ระบบจ่ายเชื้อเพลิง

FUEL AND FLUID CONTROL SYSTEM

นายแมน เคยสนิท
นายศุภกิจ สุนันตา
นายสุทธิชัย สันทาลุนัย

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมการวัดคุม
ภาควิชาวิศวกรรมการวัดคุม คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2547

FUEL AND FLUID CONTROL SYSTEM

MAN KHEOYSANITH

SUPPAKIT SUNANTA

SUTTICHAI SUNTHALUNAI

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF ENGINEERING IN INSTRUMENTATION ENGINEERING

DEPARTMENT OF INSTRUMENTATION ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2004

ภาควิชาวิศวกรรมการวัดคุม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

.....

หัวข้อปริญญานิพนธ์ ระบบจ่ายเชื้อเพลิง

FUEL AND FLUID CONTROL SYSTEM

นักศึกษาผู้จัดทำ	นายแมน	เดยสนิท	รหัสประจำตัว 45015517
	นายศุภกิจ	สุนันตา	รหัสประจำตัว 45015529
	นายสุทธิชัย	สันทาลุนัย	รหัสประจำตัว 45015537

ปริญญา วิศวกรรมศาสตรบัณฑิต

สาขาวิชา วิศวกรรมการวัดคุม

ปีการศึกษา 2547

อาจารย์ควบคุมปริญญานิพนธ์	ลายมือชื่อ
อ.ประภาส เริงรื่น	

วัน/เดือน/ปี ที่สอบ วันอังคารที่ 24 มีนาคม 2548

สถานที่สอบ ณ ห้องสอบปริญญานิพนธ์ ภาควิชาวิศวกรรมการวัดคุม

ภาควิชารับรองแล้ว

(รศ.ประสิทธิ์ จุฉเสรีวงศ์)

หัวหน้าภาควิชาวิศวกรรมการวัดคุม

หัวข้อปริญญานิพนธ์	ระบบจ่ายเชื้อเพลิง	
	Fuel and Fluid Control System	
นักศึกษาผู้จัดทำ	นายแมน	เคยสนิท
	นายศุภกิจ	สุนันตา
	นายสุทธิชัย	สันทาลุนัย
อาจารย์ที่ปรึกษา	อ. ประภาส	เริงรื่น
ปีการศึกษา	2547	

บทคัดย่อ

โครงงานนี้เป็นการควบคุมปริมาณของของไหลให้ได้ตามการควบคุม โดยใช้ไมโครคอน-โทรลเลอร์เป็นตัวควบคุม โดยใช้หลักการการนับพัลส์จากโฟลวมิเตอร์ (Flowmeter) นำการเปลี่ยนแปลงของจำนวนพัลส์มาทำการเปรียบเทียบทางกระบานการของไมโครคอนโทลเลอร์ ให้แสดงค่าปริมาณตามที่ต้องการโดยแสดงค่าทางด้าน 7-Segment และส่งค่าผ่านทางพอร์ตอนุกรม RS-232 ไปแสดงผลทางคอมพิวเตอร์ ออกทาง VISUAL BASIC และทำการทดลองเพื่อเปรียบเทียบปริมาณของของไหลที่นับได้ตามจำนวนพัลส์ให้ได้ตามค่าของของไหลจริง

Thesis Title	Fuel and Fluid Control System	
Authors	Mr.Man	Kheoysanith
	Mr.Suppakit	Sunanta
	Mr.Suttichai	Sunthalunai
Thesis Advisor	Mr.Prapart	RoengRuen
Year	2004	

ABSTRACT

This Project control flow rate of fluid by microcontroler.Principle is using pulse from flow meter compare in microcontroler. The outputs are 7-segment and port RS-232 to computer in Visual Basic. The value will calibrate with real flow rate.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ดีเพราะได้รับความเมตตาจาก อ.ประภาส เริงรื่น รองศาสตรจารย์ประภาษ อุดคกิมาพันธุ์ และ ที่ได้ให้คำแนะนำแก่ผู้วิจัย ตลอดมา อีกทั้งยังเอื้อเฟื้ออุปกรณ์และเครื่องมือต่างๆ ในการทำปริญญานิพนธ์นี้ ผู้วิจัยรู้สึกซาบซึ้ง และขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณ อาจารย์ภาควิศวกรรมการวัดคุมทุกท่าน ที่ให้คำแนะนำอันเป็นประโยชน์ต่อการทำปริญญานิพนธ์ฉบับนี้

และที่ลืมเสียมิได้คือ ขอกราบขอบพระคุณคุณพ่อคุณแม่ อันเป็นที่รัก ที่สนับสนุนและเป็นแรงบันดาลใจในการทำปริญญานิพนธ์ฉบับนี้

คุณค่าและประโยชน์อันพึงมีจากปริญญานิพนธ์ฉบับนี้ ผู้วิจัยขอมอบแด่ผู้มีพระคุณทุกท่าน

คณะผู้จัดทำ

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญภาพ

สารบัญภาพ (ต่อ)

ภาพที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความเป็นมาและเหตุจูงใจของการวิจัย

ในปัจจุบันการใช้พลังงานมีมาก และพลังงานส่วนใหญ่จะเป็นน้ำมันเชื้อเพลิง จากการที่เราทราบกันดีอยู่แล้วว่าการจะนำพลังงานมาใช้ เพื่อเป็นเชื้อเพลงให้รถ และเครื่องมือต่างๆ ที่ใช้น้ำมันเป็นเชื้อเพลิง เราจึงต้องใช้บริการในปั้มน้ำมัน

และปั้มน้ำมันแต่ละปั้มจะใช้เครื่องจ่ายน้ำมันในการให้บริการแก่ลูกค้า และเครื่องวัดปริมาณน้ำมันในปั้มน้ำมันส่วนใหญ่จะเป็นการนำเข้ามาจากต่างประเทศ จึงมีต้นทุนค่อนข้างสูง

กลุ่มโปรเจคนี้จึงต้องการที่จะศึกษาเกี่ยวกับระบบจ่ายเชื้อเพลิง ของการจ่ายน้ำมันต่อจำนวนเงินออกมา เพื่อจะเป็นแนวทางในการพัฒนาระบบจ่ายเชื้อเพลิงนี้หากมีการพัฒนาเพิ่มเติมอีก จนเราสามารถลดการสั่งซื้อเทคโนโลยีจากต่างประเทศให้ได้บ้างสักนิดก็ยังดี

1.2 วัตถุประสงค์

ในปริญญานิพนธ์นี้เป็นการศึกษาเกี่ยวกับการวัดการไหลของของเหลว (เปรียบเทียบ-เชื้อเพลิง) และการเขียนโปรแกรมเพื่อควบคุม การนับอัตราการไหล และเขียนโปรแกรมแสดงค่าของอัตราไหลออกมาเป็น จำนวนเงิน จำนวนลิตร จำนวนบาทต่อลิตร (ของของเหลว) โดยแสดงค่าออกมาทาง 7-segment และสามารถส่งค่าไปแสดงผลพร้อมทั้งควบคุมการทำงานด้วยคอมพิวเตอร์ได้

1.3 ขอบเขตของปริญญานิพนธ์

ปริญญานิพนธ์เล่มนี้จะกล่าวถึงการพัฒนาและการออกแบบ

- ควบคุมการไหลของเชื้อเพลิง
- แสดงค่าการไหลของเชื้อเพลิงเป็น 7-sigment
  - บาท
  - ลิตร
  - บาทต่อลิตร
- สามารถส่งค่าเพื่อแสดงผลทางคอมพิวเตอร์
- สามารถควบคุม Controller ผ่านทางคอมพิวเตอร์

1.4 ขั้นตอนการศึกษา

การทำโครงงานวิจัยในปริญญานิพนธ์ฉบับนี้มีขั้นตอนการศึกษาเริ่มจาก การศึกษาการทำงานของไมโครคอนโทรลเลอร์ MCS-51 (Microcontroller-MCS51) โดยศึกษาโครงสร้างของไมโครคอนโทรลเลอร์เป็นหลักว่ามีกระบวนการใดบ้างที่จะสามารถนับจำนวนพัลล์ได้ แล้วก็ศึกษาแล้วนำเอาความสามารถของไมโครคอนโทรลเลอร์ที่มีมาประกอบกับอุปกรณ์เอาต์พุต เพื่อแสดงผล ศึกษาวิชวลเบสิก (Visual Basic) เพื่อทำหน้าจอแสดงที่ส่งมาจากไมโคคอนโทรลเลอร์

บทที่ 2
สถาปัตยกรรมของไมโครคอนโทรลเลอร์ตระกูล MCS-51

2.1 บทนำ

ไมโครคอนโทรลเลอร์ (Microcontroller) เป็นชื่อของอุปกรณ์อิเล็กทรอนิกส์ที่รวบรวมความสามารถมากมาย ไม่ว่าจะเป็นหน่วยประมวลผล หน่วยคำนวณทางคณิตศาสตร์และลอจิก วงจรรับสัญญาณอินพุต วงจรขับสัญญาณออกทางเอาต์พุต หน่วยความจำ วงจรกำเนิดสัญญาณนาฬิกา ทำให้ไมโครคอนโทรลเลอร์สามารถนำไปประยุกต์ใช้งานแทนวงจรอิเล็กทรอนิกส์ที่ซับซ้อนได้เป็นอย่างดีโดยช่วยลดจำนวนของอุปกรณ์และขนาดของระบบลง ในขณะที่มีขีดความสามารถสูงขึ้น ภายใต้งบประมาณที่เหมาะสม

2.2 คุณสมบัติทางเทคนิคที่สำคัญ

ไมโครคอนโทรลเลอร์ที่ใช้อ้างอิงเพื่อการเรียนรู้ และใช้งานในที่นี้เป็นไมโครคอนโทรลเลอร์ตระกูล MCS-51 ซึ่งมีหน่วยความจำภายในแบบ แฟลช (flash memory) ของ Philips Semiconductor ในอนุกรม P89C51Rx2 โดยจะเน้นไปที่เบอร์ P89C51RD2

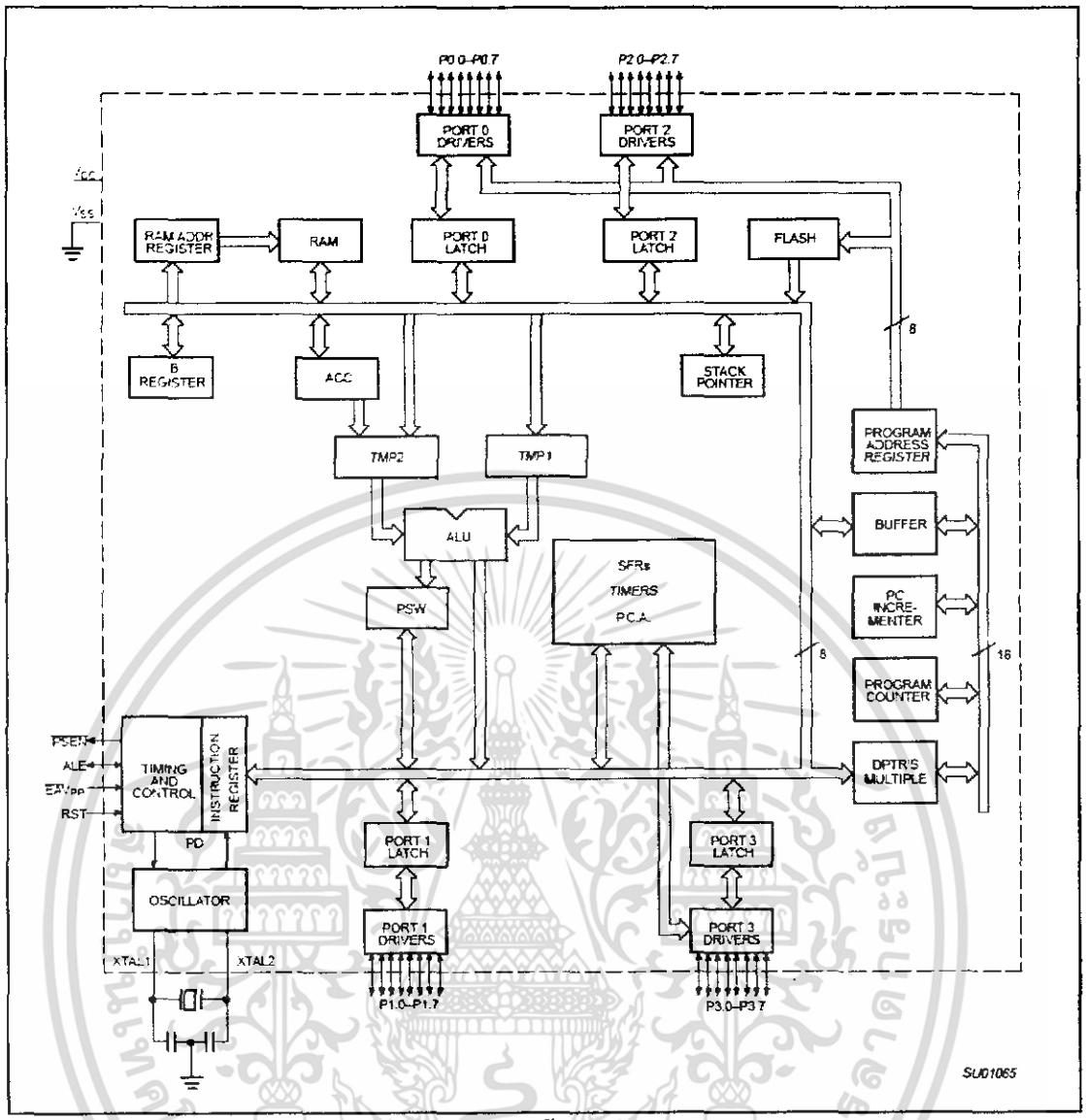
สำหรับคุณสมบัติทางเทคนิคที่โดดเด่นของไมโครคอนโทรลเลอร์ MCS-51 อนุกรมนี้ มีดังต่อไปนี้

- เป็นไมโครคอนโทรลเลอร์ 8 บิต ที่เข้ากันได้กับไมโครคอนโทรลเลอร์ MCS-51 พื้นฐานของอินเทล
- หน่วยความจำโปรแกรมภายในตัวไมโครคอนโทรลเลอร์เป็นแบบแฟลช ทำให้สามารถลบและเขียนใหม่ได้ถึงหนึ่งหมื่นครั้ง จึงสามารถใช้งานในรูปแบบไมโครคอนโทรลเลอร์ชิพเดี่ยวไม่ต้องใช้หน่วยความจำภายนอก ส่งผลให้สามารถใช้งานพอร์ตอินพุตของไมโครคอนโทรลเลอร์ได้อย่างเต็มประสิทธิภาพ ขนาดของหน่วยความจำโปรแกรมสูงถึง 64 กิโลไบต์
- หน่วยความจำข้อมูลแรมภายในมีขนาด 1 กิโลไบต์
- สามารถเขียน หรือโปรแกรมข้อมูลลงในหน่วยความจำโปรแกรมโดยไม่ต้องถอดไมโครคอนโทรลเลอร์ออกมาทำการโปรแกรมใหม่ หรือที่เรียกว่า การ **โปรแกรมในวงจร** หรือ **ในระบบ** (ISP : In-system programming) โดยภายในไมโครคอนโทรลเลอร์จะมีหน่วยความจำที่บรรจุโปรแกรมสำหรับเขียนข้อมูลลงใน

หน่วยความจำโปรแกรมหรือที่เรียกว่า **บูตรอม** (boot ROM) ทำหน้าที่ส่วนนี้แรงดันที่ใช้ในโปรแกรมแบบ ISP ขึ้นอยู่กับเบอร์ของไมโครคอนโทรลเลอร์

P89C51RD2 ใช้แรงดันในการโปรแกรมได้ทั้ง +5V และ +12V โดยใช้แรงดัน +12V จะสามารถโปรแกรมได้ 1,000 ครั้ง และถ้าใช้แรงดัน +5V สามารถโปรแกรมได้ 10,000 ครั้ง

- ความถี่สัญญาณนาฬิกาสูงสุด 33MHz ในกรณีทำงานด้วยสัญญาณนาฬิกา 12 ลูกต่อแมชีนไซเกิล และ 20MHz ในกรณีทำงานด้วยสัญญาณนาฬิกาภายใน 6 ลูกต่อแมชีนไซเกิล
- P89C51RD2 ได้รับการกำหนดให้ทำงานในโหมดเบื้องต้นในโหมดสัญญาณนาฬิกา 6 ลูกต่อแมชีนไซเกิลสามารถเลือกเปลี่ยนเป็น 12 สัญญาณนาฬิกาต่อแมชีนไซเกิลได้
- ชุดคำสั่งและสถาปัตยกรรมพื้นฐานเหมือนกับ ไมโครคอนโทรลเลอร์ MCS-51 ของผู้ผลิตอื่น ไม่ว่าจะเป็น Intel, Siemens หรือ Dallas Semiconductor
- ขาพอร์ต 8 บิต จำนวน 4 พอร์ต เป็นแบบกึ่งสองทิศทาง(quasi-bidirectional) สามารถใช้งานเป็นได้ทั้งอินพุตและเอาต์พุตภาพที่ 2-1 โครงสร้างการทำงานพื้นฐานของไมโครคอนโทรลเลอร์ P89C51RD2



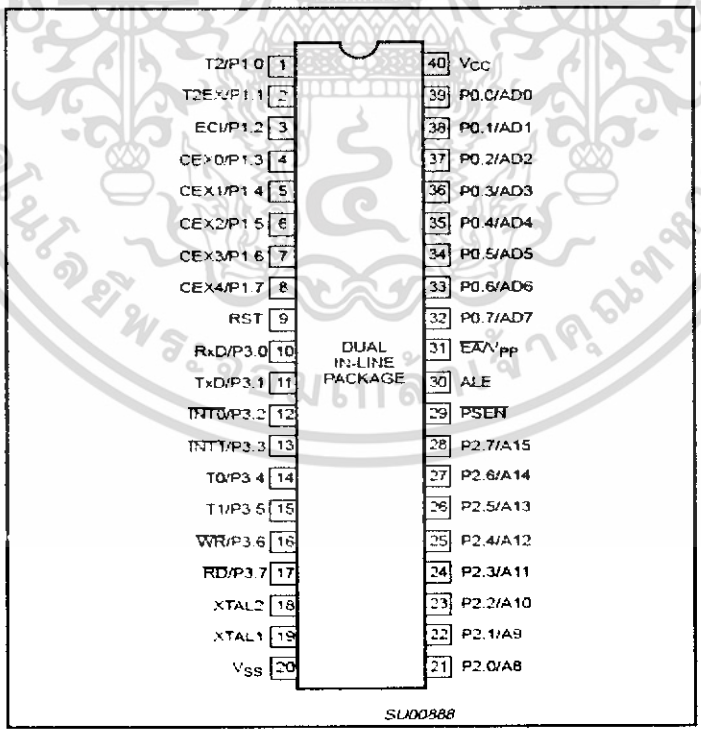
ภาพที่ 2-1 โครงสร้างการทำงานพื้นฐานของไมโครคอนโทรลเลอร์

- มีวงจรสื่อสารอนุกรมแบบฟูลดูเพล็กซ์
- ไทเมอร์/เคาน์เตอร์ขนาด 16 บิต 3 ตัว (ไทเมอร์ 0, 1 และ 2)
- มีรีจิสเตอร์ตัวชี้ตำแหน่งข้อมูลหรือ DPTR 2 ตัว
- สามารถรองรับแหล่งกำเนิดอินเตอร์รัปต์ได้ 7 ประเภท
- กำเนิดนัยสำคัญของการตอบสนองอินเตอร์รัปต์ได้ 4 ระดับ
- สามารถติดต่อหน่วยความจำภายนอกได้สูงสุด 64 กิโลไบต์
- มีวงจรวอตช์ด็อกไทเมอร์
- มีโมดูลวงจรนับโปรแกรมได้ (PCA : Programmable Counter Array) ซึ่งบรรจุวงจรตรวจจับสัญญาณ(capture), เปรียบเทียบสัญญาณ (compare), วงจรมอดูเลชั่นทางความกว้างพัลส์ (PWM) 5 โมดูลและวอตช์ด็อกไทเมอร์(watchdog timer)

ในภาพที่2-1 เป็นโครงสร้างพื้นฐานของ P89C51D+ และ P89C51RD2 จะเห็นได้ว่าคล้ายคลึงกับไมโครคอนโทรลเลอร์ตระกูล MCS-51 พื้นฐาน แต่มีส่วนประกอบที่เพิ่มขึ้น อาทิ หน่วยความจำเก็บโปรแกรมบูตรอมสำหรับทำการอ่าน-เขียนข้อมูลในหน่วยความจำโปรแกรมแบบ ISP โมดูลวงจรนับแบบโปรแกรมได้ หรือ PCA

ตารางที่ 2-1 รายละเอียดเบื้องต้นของไมโครคอนโทรลเลอร์ในอนุกรม P89C51Rx+ และ Rx2

เบอร์ของไมโครคอนโทรลเลอร์	หน่วยความจำโปรแกรมแบบแฟลช	หน่วยความจำข้อมูลแรม	ความถี่สัญญาณนาฬิกาหลักที่ใช้งาน	
			โหมด 6 หลักสัญญาณนาฬิกา	โหมด 12 หลักสัญญาณนาฬิกา
P89C51RC+	32 กิโลไบต์	512 ไบต์	-	0-33MHz
P89C51RD+	64 กิโลไบต์	1 กิโลไบต์	-	0-33MHz
P89C51RB2	16 กิโลไบต์	512 ไบต์	0-20MHz	0-33MHz
P89C51RC2	32 กิโลไบต์	512 ไบต์	0-20MHz	0-33MHz
P89C51RD2	64 กิโลไบต์	1 กิโลไบต์	0-20MHz	0-33MHz



ภาพที่ 2-2 แสดงการจัดขาของ P89C51RD2

(Programmable Counter Array) ซึ่งภายใน PCA นี้บรรจุ วอชด์ดือก ไทเมอร์, วงจรตรวจจับสัญญาณหรือแคปเจอร์, วงจรเปรียบเทียบสัญญาณ และวงจร PWM รวมถึงไทเมอร์ 2 (Timer2)

อย่างไรก็ตาม P89C51RD2 ต่างก็มีเบอร์อื่นๆที่อยู่ร่วมอนุกรมกัน ดังแสดงรายละเอียดเบื้องต้นของไมโครคอนโทรลเลอร์ในอนุกรมนี้ของ philips ในตารางที่ 2-1 ไมโครคอนโทรลเลอร์ MCS-51 ทุกเบอร์จะมีสถาปัตยกรรมและขาใช้งานพื้นฐานเหมือนกัน แต่ P89C51RD2 ที่ขาพอร์ต 1 คือ P1.0-P1.7 จะมีความสามารถพิเศษเพิ่มเติม ดังในภาพที่ 2-2 และมีรายละเอียดขั้นต้น สรุปในตารางที่ 2-1

2.3 ความเร็วในการทำงานของไมโครคอนโทรลเลอร์ P89C51RD2

นอกจาก Philips จะทำการพัฒนาไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชในอนุกรมนี้ขึ้นเพื่อรองรับการโปรแกรมแบบ ISP และขนาดของหน่วยความจำแบบแฟลชที่สูงถึง 64 กิโลไบต์ แล้วยังได้พัฒนาเรื่องของความเร็วในการทำงานด้วย โดย P89C51RD2 ถูกกำหนดให้ทำงานได้เร็ว 6 ไซเกิลสัญญาณนาฬิกาภายในต่อ 1 แมชีนไซเกิลซึ่งเรียกว่าเร็วกว่าไมโครคอนโทรลเลอร์ MCS-51 มาตรฐาน 2 เท่า แต่ก็สามารถลดความเร็วให้เท่ากับแบบมาตรฐานได้ด้วยการโปรแกรมแบบขนาน (Parallel programming) ซึ่งต้องใช้เครื่องโปรแกรมภายนอก อาทิ ALL-11 ของ Hi-Lo Systems เป็นต้น แต่เมื่อลดความเร็วลงแล้ว จะไม่สามารถเปลี่ยนกลับมาได้อีก นั่นคือ สามารถเปลี่ยนความเร็วได้เพียงครั้งเดียว

ในการทำงาน 1รอบหรือ 1 แมชีนไซเกิล ซีพียูในไมโครคอนโทรลเลอร์ MCS-51 มาตรฐานใช้เวลา 12 คาบเวลาของสัญญาณนาฬิกา นั่นคือที่สัญญาณนาฬิกา 12 MHz เวลาในการทำงาน 1 ไซเกิลมีค่าเท่ากับ 1us หรือมีความเร็วในการทำงานภายใน 1 MHz ดังนั้นถ้าต้องการทราบความเร็วของการทำงานภายในสามารถหาได้จากค่าความถี่สัญญาณนาฬิกาหารด้วย 12 และถ้าต้องการหาค่าเวลาของ 1 รอบการทำงานหรือ 1 แมชีนไซเกิล ทำได้โดการหาส่วนกลับของความเร็วในการทำงานภายในของไมโครคอนโทรลเลอร์ แต่เนื่องจาก P89C51RD2 มีความเร็วกว่าปกติเป็น 2 เท่า จึงสามารถสรุปเป็นสูตรหาความเร็วในการทำงานใหม่ได้ดังนี้

ความเร็วในการทำงานภายใน = ความถี่สัญญาณนาฬิกา (ค่าคริสตอลที่ขา XTAL1 และ XTAL2) /6

เวลาใน 1 แมชีนไซเกิล = 1/ความเร็วในการทำงานของไมโครคอนโทรลเลอร์

2.4 รายละเอียดเบื้องต้นของขาใช้งานของไมโครคอนโทรลเลอร์ P89C51RD2

ชื่อขา Vcc เป็นขาที่ 40 ชนิดอินพุต หน้าที่และการทำงานต่อไฟเลี้ยง +5V

ชื่อขา GND เป็นขาที่ 20 ชนิดอินพุต หน้าที่และการทำงานต่อกราวด์

ชื่อขา P0.0-P0.7 เป็นขาที่ 39-32 ชนิดอินพุต/เอาต์พุต หน้าที่และการทำงาน

-ใช้งานเป็นขาพอร์ตอินพุตเอาต์พุต ถ้าต้องการกำหนดให้ขาพอร์ต 0 ขาใดขาหนึ่งเป็นอินพุต สามารถทำได้โดยการเขียนข้อมูล "1" ไปยังแต่ละบิตของขาพอร์ตที่ต้องการติดต่อด้วย ทำให้มีสถานะลอย (float) อินพุตอิมพีแดนซ์สูงสามารถใช้งานเป็นขาพอร์ตอินพุตได้

-ใช้ในการติดต่อกับขาแอดเดรสไบต์ต่ำของหน่วยความจำภายนอก (A0-A7) และขาข้อมูล (D0-D7) โดยใช้การมัลติเพล็กซ์เข้าช่วยเพื่อสลับการทำงานให้เป็นทั้งขาติดต่อแอดเดรสและขาข้อมูลในการติดต่อกับหน่วยความจำภายนอก

ชื่อขา P1.0-P1.7 เป็นขาที่ 1-8 ชนิดอินพุต/เอาต์พุต หน้าที่และการทำงาน

-ใช้งานเป็นขาพอร์ตอินพุตเอาต์พุตสำหรับใช้งานทั่วไป

- เป็นขาสัญญาณของไทเมอร์ 2 และสัญญาณของโมดูล PCA รายละเอียดต่อไปนี้

T2 (P1.0 : ขา 1) เป็นขาอินพุตสำหรับนับค่าของไทเมอร์ 2 และขาเอาต์พุตสัญญาณนาฬิกาโปรแกรมได้

T2EX (P1.1 : ขา 2) เป็นขาสำหรับควบคุมการทำงานของไทเมอร์/เคาน์เตอร์ 2

ECI (P1.2 : ขา 3) เป็นขาอินพุตสัญญาณนาฬิกาจากภายนอกสำหรับโมดูล PCA

CEX0 (P1.3 : ขา 4) เป็นขาอินพุตเอาต์พุตภายนอกวงจรตรวจจับและเปรียบเทียบสัญญาณสำหรับ PCA โมดูล 0

CEX1 (P1.4 : ขา 5) เป็นขาอินพุตเอาต์พุตภายนอกวงจรตรวจจับและเปรียบเทียบสัญญาณสำหรับ PCA โมดูล 1

CEX2 (P1.5 : ขา 6) เป็นขาอินพุตเอาต์พุตภายนอกวงจรตรวจจับและเปรียบเทียบสัญญาณสำหรับ PCA โมดูล 2

CEX3 (P1.6 : ขา 7) เป็นขาอินพุตเอาต์พุตภายนอกวงจรตรวจจับและเปรียบเทียบสัญญาณสำหรับ PCA โมดูล 3

CEX4 (P1.7 : ขา 8) เป็นขาอินพุตเอาต์พุตภายนอกวงจรตรวจจับและเปรียบเทียบสัญญาณสำหรับ PCA โมดูล 4

ชื่อขา P2.0-P2.7 เป็นขาที่21-28 ชนิดอินพุต/เอาต์พุต หน้าที่และการทำงาน

- ใช้งานเป็นขาพอร์ตอินพุต/เอาต์พุตสำหรับใช้งานทั่วไป
- ใช้ติดต่อกับขาแอดเดรสไบต์สูงของหน่วยความจำภายนอก (A8-A15) เมื่อติดต่อด้วย

ชื่อขา P3.0-P3.7 เป็นขาที่10-17 ชนิดอินพุต/เอาต์พุต หน้าที่และการทำงาน

- ใช้งานเป็นขาพอร์ตอินพุต/เอาต์พุตสำหรับใช้งานทั่วไป
- ใช้เป็นขาพอร์ตหน้าที่พิเศษ ดังมีรายละเอียดต่อไปนี้

RxD (P3.0 : ขา 10) ใช้เป็นขาอินพุตสำหรับรับข้อมูลจากการสื่อสารแบบอนุกรม

TxD (P3.1 : ขา 11) ใช้เป็นขาอินพุตสำหรับรับส่งมูลจากการสื่อสารแบบอนุกรม

INT0 (P3.2: ขา 12) ใช้เป็นขาอินพุตสำหรับรับสัญญาณอินเตอร์รัปต์จากภายนอกช่อง 0

INT1 (P3.3: ขา 13) ใช้เป็นขาอินพุตสำหรับรับสัญญาณอินเตอร์รัปต์จากภายนอกช่อง 1

T0 (P3.4: ขา 14) ใช้เป็นขาอินพุตสำหรับรับสัญญาณไทเมอร์จากภายนอกช่อง 0

T1 (P3.5: ขา 15) ใช้เป็นขาอินพุตสำหรับรับสัญญาณอินเตอร์รัปต์จากภายนอกช่อง 1

WR (P3.6 : ขา 16) ใช้เป็นขาสัญญาณ WR ในกรณีที่ใช้เชื่อมต่อกับหน่วยความจำภายนอก

RD (P3.6 : ขา 16) ใช้เป็นขาสัญญาณ RD ในกรณีที่ใช้เชื่อมต่อกับหน่วยความจำภายนอก

ชื่อขา RESET เป็นขาที่ 9 ชนิดอินพุต หน้าที่และการทำงาน ใช้ในการรีเซตการทำงานของไมโครคอนโทรลเลอร์ โดยในการป้อนสัญญาณลอจิก " 1 " อย่างน้อยเป็นเวลา 2 แมชีนไซเกิล โดยที่วงจรกำเนิดสัญญาณนาฬิกายังคงทำงานต่อเนื่องไปอย่างปกติ

ชื่อขา ALE เป็นขาที่ 30 ชนิดเอาต์พุต หน้าที่และการทำงาน Address Latch Enable ออกมาทุก ๆ แมชีนไซเกิล อย่างไนก็ตาม สามารถดิสเอเบิลสัญญาณพัลส์นี้ได้ โดยการเซตบิต 0 ของรีจิสเตอร์ AUXR

ชื่อขาPSEN เป็นขาที่ 29 ชนิดเอาต์พุต หน้าที่และการทำงาน Program Store Enable : ใช้ในการส่งสัญญาณเพื่อร้องขอติดต่อกับหน่วยความจำโปรแกรมภายนอก เมื่อต้องการอ่านข้อมูลจากหน่วยความจำโปรแกรมภายนอก ไมโครคอนโทรลเลอร์จะส่งสัญญาณออกมาที่ขานี้ 2 ครั้ง นอกจากนี้ยังใช้ประกอบในการอ่านและเขียนข้อมูลลงในหน่วยความจำโปรแกรมด้วยกระบวนการ ISP

- สำหรับเบอร์ P89C51RD+ ให้ต่อขานี้ลงกราวด์ แล้วป้อนไฟ +12V (+0.5Vและ-0.5V) เข้าที่ขา EA/Vpp

- สำหรับเบอร์ P89C51RD+ ให้ต่อขานี้ลงกราวด์, ป้อนลอจิก " 1 " เข้าที่ขา P2.7 และป้อนแรงดัน +5V เข้าที่ขา EA/Vpp

ชื่อขาEA/Vpp เป็นขาที่ 31 **ชนิดอินพุต** หน้าที่และการทำงาน External Access-enable/Programming voltage input : ใช้สำหรับเลือกการติดต่อหน่วยความจำโปรแกรมจากภายนอกหรือภายในตัวไมโครคอนโทรลเลอร์

" 0 " เลือกให้ไมโครคอนโทรลเลอร์ติดต่อกับหน่วยความจำโปรแกรมภายนอก

" 1 " เลือกให้ไมโครคอนโทรลเลอร์ติดต่อกับหน่วยความจำภายใน

นอกจากนี้ ที่ขานี้ยังใช้ขาอินพุตสำหรับขาอินพุตสำหรับการโปรแกรมหน่วยความจำภายในไมโครคอนโทรลเลอร์

- สำหรับเบอร์ P89C51RD+ ต้องการแรงดัน +12V (+5V และ -5V)

- สำหรับเบอร์ P89C51RD2 ต้องการแรงดัน +5V

ชื่อขาXTAL1 เป็นขาที่ 19 **ชนิดอินพุต** หน้าที่และการทำงานขาอินพุตรับสัญญาณจากวงจรขยายออสซิลเลเตอร์ (ขา XTAL2) และจากภายนอกในการใช้งานปกติ ขานี้และขา XTAL2 คริสตอลและตัวเก็บประจุชดเชยค่าน้อยๆ

ชื่อขาXTAL2 เป็นขาที่ 18 **ชนิดเอาต์พุต** หน้าที่และการทำงานขาอินพุตรับสัญญาณจากวงจรขยายออสซิลเลเตอร์ (ขา XTAL2) และจากภายนอกในการใช้งานปกติ ขานี้และขา XTAL1 คริสตอลและตัวเก็บประจุชดเชยค่าน้อยๆ

2.5 การจัดหน่วยความจำของไมโครคอนโทรลเลอร์ P89C51RD2

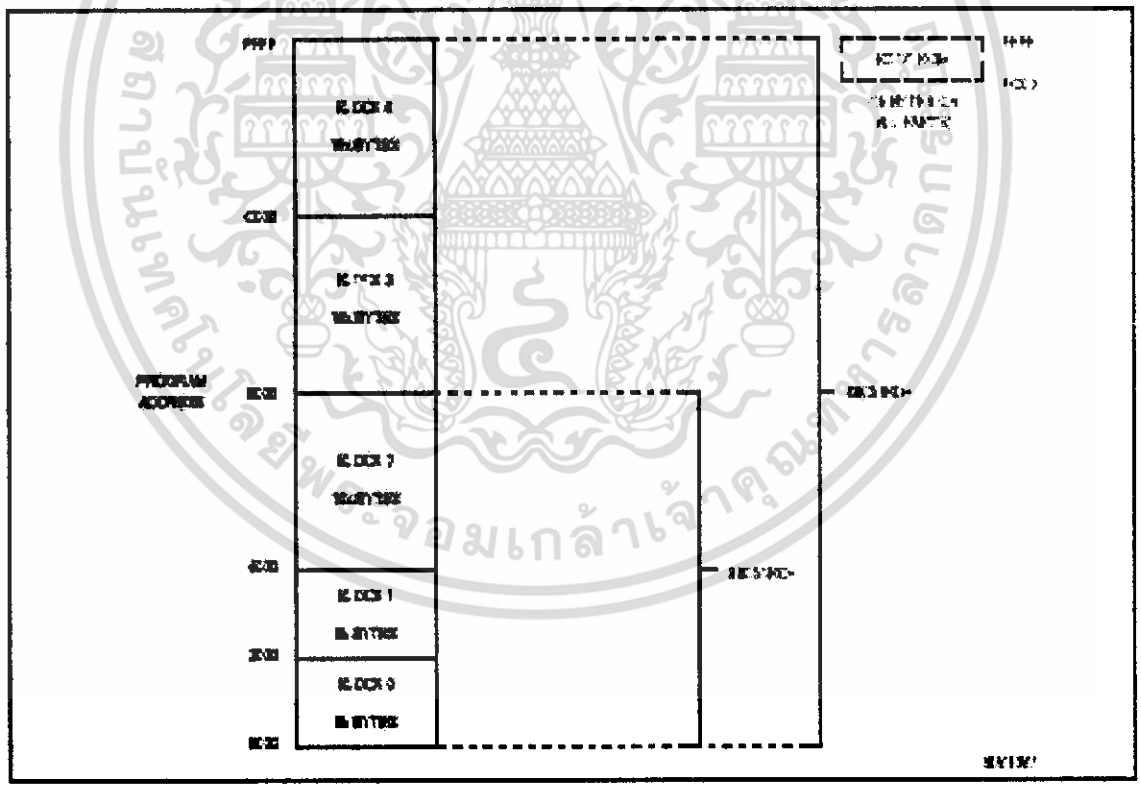
ในไมโครคอนโทรลเลอร์ MCS-51 แบบแฟลชของ Philips เบอร์ P89C51RD2 มีหน่วยความจำในหลักๆอยู่ 3 ส่วนคือ หน่วยความจำโปรแกรม (program memory) ขนาด 64 กิโลไบต์, หน่วยความจำข้อมูล (data memory) 1 กิโลไบต์ และหน่วยความจำบูตรอม (boot ROM) ขนาด 1 กิโลไบต์มีแอดเดรสอยู่ที่ FC00H-FFFFH ซึ่งเก็บโปรแกรมการอ่าน-เขียนข้อมูลในหน่วยความจำโปรแกรมของไมโครคอนโทรลเลอร์ในแบบ ISP อย่างไรก็ตามหากมีการใช้งานบูตรอม หน่วยความจำโปรแกรมจะสามารถเข้าถึงและทำงานได้ 63 กิโลไบต์ ก็สามารถทำได้ โดยการปิดและไม่ติดต่อกับบูตรอมนี้ อันจะส่งผลให้ไม่สามารถโปรแกรมหน่วยความจำแบบ ISP ในทางซอฟต์แวร์ได้ (ดูรายละเอียดได้ในคาต้าชีตของ P89C51RD2 ในหัวข้อ IAP: In-System

Application Programming) แต่ยังคงสามารถโปรแกรมด้วยกระบวนการ ISP ทางฮาร์ดแวร์ได้ ซึ่งจะกล่าวถึงต่อไป ในภาพที่ 2-3 แสดงการจัดสรรหน่วยความจำโปรแกรมของไมโครคอนโทลเลอร์ P89C51RD2

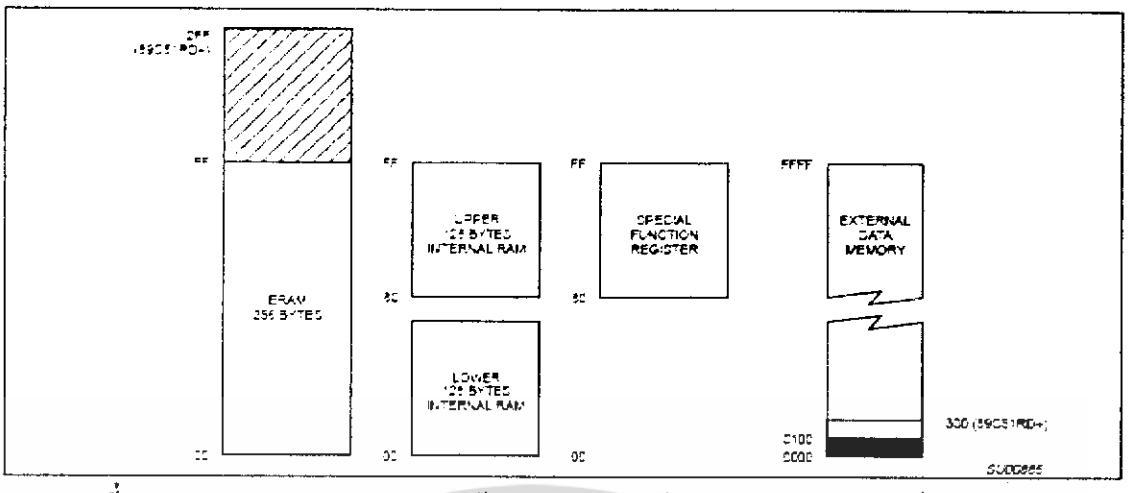
2.5.1 สำหรับหน่วยความจำข้อมูล 1 กิโลไบต์นั้นแบ่งออกเป็น 3 ส่วนคือ

- หน่วยความจำข้อมูลแรมส่วนล่าง 128 ไบต์ มีแอดเดรสอยู่ที่ 00H-7FH สามารถเข้าถึงได้ทั้งแบบโดยตรงและแบบโดยอ้อม หน่วยความจำส่วนนี้เป็นพื้นที่ของรีจิสเตอร์ใช้งานทั่วไปของไมโครคอนโทรลเลอร์ MCS-51

- หน่วยความจำข้อมูลแรมส่วนบน 128 ไบต์ มีแอดเดรสอยู่ที่ 80H-FFH เป็นพื้นที่ของรีจิสเตอร์ฟังก์ชั่นพิเศษหรือ SFR (Special Function Register) สามารถเข้าถึงได้ ในขณะเดียวกันยังเป็นพื้นที่เก็บข้อมูลโดยสามรถเข้าถึงได้ทางอ้อมเท่านั้น ซึ่งหน่วยความจำในส่วนนี้มีลักษณะเหมือนกับไมโครคอนโทรลเลอร์ MSC-51 พื้นฐานทุกประการ



ภาพที่ 2-3 การจัดสรรหน่วยความจำโปรแกรมของไมโครคอนโทรลเลอร์ P89C51RD2



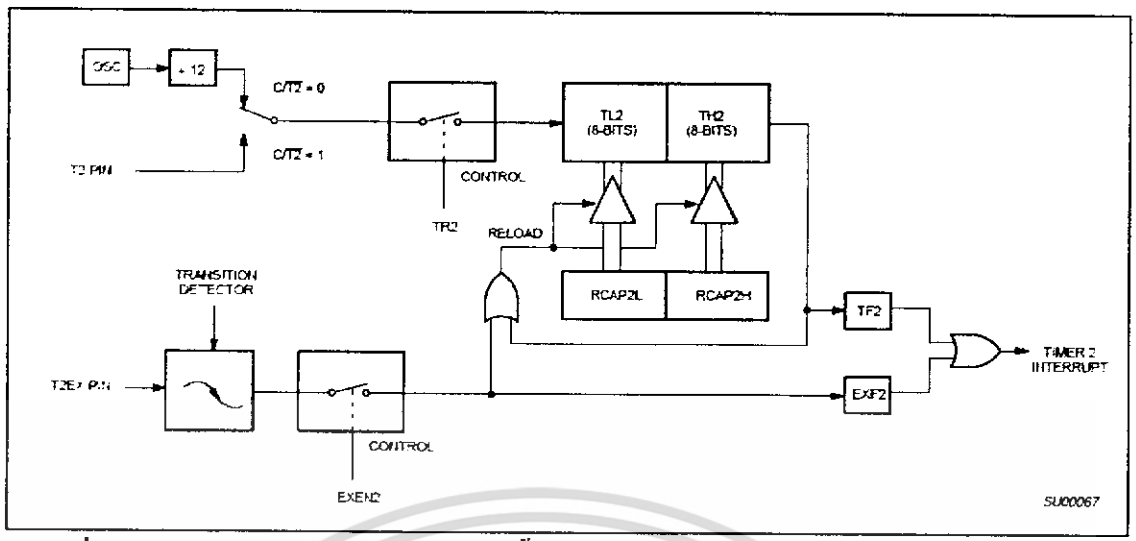
ภาพที่ 2-4 การจัดสรรหน่วยความจำข้อมูลแรมภายในไมโครคอนโทรลเลอร์ P89C51RD2

- หน่วยความจำเพิ่มเติม 768 ไบต์ หรือ EPOM (Expanded RAM) สามารถเข้าถึงได้แบบโดยอ้อมด้วยการใช้คำสั่ง MOVE เหมือนกับการติดต่อกับหน่วยความจำข้อมูลภายนอก โดยหน่วยความจำในส่วนนี้จะสามารถติดต่อได้เมื่อมีการเคลียร์บิต EXTRAM ในรีจิสเตอร์ AUXR แต่ถ้าหากบิต EXTRAM นี้เซตเป็น "1" จะเป็นการกำหนดให้ไมโครคอนโทรลเลอร์ติดต่อกับหน่วยความจำข้อมูลภายนอกแทน ในภาพที่ 2-4 แสดงการจัดสรรหน่วยความจำข้อมูลแรมภายในไมโครคอนโทรลเลอร์

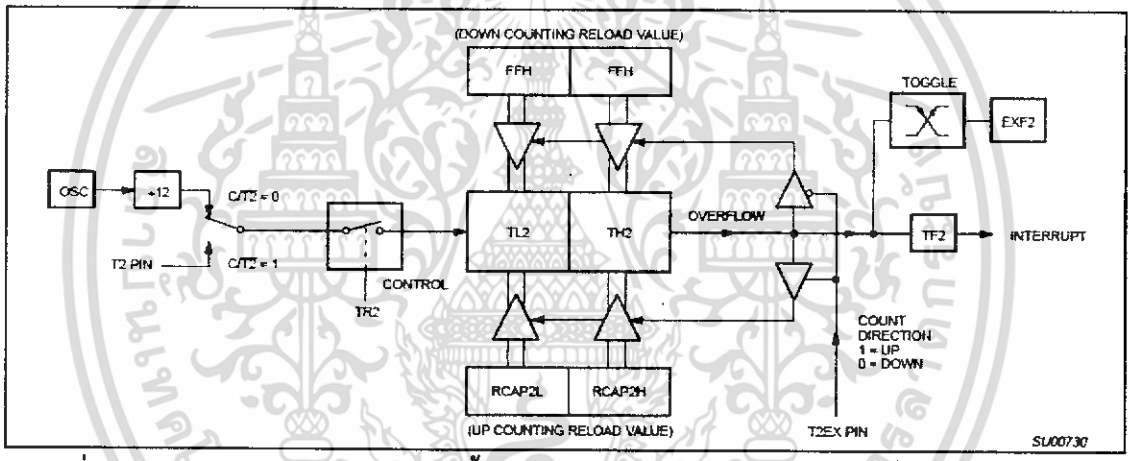
2.6 การทำงานในโหมดกำเนิดอัตราเร็วในการสื่อสารอนุกรมหรืออัตราบอด

การทำงานในโหมดนี้จะเกิดขึ้นเมื่อทำการรีเซตบิต RCLK และ /หรือ TCLK ในรีจิสเตอร์ T2CO ให้เป็น "1" เมื่อเข้าสู่การทำงานโหมดนี้ บิต TF2 จะไม่เกิดการเซต และ ไม่มีการสร้างสัญญาณอินเตอร์รัปต์ แต่การอินเตอร์รัปต์ในไทเมอร์ 2 ก็ไม่ได้ถูกดิสเอเบิล เมื่ออยู่ในโหมดนี้ บิต EXEN2 ถูกเซต การเปลี่ยนแปลงระดับลอจิกจาก "1" เป็น "0" ที่ขา T2EX จะส่งผลให้บิต EXF2 เกิดการเซต แต่ไม่มีการเรียกค่าจากรีจิสเตอร์ RCAP2L และ RCAP2H เมื่อเป็นเช่นนี้ขา T2EX จึงสามารถใช้เป็นขาอินพุตของอินเตอร์รัปต์ได้เป็นกรณีพิเศษ

เมื่อไทเมอร์ 2 ทำงาน (บิต TR2 เป็น "1") จะถูกกำหนดให้ทำงานเป็นไทเมอร์ จึงไม่สามารถอ่านหรือเขียนค่าของรีจิสเตอร์ TL2 และ TH2 ได้ ค่ารีจิสเตอร์ไทเมอร์ จะเพิ่มขึ้นทุกๆ สเตต (1 สเตตเท่ากับ 2 คาบเวลาของสัญญาณนาฬิกา) ค่าของรีจิสเตอร์นำมาใช้สร้างสัญญาณนาฬิกาของการสื่อสารข้อมูลผ่านพอร์ตอนุกรมภายในไมโครคอนโทรลเลอร์ ในภาพที่ 2-8 แสดงการทำงานของไทเมอร์ 2 ในการกำเนิดอัตราบอด



ภาพที่ 2-6 ไดอะแกรมการทำงานในโหมดตั้งค่าการนับอัตโนมัติของไทเมอร์ 2 เมื่อบิต DCEN = "0"



ภาพที่ 2-7 ไดอะแกรมกาทำงานโหมดตั้งค่าการนับอัตโนมัติของไทเมอร์ 2 เมื่อบิต DCEN = "1"

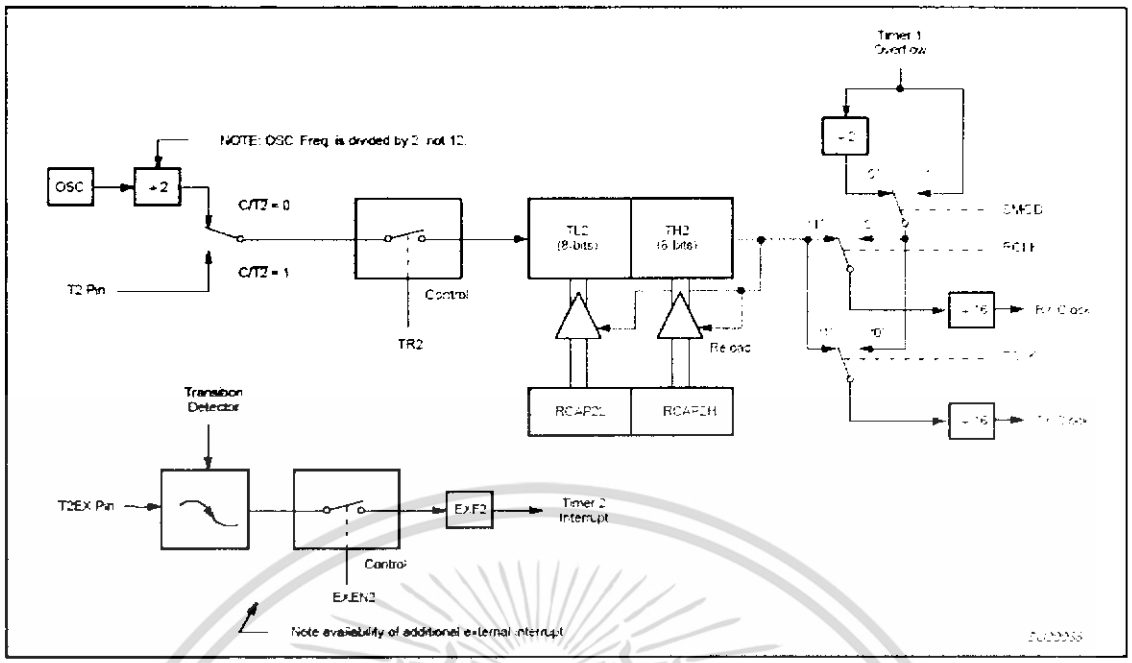
การคำนวณหาค่าอัตราบอดในโหมด 1 และ 3 ของไทเมอร์ 2 สามารถหาจากสูตร

อัตราบอด = fosc / (n*(65536-((256*RCAP2H) +RCAP2L)))

โดยที่ RCAP2H, RCAP2L เป็นข้อมูลขนาด 16 บิตแบบไม่คิดเครื่องหมาย

n มีค่าเท่ากับ 16 เมื่อทำงานในโหมด 6 ไซเกิลสัญญาณนาฬิกา และเท่ากับ 32 เมื่อทำงานในโหมด 12 ไซเกิลสัญญาณนาฬิกา

ในตารางที่ 2-2 แสดงการกำหนดค่าในรีจิสเตอร์ RCAP2H และ RCAP2L เพื่อสร้างอัตราบอดที่ความถี่สัญญาณนาฬิกาค่าต่างๆ ในโหมดการทำงาน 6 และ 12 ไซเกิลสัญญาณนาฬิกา



ภาพที่ 2-8 ไดอะแกรมการทำงานของไทเมอร์ 2 ในโหมดกำเนิดอัตราบอดสำหรับการสื่อสารข้อมูลผ่านพอร์ตอนุกรม

ตารางที่ 2-2 แสดงความสัมพันธ์ของค่าในรีจิสเตอร์ RCAP2H และ RCAP2L เมื่อใช้งาน ไทเมอร์ 2 ในการสื่อสารผ่านพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ P89C51RD2

อัตราบอด		ความถี่	ค่าของรีจิสเตอร์	
โหมด 12 ไซเกิล	โหมด 6 ไซเกิล	สัญญาณนาฬิกา	RCAP2H	RCAP2L
375 kbps	750 kbps	12 MHz	FFH	FFH
9.6 kbps	19.2 kbps	12 MHz	FFH	D9H
2.8 kbps	5.6 kbps	12 MHz	FFH	B2H
2.4 kbps	4.8 kbps	12 MHz	FFH	64H
1.2 kbps	2.4 kbps	12 MHz	FEH	C8H
300 kbps	600 kbps	12 MHz	FBH	1EH
110 kbps	220 kbps	12 MHz	F2H	AFH
300 kbps	600 kbps	12 MHz	FDH	8FH
110 kbps	220 kbps	12 MHz	F9H	57H

ตารางที่ 2-3 แสดงข้อมูลของรีจิสเตอร์ T2CON เพื่อกำหนดให้ไทเมอร์ 2 ทำงานเป็นไทมาเมอร์

โหมดการทำงานของ ไทเมอร์/เคาน์เตอร์ 2	T2CON	
	การควบคุม จากภายใน	การควบคุม จากภายนอก
16 บิตแบบตั้งค่าการนับอัตโนมัติหรือรีโหลด	00H	08H
แคปเจอร์หรือการตรวจจับสัญญาณ	01H	09H
กำเนิดอัตราบอด โดยรับและส่งเท่ากัน	34H	36H
รับข้อมูลเท่านั้น	24H	26H
ส่งข้อมูลเท่านั้น	14H	16H

หมายเหตุ

การควบคุมจากภายใน : การแคปเจอร์และรีโหลดเกิดขึ้นจากการโอเวอร์โฟลวของไทเมอร์

การควบคุมจากภายนอก : การแคปเจอร์และรีโหลดเกิดขึ้นจากการโอเวอร์โฟลวของไทเมอร์

และการเปลี่ยนแปลงระดับลอจิก “1” เป็น “0” ที่ขา T2EX ยกเว้นในกรณีที่กำหนดอัตราบอดสำหรับพอร์ตอนุกรม ในไมโครคอนโทรลเลอร์

ตารางที่ 2-4 แสดงข้อมูลของรีจิสเตอร์ T2CON เพื่อกำหนดให้ไทเมอร์ 2 ทำงานเป็นเคาน์เตอร์ ข้อมูลที่ใช้ในการเลือกโหมดการทำงานของไทเมอร์/เคาน์เตอร์ 2

โหมดการทำงานของ ไทเมอร์/เคาน์เตอร์ 2	T2CON	
	การควบคุม จากภายใน	การควบคุม จากภายนอก
16 บิตแบบตั้งค่าการนับอัตโนมัติหรือรีโหลด	02H	0AH
แคปเจอร์หรือการตรวจจับสัญญาณแบบ 16 บิต	03H	0BH

หมายเหตุ

การควบคุมจากภายใน : การแคปเจอร์และรีโหลดเกิดขึ้นจากการโอเวอร์โฟลวของไทเมอร์

การควบคุมจากภายนอก : การแคปเจอร์และรีโหลดเกิดขึ้นจากการโอเวอร์โฟลวของไทเมอร์และการเปลี่ยนแปลงระดับลอจิก "1" เป็น "0" ที่ขา T2EX ยกเว้นในกรณีที่กำหนดอัตราบอดสำหรับพอร์ตอนุกรม ในไมโครคอนโทรลเลอร์

เนื่องจากมีตัวแปรอยู่ที่ใช้การควบคุมและเลือกโหมดการทำงานของไทเมอร์/เคาน์เตอร์ 2 เช่นเดียวกับไทเมอร์/เคาน์เตอร์ 0 และ 1 โดยเฉพาะอย่างยิ่งโหมดของการทำงานที่แตกต่างกันมาก และมีความซับซ้อนเพิ่มมากขึ้น เพื่อให้เกิดความสะดวกในการงานจึงได้ทำการสรุปข้อมูลที่ใช้ในการกำเนิดรูปแบบการทำงานของไทเมอร์/เคาน์เตอร์2 ไว้ในตารางที่ 2-3 และ 2-4

2.7 การใช้งานไทเมอร์ 2 เพื่อกำเนิดสัญญาณพัลส์แบบโปรแกรมได้

อีกความสามารถพิเศษหนึ่งที่ไทเมอร์ 2 ใน ไมโครคอนโทรลเลอร์ P89C51RD2 ทำได้คือ กำเนิดสัญญาณพัลส์ดิวตี้ไซเกิล 50% แบบโปรแกรมค่าความถี่ได้ออกทางขา T2(P1.0) โดยค่าความถี่ของสัญญาณนาฬิกากำเนิดได้มีค่าอยู่ระหว่าง 61Hz - 4MHz เมื่อทำงานในโหมด 12 ไซเกิลสัญญาณ และ 122Hz – 8MHZ เมื่อทำงานในโหมด 6 ไซเกิลสัญญาณ ที่ของคริสตอล 16MHz การกำหนดให้ไทเมอร์ 2 กำเนิดสัญญาณพัลส์ในโหมดนี้สามารถทำได้โดย

1. เคลียร์บิต C/T2 ในรีจิสเตอร์ T2CON
2. เซตบิต TR2 (บิต 2) ในรีจิสเตอร์ T2CON
3. เซตบิต T2OE ในรีจิสเตอร์ T2MOD
4. กำหนดค่าในรีจิสเตอร์ RCAP2H และ RCAP2L โดยการคำนวณได้จากสูตร

อัตราบอด = fosc / (n*(65536-((256*RCAP2H) + RCAP2L)))

โดยที่ fosc คือสัญญาณนาฬิกาหลักหรือคริสตอล มีหน่วยเป็น MHz

RCAP2H, RCAP2L เป็นข้อมูลขนาด 16 บิตแบบไม่คิดเครื่องหมาย

n มีค่าเท่ากับ 16 เมื่อทำงานในโหมด 6 ไซเกิลสัญญาณนาฬิกา และเท่ากับ 32 เมื่อทำงานในโหมด 12 ไซเกิลสัญญาณนาฬิกา

อย่างไรก็ตาม เมื่อกำหนดไมโครคอนโทรลเลอร์ทำงานในงานในโหมดนี้ ไทเมอร์ 2 ต้องไม่กำเนิดสัญญาณอินเตอร์รัปต์เช่นเดียวกับการสร้างอัตราบอด

2.8 ความสามารถที่เพิ่มเติมในพอร์ตอนุกรมของ P89C51RD2

ในพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ P89C51RRD2 ได้มีการเพิ่มเติมความสามารถพิเศษอีก 2 อย่างคือ ตรวจจับการหายไปของบิตหยุดหรือปิดท้าย ซึ่งที่เรียกเหตุการณ์ที่เกิดขึ้นนี้ว่าความผิดพลาดของ เฟรมข้อมูล (framing error) และสามารถเชื่อมต่อไมโครคอนโทรลเลอร์ เข้า

ด้วยกันหลายๆตัวหรือที่เรียกว่า มัลติ – โปรเซสเซอร์ (multi-processor) โดยใช้ความสามารถรับรู้แอดเดรสในการติดต่ออัตโนมัติ (automatic address recognition)

รีจิสเตอร์ที่เกี่ยวข้องในความสามารถนี้คือ SCON ดังนั้นมีรายละเอียดการทำงานในแต่ละเอียดการทำงานในแต่ละบิตต่อไปนี้ SM0/(Serial port mode bit0/Framming error bit): ปกติจะใช้ร่วมกับบิต SM1 เพื่อกำหนดโหมดการทำงานของพอร์ตอนุกรม การเข้าถึงบิตนี้จะเกิดขึ้นได้ก็ต่อเมื่อมีการเคลียร์บิต SMOD ซึ่งก็คือ บิต 6 ของรีจิสเตอร์ PCON ในกรณีที่ใช้ความสามารถตรวจจับความผิดพลาดของแฟรมข้อมูล บิตนี้จะใช้แจ้งความผิดพลาดที่เกิดขึ้น โดยจะเซตเป็น "1" ทันทีเมื่อมีการพบว่าไม่สามารถตรวจจับบิตหรือบิตที่ปิดท้ายของข้อมูลของพอร์ตอนุกรมได้ การเอ็นเอเบิลความสามารถนำได้โดยการเซตบิต SMOD ในรีจิสเตอร์ PCON การเคลียร์บิตนี้ต้องกระทำทางซอฟต์แวร์เท่านั้น

SM1 (Serial port mode bit 1) ใช้ร่วมกับบิต SM0 ในการกำหนดโหมดการทำงานของพอร์ตอนุกรมไมโครคอนโทรลเลอร์ P89C51RD2 ดังรายละเอียดต่อไปนี้

SM0	SM1	โหมด	รายละเอียด	อัตราบอด
0	0	0	ชิฟต์รีจิสเตอร์	ความถี่สัญญาณนาฬิกา/6(ในโหมด 6 ไซเกิลสัญญาณนาฬิกา
0	1	1	UART8บิต	ปรับค่าได้
1	0	2	UART9บิต	ความถี่สัญญาณนาฬิกา/32(ในโหมด 6 ไซเกิลสัญญาณนาฬิกา
1	1	3	UART9บิต	ปรับค่าได้

SM2 (Serial port mode bit 2) : ใช้งานการเอ็นเอเบิลความสามารถการรับรู้แอดเดรสในการติดต่ออัตโนมัติ เมื่อมีการเชื่อมต่อไมโครคอนโทรลเลอร์หลายตัวเข้าด้วยกัน โดยความสามารถนี้จะใช้ได้ก็ต่อเมื่อวงจรพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ทำงานในโหมด 2 หรือ 3 ถ้าบิต SM2 เป็น "1" บิต RI จะไม่เซต เว้นแต่ข้อมูลบิตที่ 9 ที่รับเข้ามาเป็น "1" เป็นแจ้งว่า สามารถติดต่อได้ และข้อมูลที่รับเข้ามาคือ ค่าแอดเดรสที่ต้องการติดต่อด้วย

ในกรณีที่พอร์ตอนุกรมทำงานในโหมด 1 ถ้าบิต SM2 เซตบิต RI จะไม่เปลี่ยนแปลงจนกว่าจะได้รับข้อมูลบิตหยุดหรือบิตปิดท้าย และข้อมูลที่ได้รับเป็นข้อมูลแอดเดรสที่ต้องการติดต่อด้วย

ในกรณีที่วงจรพอร์ตอนุกรมทำงานในโหมด 0 บิต SM2 นี้เป็น "0"

REN (Received enable bit) : ใช้สำหรับเอ็นเอเบิลความสามารถในการรับข้อมูลของวงจรพอร์ตอนุกรม

"1" เอ็นเอเบิลการรับข้อมูล

"0" ดิสเอเบิลการรับข้อมูล

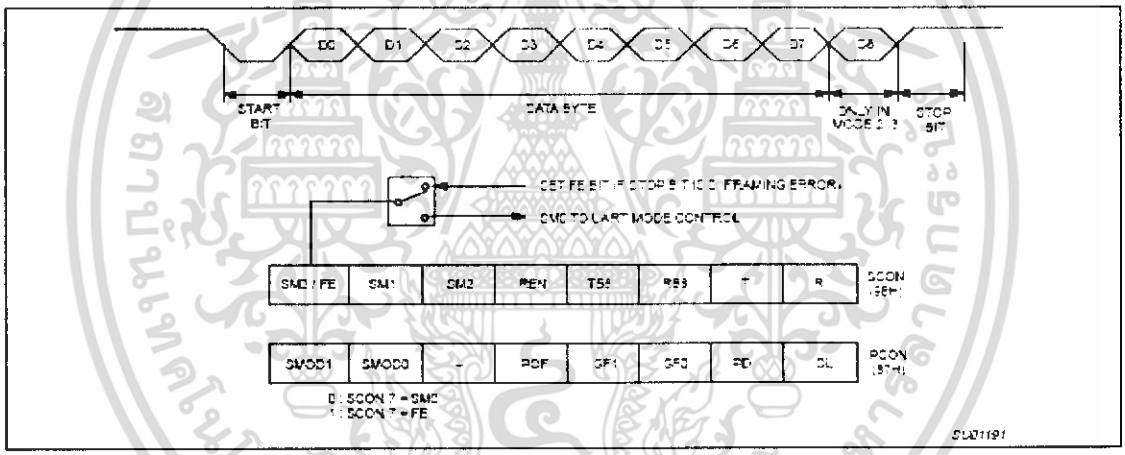
การรีเซตหรือเคลียร์บิตนี้ต้องกระทำด้วยกระบวนการทางซอฟต์แวร์

TB8 (Transmit data bit 8): ใช้เก็บข้อมูลบิต 8 หรือบิตที่ 9 ที่ต้องการส่งออกผ่านทางพอร์ตอนุกรม เมื่อทำงานในโหมด 2 และ 3

RB8 (Receive data bit 8): ใช้เก็บข้อมูลบิต 8 ของข้อมูลที่รับเข้ามาได้ของพอร์ตอนุกรม เมื่อทำงานในโหมด 2 และ 3 ในกรณีทำงานในโหมด 1 ถ้าบิต SM2 = 0 ข้อมูลของบิตหยุดเก็บไว้ที่บิต RB8 บิตนี้จะไม่ใช้งานในโหมด 0

TI (Transmit Interrupt flag): บิตแสดงการเกิดอินเตอร์รัปต์ เนื่องมาจากการส่งข้อมูลออกทางพอร์ตอนุกรม เมื่อทำงานในโหมด 0 บิตนี้จะเซตเมื่อมีการส่งข้อมูลบิต 7 หรือบิตที่ 8 ออกไป แต่ถ้าทำงานในโหมดอื่น บิตนี้จะเซตเมื่อมีการเริ่มต้นส่งบิตหยุดหรือบิตปิดท้าย การเคลียร์บิตนี้ต้องกระทำด้วยกระบวนการทางซอต์ฟแวร์

RI (Receive interrupt flag): บิตแสดงการเกิดอินเตอร์รัปต์เนื่องจากการรับข้อมูลเข้ามาของพอร์ตอนุกรม เมื่อทำงานในโหมดการรับบิตปิดท้ายดำเนินไปครึ่งทาง นอกจากนี้การเซตบิตนี้ยังมีเงื่อนไขที่กำหนดโดยบิต SM2 ร่วมด้วยการเคลียร์บิตนี้กระทำด้วยกระบวนการทางซอฟต์แวร์



ภาพที่ 2-9 แสดงรายละเอียดของกลไกในการตรวจจับความผิดพลาดของเฟรมข้อมูลในการสื่อสารผ่านพอร์ตอนุกรมของไมโครคอนโทรลเลอร์ P89C51RD2

2.9 การตรวจจับความผิดพลาดของเฟรมข้อมูล

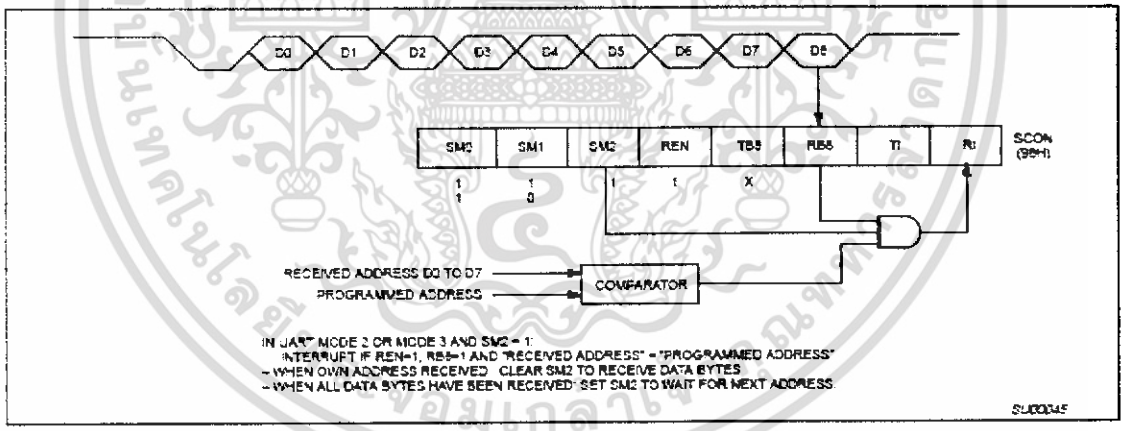
ความผิพลาดนี้จะเกิดขึ้นเมื่อบิตหยุดหรือบิตปิดท้าย (stop bit) ในเฟรมของข้อมูลชุดหนึ่งๆเกิดหายไปแล้วมีการส่งข้อมูลในชุดต่อไปมา ส่งผลให้ไมโครคอนโทรลเลอร์ไม่สามารถรับรู้การสิ้นสุดของข้อมูลแต่ละชุดได้รับข้อมูลจึงเกิดความผิดพลาดเกิดขึ้น ในวงจรสื่อสารข้อมูลผ่านพอร์ตอนุกรมภายใน ไมโครคอนโทรลเลอร์ P89C51RD2 สามารถตรวจจับความผิดพลาดที่เกิดขึ้นได้ โดยต้องทำการเอ็นเอเบิลความสามารถนี้ โดยการเซตบิต 6 ของรีจิสเตอร์ SCON ในภาพที่ 2-9 แสดงกลไกการตรวจสอบความผิดพลาดของเฟรมข้อมูล

เมื่อเกิดความผิดพลาดของเฟรมของข้อมูลขึ้น บิต 7 ของรีจิสเตอร์ SCON ซึ่งก็คือ บิต FE จะเซตบิต หากต้องการเคลียร์บิต FE นี้ต้องใช้กระบวนการทางซอฟต์แวร์เท่านั้น

2.10 การรับรู้แอดเดรสอัตโนมัติ

เมื่อมีการต่อพ่วงไมโครคอนโทรลเลอร์ P89C51RD2 เข้าด้วยกันหลายตัว โดยใช้พอร์ตอนุกรม ในการติดต่อแลกเปลี่ยนข้อมูลกันจึงต้องมีการกำหนดแอดเดรสประจำตัว ทั้งนี้เพื่อเลี่ยงความสับสน ในไมโครคอนโทรลเลอร์ P89C51RD2 จึงมีความสามารถในการรับรู้แอดเดรสอัตโนมัติ เพื่อช่วยการติดต่อแบบมัลติ-โปรเซสเซอร์สามารถกระทำได้อย่างสะดวกมกขึ้น โดยการเอ็นเอเบิลความสามารถนี้กระทำได้โดยการเซตบิต SM2 และต้องกำหนดให้พอร์ตอนุกรมการทำงานในโหมด 2 หรือ 3

กระบวนการทำงานคือในระบบมัลติ – โปรเซสเซอร์จะมีไมโครคอนโทรลเลอร์ที่เป็นตัวหลักหรือมาสเตอร์ (master) เพียงตัวเดียว ที่เหลือทั้งหมดจะเป็นตัวพ่วงหรือสเลฟ (slave) ดังนั้นในแต่ละสเลฟก็จะต้องมีการกำหนดแอดเดรสประจำตัว โดยการเก็บค่าแอดเดรสจะใช้รีจิสเตอร์ฟังก์ชั่นพิเศษ 2 ตัว ประกอบด้วย SADDR และ SADEN โดยแอดเดรสที่ได้จะเกิดจากการแอนด์กันระหว่างข้อมูลในรีจิสเตอร์ทั้งสองตัวนี้ ยกตัวอย่าง



ภาพที่ 2-10 แสดงรายละเอียดของกลไกในการใช้งานพอร์ตอนุกรมเพื่อติดต่อในลักษณะมัลติโปรเซสเซอร์ โดยใช้ความสามารถการรับรู้แอดเดรสอัตโนมัติของ P89C51RD2

ที่สเลฟ 0	ค่าในรีจิสเตอร์ SADDRเท่ากับ	1100 0000
	ค่าในรีจิสเตอร์ SADEN เท่ากับ	1111 1101
	ค่าแอดเดรสที่ได้คือ	1100 00X0
ที่สเลฟ 1	ค่าในรีจิสเตอร์ SADDR เท่ากับ	1100 0000
	ค่าในรีจิสเตอร์SADEN เท่ากับ	1111 1110

ค่าแอดเดรสที่ได้คือ 1100 000X

จะเห็นว่าข้อมูลในรีจิสเตอร์ SADDR เหมือนกัน แต่ต่างกันที่ข้อมูลในรีจิสเตอร์ SADEN ทำให้แอดเดรสของอุปกรณ์สเลฟทั้งสองตัวแตกต่างกัน อุปกรณ์สเลฟ 0 ต้องการข้อมูล "0" ที่บิต 0 และไม่สนใจข้อมูลที่บิต 1 ส่วนอุปกรณ์ สเลฟ 1 ต้องการข้อมูล "0" ที่บิต 1 และจะไม่สนใจข้อมูลที่บิต 0 ทำให้ค่าแอดเดรสของสเลฟ 0 จึงเท่ากับ11000010 ส่วนแอดเดรสเป็น 11000000 ในภาพที่ 2-10 แสดงเห็นถึงไดอะแกรมของกระบวนการรับรู้แอดเดรสอัตโนมัติเมื่อเชื่อมต่อกับไมโครคอนโทรลเลอร์ในแบบมัลติ-โปรเซสเซอร์ผ่านพอร์ตอนุกรม

ด้วยความสามารถเหล่านี้ทำให้การติดต่ออุปกรณ์ในระบบมัลติ - โปรเซสเซอร์ลดความซับซ้อนลงเพียงเขียนข้อมูลลงในรีจิสเตอร์ SADDR และSADDEN ของ ไมโครคอนโทรลเลอร์แต่ละตัวและกำหนดให้ทำงานในโหมดอย่างมากมาย ถ้าดูจากตัวอย่าง ด้วยการกำหนดค่าในรีจิสเตอร์ SADDR ที่เหมือนกัน จะเกิดอุปกรณ์สเลฟ 8 ตัวและค่าในรีจิสเตอร์ SADDR กำหนดได้มากถึง 255 ค่า จึงทำให้สามารถสร้างระบบมัลติ-โปรเซสเซอร์ที่ ไมโครคอนโทรลเลอร์ต่อร่วมกันมากถึง 2,040 ตัว หากไม่ใช้สามารถดังกล่าวการสร้างระบบมัลติ-โปรเซสเซอร์จะยุ่งยากมากต้องเสียขาพอร์ตเพื่อกำหนดแอดเดรส และต้องมีการเขียนโปรแกรมที่ซับซ้อนมาก ทั้งยังมีขีดจำกัดในจำนวนของการติดต่อด้วย

บทที่ 3
หลักการและทฤษฎีในการออกแบบ

3.1 กล่าวนำ

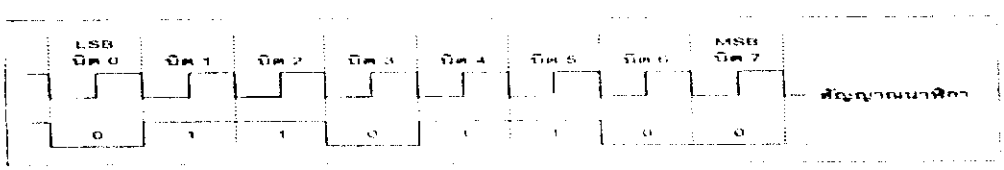
การออกแบบตัวชิ้นงานในโครงงานนี้จะกล่าวถึงหลักการและ ทฤษฎีในกระบวนการการออกแบบการติดต่อระหว่างไมโครคอนโทรลเลอร์ และคอมพิวเตอร์และอุปกรณ์ทางด้นฮาร์ดแวร์ เช่น แอลซีดี (LCD) และ เซเวนเซกเม้น (7-Segment) และการสื่อสารผ่านพอร์ตอนุกรม RS-232 เพื่อติดต่อกับคอมพิวเตอร์และไปแสดงค่าออกทาง วิชวล (Visual)

3.2 ทำความรู้จักกับพอร์ตอนุกรม

การเคลื่อนย้ายข้อมูลจากคอมพิวเตอร์ไปยังอุปกรณ์ต่อพวงอื่น ๆ หรือคอมพิวเตอร์ด้วยกันแบบอนุกรม เป็นการรับส่งข้อมูลครั้งละ 1 บิต แต่ก็สามารถรับส่งข้อมูลได้คราวละหลาย ๆ บิตได้ หากแต่จะต้องมีการตกลงกันระหว่างตัวส่งและตัวรับว่า จะรับส่งข้อมูลคราวละกี่บิต ตัวรับจะต้องรอข้อมูลมาให้ครบทุกบิตเสียก่อนจึงทำการประมวลผล ส่งผลให้การสื่อสารข้อมูลแบบอนุกรมอาจมีความเร็วต่ำกว่าแบบขนาน ในจำนวนสายสัญญาณการรับส่งข้อมูลแบบอนุกรมจะใช้จำนวนสายที่น้อยกว่ามาก แต่อย่างไรก็ตามการรับส่งข้อมูลแบบอนุกรมสามารถใช้สายสัญญาณที่มีความยาวมากกว่าแบบขนาน ทำให้ได้ระยะในการสื่อสารมากกว่าแบบขนาน

3.2.1 การสื่อสารแบบอนุกรม

การสื่อสารแบบอนุกรมนั้นจะแบ่งออกได้เป็น 2 แบบคือการสื่อสารอนุกรมแบบซิงโครนัสและการสื่อสารอนุกรมแบบอะซิงโครนัส การสื่อสารแบบซิงโครนัสจะมีสัญญาณนาฬิการ่วมอยู่กับการรับและส่งสัญญาณด้วย ตัวอย่างการส่งข้อมูลแบบซิงโครนัสก็คือคีย์บอร์ดของคอมพิวเตอร์ ซึ่งสายเส้นหนึ่งจะเป็นสายสัญญาณนาฬิกา ส่วนสายอีกเส้นจะเป็นสายข้อมูล ดังนั้นการติดต่อกันแบบซิงโครนัสนี้จะต้องใช้สายในการเชื่อมต่ออย่างน้อยที่สุด 3 เส้นคือ สัญญาณนาฬิกา, ข้อมูลและกราวด์ภาพที่ 3-1 แสดงให้เห็นถึงไทมิ่งไดอะแกรมของการส่งข้อมูลแบบซิงโครนัส



ภาพที่ 3-1 รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรม

3.2.2 การสื่อสารข้อมูลแบบอะซิงโครนัส

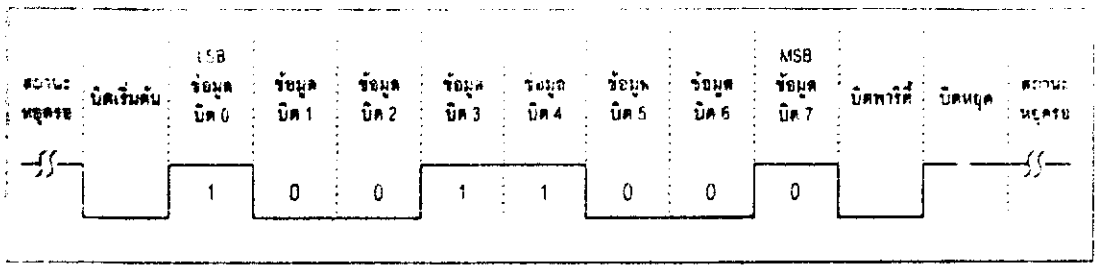
การสื่อสารข้อมูลแบบอะซิงโครนัส คือการรับและส่งข้อมูลไปในสายโดยไม่จำเป็นต้องมีสัญญาณนาฬิการ่วมด้วยเหมือนกับการรับส่งข้อมูลแบบซิงโครนัส แต่จะใช้การกำหนดค่าสัญญาณนาฬิกาทั้งภาครับและภาคส่งให้มีค่าเท่ากัน ซึ่งเรียกสัญญาณนาฬิกาที่ใช้ในการกำหนดค่าให้ภาครับและภาคส่งนี้ว่า อัตราการถ่ายทอดข้อมูล หรือ บอดเรต (baudrete) มีหน่วยเป็น บิตต่อวินาที (bit per secone : bps)

รูปแบบของข้อมูลที่ใช้ในการรับส่งแบบอะซิงโครนัสประกอบด้วย 4 ส่วนด้วยกันคือ

1. บิตเริ่มต้น (Start Bit) ซึ่งจะมีขนาด 1 บิต
2. บิตข้อมูลแบบอนุกรมจะมีขนาด 5, 6, 7 หรือ 8 บิต
3. บิตตรวจสอบพาริตี้ (Parity Bit) จะมีขนาด 1 บิตหรือไม่มี
4. บิตปิดท้าย (Stop Bit) จะมีขนาด 1, 1.5 หรือ 2 บิต

ภาพที่ 3-2 แสดงรูปแบบของข้อมูลอนุกรมแบบอะซิงโครนัส เมื่อไม่มีข้อมูลที่จะส่ง ขา DATA จะมีสถานะลอจิก " 1 " ซึ่งจะเรียกสถานะนี้ว่าสถานะหยุดรอ (waiting stage) การเริ่มต้นส่งข้อมูลจะเริ่มจากการให้ขา DATA มีลอจิก " 0 " ด้วยช่วงระยะเวลา 1 บิต เรียกบิตนี้ว่า บิตเริ่มต้น จากนั้นบิตข้อมูลจะถูกส่งออกไป โดยเริ่มจากบิตที่มีนัยสำคัญต่ำสุด (LSB) ก่อน ซึ่งข้อมูลในไบต์ที่จะส่งอาจจะมีจำนวนบิต 5, 6, 7 หรือ 8 บิตก็ได้ จากนั้นก็ตามด้วย บิตพาริตี้ ซึ่งใช้เพื่อตรวจสอบความผิดพลาดที่เกิดขึ้นจากการส่งข้อมูล บิตสุดท้ายที่ส่งคือ บิตปิดท้าย ซึ่งจะให้ขา DATA มีสถานะลอจิก " 1 " อีกครั้งด้วยระยะเวลาอย่างน้อย 1, 1.5 หรือ 2 บิต เพื่อเป็นการแสดงว่าสิ้นสุดข้อมูลแล้ว

อุปกรณ์พิเศษที่ได้รับการออกแบบมาสำหรับการรับและส่งข้อมูลแบบอะซิงโครนัสเรียกว่า Universal Asynchronous Receiver / Transmitter หรือ UART อัตราความเร็วในการรับและส่งข้อมูลของการรับส่งข้อมูลแบบอะซิงโครนัสคือ ค่าบอดเรต ซึ่งก็คือค่าจำนวนบิตต่อวินาทีที่ใช้ในการรับและส่งข้อมูล บอดเรตมาตรฐานที่ใช้สำหรับพอร์ตอนุกรม RS-232 ได้แก่ 110, 150, 300, 600, 1200, 2400, 4800, 9600 และ 19200 บิตต่อวินาที และมีค่าเพิ่มขึ้นตามเทคโนโลยีของคอมพิวเตอร์ ซึ่งการรับส่งแบบอนุกรมโดยไม่ผ่านโมเด็มอาจจะสามารถกำหนดค่าบอดเรตได้สูงถึง 115200 บิตต่อวินาที เนื่องจากบอดเรตคือจำนวนบิตของข้อมูลที่สามารถถ่ายทอดได้ภายใน 1 วินาที ยกตัวอย่าง ข้อมูลอนุกรมถูกส่งในลักษณะ 8 บิต ไม่มีการตรวจสอบพาริตี้ มีบิตเริ่มต้น 1 บิต และบิตปิดท้าย 1 บิต ความยาวของข้อมูลที่รับส่งนี้เท่ากับ 10 บิต ถ้าใช้บอดเรตในการส่งข้อมูลเท่ากับ 9600 บิตต่อวินาที ก็จะสามารถรับส่งข้อมูลได้ด้วยความเร็ว 960 ไบต์ต่อวินาที และถ้ามีการใช้พาริตี้ความเร็วในการรับส่งข้อมูลจะเหลือเป็น 872 ไบต์ต่อวินาที



ภาพที่ 3-2 รูปแบบอย่างง่ายที่สุดของข้อมูลอนุกรมแบบอะซิงโครนัส

การตรวจสอบพาริตี้สามารถกำหนดให้เป็นแบบคี่ (odd), แบบคู่ (even) หรือไม่มีการตรวจสอบพาริตี้ก็ได้ การตรวจสอบพาริตี้เป็นการตรวจสอบจำนวนรวมของบิตที่เป็นลอจิก " 1 " ภายในข้อมูลที่ส่งไป 1 ไบต์ว่ามีจำนวนรวมเป็นเลขคู่หรือเลขคี่โดยต้องรวมบิตพาริตี้เข้าไปด้วย ยกตัวอย่างข้อมูลที่จะทำการส่งมีขนาด 8 บิตและมีความยาวเท่ากับ 99 ฐานสิบหก หรือ 1001 1001 ฐานสอง จะเห็นว่าข้อมูลในไบต์นี้มีจำนวนลอจิก " 1 " จำนวน 4 ตัวซึ่งเป็นเลขคู่ ดังนั้นถ้ากำหนดค่าพาริตี้เป็นคู่ ค่าในบิตพาริตี้จะต้องมีลอจิกเป็น " 0 " แต่ถ้าพาริตี้เป็นคี่ ค่าในบิตพาริตี้จะต้องเป็น " 1 " เพื่อให้ข้อมูล 1 ไบต์รวมทั้งบิตพาริตี้มีจำนวนบิตที่มีลอจิกเป็น " 1 " รวมกันเป็นเลขคี่ ในตารางที่ 3-1 แสดงตัวอย่างของบิตพาริตี้ในการรับส่งข้อมูลอนุกรม

บิตพาริตี้ถูกสร้างขึ้นจากภาคส่งข้อมูลของ UART โดยภาครับจะต้องกำหนดคุณสมบัติการตรวจสอบพาริตี้ให้ตรงกันว่า จะตรวจสอบพาริตี้คี่หรือคู่ จากนั้นภาครับของ UART จะตรวจสอบค่าพาริตี้ที่เกิดขึ้นว่าเป็นคู่หรือเป็นคี่ โดยการนับจำนวนลอจิก " 1 " ทั้งหมดรวมทั้งบิตพาริตี้ด้วย ถ้ากำหนดพาริตี้ไว้เป็นคู่แต่อ่านค่าตัวเลขในการนับออกมาได้ตัวเลขเป็นคี่ ทางภาครับจะแจ้งข้อผิดพลาดให้ผู้ใช้ทราบ นับเป็นการตรวจสอบความผิดพลาดของการถ่ายทอดข้อมูลที่ง่ายที่สุด แต่จะเชื่อถือได้เมื่อบิตข้อมูลที่ทำการส่งผิดพลาดเพียงบิตเดียวเท่านั้น ถ้าข้อมูลที่ส่งมีบิตที่ผิดพลาดมากกว่า 1 บิต การตรวจสอบด้วยวิธีนี้จะไม่ได้ผล สำหรับการตั้งพาริตี้บิตเป็น NONE นั้นทั้งภาครับและภาคส่งจะไม่มีการตรวจสอบพาริตี้

ตารางที่ 3-1 แสดงบิตพาริตี้ของข้อมูล

ข้อมูล	บิตพาริตี้คู่	บิตพาริตี้คี่
0000 0000	0	1
0000 0001	1	0
0000 0010	1	0
0000 0011	0	1
0000 0100	1	0
1111 1110	1	0
1111 1111	0	1

คอมพิวเตอร์ในรุ่น AT เกือบทั้งหมดจะใช้ UART เบอร์ 16450 และ 16550 ส่วนคอมพิวเตอร์ในรุ่น XT ใช้ UART เบอร์ 3250 UART ชิพเหล่านี้มีระดับแรงดันเป็นแบบทีทีแอล (0 และ +5 V) แต่เพื่อให้มีแรงดันเป็นไปตามมาตรฐาน RS-232 และเพื่อให้การรับส่งข้อมูลสามารถทำได้ที่ระยะทางไกลมากขึ้น ระดับแรงดันทีทีแอลจะถูกแปลงเป็นระดับแรงดันที่สูงขึ้นโดยลอจิก " 0 " มีระดับแรงดัน +3 V ถึง +12 V ในขณะที่ลอจิก " 1 " มีระดับแรงดัน -3 V ถึง -12 V

3.2.3 มาตรฐานพอร์ตอนุกรมแบบ RS-232

มาตรฐานการเชื่อมต่อแบบอนุกรม RS-232 เป็นมาตรฐานอุตสาหกรรมที่ออกแบบมา เพื่อใช้ในการส่งข้อมูลอนุกรมแบบอะซิงโครนัส 2 ทิศทาง โดยมาตรฐาน RS-232 ในอดีตนั้นถูกออกแบบมาเพื่อการส่งผ่านข้อมูลจากคอมพิวเตอร์ไปยังโมเด็มเพียงอย่างเดียว เพื่อที่จะนำข้อมูลจากโมเด็มนี้สื่อสารผ่านสายโทรศัพท์ไปยังคอมพิวเตอร์อีกชุดซึ่งอยู่ห่างไกลกัน โดยคณะกรรมการที่เรียกว่า สมาคมอุตสาหกรรมอิเล็กทรอนิกส์ (Electronic Industries Association : EIA) ได้ว่างมาตรฐานที่มีชื่อเรียกกันว่า EIA RS-232 มาตรฐานนี้ในช่วงแรกจะใช้คอนเน็กเตอร์เป็นแบบ DB-25 โดยกำหนดความยาวสูงสุดของสายสัญญาณไว้ที่ 50 ฟุต มีระดับสัญญาณตั้งแต่ -3 V ถึง -12 V แสดงว่ามีข้อมูล (Mark) และ +3 V ถึง +12 V แสดงว่าเป็นช่องว่าง (Space)

มาตรฐาน RS-232 ได้กำหนดรูปแบบของอุปกรณ์เชื่อมต่อข้อมูล (Data Terminal Equipment : DTE) กับวงจรข้อมูลปลายทาง (Data Circuit Terminating : DCE) ไว้ว่า อุปกรณ์ DTE จะต้องเป็นอุปกรณ์ที่มีการประมวลผลในตัวเช่น ไมโครคอนโทรลเลอร์หรือไมโครคอมพิวเตอร์ ซึ่งมีความสามารถในการสร้างข้อมูลแบบอนุกรมได้ ส่วนอุปกรณ์ DCE จะทำหน้าที่เป็นเพียงตัวรับข้อมูลที่ส่งมาจาก DTE เท่านั้น โดยการรับส่งข้อมูลระหว่างอุปกรณ์ทั้งสองจะกระทำผ่านมาตรฐาน RS-232

ข้อแตกต่างของอุปกรณ์ DTE และอุปกรณ์ DCE อย่างหนึ่งที่เห็นได้ชัดคือ คอนเน็กเตอร์ของ DTE จะเป็นตัวผู้ ส่วนคอนเน็กเตอร์ของ DCE จะเป็นตัวเมีย ซึ่งพอร์ตอนุกรมของคอมพิวเตอร์ที่ใช้กันอยู่ทั่วไปจะเป็นแบบ DTE ส่วนคอนเน็กเตอร์ที่อยู่ที่โมเด็มจะเป็นแบบ DCE

สำหรับการใช้งานคอมพิวเตอร์ พอร์ตอนุกรม RS-232 มักถูกใช้เชื่อมต่อกับโมเด็มหรือเมาส์ โดยสามารถรับส่งข้อมูลได้ที่ความยาวของสายส่งสัญญาณสูงสุดถึง 20 เมตร

3.2.4 คอนเน็กเตอร์สำหรับพอร์ต RS-232 และการเชื่อมต่อ

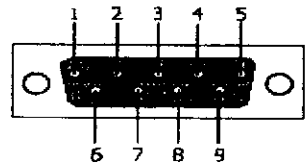
มาตรฐานการเชื่อมต่อแบบ RS-232 จะใช้คอนเน็กเตอร์แบบ DB-25 ตัวผู้หรือ DB-9 ตัวผู้ ซึ่งคอนเน็กเตอร์แบบ DB-25 จะมีขาต่อใช้งานเพียง 9 เส้นเช่นเดียวกับคอนเน็กเตอร์แบบ DB-9 เนื่องจากขาอื่น ๆ ที่เคยใช้ในอดีต ปัจจุบันมีการใช้งานไม่มากนัก จึงถูกยกเลิกไป โดยแสดงรูปร่างและตำแหน่งขาในภาพที่ 3-3

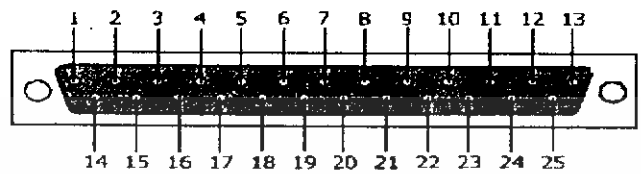
สำหรับการเชื่อมต่อคอมพิวเตอร์กับอุปกรณ์ภายนอกดังแสดงในภาพที่ 3-4 ลูกศรในรูปแสดงถึงทิศทางของข้อมูล ในภาพที่ 3-4 (ก) เป็นการเชื่อมต่อแบบ Null modem หรือการเชื่อมต่อโดยตรงโดยไม่ต้องผ่านโมเด็ม โดยมีการตรวจสอบหรือแฮนด์เช็คเต็มรูปแบบ ส่วนภาพที่ 3-4 (ข) เป็นการเชื่อมต่อแบบ Null modem ในลักษณะที่ใช้สายสัญญาณเพียง 3 เส้น โดยเส้นหนึ่งสำหรับส่งข้อมูล อีกเส้นสำหรับรับข้อมูล และเส้นสุดท้ายเป็นกราวด์ สำหับรายละเอียดหน้าที่การทำงานในแต่ละขาของพอร์ตอนุกรม RS-232 มีดังนี้

การส่งสัญญาณพาห์จากอุปกรณ์สื่อสารข้อมูลเช่น โมเด็ม สำหรับการใช้งานปกติ ขานี้จะไม่ได้ถูกใช้งานมากนัก

• **Data Carrier Detect : DCD** หรืออาจเรียกว่า Carrier Detect : CD ขานี้จะแอคทีฟเมื่อมีการส่งสัญญาณพาห์จากอุปกรณ์สื่อสารข้อมูลเช่น โมเด็ม สำหรับการใช้งานปกติ ขานี้จะไม่ได้ถูกใช้งานมากนัก

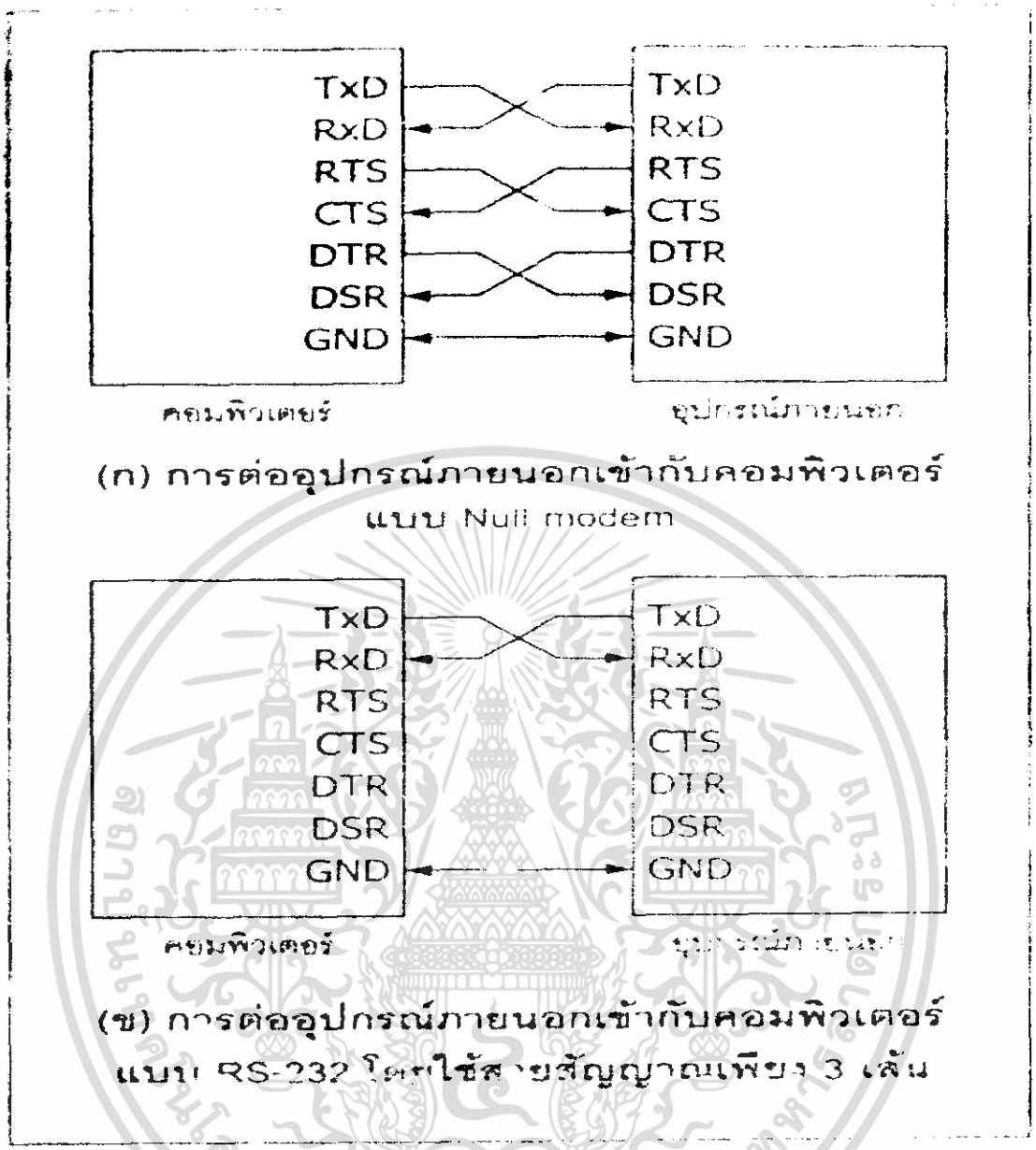
• **Receive Data :** RD หรือ RxD ขานี้ใช้เพื่อรับสัญญาณอนุกรมเข้ามายังคอมพิวเตอร์ โดยนำข้อมูลที่อ่านได้เก็บไว้ในรีจิสเตอร์บัฟเฟอร์





คอนเน็กเตอร์ DB-9	คอนเน็กเตอร์ DB-25	ชื่อของสายสัญญาณ	ชนิดของสายสัญญาณ
1	8	Data Carrier Detect : DCD	อินพุต
2	3	Received Data : RxD	อินพุต
3	2	Transmitted Data : TxD	เอาต์พุต
4	20	Data Terminal Ready : DTR	เอาต์พุต
5	7	Signal Ground	-
6	6	Data Set Ready : DSR	อินพุต
7	4	Request To Send : RTS	เอาต์พุต
8	5	Clear To Send : CTS	อินพุต
9	22	Ring Indicator : RI	อินพุต

ภาพที่ 3-3 การจัดขาของคอนเน็กเตอร์พอร์ตอนุกรมตามมาตรฐาน RS-232 ทั้งแบบ DB-9 และ DB-25



ภาพที่ 3-4 การต่ออุปกรณ์ภายนอกกับพอร์ตอนุกรมของคอมพิวเตอร์ในลักษณะต่าง ๆ

• **Data Terminal Ready :** DTR เป็นขาสัญญาณที่ส่งออกจากคอมพิวเตอร์เพื่อให้อุปกรณ์ปลายทางรับรู้ว่าต้องการติดต่อด้วย โดยขา DTR ของอุปกรณ์ปลายทางต้องเชื่อมต่อกับขา DSR ของคอมพิวเตอร์ ถ้าใช้การเชื่อมต่อแบบ Null Modem ซึ่งใช้สายในการเชื่อมต่อเพียง 3 เส้น จะต้องต่อขา DTR และ DSR ของตัวมันเองเข้าด้วยกันและต้องต่อกับขา DCD ด้วยในกรณีที่โปรแกรมสื่อสารที่ใช้มีการตรวจสอบสัญญาณพาห์

• **Signal Ground :** GND กราวด์ระบบ

• **Data Set Ready :** DSR ขานี้จะใช้คู่กับขา DTR เพื่อตรวจสอบการเชื่อมต่อกันระหว่างคอมพิวเตอร์กับอุปกรณ์ปลายทาง ซึ่งขา DSR นี้จะเป็นขาสำหรับรับข้อมูลจากภายนอก ซึ่งถูกส่งมาจากขา DTR

• **Request To Send :** RTS เป็นขาสำหรับส่งสัญญาณร้องขอให้ทางอุปกรณ์ปลายทางส่งข้อมูลกลับมายังคอมพิวเตอร์ โดยขาที่รับสัญญาณ RTS ก็คือขา CTS ในกรณีที่ใช้การเชื่อมต่อแบบ Null Modem จะต้องเชื่อมต่อขา RTS และ CTS ของตัวมันเองเข้าด้วยกัน เพื่อจะให้การรับและส่งข้อมูลสามารถเกิดขึ้นได้ตลอดเวลา

• **Clear To Send :** CTS ขานี้จะคอยรับสัญญาณจากขา RTS เมื่อรับสัญญาณได้ ข้อมูลที่ขา TxD จะถูกส่งออกไป ดังนั้นขานี้จึงถูกใช้เพื่อตรวจสอบอุปกรณ์ต่อพ่วงว่าพร้อมที่จะรับข้อมูลหรือไม่

• **Ring Indication :** RI ใช้แสดงสถานะสัญญาณเรียกจากสายโทรศัพท์ ปกติในการสื่อสารโดยทั่วไปสายนี้จะไม่ถูกใช้งาน จะใช้สายนี้ก็ต่อเมื่อมีการเชื่อมต่อกับโมเด็มและโปรแกรมมีการตรวจสอบสัญญาณนี้เท่านั้น

3.2.5 UART

มาจากคำว่า *Universal Asynchonous Receiver/ Transmitter* ซึ่งหมายถึงอุปกรณ์ที่ทำหน้าที่รับและส่งข้อมูลแบบอะซิงโครนัสนั่นเอง สำหับการสื่อสารอนุกรมบนคอมพิวเตอร์แล้ว UART ถือว่าเป็นหัวใจสำคัญของการสื่อสารอนุกรม

หน้าที่หลักของ UART คือทำหน้าที่แปลงข้อมูลที่อยู่ในรูปแบบขนานจากคอมพิวเตอร์ให้อยู่ในรูปแบบอนุกรมแบบอะซิงโครนัสแล้วส่งออกไป และทำหน้าที่แปลงสัญญาณอนุกรมแบบอะซิงโครนัสที่ป้อนเข้ามายัง UART ให้เป็นแบบขนานก่อนที่จะส่งเข้าสู่คอมพิวเตอร์ ซึ่งนอกจาก UART จะส่งข้อมูลไปยังคอมพิวเตอร์แล้ว ยังแจ้งข้อมูลอื่น ๆ ให้คอมพิวเตอร์ทราบด้วย เช่น อัตราเร็วในการรับส่งข้อมูล (บอดเรต), รูปแบบการส่งข้อมูล, ความผิดพลาดที่เกิดขึ้นระหว่างการถ่ายทอดข้อมูล (ผิดพลาดจากพาริตี้, เฟรมข้อมูล, โอเวอร์รัน) เป็นต้น

ภายใน UART จะมีส่วนของวงจรสร้างบอดเรตแบบโปรแกรมได้ โดยการกำหนดค่าตัวหารให้กับสัญญาณนาฬิกาของ UART โดยตัวหารนี้มีขนาด 16 บิต ดังนั้นจึงสามารถกำหนดตัวหารอยู่ในช่วง 1 – 65,535 UART สามารถรับส่งข้อมูลได้ทั้งแบบฮาล์ฟดูเพล็ก (Half Duplex) และฟูลดูเพล็ก (Full Duplex) โดยการส่งแบบฮาล์ฟดูเพล็กเป็นการส่งแบบทิศทางเดียว ส่วนการส่งแบบฟูลดูเพล็กนั้นสามารถรับและส่งข้อมูลได้ในคราวเดียวกัน

3.2.6 ชนิดของ UART

ในเครื่องคอมพิวเตอร์ทั่วไปมี UART ที่ใช้งานกันอยู่ 2 เบอร์คือ 8250 ซึ่งเป็น UART มาตรฐานที่ใช้กันมายาวนาน UART เบอร์นี้จะมีบัฟเฟอร์สำหรับรับและส่งข้อมูลตำแหน่งเดียวกัน ทำให้การรับและส่งข้อมูลถูกจำกัดความเร็วอยู่ที่ 57.6 กิโลบิตต่อวินาทีเท่านั้น แต่ UART เบอร์นี้ก็ถือว่าเป็นต้นแบบของ UART ที่ใช้ในคอมพิวเตอร์ โดยคอมพิวเตอร์ทุก ๆ รุ่นจะต้องสนับสนุนการทำงานตามรูปแบบของ UART เบอร์นี้

UART อีกเบอร์หนึ่งคือ 16450 มีความสามารถรับส่งข้อมูลได้ที่ความเร็ว 115,200 บิตต่อวินาที และเพิ่มรีจิสเตอร์สำหรับพักข้อมูลสำหรับ UART นอกจากนั้นยังเพิ่มส่วนของชิพรีจิสเตอร์แบบ FIFO (First In First Out) ขนาด 16 ไบต์เข้าไป ทำให้สามารถสนับสนุนความเร็วในการรับส่งข้อมูลที่ 256 กิโลบิตต่อวินาทีได้ โดยคอมพิวเตอร์ในปัจจุบันใช้ UART เบอร์นี้หรือใหม่กว่า เช่น TL16C750 ซึ่งมีรีจิสเตอร์แบบ FIFO ขนาด 64 ไบต์ ทำงานได้ที่ระดับแรงดัน + 5 V และ + 3 V มีโหมดประหยัดพลังงาน สามารถรับส่งข้อมูลได้ที่ความเร็ว 1 เมกะบิตต่อวินาที เมื่อใช้สัญญาณนาฬิกา 16 MHz

อย่างไรก็ตาม ความเร็วในการส่งข้อมูลที่มากมายของ UART เบอร์ใหม่ ๆ ก็ไม่ได้ช่วยให้การรับส่งข้อมูลของคอมพิวเตอร์เร็วขึ้น เนื่องจากว่าคอมพิวเตอร์ยังใช้ความถี่ของสัญญาณนาฬิกาในการแปลงข้อมูลเพียง 1.8432 MHz เท่านั้น

เครื่องคอมพิวเตอร์โดยทั่วไปสามารถต่อพอร์ตอนุกรมสูงสุดได้ 4 พอร์ต มีชื่อเรียกเป็น COM1, COM2, COM3 และ COM4 ซึ่งพอร์ตอนุกรมแต่ละตัวต่างใช้งาน UART ภายในคอมพิวเตอร์ในการติดต่อกับอุปกรณ์ภายนอกเช่นเดียวกัน ในภาพที่ 3-5 แสดงผังการทำงานภายในของพอร์ตอนุกรมประกอบด้วยรีจิสเตอร์ 8 บิต 8 ตัวที่ใช้งานร่วมกับ UART แอดเดรสของรีจิสเตอร์ภายในพอร์ตอนุกรมสามารถคำนวณได้จากค่ารีจิสเตอร์พื้นฐานของพอร์ตอนุกรม ยกตัวอย่างพอร์ตอนุกรม COM1 มีแอดเดรสอยู่ที่ 3F8H ตำแหน่งของรีจิสเตอร์ต่าง ๆ จะเป็นตำแหน่งที่บวกเข้าไปกับค่า 3F8H โดย

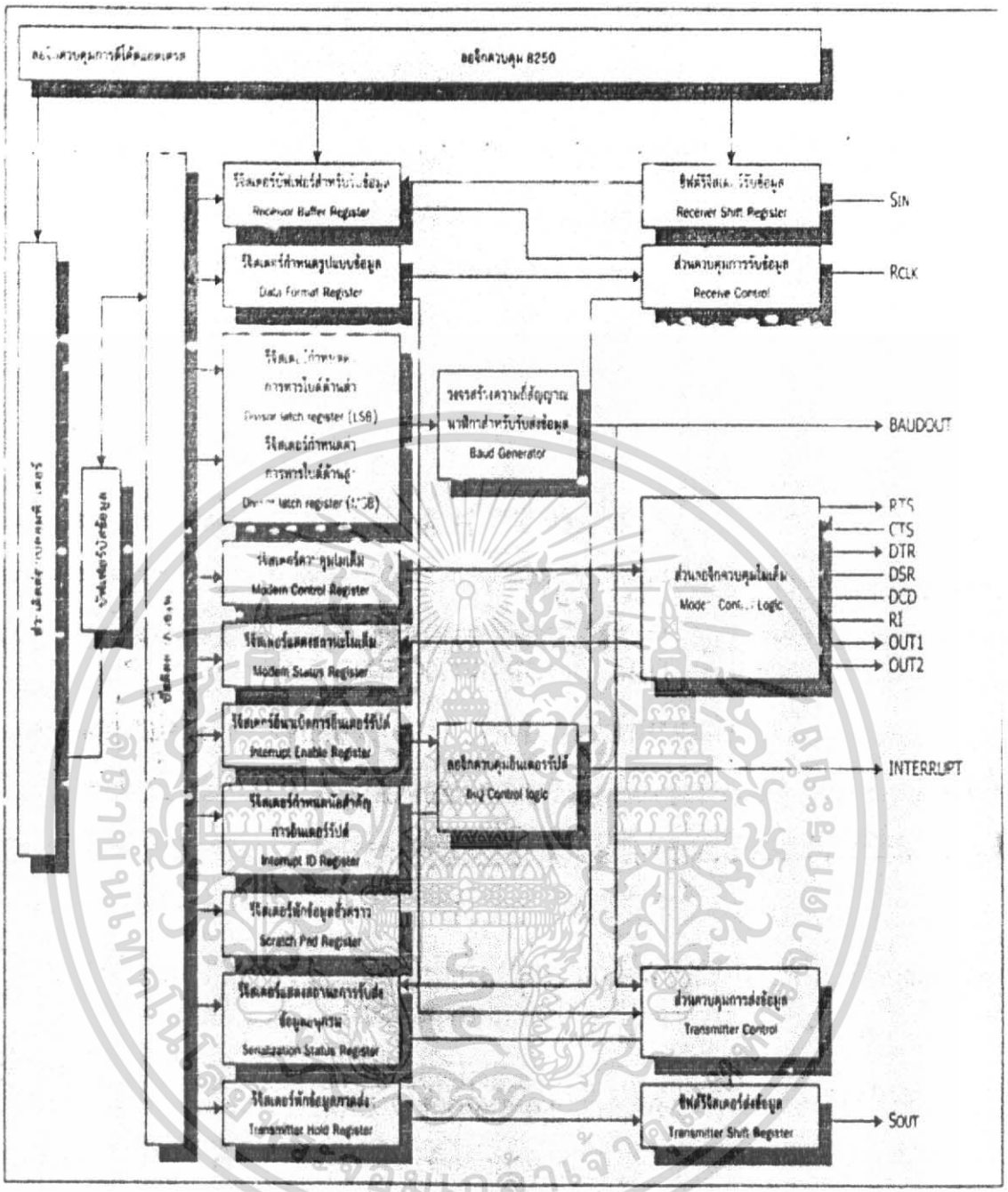
3.3 รีจิสเตอร์ที่ใช้งานกับพอร์ตอนุกรมมีดังนี้

00H เป็นรีจิสเตอร์บัฟเฟอร์สำหรับเก็บข้อมูลที่รับเข้ามาหรือเตรียมข้อมูลก่อนที่ส่งออกไป

01H รีจิสเตอร์เอ็นเอเบิลการอินเตอร์รัปต์ ใช้เซตโหมดการอินเตอร์รัปต์ของพอร์ตอนุกรม

02H รีจิสเตอร์แสดงโหมดการอินเตอร์รัปต์ ใช้เพื่อตรวจสอบโหมดของการอินเตอร์รัปต์

03H รีจิสเตอร์กำหนดรูปแบบของข้อมูล



ภาพที่ 3-5 ไดอะแกรมการทำงานภายในของพอร์ตอนุกรมของเครื่องคอมพิวเตอร์

05H รีจิสเตอร์แสดงสถานะการรับและการส่งข้อมูลแบบอนุกรม

06H รีจิสเตอร์แสดงสถานะของโมเด็มซึ่งจะแสดงสถานะของขา DCD, RI, DSR และ CTS

07H รีจิสเตอร์สำหรับการเก็บข้อมูลชั่วคราว

3.3.1 รีจิสเตอร์ตำแหน่ง 00H : รีจิสเตอร์บัฟเฟอร์

เป็นรีจิสเตอร์เก็บข้อมูลที่รับเข้ามาและส่งออก โดยการติดต่อกับรีจิสเตอร์นี้เพื่อเก็บข้อมูลจะต้องกำหนดให้บิต DLAB ในรีจิสเตอร์กำหนดรูปแบบข้อมูล (03H) มีสถานะเป็น " 0 " ซึ่งการเขียนข้อมูลมายังแอดเดรสนี้ เป็นการส่งข้อมูลไปยังรีจิสเตอร์ส่งข้อมูลและข้อมูลจะถูกส่งออกไปแบบอนุกรม สำหรับการรับข้อมูล เมื่อรับเข้ามาแล้วจะส่งต่อไปยังรีจิสเตอร์เก็บข้อมูล หลังจากอ่านค่าจากรีจิสเตอร์นี้ออกไป รีจิสเตอร์นี้จะถูกเคลียร์ และเตรียมพร้อมสำหรับรับข้อมูลใบไบต์ต่อไป

3.3.2 รีจิสเตอร์ตำแหน่ง 01H : รีจิสเตอร์เอ็นเอเบิลการอินเตอร์รัปต์

เป็นรีจิสเตอร์สำหรับการเอ็นเอเบิลการอินเตอร์รัปต์ ซึ่งเป็นการกำหนดให้ UART สร้างสัญญาณอินเตอร์รัปต์ขึ้นมา ฟังก์ชันการทำงานในแต่ละบิตของรีจิสเตอร์นี้มีดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
0	0	0	0	SINP	ERBK	TBE	RxRD

บิต 4-7 บิตเหล่านี้ไม่ถูกใช้งาน กำหนดให้เท่ากับ " 0 "

SINP เอ็นเอเบิลการอินเตอร์รัปต์เนื่องจากเกิดการเปลี่ยนแปลงสถานะที่ขาอินพุต CTS, DSR, DCD หรือขา RI
" 1 " เอ็นเอเบิลการอินเตอร์รัปต์
" 0 " ดิสเอเบิล

ERBK เอ็นเอเบิลการอินเตอร์รัปต์เนื่องจากการเกิดความผิดพลาดขึ้นด้วยสาเหตุจากพาริตี้, โอเวอร์รัน, เฟรมข้อมูล หรือการเบรกข้อมูล
" 1 " เอ็นเอเบิลการอินเตอร์รัปต์
" 0 " ดิสเอเบิล

TBE เอ็นเอเบิลการอินเตอร์รัปต์เมื่อรีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูลว่าง
" 1 " เอ็นเอเบิลการอินเตอร์รัปต์
" 0 " ดิสเอเบิล

RxRD เอ็นเอเบิลการอินเตอร์รัปต์เนื่องจากรีจิสเตอร์บัฟเฟอร์ได้รับข้อมูลแล้ว
" 1 " เอ็นเอเบิลการอินเตอร์รัปต์
" 0 " ดิสเอเบิล

3.3.3 รีจิสเตอร์ตำแหน่ง 02H : รีจิสเตอร์แสดงโหมดและสถานการณ์อินเตอร์รัปต์

มีรายละเอียดของแต่ละบิตดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
0	0	0	0	0	ID1	ID0	PND

บิต 3-7 ไม่ได้ใช้งาน อ่านค่าได้เท่ากับ " 0 "

ID1, ID0 ใช้งานร่วมกันเพื่อแจ้งสาเหตุของการเกิดอินเตอร์รัปต์

" 00 " เกิดการอินเตอร์รัปต์เนื่องจากการเปลี่ยนแปลงของขาอินพุตขึ้น การอินเตอร์รัปต์แบบนี้มีนัยสำคัญเป็นอันดับ 4

" 01 " เกิดการอินเตอร์รัปต์เนื่องจากรีจิสเตอร์บัฟเฟอร์ส่งข้อมูลว่างขึ้น การอินเตอร์รัปต์แบบนี้มีนัยสำคัญเป็นอันดับ 3

" 10 " เกิดการอินเตอร์รัปต์เนื่องจาก้อมูลถูกเก็บลงในรีจิสเตอร์บัฟเฟอร์สำหรับข้อมูลเรียบร้อยแล้ว การอินเตอร์รัปต์แบบนี้มีนัยสำคัญเป็นอันดับ 2

" 11 " เกิดการอินเตอร์รัปต์เนื่องจากความผิดพลาดในการถ่ายทอดข้อมูล หรือเกิดการเบรกการถ่ายทอดข้อมูลกะทันหัน การอินเตอร์รัปต์แบบนี้นัยสำคัญเป็นอันดับ 1 หรือมีนัยสำคัญสูงสุด

PND ใช้แสดงสถานะของการเกิดอินเตอร์รัปต์

" 1 " แสดงว่าไม่มีการอินเตอร์รัปต์

" 0 " แสดงว่ามีการอินเตอร์รัปต์เกิดขึ้น

เมื่อมีการสร้างสัญญาณอินเตอร์รัปต์ขึ้น จะต้องมีการเคลียร์ค่าก่อนที่จะให้เกิดอินเตอร์รัปต์ครั้งต่อไป โดยสามารถทำได้ดังนี้คือ

- ถ้าเกิดอินเตอร์รัปต์เนื่องจากการเปลี่ยนแปลงของขาอินพุตจะต้องอ่านค่าจากรีจิสเตอร์แสดงสถานะของโมเด็ม (รีจิสเตอร์ตำแหน่ง 06H) เพื่อเคลียร์ค่าการอินเตอร์รัปต์
- ถ้าเกิดอินเตอร์รัปต์เนื่องจากบัฟเฟอร์ส่งข้อมูลว่าง จะต้องเขียนข้อมูลไปยังรีจิสเตอร์บัฟเฟอร์ส่งข้อมูล (รีจิสเตอร์ตำแหน่ง 00 H) หรืออ่านค่ารีจิสเตอร์แสดงสถานะอินเตอร์รัปต์ (รีจิสเตอร์ตำแหน่ง 02H) เพื่อเคลียร์ค่าการอินเตอร์รัปต์
- ถ้าเกิดอินเตอร์รัปต์เนื่องจากการเก็บข้อมูลลงในรีจิสเตอร์บัฟเฟอร์ สำหรับรับข้อมูลเรียบร้อย จะต้องเคลียร์ค่าอินเตอร์รัปต์โดยการอ่านข้อมูลจากรีจิสเตอร์บัฟเฟอร์

• ถ้าเกิดอินเตอร์รัปต์เนื่องจากความผิดพลาดในการรับส่งข้อมูล หรือเกิดการเบรก จะต้องเคลียร์ค่าอินเตอร์รัปต์โดยการอ่าค่ารีจิสเตอร์แสดงสถานการณ์รับและส่งข้อมูล

3.3.4 รีจิสเตอร์ตำแหน่ง 03H : รีจิสเตอร์กำหนดรูปแบบของข้อมูล

มีรายละเอียดของแต่ละบิตดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
DLAB	BRK	PAR2	PAR1	PAR0	STOP	DAB1	DAB0

DLAB ใช้ในการกำหนดหน้าที่การทำงานของรีจิสเตอร์บัฟเฟอร์ (00H)

" 1 " เป็นการเข้าสู่โหมดการหารบอดเรต

" 0 " เป็นการเข้าถึงรีจิสเตอร์บัฟเฟอร์ (รีจิสเตอร์ตำแหน่ง 00H) และรีจิสเตอร์สำหรับเอ็นเอเบิลการอินเตอร์รัปต์ (รีจิสเตอร์ตำแหน่ง 01H) เมื่อบิต DLAB เป็น " 1 " รีจิสเตอร์บัฟเฟอร์ และรีจิสเตอร์เอนเอเบิลการอินเตอร์รัปต์จะใช้ สำหรับโหลดค่าการหารความถี่สำหรับกำหนดค่าบอดเรต โดยรีจิสเตอร์ 00H เก็บค่าตัวหารไบต์ต่ำ ส่วนรีจิสเตอร์ 01H ใช้เก็บค่าตัวหารไบต์สูง การหารค่าบอดเรตสามารถเขียนเป็นสมการได้ดังนี้

บอดเรต = 115200 / ค่าตัวหาร 16 บิต

ค่าตัวเลข 115200 มาจากความถี่ของคริสตอลในวงจร UART ภายในเครื่องคอมพิวเตอร์ โดยคริสตอลที่ใช้มีความถี่ 1.8432 MHz วงจรภายใน UART จะหารค่าความถี่นี้ด้วย 16 ทำให้ได้ค่าความถี่ 115200 Hz ออกมา

ค่าตัวหาร 16 บิต = ข้อมูลในรีจิสเตอร์ 00H + (250 * ข้อมูลในรีจิสเตอร์ 01H) ถ้าต้องการบอดเรตเท่ากับ 9600 ค่าตัวหารที่ใช้จะต้องมีค่าเท่ากับ 12 ซึ่งค่านี้จะถูกเขียนลงในรีจิสเตอร์ 00H และเขียนค่า 0 ลงในรีจิสเตอร์ 01H ค่าตัวหารที่ทำให้เกิดค่าบอดเรตสูงสุดที่ 115200 บิตต่อวินาทีคือ ค่า 0001 นั่นคือรีจิสเตอร์ 00H มีค่าเท่ากับ 1 และรีจิสเตอร์ 01H มีค่าเท่ากับ 0

BRK ใช้ควบคุมการหยุดถ่ายทอดข้อมูล

" 1 " สามารถหยุดหรือเบรกได้

" 0 " ไม่มีการหยุดหรือเบรก

PAR2, PAR1, PAR0 ใช้เพื่อกำหนดบิตพาริตี้

" 000 " ไม่ใช้บิตพาริตี้

" 001 " กำหนดพาริตี้คี่

" 011 " กำหนดพาริตี้คู่

" 101 " มาร์ก (Mark)

" 111 " ช่องว่าง (Space)

" 1 " มีบิตปิดท้าย 2 บิต

" 0 " มีบิตปิดท้าย 1 บิต

DAB1, DAB0 ใช้ร่วมกันในการกำหนดจำนวนบิตของข้อมูลที่ต้องการถ่ายทอด

" 00 " จำนวนบิตข้อมูลเท่ากับ 5 บิต

" 01 " จำนวนบิตข้อมูลเท่ กับ 6 บิต

" 10 " จำนวนบิตข้อมูลเท่ากับ 7 บิต

" 11 " จำนวนบิตข้อมูลเท่ากับ 8 บิต

3.3.5 รีจิสเตอร์ตำแหน่ง 04H : รีจิสเตอร์ควบคุมโมเด็ม

มีรายละเอียดของแต่ละบิตดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
0	0	0	LOOP	OUT2	OUT1	RTS	DTR

บิต 5-7 ไม่ได้ใช้งาน อ่านค่าได้เท่ากับ 0

LOOP " 1 " เอ็นเอเบิลการส่งค่ากลับ

" 0 " ดิสเอเบิล

OUT1, OUT2 " 1 " เอ็นเอเบิลการใช้งานภายใน

" 0 " ดิสเอเบิล

RTS ใช้ควบคุมการทำงานของขา RTS (Ready To Send)

" 1 " เอ็นเอเบิล

" 0 " ดิสเอเบิล

DTR ใช้ควบคุมการทำงานของขา DTR (Data Terminal Ready)

" 1 " เอ็นเอเบิล

" 0 " ดิสเอเบิล

3.3.6 รีจิสเตอร์ตำแหน่ง 05H : รีจิสเตอร์แสดงสถานการณ์รับและส่งข้อมูลอนุกรมของ UART

ใช้งานร่วมกับรีจิสเตอร์แสดงโหมดและสถานะของการอินเตอร์รัปต์ (รีจิสเตอร์ตำแหน่ง 02H) เพื่อแสดงสาเหตุของการเกิดอินเตอร์รัปต์ มีรายละเอียดของแต่ละบิตดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
0	TXE	TBE	BREK	FRME	PARE	OVFE	RxRD

TXE (Transmitter Empty)

" 1 " แสดงว่ารีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูลว่าง

" 0 " แสดงว่ายังคงมีข้อมูล 1 ไบต์เก็บอยู่ในรีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูล

TBE (Transmitter Buffer Empty)

" 1 " รีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูลว่าง

" 0 " ยังคงมีข้อมูล 1 ไบต์เก็บอยู่ในรีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูล

BREK (Break)

" 1 " UART ตรวจพบการเบรก

" 0 " ไม่มีการเบรก

FRME (Frame Error)

" 1 " UART ตรวจพบความผิดพลาดด้านเฟรมข้อมูล

" 0 " ไม่พบความผิดพลาดด้านเฟรมข้อมูล

PARE (Parity Error)

" 1 " UART ตรวจพบความผิดพลาดทางพาริตี้

" 0 " ไม่พบความผิดพลาดทางพาริตี้

OVRE (Overrun Error)

" 1 " UART ตรวจพบความผิดพลาดแบบโอเวอร์รัน

" 0 " ไม่พบความผิดพลาดแบบโอเวอร์รัน

RxRD (Received Data Ready)

" 1 " มีการรับข้อมูลเข้ามาเก็บไว้ในบัฟเฟอร์

" 0 " ไม่มีข้อมูล

3.3.7 รีจิสเตอร์ตำแหน่ง 06H : รีจิสเตอร์แสดงสถานะของโมเด็ม

ใช้เพื่อกำหนดสถานะสัญญาณอินพุตของพอร์ตอนุกรม RS-232 ซึ่งได้แก่ สัญญาณ DCD, DSR, CTS และ RI สำหรับการเชื่อมต่อใช้งานแบบอเนกประสงค์ ซึ่งมีรายละเอียดของแต่ละบิตดังต่อไปนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
DCD	RI	DSR	CTS	DDCD	DRI	DDSR	DCTS

DCD ใช้แสดงสภาวะของขา DCD

" 1 " แสดงว่าที่ขา DCD เป็นลอจิก " 1 "

" 0 " แสดงว่าที่ขา DCD เป็นลอจิก " 0 "

RI ใช้แสดงสภาวะของขา RI

" 1 " แสดงว่าที่ขา RI เป็นลอจิก " 1 "

" 0 " แสดงว่าที่ขา RI เป็นลอจิก " 0 "

DSR ใช้แสดงสภาวะของขา DSR

" 1 " แสดงว่าที่ขา DSR เป็นลอจิก " 1 "

" 0 " แสดงว่าที่ขา DSR เป็นลอจิก " 0 "

CTS (Clear To Send) แจ้งการเปลี่ยนแปลงที่เกิดขึ้นกับบิต CTS

" 1 " แสดงว่าบิต CTS เกิดการเปลี่ยนแปลงเมื่อเทียบจากการอ่านค่าครั้งที่แล้ว

" 0 " แสดงว่าไม่มีการเปลี่ยนแปลงเมื่อเทียบจากการอ่านค่าครั้งที่แล้ว

DDSR (Delta Data Set Ready) ใช้แจ้งการเปลี่ยนแปลงที่เกิดขึ้นของบิต DSR

" 1 " แสดงว่าบิต DSR เกิดการเปลี่ยนแปลงเมื่อเทียบจากการอ่านค่าครั้งที่แล้ว

" 0 " แสดงว่าไม่มีการเปลี่ยนแปลงเมื่อเทียบจากการอ่านค่าครั้งที่แล้ว

DRI (Delta Ring Indication) ใช้แจ้งการเปลี่ยนแปลงที่เกิดขึ้นของบิต RI

" 1 " แสดงว่าบิต RI เกิดการเปลี่ยนแปลงเมื่อเทียบจากการอ่านค่าครั้งที่แล้ว

" 0 " แสดงว่าไม่มีการเปลี่ยนแปลงเมื่อเทียบจากการอ่านค่าครั้งที่แล้ว

DDCD (Delta Data Carrier Detect) ใช้แจ้งการเปลี่ยนแปลงที่เกิดขึ้นของบิต DDCD

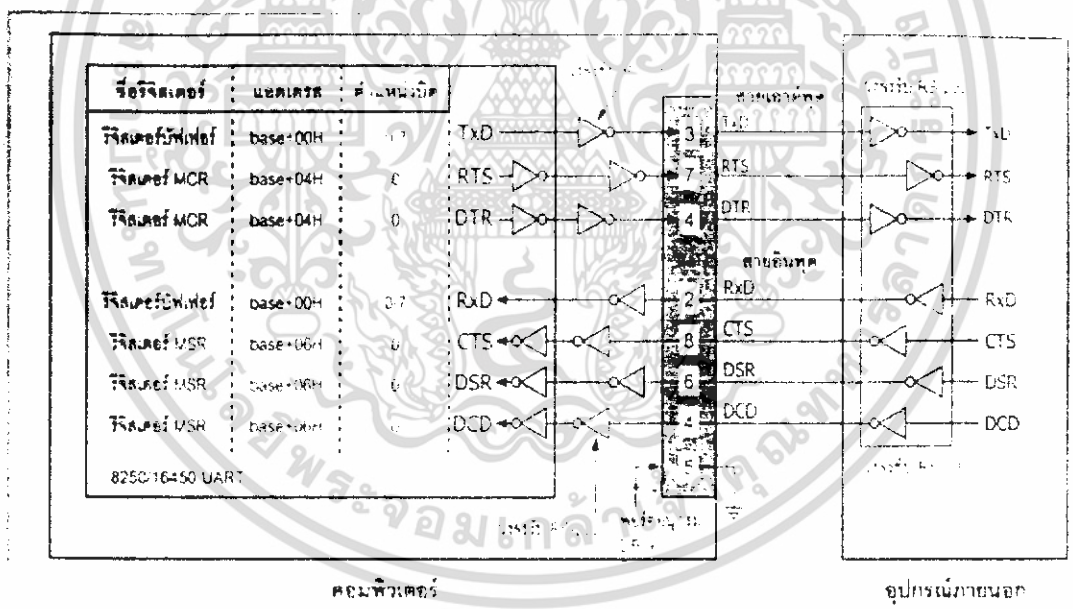
" 1 " แสดงว่าบิต CTS เกิดการเปลี่ยนแปลงเมื่อเทียบจากการอ่านค่าครั้งที่แล้ว

" 0 '' แสดงว่าไม่มีการเปลี่ยนแปลงเมื่อเทียบจากการอ่านค่าครั้งที่แล้ว

DCTS (delta Clear To Send) ใช้แสดงสภาวะของขา CTS

" 1 " แสดงว่าที่ขา CTS เป็นลอจิก " 1 "

" 0 " แสดงว่าที่ขา CTS เป็นลอจิก " 0 "



ภาพที่ 3-6 ไดอะแกรมแสดงโครงสร้างทางฮาร์ดแวร์ของพอร์ตอนุกรม

3.3.8 รีจิสเตอร์ตำแหน่ง 07H : รีจิสเตอร์สำหรับเก็บข้อมูลชั่วคราว

ทำหน้าที่เป็นหน่วยความจำแรมขนาด 1 ไบต์ การอ่านและเขียนข้อมูลที่รีจิสเตอร์ตัวนี้ไม่ส่งผลใด ๆ ต่อการใช้งาน UART

3.3.9 ลักษณะสัญญาณอินพุตและเอาต์พุตของพอร์ต RS-232

สัญญาณเอาต์พุตที่ใช้ควบคุม (RTS และ DTS) และสัญญาณแสดงสถานะอินพุต (CTS, DSR และ DCD) ของพอร์ตอนุกรม RS-232 จะถูกกลับสถานะภายในตัว UART ส่วนสัญญาณข้อมูลทั้งภาคส่งและรับจะไม่ถูกกลับสถานะ UART จะให้ระดับสัญญาณเอาต์พุตออกมาเป็นแบบทีทีแอลเท่านั้น ดังนั้นเมื่อสัญญาณถูกส่งออกมาจาก UART จึงต้องส่งเข้าสู่วงจรขับเพื่อปรับระดับแรงดันให้ได้ระดับสัญญาณเป็นไปตามมาตรฐาน RS-232 ก่อนส่งออกไปยังคอมพิวเตอร์ สำหรับอุปกรณ์ต่อเชื่อมปลายทางก็จะต้องมีวงจรขับในลักษณะนี้เช่นเดียวกัน เพื่อให้ได้ระดับสัญญาณในระดับเดียวกันแต่วงจรขับที่ใช้ทั้งภายในคอมพิวเตอร์และอุปกรณ์ต่อเชื่อมปลายทางนั้นจะถูกกลับสถานะ ดังแสดงในภาพที่ 3-6

3.3.9.1 แอดเดรสของพอร์ตอนุกรม

แอดเดรสพื้นฐานของพอร์ตอนุกรมมี 4 ตำแหน่งดังนี้คือ

COM1 : 3F8H COM3 : 3E8H

COM2 : 2F8H COM4 : 2E8H

ตารางที่ 3-2 แสดงข้อมูลในแอดเดรส 0000 : 0411H ที่ใช้แจ้งจำนวนพอร์ตอนุกรม

บิต 3	บิต 2	บิต 1	จำนวนพอร์ต
0	0	0	ไม่มีพอร์ตอนุกรม
0	0	1	มีพอร์ตอนุกรม 1 พอร์ต
0	1	0	มีพอร์ตอนุกรม 2 พอร์ต
0	1	1	มีพอร์ตอนุกรม 3 พอร์ต
1	0	0	มีพอร์ตอนุกรม 4 พอร์ต

เมื่อเริ่มเปิดเครื่องเพื่อใช้งานคอมพิวเตอร์ ไบออสภายในคอมพิวเตอร์จะทำการตรวจสอบแอดเดรสของพอร์ตอนุกรมทั้งหมด ถ้าไบออสตรวจพบแอดเดรสของพอร์ตอนุกรม ไบออสจะนำแอดเดรสที่ตรวจพบไปเก็บไว้ในหน่วยความจำขนาด 2 ไบต์ สำหรับพอร์ตอนุกรม COM1 จะเก็บไว้ที่แอดเดรส 0000 : 0400H และ 0000 : 0401H ส่วนตำแหน่งอื่น ๆ มีรายละเอียดดังนี้

COM2 = 0000 : 0402H – 0000 : 0403H

COM3 = 0000 : 0404H – 0000 : 0405H

COM4 = 0000 : 0406H – 0000 : 0407H

นอกจากนี้ที่หน่วยความจำแอดเดรส 0000 : 0411H ยังใช้สำหรับแสดงความจำของพอร์ตอนุกรมที่มีใช้อยู่ในคอมพิวเตอร์อีกด้วย โดยมีรายละเอียดดังแสดงในตารางที่ 3-2

3.4 การติดต่อโมดูล LCD ของไมโครคอนโทรลเลอร์ MCS-51 ด้วยภาษา C

โมดูล LCD เป็นอุปกรณ์แสดงผลอย่างหนึ่งที่ได้รับความนิยมในการใช้งานสูงมาก โดยเฉพาะอย่างยิ่งนำมาเชื่อมต่อกับไมโครคอนโทรลเลอร์ สำหรับในบทนี้จะนำเสนอกาติดต่อระหว่างไมโครคอนโทรลเลอร์ MCS-51 กับโมดูล LCD แบบตัวอักษรขนาด 16 ตัวอักษร 2 บรรทัด ด้วยโปรแกรมภาษา C โดยโมดูล LCD ที่ใช้ชิพเบอร์ HD44870 หรือเทียบเท่าเป็นตัวควบคุมการทำงาน

3.4.1 ข้อมูลเบื้องต้นของโมดูล LCD

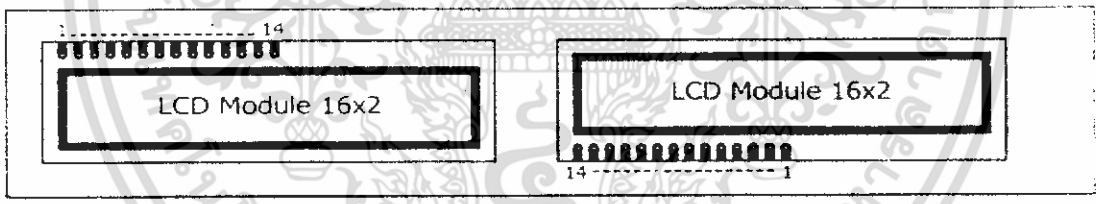
ในรูปที่ 1 แสดงการจัดขาของโมดูล LCD ขนาด 16 ตัวอักษร 2 บรรทัด ซึ่งมี 2 แบบ

Vss (ขา 1) : ต่อกราวด์

Vdd (ขา 2) : ต่อไฟเลี้ยง + 5V

Vo (ขา 3) : เป็นขาอินพุตรับแรงดันเพื่อปรับความเข้มของการแสดงผล

RS (ขา 4) : เป็นขาอินพุตใช้แยกชนิดของข้อมูลที่ประมวลผลว่าเป็นคำสั่งหรือเป็นข้อมูล โดยถ้าขานี้เป็น "0"ข้อมูลที่ส่งมาจะเป็นคำสั่ง แต่ถ้าเป็น "1" ข้อมูลที่ส่งมาจะเป็นข้อมูลแสดงผล



ภาพที่ 3-7 แสดงการจัดขาของโมดูล LCD ขนาด 16 ตัวอักษร 2 บรรทัดในแบบต่างๆ

R/W (ขา 5) : เป็นขาที่ใช้เลือกการอ่านข้อมูลกับโมดูล LCD ถ้าเป็น "0" เป็นการกำเนิดให้เขียนข้อมูล แต่ถ้าเป็น "1" จะเป็นการอ่านข้อมูล

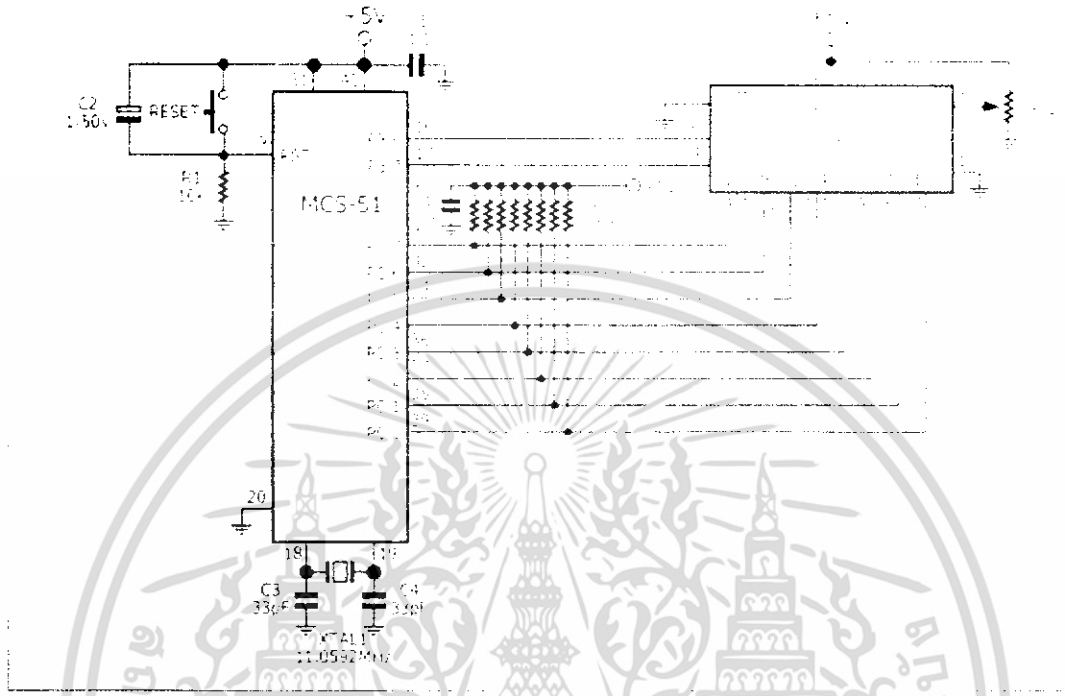
E (ขา 6) : เป็นขาสำหรับรับสัญญาณพัลส์เอ็นเอเบิลโมดูล LCD ให้ทำงาน

D0-D7 (ขา -14) : เป็นขาข้อมูลระหว่าง LCD กับอุปกรณ์ภายนอกขนาด 8 บิต

การจัดกาแอดเดรสของหน่วยความจำแสดงผลของโมดูล LCD 16 ตัวอักษร 2 บรรทัดดังนี้

0x00	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08	0x09	0x0A	0x0B	0x0C	0x0D	0x0E	0x0
M	i	c	r	o	c	o	n	t	r	o	l	l	e	r	*
i	n		t	-	p	r	o	g	r	a	m	m	i	n	g

นั่นคือหากต้องการให้แสดงข้อความที่บรรทัดบน จะต้องเขียนข้อมูลที่ต้องการแสดงผลมายังแอดเดรส 0x40-0x4F



ภาพที่ 3-8 วงจรเชื่อมต่อไมโครคอนโทรลเลอร์ MCS-51 กับโมดูล LCD 16 ตัวอักษร 2 บรรทัด

3.4.2 คำสั่งควบคุมโมดูล LCD ที่สำคัญ

3.4.2.1 คำสั่งเคลียร์จอแสดงผล

มีข้อมูลคำสั่งเป็น 0x01 เป็นคำสั่งเขียนช่องว่างหรือ space เข้าไปในหน่วยความจำสำหรับแสดงตำแหน่งสำหรับแสดงผลหรือ DDRAM ภายในโมดูล LCD เมื่อมีการเคลียร์จอแสดงผลแล้ว จะกำหนดให้เคอร์เซอร์ (cursor : สัญลักษณ์พิเศษที่ใช้ในการแสดงตัวอักษรของโมดูล LCD บางครั้งมีลักษณะเป็นรูปสี่เหลี่ยมผืนผ้าขนาด 5x7 จุดหรือเป็นขีด สามารถสั่งให้กะพริบหรือไม่ก็ได้) กลับไปอยู่ที่ตำแหน่งซ้ายมือสุดของจอแสดงผล

3.4.2.2 คำสั่ง Reture Home

มีค่าของข้อมูลเท่ากับ 0x02 หรือ 0x03 ก็ได้ (แนะนำให้ใช้ 0x02) เป็นการกำหนดให้เคอร์-เซอร์ ไปยังตำแหน่งซ้ายสุดของจอแสดงผลจะไม่มีการเปลี่ยนแปลง

3.4.2.3 คำสั่งเลือกโหมดการป้อนข้อมูล (Entry mode set)

มีรูปแบบคำสั่งดังนี้

บิต 7	บิต6	บิต 5	บิต4	บิต 3	บิต 2	บิต1	บิต 0
0	0	0	0	0	0	I/D	S

บิต I / D ใช้ในการกำหนดว่า เมื่อเขียนหรืออ่านข้อมูลแล้ว ทำให้แอดเดรสของหน่วยความจำแสดงผล (DDRAM) เพิ่มหรือลดลง

"0" แอดเดรสลง 1 แอดเดรส

"1" แอดเดรสเพิ่ม 1 แอดเดรส

บิต S ใช้กำหนดลักษณะการแสดงผล

"0" เมื่อเกิดตัวอักษรใหม่ เคอร์เซอร์เลื่อนไปทางขวามือ

"1" เมื่อเกิดตัวอักษรใหม่ เคอร์เซอร์อยู่ที่เดิม แต่ตัวอักษรเลื่อนไปทางซ้าย ที่นิยมใช้มากที่สุดคือ ข้อมูลคำสั่ง 0x06 หมายถึง กำหนดให้เมื่อเกิดข้อมูลใหม่ เคอร์เซอร์จะเลื่อนไปทางขวามือ และแอดเดรสของหน่วยความจำสำหรับแสดงผลจะเพิ่มขึ้น

3.4.2.4 คำสั่งควบคุมการแสดงผล

มีรูปแบบคำสั่งดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
0	0	0	0	1	D	C	B

บิต D ใช้ควบคุมการเปิดปิดจอแสดงผล ถ้าเป็น "0" เป็นการปิดจอแสดงผล ถ้าเป็น "1" เป็นการเปิดจอแสดงผล

บิต C ใช้ควบคุมการแสดงตัวบนจอแสดงผล ถ้าเป็น "0" เป็นการปิดตัวเคอร์เซอร์ ถ้าเป็น "1" เป็นการแสดงตัวเคอร์เซอร์

บิต B ใช้ควบคุมการกระพริบของเคอร์เซอร์ ถ้าต้องการให้เคอร์เซอร์กระพริบ ต้องกำหนดให้เป็น "1"

คำสั่งที่ใช้บ่อยคือ 0x0C เป็นการสั่งให้เปิดจแสดงผล แต่แสดงเคอร์เซอร์ และ 0x0F เป็นการสั่งให้เปิดหน้าจอแสดงผล แสดงเคอร์เซอร์

3.4.2.5 คำสั่งควบคุมการเลื่อนเคอร์เซอร์และตัวอักษร

มีรูปแบบดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
0	0	0	1	S/C	R/L	*	*

การเลื่อนเคอร์เซอร์และตัวอักษรขึ้นอยู่กับการกำหนดบิตดังนี้

S/C	R/L	ลักษณะการเลื่อน	ข้อมูลคำสั่ง
0	0	เลื่อนเคอร์เซอร์ไปทางซ้าย	0x10-0x13
0	1	เลื่อนเคอร์เซอร์ไปทางขวา	0x14-0x17
1	0	เลื่อนตัวอักษรใหม่ไปทางซ้าย	0x18-0x1B
1	1	เลื่อนตัวอักษรใหม่ไปทางซ้าย	0x1C-0x1F

3.4.2.6 คำสั่งกำหนดฟังชันก์การทำงาน

เป็นคำสั่งที่มีความสำคัญมากที่สุดอีกคำสั่งหนึ่ง มีรูปแบบของข้อมูลคำสั่งดังนี้

บิต 7	บิต 6	บิต 5	บิต 4	บิต 3	บิต 2	บิต 1	บิต 0
0	0	1	DL	N	F	*	*

บิต DL ใช้ในกำหนดจำนวนบิตในการติดต่อกับโมดูล LCD

"0" กำหนดให้ทำงานในโหมด 4 บิต

"1" กำหนดให้ทำงานในโหมด 8 บิต

บิต N ใช้ในกำหนดจำนวนบรรทัดที่ต้องการให้แสดงผล

"0" แสดงผล 1 บรรทัด

"1" แสดงผล 2 บรรทัดหรือมากกว่า

แต่สำหรับโมดูล LCD ขนาด 16 ตัวอักษร 1 บรรทัดที่มีจำหน่ายในประเทศไทย ต้องกำหนดให้บิตนี้เป็น "1" เนื่องจากหน่วยความจำสำหรับเก็บข้อมูลเพื่อแสดงผลไม่ได้อยู่ต่อเนื่องกัน กล่าวคือ แบ่งเป็น 2 ช่วงคือ ที่แอดเดรส 0x00-0x07 และ 0x40-04x47 โดยแอดเดรส 0x40 จะเป็นแอดเดรสเริ่มต้นของหน่วยความจำสำหรับการแสดงผลในบรรทัดที่สองของโมดูล LCD ที่มีมากกว่า 1 บรรทัด จึงทำให้เมื่อต้องการติดต่อในโมดูล LCD 1 บรรทัดจึงจำเป็นต้องกำหนดให้เป็น "1"

บิต F ใช้เลือกความละเอียดของตัวอักษรในการแสดงผล

“0”แสดงผลแบบ 5x7จุด

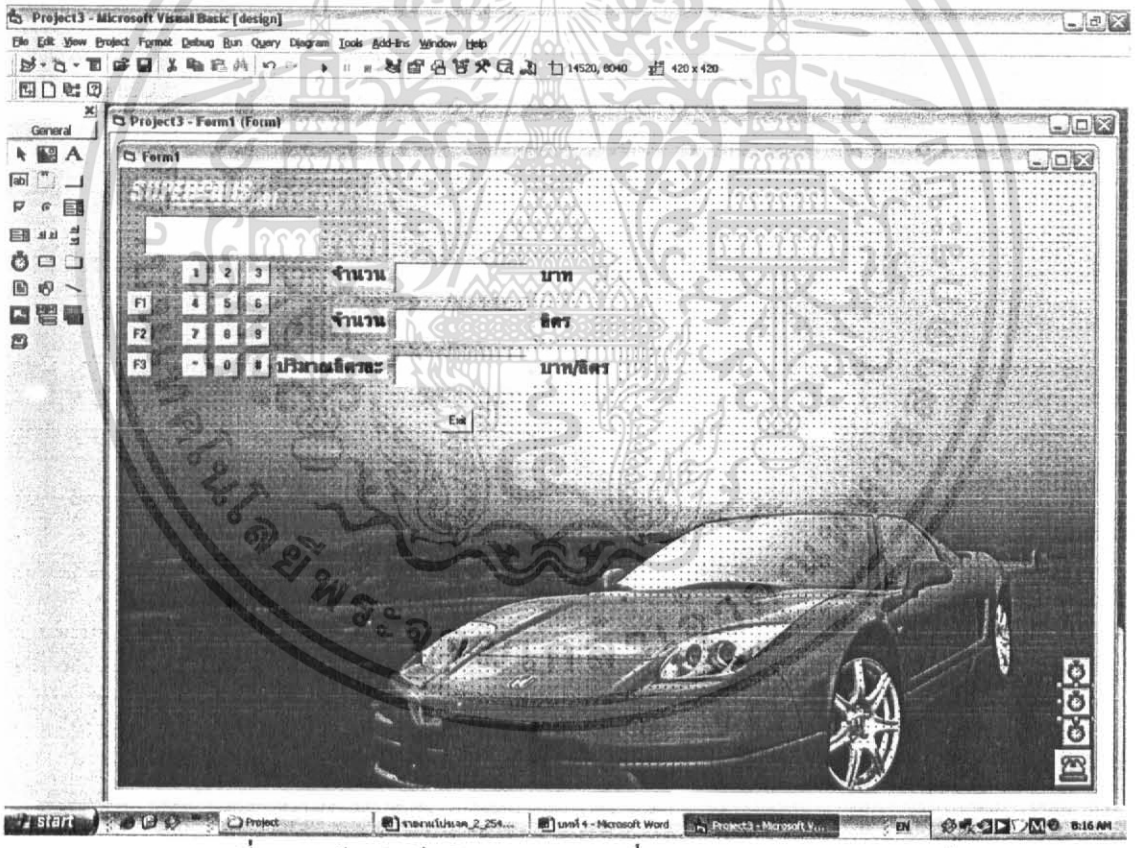
“1”แสดงผลแบบ 5x10จุด

บทที่ 4
การออกแบบตัวควบคุมและอุปกรณ์ทำงานร่วม

4.1 กล่าวนำ

บทนี้จะเป็นการออกแบบทางด้านฮาร์ดแวร์ต่าง ๆ ว่าด้วยโครงสร้างของระบบและวงจรทางไฟฟ้าที่นำมาใช้ เพื่อนำมาประกอบกันเข้าเป็นตัวชิ้นงาน ซึ่งยังรวมไปถึงการกำหนด และการเลือกใช้ตัวคอนโทรลต่าง ๆ ในการเขียนโปรแกรมวิชวลเบสิก พร้อมทั้งการกำหนดค่าให้แก่พร็อพเพอร์ตี้ที่ได้เลือกใช้อีกด้วย

4.2 การสร้างฟอร์ม วิชวล(Visual) เพื่อแสดงผลทางคอมพิวเตอร์



ภาพที่ 4.1 สร้างฟอร์ม วิชวล(Visual) เพื่อแสดงผลทางคอมพิวเตอร์

ตารางที่ 4-1 กำหนดค่าพร็อพเพอร์ตี้ให้กับคอนโทรลเลอร์ต่างๆ

คอนโทรล	พร็อพเพอร์ตี้	ค่าที่กำหนด
From	Name	From1
	Caption	From1
Text1	Name	Text1
	Caption	
Text2	Name	Text2
	Caption	
Text3	Name	Text3
	Caption	
Text4	Name	Text4
	Caption	
Label1	Name	Label1
	Caption	จำนวน
Label2	Name	Label2
	Caption	จำนวน
Label3	Name	Label3
	Caption	ปริมาณลิตรละ
Label4	Name	Label4
	Caption	บาท
Label5	Name	Label5
	Caption	ลิตร
Label6	Name	Label6
	Caption	บาท/ลิตร
Command1	Name	Command1
	Caption	1
Command2	Name	Command2
	Caption	2
Command3	Name	Command3
	Caption	3

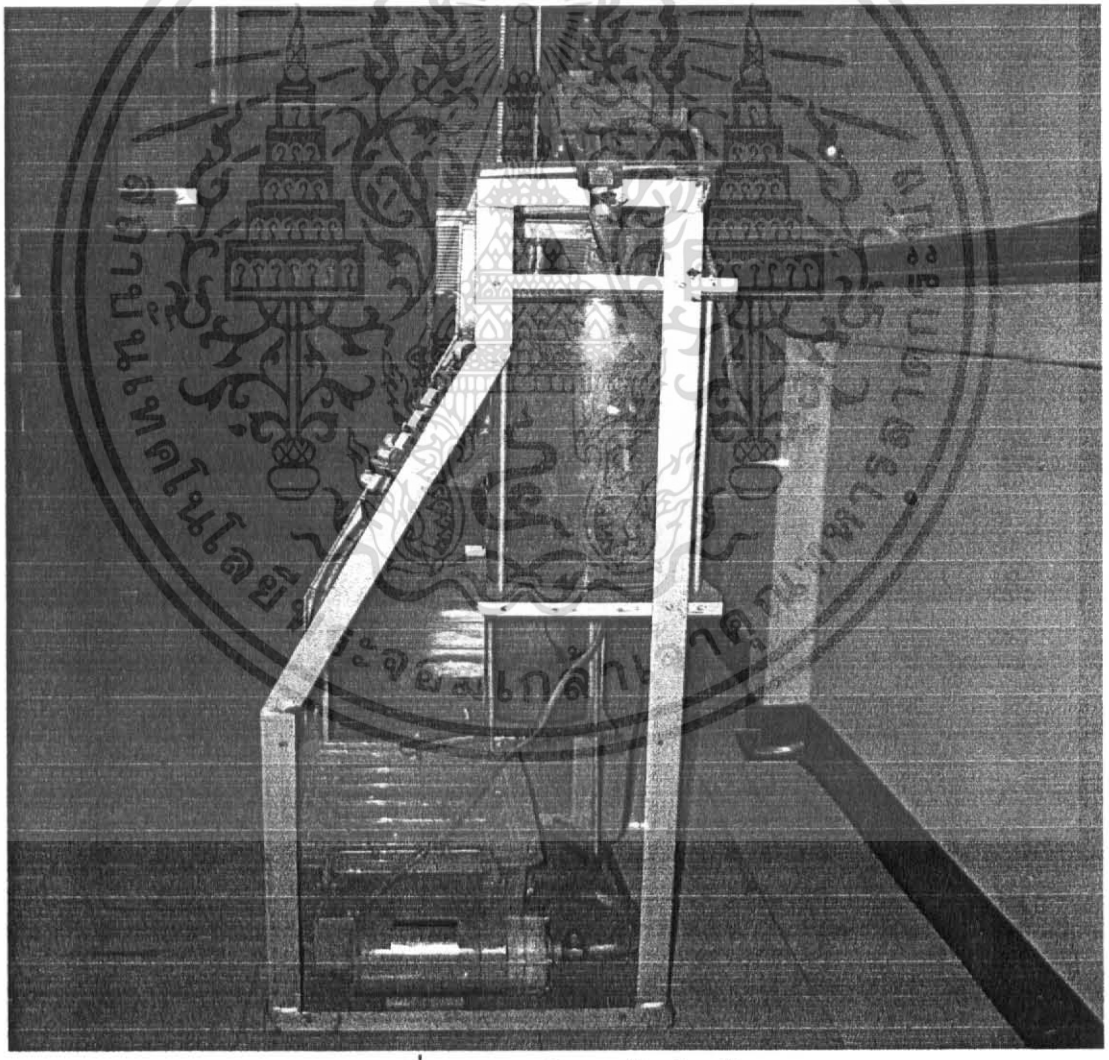
ตารางที่ 4-1 กำหนดค่าพร็อพเพอร์ตี้ให้กับคอนโทรลเลอร์ต่างๆ (ต่อ)

Command4	Name	Command4
	Caption	4
Command5	Name	Command5
	Caption	5
Command6	Name	Command6
	Caption	6
Command7	Name	Command7
	Caption	7
Command8	Name	Command8
	Caption	8
Command9	Name	Command9
	Caption	9
Command10	Name	Command10
	Caption	0
Command11	Name	Command11
	Caption	*
Command12	Name	Command12
	Caption	#
Command13	Name	Command13
	Caption	F1
Command14	Name	Command14
	Caption	F2
Command15	Name	Command15
	Caption	F3
Command16	Name	Command16
	Enable	Exit
Time1	Name	Time1
	Enable	True
	Interval	50

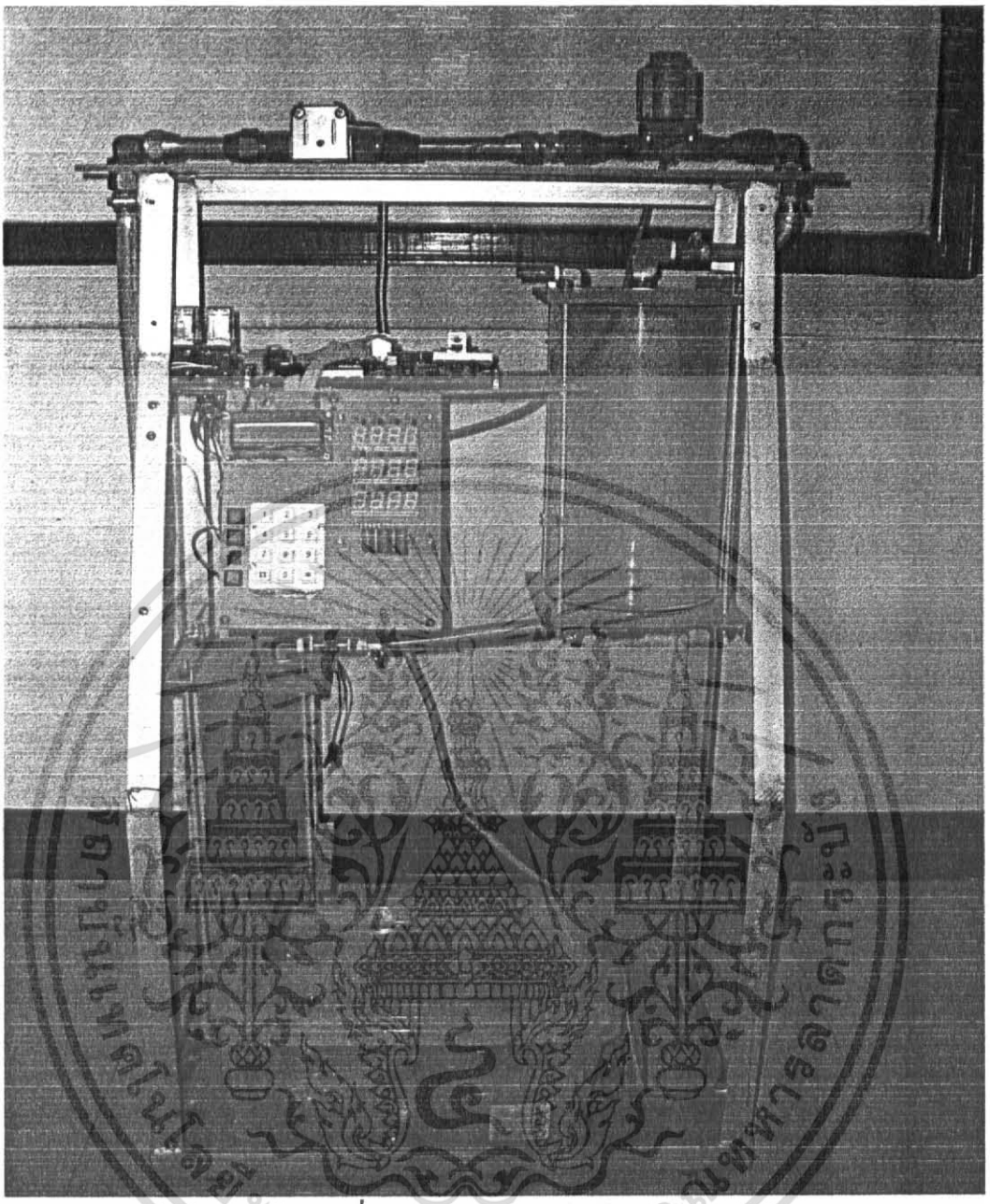
ตารางที่ 4-1 กำหนดค่าพร็อพเพอร์ตี้ให้กับคอนโทรลเลอร์ต่างๆ (ต่อ)

Time1	Name	Time1
	Enable	True
	Interval	50
Time1	Name	Time1
	Enable	True
	Interval	50
MSComm1	Name	MSComm1

4.3 การออกแบบทางด้านโครงสร้าง



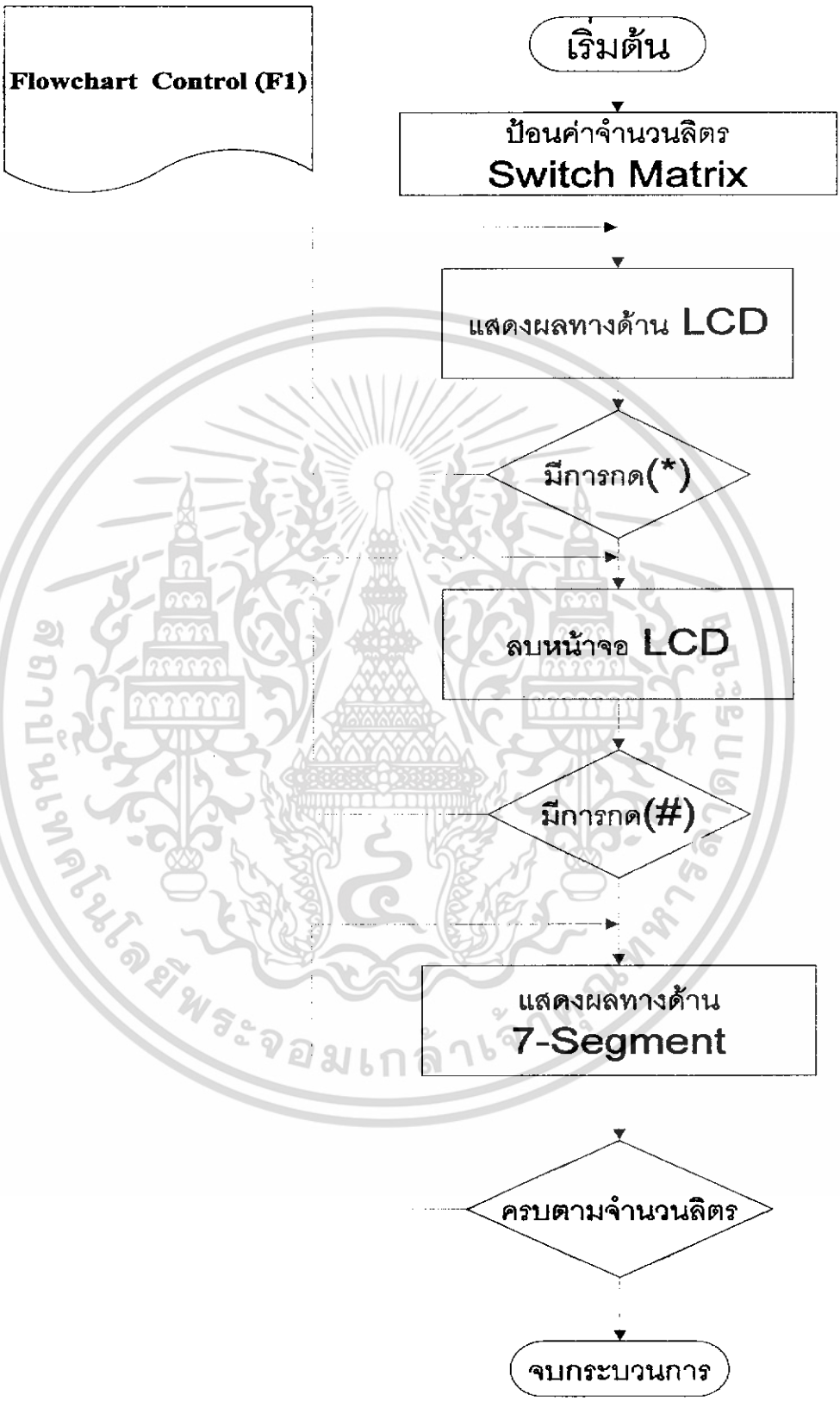
ภาพที่ 4.2 แสดงโครงสร้างด้านข้าง



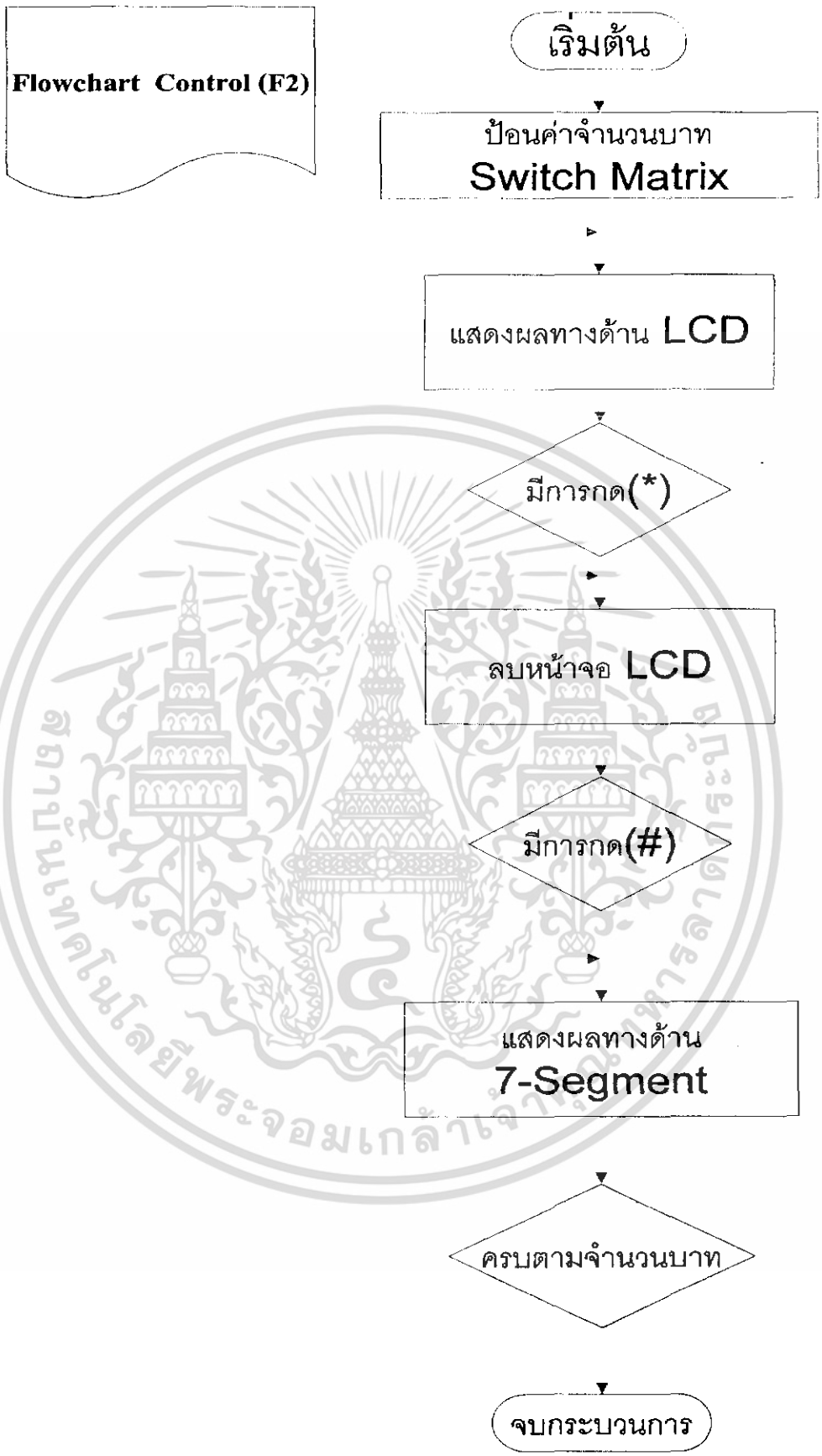
ภาพที่ 4.3 แสดงโครงสร้างด้านหน้า

4.4 โปรแกรมประกอบการควบคุมการทำงาน

4.4.1 Turbo-C Flowchart



ภาพที่ 4.4 Flowchart Control (F1)



ภาพที่ 4.5 Flowchart Control (F2)

4.4.2 C - Program

```
/*-------------------------------------------------------------------------------------------------
--*/
// Program      : Fuel and Fluid Control System
// Description  : Flow Control
// Filename     : Fuel and Fluid Control System
// C compiler   : RIDE 51 V6.1
// Present by Engineer Mr.Suttichai. Sunthalunai
/*-------------------------------------------------------------------------------------------------
--*/
#include<reg51rx.h>
#include<stdio.h>
#include<number_text.h>
#include<scankey.h>
#include<multiplex.h>
//#include<function f1.h>
//#include<i2c_1.h>
//#define PCF8574A_ID 0x70
code unsigned char bill_text[10]=" ABC  ";
sbit relay_1  = P2^6;
sbit relay_2  = P2^7;
sbit T0_P     = P3^4;
sbit e        = P3^3;
sbit rs       = P3^5;
sbit switch_1 = P3^6;
sbit switch_2 = P3^7;
sbit switch_3 = P1^7;

unsigned int  bill_a;
unsigned int  bill_b;
unsigned int  subin;
unsigned int  subin_bill;
unsigned int  number_mix;

unsigned char dat     = 0;
unsigned char dat_A   = 0;
unsigned char count   = 2;
unsigned char count_1 = 0;
unsigned char j = 0,p = 0,button = 0;
unsigned char rr    = 1;
unsigned char tt    = 1;

/*************************************************************************
**/
/************************** Function Delay time
*****************************/
/*************************************************************************
**/
void delay(int tick)
```

```
{
        int i,j;
        for(i=0;i<tick;i++)
        for(j=0;j<250;j++);
}

/*************************************************************************
**/
/************************ Function check push key
*************************/
/*************************************************************************
**/

void scan_sw_f1()
{
       if(T0_P==0)
       {
                if(bill_a<subin)
                {
                        value++;
                        bill_a++;
                        value_1++;
                        if(value_1 == subin_bill)
                        {
                          value_1 = 0;
                          value_a = value_a + 1;
                        }

                }
       }

}

/*************************************************************************
**/
/************************ Function check push key
*************************/
/*************************************************************************
**/

void scan_sw_f2()
{
        f(T0_P==0)
        {
                bill_b++;
                if(bill_b == 4)
                {
                        bill_b = 0;
                        if(bill_a<subin)
```

```
                {
                        value_a++;
                        bill_a++;
                        value_1++;
                        if(value_1 == 1)
                        {
                              value_1    = 0;
                              value      = value + subin_bill;
                         }
                }

            }

        }
}


/*************************************************************************
**/
/************************** Function Send Command
*************************/
/*************************************************************************
**/

void lcd_command(unsigned char com)
{
        rs = 0;
        e = 1;
        P0 = com;
        delay(10);
        e = 0;
        delay(10);
}

/*************************************************************/
/************************ Function Send Character ************************/
/*************************************************************/

void lcd_text(unsigned char text)
{
        rs = 1;
        e = 1;
        P0 = text;
        delay(10);
        e = 0;
        delay(10);
}
```

```
/*****************************************************************/
/****************** Function Initial LCD display *****************/
/*****************************************************************/

void lcd_init()
{
        delay(500);                     // Delay for initial LCD
        lcd_command(0x38);              // ON display ,8 bit display ,5*7 dot
        lcd_command(0x0C);              // Display ON , none cursor
        lcd_command(0x01);              // Clear screen
}


/*****************************************************************/
/***************************Function main *************************/
/*****************************************************************/
void main(void)
{
        unsigned char count_AA  = 1;
        unsigned char count_BB  = 0;
        unsigned char count_B   = 0;
        unsigned char count_A   = 1;
        unsigned int  number_c1 = 0;
        unsigned int  number_c2 = 0;
        unsigned int  number_c3 = 0;
        unsigned int  number_c4 = 0;
        unsigned char pike      = 0;
        unsigned char key       = 0;
        unsigned char key_aa    = 0;
  //unsigned char ropp     = 0;

//*****************************************************************//
        unsigned char dat_BB = 0;   // For keep data TX/RX
        TMOD = 0x21;                // Timer1 Mode2(8 bit auto reload) for serial
port
        SCON = 0x50;                // Mode serial port TX/RX data
        TH1  = 0xFA;                // Set 9600 bps Timer1 default
        TL1  = 0xFA;
        RI   = 0;                   // Clear RI flag
        TI   = 0;                   // Clear TI flag
        TR1  = 1;                   // Start Timer1
//*****************************************************************//


   switch_1 = 1;
   switch_2 = 1;

   while(1)
  {
     if(switch_1 == 0)
```

```
    pike = 1;
    if(switch_2 == 0)
    pike = 2;


//-------------------------------------------------------------------------------------------------------
--//
    if(switch_3 == 0)
    {
        while(count_B)
        {
            key_aa = scankey();
            if(key_aa != 0xff)
            {
                count_1++;

                if(count_1=1)
                {
                    number_text_e(key_aa);
                    count_B = 0;
                }
                if(count_1=2)
                {
                number_text_f(key_aa);
                count_B = 0;
                }
                //if(p=3)
                //number_text_g(key_aa);
                //if(p=4)
                //number_text_h(key_aa);
            }

        }
            number_5 = number_5 * 10;
            number_6 = number_6 * 1;
            //number_7 = number_7 * 10;
            //number_8 = number_8 * 1;
            number_mix = number_5 + number_6;  // + number_7 + number_8;
            subin_bill = 2;
    }
//-------------------------------------------------------------------------------------------------------
--//
                    switch(pike)
    {
        case 0 : break;
//*****************************************************************//
//*************************FunctionF1*****************************//
//*****************************************************************//
        case 1 :/*function f1();*/
        while(1)
```

```
{
    while(count_A && count_AA)
    {
        lcd_init();
            //lcd_command(0x01);
        lcd_command(0x80);
        while(count_A)
        {
            dat = scankey();
            if(dat != 0xff)
            {
                count++;
                if(dat == 0x0b)
                {
                    count_A = 0 ;
                    lcd_init();
                    lcd_command(0x01);
                    lcd_command(0x02);
                    lcd_command(0x80);
                }

                    if(count==16)
                {
                    count = 0;
                    lcd_command(0x02);
                    lcd_command(0xC0);
                }
                    if(count>=32)
                {
                    count = 0;
                    lcd_command(0x01);
                    lcd_command(0x02);
                    lcd_command(0x80);
                }
                    if(count == 3)
                {
                    count_BB = 2;
                    number_text_a(dat);
                }
                    if(count == 4)
                {
                    count_BB = 3;
                    number_text_b(dat);
                }
                    if(count == 5)
                {
                    count_BB = 4;
                    number_text_c(dat);
                }
```

```
                    if(count == 6)
                {
                    count_BB = 5;
                    number_text_d(dat);
                }

                    dat = dat | 0x30;
                    lcd_text(dat);
                }
                                    // stop if()
            }
                                // stop whiel()
        }

            pike = 3;
            key  = scankey();
            if(key == 0x0a)
        {
            relay_1 = 0;
            relay_2 = 0;
            lcd_init();
            lcd_command(0x01);
            lcd_command(0x02);      // Initial origin
            lcd_command(0x80);      // Set LCD address 00H

            switch(count_BB)
            {
                case 0 : break;
                case 1 : break;
                case 2 : break;
                case 3 : number_1  = number_1 * 10;
                         number_2  = number_2 * 1;
                         number_c2 = number_1 + number_2 ;
                while(rr && tt)
                {
                    subin  = number_c2 - 11;
                       if(bill_a == 5)
                       {
                           SBUF = bill_text[1];

                           while(~TI);

                           TI  = 0;

                       }


                           if(bill_a == subin)
                    {
```

```
                    SBUF = bill_text[2];

                    while(~TI);
                     TI  = 0;

                }
                    scan_sw_f1() ;
                    multiplex();
                   //if(switch_1 = 0)
                   //{
                   //tt =0;
                   //}
            }
              break;
              case 4 : number_1  = number_1 * 100;
                       number_2  = number_2 * 10;
                       number_3  = number_3 * 1;
                       number_c3 = number_1+
number_2+number_3 ;
                        while(rr && tt)
            {
              subin  = number_c3 -11;
              if(bill_a == 5)
              {
                 SBUF = bill_text[1];
                        while(~TI);

                 TI  = 0;
                 }


                 if(bill_a == subin)
              {
                 SBUF = bill_text[2];
                        while(~TI);
                 TI  = 0;
                  }

                 scan_sw_f1() ;
                 multiplex();
                 //if(switch_1 = 0)
                 //{
                 //          tt =0;
                 //}

            }
                 break;
                 case 5 : number_1  = number_1 * 1000;
                          number_2  = number_2 * 100;
                          number_3  = number_3 * 10;
```

```
                                        number_4  = number_4 * 1;
                                        number_c4 = number_1 + number_2
+ number_3 + number_4 ;
                                    while(rr && tt)
                                {
                                    subin  = number_c4 -11;
                                    if(bill_a ==1)
                                    {
                                        SBUF = bill_text[1];
                                        while(~TI);
                                        TI  = 0;
                                    }

                                        if(bill_a == subin)
                                    {
                                        SBUF = bill_text[2];
                                        while(~TI);
                                        TI  = 0;
                                                                    }
                                        scan_sw_f1();
                                        multiplex();

                                    }
                                     break;
                                    default : break;
                                }

                            }  //stop While(count_A && count_AA) fungtion 1

                    }       //stop While(1) fungtion 1
                    break;
```

```
//*********************************************************************//
//*****************************Stop F1*******************************//
//*********************************************************************//
                    case 2 : while(1)
                {
                    while(count_A && count_AA)
                    {
                       lcd_init();
                       cd_command(0x01);
```

```
lcd_command(0x80);

while(count_A)
{
      dat = scankey();
      if(dat != 0xff)
      {
          count++;
          if(dat == 0x0b)
          {
                count_A = 0 ;
                lcd_init();
                lcd_command(0x01);
                lcd_command(0x02);
                lcd_command(0x80);
                  }
                 if(count==16)
                   {
                 count = 0;
                 lcd_command(0x02);
                        lcd_command(0xC0);
          }
                 if(count>=32)
                   {
                count = 0;
                lcd_command(0x01);
                lcd_command(0x02);
                lcd_command(0x80);
                  }

                if(count == 3)
          {
                count_BB = 2;
                number_text_a(dat);
           }

                if(count == 4)
          {
                count_BB = 3;
                number_text_b(dat);
          }

                if(count == 5)
          {
                count_BB = 4;
                number_text_c(dat);
          }

                if(count == 6)
          {
```

```
                    count_BB = 5;
                    number_text_d(dat);
                }

                    dat = dat | 0x30;
                    lcd_text(dat);
            }
                    // stop whiel()
        }

        pike =3;
        key  = scankey();
        if(key == 0x0a)
        {
        lcd_init();
        lcd_command(0x01);
        lcd_command(0x02);        // Initial origin
        lcd_command(0x80);        // Set LCD address 00H

        switch(count_BB)
        {
            case 0 : break;
            case 1 : break;
            case 2 : break;
            case 3 : number_1  = number_1 * 10;
                     number_2  = number_2 * 1;
                     number_c2 = number_1 + number_2 ;
            while(1)
            {
              subin  = number_c2 - 11;
              scan_sw_f2() ;
              multiplex();
            }
              break;
              case 4 : number_1  = number_1 * 100;
                       number_2  = number_2 * 10;
                       number_3  = number_3 * 1;
                       number_c3 = number_1 + number_2
+ number_3 ;
              while(1)
            {
              subin  = number_c3 -11;
              scan_sw_f2() ;
              multiplex();
            }
              break;
              case 5 : number_1  = number_1 * 1000;
                       number_2  = number_2 * 100;
                       number_3  = number_3 * 10;
```

```
                                        number_4  = number_4 * 1;
                                        number_c4 = number_1 + number_2
+ number_3 + number_4 ;
                                    while(1)
                                {
                                    subin  = number_c4 -11;
                                    scan_sw_f2() ;
                                    multiplex();
                                }
                                    break;
                                    default : break;
                            }

                        }
                    }
                        break;

//*****************************************************************//
//*************************Stop F2*********************************//
//*****************************************************************//

            }
        }
    }
}
```

```
/*-------------------------------------------------------------------------------------------
--*/
// Program      : Fuel and Fluid Control System
// Description  : Flow Control
// Filename     : Switch matrix.h
// C compiler   : RIDE 51 V6.1
// Present by Engineer Mr.Suttichai. Sunthalunai
/*-------------------------------------------------------------------------------------------
--*/
/*****************************************************************/
//*****************************************************************
*/
sbit c1 = P1^6;          // Bit Column
sbit c2 = P1^5;          // Bit Column2
sbit c3 = P1^4;          // Bit Column3
sbit r1 = P1^3;          // Bit Row1
sbit r2 = P1^2;          // Bit Row2
sbit r3 = P1^1;          // Bit Row3
sbit r4 = P1^0;

/*****************************************************************
**/
/********************* Function Delay_ab time
*****************************/
/*****************************************************************
**/

void delay_db(int time)
{
        do                              // do-while loop for delay
        {
         time--;                        // Decrease counter
        }while(time>0);                 // If time>0 work in block
}


/*****************************************************************
**/
/************************ Function Scan keypad 4x3
**************************/
/*****************************************************************
**/

unsigned char scankey(void)             // Return value key
{
        unsigned char ret = 0xFF;       // Initial value ret = 0xFF
        c1 = 0;                         // Scan column1
        if(r1==0)                       // Check push key 1
        {
    delay_db(30000);                    // Delay debounce
```

```
            ret = 0x01;             // Return value = 0x01
        }
        if(r2==0)                   // Check push key 4
        {
            delay_db(30000);        // Delay debounce
            ret = 0x04;             // Return value = 0x04
        }
        if(r3==0)                   // Check push key 7
        {
            delay_db(30000);        // Delay debounce
            ret = 0x07;             // Return value = 0x07
        }
        if(r4==0)                   // Check push key 7
        {
            delay_db(30000);
            ret = 0x0a;
        }
c1 = 1;
        c2 = 0;                     // Scan Column2
        if(r1==0)                   // Check push key 2
        {
            delay_db(30000);        // Delay debounce

            ret = 0x02;             // Return value = 0x02
        }
        if(r2==0)                   // Check push key 5
        {
            delay_db(30000);        // Delay debounce
            ret = 0x05;             // Return value = 0x05
        }
        if(r3==0)                   // Check push key 8
        {
            delay_db(30000);        // Delay debounce
            ret = 0x08;             // Return value = 0x08
        }
        if(r4==0)                   // Check push key 0
        {
            delay_db(30000);        // Delay debounce
            ret = 0x00;             // Return value = 0x00
        }
        c2 = 1;                     // Stop check Column2

        c3 = 0;                     // Scan Column3
        if(r1==0)                   // Check push key 3
        {
            delay_db(30000);        // Delay debounce
            ret = 0x03;             // Return value = 0x03
        }
        if(r2==0)                   // Check push key 6
        {
```

```
            delay_db(30000);        // Delay debounce
            ret = 0x06;             // Return value = 0x06
        }
        if(r3==0)                   // Check push key 9
        {
            delay_db(30000);        // Delay debounce
            ret = 0x09;             // Return value = 0x09
        }
        if(r4==0)                   // Check push key 9
        {
            delay_db(30000);        // Delay debounce
            ret = 0x0b;             // Return value = 0x09
        }

        c3 = 1;                     // Stop check Column3
        return(ret);                // Return key value
}
```

```
/*-------------------------------------------------------------------------------------------
--*/
// Program      : Fuel and Fluid Control System
// Description  : Flow Control
// Filename     : multiplex.h
// C compiler   : RIDE 51 V6.1
// Present by Engineer Mr.Suttichai. Sunthalunai
/*-------------------------------------------------------------------------------------------
--*/

code unsigned int display[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};

unsigned int  value   = 0;

unsigned int  value_1 = 0;

unsigned int  value_a = 0;

unsigned int sega_1   = 0;

unsigned int sega_2   = 0;

unsigned int sega_3   = 0;

unsigned int sega_4   = 0;

unsigned int segb_1   = 0;

unsigned int segb_2   = 0;

unsigned int segb_3   = 0;
```

```
unsigned int segb_4  = 0;
unsigned int segc_1  = 0;
unsigned int segc_2  = 0;
unsigned int segc_3  = 0;
unsigned int segc_4  = 0;

/*****************************************************************************/
/********************* Function Delay_scan time ******************************/
/*****************************************************************************/

void delay_scan(int tick)
{
        int i,j;                                // For keep counter loop
        for(i=0;i<tick;i++)                     // Loop delay
        for(j=0;j<150;j++);
}

/*****************************************************************************/
/********************* Function Multiplex 7-segment display ******************/
/*****************************************************************************/
void multiplex()
{
        sega_4  = value%10;
        sega_3  = value/10;
        sega_2  = value/100;
        sega_1  = value/1000;


                segb_4 = value_a%10;
                segb_3 = value_a/10;
                segb_2 = value_a/100;
                segb_1 = value_a/1000;
```

```
                segc_4  = value%10;
                segc_3  = value/10;
                segc_2  = value/100;
                segc_1  = value/1000;
P2 = 0x03;
P0 = display[sega_4];
delay_scan(10);

        P2=0x07;
        P0 = display[segb_4];
        delay_scan(10);

                P2 = 0x2A;
                P0 = display[segc_4];
                delay_scan(10);

                        P2 = 0x02;
                        P0 = display[sega_3%10];
                        delay_scan(10);

                                P2 = 0x06;
                                P0 = display[segb_3%10];
                                delay_scan(10);
                        P2 = 0x1A;
                        P0 = display[segc_3%10];
                        delay_scan(10);

        P2 = 0x01;
        P0 = display[sega_2%10];
        delay_scan(10);
```

```
                P2 = 0x05;
                P0 = display[segb_2%10];
                delay_scan(10);


                P2 = 0x09;
                P0 = display[segc_2%10];
                delay_scan(10);

                        P2 = 0x00;
                        P0 = display[sega_1%10];
                        delay_scan(10);

                                P2 = 0x04;
                                P0 = display[segb_1%10];
                                delay_scan(10);

                        P2 = 0x08;
                        P0 = display[segc_1%10];
                        delay_scan(10);

}

/*-------------------------------------------------------------------------------------------------*/
// Program      : Fuel and Fluid Control System
// Description  : Flow Control
// Filename     : number_text.h
// C compiler   : RIDE 51 V6.1
// Present by Engineer Mr.Suttichai. Sunthalunai
/*-------------------------------------------------------------------------------------------------*/
/*************************************************************************/
unsigned int  number_1;
unsigned int  number_2;
unsigned int  number_3;
unsigned int  number_4;
```

```
unsigned int  number_5;
unsigned int  number_6;
unsigned int  number_7;
unsigned int  number_8;
unsigned int  number_9;
void number_text_a(unsigned char text_a)
{
  number_1 = text_a ;
}
/*****************************************************************************/
/*****************************************************************************/
void number_text_b(unsigned char text_b)
{
  number_2 = text_b ;
}
/*****************************************************************************/
/*****************************************************************************/
void number_text_c(unsigned char text_c)
{
  number_3 = text_c ;
}

/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
void number_text_d(unsigned char text_d)
{
  number_4 = text_d ;
}
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
```

```
void number_text_e(unsigned char text_e)
{
  number_5 = text_e ;
  number_5 = 0x00 | number_5;
}
/*****************************************************************************/
/*****************************************************************************/
void number_text_f(unsigned char text_f)
{
  number_6 = text_f ;
  number_6 = 0x00 | number_6;
}
/*****************************************************************************/
/*****************************************************************************/

/*****************************************************************************/
void number_text_g(unsigned char text_g)
{
  number_7 = text_g ;
  number_7 = 0x00 | number_7;
}
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
void number_text_h(unsigned char text_h)
{
  number_8 = text_h ;
  number_8 = 0x00 | number_8;
}

/*****************************************************************************/
/*****************************************************************************/
```

```
/*****************************************************************************/
void number_text_i(unsigned char text_i)
{
  number_9 = text_i ;
  number_9 = 0x00 | number_9;
}

/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
```

บทที่ 5

ผลการทดลอง

5.1 กล่าวนำ

จากการดำเนินการทำปริญญานิพนธ์ที่ได้ทำการศึกษาการวัดการไหลของของเหลว และการเขียนโปรแกรมเพื่อควบคุมการนับอัตราการไหล และเขียนโปรแกรมแสดงค่าของอัตราไหลออกมาเป็น จำนวนเงิน จำนวนลิตร จำนวนบาทต่อลิตร (ของของเหลว) โดยแสดงค่าออกมาทาง 7-segment และสามารถส่งค่าไปแสดงผลพร้อมทั้งควบคุมการทำงานด้วยคอมพิวเตอร์ ซึ่งได้ทำการทดลอง โดยได้ทำการทดลองวัดปริมาณของของไหลโดยคิดที่รับค่าเป็นจำนวนบาททาง Switch-matrix และทำการทดลองวัดปริมาณของของไหลโดยคิดที่รับค่าเป็นจำนวนลิตรทาง Switch matrix

5.2 ผลการทดลอง

ตารางการทดลองที่ 5.2.1

วัดปริมาณของของไหลโดยคิดที่รับค่าเป็นจำนวนบาททาง Switch matrix

ในที่นี้ให้ 20 บาท = 1 ลิตร

ป้อนค่า 20 บาท ครั้งที่	จำนวนค่าที่แสดงทาง 7 - Segment		จำนวนค่าที่แสดงทาง Computer	
	บาท	ลิตร	บาท	ลิตร
1	20.1	1.00	20.0	1.00
2	20.6	1.03	20.4	1.02
3	19.8	1.04	19.6	0.98
4	20.0	1.00	19.8	0.99
5	20.4	1.02	20.2	1.01
6	19.8	0.99	19.8	0.99
7	19.6	0.98	19.6	0.98
8	20.2.	1.01	20.0	1.02
9	20.2	1.01	19.8	0.96
10	20.0	1.00	19.8	0.98
ค่าเฉลี่ย	20.07	1.008	19.9	0.993

ตารางการทดลองที่ 5.2.2

วัดปริมาณของของไหลโดยคิดที่รับค่าเป็นจำนวนลิตรทาง Switch matrix

ในที่นี้ให้ 20 บาท = 1 ลิตร

ป้อนค่า 20 บาท ครั้งที่	จำนวนค่าที่แสดงทาง 7 - Segment		จำนวนค่าที่แสดงทาง Computer	
	บาท	ลิตร	บาท	ลิตร
1	20.1	1.00	20.0	1.00
2	20.6	1.03	20.4	1.02
3	19.8	0.96	19.6	0.98
4	20.0	0.99	19.8	0.98
5	20.4	1.02	20.2	0.99
6	19.8	0.98	19.8	0.99
7	19.6	0.96	19.6	0.98
8	20.4	1.02	20.0	1.00
9	20.2	1.01	19.8	0.99
10	20.0	1.00	19.8	0.99
ค่าเฉลี่ย	20.09	0.997	19.9	0.992

5.3 สรุปผลการทดลอง

- จากการทดลองที่ 1 และ 2 ค่าของข้อมูลที่ได้ทาง 7- Segment และคอมพิวเตอร์มีค่าแตกต่างกัน
- การทดลองของแต่ละครั้งมีค่าการแกว่งของข้อมูล
- ค่าเฉลี่ยที่ใกล้เคียงกับจำนวนจริง

บทที่ 6
สรุปผลการวิจัยและข้อเสนอแนะ

6.1 สรุปผลที่ได้จากโครงงาน

จากการได้สร้างโครงงานควบคุมกระบวนการของการไหล ทำให้เราได้ทราบถึงหลักการทำงานของไมโครคอนโทลเลอร์ MCS-51 (Microcontroller MCS-51) และการสื่อสารข้อมูลผ่านพอร์ตอนุกรมไปยังคอมพิวเตอร์ได้ และเข้าใจกระบวนการการรับและส่งค่าของรหัสแอสกี้ไปยังวิชวล (VISUAL)

6.2 ข้อเสนอแนะและแนวทางพัฒนา

โครงงานนี้เป็นโครงงานเริ่มแรก ซึ่งเกิดอาจจะมีข้อบกพร่องบางประการปัญหาที่ เกิดในการทำโครงงาน

- การใช้ภาษาซีในการเขียนโปรแกรมผู้เขียนต้องมีความรู้ มีความเข้าใจในภาษาที่เขียน
- การสื่อสารระหว่างคอมพิวเตอร์กับไมโครคอนโทรลเลอร์ต้องอาศัยช่วงเวลาในการสื่อสาร จึงมีข้อผิดพลาดข้อมูลที่ส่ง
- การนับจำนวนพัลส์ของไมโครคอนโทรลเลอร์ อาจมีการนับไม่ได้จำนวนพัลส์ที่ส่งมาจริง อาจมีตาบผ่านระหว่างรูปพัลส์หรือมีการตกไปของรูปพัลส์เพราะตัดที่รูปพัลส์ลอจิก “ 0 “
- ขนาดความยาวของท่อส่งจ่ายต่อของไหลมีผลต่อจำนนวนพัลส์ที่เกิดขึ้น
- ช่วงเวลาในการเปิดปิดรีเลย์ (Relay) เพื่อควบคุมกระบวนการมีผลต่อการนับจำนวนพัลส์
- การใช้ภาษาเบสิก(Basic)ในการเขียนวิชวล (Visual) ผู้เขียนต้องมีความรู้ความเข้าใจเกี่ยวกับหลักการเขียนที่ถูกต้อง