,NOT_AND2
        LCALL   TO_AND2
        LJMP    OUT__1
NOT_AND2: CJNE  A,#16H,NOT_FUN2
        LCALL   TO_FUN2
        LJMP    OUT__1
NOT_FUN2: CJNE  A,#18H,NOT_OR2
        LCALL   TO_OR2
        LJMP    OUT__1
NOT_OR2:  CJNE  A,#17H,NOT_NOT2
        LCALL   TO_NOT2
        LJMP    OUT__1
```

```
      NOT_NOT2: CJNE    A,#13H,NOT_OUT2
            LCALL   TO_OUT2
      OUT__1:  LCALL    DeLAY
            dec     64h
            MOV     A,64H
            cjne    A,#0ffh,down_n1
            dec     63h
      down_n1: DEC      60H
            mov     a,60h
            jnz     down_n2
      not_out2: POP     02
            POP     01
            POP     00
            RET

NUMBER_1:         PUSH    00
          PUSH    01
          PUSH    02
          mov     r2,#4h
    number_n2:    MOV     DPH,63H
          MOV     DPL,64H
          MOVX    A,@DPTR
          MOV     R1,A
          CJNE    A,#1H,NOT__1
          LJMP    NUMBER_N1
     NOT__1:    CJNE    A,#2H,NOT__0
          LJMP    NUMBER_N1
     NOT__0:    CJNE    A,#3H,NOT__5
          LJMP    NUMBER_N1
     NOT__5:    CJNE    A,#4H,NOT__4
          LJMP    NUMBER_N1
     NOT__4:    CJNE    A,#5H,NOT__9
          LJMP    NUMBER_N1
     NOT__9:    CJNE    A,#6H,NOT__C
          LJMP    NUMBER_N1
     NOT__C:    CJNE    A,#7H,NOT__D
          LJMP    NUMBER_N1
     NOT__D:    CJNE    A,#8H,NOT__8
          LJMP    NUMBER_N1
```

```
NOT__8:   CJNE    A,#9H,NOT__3
          LJMP    NUMBER_N1
NOT__3:   CJNE    A,#0AH,NOT__2
          LJMP    NUMBER_N1
NOT__2:   CJNE    A,#0BH,NOT__7
          LJMP    NUMBER_N1
NOT__7:   CJNE    A,#0CH,NOT__6
          LJMP    NUMBER_N1
NOT__6:   CJNE    A,#0DH,NOT__E
          LJMP    NUMBER_N1
NOT__E:   CJNE    A,#0EH,NOT__E
          LJMP    NUMBER_N1
NOT__E:   CJNE    A,#0FH,NOT__F
          LJMP    NUMBER_N1
NOT__F:   CJNE    A,#10H,NOT___A
          LJMP    NUMBER_N1
not___a:  cjne    a,#3ah,not___a1
          MOV     DPTR,#6002H
          MOV     A,#3AH
          MOVX    @DPTR,A
          INC     R2
          SJMP    NO_DO_TO
NUMBER_N1: LCALL  DELAY
          MOV     A,R1
          MOV     R0,#2FH          ; SET ADDRESS IN RAM = 30H
          ADD     A,R0
          MOV     R0,A
          MOV     A,@R0
          mov     dptr,#6002h
          MOVX    @DPTR,A
NO_DO_TO:  LCALL  DeLAY
          dec     64h
          MOV     A,64H
          cjne    A,#0ffH,num_n1
          dec     63h
num_n1:   DEC     60H
          mov     a,60h
          jz      not___a1
          djnz    r2,num_w1
```

```
        lcall   space_2
num_w1:     ljmp    number_n2
NOT___A1:
EXIT__3:    POP     02
            POP     01
            POP     00
            RET


TO_LD2:     MOV     DPTR,#LET1
            LCALL   WRITE_CH
            RET

TO_OUT2:    MOV     DPTR,#LET2
            LCALL   WRITE_CH
            RET

TO_AND2:    MOV     DPTR,#LET3
            LCALL   WRITE_CH
            RET

TO_OR2:     MOV     DPTR,#LET4
            LCALL   WRITE_CH
            RET

TO_FUN2:    MOV     DPTR,#LET8
            LCALL   WRITE_CH
            RET

TO_CNT2:    MOV     DPTR,#LET6
            LCALL   WRITE_CH
            RET

TO_TIM2:    MOV     DPTR,#LET7
            LCALL   WRITE_CH
            RET

TO_NOT2:    MOV     DPTR,#NOT_KEY
            LCALL   WRITE_CH
```

```
        RET
;=================================================

TABLE:      DFB    31H,30H,35H,34H,39H,43H,44H,38H
        DFB    33H,32H,37H,36H,42H,45H,46H,41H
        DFB    41H,32H,4FH,34H,4BH,46H,4EH,4FH
        DFB    53H,43H,20H,20H,20H,20H,20H,20H
        DFB    20H,52H,43H,44H,45H,44H,4AH,49H
           ;---- NUMBER TABLE -----------
MOD_LET:    DFB    20H,53H,45H,4CH,45H,43H,54H,20H,4DH,4FH
        DFB    44H,45H,20H,31H,20H,32H,20H,33H,20H,24H
           ;------   MODE SELECT ----------
MOD_2LET:   DFB    20H,50H,28H,31H,29H,20H,52H,28H,32H
        DFB    29H,20H,4DH,28H,33H,29H,20H,24H
           ;-------  P(1) R(2) M(3)  ----------
ERR_SEL:    DFB    20H,45H,52H,52H,4FH,52H,20H,53H,45H
        DFB    4CH,45H,43H,54H,20H,24H
           ;------  ERROR SELECT  -------------
ERR1:       DFB    20H,2AH,2AH,2AH,20H,45H,52H,52H
        DFB    4FH,52H,20H,2AH,2AH,2AH,20H,24H
           ;--- *** ERROR *** ----
LET1:       DFB    0FEH,4CH,44H,24H                ;---  LD  ----
LET2:       DFB    0FEH,4FH,55H,54H,24H            ;---  OUT  ----
LET3:       DFB    0FEH,41H,4EH,44H,24H            ;---  AND  ----
LET4:       DFB    0FEH,4FH,52H,24H                ;---  OR  ----
LET5:       DFB    0FEH,4EH,4FH,54H,24H            ;---  NOT  ----
LET6:       DFB    0FEH,43H,4EH,54H,24H            ;---  CNT  ----
LET7:       DFB    0FEH,54H,49H,4DH,24H            ;---  TIM  ----
LET8:       DFB    0FEH,46H,55H,4EH,24H            ;---  FUN  ----
LET9:       DFB    0FEH,20H,20H,57H,52H,20H,20H,24H    ;---  WR  ----
LET10:      DFB    20H,2AH,2AH,20H,4FH,2EH,4BH,2EH,20H,2AH,2AH,24H
           ;----- ** O.K. ** ------
LET11:      DFB    20H,30H,30H,30H,30H,24H
           ;-------  0000  --------
LET12:      DFB    20H,20H,52H,55H,4EH,20H,4DH,4FH,44H,45H,20H,20H,24H
           ;--------  RUN MODE  -------
LET13:      DFB    20H,20H,4DH,4FH,4EH,49H,54H,4FH
        DFB    52H,20H,4DH,4FH,44H,45H,20H,24H
```

```
          ;--------  MONITOR MODE ---------
LET14:      DFB    20H,50H,4CH,43H,20H,3DH,3EH,20H,50H,52H
        DFB    4FH,4AH,45H,43H,54H,20H,42H,59H,20H,24H
           ;--------  PROJECT PRESENT BY  ---------
LET15:      DFB    20H,4EH,41H,52H,4FH,4EH,47H,20H,2CH
        DFB    20H,54H,48H,41H,57H,41H,54H,20H,24H
           ;--------  MABONG AND THAWAT  ----------
LET16:      DFB    20H,50H,41H,53H,53H,57H,4FH
        DFB    52H,44H,20H,3AH,20H,24H
           ;---------  PASSWORD  ---------------
LET17:      DFB            3CH,4FH,2EH,4BH,2EH,3EH,24H
LET18:      DFB          3CH,20H,4FH,2EH,4BH,2EH,20H,3EH,24H
LET19:      DFB      3CH,20H,20H,4FH,2EH,4BH,2EH,20H,20H,3EH,24H
DISP_ADDR:  DFB      30H,31H,32H,33H,34H,35H,36H,37H,38H   ;--  01234567  ---
        DFB    39H,41H,42H,43H,44H,45H,46H,24H        ;--  89ABCDEF  ---
JMP_LET:    DFB    20H,4AH,55H,4DH,50H,20H,54H,4FH,20H,7EH,20H,24H


RAM_LOAD:   DFB    20H,52H,41H,4DH,20H,20H,4CH,4FH,41H,44H,20H
        DFB    20H,20H,0B0H,3EH,20H,20H,4FH,2EH,4BH,2EH,20H,24H
           ;-------  RAM  LOAD   -)  O.K.


END_0:      DFB            3CH,45H,4EH,44H,3EH,24H
END_1:      DFB          3CH,20H,45H,4EH,44H,20H,3EH,24H
END_2:      DFB      3CH,20H,20H,45H,4EH,44H,20H,20H,3EH,24H

PROGRAM_MODE:       DFB  20H,20H,50H,52H,4FH,47H,52H,41H,4DH,20H
            DFB  4DH,4FH,44H,45H,20H,20H,24H


              ;-------  PROGRAM MODE ----------
START_1:    DFB      3CH,20H,53H,54H,41H,52H,54H,20H,3EH,24H
           ;----------  (  START  )  -------------
NOT_KEY:        DFB    0FEH,4EH,4FH,54H,24H
           ; ------  NOT  ----------

;=================================================================
;                    END OF FILE
;=================================================================
```

ภาคผนวก ง.
โปรแกรมของส่วนหน่วยประมวลผลกลาง

```
        CPU     "8051.TBL"
        HOF     "INT8"
;
;*****************************************************
;
;MCS-51 INTERNAL REGISTERS
;
B:      EQU     0F0H    ;B REGISTER
ACC:    EQU     0E0H    ;ACCUMULATOR
PSW:    EQU     0D0H    ;PROGRAM STATUS WORD
IPC:    EQU     0B8H    ;INTERRUPT PRIORITY
P3:     EQU     0B0H    ;PORT 3
IEC:    EQU     0A8H    ;INTERRUPT ENABLE
P2:     EQU     0A0H    ;PORT 2
SBUF:   EQU     99H     ;SEND BUFFER
SCON:   EQU     98H     ;SERIAL CONTROL
P1:     EQU     90H     ;PORT 1
PCON:   EQU     87H     ;POWER CONTROL REGISTER
TCON:   EQU     88H     ;TIMER CONTROL REGISTER
TMOD:   EQU     89H     ;TIMER MODE CONTROL
TH0:    EQU     8CH     ;TIMER 0 HIGH
TL0:    EQU     8AH     ;TIMER 0 LOW
TH1:    EQU     8DH     ;TIMER 1 HIGH
TL1:    EQU     8BH     ;TIMER 1 LOW
DPH:    EQU     83H     ;DATA POINTER HIGH
DPL:    EQU     82H     ;DATA POINTER LOW
SP:     EQU     81H     ;STACK POINTER
P0:     EQU     80H     ;PORT 0

; BIT ADDRESS
C:      EQU     0D7H    ;CARRY FLAG
ES:     EQU     0ACH    ;ENABLE SERIAL INTERRUPT
TI:     EQU     099H    ;TRANSMIT INTERRUPT FLAG
EA:     EQU     0AFH    ;ENABLE ALL INTERRUPT
REN:    EQU     09CH    ;SERIAL RECEPTION ENABLE
TB8:    EQU     09BH    ;TRANSMIT BIT 8
RB8:    EQU     09AH    ;RECIEVE BIT 8
TF0:    EQU     08DH    ;TIMER 0 OVERFLOW FLAG
TF1:    EQU     08FH    ;TIMER 1 OVERFLOW FLAG
```

```
TR0:    EQU     08CH    ;TIMER 0 RUN  (NO/OFF) CONTROL BIT
TR1:    EQU     08EH    ;TIMER 1 RUN  (ON/OFF) CONTROL BIT
IE0:    EQU     089H    ;EXT INTERRUPT 0 EDGE FLAG
IE1:    EQU     08BH    ;EXT INTERRUPT 1 EDGE FLAG
IT0:    EQU     088H    ;EXT INTERRUPT 0 TYPE FLAG
IT1:    EQU     08AH    ;EXT INTERRUPT 1 TYPE FLAG

P0.0:   EQU     080H
P0.1:   EQU     081H
P0.2:   EQU     082H
P0.3:   EQU     083H
P0.4:   EQU     084H
P0.5:   EQU     085H
P0.6:   EQU     086H
P0.7:   EQU     087H

P1.0:   EQU     090H
P1.1:   EQU     091H
P1.2:   EQU     092H
P1.3:   EQU     093H
P1.4:   EQU     094H
P1.5:   EQU     095H
P1.6:   EQU     096H
P1.7:   EQU     097H
;-------LOW 16 BYTE OR 128 DIRECT BIT ADDRESS --
; ADDRESS 20H -2FH INTERNAL RAM
;  0H -7H = 20.0-20.7
;  8H -FH = 21.0-21.7
;  OR 00H -7FH
;
;RUN OPERATION REGISTER BIT  FOR PLC
;
R_REG:      EQU  00H   ;RESULT REGISTER BIT
TEMP:       EQU  01H   ;TEMPORARY BIT OPERATION
S_REG:      EQU  02H   ;STACK REGISTER BIT
TFLAG:      EQU  33H   ;TIMER FLAG FOR CHECK PRESET VAL LOAD YET?
CFLAG:      EQU  34H   ;COUNTER FLAG FOR CHECK PRESET COUNT LOAD YET?
;
;INTERNAL INPUT RELAY NUMBER XX:YY
```

```
;
RL0000:     EQU  03H   ;INTER INPUT RELAY
RL0001:     EQU  04H   ;CHANNEL 00H
RL0002:     EQU  05H   ;BIT 00H-0FH(0-15)
RL0003:     EQU  06H
RL0004:     EQU  07H
RL0005:     EQU  08H
RL0006:     EQU  09H
RL0007:     EQU  0AH
;
;INTERNAL OUTPUT RELAY NUMBER XX:YY
;
RL0400:     EQU  0BH
RL0401:     EQU  0CH
RL0402:     EQU  0DH
RL0403:     EQU  0EH
RL0404:     EQU  0FH
RL0405:     EQU  10H
RL0406:     EQU  11H
RL0407:     EQU  12H
;
;TIM/CNT RELAY NUMBER XX
;
TIM00:      EQU   13H
TIM01:      EQU   14H
TIM02:      EQU   15H
TIM03:      EQU   16H
TIM04:      EQU   17H
TIM05:      EQU   18H
TIM06:      EQU   19H
TIM07:      EQU   1AH
CNT08:      EQU   1BH
CNT09:      EQU   1CH
CNT10:      EQU   1DH
CNT11:      EQU   1EH
CNT12:      EQU   1FH
CNT13:      EQU   20H
CNT14:      EQU   21H
CNT15:      EQU   22H
```

```
;
;INTERNAL  AUXILARY RELAY  XX:YY
;
ARL0900:     EQU   23H    ;CHANNEL 09
ARL0901:     EQU   24H    ;BIT 0-15(0-FH)
ARL0902:     EQU   25H
ARL0903:     EQU   26H
ARL0904:     EQU   27H
ARL0905:     EQU   28H
ARL0906:     EQU   29H
ARL0907:     EQU   2AH
ARL0908:     EQU   2BH
ARL0909:     EQU   2CH
ARL0910:     EQU   2DH
ARL0911:     EQU   2EH
ARL0912:     EQU   2FH
ARL0913:     EQU   30H
ARL0914:     EQU   31H
ARL0915:     EQU   32H
;TIMER/COUNTER FLAG CHECK
TFLAG0:      EQU   35H
TFLAG1:      EQU   36H
TFLAG2:      EQU   37H
TFLAG3:      EQU   38H
TFLAG4:      EQU   39H
TFLAG5:      EQU   3AH
TFLAG6:      EQU   3BH
TFLAG7:      EQU   3CH
CFLAG08:     EQU   3DH
CFLAG09:     EQU   3EH
CFLAG10:     EQU   3FH
CFLAG11:     EQU   40H
CFLAG12:     EQU   41H
CFLAG13:     EQU   42H
CFLAG14:     EQU   43H
CFLAG15:     EQU   44H
;---------------------------------------------------------------------
;!!!!!! DEFINE REGISTER FOR RUN MODE          (BYTE)
TC_REG_H:    EQU    5FH
```

```
CNT_NUM:    EQU    60H      ;SAVE VALUE FROM OPERATION RUN MODE
TC_REG_L:   EQU    61H      ;SAVE DATA TEMPORARY FROM RESULT REGISTER
RELAY_NUM:  EQU    62H      ;SPECIFIED RELAY NUMBER
CH_NUM:     EQU    63H      ;CHANNEL NUMBER
OUTP_REG:   EQU    64H      ;OUTPUT REGISTER
TIM_NUM:    EQU    65H      ;TIMER  NUMBER

T0PSET_H:   EQU    66H
T0PSET_L:   EQU    67H
T1PSET_H:   EQU    68H
T1PSET_L:   EQU    69H
T2PSET_H:   EQU    6AH
T2PSET_L:   EQU    6BH
T3PSET_H:   EQU    6CH
T3PSET_L:   EQU    6DH
T4PSET_H:   EQU    6EH
T4PSET_L:   EQU    6FH
T5PSET_H:   EQU    70H
T5PSET_L:   EQU    71H
T6PSET_H:   EQU    72H
T6PSET_L:   EQU    73H
T7PSET_H:   EQU    74H
T7PSET_L:   EQU    75H

;-------------------------------------------------------------------------
        ORG 0H          ;POWER ON
        AJMP 100H

        ORG 3H          ;EXTERNAL0 INTERRUPT ROUTINE
        LJMP EX_INT0

        ORG 0BH         ;INTERRUPT FOR COUNTER
        LJMP ISR_T/H0

        ORG 13H
        LJMP EX_INT1    ;EXTERNAL1 INTERRUPT ROUTINE

        ORG 23H
        LJMP ISR_232    ;SERIAL INTERRUPT ROUTINE
```

```
;-----------------------------
; INITAIL VALUE
;-----------------------------
        ORG 100H        ;START MAIN PLC
START:
        ACALL DELAY     ;INITIAL

;
;INITIAL PORT I/O
;
        MOV DPTR,#8083H  ;DEFINE PORT output
        MOV A,#80H
        MOVX @DPTR,A
        mov dptr,#8003h  ;DEFINE PORT input
        mov a,#9bh
        movx @dptr,a

; INITIAL FOR SERIAL
;
  init:
      mov iec,#94h   ;enable external interrupt1,enable serial port
      mov ipc,#10h    ;#10 define the serial port high priority level
      mov pcon,#0h   ;set SMOD = 0
      mov tcon,#40h  ;timer 1 --> ON
      mov tmod,#20h  ;SET timer1 -> mode2 8 bit auto-reload
      mov scon,#50h  ;for single processor - mode 1 baudrate variable
      mov th1,#0E8h  ;Fosc=11.0592 Mhz ,baudrate = 1200 bps
      MOV TL1,#0H
      CLR SBUF        ;CLEAR SERIAL BUFFER
;
;INITIAL FOR TIMER0 -- TIME BASE OF SYSTEM CONTROL
;USE INTERNAL RAM ADDRESS 60H,61H
;

      ORL IEC,#82H    ;ENABLEBLE TIMER0
      ORL IPC,#2H     ;SET TIMER0 HIGH PRIORITY
      ORL TMOD,#1H    ;SOFTWARE CONTROL TIMER0 MODE1  16BIT TIMER
      ORL TCON,#10H   ;TIMER0 ->ON AND CLEAR TIMER0 FLAG
      MOV TH0,#0F0H   ;LOAD TIMER VALUE HIGH
```

```
        MOV TL0,#0H     ;LOAD TIMER VALUE LOW

        ;CLR BIT0
        MOV 60H,#0H     ;CLEAR TIMER STEP
        MOV 61H,#80H
        LJMP SERIAL
;-------------------------------------------------------------------
;CHECK MODE TO OPERATE PROCESS THAT YOU HAVE IT DONE
;                   (1) PROGRAM MODE
;                   (2) RUN MODE
;                   (3) MONITOR MODE
; ONE OF THREE MODE IS RUN AT THE CURRENT TIME
;-------------------------------------------------------------------
CHECK_MOD:
                ;CHECK MODE IN MAIN PROGRAM FROM LCD MODULE
         MOV A,SBUF
         CJNE A,#0,CHECK_SKIP
         SJMP CHECK_MOD
 CHECK_SKIP:
         ; MOV A,SBUF
         ; CJNE A,#26H,CHECK_SKIP
         ; CLR SBUF
         MOV A,SBUF                     ;SERIAL COMMUNICATION
         CJNE A,#1H,CHECK_SKIP1
         LCALL PROG_MODE                ;(1)TO PROGRAM MODE
         SJMP CHECK_MOD
CHECK_SKIP1:
         CJNE A,#2H,CHECK_SKIP2
         LCALL RUN_MODE                 ;(2)TO RUN MODE
         SJMP CHECK_MOD
CHECK_SKIP2:
         CJNE A,#3H,CHECK_MOD
         LCALL MON_MODE                 ;(3)TO MONITOR MODE
         SJMP CHECK_MOD
;-------------------------------------------------------------------
;@@@@@@@@@@@@@@@@@@  PROGRAM MODE @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
;-------------------------------------------------------------------
PROG_MODE:
```

```
        ;CHECK SERIAL
        ;GET DATA DOWN LOAD FROM LCD.MODULE
        ;FOR GET IT TO RUN

        RET
;-------------------------------------------------------------------
; ******************        RUN MODE       ***************************
;-------------------------------------------------------------------
RUN_MODE:
      MOV DPTR,#2000H           ;START SCAN I/P ADDRESS #2000H
      MOVX A,@DPTR
      CJNE A,#14H,RUN_MOD_ERR   ;FIRST ADDRESS MUST HAVE "LD"
      LCALL RUN_LD              ;YES
RUN_AGAIN:
      INC DPTR                  ;NEXT ADDRESS
      MOVX A,@DPTR
      CJNE A,#15H,RUN_SKIP1
      LCALL RUN_AND             ;"AND INSTRUCTION "
      SJMP RUN_AGAIN
RUN_SKIP1:
      CJNE A,#18H,RUN_SKIP2     ;"OR INSTRUCTION "
      LCALL RUN_OR
      SJMP RUN_AGAIN
RUN_SKIP2:
      CJNE A,#13H,RUN_SKIP3     ;"OUT INSTRUCTION "
      LCALL RUN_OUT
      SJMP RUN_AGAIN
RUN_SKIP3:
      CJNE A,#12H,RUN_SKIP4     ;"TIM INSTRUCTION "
      LCALL RUN_TIM
      SJMP RUN_AGAIN
RUN_SKIP4:
      CJNE A,#11H,RUN_SKIP5     ;"CNT INSTRUCTION"
      LCALL RUN_CNT
      SJMP RUN_AGAIN
RUN_SKIP5:
      CJNE A,#14H,RUN_SKIP6     ;"LD INSTRUCTION"
      LCALL RUN_LD
      SJMP RUN_AGAIN
```

```
RUN_SKIP6:
        CJNE A,#0AH,RUN_MOD_ERR ;" CR -> END "
        LCALL WATCHDOG
        ;CHECK OUT OF MODE Y/N ?
        ;CHECK SERIAL FOR RETURN OUT TO CHECK MODE MAIN PROGRAM
        ;IF CHANGE MODE THEN CHANGE_RUN
        LJMP RUN_MODE


RUN_MOD_ERR:
         ;SEND ERROR TO LCD MODULE

         ;CHANGE_RUN: -> RETURN TO CHECK MODE

        RET
;-------------------------------------------------------
RUN_LD:
         INC DPTR
         MOVX A,@DPTR
         CJNE A,#17H,RUNLD_SKIP   ;"LD NOT ?"
         INC DPTR                 ;YES
         MOVX A,@DPTR
         CJNE A,#12H,RUNLD_SKIP1  ;"LD NOT TIM ?"
         LCALL LD_NOT_TIM_        ;YES
         SJMP RUNLD_RET
RUNLD_SKIP1:   CJNE A,#11H,RUNLD_SKIP2  ;"LD NOT CNT?"
         LCALL LD_NOT_CNT_        ;YES
         SJMP RUNLD_RET
RUNLD_SKIP2:   LCALL DEC_DPTR
         LCALL LD_NOT_
         SJMP RUNLD_RET
RUNLD_SKIP:    CJNE A,#12H,RUNLD_SKIP3 ;"LD TIM"
         LCALL LD_TIM_
         SJMP RUNLD_RET
RUNLD_SKIP3:   CJNE A,#11H,RUNLD_SKIP4  ;"LD CNT"
         LCALL LD_CNT_
         SJMP RUNLD_RET
RUNLD_SKIP4:   LCALL DEC_DPTR
         LCALL LD_
```

```
RUNLD_RET:
        RET

;--- DEC DPTR --------
DEC_DPTR:
        MOV A,DPL               ;DEC DPTR
        CJNE A,#0H,RL_JMP
        DEC DPH
        MOV DPL,#0FFH
        SJMP RL_JMP1
   RL_JMP:
        DEC DPL
   RL_JMP1:
        RET

;SUBROUTINE OF RUN_LD ROUTINE
;

LD_:
      INC DPTR
      MOVX A,@DPTR
      MOV CH_NUM,A          ;GET CHANNEL
      INC DPTR
      MOVX A,@DPTR
      MOV RELAY_NUM,A       ;GET RELAY NUMBER
      LCALL GET_RL_BIT      ;BIT IN CARRY FLAG
                  ;SAVE RESULT IN STACK REGISTER BIT
      MOV TEMP ,C
      MOV C,R_REG
      MOV S_REG,C
      MOV C,TEMP
      MOV R_REG,C
                  ;STORE INPUT BIT SPECIFIED IN RESULT REGISTER
      LCALL SAV_IN_IP       ;SAVE STATUS TO INTERNAL I/P RELAY

      RET
;------------
;FOR SAVE STATUS I/P BIT TO INTERNAL I/P RELAY REG.
;     INPUT CH_NUM,RELAY_NUM,R_REG(BIT)
```

```
;       OUTPUT NONE
;       USED AFTER CALL  GET_RL_BIT,MOV TEMP<-C
;       FOR CH 00-03 BIT 0-FH
SAV_IN_IP:
        MOV A,CH_NUM
        CJNE A,#0H,SAV_OUT
        MOV A,RELAY_NUM
        CJNE A,#0H,SAV_SK
        MOV RL0000,TEMP
        AJMP SAV_OUT
SAV_SK:     CJNE A,#1H,SAV_SK1
        MOV RL0001,TEMP
        AJMP SAV_OUT
SAV_SK1:    CJNE A,#2H,SAV_SK2
        MOV RL0002,TEMP
        AJMP SAV_OUT
SAV_SK2:    CJNE A,#3H,SAV_SK3
        MOV RL0003,TEMP
        AJMP SAV_OUT
SAV_SK3:    CJNE A,#4H,SAV_SK4
        MOV RL0004,TEMP
        AJMP SAV_OUT
SAV_SK4:    CJNE A,#5H,SAV_SK5
        MOV RL0005,TEMP
        AJMP SAV_OUT
SAV_SK5:    CJNE A,#6H,SAV_SK6
        MOV RL0006,TEMP
        AJMP SAV_OUT
SAV_SK6:    CJNE A,#7H,SAV_OUT
        MOV RL0007,TEMP
SAV_OUT:

        RET
;-----------------
;GET CONTENT OF SPECIFIED RELAY NUMBER  (I/P)-BIT
; INPUT -CH_NUM
;       -RELAY_NUM
; OUTPUT -CARRY FLAG
;----------------
```

```
GET_RL_BIT:
        PUSH DPH
        PUSH DPL
        PUSH 00
        MOV A,CH_NUM
        CJNE A,#4H,GET_SKIP     ;INPUT CHANNEL 00-03
        AJMP GET_SKIP3
GET_SKIP:     JNC GET_SKIP3
        MOV A,RELAY_NUM
        CJNE A,#0FH,GET_SKIP1   ;RELAY BIT 00-15(0-FH)
GET_SKIP2:    MOV A,CH_NUM
        CJNE A,#0H,G_SK
        MOV DPTR,#8000H
        AJMP G_CONT
G_SK:         CJNE A,#1H,G_SK1
        MOV DPTR,#8020H
        AJMP G_CONT
G_SK1:        CJNE A,#2H,G_SK2
        MOV DPTR,#8040H
        SJMP G_CONT
G_SK2:        MOV DPTR,#8060H
G_CONT:       MOV R0,RELAY_NUM
        MOVX A,@DPTR
        CJNE R0,#0H,D_GET
        SJMP D_GET1
D_GET:        RRC A
        DJNZ R0,D_GET
D_GET1:       RRC A
        SJMP GET_OUT
GET_SKIP1:    JC GET_SKIP2
GET_SKIP3:
        ;CHECK INTERNAL AUXILARY RELAY OR NOT?
        ;IF YES THEN  RETURN INTERNAL VALUE TO CARRY
        MOV A,CH_NUM
        CJNE A,#9H,GET_OUT
        MOV A,RELAY_NUM
        CJNE A,#0H,GETSKIP
        MOV C,ARL0900
        AJMP GET_OUT
```

```
GETSKIP:        CJNE A,#1H,GETSKIP1
                MOV C,ARL0901
                AJMP GET_OUT
GETSKIP1:       CJNE A,#2H,GETSKIP2
                MOV C,ARL0902
                AJMP GET_OUT
GETSKIP2:       CJNE A,#3H,GETSKIP3
                MOV C,ARL0903
                AJMP GET_OUT
GETSKIP3:       CJNE A,#4H,GETSKIP4
                MOV C,ARL0904
                AJMP GET_OUT
GETSKIP4:       CJNE A,#5H,GETSKIP5
                MOV C,ARL0905
                AJMP GET_OUT
GETSKIP5:       CJNE A,#6H,GETSKIP6
                MOV C,ARL0906
                AJMP GET_OUT
GETSKIP6:       CJNE A,#7H,GETSKIP7
                MOV C,ARL0907
                AJMP GET_OUT
GETSKIP7:       CJNE A,#8H,GETSKIP8
                MOV C,ARL0908
                AJMP GET_OUT
GETSKIP8:       CJNE A,#9H,GETSKIP9
                MOV C,ARL0909
                AJMP GET_OUT
GETSKIP9:       CJNE A,#0AH,GETSKIP10
                MOV C,ARL0910
                AJMP GET_OUT
GETSKIP10:      CJNE A,#0BH,GETSKIP11
                MOV C,ARL0911
                AJMP GET_OUT
GETSKIP11:      CJNE A,#0CH,GETSKIP12
                MOV C,ARL0912
                AJMP GET_OUT
GETSKIP12:      CJNE A,#0DH,GETSKIP13
                MOV C,ARL0913
                AJMP GET_OUT
```

```
GETSKIP13:    CJNE A,#0EH,GETSKIP14
              MOV C,ARL0914
              AJMP GET_OUT
GETSKIP14:    MOV C,ARL0915

 GET_OUT:
              ;RETURN BIT PORT VALUE -->CARRY FLAG
              POP 00
              POP DPL
              POP DPH
              RET
;-------------
LD_MOT_:
              INC DPTR
              MOVX A,@DPTR
              MOV CH_NUM,A          ;GET CHANNEL
              INC DPTR
              MOVX A,@DPTR
              MOV RELAY_NUM,A       ;GET BIT SPECIFIED
              LCALL GET_RL_BIT
              CPL C                 ; "NOT"
              MOV TEMP,C
              MOV C,R_REG
              MOV S_REG,C           ;SAVE STATUS
              MOV C,TEMP
              MOV R_REG,C           ;STORE
              LCALL SAV_IN_IP

              RET
;--------------------
LD_TIM_:
                   ;LOAD TIMER
              INC DPTR
              MOVX A,@DPTR                ;GET TIMER NUMBER
              MOV TIM_NUM,A
              LCALL GETTIM_BIT_RL
              MOV TEMP,C
              MOV C,R_REG
              MOV S_REG,C
```

```
        MOV C,TEMP
        MOV R_REG,C

        RET
;--------------
LD_CNT_:
            ;LOAD COUNTER
        INC DPTR
        MOVX A,@DPTR
        MOV CNT_NUM,A
        LCALL GETCNT_BIT_RL
        MOV TEMP,C
        MOV C,R_REG
        MOV S_REG,C
        MOV C,TEMP
        MOV R_REG,C

        RET
;-------------
LD_NOT_TIM_:
            ;LOAD NOT TIMER
        INC DPTR
        MOVX A,@DPTR
        MOV TIM_NUM,A
        LCALL GETTIM_BIT_RL
        MOV TEMP,C
        MOV C,R_REG
        MOV S_REG,C
        MOV C,TEMP
        CPL C                   ;"LD TIM NOT"
        MOV R_REG,C

        RET
;-------------
LD_NOT_CNT_:
        INC DPTR
        MOVX A,@DPTR
        MOV CNT_NUM,A
        LCALL GETCNT_BIT_RL
```

```
        CPL C
        MOV TEMP,C
        MOV C,R_REG
        MOV S_REG,C
        MOV C,TEMP
        MOV R_REG,C

        RET
;-------------------------------------------------------------
RUN_AND:
        INC DPTR
        MOVX A,@DPTR
        CJNE A,#17H,ANDLD_SKIP   ;"AND NOT ? "
        INC DPTR
        MOVX A,@DPTR
        CJNE A,#12H,ANDLD_SKIP1   ;"AND NOT TIM ? "
        LCALL AND_NOT_TIM_
        SJMP AND_RET
ANDLD_SKIP1: CJNE A,#11H,ANDLD_SKIP2    ;"AND NOT CNT ? "
        LCALL AND_NOT_CNT_
        SJMP AND_RET
ANDLD_SKIP2: LCALL DEC_DPTR
        LCALL AND_NOT_
        SJMP AND_RET
ANDLD_SKIP:  CJNE A,#12H,ANDLD_SKIP3     ;"AND TIM ?"
        LCALL AND_TIM_
        SJMP AND_RET
ANDLD_SKIP3: CJNE A,#11H,ANDLD_SKIP4     ;"AND CNT ? "
        LCALL AND_CNT_
        SJMP AND_RET
ANDLD_SKIP4: CJNE A,#14H,ANDLD_SKIP5   ;"AND LOAD ? "
        LCALL AND_LD_
        SJMP AND_RET
ANDLD_SKIP5: LCALL DEC_DPTR
        LCALL AND_
 AND_RET:
        RET
;-----------------------------
;SUBROUTINE OF RUN AND ROUTINE
```

```
;
AND_:
     INC DPTR
     MOVX A,@DPTR          ;GET CHANNEL
     MOV CH_NUM,A
     INC DPTR
     MOVX A,@DPTR          ;GET BIT NUMBER
     MOV  RELAY_NUM,A
     LCALL GET_RL_BIT
     MOV TEMP,C            ;MOV BIT RELAY NUMBER XX:YY
     LCALL SAV_IN_IP       ;STORE BIT TO INTERNAL I/P RELAY REG.
     MOV C,R_REG
     ANL C,TEMP            ;AND LOGIC BIT
     MOV R_REG,C

     RET
;------------
AND_NOT_:
     INC DPTR
     MOVX A,@DPTR          ;GET CHANNEL
     MOV CH_NUM,A
     INC DPTR
     MOVX A,@DPTR          ;GET BIT NUMBER
     MOV  RELAY_NUM,A
     LCALL GET_RL_BIT
     MOV TEMP,C            ;MOV BIT RELAY NUMBER XX:YY
     LCALL SAV_IN_IP       ;STORE BIT TO INTERNAL I/P RELAY REG.
     MOV C,R_REG
     ANL C,/TEMP           ;AND NOT LOGIC BIT
     MOV R_REG,C

     RET
;----------
AND_NOT_TIM_:
     INC DPTR
     MOVX A,@DPTR
     MOV TIM_NUM,A
     LCALL GETTIM_BIT_RL
     MOV TEMP,C
```

```
        MOV C,R_REG
        ANL C,/TEMP

        RET
    ;-------------------
    ;GET TIMER BIT RELAY
    ; INPUT     TIM_NUM
    ; OUTPUT    CARRY FLAG(BIT SPECIFIED)
    GETTIM_BIT_RL:
              MOV A,TIM_NUM
              CJNE A,#0H,GTIMSKIP
              MOV C,TIM00
              SJMP GTIMOUT
    GTIMSKIP:       CJNE A,#1H,GTIMSKIP1
              MOV C,TIM01
              SJMP GTIMOUT
    GTIMSKIP1:      CJNE A,#2H,GTIMSKIP2
              MOV C,TIM02
              SJMP GTIMOUT
    GTIMSKIP2:      CJNE A,#3H,GTIMSKIP3
              MOV C,TIM03
              SJMP GTIMOUT
    GTIMSKIP3:      CJNE A,#4H,GTIMSKIP4
              MOV C,TIM04
              SJMP GTIMOUT
    GTIMSKIP4:      CJNE A,#5H,GTIMSKIP5
              MOV C,TIM05
              SJMP GTIMOUT
    GTIMSKIP5:      CJNE A,#6H,GTIMSKIP6
              MOV C,TIM06
              SJMP GTIMOUT
    GTIMSKIP6:      CJNE A,#7H,GT_ERR
              MOV C,TIM07
              SJMP GTIMOUT
    GT_ERR:
              ;ERROR TIMER NUMBER
    GTIMOUT:
              RET
```

```
;-------------------
AND_NOT_CNT_:
          INC DPTR
          MOVX A,@DPTR
          MOV CNT_NUM,A
          LCALL GETCNT_BIT_RL
          MOV TEMP,C
          MOV C,R_REG
          ANL C,/TEMP
          MOV R_REG,C

          RET
;--------------------
AND_TIM_:
          INC DPTR
          MOVX A,@DPTR        ;GET TIMER NUMBER
          MOV  TIM_NUM,A
          LCALL GETTIM_BIT_RL
          MOV TEMP,C
          MOV C,R_REG
          ANL C,TEMP
          MOV R_REG,C
          RET
;-------------------
AND_CNT_:
          INC  DPTR
          MOVX A,@DPTR
          MOV CNT_NUM,A
          LCALL GETCNT_BIT_RL
          MOV TEMP,C
          MOV C,R_REG
          ANL C,TEMP
          MOV R_REG,C
          RET
;-------------------
;GET   COUNTER BIT RELAY
;       INPUT CNT_NUM
;       OUTPUT CARRY FLAG BIT
;       USED BY COUNTER INSTRUCTION
```

```
;
GETCNT_BIT_RL:
        MOV A,CNT_NUM
        CJNE A,#8H,GETCSKIP
        MOV C,CNT08
        SJMP GETCOUT
GETCSKIP:       CJNE A,#9H,GETCSKIP1
        MOV C,CNT09
        SJMP GETCOUT
GETCSKIP1:      CJNE A,#0AH,GETCSKIP2
        MOV C,CNT10
        SJMP GETCOUT
GETCSKIP2:      CJNE A,#0BH,GETCSKIP3
        MOV C,CNT11
        SJMP GETCOUT
GETCSKIP3:      CJNE A,#0CH,GETCSKIP4
        MOV C,CNT12
        SJMP GETCOUT
GETCSKIP4:      CJNE A,#0DH,GETCSKIP5
        MOV C,CNT13
        SJMP GETCOUT
GETCSKIP5:      CJNE A,#0EH,GETCSKIP6
        MOV C,CNT14
        SJMP GETCOUT
GETCSKIP6:      CJNE A,#0FH,GC_ERR
        MOV C,CNT15
        SJMP GETCOUT
GC_ERR:
        ;ERROR NUMBER
GETCOUT:
        RET
;----------------
AND_LD_:
      MOV C,S_REG
      MOV TEMP,C
      MOV C,R_REG
      ANL C,TEMP
      MOV R_REG,C
      RET
```

```
;-----------------
RUN_OR:
        INC DPTR        ;GET NEXT DATA ADDRESS
        MOVX A,@DPTR
        CJNE A,#17H,RUNOR_SKIP  ;"OR NOT ? "
        INC DPTR
        MOVX A,@DPTR
        CJNE A,#12H,RUNOR_SKIP1 ;"OR NOT TIMER ? "
        LCALL OR_NOT_TIM_
        SJMP OR_RET
RUNOR_SKIP1: CJNE A,#11H,RUNOR_SKIP2  ;"OR NOT CNT ? "
        LCALL OR_NOT_CNT_
        SJMP OR_RET
RUNOR_SKIP2: LCALL DEC_DPTR
        LCALL OR_NOT_
        SJMP OR_RET
RUNOR_SKIP:  CJNE A,#12H,RUNOR_SKIP3    ;"OR TIM"
        LCALL OR_TIM_
        SJMP OR_RET
RUNOR_SKIP3: CJNE A,#11H,RUNOR_SKIP4   ;"OR CNT"
        LCALL OR_CNT_
        SJMP OR_RET
RUNOR_SKIP4: CJNE A,#14H,RUNOR_SKIP5
        LCALL OR_LD_
        SJMP OR_RET
RUNOR_SKIP5: LCALL DEC_DPTR
        LCALL OR_
OR_RET:
        RET
;---------------------------
;SUB ROUTINE OF RUN_OR ROUTINE
;
OR_:
          INC DPTR
          MOVX A,@DPTR
          MOV CR_NUM,A
          INC DPTR
          MOVX A,@DPTR
          MOV RELAY_NUM,A
```

```
        LCALL GET_RL_BIT
        MOV TEMP,C
        LCALL SAV_IN_IP
        MOV C,R_REG
        ORL C,TEMP
        MOV R_REG,C
        RET
OR_NOT_:
        INC DPTR
        MOVX A,@DPTR
        MOV CH_NUM,A
        INC DPTR
        MOVX A,@DPTR
        MOV RELAY_NUM,A
        LCALL GET_RL_BIT
        MOV TEMP,C
        LCALL SAV_IN_IP
        MOV C,R_REG
        ORL C,/TEMP      ;ORL C,/BIT
        MOV R_REG,C
        RET
OR_TIM_:
        INC DPTR
        MOVX A,@DPTR
        MOV TIM_NUM,A    ;GET TIMER NUMBER
        LCALL GETTIM_BIT_RL
        MOV TEMP,C
        MOV C,R_REG
        ORL C,TEMP
        MOV R_REG,C

        RET
OR_CNT_:
        INC DPTR
        MOVX A,@DPTR
        MOV CNT_NUM,A
        LCALL GETCNT_BIT_RL
        MOV TEMP,C
        MOV C,R_REG
```

```
        ORL C,TEMP
        MOV R_REG,C

        RET
OR_NOT_TIM_:
        INC DPTR
        MOVX A,@DPTR
        MOV TIM_NUM,A
        LCALL GETTIM_BIT_RL
        MOV TEMP,C
        MOV C,R_REG
        ORL C,/TEMP
        MOV R_REG,C
        RET
OR_NOT_CNT_:
        INC DPTR
        MOVX A,@DPTR
        MOV CNT_NUM,A
        LCALL GETCNT_BIT_RL
        MOV TEMP,C
        MOV C,R_REG
        ORL C,/TEMP
        MOV R_REG,C
        RET
OR_LD_:
        MOV C,S_REG
        MOV TEMP,C
        MOV C,R_REG
        ORL C,TEMP
        MOV R_REG,C

        RET

;-------------------------------------------------------------
RUN_OUT:
        INC DPTR
        MOVX A,@DPTR
        CJNE A,#17H,RUNOUT_SKIP
        LCALL OUT_NOT_
```

```
        SJMP OUT_RET
RUNOUT_SKIP:    LCALL DEC_DPTR
        LCALL OUT_
OUT_RET:
        RET
;----------------------
OUT_NOT_:
        INC DPTR
        MOVX A,@DPTR
        MOV CH_NUM,A    ;GET CHANNEL
        INC DPTR
        MOVX A,@DPTR
        MOV RELAY_NUM,A ;GET BIT PORT
        MOV C,R_REG
        CPL C           ;"OUT NOT"
        MOV TEMP,C
        LCALL OUTP_BIT_INRL
        LCALL OUTP_BIT_PORT
        LCALL OUT_INTER_BIT

        RET
;----------------------
OUT_:
        INC DPTR
        MOVX A,@DPTR
        MOV CH_NUM,A            ;GET CHANNEL
        INC DPTR
        MOVX A,@DPTR
        MOV RELAY_NUM,A         ;GET RELAY NUMBER
        MOV C,R_REG
        MOV TEMP,C
        LCALL OUTP_BIT_INRL
        LCALL OUTP_BIT_PORT
        LCALL OUT_INTER_BIT
        RET
;-----------------------------------------------------
;GET R_REG TO STORE IN O/P INTERNAL REGISTER
;               INPUT CH_NUM,RELAY_NUM,TEMP(FROM R_REG)
;
```

```
OUT_INTER_BIT: -
        MOV A,CH_NUM
        CJNE A,#4H,BSKIP
        MOV A,RELAY_NUM
        CJNE A,#0H,BSKIP1
        MOV C,TEMP
        MOV RL0400,C
        AJMP BSKIP
BSKIP1:     CJNE A,#1H,BSKIP2
        MOV C,TEMP
        MOV RL0401,C
        SJMP BSKIP
BSKIP2:     CJNE A,#2H,BSKIP3
        MOV C,TEMP
        MOV RL0402,C
        SJMP BSKIP
BSKIP3:     CJNE A,#3H,BSKIP4
        MOV C,TEMP
        MOV RL0403,C
        SJMP BSKIP
BSKIP4:     CJNE A,#4H,BSKIP5
        MOV C,TEMP
        MOV RL0404,C
        SJMP BSKIP
BSKIP5:     CJNE A,#5H,BSKIP6
        MOV C,TEMP
        MOV RL0405,C
        SJMP BSKIP
BSKIP6:     CJNE A,#6H,BSKIP7
        MOV C,TEMP
        MOV RL0406,C
        SJMP BSKIP
BSKIP7:     CJNE A,#7H,BSKIP
        MOV C,TEMP
        MOV RL0407,C
BSKIP:

        RET
;-------------------------------------------------
```

```
;GET R_REG VALUE TO  OUT TO RELAY INTERNAL AUXILARY COIL
;IF OUT TO INTERNAL COIL THEN NO OUTPUT
;      INPUT CH_NUM,RELAY_NUM,TEMP<-R_REG

OUTP_BIT_INRL:
          MOV A,CH_NUM
          CJNE A,#9H,OP_OUT
          MOV A,RELAY_NUM
          CJNE A,#0H,OPOUT1
          MOV C,TEMP
          MOV ARL0900,C
OP_OUT:         AJMP OPOUT
OPOUT1:         CJNE A,#1H,OPOUT2
          MOV C,TEMP
          MOV ARL0901,C
          AJMP OPOUT
OPOUT2:         CJNE A,#2H,OPOUT3
          MOV C,TEMP
          MOV ARL0902,C
          AJMP OPOUT
OPOUT3:         CJNE A,#3H,OPOUT4
          MOV C,TEMP
          MOV ARL0903,C
          AJMP OPOUT
OPOUT4:         CJNE A,#4H,OPOUT5
          MOV C,TEMP
          MOV ARL0904,C
          AJMP OPOUT
OPOUT5:         CJNE A,#5H,OPOUT6
          MOV C,TEMP
          MOV ARL0905,C
          AJMP OPOUT
OPOUT6:         CJNE A,#6H,OPOUT7
          MOV C,TEMP
          MOV ARL0906,C
          AJMP OPOUT
OPOUT7:         CJNE A,#7H,OPOUT8
          MOV C,TEMP
          MOV ARL0907,C
```

```
        AJMP OPOUT
OPOUT8:     CJNE A,#8H,OPOUT9
        MOV C,TEMP
        MOV ARL0908,C
        AJMP OPOUT
OPOUT9:     CJNE A,#9H,OPOUT10
        MOV C,TEMP
        MOV ARL0909,C
        AJMP OPOUT
OPOUT10:    CJNE A,#0AH,OPOUT11
        MOV C,TEMP
        MOV ARL0910,C
        AJMP OPOUT
OPOUT11:    CJNE A,#0BH,OPOUT12
        MOV C,TEMP
        MOV ARL0911,C
        AJMP OPOUT
OPOUT12:    CJNE A,#0CH,OPOUT13
        MOV C,TEMP
        MOV ARL0912,C
        AJMP OPOUT
OPOUT13:    CJNE A,#0DH,OPOUT14
        MOV C,TEMP
        MOV ARL0913,C
        AJMP OPOUT
OPOUT14:    CJNE A,#0EH,OPOUT15
        MOV C,TEMP
        MOV ARL0914,C
        AJMP OPOUT
OPOUT15:    CJNE A,#0FH,OPOUT
        MOV C,TEMP
        MOV ARL0915,C

OPOUT:
        RET

;-------------------------------------------------------
;GET R_REG TO OUTPUT BIT PORT
;   INPUT    CH_NUM,RELAY_NUM,TEMP<-R_REG
```

```
;
OUTP_BIT_PORT:
        PUSH DPH
        PUSH DPL
        PUSH 00
        MOV A,CH_NUM
        CJNE A,#4H,O_B_P
        MOV R0,RELAY_NUM
        MOV A,#000000001B  ;CLEAR OUTPUT BIT SPECIFIED
        CJNE R0,#0H,RO_TAT
        SJMP OBP1
RO_TAT:     RL A
        DJNZ R0,RO_TAT
OBP1:       CPL A
        ANL OUTP_REG,A
        MOV R0,RELAY_NUM
        MOV C,TEMP      ;MOV R_REG->TEMP->CARRY->BIT PORT
        CLR A
        RLC A
        CJNE R0,#0H,R_TAT
        SJMP OBP2
R_TAT:      RL A
        DJNZ R0,R_TAT
OBP2:       ORL OUTP_REG,A
        MOV A,OUTP_REG
        MOV DPTR,#8808H
        MOVX @DPTR,A      ;SEND BIT SPECIFIED TO BIT OUTPUT PORT
O_B_P:
        POP 00
        POP DPL
        POP DPH
        RET
;-----------------
RUN_TIM:
        INC DPTR
        MOVX A,@DPTR
        MOV TIM_NUM,A     ;GET TIMER NUMBER
        CJNE A,#7H,TJMP1  ;TIMER 0-7
```

```
TJMP1:      JNC TJMP

            INC DPTR
            MOVX A,@DPTR
            MOV TC_REG_H,A      ;GET PRESET VAL  HIGH
            INC DPTR
            MOVX A,@DPTR        ;GET PRESET VAL LOW
            MOV TC_REG_L,A
            MOV C,R_REG
            JC T_RESET          ;R=RESET
            MOV C,S_REG
            JNC T_RESET         ;RESET
            MOV C,TFLAG         ;START
            JNC T_LOAD
            LCALL TCOUNT
            LCALL S_T_REG
            SJMP T_RET
T_LOAD:
            ;LOAD TIME PRESET VALUE
            LCALL TLOAD
            SETB TFLAG
            SJMP T_RET
T_RESET:
            ;RESET TIMER
            LCALL TRESET
            CLR TFLAG
            SJMP T_RET
TJMP:
            ;TIMER NUMBER ERROR
T_RET:
            RET
;------------
;SUBROUTINE OF RUN_TIM
;INPUT TC_REG_H,TC_REG_L,TIM_NUM
;
TCOUNT:
            PUSH 00
            PUSH 01
            PUSH 02
```

```
        LCALL TLDPSETVAL
        MOV A,@R0           ;GET HIGT BYTE OF PRESET COUNT
        CJNE A,TC_REG_H,TCSKIP
TCSKIP:     JNC TCSKIPOUT
        INC R0
        MOV A,@R0           ;GET LOW BYTE OF PRESET COUNT
        CJNE A,TC_REG_L,TCSKIP1
TCSKIP1:    JC TCSKIPOUT
                ;PRESET VAL=PRESET COUNT
        LCALL TRESET        ;RESET
        LCALL S_T_REG       ;SET TIM BIT RELAY
TCSKIPOUT:
        POP  02
        POP  01
        POP  00
        RET
;-----------------------------
;TIMER LOAD PRESET VALUE
;INPUT TC_REG_H,TC_REG_L,TIM_NUM
;OUTPUT TO T?PSET_H,T?PSET_L(?=0-7)
;
TLOAD:
        PUSH 00
        LCALL TLDPSETVAL    ;GET TIMER PRESET VALUE ADDRESS
        MOV A,TC_REG_H
        MOV @R0,A
        INC R0
        MOV A,TC_REG_L
        MOV @R0,A
        POP 00
        RET
TLDPSETVAL:                         ;INPUT >TIM_NUM
        MOV A,TIM_NUM
        CJNE A,#0H,TLD1
        MOV R0,#66H
        SJMP TLD_RET
TLD1:       CJNE A,#1H,TLD2
        MOV R0,#68H
        SJMP TLD_RET
```

```
TLD2:   .       CJNE A,#2H,TLD3
        MOV R0,#6AH
        SJMP TLD_RET
TLD3:           CJNE A,#3H,TLD4
        MOV R0,#6CH
        SJMP TLD_RET
TLD4:           CJNE A,#4,TLD5
        MOV R0,#6EH
        SJMP TLD_RET
TLD5:           CJNE A,#5H,TLD6
        MOV R0,#70H
        SJMP TLD_RET
TLD6:           CJNE A,#6H,TLD7
        MOV R0,#72H
        SJMP TLD_RET
TLD7:           CJNE A,#7H,TLD8
        MOV R0,#74H
        SJMP TLD_RET
TLD8:
        ;TIM NUM ERROR
TLD_RET:
        RET
;-----------------------------
S_T_REG:        ;SET TIM BIT IF TIMER =PRESET VAL INPUT>TIM_NUM
        MOV A,TIM_NUM
        CJNE A,#0H,STREG1
        SETB TIM00
        SJMP STRET
STREG1:         CJNE A,#1H,STREG2
        SETB TIM01
        SJMP STRET
STREG2:         CJNE A,#2H,STREG3
        SETB TIM02
        SJMP STRET
STREG3:         CJNE A,#3H,STREG4
        SETB TIM03
        SJMP STRET
STREG4:         CJNE A,#4H,STREG5
        SETB TIM04
```

```
        SJMP STRET
STREG5:     CJNE A,#5H,STREG6
        SETB TIM05
        SJMP STRET
STREG6:     CJNE A,#6H,STREG7
        SETB TIM06
        SJMP STRET
STREG7:     SETB TIM07
STRET:
        RET
;-----------------------------
;TIMER RESET PRESET VALUE
;INPUT TIM_NUM,TC_REG_H,TC_REG_L
;
TRESET:
        PUSH 00
        LCALL TLDPSETVAL
        MOV @R0,#0H
        INC R0
        MOV @R0,#0H
        POP 00
        RET
;------------
RUN_CNT:
         INC DPTR
         MOVX A,@DPTR
         MOV CNT_NUM,A

         RET
;-------------------------------------------------------------
;***************  MONITOR MODE ***************************************
;-------------------------------------------------------------
MON_MODE:
      ;CHECK I/O
      ;GET DATA FROM I/O
      ;FORCE O/P RELAY


      RET
```

```
;-------------------------------------------------------------------
;START SEND/RECIEVE  .. SERIAL  ROUTINE
;------------------------------
 SERIAL:
       MOV TH1,#0E8H   ;BUADRATE =1200   START SERIAL COMUNICATION
       MOV R0,#0H
       CLR SBUF
       MOV A,#26H              ;" & " FOR START
       MOV SBUF,A
       LCALL D_LAY_2
       CLR SBUF
LOOP:
       MOV DPTR,#TRANSMIT1
       MOV A,R0
       MOVC A,@A+DPTR
       CJNE A,#24H,CONT        ;" $ " FOR END
       SJMP SKIP_OUT
  CONT:

       CLR SBUF
       LCALL D_LAY_2
       MOV SBUF,A
       INC R0                  ;STRING LENTH 256 CHARACTERS TO  SEND
       SJMP LOOP
 SKIP_OUT:
       MOV R0,#0H
       LCALL D_LAY_2
       MOV SBUF,A
       CLR ES
       CLR TI
       CLR SBUF

       RET
;
;WATCHDOG PROGRAM
;
WATCHDOG:
       ;
       RET
```

```
;-------------------------------------------------------
;
;           INTERRUPT SERVICE ROUTINE
;-------------------------------------------------------

EX_INT0:
      ;FOR EXTERNAL INTERRUPT0
       RETI
;---------------------------------
;TIMER 0 INTERRUPT SERVICE ROUTINE
;
ISR_TIM0:
      PUSH PSW
      PUSH ACC
      CLR A
      CLR  EA              ;DISABLE ALL INTERRUPT FLAG
                ;FOR TIMER0/COUNTER0 INTERRUPT
      MOV TH0,#0FFH        ;LOAD TIMER VALUE 65536 STEP
      MOV TL0,#0FFH
      CLR  TF0             ;CLEAR TIMER INTERRUPT FLAG



      INC 60H
      MOV A,60H
      CJNE A,#0EH,T0_SKIP
      SJMP T0_OUT
T0_SKIP: JC T0_OUT
      MOV 60H,#0H          ;CLEAR TIMER STEP TO INITIAL VALUE
      CLR A
      MOV A,61H            ;DATA FOR OUT PUT PORT
      RL A
      MOV 61H,A
      MOV DPTR,#8080H
      MOVX @DPTR,A
      ;MOV C,BIT0
      ;MOV P1.0,C
      ;CPL C
      ;MOV BIT0,C
T0_OUT:
```

```
        SETB EA              ;ENABLE ALL INTERRUPT
       POP ACC
       POP PSW
        RETI
;-----------------------------------
;EXTERNAL INTERRUPT 1 SERVICE ROUTINE
;
EX_INT1:

       ;FOR EXTERNAL INTERRUPT


      RETI
;-----------------------------------
;SERIAL INTERRUPT SERVICE ROUTINE
;
ISR_232:
        PUSH  ACC
        PUSH  PSW
        CLR EA
        CLR ES
        CLR TI
        SETB ES
        SETB EA
        POP PSW
        POP ACC

        RETI
;-------------------------------------------------
D_LAY_2:                          ;DELAY TIME ---> 3s
         PUSH  00
         PUSH  01
         PUSH  02
         MOV   R0,#25H
 D_LAY1:    MOV   R1,#50H
 D_LAY2:    MOV   R2,#64H
 D_LAY3:    DJNZ  R2,D_LAY3
        DJNZ  R1,D_LAY2
        DJNZ  R0,D_LAY1
```

```
        POP   02
        POP   01
        POP   00
        RET

DELAY:                          ;DELAY TIME ---> 30ms
        PUSH  00
        PUSH  01
        PUSH  02
        MOV   R0,#02H
  DLAY1:    MOV   R1,#50H
  DLAY2:    MOV   R2,#64H
  DLAY3:    DJNZ  R2,DLAY3
        DJNZ  R1,DLAY2
        DJNZ  R0,DLAY1
        POP   02
        POP   01
        POP   00
        RET
;-------------------------------------------------------
TRANSMIT1:    DFB   20H,4EH,41H,52H,4FH,4EH,47H,2EH,2EH,24H

        END START
```

ภาคผนวก จ.
ข้อมูลของหน่วยต่างๆ

ตารางคำสั่ง HD44780

Instruction	Code RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Description	Execution time (when fosc is 250 kHz) Note 1	Execution tir (when fosc i 160 kHz) Note 2
Clear display	0	0	0	0	0	0	0	0	0	1	Clears all display and returns the cursor to the home position (Address 0).	82 μs ~ 1.64 ms	120 μs ~ 4.9
Return home	0	0	0	0	0	0	0	0	1	*	Returns the cursor to the home position (Address 0). Also returns the display being shifted to the original position. DD RAM contents remain unchanged.	40 μs ~ 1.6 ms	120 μs ~ 4.8
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets the cursor move direction and specifies or not to shift the display. These operations are performed during data write and read.	40 μs	120 μs
Display ON/OFF control	0	0	0	0	0	0	1	D	C	B	Sets ON/OFF of all display (D), cursor ON/OFF (C), and blink of cursor position character (B).	40 μs	120 μs
Cursor and display shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves the cursor and shifts the display without changing DD RAM contents	40 μs	120 μs
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL) number of display lines (L) and character font (F).	40 μs	120 μs
Set CG RAM address	0	0	0	1	ACG						Sets the CG RAM address. CG RAM data is sent and received after this setting.	40 μs	120 μs
Set DD RAM address	0	0	1	ADD							Sets the DD RAM address. DD RAM data is sent and received after this setting.	40 μs	120 μs
Read busy flag & address	0	1	BF	AC							Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.	1μs	1 μs
Write data to CG or DD RAM	1	0	Write Data								Writes data into DD RAM or CG RAM.	40 μs	120 μs
Read data to CG or DD RAM	1	1	Read Data								Reads data from DD RAM or CG RAM.	40 μs	120 μs

I/D = 1 Increment (+1) I/D = 0 Decrement (−1)
S = 1 Accompanies display shift.
S/C = 1: Display shift S/C = 0: Cursor move
R/L = 1: Shift to the right.
R/L = 0. Shift to the left.
DL = 1 8 bits DL = 0. 4 bits
N = 1 2 lines N = 0 1 line
F = 1 5 x 10 dots F = 0 5 x 7 dots
BF = 1: Internally operating
BF = 0. Can accept instruction

DD RAM: Display date RAM
CG RAM. Character generator RAM
ACG: CG RAM address
ADD: DD RAM address Corresponds to cursor address.
AC: Address counter used for both of DD and CG RAM address.

Execution time changes when frequency changes.
(Example)
When fosc is 270 kHz:

$$40\ \mu s \times \frac{250}{270} = 37\ \mu s$$

*No effect
Notes 1. Applied to models driven by 1/8 duty or 1/11 duty.
2. Applied to models driven by 1/16 duty.

Table 10. 8051 Instruction Set Summary

Interrupt Response Time: Refer to Hardware Description Chapter.

Instructions that Affect Flag Settings(1)

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

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

Note on instruction set and addressing modes:

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

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

Table 10. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
ARITHMETIC OPERATIONS (Continued)				
INC	DPTR	Increment Data Pointer	1	24
MUL	AB	Multiply A & B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12
LOGICAL OPERATIONS				
ANL	A,Rn	AND Register to Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND indirect RAM to Accumulator	1	12
ANL	A,# data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,# data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12
ORL	A,# data	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,# data	OR immediate data to direct byte	3	24
XRL	A,Rn	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,# data	Exclusive-OR immediate data to Accumulator	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12
XRL	direct,# data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement	1	12

Mnemonic		Description	Byte	Oscillator Period
LOGICAL OPERATIONS (Continued)				
RL	A	Rotate Accumulator Left	1	12
RLC	A	Rotate Accumulator Left through the Carry	1	12
RR	A	Rotate Accumulator Right	1	12
RRC	A	Rotate Accumulator Right through the Carry	1	12
SWAP	A	Swap nibbles within the Accumulator	1	12
DATA TRANSFER				
MOV	A,Rn	Move register to Accumulator	1	12
MOV	A,direct	Move direct byte to Accumulator	2	12
MOV	A,@Ri	Move indirect RAM to Accumulator	1	12
MOV	A,# data	Move immediate data to Accumulator	2	12
MOV	Rn,A	Move Accumulator to register	1	12
MOV	Rn,direct	Move direct byte to register	2	24
MOV	Rn,# data	Move immediate data to register	2	12
MOV	direct,A	Move Accumulator to direct byte	2	12
MOV	direct,Rn	Move register to direct byte	2	24
MOV	direct,direct	Move direct byte to direct	3	24
MOV	direct,@Ri	Move indirect RAM to direct byte	2	24
MOV	direct,# data	Move immediate data to direct byte	3	24
MOV	@Ri,A	Move Accumulator to indirect RAM	1	12

Table 10. 8051 Instruction Set Summary (Continued)

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

Mnemonic		Description	Byte	Oscillator Period
BOOLEAN VARIABLE MANIPULATION				
CLR	C	Clear Carry	1	12
CLR	bit	Clear direct bit	2	12
SETB	C	Set Carry	1	12
SETB	bit	Set direct bit	2	12
CPL	C	Complement Carry	1	12
CPL	bit	Complement direct bit	2	12
ANL	C,bit	AND direct bit to CARRY	2	24
ANL	C,/bit	AND complement of direct bit to Carry	2	24
ORL	C,bit	OR direct bit to Carry	2	24
ORL	C,/bit	OR complement of direct bit to Carry	2	24
MOV	C,bit	Move direct bit to Carry	2	12
MOV	bit,C	Move Carry to direct bit	2	24
JC	rel	Jump if Carry is set	2	24
JNC	rel	Jump if Carry not set	2	24
JB	bit,rel	Jump if direct Bit is set	3	24
JNB	bit,rel	Jump if direct Bit is Not set	3	24
JBC	bit,rel	Jump if direct Bit is set & clear bit	3	24
PROGRAM BRANCHING				
ACALL	addr11	Absolute Subroutine Call	2	24
LCALL	addr16	Long Subroutine Call	3	24
RET		Return from Subroutine	1	24
RETI		Return from interrupt	1	24
AJMP	addr11	Absolute Jump	2	24
LJMP	addr16	Long Jump	3	24
SJMP	rel	Short Jump (relative addr)	2	24

Table 13. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
PROGRAM BRANCHING (Continued)				
JMP	@A+DPTR	Jump indirect relative to the DPTR	1	24
JZ	rel	Jump if Accumulator is Zero	2	24
JNZ	rel	Jump if Accumulator is Not Zero	2	24
CJNE	A,direct,rel	Compare direct byte to Acc and Jump if Not Equal	3	24
CJNE	A,#data,rel	Compare immediate to Acc and Jump if Not Equal	3	24
CJNE	Rn,#data,rel	Compare immediate to register and Jump if Not Equal	3	24
CJNE	@Ri,#data,rel	Compare immediate to indirect and Jump if Not Equal	3	24
DJNZ	Rn,rel	Decrement register and Jump if Not Zero	2	24
DJNZ	direct,rel	Decrement direct byte and Jump if Not Zero	3	24
NOP		No Operation	1	12

All mnemonics copyrighted © Intel Corporation 1980

The relation between the operation and the combination of RS, R/W

RS	RW	E	OPERATION
0	0		Write instruction code
0	1		Read busy flag and address counter
1	0		Write data
1	1		Read data

When performing data and instruction code by 4 bit, transfer RS, R/W every time.

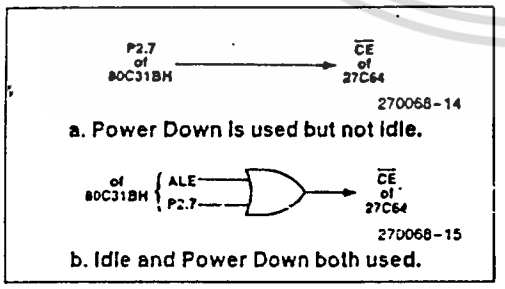


Figure 15. Modifications to 80C31BH/27C64 Interface

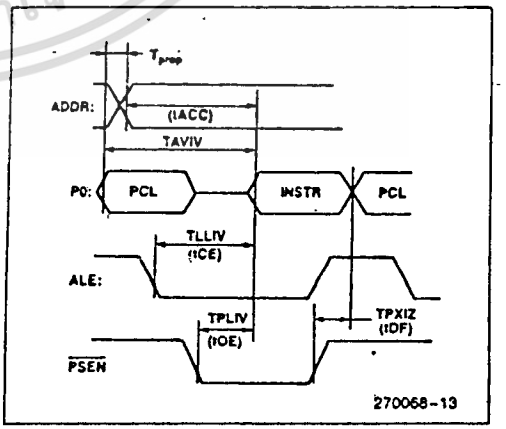
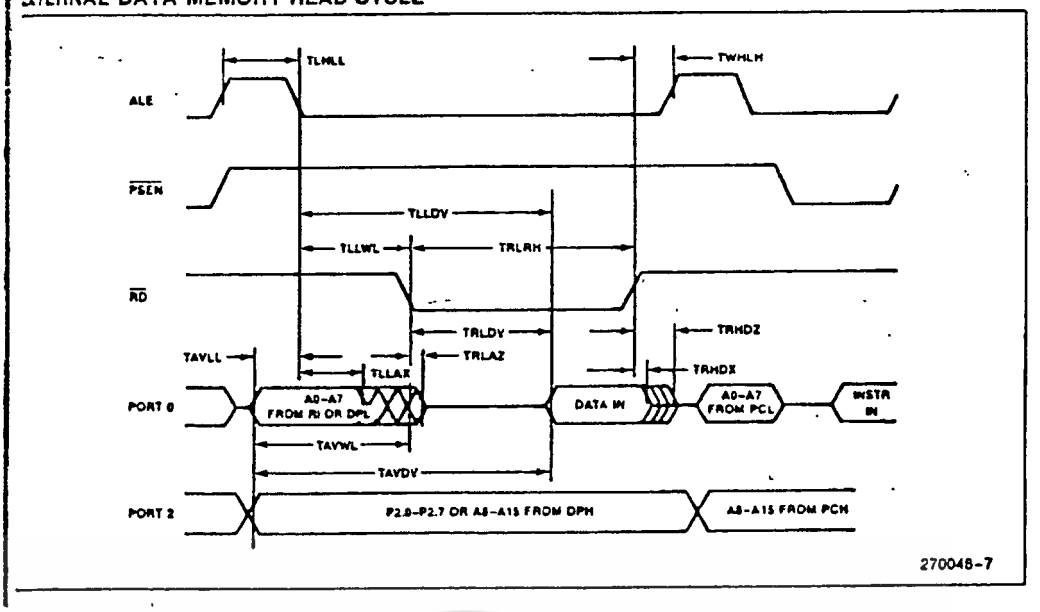
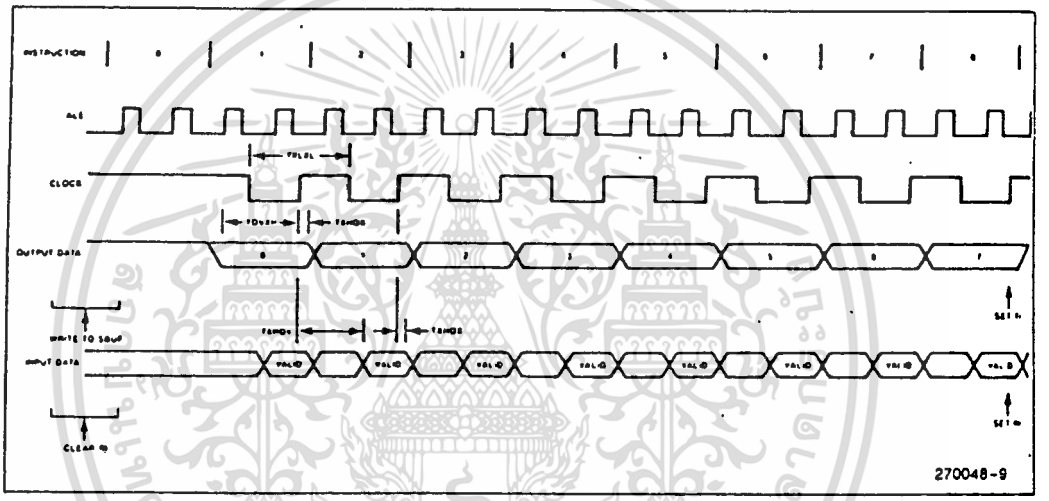


Figure 14b. Timing Waveforms for 80C31BH + 27C64

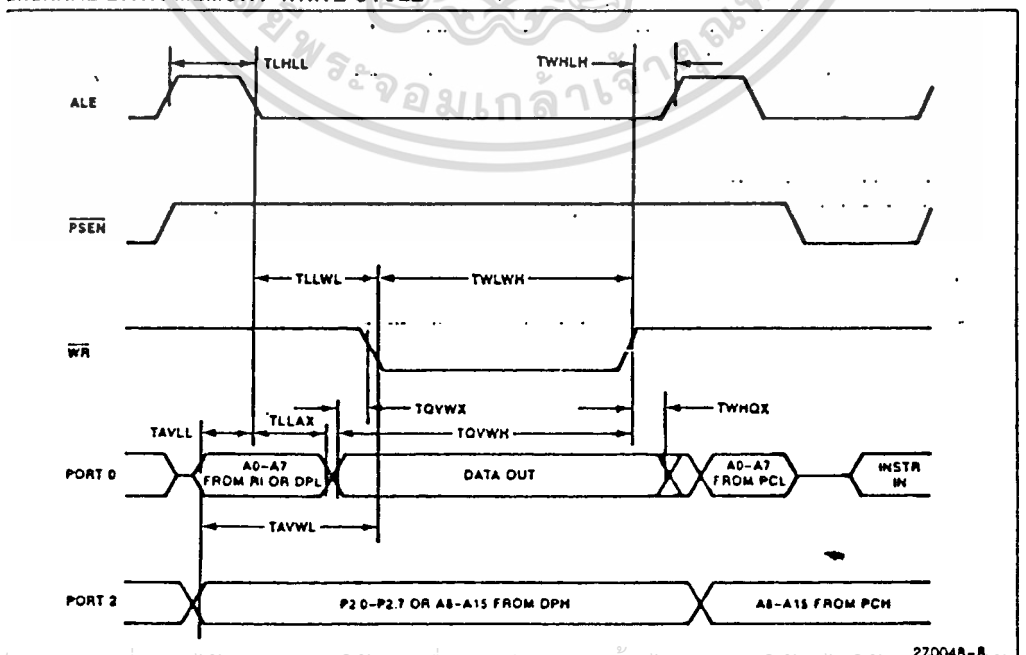
EXTERNAL DATA MEMORY READ CYCLE



SHIFT REGISTER TIMING WAVEFORMS



EXTERNAL DATA MEMORY WRITE CYCLE



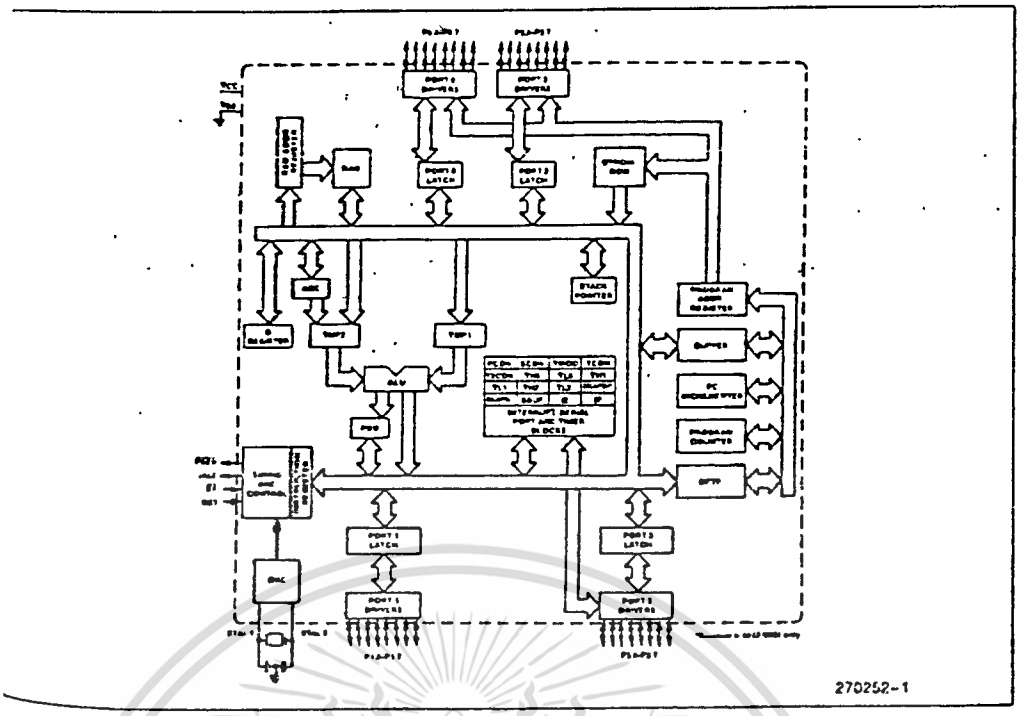


Figure 1. MCS-51 Architectural Block Diagram

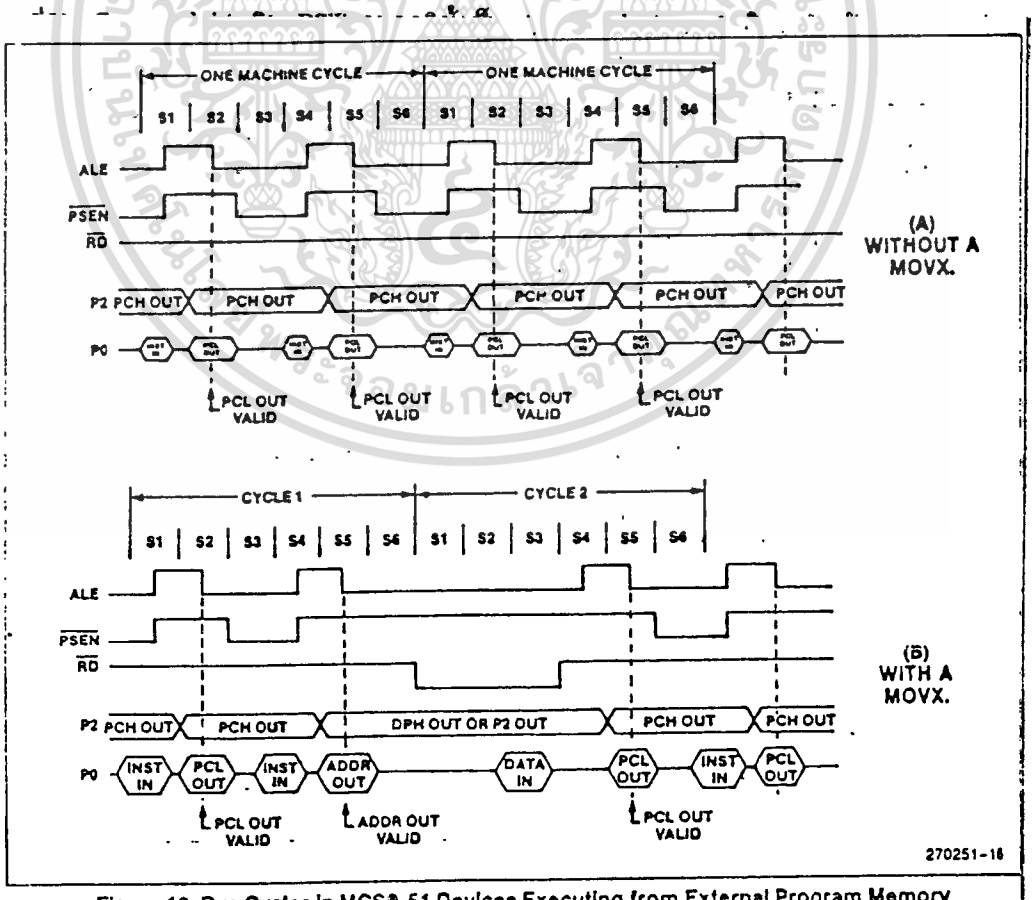


Figure 18. Bus Cycles In MCS®-51 Devices Executing from External Program Memory

The internal RAM is not affected by reset. On power up the RAM content is indeterminate.

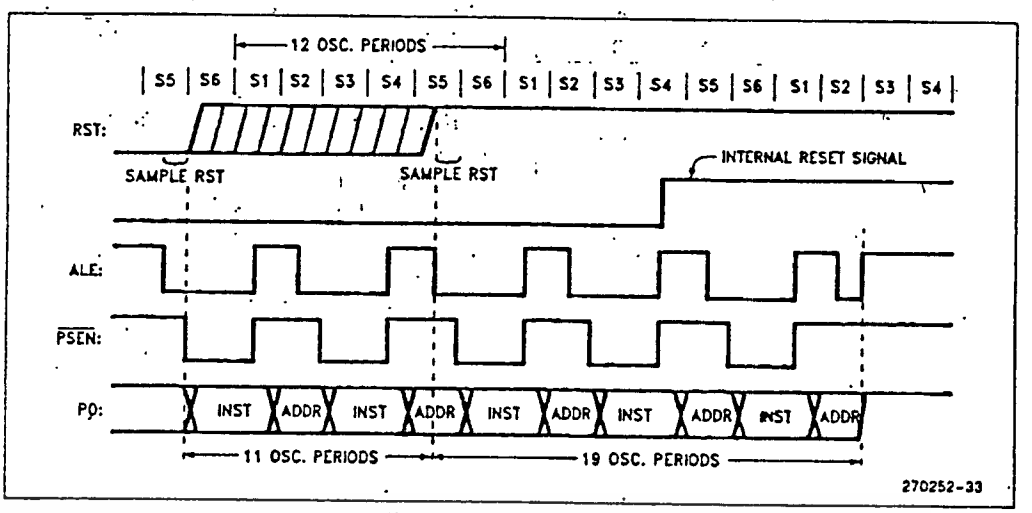


Figure 25. Reset Timing

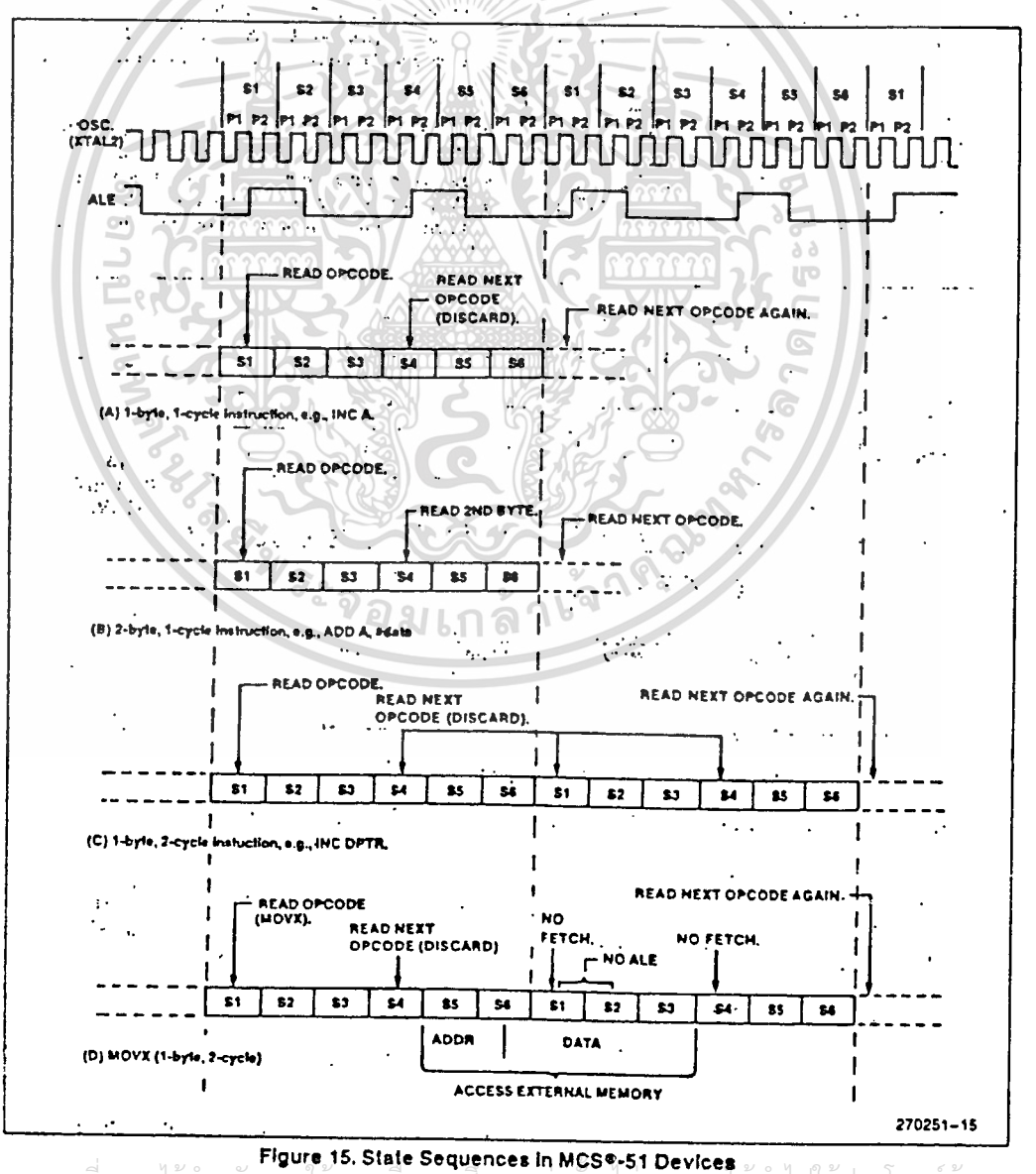
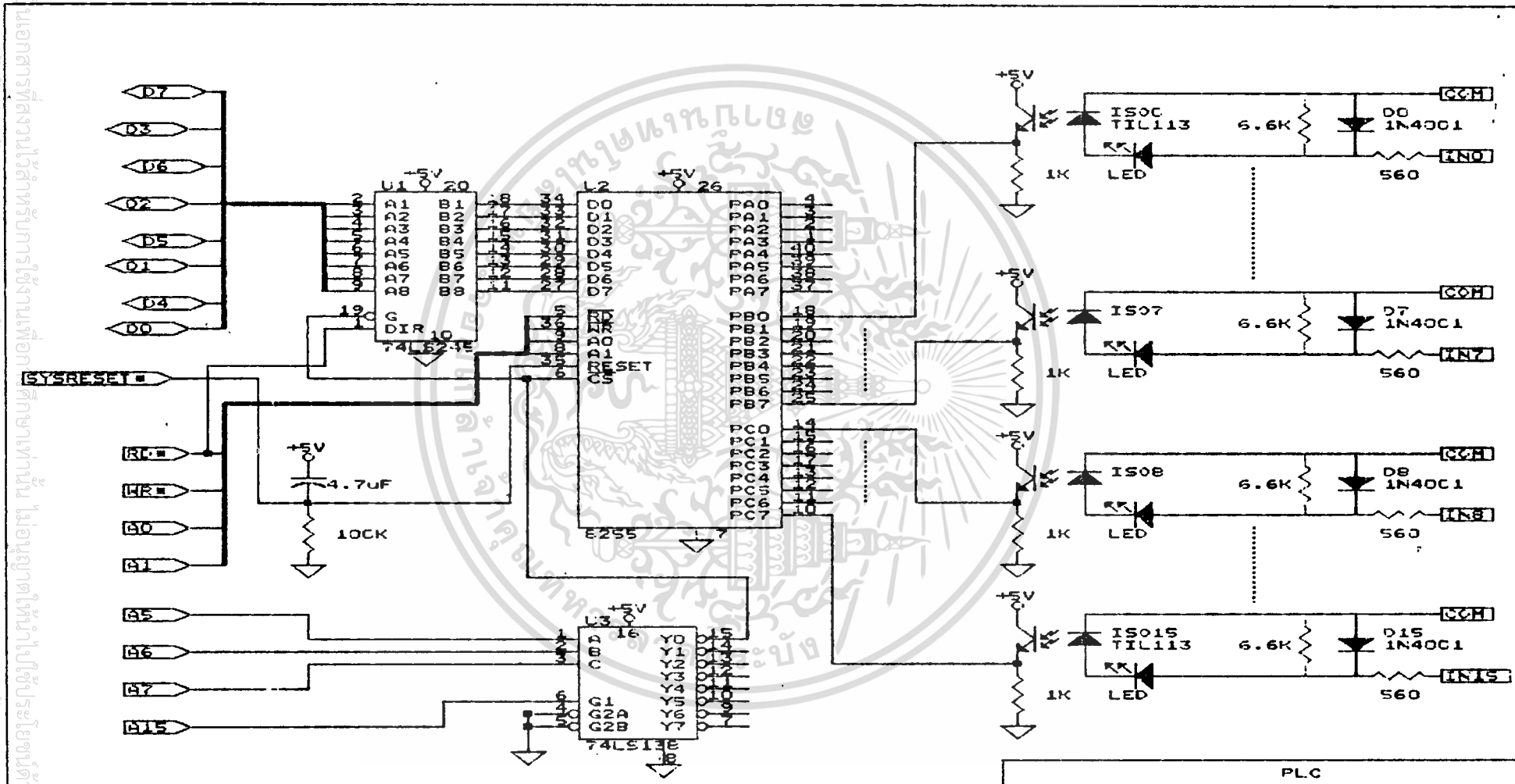
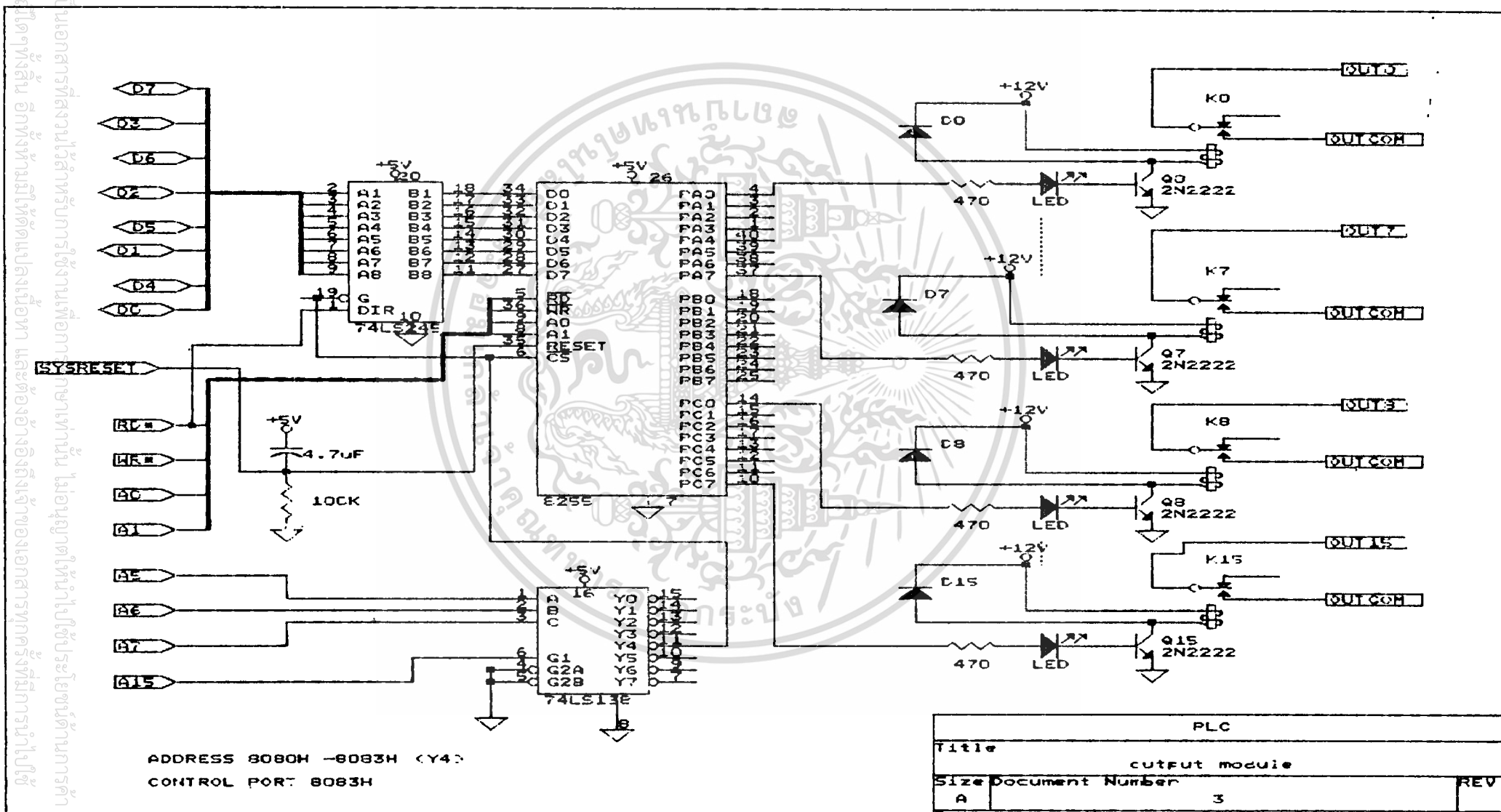


Figure 15. State Sequences in MCS®-51 Devices

ภาคผนวก ฉ.
ลายวงจรของหน่วยต่างๆ




D7
D3
D6
D2
D5
D1
D4
D0
SYSRESET
RD
WR
A0
A1
A5
A6
A7
A15
+5V
74LS245
8255
74LS138
4.7uF
10CK
470
LED
+12V
D0
D7
D8
D15
Q0 2N2222
Q7 2N2222
Q8 2N2222
Q15 2N2222
K0
K7
K8
K15
OUT0
OUT7
OUT8
OUT15
OUTCOM
ADDRESS 8080H -8083H (Y4)
CONTROL PORT 8083H
PLC
Title
cutput module
Size
A
Document Number
3
REV
Date:
May 4, 1991
Sheet
3 of
3

กิตติกรรมประกาศ

(ACKNOWLIDGIMENT)

โครงงาน ฉบับนี้สำเร็จลงได้ด้วยความร่วมมือจากหลายๆฝ่ายด้วยกัน จากอาจารย์ วัชระ ฉัตรวิริยะ ซึ่งเป็นอาจารย์ที่ปรึกษา อาจารย์ท่านอื่น ๆ และผู้มีส่วนเกี่ยวข้องทุก ๆ ท่าน ที่ได้ให้ความช่วยเหลือ จนทำให้โครงงานฉบับนี้สำเร็จลุล่วงลงได้ด้วยดี ดังนั้นจึงขอขอบคุณมา ณ โอกาศนี้ด้วยความร่วมมือจากหลายฝ่ายด้วยกันดังนั้นจึงขอขอบ

เอกสารอ้างอิง

1. สุเธียร เกียรติสุนทร หลักการทำงานและเทคนิคการประยุกต์ใช้งาน PC/PLC บริษัท ซีเอ็ดยูเคชั่น จำกัด
2. Avtar Singh., and Walter A. Triebel, "THE 8088 MICROPROCESSOR programming, interfacing, software, hardware, and applications", Prentice-Hall International Editions.
3. Christopher A. Titus, Jonathan A. titus, and David G. Larsen, "STD BUS Interfacing, Howard W. Sams & Co., Inc.
4. Frank D. Petruzella, PROGRAMMABLE LOGIC CONTROLLERS, McGraw-Hill Book Company.
5. Omron, "C 20 Programmable Controller System C-Series : User Manual" Omron Tateisi Electronics.
6. Peter D. Lawrence, and Konrad Mauch, REAL-TIME MICROCOMPUTER SYSTEM DESIGN An Introduction.", McGraw-Hill Book Company.
7. Peter R. Rony, "Interfacing and Scientific Data Communication Experiments", Howard W. Sams & Co.,Inc.
8. "Embedded Controller Handbook Volume I 8-Bit", Intel
9. เทคนิค เครื่องกล.ไฟฟ้า.อุตสาหการ ฉบับพิเศษ บริษัท ย.ที.อี. จำกัด