ระบบรหัสแท่งแบบเครือข่าย

BARCODE NETWORK SYSTEM

โดย

นายอาวุธ ธรรมาคม รหัสประจำตัว 37.013280

นายพีระพงษ์ เกษรบัว รหัสประจำตัว 37.013257

นายไมตรี สินไหม รหัสประจำตัว 37.013259

อาจารย์ที่ปรึกษา

อาจารย์ ชินภัทร นันทจิวาภรชัย

ปริญญานิพนธ์สำหรับปริญญาตรีวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

ปริญญานิพนธ์ ปีการศึกษา 2539

ภาควิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบรหัสแท่งแบบเครือข่าย

ผู้จัดทำ 1. นายอาวุธ ธรรมาคม 37.013280

2. นายพีระพงษ์ เกษรบัว 37.013257

3. นายไมตรี สินไหม 37.013259

..... อาจารย์ที่ปรึกษา

(อ.ชินภัทร นันทจิวาภรชัย)

ระบบรหัสแท่งแบบเครือข่าย

อาวุธ ธรรมาคม
พีระพงษ์ เกษรบัว
ไมตรี สินไหม
อ.ชินภัทร นันทจิวาภรชัย อาจารย์ที่ปรึกษา
ปีการศึกษา 2539

บทคัดย่อ

ในปริญญานิพนธ์ฉบับนี้เป็นรายงานการศึกษาและออกแบบสร้างโครงงานเครื่องอ่านรหัสแท่ง โดยใช้รหัส 39 (Code 39) เป็นรูปแบบในการอ่านข้อมูลและถอดรหัส ซึ่งจะทำการอ่านข้อมูลรหัสแท่งผ่านหัวอ่านและนำสัญญาณที่อ่านได้นั้นไปประมวลผลด้วยไมโครคอลโทรลเลอร์ #8031 แล้วทำการแปลงสัญญาณที่อ่านเข้ามาให้เป็นรหัสแอสกี้ (ASCII Code) ส่งไปแสดงผลบนจอ LCD

BARCODE NETWORK SYSTEM

Avut Thanmacom

Peerapong Kasornbau

Maitree Sinmai

Chinapatra Nanthajiwapornchai Advisor

1997

ABSTRACT

This thesis presents a delevopment of Barcode Network System. Based on the code 39, the system operate in the network for scanning barcode at any where. The system is controled by a microcontroller MCS-51. It has data-memory and display by a LCD module. It interfaces to Microcomputer with RS-232C serial port and record data into database for processing and searching data by a microcomputer.

สารบัญ

สารบัญตาราง

สารบัญรูป

หน้า

บทที่ 1
บทนำ

ในระบบที่เป็นการปฏิบัติงานอัตโนมัติที่มีจำนวนงานมาก ๆ เครื่องจักรจะถูกนำมาใช้เพื่ออำนวยความสะดวกโดยการเปลี่ยนรูปแบบจากเดิมที่มนุษย์เข้าใจเป็นรูปแบบของรหัสแทน ซึ่งรหัสนี้อาจใช้แทนตัวแปรเดียวหรือหลายๆ ตัวก็ได้ สำหรับงานที่แตกต่างกันไป

รหัสแท่ง (Barcode) คือรหัสที่ใช้แทนสิ่งเหล่านั้นในรูปแบบของเครื่องจักร ที่อ่านแท่งสี่เหลี่ยมทึบและช่องว่างสีขาวในอัตราส่วนที่กำหนด โดยมีตัวตรวจจับ (sensor) เป็นตัวอ่านความหมายจากแท่งทึบและช่องว่างนั้นออกมา เพื่อประมวลผลต่อไปในขั้นตอนของสัญญาณทางไฟฟ้า กรรมวิธีในการทำงานนั้นอาจเปรียบเทียบได้กับกระบวนการทำงานในร่างกายของมนุษย์ที่มีสายตาเป็นตัวตรวจจับ และสมองเป็นตัวประมวลผลหรือสั่งงาน

รหัสแท่งจัดเป็นรูปแบบการใช้งานที่ง่ายที่สุด รวมถึงราคาและความน่าเชื่อถือได้ นับว่าเหมาะสมที่สุดที่จะใช้งานกับระบบข้อมูลของคอมพิวเตอร์ ตัวอย่างการนำรหัสแท่งไปใช้งาน และก็ช่วยงานได้มากคือ ระบบไปรษณีย์อัตโนมัติ โดยการนำไปใช้คัดเลือกชนิดของจดหมายและปลายทางที่จะส่งไป

ชนิดของตัวตรวจจับรหัสแท่งทั่วๆไปแบ่งได้ 2 ประเภท คือ ชนิดมือถือ และแบบที่ตั้งอยู่กับที่ สำหรับแบบมือถือนั้น ผู้ปฏิบัติงานฝึกหัดเพียงเล็กน้อย ก็สามารถที่จะทำงานได้และสามารถที่จะทำงานได้รวดเร็ว และถูกต้องกว่าการใช้คนป้อนข้อมูลมาก อีกทั้งการเปลี่ยนแปลงรูปแบบรหัสแท่งในผลิตภัณฑ์นั้นก็ง่ายมาก มีข้อมูลที่น่าสนใจเปรียบเทียบให้เห็นถึงความผิดพลาดซึ่งเกิดจากการใช้รหัสแท่งจะมีแค่ 1 ใน 10000 ในขณะที่หากใช้คนป้อนข้อมูลความผิดพลาดจะสูงถึง 1 ใน 300 อัตราการผิดพลาดจากการใช้รหัสแท่งสามารถลดลงได้ โดยใช้วิธีการตรวจเช็คตัวเลขและเทคนิคการป้องกันข้อมูลในรูปแบบอื่นร่วมด้วย

ข้อแตกต่างของรูปแบบต่าง ๆ ของรหัสแท่งทุกวันนี้มีอยู่มากมายขึ้นอยู่กับความเหมาะสมในงานแต่ละชนิดไป โดยจะขึ้นอยู่กับตัวถอดรหัสแท่ง (Barcode Decoder) ซึ่งจะต้องตรงกับชนิดของรหัสแท่งที่ใช้ นอกเหนือจากนั้นต้องมีระบบของแหล่งกำเนิดแสงและตัวตรวจจับแสง ส่วนการแสดงผลจะแสดงออกทางตัวแสดงผลแอลอีดี (LED) หรือ จอแสดงผลแบบผลึกเหลว หรือต่อไปยังอินพุตของระบบคอมพิวเตอร์ โดยการต่อผ่านทางพอร์ต RS-232C มาตรฐาน หรือต่อเชื่อมกับแป้นพิมพ์

การใช้แสงเลเซอร์ในการอ่านรหัสแท่งเพิ่งจะมีมาได้เมื่อต้นทศวรรษที่ 70 โดยใช้งานร่วมกับระบบไมโครโปรเซสเซอร์ซึ่งการประมวลผลจากตัวอ่านนี้หากว่ามีความสามารถในการอ่านรูปแบบรหัสของรหัสแท่งที่แตกต่างกันจะมีความยืดหยุ่นในการใช้งานมากกว่า โดยเฉพาะงานที่ต้องใช้รูปแบบในสายงานการผลิต ยกตัวอย่างในอุตสาหกรรมเวชภัณฑ์จะใช้รหัสยูพีซี (UPC) เป็นหลักในขณะที่ใช้รหัส 39 (code 39) สำหรับการใช้งานรูปแบบอื่น ๆ

1.1 วัตถุประสงค์ของโครงงาน

1.1.1 เพื่อศึกษารหัสแท่งแบบต่าง ๆ และการนำมาประยุกต์ใช้งาน

1.1.2 เพื่อศึกษาการออกแบบวงจรเครื่องอ่านรหัสแท่ง

1.1.3 เพื่อศึกษาและประยุกต์ใช้งานไมโครคอนโทรลเลอร์ตระกูล MCS-51

1.1.4 เพื่อศึกษาการประยุกต์ใช้งานไมโครคอมพิวเตอร์ และการสื่อสารข้อมูลแบบอนุกรมผ่านพอร์ตอนุกรม RS-232C และ RS-485

1.1.5 เพื่อศึกษาและพัฒนาโปรแกรมจัดฐานข้อมูลแบบ FoxPro

1.2 ขอบเขตของโครงงาน

1.2.1 หัวอ่านรหัสแท่งแบบลำแสงคงที่ (Fix Beam) จำนวน 2-3 หัวอ่าน ประกอบกันเป็นเครือข่ายสำหรับใช้รูดบัตรที่มีรหัสแท่งตามสถานที่ต่างๆ มากกว่าหนึ่งสถานที่

1.2.2 เครื่องอ่านรหัสแท่งและบันทึกเวลา ใช้ไมโครคอนโทรลเลอร์ตระกูล MCS-51 เป็นหน่วยประมวลผลกลาง มีหน่วยความจำ (RAM) สำหรับเก็บข้อมูลชั่วคราว และแสดงผลได้ด้วยจอแสดงผลแบบผลึกเหลว (LCD, Liquid Crystal Display)

1.2.3 เชื่อมต่อกับไมโครคอมพิวเตอร์โดยผ่านทางพอร์ตอนุกรม (RS-232C) ด้วยอัตราความเร็ว 9600 บอด

1.2.4 บันทึกข้อมูลจากเครื่องอ่านรหัสแท่งเก็บไว้ในฐานข้อมูล บนไมโครคอมพิวเตอร์เพื่อนำไปประมวลผล (ค้นคืนข้อมูล)

1.3 วิธีดำเนินงาน

1.3.1 ศึกษารายละเอียดเกี่ยวกับรหัสมาตราฐานแบบต่างๆที่มีใช้กันในปัจจุบัน

1.3.2 ศึกษาการออกแบบสร้างวงจรหัวอ่านรหัสแท่งแบบลำแสงคงที่ (Fix Beam)

1.3.3 ศึกษาการสื่อสารข้อมูลแบบอนุกรม RS-232C และ RS-485

1.3.4 ศึกษาการออกแบบและสร้าง เครื่องอ่านรหัสแท่งบันทึกเวลา

1.3.5 ศึกษาและพัฒนาโปรแกรม เพื่อควบคุมการทำงานของเครื่องอ่านรหัสแท่งบันทึกเวลา พร้อมทั้งการตรวจสอบและปรับปรุงแก้ไข

1.3.6 ศึกษาและพัฒนาโปรแกรมจัดการฐานข้อมูลด้วยโปรแกรม FoxPro

1.3.7 เชื่อมต่อ (Interface) เครื่องอ่านรหัสแท่งบันทึกเวลาเข้ากับไมโครคอมพิวเตอร์ และทดสอบซอฟท์แวร์ร่วมกับฮาร์ดแวร์ที่สร้างขึ้นมา ตรวจสอบและดำเนินการแก้ไข

1.3.8 สรุปผลการทดลองของโครงงานนี้ทั้งหมด ข้อเสนอแนะ และปัญหาในการทำโครงงาน พิมพ์รายงาน แก้ไขและเสนอรายงาน

บทที่ 2
ทฤษฎีรหัสแท่ง

รหัสแท่ง (Barcode) คือสัญลักษณ์พิเศษแบบหนึ่งที่ถูกออกแบบมาเพื่อประโยชน์ทางด้านการบันทึกข้อมูลเข้าเครื่องคอมพิวเตอร์โดยเฉพาะข้อมูลที่ซ้ำๆกันหรือข้อมูลที่อาจจะทำให้เกิดความผิดพลาดได้ง่าย หรือต้องการพัฒนาความรวดเร็วในการทำงานด้านการบันทึกข้อมูลต่างๆ แต่อย่างไรก็ดี เป้าหมายหลักของรหัสแท่งก็คือใช้แทนการบันทึกข้อมูลจากการกดแป้นพิมพ์

ทั้งนี้เนื่องจากการอ่านข้อมูลจากรหัสแท่งจะทำงานได้เร็วกว่าการบันทึกข้อมูลเข้าเครื่องโดยการใช้แป้นพิมพ์ค่อนข้างมากสมมติว่า ถ้าเรามีข้อมูล "8850427130017" ที่จะต้องคีย์เข้าเครื่องคอมพิวเตอร์ จะเห็นว่าถ้าเราใช้วิธีการบันทึกข้อมูลจากแป้นพิมพ์ เราจะต้องกดแป้นตัวเลขทั้งหมด 13 ครั้ง ซึ่งสำหรับคนที่คีย์ข้อมูลได้เร็วมากและเกิดความผิดพลาดน้อยเวลาที่ใช้เฉลี่ยจะประมาณ 8 วินาที สำหรับใส่ข้อมูลชุดนี้เข้าไปในเครื่องคอมพิวเตอร์ สาเหตุเกิดจากขณะที่เรากดแป้นพิมพ์แต่ละตัวนั้น ตามธรรมชาติของคนจะเกิดการชะงักขณะจะเลื่อนนิ้วไปกดแป้นพิมพ์ตัวถัดไป แต่ถ้าเราใช้วิธีการอ่านค่า (Scan) จากรหัสแท่ง ที่ใช้แทนข้อมูลชุดนี้ เราจะใช้เวลาประมาณ 1 วินาที และสามารถรับประกันได้ว่าข้อมูลที่ได้จากการอ่านรหัสแท่งจะไม่ผิดเลย

โดยทั่วไปรหัสแท่งจะประกอบไปด้วยองค์ประกอบหลักอยู่ 2 อย่าง คือ แท่งสี่เหลี่ยมแบบทึบ (หรือสีเข้ม) กับแท่งสี่เหลี่ยมแบบสว่าง (หรือช่องว่าง) ซึ่งองค์ประกอบทั้ง 2 ส่วนนี้จะถูกนำมากำหนดประเภทของรหัสแท่ง เช่น รหัสแท่งประเภทที่ใช้แท่งทึบแคบกับแท่งทึบกว้างสำหรับแทนค่า 0 กับ 1 ในระบบเลขฐานสอง หรือประเภทที่ใช้แท่งทึบมืดกับแท่งทึบสว่างแทนค่า 0 กับ 1 ในระบบเลขฐานสอง เป็นต้น และองค์ประกอบทั้งสองส่วนนี้ จะถูกนำมาผสมกันตามรูปแบบของรหัสแท่งแต่ละชนิด (Type) ซึ่งในปัจจุบันมีรหัสแท่งชนิดต่างๆ อยู่ด้วยกันหลายร้อยชนิด แต่มีเพียงไม่กี่ชนิดที่นิยมใช้กัน หรือเห็นกันอยู่ทั่วไป

นอกจากองค์ประกอบหลักๆ ตามที่กล่าวข้างต้นที่รหัสแท่งทุกชนิดจะมีเหมือนกันแล้ว ในรหัสแท่งแต่ละชนิดยังต้องประกอบด้วยส่วนประกอบที่สำคัญอีก 3 ส่วน ด้วยกันคือ

1. ส่วนเริ่มต้นเป็นแท่งทึบที่อยู่ด้านซ้ายสุดของชุดรหัสแท่ง แบบที่วางตัวในแนวเส้นตรง (มีรหัสแท่งบางชนิดจะมีการวางแท่งทึบรหัสแท่งเป็นวงกลม) ซึ่งจะใช้เป็นจุดเริ่มต้นของการอ่านรหัสแท่ง นอกจากนี้ยังใช้เป็นตัวแบ่งแยกชนิดของรหัสแท่งด้วย

2. แท่งทึบเส้นรหัสแท่งที่ใช้แทนข้อมูลเป็นแท่งทึบรหัสแท่งที่ใช้แทนข้อมูล ซึ่งรหัสแท่งบางชนิดจะใช้แทนค่าตัวเลขได้เพียงอย่างเดียวเช่น รหัสเอียน13 , รหัสแทรก 2ใน5 เป็นต้น หรือบางชนิดสามารถใช้แทนข้อมูลได้ทั้งที่เป็นตัวเลขและตัวอักษรอื่นๆ เช่น รหัส39 เป็นต้น

3. ส่วนปิดท้ายเป็นแท่งทึบเส้นรหัสแท่งที่อยู่ด้านขวาสุดของตัวรหัสแท่งแบบที่วางตัวในแนวเส้นตรง ซึ่งจะใช้เป็นตัวบอกจุดสิ้นสุดของการอ่านรหัสแท่ง และใช้ประกอบในการแยกชนิดของรหัสแท่งในเครื่องอ่านรหัสแท่งอีกด้วย

ในรหัสแท่งทุกชนิดจะต้องมีส่วนประกอบหลักตามที่กล่าวมาข้างต้น (ดูรูปที่ 2.1) แต่บางชนิดอาจจะมีส่วนประกอบพิเศษอื่นๆ เพิ่มขึ้นมา เช่น รหัสแท่งในกลุ่ม ยูพีซี/เอียน จะมีส่วนที่เรียกว่า ส่วนกั้นกลาง (Center Bar) เพิ่มขึ้นมา

รูปที่ 2.1 ส่วนประกอบของรหัสแท่งทั่วไป

ความแตกต่างของรหัสแท่งที่ได้รับการพัฒนาขึ้นมาใช้งานในทุกวันนี้ มีลักษณะรูปแบบต่างๆ มากมาย ซึ่งจะขึ้นอยู่กับรูปแบบการตรวจเช็คความผิดพลาด ความหนาแน่นในการพิมพ์รหัสตัวอักขระต่อนิ้ว ชนิดของตัวอักขระที่ใช้งาน ไม่ว่าจะเป็นตัวอักขระหรือว่าตัวเลข ซึ่งสามารถนำมาเข้ารหัสและประยุกต์ใช้งานจริงได้ รูปแบบของรหัสแท่งที่นิยมใช้กันในปัจจุบันมีดังนี้

2.1 <u>รหัส 39 (Code 39)</u>

รหัส 39 ประกอบด้วยส่วนประกอบส่วนกว้าง 3 ส่วน ซึ่งเป็นแท่งทึบ (Bar) และ แท่งขาว หรือ ช่องว่าง (Space) จากทั้งหมด 9 ส่วน รหัสแท่ง 1 ชุด จะประกอบด้วย

1. บริเวณขอบเผื่อ (Quiet Zone) ที่อยู่แต่ละด้านของรหัสแท่ง
2. ส่วนแสดงการเริ่มต้นและหยุดของรหัสแท่ง
3. ข้อมูลของตัวอักขระ

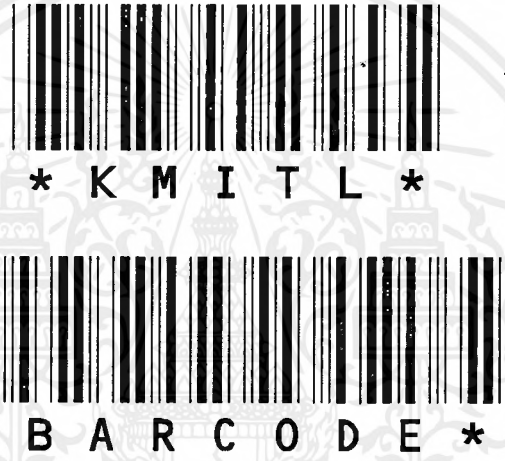
ในรหัส 39 จะแทนที่รหัสเลขฐานสอง ด้วยความกว้างและแคบของแท่งทึบและช่องว่าง โดยแท่งทึบหรือช่องว่างที่แคบจะแทนด้วยเลขฐานสอง "0" และแท่งทึบหรือช่องว่างที่กว้างจะแทนด้วยเลขฐานสอง "1" ดังนั้นรหัสข้อมูล 1 อักขระของรหัส 39 จะประกอบด้วยส่วนกว้าง 3 ส่วน จึงมีเลขฐานสอง "1" อยู่ 3 บิท และที่เหลือจะเป็น "0" อีก 6 บิท

ตารางที่ 2.1 ชุดอักขระและรูปแบบของรหัส 39

อักขระ	รูปแบบ	แท่งทึบ ช่องว่าง	อักขระ	รูปแบบ	แท่งทึบ ช่องว่าง
1		10001 0100	M		11000 0001
2		01001 0100	N		00101 0001
3		11000 0100	O		10100 0001
4		00101 0100	P		01100 0001
5		01100 0100	Q		00011 0001
6		01100 0100	R		10010 0001
7		00011 0100	S		01010 0001
8		10010 0100	T		00110 0001
9		01010 0100	U		10001 1000
0		00110 0100	V		01001 1000
A		10001 0010	W		11000 1000
B		01001 0010	X		00101 1000
C		11000 0010	Y		10100 1000
D		00101 0010	Z		01100 1000
E		10100 0010	-		00011 1000
F		01100 0010	.		10010 1000
G		00011 0010	Space		01010 1000
H		10010 0010	*		00110 1000
I		01010 0010	$		00000 1110
J		00110 0010	/		00000 1101
K		10001 0001	+		00000 1011
L		01001 0001	%		00000 0111

รหัส 39 จะประกอบด้วยรหัสเริ่มต้น (Start ,"*") ทางด้านซ้ายสุด ขอบเขตระหว่างรหัสเริ่มต้นและรหัสหยุด (Stop , "*") จะเป็นส่วนของข้อมูลซึ่งสามารถบรรจุข้อมูลสูงสุดได้ถึง 32 ตัวอักขระ แต่ก็จะขึ้นอยู่กับความสามารถของอุปกรณ์ที่ใช้ร่วมด้วย

ตัวอย่างการแทนข้อมูลคำว่า "KMITL"และ"BARCODE" ของรหัส 39 แสดงในรูปที่ 2.2 ตัวอักขระทั้งหมดและรูปแบบการเข้ารหัสของรหัส 39 แสดงไว้ในตารางที่ 2.1 แท่งทึบ (Bar) และช่องว่าง (Space) แต่ละส่วนจะแคบหรือกว้าง ขึ้นอยู่กับการเข้ารหัสตัวอักขระแต่ละตัว ซึ่งประกอบด้วยส่วนกว้าง 3 ส่วน และส่วนแคบ 6 ส่วน เลขฐานสอง "1" ใช้แทนส่วนกว้าง และเลขฐานสอง "0" แทนส่วนแคบ ตัวอักษรอื่นๆแบ่งแยกโดยช่องว่างระหว่างรหัสตัวอักขระ



รูปที่ 2.2 รูปแบบของรหัส 39

2.2 <u>รหัสแทรก 2 ใน 5 (Interleaved 2 of 5)</u>

รหัสแทรก 2 ใน 5 เป็นรหัสแท่งที่ใช้ในการขนส่งพัสดุหีบห่อ รหัสที่ใช้มีเพียงตัวเลข 0-9 ดังที่แสดงไว้ในตารางที่ 2.2 โดยมีความกว้าง 2 ระดับ ทั้งแท่งทึบและช่องว่าง อัตราส่วนของความกว้างต่อความแคบอยู่ระหว่าง 2 ถึง 3 เท่า ต่อ 1

รหัสอักขระแต่ละตัวถูกแทนด้วยส่วนกว้าง 2 ส่วน ในจำนวนทั้งหมด 5 ส่วน โดยที่แท่งทึบและช่องว่างแคบถูกแทนด้วยเลขฐานสอง "0" เช่นเดียวกับแท่งทึบหรือช่องว่างที่กว้างแทนด้วยเลขฐานสอง "1" รหัสแทรก 2 ใน 5 นี้จะเริ่มต้นด้วยรหัส 0000 และปิดท้ายด้วยรหัส 100 โดยข้อความจะบรรจะอยู่ระหว่างรหัสเริ่มต้นและรหัสสิ้นสุด คือ ถูกสอดแทรก (Interleaved) นั่นเอง ดังนั้นรหัสอักขระตัวแรกจะตามรหัสเริ่มต้น และรหัสอักขระตัวที่สองจะแทรกอยู่ในช่องว่าง ของอักขระตัวแรก (FSFSFSFSFS) โดยให้ F แทนรหัสอักขระตัวแรก และ S แทนรหัสอักขระตัวที่สอง

ตารางที่ 2.2 รูปแบบของรหัสแทรก 2 ใน 5

ตัวอักขระ	รหัสแทรกฐานสอง	รูปแบบ
1	10001	
2	01001	
3	11000	
4	00101	
5	10100	
6	01100	
7	00011	
8	10010	
9	01010	
0	00110	

รหัสแทรก 2 ใน 5 เป็น รหัสแท่งที่มีความจุสูง นั่นคือ จำนวนรหัสต่อตัวอักขระน้อย ความจริงนั้นรหัสตัวอักขระที่ถูกแทรกเข้าไปนั้นยังหมายถึงการแบ่งแยกรหัส (รหัสเป็นลักษณะต่อเนื่อง) แต่ละรหัสอักขระจะมีการตรวจสอบตัวเอง โดยรหัสอักขระแต่ละตัวประกอบไปด้วยส่วนกว้าง 2 ส่วน และส่วนแคบ 3 ส่วน

รหัสนี้จะมีค่า check sum ที่เลือกขึ้นมาจากเกณฑ์การคูณด้วย 10 ตัวอย่างเช่น ข้อมูล "57654823" check sum ถูกคำนวณโดยกำหนดให้รหัสอักขระด้านขวาสุดเป็นตำแหน่ง E (even) ดังนั้นผลรวมของรหัสอักขระในตำแหน่ง O (odd) เท่ากับ 17 ส่วนผลรวมในตำแหน่ง E (even) เท่ากับ 23 แล้วนำไปคูณด้วย 3 ได้ 69 จากนั้นนำผลการคำนวณทั้งสองมารวมกัน ได้ผลลัพธ์เท่ากับ 86 (69+17) ดังนั้นจะได้ check sum เท่ากับ 4 (check sum คือเลขจำนวนที่น้อยที่สุด ซึ่งบวกเข้ากับผลลัพธ์ทั้งหมดที่หาได้แล้วหารด้วย 10 ลงตัว (86+4 = 90)

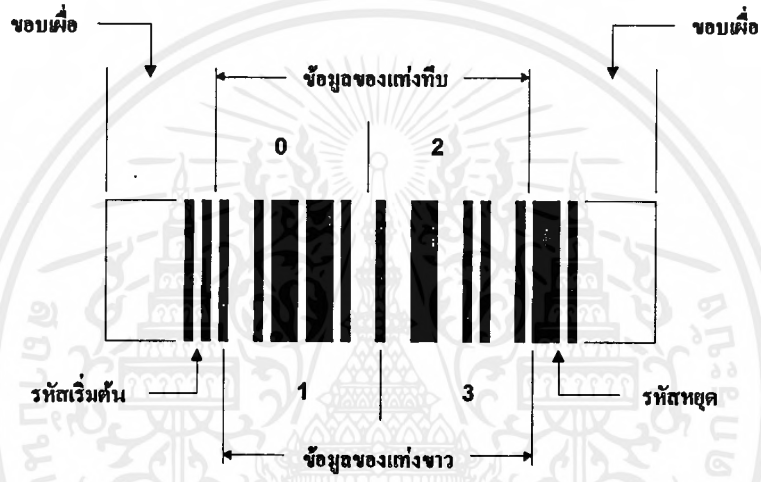
ดังนั้นข้อมูลที่เพิ่มอักขระตรวจสอบคือ "576548234" อย่างไรก็ตาม จำนวนตัวอักขระที่จะเข้ารหัสต้องเป็นคู่ๆ ดังนั้นถ้าข้อมูลนั้นไม่เป็นคู่ เราจะทำให้เป็นจำนวนคู่โดยการเติม 0 ข้างหน้าข้อมูลนั้น ฉะนั้นข้อมูลใหม่จะถูกเข้ารหัสเป็น "0576548234"

ตัวอย่างการเข้ารหัสของข้อมูล "0123" โดยไม่มี check sum ทำได้โดยการแบ่งข้อมูลออกเป็นคู่ คู่แรกของข้อมูลคือ "01" และคู่หลังคือ "23" จากนั้นก็แปลงข้อมูลคู่แรกคือ "01" เป็นรหัสเลขฐานสองโดยตัวเลข 0 แทนด้วยรหัส 00110 และตัวเลข 1 แทนด้วย 10001 แล้วนำรหัสที่ได้มารวมกันแบบสอดแทรก นั้นคือให้อักขระตัวหลัง (1) อยู่ในตำแหน่งคี่และอักขระตัวแรก (0) อยู่ในตำแหน่งคู่ ฉะนั้นจะได้รหัสของคู่แรกคือ 01 เป็น 0100101001 ส่วนการแปลงรหัสในคู่หลังนั้นก็เหมือนกับคู่แรกคือ อักขระเลข 2 แทนด้วยรหัส 01001 และอักขระเลข 3 แทนด้วย 11000

แล้วนำมาแทรกสอดกันได้รหัสของคู่หลังเป็น 0111000010

ผลสุดท้ายก็นำรหัสที่ได้จากข้อมูลในคู่แรกกับคู่หลังมารวมกันโดยเพิ่มรหัสเริ่มต้นคือ 0000 ไว้ข้างหน้า และรหัสหยุดคือ 100 ไว้ข้างหลังสุด ดังนี้ 0000/0100101001011100010/100

ส่วนแสดงการเริ่มต้นและสิ้นสุดของรหัสแท่ง แสดงในรูปที่ 2.3 รหัสเริ่มต้นจะอยู่ทางซ้ายของข้อมูลทั้งหมด ประกอบด้วย 4 ส่วนแคบ ๆ (0000) โดยสลับกันระหว่างแท่งทึบและช่องว่าง รหัสหยุดจะอยู่ทางขวาของข้อมูลทั้งหมด ประกอบด้วยแท่งทึบกว้างและช่องว่างแคบ ตามด้วยแท่งทึบแคบ (100) และในส่วนประกอบของรหัสแท่งจะมีส่วนช่องว่างที่เป็นขอบเผื่อ (Quit Zone) ที่อยู่ปิดหัวท้ายของรหัสแท่ง ดังแสดงในรูปที่ 2.3



รูปที่ 2.3 รูปแบบของรหัสแทรก 2 ใน 5

2.3 รหัสโคด้าบาร์ (Codabar)

รหัสโคด้าบาร์ มีความกว้าง 2 ระดับ สามารถใช้กับข้อมูลตัวเลขและตัวอักขระพิเศษ 6 ตัว คือ $, - , : , / , . และ + และตัวอักษรอีก 4 ตัวที่ใช้เป็นรหัสแสดงการเริ่มต้นและหยุด คือ A,B,C และ D

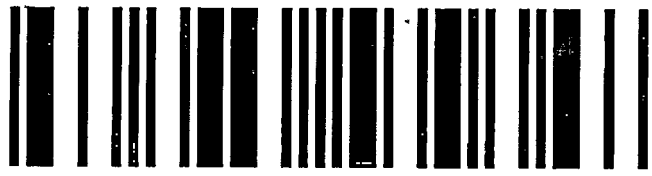
รหัสโคด้าบาร์แต่ละชุดประกอบด้วยขอบเขตแสดงการสิ้นสุด,ส่วนแสดงการเริ่มต้นหรือหยุดและส่วนของข้อมูล ซึ่งสามารถบรรจุข้อมูลได้ถึง 32 ตัวอักษร ตัวอย่างดังแสดงในรูป 2.4 รหัสอักขระแต่ละตัวแทนด้วยส่วนประกอบ 7 ส่วน แบ่งเป็นแท่งทึบ 4 แท่ง และช่องว่าง 3 ช่อง เนื่องจากใช้ 7 บิตต่อ 1 อักขระ ดังนั้นจึงสามารถมีรหัสได้ถึง $2^7 = 128$ รหัส แต่นำมาใช้เพียง 20 รหัสเท่านั้น ทำให้รหัสแบบนี้เป็นแบบตรวจสอบตัวเองโดยธรรมชาติ และไม่มีข้อกำหนดเกี่ยวกับรหัสตรวจสอบ (check sum)

ตารางที่ 2.3 รูปแบบของรหัสโคด้าบาร์

ตัวอักขระ	รหัสเลขฐานสอง	รูปแบบของรหัสแท่ง
0	0000011	
1	0000110	
2	0001001	
3	1100000	
4	0010010	
5	1000010	
6	0100001	
7	0100100	
8	0110000	
9	1001000	
-	0001100	
$	0011000	
:	1000101	
/	1010001	
.	1010100	
+	0010101	
A	0011010	
B	0101001	
C	0001011	
D	0001110	

รหัสโคด้าบาร์ทั้งหมด แสดงไว้ในตารางที่ 2.3 ซึ่งส่วนกว้างแทนเลขฐานสอง “1” และส่วนแคบแทนเลขฐานสอง “0” สำหรับข้อมูล “2345” ถ้าเข้ารหัสแบบโคด้าบาร์ โดยมี “A” เป็นรหัสเริ่มต้นและจบ (“A2345A”) ก็จะได้รหัสดังนี้

“0011010/0001001/1100000/0010010/1000010/0011010”



รูปที่ 2.4 รูปแบบรหัสแท่ง ของรหัสโคด้าบาร์

2.4 รหัสยูพีซี (UPC , Universal Product Code)

รหัสแท่งชนิดนี้ใช้แทนตัวเลขที่มีจำนวนที่แน่นอนไม่สามารถใช้แทนตัวอักษรได้ จะใช้แทนรหัสสินค้าแบบ 10 หลัก โดย 5 หลักทางด้านซ้ายจะหมายถึงรหัสผู้ผลิต (Manufactury Code) และ 5 หลักทางด้านขวาจะใช้แทนลำดับที่ของชนิดสินค้าที่ผู้ผลิตผลิตออกจำหน่าย (Product Item) นอกจากนี้ยังมีตัวเลขเสริมอีก 2 หลัก คือ ตัวเลขระบบ (Number System) จะอยู่ทางด้านซ้ายก่อนรหัสผู้ผลิต สำหรับตัวเลขระบบนี้จะมีค่าตั้งแต่ 0 ถึง 9 ซึ่งในการใช้งานได้มีการกำหนดความหมายเฉพาะไว้ดังตารางที่ 2.4 ส่วนตัวเลขอีกชุดหนึ่งเป็นตัวเลขสำหรับตรวจสอบความถูกต้องของรหัส (Check Digit) ซึ่งจะอยู่ทางด้านซ้ายถัดจากเลขชุดลำดับของชนิดสินค้าของผู้ผลิต ซึ่งตัวเลขทั้งสองหลัก ทางผู้ผลิตจะแสดงค่าตัวเลขประจำหลักบนรหัสแท่งหรือไม่ก็ได้ และไม่ว่าจะแสดงตัวเลขดังกล่าวให้มองเห็นหรือไม่ แต่เมื่ออ่านด้วยเครื่องอ่านก็สามารถตรวจสอบค่าทั้งสองได้เช่นกัน

ตารางที่ 2.4 ตัวเลขระบบ (Number System) ของรหัสยูพีซี

ค่าตัวเลข	การนำไปใช้งาน
0	ใช้กับสินค้าอุปโภค
2	ใช้กับสินค้าที่ต้องชั่งน้ำหนัก เช่นพืชผักผลไม้และเนื้อสัตว์
3	ใช้กับสินค้าพวกยา และ สินค้าเกี่ยวกับสุขภาพ
4	ใช้กับสินค้าทั่วไปที่ร้านค้ากำหนดขึ้นเอง
5	ใช้เป็นรหัสสำหรับคูปอง
อื่นๆ	สำรองไว้ใช้งาน

รหัสยูพีซี เกิดจากการแทนด้วยเลขฐานสอง 7 บิต จำนวน 2 ชุดด้วยกัน โดยทั้ง 2 ชุดจะคอมพลีเมนท์ (Complement) ซึ่งกันและกัน ซึ่งชุดหนึ่งจะใช้แทนเลขทางด้านซ้าย (Manufactury Code) และอีกชุดหนึ่งจะใช้แทนตัวเลขที่อยู่ด้านขวา (Product Item) และจะแทนแท่งทึบด้วยเลขฐานสองค่า “1” และแทนช่องว่างด้วยค่าเลขฐานสอง “0” ดังตารางที่ 2.5 เป็นตารางรูปแบบ ของระบบเลขฐานสอง ที่ใช้แทนตัวเลขของรหัสยูพีซีทั้ง 2 ชุด โดยแสดงในรูปของแท่งรหัสแท่งประกอบด้วย จะพบข้อสังเกตอย่างหนึ่งคือรูปแบบรหัสด้านซ้ายจะจบด้วยแท่งทึบเสมอ ส่วนชุดที่ใช้แทนรหัสด้านขวาจะจบด้วยช่องว่างเสมอ ซึ่งเป็นกฎเกณฑ์อย่างหนึ่งในการกำหนดระบบเลขฐานสอง ที่ใช้แทนค่าตัวเลขของรหัสยูพีซี และเหตุผลที่รหัสทั้ง 2 ชุดเป็นคอมพลีเมนท์กัน ก็เพื่อให้การอ่านรหัสแท่งโดยเครื่องอ่านสามารถทำการอ่านได้ ทั้งจากซ้ายไปขวา และจากขวามาซ้าย ซึ่งเป็นคุณสมบัติอย่างหนึ่งในรหัสแท่งหลายชนิด

ตารางที่ 2.5 รูปแบบของรหัสยูพีซี

ค่าตัวเลข	รหัสแถบฐานสอง-ด้านซ้าย	รหัสแถบฐานสอง-ด้านขวา
0	0001101	1110010
1	0011001	1100110
2	0010011	1101100
3	0111101	1000010
4	0100011	1011100
5	0110001	1001110
6	0101111	1010000
7	0111011	1000100
8	0110111	1001000
9	0001011	1110100

นอกจากนี้ในรหัสยูพีซียังมีรหัสที่ไม่สามารถอ่านค่าเป็นตัวเลขได้อีก 3 ชุดคือ

1. รหัสกั้นซ้าย (Left Guard Band) เป็นเส้นแท่งที่อยู่ทางด้านซ้ายสุดของตัวรหัสแท่งจะใช้เป็นตัวบอกจุดเริ่มต้นของรหัสแท่ง (หรือเป็นจุดสิ้นสุดการอ่านรหัสแท่ง กรณีเริ่มอ่านจากด้านขวา) โดยเส้นแท่งบริเวณนี้จะให้ค่าเป็นเลขฐานสอง 3 บิต คือ 101

2. รหัสกั้นขวา (Right Guard Band)เป็นเส้นแท่งที่อยู่ทางขวาสุด ใช้ในการบอกจุดสิ้นสุด (หรือจุดเริ่มในการอ่านจากขวามาซ้าย) ของการอ่านรหัสแท่งเช่นเดียวกับรหัสกั้นซ้าย เส้นแท่งบริเวณนี้จะให้ค่าเป็นเลขฐานสองซึ่งมีสองแบบคือ ถ้าเป็น ยูพีซี-เอ จะให้เป็นเลขฐานสอง 3 หลัก คือ 101 ถ้าเป็น ยูพีซี-อี จะให้ค่าเป็นเลขฐานสอง 6 หลักคือ 010101

3. รหัสกั้นกลาง (Center Bar) ซึ่งจะทำหน้าที่ในการแยกรหัสตัวเลขด้านซ้ายและด้านขวา โดยจะให้ค่าเป็นเลขฐานสอง 5 หลักคือ 01010

การตรวจสอบความถูกต้องของรหัส (Check Digit) ของรหัสยูพีซี มีวิธีการคิดดังนี้

1. หาผลรวมของตัวเลขจากด้านขวามาซ้ายทีละหลักเว้นหลัก โดยไม่รวมตัว Check Digit
2. นำผลรวมจากข้อ 1 คูณด้วย 3
3. หาผลรวมของหลักที่เหลือ
4. นำข้อ 2 บวก ข้อ 3
5. หาตัวเลข 0-9 ที่เมื่อมาบวกกับข้อ 4 แล้วมีค่าเต็ม 10

รูปที่ 2.5 รูปแบบของรหัสยูพีซี

รหัสยูพีซีแบ่งเป็นหลายประเภทดังนี้ คือ

2.4.1 รหัสยูพีซี-เอ (UPC-A)

รหัสยูพีซี-เอ เป็นรหัสพื้นฐานของรหัสยูพีซี ที่ได้ถูกสร้างขึ้นเป็นแบบแรก มีโครงสร้างที่เป็นพื้นฐานของรหัสยูพีซีแบบอื่น ตามที่กล่าวมาแล้วข้างต้น นิยมใช้ในสินค้าอุปโภค และบริโภค สำหรับประเทศสหรัฐอเมริกาและแคนาดา

2.4.2 รหัสยูพีซี-อี (UPC-E)

รหัสยูพีซี-อี เป็นรหัสที่ใช้ในการแทนค่ารหัสเหมือนกับรหัสยูพีซี-เอ แต่โครงสร้างจะแตกต่างกันเล็กน้อยคือ รหัสยูพีซี-อี จะมีจำนวนข้อมูลเพียง 6 ตัว ซึ่งเสมือนการตัดเอาเฉพาะข้อมูลด้านซ้ายของรหัสยูพีซี-เอมาใช้ คือมีเฉพาะรหัสกั้นซ้ายรหัสข้อมูลและรหัสกั้นกลางกลาง

2.4.3 รหัสยูพีซี-บี (UPC-B)

รหัส ยูพีซี-บี เป็นรหัสยูพีซีแบบที่พัฒนาขึ้นมาจากรหัสยูพีซี-เอ เพื่อใช้ในงานด้านยา และสาธารณสุขแห่งชาติของประเทศสหรัฐอเมริกา โดยโครงสร้างของรหัสที่แตกต่างก็เพียงแต่ รหัสยูพีซี-บี จะไม่มีรหัสตรวจสอบ คือรหัสตัวสุดท้ายของข้อมูลด้านด้านขวาจะไม่ใช่รหัสตรวจสอบ แต่จะเป็นรหัสข้อมูล

2.4.4 รหัสยูพีซี-ซี (UPC-C)

รหัสยูพีซี-ซี เป็นรหัสยูพีซีที่พัฒนาขึ้นมาเพื่อใช้ในโรงงานอุตสาหกรรม เนื่องจากเดิมโรงงานอุตสหกรรมยังมิได้มีการนำรหัสแท่งไปใช้งาน (ใช้รหัสตัวเลขธรรมดา) จึงได้มีการพัฒนารหัสยูพีซี-ซี ขึ้นมารองรับความต้องการ โดยโครงสร้างของรหัสที่แตกต่างจากรหัสยูพีซี แบบมาตรฐาน คือ จะมีรหัสข้อมูล 12 ตัว กับรหัสตรวจสอบและรหัสบอกชนิดสินค้า รวมทั้งหมด 14ตัว

2.5 รหัสเอียน (EAN , European - Article Numbering)

รหัสเอียนมีลักษณะคล้ายกับรหัสยูพีซีโดยจะใช้จำนวนตัวเลข 5 หรือ 10 หลักเป็นรหัสหลัก และเพิ่มได้อีก 2 หรือ 5 หลัก โดยใช้เพียงอักขระตัวเลข 0-9 เท่านั้น รหัสเอียนมีความกว้าง 4 ระดับ คือ ในแต่ละแท่งทึบหรือช่องว่างจะมีระดับความกว้าง 1, 2, 3 หรือ 4 โดยให้แท่งทึบแทนด้วยเลขฐานสอง “1” ในขณะที่ช่องว่างแทนด้วยเลขฐานสอง “0” ตัวอย่างเช่น รหัส 00011 จะถูกแทนด้วยช่องว่างที่มีความกว้าง 3 ส่วน และตามด้วยแท่งทึบที่มีความกว้าง 2 ส่วน

รหัสอักขระแต่ละตัวถูกสร้างขึ้นด้วยเลขฐานสองขนาด 7 บิต โดยรหัสแท่งชุดหนึ่ง ๆ จะประกอบด้วยรหัสกั้นซ้าย รหัสกั้นกลาง และรหัสกั้นขวา รหัสที่อยู่ทางด้านซ้ายของรหัสกั้นกลางถูกเข้ารหัสโดยการใช้คอลัมน์ด้านด้านซ้าย ดังแสดงในตารางที่ 2.6 ดังนั้นอักขระที่อยู่ทางด้านด้านขวาของรหัสกั้นกลางก็จะเข้ารหัสโดยใช้คอลัมน์ด้านด้านขวา ความแตกต่างระหว่างสองคอลัมน์ทางด้านด้านซ้ายนั่นคือ คอลัมน์ A จะใช้กับข้อมูลที่เป็นพาริตี้คี่ (odd parity) และคอลัมน์ B สำหรับข้อมูลที่เป็นพาริตี้คู่ (even parity)

ตารางที่ 2.6 รูปแบบของรหัสเอียน

ตัวอักขระ	รหัสเลขฐานสอง ด้านซ้าย A	รหัสเลขฐานสอง ด้านซ้าย B	รหัสเลขฐานสอง ด้านขวา
0	0001101	0100111	1110010
1	0011001	0110011	1100110
2	0010011	0011011	1101100
3	0111101	0100001	1000010
4	0100011	0011101	1011100
5	0110001	0111001	1001110
6	0101111	0000101	1010000
7	0111011	0010001	1000100
8	0110111	0001001	1001000
9	0001011	0010111	1110100

รหัสเอียนจะใช้ข้อมูล 3 หลักแรกเป็นรหัสประเทศ (country code) ของผู้ผลิตสินค้า ดังแสดงในตารางที่ 2.7 รหัสเอียน-8 เป็นรหัสที่มีความยาวของข้อมูล 5 หลัก ส่วนรหัสเอียน-13 นั้นเป็นรหัสที่มีความยาวของข้อมูล 10 หลัก โดยที่ 4 หลัก เป็นรหัสของผู้ผลิตสินค้า (manufacture identify code) 5 หลัก ถัดมาเป็นรหัสสินค้า (product code) หลักสุดท้ายคือรหัสตรวจสอบ (check code) รหัสเอียนทั้ง 2 ชนิดนี้ก็ประกอบไปด้วยรหัสกั้นหน้า รหัสกั้นกลาง และรหัสกั้นหลัง รหัสแฟลก(Flag Character) 2 ตัว และอาจจะมี รหัส 2 หลัก หรือ รหัส 5 หลักเพิ่มขึ้นมาอีก

ตารางที่ 2.7 รหัสประเทศของรหัสเอียน

เลขนำหน้า	รหัสของประเทศ	เลขนำหน้า	รหัสของประเทศ
00 - 09	สหรัฐอเมริกาและแคนาดา	76	สวิทเซอร์แลนด์
20 - 29	สำรองไว้	770	โคลัมเบีย
30 - 37	ฝรั่งเศส	773	อุรุกวัย
40 - 43	เยอรมันตะวันตก	775	เปรู
440	เยอรมันตะวันออก	779	อาเจนตินา
460 - 469	โซเวียต	780	ชิลี
471	ไต้หวัน	789	บราซิล
489	ฮ่องกง	80 - 83	อิตาลี
49	ญี่ปุ่น	84	สเปน
50	อังกฤษและไอร์แลนด์	859	เชคโกสโวาเกีย
520	กรีซ	860	ยูโกสลาเวีย
529	ไซปรัส	869	ตุรกี
54	เบลเยี่ยมและลักเซมเบิร์ก	87	เนเธอแลนด์
560	โปรตุเกส	880	เกาหลีใต้
569	ไอซแลนด์	885	ไทย
57	เดนมาร์ก	888	สิงคโปร์
599	ฮังการี	90 - 91	ออสเตรีย
600 - 601	แอฟริกาใต้	93	ออสเตรเลีย
64	ฟินแลนด์	94	นิวซีแลนด์
70	นอรเวย์	955	มาเลเซีย
729	อิสราเอล	959	ปาปัว นิวกีนี
73	สวีเดน	977	รหัสสำหรับวารสาร
750	เม็กซิโก	978 - 979	รหัสสำหรับหนังสือ
759	เวเนซุเอลา	98 - 99	คูปอง

2.5.1 รหัสเอียน-8

รหัสเอียน-8 นั้นประกอบไปด้วยส่วนต่างตามลำดับดังนี้

- รหัสกั้นหน้า ซึ่งเข้ารหัสด้วย 101
- อักขระแฟลก 2 ตัว เข้ารหัสด้วยคอลัมน์ด้านซ้าย A (ดูตารางที่ 2.6)
- ข้อมูลอักขระสองตัวแรกเข้ารหัสด้วยรหัสคอลัมน์ด้านซ้าย A เช่นกัน
- รหัสกั้นกลาง ซึ่งการเข้ารหัสด้วย 01010
- ข้อมูลอักขระ 3 ตัวหลังเข้ารหัสด้วยคอลัมน์ด้านขวา
- อักขระตรวจสอบซึ่งเข้ารหัสด้วยคอลัมน์ด้านขวา
- รหัสกั้นหลังเข้ารหัสด้วย 101

รหัสเอียน-8 ซึ่งมีข้อมูลเป็น “80123453” สามารถแทนด้วยรหัสเลขฐานสองได้ดังนี้ คือ 101/0110111/0001101/0011001/0010011/01010/1000010/1011100/1001110/100010/101 (เพื่อให้ดูง่ายๆ จะใช้เครื่องหมาย “/ “ ขั้นระหว่างแต่ละรหัส)

อักขระตรวจสอบสามารถหาได้โดยการสมมติว่า ตัวอักขระขวาสุดเป็นตำแหน่งที่ (Odd) □ EOEOEO และบวกอักขระทั้งหมดในตำแหน่งคี่ (Odd) แล้วคูณด้วย 3 ได้เป็นผลลัพธ์แรก ส่วนผลลัพธ์ที่ 2 หาจากผลรวมของรหัสอักขระทั้งหมดในตำแหน่งคู่ (even) ดังนั้นผลลัพธ์ที่ได้คือ ผลรวมของทั้ง 2 กรณีข้างต้น

อักขระตรวจสอบคือจำนวนเลขที่น้อยที่สุดที่บวกเข้ากับผลลัพธ์และสามารถบวกเข้ากับจำนวนนั้นได้ 10 ลงตัว (หรือบวกให้หลักหน่วยเป็น 0)

รูปที่ 2.6 รูปแบบรหัสเอียน-8

2.5.2 รหัสเอียน-13

รหัสเอียน-13 นั้นก็มีส่วนประกอบคล้ายคลึงกับรหัสเอียน-8 ดังนี้

- รหัสกั้นหน้า ซึ่งเข้ารหัสด้วย 101
- อักขระแฟลกที่ 2 (second flag character) 1 ตัว เข้ารหัสด้วยคอลัมน์ด้านซ้าย A หรือ B
- ข้อมูลอักขระ 5 ตัวแรกเข้ารหัสด้วยคอลัมน์ด้านซ้าย A หรือ B

- รหัสกั้นกลาง ซึ่งการเข้ารหัสด้วย 01010
- ข้อมูลอักขระ 5 ตัวหลังเข้ารหัสด้วยคอลัมน์ด้านขวา
- อักขระตรวจสอบซึ่งเข้ารหัสด้วยคอลัยน์ด้านขวา
- รหัสกั้นหลังเข้ารหัสด้วย 101

อักขระแฟลกที่ 1 ของรหัสเอียน-13 ถูกเข้ารหัสด้วยการใช้รูปแบบพาริตี้ (Parity Pattern) ของอักขระแฟลกที่ 2 ข้อมูลอักขระ 5 ตัวแรกดังตารางที่ 2.8

ตารางที่ 2.8 รหัสสำหรับอักขระแฟลกที่หนึ่งของ เอียน-13

ตัวเลข	แฟล็ก 2	ข้อมูล 1	ข้อมูล 2	ข้อมูล 3	ข้อมูล 4	ข้อมูล 5
0	A	A	B	B	A	B
3	A	A	B	B	B	A
4	A	B	A	A	B	B
5	A	B	B	A	A	B
6	A	B	B	B	A	A
7	A	B	A	B	A	B
8	A	B	A	B	B	A
9	A	B	B	A	B	A

ตัวอย่างรหัสเอียน-13 กำหนดให้อักขระแฟลก คือ "97" และมีข้อมูลเป็น "7095983300" ตำแหน่งของคู่/คี่ (Even/Odd) เป็นดังนี้ EOEOEOEOEOEO ดังนั้นผลรวมของเลขในตำแหน่งคี่คูณด้วย 3 จะได้เท่ากับ 69 ([7+0+5+8+3+0]*3) และผลรวมของเลขในตำแหน่งคู่เท่ากับ 37 (9+7+9+9+3+0) จากนั้นนำผลรวมทั้งสองข้างต้นมาบวกกันได้เป็นผลลัพธ์เท่ากับ 106 (69+37) ดังนั้นเมื่อนำ 4 บวกเข้ากับ 106 จะได้เท่ากับ 110 ซึ่งหารด้วย 10 ได้ลงตัวพอดีฉะนั้นอักขระตรวจสอบคือ 4 และได้ข้อความเต็มๆดังนี้ 9770959833004

รูปที่ 2.7 รูปแบบรหัสเอียน-13

2.5.3 รหัสเอียนแบบเพิ่ม 2 หลัก

รหัสสองหลักที่เพิ่มขึ้นมาซึ่งอยู่ด้านหน้าของรหัสหลักนั้นจะแสดงหมายเลขเดือน โดยเริ่มจาก 01 สำหรับเดือนมกราคม

ตารางที่ 2.9 รูปแบบพาริตี้ของ 2 หลักที่เพิ่มมาของรหัสเอียน

A-A	A-B	B-A	B-B
00	01	02	03
04	05	06	07
08	09	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31
32	33	34	35
36	37	38	39
40	41	42	43
44	45	46	47
48	49	50	51
52	53	54	55
56	57	58	59
60	61	62	63
64	65	66	67
68	69	70	71
72	73	74	75
76	77	78	79
80	81	82	83
84	85	86	87
88	89	90	91
92	93	94	95
96	97	98	99

รหัส 2 หลักที่เพิ่มขึ้นมานี้ จะประกอบด้วยส่วนต่างๆตามลำดับดังนี้

- รหัสกั้นซ้าย-ขวา เข้ารหัสด้วย 1011
- ข้อมูลอักขระตัวแรก เข้ารหัสด้วยคอลัมน์ด้านซ้าย A หรือ B
- อักขระแยก (character delineator) เข้ารหัสด้วย 01
- ข้อมูลอักขระตัวสอง เข้ารหัสด้วยคอลัมน์ด้านซ้าย A หรือ B

ข้อมูลอักขระที่ถูกเข้ารหัสโดยใช้คอลัมน์ด้านซ้าย A หรือ B นั้น ขึ้นอยู่กับหลักของอักขระนั้น ตัวอย่างเช่น ถ้าส่วนที่เพิ่มขึ้นมานั้นคือ 13 เราจะใช้ตารางที่ 2.9 ในการอ้างอิง ซึ่งจะใช้คอลัมน์ด้านซ้าย A ใช้สำหรับเลข 1 และ 3 ก็จะเข้ารหัสจากคอลัมน์ด้านซ้าย B (เลข 13 อยู่ในคอลัมน์ A-B)

2.5.4 รหัสเอียนแบบเพิ่ม 5 หลัก

รหัสเอียนแบบที่เพิ่มขึ้นมา 5 หลัก นั้นส่วนมากมักจะพบเห็นกันบนหนังสือ หรือนิตยสารที่ป้ายบอกราคาหรือปกหนังสือ ใน 5 หลักที่เพิ่มขึ้นมานั้นประกอบไปด้วย ส่วนต่างๆ ตามลำดับดังนี้

- รหัสกั้นซ้าย-ขวา เข้ารหัสด้วย 1011
- ข้อมูลอักขระตัวแรก เข้ารหัสด้วยคอลัมน์ด้านซ้าย A หรือ B
- อักขระแยกเข้ารหัสด้วย 1
- ข้อมูลอักขระตัวที่สอง เข้ารหัสด้วยคอลัมน์ด้านซ้าย A หรือ B
- อักขระแยกเข้ารหัสด้วย 01
- ข้อมูลอักขระตัวที่สาม เข้ารหัสด้วยคอลัมน์ด้านซ้าย A หรือ B
- อักขระแยกเข้ารหัสด้วย 1
- ข้อมูลอักขระตัวที่สี่ เข้ารหัสด้วยคอลัมน์ด้านซ้าย A หรือ B
- อักขระแยกเข้ารหัสด้วย 01
- ข้อมูลอักขระตัวที่ห้า เข้ารหัสด้วยคอลัมน์ด้านซ้าย A หรือ B

รูปที่ 2.8 รูปแบบรหัสเอียน-8 แบบเพิ่ม 5 หลัก

ตารางที่ 2.10 รูปแบบพาริตี้ของ 5 หลักที่เพิ่มมาของรหัสเอียน

ตัวเลข	ข้อมูล 1	ข้อมูล 2	ข้อมูล 3	ข้อมูล 4	ข้อมูล 5
0	B	B	A	A	A
1	B	A	B	A	A
2	B	A	A	B	A
3	B	A	A	A	B
4	A	B	B	A	A
5	A	A	B	B	A
6	A	A	A	B	B
7	A	B	A	B	A
8	A	B	A	A	B
9	A	A	B	A	B

เช่นเดียวกันตัวอักขระจะถูกเข้ารหัสโดยการใช้คอลัมน์ด้านซ้าย ซึ่งคอลัมน์ที่ใช้นั้นคิดจากค่า check sum ที่คิดเหมือนกับอักขระตรวจสอบของข้อความหลัก โดยสมมติให้อักขระด้านขวาสุดเป็นตำแหน่งคี่ จากนั้นนำอักขระที่อยู่ในตำแหน่งคี่มาบวกกันแล้วคูณด้วย 3 และนำอักขระที่อยู่ตำแหน่งคู่มาบวกกันแล้วคูณด้วย 9 เสร็จแล้วนำผลลัพธ์ทั้งสองมาบวกกัน ก็จะได้ค่า ๆ หนึ่ง โดยเราจะสนใจเฉพาะเลขในหลักหน่วยของค่าผลลัพธ์ที่ได้ซึ่งก็คือหมายเลขของรูปแบบพาริตี้ในตารางที่ 2.10

บทที่ 3
เครื่องอ่านรหัสแท่ง

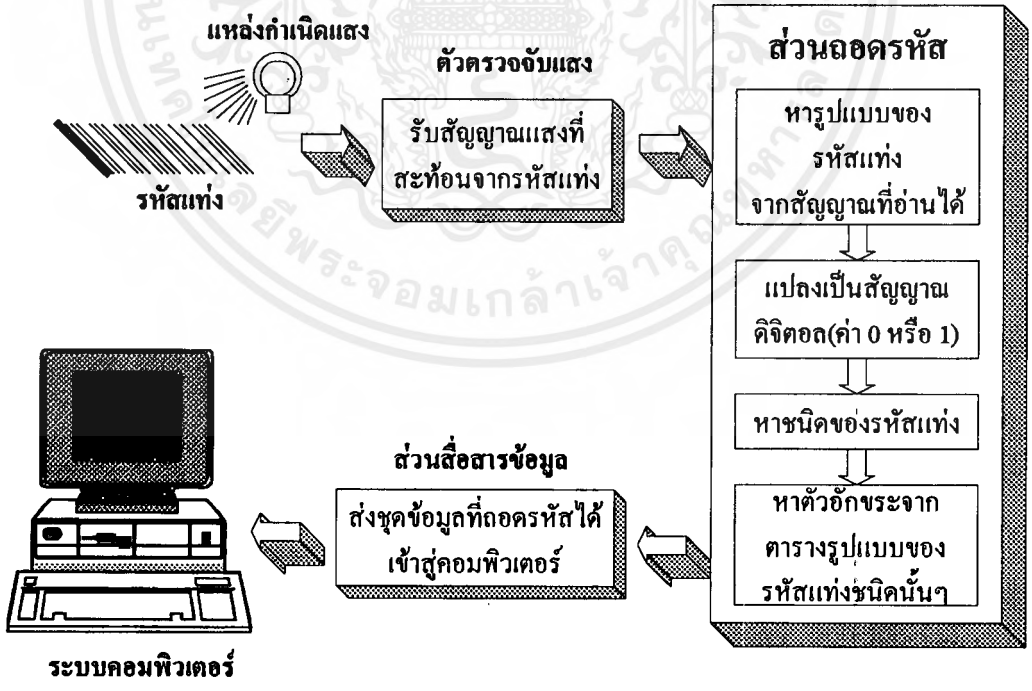
เครื่องอ่านรหัสแท่งใช้ในการแยกข้อมูลที่ถูกจัดเข้าเป็นรหัสแท่ง และแปลงข้อมูลนั้นให้อยู่ในรูปแบบของสัญญาณดิจิตอล (0 และ 1) หลังจากนั้นจึงนำข้อมูลมาทำการถอดรหัส ด้วยหน่วยประมวลผล (Microprocesser unit) ข้อมูลที่ถูกถอดรหัสแล้วจะถูกส่งตรงไปยังเครื่องคอมพิวเตอร์เพื่อนำไปประมวลผลต่อไปหรือเก็บข้อมูลไว้ก่อนเพื่อรอการเรียกใช้กายหลัง หรือใช้กับโปรแกรมประยุกต์ที่อยู่ภายในตัวเครื่องอ่านรหัสแท่งเอง

โดยทั่วไปแล้วเครื่องอ่านรหัสแท่งประกอบด้วย 2 ส่วนดังนี้คือ

- ส่วนหัวอ่าน เป็นส่วนนำข้อมูลเข้าประกอบด้วย แหล่งกำเนิดแสง (Light Source) และตัวตรวจจับแสง (Light Detector)

- ส่วนถอดรหัส เป็นส่วนประมวลผลข้อมูลที่ได้จากหัวอ่านซึ่งจะใช้ซอฟท์แวร์เป็นหลัก

ดังแสดงในรูปที่ 3.1



รูปที่ 3.1 ระบบเครื่องอ่านรหัสแท่ง (Barcode Reader System)

ข้อกำหนดพื้นฐาน 5 ประการสำหรับส่วนถอดรหัส

1. สามารถพิจารณาความกว้างแคบของแท่งทึบ (Bar) และแท่งขาวหรือช่องว่าง (Space)

2. สามารถจัดแบ่งระดับของความกว้าง ทั้งนี้ขึ้นอยู่กับชนิดของรหัสแท่งที่ใช้งาน เช่น

- แบ่งความกว้างเป็น 2 ระดับ สำหรับรหัส 39 , รหัสแทรก 2 ใน 5 เป็นต้น
- แบ่งความกว้างเป็น 4 ระดับ สำหรับรหัสแท่งกลุ่ม ยูพีซี/เอียน

3. ให้ความมั่นใจได้ว่าความกว้างที่ถูกจัดแบ่งเหมาะสมกับการถอดรหัสแต่ละชนิดสามารถเปรียบเทียบโครงสร้างความกว้างที่อ่านได้กับตารางรูปแบบหลักของรหัสแท่งนั้นๆ เพื่อเปลี่ยนเป็นรหัสแอสกี้ (ASCII code)

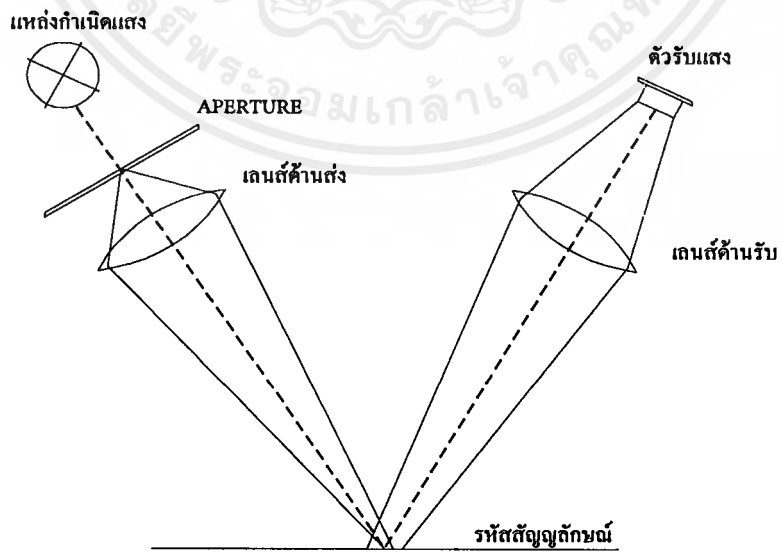
4. ถ้าการกลับลำดับ (Reverse) มีผลต่อการถอดรหัส ทิศทางการอ่านจะถูกกำหนดโดยการตรวจสอบรหัสเริ่มต้น (Start code) และ รหัสหยุด (Stop code)

5. ยืนยันได้ว่ามีบริเวณขอบเผื่อ (Quite Zone) ที่ปลายทั้งสองข้างของรหัสแท่ง

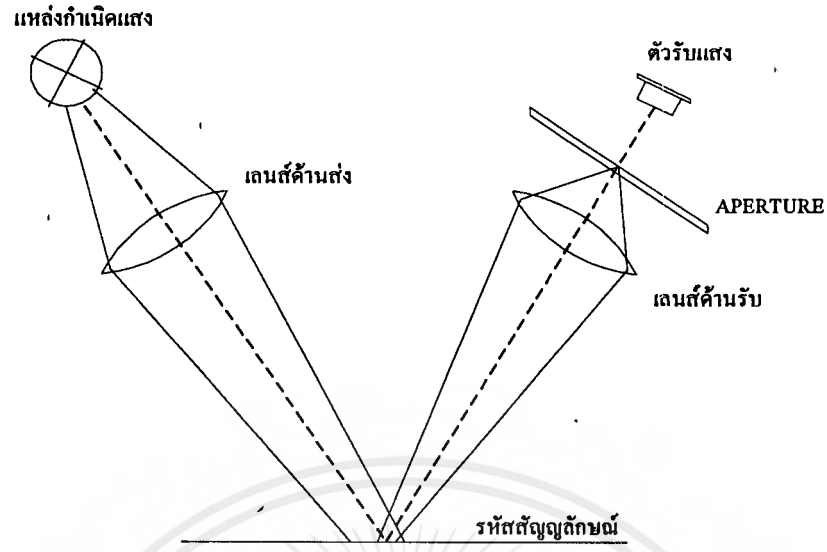
3.1. หัวอ่านรหัสแท่ง (BarCode Scanner)

เป็นส่วนนำเข้าข้อมูล (Input Device) ที่ใช้เทคนิคการเปลี่ยนแสงไปเป็นสัญญาณไฟฟ้าในการสแกน (Scan) ผ่านรหัสแท่ง ลักษณะการสแกนขึ้นอยู่กับการเคลื่อนไหวมือของผู้ใช้หรือขึ้นกับการสแกนภายในหัวอ่านเอง สัญญาณเอาท์พุทที่ได้คือผลของการสะท้อนแสง ณ จุดที่ถูกสแกน

อุปกรณ์ส่วนนี้มักจะเป็นแบบแอคทีฟ (Active System) ซึ่งทำงานโดยการส่งพลังงานแสงไปยังรหัสแท่ง แล้วตรวจสอบพลังงานแสงที่สะท้อนกลับมา ส่วนที่เป็นช่องว่างจะสะท้อนแสงได้มากกว่าส่วนที่เป็นแท่งทึบ

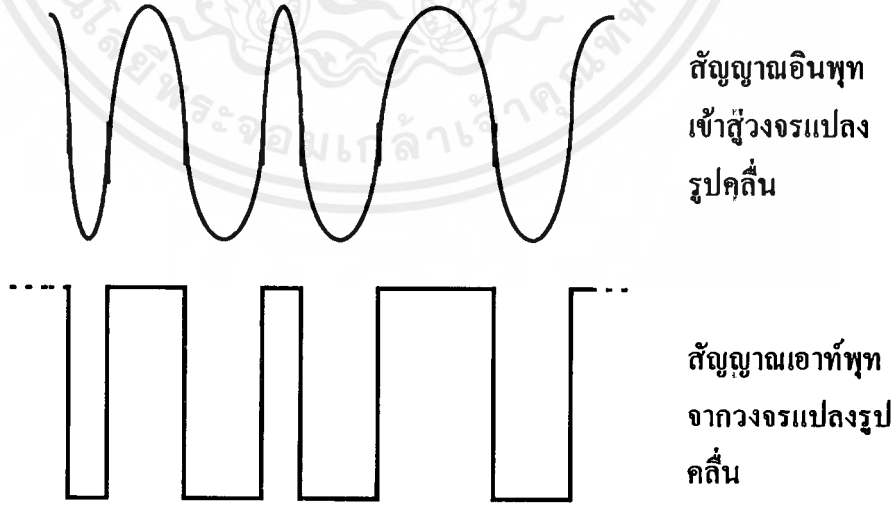


รูปที่ 3.2 แสดงการโฟกัสแสงที่แม่นยำ



รูปที่ 3.3 แสดงการบีบแสงโฟกัสผ่านช่องเก็บแสง

พื้นที่ของรหัสแท่งซึ่งถูกตรวจสอบนี้เรียกว่า สปอต (spot) สปอตควรจะประกอบด้วยความกว้างที่แคบที่สุดของรหัสแท่งที่ถูกสแกน สปอตสามารถอยู่ในแบบที่มีการเก็บแสงแบบกว้าง จากแสงที่โฟกัสแม่นยำดังรูปที่ 3.2 หรือโดยการปล่อยแสงแบบกว้างแล้วให้แสงถูกโฟกัสผ่านช่องเก็บแสง ดังรูปที่ 3.3



รูปที่ 3.4 รูปคลื่นสัญญาณอินพุทและเอาท์พุทของหัวอ่าน

แสงที่สะท้อนจากสปอต จะถูกบังคับทิศทางไปยังตัวรับแสง (Light Detecter) ซึ่งจะสร้างกระแสค่าหนึ่งไม่มากนักแต่เป็นสัดส่วนกับแสงที่สะท้อนมา วงจรขยายสัญญาณ (Amplifier) ในหัวอ่านจะขยายสัญญาณจากตัวรับแสงให้อยู่ในระดับใช้งาน แรงดันไฟฟ้าที่ได้จะเป็นสัดส่วนกับการสะท้อนแสงเพื่อจะแยกแท่งขาวและดำ แรงดันไฟฟ้าจะถูกแปลงเป็นรูปคลื่นดิจิตอล (Digital Waveform) ด้วยวงจรแปลงรูปคลื่น (Wave Shaper) ดังรูปที่ 3.4

ในการสแกนด้วยความเร็วคงที่ จะสามารถตรวจสอบความกว้างและแคบของแท่งขาวกับแท่งทึบด้วยการวัดค่าเวลา หรือสามารถเปลี่ยนจากไทม์โดเมน (Time Domain) เป็นสเปซโดเมน (Space Domain) ได้ง่ายและแม่นยำ แต่ถ้าการสแกนมีความเร่งหรือมีความเร็วไม่คงที่ การวัดค่าเวลาจากความกว้าง-แคบของรหัสแท่งขาวดำจะยากมากขึ้น (ความแม่นยำในการวัดน้อยลง) และจากการตรวจสอบแรงดันไฟฟ้าจริงจากหัวอ่าน ความแม่นยำและความถูกต้องในการวัดค่าจะขึ้นอยู่กับซอฟท์แวร์ในส่วนถอดรหัส (Decoder Part)

3.2 ชนิดของหัวอ่านรหัสแท่ง

หัวอ่านรหัสแท่งแบ่งตามลักษณะการใช้งานและการทำงานได้ดังนี้

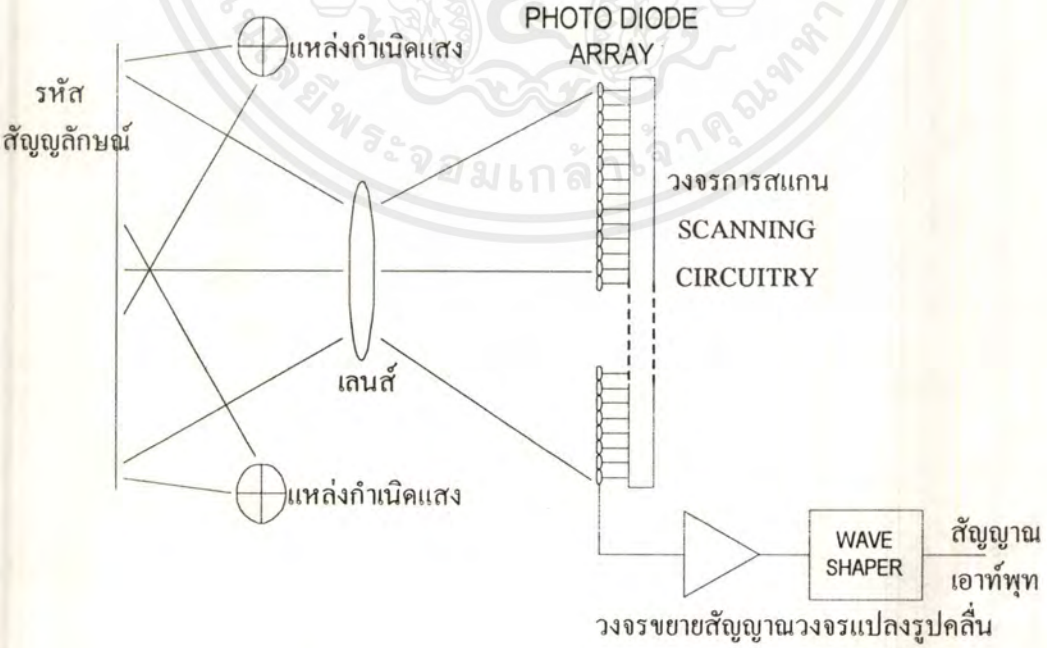
3.2.1 หัวอ่านแบบลำแสงกวาด (Moving Beam Scanner)

หัวอ่านแบบลำแสงกวาด ใช้แสงเลเซอร์ (Laser) หรือฮีเลียม-นีออน เป็นแหล่งกำเนิดแสง โดยกวาดลำแสงผ่านรหัสแท่งได้เอง (self scanning) ลักษณะการสแกนทางแนวนอน โดยมีกระจกสะท้อนแสงซึ่งหมุนได้มากกว่า 4 ทิศทาง เพื่อบันทึกการสะท้อนของลำแสงจากรหัสแท่งถูกสร้างโดยการหมุนกระจกภายในหัวอ่านและฉายแสงกวาดไปยังรหัสแท่ง มีทั้งแบบอยู่กับที่ (Fixed) และแบบมือถือ (Hand Held) ดังรูปที่ 3.5

รูปที่ 3.5 หัวอ่านแบบลำแสงกวาด

หัวอ่านแบบซีซีดี (CCD, Charged-Couple Device) ดังรูปที่ 3.6 ก็จัดเป็นหัวอ่านแบบนี้ แต่ใช้อุปกรณ์แบบพาสสีฟ (passive device) หลักการของมันก็คือ ฉายแสงคลุมทั้งรหัสแท่ง แล้วทำการแยกสัญญาณ (digitize) โดยอาศัยแผงลิเนียร์โฟโตไดโอด (Linear photo diode array) ส่วนการสแกนจริงๆ ทำโดยวงจรการสแกน (Scaning circuitry) จากไดโอดแต่ละตัวในแบบลำดับ (sequential) และอาศัยเทคโนโลยีของอุปกรณ์ Charged-Couple Device ดังรูปที่ 3.7

รูปที่ 3.6 หัวอ่านแบบซีซีดี



รูปที่ 3.7 วงจรการทำงานของหัวอ่านแบบซีซีดี

3.1.2 หัวอ่านแบบอยู่กับที่ลำแสงคงที่ (Fix Mount Fix Beam Scanner)

หัวอ่านแบบนี้จะอยู่กับที่ ลำแสงไม่เคลื่อนที่ การสแกนรหัสอาศัยการเคลื่อนตัวรหัสแท่งผ่านแหล่งกำเนิดแสงซึ่งอยู่กับที่ ตัวอย่างของหัวอ่านแบบนี้คือ Barcode Slot Scanner ดังรูปที่ 3.8

การประยุกต์ใช้งานหัวอ่านแบบลำแสงคงที่ในโรงงานอุตสาหกรรมทั่วไป คือติดรหัสแท่งไว้บนวัสดุที่เคลื่อนไปบนสายพาน ดังนั้นจึงมีโอกาสสแกนเพียงครั้งเดียว จึงต้องใช้การพิมพ์รหัสแท่งที่มีคุณภาพสูง

รูปที่ 3.8 หัวอ่านแบบ Slot Scanner

3.2.3 หัวอ่านแบบมือถือลำแสงคงที่ (Hand Held Fix Beam Scanner)

เป็นหัวอ่านแบบที่ใช้คนควบคุม การสแกนจะขึ้นอยู่กับผู้ใช้เป็นหลัก ส่วนประกอบภายในของหัวอ่านแบบนี้จะไม่มีส่วนกลไกใด ๆ ที่จะช่วยการสแกนให้เป็นแบบอัตโนมัติ ลำแสงที่ใช้จะอยู่กับที่ แบ่งเป็น 2 แบบ คือแบบสัมผัสรหัส (Contact) และไม่สัมผัสรหัส (Non Contact)

หัวอ่านแบบสัมผัสรหัสนั้น รอยสัมผัสระหว่างหัวอ่านกับตัวรหัสควรจะมีขนาดเล็กที่สุดเพื่อลดความเสียหายจากรอยขีดข่วนที่จะเกิดกับตัวรหัส ตัวอย่างหัวอ่านแบบสัมผัสรหัส คือ Wand หรือ Light Pen และ Contact Gun สำหรับหัวอ่านแบบไม่สัมผัสรหัส จะมีจุดโฟกัสอยู่ห่างจากปลายหัวอ่านออกมา จึงไม่จำเป็นต้องสัมผัสกับตัวรหัสแท่ง

รูปที่ 3.9 หัวอ่านแบบ Wand

ข้อผิดพลาดที่มักจะเกิดขึ้นในการใช้งานของหัวอ่านแบบมือถือลำแสงคงที่

- การสแกนช้าไปทำให้เกิดการกระตุก
- การหยุดสแกนก่อนจะสิ้นสุดรหัสจริงๆ (บริเวณขอบเผื่อด้านขวา)
- ไม่เริ่มการสแกนที่จุดเริ่มต้นของรหัสแท่ง (บริเวณขอบเผื่อด้านซ้าย)
- ต้องสแกนผ่านเลยแท่งทึบแท่งสุดท้าย

แม้ว่าจะมีข้อจำกัดในการใช้งาน แต่หัวอ่านชนิดนี้ก็เป็นที่นิยมใช้ เพราะมีราคาถูก

3.3 ส่วนถอดรหัสแท่ง (Barcode Decoder)

ส่วนถอดรหัสของเครื่องอ่านรหัสแท่ง จะทำหน้าที่วิเคราะห์สัญญาณดิจิตอล ที่ได้จากหัวอ่าน แปลงสัญญาณนั้นให้ตรงตามรูปแบบของรหัสแท่งด้วยขั้นตอนการถอดรหัส

ขั้นตอนการถอดรหัสมักจะใช้ซอฟท์แวร์ ที่ทำงานบนไมโครโปรเซสเซอร์ หรืออาจจะใช้ฮาร์ดแวร์แทนก็ได้ ใช้วิธีการใดก็ตามต้องมีขั้นตอนดังต่อไปนี้

1. แบ่งแยกแท่งขาวและแท่งทึบจากการสแกน
2. วัดความกว้างของแต่ละสัญญาณที่สแกนได้
3. แบ่งแยกระดับความกว้างของสัญญาณโดยอาศัยอัลกอริทึม (Algorithm) แบบต่างๆ เปลี่ยนความกว้างเป็นรหัสเลขฐานสอง
4. ถอดรหัสจากรหัสเลขฐานสอง โดยนำไปเปรียบเทียบกับค่าในตารางรูปแบบรหัสแท่งสำหรับแต่ละรหัสอักขระ
5. มีอัลกอริทึมบังคับทิศทางการสแกน ถ้ารหัสชนิดนั้นสแกนได้ทิศทางเดียว

6. ตรวจสอบเพิ่มเติมเพื่อยืนยันความถูกต้องของการสแกนรหัสแท่ง

- บริเวณขอบเผื่อที่ถูกต้อง (Quiet Zone)
- อักขระตรวจสอบ (Check Digit)
- ความเร็วของการสแกนจะต้องอยู่ในค่าจำกัดที่กำหนดไว้แล้ว

7. การส่งข้อมูลที่ถอดรหัสได้ที่มีผลต่อการถอดรหัสครั้งต่อไป

3.4 ชนิดของเครื่องอ่านรหัสแท่ง

เครื่องอ่านรหัสแท่งแบ่งตามหน้าที่การใช้งานได้ 2 แบบ ดังนี้

3.4.1. เครื่องอ่านรหัสแท่งแบบออนไลน์ (On line barcode readder)

เครื่องอ่านรหัสแท่งแบบนี้ จะมีการสื่อสารถ่ายเทข้อมูลกับเครื่องคอมพิวเตอร์ที่ใช้ประมวลผลอยู่ตลอดเวลา ตัวอย่างของเครื่องอ่านรหัสแบบนี้ ได้แก่ เครื่องอ่านรหัสแท่งในระบบ POS (Point of Sale) ที่ใช้ในธุรกิจค้าปลีก ดังรูปที่ 3.10

รูปที่ 3.10 เครื่องอ่านรหัสแท่งในระบบ POS

3.4.2. เครื่องอ่านรหัสแท่งแบบพอร์ทเทเบิล (Portable barcode readder)

เครื่องอ่านรหัสแท่งแบบพอร์ทเทเบิล ประกอบด้วย หน่วยประมวลผล หน่วยความจำในการเก็บข้อมูลรหัสแท่ง และส่วนแสดงผลรวมอยู่ภายในเครื่อง ใช้แบตเตอรี่เป็นแหล่งจ่ายไฟ ตัวเครื่องมีขนาดเล็ก พกพาสะดวก ดังรูปที่ 3.11 มักจะนำไปใช้ในการตรวจสอบรหัสแท่งที่อยู่กับที่ เช่น การตรวจสอบคลังสินค้า ข้อมูลที่ได้จะถูกเก็บเอาไว้ในเครื่องก่อน เมื่อต้องการนำข้อมูลไปประมวลผลจึงนำไปต่อเข้ากับเครื่องคอมพิวเตอร์

รูปที่ 3.11 เครื่องอ่านรหัสแท่งแบบพอร์ทเทเบิล

บทที่ 4
ไมโครคอนโทรลเลอร์ MCS-51

4.1 สมาชิกของไมโครคอนโทรลเลอร์ตระกูล MCS-51

ไมโครคอนโทรลเลอร์ "MCS-51" หมายถึง ชิปไมโครคอนโทรลเลอร์เบอร์ 8051 และเบอร์ที่สำคัญอื่นๆ ในตระกูล MCS-51 ด้วย เช่น เบอร์ 8052, 8031, 8032 หรือ 8751 เป็นต้น แต่ละเบอร์จะมีความสามารถพิเศษมากน้อยแตกต่างกันไป ดังแสดงให้เห็นในตาราง 4.1

คุณสมบัติของ MCS-51

คุณสมบัติที่สำคัญๆ ของชิปไมโครคอนโทรลเลอร์ตระกูล MCS-51 มีดังนี้

- ต้องการแหล่งจ่ายไฟ 5 โวลต์ เพียงชุดเดียว
- มีหน่วยความจำสำหรับเก็บโปรแกรมควบคุมการทำงาน อยู่ภายในชิปจำนวน 4 กิโลไบท์ (เบอร์ 8031,8032 ไม่มีหน่วยความจำส่วนนี้ เบอร์ 8052 มีหน่วยความจำ 8 กิโลไบท์ และเบอร์ 83C51FB จะมีหน่วยความจำ 16 กิโลไบท์)
- มีหน่วยความจำสำหรับเก็บข้อมูลทั่วไป (RAM) อยู่ภายในชิปจำนวน 128 ไบท์ (ใน 8031, 8051) หรือ 256 ไบท์ (ในเบอร์ 8032, 8052)
- สามารถใช้หน่วยความจำสำหรับโปรแกรม และข้อมูลที่อยู่ภายนอกชิปได้อย่างละ 64 กิโลไบท์ แยกจากกัน
- คำสั่งส่วนใหญ่ใช้เวลาทำงานเพียง 1 ไมโครวินาที เมื่อใช้คริสตอลความถี่ 12 เมกกะเฮิรตซ์
- มีพอร์ตที่สามารถรับหรือส่งข้อมูลได้ทั้ง 2 ทิศทาง จำนวน 4 พอร์ตๆ ละ 8 บิท หรือสามารถใช้งานเป็นพอร์ตขนาด 1 บิทแยกจากกัน ทำให้เสมือนมีพอร์ตขนาด 1 บิทใช้งาน รวมทั้งสิ้น 32 พอร์ต
- รับและส่งข้อมูลแบบอนุกรมได้ในตัวเอง โดยสามารถกำหนดอัตราเร็วในการรับและส่งข้อมูล (baud rate) ได้ตั้งแต่ 300 ถึง 375 กิโลบิทต่อวินาที
- จัดลำดับความสำคัญของสัญญาณอินเตอร์รัพท์ได้ 2 ระดับ
- มีรีจิสเตอร์สำหรับใช้งานเป็นไทม์เมอร์หรือเคาน์เตอร์ เพื่อนับจำนวนสัญญาณนาฬิกาภายในชิป หรือนับการเปลี่ยนสถานะของสัญญาณภายนอก ขนาด 16 บิท จำนวน 2 ตัว เพื่อใช้สำหรับนับจำนวนพัลส์ วัดความกว้างของพัลส์ หรือใช้วัดช่วงเวลา (ส่วนเบอร์ 8052 จะมี 3 ตัว)

- หน่วยความจำสำหรับเก็บข้อมูลภายในบางส่วน สามารถเข้าถึงข้อมูลได้ทั้งระดับไบท์และระดับบิท เพื่อให้การออกแบบโปรแกรมและการควบคุมระบบทำให้ง่ายขึ้น

- มีคำสั่งคูณและหารเลขขนาด 8 บิทในตัวเอง

- สามารถประมวลผลแบบบูลีน เพื่อใช้ในงานควบคุมโดยเฉพาะ

- ใช้โปรแกรมของไมโครคอนโทรลเลอร์ตระกูล MSC-48 (upwardly compatible) ได้ด้วย

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 เบอร์ที่จัดว่าเป็นเบอร์พื้นฐานในตระกูลนี้ คือเบอร์ 8051, 8031 และ 8751 มีจำนวนขาภายนอก 40 ขาเท่ากันใช้เวลาและสัญญาณในการปฏิบัติคำสั่งแต่ละคำสั่งเท่ากัน (มีไทม์มิ่งไดอะแกรมเหมือนกัน) ใช้แรงดันไฟฟ้าเท่ากัน สิ่งที่ต่างกันระหว่างเบอร์ทั้งสามคือขนาดหน่วยความจำสำหรับเก็บโปรแกรมภายในชิป (onchip program memory) มีไว้เพื่อตอบสนองความต้องการที่ไม่เหมือนกัน ดังจะกล่าวต่อไปนี้

เบอร์-8751 มีหน่วยความจำสำหรับเก็บโปรแกรมภายในชิปเป็น EPROM (Erasable Programable Read Only Memory) ขนาด 4 กิโลไบท์ ทำให้สามารถใช้รังสีอัลตราไวโอเลตในการลบโปรแกรมเก่าที่มีอยู่ และบรรจุโปรแกรมใหม่ลงไปได้ทันที เพื่อความสะดวกในการแก้ไขหรือปรับปรุงโปรแกรม ไมโครคอนโทรลเลอร์ MCS-51 เบอร์-8751 จะใช้งานเป็นการพัฒนาเบื้องต้น (prototypeing) ซึ่งจำเป็นต้องทดสอบโปรแกรมเพื่อหาข้อผิดพลาด (bugs) และแก้ไขเรียบร้อยก่อนทำการผลิตจริง การแก้ไขโดยการใช้รังสีอัลตราไวโอเลตและการบรรจุโปร แกรมที่แก้ไขใหม่สามารถทำได้ในจำนวนครั้งที่จำกัด ทั้งนี้เพราะหน่วยความจำที่เป็น EPROM เมื่อใช้ไปนานๆ จะเกิดเสื่อมสภาพ ทำให้ไม่สามารถบรรจุโปรแกรมเข้าไปได้

เบอร์-8051 หลังจากทดสอบโปรแกรมจนไม่พบข้อผิดพลาดแล้ว จะเป็นช่วงของการผลิตจริง ซึ่งต้องพิจารณาถึงต้นทุนเป็นอันดับแรก และในการผลิตจริงจะใช้ไมโครคอนโทรลเลอร์เบอร์ 8051 มีหน่วยความจำสำหรับเก็บโปรแกรมภายในเป็น ROM (Read Only Memory) ขนาด 4 กิโลไบท์แทน เพราะราคาต่ำกว่ามาก แต่มีข้อจำกัดตรงที่ไม่สามารถแก้ไขโปรแกรมที่ได้บรรจุไปแล้วไม่ว่าจะด้วยวิธีใดก็ตาม

เบอร์-8031 เบอร์นี้ไม่มีหน่วยความจำสำหรับเก็บโปรแกรมภายในชิป แต่สามารถใช้หน่วยความจำ เพื่อเก็บโปรแกรมที่อยู่ภายนอกได้มากถึง 64 กิโลไบท์ อาจจะใช้เป็น ROM, PROM, EPROM ตามความต้องการของผู้ผลิต เบอร์-8031 นี้มีไว้ใช้ในกรณีที่โปรแกรมมีขนาดเล็กกว่า 4 กิโลไบท์ หรือมากกว่า 4 กิโลไบท์มาก (เบอร์ 8751 และ 8051 จะใช้โปรแกรมจากหน่วยความจำภายนอกได้เอง เมื่อโปรแกรมมีความยาวเกิน 4 กิโลไบท์ หรืออาจบังคับให้ไมโครคอนโทรลเลอร์ทั้งสองเบอร์ใช้โปรแกรมจากหน่วยความจำภายนอกเพียงอย่างเดียว ด้วยการต่อขา 31 ลงกราวด์ ทำให้มีคุณสมบัติเหมือนเบอร์ 8031 ที่ไม่มีหน่วยความจำสำหรับเก็บโปรแกรมภายในชิป)

ตารางที่ 4.1 แสดงความแตกต่างของสมาชิกไมโครคอนโทรลเลอร์

Device	ROMless Version	EPROM Version	ROM Bytes	RAM Bytes	8-Bit I/O Ports	16-Bit Timer/ Counters	Interrupt Sources/ Vectors
8051	8031	-	4K	128	4	2	6/5
8051AH	8031AH	8751H 8751BH	4K	128	4	2	6/5
8052AH	8032AH	8752BH	8K	256	4	3	8/6
80C51BH	80C31BH	87C51	4K	128	4	2	6/5
83C51FA	80C51FA	87C51FA	8K	256	4	3	14/7
83C51FB	80C51FA	87C51FB	16K	256	4	3	14/7
83C51GA	80C51GA	87C51GA	4K	128	4	2	8/7
83C152JA	50C152JA	-	8K	256	5	2	19/11
-	80C152JB	-	-	256	7	2	19/11
83C152JC	80C152JC	-	8K	256	5	2	19/11
-	80C152JD	-	-	256	7	2	19/11
83C451	80C451	-	4K	128	7	2	6/5
83C452	80C452	87C452P	8K	256	5	2	9/8

4.2 โครงสร้างของไมโครคอนโทรลเลอร์ตระกูล MCS-51

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 มีสมาชิกในตระกูลหลายเบอร์ด้วยกัน แต่ละเบอร์จะมีคุณสมบัติพิเศษบางอย่างแตกต่างกัน เช่น มีหน่วยความจำภายในสำหรับเก็บโปรแกรม และข้อมูลภายในชิปเพิ่มขึ้น มีวงจรเปลี่ยนค่าสัญญาณอนาล็อกเป็นดิจิตอล (Analog to Digital , A/D) ในตัว สามารถรับอินเตอร์รัพท์ได้หลายชนิด ทำกระบวนการ DMA (Direct Memory Access) ได้ ในตัวมีรีจิสเตอร์สำหรับใช้เป็นไทมเมอร์ หรือเคาน์เตอร์เพิ่มมากขึ้น คุณสมบัติพิเศษที่แตกต่างกันของไมโครคอนโทรลเลอร์แต่ละเบอร์ในตระกูลนี้แสดงในตาราง 4.1 ที่ผ่านมา

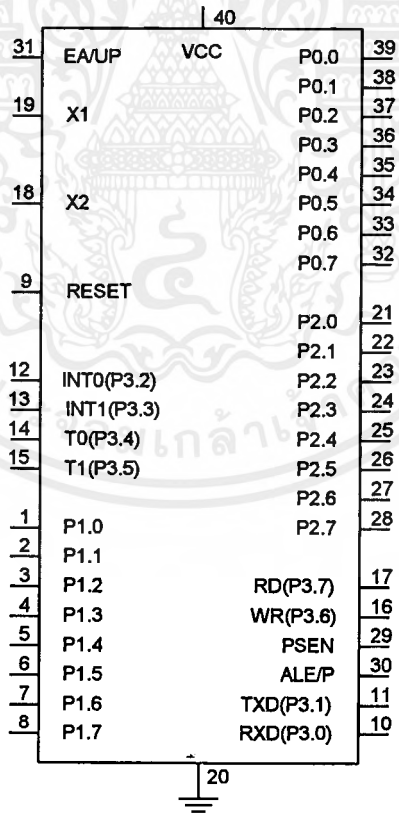
ไมโครคอนโทรลเลอร์เบอร์ที่นับได้ว่าเป็นเบอร์พื้นฐานสำหรับตระกูล MCS-51 นี้ ได้แก่ เบอร์ 8051, 8031, 8751 โดยเบอร์ 8051 จัดเป็นสมาชิกตัวแรกในตระกูล มีหน่วยความจำสำหรับเก็บโปรแกรมภายในชิปเป็น ROM ขนาด 4 กิโลไบท์ และหน่วยความจำสำหรับเก็บข้อมูลทั่วไปภายใน MCS-51 (RAM) เองจำนวน 128 ไบท์ มีพอร์ตขนาด 8 บิท 4 พอร์ต มีรีจิสเตอร์สำหรับใช้เป็นไทม์เมอร์หรือเคาน์เตอร์ขนาด 16 บิท รวม 2 ตัว รับสัญญาณอินเตอร์รัพท์ จากภายนอกได้ 2 ชนิด สามารถรับและส่งข้อมูลแบบอนุกรมผ่านทางพอร์ตสื่อสารข้อมูลแบบอนุกรม มีวงจรออสซิลเลเตอร์เพื่อสร้างสัญญาณนาฬิกาควบคุมการทำงานในตัวเอง

ส่วนเบอร์-8751 จะมีคุณสมบัติเหมือน เบอร์-8051 ทุกอย่าง ต่างกันเพียงชนิดของหน่วยความจำสำหรับเก็บโปรแกรมภายในชิป เบอร์-8751 จะเป็นอีพรอม (EPROM) แทนที่จะเป็นรอม (ROM) ส่วนเบอร์-8031 จะเหมือนกับเบอร์-8051 ต่างกันเพียงเบอร์-8031 ไม่มีหน่วยความจำสำหรับเก็บโปรแกรมภายในชิปเท่านั้น

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 ทุกเบอร์ใช้แรงดันไฟเพียง 5 โวลต์ ในการทำงาน ส่วนกระแสไฟฟ้าที่ใช้จะแตกต่างกันไป ตามชนิดของเทคโนโลยีที่ใช้ในการผลิต เบอร์ของไมโครคอนโทรลเลอร์ที่มีตัวอักษร C อยู่ตรงกลางเบอร์ เช่น 80C31, 80C51 เป็นเบอร์ของชิปที่ผลิตโดยอาศัยเทคโนโลยี CHMOS ใช้พลังงานในการทำงานน้อยกว่า และสามารถควบคุมการใช้พลังงานของตัวชิปได้จากโปรแกรม เพื่อการประหยัดพลังงานในระบบ

4.3 ตำแหน่งขาของ MCS-51

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 ทุกเบอร์มีตำแหน่งขาพื้นฐานที่เหมือนกัน ดังแสดงในรูปที่ 4.1



รูปที่ 4.1 แสดงตำแหน่งขาของชิปไมโครคอนโทรลเลอร์ MCS-51

หน้าที่การใช้งานแต่ละขาของชิปไมโครคอนโทรลเลอร์ในตระกูล มีดังนี้

- ขา Vss (ขา 20) สำหรับต่อลงกราวด์
- ขา Vcc (ขา 40) สำหรับต่อแหล่งจ่ายแรงดันกระแสตรงขนาด 5 โวลต์ (DC 5 V)
- ขาพอร์ต 0 (ขา 32-39) มี 8 ขา ใช้เป็นขาสำหรับพอร์ต 0 ขนาด 8 บิท (P0.0-P0.7) แบบ Open Drain Bidirectional พอร์ตนี้ สามารถใช้งานเป็นอินพุทเอาต์พุทพอร์ตทั่วไปได้โดยหากใช้งานเป็นอินพุทพอร์ตต้องโหลดค่า 1 ไปยังแต่ละบิทของพอร์ตนี้ เพื่อบังคับให้ขาอยู่ในสถานะถูกปล่อยลอย (มีสถานะ high impedance) นอกจากใช้งานเป็นอินพุทเอาต์พุทพอร์ตแล้ว พอร์ต 0 ยังใช้ในการติดต่อหน่วยความจำสำหรับเก็บโปรแกรม และข้อมูลภายนอกชิปด้วยโดยส่งแอดเดรสบัสไบท์ต่ำ (A0-A7) และมัลติเพลกซ์กับการรับส่งข้อมูล (D0-D7) จากหน่วยความจำภายนอก ในระหว่างการเขียนหรืออ่านข้อมูล โดยมีวงจรพูลอัพ (pull up) ภายใน
- ขาพอร์ต 1 (ขา 1-8) มี 8 ขาใช้เป็นขาสำหรับพอร์ต 1 (P1.0-P1.7) สามารถใช้เป็นอินพุทหรือเอาต์พุทพอร์ตทั่วไปได้ หากต้องการใช้งานเป็นอินพุทพอร์ตต้องโหลดค่า 1 ไปยังแต่ละบิทของพอร์ตนี้ เพื่อให้มีสถานะ high impedance โดยมีวงจรพูลอัพ (pull up) ภายใน
- ขาพอร์ต 2 (ขา 21-28) มี 8 ขาใช้เป็นขาสำหรับพอร์ต 2 (P2.0-P2.7) ขนาด 8 บิท แบบ Open Drain Bidirectional พอร์ตนี้สามารถใช้งานเป็นอินพุทเอาต์พุทพอร์ตทั่วไปได้ โดยหากใช้งานเป็นอินพุทพอร์ตต้องโหลดค่า 1 ไปยังแต่ละบิทของพอร์ตนี้ เพื่อบังคับให้ขาอยู่ในสถานะ highimpedance นอกจากจะใช้งานเป็นอินพุทเอาต์พุทพอร์ตทั่วไปแล้ว พอร์ต 2 ยังใช้ในการติดต่อหน่วยความจำสำหรับเก็บโปรแกรมและข้อมูลภายนอกด้วย โดยใช้สำหรับส่งค่าแอดเดรสบัสไบท์สูง (A8-A15) และมีวงจรพูลอัพ (pull up) ภายใน
- ขาพอร์ต 3 (ขา 10-17) มี 8 ขาใช้เป็นขาสำหรับพอร์ต 3 (P3.0-P3.7) สามารถใช้งานเป็นอินพุทเอาต์พุทพอร์ตทั่วไปได้ หากต้องการใช้งานเป็นอินพุทพอร์ตต้องโหลดค่า 1 ไปยังแต่ละบิทของพอร์ตนี้ เพื่อให้มีสถานะ high impedance โดยใช้วงจรพูลอัพ (Pull up) ภายใน นอกจากนี้ยังใช้งานในหน้าที่พิเศษต่าง ๆ อีกหลายอย่างดังนี้

ขา P3.0 ใช้รับข้อมูลจากภายนอกแบบอนุกรม

ขา P3.1 ใช้ส่งข้อมูลออกไปภายนอกแบบอนุกรม

ขา P3.2 ใช้เป็นอินพุทเพื่อรับสัญญาณอินเตอร์รัพท์ชนิดที่ 0

ขา P3.3 ใช้เป็นอินพุทเพื่อรับสัญญาณอินเตอร์รัพท์ชนิดที่ 1

ขา P3.4 สัญญาณอินพุทให้เคาน์เตอร์ของไทม์เมอร์ 0

ขา P3.5 สัญญาณอินพุทให้เคาน์เตอร์ของไทม์เมอร์ 1

ขา P3.6 สัญญาณควบคุมการเขียนข้อมูลไปยังหน่วยความจำสำหรับเก็บข้อมูลนอกชิป

ขา P3.7 สัญญาณควบคุมการอ่านข้อมูลจากหน่วยความจำสำหรับเก็บข้อมูลภายนอกชิป

การใช้งานพอร์ต 3 ในหน้าที่พิเศษดังกล่าวนี้ จะต้องโหลดค่า 1 ไปยังแต่ละบิทที่ต้องการใช้ก่อนทุกครั้ง

- ขา RST (ขา 9) ใช้สำหรับการรีเซตวงจรทุกอย่างภายในชิป เพื่อเริ่มต้นการทำงานใหม่ การรีเซตใช้เมื่อเริ่มจ่ายพลังงานหรือเมื่อโปรแกรมเกิดทำงานผิดพลาด เมื่อต้องการรีเซตชิป MCS-51 ขานี้ต้องมีสถานะ 1 เป็นเวลาอย่างน้อย 2 แมชชีนไซเกิล ระหว่างที่ออสซิลเลเตอร์ยังทำงานอยู่ โดยต้องต่อตัวต้านทานค่า 8.2 กิโลโอห์ม เพื่อจะทำหน้าที่พูลดาวน์ (รักษาค่าแรงดันไฟฟ้าให้มีสถานะเป็นกราวด์) และเพื่อให้ตัวชิปรีเซตเอง เมื่อเริ่มจ่ายพลังงานให้ตัวเก็บประจุขนาด 10 mF คร่อมระหว่างขา RST กับ Vcc ดังแสดงในรูปที่ 4.2

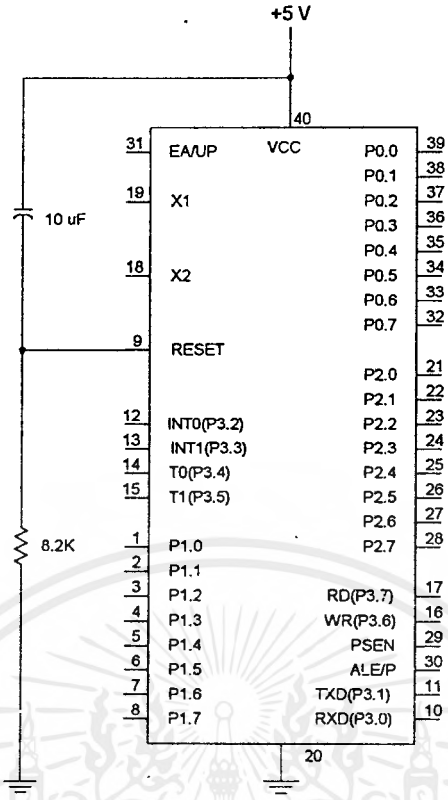
- ขา ALE/PROG (ขา 30) เป็นขาสำหรับใช้ส่งสัญญาณออกไปภายนอก เพื่อควบคุมแลตช์แอดเดรสไบท์ต่ำ (address latch enable) จากพอร์ต 0 ในระหว่างการติดต่อหน่วยความจำสำหรับเก็บโปรแกรมหรือข้อมูลภายนอกโดยปกติ เมื่อไม่มีการติดต่อหน่วยความจำภายนอกขานี้จะลดลงครึ่งหนึ่งในระหว่างที่ติดต่ออยู่กับหน่วยความจำสำหรับเก็บข้อมูลที่อยู่ภายนอกชิป นอกจากนี้ขา ALE ยังใช้สำหรับควบคุมการเขียนโปรแกรมลงไปใน EPROM สำหรับ MCS-51 ที่มีหน่วยความจำสำหรับเก็บโปรแกรมภายในชิปเป็น EPROM

- ขา PSEN (ขา 29) ใช้ส่งสัญญาณสโตรบ เพื่ออ่านคำสั่งจากโปรแกรมที่เก็บไว้ในหน่วยความจำภายนอกชิป (program strobeenable) เมื่อชิปทำงานด้วยโปรแกรมจากภายนอกขานี้จะส่งสัญญาณสโตรบสองครั้งในแต่ละแมชชีนไซเกิล แต่ในช่วงการเขียน หรืออ่านข้อมูลกับหน่วยความจำภายนอกหรือเมื่อใช้โปรแกรมจากหน่วยความจำสำหรับเก็บโปรแกรมภายในชิปจะไม่มีสัญญาณออกมาจากขานี้

- ขา EA/Vpp (ขา 31) เป็นขาสำหรับใช้เลือกให้ MCS-51 ทำงานจากโปรแกรมที่อยู่ภายในหรือภายนอกชิป โดยถ้าหากขานี้มีสถานะเป็น 0 ก็จะหมายถึงการให้ใช้โปรแกรมจากหน่วยความจำที่เก็บโปรแกรมภายนอก หากขานี้มีสถานะเป็น 1 หมายถึงบังคับให้ MCS-51 ใช้โปรแกรมจากหน่วยความจำสำหรับเก็บโปรแกรมภายในชิป และสำหรับ MCS-51 ที่มีหน่วยความจำสำหรับเก็บโปรแกรมภายในชิป สามารถเลือกให้ทำงานได้ ทั้งจากโปรแกรมที่เก็บในหน่วยความจำภายในชิปหรือจากโปรแกรมที่เก็บไว้ในหน่วยความจำภายนอกชิป ด้วยการต่อขา EA กับไฟเลี้ยงหรือกราวด์ตามลำดับ ส่วนใน MCS-51 ที่ไม่มีหน่วยความจำสำหรับเก็บโปรแกรมภายในชิป ให้ต่อขานี้ลงกราวด์เสมอ

- ขา XTAL (ขา 19) ใช้ต่อคริสตอลจากภายนอกโดยใช้เป็นขาอินพุทเข้าสู่วงจรออสซิลเลเตอร์

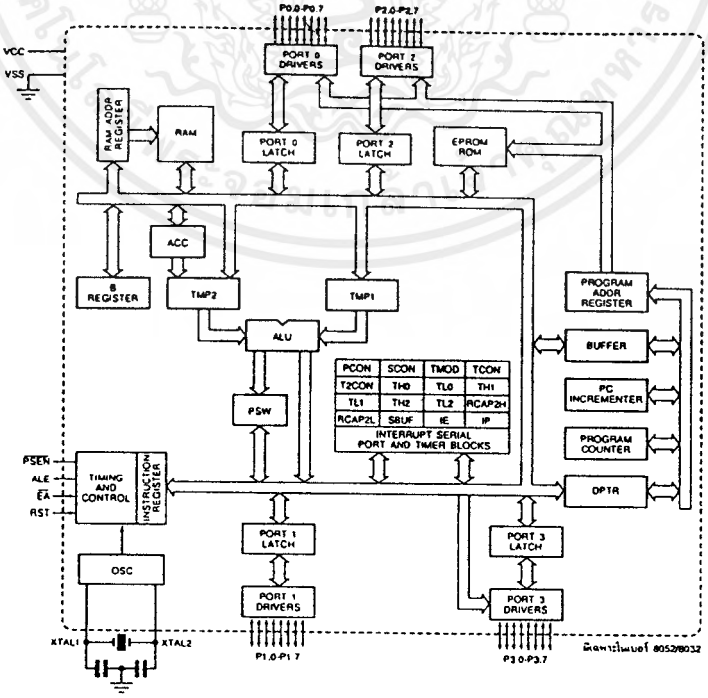
- ขา XTAL (ขา 18) ใช้ต่อคริสตอลภายนอกโดยเป็นขาเอาต์พุทออกจากวงจรออสซิลเลเตอร์



รูปที่ 4.2 แสดงวงจรสำหรับรีเซตชิปไมโครคอนโทรลเลอร์ MCS-51

4.4 โครงสร้างภายในของ MCS-51

โครงสร้างภายในของไมโครคอนโทรลเลอร์ตระกูล MCS-51 ดังแสดงในรูปที่ 4.3



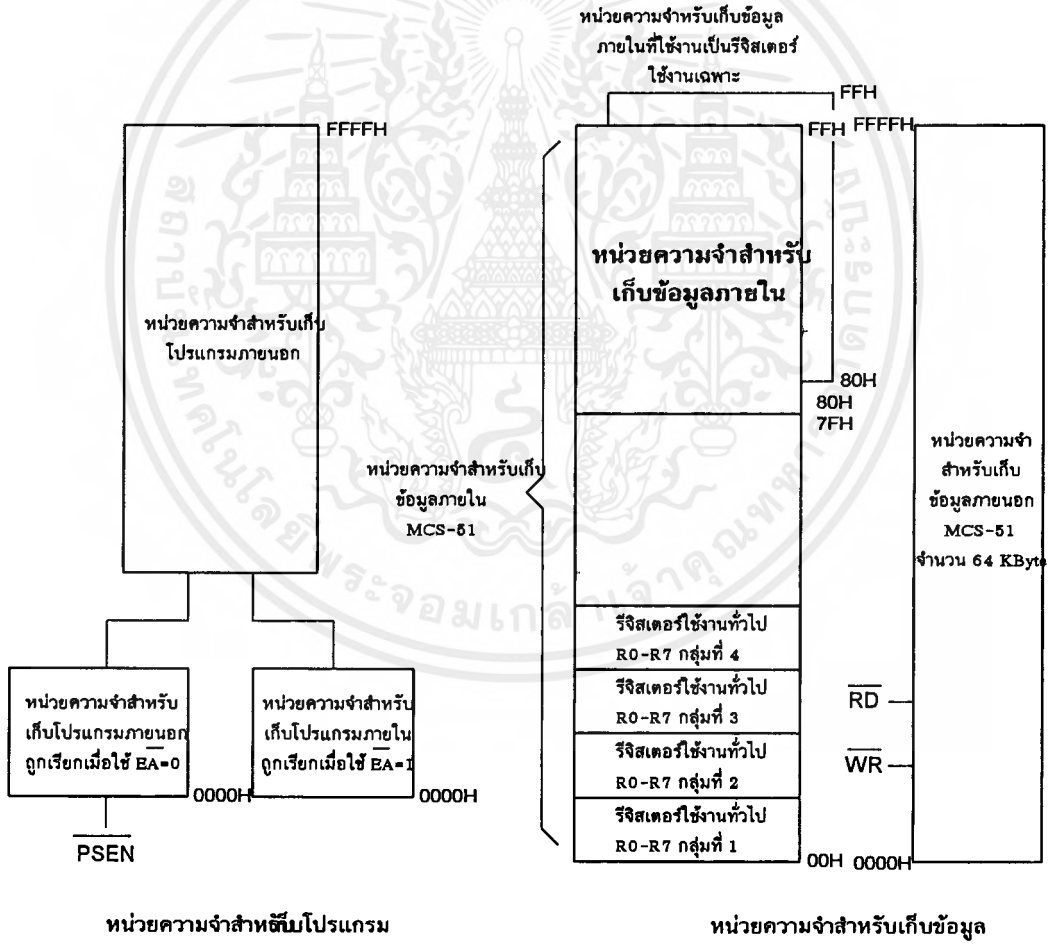
รูปที่ 4.3 แสดงโครงสร้างภายในของชิปไมโครคอนโทรลเลอร์ MCS-51

4.4.1 โครงสร้างหน่วยความจำภายใน MCS-51

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 ทุกเบอร์ จะแบ่งหน่วยความจำออกเป็น 2 ส่วนคือ

- หน่วยความจำสำหรับเก็บโปรแกรม (program memory)
- หน่วยความจำสำหรับเก็บข้อมูล (data memory)

หน่วยความจำสำหรับเก็บโปรแกรม จะใช้เก็บโปรแกรมควบคุมการทำงานของชิป MCS-51 บางเบอร์ มีหน่วยความจำส่วนนี้อยู่ภายในชิป แต่บางเบอร์ไม่มี ทำให้ต้องเก็บโปรแกรมไว้ในหน่วยความจำภายนอกทั้งหมด ส่วนหน่วยความจำส่วนที่สอง คือ หน่วยความจำสำหรับเก็บข้อมูล ซึ่งใช้สำหรับเก็บข้อมูลระหว่างการทำงาน MCS-51 ทุกเบอร์จะมีหน่วยความจำส่วนนี้อยู่ภายในชิปจำนวนหนึ่งแต่จะมีจำนวนมากหรือน้อยเท่าใดขึ้นกับเบอร์ของชิปแสดงในตาราง 4.1 โครงสร้างหน่วยความจำทั้งหมดของ MCS-51 มีดังแสดงในรูปที่ 4.4



รูปที่ 4.4 แสดงโครงสร้างหน่วยความจำทั้งหมดของ MCS-51

- **หน่วยความจำสำหรับเก็บโปรแกรม**

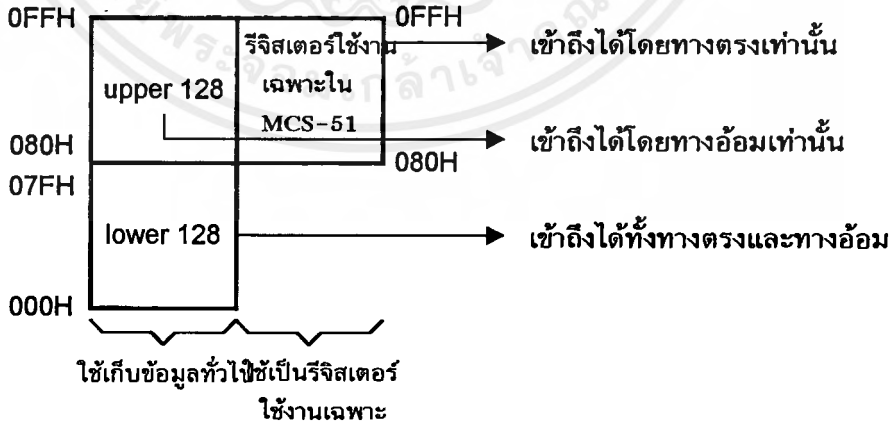
หน่วยความจำสำหรับเก็บโปรแกรมในชิบ MCS-51 จะแบ่งออกเป็น 2 ส่วน คือ หน่วยความจำสำหรับเก็บโปรแกรมภายในชิป (internal program memory) และหน่วยความจำสำหรับเก็บโปรแกรมภายนอกชิป (external program memory) ขนาดของหน่วยความจำสำหรับเก็บโปรแกรมภายในชิปมีได้ตั้งแต่ 0, 4, 8, 16 กิโลไบท์ ขึ้นอยู่กับเบอร์ของชิป

- **หน่วยความจำสำหรับเก็บข้อมูล**

หน่วยความจำสำหรับเก็บข้อมูลของ MCS-51 จะแบ่งออกเป็น 2 ส่วน คือ หน่วยความจำสำหรับเก็บข้อมูลภายในชิป และหน่วยความจำสำหรับเก็บข้อมูลภายนอกชิป หน่วยความจำสำหรับเก็บข้อมูลภายในชิปของ MCS-51 ยังแบ่งออกเป็น 3 ส่วนย่อยดังนี้

- ส่วนที่ใช้เก็บข้อมูลทั่วไป (internal ram) บริเวณ 128 ไบท์ล่าง (lower 128)
- ส่วนที่ใช้เก็บข้อมูลทั่วไป (internal ram) บริเวณ 128 ไบท์บน (upper 128)
- ส่วนที่ใช้เป็นรีจิสเตอร์ใช้งานเฉพาะ (special function register)

หน่วยความจำส่วนที่ใช้เก็บข้อมูลทั่วไปภายในชิป เป็นหน่วยความจำสำหรับเก็บข้อ มูลที่มีอยู่ภายใน MCS-51 หน่วยความจำส่วนนี้มีไว้สำหรับเก็บข้อมูลในขณะทำงาน ส่วนหน่วยความจำสำหรับเก็บข้อมูลภายในชิปที่ใช้เป็นรีจิสเตอร์ ใช้งานเฉพาะเป็นหน่วยความจำสำหรับเก็บข้อมูลภายใน MCS-51 ซึ่งถูกกำหนดให้เป็นรีจิสเตอร์ใช้งานเฉพาะ เพื่อควบคุมการทำงานและบอกสถานะของซีพียู ซึ่งแผนภาพแสดงหน่วยความจำสำหรับเก็บข้อมูลภายในชิปทั้งสองบริเวณ ดังแสดงในรูปที่ 4.5



รูปที่ 4.5 แผนภาพแสดงหน่วยความจำสำหรับเก็บข้อมูลภายในชิป MCS-51

4.4.2 ไทม์เมอร์/เคาน์เตอร์

ใน MSC-51 มีรีจิสเตอร์ใช้งานเฉพาะที่สามารถนับจำนวนสัญญาณนาฬิกา หรือแมชชีนไซเกิลของวงจรออสซิลเลเตอร์ภายใน(ทำงานเป็นตัวไทม์เมอร์) หรือนับจำนวนครั้งของการเปลี่ยนสถานะของสัญญาณภายนอก (นับจำนวนพัลส์ภายนอก) ที่ขา T0, T1 ของพอร์ต 3 (ทำงานเป็นเคาน์เตอร์) รีจิสเตอร์ที่ใช้เป็นไทม์เมอร์หรือเคาน์เตอร์มีขนาด 16 บิทจำนวน 2 ตัว คือ รีจิสเตอร์ไทม์เมอร์ 0 และรีจิสเตอร์ไทม์เมอร์ 1 ตามลำดับ (ในเบอร์ 8052 มีรีจิสเตอร์ไทม์เมอร์ 2 เพิ่มให้อีก 1 ตัว) เมื่อต้องการใช้ไทม์เมอร์ 0 หรือไทม์เมอร์ 1 จะต้องโหลดค่าที่ต้องการนับไปไว้ในรีจิสเตอร์ไทม์เมอร์ 0 หรือรีจิสเตอร์ไทม์เมอร์ 1 และเมื่อนับครบจำนวนที่ตั้งไว้จะมีสัญญาณอินเตอร์รัพท์เพื่อบอกให้ซีพียูทราบ

การควบคุมการทำงานของไทม์เมอร์หรือเคาน์เตอร์ สามารถควบคุมได้จากวงจรภายนอก (ควบคุมด้วยสัญญาณที่ขา INT0, INT1) หรือควบคุมจากคำสั่งโปรแกรม ดังนั้นรีจิสเตอร์ที่ใช้เป็นไทม์เมอร์ใน MCS-51 จะสามารถวัดช่วงห่างของเวลา วัดความกว้างของพัลส์ หรือนับจำนวนครั้งของเหตุการณ์ที่เกิดขึ้นภายนอกที่เปลี่ยนให้อยู่ในรูปของสัญญาณไฟฟ้าแล้ว รวมทั้งใช้ในการกำเนิดสัญญาณอินเตอร์รัพท์ที่มีคาบเวลาแน่นอนได้

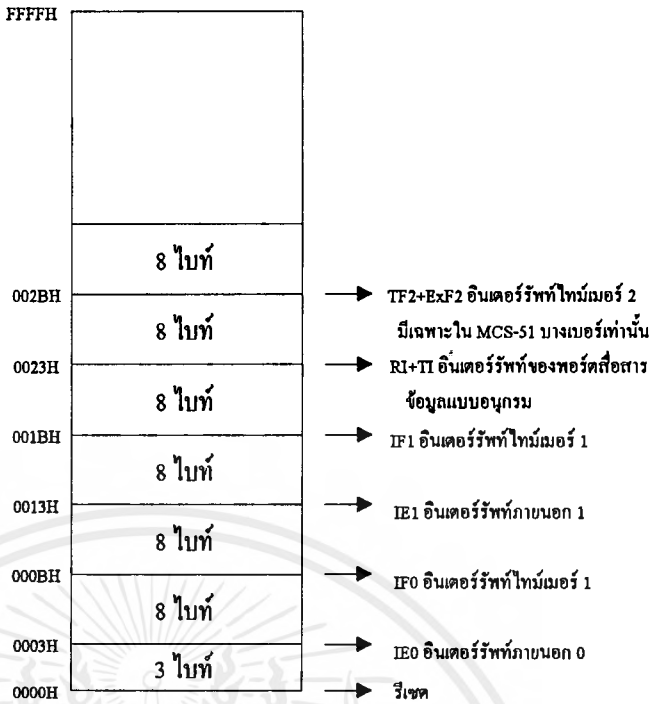
4.4.3 พอร์ตสื่อสารข้อมูลแบบอนุกรม MCS-51

สามารถรับและส่งข้อมูลแบบอนุกรมได้ โดยไม่ต้องพึ่งอุปกรณ์ภายนอกอื่นๆ แต่อย่างใดในด้านอัตราเร็วของการรับส่งข้อมูลก็จะสามารถกำหนดค่าได้ตามความต้องการของผู้ใช้ โดยสามารถเลือกอัตราเร็วในการรับส่งข้อมูล (baud rate) มาตรฐานได้ตั้งแต่ 300, 1.2K, 2.4K, 4.8K, 9.6K, 19.2K, 375K ตามมาตรฐานของ UART นอกจากนี้สามารถกำหนดการทำงานที่แตกต่างกันถึง 4 รูปแบบ ตามความเหมาะสมในแต่ละงาน

4.4.4 โครงสร้างการอินเตอร์รัพท์ MCS-51

สามารถรับสัญญาณอินเตอร์รัพท์ได้ถึง 5 ชนิด โดยจะเป็นสัญญาณอินเตอร์รัพท์ ที่เกิดจากภายนอก 2 ชนิดและที่เกิดจากภายในชิปอีก 3 ชนิด เมื่อมีสัญญาณอินเตอร์รัพท์เกิดขึ้น MCS-51จะละการทำงานโปรแกรมที่กำลังทำอยู่และข้ามไปทำงานโปรแกรมบริการอินเตอร์รัพท์ (interrupt service routine) ที่อยู่ในหน่วยความจำตำแหน่งต่าง ๆ ขึ้นอยู่กับชนิดของสัญญาณอินเตอร์รัพท์ดังแสดงในรูปที่ 4.6

เราสามารถเลือกให้ซีพียูใน MCS-51 ถูกอินเตอร์รัพท์โดยสัญญาณอินเตอร์รัพท์ที่เกิดขึ้นได้ โดยการกำหนดค่าในรีจิสเตอร์ใช้งานเฉพาะ IE นอกจากนี้ยังสามารถควบคุมลำดับความสำคัญในการตอบสนองต่อสัญญาณอินเตอร์รัพท์ของ MCS-51 ได้ด้วยรีจิสเตอร์ใช้งานเฉพาะ IP



รูปที่ 4.6 แสดงตำแหน่งหน่วยความจำของโปรแกรมบริการอินเตอร์รัพท์แต่ละชนิดใน MCS-51

- **IE (interrupt Enable-Register)** เข้าถึงข้อมูลได้ในระดับบิท รายละเอียดมีดังแสดงใน รูปที่ 4.7

IE7	IE6	IE5	IE4	IE3	IE2	IE1	IE0
EA	-	ET2	ES	ET1	EX1	ET0	EX0

รูปที่ 4.7 รีจิสเตอร์ใช้งานเฉพาะ IE

บิท	ชื่อบิท	ความหมาย
IE7	EA	ใช้ควบคุมการตอบสนองต่อสัญญาณอินเตอร์รัพท์ทั้งหมด 0 : MCS-51 จะไม่ตอบสนองต่อสัญญาณอินเตอร์รัพท์ใด ๆ ทั้งสิ้น 1 : อินเตอร์รัพท์แต่ละชนิดจะถูกควบคุมการตอบสนองอย่างอิสระจากบิทในรีจีสเตอร์นี้
IE6	-	ไม่ถูกกำหนดการใช้งาน
IE5	ET2	ควบคุมการตอบสนองต่อสัญญาณอินเตอร์รัพท์ของ Timmer 2 เมื่อ Overflow
IE4	ES	ควบคุมการตอบสนองต่อสัญญาณอินเตอร์รัพท์ของพอร์ตสื่อสารอนุกรม
IE3	ET1	ควบคุมการตอบสนองต่อสัญญาณอินเตอร์รัพท์ของ Timmer 1 เมื่อ Overflow
IE2	EX1	ควบคุมการตอบสนองต่อสัญญาณอินเตอร์รัพท์ภายนอกชนิด 1
IE1	ET0	ควบคุมการตอบสนองต่อสัญญาณอินเตอร์รัพท์ของ Timmer 0 เมื่อ Overflow
IE0	EX0	ควบคุมการตอบสนองต่อสัญญาณอินเตอร์รัพท์ภายนอกชนิด 0

หมายเหตุ ถ้าบิทที่ควบคุมการตอบสนองต่ออินเตอร์รัพท์แต่ละบิทมีค่าเป็น 1 หมายถึงอนุญาตให้ MCS-51 ตอบสนองต่ออินเตอร์รัพท์ได้ หากมีค่าเป็น 0 หมายถึงไม่ให้ MCS-51 ตอบสนองต่อสัญญาณอินเตอร์รัพท์ที่เกิดขึ้น

- **IP (interruptPriorityRegister)** เข้าถึงข้อมูลได้ในระดับบิท รายละเอียดมีดังแสดงในรูปที่ 4.8

IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0
-	-	PT2	PS	PT1	PX1	PT0	PX0

รูปที่ 4.8 รีจิสเตอร์ใช้งานเฉพาะ IP

บิท	ชื่อบิท	ความหมาย
IP7	-	ไม่ถูกกำหนดการใช้งาน
IP6	-	ไม่ถูกกำหนดการใช้งาน
IP5	PT2	กำหนดลำดับความสำคัญการตอบสนองต่อสัญญาณอินเตอร์รัพท์ Timmer 2
IP4	PS	กำหนดลำดับความสำคัญการตอบสนองต่อสัญญาณอินเตอร์รัพท์ ของพอร์ตสื่อ-สารอนุกรม
IP3	PT1	กำหนดลำดับความสำคัญการตอบสนองต่อสัญญาณอินเตอร์รัพท์ Timmer 1
IP2	PX1	กำหนดลำดับความสำคัญการตอบสนองต่อสัญญาณอินเตอร์รัพท์ภายนอกชนิด 1
IP1	PT0	กำหนดลำดับความสำคัญการตอบสนองต่อสัญญาณอินเตอร์รัพท์ Timmer 0
IP0	PX0	กำหนดลำดับความสำคัญการตอบสนองต่อสัญญาณอินเตอร์รัพท์ภายนอกชนิด 0

4.5 วิธีการเข้าถึงข้อมูล

วิธีการเข้าถึงข้อมูลในคำสั่งของ MCS-51 มี 6 วิธีคือ

- การเข้าถึงข้อมูลโดยตรง (direct addressing)
- การเข้าถึงข้อมูลโดยทางอ้อม (indirect addressin)
- การเข้าถึงข้อมูลในรีจิสเตอร์ใช้งานทั่วไป (register instructions)
- การเข้าถึงข้อมูลในรีจิสเตอร์เฉพาะของคำสั่ง (register-specific instructions)
- การเข้าถึงข้อมูลที่กำหนดเองโดยตรง (immediate constants)
- การเข้าถึงข้อมูลที่มีตัวชี้อ้างอิง (indexed addressing)

4.5.1 การเข้าถึงข้อมูลโดยตรง

วิธีนี้จะระบุค่าตำแหน่ง หน่วยความจำที่เก็บข้อมูลโดยตรงในคำสั่ง ข้อมูลที่นำมาประมวลผลโดยวิธีนี้จะเป็นค่าของข้อมูลในหน่วยความจำสำหรับเก็บข้อมูล ที่ใช้เก็บข้อมูลทั่วไปเฉพาะในบริเวณ 128 ไบท์ล่าง และหน่วยความจำสำหรับเก็บข้อมูลที่ใช้เป็นรีจิสเตอร์ ใช้งานเฉพาะเท่านั้น และเนื่องจากหน่วยความจำสำหรับเก็บข้อมูลที่ใช้เก็บข้อมูลทั่วไปในบริเวณ 128 ไบท์ล่างกับหน่วยความจำสำหรับเก็บข้อมูลที่ใช้เป็นรีจิสเตอร์ใช้งานเฉพาะมีขนาดรวมกันทั้งสิ้น 256 ไบท์ ดังนั้นค่าตำแหน่งหน่วยความจำที่ใช้ต้องเป็นเลขไบนารีขนาด 8 บิท เท่านั้น

4.5.2 การเข้าถึงข้อมูลโดยทางอ้อม

ค่าตำแหน่งหน่วยความจำที่ต้องการติดต่อ จะเก็บไว้ในรีจิสเตอร์เฉพาะของคำสั่ง ดังนั้นวิธีนี้จึงถือเป็นวิธีการเข้าถึงข้อมูลโดยทางอ้อม คือแทนที่ผู้เขียนโปรแกรม จะระบุค่าตำแหน่งข้อมูลโดยตรง วิธีนี้จะใช้ค่าที่เก็บไว้ในรีจิสเตอร์ที่ระบุในรหัสคำสั่ง ชี้ไปยังตำแหน่งของหน่วยความจำแทน หน่วยความจำที่สามารถใช้วิธีการเข้าถึงข้อมูลแบบนี้ จะเป็นหน่วยความจำสำหรับเก็บข้อมูลที่ใช้เก็บข้อมูลทั่วไปในบริเวณ 128 ไบท์ล่าง และ 128 ไบท์บน รวมทั้งหน่วยความจำสำหรับเก็บข้อมูลที่อยู่ภายนอกชิป

4.5.3 การเข้าถึงข้อมูลในรีจิสเตอร์ใช้งานทั่วไป

วิธีนี้เป็นการเข้าถึงข้อมูลที่อยู่ในรีจิสเตอร์ R0-R7 ของรีจิสเตอร์ใช้งานทั่วไป แต่ละกลุ่ม รีจิสเตอร์ใช้งานทั่วไปแต่ละกลุ่มทั้ง 4 กลุ่ม คือบริเวณหนึ่งของหน่วยความจำสำหรับเก็บข้อมูลที่ใช้เก็บข้อมูลในบริเวณ 128 ไบท์ล่าง หรือตำแหน่ง 32 ไบท์ล่างสุดของหน่วยความจำสำหรับเก็บข้อมูลที่ใช้เก็บข้อมูลทั่วไป ดังนั้นหากผู้เขียนโปรแกรมต้องการอ่านหรือเขียนข้อมูลในรีจิสเตอร์ทั้ง 32 ตัว แต่ละตัวมีตำแหน่งในหน่วยความจำที่แน่นอน ก็สามารถชี้ตำแหน่งของหน่วยความจำที่ตรงกับรีจิสเตอร์แต่ละตัว ด้วยวิธีการเข้าถึงข้อมูลโดยตรงหรือโดยทางอ้อม

4.5.4 การเข้าถึงข้อมูลในรีจิสเตอร์เฉพาะของคำสั่ง

คำสั่งบางคำสั่งของ MCS-51 จะระบุไว้แล้วว่าต้องปฏิบัติการกับข้อมูล ในรีจิสเตอร์ตัวใด เช่น รีจิสเตอร์ A , รีจิสเตอร์ DPTR , รีจิสเตอร์ SP ในรหัสคำสั่งของคำสั่งที่ใช้วิธีการเข้าถึงข้อมูลประเภทนี้ MCS-51 จะทราบเองว่าต้องทำงานกับรีจิสเตอร์ตัวใด โดยไม่จำเป็นต้องระบุตำแหน่งรีจิสเตอร์ที่ใช้โดยคำสั่งเองเลย

4.5.5 การเข้าถึงข้อมูลที่กำหนดเองโดยตรง

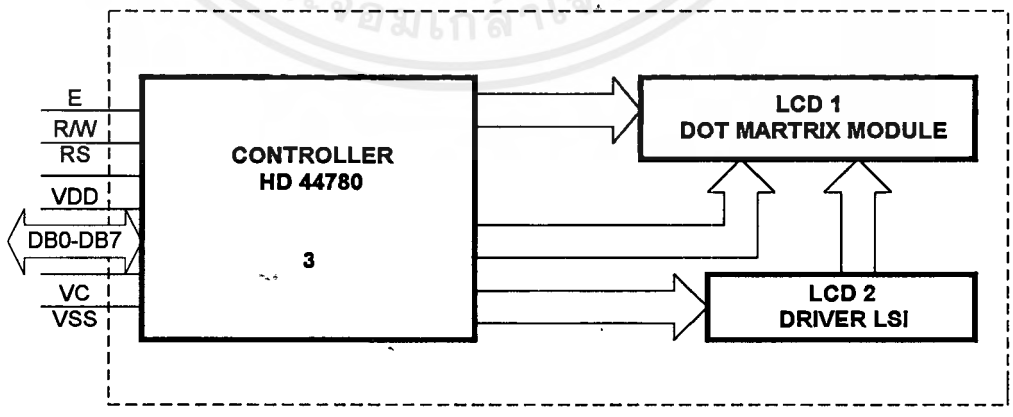
เป็นการกำหนดค่าข้อมูลที่จะนำไปประมวลผลโดยตรง ข้อมูลที่นำมาประมวลผลในคำสั่งจะอยู่ตามหลังรหัสคำสั่งโดยการใช้เครื่องหมาย "#" ระบุหน้าข้อมูลที่ต้องการ

4.5.6 การเข้าถึงข้อมูลโดยใช้ตัวชี้อ้างอิง

ข้อมูลที่ใช้วิธีการอ้างแบบนี้ เป็นข้อมูลที่อยู่ในหน่วยความจำสำหรับเก็บโปรแกรมเท่านั้น นั่นแสดงว่าเราสามารถอ่านข้อมูลนี้ออกมาได้แต่ไม่สามารถนำข้อมูลไปเก็บโดยวิธีนี้ได้จุดประสงค์ของการอ้างข้อมูลแบบนี้มีไว้ เพื่อใช้ในการเปิดหาค่าข้อมูล ในหน่วยความจำสำหรับเก็บโปรแกรม ซึ่งเป็นหน่วยความจำชนิดถาวร (ROM) ข้อมูลในส่วนนี้จะไม่สูญหายไปแม้ไม่มีพลังงาน ในการทำงานของคำสั่งที่ใช้การเข้าถึงข้อมูลวิธีนี้จะใช้ค่าของรีจิสเตอร์ใช้งานเฉพาะ DPTR หรือ PC มารวมกับค่าในรีจิสเตอร์ A เพื่อชี้ไปยังตำแหน่งของหน่วยความจำสำหรับเก็บโปรแกรมที่เก็บข้อมูลไว้ ดังนั้นค่าในรีจิสเตอร์ใช้งานเฉพาะ DPTR หรือ PC จะต้องมีค่าเท่ากับ ตำแหน่งต้นของหน่วยความจำสำหรับเก็บโปรแกรมในส่วนที่เก็บข้อมูล โดยใช้ค่าของรีจิสเตอร์ A เป็นตัวระบุว่าข้อมูลอยู่ห่างจากตำแหน่งเริ่มต้นในรีจิสเตอร์ใช้งานเฉพาะ DPTR หรือ PC เท่าใด จุดประสงค์ของวิธีการอ้างข้อมูลแบบนี้คือใช้ในการเปิดหาค่าข้อมูลในตารางซึ่งอยู่ในหน่วยความจำสำหรับเก็บโปรแกรมเรียงต่อกันไป

4.6 การประยุกต์ใช้ส่วนแสดงผลชนิด LCD module กับ MCS-51

ปัจจุบัน LCD ที่มีขายในท้องตลาดส่วนใหญ่จะประกอบเป็นโมดูล เพื่อให้สะดวกในการใช้งานโดยจะมีส่วนประกอบทั่วไปดังในรูปที่ 4.9



รูปที่ 4.9 แสดงโครงสร้างทั่วไปของ LCD module

LCD module ที่กล่าวต่อไปนี้ จะกล่าวถึงเฉพาะ Character LCD module ซึ่งจะเรียกย่อๆ ว่า LCM โดยมีส่วนประกอบที่สำคัญดังนี้

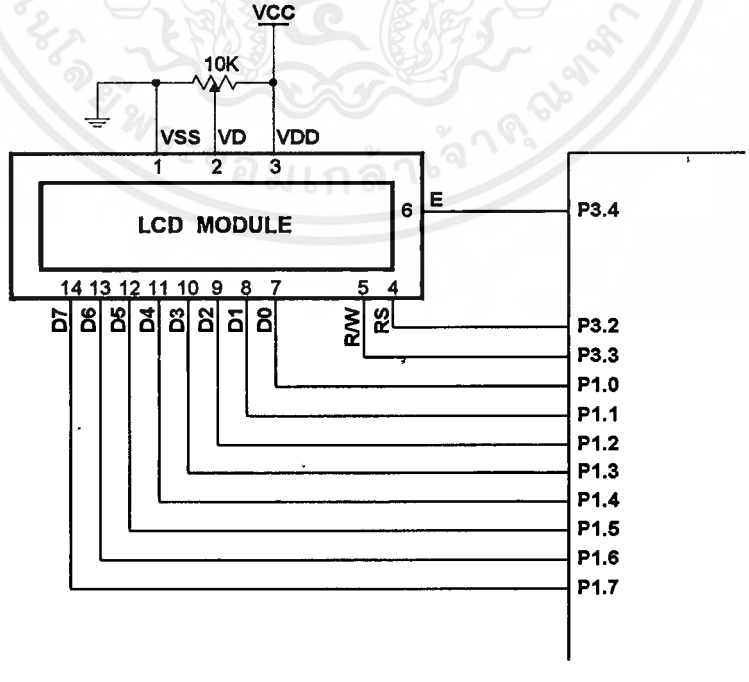
1. Dot Matrix LCD : เป็นส่วนที่ทำหน้าที่แสดงผล ใช้หลักการหักเหของแสงผ่านผลึก โดยจะประกอบไปด้วยจุด (pixel) จำนวนมากที่สามารถบังคับให้ติดหรือดับได้ทุกจุด

2. Driver : เป็นวงจรที่ใช้ขับ LCD ส่วนใหญ่จะใช้ชิปเบอร์ HD44110H

3. Controller : เป็นส่วนที่ใช้ควบคุมการทำงานทั้งหมดของ LCD module โดยจะรับข้อมูลจากภายนอกมาจัดการให้ LCD แสดงผลในรูปแบบต่างๆ ส่วนใหญ่จะใช้ชิปเปอร์เบอร์ HD44780 ซึ่งมีใช้งานแบบ character LCD module

การใช้ LCD module ผู้ใช้เพียงแต่ศึกษา และทำความเข้าใจในส่วนคอนโทรลเลอร์ของ LCM เท่านั้น เพราะส่วนนี้เป็นส่วนที่รับข้อมูลที่ต้องการแสดงผลจากวงจรภายนอก และควบคุมการทำงานทั้งหมดของ LCM โดยจะขอกล่าวถึงเฉพาะชิปที่ใช้เป็นคอนโทรลเลอร์เบอร์ HD44780 เท่านั้น ส่วนชิปคอนโทรลเลอร์เบอร์อื่นส่วนใหญ่จะมีการใช้งานที่คล้ายกับเบอร์นี้

ชิปคอนโทรลเลอร์เบอร์ HD44780 เป็นชิปของบริษัท HITACHI สามารถต่อใช้งานเพื่อควบคุม LCM กับชิปไมโครคอนโทรลเลอร์หรือไมโครโปรเซสเซอร์ได้ทั้งแบบ 4 bit 2 operation หรือ แบบ 8 bit 1 operation ดังนั้นชิปเบอร์นี้สามารถอินเตอร์เฟสกับไมโครโปรเซสเซอร์ได้ ทั้งแบบ 4 บิท และ 8 บิท เนื่องจากไมโครคอนโทรลเลอร์ตระกูล MCS-51 มีดาต้าบัสขนาด 8 บิท ดังนั้นเราจะกล่าวถึงเฉพาะการติดต่อในแบบ 8 bit 1 operation เท่านั้น ตัวอย่างวงจรการอินเตอร์เฟส MCS-51 กับ LCM ดังแสดงในรูปที่ 4.10



รูปที่ 4.10 ตัวอย่างการอินเตอร์เฟส MCS-51 กับ LCM

จากในรูปที่ 4.10 จะเห็นว่า LCM ติดต่อกับ MCS-51 โดย

- ใช้ขา P1.0 - P1.7 เป็นดาต้าบัส (DB0-DB7) ในการติดต่อ
- ใช้ขา P3.2 เป็น สัญญาณ RS
- ใช้ขา P3.3 เป็น สัญญาณ R/W
- ใช้ขา P3.4 เป็น สัญญาณ EN (E)

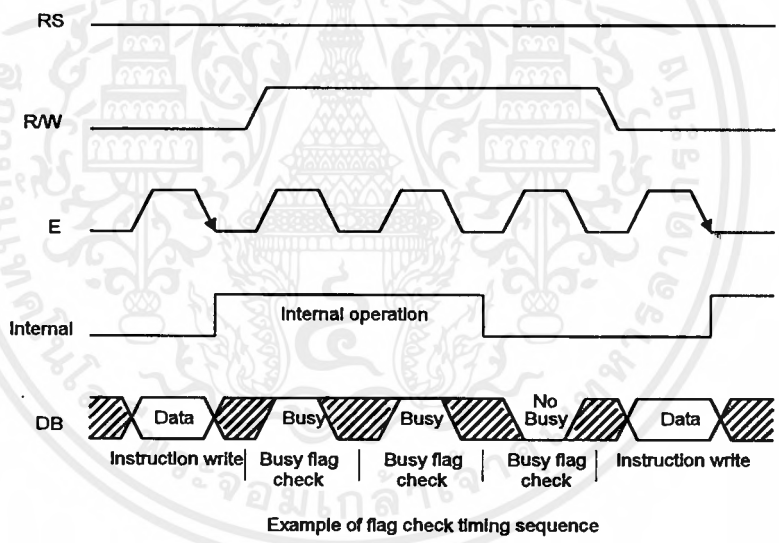
การทำความเข้าใจการใช้งาน LCM จำเป็นต้องทราบรายละเอียดดังต่อไปนี้เสียก่อน

1. LCM มีหลายขนาด แต่ทุกขนาด (ที่เป็นเฉพาะ character LCD module) จะมีคำสั่งในการควบคุมเหมือนกัน แตกต่างกันเพียงขนาดของหน่วยความจำในการแสดงผล หรือ DDRAM (Data Display RAM) เท่านั้น

2. แผนผังเวลา (timing diagram) ในการติดต่อ LCM

3. คำสั่งในการควบคุม LCM

แผนผังเวลาในการติดต่อ LCM ดังแสดงในรูปที่ 4.11



รูปที่ 4.11 แสดงแผนผังเวลาในการติดต่อกับ LCM

จากในรูปที่ 4.11 แสดงเฉพาะในช่วงที่สัญญาณ RS เป็น 0 เท่านั้น ส่วนในช่วงที่สัญญาณ RS เป็น 1 จะมีแผนผังเวลาเหมือนกัน รายละเอียดของแต่ละสัญญาณมีดังนี้

1. RS : เนื่องจากในชิปคอนโทรลเลอร์มีรีจิสเตอร์อยู่ 2 ประเภท คือ Command register หรือ instruction register และ data register โดยที่รีจิสเตอร์ทั้งสองจะถูกเลือก โดยสัญญาณ RS ดังนี้

สัญญาณ RS = 0 หมายถึงเลือกใช้ data register

สัญญาณ RS = 1 หมายถึงเลือกใช้ instruction register

2. R/W (Read/Write) : เป็นสัญญาณที่ใช้เลือกว่าจะเขียนหรืออ่านข้อมูลจาก LCM โดย

สัญญาณ R/W = 0 หมายถึงต้องการอ่านข้อมูลจาก LCM

สัญญาณ R/W = 1 หมายถึงต้องการเขียนข้อมูลไปยัง LCM

3. E (Enable) : มีรายละเอียดดังแสดงในตารางที่ 4.2

จากแผนผังเวลาในการตรวจสอบ busy flag และจากตารางจะเห็นว่า ในการเขียนรหัสคำสั่ง (instruction code) ทุกครั้ง

- RS และ RW ต้องมีค่าเป็น 0 และส่งข้อมูลไปในขณะที่สัญญาณ E เปลี่ยนจาก 1 เป็น 0 ในการเขียนข้อมูลทุกครั้ง

- RS = 1 และ RW = 0 และส่งข้อมูลไปขณะที่สัญญาณ E เปลี่ยนจาก 1 เป็น 0 ในการอ่าน busy flag และ address counter ทุกครั้ง

ตารางที่ 4.2 แสดงการทำงานของสัญญาณ E

RS	R/W	E	OPERATION
0	0		write instruction code
0	1		read busy flag and address counter
1	0		write data
1	1		read data

- RS = 0 และ RW = 1 และรับข้อมูลเข้ามาในขณะที่สัญญาณ E เป็น 1 ในการอ่านข้อมูลทุกครั้ง

- RS = 1 และ RW = 1 และรับข้อมูลเข้ามาในขณะที่สัญญาณ E เป็น 1

จากแผนผังเวลา (ในรูป 3.11) จะเห็นว่า DB0-DB7 มีสถานะเป็น high impedance เมื่อสัญญาณ E มีสถานะเป็น 0 ดังนั้นในการใช้งานจริงเมื่อเราเลิกติดต่อกับ LCM ควรจัดการส่งสัญญาณ E ให้มีค่าเป็น 0 เพื่อให้ P1.0 - P1.7 ของ MCS-51 มีสถานะเป็น high impedance ด้วย ทั้งนี้เพื่อเราจะได้ใช้งาน P1.0-P1.7 อย่างอื่นได้ด้วย และเนื่องจากเวลาในการทำงานคำสั่งต่างๆ ของ HD44780 ไม่เท่ากัน เราจึงควรที่จะตรวจสอบสัญญาณ busy flag ทุกๆ ครั้งก่อนที่จะทำการเขียนข้อมูลใดๆ ลงไป ทั้งนี้เพื่อป้องกันการเขียนข้อมูลทับนั่นเอง

รายละเอียดของคำสั่ง HD44780

1. CLEAR DISPLAY

Clear display

	RS	R/W	DB7						DB0
Code	0	0	0	0	0	0	0	0	1

คำสั่งนี้จะเป็นการเขียนช่องว่างหรือ SPACE (ASCII 20H) เข้าไปใน DD RAM ทั้งหมด และทำการ set DD RAM ADDRESSER เป็นศูนย์ ตัว cursor จะกลับไปอยู่ตำแหน่งบนสุดซ้ายมือของจอภาพ SET I/D = 1,S ไม่มีการเปลี่ยน

2. RETURN HOME

Return home

	RS	R/W	DB7						DB0
Code	0	0	0	0	0	0	0	0	*

* No effect

คำสั่งนี้จะทำการ set DD RAM ADDRESSER เป็นศูนย์ ตัว cursor จะกลับไปอยู่ตำแหน่งบนสุด ซ้ายมือของจอภาพ ข้อมูลในจอภาพไม่เปลี่ยน

3. ENTRY MODE SET

Entry mode set

	RS	R/W	DB7						DB0
Code	0	0	0	0	0	0	0	I/D	S

- BIT I/D : โดยจะเป็นตัวกำหนดให้ว่าเมื่อเขียนหรืออ่านข้อมูลแล้วจะทำให้ DD RAM ADDRESS เพิ่มขึ้นหนึ่งหรือลดลงหนึ่งโดย 1 = เพิ่ม, 0 = ลด

- BIT S : เป็นตัวกำหนดการแสดงผลโดยถ้า S = 1 จะเป็นการใส่ข้อมูล แล้วตัว cursor อยู่กับที่ข้อมูลจะถูกดันไปทางซ้าย ถ้า S = 0 ข้อมูลจะอยู่กับที่ ตัว cursor จะถูกดันไปทางขวามือ

4. DISPLAY ON/OFF CONTROL

Display on/off control

	RS	R/W	DB7						DB0
Code	0	0	0	0	0	0	1	D	C	B

- BIT D : เป็น BIT ให้เปิดปิดหน้าจอภาพโดยถ้า D = 1 จะ ON, D = 0 จะ OFF

- BIT C : จะให้แสดง cursor ถ้า C = 1 และไม่ต้องการแสดง cursor ถ้า C = 0 โดยตัว cursor จะอยู่ line ที่ 8 ในแบบ 5x7 DOT และจะอยู่ line ที่ 11 ในแบบ 5x10 DOT

- BIT B : เป็น bit set การกระพริบของ cursor โดย B = 1 กระพริบ B = 0 ไม่มีการกระพริบ มีระยะเวลาการกระพริบประมาณ 379.2 ms

5. CURSOR OR DISPLAY SHIFT

Cursor or display shift

	RS	R/W	DB7							DB0
Code	0	0	0	0	0	1	S/L	R/L	*	*

* No effect

เป็นคำสั่งกำหนดให้ตำแหน่ง cursor หรือข้อมูลไปเกิดทางซ้ายหรือขวา โดยไม่ต้องใช้คำสั่งเขียนหรืออ่านโดย

S/C	R/L	
0	0	ทำการย้าย CURSOR ไปจากตำแหน่งเดิมไปซ้ายมือ 1 ตำแหน่ง
0	1	ทำการย้าย CURSOR ไปจากตำแหน่งเดิมไปขวามือ 1 ตำแหน่ง
1	0	เป็นการดันตัวอักษรที่เกิดไปทางซ้าย
1	1	เป็นการดันตัวอักษรที่เกิดไปทางขวา

6. FUNCTION SET

Function set

	RS	RW	DB7							DB0
Code	0	0	0	0	1	DL	N	F	*	*

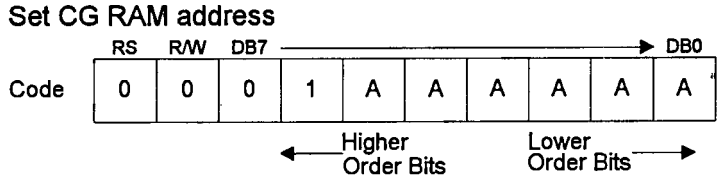
* No effect

- BIT DL : เป็นการ set การติดต่อว่าจะให้เป็นแบบ 8 bit หรือ 4 bit โดยถ้าต้องการติดต่อ 4 bit DL = 0 และ 8 bit DL = 1

- N : เป็นการ set บรรทัดการแสดงผล N = 0 แสดง 1 บรรทัด, N = 1 แสดง 2 บรรทัด ในกรณีมากกว่า 2 บรรทัดก็ให้ set N =1

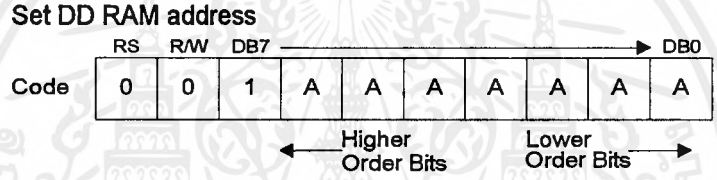
- F : เป็นการ set ขนาด DOT การแสดงผล โดย F = 0 เป็นแบบ 5x7 และ F = 1 เป็นแบบ 5 x10

7. SET CG RAM ADDRESS



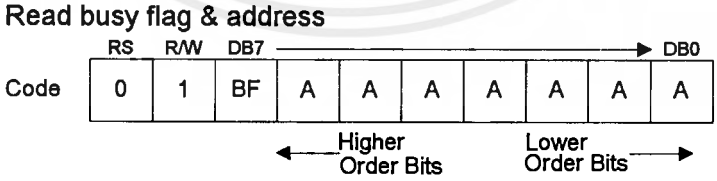
ใน HD44780 นั้นจะมีหน่วยความจำอยู่ 2 ชุด คือ DISPLAY DATA RAM (DD RAM) จำนวน 80x8 bit และ CHARACTER GENERATOR ROM CG RAM จำนวน 512 bit และ 7200 bit คำสั่งนี้จะเป็นการ set address ใน CG RAM โดยต้องทำการ set address ก่อนเขียนหรืออ่านข้อมูลจาก CG RAM ด้วย

8. SET DD RAM ADDRESS



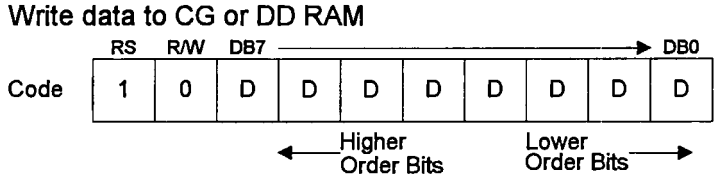
เป็นคำสั่ง set ค่า address ใน DD RAM ในการเขียนหรืออ่านค่าจาก DD RAM (DD RAM คือ ส่วนที่จะแสดงผลหน้าจอ LCD) โดยจำนวน address ที่จะเกิดขึ้นบนจอ LCD จะอยู่กับการ set ค่า N ด้วย N = 0 (1 บรรทัด) address จะอยู่ที่ 00H - 4FH และถ้า N = 1 (2 บรรทัด) address จะอยู่ที่ 0HH - 27H สำหรับบรรทัดที่ 1 และ 40H - 67H สำหรับบรรทัดที่ 2

9. READ BUSY FLAG AND ADDRESS



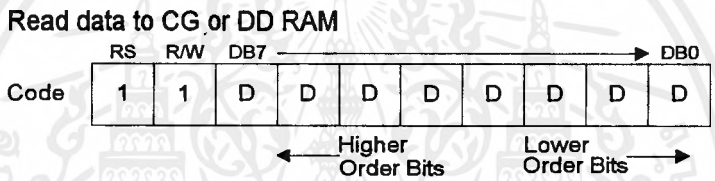
เป็นคำสั่งอ่านค่า busy flag ซึ่งจะเป็นตัวบอกว่าตัว HD44780 นี้อยู่ในขบวนการทำงานภายในอยู่หรืออยู่ในสภาพพร้อมรับข้อมูล โดย BF = 1 อยู่ในกระบวนการ ภายในไม่พร้อมจะรับข้อมูลหรือคำสั่ง และถ้า BF = 0 พร้อมจะรับข้อมูลหรือคำสั่งได้ และนอกจากนี้ยังเป็นคำสั่งอ่านข้อมูล address ของ CG RAM หรือ DD RAM ด้วย

10. WRITE DATA TO CG หรือ DD RAM



เป็นคำสั่งเขียนข้อมูลเข้าไปใน CG RAM หรือ DD RAM โดยเมื่อเขียนข้อมูลและ address จะเพิ่มหรือลดโดยอัตโนมัติตามคำสั่งที่ set ใน ENTRY MODE ข้อกำหนดที่จะรู้ว่าเป็นการเขียนข้อมูลของ CG RAM หรือ DD RAM ทำได้โดยการ set address ของ CG RAM หรือ DD RAM ขึ้นมาก่อนจะเขียนข้อมูล

11. READ DATA FROM CG OR DD RAM



เป็นคำสั่งอ่านค่าข้อมูลจาก CG RAM หรือ DD RAM โดยก่อนอ่านค่าควรจะใช้คำสั่ง set address ก่อน เพื่อให้รู้ว่าข้อมูลที่อ่านได้นั้นเป็น DD หรือ CG RAM จากตารางการทำงานจะเห็นว่าการใช้งาน LCD MODULE นั้นง่าย เพียงแต่เราส่งคำสั่งเริ่มแรกและ set ความต้องการขนาดตัวอักษร, cursor หลังจากนั้นเราก็สามารถเขียนตัวอักษรเข้าไปใน DD RAM ตามตารางตัวอักษรที่ให้มานั้นก็จะเกิดอักษรในจอภาพ LCD เรายังสามารถกำหนดตำแหน่งตัวอักษรที่จะให้เกิดบนจอได้โดยการ set DD RAM ตามตารางที่ให้มา

เราสามารถเขียนข้อมูลได้โดยกำหนด address ของ CG RAM ก่อน โดยเขียนได้ 64 ตำแหน่ง bit 5 - bit 0 และเมื่อกำหนด address แล้วก็จะทำการเขียนข้อมูลลงใน CG RAM โดยเป็นลักษณะ bit ต่อ bit บนจอ 1 ตัวอักษรคือ 5x7 DOT นั้นจะใช้ข้อมูล bit 4 ถึง bit 0 ต่อ 1 byte เท่านั้น 1 ตัวอักษรจะใช้ข้อมูล 8 byte และเมื่อเขียนลงใน CG RAM แล้ว เวลาเราจะใช้งานก็ให้เขียนข้อมูลลงใน DD RAM คือข้อมูลตำแหน่งในตารางที่ตำแหน่ง 00H - 07H การใช้งาน LCD module นั้นที่สำคัญคือ ต้องเข้าใจในตัว controller ของ LCD module นั้น โดย controller ทุกๆ บริษัทจะมีการทำงานที่เหมือนๆ กันเป็นส่วนใหญ่

บทที่ 5

การสื่อสารข้อมูลและเครือข่ายข้อมูล

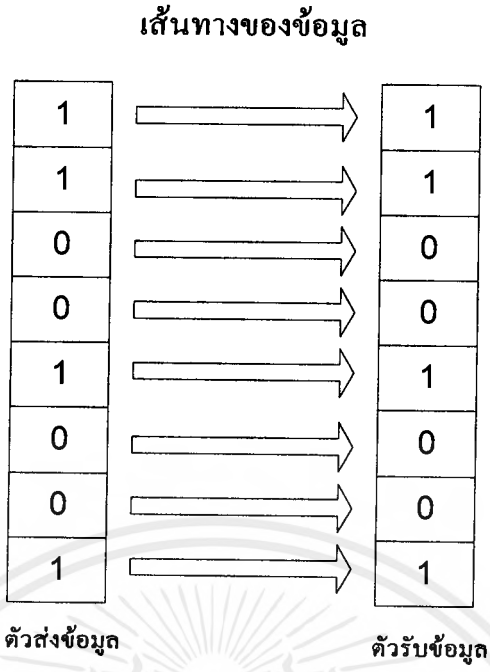
โครงสร้างระบบบรหัสแท่งแบบเครือข่ายนี้ ได้ออกแบบให้มีลักษณะเป็นเครือข่าย จึงต้องมีส่วนของการเชื่อมโยง และสื่อสารข้อมูลกันในระบบ เพื่อใช้สำหรับติดต่อระหว่างอุปกรณ์ต่างๆ ของระบบ ซึ่งสามารถแบ่งออกเป็นหลักการได้ ดังนี้

5.1 ประเภทของการส่งผ่านข้อมูล

5.1.1 การส่งผ่านข้อมูลแบบขนาน

การส่งผ่านข้อมูลแบบขนานนั้น จะทำการส่งผ่านข้อมูลครั้งละหลาย ๆ บิท เช่น ส่งข้อมูล 11001001 ทั้ง 8 บิท ออกไปพร้อม ๆ กัน โดยผ่านสายส่งสัญญาณที่มี 8 เส้น โดยที่ทุกๆ บิทที่ส่งไปนั้นจะถึงปลายทางพร้อมกัน ดังแสดงในรูปที่ 5.1 จำนวนของสัญญาณ หรือจำนวนของสายจะมีจำนวนที่ไม่แน่นอน ขึ้นอยู่กับโครงสร้างการประมวลผลข้อมูลของระบบที่ใช้ ข้อดีของการส่งผ่านข้อมูลขนานคือ สามารถส่งผ่านข้อมูลได้รวดเร็ว ในระยะเวลาสั้นๆ ส่วนข้อเสียก็คือ การสิ้นเปลืองสายสัญญาณจำนวนมาก และหากใช้ในการสื่อสารระยะทางไกล ๆ นอกจากจะเสียค่าใช้จ่ายสูงแล้ว ยังเกิดการลดทอนของสัญญาณด้วย

โดยทั่วไปแล้วการส่งผ่านข้อมูลแบบขนาน จะใช้กับการส่งผ่านข้อมูลในระยะทางใกล้ ๆ และต้องการส่งผ่านข้อมูลด้วยความเร็วสูง เช่น การเคลื่อนย้ายข้อมูลระหว่างเครื่องคอมพิวเตอร์กับอุปกรณ์รอบข้างตัวอย่างเช่น เครื่องพิมพ์ เครื่องขับแผ่นจานแม่เหล็ก เครื่องขับแผ่นซีดีรอม เป็นต้น หรือใช้ในการส่งผ่านข้อมูลระหว่าง เครื่องมือวัดและอุปกรณ์ต่างๆ กับเครื่องคอมพิวเตอร์ตามมาตรฐาน IEEE-488



รูปที่ 5.1 แสดงโครงสร้างการสื่อสารข้อมูลแบบขนาน

5.1.2 การส่งผ่านข้อมูลแบบอนุกรม

การส่งผ่านข้อมูลแบบอนุกรม ข้อมูลจะถูกส่งออกมาทีละบิท จากต้นทางถึงปลายทาง โดยที่ตัวกลางการสื่อสารใช้ช่องสัญญาณเพียงช่องเดียว หรือสายเพียงคู่เดียว ได้แก่ สายสัญญาณข้อมูล และสายกราวด์เปรียบเทียบ จะเห็นว่าเมื่อเปรียบเทียบกับการส่งผ่านข้อมูลแบบขนาน ที่จำนวนข้อมูล และอัตราความเร็วในการส่งผ่านข้อมูลเท่ากันแล้ว การส่งผ่านข้อมูลแบบอนุกรม จะช้ากว่าแบบขนาน สำหรับข้อดีของการส่งผ่านข้อมูลแบบอนุกรมนี้ คือค่าใช้จ่ายถูกกว่า เพราะใช้สายสัญญาณน้อยกว่า ทำให้นิยมใช้ในการส่งสัญญาณระยะทางไกล ๆ แม้ว่าอัตราการลดทอนหรือผิดเพี้ยนของ สัญญาณที่มีผลจากความยาวของสายสัญญาณจะมีค่าเท่ากับการส่งผ่านข้อมูลแบบขนาน แต่การส่งผ่านข้อมูลแบบอนุกรมมีวิธีการที่จะลดผลจากการลดทอนและความผิดเพี้ยนของสัญญาณนี้โดยอาศัยวิธีการ 2 แบบ คือ

1. การสื่อสารอนุกรมแบบซิงโครนัส

2.การสื่อสารอนุกรมแบบอะซิงโครนัส

รายละเอียดของวิธีการทั้งสองแบบจะได้กล่าวถึงต่อไป

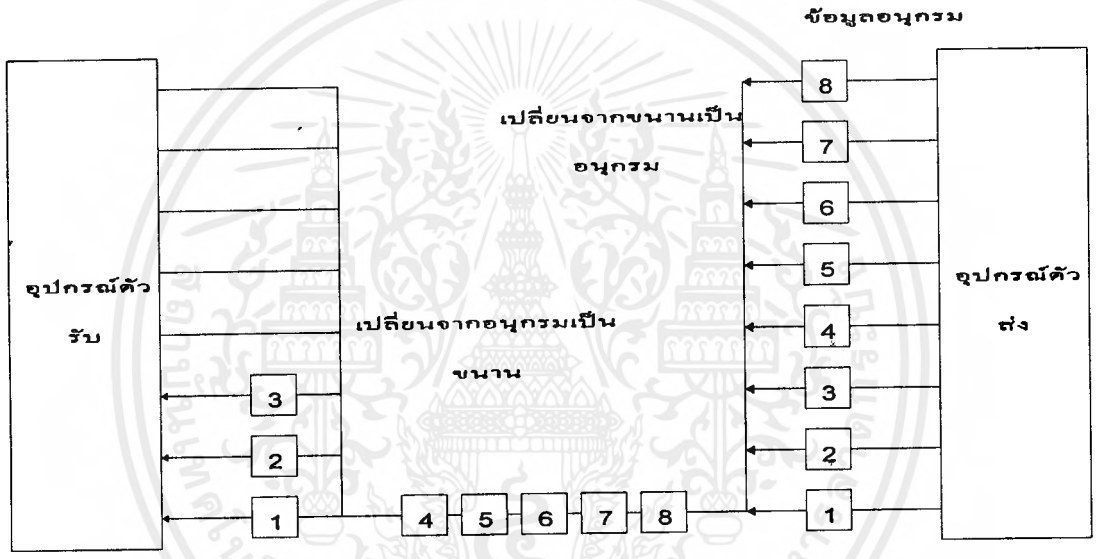
5.2 รูปแบบของการสื่อสารแบบอนุกรม

การสื่อสารแบบอนุกรม แบ่งตามลักษณะของทิศทางในการสื่อสาร ได้ 3 แบบ ดังนี้คือ

1. แบบทางเดียว (Simplex) สัญญาณจะถูกส่งในทิศทางเดียวเท่านั้น เช่น การส่งสัญญาณวิทยุกระจายเสียง การส่งสัญญาณโทรทัศน์ เป็นต้น

2. แบบกึ่งสองทาง (Half-duplex) สามารถส่งสัญญาณได้ทั้งสองทิศทาง แต่ต้องผลัดกันส่งและรับ ไม่สามารถส่งและรับข้อมูลพร้อมกันได้ เช่น วิทยุสื่อสารแบบต่าง ๆ

3. แบบสองทาง (Full-duplex) สามารถส่งและรับสัญญาณในเวลาเดียวกันได้ เช่น การสื่อสารทางโทรศัพท์



รูปที่ 5.2 แสดงโครงสร้างการสื่อสารข้อมูลแบบอนุกรม

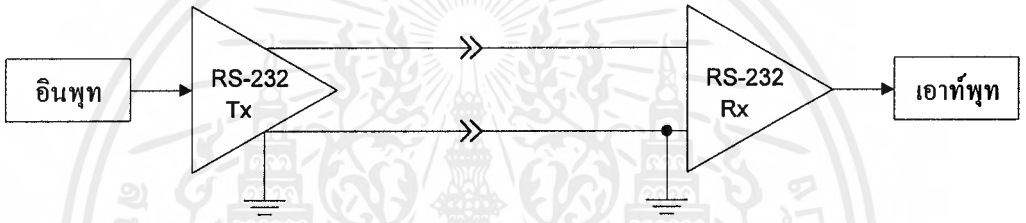
Simplex เป็นการสื่อสารข้อมูลในทิศทางใดก็จะใช้ทิศทางนั้นตลอดเวลาไม่มีการเปลี่ยนทิศทาง เช่น การส่งสัญญาณภาพจากสถานีไปยังเครื่องรับ หรือ การส่งข้อมูลไปยังวิทยุติดตามตัวหรือเพจเจอร์

Half-Duplex เป็นการสื่อสารข้อมูลใน 2 ทิศทาง แต่ในขณะเวลาหนึ่งนั้นสัญญาณจะไปในทิศทางเดียวเท่านั้น ดังนั้น อุปกรณ์แต่ละตัวที่จะเชื่อมต่อหรือสื่อสารข้อมูลในลักษณะนี้จะต้องเป็นได้ทั้งตัวรับและตัวส่ง ซึ่งมีชื่อเรียกว่า ทรานซิฟเวอร์ (tranceiver) และจะต้องมีวงจรที่จะเลือกว่า ณ เวลานั้นจะมีการทำงานเป็นตัวรับหรือตัวส่ง

Full-duplex เป็นการสื่อสารข้อมูลที่คล้ายกับ half-duplex แต่เป็นการสื่อสารข้อมูลใน 2 ทิศทางตลอดเวลา ดังแสดงลักษณะการสื่อสารข้อมูลแบบอนุกรมในแบบต่างๆ ในรูปที่ 5.2

5.2.1 การสื่อสารข้อมูลแบบอนุกรมตามมาตรฐาน RS-232C

การสื่อสารข้อมูลแบบอนุกรมที่ใช้งานอยู่ในปัจจุบันนั้นได้มีการกำหนดมาตรฐานการรับ-ส่งข้อมูลไว้หลายแบบด้วยกัน แต่ที่ได้รับความนิยมนำมาใช้งานอย่างมาก คือ การสื่อสารข้อมูลแบบอนุกรมตามมาตรฐาน RS-232C และที่มาตรฐานนี้เป็นที่นิยม เนื่องจากเป็นระบบการสื่อสารข้อมูลที่ใช้ในเครื่องไมโครคอมพิวเตอร์ IBM PC ซึ่งเป็นคอมพิวเตอร์ที่มีใช้อย่างแพร่หลายมากจากอดีตจนถึงปัจจุบัน มาตรฐานการสื่อสารนี้ในการออกแบบเบื้องต้นได้ออกแบบมาเพื่อสำหรับการเชื่อมต่อกับเครื่องโมเด็ม (MODEM, Modulator/Demodulator) ซึ่งเป็นอุปกรณ์ที่ใช้การสื่อสารข้อมูลระหว่างคอมพิวเตอร์ผ่านทางสายโทรศัพท์ ซึ่งทำให้อัตราการรับส่งข้อมูลถูกจำกัดให้มีค่าที่ค่อนข้างต่ำและมาตรฐาน RS-232C นี้ ได้ออกแบบให้มีโครงสร้างการสื่อสารเป็นจุดต่อจุดเท่านั้น โดยมีลักษณะสมบัติทางไฟฟ้าและทางกายภาพ ดังแสดงในตารางที่ 5.1 และรูปที่ 5.3

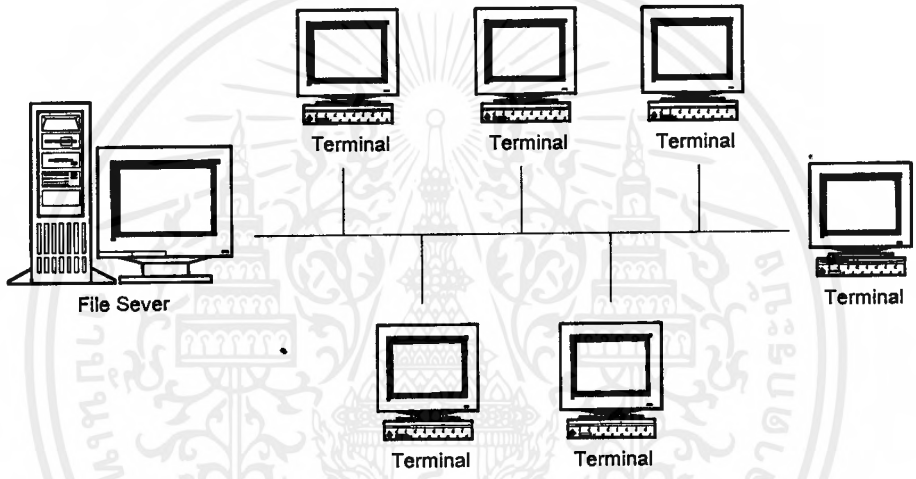


รูปที่ 5.3 แสดงโครงสร้างของการสื่อสารข้อมูลแบบอนุกรมตามมาตรฐาน RS-232C

5.2.2 การสื่อสารข้อมูลแบบอนุกรมตามมาตรฐาน RS-422A

ในการออกแบบระบบการสื่อสารข้อมูลจากที่ผ่านมาจนถึงปัจจุบันได้มีการพยายามที่จะออกแบบให้การสื่อสารข้อมูลได้รวดเร็วยิ่งขึ้น และมีระยะในการสื่อสารที่มากขึ้นด้วย ซึ่งที่ผ่านมาการสื่อสารข้อมูลตามมาตรฐาน RS-232C ได้ออกแบบเพื่อใช้เชื่อมต่อกับโมเด็มเท่านั้น จึงไม่ได้คำนึงถึงความเร็วและระยะทางในการสื่อสาร แต่ในปัจจุบันได้มีมาตรฐานการสื่อสารข้อมูลใหม่ที่ได้ออกแบบมาเพื่อรองรับความต้องการของผู้ใช้งานที่ต้องการให้การรับส่งข้อมูลได้ระยะทางไกลและรวดเร็วขึ้น มาตรฐานนี้คือ RS-422A ซึ่งการที่มาตรฐานการสื่อสารนี้ สามารถรับส่งข้อมูลได้ไกลและรวดเร็วขึ้นมาจากหลักการที่ใช้สัญญาณเป็นแบบดิฟเฟอเรนเชียล ดังแสดงในรูปที่ 5.4 ซึ่งหลักการก็คือ สัญญาณที่รับ-ส่งจะเป็นการเปรียบเทียบระหว่างสายสัญญาณ 2 เส้น เทียบกับมาตรฐาน RS-232C ที่สัญญาณทุกสัญญาณจะเทียบกับกราวด์ ซึ่งในการสื่อสารในระยะทางไกลๆ แล้วสัญญาณจะถูกลดทอนไปและเมื่อสัญญาณถูกลดทอนถึงจุดๆ หนึ่งสัญญาณนั้นก็จะผิดพลาดไปจากความเป็นจริงก็ทำให้การรับ-ส่งข้อมูลเกิดความผิดพลาดขึ้นแต่สำหรับสัญญาณแบบดิฟเฟอเรนเชียลแล้วการลดทอนของสัญญาณก็จะไปลดทอนทั้งสองสายด้วยค่าที่เท่ากันหรือใกล้เคียงกัน และความ

กระจายไปยังอุปกรณ์ที่ต่ออยู่กับเครือข่ายได้พร้อมกันด้วยการส่งเพียงครั้งเดียว ทำให้สิ้นเปลืองสายสัญญาณน้อยกว่า และสามารถสื่อสารข้อมูลใน 2 ทิศทางแบบสลับเวลา หรือ Half-duplex และในกรณีที่ต้องการให้การสื่อสารข้อมูลรวดเร็วขึ้น ยังสามารถจะขยายระบบให้การสื่อสารข้อมูลพร้อมกันได้ใน 2 ทิศทางแบบตลอดเวลา หรือ Full-duplex โดยการเพิ่มสายหรือช่องสัญญาณที่ขนานไปกับสายหรือช่องสัญญาณเดิมอีก 1 ชุด จากที่กล่าวจะแสดงให้เห็นถึงข้อดีของโครงสร้างของเครือข่ายแบบนี้แต่เครือข่ายแบบนี้ก็มีข้อเสียที่สำคัญ คือ ในกรณีระบบเกิดทำงานผิดปกติการตรวจหาจุดที่เกิดปัญหานั้นทำได้ยากและอาจทำให้เครือข่ายทั้งระบบทำงานไม่ได้ เพราะสายหรือช่องสัญญาณในการสื่อสารข้อมูลมีเพียงชุดเดียว ซึ่งในการพัฒนาต่อมาได้ทำให้โหนดที่เกิดทำงานผิดปกติสามารถตัดออกจากเครือข่ายได้โดยไม่รบกวนต่อระบบ

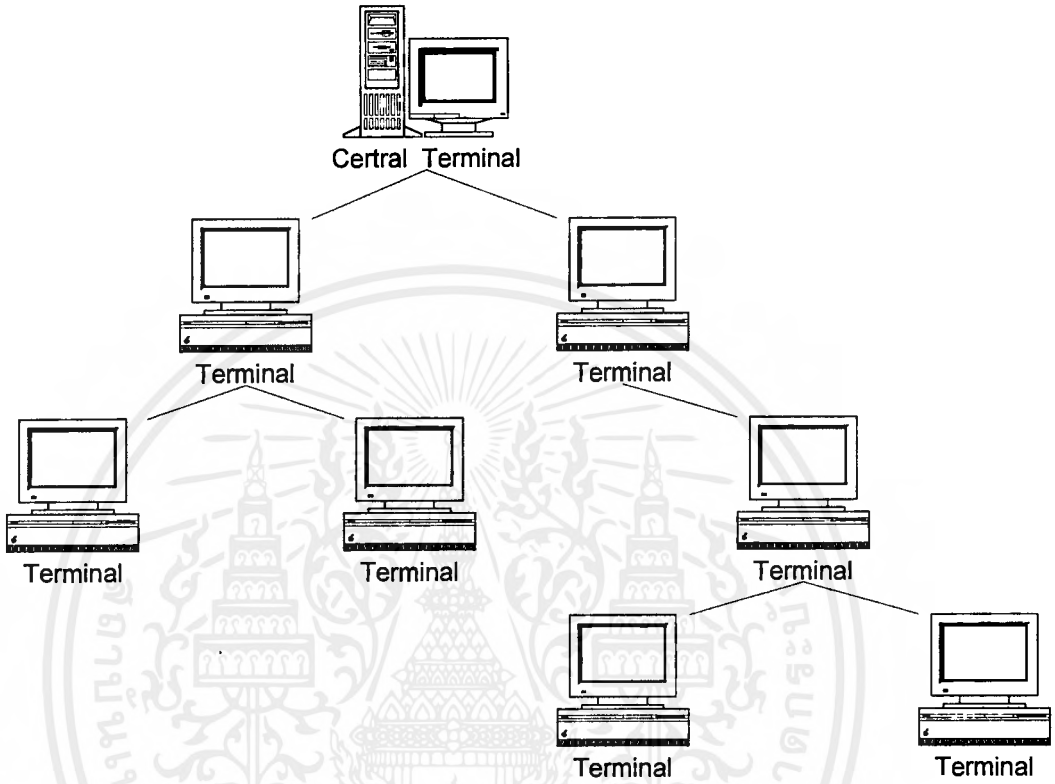


รูปที่ 5.6 แสดงโครงสร้างของเครือข่ายแบบบัส

5.3.2 การสื่อสารข้อมูลเป็นเครือข่ายแบบทรี

เครือข่ายการสื่อสารข้อมูลแบบทรี (Tree Network) เป็นโครงสร้างของเครือข่ายข้อมูลที่สามารถควบคุมระบบได้ง่าย โดยโครงสร้างของเครือข่ายข้อมูลแบบนี้ มีการกระจายศูนย์กลางควบคุมออกไปอย่างเป็นลำดับชั้น คล้ายกับการแผ่กิ่งก้านของต้นไม้ ดังในรูปที่ 5.7 ซึ่งที่ทุกจุดแยกจะมีศูนย์กลางย่อย เพื่อทำหน้าที่ในการควบคุมการทำงานของโหนดย่อยที่แยกออกไปจากจุดนี้ ทำให้การควบคุม, การหาจุดผิดปกติ และการแก้ไขสามารถทำได้สะดวกรวดเร็วกว่า และยังเป็นการลดความหนาแน่นของงานที่ศูนย์กลางหลักของระบบ อย่างไรก็ตามโครงสร้างของเครือข่ายแบบนี้ ก็ยังมีข้อเสีย คือมีโอกาสที่ข้อมูลจะกระจุกเป็นคอขวดอยู่ที่ตำแหน่งศูนย์กลางได้ และถ้าโหนดที่มีลำดับชั้นที่สูงกว่าเกิดมีปัญหาขึ้นมาจะมีผลต่อโหนดที่ลำดับชั้นต่ำกว่าด้วย และจะมีผลกระทบมากยิ่งขึ้นถ้าโหนดที่ทำงานผิดปกติเป็นศูนย์กลางย่อย ผลจากการเกิดเหตุการณ์นี้อาจมีผลให้ข้อมูลเกิดการสูญหายหรือผิดพลาด และเป็นการลดความเชื่อถือได้ของระบบอีกด้วย ยิ่งไปกว่านั้นการที่จะสื่อสาร

กันระหว่างโหนดที่มีลำดับชั้นต่างกันจะทำได้ยาก เพราะจะต้องผ่านข้อมูลหลายขั้นตอนตามลำดับชั้นของเครือข่าย แต่อย่างไรก็ตามโครงสร้างการสื่อสารแบบนี้ก็ยังได้รับความนิยมนำไปใช้งาน เนื่องจากสามารถเพิ่มจำนวนโหนดได้ง่าย

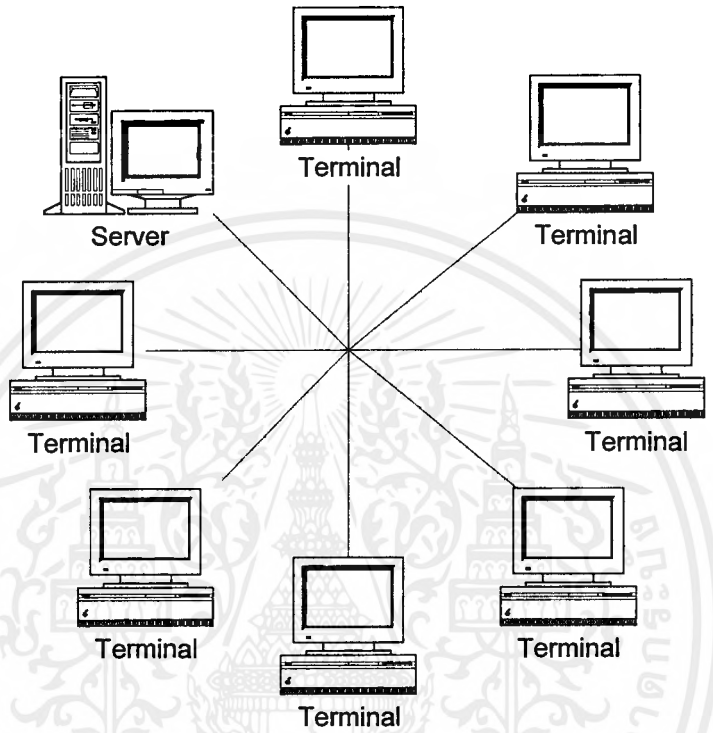


รูปที่ 5.7 แสดงโครงสร้างของเครือข่ายแบบทรี

5.3.3 การสื่อสารข้อมูลเป็นเครือข่ายแบบสตาร์

โครงสร้างของเครือข่ายการสื่อสารแบบสตาร์ (Star Network) เป็นโครงสร้างที่ได้รับความนิยมนำมาใช้อย่างมากจากอดีตจนถึงปัจจุบัน เนื่องจากมีความง่าย และความสะดวกในการควบคุม รวมทั้งโปรแกรมควบคุมก็ไม่ซับซ้อน โดยโครงสร้างของเครือข่ายการสื่อสารข้อมูลแบบนี้ ต้องเป็นการสื่อสารผ่านศูนย์กลางหลักของระบบเท่านั้น และสายสัญญาณของแต่ละโหนด จะกระจายออกไปจากศูนย์กลางข้อมูลเพียงชุดเดียว ดังแสดงในรูปที่ 5.8 ซึ่งจากเครือข่ายแบบนี้จะคล้ายกับเครือข่ายแบบทรี แต่เครือข่ายแบบสตาร์จะมีข้อจำกัดในแง่ของจำนวนโหนดที่มาต่อเข้ากับระบบ และยังมีข้อเสียที่คล้ายกับเครือข่ายแบบทรี คือมีโอกาสที่จะเกิดการกระจุกของข้อมูล รวมทั้งการเสียหายหรือการทำงานผิดปกติของศูนย์กลางของระบบ จะมีผลให้เครือข่ายทำงานไม่ได้ หรือทำให้เกิดความผิดพลาดของข้อมูลขึ้นได้ จึงเป็นการลดความน่าเชื่อถือของข้อมูลภายในระบบ ซึ่งในปัจจุบันได้มีการพยายามเพิ่มความน่าเชื่อถือ โดยเพิ่มศูนย์กลางสำรองที่จะทำงานแทนศูนย์กลางข้อมูลหลัก

ในกรณีที่ไม่สามารถทำงานได้ อย่างไรก็ตามเครือข่ายแบบนี้ก็มีข้อดี ที่สามารถแยกโหนดที่ทำงานผิดปกติได้ง่ายกว่า และเส้นทางในการสื่อสารของแต่ละโหนดจะแยกจากกันเป็นอิสระ ทำให้โหนดแต่ละตัวภายในเครือข่ายแบบนี้สามารถสื่อสารข้อมูลกันได้ แม้ว่าจะมีอัตราการสื่อสารข้อมูล มาตรฐานการสื่อสารข้อมูลและโครงสร้างของข้อมูลที่ต่างกัน

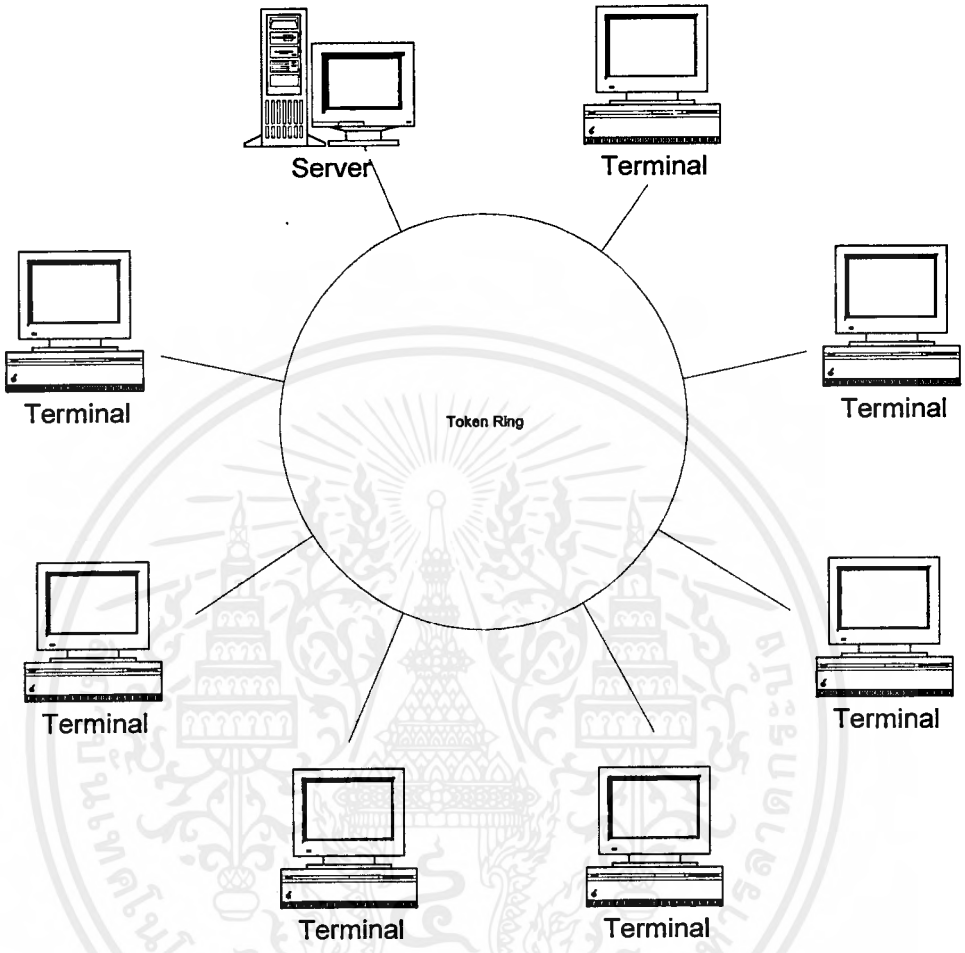


รูปที่ 5.8 แสดงโครงสร้างของเครือข่ายแบบสตาร์

5.3.4 การสื่อสารข้อมูลเป็นเครือข่ายแบบวงแหวน

โครงสร้างของเครือข่ายแบบวงแหวน (Ring Network) นี้ก็เป็นเครือข่ายอีกแบบหนึ่งที่ได้รับความนิยมนำไปใช้งาน สาเหตุที่เรียกเครือข่ายแบบนี้ว่าเป็นวงแหวนเนื่องจากโครงสร้างทางกายภาพของระบบ ที่การไหลเวียนของข้อมูลจะวนอยู่ในลักษณะเป็นวงกลม และข้อมูลจะไหลไปในทิศทางเดียว ดังแสดงในรูปที่ 5.9 โดยการทำงานเริ่มจากเมื่อข้อมูลเข้ามาที่โหนด ก็จะทำการตรวจสอบว่าเป็นข้อมูลของโหนดที่กำลังติดต่อสื่อสารกันอยู่หรือไม่ ถ้าใช่ก็จะรับข้อมูลนี้เข้าไป แต่ถ้าไม่ใช่ก็จะทำการส่งข้อมูลไปยังโหนดถัดไป และทำในลักษณะเดียวกันต่อไปภายในเครือข่าย โดยข้อดีของเครือข่ายแบบนี้คือ จะไม่เกิดการกระจุกกันของข้อมูลและมีโครงสร้างที่ง่ายต่อการควบคุม เพราะโหนดทุกโหนดจะทำงานในลักษณะเหมือนกัน แต่ก็มีข้อเสียที่สำคัญคือ ถ้าสายสัญญาณภายในเครือข่ายเกิดเสีย หรือขาดก็จะทำให้เครือข่ายของระบบหยุดทำงาน และมีโอกาสที่ข้อมูลซึ่งไม่ต้อง

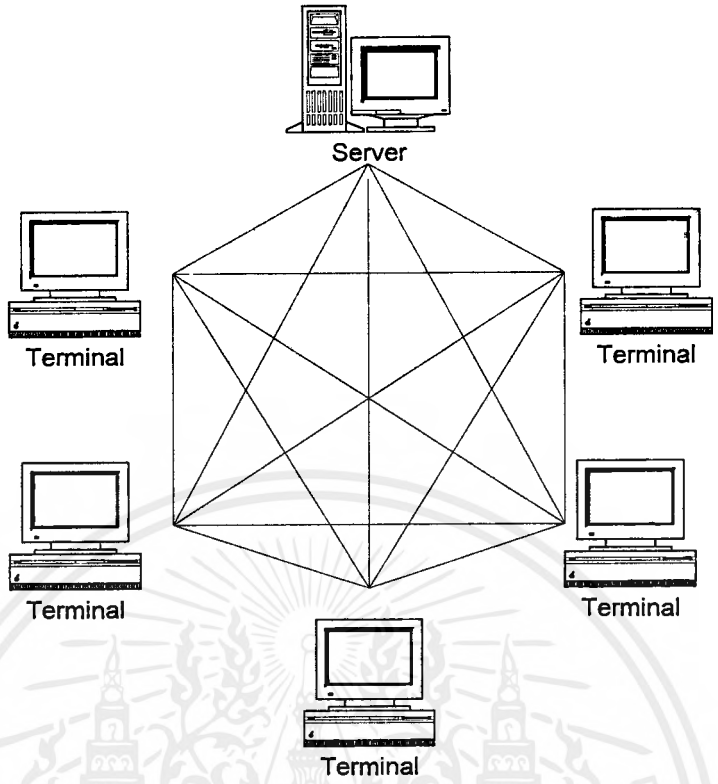
การ หรือไม่ใช้งานแล้ว ยังคงวิ่งวนอยู่ภายในเครือข่าย ทำให้ความสามารถในการสื่อสารข้อมูลของระบบลดลง



รูปที่ 5.9 แสดงโครงสร้างของเครือข่ายแบบวงแหวน

5.3.5 การสื่อสารข้อมูลเป็นเครือข่ายแบบเมช

โครงสร้างของเครือข่ายข้อมูลแบบเมช (Mesh Network) เป็นเครือข่ายที่จะมีช่องสัญญาณที่จะให้โหนดแต่ละตัวสามารถจะสื่อสารข้อมูลไปยังโหนดอื่นทุกตัว ได้โดยไม่ผ่านโหนดอื่น ๆ ดังแสดงในรูปที่ 5.10 โดยเครือข่ายแบบนี้มีข้อดีที่สำคัญ คือ สามารถแก้ปัญหาการกระจุกของข้อมูลและปัญหาการที่อุปกรณ์และช่องสัญญาณเกิดเสียหรือผิดปกติ อย่างไรก็ตามเครือข่ายแบบนี้ก็มีข้อเสีย ที่การวางเครือข่ายจะมีความสลับซับซ้อน, การควบคุมระบบทำได้ยาก, มีช่องสัญญาณที่ไม่ได้ใช้งาน (ใช้งานช่องสัญญาณไม่คุ้มค่า) และความสิ้นเปลืองสายสัญญาณเป็นจำนวนมากถ้ามีจำนวนโหนดมากขึ้น จึงทำให้ไม่ค่อยได้รับความนิยมนำไปใช้งาน แต่ถึงกระนั้นก็ตามถ้าผู้ใช้งานเครือข่ายที่ต้องการความเชื่อถือได้ของข้อมูลสูงแล้วก็ควรจะเลือกใช้เครือข่ายแบบเมช



รูปที่ 5.10 แสดงโครงสร้างของเครือข่ายแบบเมช

ในปริญญานิพนธ์นี้เลือกใช้โครงสร้างของเครือข่ายข้อมูลแบบบัส ในการสื่อสารข้อมูล เนื่องจากโครงสร้างพื้นฐานของระบบที่ใช้ จะมีศูนย์กลางข้อมูลเพียงจุดเดียว จึงไม่เหมาะสมที่จะใช้เครือข่ายแบบทรีและแบบเมช สำหรับเครือข่ายแบบวงแหวนจะมีขั้นตอนและวิธีการควบคุมที่ยุ่งยากและสลับซับซ้อนมาก ส่วนเครือข่ายแบบสตาร์มีข้อจำกัดในเรื่องจำนวนโหนดที่จะนำมาต่อกับระบบ และจะใช้สายสัญญาณมากกว่าแบบบัสเมื่อใช้จำนวนโหนดเท่ากัน เหตุผลหลักในการเลือใช้เครือข่ายแบบบัส เพราะสะดวกและง่ายทั้งในการเดินสายสัญญาณและการเขียนโปรแกรมควบคุมระบบ โดยเราออกแบบให้ตัวคอมพิวเตอร์ศูนย์กลางเป็นตัวเลือกในการที่จะติดต่อกับไมโครคอนโทรลเลอร์ที่ตัวเครื่องอ่านรหัสแท่ง เมื่อติดต่อกันได้แล้วไมโครคอนโทรลเลอร์ตัวนั้น จะทำการส่งข้อมูลให้ศูนย์กลางข้อมูลจนกว่าจะหมดข้อมูล ส่วนตัวอื่นที่ไม่ถูกเลือกจะไม่สามารถส่งข้อมูลได้ ซึ่งจะเป็นการง่ายต่อการควบคุมระบบ

5.4 โปรโตคอล

องค์ประกอบสำคัญสำหรับการเป็นเครือข่ายข้อมูลนอกจากหลักการและโครงสร้างของเครือข่ายที่ได้กล่าวถึงแล้ว ยังมีองค์ประกอบที่สำคัญและขาดไม่ได้สำหรับการเป็นเครือข่าย คือ กฎหรือ

ข้อกำหนดในการสื่อสารข้อมูล หรือที่นิยมเรียกว่า โปรโตคอล (protocols) ซึ่งเป็นส่วนที่จะกำหนดมาตรฐานของในการควบคุมและจัดการระบบการสื่อสารข้อมูลในส่วนต่างๆ ซึ่งสามารถแบ่งโปรโตคอลได้เป็นหลายลำดับชั้น (layer) ซึ่งมาตรฐานการแบ่งระดับนั้นได้มีองค์กรต่างๆ แบ่งไว้หลายแบบ แต่ที่ได้รับความนิยมมากที่สุด คือการแบ่งลำดับชั้นตามมาตรฐาน OSI (open system interconnection) ซึ่งจะแบ่งออกเป็น 7 ลำดับชั้น ดังนี้

1. physical layer
2. data-link layer
3. network layer
4. transport layer
5. session layer
6. presentation layer
7. application layer

โดยในลำดับชั้นที่ 1-3 จะเป็นการควบคุมและจัดการการสื่อสารข้อมูลทางกายภาพ ส่วนในลำดับชั้น 5-7 เป็นลำดับชั้นที่ควบคุมและจัดการในส่วนที่ติดต่อกับผู้ใช้งาน และสำหรับลำดับชั้นที่ 4 เป็นส่วนที่เชื่อมโยงระหว่าง 3 ลำดับชั้นล่าง (ชั้น 1-3) กับ 3 ลำดับชั้นบน (ชั้น 5-7)สำหรับรายละเอียดที่กล่าวในที่นี้จะเกี่ยวข้องเฉพาะในส่วนของโปรโตคอลการควบคุมการเชื่อมโยงข้อมูล (data link control protocols หรือ DLCP) ซึ่งจัดการในส่วนของขั้นตอนและหลักการต่างๆ คือ โครงสร้างและรายละเอียดของข้อมูล, วิธีในการสื่อสารข้อมูล, การตรวจสอบและแก้ไขความผิดพลาดของข้อมูล, และขบวนการในการควบคุมการติดต่อสื่อสาร โดย DLCP แบ่งได้ตามโครงสร้างของข้อมูล 2 แบบ คือ Byte-oriented protocols และ Bit-oriented protocols

5.4.1 ไบท์โอเรียนท์โปรโตคอล (Byte-oriented protocols)

โปรโตคอลแบบนี้เป็นโปรโตคอลที่การสื่อสารข้อมูลและการควบคุมการทำงานจะทำโดยใช้ลักษณะข้อมูลที่เป็นตัว (character) หรือ ไบท์ (byte)

1. อะซิงโครนัสโปรโตคอล (Asynchronous protocol)

โปรโตคอลในการสื่อสารข้อมูลนี้จะใช้การสื่อสารข้อมูลแบบ half-duplex ที่มีลักษณะการสื่อสารข้อมูลแบบอะซิงโครนัส ซึ่งเป็นการสื่อสารข้อมูลแบบพื้นฐานที่ใช้งานมาเป็นเวลานานแล้ว จึงมีรายละเอียด และขั้นตอนในการสื่อสารข้อมูล ที่ทำให้มีโอกาสเกิดความผิดพลาดได้ง่าย แต่อย่างไรก็ตามการสื่อสารข้อมูลแบบนี้ ก็มีข้อดีที่มีโครงสร้างการทำงานที่ง่าย อุปกรณ์ที่ใช้ในการสื่อ

สารข้อมูลก็ไม่สลับซับซ้อนและมีราคาถูก โปรโตคอลแบบนี้จึงเหมาะสำหรับใช้ในระบบขนาดเล็กและระบบที่มีการสื่อสารข้อมูลด้วยความเร็วไม่สูงมากนัก

2. ไบนารีซิงโครนัสโปรโตคอล (Binary synchronous protocol)

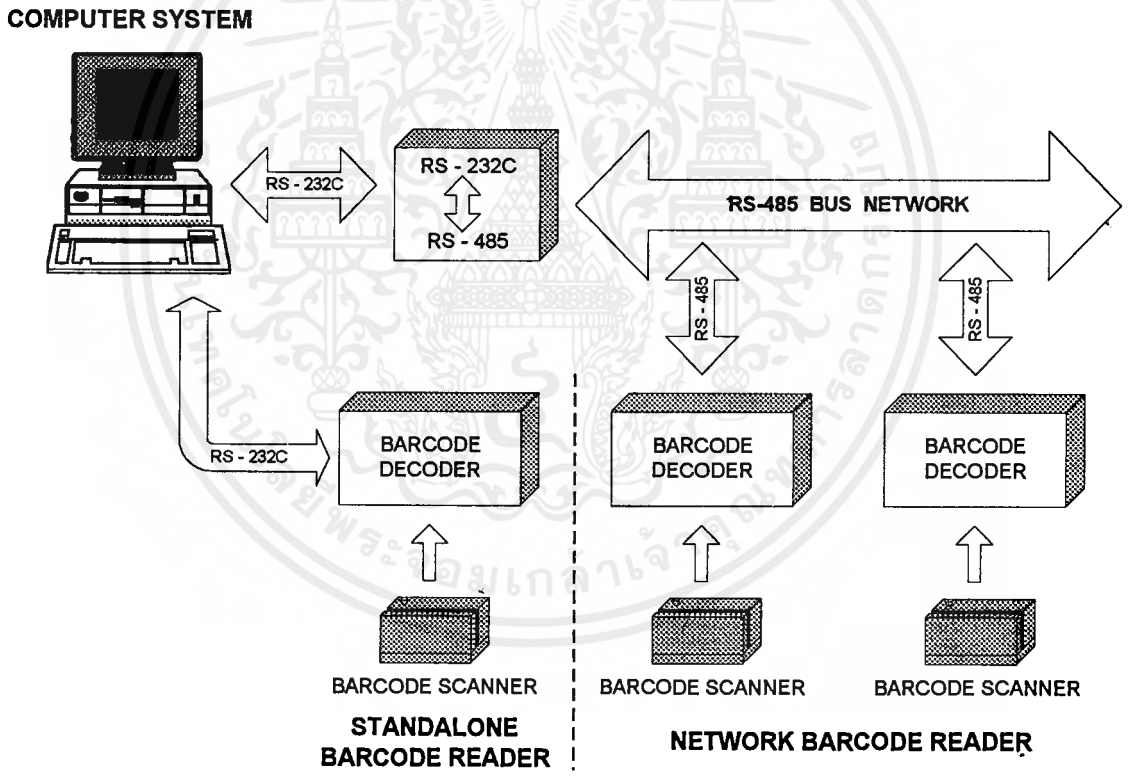
โปรโตคอลแบบนี้จะมีลักษณะการทำงานที่ใช้งานข้อมูลเป็นในลักษณะไบท์ และยังคงใช้การสื่อสารข้อมูลแบบ Half-duplex ซึ่งจะคล้ายกับซิงโครนัสโปรโตคอล แต่ที่ต่างกันคือ จะใช้การสื่อสารข้อมูลแบบอะซิงโครนัส และรายละเอียดและขั้นตอนในการสื่อสารข้อมูลที่ให้ความน่าเชื่อถือมากกว่า อีกทั้งยังสามารถใช้อัตราเร็วในการสื่อสารข้อมูลที่สูงกว่า โดยตัวอย่างของการสื่อสารข้อมูลแบบนี้ได้มีการกำหนดเป็นมาตรฐานแล้ว คือการสื่อสารข้อมูลตามมาตรฐาน BSC (Binary synchronous communications) ซึ่งเป็นโปรโตคอลที่มีลักษณะของข้อมูลแบบไบท์ ที่ได้รับความนิยมนำไปใช้งาน

5.4.2 บิทโอเรียนท์โปรโตคอล (Bit-oriented Protocols)

โปรโตคอลแบบนี้ เป็นโปรโตคอลที่การสื่อสารข้อมูลและการควบคุมจะทำโดยใช้ลักษณะข้อมูลที่เป็นบิท โดยมีตัวอย่างของการสื่อสารข้อมูลในลักษณะนี้ มีการกำหนดขึ้นเป็นมาตรฐานแล้ว คือ HDLC (high-level data link control) โดยมีโครงสร้างของข้อมูลแบบซิงโครนัส เช่นเดียวกับ BSC แต่ต่างกันที่มีลักษณะของข้อมูลเป็นแบบบิท ซึ่งโปรโตคอลแบบนี้มีข้อดีที่ที่สามารถสื่อสารข้อมูลแบบ Full-duplex ได้ทำให้การสื่อสารข้อมูลได้รวดเร็วกว่า แต่โปรโตคอลแบบนี้ก็มีรายละเอียดและโครงสร้างในการสื่อสารข้อมูลที่สลับซับซ้อนมากทำให้การควบคุมการทำงานทำได้ยากและต้องใช้อุปกรณ์ที่มีราคาสูง จึงไม่เหมาะที่จะนำไปใช้งานกับระบบขนาดเล็ก

บทที่ 6
โครงสร้างและการออกแบบระบบ

โครงสร้างของระบบบรหัสแท่งแบบเครือข่ายแสดงดังรูปที่ 6.1 ซึ่งแบ่งโครงสร้างระบบออกเป็น 2 ส่วน คือส่วนของฮาร์ดแวร์ และส่วนของซอฟท์แวร์ โดยที่ส่วนของฮาร์ดแวร์ คือเครื่องอ่านรหัสแท่ง และวงจรส่วนที่ใช้ในการสื่อสารข้อมูล ระหว่างเครื่องไมโครคอมพิวเตอร์กับเครื่องอ่านรหัสแท่ง ส่วนของซอฟท์แวร์คือโปรแกรมการถอดรหัสแท่งในเครื่องอ่านรหัสแท่ง และโปรแกรมจัดการข้อมูล กับโปรแกรมการสื่อสารข้อมูลบนไมโครคอมพิวเตอร์



รูปที่ 6.1 โครงสร้างระบบบรหัสแท่งแบบเครือข่าย

รหัสแท่งที่เลือกใช้ในระบบคือ รหัส39 (Code 39) สำหรับหัวอ่านรหัสแท่งเป็นแบบสล็อต (Slot barcode scanner) จุดประสงค์ที่ใช้แบบนี้เพื่อใช้ในการรูดบัตรเข้า-ออกห้องปฏิบัติการของนักศึกษา (บันทึกรหัสนักศึกษา และ วัน-เวลา) โดยกำหนดให้เครื่องอ่านรหัสแท่งแต่ละตัวเป็นโหนด

(Node) ของระบบเครือข่ายที่ใช้โทโปโลยีแบบบัส (Bus Network) สำหรับห้องปฏิบัติการหลาย ๆ ห้อง ในกรณีที่ห้องปฏิบัติการอยู่ห่างไกลจาก เครื่องไมโครคอมพิวเตอร์ที่ใช้เป็นศูนย์กลางข้อมูลของระบบมากเกินกว่าที่จะเดินสายได้ ก็สามารถนำเครื่องอ่านรหัสแท่งนี้ ไปใช้แบบเครื่องเดี่ยว ๆ (Standalone) ได้ และนำมาเชื่อมต่อกับศูนย์กลางข้อมูลของระบบหลังจากเก็บข้อมูลการบันทึกเวลาที่ต้องการได้แล้ว โดยส่วนนี้ใช้การเชื่อมต่อสื่อสารอนุกรมมาตรฐาน RS-232C

6.1 โครงสร้างของเครื่องอ่านรหัสแท่ง

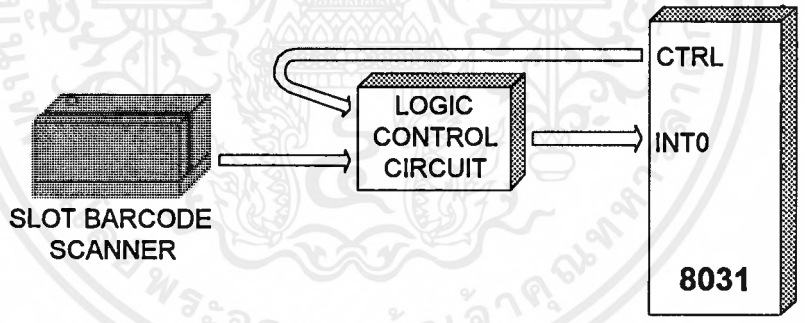
หลักการทำงานของเครื่องอ่านรหัสแท่ง คือ เมื่ออ่านรหัสแท่งและประมวลผลจนได้ข้อมูลที่ถูกต้องแล้วจะทำการแปลงข้อมูลให้เป็นรหัสแอสกี้ (ASCII code) สัญญาณที่เหมือนกับสัญญาณที่ได้จากการกดแป้นข้อมูลนั้น ดังนั้นเมื่อสิ้นสุดการอ่านรหัสแต่ละครั้ง (ที่ผลการอ่านถูกต้อง) จะมีข้อมูลของรหัสแท่งแสดงผลอยู่ที่หน้าจอแสดงผลของเครื่องไมโครคอมพิวเตอร์เสมือนกับการป้อนข้อมูลผ่านทางแป้นพิมพ์ โดยโครงสร้างของเครื่องอ่านรหัสแท่งแบบนี้จะแสดงในรูปที่ 6.1

6.1.1 โครงสร้างและการออกแบบวงจรการอ่านรหัสแท่ง

ในการอ่านรหัสแท่งนั้นสัญญาณที่ได้จากตัวหัวอ่านเป็นสัญญาณในรูปของพัลส์ที่แปรผันตามขนาดความกว้างของรหัสแท่ง ดังแสดงในรูปที่ 6.2 ซึ่งเห็นได้ว่าสัญญาณที่ได้จะมี 2 สภาวะ หรือเป็นลอจิก คือ “0” และ “1” โดยเมื่อรูดหัวอ่านผ่านแท่งดำก็จะได้ลอจิก “1” ส่วนแท่งขาวจะได้เป็นลอจิก “0”

รูปที่ 6.2 สัญญาณที่ได้จากหัวอ่านรหัสแท่ง

การแยกแท่งรหัส หรือการแทนค่าแท่งรหัส จะอาศัยความแตกต่างจากความกว้างของแท่งรหัส ดังนั้นในการอ่านรหัสแท่ง จึงต้องมีส่วนที่จะทำหน้าที่ ในการหาขนาดความกว้างของแท่งแต่ละแท่ง โดยหลักการที่ใช้ทั่วไปจะใช้ตัวจับเวลา หรือไทม์เมอร์ (Timer) ทำหน้าที่นี้ ซึ่งค่าที่ได้จะเป็นค่าตัวเลขจำนวนนับ ที่บอกถึงค่าความกว้างของแท่ง ที่จะนำไปใช้หาขนาดของแท่ง ในการเริ่มต้นหรือสิ้นสุดการนับค่าความกว้างของแท่งนั้นจะมีสัญญาณที่ไปทริก (trig) ให้ตัวไทม์เมอร์ทำการเก็บค่าและเริ่มต้นนับค่าความกว้างของแท่งถัดไป เนื่องจากหน่วยประมวลผลกลางที่เลือกใช้ในการควบคุมการทำงาน คือ ไมโครคอนโทรลเลอร์ 8031 ซึ่งมีไทม์เมอร์อยู่ภายในตัวแล้ว จึงทำให้การหาความกว้างของแท่งทำได้ง่าย และสัญญาณทริกที่จะใช้ในการเริ่มต้น หรือสิ้นสุดการนับค่าความกว้าง จะใช้การทริกที่ขาอินเทอร์รัพท์ (interrupt, INT) ซึ่งจะเกิดการอินเทอร์รัพท์ เมื่อมีสภาวะลอจิกเป็น “0” เท่านั้น แต่จากสัญญาณที่ได้จากหัวอ่านรหัสแท่งในรูปที่ 6.2 จะเห็นได้ว่าจะเกิดการอินเทอร์รัพท์เมื่อรูดหัวอ่านผ่านแท่งขาวเท่านั้น และจะเกิดอินเทอร์รัพท์ตลอดช่วงที่รูดผ่านแท่งขาว แต่สิ่งที่ต้องการในการออกแบบ คือ สภาวะปกติมีลอจิกที่ขาอินเทอร์รัพท์เป็น “1” และให้เกิดอินเทอร์รัพท์เมื่อรูดหัวอ่านจากแท่งดำเป็นแท่งขาว หรือจากแท่งขาวเป็นแท่งดำ โดยการอินเทอร์รัพท์ที่เกิดขึ้นข้างต้นจะต้องเกิดจากสัญญาณพัลส์เพียง 1 ลูกเท่านั้น (หลังจากนั้นต้องกลับสู่สภาวะปกติเพื่อเตรียมสำหรับการอินเทอร์รัพท์ครั้งต่อไป)

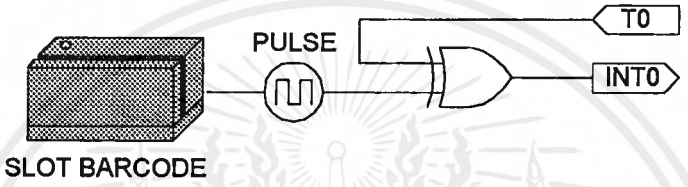


รูปที่ 6.3 โครงสร้างของส่วนอ่านรหัสแท่ง

<u>ตารางที่ 6.1</u> แสดงสภาวะที่ขาสัญญาณต่างๆ ของส่วนอ่านรหัสแท่ง

สภาวะ	ลอจิกที่ได้จากหัวอ่าน	ลอจิกที่ขาควบคุม	ลอจิกที่ขาอินเตอร์รัพท์
เริ่มต้น	0	1	1 (ไม่อินเตอร์รัพท์)
แท่งขาว เป็น แท่งดำ	1	1	0 (อินเตอร์รัพท์)
กลับสภาวะที่ขาควบคุม	1	0	1 (ไม่อินเตอร์รัพท์)
แท่งดำ เป็น แท่งขาว	0	0	0 (อินเตอร์รัพท์)

ดังนั้นจึงต้องมีการเพิ่มวงจรลอจิกขึ้นมา และให้ขาอินพุทข้างหนึ่งต่อกับขาสัญญาณที่ได้จากหัวอ่าน ส่วนขาอินพุทอีกด้านหนึ่งต่อกับขาควบคุม ดังแสดงในรูปที่ 6.3 เพื่อให้ได้สภาวะตามที่ต้องการจึงได้เลือกใช้ลอจิกเกทแบบเอกซ์คลูซีฟออร์ (exclusive-OR, XOR) ดังแสดงสภาวะต่างๆ ที่เกิดขึ้นในตารางที่ 6.1 ซึ่งจะเกิดการอินเทอร์รัพท์ เพื่อให้นำค่าความกว้างของแท่งมาเก็บไว้ และเริ่มต้นนับค่าความกว้างของแท่งต่อไป ก็ต่อเมื่อมีการรูดหัวอ่านผ่านจากแท่งดำไปยังแท่งขาว หรือจากแท่งขาวไปแท่งดำเท่านั้น โดยสภาวะในตารางจะเป็นลักษณะที่เกิดจากการสแกนของหัวอ่าน และจะเกิดขึ้นวนเวียนกันไปจนสิ้นสุดการอ่านรหัสแท่ง สำหรับวงจรที่ได้ออกแบบใช้งานจริงจะแสดงในรูปที่ 6.4 โดยเลือกใช้ไอซีทีทีแอล 74HCT86 ซึ่งเป็น XOR แบบ 2 อินพุท

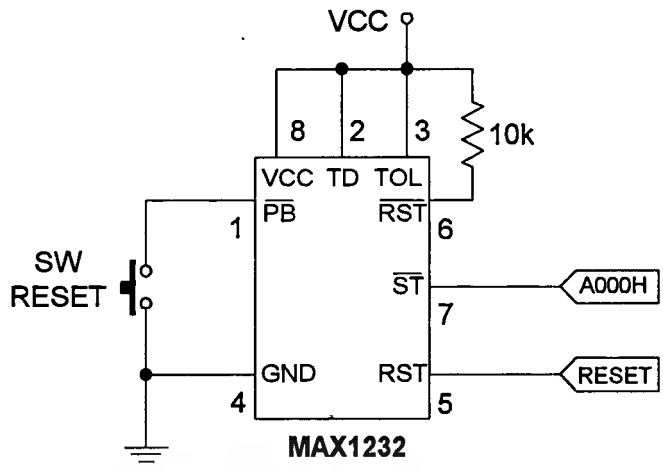


รูปที่ 6.4 วงจรส่วนอ่านรหัสแท่ง

6.1.2 โครงสร้างและการออกแบบวงจรวอทช์ด๊อก (Watch dog)

อุปกรณ์ที่ใช้ไมโครโปรเซสเซอร์ เป็นหน่วยประมวลผล และควบคุมการทำงาน จะต้องมีส่วนที่ทำหน้าที่ในการตรวจสอบการทำงานของระบบเข้ามาเกี่ยวข้องด้วยเสมอ ซึ่งวงจรส่วนนี้เรียกว่า วงจรวอทช์ด๊อก (Watch dog) โดยการทำงานของวงจรส่วนนี้จะทำการตรวจสอบสัญญาณพัลส์ที่หน่วยประมวลผลกลางส่งมายังวงจร ในช่วงเวลาที่กำหนดว่ายังมีสัญญาณมาเป็นปกติหรือไม่ ซึ่งถ้าไม่มีสัญญาณพัลส์มายังวงจรวอทช์ด๊อก มันก็จะส่งสัญญาณไปทำการรีเซตระบบ และวงจรนี้ยังสามารถตรวจสอบระดับแรงดันไฟเลี้ยงของวงจรว่ายังอยู่ในช่วงที่สามารถทำงานได้หรือไม่

ในการออกแบบได้เลือกใช้ไอซีเบอร์ MAX1232 ซึ่งเป็นไอซีวงจรวอทช์ด๊อก ดังแสดงในรูปที่ 6.5 ที่สามารถกำหนดช่วงเวลา ในการตรวจสอบการทำงานของหน่วยประมวลผลกลางได้ 3 ค่า คือ 150mS , 600 mS และ 1.2 Sec ส่วนการตรวจสอบระดับแรงดันสามารถเลือกได้ 2 ค่า คือ ตรวจจับที่ระดับแรงดันลดลงจากปกติ 5% หรือ 10% และการรีเซตระบบ ยังสามารถเลือกได้ว่า จะเป็นสัญญาณรีเซตที่สภาวะลอจิก “0” หรือ “1” รวมทั้งมีวงจร Power on Reset ซึ่งทำหน้าที่ในการสร้างสัญญาณรีเซตระบบในช่วงเริ่มต้นการทำงาน และมีขาสำหรับต่อปุ่มรีเซทได้โดยตรง



รูปที่ 6.5 วงจรวอทช์ด็อก (Watch dog)

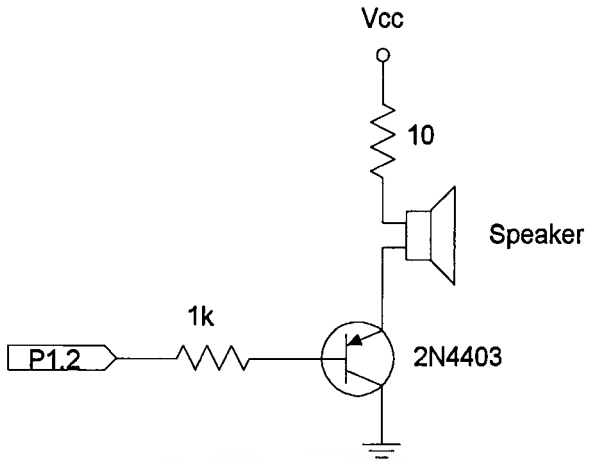
6.1.3 โครงสร้างและการออกแบบวงจรภาคแสดงผล

การออกแบบภาคแสดงผลเลือกใช้จอแสดงผลแบบผลึกเหลว หรือ LCD (Liquid Crystal Display) เนื่องจากใช้พลังงานน้อยกว่าภาคแสดงผลอื่น ๆ โดยเลือกใช้จอแสดงผล LCD รุ่น DV 16116H ขนาด 16 ตัวอักษร 1 บรรทัด (รายละเอียดอธิบายไว้ในบทที่ 4 หัวข้อ 4.6) ที่มีลักษณะการเชื่อมต่อของข้อมูลเป็นแบบขนานมีขาสัญญาณที่เกี่ยวข้องในการติดต่อกับภาคแสดงผล 14 ขา สัญญาณ โดยเป็นขาสัญญาณที่เกี่ยวข้องกับการควบคุมการทำงาน 11 ขา คือ สัญญาณควบคุม 3 ขา และข้อมูล 8 ขา ส่วนอีก 3 ขาสัญญาณ คือ ไฟเลี้ยง กราวด์ โดยสามารถเลือกใช้ขนาดของข้อมูลได้ 2 ขนาด คือ ขนาด 4 บิท และ 8 บิท

6.1.4 โครงสร้างและการออกแบบวงจรกำเนิดเสียง

การอ่านรหัสแท่งของเครื่องอ่านรหัสแท่งนี้ ถึงแม้จะมีภาคแสดงผลแบบผลึกเหลว (LCD display) ที่สามารถแสดงผลให้ทราบว่าการอ่านรหัสแท่งนั้นถูกต้องหรือไม่ แต่เนื่องจากข้อเสียของจอแอลซีดี คือมองเห็นได้ไม่ชัดเจนในบางมุมมอง จึงได้มีการเพิ่มส่วนวงจรสร้างสัญญาณเสียงในการบอกให้ทราบว่าการอ่านรหัสแท่งนั้นๆ ถูกต้องหรือไม่อย่างไร อีกทางหนึ่งด้วย

วงจรกำเนิดเสียงที่ได้ออกแบบใช้งาน เลือกใช้ลำโพงแบบ เปียโซอิเล็กทริก (Piezoelectric Speaker) ซึ่งมีขนาดเล็ก วงจรที่ใช้ขับลำโพงก็ไม่สลับซับซ้อนและใช้อุปกรณ์น้อย เนื่องจากลำโพงแบบนี้จะตอบสนองความถี่ได้ดีในช่วงที่ค่อนข้างแคบ ดังนั้นการขับลำโพงจึงได้เลือกใช้การกำเนิดสัญญาณพัลส์ หรือ สัญญาณซายน์ (sine wave) ในย่านความถี่ที่ลำโพงนี้ตอบสนองดีที่สุดก็จะได้เสียงที่ชัดเจน เพียงพอที่จะใช้ในการแสดงความถูกต้อง และผิดพลาดในการอ่านรหัสแท่ง ดังแสดงวงจรในรูปที่ 6.6



รูปที่ 6.6 วงจรกำเนิดเสียงโดยใช้ลำโพงแบบเปียโซอิเล็กทริก

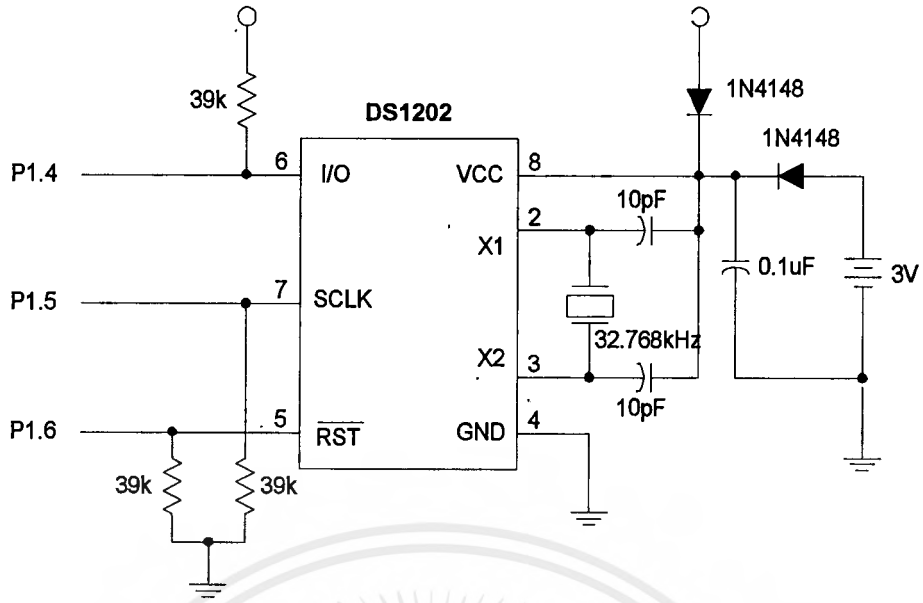
6.1.5 โครงสร้างและการออกแบบวงจรสร้างฐานเวลาจริง

ส่วนของวงจรสร้างฐานเวลาจริง (Real Time Clock , RTC) ใช้ไอซีเบอร์ DS1202 ซึ่งเป็นไอซีที่ทำหน้าที่เป็นนาฬิกาที่สามารถส่งข้อมูลเวลาในขณะใดๆ ให้แก่หน่วยประผลกลางได้ ในการติดต่อกับหน่วยประผลกลางใช้เพียง 3 ขา เท่านั้น เพราะใช้การสื่อสารแบบอนุกรม ขาที่ใช้คือ RTS (reset), I/O (data line) และ SCLK (serial clock) ดังแสดงในรูปที่ 6.7 ขา I/O ใช้เป็นขาสัญญาณข้อมูล ขา SCLK เป็นขาสัญญาณที่ใช้ควบคุมการรับ-ส่งข้อมูล และขา RTS เป็นขาสัญญาณที่เลือกการทำงานว่าจะอยู่ในสภาวะปกติ หรือ สภาวะที่พร้อมในการทำงาน

คุณสมบัติของ RTC เบอร์ DS1202 มีดังนี้

- ทำหน้าที่นับวินาที นาที ชั่วโมง วัน เดือน ปี รวมทั้งคำนวณปีอธิกสุรทินให้เองโดยอัตโนมัติ
- มีหน่วยความจำขนาด 24 ไบต์ สำหรับเก็บข้อมูลทั่ว ๆ ไป
- ใช้การติดต่อสื่อสารแบบอนุกรม จึงใช้สายเชื่อมต่อกับระบบเพียง 3 เส้น
- ใช้แรงดันไฟฟ้าเพียง 2.0 ถึง 5.5 โวลต์ และใช้กระแสเพียง 300 นาโนแอมแปร์ที่ระดับแรงดัน 2.0 โวลต์
- การโอนย้ายข้อมูลทำได้ทั้งแบบครั้งละ 1 ไบต์ หรือครั้งละ หลาย ๆ ไบต์
- ระดับสัญญาณ TTL (Vcc = 5 V)
- ช่วงอุณหภูมิในการใช้งานกว้าง ตั้งแต่ -40 ถึง 88 องศาเซลเซียส

วงจรสร้างฐานเวลาจริง จะใช้ในการส่งค่าวัน-เวลาจริง ให้กับไมโครคอนโทรลเลอร์ ณ เวลาที่มีการอ่านรหัสแท่งและถอดรหัสได้สมบูรณ์ และข้อมูลวัน-เวลานี้ จะถูกเก็บรวมกับข้อมูลจากรหัสแท่งในหน่วยความจำ เพื่อรอการส่งไปยังศูนย์กลางข้อมูลของระบบ

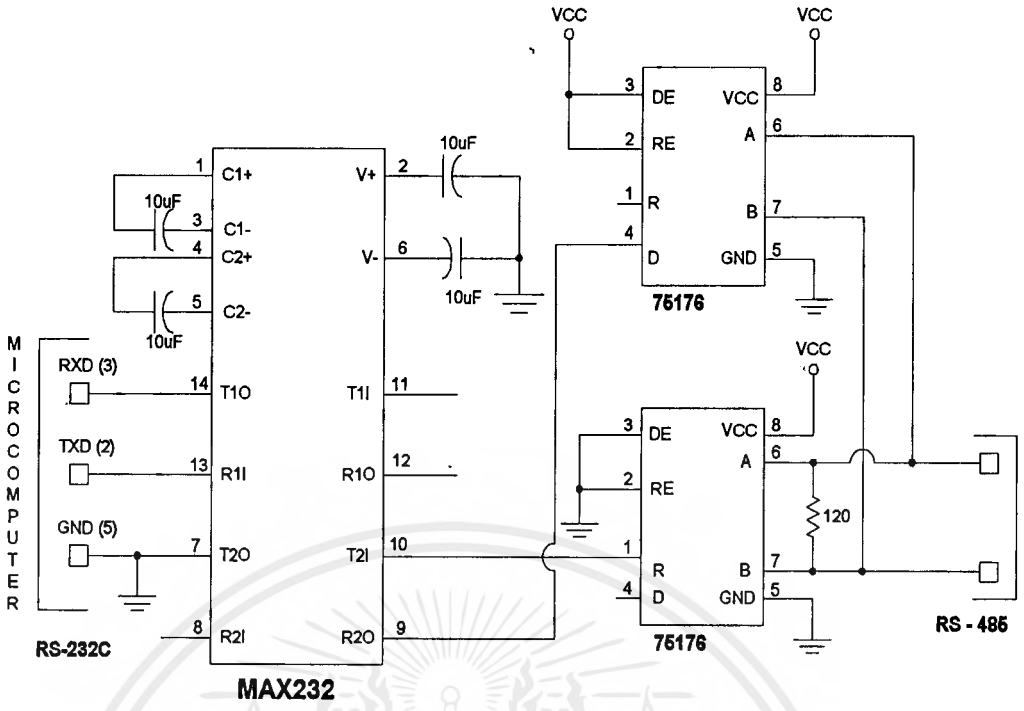


รูปที่ 6.7 วงจรสร้างฐานเวลาจริงโดยใช้ไอซีเบอร์ DS1202

6.2 ศูนย์กลางข้อมูลของระบบ

โดยโครงสร้างและหลักการของของระบบรหัสแท่งแบบเครือข่ายนั้น จะมีส่วนที่ทำหน้าที่ในการเป็นศูนย์กลางในการเก็บข้อมูลของระบบทั้งหมด ซึ่งจะต่อกันเป็นเครือข่ายเพื่อให้สามารถเก็บข้อมูลจากโหนดต่าง ๆ ที่กระจายออกไปจากศูนย์กลางข้อมูลได้สะดวกรวดเร็ว โดยใช้ไมโครคอมพิวเตอร์เป็นศูนย์กลางข้อมูล และใช้มาตรฐานการสื่อสารข้อมูลแบบหลายจุด RS-485 ที่ต่อกันเป็นเครือข่ายข้อมูลแบบบัส ซึ่งการควบคุมและจัดการเครือข่ายจะทำโดยตัวศูนย์กลางข้อมูลทั้งหมดโดยที่โหนด (ในที่นี้ คือ เครื่องอ่านรหัสแท่ง) แต่ละตัวจะทำหน้าที่ในการอ่านรหัสแท่งแล้วทำการประมวลผลจนได้ข้อมูลที่ถูกต้องจากนั้นจะรอจนกว่าศูนย์กลางข้อมูลร้องถาม จึงสามารถส่งข้อมูลเข้าไปในเครือข่ายไปยังศูนย์กลางข้อมูลของระบบได้

ในการออกแบบวงจรสำหรับส่วนนี้ เนื่องจากไมโครคอมพิวเตอร์นั้น จะมีพอร์ทสำหรับการสื่อสารข้อมูลแบบอนุกรมตามมาตรฐาน RS-232C อยู่แล้ว จึงได้สร้างวงจรแปลงสัญญาณให้เป็นไปตามมาตรฐาน RS-485 เพื่อที่จะได้สามารถต่อเป็นเครือข่ายข้อมูล วงจรที่ใช้งานจะใช้ไอซี 3 ตัว คือ IC MAX232 ที่จะทำหน้าที่ในการแปลงสัญญาณที่ได้จากพอร์ท RS-232C ของไมโครคอมพิวเตอร์ ให้เป็นสัญญาณ TTL ตามปกติ จากนั้นจะใช้ IC SN75176 จำนวน 2 ตัวทำการแปลงให้เป็นสัญญาณตามมาตรฐาน RS-422 (Full-Duplex) ดังแสดงวงจรในรูปที่ 6.8 แต่เนื่องจากต้องการใช้สายสัญญาณเพียงคู่เดียวในการส่งและรับข้อมูล ทำให้ในขณะเวลาหนึ่ง ๆ จะมีข้อมูลไปในทิศทางเดียวเท่านั้น ซึ่งเป็นการสื่อสารแบบ Half-Duplex (มาตรฐาน RS-485) จึงต้องมีการควบคุมทิศทางของข้อมูลด้วยโปรแกรมสื่อสารข้อมูล



รูปที่ 6.8 วงจรแปลง RS-232C ไปเป็น RS-485 ของศูนย์กลางข้อมูลของระบบ

6.3 ลักษณะโครงสร้างข้อมูลในเครื่องอ่านรหัสแท่ง

ลักษณะโครงสร้างของข้อมูลที่ใช้ในเครื่องอ่านรหัสแท่ง สำหรับใช้ในการส่งผ่านข้อมูลในเครือข่ายข้อมูล ระหว่างศูนย์กลางข้อมูลของระบบกับเครื่องอ่านรหัสแท่งนั้น ข้อมูลที่ใช้จะมีโครงสร้างเป็นดังรูปที่ 6.9 โดยข้อมูลแต่ละชุดประกอบด้วย 4 ส่วน คือ

1. ข้อมูลรหัสนักศึกษา ขนาด 8 ไบต์
2. ข้อมูลวัน/เดือน/ปี ขนาด 6 ไบต์
3. ข้อมูลเวลา ขนาด 4 ไบต์
4. อักขระตรวจสอบ ขนาด 1 ไบต์

โดยข้อมูลทั้งหมดจะแทนด้วยรหัสตามมาตรฐานของรหัสแอสกี้ (ASCII) และข้อมูลแต่ละชุดจะถูกแยกด้วยตัวกั้นคืออักขระ "," ขนาด 1 ไบต์

รหัสนักศึกษา STUDENT ID	วัน/เดือน/ปี DATE	เวลา TIME	อักขระตรวจสอบ CHECK SUM
8 ไบต์	6 ไบต์	4 ไบต์	1 ไบต์

รูปที่ 6.9 แสดงลักษณะชุดข้อมูลที่ใช้ส่งผ่านในระบบเครือข่ายข้อมูล

สำหรับการนำข้อมูลไปใช้งานนั้น เนื่องจากได้ออกแบบให้ใช้กับโปรแกรมประเภท xbase จึงจัดเก็บข้อมูลแบบ DBF format โดยในระบบนี้ได้เลือกใช้งานกับโปรแกรมฟอกซ์โปร (FoxPro) ซึ่งเป็นโปรแกรมจัดฐานข้อมูลที่นิยมใช้กันทั่วไป

6.4 หลักการออกแบบโปรแกรม

6.4.1 หลักการออกแบบโปรแกรมการอ่านรหัสแท่ง

โดยหลักการแล้วในการอ่านและการแทนรหัสจะอาศัยการแยกจากความแตกต่างของลักษณะความกว้างของข้อมูล ซึ่งรหัสแท่งแต่ละรูปแบบจะมีโครงสร้างและลักษณะการจัดเรียงของความกว้างของข้อมูลที่แตกต่างกัน สำหรับในส่วนของโปรแกรมในการอ่านรหัสแท่งนั้น เริ่มด้วยส่วนการหาค่าความกว้างของรหัสแท่ง และส่วนที่เป็นการถอดรหัสเพื่อหาข้อมูลที่แท้จริง

ในระบบนี้เลือกใช้รหัสแท่งเป็นแบบรหัส 39 อาศัยโครงสร้างของแท่งรหัสที่ประกอบด้วยแท่งดำที่กว้างและแคบ แท่งขาวที่กว้างและแคบ นำมาจัดเรียงต่อกันสลับดำ-ขาว เริ่มต้นด้วยแท่งดำ ไม่ว่าจะเป็นแท่งขาวหรือแท่งดำก็ตาม การคิดเป็นลอจิกจะพิจารณาจากแท่งกว้างเป็น “1” และแท่งแคบเป็น “0” รหัส 39 นั้น 1 รหัสอักขระประกอบด้วยแท่งดำ 5 แท่ง แท่งขาว 4 แท่ง รวมเป็น 9 แท่ง และเป็นแท่งกว้างทั้งหมด 3 แท่ง

ดังนั้นในการออกแบบโปรแกรมจึงใช้ตัวจับเวลา (TIMER) ภายในไมโครคอนโทรลเลอร์นับค่าความกว้างของแท่งที่อ่านเข้ามาได้ ทำการนับในลักษณะเดียวกันไม่ว่าจะเป็นแท่งขาวหรือดำ เนื่องจากหัวอ่านที่ใช้เป็นแบบ Slot scanner ซึ่งเป็นหัวอ่านแบบลำแสงคงที่ (Fix Beam) อัตราเร็วในการลากรหัสแท่งผ่านหัวอ่าน จึงเป็นสิ่งที่ต้องนำมาพิจารณาประกอบกับการออกแบบโปรแกรมด้วย การเปรียบเทียบความกว้างและแคบของสัญญาณที่อ่านได้ จะเปรียบเทียบกันในโมดูลเดียวเท่านั้น (1 Module = 1 รหัสอักขระ) เพื่อลดข้อผิดพลาดที่เกิดจากการใช้ความเร็วไม่คงที่

ในการกำหนดว่าแท่งใดกว้างหรือแคบ ต้องพิจารณาเปรียบเทียบกับค่ากำหนด (Threshold , TH) ค่าหนึ่งในแต่ละโมดูล ค่ากำหนดนี้จะหาจากแท่งกว้างและแท่งแคบในโมดูลที่พิจารณาอยู่ โดยแยกแท่งดำกับแท่งขาวออกจากกัน แล้วเลือกแท่งที่กว้างที่สุด (Max) กับแท่งที่แคบที่สุด (Min) นำค่าทั้งสองนี้มาเฉลี่ย (TH = [Max+Min] / 2) แท่งรหัสใดที่มีค่ามากกว่าค่ากำหนด (TH) ก็ให้แท่งนั้นเป็นลอจิก “1” ถ้าน้อยกว่าให้เป็นลอจิก “0”

ในการเขียนโปรแกรมต้องกำหนดโหมดต่าง ๆ ในการอินเตอร์รัพท์ ดังนี้

TIMER 1 โหมด 1 เป็นการควบคุมภายใน

TIMER 0 โหมด 1 เป็นการควบคุมภายนอก

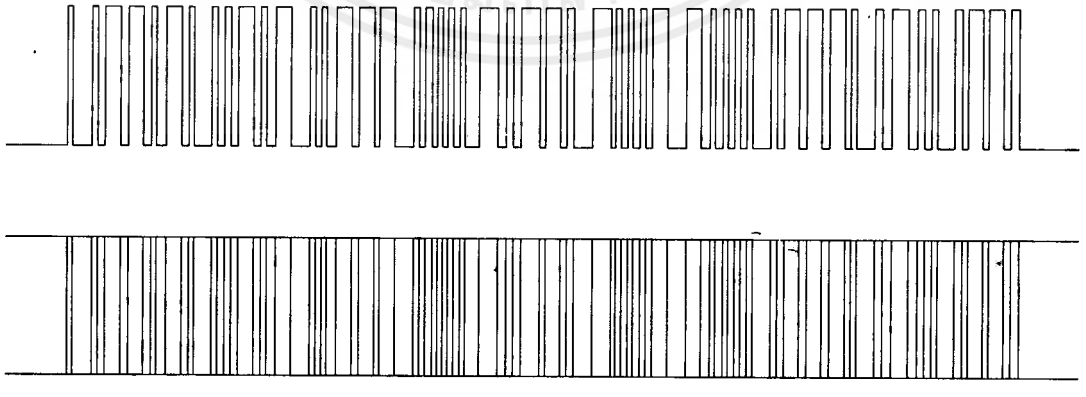
INTERRUPT 0 เป็นการควบคุมแบบทริกขอบ (edge trig)

เพื่อให้ TIMER 0 นับค่าความกว้างของแท่งได้ จะต้องแปลงสัญญาณจากหัวอ่านให้เป็นดังรูปที่ 6.10 นอกจากนี้ยังต้องมีการชดเชยเวลา เนื่องจากการเปลี่ยนสถานะของสัญญาณเอาท์พุทที่หัวอ่าน ด้วยการตั้งค่าฐานเวลาในการนับที่ TIMER 0

เพื่อให้แน่ใจว่ามีการใช้งานจริง จึงต้องมีการตรวจสอบพื้นที่ขอบเผื่อ (Quiet zone) ก่อนที่จะถึงแท่งดำแท่งแรก โดยใช้ TIMER 0 ส่วน TIMER 1 ใช้ตรวจสอบความผิดพลาดในค่าเวลาของแท่งรหัส เพราะอาจจะเกิดการกระตุก หรือการลากที่เร็วหรือช้าเกินไปในบางช่วง

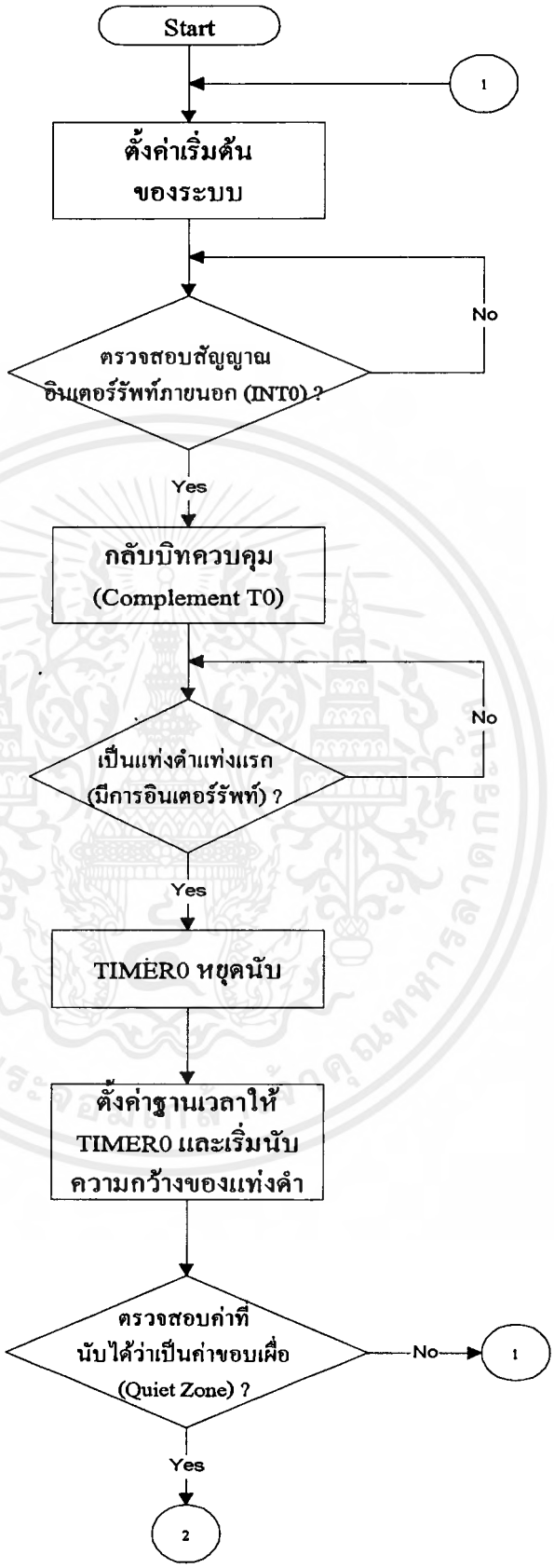
การตั้งค่าฐานเวลาในการนับของ TIMER 0 จะพิจารณาจากค่าเวลาของแท่งแคบซึ่งจะต้องไม่น้อยกว่า 70 mS มิฉะนั้นแล้ว TIMER 0 จะเกิดโอเวอร์โฟลว์ (overflow) เพื่อความสะดวกในการเขียนโปรแกรมถอดรหัส จึงตั้งค่าฐานเวลาของ TIMER 1 ไว้ที่ค่าเวลาเดียวกันนี้ สำหรับการตรวจสอบขนาดของแท่งรหัส ต้องมีค่าความกว้างอย่างน้อยที่สุดเท่ากับค่าฐานเวลา ถ้าน้อยกว่านี้เท่าไร การผิดพลาดที่จะเกิดจาก TIMER 1 ก็จะเกิดขึ้นได้มากเท่านั้น การโอเวอร์โฟลว์จาก TIMER 0 และ TIMER 1 จะเป็นการออกไปสู่การถอดรหัส

โปรแกรมการถอดรหัสจะเปรียบเทียบค่าที่นับได้ของแท่งรหัสในโมดูลเดียวกัน โดยแยกพิจารณาแท่งขาวและแท่งดำออกจากกัน หาแท่งดำที่แคบที่สุดและกว้างที่สุด กับแท่งขาวที่กว้างที่สุดและ แคบที่สุด ค่ากำหนด (TH) ก็คิดแยกกัน จากนั้นทำการเปรียบเทียบระหว่างแท่งรหัสกับค่ากำหนดภายในโมดูลนั้น เพื่อเปลี่ยนเป็นลอจิก 0 และ 1 ตามรูปแบบการเข้ารหัส 39 สามารถที่จะถอดรหัสจากรหัสเลขฐานสองเหล่านี้เป็นรหัสแอสกี้ได้โดยสร้างตารางการถอดรหัสขึ้นมา และจะมีเพียงอักขระเดียวเท่านั้นที่จะตรงกับโมดูลรหัสแท่งที่อ่านได้ จากรูปแบบของรหัส 39 ใช้อักขระ “*” เป็นรหัสเริ่มต้นและรหัสหยุด จึงต้องตรวจสอบอักขระ “ * “ ก่อนที่จะถอดรหัสตัวต่อไปและออกจากการถอดรหัสด้วยอักขระ “ * “ เช่นกัน

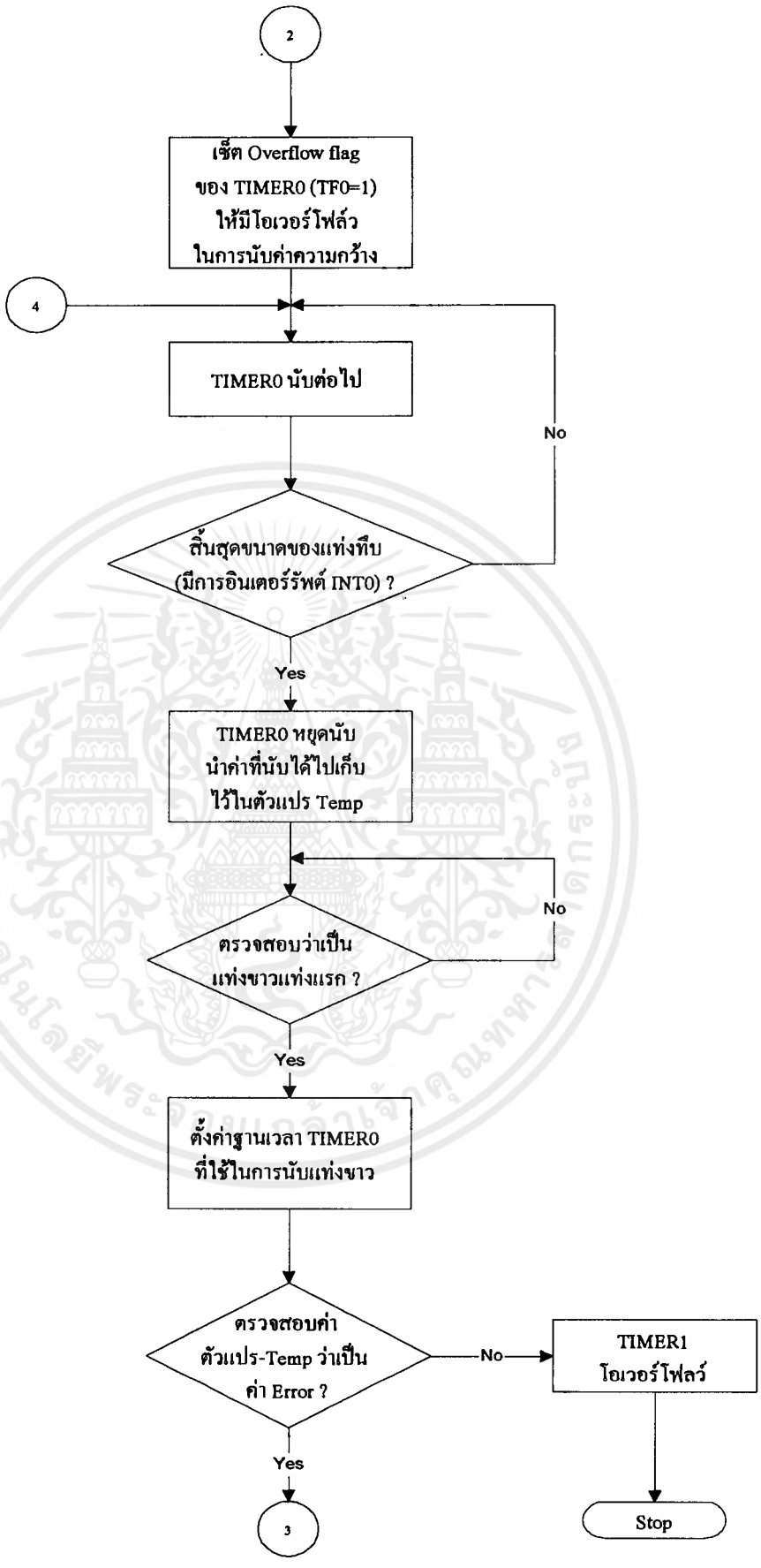


รูปที่ 6.10 สัญญาณที่ใช้ในการอินเตอรรัพท์

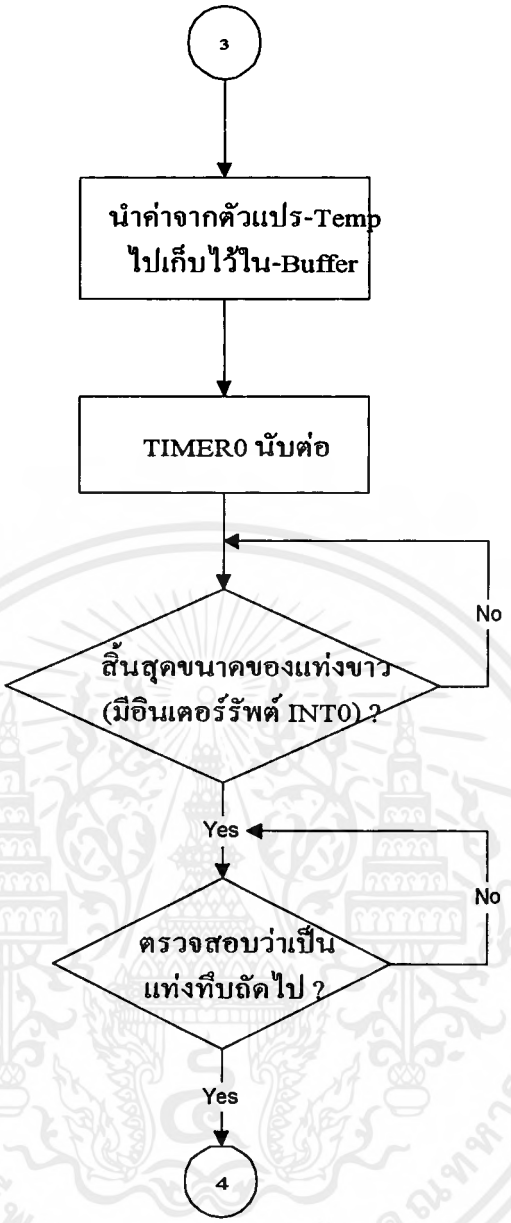
ผังโปรแกรมการวัดความกว้างของรหัสแท่ง



รูปที่ 6.11 ผังโปรแกรมการวัดความกว้างของรหัสแท่ง

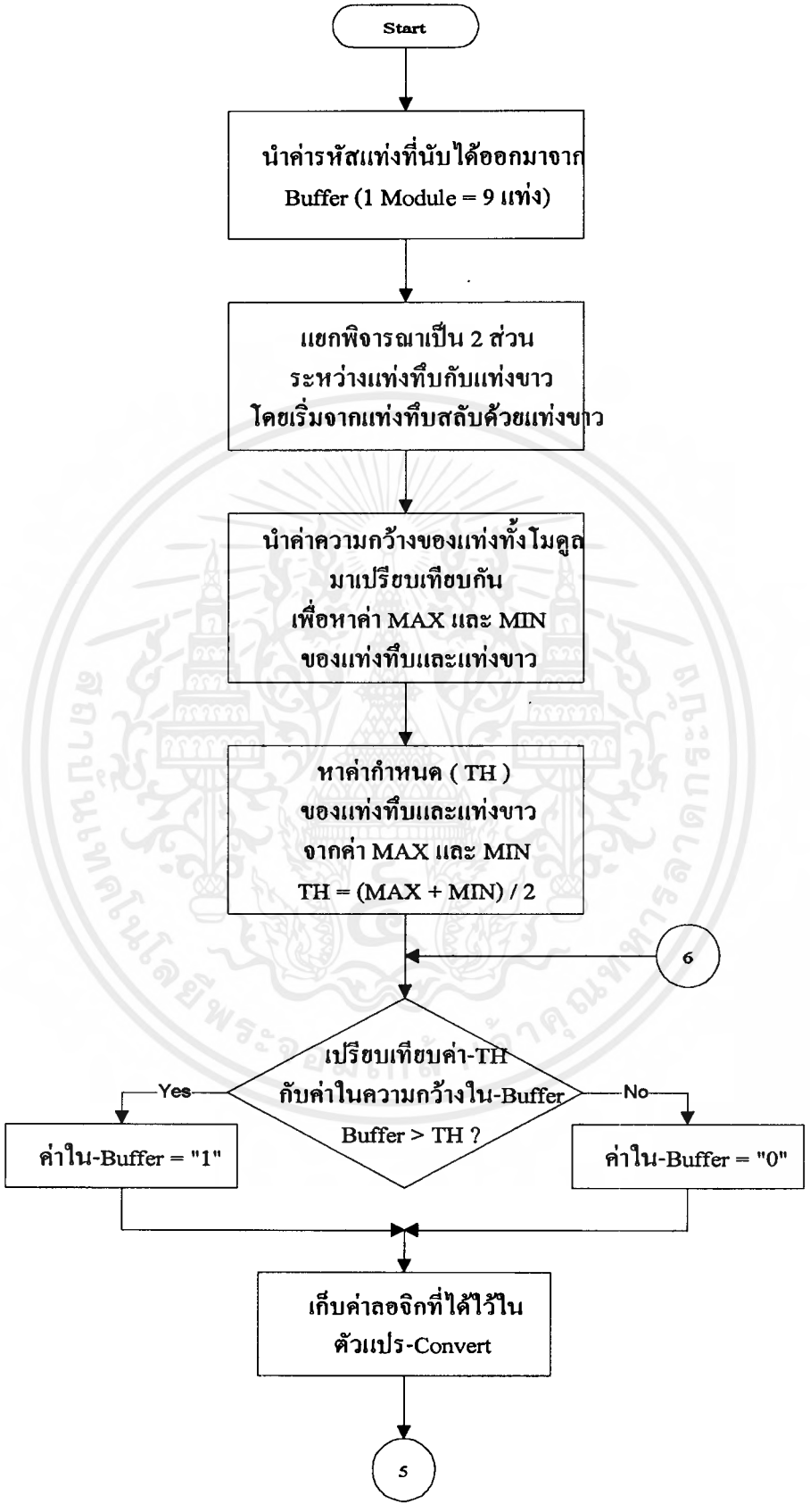


รูปที่ 6.12 ผังโปรแกรมการวัดความกว้างของรหัสแท่ง (ต่อ)

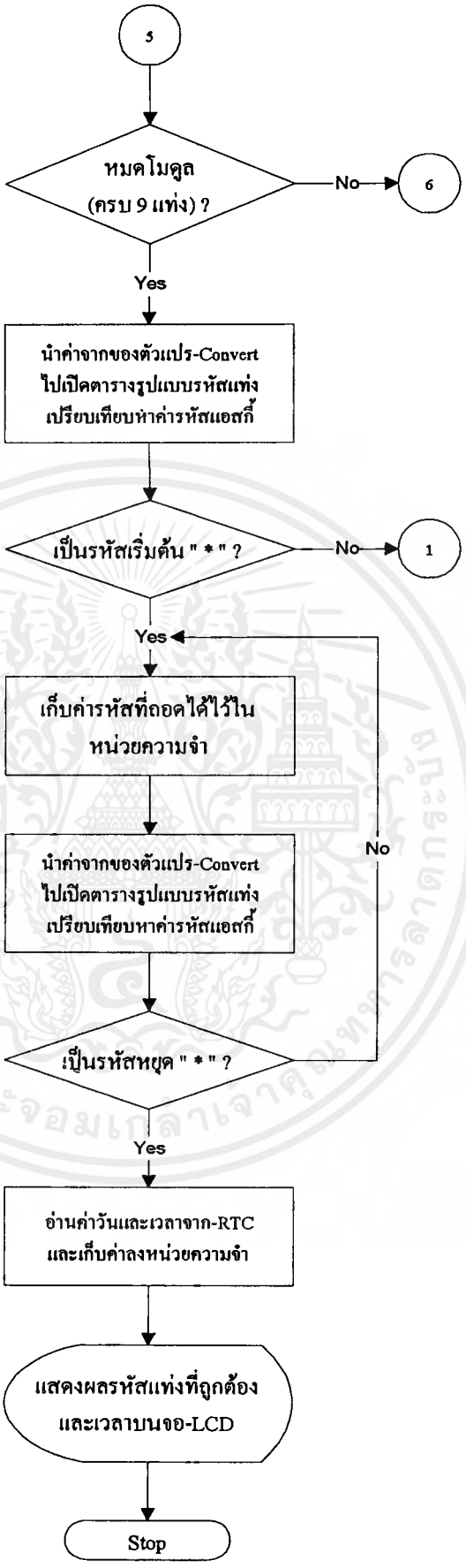


รูปที่ 6.13 ผังโปรแกรมการวัดความกว้างของรหัสแท่ง (ต่อ)

ผังโปรแกรมการถอดรหัสแท่ง

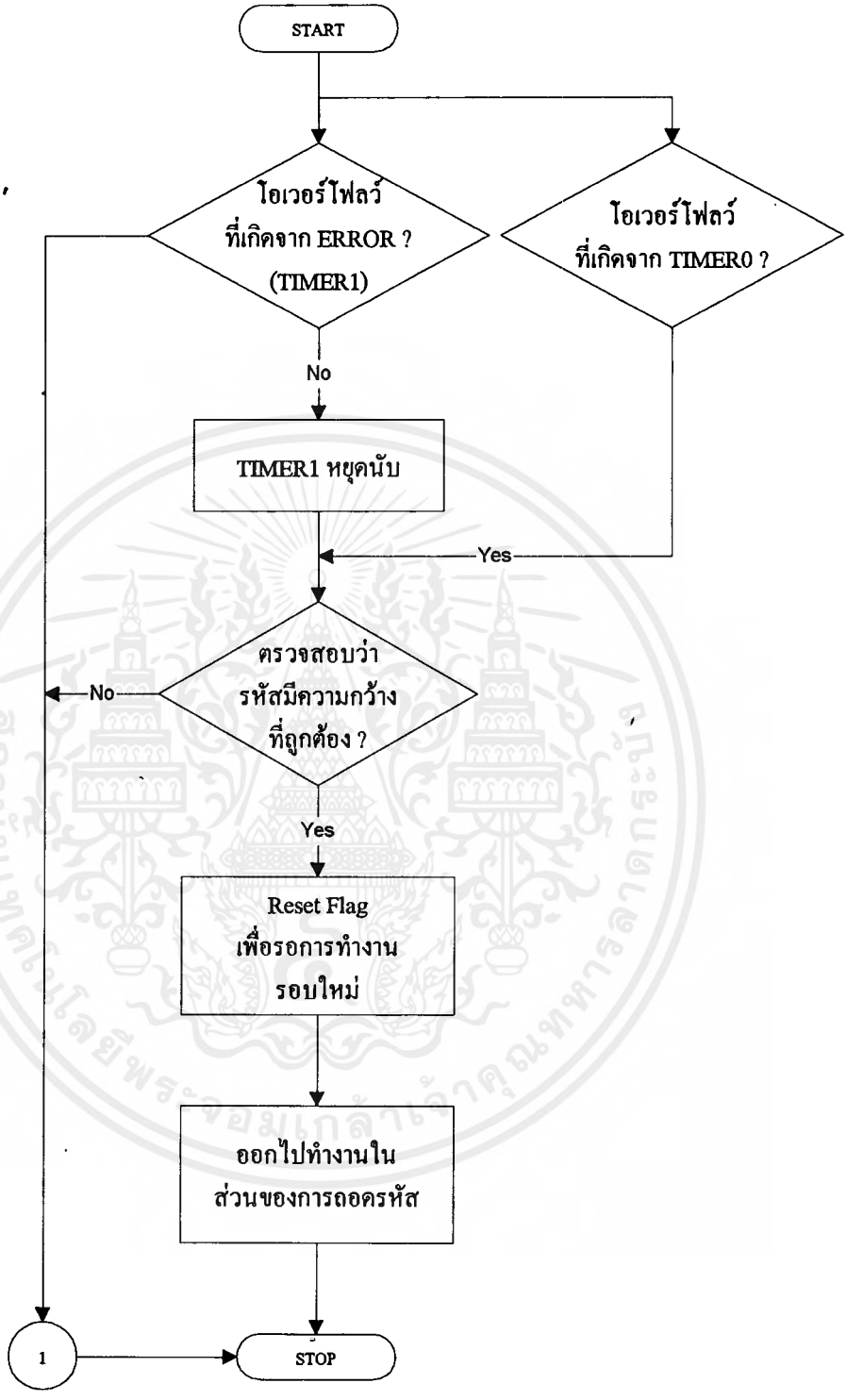


รูปที่ 6.14 ผังโปรแกรมการถอดรหัสแท่ง



รูปที่ 6.15 ผังโปรแกรมการถอดรหัสแท่ง (ต่อ)

ผังโปรแกรมย่อยบริการอินเตอร์รัพท์



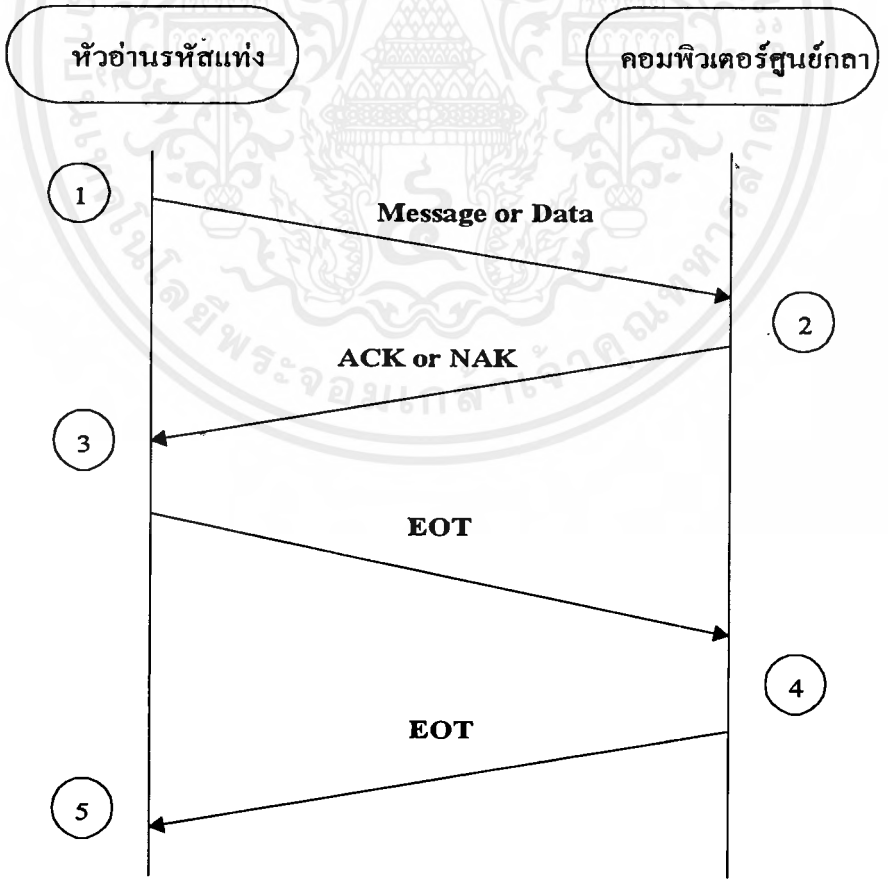
รูปที่ 6.16 ผังโปรแกรมย่อยบริการอินเตอร์รัพท์

6.4.2 หลักการออกแบบโปรแกรมควบคุมการเชื่อมต่อและสื่อสารข้อมูล

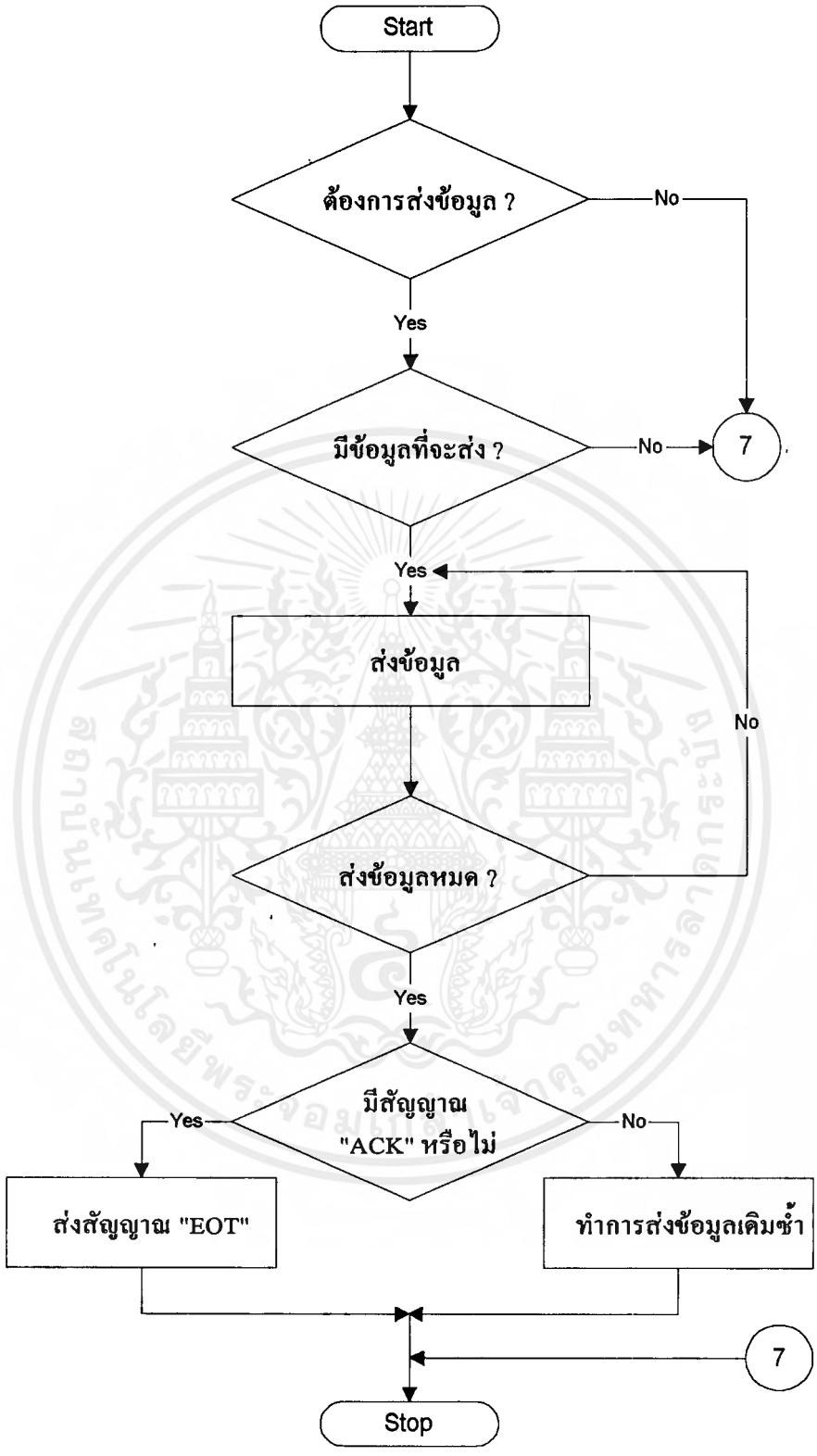
สำหรับหลักการในส่วนของการเชื่อมต่อและสื่อสารข้อมูลนั้น ถ้าเป็นการสื่อสารข้อมูลที่ใช้มาตรฐานเดียวกันจะมีโครงสร้างและโปรแกรมควบคุมการทำงานที่เหมือนกัน โดยในระบบนี้จะมีส่วนสำคัญดังนี้

โปรแกรมควบคุมการสื่อสารข้อมูลจุดต่อจุด

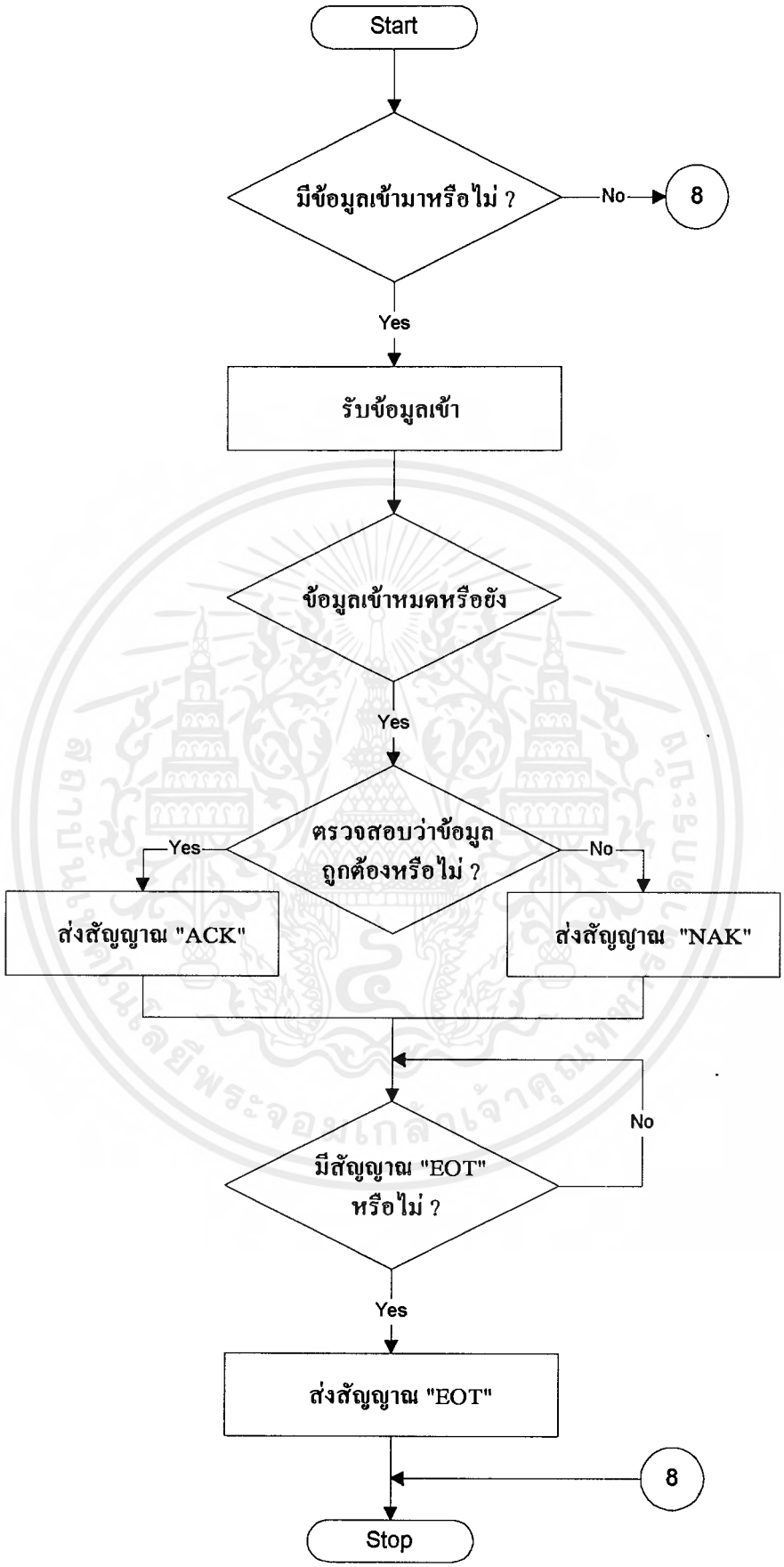
ในส่วนของการสื่อสารข้อมูลจากแบบจุดต่อจุดสำหรับระบบนี้จะเป็นการสื่อสารข้อมูลตามมาตรฐาน RS-232C ซึ่งเป็นการเชื่อมต่อระหว่างเครื่องอ่านรหัสแท่งกับคอมพิวเตอร์ศูนย์กลางข้อมูล โดยที่ทั้งสองส่วนนี้จะมีโครงสร้างและลำดับขั้นในการเชื่อมต่อและสื่อสารข้อมูลที่สอดคล้องกัน และยังมีการกำหนดโปรโตคอลในการสื่อสารข้อมูลระหว่างกันไว้ด้วย ดังแสดงในรูปที่ 6.15 ซึ่งเป็นโปรโตคอลพื้นฐาน ที่จะมีเพียงการตรวจสอบว่ารหัสตรวจสอบข้อมูล (ซึ่งเป็นรหัสที่ได้จากการนำข้อมูลทั้งหมดไปทำการประมวลผลด้วยวิธีที่กำหนดไว้) ว่าถูกต้องหรือไม่ ถ้าถูกต้องให้ส่งรหัส “ACK” แต่ถ้าไม่ถูกต้องให้ส่งรหัส “NAK” ซึ่งจะทำให้เครื่องอ่านรหัสแท่งต้องทำการส่งข้อมูลชุดเดิมซ้ำอีกครั้ง สำหรับโครงสร้างการทำงานของแต่ละส่วนแสดงในรูปที่ 6.16 และ รูปที่ 6.17



รูปที่ 6.17 แสดงโปรโตคอลในการเชื่อมต่อและสื่อสารข้อมูลแบบจุดต่อจุด



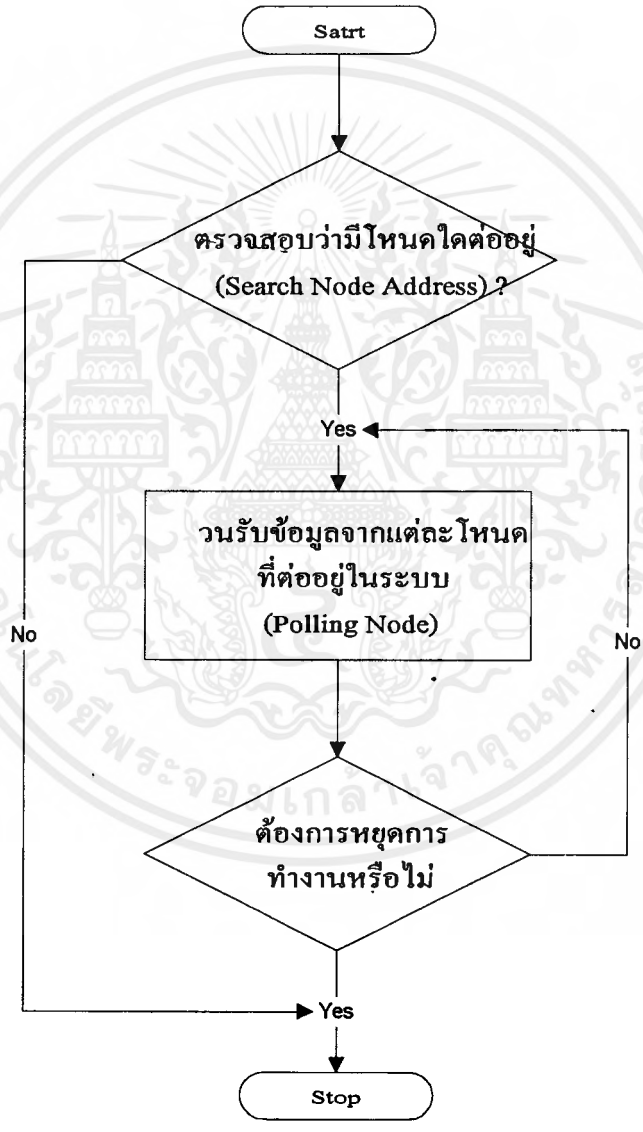
รูปที่ 6.18 ผังงานแสดงโครงสร้างการสื่อสารข้อมูลของเครื่องอ่านรหัสแท่งแบบจุดต่อจุด



รูปที่ 6.19 ผังงานแสดงการสื่อสารข้อมูลของคอมพิวเตอร์ศูนย์กลางข้อมูลแบบจุดต่อจุด

โปรแกรมควบคุมการสื่อสารข้อมูลเป็นเครือข่าย

ในส่วนของการสื่อสารข้อมูลแบบเป็นเครือข่ายที่ใช้ในระบบนี้ จะเป็นการสื่อสารข้อมูลตามมาตราฐาน RS-485 ซึ่งเป็นการสื่อสารข้อมูลในลักษณะเครือข่ายข้อมูล ระหว่างคอมพิวเตอร์ศูนย์กลางข้อมูลกับเครื่องอ่านรหัสแท่งเช่นเดียวกัน โดยโครงสร้างและลำดับขั้นตอนการทำงานจะแสดงดังในรูปที่ 6.18 ซึ่งเป็นการทำงานของคอมพิวเตอร์ศูนย์กลางข้อมูล และได้มีการกำหนดโปโตรคอลในการสื่อสารข้อมูลระหว่างกัน โดยจะมี 2 ส่วน คือส่วนของการค้นหาว่ามีโหนด หรือต่ออยู่กี่ตัว และที่แอดเดรสใดบ้างดังแสดงในรูปที่ 6.19



รูปที่ 6.20 ผังงานแสดงโครงสร้างการทำงานเป็นเครือข่ายข้อมูล

โดยลำดับการทำงานในการค้นหาว่ามีโหนดต่ออยู่เท่าใด และอยู่ที่แอดเดรสใด หรือการ Search node ดังในรูปที่ 6.19

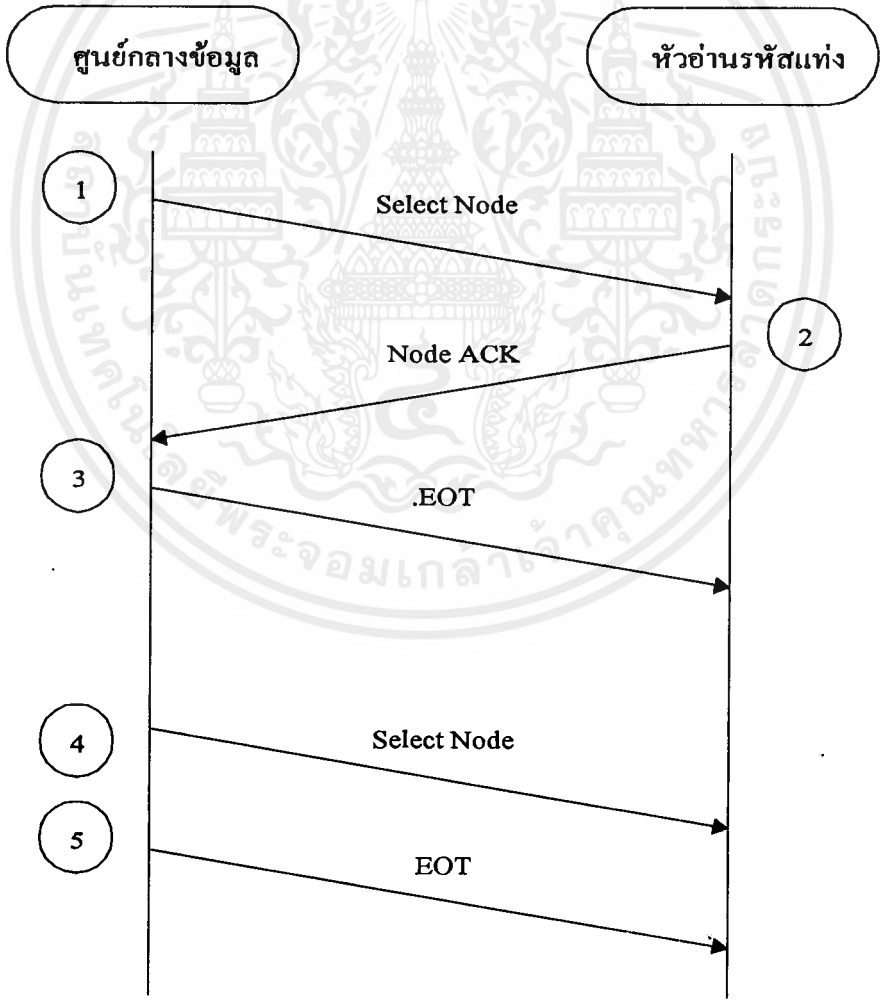
ในขั้นตอนที่ 1 เป็นการส่งรหัสเพื่อไปตรวจสอบโหนดแต่ละตัว

ในขั้นตอนที่ 2 ถ้ามีโหนดต่ออยู่ที่แอดเดรสใดก็จะมีรหัสตอบจากโหนดนั้น

ในขั้นตอนที่ 3 เมื่อได้รับสัญญาณตอบสนองก็จะส่งรหัส EOT เพื่อเป็นการสิ้นสุดการติดต่อ และนำค่าแอดเดรสที่ตอบสนองนี้ไปเก็บในตารางที่แสดงว่ามีโหนดแอดเดรสใดต่ออยู่เพื่อใช้ในการวนรับข้อมูลในส่วนต่อไป

ในขั้นตอนที่ 4 เป็นการส่งรหัสเพื่อตรวจสอบโหนดถัดไป

ในขั้นตอนที่ 5 เป็นกรณีที่แอดเดรสที่กำลังทำการตรวจสอบอยู่แล้วไม่มีการตอบสนองกลับภายในระยะเวลาที่กำหนด (หรือ Time out) ก็จะส่งรหัส EOT เพื่อเป็นการสิ้นสุดการติดต่อ และไปตรวจสอบโหนดแอดเดรสถัดไป



รูปที่ 6.21 แสดงขั้นตอนของโปรโตคอลในการค้นหาโหนด

6.4.3 หลักการออกแบบโปรแกรมจัดการข้อมูล

โปรแกรมจัดการข้อมูลบนเครื่องไมโครคอมพิวเตอร์ที่ใช้เป็นศูนย์กลางข้อมูล จะทำหน้าที่ประมวลผลข้อมูลที่รับมาจากเครื่องอ่านรหัสแท่งบันทึกเวลาานทางพอร์ตอนุกรม RS-232C ในการออกแบบโปรแกรมได้ทำการจัดแบ่งโปรแกรมจัดการข้อมูลออกเป็นส่วนๆ ดังนี้

1. โปรแกรมรับข้อมูลจากแบบเครือข่าย (RS-485)
2. โปรแกรมรับข้อมูลจากแบบจุดต่อจุด (RS-232C)
3. โปรแกรมตั้งเวลาของเครื่องบันทึกเวลา
4. โปรแกรมแสดงรายงานการลงเวลาของนักศึกษา
5. โปรแกรมสร้างและแก้ไขแฟ้มข้อมูลนักศึกษา

6.4.3.1 โปรแกรมรับข้อมูลแบบเครือข่าย

โปรแกรมส่วนนี้ได้ทำการออกแบบให้รับข้อมูลจากการเชื่อมต่อแบบเครือข่าย โดยการเรียกไปยังโหนดแต่ละโหนดที่ต่ออยู่กับระบบ เมื่อมีการตอบกลับมาก็จะรอรับข้อมูลจากโหนดที่ตอบกลับมานั้นทีละโหนด และเก็บข้อมูลไว้ในแฟ้มข้อมูล Temp1 เมื่อรับข้อมูลครบทุกโหนดแล้วจึงทำการโอนถ่ายข้อมูลมายังแฟ้มข้อมูล Temp2 เพื่อจัดรูปแบบตามที่ต้องการใช้งานก่อนที่จะถ่ายข้อมูลไปยังแฟ้มข้อมูล DateTime ซึ่งเป็น Transation file อีกทีหนึ่ง ในการเขียนโปรแกรมได้ใช้ Library ของ CompuSolveซึ่งใช้ร่วมกับFoxProทำให้การเขียนโปรแกรมส่วนติดต่อสื่อสารทำได้ง่ายขึ้นโดยใช้ได้เหมือนกับเป็นคำสั่งมาตรฐานของ FoxPro เอง

6.4.3.2 โปรแกรมรับข้อมูลแบบจุดต่อจุด

โปรแกรมส่วนนี้ได้ทำการออกแบบและเขียนเป็นภาษาแอสเซมบลี้ (8086) ให้ทำงานร่วมกับ FoxPro เพื่อทำการรับข้อมูลจากเครื่องอ่านรหัสแท่งบันทึกเวลาผ่านทางพอร์ตสื่อสารอนุกรม RS- 232C โดยตรงแบบจุดต่อจุด

ลำดับขั้นตอนการทำงานของโปรแกรมมีดังนี้

- ทำการส่งสัญญาณ NAK ไปยังเครื่องอ่านรหัสแท่งบันทึกเวลา ซึ่งสัญญาณนี้จะไปอินเตอร์รัพท์ให้ทำการส่งข้อมูลมาให้
- ทำการรับข้อมูลด้วยอัตราบอดเรต 9600 โดยอัตราในการส่งข้อมูลของเครื่องอ่านรหัสแท่งบันทึกเวลาจะเท่ากับ 9600 บอด เช่นกัน
- ทำการตรวจสอบข้อมูลว่าถูกต้องหรือไม่ ถ้าไม่จะส่งสัญญาณ NAK ไปอีกครั้ง
- ทำการเก็บข้อมูลที่ถูกต้องลงในแฟ้มข้อมูลชั่วคราว Temp
- ทำการตรวจสอบว่าข้อมูลหมดหรือยัง ถ้ายังจะกลับไปเริ่มต้นรับข้อมูลใหม่

-รับข้อมูลจนเสร็จสิ้นสมบูรณ์แล้วจะการโอนถ่ายข้อมูลจากแฟ้มข้อมูลชั่วคราวTempไปยังแฟ้มข้อมูลทรานแซคชั่น DateTime

- ทำการปิดแฟ้มข้อมูลทั้งหมดเมื่อจบการทำงาน

6.4.3.3 โปรแกรมตั้งเวลาของเครื่องบันทึกเวลา

การออกแบบโปรแกรมส่วนนี้มีจุดประสงค์เพื่อใช้ตั้ง เวลา, วัน, เดือน และปี ของเครื่องอ่านรหัสแท่งบันทึกเวลา โดยโปรแกรมส่วนนี้ได้ใช้ภาษาแอสแซมบลี้ (8086) ซึ่งสามารถ link กับโปรแกรม FoxPro ได้ การส่งผ่านค่าพารามิเตอร์ใช้การส่งผ่านทางรีจิสเตอร์ BX

ลำดับขั้นการทำงานมีดังนี้

- รับค่าเวลา และ วัน/เดือน/ปี เก็บไว้ในตัวแปรเป็น ASCII ขนาด 10 ไบท์
- ทำการ Initial series port
- ส่งค่ารหัสเพื่อบอกเครื่องอ่านรหัสแท่งบันทึกเวลาว่าจะส่งค่าการตั้งเวลาให้
- ส่งค่าวัน-เวลาพร้อมรหัสตรวจสอบไปให้กับเครื่องอ่านรหัสแท่งบันทึกเวลา
- รอรับรหัสตรวจสอบว่าการส่งสมบูรณ์หรือไม่
- ถ้าไม่สมบูรณ์จะแสดงข้อความว่าเกิดการผิดพลาดขึ้นในการติดต่อ
- จบการทำงาน คืนค่า Stack ให้กับ FoxPro

วันที่	เดือน	ปี	ชั่วโมง	นาที	รหัสตรวจสอบ
2 ไบท์	2 ไบท์	2 ไบท์	2 ไบท์	2 ไบท์	1 ไบท์

รูปที่ 6.22 แสดงรูปแบบของข้อมูลเวลา

6.4.3.4 โปรแกรมแสดงรายงานการลงเวลาของนักศึกษา

ในส่วนนี้ได้ทำการออกแบบ ให้นำเอาข้อมูลที่ได้จากเครื่องอ่านรหัสแท่งบันทึกเวลามาประมวลผล และแสดงให้ผู้ใช้ทราบถึงการเข้าห้องปฏิบัติการของนักศึกษา โดยสามารถเลือกการแสดงผลได้ทั้งทางจอแสดงผล และทางเครื่องพิมพ์

ลำดับขั้นของโปรแกรมมีดังนี้

- เปิดแฟ้มข้อมูลประจำวัน(DateTime) และแฟ้มข้อมูลนักศึกษา (Student)
- ให้เลือกประเภทของรายงานที่ต้องการแสดง ได้แก่ รายงานประจำวัน สรุปรายงานประจำเดือน และรายงานสำหรับนักศึกษาเฉพาะราย
- ให้เลือกการแสดงผลทางจอแสดงผลหรือทางเครื่องพิมพ์

- ทำการประมวลผลข้อมูลจากแฟ้มข้อมูลทั้ง 2 แฟ้มข้างต้น ตามที่ได้เลือกประเภทเอาไว้ และนำรายงานที่ได้มาแสดงผล

- เมื่อจบการทำงานจะทำการปิดแฟ้มข้อมูลที่ใช้อยู่ทั้งหมด

รหัสของนักศึกษา	วันที่มา	เวลาเข้า	เวลาออก
37013258	10/04/39	08.30	16.00
37013257	11/04/39	08.22	
37013280	11/04/39	08.00	16.30
37013279	11/04/39	15.00	16.40

รูปที่ 6.23 แสดงรูปแบบของรายงานประจำวัน

6.4.3.5 โปรแกรมสร้างและแก้ไขแฟ้มข้อมูลนักศึกษา

โปรแกรมส่วนนี้ได้ออกแบบให้เป็นส่วนที่ใช้ในการเพิ่มเติม, แก้ไข และลบข้อมูลในแฟ้มข้อมูลนักศึกษาที่ใช้เป็นแฟ้มข้อมูลหลักสำหรับโปรแกรมจัดการข้อมูล โดยใช้รหัสนักศึกษาเป็น key ในการค้นหาข้อมูลนักศึกษาที่ต้องการเพิ่มเติม, แก้ไข และลบข้อมูล ดังนี้

- ถ้าค้นหารหัสนักศึกษาไม่พบ จะเป็นการเพิ่มข้อมูลใหม่

- ถ้าค้นหารหัสนักศึกษาพบ จะนำข้อมูลที่มีอยู่มาแสดง เพื่อให้แก้ไข หรือ ลบข้อมูลนั้นจากแฟ้มข้อมูล

บทที่ 7

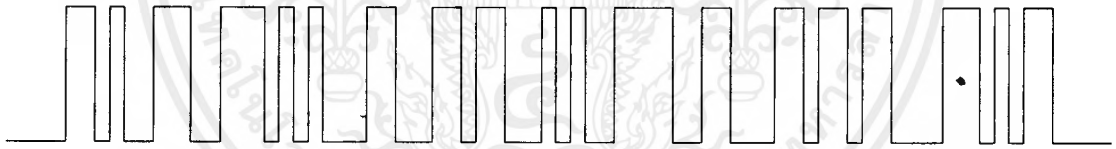
การทดลองและผลการทดลอง

ในการทดสอบการทำงานของระบบนี้ได้แบ่งการทดสอบออกเป็น 2 ส่วน ดังนี้

1. หัวอ่านรหัสแท่ง
2. เครื่องอ่น รหัสแท่งบันทึกเวลา

7.1 หัวอ่านรหัสแท่ง

ทดสอบการอ่านรหัสแท่งโดยใช้ Digital Storage Scope วัดสัญญาณที่เอาท์พุทของหัวอ่าน แล้วนำบัตรทดสอบรหัสแท่งที่เป็นรหัส 39 (ข้อมูลบนบัตร คือ "150") มาทำการรูดบัตรทดสอบจะได้รูปสัญญาณดังรูปที่ 7.1 สัญญาณที่ได้จะมี 2 สภาวะ คือ "0" และ "1" โดยเมื่อหัวอ่านสแกน ผ่านแท่งดำก็จะได้สัญญาณลอจิกเป็น "1" และเมื่อสแกนผ่านแท่งขาวจะได้เป็นลอจิก "0"



รูปที่ 7.1 แสดงสัญญาณที่ได้จาก Digital Storage Scope

ผลการวัดความเร็วในการรูดบัตร

ได้ทำการวัดความเร็วที่ใช้ในการรูดบัตรผ่านหัวอ่าน โดยใช้บัตรนักศึกษาของสถาบัน ฯ ซึ่งใช้รหัส 39 ที่มีขนาดของความยาวใน 1 รหัสอักขระเท่ากับ 5 มิลลิเมตร ดังแสดงรูปที่ 7.2 โดยพิจารณาที่ความเร็วที่สามารถถอดรหัสได้ ดังนี้

รูปที่ 7.2 รหัส 39 ที่ใช้อยู่บนบัตรนักศึกษา

ความเร็วในการรูดบัตรที่สามารถถอดรหัสได้สมบูรณ์

ความเร็วในการรูดบัตรอย่างช้า = 110 มิลลิเมตร/วินาที

ความเร็วในการรูดบัตรอย่างเร็ว = 450 มิลลิเมตร/วินาที

ความเร็วในการรูดบัตรโดยเฉลี่ย = 280 มิลลิเมตร/วินาที

7.2 เครื่องอ่านรหัสแท่งบันทึกเวลา

เครื่องอ่านรหัสแท่งบันทึกเวลา ทำหน้าที่ถอดรหัสแท่งจากบัตรนักศึกษา และบันทึกเวลาของนักศึกษาที่จะเข้าห้องปฏิบัติการ วิธีการทดสอบทำโดย

1. ป้อนไฟเข้าเครื่องอ่านรหัสแท่งบันทึกเวลา แล้วเชื่อมต่อสายกับพอร์ตอนุกรมมาตรฐาน RS-232C กรณีที่สถานที่ที่ติดตั้งใช้งานอยู่ไม่ห่างจากคอมพิวเตอร์ศูนย์กลางข้อมูล (ไม่เกิน 15 เมตร) หรือต่อสายกับพอร์ตอนุกรมมาตรฐาน RS-485 ในกรณีที่สถานที่ที่ติดตั้งใช้งานอยู่ไกลจากคอมพิวเตอร์ศูนย์กลางข้อมูล (ประมาณ 15 - 100 เมตร)

2. ทำการตั้ง วัน/เดือน/ปี และ เวลา (ชั่วโมง:นาที) บนเครื่องอ่านรหัสแท่งบันทึกเวลาด้วยคอมพิวเตอร์ศูนย์กลางข้อมูล (รายการที่ 3 บนโปรแกรมจัดการข้อมูล)

3. เมื่อเครื่องอ่านรหัสแท่งบันทึกเวลาแสดงวันและเวลาที่ถูกต้อง (ดูผลที่จอ LCD) ตรงกับวันและเวลาบนคอมพิวเตอร์ศูนย์กลางข้อมูล แสดงว่าเครื่องพร้อมที่จะให้รูดบัตรและบันทึกเวลาที่ถูกต้องได้

4. ทดสอบการถอดรหัสแท่งโดยใช้บัตรนักศึกษารหัส 37013280 ที่จอแสดงผล LCD ของเครื่องอ่านรหัสแท่งบันทึกเวลา จะแสดงรหัสนักศึกษาของบัตรที่รูด คือ 37013280 และเวลาขณะที่รูดบัตรเสร็จ ดังแสดงในรูปที่ 7.3 พร้อมทั้งส่งเสียงบี๊บสั้น ๆ 1 ครั้ง ซึ่งได้ผลถูกต้องตามที่ได้ออกแบบไว้

รูปที่ 7.3 รหัสนักศึกษาและเวลาที่แสดงบนจอแสดงผล LCD

การทดสอบผลการถอดรหัสแท่งของเครื่องอ่านรหัสแท่งบันทึกเวลา ทำโดยใช้บัตรนักศึกษาจำนวน 5 ใบ รูดผ่านหัวอ่านจำนวน 100 ครั้ง / 1 บัตร ได้ผลการทดสอบดังตารางที่ 7.1

ตารางที่ 7.1 แสดงผลการทดลองที่ได้จากการรูดบัตรของนักศึกษา

รหัสนักศึกษา	จำนวนครั้งที่รูดบัตร	จำนวนครั้งที่ได้ข้อมูลถูกต้อง	จำนวนครั้งที่เกิด Error
37013030	100	98	2
37013259	100	99	1
37013279	100	97	3
37013280	100	100	0
37013106	100	99	1

บทที่ 7

การทดลองและผลการทดลอง

ในการทดสอบการทำงานของระบบนี้ได้แบ่งการทดสอบออกเป็น 2 ส่วน ดังนี้

1.หัวอ่านรหัสแท่ง

2.เครื่องอ่านรหัสแท่งบันทึกเวลา

7.1 หัวอ่านรหัสแท่ง

ทดสอบการอ่านรหัสแท่งโดยใช้ Digital Storage Scope วัดสัญญาณที่เอาท์พุทของหัวอ่าน แล้วนำบัตรทดสอบรหัสแท่งที่เป็นรหัส 39 (ข้อมูลบนบัตร คือ "150") มาทำการรูดบัตรทดสอบจะได้รูปสัญญาณดังรูปที่ 7.1 สัญญาณที่ได้จะมี 2 สภาวะ คือ "0" และ "1" โดยเมื่อหัวอ่านสแกน ผ่านแท่งดำก็จะได้สัญญาณลอจิกเป็น "1" และเมื่อสแกนผ่านแท่งขาวจะได้เป็นลอจิก "0"

รูปที่ 7.1 แสดงสัญญาณที่ได้จาก Digital Storage Scope

ผลการวัดความเร็วในการรูดบัตร

ได้ทำการวัดความเร็วที่ใช้ในการรูดบัตรผ่านหัวอ่าน โดยใช้บัตรนักศึกษาของสถาบัน ฯ ซึ่งใช้รหัส 39 ที่มีขนาดของความยาวใน 1 รหัสอักขระเท่ากับ 5 มิลลิเมตร ดังแสดงรูปที่ 7.2 โดยพิจารณาที่ความเร็วที่สามารถถอดรหัสได้ ดังนี้

รูปที่ 7.2 รหัส 39 ที่ใช้อยู่บนบัตรนักศึกษา

ความเร็วในการรูดบัตรที่สามารถถอดรหัสได้สมบูรณ์

ความเร็วในการรูดบัตรอย่างช้า = 110 มิลลิเมตร/วินาที

ความเร็วในการรูดบัตรอย่างเร็ว = 450 มิลลิเมตร/วินาที

ความเร็วในการรูดบัตรโดยเฉลี่ย = 280 มิลลิเมตร/วินาที

7.2 เครื่องอ่านรหัสแท่งบันทึกเวลา

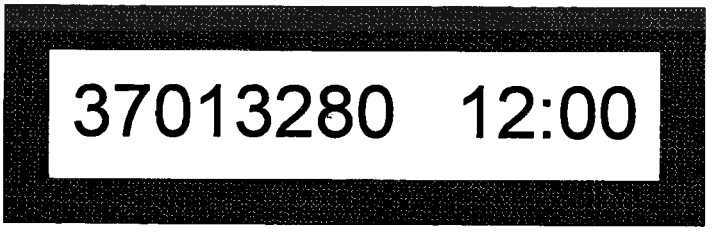
เครื่องอ่านรหัสแท่งบันทึกเวลา ทำหน้าที่ถอดรหัสแท่งจากบัตรนักศึกษา และบันทึกเวลาของนักศึกษาที่จะเข้าห้องปฏิบัติการ วิธีการทดสอบทำโดย

1. ป้อนไฟเข้าเครื่องอ่านรหัสแท่งบันทึกเวลา แล้วเชื่อมต่อสายกับพอร์ตอนุกรมมาตรฐาน RS-232C กรณีที่สถานที่ที่ติดตั้งใช้งานอยู่ไม่ห่างจากคอมพิวเตอร์ศูนย์กลางข้อมูล (ไม่เกิน 15 เมตร) หรือต่อสายกับพอร์ตอนุกรมมาตรฐาน RS-485 ในกรณีที่สถานที่ที่ติดตั้งใช้งานอยู่ไกลจากคอมพิวเตอร์ศูนย์กลางข้อมูล (ประมาณ 15 - 100 เมตร)

2. ทำการตั้ง วัน/เดือน/ปี และ เวลา (ชั่วโมง:นาที) บนเครื่องอ่านรหัสแท่งบันทึกเวลาด้วยคอมพิวเตอร์ศูนย์กลางข้อมูล (รายการที่ 3 บนโปรแกรมจัดการข้อมูล)

3. เมื่อเครื่องอ่านรหัสแท่งบันทึกเวลาแสดงวันและเวลาที่ถูกต้อง (ดูผลที่จอ LCD) ตรงกับวันและเวลาบนคอมพิวเตอร์ศูนย์กลางข้อมูล แสดงว่าเครื่องพร้อมที่จะให้รูดบัตรและบันทึกเวลาที่ถูกต้องได้

4. ทดสอบการถอดรหัสแท่งโดยใช้บัตรนักศึกษารหัส 37013280 ที่จอแสดงผล LCD ของเครื่องอ่านรหัสแท่งบันทึกเวลา จะแสดงรหัสนักศึกษาของบัตรที่รูด คือ 37013280 และเวลาขณะที่รูดบัตรเสร็จ ดังแสดงในรูปที่ 7.3 พร้อมทั้งส่งเสียงบี๊บสั้น ๆ 1 ครั้ง ซึ่งได้ผลถูกต้องตามที่ได้ออกแบบไว้



รูปที่ 7.3 รหัสนักศึกษาและเวลาที่แสดงบนจอแสดงผล LCD

การทดสอบผลการถอดรหัสแท่งของเครื่องอ่านรหัสแท่งบันทึกเวลา ทำโดยใช้บัตรนักศึกษาจำนวน 5 ใบ รูดผ่านหัวอ่านจำนวน 100 ครั้ง / 1 บัตร ได้ผลการทดสอบดังตารางที่ 7.1

ตารางที่ 7.1 แสดงผลการทดลองที่ได้จากการรูดบัตรของนักศึกษา

รหัสนักศึกษา	จำนวนครั้งที่รูดบัตร	จำนวนครั้งที่ได้ข้อมูลถูกต้อง	จำนวนครั้งที่เกิด Error
37013030	100	98	2
37013259	100	99	1
37013279	100	97	3
37013280	100	100	0
37013106	100	99	1

บทที่ 8
บทสรุป

ในโครงงานนี้ได้เลือกใช้หัวอ่านแบบ Slot scanner แล้วใช้บัตรนักศึกษารูดผ่านทำให้ได้รูปสัญญาณที่ถูกต้อง,และแน่นอน จึงทำให้ในขั้นตอนของการถอดรหัสจากรูปสัญญาณที่ได้ ทำได้ถูกต้องด้วย ซึ่งหากว่าเลือกใช้หัวอ่านที่มีความแม่นยำในการอ่านสัญญาณต่ำ ก็จะทำให้ได้สัญญาณที่ไม่ถูกต้อง ซึ่งจะมีผลทำให้ในขั้นตอนการถอดรหัสทำได้ไม่ถูกต้องด้วย โครงงานระบบรหัสแท่งแบบเครือข่ายนี้ สามารถเลือกใช้งานตามลักษณะของการเชื่อมต่อและสื่อสารข้อมูลได้ 2 ลักษณะ คือ

1. สามารถใช้ในการสื่อสารข้อมูลเป็นเครือข่ายแบบบัส กับเครื่องไมโครคอมพิวเตอร์ที่ใช้เป็นศูนย์กลางข้อมูลโดยใช้มาตราฐาน RS-485 ซึ่งสามารถสื่อสารข้อมูลกันได้ตลอดเวลา ทำให้เกิดความสะดวกและรวดเร็วต่อการนำข้อมูลไปใช้งาน

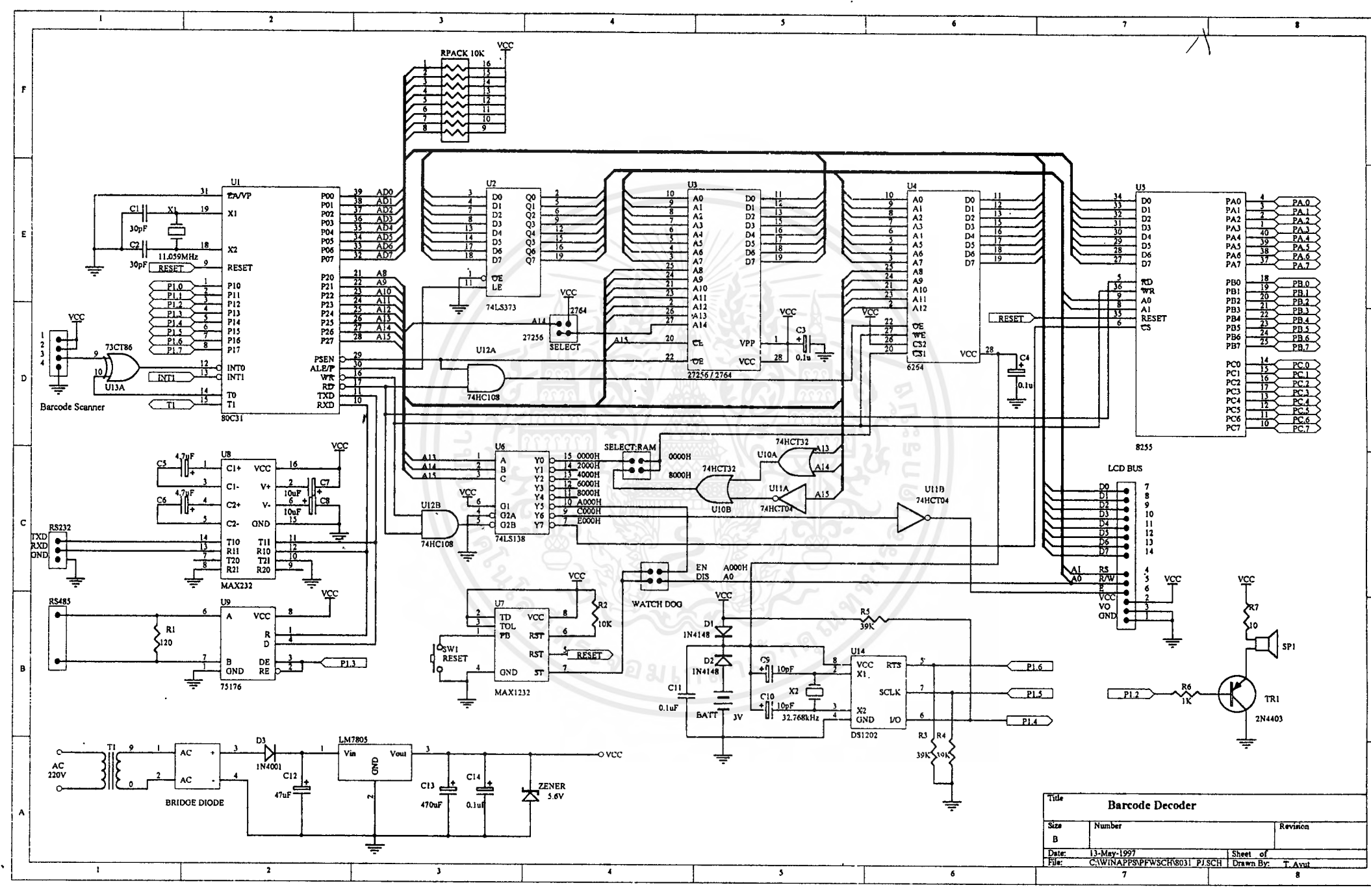
2. สามารถนำไปใช้แบบเดี่ยวๆ (Standalone) ได้โดยใช้การเชื่อมต่อและสื่อสารข้อมูลแบบอนุกรมมาตราฐาน RS-232C ซึ่งใช้สำหรับการส่งถ่ายข้อมูลไปยังเครื่องไมโครคอมพิวเตอร์ซึ่งเป็นศูนย์กลางข้อมูลแบบจุดต่อจุดในระยะทางใกล้ โดยสามารถเชื่อมกับไมโครคอมพิวเตอร์ซึ่งมีพอร์ตอนุกรม RS-232C อยู่ในตัวได้โดยตรง

จากลักษณะการใช้งานทั้งสองแบบนี้สามารถนำไปใช้ในห้องปฏิบัติการอิเล็กทรอนิกส์เพื่อบันทึกเวลาของนักศึกษาที่มาปฏิบัติการทดลองได้

ข้อเสนอแนะและแนวทางในการพัฒนา

เนื่องจากระบบรหัสแท่งแบบเครือข่าย ที่ได้ออกแบบสร้างและพัฒนาขึ้นมานี้ สามารถใช้ได้กับรหัส 39 เท่านั้น เพื่อให้สามารถใช้งานได้หลากหลายรหัสในการพัฒนาขั้นต่อไปควรจะเพิ่มส่วนของโปรแกรมการถอดรหัสแท่งให้ทำการถอดรหัสได้หลายรูปแบบมากขึ้น เช่น รหัสเอียน หรือ รหัสแทรก 2 ใน 5 เป็นต้น การจะนำโครงงานนี้ไปประยุกต์ใช้งานสามารถใช้ได้กับสถานที่ทำงานทั่วไปที่มีการลงเวลางาน โดยนำไปใช้แทนการตอกบัตร หรือการเซ็นชื่อลงเวลาทำงานได้ โดยปรับปรุงแก้ไขส่วนของโปรแกรมจัดการข้อมูลเพียงเล็กน้อยเท่านั้น

ภาคผนวก ก.
วงจรรวมของเครื่องอ่านรหัสแท่งบันทึกเวลา


RPACK 10K
VCC
U1
80C31
X1
11.059MHz
30pF
RESET
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
Barcode Scanner
73CT86
U13A
INT1
T1
U2
74LS373
U3
27256/2764
U12A
74HC108
SELECT
U4
6264
U5
8255
LCD BUS
U6
74LS138
SELECT:RAM
0000H
8000H
74HCT32
U10A
U10B
U11A
74HCT04
U11B
U8
MAX232
RS232
TXD
RXD
GND
U9
75176
RS485
R1
120
P1.3
U7
MAX1232
SW1
RESET
R2
10K
WATCH DOG
EN
DIS
A000H
U14
DS1202
X2
32.768kHz
BATT
3V
D1
D2
1N4148
C11
0.1uF
C9
C10
10pF
R5
39K
R3
R4
P1.6
P1.5
P1.4
P1.2
R6
1K
TR1
2N4403
SP1
R7
10
AC
220V
T1
BRIDGE DIODE
D3
1N4001
C12
47uF
LM7805
Vin
Vout
GND
C13
470uF
C14
0.1uF
ZENER
5.6V
Title
Barcode Decoder
Size
B
Number
Revision
Date: 13-May-1997
Sheet of
File: C:\WINAPPS\PFWSCH\8031_PJ.SCH
Drawn By: T. Ayut

ภาคผนวก ข.

โปรแกรมถอดรหัสแท่ง

```
;=========================================;
;    BARCODE TIME RECODER      ;
;     ASSEMBLER : SXA51        ;
;       AUTHOR : T.AVUT        ;
;=========================================;
;=========================================;
;    Define constants          ;
;=========================================;
;
NODE_ID   EQU   2
;
;    LCD Address const.
;
LCD_CMD   EQU   0C000H      ;Read-Write Register
RD_BUSY   EQU   0C001H      ;Read BF(Busy Flag) and address
WR_DATA   EQU   0C002H      ;Write character
RD_DATA   EQU   0C003H      ;Read Data from DD ram
;
;    Port 1 interface
;
SPEAKER   EQU   P1.2
RS_485    EQU   P1.3
RTC_IO    EQU   P1.4
RTC_CLK   EQU   P1.5
RTC_RST   EQU   P1.6
CTRLBIT   EQU   P3.4
;
;    Buffer constants
;
TEMP_BUF  EQU   3AH         ;3AH - 49H
TIME_BUF  EQU   4AH         ;4AH - 59H
HEX_BUF   EQU   5AH         ;5AH - 5FH
LCD_BUF   EQU   60H         ;60H - 6FH
BAR_BUF   EQU   0010H       ;0010H - 010BH
DATA_BUF  EQU   010CH       ;010CH - 1FFFH
;
;    Other constants
;
MAXID     EQU   8
```

```
MAXCH     EQU   8
LINES     EQU   4
DATA_NO   EQU   19
TIME_NO   EQU   11
TDELAY    EQU   230
;
;=============================================;
;     Define variables          ;
;=============================================;
          DS    20H          ;Start register uses
FLAG:     DS    1
PASS      EQU   00H
MIR       EQU   01H
ERR       EQU   02H
CODE      EQU   03H
END_ID    EQU   04H
CONV:     DS    1
L_WIDTH:  DS    1
H_WIDTH:  DS    1
L_INDX1:  DS    1
H_INDX1:  DS    1
L_INDX2:  DS    1
H_INDX2:  DS    1
P_LOOP:   DS    1
L_AVG:    DS    1
H_AVG:    DS    1
L_TMP1:   DS    1
H_TMP1:   DS    1
L_TMP2:   DS    1
H_TMP2:   DS    1
ID_NUM:   DS    1
;
NMSL:     DS    1
NMSH:     DS    1
ONEMS:    DS    1
INDEX:    DS    1
COUNT:    DS    1
;
;
```

```
        ORG   0000H        ;Start address
        LJMP  MAIN
;
        ORG   0003H
        LJMP  INT_INT0
;
        ORG   000BH
        LJMP  INT_TC0
;
        ORG   001BH
        LJMP  INT_TC1
;
        ORG   0023H
        LJMP  INT_SER
;
;=============================================;
;   MAIN PROGRAM          ;
;=============================================;
MAIN:   MOV   TMOD,#19H    ;
        MOV   TCON,#00H    ;
        MOV   IP,#00H      ;
        MOV   PSW,#00H     ;
        CALL  DISPLAY_1
        CALL  CLR_RAM
;
START:  MOV   SP,#08H      ;Set STACK
        MOV   L_INDX1,#10H ;Start bar buffer index
        MOV   H_INDX1,#00H ;Addr. = 0010H
        CLR   CTRLBIT      ;Clear Crtl. bit (T0 = 0)
;
        SETB  EA           ;Enable all interrupt
NXT_TIME: CALL DISP_TIME
        MOV   R1,#TEMP_BUF ;Set start internal buffer addr.
        CLR   A            ;Clear Acc
        MOV   TL0,A        ;Clear Timer0 low byte
        MOV   TH0,A        ;Clear Timer0 high byte
        MOV   CONV,A       ;Clear Convert var.
        CLR   TF0          ;Clear Timer0 overflow flag (TF0 = 0)
        SETB  TR0          ;Set Timer0 to strat/continue count
```

```
        CLR   IE0          ;Clear external int. (INT0)
        SETB  CTRLBIT      ;Set Ctrl_Bit (T0 = 1) to next interrupt
                           ;next bar come in
        SETB  EX0          ;Set external interrupt (INT0)
;
        SJMP  $            ;Dynamic end
;
;
;=============================================;
;   DECODE PART            ;
;=============================================;
        MOV   DPTR,#BAR_BUF  ;Set external buffer addr.
;
DECODE:  PUSH  DPL         ;Save low buffer addr.
        CALL  MOV_BAR      ;Move Bar width to Temp var.
;
        MOV   B,#07H       ;Set compare loop
NXT_COMP: CALL  COMPARE    ;Compare barcode width
        DJNZ  B,NXT_COMP   ;
;
        POP   DPL          ;Return low buffer addr.
        CALL  DIV_2        ;Division by 2
;
        MOV   B,#09H       ;Set barcode module convert loop (B = 9)
        MOV   CONV,#00H    ;Clear Convert var.
CONVERT:  CLR   C          ;Clear Carry
        MOVX  A,@DPTR      ;A = BarWidth (low byte)
        INC   DPTR         ;
        SUBB  A,L_AVG      ;If A < Average_Low, C = 1., Else, C = 0
        MOVX  A,@DPTR      ;A = BarWidth (high byte)
        INC   DPTR         ;
        SUBB  A,H_AVG      ;If A < Average_High, C = 1., Else, C = 0
        CPL   C            ;If BarWidth > Average = 1, Else bar = 0
        MOV   A,CONV       ;A = Convert var.
        RLC   A            ;A bit 0 = Carry (convert bit)
        MOV   CONV,A       ;Convert var. = A
        DJNZ  B,CONVERT    ;B = B - 1 ,Loop if B <> 0
;
        INC   DPTR         ;Increment pointer
```

```
        INC   DPTR          ;Increment pointer
        PUSH  DPL           ;Save buffer pointer
        PUSH  DPH           ;
        CJNE  A,#52H,N_TABLE ;If Start code <> 52H, select normal table
        SETB  CODE          ;CODE flag = 1 (reverse table)
N_TABLE: JB   CODE,R_TABLE  ;If CODE = 1, select reverse table
        MOV   DPTR,#CODE_39N ;Code 39 (normal) to ASCII table
        SJMP  MOV_VAL       ;
R_TABLE: MOV  DPTR,#CODE_39R ;Code 39 (reverse) to ASCII table
MOV_VAL: MOVC  A,@A+DPTR    ;A = value in table
        JB    ERR,CHK_ERR   ;if ERR flag = 1, check error
        CJNE  A,#'*',OUT_CODE ;If not start code ? (A <> '*'), exit
        SETB  ERR           ;ERR flag = 1
        CALL  WRT_BUF1      ;Save ASCII decode to buffer
;
NXT_CODE: POP  DPH          ;Return pointer
        POP   DPL           ;
        SJMP  DECODE        ;
;
CHK_ERR: CJNE  A,#'?',NO_ERR ;If A <> '?', no error (continue)
        CALL  DISP_ERR1
        SJMP  OUT_CODE      ;Else, exit decode
;
NO_ERR:  CALL  WRT_BUF1     ;Save ASCII decode to buffer
        CJNE  A,#'*',NXT_CODE ;If not stop code, next decode
        POP   ACC           ;Return Acc
        POP   ACC           ;Return Acc
        LJMP  CHK_ID        ;
;
;
OUT_CODE: POP  ACC          ;
        POP   ACC           ;
        LJMP  START         ;
;
;
MOV_BAR: MOVX  A,@DPTR      ;Move Bar value to Temp
        MOV   L_TMP1,A      ;Temp1_low = Bar Value (low byte)
        MOV   L_TMP2,A      ;Temp2_low = Bar Value (low byte)
        INC   DPTR          ;
```

```
        MOVX   A,@DPTR      ;
        MOV    H_TMP1,A     ;Temp1_high = Bar Value (high byte)
        MOV    H_TMP2,A     ;Temp2_high = Bar Value (high byte)
        INC    DPTR         ;
        RET
;
;
COMPARE:  MOVX   A,@DPTR    ;Compare Bar width with Temp
        MOV    L_WIDTH,A    ;BarWidth (low byte)
        INC    DPTR         ;
        MOVX   A,@DPTR      ;
        MOV    H_WIDTH,A    ;BarWidth (high byte)
        INC    DPTR         ;
        CLR    C            ;Clear Carry
        MOV    A,L_TMP1     ;Compare Temp1_Low with BarWidth (low)
        SUBB   A,L_WIDTH    ;
        MOV    A,H_TMP1     ;Compare Temp1_High with BarWidth (high)
        SUBB   A,H_WIDTH    ;
        JC     MIN_COMP     ;If Temp1 < BarWidth, Minimum Compare
        MOV    L_TMP1,L_WIDTH ;Else, Temp1 = BarWidth (Maximum)
        MOV    H_TMP1,H_WIDTH ;
        RET
;
;
MIN_COMP:  CLR   C          ;Clear Carry
        MOV    A,L_TMP2     ;Compare Temp2_Low with BarWidth (low)
        SUBB   A,L_WIDTH    ;
        MOV    A,H_TMP2     ;Compare Temp2_High with BarWidth (high)
        SUBB   A,H_WIDTH    ;
        JC     MIN_VAL      ;If Temp2 < BarWidth, Minimum Value
        RET                 ;Else, return.
;
;
MIN_VAL:   MOV   L_TMP2,L_WIDTH ;Temp2 = BarWidth (Minimum)
        MOV    H_TMP2,H_WIDTH ;
        RET
;
;
DIV_2:    MOV   A,H_TMP2    ;Fine Average value
```

```
        ADDC   A,H_TMP1      ;Average_High = Temp1_High + Temp2_High
        RRC    A             ;
        MOV    H_AVG,A       ;Average_High = Average_High / 2
        MOV    A,L_TMP2      ;
        ADD    A,L_TMP1      ;Average_Low = Temp1_Low + Temp2_Low
        RRC    A             ;
        MOV    L_AVG,A       ;Average_Low = Average_Low / 2
        RET
;

;=========================================;
; LCD_DISP Subroutine           ;
; Display string from external ram ;
;    Input : DPTR               ;
; Uses reg. : A, R0, DPTR       ;
; Sub. call : LCD_WRITE,NEW_LINE  ;
;=========================================;
LCD_DISP:  CALL   LCD_INIT        ;Initialize & Clear LCD display
       MOV    R0,#0
WRT_CHR:   MOV    A,R0
       INC    R0             ;increase index
       MOVC   A,@A+DPTR      ;A = character
       CALL   LCD_WRITE      ;write character to LCD
       CJNE   R0,#MAXCH,CHK_MAX  ;check counter for new line
       CALL   NEW_LINE       ;set new LCD-address (8 chars. last)
CHK_MAX:   CJNE   R0,#MAXCH*2,WRT_CHR
       CALL   DELAY3
       RET

;=========================================;
; BUF_DISP Subroutine           ;
; Display string from internal ram ;
;    Input : R1 = internal buffer ;
; Uses reg. : A, R0, R1         ;
; Sub. call : LCD_WRITE,NEW_LINE  ;
;=========================================;
BUF_DISP:  CALL   LCD_INIT        ;Initialize & Clear LCD display
       MOV    R0,#0
       MOV    R1,#LCD_BUF
```

```
WRT_CHR1:  MOV    A,@R1          ;A = character
           INC    R1
           INC    R0             ;increase index
           CALL   LCD_WRITE      ;write character to LCD
           CJNE   R0,#MAXCH,CHK_MAX1  ;check counter for new line
           CALL   NEW_LINE       ;set new LCD-address (8 chars. last)
CHK_MAX1:  CJNE   R0,#MAXCH*2,WRT_CHR1
           RET

;=========================================;
;  LCD_CLEAR Subroutine         ;
;  Uses reg. : DPTR             ;
;  Sub. call : LCD_DISP         ;
;=========================================;
LCD_CLEAR: PUSH   DPL
           PUSH   DPH
           MOV    DPTR,#SPACETAB ;
           CALL   LCD_DISP
           POP    DPH
           POP    DPL
           RET

;=========================================;
;  DISPLAY_1 Subroutine         ;
;  Uses reg. : DPTR             ;
;  Sub. call : LCD_DISP         ;
;=========================================;
DISPLAY_1: MOV    DPTR,#CHAR1    ;
           CALL   LCD_DISP
           MOV    DPTR,#CHAR2    ;
           CALL   LCD_DISP
           MOV    DPTR,#CHAR3    ;
           CALL   LCD_DISP
           RET

;=========================================;
;  DISPLAY_2 Subroutine         ;
;  Uses reg. : DPTR             ;
;  Sub. call : LCD_DISP         ;
```

```
;=========================================;
DISPLAY_2: PUSH   DPL
       PUSH   DPH
       MOV    DPTR,#CHAR2   ;
       CALL   LCD_DISP
       POP    DPH
       POP    DPL
       CALL   DELAY3
       RET

;=========================================;
;  DISPLAY_3 Subroutine        ;
;  Uses reg. : DPTR            ;
;  Sub. call : LCD_DISP        ;
;=========================================;
DISPLAY_3: MOV    DPTR,#CHAR3   ;
       CALL   LCD_DISP
       RET

;=========================================;
;  DISP_ERR1 Subroutine        ;
;  Uses reg. : DPTR            ;
;  Sub. call : LCD_DISP        ;
;=========================================;
DISP_ERR1: CALL   XBEEP
       MOV    DPTR,#ERROR1   ;
       CALL   LCD_DISP
       RET

;=========================================;
;  DISP_ERR2 Subroutine        ;
;  Uses reg. : DPTR            ;
;  Sub. call : LCD_DISP        ;
;=========================================;
DISP_ERR2: CALL   XBEEP
       MOV    DPTR,#ERROR2   ;
       CALL   LCD_DISP
       RET
```

```
;=========================================;
; DISP_ERR3 Subroutine          ;
; Uses reg. : DPTR              ;
; Sub. call : LCD_DISP          ;
;=========================================;
DISP_ERR3: CALL   XBEEP
        MOV    DPTR,#ERROR3   ;
        CALL   LCD_DISP
        RET

;=========================================;
; DISP_TIME Subroutine          ;
; Uses reg. : R1                ;
; Sub. call : BUF_DISP          ;
;=========================================;
DISP_TIME: CALL   TIME_READ
        CALL   TIMETOBUF
        CALL   BUF_DISP
        RET

;=========================================;
; DMSEC Subroutine              ;
; Delay 1/1000 second           ;
;    Input : R2,R3              ;
; Uses reg. : A, R2, R3, R4     ;
;=========================================;
DMSEC:    MOV    R4,#230       ;1 MSEC LOOP
DMSEC1:   NOP
        NOP
        DJNZ   R4,DMSEC1
        DJNZ   R3,DMSEC
        MOV    A,R2
        CJNE   A,#0,DMSEC2
        RET
DMSEC2:   DEC    R2
        SJMP   DMSEC

;=========================================;
; DTSEC Subroutine              ;
```

```
;  Delay 1/10 second            ;
;     Input : R2                ;
;  Uses reg. : R2, R3, R4       ;
;=================================;
DTSEC:    MOV    R3,#179
DTSEC1:   MOV    R4,#0
          DJNZ   R4,$
          NOP
          NOP
          DJNZ   R3,DTSEC1
          DJNZ   R2,DTSEC
          RET

;=================================;
;  DSEC Subroutine              ;
;  Delay second                 ;
;     Input : R1                ;
;  Uses reg. : R1, R2, R3, R4   ;
;=================================;
DSEC:     MOV    R2,#10
          CALL   DTSEC
          DJNZ   R1,DSEC
          RET

;=================================;
;  DELAY Subroutine             ;
;  Delay time in millisecond    ;
;     Input : NMSL = Low byte  (ms)  ;
;             NMSH = High byte (ms)  ;
;  Uses reg. : A                ;
;  Uses var. : ONEMS, NMSL, NMSH     ;
;=================================;
DELAY:    PUSH   ACC
SET_D:    MOV    ONEMS,#TDELAY  ;set time delay = 230
MIL1:     NOP                   ;4 cycle
          NOP                   ;4 cycle
          DJNZ   ONEMS,MIL1     ;time= 1.000434 ms
          NOP
          NOP                   ;time=12*(1/11.0592MHz)*[(4*230)+2]
```

```
        DJNZ   NMSL,SET_D     ;timedelay = 1.000434*NMSL
        PUSH   ACC
        MOV    A,NMSH
        CJNE   A,#0,HIDOWN
        POP    ACC
        SJMP   DONE
HIDOWN:  DEC    A
        MOV    NMSH,A
        POP    ACC
        SJMP   SET_D
        POP    ACC
DONE:     RET

;=============================================;
;  LCD_INIT Subroutine          ;
;  Initialize LCD module        ;
; Uses reg. : A,DPTR            ;
; Sub. call : WAIT_BUSY         ;
;=============================================;
LCD_INIT:  PUSH   DPL
        PUSH   DPH
        MOV    DPTR,#LCD_CMD  ;LCD command address
        MOV    A,#3CH         ;8 bit, 2 line, 5x10 dot
        MOVX   @DPTR,A        ;
        CALL   WAIT_BUSY      ;wait LCD ready
        MOV    A,#0CH         ;Display on, cursor off
        MOVX   @DPTR,A        ;
        CALL   WAIT_BUSY      ;wait LCD ready
        MOV    A,#01H         ;Clear all & home
        MOVX   @DPTR,A        ;
        CALL   WAIT_BUSY      ;wait LCD ready
        POP    DPH
        POP    DPL
        RET

;=============================================;
;  NEW_LINE Subroutine          ;
;  New line display (8 bytes last) ;
;  Uses reg. : A,DPTR           ;
```

```
; Sub. call : WAIT_BUSY          ;
;=========================================;
NEW_LINE:  PUSH   DPL
           PUSH   DPH
           MOV    DPTR,#LCD_CMD  ;LCD command address
           MOV    A,#0C0H        ;New line command
           MOVX   @DPTR,A        ;
           CALL   WAIT_BUSY      ;Wait LCD ready
           POP    DPH
           POP    DPL
           RET

;=========================================;
;  LCD_WRITE Subroutine          ;
;  Write character to LCD        ;
;     Input : A = Character      ;
;  Uses reg. : A, DPTR           ;
;  Sub. call : WAIT_BUSY         ;
;=========================================;
LCD_WRITE: PUSH   DPL
           PUSH   DPH
           MOV    DPTR,#WR_DATA  ;LCD write address
           MOVX   @DPTR,A        ;
           CALL   WAIT_BUSY      ;Wait LCD ready
           POP    DPH
           POP    DPL
           RET

;=========================================;
;  WAIT_BUSY Subroutine          ;
;  Wait for ready                ;
;  by mean of check busy flag    ;
;  Uses reg. : A, DPTR           ;
;=========================================;
WAIT_BUSY: PUSH   DPL
           PUSH   DPH
           MOV    DPTR,#RD_BUSY  ;Read busy address
READY:     MOVX   A,@DPTR
           JB     ACC.7,READY    ;Check Busy Flag
```

```
        POP   DPH
        POP   DPL
        RET

;=============================================;
; TIME_READ Subroutine          ;
; Read date & time from RTC DS1202 ;
; Uses reg. : A, R0, R6, R7     ;
; Sub. call : RTC_READ, WRT_BUF0  ;
;=============================================;
TIME_READ: MOV   R0,#HEX_BUF   ;Hex timer buffer
        MOV   R6,#87H      ;Read day
        CALL  RTC_READ     ;
        MOV   A,R7
        CALL  WRT_BUF0
        MOV   R6,#89H      ;Read month
        CALL  RTC_READ
        MOV   A,R7
        CALL  WRT_BUF0
        MOV   R6,#8DH      ;Read year
        CALL  RTC_READ
        MOV   A,R7
        CALL  WRT_BUF0
        MOV   R6,#85H      ;Read hour
        CALL  RTC_READ
        MOV   A,R7
        CALL  WRT_BUF0
        MOV   R6,#83H      ;Read minute
        CALL  RTC_READ
        MOV   A,R7
        CALL  WRT_BUF0
        MOV   R6,#81H      ;Read second
        CALL  RTC_READ
        MOV   A,R7
        CALL  WRT_BUF0
        RET

WRT_BUF0: MOV   @R0,A
        INC   R0
```

```
        RET

;=============================================;
; TIMETOBUF Subroutine          ;
; Write time to LCD buffer      ;
; Subs. call : HTOA, WRT_BUF1   ;
; Use regs. A,B,R0,R2,R3        ;
;=============================================;
TIMETOBUF: MOV   R0,#HEX_BUF   ;Hex timer buffer
        MOV   B,#3          ;Count for read date
        MOV   R1,#LCD_BUF   ;LCD display buffer
NXT_BUF1:  CALL   OUT_BUF0     ;Read data
        CALL  HTOA          ;Convert Hex to ASCII
        MOV   A,R2          ;Write date (high byte)
        CALL  WRT_BUF1      ;
        MOV   A,R3          ;Write date (low byte)
        CALL  WRT_BUF1      ;
        MOV   A,#'-'        ;Write dash
        CALL  WRT_BUF1      ;
        DJNZ  B,NXT_BUF1
;
        MOV   R1,#LCD_BUF+8
        MOV   B,#3          ;Count for write space
NXT_BUF2:  MOV   A,#' '        ;
        CALL  WRT_BUF1      ;
        DJNZ  B,NXT_BUF2
;
        MOV   B,#2
        MOV   R0,#HEX_BUF+3  ;Hex timer buffer (time)
NXT_BUF3:  CALL   OUT_BUF0     ;Read time
        CALL  HTOA          ;Convert Hex to ASCII
        MOV   A,R2          ;Write time (high byte)
        CALL  WRT_BUF1      ;
        MOV   A,R3          ;Write time (low byte)
        CALL  WRT_BUF1      ;
        MOV   A,#':'        ;Write colon
        CALL  WRT_BUF1
        DJNZ  B,NXT_BUF3
        RET
```

```
;
WRT_BUF1:  MOV    @R1,A
        INC    R1
        RET
;
OUT_BUF0:  MOV    A,@R0
        INC    R0
        RET


;=============================================;
;  RTC_SET Subroutine            ;
;  Initialize DS1202 RTC         ;
;     Input : R6 = command       ;
;             R7 = data          ;
;  Uses reg. : A, R6, R7         ;
;  Sub. call : RTC_WRITE, ASC2HEX  ;
;=============================================;
RTC_SET:   CLR    RTC_RST      ;RTS = 0
        CLR    RTC_CLK      ;CLK = 0
        MOV    R2,#2
        CALL   DTSEC
        MOV    R6,#8EH      ;Write protection command
        MOV    R7,#00H      ;Clear write protect
        CALL   RTC_WRITE
        MOV    R6,#86H      ;Write day
        MOV    R0,#TIME_BUF
        CALL   ASC2HEX
        MOV    R7,A         ;
        CALL   RTC_WRITE
        MOV    R6,#88H      ;Write month
        MOV    R0,#TIME_BUF+2
        CALL   ASC2HEX
        MOV    R7,A
        CALL   RTC_WRITE
        MOV    R6,#8CH      ;Write year
        MOV    R0,#TIME_BUF+4
        CALL   ASC2HEX
        MOV    R7,A
```

```
        CALL   RTC_WRITE
        MOV    R6,#84H      ;Write hour
        MOV    R0,#TIME_BUF+6
        CALL   ASC2HEX
        MOV    R7,A
        CALL   RTC_WRITE
        MOV    R6,#82H      ;Write minute
        MOV    R0,#TIME_BUF+8
        CALL   ASC2HEX
        MOV    R7,A
        CALL   RTC_WRITE
        MOV    R6,#80H      ;Write second and clear CHflag
        MOV    R7,#00H      ;second=0
        CALL   RTC_WRITE
        MOV    R6,#8EH      ;Write protection "ACTIVE"
        MOV    R7,#80H      ;Set write protect
        CALL   RTC_WRITE
        RET

;=======================================;
;  ASC2HEX Subroutine          ;
;  Convert ASCII to Hex        ;
;    Input  : R0 = buffer addr.  ;
;    Output : A  = Hex number    ;
; Uses reg. : A,R0,R2,R3       ;
; Sub. call : ATOH, OUT_BUF0    ;
;=======================================;
ASC2HEX:   CALL   OUT_BUF0
        MOV    R2,A
        MOV    A,@R0
        MOV    R3,A
        CALL   ATOH
        RET

;=======================================;
;  RTC_WRITE Subroutine        ;
;  Write command/data to DS1202  ;
;    Input : R6 = command      ;
;            R7 = data         ;
```

```
; Uses reg. : A, R6, R7          ;
; Sub. call : R_DELAY, WRT_8BIT  ;
;=============================================;
RTC_WRITE: CLR   RTC_CLK      ;CLK = 0
       CALL  R_DELAY
       SETB  RTC_RST      ;RST = 1
       CALL  R_DELAY
       MOV   A,R6         ;Command
       CALL  WRT_8BIT
       MOV   A,R7         ;Data
       CALL  WRT_8BIT
       CLR   RTC_RST      ;RST = 0
       CALL  R_DELAY
       RET

;=============================================;
;  WRT_8BIT Subroutine          ;
;  Write command/data to DS1202  ;
;  Uses reg. : A, B             ;
;  Sub. call : R_DELAY          ;
;=============================================;
WRT_8BIT:  MOV   B,#8          ;
RTC_WRT:   RRC   A             ;Shift right Acc, Carry = Acc bit 0
       MOV   RTC_IO,C      ;I/O = carry
       SETB  RTC_CLK       ;Rising edge clock
       CALL  R_DELAY
       CLR   RTC_CLK       ;CLK = 0
       CALL  R_DELAY
       DJNZ  B,RTC_WRT
       RET

;=============================================;
;  RTC_READ Subroutine          ;
;  Read command from DS1202 RTC  ;
;    Input : R6 = command       ;
;   Output : R7 = data          ;
; Uses reg. : B, R6, R7         ;
; Sub. call : R_DELAY, WRT_8BIT  ;
;=============================================;
```

```
RTC_READ:  CLR    RTC_CLK      ;CLK = 0
           CALL   R_DELAY
           SETB   RTC_RST      ;RST = 1
           CALL   R_DELAY
;
           MOV    A,R6         ;Write RTC command
           CALL   WRT_8BIT
;
           MOV    B,#8         ;Set loop 8 bit
           CLR    A            ;Clear Acc, carry = 0
RTC_RD1:   CLR    RTC_CLK      ;CLK = 0
           CALL   R_DELAY      ;
           MOV    C,RTC_IO     ;Carry = IO
           RRC    A            ;Shift right Acc, Acc bit 8 = carry
           SETB   RTC_CLK      ;CLK = 1
           CALL   R_DELAY
           DJNZ   B,RTC_RD1
           MOV    R7,A         ;Read RTC data
;
           CLR    RTC_RST      ;RST = 0
           CALL   R_DELAY
           RET

;=============================================;
; R_DELAY Subroutine          ;
; Pulse delay time            ;
; Uses reg : R1               ;
;=============================================;
R_DELAY:   MOV    R5,#4
           DJNZ   R5,$
           RET

;=============================================;
; HTOA Subroutine             ;
; Convert Hex to ASCII        ;
;   Input  : A  = Hex number  ;
;   Output : R2 = low byte    ;
;            R3 = high byte   ;
; Uses reg. : A, R2, R3       ;
```

```
;=============================================;
HTOA:    PUSH  ACC
         SWAP  A
         CALL  HTOAS
         MOV   R2,A
         POP   ACC
         CALL  HTOAS
         MOV   R3,A
         RET
;
HTOAS:   ANL   A,#0FH
         CJNE  A,#0AH,$+3
         JNC   HTOAS1
         ORL   A,#30H
         RET
;
HTOAS1:  SUBB  A,#9
         ORL   A,#40H
         RET

;=============================================;
; ATOH Subroutine           ;
; Convert ascii to hex      ;
;   Input  : R2,R3 = ASCII  ;
;   Output : A  = Hex number ;
; Uses Reg. : A, R2, R3     ;
;=============================================;
ATOH:    MOV   A,R2
         CALL  ATOHS
         SWAP  A
         MOV   R2,A
         MOV   A,R3
         CALL  ATOHS
         ORL   A,R2
         RET
;
ATOHS:   CJNE  A,#'A',$+3
         JC    ATOHS1
         ADD   A,#9
```

```
;
ATOHS1:   ANL   A,#0FH
          RET
;
;=============================================;
; SOUND Subroutine              ;
; Generate beep sound           ;
;   Input  : R2 = Frequency     ;
;            R3 = Lenght        ;
; Uses Reg. : A,R2,R3,R4,R5,DPTR ;
;=============================================;
SOUND:    MOV   R5,#0         ;End flag
          MOV   R4,#80H       ;Delay constant
SOUND1:   CALL  SOUNDS
          CJNE  R5,#1,SOUND1
          RET
;
SOUNDS:   SETB  SPEAKER
          CALL  SOUNDX
          CLR   SPEAKER
          CALL  SOUNDX
          RET
;
SOUNDX:   MOV   A,R2          ;Frequency delay
SOUNDX1:  CALL  SOUNDY
          DEC   A
          JNZ   SOUNDX1
          RET
;
SOUNDY:   DJNZ  R4,SOUNDY1    ;Length count down
          MOV   R4,#80H
          DJNZ  R3,SOUNDY1
          MOV   R5,#1
SOUNDY1:  RET

;=============================================;
; xBEEP Subroutine              ;
; Uses Reg. : R2,R3             ;
; Sub. call : SOUND             ;
```

```
;=============================================;
FBEEP:    MOV    R2,#55H
          MOV    R3,#15H
          CALL   SOUND
          RET

XBEEP:    MOV    R2,#50H
          MOV    R3,#15H
          CALL   SOUND
          RET

OBEEP:    MOV    R2,#45H
          MOV    R3,#15H
          CALL   SOUND
          RET
;
;=============================================;
; COM_INIT Subroutine          ;
; Initialize serial port       ;
;=============================================;
COM_INIT: MOV    TH1,#0FDH     ;set baud rate 9600
          ;MOV   TL1,#0FDH
          MOV    TMOD,#20H     ;set timer1 8 bit auto reload
          MOV    TCON,#40H
          ;MOV   SCON,#50H     ;set 8 bit UART
          MOV    SCON,#40H     ;set 8 bit UART
          SETB   TR1           ;
          CLR    RS_485
          RET
;
;=============================================;
; DISP_HEX Subroutine          ;
; Display Hex digit to LCD     ;
; Uses reg. : A, R2, R3        ;
;=============================================;
DISP_HEX: CALL   HTOA
          MOV    A,R2          ;high byte
          CALL   LCD_WRITE
          MOV    A,R3          ;low byte
```

```
        CALL   LCD_WRITE
        RET
;
;=============================================;
; SET_INDX Subroutine          ;
; Set external buffer index    ;
; Uses reg. : A, DPTR          ;
;=============================================;
SET_INDX:  MOV    DPTR,#0000H   ;Save start buffer index
        MOV    A,#01H
        MOVX   @DPTR,A
        INC    DPTR
        MOV    A,#0CH
        MOVX   @DPTR,A
        INC    DPTR
        RET

;=============================================;
; CLR_RAM Subroutine           ;
; Clear external RAM           ;
; Uses reg. : A, R2, R3, DPTR  ;
;=============================================;
CLR_RAM:   MOV    DPTR,#0000H
        MOV    R2,#20H
CLEAR1:    MOV    R3,#00H
CLEAR2:    MOV    A,#0FFH
        MOVX   @DPTR,A
        INC    DPTR
        DJNZ   R3,CLEAR2
        DJNZ   R2,CLEAR1
        MOV    L_INDX2,#0CH  ;Start bar buffer index
        MOV    H_INDX2,#01H  ;Addr. = 0010H
        RET

;=============================================;
; SET_RAM Subroutine           ;
; Clear external RAM           ;
; Uses reg. : A, R2, R3, DPTR  ;
;=============================================;
```

```
SET_RAM:   MOV    DPTR,#0000H
           MOV    R2,#20H
SET_1:     MOV    R3,#00H
SET_2:     MOV    A,#0FFH
           MOVX   @DPTR,A
           MOV    L_INDX2,DPL    ;
           MOV    H_INDX2,DPH    ;
           INC    DPTR
           CJNE   A,#0FFH,OUT_SET
           DJNZ   R3,SET_2
           DJNZ   R2,SET_1
OUT_SET:   RET

;
DISP_RAM:  MOV    DPTR,#0000H
           MOV    R2,#10H
NXT_RAM:   MOVX   A,@DPTR
           CALL   LCD_WRITE
           INC    DPTR
           DJNZ   R2,NXT_RAM
           RET

;=============================================;
;  INTERRUPT SERVICE ROUTINES     ;
;=============================================;
;  EXTERNAL INTERRUPT INT0        ;
;=============================================;
INT_INT0:  PUSH   ACC          ;Save Acc
           PUSH   PSW          ;Save PSW
           PUSH   DPL          ;Save DPTR
           PUSH   DPH          ;
           MOV    L_WIDTH,TL0    ;BarWidth = Timer0
           MOV    H_WIDTH,TH0    ;
           MOV    TH0,#00H       ;Set base time
           MOV    TL0,#11H       ;Timer0 = 0011H
           CPL    CTRLBIT        ;Complement Ctrl_bit (T0 = not(T0))
;
           MOV    DPL,L_INDX1    ;DPL = L_INDX1
           MOV    DPH,H_INDX1    ;DPH = H_INDX1, addr. = H_INDX1,L_INDX1
```

```
        JNB    CTRLBIT,CHK_BAR ;If Ctrl_Bit = 0, check BarWidth
        CLR    TR1             ;Stop counter Timer1
        JB     PASS,SAVE_BAR   ;If PASS flag = 1, save BarWidth
        SJMP   OUT_INT0        ;Else, exit Interrupt0
;
CHK_BAR:  CLR    TR1           ;Stop counter Timer1
        JB     PASS,NXT_CHK    ;If PASS flag = 1,
        JB     TF0,SET_FLAG    ;If Timer0 overflow (TF0=1), set flag
        MOV    A,H_WIDTH       ;Else, check BarWidth (high byte)
        SUBB   A,#0FH          ;
        JC     OUT_INT0        ;If BarWidth (high) < 15, exit INT0
;                              ;Else, set flag
SET_FLAG: SETB   PASS          ;PASS flag = 1
        MOV    P_LOOP,#05H     ;P_LOOP = 5
        CLR    TF0             ;Clear Timer0 overfow flag (TF0=0)
        SETB   ET0             ;Set interrupt Timer0 overflow (ET0=1)
        SJMP   OUT_INT0        ;Exit interrupt service routine INT0
;
NXT_CHK:  JB     MIR,SAVE_BAR  ;If MIR flag = 1, save bar width
        DJNZ   P_LOOP,SAVE_BAR ;P_LOOP = P_LOOP-1, if P_LOOP <> 0, save
        SETB   MIR             ;MIR flag = 1
;
SAVE_BAR: MOV    L_TMP1,L_WIDTH ;Temp1 = BarWidth
        MOV    H_TMP1,H_WIDTH  ;
        CALL   MULTI_2         ;Multiple with 2
        MOV    A,L_TMP1        ;Timer1 = not(BarWidth * 2)
        CPL    A               ;
        MOV    TL1,A           ;TL1 = not(BarWidth_low * 2)
        MOV    A,H_TMP1        ;
        CPL    A               ;
        MOV    TH1,A           ;TH1 = not(BarWidth_high * 2)
        CLR    TF1             ;Clear Timer1 overflow flag (TF1=0)
        SETB   TR1             ;Timer1 start/continue count (TR1=1)
        SETB   ET1             ;Set interrupt Timer1 overflow (ET1=1)
        MOV    A,L_WIDTH       ;Save BarWidth to external buffer
        MOVX   @DPTR,A         ;BarWidth (low byte)
        INC    DPTR            ;
        MOV    A,H_WIDTH       ;
        MOVX   @DPTR,A         ;BarWidth (high byte)
```

```
        INC   DPTR        ;
        MOV   L_INDX1,DPL   ;Save buffer addr. to Index
        MOV   H_INDX1,DPH   ;
;
OUT_INT0: POP   DPH         ;Save DPTR
        POP   DPL         ;
        POP   PSW         ;Save PSW
        POP   ACC         ;Save Acc
        RETI
;
;
MULTI_2:  MOV   B,#03H      ;Set multiple loop
NXT_MUL:  CLR   C           ;Clear Carry
        MOV   A,L_TMP1      ;Temp1 = Temp1 * 2
        RLC   A           ;
        MOV   L_TMP1,A      ;Temp1_low = Temp1_low * 2
        MOV   A,H_TMP1      ;
        RLC   A           ;
        MOV   H_TMP1,A      ;Temp1_high = Temp1_high * 2
        JC    SET_MAX       ;If Temp1 >= 80H, set max value
        DJNZ  B,NXT_MUL     ;
        RET
;
SET_MAX:  MOV   L_TMP1,#0FFH  ;Temp1 = 0FFFFH
        MOV   H_TMP1,#0FFH  ;
        RET

;=========================================;
;  TIMER/COUNTER 0 INTERRUPT     ;
;=========================================;
INT_TC0:  MOV   A,L_INDX1     ;
        SUBB  A,#05H        ;
        MOV   DPTR,#005AH   ;
        JNC   OUT_TC0       ;
        MOV   DPTR,#0034H   ;
;
OUT_TC0:  POP   ACC         ;Return Acc
        POP   ACC         ;Return Acc
        PUSH  DPL         ;Return DPTR
```

```
        PUSH   DPH          ;
        CLR    ET0          ;ET0 = 0
        CLR    EX0          ;EX0 = 0
        ANL    FLAG,#00H    ;Clear all bit flag
        RETI

;=============================================;
;  TIMER/COUNTER 1 INTERRUPT     ;
;=============================================;
INT_TC1:  JB     MIR,OUT_TC1   ;If MIR flag = 1, Out interrupt TC1
        MOV    H_INDX1,#00H  ;Start index buffer
        MOV    L_INDX1,#10H  ;Index addr. = 0010H
        ANL    FLAG,#0FCH    ;Clear PASS & MIR flag
        CLR    TR1           ;Stop counter Timer1
        RETI
;
;
OUT_TC1:  CLR    TR1           ;Stop counter Timer1
        CLR    ET1           ;Clear Timer1 overfow
        AJMP   INT_TC0       ;Jump to interrupt TC0
;
;  Check ID_CODE maximum
;
CHK_ID:   MOV    R1,#TEMP_BUF+1    ;CODE 39 decode buffer
        MOV    R0,#0         ;ID_num = 0
NXT_R1:   INC    R1            ;
        INC    R0
        CJNE   @R1,#'*',NXT_R1   ;If ID_code <> '*', next index
        CJNE   R0,#MAXID+1,CHK_RAM
ID_ERR:   CALL   DISP_ERR2
        AJMP   OUT_WRT

;
;  Check RAM addr., Is it Full ?
;
CHK_RAM:  MOV    A,H_INDX2     ;Check start index data buffer
        CJNE   A,#00H,CHK_FULL ;If High_Index <> 00H, check full.
        JMP    SET_START     ;Else, set start index buffer
;
```

```
CHK_FULL:  MOV    A,H_INDX2     ;Check High_Index buffer
           CLR    C
           SUBB   A,#20H        ;
           JC     SET_ADDR      ;If High_Index < 20H, set buffer addr.
           CALL   DISP_ERR3
           AJMP   OUT_WRT
;
SET_START: MOV    L_INDX2,#0CH  ;Else, set start buffer addr.
           MOV    H_INDX2,#01H

SET_ADDR:  MOV    DPL,L_INDX2   ;Set ext. buffer addr.
           MOV    DPH,H_INDX2   ;
;
SAVE_ADDR: PUSH   DPL           ;Save ext. buffer addr.
           PUSH   DPH
;
           JB     CODE,REVERSE  ;If CODE flag = 1, write reverse ID
;                                ;Else, write normal ID
;  Write ID code to data buffer
;
           MOV    R1,#TEMP_BUF+1 ;Write normal ID to data buffer
           MOV    B,#MAXID
WRT_NML:   MOV    A,@R1         ;
           CALL   WRITE_BUF     ;Write ID to buffer
           INC    R1            ;Increase index
           DJNZ   B,WRT_NML
           SJMP   WRT_TIME      ;
;
;  Write reverse ID code to buffer
;
REVERSE:   MOV    R1,#TEMP_BUF+MAXID ;
           MOV    B,#MAXID
WRT_REV:   MOV    A,@R1         ;
           DEC    R1            ;Decrease index
           CALL   WRITE_BUF     ;Write ID to buffer
           DJNZ   B,WRT_REV
;
;  Write date & time to data buffer
;
```

```
WRT_TIME:  CALL   TIME_READ     ;Read time from RTC (hex)
       MOV    R0,#HEX_BUF    ;Hex time buffer addr.
       MOV    B,#5           ;Loop for read date (5 byte)
NXT_BUFF:  CALL   OUT_BUF0      ;Hex time data
       CALL   HTOA           ;Convert Hex to ASCII
       MOV    A,R2           ;Write time (high byte)
       CALL   WRITE_BUF      ;
       MOV    A,R3           ;Write time (low byte)
       CALL   WRITE_BUF      ;
       DJNZ   B,NXT_BUFF     ;
       MOV    A,#'$'         ;End data1
       CALL   WRITE_BUF      ;
       MOV    A,#'*'         ;End data2
       CALL   WRITE_BUF      ;
;
       MOV    L_INDX2,DPL    ;Save data index
       MOV    H_INDX2,DPH
;
;  Display ID code to LCD
;
       POP    DPH            ;Return buffer addr. to display
       POP    DPL
;
       CALL   LCD_INIT       ;Initialize & Clear LCD display
       MOV    B,#MAXID       ;Loop for write ID number
ID_DISP:   CALL   READ_BUF      ;
       CALL   LCD_WRITE      ;
       INC    R1
       DJNZ   B,ID_DISP
;
       CALL   NEW_LINE       ;LCD new line display
       MOV    B,#3
SP_DISP:   MOV    A,#' '         ;Display space
       CALL   LCD_WRITE
       DJNZ   B,SP_DISP
;
       MOV    B,#6           ;Increment pointer to timer
NEXT_6:    INC    DPTR
       DJNZ   B,NEXT_6
```

```
;
        MOV   B,#2          ;Display hour
HR_DISP:  MOVX  A,@DPTR
        INC   DPTR
        CALL  LCD_WRITE
        DJNZ  B,HR_DISP
;
        MOV   A,#':'
        CALL  LCD_WRITE
;
        MOV   B,#2          ;Display second
MN_DISP:  MOVX  A,@DPTR
        INC   DPTR
        CALL  LCD_WRITE
        DJNZ  B,MN_DISP
;
        CALL  FBEEP
        CALL  DELAY3
OUT_WRT:  MOV   TMOD,#19H    ;
        ANL   IE,#80H       ;
        LJMP  START         ;
;
WRITE_BUF: MOVX  @DPTR,A
        INC   DPTR
        RET
;
READ_BUF:  MOVX  A,@DPTR
        INC   DPTR
        RET

;
DELAY3:   MOV   R0,#05H     ;
LOOP1:    MOV   R1,#00H     ;
LOOP2:    MOV   R2,#00H
LOOP3:    NOP               ;
        NOP               ;
        DJNZ  R2,LOOP3      ;
        DJNZ  R1,LOOP2      ;
        DJNZ  R0,LOOP1      ;
```

```
        RET

;=============================================;
;    SERIAL INTERRUPT        ;
;=============================================;
INT_SER:  JNB   RI,$
        CLR   ES
        MOV   A,SBUF
        CLR   RI
        CJNE  A,#00H,TRANS
SET_RTC:  MOV   R0,#TIME_BUF
        MOV   COUNT,#TIME_NO
WAIT_RI:  JNB   RI,WAIT_RI
        MOV   A,SBUF
        CLR   RI
        CALL  WRT_BUF0
        DJNZ  COUNT,WAIT_RI
;
CHK_SUM:  MOV   R7,#0
        MOV   R0,#TIME_BUF
        MOV   COUNT,#TIME_NO-1
SUM_CHK:  CALL  OUT_BUF0
        XRL   A,R7
        MOV   R7,A
        DJNZ  COUNT,SUM_CHK
;
        SETB  RS_485
        MOV   A,@R0
        CJNE  A,07H,ER_SEND
        MOV   A,#'*'
        CALL  SEND
;
RX_END:   CALL  RTC_SET
        MOV   DPTR,#TIME_RDY
        CALL  LCD_DISP
        CALL  FBEEP
        LJMP  SER_OUT
;
TRANS:    CJNE  A,#11H,CHK_NODE
```

```
        SJMP   TRAN1
CHK_NODE:  CJNE   A,#NODE_ID,SER_OUT
TRAN1:    SETB   RS_485
        MOV    DPTR,#DATA_BUF
        MOV    INDEX,#0
TRAN2:    ;CJNE   A,#02H,COM_ERR
        MOV    R7,#0
        ;MOV    COUNT,#DATA_NO
NXT_TX:   MOV    A,INDEX
        MOVC   A,@A+DPTR
        CALL   SEND
        INC    INDEX
        CLR    C
        CJNE   A,#0FFH,CHECK_1
        SJMP   TX_OUT
;
CHECK_1:  CJNE   A,#'$',XOR_DAT
        MOV    A,R7
        CALL   SEND
        SJMP   CHK_RI
XOR_DAT:  XRL    A,R7
        MOV    R7,A
        SJMP   NXT_TX
;
CHK_RI:   CLR    RS_485
        JNB    RI,CHK_RI
        MOV    R6,SBUF
        CLR    RI
        SETB   RS_485
        SJMP   TRAN2
;
TX_OUT:   MOV    DPTR,#TX_READY
        CALL   LCD_DISP
        CALL   FBEEP
        SJMP   SER_OUT
;
ER_SEND:  MOV    A,#' '
        CALL   SEND
COM_ERR:  MOV    DPTR,#TR_ERROR
```

```
        CALL   LCD_DISP
        CALL   XBEEP
        MOV    NMSL,#60      ;set delay time
        MOV    NMSH,#2       ;delay 5*60 ms
        CALL   DELAY
        CALL   XBEEP
;
SER_OUT:  CLR   RS_485
        CALL   DISP_TIME
        RETI
;
SEND:     CLR   TI
        MOV    SBUF,A
        JNB    TI,$
        RET
;
;=============================================;
;  Characters table for LCD display   ;
;=============================================;
CHAR1:    DB    '   KMITL    '
CHAR2:    DB    ' ELECTRONICS  '
CHAR3:    DB    'BARCODE NETWORK'
CHAR4:    DB    '  SYSTEM    '
CHAR5:    DB    'AVUT THANMACOM'
;
SPACETAB:  DB    '            '

TX_READY:  DB    'TRANFER READY'
;
TR_ERROR:  DB    'COMMUNICATION '
        DB    '  ERROR !!   '
;
TIME_RDY:  DB    'SET TIME READY'
;
ERROR1:    DB    'DECODE ERROR !!'
ERROR2:    DB    'ID_CODE ERROR'
ERROR3:    DB    'MEMORY FULL !!'
;
TXTBUF:    DB    '370132801203970900'
```

```
        DB    '370132651203970901'
        DB    '370132751203970905'
        DB    '370132601203970908'
        DB    '370132751203971600'
        DB    '370132651203971601'
        DB    '370132601203971605'
        DB    '370132801203971607',0FFH
;
;=============================================;
; Code 39 to ASCII normal table     ;
;      "?" is not code 39.          ;
;=============================================;
;       Column  0123456789ABCDEF
CODE_39N: DB    '???K??RQ?A??HG??'  ;Row 00H
        DB    '??ON??T?ED??J???'  ;Row 01H
        DB    '?1??87????%?????'  ;Row 02H
        DB    '54??0???????????'  ;Row 03H
        DB    '??ML??S?CB??I???'  ;Row 04H
        DB    '??P?????F???????'  ;Row 05H
        DB    '32??9???????????'  ;Row 06H
        DB    '6???????????????'  ;Row 07H
        DB    '?U??.-????+?????'  ;Row 08H
        DB    'YX??*???????????'  ;Row 09H
        DB    '??/?????$???????'  ;Row 0AH
        DB    '???????????????'  ;Row 0BH
        DB    'WV?? ???????????'  ;Row 0CH
        DB    'Z???????????????'  ;Row 0DH
        DB    '???????????????'  ;Row 0EH
        DB    '???????????????'  ;Row 0FH

;=============================================;
; Code 39 to ASCII reverse table    ;
;      "?" is not code 39.          ;
;=============================================;
;       Column  0123456789ABCDEF
CODE_39R: DB    '???U??VW?1??23??'  ;Row 00H
        DB    '??XY??Z?45??6???'  ;Row 01H
        DB    '?A??BC????$?????'  ;Row 02H
        DB    'DE??F???????????'  ;Row 03H
```

```
        DB    '??-.?? ?78??9???'  ;Row 04H
        DB    '??*?????0???????'  ;Row 05H
        DB    'GH??I???????????'  ;Row 06H
        DB    'J???????????????'  ;Row 07H
        DB    '?K??LM????/?????'  ;Row 08H
        DB    'NO??P???????????'  ;Row 09H
        DB    '??+?????%???????'  ;Row 0AH
        DB    '???????????????'  ;Row 0BH
        DB    'QR??S???????????'  ;Row 0CH
        DB    'T???????????????'  ;Row 0DH
        DB    '???????????????'  ;Row 0EH
        DB    '???????????????'  ;Row 0FH
;
        END
```

ภาคผนวก ค.
โปรแกรมจัดการข้อมูล

```
*********************************************
*       PROCEDURE FILE : KMITL.PRG          *
*********************************************
DO WHILE .T.
  CLEAR ALL
  DO Setinit
  DO Public
  DO Constant
  LOAD Cur_On
  LOAD Cur_Off
  CALL Cur_Off
  LOAD RecvData
  LOAD SetTime
  LOAD COMETMP
  CALL COMETMP
  ON ERROR DO ErrHand WITH ERROR(), MESSAGE(), MESSAGE(1), PROGRAM(), LINENO()
  gnMenu = 6
  DIMENSION gaChoice(gnMenu)
  gaChoice(1) = ' 1. รับข้อมูลแบบเครือข่าย      '
  gaChoice(2) = ' 2. รับข้อมูลแบบจุดต่อจุด      '
  gaChoice(3) = ' 3. ตั้งเวลาเครื่องบันทึกเวลา  '
  gaChoice(4) = ' 4. รายงานการเข้าห้องทดลอง     '
  gaChoice(5) = ' 5. แฟ้มข้อมูลนักศึกษา         '
  gaChoice(6) = ' 6. เลิกการทำงาน               '
  gcTitle = 'KMITL BARCODE TIME RECODER NETWORK'
  gcDate  = TDTOC(DATE())
  gcTime  = TIME()
  gnRow  = 1
  gnCol  = 5
  gnCMax = 69
  gnRMax = 22
  gnCSet = 17
  gnRSet = 5
  ACTIVATE SCREEN
  SET COLOR TO
  SET CLOCK TO gnRow+2,gnCol+gnCMax-12
```

```
@ 00,00 CLEAR TO 24,79
DO 3DBox WITH gnRow,gnCol,gnRow+gnRMax,gnCol+gnCMax,'W+/GB','N/GB'
DO 3DBox WITH gnRow+1,gnCol+3,gnRow+3,gnCol+gnCMax-3,'N/GB','W+/GB'
@ gnRow+2,gnCol+4 SAY PADC(gcTitle,gnCMax-7) COLOR GR+/GB
@ gnRow+2,gnCol+5 SAY gcDate COLOR GR+/RB
DO 3DBox WITH gnRow+gnRSet,gnCol+gnCSet,gnRow+(gnMenu*2)+gnRSet,gnCol+gnCMax-gnCSet, ;
      'W+/W','N/W'
SET COLOR TO N/W,N/G
@ gnRow+gnRSet+01,gnCol+gnCSet+2 PROMPT gaChoice(1)
@ gnRow+gnRSet+03,gnCol+gnCSet+2 PROMPT gaChoice(2)
@ gnRow+gnRSet+05,gnCol+gnCSet+2 PROMPT gaChoice(3)
@ gnRow+gnRSet+07,gnCol+gnCSet+2 PROMPT gaChoice(4)
@ gnRow+gnRSet+09,gnCol+gnCSet+2 PROMPT gaChoice(5)
@ gnRow+gnRSet+11,gnCol+gnCSet+2 PROMPT gaChoice(6)
SET COLOR TO
DO 3DBox WITH gnRMax-2,gnCol+2,gnRMax,gnCol+gnCMax-2,'N/W','W+/W'
@ gnRMax-1,gnCol+3 SAY PADC ('Choose      and <Enter> or Number to Selection',gnCMax-5) ;
      COLOR N/W
MENU TO gnChoice
DO SetColor
SET CLOCK OFF
DO CASE
CASE gnChoice = 1
  DO NetRecv
CASE gnChoice = 2
  DO RecvData
CASE gnChoice = 3
  DO SetTime
CASE gnChoice = 4
  DO LReport
CASE gnChoice = 5
  DO StudFile
CASE gnChoice = 6 .OR. LastKey()=27
  QUIT
OTHER WISE
  LOOP
```

```
   ENDCASE
ENDDO
RETURN

**********************************
*       Procedure : 3DBOX        *
**********************************
PROCEDURE 3DBox
PARAMETER xnRow1,xnCol1,xnRow2,xnCol2,xcColor1,xcColor2
lnThai=0
DO CASE
CASE lnThai=0        && English
   lcUpLeft  = '┌'
   lcUpRight = '┐'
   lcDnLeft  = '└'
   lcDnRight = '┘'
   lcSideLn  = '│'
   lcDashLn  = '─'
CASE lnThai=1        && Thails
   lcUpLeft  = 'ï'
   lcUpRight = 'ð'
   lcDnLeft  = 'Ø'
   lcDnRight = 'Ù'
   lcSideLn  = 'Þ'
   lcDashLn  = 'ß'
CASE lnThai=2        && vThai
   lcUpLeft  = '•'
   lcUpRight = '—'
   lcDnLeft  = '˜'
   lcDnRight = '™'
   lcSideLn  = 'š'
   lcDashLn  = '›'
ENDCASE
@ xnRow1,xnCol1 CLEAR TO xnRow2,xnCol2
@ xnRow1,xnCol1 FILL TO xnRow2,xnCol2 COLOR (xcColor2)
@ xnRow1,xnCol1 SAY lcUpLeft+REPL(lcDashLn,xnCol2-xnCol1-1) COLOR (xcColor1)
```

```
@ xnRow1,xnCol2 SAY lcUpRight COLOR (xcColor2)
@ xnRow2,xnCol1 SAY lcDnLeft  COLOR (xcColor1)
@ xnRow2,xnCol1+1 SAY REPL(lcDashLn,xnCol2-xnCol1-1)+lcDnRight COLOR (xcColor2)
FOR I = xnRow1+1 TO xnRow2-1
  @ I,xnCol1 SAY lcSideLn COLOR (xcColor1)
  @ I,xnCol2 SAY lcSideLn COLOR (xcColor2)
NEXT
RETURN

**********************************
*      Procedure : TITLE         *
**********************************
PROCEDURE Title
PARAMETER xcTitle
SET SYSMENU OFF
DEACTIVATE WINDOW ALL
CLEAR
ACTIVATE SCREEN
@ 0,0 SAY PADC (ccCompName,cnMaxCol) COLOR B/GB
@ 1,0 SAY PADC (xcTitle,cnMaxCol) COLOR GR+/B
DO CASE
CASE MONTH(DATE()) = 2
  vcSpace = SPACE(3)
CASE MONTH(DATE()) = 6
  vcSpace = SPACE(2)
CASE MONTH(DATE())=1.OR.MONTH(DATE())=4.OR.MONTH(DATE())=5.OR.MONTH(DATE())=7
  vcSpace = ''
OTHERWISE
  vcSpace = SPACE(1)
ENDCASE
@ 2,0 SAY PADC (vcSpace+ThDate(DATE()),cnMaxCol)
RETURN

**********************************
*     Function : SAYMESS()       *
**********************************
```

```
FUNCTION SayMess
PARAMETER xnRow,xcMess,xcColor
ACTIVATE SCREEN
@ xnRow,0 SAY PADC(xcMess,cnMaxCol) COLOR (xcColor)
RETURN .T.

**********************************
*       Procedure : ERRHAND      *
**********************************
PROCEDURE Errhand
PARAMETER xcError, xcMess, xcMess1, xcPrgName, xnLineNo
SET PRINT OFF
SET CONSOLE ON
SET DEVICE TO SCREEN
DEFINE WINDOW wcError FROM 06,00 TO 14,78 SHADOW COLOR SCHEME 10
DEFINE WINDOW wcPrint FROM 14,24 TO 18,54 SHADOW COLOR SCHEME 7
DEFINE WINDOW wcDisk  FROM 12,21 TO 16,58 SHADOW COLOR SCHEME 10
POP KEY ALL
?? cbBell
?? cbBell
DO CASE
CASE xcError=125
  ACTIVATE WINDOW wcPrint
  @ 0,0 CLEAR TO 2,30
  @ 0,0 SAY SPACE(6)+'เครื่องพิมพ์ไม่พร้อม...'
  @ 2,1 GET lnSelect FUNCTION '*RTH ลองใหม่  ;ยกเลิก   ' DEFAULT 2 ;
    MESSAGE SPACE(7)+'ลองพิมพ์ใหม่ หรือ ยกเลิกการพิมพ์'
  READ CYCLE
  IF lnSelect=1
    RETRY
  ENDIF
CASE xcError=56
  ACTIVATE WINDOW wcDisk
  @ 1,3 SAY 'เหลือที่ว่างบน Disk ไม่เพียงพอ !!'
  READ
  SET DEFAULT TO (ccPrgPath)
```

```
  CASE xcError=202
    ACTIVATE WINDOW wcDisk
    @ 1,5 SAY 'หา Drive หรือ Path ไม่พบ !!'
    READ
    SET DEFAULT TO (ccPrgPath)
  CASE xcError=1705
    ACTIVATE WINDOW wcDisk
    @ 1,5 SAY 'แผ่น Disk write protect !!'
    READ
    SET DEFAULT TO (ccPrgPath)
  OTHERWISE
    ACTIVATE WINDOW wcError
    ? '                    *** โปรแกรมขัดข้อง !! ***'
    ? 'Error Number  : ' + LTRIM(STR(xcError))
    ? 'Error Message : ' + xcMess
    ? 'Line of Error : ' + xcMess1
    ? 'Line Number   : ' + LTRIM(STR(xnLineNo))
    ? 'Program Name  : ' + xcPrgName
    ? '              *** โปรดจดข้อความนี้บอกโปรแกรมเมอร์ ***'
    READ
  ENDCASE
  IF Esc_key()
    RETURN TO MASTER
  ENDIF
  CLEAR WINDOWS
  RELEASE WINDOWS ALL
  CLEAR
  RETURN TO MASTER

  *** Procedure & Function Accept key
  ************************************
  *      Function : ESC_KEY()        *
  ************************************
  FUNCTION Esc_key
  IF LASTKEY()=27 .OR. READKEY()=20
    RETURN .T.
```

```
ENDIF
RETURN .F.

**********************************
*      Function : TIME_OUT()          *
**********************************
FUNCTION Time_Out
IF READKEY()=20
  RETURN .T.
ENDIF
RETURN .F.

**********************************
*      Function : TDTOC()             *
**********************************
FUNCTION TDtoC
PARAMETER xdDate
RETURN IIF(DAY(xdDate)=0,'',SUBSTR(DTOC(xdDate),1,6)+SUBSTR(STR(YEAR(xdDate)+543,4),3,2))

**********************************
*      Function : TCTOD()             *
**********************************
FUNCTION TCtoD
PARAMETER xcDate
RETURN CTOD(SUBSTR(xcDate,1,6)+SUBSTR(STR(VAL('25'+SUBSTR(xcDate,7,2))-543,4),3,2))

**********************************
*      Function : CHKDATE()           *
**********************************
FUNCTION ChkDate
PARAMETER xcDate
IF TCtoD(xcDate)=CTOD('  /  /  ')
  RETURN .F.
ENDIF
RETURN .T.
```

```
************************************
*     Function : THDATE()          *
************************************
FUNCTION thDate
PARAMETER xdDate
RETURN (LTRIM(STR(DAY(xdDate),2))+' '+thMONTH(MONTH(xdDate))+' '+;
      STR(YEAR(xdDate)+543,4))

************************************
*     Function : TCDATE()          *
************************************
FUNCTION tcDate
PARAMETER xdDate
RETURN (LTRIM(STR(DAY(xdDate),2))+' '+tcMonth(MONTH(xdDate))+' ';
      +SUBSTR(STR(YEAR(xdDate)+543,4),3,2))

************************************
*     Function : THMONTH()         *
************************************
FUNCTION ThMonth
PARAMETER xnMONTH
DIMENSION vaMonth(12)
vaMonth(1)='มกราคม'
vaMonth(2)='กุมภาพันธ์'
vaMonth(3)='มีนาคม'
vaMonth(4)='เมษายน'
vaMonth(5)='พฤษภาคม'
vaMonth(6)='มิถุนายน'
vaMonth(7)='กรกฎาคม'
vaMonth(8)='สิงหาคม'
vaMonth(9)='กันยายน'
vaMonth(10)='ตุลาคม'
vaMonth(11)='พฤศจิกายน'
vaMonth(12)='ธันวาคม'
RETURN vaMonth(xnMonth)
```

```
**********************************
*      Function : TCMONTH()        *
**********************************
FUNCTION tcMonth
PARAMETER xnMonth
DIMENSION vaMonth(12)
vaMonth(1)='ม.ค.'
vaMonth(2)='ก.พ.'
vaMonth(3)='มี.ค.'
vaMonth(4)='เม.ย.'
vaMonth(5)='พ.ค.'
vaMonth(6)='มิ.ย.'
vaMonth(7)='ก.ค.'
vaMonth(8)='ส.ค.'
vaMonth(9)='ก.ย.'
vaMonth(10)='ต.ค.'
vaMonth(11)='พ.ย.'
vaMonth(12)='ธ.ค.'
RETURN vaMonth(xnMonth)

**********************************
*      Function : THIDX()          *
**********************************
FUNCTION ThIDX
PARAMETER xcTag
vcTag=xcTag
IF 'เ' $ xcTAG
  vnNum=AT('เ',vcTAG)
  vcTag=SUBSTR(vcTAG,1,vnNum-1)+SUBSTR(vcTAG,vnNum+1,1)+SUBSTR(vcTAG,vnNum,1)+
SUBSTR(vcTAG,vnNum+2,LEN(xcTAG))
ENDIF
IF 'แ' $ xcTAG
  vnNum=AT('แ',vcTAG)
  vcTag=SUBSTR(vcTAG,1,vnNum-1)+SUBSTR(vcTAG,vnNum+1,1)+SUBSTR(vcTAG,vnNum,1)+
SUBSTR(vcTAG,vnNum+2,LEN(xcTAG))
ENDIF
```

```
IF 'ไ' $ xcTAG
   vnNum=AT('ไ',vcTAG)
   vcTag=SUBSTR(vcTAG,1,vnNum-1)+SUBSTR(vcTAG,vnNum+1,1)+SUBSTR(vcTAG,vnNum,1)+
SUBSTR(vcTAG,vnNum+2,LEN(xcTAG))
ENDIF
IF 'ไ' $ xcTAG
   vnNum=AT('ไ',vcTAG)
   vcTag=SUBSTR(vcTAG,1,vnNum-1)+SUBSTR(vcTAG,vnNum+1,1)+SUBSTR(vcTAG,vnNum,1)+
SUBSTR(vcTAG,vnNum+2,LEN(xcTAG))
ENDIF
IF 'ไ' $ vcTAG
   vnNum=AT('ไ',vcTAG)
   vcTag=SUBSTR(vcTAG,1,vnNum-1)+SUBSTR(vcTAG,vnNum+1,1)+SUBSTR(vcTAG,vnNum,1)+
SUBSTR(vcTAG,vnNum+2,LEN(xcTAG))
ENDIF
RETURN vcTag
```

```
**********************************************
*       Procedure file : SETINIT.PRG          *
**********************************************
CLOSE ALL
SET ESCAPE OFF
SET TALK OFF
SET SCORE OFF
SET STAT OFF
SET INTEN OFF
SET DELI TO '[]'
SET DELI ON
SET DATE DMY
*SET CENT ON
SET HOURS TO 24
SET CLOCK TO 0,70
SET CLOCK OFF
SET DELETED ON
SET REFRE TO 10
SET FUNCTION F1 TO ''
SET FUNCTION F2 TO ''
SET FUNCTION F3 TO ''
SET FUNCTION F4 TO ''
SET FUNCTION F5 TO ''
SET FUNCTION F6 TO ''
SET FUNCTION F7 TO ''
SET FUNCTION F8 TO ''
SET FUNCTION F9 TO ''
SET FUNCTION F10 TO ''
SET RESO  OFF
SET EXCLU OFF
SET SYSMENU OFF
RETURN
*: EOF: SETINIT.PRG
```

```
*********************************************
*          PROCEDURE FILE : PUBLIC.PRG      *
*********************************************
PUBLIC ccPrgPath, ccDataPath,  ccCompName, ccCompeng,  ccWallChr, ccBlnkChr,;
      cnF2, cnF3, cnF4, cnF5, cnEsc, cbBell, cnCtrlW, cnEnter, cnBkSp, cnLeft, cnDel,;
      cnRight, cnDown, cnUp, cnPgDn, cnPgUp, cnEnd, cnHome, cnDelay, cnMaxCol, ;
      cbNormal, cbCpi_10, cbCpi_12, cbCpi_15, cbCondense,;
      cckey, ccEsc0, ccEsc1, ccEnter, ccF2, ccF3, ccF4, ccF5, ccF9,;
      gcEvent, gcMsg, gcChkCmd, gnThresh, gcLastMsg,;
      ccComPort, ccComAddr, ccComIRQn, ccComBaud, ccComPrty, ccComDBts, ;
      ccComFlow, ccOpenCmd, ccCloseCmd, ccTranHow, ccTranFile, cnTimeOut
RETURN

*: EOF: PUBLIC.PRG
```

```
*********************************************
*      PROCEDURE FILE : CONSTANT.PRG        *
*********************************************

  **********************************
  *          LastKey Code          *
  **********************************
  cnF1    = 28
  cnF2    = -1
  cnF3    = -2
  cnF4    = -3
  cnF5    = -4
  cnF6    = -5
  cnF7    = -6
  cnF8    = -7
  cnF9    = -8
  cnF10   = -9
  cnHome  = 1
  cnPgDn  = 3
  cnPgUp  = 18
  cnLeft  = 19
  cnRight = 4
  cnUp    = 5
  cnDown  = 24
  cnEnd   = 6
  cnDel   = 7
  cnIns   = 22
  cnEsc   = 27
  cnEnter = 13
  cnCtrlN = 14
  cnCtrlW = 23
  cnBkSp  = 127

  **********************************
  *          Printer Code          *
  **********************************
```

```
cbInitPrt = CHR(27)+"@"

cbCR = CHR(13)
cbFF = CHR(12)
cbLF = CHR(10)

cbCPI_10 =  CHR(27)+"P"
cbCPI_12 =  CHR(27)+"M"
cbCPI_15 =  CHR(27)+"g"
cbDraft  =  CHR(27)+"x"+CHR(0)
cbLQ     =  CHR(27)+"x"+CHR(1)

cbNormal  = CHR(18)
cbCondense = CHR(15)

cbItalic  = CHR(27)+"4"
cbItlcOff = CHR(27)+"5"
cbBold    = CHR(27)+"E"
cbBdOff   = CHR(27)+"F"
cbEnlarge = CHR(27)+"W"+CHR(1)
cbEnOff   = CHR(27)+"W"+CHR(0)

cbNormal0 = CHR(27)+"q"+CHR(0)
cbOutLine = CHR(27)+"q"+CHR(1)
cbShadow  = CHR(27)+"q"+CHR(2)
cbOutShdw = CHR(27)+"q"+CHR(3)

**********************************
*        Other  Constant         *
**********************************
ccEsc0  = '[Esc]-ออก'
ccEsc1  = '[Esc]-ยกเลิก'
ccEnter = '[Enter]-เลือก'
ccF2  = '[F2]-ดูชื่อ'
ccF3  = '[F3]-ค้นหา'
ccF4  = '[F4]-เลือกข้อมูล'
```

```
ccF5  = '[F5]-บันทึกข้อมูล'
ccF9  = '[F9]-พิมพ์'
ccKey = '[],[],[PgUp],[PgDn],[Home],[End]-เลื่อน'

cbBell  = CHR(7)
cnMaxCol = 80
cnDelay  = 200
cnDelay2 = 2000

ccCompName = SPACE(6)+'สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง'
ccCompEng  = 'KING MONGKUT INSTITUE TECHNOLOGY LADKRABANG'


**********************************
*      Directory  Constant              *
**********************************
IF SYS(5) = "C:"
   ccPrgPath  = 'C:'+CURDIR()
   ccDataPath = ccPrgPath+'DATA'
   SET PATH TO (ccDataPath)
ELSE
   IF SYS(5) = "D:"
      ccPrgPath  = 'D:'+CURDIR()
      ccDataPath = ccPrgPath+'DATA'
      SET PATH TO (ccDataPath)
   ELSE
      SET PATH TO 'DATA'
   ENDIF
ENDIF

lnThai=1
DO CASE
CASE lnThai=1       && For THAILS
   STORE '□' TO ccWallChr
   STORE ' ' TO ccBlnkChr
CASE lnThai=2       && For VTHAI
```

```
    STORE '' TO ccWallChr
    STORE ' ' TO ccBlnkChr
  CASE lnThai=3       && For AxTHAI
    STORE '—' TO ccWallChr
    STORE ' ' TO ccBlnkChr
  ENDCASE

RETURN

*: EOF: CONSTANT.PRG
```

```
*********************************************
*       PROCEDURE FILE : NETRECV.PRG        *
*********************************************
DEFINE WINDOW wsTalk1 FROM 06,11 TO 17,68 DOUBLE SHADOW FLOAT COLOR SCHEME 5
*
USE DateTime IN 1 EXCLU
*CREATE CURSOR Temp1 (cTemp C(20))
*CREATE CURSOR Temp2 (cTemp C(20))
USE Temp1 IN 2
USE Temp2 IN 3
*
llRecv = .T.
DO WHILE llRecv
  SET SAFETY OFF
  SELE Temp1
  ZAP
  SELE Temp2
  ZAP
  SET SAFETY ON

  CALL COMETMP WITH ccCloseCmd
  CALL COMETMP WITH ccOpenCmd
  *
  gcMsg = "Ready to receive ! Esc - Exit !"
  gcLastMsg = gcMsg
  NULL = ShowMess(gcMsg)
  ACTIVATE WINDOW wsTalk1
  ? " Ready to receive..."
  ?
  lnMaxNode = 10
  FOR I = 1 TO lnMaxNode
    ACTIVATE WINDOW wsTalk1
    ? 'NODE ID : '+TRAN(I,'99')
    ?
    J=0
    DO WHILE .T.
```

```
      lcOutput = "OUTPUT #"+ccComPort+","+CHR(I)
      CALL COMETMP WITH lcOutput
      *
      lcInput = "INPUT #"+ccComPort+","  + SPACE(100)
      CALL COMETMP WITH lcInput
      *
      lnAmtRetd = VAL(SUBSTR(lcInput,10,5))
      llCOMactive = IIF(lnAmtRetd > 0, .T., .F.)
      *
      IF llCOMactive
        lcComData = SUBSTR(lcInput,15,lnAmtRetd)
        SELE TEMP1
        APPEN BLANK
        REPL cTemp WITH lcComData
        IF SUBSTR(lcComData,LEN(lcComData),1)=CHR(255)
          EXIT
        ENDIF
        IF lcComData # CHR(7)
          ?? lcComData
        ENDIF
        IF lcComData = CHR(I)
          J = J+1
        ENDIF
      ELSE
        EXIT
      ENDIF
      lnKey = INKEY()
      IF lnkey = cnEsc .OR. J>19
        EXIT
      ENDIF
    ENDDO
  NEXT

  lcTemp = ''
  lnLen = 0
  lnRec = 20
```

```
llFile = .F.
SELE TEMP1
SCAN
  lcTemp = lcTemp+ALLTRIM(Temp1.cTemp)
  lnLen = lnLen+LEN(ALLTRIM(Temp1.cTemp))
  IF lnLen >= lnRec
    SELE TEMP2
    APPEN BLANK
    REPL cTemp WITH SUBSTR(lcTemp,1,lnRec)
    lcTemp = SUBSTR(lcTemp,lnRec+1,LEN(lcTemp)-lnRec+1)
    lnLen = LEN(lcTemp)
    llFile = .T.
  ENDIF
ENDSCAN
*
IF llFile
  ACTIVATE WINDOW wsTalk1
  ? ' Save File...'
  SELE TEMP2
  GO TOP
  SCAN
    WAIT 'Save File...Please Wait' WINDOW NOWAIT
    lcDate=SUBSTR(Temp2.cTemp,10,2)+'/'+SUBSTR(Temp2.cTemp,12,2)+'/'+SUBSTR
(Temp2.cTemp,14,2)
    ldDate=CTOD(lcDate)
    IF DTOC(ldDate) # '  /  /  '
      SELE DateTime
      APPE BLANK
      REPL cID WITH SUBSTR(Temp2.cTemp,2,9),;
        dDate WITH ldDate,;
        cTime WITH SUBSTR(Temp2.cTemp,16,2)+':'+SUBSTR(Temp2.cTemp,18,2)
      *llRecv = .F.
    ELSE
      *llRecv = .T.
    ENDIF
  ENDSCAN
```

```
    ? ' Tranfer data complete...'
    WAIT ' สิ้นสุดการรับข้อมูล...' WINDOW NOWAIT
  ENDIF
  llRecv = .F.
*
ENDDO
RELEASE WINDOW wsTalk1
CALL COMETMP WITH 'ONTIME '
RETURN

**************************************
*      Function : SHOWMESS()         *
**************************************
FUNCTION ShowMess
PARAMETERS xcMsg
ACTIVATE SCREEN
@ gnRMax-1,gnCol+3 SAY PADC(xcMsg,gnCMax-5) COLOR N/W
ACTIVATE WINDOW wsTalk1
RETURN

*: EOF: NETRECV.PRG
```

```
**********************************************
*      PROCEDURE FILE : RECVDATA.PRG         *
**********************************************

DEFINE WINDOW wsTalk1 FROM 19,07 TO 21,72 DOUBLE SHADOW FLOAT COLOR SCHEME 5
USE DateTime IN 1 EXCLU
CREATE CURSOR Temp (cTemp C(19))
*
lnData  = LEN(cTemp)
llError = .T.
lcTemp  = SPACE(lnData)
lcBreak = REPL('?',lnData-1)
lcEnd   = REPL('*',lnData-1)
*
ACTIVATE WINDOW wsTalk1
? ' กำลังรับข้อมูลจากเครื่องบันทึกเวลา...'
*
DO WHILE .T.
  WAIT '  คอยสักครู่กำลังรับข้อมูล...[Esc]-ยกเลิก' WINDOW NOWAIT
  CALL RecvData WITH lcTemp
  ? ' ข้อมูล : '+SUBSTR(lcTemp,1,lnData-1)
  IF !EMPTY(lcTemp)
    DO CASE
    CASE SUBSTR(lcTemp,1,lnData-1) = lcBreak
      llError=.T.
      EXIT
      WAIT ' ยกเลิกการรับข้อมูล !!' WINDOW NOWAIT
    CASE SUBSTR(lcTemp,1,lnData-1) = lcEnd
      llError=.F.
      EXIT
    OTHERWISE
      SELE TEMP
      APPE BLANK
      REPL cTemp WITH lcTemp
    ENDCASE
  ELSE
```

```
      EXIT
      WAIT ' การรับข้อมูลผิดพลาด !!' WINDOW NOWAIT
    ENDIF
  ENDDO
  *
  SELE TEMP
  IF !llError
    ACTIVATE WINDOW wsTalk1
    ? ' กำลังบันทึกลงแฟ้มข้อมูล...'
    SELE TEMP
    GO TOP
    SCAN
      WAIT '  คอยสักครู่กำลังบันทึกข้อมูล...' WINDOW NOWAIT
      lcDate=SUBSTR(Temp.cTemp,9,2)+'/'+SUBSTR(Temp.cTemp,11,2)+'/'+SUBSTR(Temp.cTemp,13,2)
      ldDate=CTOD(lcDate)
      IF DTOC(ldDate) # '  /  /  '
        SELE DateTime
        APPE BLANK
        REPL cID WITH SUBSTR(Temp.cTemp,1,8),;
            dDate WITH ldDate,;
            cTime WITH SUBSTR(Temp.cTemp,15,2)+':'+SUBSTR(Temp.cTemp,17,2)
      ENDIF
    ENDSCAN
    ? ' การส่งผ่านข้อมูลสมบูรณ์...'
    WAIT ' สิ้นสุดการรับข้อมูล...' WINDOW NOWAIT
  ELSE
    IF EMPTY(lcTemp)
      WAIT ' การรับข้อมูลผิดพลาด !!' WINDOW NOWAIT
    ELSE
      WAIT ' ยกเลิกการรับข้อมูล !!' WINDOW NOWAIT
    ENDIF
  ENDIF
  CLOSE DATA
  RETURN

  *: EOF: RECVDATA.PRG
```

```
**********************************************
*       PROCEDURE FILE : SETTIME.PRG         *
**********************************************
DEFINE WINDOW wrGET  FROM 05,28 TO 08,50 DOUBLE SHADOW COLOR SCHEME 2
CALL CUR_ON
DO WHILE .T.
  DO TITLE WITH SPACE(6)+'ตั้งเวลาเครื่องบันทึกเวลา'
  lcTime  = SUBSTR(TIME(),1,5)
  lcDate  = TDTOC(DATE())
  lnData  = 11
  lcDTSet = SPACE(lnData)
  lcBreak = REPL('?',lnData)
  lcEnd   = REPL('*',lnData)
  lcCheck = '0'
  ACTIVATE WINDOW wrGET
  @ 00,01 SAY 'เวลา    [  :  ]'
  @ 01,01 SAY 'วันที่  [  /  /  ]'
  @ 00,09 GET lcTime  PICT '99:99' ;
    MESS 'บอกเวลาที่ต้องการบันทึกหรือแก้ไข  [Esc]-ออก'
  @ 01,09 GET lcDate PICT '99/99/99' VALID ChkDate(lcDate);
    ERROR SPACE(5)+'วันที่ไม่ถูกต้อง !!' ;
    MESS 'บอกวันที่ที่ต้องการบันทึกหรือแก้ไข  [Esc]-ออก'
  READ TIMEOUT(cnDelay) COLOR SCHEME 1
  IF Esc_key()
    EXIT
  ENDIF
  lcDate = DTOC(TCTOD(lcDate))
  lcDate = SUBSTR(lcDate,1,2)+SUBSTR(lcDate,4,2)+SUBSTR(lcDate,7,2)
  lcDTSet = lcDate+SUBSTR(lcTime,1,2)+SUBSTR(lcTime,4,2)+lcCheck
  WAIT ' คอยสักครู่กำลังส่งข้อมูล...' WINDOW NOWAIT
  CALL SetTime WITH lcDTSet
  IF EMPTY(lcDTset)
    WAIT ' การตั้งเวลาผิดพลาด !!' WINDOW NOWAIT
  ELSE
    IF lcDTSet = lcBreak
      WAIT ' ยกเลิกการตั้งเวลา !!' WINDOW NOWAIT
```

```
      ELSE
        WAIT ' การตั้งเวลาสมบูรณ์...' WINDOW NOWAIT
      ENDIF
      EXIT
    ENDIF
ENDDO
RETURN

*: EOF: SETTIME.PRG
```

```
*********************************************
*       PROCEDURE FILE : LREPORT.PRG        *
*********************************************
CLOSE ALL
*
USE Student  IN 1
USE DateTime IN 2
*USE RepFile  IN 3 EXCLU
*
DEFINE WINDOW wrFunc1 FROM 03,02 TO 07,27 SHADOW COLOR SCHEME 4
DEFINE WINDOW wrFunc2 FROM 04,57 TO 07,76 SHADOW COLOR SCHEME 4
DEFINE WINDOW wrDate  FROM 04,31 TO 07,53 SHADOW COLOR SCHEME 2
DEFINE WINDOW wbRep1  FROM 09,02 TO 22,76 SHADOW COLOR SCHEME 13 ;
  TITLE SPACE(4)+' '
DEFINE WINDOW wbRep2  FROM 09,01 TO 22,77 SHADOW COLOR SCHEME 15 ;
  TITLE SPACE(4)+' '
DEFINE WINDOW wrName  FROM 03,29 TO 07,55 SHADOW COLOR SCHEME 2
*
CALL Cur_On
lcStart = '09:30'
lcStop  = '15:30'
*
DO WHILE .T.
  CREAT CURSOR MonFile (cID C(8), nIN N(2), nOUT N(2), nLate N(2), nNON N(2))
  DO TITLE WITH SPACE(5)+'รายงานการเข้าห้องปฏิบัติการ'
  STORE 1 TO lnChoice1,lnChoice2
  STORE .F. TO llFound
  STORE MONTH(DATE()) TO lnMonth
  STORE YEAR(DATE())+543 TO lnYear
  STORE DATE() TO ldDate,ldDate1,ldDate2
  STORE '' TO lcSeek,lcName,lcDate,lcDate1,lcDate2,lcTitle
  ACTIVATE WINDOW wrFunc1
  @ 0,0 GET lnChoice1 FUNCTION ;
    '*RTV รายงานประจำวัน     ;รายงานประจำเดือน    ;รายงานเฉพาะราย     ';
    DEFAULT 1 MESSAGE 'เลือกประเภทของรายงานที่ต้องการดู  [Esc]-ออก'
  READ CYCLE TIMEOUT(cnDelay)
```

```
IF Esc_key()
   EXIT
ENDIF
*
SELE RepFile
SET SAFETY OFF
ZAP
SET SAFETY ON
ACTIVATE WINDOW wrDate
DO CASE
CASE lnChoice1=1
   SELE DateTime
   SET ORDER TO dDate
   lcDate = tdtoc(ldDate)
   @ 0,1 SAY 'วันที่ '
   @ 0,9 GET lcDate PICT '99/99/99' ;
      VALID SeekDate(lcDate) .OR. Time_Out();
      ERROR ' ไม่มีข้อมูลวันที่ '+lcDate ;
      MESSAGE 'บอกวันที่ที่ต้องการดูรายงาน  [Esc]-ยกเลิก'
   READ COLOR SCHEME 1 TIMEOUT(cnDelay)
   IF Esc_key()
      LOOP
   ENDIF
   ldDate = Tctod(lcDate)
   lcSeek = DTOS(ldDate)
   lcTitle = 'รายงานการเข้าห้องปฏิบัติการประจำวันที่ '+LTRIM(thdate(ldDate))
   SELE DateTime
   SET ORDER TO dDATE
   SEEK lcSeek
   SCAN WHILE DTOS(dDate)=lcSeek
      SELE RepFile
      SET ORDER TO cID
      IF !SEEK(DateTime.cID)
         APPEND BLANK
         REPL cID WITH DateTime.cID,;
            dDate WITH DateTime.dDate,;
```

```
        cT_IN WITH DateTime.cTime
      ELSE
        IF EMPTY(cT_OUT)
          REPL cT_OUT WITH DateTime.cTime
        ENDIF
      ENDIF
    ENDSCAN
*
CASE lnChoice1=2
  SELE DateTime
  SET ORDER TO dDate
  @ 0,01 SAY 'เดือน    '
  @ 1,01 SAY 'ปี พ.ศ.  '
  @ 0,10 GET lnMonth PICT '99' RANG 1,12 ;
    MESSAGE 'บอกเดือนที่ต้องการดูรายงาน  [Esc]-ยกเลิก'
  @ 1,10 GET lnYear PICT '9999' RANG 2535,2999 ;
    MESSAGE 'บอกปีที่ต้องการดูรายงาน  [Esc]-ยกเลิก'
  READ COLOR SCHEME 1 TIMEOUT(cnDelay)
  IF Esc_key()
    LOOP
  ENDIF
  lcTitle='รายงานการเข้าห้องปฏิบัติการประจำเดือน '+TRIM(thmonth(lnMonth))+' '+STR(lnYear,4)
  lnYear=lnYear-543
  lcSeek=STR(lnYear,4)+IIF(lnMonth>9,STR(lnMonth,2),'0'+STR(lnMonth,1))
  IF SEEK(lcSeek)
    SELE Student
    SET ORDER TO cID
    SCAN
      lcID = Student.cID
      SELE DateTime
      SET ORDER TO cID
      SEEK lcID
      SCAN WHILE cID=lcID FOR DTOS(dDate)=lcSeek
        SELE RepFile
        SET ORDER TO cID
        IF !SEEK(lcID+DTOS(DateTime.dDate))
```

```
        APPEND BLANK
        REPL cID WITH DateTime.cID,;
          dDate WITH DateTime.dDate,;
          cT_IN WITH DateTime.cTime
      ELSE
        IF EMPTY(cT_OUT)
          REPL cT_OUT WITH DateTime.cTime
        ENDIF
      ENDIF
    ENDSCAN
    *
    SELE MonFile
    APPEN BLANK
    REPL cID WITH lcID
  ENDSCAN
  *
  SELE MonFile
  SCAN
    STORE 0 TO lnIN,lnLate,lnOUT,lnNon
    *
    SELE RepFile
    SET ORDER TO cID
    SEEK MonFile.cID
    SCAN WHILE cID=MonFile.cID
      IF cT_IN > lcStart
        lnLate = lnLate+1
      ELSE
        lnIN = lnIN+1
      ENDIF
      IF EMPTY(cT_OUT)
        lnNON = lnNON+1
      ELSE
        IF cT_OUT < lcStop
          lnOUT = lnOUT+1
        ENDIF
      ENDIF
```

```
        ENDSCAN
        *
        SELE MonFile
        REPL nIN WITH lnIN,;
          nLate WITH lnLate,;
           nOUT WITH lnOut,;
           nNon WITH lnNon
      ENDSCAN
    ELSE
      WAIT ' ไม่มีข้อมูลเดือน '+TRIM(tcmonth(lnMonth))+' '+STR(lnYear+543,4) ;
        WINDOW NOWAIT
      LOOP
    ENDIF
  CASE lnChoice1=3
    ACTIVATE WINDOW wrName
    SELE Student
    SET ORDER TO cID
    SCATTER MEMVAR BLANK
    @ 0,1 SAY 'รหัส  '
    @ 0,14 GET m.cID PICT '@!' ;
      VALID !EMPTY(ScanID(m.cID)) .OR. Time_out() ;
      MESSAGE 'บอกรหัสนักศึกษาที่ต้องการ     [Esc]-ยกเลิก' ;
      ERROR SPACE(5)+'ยกเลิกการค้นหารหัส !!'
    READ COLOR SCHEME 1 TIMEOUT(cnDelay)
    IF Esc_key()
      LOOP
    ENDIF
    SCATTER MEMVAR
    @ 0,14 SAY '['+m.cID+']'
    lcSeek = m.cID
    lcName = TRIM(m.cPFIX)+TRIM(m.cNAME)+' '+TRIM(m.cSNAME)
    lcTitle='รายงานการเข้าห้องปฏิบัติการ : '+lcName
    WAIT SPACE(3)+'กำลังหาช่วงวันที่...' WINDOW NOWAIT
    SELE DateTime
    SET ORDER TO cID
    SEEK lcSeek
```

```
STORE dDate TO ldDate1,ldDate2
SCAN WHILE cID=lcSeek
  ldDate1=IIF(ldDate1<=dDate,ldDate1,dDate)
  ldDate2=IIF(ldDate2>=dDate,ldDate2,dDate)
  llFound=.T.
ENDSCAN
WAIT CLEAR
IF !llFound
  WAIT ' ไม่พบข้อมูลของ : '+lcName WINDOW NOWAIT
  LOOP
ENDIF
STORE Tdtoc(ldDate1) TO lcDate1
STORE Tdtoc(ldDate2) TO lcDate2
@ 01,01 SAY 'จากวันที่ '
@ 02,01 SAY 'ถึงวันที่ '
@ 01,14 GET lcDate1 PICT '99/99/99' VALID ChkDate(lcDate1) .OR. Time_Out();
 MESSAGE 'บอกวันที่เริ่มต้นของรายงาน [Esc]-ยกเลิก' ;
 ERROR SPACE(4)+'วันที่ไม่ถูกต้อง !!'
@ 02,14 GET lcDate2 PICT '99/99/99' VALID ChkDate(lcDate2) .OR. Time_Out();
  MESSAGE 'บอกวันที่สุดท้ายของรายงาน [Esc]-ยกเลิก' ;
  ERROR SPACE(4)+'วันที่ไม่ถูกต้อง !!'
READ COLOR SCHEME 1 TIMEOUT(cnDelay)
STORE tctod(lcDate1) TO ldDate1
STORE tctod(lcDate2) TO ldDate2
IF Esc_key()
  LOOP
ENDIF
SELE DateTime
SET ORDER TO cID
SEEK lcSeek
SCAN WHILE cID=lcSeek FOR BETWEEN(dDATE,ldDate1,ldDate2)
  SELE RepFile
  SET ORDER TO dDate
  IF !SEEK(DTOS(DateTime.dDate))
    APPEND BLANK
    REPL cID WITH DateTime.cID,;
```

```
        dDate WITH DateTime.dDate,;
        cT_IN WITH DateTime.cTime
      ELSE
        IF EMPTY(cT_OUT)
          REPL cT_OUT WITH DateTime.cTime
        ENDIF
      ENDIF
    ENDSCAN
  ENDCASE

  ACTIVATE WINDOW wrFunc2
  @ 0,0 GET lnChoice2 FUNCTION '*RTV จอภาพ        ;เครื่องพิมพ์ ' DEFAULT 1 ;
    MESSAGE 'เลือกว่าจะ ดูทางจอภาพ หรือ พิมพ์ออกเครื่องพิมพ์ [Esc]-ยกเลิก'
  READ CYCLE TIMEOUT(cnDelay)
  IF Esc_key()
    LOOP
  ENDIF
  IF lnChoice2=1
    ACTIVATE SCREEN
    =SayMess(24,ccKey+'  '+ccEsc0,'GB+/GR+')
    PUSH KEY
    GO TOP
    ON KEY LABEL Home GO TOP
    ON KEY LABEL End  GO BOTTOM
    IF lnChoice1 = 2
      SELE MonFile
      BROW FIELD cID:H='รหัส  ',;
        cName=NAME(cID):H=' นักศึกษา  ':22,;
        nIN:H=' มาทัน  ':P='99   ',;
        nlate:H=' มาสาย  ':P='99   ',;
        nOUT:H=' ออกก่อน  ':P='99   ',;
        nNon:H=' ไม่ลงเวลา  ':P='99   ' ;
        WINDOW wbRep2 NOMODIFY TIMEOUT(cnDelay)
    ELSE
      SELE RepFile
      SET ORDER TO cID
```

```
        BROW FIELD cID:H='รหัส ':9,;
          cName=NAME(cID):H=' นักศึกษา ':29,;
          dd=' '+tdtoc(dDate)+' ':H=' วันที่ ':10,;
          cTIN=' '+cT_IN:H='เวลาเข้า ',;
          cTOUT=' '+cT_OUT:H='เวลาออก ' ;
          WINDOW wbRep1 NOMODIFY TIMEOUT(cnDelay)
      ENDIF
    ELSE
      WAIT ' พิมพ์รายงาน...' WINDOW NOWAIT
      SET PRINT ON
      ??? cbNORMAL
      ??? cbCPI_12
      IF lnChoice1=2
        SELE MonFile
        REPORT FORM MonRep TO PRINT NOCONSOLE
      ELSE
        SELE RepFile
        REPORT FORM DayRep TO PRINT NOCONSOLE
      ENDIF
      SET PRINT OFF
      WAIT CLEAR
    ENDIF
ENDDO
CLOSE DATA
RETURN

*************************************
*      Function : SEEKDATE()        *
*************************************
FUNCTION SeekDate
PARAMETER xcDate
IF ChkDate(xcDate) .AND. SEEK(DTOS(TCtoD(xcDate)))
  RETURN .T.
ENDIF
RETURN .F.
```

```
************************************
*       Function : NAME()          *
************************************
FUNCTION Name
PARAMETER xcID
STORE ALIAS() TO vcAlias
vcName=''
SELE Student
SET ORDER TO cID
IF SEEK(xcID)
  STORE TRIM(cPFIX)+TRIM(cNAME)+' '+TRIM(cSNAME) TO vcName
ENDIF
SELE (vcAlias)
RETURN vcName

************************************
*       Function : SCANID()        *
************************************
FUNCTION ScanID
PARAMETER xcID
DEFINE WINDOW wbStud FROM 12,18 TO 22,70 SHADOW COLOR SCHEME 9 ;
 TITLE SPACE(2)+'นักศึกษา'
ACTIVATE SCREEN
=SayMess(24,ccKey+' '+ccF3+' '+ccEnter+'  '+ccEsc1,'GB+/GR+')
PUSH KEY
IF ! SEEK (xcID) .OR. EMPTY(xcID)
  SET ORDER TO cNAME
  GO TOP
  ON KEY LABEL Home GO TOP
  ON KEY LABEL End  GO BOTTOM
  ON KEY LABEL F3 DO SeekName
  ON KEY LABEL Enter KEYBOARD CHR(23)
  BROW FIELD cID:H='รหัส  ',;
    cNN=TRIM(cPFIX)+TRIM(cNAME)+' '+TRIM(cSNAME):H='ชื่อ  ':40:W=.F. ;
    WINDOW wbStud NOAPPEND NOEDIT NODELETE NORGRID
ENDIF
```

```
STORE cID TO xcID
POP KEY
RELEASE WINDOW wbStud
IF Esc_key()
  xcID = ''
ENDIF
RETURN xcID

*************************************
*       Function : SEEKNAME()       *
*************************************
FUNCTION SeekName
STORE SPACE(LEN(cNAME)) TO xcName
DEFINE WINDOW wrGetName FROM 08,23 TO 10,65 SHADOW COLOR SCHEME 1
ACTIVATE WINDOW wrGetName
@ 0,1 SAY 'ชื่อ '
@ 0,7 GET xcName  MESS 'ใส่ชื่อ (บางส่วนก็ได้)  [Esc]-ยกเลิก'
READ TIMEOUT(cnDelay) COLOR SCHEME 8
=SayMess(24,ccKey+' '+ccF3+' '+ccEnter+'  '+ccEsc1,'GB+/GR+')
RELEASE WINDOW wrGetName
IF Esc_key()
  RETURN .F.
ENDIF
SEEK THIDX(ALLTRIM(xcName))
RETURN .T.

*: EOF: LREPORT.PRG
```

```
**********************************************
*       PROCEDURE FILE : STUDFILE.PRG        *
**********************************************
USE STUDENT IN 1 EXCLU
*
DEFINE WINDOW wrGET  FROM 04,13 TO 12,65 DOUBLE SHADOW COLOR SCHEME 2
DEFINE WINDOW wrFunc FROM 15,20 TO 17,58 SHADOW COLOR SCHEME 4
DEFINE WINDOW wrOK   FROM 19,25 TO 21,53 SHADOW COLOR SCHEME 4
DEFINE WINDOW wsTalk FROM 19,07 TO 21,72 DOUBLE SHADOW FLOAT COLOR SCHEME 5
*
llPack = .F.
CALL Cur_On
DO WHILE .T.
  DO TITLE WITH SPACE(6)+'รายการบันทึกข้อมูลนักศึกษา'
  STORE " TO lcID,lcNameErr
  STORE 1 TO lnChoice,lnOK
  SELE Student
  SET ORDER TO cID
  SCATTER MEMVAR BLANK
  ON KEY LABEL F3 DO SeekName WITH m.cID
  ACTIVATE WINDOW wrGET
  @ 00,01 SAY 'รหัสนักศึกษา  '
  @ 01,01 SAY 'คำนำหน้าชื่อ  [      ]'
  @ 02,01 SAY 'ชื่อนักศึกษา  ['+m.cName+']'
  @ 03,01 SAY 'นามสกุล       ['+m.cSName+']'
  @ 04,01 SAY 'ชั้นปี        ['+m.cClass+']'
  @ 05,01 SAY 'ภาควิชา       ['+m.cDepart+']'
  @ 06,01 SAY 'คณะ           ['+m.cFacult+']'
  @ 00,15 GET m.cID  PICT '@!' VALID Seek_ID(m.cID) .OR. time_out() ;
    'MESS 'บอกรหัสนักศึกษาที่ต้องการบันทึกหรือแก้ไข  [F3]-ค้นหานักศึกษา  [Esc]-ออก';
    ERROR ' ใส่รหัสนักศึกษา !!'
  READ TIMEOUT(cnDelay) COLOR SCHEME 1
  ON KEY LABEL F3
  IF Esc_key()
    EXIT
  ENDIF
```

```
@ 01,15 GET m.cPFIX VALID Prefix(m.cPFIX) .OR. time_out();
  MESS 'บันทึกคำนำหน้าชื่อนักศึกษา  [Esc]-ยกเลิก' ERROR ' ใส่คำนำหน้าชื่อ !!'
@ 02,15 GET m.cNAME VALID !EMPTY(m.cName) .OR. time_out();
  MESS 'บันทึกชื่อนักศึกษา     [Esc]-ยกเลิก' ERROR ' ใส่ชื่อนักศึกษา !!'
@ 03,15 GET m.cSName VALID ChkName(ALLTRIM(m.cNAME)+ALLTRIM(m.cSNAME)) ;
  .OR. time_out() MESS 'บันทึกนามสกุลนักศึกษา  [Esc]-ยกเลิก';
  ERROR lcNameErr+' !!'
@ 04,15 GET m.cClass VALID !EMPTY(m.cClass) .OR. time_out();
  PICT '@!' MESS 'บันทึกชั้นปีที่ศึกษา  [Esc]-ยกเลิก' ERROR ' ใส่ชั้นปี  !!'
@ 05,15 GET m.cDepart VALID Depart(m.cDepart) .OR. time_out();
  MESS 'บันทึกภาควิชา       [Esc]-ยกเลิก' ERROR ' ใส่ภาควิชา !!'
@ 06,15 GET m.cFacult VALID Faculty(m.cFacult) .OR. time_out();
  MESS 'บันทึกคณะ  [Esc]-ยกเลิก' ERROR ' ใส่คณะวิชา !!'
READ TIMEOUT(cnDelay) COLOR SCHEME 1
IF Esc_key() .OR. EMPTY(m.cName)
  LOOP
ENDIF
*
DEACTIVATE WINDOW wsLast
ACTIVATE WINDOW wrFunc
@ 00,01 GET lnChoice FUNCTION '*RTH เพิ่ม/แก้ไข  ;ลบข้อมูล     ';
  MESS 'เลือกว่าต้องการ เพิ่ม/แก้ไขข้อมูล หรือ ลบข้อมูล  [Esc]-ยกเลิก'
READ CYCLE TIMEOUT(cnDelay)
IF Esc_key()
  LOOP
ENDIF
IF lnChoice=2
  ACTIVATE WINDOW wrOK
  @ 0,01 GET lnOK FUNCTION '*RTH ตกลง    ;ยกเลิก  ' DEFAULT 2 ;
    MESSAGE SPACE(10)+'โปรดยืนยันการลบข้อมูลนักศึกษานี้ ตกลง หรือ ยกเลิก'
  READ CYCLE TIMEOUT(cnDelay)
  IF lnOK = 1
    DELETE
    llPack = .T.
  ENDIF
ELSE
```

```
      SET ORDER TO cID
      IF !SEEK(m.cID)
        APPE BLANK
      ENDIF
      GATHER MEMVAR
    ENDIF
  ENDDO
  IF llPack
    =SayMess(24,SPACE(4)+'คอยสักครู่...โปรแกรมกำลังทำงาน','GB+/GR+')
    ACTIVATE WINDOW wsTalk
    SET TALK ON
    ? 'ปรับปรุงแฟ้มข้อมูลนักศึกษา...'
    PACK
    SET TALK OFF
  ENDIF
  CLOSE DATA
  RETURN

  ************************************
  *       Function : SEEKNAME()      *
  ************************************
  FUNCTION SeekName
  PARAMETER xcID
  xcID=ScanStud(xcID)
  IF EMPTY(xcID)
    SCATTER MEMVAR BLANK
    RETURN .F.
  ENDIF
  SCATTER MEMVAR
  DO ShowGet
  RETURN .T.

  ************************************
  *       Function : SCANSTUD()      *
  ************************************
  FUNCTION ScanStud
```

```
PARAMETER xcCode
SELE Student
SET ORDER TO cID
DEFINE WINDOW wbStud FROM 14,07 TO 22,71 SHADOW COLOR SCHEME 9 ;
  TITLE SPACE(3)+'แ ฟ้ ม นั ก ศึ ก ษ า'
ACTIVATE SCREEN
=SayMess(24,ccKey+' '+ccEnter+'  '+ccEsc1,'GB+/GR+')
ON KEY LABEL F3
PUSH KEY
IF ! SEEK (xcCode) .OR. EMPTY(xcCode)
   SET ORDER TO cNAME
   GO TOP
   ON KEY LABEL Home GO TOP
   ON KEY LABEL End  GO BOTTOM
   ON KEY LABEL Enter KEYBOARD CHR(23)
   BROW FIELD cID:H='รหัส  ';;
     cFNAME=TRIM(cPfix)+TRIM(cName)+' '+TRIM(cSName):H='     ชื่อ  ':W=.F.:25,;
     cDepart:H='ภาควิชา  ':W=.F.:20,;
     cClass:H='ปี  ':W=.F. ;
     WINDOW wbStud NOAPPEND NOEDIT NODELETE NORGRID
ENDIF
POP KEY
ON KEY LABEL F3 DO SeekName WITH m.cID
STORE cID TO xcCode
RELEASE WINDOW wbStud
IF Esc_key()
  RETURN SPACE(LEN(xcCODE))
ENDIF
RETURN xcCode

**************************************
*      Function : SEEK_ID()          *
**************************************
FUNCTION Seek_ID
PARAMETER xcID
IF EMPTY(xcID)
```

```
    RETURN .F.
  ENDIF
  SET ORDER TO cID
  IF SEEK(xcID)
    SCATTER MEMVAR
    DO ShowGet
  ELSE
    SCATTER MEMVAR BLANK
    m.cID=xcID
  ENDIF
  RETURN .T.

  ************************************
  *       Function : CHKNAME()       *
  ************************************
  FUNCTION ChkName
  PARAMETER xcName
  IF !EMPTY(m.cSName)
    SET ORDER TO cID
    IF !SEEK(m.cID)
      SET ORDER TO cNAME
      SET EXAC ON
      IF SEEK(ThIDX(xcNAME))
        lcID=cID
        SET EXAC OFF
        lcNameErr=' รายชื่อนักศึกษาซ้ำ รหัส ['+lcID+']'
        RETURN .F.
      ENDIF
      SET EXAC OFF
    ENDIF
  ELSE
    lcNameErr=' ใส่นามสกุล'
    RETURN .F.
  ENDIF
  RETURN .T.
```

```
************************************
*       Procedure : SHOWGET        *
************************************
PROCEDURE ShowGet
ACTIVATE WINDOW wrGET
@ 01,15 GET m.cPFIX
@ 02,15 GET m.cNAME
@ 03,15 GET m.cSNAME
@ 04,15 GET m.cCLASS
@ 05,15 GET m.cDEPART
@ 06,15 GET m.cFACULT
CLEAR GETS
RETURN

************************************
*       Function : PREFIX()        *
************************************
FUNCTION Prefix
PARAMETER xcPfix
DEFINE WINDOW wrPfix FROM 05,46 TO 10,59 SHADOW COLOR SCHEME 4
IF !EMPTY(xcPfix)
  RETURN .T.
ENDIF
=SayMess(24,SPACE(9)+'[],[]-เลื่อน  [Enter]-เลือกคำนำหน้าชื่อที่ต้องการ','GR+/GR+')
ACTIVATE WINDOW wrPfix
@ 00,00 GET vnPfix FUNCTION ;
  '*RTV นาย    ;นางสาว  ;นาง    ;อื่น ๆ  ';
  DEFAULT 1
READ CYCLE TIMEOUT(cnDelay)
RELEASE WINDOW wrPfix
IF Esc_key()
  RETURN .F.
ENDIF
DO CASE
CASE vnPfix = 1
  vcPfix = 'นาย  '
```

```
CASE vnPfix = 2
  vcPfix = 'นางสาว'
CASE vnPfix = 3
  vcPfix = 'นาง   '
CASE vnPfix = 4
  vcPfix = SPACE(LEN(m.cPfix))
  RETURN .F.
ENDCASE
m.cPFIX = vcPfix
RETURN .T.

**************************************
*       Function : DEPART()          *
**************************************
FUNCTION Depart
PARAMETER xcDept
DEFINE WINDOW wrDept FROM 05,43 TO 16,59 SHADOW COLOR SCHEME 4
IF !EMPTY(xcDept)
  RETURN .T.
ENDIF
=SayMess(24,SPACE(9)+'[],[]-เลื่อน  [Enter]-เลือกภาควิชาที่ต้องการ','GR+/GR+')
ACTIVATE WINDOW wrDept
@ 00,00 GET vnDept FUNCTION ;
  '*RTV โทรคมนาคม  ;ไฟฟ้า     ;อิเล็กฯ    ;คอมพิวเตอร์;การวัดคุมฯ ;ระบบควบคุม ;เครื่องกล  ;เกษตร
;โยธา      ;อื่น ๆ    ';
  DEFAULT 1
READ CYCLE TIMEOUT(cnDelay)
RELEASE WINDOW wrDept
IF Esc_key()
  RETURN .F.
ENDIF
DO CASE
CASE vnDept = 1
  vcDept = 'วิศวกรรมโทรคมนาคม      '
CASE vnDept = 2
  vcDept = 'วิศวกรรมไฟฟ้า          '
```

```
CASE vnDept = 3
   vcDept = 'อิเล็กทรอนิกส์          '
CASE vnDept = 4
   vcDept = 'วิศวกรรมคอมพิวเตอร์     '
CASE vnDept = 5
   vcDept = 'การวัดคุมทางอุตสาหกรรม  '
CASE vnDept = 6
   vcDept = 'วิศวกรรมระบบควบคุม      '
CASE vnDept = 7
   vcDept = 'วิศวกรรมเครื่องกล        '
CASE vnDept = 8
   vcDept = 'วิศวกรรมเกษตร            '
CASE vnDept = 9
   vcDept = 'วิศวกรรมโยธา             '
CASE vnDept = 10
   vcDept = SPACE(LEN(m.cDepart))
ENDCASE
m.cDepart = vcDept
RETURN .T.

************************************
*       Function : FACULTY()       *
************************************
FUNCTION Faculty
PARAMETER xcFact
IF EMPTY(xcFact)
   m.cFacult = 'วิศวกรรมศาสตร์        '
ENDIF
RETURN .T.

*: EOF: STUDFILE.PRG
```

```
;=============================================;
;              CUR_OFF.ASM                    ;
;=============================================;
.286c
seg_a   segment
        assume  cs:seg_a, ds:seg_a
        org     100h
cur_off proc    far
start:  mov     ah,1
        mov     ch,0
        mov     cl,1            ;
        int     10h             ; Video display   ah=functn 01h
                                ;  set cursor mode in cl
        retf                    ; Return far
cur_off endp
seg_a   ends
        end     start

;=============================================;
;              CUR_ON.ASM                     ;
;=============================================;
.286c
seg_a   segment
        assume  cs:seg_a, ds:seg_a
        org     100h
cur_on  proc    far
start:  mov     ah,1
        mov     ch,6
        mov     cl,7
        int     10h     ; Video display   ah=functn 01h
                        ; set cursor mode in cx
        retf            ; Return far
cur_on  endp
seg_a   ends
        end     start
```

ภาคผนวก ง.

โปรแกรมสื่อสารข้อมูลผ่านพอร์ตอนุกรม RS-232C

```
;*************************************************
;* Program name : RECVDATA.ASM                   *
;*        Use by  : xBASE application (.bin)     *
;*    Sample call : RECVDATA with Parameter      *
;*      Parameter : 19 characters (string)       *
;*************************************************
CODE_SEG    SEGMENT BYTE PUBLIC "CODE_SEG"
COMPORT     EQU   0                    ;COM#1 = 0, COM#2 = 1
SET_COM     EQU   11100011B            ;Bit 0,1   = 11 to 8 bit
                                       ;Bit 2     = 0  to 1 stop bit
                                       ;Bit 3,4   = 00 to non parity
                                       ;Bit 5,6,7 = 111 to baud rate 9600
TX_CODE     EQU   01H                  ;Transmitt code
RX_CODE     EQU   02H                  ;Received code
ER_CODE     EQU   03H                  ;Error code
DATA_NO     EQU   19                   ;Number of data
LOOP_NO     EQU   5                    ;Number of loop
ERR_CHR     EQU   ' '                  ;Error character
BRK_CHR     EQU   '?'                  ;Break character
END_CHR     EQU   '*'                  ;End character
END_CODE    EQU   0FFH                 ;Received end character
;
            ASSUME  CS:CODE_SEG, DS:CODE_SEG

RECEIVE     PROC  FAR
;
START:      PUSH  AX
            PUSH  BX
            PUSH  CX
            PUSH  DX
            PUSH  DI
            MOV   DI,BX
;
;Initialize communication port
;
COM_INIT:   MOV   DX,COMPORT           ;Set serial port
            MOV   AH,0                 ;Set parameter function
            MOV   AL,SET_COM           ;Initial parameter
            INT   14H                  ;RS-232 DX=com#, AH=func 10h
                                       ;reset port, AL=init parameter
            MOV   CX,LOOP_NO           ;Set loop
SEND_CODE:  MOV   DX,COMPORT           ;Set serial port
            MOV   AL,RX_CODE           ;Send code to receive data
            MOV   AH,1                 ;Transmitt function
            INT   14H                  ;RS-232 DX=com#, AH=func 11h
                                       ;write char AL,  AH=retn status
            JMP   CHK_KEY1
TEST1:      TEST  AH,80H               ;Test error AH status (bit 7)
            JZ    RECV_DATA            ;Jump if status = 0 (no error)
            LOOP  SEND_CODE            ;Loop if CX > 0
            JMP   ERROR2               ;Jump if CX = 0
;
CHK_KEY1:   MOV   AH,1                 ;Check if keypress
            INT   16H
            JNZ   READ_KEY1            ;Yes, read keyboard
            JMP   TEST1                ;no , check acknownlage
;
READ_KEY1:  MOV   AH,0                 ;Read char form keyboard
            INT   16H
```

```
                CMP   AL,27              ;Check if key = Esc
                JE    BREAK              ;yes, send break char & exit
                JMP   TEST1              ;no , check acknownlage
;
;Receive characters (data+check)
;
RECV_DATA:      MOV   CX,DATA_NO         ;Set char number to receive
RX_LOOP1:       MOV   BX,LOOP_NO         ;Set loop
RX_LOOP2:       MOV   DX,COMPORT         ;Set port
                MOV   AH,2               ;Received function
                INT   14H
;
                JMP   CHK_KEY2
TEST2:          TEST  AH,AH              ;Test error AH status (bit 0-7)
                JZ    RECV_CH            ;Jump if status = 0 (no error)
                DEC   BX                 ;Decrease loop
                JNZ   RX_LOOP2           ;Jump if status <> 0 (error)
                JMP   ERROR1
;
RECV_CH:        CMP   AL,END_CODE        ;Test End code character
                JZ    END_RX             ;
                MOV   [DI],AL            ;Receive character
                INC   DI                 ;Next char
                LOOP  RX_LOOP1           ;Loop if CX > 0
;
CHK_KEY2:       MOV   AH,1               ;Check if keypress
                INT   16H
                JNZ   READ_KEY2          ;Yes, read keyboard
                JMP   TEST2              ;no , check acknownlage
;
READ_KEY2:      MOV   AH,0               ;Read char form keyboard
                INT   16H
                CMP   AL,27              ;Check if key = Esc
                JE    BREAK              ;yes, send break char & exit
                JMP   TEST2              ;no , check acknownlage
;
                POP   DI                 ;
                MOV   BX,DI
                PUSH  DI                 ;
                MOV   CX,DATA_NO-1       ;Set number of check_sum
                MOV   AL,0               ;
;
CHK_SUM:        XOR   AL,[BX]            ;Sum character to check_sum
                INC   BX                 ;
                LOOP  CHK_SUM            ;
                CMP   AX,[BX]            ;Test check_sum character
                JZ    EXIT               ;Exit  (received complete)

BREAK:          MOV   AL,ER_CODE         ;Send error code
                MOV   DX,COMPORT         ;Set port
                MOV   AH,1               ;Transmitt function
                INT   14H                ;
;
                POP   DI
                MOV   BX,DI
                PUSH  DI
                MOV   CX,DATA_NO         ;Set character number
                MOV   AL,BRK_CHR         ;Write end character
NEXT_B:         MOV   [BX],AL            ;Send character
```

```
                INC     BX                      ;Next character
                LOOP    NEXT_B                  ;Decrease CX ,loop if CX > 0
                JMP     EXIT
;
ERROR1:         MOV     CX,LOOP_NO              ;Set loop
ER_NXT:         MOV     AL,ER_CODE              ;Send error code
                MOV     DX,COMPORT              ;Set port
                MOV     AH,1                    ;Transmitt function
                INT     14H                     ;
;
                JMP     CHK_KEY3
;
TEST3:          TEST    AH,80H                  ;Test error AH status (bit 7)
                JZ      ERROR2                  ;Jump if status = 0 (no error)
                LOOP    ER_NXT                  ;Loop if CX > 0
;
CHK_KEY3:       MOV     AH,1                    ;Check if keypress
                INT     16H
                JNZ     READ_KEY3               ;Yes, read keyboard
                JMP     TEST3                   ;no , check acknownlage
;
READ_KEY3:      MOV     AH,0                    ;Read char form keyboard
                INT     16H
                CMP     AL,27                   ;Check if key = Esc
                JE      BREAK                   ;yes, send break char & exit
                JMP     TEST3                   ;no , check acknownlage
;
ERROR2:         POP     DI
                MOV     BX,DI
                PUSH    DI
                MOV     CX,DATA_NO              ;Set character number
                MOV     AL,ERR_CHR
NEXT_E:         MOV     [BX],AL                 ;Send character
                INC     BX                      ;Next character
                LOOP    NEXT_E                  ;Decrease CX ,loop if CX > 0
                JMP     EXIT
;
END_RX:         POP     DI
                MOV     BX,DI
                PUSH    DI
                MOV     CX,DATA_NO              ;Set character number
                MOV     AL,END_CHR              ;Write end character
NEXT_C:         MOV     [BX],AL                 ;Send character
                INC     BX                      ;Next character
                LOOP    NEXT_C                  ;Decrease CX ,loop if CX > 0
;
EXIT:           POP     DI
                POP     DX
                POP     CX
                POP     BX
                POP     AX
                RET
;
RECEIVE         ENDP
;
CODE_SEG        ENDS
;
                END     START
```

```
;***********************************************
;*  PROGRAM NAME : SETTIME.ASM                 *
;*            USE BY : XBASE APPLICATION (.BIN) *
;*    SAMPLE CALL : SETTIME WITH PARAMETER      *
;*      PARAMETER : 11 CHARACTERS (STRING)      *
;***********************************************
CSEG        SEGMENT BYTE   PUBLIC  "CSEG"
COMPORT     EQU    01H             ;COM#1 = 0, COM#2 = 1
SET_COM     EQU    11100011B       ;Bit 0,1   = 11 to 8 bit
                                   ;Bit 2     = 0  to 1 stop bit
                                   ;Bit 3,4   = 00 to parity non
                                   ;Bit 5,6,7 = 111 to baud rate 9600
TX_CODE     EQU    01H             ;Transmitt code
RX_CODE     EQU    02H             ;Received code
ER_CODE     EQU    03H             ;Error code
DATA_NO     EQU    11              ;Number of data
LOOP_NO     EQU    5               ;Loop number
ERR_CHR     EQU    ' '             ;Error character
END_CHR     EQU    '*'             ;End character
            ASSUME CS:CSEG,DS:CSEG
SET_TIME    PROC   FAR
START:      PUSH   AX
            PUSH   BX
            PUSH   CX
            PUSH   DX
            PUSH   DI
            MOV    DI,BX
            PUSH   DI
;
COM_INIT:   MOV    DX,COMPORT      ;Set serial port
            MOV    AH,0            ;Set comm-parameter function
            MOV    AL,SET_COM      ;Initialize paramiter
            INT    14H             ;RS-232 DX=com#, AH=func 20h
                                   ;reset port, AL=init parameter
            MOV    AL,0            ;Sum data to sending
            MOV    CX,DATA_NO-1    ;Set number to sum
```

```
SUM_CHK:    XOR    AL,[BX]          ;
            INC    BX               ;
            LOOP   SUM_CHK          ;Loop if CX > 0
            MOV    [BX],AL          ;Check_sum char (char #11)
            MOV    CX,LOOP_NO
SEND_CODE:  MOV    AL,TX_CODE       ;
            MOV    DX,COMPORT
            MOV    AH,1             ;Transmitt function
            INT    14H              ;RS-232 DX=com#, AH=func 21h
                                    ;write char AL,  AH=retn status
            TEST   AH,80H           ;Test error AH status (bit 7)
            JZ     SEND_CHAR        ;If no error, send char
            LOOP   SEND_CODE        ;Loop if CX > 0
            JMP    EXIT             ;Else if error, exit
;
SEND_CHAR:  MOV    CX,DATA_NO       ;Send 11 characters
TX_LOOP1:   MOV    BX,LOOP_NO
TX_LOOP2:   MOV    DX,COMPORT
            MOV    AL,[DI]
            MOV    AH,1
            INT    14H              ;RS-232 DX=com#, AH=func 21h
                                    ;write char AL,  AH=retn status
            TEST   AH,80H           ;Test AH status (bit 7)
            JZ     NEXT_CH          ;Jump if status = 0 (no error)
            DEC    BX               ;Decrease loop
            JNZ    TX_LOOP2         ;Jump if status <> 0 (error)
            JMP    ERROR1           ;Error if BX = 0
NEXT_CH:    INC    DI               ;Next char to send
            LOOP   TX_LOOP1         ;Loop if CX > 0
;
            MOV    CX,LOOP_NO
RX_CHK:     MOV    DX,COMPORT       ;Comm port #
            MOV    AH,2             ;Receive function
            INT    14H              ;RS-232 DX=com#, AH=func 11h
                                    ;write char AL,  AH=retn status
            TEST   AH,0FFH          ;Check error AH status (bit 0 - 7)
```

```
                JZ      CHK_END         ;Jump if status = 0 (no error)
                LOOP    RX_CHK          ;Loop if CX > 0
                JMP     ERROR1          ;Jump when CX = 0
;
CHK_END:        CMP     AL,END_CHR      ;Test if end char
                JZ      EXIT            ;yes, Exit
;
ERROR1:         MOV     CX,LOOP_NO
ER_NXT:         MOV     AL,ER_CODE
                MOV     DX,COMPORT      ;Comm port #
                MOV     AH,1            ;Transmitt function
                INT     14H             ;RS-232 DX=com#, AH=func 11h
                                        ;write char AL,  AH=retn status
                TEST    AH,80H          ;Check error AH status (bit 7)
                JZ      ERROR2          ;Exit if status = 0 (no error)
                LOOP    ER_NXT          ;Loop if CX > 0
ERROR2:         POP     DI
                MOV     BX,DI
                PUSH    DI
                MOV     CX,DATA_NO      ;data number
                MOV     AL,ERR_CHR      ;write error charactor
NEXT_E:         MOV     [BX],AL         ;send error charactor
                INC     BX              ;next character
                LOOP    NEXT_E          ;loop if CX > 0
;
EXIT:           POP     DI
                POP     DI
                POP     DX
                POP     CX
                POP     BX
                POP     AX
                RET
;
SET_TIME        ENDP
CSEG            ENDS
                END     START
```

ภาคผนวก จ.

โครงสร้างแฟ้มข้อมูลของโปรแกรมจัดการข้อมูล

Structure for database: C:\AAA\PRJ\STUDENT.DBF
Number of data records: 46
Date of last update : 05/07/97
Code Page : 0

Field	Field Name	Type	Width	Dec	Index	Collate
1	CID	Character	8		Asc	Machine
2	CPFIX	Character	8			
3	CNAME	Character	30		Asc	Machine
4	CSNAME	Character	30		Asc	Machine
5	CCLASS	Character	4		Asc	Machine
6	CDEPART	Character	2		Asc	Machine
7	CFACULT	Character	25			
8	NOTE	Character	10			
** Total **			141			

Structure for database: C:\AAA\PRJ\DATETIME.DBF
Number of data records: 7
Date of last update : 05/07/97
Code Page : 0

Field	Field Name	Type	Width	Dec	Index	Collate
1	CID	Character	8		Asc	Machine
2	DDATE	Date	8		Asc	Machine
3	CTIME	Character	5			
** Total **			22			

Structure for database: C:\AAA\PRJ\REPFILE.DBF
Number of data records: 4
Date of last update : 05/07/97
Code Page : 0

Field	Field Name	Type	Width	Dec	Index	Collate
1	CID	Character	8		Asc	Machine
2	DDATE	Date	8		Asc	Machine
3	CT_IN	Character	5			
4	CT_OUT	Character	5			
** Total **			27			

Structure for database: C:\AAA\PRJ\TEMP.DBF
Number of data records: 8
Date of last update : 03/31/97
Code Page : 0

Field	Field Name	Type	Width	Dec	Index	Collate
1	CTEMP	Character	19			
** Total **			20			

Structure for database: C:\AAA\PRJ\TEMP1.DBF
Number of data records: 0
Date of last update : 05/07/97
Code Page : 0

Field	Field Name	Type	Width	Dec	Index	Collate
1	CTEMP	Character	20			
** Total **			21			

Structure for database: C:\AAA\PRJ\TEMP2.DBF
Number of data records: 0
Date of last update : 05/07/97
Code Page : 0

Field	Field Name	Type	Width	Dec	Index	Collate
1	CTEMP	Character	20			
** Total **			21			

กิตติกรรมประกาศ

ขอขอบพระคุณอาจารย์ชินภัทร นันทจิวาภรชัย อาจารย์ที่ปรึกษาที่ได้กรุณาให้คำแนะนำปรึกษาเป็นอย่างดียิ่ง ขอขอบพระคุณอาจารย์ทุกท่านที่ได้ประสิทธิ์ประสาทวิชาความรู้ตลอดระยะเวลาที่ได้ศึกษาอยู่ ขอขอบคุณการสื่อสารแห่งประเทศไทยสำหรับเครื่องมือและอุปกรณ์ต่าง ๆ ในการทดลอง ขอขอบคุณบริษัทศิลารีเสิร์ชจำกัดสำหรับวงจรที่ใช้ในการทดลองและขอขอบคุณเพื่อน ๆ ทุกคนที่ให้กำลังใจ จนทำให้ปริญญานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยดี

ผู้จัดทำ

หนังสืออ้างอิง

1. จรณิต แก้วกังวาล, "การออกแบบและจัดการฐานข้อมูล", ซีเอ็ดยูเคชั่น, 214 หน้า, 2536.
2. จิรศักดิ์ เหลืองอุไร, "การสื่อสารอนุกรมบน PC ", ซีเอ็ดยูเคชั่น, 372 หน้า, 2538.
3. ปรเมษฐ์ ประนยานันท์ และ ปิยพงศ์ เผ่าวณิช , "คู่มือและการประยุกต์ใช้งานไมโครคอนโทรลเลอร์ ", ซีเอ็ดยูเคชั่น, 380 หน้า, 2536.
4. พิพัฒน์ เลาหสงคราม, "ไมโครคอนโทรลเลอร์ MCS-48 MCS-51", คณะวิศวกรรมศาสตร์ สถาบันเทคโน โลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 405 หน้า , 2537.
5. พิพัฒน์ เลาหสงคราม, "ภาษาแอสเซมบลีสำหรับไมโครคอนโทรลเลอร์ MCS-51", คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 312 หน้า , 2537.
6. ยืน ภู่วรวรรณ และไพศาล สงวนหมู่, "การสื่อสารข้อมูลและไมโครคอมพิวเตอร์เนตเวอร์ค", ซีเอ็ดยูเคชั่น, 244 หน้า, 2536.
7. รัชนัย อินทุใส และไตรภพ อินทุใส, " ไมโครคอนโทรลเลอร์ 8051", ฟิสิกส์เซนเตอร์, 174 หน้า, 2537.
8. ศิริชัย ชนบดีเฉลิมรุ่ง และ คณะ, "เทคนิคการใช้ FoxPro 2.5", สกายบุ๊กส์, 323 หน้า, 2537.
9. ศิริวรรณ ฉันทาดิศัย, "หลักการเขียนโปรแกรมภาษาแอสเซมบลี 8088", คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์, 384 หน้า, 2534.
10. สุนทร วิทูสุรพจน์, "การใช้งานไมโครคอนโทรลเลอร์ตระกูล 8051", ซีเอ็ดยูเคชั่น, 182 หน้า, 2537.
11. สุนทร วิทูสุรพจน์, "การโปรแกรมภาษาแอสเซมบลีของไมโครคอนโทรลเลอร์ตระกูล 8051", ซีเอ็ดยูเคชั่น, 182 หน้า, 2537.
12. สุวิพล สิทธิชีวภาค,"พื้นฐานแห่งการสื่อสารข้อมูล",สมาคมโทรคมนาคมแห่งประเทศไทย,286 หน้า, 2536.
13. ทวีศักดิ์ ไพศาลภาณุมาศ, "สาระสำคัญของบาร์โค้ด", วารสารเซมิคอนดัคเตอร์อิเล็คทรอนิคส์, ฉบับที่ 145, 2537-2538, หน้า 105-113.
14. วรวุฒิ วิบูลเจริญกิจจา, "รู้และเข้าใจบาร์โค้ด", วารสารเซมิคอนดัคเตอร์อิเล็คทรอนิคส์, ฉบับที่ 120, 2535, หน้า 58-65.
15. สมพัตร์ เบ็ญจชัยพร, "รหัสแถบและโปรแกรมสำหรับพิมพ์รหัสแถบด้วยไมโคร", วารสารไมโครคอมพิวเตอร์, ฉบับที่ 41, 2531, หน้า 167-176.
16. นฤมล เลาหชัยบุณย์ และ ลลิตา ยาแก้ว, "รหัสแถบ", ปริญญานิพนธ์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2534.
17. ไพบูลย์ ศิระพัฒน์, "ระบบเก็บข้อมูลรหัสแถบ", วิทยานิพนธ์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 2536.

18. วิเชียร เกียรติขจิตมั่น และ คณะ, "เครื่องอ่านรหัสแถบบันทึกเวลา", ปริญญานิพนธ์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง 2533.
19. Adams, Pat and Powell, Jordan, "FOXPRO 2.5 Advanced Developer's Handbook" , Brady Publishing, 872 p., 1993.
20. Granillo, Robert, "Illustrated FoxPro", Wordware Publishing,Inc., 1991.
21. Intel Corporation, "Microcontroller HandBook", Intel Corporation, 1992.
22. Hall, Douglas V., "Microprocessors and interfacing: Programming & Hardware", 2nd ed., McGraw-Hill , 624 p., 1992.
23. McClanahan, David and Owens, Ray, "The Complete FoxPro 2.5 Language Reference", SYBEX,Inc., 600 p., 1993.