เครื่องบันทึกข้อมูลแบบพกพา

PORTABLE DATA LOGGER

นายภูธร พงษ์ไทย

นายสุรชัย โตอนันต์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมการวัดคุม

ภาควิชาวิศวกรรมการวัดคุม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

PORTABLE DATA LOGGER

PUTHON PONGTHAI

SURACHAI TOANAN

THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

BACHELOR OF ENGINEERING IN INSTRUMENTATION ENGINEERING

DEPARTMENT OF INSTRUMENTATION ENGINEERING

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2002

ภาควิชาวิศวกรรมการวัดคุม

คณะวิศวกรรมศาตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

หัวข้อปริญญานิพนธ์ เครื่องบันทึกข้อมูลแบบพกพา

PORTABLE DATA LOGGER

นักศึกษาผู้จัดทำ นายภูธร พงษ์ไทย **รหัสประจำตัว** 43015578

นายสุรชัย โตอนันต์ **รหัสประจำตัว** 43015595

ปริญญา วิศวกรรมศาสตรบัณฑิต

สาขาวิชา วิศวกรรมการวัดคุม

ปีการศึกษา 2545

อาจารย์ผู้ควบคุมปริญญานิพนธ์	**ลายมือชื่อ**
ผศ. ประสิทธิ์ จุลเสรีวงศ์ อ.พิทยา ปานนิล อ.อัมพวัน ใจกล้า	

วัน/เดือน/ปี ที่สอบ วันอังคารที่ 25 มีนาคม พ.ศ. 2546

สถานที่สอบ ณ ห้องสอบปริญญานิพนธ์ ภาควิชาวิศวกรรมการวัดคุม

ภาควิชารับรองแล้ว

(ผศ.ประสิทธิ์ จุลเสรีวงศ์)

หัวหน้าภาควิชาวิศวกรรมการวัดคุม

หัวข้อปริญญานิพนธ์	เครื่องบันทึกข้อมูลแบบพกพา
	PORTABLE DATA LOGGER
นักศึกษาผู้จัดทำ	นายภูธร พงษ์ไทย
	นายสุรชัย โตอนันต์
อาจารย์ที่ปรึกษา	ผศ.ประสิทธิ์ จุลเสรีวงศ์
	อ.พิทยา ปานนิล
	อ.อัมพวัน ใจกล้า
ปีการศึกษา	2545

บทคัดย่อ

เครื่องบันทึกข้อมูลแบบพกพา เป็นอุปกรณ์ที่ทำหน้าที่บันทึกสัญญาณอนาลอกที่ได้จากตัวตรวจจับ เพื่อนำข้อมูลที่เปลี่ยนแปลงเหล่านี้เก็บในฐานข้อมูลในบอร์ดไมโครคอนโทรลเลอร์ โดยสามารถปรับแต่งฐานเวลาในการบันทึกข้อมูลได้ และในขณะที่บันทึกข้อมูลสามารถทำการเรียกค่าข้อมูลดูได้จากหน้าจอของเครื่องบันทึกได้ นอกจากนี้ยังสามารถนำเครื่องบันทึกข้อมูลนี้ไปเชื่อมต่อกับเครื่องคอมพิวเตอร์ส่วนบุคคล เพื่อแสดงข้อมูลทั้งหมดที่บันทึกมาเป็นกราฟหรือข้อมูลเชิงสถิติด้วยโปรแกรมที่เขียน โดย Visual Basic และสามารถแสดงผลผ่านเครื่องพิมพ์ได้อีก

Thesis Title	Potable Data Logger
Authors	Mr.Puthon Pongthai
	Mr.Surachai Toanan
Thesis Advisor	Asst.Prof.Prasit Julsereewong
	Mr.Pittaya Pannil
	Miss.Amphawan Chaikla
Year	2002

ABSTRACT

This project presents a portable data logger. It is an equipment to record the analog signal received from the sensors. The input data will be stored in the database on the microcontroller board. The record time figure can be adjusted to save the data. Moreover, the data value can be shown on the display of recorder while it is being saved. The proposed data logger can additionally connect to the personal computer (PC) to show the total saved data as a graph or statistical data, which can be obtained by using the program written in Visual Basic. Furthermore, the data can be printed out through the printer.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ดี เพราะได้รับความเมตตาจาก ผศ.ประสิทธิ์ จุลเสรีวงศ์ ที่ได้ให้คำแนะนำแก่ผู้วิจัยตลอดมา อีกทั้งยังเอื้อเฟื้ออุปกรณ์และเครื่องมือต่างๆ ที่จำเป็น ในการทำปริญญานิพนธ์นี้ ทางคณะผู้วิจัยรู้สึกซาบซึ้งและขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณ อาจารย์ภาควิชาวิศวกรรมการวัดคุมทุกท่านที่ให้คำแนะนำอันเป็นประโยชน์ต่อการทำปริญญานิพนธ์ฉบับนี้

และที่ลืมมิได้คือ ขอกราบขอบพระคุณคุณพ่อและคุณแม่ ที่สนับสนุนและเป็นแรงบันดาลใจในการทำปริญญานิพนธ์ฉบับนี้

คุณค่าและประโยชน์อันพึงมีจากปริญญานิพนธ์ฉบับนี้ ทางคณะผู้วิจัยขอมอบแด่ผู้มีพระคุณทุกท่าน

คณะผู้จัดทำ

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญภาพ

ภาพที่ หน้า

สารบัญภาพ (ต่อ)

ภาพที่ หน้า

บทที่ 1
บทนำ

1.1 กล่าวนำ

เครื่องบันทึกข้อมูล (Data Logger) คือ เครื่องมืออิเล็กทรอนิกส์ที่ใช้บันทึกผลการวัด เช่น อุณหภูมิ, ความชื้นสัมพัทธ์, ความดันและอื่นๆ ได้เป็นเวลานาน ยกตัวอย่างเช่น เครื่องบันทึกข้อมูลที่ใช้แบตเตอรี่เป็นแหล่งจ่ายกำลังงาน ทำหน้าที่เป็นไมโครโปรเซสเซอร์ (Microprocessor) เก็บค่าข้อมูลที่วัดได้จากตัวตรวจจับ (Sensor)โดยส่วนมากเครื่องบันทึกข้อมูลจะมีโปรแกรมหรือซอฟแวร์ให้สามารถใช้งานกับเครื่องคอมพิวเตอร์ส่วนบุคคล (PC) เพื่อดูค่าข้อมูลตั้งแต่เริ่มแรกจนถึงเวลาที่ตั้งไว้เพื่อวิเคราะห์การเปลี่ยนแปลง

การใช้งานเครื่องบันทึกข้อมูลจะใช้ในพื้นที่ ที่ห่างไกลหรือทุกๆที่ ที่ต้องการบันทึกค่าโดยอาจจะต้องอาศัยแบตเตอรี่เป็นแหล่งจ่ายกำลังงาน การทำงานจะเริ่มตั้งแต่กดสวิทช์เปิดเครื่องและทำการตั้งค่าพารามิเตอร์ เช่น ค่าช่วงเวลาในการสุ่ม Sampling intervals เวลาเริ่มต้น Start time จากนั้นจะต้องทำการต่อกับตำแหน่งที่ต้องการบันทึกค่า โดยการต่อกับสัญญาณอนาลอกที่ได้จากอุปกรณ์ตรวจจับ การทำงานของเครื่องบันทึกข้อมูลนั้นจะเก็บค่าการวัดไว้ในหน่วยความจำตามเวลาที่ตั้งไว้แล้วจึงนำมาต่อเข้ากับคอมพิวเตอร์เพื่อแสดงค่าที่บันทึกมาเป็นกราฟและตารางข้อมูล

1.2 วัตถุประสงค์ของปริญญานิพนธ์

1. เพื่อศึกษาการทำงานของเครื่องบันทึกข้อมูล
2. เรียนรู้และใช้งานไมโครคอนโทรลเลอร์เบอร์ PIC 16F877
3. ศึกษาและเขียนโปรแกรมการใช้งานไมโครคอนโทรลเลอร์เบอร์ 16F877 เพื่อทำการแปลงสัญญาณอนาลอกเป็นดิจิตอล
4. ศึกษาและเขียนโปรแกรม Visual Basic เพื่อติดต่อระหว่างไมโครคอนโทรลเลอร์เบอร์ 16F877 กับคอมพิวเตอร์ โดยใช้มาตรฐาน RS 232

1.3 ขอบเขตของปริญญานิพนธ์

เครื่องบันทึกข้อมูลแบบพกพาใช้ไมโครคอนโทรลเลอร์เบอร์ PIC16F877 ในการควบคุมการทำงานทั้งหมด โดยการออกแบบได้เน้นการใช้ประโยชน์จากคุณลักษณะทั้งหมดที่มีอยู่ภายในตัวไมโครคอนโทรลเลอร์ให้มากที่สุดทำให้การประยุกต์ใช้งานมีความสะดวก และขนาดโครงสร้างของวงจรโดยรวมเล็กลงอีกด้วย ซึ่งได้กำหนดคุณสมบัติของเครื่องไว้ดังนี้

1. สามารถรับสัญญาณอนาลอกอินพุท ที่อยู่ในรูปของสัญญาณแรงดันที่มีขนาด 0–5 Vdc จำนวน 4 ช่องสัญญาณ โดยสามารถปรับค่าแรงดันอ้างอิงในช่วง 1 - 5 V ได้

2. นำสัญญาณอนาลอกที่ได้แปลงเป็นข้อมูลดิจิตอลก่อน แล้วบันทึกลงในหน่วยความจำภายนอกที่ต่อเพิ่มเข้ามา

3. สามารถตั้งค่าเวลาในการบันทึกค่าของแต่ละช่องสัญญาณได้โดยใช้คีย์สวิทช์

4. สามารถแสดงค่าการแปลงสัญญาณอนาลอกอินพุทเป็นสัญญาณดิจิตอลที่หน้าจอ LCD

5. สามารถนำข้อมูลที่บันทึกไว้ในหน่วยความจำมาแสดงค่าข้อมูลแบบกราฟ หรือตารางแสดงข้อมูลที่หน้าจอคอมพิวเตอร์

6. สามารถนำค่าที่ได้มาพิมพ์ออกทางเครื่องพิมพ์(Printer)

1.4 รายละเอียดของปริญญานิพนธ์

ปริญญานิพนธ์นี้จะมีเนื้อหาซึ่งจะกล่าวถึง เครื่องบันทึกข้อมูล วัตถุประสงค์ในการศึกษา ขอบเขตในการทำ รายละเอียดของตัวอุปกรณ์ที่นำมาใช้ รวมทั้งทฤษฎีเกี่ยวกับโปรแกรมที่ใช้เขียนวงจรการทำงาน การทดสอบ บทสรุปและวิจารณ์ สำหรับปริญญานิพนธ์นี้สามารถแบ่งเนื้อหาออกได้ทั้งหมด 7 บทดังนี้

บทที่ 1 บทนำ

บทที่ 2 ไมโครคอนโทรลเลอร์ 16F877

บทที่ 3 การสื่อสารแบบอนุกรม

บทที่ 4 ทฤษฎีเกี่ยวกับโปรแกรมที่ใช้เขียน

บทที่ 5 การออกแบบเครื่องบันทึกข้อมูลแบบพกพา

บทที่ 6 การทดสอบการทำงาน

บทที่ 7 บทสรุปและวิจารณ์

บทที่ 2
ไมโครคอนโทรลเลอร์ PIC16F877

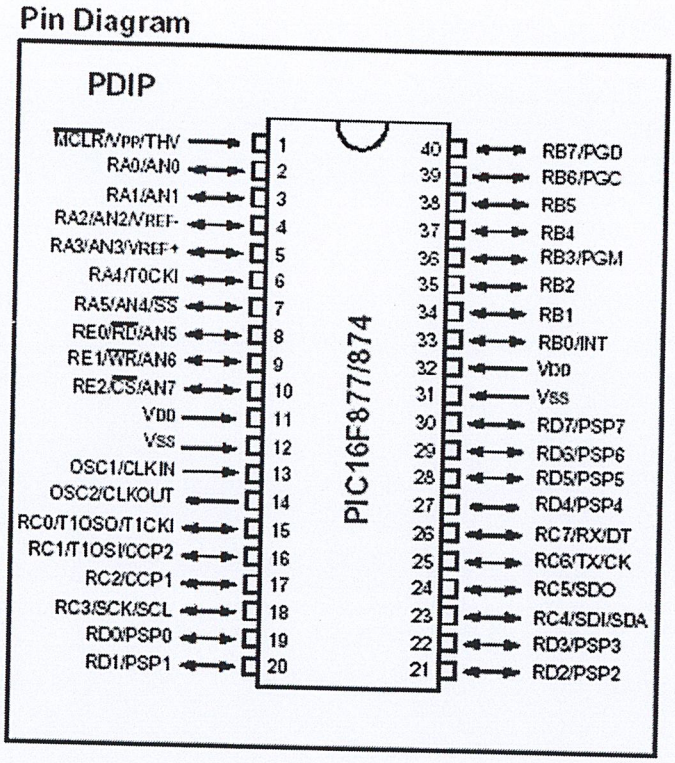
ในปัจจุบันมีไมโครคอนโทรลเลอร์หลายรุ่นหลายตระกูลถูกผลิตขึ้น เพื่อตอบสนองการใช้งานที่แตกต่างกัน โดยมีความแตกต่างทั้งความเร็วสัญญาณนาฬิกา ความยาวเวิร์ดข้อมูล ขนาดของหน่วยความจำ จำนวนอินพุท เอาท์พุทพอร์ท แต่โดยทั่วไปไมโครคอนโทรลเลอร์ สามารถจำแน กได้เป็น 2 กลุ่มตามสถาปัตยกรรมที่ใช้ คือไมโครคอนโทรลเลอร์แบบ CISC (Complex Instruction Set Computer) และ ไมโครคอนโทรลเลอร์แบบ RISC (Reduce Instruction Set Computer) โดย CISC เช่น MCS-51 มีจุดเด่นที่มีจำนวนชุดคำสั่งมาก แต่ละคำสั่งใช้จำนวนสัญญาณนาฬิกาต่างกัน ส่วน RISC มีชุดคำสั่งน้อยกว่า โดยชุดคำสั่งถูกจัดให้มีโครงสร้างการทำงานคล้ายกัน ทำให้สามารถทำงานแบบไปป์ไลน์ได้ โดยใช้สัญญาณนาฬิกาเพียงหนึ่งลูกต่อหนึ่งคำสั่ง ดังนั้น ความเร็วของ RISC จะสูงกว่า CISC ประมาณสองถึงสี่เท่า ที่ความเร็วสัญญาณนาฬิกาเดียวกัน แต่ว่าอาจจะไม่เป็นเช่นนั้นเสมอไป เนื่องจากชุดคำสั่งที่น้อยกว่าทำให้ RISC ต้องมีคำสั่งมากกว่าหนึ่งคำสั่งเพื่อทำงานอย่างเดียวกับ CISC ที่ใช้เพียงคำสั่งเดียวนอกจากนี้ชุดคำสั่งที่ง่ายกว่าของ RISC จึงส่งผลให้ขนาดของวงจรในชิปเล็กกว่า การเพิ่มวงจรพิเศษเช่น วงจรแปลงสัญญาอนาลอกเป็นดิจิตอลจึงทำได้

ไมโครคอนโทรลเลอร์ตระกูล PIC ของบริษัทไมโครชิพเทคโนโลยี ที่ใช้สถาปัตยกรรมแบบ RISC (Reduce Instruction Set Computer) ซึ่งแตกต่างจากไมโครคอนโทรลเลอร์ตระกูล 8051 ที่เป็นสถาปัตยกรรมแบบ CISC (Complex Instruction Set Computer)

สำหรับตัวไมโครคอนโทรลเลอร์เบอร์ PIC16F877 ออกสู่ตลาดครั้งแรกในปี ค. ศ.1998 มีหน่วยความจำโปรแกรมแบบแฟลชขนาด 8 กิโลเวิร์ด นอกจากนั้น ยังมีไมโครคอนโทรลเลอร์ในรุ่นเดียวกันอีก 3 เบอร์ คือ PIC16F873, PIC16F874 และ PIC16F876 ซึ่งมีขนาดหน่วยความจำโปรแกรมและจำนวนพอร์ทอินพุท/เอาท์พุทแตกต่างกันไป

2.1 คุณสมบัติของไมโครคอนโทรลเลอร์ PIC16F877

ไมโครคอนโทรลเลอร์เบอร์ PIC16F877 เป็นผลิตภัณฑ์ของบริษัทไมโครชิพเทคโนโลยี (Microchip Technology Inc.) เป็นไมโครคอนโทรลเลอร์แบบRISC มีโมดูลสำหรับจัดการสิ่งต่าง ๆ หลายส่วน อาทิ วงจรแปลงสัญญาณอนาลอกเป็นดิจิตอล ส่วนติดต่อกับบัสแบบ I^2C ส่วนติดต่อส่งข้อมูลอนุกรมอะซิงโครนัส UART สามารถทำงานที่สัญญาณนาฬิกาสูงสุดถึง 20 เมกกะเฮิร์ทซ์ที่กระแส 7 มิลลิแอมป์ ภาพที่ 2.1 คือลักษณะภายนอกพร้อมคำอธิบายตำแหน่งขาต่อใช้งาน



ภาพที่ 2.1 ไมโครคอนโทรลเลอร์เบอร์ PIC16F877

ส่วนสถาปัตยกรรมภายในของไมโครคอนโทรลเลอร์ PIC16F877 ดูได้จากภาพที่ 2.2 ซึ่งจะประกอบไปด้วย โครงสร้างภายในและแบ่งออกเป็นส่วนการทำงานแต่ละส่วนพร้อมรายละเอียดในหัวข้อที่ 2.2

2.2 สถาปัตยกรรมแบบ RISC (Reduce Instruction Set Computer)

แบ่งออกเป็นส่วนการทำงานแต่ละส่วนพร้อมรายละเอียดดังนี้

1. ใช้การประมวลผลแบบไปป์ไลน์ (Pipelining)
2. ทุกคำสั่งประมวลผลเสร็จภายใน 1 รอบสัญญาณนาฬิกา ยกเว้นการเรียกโปรแกรมย่อย ซึ่งใช้ 2 รอบสัญญาณนาฬิกา
3. ความเร็วสัญญาณนาฬิกาสูงสุด 20 เมกะเฮิร์ทซ์
4. บัสคำสั่งขนาด 14 บิท แยกอิสระกับบัสข้อมูลขนาด 8 บิท
5. หน่วยความจำโปรแกรมแบบแฟลช ขนาด8 กิโลเวิร์ด (1 เวิร์ดของคำสั่งมีขนาด14 บิท)
6. หน่วยความจำข้อมูลแบบ SRAM (รีจิสเตอร์) ขนาด 368 ไบท์
7. หน่วยความจำข้อมูลแบบ EEPROM ขนาด 256 ไบท์

8. สามารถอ้างอิงแอดเดรสแบบโดยตรง โดยอ้อมและแบบสัมพัทธ์

9. มีฮาร์ดแวร์แสตก 8 ระดับ

10. อินเตอร์รัพท์สูงสุด 14 แหล่ง

11. อินพุท / เอาท์พุทพอร์ทรวม 33 บิท (พอร์ท A, B, C, D, E)

12. ตัวจับเวลา / ตัวนับขนาด 8 บิท 2 ตัวและขนาด 16 บิท 1 ตัว วงจรตรวจจับเปรียบเทียบและกำเนิดสัญญาณแบบ PWM 2 ตัว

13. สนับสนุนการสื่อสารอนุกรมแบบ Synchronous Serial Port (SSP), I^2C (Master / Slave) และ Universal Synchronous Asynchronous Receiver Transmitter (USART)

14. สนับสนุนการสื่อสารแบบขนาน ขนาด 8 บิท (Parallel Slave Port: PSP)

15. วงจรแปลงสัญญาณอนาลอกเป็นดิจิตอล ความละเอียด 10 บิท 8 ช่องสัญญาณ

16. สามารถจะโปรแกรมไมโครคอนโทรลเลอร์ ที่อยู่ภายในระบบได้โดยตรงผ่านพอร์ท 2 พอร์ท (In circuit Serial Programming :ICSP)

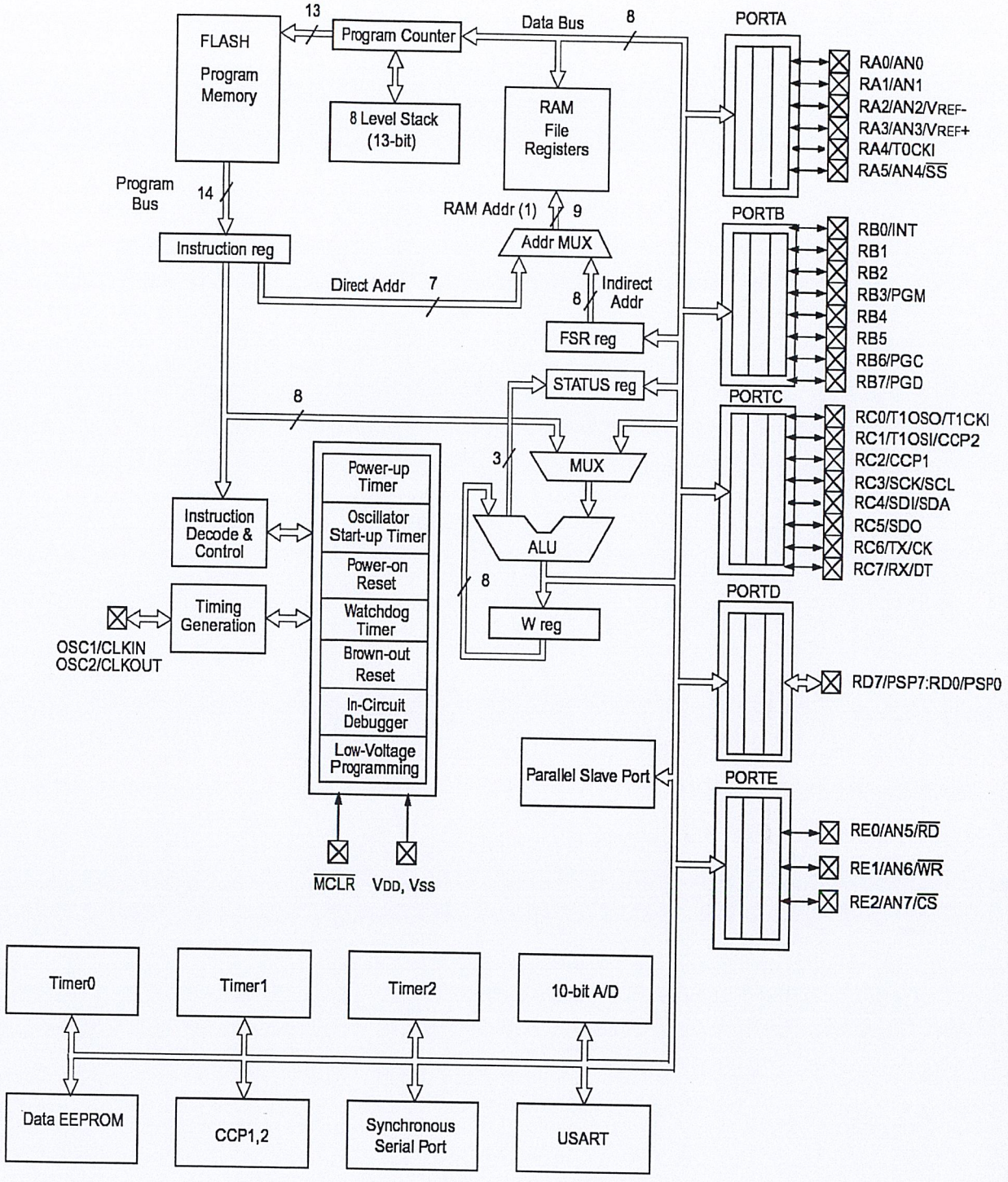
17. ระบบรีเซตอัตโนมัติเมื่อเริ่มจ่ายกระแสไฟฟ้าเข้าสู่ไมโครคอนโทรลเลอร์ (Power On Reset: POR)

18. มีการหน่วงเวลารอไฟเลี้ยง และออสซิลเลเตอร์ให้อยู่ในสภาวะที่เสถียร (Power - Up Timer: PWRT, Oscillator Start Up Timer:OST)

19. ใช้กระแสน้อยกว่า 2 มิลลิแอมป์ที่ 4 เมกะเฮิร์ทซ์ และน้อยกว่า 1 ไมโครแอมป์เมื่ออยู่ในโหมด Sleep

20. สามารถป้องกันการอ่านโปรแกรมจากไอซีได้

21. จำนวนรอบการเขียนหน่วยความจำโปรแกรมแบบแฟลช 1000 ครั้ง และรอบการเขียนหน่วยความจำข้อมูลแบบ EEPROM 10,000,000 ครั้ง



ภาพที่ 2.2 สถาปัตยกรรมภายในของ PIC16F877

ตารางที่ 2.1 แสดงการจัดขาและหน้าที่ของแต่ละขาใน PIC16F877

ชื่อขา	ตำแหน่งขา	รูปแบบ	อธิบาย (เฉพาะหน้าที่ที่สำคัญ)
OSC1/CLKIN	13	I	สัญญาณออสซิลเลเตอร์ขาเข้า
OSC2/CLKOUT	14	O	สัญญาณออสซิลเลเตอร์ขาออก
MCRL/Vpp/THV	1	I/P	รีเซตเมื่อเป็นลอจิกต่ำ/แรงดันสูงเพื่อโปรแกรม IC
			พอร์ท A เป็นพอร์ท 2 ทิศทาง
RA0/AN0	2	I/O	RA0 เป็นอนาลอกอินพุทช่อง 0
RA1/AN1	3	I/O	RA1 เป็นอนาลอกอินพุทช่อง 1
RA2/AN2/Vref-	4	I/O	RA2 เป็นอนาลอกอินพุทช่อง 2 หรือแรงดันอ้างอิงลบ
RA3/AN3/Vref+	5	I/O	RA3 เป็นอนาลอกอินพุทช่อง 3 หรือแรงดันอ้างอิงบวก
RA4/T0CKI	6	I/O	
RA5/SS/AN4	7	I/O	RA5 เป็นอนาลอกอินพุทช่อง 4
			พอร์ท B เป็นพอร์ท 2 ทิศทาง โปรแกรมให้มีตัวต้านทาน
RB0/INT	33	I/O	พูลอัพได้
RB1	34	I/O	RB0 เป็นขารับอินเตอร์รัพท์ภายนอก
RB2	35	I/O	
RB3/PGM	36	I/O	
RB4	37	I/O	RB3 เป็นขาโปรแกรมแรงดันต่ำ
RB5	38	I/O	สร้างอินเตอรัพท์เมื่อขาเปลี่ยนสถานะ
RB6/PGC	39	I/O	สร้างอินเตอรัพท์เมื่อขาเปลี่ยนสถานะ
RB7/PGD	40	I/O	สร้างอินเตอรัพท์เมื่อขาเปลี่ยนสถานะ หรือขารับสัญญาณนาฬิกาสำหรับการโปรแกรม IC สร้างอินเตอรัพท์เมื่อขาเปลี่ยนสถานะ หรือขารับสัญญาณข้อมูลสำหรับการโปรแกรม IC
			พอร์ท C เป็นพอร์ท 2 ทิศทาง
RC0/T1OSO/T1CLK	15	I/O	
RC1/T1OSI/CCP2	16	I/O	
RC2/CCP1	17	I/O	
RC3/SCK/SCL	18	I/O	RC3 เป็นขาสัญญาณนาฬิกาสำหรับการสื่อสารแบบ I^2C
RC4/SDI/SDA	23	I/O	RC3 เป็นขาสัญญาณข้อมูลสำหรับการสื่อสารแบบ I^2C
RC5/SDO	24	I/O	
RC6/TX/CK	25	I/O	RC6 เป็นขาส่งข้อมูลสำหรับ USART แบบอะซิงโครนัส
RC7/RX/DT	26	I/O	RC7 เป็นขาส่งข้อมูลสำหรับ USART แบบอะซิงโครนัส

I = Input, O = Output, P = Power

ตารางที่ 2.1 (ต่อ)

ชื่อขา	ตำแหน่งขา	รูปแบบ	อธิบาย (เฉพาะหน้าที่ที่สำคัญ)
			พอร์ท D เป็นพอร์ท 2 ทิศทาง หรือพอร์ทขนาน
RD0/PSP0	19	I/O	
RD1/PSP1	20	I/O	
RD2/PSP2	21	I/O	
RD3/PSP3	22	I/O	
RD4/PSP4	27	I/O	
RD5/PSP5	28	I/O	
RD6/PSP6	29	I/O	
RD7/PSP7	30	I/O	
			พอร์ท E เป็นพอร์ท 2 ทิศทาง
RE0/RD/AN5	8	I/O	RE0เป็นสัญญาณอ่าน สำหรับพอร์ทขนาน หรือเป็นอนาลอกอินพุทช่อง 5
RE1/WR/AN6	9	I/O	RE1เป็นสัญญาณเขียน สำหรับพอร์ทขนาน หรือเป็นอนาลอกอินพุทช่อง 6
RE2/CS/AN7	10	I/O	RE2เป็นสัญญาณเลือก สำหรับพอร์ทขนาน หรือเป็นอนาลอกอินพุทช่อง 7
Vss	12, 31	P	กราวนด์
Vdd	11, 32	P	แรงดันบวก

I = Input, O = Output, P = Power

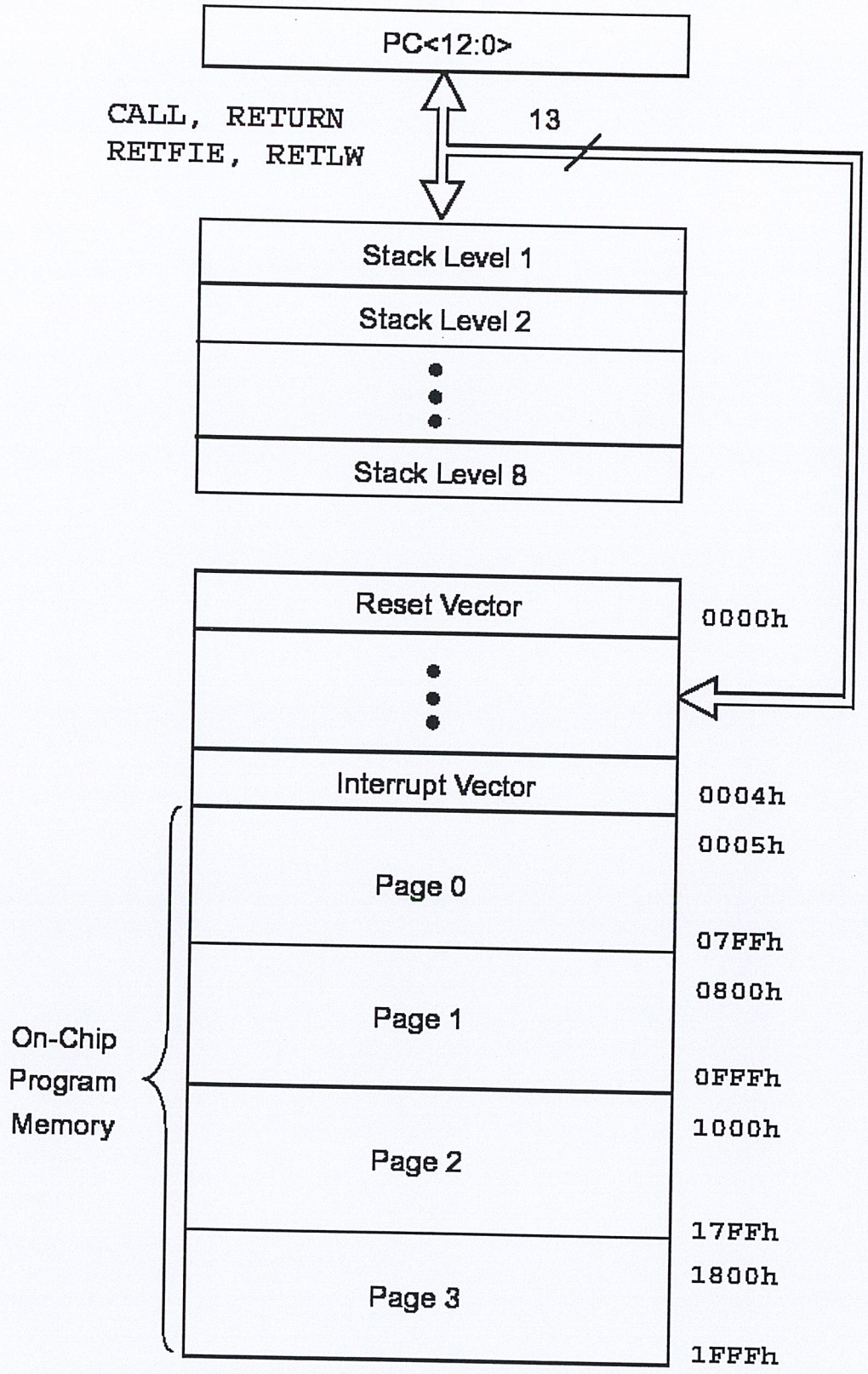
2.3 การจัดการหน่วยความจำ

หน่วยความจำของไมโครคอนโทรลเลอร์ PIC16F877 แบ่งได้เป็น 3 ส่วน คือหน่วยความจำโปรแกรม, หน่วยความจำข้อมูล หรือรีจิสเตอร์และหน่วยความจำข้อมูลแบบ EEPROM ซึ่งข้อมูลจะไม่สูญหายแม้ไม่จ่ายไฟเลี้ยง

บัสของหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลจะถูกแยกออกจากกัน ดังนั้นการเข้าถึงหน่วยความจำทั้งสองแบบสามารถเกิดขึ้นได้พร้อมกัน

2.3.1 หน่วยความจำโปรแกรม

หน่วยความจำโปรแกรมขนาด 8 กิโลเวิร์ดถูกแบ่งเป็น 4 เพจ (page) เพจละ 2 กิโลเวิร์ดโดยมีตำแหน่งของรีเซตเวคเตอร์อยู่ที่ 0000h และตำแหน่งของอินเตอร์รัพท์เวคเตอร์อยู่ที่ 0004h ดูภาพที่ 2.3 ประกอบ

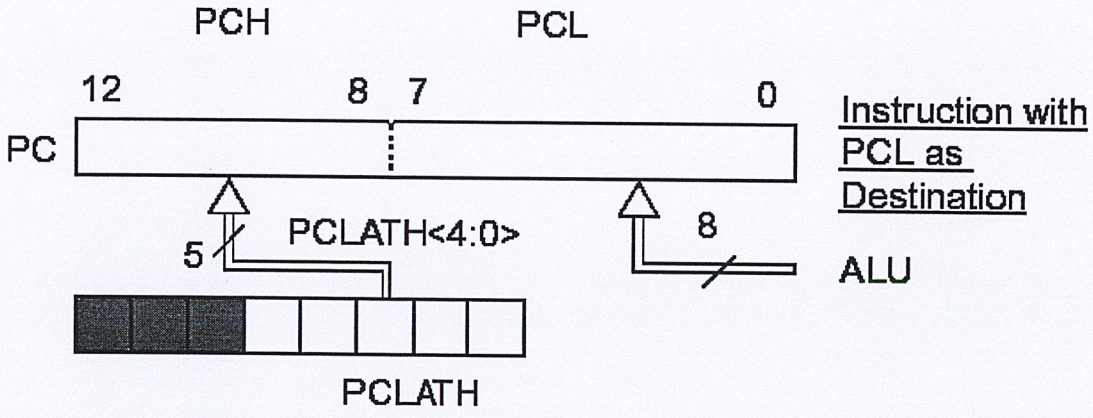


ภาพที่ 2.3 แผนผังหน่วยความจำโปรแกรม และแสตกของ PIC16F877

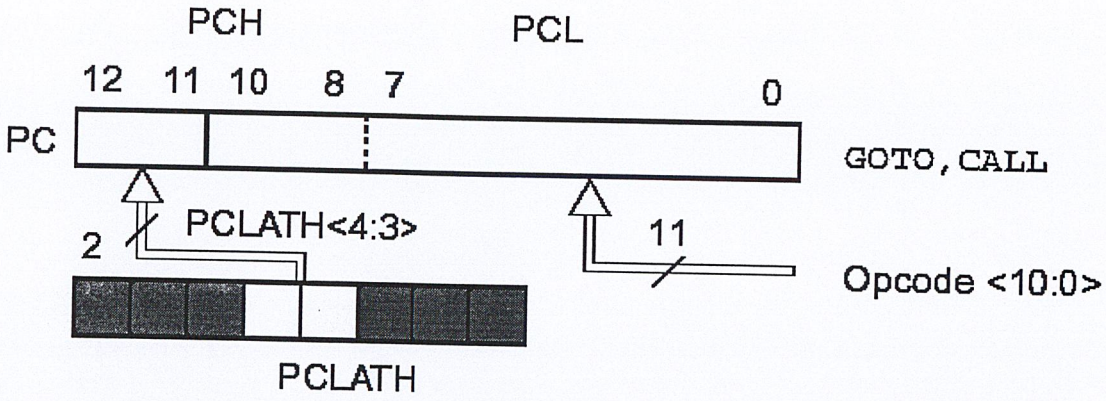
2.3.2 โปรแกรมเคาน์เตอร์ (Program counter: PC) และแสตก (Stack)

โปรแกรมเคาน์เตอร์ มีขนาด 13 บิท ได้ไบท์ต่ำได้จากรีจิสเตอร์ PCL ซึ่งสามารถอ่านและเขียนได้ ส่วนไบท์สูง (PC<12:8>) ไม่สามารถอ่านได้แต่สามารถเขียนได้โดยอ้อมผ่านทางรีจิสเตอร์ PCLATH เมื่อเกิดการรีเซต รีจิสเตอร์ทั้งสองจะกลายเป็นศูนย์

ภาพที่ 2.4 แสดงการใส่ค่าลงในโปรแกรมเคาน์เตอร์ในสองสถานการณ์ รูปแรกค่าไบท์ต่ำได้จาก PCL และไบท์สูงได้จาก PCLATH<4:0> ส่วนรูปที่สองนั้นเกิดจากการทำคำสั่ง GOTO หรือ CALL ซึ่ง 11 บิทล่างของโปรแกรมเคาน์เตอร์ได้จาก Opcode<10:0> ส่วน 2 บิทบนนั้นจะได้มาจาก PCLATH<4:3> ทำให้การกระโดดทำได้ภายใน 2048 ตำแหน่งในเพจเดียวกันเท่านั้นการอ้างถึงตัวแอดเดรสที่ไกลกว่านี้ทำได้ โดยเปลี่ยนแปลงค่าใน PCLATH<4:3> ซึ่งเป็นค่าที่จะถูกใส่เอาไว้ในตัวโปรแกรมเคาน์เตอร์ 2 บิทบน ทำให้สามารถอ้างแอดเดรสได้ตลอด 8192 ตำแหน่งแต่ถ้าเป็นการอ้างแอดเดรสที่สูงกว่ากำหนดจะทำให้เกิดการวนซ้ำตำแหน่งขึ้น



ภาพที่ 2.4 (ก) การใส่ค่าลงในโปรแกรมเคาน์เตอร์ค่าไบท์ต่ำได้จาก PCL และไบท์สูงได้จาก PCLATH



ภาพที่ 2.4 (ข) การใส่ค่าลงในโปรแกรมเคาน์เตอร์จากการทำคำสั่ง GOTO

แสตกของไมโครคอนโทรลเลอร์ PIC16F877 มี 8ระดับ แสตกไม่ใช่ส่วนของโปรแกรมหรือข้อมูลและแสตกพอยน์เตอร์ก็ไม่สามารถอ่านหรือเขียนได้ โปรแกรมเคาน์เตอร์จะถูกใส่ (Push) ลงในแสตกเมื่อเกิดการทำคำสั่ง CALL หรือเกิดการเรียกโปรแกรมบริการอินเตอร์รัพท์ และจะคืน (Pop) สู่โปรแกรมเคาน์เตอร์เมื่อทำคำสั่ง RETURY, RETLW หรือ RETFIE

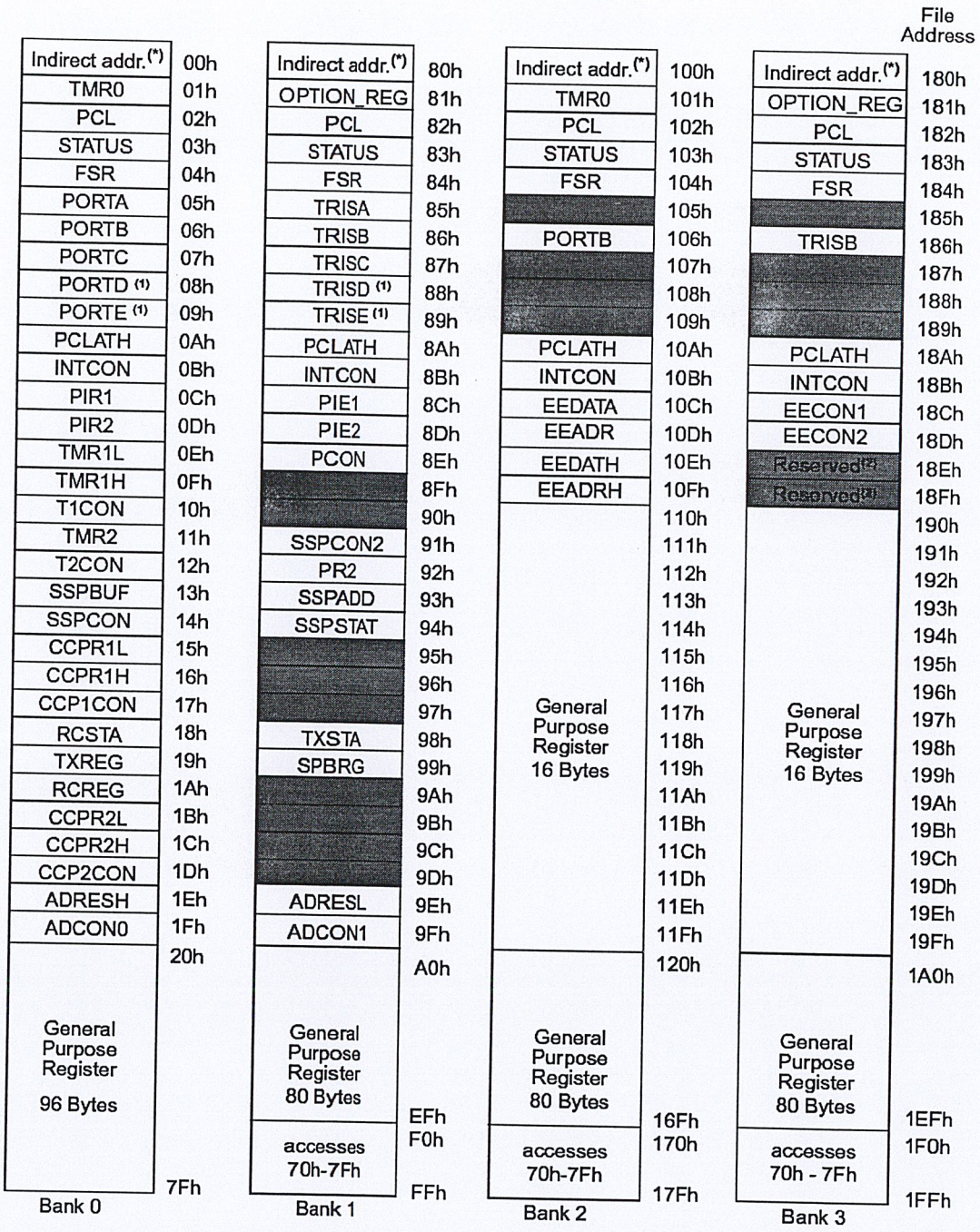
หากแสตกถูกใส่เกิน 8 ครั้ง ครั้งที่ 9 ค่าในโปรแกรมเคาน์เตอร์จะทับค่าที่ใส่ในแสตกครั้งแรกและวนทับไปเรื่อย ๆ ใน PIC16F877 จะไม่มีแฟลกที่บ่งบอกการเกิดแสตกโอเวอร์โฟลว์ และแสตกอันเดอร์โฟลว์

2.3.3 หน่วยความจำข้อมูล (รีจิสเตอร์)

หน่วยความจำข้อมูลถูกแบ่งออกเป็น4 แบงก์ แต่ละแบงก์มี128 ไบท์ ประกอบด้วยรีจิสเตอร์ทั่วไปคือ (General Purpose Register) และรีจิสเตอร์เฉพาะคือ (Special Function Register) การเข้าถึงแต่ละแบงก์ทำโดยการเปลี่ยนบิท RP1 และ RP0 ในรีจิสเตอร์สถานะ รีจิสเตอร์เฉพาะบางตัวที่สำคัญจะมีซ้ำอยู่ในหลายแบงก์เพื่อความรวดเร็วในการเข้าถึง

การเข้าถึงรีจิสเตอร์สามารถอ้างอิงแบบโดยตรง หรือโดยอ้อมผ่านทางไฟล์ซีเล็กรีจิสเตอร์ (File Select Register: FSR)

รีจิสเตอร์เฉพาะสามารถแบ่งเป็นรีจิสเตอร์ที่ถูกใช้โดยหน่วยประมวลผลกลางและรีจิสเตอร์สำหรับโมดูลที่ทำหน้าที่พิเศษเช่นส่วนแปลงสัญญาณอนาลอกเป็นดิจิตอลสำหรับรีจิสเตอร์เฉพาะที่ใช้โดยหน่วยประมวลกลางที่สำคัญคือ รีจิสเตอร์สถานะ



ภาพที่ 2.5 แผนผังหน่วยความจำข้อมูล (รีจิสเตอร์)

2.3.4 รีจิสเตอร์สถานะ (Status Register)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	**RP1**	**RP0**	**/TO**	**/PD**	**Z**	**DC**	**C**
บิท7							บิท0

IRP บิทเลือกรีจิสเตอร์แบงก์ (ใช้สำหรับการอ้างอิงแอดเดรสโดยอ้อม)

1 = แบงก์ 2, 3 (100h-1FFh) 0 = แบงก์ 0,1 (00h-FFh)

RP1:RP0 บิทเลือกรีจิสเตอร์แบงก์ (ใช้สำหรับการอ้างอิงแอดเดรสโดยตรง)

11 = แบงก์ 3 (180h-1FFh) 10 = แบงก์ 2 (100h-17Fh)

01 = แบงก์ 1 (80h-1FFh) 00 = แบงก์ 0 (00h-17Fh)

/TO บิทว๊อชด็อกไทม์เอาท์

1 = หลังจากจ่ายกระแสไฟ หรือใช้คำสั่ง CLRWDT หรือ SLEEP

0 = เกิดว๊อชด็อกไทม์เอาท์

/PD บิทลดพลังงาน

1 = หลังจากจ่ายกระแสไฟ หรือใช้คำสั่ง CLRWDT

0 = หลังใช้คำสั่ง SLEEP

Z บิทศูนย์

1 = ผลลัพท์ของการกระทำทางคณิตศาสตร์ หรือตรรกะ เป็น 0

0 = ผลลัพท์ของการกระทำทางคณิตศาสตร์ หรือตรรกะ ไม่เป็น 0

DC บิทตัวทด/ยืม* (เป็นผลจากการทำคำสั่ง ADDWF, ADDLW, SUBLW, SUBWF)

1 = เกิดการทดจาก 4 บิทล่าง 0 = ไม่เกิดการทดจาก 4 บิทล่าง

C บิทตัวทด/ยืม* (เป็นผลจากการทำคำสั่ง ADDWF, ADDLW, SUBLW, SUBWF)

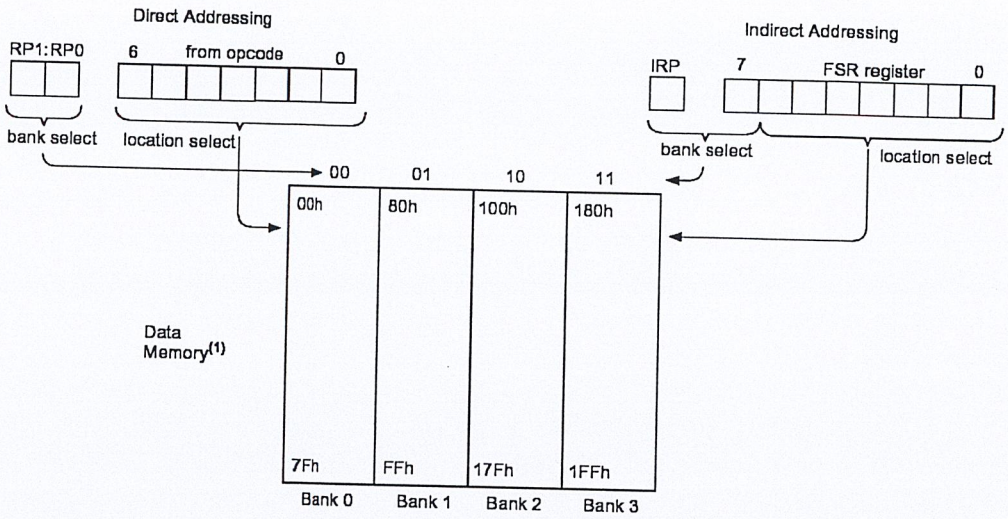
1 = เกิดการทดจาก 4 บิทบน 0 = ไม่เกิดการทดจาก 4 บิทบน

* สำหรับการยืม ผลที่บิทนี้จะกลับกัน

2.3.4 การอ้างอิงแอดเดรสของรีจิสเตอร์โดยอ้อม

การอ้างอิงแอดเดรสของรีจิสเตอร์โดยอ้อมทำได้ โดยใช้รีจิสเตอร์ INDF ซึ่งคำสั่งใดๆ ที่ใช้รีจิสเตอร์ตัวนี้จะเป็นการเข้าถึงรีจิสเตอร์ตัวที่ถูกชี้โดยไฟล์ซีเล็กรีจิสเตอร์ (FSR) ซึ่งในการอ่านค่าจากรีจิสเตอร์ INDF เอง (FSR ชี้ไปที่ตำแหน่ง 00) จะได้ค่า 00h และการเขียนค่าไปที่ตัวมันเอง จะไม่มีอะไรเกิดขึ้น (No Operation)

เนื่องจากรีจิสเตอร์ FSR มีเพียง 8 บิท การอ้างถึงรีจิสเตอร์ได้ทั้งหมดจะต้องใช้อีกหนึ่งบิทจาก บิท IRP (STATUS<7>) ดูภาพที่ 2.6 ประกอบ



ภาพที่ 2.6 การอ้างอิงแอดเดรสของรีจิสเตอร์โดยตรงและโดยอ้อม

2.4 ความสามารถทั่วไปของไมโครคอนโทรลเลอร์

2.4.1 การอินเตอร์รัพท์

PIC16F877 มีการอินเตอร์รัพท์รวม 14 แบบโดยมีบิท GIE (Global Interrupt Enable) ในรีจิสเตอร์ INTCON เป็นตัวควบคุมการอินเตอร์รัพท์ทั้งหมด นอกจากนี้ในรีจิสเตอร์ INTCON ยังมีบิทควบคุมการอินเตอร์รัพท์ของพอร์ท RB0/INT, พอร์ท B และ TMR0 อีกด้วยส่วนอินเตอร์รัพท์อื่น ๆ มีรีจิสเตอร์ PIE1, PIE2, PIR1 และ PIR2 เป็นตัวควบคุม

เมื่อเกิดการอินเตอร์รัพท์ขึ้นบิท GIE จะถูกเคลียร์เพื่อป้องกันการเกิดอินเตอร์รัพท์ซ้อน ค่าในโปรแกรมเคาน์เตอร์ขณะนั้นจะถูกเก็บในแสตกและโปรแกรมเคาน์เตอร์จะชี้ไปที่ตำแหน่ง0004h ซึ่งเป็นตำแหน่งของโปรแกรมการบริการอินเตอร์รัพท์ (Interrupt Service Routine) ด้วยการโพลลิ่ง (Polling) อินเตอร์รัพท์แฟลก จะสามารถตรวจสอบได้ว่าเกิดอินเตอร์รัพท์จากที่ใด

คำสั่ง RETFIE ใช้เพื่อออกจากโปรแกรมบริการอินเตอร์รัพท์ โดยบิท GIE จะถูกเซตโดยอัตโนมัติเพื่ออนุญาตการอินเตอร์รัพท์อีกครั้ง

อินเตอร์รัพท์แฟลกจะต้องถูกเคลียร์โดยโปรแกรมก่อนที่จะออกจากโปรแกรมบริการ หรืออินเตอร์รัพท์ หากไม่มีการเคลียร์อินเตอร์รัพท์แฟลก เมื่อบิท GIE ถูกเซต จะเกิดการเรียกโปรแกรมบริการอินเตอร์รัพท์ซ้ำ

รีจิสเตอร์ควบคุมการอินเตอร์รัพท์ (Interrupt Control Register: INTCON)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	**PEIE**	**T0IE**	**INTE**	**RBIE**	**T0IF**	**INIF**	**RBIF**
บิท7							บิท0

GIE บิทควบคุมการอินเตอร์รัพท์ทั้งหมด

1 = อนุญาตการอินเตอร์รัพท์ทังหมด

0 = ไม่อนุญาตการอินเตอร์รัพท์ทั้งหมด

PEIE บิทควบคุมการอินเตอร์รัพท์ย่อย

1 = อนุญาตการอินเตอร์รัพท์

0 = ไม่อนุญาตการอินเตอร์รัพท์

T0IE บิทควบคุมการอินเตอร์รัพท์เมื่อ TMRO เกิดโอเวอร์โฟลว์

1 = อนุญาตการอินเตอร์รัพท์ของ TMRO

0 = ไม่อนุญาตการอินเตอร์รัพท์ของ TMRO

INTE บิทควบคุมการอินเตอร์รัพท์ของขา RB0/INT

1 = อนุญาตการอินเตอร์รัพท์ของ RB0/INT

0 = ไม่อนุญาตการอินเตอร์รัพท์ของ RB0/INT

RBIE บิทควบคุมการอินเตอร์รัพท์เมื่อพอร์ท B เปลี่ยนสถานะ

1 = อนุญาตการอินเตอร์รัพท์ของพอร์ท B

0 = ไม่อนุญาตการอินเตอร์รัพท์ของพอร์ท B

T0IE บิทแสดง TMRO เกิดโอเวอร์โฟลว์

1 = TMRO เกิดโอเวอร์โฟลว์ (ต้องรีเซตโดยโปรแกรม)

0 = TMRO ไม่เกิดโอเวอร์โฟลว์

INTE บิทแสดงการอินเตอร์รัพท์ของขา RB0/INT

1 = เกิดการอินเตอร์รัพท์ของขา RB0/INT (ต้องรีเซตโดยโปรแกรม)

0 = ไม่เกิดการอินเตอร์รัพท์ของขา RB0/INT

RBIF บิทแสดงการอินเตอร์รัพท์เมื่อพอร์ท B เปลี่ยนสถานะ

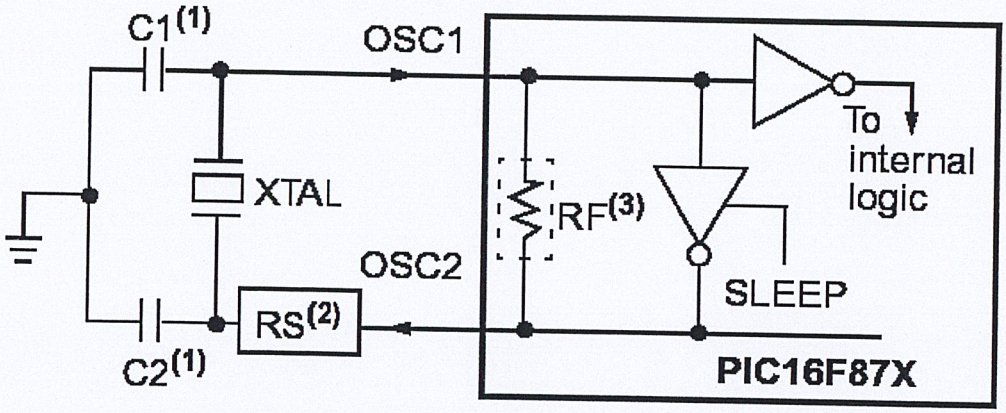
1 = เกิดการอินเตอร์รัพท์ของพอร์ท B (ต้องรีเซตโดยโปรแกรม)

0 = ไม่เกิดการอินเตอร์รัพท์ของพอร์ท B

2.4.2 การใช้งานออสซิลเลเตอร์

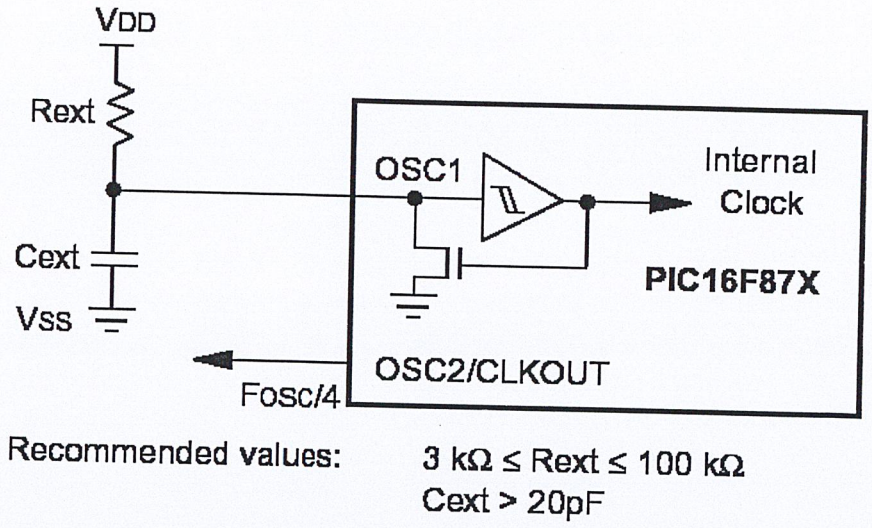
PIC16F877สามารถเลือกทำงานได้กับออสซิลเลเตอร์4 แบบ โดยการโปรแกรมบิท FOSC1 และ FOSC0 ในรีจิสเตอร์ Configuration ออสซิลเลเตอร์ 4 แบบมีดังนี้

- LP คริสตอลพลังงานต่ำ
- XT คริสตอล/เรโซเนเตอร์
- HS คริสตอล/เรโซเนเตอร์ ความเร็วสูง
- RC รีซิสเตอร์ – คาปาซิเตอร์



ภาพที่ 2.7 การต่อคริสตอลออสซิลเลเตอร์

สำหรับคริสตอลออสซิลเลเตอร์ ต่อดังภาพที่ 2.7 และควรเลือกใช้คริสตอลแบบตัดขนาน (Parallel cut crystal) เพื่อให้ได้ความถี่ใกล้เคียงกับที่ระบุไว้มากที่สุด



ภาพที่ 2.8 การต่อรีซิสเตอร์ คาปาซิเตอร์ออสซิลเลเตอร์

การใช้รีซิสเตอร์คาปาซิเตอร์ออสซิลเลเตอร์เหมาะสำหรับงานที่ไม่ต้องการความถี่เที่ยงตรงมากนัก โดยค่าความถี่จะขึ้นอยู่กับแรงดันV_{DD}, อุณหภูมิ, ค่าความต้านทาน และความจุของรีซิสเตอร์และคาปาซิเตอร์ ภาพที่ที่ 2.8 แสดงการต่อรีซิสเตอร์คาปาซิเตอร์ออสซิลเลเตอร์

2.4.3 โหมดประหยัดพลังงาน (Sleep mode)

เมื่อไมโครคอนโทรลเลอร์ทำคำสั่ง SLEEP จะเป็นการเข้าสู่โหมดประหยัดพลังงาน บิทลดพลังงานในรีจิสเตอร์สถานะจะถูกเคลียร์และออสซิลเลเตอร์จะถูกปิดแต่พอร์ทอินพุทเอาท์พุท จะยังคงสถานะเดิมอยู่ (1, 0 หรือ Hi - Impedance)

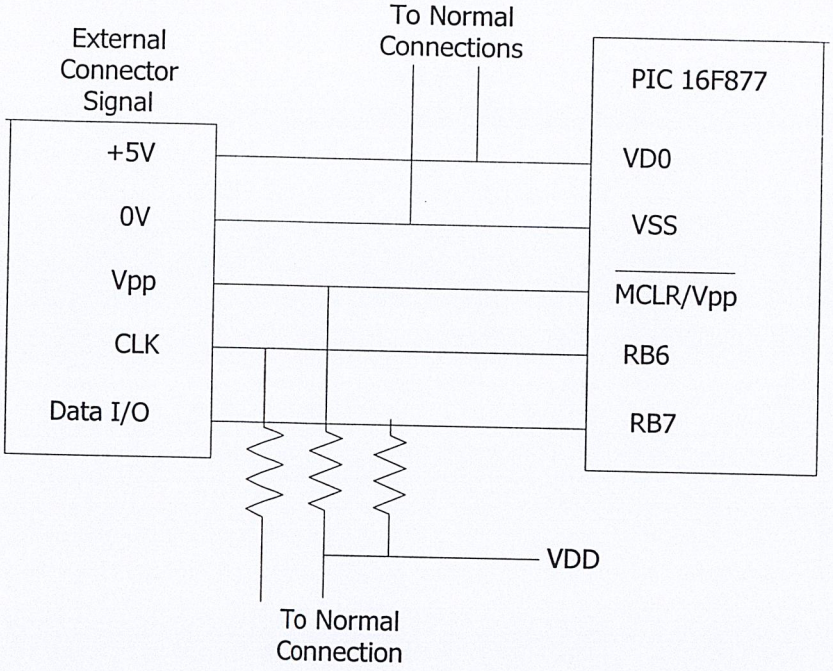
การออกจากโหมดประหยัดพลังงาน จะเกิดขึ้นเมื่อมีเหตุการณ์หนึ่งดังต่อไปนี้

- มีสัญญาณรีเซตที่ขา / MCRL
- การกระตุ้นจากวอชด็อกไทเมอร์
- เกิดการอินเตอร์รัพท์

การเกิดอินเตอร์รัพท์จะทำให้ออกจากโหมดประหยัดพลังงาน โดยไม่สนว่าได้อนุญาตการอินเตอร์รัพท์หรือไม่ และหากไม่อนุญาตการอินเตอร์รัพท์ คำสั่งต่อจาก SLEEP จะถูกประมวลผลต่อไปตามปกติ หากอนุญาตการอินเตอร์รัพท์ คำสั่งต่อจาก SLEEP จะถูกประมวลผล แล้วกระโดดไปยังโปรแกรมบริการอินเตอร์รัพท์

2.4.4 การโปรแกรมไมโครคอนโทรลเลอร์ในระบบ (In-Circuit Serial Programming)

PIC 16F877 สามารถโปรแกรมได้แม้อยู่ในวงจร โดยมีสัญญาณนาฬิกาและสัญญาณข้อมูลไฟเลี้ยง กราวนด์ และแรงดันไฟสูงสำหรับการโปรแกรม ดังภาพที่ 2.9



ภาพที่ 2.9 การต่อไมโครคอนโทรลเลอร์เพื่อทำการโปรแกรม

2.5 ส่วนแปลงสัญญาณอนาลอกเป็นดิจิตอล

ส่วนแปลงสัญญาณอนาลอกเป็นดิจิตอลในไมโครคอนโทรลเลอร์ มีอนาลอกอินพุท 8 ช่อง และสามารถเปลี่ยนการใช้งานเป็นดิจิตอลพอร์ทได้ด้วย ขั้นตอนการทำงานเริ่มที่แรงดันที่อนาลอกอินพุทจะถูกประจุให้ตัวเก็บประจุแซมเปิ้ลแอนด์โฮล (Sample and Hold Capacitor) จากนั้นจะถูกแปลงด้วยวิธี Successive Approximation ที่ความละเอียด 10 บิท แรงดันอ้างอิงทั้งบวกและลบสามารถเลือกใช้จากวงจรภายนอก หรือจาก V_{DD} และ V_{SS}

การแปลงสัญญาณทำได้แม้อยู่ในสภาวะ Sleep mode โดยต้องเลือกสัญญาณนาฬิกาที่ใช้ในการแปลงจากวงจรรีซิสเตอร์คาปาซิเตอร์ออสซิลเลเตอร์ภายใน

2.5.1 รีจิสเตอร์ที่เกี่ยวข้องกับการแปลงสัญญาณ

มีรีจิสเตอร์ 4 ตัวที่เกี่ยวข้องกับการแปลงสัญญาณ ได้แก่

- ADCON0 ควบคุมการทำงาน
- ADCON1 ควบคุมหน้าที่ของพอร์ท
- ADRESH เก็บผลลัพธ์ไบท์สูง
- ADRESL เก็บผลลัพธ์ไบท์ต่ำ

รีจิสเตอร์ ADCON0

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	**ADCS0**	**CHS2**	**CHS1**	**CHS0**	**GO/DONE**	**-**	**ADON**
บิท7							บิท0

ADCS1: ADCS0 บิทเลือกความถี่สัญญาณนาฬิกา

00 = $F_{osc}/2$ 01 = $F_{osc}/8$

10 = $F_{osc}/32$ 11 = F_{RC}

CHS2: CHS0 บิทเลือกช่องสัญญาณอนาลอก

000 = ช่อง 0 (RA0/AN0) 001 = ช่อง 1 (RA1/AN1)

010 = ช่อง 2 (RA2/AN2) 011 = ช่อง 3 (RA3/AN3)

100 = ช่อง 4 (RA5/AN4) 101 = ช่อง 5 (RE0/AN5)

101 = ช่อง 6 (RE1/AN6) 111 = ช่อง 7 (RE2/AN7)

GO/DONE บิทสถานะของส่วนแปลงสัญญาณ

1 = ส่วนแปลงสัญญาณกำลังทำงาน (การเซตบิทนี้เป็นคำสั่งเริ่มการทำงาน)

0 = ไม่มีการทำงาน (ถูกเคลียร์โดยฮาร์ดแวร์)

ADON เปิด/ปิด ส่วนแปลงสัญญาณ

1 = เปิดส่วนแปลงสัญญาณ

0 = ปิดส่วนแปลงสัญญาณ ทำให้ส่วนนี้ไม่มีการใช้พลังงาน

รีจิสเตอร์ ADCON1

R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	**-**	**-**	**-**	**PCFG3**	**PCFG2**	**PCFG1**	**PCFG0**
บิท7							บิท0

ADFM รูปแบบของผลลัพธ์ในรีจิสเตอร์ ADRESH และ ADRESL

1 = จัดชิดขวา โดย 2 บิทบนเก็บใน ADRESH และ 8 บิทล่างเก็บใน ADRESL

0 = จัดชิดซ้าย โดย 8 บิทบนเก็บใน ADRESH และ 2 บิทล่างเก็บใน ADRESL

ตารางที่ 2.2 PCFG3: PCFG0 กำหนดหน้าที่ของพอร์ท

PCFG3 : PCFG0	AN7 RE2	AN6 RE1	AN5 RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	V_{REF+}	V_{REF-}	Chan /Ref
0000	A	A	A	A	A	A	A	A	V_{DD}	V_{SS}	8/0
0001	A	A	A	A	V_{REF+}	A	A	A	RA3	V_{SS}	7/1
0010	D	D	D	A	A	A	A	A	V_{DD}	V_{SS}	5/0
0011	D	D	D	A	V_{REF+}	A	A	A	RA3	V_{SS}	4/1
0100	D	D	D	D	A	D	A	A	V_{DD}	V_{SS}	3/0
0101	D	D	D	D	V_{REF+}	D	A	A	RA3	V_{SS}	2/1
0111	D	D	D	D	D	D	D	D	V_{DD}	V_{SS}	0/0
1000	A	A	A	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	V_{DD}	V_{SS}	6/0
1010	D	D	A	A	V_{REF+}	A	A	A	RA3	V_{SS}	5/1
1011	D	D	A	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2	4/2
1100	D	D	D	A	V_{REF+}	V_{REF-}	A	A	RA3	RA2	3/2
1101	D	D	D	D	V_{REF+}	V_{REF-}	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	V_{DD}	V_{SS}	1/0
1111	D	D	D	D	V_{REF+}	V_{REF-}	D	A	RA3	RA2	1/2

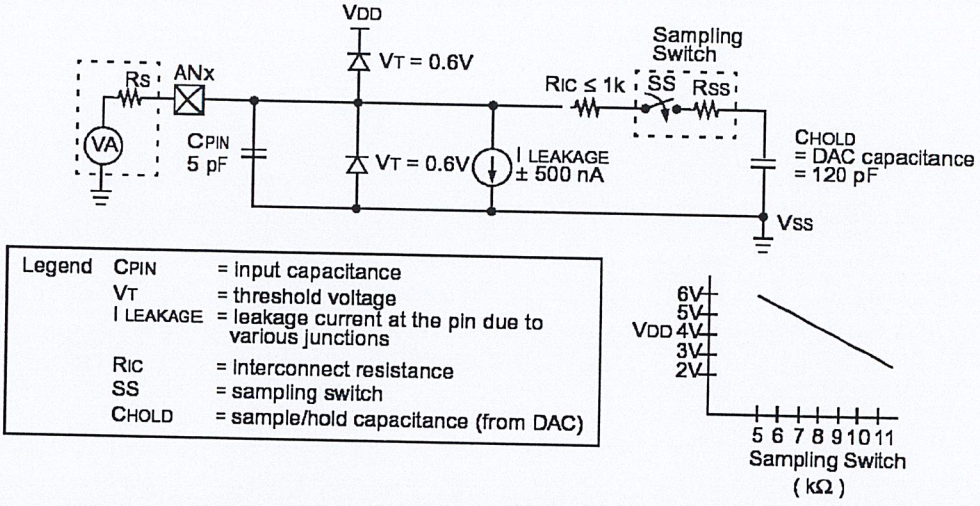
A = ใช้พอร์ทเป็นอนาลอกอินพุท D = ใช้พอร์ทเป็นดิจิตอล

รีจิสเตอร์เก็บผลลัพธ์

รีจิสเตอร์ ADRESH และ ADRESL ใช้เก็บผลลัพธ์ไบท์สูงและไบท์ต่ำที่ได้จากการแปลงสัญญาณอนาลอกเป็นดิจิตอลขนาด 10 บิท แต่ขนาดรวมของรีจิสเตอร์สองตัวนี้เป็น 16 บิท ทำให้ผู้ใช้สามารถเลือกรูปแบบการเก็บข้อมูลในรีจิสเตอร์ให้จัดชิดซ้ายหรือชิดขวาได้โดยการกำหนดบิท ADFM ในรีจิสเตอร์ ADCON1

2.5.2 ข้อกำหนดในการใช้งาน

เพื่อให้การแปลงสัญญาณได้ความแม่นยำมากที่สุด จะต้องออกแบบให้มีเวลาในการประจุคาปาซิเตอร์ (C_{HOLD}) เพียงพอ เวลาที่ใช้ขึ้นอยู่กับค่าของทรานซิสเตอร์สวิทช์ภายในโมดูล (R_{SS}) และค่าอิมพีแดนซ์ของแหล่งกำเนิดสัญญาณ (R_{SS}) โดยค่าอิมพีแดนซ์ของแหล่งกำเนิดสัญญาณไม่ควรมากเกิน 10 กิโลโอห์ม ดูภาพที่ 2.10 ประกอบ



ภาพที่ 2.10 วงจรเสมือนแสดงการประจุคาปาซิเตอร์ C_{HOLD}

บทที่ 3
การสื่อสารแบบอนุกรม

การสื่อสารมีด้วยกัน 2 รูปแบบคือ การสื่อสารแบบขนานและการสื่อสารแบบอนุกรม การสื่อสารแบบขนาน เป็นการรับหรือส่งข้อมูลคราวละมากกว่า 1 บิทในเวลาเดียวกัน ทำให้การรับและส่งข้อมูลมีความเร็วสูง แต่จำนวนของสายสัญญาณที่ใช้ในการส่งผ่านข้อมูลต้องมีมากตามจำนวนบิทของข้อมูลที่ทำการส่ง นอกจากนั้นยังมีสายที่ใช้สำหรับควบคุมและตรวจสอบการรับส่งข้อมูลด้วย ซึ่งอาจต้องใช้สายมากเป็น 2 เท่าของจำนวนบิทข้อมูลก็ได้ ในขณะที่การรับส่งข้อมูลแบบอนุกรมจะเป็นการรับส่งข้อมูลครั้งละ1 บิทการรับส่งข้อมูลแบบอนุกรมจึงมีข้อดีในเรื่องของจำนวนสายสัญญาณที่น้อยมากและไม่แปรตามจำนวนบิทของข้อมูลแต่ความเร็วในการสื่อสารจะลดลงและโปรแกรมควบคุมจะมีความซับซ้อนมากกว่า

การสื่อสารแบบอนุกรมยังสามารถแบ่งได้เป็น 2 แบบคือการสื่อสารอนุกรมแบบซิงโครนัส (Synchronous)และการสื่อสารอนุกรมแบบอะซิงโครนัส(Asynchronous)การสื่อสารแบบซิงโครนัสจะมีสัญญาณนาฬิการ่วมกับการรับและส่งสัญญาณด้วยซึ่ง ตัวอย่างการส่งข้อมูลแบบซิงโครนัสคือคีย์บอร์ดของคอมพิวเตอร์หรือบัสแบบ I^2C ซึ่งสายเส้นหนึ่งเป็นสายของสัญญาณนาฬิกาส่วนสายอีกเส้นเป็นสายของข้อมูล ดังนั้นการติดต่อกันแบบซิงโครนัสนี้จะต้องใช้สายในการเชื่อมต่ออย่างน้อยที่สุด 3 เส้นคือ สัญญาณนาฬิกา, ข้อมูลและกราวนด์

การสื่อสารอนุกรมแบบอะซิงโคนัสรับและส่งข้อมูล โดยไม่จำเป็นต้องมีสัญญาณนาฬิการ่วมด้วย แต่จะใช้การกำหนดค่าอัตราความเร็วในการรับและส่งข้อมูลให้มีค่าเท่ากัน ซึ่งเรียกอัตราความเร็วนี้ว่า อัตราบอดหรือบอดเรท (Baud rate) มีหน่วยเป็น บิทต่อวินาที (Bit Per Second: Bps)

3.1 การสื่อสารอนุกรมแบบ RS-232

สมาคมอุตสาหกรรมอิเล็กทรอนิกส์ (Electronic Industries Association: EIA) ได้วางมาตรฐานที่มีชื่อว่าEIA RS-232 เป็นมาตรฐานสื่อสารอนุกรมแบบอะซิงโครนัส 2 ทิศทาง มาตรฐานนี้ในช่วงแรกจะใช้คอนเน็กเตอร์เป็นแบบ DB-25 โดยกำหนดความยาวสูงสุดของสายสัญญาณไว้ที่ 50 ฟุต ระดับสัญญาณตั้งแต่ –3 โวลท์จนถึง –12 โวลท์แสดงว่ามีข้อมูล (Mark) หรือเทียบเท่าลอจิกหนึ่ง และ +3 โวลท์ ถึง +12 โวลท์ แสดงว่าเป็นช่องว่าง (Space) หรือลอจิกศูนย์

มาตรฐาน RS-232 กำหนดรูปแบบการสื่อสารข้อมูลกันระหว่างอุปกรณ์เชื่อมต่อข้อมูล (Data Terminal Equipment: DTE) กับวงจรข้อมูลปลายทาง (Data Circuit Terminating: DEC) อุปกรณ์ DTEนั้นจะต้องเป็นอุปกรณ์ที่มีการประมวลผลในตัวอย่างเช่นไ มโครคอนโทรลเลอร์ หรือ

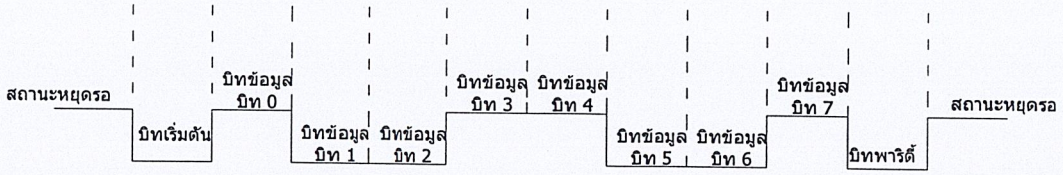
ไมโครคอมพิวเตอร์ ซึ่งมีความสามารถในการสร้างบิทข้อมูลแบบอนุกรมได้และใน ส่วนอุปกรณ์ DEC ทำหน้าที่เป็นเพียงตัวรับข้อมูลที่ส่งมาจาก DTE เท่านั้น

ปัจจุบันมาตรฐานนี้ได้พัฒนาถึงเวอร์ชัน C สำหรับการใช้งานในคอมพิวเตอร์โดยส่วนใหญ่ พอร์ทอนุกรม RS-232 ถูกใช้เพื่อเชื่อมต่อกับโมเด็ม (Modem) หรือเม้าส์ (Mouse)

3.1.1 รูปแบบการส่งข้อมูลแบบอะซิงโครนัส

สัญญาณที่ใช้ในการรับส่งข้อมูลแบบอะซิงโครนัส ประกอบด้วย 4 ส่วนด้วยกันคือ

1. บิทเริ่มต้น (Start bit) 1 บิท
2. บิทข้อมูลแบบอนุกรม 5, 6, 7 หรือ 8 บิท
3. บิทตรวจสอบพาริตี้ (Parity bit) 1 บิทหรือไม่มี
4. บิทปิดท้ายหรือบิทหยุด (Stop bit) 1, 1.5 หรือ 2 บิท



ภาพที่ 3.1 รูปแบบของข้อมูลอนุกรมแบบอะซิงโครนัส

ภาพที่ 3.1 แสดงรูปแบบการส่งข้อมูลอนุกรมแบบอะซิงโครนัส เมื่อไม่มีการส่งข้อมูลที่ขาข้อมูลจะมีสถานะลอจิกหนึ่งเรียกสถานะนี้ว่าสถานะหยุดรอ (Waiting stage) การเริ่มต้นส่งข้อมูลจะเริ่มจากการให้ขาข้อมูลมีลอจิกศูนย์ด้วยช่วงระยะเวลา 1 บิท เรียกบิทนี้ว่า บิทเริ่มต้น (Start bit) จากนั้นบิทข้อมูลจะถูกส่งออกไปโดยเริ่มจากบิทที่มีนัยสำคัญต่ำสุด (Least Significant Bit: LSB) ก่อน ซึ่งข้อมูลที่ต้องการส่งอาจมีจำนวน 5,6,7 หรือ 8 บิทก็ได้โดยต้องกำหนดให้ตรงกัน จากนั้นตามด้วยบิทพาริตี้ (Parity bit) ซึ่งใช้ในการตรวจสอบความผิดพลาดที่เกิดขึ้นจากกรส่งข้อมูล บิทสุดท้ายที่จะส่งคือ บิทปิดท้ายหรือบิทหยุด (Stop bit) โดยจะเป็นการทำให้ขาข้อมูลมีสถานะลอจิกหนึ่งอีกครั้งด้วยระยะเวลา 1 บิท, 1.5 บิทหรือ 2 บิท เพื่อเป็นการแสดงว่าสิ้นสุดไบท์ข้อมูลแล้ว

การตรวจสอบพาริตี้สามารถกำหนดให้เป็นพาริตี้คี่ (Odd Parity), พาริตี้คู่ (Even Parity) หรือไม่มีการตรวจสอบพาริตี้ก็ได้ พาริตี้คี่หรือพาริตี้คู่ที่แสดงถึงจำนวนลอจิกหนึ่งทั้งหมดถ้าภายในข้อมูลที่ส่งไป1ไบท์รวมบิทพาริตี้ว่ามีจำนวนเป็นเลขคู่หรือคี่ ยกตัวอย่างข้อมูลที่จะทำการส่งมีขนาด 8 บิท มีค่า 10011001 จะเห็นว่าข้อมูลในไบท์นี้มีจำนวนลอจิกหนึ่งจำนวน 4 ตัวซึ่งเป็นเลขคู่ ดังนั้น

ถ้ากำหนดค่าเป็นพาริตี้คู่ ค่าของบิทพาริตี้จะต้องมีลอจิกเป็นศูนย์ แต่ถ้ากำหนดเป็นพาริตี้คี่ค่าของบิทพาริตี้จะต้องเป็นหนึ่ง เพื่อให้จำนวนบิทที่เป็นหนึ่งรวมบิทพาริตี้มีค่าเป็นคี่

บิทพาริตี้ถูกสร้างขึ้นจากภาคส่งข้อมูล ซึ่งทางภาครับต้องกำหนดคุณสมบัติการตรวจสอบพาริตี้ที่ตรงกันไว้ด้วย หากเกิดความผิดพลาดในการส่งสัญญาณทางภาครับจะแสดงข้อผิดพลาดให้ผู้ใช้ทราบ กระบวนการดังกล่าวเป็นวิธีการตรวจสอบความผิดพลาดที่เกิดขึ้นในการรับส่งข้อมูลที่ง่ายที่สุด แต่สามารถตรวจสอบได้เมื่อบิทของข้อมูลที่ทำการรับส่งผิดพลาดเพียงบิทเดียวเท่านั้น ถ้าข้อมูลที่ทำการส่งมีบิทที่ผิดพลาดมากกว่า 1 บิทการตรวจสอบด้วยวิธีนี้จะไม่ได้ผล สำหรับการตั้งพาริตี้เป็น None นั้นทั้งภาครับและภาคส่ง จะไม่มีการตรวจสอบพาริตี้

3.1.2 อัตราความเร็วในการรับส่งข้อมูลแบบอะซิงโครนัส

อัตราความเร็วในการรับส่งข้อมูลของการรับส่งข้อมูลแบบอะซิงโครนัสหรือ อัตราบอดหรือบอดเรทที่ใช้สำหรับพอร์ทอนุกรม RS-232 มีหลายค่า ได้แก่ 110, 150, 300, 600, 1200, 2400, 4800, 9600 และ 19200 บิทต่อวินาที โดยมีค่าเพิ่มมากขึ้นตามเทคโนโลยีของคอมพิวเตอร์เนื่องจากบอดเรทคือค่าของจำนวนบิทที่สามารถส่งได้ใน 1 วินาที สมมติว่า ข้อมูลอนุกรมมีขนาด 8 บิท ไม่มีการตรวจสอบพาริตี้ มีบิทเริ่มต้น 1 บิท และบิทปิดท้าย 1 บิท ความยาวของข้อมูล 1 ไบท์จะเท่ากับ 10 บิท ถ้าใช้บอดเรทในการส่งข้อมูลเท่ากับ 9,600 บิทต่อวินาที ก็จะสามารถรับส่งข้อมูลได้ด้วยความเร็ว 960 ไบท์ต่อวินาที

3.1.3 คอนเน็กเตอร์สำหรับพอร์ท RS-232 และการเชื่อมต่อ

มาตรฐานการเชื่อมต่อแบบ RS-232 จะใช้คอนเน็กเตอร์แบบ DB-25 หรือ DB-9 ซึ่งคอนเน็กเตอร์แบบ DB-25 จะมีขาต่อใช้งานเพียง 9 เส้นเช่นเดียวกับ คอนเน็กเตอร์แบบ DB-9 เนื่องจากขาอื่น ๆที่เคยมีการใช้งานในอดีตไม่ค่อยสำคัญมากนักจึงถูกยกเลิกไปโดยประกอบไปด้วยขาต่างๆดังนี้

Receive Data: RD หรือ RxD ขานี้ใช้เพื่อรับข้อมูบอนุกรมเข้ามายังคอมพิวเตอร์โดยจะนำข้อมูลที่อ่านได้ไปเก็บไว้ในรีจิสเตอร์บัฟเฟอร์

Transmitted Data: TD หรือ TxD ขานี้ใช้เพื่อส่งข้อมูลอนุกรมออกจาก คอมพิวเตอร์โดยนำข้อมูลที่เก็บอยู่ในบัฟเฟอร์สำหรับส่งข้อมูลออกไป

Signal Ground: GND เป็นขากราวนด์ของสัญญาณ

Data Terminal Ready: DTR เป็นขาที่ใช้สำหรับส่งสัญญาณออกไปจากคอมพิวเตอร์เพื่อให้อุปกรณ์ปลายทางรับรู้ว่าต้องการที่จะติดต่อกับอุปกรณ์ปลายทาง โดยขา DTR นี้จะต้องเชื่อมต่อกับ

ขา DSR ของอุปกรณ์ปลายทางและขา DTR ของอุปกรณ์ปลายทางจะต้องเชื่อมกับขา DSR ของคอมพิวเตอร์ และถ้าใช้การเชื่อมต่อแบบ 3 สาย ต้องเชื่อมต่อขา DTR และ DSR ของพอร์ทอนุกรมเข้าด้วยกันและต้องเชื่อมเข้ากับขา DCDให้ถึงกันด้วยในกรณีที่โปรแกรมสื่อสารที่ใช้มีการตรวจจับสัญญาณพาห์

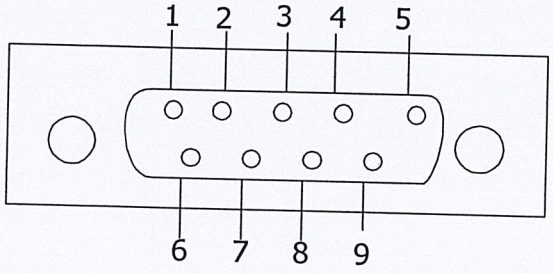
Data Set Ready: DSRขานี้จะใช้ควบคู่กับขา DTR เพื่อใช้สำหรับตรวจสอบการเชื่อมต่อกันระหว่างคอมพิวเตอร์กับอุปกรณ์ปลายทาง ซึ่งขา DSR นี้จะเป็นขาสำหรับรับข้อมูลจากภายนอก

Request To Send: RTS เป็นขาเอาท์พุท สำหรับส่งสัญญาณร้องขอให้อุปกรณ์ปลายทางส่งข้อมูลมาให้คอมพิวเตอร์โดยขาที่รับสัญญาณRTSก็คือขาCTSซึ่งในกรณีที่มีการเชื่อมต่อแบบ 3 สาย จะต้องเชื่อมต่อขา RTS และ CTS เข้าด้วยกัน เพื่อให้การรับและส่งข้อมูลสามารถเกิดขึ้นได้ตลอดเวลา

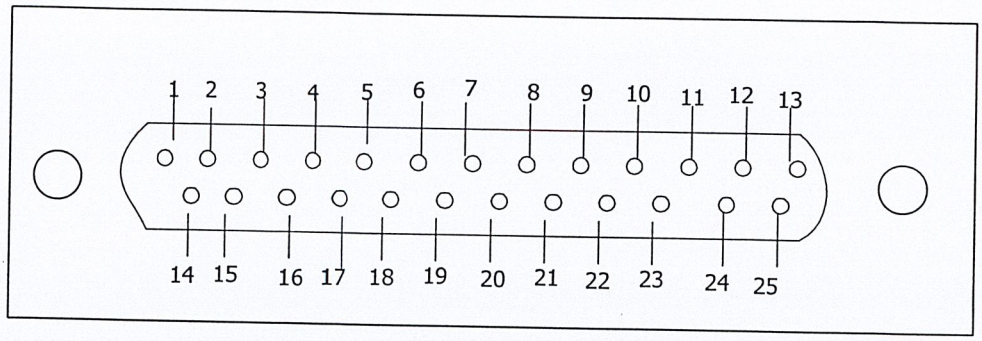
Clear To Send: CTS จะเป็นขาอินพุทที่ทำหน้าที่รอรับสัญญาณที่ส่งถูกเข้ามา เมื่อมีการส่งสัญญาณเข้ามาที่ขานี้ ข้อมูลจะถูกส่งออกไปที่ขา TxD ขานี้จะใช้เพื่อตรวจสอบอุปกรณ์ต่อพ่วงว่าพร้อมที่จะรับข้อมูลแล้วหรือยัง

Data Carrier Detec: DCD หรือ Carrier Detect: CD ขานี้จะแอกทีฟเมื่อได้รับสัญญาณพาห์จากอุปกรณ์สื่อสารข้อมูลเช่น โมเด็มสำหรับการใช้งานปกติขานี้ไม่ได้ถูกใช้งานมากนัก

Ring Indicator: RI ใช้แสดงสถานะสัญญาณเรียกจากสายโทรศัพท์ ซึ่งปกติในการสื่อสารโดยทั่วไปสายนี้จะไม่ถูกใช้งาน จะใช้งานก็ต่อเมื่อมีการเชื่อมต่อกับโมเด็มแล้วต้องการตรวจสอบสัญญาณเรียกจากสายโทรศัพท์ ดูลักษณะของคอนเน็กเตอร์ได้ในภาพที่ 3.2 (ก)และ3.2 (ข)



ภาพที่ 3.2(ก) คอนเน็กเตอร์อนุกรม 9 ขาหรือแบบ DB 9



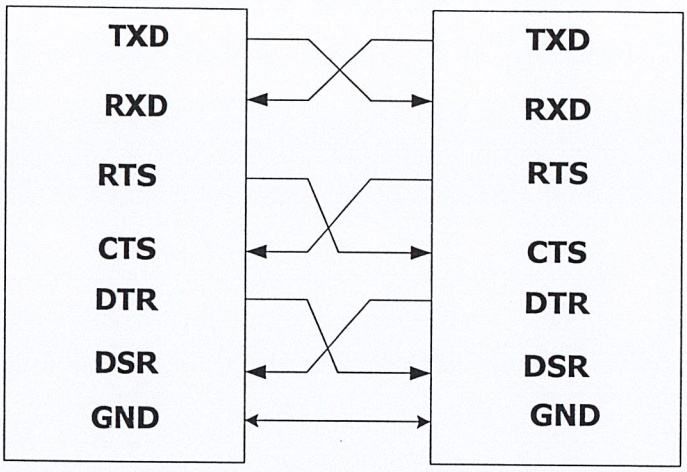
ภาพที่ 3.2 (ข) คอนเน็กเตอร์อนุกรม 25 ขาหรือแบบ DB 25

ภาพที่ 3.2 แสดงการจัดขาสัญญาณของพอร์ทอนุกรม และหน้าที่การทำงาน

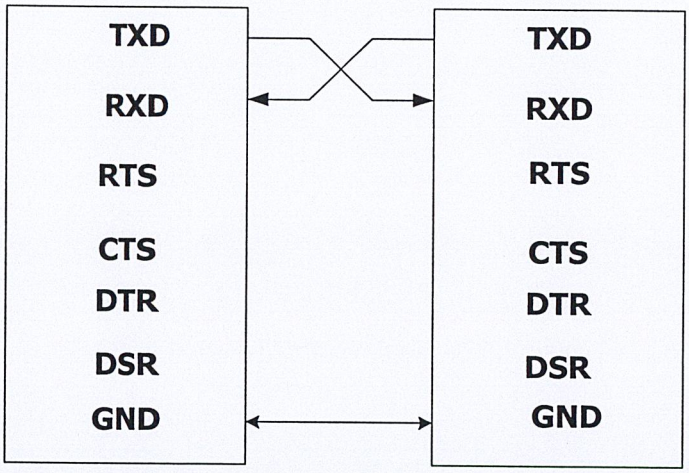
ตารางที่ 3.1 รายละเอียดของขาคอนเน็กเตอร์

คอนเน็กเตอร์ DB-9	คอนเน็กเตอร์ DB-25	ชื่อของสายสัญญาณ	ชนิดของสายสัญญาณ
1	8	Data Camer Detect: DCD	อินพุท
2	3	Received Data: RxD	อินพุท
3	2	Transmited Data: Txd	เอาท์พุท
4	20	Data Terminal Ready: DTR	เอาท์พุท
5	7	Signal Ground: GND	-
6	6	Data Set Ready: DST	อินพุท
7	4	Request To Send: RTS	เอาท์พุท
8	5	Clear To Send: CTS	อินพุท
9	22	Ring Indicator: RI	อินพุท

สำหรับการเชื่อมต่อสายระหว่างคอมพิวเตอร์กับอุปกรณ์ภายนอกโดยแสดงดังในภาพที่ 3.3 ลูกศรในรูปแสดงถึงทิศทางของข้อมูล และการเชื่อมต่อในภาพที่ 3.3 (ก) จะเป็นการเชื่อมต่อแบบ Null Modem ส่วนการเชื่อมต่อในภาพที่3.3 (ข) เป็นการเชื่อมต่อโดยใช้สายสัญญาณน้อยที่สุดเพียง 3 เส้น โดยเส้นหนึ่งสำหรับส่งข้อมูล อีกเส้นสำหรับรับข้อมูล และเส้นสุดท้ายเป็นกราวนด์



ภาพที่ 3.3 (ก) การเชื่อมต่อแบบ Null Modem



ภาพที่3.3 (ข) การเชื่อมต่อโดยใช้สายสัญญาณน้อยที่สุดเพียง3 เส้น

3.1.4 Universal Asynchronous Receiver Transmitter

Universal Asynchronous Receiver Transmitter หรือ UART หมายถึงอุปกรณ์ที่ทำหน้าที่รับและส่งข้อมูลแบบอะซิงโครนัส หน้าที่หลักของ UART คือแปลงสัญญาณข้อมูลแบบขนานที่ส่งมาจากหน่วยประมวลผลกลางให้เป็นสัญญาณอนุกรมแบบอะซิงโครนัส แล้วทำการส่งออกไป และแปลงสัญญาณอนุกรมแบบอะซิงโครนัสที่ป้อนเข้ามายัง UART ให้เป็นแบบขนานก่อนที่จะส่งเข้าสู่หน่วยประมวลผลกลาง ซึ่งนอกจาก UART จะทำหน้าที่แปลงรูปของข้อมูลแล้ว ยังแจ้งรายละเอียดอื่น ๆ ของข้อมูลให้คอมพิวเตอร์รับทราบด้วย อาทิ อัตราเร็วในการรับส่งหรือบอดเรท, รูปแบบการส่งข้อมูล, ความผิดพลาดที่เกิดขึ้นระหว่างการส่งข้อมูล

ภายในUARTจะมีวงจรสร้างบอดเรทที่โปรแกรมได้ (Programmable Baudrate Generator) โดยกำหนดค่าตัวหารให้กับสัญญาณนาฬิกาของUARTโดยตัวหารนี้จะมีขนาด16 บิททำให้สามารถกำหนดตัวหารอยู่ในช่วง 1 ถึง 65,535

ในเครื่องคอมพิวเตอร์ทั่วไปจะมี UART ที่ใช้งานกันอยู่ 2 เบอร์คือ เบอร์ 8250 และ 16550 สำหรับ UART เบอร์ 8250 เป็น UART มาตรฐานที่มีใช้ในคอมพิวเตอร์รุ่น XT โดย UART เบอร์นี้มีบัฟเฟอร์สำหรับรับและส่งข้อมูลเป็นตำแหน่งเดียวกันทำให้การรับและส่งข้อมูลถูกจำกัดความเร็วอยู่ที่ 57.6 กิโลบิทต่อวินาที สำหรับ UART เบอร์ 16550 ถูกใช้ในคอมพิวเตอร์รุ่น AT จะเพิ่มส่วนของชิฟท์รีจิเตอร์แบบ FIFO (First In First Out) ขนาด 16 ไบท์เข้าไป ทำให้สามารถสนับสนุนความเร็วในการรับส่งข้อมูลที่ระดับ 256 กิโลบิทต่อวินาทีได้

ไอซี UART เหล่านี้จะมีระดับแรงดันของลอจิกเป็นแบบทีทีแอล (TTL) แต่เพื่อให้การรับส่งข้อมูลสามารถทำให้ได้ที่ระยะทางมากขึ้น ระดับแรงดันทีทีแอลจะถูกแปลงเป็นระดับแรงดันที่สูงขึ้นโดยลอจิกศูนย์จะมีระดับแรงดัน -3 โวลท์ ถึง -12-โวลท์และลอจิกหนึ่งมีระดับแรงดัน+3 โวลท์จนถึง +12 โวลท์

3.1.5 ลักษณะสัญญาณอินพุทและเอาท์พุทของพอร์ท RS-232

สัญญาณเอาท์พุท RTS และ DTR รวมทั้งสัญญาณแสดงสถานะอินพุท CTS, DSR และ DCD จะถูกกลับสถานะภายในตัว UART ส่วนสัญญาณข้อมูลทั้ง RxD และ TxD จะไม่ถูกกลับสถานะ UART จะให้ระดับสัญญาณเอาท์พุทออกมาเป็นแบบทีทีแอล ดังนั้นสัญญาณที่ส่งออกมาจาก UART จะเข้าสู่วงจรขับ เพื่อแปลงระดับสัญญาณให้เป็นไปตามมาตรฐาน RS-232 ก่อนส่งออกจากคอมพิวเตอร์ สำหรับอุปกรณ์ต่อเชื่อมปลายทางก็จะต้องมีวงจรขับในลักษณะนี้เช่นเดียวกันเพื่อให้ได้สัญญาณในระดับเดียวกันได้ แต่สำหรับวงจรขับที่ใช้ทั้งภายในคอมพิวเตอร์และอุปกรณ์ต่อเชื่อมปลายทางนั้นจะกลับสถานะสัญญาณ

3.1.6 แอดเดรสของพอร์ทอนุกรม

แอดเดรสฐาน (Base) ของพอร์ทอนุกรมในคอมพิวเตอร์มี 4 ตำแหน่ง เมื่อเริ่มเปิดเครื่องเพื่อใช้งานคอมพิวเตอร์ไบออส (BIOS) จะทำการตรวจสอบแอดเดรสของพอร์ทอนุกรมทั้งหมด ถ้าไบออสตรวจพบแอดเดรสของพอร์ทอนุกรม จากนั้นไบออสจะนำแอดเดรสที่ตรวจพบไปเก็บไว้ในหน่วยความจำขนาด 2 ไบท์ นอกจากนั้นบิทที่ 3 ถึง 1 ของหน่วยความจำตำแหน่ง 0000:041H ยังใช้เพื่อแสดงจำนวนของพอร์ทอนุกรมที่มีอยู่ในคอมพิวเตอร์อีกด้วย

ตารางที่ 3.2 แอดเดรสฐานและหน่วยความจำที่เก็บแอดเดรสฐานของพอร์ทอนุกรม

พอร์ท	แอดเดรสฐาน	หน่วยความจำที่เก็บแอดเดรสฐาน
COM1	3F8H	0000:0400H ถึง 0000:0401H
COM2	2F8H	0000:0402H ถึง 0000:0403H
COM3	3E8H	0000:0404H ถึง 0000:0405H
COM4	2E8H	0000:0406H ถึง 0000:0407H

3.1.7 รีจิสเตอร์ของพอร์ทอนุกรม

พอร์ทอนุกรมมีรีจิสเตอร์ขนาด 8 บิท 8 ตัวที่ใช้งานร่วมกับ UART แอดเดรสของรีจิสเตอร์ภายใน พอร์ทอนุกรมสามารถคำนวณได้จากแอดเดรสฐานของพอร์ทอนุกรม ยกตัวอย่างเช่น พอร์ทอนุกรม COM1 มีแอดเดรสฐานอยู่ที่ 3F8H แอดเดรสของรีจิสเตอร์ต่างๆ จะเป็นตำแหน่งที่บวกเข้าไปกับค่า 3F8H โดยรีจิสเตอร์ที่ใช้งานกับพอร์ทอนุกรมดังนี้

รีจิสเตอร์บัฟเฟอร์ 00H

เป็นรีจิสเตอร์สำหรับเก็บข้อมูลที่รับเข้ามา หรือพักข้อมูลก่อนที่จะส่งออกไปมีแอดเดรสอยู่ที่ Base + 00H ถ้าเป็น COM1จะมีแอดเดรสอยู่ที่ 3F8H การติดต่อกับรีจิสเตอร์นี้เพื่อส่งข้อมูลจะต้องกำหนดให้บิทDLABในรีจิสเตอร์กำหนดรูปแบบข้อมูล (03H) มีสถานะเป็นศูนย์เมื่อเขียนข้อมูลไปยังแอดเดรสนี้ ข้อมูลจะถูกส่งออกไปแบบอนุกรม ในกรณีรับข้อมูล เมื่อ UART รับข้อมูลเข้ามาและแปลงเป็นแบบขนานแล้วข้อมูลแบบขนานจะถูกส่งไปยัง รีจิสเตอร์บัฟเฟอร์ หลังจากมีการอ่านค่าจากรีจิสเตอร์นี้แล้ว รีจิสเตอร์นี้จะถูกเคลียร์ เพื่อให้พร้อมสำหรับการรับข้อมูลในไบท์ต่อๆ ไป

รีจิสเตอร์ 01H

เป็นรีจิสเตอร์อนุญาตการอินเตอร์รัพท์ ซึ่งใช้เพื่อเซตโหมดในการอินเตอร์รัพท์ของพอร์ทอนุกรมซึ่งเป็นการกำหนดให้ UART สร้างสัญญาณอินเตอร์รัพท์ขึ้นมามีแอดเดรสอยู่ที่ Base + 01H ถ้าเป็น COM1 จะมีแอดเดรสอยู่ที่ 3F9H รายละเอียดในแต่ละบิทของรีจิสเตอร์ 01H มีดังนี้

-	-	-	-	**SINP**	**ERBK**	**TBE**	**RxRD**

บิท7 บิท0

SINP 1 = อนุญาตการอินเตอร์รัพท์เมื่อมีการเปลี่ยนสถานะที่ขาอินพุท CTS, DSR, DCD หรือขา RI

0 = ไม่มีการใช้อินเตอร์รัพท์นี้

ERBK 1 = อนุญาตการอินเตอร์รัพท์เมื่อเกิดความผิดพลาดขึ้นจากบิทพาริตี้, โอเวอร์รันที่เฟรมข้อมูล หรือการหยุดข้อมูล

0 = ไม่มีการใช้อินเตอร์รัพท์นี้

TNE 1 = อนุญาตการอินเตอร์รัพท์เมื่อรีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูลว่าง

0 = ไม่มีการใช้อินเตอร์รัพท์นี้

RxRD 1 = อนุญาตการอินเตอร์รัพท์เมื่อข้อมูล 1 ไบท์ถูกเก็บลงในรีจิสเตอร์บัฟเฟอร์แล้ว

0 = ไม่มีการใช้อินเตอร์รัพท์นี้

รีจิสเตอร์ 02H

เป็นรีจิสเตอร์แสดงโหมดการอินเตอร์รัพท์ ใช้เพื่อตรวจสอบโหมดของการอินเตอร์รัพท์เมื่อมีการอินเตอร์รัพท์เกิดขึ้น มีแอดเดรสอยู่ที่ Base + 02H โดยมีรายละเอียดของแต่ละบิทดังนี้

-	-	-	-	-	**ID1**	**ID0**	**PND**

บิท7 บิท0

ID1:ID0 00 = เกิดการอินเตอร์รัพท์เนื่องจากการเปลี่ยนแปลงของขาอินพุท (มีนัยสำคัญเป็นอันดับ 4 หรือนัยสำคัญต่ำสุด)

01 = เกิดการอินเตอร์รัพท์เนื่องจากรีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูลที่ส่งไป นั้น (มีนัยสำคัญอันดับ 3)

10 = เกิดการอินเตอร์รัพท์เนื่องจากข้อมูลจะถูกเก็บลงภายในตัวรีจิสเตอร์บัฟเฟอร์สำหรับรับข้อมูลแล้ว (มีนัยสำคัญอันดับ 2)

11 = เกิดการอินเตอร์รัพท์เนื่องจากความผิดพลาดในการที่ส่งข้อมูลออก หรือเกิดการหยุดกะทันหัน (มีนัยสำคัญเป็นอันดับ 1 หรือนัยสำคัญสูงสุด)

เมื่อมีการอินเตอร์รัพท์ขึ้นนั้นจะต้องมีการเคลียร์ค่าก่อนที่จะให้เกิดอินเตอร์รัพท์ครั้งต่อไป โดยสามารถทำได้ดังนี้

1. กรณีเกิดอินเตอร์รัพท์เนื่องจากการเปลี่ยนแปลงของขาอินพุท (ID1:ID0 = 0:0) จะต้องอ่านค่าจากรีจิสเตอร์ 06H เพื่อเคลียร์ค่าการอินเตอร์รัพท์

2. กรณีเกิดอินเตอร์รัพท์เนื่องจากบัฟเฟอร์สำหรับส่งข้อมูลว่าง (ID1:ID0 = 0:1) จะต้องเขียนข้อมูลไปยังรีจิสเตอร์บัฟเฟอร์ส่งข้อมูล (รีจิสเตอร์ 00H) หรืออ่านค่ารีจิสเตอร์แสดงสถานะ

อินเตอร์รัพท์ (รีจิสเตอร์ 02H) เพื่อเคลียร์ค่าการอินเตอร์รัพท์

3. กรณีเกิดอินเตอร์เนื่องจากเก็บข้อมูลลงในรีจิสเตอร์บัฟเฟอร์ สำหรับรับข้อมูลเรียบร้อย สามารถเคลียร์ค่าของอินเตอร์รัพท์ หรือโดยการอ่านข้อมูลจากรีจิสเตอร์บัฟเฟอร์สำหรับเก็บข้อมูล (รีจิสเตอร์ 00H)

4. กรณีเกิดอินเตอร์รัพท์จากความผิดพลาดในการรับส่งข้อมูล หรือเกิดการหยุดกะทันหัน จะต้องเคลียร์ค่าของอินเตอร์รัพท์ โดยการที่อ่านค่าจากรีจิสเตอร์แสดงสถานะการรับและส่งข้อมูลแบบอนุกรม (รีจิสเตอร์ 05H)

PND 1 = ไม่มีการอินเตอร์รัพท์ 0 = มีการอินเตอร์รัพท์เกิดขึ้น

รีจิสเตอร์ 03H

เป็นรีจิสเตอร์กำหนดรูปแบบของข้อมูล มีรายละเอียดของแต่ละบิทดังนี้

DLAB	BRK	PAR2	PAR1	PAR0	STOP	DAB1	DAB0
บิท7							บิท0

DLAB 1 = เข้าสู่โหมดการหารค่าบอดเรท

0 = เป็นการเข้าถึงรีจิสเตอร์บัฟเฟอร์ (รีจิสเตอร์ 00H) และรีจิสเตอร์ตัวที่ใช้สำหรับการอินเตอร์รัพท์ (รีจิสเตอร์ 01H) เมื่อบิท DLAB มีสถานะเป็น 1 รีจิสเตอร์บัฟเฟอร์ และรีจิสเตอร์สำหรับการอินเตอร์รัพท์จะถูกใช้สำหรับการโหลดค่าการหารความถี่สำหรับการกำหนดค่าบอดเรท รีจิสเตอร์ 00H ใช้ในการกำหนดค่าการหารในไบท์ต่ำ ส่วนรีจิสเตอร์ 01H นั้น ใช้ในการกำหนดค่าการหารในไบท์สูง โดยค่าบอดเรทเท่ากับ 115200 หารด้วยค่าตัวหาร 16 บิท (ค่า 115200 มาจากการหารคริสตอลความถี่ 1.8432 เมกกะเฮิร์ทซ์ที่อยู่ในวงจร UART)

BRK 1 = กำหนดให้สามารถหยุดการรับส่งข้อมูลกะทันหันได้

0 = ไม่มีการหยุด

PAR2: PAR1: PAR0 ใช้กำหนดบิทพาริตี้

000 = ไม่ใช้บิทพาริตี้ 001 = กำหนดพาริตี้คี่

011 = กำหนดพาริตี้คู่ 101 = มาร์ค (Mark)

111 = ช่องว่าง (Space)

STOP 1 = มีบิทปิดท้าย 2 บิท

DAB1: DAB0 ใช้ในการกำหนดจำนวนบิทของข้อมูล

00 = จำนวนบิทข้อมูลเท่ากับ 5 บิท 01 = จำนวนบิทข้อมูลเท่ากับ 6 บิท

10 = จำนวนบิทข้อมูลเท่ากับ 7 บิท 11 = จำนวนบิทข้อมูลเท่ากับ 8 บิท

รีจิสเตอร์ 04H

เป็นรีจิสเตอร์ควบคุมโมเด็ม ที่ใช้ตรวจสอบบิทสำหรับการติดต่อกับโมเด็ม เช่น RTS หรือ DTR มีแอดเดรสอยู่ที่ Base + 04H มีรายละเอียดหน้าที่ของแต่ละบิทดังนี้

			LOOP	**OUT2**	**OUT1**	**RTS**	**DTR**

บิท7 บิท0

LOOP 1 = อนุญาตการส่งค่ากลับ 0 = ไม่อนุญาตการส่งค่ากลับ

RST 1 = อนุญาตการใช้งานขา RTS 0 = ไม่อนุญาตการใช้งาน

DTR 1 = อนุญาตการใช้งานขา DTR 0 = ไม่อนุญาตการใช้งาน

รีจิสเตอร์ 05H

เป็นรีจิสเตอร์แสดงสถานะ การรับและการส่งข้อมูลแบบอนุกรมของ UART ซึ่งจะใช้งานร่วมกับรีจิสเตอร์สำหรับแสดงโหมดและสถานะของการอินเตอร์รัพท์ (รีจิสเตอร์ 02H) เพื่อแสดงสาเหตุของการเกิดอินเตอร์รัพท์ มีรายละเอียดหน้าที่ของแต่ละบิทดังนี้

	TXE	**TBE**	**BREK**	**FRME**	**PARE**	**OVFE**	**RxRD**

บิท7 บิท0

TXE **(Transmitter Empty)**

1 = รีจิสเตอร์บัฟเฟอร์ส่งข้อมูลและชิฟท์รีจิสเตอร์ว่าง

0 = มีข้อมูลเก็บอยู่ในรีจิสเตอร์บัฟเฟอร์ส่งข้อมูลและชิฟท์รีจิสเตอร์

TBE **(Transmitter Buffer Empty)**

1 = รีจิสเตอร์บัฟเฟอร์ส่งข้อมูลว่าง

0 = มีข้อมูลเก็บอยู่ในรีจิสเตอร์บัฟเฟอร์ส่งข้อมูล

BREK **(Break)**

1 = UART ตรวจพบการหยุดรับส่งข้อมูลกะทันหัน

0 = ไม่มีการหยุดรับส่งข้อมูลกะทันหัน

FRAME **(Frame Error)**

1 = UART ตรวจพบความผิดพลาดแบบเฟรม

0 = ไม่มีข้อผิดพลาด

PARE **(Parity Error)**

1 = UART ตรวจพบความผิดพลาดแบบพาริตี้

0 = ไม่มีข้อผิดพลาด

OVRE **(Overrun Error)**

1 = UART ตรวจพบความผิดพลาดแบบโอเวอร์รัน

0 = ไม่มีข้อผิดพลาด

RxRD **(Received Data Ready)**

1 = บัฟเฟอร์รับข้อมูลเต็ม

0 = ไม่มีข้อมูลในบัฟเฟอร์รับข้อมูล

รีจิสเตอร์ 06H

เป็นรีจิสเตอร์แสดงสถานะของโมเด็ม (Modem Status Register: MSR) ใช้แสดงสถานะของขา DCD, RI, DSR และ CTS มีแอดเดรสอยู่ที่ Base + 06H มีรายละเอียดหน้าที่ของแต่ละบิทดังนี้

DCD	RI	DSR	CTS	DDCD	DRI	DDSR	DCTS

บิท7 บิท0

DCD	1 = สัญญาณที่ขา DCD เป็น 1	0 = สัญญาณที่ขา DCD เป็น 0
RI	1 = สัญญาณที่ขา RI เป็น 1	0 = สัญญาณที่ขา RI เป็น 0
DSR	1 = สัญญาณที่ขา DSR เป็น 1	0 = สัญญาณที่ขา DSR เป็น 0
CTS	1 = สัญญาณที่ขา CTS เป็น 1	0 = สัญญาณที่ขา CTS เป็น 0

DDCD **(Delta Data Carrier Detect)**

1 = บิท DCD มีการเปลี่ยนแปลงเมื่อเทียบกับการอ่านค่าครั้งที่แล้ว

0 = บิท DCD ไม่มีการเปลี่ยนแปลง

DRI **(Delta Ring Indicator)**

1 = บิท RI มีการเปลี่ยนแปลงเมื่อเทียบกับการอ่านค่าครั้งที่แล้ว

0 = บิท RI ไม่มีการเปลี่ยนแปลง

DDSR **(Delta Data Set Ready)**

1 = บิท DSR มีการเปลี่ยนแปลงเมื่อเทียบกับการอ่านค่าครั้งที่แล้ว

0 = บิท DSR ไม่มีการเปลี่ยนแปลง

DCTS **(Delta Clear To Send)**

1 = บิท CTS มีการเปลี่ยนแปลงเมื่อเทียบจากการอ่านค่าครั้งที่แล้ว

0 = บิท CTS ไม่มีการเปลี่ยนแปลง

รีจิสเตอร์ 07H

เป็นรีจิสเตอร์สำหรับการเก็บข้อมูลชั่วคราว หรือใช้ทำหน้าที่เป็นหน่วยความจำแรมขนาด 1 ไบท์ ซึ่งจะไม่มีผลใด ๆ กับการใช้งาน UART

3.2 ระบบบัส I^2C

ระบบบัสแบบ I^2C ย่อมาจาก Inter IC ถูกพัฒนาโดยบริษัทฟิลลิปส์ (Phillips) เป็นการสื่อสารอนุกรมแบบซิงโครนัส 2 ทิศทาง โดยมีลักษณะสำคัญดังนี้

1. ใช้สายสัญญาณเพียง 2 สายในการติดต่อสื่อสารระหว่างกันไอซีมาสเตอร์กับไอซีสเลฟ สายหนึ่งเป็นสัญญาณข้อมูลหรือ SDA อีกสายเป็นสัญญาณนาฬิกาหรือ SCL

2. ไอซีที่ต่อพ่วงบนบัสแต่ละตัวมีแอดเดรสเฉพาะที่แตกต่างกันสามารถอ้างถึงได้ โดยใช้ซอฟแวร์

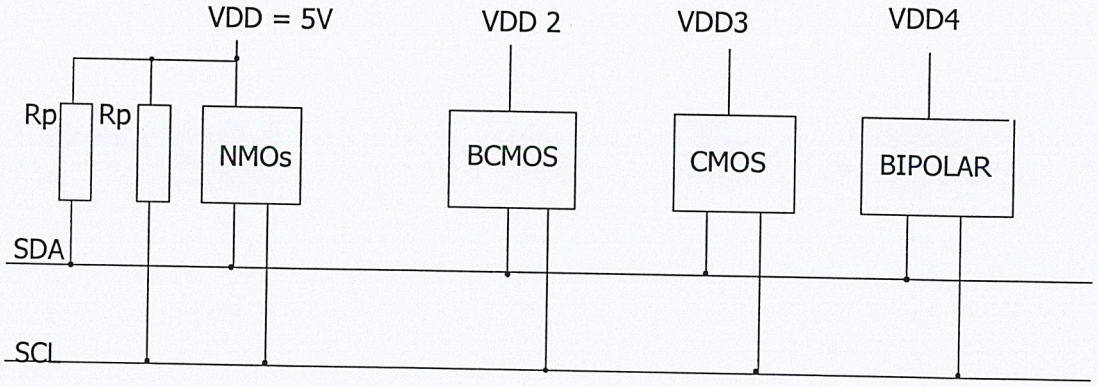
3. บนบัสเดียวกันสามารถมีไอซีมาสเตอร์ได้มากกว่าหนึ่งตัวเนื่องจากมีระบบป้องกันการใช้งานบัสพร้อมกัน

4. การส่งข้อมูลเป็นแบบอนุกรม 8 บิทสองทิศทาง ที่ความเร็ว 100 กิโลบิทต่อวินาทีใน Standard Mode, 400 กิโลบิทต่อวินาทีใน Fast Mode และ 3.4 เมกกะบิทต่อวินาทีใน High-Speed Mode

5. จำนวนไอซีที่ต่อบนบัส สามารถมีได้มากเท่าที่ค่าความจุรวมของบัสทั้งหมดที่รวมแล้วไม่เกิน400 พิโคฟารัด

3.2.1 การต่อเชื่อมไอซีบนบัส

ไอซีแต่ละตัวสามารถต่อเชื่อมเข้ากับบัสได้โดยตรงโดยสายสัญญาณ SDA และ SCLจำเป็นต้องมีตัวต้านทานพูลอัพอยู่กับแรงดันบวกตลอดเวลา เพื่อทำให้แรงดันในสายเป็นลอจิกหนึ่งเมื่อไม่มีการส่งข้อมูล และบัสแบบ I^2C สามารถใช้กับไอซีที่มีระดับแรงดันต่างกันได้ด้วย ดูภาพที่ 3.4 ประกอบ



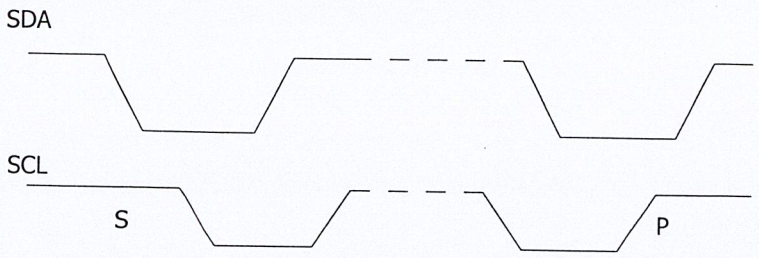
ภาพที่ 3.4 การต่อไอซีบนบัส I^2C

3.2.2 สภาวะต่าง ๆ ของบัส

สภาวะบัสว่าง (Bus Not Busy) สายสัญญาณ SDA และ SCL เป็นลอจิกสูงทั้งคู่ คือไม่มีอุปกรณ์ตัวใดใช้บัสอยู่

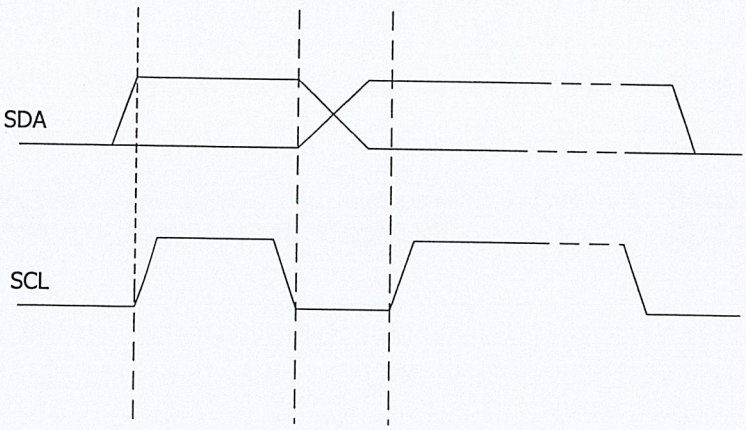
สภาวะเริ่มต้น (Start) สายสัญญาณ SDA เปลี่ยนจากลอจิกหนึ่งเป็นศูนย์ ในขณะที่สายสัญญาณ SCL เป็นลอจิกหนึ่ง เป็นการเริ่มต้นการติดต่อ

สภาวะหยุด (Stop) สายสัญญาณ SDA เปลี่ยนจากลอจิกศูนย์เป็นหนึ่ง ในขณะที่สายสัญญาณ SCL เป็นลอจิกหนึ่ง เป็นการหยุดการติดต่อดูภาพที่ 3.5 ประกอบ



ภาพที่ 3.5 สภาวะเริ่มต้นและสภาวะหยุดของบัส I^2C

สภาวะส่งข้อมูล (Data Validity) นั้นสายสัญญาณ SDA จะต้องไม่มีการเปลี่ยนสถานะเมื่อสายสัญญาณ SCL มีลอจิกหนึ่ง ข้อมูลบิทนั้นจะเท่ากับสถานะของสายสัญญาณ SDA และเมื่อสายสัญญาณ SCL เปลี่ยนกลับมาเป็นลอจิกศูนย์ จึงจะสามารถเปลี่ยนสถานะในสายสัญญาณ SDA ได้ ดูภาพที่ 3.6 ประกอบ



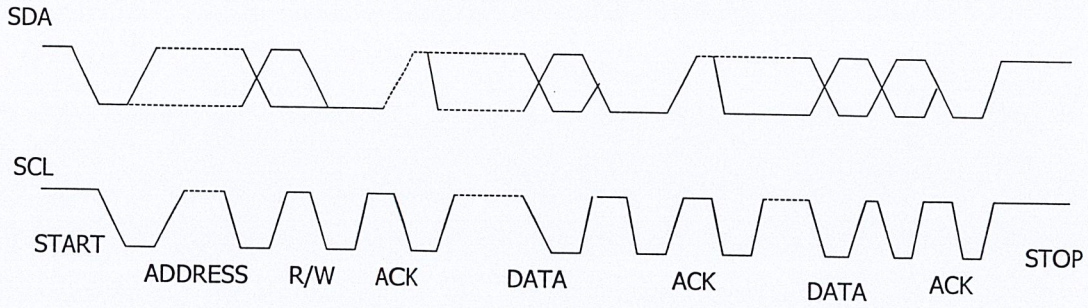
ภาพที่ 3.6 สภาวะส่งข้อมูลของบัส I^2C

สภาวะรับรู้ (Acknowledge) คือเมื่อตัวส่งข้อมูลส่งครบ 1 ไบท์ หรือ8 บิท แล้วตัวรับข้อมูลจะต้องตอบสนองการส่งข้อมูลนั้น โดยดึงสายสัญญาณ SDA ลงเป็นลอจิกศูนย์

3.2.3 รูปแบบการติดต่อสื่อสาร

การติดต่อสื่อสารกันระหว่างไอซี ไอซีมาสเตอร์จะเป็นตัวควบคุมสายสัญญาณ SCL หรือเป็นตัวสร้างสัญญาณนาฬิกาตลอดการติดต่อ การติดต่อเริ่มจากไอซีมาสเตอร์สร้างสภาวะเริ่ม แล้ว

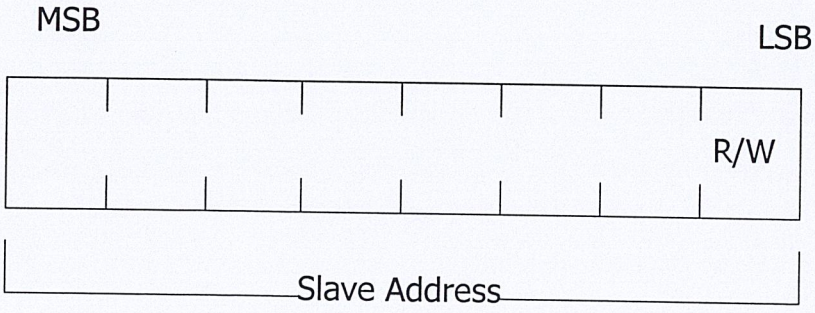
ส่งแอดเดรสและสัญญาณอ่านหรือเขียนข้อมูลลงบนบัส ไอซีเลฟตัวที่มีแอดเดรสตรงกับแอดเดรสที่ส่งมาจะสร้างสัญญาณรับรู้ตอบกลับไป จากนั้นไอซีมาสเตอร์อาจส่งสัญญาณควบคุม หรือข้อมูลให้ไอซีสเลฟนั้น เมื่อครบแต่ละบิทไอซีสเลฟจะต้องสร้างสัญญาณรับรู้กลับไปทุกครั้ง แต่ถ้าเป็นการอ่านข้อมูลจากไอซีสเลฟไอซีมาสเตอร์จะต้องสร้างสัญญาณรับรู้ให้ไอซีสเลฟเมื่อเสร็จสิ้นการติดต่อ ไอซีมาสเตอร์จะสร้างสภาวะหยุดขึ้น ดังภาพที่ 3.7 เป็นตัวอย่างการติดต่อสื่อสารกันอย่างสมบูรณ์



ภาพที่ 3.7 สัญญาณสื่อสารของบัส I^2C

3.2.4 การอ้างอิงแอดเดรส

การอ้างอิงแอดเดรสแบ่งเป็นแบบ 7 บิทและ10 บิทโดยการอ้างอิงแอดเดรสแบบ7บิทข้อมูลใน 7 บิทแรกจะเป็นค่าแอดเดรสเริ่มจากบิทที่มีนัยสำคัญสูงสุด และบิทที่เหลือจะเป็นบิทควบคุมว่าจะอ่านหรือเขียนข้อมูล (R/W) ดังภาพที่ 3.8



ภาพที่ 3.8 รูปแบบแอดเดรส 7 บิทของบัส I^2C

สำหรับการอ้างอิงแอดเดรสแบบ 10 บิท จะต้องใช้ 2 ไบท์ โดย 5 บิทในไบท์แรกจะต้องเป็น 11110 เท่านั้น ตามด้วยแอดเดรส 2 บิท และบิทควบคุม R/W ส่วนไบท์ที่สองจะเป็นแอดเดรส 8 บิทที่เหลือ

3.3 การขับโมดูลแสดงผลแบบผลึกเหลว (LCD)

อุปกรณ์ในปัจจุบันนี้ในส่วนแสดงผลนั้นจะใช้ LCD (Liquid Crystal Display) เป็นส่วนใหญ่ ไม่ว่าจะเป็นเครื่องเล่นวีดีโอ เครื่องวัดคุมต่าง ๆ เราพอจะแบ่งออกเป็นพวก ๆ ได้ดังนี้

1. Character LCD Module
2. Graphic LCD Module
3. Segment Display Type LED Module

ในโมดูล LCD จะมีส่วนประกอบหลัก ๆ 3 ส่วนดังนี้

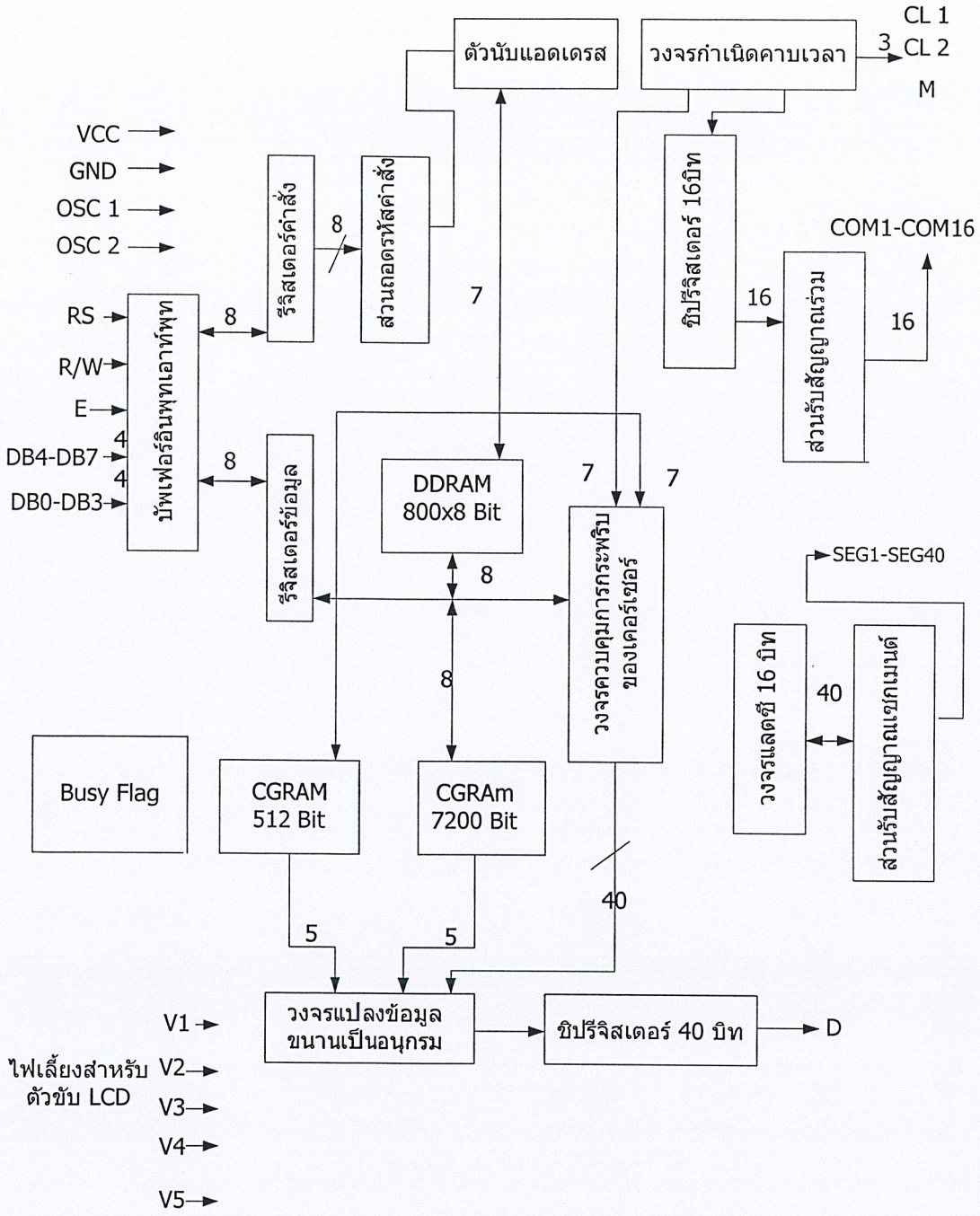
1. ตัวแสดงผล (Display) ภายในเป็นผลึกเหลวที่สามารถแสดงผลให้เห็นโดยอาศัยแสดงจากภายนอก ดังนั้น จึงต้องเป็นมุมในการมองข้อมูลที่แสดงผลบนจอ LCD

2. ตัวควบคุม (Controller) เป็นตัวรับข้อมูลจากอุปกรณ์ภายนอกมาควบคุมการทำงานของโมดูล LCD เช่น ลบจอภาพแสดงตัวอักษรหรือเลื่อนเคอร์เซอร์ เป็นต้น ตัวควบคุมนี้ใช้ชิปควบคุมโดยเฉพาะ ชิปที่นิยมใช้คือ เบอร์ HD44780 และ HD44780 จะใช้ควบคุม LCD แบบอักขระ ส่วน HD61830 ใช้ควบคุม LCD แบบกราฟฟิก

3. ตัวขับ (Driver) เป็นตัวรับสัญญาณจากตัวควบคุมมาขับให้ตัวแสดงผลแสดงผลตามที่กำหนด ชิปที่ใช้ทำหน้าที่เป็นตัวขับได้แก่ เบอร์ HD444100H และ MSM5259 เป็นต้น

3.3.1 โครงสร้างภายในของตัวควบคุมโมดูล LCD

ในการใช้งานโมดูล LCD จำเป็นต้องทำความเข้าใจเกี่ยวกับโครงสร้างและคำสั่งที่ใช้ในการควบคุมให้ดีเสียก่อน โดยโมดูล LCD แบบอักขระสามารถเข้าใจได้ง่ายโดยดูจากภาพที่ 3.9 เป็นบล็อกไดอะแกรมภายในของชิปควบคุม LCD เบอร์ HD44780 ซึ่งใช้ในโมดูล LCD แบบอักขระประกอบด้วย



ภาพที่ 3.9 ไดอะแกรมการทำงานของโมดุล LCD แบบอักขระ

บัฟเฟอร์อินพุทเอาท์พุท เป็นส่วนที่ใช้ในการติดต่อรับส่งข้อมูลกับอุปกรณ์ภายนอก เพื่อที่จะถ่ายทอดข้อมูลเข้าออกภายในตัวควบคุม

รีจิสเตอร์คำสั่ง (Instruction Register: IR) เป็นรีจิสเตอร์ใช้รับข้อมูลคำสั่งจากอุปกรณ์ภายนอก เพื่อนำไปควบคุมการแสดงผล

รีจิสเตอร์ข้อมูล (Data Register:DR) เป็นรีจิสเตอร์ใช้รับข้อมูลจากอุปกรณ์ภายนอกเพื่อถ่ายทอดไปยังหน่วยความจำที่ทำหน้าที่เก็บข้อมูลแสดงผล หรือนำข้อมูลไปสร้างตัวอักษรเพิ่มเติมในแรมเก็บตัวอักษร

แรมเก็บข้อมูลแสดงผล (Display Data Ram: DDRAM) เป็นหน่วยความจำแรมทำหน้าที่เก็บข้อมูลที่มาจากรีจิสเตอร์DRตัวควบคุมจะนำข้อมูลในDDRAM ไปเปิดตาราง (Look up- Table) ของตัวอักษรที่เก็บไว้ในหน่วยความจำรอมและแรมเก็บตัวอักษร เพื่อนำไปที่ตัวแสดงผล

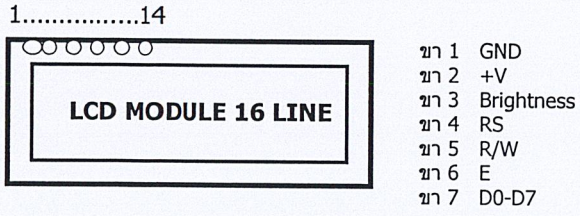
รอมเก็บตัวอักษร (Charater Generator ROM: CGROM) เป็นหน่วยความจำรอมที่ใช้เก็บข้อมูลตัวอักษรหรือสัญลักษณ์ที่สามารถอ่านออกไปแสดงผลได้ มีขนาด 7200 บิท โดยจะถูกอ่านด้วยค่าของข้อมูลใน DDRAM

แรมเก็บตัวอักษร (Character Generator RAM: CGRAM) เป็นหน่วยความจำแรมที่ใช้เก็บอักษรที่มีการสร้างเพิ่มเติมขึ้นใหม่ ในกรณีที่ตัวอักษรใน CGRAM ไม่เพียงพอ มีขนาด 512 บิท การเขียนและอ่านค่าไปใช้นั้นทำได้เช่นเดียวกับ CGROM คือเขียนข้อมูลลงใน DDRAM แล้วตัวควบคุมจะมาอ่านค่าจาก CGRAM เอง

แฟลก BUSY เป็นส่วนที่ทำหน้าที่แจ้งสถานะ การทำงานของตัวควบคุมให้อุปกรณ์ที่อยู่ภายนอกทราบว่า ตัวควบคุมพร้อมที่จะรับข้อมูลหรือคำสั่งหรือไม่ ดังนั้นก่อนการส่งข้อมูลหรือคำสั่งมายังตัวควบคุมต้องตรวจสอบสถานะของแฟลก BUSY นี้เสียก่อน

3.3.2 โมดูล LCD ขนาด 16 ตัวอักษร 1 บรรทัด

สำหรับโมดูล LCD ที่ใช้ในการทำโครงงาน เป็นขนาด 16 ตัวอักษร 1บรรทัด เนื่องจากราคาถูก และเป็นโมดูล LCD ที่มีโครงสร้างเป็นมาตรฐาน มีผู้ผลิตหลายราย และมีการระบุเบอร์แตกต่างกันออกไปตามผู้ผลิต อาทิ เช่น LM020L ของฮิตาชิ, DMC-16117Aของคอปเทร็กซ์ เป็นต้น แต่อย่างไรก็ตามคอนโทรลเลอร์ที่ใช้คือเบอร์เดียวกันคือ HD44780 ของ ฮิตาชิ โมดูลขนาด 16x1 มีขาที่ต่อใช้งานทั้งสิ้น 14 ขา มีการจัดขาดังในภาพที่ 3.10



ภาพที่ 3.10 รูปร่างและการจัดขาโมดูล LCD

สำหรับรายละเอียดการทำงานของแต่ละขามีดังนี้

1. Vss (ขา 1) ต่อกราวนด์

2. Vdd (ขา 2) ต่อไฟเลี้ยง + 5โวลต์

3. Vo (ขา 3) เป็นขาอินพุทรับแรงดันเพื่อปรับความเข้มของการแสดงผล

4. RS (ขา 4) เป็นขาอินพุทใช้ในการแยกชนิดของข้อมูลที่ทำการประมวลผลในขณะนั้นว่าเป็นคำสั่งสำหรับรีจิสเตอร์ IR เป็นข้อมูลสำหรับรีจิสเตอร์ DR โดยถ้าขานี้เป็น “0” ข้อมูลที่ส่งมาจะเป็นคำสั่ง แต่ถ้าขานี้เป็น “1” ข้อมูลที่ส่งมาจะเป็นข้อมูลสำหรับการแสดงผล

5. R/W (ขา 5) เป็นขาที่ใช้เลือกการอ่านหรือเขียนข้อมูลกับ LCD ถ้าเป็น “0” เป็นการกำหนดให้เขียนข้อมูล แต่ถ้าเป็น “1” จะเป็นการอ่านข้อมูล

6. E (ขา 6) เป็นขาอีนาเบิล LCD ให้ทำงาน

7. D0-D7 (ขา 7-14) เป็นขาที่ใช้เป็นทางผ่านของข้อมูลระหว่าง LCD กับอุปกรณ์ภายนอกขนาด 8 บิท

บทที่ 4
ทฤษฎีเกี่ยวกับโปรแกรมที่ใช้เขียน

4.1 โปรแกรมภาษาซี

การเขียนโปรแกรมสำหรับไมโครคอนโทรลเลอร์ด้วยภาษาแอสเซมบลี มีส่วนโครงสร้างของโปรแกรมจะดูยาก การควบคุมทิศทางของโปรแกรมก็เข้าใจได้ยาก แม้แต่ผู้เขียนโปรแกรมเองเมื่อเวลาผ่านไปแล้วกลับมาดูงานที่ตัวเองเขียนก็อาจไม่เข้าใจและยุ่งยากเสียเวลา แต่ในปัจจุบันได้มีการพัฒนาคอมไพเลอร์ให้สามารถแปลภาษาระดับกลาง ซึ่งมีโครงสร้างทางภาษาที่เข้าใจง่ายอย่างเช่นภาษาซี ซึ่งจะเขียนชุดคำสั่งบนโปรแกรม Editor ให้มีนามสกุลเป็นเป็น. C

4.1.1 โครงสร้างของภาษาซี

ด้วยภาษาซีที่สามารถที่สามารถเขียนโปรแกรมเป็นแบบโครงสร้างได้ โดยโปรแกรมจะแบ่งการทำงานต่างๆออกเป็นกลุ่มหรือฟังก์ชันเหล่านั้นสามารถเรียกขึ้นมาใช้ใหม่ได้ ในการเขียนโปรแกรมจะต้องระบุไว้ว่าในโปรแกรมนั้นมีฟังก์ชันใดให้ใช้บ้าง แต่ทุกโปรแกรมต้องมีฟังก์ชันหลักที่ชื่อว่า main () เสมอ

4.1.2 ตัวแปรและค่าคงที่

การใช้งานตัวแปรและค่าคงที่ต่างๆ จะต้องมีการประกาศชื่อตัวแปรขึ้นมาเสียก่อนเมื่อมีการคอมไพล์โปรแกรมตัวคอมไพเลอร์จะเตรียมพื้นที่ในหน่วยความจำแรมเอาไว้สำหรับเก็บตัวแปรและค่าคงที่เหล่านั้นในการประกาศตัวแปรสามารถทำได้ดังนี้

ประเภทของข้อมูล ชื่อตัวแปร[,…. ,];

การประกาศตัวแปรจะต้องเริ่มด้วยการชื่อตัวแปร โดยจะประกาศครั้งละกี่ตัวก็ได้ ส่วนชื่อของตัวแปรนั้นจะซ้ำกันไม่ได้และต้องไม่ซ้ำกับชื่อของคำสงวน (Keyword) ของคอมไพเลอร์ตัวนั้นๆ สำหรับประเภทของข้อมูลในการประกาศตัวแปรแสดงไว้ในตารางที่ 4.1

ตารางที่ 4.1 ตัวแปรในภาษาซีและคุณสมบัติ

ประเภทของข้อมูล	ขนาดบิท	ค่าที่เก็บได้
Bit	1	0 ถึง 1
char	8	-128 ถึง 127
unsigned char	8	0 ถึง 255
int	16	-32768 ถึง 32767
unsigned int	16	0 ถึง 65535
long	32	-2147483648 ถึง +2147483647
unsigned long	32	0 ถึง 4294967295
float	32	-1.17549 e-32ถึง e+ 3.402823+38

4.1.3 ตัวดำเนินการในภาษาซี

ตัวดำเนินการจะเป็นตัวที่ใช้กระทำกับตัวแปร ค่าคงที่ต่างๆ ให้รวมเป็นค่าเดียวกัน โดยอาจจะกระทำทางคณิตศาสตร์ หรือกระทำทางลอจิกก็ได้ในการเขียนโปรแกรมด้วยภาษาซีนั้นตัวดำเนินการจะแบ่งออกได้เป็นสองกลุ่มใหญ่ๆ คือตัวดำเนินการที่กระทำกับตัวถูกกระทำตัวเดียว (Single operands operators) และตัวดำเนินการที่กระทำกับตัวถูกกระทำสองตัว (Two operands operators)

4.2 โปรแกรม Visual Basic 6.0

Visual Basic เป็นเครื่องมือในการพัฒนาโปรแกรมบน Microsoft Windows 95, โปรแกรม Windows98และWindows NT ที่ได้รับการพัฒนาโดยบริษัทไมโครซอฟท์ (Microsoft) ซึ่งถือเป็นคอมไพเลอร์ (Compiler) ที่ได้รับความนิยมสูงโดยเฉพาะอย่างยิ่งในเมืองไทยมีการใช้งานอย่างกว้างขวาง

Visual Basic 6.0 ประกอบไปด้วยเครื่องมือต่างๆที่ช่วยให้การพัฒนาปรแกรมสามารถทำได้ด้วยความรวดเร็ว หรือที่เรียกกันว่า Rapid Application Development (RAD) อีกทั้งยังช่วยให้เราสามารถเขียนโปรแกรมได้ง่ายดาย เนื่องจากการเขียนโปรแกรมมีพื้นฐานมาจากภาษา BASIC ซึ่งทำให้ผู้ที่ไม่เคยเขียนโปรแกรมเรียนรู้ได้ไม่ยากการเขียนโปรแกรมอาศัยหลักการของ Object Oriented จึงทำให้ประหยัดเวลาในการเขียนโปรแกรมลงไปได้มาก และเราสามารถนำส่วนของโปรแกรมที่เขียนขึ้นไปใช้ในโปรแกรมอื่นที่เกี่ยวข้องได้อีก

4.2.1 องค์ประกอบต่าง ๆ ของ Visual Basic 6.0

เมื่อเราเปิดใช้งาน Visual Basic 6.0 จะพบกับลักษณะการทำงานที่เรียกว่า IDE (Integrated Development Environment) คือรวบรวมเครื่องไม้เครื่องมือข้อมูลที่ใช้งานต่างๆ ไว้ในหน้าจอเดียว ทำให้เรียกใช้งานได้ง่ายประกอบไปด้วย

- Menubar เมนูบาร์เป็นส่วนที่รับคำสั่งในรูปแบบเมนู เมื่อเราทำการสร้างแอพพลิเคชันด้วย Visual Basic เป็นเหมือนศูนย์กลางที่ควบคุมการสร้างแอพพลินเคชัน
- ToolBar ในการใช้งานเมนูบาร์สั่งงานอาจจะมีขั้นตอนที่ยุ่งยาก เพื่อลดขั้นตอนลง เราจะคลิกที่ทูลบาร์เพียงครั้งเดียวก็สามารถสั่งงานที่เราต้องการได้ (เป็นเหมือนคีย์ลัดในการทำงาน)
- ToolBox ทูลบ็อกซ์ เป็นกล่องเก็บ ActiveX Control ซึ่งเราจะนำมาประกอบกันเป็นส่วนต่างๆ ของแอพพลิเคชัน
- Project Explorer เป็นเครื่องมือที่ใช้ควบคุมการทำงานของโปรเจ็กต์
- Properties Window เป็นส่วนที่กำหนดพร็อพเพอร์ตี้ให้กับส่วนออบเจ็กต์ต่างๆ ในแอพพลิเคชัน
- Form Layout ฟอร์มเลย์เอาท์เป็นหน้าตาคร่าวๆ ของฟอร์มที่ได้ จากการรันแอพพลิเคชันทำให้เราทราบตำแหน่งที่มันจะปรากฏบนจอภาพเมื่อแอพพลิเคชันทำงาน
- Form Designer ฟอร์มดีไซเนอร์เป็นส่วนที่เรามองเห็นได้ ขณะออกแบบแอพพลิเคชันของ Visual Basic ซึ่งเราจะออกแบบหน้าตาของแอพพลิเคชันผ่านฟอร์มดีไซเนอร์
- Code Window โค้ดวินโดว์เป็นส่วนที่เราเขียนโปรแกรมหรือ (เรียกสั้นๆ ว่าเขียนโค้ด) เพื่อควบคุมการทำงานของแอพพลิเคชัน

4.2.2 เริ่มสร้างแอพพลิเคชันด้วย Visual Basic

สำหรับขั้นตอนการสร้างแอพพลิเคชันนั้นมีขั้นตอนที่ควรทำดังนี้

ขั้นแรก : ออกแบบแอพพลิเคชัน

ก่อนจะสร้างแอพพลิเคชัน หรือเขียนโปรแกรมนั้นสิ่งแรกที่ต้องทำคือต้องทราบให้แน่ชัดก่อนว่าแอพพลิเคชันที่เราจะสร้างนั้นจะใช้ประโยชน์อะไร ต้องมีความสามารถอะไรบ้างต้องการให้รูปร่างหน้าตาเป็นอย่างไรซึ่งจำเป็นอย่างยิ่งที่ต้องคิดให้รอบคอบและเขียนออกมาให้ชัดเจน (อาจจะอยู่ในรูปของแผนผังลำดับการทำงาน (Flowchart) หรือเขียนร่างง่าย ๆ ในกระดาษ)

ขั้นที่ 2 : ตกแต่งหน้าตาแอพพลิเคชัน

สำหรับขั้นตอนนี้จะเป็นการตกแต่งรูปร่างของแอพพลิเคชันตามที่ได้ออกแบบไว้พร้อมๆ กับการกำหนดค่าพร็อพเพอร์ตี้ต่างๆ ให้กับคอนโทรลแต่ละตัวในแอพพลิเคชันโดยมีขั้นตอนย่อย ๆ ดังนี้

1. เรียกใช้งาน Visual Basic
2. ดับเบิ้ลคลิกที่คอนโทรลภายใน ToolBox คอนโทรลนั้นจะปรากฏบนฟอร์มดีไซเนอร์
3. จัดวางตำแหน่ง และปรับขนาดของคอนโทรลให้ได้ตามรูปแบบที่เราต้องการ
4. กำหนดค่าให้กับพร็อพเพอร์ตี้ของคอนโทรลนั้น โดยคลิกที่คอนโทรลที่ต้องการแล้วเลือกพร็อพเพอร์ตี้ที่ต้องการกำหนดใน Properties Window
5. นำคอนโทรลตัวอื่นๆ เข้ามาในฟอร์มดีไซเนอร์แล้วปรับหน้าตาจัดวางตำแหน่ง แล้วกำหนดค่าให้พร็อพเพอร์ตี้ของคอนโทรลต่างๆ

ขั้นตอนที่ 3 : เขียนโค้ดกำกับการทำงานของแอพพลิเคชัน

หลังจากตกแต่งหน้าตาเสร็จแล้วขั้นต่อไปคือ การเขียนโค้ดเขียนโปรแกรมเพื่อควบคุมการทำงานต่างๆ ซึ่งเราจะใช้การเขียนโปรแกรมแบบ Event Driven Programming ซึ่งจะเป็นการเขียนโค้ดเพื่อรองรับกับเหตุการณ์ต่าง ๆ ที่จะเกิดขึ้นกับคอนโทรลต่างๆ ในแอพพลิเคชันของเรา ซึ่งมีขั้นตอนดังนี้

1. ดับเบิ้ลคลิกที่คอนโทรลที่ต้องการควบคุม ซึ่งจะปรากฏ Code Window ขึ้นมา
2. ให้เลือกเหตุการณ์ที่เราต้องการจะควบคุม จากลิสต์บ็อกซ์ของ Code Window ในที่นี้เราสนใจตอนที่ผู้ใช้งานคลิกปุ่ม จึงเลือกเหตุการณ์ Click
3. เขียนโค้ดในภาษา Basic เพื่อจัดการกับเหตุการณ์นั้นๆ (ซึ่งควรมีคำอธิบายโปรแกรมด้วย) ในที่นี้เราจะนำเอาชื่อที่ใช้กรอกเอาไว้ (ในคอนโทรล TextBox) มาแสดงพร้อมกับวันเวลา ณ ขณะที่คลิกปุ่มนั้น
4. เลือกคอนโทรลตัวอื่นๆ มาเขียนโดยแต่ละคอนโทรลก็จะมีเหตุการณ์ที่เราจะต้องการไม่เหมือนกัน

ขั้นที่ 4: ทดสอบการทำงานของแอพพลิเคชัน

เมื่อเขียนโค้ดเสร็จแล้วก็ถึงเวลาที่จะทดสอบการทำงานของแอพพลิเคชันที่เราจะสร้างขึ้น ซึ่งประกอบไปด้วยคอนโทรลต่างๆ ที่ปรับต่างไว้และโค้ดที่เขียนเพื่อจัดการกับเหตุการณ์ต่างๆ โดยเราจะทดสอบการทำงานโดยกด <F5> (หรือคลิกปุ่ม ..... บนทูลบาร์ก็ได้)

Visual Basic จะพาเราออกจากโหมดการสร้างโปรแกรม (Design Mode) ไปสู่โหมดการทำงาน (Run Mode) เราก็สามารถทดสอบการทำงานต่างๆ ที่เราสร้างขึ้นไว้ได้ทันที

เมื่อเราต้องการสิ้นสุดการทดสอบก็ให้คลิกที่ปุ่มจบการทำงานหรือเรียกใช้เมนู Run>End ก็ได้

กรณีที่เกิดความผิดพลาดก็จะมีการแจ้งจุดที่ผิดพลาดในโปรแกรมให้ทราบ ซึ่งเราสามารถแก้ไขได้อย่างรวดเร็ว (แล้วก็ทดสอบใหม่)

ขั้นตอนที่ 5: บันทึกเก็บไว้ในคอมพิวเตอร์

หลังจากทดสอบจนแน่ใจแล้วว่าแอพพลิเคชันที่สร้างนั้นทำงานได้ถูกต้อง เราจึงบันทึกเก็บไว้ซึ่งสามารถแก้ไขและเพิ่มเติมความสามารถอื่นๆ ได้ในภายหลังด้วยวิธีการดังนี้

1. เลือกเมนู File > Save Project
2. จะปรากฏไดอะล็อกบ็อกซ์ Save File As ให้เราตั้งชื่อไฟล์ที่เก็บฟอร์มทุกๆ ฟอร์มที่มีแอพพลิเคชัน (ในที่นี้มีแค่ฟอร์มเดียว)
3. คลิกปุ่ม SAVE เพื่อบันทึก
4. จากนั้นจะปรากฏไดอะล็อกบ็อกซ์ Save Project As ให้เราตั้งชื่อแอพพลิเคชันของเราว่าชื่ออะไร
5. คลิกปุ่ม SAVE เพื่อบันทึก
6. จะเห็นว่ามีการเปลี่ยนแปลงเกิดขึ้นกับจุดต่างๆ ของVisual Basic หลังจากที่เราบันทึกเก็บไว้

ขั้นที่ 6 : การสร้างไฟล์ . EXE (Make)

เมื่อเราสร้างแอพพลิเคชันเสร็จแล้วเราอาจจะต้องการนำแอพพลิเคชันที่สร้างขึ้น เรียกใช้งานได้เองโดยไม่ต้องเรียกผ่าน Visual Basic หรือต้องการนำไปใช้งานในคอมพิวเตอร์เครื่องอื่นๆ ซึ่งเราจะทำได้โดยการสร้างไฟล์เอ็กซ์คิวต์ (ไฟล์ที่มีนามสกุลเป็น .EXE) โดยมีวิธีการดังนี้

1. เลือกเมนู File > Make ชื่อโปรเจ็กต์ ...
2. ในไดอะล็อกบ็อกซ์ Make Project ให้เราตั้งชื่อไฟล์เอ็กซ์คิวต์ที่ต้องการ
3. คลิกปุ่ม ..... ก็จะได้ไฟล์ .EXE ตามที่ต้องการ

บทที่ 5
การออกแบบเครื่องบันทึกข้อมูลแบบพกพา

5.1 การทำงานของเครื่องบันทึกข้อมูลแบบพกพา

การทำงานของเครื่องบันทึกข้อมูลแบบพกพามีรายละเอียดการทำงานดังนี้

1. เก็บข้อมูลที่เป็นสัญญาณอนาลอกในรูปของสัญญาณแรงดัน 1-5 Vdcได้ 4 ช่องสัญญาณ โดยแต่ละช่องสัญญาณสามารถเก็บข้อมูลได้ 1000 ค่า

2. ทำการแปลงสัญญาณอนาลอกเป็นสัญญาณดิจิตอลขนาด 10 บิทโดยใช้วงจร ADC ที่อยู่ภายในตัวไมโครคอนโทรลเลอร์

3. สามารถปรับแต่งฐานเวลาในการบันทึกข้อมูลได้ โดยใช้คีย์สวิทช์ และสามารถกำหนดให้แต่ละช่องสัญญาณมีฐานเวลาในการเก็บข้อมูลต่างกันได้โดยตั้งค่าที่ MENU REC EVERY

4. ในขณะที่บันทึกข้อมูลสามารถเรียกดูค่าข้อมูลดูได้จากหน้าจอ LCD ของเครื่องบันทึกได้ โดยข้อมูลในแต่ละช่องสัญญาณจะแสดงเป็นเลขฐานสิบตั้งแต่ 0 ถึง 1023 พร้อมทั้งแสดง วัน เดือน ปี และช่วงเวลาการเก็บ

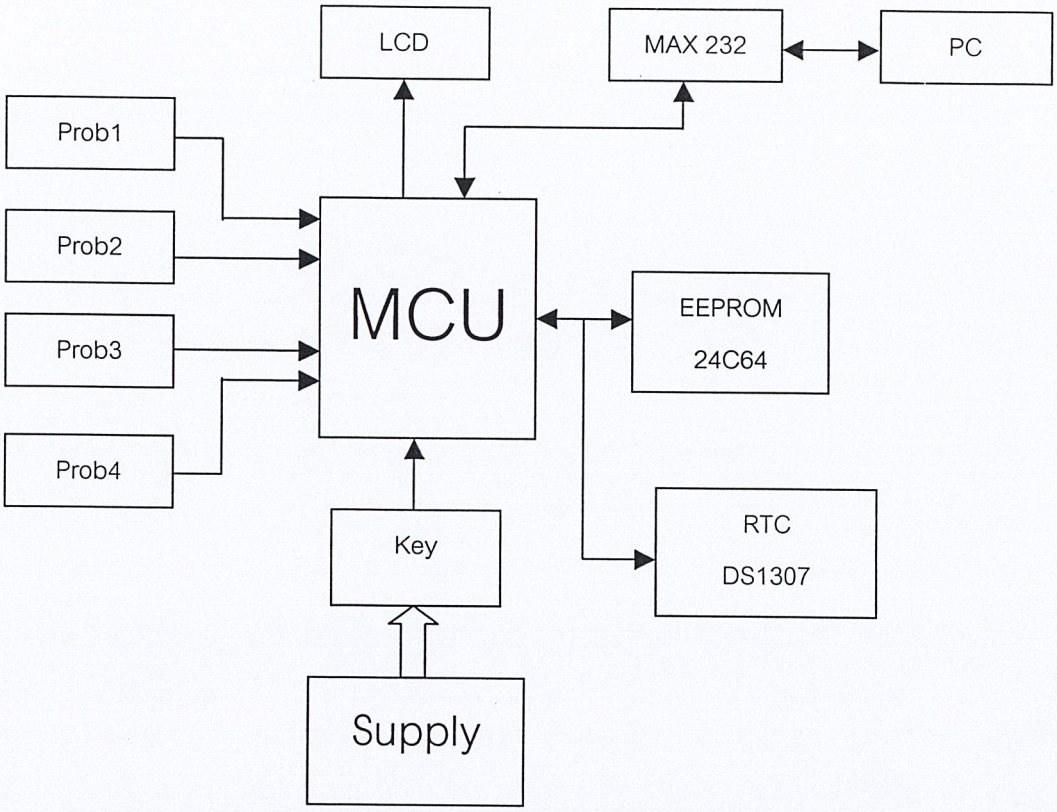
5. สามารถนำเครื่องบันทึกข้อมูลนี้เชื่อมต่อกับคอมพิวเตอร์ส่วนบุคคลผ่านพอร์ท RS-232 ที่ความเร็ว 9600 บิทต่อวินาที เพื่อแสดงข้อมูลทั้งหมดที่บันทึกมาเป็นกราฟหรือข้อมูลเชิงสถิติด้วยโปรแกรมที่เขียนโดย Visual Basic พร้อมทั้งสามารถแสดงข้อมูลผ่านเครื่องพิมพ์ได้อีก

5.2 โครงสร้างทางฮาร์ดแวร์

โครงสร้างทางฮาร์ดแวร์ของเครื่องบันทึกข้อมูลแบบพกพาที่ได้ออกแบบแสดงไว้ ดังภาพที่ 5.1 ซึ่งประกอบไปด้วย

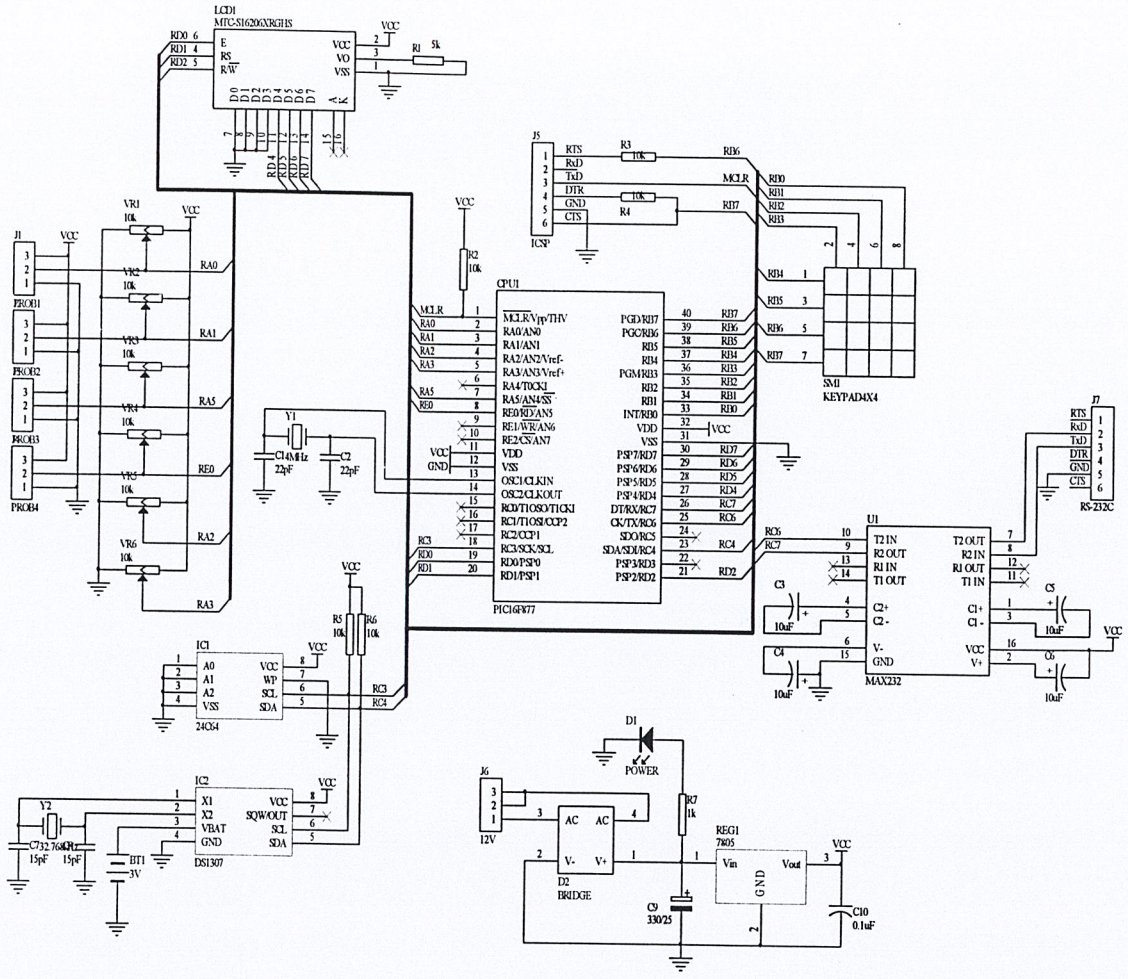
1. ไมโครคอนโทรลเลอร์ PIC16F877 ควบคุมการทำงานทั้งหมด
2. ส่วนแปลงสัญญาณอนาลอกเป็นดิจิตอลอยู่ภายในไมโครคอนโทรลเลอร์ขนาด 10 บิท
3. หน่วยความจำเก็บข้อมูลดิจิตอลที่แปลงได้ ใช้หน่วยความจำอนุกรม EEPROM เบอร์ 24C64 ขนาด 8 กิโลไบท์ต่อบนบัสแบบ I^2C
4. เวลาถูกควบคุมโดยไอซีคำนวณค่าเวลาจริงเบอร์ RTC DS1307
5. ส่วนติดต่อกับพอร์ทอนุกรมRS232-Cของคอมพิวเตอร์ใช้ไอซี MAX 232ระบบแรงดัน
6. ส่วนการติดต่อกับผู้ใช้ LCD ขนาด 16 ตัวอักษร 1 บรรทัดแสดงผล
7. คีย์สวิทช์ขนาด 4x4 Key

8. แหล่งจ่ายไฟ อแด็ปเตอร์ 12V / 500 mA (Supply)
9. อินพุทอนาลอก 4 ช่องสัญญาณ (Prob1-Prob4)
10. คอมพิวเตอร์แสดงการบันทึกข้อมูล(PC)



ภาพที่ 5.1 แสดงบล็อกไดอะแกรมของเครื่องบันทึกข้อมูลแบบพกพา

สำหรับวงจรที่ได้ออกแบบของเครื่องบันทึกข้อมูลแบบพกพาแสดงไว้ในภาพที่ 5.2



ภาพที่ 5.2 วงจรที่ได้ออกแบบของเครื่องบันทึกข้อมูลแบบพกพา

การทำงานเครื่องบันทึกข้อมูลแบบพกพา

เครื่องบันทึกข้อมูลแบบพกพาจะใช้ไมโครคอนโทรลเลอร์ PIC16F877 สำหรับควบคุมการทำงาน ซึ่งในไมโครคอนโทรลเลอร์ตัวนี้จะมี ADC ขนาด10 บิท 8 แชนแนลอยู่ภายในซึ่งใช้สำหรับการวัดสัญญาณอนาลอกจากโปรบทั้ง 4 โปรบ ADC ที่อยู่ใน PIC16F877 นี้จะใช้วัดแรงดันอนาลอกที่มีขนาดแรงดันอยู่ที่ 0 ถึง 5 V โดยจะออกแบบให้มี VR สำหรับปรับแรงดันอ้างอิงที่ด้านไฟต่ำได้ ซึ่งจะทำให้สามารถปรับให้วัดแรงดันอนาลอกได้ประมาณ 1 ถึง 5 V เนื่องจากปรับให้มีความแคบกว่านี้ จะทำให้ไม่สามารถวัดแรงดันที่ละเอียดมากๆ ได้

โปรบทั้ง 4 ตัวจะแยกกันทำงานโดยอิสระสามารถบันทึกค่าแยกกันได้ โดยการเก็บบันทึกค่าจะใช้หน่วยความจะแบบ EEPROM เบอร์ 24C64 ซึ่งเป็น I^2C EEPROM มีขนาดหน่วยความจำ 8 กิโลไบท์ รูปแบบการเก็บข้อมูลจะแบ่งหน่วยความจำออกเป็น 4 ส่วนตามจำนวนโปรบทำให้มี

พื้นที่ 2 กิโลไบท์ต่อโปรบโดย 1 ข้อมูลที่เกิดจากการวัดแต่ละครั้งจะมีขนาด 10 บิท ตามความละเอียดของ ADC จึงใช้พื้นที่ของหน่วยความจำ 2 ไบท์เพื่อเก็บผลการวัด1 ค่า ดังนั้นจึงสามารถเก็บข้อมูลการวัดแต่ละโปรบได้สูงสุด 1023 ครั้ง

ความถี่ในการวัดค่าของแต่ละโปรบสามารถตั้งได้จาก 1 วินาทีไปจนถึง 1วันต่อครั้งแยกกันอย่างอิสระทั้ง 4 โปรบ ซึ่งได้ใช้ไอซีฐานเวลาเบอร์ DS1307 สำหรับกำหนดช่วงเวลาของกาวัด ไอซีตัวนี้ใช้การเชื่อมต่อแบบ I^2C เหมือนหน่วยความจำ EEPROM จึงใช้พอร์ทเชื่อมต่อร่วมกันได้

การแสดงผลจะใช้จอแสดงผลแบบผลึกเหลว (LCD) ขนาด 16 ตัวอักษร 1 บรรทัดใช้แสดงผลการวัดของแต่ละโปรบ โดยให้แสดงค่าที่ได้จาก ADC ขนาด 10 บิท ตรงๆ ซึ่งสามารถนำไปใช้คำนวณเพื่อแสดงค่าสัญญาณอนาลอกที่วัดได้ ในคอมพิวเตอร์การเชื่อมต่อ LCD เป็นแบบบัสข้อมูล 4 บิทเพื่อการประหยัดพอร์ทของไมโครคอนโทรลเลอร์

ส่วนของการเชื่อมต่อกับเครื่องคอมพิวเตอร์ผ่านพอร์ทอนุกรม RS-232 ได้ใช้ไอซีเบอร์ MAX232 เป็นตัวอินเตอร์เฟส ระหว่าง PIC16F877 ซึ่งมีโมดูลสื่อสารข้อมูลอนุกรม (UART) อยู่ภายในเข้ากับเครื่องคอมพิวเตอร์

การทำงานของฮาร์ดแวร์ สำหรับควบคุมการวัด โดยมีเมนูให้สามารถปรับตั้งค่าต่างๆ ได้ โดยกดปุ่ม 'E' เมื่ออยู่ในโหมดแสดงผลการวัด ก็จะเป็นการเข้าสู่โหมดเมนู ซึ่งประกอบไปด้วย

1. เมนู DISPLAY สำหรับเลือกโปรบที่จะแสดงผลบน LCD สามารถเลือกได้ทีละโปรบ จาก 1 ถึง 4 และสามารถเลือกจากคีย์ลัดได้ โดยกดปุ่มหมายเลข 1 ถึง 4 เพื่อแสดงผลโปรบแต่ละช่องขณะอยู่ในโหมดแสดงผลการวัด
2. เมนู DATE/TIME สำหรับตั้งวันที่และเวลา ของไอซีฐานเวลา DS1307
3. เมนูRECEVERY สำหรับตั้งช่วงเวลาสำหรับการบันทึกผลการวัดของแต่ละโปรบ ซึ่งสามารถตั้งได้ จาก 1 วินาที ไปจนถึง 1 วัน โดยการเก็บข้อมูลจะเริ่มต้นทันทีหลังจากตั้งช่วงเวลาเสร็จและลบข้อมูลเดิมที่อยู่ในหน่วยความจำเมื่อเก็บข้อมูลเกิน 1023 ครั้งก็จะทำการเขียนทับข้อมูลตัวที่เก่าที่สุดไปเรื่อยๆ โดยอุปกรณ์แต่ละส่วนมีหน้าที่การทำงานดังนี้

ไมโครคอนโทรลเลอร์

ไมโครคอนโทรลเลอร์ PIC16F877 ใช้สัญญาณนาฬิกาจากตัวผลึกคริสตอลที่สร้างความถี่ 4 เมกกะเฮิร์ทซ์ เป็นตัวกำหนดจังหวะการทำงาน

วงจรแปลงสัญญาณอนาลอกเป็นดิจิตอล

วงจรแปลงสัญญาณอนาลอกเป็นดิจิตอลภายในไมโครคอนโทรลเลอร์โดยใช้วิธีแปลงแบบ Successive Approximation ที่ความละเอียด 10 บิท

หน่วยความจำแบบ EEPROM

หน่วยความจำแบบ EEPROM ใช้ไอซีเบอร์ 24C64 มีความจุ 8 กิโลไบท์ x8 บิทต่อบนบัสแบบ I^2C เข้าสู่ไมโครคอนโทรลเลอร์ที่ขา SDA และ SCL ตัวต้านทาน R5 และ R6 เป็นตัวต้านทานพูลอัพของขา SDA และ SCL

ส่วนติดต่อกับพอร์ทอนุกรม RS-232

เนื่องจากระดับแรงดันสำหรับลอจิกศูนย์และหนึ่งของพอร์ทอนุกรม RS-232 แตกต่างจากระดับแรงดันของทีทีแอล (TTL) หรือซีมอส (CMOS) จึงจำเป็นที่จะต้องแปลงระดับแรงดันเพื่อให้สามารถติดต่อสื่อสารกันได้โดยใช้ไอซี MAX232 ซึ่งทำหน้าที่นี้โดยเฉพาะ โดยเมื่อระดับสัญญาณทีทีแอลที่ขา Txin จะแปลงเป็น RS-232 ออกที่ขา Txout และระดับสัญญาณ RS-232 ที่ขา Rxin จะถูกแปลงเป็นทีทีแอลออก ที่ขา Rxout

ส่วนแสดงผล

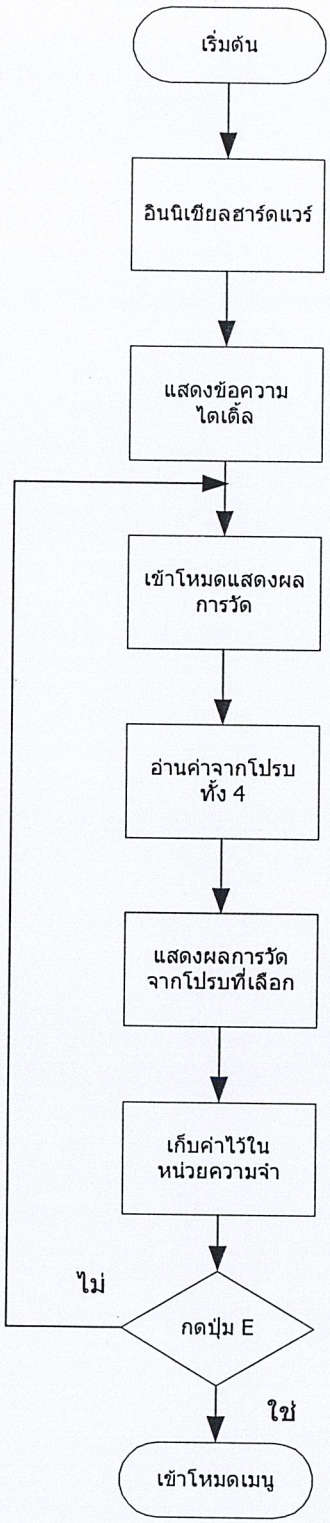
จอ LCD แสดงผลขนาด 16 ตัวอักษร 1 บรรทัดใช้บัสแอดเดรสและข้อมูลร่วมกันจึงต่อเพียง 4 เส้นส่วนขา R/W และขา RS ต่อกับขา RD1และ RD2 ตามลำดับและขา Enable ต่อกับขา RD0

5.3 โปรแกรมการแสดงผล

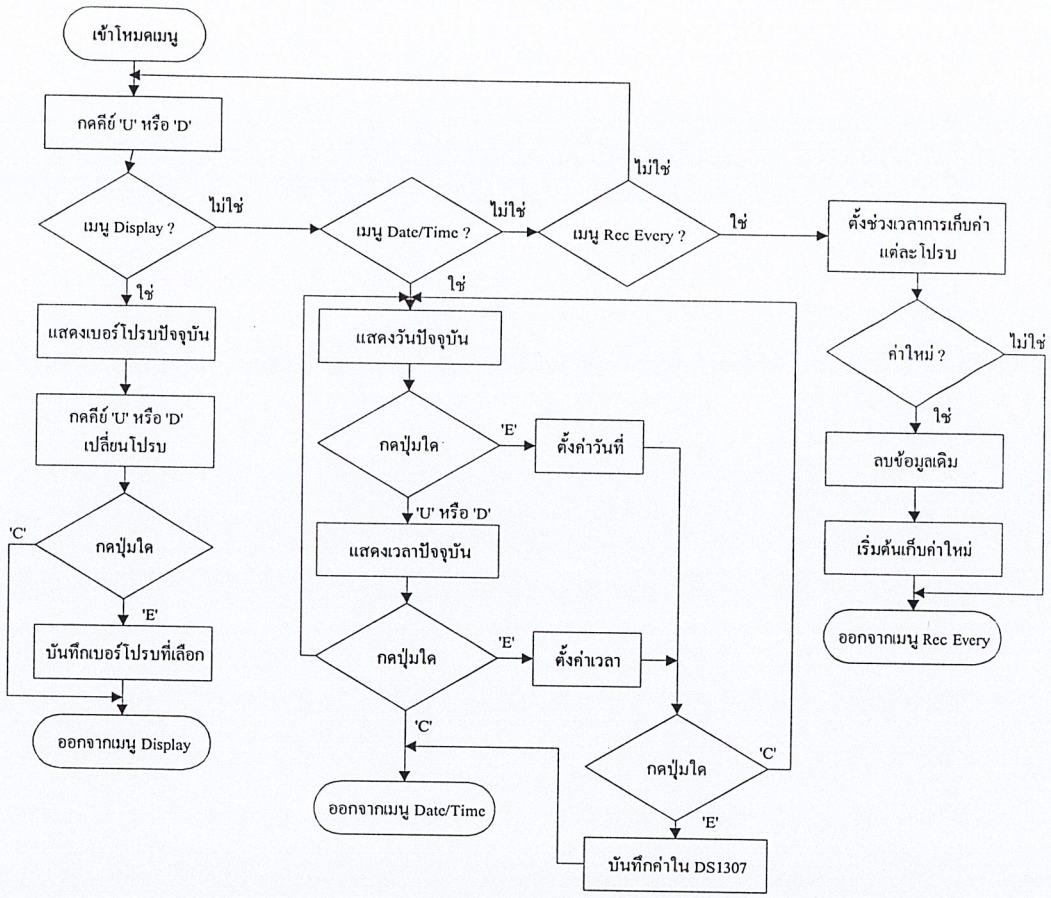
โปรแกรมที่ใช้เขียนในการทำปริญญานิพนธ์นี้จะประกอบไปด้วย โปรแกรมภาษาซี ซึ่งจะใช้เขียนโปรแกรมให้ตัวคอนโทรลเลอร์และโปรแกรม Visual Basic ที่ใช้เขียนให้แสดงผลที่หน้าจอคอมพิวเตอร์มีรายละเอียดดังนี้คือ

5.3.1 แผนผังลำดับการทำงานของโปรแกรมภาษาซี

แผนผังการทำงานจะแบ่งเป็น 2 ส่วนคือ เริ่มต้นก่อนเข้าสู่โหมดเมนูการทำงานโปรแกรมจะทำการตั้งค่าเริ่มต้นให้กับส่วนของการใช้งานที่เกี่ยวข้องกับฮาร์ดแวร์และตัวพารามิเตอร์ตามหลักโครงสร้างของอุปกรณ์แต่ละตัวก่อน จากนั้นจะตรวจสอบการกดสวิทช์ให้เข้ามายังโหมดเมนูและตรวจสอบการทำงานตามเงื่อนไขของฟังก์ชันแต่ละตัว แสดงไว้ในภาพที่ 5.3 (ก) และ 5.3 (ข)



ภาพที่ 5.3 (ก) แผนผังลำดับการทำงานของโปรแกรมภาษซีก่อนเข้าสู่โหมดเมนู



ภาพที่ 5.3 (ข) แผนผังลำดับการทำงานของโปรแกรมภาษาซีเมื่อเข้าสู่โหมดเมนู

Source Program ภาษาซี

```
#include "DataLog.h"
#include "amPicLib.h"
//============================== Hardware
Driver=======================================
//------------------------------------------------------- Key Matrix
char const keyLayout[17] = {' ', '1', '2', '3', 'U',    // [0][0], [0][1], [0][2], [0][3]
                '4', '5', '6', 'D',
                '7', '8', '9', 'L',
                'C', '0', 'E', 'R'};
int keyNow=0, wKey=0;
#define keyLayoutNow keyLayout[keyNow]
```

```
//------------------------------------------------------- EEPROM 2416
#define i2c_SDA  PIN_C4
#define i2c_SCL  PIN_C3
#use i2c(master, sda=i2c_SDA, scl=i2c_SCL)
void init_i2c() {
output_float(i2c_SCL);
 output_float(i2c_SDA);
}
void write_2464(long address, byte data) {
 i2c_start();
 i2c_write(0xa0);
i2c_write(make8(address, 1));
i2c_write(address);
 i2c_write(data);
 i2c_stop();
 delay_ms(11);
}
byte read_2464(long address) {
byte data;
 i2c_start();
 i2c_write(0xa0);
 i2c_write(make8(address, 1));
i2c_write(address);
 i2c_start();
i2c_write(0xa1);
data = i2c_read(0);
 i2c_stop();
return(data);
}
//------------------------------------------------------------ DS1307 RTC
enum {tSec, tMinute, tHour, tDay, tDate, tMonth, tYear};
int const dsMax[2][3] = {{0x31, 0x12, 0x99}, {0x23, 0x59, 0x59}};
```

```
int const dsMin[2][3] = {{0x01, 0x01, 0x00}, {0x00, 0x00, 0x00}};
void write_DS1307 (byte addr, data) {
i2c_start();
i2c_write(0xd0);
i2c_write(addr);
i2c_write(data);
i2c_stop();
delay_ms(8);
}
int read_DS1307 (byte addr) {
int data;
i2c_start();
i2c_write(0xd0);
i2c_write(addr);
i2c_start();
i2c_write(0xd1);
data = i2c_read(0);
i2c_stop();
return(data);
}
int BCDtoDEC (int bcd) {
return (((bcd>>4)%10)*10) + ((bcd & 0xf)%10);
}
//------------------------------------------------------------------ LCD 16x1
struct lcd_pin_map {
boolean enable;
boolean rs;
boolean rw;
boolean unused;
int     data : 4;
} lcd;
```

```
#byte lcd = 0x08              // port_D
#define lcd_type 2            // 0=5x7, 1=5x10, 2=2 lines
#define lcd_line_two 0x40     // LCD RAM address for the second line
byte const LCD_INIT_STRING[4] = {0x20 | (lcd_type << 2), 0xc, 1, 6};
STRUCT lcd_pin_map const LCD_WRITE = {0,0,0,0,0}; // For write mode all pins are
out
STRUCT lcd_pin_map const LCD_READ = {0,0,0,0,15}; // For read mode data pins are
in
enum {rs_cmd, rs_data};
// lcd command
#define ON_CURSOR    0x0e
#define OFF_CURSOR   0x0c
#define CLEAR_SCREEN 0x01
int nCursor=0;
byte lcd_read_byte() {
byte low, high;
set_tris_d(LCD_READ);
lcd.rw = 1;
delay_cycles(1);
lcd.enable = 1;
delay_cycles(1);
high = lcd.data;
lcd.enable = 0;
delay_cycles(1);
 lcd.enable = 1;
 delay_us(1);
low = lcd.data;
lcd.enable = 0;
set_tris_d(LCD_WRITE);
 return( (high<<4) | low);
}
```

```
void lcd_send_nibble( byte n ) {
 lcd.data = n;
delay_cycles(1);
lcd.enable = 1;
delay_us(2);
lcd.enable = 0;
}
void lcd_send_byte( byte address, byte n ) {
 lcd.rs = 0;
 while ( bit_test(lcd_read_byte(),7) ) ;
lcd.rs = address;
delay_cycles(1);
lcd.rw = 0;
delay_cycles(1);
lcd.enable = 0;
lcd_send_nibble(n >> 4);
lcd_send_nibble(n & 0xf);
}
void lcd_init() {
byte i;
set_tris_d(LCD_WRITE);
 lcd.rs = 0;
lcd.rw = 0;
lcd.enable = 0;
delay_ms(15);
for(i=1;i<=3;++i) {
lcd_send_nibble(3);
 delay_ms(5);
}
 lcd_send_nibble(2);
for(i=0;i<=3;++i) lcd_send_byte(0,LCD_INIT_STRING[i]);
}
```

```
void lcd_clear() {
lcd_send_byte(rs_cmd, CLEAR_SCREEN);
nCursor = 0x00;
}
void lcd_gotox(int x) {
 if (x<9) nCursor=0x00; else nCursor=0x40-8;
 nCursor += x-1;
 lcd_send_byte(rs_cmd, 0x80|nCursor);
}
void lcd_putc( char c) {
if (c=='\n') lcd_gotox(1);
 else {
 lcd_send_byte(rs_data, c);
 nCursor++;
 if (nCursor==0x08) lcd_gotox(9);
if (nCursor==0x80) lcd_gotox(1);
}
}
//=================================================================
declaration
 // system
int sysclk=0, sysclk2=0;
 // parameter
int nowProb;
int nowDateTime[8];
int32 nowTRec[4];
int const trMax[3] = {23, 59, 59};
long const trInc[3] = {3600, 60, 1};
int32 const trSub[3] = {82800, 3540, 59};
short bReadRecord=false, bClearRecord=false;
```

```
 // Result adc
long ResultProb[4];
int prb=0;
 //
enum {regProb,
regtr00, regtr01, regtr02, regtr03,
 regtr10, regtr11, regtr12, regtr13,
regtr20, regtr21, regtr22, regtr23,
regtr30, regtr31, regtr32, regtr33,
recIn00, recIn01, recOut00, recOut01,
recIn10, recIn11, recOut10, recOut11,
recIn20, recIn21, recOut20, recOut21,
 recIn30, recIn31, recOut30, recOut31
};
int const regTRec[4] = {regtr00, regtr10, regtr20, regtr30};
int const recIn[4] = {recIn00, recIn10, recIn20, recIn30};
int const recOut[4] = {recOut00, recOut10, recOut20, recOut30};
int32 recCount[4]={0, 0, 0, 0};
#rom 0x2100 = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
 // variable
int gi, gia[3];
int nRow, nCol;
int32 gll;
 // Internal EEPROM
//=============================================================
Software Function
int32 read32_eeprom(int hiAddr) {
int32 ll;
ll = make32(read_eeprom(hiAddr), read_eeprom(hiAddr+1), read_eeprom(hiAddr+2),
read_eeprom(hiAddr+3));
return ll;
```

```
}
void write32_eeprom(int hiAddr, int32 val) {
write_eeprom(hiAddr+0, make8(val, 3));
 write_eeprom(hiAddr+1, make8(val, 2));
write_eeprom(hiAddr+2, make8(val, 1));
write_eeprom(hiAddr+3, make8(val, 0));
}
long read16_eeprom(int hiAddr) {
long l;
 l = make16(read_eeprom(hiAddr), read_eeprom(hiAddr+1));
return l;
}
void write16_eeprom(int hiAddr, long val) {
 write_eeprom(hiAddr+0, make8(val, 1));
 write_eeprom(hiAddr+1, make8(val, 0));
}
void GetTTime(int32 time, int *h, *m, *s) {
for (*h=0; time>=3600; (*h)++) time-=3600;
for (*m=0; time>=60; (*m)++) time-=60;
*s = time;
}
void CmdRecord(int prob, long val) {
long lIn, lOut, l;
 l = make16(prob<<3, 0);
lIn = read16_eeprom(recIn[prob]);
lOut = read16_eeprom(recOut[prob]);
write_2464((lIn++)+l, make8(val, 1));
 write_2464((lIn++)+l, make8(val, 0));
 lIn &= 0x7ff;
 if (lIn==lOut) {
lOut = (lOut+2) & 0x7ff;
write16_eeprom(recOut[prob], lOut);
```

```
}
write16_eeprom(recIn[prob], lIn);
}
//-------------------------------- Display ----------------------------------
enum {modeTitle,
modeDisp,
modeMenu,
modeMenuDisplay,
modeMenuDateTime,
modeMenuRecEvery
};
int dspMode;
short dspBlink=0;
int dspRefresh=1;
#define RefreshNow dspRefresh=1
// sentence ----
char const txtMenu[3][10] = {"Display", "Date/Time", "Rec Every"};
char const txtMenuDateTime[2][5] = {"Date", "Time"};
void ChangeMode (int mode) {
int i, j;
dspMode = mode;
lcd_clear();
lcd_send_byte(rs_cmd, OFF_CURSOR);
switch (dspMode) {
case modeTitle :
gi = 0;
 break;
case modeMenu :
nRow = 0;
break;
case modeMenuDisplay :
 nRow = nowProb;
```

```
 break;
case modeMenuDateTime :
nRow = 0;
nCol = 0;
gi = 0;
break;
case modeMenuRecEvery :
nRow = 0;
nCol = 0;
gi = 0;
break;
default :
}
RefreshNow;
}
void DisplayRefresh() {
int i;
char c;
dspRefresh = 25;   // 500 mS
dspBlink = ~dspBlink;
switch (dspMode) {
case modeTitle :
gi++;
if (gi<40) {
lcd_clear();
if (gi<17) { lcd_gotox(17-gi); lcd_putc("<<<"); }
if (gi==17) lcd_putc("<<");
if (gi==18) lcd_putc("<");
if ((gi>=5)&&(gi<=17)) lcd_gotox(21-gi);
if (gi>17) lcd_gotox(4);
lcd_putc("DATA LOGGER");
dspRefresh = 5;   // 100 mS
```

```
}
else ChangeMode(modeDisp);
 break;
case modeDisp :
lcd_clear();
printf(lcd_putc, "PROB %u:%4lu Unit", nowProb+1, ResultProb[nowProb]);
if (!dspBlink) {
lcd_gotox(7);
lcd_putc(' ');
}
break;
case modeMenu : i=4; goto MNU1;
case modeMenuDisplay : i=5; goto MNU1;
MNU1:
lcd_gotox(i);
if (dspBlink) c=':'; else c=' ';
lcd_putc(c);
break;
case modeMenuDateTime :
lcd_send_byte(rs_cmd, OFF_CURSOR);
if (!nRow) {
c = '/';
if (!gi) {
gia[0] = nowDateTime[tDate];
gia[1] = nowDateTime[tMonth];
gia[2] = nowDateTime[tYear];
}
}
else {
c = ':';
if (!gi) {
gia[0] = nowDateTime[tHour];
```

```
gia[1] = nowDateTime[tMinute];
gia[2] = nowDateTime[tSec];
}
}
lcd_clear();
        printf(lcd_putc, "%s: %02x%c%02x%c%02x", txtMenuDateTime[nRow],
gia[0], c, gia[1], c, gia[2]);
if (dspBlink) {
if (gi) {
lcd_gotox((nCol*3)+8);
lcd_send_byte(rs_cmd, ON_CURSOR);
}
}
else {
if (!gi) {
lcd_gotox(5);
lcd_putc(' ');
}
}
 break;
case modeMenuRecEvery :
lcd_send_byte(rs_cmd, OFF_CURSOR);
if (!gi) {
lcd_gotox(3);
if (dspBlink) c=':'; else c=' ';
lcd_putc(c);
}
else {
if (dspBlink) {
lcd_gotox((nCol*3)+6);
lcd_send_byte(rs_cmd, ON_CURSOR);
}
```

```
}
break;
}
}
//============================================================
Events Operating
void Key_OnDown() {
char key, c;
int i, j, k;
key = keyLayoutNow;
switch (dspMode) {
case modeTitle :
ChangeMode(modeDisp);
break;
case modeDisp :
switch (key) {
case 'E' : MDP1: ChangeMode(modeMenu); goto MMU1;
case '1' : nowProb = 0; goto MDP2;
case '2' : nowProb = 1; goto MDP2;
case '3' : nowProb = 2; goto MDP2;
case '4' : nowProb = 3; goto MDP2;
MDP2:
write_eeprom(regProb, nowProb);
 dspBlink=false;
RefreshNow;
break;
 }
break;
case modeMenu :
switch (key) {
case 'U' : if (!nRow) nRow=3; nRow--; goto MMU1;
case 'D' : nRow = (nRow+1) % 3;
```

```
MMU1:
lcd_clear();
printf(lcd_putc, "SET:[%u]%s", nRow+1, txtMenu[nRow]);
dspBlink=false;
RefreshNow;
break;
case 'E' :
switch (nRow) {
case 0 : ChangeMode(modeMenuDisplay); goto MMD1;
case 1 : ChangeMode(modeMenuDateTime); dspBlink=false; RefreshNow; break;
case 2 : ChangeMode(modeMenuRecEvery); goto MRE1;
}
break;
case 'C' :
ChangeMode(modeDisp);
default:
}
break;
case modeMenuDisplay :
switch (key) {
case 'U' : if (!nRow) nRow=4; nRow--; goto MMD1;
case 'D' : nRow = (nRow+1) % 4;
MMD1:
lcd_clear();
printf(lcd_putc, "PROB: %u", nRow+1);
dspBlink=false;
RefreshNow;
break;
case 'E' :
nowProb = nRow;
write_eeprom(regProb, nowProb);
case 'C' : ChangeMode(modeMenu); goto MMU1;
```

```
}
 break;
case modeMenuDateTime :
if (!gi) {
switch (key) {
case 'U' : if (!nRow) nRow=2; nRow--; break;
case 'D' : nRow = (nRow+1) % 2; break;
case 'E' : gi = 1; nCol = 0; break;
case 'C' : ChangeMode(modeMenu); goto MMU1;
}
}
else {
switch (key) {
case 'L' : if (!nCol) nCol=3; nCol--; break;
case 'R' : nCol = (nCol+1) % 3; break;
case 'U' :
i = gia[nCol];
if (i<dsMax[nRow][nCol]) {
i++;
if ((i&0x0f)>9) i+=6;
}
else i = dsMin[nRow][nCol];
gia[nCol] = i;
 break;
case 'D' :
i = gia[nCol];
if (i>dsMin[nRow][nCol]) {
i--;
if ((i&0x0f)>9) i-=6;
}
else i = dsMax[nRow][nCol];
 gia[nCol] = i;
```

```
 break;
 case 'E' :
if (!nRow) {
 write_DS1307(tDate, gia[0]);
 write_DS1307(tMonth, gia[1]);
 write_DS1307(tYear, gia[2]);
}
else {
 write_DS1307(tHour, gia[0]);
 write_DS1307(tMinute, gia[1]);
 write_DS1307(tSec, gia[2]);
}
for (i=0; i<7; i++) nowDateTime[i] = read_DS1307(i);
 case 'C' : gi=0; break;
}
}
dspBlink=false;
RefreshNow;
  break;
case modeMenuRecEvery :
if (!gi) {
switch (key) {
case 'U' : if (!nRow) nRow=4; nRow--; goto MRE1;
case 'D' : nRow = (nRow+1) % 4;
MRE1:
GetTTime(nowTRec[nRow], &i, &j, &k);
MRE2:
lcd_clear();
printf(lcd_putc, "P%u: %02u:%02u:%02u", nRow+1, i, j, k);
break;
case 'E' :
gll = nowTRec[nRow];
```

```
 gi = 1;
 nCol = 0;
 goto MRE3;
case 'C' : ChangeMode(modeMenu); goto MMU1;
 }
}
 else {
switch (key) {
case 'L' : if (!nCol) nCol=3; nCol--; break;
case 'R' : nCol = (nCol+1) % 3; break;
 case 'U' :
 GetTTime(gll, &gia[0], &gia[1], &gia[2]);
 if (gia[nCol] < trMax[nCol]) gll+=trInc[nCol]; else gll-=trSub[nCol];
 goto MRE3;
 case 'D' :
GetTTime(gll, &gia[0], &gia[1], &gia[2]);
if (gia[nCol]) gll-=trInc[nCol]; else gll+=trSub[nCol];
MRE3:
GetTTime(gll, &i, &j, &k);
goto MRE2;
case 'E' :
nowTRec[nRow] = gll;
 write16_eeprom(recOut[nRow], 0);
write16_eeprom(recIn[nRow], 0);
recCount[nRow] = 0;
 write32_eeprom(regTRec[nRow], gll);
case 'C' : gi=0; break;
}
}
dspBlink=false;
RefreshNow;
 break;
```

```
default:
 }
}
void Key_OnClick() {
}
void Key_OnFPress() {
Key_OnDown();
wKey = 7;
}
void Key_OnUp() {
}
//=============================================================
Create Events
#define codeReadRecord 'G'
#define codeClearRecord 'H'
void Scan_ReadRecord() {
long lIn, lOut, l;
int i, j, k;
if (bReadRecord) {
for (i=0; i<4; i++) {
l = make16(i<<3, 0);
lIn = read16_eeprom(recIn[i]);
lOut = read16_eeprom(recOut[i]);
while (lOut!=lIn) {
j = read_2464((lOut++)+l);
j = (j&3) + (i<<2);
if (j<10) putc(j+'0'); else putc(j+('A'-10));
 j = read_2464((lOut++)+l);
k = j>>4;
j &= 0x0f;
if (k<10) putc(k+'0'); else putc(k+('A'-10));
if (j<10) putc(j+'0'); else putc(j+('A'-10));
```

```
 putc('x');
lOut &= 0x7ff;
}
}
putc('z');
bReadRecord = false;
}
if (bClearRecord) {
for (i=0; i<4; i++) {
write16_eeprom(recOut[i], 0);
write16_eeprom(recIn[i], 0);
}
 bClearRecord = false;
}
}
void Scan_Recording() {
int i;
for (i=0; i<4; i++) {
if (!recCount[i]) recCount[i]=nowTRec[i];
if (recCount[i]) {
recCount[i]--;
if (!recCount[i]) CmdRecord(i, ResultProb[i]);
}
}
}
void Scan_Timer() {
int i;
 i = read_DS1307(tSec);
if (i!=nowDateTime[tSec]) {
nowDateTime[tSec] = i;
Scan_Recording();
}
```

```
if (!i) for (i=1; i<7; i++) nowDateTime[i] = read_DS1307(i);
}

void Scan_ReadProb() {
int const ChAdc[4] = {0, 1, 4, 5};
ResultProb[prb] = read_adc();
 prb = (prb+1) % 4;
set_adc_channel(ChAdc[prb]);
}

void Scan_DisplayRefresh() {
if (dspRefresh) {
dspRefresh--;
if (!dspRefresh) DisplayRefresh();
}
}
#byte TRISB = 0x86
void Scan_KeyMatrix() {
int i, j, keyResult=0;
for (i=0; i<4; i++) {
bit_clear(TRISB, i+4);
bit_clear(Port_B, i+4);
for (j=0; j<4; j++) {
if (!bit_test(port_B, j)) {
 keyResult = (i*4)+j+1;
i = 4;
break;
}
}
TRISB = 0xff;
 }
 if (keyResult != keyNow) {
```

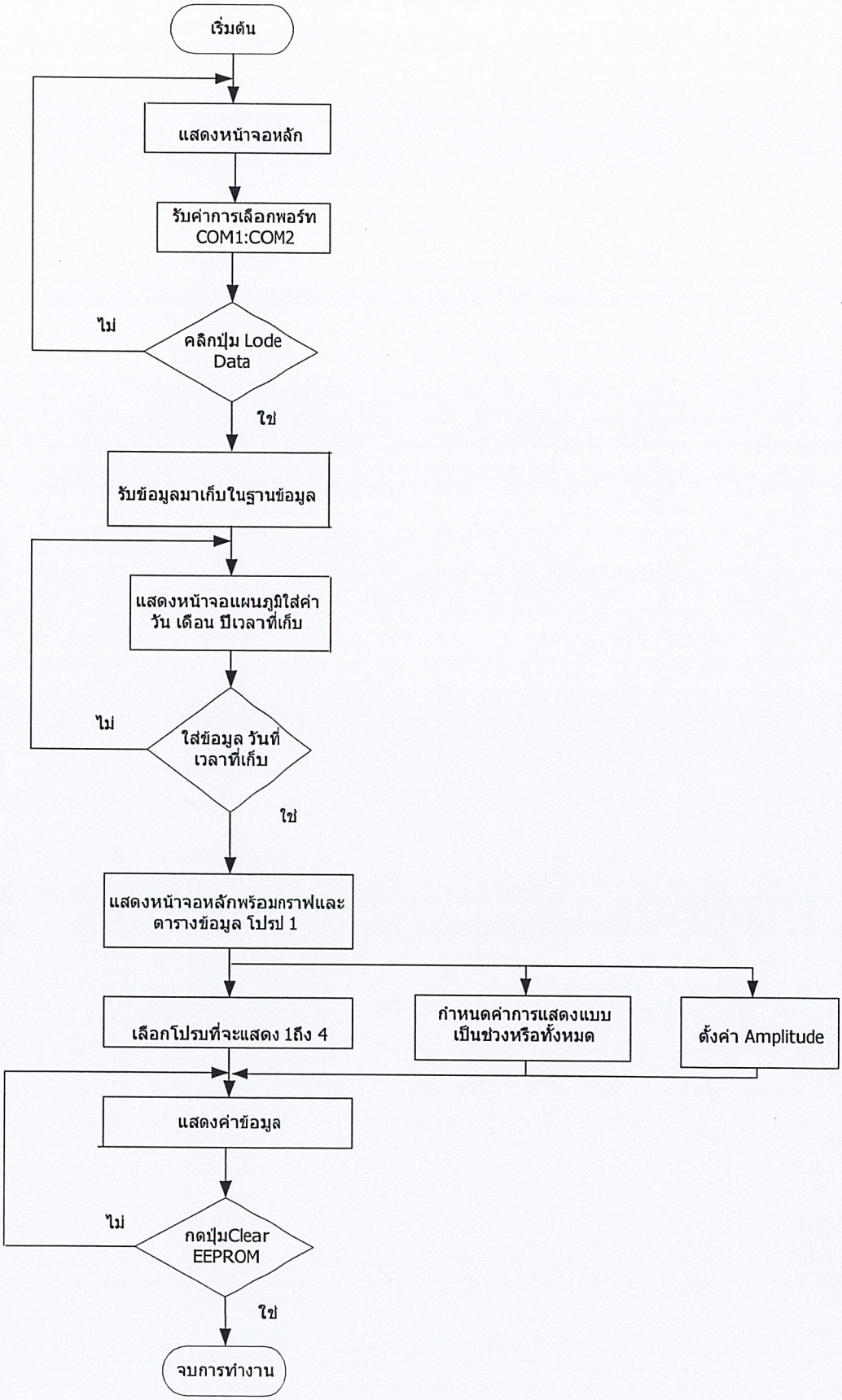
```
 if (keyNow) {
if (wKey<50) {
if (wKey) Key_OnClick();
Key_OnUp();
keyNow = 0;
}
wKey = 0;
}
if (keyResult) {
 keyNow = keyResult;
wKey = 52;              // 1 Sec
}
}
}
void Scan_KeyOnFPress() {
if (wKey) {
 wKey--;
if (wKey==50) Key_OnDown();
if (!wKey) Key_OnFPress();
}
}
//=================================================================
Multitasking
Task_successive() {
Scan_KeyMatrix();
}
Task_20mS() {
Scan_KeyOnFPress();
Scan_DisplayRefresh();
}
Task_100mS() {
Scan_ReadProb();
```

```
Scan_ReadRecord();
}
Task_500mS() {
Scan_Timer();
}
Task_1S() {
}
//============================================================ isr
#int_RDA
RDA_isr() {
char c;
c = getc();
switch (c) {
case codeReadRecord : bReadRecord = true; break;
 case codeClearRecord : bClearRecord = true; break;
}
}
#int_RTCC
RTCC_isr() {
sysclk++;
}
//=================================================================
========== Main
void main() {
int i;
  set_tris_c(0xbf);
  port_b_pullups(TRUE);
setup_adc_ports(ANALOG_NOT_RE1_RE2_REF_RA3_RA2);
setup_adc(ADC_CLOCK_DIV_8);
set_adc_channel(0);
setup_spi(FALSE);
setup_psp(PSP_DISABLED);
```

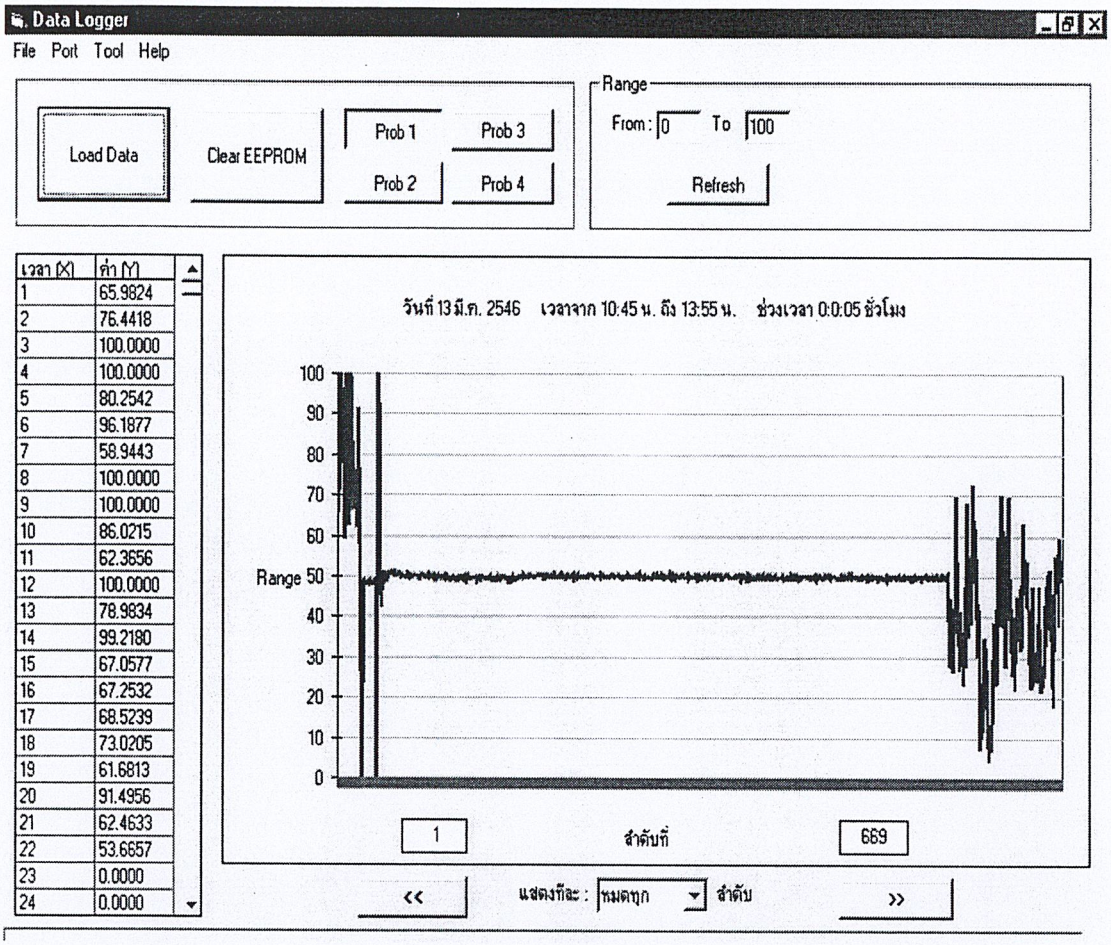
```
setup_counters(RTCC_INTERNAL,RTCC_DIV_4);
setup_timer_1(T1_DISABLED);
setup_timer_2(T2_DISABLED,0,1);
setup_ccp1(CCP_OFF);
setup_ccp2(CCP_OFF);
enable_interrupts(INT_RTCC);
enable_interrupts(INT_RDA);
enable_interrupts(global);
 // Initial
lcd_init();
lcd_clear();
 init_i2c();
for (i=0; i<7; i++) nowDateTime[i] = read_DS1307(i);
nowProb = read_eeprom(regProb);
for (i=0; i<4; i++) nowTRec[i] = read32_eeprom(regTRec[i]);
ChangeMode(modeTitle);
//  prmDisplay = read_eeprom(eepDisplay);
  while (true) {
 Task_successive();
 if (sysclk >= 20) {
sysclk %= 20;
Task_20mS();
sysclk2++;
if (!(sysclk2 % 5)) Task_100mS();
if (!(sysclk2 % 25)) Task_500mS();
 if (!(sysclk2 % 50)) {
sysclk2 =0;
Task_1S();
 }
}
}
}
```

5.3.2 แผนผังลำดับการทำงานของโปรแกรม Visual Basic

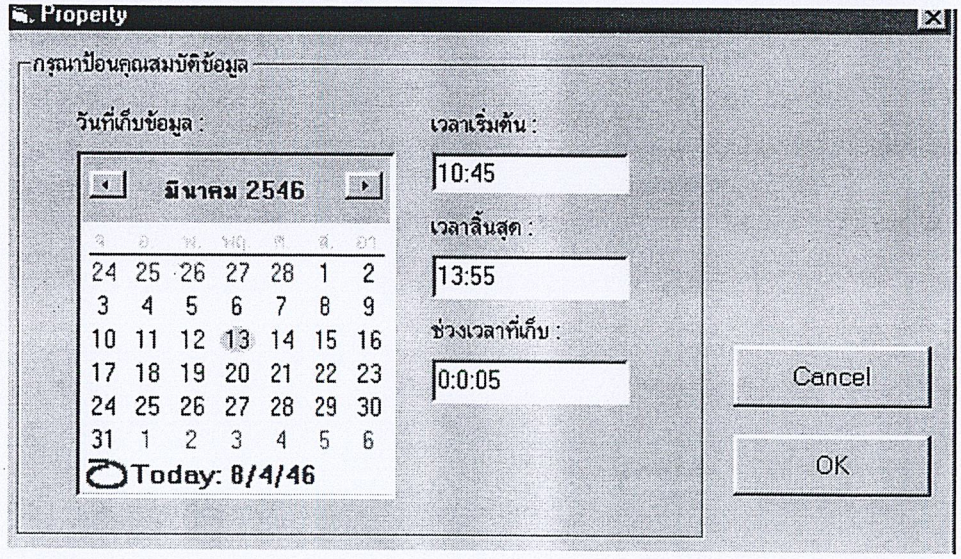
ในส่วนของโปรแกรมแสดงผลนั้นจะเขียนโดยโปรแกรม Visual Basic โดยจะเชื่อมต่อกับเครื่องบันทึกข้อมูลผ่านทางพอร์ทอนุกรมโดยใช้อัตราบอดเรทที่ 9600 บิทต่อวินาทีเมื่อเริ่มทำงานโปรแกรมจะแสดงหน้าจอหลักขึ้นมาเพื่อรับคำสั่งในการโหลดข้อมูล และเลือกพอร์ทจากผู้ใช้แล้วจึงส่งคำสั่งไปยังตัวไมโครคอนโทรลเลอร์ เพื่อสั่งให้ส่งบิทข้อมูลที่บันทึกไว้ในหน่วยความจำมายังส่วนของฐานข้อมูลในโปรแกรม Visual Basic แล้วจะแสดงหน้าจอแผนภูมิเพื่อรับข้อมูลวันที่เก็บเวลาที่เก็บช่วงเวลาในการเก็บจากนั้นจะนำค่าทั้งหมดมาแสดงเป็นกราฟ และตารางข้อมูลที่หน้าจอโดยจะมีการตรวจสอบว่าต้องการให้แสดงที่โปรบใดและจะกำหนด Amplitude ทางด้านแกน y ได้โดยการใส่ค่าลงในช่องรับค่าแล้วนำค่าไปคำนวณและปรับขนาดของ Amplitude ตามค่าที่รับเข้ามาและตรวจสอบการเลือกแสดงแถบข้อมูลให้แสดงทั้งหมด หรือแสดงเป็นช่วงแล้วจะตรวจสอบว่ามีการสั่งให้เคลียร์ค่าข้อมูลในหน่วยความจำของเครื่องบันทึกข้อมูลก่อนที่จะทำการ Disconnect หรือจบการทำงานแสดงแผนผังลำดับการทำงานไว้ในภาพที่ 5.4



ภาพที่ 5.4 แผนผังลำดับการทำงานของโปรแกรม Visual Basic



ภาพที่ 5.5 หน้าจอหลักของโปรแกรมแสดงผล



ภาพที่ 5.6 หน้าจอแผนภูมิวันที่และเวลาที่เก็บ

ภาพที่ 5.7 เครื่องบันทึกข้อมูลแบบพกพา

รายละเอียดและลักษณะภายนอก

ขนาด ความกว้าง 140 มม.

ความยาว 153 มม.

ความสูง 67 มม.

น้ำหนักโดยประมาณ 525 กรัม

บทที่ 6

การทดสอบการทำงาน

6.1 ทดสอบกับชุดทดลองการควบคุมอุณหภูมิ

1. ต่อวงจรเพื่อวัดค่าสัญญาณเอาท์พุทที่เป็นแรงดันอนาลอก 1 ถึง 5V จากชุดทดลองการควบคุมอุณหภูมิตามภาพที่ 6.1 โดยใช้ช่องสัญญาณอินพุทช่องที่ 1 ของเครื่องบันทึกข้อมูล โดยตั้งค่าการทำงานของชุดทดลองการควบคุมอุณหภูมิไว้ดังนี้

   ตั้งที่ตัวคอนโทรลเลอร์ของชุดควบคุมอุณหภูมิ

   Mode Auto

   Range 0 ถึง 50 °C

   SV 50 °C

   PV เริ่มจากอุณหภูมิห้องประมาณ 28 °C

2. ทำการตั้งค่าการทำงานของเครื่องบันทึกข้อมูลดังนี้

   เปิดเครื่องเริ่มการทำงาน เข้าโหมด

   Menu Set Probe 1 Display (แสดงหน้าจอการวัดค่า Probe 1)

   Menu REC EVERY Probe 1 Time 00:01:00 (ให้เก็บค่าทุกๆ 1 นาที)

3. บันทึกค่าข้อมูลโดยจับเวลาให้เครื่องทำการเก็บค่าเป็นเวลา 2 ชั่วโมง

4. นำเครื่องบันทึกข้อมูลมาต่อเข้าที่พอร์ทอนุกรม 1ของเครื่องคอมพิวเตอร์เพื่อจะโหลดข้อมูลมาวิเคราะห์โดยเปิดโปรแกรม Visual Basic นำแผ่นโปรแกรมที่ใช้ Run มาเปิดจะปรากฏหน้าจอหลักขึ้นมา

5. คลิกที่ปุ่ม Port เลือก Port 1 จากนั้นคลิกปุ่ม Lode Data

6. เครื่องจะแสดงหน้าจอแผนภูมิวันที่และเวลา ทำการป้อนข้อมูลดังนี้

   ตั้ง วันที่เก็บ 15 / มีนาคม / 2546

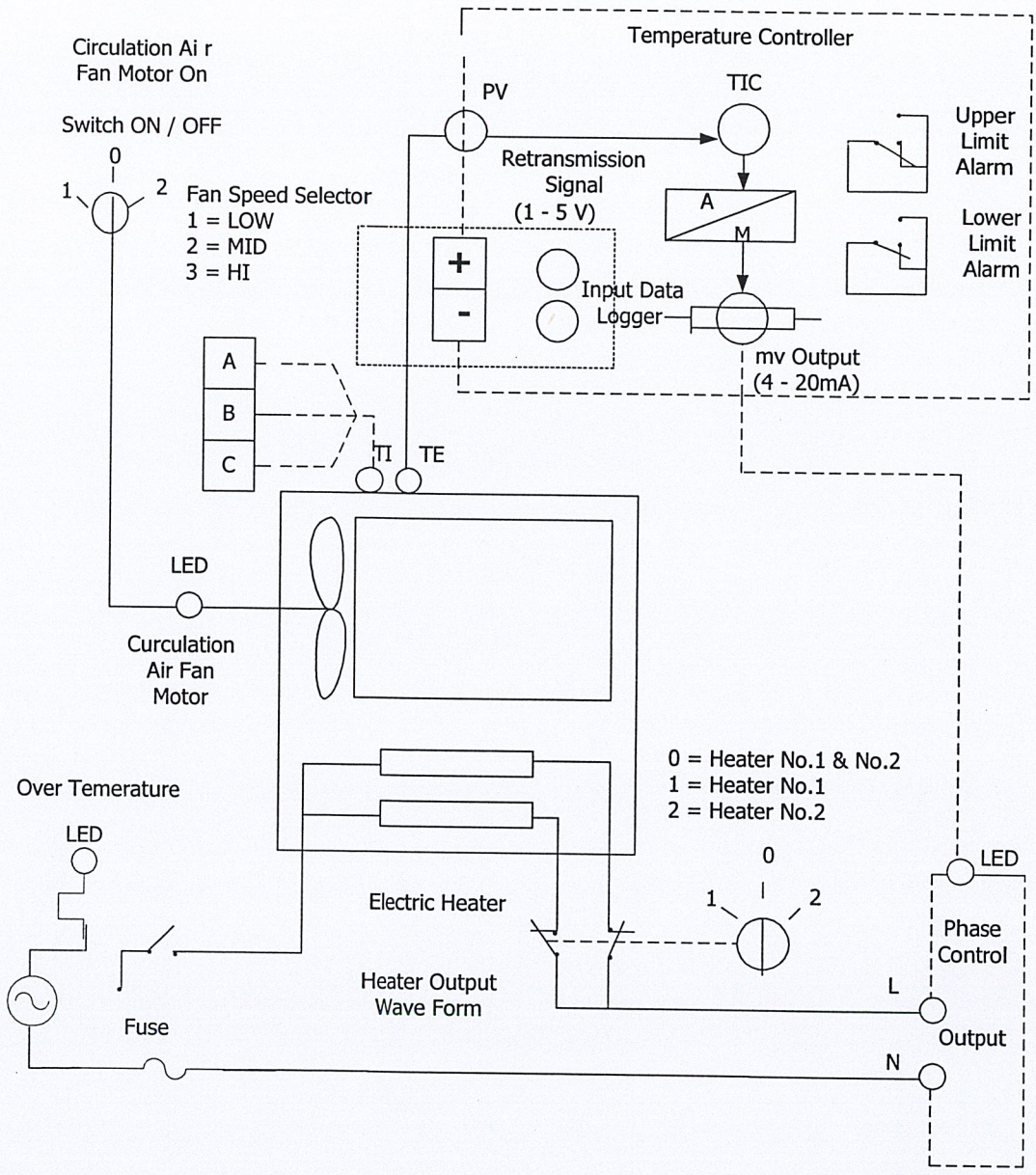
   เวลาที่เก็บ 13.15 น. ถึงเวลา 15.15 น.

   ช่วงเวลาที่เก็บ 00:01:00 น. (ทุก 1 นาที)

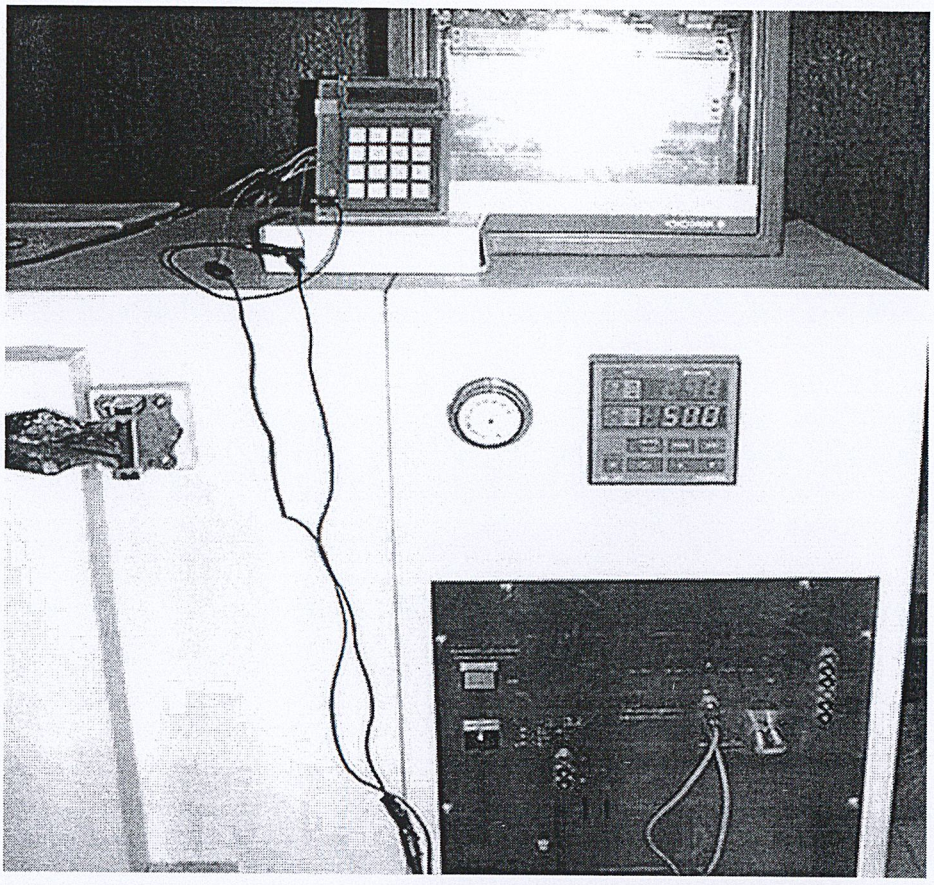
จากนั้นก็คลิกปุ่ม OK จะแสดงหน้าจอหลักกราฟข้อมูลและตารางข้อมูลที่รับเข้ามาซึ่งมีข้อมูลเข้ามาทั้งหมด 120 ข้อมูล

7. ตั้งค่า Amplitude ทางแกน Y ที่ค่า 0 ถึง 100 จากนั้นคลิกปุ่ม Refresh

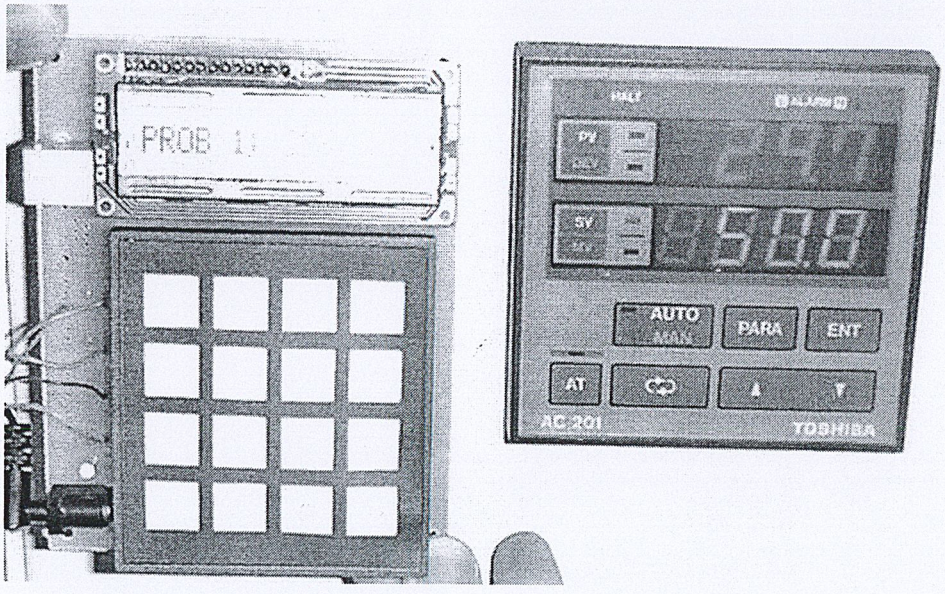
8. ตั้งค่าการอ่านกราฟแสดงข้อมูลแบบทั้งหมดและแบบเพิ่มทีละ 50 จะได้กราฟแสดงผลของข้อมูลดังภาพที่ 6.5 ถึง 6.8 ตามลำดับ



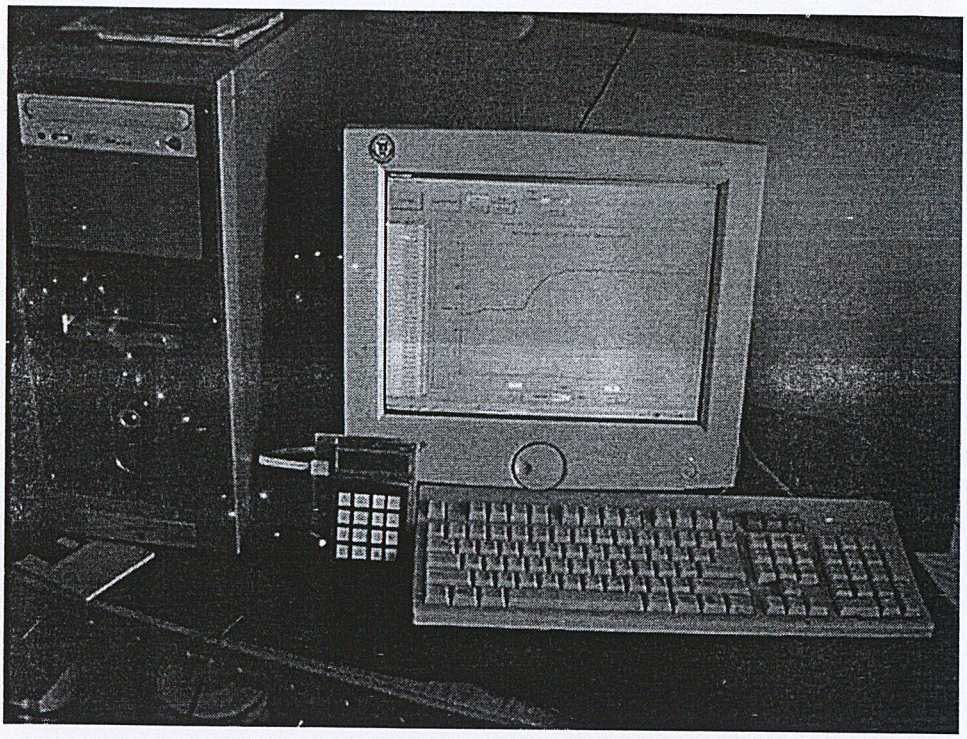
ภาพที่ 6.1 วงจรการต่อวัดของเครื่องบันทึกข้อมูลกับชุดทดลองการควบคุมอุณหภูมิ



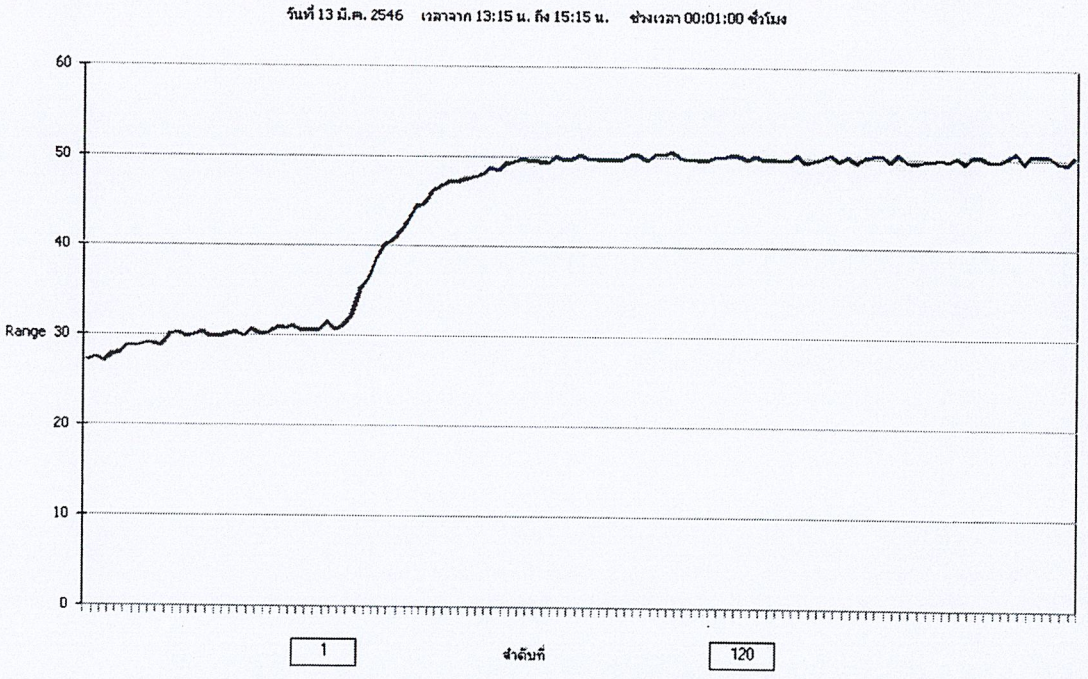
ภาพที่ 6.2 การเชื่อมต่อเครื่องบันทึกข้อมูลแบบพกพากับชุดทดลองการควบคุมอุณหภูมิ



ภาพที่ 6.3 หน้าจอของเครื่องบันทึกข้อมูล ขณะที่กำลังวัดเปรียบเทียบกับหน้าจอของของตัวคอนโทรลเลอร์ในชุดทดลองการควบคุมอุณหภูมิ



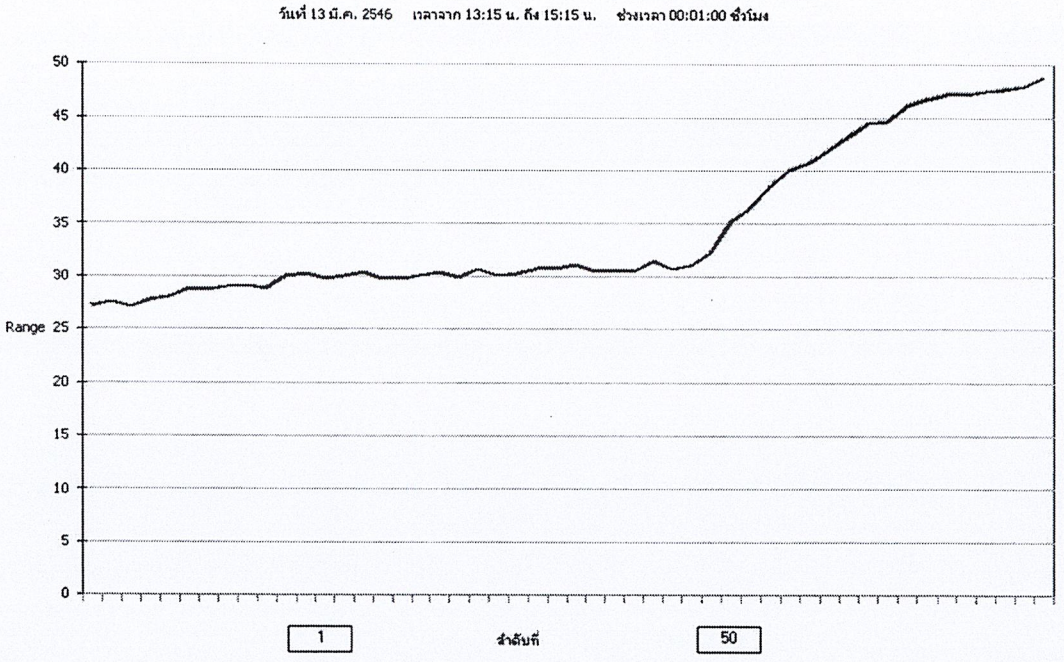
ภาพที่ 6.4 การแสดงข้อมูลด้วยโปรแกรม Visual Basic



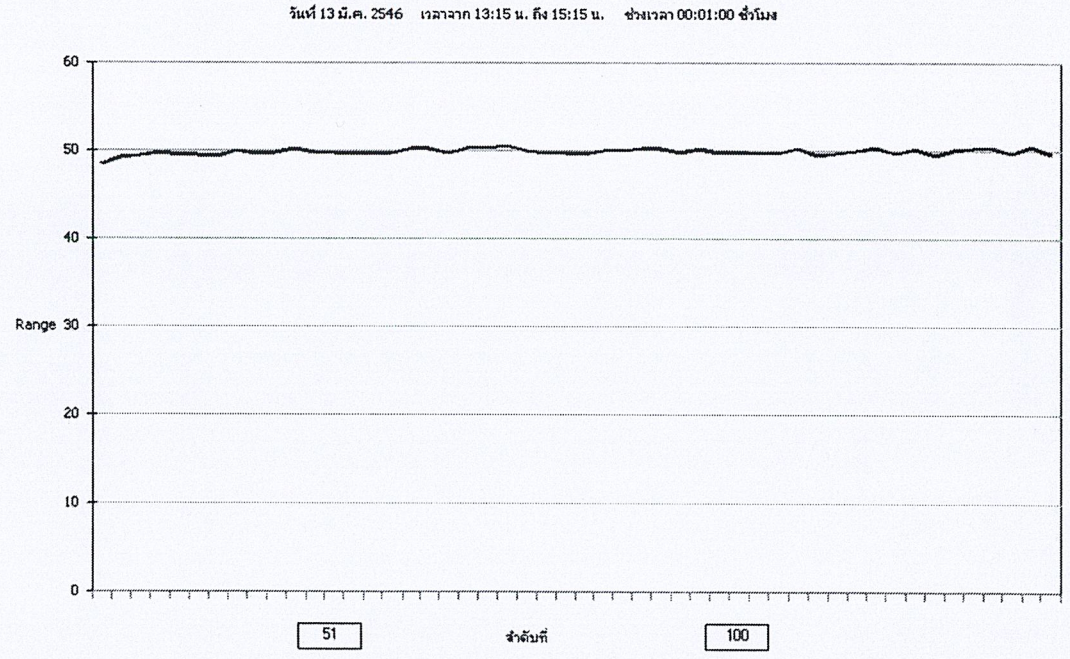
ภาพที่ 6.5 กราฟแสดงข้อมูลที่บันทึกได้ทั้งหมด

ตารางที่ 6.1 ค่าข้อมูลที่วัดได้ทั้งหมด

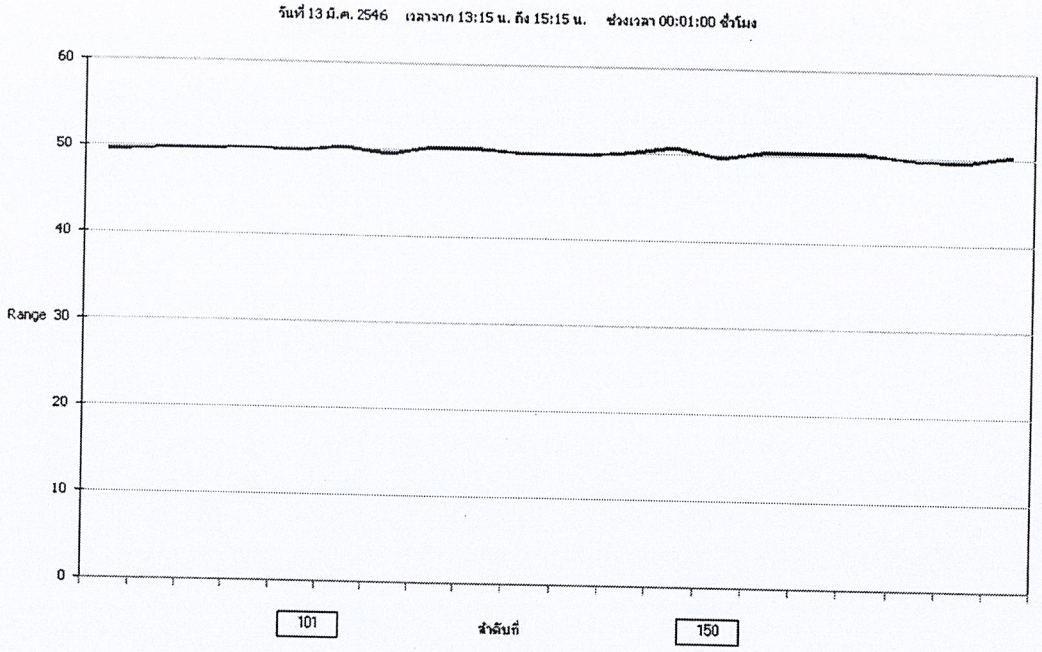
ลำดับที่	ค่าที่วัดได้	ลำดับที่	ค่าที่วัดได้	ลำดับที่	ค่าที่วัดได้	ลำดับที่	ค่าที่วัดได้
1	27.1750	31	31.6940	61	50.2444	91	50.4399
2	27.5660	32	31.0850	62	49.8534	92	49.7556
3	27.0772	33	32.2581	63	49.7556	93	50.2444
4	27.7165	34	35.1906	64	49.6579	94	49.5601
5	28.0547	35	36.4614	65	49.6579	95	50.1466
6	28.8368	36	38.5142	66	49.7566	96	50.3421
7	28.7390	37	40.0782	67	50.3421	97	50.4399
8	29.0323	38	40.7625	68	50.2444	98	49.7556
9	29.1300	39	41.9355	69	49.7556	99	50.5376
10	28.8368	40	43.2063	70	50.3421	100	49.7556
11	30.0098	41	44.4770	71	50.3421	101	49.5601
12	30.3030	42	44.6725	72	50.6554	102	49.7556
13	29.8143	43	46.1388	73	50.0489	103	49.7556
14	30.0098	44	46.7253	74	49.7556	104	50.0489
15	30.4008	45	47.2141	75	49.7556	105	49.7556
16	29.8143	46	47.2141	76	49.6579	106	50.2444
17	29.8143	47	47.5073	77	50.0489	107	49.5601
18	30.1075	48	47.7028	78	50.0489	108	50.4399
19	30.4008	49	47.9961	79	50.3424	109	50.2444
21	30.6940	51	48.4848	81	49.8534	111	49.7556
22	30.1075	52	49.2669	82	50.2444	112	50.1466
23	30.3030	53	49.4624	83	49.8534	113	50.8309
24	30.7918	54	49.4634	84	49.9511	114	49.6579
25	30.7918	55	49.5601	85	49.7556	115	50.5376
26	31.0850	56	49.5601	86	49.7556	116	50.4399
27	30.4985	57	49.3646	87	50.3421	117	50.4399
28	30.4985	58	50.0489	88	49.5601	118	50.7556
29	30.4985	59	49.6579	89	49.7556	119	49.5601
30	31.4761	60	49.8534	90	50.0489	120	50.3421



ภาพที่ 6.6 กราฟแสดงข้อมูลช่วง 0 ถึง 50



ภาพที่ 6.7 กราฟแสดงข้อมูลช่วง 50 ถึง 100



ภาพที่ 6.8 กราฟแสดงข้อมูลช่วง 100 ถึง 120

6.2 สรุปผลการทดลอง

จากการทดลองทำการวัดค่าในกระบวนการ(PV) ของชุดทดลองการควบคุมอุณหภูมิ โดยตั้งค่าเป้าหมาย (Set point) ที่ 50 °C เมื่ออุณหภูมิเริ่มต้นจากอุณหภูมิห้องมีค่าประมาณ 20 °C สังเกตค่าทางแกน y ของกราฟที่ได้จากเครื่องบันทึกข้อมูลจะมีค่าเริ่มต้นที่ประมาณ 28 เปลี่ยนแปลงเพิ่มขึ้นไปเรื่อยๆ จนถึงค่าที่ 50 และทางด้านแกน X จะมีค่าจากลำดับที่ 0 ถึง 50 เมื่อตั้งค่าช่วงเวลาที่วัดไว้ทุกๆ 1 นาที นั่นคือแต่ละขีดของกราฟตามแกน X จะมีค่าเป็น 1 นาที นั่นคืออุณหภูมิจะเพิ่มจาก 28 °C ไปเป็นเวลาประมาณ 50 นาทีก็จะรักษาค่าอุณหภูมิไว้ที่ 50 °C ซึ่งค่าทางด้านแกน y ได้จากการตั้งค่า Amplitude ไว้ที่ 0 ถึง 100 อุณหภูมิจะรักษาระดับไว้ที่ 50 °C ตามการทำงานของคอนโทรลเลอร์ที่อยู่ภายในชุดทดลองการควบคุมอุณหภูมิที่ตั้งค่าค่าเป้าหมายไว้ที่ 50 °C รูปกราฟหลังจากเวลาผ่านไป 50 นาที จึงมีลักษณะคล้ายเป็นเส้นตรงที่ระดับ 50 ซึ่งถือว่าตรงกับค่าเป้าหมายตั้งไว้และอ่านได้จากหน้าจอของตัวคอนโทรลเลอร์ของชุดควบคุมอุณหภูมิ

บทที่ 7
บทสรุปและแนวทางการพัฒนา

7.1 สรุปผลการทำงาน

จากการทดสอบการทำงานสามารถสรุปได้ดังนี้

1. วงจรแปลงสัญญาณอนาลอกเป็นดิจิตอลสามารถแปลงค่าได้ถูกต้อง ที่ความละเอียด 10 บิท

2. สามารถรับข้อมูลอนาลอกอินพุทแรงดัน 1 - 5 โวลท์พร้อมกันทั้ง 4 ช่องสัญญาณพร้อมทั้งตั้งค่าเวลาจากคีย์สวิทช์และเก็บค่าข้อมูลไว้ในหน่วยความจำของเครื่องได้

3. การอ่านเขียนข้อมูลกับหน่วยความจำ EEPROM สามารถอ่านเขียนได้ถูกต้อง

4. สามารถแสดงข้อมูลตามค่าเวลาที่เก็บข้อมูลได้เมื่อนำมาโหลดเข้าเครื่องคอมพิวเตอร์

5. สามารถแสดงผลในรูปแบบของกราฟและตารางค่าของข้อมูลได้

6. สามารถนำรูปกราฟที่ได้พิมพ์ออกทางเครื่องพิมพ์(Printer) ได้

7.2 ปัญหา

1. การทำงานของเครื่องยังไม่เป็นอัตโนมัติ คือไม่สามารถตั้งเวลาเริ่มต้นการทำงานสิ้นสุดการทำงานได้ด้วยตัวเอง

2. หน่วยความจำยังไม่มากพอจึงเก็บข้อมูลได้ไม่มาก

3. ยังไม่สามารถเคลียร์หน่วยความจำบนตัวเครื่องได้ต้องนำมาต่อกับคอมพิวเตอร์เพื่อที่จะเคลียร์ข้อมูล

7.3 แนวทางการพัฒนา

1. สามารถตั้งค่าเวลาการทำงานและสิ้นสุดการทำงานแบบอัตโนมัติได้และมีเสียงเตือนเมื่อสิ้นสุดการทำงานตามค่าเวลาที่ตั้งไว้และมีปุ่มรีเซ็ตค่าข้อมูลบนเครื่อง

2. เพิ่มช่องสัญญาณในการวัดให้สามารถวัดค่าอินพุทที่เป็นกระแสได้

3. เพิ่มพื้นที่ของหน่วยความจำให้เก็บข้อมูลได้มากขึ้น

4. เพิ่มความสามารถในการเก็บข้อมูลของโปรแกรมคอมพิวเตอร์เช่นจัดเก็บเป็นฐานข้อมูล (Data Base)

บรรณานุกรม

1. กิตติ ภัคคีวัฒนะกุล. "Visual Basic 6 ฉบับโปรแกรมเมอร์", พิมพ์ครั้งที่ 7.บริษัทเคทีพี คอมพ์แอนด์คอนซัลท์ จำกัด. 2543.
2. ชัยวัฒน์ ลิ้มพรจิตวิไล. "เรียนรู้การเชื่อมต่อไอซีกับอุปกรณ์ภายนอกผ่านพอร์ทอนุกรม", บริษัทอินโนเวตีฟ เอ็กเฟอร์เมนต์ จำกัด. 2542
3. ธาริน สิทธิธรรมธารี. "ฉบับเพื่อการใช้งานจริง", พิมพ์ครั้งที่ 6. บริษัทซัคเซสมีเดียจำกัด. 2544.
4. ธีรวัฒน์ ประกอบผล. "การประยุกต์ใช้งานไมโครคอนโทรลเลอร์", สมาคมส่งเสริมเทคโนโลยี (ไทย-ญี่ปุ่น). พิมครั้งที่ 3. 2542.
5. นพพณ เลิศชูวงศา. "อุปกรณ์รวบรวมข้อมูลเคลื่อนที่", วิทยานิพนธ์ภาควิชาโทรคมนาคม คณะวิศวกรรมศาสตร์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง. 2541.
6. สัจจะ โสรุ่งรจีวร. "คู่มือการเขียนโปรแกรมและใช้งาน Visual Basic 6", พิมพ์ครั้งที่1. สำนักพิมพ์อินโพเพรส. 2544.
7. Pei An, Dr. "Stand-alone data logger", page 194-200. Electronics World, March 1998
8. "PIC16F8X", DS30430C. Microchip Technology Incorporated. 1998.
9. "PIC16F87X", DS30292B. Microchip Technology Incorporated.1999.
10. "The I^2C-Bus Specification", Phillips Semiconductors. 2000.

ภาคผนวก

MICROCHIP

PIC16F87X

28/40-pin 8-Bit CMOS FLASH Microcontrollers

Devices Included in this Data Sheet:

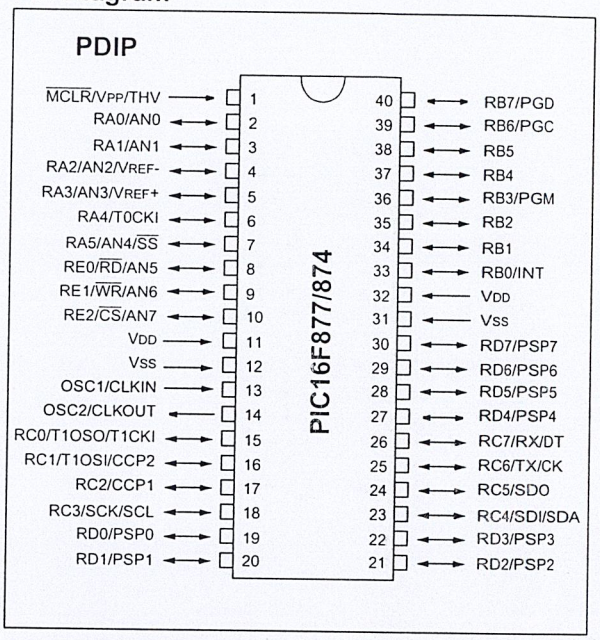
- PIC16F873
- PIC16F874
- PIC16F876
- PIC16F877

Microcontroller Core Features:

- High-performance RISC CPU
- Only 35 single word instructions to learn
- All single cycle instructions except for program branches which are two cycle
- Operating speed: DC - 20 MHz clock input
  DC - 200 ns instruction cycle
- Up to 8K x 14 words of FLASH Program Memory,
  Up to 368 x 8 bytes of Data Memory (RAM)
  Up to 256 x 8 bytes of EEPROM data memory
- Pinout compatible to the PIC16C73B/74B/76/77
- Interrupt capability (up to 14 sources)
- Eight level deep hardware stack
- Direct, indirect and relative addressing modes
- Power-on Reset (POR)
- Power-up Timer (PWRT) and
  Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code-protection
- Power saving SLEEP mode
- Selectable oscillator options
- Low-power, high-speed CMOS FLASH/EEPROM technology
- Fully static design
- In-Circuit Serial Programming™ (ICSP) via two pins
- Single 5V In-Circuit Serial Programming capability
- In-Circuit Debugging via two pins
- Processor read/write access to program memory
- Wide operating voltage range: 2.0V to 5.5V
- High Sink/Source Current: 25 mA
- Commercial and Industrial temperature ranges
- Low-power consumption:
  - < 2 mA typical @ 5V, 4 MHz
  - 20 μA typical @ 3V, 32 kHz
  - < 1 μA typical standby current

Pin Diagram

PDIP

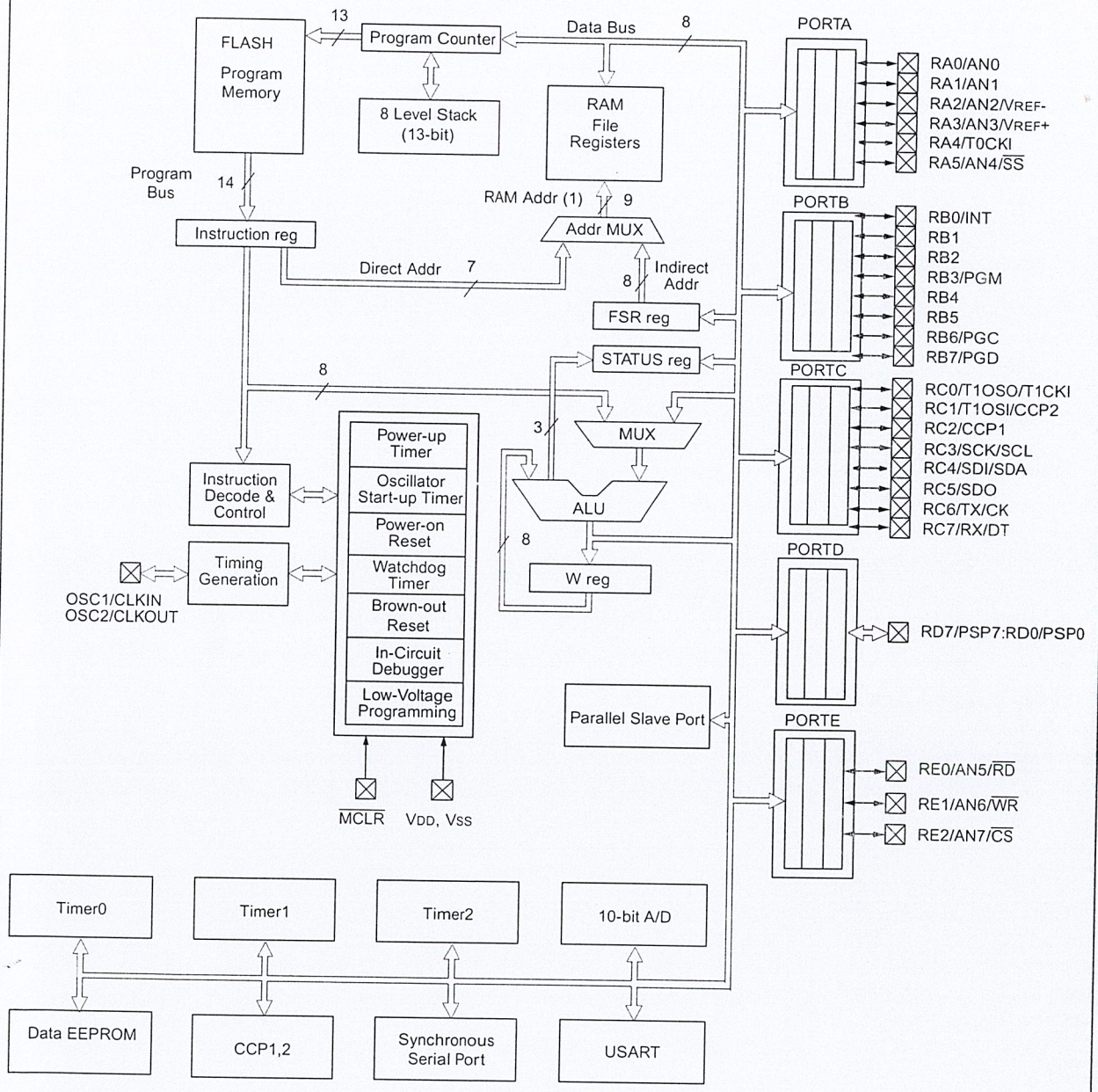


Peripheral Features:

- Timer0: 8-bit timer/counter with 8-bit prescaler
- Timer1: 16-bit timer/counter with prescaler, can be incremented during sleep via external crystal/clock
- Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler
- Two Capture, Compare, PWM modules
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10-bit
- 10-bit multi-channel Analog-to-Digital converter
- Synchronous Serial Port (SSP) with SPI™ (Master Mode) and I^2C™ (Master/Slave)
- Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection
- Parallel Slave Port (PSP) 8-bits wide, with external RD, WR and CS controls (40/44-pin only)
- Brown-out detection circuitry for Brown-out Reset (BOR)

FIGURE 1-2: PIC16F874 AND PIC16F877 BLOCK DIAGRAM

Device	Program FLASH	Data Memory	Data EEPROM
PIC16F874	4K	192 Bytes	128 Bytes
PIC16F877	8K	368 Bytes	256 Bytes



Note 1: Higher order bits are from the STATUS register.

TABLE 1-2: PIC16F874 AND PIC16F877 PINOUT DESCRIPTION

Pin Name	DIP Pin#	PLCC Pin#	QFP Pin#	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	13	14	30	I	ST/CMOS(4)	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	14	15	31	O	—	Oscillator crystal output. Connects to crystal or resonator in crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.
$\overline{MCLR}$/VPP/THV	1	2	18	I/P	ST	Master clear (reset) input or programming voltage input or high voltage test mode control. This pin is an active low reset to the device.
						PORTA is a bi-directional I/O port.
RA0/AN0	2	3	19	I/O	TTL	RA0 can also be analog input0
RA1/AN1	3	4	20	I/O	TTL	RA1 can also be analog input1
RA2/AN2/VREF-	4	5	21	I/O	TTL	RA2 can also be analog input2 or negative analog reference voltage
RA3/AN3/VREF+	5	6	22	I/O	TTL	RA3 can also be analog input3 or positive analog reference voltage
RA4/T0CKI	6	7	23	I/O	ST	RA4 can also be the clock input to the Timer0 timer/counter. Output is open drain type.
RA5/$\overline{SS}$/AN4	7	8	24	I/O	TTL	RA5 can also be analog input4 or the slave select for the synchronous serial port.
						PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	33	36	8	I/O	TTL/ST(1)	RB0 can also be the external interrupt pin.
RB1	34	37	9	I/O	TTL	
RB2	35	38	10	I/O	TTL	
RB3/PGM	36	39	11	I/O	TTL	RB3 can also be the low voltage programming input
RB4	37	41	14	I/O	TTL	Interrupt on change pin.
RB5	38	42	15	I/O	TTL	Interrupt on change pin.
RB6/PGC	39	43	16	I/O	TTL/ST(2)	Interrupt on change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	40	44	17	I/O	TTL/ST(2)	Interrupt on change pin or In-Circuit Debugger pin. Serial programming data.
						PORTC is a bi-directional I/O port.
RC0/T1OSO/T1CKI	15	16	32	I/O	ST	RC0 can also be the Timer1 oscillator output or a Timer1 clock input.
RC1/T1OSI/CCP2	16	18	35	I/O	ST	RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 output.
RC2/CCP1	17	19	36	I/O	ST	RC2 can also be the Capture1 input/Compare1 output/PWM1 output.
RC3/SCK/SCL	18	20	37	I/O	ST	RC3 can also be the synchronous serial clock input/output for both SPI and I^2C modes.
RC4/SDI/SDA	23	25	42	I/O	ST	RC4 can also be the SPI Data In (SPI mode) or data I/O (I^2C mode).
RC5/SDO	24	26	43	I/O	ST	RC5 can also be the SPI Data Out (SPI mode).
RC6/TX/CK	25	27	44	I/O	ST	RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	26	29	1	I/O	ST	RC7 can also be the USART Asynchronous Receive or Synchronous Data.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as an external interrupt.
2: This buffer is a Schmitt Trigger input when used in serial programming mode.
3: This buffer is a Schmitt Trigger input when configured as general purpose I/O and a TTL input when used in the Parallel Slave Port mode (for interfacing to a microprocessor bus).
4: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

TABLE 1-2: PIC16F874 AND PIC16F877 PINOUT DESCRIPTION (CONTINUED)

Pin Name	DIP Pin#	PLCC Pin#	QFP Pin#	I/O/P Type	Buffer Type	Description
						PORTD is a bi-directional I/O port or parallel slave port when interfacing to a microprocessor bus.
RD0/PSP0	19	21	38	I/O	ST/TTL(3)	
RD1/PSP1	20	22	39	I/O	ST/TTL(3)	
RD2/PSP2	21	23	40	I/O	ST/TTL(3)	
RD3/PSP3	22	24	41	I/O	ST/TTL(3)	
RD4/PSP4	27	30	2	I/O	ST/TTL(3)	
RD5/PSP5	28	31	3	I/O	ST/TTL(3)	
RD6/PSP6	29	32	4	I/O	ST/TTL(3)	
RD7/PSP7	30	33	5	I/O	ST/TTL(3)	
						PORTE is a bi-directional I/O port.
RE0/$\overline{RD}$/AN5	8	9	25	I/O	ST/TTL(3)	RE0 can also be read control for the parallel slave port, or analog input5.
RE1/$\overline{WR}$/AN6	9	10	26	I/O	ST/TTL(3)	RE1 can also be write control for the parallel slave port, or analog input6.
RE2/$\overline{CS}$/AN7	10	11	27	I/O	ST/TTL(3)	RE2 can also be select control for the parallel slave port, or analog input7.
VSS	12,31	13,34	6,29	P	—	Ground reference for logic and I/O pins.
VDD	11,32	12,35	7,28	P	—	Positive supply for logic and I/O pins.
NC	—	1,17,28, 40	12,13, 33,34		—	These pins are not internally connected. These pins should be left unconnected.

Legend: I = input O = output I/O = input/output P = power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as an external interrupt.
2: This buffer is a Schmitt Trigger input when used in serial programming mode.
3: This buffer is a Schmitt Trigger input when configured as general purpose I/O and a TTL input when used in the Parallel Slave Port mode (for interfacing to a microprocessor bus).
4: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

2.0 MEMORY ORGANIZATION

There are three memory blocks in each of these PICmicro MCUs. The Program Memory and Data Memory have separate buses so that concurrent access can occur and is detailed in this section. The EEPROM data memory block is detailed in Section 4.0.

Additional information on device memory may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023).

2.1 Program Memory Organization

The PIC16F87X devices have a 13-bit program counter capable of addressing an 8K x 14 program memory space. The PIC16F877/876 devices have 8K x 14 words of FLASH program memory and the PIC16F873/874 devices have 4K x 14. Accessing a location above the physically implemented address will cause a wrap-around.

The reset vector is at 0000h and the interrupt vector is at 0004h.

FIGURE 2-1: PIC16F877/876 PROGRAM MEMORY MAP AND STACK

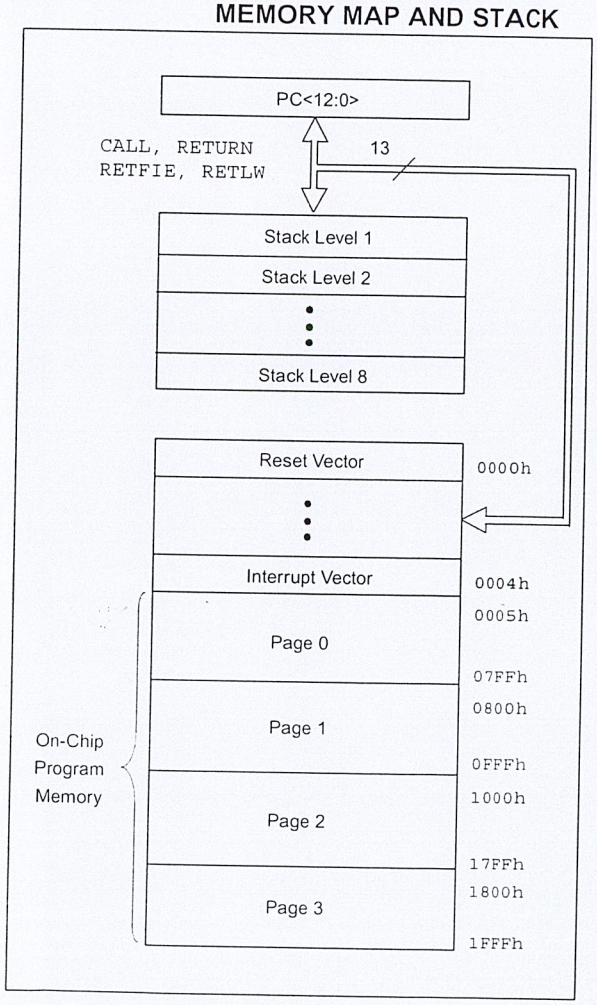
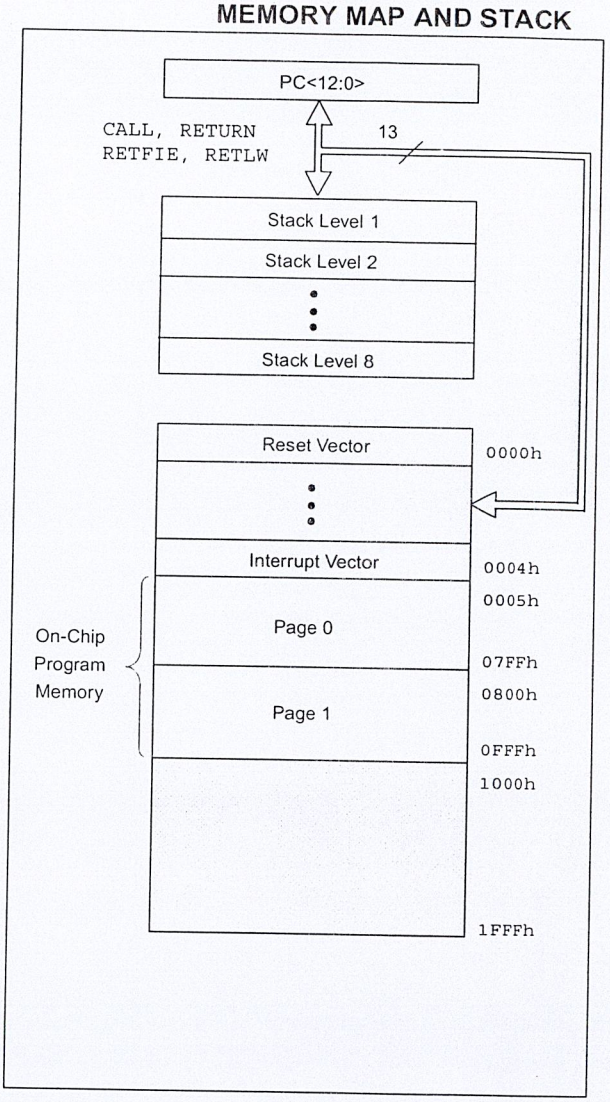


FIGURE 2-2: PIC16F874/873 PROGRAM MEMORY MAP AND STACK



2.2 Data Memory Organization

The data memory is partitioned into multiple banks which contain the General Purpose Registers and the Special Function Registers. Bits RP1(STATUS<6>) and RP0 (STATUS<5>) are the bank select bits.

RP1:RP0	Bank
00	0
01	1
10	2
11	3

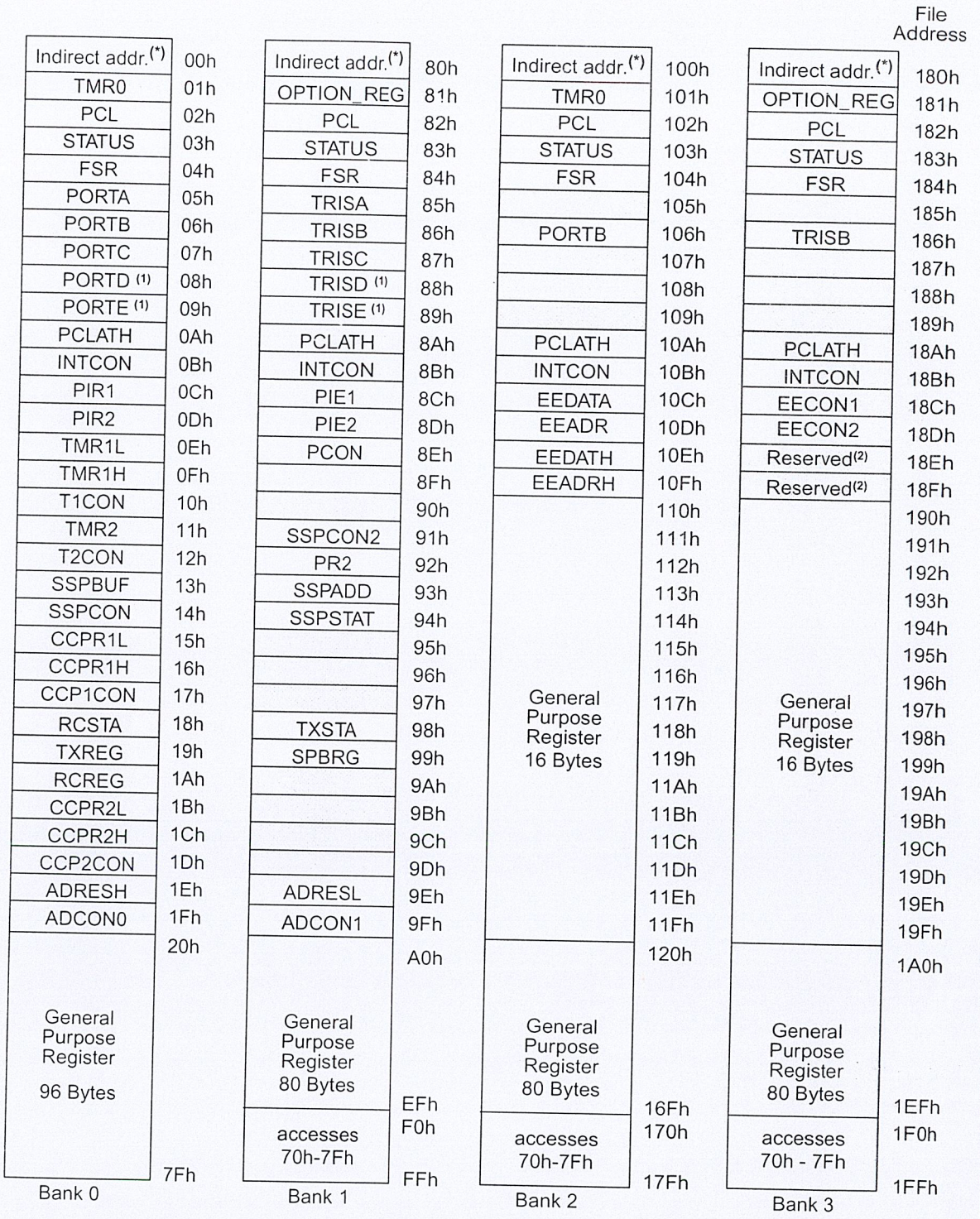
Each bank extends up to 7Fh (128 bytes). The lower locations of each bank are reserved for the Special Function Registers. Above the Special Function Registers are General Purpose Registers, implemented as static RAM. All implemented banks contain Special Function Registers. Some "high use" Special Function Registers from one bank may be mirrored in another bank for code reduction and quicker access.

Note: EEPROM Data Memory description can be found in Section 4.0 of this Data Sheet

2.2.1 GENERAL PURPOSE REGISTER FILE

The register file can be accessed either directly, or indirectly through the File Select Register FSR.

FIGURE 2-3: PIC16F877/876 REGISTER FILE MAP



— Unimplemented data memory locations, read as '0'.
* Not a physical register.

Note 1: These registers are not implemented on 28-pin devices.
2: These registers are reserved, maintain these registers clear.