หลักการเชียนโปรแกรมไฟล์ทรานเฟอร์

FILE TRANSFER PROTOCOL

โดย

นาย เจษฎา ไชยจารุวณิช

นาย ศักดิ์ชัย ตรีทวีพรทรัพย์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตร์บัณฑิต

สาขา เทคโนโลยีโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2535

ปริญญานิพนธ์การศึกษา 2535

สาขาวิชา เทคโนโลยีโทรคมนาคม

ภาควิชา เทคนิคอุตสาหกรรม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง

เรื่อง หลักการเขียนโปรแกรมไฟล์ทรานเฟอร์

ผู้จัดทำ

1. นาย เจษฎา ไชยจารุวัฒิ 34132102
2. นาย คักดิ์ชัย ตรีทวีพรทรัพย์ 34132126

..... อาจารย์ที่ปรึกษา

(อ.กฤดากร กล่อมการ)

หลักการเขียนโปรแกรมไฟล์ทรานเฟอร์

File Transfer Protocol

นาย เจษฎา ไชยจารุวณิช 34132102

นาย ศักดิ์ชัย ตรีทวีพรทรัพย์ 34132126

อาจารย์ที่ปรึกษา

อ.กฤตากร กล่อมการ

ปีการศึกษา 2535

บทคัดย่อ

บริญญานิพนธ์นี้ มีวัตถุประสงค์เพื่อพัฒนา Software การสื่อสารข้อมูล ระหว่าง Terminal ผ่านทาง Port สื่อสารอนุกรม ตามมาตรฐาน RS-232 ของ Computer โดยใช้ภาษา C เขียน สำหรับ Software ที่ได้จัดทำขึ้นในโครงงานนี้ ได้แก่ โปรแกรมการส่ง file ข้อมูล ตามมาตรฐาน X-MODEM ,โปรแกรมตรวจสอบความผิดพลาดของการส่งข้อมูลแบบ CRC (Cyclic Redundancy Check),โปรแกรมเข้ารหัสลับข้อมูล (Data Encryption) เพื่อป้องกันการลักลอบขโมยข้อมูล ในระหว่างการรับส่ง, และโปรแกรมลดขนาดข้อมูล (Data Compression) โดยใช้เทคนิคของ HUFF MAN และ LZW เพื่อทำให้ file ที่จะส่งมีขนาดลดลง รวมทั้งโปรแกรมสนับสนุนต่างๆ เพื่อเพิ่มความสามารถในการใช้งาน

FILE TRANSFER PROTOCOL

Mr. JADESADA CHAIJARUWANICH

Mr. SAKCHAI TREETAWEEPORNSUB

Mr. KITDAKORN KLOMKARN Advisor

1992

Abstract

Main even of this paper is to develope the software of data communication between one terminal and other. This has used serial communication port which is standard of RS-232 of computer using C language. The softwares Which have created for this project such CRC oriented data encoding of data (Data Encryption) Which can protect data from interferrence while sending or receiving, and compression of data using The technique of HUFF MAN and LZW. Result of these process makes data and support programmes decrese and these also have increased optimum in performance of general data communication.

สารบัญ

		หน้า
บทที่ 1	บทนำ	1-1
บทที่ 2	การรับส่งไฟล์ข้อมูล	2-1 ถึง 2-27
บทที่ 3	การลดขนาดข้อมูล	3-1 ถึง 3-25
บทที่ 4	การเข้ารหัสลับข้อมูล	4-1 ถึง 4-16
บทที่ 5	การตรวจสอบความผิดพลาดในการรับส่งข้อมูล	5-1 ถึง 5-18
ภาคผนวก		
หนังสืออ้างอิง		

บทที่ 1

บทนำ

การสื่อสารทางด้านข้อมูล (Data communication) ระหว่างคอมพิวเตอร์ ปัจจุบันเราต้องใช้ software จากต่างประเทศ เช่น Procom, Lablink แม้นว่า software เหล่านี้จะมีใช้งานอยู่แล้วก็ตาม แต่มันก็มีขีดจำกัดในการใช้งาน หรือใช้งานได้เฉพาะอย่าง ทำให้ความต้องการที่จะใช้งานบางอย่างไม่สามารถทำได้ โครงงานนี้จึงจัดทำขึ้นโดยมีวัตถุประสงค์ที่จะพัฒนา โปรแกรมการสื่อสารข้อมูลที่สามารถกำหนดขอบเขตและเงื่อนไขของการสื่อสารเองได้ เพื่อให้เกิดประสิทธิ์ภาพในการรับส่งข้อมูล จึงได้เพิ่มเติม Algorithm ต่างๆ รวบรวมเป็นโปรแกรมรับส่งข้อมูล ซึ่งประกอบด้วยโปรแกรมต่างๆ ดังนี้ 1.โปรแกรมรับส่งแฟ้มข้อมูล (Transfer file) 2.โปรแกรมลดขนาดข้อมูล โดยมี Algorithm ของ Huffman และ Lempel Ziv Welch (LZW) ให้เลือกใช้ 3.โปรแกรมเข้ารหัสลับข้อมูล (Data Encryption) นอกจากนี้ยังมีโปรแกรมตรวจสอบ และแก้ไขข้อผิดพลาดในการส่งข้อมูล โดยใช้ Algorithm ของ Cyclic Redundancy Check (CRC)

ใน Project 2 นี้ได้เพิ่มโปรแกรมสนับสนุนเข้ามาร่วมด้วย เช่น โปรแกรมค้นหาชื่อแฟ้มข้อมูล (DIR), โปรแกรมแสดงแฟ้มข้อมูล (LISTER), โปรแกรมเปลี่ยนชื่อแฟ้มข้อมูล (RENAME), โปรแกรมรับส่งอักขระระหว่าง Terminal (CHAT MODE) เพื่อให้ผู้ใช้งานโปรแกรมนี้ได้ง่ายและสะดวกจึงจัดเป็น MENU ให้ผู้ใช้ เลือกใช้ Algorithm และ SET ค่าต่างๆ ได้ตามต้องการ

อย่างไรก็ตามโปรแกรมรับส่งข้อมูลนี้ ยังไม่สมบูรณ์เท่าที่ควร ยังสามารถพัฒนาโปรแกรมต่อไปได้อีก โดยการเพิ่มโปรแกรมสนับสนุน เพื่อให้ใช้งานได้สะดวกยิ่งขึ้น และที่สำคัญ เพื่อให้สามารถรองรับกับระบบการสื่อสารข้อมูลในรูปแบบต่างๆ เช่น การรับส่งข้อมูลผ่านทาง MODEM หรือ อุปกรณ์สื่อสารชนิดอื่นๆ

ผู้จัดทำหวังว่าปริญญานิพนธ์ฉบับนี้ จะเป็นประโยชน์กับผู้อ่านไม่มากก็น้อย หากมีที่ใดผิดพลาดผู้จัดทำก็ขออภัยมา ณ ที่นี้ด้วย

บทที่ 2

การรับส่งไฟล์ข้อมูล

การสื่อสารภายในระบบคอมพิวเตอร์

การสื่อสารระหว่างคีย์บอร์ดกับเครื่อง อาจเป็นได้ทั้งแบบอนุกรม หรือแบบขนาน สำหรับไมโครคอมพิวเตอร์ IBM PC ที่เป็นพิมพ์มีวงจรควบคุมคีย์บอร์ด คอยตรวจสอบการกดแป้นพิมพ์ เมื่อมีการกดคีย์ ส่วนวงจรควบคุมจะส่งสัญญาณต่อไปยังเมนซีพียู แต่ส่งไปแบบอนุกรมแล้วก็ไม่ได้เข้ารหัส ASCII ส่งไปแต่เป็นรหัสประจำแป้นพิมพ์เท่านั้น ว่าคีย์ไหนถูกกดไว้ ให้เมนซีพียูไปแปลงเอาเอง มาถึงปัญหาว่าเมนซีพียูจะรู้ได้อย่างไรว่าแป้นพิมพ์ถูกกด และข้อมูลที่ส่งมาก็เป็นพัลส์ ๆ ต่อเนื่องกันแบบอนุกรมซีพียูรู้ได้อย่างไรว่าตรงไหนควรจะเป็น "1" หรือควรจะเป็น "0" ในสัญญาณที่ส่งมาจากคีย์บอร์ดของไอบีเอ็ม จะมีสัญญาณอินเตอร์รัพต์มาด้วยเส้นหนึ่ง เป็นการชี้จังหวะการทำงานของซีพียู สัญญาณนี้จะทำให้ซีพียูกระโดดไปทำงานในรูทีนอันหนึ่งใน BIOS (โปรแกรมจัดการ อินพุต-เอาต์พุต พื้นฐานที่มากับเครื่องอยู่ในรูปของ ROM หรือ PROM) ซึ่งทำหน้าที่รับข้อมูลจากคีย์บอร์ด โดยเฉพาะรับเสร็จก็เอาไปเก็บไว้ก่อนยังไม่เอาไปใช้งานทันทีรอให้โปรแกรมที่ทำงานจริงๆ มาเรียกไปใช้งานอีกที (โดยที่ผู้ใช้เครื่องไม่รู้ตัว) คราวนี้เมนซีพียูจะรู้ได้อย่างไรว่าคีย์บอร์ดส่งอะไรมา ในเมื่อสัญญาณที่ส่งมามีแต่รูปพัลส์ที่สูง 5 กับ 0 โวลต์ ต้องหาทางแยกให้ได้ว่าตรงไหนควรจะเป็นบิตที่ 1 หรือที่ 2 หรือ 3.. และ จำเป็นจะต้องมีการซิงโครไนซ์หรือบอกให้รู้แน่นอนว่าตรงไหนคือข้อมูล คีย์บอร์ดก็จะมีสัญญาณอีกตัวหนึ่งที่เรียกว่าสัญญาณนาฬิกา สัญญาณนาฬิกาจะบอกให้ตัวเปลี่ยนข้อมูลจากอนุกรมเป็นแบบขนานบนซีพียูบอร์ด ได้รู้ว่าตรงนั้นคือ บิตที่เท่าไรของข้อมูล ถ้าเลือกบิตเอาตามใจแล้วจะยุ่งแน่ ผู้ส่งกับผู้รับก็ตีความหมายไม่เหมือนกัน

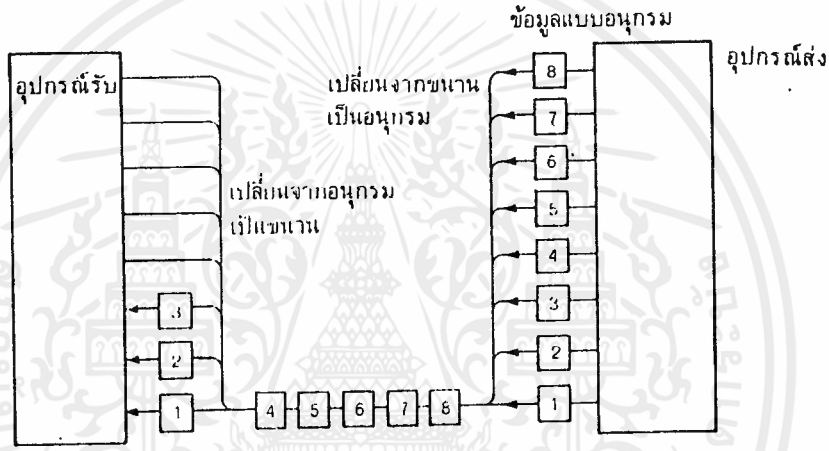
เมนซีพียูจะติดต่อกับเครื่องพิมพ์ทางช่องสื่อสาร อาจจะเป็นแบบอนุกรมหรือแบบขนาน ส่วนมากจะเป็นแบบขนาน ที่เตกต่างออกไปคือ ความเร็วในการพิมพ์ของเครื่องพิมพ์ ยังน้อยกว่าความเร็วของซีพียูมาก ฉะนั้นจำเป็นต้องมีการโต้ตอบระหว่างเครื่องพิมพ์ แบบ Hand-shake

การสื่อสารทั้งสองแบบ คือ แบบอนุกรม และแบบขนาน นอกเหนือจากการติดต่อภายในของเครื่องไมโครคอมพิวเตอร์แล้ว บางครั้งจำเป็นจะต้องติดต่อกับคอมพิวเตอร์ตัวอื่น เพื่อเพิ่มความสามารถของมันเองให้สูงขึ้น เช่นไมโครคอมพิวเตอร์หลายๆ อาจสื่อสารเข้าหากันเป็นเน็ตเวอร์ค เพื่อประโยชน์ในการใช้อุปกรณ์ที่จำเป็นต่างๆ ร่วมกัน และที่สำคัญก็คือเอาไว้ใช้ข้อมูลที่เก็บไว้ในที่ต่างๆ ร่วมกันได้

คอมพิวเตอร์โดยเฉพาะไมโครคอมพิวเตอร์ มีข้อจำกัดของมันเองหลายประการ เช่น ขนาดของความจำ จำกัด ความเร็วในการทำงานจำกัด และอื่นๆ การติดต่อสื่อสารกับภายนอก โดยเฉพาะคอมพิวเตอร์เครื่องใหญ่ที่มีขนาดความจำมาก ๆ ก็เป็นการเพิ่มความสามารถให้สูงขึ้น

การโอนข้อมูลแบบอนุกรม

การถ่ายโอนข้อมูลแบบอนุกรม ข้อมูลถูกส่งออกมาทีละบิต ระหว่างจุดส่งและจุดรับ การส่งข้อมูลแบบนี้จะช้ากว่าแบบขนาน การที่ต้องส่งแบบนี้เพราะตัวกลางการสื่อสาร ต้องการเพียงช่องเดียวหรือสายเพียงคู่เดียว ค่าใช้จ่ายในสื่อกลางจะถูกกว่าแบบขนาน สำหรับการส่งระยะทางไกลๆ โดยเฉพาะเมื่อเรามีระบบการสื่อสารทางโทรศัพท์ไว้ใช้งานอยู่แล้ว ย่อมจะเป็นการประหยัดกว่าที่จะทำการติดต่อสื่อสารทีละ 8 ช่อง เพื่อการโอนข้อมูลแบบขนานอย่างแน่นอน



รูปที่ 1 การส่งข้อมูลแบบอนุกรม

รูป 1 แสดงการส่งข้อมูลแบบอนุกรม ข้อมูลจากจุดส่งจะถูกเปลี่ยนให้เป็นอนุกรมก่อนแล้วค่อยทยอยส่งออกทีละบิตไปยังจุดรับ ณ ที่จุดรับจะต้องมีกลไกในการเปลี่ยนข้อมูลที่ส่งมาทีละบิต ให้เป็นสัญญาณแบบขนานซึ่งลงตัวพอดีนั่นคือ บิต 1 ลงที่บัสข้อมูลเส้นที่ 1 พอดี การที่จะทำให้การแปลงสัญญาณจากอนุกรมทีละบิตให้ลงพอดีนั้นจำเป็นจะต้องมีกลไกที่เหมาะสม เพื่อป้องกันการผิดพลาดในการรับ กลไกที่ว่านี้แบ่งเป็น 2 แบบ คือ 1.การสื่อสารแบบซิงโครนัส 2.การสื่อสารแบบอะซิงโครนัส

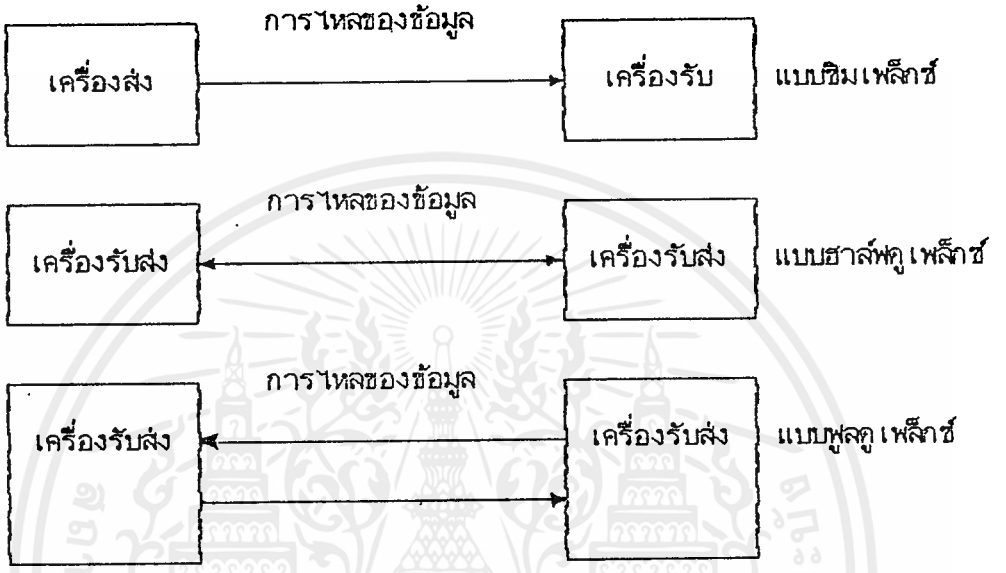
รูปแบบการติดต่อสื่อสารแบบอนุกรม

แบ่งตามรูปลักษณะได้ 3 แบบ ตามรูปที่ 2

1. แบบซิมเพล็กซ์ (simplex) ข้อมูลส่งได้ในทางเดียวเท่านั้น บางครั้งก็เรียกว่าการส่งทิศทางเดียว (Unidirectional data bus)

2. แบบฮาล์ฟดูเพล็กซ์ (half duplex) ข้อมูลสามารถส่งได้ทั้งสองด้าน แต่จะต้องผลัดกันส่งและผลัดกันรับ จะส่งและรับพร้อมกันไม่ได้

3. แบบฟูลดูเพล็กซ์ (full duplex) ทั้งสองด้านสามารถรับและส่งได้ในเวลาเดียวกัน



รูปที่ 2 รูปแบบของการติดต่อสื่อสารข้อมูลแบบอนุกรม

การส่งแบบฟูลดูเพล็กซ์และฮาล์ฟดูเพล็กซ์ ไม่ขึ้นอยู่กับจำนวนของสายในการติดต่อ บางครั้งคำว่า ทูไวร์ (two wire) หรือ 2 เส้น และโฟร์ไวร์ (four wire) หรือ 4 เส้น ใช้ในการบรรยายถึงลักษณะการสื่อสารข้อมูล ซึ่งอาจจะทำให้เข้าใจ และฮาล์ฟดูเพล็กซ์ สายโทรศัพท์ทั่วไปเป็นแบบ 2 เส้น ส่วนในสายที่เป็นแบบเช่า (lease line) นั้นส่วนมากจะเป็น 4 เส้น

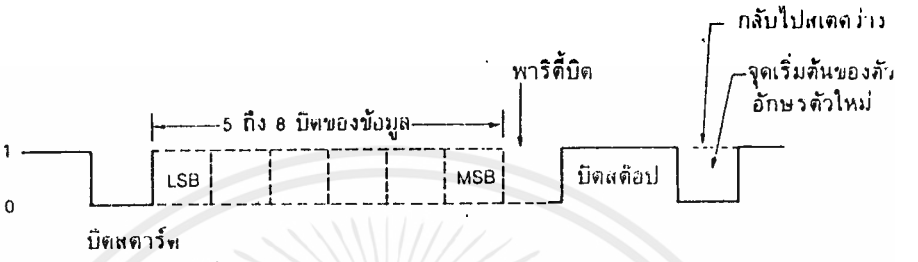
ความเร็วในการถ่ายโอนข้อมูลแบบอนุกรม

ความเร็วของการโอนข้อมูลแบบอนุกรม หน่วยวัดเป็นบิตต่อวินาที (bps) หน่วยที่บรรยายถึงการเปลี่ยนแปลงของสัญญาณใน 1 วินาที เรียกว่าบอดเรด (baud rate) หรืออัตราบอด การเปลี่ยนแปลงของสัญญาณ 1 ครั้ง อาจจะแสดงถึงการส่งข้อมูลแบบอนุกรมมากกว่า 1 บิต ถ้าเขียนในรูปของสมการทางคณิตศาสตร์เราก็จะได้

อัตราบิต (bit rate) = อัตราบอด (baud rate) * บิตใน 1 บอด

การสื่อสารแบบอะซิงโครนัส

การส่งแบบอะซิงโครนัสนี้ พัฒนามาจากการส่งโทรพิมพ์ในสมัยก่อน ลักษณะของสัญญาณแสดงในรูปที่ 3 เพื่อเพิ่มกลไกในการรับส่งอย่างถูกต้อง สัญญาณอะซิงโครนัสจะประกอบด้วย บิตเริ่มต้นหรือสตาร์บิต (start bit) และบิตสิ้นสุดหรือสต็อปบิต (stop bit)



รูปที่ 3 ฟอร์แมตการสื่อสารแบบอะซิงโครนัส

ขณะที่สถานะของการส่งเป็นแบบว่าง (idle) คือยังไม่มีสัญญาณส่งออกมา จะมีสัญญาณหรือแรงดัน (หรือกระแส) ตลอดเวลา เพื่อความแน่ใจว่าฝ่ายรับยังติดต่ออยู่กับฝ่ายส่ง เมื่อเริ่มจะส่งข้อมูล สัญญาณของอะซิงโครนัสจะเป็น 0 หนึ่งช่วงสัญญาณนาฬิกา บิตนี้เรียกว่าสตาร์บิต ตามหลังสตาร์บิตก็จะเป็นข้อมูลสำหรับ 1 ตัวอักษร ซึ่งอาจจะมีขนาดตั้งแต่ 5 บิต จนถึง 8 บิต โดยบิตที่มีค่าน้อยที่สุด (LSB) จะถูกส่งออกมาก่อน ไล่ไปจนถึงบิตที่มีค่ามากที่สุด (MSB) การเข้ารหัสอักขระนี้ส่วนมากจะนิยมใช้รหัส ASCII แรกเริ่มในงานโทรพิมพ์ใช้รหัส Baudot ซึ่งใช้ 5 บิต ในการแทนอักขระ 1 ตัว ตามหลังข้อมูลก็จะเป็นพาริตี้บิต ซึ่งอาจจะใช้หรือไม่ใช้ก็ได้ พาริตี้บิตทำหน้าที่เป็นตัวตรวจสอบ ความถูกต้องของสัญญาณที่ได้รับ พาริตี้บิตอาจเป็นแบบคู่ (Even) หรือแบบ คี่ (Odd) ถ้าหากเป็นคู่ จำนวนบิตที่เป็น 1 ในช่วงบิตข้อมูลกับบิตพาริตี้รวมแล้วจะต้องเป็นจำนวนคู่ ผู้ส่งจะต้องทำหน้าที่ตรวจสอบข้อมูลแล้วใส่พาริตี้บิตเอง ฝ่ายรับเมื่อรับแล้วก็ต้องตรวจสอบดูว่า เป็นจริงดังสถานการณ์ที่ตั้งเอาไว้หรือไม่ หากผิดพลาดก็หมายความว่าสัญญาณที่รับนั้นผิดพลาดไปจากสถานีส่งที่ส่งออกมา ทั้งนี้ทั้งนั้นจะต้องผิดเป็นจำนวนคี่เท่านั้น เช่นผิดไป 1 บิต 3 บิต หรือ 5 บิต พร้อมกันจึงตรวจสอบได้ว่าผิด มองเห็นง่ายๆ ว่าถ้าผิดเป็นจำนวนคู่ ผลรวมของจำนวนหนึ่งก็ยังเป็นคู่อยู่ดี ทั้งนี้ไม่ได้หมายความว่าพาริตี้คี่ (Odd Parity) จะตรวจสอบการผิดพลาดเป็นจำนวนคี่ ความจริงแล้วตรวจสอบความผิดพลาดได้เหมือนกับพาริตี้คู่ (Even Parity) แต่แทนที่จะตรวจสอบดูว่าสัญญาณที่รับเข้ามามีจำนวนคู่ ก็จะตรวจสอบดูว่ามีจำนวนคี่หรือไม่ อย่างไรก็ตามดอกาสที่จะผิดพลาด 2 บิตพร้อมกันมีน้อยมาก

กลับมาดูสัญญาณอะซิงโครนัสใหม่ หลังจากบิตพาริตี้แล้วก็จะต้องมีสต็อปบิต ซึ่งเป็น 1 ความกว้างของสต็อปบิตอาจเป็น 1,1.5 หรือ 2 พัลส์ของสัญญาณนาฬิกา แล้วแต่ผู้รับและผู้ส่งจะตกลงใช้กันเอง การเริ่มใช้พอร์ตอนุกรม (ทางออกอนุกรม) จึงจำเป็นจะต้องตั้งค่าต่างๆ สำหรับเป็นการส่งแบบอนุกรม อันได้แก่

1. ความเร็วในการส่ง
2. ความยาวรหัส 1 อักขระ
3. บิตตรวจสอบ
4. จำนวนสต็อปบิต

ในการส่งโทรพิมพ์หรือโทรเลขเมื่อก่อนนี้ใช้ความเร็วแค่ 70 บอด และ 110 บอด สำหรับคอมพิวเตอร์ความเร็วในการส่งมีให้เลือกตั้งแต่ 110,200,300,1200,2400,4800,9600 บอด และสูงไปกว่านั้น เนื่องจากมี IC หลายเบอร์ ทำหน้าที่รับส่งแบบอะซิงโครนัสให้ใช้ การส่งแบบอนุกรมจึงสะดวกสบายสำหรับคนออกแบบพอร์ตอนุกรม

กลไกในการซิงโครนัสของการสื่อสารอะซิงโครนัส มีลักษณะเป็นไปทีละตัวอักขระ จำนวนพัลส์ของสัญญาณที่ส่งออกยังมีบางส่วนใช้ในการควบคุมการส่งอยู่ได้แก่ บิตสตาร์ต บิตสต็อป และ บิตพาริตี้ ทำให้ความเร็วการส่งอักขระต่อวินาทีน้อยลงไป การส่งสัญญาณด้วยความเร็ว 300 บอด สำหรับการเข้ารหัส 7 บิต ไม่ได้หมายความว่าส่งได้ 300 หารด้วย 7 อักขระต่อวินาที

การสื่อสารข้อมูลแบบอะซิงโครนัสที่มีการแมตช์ความเร็ว

การโอนข้อมูลจากแบบอนุกรมจากอุปกรณ์เครื่องหนึ่งไปยังอีกเครื่องซึ่งอาจจะเป็นคอมพิวเตอร์หรืออุปกรณ์สื่อสารชนิดอื่น โดยสมมติฐานว่าความเร็วในการเปลี่ยนสัญญาณ จากขนานเป็นอนุกรมได้เร็วพอและฝ่ายรับเปลี่ยนจากอนุกรมเป็นขนานแล้วนำไปแสดงบนจอ พิมพ์ออกที่เครื่องพิมพ์หรือเก็บไว้ในดิสก์ทันทีได้ทันเวลาด้วยความเร็วในแต่ละการทำงานเท่ากันทั้งฝ่ายรับและฝ่ายส่ง ไม่มีการหน่วงเวลาหรือการอินเตอร์รัพต์ระหว่างกลาง อย่างไรก็ตามสมมติฐานนี้ย่อมไม่เป็นความจริง ฝ่ายส่งทำหน้าที่ส่งอย่างเดียว แต่ฝ่ายรับอาจต้องทำหน้าที่หลายอย่าง เช่น รับ แสดงผล เก็บ พิมพ์ เป็นต้น ความเร็วของฝ่ายรับหากไม่เพียงพอที่จะทำงานหลายอย่างให้ทันกับฝ่ายส่ง (แน่นอนย่อมขึ้นอยู่กับความเร็วในการส่ง) ก็จำเป็นจะต้องมีกลไกในการควบคุมการถ่ายโอน เทคนิคในการควบคุมความเร็วในการส่งมีอยู่หลายรูปแบบ ซึ่งอาจแบ่งออกได้เป็น 2 ลักษณะคือส่งข้อมูลบอกการ ON หรือ OFF การทำงาน(on - off data flow toggle) และหาที่เก็บข้อมูลชั่วคราวหรือสร้างบัฟเฟอร์ (Temporary data storage mechanism)

การควบคุมการส่ง เมื่อความเร็วในการทำงานของฝ่ายรับและฝ่ายส่งไม่เท่ากัน

เนื่องจากการใช้ภาษาในระดับสูง เขียนเป็นโปรแกรม สำหรับการควบคุมการทำงานของการถ่ายโอนข้อมูลแบบอนุกรม อาจจะใช้เวลามากกว่าที่จะรับข้อมูลเข้ามาได้ทันกับด้านส่งข้อมูลมา จำเป็นจะต้องมีวิธีการควบคุมไม่ให้เกิดการสูญหายของข้อมูลที่ด้านส่งส่งมา มีอยู่หลายวิธีคือ

1. การมีบัฟเฟอร์ในการสื่อสารข้อมูล

บัฟเฟอร์สำหรับการสื่อสารก็คือหน่วยความจำในคอมพิวเตอร์ ซึ่งแบ่งแยกออกมาจากหน่วยความจำหลักสำหรับเก็บพักข้อมูลในการติดต่อชั่วคราว บัฟเฟอร์สำหรับการสื่อสารนี้ ส่วนมากใช้สำหรับฝ่ายรับเท่านั้น เนื่องจากฝ่ายรับจำเป็นต้องตามฝ่ายส่งให้ทัน ถ้าหากฝ่ายรับใช้ภาษาแอสเซมบลี้ควบคุมมีความเร็วพออาจไม่จำเป็นต้องใช้บัฟเฟอร์สำหรับการสื่อสาร เนื่องจากภาษาแอสเซมบลี้มีความเร็วสูง

ข้อมูลที่จัดส่งให้คอมพิวเตอร์ที่เป็นฝ่ายรับ ส่วนมากจะอ่านมาจากแผ่นที่บันทึกไว้ในดิสค์หากพิจารณาระหว่างการส่งข้อมูลออก ข้อมูลที่อ่านมาจากดิสค์จะมีลักษณะเป็นกลุ่มได้รับการนำมาสู่บัฟเฟอร์ การอ่านแต่ละกลุ่มดำเนินไป จนกระทั่งบัฟเฟอร์เต็ม การอ่านจะหยุดลงจนกระทั่งบัฟเฟอร์ ถูกส่งออกไปหมดในลักษณะของเข้าก่อนออกก่อน ข้อมูลก็จะถูกอ่านออกมาใส่ในบัฟเฟอร์ส่งอีกครั้ง โดยปกติบัฟเฟอร์ส่งจะมีขนาด 255 ตัวอักษร หรือประมาณ 3 บรรทัดของ 80 อักษร

บัฟเฟอร์รับของฝ่ายรับมีผลกระทบต่อการรับ - ส่งข้อมูลมากกว่าบัฟเฟอร์ส่ง บัฟเฟอร์รับทำหน้าที่เช่นเดียวกับบัฟเฟอร์ส่ง แต่ทิศทางการไหลของข้อมูลอยู่ในทางตรงกันข้าม ฝ่ายรับรับข้อมูลเข้ามาเก็บไว้ในบัฟเฟอร์รับก่อนจนกว่าโปรแกรมควบคุมการสื่อสารจะนำข้อมูลออกไปจากบัฟเฟอร์รับ เพื่อไปแสดงหรือพิมพ์ หรือเก็บไว้ในแผ่นก็แล้วแต่ ในระบบควบคุมการทำงานคอมพิวเตอร์อย่างเช่น IBM PC มีกลไกบัฟเฟอร์รับส่งนี้ไว้อยู่ โปรแกรมในระดับสูงก็เพียงแต่ทำหน้าที่ดึงเอาข้อมูลจากบัฟเฟอรนี้ไปใช้

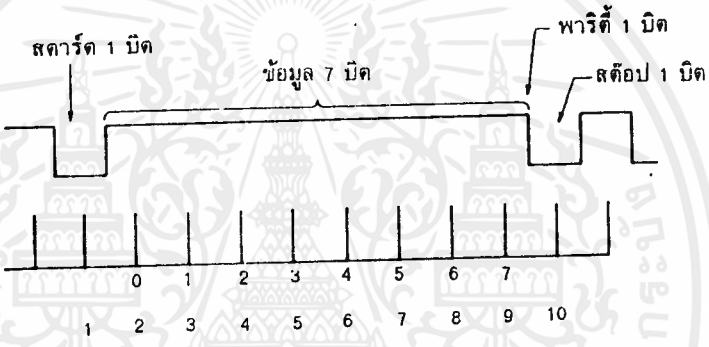
เราจะเห็นได้ชัดถึงความจำเป็นในการใช้บัฟเฟอร์ เมื่อความเร็วในการส่งสูงเกินกว่า 600 บอด ภาษาในระดับสูงเช่น เบสิกไม่สามารถที่จะรับข้อมูลจากพอร์ตอนุกรมได้ทัน ระบบควบคุมการทำงานจึงถูกออกแบบมาเพื่อการสื่อสารข้อมูล โดยการใช้อินเตอร์รัพท์เข้าช่วย เมื่อมีข้อมูลเข้ามาที่พอร์ตอนุกรมเมื่อไร ระบบควบคุมจะอินเตอร์รัพท์การทำงาน เพื่อดึงข้อมูลไปใส่ในบัฟเฟอร์รับทันที เพื่อไม่ให้ข้อมูลที่รับหายไปก่อนเมื่อมีตัวใหม่ส่งมาทับที่พอร์ตอนุกรม

หน้าที่ของโปรแกรมควบคุมการรับส่งคือ การอ่านข้อมูลจากบัฟเฟอร์รับไปใช้ เมื่อถูกอ่านจากบัฟเฟอร์รับไปแล้ว ตัวที่อ่านออกไปก็จะหายไปจากบัฟเฟอร์ จะเห็นว่าฝ่ายหนึ่งคือระบบควบคุมการทำงาน (OS) รับข้อมูลจากพอร์ตอนุกรมใส่บัฟเฟอร์ อีกฝ่ายหนึ่งคือโปรแกรมควบคุมการรับส่งดึงข้อมูลออกจากบัฟเฟอร์ เปรียบเสมือนคนหนึ่งดักน้ำใส่หุ่ม อีกคนตักออกจากหุ่ม ถ้าฝ่ายตักออกมีความเร็วมากกว่าหุ่มก็มีโอ

กาสเห้ง ในทางตรงกันข้ามถ้าฝ่ายดักออกช้ากว่า โอกาสที่จะล้นตุ่มก็ย่อมจะมี ในทางการสื่อสารเรียกว่า บัฟเฟอร์โอเวอร์โฟล์ว (receive buffer overflow) การไหลล้นดังกล่าวทำให้ข้อมูลที่ได้รับหายไป

การควบคุมโดยการใช้ XON/XOFF

ถึงแม้นว่าเราจะมีบัฟเฟอร์ สำหรับการสื่อสารแล้วก็ตาม ในบางครั้งการถ่ายโอนข้อมูลด้วยความเร็วสูง และด้วยขนาดของแฟ้มที่จะทำการถ่ายโอน มีขนาดใหญ่กว่าบัฟเฟอร์การสื่อสาร โอกาสที่ข้อมูลจะหายไปมีอยู่มาก ลองคำนวณดู เช่น เราใช้ความเร็วในการโอนถ่ายข้อมูล 9600 บิตต่อวินาที สต็อปบิต เป็น 1 บิต ข้อมูล 7 บิต และพาริตี้เป็นคู่ ใน 1 ตัวอักษรจะต้องใช้ 11 บิต



รูปที่ 4 รูปแบบของข้อมูล 1 ตัวอักษร

เพราะฉะนั้นฝ่ายรับจะต้องอ่านข้อมูลจากพอร์ตอนุกรมทุก 110/9600 ได้ผลประมาณ .001 วินาที หรือ 1 มิลลิวินาที ถ้าเปรียบเป็นพัลซ์ ได้ประมาณ 5000 พัลซ์นาฬิกา (ความเร็วนาฬิกาของ IBM PC = 4.77 MHz หรือประมาณ .2 ไมโครวินาทีต่อหนึ่งพัลซ์ 1000/.2 = 5000) ในเมื่อบัฟเฟอร์ไม่เพียงพอ เราก็จำเป็นจะต้องควบคุมการรับส่ง โดยการบอกให้ฝ่ายส่งหยุดส่งชั่วคราว (XOFF) จนกว่าฝ่ายรับจะจัดการเอาข้อมูล ออกจากบัฟเฟอร์สื่อสารหมดเสียก่อน จึงบอกให้ฝ่ายส่งจัดการส่งต่อไป (XON) ในรหัส ASCII XON มีค่าเท่ากับ 17 XOFF มีค่าเท่ากับ 19 เป็นหน้าที่ของนักเขียนโปรแกรมที่จะต้องจัดการส่ง XOFF ออกไปให้ฝ่ายส่งได้รู้ก่อนที่บัฟเฟอร์สื่อสารจะเต็มก่อน

คอมพิวเตอร์เมนเฟรมส่วนมากจะมีระบบ XON/XOFF ให้สำหรับการเชื่อมต่อทางด้านความเร็ว (Speed Matching) แต่โปรแกรมสื่อสารที่มีขาย สำหรับ IBM PC ไม่มี XON/XOFF ทุกตัว

การใช้โปรโตคอล

การส่งโปรโตคอล (Protocol Tranfer) เทคนิคนี้จำเป็นต้องมีเหมือนกันทั้งฝ่ายรับและฝ่ายส่ง โดยการใช้อักขระควบคุมในตารางของ ASCII สำหรับควบคุมการส่งข้อมูลออกมาเป็นกลุ่มที่มีขนาดคงที่ การใช้โปรโตคอลอาจจะใช้อักขระดังนี้ ในการควบคุมการส่งข้อมูลเป็นกลุ่มๆ

ETB End of Transmission block

มีค่า = 23 ในตาราง ASCII เป็นการบอกฝ่ายรับว่าขณะนี้สิ้นสุดการรับข้อมูลกลุ่มหนึ่งแล้ว

ETX End of Text

มีค่า = 03 เป็นการบอกฝ่ายรับว่า ขณะนี้สิ้นสุดการส่งแล้ว

ENG Enquiry

มีค่า = 05 เป็นการขอให้ฝ่ายส่งส่งข้อมูลมา

NAK Negative Acknowledge

มีค่า = 021 เป็นการบอกฝ่ายส่งว่าข้อมูลที่ได้รับนั้นผิดพลาด

ACK Acknowledge

มีค่า = 06 เป็นการบอกฝ่ายส่งว่า ข้อมูลที่ได้รับนั้นถูกต้องแล้ว

โปรโตคอลที่ใช้ใน IBM PC ส่วนมากจะเป็นโปรโตคอล Xmodem ฝ่ายส่งจะยังไม่ส่งข้อมูลจนกว่าจะได้รับ NAK จากฝ่ายรับ ฝ่ายส่งจะส่งข้อมูลออกไปโดยมีรูปแบบเริ่มต้นด้วย SOH ตามด้วยอักขระ 2 ตัว สำหรับบอกกลุ่มของข้อมูลที่ส่งและตามด้วยส่วนเติมเต็ม 1(1's complement) ของกลุ่มต่อไปที่จะส่ง ต่อจากนั้นก็จะเป็นข้อมูล 128 ไบต์ ตามด้วยการตรวจสอบข้อผิดพลาด โดยวิธีการตรวจสอบผลบวก (Checksum) การ Checksum คำนวณมาจากการบวกค่า ASCII ของข้อมูลที่ส่งออกไปทั้งหมด 128 ไบต์ แล้วหารด้วย 255 เศษที่เหลือก็คือค่า checksum ฝ่ายรับเมื่อแยกเอา SOH และหมายเลขบล็อก (Block number) ทั้ง 2 ออกไปแล้วก็จะเอาข้อมูลทั้ง 128 ไบต์มารวมกัน เพื่อหาค่า checksum เอา checksum ที่หาได้เปรียบเทียบกับค่าที่ได้รับ หากตรงกันก็ถือว่าข้อมูลที่ได้รับถูกต้อง จึงส่งสัญญาน ACK ไปให้ฝ่ายส่งได้รู้ว่า ขณะนี้ได้รับข้อมูลไว้ถูกต้อง แล้วส่งกลุ่มของข้อมูลต่อไปมาได้ ถ้าหากค่า CHECKSUM ไม่ถูกต้อง ฝ่ายรับก็จะส่ง NACK ให้ฝ่ายส่ง เพื่อบอกว่าข้อมูลที่ได้รับผิดพลาด ให้ส่งข้อมูลเดิมมาอีกที ฝ่ายส่งจะส่งข้อมูลเดิมมาให้อีกครั้ง (การส่งดำเนินไป 9 ครั้ง หากยังคงได้รับแต่ NAK ฝ่ายส่งจะหยุดทำงาน แสดงว่าตัวกลางการสื่อสารแย่มาก

การที่ Xmodem ใช้เลขนอกกลุ่ม (Block Number) 2 ตัว (ตัวหนึ่งบอกกลุ่มที่ส่งขณะนี้อีกตัวหนึ่งเป็นส่วนเติมเต็ม 1 ของกลุ่มต่อไป) เพื่อความแน่นอนว่ากลุ่มเดียวกันจะไม่ถูกส่งไปสองครั้งถ้าหากอักขระควบคุมการส่งเกิดสูญหายไประหว่างการส่ง ฝ่ายรับจะตรวจสอบดูว่า กลุ่มของข้อมูลที่ส่งมา เป็นกลุ่มที่ฝ่ายรับต้องการหรือไม่ ถ้าหากกลุ่มเก่าเกิดส่งมาใหม่อีกด้วยความผิดพลาด จาก ACK เป็น NACK ของฝ่ายส่ง ฝ่ายรับก็จะตัดข้อมูลที่รับมาทิ้งไป เมื่อทุกอย่างดำเนินไปอย่างเรียบร้อยจนสิ้นสุดแฟ้มที่จะส่งฝ่ายส่งจะส่ง ETX เป็นการบอกว่าฝ่ายรับหมดข้อความที่จะส่งแล้ว

ข้อดีของระบบควบคุม การรับส่งข้อมูลแบบอนุกรม โดยการใช้ระบบโปรโตคอล คือ

1. ใช้อักขระควบคุมที่มีอยู่แล้วใน ASCII
2. สามารถใช้ภาษาระดับสูงควบคุมได้ เช่น ภาษา C,ปาสคาล
3. ต้องการบัฟเฟอร์สื่อสารเพียง 256 ไบต์
4. ระบบริการข่าวสาร ด้วยคอมพิวเตอร์ โดยทั่วไปใช้โปรโตคอล Xmodem
5. โปรโตคอลบางชนิดสามารถเลือกขนาดของกลุ่มข้อมูลได้
6. สามารถส่งข้อมูลที่ไม่ใช่ ASCII ได้ โดยไม่ต้องกลัวว่ารหัสนั้นจะไปทับกับรหัสควบคุมของ ASCII
7. การตรวจสอบโดยวิธี checksum มีความสามารถตรวจสอบความผิดพลาดได้ดีกว่า ใช้บิตพาริตี้ในอะซิงโครนัสในขณะที่บิตพาริตี้สามารถให้ประสิทธิผลได้ 95% แต่ checksum สามารถให้ประสิทธิผล ถึง 99.5% หากพบการผิดพลาดด้วยบิตพาริตี้ ไม่ทำให้เกิดการส่งใหม่เกิดขึ้นแต่ข้อผิดพลาดจากการ checksum ทำให้เกิดการส่งข้อมูลมาใหม่

พอร์ตสื่อสารอนุกรม

พอร์ตสื่อสารนี้มีชื่อเรียกอีกอย่างหนึ่งว่า com port ผู้ออกแบบต้องการให้เป็นไป ตามมาตรฐานการเชื่อมต่อแบบอนุกรม ที่เรียกว่า RS 232 c อย่างไรก็ตามผู้ออกแบบให้ทางเลือกในการสื่อสารด้วยกระแสวนลูป หรือแบบแรงดัน RS 232 c โดยใช้จัมเปอร์เพื่อเลือกระบบ วงจรพอร์ตสื่อสารของไมโครคอมพิวเตอร์ 16 บิตนี้ใช้ไอซีเบอร์ 8250 มีขนาด 40 ขา ซึ่งความสามารถพิเศษมีดังนี้

- มีบัฟเฟอร์ในตัว ทำให้ไม่จำเป็นต้องซิงโครนัสการรับส่ง
- ใช้สัญญานนาฬิกาอิสระ ต่างหากไม่ขึ้นกับสัญญานนาฬิกาของระบบ
- มีสัญญานรับตอบควบคุมรวมเต็ม CTS,RTS,DSR,DTR,RC และสัญญานดีเทคตัวพาหะ (Carrier detect)
- ตรวจสอบสตาร์ตบิตที่ผิดพลาด

- ตรวจสอบและสร้างสัญญาณสายขาด (line break) เพื่อใช้ในการตรวจสอบการทำงานของระบบ

โครงสร้างของพอร์ตสื่อสาร

CPU ติดต่อกับ 8250 ในลักษณะพอร์ตที่เป็น INPUT OUTPUT การจัดพอร์ตนี้ กำหนดหมายเลขพอร์ตอย่างเจาะจง ระบบไมโครคอมพิวเตอร์ 16 บิต มีพอร์ตสื่อสาร 2 พอร์ตคือ com1 และ com2 ทั้ง 2 พอร์ตนี้มีขา I/P O/P พอร์ต ดังตารางที่ 1 การเลือกหมายเลข I/P O/P พอร์ต แยกเป็น 2 กลุ่ม กลุ่มหนึ่งคือ COM1 จะกำหนดหมายเลขพอร์ตจาก 3F8 ถึง 3FE อีกกลุ่มหนึ่งถ้ากำหนดหมายเลขพอร์ตเป็น 2F8 - 2FE ในการเลือกหมายเลขรีจิสเตอร์ภายใน กำหนดด้วยแอดเดรส 3 บิต คือ A_0 A_1 และ A_2 สำหรับการเลือก COM1 COM2 เราใช้ A8 เป็นตัวเลือก

การอินเตอร์รัพต์

ระบบสื่อสารอนุกรมได้กำหนดสัญญาณการอินเตอร์รัพต์ไว้แล้วคือ IRQ4 เป็นสัญญาณอินเตอร์รัพต์ของระบบ COM1 และ IRQ3 เป็นของ COM2 ในการที่จะส่งอินเตอร์รัพต์ต้องให้บิต 3 ของรีจิสเตอร์ควบคุมโมเด็มได้รับการเซตค่าเป็น "1" ก่อน จากนั้นข้อมูลอินเตอร์รัพต์ ที่อยู่ในรีจิสเตอร์อีนาเบิลอินเตอร์รัพต์ จะเป็นตัวส่งอินเตอร์รัพต์

การรีจิสเตอร์ต่างๆ บน 8250

Line Control Register ในการควบคุมรูปแบบของข้อมูล แบบอะซิงโครนัสนั้นผู้โปรแกรมจะต้องกำหนดค่าลงใน Register ควบคุมสายสื่อสาร Register ตัวนี้มีขนาด 8 บิต โดยแต่ละบิตมีความหมายดังนี้

บิต 0 และ 1	กำหนดความยาวในการรับส่งข้อมูล
บิต 2	กำหนดจำนวน stop bit
บิต 3	แสดงการอีนาเบิลให้มีการตรวจสอบพาริตี้
บิต 4	เป็นการกำหนดพาริตี้ว่าจะให้เป็นคู่ หรือ คี่
บิต 5	การแทรกหรือตรวจสอบพาริตี้
บิต 6	เป็นตัวควบคุมการเบรค
บิต 7	ทำหน้าที่ DLAB

ตารางที่ 1 หมายเลข I/P O/P PORT ของ COM1 และ COM2

Input Output PORT		เลือกรีจิสเตอร์	สถานะ DLAB
พอร์ต COM1	พอร์ต COM2		
3F8	2F8	บัฟเฟอร์ IX	DLAB = 0 (เขียน)
3F8	2F8	บัฟเฟอร์ RX	DLAB = 0 (อ่าน)
3F8	2F8	แลตช์ตัวหาร (LSB)	DLAB = 1
3F9	2F9	แลตช์ตัวหาร (MSB)	DLAB = 1
3F9	2F9	อีนาเบิลอินเตอร์รัพต์	
3FA	2FA	กำหนดอินเตอร์รัพต์	
3FB	2FB	ควบคุมสายสื่อสาร	
3FC	2FC	ควบคุมโมเด็ม	
3FD	2FD	แสดงสถานะสายสื่อสาร	
3FE	2FE	แสดงสถานะโมเด็ม	

การโปรแกรมอัตราบอด

อัตราบอดได้รับการกำหนดเทียบกับสัญญานนาฬิกา boud rate generater สามารถโปรแกรมตัวหารได้ตั้งแต่ 1-(2^{16} - 1) ค่าความถี่ output ของตัวกำหนดอัตราบอดมีค่าเท่ากับ 16 x อัตราบอดดังนั้น ตัวหาร = ความถี่สัญญานนาฬิกา / (อัตราบอด x 16) การกำหนดอัตราบอดด้วยการกำหนดตัวหารนี้ จึงเป็นค่าที่กำหนดในรีจิสเตอร์สองตัว ตัวหารนี้จะถูกกำหนดค่าก่อนแล้วโปรแกรมลงในรีจิสเตอร์ line status register รีจิสเตอร์ตัวนี้เป็นรีจิสเตอร์ที่จะให้ข้อมูลแก่ CPU เกี่ยวกับการสื่อสารข้อมูลในสายสื่อสาร

รีจิสเตอร์กำหนดอินเตอร์รัพต์ 8250 กำหนดความสำคัญของอินเตอร์รัพต์ไว้ 4 ระดับคือ

1. การรับข้อมูลจากสายสื่อสาร
2. การพร้อมรับข้อมูล
3. ขณะรีจิสเตอร์โฮลดิ้งสำหรับส่งว่าง
4. สัญญานสถานะโมเด็ม

ในขณะที่มีความต้องการอินเตอร์รัพต์ หลายระดับพร้อมกัน 8250 จะให้ระดับที่มีความสำคัญน้อยกว่ารอไว้ก่อนโดยเก็บสถานะการอินเตอร์รัพต์นี้ไว้ในรีจิสเตอร์กำหนดอินเตอร์รัพต์

รีจิสเตอร์อีนาเบิลอินเตอร์รัพต์

ใน com1 เมื่อให้ DLAB = 0 พอร์ต 3F9 จะเป็นรีจิสเตอร์อีนาเบิลอินเตอร์รัพต์ ผู้ใช้สามารถกำหนดให้เกิดอินเตอร์รัพต์หรือไม่ก็ได้ โดยการกำหนดค่าลงในรีจิสเตอร์นี้

รีจิสเตอร์ควบคุมโมเด็ม

รีจิสเตอร์ตัวนี้มีไว้ให้ CPU ส่งผ่านข้อมูลมาเก็บเพื่อเป็นรหัสสำหรับควบคุมการทำงานของโมเด็ม

รีจิสเตอร์แสดงสถานะโมเด็ม

ทำหน้าที่อ่านสถานะจากโมเด็มมาเก็บไว้เพื่อให้ CPU สามารถอ่านตรวจสอบดูได้

รีจิสเตอร์บัฟเฟอร์

สำหรับตัวรับข้อมูล เป็นรีจิสเตอร์สำหรับตัวรับข้อมูลที่มาจากสายสื่อสารสัญญาน

รีจิสเตอร์โฮลดิ้งสำหรับตัวส่งข้อมูล

เป็นรีจิสเตอร์สำหรับส่งข้อมูล ซึ่งจะรับข้อมูลจาก CPU ข้อมูลของ CPU จะถูกส่งไปยังสายข้อมูล

protocol

protocol มีหน้าที่สาเนาไฟล์จากต้นทางสู่ปลายทาง โดยผ่านทางสายการสื่อสารการทำงานของโปรโตคอลมีลักษณะคล้ายกับ คำสั่ง copy ใน dos คือสามารถส่งข้อมูลทีละไฟล์หรือหลายไฟล์ก็ได้ ในการส่งเราเรียกว่า Upload และการรับเราเรียกว่า Download การทำงานของโปรโตคอลแต่ละชนิดมีรูปแบบแตกต่างกันไป แต่หลักการโดยทั่วไปนั้น จะมีการติดต่อกับฝ่ายตรงกันข้ามก่อน เมื่อติดต่อสำเร็จจะมีการส่งข้อมูลเป็น Block โดยในการส่งแต่ละ Block นั้นก็มีการตรวจสอบก่อนแล้วจึงส่ง Block ต่อมาได้ และในลักษณะการส่งไฟล์นั้นจะมีรูปแบบการส่งไฟล์อยู่ 2 ชนิด คือ แบบ Single File protocol และ Tranfer Batch protocol ซึ่งความเร็วของชนิดหลังนี้จะมีความเร็วมากกว่าแบบแรก เนื่องจากว่า ในการส่งข้อมูลนั้น ฝ่ายรับไม่จำเป็นต้องรู้ชื่อไฟล์ล่วงหน้า

protocol ชนิดต่าง ๆ

- Packet ในการส่งไฟล์โปรโตคอลจะแบ่งไฟล์เป็นบล็อกๆ และเท่าๆ กัน แล้วจัดส่งทีละ บล็อก โดยในแต่ละบล็อกจะประกอบด้วยข้อมูลอื่นๆ ด้วย เช่น หมายเลขรหัสการตรวจสอบความผิดพลาด(CRC) และข้อมูลอื่นๆ เช่นในการส่ง packet ในระบบสายโทรศัพท์จะส่ง packet ขนาดเล็กทีละ 128 อักษรเท่านั้น เพราะหากเกิดความผิดพลาดขึ้นมาก็จะส่งได้ใหม่ในเวลาน้อย

- Error Checking Code เพื่อให้การรับส่งไม่เกิดความผิดพลาด จึงต้องมีการตรวจสอบในแต่ละ packet เมื่อฝ่ายส่งจะส่งข้อมูล โดยมีการคำนวณค่าตรวจสอบ หลังจากนั้นก็จะส่งไปยังด้านรับ ทางด้านรับก็จะคำนวณค่าตรวจสอบข้อมูล ที่ได้ไปเปรียบเทียบกัน ถ้าเท่ากันแสดงว่าข้อมูลถูกต้อง และถ้าไม่เท่ากันก็จะมีการส่ง packet เดิมกลับมาใหม่ วิธีการตรวจสอบอีกวิธีที่นิยมใช้และมีความแน่นอน คือ CRC (Cyclic Redundancy Check) ใช้วิธีแบบพหุนาม (polynomial) เป็นมาตรฐานที่ใช้ในการตรวจสอบข้อมูลโดยทั่วไป มีทั้งแบบ 16 และ 32 บิต

- Xmodem เป็นประเภทที่ส่งไฟล์ได้ครั้งละหนึ่งไฟล์ มีขนาด Packet เท่ากับ 128 ตัวอักษร ตรวจสอบความผิดพลาดด้วยวิธี CheckSum และยังมี version ที่พัฒนาขึ้นมาของ Xmodem เช่น Xmodem/crc ใช้วิธีการตรวจสอบความผิดพลาดแบบ crc-16 และ 1K-Xmodem เป็น Xmodem ที่มีขนาดของ packet เป็น 1024 ตัวอักษรตรวจสอบความผิดพลาดด้วย crc-16

- Ymodem เป็น Xmodem/crc ที่สามารถส่งไฟล์แบบ Batch คือส่งได้ครั้งละหลายๆ ไฟล์

- Zmodem เป็นโปรโตคอลที่นิยมใช้มากที่สุด เพราะมีความสะดวกรวดเร็วคือเป็น Batch มีขนาดเป็น packet เป็น 1024 ตัวอักษร ตรวจสอบความผิดพลาดแบบ crc-32 zmodem มีข้อดีกว่าชนิดอื่นคือ สามารถรับข่าวสารโดยไม่ต้องใช้ปุ่มใดๆ เมื่อฝ่ายส่ง ๆ ข้อมูลมาฝ่ายรับก็จะรับข้อมูลโดยอัตโนมัติ และยังใช้ในการติดต่อกับเครื่องใหญ่ เช่น Packet Switching Net-work

- Kermit เป็นการออกแบบเพื่อให้สามารถรับส่งข้อมูลและไฟล์ ระหว่างเครื่องที่มีสถาปัตยกรรมต่างกัน เช่นจาก Mainframe มายัง Microcomputer การส่ง packet จะเป็นแบบตัวอักษร ด้วยเหตุนี้เองจึงทำให้การรับส่งไฟล์เป็นไปได้ช้าจึงไม่นิยมใช้ในโมเด็ม

- Bimodem เป็นการรับส่งไฟล์ในเวลาเดียวกัน และสามารถส่งได้ 2 ทางได้ความเร็วในการส่งเป็น 2 เท่า

protocol สำหรับการโอนถ่ายแฟ้มข้อมูล

แฟ้มข้อมูลถ้ามองในแง่ของการโอนถ่าย ก็จะมีอยู่ 2 ชนิด คือ แฟ้มที่เป็นข้อความ (Text File) คือแฟ้มที่มีเฉพาะรหัสที่เป็นอักขระเท่านั้น ไม่รวมถึงรหัสควบคุมที่นอกเหนือไปจาก รหัสควบคุมการแสดงข้อความ เช่น LF,CR,FF,HT,VT ตัวอย่างเช่น แฟ้มที่สร้างขึ้นโดย Word star หรือ Note Pad ของ Side Kick แฟ้มอีกประเภทหนึ่ง เป็นแฟ้มที่มีข้อมูลที่อาจมีค่าได้ตั้งแต่ 0-255 เราเรียกแฟ้มนี้ว่า Binary file ตัวอย่างเช่น แฟ้มคำสั่งที่ลงท้ายด้วย COM หรือลงท้ายด้วย EXE การรับส่งแฟ้มประเภทข้อความนั้นไม่มีปัญหาในเรื่องที่จะทำให้ตัวโมเด็มเอง หรือตัวเทอร์มินอลตีความหมายผิดไป สำหรับแฟ้มประเภท Binary แล้ว รหัสข้อมูลภายในแฟ้มมีโอกาสเป็นไปได้ทุกชนิด หากส่งในลักษณะธรรมดาเหมือนข้อความ อาจทำให้เทอร์มินอลตีความหมายผิดๆ ไปได้ ดังนั้นในการรับส่งแฟ้มข้อมูลไม่ว่าจะเป็น แฟ้มข้อความ หรือแฟ้มเลขฐานสอง ทางที่ดีควรจะมี ระเบียบปฏิบัติสำหรับการรับส่งที่เป็นที่เข้าใจทั้งฝ่ายรับและฝ่ายส่ง ระเบียบปฏิบัติที่ว่านี้ก็คือ โปรโตคอลในการรับส่งข้อมูลนั่นเอง นอกจากจะช่วยในการรับส่งข้อมูลให้เป็นแบบแผนแล้วโปรโตคอลควรจะมีวิธีการ

1. ตรวจสอบข้อผิดพลาดในการรับส่ง
2. มีขบวนการที่แน่นอน ที่จะต้องปฏิบัติเมื่อมีข้อผิดพลาดในการรับส่ง
3. มีวิธีการเรียงลำดับของกลุ่มข้อมูลที่ส่ง
4. มีวิธีการที่จะทำให้ข้อมูลประเภทแฟ้มเลขฐานสองไปถึงฝ่ายรับได้

โปรโตคอลในการรับข้อมูล ส่วนมากจะอยู่ในการสื่อสารอนุกรม แบบ Synchronous เช่น BISYNCH, HDLC, SDLC เป็นต้น เมื่อการรับส่งแบบ Asynchronous มีปัญหา ก็จำเป็นต้องมี โปรโตคอลในการรับส่งข้อมูลได้

โปรโตคอลสำหรับรับส่งข้อมูลผ่านโมเด็มที่รู้จักกันดีคือ X MODEM, KERMIT, Y MODEM เป็นต้น X MODEM เป็นโปรโตคอลตัวแรกที่ใช้ใน Bulletin board และเริ่มมาใช้ครั้งแรกใน Bulletin bloard ส่วน KERMIT พัฒนาขึ้นมาทีหลัง เป็นการเพิ่มประสิทธิภาพในการรับส่งข้อมูลไกลๆ

แนวทางในการเขียนโปรแกรมรับส่งไฟล์ด้วย X MODEM

X MODEM รับส่งข้อมูลทีละ 128 ไบต์ เรียกว่า 1 Packet กลุ่มของข้อมูล (Packet) เริ่มต้นด้วย SOH (Start Of Heading) มีรหัสเอสกี้เป็น 1 ตามหลังด้วยหมายเลขกลุ่ม ของข้อมูล 8 บิต เริ่มต้นที่ 1 ไม่ได้เริ่มต้นที่ 0 ดังนั้นหมายเลขของกลุ่มข้อมูล ก็จะเริ่มต้นที่ 1 ไปจนถึง 255,0 แล้วก็ขึ้น 1 ใหม่ 8 บิต ที่ตามมาเป็นส่วนเติมเต็มหนึ่ง ของหมายเลขกลุ่มข้อมูล เช่น ถ้าหมายเลขกลุ่มเป็น 1 ส่วนเติมเต็มหนึ่งของ 1 ก็คือ 254 หรือ FE ในเลขฐาน 16 การที่มีทั้งหมายเลขของกลุ่มข้อมูลและส่วนเติมเต็มหนึ่ง ก็เพื่อต้องการตรวจสอบความถูกต้อง ของหมายเลขกลุ่มข้อมูล เมื่อไปถึงปลายทาง ถ้าทั้งสองไบต์นี้ไม่เป็นส่วนเติมเต็มหนึ่งของกันและกันแล้ว แสดงว่ามีอะไรผิดพลาดเกิดขึ้นแล้ว

ข้อมูล 128 ไบต์ ภายในกลุ่มก็จะตามมาไบต์สุดท้ายของกลุ่มก็จะเป็น ไบต์ตรวจสอบที่เรียกว่า check sum ซึ่งก็คือ ผลรวมของข้อมูล 128 ไบต์ ใน Modulo ของ 256 หรือเอาเฉพาะ 8 บิต สุดท้ายของการรวม ถ้าหากฝ่ายรับ รับแล้วปรากฏว่า รวมข้อมูลออกมาแล้วตรวจสอบกับ Check Sum ไม่เท่ากันก็แสดงว่ามีข้อผิดพลาดเกิดขึ้นในการรับส่งข้อมูลแล้ว วิธีการตรวจสอบแบบ ตรวจสอบผลรวม ยังมีจุดอ่อนหลายประการ เช่น เกิดมีการส่งไปแล้วมีสัญญาณรบกวนทำให้เมื่อไปถึงฝ่ายรับข้อมูลผิดไป 2 ไบต์ เผอิญที่ผิดไป รวมกันได้ 256 พอดี กรณีนี้การตรวจสอบผลรวมจะใช้ไม่ได้

จุดอ่อนของ X MODEM คือ ฝ่ายส่งต้องรอรับสัญญาณ Acknow Ledge จากฝ่ายรับทุกครั้งที่ส่งไป 1 กลุ่มข้อมูล ทำให้เสียเวลา ถ้าหากระยะทางของการสื่อสารยาวมากๆ เป็นต้นว่าส่งข้ามทวีป หรือข้ามประเทศที่ต้องผ่านดาวเทียม delay time ของสัญญาณมีมาก ประสิทธิภาพของการรับส่งก็จะด้อยลงไป

วิธีการของ Xmodem เริ่มต้น ฝ่ายรับส่งสัญญาณ Nagative Acknowledge (NAK : ASCII = 15H) เมื่อฝ่ายส่งได้สัญญาณ NAK แล้วถือเป็นการเริ่มต้นการส่ง ฝ่ายส่งจะเริ่มส่ง Packet แรกออกไป โดย Packet จะมีรูปแบบดังรูป 1 โดยแต่ละ Packet มีความหมายดังต่อไปนี้

SOH 8 bit	Packet Sequence NO 8 bit	1's Complement of Packet Sequence NO.	DATA (128 bytes)	CRC 16 bit

All fields but DATA are 1 byte

SOH มีรหัส 01H ใน ASCII Table บอกการเริ่มต้นของ Packet

Packet Sequence Number บอกว่าเป็น Packet ที่เท่าไร เพื่อให้ฝ่ายรับ ไปเรียงได้ถูกต้อง หมายเลข Packet เป็น 1 และถ้าเกิน 255 หรือ F ในฐาน 16 ไปเริ่มที่ 0

1's Complement of Packet Squence Number เป็นค่าคอมพรีเมนหรืออินเวอร์สของหมายเลข Packet

DATA เป็นข้อมูลที่ตัดมาจากไฟล์ ครั้งละ 128 ไบต์ ถ้ามีข้อมูลที่จะส่ง ไม่ถึง 128 ไบต์ ก็จะเติมให้เต็มด้วย End of file

Check Sum เป็นค่าผลบวกของข้อมูลทั้งหมดใน DATA field โดยนำมาทำ Modulo-256 ก่อน เมื่อฝ่ายรับได้รับ Packet แล้วจะตรวจสอบความถูกต้อง ถ้าถูกต้องก็รับด้วย ACK ฝ่ายส่งก็จะส่ง Packet ต่อไป เมื่อสิ้นสุด Packet ก็จะส่ง EOT (End of Text) ไปยังด้านรับ ซึ่งจะสรุปขั้นตอนได้ดังนี้

1. ฝ่ายรับส่งสัญญาณ NAK เพื่อบอกว่าพร้อมจะรับไฟล์แล้ว
2. ฝ่ายส่งเริ่มส่งข้อมูล Packet ทันที
3. ฝ่ายรับตรวจสอบข้อมูลแล้วว่าถูกต้องจึงส่ง ACK เพื่อบอกให้ส่งข้อมูลต่อไปได้
4. ฝ่ายส่งว Packet ที่สอง
5. ฝ่ายรับตรวจสอบแล้วถ้าพบว่าเกิดการผิดพลาดขึ้นใน Packet จะส่งสัญญาณ NAK เพื่อขอให้มีการส่ง Packet ที่ผิดพลาดนั้นมาใหม่
6. ฝ่ายส่งว Packet ที่สองมาใหม่
7. ตรวจสอบแล้วถ้าเห็นว่าถูกต้องจะส่งสัญญาณ ACK
8. ส่งข้อมูล Packet สุดท้ายซึ่งมีข้อมูลจริงเพียง 100 ไบต์ เท่านั้น ที่เหลืออีก 128 ไบต์ เป็นรหัสสุดไฟล์ CP/M
9. ตรวจสอบแล้วส่ง ACK
10. ส่งสัญญาณ EOT บอกการสิ้นสุดการส่งไฟล์
11. ฝ่ายรับตอบรับด้วย ACK เหมือนเดิม

รหัสในการสื่อสารและการควบคุม

รหัสในการสื่อสาร เป็น มาตรฐานที่มีความหมายอย่างเดียวกัน สำหรับผู้รับส่งหากไม่มีมาตรฐานเสียแล้วรหัสเหล่านี้ก็ไม่มีความหมายอะไร แต่ละเครื่องที่ใช้ในการสื่อสารก็จะเข้ากันไม่ได้ โดยเฉพาะเมื่อมีผู้ผลิตหลายยี่ห้อหลายสถานที่

มีรหัสที่ใช้กันอย่างแพร่หลายอยู่ 2 รหัสคือ ASCII ย่อมาจาก American Standard Code for Information Interchange ซึ่งใช้กันแพร่หลายทั่วโลก อีกรหัสหนึ่งเป็นของบริษัท IBM ซึ่งใช้กับเครื่องคอมพิวเตอร์ขนาดใหญ่และขนาดกลางมานาน รหัสที่ว่าคือ EBCDIC ย่อมาจาก Extended Binary Coded Decimal Interchange Code

เนื่องจาก ASCII เป็นรหัสที่ใช้กันอย่างแพร่หลายในระดับเครื่องไมโครคอมพิวเตอร์ ซึ่งแม้แต่ IBM PC ก็ยังใช้รหัส ASCII

อักขระในชุดของ ASCII

ใน 128 ตัวแรกของรหัส ASCII ได้รับการกำหนดมาตรฐานเอาไว้ใน ANSIX 3.4 ปี ค.ศ. 1977 (standard ASCII character set) เครื่องไมโครคอมพิวเตอร์เกือบทั่วโลกใช้รหัสนี้ นับว่าไมโครคอมพิวเตอร์เป็นตัวอย่างอันดีในการใช้มาตรฐานเดียวกัน

ตารางที่ 2 ตารางรหัส ASCII

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

ตารางที่ 2 เป็นตารางที่แสดงถึงรหัส 128 ตัวแรกของ ASCII ซึ่งประกอบด้วยอักขระภาษาอังกฤษ 26 ตัวใหญ่ 26 ตัวเล็ก เลขอารบิก 10 ตัว และอักขระพิเศษ ที่มีอยู่ตามเครื่องพิมพ์ดีดทั่วไป รหัสการสื่อสารถูกรวมเข้าไปใช้อยู่ในที่นี้ด้วย

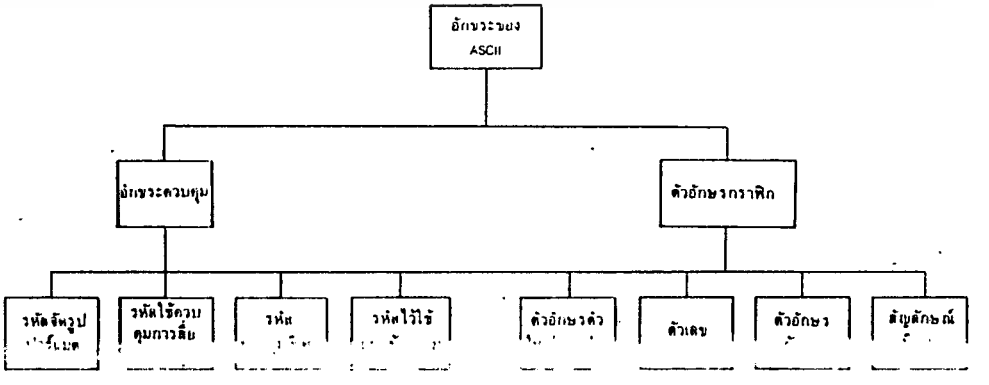
ถึงแม้ว่ารหัส ASCII ใช้แค่ 7 บิต ส่วนมากไมโครคอมพิวเตอร์จะใช้ 8 บิต ในการเข้ารหัสอักขระ รหัสที่เป็นมาตรฐานของ ASCII จะมีบิตที่ 8 เป็น 0 นั่นคือ 128 ตัวแรกในจำนวน 256 ตัว จะเป็นรหัสมาตรฐาน ASCII ส่วนอีก 128 ตัวหลังจึงเริ่มจาก 128 ถึง 255 ซึ่งมีบิตที่ 8 เป็น 1 นั้น ไมโครคอมพิวเตอร์ส่วนมาก ใช้เป็นรหัสสำหรับสัญลักษณ์ทางด้านกราฟฟิก ซึ่งแต่ละยี่ห้อของคอมพิวเตอร์จะไม่เหมือนกัน สำหรับคนไทยนำเอา 128 ตัวหลังนี้มาเข้ารหัสเป็นรหัสอักขระภาษาไทย

ANSI (American National Standard Institute) กำหนดมาตรฐานของรหัส ASCII ออกเป็นสองพวกใหญ่ๆ ตามลักษณะการใช้งานคือ อักขระที่ทำให้เกิดข้อความที่อ่านเข้าใจได้เรียกว่า ตัวอักขระกราฟฟิก (Graphic character) และอักขระที่ใช้ทำให้เกิดการควบคุม เรียกว่าอักขระควบคุม (Control character)ชื่อของพวกแรกอาจจะทำให้เข้าใจผิดได้ เนื่องจากอักขระกราฟฟิกก็คืออักขระที่เป็นพวกที่สามารถอ่านได้เข้าใจจริงๆ ไม่รวมถึงพวกที่จะทำให้เกิดรูปกราฟฟิกตามชื่อ พวกที่เป็นกราฟฟิกในไมโครคอมพิวเตอร์ทั่วไป ควรจะอยู่ในจำพวกตัวอักขระใช้งานเฉพาะ (special extended - characters)

อักขระควบคุมของ ASCII แบ่งออกเป็นกลุ่มย่อยตามการใช้งาน คือ

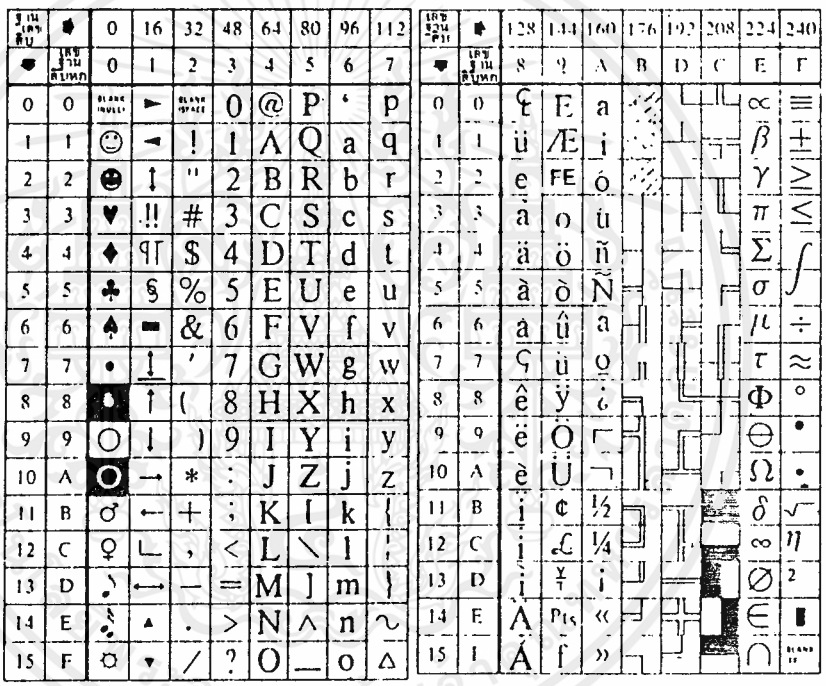
- รหัสที่ควบคุมการสื่อสารข้อมูล (communication control)
- รหัสจัดรูปฟอร์แมต (format effectors)
- รหัสไว้ใช้แยกข้อความ (information separator)
- รหัสควบคุมพิเศษ (special control characters)

รูปที่ 5 แผนภูมิของอักขระ ASCII



รูปที่ 5 แสดงถึงแผนภูมิของอักขระในรหัสของ ASCII กลุ่มของรหัสที่ใช้ในการควบคุมทั้งหมด บางทีก็เรียกว่า รหัสที่ไม่สามารถพิมพ์ออกมาดูได้ (non printing control character) แต่ในไมโครคอมพิวเตอร์บางเครื่องใช้อักขระเหล่านี้ สำหรับการเข้ารหัสรูปกราฟฟิก ดังตัวอย่างของ IBM PC แสดงไว้ใน ตารางที่ 3

ตารางที่ 3 รหัสที่ใช้ในเครื่อง IBM PC



ตารางที่ 4 อักขระควบคุมของ ASCII

ASCII value	IBM-PC Character	Control Character	ASCII Character	Group	Communication Usage
000	(null)	^@	NUL	CC	Null character -- filler
001	☺	^A	SOH	CC	Start of heading
002	☻	^B	STX	CC	Start of text
003	♥	^C	ETX	CC	End of text
004	♦	^D	EOT	CC	End of transmission
005	♣	^E	ENQ	CC	Enquiry
006	♠	^F	ACK	CC	Acknowledge affirmative
007	(beep)	^G	BEL	SC	Audible alarm
008	(backspace)	^H	BS	FE	Backspace one position
009	(tab)	^I	HT	FE	Physical horizontal tab
010	(line feed)	^J	LF	FE	Line feed
011	(home)	^K	VT	FE	Physical vertical tab
012	(form feed)	^L	FF	FE	Form feed
013	(enter)	^M	CR	FE	Carriage return
014	♫	^N	SO	SC	Shift out
015	☼	^O	SI	SC	Shift in
016	►	^P	DLE	CC	Data link escape
017	◄	^Q	DC1	SC	XON or resume
018	↕	^R	DC2	SC	Device control 2
019	‼	^S	DC3	SC	XOFF or pause
020	¶	^T	DC4	SC	Device control 4
021	§	^U	NAK	CC	Negative acknowledgment
022	▬	^V	SYN	CC	Synchronous idle
023	↨	^W	ETB	CC	End of transmission block
024	↑	^X	CAN	SC	Cancel
025	↓	^Y	EM	SC	End of medium
026	→	^Z	SUB	SC	Substitute
027	←	^[	ESC	SC	Escape
028	(cursor right)	^\	FS	IS	File separator
029	(cursor left)	^]	GS	IS	Group separator
030	(cursor up)	^^	RS	IS	Record separator
031	(cursor down)	^_	US	IS	Unit separator
127	⌂		DEL	SC	Delete

กลุ่มที่เป็น CC คือ รหัสควบคุมการสื่อสารข้อมูล (communication control) ใช้งานทางด้านการควบคุมการรับส่งในช่ายการสื่อสาร อักขระเหล่านี้ เช่น SYN,SOH,STX,ETX และ DLE แยกกล่าวไว้แล้วในโปรโตคอล Bisyn อักขระเหล่านี้ใช้ประจำในการสื่อสารแบบซิงโครนัสเป็นการบอกให้ฝ่ายรับได้รู้ว่าอะไรควรจะเกิดขึ้นต่อไปและบอกให้รู้ว่าข้อมูลที่ส่งมาเป็นชนิดไหน

กลุ่ม FE หรือ Format Effector ใช้ในการควบคุมตำแหน่งของ อักขระที่จะพิมพ์หรือแสดงผล ตัวอย่าง เช่น HT,VT,FF,CR,SO, และ SI เป็นต้น การส่งอักขระเหล่านี้ไปยังเครื่องพิมพ์จะเป็นการควบคุมตำแหน่งที่จะพิมพ์โดยตรง

กลุ่ม SI หรือ Information Separator ใช้ในการแยกกลุ่มของข้อมูล ในการส่งผ่านช่องการสื่อสาร อักขระเหล่านี้ในค่อยได้ใช้ในการสื่อสารของไมโครคอมพิวเตอร์ เช่น GS,FS และ US

กลุ่ม SC หรือ Special Character ใช้ในการควบคุมการพิมพ์ ควบคุมการปรับความเร็ว (speed matching) หรือใช้เป็นสัญญาณบอกความผิดพลาดอักขระบางตัวเคยกล่าวถึงแล้วใน Bisyn

[illegible] ESC,DEL

อักขระควบคุมการสื่อสาร

ANSI ได้ให้คำนิยามในการใช้อักขระควบคุมในการสื่อสาร เพื่อความเข้าใจได้ตรงกันดังนี้

NUL

เป็นอักขระว่าง คือไม่มีอะไรเลย อย่าสับสนกับ Blank เพราะ Blank ยังมีอักขระอยู่อีก คือ " " ในภาษาเบสิก ส่วน NUL = " " ในภาษาเบสิก NUL อาจจะแทรกหรือดึงออกจาก สายต่อเนื่องของอักขระโดยไม่มีผลกระทบกระเทือน การแทรกหรือ ดึง NUL เข้าออก เพียงเพื่อต้องการผลในแบบแปลนในการส่งข้อมูลเท่านั้น (เช่น ทำให้ได้ความยาวของข้อมูลตามต้องการ)อักขระนี้มีประโยชน์สำหรับอุปกรณ์เครื่องพิมพ์ซึ่งต้องการ เวลาที่แน่นอนในการเลื่อนหัวพิมพ์ เครื่องพิมพ์บางตัวต้องการอักขระ NULL อย่างน้อยสองตัวตามหลัง CR (Carriage Return) เพื่อเป็นการตั้งหัวพิมพ์ให้เข้าทางซ้ายก่อนที่จะรับอักขระต่อไปได้

SOH

Start of Heading (SOH) ใช้ในการควบคุมในสายข้อมูลของ Bisyn ซึ่งหมายถึงจุดเริ่มต้นของส่วนนำหน้าของข้อมูล สถานีในข่ายการสื่อสารจะตรวจสอบอักขระที่ตามหลัง SOH สำหรับตีความว่าจะเป็นผู้รับข้อความที่ตามหาข้างหลังนี้หรือไม่

SOH บางครั้งก็ใช้ในแบบอะซิงโครนัส สำหรับการโอนถ่ายข้อมูลหลาย ๆ แฟ้มติดต่อกันโดยไม่เยกแฟ้มส่ง SOH ใช้สำหรับบอกถึงจุดเริ่มต้นของแต่ละแฟ้ม ในการสื่อสารแบบอะซิงโครนัส มีผู้รับอยู่สถานีเดียวจึงไม่จำเป็นต้องใช้ตำแหน่งที่อยู่ของสถานีรับ เพียงแต่ใส่ชื่อแฟ้มตามหลัง เท่านั้น

การรับส่งแบบนี้ใช้ เฉพาะใน โปรแกรมสำเร็จรูปที่เหมือนกันเท่านั้น ยังไม่ได้เป็นมาตรฐานของการสื่อสาร SOH ใช้ในโปรโตคอล XMODEM เพื่อเป็นการบอกให้รู้ว่า กลุ่มของข้อมูลอีก 128 ไบต์ กำลังตามมา

STX

Start of Text (STX) ใช้ใน Bisyn เป็นการเตือนให้ฝ่ายรับได้รู้ว่าข้างหลังที่ตามมาเป็นข้อความที่ต้องการจะส่งให้ และเป็นการบอกว่าส่วนหัวได้สิ้นสุดลงแล้ว

ETX

End of Text (ETX) ใช้ใน Bisyn เหมือนกัน เป็นการบอกฝ่ายรับว่าข้อความที่ต้องการส่งในกลุ่มนี้หมดแล้ว

EOT

End of Transmission (EOT) เป็นอักขระควบคุมการสื่อสาร เพื่อเป็นการบอกให้รู้ว่าสิ้นสุดการส่งข้อมูลไปยังอุปกรณ์ที่บ่งบอกแล้ว อักขระนี้ยังเป็นการบอกให้อุปกรณ์อื่น ๆ ที่อยู่ในข่ายสื่อสาร คอยตรวจสอบว่าจะมีข้อความ ส่งมาถึงตัวเองหรือเปล่า EOT เป็นตัวชี้ถึงจุดสุดท้ายของเฟรมที่เริ่มต้น โดยอักขระ SOH ในโปรโตคอล XMODEM ใช้อักขระตัวนี้คอยบอกการสิ้นสุดของการถ่ายโอนแฟ้มข้อมูล

ETB

End of Transmission Block (ETB) เป็นอักขระควบคุมการสื่อสาร เป็นตัวบ่งชี้ถึงการสิ้นสุดกลุ่มข้อมูลที่ส่ง Bisyn ใช้อักขระตัวนี้แทน ETX เมื่อข้อมูลที่ส่งมากกว่าสองกลุ่ม

ENQ

Enquiry (ENQ) อักขระควบคุมตัวนี้ใช้ในการเรียกการตอบสนองจากฝ่ายรับ อาจจะใช้ในการเรียกให้อุปกรณ์ในข่ายการสื่อสารแสดงตัวเอง หรืออาจจะใช้ในการบอกสถานภาพของการสื่อสาร โปรแกรมสำเร็จรูปที่ใช้กับ IBM PC บางตัวใช้อักขระตัวนี้ในโปรโตคอลการส่งแฟ้มข้อมูล เพื่อให้เสดงการตอบสนองด้วย ENQ ฝ่ายรับอาจจะต้องการที่จะให้ส่งกลุ่มข้อมูลที่รับไปหลังสุด อย่างไรก็ตามนี้เป็นการใช้ ENQ อย่างไม่เป็นมาตรฐาน

ACK

Acknowledge (ACK) อักขระควบคุมใช้ในการบ่งบอกถึงการสื่อสาร และถ่ายโอนข้อมูลอย่างไม่ผิดพลาด ระหว่างด้านรับและด้านส่ง หลังจากรับกลุ่มข้อมูลอย่างถูกต้องแล้ว ฝ่ายรับจะต้องส่ง ACK ให้ฝ่ายส่งได้รับรู้ว่าข้อความที่ส่งไปได้รับถูกต้องแล้ว

NAK

Negative Acknowledge (NAK) ใช้ระหว่างฝ่ายรับกับฝ่ายส่งหรือบ่งบอกว่ามีการผิดพลาดเกิดขึ้นในการสื่อสาร โดยทั่วไป NAK จะส่งโดยฝ่ายรับไปยังฝ่ายส่ง เมื่อมีการผิดพลาดเกิดขึ้น และเป็นการขอให้ฝ่ายส่งส่งข้อความเก่ามาใหม่ ENQ,ACK และ NAK ส่วนมากจะใช้ด้วยกันในโปรโตคอลการรับส่ง แต่ผู้ใช้จะไม่ยุ่งเกี่ยวกับรหัสเหล่านี้

BEL

BELL เป็นอักขระควบคุมที่ทำงาน คือส่งเสียงเรียกร้องความสนใจจากผู้ใช้ อาจจะรวมอยู่ในแฟ้มหรือรับส่งระหว่างการสื่อสารที่เป็นลักษณะสนทนาใน IBM PC อาจจะทำได้ทันทีโดยกด CTRL และ G พร้อมกัน

BS

Back Space (BS) เป็นอักขระสำหรับจัดรูปร่าง เพื่อควบคุมเครื่องพิมพ์และการแสดงผลบนหน้าจอ โดยจะมีผลให้เคอร์เซอร์หรือหัวพิมพ์เลื่อนไปทางซ้าย 1 ตำแหน่ง ในขณะที่เคอร์เซอร์ไม่ได้อยู่ในตำแหน่งที่ 1 อักขระที่อยู่ในตำแหน่งที่ เคอร์เซอร์เลื่อนไปทับ จะหายไป บางครั้งเราเรียกอักขระนี้ว่า อักขระถอยหลังและลบ

HT

Horizontal Tabulation (HT) เป็นอักขระสำหรับจัดรูปร่าง ทำให้เครื่องพิมพ์และการแสดงผลบนหน้าจอ เลื่อนไปตำแหน่งที่ตั้งค่าเอาไว้ก่อนแล้ว ก่อนที่จะพิมพ์อักขระตัวต่อไป

LF

Line Feed (LF) เป็นอักขระควบคุมทำให้ตำแหน่งที่พิมพ์เลื่อนไป 1 บรรทัด นับต่อไปในแถวตั้งอันเดียวกัน อาจจะสับสนกับ Carriage Return (CR) แต่ว่า LF ไม่ได้เลื่อนหัวพิมพ์ หรือเคอร์เซอร์ไปยังแถวตัวที่ 1 ของบรรทัดถัดไป นอกจากจะตามด้วย CR

VT

Vertical Tabulation (VT) เป็นอักขระควบคุมการจัดรูปร่าง ซึ่งจะทำให้เครื่องพิมพ์หรือการแสดงผลบนหน้าจอเลื่อนไปยังแถวตั้งเดิมแค่จำนวนบรรทัดห่างออกไปเท่ากับจำนวนบรรทัดที่ตั้งเอาไว้ก่อนหน้านี้

FF

Form Feed (FF) เป็นอักขระควบคุมรูปร่าง ใช้ในการเลื่อนหัวพิมพ์ไปยังส่วนบนของแบบฟอร์มที่ตั้งเอาไว้ของหน้าถัดไป หรือคือการขึ้นหน้าใหม่ของเครื่องพิมพ์

CR

Carriage Return (CR) เป็นอักขระควบคุมการจัดรูปร่าง ซึ่งจะทำให้เครื่องพิมพ์หรือเคอร์เซอร์ของการแสดงผลบนหน้าจอเลื่อนไปยังตำแหน่งที่ 1 ของบรรทัดเดียวกัน

SO

Shift Out (SO) เป็นอักขระควบคุมพิเศษ ซึ่งทำหน้าที่ขยายอักขระมาตรฐานของ ASCII ออกไป สำหรับเครื่องพิมพ์อาจจะใช้รหัสควบคุมตัวนี้ ทำให้เปลี่ยนโหมดการพิมพ์เป็นพิมพ์ตัวใหญ่หรือความหนาเป็นสองเท่า

SI

Shift In (SI) เป็นอักขระควบคุมพิเศษซึ่งอาจจะใช้ ในการตั้งค่าใหม่ของ เครื่องรับกลับไปยังอักขระมาตรฐานของ ASCII ในเครื่องพิมพ์อาจจะใช้อักขระตัวนี้ ในการโคดกลับเข้ามาพิมพ์ตัวขนาดมาตรฐานหลังจากที่ใช้ SO พิมพ์ตัวหนาสองเท่า

DLE

Data Link Escape (DLE) เป็นอักขระควบคุมการสื่อสาร ใช้ในการดัดแปลงแก้ไขความหมายของอักขระที่ตามหลังมา ใช้ใน Bisyn สำหรับหมายถึงการเริ่มต้นของข้อมูล Transparent

DC1

Device Control 1 (DC1) เป็นอักขระสำหรับกิจกสวิตช์ทางอิเล็กทรอนิกส์ การใช้งานจริง ๆ อาจจะแตกต่างกันไปสำหรับบริษัทผู้ผลิตอุปกรณ์สื่อสาร หรือซอฟต์แวร์ของแต่ละบริษัท แต่โดยทั่วไปแล้วใช้ในการควบคุมอุปกรณ์ที่เกี่ยวข้องเอาไว้สำหรับการใช้แสดงผล อักขระตัวนี้มีค่าเท่ากับ ctrl-Q จะทำให้เกิดการเริ่มต้นการแสดงผลใหม่ หลังจากหยุดไปเพราะ DC3 หรือ ctrl-S ในทางการสื่อสารข้อมูลอักขระตัวนี้ส่วนมากจะมีค่าเท่ากับ XON และใช้สำหรับการเริ่มถ่ายโอนใหม่หลังจากหยุดไปเพราะ XOFF

DC2

Device Control 2 (DC2) เช่นเดียวกับ DC1 ความหมายอาจจะแตกต่างกันออกไปแล้วแต่ผู้ผลิตจะนำไปใช้

DC3

Device Control 3 (DC3) เป็นอักขระสำหรับ กิจกสวิตช์อิเล็กทรอนิกส์อีกอันหนึ่ง ส่วนมากจะใช้ร่วมกับ DC1 เสมอ สำหรับ (Speed Matching) ปรับความเร็วให้เข้ากัน DC3 ก็คือ XOFF และใช้ในการหยุดส่งชั่วคราว

DC4

Device Control 4 (DC4) ก็เช่นเดียวกับ DC2 ความหมายอาจจะแตกต่างกันไปแล้วแต่ผู้ใช้

SYN

Synchronous Idle (SYN) เป็นอักขระควบคุมการสื่อสาร ใช้ในโปรโตคอล Bisyn สำหรับเริ่มต้นการสื่อสาร หรือเมื่อยังไม่มีข้อมูลจะส่ง สำหรับให้ฝ่ายรับปรับสัญญาณนาฬิกาให้เข้ากับฝ่ายส่งเสมอ

CAN

CANCEL (CAN) เป็นอักขระควบคุมพิเศษ อาจจะนำไปใช้งานต่างๆ กัน สำหรับผู้ใช้ในแต่ละราย แต่ส่วนมากจะหมายถึง เกิดมีข้อผิดพลาดในการส่งข้อมูล อักขระตัวนี้เป็นตัวบ่งบอกว่าข้อมูลที่ได้รับเข้าก่อนหน้านี้ผิดพลาด (เพื่อจากฝ่ายส่งเอง) ทิ้งไปก่อน

EM

End of Medium (EM) เป็นอักขระควบคุมพิเศษใช้ในการบอก การสิ้นสุดของตัวกลางในทางพิสิกส์ (หน่วยบันทึกข้อมูล อันเป็นตัวแทนของตัวกลางการสื่อสาร) หรือสิ้นสุดของตัวกลางที่มีข้อมูลอยู่

SUB

Substitute (SUB) อักขระนี้ในการสื่อสาร เพื่อการควบคุมความเม้นยำ เครื่องรับใช้ SUB แทนอักขระที่ถูกตีความว่าผิดพลาด ไม่ถูกต้อง หรือเป็นไปไม่ได้ ในการส่งให้เครื่องพิมพ์หรือหน่วยแสดงผล

ESC

Escape (ESC) เป็นอักขระควบคุม ใช้กันอย่างแพร่หลายในการสื่อสารระหว่าง เครื่องพิมพ์กับคอมพิวเตอร์ ปกติจะใช้ส่งออกไปก่อนที่จะตามด้วยอักขระ หรือตัวเลขที่เป็นรหัสเสริมต่อหรืออักขระเสริมต่อ ซึ่งเปลี่ยนแปลงไปตามผู้ผลิตเครื่องพิมพ์ที่อักขระเสริมต่อนั้นจะมีความหมายอย่างไร

FS

File Separator (FS) เป็นตัวรับข่าวสารใช้สำหรับแสดงถึงขอบเขตในทางลอจิกระหว่างแฟ้มที่ถูกถ่ายโอน

GS

Group Separator (GS) เป็นอักขระคั่นกลางใช้ในการแสดงขอบเขต ในทางลอจิกของกลุ่มข้อมูลที่กำลังส่งออกไป

RS

Record Separator (RS) เป็นอักขระสำหรับคั่นกลาง เหมือนกันใช้ในคั่นระหว่าง เรคอร์ด (Record) ที่กำลังส่งออก

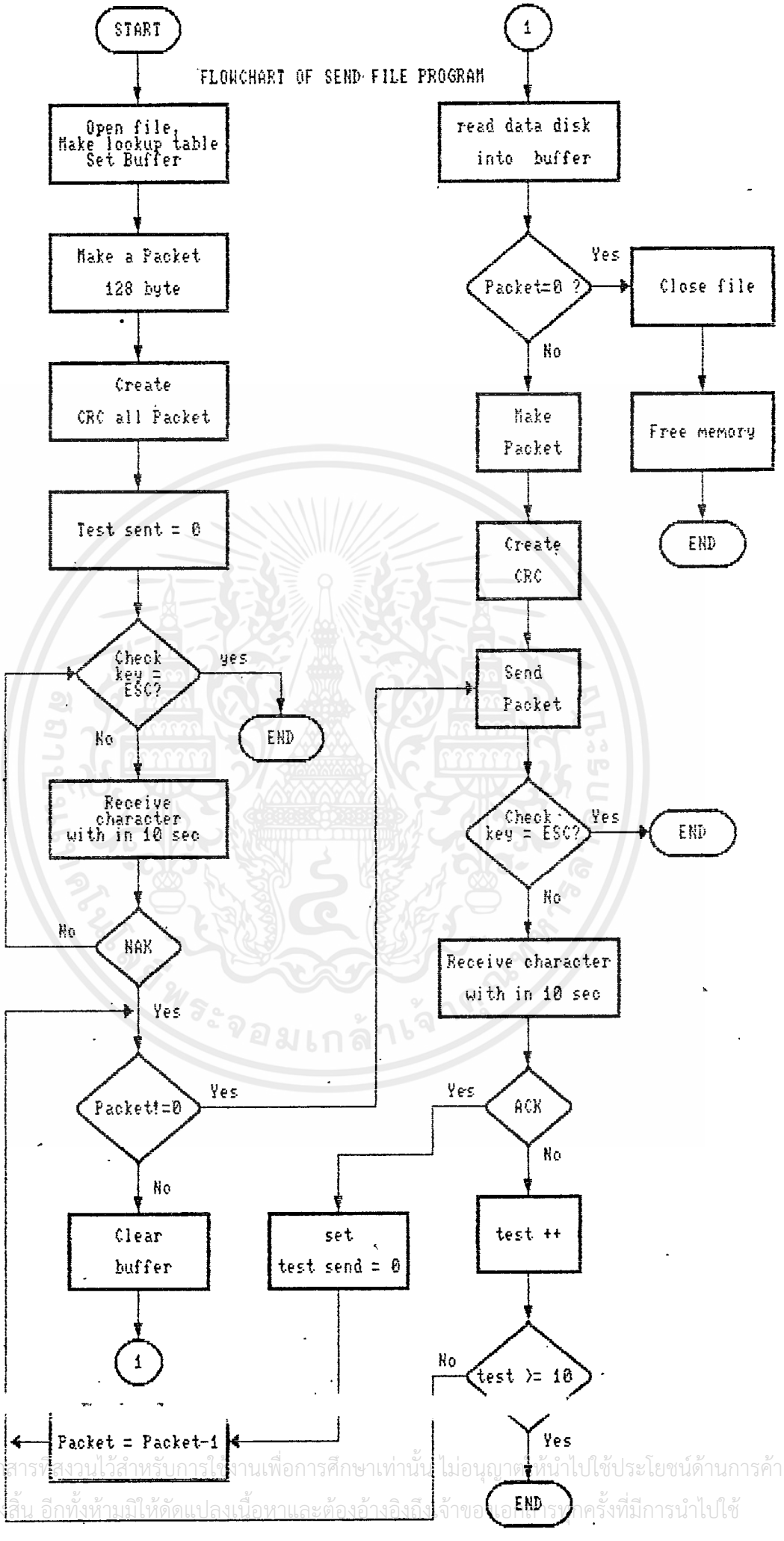
US

Unit Separator (US) ใช้ในการคั่นระหว่างข้อมูลที่แตกต่างกัน

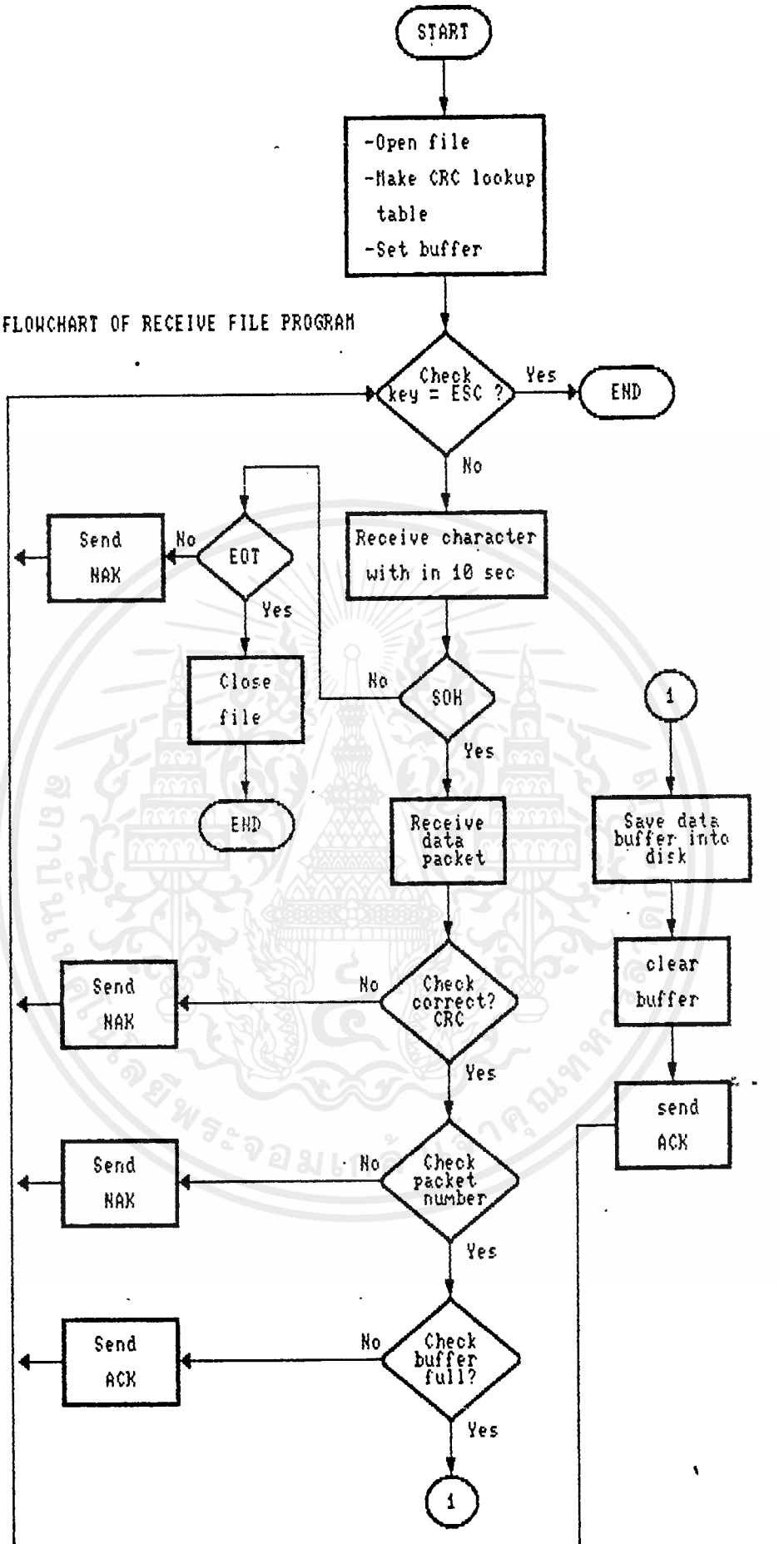
DEL

Delete (DEL) ใช้ในการลบอักขระ สำหรับเครื่องพิมพ์ใช้ใน การลบอักขระที่ได้รับตัวท้ายสุด ถ้าเป็นหน่วยแสดงผลก็เป็นการลบอักขระใต้เคอร์เซอร์ออก

FLOWCHART OF SEND FILE PROGRAM



FLOWCHART OF RECEIVE FILE PROGRAM



บทที่ 3

การลดขนาดข้อมูล

HUFFMAN ALGORITHM

ก่อนที่จะเข้าไปในรายละเอียดของ huffman algorithm จะขอชี้แจงภาพรวม ๆ ของวิธีเก็บข้อมูลภายในเครื่องคอมพิวเตอร์ เพื่อให้เข้าใจหลักการได้ง่าย

เนื่องจากตัวอักษรภาษาอังกฤษ (Text File) มีลักษณะการเก็บในเครื่องคอมพิวเตอร์โดยการแทนด้วยรหัสต่าง ๆ เช่น เอ็บซีดิกหรือแอสกี ซึ่งเครื่องจะเก็บตัวอักษรต่าง ๆ ด้วยรหัสแอสกี 8 บิต ขนาดของตัวอักษร 1 ตัวจะใช้รหัสไบนารีจำนวน 8 บิตแทนได้ และกินเนื้อที่ในการเก็บเท่ากับ 1 ไบต์ จัดเป็นความยาวที่แน่นอนแปรเปลี่ยนไม่ได้ (Fix Length Code) ดังนั้นถ้าขนาดของเท็กไฟล์ใหญ่โตกว่าเดิม ก็ต้องใช้เนื้อที่ในการเก็บมากขึ้นด้วย โดยขนาดของเนื้อที่จะขึ้นอยู่กับจำนวนตัวอักษร หากเราสามารถกำหนดเนื้อที่ที่ใช้เก็บตัวอักษรแต่ละตัว ให้มีขนาดเปลี่ยนแปลงได้ตามความเหมาะสม เช่น ประโยค "For those of you who remain skeptical about the ability of this method to shorten file. Let's take an example the sentence you are now reading " ประโยคนี้ถ้าใช้วิธีปกติจะใช้เนื้อที่จำนวน 147 ไบต์หรือเท่ากับ 147 x 8 = 1176 บิต ลองพิจารณาตัวอักษรในประโยคนี้ จะพบว่ามีอักษรต่างกันเพียง 27 ตัวเท่านั้น เราสามารถแทนตัวอักษรแต่ละตัวด้วยรหัสขนาด 5 บิต ก็เป็นการเพียงพอแล้ว เพราะมันสามารถกระจายตัวอักษรที่ต่างกันได้มากถึง $2^5 = 32$ ตัว เราจะใช้เนื้อที่ในการเก็บเพียง 147 x 5 = 715 บิต วิธีนี้มีประสิทธิภาพพอควร แต่ยังคงใช้ลักษณะการเก็บแบบ Fix Length Code เหมือนเดิม ถ้าเราสามารถเก็บตัวอักษรแต่ละตัวด้วยขนาดของบิตที่ไม่เท่ากันได้ (Variable Length Code) จะช่วยลดขนาดของเนื้อที่เก็บข้อมูลได้อีกมาก วิธีที่พิจารณาว่าตัวอักษรใดควรเก็บด้วยจำนวนบิตเท่าไรนั้น เราสามารถใช้ความถี่ของตัวอักษรที่ปรากฏอยู่ในประโยคมาเป็นหลักเกณฑ์ในการพิจารณา โดยที่ ถ้าหากว่าประโยคนั้นมีความถี่ ของตัวอักษรหลาย ๆ ตัว เราก็นำตัวอักษรนั้นมาแทนด้วยไบนารีโค้ดสั้น ๆ ส่วนตัวอักษรตัวใดปรากฏน้อยครั้ง เราก็แทนด้วยไบนารีโค้ดที่มีขนาดยาวขึ้นหน่อย จากนั้นก็นำไบนารีโค้ดที่มีความยาวแตกต่างกันตามความถี่ของตัวอักษร มาเรียงต่อกันทำให้ประหยัดเนื้อที่ที่ใช้เก็บข้อมูลได้มากกว่า ซึ่งวิธีการนี้คือแนวคิดของ Huffman Algorithm จากการใช้หลักการนี้ทดสอบกับประโยคดังกล่าวนี้จะเหลือเพียง 612 บิต(เท่ากับ 612/8 ประมาณ 77 ไบต์) เท่านั้นการแทนตัวอักษรดังกล่าวได้แสดงไว้ในตาราง ที่ 1

การเก็บข้อมูลแบบ Variable Length Code เราพบว่ามีปัญหาใหญ่ ๆ เกิดขึ้น 3 ประการคือ

1. Variable Length Code สามารถทำการ Compress Data ได้ก็จริงแต่ในระบบ จะยอมรับเฉพาะการทำงานแบบ Fix Length Code ที่มีขนาด 8 บิตเท่านั้น เราแก้ไขโดยนำบิตของแต่ละตัวอักษรมาเรียงต่อ ๆ กันเมื่อครบ 8 บิต ก็แทนด้วยตัวอักษร 1 ตัว ทำเช่นนี้ไปเรื่อย ๆ จนข้อมูลหมด

2. การเก็บข้อมูลแบบนี้อาจทำลายข้อมูลทั้งหมดได้ง่าย เพราะการนำบิตมาเรียงต่อกัน ถ้าเกิดการ encode ผิดเพียงบิตเดียวจะ ส่งผลให้ Decompress ไม่สำเร็จข้อมูลที่ได้จะขาดความถูกต้อง ซึ่งจุดนี้เป็นเรื่องที่ควรระวังมากที่สุด

3. ปัญหาในการแยกแยะความแตกต่างของข้อมูลแต่ละบิตออกจากกัน ขณะทำการ Decompress

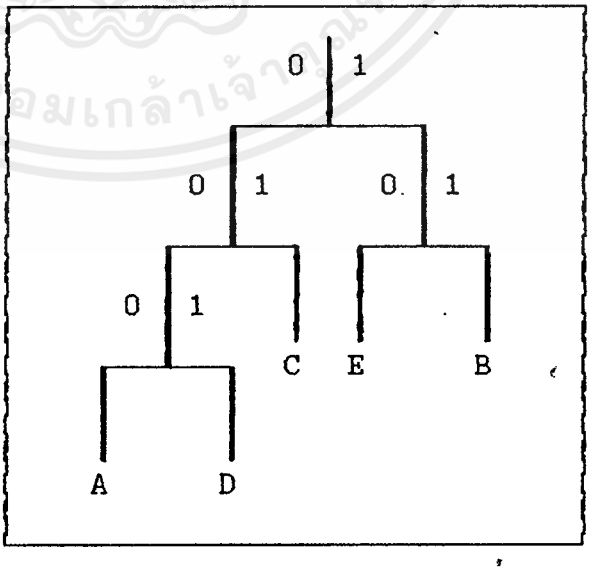
ถ้าพิจารณา Frequency Table ในตารางที่ 1 ซึ่งการ encode อักษรแต่ละตัวจะมีคุณสมบัติ Prefix property ดังที่ได้กล่าวมาแล้ว เราจะพบว่ารหัสที่มีคุณสมบัติ Prefix Property นั้นจะสามารถแทนได้ด้วยไบนารีทรี ซึ่งก็คือ Tree ที่มี Children อยู่เพียง 2 Children ในแต่ละโหนดเท่านั้น โดยเราให้ Left Child แทนด้วยรหัส 0 และ Right Child แทนด้วยรหัส 1 พิจารณาการแทนตัวอักษรในรูปที่ 1

ในรูป 1 เราสามารถแทนตัวอักษรด้วยไบนารีโค้ดโดยการ trace จาก Rootไปยัง Leaf หากเรา trace ไปทาง Left Branch เราก็จะได้ '0' เป็นส่วนหนึ่งของรหัส และถ้า trace ไปทาง Right Branch จะได้ '1' เป็นส่วนหนึ่งของรหัส ด้วยวิธีการนี้เราจะพบว่าไม่มีตัวอักษรใดเลยที่เป็น Prefix ของตัวอักษรอื่น เพราะตัวอักษรทุกตัวจะปรากฏอยู่ที่ Leaf เท่านั้น

ต่อไปจะพิจารณาถึงการสร้างไบนารีทรี ที่จะใช้เป็นส่วนหนึ่งในการทำ Data Compress โดยดูจากอัลกอริทึม ในรูป 2 ซึ่งเริ่มต้นจากการนำคาแรกเตอร์ที่ปรากฏใน Frequency Table มาบรรจุลงใน List ของ Subtree แล้วพิจารณาหาคาแรกเตอร์ที่มีความถี่น้อยที่สุดจำนวน 2 ตัว อาจจะมีความถี่เท่ากันหรือไม่ก็ได้ แล้วนำโหนดทั้งสองมาผนวกเข้าด้วยกันให้เป็น Subtree โดยมี Children เป็นคาแรกเตอร์ของทั้งคู่ assign ผลรวมความถี่ของทั้ง 2 โหนดให้เป็นความถี่ของ Subtree จากนั้นนำ Subtree ที่ได้บรรจุกลับไปยัง List ของ Subtree ต่อจากนั้นให้พิจารณาหา Subtreeที่มีความถี่น้อยที่สุดจำนวน 2 Subtree แล้วนำมาทำการสร้าง Subtree ขึ้นใหม่ โดยยึดหลักการเดิม อัลกอริทึมนี้จะผนวก Subtree จนกระทั่งเหลือ Subtree ที่ใหญ่ที่สุดเพียงชุดเดียวหรือเรียกว่า Huffman Code Tree ดูรูป 3

คาแรกเตอร์	Frequency	รหัส	คาแรกเตอร์	Frequency	รหัส
space	26	101	m	3	111011
e	17	010	w	2	111111
t	13	1000	k	2	111110
o	12	1001	p	2	111100
a	10	1101	b	2	111101
s	8	0111	d	2	011001
i	7	0000	c	2	011000
h	7	0001	x	1	1110101
n	7	0010	g	1	1110100
l	5	1101	F	1	1110010
r	5	11000	'	1	1110011
f	3	01101	,(comma)	1	1110001
y	3	00111	.(period)	1	1110000
u	3	00110			

ตารางที่ 1 แสดงความถี่ของตัวอักษรพร้อมรหัส

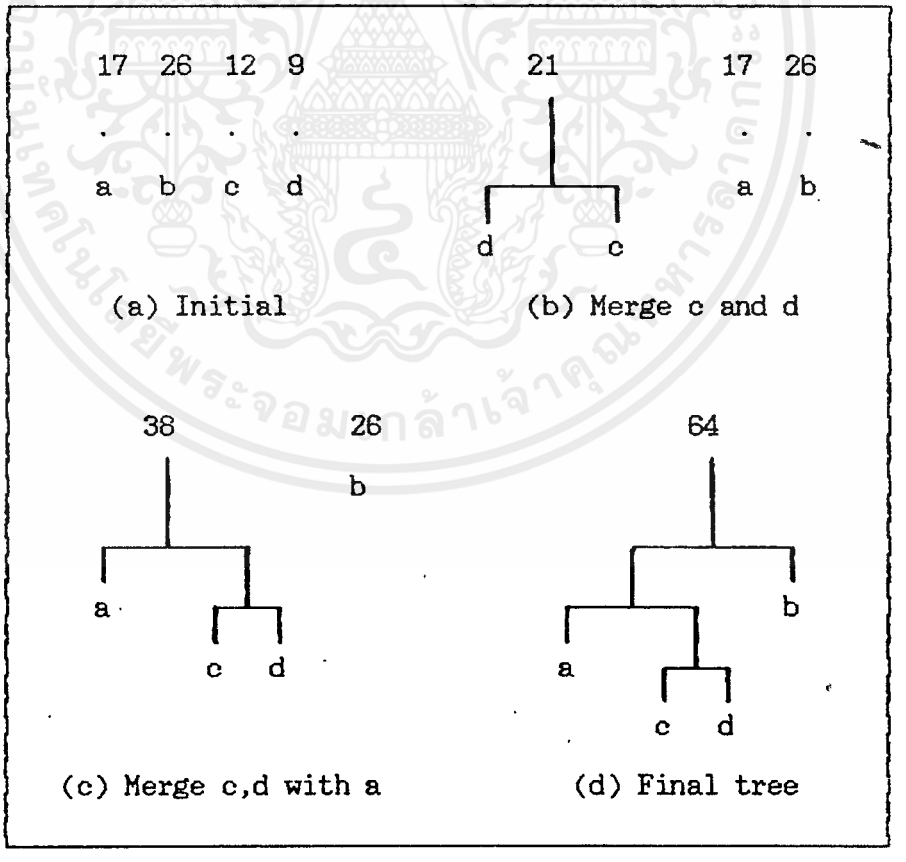


รูปที่ 1 แสดง Simple Code Tree

HUFFMAN ALGORITHM

For each character with a nonzero frequency
add the character to the list of subtree
While the list of subtree contains more than one subtree;
remove from the list the two subtrees
with the smallest frequencies;
make them the children of a new subtree whose frequency
is the sum of their frequencies;
add the new subtree to the list;
The remaining subtree is the Huffman code tree

รูปที่ 2 แสดงฮัฟฟ์แมนอัลกอริทึม



รูปที่ 3 แสดงขั้นตอนการสร้างต้นไม้ Huffman [illegible]

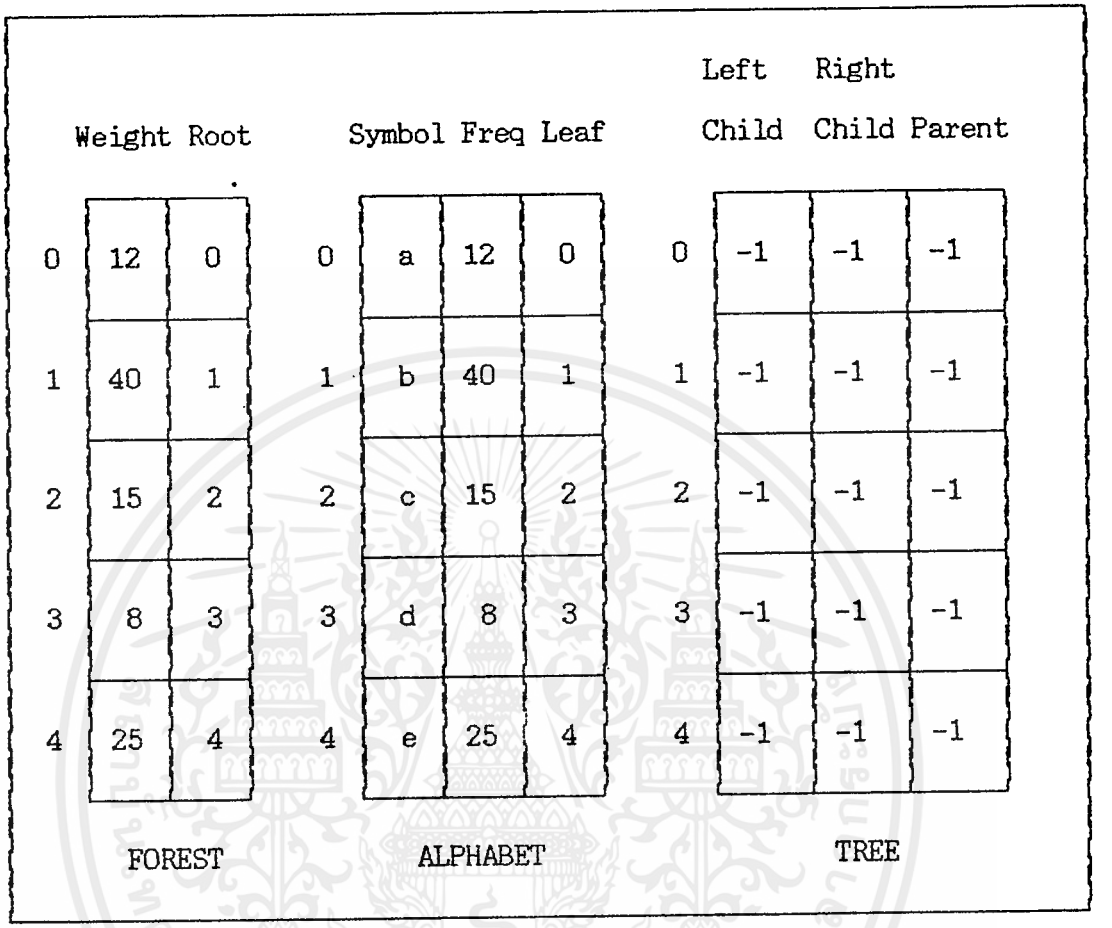
```
(1) While there is more than one tree in the forest do
    begin
(2) i := index of the tree in FOREST with smallest weight;
(3) J := index of the tree in FOREST with second smallest weight;
(4) create a new node with left child FOREST[i].root and
    right child FOREST[j].root;
(5) replace tree i in FOREST by a tree whose root is the new
    node and whose weight is
    FOREST[i].weight+FOREST[j].weight;
(6) delete tree j form FOREST
    end;
```

รูปที่ 4 แสดงโครงร่างคร่าวๆ ของ Huffman tree construction

```
procedure lightones (var least,second:integer);
 { sets least and second to the indices in FOREST of
the tree of smallest weight. We assume lasttree >= 2 ¦ var
i : integer;
begin {initialize least and second, considering first two
       trees }
     if FOREST[1].weight <= FOREST[2].weight then
     begin least := 1; second := 2; end
else
```

```
      begin least := 2; second := 1 end;
  { Now let i run from 3 to lasttree. At each iteration least
    is the tree of smallest weight among the first  i trees
    in FOREST ,and second is the next smallest of these  }
    for i := 3 to lasttree do
         if FOREST[i].weight < FOREST[least].weight then
           begin second := least; least := i end
      else if FOREST[i].weight < FOREST[second].weight then
           second := i
    end { lightones }
function create (lefttree,righttree : integer) : integer;
  { returns new node whose left and right children are
     FOREST[lefttree].root and FOREST[righttree].root }
begin
  lastnode := lastnode + 1;
   { cell for new node is TREE[lastnode] }
  TREE[lastnode].leftchild := FOREST[lefttree].root;
  TREE[lastnode.rightchild := FOREST[righttree].root;
  { now enter parent pointers for new node and its children }
  TREE[lastnode].parent := 0;
  TREE[lastnode].parent := 0;
  TREE[FOREST[lefttree].root].parent := lastnode;
  TREE[FOREST[righttree].root].parent := lastnode;
  return (lastnode)
end; ( create )
```

```
precedure Huffman;
 var
   i,j: integer; { the two trees of least weight in FOREST }
   newroot: integer;
 begin
   while lasttree > 1 do begin
     lightones(i,j);
     newroot := create(i,j);
     { Now replace tree i by the tree whose root is newroot }
     FOREST[i].weight := FOREST[i].weight + FOREST[j].weight;
     FOREST[i].root := newroot;
     { next,replace tree j,which is no longer needed,by
     lasttree, and shrink FOREST by one }
     FOREST[j] := FOREST[lasttree];
     lasttree := lasttree - 1;
 end
end; [Huffman]
```



	Weight	Root
0	12	0
1	40	1
2	15	2
3	8	3
4	25	4

FOREST

	Symbol	Freq	Leaf
0	a	12	0
1	b	40	1
2	c	15	2
3	d	8	3
4	e	25	4

ALPHABET

	Left Child	Right Child	Parent
0	-1	-1	-1
1	-1	-1	-1
2	-1	-1	-1
3	-1	-1	-1
4	-1	-1	-1

TREE

Algorithm Writetree โดยใช้ tree node เป็นอินพุต

ถ้าโหนดที่เข้ามาเป็นลีฟ

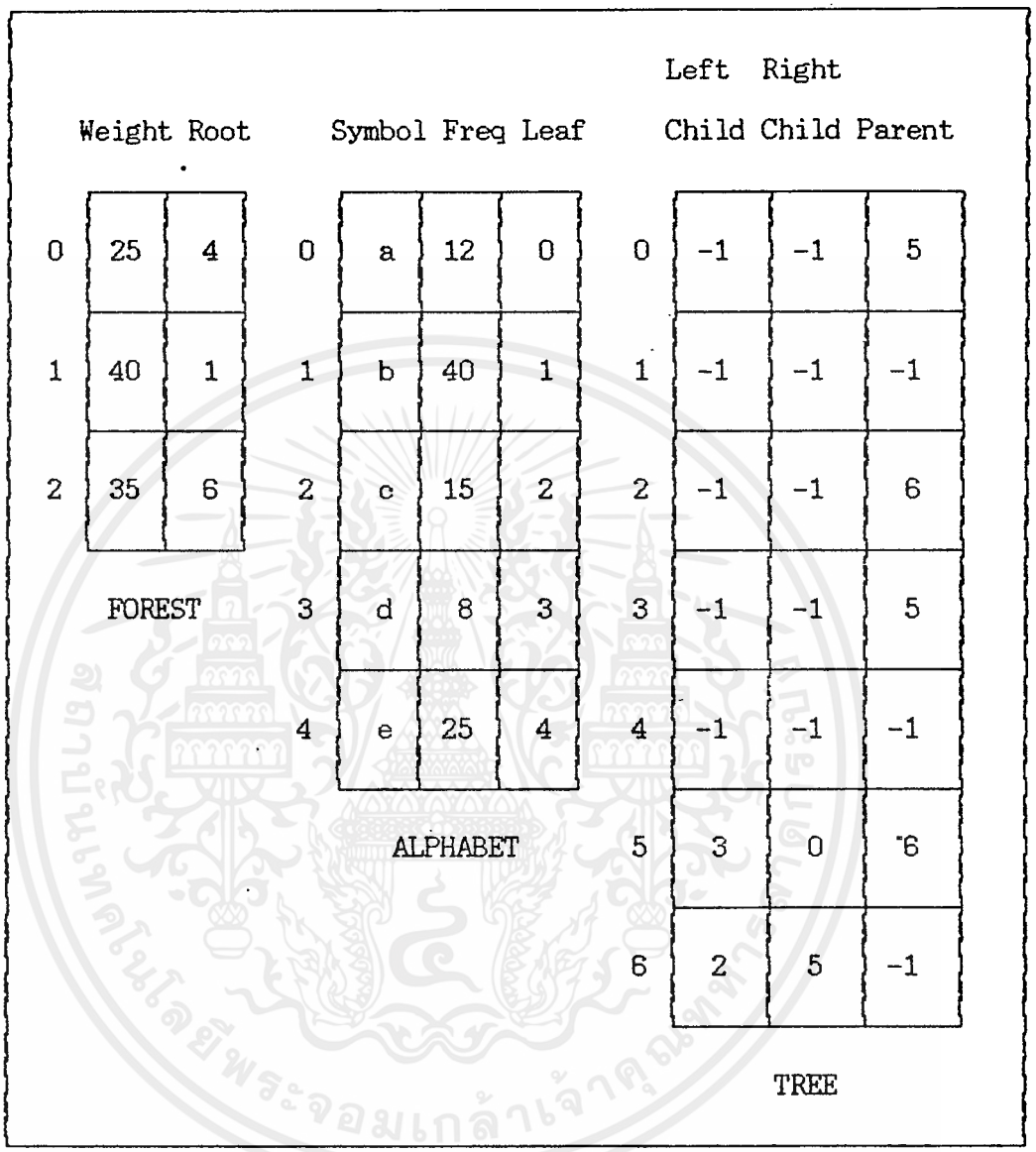
ให้เอาต์พุตเป็น 0 แล้วตามด้วยรหัส 8 บิตของตัวอักษรที่อยู่ที่ลีฟนั้น

แต่ถ้าโหนดที่เข้ามาไม่ใช่ลีฟ

ให้เอาต์พุตเป็น 1

ทำการ Recursive อัลกอริทึมนี้ โดยใช้ Leftchild ในการผ่านค่า

medki Recursive อัลกอริทึมนี้ โดยใช้ Rightchild ในการผ่านค่า



```
Algorithm Readtree จะทำการ Return Tree
อ่านค่าบิตจาก file;
ถ้าหากบิตที่อ่านเข้ามาเป็น 1
   ทำการ Recursive Readtree แล้วให้ค่าเก็บไว้ใน Left;
   ทำการ Recursive Readtree แล้วให้ค่าเก็บไว้ใน Right;
สร้าง Tree ขึ้นมาโดยให้ Left และ Rightchild เป็น left และ Right ตามลำดับ
   ให้ Symbol ของ Tree นั้นเป็น Null character(00H);
   Return(Tree)
ถ้าหากบิตที่อ่านเข้ามาเป็น 0 จะทำการอ่านค่า 8 บิตถัดไปแล้วนำมาแปลงกลับเป็นตัวอักษร;
สร้าง Tree ขึ้นมาโดยให้ Leftchild และ Rightchild เป็น NULL;
ให้ Symbol ของ Tree เป็นตัวอักษรตัวที่ได้จากการแปลง;
Return(Tree);
```

ข้อสังเกต ฮัฟฟ์แมนนี้ไม่มีหลักการที่แน่นอนในการผนวก Subtree เข้าด้วยกัน หรือบ่งบอกว่าควรจะเอา Subtree ใดเป็น Leftchild และ/หรือ Subtree ใดเป็น Right Child ดังนั้นการ compress ด้วยวิธีนี้จึงอาศัยความถี่ของตัวอักษร แต่บางกรณีถ้าเรา commpress เป็น Word โดยใช้อัลกอริทึมเดิม เราสามารถขนาดข้อมูลได้มากขึ้นกว่าการ compress เป็นตัวอักษร เช่น ประโยค "John where Bill had had had had had had had " เราแทนคำว่า had ด้วย 1 แทนBill ด้วย 01 แทน John ด้วย 000 และแทน where ด้วย 001 เราจะพบว่าขนาดของข้อมูลจะเล็กลงไป อีกทว่าในความเป็นจริง เราไม่สามารถทำได้ดังตัวอย่างที่ยกมา เพราะปกติแล้วการเก็บไบนารีรหัสต่าง ๆ ในคอม จะใช้รหัสขนาด 8 บิต ซึ่งสามารถทำการ represent ค่าแตกต่างกันได้ 256 Words ไม่เพียงพอที่จะใช้ในการทำงานจริง และแม้นว่าเราจะฝืนใช้หลักการนี้ เราพบว่าจำเป็นต้องแทน word ต่าง ๆด้วยรหัสที่มีขนาดมากกว่า 8 บิตอย่างแน่นอน ทำให้ขนาดของข้อมูลที่ผ่านการ compress แล้วใหญ่โตขึ้น ซึ่งผิดจากแนวความคิดของการ Compression ที่ต้องการให้ขนาดของข้อมูลเล็กลง

จากนี้เราจะพูดถึงการ Implement ฮัฟฟ์แมนอัลกอริทึมให้เป็นโปรแกรมที่สามารถทำงานได้จริง สิ่งแรกที่ต้องทำคือ เราต้องมี FOREST (หมายถึง List ของ Subtree ที่จะนำมา combine กัน)

โดยที่ FOREST จะเก็บ(ความถี่) และพอยน์เตอร์ที่ชี้ไปยัง Subtree นั้น นอกจากนี้เรายังต้องสร้าง TREE ขึ้นเพื่อใช้ในการเก็บ Huffman Code Tree และสิ่งที่ขาดไม่ได้คือเราจะต้องสร้าง ALPHABET เพื่อใช้ในการเก็บตัวอักษร และพอยน์เตอร์ที่ชี้มายังตัวของมันเองให้เป็นประโยชน์ในการอ้างอิงภายหลัง ดังนั้นโปรแกรมที่ผู้เขียนสร้างขึ้นจึงประกอบด้วยเรคอร์ดหลัก ๆ 3 เรคอร์ดคือ

1. TREE เรคอร์ดของ Tree ซึ่งจะเป็นตัวเก็บ Tree ที่มีขนาดโตขึ้นเรื่อยๆ จากการผนวก Sub tree จนในที่สุดจะได้

2. ALPHABET เรคอร์ดของ Alphabet ซึ่งจะใช้ในการอ้างอิงต่อไป

3. FOREST เรคอร์ดของ Forest จะเป็นตัวที่ใช้เก็บ List ของ Subtree

การทำงานของโปรแกรมจะเริ่มต้นจากการอ่านซอร์สไฟล์ที่ต้องการ compress โดยอ่านเข้ามาทีละคาแรกเตอร์แล้วสร้าง Frequency ขึ้นมาซึ่งตารางนี้ก็คืออะเรย์ของความถี่ที่สามารถอินเด็กซ์ได้ด้วยตัวอักษรนั้นหลังจากนั้นเราจะนำอะเรย์ของ ALPHABET มาบรรจุลงในอะเรย์ของ ALPHABET เพื่อใช้อ้างอิงในภายหลังและบรรจุลงในอะเรย์ของ FOREST ซึ่งการบรรจุลงในอะเรย์ของ FOREST นี้ก็เป็นการinitial ค่าโหนดเริ่มต้น (Leaf) นั้นเอง

โปรแกรมจะทำการหาโหนดที่น้อยที่สุดจำนวน 2 โหนดมา combine กัน โดยจะสร้าง Subtree ขึ้นมาใหม่ให้มี Childrenเป็น 2 โหนดโดยให้มี Leftchild เป็นโหนดที่มี Weight (ความถี่) น้อยกว่าโหนดที่เป็น Rightchild แล้ว assign Weight ให้กับ Subtree เสียใหม่ โดยการนำเอา Weight จากโหนดทั้งสองมารวมกันแล้วบรรจุ Subtree ใหม่นี้กลับคืนไปให้ FOREST

โปรแกรมจะสร้าง Subtree ที่มีลักษณะเหมือน Subtree ที่บรรจุใน FOREST ลงใน TREE ด้วยจากนั้นโปรแกรมจะทำงานในลักษณะนี้ ตั้งแต่ต้นจนจบ เมื่อ Subtree ถูก combine หมดจะได้ Huff Code Tree ลองพิจารณาโครงร่างการทำงานอย่างคร่าวๆในรูปที่ 4,5 และ 6 รูปที่ 7 และ 8 แสดงโครงสร้างข้อมูลหลังจากการทำงานตามอัลกอริทึมของฮัฟฟ์แมนผ่านไปสองรอบ จากรูปที่ 5 ใช้อินทีเจอร์แทนการใช้พอยน์เตอร์ เพื่อให้เกิดความเข้าใจได้ง่ายขึ้น

รายละเอียดของโปรแกรม Compression นั้นจำเป็นต้องมีโปรแกรมหลัก 2 โปรแกรมคือ โปรแกรม compress ทำหน้าที่ encoding และโปรแกรม decompress หรือ decoding โดยที่อินพุตของโปรแกรม Encoding จะเป็น Textfile ทั่ว ๆไป เมื่อผ่านการ compress แล้วจะได้ output เป็น Encoding ไฟล์ ซึ่งจะต้องนำมาใช้เป็น input ของโปรแกรม Decoding ในภายหลัง

ความยุ่งยากของโปรแกรมคือเราต้องหาวิธีเก็บ Huffman Code Tree ลงในดิสก์ให้ได้ในรูปของรหัสไบนารี 8 บิตเพื่อใช้ในการ decode ต่อไป ลองพิจารณาอัลกอริทึม ในรูปที่ 9 ที่ใช้ในการ Write Tree ให้อยู่ในรูปของไบนารีโค้ดเพื่อการบันทึกลงดิสก์ได้

สร้างโปรแกรมเข้ารหัส

โปรแกรมจะเริ่มต้นจากการอ่านเท็กซ์ไฟล์แล้วทำการสร้าง Frequency Table ซึ่งเป็นอะเรย์ของ Integer Indexed ที่แสดงด้วยคาแรกเตอร์ พร้อมทั้งเก็บจำนวนตัวอักษรทั้งหมดที่อยู่ในเท็กซ์ไฟล์นี้ไว้ เพื่อจะได้ใช้งานในการ Decode ส่วน Frequency Table จะนำไปใช้ในการสร้าง Huffman Code Tree ตามอัลกอริทึมของฮัฟฟ์แมน หลังจากนั้น โปรแกรมจะทำการ represent แต่ละคาแรกเตอร์ด้วยรหัสที่ได้จากการ travel ไปใน Branch ต่างๆ โดยโปรแกรมจะเริ่มต้น travel ที่ Root ออกไปยัง Left Branch และ Right Branch ตามลำดับ หากเป็นการ travel ไปทางซ้าย โปรแกรมจะให้เอาต์พุทเป็น 0 ถ้าโปรแกรม travel ไปทางขวาจะให้เอาต์พุทเป็น 1 วิธีนี้เราจะได้รหัสที่มีขนาดบิตต่างกัน สามารถใช้แทนตัวอักษรต่างๆ ได้ เราก็จะเก็บรหัสของตัวอักษรเหล่านี้ไว้ในอะเรย์เพื่อใช้ในการ encode จากนั้นโปรแกรมจะทำการเปิดเอาต์พุทไฟล์ เริ่มต้นด้วยการบรรจุจำนวนตัวอักษรทั้งหมดที่อยู่ในเท็กซ์ไฟล์ที่ได้เก็บไว้โดยใช้เนื้อที่จำนวน 2 ไบต์ สามารถเก็บขนาดของอินพุทไฟล์ที่ใหญ่สุดได้ถึง 65535 ไบต์ โปรแกรมจะทำการเก็บ Huffman Code Tree ที่ผ่านอัลกอริทึม Writetree แล้ว ซึ่งจะมีรูปแบบเป็นคาแรกเตอร์หลายๆ ตัวเรียงกัน โดยจำนวนคาแรกเตอร์จะขึ้นอยู่กับขนาดของ Huffman Code Tree นั่นเอง

เมื่อเสร็จสิ้นการ write Heading ของเอาต์พุตแล้วจะเป็นการ write ข้อมูลจริงซึ่งโปรแกรมจะเริ่มอ่านอินพุทไฟล์(เท็กซ์ไฟล์) อีกครั้งทีละตัวอักษร แล้วนำมาเปลี่ยนเป็นรหัสที่ได้เตรียมไว้ แล้วนำรหัสเหล่านั้นมาเรียงต่อกัน (อย่าลืมว่ารหัสเหล่านี้อยู่ในรูปของบิต) เมื่อครบ 8 บิตจะทำการ write ลงเอาต์พุตไฟล์ โปรแกรมจะทำงานในลักษณะนี้จนกว่าตัวอักษรในอินพุทไฟล์จะหมด เราก็จะได้ไฟล์ที่ผ่านการ compress แล้ว

ถอดรหัสโปรแกรม

วิธีการถอดรหัสเริ่มต้นด้วยการอ่านจำนวนเท็กซ์ไฟล์จากตัวอักษร 2 ตัว (ซึ่งก็คือ 2 ไบต์แรกใน Heading ของ Compressed ไฟล์) จากนั้นโปรแกรมจะอ่านตัวอักษรต่อไป แล้วนำมา converse กลับอีกครั้งให้เป็น Huffman Code Tree สำหรับขั้นตอนในการ reconstruct Huffman Code Tree จากตัวอักษรที่อ่านเข้ามา ได้แสดงเป็นอัลกอริทึมไว้ในรูปที่ 10

เมื่อเราจัดการ reconstruct เสร็จเรียบร้อยโปรแกรมจะอ่านบิตถัดไปซึ่งก็คือส่วนหนึ่งของรหัสแทนตัวอักษรที่ compress ไว้แล้ว โปรแกรมจะคอยตรวจสอบว่าบิตที่อ่านเข้ามาเป็น 0 หรือ 1 ถ้าเป็น 0 โปรแกรมจะทำการ travel ไปทางซ้าย หรือ travel ไปทางขวาเมื่อบิตที่ได้เป็น 1 โปรแกรมจะอ่านและตรวจสอบเช่นนี้ไปเรื่อยๆ จนกระทั่งถึง Leaf ของ Tree นั้น เพื่อให้ได้ตัวอักษรออกมา 1 ตัว

โปรแกรมจะทำงานในลักษณะเดิม พร้อมทั้งนับจำนวนตัวอักษรที่ได้ออกมาว่ามีค่าเท่ากับเท็กซ์ไฟล์ที่เก็บไว้ตั้งแต่แรกหรือยัง

ปัญหาที่เกิดขึ้นกับโปรแกรมคือ โดยปกติเมื่อทำการ compress ไฟล์แล้วจะได้ไบนารีโค้ดขึ้นมา แต่เราไม่สามารถรู้ได้เลยว่า ไบนารีโค้ดนี้ใช้แอสกีโค้ดอะไร ซึ่งในการทำงานจริงย่อมมีโอกาสที่จะได้รหัส End-of-file ดังนั้นในการ decompress เมื่อโปรแกรมอ่านตัวอักษรแล้วพบรหัสจบแฟ้มจะทำให้โปรแกรมเกิดการ terminate การ Decompress จึงไม่สมบูรณ์

การแก้ปัญหาจะพิจารณาจากแนวความคิดเดิม นั่นคือ โปรแกรมจะจัดการ encode หรือ decode กับข้อมูลที่มีจำนวนครบ 8 บิตเท่านั้น บิตต่างๆ เหล่านี้ล้วนเกิดจาก Information ของ input data ทั้งสิ้นแต่ถ้าเราดัดแปลงความคิดนี้ โดยการให้ encode หรือ decode กับบิตที่มีขนาดเพียง 7 บิต แล้วทำการ initial ค่าให้กับบิตซ้ายสุดด้วย 1 ดังนั้นช่วงของ Compressed Character ที่ได้จะอยู่ในช่วงระหว่าง 80H จนถึง 255H (10000000 ถึง 11111111) ด้วยวิธีการนี้จะทำให้ Encode Character ไม่มีโอกาสเป็นรหัส End-of-file เลย ทั้งนี้เนื่องจากรหัส End-fo-file มีค่าเป็น 1AH (00011010) วิธีการนี้จะทำให้ Compressed ไฟล์มีขนาดใหญ่ขึ้นกว่าเดิมอีกเล็กน้อย ส่วนหลักการทำงานของการ Compress และ Decompress โปรแกรมยังคงใช้หลักการเดิมอยู่

LZW

Jacob Ziv และ Abraham Lempel ได้เสนอวิธีการลดขนาดข้อมูลแบบพจนานุกรมไว้ 2 วิธีคือ LZ1 และ LZ2 ในปี ค.ศ.1984 วิธี LZ2 ก็ได้ถูกดัดแปลงโดย Terry A.Welch จนกลายมาเป็นวิธี LZW (Lempel Ziv Welch) ส่วนสำคัญของอัลกอริทึม LZW คือตารางรหัส (String table) หรือ (Translation table) รหัสแต่ละตัวในตารางจะประกอบด้วย ส่วนสตริงนำหน้า (Prefix string หรือ w) และส่วนตัวอักษร (extension character หรือ k)

ข้อมูลแต่ละตัวอักษรจะถูกพิจารณาตามลำดับ เพื่อหาสตริงที่ยาวที่สุดที่ตรงกับสตริงในตาราง จนกระทั่งไม่สามารถหาได้จึงทำการเพิ่มสตริงที่มาใหม่นั้นเข้าไว้ในตาราง แล้วส่งรหัสของสตริงที่ยาวที่สุดนั้นออกเอาต์พุต เมื่อทำขั้นตอนที่ว่านี้จนจบ ก็จะได้ข้อมูลที่ลดขนาดแล้ว

ขั้นตอนที่กล่าวมาข้างต้น สามารถเขียนอยู่ในรูปภาษาเทียมได้ดังนี้

```
initialize table
read first input character -> prefix string or w
step: read next input character K
```

```
if no such K (end of input); code(w) ->output; EXIT
if wK exists in string table; wK -> w;
else: wK -> string table;
      code(w) -> output;
      K -> w;
loop Step.
```

จะเห็นว่าทุก ๆ รอบการทำงาน สตริงนำหน้า w จะมีการเปลี่ยนแปลง ถ้าสตริงนำหน้ารวมกับตัวอักษรเป็นสตริง wK มีอยู่ในตารางรหัส สตริงนำหน้าก็จะยาวขึ้นอีก 1 ตัวอักษร (เพิ่มอักษร K เข้าไป) แต่ถ้า wK ไม่อยู่ในตาราง ก็จะเพิ่มสตริงใหม่เข้าไปไว้ในตาราง แล้วส่งรหัสของสตริงนำหน้าออกทางเอาต์พุต สตริงนำหน้าก็จะเริ่มต้นใหม่มีค่าเป็น K ยาวเพียง 1 ตัวอักษร

ตัวอย่างการทำงานของอัลกอริทึมเป็นดังรูปที่ 1 ซึ่งสมมติว่ามีตัวอักษรในชุดรหัสเพียง 3 ตัวคือ a,b,c

รูปที่ 1

ตัวอักษรที่เข้ามา	a	b	a	b	c	b	a	b	a	b	a	a	a	a	a	a
		4		6		8				10				12		
สตริงที่เพิ่มเข้าไว้ในตาราง	—		——		—				—		———					
		—		—		———				———						
			5		7		9				11					
รหัสที่ออกเอาต์พุต	-	-	—	-	—	——	—	—	———	—						
	1	2	4	3	5	8	1	10	11							

ตารางที่ 1

ตารางรหัส		ตารางในรูปแบบ (w,K)	
1	a	1	a
2	b	2	b
3	c	3	c
4	ab	4	1b
5	ba	5	2a
6	abc	6	4c
7	cb	7	3b
8	bab	8	5b
9	baba	9	8a
10	aa	10	1a
11	aaa	11	10a
12	aaaa	12	11a

จากรูป 1 ตัวอักษรถูกอ่านเข้ามาจากทางซ้ายไปขวา เริ่มต้นที่การอ่านตัว "a" เข้ามาแล้วกำหนดค่าให้แก่ w และตัวถัดมาคือ "b" กำหนดค่าให้แก่ K เนื่องจาก wK ซึ่งเท่ากับ "ab" ไม่มีในตารางรหัส (ในตอนแรกตารางมีเพียง 3 ตัวคือ a,b,c) จึงเพิ่ม "ab" เข้าไว้ในตารางเป็นรหัสหมายเลข 4 และส่ง w คือรหัสหมายเลข 1 ออกทางเอาต์พุต ต่อจากนั้นจึงกำหนดให้ w มีค่าเป็น K ซึ่งเท่ากับ "b"

w <..."a"

K <..."b"

ดังนั้น wK == "ab" ซึ่งไม่มีในตาราง ให้กลับไปเพิ่มค่ารหัสในตาราง หรือรหัสที่ 4 (ในการลดข้อมูลจริง ๆ ที่เป็น ASCII ชนิด 8 บิต ซึ่งมีรหัสอักษร 256 ตัว เราก็จะเพิ่มเข้าไปเป็นรหัสที่ 257 แทนหมายเลข 4)

output <... "1" ส่งรหัสหมายเลข 1 หรืออักษร นั้นเอง ออกไปยังเอาต์พุต

w <... K ถึงตรงนี้ ตัวแปร ก็จะเก็บค่าเอาไว้แทนแล้ว

ต่อไปเป็นการอ่านตัวอักษร "a" (ซึ่งเป็นอยู่เป็นตัวที่ 3 จากซ้ายมือ) เข้ามาเก็บไว้ในตัวแปร K ในรอบนี้ wK คือ "ba" ก็ไม่มีในตารางจึงเพิ่มเข้าไปเป็นรหัสอักษรหมายเลข 5 ส่งรหัสอักษรหมายเลข 2 ออกทางเอาต์พุต กำหนดให้ w มีค่าเท่ากับ "a"

จากนั้นอ่าน "b" (ตัวที่ 4 จากซ้ายมือ) เข้ามากำหนดค่าให้กับ K เนื่องจาก wK ในรอบนี้คือ "ab" มีอยู่ในตารางเป็นรหัสหมายเลข 4 รอบนี้จึงเพียงกำหนดค่าให้ w เท่ากับ "ab" ไปเลย แล้วส่งรหัสอักษรหมายเลข 4 นี้ออกสู่เอาต์พุต จากนั้นจึงอ่าน "c" เข้ามาเก็บในตัว K และเนื่องจากค่า wK ตัวใหม่นี้เป็นอักษร "abc" ซึ่งยังไม่มีอยู่ในตาราง ดังนั้นจึงทำการเพิ่มรหัสอักษรเข้าไปในตาราง ซึ่งจะเป็นหมายเลข 6 นั้นเอง แล้ววกกลับไปอ่านข้อมูลตัวต่อไป ทำเช่นนี้ไปเรื่อยจนหมดข้อมูล

จะเห็นว่าตัวอัลกอริทึมนั้น ประกอบด้วยขั้นตอนเพียงไม่กี่ขั้นตอนเท่านั้น ขั้นตอนที่ยุ่งยากที่สุดน่าจะเป็นการหาว่า wK มีอยู่ในตารางหรือไม่ จากตารางที่ 1 จะเห็นว่าเราสามารถจัดรหัสในตารางให้เหมาะสำหรับการค้นหาได้ โดยแยกรหัสออกเป็นหมายเลขรหัสของสตริงนำหน้า และตัวอักษร (w,K) แทนที่จะเก็บรหัสเป็นสตริงที่มีขนาดไม่แน่นอน

การขยายกลับข้อมูล (Decompression)

ข้อดีของ LZW คือ ไม่ต้องมีการเก็บ "ตารางรหัส" ไว้ในข้อมูลเพื่อใช้ในการขยายกลับทั้งนี้เพราะตัวขยายกลับ (decompressor) จะสามารถสร้างตารางรหัสได้เอง จากข้อมูลที่มันจะทำการขยายนั่นเอง โดยตารางรหัสนั้นก็จะเป็นตารางเดียวกันกับตอนทำการลดขนาด

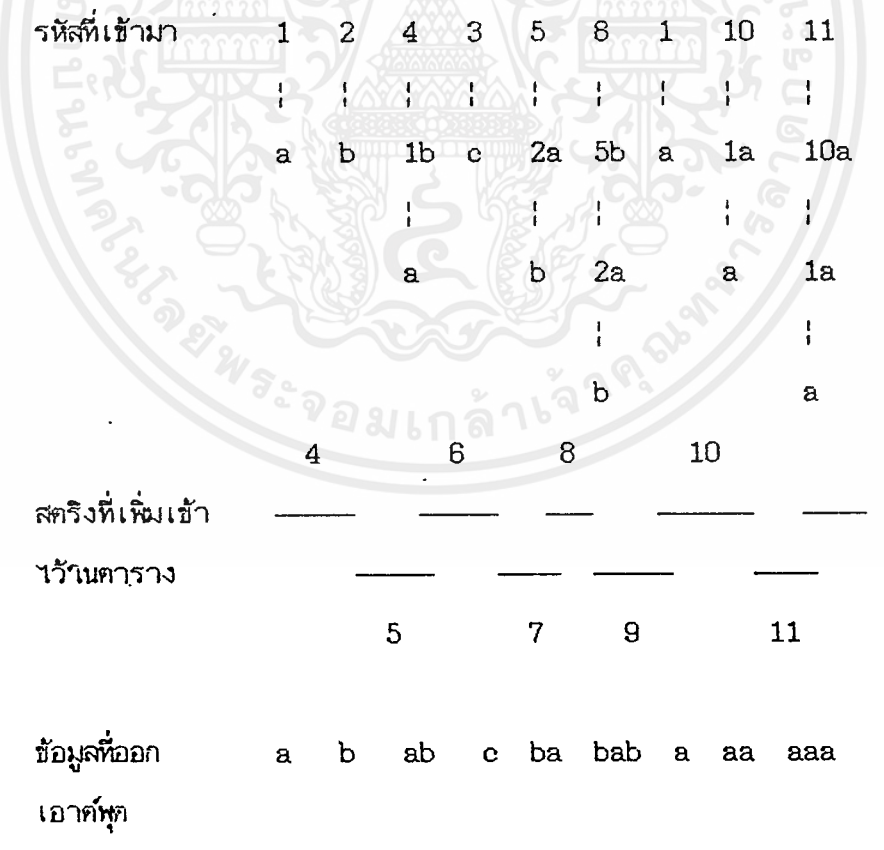
เมื่อถอดรหัสที่เข้ามาจาก wK จนเหลือเพียง K K ที่ได้จะนำไปรวมกับรหัสที่เข้ามาก่อนหน้า (wK) กลายเป็นรหัสตัวใหม่ในตารางรหัส (wK)K อัลกอริทึมในเบื้องต้นสามารถเขียนได้ดังนี้

```
Decompression read first input code -> CODE -> OLDcode;
              with CODE = code(K), K -> output;
Next Code:    read next input code -> CODE -> INcode;
              if no new code; EXIT.
Next Symbol:  if CODE-code(wK) ; K -> output;
                                  code(w) -> CODE;
                                  repeat Next Symbol
```

```
if CODE = code(K);
        K -> output;
        code(OLDcode,K)-> string table;
        INcode -> OLDcode;
        repeat Next Code.
```

ตัวอัลกอริทึมข้างต้นนี้มีปัญหาอยู่ 2 ประการคือ การถอดรหัสจะได้ตัวอักษรในลำดับกลับหลัง เช่น รหัส "cab" (มองในรูป wK คือ "ca""b")จะถอดได้เป็น "bac" ทั้งนี้เพราะ K ที่ได้ในแต่ละครั้งจะเป็นตัวอักษรท้ายสุดของรหัส การแก้ปัญหาทำได้โดยการใช้ stack เก็บตัวอักษรที่ได้แล้วจึงส่งออกเอาต์พุตในตอนหลัง และปัญหาต่อมาคือ ในบางกรณีการสร้างรหัสจะไม่ทันการอ้างถึงรหัส พิจารณาตัวอย่างในรูปที่ 2

รูปที่ 2



ปัญหาจะเกิดขึ้นเมื่อรหัส 8 เข้ามาเพราะในตารางเพิ่งมีถึงรหัส 7 เท่านั้น ปัญหานี้จะเกิดขึ้นเมื่อรหัสเข้ามาเป็นลำดับ KwKwK อย่างไรก็ตามเราสามารถสร้างรหัส 8 ได้ถ้ารู้ว่า K สุดท้าย (อักษรตัวแรกสุดของรหัส) ที่จะได้จากรหัส 8 นี้คืออะไร ซึ่งจากการพิจารณาจะเห็นว่าในกรณีที่เกิดปัญหาเช่นนี้ขึ้น K สุดท้ายของรหัสที่ต้องการก็คือ K สุดท้ายของรหัสที่เข้ามาก่อนหน้านั่นเอง ในที่นี้ K สุดท้ายของรหัส 8 จะเท่ากับ K สุดท้ายของรหัส 5 คือ "b" ดังนั้นการแก้ปัญหาจึงทำได้โดยการเก็บ K สุดท้าย ของรหัสก่อนหน้าไว้ทุกครั้ง เพื่อนำมาใช้สร้างรหัสเมื่อเกิดกรณีพิเศษดังกล่าวขึ้น

อัลกอริทึมที่ใช้งานได้จริง เป็นดังนี้

```
Decompression: read first input code -> CODE -> OLDcode;
               with CODE = code(K),  K-> output;
                                     K-> Finchar;
Next Code      reac next input code -> CODE -> INcode;
               if no new code; EXIT.
               if CODE not defined (special case);
                 Finchar -> output;
                 OLDcode -> CODE;
                 code(OLDcode,FINchar) -> incode;
Next Symbol    if CODE = code(wK): K -> stack;
                                   code(w) -> CODE;
                                   repeat Next Symbol
               if CODE = code(K);
                         K -> output;
                         K -> Finchar;
                         do while stack not empty;
                            stack top -> output; POP stack;
                         code(OLDcode,K) -> string table;
                         INcode -> OLDcode;
                         repeat Next Code.
```

เนื่องจากรหัสที่เข้ามาสามารถนำไปอ้างถึงตารางได้โดยตรง อัลกอริทึมของการขยายกลับจึงมีประสิทธิภาพดีกว่าอัลกอริทึมของการลดขนาดข้อมูล

การพัฒนาโปรแกรม LZW

การพัฒนาโปรแกรมจากอัลกอริทึม LZW โดยให้ความสำคัญกับเรื่องความสามารถในการย้ายระบบหรือ Portability เป็นอันดับแรก เพื่อให้สามารถใช้ได้กับเครื่องขนาดใดก็ได้ ตั้งแต่ไมโครคอมพิวเตอร์ จนถึงเมนเฟรมคอมพิวเตอร์ และข้อมูลที่ผ่านการลดขนาดนั้นสามารถนำมาขยายกลับในเครื่องใดก็ได้ไม่ว่าข้อมูลนั้นจะได้จากเครื่องใด ดังนั้นโปรแกรมที่ได้นี้จึงมีข้อจำกัดอยู่บางส่วน เช่น โปรแกรมเป็นภาษา C ทั้งหมดเพราะมีความสามารถในการย้ายระบบสูง,จำนวนบิตของรหัสไม่เกิน 13 บิต เพื่อให้ใช้ได้กับเครื่องระดับไมโครฯ, การเขียนและอ่านข้อมูลทีละ Word (2 ไบต์) ซึ่งช่วยให้โปรแกรมทำงานเร็วขึ้นนั้น อาจทำให้เกิดปัญหาเพราะบางเครื่องมีการสลับไบต์ จึงต้องอ่านและเขียนทีละไบต์

โปรแกรมนี้ได้ผ่านการทดสอบบนเครื่อง Cyber ซึ่งเป็นระดับเมนเฟรม เครื่องมินิรุ่น CD 4000, เวิร์คสเตชันของ SUN และ PC โดยสามารถคอมไพล์และทดสอบรันโดยไม่ต้องแก้ไขใดๆ

โปรแกรมใช้ตัวแปร *symtbl เป็นที่เก็บตารางรหัส ซึ่งแต่ละรหัสประกอบด้วยรหัสนำหน้า (W) และตัวอักษร (K) ขนาดของรหัส (bitsiz) จะเพิ่มจาก 9 บิตขึ้นไปถึง 13 บิต เมื่อรหัส (tblcnt) เพิ่มขึ้นถึงค่าสูงสุด (tblmax) ของ bitsiz นั้นๆ Compressor จะส่งรหัส tblmax ไปยังเอาต์พุต เป็นการบอกถึงการเพิ่ม bitsiz และเพิ่มค่า tblmax,bitsiz ขึ้นไปจนกระทั่งตารางรหัสเต็มหรือ tblmax มีค่ามากกว่า MaxTblIdx (สำหรับ 13 บิตคือ 0x1FFF,8192) ก็จะมีการลบตารางเก่าทิ้งทั้งหมดแล้วกลับมาใช้ bitsiz ขนาด 9 บิตกันใหม่ ทำเช่นนี้จนกระทั่งสิ้นสุดข้อความ

ดังที่ได้กล่าวไว้ในตอนท้ายอัลกอริทึมการลดขนาดข้อมูลว่า ขั้นตอนที่ยุ่งยากที่สุดคือการหาว่ารหัส ฯ หนึ่งมีอยู่ในตารางหรือไม่ ในที่นี้ใช้วิธี Hashing โดยเพิ่มตัวแปร *symidx เป็นที่เก็บ index ของ symtbl อีกที เมื่อนำ w และ K เข้า hashing function จะได้ค่าฯ หนึ่ง

สมมติว่าเป็น h &symidx[h] -> idx ที่ได้จึงจะเป็น index ของ symtbl แต่เนื่องจากอาจเกิด hashing collision หรือกรณีที่ $(wK)_1$ และ $(wK)_2$ ให้ค่า h เดียวกัน symidx จึงต้องเป็น link list เพื่อให้สามารถเก็บ index ทั้งหมดไว้ใน symidx[h] ได้

หากเทียบกับโปรแกรมทางด้านลดขนาดข้อมูลที่ใช้กันอยู่ เช่น PKZIP แล้ว โปรแกรมที่ได้ก็ยังถือว่าทำงานได้ช้า อย่างไรก็ตามเราอาจปรับปรุงโปรแกรมได้อีกหลายทาง เป็นต้นว่าการใช้ trie (tree ชนิดหนึ่ง) แทนการใช้วิธี Hashing ในการเก็บและค้นหา index ของตารางรหัส หรือหา hashing function ที่เหมาะสมยิ่งขึ้นเพื่อลดปัญหา hashing collision ทั้งนี้เพราะ hashing collision

ทำให้การเก็บและค้นหาข้อมูลช้าลงมาก อีกวิธีหนึ่งซึ่งน่าจะช่วยด้านความเร็วได้มากคือการใช้ภาษาแอสแซมบลีในรูทีนสำคัญ หรือทั้งโปรแกรมแต่วิธีนี้ก็จะทำให้โปรแกรมขาด portability ไป

ส่วนการปรับปรุงด้านการลดขนาดข้อมูลนั้น ทำได้โดยการตรวจสอบเมื่อตารางรหัสเต็มว่าควรจะลบตารางเดิมทิ้งทั้งหมด หรือลบเพียงบางส่วนหรือใช้ตารางเดิมไปก่อน ทั้งนี้โดยการอัตราการลดขนาด (Compression ratio) ไว้ตลอดการทำงาน อาจจะรอจนกระทั่งอัตราการลดขนาด เริ่มต่ำลงจึงลบตารางรหัส

ตัวอย่าง แสดงการทำงานของอัลกอริทึม Compression

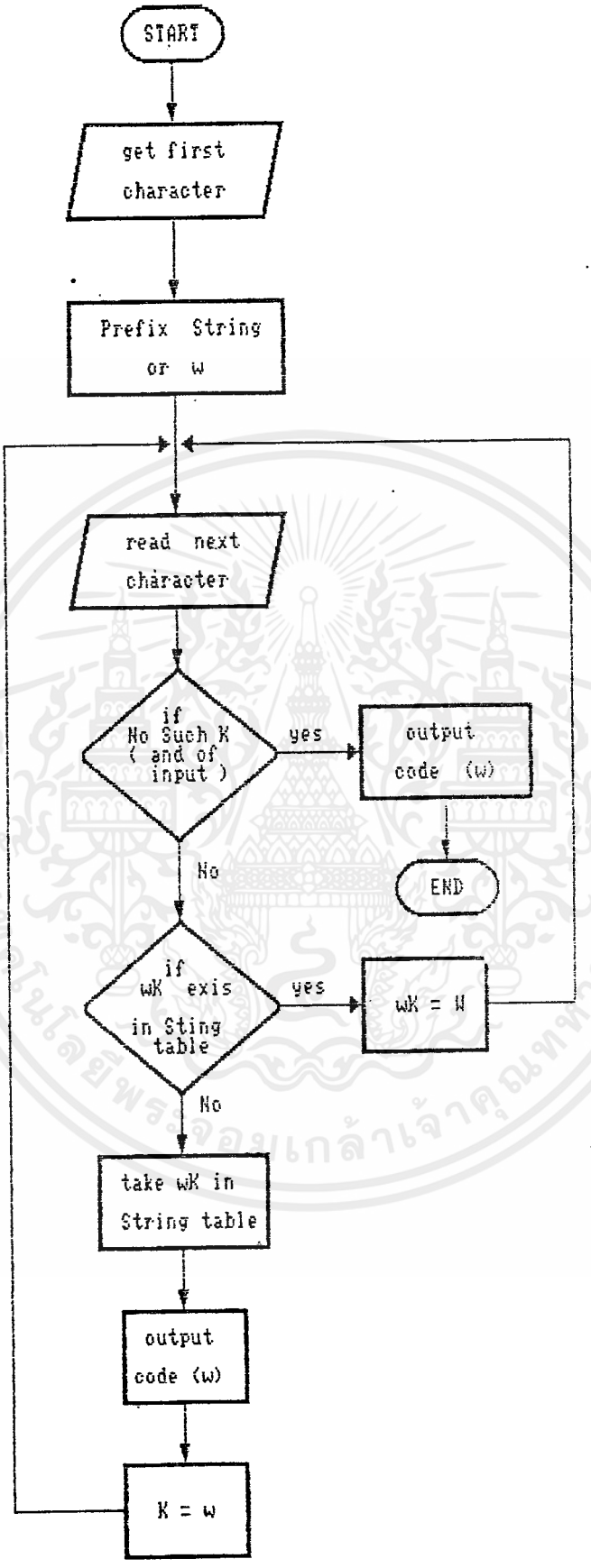
Input String : "WED WE WEE WEB WET"

Character input	Code Output	New code value and associated string
"W"	" "	256 = " W"
"E"	"W"	257 = "WE"
"D"	"E"	258 = "ED"
" "	"D"	259 = "D "
"WE"	256	260 = " WE"
" "	" "	261 = "E "
"WEE"	260	262 = " WEE"
" W"	261	263 = "E W"
"EW"	257	264 = "WEB"
" "	"B"	265 = "B "
"WET"	260	266 = "WET"
<EOF>	T	

ตัวอย่าง แสดงการทำงานของอัลกอริทึม Decompression

Input Code : "WED<256>E<260><261><257>B<260>T"

Input/NEW_CODE	OLD_CODE	STRING/Output	CHARACTER	New table entry
" "	" "	" "		
"W"	" "	"W"	"W"	256 = " W"
"E"	"W"	"E"	"E"	257 = "WE"
"D"	"E"	"D"	"D"	258 = "ED"
256	"D"	"W"	" "	259 = " D"
"E"	256	"E"	"E"	260 = " WE"
260	"E"	" WE"	" "	261 = " E"
261	260	"E "	"E"	262 = " WEE"
257	261	"WE"	"W"	263 = "E W"
"B"	257	"B"	"B"	264 = "WEB"
260	"B"	" WE"	" "	265 = "B "
T	260	T	T	266 = " WET"



FLOWCHART PROGRAM COMPRESS FILE OF LZW