ระบบวิทยุติดตามตัว

PAGER SYSTEM

โดย

นางสาวเกษราภรณ์	ศรีพงศ์ธรพิบูล	รหัส	34101035
นายวรพจ	ปานพุ่มชื่น	รหัส	34106304
นางสาววาทิณี	ลีลาศรีชัย	รหัส	34106316

อาจารย์ที่ปรึกษา

ผศ.ดร.รัตติกร วรากูลศิริพันธ์

วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิชาวิศวกรรมอิเล็คทรอนิคส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้า วิทยาเขตเจ้าคุณทหาร ลาดกระบัง

ปีการศึกษา 2537

ปริญญานิพนธ์ปีการศึกษา 2537

ภาควิชาอิเล็คทรอนิคส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบวิทยุติดตามตัว (Pager System)

ผู้จัดทำ

1.นางสาวเกษราภรณ์ ศรีพงศ์ธรพิบูล
2.นายวรพจ ปานพุ่มชื่น
3.นางสาววาทิณี สีลาศรีชัย

อาจารย์ที่ปรึกษา

(ผศ.ดร.รัตติกร วราภูลศิริพันธ์)

ระบบวิทยุติดตามตัว

เกษราภรณ์ ศรีพงศ์ธรพิบูล
วรพจ ปานพุ่มชื่น
วาทิณี ลีลาศรีชัย
ผศ. ดร. รัตติกร วราภูลศิริพันธ์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2537

บทคัดย่อ

ในปัจจุบันโทรคมนาคมได้มีความสำคัญมากสำหรับมนุษย์ เครื่องมือการสื่อสารถูกนำมาใช้อย่างกว้างขวางในโลกธุรกิจ เช่น โทรศัพท์ และวิทยุติดตามตัว

ในปริญญานิพนธ์ฉบับนี้ ได้นำเสนอเนื้อหาเกี่ยวกับการสร้างระบบวิทยุติดตามตัว ทำการส่งโดยใช้เครื่องคอมพิวเตอร์เป็นตัวกำเนิดสัญญาณข้อมูล ส่งไปยังวงจรภาคส่งแล้วทำการมอดดูเลทแบบผสมทางความถี่ (Amplitude Modulation) ชนิด Amplitude shift - keying (ASK) กับคลื่นความถี่วิทยุความถี่ประมาณ 250 MHz ออกอากาศไปยังเครื่องรับ เครื่องรับจะรับสัญญาณเข้ามาและทำการแยกสัญญาณข้อมูลออกมาก่อนที่จะแสดงผลออกทางจอ LCD

โดยระบบวิทยุติดตามตัวนี้ ประกอบด้วย

1) เครื่องคอมพิวเตอร์ที่ศูนย์ควบคุม

2) เครื่องส่ง ประกอบด้วยวงจร RS - 232 Interface และวงจรผสมคลื่นทางความสูง (AM)

3) เครื่องรับ ประกอบด้วยวงจรถอดรหัสสัญญาณ (Demodulator) และวงจรส่วนควบคุม

PAGER SYSTEM

KETSARAPORN SRIPONGTORNPIBOON
WORAPOCH PANPOOMCHUEN
WATINEE LEELASRICHAI
ASST. PROF. RUTIKORN WARAKULSIRIPUN
1994

Abstract

In the present , telecommunication is very important for person . Communication implement are widely used in the business 's world , for instance telephone and pager .

This thesis proposes a " Character - Pager " . Some messages , originated by a personal computer , will transmit via the transmitter circuit . At the transmitter , the data signal and the carrier signal will have been mixed in the amplitude modulation (AM) - amplitude shift - keying (ASK) . The modolated signal has frequency about 250 MHz transmitted on air . At the receiver , the modulated signal is received and demodulated before messages will display on the LCD board .

The character pager consists of

1. a personal computer at the controller center.

2. transmitter consists of a RS - 232 Interface and an amplitude modulator .

3. Receiver consists of a demodulator and a controller circuit .

สารบัญ

บทที่ 1

บทนำ

ระบบวิทยุติดตามตัวหรือเพจเจอร์ (Pager) จัดเป็นเครื่องมือสื่อสารชนิดเคลื่อนที่แบบหนึ่งที่เป็นที่นิยมในหมู่นักธุรกิจ , นายแพทย์ , เซลล์แมน , วิศวกร , หรือช่างที่ให้บริการนอกสถานที่ และผู้มีภาระหน้าที่ซึ่งไม่ประจำที่ แต่จำเป็นต้องมีการติดต่อได้ตลอดเวลา เช่นเดียวกับพวกวิทยุโทรศัพท์ หรือโทรศัพท์เคลื่อนที่ แต่จะมีลักษณะการสื่อสารในทิศทางเดียวจากผู้ส่งไปยังผู้รับ (simplex) ข้อเด่นของเพจเจอร์คือ มีขนาดเล็กและกระทัดรัดสามารถพกติดตัวได้ตลอดเวลา

การติดต่อสื่อสารระหว่างผู้ส่งกับผู้รับอาจเป็นการแจ้งให้ผู้รับหาทางติดต่อกลับไปยังผู้ส่งหรืออาจจะเป็นข่าวสารที่ต้องการติดต่อไปยังผู้รับโดยตรงก็ได้ เครื่องรับวิทยุติดตามตัวที่มีใช้กันอยู่ในปัจจุบันแบ่งได้เป็น 5 แบบ คือ

1. ระบบวิทยุติดตามตัวแบบส่งข่าวสารเป็นเสียงพูด (voice pager) ส่งข่าวสารเป็นเสียงพูดไปยังผู้รับ

2. ระบบวิทยุติดตามตัวแบบส่งข่าวสารเป็นตัวเลข (digital display pager) ซึ่งตัวเลขนี้อาจแทนข่าวสารที่ต้องการ เช่นเป็นเบอร์โทรศัพท์ หรือรหัสที่มีความหมายเป็นที่เข้าใจกันระหว่างผู้ส่งกลับผู้รับ

3. ระบบวิทยุติดตามตัวแบบส่งข่าวสารเป็นเสียงพูด (alpha numerlic pager) แทนที่จะเป็นตัวเลขเพียงอย่างเดียว ก็สามารถส่งเป็นตัวอักษรได้ ทำให้สามารถส่งข่าวสารได้ละเอียดขึ้น แต่จำเป็นต้องใช้ระบบที่พิเศษขึ้น

4. ระบบวิทยุติดตามตัวแบบส่งสัญญาณเตือน (tone - alert pager) เป็นการส่งสัญญาณเสียงเพื่อเป็นการบอกให้ผู้รับติดต่อกลับไปยังศูนย์ เพื่อรับข่าวสารอีกทีหนึ่ง

5. ระบบวิทยุติดตามตัวแบบแอดเดรสคู่ (dual address pager) เหมือนกลับระบบวิทยุติดตามตัวแบบส่งเสียงเตือน เพียงแต่สามารถส่งสัญญาณเสียงได้ 2 ลักษณะ เพื่อให้ผู้รับทราบว่า จะติดต่อกลับไปยังที่ใด

ในระบบหนึ่ง ๆ ของวิทยุติดตามตัว สามารถใช้ร่วมกันในระบบได้ทั้ง 5 แบบ

ดังที่กล่าวมาแล้ว แต่นิยมที่จะแยกความถี่ใช้งานสำหรับระบบวิทยุติดตามตัวแต่ละระบบเพื่อให้ได้คุณภาพของสัญญาณที่ดี

ข้อดีของระบบวิทยุติดตามตัวเมื่อเปรียบเทียบกับระบบสื่อสารเคลื่อนที่ชนิดสองทิศทาง (ตอบโต้กันได้) คือ

- สามารถพกติดตามตัวได้สะดวก เนื่องจากขนาดเล็ก น้ำหนักเบา และไม่ต้องการไฟฟ้าจากภายนอก
- ค่าใช้จ่ายในการเช่าถูกกว่ามาก
- มีจำนวนผู้เช่าใช้ได้มากกว่าต่อหนึ่งความถี่
- เครื่องรับสามารถใช้ร่วมกันได้โดยส่วนใหญ่จากหลาย ๆ บริษัท
- แทบจะไม่ต้องมีการบำรุงรักษาเลยที่ตัวเครื่องรับวิทยุติดตามตัว
- ไม่ต้องมีการติดตั้งใด ๆ ในส่วนของผู้เช่าใช้
- เครื่องส่วนใหญ่สามารถบันทึกข่าวสารไว้ได้ในตัว

สำหรับในโครงงานนี้ จำลองระบบวิทยุติดตามตัว ซึ่งเลือกใช้ระบบที่ส่งข่าวสารเป็นตัวอักษร เนื่องจากสามารถส่งข่าวสารได้ละเอียด รับข่าวสารได้อย่างถูกต้องทันทีไม่ต้องเสียเวลาในการติดต่อกลับไปยังผู้ส่งหรือศูนย์อีก

บทที่ 2

ทฤษฎีการส่งผ่านข้อมูลอนาลอกและดิจิตอล

(Analog and Digital Data Transmission)

2.1 การส่งผ่านข้อมูลอนาลอกและดิจิตอล

ในการส่งผ่านข้อมูลจากต้นทางไปยังปลายทาง มีคุณสมบัติธรรมชาติของสัญญาณข้อมูลอย่างหนึ่งที่น่าสนใจก็คือ การเคลื่อนที่ของข้อมูลและกระบวนการจัดการกับข้อมูลเพื่อให้มั่นใจได้ว่าข้อมูลที่ส่งไปและรับมาสามารถจะเข้าใจกันได้ สำหรับการศึกษาทั้งหมดนี้มีจุดสำคัญที่เราจะเข้าไปเกี่ยวข้องด้วยก็คือ ปริมาณชนิดอนาลอก และชนิดดิจิตอล

ปริมาณอนาลอก และดิจิตอล เป็นปริมาณที่สอดคล้องกับปริมาณที่มีค่าแบบต่อเนื่องและไม่ต่อเนื่องตามลำดับ ศัพท์ 2 คำนี้จะถูกนำมาใช้ในเรื่องสามเรื่องดังต่อไปนี้คือ

- ข้อมูล (Data) คืออนุภาคหรือปริมาณที่ทำหน้าที่สื่อนำความหมาย
- สัญญาณ (Signal) คือการนำเอากระแสไฟฟ้าหรือคลื่นแม่เหล็กมาเข้ารหัสแทนข้อมูล
- การส่งผ่าน (Transmission) คือ การสื่อสารข้อมูลผ่านตัวกลางโดยอาจจะอาศัยกระบวนการบางอย่าง

ข้อมูล (Data)

ข้อมูลแบบอนาลอกนั้น เป็นข้อมูลที่เกิดขึ้นแล้วมีค่าที่ต่อเนื่องในช่วงเวลาที่พิจารณา เป็นต้นว่า เสียงหรือภาพจะเป็นข้อมูลที่มีการแปรเปลี่ยนรูปแบบของความเข้มอย่างต่อเนื่อง อุณหภูมิ และความดันก็เป็นปริมาณที่ต่อเนื่องที่จัดได้ว่าให้ข้อมูลแบบอนาลอก แต่สำหรับข้อมูลดิจิตอลเป็นข้อมูลที่มีค่าที่เฉพาะที่กระโดด เป็นค่าที่ไม่ต่อเนื่องเช่น ค่าเลขจำนวนเต็ม เป็นต้น

ตัวอย่างของข้อมูลดิจิตอลอย่างหนึ่งได้แก่ ตัวหนังสือหรือตัวอักษร ซึ่งประสาทสัมผัสของคนสามารถรับรู้และเข้าใจได้ แต่มันไม่สามารถจะสื่อความหมายนั้นโดยตรงให้ระบบสื่อสารต่าง ๆ รับรู้เข้าใจ เหมือนกับที่มนุษย์เราเข้าใจได้

ระบบในการสื่อสารบางระบบจะถูกออกแบบไว้สำหรับข้อมูลเลขฐานสอง ซึ่งจำนวนของรหัสที่ใช้กันอย่างแพร่หลายที่สุดชนิดหนึ่งคือ รหัส ASCII (American Standard Code for Information Interchanges) ตัวอักขระแต่ละตัวของรหัสนี้แทนได้ด้วยเลขฐานสอง 7 หลัก ซึ่งเลขฐานสอง 7 หลัก แทนอักขระที่ไม่ซ้ำกันได้ทั้งหมดถึง 128 ตัว รหัสเลขฐานสองบางตัวจึงถูกนำมาแทนตัวอักขระสำหรับการควบคุม และตัวอักขระของการควบคุมส่วนหนึ่งก็ใช้ในการควบคุมการพิมพ์ และรหัสเลขฐานสองที่แทนตัวอักขระส่วนที่เหลือบางตัวก็จะใช้ในระบบสื่อสาร โดยทั่วไปแล้วการเข้ารหัสในการเก็บและส่งผ่านข้อมูลจะใช้จำนวนเลขไบนารี 8 บิท ต่อ 1 ตัวอักษร โดยบิทที่ 8 ก็คือพาริตี้บิท (Palarity Bit) ที่ใช้สำหรับการตรวจสอบความผิดพลาด

สัญญาณ (Signal)

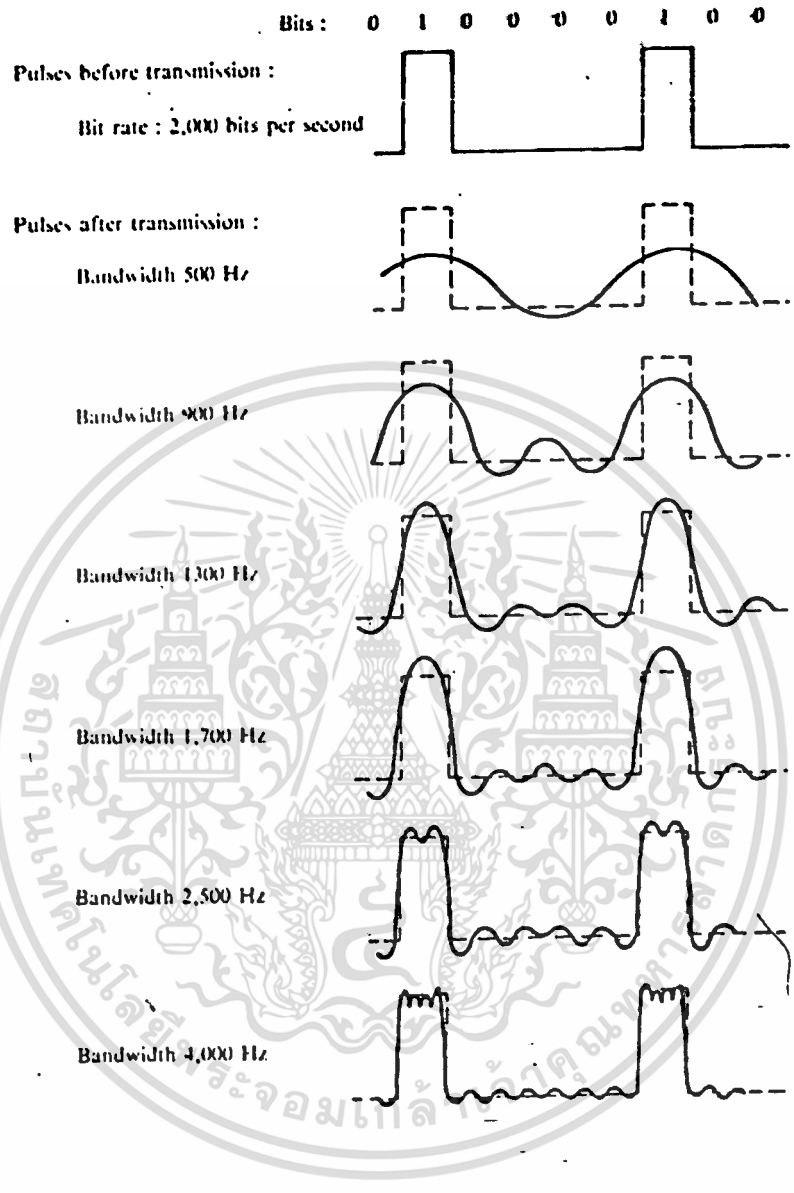
ในระบบของการสื่อสาร ข้อมูลจะเคลื่อนที่จากจุดหนึ่งไปยังอีกจุดหนึ่งในรูปแบบของสัญญาณไฟฟ้า ซึ่งสัญญาณอนาลอกก็คือ คลื่นแม่เหล็กไฟฟ้าที่มีการเปลี่ยนแปลงอย่างต่อเนื่อง และสัญญาณดังกล่าวนี้เองที่จะถูกส่งผ่านเข้าไปในตัวกลางชนิดต่าง ๆ ทั้งนี้ก็ขึ้นอยู่กับสเปคตรัมของสัญญาณ ตัวกลางอาจเป็นสาย เช่น สายคู่บิดเกลียว สายโคแอคเชียล สายใยนำแสง หรืออาจจะเป็นตัวกลางแบบไร้สาย เช่น ชั้นบรรยากาศหรือสูญญากาศ ส่วนสัญญาณดิจิตอลนั้นอาจเป็นขบวนของพัลส์โวลเตจที่ใช้ระดับของพัลส์ที่ส่งไปเป็นตัวแทนข้อมูล เช่น ใช้ระดับโวลเตจคงที่ค่าบวกจะแทนค่าไบนารี 1 และระดับโวลเตจลบแทนค่าไบนารี 0

ข้อมูลดิจิตอลไบนารี สัญญาณที่ใช้แทนข้อมูลจะใช้ศักย์ไฟฟ้ากระแสตรง 2 ระดับ โดยที่ระดับโวลเตจค่าหนึ่งแทนเลขฐานสองค่า 1 และอีกค่าหนึ่งแทนเลขฐานสองค่า 0

จากรูปที่ 2.1 จะเห็นได้ว่า แบนด์วิดธ์ที่กว้างนั้นก็จะทำให้เกิดสัญญาณที่มีความผิดเพี้ยนน้อย สำหรับข้อมูลที่มีอัตรา x บิทต่อวินาที สามารถที่จะส่งได้โดยใช้สายส่งที่มีแบนด์วิดธ์ x Hz ด้วยคุณภาพดีพอควร แต่จะดีมากถ้าแบนด์วิดธ์เพิ่มขึ้นเป็น 2x Hz

ข้อมูลและสัญญาณ (Data and Signal)

จากที่ได้กล่าวมาแล้ว เราก็สังสนใจสัญญาณที่ใช้แทนข้อมูลอนาลอก และข้อมูลดิจิตอลในสถานการณ์โดยทั่วไปแล้วข้อมูลอนาลอกจะเป็นฟังก์ชั่นกับเวลา และมีค่าสเปคตรัม



รูปที่ 2.1 แสดงอิทธิพลของแบนด์วิดธ์กับข้อมูลดิจิตอล

ที่จำกัดอยู่ค่าหนึ่ง ซึ่งเราสามารถแทนข้อมูลอนาลอกดังกล่าว ได้ด้วยสัญญาณคลื่นแม่เหล็กไฟฟ้าที่มีค่าสเปคตรัมค่าเดียวกัน และข้อมูลดิจิตอลนั้นสามารถแทนได้ด้วยสัญญาณดิจิตอลระดับโวลเตจแตกต่างกันสองระดับเพื่อแทนค่าเลขฐานสอง 0 และ 1

การส่งผ่าน (Transmission)

ความแตกต่างระหว่างสัญญาณดิจิตอลและสัญญาณอนาลอกนั้น ควรพิจารณาให้ชัดเจนขึ้นอีก ทั้งสัญญาณอนาลอกและสัญญาณดิจิตอลนั้นจะถูกนำส่งผ่านตัวกลางที่เหมาะสม และขบวนการในการดัดแปลงสัญญาณ เพื่อให้เหมาะสมกับตัวกลางการส่งผ่านก็เป็นหน้าที่ของระบบการส่งผ่าน (Transmission system) การส่งผ่านในรูปแบบของอนาลอกเป็นวิธีการส่งผ่านสัญญาณอนาลอกไปโดยไม่มีการเปลี่ยนแปลงสาระข้อมูล ไม่ว่าจะเป็นกรณีใด ๆ สัญญาณอนาลอกจะเกิดการลดทอนหลังจากที่เดินทางผ่านเข้าไปในตัวกลาง ดังนั้นเพื่อให้สัญญาณสามารถเดินทางไปถึงปลายทางได้ระยะทางไกล ๆ ระบบของการส่งผ่านแบบอนาลอกก็จะมีตัวเพิ่มความแรงของสัญญาณ เพื่อเพิ่มพลังงานให้แก่สัญญาณ แต่ยังมีปัญหาเพราะว่าตัวเพิ่มความแรงของสัญญาณ นอกจากจะเพิ่มพลังงานให้กับสัญญาณแล้วยังเพิ่มพลังงานให้กับสัญญาณรบกวนอีกด้วย และยิ่งมีการต่อใช้ตัวขยายหลาย ๆ ตัวแบบอนุกรมเพื่อให้ได้ระบบในการส่งสัญญาณได้ไกล ๆ ยิ่งทำให้สัญญาณผิดเพี้ยนไปยิ่งขึ้น

สำหรับข้อมูลแบบอนาลอกเช่นเสียง ความผิดเพี้ยนเพียงส่วนเล็กน้อยสามารถยอมให้เกิดขึ้นได้เพราะว่าข้อมูลยังสามารถเข้าใจได้ แต่สำหรับข้อมูลดิจิตอลการต่อตัวขยายแบบอนุกรมจะทำให้เกิดความผิดพลาด

การส่งผ่านข้อมูลดิจิตอล (Digital Transmission)

การส่งผ่านข้อมูลด้วยวิธีการนี้จะเกี่ยวข้องกับเนื้อหาของสัญญาณ สัญญาณดิจิตอลจะถูกส่งไปได้ในระยะทางที่จำกัด ก่อนที่การลดทอนจะทำอันตรายต่อองค์ประกอบของข้อมูล ดังนั้นเพื่อให้การส่งข้อมูลสามารถทำได้เป็นระยะทางที่ไกล ๆ เราจึงใช้ตัวทวนสัญญาณ (Repeater) เพื่อกู้สัญญาณดั้งเดิมกลับคืนมาโดยที่ตัวทวนสัญญาณเมื่อได้รับสัญญาณดิจิตอลมาแล้ว ก็จะทำการกู้รูปแบบของ 1 และ 0 กลับคืนมาอีกครั้งและส่งต่อออกไปใหม่ ซึ่งทำให้สามารถเอาชนะการลดทอนลงไปได้

ด้วยเทคนิคอย่างเดียวกันกับที่ได้กล่าวมาแล้ว อาจจะนำมาใช้ได้กับสัญญาณอนาลอกที่ใช้เป็นตัวส่งถ่ายข้อมูลดิจิตอล ณ ตำแหน่งในพื้นที่ที่เหมาะสม ระบบของการส่งผ่านก็จะใช้เครื่องทวนสัญญาณแทนที่จะเป็นตัวขยายสัญญาณ ตัวทวนสัญญาณจะกู้สัญญาณดิจิตอลกลับคืนมาจากสัญญาณอนาลอก และสร้างสัญญาณอนาลอกขึ้นมาใหม่ ทำให้ไม่เกิดการสะสมของ

สัญญาณรบกวนต่อ ๆ ไป

ในปัจจุบันมีแนวโน้มที่จะหันมาใช้การสื่อสารระบบดิจิตอลที่ได้แทนระบบอนาลอก แม้ว่าจะไม่ได้มีการลงทุนใช้ระบบอนาลอกมาก่อนอย่างมากก็ตาม เหตุผลที่สำคัญก็คือ

- ดิจิตอลเทคโนโลยี การพัฒนาเทคโนโลยีของวงจรดิจิตอล LSI และ VLSI ทำให้ราคาและขนาดของวงจรลดลงในขณะที่เครื่องมือทางอนาลอกไม่ได้ลดลง

- คุณภาพของข้อมูล สำหรับขบวนการทางดิจิตอลการใช้ตัวทวนสัญญาณแทนที่จะใช้ตัวขยายสัญญาณทำให้อิทธิพลของสัญญาณรบกวนไม่ถูกสะสม ทำให้เราสามารถส่งข้อมูลไปได้ระยะทางไกล ๆ แม้ว่าคุณภาพของสายจะไม่ดีก็ตาม

- ความจุของการใช้งานมีมาก มันเป็นเรื่องที่สิ้นเปลืองมากในการที่เราจะต้องสร้างทางเดินการส่งผ่านข้อมูลที่มีแบนด์วิดธ์กว้างมาก ๆ เช่น ช่องสัญญาณดาวเทียมและเส้นใยนำแสง ดังนั้นการนำเอาขบวนการในการมัลติเพลกซ์เข้ามาใช้งานดูจะเป็นประโยชน์อย่างมากในเรื่องของความจุ ซึ่งขบวนการมัลติเพลกซ์ทางด้านเวลาดูจะเป็นวิธีการที่ง่ายและราคาถูกกว่าการมัลติเพลกซ์ทางด้านความถี่

- ความปลอดภัยและความเป็นส่วนตัว เทคนิคการย่อข้อมูลพร้อมที่จะนำเข้ามาใช้กับข้อมูลดิจิตอลและพร้อมที่จะนำมาใช้กับข้อมูลอนาลอกที่ถูกดิจิไตซ์แล้ว

- การรวมกันเข้าเป็นหนึ่งเดียวกัน ด้วยขบวนการทางข้อมูลของอนาลอกและดิจิตอลสัญญาณจะมีรูปแบบที่เหมือนกัน และสามารถดำเนินการได้ในลักษณะเดียวกัน ซึ่งมันก็จะทำให้ประหยัดและสะดวกในการรวมเสียง ภาพและข้อมูลดิจิตอลเข้าไว้ด้วยกัน

2.2 อุปสรรค และปัญหาในการส่งสัญญาณ

เมื่อเราส่งสัญญาณผ่านตัวกลาง จะเกิดปัญหาหลัก ๆ โดยทั่วไปจากสาเหตุดังต่อไปนี้คือ

- การลดทอนและการผิดเพี้ยนจากการลดทอน
- การเสียรูปของสัญญาณเนื่องจากความล่าช้า (Delay)
- สัญญาณรบกวน (Noise)

การลดทอน (Attenuation)

เมื่อสัญญาณเดินทางผ่านตัวกลาง ความแรงของสัญญาณจะลดลงตามระยะที่สัญญาณนั้นเดินทางผ่าน ความแรงของสัญญาณที่ลดลงหรือการลดทอนนี้โดยทั่วไปจะเป็นฟังก์ชั่นลอการิธึมซึ่งสามารถแสดงเป็นจำนวนคงที่ได้ในหน่วยเดซิเบลต่อระยะทางหนึ่งหน่วยสำหรับตัวกลางชนิดสายและสำหรับตัวกลางที่ไร้สาย การลดทอนสัญญาณจะเป็นฟังก์ชั่นที่ซับซ้อนกับระยะทาง ข้อระวังเกี่ยวกับเรื่องการลดทอนมี 3 ประการในวิศวกรรมการส่งผ่านข้อมูล ประการที่ 1 สัญญาณที่รับได้จะต้องมีความแรงเพียงพอ เพื่อที่วงจรอิเล็กทรอนิกส์ของตัวรับสามารถที่จะตรวจพบและแปลความหมายของสัญญาณนั้นได้ ประการที่ 2 ความแรงของสัญญาณจะต้องรักษาระดับให้สูงกว่าระดับสัญญาณรบกวนเพียงพอ เพื่อให้ตัวรับสามารถรับสัญญาณได้โดยไม่เกิดความผิดพลาด ประการที่ 3 การลดทอนของสัญญาณมักเป็นฟังก์ชั่นของความถี่

ประการที่ 1 และที่ 2 จะเกี่ยวข้องกับความแรงของสัญญาณซึ่งก็ขึ้นอยู่กับตัวขยายหรือตัวทวนสัญญาณสำหรับการเชื่อมต่อทางเดินข้อมูลแบบจุดต่อจุด ความแรงของสัญญาณของตัวส่งผ่านจะต้องแรงเพียงพอที่เครื่องรับสามารถจะเข้าใจได้ และจะต้องไม่แรงจนทำให้เกิดการเป็นภาระแก่ตัวส่ง ซึ่งอาจทำให้เกิดการเพี้ยนของสัญญาณที่สร้างขึ้นมาได้ ในระยะทางไกล ๆ ถ้าการลดทอนเกิดขึ้นมากจนยอมรับไม่ได้ ก็จะมีการใช้ตัวทวนสัญญาณเพื่อเพิ่มความแรงของสัญญาณขึ้น แต่สำหรับในระบบหลายจุดปัญหาดังกล่าวจะเป็นเรื่องที่สลับซับซ้อน เนื่องจากระยะทางจากตัวส่งถึงตัวรับจะไม่คงที่

สำหรับปัญหาประการที่ 3 เป็นจุดเด่นเฉพาะสำหรับสัญญาณอนาลอก เพราะว่าการลดทอนจะเปลี่ยนแปลงเป็นฟังก์ชั่นกับความถี่ สัญญาณที่รับได้จะเกิดการผิดเพี้ยนและความสามารถในการเข้าใจเนื้อหาข้อมูลก็จะลดลง เพื่อแก้ปัญหานี้เราจะใช้เทคนิคการยกระดับของสัญญาณลดทอนในย่านความถี่นั้น ๆ ซึ่งเทคนิคอันนี้ส่วนใหญ่ใช้กับสายโทรศัพท์โดยใช้ขดลวดภาระ (Loading Coil) แก้ไขคุณสมบัติทางไฟฟ้าของสาย ผลก็คือทำให้อัตราการลดทอนค่อนข้างสม่ำเสมอ อีกวิธีหนึ่งก็คือใช้ตัวขยายสัญญาณที่มีคุณสมบัติขยายสัญญาณที่สูงมากกว่าความถี่ต่ำ

การลดทอน (Attenuation)

เมื่อสัญญาณเดินทางผ่านตัวกลาง ความแรงของสัญญาณจะลดลงตามระยะที่สัญญาณนั้นเดินทางผ่าน ความแรงของสัญญาณที่ลดลงหรือการลดทอนนี้โดยทั่วไปจะเป็นฟังก์ชั่นลอการิธึมซึ่งสามารถแสดงเป็นจำนวนคงที่ได้ในหน่วยเดซิเบลต่อระยะทางหนึ่งหน่วยสำหรับตัวกลางชนิดสายและสำหรับตัวกลางที่ไร้สาย การลดทอนสัญญาณจะเป็นฟังก์ชั่นที่ซับซ้อนกับระยะทาง ข้อระวังเกี่ยวกับเรื่องการลดทอนมี 3 ประการในวิศวกรรมการส่งผ่านข้อมูล ประการที่ 1 สัญญาณที่รับได้จะต้องมีความแรงเพียงพอ เพื่อที่วงจรอิเล็กทรอนิกส์ของตัวรับสามารถที่จะตรวจพบและแปลความหมายของสัญญาณนั้นได้ ประการที่ 2 ความแรงของสัญญาณจะต้องรักษาระดับให้สูงกว่าระดับสัญญาณรบกวนเพียงพอ เพื่อให้ตัวรับสามารถรับสัญญาณได้โดยไม่เกิดความผิดพลาด ประการที่ 3 การลดทอนของสัญญาณมักเป็นฟังก์ชั่นของความถี่

ประการที่ 1 และที่ 2 จะเกี่ยวข้องกับความแรงของสัญญาณซึ่งก็ขึ้นอยู่กับตัวขยายหรือตัวทวนสัญญาณสำหรับการเชื่อมต่อทางเดินข้อมูลแบบจุดต่อจุด ความแรงของสัญญาณของตัวส่งผ่านจะต้องแรงเพียงพอที่เครื่องรับสามารถจะเข้าใจได้ และจะต้องไม่แรงจนทำให้เกิดการเป็นภาระแก่ตัวส่ง ซึ่งอาจทำให้เกิดการเพี้ยนของสัญญาณที่สร้างขึ้นมาได้ ในระยะทางไกล ๆ ถ้าการลดทอนเกิดขึ้นมากจนยอมรับไม่ได้ ก็จะมีการใช้ตัวทวนสัญญาณเพื่อเพิ่มความแรงของสัญญาณขึ้น แต่สำหรับในระบบหลายจุดปัญหาดังกล่าวจะเป็นเรื่องที่สลับซับซ้อน เนื่องจากระยะทางจากตัวส่งถึงตัวรับจะไม่คงที่

สำหรับปัญหาประการที่ 3 เป็นจุดเด่นเฉพาะสำหรับสัญญาณอนาลอก เพราะว่าการลดทอนจะไปเปลี่ยนแปลงเป็นฟังก์ชั่นกับความถี่ สัญญาณที่รับได้จะเกิดการผิดเพี้ยนและความสามารถในการเข้าใจเนื้อหาข้อมูลก็จะลดลง เพื่อแก้ปัญหานี้เราจะใช้เทคนิคการยกระดับของสัญญาณลดทอนในย่านความถี่นั้น ๆ ซึ่งเทคนิคอันนี้ที่นำไปใช้กับสายโทรศัพท์โดยใช้ขดลวดภาระ (Loading Coil) แก้ไขคุณสมบัติทางไฟฟ้าของสาย ผลก็คือทำให้อัตราการลดทอนค่อนข้างสม่ำเสมอ อีกวิธีหนึ่งก็คือใช้ตัวขยายสัญญาณที่มีคุณสมบัติขยายสัญญาณที่สูงมากกว่าความถี่ต่ำ

ความเพี้ยนของสัญญาณเนื่องจากความล่าช้า

ความเพี้ยนของสัญญาณเนื่องจากความล่าช้า เป็นปรากฏการณ์ที่มักจะเกิดขึ้นกับตัวกลางการส่งผ่านแบบมีสาย เพราะว่าความเร็วในการเคลื่อนที่ของสัญญาณผ่านตัวกลางแบบมีสายจะเปลี่ยนแปลงตามความถี่ สำหรับสัญญาณที่มีย่านความถี่จำกัด ความเร็วของสัญญาณจะมีค่าสูงสุดใกล้กับความถี่ศูนย์กลาง และจะลดลงไปทั้งสองด้านของย่านความถี่ ดังนั้นองค์ประกอบของสัญญาณที่มีความถี่ต่างกันจะเดินทางไปถึงตัวรับไม่พร้อมกัน

ความเพี้ยนเนื่องจากการล่าช้านี้เป็นปัญหาที่ร้ายแรงสำหรับข้อมูลดิจิตอลซึ่งถ้าพิจารณาจากลำดับของบิทที่กำลังส่งผ่านโดยใช้สัญญาณอนาลอกหรือสัญญาณดิจิตอลชนิดใดชนิดหนึ่ง ความล่าช้าเนื่องจากองค์ประกอบของสัญญาณบางความถี่ไม่เท่ากัน เป็นเหตุให้เกิดอินเตอร์ซิมโบล์อินเตอร์เฟอเรนซ์ (Intersymbol interferrence) ซึ่งเป็นสิ่งสำคัญในการจำกัดอัตราการส่งข้อมูลของตัวควบคุมการส่งผ่าน

ดีเลย์ อีควอไลเซอร์สามารถนำมาแก้ความเพี้ยนได้ของสัญญาณเนื่องจากความล่าช้าได้

สัญญาณรบกวน (Noise)

สำหรับกระบวนการส่งข้อมูลใด ๆ สัญญาณที่ได้รับจะประกอบไปด้วยสัญญาณที่ที่ส่งมาจากเครื่องส่งซึ่งจะมีการเปลี่ยนแปลงไปด้วยความเพี้ยนของสัญญาณที่เกิดจากการส่งผ่าน ความเพี้ยนที่เพิ่มเข้ามาเพราะสัญญาณที่เราไม่ต้องการ ที่เกิดขึ้นในระหว่างการส่งและการรับ

สัญญาณรบกวนอาจแบ่งได้เป็น 4 ชนิด ดังนี้คือ

1) สัญญาณรบกวนเนื่องจากความร้อน (Thermal noise)

เกิดจากการสั่นของอิเลคตรอนในตัวนำเนื่องจากความร้อน ซึ่งมักเกิดขึ้นกับอุปกรณ์อิเลคทรอนิกส์และตัวกลาง เป็นฟังก์ชั่นของอุณหภูมิ สัญญาณรบกวนเนื่องจากความร้อนจะมีการกระจายสเปคตรัมที่สม่ำเสมอเท่ากันในย่านความถี่ ซึ่งไม่สามารถกำจัดได้ จึงเป็นตัวจำกัดขอบเขตในการทำงานของระบบการสื่อสาร

$$N = KTB$$

โดยที่

N = ความหนาแน่นพลังงานของสัญญารบกวน, วัตต์/เฮิร์ต

K = ค่าคงที่ของโบล์ทมาน (Boltzmann's constant) = $1.38*10^{-23}$ J/K

B = แบนด์วิดธ์ของสัญญาณ, เฮิร์ต

2) สัญญาณรบกวนแบบอินเตอร์มอดดูเลชั่น (Intermodulation noice)

จะเกิดขึ้นเมื่อมีความไม่เป็นเชิงเส้นในการประมวลผลสัญญาณในตัวส่ง, ตัวรับ, หรือในระบบการส่งผ่าน ทำให้เกิดความถี่ที่เป็นผลรวมและผลต่างของสัญญาณเดิมสองความถี่ หรือผลคูณของสัญญาณความถี่ทั้งสอง ซึ่งสัญญาณที่ได้มานี้จะแทรกสอดกับสัญญาณที่เราต้องการ ความไม่เป็นเชิงเส้นของระบบมักเกิดขึ้นจากการทำงานผิดพลาดของอุปกรณ์หรือใช้สัญญาณแรงเกินไป

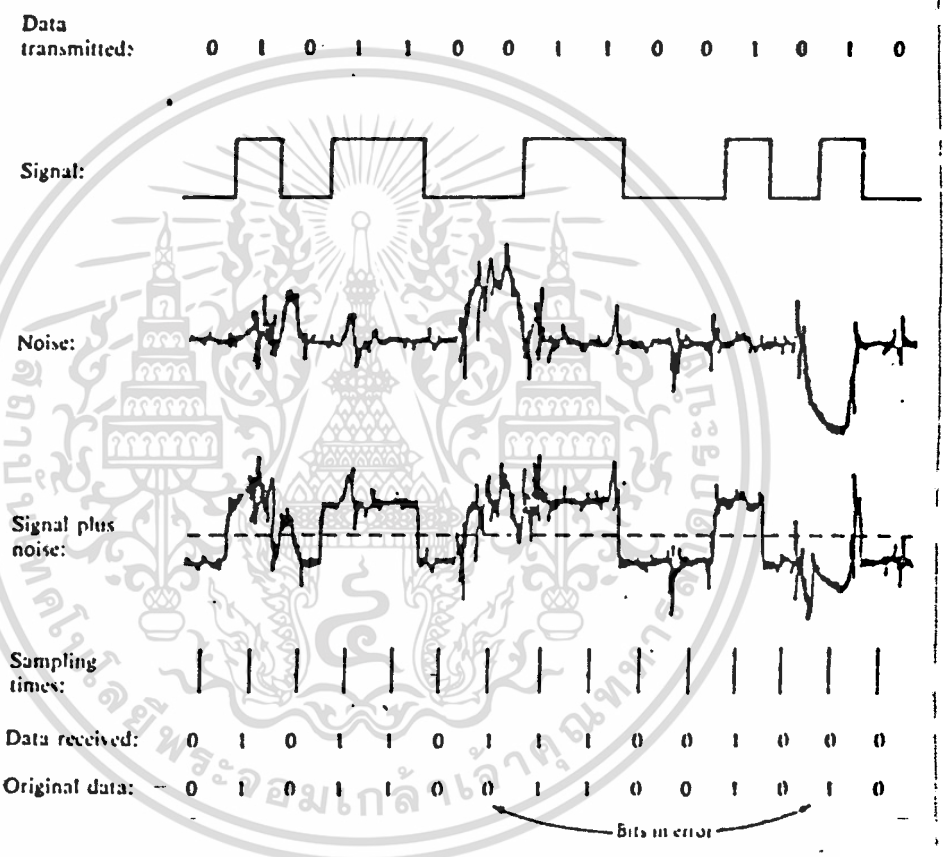
3) สัญญาณรบกวนแบบครอสทอล์ค (cross talk)

สัญญาณรบกวนแบบนี้เกิดขึ้นโดยการเหนี่ยวนำทางไฟฟ้าของสัญญาณระหว่างช่องสัญญาณหรือสายที่อยู่ใกล้ ๆ กัน เช่นในขณะที่มีการใช้โทรศัพท์เราอาจได้ยินเสียงคนอื่นที่ไม่ใช่คู่สนทนาของเราแทรกสอดเข้ามา นอกจากนี้ครอสทอล์คยังสามารถเกิดขึ้นได้โดยสาเหตุอื่น เช่น สัญญาณที่เราไม่ต้องการอาจถูกรับมาโดยสายอากาศไมโครเวฟ แม้ว่าจานสายอากาศจะอยู่ในตำแหน่งที่รับสัญญาณได้แรงที่สุดแล้วก็ตาม แต่พลังงานของคลื่นไมโครเวฟก็ยังคงกระจายไปในระหว่างการเดินทาง สัญญาณรบกวนแบบครอสทอล์คในระบบต่าง ๆ นั้นควรมีค่าน้อยกว่าสัญญาณรบกวนจากอุณหภูมิ

4) สัญญาณรบกวนแบบอิมพัลส์ (Impulse noise)

โดยทั่วไปแล้วจะรบกวนข้อมูลอนาลอก เกิดขึ้นอย่างไม่ต่อเนื่องด้วยพัลส์ผิดปกติหรือสัญญาณรบกวนแบบกระชากในช่วงเวลาสั้น ๆ และมีขนาดความแรงของสัญญาณสูง มักเกิดขึ้นจากหลายสาเหตุ รวมทั้งการรบกวนของสัญญาณภายนอก เช่น ฟ้าแลบ ซึ่งมักทำลายข้อมูลเสียง

อย่างไรก็ตามสัญญาณรบกวนแบบอิมพัลส์ก็เป็นแหล่งกำเนิดเบื้องต้นของความผิดพลาด สำหรับการสื่อสารข้อมูลดิจิตอล ตัวอย่างเช่น สัญญาณกระชากที่เปลี่ยนแปลงพลังงานอย่างรวดเร็วภายในเวลา 0.01 วินาที อาจจะไม่สามารถทำลายข้อมูลเสียง แต่มันจะทำลายบิทของข้อมูลดิจิตอลที่ส่งผ่านด้วยความเร็ว 4,800 bps ไปถึง 50 บิท ดังแสดงในรูปที่



รูปที่ 2.2 แสดงผลของสัญญาณรบกวน

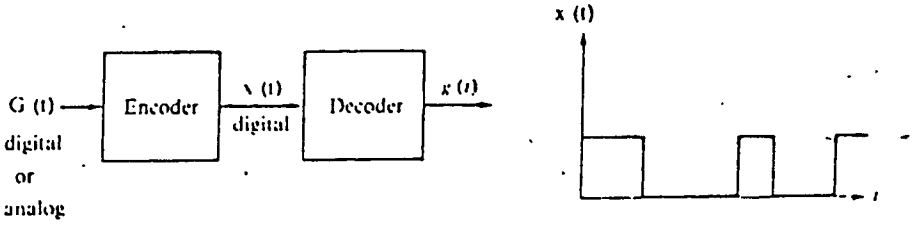
2.3 การเข้ารหัสข้อมูล (Data Encoding)

ในการเข้ารหัสและการมอดดูเลชั่น ซึ่งเป็นการแปลงข้อมูลต่าง ๆ ให้เป็นสัญญาณในรูปแบบต่าง ๆ ที่ต้องการให้เหมาะสมกับตัวกลางส่งผ่านและวิธีการส่ง ขบวนการดังกล่าวอาจมีข้อมูลต้นกำเนิด เป็นอนาลอกหรือดิจิตอลก็ได้ และสำหรับเทคนิคในการเข้ารหัสแบบดิจิตอลจะมีรูปแบบการเข้ารหัสที่หลากหลายกว่าอนาลอก ซึ่งก็ขึ้นอยู่กับการเลือกเพื่อให้เหมาะสมและได้ประสิทธิภาพสูงสุดในการส่งผ่านข้อมูลเข้าไปในตัวกลางนั้น ๆ ตัวอย่างเช่น เพื่อให้สอดคล้องกับแบนด์วิดธ์ หรือเพื่อทำให้เกิดความผิดพลาดน้อยที่สุด

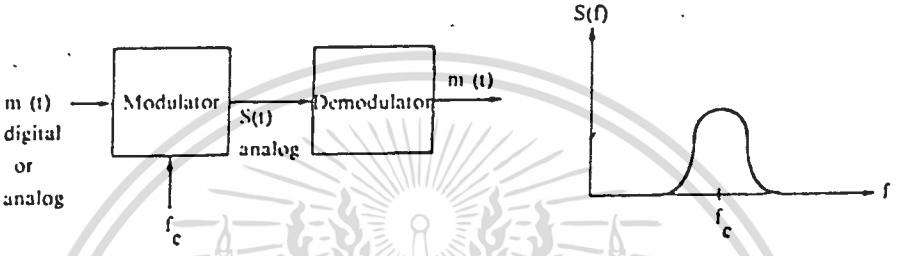
กรณีของสัญญาณอนาลอก ซึ่งมีคุณสมบัติพื้นฐานคือมีค่าความถี่คงที่ต่อเนื่องเหมาะที่จะใช้เป็นสัญญาณนำพาข้อมูลและความถี่ของสัญญาณตัวนำพา จะถูกเลือกให้เหมาะสมกับตัวกลางที่ใช้ในการส่งผ่าน โดยการนำพาข้อมูลนั้นอาศัยวิธีการมอดดูเลชั่นซึ่งเป็นกระบวนการของการเข้ารหัสข้อมูลไปบนสัญญาณตัวนำมา ซึ่งมีความถี่คงที่ความถี่หนึ่ง f_c เป็นหลัก การมอดดูเลชั่นทั้งหมดจะอยู่ภายในขอบเขตของการไปกระทำการเปลี่ยนแปลงพารามิเตอร์ของสัญญาณตัวนำพา ซึ่งได้แก่

- ขนาด (Amplitude)
- ความถี่ (Frequency)
- เฟส (Phase)

ในรูปที่ 2.3: สัญญาณอินพุท m(t) จะมีรูปแบบเป็นดิจิตอลหรืออนาลอกก็ได้ ซึ่งเรียกว่า สัญญาณเบสแบนด์ (Baseband) หรือสัญญาณมอดดูเลทติ้ง (Modulating Signal) ผลของการมอดดูเลทข้อมูลกับสัญญาณตัวนำพาจะได้สัญญาณ s(t) ซึ่งมีแบนด์วิดธ์ที่สัมพันธ์กับ f_c โดยมี f_c เป็นตำแหน่งศูนย์กลางของแบนด์วิดธ์



(a) Encoding onto a digital signal



(b) Modulation onto an analog signal

รูปที่ 2.3 เทคนิคการเข้ารหัสและการมอดดูเลชั่น

การจัดส่งสัญญาณนั้นมี 4 วิธีตามลักษณะของข้อมูลและสัญญาณ ซึ่งข้อพิจารณาในการเลือกใช้วิธีการจัดส่งข้อมูลทั้ง 4 วิธีนั้นมีดังต่อไปนี้

- Digital data / Digital signal : โดยทั่วไปแล้วอุปกรณ์ที่ใช้ในการเข้ารหัสข้อมูลดิจิตอลจะไม่ยุ่งยากซับซ้อน และมีราคาถูกกว่าอุปกรณ์ที่ใช้มอดดูเลทข้อมูลจากดิจิตอลไปเป็นอนาลอก

- Analog data / Digital signal : การแปลงข้อมูลอนาลอกให้เป็นสัญญาณดิจิตอลได้มีการนำมาใช้กันในการส่งผ่านดิจิตอลแบบใหม่กับอุปกรณ์สวิตช์ชิ่ง

- Digital data / Analog signal : เหมาะสำหรับตัวกลางการส่งผ่านบางตัว เช่นไฟเบอร์ออพติกและตัวกลางแบบไร้สาย ซึ่งจะยอมให้สัญญาณอนาลอกเคลื่อนที่ไปได้เพียงอย่างเดียว

- Analog data / Analog signal : ข้อมูลอนาลอกสามารถส่งผ่านในลักษณะของสัญญาณเบสแบนด์ได้ง่ายและราคาถูก เช่นการส่งผ่านเสียงไปตามสายโทรศัพท์

ประโยชน์อันหนึ่งที่สำคัญคือ การเลื่อนแบนด์วิดธ์ของสัญญาณเบสแบนด์ไปยังอีกส่วนหนึ่งของสเปคตรัม ดังนั้นสัญญาณหลาย ๆ ตัว โดยแต่ละตัวอยู่ในตำแหน่งของสเปคตรัมที่ต่างกัน สามารถใช้ตัวกลางในการส่งผ่านร่วมกันได้ วิธีการนี้เรียกว่าการมัลติเพลกซ์ทางความถี่ (Frequency Multiplextion)

สำหรับในรายงานนี้จะกล่าวถึงการมอดดูเลทข้อมูลดิจิตอลไปเป็นสัญญาณอนาลอกเพียงอย่างเดียว เพราะเหมาะสมสำหรับงานในการส่งข้อมูลข้อความ ซึ่งเป็นข้อมูลดิจิตอลโดยการมอดดูเลทกับคลื่นวิทยุ ส่งออกอากาศไปยังเครื่องรับ

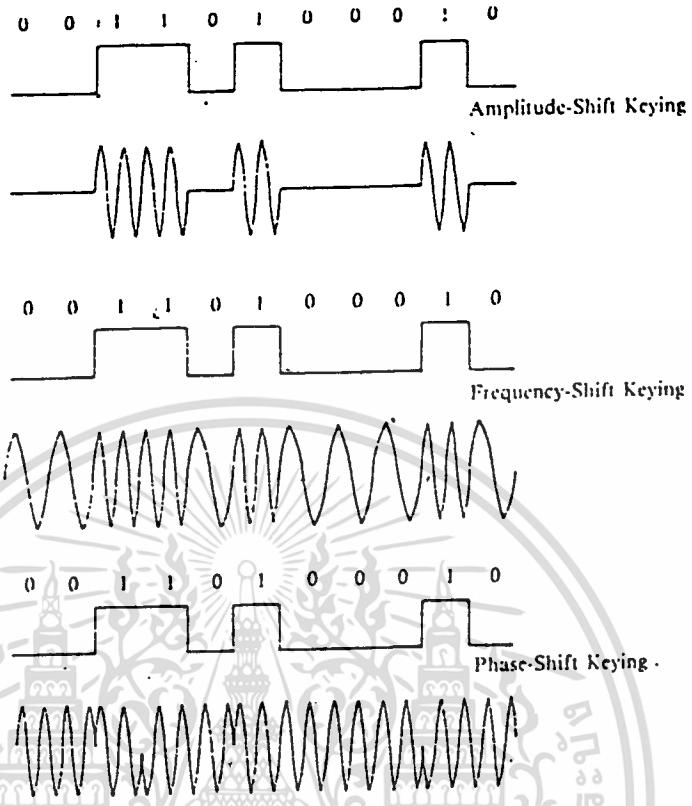
Digital data / Analog signal

สำหรับหัวข้อนี้จะเป็นการพิจารณาการส่งผ่านข้อมูลดิจิตอล โดยใช้สัญญาณอนาลอก กรณีที่คุ้นเคยมากที่สุดได้แก่ การส่งผ่านข้อมูลดิจิตอลเข้าไปในโครงข่ายของโทรศัพท์ ซึ่งต้องอาศัยอุปกรณ์โมเด็ม (Modem) ต่อเข้าไป เป็นตัวทำหน้าที่เปลี่ยนข้อมูลดิจิตอลไปเป็นสัญญาณอนาลอก และในทางกลับกันก็เปลี่ยนสัญญาณอนาลอกกลับมาเป็นข้อมูลดิจิตอลในทางด้านรับ

วิธีการเข้ารหัส

จากที่กล่าวมาแล้วว่าการมอดดูเลชั่น เป็นขบวนการที่เกี่ยวข้องกับการเปลี่ยนพารามิเตอร์อันใดอันหนึ่งของสัญญาณคลื่นตัวนำพา (ความถี่ของคลื่นพาหะ - Carrier Frequency) ซึ่งได้แก่ ขนาด ความถี่ และเฟส สำหรับการเปลี่ยนข้อมูลดิจิตอลให้เป็นสัญญาณอนาลอก แสดงดังรูปที่ 2.4 มี 3 แบบ คือ

* Amplitude - shift keying (ASK)
* Frequency - shift keying (FSK)
* Phase - shift keying (PSK)



รูปที่ 2.4 การมอดดูเลชั่นข้อมูลดิจิตอลให้เป็นสัญญาณอนาลอก

ทุก ๆ กรณีที่กล่าวมานี้สัญญาณที่ได้จะอยู่ในแบนด์วิดธ์ที่มีความถี่คลื่นพาหะเป็นความถี่ศูนย์กลางในเทคนิค ASK ค่าไบนารีสองค่าจะแทนด้วยค่าแอมปลิจูดที่แตกต่างกันสองค่าของความถี่คลื่นพาหะ เช่น ไบนารี 1 แทนด้วยแอมปลิจูดคงที่ของคลื่นพาหะ ส่วนไบนารี 0 แทนด้วยคลื่นพาหะที่ขาดหายไป ผลของสัญญาณเป็นดังนี้ คือ

$$s(t) = \begin{cases} A \cos (2\pi f_c t + \theta_c) & : \text{ไบนารี 1} \\ 0 & : \text{ไบนารี 0} \end{cases}$$

โดยที่สัญญาณคลื่นพาหะก็คือ $A \cos (2\pi f_c t + \theta_c)$ เทคนิคของ ASK ถูกนำไปใช้ในการส่งข้อมูลดิจิตอลลงไปบนเส้นใยแก้วนำแสง กล่าวคือสัญญาณตัวหนึ่งแทนด้วยพัลส์แสงในขณะที่อีกสัญญาณหนึ่งแทนด้วยการไม่มีแสง

สำหรับ FSK ค่าของไบนารีจะถูกแทนด้วยความถี่ 2 ความถี่ที่แตกต่างกันอยู่ใกล้ ๆ กับความถี่คลื่นพาหะ ผลลัพธ์ของสัญญาณจะเป็นดังนี้

$$s(t) = \begin{cases} A \cos (2\pi f_1 + \theta_c) & : \text{ไบนารี } 1 \\ A \cos (2\pi f_2 + \theta_c) & : \text{ไบนารี } 0 \end{cases}$$

โดยที่ f_1 และ f_2 เป็นความถี่ที่แตกมาจากความถี่คลื่นพาหะ f_c โดยมีจำ
เท่ากันแต่ตรงกันข้ามกัน

สำหรับ PSK เฟสของสัญญาณคลื่นพาหะจะถูกเปลี่ยนแปลงเพื่อแทนข้อมูลไบนารี 0 จะถูกแทนด้วยการส่งสัญญาณเฟสเดียวกับสัญญาณที่ส่งก่อนหน้านี้ ส่วนไบนารี 1 จะแทนด้วยการส่งสัญญาณเฟสตรงข้ามกับสัญญาณที่เกิดขึ้นก่อนหน้านี้ ซึ่งการมอดดูเลทแบบนี้ เรียกว่า DPSK (Double Phase shift - Keying) เป็นตัวอย่างของระบบ 2 เฟส ผลของสัญญาณเป็นดังนี้

$$s(t) = \begin{cases} A \cos (2\pi f_c t + \pi) & : \text{ไบนารี } 1 \\ A \cos (2\pi f_c t) & : \text{ไบนารี } 0 \end{cases}$$

โดยเฟสที่จัดจะเทียบกับช่องของบิทก่อนหน้านี้

ประสิทธิภาพของแบนด์วิดธ์จะมีค่ามาก ถ้าอนุภาคของแต่ละสัญญาณ สามารถใช้แทนจำนวนบิทหลายบิท ตัวอย่างเช่น ทำการเลื่อนเฟสไปเพียง 90° ดังนั้นใน 1 วัฏภาค จะแทนข้อมูลได้ 4 สถานะดังนี้

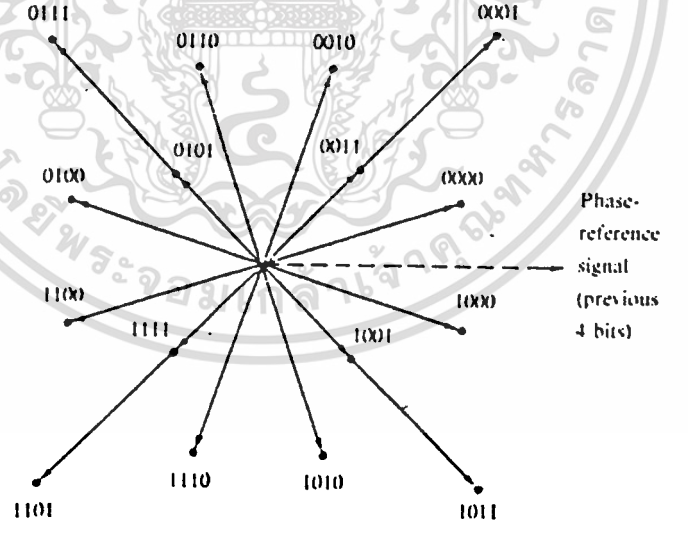
$A \cos (2\pi f_c t + 45^\circ)$ แทน 11

$A \cos (2\pi f_c t + 135^\circ)$ แทน 10

$A \cos (2\pi f_c t + 225^\circ)$ แทน 00

$A \cos (2\pi f_c t + 315^\circ)$ แทน 01

โครงสร้างอันนี้สามารถขยายออกไปได้อีก เช่นการส่งบิทเป็น 3 เท่า โดยใช้ความแตกต่างของเฟสทั้งหมด 8 เฟส ไม่เพียงเท่านั้น เรายังสามารถ เพิ่มจำนวนข้อมูลได้โดยการกำหนดให้แต่ละมุมมีค่าแอมปลิจูดได้มากกว่า 1 ค่า ดังรูปที่ 2.5



รูปที่ 2.5 มุมเฟสการส่งผ่านข้อมูลความเร็ว 9600 บิท/วินาที

2.4 รูปแบบการสื่อสารข้อมูลดิจิตอล

ในการเชื่อมต่ออุปกรณ์สองตัวเข้าด้วยกันผ่านตัวกลางการส่งผ่าน เพื่อแลกเปลี่ยนข้อมูลจะมีการทำงานร่วมกันอย่างมีกฎเกณฑ์ ตัวอย่างเช่น ข้อมูลที่ถูกส่งเป็นหนึ่งบิทที่เวลาใดเวลาหนึ่งในตัวกลางระยะเวลาของบิทเหล่านี้ (rate , duration , spacing) จะต้องเท่ากันทั้งฝ่ายรับและฝ่ายส่ง การสื่อสารข้อมูลนี้แบ่งเป็นสองชนิดตามรูปแบบของการส่งสัญญาณคือ asynchronous และ synchronous

การส่งข้อมูลแบบอะซิงโครนัส และแบบซิงโครนัส

เราจะกล่าวถึงการส่งผ่านข้อมูลแบบอนุกรม ซึ่งข้อมูลจะถูกเคลื่อนไปบนทางเดินของการสื่อสารเพียงสายเดียว แทนที่จะเป็นกลุ่มของสายที่ขนานกันที่เรียกว่า บัส การส่งข้อมูลแบบอนุกรมนั้น องค์ประกอบที่สำคัญของสัญญาณจะถูกส่งลงไปบนสายเรียงกันไปครั้งละหนึ่งบิท ซึ่งการซิงโครไนซ์เป็นหัวใจสำคัญในงานทางด้านการสื่อสารข้อมูล โดยตัวส่งจะต้องส่งข่าวสาร 1 บิท ที่เวลาหนึ่งผ่านตัวกลางไปยังผู้รับ ผู้รับจะต้องแยกให้ได้ว่าที่ตำแหน่งใดเป็นตำแหน่งเริ่มต้นหรือตำแหน่งสิ้นสุดของบล็อกของบิท และจะต้องรู้ถึงค่าระยะเวลาของสัญญาณ 1 บิท เพื่อที่จะได้สามารถสุ่มสัญญาณในสายได้ถูกต้องและอ่านค่าของแต่ละบิทที่ได้

ตัวอย่างเช่น ผู้รับจะพยายามสุ่มเอาค่าสัญญาณจากตัวกลางที่ตำแหน่งเวลากึ่งกลางของแต่ละบิท แต่ถ้ามีความแตกต่างของเวลาเกิดขึ้นระหว่างตัวรับและตัวส่ง 5% ของแต่ละบิท ดังนั้นที่การสุ่มค่าตัวอย่างของสัญญาณครั้งที่ 10 จะทำให้ตัวรับเกิดการเก็บข้อมูลผิดพลาด แต่สำหรับความแตกต่างของเวลาบิทที่น้อยมาก ๆ ความผิดพลาดก็จะเกิดขึ้นที่ตำแหน่งไกลออกไปอีก ถึงกระนั้นก็ดีก็ยังทำให้ตัวรับและตัวส่งเกิดการซิงค์โครไนซ์กันไม่ได้

มีวิธีการง่าย ๆ อยู่สองวิธีที่ใช้ในการแก้ปัญหาการซิงค์ วิธีการแรกเรียกว่า Asynchronous Transmission ซึ่งเป็นวิธีการที่บิททั้งหลายถูกส่งไปเป็นบล็อก ๆ ละหนึ่งตัวอักษร โดยปรกติแต่ละอักขระจะมีความยาว 5 ถึง 8 บิท เวลาหรือการซิงค์จะถูกรักษาให้อยู่แต่เพียงในแต่ละอักขระ ตัวรับจะถือโอกาสซิงค์โครไนซ์ใหม่ที่จุดเริ่มต้นของ

อักขระใหม่แต่ละตัว สำหรับอีกวิธีหนึ่งคือ Synchronous Transmission จะเป็นการส่งผ่านสัญญาณข้อมูลที่มีบล็อกที่ยาวมากใน 1 ครั้ง ซึ่งตัวรับจะต้องรักษาการซิงโครนัสกับตัวส่งเป็นระยะเวลาที่ยาวนาน

เราอาจจะกล่าวได้ว่ามีงานที่จะต้องทำในการควบคุมการซิงค์โครไนซ์อยู่ดังต่อไปนี้

- Bit synchronization : จะต้องจำได้ว่าตรงไหนเป็นการเริ่มต้นและสิ้นสุดของแต่ละบิท
- Character or word synchronization : จะต้องมีการจำ การเริ่มต้นและสิ้นสุดของแต่ละอักขระ หรือก็คือ ข้อมูลหน่วยเล็ก ๆ
- Block or message : จะต้องมีการจำการเริ่มต้นและสิ้นสุดของข้อมูลหน่วยใหญ่

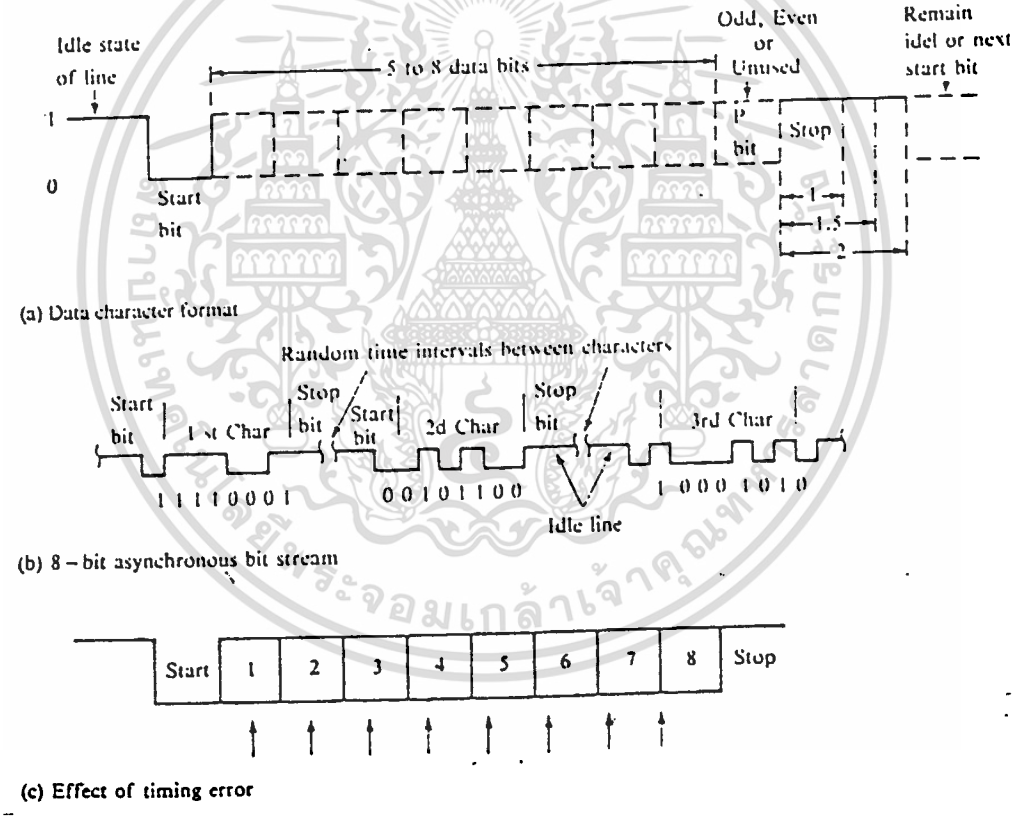
Asynchronous Transmission

เป็นการส่งบล็อกที่ประกอบด้วยบิทจำนวนน้อย ๆ ในแต่ละบล็อก และทำการซิงโครนัสใหม่ทุกครั้ง ที่ตำแหน่งเริ่มต้นของแต่ละบล็อก และวิธีที่เป็นที่รู้จักก็ได้แก่ start - stop bit การส่งผ่านแบบอะซิงโครนัส นับว่าเป็นการส่งแบบที่เรียกว่า character-oriented โดยมีจำนวนบิทใน 1 อักขระจะกำหนดให้มีได้ตั้งแต่ 5 ถึง 8 บิท

เทคนิคอันนี้สามารถอธิบายได้ง่าย ๆ โดยอ้างอิงรูป 2.6 คือเมื่อไม่มีการส่งอักขระสายระหว่างผู้รับและผู้ส่งจะอยู่ในสถานะ " idle " ซึ่งตามมาตรฐานจะกำหนดให้มีสถานะเป็นมาร์ค (1) ที่ตำแหน่งเริ่มต้นของอักขระที่เรียกว่า บิทเริ่มต้น (start bit)ค่าของไบนารี จะมีค่าเป็น "0" ซึ่งต่อไปก็จะตามด้วยกลุ่มอักขระตั้งแต่ 5 ถึง 8 บิท ในบางกรณีก็จะมีบิทพาริตี้ด้วย บิทพาริตี้นี้จะกำหนดโดยผู้ส่ง เพื่อให้ผู้รับใช้สำหรับตรวจจับความผิดพลาด และบิทสุดท้ายของอักขระจะตามด้วย บิทสิ้นสุด (stop bit) ซึ่งมีค่าไบนารีเป็น "1" ความยาวต่ำสุดของบิทสิ้นสุดจะกำหนดให้เป็น 1 บิท โดยทั่วไปแล้วจะกำหนดให้เป็นได้ 1, 1.5, 2 เท่าของบิท สำหรับค่าที่นานที่สุดจะไม่ถูกกำหนดเพราะว่าบิทสิ้นสุด

จะไปเหมือนสถานะ "idle" ตัวส่งจะส่งบิทสิ้นสุดต่อเนื่องจนกระทั่งมันพร้อมที่จะส่งอักขระตัวถัดไป ถ้าขบวนของอักขระถูกส่งอย่างต่อเนื่องคงที่สม่ำเสมอ ช่องว่างระหว่างอักขระจะมีค่าเท่ากันตลอดซึ่งจะเท่ากับบิทสิ้นสุด ตัวอย่างเช่น ถ้าบิทสิ้นสุดมีความยาว 1 บิท และมีการส่งอักขระ ABC โดยอาศัยรหัสแอสกี้ (ASCII)ไม่มีพาริตี้บิทรูปแบบจะเป็นดังนี้

0100000110010000110110000011...1111 เริ่มต้นด้วยบิทเริ่มต้น (0) และตามด้วยบิทที่เป็นอักขระแอสกี้ 7 บิท และตามด้วยบิทสิ้นสุดในสถานะ " idle " ตัวรับจะมองดูการเปลี่ยนสถานะจาก 1 เป็น 0 เมื่อเริ่มต้นตัวอักขระแล้วทำการสุ่มสัญญาณที่ตำแหน่งระหว่างบิทหนึ่ง ๆ ทั้งหมด 7 ครั้ง และจะจับดูการเปลี่ยนแปลงสถานะจาก 1 ไปเป็น 0 เพื่อทำการรับอักขระตัวต่อไป



รูปที่ 2.6 รูปแบบการส่งผ่านแบบอะซิงโครนัส

ความจำเป็นในเรื่องความถูกต้องทางเวลาสำหรับโครงสร้างนี้จะถูกกำหนดไว้พอประมาณ ตัวอย่างเช่น อักขระแบบแอสกี้ชนิด 8 บิทรวมทั้งพาริตี้บิทด้วยนั้น ถ้าตัวรับช้า

หรือเร็วกว่าตัวส่ง 5% บิทข่าวสาร ตัวที่ 8 จะถูกสุ่มผิดตำแหน่งไปจากกลางบิท 45% ซึ่งก็ยังคงได้ค่าที่ถูกต้อง รูปที่ 2.6c แสดงให้เห็นถึงอิทธิพลเนื่องจากความผิดพลาดทางเวลาที่มีขนาดพอสมควรที่จะทำให้เกิดความผิดพลาดในการรับ

Synchronous Transmission

แม้ว่าชนิดของเฟรมจะเป็นตัวกำหนดความแตกต่างระหว่างการส่งข้อมูลแบบซิงโครนัสและอะซิงโครนัสก็ตาม แต่ความแตกต่างของหลักการพื้นฐานระหว่างวิธีทั้งสองก็คือ การส่งผ่านแบบอะซิงโครนัส สัญญาณนาฬิกาที่ควบคุมการส่งผ่านของตัวส่งและตัวรับไม่จำเป็นที่จะต้องซิงค์กันพอดีก็ได้ ในขณะที่การส่งผ่านแบบซิงโครนัส สัญญาณนาฬิกาของตัวส่งและตัวรับจะต้องซิงค์กันพอดี ซึ่งอาจจะทำได้โดยการต่อสายเพิ่มระหว่างอุปกรณ์ทั้งสองเพื่อส่งสัญญาณนาฬิกาไปด้วย แต่อย่างไรก็ตามในทางปฏิบัติ วิธีการที่ทำกันปกติคือใช้สายข้อมูลเส้นเดียว แต่ผากข้อมูลของสัญญาณนาฬิกามาร่วมกันกับรูปคลื่นที่ใช้ส่ง โดยวิธีการนี้สัญญาณนาฬิกาในการสุ่มของตัวรับจะต้องถูกแยกออกมาจากขบวนสัญญาณข้อมูลที่เข้ามาด้วยวงจรแยกสัญญาณที่เหมาะสม

มีวิธีการให้เลือกอยู่ 2 วิธีในการจัดการกับการเชื่อมต่อข้อมูลแบบ ซิงโครนัส ซึ่งได้แก่การจัดแบบ character oriented (or byte) และการจัดแบบ bit oriente ข้อแตกต่างที่สำคัญของทั้งสองวิธีได้แก่ วิธีการในการหาจุดเริ่มต้นและสิ้นสุดเฟรม ในระบบการจัดแบบ bit oriented ตัวรับสามารถตรวจตัดสินการสิ้นสุดของเฟรมที่บิทใดบิทหนึ่ง โดยไม่ต้องขึ้นกับการจำกัดวงอยู่กับขอบเขตของ 8 บิท ซึ่งวิธีนี้หมายความว่าเฟรมอาจจะมีความยาว N บิท อย่างไรก็ตามในทางปฏิบัติ โครงสร้างนี้ไม่ค่อยได้ใช้เพราะว่าในการใช้งานส่วนใหญ่เราจะใช้เฟรมซึ่งมีความยาวเป็นจำนวนเท่าของ 8 บิท อย่างไรก็ดีระบบการส่งแบบ bit oriented นั้นจะมีศักยภาพที่จะเพิ่ม throughput ขึ้นเป็น 2 เท่า

Character oriented

รูปแบบของ character oriented แต่ละเฟรมจะถูกจัดให้มีจำนวนบิทเป็นจำนวน

เท่าของ 1 ตัวอักษร (7 หรือ 8 บิท) ซึ่งจะถูกส่งไปเป็นสายต่อเนื่องโดยไม่มีการขาดช่วง ดังนั้นอุปกรณ์ทางด้านรับจึงสามารถสร้างกระบวนการแยกสัญญาณนาฬิกาที่สัมพันธ์ (synchronize) กับทางเครื่องส่งคืนมาได้ทำให้สามารถ

1) ตรวจจับการเริ่มต้นและการสิ้นสุดของแต่ละตัวอักษร เรียกว่า การซิงโครไนซ์ตัวอักษร (character synchronism) และ

2) ตรวจตัดสินการเริ่มต้นและการสิ้นสุดของเฟรมที่สมบูรณ์ หรือเรียกว่า การซิงโครไนซ์เฟรม (frame synchronism)

ได้มีหลักการต่าง ๆ เกิดขึ้นมากมายเพื่อให้บรรลุถึงเป้าหมายดังที่กล่าวโดยมีจุดมุ่งหมายหลักคือ การสร้างกระบวนการซิงโครไนซ์เซชั่นที่ไม่ขึ้นกับเนื้อหาของข้อมูลในเฟรมหลักซึ่งเรียกการซิงโครไนซ์แบบนี้ว่า โปร่งใส (transparent) จากเนื้อหาของเฟรม หรือเรียกว่า ข้อมูลโปร่งใส (data transparent) การจัดเฟรมแบบ character oriented ที่เป็นที่นิยมคือการใช้โปรโตคอลควบคุมแบบไบนารีซิงโครนัส (binary synchronous control protocal) ที่รู้จักกันในชื่อ "Basic Mode" ซึ่งส่วนใหญ่ใช้ในการส่งถ่ายข้อมูลตัวอักขระระหว่างคอมพิวเตอร์กับสถานีข้อมูลปลายทาง

รูปที่ 2.9a แสดงตัวอย่างของรูปแบบหนึ่งที่ใช้โปรโตคอลนี้ที่ใช้ส่งบล็อกข้อมูลเฟรมข่าวสาร การซิงโครไนซ์ตัวอักษรจะทำได้โดยเครื่องส่งจะส่งอักษรพิเศษสำหรับการซิงค์ SYN มักใช้ 2 ตัวหรือมากกว่าในทันทีก่อนที่จะส่งเฟรมทุก ๆ เฟรม ในเวลาเริ่มต้นหรือหลังจากคาบเวลาของ " idle " เครื่องรับจะทำการกวาดตรวจขบวนบิทที่รับมาได้ทีละบิทจนกระทั่งพบรูปแบบของตัวอักษร SYN ที่รู้จัก เครื่องรับก็จะสามารถทำการซิงโครไนซ์ได้ และขบวนที่ตามมาจะถูกพิจารณาจัดเป็นกลุ่มที่ประกอบเป็นตัวอักษรกลุ่มละ 7 หรือ 8 บิท ตามชนิดของระบบที่ได้จัดเตรียมไว้

ตัวอักษร STX หมายถึง start-of-text ใช้เพื่อเป็นสัญญาณบอกการเริ่มต้นเฟรม และตัวอักษร ETX หมายถึง end-of-text เพื่อเป็นสัญญาณบอกการสิ้นสุดของเฟรม ดังนั้น ตัวอักษรแต่ละตัวในเฟรมที่รับมาหลังจากรับ STX แล้ว จะถูกเปรียบเทียบกับ ETX ถ้าไม่ใช่ ETX ก็จะถูกเก็บเอาไว้

1) การควบคุมความผิดพลาดล่วงหน้า (Forward error control)

ในการส่งอักษรหรือเฟรมแต่ละครั้งจะมีการเพิ่มข่าวสารส่วนเกิน (redundant) เข้าไปเพื่อให้เครื่องรับไม่เพียงแต่ตรวจสอบว่ามีความผิดพลาดเท่านั้น แต่จะสามารถทราบว่ามีความผิดพลาดที่ใดและแก้ไขให้ถูกต้องได้ด้วย

2) การควบคุมความผิดพลาดด้วยการป้อนกลับ (Feed error control)

ในการส่งอักษรหรือเฟรมในแต่ละครั้ง จะมีการเพิ่มข่าวสารที่เพียงพอที่จะทำให้ผู้รับสามารถที่จะตรวจจับความผิดพลาดที่เกิดขึ้นได้ แล้วก็ขอร้องให้มีการส่งข้อมูลชุดเดิมซ้ำกลับมาใหม่ ดังนั้นการควบคุมความผิดพลาดด้วยการป้อนกลับจะเป็นวิธีการที่นิยมมากกว่าในระบบการสื่อสารต่าง ๆ ในปัจจุบัน

การควบคุมความผิดพลาดแบบป้อนกลับ สามารถแบ่งออกเป็น 2 ส่วน ส่วนที่ 1 เป็นส่วนที่เพื่อทำให้เกิดความมั่นใจว่าความผิดพลาดที่เกิดขึ้นสามารถที่จะถูกตรวจสอบพบ และส่วนที่ 2 จะเป็นอัลกอลิธึมที่นำมาใช้ในการควบคุม การส่งข้อมูลซ้ำกลับมาใหม่

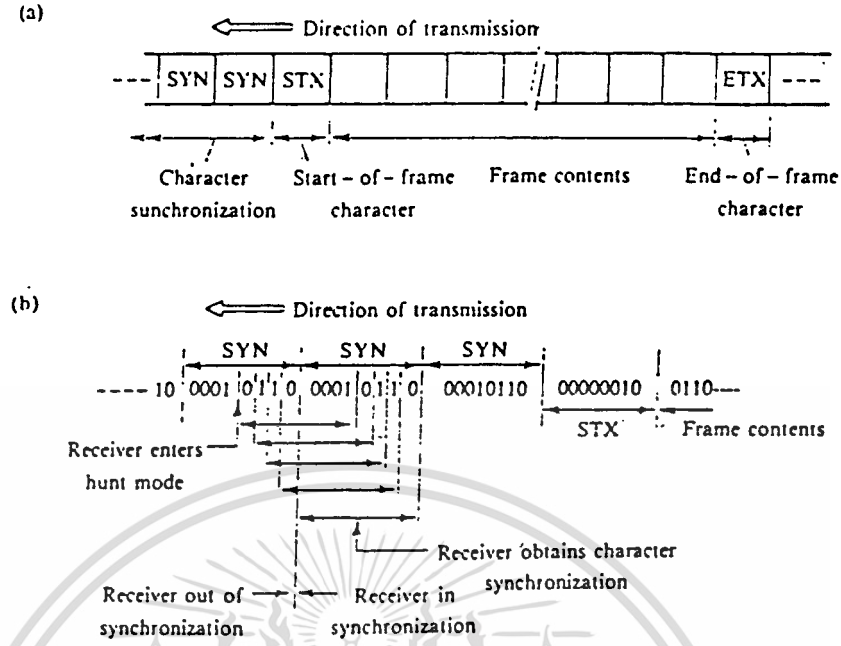
เทคนิคการตรวจจับความผิดพลาดที่นิยมใช้กันมากในปัจจุบัน ได้แก่

พาริตี้ (Parity)

เป็นวิธีที่ง่ายที่สุดที่ใช้ในการตรวจสอบความผิดพลาด เมื่อจำนวนบิทข่าวสารมีจำนวนน้อย ๆ และความน่าจะเป็นของการเกิดความผิดพลาดต่ำ ๆ วิธีการนี้ใช้การเพิ่มพาริตี้บิทเข้ามาอีกหนึ่งบิทต่อการส่ง 1 เฟรม หรือ 1 ตัวอักษร วิธีนี้เหมาะกับการส่งผ่านแบบอะซิงโครนัส

ในการใช้วิธีของพาริตี้ บิทข้อมูลในแต่ละตัวอักษร จะต้องถูกตรวจก่อนจะมีการส่งผ่านและจะมีการคำนวณค่าพาริตี้ โดยบิทพาริตี้ที่คำนวณได้จะถูกเพิ่มเข้าไปเพื่อให้จำนวน 1 เป็นคี่หรือเป็นคู่ตามที่เรากำหนดว่าใช้การตรวจพาริตี้คี่หรือคู่ ทางฝ่ายรับก็จะคำนวณหาค่าพาริตี้เมื่อรับตัวอักษรเข้ามา และหาว่ามีความผิดพลาดเกิดขึ้นหรือไม่ แสดงดังรูป 2.9

ความสามารถของการตรวจจับความผิดพลาดแต่ละชนิดจะได้ผลดีหรือไม่ก็ขึ้นอยู่กับชนิดของความผิดพลาดที่เกิดขึ้นในระหว่างการสื่อสาร สำหรับการตรวจพาริตี้นี้จะป้องกันความผิดพลาดที่เกิดขึ้นได้เพียง 1 บิท ถ้ามีความผิดพลาดเกิดขึ้น 2 บิทจะไม่สามารถตรวจพบได้



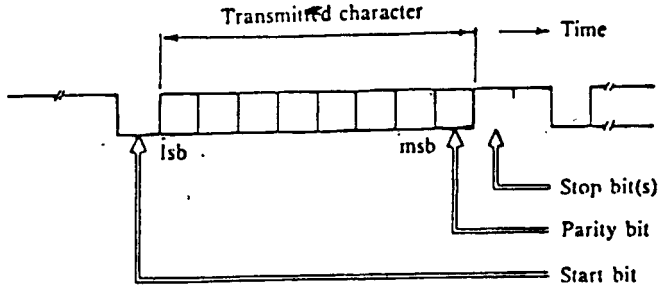
รูป 2.7 รูปแบบการส่งผ่านแบบซิงโครนัสชนิด (character-oriented)

Bit oriented

สำหรับ bit oriented นี้จะขอกล่าวเพียงสั้นเท่านั้นคือ แต่ละเฟรมที่ส่งออกไปอาจจะประกอบไปด้วยจำนวนบิทเท่าไรก็ได้ ซึ่งไม่จำเป็นต้องเป็นจำนวนเท่าของ 8

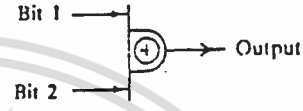
วิธีการตรวจสอบความผิดพลาด (ERROR-DETECTION METHODS)

ในระหว่างการส่งผ่านข้อมูล โดยธรรมชาติแล้วมักจะอยู่ในสภาพแวดล้อมที่มีการรบกวนจากสัญญาณไฟฟ้า เช่นการสวิตช์ภายในโครงข่ายโทรศัพท์ สัญญาณไฟฟ้าที่แทนขบวนบิทที่ใช้ส่งผ่านข้อมูลถูกเปลี่ยนแปลงด้วยสนามแม่เหล็กจากการแทรกสอดโดยอุปกรณ์ไฟฟ้าที่อยู่ใกล้เคียง ซึ่งสัญญาณไบนารี " 1 " อาจถูกแปลความหมายโดยตัวรับเป็นสัญญาณ " 0 " หรืออาจกลับกัน เพื่อให้แน่ใจว่าข่าวสารที่ได้รับปลายทางจะมีความเป็นไปได้สูงในการที่จะรับข้อมูลที่แท้จริงที่เครื่องส่งส่งมาให้ ซึ่งถ้าเครื่องรับตรวจพบความผิดพลาด ก็จะต้องมีกระบวนการที่ใช้สำหรับให้ได้มาซึ่งข่าวสารที่ถูกต้องกลับคืนมาด้วย มีอยู่ 2 วิธีการ คือ

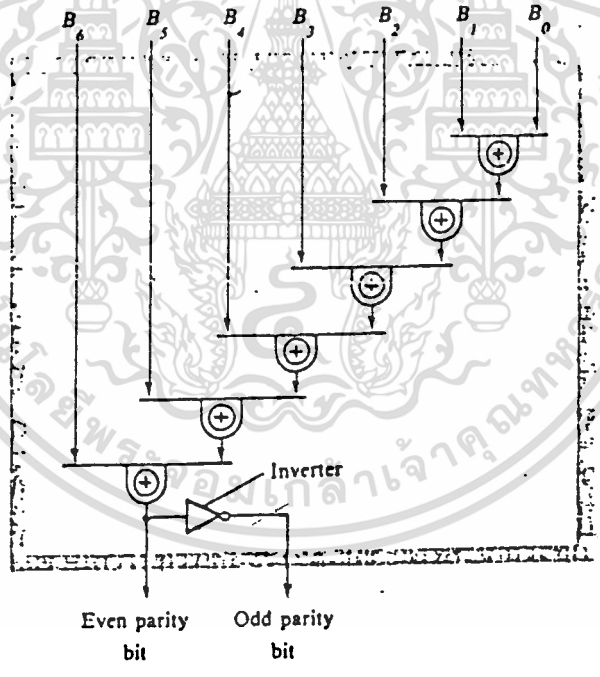


(b)

Bit 1	Bit 2	XOR
0	0	0
0	1	1
1	0	1
1	1	0



(c)



พาริตี้ถูกนำไปใช้กับทั้งวิธีการส่งผ่านแบบอะซิงโครนัสและวิธีการส่งผ่านแบบซิงโครนัสชนิด character-oriented เพราะมีความสะดวกหลาย ๆ ด้าน คือ

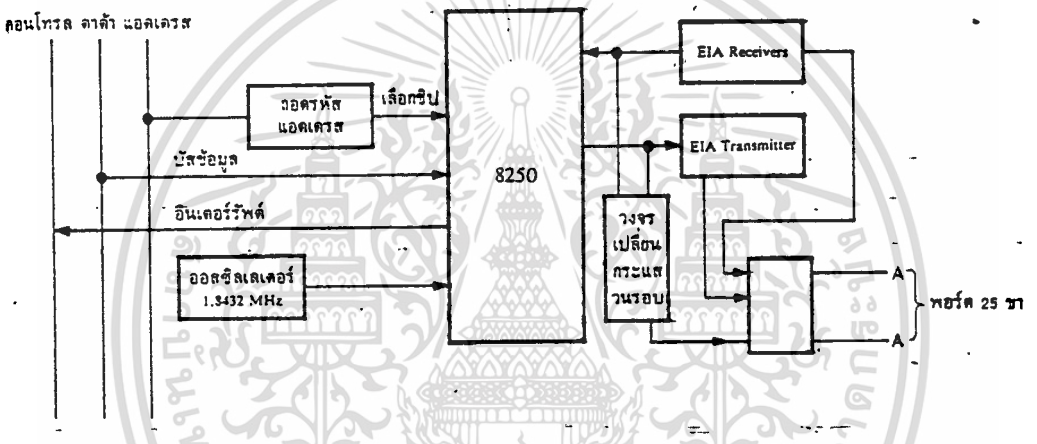
1) สามารถคำนวณและใส่พาริตี้ที่เหมาะสมเข้าไปในการส่งผ่านแต่ละตัวอักษรได้อย่างอัตโนมัติก่อนจะมีการส่งผ่าน

2) สามารถคำนวณพาริตี้ใหม่ซ้ำ สำหรับตัวอักษรทุก ๆ ตัวที่รับมาได้และจะส่งสัญญาณออกมาให้ทราบเมื่อมีการตรวจพบความผิดพลาด

ค่าเอาท์พุทของ XOR จะเหมือนกับการบวกเลขสองจำนวนโดยไม่มีการทด คู่แรกของบิทที่มีความสำคัญน้อยที่สุดจะถูก XOR กันก่อน และนำเอาท์พุทที่ได้ไป XOR กับบิทที่มีความสำคัญมากขึ้นตามลำดับจนหมด ตัวแสดงในรูป 2.8c เอาท์พุทตัวสุดท้ายก็จะเป็นพาริตี้บิทที่ต้องการ ที่เครื่องรับบิทพาริตี้จะถูกคำนวณขึ้นและเปรียบเทียบกับพาริตี้บิทที่ได้มา ถ้าไม่ตรงกันแสดงว่าเกิดความผิดพลาดขึ้น

2.5 โครงสร้างของพอร์ตสื่อสาร

พอร์ตสื่อสารของเครื่องไมโครคอมพิวเตอร์ 16 บิตที่จะกล่าวถึงเป็นระบบมาตรฐานตามแบบเครื่องไอบีเอ็มพีซีเอ็กซ์ที โครงสร้างบล็อกไดอะแกรมแสดงดังรูป



รูปที่ 2.4 โครงสร้างพอร์ตสื่อสารข้อมูลที่ใช้ 8250

พิจารณาได้ว่า 8250 เป็นตัวรับข้อมูลจากบัสของระบบ ซึ่งก็คือสล็อตขนาด 31*2 (62)ขาของระบบนั่นเอง ซีพียูติดต่อกับ 8250 ในลักษณะพอร์ตที่เป็นอินพุตเอาต์พุต การจัดพอร์ตนี้กำหนดหมายเลขพอร์ตอย่างเจาะจง ระบบไมโครคอมพิวเตอร์ 16 บิต มีพอร์ตสื่อสารสองพอร์ต คือ คอม 1 (COM_1) และ คอม 2 (COM_2) ทั้ง COM_1และ COM_2 มีหมายเลขอินพุตเอาต์พุตพอร์ตดังตารางที่ 2.1

อินพุตเอาต์พุตพอร์ต		เลือกรีจิสเตอร์	สถานะDLAB
พอร์ต COM1	พอร์ต COM2		
3F8	2F8	บัฟเฟอร์ IX	DLAB = 0(เขียน)
3F8	2F8	บัฟเฟอร์ RX	DLAB = 0(อ่าน)
3F8	2F8	แลตช์ตัวหาร	DLAB = 1
3F9	2F9	แลตช์ตัวหาร	DLAB = 1
3F9	2F9	อีนาเบิลอินเตอร์รัพต์	
3FA	2FA	กำหนดอินเตอร์รัพต์	
3FB	2FB	ควบคุมสายสื่อสาร	
3FC	2FC	ควบคุมโมเด็ม	
3FD	2FD	แสดงสถานะสายสื่อสาร	
3FE	2FE	แสดงสถานะโมเด็ม	

ตารางที่ 2.1 หมายเลขอินพุตเอาต์พุตพอร์ตของ COM_1และ COM_2

การเลือกหมายเลขอินพุตเอาต์พุตพอร์ตแยกเป็นสองกลุ่ม กลุ่มหนึ่งคือ COM_1 จะกำหนดหมายเลขพอร์ตจาก 3F8-3FE อีกกลุ่มหนึ่งถ้ากำหนดหมายเลขพอร์ตเป็น 2F8-2FE ในการเลือกหมายเลขรีจิสเตอร์ภายในกำหนดด้วยแอดเดรส 3 บิต คือ A_0, A_1 และ A_2 สำหรับการเลือก COM_1และ COM_2 เราใช้แอดเดรส A_8 เป็นตัวเลือก การเลือกนี้กำหนดเป็นตารางได้ดังตารางที่ 2.2

การอินเตอร์รัพท์

โครงสร้างการควบคุมอินเตอร์รัพต์ของชิป 8259 ได้จัดลำดับการอินเตอร์รัพต์ไว้ 8 ระดับ สัญญาณรับอินเตอร์รัพต์ที่เข้าทางชิป 8259 มี 8 เส้น คือ IRQ_0-IRQ_7 สำหรับกรณีของระบบสื่อสารอนุกรมได้ กำหนดสัญญาณการอินเตอร์รัพต์ไว้แล้วคือให้ IRQ_4 เป็นสัญญาณอินเตอร์รัพต์ของระบบ COM_1 และ IRQ_3 เป็นของ COM_2 ในการที่จะส่งสัญญาณอินเตอร์รัพต์ต้องให้บิต 3 ของรีจิสเตอร์ควบคุมโมเด็มได้รับการเซตค่าเป็น "1" ก่อน จากนั้นข้อมูลอินเตอร์รัพต์ที่อยู่ในรีจิสเตอร์อีนาเบิลอินเตอร์รัพต์จะเป็นตัวสั่งการอินเตอร์รัพต์

แอดเดรส 3F8-3FF และ 2F8-2FE

A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	DLAB	รีจีสเตอร์
1	1/0	1	1	1	1	1	X	X	X		
							0	0	0	0	บัฟเพอร์สำหรับตัวรับข้อมูล (อ่าน) โฮลดิงสำหรับตัวส่งข้อมูล (เขียน)
							0	0	1	0	อีนาเบิลอินเตอร์รัพต์
							0	1	0	X	กำหนดอินเตอร์รัพต์
							0	1	1	X	ควบคุมสายสื่อสาร
							1	0	0	X	แสดงสถานะสายสื่อสาร
							1	1	0	X	แสดงสถานะโมเด็ม
							1	1	1	X	ไม่ใช้
							0	0	0	1	แลตช์ตัวหาร (LSB)
							0	0	1	1	แลตช์ตัวหาร (MSB)

ตารางที่ 2.2 การกำหนดแอดเดรสสำหรับหมายเลขพอร์ตต่าง ๆ

การเลือกหมายเลขอินพุตเอาต์พุตพอร์ตแยกเป็นสองกลุ่ม กลุ่มหนึ่งคือ COM_1 จะกำหนดหมายเลขพอร์ตจาก 3F8-3FE อีกกลุ่มหนึ่งถ้ากำหนดหมายเลขพอร์ตเป็น 2F8-2FE ในการเลือกหมายเลขรีจิสเตอร์ภายในกำหนดด้วยแอดเดรส 3 บิต คือ A_0, A_1 และ A_2 สำหรับการเลือก COM_1และ COM_2 เราใช้แอดเดรส A_8 เป็นตัวเลือก การเลือกนี้กำหนดเป็นตารางได้ดังตารางที่ 2.2

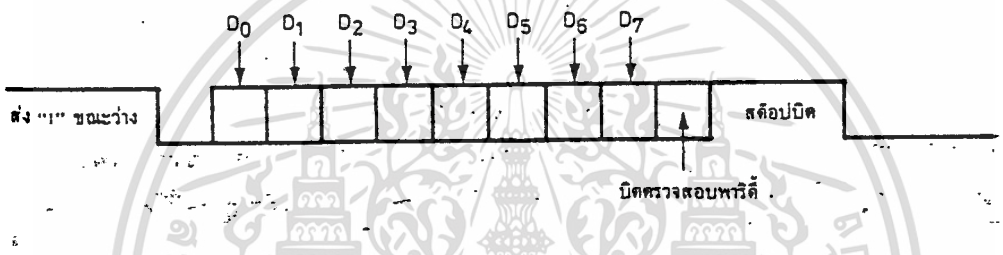
การอินเตอร์รัพท์

โครงสร้างการควบคุมอินเตอร์รัพต์ของชิป 8259 ได้จัดลำดับการอินเตอร์รัพต์ไว้ 8 ระดับ สัญญาณรับอินเตอร์รัพต์ที่เข้าทางชิป 8259 มี 8 เส้น คือ IRQ_0-IRQ_7 สำหรับกรณีของระบบสื่อสารอนุกรมได้ กำหนดสัญญาณการอินเตอร์รัพต์ไว้แล้วคือให้ IRQ_4 เป็น

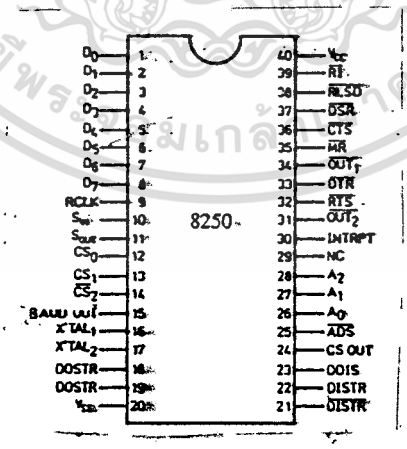
สัญญาณอินเตอร์รัพต์ของระบบ COM_1 และ IRQ_3 เป็นของ COM_2 ในการที่จะส่งสัญญาณอินเตอร์รัพต์ต้องให้บิต 3 ของรีจิสเตอร์ควบคุมโมเด็มได้รับการเซตค่าเป็น "1" ก่อน จากนั้นข้อมูลอินเตอร์รัพต์ที่อยู่ในรีจิสเตอร์อีนาเบิลอินเตอร์รัพต์จะเป็นตัวสั่งการอินเตอร์รัพต์

รูปแบบข้อมูลที่รับหรือส่ง

การสื่อสารข้อมูลของระบบนี้เป็นการสื่อสารแบบอะซิงโครนัส รูปแบบของข้อมูลจะมีสตาร์ตบิต บิตตรวจสอบพาริตี้ และสต็อปบิต โครงสร้างของข้อมูลแต่ละเฟรมเป็นดังรูปที่ 2.10



รูปที่ 2.10 รูปแบบข้อมูล 1 เฟรม



รูป2.11การจัดวางขาของไอซี

ข้อมูลบิตแรกของการส่งเป็นสตาร์ตบิต ระบบจะส่งสตาร์ตบิตก่อนหลังจากนั้นจะนั้นตามด้วยข้อมูลและแทรกด้วยบิต ตรวจสอบพาริตี้ตามด้วยสต็อปบิต ขนาดของข้อมูลมีค่าได้ตั้งแต่ 5-8 บิต แต่ทั่ว ๆ ไปใช้ 8 บิต สต็อปบิตมีได้ 1 ,1.5 หรือ 2 บิต ค่าเหล่านี้สามารถกำหนดลงไปในรีจิสเตอร์ควบคุมสายสื่อสาร

ชิป 8250

8250 เป็นไอซีขนาด 40 ขามีการทำงานเพื่อควบคุมสิ่งต่าง ๆ ที่เกี่ยวกับการสื่อสารแบบอนุกรมได้หมด โครงสร้างทางฮาร์ดแวร์ของอะแดปเตอร์การ์ดนี้จึงไม่ยุ่งยากมากนัก การจัดวางขาของไอซี 8250 มีรายละเอียดดังรูปที่ 2.15

ขาสัญญาณอินพุต ชิป CS_0,CS_1,CS_2(chip select) คือขา 12 ขา 13 และขา 14 ตามลำดับ เป็นสัญญาณเลือกชิป โดยที่เมื่อต้องการเลือกชิปจะให้ CS_0,CS_1เป็น "1" และ CS_2 เป็น "0" สัญญาณเลือกชิปนี้จะได้รับการเลือกแลตช์ไว้ในขณะที่สัญญาณ ADS มีค่าเป็น "0" การเลือกชิปนี้จะมีไว้เพื่อให้ซีพียูติดต่อกับ 8250

ขาสโตรบข้อมูลอินพุต (data input strobe) คือ DUSTR (ขา 22) $\overline{\text{DISTR}}$ (ขา 21)เมื่อสัญญาณที่ DISTR เป็น "1" และ $\overline{\text{DISTR}}$ เป็น "0" ในขณะที่มีการเลือกชิปเป็นขณะที่ซีพียูจะย้ายข้อมูลจากรีจิสเตอร์ภายในที่มีการกำหนดไว้แล้วมายังซีพียู สัญญาณนี้จึงเป็นสัญญาณอ่านข้อมูลหรือ read นั่นเอง

ขาสโตรบข้อมูลเอาต์พุต คือ DOSTR (ขา 19) $\overline{\text{DOSTR}}$ (ขา 18) เป็นสัญญาณที่แอกทีฟขึ้นเพื่อให้ซีพียูเขียนข้อมูลลงมายังรีจิสเตอร์ของ 8250

ขาสโตรบแอดเดรส คือ $\overline{\text{ADS}}$(ขา 25) เมื่อมีค่าเป็น "0" จะแอกทีฟเพื่อแลตช์ค่า A_0-A_1 เลือกรีจิสเตอร์ภายใน การเลือกรีจิสเตอร์จะทำขณะที่การเลือกชิปแอกทีฟอยู่

ขาเลือกรีจิสเตอร์ คือ A_0(ขา 26),A_1(ขา 27),A_2(ขา 28) ตามลำดับ เป็นตัวกำหนดแอดเดรสของรีจิสเตอร์ภายใน เพื่อให้ซีพียูทำการติดต่อกับ 8250 ตามค่ารีจิสเตอร์ที่กำหนดเพื่อการเขียนหรืออ่าน อย่างไรก็ตามการทำงาน A_0-A_2 นี้ก็ขึ้นกับสัญญาณเลือกตัวหาร LAB-divisor latch access bit ซึ่งกำหนดค่ารีจิสเตอร์ได้ดังตารางที่ 2.3

ขารีเซต (master reset) คือ MR(ขา 35)เมื่อมีค่าเป็น "1" จะรีเซตการทำงานของชิป 8250 โดยทำให้ค่าต่าง ๆ ในรีจิสเตอร์ถูกเคลียร์หมด (ยกเว้นบัฟเฟอร์ของตัวรับ ตัวส่งและตัวหาร) ขณะทำการรีเซตจะมีผลต่อสัญญาณเอาต์พุตด้วย ผลของการรีเซตแสดงไว้ในตารางที่ 2.3

DLAB	A2	A1	A0	รีจิสเตอร์
0	0	0	0	บัฟเฟอร์สำหรับตัวรับข้อมูล (อ่าน)
0				โฮลดิงสำหรับตัวส่งข้อมูล (เขียน)
0	0	0	1	อีนาเบิลอินเตอร์รัพต์
X	0	1	0	กำหนดอิสเตอร์รัพต์
X	0	1	1	ควบคุมสายสื่อสาร
X	1	0	0	ควบคุมโมเด็ม
X	1	0	1	แสดงสถานะสายสื่อสาร
X	1	1	0	แสดงสถานะโมเด็ม
X	1	1	1	ไม่ใช้
1	0	0	0	แลตช์ตัวหาร (LSB)
1	0	0	1	แลตช์ตัวหาร (MSB)

ตารางที่ 2.3 การกำหนดค่ารีจิสเตอร์

ขาสัญญาณนาฬิกาตัวรับ (receiver clock) คือ RCLK(ขา 9) เป็นขาที่ตัวรับสัญญาณนาฬิกาเพื่อกำหนดอัตราบอด สัญญาณนาฬิกานี้จะมีค่าเป็น 16 เท่าของที่นำมาใช้

ขาอินพุตข้อมูลอนุกรม (serial input) คือ SIN (ขา 10) เป็นขารับข้อมูลอนุกรมจากสายส่งในการเชื่อมโดยการติดต่อสื่อสาร

ขาเคลียร์ทูเซนด์ (clear to send) คือ $\overline{\text{CTS}}$(ขา 36) เป็นสัญญาณที่ใช้ในการติดต่อกับโมเด็ม เงื่อนไขของสัญญาณนี้สามารถเก็บไว้ภายในชิป 8250 ที่จะให้ซีพียูอ่านไป

ตรวจสอบได้โดยเก็บไว้ที่บิต 4 ของรีจิสเตอร์ แสดงสถานะโมเด็ม ส่วนบิตของรีจิสเตอร์แสดงสถานะจะเป็นตัวบอกว่า CTSได้เปลี่ยนสถานะไปหลังจากการอ่านครั้งก่อนแล้วหรือไม่

ขาดาต้าเซตรีดี้ (data set ready) คือ $\overline{DSR}$(ขา 37)เมื่อเป็น "0" จะแสดงว่าโมเด็มหรือข้อมูลได้รับการเซตเตรียมพร้อมแล้วสำหรับการเชื่อมต่อกับสายสื่อสาร และส่งข้อมูลระหว่าง 8250 กับโมเด็มสัญญาณ DSRเป็นสัญญาณอินพุตของ 8250 ที่ซีพียูสามารถอ่านไปดูได้ทางบิตที่ 5 ของรีจิสเตอร์แสดงสถานะ ส่วนบิต 1 ของรีจิสเตอร์แสดงสถานะเป็นตัวบอกว่า สัญญาณ DSR ได้เปลี่ยนสถานะไปหลังจากที่อ่านครั้งก่อนแล้วหรือไม่

หมายเหตุ ทั้ง $\overline{CTS}$ และ $\overline{DSR}$ เมื่อมีการเปลี่ยนสถานะและถ้าได้รับการอีนาเบลที่ modem status interrupt จะส่งผลในการสร้างสัญญาณอินเตอร์รัพต์

ขาตรวจสอบสายสื่อสาร (received line signal detect) คือ $\overline{RLSD}$ (ขา 38) ถ้าเป็น "0" หมายถึงแอกทีฟ คือ 8250 รับสัญญาณตรวจสอบสัญญาณพาหะจากโมเด็มว่า โมเด็มตรวจสอบได้แล้ว ซีพียูสามารถตรวจสอบสัญญาณนี้ทางบิต 1 ของรีจิสเตอร์แสดงสถานะ ส่วนบิต 3 จะเป็นบิตที่แสดงสถานะว่าสัญญาณนี้ได้รับการเปลี่ยนแปลงหลังจากอ่านไปแล้วหรือยัง

ขาแสดงวงจรเรียก (ring indicator) คือ RI (ขา 39)สัญญาณนี้แอกทีฟด้วยลอจิก "0" เป็นสัญญาณที่ส่งมาจากโมเด็ม โมเด็มตรวจสอบสัญญาณการเรียก (ringing) สัญญาณนี้ตรวจสอบได้ทางบิต 6 และดูสถานะการเปลี่ยนหลังจากอ่านแล้วจากบิต 2

ขารีเควสต์ทูเซน (request to sent) คือ $\overline{RTS}$(ขา 32) เริ่มขานี้มีลอจิกเป็น "0" หมายความว่า 8250 พร้อมที่จะส่งข้อมูลแล้ว สัญญาณนี้จะได้รับการเซตให้แอกทีฟด้วยการโปรแกรมค่าลงไปในรีจิสเตอร์ควบคุมที่บิต 1

ขาเอาต์พุต 1 คือ $\overline{OUT1}$ (ขา 34) เป็นขาที่ผู้ใช้สามารถโปรแกรมให้แอกทีฟเป็น "0" ด้วยการโปรแกรมลงไปในบิต 2 ของรีจิสเตอร์ควบคุมโมเด็ม

ขาเอาต์พุต 2 คือ $\overline{OUT2}$ (ขา 31) เป็นขาที่ผู้ใช้สามารถโปรแกรมให้แอกทีฟเป็น "0" ด้วยการโปรแกรมค่าลงในรีจิสเตอร์ควบคุมโมเด็มทางบิต 3

ขาเลือกชิปเอาต์ (chip select out) คือ CSOUT(ขา 24)เมื่อมีค่าเป็น "1" จะบอกว่าชิปนี้ได้รับการเลือกชิปโดยซีพียูทางขา CS_0,CS_1และ CS_2

ขาไดรด์เวอร์ดีสเอเบิล (driver disable) คือ DDIS (ขา 23) เป็นลอจิก

"0" เมื่อซีพียูกำลังอ่านข้อมูลจาก 8250 สัญญาณ DDIS เป็น "1" มีไว้สำหรับการดิสเอเบิลการรับส่งภายนอกในกรณีที่ใช้ 8250 กับซีพียูผ่านทางบิต D_0-D_7 เพื่อบอกเวลาว่าซีพียูและ 8250 ติดต่อกันอย่างไร

ขาสัญญาณกำหนดบอด คือ $\overline{\text{BAUDOUT}}$ (ขา 15) เป็นสัญญาณนาฬิกาที่มีความถี่เป็น 16 เท่าของสัญญาณนาฬิกา แล้วหารด้วยค่าที่โปรแกรมกำหนดในตัวหาร

ขาอินเตอร์รัพต์ คือ INTRPT(ขา 30) เป็น "1" เป็นการส่งสัญญาณอินเตอร์รัพต์ออกไปจาก 8250

ขาข้อมูลเอาต์พุต คือ SOUT (ขา 11)เป็นขาที่ใช้ส่งข้อมูลแนุกรมออกไปยังสายสื่อสาร

ขาสัญญาณอินพุต/เอาต์พุต ข้อมูล D_6-D_7 เป็นสัญญาณต่อเชื่อมกับบัสข้อมูลของระบบ ขาสัญญาณ $X'TAL_1$, $X'TAL_2$ คือขา 16,ขา 17 เป็นขาต่อกับคริสตอลเพื่อสร้างสัญญาณนาฬิกา

สถานะของ 8250 เมื่อเริ่มต้น

ก่อนการทำงานหรือโปรแกรมเราควรจะทราบว่าสถานะต่าง ๆ ของ 8250 เป็นอย่างไร โดยเฉพาะอย่างยิ่งเมื่อเริ่มเปิดเครื่อง จะมีสัญญาณรีเซตซึ่งเป็นสัญญาณเดียวกับซีพียูมาทำการรีเซต 8250 8250 ขณะนั้นจะมีสถานะเป็นอย่างไร เอาต์พุตของวงจรจะให้สัญญาณอะไร จะทราบได้จากตารางที่ 2.4

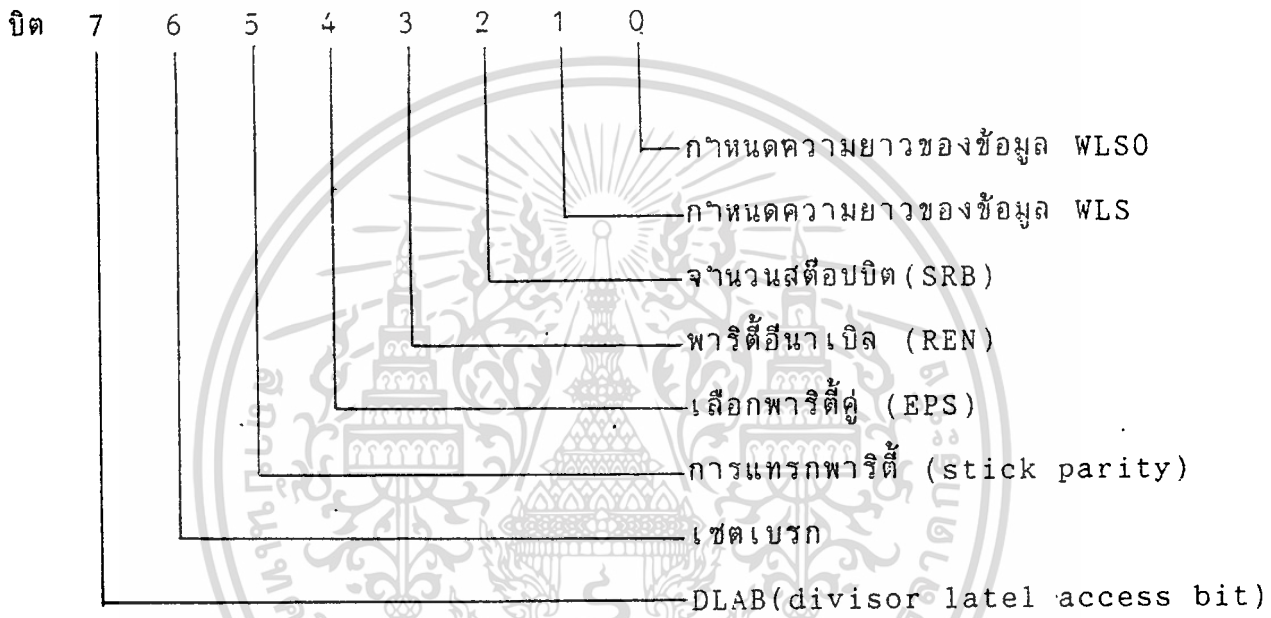
รีจิสเตอร์/สถานะ	การควบคุม	สถานะเมื่อรีเซต
อินเตอร์รัพต์รีจิสเตอร์ภายใน	มาสเตอร์รีเซต	ทุกบิตเป็น 0 (0-3 ถูกกำหนด 4-7 จะเป็นถาวร)
รีจิสเตอร์กำหนดอินเตอร์รัพต์	มาสเตอร์รีเซต	บิต 0 เป็น 1 บิต 1-2 เป็น 0 บิต 3-7 เป็น 0 ถาวร
รีจิสเตอร์ควบคุมสายสื่อสาร	มาสเตอร์รีเซต	ทุกบิตเป็น 0
รีจิสเตอร์ควบคุมโมเด็ม	มาสเตอร์รีเซต	ทุกบิตเป็น 0
รีจิสเตอร์แสดงสถานะสายสื่อสาร	มาสเตอร์รีเซต	ยกเว้นบิต 5 และ 6 เป็น 1
รีจิสเตอร์แสดงสถานะโมเด็ม	มาสเตอร์รีเซต	บิต 0-3 เป็น 0 บิต 4-7 เป็นสัญญาณอินพุต
SOUT	มาสเตอร์รีเซต	เป็น 1
INTRPT(RCVR ERRORS)	Read LSR/ มาสเตอร์รีเซต	เป็น 0
TNTRPT(RCVR Data Ready)	Read RBR/ มาสเตอร์รีเซต	เป็น 0
INTRPT(RCVR Data Ready)	Read IIR/ มาสเตอร์รีเซต	เป็น 0
INTRPT (เปลี่ยนสถานะโมเด็ม)	Read MSR/ มาสเตอร์รีเซต	เป็น 0
OUT2	มาสเตอร์รีเซต	เป็น 1
RTS	มาสเตอร์รีเซต	เป็น 1
DTR	มาสเตอร์รีเซต	เป็น 1
OUT1	มาสเตอร์รีเซต	เป็น 1

ตารางที่ 2.4 ค่าเริ่มต้นและเอาต์พุตของ 8250

การใช้งานรีจิสเตอร์ต่าง ๆ บน 8250

รีจิสเตอร์ควบคุมสายสื่อสาร (line control register) ในการควบคุมรูปแบบของข้อมูลแบบอะซิงโครนัสนั้น ผู้โปรแกรมจะต้องกำหนดค่าลงในรีจิสเตอร์ควบคุมสายสื่อสาร รีจิสเตอร์ตัวนี้มี 8 บิต โดยแต่ละบิตมีความหมายดังนี้

พอร์ตแอดเดรสหมายเลข 3FB



รูปที่ 2.12 ค่าของบิตในรีจิสเตอร์ควบคุมสายการสื่อสาร

บิต 0 และ 1 เป็นตัวกำหนดความยาวของข้อมูลในการรับส่ง โดยที่

บิต 0	บิต 1	ความหมาย
0	0	หมายถึงข้อมูลขนาด 5 บิต
0	1	หมายถึงข้อมูลขนาด 6บิต
1	0	หมายถึงข้อมูลขนาด 7 บิต

บิต 0	บิต 1	ความหมาย
1	1	หมายถึงข้อมูลขนาด 8 บิต

บิต 2 เป็นบิตที่ใช้ในการกำหนดจำนวนสตือบบิต ถ้าเป็น "0" หมายถึงใช้สตือบบิต 1 บิต แต่ถ้าบิต 2 เป็น "1"ในกรณีส่งแบบ 5 บิตจะมีความยาวของสตือบบิตเป็น 1.5 บิต แต่ถ้าส่งแบบ 6,7 หรือ 8 บิต ความยาวของสตือบบิตจะเป็น 2

บิต 3 บิตนี้เป็นบิตแสดงการอีนาเบิลให้มีการตรวจสอบพาริตี้ โดยถ้าบิตนี้มีค่าเป็น 1 จะมีการเพิ่มพาริตี้บิต

บิต 4 มีค่าเป็น "0" และบิต 3 มีค่าเป็น"1" จะมีการกำหนดเป็นพาริตี้คี่ แต่ถ้าบิตนี้มีค่าเป็น "1" จะเป็นพาริตี้คู่

บิต 5 เมื่อบิต 3 มีค่าเป็น "1" และบิต 4 มีค่าเป็น "1" และบิต 5 มีค่าเป็น "1" จะมีการแทรกหรือตรวจสอบพาริตี้ (stick parity) ด้วยเงื่อนไขกำหนดให้เป็น "0" และถ้าบิต 4 มีค่าเป็น "0" บิต 3 มีค่าเป็น "1" และ บิต 5 มีค่าเป็น "1" จะมีการกำหนดบิตพาริตี้เป็น "1"

บิต 6 เป็นบิตที่ควบคุมการเบรก เมื่อบิต 6 มีค่าเป็น "1" ส่วนของ SOUT จะได้รับการกำหนดให้เป็น "0" ตลอด

บิต 7 บิตนี้ทำหน้าที่เป็น DLAB บิตที่จะมีผลต่อการแลตช์ตัวหารดังที่กล่างตารางที่ 2.1และ ตารางที่ 2.2

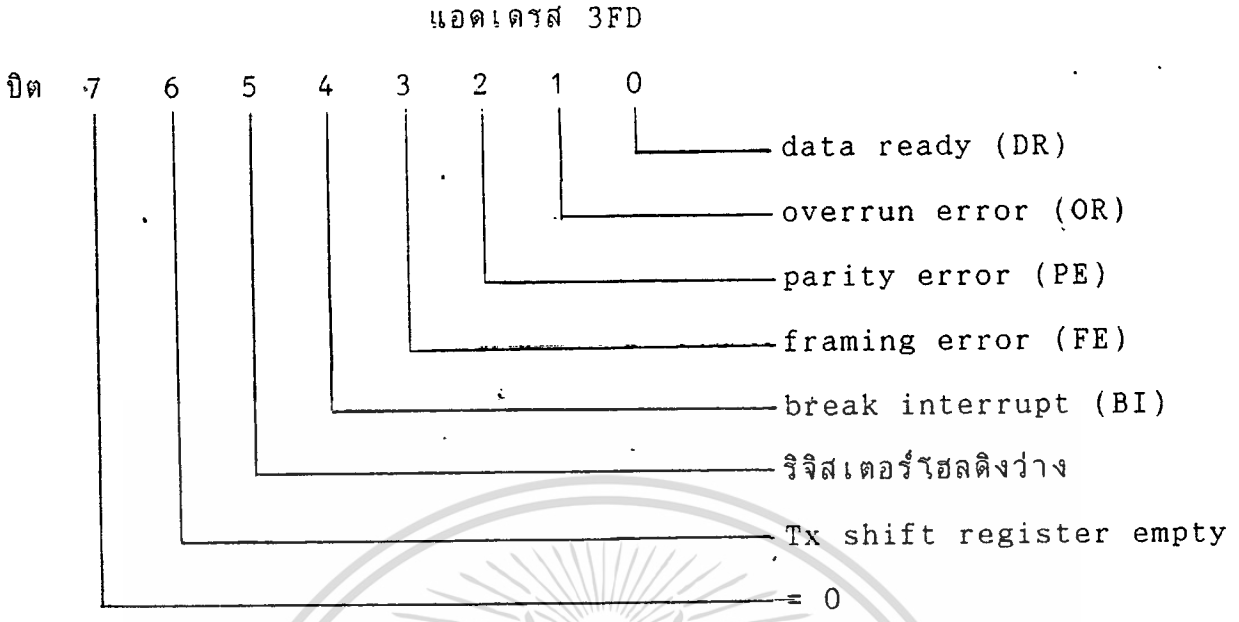
การโปรแกรมอัตราบอด (boud rate generator) อัตราบอดได้รับการกำหนดเทียบกับสัญญาณนาฬิกา 1.8432 MHz และสามารถโปรแกรมตัวหารได้ตั้งแต่ 1-(2^{16}-1) ค่าความถี่เอาต์พุตของตัวกำหนดอัตราบอดมีค่าเท่ากับ 16*อัตราบอด ดังนั้น
ตัวหาร = ความถี่สัญญาณนาฬิกา/(อัตรบอด*16) การกำหนดอัตราบอดด้วยการกำหนดตัวหารนี้ ตัวหารจึงเป็นค่าที่กำหนดในรีจิสเตอร์ 2 ตัว ตัวหารนี้จะต้องถูกกำหนดค่าก่อนแล้วโปรแกรมลงมาในรีจิสเตอร์นี้ การกำหนดต้องให้ DLAB = 1 แล้วให้ลดลงมาในรีจิสเตอร์ $3F_8$ ซึ่งเรียงกันเป็น LSB ของตัวหารส่วน $3F_9$ เมื่อ DLAB = 1 จะเป็นค่าของตัวหาร MSB ค่าของตัวหารเมื่อเทียบกับสัญญาณ 1.8432 MHz เป็นดังตารางที่ 2.5

อัตราบอด	ตัวหาร		ค่าผิดพลาด
	ฐานสิบ	ฐานสิบหก	
50	2304	900	-
75	1536	600	-
110	1047	417	0.026
134.5	857	359	0.058
150	768	300	-
300	384	180	-
600	192	0C0	-
1200	96	060	-
1800	64	040	-
2000	58	03A	0.69
2400	48	030	-
3600	32	020	-
4800	24	018	-
7200	16	010	-
9600	12	00C	-

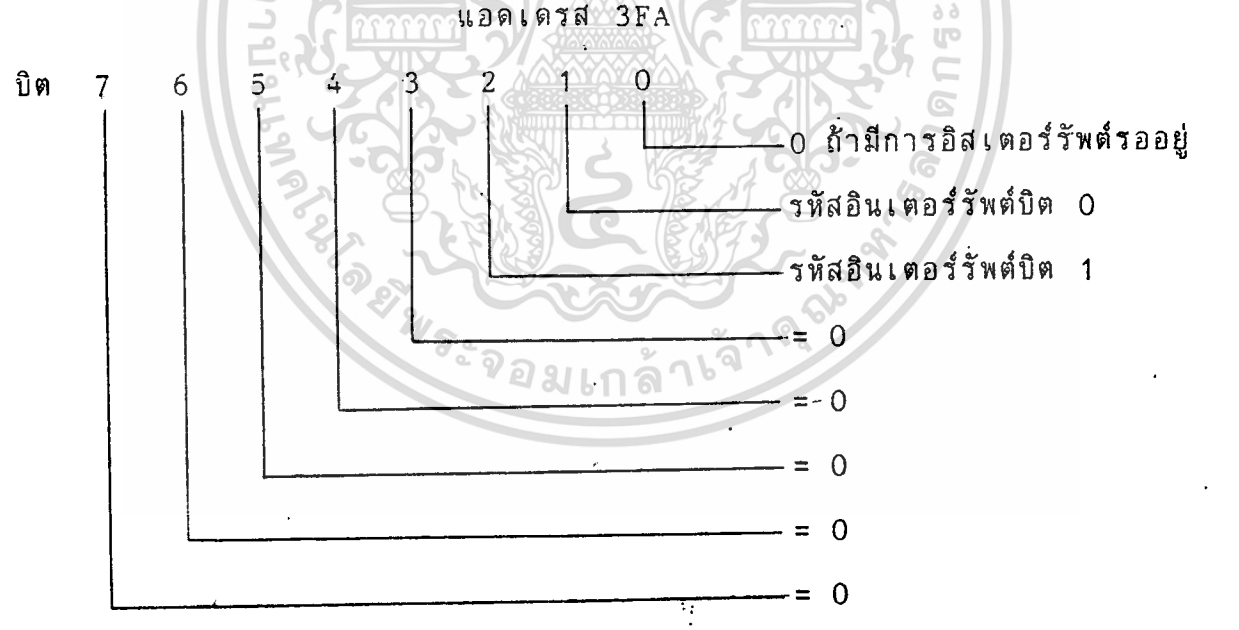
ตารางที่ 2.5 ค่าตัวหารสำหรับการกำหนดอัตราบอด

รีจิสเตอร์แสดงสถานะสายสื่อสาร (line status register) รีจิสเตอร์ตัวนี้เป็นรีจิสเตอร์ที่จะให้ข้อมูลแก่ซีพียูที่เกี่ยวกับการสื่อสาร ข้อมูลในการสื่อสาร ค่าของบิตต่าง ๆ ในรีจิสเตอร์นี้เป็นดังรูปที่ 2.13

บิต 0 บิตนี้เป็นบิตที่บอกสถานะการรับข้อมูล ถ้าบิตนี้เป็น "1" แสดงว่าการรับข้อมูลเข้ามาในบัฟเฟอร์ได้ครบทุกบิตแล้ว บิตนี้จะได้รับการรีเซตให้เป็น "0" เมื่อซีพียูได้อ่านข้อมูลในบัฟเฟอร์ไปแล้ว หรือจะให้ซีพียูเขียนข้อมูลกลับมายังรีจิสเตอร์นี้ก็ได้



รูปที่2.13ค่าของบิตในรีจิสเตอร์แสดงสถานะสายสื่อสาร



รูปที่2.14ค่าของบิตในรีจิสเตอร์กำหนดอินเตอร์รัพต์

บิต 1 บิตนี้ถ้ามีค่าเป็น "1" แสดงว่าเกิด overrun error (OR) กล่าวคือขณะ

ที่มีข้อมูลที่บัฟเฟอร์แต่ซีพียูยังไม่ได้อ่านไป ปรากฏว่ามีข้อมูลชุดใหม่มาเขียนทับ

บิต 2 บิตนี้ถ้ามีค่าเป็น "1" แสดงว่าเกิน parity error (PE) กล่าวคือถ้ามีการตรวจสอบบิตพาริตี้แล้วไม่เป็นไปตามที่กำหนดไว้ บิตนี้จะได้รับการรีเซตโดยซีพียู เมื่อซีพียูอ่านค่าจากรีจิสเตอร์นี้ไปแล้ว

บิต 3 บิตนี้ถ้ามีเฟรมของข้อมูลไม่เป็นไปตามที่กำหนด เช่น ตรวจสอบจำนวนบิตโดยดูที่พาริตี้และสตีอปบิตไม่เป็นไปตามที่กำหนด

บิต 4 บิตนี้เรียกว่า break interrupt (BI) บิตนี้จะได้รับการเซตให้มีค่าเป็น "1" ถ้าหากว่ารับข้อมูลอินพุตเป็น "0" เป็นเวลายาวนานกว่าเวิร์ดของการสื่อสาร

บิต 5 บิตนี้เป็นบิตที่บอกว่า 8250 พร้อมที่จะรับข้อมูลจากสายสื่อสาร บิตนี้จะได้รับการเซตให้มีค่าเป็น "1" บิตนี้ยังคงสร้างสัญญาณอินเตอร์รัพต์เพื่อส่งไปบอกซีพียูด้วย นอกจากนี้จะมีสถานะเซตเมื่อมีการส่งถ่ายข้อมูลจากโฮลด์รีจิสเตอร์ไปยังชิปรีจิสเตอร์เพื่อพร้อมที่จะส่ง

บิต 6 เป็นบิตที่จะบอกว่าชิปรีจิสเตอร์ว่างเปล่า บิตนี้จะได้รับการเซตให้มีค่าเป็น "1" เพื่อบอกว่าพร้อมส่งแล้ว

บิต 7 จะเป็น "0" ตลอด

รีจิสเตอร์กำหนดอินเตอร์รัพต์(IIR-interrupt identification register) ไอซี 8250 มีขีดความสามารถในการส่งอินเตอร์รัพต์ภายในชิป เพื่อว่าให้การทำงานระหว่าง 8250 กับซีพียูเป็นไปอย่างมีประสิทธิภาพสูง และเพื่อให้ผู้เขียนซอฟต์แวร์สามารถเขียนซอฟต์แวร์ได้ง่ายและสั้นลงได้มาก 8250 กำหนดความสำคัญของอินเตอร์รัพต์ไว้ 4 ระดับคือ ระดับแรก สถานะการรับข้อมูลจากสายสื่อสาร ระดับที่สอง การพร้อมรับข้อมูล ระดับที่สาม ขณะรีจิสเตอร์โฮลดิงสำหรับส่งข่าว ระดับที่สี่ สัญญาณสถานะโมเด็ม

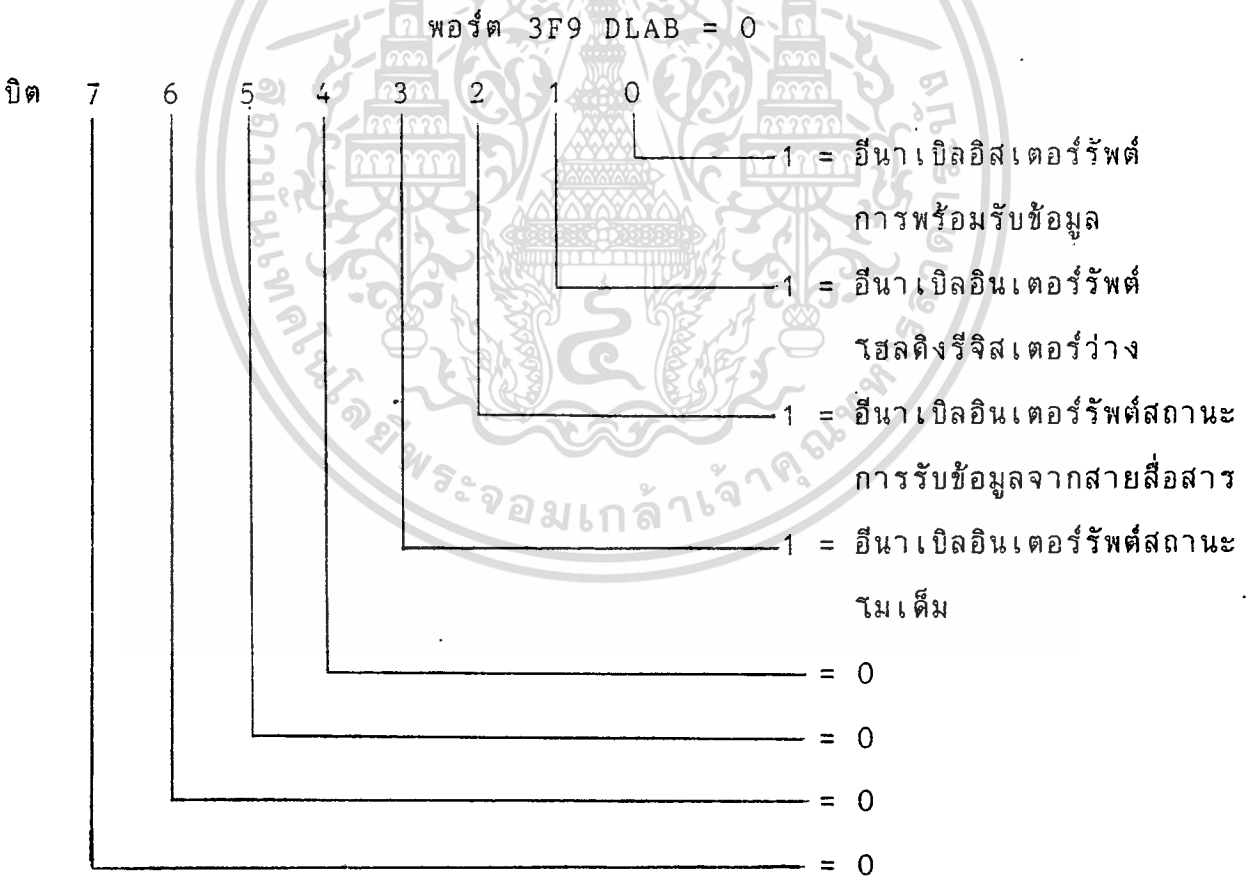
ในขณะที่มีความต้องการอินเตอร์รัพต์หลายระดับพร้อมกัน 8250 จะให้ระดับที่มีความสำคัญน้อยกว่ารอไว้ก่อน โดยเก็บสถานะการอินเตอร์รัพต์นี้ไว้ในรีจิสเตอร์กำหนดอินเตอร์รัพต์ความหมายของรีจิสเตอร์นี้มีดังนี้

บิต 0 เป็นบิตที่ใช้แสดงว่ามีอินเตอร์รัพต์เกิดขึ้นหรือไม่ ซึ่งสามารถให้ซีพียูตรวจสอบดูด้วยวิธีการ polling ได้ถ้าบิตนี้เป็น "1" หมายถึง ไม่มีอินเตอร์รัพต์เกิดขึ้น

บิต 1-2 เป็นบิตที่แสดงความหมายบอกว่าการอินเตร์รัพต์ที่เกิดขึ้นนั้นมาจากการอินเตอร์รัพต์ตามฟังก์ชันใด

บิต 3-7 มีค่าเป็น "0"

รีจิสเตอร์อีนาเบิลอินเตอร์รัพต์ (INTRPT-interrupt enable register)ใน COM_1เมื่อให้ DLAB = 0 พอร์ต $3F_9$ จะเป็นรีจิสเตอร์อีนาเบิลอินเตอร์รัพต์ ผู้ใช้สามารถกำหนดให้เกิดอินเตอร์รัพต์หรือไม่ก็ได้ โดยการกำหนดค่าลงในรีจิสเตอร์นี้ จากที่กล่าวแล้วว่าการอินเตอร์รัพต์ของ 8250 นี้มี 4 แบบ ดังนั้นจึงต้องกำหนดการอีนาเบิลได้ทั้ง 4 แบบ โดยการใช้ข้อมูลแต่ละบิตของรีจิสเตอร์นี้ เพื่อการกำหนดการอีนาเบิล ข้อมูลที่อยู่ในรีจิสเตอร์นี้มีความหมายดังนี้



รูปที่2.15ค่าของบิตในรีจิสเตอร์พีนาเบิลอินเตอร์รัพต์

บิต2 1 0	ระดับความสำคัญ	ชนิดของอินเตอร์รัพต์	แหล่งเกิดอินเตอร์รัพต์	การรีเซตควบคุมอินเตอร์รัพต์
0 0 1 1 1 0	สูงสุด	ไม่เกิด สถานะการรับข้อมูลจากสายสื่อสาร	ไม่เกิด overrun error parity error framming error break interrupt	อ่านข้อมูลจากรีจิสเตอร์สถานะสายสื่อสาร
1 0 0	ที่สอง	การพร้อมรับข้อมูล	มีข้อมูลที่ตัวรับ	การอ่านข้อมูลจากบัฟเฟอร์
0 1 0	ที่สาม	โฮลดิงรีจิสเตอร์สำหรับส่งข่าว	โฮลดิงรีจิสเตอร์สำหรับส่งข่าว	อ่านรีจิสเตอร์กำหนดอินเตอร์รัพต์ IIR หรือเขียนลงไปยังโฮลดิงรีจิสเตอร์สำหรับส่ง
0 0 0	ที่สี่	สถานะโมเด็ม	CIS DSR RI ตรวจสอบสายส่งโดยตรง	อ่านรีจิสเตอร์แสดงสถานะของโมเด็ม

ตารางที่ 2.6 ฟังก์ชันการอินเตอร์รัพต์รหัสอินเตอร์รัพต์

บิต 0 บิตนี้ได้รับการเซตเป็น "1" เมื่อต้องการอีนาเบิลอินเตอร์รัพต์การพร้อมรับข้อมูล

บิต 1 บิตนี้ได้รับการเซตเป็น "1" เมื่อต้องการอีนาเบิลอินเตอร์รัพต์โฮลดิงรีจิสเตอร์ว่าง

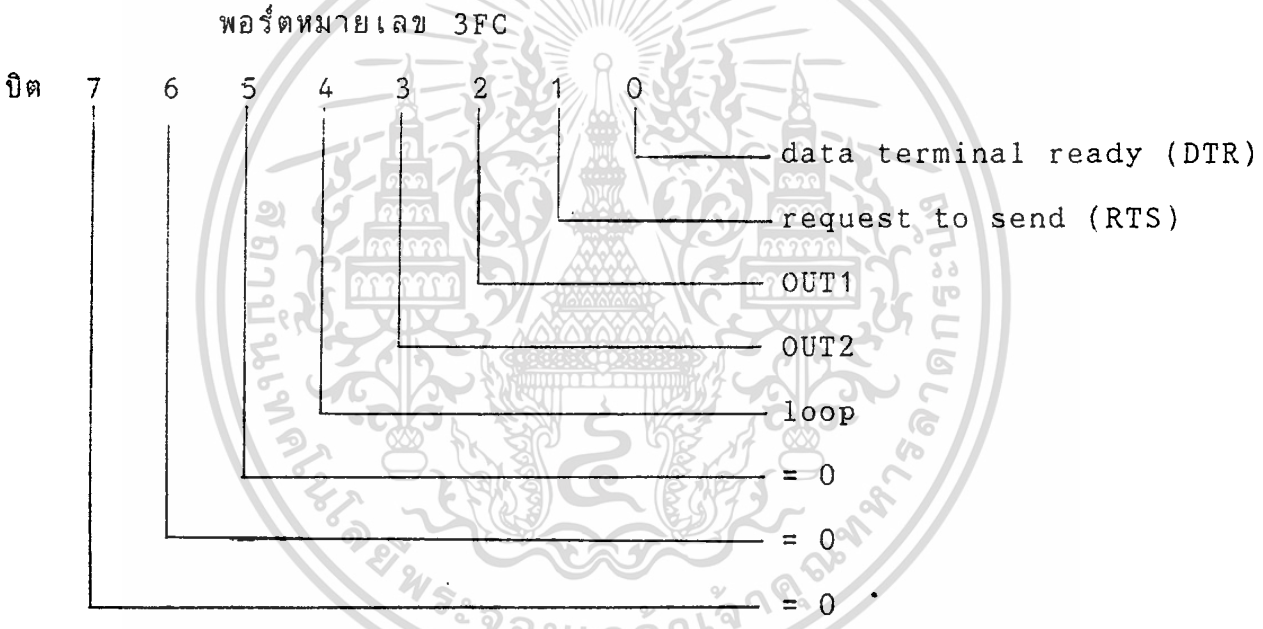
บิต 2 บิตนี้ได้รับการเซตเป็น "1" เมื่อต้องการอีนาเบิลอินเตอร์รัพต์จากสถานะ

การรับข้อมูลจากสายสื่อสาร

บิต 3 บิตนี้ได้รับการเซตเป็น "1" เมื่อต้องการอีนาเบิลอินเตอร์รัพต์จากสถานะโมเด็ม

บิต 4-7 ได้รับการกำหนดให้เป็น 0 เสมอ

รีจิสเตอร์ควบคุมโมเด็ม (modem control register) รีจิสเตอร์ตัวนี้มีไว้ให้ซีพียูส่งผ่านข้อมูลมาเก็บเพื่อเป็นรหัสสำหรับควบคุมการทำงานของโมเด็ม การกำหนดพอร์ตของรีจิสเตอร์ตัวนี้คือ 3FC ข้อมูลต่าง ๆ ที่มีในรีจิสเตอร์ตัวนี้มีความหมายดังนี้



รูปที่2.16ค่าของบิตในรีจิสเตอร์ควบคุมโมเด็ม

บิต 0 บิตนี้มีความหมายถึงการควบคุมสัญญาณ $\overline{\text{DTR}}$ เมื่อบิตนี้มีค่าเป็น "1" เอาต์พุตที่ $\overline{\text{DTR}}$ จะได้รับการกำหนดให้เป็น "0" และถ้าบิตนี้มีค่าเป็น "0" เอาต์พุตที่ $\overline{\text{DTR}}$ จะได้รับการกำหนดให้เป็น "1"

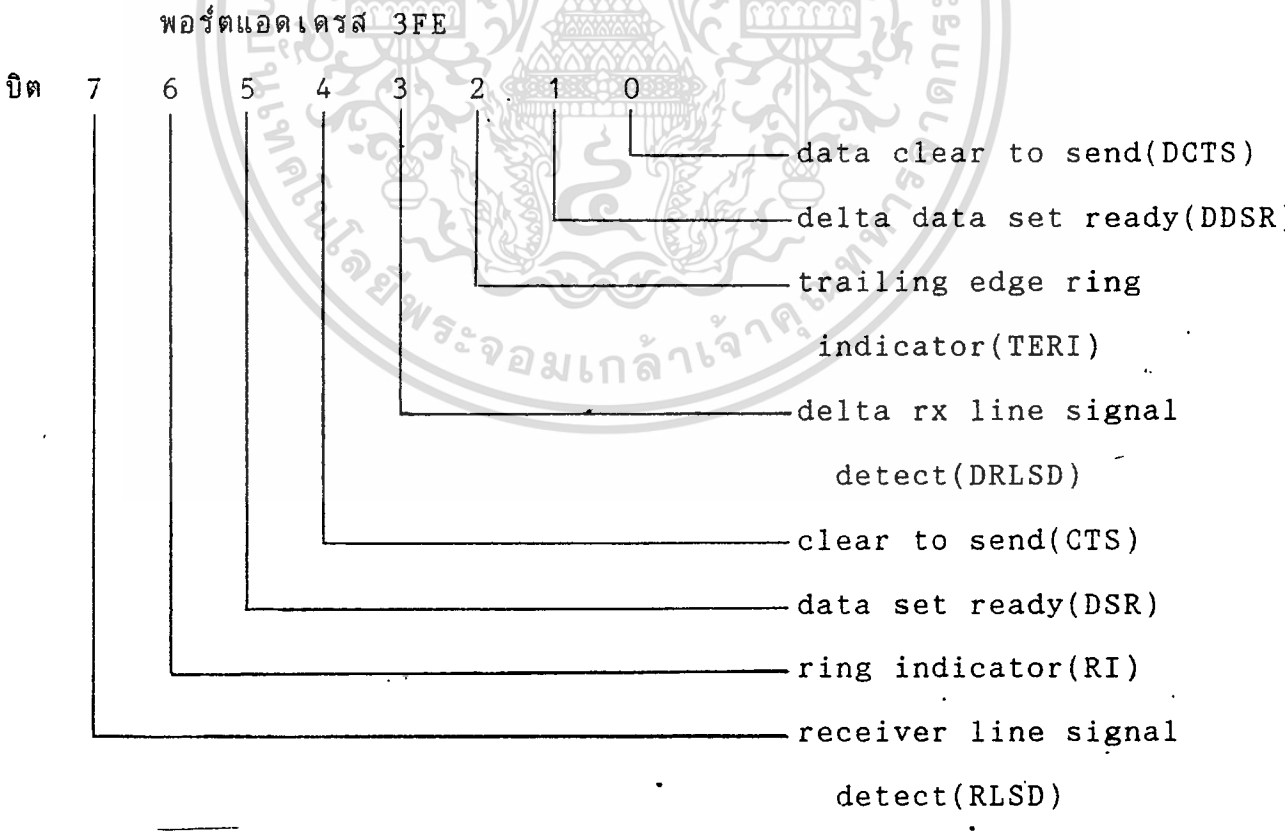
บิต 1 บิตนี้มีความหมายถึงสัญญาณ RTS ซึ่งจะมีผลเหมือนกับบิต 0 ในกรณีของ DTR

บิต 2 บิตนี้ใช้ควบคุมขาเอาต์พุต 1(OUT1)ซึ่งจะมีผลเหมือนบิต 0

บิต 3 บิตนี้ใช้ควบคุมขาเอาต์พุต 2(OUT2) ซึ่งจะมีผลเหมือนบิต 0

บิต 4 บิตนี้จะใช้สำหรับการกำหนดวงรอบสำหรับการตรวจสอบ 8250 เมื่อบิต 4 นี้ได้รับการเซตเป็น "1" สิ่งที่จะเกิดขึ้นเป็นดังนี้ ข้อมูลที่ SOUT จะได้รับการเซตให้เป็นลอจิก "1" ขาข้อมูลอินพุต SIN จะได้รับการแยกตัวออก ข้อมูลของเอาต์พุตชิบรีจิสเตอร์จะได้รับการป้อนกลับมายังรีจิสเตอร์ข้อมูลอินพุต ส่วนสัญญาณ CTS,DSR,RLSD และ RI จะได้รับการแยกออกจากระบบแต่สัญญาณควบคุมโมเด็มคือ DTR,RTS,OUT1 และ OUT2 จะต่อเข้ากับสัญญาณทั้งสี่ที่เป็นอินพุต ดังนั้นจึงตรวจสอบระบบการทำงานได้

รีจิสเตอร์แสดงสถานะโมเด็ม รีจิสเตอร์ตัวนี้จะเป็นตัวที่รับสถานะจากโมเด็มมาเก็บไว้เพื่อให้ซีพียูสามารถอ่านตรวจสอบดูได้ สถานะของข้อมูลจะแอกทีฟเมื่อมีข้อมูลเป็น "1" และจะได้รับการรีเซตเมื่อซีพียูอ่านข้อมูลในรีจิสเตอร์นี้ไป พอร์ตที่ใช้กำหนดเป็นพอร์ตหมายเลข 3FE ข้อมูลภายในรีจิสเตอร์เป็นดังนี้



รูปที่2.๙ ค่าของบิตในรีจิสเตอร์แสดงสถานะโมเด็ม

บิต 0 บิตนี้ใช้สำหรับแสดงการเปลี่ยนแปลงของสัญญาณ CTS กล่าวคือเมื่อขา CTS ของ 8250 ได้เปลี่ยนสถานะหลังจากที่ซีพียูได้อ่านสถานะนี้ไปแล้ว บิตนี้ก็จะบอกด้วยเซตและเมื่อซีพียูอ่านก็จะได้รับการรีเซต "0" และจะได้รับการเซต "1" เมื่อมีการเปลี่ยนสถานะที่ขา CTS

บิต 1 เหมือนบิต 0 แต่เป็นบิตที่แสดงสถานะการเปลี่ยนแปลงของขา DSR

บิต 2 บิตนี้เป็นบิตแสดงว่าสัญญาณ RI ซึ่งเป็นอินพุตของ 8250 ได้รับการเปลี่ยนจากออน "1" มาเป็นออฟ "0"

บิต 3 บิตนี้เหมือนบิต 0 แต่เป็นบิตแสดงสถานะการเปลี่ยนแปลงของ line signal detector ซึ่งเป็นขาอินพุต RLSD

บิต 4 บิตนี้เก็บสัญญาณคอมพลีเมนต์กับสัญญาณที่ขา CTS

บิต 5 บิตนี้เก็บสัญญาณคอมพลีเมนต์กับสัญญาณที่ขา DSR

บิต 6 บิตนี้เก็บสัญญาณคอมพลีเมนต์กับสัญญาณที่ขา RI

บิต 7 บิตนี้เก็บสัญญาณคอมพลีเมนต์กับสัญญาณที่ขา RLSD

อนึ่งถ้าบิต 4 ของ MCR ได้รับการเซตหรือให้ทำลูป (loop) ตรวจสอบข้อมูลในบิต 4 จะเหมือนกับ RTS ใน MCR ข้อมูลในบิต 5 จะเหมือนกับ DTR ใน MCR ข้อมูลในบิต 6 จะเหมือนกับ OUT1 ใน MCR ข้อมูลในบิต 7 จะเหมือนกับ OUT2 ใน MCR รีจิสเตอร์บัฟเฟอร์สำหรับตัวรับข้อมูล (receiver buffer register)เป็นรีจิสเตอร์สำหรับการรับข้อมูลที่มาจากสายสื่อสารสัญญาณ พอร์ตที่กำหนดคือ หมายเลขแอดเดรส 3F8 ขณะที่ DLAB = 0 หากซีพียูอ่านข้อมูลที่รีจิสเตอร์นี้ก็หมายถึงได้อ่านข้อมูลที่มาจากสายสัญญาณสื่อสารนั่นเอง

รีจิสเตอร์โฮลดิงสำหรับตัวส่งข้อมูล (transmitter holding register)เป็นรีจิสเตอร์บัฟเฟอร์สำหรับส่งข้อมูล รีจิสเตอร์นี้จะรับข้อมูลจากซีพียู โดยที่กำหนดพอร์ตเป็นหมายเลข 3F8 เมื่อ DLAB = 0 ข้อมูลของซีพียูที่เอาต์พุตมาที่พอร์ตนี้ก็เพื่อจะส่งต่อออกไปยังสายส่งข้อมูล

บทที่ 3

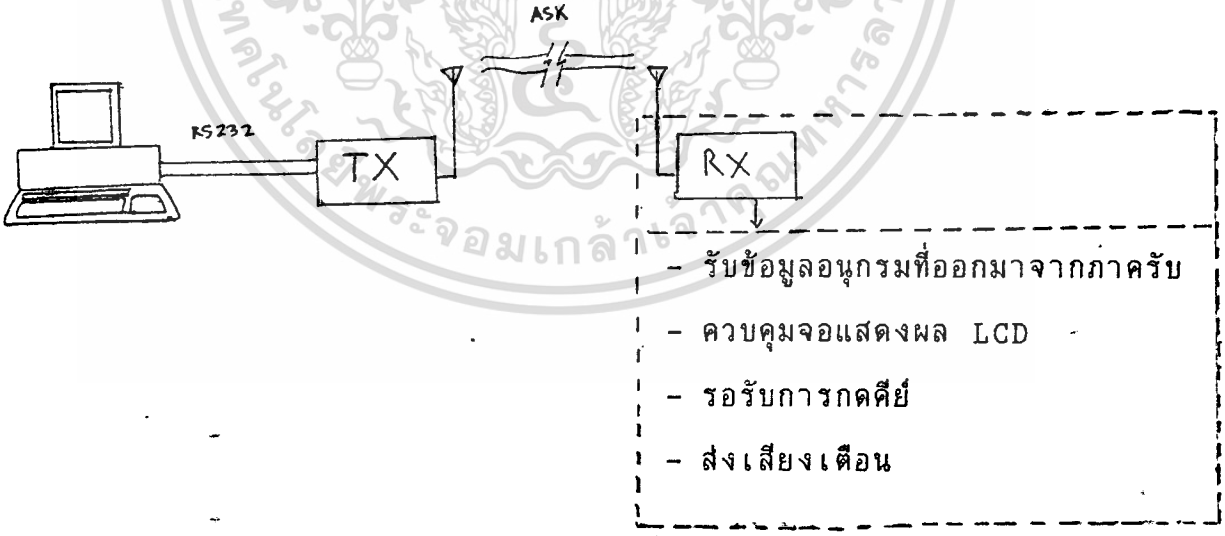
การคำนวณและการสร้าง

หลักการของระบบวิทยุติดตามตัว (Pager)

ในโครงงานนี้ระบบวิทยุติดตามตัว ประกอบด้วย 3 ส่วนหลักคือ

* ศูนย์รับข้อมูลและจัดเก็บข้อมูล โดยใช้เครื่องคอมพิวเตอร์
* เครื่องส่งข้อมูล
* เครื่องรับข้อมูล และแสดงผลข้อมูล

ซึ่งมีบล็อกไดอะแกรมแสดงโครงสร้างของระบบวิทยุติดตามตัวดังรูปนี้



รูปที่ 3.1 แสดงโครงสร้างของระบบวิทยุติดตามตัว

รายละเอียดหน้าที่ และการทำงานของส่วนต่าง ๆ ของโครงสร้างระบบวิทยุติดตามตัว มีดังนี้

3.1 ศูนย์รับข้อมูลและจัดเก็บข้อมูล

เครื่องคอมพิวเตอร์ที่ศูนย์รับข้อมูลและจัดเก็บข้อมูล จะทำหน้าที่ในการจัดการเกี่ยวกับการรอรับข้อมูลจากผู้เรียก การเก็บข้อมูล และการจัดส่งข้อมูล สำหรับข้อมูลที่จะส่งออกไปจากเครื่องคอมพิวเตอร์นั้น จะส่งออกทางพอร์ทอนุกรมตามมาตรฐานแบบ RS - 232 ดังที่ได้อธิบายมาแล้ว ซึ่งรูปแบบการจัดเรียงข้อมูลอนุกรมที่ส่งออกมาจากเครื่องคอมพิวเตอร์ ได้แสดงไว้ดังรูปที่ 3.2

รูปที่ 3.2 รูปแบบข้อมูลที่ส่งออกมาจากเครื่องคอมพิวเตอร์

a) รูปแบบของข้อมูลแต่ละไบท์ (Byte) ต่อ 1 อักขระ

b) รูปแบบของข้อมูล และจำนวนไบท์ (Byte) ต่อ 1 ชุดข้อมูล

ภายใน 1 ชุดข้อมูล จะประกอบด้วยจำนวนไบท์ ตั้งแต่ 4 - 51 ไบท์ โดยจะมีจำนวนอักขระ (Character) ที่จะส่งได้ไม่เกิน 48 อักขระ เนื่องจากภายในตัวไมโครคอนโทลเลอร์เบอร์ PIC 16C57 จะมีหน่วยความจำสำหรับเก็บข้อมูลได้ 48 ไบท์ โดยแต่ละอักขระจะเป็นรหัสแอสกี้ (ASCII) ที่ส่งออกมาจากพอร์ทอนุกรม RS - 232 ชุดข้อมูลจะถูกแบ่งออกเป็นส่วนต่าง ๆ เรียงลำดับดังต่อไปนี้

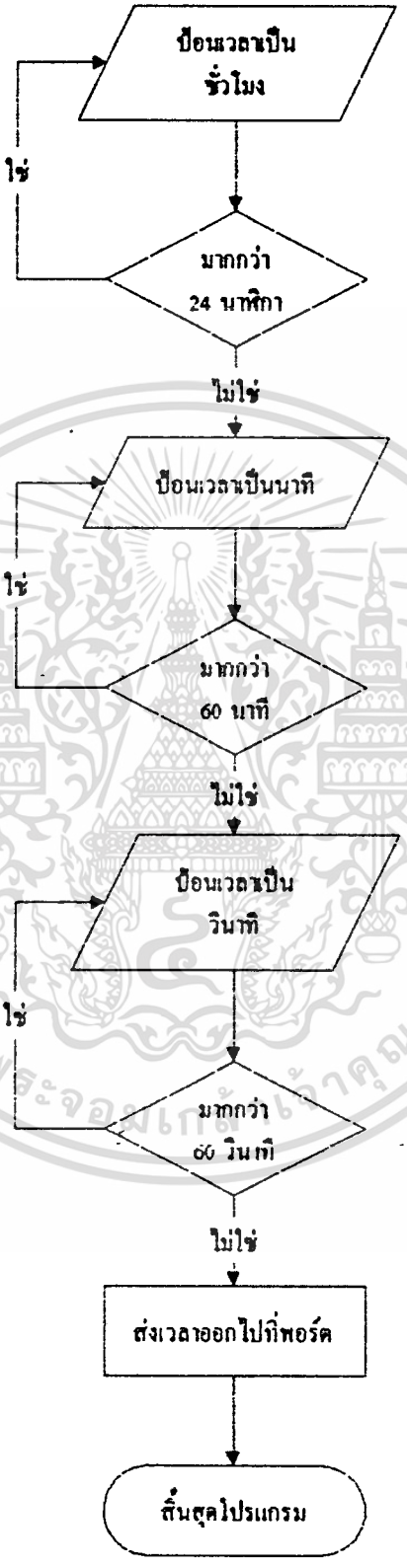
- **ไบท์เริ่มต้น** (Start Byte) จะเป็นไบท์แรกที่ส่งออกมาเพื่อให้เครื่องรับรู้ว่าเครื่องส่งได้เริ่มส่งชุดข้อมูลออกมาแล้ว ซึ่งจะต้องมีการกำหนดรูปแบบของไบท์ที่ตายตัว และจะต้องตรงกันระหว่างเครื่องส่งและเครื่องรับ

- **แอดเดรสไบท์** (Address Byte) เป็นตัวบอกแอดเดรสหรือหมายเลขของเครื่องรับวิทยุติดตามตัว ซึ่งเครื่องรับแต่ละเครื่องจะมีหมายเลยเครื่องที่ต่างกัน โดยจะมีการตั้งค่ากำหนดหมายเลขเครื่องของแต่ละเครื่องไว้ภายในส่วนโปรแกรมควบคุมของตัวไมโครคอนโทลเลอร์เบอร์ PIC 16C57 ที่เครื่องรับ เมื่อมีการส่งข้อมูลมาเครื่องรับจะทำการรับข้อมูลเฉพาะที่มีแอดเดรสไบท์ตรงกับเครื่องของตัวเองเท่านั้น

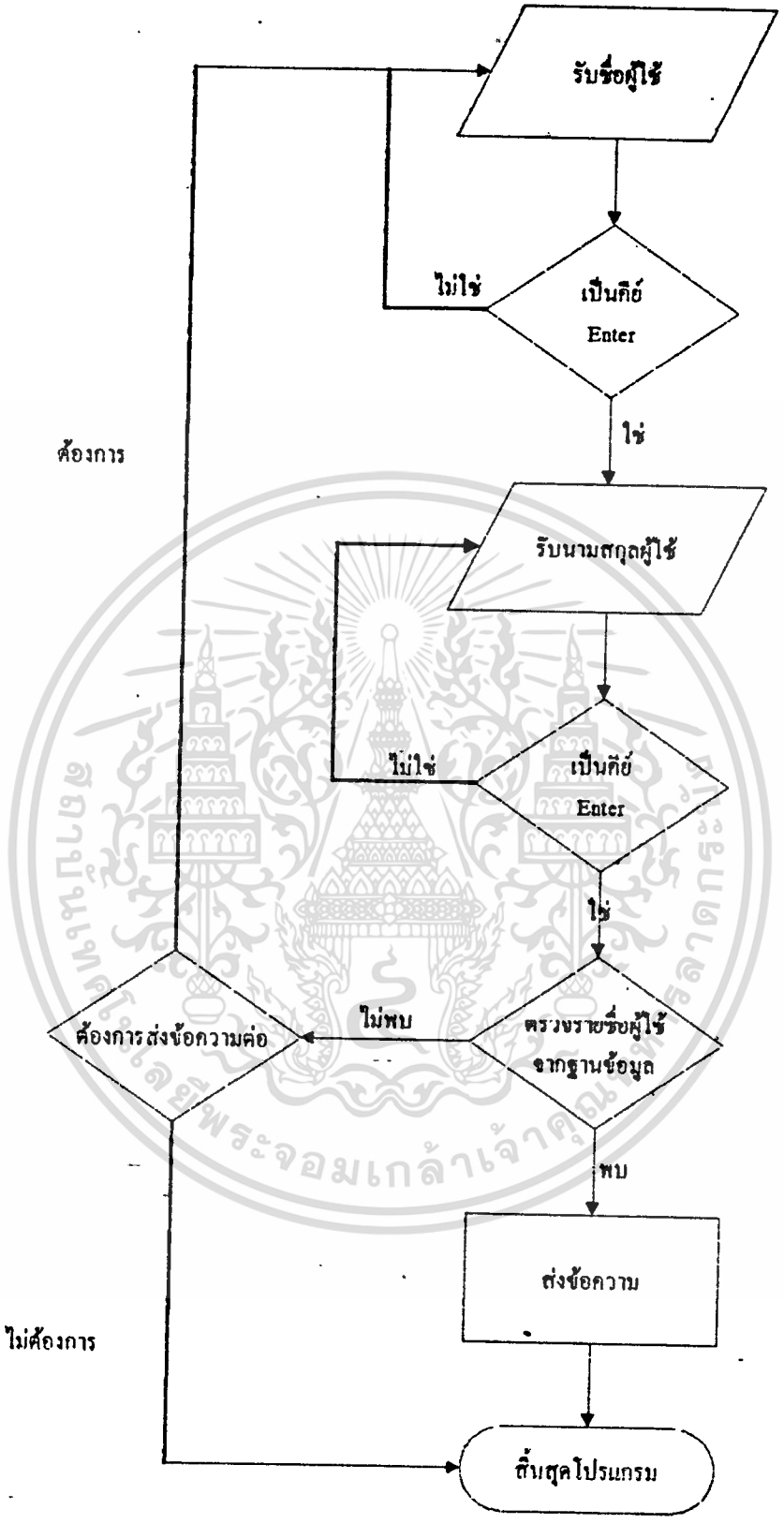
- **ข้อมูลข่าวสาร** (Information Byte) จะเป็นชุดอักขระรหัสแอสกี้ของข้อความที่ต้องการจะส่งจากผู้เรียกไปยังเครื่องรับวิทยุติดตามตัวของผู้รับ ซึ่งจะมีความยาวได้ตั้งแต่ 1 - 48 อักขระ

- **ไบท์สิ้นสุด** (Stop Byte) เป็นไบท์สุดท้ายของชุดข้อมูลที่ส่งมา เพื่อบอกให้เครื่องรับรู้ว่าข้อมูลที่จัดส่งมานั้นหมดแล้วเป็นการสิ้นสุดขบวนการรับส่งข้อมูล

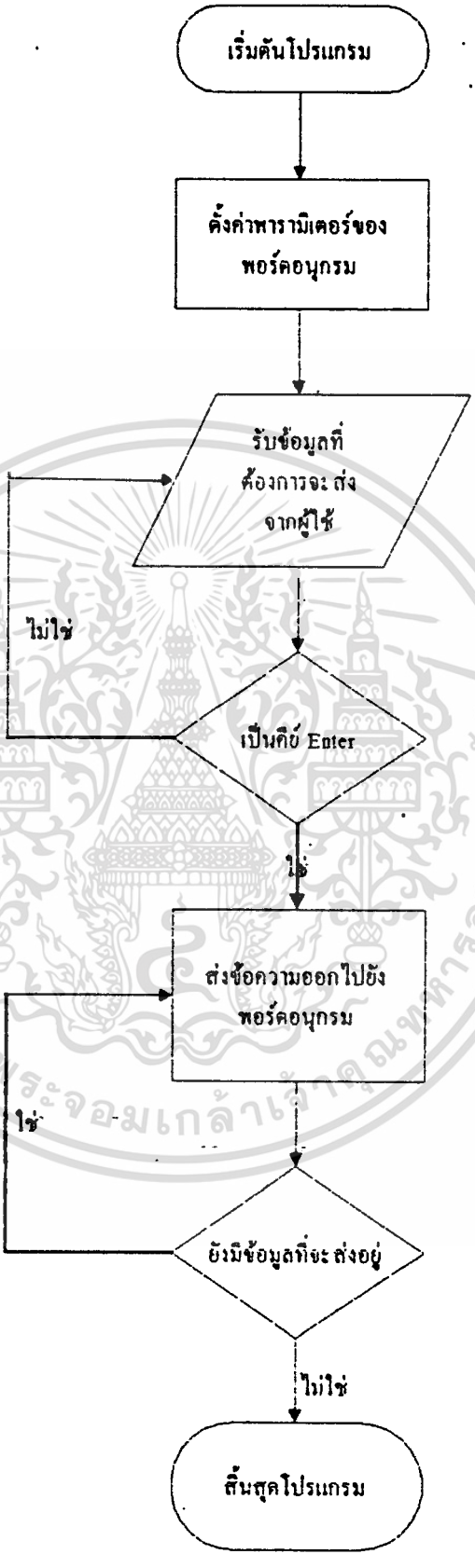
โดยชุดข้อมูลเหล่านี้จะถูกส่งต่อไปยังภาคส่งเพื่อทำการมอดดูเลทส่งออกอากาศต่อไป



รูปที่ 3.3 การตั้งเวลา



รูปที่ 3.4 การรับข้อความและส่งข้อมูล



รูปที่ 3.5 การส่งข้อมูล

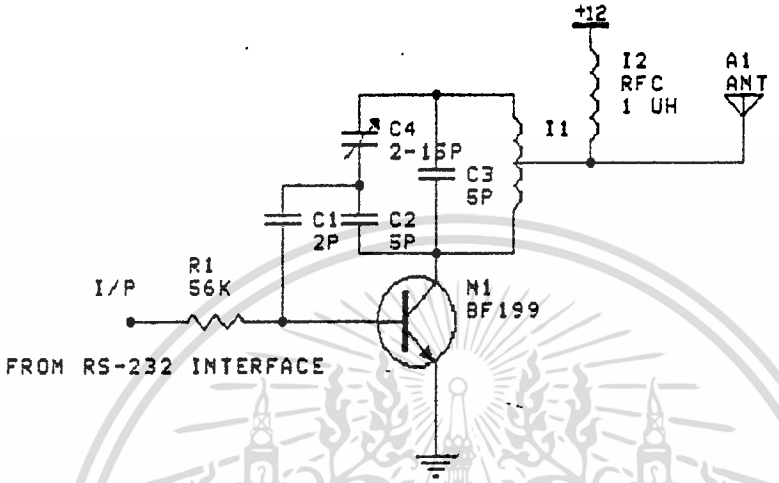
ในส่วนของโปรแกรมนี้เริ่มต้นที่ เมื่อมีข้อความที่ต้องการส่งเข้ามา จะสามารถส่งข้อความได้โดยใช้ชื่อของเจ้าของเพจเจอร์ หรือ หมายเลขเพจเจอร์ก็ได้ โดยสามารถเลือกได้จากเมนู เมื่อพบหมายเลขที่ต้องการแล้ว ในส่วนของโปรแกรมจะทำการส่งข้อความผ่านทางพอร์ตอนุกรม (RS-232) ไปยังเครื่องรับ โดยมีหลักการทำงานดัง Flow chart ที่แสดง

นอกจากนี้ในส่วนของเมนู ยังมีรายการอื่น ๆ ที่สามารถทำได้อีก คือ รายการรายชื่อผู้ใช้ทั้งหมด หมายเลขเพจเจอร์ทั้งหมด การตั้งเวลาโดยไม่ต้องกลับไปตั้งเวลาที่ส่วนของ Dos การเพิ่มรายชื่อผู้ใช้ การลบรายชื่อผู้ใช้ ซึ่งมี Flow chart แสดง การทำงานในส่วนของโปรแกรมตั้งเวลา ดังที่ได้แสดงไว้แล้ว

ในส่วนของโปรแกรมหลัก (main) นั้น ไม่มี Flow chart แสดงไว้ เนื่องจากเป็นส่วนของการสร้างเมนูต่าง ๆ ให้เลือกใช้

การทำงานของโปรแกรมที่ทำการส่งข้อความ เริ่มจากการตั้งค่าพารามิเตอร์ต่างๆ ของพอร์ตอนุกรม เช่น อัตราการส่งตัวอักขระ (boud rate) จำนวนสตอปบิต ชนิดของพาริตี้บิท จากนั้นจะทำการรับข้อความจากโอเปอร์เรเตอร์ (operater) โดยเริ่มด้วยหมายเลขเพจเจอร์ (อาจเป็นชื่อของผู้ใช้ก็ได้ โดยในส่วนโปรแกรมจะทำการค้นหมายเลขของผู้ใช้ให้) ตามด้วยข้อความที่ต้องการส่ง โดยโปรแกรมจะทำการตรวจสอบว่าคีย์ที่กดเป็นคีย Enter หรือไม่ ในทุกครั้งที่มีการกดคีย์ ถ้าไม่ใช่ก็จะรับข้อมูลต่อไป แต่ถ้าใช้ก็จะส่งข้อความออกไป จนกว่าจะหมดชุดข้อความ

3.2 ภาคส่ง



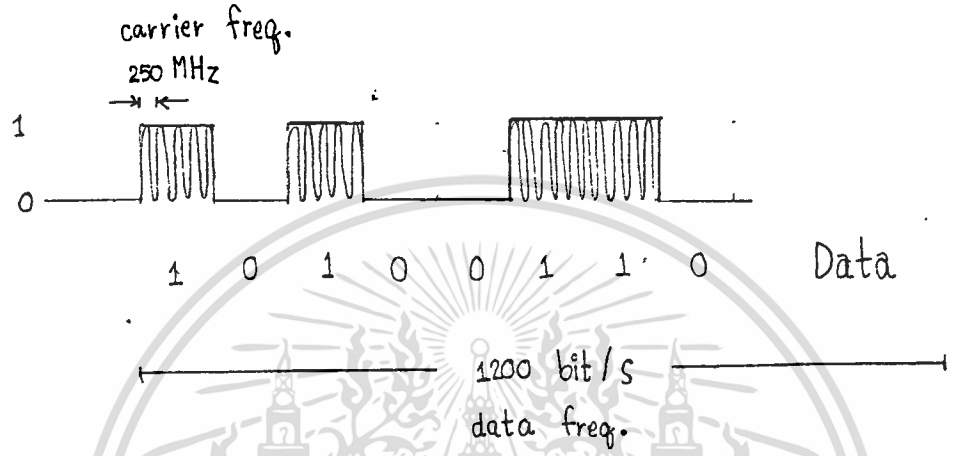
รูปที่ 3.6 วงจรเครื่องส่ง

จากรูปเป็นวงจรของเครื่องส่ง ข้อมูลที่ส่งออกมาจากเครื่องคอมพิวเตอร์จะถูกส่งเข้ามายังภาคส่งเพื่อทำการมอดดูเลทแบบ Amplitude shift - Frequency (ASK) ซึ่งก็คือการมอดดูเลทแบบ Amplitude Modulation (AM) นั่นเอง

- เมื่อข้อมูลที่มีค่าไบนารีบิทเป็น 1 เข้ามาที่อินพุทของวงจร ทำให้เกิดกระแสไบอัสทรานซิสเตอร์ BF 199 ทำให้วงจรเกิดการออสซิเลทส่งคลื่นความถี่วิทยุออกไปด้วยความถี่ประมาณ 250 MHz ซึ่งจำนวนข้อมูลที่ส่งออกมาจากพอร์ทอนุกรม RS - 232 ของเครื่องคอมพิวเตอร์จะมีความเร็ว 1200 bps (อัตราเร็วของการส่งบิทข้อมูล Bit rate ทำให้ภาครับสามารถตรวจจับรับความถี่นี้ได้เนื่องจากมีความถี่ที่ตรงกันระหว่างเครื่องส่งและเครื่องรับ

- เมื่อข้อมูลที่มีค่าไบนารีบิทเป็น 0 เข้ามาจะไม่ทำให้เกิดกระแสไบอัสทรานซิส-เตอร์ BF 199 วงจรจึงไม่เกิดการออสซิลเลทความถี่ออกมา

ดังนั้นรูปสัญญาณที่ถูกส่งออกทางสายอากาศจะเป็นดังรูปข้างล่าง



รูป 3.7 แสดงข้อมูลอนุกรม 1 ไบท์ที่มอดดูเลทแล้ว

หมายเหตุ

ข้อมูลที่ออกมาทางพอร์ทอนุกรมของเครื่องคอมพิวเตอร์นั้นต้องนำไปผ่านวงจรอินเตอร์เฟส RS-232 เพื่อทำการกลับเฟส 180 เนื่องจากระดับลอจิกของ RS-232 จะให้ข้อมูลไบนารี " 1 " อยู่ในช่วง -5 ถึง -15 โวลท์ และข้อมูลไบนารี " 0 " อยู่ในช่วง +5 ถึง +15 โวลท์

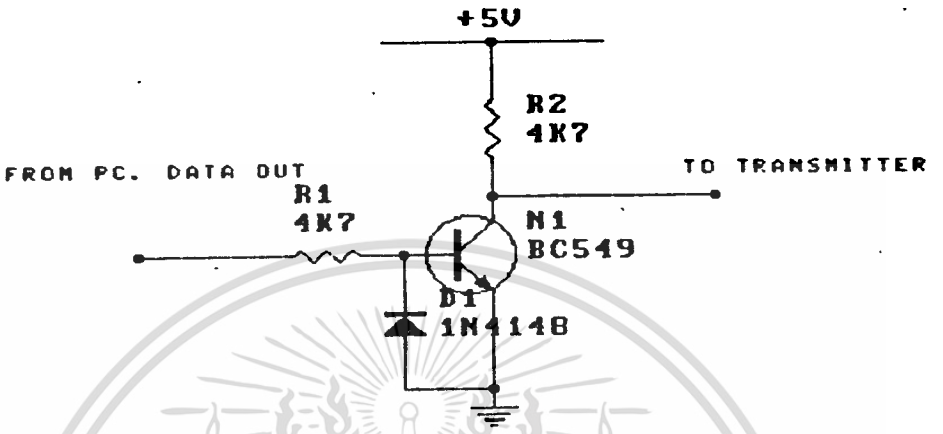
วงจรแปลงแรงดันอย่างง่าย ๆ คือ ใช้ทรานซิสเตอร์เป็นตัวแปลงสัญญาณแรงดันจาก $^{+}12$ โวลท์เป็น +5 โวลท์ และ 0 โวลท์ ดังนั้นค่าไบนารีจะมีการเปลี่ยนไปดังนี้

- ถ้าค่าไบนารีที่ออกจากพอร์ทอนุกรม RS - 232 มีค่าเป็นโลจิก " 1 " เมื่อผ่านวงจร RS - 232 Interface จะถูกเปลี่ยนค่าแรงดันเป็น +5 โวลท์

- ถ้าค่าไบนารีที่ออกจากพอร์ทอนุกรม RS - 232 มีค่าเป็นโลจิก " 0 " เมื่อ

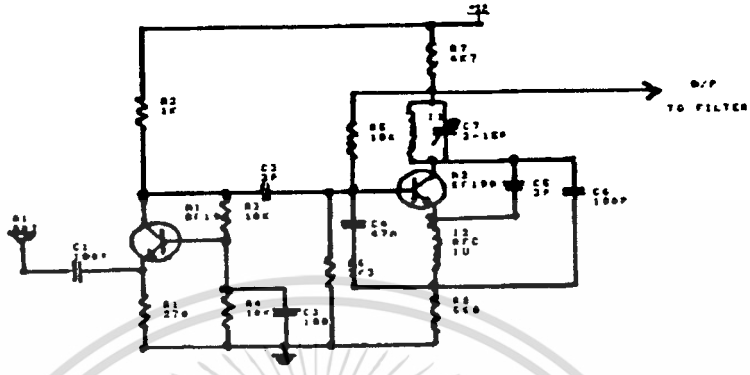
ผ่านวงจรอินเตอร์เฟส RS - 232 จะถูกเปลี่ยนค่าแรงดันเป็น 0 โวลท์

รูปวงจรอินเตอร์เฟส RS - 232 แสดงดังรูปข้างล่าง

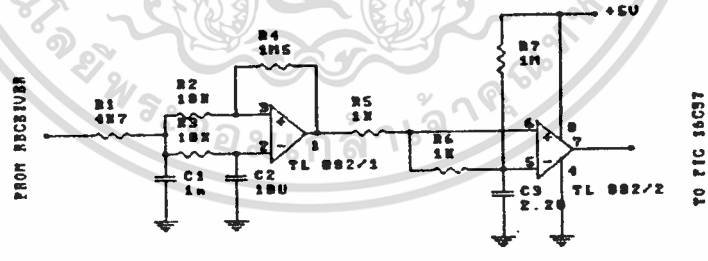


รูปที่ 3.8 แสดงวงจร อินเตอร์เฟส RS - 232

3.3 ภาครับ



รูปที่ 3.9 วงจรดีเทคสัญญาณคลื่นพาหะ



วงจรกรองสัญญาณความถี่ต่ำ

รูปที่ 3.10 แสดงวงจรภาครับ

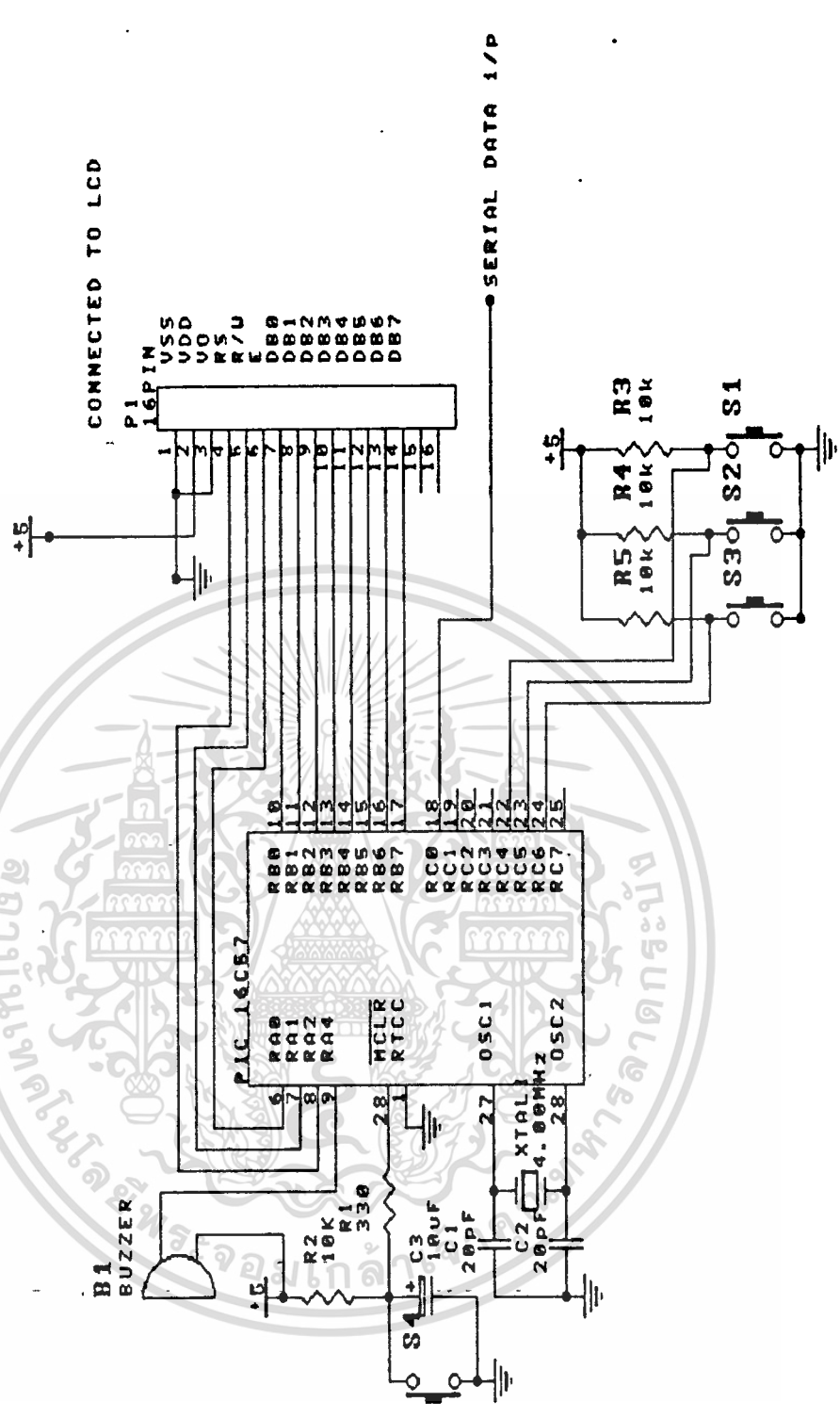
หลักการทำงานของวงจรภาครับ

จากรูปวงจรภาครับจะเป็นแบบ self-generative คือทรานซิสเตอร์ BF 199 ตัวที่ 2 จะทำหน้าที่เป็นทั้งออสซิลเลเตอร์ และมิกเซอร์ภายในตัวมันเอง โดยที่วงจรดักความถี่ (Tank circuit) จะมีค่าความถี่ LC - Resonance ตั้งไว้ตรงกับความถี่ที่ต้องการรับเข้ามาคือประมาณ 250 MHz เมื่อมีสัญญาณความถี่ที่ตรงกับเข้ามา วงจรดักความถี่ จะเกิดการเรโซแนนซ์ได้สัญญาณแรงเข้ามา สัญญาณที่รับได้นี้จะมีการหักล้างสัญญาณความถี่คลื่นพาหะกับความถี่ออสซิลเลเตอร์ที่วงจรดักความถี่ ซึ่งมีความถี่ตรงกัน ทำให้เหลือ แต่สัญญาณของข้อมูลดั้งเดิมที่ส่งมาจาก Serial Port RS - 232 ไปเข้าวงจรกรองความถี่ต่ำอีกทีหนึ่ง เพื่อกรองให้ได้สัญญาณความถี่ 1200 Hz ที่สมบูรณ์

สำหรับวงจรภาครับส่วนหน้าที่ต่ออยู่กับทรานซิสเตอร์ BF 199 ตัวที่หนึ่งนั้น จะต่อมาจากสายอากาศซึ่งทำหน้าที่เป็นบัฟเฟอร์ (Buffer) ให้ค่าอิมพีแดนซ์ของสายอากาศคงที่ไม่เปลี่ยนแปลง ถ้าต่อสายอากาศโดยตรงเข้าที่วงจรภาครับส่วนหลังที่ทำหน้าที่ดีเทคสัญญาณ จะทำให้ค่าอิมพีแดนซ์ของสายอากาศเปลี่ยนแปลงได้ง่าย ทำให้มีผลต่อการดีเทคสัญญาณที่ภาครับอาจมีความผิดพลาดได้

หลักการทำงานของวงจรกรองความถี่ต่ำ

วงจรกรองนี้อาศัยหลักการการขยายความแตดต่างของสัญญาณที่เข้ามา โดยที่ขาบวกของออปแอมป์ตัวแรกจะเป็นสัญญาณดั้งเดิม ส่วนสัญญาณที่เข้ามาทางขาลบจะถูกหน่วงเวลาออกไป ซึ่งความแตกต่างของสัญญาณของขาออปแอมป์ทั้งสองจะถูกขยายออกไปทางเอาท์พุทด้วยอัตราขยายที่กำหนด หลังจากนั้นสัญญาณที่ได้จะส่งต่อไปยังออปแอมป์ตัวที่สอง เพื่อทำการปรับรูปสัญญาณให้ดีขึ้น โดยทำหน้าที่คล้ายกับวงจรชมิททริกเกอร์ ทำให้ระดับสัญญาณแกว่งขึ้นไปจนถึงค่า Vcc



รูปที่ 3.11 วงจรส่วนควบคุมที่ภาครับ

3.4 ส่วนควบคุมของภาครับ

ส่วนควบคุมของภาครับนี้ใช้ไมโครคอนโทรเลอร์ PIC 16C57 ซึ่งผลิตโดยบริษัท ไมโครชิพ เทคโนโลยี (สำหรับรายละเอียดของไมโครคอนโทรเลอร์เบอร์นี้แสดงไว้ในภาคผนวก) เป็นตัวควบคุมการรับข้อมูลอนุกรมที่ออกมาจากภาครับ, การควบคุมการแสดงผลบนจอ LCD, การส่งเสียงเตือน, และการรับคีย์ต่าง ๆ

สำหรับคำอธิบายและโครงสร้าง (flow chart) ของโปรแกรมต่อไปนี้ จะแสดงเฉพาะโปรแกรมส่วนที่สำคัญ ๆ ดังนี้

- โปรแกรมการรับข้อมูลอนุกรม ซึ่งยังแบ่งเป็นโปรแกรมรับข้อมูล 1 ไบท์ และโปรแกรมรับชุดข้อมูล
- โปรแกรมหลัก สำหรับเชื่อมต่อโปรแกรมย่อยแต่ละส่วนเข้าด้วยกัน

โปรแกรมรับข้อมูลอนุกรม

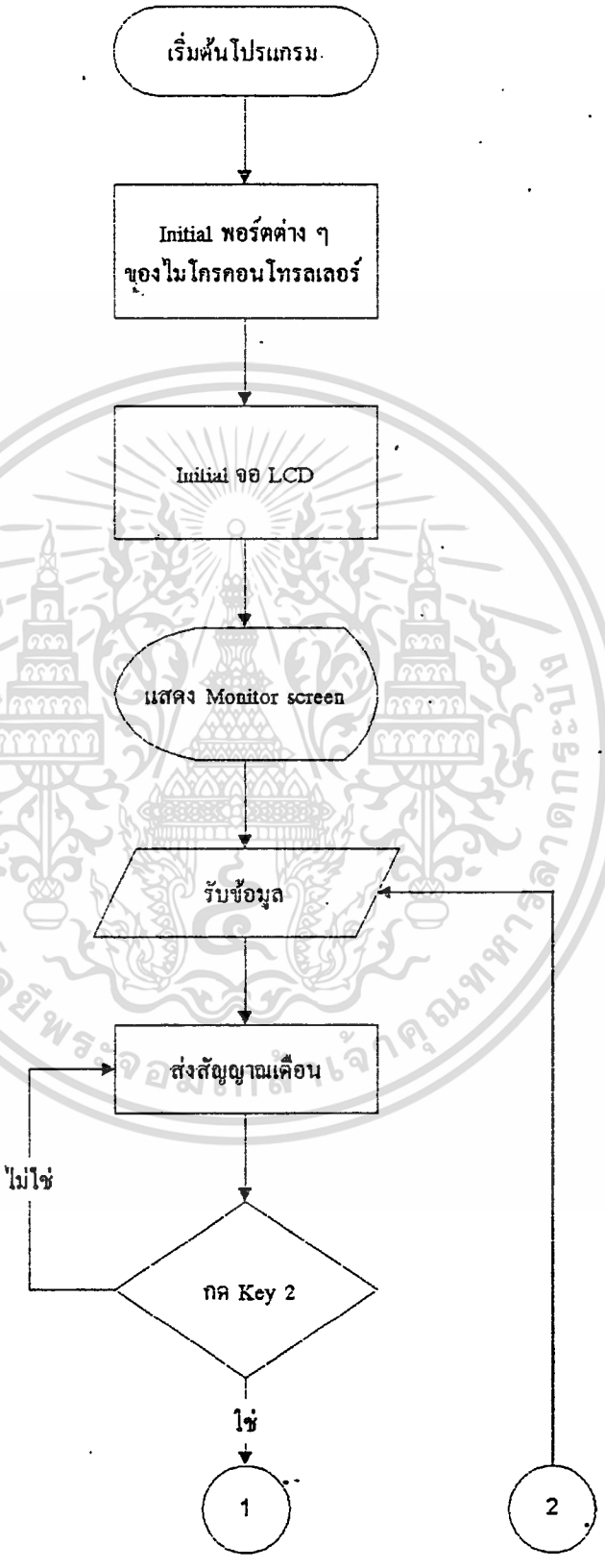
1) โปรแกรมรับข้อมูล 1 ไบท์

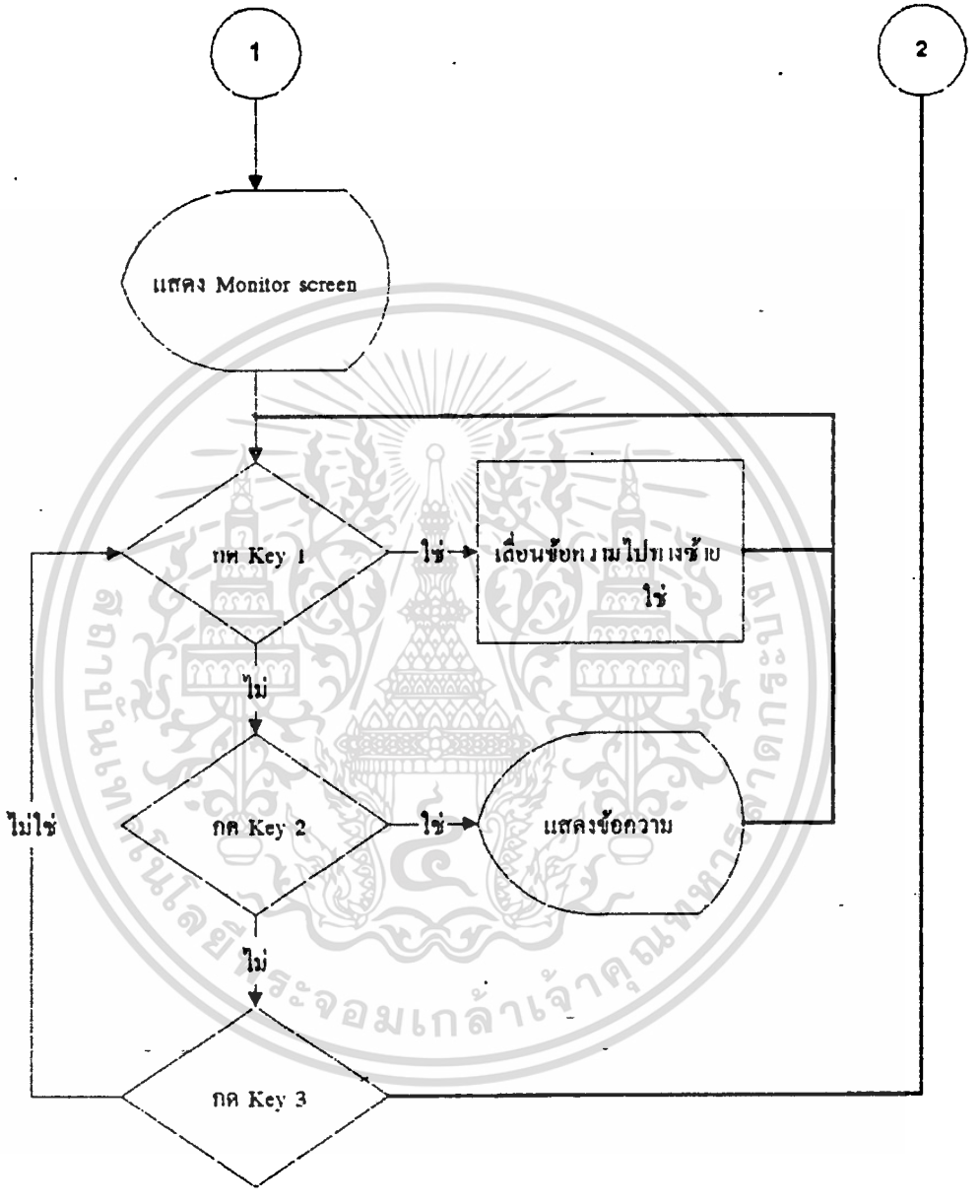
โปรแกรมส่วนนี้จะสั่งให้คอนโทรเลอร์ตรวจสอบหาบิทเริ่มต้นของข้อมูลอนุกรม ถ้าไม่ใช่จะวนตรวจสอบไปเรื่อย ๆ แต่ถ้าเป็นบิทเริ่มต้น โปรแกรมจะสั่งให้หน่วงเวลาไปครึ่งบิท และจะหน่วงเวลาต่อไปอีก 1 บิท เพื่อให้การสุ่มบิทข้อมูลบิทถัดไป กระทำที่กึ่งกลางของบิท จากนั้นจึงเอาค่าไบนารีของบิทที่ทำการสุ่มมาพักไว้ยังบิทตัวทด (carry bit) แล้วทำการเลื่อนบิทข้อมูลของรีจิสเตอร์เก็บข้อมูลไปทางขวา ข้อมูลที่อยู่ในบิทตัวทดจะถูกนำเข้าไปไว้ในบิทที่มีนัยสำคัญสูงสุดของรีจิสเตอร์สำหรับเก็บข้อมูล บิทข้อมูลอนุกรมบิทถัดไปจะถูกสุ่มเข้ามายังบิทตัวทดและทำการเลื่อนข้อมูลในรีจิสเตอร์ไปทางขวาไปเรื่อย ๆ จนครบ 8 บิท โปรแกรมจะสั่งให้หน่วงเวลาไปจนครบความยาวของบิทสิ้นสุด

2) โปรแกรมรับชุดข้อมูล

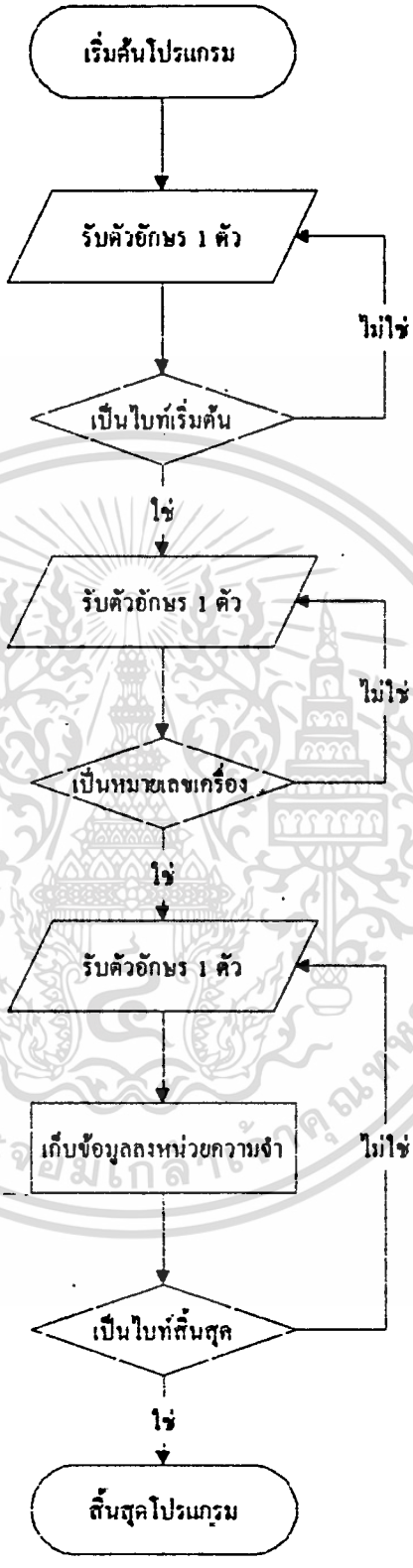
โปรแกรมส่วนนี้ทำหน้าที่รับชุดข้อมูลโดยอาศัยการเรียกโปรแกรมรับข้อมูล 1 ไบท์ ชุดข้อมูลที่ส่งมาจะแบ่งออกเป็นส่วน ๆ ดังที่ได้อธิบายมาแล้ว คือไบท์เริ่มต้น,

ไบท์แอดเดรส, ขบวนไบท์ข้อความ, และไบท์สิ้นสุด โปรแกรมจะสั่งให้คอนโทรเลอร์ทำการตรวจสอบหาไบท์เริ่มต้นของชุดข้อมูล ถ้าไม่พบก็ให้วนตรวจสอบไปเรื่อย ๆ เมื่อพบไบท์เริ่มต้น โปรแกรมจะสั่งให้รับไบท์ข้อมูลถัดมาซึ่งเป็นไบท์แอดเดรส ทำการตรวจสอบเปรียบเทียบกับแอดเดรสหรือหมายเลขของเครื่องรับที่กำหนดไว้ในโปรแกรม ถ้าไม่ตรงกันจะให้เริ่มกระบวนการรับชุดข้อมูลใหม่ แต่ถ้าตรง คอนโทรเลอร์จะรู้ว่าข้อมูลที่ตามเป็นข้อมูลข่าวสาร ให้ดำเนินการรับข้อมูลเข้ามาและเก็บไว้ในหน่วยความจำเรื่อย ๆ จนกระทั่งรับไบท์สิ้นสุดเข้ามา โปรแกรมจะสั่งให้หยุดการรับชุดข้อมูล





รูปที่ 3.12 โครงสร้างของโปรแกรมหลัก



รูปที่ 3.13 โครงสร้างของโปรแกรมรับชุดข้อมูล

รูปที่ 3.14 โครงสร้างของโปรแกรมรับข้อมูล 1 ไบท์

บทที่ 4

การทดลองและผลการทดลอง

ขั้นตอนการทดลอง

การทดลองที่ 1 การปรับวงจรภาครับ

1) ทำการส่งข้อมูล 'A' อย่างต่อเนื่องออกจากเครื่องส่ง วัดสัญญาณทางเอาท์พุทของภาครับ ทำการปรับค่าตัวเก็บประจุที่ปรับค่าได้ทางภาครับเพื่อให้มีความถี่ตรงกับภาคส่ง โดยวัดสัญญาณแรงดันทางเอาท์พุทของภาครับ ซึ่งสามารถสังเกตได้จากเครื่องวัดออสซิลโลสโคปจะต้องได้รูปขบวนรหัสแอสกี้ของ 'A' ที่สวยงาม

2) เขียนโปรแกรมควบคุมไมโครคอนโทรลเลอร์ที่ภาครับให้รับข้อมูลที่เข้ามาและแสดงออกจอ LCD

3) ทำการส่งข้อมูลออกจากภาคส่ง สังเกตว่าตรงกับที่ออกจอ LCD หรือไม่

การทดลองที่ 2 ความถูกต้องของการรับส่งข้อมูล

1) ทำการส่งข้อความจากเครื่องส่งไปยังเครื่องรับ 10 ครั้ง โดยเครื่องส่งและเครื่องรับอยู่ห่างจากเครื่องส่งประมาณ 3 เมตร สายอากาศเครื่องรับยาว 5 เซนติเมตร บันทึกจำนวนครั้งที่เครื่องรับรับข้อความได้อย่างถูกต้องลงในตารางบันทึกผล

2) ทำการทดลองเช่นเดียวกับการทดลองในข้อ 1) แต่ทำการเปลี่ยนระยะห่างระหว่างเครื่องส่งและเครื่องรับออกไปอีกเป็น 6 ,9 ,,10 ,12 และ 14 เมตรตามลำดับ

3) ทำการทดลองข้อ 1) และ ข้อ 2) ซ้ำแต่เพิ่มความยาวของสายอากาศเครื่องรับเป็น 10 และ 30 เซนติเมตรตามลำดับ บันทึกผลการทดลองลงตาราง

ผลการทดลอง

ระยะทางระหว่างเครื่องส่ง-เครื่องรับ (เมตร)	จำนวนครั้งที่รับข้อมูลได้ถูกต้อง			เปอร์เซ็นต์ความผิดพลาด		
	สายอากาศ (เซนติเมตร)			สายอากาศ (เซนติเมตร)		
	5	10	30	5	10	30
3	10	10	10	0	0	0
6	8	10	10	20	0	0
9	4	9	10	60	10	0
10	0	3	9	100	70	30
12	0	0	7	100	100	30
14	0	0	2	100	100	80

จากการทดลองจะเห็นได้ว่า ระยะทางระหว่างเครื่องส่งและเครื่องรับ รวมทั้งความยาวของสายอากาศมีผลต่อการรับส่งข้อมูลระหว่างเครื่องส่งและเครื่องรับ คือถ้าระยะทางไกลขึ้นเครื่องรับจะไม่สามารถรับได้หรือรับได้แต่ข้อมูลนั้นจะผิดพลาดสำหรับสายอากาศนั้น ถ้าใช้สายอากาศยาวประมาณ ความยาวคลื่น/4 ก็จะทำให้เครื่องรับสามารถรับได้ไกลขึ้น ซึ่งความถี่ที่ใช้ประมาณ 250 MHz ดังนั้นจะได้ค่าความยาวคลื่นเท่ากับ 1.2 เมตร ซึ่งจะได้สายอากาศยาว 30 เซนติเมตร

หมายเหตุ สัญญาณเอาท์พุทที่ต่อออกมาจากภาครับก่อนที่จะนำไปเข้าที่อินพุทของไมโครคอนโทลเลอร์จะต้องมีการต่อค่าความต้านทาน 2.2 กิโลโอห์ม ไว้พูลดาวน์สัญญาณด้วย เนื่องจากว่าสัญญาณเอาท์พุทที่ออกมาจากภาครับนั้น ระดับกราวด์ของสัญญาณยังไม่ถึง 0 โวลท์ แต่จะยกระดับขึ้นมาประมาณ 1 โวลท์ ทำให้ไมโครคอนโทลเลอร์ไม่สามารถตรวจข้อมูลไบนารี '0' ได้ จึงรับข้อมูลที่ส่งมาไม่ได้

บทที่ 5

บทวิจารณ์และสรุป

โครงงานนี้เป็นเพียงการจำลองระบบวิทยุติดตามตัวเพื่อศึกษาการทำงานของระบบ ซึ่งจะเห็นได้ว่าการรับส่งข้อความนั้นยังไม่สามารถนำไปใช้งานได้ในชีวิตประจำวัน เนื่องจากสาเหตุต่าง ๆ ได้แก่ กำลังของเครื่องส่งไม่สูงทำให้ส่งได้ระยะทางไม่ไกลประมาณ 10 เมตร แต่ถ้าระยะทางไกลความผิดพลาดของข้อมูลจะเกิดขึ้น , ขนาดของเครื่องรับและแหล่งจ่ายไฟของเครื่องรับ เป็นต้น

ปัญหาของการรับส่งข้อมูลของระบบวิทยุติดตามตัวในโครงงานนี้ ซึ่งมีผลทำให้ประสิทธิภาพของการรับส่งลดลงประกอบด้วยสาเหตุดังต่อไปนี้ คือ

1. สัญญาณรบกวนจากภายนอก ซึ่งเข้ามาในระหว่างการส่งและรับข้อมูล

2. กำลังส่งของเครื่องส่ง ซึ่งมีกำลังต่ำทำให้ส่งได้ระยะทางไม่ไกล ซึ่งถ้าเครื่องรับอยู่นอกระยะที่เหมาะสมของเครื่องส่งแล้ว จะทำให้เครื่องรับรับสัญญาณข้อมูลที่ส่งมาไม่ได้หรือถ้ารับได้ก็อาจจะรับข้อมูลไม่ถูกต้องตามที่ส่งมา

3. เครื่องรับยังต้องใช้แหล่งจ่ายไฟสูงอยู่คือ 12 โวลท์ ทำให้ไม่เกิดการประหยัดพลังงาน ทั้งยังเพิ่มน้ำหนักของแหล่งจ่ายไฟเข้ามาด้วย

4. ขนาดของเครื่องรับซึ่งยังไม่สามารถพกพาได้สะดวก

สำหรับระบบวิทยุติดตามตัวที่นิยมใช้กันอยู่ในชีวิตประจำวันนั้น มีขนาดเล็กเนื่องจากวงจรภาครับที่ใช้นั้นจะรวมอยู่ภายในชิพไอซีเพียงตัวเดียว ทั้งยังเป็นการลดผลการรบกวนจากอุปกรณ์ต่างในวงจรลงและยังต้องการพลังงานต่ำ

ข้อเสนอแนะและแนวทางในการพัฒนา

1. การรบกวนของสัญญาณ ต้องพยายามลดการรบกวนให้น้อยที่สุด เช่น การบัดกรีขาอุปกรณ์ควรทำให้สั้น เป็นต้น

2. ทางด้านเครื่องส่ง หากสามารถเพิ่มกำลังส่งของเครื่องส่งขึ้นได้ก็จะทำให้สามารถส่งได้ไกลขึ้น และสัญญาณข้อมูลจะถูกรบกวนจากสัญญาณภายนอกลดลง อาจทำได้โดยเพิ่มวงจรขยายกำลังเข้าไป นอกจากนี้ถ้าหากเปลี่ยนวงจรภาคส่งให้มีการมอดดูเลท

จากแบบ AMPLITUDE SHIFT KEYING (ASK) มาเป็นแบบ FREQUENCY SHIFT KEYING (FSK)ได้ก็จะดีขึ้น เนื่องจากสัญญาณรบกวนจะเข้าไปที่แอมปลิจูดของสัญญาณซึ่งจะไม่มีผลต่อ FSK อีกทั้งการแอมปลิจูดของสัญญาณก็จะไม่ส่งผลต่อการเปลี่ยนแปลงความถี่ เช่น การลดลงของแอมปลิจูดของสัญญาณเนื่องจากระยะทาง เป็นต้น แต่จะมีผลใน ASK

3. ทางด้านภาครับ ควรหาวงจรภาครับที่ต้องการพลังงานต่ำ ขนาดเล็ก เพื่อให้เกิดการประหยัดทั้งทางด้านพลังงานและขนาด ทำให้สามารถพกพาได้

ดังนั้น หากพัฒนาปรับปรุงแก้ไขโครงงานนี้ต่อไปแล้วเชื่อว่าโครงงานนี้คงสามารถนำไปใช้งานได้จริงในชีวิตประจำวัน

ภาคผนวก ก.

ไมโครคอมพิวเตอร์ชิพเดียวตระกูล PIC

PIC เป็นอนุกรมไมโครคอมพิวเตอร์ชิพเดียวของ General Instrument MCU (MCU ย่อมาจาก Micro Computer Unit) ในตระกูล PIC นี้สร้างขึ้นโดยใช้เทคโนโลยีของ MOS/LSI และมีบางเบอร์ที่ใช้ CMOS เทคโนโลยี โครงสร้างภายในประกอบไปด้วยหน่วยหลักต่าง ๆ คือ หน่วยประมวลผลกลาง (CPU),หน่วยความจำ (memory) และอินพุท/เอาท์พุท (Input/Output) คุณลักษณะโดยทั่วไปของ MCU แต่ละเบอร์ในอนุกรมนี้แสดงดังตารางที่ 1

MCU ในตระกูลนี้ มีลักษณะโครงสร้างที่น่าสนใจ คือเป็นโครงสร้างข้อมูลขนาด 8 บิท แต่คำสั่งเป็นคำสั่งที่มีความกว้างขนาด 12 บิท (หรือ 13 บิท ซึ่งขึ้นอยู่กับว่าเป็นเบอร์ไหน) ที่เป็นเช่นนี้เพราะ GI(General Instrument) ต้องการที่จะให้มีชุดคำสั่งที่ง่ายในการเรียกใช้ และมีประสิทธิภาพสูง นอกจากนี้แล้วยังมีคุณลักษณะอื่น ๆ ที่น่าสนใจคือ RAM ภายในทุกตัวหน้าที่เสมือนรีจิสเตอร์ ซึ่งทำให้สามารถอ้างแอดเดรสได้โดยตรงและยังมีคำสั่งที่ทำการเซทหรือเคลียร์บิทใด ๆ ในรีจิสเตอร์นั้น ๆ ได้ ซึ่งหน้าที่และคำสั่งต่าง ๆ จะได้กล่าวถึงต่อไป

คุณสมบัติ

-ราคาถูก

-ใช้กำลังงานต่ำ

-เป็นอุปกรณ์ตระกูลซีมอส

-ภายในประกอบด้วย หน่วยประมวลผล,หน่วยความจำ และ อินพุทเอาท์พุท

-รับสัญญาณนาฬิกาได้ตั้งแต่ย่าน DC-20 MHz

-มีพอร์ตอินพุต-เอาต์พุตตั้งแต่ 8-20 เส้น

PIC16C5X จะถูกใช้งานในลักษณะช่วยลดขนาดของคำสั่งที่ใช้ในการเซตระบบ

คอมพิวเตอร์ RISC (reduce instruction set computer) ทำให้การสั่งงานใช้เพียงคำสั่งเดียว และในวงจรการทำงานรอบเดียว ภายในจำนวนเพียง 33 รอบคำสั่ง โดยในแต่ละรอบคำสั่งจะใช้เวลาประมาณ 200 nS ซึ่งอุปกรณ์ PIC นี้เป็นผลิตภัณฑ์ทางด้านไมโครคอนโทรลเลอร์ที่น่าสนใจ และในอนาคตจะช่วยให้ลดต้นทุนในการพัฒนาซอฟต์แวร์และฮาร์ดแวร์ลงไปได้มาก

ตารางที่ 1 คุณสมบัติทางไฟฟ้าของไอซีตระกูล PIC

คุณสมบัติ	ค่า
แรงดันทำงาน	2.5 - 6.25 V
การสูญเสียกำลังงานที่ 5 V 4 MHz	< 2mA
ความเร็วในการทำงาน (สัญญาณนาฬิกาอินพุต)	DC-20 MHz
ความเร็วในการทำงาน (วงรอบของคำสั่งข้อมูล)	DC-200 nS
ขนาดคำสั่ง	12 BITS
ขนาดข้อมูล	8 BITS
หน่วยความจำโปรแกรมอีพรอม (EPROM)	512-2k*12 kbyte*bit
ขนาดรีจิสเตอร์ (SRAM) โดยทั่วไป	25 - 72*8 kbyte*bit
อุณหภูมิโดยรอบขณะทำงาน	-55 ถึง +125 C^{o}
แรงดันที่ขา MCLR เทียบกับกราวด์	0 - 14 V
การสูญเสียกำลังงานโดยรวม	800 mW
กระแสเอาต์พุตสูงสุดที่ขา V_{ss}	150 mA
กระแสเอาต์พุตสูงสุดที่ขา V_{DD}	50 mA

ตาราง 2 ความแตกต่างของหน่วยความจำและส่วนประกอบอื่นที่ตัวถังบรรจุ

เบอร์PIC	EPROM	RAM	INPUT/ OUTPUT	แรงดันไฟ (V)	ย่านความถี่ (MHz)	ตัวถังบรรจุ
PIC16C54	512*12	32*8	13	4.0-5.5	DC-20	DIP 18 ขา
PIC16C55	512*12	32*8	21	4.0-5.5	DC-20	DIP 18 ขา
PIC16C56	1K*12	32*8	13	4.0-5.5	DC-20	DIP 18 ขา
PIC16C57	2K*12	80*8	21	4.0-5.5	DC-20	DIP 18 ขา

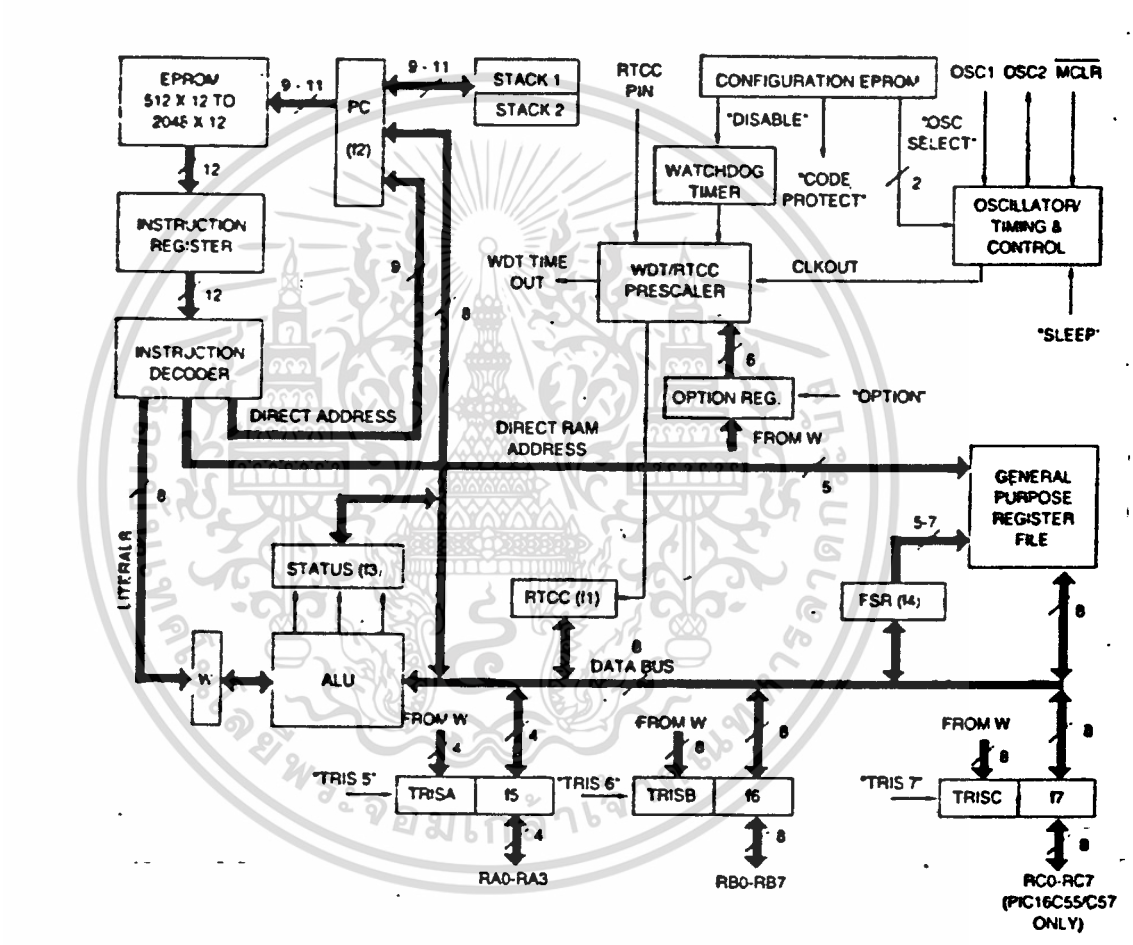
โครงสร้างโดยทั่วไปของ PIC

โครงสร้างพื้นฐานของ MCU ในตระกูล PIC นี้ แสดงในรูปที่ 2 อันเป็นโครงสร้างพื้นฐานของ MCU ในตระกูล PIC ทุกเบอร์ที่จะต้องมี และเพื่อให้เกิดความเข้าใจได้โดยง่าย ให้พิจารณาการจัดระบบของหน่วยความจำประเภทแรม ซึ่งใช้เป็นรีจิสเตอร์ทั่วไป และยังมีบางแอดเดรสที่ทำหน้าที่เป็น I/O (เป็นลักษณะ memory I/O mapping) ดังในรูปที่ 1 และรูปที่ 2

W	Working register
F0	Indirect address pointer register
F1	RTCC register
F2	Program Counter
F3	Status register
F4	File select register
F5	I/O register (port A)
F6	I/O register (port B)
F7	PIC 16C55/57 - I/O register (port C) PIC 16C54/56 - General purpose register

F8 - F31	General purpose registers. Note this syntax assumes a default radix of DECIMAL. The number following the 'F' must be in the current radix.
F32 - 79	General purpose registers for PIC 16C57 only. The number following the 'F' must be in the current radix.
IOA	I/O TRIS status register, port A
IOB	I/O TRIS status register, port B
IOC	I/O TRIS status register, port C (for PIC 16C55 only)
OPT	Prescaler assignment, signal source and signal edge

รูปที่ 1 การจัดระบบของรีจิสเตอร์ใน PIC 16c5x



รูปที่ 2 แผนผังภายในของ MCU ในตระกูล PIC

คุณสมบัติที่ดีมากอีกประการหนึ่งของหน่วยความจำแรม ใน PIC นี้คือ ทุกแอดเดรสของแรมจะทำหน้าที่เป็นรีจิสเตอร์ได้ทุกตัว ซึ่งสามารถอ้างแอสเดรสได้โดยตรง (direct address)ได้ และสามารถกระทำทางคณิตศาสตร์ได้โดยตรงกับรีจิสเตอร์ นอกจากนั้นผล

ของการกระทำที่ได้ยังสามารถที่จะเก็บไว้ในรีจิสเตอร์ที่มากระทำร่วมกันนั้นอีกด้วย ซึ่งทั้งหมดนี้สามารถทำเสร็จสิ้นใน 1 คำสั่งเท่านั้น

ต่อไปจะกล่างถึงรายละเอียดบลีกย่อยในแต่ละบล็อคซึ่งมีความสัมพันธ์กับรูปที่ 1 และ 2 ในแต่ละส่วนดังต่อไปนี้

1. ส่วนถอดรหัสและควบคุม (Instruction Decode and Control Unit)

จากรูปที่ 2 จะเห็นว่าส่วนถอดรหัสและควบคุมนี้จะติดต่อโดยตรงกับ ROM ซึ่งเป็นโปรแกรมคำสั่ง เมื่อส่วนนี้ตีความหมายของคำสั่งได้แล้วจะไปควบคุมการทำงานของอุปกรณ์หรือหน่วยอื่น ๆ ในชิพนั้นทั้งหมด

2. โปรแกรมเคาน์เตอร์ (Program Counter) (F2)

โปรแกรมเคาน์เตอร์ เป็นรีจิสเตอร์หนึ่งซึ่งทำหน้าที่ชี้คำสั่งต่อไปที่จะนำมาตีความหมายและทำงาน โดยปรกติแล้วค่าในโปรแกรมเคาน์เตอร์จะเพิ่มขึ้น 1 ค่าทุกครั้งที่ทำไปคำสั่งหนึ่ง เว้นแต่เมื่อทำตามคำสั่งประเภทกระโดด เช่น GO TO หรือคำสั่ง CALL เป็นต้น โครงสร้างของโปรแกรมเคาน์เตอร์เป็นขนาด 9 บิท โดยที่บิทที่ 0 ถึงบิทที่ 7 สามารถที่จะถ่ายเทข้อมูลไปสู่สแต็คได้ ส่วนบิท 8 นั้นโดยปรกติจะเป็น 0 ซึ่งในลักษณะการทำงานของโปรแกรมเคาน์เตอร์จะกล่าวถึงอีกครั้ง เมื่อถึงคำสั่ง CALL

3. สแต็ค (Stack)

สแต็คเป็นรีจิสเตอร์หนึ่ง ซึ่งทำหน้าที่เก็บโปรแกรมเคาน์เตอร์ของคำสั่งต่อไปที่จะทำงานหลังจากที่ได้ทำโปรแกรมย่อยเสร็จเรียบร้อยแล้ว สแต็คใน PIC นี้เป็นฮาร์ดแวร์สแต็ค (hardware stack) ขนาด 8 บิท 2 ระดับ ซึ่งหมายถึงสามารถที่จะเรียกโปรแกรมย่อยซ้อนกันได้ 2 โปรแกรม

4. File Select Register (F4)

เรียกสั้น ๆ ว่า FSR มีตำแหน่งอยู่ที่ F4 (จากรูปที่ 1 และ 2) เป็นรีจิสเตอร์ขนาด 5 บิท มีไว้เพื่อทำหน้าที่เป็นตัวชี้แอดเดรสในคำสั่งประเภท indirect Mode ที่เกี่ยวกับรีจิสเตอร์ ตัวอย่างเช่น คำสั่ง ADDWF 0,W ข้อมูลที่เก็บใน F4 นั้นจะเป็นตัวชี้ข้อมูลในรีจิสเตอร์ที่ระบุด้วย F4 แล้วบวกกับ W (W รีจิสเตอร์เป็นแอคคิวมูเลเตอร์) ผลที่ได้เก็บไว้ใน W ในการโหลดข้อมูลเข้าไปยัง F4 นี้จะใช้คำสั่งโหลดข้อมูลขนาด 8 บิท

แต่ F4 จะรับได้เพียง 5 บิทล่างเท่านั้น (บิทที่ 0 ถึงบิทที่ 4)

5. ส่วนคำนวณทางคณิตศาสตร์ (Arithmatic Logic Unit) (ALU)

เป็นหน่วยคำนวณของ MCU ซึ่งมีหน้าที่หลักดังนี้

1. บวกและลบค่า
2. เพิ่มและลดค่าทีละ 1
3. กระทำทางตรรก เช่น AND OR และ Exclusive OR
4. Complement และ Clear
5. Rotate ข้อมูล ทางซ้าย/ขวา หรือสลับค่าระหว่างข้อมูลใน 4 บิทบน (b_4-b_7) กับข้อมูลใน 4 บิทล่าง (b_0-b_3)

ซึ่งหน้าที่หลักต่าง ๆ ดังกล่าวนี้เพียงพอที่จะนำไปสร้างชุดคำสั่งด้านคำนวณชั้นสูงขึ้นไปได้

6. รีจิสเตอร์ทำงาน (Work Register) (W)

W รีจิสเตอร์ในที่นี้ก็คือ แอคคิวมูเลเตอร์ ของ ALU นั่นเอง

7. รีจิสเตอร์บอกสถานะ (Status Word Register) (F3)

เป็นรีจิสเตอร์ที่ทำหน้าที่เก็บสถานะต่าง ๆ ของ accmulater ซึ่งรู้จักกันดีในชื่อของแฟลก (flag) ซึ่งมีทั้งหมด 4 แฟลก ตั้งแต่บิทที่ 0 ถึง 3 บิทอื่น ๆ นอกเหนือจากนี้แล้วไม่ใช้แฟลกทั้งสี่มีดังรูปที่ ๆๆ

บิทที่	7-4	3	2	1	0
หน้าที่	NOT USE	OV	Z	DC	C

รูปที่ 3 Status Register ของ PIC

– C (Carry) : บิทที่ 0 เป็นแฟลกตัวทดจะเปลี่ยนไปในกรณีที่มีการใช้คำสั่ง Rotate บิทผ่านแฟลกตัวทดนี้ และเมื่อมีการใช้คำสั่งบวกหรือลบเลขระหว่างรีจิสเตอร์และแอคคิวมูเลเตอร์ เมื่อมีการทดหรือยืมเกิดขึ้นแฟลกตัวทดจะเซทเป็น 1

- DC (Digit Carry) : บิทที่ 1 รู้จักกันในชื่อของ Auxiriary Carry Flag DC จะเซทเป็น 1 ต่อเมื่อมีการบวกหรือลบเลข 2 จำนวน และค่าที่ได้นั้นมีการยืมหรือทดในระหว่างบิทที่ 3 และ 4

- Z (Zero) : บิทที่ 2 บิทนี้จะเซทเป็น 1 ต่อเมื่อมีการกระทำกับคณิตศาสตร์หรือลอจิก แล้วหาผลลัพธ์ที่ได้มีค่าเป็น 0

- OV (Overflow) : บิทที่ 3 บิทนี้จะเซทเป็น 1 ต่อเมื่อมีการทดจากบิทสูงสุดของผลลัพธ์ที่ได้คือทดจากบิทที่ 6 ไปบิทที่ 7 (ในที่นี้หมายถึงบิทที่ 7 เป็น sign บิท)

8. เคาน์เตอร์รีจิสเตอร์ (Real - Time Clock/Counter Register)

คุณลักษณะที่ดีอีกประการหนึ่งของ PIC คือ มีเคาน์เตอร์รีจิสเตอร์อยู่ภายในรีจิสเตอร์นี้สามารถทำเป็น real - time clock หรือเป็นเคาน์เตอร์นับขึ้นก็ได้ ทำให้มีประโยชน์มากในการทำลูปดีเลย์ต่าง ๆ ไม่ว่าจะทำหน้าที่เป็นไทม์เมอร์ (timer) หรือเคาน์เตอร์รีจิสเตอร์นี้เรียกว่า RTCC รีจิสเตอร์

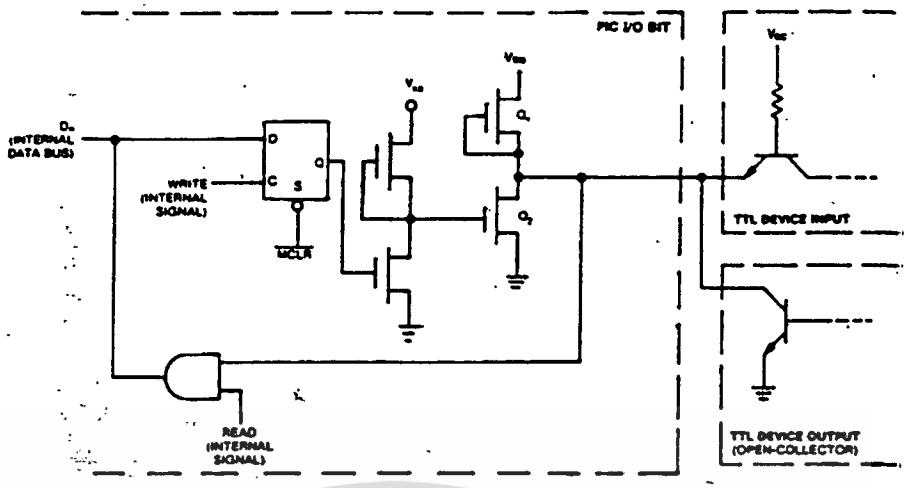
9. อินพุทเอาท์พุทรีจิสเตอร์ (I/O Register)

จากตารางที่ ๆๆ จะเห็นได้ว่าอินพุทเอาท์พุท ของ PIC ในแต่ละเบอร์นั้นมีจำนวนไม่เท่ากัน ซึ่งก็แล้วแต่ความเหมาะสมของงานที่จะเลือกนำไปใช้ ซึ่งก็มีจำนวนอินพุทเอาท์พุทบิทตั้งแต่ 12 บิท ไปจนถึง 32 บิท โครงสร้างของแต่ละบิทส่วนใหญ่เป็นสองทิศทางโดยไม่ต้องมีการโปรแกรมล่วงหน้าว่าเป็นอินพุทหรือเอาท์พุท จากรูปที่ 4 จะเห็นวงจรเทียบเท่าของแต่ละอินพุทเอาท์พุทบิท และการอินเตอร์เฟสกับไอซีตระกูลทีทีแอล (TTL)

ในลักษณะของการเอาท์พุท สัญญาณจะถูกแลทช์ด้วยฟลิป-ฟลอป ตามคำสั่งเขียน (write) ส่วนในลักษณะของการอินพุทในโปรแกรมจะต้องมีการสั่งเอาท์ค่า 1 ออกไปที่อินพุทเอาท์พุทบิทนั้น ๆ แล้วจึงสั่งอินพุทเข้ามา จะไม่ค้าง (latch) แต่จะอ่านทันทีที่มีสัญญาณอ่าน (read)

10.หน่วยความจำโปรแกรม (Program Memory) (ROM)

เป็นหน่วยความจำที่ทำหน้าที่เก็บโปรแกรมการทำงานของเครื่อง มีความกว้าง 12 บิท(สำหรับอนุกรม 16c5x) หรือ 13 บิท(สำหรับอนุกรม 16c7x)ทุกคำสั่งเป็นคำสั่งขนาดคำ (word) เดียว และทำงานเพียง 1 รอบการทำงาน(machine cycle)เท่านั้น



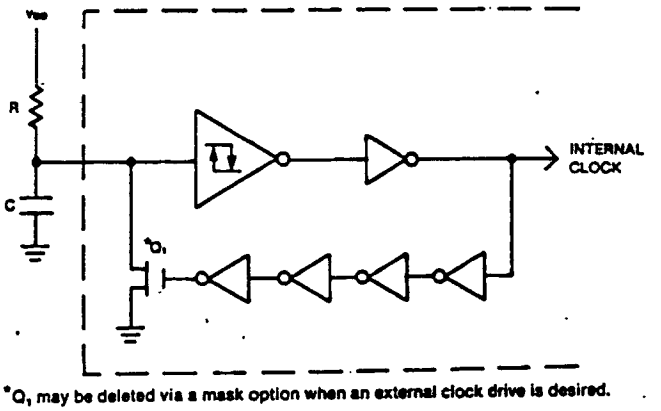
รูปที่ 4 วงจรเทียบเท่าของ I/O แต่ละบิท

11. หน่วยความจำข้อมูล (Data Memory) (RAM)

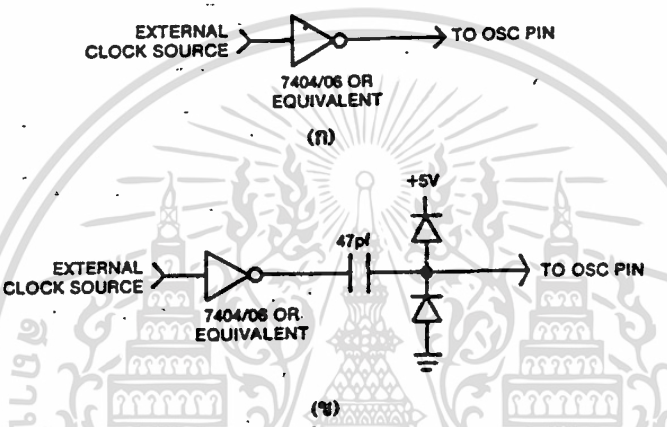
เป็นส่วนที่เก็บข้อมูลทั้งหมดที่ใช้ในการทำงาน ซึ่งทำหน้าที่เป็นรีจิสเตอร์ทั่วไปและรีจิสเตอร์พิเศษอื่น ๆ เช่น โปรแกรมเคาน์เตอร์ เป็นต้น

12. ส่วนกำเนิดสัญญาณนาฬิกา (Clock Generater)

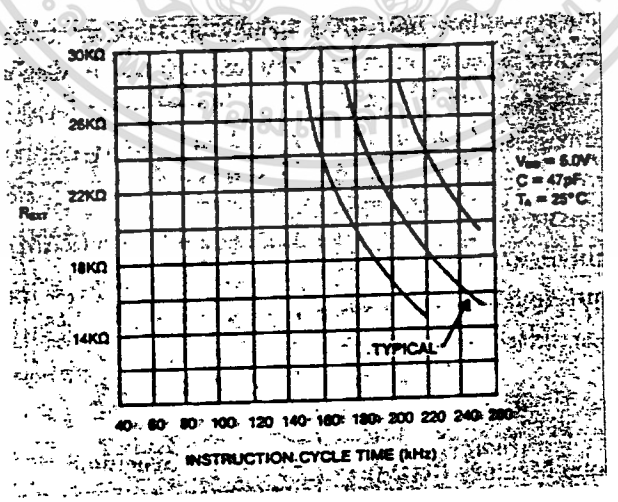
เป็นส่วนกำเนิดสัญญาณนาฬิกาซึ่งเกิดจากออสซิลเลเตอร์ (oscillator) ที่ขา OSC ที่มีความถี่ตั้งแต่ 200 kHz ถึง 1 MHz ได้จากการต่อ R ,C ภายนอกดังรูปที่ 5 หรือจากสัญญาณนาฬิกาภายนอก (external clock) ดังรูปที่ 6. สัญญาณจากออสซิลเลเตอร์ หากมีความถี่ 1 MHz จะถูกหารด้วย 4 จากวงจรภายใน ซึ่งจะได้ความถี่ 0.25 MHz หรือ 1 คาบ = 4 us ซึ่งเป็นช่วงเวลาของการทำงานใน 1 รอบการทำงาน ความถี่ออสซิลเลเตอร์นี้กำหนดให้อยู่ในช่วงระหว่าง 200 kHz ถึง 1 MHz รูปที่ 7 เป็นตารางการเลือกค่า R และ C ที่เหมาะสมที่จะให้ได้ความถี่ออสซิลเลทขนาดต่าง ๆ กัน



รูปที่ 5: การต่อ R,C ที่ขา OSC



รูปที่ 6 การต่อ external clock ที่ OSC



รูปที่ 7 ตารางเลือกค่า R และ C

การทำงานหรือหน้าที่การใช้งานในแต่ละขาของไอซี PIC มีดังนี้

- RA_0-RA_3 (I/O port A) เป็นขาอินพุทเอาท์พุทของพอร์ท A มีทั้งหมด 4 อินพุทเอาท์พุทบิท

- RB_0-RB_7 (I/O port B) เป็นขาอินพุทเอาท์พุทของพอร์ท B มีทั้งหมด 8 อินพุทเอาท์พุทบิท

- RC_0-RC_7 (I/O port C) เป็นขาอินพุทเอาท์พุทของพอร์ท C มีทั้งหมด 8 อินพุทเอาท์พุทบิท มีเฉพาะใน PIC16C55 และ PIC16C57 เท่านั้น

- RTCC (real time clock/counter) เป็นขาชมิตทริกเกอร์อินพุท ซึ่งจะรับสัญญาณนาฬิกาเข้ามาทางอินพุตไปสู่รีจิสเตอร์ โหมดนี้ในบางครั้งหากไม่ต้องการใช้งานที่ขานี้จะนำไปต่อเข้ากับไฟบวก (V_{DD}) หรือกราวด์ (V_{SS}) ก็ได้ โดยเฉพาะในโหมดทดสอบ และทำให้สิ้นเปลืองกำลังงานลดลงด้วย

- MCLR (master clear) เป็นขาชมิตทริกเกอร์อินพุท เช่นกัน เมื่อได้รับแรงดันเป็นลอจิก 0 จะเป็นการรีเซตอินพุท สำหรับ PIC ตระกูล PIC16C5X และขณะที่แรงดันกระตุ้นเพิ่มสูงขึ้น วงจรออสซิลเลเตอร์ก็จะทำงาน เป็นการตั้งเวลาในการรีเซท ซึ่งจะกินเวลาหน่วงไว้ประมาณ 18 mS โดยที่จาก input ทริกเกอร์นี้สามารถต่อกับวงจรภายนอกได้โดยตรง หรือต่อกับตัวต้านทานพูลอัพที่ต่อเข้ากับไฟบวก

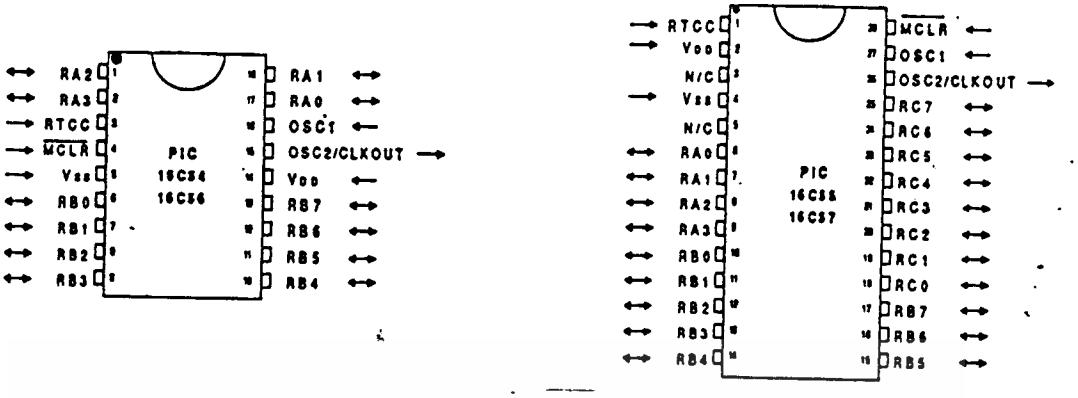
- OSC_1 (oscillator input)เป็นขาออสซิลเลเตอร์อินพุต ซึ่งสามารถใช้ต่อกับอุปกรณ์กำเนิดความถี่ได้หลายแบบคือ จากคริสตอล เซรามิกเรโซเนเตอร์ RC oscillator หรือแหล่งกำเนิดสัญญาณนาฬิกาภายนอก

- OSC_2/CLK_{out} (oscillator output) เป็นวงจรออสซิลเลเตอร์เอาต์พุตจะใช้งานร่วมกับขา OSC_1 จะไม่สามารถต่อขา OSC_2 นี้ไปเข้ากับวงจรภายนอกได้ เนื่องจากจะถูกโหลดมากเกินไป แต่ถ้าหากว่าใช้แหล่งกำเนิดสัญญาณนาฬิกาจากภายนอกป้อนเข้าทางขา OSC_1 จึงจะสามารถนำขา OSC_2 ไปต่อกับวงจรภายนอกได้และความถี่ที่ออกมาทางขานี้จะเป็น 1/4 ของความถี่จาก OSC_1

- V_{DD} (power supply)ขาแรงดันไฟบวกเลี้ยงตัวไอซี

- V_{ss} (ground) ขากราวด์ของไอซี

- N/C (No (intenal) connection) เป็นขาที่ไม่ได้ต่อใช้งาน



รูปที่ 8 การจัดวางตำแหน่งขาต่าง ๆ ของ PIC เบอร์ต่าง ๆ

จากหน้าที่การทำงานของขาใช้งานต่าง ๆ ก็จะนำไปสู่การประยุกต์ใช้งาน PIC 16C5X การประยุกต์ใช้งานนั้นมีมากมายหลายรูปแบบ เช่น การออกแบบให้ควบคุมมอเตอร์ รวมทั้งการกำหนดตำแหน่งด้วย การออกแบบใช้งานในโปรแกรม การตั้งเวลา เป็นต้น

BYTE -ORIENTED FILE REGISTER OPERATIONS

(11-6)	(5)	(4 - 0)
OPCODE	d	f(FILE #)

d = 0 for destination W
d = 1 for destination f
f = 5 bit file register address

Instruction-Binary	(Hex)	Name	Mnemonic,Operands	Operation	Status Affected	Notes
0001 11df ffff	1Cf	Add W and f	ADDWF f,d	W + f → d	C,DC,Z	1,2,4
0001 01df ffff	14f	AND W and f	ANDWF f,d	W & f → d	Z	2,4
0000 011f ffff	06f	Clear f	CLRF f	0 → f	Z	4
0000 0100 0000	040	Clear W	CLRW -	0 → W	Z	
0010 01df ffff	24f	Complement f	COMF f,d	$\bar{f}$ → d	Z	2,4
0000 11df ffff	0Cf	Decrement f	DECF f,d	f -1 → d	Z	2,4
0010 11df ffff	2Cf	Decrement f,Skip if Zero	DECFSZ f,d	f - 1 → d, skip if zero	None	2,4
0010 10df ffff	28f	Increment f	INCF f,d	f + 1 → d	Z	2,4
0011 11df ffff	3Cf	Increment f,Skip if Zero	INCFSZ f,d	f + 1 → d, skip if zero	None	2,4
0001 00df ffff	10f	Inclusive OR W and f	IORWF f,d	W v f → d	Z	2,4
0010 00df ffff	20f	Move f	MOVF f,d	f → d	Z	2,4
0000 001f ffff	02f	Move W to f	MOVWF f	W → f	None	1,4
0000 0000 0000	000	No Operation	NOP -	-	None	
0011 01df ffff	34f	Rotate left f	RLF f,d	f(n)→d(n+1),C →d(0), f(7)→C	C	2,4
0011 00df ffff	30f	Rotate right f	RRF f,d	f(n)→ d(n-1),C → d(7),f(0) → C	C	2,4
0000 10df ffff	08f	Subtract W from f	SUBWF f,d	f - W → d [f + $\bar{W}$ + 1 → d]	C,DC,Z	1,2,4
0011 10df ffff	38f	Swap halves f	SWAPF f,d	f(0-3) ↔ f(4-7) → d	None	2,4
0001 10df ffff	18f	Exclusive OR W and f	XORWF f,d	W ⊕ f → d	Z	2,4

BIT- ORIENTED FILE REGISTER OPERATIONS

(11-8)	(7-5)	(4 - 0)
OPCODE	b(BIT #)	f(FILE #)

b = 3 bit address
f = 5 bit file register address

Instruction-Binary	(Hex)	Name	Mnemonic,Operands	Operation	Status Affected	Notes
0100 bbbf ffff	4bf	Bit Clear f	BCF f,b	0 → f(b)	None	2,4
0101 bbbf ffff	5bf	Bit Set f	BSF f,b	1 → f(b)	None	2,4
0110 bbbf ffff	6bf	Bit Test f,Skip if Clear	BTFSC f,b	Test bit (b) in file (f): Skip if clear	None	
0111 bbbf ffff	7bf	Bit Test f, Skip if Set	BTFSS f,b	Test bit (b) in file (f): Skip if set	None	

LITERAL AND CONTROL OPERATIONS

(11-8)	(7 - 0)
OPCODE	k (LITERAL)

k = 8 bit immediate value

Instruction-Binary	(Hex)	Name	Mnemonic,Operands	Operation	Status Affected	Notes
1110 kkkk kkkk	Ekk	AND Literal and W	ANDLW k	k & W → W	Z	
1001 kkkk kkkk	9kk	Call subroutine	CALL k	PC + 1 → Stack, k → PC	None	1
0000 0000 0100	004	Clear Watchdog timer	CLRWDT -	0→WDT (and prescaler,if assigned)	TO, PD	
101k kkkk kkkk	Akk	Go To address(k is 9 bit)	GOTO k	k → PC (9 bits)	None	
1101 kkkk kkkk	Dkk	Incl. OR Literal and W	IORLW k	k v W → W	Z	
1100 kkkk kkkk	Ckk	Move Literal to W	MOVLW k	k → W	None	
0000 0000 0010	002	Load OPTION register	OPTION -	W → OPTION register	None	
1000 kkkk kkkk	8kk	Return,place Literal in W	RETLW k	k → W, Stack → PC	None	
0000 0000 0011	003	Go into standby mode	SLEEP -	0 → WDT, stop oscillator	TO, PD	
0000 0000 0fff	00f	Tristate port f	TRIS f	W → I/O control register f	None	3
1111 kkkk kkkk	Fkk	Excl. OR Literal and W	XORLW k	k ⊕ W → W	Z	

PASM Directives			Function
label	=	value	Assign value to label
label	EQU	value	Assign value to label
	ORG	address	Set origin to address
	DS	bytes	Define space of bytes
	DEVICE	x,x,x,x	Define device (see PIC.EQU)
	ID	value16	Assign value16 to device id
	ID	CHECKSUM	Assign computed checksum to device id
	INCLUDE	'file'	Include file into source
	RESET	addr9	Assemble into last location JMP addr9
	END		End assembly (optional)

PASM Pre-Defined Symbols			Description
(Symbols used with DEVICE directive - choose one from each group			
DEVICE PIC16C54,XT_OSC,WDT_OFF,PROTECT_OFF)			
PIC16C54	=	0	Devices
PIC16C55	=	1	
PIC16C56	=	2	
PIC16C57	=	3	
PIC16C58	=	4	
LP_OSC	=	11000000b	Oscillators
XT_OSC	=	11000001b	
HS_OSC	=	11000010b	

```
RC_OSC          =   11000011b

WDT_OFF         =   10110000b       Watchdog status
WDT_ON          =   10110100b

PROTECT_ON      =   01110000b       Code-protect status
PROTECT_OFF     =   01111000b

(Special file register equates)
INDIRECT        =   0               Indirect data addressing
RTCC            =   1               Real time clock/counter register
PC              =   2               Program counter
STATUS          =   3               Status word register
FSR             =   4               File sclect register
RA              =   5               Port A I/O register
RB              =   6               Port B I/O register
RC              =   7               Port C I/O register

(Status register bit equates)
C               =   STATUS.0        Carry bit
DC              =   STATUS.1        Digit carry bit
Z               =   STATUS.2        Zero bit
PD              =   STATUS.3        Power down bit
TO              =   STATUS.4        Time out bit
PA0             =   STATUS.5        Page address bit 0
PA1             =   STATUS.6        Page address bit 1
PA2             =   STATUS.7        Page address bit 2
```

ADD	fr,#lit	2	CSNE	fr,#lit	3	MOVB	bit1,/bit2	
ADD	fr,W		CSNE	fr1,fr2	3	MOVSZ	W,++fr	
ADD	fr1,fr2	2	DEC	fr		MOVSZ	W,--fr	
ADD	W,fr		DECSZ	fr		NEG	fr	
ADDB	fr,bit	2	DJNZ	fr,addr9	2	NOP		
AND	fr,#lit	2	IJNZ	fr,addr9	2	NOT	fr	
AND	fr,W		INC	fr		NOT	W	
AND	fr1,fr2	2	INCSZ	fr		OR	fr,#lit	
AND	W,#lit		JB	bit,addr9	2	OR	fr,W	
AND	W,fr		JC	addr9	2	OR	fr1,fr2	
CALL	addr8		JMP	addr9		OR	W,#lit	
CJA	fr,#lit,adr9	4	JMP	PC+W		OR	W,fr	
CJA	fr1,fr2,adr9	4	JMP	W		RET		
CJAE	fr,#lit,adr9	4	JNB	bit,addr9	2	RETW	lit,lit,...	
CJAE	fr1,fr2,adr9	4	JNC	addr9	2	RL	fr	
CJB	fr,#lit,adr9	4	JNZ	addr9	2	RR	fr	
CJB	fr1,fr2,adr9	4	JZ	addr9	2	SB	bit	
CJBE	fr,#lit,adr9	4	LCALL*	addr	1-3	SC		
CJBE	fr1,fr2,adr9	4	LJMP *	addr	1-3	SETB	bit	
CJE	fr,#lit,adr9	4	LSET *	addr	0-2	SKIP		
CJE	fr1,fr2,adr9	4	MOV	fr,#lit	2	SLEEP		
CJNE	fr,#lit,adr9	4	MOV	fr,W		SNB	bit	
CJNE	fr1,fr2,adr9	4	MOV	fr1,fr2	2	SNC		
CLC			MOV	OPTION,#lit	2	SNZ		
CLR	fr		MOV	OPTION,fr	2	STC		
CLR	W		MOV	OPTION,W		STZ		
CLR	WDT		MOV	!port_fr,#lit	2	SUB	fr,#lit	

CLRB	bit		MOV	!port_fr,W		SUB	fr,W
CLZ			MOV	!port_fr1,fr2	2	SUB	fr1,fr2
CSA	fr,#lit	3	MOV	W,#lit		SUBB	fr,bit
CSA	fr1,fr2	3	MOV	W,fr		SWAP	fr
CSAE	fr,#lit	3	MOV	W,/fr		SZ	
CSAE	fr1,fr2	3	MOV	W,fr-w		TEST	fr
CSB	fr,#lit	3	MOV	W,++fr		TEST	W
CSB	fr1,fr2	3	MOV	W,--fr		XOR	fr,#lit
CSBE	fr,#lit	3	MOV	W,<<fr		XOR	fr,W
CSBE	fr1,fr2	3	MOV	W,<>fr		XOR	fr1,fr2
CSE	fr,#lit	3	MOV	W,>>fr		XOR	W,#lit
CSE	fr1,fr2	3	MOVB	bit1,bit2	4	XOR	W,fr

* These instruction are not avillable in PASMX.

สายเคเบิลของ RS - 232C

การรับส่งข้อมูลแบบอนุกรมของคอมพิวเตอร์หรือที่เรียกว่า RS-232C นั้น ใช้กันมากในการรับส่งข้อมูลระหว่างคอมพิวเตอร์กับคอมพิวเตอร์ คอมพิวเตอร์กับโมเด็ม คอมพิวเตอร์กับอุปกรณ์ต่อพ่วงแบบต่าง ๆ เช่น เมาส์ เครื่องวาดภาพ (Plotter) เครื่องพิมพ์บางชนิดที่ใช้พอร์ตอนุกรม รวมทั้งอุปกรณ์วัดสัญญาณต่าง ๆ ทางวิทยาศาสตร์ก็มักรับส่งข้อมูลกับคอมพิวเตอร์ผ่านทาง RS - 232C นี้ การส่งข้อมูลแบบอนุกรมจึงมักเป็นมาตรฐานที่ใช้กันกว้างขวางวิธีหนึ่ง

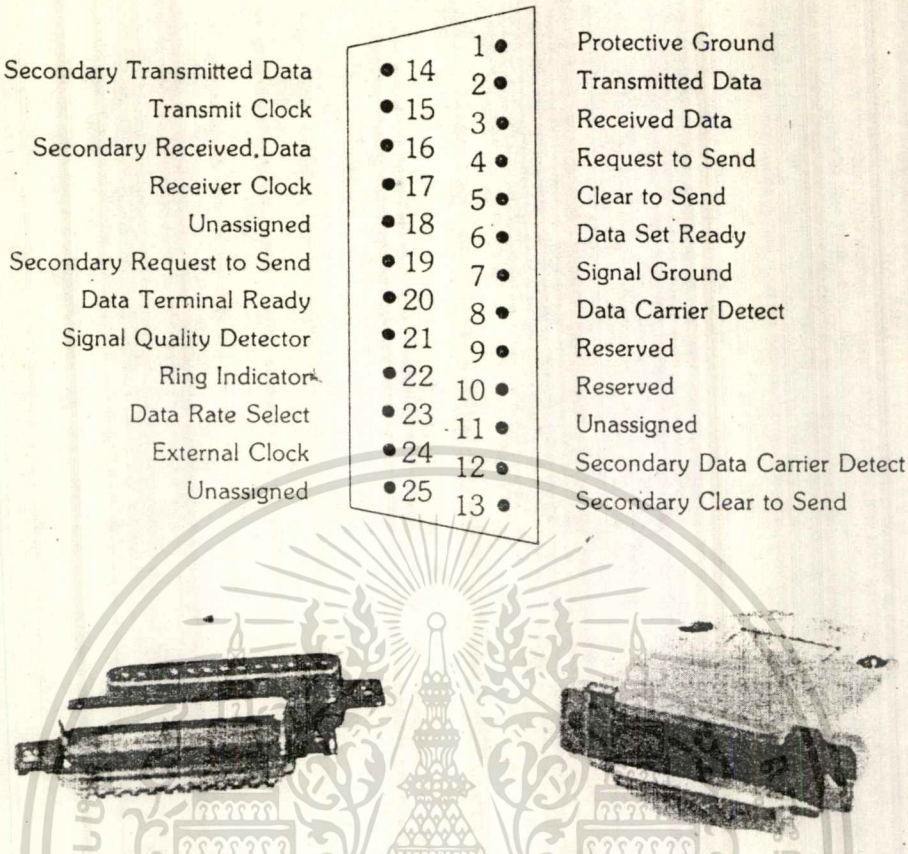
มาตรฐานของการรับส่งข้อมูลแบบอนุกรม (RS - 232C) นี้ได้มีการกำหนดขึ้นมาเพื่อให้คอมพิวเตอร์ต่างยี่ห้อกัน หรืออุปกรณ์ต่อพ่วงแต่ละชนิดรับส่งข้อมูลกันได้ เมื่อทำตามมาตรฐานนี้ โดยไม่สนใจว่าอุปกรณ์หรือคอมพิวเตอร์นั้นจะผลิตมาจากที่ใด โดยมีการกำหนดรายละเอียดในการรับส่งข้อมูล เช่น ข้อต่อ (Connector) ที่ใช้เป็นแบบใด มีสัญญาณที่ใช้กี่เส้น แต่ละสัญญาณทำหน้าที่อะไร และใช้ระดับแรงดันไฟฟ้าเท่าไรในการรับส่งข้อมูล ความเร็วในการรับส่งข้อมูลจะเป็นเท่าใดบ้าง ใช้ข้อมูลกี่บิตในการรับส่งข้อมูล ฯลฯ อุปกรณ์หรือคอมพิวเตอร์ก็จะทำตามมาตรฐานนี้ ทำให้สามารถรับส่งข้อมูลได้อย่างไม่มีปัญหา

ขาต่าง ๆ ของ RS-232C

เริ่มจากหัดข้อต่อ (Connector) ระหว่างสายเคเบิลทั้งสองปลาย จะใช้ข้อต่อแบบ 25 Pin รูปร่างหน้าตัดคล้ายตัว " D " มีชื่อเรียกว่า DB-25 ดังแสดงในรูปที่ 1

กำหนดการใช้งานเอาไว้ทั้งหมด 22 ขา ไม่ได้ใช้ 3 ขา สัญญาณแต่ละขาจะทำหน้าที่ของมันตามที่กำหนดเอาไว้ แต่ปกติแล้วในการรับส่งข้อมูลทั่วไปเราใช้สัญญาณเพียง 8 ถึง 9 เส้นเท่านั้นก็พอ สัญญาณที่เหลือเราไม่นำมาใช้ เนื่องจากว่าบางเส้นเป็นสัญญาณรับส่งข้อมูล และสัญญาณควบคุมของช่องสัญญาณสำรอง (Secondary Channel) บางเส้นปล่อยว่างไว้ และบางเส้นใช้สำหรับงานพิเศษบางอย่างเท่านั้น

สายเคเบิลที่ใช้รับส่งข้อมูลส่วนมากจึงใช้สายเพียง 8 ถึง 9 เส้นเท่านั้นจากข้อต่อ 25 ขาสัญญาณแต่ละเส้นเรียงตามลำดับดังนี้คือขาที่ 1,2,3,4,5,6,7,8 และ 20 กับ 22 โดยที่ขาที่ 1 (Protective Ground) นั้น มักจะไม่จำเป็นต้องต่อใช้งาน จึงเหลือ



รูปที่ 1 แสดงข้อต่อแบบ DB - 25 และขาต่าง ๆ

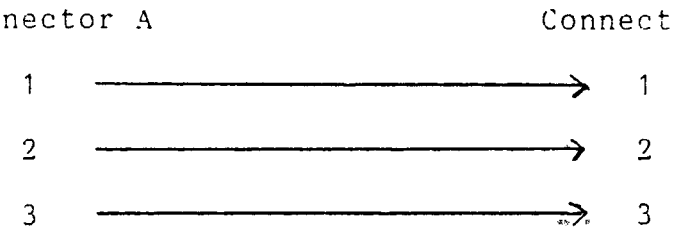
จานวนสายที่ใช้เพียง 9 เส้น หน้าที่ของสัญญาณแต่ละเส้นก็คือ

- ขาที่ 1 (Protective Ground) เป็นสายดินของอุปกรณ์
- ขาที่ 2 (Transmitted Data) ใช้สาหรับส่งข้อมูล
- ขาที่ 3 (Received Data) ใช้สาหรับรับข้อมูล
- ขาที่ 4 (Request to Send) เป็นสัญญาณขอทาการส่งข้อมูล
- ขาที่ 5 (Clear to Send) เป็นสัญญาณตอบรับว่าเริ่มส่งข้อมูลได้
- ขาที่ 6 (Data Set Ready) เป็นสัญญาณแสดงว่าตัวรับพร้อมที่จะรับข้อมูลแล้ว
- ขาที่ 7 (Signal Ground) เป็นสายดินของสัญญาณรับส่ง
- ขาที่ 8 (Data Carrier Detect) เป็นตัวบอกว่าทั้งตัวรับและตัวส่งต่อถึงกันเรียบร้อยแล้ว และพร้อมที่จะทาการรับส่งข้อมูล ในกรณีที่ใช้ต่อกับโมเด็ม ขา

DB-9 Pin	DB-25 Pin	Assignmeng/Function
1	8	Carrier detect
2	3	Receive data
3	2	Transmit data
4	20	Data terminal ready
5	7	Signal Ground
6	6	Data set ready
7	4	Request to send
9	22	Ring indicator

รูปที่ 2 การใช้งานรับส่งข้อมูลอนุกรม เราใช้สัญญาณเพียง 9 ขาเท่านั้น

สายเคเบิลของการรับส่งข้อมูลอนุกรมแบ่งออกได้เป็นสองแบบ คือสายตรงและสายสลับ ที่ต้องมีสายสองแบบนี้ก็เพราะว่าการเชื่อมต่อส่งข้อมูลมีสองกรณีคือ คอมพิวเตอร์ต่อกับคอมพิวเตอร์และคอมพิวเตอร์ต่อเข้ากับอุปกรณ์ต่าง ๆ เมื่อเราต่อคอมพิวเตอร์เข้ากับคอมพิวเตอร์เพื่อรับส่งข้อมูลกัน สายสัญญาณรับส่งข้อมูลต้องสลับไขว้กัน เพื่อให้สัญญาณส่งของตัวแรกไปเข้าสัญญาณรับของตัวที่สอง เราจึงเรียกสายเคเบิลแบบนี้ว่าสายสลับ ส่วนการต่อคอมพิวเตอร์เข้ากับอุปกรณ์ต่อพ่วงนั้น สายสัญญาณของอุปกรณ์ต่อพ่วง เช่น โมเด็ม และพล็อตเตอร์ (plotter) มักจะสลับสัญญาณรอรับไว้ภายในแล้ว สายเคเบิลจากเครื่องคอมพิวเตอร์จึงต่อตรงเข้าแต่ละเส้นของอุปกรณ์ได้เลย เราจึงเรียกสายเคเบิลแบบนี้ว่าสายตรง กรณีที่วงจรของอุปกรณ์ต่อพ่วงไม่ได้สลับสายไว้ภายใน เราก็ต้องใช้สายสลับต่อระหว่างคอมพิวเตอร์กับอุปกรณ์นั้น ไม่จำเป็นต้องใช้สายตรงเสมอไป



นี้จะเป็นตัวบอกว่าโมเด็มทั้งสองด้านต่อถึงกันได้แล้วโดยมีสัญญาณ Carrier ส่งถึงกัน

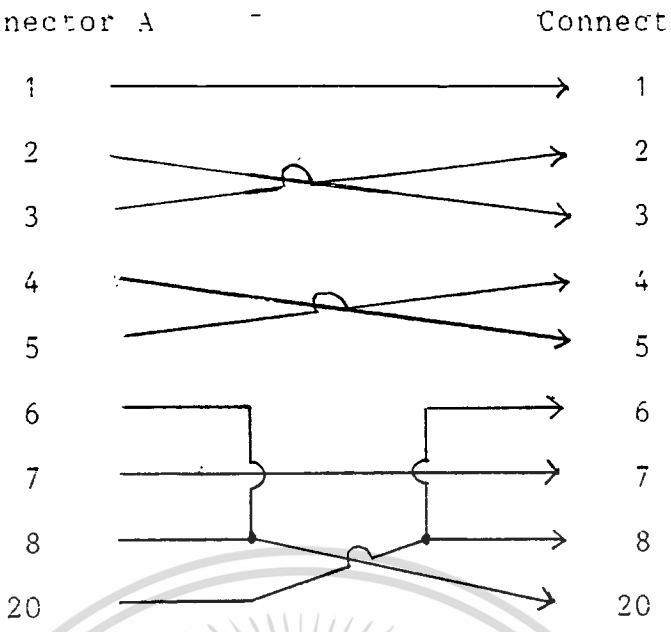
- ขาที่ 20 (Data Tominal Ready) เป็นสัญญาณแสดงว่าตัวส่งพร้อมที่จะส่งข้อมูล

- ขาที่ 22 (Ring Indicator) เป็นขาแสดงแทนกริ่งโทรศัพท์ที่เรียกเข้ามา การเชื่อมต่อบางอย่างก็จะไม่ใช้ขาที่ 22 นี้ในการทำงาน

ส่วนขาอื่น ๆ ที่เหลือนั้น ส่วนมากมีหน้าที่คล้ายกับ 8 ขาแรกที่กล่าวมา และบางเส้นใช้กับงานพิเศษเท่านั้นจึงไม่กล่าวในที่นี้

DB25 และ DB29

จากการที่ข้อต่อแบบ 25 ขาเราใช้งานจริงเพียง 9 ขาเท่านั้นเครื่องคอมพิวเตอร์รุ่นใหม่ ๆ จึงได้ลดข้อต่อลงมาใช้แบบ 9 ขาแทน ซึ่งเราเรียกข้อต่อแบบนี้ว่า DB-9 การใช้ข้อต่อแบบ DB-9 นี้มีข้อดีหลายอย่างคือ ขนาดเล็กกะทัดรัด ราคาของข้อต่อถูกกว่า การต่อสายเคเบิลสะดวกขึ้น และการใช้งานคล่องตัวกว่า DB-25 สัญญาณต่าง ๆ ของข้อต่อแบบ DB-9 บางเส้นจะตรงกับที่ใช้ใน DB-25 ดังที่แสดงในตารางเปรียบเทียบ เครื่องคอมพิวเตอร์แบบไอบีเอ็มเอทีและรุ่นใหม่ ๆ มักจะใช้ข้อต่อแบบ DB-9 สำหรับรับส่งข้อมูลอนุกรมทั้งนั้น แต่อุปกรณ์ต่อพ่วงส่วนมากยังคงใช้ข้อต่อแบบ DB-25 อยู่ เราจึงต้องใช้สายเคเบิลที่เหมาะสมสำหรับทั้งสองด้านในการรับส่งข้อมูล



รูปที่ 4 การต่อสายสลับของ RS-232C หรือ Null Modem Cable

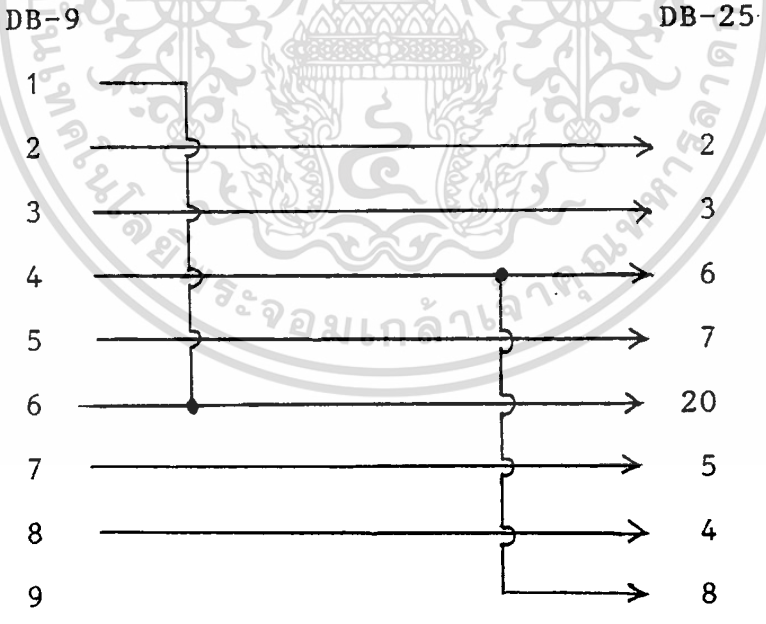
สายสลับนี้มีชื่อเรียกอีกอย่างหนึ่งว่า Null Modem Cable ซึ่งหมายถึงการเชื่อมต่อระหว่างคอมพิวเตอร์สองเครื่องโดยไม่ผ่านโมเด็มนั่นเอง ข้อดีของการใช้สายเคเบิลส่งข้อมูลไม่ผ่านโมเด็มก็คือเราสามารถรับส่งข้อมูลด้วยความเร็วสูงสุดถึง 9600 บิตต่อวินาที หรือ 19200 บิตต่อวินาที ระหว่างเครื่องคอมพิวเตอร์ได้ในระยะทางใกล้ ๆ ซึ่งสะดวกรวดเร็วกว่าการส่งข้อมูลผ่านโมเด็มมาก เช่น ใช้ในการรับส่งไฟล์ระหว่าง LAPTOP กับเครื่องตั้งโต๊ะ เป็นต้น

ส่วนการต่อระหว่างข้อต่อแบบ DB-25 ไปยังข้อต่อแบบ DB-9 นั้น เราก็เปรียบเทียบกับการต่อในแบบ DB-25 กับ DB-25 โดยดูจากชื่อของสัญญาณที่ต่อเข้าหากันเป็นหลัก จะต่อตามเบอร์ของแต่ละขาเข้าด้วยกันตรง ๆ ไม่ได้ สายเคเบิลแบบสายตรงจาก DB-25 ไป DB-9 จะเป็นดังรูปที่ 5

และสายแบบสลับจากข้อต่อ DB-9 เข้ากับ DB-25 เป็นดังในรูปที่ 6

DB-25 Assignment	Pin		Pin	DB-9 Assignment
carrier detect	8	→	1	carrier detect
Recieve data	3	→	2	Receive data
Transmit data	2	→	3	Transmit data
Data terminal ready	20	→	4	Data terminal ready
Signal ground	7	→	5	Signal ground
Data set ready	6	→	6	Data set ready
Request to send	4	→	7	Request to send
Clear to send	5	→	8	Clear to send
Ring indicator	22	→	9	Ring indicator

รูปที่ 5 การต่อสายตรงจาก DB-25 ไปยัง DB-9



รูปที่ 6 การต่อสายสลับจาก DB-9 ไปยัง DB-25

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <bios.h>
#include <dos.h>
#include <dir.h>
#include <stdlib.h>
#include <alloc.h>
#include <ctype.h>
#include <alloc.h>
#include <mem.h>

#define  BKSP          8
#define        ENTER   13
#define ESC            27
#define UP                     72
#define END            79
#define DOWN     80
#define  DELETE 83
//others
#define  MAXDIG   4
#define  TRUE          1
#define  FALSE         0
#define ERROR          -1
#define  NOERROR       1
#define COM1PORT (0x03F8)
#define COM2PORT (0x02F8)

#define BUF (0)
#define DLL (0)
#define IER (1)
#define DLH (1)
#define IIR (2)
#define LCR (3)
#define MCR (4)
#define LSR (5)
#define MSR (6)

#define CommBufSize (20)

#define NOTPASS (2)
#define PASS (1)
#define FAIL (0)
#define NOTFOUND (-1)
#define ON       (1)
#define OFF      (0)

#define EOI (0x20)

#define COM1 (1)
#define COM2 (2)

#define IRQ4FLAG (0x10) /* INT 0x0C */
#define IRQ3FLAG (0x08) /* INT 0x0B */

#define COM1INT (0x0C)
#define COM2INT (0x03)

#define IMR (0x21)

#define B1200 (0)
#define B2400 (1)
#define B4800 (2)
#define B9600 (3)
#define B19200 (4)
```

```
#define S1 (0)
#define S2 (1)

#define PMONB (0)
#define PODD (1)
#define PSVSN (2)

#define TIMEOUT (1) /* No response from SX */
#define B_COMM (2)
#define B_BAUD (3)
#define B_DATABIT (4)
#define B_STOPBIT (5)
#define B_PARITY (6)
#define B_MEMORY (7)
#define B_INSTALLED (8)
#define B_RANGE (9)
#define EMPTY (10)
#define FMT_HEAD (11)
#define FMT_FDBS (12)
#define FMT_DC3 (13)
#define FMT_FTER (14)
#define B_LRC (15)

#define DISABLE (0)
#define ENABLE (1)

#define TXCOUNT (32767)

#define START_BYTE (0x02)
#define  STOP_BYTE (0xFF)

//#define  NOTFOUND -2

void background(char ch,int startx,int endx,int starty,int endy,int attribute);
void border(int code,int startx,int endx,int starty,int endy,int attribute);
void puts_center(int stline,char *text,int startx,int endx,int starty,int attribute1,int attribute2);
void restore_video(int startx,int endx,int starty,int endy,unsigned char *buffer);
void save_video(int startx,int endx,int starty,int endy,unsigned char *buffer);
void shadow(int startx,int endx,int starty,int endy);
void write_x_y_axis(int mode,char ch,int start,int end,int start_axis,int attribute);
void write_char(int x,int y,char ch,int attribute);
void write_menu(int startx,int starty,char *text,int attribute1,int attribute2);
void sizecursor(int start,int end);
void tablemain(int mode);
void pagmain(int);
void pagedis(int,int,int,int);
void namedis(int,int,int,int);
void DoTable(int,int,int,int,int,int,int);
void Do_Choice(int,int,int,int,int,int,int,int,int);
void indatmain(int mode);
void name_trans(void);
void list_num(void);
void list_name(void);
void input_dat(void);
void del_dat(void);
void setclock(void);

int Rskmain(char *,char *);
int do_choice(int);
int get_resp(int x,int starty,char *menu[],char *keys,int count,int attribute1,int attribute2,int shtkey_att1,int shtkey_att2);
int is_in(char *,char);
int popup(char *menu[],char *keys,int count,int startx,int starty,int attribute1,int attribute2,int shtkey_att1,int shtkey_att2);
int video_mode(void);
```

```
struct data{
            char name[20];
            char srename[20];
            char address[50];
            char tel[10];
            char number[7];
            }dat;

struct time now;
union inkey{
            char cha[2];
            int i;
            }c;

FILE  *fp;
char tem[200];

char far *vid_mem;
char *main_menu[]={
       "%Call user with his name   ",
       "%List user name            ",
       "C%all user with  his number",
       "L%ist user number          ",
       "I%nput new user data       ",
       "%Delete record             ",
       "%Setclock                  ",
       "%Quit                      ",
       };
char *mainkey="claindsq";

//char mass[50];
char count;
//void no_trans(void);

void main(void)
{
       int dochoice;
       int vmode;
       int cont=8;
       vmode=video_mode();
       if((vmode!=2)&&(vmode!=3)&&(vmode!=7))
       {
            printf("Video is'n in 80 text mode. Video error.");
            exit(1);
       }
       if(vmode==7) vid_mem=(char far *) 0xB0000000;
       else vid_mem=(char far *) 0xB8000000;
       do
            {
       _setcursortype(_NOCURSOR);
       background('%',0,79,0,24,0x1F);
       dochoice = popup(main_menu,mainkey,8,20,8,0x1F,0x71,0x12,0x74);
       cont=   do_choice(dochoice);
            }while(cont!=7);
}
void shadow(startx,endx,starty,endy)
int startx,endx,starty,endy;
{
       int x,y;
       char far *v;
       for(x=startx;x<=endx;x++)
       {
            v=vid_mem;
            v+=x*2+endy*160;
            v++;
            *v=0x08;
```

```
        endx--; //this is solve problem double in y axis
        for(y=starty;y<=endy;y++)
        {
                v=vid_mem;
                v+=endx*2+y*160;
                v++;
                *v++=0x08;
                v++;
                *v=0x08;
        }
}
void background(char ch,int startx,int endx,int starty,
int endy,int attrib)
{
        char far *v;
        int i,j;
        for(i=starty;i<=endy;i++)
                for(j=startx;j<=endx;j++)
                {
                        v=vid_mem;
                        v+=(j*2+i*160);
                        *v++=ch;
                        *v=attrib;
                }
}
/*****code:1==all single line*****/
/*****code:2==all double line*****/
/*****code:3==top double line*****/
/*****code:4==bottom double line*****/
/*****code:5==side and bottom double line*****/
/*****code:6==side and top double line*****/
/*****code:7==left and top double line*****/
/*****code:8==left and bottom double line*****/
/*****code:9==right and top double line*****/
/*****code:10==right and bottom double line*****/
/*****code:11==top and bottom double line*****/
void border(code,startx,endx,starty,endy,attrib)
int code,startx,endx,starty,endy,attrib;
{
        char far *v;
        int x,y;
        char side_left=179,side_right=179,top=196,bottom=196;
        char top_left=218,bottom_left=192,top_right=191,bottom_right=217;
        switch (code)
        {
                case 1 : break;
                case 2 :
                        {
                                side_left=186;side_right=186;top=205;bottom=205;
                                top_left=201;bottom_left=200;top_right=187;bottom_right=188;
                        } break;
                case 3 :
                        {
                                side_left=179;side_right=179;top=205;bottom=196;
                                top_left=213;bottom_left=192;top_right=184;bottom_right=217;
                        } break;
                case 4 :
                        {
                                side_left=179;side_right=179;top=196;bottom=205;
                                top_left=218;bottom_left=212;top_right=191;bottom_right=190;
                        } break;
                case 11:
                        {
                                side_left=179;side_right=179;top=205;bottom=205;
                                top_left=213;bottom_left=212;top_right=184;bottom_right=190;
```

```
                default : break;
        }
        x=startx;
        for(y=starty;y<=endy;y++)
        {
                v=vid_mem;
                v+=(x*2+y*160);
                *v++=side_left;
                *v=attrib;
        }
        x=endx;
        for(y=starty;y<=endy;y++)
        {
                v=vid_mem;
                v+=(x*2+y*160);
                *v++=side_right;
                *v=attrib;
        }
        y=starty;
        for(x=startx;x<=endx;x++)
        {
                v=vid_mem;
                v+=(x*2+y*160);
                *v++=top;
                *v=attrib;
        }
        y=endy;
        for(x=startx;x<=endx;x++)
        {
                v=vid_mem;
                v+=(x*2+y*160);
                *v++=bottom;
                *v=attrib;
        }
        write_char(startx,starty,top_left,attrib);
        write_char(startx,endy,bottom_left,attrib);
        write_char(endx,starty,top_right,attrib);
        write_char(endx,endy,bottom_right,attrib);
}
int is_in(s,c)//test hot key or special key
char *s,c;
{
        int i;
        for(i=0;*s;i++)
                if(*s++==c) return i+1;
        return 0;
}
int get_resp(x,starty,menu,keys,count,attrib1,attrib2,shtkey_att1,shtkey_att2)
int x,starty;
char *menu[];
char *keys;
int count,attrib1,attrib2,shtkey_att1,shtkey_att2;
{
        union inkey {
                char ch[2];
                int i;
                } c;
        int i,y,arrow_choice=0,key_choice;

        y=starty+1;
        for(i=0;i<count;i++,y++)
                write_menu(x,y,menu[i],attrib1,shtkey_att1);
        y=starty+1;
        write_menu(x,y,menu[0],attrib2,shtkey_att2);
```

```
    {
        while(!bioskey(1));
        c.i=bioskey(0);
        if(c.ch[0])
        {
            key_choice=is_in(keys,tolower(c.ch[0]));
            if(key_choice) return key_choice - 1;

            switch(c.ch[0])
            {
                case '\r' : return arrow_choice;
                case BSC  : return -1;
            }
        }
        else
        {
            switch(c.ch[1])
            {
                case UP :
                    {
                        write_menu(x,y+arrow_choice,menu[arrow_choice],
                            attrib1,shtkey_att1);
                        arrow_choice--;
                    } break;
                case DOWN :
                    {
                        write_menu(x,y+arrow_choice,menu[arrow_choice],
                        attrib1,shtkey_att1);
                        arrow_choice++;
                    } break;
            }
        }
        if(arrow_choice==count) arrow_choice=0;
        if(arrow_choice < 0) arrow_choice=count-1;
        write_menu(x,y+arrow_choice,menu[arrow_choice],attrib2,shtkey_att2);
    }
}
int popup(menu,keys,count,startx,starty,attrib1,attrib2,shtkey_att1,shtkey_att2)
char *menu[];
char *keys;
int count;
int startx,starty,attrib1,attrib2,shtkey_att1,shtkey_att2;
{
    int len=strlen(menu[0]);
    int endx,endy,choice;
    unsigned char *p;
    if( (startx<0)||(startx>79)||(starty<0)||(starty>24) )
    {
        printf("Range of popup menu error");
        return ERROR;
    }
    endx=len+startx;
    endy=count+starty+1;
    if((endx>79)||(endy>24))
    {
        printf("Menu won't fit");
        return ERROR;
    }
    p=(unsigned char *) farmalloc(2*((endx-startx+2)*(endy-starty+2)));
    if(!p) return ERROR;
//  save_video(startx,endx+1,starty,endy+1,p);

    /***** +1 because Shadow menu*****/
    background(' ',startx,endx,starty,endy,attrib1);
    shadow(startx+1,endx+1,starty+1,endy+1);
```

```
        border(1,startx,endx,starty,endy,attrib1);//send code to border
        choice=get_resp(startx+1,starty,menu,keys,count,attrib1,
                        attrib2,shtkey_att1,shtkey_att2);
//      restore_video(startx,endx+1,starty,endy+1,p);
        farfree(p);
        return choice;
}
void sizecursor(start,end)
int start,end;
{
        union REGS r;
        r.h.ah=1;
        r.h.ch=start;
        r.h.cl=end;
        int86(0x10,&r,&r);
}
void restore_video(startx,endx,starty,endy,buf_ptr)
int startx,endx,starty,endy;
unsigned char *buf_ptr;
{
        int i,j;
        char far *v;

        for(i=starty;i<=endy;i++)
                for(j=startx;j<=endx;j++)
                {
                        v=vid_mem;
                        v+=(j*2+i*160);
                        *v++=*buf_ptr++;
                         *v =*buf_ptr++;
                }
}
void save_video(startx,endx,starty,endy,buf_ptr)
int startx,endx,starty,endy;
unsigned char *buf_ptr;
{
        int i,j;
        char far *v;

        for(i=starty;i<=endy;i++)
                for(j=startx;j<=endx;j++)
                {
                        v=vid_mem;
                        v+=(j*2+i*160);
                        *buf_ptr++=*v++;
                        *buf_ptr++=*v;
                }
}
int video_mode()
{
        union REGS r;
        r.h.ah=15;
        return int86(0x10,&r,&r)&255;
}
void write_char(x,y,ch,attrib)
int x,y,attrib;
char ch;
{
        char far *v;
        v=vid_mem;
        v+=(x*2+y*160);
        *v++=ch;
        *v=attrib;
}
void write_menu(x,y,str,attrib1,attrib2)
int x,y;
```

```
int attrib1,attrib2;
{
        int i;
        char far *v;
        v=vid_mem;
        v+=x*2+y*160;
        for(i=x;*str;i++)
        {
                if(*str=='\')
                {
                        str++;
                        *v++=*str++;
                        *v++=attrib2;
                }
                else
                {
                        *v++=*str++;
                        *v++=attrib1;
                }
        }
}
void write_x_y_axis(mode,ch,start,end,axis_start,attrib)//calculate position
int mode;
char ch;
int start,end,axis_start,attrib;
{
        char far *v;
        int i;
        if(mode==1)
        {       //write x_axis at starty
                for(i=start;i<=end;i++)
                {
                        v=vid_mem+(i*2+axis_start*160);
                        *v++=ch;
                        *v=attrib;
                }
        }
        else if(mode==2)
        { //write y_axis at startx
                for(i=start;i<=end;i++)
                {
                        v=vid_mem+(axis_start*2+i*160);
                        *v++=ch;
                        *v=attrib;
                }
        }
}
int do_choice(no_choice)
int no_choice;

{
        int startx= 5,endx=70,starty=3,endy=21;
        unsigned char *q;

        q=(unsigned char *) farmalloc(2*((endx-startx+2)*(endy-starty+2)));
        if(!q) return ERROR;
        save_video(startx,endx+1,starty,endy+1,q);//stx,enx+1,sty,eny+1
//      border(11,startx,endx,starty,endy,0x1F);

switch(no_choice)
        {
        case 0:pogmain(1);
                break;
        case 1:tablemain(1);
                break;
```

```
        case 3:tablemain(2);//num
                break;
        case 4:indatamain(2);//input
                break;
        case 5:indatmain(1);//del
                break;
        case 6:setclock();
                break;
    case 7:return 7;
        }
restore_video(startx,endx+1,starty,endy+1,q);
farfree(q);
}

void setclock(void)
{
char  temp;

        gotoxy(2,2);
        printf("ENTER HOUR MINUTE SECOND\n");
        scanf("%c%c%c",&temp,now.ti_min,now.ti_sec);
        now.ti_hour = temp;
        settime(&now);
        gettime(&now);
}

int xx,yy;
int y;
int choice,cur;
int edn;
int check_key(void);
char *name={"NAME"};
char *suen={"SURNAME"};
char *page={"PAGER NUMBER"};
char *headpage={"Enter pager number"};
void writemenu(int,int,int,int);
void outframe(int,int,int,int,int,int,int,int);
void clr(int,int,int,int,int,int,int);

void tablemain(mode)
int mode;
{
int startx =5,endx= 70,starty = 5,endy = 22;
int countkey;

sizecursor(7,8);
clrscr();
if (mode ==2)
       pagedis(startx,endx,starty,endy);
else
       namedis(startx,endx,starty,endy);
}

void namedis(startx,endx,starty,endy)
int startx,endx,starty,endy;
{
int result=strlen(name)+strlen(suen)+startx;
int ftcl,scl,tcl,focl;
int mode =1;

       ftcl=startx;
       scl=24;//      17;
       tcl= 48 ;//41;
       focl=endx;
```

```
        write_menu(startx+32,starty+1,suen,16,16);
        write_menu(result+40,starty+1,page,16,16);
//      x = startx+2;
//for(
        if((fp=fopen("a:\\prog\\calling.nex","r+b")) == NULL)
        {
                printf("ERROR IN OPEN FILE");
                exit(1);
        }
        else
        {
                DoTable(startx,endx,starty,endy,mode,scl,tcl);
        }
}
void pagedis(startx,endx,starty,endy)
int startx,endx,starty,endy;
{
//int result=strlen(name)-strlen(suen)+startx;
int ftcl,scl,tcl,focl;
int mode=2;

        ftcl=startx;
        scl = 17 ;//24;
        tcl = 41 ;//28;
        focl=endx;
        outframe(startx,endx,starty,endy,ftcl,scl,tcl,focl);
        write_menu(startx+2,starty+1,page,16,16);
        write_menu(scl+10,starty+1,name,16,16);
        write_menu(scl+32,starty+1,suen,16,16);
//      x = startx+2;
//for(
        if((fp=fopen("a:\\prog\\calling.nex","r+b")) == NULL)
        {
                printf("ERROR IN OPEN FILE");
                exit(1);
        }
        else
        {
                DoTable(startx,endx,starty,endy,mode,scl,tcl);
        }
}
void DoTable(startx,endx,starty,endy,mode,scl,tcl)
int starty,endy,startx,endx,mode,scl,tcl;
{
int endn,x,y;
int staty,edy,cur;

//      x=startx+2;
        fseek(fp,0,2);
        edn = ftell(fp)/sizeof dat;
        rewind(fp);
//      a = 0;//order struct
        staty=starty+3;//insert
        cur = staty;
        edy=endy-1;
        for(y=staty;y<=edy;y++)
                {
                if(fread(&dat,sizeof dat,1,fp) == 1)
                        {
                        writemenu(mode,startx+2,y,18);
                        }
                }
        fseek(fp,0,0);
        fread(&dat,sizeof dat,1,fp);
```

```
        writemenu(mode,startx+2,cur,16);
        Do_Choice(mode,startx,endx,starty,endy,staty,edy,scl,tcl);
        return;
}
void Do_Choice(mode,startx,endx,starty,endy,staty,edy,scl,tcl)
int starty,endy,startx,endx,staty,edy,mode,scl,tcl;
{
int n=0,cur =staty;
choice = 0;

do{
        do
        {
        while(!bioskey(1));
                c.i=bioskey(0);//read key value
        }while//((((!c.cha[0])&&(c.cha[1]=72)||(c.cha[1]==80))||(c.cha[0]&&(c.cha[1]==13)));
        (!(((!(c.cha[0])&&((c.cha[1]==72)||(c.cha[1]==80)))||(c.cha[0]==13)));
if(!c.cha[0])
        {
        switch(c.cha[1])//sp key
                {
                case 72:if(cur!=staty)//up key
                                        {
                                        fseek(fp,n*sizeof dat,0);
                                        fread(&dat,sizeof dat,1,fp);
                                        writemenu(mode,startx+2,cur,18);//y,attrib
//                                      countkey--;
                                        cur--;
                                        n--;
                                        fseek(fp,n*sizeof dat,0);
                                        fread(&dat,sizeof dat,1,fp);
                                        writemenu(mode,startx+2,cur,16);
                                        break;
                                        }
                                if(cur==staty)
                                        {
                                        if(n==0)//start line
                                                {
                                                printf("\007");
                                                break;
                                                }
                                        else//change page
                                                {
                                                clr(starty,endy,startx,scl,tcl,endx,18);
                                                y=staty;
                                                n--;
                                                fseek(fp,n*sizeof dat,0);
                                                fread(&dat,sizeof dat,1,fp);
                                                writemenu(mode,startx+2,y,16);
                                                y++;
                                                do
                                                        {
                                                        n++;
                                                        fseek(fp,(n)*sizeof dat,0);//read next page
                                                        fread(&dat,sizeof dat,1,fp);
                                                        if(ferror(fp))
                                                                {
                                                                printf("Error in file");
                                                                exit(1);
                                                                }
                                                        writemenu(mode,startx+2,y,18);//printf("%s",dat.name);
                                                        y++;
                                                        }while(y<=edy);
                                                        n-=13;
```

```
            case 80: if(cur!=edy)//down key
                                {
                                fseek(fp,n*sizeof dat,0);
                                fread(&dat,sizeof dat,1,fp);
                                writemenu(mode,startx+2,cur,18);
//                              countkey++;
                                cur++;
                                n++;
                                fseek(fp,n*sizeof dat,0);
                                fread(&dat,sizeof dat,1,fp);
                                writemenu(mode,startx+2,cur,16);
                                break;
                                }
                           if(cur==edy)
                                {
                                if( n==edn)//last struct
                                        {
                                        printf("\007");
                                        break;
                                        }
                                else//last line not last page
                                        {
                                         n++;
                                        n=n-(edy-staty);
                                        clr(starty,endy,startx,scl,tcl,endx,18);
//                                      outframe(startx,endx,starty,endy);
                                        y=starty+3;
                                        do
                                                {
                                                fseek(fp,n*sizeof dat,0);
                                                fread(&dat,sizeof dat,1,fp);
                                                if(ferror(fp))
                                                        {
                                                        printf("Error in file");
                                                        exit(1);
                                                        }
                                                writemenu(mode,startx+2,y,18);
                                                n++;
                                                y++;
                                                }while(y<edy);
                                        fseek(fp,n*sizeof dat,0);
                                        fread(&dat,sizeof dat,1,fp);
                                        writemenu(mode,startx+2,y,16);
//                                      gotoxy(1,edy);//I'm insert you can delete it,it don't effect you program
                                        }
                                }
                           break;
                }
        }
        }while(!(c.ch[0]==13));
        fclose(fp);
        rewind(fp);
}

void outframe(startx,endx,starty,endy,ftcl,scl,tcl,focl)
int startx,endx,starty,endy,ftcl,scl,tcl,focl;
{
int AttBack=20;
char ChrBack=176;
//int result=strlen(name)+strlen(suen)+startx;

background(ChrBack,1,startx-1,1,25,AttBack);
background(ChrBack,endx+1,75,1,25,AttBack);
```

```
    background(ChrBack,startx,endx,endy+1,25,AttBack);
    border(11,startx,endx,starty,endy,18);
    background(' ',startx+1,endx-1,starty+1,endy-1,18);
    //write_menu(startx+10,starty+1,name,16,16);
    //write_menu(startx+32,starty+1,suen,16,16);
    //write_menu(result+40,starty+1,page,16,16);
    write_x_y_axis(2,179,starty+1,endy-1,startx-scl,18);
    write_x_y_axis(2,179,starty+1,endy-1,startx+tcl,18);
    write_char(startx+scl,starty,209,18);
    write_char(startx+scl,endy,207,18);
    write_char(startx+tcl,starty,209,18);
    write_char(startx+tcl,endy,207,18);
    write_x_y_axis(1,196,startx+1,endx-1,starty+2,18);
    write_char(startx,starty+2,195,18);
    write_char(startx+scl,starty+2,197,18);
    write_char(startx+tcl,starty+2,197,18);
    write_char(endx,starty+2,180,18);
    }
    void writemenu(mode,x,y,attr)
    int attr,y,mode;
    {
    switch(mode)
           {
           case 1:
                        write_menu(x,y,dat.name,attr,attr);//printf("%s",dat.name);
                        write_menu(x+24,y,dat.suename,attr,attr);
                        write_menu(x+52,y,dat.number,attr,attr);
                        break;
           case 2:
                        write_menu(x,y,dat.number,attr,attr);//printf("%s",dat.name);
                        write_menu(x+18,y,dat.name,attr,attr);
                        write_menu(x+42,y,dat.suename,attr,attr);
                        break;
           }
    }

    void clr(starty,endy,ftclm,sclm,tclm,fclm,attrib)//calculate position
    int ftclm,sclm,tclm,attrib,starty,endy,fclm;
    {
    int i;

    sclm--;
    tclm--;
    for(i=3;i<endy-starty;++i)
           {
           gotoxy(ftclm+2,starty+i+1);
           write_x_y_axis(1,' ',ftclm+1,sclm-1,starty+i,attrib);
           };
    for(i=3;i<endy-starty;++i)
           {
           gotoxy(sclm+2,starty+i+1);
           write_x_y_axis(1,' ',sclm+1,tclm-1,starty+i,attrib);
           };
    for(i=3;i<endy-starty;++i)
           gotoxy(tclm+2,starty+1+i);
           write_x_y_axis(1,' ',tclm+1,fclm-1,starty+i,attrib);
    }
    //#define  BKSP        8
    //#define       TA3KBY  9
    //#define       ENTER   13
    //#define       SHFTAB  15
    //#define       ESC             27
    //#define       HOME            71
    //#define       UP                      72
    //#define  LEFT     75
```

```
//#define RIGHT  77
//#define  END          79
//#define  DOWN    80
//#define DELETE     83

int check_key(void);
int get_resp(int x,int starty,char *menu[],char *keys,int count,int attribute1,int attribute2,int shtkey_att1,int shtkey_att2);
int is_in(char *,char);
int popup(char *menu[],char *keys,int count,int startx,int starty,int attribute1,int attribute2,int shtkey_att1,int shtkey_att2);
int video_mode(void);
char *headmess={"Message     : "};
char *headadd={"Enter user address"};
char *headtel={"Enter user tel.no"};
char *headcon={"Do you want to send another message  Y/N"};
char *headnum={"Enter pager number"};
char *headname={"Enter user name   :"};
char *headsue={"Enter user suename:"};
char *pager={"PAGER NUMBER:"};
int xx,yy,lin,tempxx;
char tem[200];
int ii=0,tempii=0,tempcon=0;
char cho,ans;
char attr;
int stx,sty,eix,eiy;
int startx=5,endx=70,starty=5,endy=20;
//int xx,yy;//use in display screen

void sayerror(void);
char *do_key(int startx,int starty,int maxchar,int linelength,int Tcolor,int Bcolor);
char far *vidmem={char *}0xB8000000;
void hokeymain(int,int,int,int,int,int);
void num_trans(void);
void name_trans(void);

void hokeymain(int startx,int starty,int maxchar,
               int linelenght,int   tcolor,int bcolor)
{
       char *intext;
       char message[200];

       intext=(char *)malloc(200);
       if(!intext) exit(0);
       intext=do_key(startx,starty,maxchar,linelenght,
                          tcolor,bcolor);
       strcpy(tem,intext);
       free(intext);
       return ;
}
void sayerror(void)
{
       sound(200);delay(100);nosound();
}
char *do_key(int startx,int starty,int maxchar,int linelength,int Tcolor,int Bcolor)
{
       union inkey {
              char ch[2];
              int i;
              } c;
       int i;
       char Inputstring[100]="";
       int leftx=0,curx=0,Stringlen=0;

//     clrscr();
       sizecursor(6,7);
       gotoxy(curx+startx+1,starty+1);
       for(;;)
```

```
{
        while(!bioskey(1)) ;
        c.i=bioskey(0);
        if(c.ch[0])
        {
                switch(c.ch[0])
                {
                        case BSC:
                        {
                                Inputstring[0]=0;       sizecursor(1,0);
                                for(i=0;i<linelength;i++)
                                        write_char(startx+i,starty,' ',3color);
                                return(Inputstring);
                        }
                        case ENTER:
                        {
                                Inputstring[Stringlen]=0;sizecursor(1,0);
                                3color=16;
                                if(Stringlen<linelength)
                                        linelength=Stringlen;
                                for(i=0;i<linelength;i++)
                                        write_char(startx+i,starty,Inputstring[i],3color);
                                return(Inputstring);
                        }
                        case 3KSP:
                        {
                                if(leftx)
                                {
                                        Stringlen--;
                                        leftx--;
                                        for(i=0;i<linelength;i++)
                                                write_char(startx+i,starty,Inputstring[i+leftx],Tcolor);
                                        break;
                                }
                                else
                                {
                                        if(curx)
                                        {
                                                curx--;
                                                write_char(startx+curx,starty,' ',3color);
                                                gotoxy(startx+curx+1,starty+1);
                                                Stringlen--;
                                                break;
                                        }
                                        else
                                        {
                                                sayerror();break;
                                        }
                                }
                        }
                        default:
                        {
                                if(Stringlen<maxchar)
                                {
                                        if(Stringlen>=linelength)
                                        {
                                                Inputstring[Stringlen++]=c.ch[0];
                                                leftx++;
                                                for(i=0;i<linelength;i++)
                                                        write_char(startx+i,starty,Inputstring[i+leftx],Tcolor);
                                        }
                                        else
                                        {
                                                Inputstring[Stringlen++]=c.ch[0];
                                                write_char(startx+curx,starty,c.ch[0],Tcolor);
```

```
                                        curx++;gotoxy(startx+curx+1,starty+1);
                                        break;
                                }
                        }
                        else
                        {
                                sayerror();break;
                        }
                }
        }
    }
}

void pogmain(int mode)

{
//int stx,sty,eix,ety;//use for user menu
int sttr1,attr2;
int vaule;
//char numline[3];

//char temp_name[20],temp_sue[20];
char cho;
char attr;

if(mode==1)
        name_trans();
else if (mode ==2)
        num_trans();
//background('a',startx,endx,starty,endy,18);
}

void num_trans(void)
{
char p_num[7];
char *pt;
//dismain();//draw background
ans = 'Y';
do
        {
        clrscr();
        background(' ',startx,endx,starty,endy,18);
        write_menu(startx+2,starty+1,headnum,16,16);
        xx = startx+2+strlen(headnum)+2;
        yy = starty+1;//ey = starty+1 ex = endx
        dokeymain(xx,yy,7,7,16,17);
        strcpy(p_num,tem);
        if(( fp = fopen("a:\\prog\\calling.nex","r+b"))== NULL)
                {
                printf("ERROR IN OPEN FILE");
                exit(1);
                }
        else
                {
                while((fread(&dat,sizeof dat,1,fp) ==1) && strcmp(p_num,dat.number))
                        {
                        if(ferror(fp))
                                {
                                printf("ERROR IN FILE");
                                exit(1);
                                }
                        }
                if(!strcmp(p_num,dat.number))
                        {
                        write_menu(startx+2,starty+3,name,16,16);
```

```
                write_menu(startx+4+strlen(name),starty+3,dat.name,16,16);
                write_menu(startx+2,starty+5,suen,16,16);
                write_menu(startx+4+strlen(suen),starty+5,dat.suename,18,18);
                write_menu(startx+2,starty+7,headmass,16,16);
                xx = startx+2+strlen(headnum)+2;
                yy = starty+4;//ey = starty+1 ex = endx
                dokeymain(xx,yy,50,80,16,17);     //error strlen
                Rskmain(dat.number,tem);
                }
        else//not error in open file but don't have that num
                {
                gotoxy(22,22);
                printf("DO NOT HAVE THAT NUMBER");
                }
        rewind(fp);
        write_menu(startx+2,starty+9,headcon,148,130);
        ans=getch();
        ans = toupper(ans);
        fclose(fp);
        }
    }while(ans != 'N');
return;
}
void name_trans(void)
{
char temp_name[25],temp_sue[25];
char *pt;
//dismain();//draw background
ans = 'Y';
do
    {
    clrscr();
    background(' ',startx,endx,starty,endy,18);
    write_menu(startx+2,starty+1,headname,16,16);
// call to blink
    lin = 1;
    xx = startx+2+strlen(headsue)+2;
    yy = starty+1;//ey = starty+1 ex = endx
//  cho = 0;
    dokeymain(xx,yy,25,25,16,17);
    strcpy(temp_name,tem);
    write_menu(startx+2,starty+3,headsue,16,16);
    yy=starty+3;
    dokeymain(xx,yy,25,25,16,17);
    strcpy(temp_sue,tem);
    strcpy(temp_name,"eeeee");
    strcpy(temp_sue,"eeee");
    if(( fp = fopen("c:\\reserve\\prog\\calling.nex","r+b"))== NULL)
        {
        printf("ERROR IN OPEN FILE");
        exit(1);
        }
    else
        {
        while((fread(&dat,sizeof dat,1,fp) ==1) && strcmp(temp_name,dat.number) && strcmp(temp_sue,dat.suename))
            {
            if(ferror(fp))
                {
                printf("ERROR IN FILE");
                exit(1);
                }
            }
        if((!strcmp(temp_name,dat.name))&&(!strcmp(temp_sue,dat.suename)))
            {
            write_menu(startx+2,starty+5,pager,16,16);
            write_menu(startx+4+strlen(pager),starty+5,dat.number,16,16);
```

```
//                      gotoxy(startx+4+strlen(suen),starty+3);
//                      printf("%s",dat.suename);
                        write_menu(startx+2,starty+7,headmass,16,16);
                        lin = 1;
                        xx = startx+2+strlen(headmsa)+2;
                        yy = starty+9;//ey = starty+1 ex = endx
                        cho = 0;
                        dokeymain(xx,yy,100,50,16,17);
                        Rskmain(dat.number,tem);
                        }
                else//not error in open file but don't have that num
                        {
                        gotoxy(22,22);
                        printf("DO NOT HAVE THAT NAME");
                        }
                rewind(fp);
                write_menu(startx+2,starty+11,headcon,143,16);
                ans=getch();
                ans = toupper(ans);
                fclose(fp);
                }
        id = 0;
        do
                {
                tem[id]='\0';
                id++;
                }while(id!=99);
        id = 0;
        }while(ans != 'N');
return;
}
int check_key(void);
int get_resp(int x,int starty,char *menu[],char *keys,int count,int attribute1,int attribute2,int shtkey_att1,int shtkey_att2);
char *headindat={"Name (ENTER = QUIT):\n"};
char *comdel={"ARE YOU TO DELETE RECORD Y/N"};
//int xx,yy,lin,tempxx;
//char tem[200];
//int id=0,tempid=0,tempcom=0;
//union inkey
//      {
//      char cha[2];
//      int i;
//      }c;
char cho,ans;
char attr;
int stx,sty,edx,edy;
//int startx=5,endx=70,starty=5,endy=20;
//int xx,yy;//use in display screen

//void sayerror(void);
//char *do_key(int startx,int starty,int maxchar,int linelength,int Tcolor,int Bcolor);
//char far *vidmem=(char *)0xb8000000;
//void dokeymain(int,int,int,int,int,int);
//void num_trans(void);
//void name_trans(void);


void indatmain(int mode)
{
//int stx,sty,edx,edy;//use for user menu
int sttr1,attr2;
int vmode;
//char numline =3;

//char temp_name[20],temp_sue[20];
char cho;
```

```
 if(mode == 1)
        del_dat();
 else if(mode == 2)
        input_dat();
 }

 void input_dat(void)
 {
 //gotoxy(1,1);
 if((fp=fopen("a:\\prog\\calling.nex","a+b")) == NULL)
        {
        printf("ERROR IN OPEN FILE");
        exit(1);
        }
 do
        {
        clrscr();
        background(' ',startx,endx,starty,endy,13);
        border(2,startx,endx,starty,endy,16);
        write_menu(startx+2,starty+1,headindat,16,16);
        xx=startx+4+strlen(headindat);
        yy=starty+1;
        dokeymain(xx,yy,25,25,16,17);
        strcpy(dat.name,tem);
        if(dat.name[0] != '\0')
                {
                yy=starty+4;
                write_menu(startx+2,yy,headsue,16,16);
                xx=startx+2+strlen(headsue);
                dokeymain(xx,yy,25,25,16,17);
                strcpy(dat.suename,tem);
                yy=starty+6;
                write_menu(startx+2,yy,headadd,16,16);
                xx=startx+2+strlen(headadd);
                dokeymain(xx,yy,100,50,16,17);
                strcpy(dat.address,tem);
                yy=starty+8;
                write_menu(startx+2,yy,headtel,16,16);
                xx=startx+2+strlen(headtel);
                dokeymain(xx,yy,10,10,16,17);
                strcpy(dat.tel,tem);
                yy=starty+10;
                write_menu(startx+2,yy,headpage,16,16);
                xx=startx+2+strlen(headpage);
                dokeymain(xx,yy,5,5,16,17);
                strcpy(dat.number,tem);
                fwrite(&dat,sizeof dat,1,fp);
                if(ferror(fp))
                        {
                        printf("ERROR IN WRITE TO FILE");
                        exit(1);
                        }
                }
        }while(dat.name[0] != '\0');
//      printf("\007");
        fclose(fp);
}

void del_dat(void)
{
char temp_name[25],temp_suen[25],temp_num[7];
struct data *pt;
char ans='Y';
int i,n_record,rec;
```

```
if((fp=fopen("a:\\prog\\calling.nex","r+b")) == NULL)
        {
        printf("ERROR IN OPEN FILE");
        exit(1);
        }
do
        {
//       pp=ftp;
//  rec=(fp-pp)/sizeof dat;
        clrscr();
        background(' ',startx,endx,starty,endy,13);
        border(2,startx-1,endx+1,starty-1,endy+1,16);
        write_menu(startx+2,starty+1,headname,16,16);
        //printf("Name (ENTER = QUIT):\n");
        xx=startx+4+strlen(headname);
        yy=starty+1;
        dokeymain(xx,yy,25,25,16,17);
        strcpy(temp_name,tem);
        if(temp_name != '\0')
                {
                yy=starty+4;
                write_menu(startx+2,yy,headsue,16,16);
                xx=startx+2+strlen(headsue);
                dokeymain(xx,yy,25,25,16,17);
                strcpy(temp_suen,tem);
                yy=starty+6;
                write_menu(startx+2,yy,headpage,16,16);
                xx=startx+2+strlen(headpage);
                dokeymain(xx,yy,5,5,16,17);
                strcpy(temp_num,tem);
                rec=0;
                do
                        {
                        if(fread(&dat,sizeof dat,1,fp))//can read or not end file
                                {
                                if(strcmp(dat.name,temp_name)||strcmp(dat.suename,temp_suen))
                                        rec++;//not same
                                }
                        else if(ferror(fp))
                                {
                                gotoxy(22,22);
                                printf("error in file");
                                exit(1);
                                }
                        else if(strcmp(temp_name,dat.name)||strcmp(temp_suen,dat.suename))
                                {
                                gotoxy(22,22);
                                printf("Do not have that name");
                                fclose(fp);
                                return;
                                }
                        }while(strcmp(dat.name,temp_name)||strcmp(dat.suename,temp_suen));//not same
                        fseek(fp,0,2);//end of file
                        n_record=ftell(fp)/sizeof dat;
                        if((pt=malloc(n_record*sizeof dat)) == NULL)
                                {
                                gotoxy(22,22);
                                printf("OUT OF MEMORY");
                                exit(1);
                                }
                        rewind(fp);//start rec
                        for(i=1;i<=n_record-1;++i)
                                {
```

```
                        if(ftell(fp)==(long)(rec)*sizeof dat)
                                fseek(fp,sizeof dat,1);
                        fread(pt,sizeof dat,1,fp);
                        if(ferror(fp))
                                {
                                gotoxy(22,22);
                                printf("error in reading");
                                exit(1);
                                }
                        ++pt;
                        }
                fclose(fp);
                if((fp=fopen("a:\\prog\\calling.nex","w+b"))==NULL)
                        {
                        gotoxy(22,22);
                        printf("error");
                        exit(1);
                        }
                for(i=1;i<=n_record-1;++i)
                 --pt;
                for(i=1;i<=n_record-1;++i)
                        {
                        fwrite(pt,sizeof dat,1,fp);
                        if(ferror(fp))
                                {
                                gotoxy(22,22);
                                printf("error in write file");
                                exit(1);
                                }
                        ++pt;
                        }
                rewind(fp);
                free(pt);
//              }
        gotoxy(22,22);
        printf("do you want to delete another record");
        ans=getchar();
        ans=toupper(ans);
        }
    }while(ans != 'N');
//  printf("\007");
    fclose(fp);
}

#define StaByte  0x0F
#define StoByte  0x0F

char    NodeID;
char    *CommBuf;
int     CommAddr;
int     installed = 0;
unsigned CommDataPos = 0;
unsigned ReceiveFlag = 0; //OFF

unsigned baud[5] = { 96,48,24,12,6 }; /* for 1200,2400,4800,9600 & 19200 bps */
char Data[2] = { 0x03,0x02 };         /* feature bit of 8-bit , 7-bit */
char stop[2] = { 0x00,0x04 };         /* feature bit of 1-bit , 2-bit */
char pari[3] = { 0x00,0x08,0x18 };    /* feature bit of none,odd,even */
unsigned char ch;
char mass[50];
char count =0;
char cnt;
char btran[60];

void interrupt (*old1)(),interrupt (*old2)();
int InitSerialComm(int,int,int,int,int);
```

```
int xmit(unsigned char);
void tran(unsigned char);


void interrupt com1()
{
  CommBuf[ CommDataPos++ ] = inportb( CommAddr );
  ReceiveFlag = ON;
  if ( CommDataPos == CommBufSize )
        CommDataPos = 0;
  outportb( 0x20 , 20I );//80I = 0x20
}
void interrupt com2()
{
  CommBuf[ CommDataPos++ ] = inportb( CommAddr );
  ReceiveFlag = ON;
  if ( CommDataPos == CommBufSize )
        CommDataPos = 0;
  outportb( 0x20 , 80I );
}

/*****************************************************************************

        Procedure : InitSerialComm(...) -

              Parameters :

                    Name        Available selections
                  ============================================================
                  comno       - COM1 or COM2
                  baudrate    - B1200, B2400, B4800, B9600 or B19200
                  databit     - D7 or D8
                  stopbit     - S1 or S2
                  parity      - PNONE, PODD or PEVEN

                Return value : (defined in "serial.h")

                  PASS        - ISR has been installed successfully
                  B_INSTALLED - This comm. interrupt routine has already install.
                  B_COMM      - 'comno' not COM1 or COM2
                  B_BAUD      - 'baudrate' out of range
                  B_DATABIT   - 'databit' out of range
                  B_STOPBIT   - 'stopbit' out of range
                  B_PARITY    - 'parity' out of range
                  B_MEMORY    - Memory not enough

 ***************************************************************************** */
int InitSerialComm(comno,baudrate,databit,stopbit,parity)
int comno,baudrate,databit,stopbit,parity;
{
  char low_divisor,high_divisor,comm_spec;

  if ( installed )
        return( B_INSTALLED );

  if ( comno != COM1 && comno != COM2 )
        return( B_COMM );

  if ( baudrate < B1200 || baudrate > B19200 )
        return( B_BAUD );

  if ( databit != D7 && databit != D8 )
        return( B_DATABIT );
```

```
    if ( parity < PNONE || parity > PEVEN )
          return( E_PARITY );

    switch( comno )
    {
          case COM1 : CommAddr = COM1PORT; break;//COM1PORT=0x03F8
          case COM2 : CommAddr = COM2PORT; break;
    }

    CommBuf = (char *) malloc(CommBufSize);
    if ( CommBuf == NULL )
          return( E_MEMORY );

    low_divisor = baud[baudrate];
    high_divisor = 0;

    comm_spec = Data[databit] | stop[stopbit] | pari[parity];

    outportb( CommAddr + LCR , comm_spec | 0x80 );
    outportb( CommAddr + DLL , low_divisor      );
    outportb( CommAddr + DLH , high_divisor     );
    outportb( CommAddr + LCR , comm_spec & 0x7F );
    outportb( CommAddr + IER , 1                ); /* enable int : Rx Ready */
    outportb( CommAddr + MCR , 0x03             ); /* OUT2(3),RTS(1) */
    inportb( CommAddr + LSR );
    inportb( CommAddr + MSR );
    inportb( CommAddr + IIR );

    switch( comno )
    {
          case COM1 : old1 = getvect( COM1INT );//keep old int vector
                      setvect( COM1INT , com1 );//set new vector
                      outportb( IMR , inportb(IMR) & (~IRQ4FLAG) );
                      break;

          case COM2 : old2 = getvect( COM2INT );
                      setvect( COM2INT , com2 );
                      outportb( IMR , inportb(IMR) & (~IRQ3FLAG) );
                      break;
    }

    installed = comno;
    return( PASS ); //PASS =1
}
/* ****************************************************************************

          QuitSerialComm()

             If InitSerialComm(...) is executed, you will have to call this procedure
               before quitting program.

    **************************************************************************** */

void QuitSerialComm(void)//return int value
{
    if ( ! installed )  //if installed=1 from line 133 installed=comno
          return;

    switch( installed )
    {
          case COM1 : setvect( COM1INT , old1 );
                      outportb( IMR , inportb(IMR) | IRQ4FLAG );
                      break;
```

```
        case COM2 : setvect( COM1INT , old2 );
                                    outportb( IMR , inportb(IMR) | IRQ3FLAG );
                                    break;
    }
    free(CommBuf);
}
void ResetBuffer(void)
{
  CommDataPos=0;
  memset((char *)CommBuf,0,CommBufSize);
}
int xmit(ch)      /* transmit a character via serial port */
unsigned char ch;
{
  int i;

  for ( i=0; i<TXCOUNT; i++ ),/TXCOUNT = 32767
        if ( inportb( CommAddr + LSR ) & 0x20 ) /* THR empty? */ ,//LSR = 5->status if communication line
        {
              outportb( CommAddr , ch );
              return( PASS );
        }
  return(FAIL); /* TXCOUNT too small or Serial Comm. Port error */
}
int Askmain(char *no,char *pass)
{
      int com_status,s_status,i;
      char choice='N';
      //char count = 0;//use for count char

      ch ='A';
      clrscr();
      while( (choice!='1')&&(choice!='2'))
      {
            printf("Use COM1 type (1)Use COM2 type (2,\n");
            choice=getch();
      }
      if(choice=='1')
            com_status=InitSerialComm(COM1,19600,D8,S2,PNONE);
      else
            com_status=InitSerialComm(COM2,19600,D8,S2,PNONE);
      choice='N';
      switch(com_status)
      {
            case PASS:;break;
    default:exit(1);
      }
      ResetBuffer();

      count = 0;
      {
            btran[count] = StaByte;//CONFIRM PATTERN OF TRANSMITT
            count++;
            btran[count] = no[0];
            do
            {
                  count++;
                  btran[count]=mass[cnt];
                  cnt++;
            }while(mass[cnt]!='\x0');
            count++;
            btran[count] = StoByte;
            count = 0;
            do
            {
```

```
                    tran(ch);
                    count++;
                }while(btran[count] != StoByte);
        ch=btran[count];
        tran(ch);
        QuitSerialComm();
        return 0;
          }
}
void tran(ch)
unsigned char ch;
{
int s_status;
                    s_status        = xmit(ch);
                    if(s_status==FAIL) printf("Failure... TXCOUNT is small\nPress character to send\n");//txcount = 32767
                    if(ReceiveFlag==ON)
                    {
                        putchar(CommBuf[CommDataPos-1]);
                        ReceiveFlag=OFF;
                    }
}
```

```
RS              =       ra.2            ;0 = instruction, 1 = data
RW              =       ra.1            ;0 = write, 1 = read
E               =       ra.0            ;0 = disable, 1 = enable
data            =       rb              ;Data to LCD
Buzzer          =       ra.3
serial_in       =       rc.0            ;serial input pin
Key_shift_L     =       rc.4
Key_enter       =       rc.5
Key_reset       =       rc.6

; Declare constant for  frame format of data sended from PC.

start_byte      =       0fh
stop_byte       =       0ffh
addr_byte       =       'A'

; Declare constants for common LCD instructions. To perform the functions
; listed below, clear bit RS and send constant to the display.  Remember
; to set RS again before sending characters to the LCD.

clearLCD        =       1               ;Clears the display (fills with blanks)
Home            =       2               ;Returns display to the home position.
shift_l         =       24              ;Shifts display to the left.
shift_r         =       28              ;Shifts display to the right.
crsr_l          =       16              ;Moves cursor to the left.
crsr_r          =       20              ;Moves cursor to the right.
blink_c         =       11              ;Blinks whole character to indicate
                                        ;cursor position.
no_crsr         =       8               ;Turns off the cursor.

Buf1            =       30h             ;Start address of temp line.
Buf2            =       50h
Buf3            =       70h
numRAM          =       48


; Set RAM origin above special registers, declare variables, and set code
; origin.

                org     8
temp            ds      1
temp2           ds      1
ptr             ds      1               ;Pointer of RAM.
counter         ds      1               ;Index variable.
delay_cntr      ds      1               ;Delay counter
bit_cntr        ds      1               ;Counter of the number of data
                                        ;bit of 1 frame.
rcv_byte        ds      1               ;Receive serial data .
runchar         ds      1
                org     10h
data'           ds      1
A2              ds      1
B'              ds      1
F2              ds      1

; receiving RS-232 data rate
bitK            =       206             ;for  1,200-baud operation @ 4 MHz
; bitK          =       103             ;for  2,400-baud operation @ 4 MHz
; bitK          =       51              ;for  4,800-baud operation @ 4 MHz
; bitK          =       24              ;for  9,600-baud operation @ 4 MHz
; bitK          =       12              ;for 19,200-baud operation @ 4 MHz
half_bit        =       bitK/2          ;as shown in table.

                org     0
```

```
; Device data
            device  pic16c57,xt_osc,wdt_off,protect_off
            reset   start

start   setb    pa0
        call    initport
        call    wait                    ;wait LCD start up.
        setb    pa0
        call    initLCD
reset   mov     temp2,#clearLCD
        call    Wcmd
        call    wait
        call    wait
        mov     w,#1                    ;Display main screen.
        setb    pa0                     ;Select page 1.
        call    Wstring                 ;Select back page 0.

;*******Start to receive data frame.**********************
start_R
        setb    pa0
        call    Byte_sync


        setb    pa0                     ;Selected page 1
        call    Receive
        cjne    rcv_byte,#addr_byte,start_R     ;Check address of called
                                        ;pager.

;******Receive and save data(ASCII) in RAM buffer.*******

        clr     numchar                 ;numchar is used to check
                                        ;end of RAM.
        clr     temp                    ;Set before call Save.
        mov     counter,#16
        mov     ptr,#Buf1               ;Set ptr pointing to origin
                                        ;of RAM.

save_lp setb    pa0                     ;Selected page 1
        call    Receive
        call    Save
        cje     rcv_byte,#stop_byte,_end_s      ;Check end of frame.
        inc     numchar
        cje     numchar,#numRAM,_end_s          ;Check end of RAM.
        jmp     save_lp
_end_s

;**********BEEP and check key use temp,temp2
beep    clrb    Buzzer
        mov     temp2,#home             ;Set DD RAM 0,0
        call    Wcmd
        mov     w,#2                    ;Display ((-)).
        setb    pa0                     ;Select pagBe 1.
        call    Wstring                 ;Select back page 0.
        mov     delay_cntr,#6           ;beep 1 sec
L_lp    call    wait
        jnb     Key_enter,_end_B        ;check key
        djnz    delay_cntr,L_lp

        setb    Buzzer
        mov     temp2,#home             ;Set DD RAM 0,0
        call    Wcmd
        mov     w,#1                    ;clear ((-)).
        setb    pa0                     ;Select page 1.
        call    Wstring                 ;Select back page 0.
        mov     delay_cntr,#6           ;beep 1 sec
H_lp    call    wait
```

```
        djnz   delay_cntr,H_lp
        jmp    beep
_end_B
        setb   Buzzer
        mov    temp2,#home.            ;Set DD RAM 0,0
        call   Wcmd
        mov    w,#3
        setb   pa0                     ;Select page 1.
        call   Wstring                 ;Select back page 0.

enter_lp jb    Key_enter,key_3
        call   delay_key
        jmp    display
key_3   jb     Key_reset,enter_lp
        call   delay_key
        setb   pa0
        call   Clear
        jmp    reset

;******Display 1 line on LCD if press key enter.**************
display
        mov    temp2,#clearLCD
        call   Wcmd
        clr    bit_cntr                ;Set before call Get.
                                       ;('bit_cntr' is used in stead of
                                       ; used temp in Save to remain
                                       ; the value of pointer that point
                                       ; to Buf_table.)
        mov    counter,#16
        mov    ptr,#Buf1               ;Set ptr pointing to origin.
;       mov    temp2,#home             ;Set DD RAM go to 0,0
;       call   Wcmd
line_1
        call   Get_F
        cje    rcv_byte,#stop_byte,_end_d      ;Check end of the frame.
        cjne   counter,#7,half
        mov    temp2,#0C0H
        call   Wcmd

half    mov    temp2,rcv_byte
        call   Wdata
        cje    ptr,#40h,_end_d                 ;Check end of 1 line.
        jmp    line_1

_end_d

        mov    A2,#87H
        mov    B1,#0C0H
        mov    B2,#3C7H

;*******Wait for key*******************************************
wait_key jt    Key_shift_L,key_2       ;If Key_shift then shift left
                                       ;the display on LCD and get char.
                                       ;from RAM to LCD on the BOL.
        cjne   numchar,#16,wait_key    ;Check,if number of char. below
                                       ;or equal 16 char. then don't
                                       ;shift.

        call   Get_F
        cje    rcv_byte,#stop_byte,display     ;If end of text then
                                               ;display again on LCD.
        mov    temp2,B1                ;Set address to #C0Oh+1
        call   Wcmd                    ;before shift.
        setb   pa0
        call   Rdata                   ;Read data from DD RAM.
```

```
        mov   temp2,#shift_l        ;Shift left DD.
        call  Wcmd
        inc   A2                    ;Increase DD RAM.
        inc   B1
        inc   B2
        mov   temp2,A2              ;Set DD RAM to the end of
        call  Wcmd                  ;first half.
        mov   temp2,data1
        call  Wdata

        mov   temp2,B2              ;Set DD RAM to the End Of Line.
        call  Wcmd
        mov   temp2,rcv_byte
        call  Wdata

        call  delay_key
        jmp   wait_key


key_2   jb    Key_enter,wait_key
        call  delay_key
        jmp   _end_B



;************ END Program **************


;*** Epulse Wcmd Wdata ***
; Write data or instructions (in variable temp2) to the LCD.
; use temp,pa2

Wcmd    clrb  RS                    ;RS=0 to send instruction.
        jmp   Epulse
Wdata   setb  RS                    ;RS=1 to send Data

Epulse  movb  pa2, RS               ;Store RS in unused bit.
        clrb  RS                    ;Clear RS to send instruction.
        setb  RW                    ;Set RW to read.
        mov   !rb, #255             ;Port rb: all inputs.

loop    setb  E                     ;Enable the LCD.
        clr   delay_cntr
Ep1     djnz  delay_cntr,Ep1
        mov   temp,data
        clrb  E                     ;Disable LCD.
        snb   temp.7                ;Is the LCD busy?
        jmp   loop                  ;Yes, try again later.
        movb  RS, pa2               ;No, send the data or instruction.
        clrb  RW                    ;RW=0 Write data
        mov   !rb, #0               ;set to out port
        mov   data, temp2
        setb  E                     ;send pulse E
        clr   delay_cntr
Ep2     djnz  delay_cntr,Ep2
        clrb  E
        ret


;*****************************************************
;WAIT no input
;use temp,temp2
wait            mov   temp2,#200    ;Create a delay for LCD power-on reset.
:loop           djnz  temp,:loop
```

```
                ret
;Delay  0.3 s
;call wait
delay_key       mov     delay_cntr,#2
key_lp          call    wait
                djnz    delay_cntr,key_lp
                ret

;*******Save , Get data 1 character*********
;(call  Receive before)
;call   Buf_table
;use    temp,w,counter,ptr,fsr,indirect,pa2(bit)
;***Save:
;input   rcv_byte
;output  RAM
;Buf1           =       30h             ;Start address of RAM line.
;Buf2           =       50h
;Buf3           =       70h

;set begin addr before call
;set counter = 16 before call
;clr temp before call.

Save    cjne    counter,#0,save_nxt
        mov     counter,#16
        inc     temp                    ;File 'temp' is used to
        mov     w,temp                  ;point to the Buf_table.
        call    Buf_table               ;table of buffer address.
        mov     ptr,w                   ;load pointer with begin address.
save_nxt mov    fsr,ptr
        mov     indirect,rcv_byte       ;save data to RAM.
        clr     fsr
        inc     ptr
        dec     counter
        ret

Buf_table       jmp     pc+w
                retw    Buf1,Buf2,Buf3

;call   Buf_table
;use    bit_cntr,w,counter,ptr,fsr,indirect
;***Get_ch:
;input  RAM
;output  rcv_byte
;Buf1           =       30h             ;Start address of RAM line.
;Buf2           =       50h
;Buf3           =       70h

;set begin addr before call
;set counter = 16 before call
;clr bit_cntr before call.
Get_F   cjne    counter,#0,get_nxt
        mov     counter,#16
        inc     bit_cntr                ;File 'temp' is used to
        mov     w,bit_cntr              ;point to the Buf_table.
        call    Buf_table               ;table of buffer address.
        mov     ptr,w                   ;load pointer with begin address.
get_nxt mov     fsr,ptr
        mov     rcv_byte,indirect       ;move RAM to rcv_byte.
        clr     fsr
        inc     ptr
        dec     counter
        ret
```

```
                org     200h
;*****These below routines are on memory page 2*********
;*****,must select page by pa1 and pa0 when call *******
;*****across memory page.                      ******
initport mov    ra, #0000 1000b         ;Initialize ports, power LCD.
         mov    rb, #0
         mov    rc, #0
         mov    !ra,#0h                 ;Set control lines to output.
         mov    !rb,#0h                 ;Set data lines to output.
         mov    !rc,#0111 0001b         ;Set rc0 to serial input and
                                        ;rc4,rc4,rc6 to input key.
         clrb   pa0
         ret

initLCD  mov    temp2,#0011 1000b       ;Initialize LCD: set 8-bit,1-line
         clrb   pa0
         call   Epulse                  ;(DL N F . .)
         mov    temp2,#0000 1100b       ;Display on/off (D=1 ON,C=0 OFF,B=0 OFF)
         call   Epulse
         mov    temp2,#0000 0110b       ;Entry mode set (I/D=1 inc,S=0 right)
         call   Epulse
         clrb   pa0
         ret

;********Clear  ram*************
;call   Buf_table
;use    temp,numchar,counter,rcv_byte
;       w,ptr,fsr,indirect
;output #00 to ram

Clear    clr    temp
         clr    numchar
         mov    counter,#16
         mov    ptr,#Buf1
         clr    rcv_byte
clr_lp   cjne   counter,#0,c_nxt        ;Check end of line.
         mov    counter,#16
         inc    temp
         mov    w,temp
         clrb   pa0
         call   buf_table               ;Table of buffer address.
                                        ;Load pointer with begin address.
         setb   pa0
         mov    ptr,w
c_nxt    mov    fsr,ptr
         mov    indirect,rcv_byte
         clr    fsr
         inc    ptr
         dec    counter
         inc    numchar
         cje    numchar,#numRAM,_end_c  ;Check end of ram.
         jmp    clr_lp

_end_c   clrb   pa0
         ret

;*** Wstring write stream of char end with 0Dh ***
;input number of massage in W
;use counter,data1,temp2
;call   Epulse
Wstring  mov    data1,w                ;save number of massage
         setb   RS                     ;RS=1 Data
         clr    counter
;
:Select  cje    data1,#1,:mas1
```

```
        cje   data1,#3,:mas3

:mas1   mov   w,counter
        call  msg1
        jmp   :countmsg
:mas2   mov   w,counter
        call  msg2
        jmp   :countmsg
:mas3   mov   w,counter
        call  msg3
        jmp   :countmsg
;
:countmsg mov.  temp2,w                 ;write LCD
          cje   temp2,#0dh,:exit        ;end of string
          clrb  pa0
          call  Wdata                   ;Epulse
          setb  pa0
          inc   counter                 ;next char
          cjne  counter,#8,:f_half
          mov   temp2,#0C0H
          clrb  pa0
          call  Wcmd
          setb  pa0

:f_half   jmp   :Select
:exit     clrb  pa0
          ret
;
msg1            jmp     pc+w            ;Table of ASCII and custom characters
                                        ;to display.
                retw    '>              !',0dh
msg2            jmp     pc+w
                retw    '>    ((-))     !',0dh
msg3            jmp     pc+w
                retw    '>  *MESSAGE*   !',0ch

;*****Delay routine***************
bit_delay       mov     delay_cntr,#bitK        ;bit delay time.
:loop           nop
                djnz    delay_cntr, :loop
                ret

start_delay     mov     delay_cntr,#half_bit    ;half delay time.
:loop           nop
                djnz    delay_cntr, :loop
                ret

;*******Receive serial data*******
;output rcv_byte
;use   delay_cntr,bit_cntr
;call  bit_delay,start_delay

Receive
:start_bit snb  serial_in               ;Detect start bit. Change to
                                        ;sb serial_in if using 22k
                                        ;resistor input.
          jmp   :start_bit              ;No start bit yet? Keep watching.
          call  start_delay             ;Wait one-half bit time to the
                                        ;middle of the start bit.
          jb    Serial_in,:start_bit    ;If the start bit is still
                                        ;good,continue. Otherwise,
                                        ;resume waiting.Change to jnb
                                        ;Serial_in,:start_bitif using
                                        ;22k resistor input.
          mov   bit_cntr, #8            ;Set to receive 8 data bits
```

```
        clr   rcv_byte          ;Clear the receive byte to get
                                ;ready for new data.
:receive call bit_delay         ;Wait one bit time.
        movb  c,Serial_in       ;Put the data bit into carry.
                                ;Change to movb c,/Serial_in
                                ;if using 22k resistor input.
        rr    rcv_byte          ;Rotate the carry bit into
                                ;the receive byte.
        djnz  bit_cntr,:receive ;Not eight bits yet? Get next bit.
        call  bit_delay         ;Wait for stop bit.
        jb    Serial_in,exit    ;check stop bit
        jmp   :start_bit        ;Framing Err goto start

exit    clrb  pa0
        ret

;*************************************
Byte_sync
begin_st  snb   serial_in       ;Detect start bit. Change to
                                ;sb serial_in if using 22k
                                ;resistor input.
        jmp   begin_st          ;No start bit yet? Keep watching.
        call  start_delay       ;Wait one-half bit time to the
                                ;middle of the start bit.
        jb    Serial_in,begin_st ;If the start bit is still
                                ;good,continue. Otherwise,
                                ;resume waiting.Change to jnb
                                ;Serial_in,:start_bitif using
                                ;22k resistor input.
        call  bit_delay         ;Wait one bit time.

        clr   temp
L_sect  sb    serial_in         ;Check 4 bits low 1111(F).
        jmp   begin_st
        call  bit_delay
        inc   temp
        cjne  temp,#4,L_sect

        clr   temp
H_sect  snb   serial_in         ;Check 4 bits high 0000(0).
        jmp   begin_st
        call  bit_delay
        inc   temp
        cjne  temp,#4,H_sect

        call  bit_delay         ;Wait for stop bit.
        jb    Serial_in,end_st  ;check stop bit
        jmp   begin_st          ;Framing Err goto start
end_st  clrb  pa0
        ret

;************Read data from LCD DI RAM.********
Rdata   setb  RS                ;RS=1 to send Data

        movb  pa2, RS           ;Store RS in unused bit.
        clrb  RS                ;Clear RS to send instruction.
        setb  RW                ;Set RW to read.
        mov   !rb, #255         ;Port rb: all inputs.

:loop   setb  E                 ;Enable the LCD.
        clr   delay_cntr
:Dp1    djnz  delay_cntr,:Dp1
        mov   temp,data
        clrb  E                 ;Disable LCD.
        snb   temp.7            ;Is the LCD busy?
        jmp   :loop             ;Yes, try again later.
```

```
        movb    RS, pa2         ;No, send the data or instruction.
        setb    RW              ;RW=1 Read data
        mov     !rb, #255       ;set to in port
        setb    E               ;send pulse E
        clr     delay_cntr
:Ep2    djnz    delay_cntr,:Ep2
        mov     temp2,data
        clrb    E
        clrb    pa0
        ret
```

กิตติกรรมประกาศ

ขอขอบคุณ ผศ.ดร. รัตติกร วรากูลศิริพันธ์ และพี่ศมิทธ์ เอมสมบัติ ที่ได้เอื้อเฟื้อเครื่องมือในการทดลอง และให้แนวความคิดรวมทั้งคำแนะนำต่าง ๆ ในการจัดทำโครงงานนี้

เอกสารอ้างอิง

1. กองบรรณาธิการ, "แพคลิ้งค์ วิทยุติดตามตัวระบบดิจิตอล", วารสารเซมิคอนดัคเตอร์, ฉบับที่ 73, 2529, หน้า 162-169.
2. เศกสิทธิ์ คาชมภู, "อุปกรณ์น่าใช้ / IC น่าสน : PIC16C5X ไมโครคอนโทรเลอร์", วารสารเซมิคอนดัคเตอร์, ฉบับที่ 137, 2537, หน้า 107-111.
3. Microchip Technology Inc., "PIC16CXX MICROCONTROLLER ASSEMBLER GUIDE", 1992.
4. บราโมทย์ วาดเขียน, "พื้นฐานการสื่อสารข้อมูล", คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 218 หน้า, 2536.
5. ยืน ภู่วรวรรณ, "เทคโนโลยีฮาร์ดแวร์ IBM PC", คณะวิศวกรรมศาสตร์ มหาวิทยาลัยเกษตรศาสตร์, 335 หน้า, 2537.