การสื่อสารข้อมูลระหว่างไมโครคอมพิวเตอร์กับเมนเฟรมคอมพิวเตอร์เพื่อการประมวลสัญญาณ

DATA COMMUNICATION BETWEEN A MICROCOMPUTER AND A MAINFRAME COMPUTER FOR SIGNAL PROCESSING

มนูร ชิณการณ์

MANOON CHINNAKARN

อาจารย์ที่ปรึกษา

ศ. ดร. ไพรัช ธัชยพงษ์

Prof. Dr. Pairash Thajchayapong

B.Sc. (1st Hons, London)

A.C.G.I., Ph.D. (Cantab)

วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิศวกรรมไฟฟ้า

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2529

สารบัญ

บทคัดย่อ

ในการศึกษาทางด้านการประมวลสัญญาณดิจิตอล มีความต้องการคอมพิวเตอร์ที่มีสมรรถนะในการคำนวณ และความจุในการเก็บข้อมูลสูง คอมพิวเตอร์ในระดับเครื่องเมนเฟรมคอมพิวเตอร์ และมินิคอมพิวเตอร์ จึงได้ถูกนำมาใช้งานทางด้านนี้อย่างกว้างขวาง ในขณะเดียวกันไมโครคอมพิวเตอร์ซึ่งมีความคล่องตัวในการอินเทอร์เฟสสูง ก็ได้ถูกนำมาดัดแปลงเพื่อใช้งานเป็นอุปกรณ์แสดงภาพ และอุปกรณ์เก็บข้อมูลสัญญาณภาพ เพื่อใช้ร่วมกับการประมวลสัญญาณในงานดังกล่าว ปัญหาที่เกิดขึ้นคือความยุ่งยากในการถ่ายเทข้อมูลระหว่างเครื่องเมนเฟรมคอมพิวเตอร์ และไมโครคอมพิวเตอร์ ซึ่งในอดีตที่ผ่านมาการถ่ายเทข้อมูลจะทำได้ โดยการถ่ายข้อมูลจากเครื่องเมนเฟรมคอมพิวเตอร์ลงในดิสเกตต์ขนาด 8 นิ้ว นำมาผ่านขบวนการแปลงฟอร์แมท และถ่ายข้อมูลไปยังดิสเกตต์ที่รับรู้ได้ โดยระบบจัดการของไมโครคอมพิวเตอร์ ซึ่งเป็นขบวนการที่ยุ่งยากและซับซ้อน

วิทยานิพนธ์เรื่องนี้ได้เสนอการนำวิธีการทางด้านการสื่อสารข้อมูล มาช่วยในการแก้ปัญหาดังกล่าว โดยเริ่มจากการวิเคราะห์โปรโตคอลของเครื่องเมนเฟรมคอมพิวเตอร์ ที่ใช้การออกแบบโปรโตคอลคอนเวอร์เตอร์ให้เหมาะกับการใช้งานกับข้อมูลที่ต้องการ และการประยุกต์โปรโตคอลคอนเวอร์เตอร์ ในการถ่ายเทข้อมูลสัญญาณภาพ จากขั้นตอนการออกแบบและทดลอง จะได้โปรโตคอล คอนเวอร์เตอร์ที่จะช่วยในการแก้ปัญหาการถ่ายเทข้อมูลได้ตามความมุ่งหมาย

ABSTRACT

Data Communication between a Microcomputer and a Mainframe Computer for Signal Processing is a research designed to solve the problems arising in the study of Digital Signal Processing and Digital Image Processing at Computer Research and Service Center, King Mongkut's Institute of Technology. The study of Digital Signal Processing and Digital Image Processing require a mainframe computer and a microcomputer with peripherals working as an image work station. The difficulty is that a mainframe computer and a microcomputer cannot communicate with each other directly. Data must be transfered manually, or by way of a complicated procedure using a storage media. This research is intended to solve the problem.

In this research, first the communication protocol is analyzed. Then, the information from the analysis is used in the design and in the construction ofthe protocol converter. The converter is then connected between the microcomputer and the mainframe computer. From the test, it is proved that data can be communicated in both directions.

บทที่ 1

บทนำ

การประมวลสัญญาณทางดิจิตอล (Digital Signal Processing) และการประมวลสัญญาณภาพทางดิจิตอล (Digital Image Processing) เป็นการประยุกต์ใช้ในงานคอมพิวเตอร์ระดับสูง ที่ต้องอาศัยคอมพิวเตอร์ ที่มีความสามารถในการคำนวณสูง และมีหน่วยความจำขนาดใหญ่ เพื่อรองรับปริมาณของข้อมูลที่สูงมาก ดังนั้นคอมพิวเตอร์ ที่เหมาะสมกับงานทางด้านนี้ จึงเป็นเครื่องในระดับเมนเฟรม หรือมินิคอมพิวเตอร์

ทางสำนักวิจัยและบริการคอมพิวเตอร์ สถาบันเทคโนโลยีพระจอมเกล้า ลาดกระบัง ได้ทำการติดตั้งคอมพิวเตอร์เมนเฟรม NEC ระบบ 300 และระบบจัดการ แบบ ACOS-4 (ซึ่งเรียกรวมกันว่า NEC ACOS-300) โดยได้รับความช่วยเหลือจากรัฐบาลญี่ปุ่น ในปี พ.ศ 2523 สำหรับงานบริการนักศึกษาและการวิจัย และได้ทำการวิจัยทางด้าน การประมวลสัญญาณภาพ และภาพถ่ายทางดาวเทียมจากดาวเทียมสำรวจทรัพยากรธรรมชาติ LANDSAT เรื่อยมา.

ในงานวิจัยดังกล่าว จำเป็นจะต้องนำข้อมูลของสัญญาณภาพ ที่ได้จากอุปกรณ์แปลงสัญญาภาพ (Image Digitizer) ซึ่งประกอบอยู่กับไมโครคอมพิวเตอร์ ไปป้อนเข้าสู่เครื่องเมนเฟรม ในทางตรงกันข้ามก็จะต้องนำเอาท์พุท ที่ได้จากเครื่องเมนเฟรม มาแสดงด้วยอุปกรณ์แสดงภาพ (Image Display Monitor) และยังได้มีการเตรียมข้อมูลสัญญาณภาพดาวเทียม สำหรับไมโครคอมพิวเตอร์เพื่อใช้ในการวิเคราะห์ทางด้านต่าง ๆ ซึ่งทั้งหมดนี้จะต้องมีการถ่ายเทข้อมูลระหว่าง เครื่องเมนเฟรมกับไมโครคอมพิวเตอร์ โดยการถ่ายข้อมูลจากเครื่องเมนเฟรม ลงในดิสเกตต์ขนาด 8 นิ้ว และนำดิส-เกตต์ดังกล่าว มาผ่านขบวนการแปลงฟอร์แมท และถ่ายเทข้อมูลสู่ดิสเกตต์ ขนาด 5.25 นิ้ว สำหรับไมโครคอมพิวเตอร์ อีกครั้งหนึ่ง ซึ่งนับว่าเป็นขั้นตอนที่ยุ่งยาก และขาดประสิทธิภาพ

วิทยานิพนธ์ฉบับนี้ได้เสนอถึงวิธีการออกแบบ โปรโตคอล คอนเวอร์เตอร์ สำหรับการสื่อสารข้อมูลระหว่างไมโครคอมพิวเตอร์ กับเครื่องเมนเฟรม NEC ACOS-300 เพื่อการประมวลสัญญาณ ทางดิจิตอล โดยจะกล่าวถึงทฤษฎี การทดลอง และการออกแบบ โปรโตคอล คอนเวอร์เตอร์ เพื่องานดังกล่าว จึงได้แบ่งเนื้อหาในวิทยานิพนธ์นี้ออกเป็น 5 บท ดังนี้

บทที่ 1 คือบทนำนี้ ซึ่งได้กล่าวถึงวัตถุประสงค์ และความเป็นมาของการวิจัย

บทที่ 2 กล่าวถึงทฤษฎีการสื่อสารของเครื่องคอมพิวเตอร์เมนเฟรม NEC ACOS-300 ซึ่งได้รวบรวมจากเอกสารอ้างอิง และการทดลอง อันประกอบด้วยข้อมูลที่จำเป็นในการออกแบบ เช่น ระบบข้อมูลในการสื่อสาร ขั้นตอน และการควบคุมการสื่อสาร เป็นต้น

บทที่ 3 เป็นการทดลอง ในการวิเคราะห์โปรโตคอลของระบบ NEC ACOS-300 ได้แสดงถึงเทนิคในการวิเคราะห์โปรโตคอล โดยใช้เครื่องมือพื้นฐานในห้องทดลอง เพื่อรวบรวมข้อมูลที่จำเป็นในการออกแบบ และตรวจสอบต้นแบบ

บทที่ 4 อธิบายถึงวิธีการออกแบบ โปรโตคอล คอนเวอร์เตอร์ ต้นแบบสำหรับการทดสอบและพัฒนาฮาร์ดแวร์ และซอฟท์แวร์ โดยทางด้านฮาร์ดแวร์ ใช้เทคนิคของการใช้หน่วยความจำร่วมกันระหว่างตัวคอนเวอร์เตอร์ กับไมโครคอมพิวเตอร์ ที่ใช้เป็นเครื่องมือในการพัฒนา และทางด้านซอฟท์-แวร์ ใช้ระบบแกนจัดการแบบมัลติทาส์ค (Multitasking Kernel) เพื่อลดความยุ่งยากในการพัฒนาโปรแกรม และระบบฮาร์ดแวร์

บทที่ 5 แสดงถึงการประยุกต์ต้นแบบเพื่อใช้งาน โดยดัดแปลงเพื่อใช้กับไมโครคอมพิวเตอร์ IBM PC การออกแบบซอฟท์แวร์ และฮาร์ดแวร์ ส่วนอินเทอร์เฟส กับ IBM PC เพิ่มเติม และการเรียกใช้ คอนเวอร์เตอร์ จากโปรแกรมใช้งาน (Application program)

บทที่ 6 เป็นบทสุดท้าย จะสรุปผลการออกแบบ และทดลองใช้งาน พร้อมทั้งข้อเสนอแนะ ในการพัฒนาขั้นต่อไป

ในภาคผนวก ได้แสดงโฟล์วชาร์ท โปรแกรมควบคุมของ คอนเวอร์เตอร์ และโปรแกรมตัวอย่าง เพื่อการประยุกต์ใช้งานในแบบต่าง ๆ ต่อไป

บทที่ 2

ระบบการสื่อสารข้อมูลและการควบคุมการสื่อสาร

โดยทั่วไปเครื่องเมนเฟรมคอมพิวเตอร์ สามารถแบ่งได้ 2 ประเภท ตามลักษณะการใช้งาน คือ

ระบบที่มีการประมวลผลแบบแบทช์ (Batch processing) ซึ่งระบบประเภทนี้ผู้ใช้จะต้องทำโปรแกรมหรืองานที่ต้องการประมวลผล ป้อนเข้าสู่เครื่องคอมพิวเตอร์ ทางหน่วยอินพุทของเครื่องโดยตรง จึงไม่จำเป็นที่จะต้องมีระบบการสื่อสารข้อมูล ในการส่งผ่านข้อมูลไปยังเครื่องคอมพิวเตอร์

ระบบที่มีการประมวลผลในลักษณะออนไลน์ (Online processing) ซึ่งระบบประเภทนี้จะมีเครื่องเมนเฟรม ทำงานเป็นโฮสท์คอมพิวเตอร์ (Host computer) และมีเทอร์มินัล ในการอินพุท และเอาท์พุทข้อมูล ในบริเวณที่ห่างไกลออกไป จึงต้องมีระบบสื่อสารข้อมูล ในการส่งผ่านข้อมูลระหว่างเทอร์มินัลและโฮสท์คอมพิวเตอร์ จากระยะไกล

2.1 ระบบการสื่อสารข้อมูลของเครื่องคอมพิวเตอร์ NEC

ในระบบการสื่อสารข้อมูลของเครื่องคอมพิวเตอร์ NEC ได้มีการแบ่งออกเป็น 3 ประเภทตามชนิดของโปรโตคอลในการสื่อสารคือ

2.1.1 Level 0 procedure

เป็นการสื่อสารข้อมูลแบบ อะซิงโครนัส มีการทำงานเช่นเดียวกับของระบบสำหรับ เครื่อง TTY (Teletypewriter) ซึ่งไม่มีกฎเกณฑ์บังคับตายตัว ใช้สำหรับอุปกรณ์อินพุท เอาท์พุทความเร็วต่ำ หรือสำหรับคอนโซลของโอปะเรเตอร์

2.1.2 Level 2 procedure

ในการสื่อสารข้อมูลประเภทนี้ จะมีการตรวจสอบการผิดพลาดข้อมูลในระหว่างการรับส่งและส่งข้อมูลซ้ำเพื่อแก้ไขข้อมูลที่ผิดพลาด แบ่งออกเป็น 2 ชนิด

- Level 2A มีการสื่อสารแบบ อะซิงโครนัส สำหรับความเร็วในการรับส่งไม่เกิน 1200 BPS (บิทต่อวินาที)

- Level 2B มีการสื่อสารแบบ ซิงโครนัส สำหรับความเร็วในการรับส่งระหว่าง 2400 BPS ถึง 9600 BPS ใช้สำหรับการรับส่งข้อมูลที่เป็นตัวอักษร เนื่องจากมีขีดจำกัดของรหัสและโปรโตคอลที่ใช้

2.1.3 Level 4 procedure

ใช้ระบบการสื่อสารข้อมูล แบบ HDLC (High Level Data Link Control) เพื่อแก้ปัญหาขีดจำกัดในการรับส่งข้อมูลที่อยู่ในรูปไบนารี่ และต้องการใช้ความเร็วในการรับส่งสูง โดยจะมีความเร็วสูงได้ถึง 48 KBPS ใช้ในการสื่อสารระหว่างคอมพิวเตอร์กับเทอร์มินัล หรือระหว่างคอมพิวเตอร์ด้วยกัน ในระบบเครือข่ายคอมพิวเตอร์ (Computer Network System)

2.2 ขั้นตอนของการสื่อสารข้อมูล

2.2.1 ขั้นตอนที่ 1 เริ่มต้นการติดต่อ (Phase 1 Connecting the line.)

ในระบบการสื่อสารข้อมูล ที่มีการติดต่อผ่านข่ายการสื่อสารข้อมูล (Data communication Network) หรือผ่านระบบโทรศัพท์ (Public Telephone Network) ในตอนเริ่มต้นของการติดต่อ จะต้องมีการหมุนหมายเลข อาจจะเป็นด้วยมือหรืออัตโนมัติ เพื่อให้เกิดการเชื่อมต่อถึงกัน ระหว่างต้นทางถึงปลายทาง ซึ่งอาจจะเป็นจากเทอร์มินัล ถึงคอมพิวเตอร์ หรือจากคอมพิวเตอร์ ถึงคอมพิวเตอร์ เป็นต้น แต่สำหรับในระบบที่มีสายสำหรับการสื่อสารข้อมูลเป็นแบบอิสระ (Leased Communication Line) ซึ่งมีการเชื่อมต่อระหว่างต้นทางกับปลายทาง ถึงกันอย่างถาวร ก็ไม่จำเป็นต้องมีขั้นตอนนี้ได้

2.2.2 ขั้นตอนที่ 2 การจัดเตรียมเพื่อการส่งผ่านข่าวสาร (Phase 2 Establishing a data link.)

การจัดเตรียมเพื่อการส่งผ่านข่าวสาร เป็นขั้นก่อนหน้าที่จะมีการรับส่งข่าวสาร ซึ่งจะต้องมีการตรวจสอบ เพื่อความแน่ใจว่าทางปลายทางที่กำลังติดต่อด้วย คือ คอมพิวเตอร์ หรือเทอร์มินัล ที่ต้องการรับหรือส่งข้อมูลที่ถูกต้อง ซึ่งในการจัดการจะประกอบด้วย ข้อมูลสำหรับการควบคุมซึ่งมีหน้าที่ต่าง ๆ ดังนี้

- ส่งสัญญาณเรียกไปยังปลายทาง

- ตรวจสอบว่าปลายทางเป็นผู้รับที่ถูกต้องหรือไม่
- บอกว่าทางฝ่ายไหนจะเป็นผู้ส่งหรือรับ
- เลือกอุปกรณ์ปลายทางที่ต้องการให้เป็นอินพุทหรือเอาท์พุท
- ตรวจสอบสถานะของอุปกรณ์ปลายทางว่าพร้อมที่จะรับหรือส่งหรือไม่

2.2.3 ขั้นตอนที่ 3 การส่งผ่านข่าวสาร (Phase 3 Transfer of Information.)

เมื่อจะมีการรับหรือส่งข่าวสารระหว่างคอมพิวเตอร์ กับเทอร์มินัล หลังจากได้ผ่านขั้นตอนที่ 2 ซึ่งมีการตรวจสอบและกำหนดผู้รับส่ง เป็นที่เรียบร้อยแล้ว ในขั้นตอนที่ 3 นี้ ทางฝ่ายที่เป็นผู้ส่ง ก็จะส่งข่าวสาร พร้อมกับมีการตรวจสอบ และแก้ไขความผิดพลาด อันอาจเกิดขึ้นได้ระหว่างการส่งผ่านข่าวสารตามกฎเกณฑ์ของโปรโตคอล จนกระทั่งจบข่าวสารที่ต้องการส่ง

2.2.4 ขั้นตอนที่ 4 การสิ้นสุดการส่งผ่านข่าวสาร (Phase 4 Termination.)

หลังจากสิ้นสุดการส่งข่าวสารแล้ว ก็จะเข้าสู่ขั้นตอนที่ 4 ซึ่งทางฝ่ายผู้ส่งจะส่งสัญญาณบอกทางฝ่ายผู้รับ เพื่อแสดงว่าสิ้นสุดข่าวสารที่จะส่งแล้ว และทางฝ่ายผู้รับก็จะส่งสัญญาณแสดงการรับทราบ เป็นการยืนยันการทำงานอีกขั้นหนึ่ง และจะกลับไปสู่สถานะก่อนขั้นตอนที่ 2 เพื่อเตรียมสำหรับการรับส่งข่าวสารครั้งต่อไป

2.2.5 ขั้นตอนที่ 5 การยกเลิกการติดต่อ (Phase 5 Disconnecting the line.)

ขั้นตอนนี้มีการทำงานตรงกันข้ามกับขั้นตอนที่ 1 เพื่อยกเลิกการติดต่อ ซึ่งได้เกิดขึ้นจากขั้นตอนที่ 1 ดังนั้น สำหรับในระบบที่มีสายสำหรับการสื่อสารเป็นอิสระ ก็ไม่จำเป็นต้องมีขั้นตอนนี้เช่นกัน

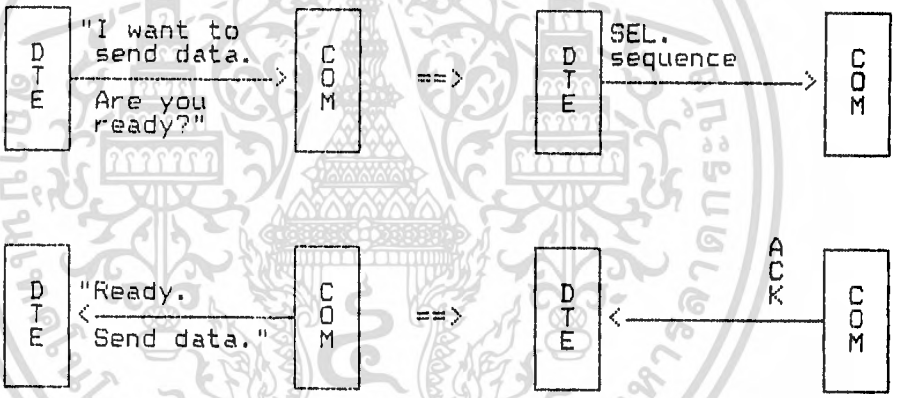
2.3 ขั้นตอนในการควบคุมการสื่อสารของโปรโตคอลแบบ Level 2B

ในการควบคุมการสื่อสาร ระหว่างคอมพิวเตอร์ กับเทอร์มินัล ของโปรโตคอล แบบ Level 2B มีการใช้สัญญาณและอักษรควบคุม ซึ่งจะได้อธิบายในหัวข้อ 2.5 โดย โปรโตคอล แบบ Level 2B นี้ จะรับผิดชอบการจัดการเฉพาะจาก ขั้นตอนที่ 2 ถึง ขั้นตอนที่ 4 เท่านั้น

2.3.1 ขั้นตอนที่ 2 การจัดเตรียมการส่งผ่านข่าวสาร

โปรโตคอล แบบ Level 2B มีการจัดเตรียมการส่งผ่านข่าวสาร 2 ลักษณะ คือ Contention mode และ Polling and Selecting mode.

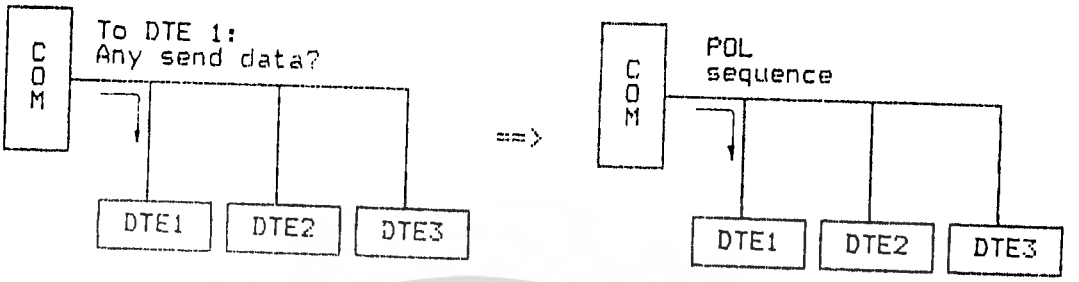
1) Contention mode ในการสื่อสารลักษณะนี้ ทั้งทางด้านคอมพิวเตอร์ และเทอร์มินัล จะมีสิทธิเท่าเทียมกัน ในการส่งข่าวสาร โดยทางฝ่ายที่ต้องการส่งข่าวสาร จะเป็นผู้ส่งสัญญาณในการจัดเตรียมการส่งข่าวสาร เช่น ในกรณีที่เทอร์มินัลต้องการส่งข่าวสารไปยังคอมพิวเตอร์ จะมีการทำงานดังแสดงในรูป 2.3.11 ส่วนในกรณีที่คอมพิวเตอร์ต้องการส่งข่าวสารไปยังเทอร์มินัล ก็จะมีการทำงานในลักษณะเดียวกัน แต่เป็นในทิศทางตรงกันข้ามกัน



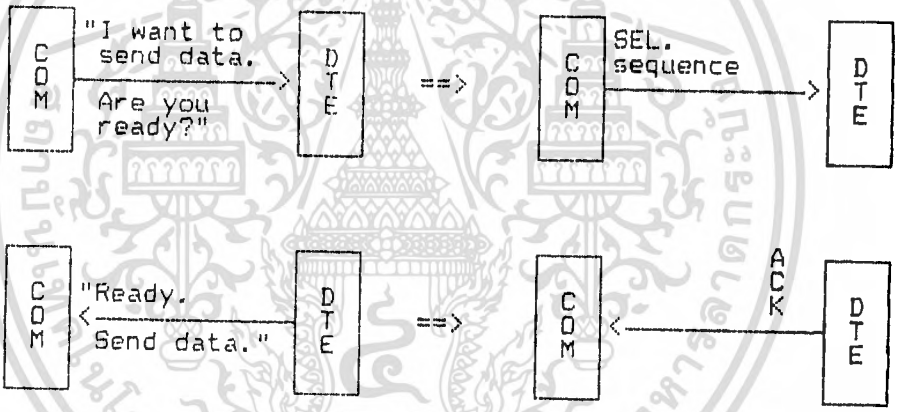
รูปที่ 2.3.11 การทำงานใน Contention mode.

2) Polling and Selecting mode ในลักษณะนี้คอมพิวเตอร์ จะมีหน้าที่ในการจัดเตรียมการส่งผ่านข่าวสาร ในกรณีที่เทอร์มินัล ต้องการส่งข่าวสารมาให้คอมพิวเตอร์ เทอร์มินัล จะต้องรอจนกว่า คอมพิวเตอร์จะส่งสัญญาณควบคุมไปสอบถาม (Polling) ดังแสดงในรูป 2.3.12 การทำงานลักษณะนี้ เหมาะสำหรับระบบที่มีวงจรการติดต่อเพียงวงจรเดียวแต่มีหลายเทอร์มินัล ต่อรวมอยู่ด้วยกัน

เมื่อคอมพิวเตอร์ต้องการส่งข่าวสารไปยังเทอร์มินัล คอมพิวเตอร์ก็จะส่งสัญญาณเพื่อเลือก (Selecting) เทอร์มินัลตัวที่ต้องการ ดังแสดงในรูป 2.3.13



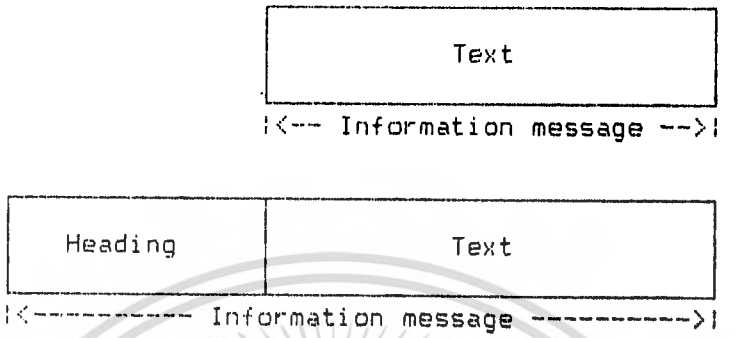
รูปที่ 2.3.12 การ Polling



รูปที่ 2.3.13 การ Selecting

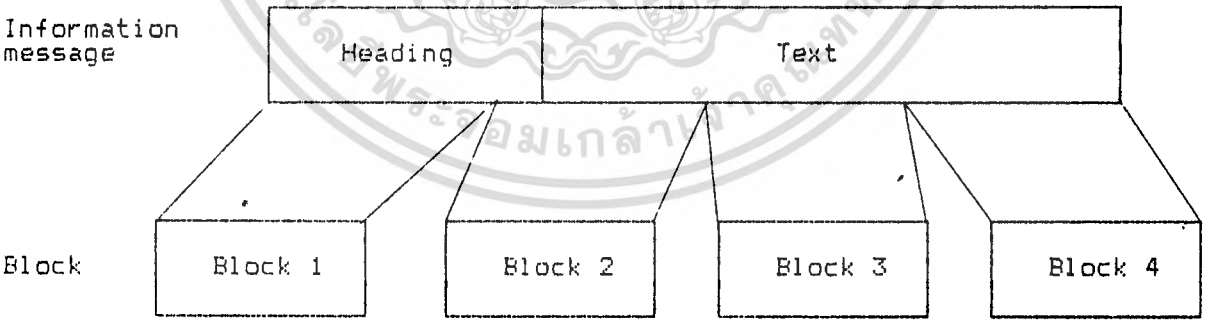
2.3.2 ขั้นตอนที่ 3 การส่งผ่าน ข่าวสาร

ข่าวสารในระบบโปรโตคอล แบบ Level 2B ประกอบด้วย ส่วนที่เป็นข้อมูลจริง (Text) ที่ส่งให้กับทางฝ่ายผู้รับเพื่อนำไปใช้งาน และส่วนที่เป็นการควบคุมในรูปแบบต่าง ๆ (Heading) [1] ซึ่งในแต่ละชุดของข้อมูลที่จะส่งไป อาจประกอบด้วยส่วนที่เป็น Text อย่างเดียว หรือมีทั้ง Heading และ Text ก็ได้ดังแสดงในรูป 2.3.21

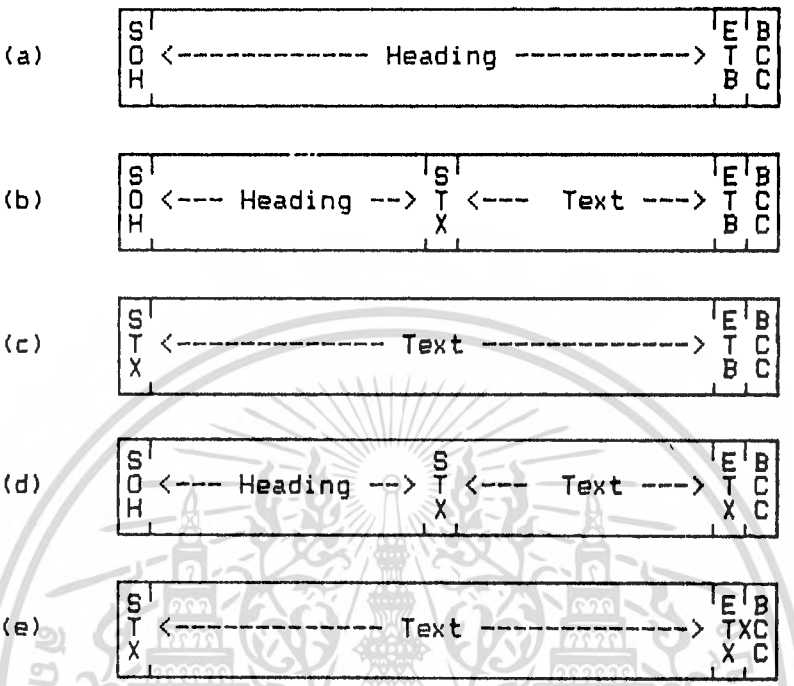


รูปที่ 2.2.21 รูปแบบของข่าวสาร

ข่าวสารแต่ละชุดที่ถูกส่งไปยังฝ่ายรับจะถูกแบ่งออกเป็นบล๊อค (Blocks) ที่มีความยาวคงที่ เพื่อความสดวกในการควบคุมผิดพลาด โดยแต่ละบล๊อคจะมีความยาว 256 ตัวอักษร ดังแสดงในรูปที่ 2.3.22



รูปที่ 2.3.22 ชุดของข่าวสาร และบล๊อค

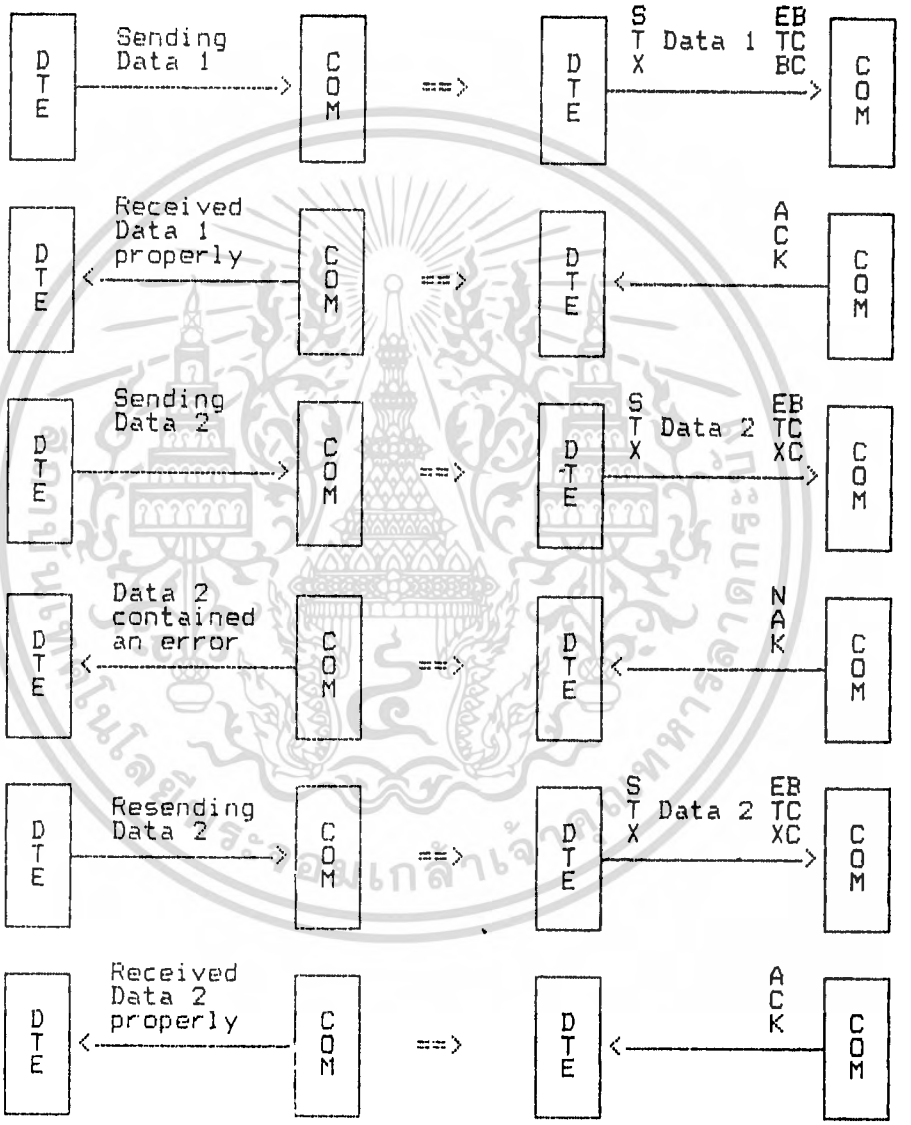


รูปที่ 2.3.23 ฟอร์แมทของข่าวสารในแต่ละบล๊อค

ข่าวสารในแต่ละบล๊อค จะถูกกำหนดให้มีฟอร์แมท ดังแสดงในรูป 2.2.23 โดยที่ข่าวสารแต่ละชุดอาจจะเริ่มต้นด้วยบล๊อคในแบบใดก็ได้ แต่บล๊อคสุดท้ายของข่าวสารแต่ละชุดจะต้องเป็นแบบ (d) หรือ (e) เท่านั้น ตัวอย่างเช่นชุดข้อมูลในรูป 2.3.22 จะถูกแบ่งเป็นบล๊อคแบบต่าง ๆ คือ (a), (b),(c) และ (e) เป็นต้น สำหรับข่าวสารที่มีความยาวไม่เกิน 256 ตัวอักษร ก็จะมีเพียงบล๊อคเดียว อาจจะเป็นแบบ (d) หรือ (e) ก็ได้

ในฟอร์แมทของบล๊อคข่าวสารแต่ละแบบ จะมี SOH, STX, ETB, ETX เป็นอักษรควบคุมการส่ง และ BCC (Block Check Character) เป็นตัวอ้างอิง ในการตรวจสอบความผิดพลาดของข่าวสาร สำหรับทางฝ่ายผู้รับ เมื่อฝ่ายผู้รับ ได้รับข่าวสารแต่ละบล๊อค ก็จะทำการคำนวณเพื่อตรวจสอบความผิดพลาด และส่งสัญญาณตอบรับ เพื่อแสดงผลว่าได้รับข่าวสารถูกต้อง หรือมีความผิดพลาด ถ้าข่าวสารที่รับได้ถูกต้อง ก็จะตอบรับด้วยอักษรควบคุม ACK (Acknowledge) ถ้าข่าวสารมีความผิดพลาด

ก็จะตอบปฏิเสธด้วย NAK (Negative Acknowledge) เพื่อให้ทางฝ่ายส่งทำการส่งข่าวสารซ้ำอีกครั้ง

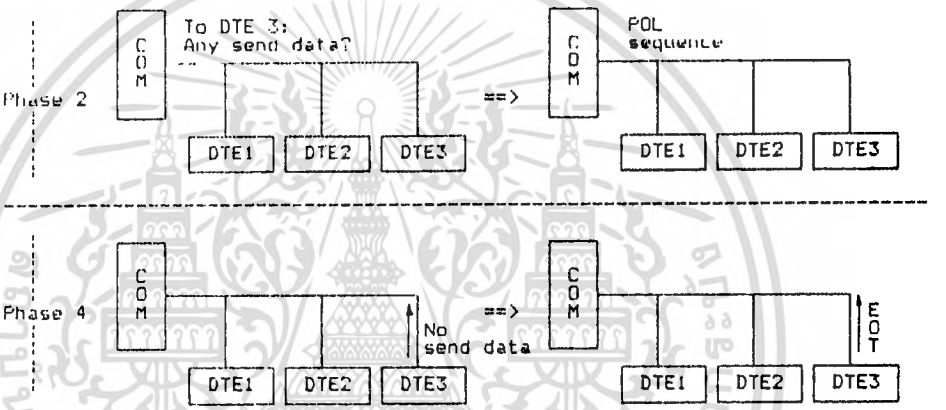


รูปที่ 2.3.24 การส่งผ่านข่าวสารและการตอบรับ

2.3.3 ขั้นตอนที่ 4 การสิ้นสุดการส่งข่าวสาร

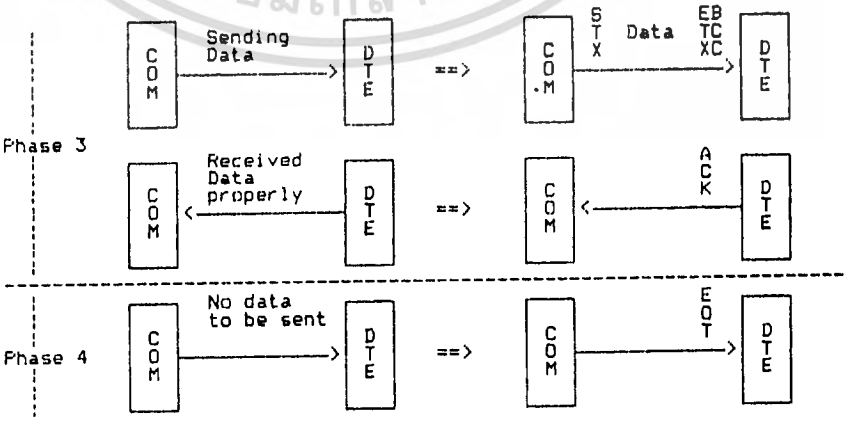
การแสดงถึงการสิ้นสุดการส่งข่าวสาร จะกระทำได้โดย ฝ่ายที่เป็นผู้ส่งข่าวสารจะส่งอักษรควบคุม EOT บอกทางฝ่ายผู้รับ เพื่อแสดงถึงการสิ้นสุดข่าวสาร

ในกรณีที่เทอร์มินัล เป็นผู้ส่งข่าวสาร และเมื่อข่าวสารสิ้นสุดลง หรือเกิดข้อขัดข้องที่ทำให้ไม่สามารถส่งข่าวสารต่อไปได้ ก็จะส่งอักษรควบคุม EOT เพื่อแสดงให้คอมพิวเตอร์รู้ ดังแสดงในรูปที่ 2.3.31



รูปที่ 2.3.31 การสิ้นสุดการส่งข่าวสารจากเทอร์มินัล

ในทำนองเดียวกันถ้าทางฝ่ายคอมพิวเตอร์เป็นผู้ส่งข่าวสาร เมื่อข่าวสารสิ้นสุดลงก็จะส่งสัญญาณเพื่อบอกให้เทอร์มินัล รับรู้ถึงการสิ้นสุดได้ ดังแสดงในรูป 2.3.32



รูปที่ 2.3.32 การสิ้นสุดการส่งข่าวสารจากคอมพิวเตอร์

2.4 ฟอร์แมทของการโพลลิ่งและซีเลคติ้ง

อักษรควบคุม ENQ จะเป็นตัวแสดงว่าเป็นสัญญาณควบคุม ในลักษณะการโพลลิ่ง หรือซีเลคติ้ง โดยมีรหัส SA (Station Address) และ UA (Unit Address) เป็นตัวบ่งบอกปลายทางและอุปกรณ์ที่อ้างถึง

2.4.1 การกำหนดรหัส SA

รหัส SA จะเป็นตัวกำหนดแอดเดรสของตัวควบคุมเทอร์มินัล (Terminal control unit หรือ TCU) การกำหนดรหัส SA ทั้งในการโพลลิ่ง และในบล็อคข่าวสารจะเป็นรหัสตัวเดียวกัน โดยมีวิธีการกำหนดขึ้นอยู่กับ ระบบของตัวกลางการสื่อสารในแต่ละแบบ ดังนี้

1) แบบ Point to Point ในระบบที่มีสายการสื่อสารอิสระ รหัส SA จะถูกกำหนดให้มีค่าคงที่ เท่ากับค่าของ SA0 ในตาราง ที่ 2.4.1

2) แบบ Multidrop ในระบบที่มีสารการสื่อสารอิสระ รหัส SA จะสามารถกำหนดได้ ดังตารางที่ 2.4.1 โดยเทอร์มินัลแต่ละตัวที่อยู่บนสายการสื่อสารเดียวกัน จะมีรหัส SA ไม่ซ้ำกัน

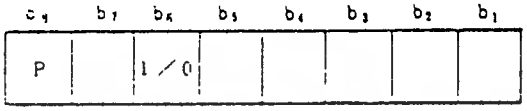
3) แบบ Switched Network รหัส SA จะสามารถกำหนดให้เป็นค่าใด ๆ ก็ได้แต่เทอร์มินัลที่อยู่ในกลุ่ม (System Group) เดียวกันจะต้องมีรหัส SA ที่มีค่าตรงกัน

	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1
SA1	P	0	1	1	0	0	0	0
SA2	P	0	1	1	0	0	0	1
SA3	P	0	1	1	0	0	1	0
SA4	P	0	1	1	0	0	1	1
SA5	P	0	1	1	0	1	0	0
SA6	P	0	1	1	0	1	0	1

ตารางที่ 2.4.1 รหัส SA

2.4.2 การกำหนดระหัส UA

รหัส UA จะเป็นตัวบ่งบอกถึงอุปกรณ์อินพุท หรือเอาท์พุท แต่ละตัวของเทอร์มินัลที่ถูกกำหนดโดย SA อีกทีหนึ่ง ข้อมูลในแต่ละบิทของรหัส UA จะมีหน้าที่ต่าง ๆ ดังนี้



บิท b8 : Parity bit

บิท b6 : "1" เป็นรหัส UA สำหรับอุปกรณ์ เอาท์พุท

"0" เป็นรหัส UA สำหรับอุปกรณ์ อินพุท

บิท b7,b5-b1 : หมายเลขของอุปกรณ์อินพุท เอาท์พุท

ในการกำหนดรหัส UA จะพิจารณาตามสภาวะในการทำงาน โดยแบ่งได้ดังนี้

1) UA ในการชี้เลคตั้งจากคอมพิวเตอร์มายังเทอร์มินัล

- UA จะบ่งบอกถึงอุปกรณ์เอาท์พุท
- UA ในการชี้เลคตั้ง ไม่จำเป็นจะต้องตรงกับในบล็อคของข่าวสาร
- UA ไม่จำเป็นจะต้องเป็นตัวกำหนดอุปกรณ์เอาท์พุทในการรับข่าวสาร จะเป็นแต่เพียงรหัสสำหรับตรวจสอบความพร้อม ของอุปกรณ์ตัวนั้น ๆ

2) UA ในการโพลลิ่ง

- UA จะบ่งบอกถึงอุปกรณ์อินพุท
- ทางด้านเทอร์มินัล เมื่อได้รับ UA ที่บ่งถึงอุปกรณ์อินพุท ก็จะแปลความหมายได้ว่าเป็นการโพลลิ่ง
- UA ในการโพลลิ่ง ไม่ได้เป็นตัวกำหนดอุปกรณ์อินพุท ที่จะต้องส่งข้อมูล แต่จะเป็นการสอบถามโดยรวม ว่าอุปกรณ์อินพุทตัวใด ต้องการที่จะส่งข้อมูล และ UA ในการโพลลิ่ง ไม่จำเป็นต้องตรงกับ UA ในบล็อคของข่าวสาร เช่นกัน

3) UA ในการชี้เลคตั้งจากเทอร์มินัล ในระบบ Contention

- UA จะบ่งบอกถึงอุปกรณ์อินพุท

- โดยปกติ UA ในการชี้เลคตั้งจะตรงกับ UA ในบล๊อคข่าวสาร แต่สำหรับเทอร์มินัลที่เป็นแบบ Cluster UA ดังกล่าว ก็ไม่จำเป็นจะต้องตรงกัน

4) UA ในการชี้เลคตั้งจากคอมพิวเตอร์ ในระบบ Contention จะมีลักษณะเดียวกับในหัวข้อ (1)

5) UA ในบล๊อคข่าวสารที่รับได้ จะบ่งบอกถึงอุปกรณ์เอาท์พุทสำหรับข่าวสารนั้น ๆ

6) UA ในบล๊อคของข่าวสารที่ถูกส่งออกไป จะบ่งบอกถึงอุปกรณ์อินพุทที่เป็นผู้ส่งข่าวสารนั้น ๆ

2.5 การรับส่งข่าวสารและการควบคุมการสื่อสาร

2.5.1 อักษรที่ใช้ในการควบคุมการสื่อสาร

ในข่าวสารที่มีการรับส่ง ระหว่างคอมพิวเตอร์ กับเทอร์มินัล แต่ละชุดจะประกอบด้วยส่วนที่เป็น Text ซึ่งอาจจะเป็นข่าวสาร หรือข้อความที่เป็นตัวหนังสือ หรือข้อมูลที่เป็นตัวเลขจะถูกแทนด้วยรหัสตามตารางมาตราฐาน JIS-7 (ดังแสดงในภาคผนวก) และอักษรควบคุมสำหรับควบคุมการสื่อสาร โดยจะถูกแทนด้วยรหัสที่ได้มีการกำหนดไว้โดยเฉพาะ ให้มีความหมายในการควบคุมการสื่อสาร และการรับส่งข่าวสาร ดังแสดงในตาราง 5.2.1

รหัสย่อ	เลขฐาน 16	ชื่อรหัส	ความหมาย
SOH	01H	Start of Heading	ใช้กำหนดจุดเริ่มต้นของ Heading ของข่าวสาร
STX	02H	Start of Text	ใช้กำหนดจุดเริ่มต้นของ Text
ETX	03H	End of Text	ใช้กำหนดจุดสิ้นสุดของ Text
EOT	04H	End of Transmittion	ใช้กำหนดจุดสิ้นสุดของข่าวสาร หรือการรับส่ง
ENQ	05H	Enquire	ใช้สอบถามสถานะของฝั่งตรงข้าม
ACK	06H	Acknowledge	ใช้ตอบรับ เพื่อแสดงความเรียบร้อย และถูกต้อง กลับไปยังฝั่งที่เป็นผู้สอบถาม
DLE	10H	Data Link Escape	ใช้นำหน้าอักษรตัวอื่น ๆ เพื่อเปลี่ยนความหมายของอักษรนั้น ๆ
NAK	15H	Negative Acknowledge	ใช้ปฏิเสธการสอบถาม แสดงความไม่เรียบร้อย หรือความผิดพลาดของข่าวสาร
SYN	16H	Synchronous Idle	ใช้ในการซิงโครไนซ์ ระบบการรับส่งข่าวสาร
ETB	17H	End of Transmittion	ใช้กำหนดจุดสิ้นสุดของข่าวสารในแต่ละบล๊อค ในกรณีที่ ข่าวสารถูกแบ่งออกเป็นหลายบล๊อค

ตาราง 2.5.1 รหัสควบคุมการสื่อสาร

2.5.2 ชุดคำสั่งควบคุมการสื่อสารของโปรโตคอล แบบ Level 2B

(1) Enquire sequenceใช้เป็นสัญญาณสอบถามในการโพลลิ่ง และซีเลคติ้ง เพื่อตรวจสอบความเรียบร้อยของฝ่ายตรงข้ามที่มีการติดต่อสื่อสารด้วย มีลักษณะฟอร์แมทดังแสดงในรูป

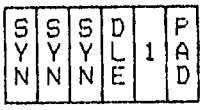
รูปที่ 2.5.21 Enquire sequence.

(2) Acknowledge sequence เป็นสัญญาณตอบรับในการซีเลคติ้ง และยืนยันความถูกต้องของบล็อคข่าวสารที่ได้รับ มี 2 ลักษณะคือ

(ก) DLE 0 ใช้สำหรับตอบรับแสดงความเรียบร้อยของอุปกรณ์เอาท์พุทเมื่อมีการซีเลคติ้งจากฝ่ายตรงข้าม เมื่อฝ่ายตรงข้ามต้องการส่งข่าวสาร และใช้แสดงการยืนยันความถูกต้องของข่าวสารที่ได้รับ สำหรับบล็อคของข่าวสารที่มีลำดับเป็นจำนวนคู่ เช่น บล็อคที่ 2,4,6,... เป็นต้น

รูปที่ 2.5.22 DLE 0.

(ข) DLE 1 มีความหมายในการใช้งานเช่นเดียวกันกับ DLE 1 แต่แตกต่างกันที่ใช้ยืนยันความถูกต้อง ของบล็อคข่าวสาร ที่มีลำดับเป็นจำนวนคี่ เช่น บล็อคที่ 1,3,5,... เป็นต้น

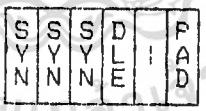


รูปที่ 2.5.23 DLE 1

(6) Wait before transmission sequence (WABT) DLE ? ใช้สำหรับให้ทางฝ่ายรับส่งไปยังฝ่ายผู้ส่ง เพื่อพักการส่งชั่วขณะ เนื่องจากในขณะนั้นอุปกรณ์เอาท์พุท ของทางฝ่ายรับ ยังไม่พร้อมที่จะรับข้อมูลบล็อคต่อไป เช่น การส่งข่าวสารไปพิมพ์ยังเครื่องพิมพ์ ซึ่งเป็นอุปกรณ์เอาท์พุท ที่ทำงานช้าก็จำเป็นต้องใช้สัญญาณนี้ในการควบคุมการส่งข่าวสารจากทางฝ่ายผู้ส่ง

รูปที่ 2.5.27 WABT sequence.

(7) Break sequence DLE : ใช้สำหรับให้ทางฝ่ายรับ ส่งไปบอกทางฝ่ายส่ง เพื่อให้ยุติการส่ง ในกรณีที่อุปกรณ์เอาท์พุท ทางฝ่ายรับเกิดเหตุขัดข้อง ไม่สามารถทำการรับข่าวสารอีกต่อไปได้



รูปที่ 2.5.28 Break sequence

จะสังเกตุได้ว่า อักษรตัวสุดท้ายสำหรับทุก ๆ สัญญาณ จะเป็นอักษร PAD ซึ่งจะมีรหัสเป็น FFH (เลขฐาน 16) หรือมีค่าเป็น 1 ทั้ง 8 บิท มีไว้เพื่อการตรวจสอบความถูกต้องของสัญญาณแต่ละชุด เนื่องจากสัญญาณดังกล่าวนี้ประกอบด้วยชุดอักษรเพียงสั้น ๆ และไม่มีการตรวจสอบด้วยการใช้ BCC ใน PAD นี้ ทางฝ่ายรับ จะตรวจสอบเพียง 4 บิทแรก เพื่อความแน่ใจว่าเป็น ลักษณะของการสิ้นสุดของสัญญาณแต่ละชุดเท่านั้น

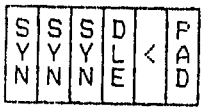
(3) Negative acknowledge sequence NAK มีความหมายในทางตรงกันข้ามกับ Acknowledge sequence ในข้อ (ก) กล่าวคือใช้เป็นสัญญาณในการปฏิเสธ เพื่อแสดงว่าในขณะที่มีการซีเลคติ้ง อุปกรณ์เอาท์พุท ที่ถูกอ้างถึง ยังไม่พร้อมที่จะรับข้อมูลได้ และใช้ในการรายงานผลการรับข่าวสาร เมื่อข่าวสารที่ได้รับมีความผิดพลาด

รูปที่ 2.5.24 Negative acknowledge sequence.

(4) End of transmission sequence EOT มีความหมายในการใช้งานได้สองลักษณะ คือ ใช้แสดงการสิ้นสุดการส่งผ่านข่าวสาร หรือมีเหตุขัดข้องที่ทำให้ทางฝ่ายที่กำลังส่งไม่สามารถทำการส่งต่อไปได้ และใช้แสดงถึงสภาวะปกติ เมื่อมีการโพลลิ่ง และทางด้านเทอร์มินัล ยังไม่พร้อมที่จะส่งข่าวสาร

รูปที่ 2.5.25 End of transmission sequence.

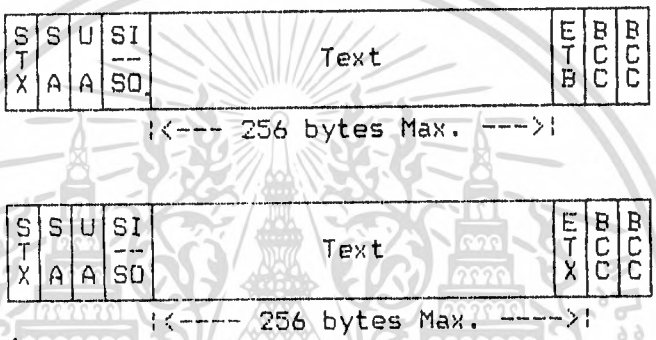
(5) Interupt sequence DLE < มีไว้เพื่อให้ทางฝ่ายที่กำลังเป็นผู้รับข่าวสารส่งไปร้องขอ เพื่อให้ทางฝ่ายส่งหยุดการส่งชั่วคราว ในกรณีที่ทางฝ่ายรับมีข่าวสารที่ต้องการส่งในขณะนั้น



รูปที่ 2.5.26 Interupt sequence

2.5.3 อักษรควบคุมที่ใช้ในบล๊อคข่าวสาร

ดังที่ได้กล่าวมาบ้างแล้วในหัวข้อ 2.3.2 ข่าวสารจะถูกแบ่งออกเป็นบล๊อค ๆ และส่งไปทีละบล๊อค ในโปรโตคอลแบบ Level 2B จะมีการเพิ่มอักษรควบคุมเข้าไปในตอนต้นของส่วนที่เป็น Text เพื่อใช้ในการกำหนดอุปกรณ์ และเลือกชุดอักษร ลงในข้อมูลแต่ละบล๊อค ดังแสดงในรูป 2.5.3



รูปที่ 2.5.3 อักษรควบคุมที่ใช้ในบล๊อคข่าวสาร

STX เป็นตัวบอกถึงการเริ่มต้นของ Text ในแต่ละบล๊อค

SA,UA เป็นตัวบอกแอดเดรสของปลายทาง และหมายเลขของอุปกรณ์อินพุท เอาท์พุท ดังได้กล่าวมาแล้วในหัวข้อ 2.4.1 และ 2.4.2

SI/SO ใช้ในการเลือกชุดของรหัสที่จะใช้ในส่วนที่เป็น Text ตามรหัสมาตราฐาน JIS-7.

ETB เป็นตัวแสดงการสิ้นสุดของ Text ในแต่ละบล๊อค

ETX เป็นตัวแสดงการสิ้นสุดของ Text ในแต่ละบล๊อค และบอกถึงการสิ้นสุดของข่าวสารแต่ละชุดด้วย

BCC เป็นรหัสกำกับ ใช้ในการตรวจสอบความผิดพลาด แบบ CRC ในการรับส่งข่าวสาร

2.5.4 ระบบการตรวจสอบความผิดพลาดของข่าวสาร

ระบบโปรโตคอล แบบ Level 2B ได้กำหนดให้มีการตรวจสอบความผิดพลาดไว้ 2 ลักษณะ คือ

1) การตรวจสอบพาริตี้ (Parity Checking) ในแต่ละไบท์ของข้อมูล บิทสูงสุด (บิทที่ 8) ของข้อมูลจะถูกกำหนดให้เป็นพาริตี้บิท และมีการตรวจสอบแบบ Odd parity.

2) การตรวจสอบ CRC (Cyclic Redundancy Checking) ในแต่ละบล็อคของข่าวสารจะมีการสร้างรหัสสำหรับตรวจสอบแบบ CRC โดยใช้โพลีโนเมียล $X^{16} + X^{12} + X^5 + 1$ การคำนวณหาค่า CRC ในแต่ละบล็อค จะเริ่มต้นการคำนวณหลังจากรหัส STX คือจะเริ่มจาก SA, UA เรื่อยไปจนถึง ETX หรือ ETB และผลลัพท์ที่ได้ซึ่งมีความยาว 2 ไบท์ จะถูกส่งรวมไปกับข่าวสารแต่ละบล็อค เป็น BCC (Block Check Character) ดังแสดงในหัวข้อที่ 2.5.3 ส่วนทางฝ่ายรับ เมื่อรับข่าวสารแต่ละบล็อค ก็จะทำการคำนวณ CRC ตรวจสอบผลลัพท์อีกครั้งหนึ่ง เพื่อดูว่าเกิดความผิดพลาดขึ้นในข่าวสารที่รับได้หรือไม่ ตามขั้นตอนในการคำนวณ CRC ดังแสดงในภาคผนวก ก.

หมายเหตุ

[1] ในระบบโปรโตคอล แบบ Level 2B ส่วนที่เป็น Text จะเป็นสาระสำคัญของข่าวสารในการส่ง และอักษรควบคุม (ดังแสดงในหัวข้อ 2.5.3) ส่วน Heading จะเป็นข้อมูลในการควบคุมย่อย (Auxillary Function) สำหรับการส่งข่าวสาร เช่น หมายเลขของข่าวสาร แบบฟอร์ม และลำดับความสำคัญข่าวสาร เป็นต้น ซึ่งจะมีใช้กับในเทอร์มินัลเฉพาบางรุ่นเท่านั้น (ไม่มีใช้ในรุ่น NEC 6300/50N ที่ติดตั้งที่สำนักวิจัยฯ)

บทที่ 3

การวิเคราะห์โปรโตคอลของคอมพิวเตอร์ NEC ACOS-300

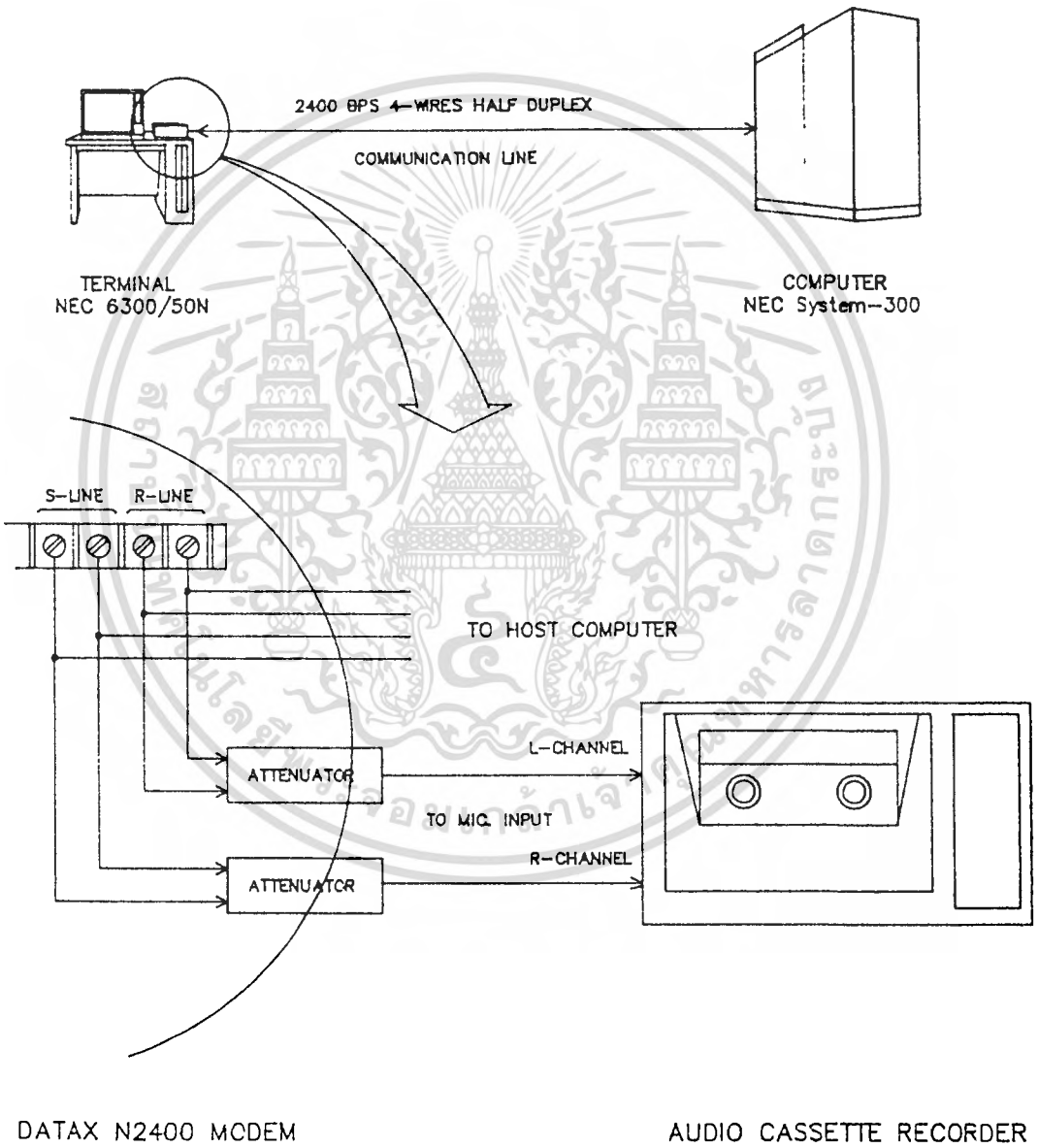
โดยทั่วไป ในการวิเคราะห์โปรโตคอลในการสื่อสารข้อมูลของคอมพิวเตอร์แต่ละระบบจะสามารถทำได้ โดยการใช้เครื่องมือที่เรียกว่า เครื่องวิเคราะห์โปรโตคอล (Protocol analyzer) แต่สำหรับในงานวิจัยครั้งนี้ มีความประสงค์ที่จะทำการศึกษาในรายละเอียดเกี่ยวกับการสื่อสารข้อมูลในแต่ละขั้นตอน อีกทั้งเครื่องมือดังกล่าว มีใช้น้อย และมีราคาแพงจึงได้จัดเตรียมการวิเคราะห์ขึ้นเอง เท่าที่เครื่องมือที่มีอยู่ในห้องทดลองของสำนักวิจัย ฯ จะเอื้ออำนวย โดยได้แบ่งการวิเคราะห์ออกเป็นขั้นตอนดังนี้.

3.1 การเก็บตัวอย่างสัญญาณในการสื่อสารของคอมพิวเตอร์ NEC ACOS-300 กับเทอร์มินัล

ในการวิเคราะห์ จำเป็นที่จะต้องมีตัวอย่างสัญญาณของการรับส่งข้อมูลเก็บไว้ในอุปกรณ์ที่สามารถเรียกกลับมาดูได้เมื่อต้องการ โดยไม่จำเป็นต้องมีการเฝ้าสังเกตุจากการรับส่งข้อมูลจริง เพื่อที่จะได้มีอิสระในการวิเคราะห์ และไม่ต้องเดินเครื่องคอมพิวเตอร์ตลอดเวลา ในขณะการวิเคราะห์ต้องใช้เวลาค่อนข้างยาวนาน การเก็บข้อมูลตัวอย่างทำได้ง่าย ๆ โดยการใช้เทปบันทึกเสียงสเตริโอ แบบกระเป๋าหิ้วขนาดเล็ก ทำการบันทึกสัญญาณความถี่เสียงจากโมเดม ที่เชื่อมระหว่างคอมพิวเตอร์ กับเทอร์มินัล ดังรูป 3.1.1 โดยให้ช่องเสียงทางซ้ายเป็นสัญญาณของข้อมูล ที่ส่งจากคอมพิวเตอร์ไปยังเทอร์มินัล และให้ช่องเสียงทางขวา เป็นสัญญาณของข้อมูลที่ส่งจากเทอร์มินัล ไปยังคอมพิวเตอร์

โมเดมที่ใช้เชื่อมต่อระหว่างเครื่องคอมพิวเตอร์ เป็นรุ่น NEC DATAX N2400 ซึ่งสามารถปรับความแรงของสัญญาณในการส่งได้ ตั้งแต่ 0 ถึง -31 dBm. แต่เนื่องจากในขณะที่มีการใช้งานจะถูกตั้งไว้ให้มีความแรงของสัญญาณคงที่ ที่ -16 dBm. จะมีระดับโวล์ทเตจประมาณ 120 mV. ที่ 600 โอห์ม ซึ่งมีความแรงมากเกินกว่าที่เครื่องบันทึกเสียงจะรับได้ดังนั้น จึงต้องใช้วงจรช่วยในการลดทอนสัญญาณ ให้มีระดับ โวล์ทเตจ อยู่ในช่วงที่เครื่องบันทึกเสียงรับได้ คือ ไม่เกิน 50 mV.

การบันทึกสัญญาณการสื่อสารติดต่อระหว่างคอมพิวเตอร์ กับเทอร์มินัล เริ่มตั้งแต่มีการติดต่อและบันทึกการรับส่งข้อมูลไปกลับจนกระทั่งเลิกการติดต่อ ก็จะได้สัญญาณของการรับส่งข้อมูล เพื่อไว้ใช้สำหรับการวิเคราะห์ในขั้นตอนต่อไป

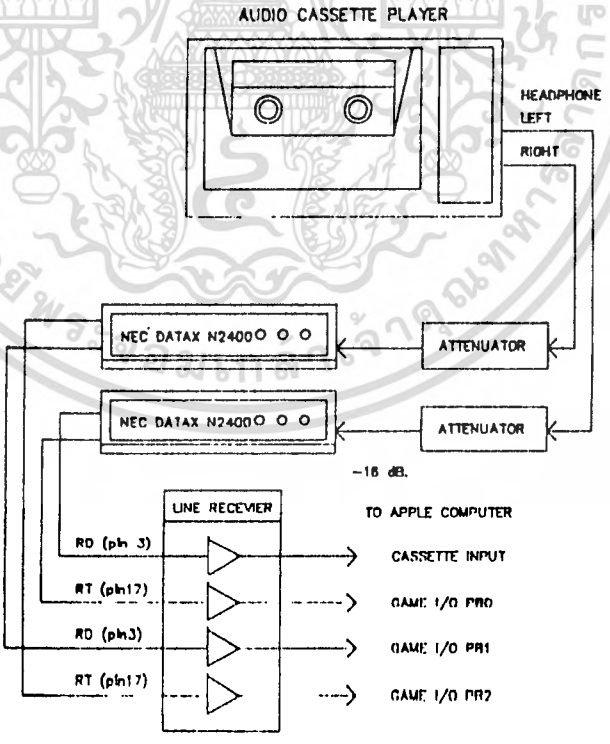


รูป 3.1.1 แสดงการเก็บตัวอย่างสัญญาณของการรับส่งข้อมูล.

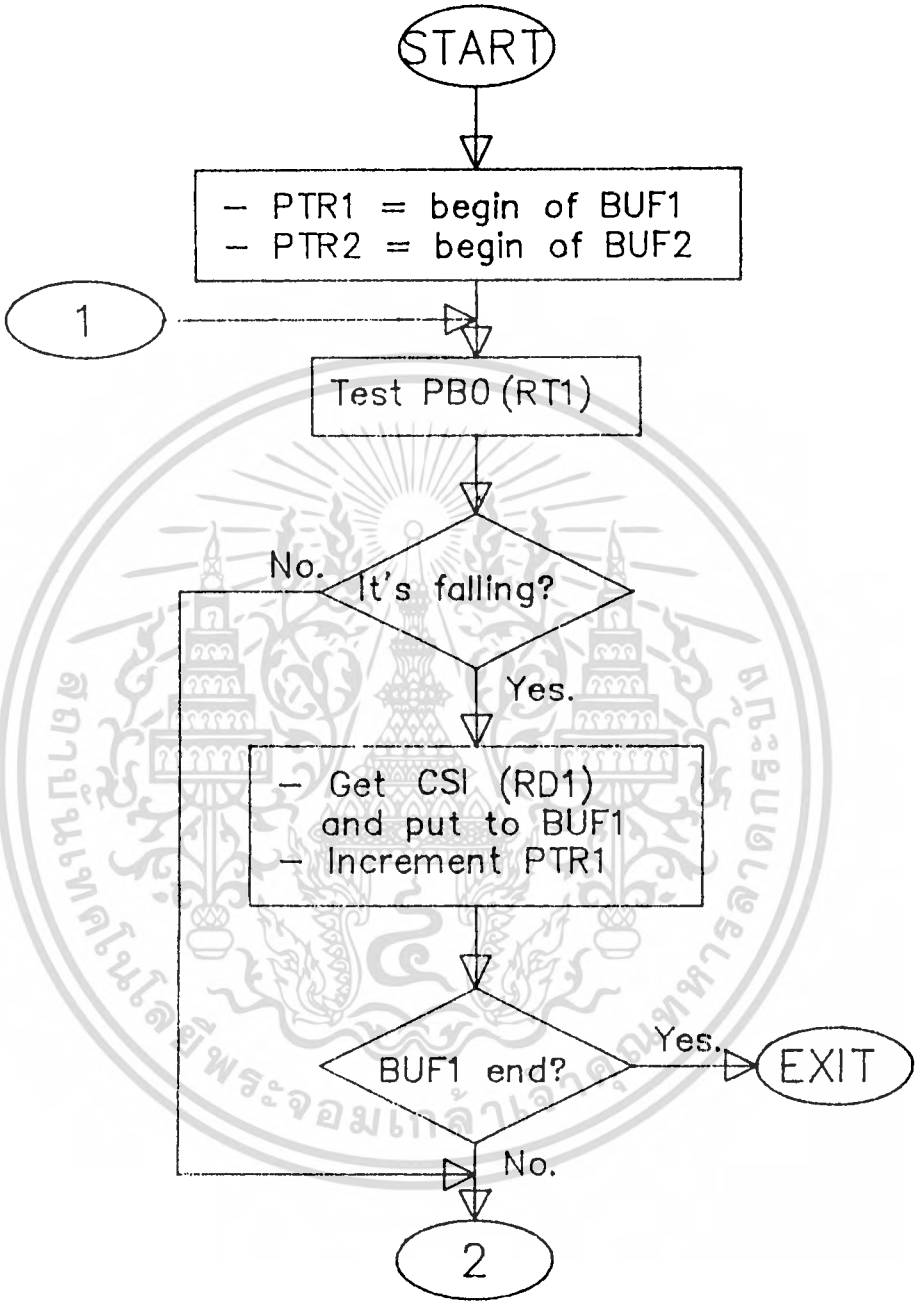
3.2 แปลงสัญญาณข้อมูลในรูปความถี่เสียงเป็นข้อมูลดิจิตอล.

เนื่องจากในขั้นตอนแรก จะได้ตัวอย่างการรับส่งข้อมูลมาในรูปของความถี่เสียง ที่เป็นการมอ-ดูเลทของข้อมูล ตามมาตราฐาน CCITT. แบบ V.26bis. จากโมเดมดังกล่าว และข้อมูลที่เราต้องการมาใช้ในการวิเคราะห์ คือ ข้อมูลดิจิตอล ดังนั้นการจะแปลงสัญญาณความถี่เสียงดังกล่าวมาเป็นข้อมูลจะทำได้โดยการใช้โมเดม 2 ตัว โดยตัวแรกใช้แปลงสัญญาณความถี่เสียงจากเครื่องเล่นเทปช่องซ้าย ซึ่งข้อมูลที่ได้ จะเป็นข้อมูลของการส่ง จากคอมพิวเตอร์ สู่เทอร์มินัล และโมเดมตัวที่สอง ต่อกับสัญญาณเสียงทางช่องขวา ซึ่งจะเป็นข้อมูลที่ส่ง จากเทอร์มินัล สู่คอมพิวเตอร์

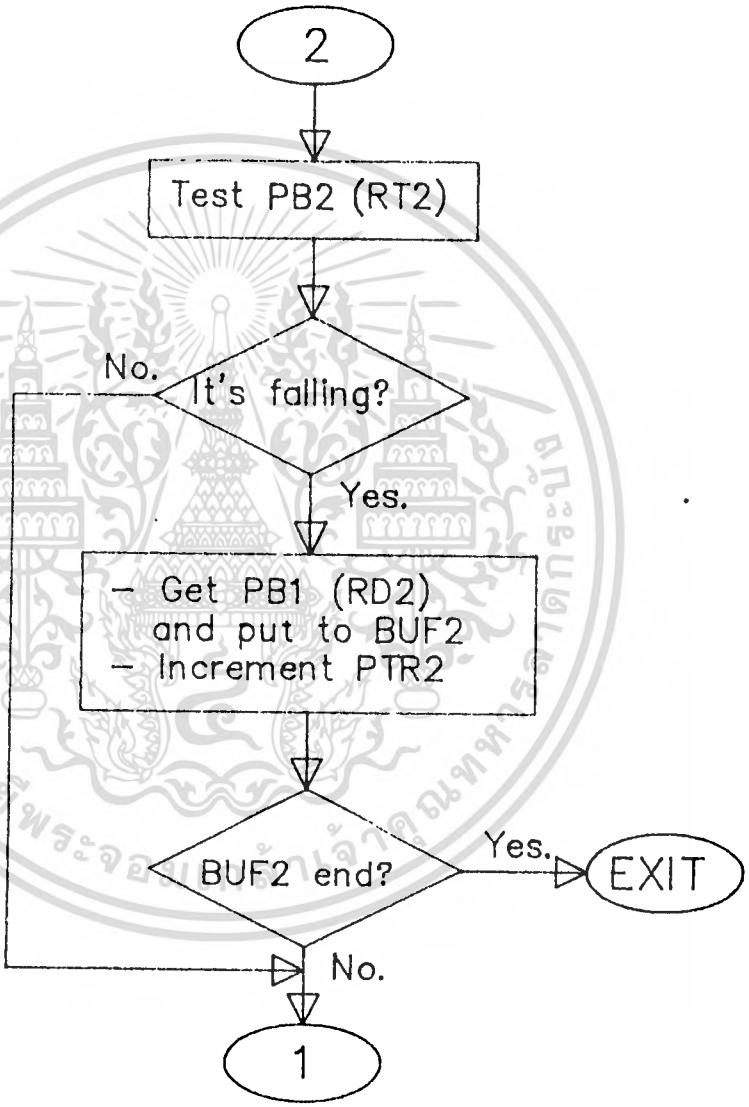
ข้อมูลจากโมเดมทั้งสอง จะป้อนเข้าสู่ไมโครคอมพิวเตอร์ โดยใช้เครื่อง AppleII+ เพื่อที่จะได้เก็บบันทึกข้อมูลดิจิตอล ไว้ในแผ่นดิสเกตต์ เพื่อใช้ในการวิเคราะห์ดังรูป 3.2.1 โปรแกรมในการเก็บบันทึกข้อมูล ดังแสดงในรูป 3.2.2 และตัวอย่างข้อมูลที่ได้ ดังแสดงในรูป 3.2.3.



รูปที่ 3.2.1 แสดงการจัดอุปกรณ์สำหรับการแปลงสัญญาณข้อมูลเป็นข้อมูลดิจิตอล



รูปที่ 3.2.2 โฟล์วชาร์ทในเก็บข้อมูลอนุกรมจากโมเดม



รูปที่ 3.2.2 (ต่อ)

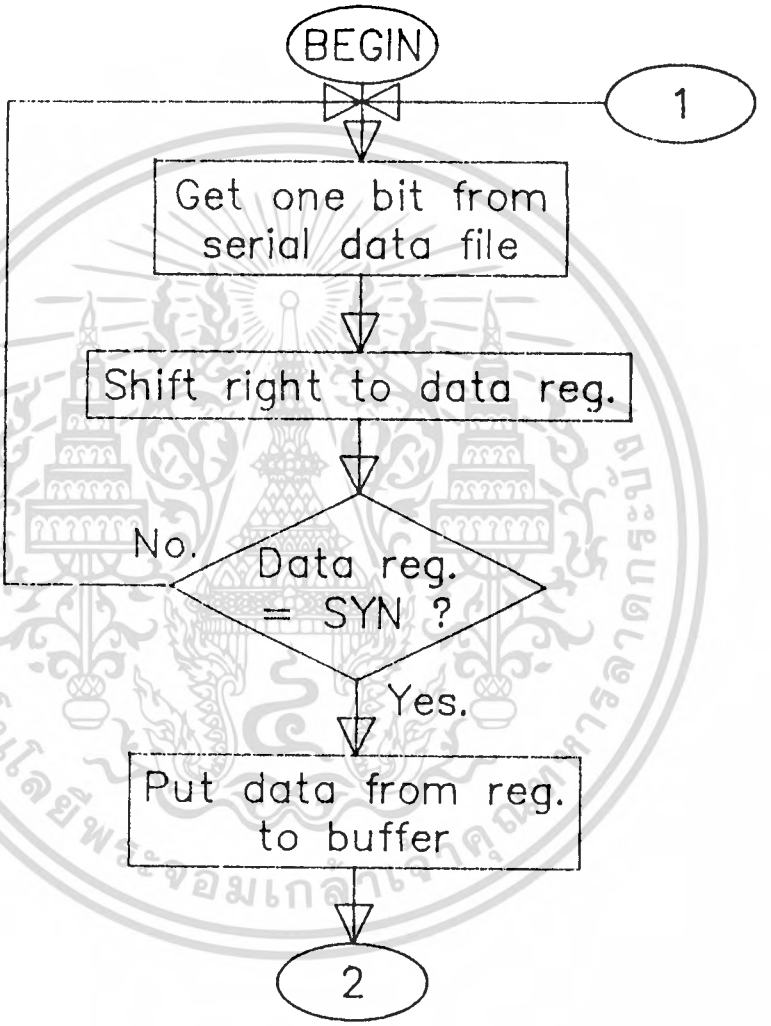
```
0100  31 31 31 31 31 31 31 31-31 31 31 31 31 31 31 31   1111111111111111
0110  30 30 30 30 31 30 30 30-30 31 31 30 31 30 30 30   0000100001101000
0120  30 31 31 30 31 30 30 30-30 30 31 30 30 30 30 30   0110100000100000
0130  31 31 31 31 31 31 31 31-31 31 31 31 31 31 31 31   1111111111111111
0140  31 30 30 30 30 30 30 30-30 31 31 31 31 31 31 31   1000000011111111
                                                        PAD
0150  31 30 31 31 30 31 30 30-30 30 31 31 30 31 30 30   1011010000110100
                                                        16H(SYN) 16H(SYN)
0160  30 30 31 31 30 31 30 30-30 30 30 30 30 31 31 30   0011010000000110
                                                        16H(SYN) B0(SA)
0170  31 30 30 30 30 30 30 31-30 31 30 31 30 30 30 30   1000000101010000
                                                        40H(DA) 35H(ENQ)
0180  31 31 31 31 31 31 31 31-31 31 31 31 31 31 31 31   1111111111111111
                                                        PAD
```

รูปที่ 3.2.3 ตัวอย่างข้อมูลอนุกรมแบบ ซิงโครนัส

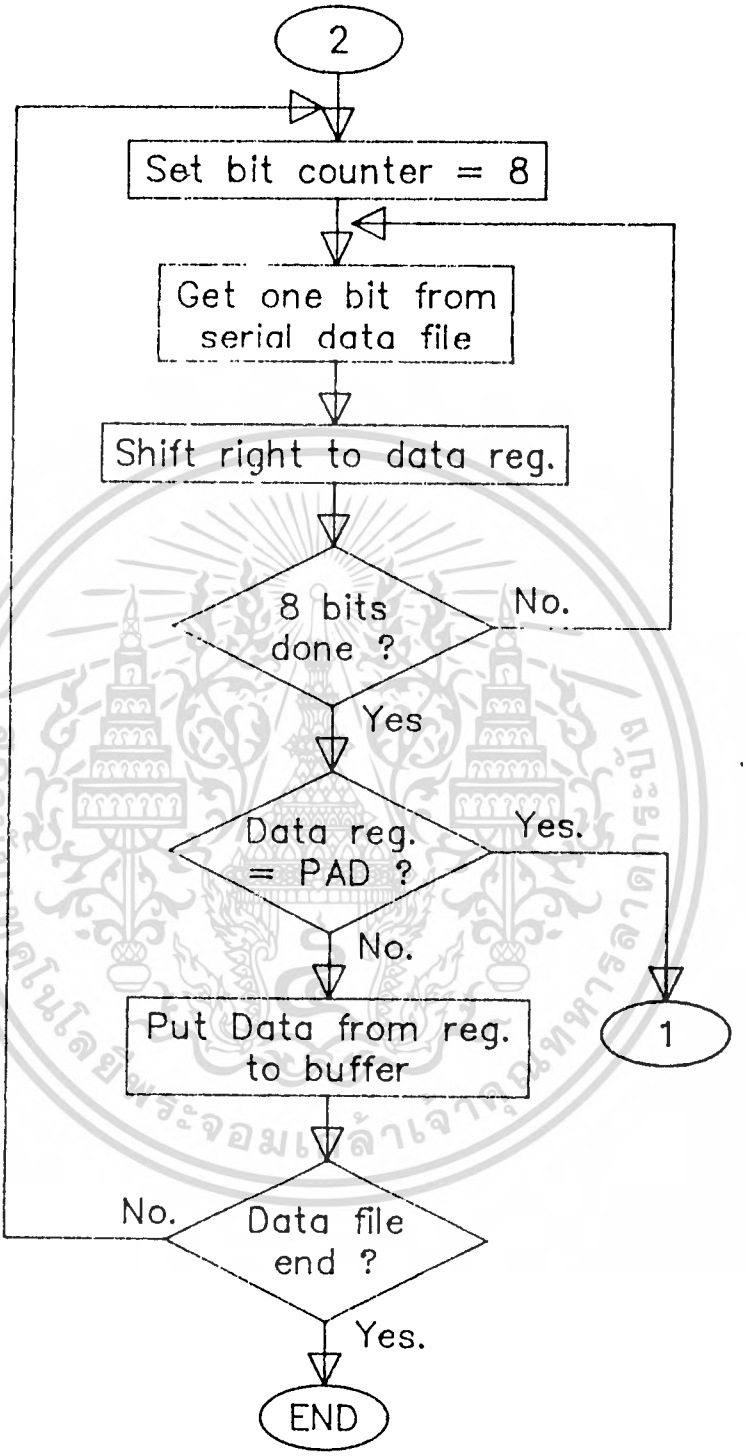
3.3 แปลงข้อมูลดิจิตอลแบบไบนารี่ให้อยู่ในรูปของรหัส ASCII.

จากขั้นตอนที่สองจะวิเคราะห์รายละเอียดในบางส่วนได้ดังรูป 3.2.3 ซึ่งจะเห็นว่ารหัสที่ใช้ในการสื่อสารของเครื่อง NEC ACOS-300 ซึ่งเป็นแบบมาตราฐาน JIS-7 มีลักษณะคล้ายกับรหัสมาตราฐาน ASCII และใช้อักษรในการซิงโครไนซ์ เป็นอักษร SYN (Sync character 16H) ในตารางรหัสมาตราฐาน JIS 7 และเป็นการสื่อสารแบบ Character Oriented protocol

ดังนั้น จะสามารถสร้างโปรแกรมในการแยกรหัส และอักษรในการรับส่งข้อมูลได้ ดังแสดงในรูปที่ 3.3.1 และ ตัวอย่างรหัสที่ได้ในรูป 3.3.2



รูปที่ 3.3.2 โฟล์วชาร์ทแสดงการแปลงข้อมูลอนุกรมแบบ ซิงโครนัส เป็นรหัส ASCII



รูปที่ 3.3.2 (ต่อ)

ก). ข้อมูลที่ส่งจากเครื่องคอมพิวเตอร์

```
1000  16 16 16 04 16 16 16 B0-40 85 16 16 16 04 16 16   .......0@.......
1010  16 B0 40 85 16 16 16 04-16 16 16 B0 40 85 16 16   .0@.........0@...
1020  16 10 31 16 16 16 B0 A8-85 16 16 16 02 B0 A8 8F   ..1...0(.....0(.
1030  9B B5 D0 C1 32 B0 BA B5-31 20 20 CD 43 CE 57 B9   .5PA20:51  MCNW9
1040  B0 20 20 20 20 20 20 20-43 4F CE 52 45 51 20 43   0       CONREQ C
1050  4F CD D0 4C 45 54 45 C4-83 70 7C 16 16 16 04 16   OMPLETED.p|.....
1060  16 16 04 16 16 16 B0 40-85 16 16 16 04 16 16 16   .......0@.......
1070  B0 40 85 16 16 16 04 16-16 16 B0 40 85 16 16 16   0@.........0@....
1080  04 16 16 16 B0 40 85 16-16 16 04 16 16 16 B0 40   ....0@.........0@
1090  85 16 16 16 04 16 16 16-B0 40 85 16 16 16 04 16   .........0@......
10A0  16 16 B0 40 85 16 16 16-04 16 16 16 B0 40 85 16   ..0@.........0@..
```

ข). ข้อมูลที่ส่งจากเทอร์มินัล

```
2000  FB 16 16 16 04 16 16 16-04 16 16 16 04 16 16 16   {...............
2010  04 16 16 16 04 16 16 16-04 3F F7 16 16 16 04 16   .........?w.....
2020  16 16 04 16 16 16 04 7F-16 16 16 04 7F 16 16 16   ................
2030  02 B0 C8 8F CB AB 20 89-DC 43 4F CE 20 D0 43 46   .0H.K+ .\CON PCF
2040  83 42 69 16 16 16 04 16-15 16 10 B0 16 16 16 10   .Bi.........0....
2050  31 16 16 16 04 16 16 16-04 16 16 16 04 16 16 16   1...............
2060  04 16 16 16 04 16 16 16-04 7F FE 16 16 16 04 16   ............~.....
2070  16 16 04 16 16 16 04 7F-E9 16 16 16 04 7F AD 16   .........i.....-.
2080  16 16 04 7F 16 16 16 04-7F 16 16 16 04 16 16 16   ................
2090  04 16 16 16 04 7F E7 16-16 16 04 7F DD 16 16 16   ......g.....]...
```

รูปที่ 3.3.1 ตัวอย่างข้อมูลในการสื่อสารแบบ ซิงโครนัส

3.4 รวบรวมและจัดกลุ่มของรหัสที่ใช้ในการควบคุมการสื่อสาร

จากขั้นตอนที่ 3 จะเห็นว่าการสื่อสารข้อมูลแบบ Level 2B ในเครื่อง NEC ACOS-300 มีลักษณะใกล้เคียงกับการสื่อสารข้อมูลแบบ BISYNC ของบริษัท IBM มาก ซึ่งช่วยให้การวิเคราะห์ทำได้ง่ายและรวดเร็วขึ้น ซึ่งรหัสควบคุมต่าง ๆ ที่รวบรวมได้เปรียบเทียบกันในแต่ละฟังก์ชั่นการทำงาน ดังแสดงเปรียบเทียบกับแบบ BISYNC ได้ดังตารางที่ 3.4 และจะใช้ในการออกแบบในขั้นตอนต่อไป

Communication Control Function	IBM BISYNC	NEC LEVEL 2B
Synchronous Idle	SYN	SYN
Start of Heading	SOH	SOH
Start of Text	STX	STX
End of Transmission Block	ETB	ETB
End of Text	ETX	ETX
End of Transmission	EOT	EOT
Enquiry	ENQ	ENQ
Affirmative Acknowledgment	DLE 0 / DLE 1	DLE 0 / DLE 1
Negative Acknowledgment	NAK	NAK
Transparent Text Escape Code	DLE	-
End of Intermediate Transmission Block	ITB	-
Wait Before Transmit Positive Ack. (WACK)	DLE ;	-
Wait before transmit Negative Ack. (WABT)	-	DLE ?
Reverse Interupt (RVI)	DLE <	DLE <
Temporary Text Delay (TTD)	STX ENQ	-
Break Transmission	-	DLE !
Disconnect	DLE EOT	DLE EOT

ตารางที่ 3.4 แสดงเปรียบเทียบฟังก์ชั่นควบคุมการสื่อสาร ของโปรโตคอล

แบบ BISYNC ของ IBM กับแบบ LEVEL 2B ของ NEC

บทที่ 4

การออกแบบโปรโตคอล คอนเวอร์เตอร์

จากรายละเอียดและข้อมูลที่ได้กล่าวมาแล้วในบทที่ 2 และ 3 จะเห็นว่าโปรโตคอล สำหรับการสื่อสารข้อมูล ของคอมพิวเตอร์ NEC ACOS-300 มีกฎเกณฑ์ในการรับส่งข้อมูลและการตรวจสอบความผิดพลาดของข้อมูลต่าง ๆ ที่แน่นอน แตกต่างกับระบบการสื่อสารบนเครื่องไมโครคอมพิวเตอร์ทั่ว ๆ ไป ซึ่งส่วนใหญ่จะเป็นแบบ อะซิงโครนัส และไม่มีโปรโตคอล การสื่อสารที่แน่นอน หรือ มีเฉพาะสำหรับการใช้งานเฉพาะอย่าง ซึ่งจะไม่สามารถนำมาติดต่อสื่อสารกับคอมพิวเตอร์ NEC ACOS-300 โดยตรงได้

ดังนั้น ขั้นตอนที่สองของการวิจัย คือ การออกแบบโปรโตคอล คอนเวอร์เตอร์ ซึ่งจะเป็นตัวกลางในการติดต่อ ระหว่างเครื่อง NEC ACOS-300 กับไมโครคอมพิวเตอร์ดังกล่าว เพื่อทำหน้าที่ในการแปลงโปรโตคอล ของเครื่อง NEC ACOS-300 ให้อยู่ในรูปของชุดข้อมูลที่เหมาะสมสำหรับไมโครคอมพิวเตอร์ และจัดข้อมูลจากไมโครคอมพิวเตอร์ ให้เป็นโปรโตคอลเดียวกับเครื่อง NEC ACOS-300.

ในการออกแบบโปรโตคอล คอนเวอร์เตอร์ นี้ได้กำหนดลักษณะที่จะออกแบบ เพื่อให้อุปกรณ์ชิ้นนี้มีความเหมาะสม สำหรับการศึกษาและพัฒนา ไว้ดังนี้

- เป็นระบบที่มีไมโครโปรเซสเซอร์ควบคุมการทำงานโดยอิสระ เพื่อให้สะดวกในการดัดแปลงเข้ากับไมโครคอมพิวเตอร์เครื่องอื่นๆ
- โปรแกรมควบคุมการแปลงโปรโตคอล จะขึ้นอยู่กับไมโครโปรเซสเซอร์ควบคุมเพียงตัวเดียว ไม่ต้องออกแบบโปรแกรมใหม่ เมื่อใช้เชื่อมกับไมโครคอมพิวเตอร์ อื่น ๆ
- โปรแกรมควบคุมเป็นลักษณะโมดูล่า เพื่อง่ายต่อการตรวจสอบและพัฒนาโปรแกรม สามารถพัฒนาฮาร์ดแวร์ได้ง่าย โดยใช้เครื่องมือพื้นฐานที่มีอยู่ เช่น ไมโครคอมพิวเตอร์ ออสซิ-โลสโคป และเทอร์มินัล เป็นต้น
- สามารถนำไปประยุกต์ใช้งานได้ง่าย

4.1 การออกแบบส่วนฮาร์ดแวร์ของโปรโตคอล คอนเวอร์เตอร์

ในการออกแบบวงจรส่วนนี้ สิ่งที่สำคัญที่สุดที่จะต้องพิจารณาเป็นอันอับแรก คือ อุปกรณ์สำหรับการแปลงข้อมูลระหว่างอนุกรมและขนาน หรือ USART (Universal Synchronous/Asynchronous Receiver Transmitter) ซึ่งต้องเป็นชนิดที่เหมาะสม คือสามารถทำงานในแบบซิงโครนัส ดังกล่าวมาแล้วได้ และการใช้งานร่วมกับไมโครโปรเซสเซอร์ที่จะเลือกใช้ จะต้องไม่ยุ่งยากจนเกินไป และสิ่งรองลงมาในการพิจารณา คือ ไมโครโปรเซสเซอร์ หรือ CPU จะต้องเป็นแบบที่สามารถพัฒนาโปรแกรมได้ง่าย และใช้อุปกรณ์ประกอบในการจัดระบบไม่มากนัก

จากหลักเกณฑ์ในการพิจารณาข้างต้น อุปกรณ์ที่เห็นว่าเหมาะสมกับการใช้งาน คือ ใช้อุปกรณ์แปลงข้อมูลอนุกรม แบบ Z80 SIO (Serial Input/Output interface Unit) และไมโคร-โปรเซสเซอร์ แบบ Z80 CPU ซึ่งการที่อุปกรณ์ที่เลือกได้ทั้งสองอย่างอยู่ในตระกูลเดียวกัน ทำให้การออกแบบสะดวกยิ่งขึ้น

ระบบที่ได้ออกแบบเลือกใช้ไมโครคอมพิวเตอร์ แบบ AppleII+ พร้อมชุดประกอบสำหรับการทำงานภายใต้โปรแกรมจัดการระบบ แบบ CP/M 80 ซึ่งจะใช้ในการพัฒนาโปรแกรมควบคุมของส่วนโปรโตคอล คอนเวอร์เตอร์ เองด้วย โดยมีส่วนประกอบ ดังแสดงด้วยบล๊อคไดอะแกรม ในรูปที่ 4.1

4.1.1 CPU. หน่วยความจำแบบ ROM และ RAM.

สามส่วนนี้ใช้ประกอบร่วมกันในการจัดการและควบคุม โดย CPU จะเป็นผู้ดำเนินการตามโปรแกรม ที่อยู่ในหน่วยความจำแบบ ROM ส่วนหน่วยความจำแบบ RAM ใช้ในการเก็บโปรแกรมควบคุมในขณะที่อยู่ระหว่างการพัฒนา และเก็บข้อมูลในการควบคุมเฉพาะบางส่วนเนื่องจากข้อมูลส่วนใหญ่ CPU สามารถนำไปเก็บในหน่วยความจำของไมโครคอมพิวเตอร์ ได้โดยตรง

4.1.2 ส่วนอินเทอร์เฟสกับไมโครคอมพิวเตอร์

เป็นส่วนที่ออกแบบเป็นพิเศษ ให้เหมาะสำหรับการพัฒนาโปรแกรมควบคุม และตรวจสอบการทำงานของระบบ ด้วยการออกแบบให้ CPU ที่ควบคุมการทำงานของส่วนคอนเวอร์เตอร์ สามารถถ่ายโปรแกรมที่ถูกพัฒนาขึ้นบนไมโครคอมพิวเตอร์ เข้ามายังหน่วยความจำแบบ RAM และเข้าไปทำงานในโปรแกรมดังกล่าวได้

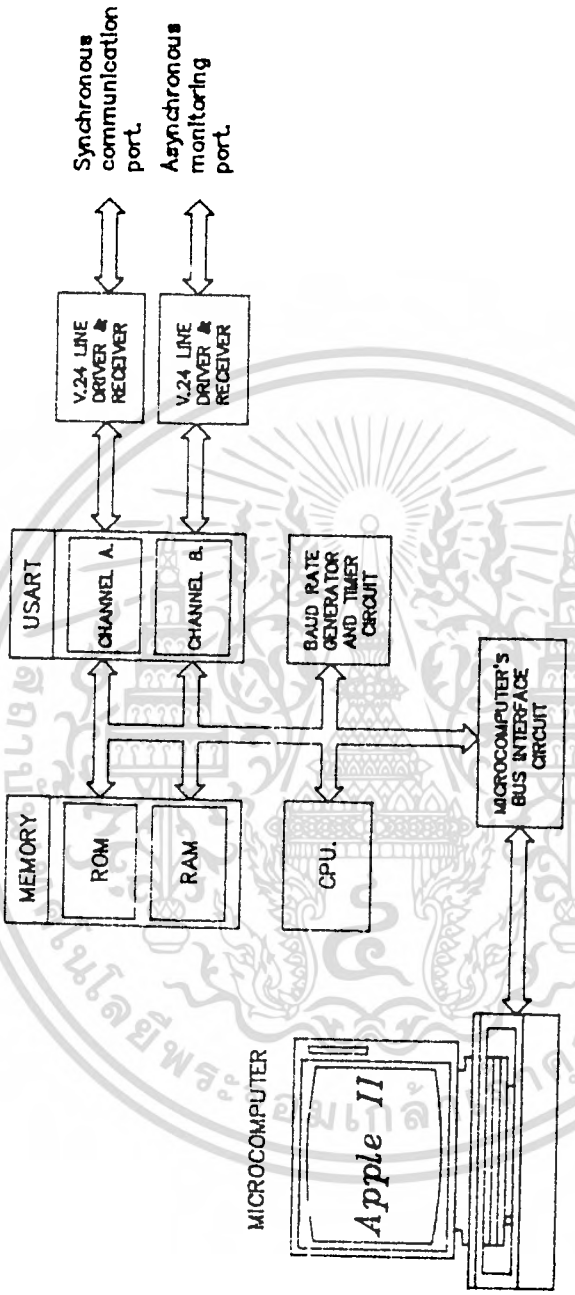
อีกประการหนึ่ง คือ สามารถติดต่อกับหน่วยความจำของไมโครคอมพิวเตอร์ได้โดยตรง ในขณะที่ไมโครคอมพิวเตอร์กำลังทำงาน ในลักษณะขนานกันได้ เพื่อที่จะสังเกตุการทำงาน และตรวจสอบข้อมูล ในขณะที่ส่วนคณเวอร์เตอร์ กำลังติดต่อสื่อสารกับคอมพิวเตอร์ NEC ACOS-300 โดยได้แบ่งให้มีการจัดเรียงหน่วยความจำ ดังแสดงในรูป 4.1.2

4.1.3 อุปกรณ์แปลงข้อมูลอนุกรม วงจรสร้างความถี่บอดเรท และวงจรอินเทอร์เฟสแบบ V.24

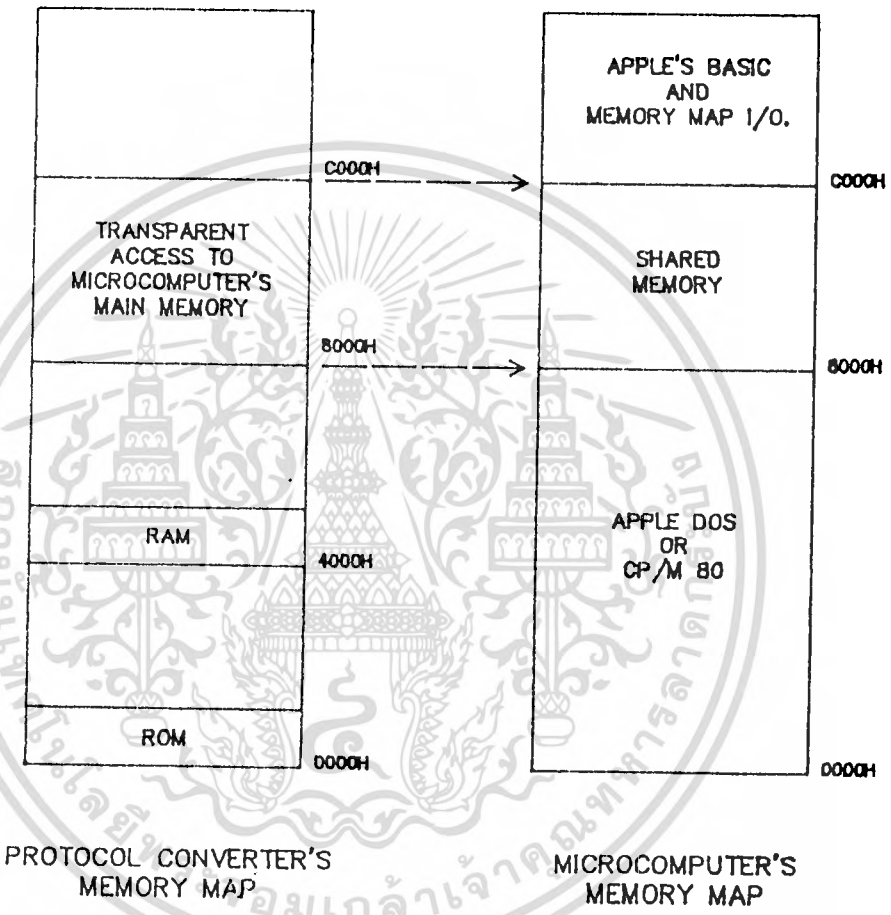
สามส่วนประกอบกันทำหน้าที่ในการแปลงข้อมูลไปกลับ ระหว่างข้อมูลแบบขนานและอนุกรม โดยในส่วน USART ที่ใช้เป็นแบบ Z80 SIO ที่มีวงจรแปลงข้อมูล 2 ช่อง ได้เลือกใช้ช่อง A เป็นช่องสำหรับการสื่อสารแบบซิงโครนัส สำหรับติดต่อกับเครื่อง NEC ACOS-300 ส่วนช่อง B เป็นช่องสำหรับการสื่อสารแบบอะซิงโครนัส เพื่อต่อกับเทอร์มินัล สำหรับเป้าตรวจดูการติดต่อ

ในส่วนของอุปกรณ์แปลงข้อมูลอนุกรม Z80 SIO จะมีการติดต่ออินพุท และเอาท์พุท ในระดับสัญญาณแบบ TTL แต่การอินเทอร์เฟสเข้ากับเทอร์มินัล หรือโมเดม ซึ่งมีการอินเทอร์เฟสแบบ มาตรา-ฐาน V.24 จึงต้องมีวงจรสำหรับอินเทอร์เฟสแบบ V.24 ช่วยปรับการอินเทอร์เฟสให้เหมาะสม

ในการสื่อสารแบบซิงโครนัส โดยทั่วไปแล้วสัญญาณคล็อก ที่ใช้ในการรับและส่งข้อมูลจะได้จากโมเดม แต่สำหรับการสื่อสารแบบอะซิงโครนัส จะต้องใช้สัญญาณคล็อก ในการรับส่งข้อมูลภายในส่วนวงจรแปลงข้อมูลอนุกรมเอง จึงได้ออกแบบให้มีวงจรกำเนิดสัญญาณบอดเรท ไว้สำหรับการสื่อสารแบบอะซิงโครนัส ด้วย



รูปที่ 4.1 แสดงบล็อกไดอะแกรมของโปรโตคอล คอมเวอร์เคอร์



รูปที่ 4.1.2 แสดงการจัดหน่วยความจำของคอนเวอร์เตอร์ และไมโครคอมพิวเตอร์

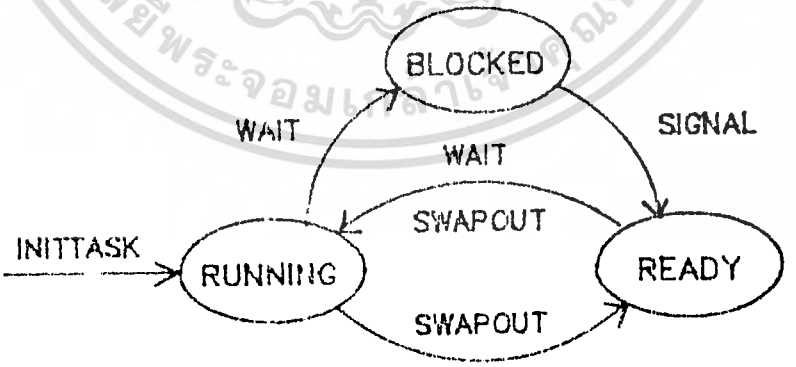
4.2 การออกแบบส่วนซอฟท์แวร์ของโปรโตคอล คอนเวอร์เตอร์

โปรแกรมส่วนควบคุมทั้งหมด ได้พัฒนาขึ้นด้วยภาษาแอสแซมบลี้ สำหรับ Z80 CPU (Z80 Assembly language) บนเครื่องไมโครคอมพิวเตอร์ AppleII+ ภายใต้ระบบควบคุมการทำงานแบบ CP/M 80 โดยได้ออกแบบให้มีลักษณะเป็นแบบโมดูล่า (Modular) ดังได้กล่าวในตอนต้นเพื่อให้สามารถพัฒนาได้ง่าย และสดวกต่อการแก้ไขปรับปรุงแต่ละโมดูลเข้าด้วยกัน

4.2.1 ระบบแกนจัดการแบบ มัลติทาส์ค (Multitasking Kernel)

เนื่องจากโปรโตคอล คอนเวอร์เตอร์ เป็นอุปกรณ์ที่มีการทำงานขึ้นตรงต่อเวลาจริง (Real time) ดังนั้นโปรแกรมควบคุมการทำงาน จะต้องมีประสิทธิภาพ และสามารถตอบสนองการสื่อสารข้อมูลได้รวดเร็วเพียงพอ

ระบบแกนการจัดการแบบ มัลติทาส์ค จึงเป็นระบบที่เหมาะสมที่สุด ซึ่งจะสามารถจัดการให้โมดูลต่าง ๆ ทำงานพร้อม ๆ กัน หรือทำงานในจังหวะที่พอเหมาะได้ โดยมีขั้นตอนในการทำงาน ดังแสดงในรูปที่ 4.2.1



รูปที่ 4.2.1 การทำงานของระบบแกนจัดแบบมัลติทาสค์

ไดอะแกรมในรูป 4.2.1 แสดงสถานะที่แต่ละทาสค์จะสามารถเป็นได้ (รูปวงรี) และคำสั่งควบคุม ในการทำให้มีการเปลี่ยนแปลงจากสถานะหนึ่งไปยังอีกสถานะหนึ่ง (เส้นลูกศร)

ไดอะแกรมนี้เป็นเฉพาะของเพียงทาสค์เดียว ซึ่งสำหรับทุก ๆ ทาสค์ ก็จะมีขั้นตอนเช่นเดียวกัน แต่จะมีเพียงทาสค์เดียวเท่านั้น ที่จะอยู่ในสถานะที่กำลังทำงาน (Running) ได้ และสามารถทำคำสั่งควบคุม เพื่อเปลี่ยนสถานะตัวเองเข้าสู่สถานะอื่นได้

```
Program Task1;
begin
   repeat
      begin
         if SyncRxFlag = Ready then
            if SyncRx = NoError then          { check receive status       }
            begin                             { receive ok.                }
               SyncTx(OK);                    {  reply to sender           }
               Signal(SyncRxFrameAvail);      {  and signal to others task }
            end
          else
            begin                             { receive error              }
               SyncTx(ERROR);                 {  reply to sender           }
               Swapout;                       {  and let others task run   }
            end;
          else Swapout;                       { receive frame not ready    }
      end;                                    {  let others task run       }
   until True=False;                          { repeat forever             }
end.

Program Task2;
begin
   repeat
      Wait(SyncRxFrameAvail);                 { wait until receive frame   }
      Transfer(ReceiveBfuuer,OutputBuffer);   {  ready then transfer to    }
      SetFlag(OutputReady);                   {  Output Buffer,  set flag  }
      Swapout;                                {  and let others task run   }
   until True=False;                          { repeat forever             }
end.
```

ตัวอย่าง โครงร่างโปรแกรมการรับส่งข้อมูลภายใต้ระบบมัลติทาสค์

จากโปรแกรมตัวอย่างจะเห็นว่า เราสามารถเขียนโปรแกรมในการรับและส่งข้อมูล ในลักษณะที่เป็นโมดูลแยกจากกันได้ และโมดูลทั้งสองจะทำงานร่วมกันและมีการติดต่อถึงกัน โดยคำสั่งในการจัดการแบบ มัลติทาส์ค ซึ่งจะอธิบายการทำงานได้ดังนี้

ในทาส์คที่ 1 โปรแกรมจะทำการตรวจสอบ SyncRxFlag วนซ้ำกันไปจนกว่าจะ Ready แต่ในขณะที่วนอยู่ คำสั่ง Swapout จะเปลี่ยนสถานะของทาส์คที่ 1 ไปสู่ Ready เพื่อให้ทาส์คที่ 2 หรือทาส์คอื่น ๆ ได้ทำงาน Running ต่อ

ในทาส์คที่ 2 โปรแกรมจะถูกสั่งให้หยุดรอด้วยคำสั่ง Wait และเข้าสู่สถานะ Blocked และรอจนกว่าจะได้รับการกระตุ้นด้วยคำสั่ง Signal จากทาส์คที่ 1 ในขณะที่ทาส์คที่ 2 กำลังหยุดรออยู่นี้ ทาส์คที่ 1 หรือทาส์คอื่นๆ ก็มีโอกาสเข้าสู่สถานะ Running ได้เช่นกัน

และเมื่อเหตุการณ์ใน ทาส์คที่ 1 เป็นจริง คือ SyncRxFlag = Ready และไม่เกิดการผิดพลาด (Error) SyncRxFrameAvail ก็จะถูกกระตุ้น ทำให้ทาส์คที่ 2 เปลี่ยนจากสถานะ Blocked เข้าสู่ Ready ในขณะเดียวกัน ทาส์คที่ 1 ก็จะเปลี่ยนจาก Running เข้าสู่ Ready เช่นกัน

ดังนั้น ทาส์คที่ 2 ซึ่งเข้าสู่ Ready ก่อน ก็จะได้เข้าสู่ Runngin และทำงานในโปรแกรมส่วนที่เหลือ จนกระทั่งถึงคำสั่ง Swapout ก็จะสลับให้ทาส์คอื่น ๆ ทำงานอีกต่อไป

ในการออกแบบและแยกโมดูล เพื่อการทำงานในระบบมัลติทาส์ค ควรจะเลือกเฉพาะโมดูลที่มีความซับซ้อน ต้องการการตอบสนองที่รวดเร็ว เนื่องจากการมีจำนวนทาส์คมากจะทำให้สูญเสียเวลาของระบบไปกับส่วนของการจัดการมากเกินไป ซึ่งจะเลือกให้โมดูลที่ทำหน้าที่ในการรับข้อมูล และส่วนควบคุมการรับส่ง เป็นส่วนสำคัญที่จะต้องทำงานแบบมัลติทาส์ค เพียง 3 ทาส์ค ดังจะได้กล่าวต่อไป

4.2.2 โมดูลสำหรับรับข้อมูลจากไมโครคอมพิวเตอร์ (ทำงานในทาส์คที่ 1)

เนื่องจากคอนเวอร์เตอร์ ที่ออกแบบสามารถติดต่อกับหน่วยความจำ ของไมโครคอนพิวเตอร์ได้โดยตรง และมีบัฟเฟอร์ในการส่งผ่านข้อมูลจากไมโครคอมพิวเตอร์อยู่ในบริเวณหน่วยความจำดังกล่าว เมื่อไมโครคอมพิวเตอร์ต้องการส่งข้อมูล ก็จะใส่ข้อมูลลงในบัฟเฟอร์และจัดแฟลกเพื่อบอกให้รู้ว่ามีข้อมูลรอที่จะส่ง โมดูลส่วนนี้จะตรวจสอบแฟลกและส่งสัญญาณ ให้โมดูลควบคุมการรับ-ส่งข้อมูลแบบ ซิงโครนัส

รู้ดังนั้นโมดูลส่วนนี้จึงยังไม่มีภารกิจที่จะต้องทำมากนัก แต่ได้เตรียมไว้สำหรับการปรับปรุงเพื่อนำไปเชื่อมกับไมโครคอมพิวเตอร์เครื่องอื่น ๆ ดังจะได้กล่าวในบทต่อไป

4.2.3 **โมดูลสำหรับรับข้อมูลแบบซิงโครนัส** (ทำงานในทาส์คที่ 2)

ในการรับข้อมูลแบบซิงโครนัส ข้อมูลจะถูกส่งเข้ามายังคอนเวอร์เตอร์ โดยการอินเทอรัพท์ ในส่วนโปรแกรมสนับสนุนการอินเทอรัพท์ จะทำหน้าที่จัดรวบรวมข้อมูลในเฟรม แยกเฟรมควบคุมและเฟรมข้อมูลออกจากกัน โดยโมดูลส่วนนี้จะทำหน้าที่เป็นตัวเชื่อมในการรับข้อมูล และส่งเฟรมต่าง ๆ ให้กับโมดูลควบคุมการรับ-ส่งอีกทีหนึ่ง ด้วยการส่งสัญญาณ ไปยังโมดูลดังกล่าว

4.2.4 **โมดูลควบคุมการรับ-ส่งข้อมูลแบบซิงโครนัส** (ทำงานในทาส์คที่ 3)

โมดูลส่วนนี้ จะรับหน้าที่ในการจัดการเกี่ยวกับการรับและส่งข้อมูล ในแบบ ซิงโครนัส ตามกฎเกณฑ์ของโปรโตคอล ในการสื่อสารข้อมูลแบบ Level 2B ของเครื่อง NEC ACOS-300 รวมทั้งจัดขั้นตอนการทำงานงาน ของสัญญาณควบคุมโมเดม

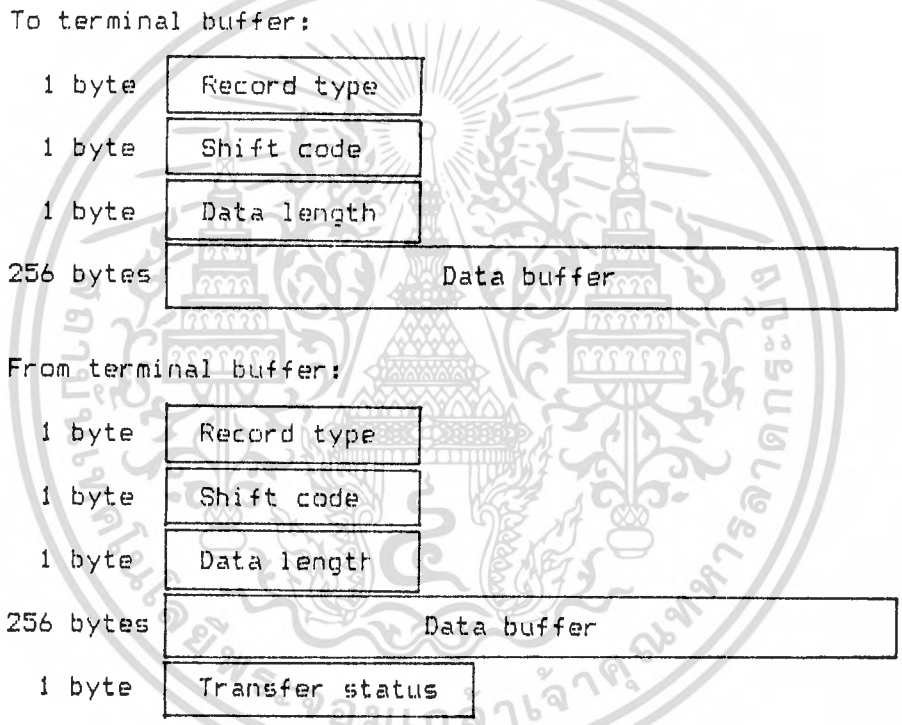
ในการรับข้อมูล เฟรมควบคุมและข้อมูล จะถูกส่งมาจากโมดูลในทาส์คที่ 2 เฟรมควบคุมการสื่อสาร จะได้รับการประมวลผล และตอบรับ ตามกฎเกณฑ์ของโปรโตคอล ส่วนเฟรมที่เป็นข้อมูล จะถูกแยกและฟอร์แมท เพื่อส่งให้กับโมดูลที่จัดการในการส่งผ่านสู่ไมโครคอมพิวเตอร์ โดยผ่านบัฟเฟอร์ในการติดต่อ

ส่วนในการส่งข้อมูล จะทำการส่งได้เมื่อมีสอบถามหรือโพลลิ่ง จากเครื่องเมนเฟรมซึ่งจะรู้ได้จากเฟรมควบคุมที่รับได้ และถ้าในขณะนั้นมีการส่งสัญญาณมาจากโมดูลในทาส์คที่ 1 แสดงว่ามีข้อมูลพร้อมที่จะส่ง ก็จะส่งสัญญาณควบคุมโมเดมเพื่อเตรียมการส่ง และข้อมูลจากบัฟเฟอร์ จะถูกนำมาฟอร์แมทใหม่ ส่งไปในแบบซิงโครนัส และตรวจสอบการตอบรับจากเครื่องเมนเฟรม ตามกฎเกณฑ์ของโปรโตคอล

4.3 ลักษณะฟอร์แมทของบัฟเฟอร์ในการรับส่งข้อมูลจากไมโครคอมพิวเตอร์

เนื่องจากโปรโตคอล คอนเวอร์เตอร์ ที่ออกแบบมีการกำหนดโปรโตคอลที่แน่นอนเพียงทางเดียว คือด้านที่มีการสื่อสารแบบ ซิงโครนัส ซึ่งมีการติดต่อสื่อสารกับเครื่องเมนเฟรม ส่วนทางด้านที่มีการติดต่อกับไมโครคอมพิวเตอร์ ไม่ได้มีการกำหนดโปรโตคอลไว้ เพราะต้องการให้สามารถเชื่อมกับ

โปรแกรมที่ใช้งาน (Application Program) ได้ง่าย ตัวคอนเวอร์เตอร์ จึงทำหน้าที่ในการแปลงโปรโตคอลจาก แบบ Level 2B มาอยู่ในรูปของการจัดเรียงข้อมูลอยู่ในบัฟเฟอร์ และในการส่งก็จะแปลงข้อมูลจากบัฟเฟอร์ไปเป็นแบบ Level 2B ซึ่งเป็นลักษณะการทำงานแบบ เวอร์ช่วล โปรโตคอล (Virtual protocol) บัฟเฟอร์ดังกล่าว มีโครงสร้างดังแสดงในรูป 4.3.



รูปที่ 4.3 แสดงบัฟเฟอร์ข้อมูลในการติดต่อกับไมโครคอมพิวเตอร์

บทที่ 5

การดัดแปลงโปรโตคอล คอนเวอร์เตอร์และการประยุกต์ใช้งาน

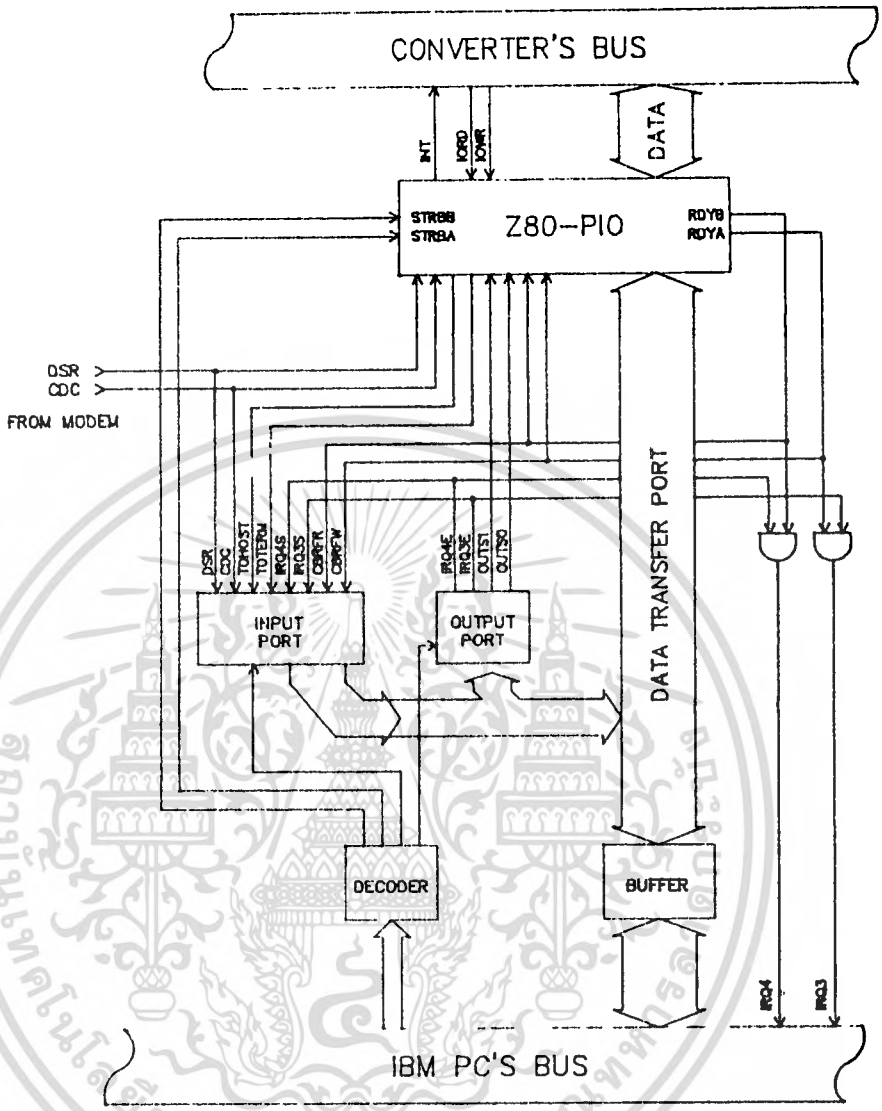
ในการทำการวิจัยทางด้านการประมวลสัญญาณภาพ ผู้ทำการวิจัยส่วนใหญ่จะเลือกใช้ไมโครคอมพิวเตอร์ แบบ IBM PC หรือเครื่องอื่น ๆ ที่มีคุณสมบัติใกล้เคียงกันเนื่องจากเป็นไมโครคอมพิวเตอร์ขนาด 16 บิท มีความสามารถในการคำนวณสูงพอสมควร และสามารถต่อหรือออกแบบอุปกรณ์สนับสนุน (Peripheral) เช่น อุปกรณ์แปลงสัญญาณภาพ หรืออุปกรณ์แสดงภาพได้ง่าย

ดังนั้นงานวิจัยในส่วนออกแบบสร้างโปรโตคอล คอนเวอร์เตอร์ เพื่อสนับสนุนงานวิจัยทางด้านการประมวลสัญญาณภาพบนเครื่อง ไมโครคอมพิวเตอร์ดังกล่าว ขึ้นมาอีกชุดหนึ่ง โดยดัดแปลงในส่วนของฮาร์ดแวร์บางส่วน และออกแบบโมดูลซอฟท์แวร์ในการรับส่งข้อมูลระหว่าง คอนเวอร์เตอร์ และไมโครคอมพิวเตอร์เพิ่มเติม

5.1 ส่วนฮาร์ดแวร์สำหรับการอินเทอร์เฟสกับ IBM PC

วงจรส่วนนี้ประกอบด้วยพอร์ทรับส่งข้อมูลแบบขนาน 8 บิท และพอร์ทสำหรับสัญญาณควบคุมในการรับส่งข้อมูลโดยทางด้านโปรโตคอล คอนเวอร์เตอร์ใช้ Z80 PIO (Parallel Input/Output interface) เป็นตัวเชื่อมต่อ โดยให้พอร์ท A ทำงานในลักษณะเป็นพอร์ทรับส่งข้อมูล 2 ทิศทาง แบบมีสัญญาณควบคุมการรับส่ง (Hand Shaking) และพอร์ท B แบ่งทำงานเป็นอินพุทพอร์ท และเอาท์พุทพอร์ท สำหรับการตรวจสอบและแสดงสถานะ (Status) ของคอนเวอร์เตอร์ ส่วนทางด้านไมโครคอมพิวเตอร์ จะประกอบด้วยพอร์ทแบบขนาน 2 ทิศทาง (โดยวงจรอินเทอร์เฟสจะออกแบบให้ไมโครคอมพิวเตอร์มองเห็น Z80 PIO ดังกล่าวเป็นพอร์ทของ ไมโครคอมพิวเตอร์ด้วย) เอาท์พุทพอร์ทสำหรับควบคุมการอินเทอร์รัพท์ และแสดงสถานะของโปรโตคอล คอนเวอร์เตอร์ และอินพุทพอร์ทสำหรับตรวจสอบสถานะของโมเดม

ในการส่งผ่านข้อมูลระหว่างไมโครคอมพิวเตอร์กับ คอนเวอร์เตอร์ จะขึ้นอยู่กับการควบคุมจากไมโครคอมพิวเตอร์ โดยการอ่านหรือเขียนข้อมูลไปยังพอร์ทดังกล่าว ข้อมูลจะถูกส่งเข้าหรือนำออกจากคอนเวอร์เตอร์ โดยการอินเทอรัพท์ทุก ๆ ครั้งที่มีการอ่านหรือเขียนโดยไมโครคอมพิวเตอร์



รูปที่ 5.1 บล็อคไดอะแกรมของส่วนอินเทอร์เฟสกับ IBM PC

5.2 โมดูลรับส่งข้อมูลสำหรับการอินเทอร์เฟสกับ IBM PC

โมดูลนี้มีการแก้ไขเพิ่มเติมจากโมดูลในการรับข้อมูลจากไมโครคอมพิวเตอร์ ประกอบด้วยหลายส่วน ที่ทำหน้าที่ต่าง ๆ กันเพื่อสนับสนุนการทำงานของโปรโตคอล คอนเวอร์เตอร์ เพื่อให้มีการติดต่อกับไมโครคอมพิวเตอร์ ได้ถูกต้อง เช่น การตรวจสอบและแสดงสถานะของส่วน คอนเวอร์เตอร์ การส่งผ่านข้อมูลระหว่าง คอนเวอร์เตอร์ และไมโครคอมพิวเตอร์ รวมทั้งโปรแกรมช่วยเหลือในการตรวจสอบการทำงานของ คอนเวอร์เตอร์เองด้วย

5.2.1 การตรวจสอบ และแสดงสถานะของ คอนเวอร์เตอร์

เนื่องจากตัว คอนเวอร์เตอร์ ได้ถูกออกแบบให้ทำงานร่วมกับไมโครคอมพิวเตอร์ IBM PC ในลักษณะที่เป็น สเลฟโปรเซสเซอร์ (Slave Processer) ที่จัดการในด้านการสื่อสารแบบ ซิงโครนัส ดังนั้นในการที่ไมโครคอมพิวเตอร์จะให้ คอนเวอร์เตอร์ทำงานในแต่ละอย่าง ก็จำเป็นจะรู้ถึงสถานะของคอนเวอร์เตอร์ก่อนทุกครั้ง โดยการแสดงสถานะของคอนเวอร์เตอร์ แบ่งได้เป็น 2 ลักษณะ คือ

1) สถานะทางฮาร์ดแวร์ เป็นส่วนที่ไมโครคอมพิวเตอร์สามารถตรวจสอบได้โดยตรง ประกอบด้วยอินพุทรีจิสเตอร์ในการแสดงสถานะ 4 บิท 2 ชุด เอาท์พุทรีจิสเตอร์ในการควบคุม ขนาด 4 บิท 1 ชุด และพอร์ทสำหรับการรีเซ็ท คอนเวอร์เตอร์

ก) อินพุทรีจิสเตอร์ CBstatPA ตรงกับแอดเดรสของอุปกรณ์อินพุท ในตำแหน่ง 3ACH ของเครื่อง IBM PC

b8 – b5	b4	b3	b2	b1
X X X X	DSR	CDC	TOHOST	TOTERM

DSR แสดงสถานะสัญญาณ DSR จากโมเดม

CDC แสดงสถานะสัญญาณ CDC จากโมเดม

TOHOST แสดงสถานะของบัฟเฟอร์ ที่ใช้ในการรับข่าวสารจากไมโครคอมพิวเตอร์

TOTERM แสดงสถานะของบัฟเฟอร์ ที่ใช้ในการส่งข่าวสารมายังไมโครคอมพิวเตอร์

ข) อินพุทรีจิสเตอร์ CBstatPB ตรงกับแอดเดรสของอุปกรณ์อินพุท ในตำแหน่ง 3ADH ของเครื่อง IBM PC

b8 – b5	b4	b3	b2	b1
X X X X	IRQ4S	IRQ3S	CBRFR	CBRFW

IRQ4S, IRQ3S แสดงสถานะการใช้ IRQ3, IRQ4 ของเครื่อง IBM PC

CBRFR Converter Board is Ready For Read operation
แสดงสถานะเมื่อ คอนเวอร์เตอร์พร้อมที่จะให้เครื่อง IBM PC ทำการอ่านข้อมูลได้

CBRFW Converter Board is Ready For Write operation
แสดงสถานะเมื่อ คอนเวอร์เตอร์พร้อมที่จะให้เครื่อง IBM PC ทำการเขียนข้อมูลได้

ค) เอาท์พุทรีจิสเตอร์ CBstatPC ตรงกับแอดเดรสของอุปกรณ์เอาท์พุท ในตำแหน่ง 3ACH ของเครื่อง IBM PC

b8 – b5	b4	b3	b2	b1
X X X X	IRQ4E	IRQ3E	OUTS1	OUTS0

IRQ4E,IRQ3E เป็นสัญญาณควบคุม การใช้ IRQ3-IRQ4 ของเครื่อง IBM PC

OUTS1,OUTS0 ไม่ได้ใช้ (เผื่อไว้สำหรับการแสดงสถานะ จากเครื่อง IBM PC ไปยังคอนเวอร์เตอร์

2) สถานะการทำงานของโปรแกรมควบคุมของคอนเวอร์เตอร์ นอกเหนือจากการที่ไมโครคอมพิวเตอร์จะตรวจสอบสถานะทางฮาร์ดแวร์แล้ว ในขณะที่โปรแกรมควบคุมการแปลงโปรโตคอลกำลังทำงาน ไมโครคอมพิวเตอร์จะสามารถใช้คำสั่งควบคุม RdStatCmd ในการตรวจสอบสถานะของบัฟเฟอร์ในการรับ และบัฟเฟอร์ในการส่งแบบ ซิงโครนัส หรือคำสั่งควบคุมที่ได้รับจากเครื่องเมนเฟรมได้ ซึ่งจะได้กล่าวในรายละเอียดของ RdStatCmd ต่อไป

5.2.2 การส่งผ่านข้อมูลระหว่างไมโครคอมพิวเตอร์กับคอนเวอร์เตอร์

ในโมดูลส่วนนี้จะรับรู้คำสั่งควบคุมในการส่งผ่านข้อมูลจากไมโครคอมพิวเตอร์ จากไมโครคอมพิวเตอร์ 2 คำสั่ง คือ SetRecCmd ใช้ในการส่งข้อมูลจากไมโครคอมพิวเตอร์มาสู่คอนเวอร์เตอร์ และคำสั่ง GetRecCmd ใช้ในการส่งข้อมูลจากคอนเวอร์เตอร์ไปยังคอมพิวเตอร์โดยมีฟอร์แมทในการส่งผ่าน

ดังแสดงในรูป 5.2.2

Record type 1 byte	Shift code 1 byte	Data length 1 byte	Data

Record Type เป็นตัวบอกประเภทของบล็อคข้อมูลว่าเป็น แบบ ETB หรือ ETX

Shift Code เป็นอักษรควบคุม SI หรือ SO ตามตารางรหัสมาตราฐาน JIS-7

Data length เป็นตัวบอกจำนวนไบท์ของข้อมูล โดยมีกฎเกณฑ์ตามตาราง

Record type	Data length	# of data byte
ETX	0	256 bytes
	1 - 255	1 - 255 bytes
ETB	0	0 bytes
	1 - 255	1 - 255 bytes

รูปที่ 5.2.2 ฟอร์แมทในการส่งผ่านข้อมูล

5.2.3 ส่วนสนับสนุนการตรวจสอบการทำงานของคอนเวอร์เตอร์

อย่างไรก็ตามเนื่องจากคอนเวอร์เตอร์ ได้ออกแบบเป็นการวิจัย ที่ยังต้องมีขั้นตอนในการพัฒนาเพื่อการใช้งานในด้านต่าง ๆ ต่อไปอีก จึงได้ออกแบบให้มีคำสั่ง เพื่อช่วยในการตรวจสอบการทำงาน เช่นการต่อเทอร์มินัล เพื่อดูการไหลของข้อมูลในการติดต่อกับเครื่องเมนเฟรม และการดึงเอาข้อมูลจากหน่วยความจำของคอนเวอร์เตอร์ มาตรวจสอบ โดยใช้คำสั่ง SetMONCmd และ CBdumpCmd ดังแสดงในหัวข้อ 5.3.2

5.3 การออกแบบโปรแกรมบนเครื่อง IBM PC เพื่อติดต่อกับคอนเวอร์เตอร์

ดังที่ได้กล่าวมาแล้วในหัวข้อ 5.2 ถึงการทำงานของโมดูลในการติดต่อกับเครื่อง IBM PC ซึ่งเป็นการติดต่อที่อยู่ทางด้านคอนเวอร์เตอร์ ส่วนทางด้านเครื่อง IBM PC เองก็จำเป็นจะต้องมีโปรแกรมเพื่อการติดต่อใช้งานกับคอนเวอร์เตอร์

5.3.1 การอ่านและเขียนข้อมูลไปยังคอนเวอร์เตอร์

ในการอ่านข้อมูล จากคอนเวอร์เตอร์ดังได้กล่าวมาแล้วในหัวข้อม 5.2 ซึ่งจะประกอบด้วยส่วนแสดงสถานะที่สามารถตรวจสอบได้โดยตรง และส่วนที่จะต้องมีการอ่านหรือเขียนผ่าน อินเทอร์เฟสพอร์ทจะต้องมีการตรวจสอบสถานะ CBRFR เพื่อให้แน่ใจว่าคอนเวอร์เตอร์ อยู่ในสถานะพร้อมที่จะให้อ่านได้ทุก ๆ ครั้งก่อนที่จะมีการอ่าน และในทำนองเดียวกัน เมื่อต้องการเขียนข้อมูลก็จะต้องตรวจสอบสถานะ CBRFW ก่อนทุก ๆ ครั้งเช่นกัน

```
(* Write one byte to Converter *)

procedure  WriteCB(data:byte);
begin
   Repeat until (Port[CBstatPB] and CBRFW) = CBRFW;
   Port[CBdataPx] := data;
end;


(* Read one byte from Converter *)

procedure  ReadCB(var data:byte);
begin
   repeat until (Port[CBstatPB] and CBRFR) = CBRFR;
   data := Port[CBdataPx];
end;
```

ตัวอย่าง การอ่านและเขียนข้อมูลสู่คอนเวอร์เตอร์

5.3.2 ชุดคำสั่งในการควบคุมการทำงานของคอนเวอร์เตอร์

ดังที่ได้กล่าวมาแล้วถึงโมดูลในการอินเทอร์เฟส ซึ่งจะทำหน้าที่ในการติดต่อกับเครื่อง IBM PC และเนื่องจากการติดต่อมีหลายลักษณะที่มีหน้าที่ต่าง ๆ กัน ดังนั้นในแต่ละหน้าที่จึงต้องมีคำสั่งควบคุม เป็นตัวบอกถึงขั้นตอนต่อไปของงานที่จะทำ โดยได้ออกแบบให้มีคำสั่งพื้นฐาน 6 คำสั่ง และมีขั้นตอนในการใช้คำสั่งดังนี้ คือ

1) RdStatCmd (Read status command) ใช้สำหรับอ่านสถานะการทำงานของคอนเวอร์เตอร์

RdStatCmd

TRANSFER BYTE	OPERATION
0 0 0 0 0 0 0 1	Write
Status byte	Read

Status byte:

b8	b7	b6	b5	b4	b3	b2	b1
SSBFST	SRBFST	TRBFST	TWBFST	HWABT	HBRK	HINT	NA

SRBFST แสดงสถานะของบัฟเฟอร์ ในการรับข้อมูลแบบ ซิงโครนัส

SSBFST แสดงสถานะของบัฟเฟอร์ ในการส่งข้อมูลแบบ ซิงโครนัส

TRBFST แสดงสถานะของบัฟเฟอร์ ที่ใช้รับข่าวสารจากไมโครคอมพิวเตอร์

TWBFST แสดงสถานะของบัฟเฟอร์ ที่ใช้ส่งข่าวสารให้กับไมโครคอมพิวเตอร์

HWABT แสดงสถานะของคอนเวอร์เตอร์ เมื่อได้รับสัญญาณ DLE ?

HINT แสดงสถานะของคอนเวอร์เตอร์ เมื่อได้รับสัญญาณ DLE <

HBRK แสดงสถานะของคอนเวอร์เตอร์ เมื่อได้รับสัญญาณ DLE :

2) SetADRCmd (Set Address command) ใช้ในการกำหนดค่าแอดเดรส SA และ UA สำหรับการเปรียบเทียบในการโพลลิ่งให้กับคอนเวอร์เตอร์

SetADRCmd

TRANSFER BYTE	OPERATION
0 0 0 0 0 0 1 0	Write
S A	Write
U A	Write

SA หมายเลขแอดเดรสประจำเทอร์มินัล ในการโพลลิ่งและซีเลคติ้ง

UA หมายเลขแอดเดรสของอุปกรณ์อินพุท

3) SetRECCmd (Set record to be sent command) ใช้ในการส่งข้อมูลที่ต้องการส่งไปยังเครื่องเมนเฟรม ให้กับคอนเวอร์เตอร์

SetRECCmd

TRANSFER BYTE	OPERATION
0 0 0 0 0 0 1 1	Write
Record type	Write
Shift code	Write
Data length	Write
≈ Data ≈	Write ≈

Record Type ประเภทของ Record ที่ต้องการส่ง (ETB/ETX)

Shift Code รหัส Shift ในการเลือกชุดอักษรตามตารางมาตราฐาน JIS-7 (SI/SO)

Data length จำนวนไบท์ของข้อมูลที่ต้องการส่ง

Data ข้อมูลที่ต้องการส่งมีจำนวนไบท์เท่ากับที่กำหนดด้วย Data length

4) GetRECCmd (Get received record command) ใช้ในการอ่านข้อมูลที่รับได้จากเครื่องเมนเฟรม ออกจากคอนเวอร์เตอร์ สู่เครื่อง IBM PC

GetRECCmd

TRANSFER BYTE	OPERATION
0 0 0 0 0 1 0 0	Write
Record type	Read
Shift code	Read
Data length	Read
≈ Data ≈	Read ≈

Record Type ประเภทของ Record ที่รับ (ETB/ETX)

Shift code รหัส Shift ในการเลือกชุดอักษร เพื่อใช้กับข้อมูลที่รับได้

Data length จำนวนไบท์ของข้อมูลที่รับได้

Data ข้อมูลที่รับได้ มีจำนวนไบท์เท่ากับที่กำหนดด้วย Data length

5) SetMONCmd (Set monitor mode command) ใช้ในการเปิดหรือปิดแฟลกสำหรับการมอนิเตอร์ หรือดูการรับส่งข้อมูล ผ่านทางอะซิงโครนัสพอร์ท

SetMONCmd

TRANSFER BYTE	OPERATION
0 0 0 0 1 1 1 0	Write
Control flag	Write

Control flag:

b8	b7	b6	b5	b4	b3	b2	b1
SRXMON	STXMON	NA	NA	NA	NA	NA	NA

SRXMON แฟลกควบคุมการมอนิเตอร์ ขณะมีการรับข้อมูลแบบ ซิงโครนัส

STXMON แฟลกควบคุมการมอนิเตอร์ ขณะมีการส่งข้อมูลแบบ ซิงโครนัส

6) CBDumpCmd (Dump converter's memory command) ใช้ในการถ่ายข้อมูลจากหน่วยความจำบางส่วนของคอนเวอร์เตอร์มายังเครื่อง IBM PC เพื่อใช้ในการตรวจสอบการทำงานของคอนเวอร์เตอร์

CBDumpCmd

TRANSFER BYTE	OPERATION
0 0 0 0 1 1 1 1	Write
Address Low	Write
Address High	Write
Data length	Write
≈ Data ≈	Read ≈

Address-Low	ไบท์สูงของค่าแอดเดรส เริ่มต้นของหน่วยความจำที่ต้องการ Dump
Address-High	ไบท์สูงของค่าแอดเดรส เริ่มต้นของหน่วยความจำที่ต้องการ Dump
Data length	จำนวนไบท์ที่ต้องการ Dump 1-256 ไบท์
Data	ข้อมูลจากการ Dump มีจำนวนไบท์เท่ากับที่กำหนดด้วย Data length

5.4 การจัดฟอร์แมทข้อมูล 8 บิท เพื่อส่งผ่านโดยโปรโตคอลแบบ Level 2B

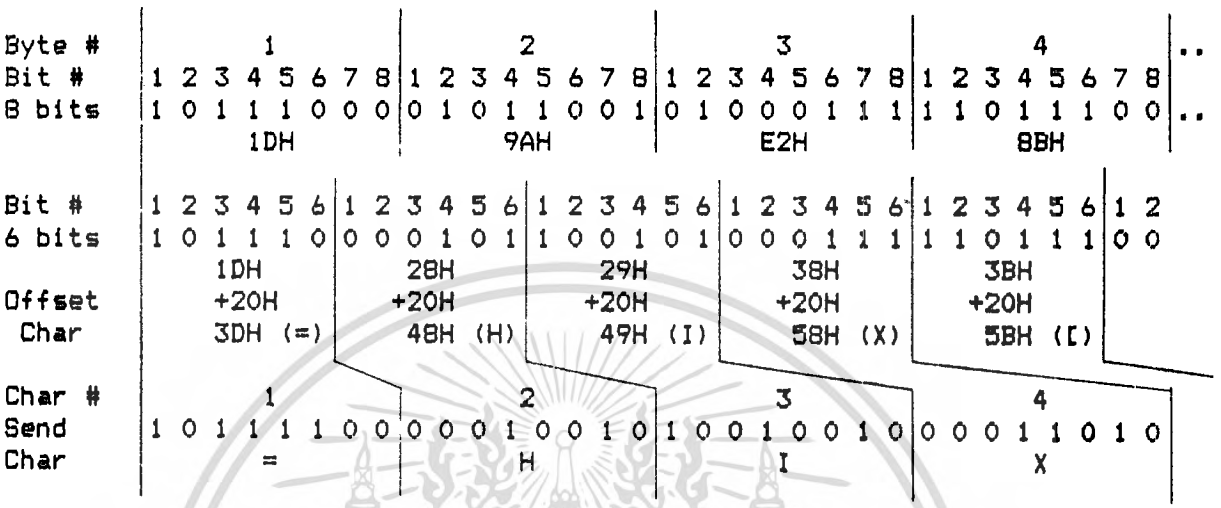
ในการประมวลสัญญาณทางดิจิตอล ไม่ว่าจะเป็นข้อมูลของสัญญาณที่ถูกแปลงมาจากสัญญาณอนาลอก ข้อมูลสัญญาณภาพ หรือข้อมูลภาพถ่ายดาวเทียมจากดาวเทียมสำรวจทรัพยากรธรรมชาติ จะอยู่ในรูปของข้อมูลที่แทนระดับของสัญญาณหรือแทนความเข้มของภาพ อาจจะถูกแบ่งออกเป็น 256 ระดับ และแทนด้วยข้อมูลขนาด 8 บิท หรืออาจจะมีจำนวนระดับมากกว่า และมีจำนวนบิทมากกว่า

สำหรับข้อมูลขนาด 8 บิท จะมีปัญหาในการส่งผ่านโดยโปรโตคอล แบบ Level 2B ดังที่กล่าวมาแล้วในบทที่ 2 เนื่องจากโปรโตคอลแบบนี้ ได้ถูกออกแบบไว้สำหรับรับส่งข้อมูลที่เป็น Text อันประกอบด้วยอักษรที่สามารถแสดงได้ (Printable character) และอักษรควบคุมบางตัวเท่านั้น ดังนั้น ข้อมูลขนาด 8 บิท จะต้องมีการแปลงฟอร์แมทให้อยู่ในรูปที่จะสามารถส่งผ่านโปรโตคอลได้ ซึ่งจะทำได้ 2 วิธีคือ

1) ส่งข้อมูล 8 บิท ในรูปของเลขฐาน 16 ข้อมูลขนาด 8 บิท แต่ละค่าจะถูกแยกออกเป็น 4 บิทบน และ 4 บิทล่าง และแทนด้วยเลขฐาน 16 เพื่อที่จะส่งออกไปในลักษณะของตัวอักษร (0-9 และ A-F ตามตารางรหัสมาตราฐาน JIS-7)

2) ส่งข้อมูลครั้งละ 6 บิท แทนด้วยตัวอักษรที่สามารถแสดงได้ ถ้าพิจารณาจาก ตารางรหัสมาตราฐาน JIS-7 จะเห็นว่ารหัสที่จะสามารถส่งผ่านโดยโปรโตคอล แบบ Level 2B ได้ จะอยู่ในช่องที่เป็นอักษรที่แสดงได้ ซึ่งมีค่าจาก 20H ถึง 5FH เป็นจำนวน 64 ค่าเท่านั้น ถ้าจะใช้แทนข้อมูลสัญญาณก็จะแทนได้ 64 ระดับ หรือจะส่งได้ครั้งละ 6 บิท จากข้อมูลจริง 8 บิท

ในการส่งข้อมูลที่อยู่เรียงลำดับกันค่าละ 8 บิท จะถูกเลื่อน (Shift) เพื่อส่งออกครั้งละ 6 บิท ข้อมูลทุก ๆ 6 บิทที่จะส่งออกมีค่าระหว่าง 00H ถึง 3FH จะถูกบวกด้วยค่า 20H เพื่อปรับค่าให้อยู่ในช่วงเดียวกับอักษรที่แสดงได้ในตารางรหัสมาตราฐาน JIS-7 (20H ถึง 5FH) และส่งออกไปในรูปของตัวอักษร 1 ตัวแทนค่าข้อมูลครั้งละ 6 บิท ดังแสดงในรูป 5.4



รูปที่ 5.4 แสดงการแปลงข้อมูล 8 บิท เป็นอักษรที่สามารถส่งผ่าน โปรโตคอลแบบ Level 2B ได้

ส่วนตัวอย่างโปรแกรมในการแปลงข้อมูล ทั้งจากข้อมูลเป็นอักษร และจากอักษรเป็นข้อมูล ได้แสดงไว้ในภาคผนวก ค.

บทที่ 6

บทสรุป

โปรโตคอล คอนเวอร์เตอร์ ที่ได้ออกแบบในบทที่ 5 ได้นำไปประยุกต์ใช้งาน เพื่อการถ่ายข้อมูลภาพถ่ายดาวเทียมสำรวจทรัพยากรธรรมชาติ LANDSAT โดยข้อมูลที่ได้รับจากสถานีรับภาพดาวเทียมอยู่ในรูปเทปแม่เหล็ก CCT (Computer compatible tape) ความหนาแน่นข้อมูล 1600 BPI นำมาอ่านบนเครื่องเมนเฟรม NEC ACOS-300 และส่งผ่านระบบสื่อสารข้อมูล มายังเครื่องคอมพิวเตอร์ เพื่อบันทึกลงในแผ่นดิสเกตต์ บนเครื่องไมโครคอมพิวเตอร์ IBM PC ภายใต้โปรแกรมควบคุมระบบแบบ MS-DOS.

โปรแกรมที่ถูกพัฒนาขึ้นบนเครื่อง NEC ACOS-300 ใช้ภาษาโคบอล (COBOL) เป็นตัวจัดการในการอ่านข้อมูลจากเทปแม่เหล็ก และจัดการด้านการสื่อสาร ส่วนการแปลงฟอร์แมทข้อมูล จากข้อมูลภาพขนาด 8 บิท ให้อยู่ในรูปที่สามารถส่งผ่านระบบสื่อสารได้ โดยการใช้โปรแกรมย่อย ซึ่งพัฒนาขึ้นโดยใช้ภาษาฟอร์แทรน-4 (FORTRAN-IV) ส่วนทางด้านเครื่องไมโครคอมพิวเตอร์ ใช้ภาษาปาสคาล (Turbo Pascal) ในการติดต่อกับคอนเวอร์เตอร์ และบันทึกข้อมูลลงดิสเกตต์

ในการถ่ายไฟล์ข้อมูลด้วยระบบเดิม คือถ่ายจากเครื่องเมนเฟรมลงดิสเกตต์ขนาด 8 นิ้ว และถ่ายแปลงฟอร์แมทมาสู่ดิสเกตต์ 5.25 นิ้ว บนเครื่อง IBM PC โดยใช้ข้อมูลในการทดลอง ดังนี้

ข้อมูลภาพถ่าย	256 ระดับ (บิท)
ขนาดภาพ	128 x 128 จุด (Pixels)
จำนวนแบนด์ของภาพ	4 แบนด์
คิดเป็นจำนวนข้อมูล	64 กิโลไบท์

เวลาที่ใช้ในการถ่ายข้อมูลตลอดขบวนการ แบ่งได้เป็น 3 ขั้นตอน และแต่ละขั้นตอนจะใช้เวลาโดยประมาณดังนี้

ขั้นตอนที่ 1	เตรียมโปรแกรมสำหรับเครื่องเมนเฟรม	3 นาที
ขั้นตอนที่ 2	ถ่ายข้อมูลจากดิสก์ของเครื่องเมนเฟรมสู่ดิสเกตต์ 8 นิ้ว	5 นาที
ขั้นตอนที่ 3	แปลงข้อมูลจากดิสเกตต์ 8 นิ้ว สู่ 5.25 นิ้ว	12 นาที
รวมเวลาตลอดขบวนการประมาณ		20 นาที

สำหรับการถ่ายข้อมูลผ่านระบบการสื่อสารข้อมูลและโปรโตคอล คอนเวอร์เตอร์ที่ออกแบบ และโปรแกรมที่พัฒนาขึ้นมา โดยใช้ระบบการส่งผ่านข้อมูลแบบเลขฐาน 16 (ดังได้กล่าวมาแล้วในหัวข้อ 5.4) ได้ผลดังนี้

ความยาวข้อมูลบล๊อคละ	256 ตัวอักษร หรือ 128 ไบท์
จำนวนบล๊อคในการส่ง คือ	128x4 = 512 บล๊อค
เวลารวมที่ใช้ตลอดการส่งโดยประมาณ	36 นาที

จากผลการทดลองข้างต้นจะเห็นว่า ความเร็วในการถ่ายข้อมูลยังต่ำมากเมื่อเทียบกับระบบเดิม ขั้นต่อไปจึงได้แก้ไขโดยการส่งครั้งละ 6 บิท ต่อตัวอักษร (ดังได้กล่าวมาแล้วในหัวข้อ 5.4) ซึ่งได้ผลการทดลองดังนี้

ความยาวข้อมูลบล๊อคละ	256 ตัวอักษร หรือ 192 ไบท์
จำนวนบล๊อคในการส่ง คือ	65,536/192 = 342 บล๊อค
เวลารวมที่ใช้ตลอดการส่งโดยประมาณ	28 นาที

- การปรับปรุงในขั้นที่สองได้ผลที่ดีขึ้น แต่ยังใช้เวลานานกว่าการถ่ายโดยวิธีเดิม จึงได้วิเคราะห์การทำงานของระบบอีกครั้งหนึ่ง พบว่าในขณะที่มีการรับข้อมูลสู่คอนเวอร์เตอร์ มีการส่งสัญญาณ WABT

(Wait before transmission) ไปยังเครื่องเมนเฟรม อันเนื่องมาจากโปรแกรมที่พัฒนาด้วยภาษาปาสคาล ได้มีการแสดงผลไปด้วย ในขณะที่กำลังรับข้อมูลจากเครื่องเมนเฟรม ทำให้ไม่สามารถถ่ายข้อมูลออกจากคอนเวอร์เตอร์ได้เร็วเพียงพอ ก่อนที่ข้อมูลบล๊อคต่อไปจะถูกส่งจากเครื่องเม่นเฟรม จึงได้แก้ไขในส่วนดังกล่าว และทำการทดลองใหม่ โดยส่งครั้งละ 6 บิทต่อตัวอักษร ซึ่งได้ผลการทดลองดังนี้

ความยาวข้อมูลบล๊อคละ	256 ตัวอักษร หรือ 192 ไบท์
จำนวนบล๊อคในการส่ง	342 บล๊อค
เวลารวมที่ใช้ตลอดการส่งโดยประมาณ	18 นาที

จากการปรับปรุงครั้งนี้ผลที่ได้เป็นที่น่าพอใจ เมื่อเทียบกับการถ่ายโดยวิธีเดิมซึ่งให้ผลดีกว่าประ 10 เปอร์เซ็นต์

เนื่องจากการสื่อสารที่ใช้เป็นแบบ Level 2B ที่ความเร็ว 2400 BPS ดังนั้นในการปรับปรุงขั้นต่อไปเพื่อให้ได้ความเร็วสูงขึ้น อาจทำได้ดังนี้

1. เพิ่มความเร็วในการรับส่งข้อมูลเป็น 4800 BPS หรือ 9600 BPS โดยการเปลี่ยนโมเดม และปรับปรุงระบบให้สามารถทำงานที่ความเร็วดังกล่าวได้

2. เปลี่ยนระบบโปรโตคอลการสื่อสารที่ใช้เป็นแบบ Level 4 เพื่อเพิ่มความเร็วในการรับส่ง และแก้ปัญหาเรื่องขีดจำกัดของข้อมูลและรหัสที่ใช้เป็นอักษรควบคุม

อย่างไรก็ตาม โปรโตคอล คอนเวอร์เตอร์นี้ ได้ออกแบบเพื่อให้มีความสามารถในการส่งผ่านข้อมูล สำหรับการประมวลสัญญาณทางด้านดิจิตอล เท่าที่ความสามารถของระบบการสื่อสารข้อมูลของเครื่องเมนเฟรม NEC ACOS-300 ที่มีอยู่จะเอื้ออำนวย และคาดว่าในการปรับปรุงความขั้นตอนดังกล่าวจะช่วยให้คอนเวอร์เตอร์นี้ มีประโยชน์ในการใช้งานมากขึ้น ซึ่งจะสามารถนำไปประยุกต์ใช้งานในด้านต่าง ๆ ได้เช่น

- การวิเคราะห์ภาพถ่ายดาวเทียม
- การสร้างและปรับปรุงคุณภาพของภาพด้วยคอมพิวเตอร์
- การเก็บภาพ X-ray ทางด้านการแพทย์
- การทำระบบภาพสำหรับแผนกบุคคลากร
- การเก็บข้อมูลโดยระบบออนไลน์ เป็นต้น

และสามารถช่วยให้การศึกษาทางด้านการประมวลสัญญาณเป็นที่กว้างขวางขึ้น ตามจุดมุ่งหมายของการทำวิทยานิพนธ์ครั้งนี้

กิตติกรรมประกาศ

ขอขอบพระคุณ ศ.ดร. ไพรัช อัชยพงษ์ เป็นอย่างสูงที่ได้ให้การประสิทธิประสาทวิชา แก่ผู้เขียน และได้ให้โอกาสผู้เขียนได้ใช้เครื่องมือ และคอมพิวเตอร์ ในการทำวิจัยจนวิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยดี ขอขอบคุณกลุ่มนักศึกษาปริญญาตรี ที่ได้ช่วยเหลือในการทำวิทยานิพนธ์ในขั้นตอนของการประยุกต์ใช้งาน เพื่อให้การทำวิทยานิพนธ์สมบูรณ์ยิ่งขึ้น

และขอขอบคุณ บริษัท คอมโพเน้นท์ ซัพพลาย จำกัด ที่ได้ให้ความช่วยเหลือสนับสนุนอุปกรณ์ในการจัดทำต้นฉบับวิทยานิพนธ์ฉบับนี้ ไว้ ณ. ที่นี้ด้วย

เอกสารอ้างอิง

1. A.Osborne,"Some Real Support Devices.",An introductuion to Microcomputers, vol. 3,CA.:Osborne & Associates, Inc.,1978.

2. A.S.Tanenbaum,"Computer Networks.",NJ.:Prentice-Hall,1981.

3. Borland International,"Turbo Pascal version 3.0 Reference Manual.",CA.,1985.

4. EXTEL Corporation,"General Protocol Conversion.",Application Notes,C-CDW1-1.

5. Hewlett-Packard,"Data Communication testing.",Training manual, Part No. 5952-4973.

6. IBM Corporation,"General Information - Binary Synchronous Communications.",GA27-3004-1.

7. IBM Corporation,"Technical Reference Personal Computer XT.", IBM Computer Hardware Reference Library,6936808.

8. J.Holter,"Add Multiple Tasks to Your Communication and Control Program.",Byte journal,pp.445-478,Sept.1983.

9. NEC Corporation,"Communication Function Description.",XAE41E-2.

10. NEC Corporation,"Instruction manual for Datax N2400 A4 & N2400 B4 Modem.",DOI-E01572.

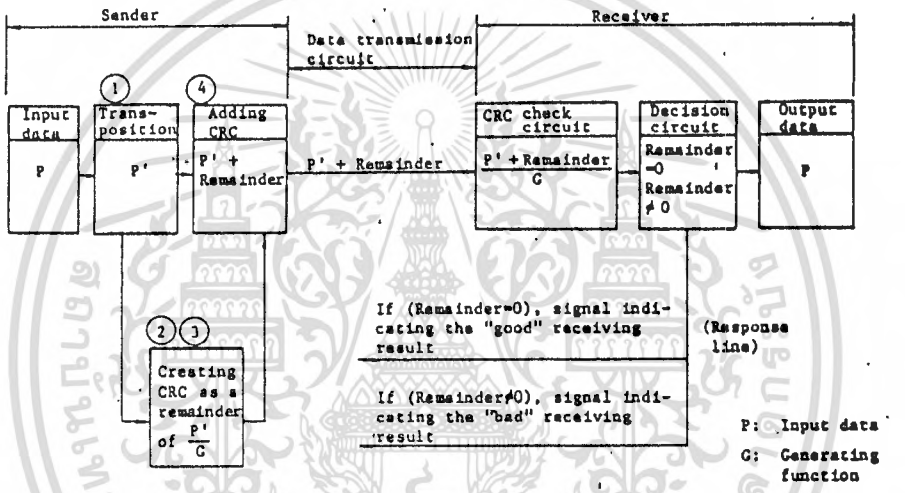
11. NTT.,"Transmittion Control Procedure. (Basic)",Tokyo:NTT.

12. W. Rujeraprapa,"A Design and Development of NEC ACOS-4 Online Network.",M.Eng. thesis,Faculty of engineering,KMITL,1982.

ภาคผนวก ก.

การตรวจสอบความผิดพลาดของข่าวสารโดยวิธี CRC

1. การสร้างรหัสตรวจสอบ CRC.



ระบบการตรวจสอบความผิดพลาดของข่าวสารโดยวิธี CRC.

(ก) ข้อมูลขาเข้า P จะถูกทรานสฟอร์มโดยกฎในข้อ (จ) ได้เป็น P'

(ข) P' จะถูกหารด้วยโพลีโนเมียล G (Generating Function) ที่มีค่าคงที่

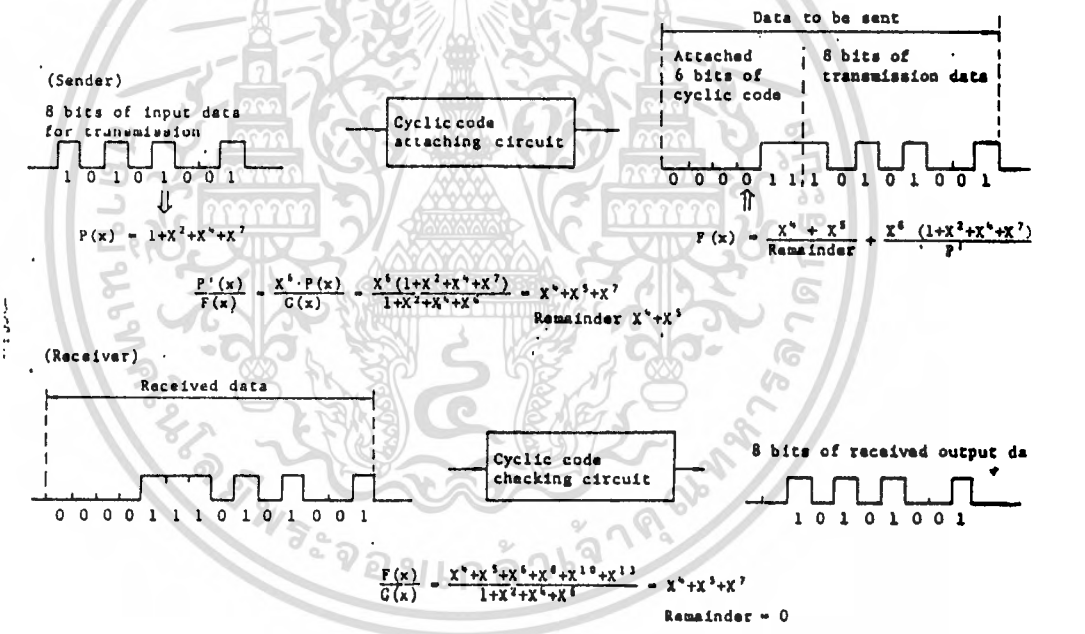
(ค) เศษที่ได้จากการหารในข้อ (ข) จะเป็นรหัส CRC ในการตรวจสอบ

(ง) รหัสในการตรวจสอบ จะถูกบวกเข้ากับ P เป็นข้อมูลที่ใช้ส่ง

(จ) P' สามารถหาได้จากการคูณ P ด้วยเทอมที่มีกำลังสูงสุดของโพลีโนเมียล G.

2. การตรวจสอบความผิดพลาด

ทางฝ่ายรับ จะรับข้อมูลได้ในรูป P'+ เศษของการหาร การตรวจสอบทำได้โดยการหารข้อมูลที่ได้รับได้ด้วยโพลีโนเมียล G ตัวเดียวกับที่ใช้ทางด้านส่ง ถ้าผลลัพธ์จากการหารครั้งนี้ หารลงตัว คือมีเศษ เป็น 0 แสดงว่าข้อมูลที่รับได้ถูกต้อง (Good received) ถ้าเศษดังกล่าวมีค่าไม่เป็น 0 แสดงว่าข้อมูลที่รับได้มีการผิดพลาด (Error)



ตัวอย่างการส่งและรับข้อมูล ที่มีการตรวจสอบ CRC.

(ก) การแทนบิทของข้อมูลด้วย โพลีโนเมียล

ข้อมูล ดังแสดงในรูป มีค่าเป็น 10010101 ในระบบฐานสอง (ซึ่งมีลำดับกลับกันกับในรูป เนื่องจากในรูปมีการเรียงบิทตามลำดับการส่ง) จะสามารถแทนด้วย สมการโพลีโนเมียล คือ

เลขฐานสอง	1	0	0	1	0	1	0	1
โพลีโนเมียล	x^7			$+ x^4$		$+ x^2$		$+ 1$

(ข) การบวกหรือ ลบทางลอจิกในการคำนวณ CRC.

ในการคำนวณ CRC การบวกหรือลบทางลอจิคเป็นแบบ Modulo-2 ซึ่งมีการทำงานดังนี้

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 0$$
$$-1 = 1$$

(ค) การคำนวณเพื่อสร้างบิทตรวจสอบ

- ข้อมูลขาเข้าดังตัวอย่างในรูป คือ P(x) ซึ่งมีค่าเป็น $x^7 + x^4 + x^2 + 1$
- เทอมที่มีกำลังสูงสุดของ G(x) ($x^6+x^4+x^2+1$) คือ x^6 ดังนั้นจะได้

$$P'(x) = x^6(x^7 + x^4 + x^2 + 1)$$
$$= x^{13} + x^{10} + x^8 + x^6$$

- ทำการหาร P'(x) ด้วย G(x)

$$
\begin{array}{rl}
 & x^7 + x^5 + x^4 \\
x^6+x^4+x^2+1 \;\big)\; & x^{13} + x^{10} + x^8 + x^6 \\
 & \underline{x^{13} + x^{11}} \\
 & x^{11} + x^{10} + x^9 + x^8 + x^7 + x^6 \\
 & \underline{x^{11} + x^9 + x^7 + x^5} \\
 & x^{10} + x^8 + x^6 + x^5 \\
 & \underline{x^{10} + x^8 + x^6 + x^4} \\
 & x^5 + x^4
\end{array}
$$

เศษจากการหารคือ x^5+x^4 บิทตรวจสอบหรือรหัส CRC.

(ง) ข้อมูลที่จะทำการส่ง

เมื่อ F(x) คือข้อมูลที่จะทำการส่ง ซึ่งหาได้จาก P'(x)+ เศษของการหาร

$$F(x) = P'(x) + R(x)$$

$$= x^{13} + x^{10} + x^8 + x^6 + x^5 + x^4$$

(จ) การตรวจสอบความผิดพลาดทางด้านรับ

ข้อมูลที่รับได้คือ F(x) และตรวจสอบได้โดยการหารด้วยโพลีโนเมียล G(x) เพื่อดูผลลัพท์ที่ได้

$$
\begin{array}{l}
\qquad\qquad x^7 + x^5 + x^4 \\
x^6+x^4+x^2+1 \;\overline{)}\; x^{13} + x^{10} + x^8 + x^6 \\
\qquad\quad \underline{x^{13} + x^{11}} \\
\qquad\qquad x^{11} + x^{10} + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 \\
\qquad\quad \underline{x^{11} + x^9 + x^7 + x^5} \\
\qquad\qquad\quad x^{10} + x^8 + x^6 + x^4 \\
\qquad\qquad\quad \underline{x^{10} + x^8 + x^6 + x^4} \\
\qquad\qquad\qquad\qquad 0 \\
\qquad\qquad\qquad\qquad \underline{\text{No remainder}}
\end{array}
$$

เมื่อผลลัพท์ที่ได้มีค่าเป็น 0 แสดงว่าข้อมูลที่รับได้ถูกต้อง

ส่วนของข้อมูลที่ต้องการจะหาได้จากการ หารข้อมูล F(x) ด้วยเทอมที่มีค่าสูงสุด ของโพลีโนเมียล G(x) และตัดเศษที่เหลือทิ้ง หรืออีกวิธีหนึ่งโดยการเลื่อน (Shift) ข้อมูล F(x) ไปทางขวาเป็นจำนวนบิทเท่ากับจำนวนบิทของ CRC.

โพลีโนเมียล G(x) ที่ใช้ในระบบโปรโตคอลแบบ Level 2B มีค่าเป็น $x^{16} + x^{12} + x^5 + 1$ ซึ่งจะให้ค่าเศษหรือรหัส CRC มีความยาว 16 บิท หรือ 2 ไบท์ ดังนั้น BCC (Block Check Character) ในโปรโตคอล แบบ Level 2B ก็จะประกอบด้วยรหัส CRC จำนวน 2 ไบท์

ภาคผนวก ข.

โปรแกรมตัวอย่างการเรียกใช้โปรโตคอล คอนเวอร์เตอร์ ในการประยุกต์ใช้งาน ซึ่งเขียนขึ้นโดยใช้ภาษา Pascal (Turbo Pascal Version 3.0) ให้ทำงานเป็นเทอร์มินัลอย่างง่าย ๆ ของเครื่องเมนเฟรม NEC ACOS-300 ใช้ในการทดสอบการสื่อสาร และใช้ช่วยในการออกแบบโปรแกรม การรับส่งข้อมูลบนเครื่องเมนเฟรม

```
(*********************************************************************)
(*                                                                   *)
(*    Demonstration program for NEC Level 2B protocol converter.     *)
(*                    By  Manoon Chinnakarn.                         *)
(*                                                                   *)
(*   Computer Research and Service Center, KMIT Ladkrabang 1986.     *)
(*********************************************************************)

program  Protocol_sample;

const
     (* Converter Board I/O Port's bit definitions.          *)
     (*  CBstatPA     ! XXXX ! DSR ! CDC !ToHost!ToTerm!     *)
     (*  CBstatPB     ! XXXX !IRQ4S!IRQ3S!CBRFR!CBRFW!      *)
     (*  CBdataPx     ! -- 8bit bidirection port --  !      *)
     (*  CBstatPC     ! XXXX !IRQ4E!IRQ3E!OUTS1!OUTS0!      *)
         CBstatPA    = $3ac;
           CDCbit    = $08;
           DSRbit    = $04;
           ToHost    = $02;
           ToTerm    = $01;
         CBstatPB    = $3ad;
           IRQ4E     = $08;
           IRQ4S     = $04;
           CBRFW     = $02;
           CBRFR     = $01;
         CBdataPx    = $3ae;
         CBstatPC    = $3ac;
         CBreset     = $3af;

     (* Coverter Board Command code *)
         RdStatCmd   = 1;              (* Read converter board status    *)
         SetAdrCmd   = 2;              (* Set DTE address                *)
         SetRecCmd   = 3;              (* Set send record to conv.       *)
         GetRecCmd   = 4;              (* Get received record from conv. *)
         SetMONCmd   = $0e;            (* Turn monitoring port ON/OFF.   *)
         CBdumpCmd   = $0f;            (* Dump converter board memory    *)

     (* Terminal unit addresses *)
         StationAdr = $30;             (* Point-to-Point config.         *)
         DispAdr    = $28;             (* Output device is CRT display   *)
         KbrdAdr    = $48;             (* Input device is keyboard       *)

     (* Communication uses ASCII code *)
         ETX = $03; ETB = $17; SO = $0e; SI = $0f;
```

-To be continue-

```
    (* Status Field equates for status line display *)
        JobStatFld  = 1;
        TermStatFld = 15;
        KbrdStatFld = 20;
        FnStatFld   = 30;
        ErrStatFld  = 42;
        SegStatFld  = 52;
        RxStatFld   = 70;
        LineStatFld = 71;

        ScreenSeg   = $b800;                    { For monocrome display }
        TouchString = 'Touch any key to continue.';

type
        AnyString  = String[80];
        ShotString = String[20];
        LongString = String[255];
        DataRecord = array[0..255] of byte;
        BytePtr    = ^Byte;
        WordPtr    = ^Integer;
        CommBlock  = array[0..1023] of byte;
        Tristate   = (close,unknow,open);

var
        ComRecvBlock,
        ComSendBlock  : CommBlock;
        ComRecvCount,
        ComSendCount  : integer;
        CommFinish,OK,
        FunctionKey   : Boolean;
        p1,p2         : WordPtr;
        s1            : AnyString;
        r1            : DataRecord;
        ch            : char;
        i             : integer;
        LineSt        : Tristate;
        CurPosX,
        CurPosY       : integer;


(* Reset converter board. *)

procedure  ComBoardReset;
begin
   write(chr(7));                                          { beep! }
   Port[CBreset] := 0;                              { issue reset command }
```

-To be continue-

```
   Delay(300);                          { delay for conv. board initialization }
end;


(* Write one byte to Converter *)

procedure  WriteCB(data:byte);
begin
   Repeat until (Port[CBstatPB] and CBRFW) = CBRFW;    { wait CBRFW status }
   Port[CBdataPx] := data;                             { send data to conv.}
end;


(* Read one byte from Converter *)

procedure  ReadCB(var data:byte);
begin
   repeat until (Port[CBstatPB] and CBRFR) = CBRFR;    { wait CBRFR status }
   data := Port[CBdataPx];                            { get data from conv.}
end;


(* Set Converter board command *)

procedure  SetCBcmd(Cmd:byte);
begin
   WriteCB(Cmd);
end;                                          { write command byte to conv. }


(* Set DTE address for specified configuration *)

procedure  SetAddress(WSA,DUA,KUA:byte);
begin
   SetCBcmd(SetADRcmd);                       { write set address command  }
   WriteCB(WSA);                              {  work statsion address     }
   WriteCB(KUA);                              {  and keyboard unit address }
end;


(* Set Monitor mode on for debugging purpose *)

procedure  Monitoring(mon:boolean);
begin
   SetCBcmd(SetMONcmd);                        { write set monitor command }
   if mon then WriteCB($ff) else WriteCB($0); {  and set corresponse to   }
```

-To be continue-

```
end;                                          {  variable mon              }


(* Test Modem CD line *)

function  CarrierDetected:Boolean;
begin
   if (Port[CBstatPA] and CDCbit)=0 then      { test carrier detected bit }
      CarrierDetected := True                 {  and reflect to status    }
   else
      CarrierDetected := False;
end;


(* Test Modem DSR line *)

function  DataSetReady:Boolean;
begin
   if (Port[CBstatPA] and DSRbit)=0 then      { test data set ready bit  }
      DataSetReady := True                    {  and reflect to status   }
   else
      DataSetReady := False;
end;


(* Get converter board status *)

function  CBstat : byte;
var st : byte;
begin
   SetCBcmd(RdStatCmd);                       { write read status command }
   ReadCB(st);                                {  and read status back     }
   CBstat := st;
end;


(* Checking for Received frame available *)

function  RxFrameAvail:Boolean;
begin
   if (Port[CBstatPA] and ToTerm)=ToTerm then{ test status bit from conv. }
      RxFrameAvail := True                   {  and reflect thru function }
   else
      RxFrameAvail := False;
end;
```

-To be continue-

```
(* Checking send buffer *)

function  TxBufferEmpty:Boolean;
begin
   if (Port[CBstatPA] and ToHost)=ToHost then{ test status bit from conv. }
      TxBufferEmpty := False                 {  and reflect thru function }
    else
      TxBufferEmpty := True;
end;

procedure WrtHex(hx:integer);

  procedure WrtByte(hb:byte);                          { write a data byte in }
  var hh, hl : byte;                                   {  hex format          }
  begin
     hh := hb div $10;
     hl := hb mod $10;
     if hh>9 then Write(chr(hh-10+$41)) else Write(hh:1);
     if hl>9 then Write(chr(hl-10+$41)) else Write(hl:1);
  end;

begin
   if hx>$ff then                                   { if data>255 write 2 data  }
      begin                                         {  bytes in hex             }
         WrtByte(hx div $100);
         WrtByte(hx mod $100);
      end
   else                                             { if data<=255 write 1 data }
      WrtByte(hx);                                  {  byte in hex              }
end;

(* Receive one information record *)

procedure  RecvFromCB(var Shift,RecType:Byte; var Count:integer;
                      var XferRec:DataRecord);
var   Btemp    : Byte;
      index    : integer;
begin
   repeat until RxFrameAvail;                              { waiting for frame }
   SetCBcmd(GetRecCmd);                                { set receive record command }
   ReadCB(RecType);                                        { get record type   }
   ReadCB(Shift);                                          { get shift code    }
   ReadCB(Btemp);   Count := Btemp;                        { get data length   }
   if (RecType=ETB) and (Count=0) then Count := 256;
   for index:=0 to Count-1 do          { and get data from conv. to buffer }
```

-To be continue-

```
      ReadCB(XferRec[index]);
   for index:=1 to 5 do                          { Flush all remaining bytes }
      Btemp := Port[CBdataPx];
end;


(* Send one information record *)

procedure  SendToCB(var Shift,RecType:Byte; var Count:integer;
                              var XferRec:DataRecord);
var   Btemp     : byte;
      index     : integer;

begin
   repeat until TxBufferEmpty;                    { waiting for buffer empty }
   Btemp := Count;
   SetCBcmd(SetRecCmd);                           { set write record command }
   WriteCB(RecType);                                    { set record type  }
   WriteCB(Shift);                                      { set shift code   }
   WriteCB(Btemp);                                      { set data length  }
   for index:=0 to Count-1 do          { and send data from buffer to conv. }
         WriteCB(XferRec[index]);
end;

procedure  DumpFromCB(Addr, count : integer);
var   Btemp     : Byte;
      index     : integer;
begin
   SetCBcmd(CBdumpCmd);                     { set dump conv.'s memory command }
   WriteCB(Addr mod 256);                   { set address of memory to dump   }
   WriteCB(Addr div 256);                   { low byte / High byte            }
   if count>255 then count := 256;
   WriteCB(count mod 256);                        { set data length to dump }
   for index:=0 to Count-1 do
      begin
         if (count mod 16)=0 then
            begin                                 { dump format :              }
               Writeln;                           { aaaa>> hh hh ..16 bytes..  }
               WrtHex(Addr+count);                { aaaa=address, hh=hex data  }
               Write('>>');
            end;
         ReadCB(Btemp);
         WrtHex(Btemp);
         Write(' ');
      end;
      Writeln;
```

-To be continue-

```
end;

(****   Dressing up screen with Turbo Pascal function   ****)

procedure WrtInv(S:AnyString);              { display string on screen with }
var   CurPos : BytePtr;                     {  invert video attribute       }
      i      : integer;
      C999   : Char;
begin
   for i:=1 to length(S) do
      begin
         C999 := Copy(S,i,1);
         Write(C999);
         If (C999<>#08) and (C999<>#127) and (C999<>#13) then
            begin
               CurPos := ptr(ScreenSeg,(WhereY*160)+(WhereX*2-3));
               CurPos^ := $70;
            end;
     end;
end;

procedure WrtInvLn(S:AnyString);              { display invert video line }
begin
   WrtInvLn(S);
   WriteLn;
end;

procedure TopLine(Pos:integer;I:char;S:AnyString);
var   Ptr1 : byteptr;
begin                                         { display top line of screen }
   if Pos in [1..80] then                     {  as status line            }
      begin
         CurPosX:=WhereX; CurPosY:=WhereY;
         Window(1,1,80,25);
         if Upcase(i) in ['H','L'] then
            case Upcase(i) of
               'H' : HighVideo;
               'L' : LowVideo;
            end;
         GotoXY(Pos,1); Write(S); HighVideo;
         Window(1,2,80,25);
         GotoXY(CurPosX,CurPosY);
     end;
end;
```

-To be continue-

```
procedure ClrTopLine;                                    { clear status line }
var   i : integer;
begin
   CurPosX:=WhereX; CurPosY:=WhereY;
   Window(1,1,80,25);
   GotoXY(1,1);
   for  i:=1 to 80 do Write(' ');
   Window(1,2,80,25);
   GotoXY(CurPosX,CurPosY);
end;

procedure BottomLine(S:AnyString);          { display bottom line of screen }
begin                                       {  as prompt line               }
   CurPosX:=WhereX; CurPosY:=WhereY;
   LowVideo;
   GotoXY(1,24);
   ClrEol;
   Write(S);
   HighVideo;
   GotoXY(CurPosX,CurPosY);
end;

procedure  LinkStat;                    { diplay status of communication line }
begin                                   {  carrier=open, no carrier=close     }
   if CarrierDetected and (LineSt<>Open) Then
      begin
         TopLine(LineStatFld,'1','L-Open ');
         LineSt := Open;
      end;
   if (Not CarrierDetected) and (LineSt<>Close) then
      begin
         TopLine(LineStatFld,'1','L-Close');
         LineSt := Close;
      end;
end;

function GetKey(var FunctionKey: Boolean): char;
var Ch: char;
begin                                          { get IBM-PC's function key }
   read(kbd,Ch);
   If (Ch = #27) and KeyPressed Then
      begin
         read(kbd,Ch);          { Return value for F1..f10 is ;<=>?@ABCD }
         FunctionKey := True;
      end
   else FunctionKey := False;
```

-To be continue-

```
   GetKey := Ch;
end;

procedure GetKbdLine(var S:AnyString);     { set prompt line and get input }
const   wrtID = #$4b;                      {  line from keyboard           }
        MTB   = #$09;
var     CCA,CRA,Ch : Char;
        CurPos : BytePtr;
begin
   S := '';
   repeat
      CRA := Chr(WhereY+31);  CCA := Chr(WhereX+31);
      Read(Kbd,Ch);
      WrtInv(Ch);
      if (Ch=#08) or (Ch=#127) then Delete(S,Length(S),1)
      else S := Concat(S,Upcase(Ch));
   until (Ch=#$0d) or (WhereX=80);
   S := Concat(wrtID,CCA,CRA,MTB,S);
   Delete(S,Length(s),1);
end;

procedure  RecvOneBlock(var BlockLength:integer; var RecvBuffer:CommBlock);
var  ShiftCode,RecType : byte;
     RecLength,index   : integer;
     RecvRecord        : DataRecord;
begin
   BlockLength := 0;
   TopLine(TermStatFld,'1','[Rx]  Wait  ');   { set status when receiving }
   repeat
      RecvFromCB(ShiftCode,RecType,RecLength,RecvRecord);{ receive record }
      for index:=0 to RecLength-1 do                { and concat to buffer }
         RecvBuffer[BlockLength + index] := RecvRecord[index];
      BlockLength := BlockLength + RecLength;
      if BlockLength > 1023 then BlockLength := 0; { 1 blk limited to 1kB }
   until RecType = ETX;                           { repeat until 1 blk done }
end;

procedure  SendOneBlock(var BlockLength:integer; var SendBuffer:CommBlock);
var  RecType, ShiftCode : byte;
     RecLength, ByteCount : integer;
     BlockEnd   : boolean;
     SendRecord : DataRecord;
begin
   ShiftCode := SI;
   ByteCount := 0;
   RecType   := ETB;
```

-To be continue-

```
   BlockEnd  := False;
   TopLine(TermStatFld,'l','[Tx]  Wait  ');      { set status when sending }
   repeat
      RecLength := 0;
      while (RecLength<256) and (BlockEnd=False) do
      begin
         if (ByteCount>=BlockLength) then
            begin
               BlockEnd := True;                    { set block end status }
               RecType := ETX;                      {  and record type     }
            end
         else
            begin                          { cut block into record to be send }
               SendRecord[RecLength] := SendBuffer[ByteCount];
               ByteCount := ByteCount+1; RecLength := RecLength+1;
            end;
      end;
      SendToCB(ShiftCode,RecType,RecLength,SendRecord); { send record out }
  until RecType = ETX;                          { repeat until 1 block done }
end;


procedure Terminal;                             { simple terminal begin here }
Label TRM99;
var   i  : integer;
      ch : char;
      Finish : boolean;

procedure  TermRecv;                            { terminal receiving phase }
var   rc : char;
begin
   window(1,2,80,23);
   While RxFrameAvail do                                   { if frame valid }
      begin
         RecvOneBlock(ComRecvCount,ComRecvBlock);          { receives block }
         for i:=0 to ComRecvCount-1 do { display received block on screen }
            begin
               rc := chr(ComRecvBlock[i] and $7f);
               if rc >= ' ' then write(rc)
               else if rc=^M then
                       write(^M,^J)
                    else
                       begin                        { display control char. }
                          Lowvideo;                 {  in low video mode    }
                          Write(chr(ord(rc)+$40));
                          NormVideo;
```

-To be continue-

```
                        end;
                end;{for}
            Delay(200);                          { wait.. if next block coming? }
        end;{while}
    WriteLn;
    CurPosX:=WhereX; CurPosY:=WhereY;
    window(1,2,80,25);
    TopLine(TermStatFld,'l',' KB.  Ready '); {show ready status when finish}
    GotoXY(CurPosX,CurPosY);
end;

procedure  TermSend;
var  SendComplete : boolean;
     TSL1 : integer;
     Ptr1 : byteptr;
begin
   SendComplete := True;
   repeat
      if SendComplete = False Then
         TopLine(TermStatFld,'h',' KB.  Retry '); { retry status if error }
      GotoXY(1,23);
      Writeln('Send? ');                         { prompt for keyboard input }
      for i:=1 to 79 do WrtInv(' ');
      GotoXY(1,24);
      GetKbdLine(s1);                            { get keyboard input line }
      if Copy(s1,5,2)<>'\*' then                 { check \* (quit) code    }
         begin
            for i:=1 to Length(s1) do ComSendBlock[i-1]:=ord(s1[i]);
            if RxFrameAvail then                 { allow incoming frame coming }
               begin                             {  if available               }
                  TermRecv;
                  SendComplete := False  { and set retry flag for sending }
               end
            else
               begin
                  TSL1 := Length(s1);
                  SendOneBlock(TSL1,ComSendBlock);          { send block }
                  SendComplete := True;          { and set complete status }
               end;
         end
      else
         begin
            SendComplete := True;                { set complete and finish flag }
            Finish := True;                      {  if quit code entered        }
         end;
   until SendComplete;
```

-To be continue-

```
   GotoXY(1,23);      ClrEol;
   GotoXY(1,24);      ClrEol;
   TopLine(TermStatFld,'1',' KB.  Ready ');        { set status when ready }
   GotoXY(CurPosX,CurPosY);
end;

begin                                        { Main program of sample terminal }
   ClrScr;
   TopLine(JobStatFld,'1','<EASY TERM>   ');        { show activity status }
   TopLine(TermStatFld,'1',' KB.  Ready ');         { show ready status    }
   Finish := False;
   CurPosX :=1; CurPosY :=1;
   repeat
      LinkStat;                                     { display line status  }
      if KeyPressed then
         begin
            Read(Kbd,Ch);                        { check IBM-PC's function key }
            if Ch = #27 then
               begin
                  Read(Kbd,Ch);
                  if Ch = ';' then TermSend;     { term send if F1 pressed }
               end;
         end
      else if RxFrameAvail then TermRecv;     { do receive if frame coming }
   until Finish;
TRM99:
end;

procedure initialize;                       { initialization for sample terminal }
Label  init99;
var    Btemp  : byte;
begin
   ClrScr;
   Window(1,2,80,25);
   BottomLine(TouchString);                                  { wait user touch }
   Read(Kbd,ch);
   ComBoardReset;                                    { reset converter board }
   Port[CBdataPx] := RdStatCmd;                 { throw away first status byte }
   ReadCB(Btemp);
   SetAddress(StationAdr,DispAdr,KbrdAdr);          { set terminal addresses }
(*
   Monitoring(True);                                 { set monitoring if need }
*)
   while DataSetReady=False do                { test modem status and wwarn user }
      begin
         Topline(7,'h','MODEM NOT READY');
```

-To be continue-

```
          Bottomline(concat('Turn on MODEM and ',TouchString));
          Read(kbd,Ch);
          if ch=#27 then                    { ignore modem if escape entered }
             begin
                TopLine(7,'1','                ');
                Goto init99;
             end;
       end;
init99:
end;

BEGIN
   Initialize;                                          { Main program }
   Terminal;                                { initialize terminal program }
END.                                        { perform terminal program    }
```

ภาคผนวก ค

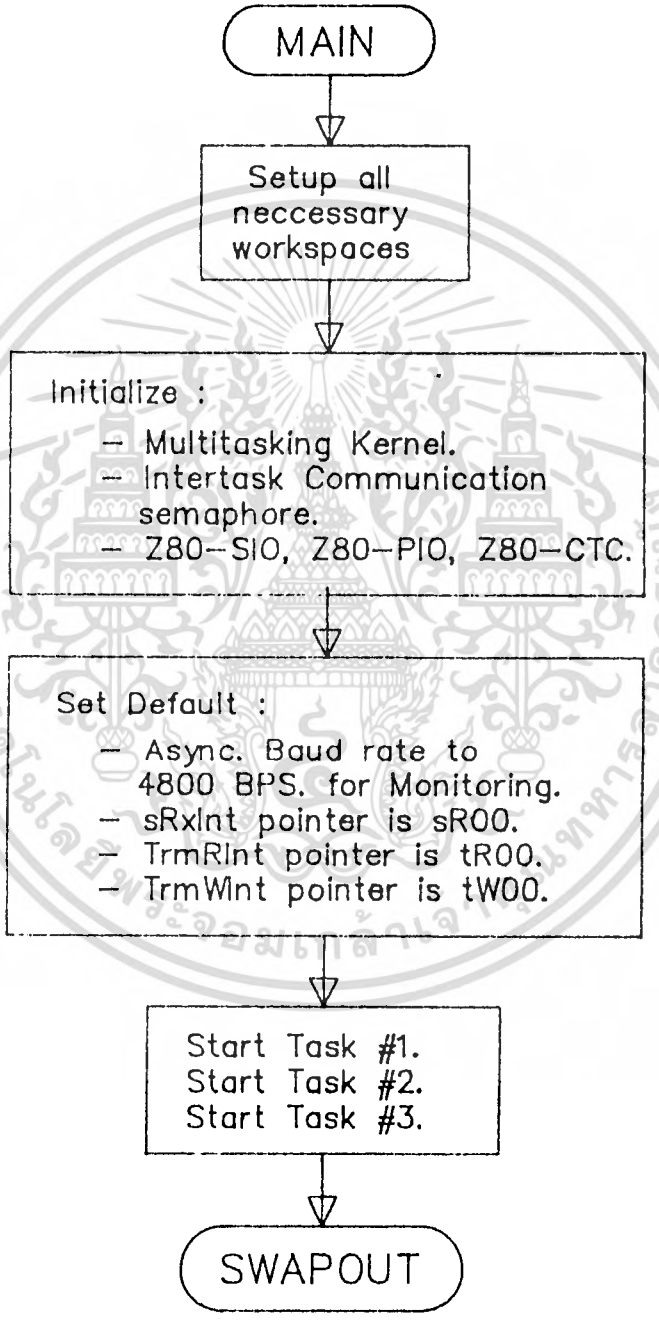
```
;
;         File CBCODE.ASM
;         Assemble sequence :
;             masm cbcode;
;             link cbcode,,;
;             exe2bin cbcode.exe cbcode.bin

code      segment 'code'
          assume  cs:code

;*****************************************************************
;         Function Encode6(var:bf:buffer; i6:integer):char;
;           { buffet type is array [0..BufMax] of byte; }
;         This function will retrive 6 bits word of data
;           indexed by i6 and convert to printable character
;           on return.
;*****************************************************************
;
Encode6   proc    near          ;must be near for Turbo Pascal
          push    bp            ;small memory model.
          mov     bp,sp
          mov     ax,[bp+4]     ;get 6-bits code index
          mov     cx,6          ;calculate byte#, and bit#
          mul     cx            ; byte# := index*6 div 8;
          mov     cx,8          ; bit#  := index*6 mod 8;
          div     cx
          mov     cx,dx         ;No. of bit to be shift
          les     di,[bp+6]     ;get buffer start address
          add     di,ax         ; offset to required byte
          mov     al,es:[di]    ;retrive 6 bits data from
          mov     ah,es:[di+1]  ; from buffer
          ror     ax,cl
          and     ax,003fh
          add     ax,20h        ;adjust to printable char.
          pop     bp
          ret     7             ;clean stack
Encode6   endp
```

```
;*****************************************************************
;         Procedure Decode6(ch:char; var:bf:buffer; i6:integer);
;           { buffet type is array [0..BufMax] of byte; }
;         This procedure will convert character ch to 6 bits
;           data word and put into buffer indexed by i6.
;*****************************************************************
;
Decode6  proc    near
         push    bp
         mov     bp,sp
         mov     ax,[bp+4]     ;get 6-bits code index
         mov     cx,6          ;calculate byte#, and bit#
         mul     cx            ; byte# := index*6 div 8;
         mov     cx,8          ; bit#  := index*6 mod 8;
         div     cx
         mov     cx,dx
         mov     dl,[bp+10]    ;get character
         sub     dl,20h        ; and convert to 6 bits data
         and     dx,003fh
         les     di,[bp+6]     ;get index
         add     di,ax         ; point to location to put
         mov     al,es:[di]    ;merge new 6 bits data to
         mov     ah,es:[di+1]  ; buffer.
         ror     ax,cl
         and     ax,0ffc0h
         or      ax,dx
         rol     ax,cl
         mov     es:[di],al
         mov     es:[di+1],ah
         pop     bp
         ret     8             ;clean stack
Decode6  endp
code     ends
         end
;*****************************************************************
```

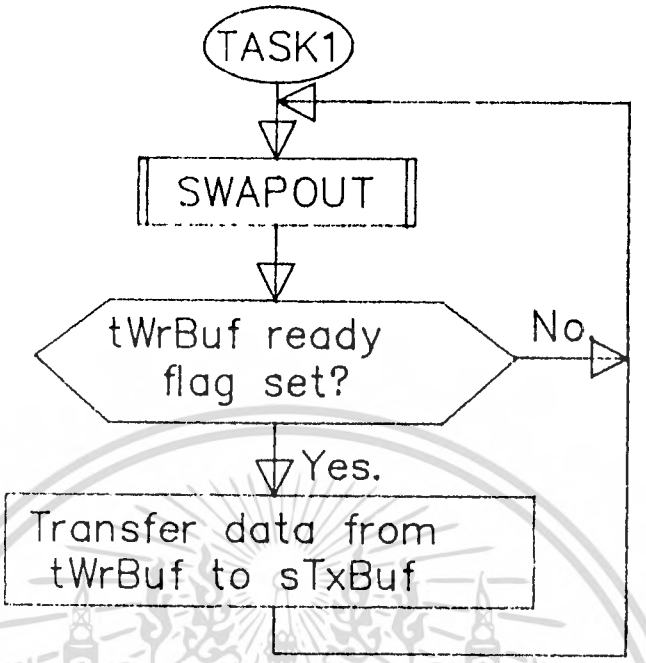
```
(*************************************************************************)
(*  Turbo Pascal example program to call ready made assembly subroutine *)
(*    to convert 8 bits binary data from buffer to JIS-7 character for  *)
(*    Level 2B protocol transmittion (Encode6) and convert character    *)
(*    back to binary data (Decode6).                                    *)
(*************************************************************************)

program  CallExample;

Const BufSize = 1023;     (* Buffer can be any size from 1 to 32768 *)

type  buffer = array [0..BufSize] of byte;

var   index, ii : integer;
      cc    : char;
      ibuf  : buffer;

procedure Coding6;                                    external 'CBCODE.BIN';
function  Encode6(var x:buffer;i6:integer):char;      external coding6[0];
procedure Decode6(c:char;var x:buffer;i6:integer);    external coding6[42];

begin
      ii := 64; writeln;
      writeln('Assume that data is received.');
      writeln('Decode and put in to buffer. (',ii,' bytes)');
      for index := 0 to ii-1 do
         begin
            cc := chr($20+index);
            write(cc);
         (*************************)
            Decode6(cc,ibuf,index);
         (*************************)
         end;

      ii := 63*6 div 8; writeln; writeln;
      writeln('Data in buffer [Decimal] :   (',ii,' bytes)');
      for index := 0 to ii-1 do
         write(ibuf[index]:3,' ');

      ii := 64; writeln; writeln;
      writeln('Encode data to character and send out.');
      for index := 0 to ii-1 do
      (*******************************)
         write(Encode6(ibuf,index));
      (*******************************)
end.
```

ภาคผนวก ง.

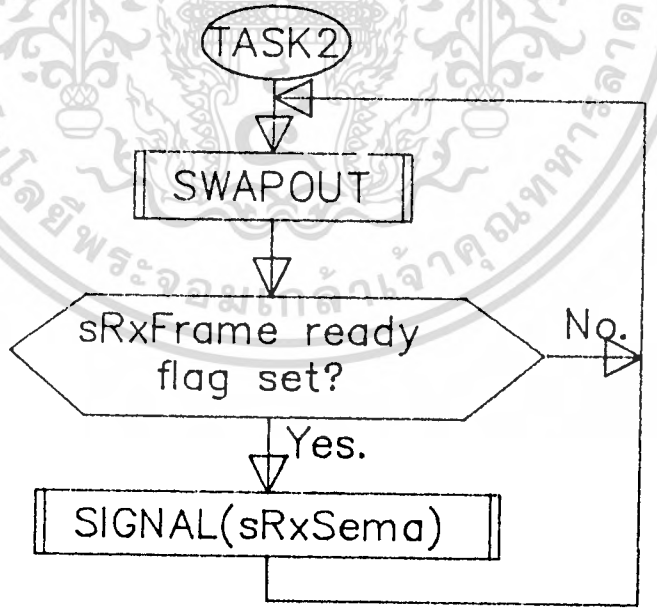
ฟลอ์วชาร์ทแสดงการทำงานของโปรโตคอล คอนเวอร์เตอร์ ที่ออกแบบ.



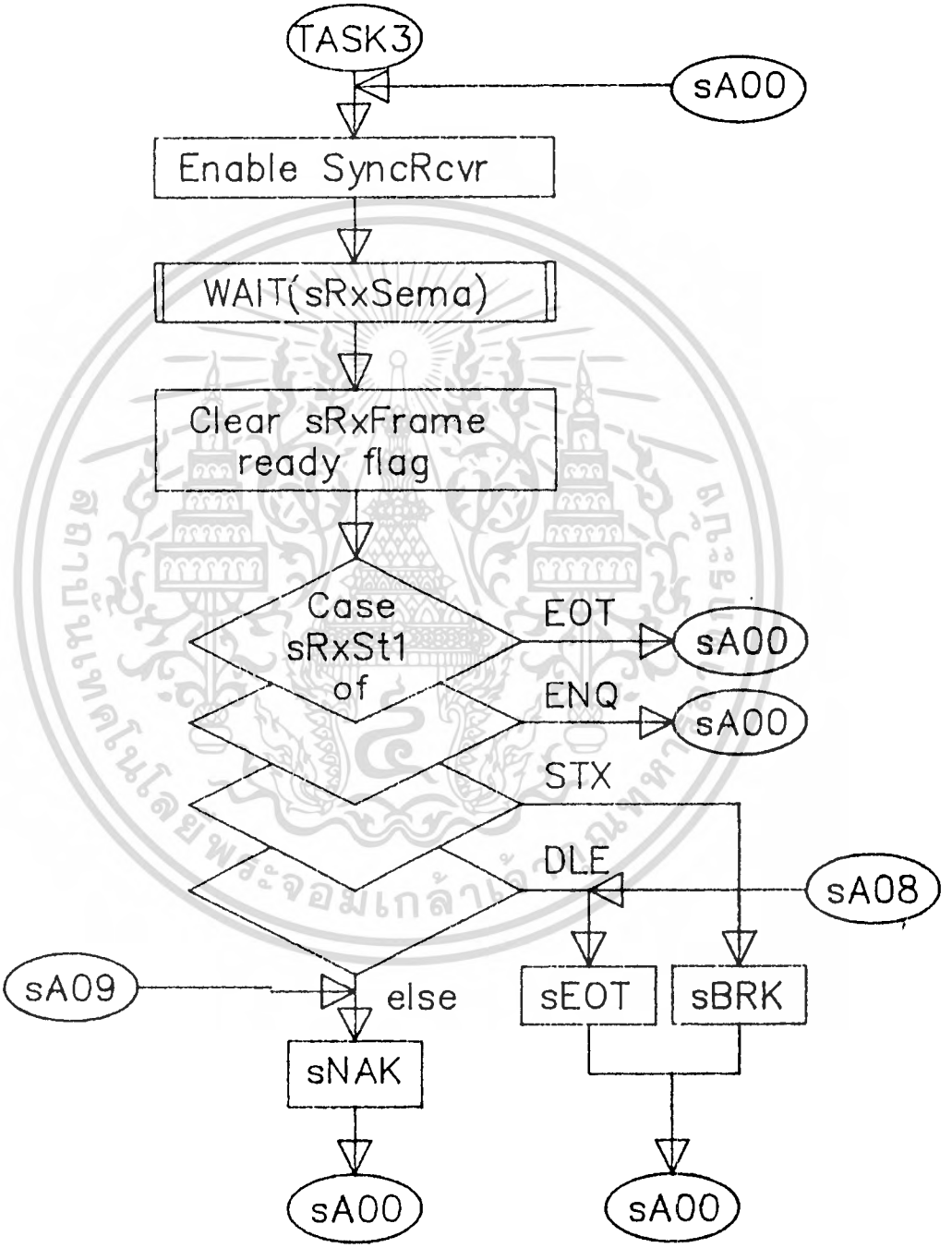
รูป ง.1 โปรแกรมหลักในการจัดเตรียมการทำงานของคอนเวอร์เตอร์



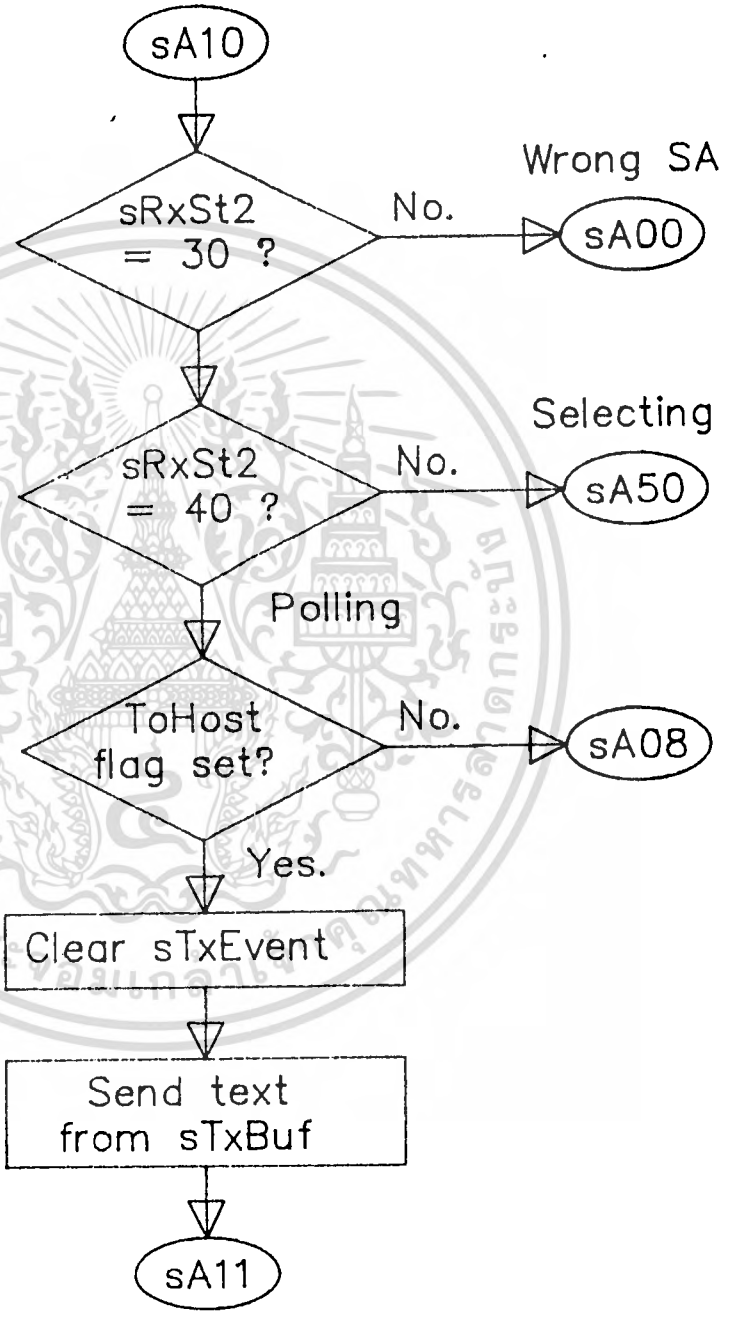
รูป ง.2 ทาส์คที่ 1 รับข้อมูลจากไมโครคอมพิวเตอร์



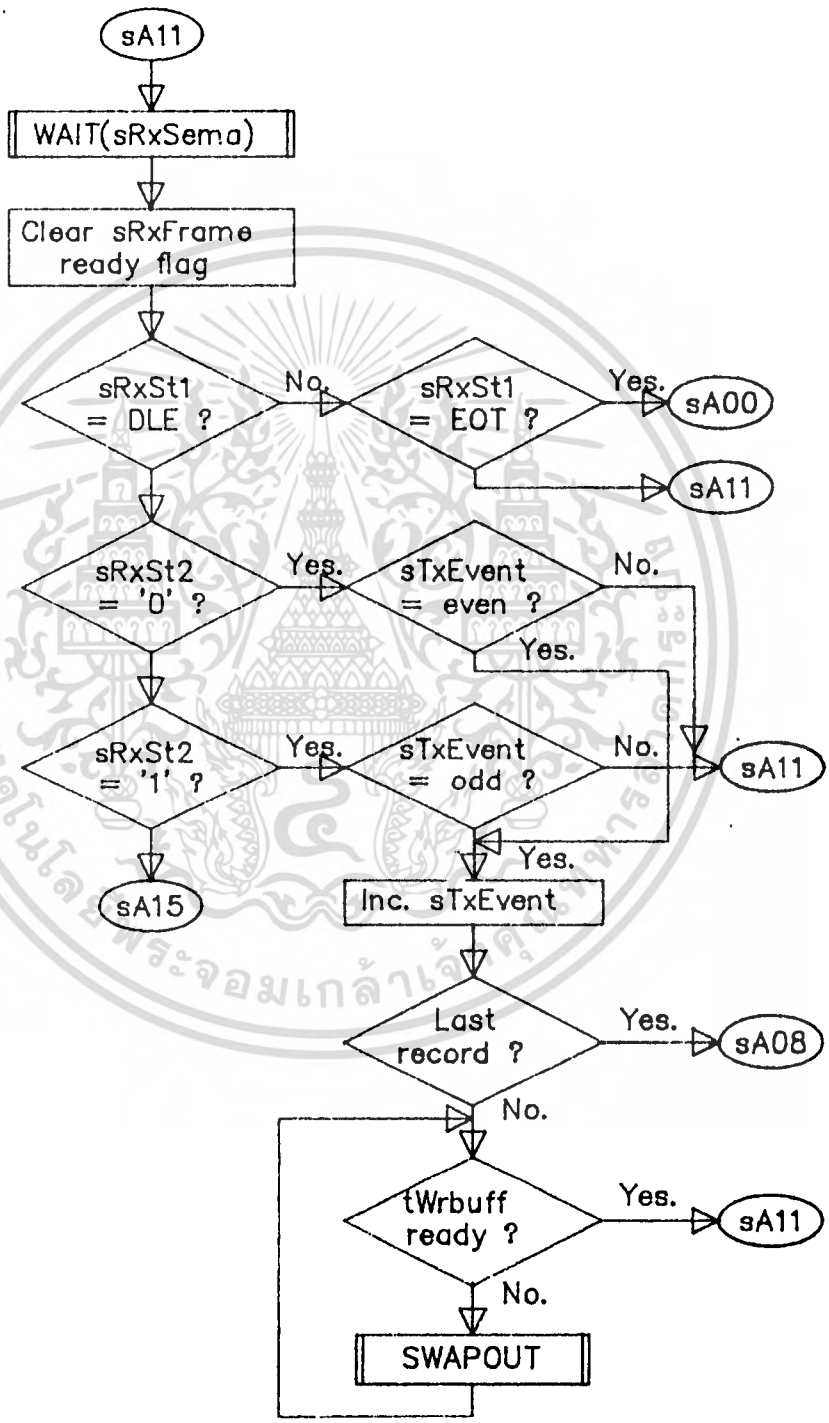
รูป ง.3 ทาส์คที่ 2 รับข้อมูลแบบซิงโครนัส



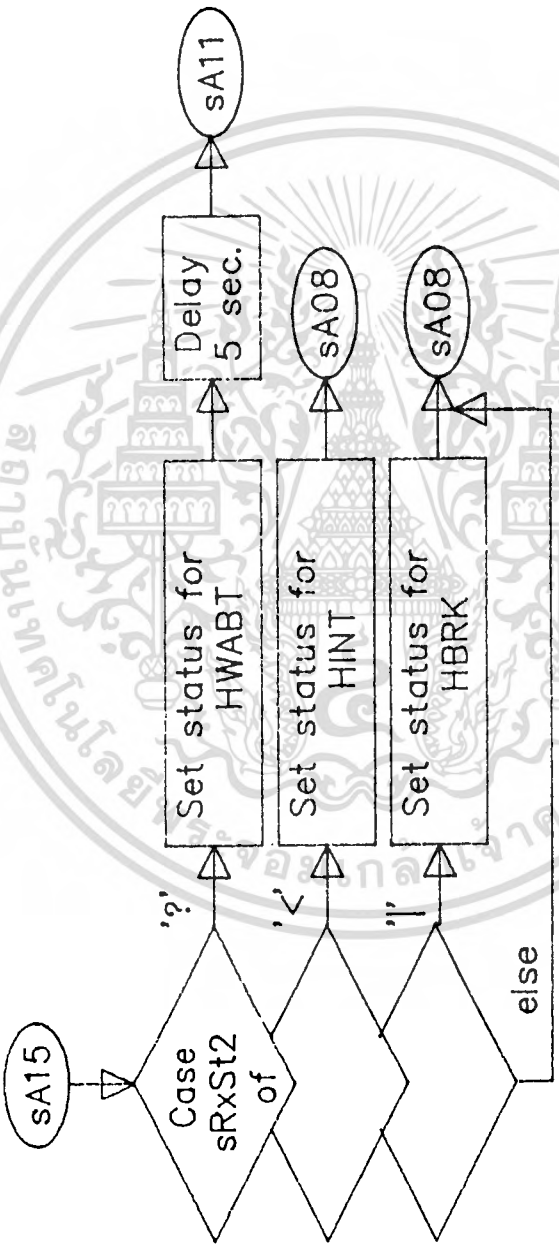
รูป ง.4.1 ทาส์คที่ 3 ควบคุมการรับ-ส่งข้อมูลแบบซิงโครนัส



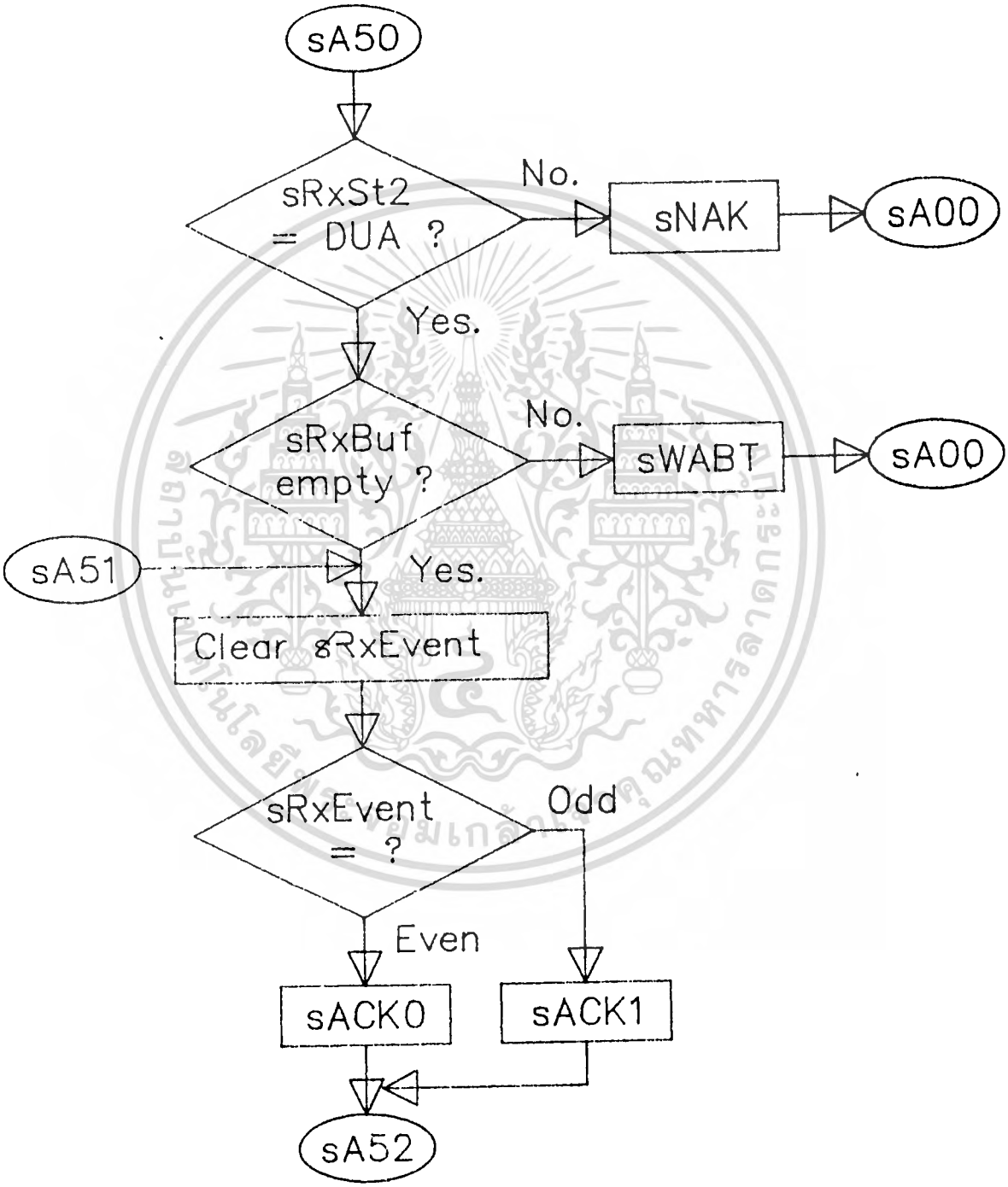
รูป ง.4.2 หาส์คที่ 3 ควบคุมการรับ-ส่งข้อมูลแบบซิงโครนัส (ต่อ)



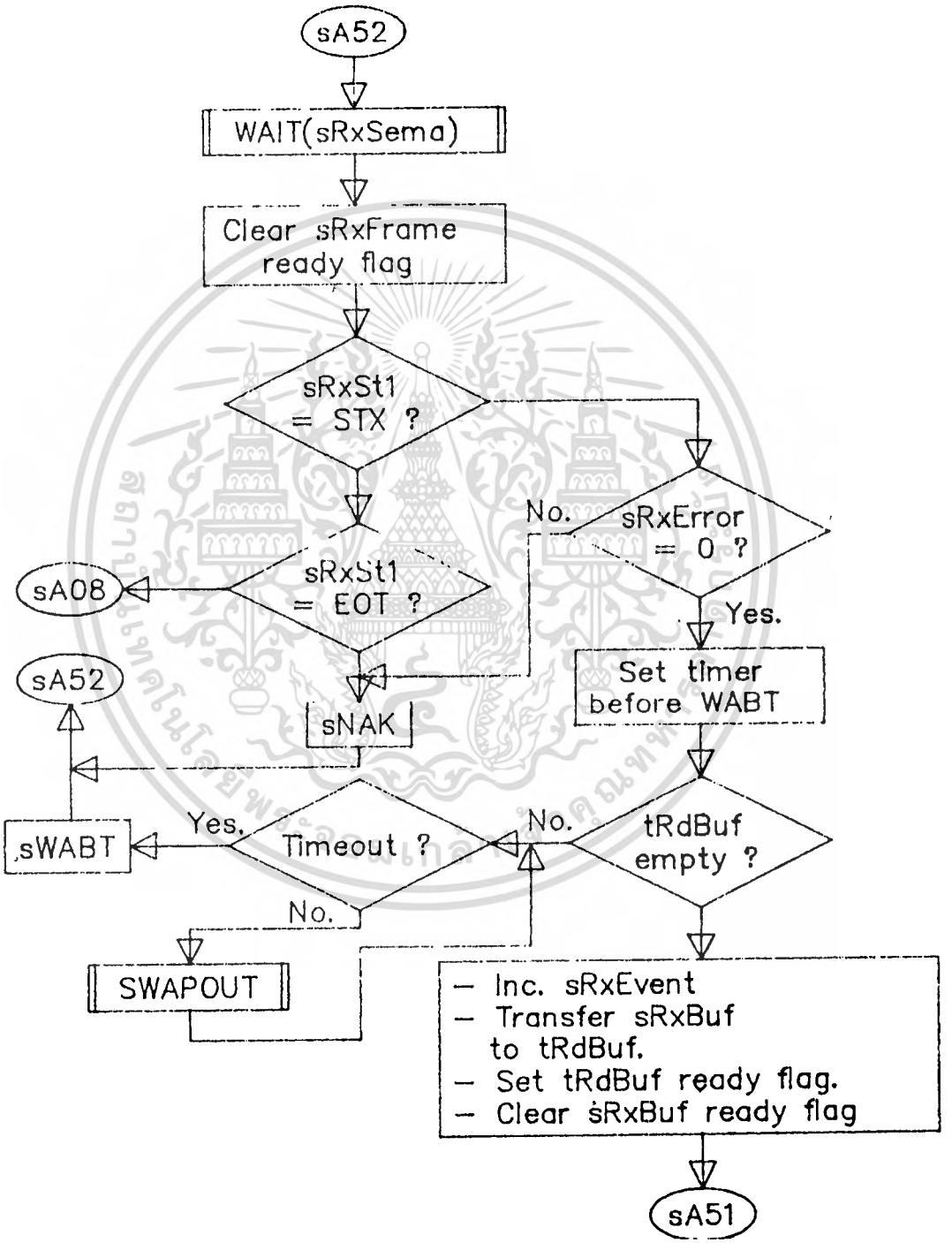
รูป ง.4.3 ทาส์คที่ 3 ควบคุมการรับ-ส่งข้อมูลแบบซิงโครนัส (ต่อ)



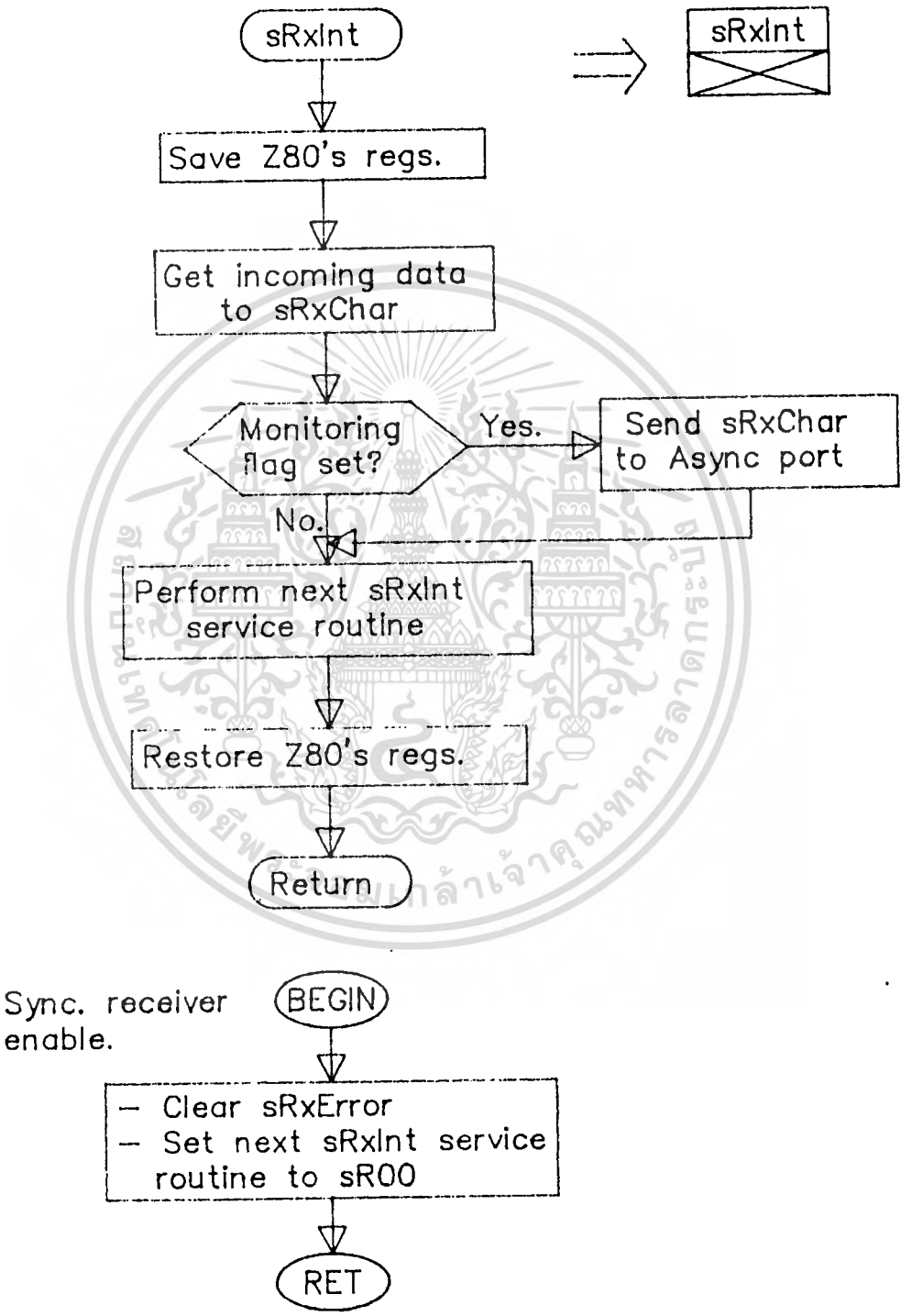
รูป ง.4.4 ทาสก์ที่ 3 ควบคุมการรับ-ส่งข้อมูลแบบอิงจรนัส (ต่อ)



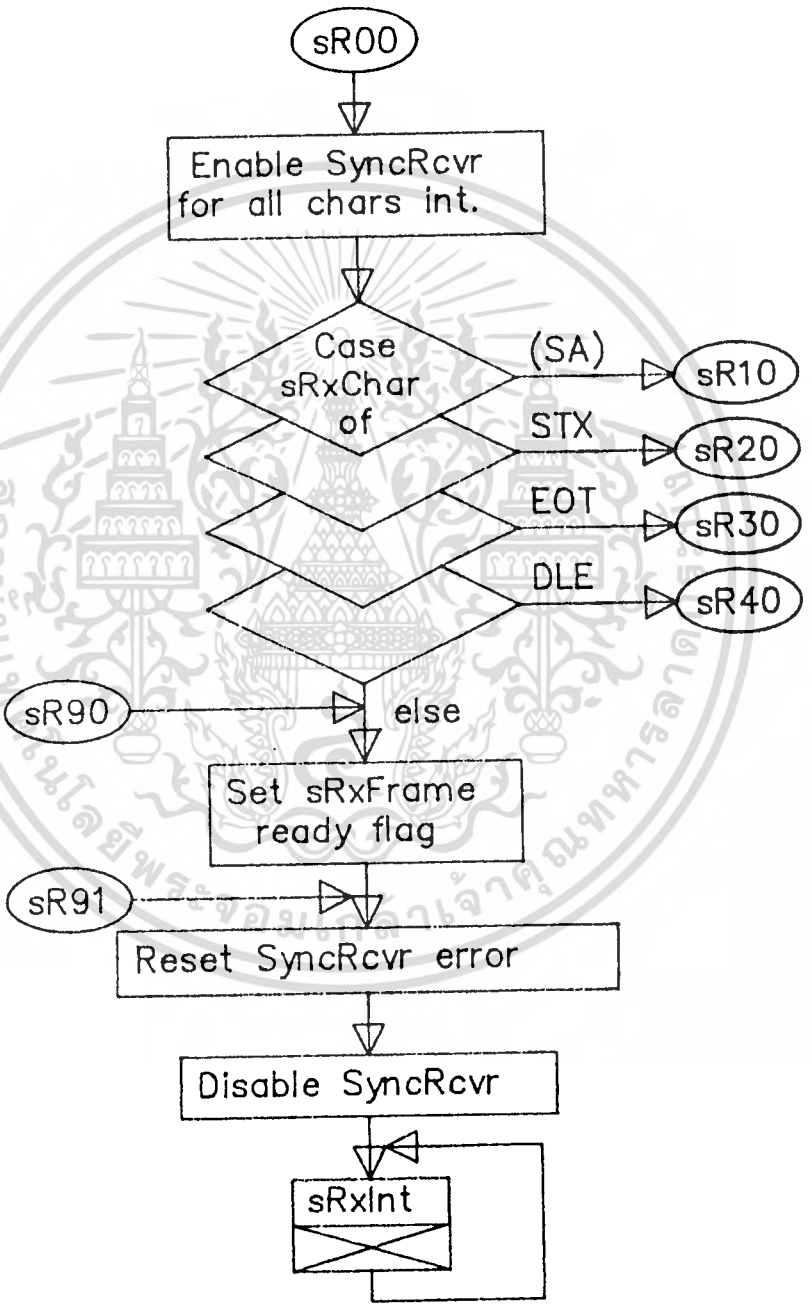
รูป ง.4.5 ทาส์คที่ 3 ควบคุมการรับ-ส่งข้อมูลแบบซิงโครนัส (ต่อ)



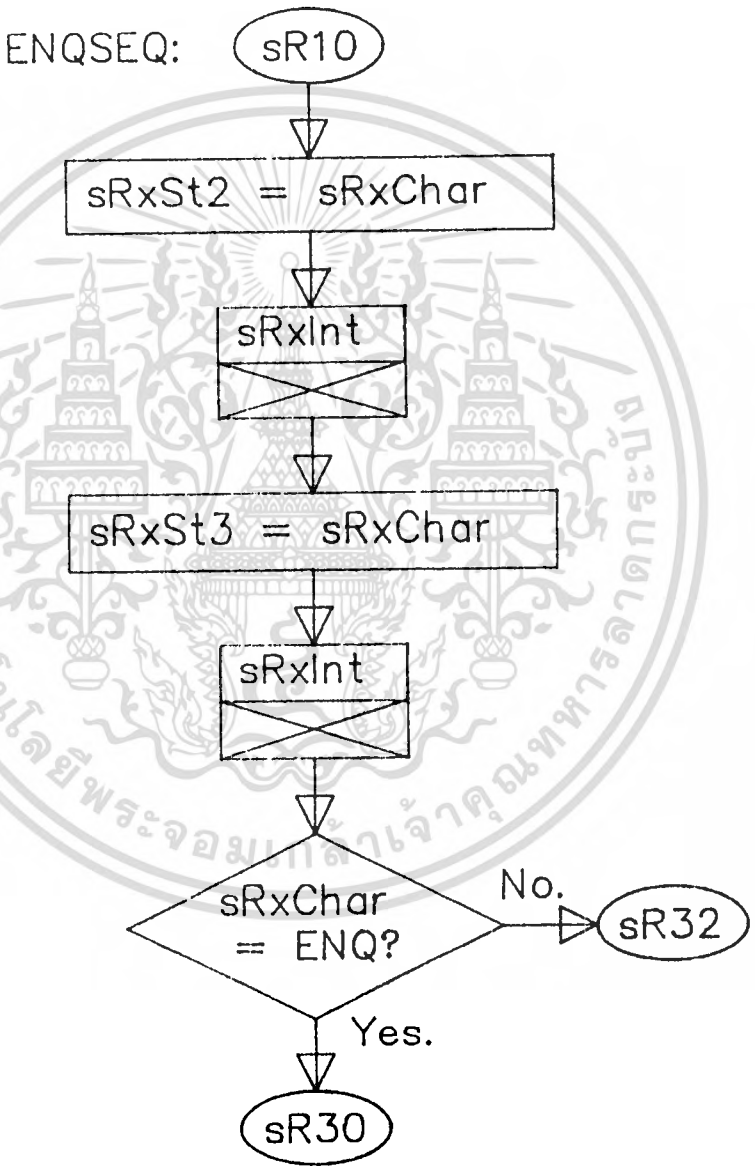
รูป ง.4.6 หาส์คที่ 3 ควบคุมการรับ-ส่งข้อมูลแบบซิงโครนัส (ต่อ)



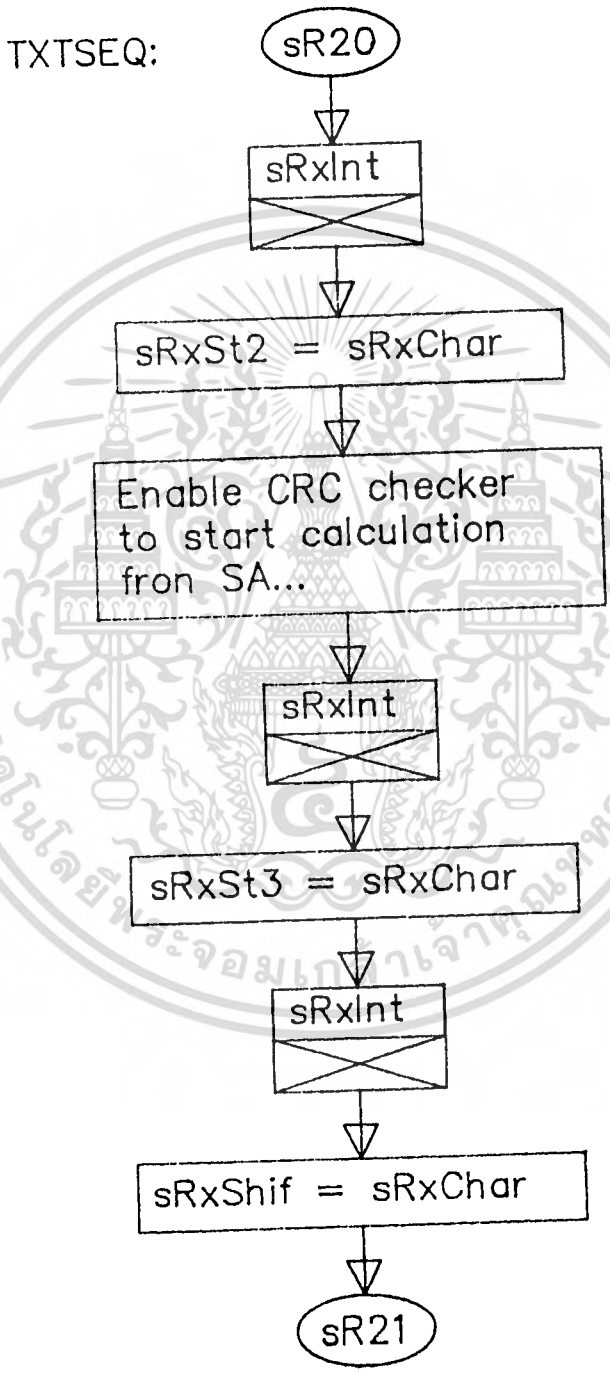
รูป ง.5 โปรแกรมสนับสนุนการอินเทอรัพท์ในการรับแบบซิงโครนัส



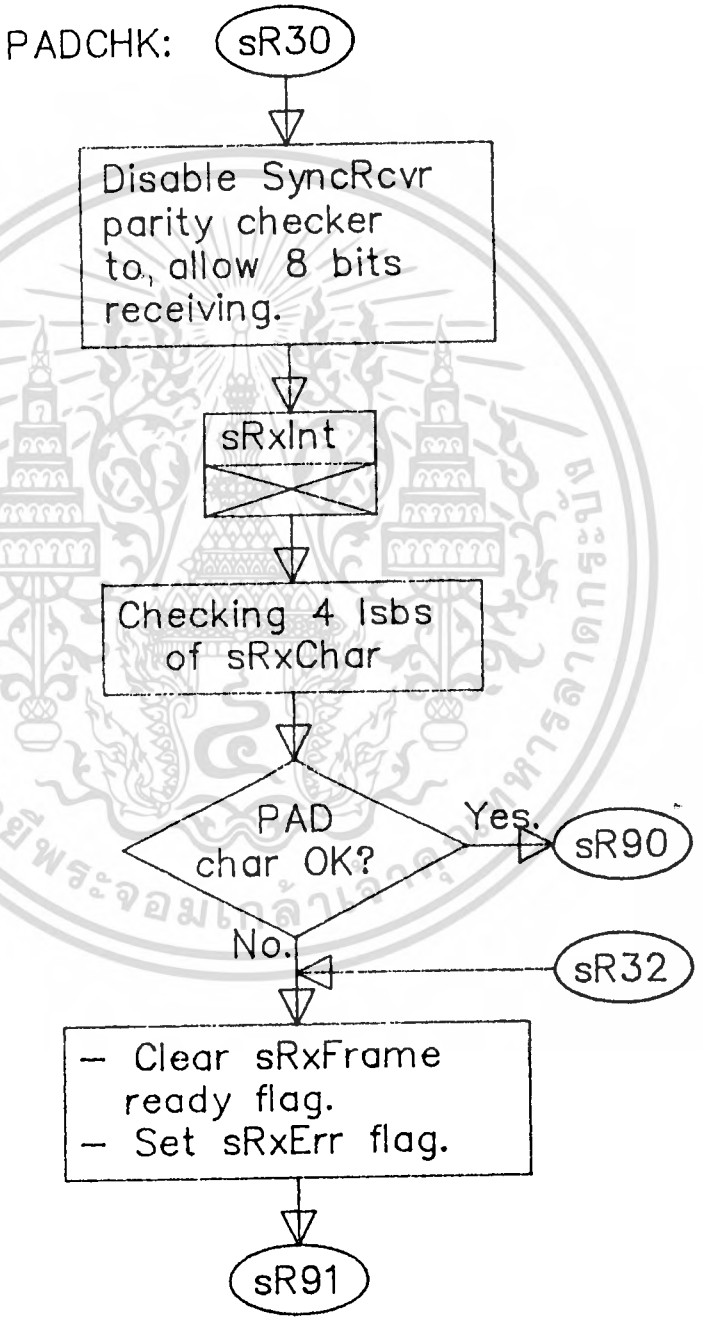
รูป ง.6.1 วิเคราะห์เฟรมข้อมูลแบบซิงโครนัส (แยกประเภท)



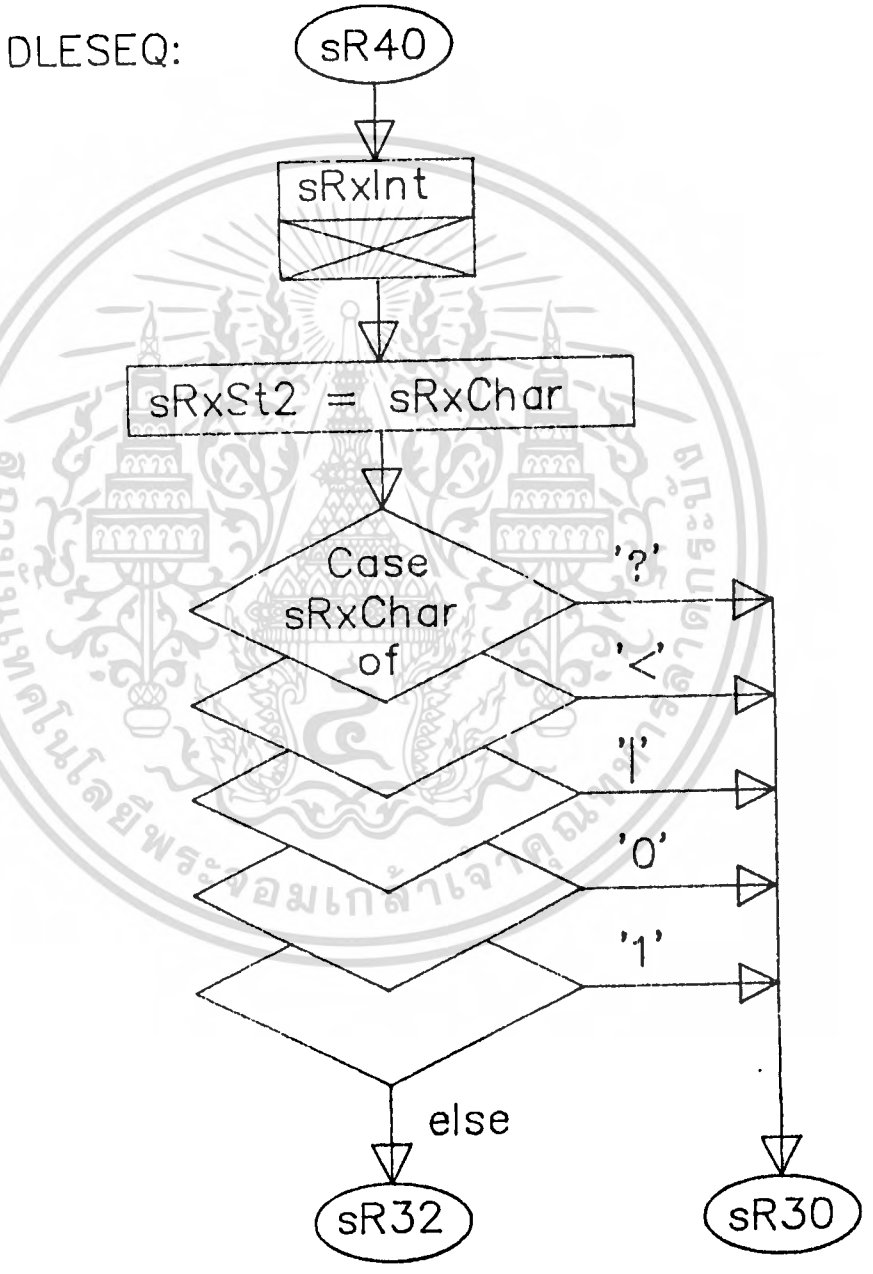
รูป ง.6.2 วิเคราะห์เฟรมข้อมูลแบบซิงโครนัส (Enquire sequence)



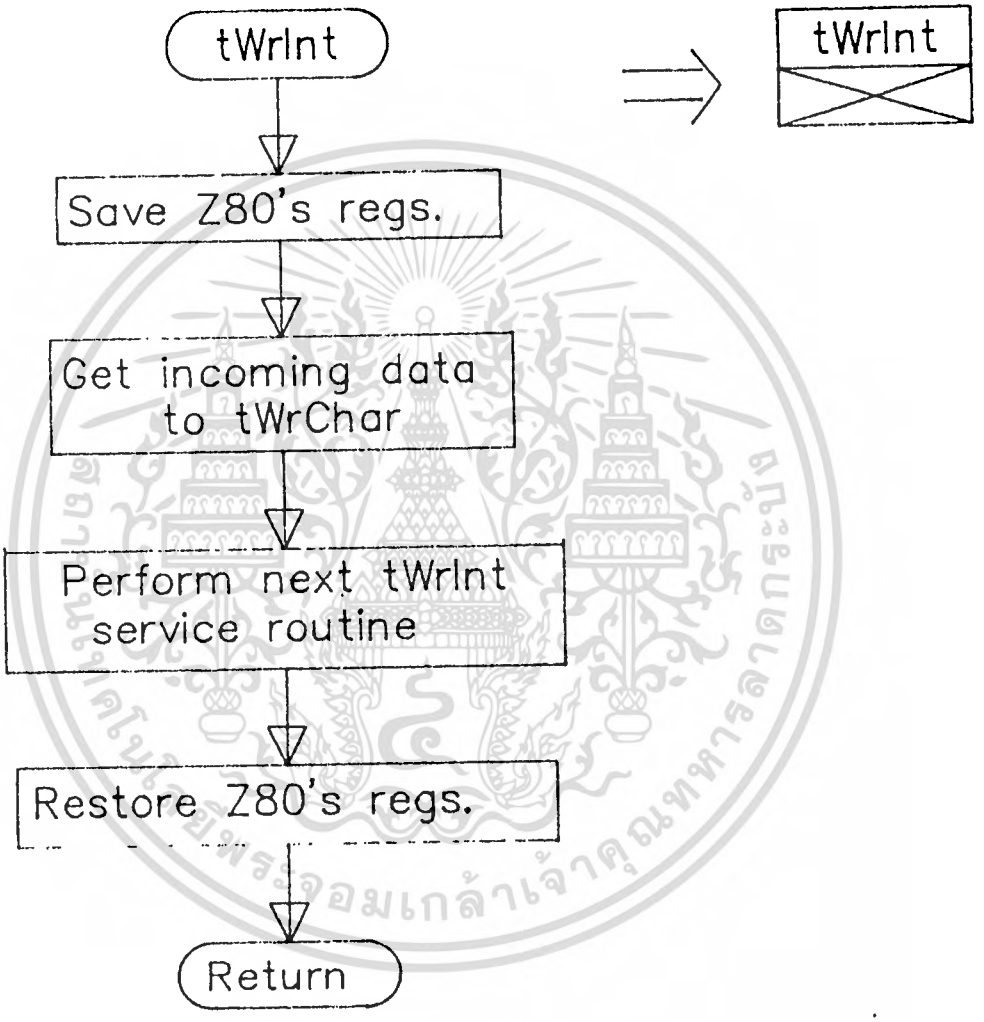
รูป ง.6.3 วิเคราะห์เฟรมข้อมูลแบบซิงโครนัส (Text frame)



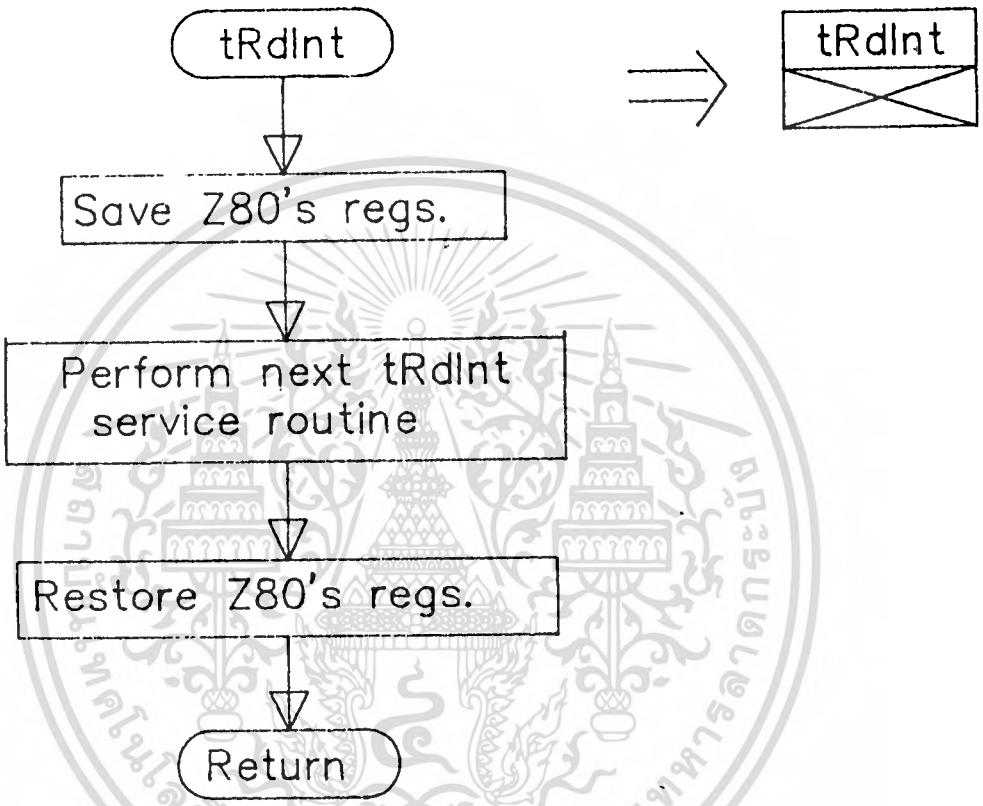
รูป ง.6.4 วิเคราะห์เฟรมข้อมูลแบบซิงโครนัส (ตรวจสอบ PAD)



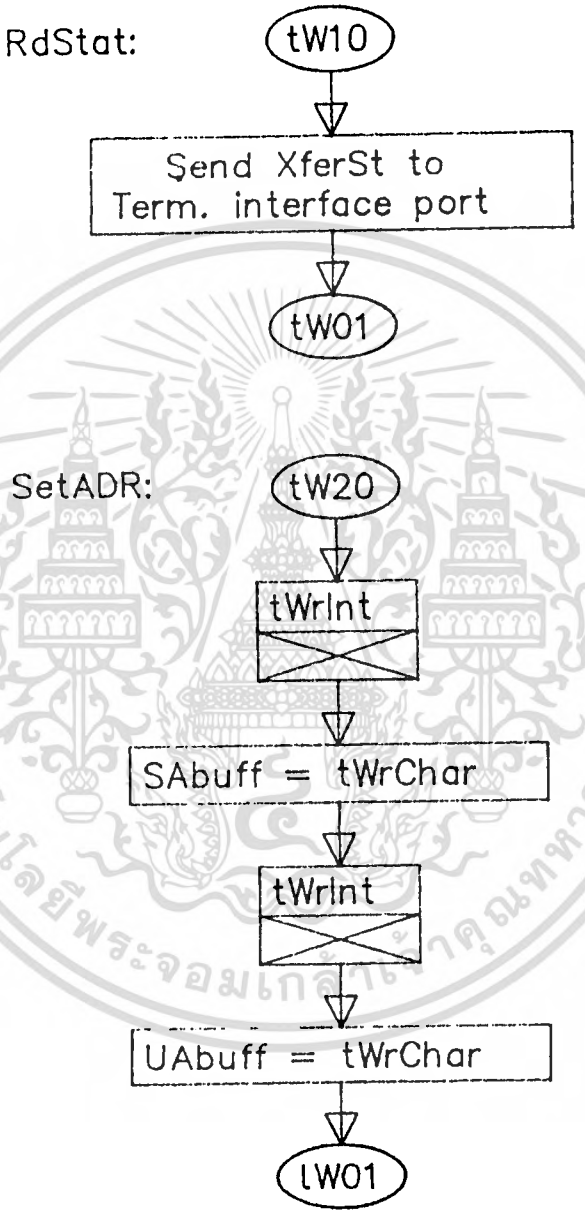
รูป ง.6.5 วิเคราะห์เฟรมข้อมูลแบบซิงโครนัส (ตรวจสอบสัญญาณควบคุม)



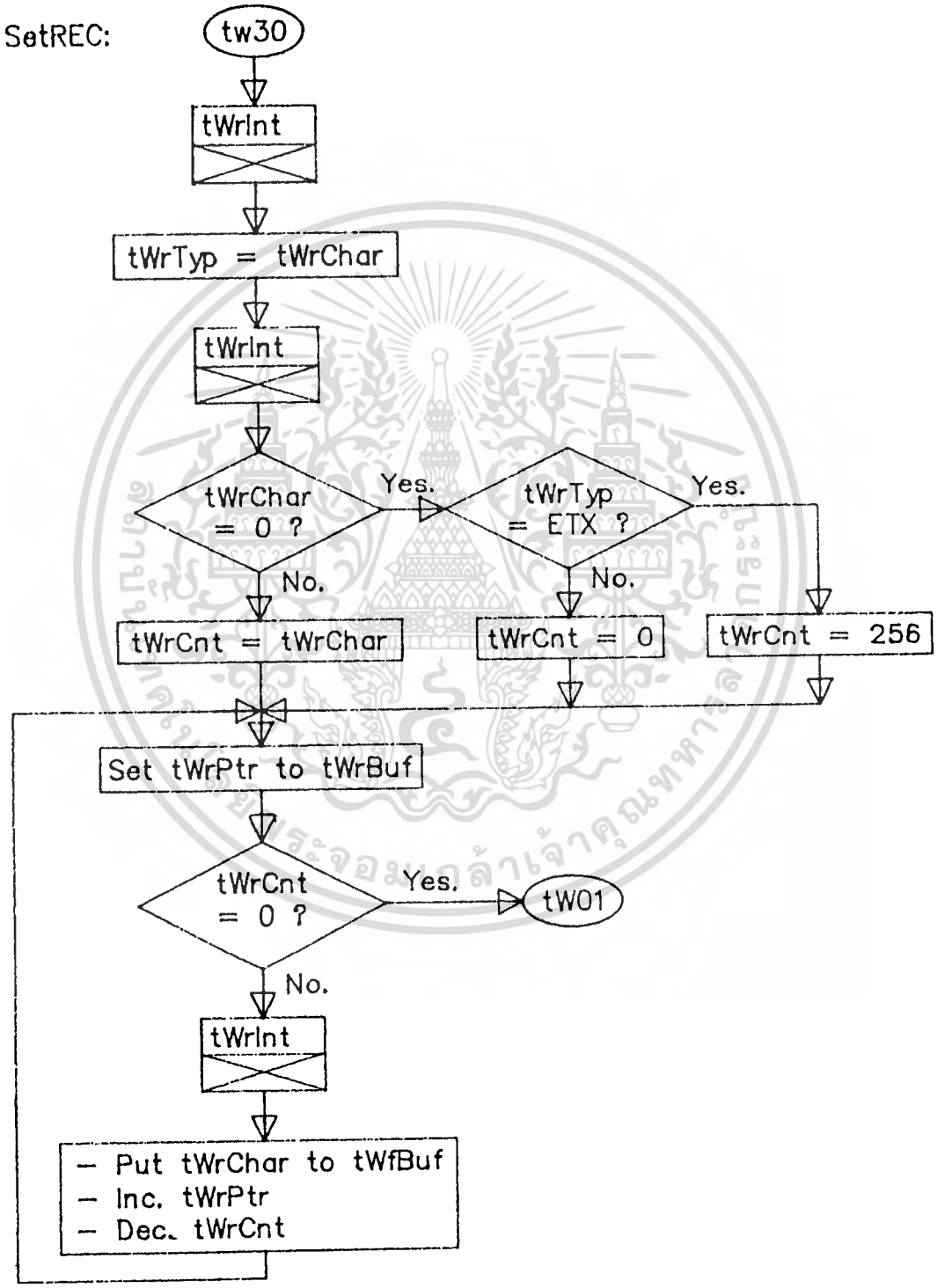
รูป ง.7 โปรแกรมสนับสนุนการอินเทอรัพท์ในการรับข้อมูลจากไมโครคอมพิวเตอร์



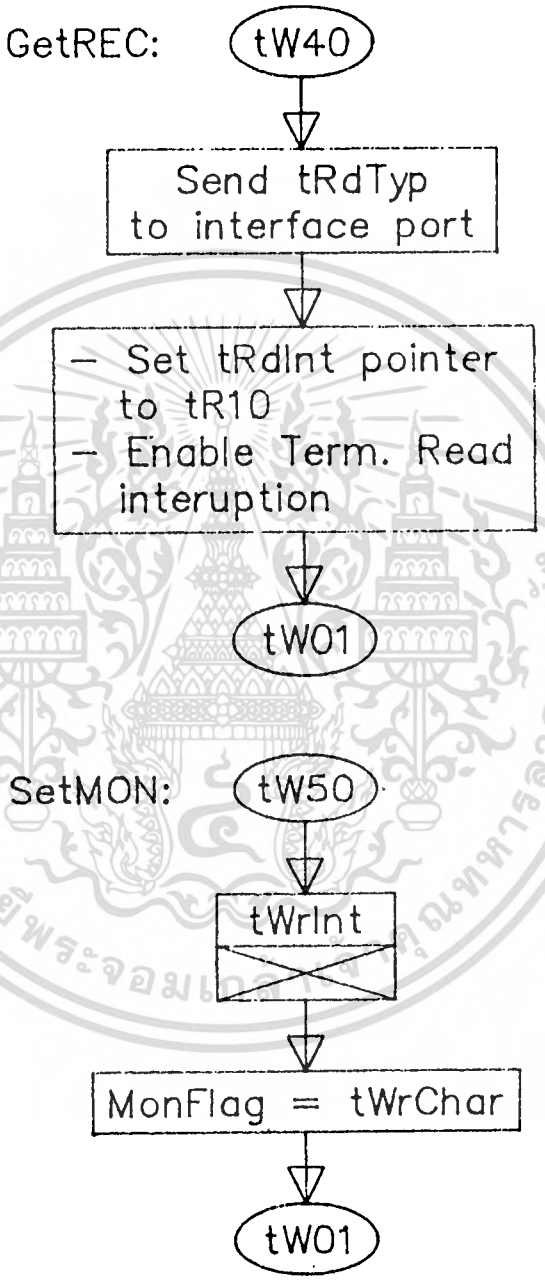
รูป ง.8 โปรแกรมสนับสนุนการอินเทอรัพท์ในการส่งข้อมูลมายังไมโครคอมพิวเตอร์



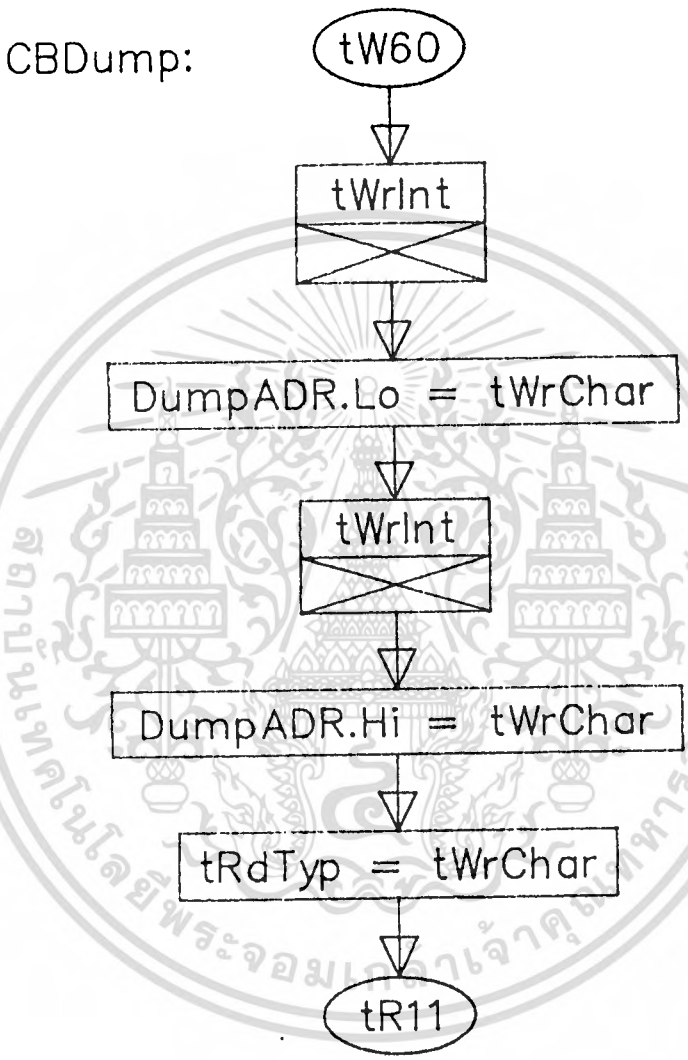
รูป ง.10 การจัดการคำสั่ง RdStatCmd และ SetADRCmd



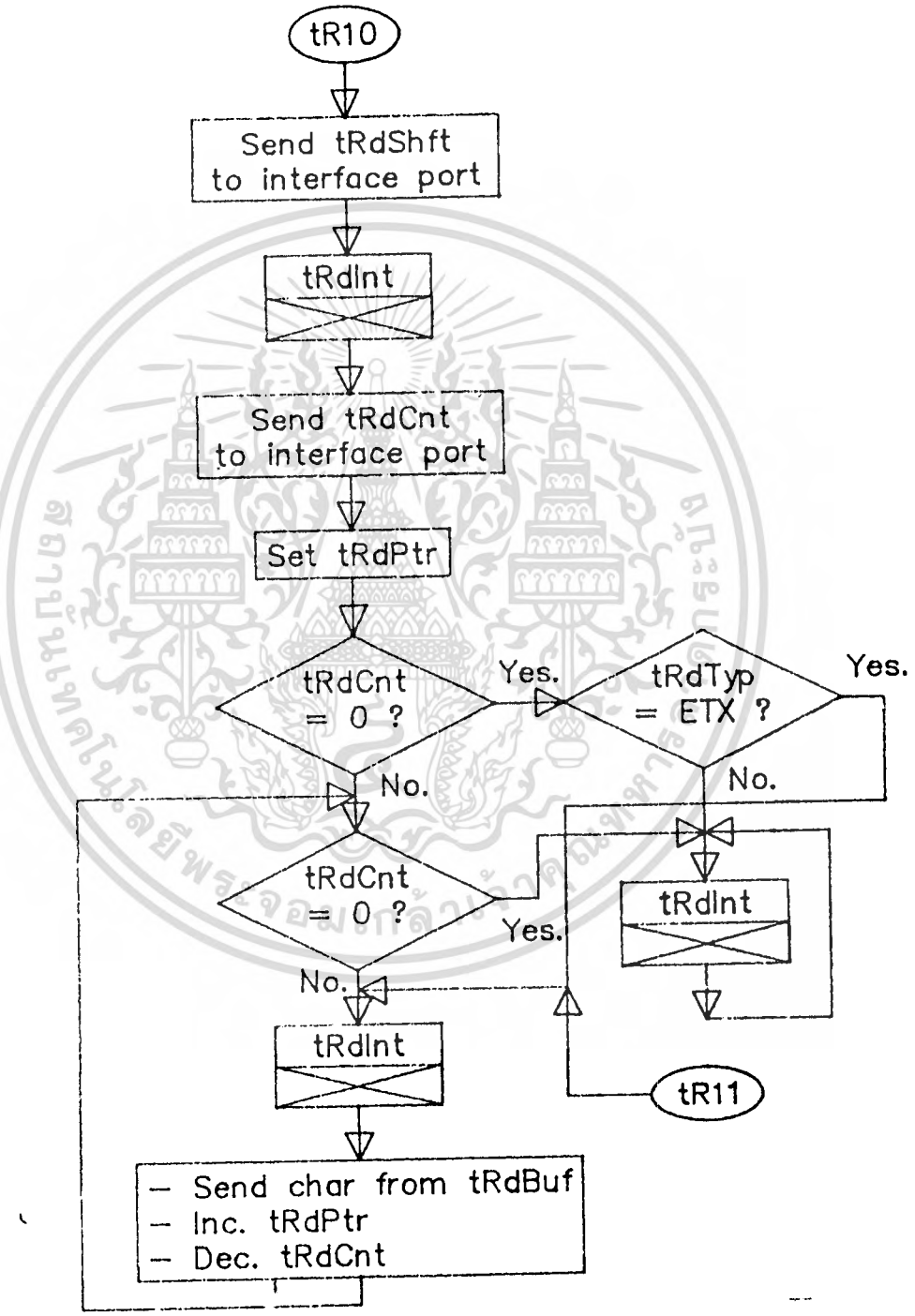
รูป ง.11 การจัดการคำสั่ง SetRecCmd



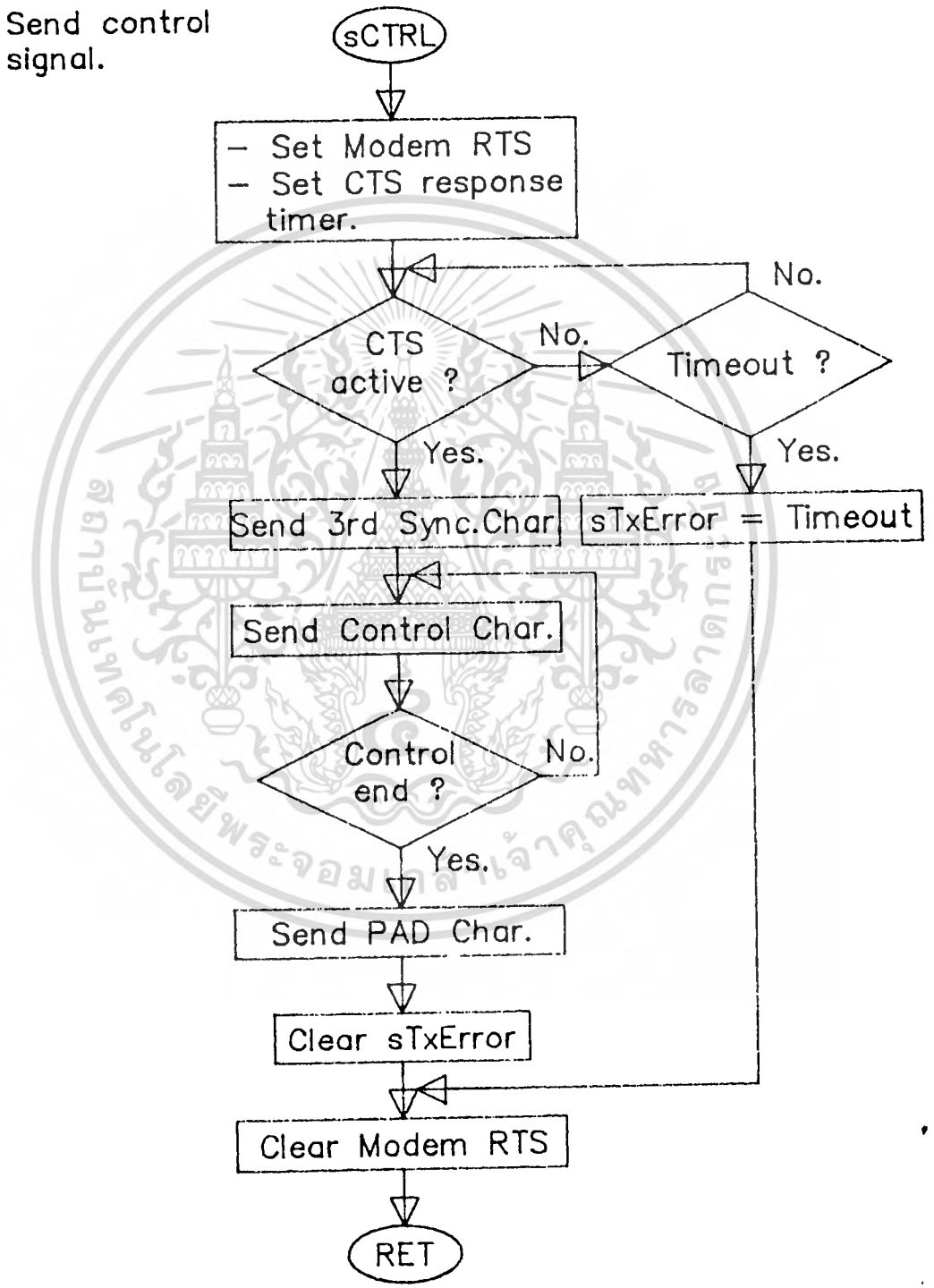
รูป ง.12 การจัดการคำสั่ง GetRECCmd และ SetMONCmd



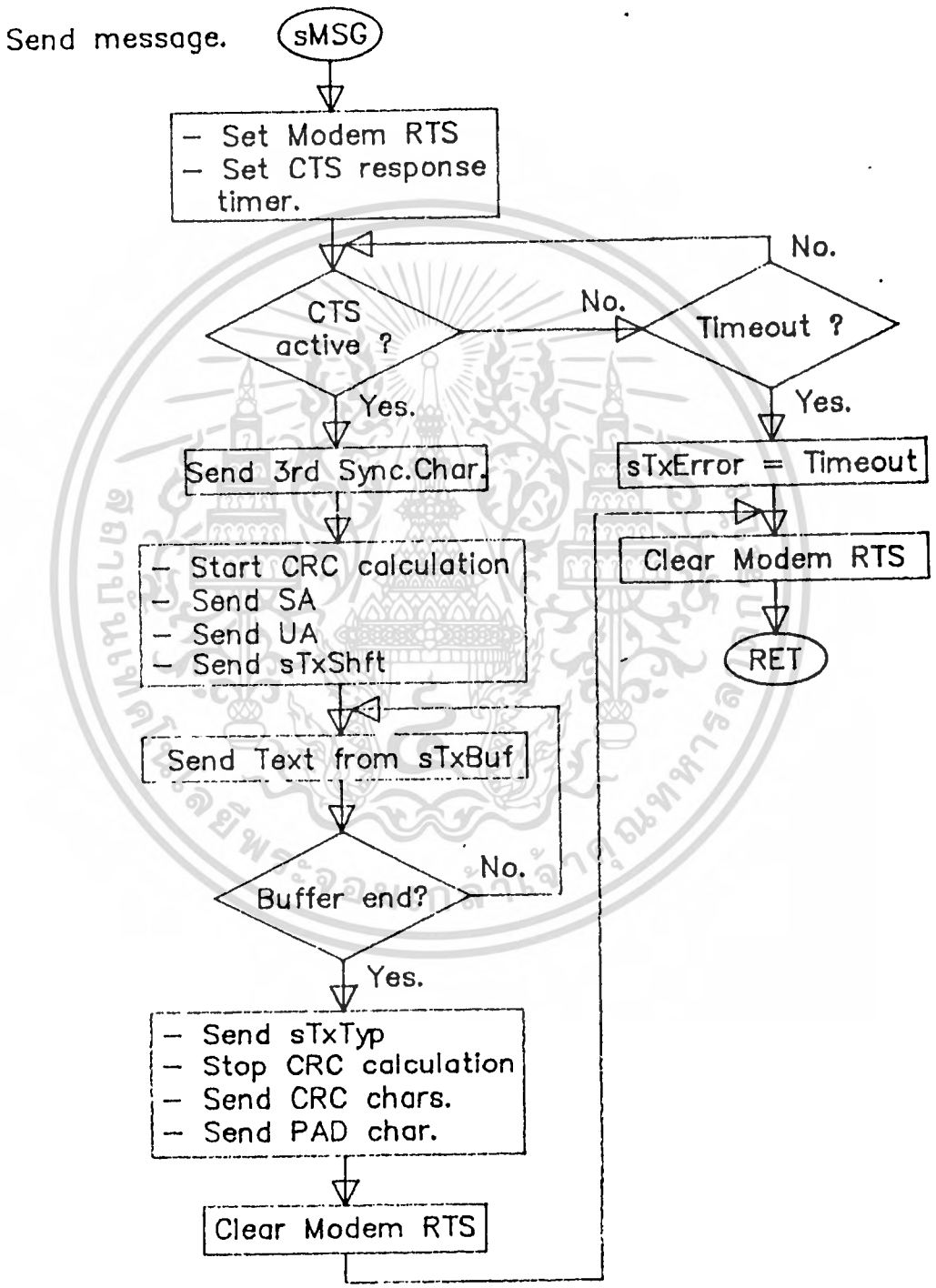
รูป ง.13 การจัดการคำสั่ง CBDumpCmd



รูป ง.14 การส่งผ่านข้อมูลให้กับไมโครคอมพิวเตอร์ภายใต้คำสั่ง GetRECCmd และ CBDumpCmd



รูป ง.15 การส่งเพรมสัญญาณควบคุมานแบบซิงโครนัส



รูป ง.16 การส่งเฟรมข้อมูลในแบบซิงโครนัส

ภาคผนวก จ.

ตารางรหัสมาตราฐานแบบต่างที่ใช้ในระบบการสื่อสารข้อมูล.

CODE CHARTS

EBCDIC Katakana Charcter Code Chart

Bit number 8	7	6	5	4	3	2	1	Column / Row	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
									0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
									0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
									0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
									0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
				0	0	0	0	0	NUL	DLE	DS		SP	&	-			ソ			{	}	\	0
				0	0	0	1	1	SOH	DC1	SOS		・	ｴ	/		ｱ	ﾀ			A	J		1
				0	0	1	0	2	STX	DC2	FS	SYN	「	ｵ			ｲ	ﾁ	ﾍ		B	K	S	2
				0	0	1	1	3	ETX	TM			」	ﾔ			ｳ	ﾂ	ﾎ		C	L	T	3
				0	1	0	0	4	PF	RES	BYP	PN	、	ｭ			ｴ	ﾃ	ﾏ		D	M	U	4
				0	1	0	1	5	HT	NL	LF	RS	・	ﾖ			ｵ	ﾄ	ﾐ		E	N	V	5
				0	1	1	0	6	LC	BS	ETB	UC	ヲ	ﾂ			ｶ	ﾅ	ﾑ		F	O	W	6
				0	1	1	1	7	DEL	IL	ESC	EOT	ｧ				ｷ	ﾆ	ﾒ		G	P	X	7
				1	0	0	0	8	GE	CAN			ｨ	￣			ｸ	ﾇ	ﾓ		H	Q	Y	8
				1	0	0	1	9	RLF	EM			ｳ				ｹ	ﾈ	ﾔ		I	R	Z	9
				1	0	1	0	A	SMM	CC	SM		ｿ	!		:	ｺ	ﾉ	ﾕ	ﾚ	゛			LVM
				1	0	1	1	B	VT	CU1	CU2	CU3	・	¥	'	#	｡			ﾛ				
				1	1	0	0	C	FF	IFS		DC4	<	*	ｽ	@	ｻ		ﾖ	ﾜ				
				1	1	0	1	D	CR	IGS	ENQ	NAK	(	)	_	'	ｼ	ﾊ	ﾗ	ﾝ				
				1	1	1	0	E	SO	IRS	ACK		+	;	>	=	ｽ	ﾋ	ﾘ	"				
				1	1	1	1	F	SI	IUS	BEL	SUB		¬	?	"	ｾ	ﾌ	ﾙ	・				EO

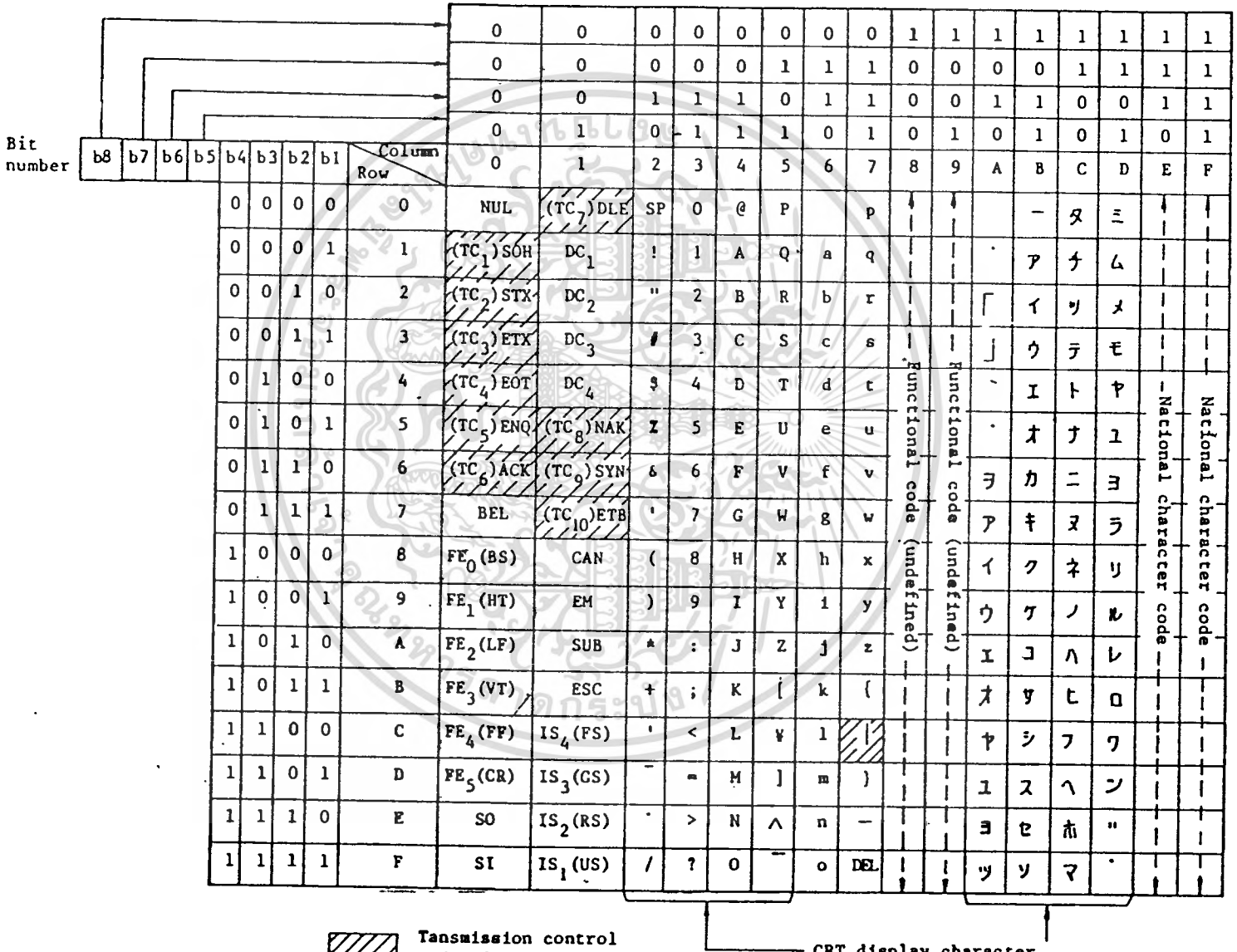
JIS Code Chart (7-bit code)

	SHIFT IN								SHIFT OUT							
b7	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
b6	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
b5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

b8	b7	b6	b5	b4	b3	b2	b1
With even parity bit				0	0	0	0
				0	0	0	1
				0	0	1	0
				0	0	1	1
				0	1	0	0
				0	1	0	1
				0	1	1	0
				0	1	1	1
				1	0	0	0
				1	0	0	1
				1	0	1	0
				1	0	1	1
				1	1	0	0
				1	1	0	1
				1	1	1	0
				1	1	1	1

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	DLE	SPACE	0	@	P				DLE		—	タ	ミ		
1	SOH		!	1	A	Q			SOH		・	ア	チ	ム		
2	STX		"	2	B	R			STX		「	イ	ツ	メ		
3	ETX		#	3	C	S			ETX		」	ウ	テ	モ		
4	EOT		$	4	D	T			EOT		、	エ	ト	ヤ		
5	ENQ	NAK	%	5	E	U			ENQ	NAK	・	オ	ナ	ユ		
6	ACR	SYN	&	6	F	V			ACK	SYN	ヲ	カ	ニ	ヨ		
7	BEL	ETB	'	7	G	W			BEL	ETB	ア	キ	ヌ	ラ		
8	BS	CAN	(	8	H	X			BS	CBN	イ	ク	ネ	リ		
9	HT		)	9	I	Y			HT		ウ	ケ	ノ	ル		
A	NL(IF)		*	:	J	Z			HI(IF)		エ	コ	ハ	レ		
B	VT	ESC	+	;	K	[		{	VT	ESC	オ	サ	ヒ	ロ		
C	FF		,	<	L	¥		I(SIP)	FF		ヤ	シ	フ	ワ		
D			-	=	M	]					ユ	ス	ヘ	ン		
E	SO		.	>	N	^			SO		ヨ	セ	ホ	゛		
F	SI		/	?	O	-		DEL	SI		ツ	ソ	マ	゜		

JIS Code Chart (8-bit code)



Bit number b8	b7	b6	b5	b4	b3	b2	b1	Row \ Column	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
									0	0	0	0	0	1	1	1	0	0	0	0	1	1	1	1
									0	0	1	1	1	0	1	1	0	0	1	1	0	0	1	1
									0	1	0	1	1	1	0	1	0	1	0	1	0	1	0	1
									0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
				0	0	0	0	0	NUL	(TC$_7$)DLE	SP	0	@	P		p				ー	タ	ミ		
				0	0	0	1	1	(TC$_1$)SOH	DC$_1$	!	1	A	Q	a	q			。	ア	チ	ム		
				0	0	1	0	2	(TC$_2$)STX	DC$_2$	"	2	B	R	b	r			「	イ	ツ	メ		
				0	0	1	1	3	(TC$_3$)ETX	DC$_3$	#	3	C	S	c	s			」	ウ	テ	モ		
				0	1	0	0	4	(TC$_4$)EOT	DC$_4$	$	4	D	T	d	t			、	エ	ト	ヤ		
				0	1	0	1	5	(TC$_5$)ENQ	(TC$_8$)NAK	%	5	E	U	e	u			・	オ	ナ	ユ		
				0	1	1	0	6	(TC$_6$)ACK	(TC$_9$)SYN	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
				0	1	1	1	7	BEL	(TC$_{10}$)ETB	'	7	G	W	g	w			ァ	キ	ヌ	ラ		
				1	0	0	0	8	FE$_0$(BS)	CAN	(	8	H	X	h	x			ィ	ク	ネ	リ		
				1	0	0	1	9	FE$_1$(HT)	EM	)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
				1	0	1	0	A	FE$_2$(LF)	SUB	*	:	J	Z	j	z			ェ	コ	ハ	レ		
				1	0	1	1	B	FE$_3$(VT)	ESC	+	;	K	[	k	{			ォ	サ	ヒ	ロ		
				1	1	0	0	C	FE$_4$(FF)	IS$_4$(FS)	,	<	L	¥	l	|			ャ	シ	フ	ワ		
				1	1	0	1	D	FE$_5$(CR)	IS$_3$(GS)	-	=	M	]	m	}			ュ	ス	ヘ	ン		
				1	1	1	0	E	SO	IS$_2$(RS)	.	>	N	^	n	~			ョ	セ	ホ	゛		
				1	1	1	1	F	SI	IS$_1$(US)	/	?	O	_	o	DEL			ッ	ソ	マ	゜		

Functional code (undefined) — Functional code (undefined) — National character code — National character code

Tansmission control code N6300

CRT display character N6300

ASCII Character Code Chart

Bit number 8	7	6	5	4	3	2	1	Column / Row	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
				0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p								
				0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q								
				0	0	1	0	2	STX	DC2	"	2	B	R	b	r								
				0	0	1	1	3	ETX	DC3	#	3	C	S	c	s								
				0	1	0	0	4	EOT	DC4	¥	4	D	T	d	t								
				0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u								
				0	1	1	0	6	ACK	SYN	&	6	F	V	f	v								
				0	1	1	1	7	BEL	ETB	'	7	G	W	g	w								
				1	0	0	0	8	BS	CAN	(	8	H	X	h	x								
				1	0	0	1	9	HT	EM	)	9	I	Y	i	y								
				1	0	1	0	A	LF	SUB	*	:	J	Z	j	z								
				1	0	1	1	B	VT	ESC	+	;	K	[	k	{								
				1	1	0	0	C	FF	FS	,	<	L	\	l	|							
				1	1	0	1	D	CR	GS	-	=	M	]	m	}								
				1	1	1	0	E	SO	RS	.	>	N	^	n	~								
				1	1	1	1	F	SI	US	/	?	O	_	o	DEL								EO