PC NETWORK SYSTEM

โดย

นายทีฆายุ ไชยบุรี

นายวิศาล อัครวิเนศ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2534

ปริญญานิพนธ์

ประจำปีการศึกษา 2534

เรื่อง PC-Network System

ผู้จัดทำ

ทีฆายุ ไชยบุรี 31.1072

วีคาล อัครวิเนค 31.1262

ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหาร ลาดกระบัง

อาจารย์ บรรจง ปิยะธำรง

รศ. ประทีป บัญญัตินพรัตน์

อาจารย์ที่ปรึกษา

กิตติกรรมประกาศ

ขอขอบคุณบริษัท อาร์แอนด์ดี คอมนิวเตอร์ ซิสเต็ม จำกัด ที่ให้การสนับสนุนการวิจัยและพัฒนาโครงงานให้สำเร็จลงได้ด้วยดี

ขอขอบคุณห้างหุ้นส่วนจำกัด อินเสริทท์ เอ็นจิเนียริ่ง ที่ให้ความอนุเคราะห์ในการจัดพิมพ์ปริญญานิพนธ์

และขอขอบคุณ อาจารย์ที่ปรึกษา ที่ทำให้การทำปริญญานิพนธ์สำเร็จลุล่วงลงด้วยดี

ระบบเครือข่ายไมโครคอมพิวเตอร์

โดย

ทีฆายุ ไชยบุรี 31.1072

วิศาล อัครวิเนต 31.1262

อาจารย์ที่ปรึกษา

อ.บรรจง ปิยะธำรง

รศ.ประทีป บัญญัตินพรัตน์

บทคัดย่อ

ปัจจุบันเครื่องไมโครคอมพิวเตอร์เข้ามามีบทบาทในชีวิตประจำวันมากขึ้น สำนักงานทั่วไปมักจะมีเครื่องไมโครคอมพิวเตอร์ สำหรับช่วยการทำงานทั้งทางด้านการคำนวณ ออกแบบ และฐานข้อมูล ซึ่งโดยปกติมักจะใช้งานแบบเครื่องเดี่ยว สำหรับพิมพ์เอกสาร หรือจัดแฟ้มบุคคลในฐานข้อมูล แต่ในความเป็นจริง หากมีการติดต่อสื่อสารกันเป็นระบบเครือข่าย สามารถช่วยให้ทำงานได้สะดวก และมีมาตรฐานเดียวกัน ทั้งยังช่วยประหยัดส่วนสำรองข้อมูลเช่น ฟลอปปีดิสก์ ฮาร์ดดิสก์ ระบบเครือข่าย PC-Network System เป็นระบบที่สนับสนุนการทำงานดังกล่าว ระบบ PC-Network System เป็นซอฟแวร์ที่ใช้ในหน่วยงานเล็กๆที่มีปริมาณงานไม่มากนัก เนื่องจากต้นทุนต่ำกว่าระบบ LAN และ ไม่จำเป็นต้องมีฮาร์ดแวร์ประกอบการทำงาน ความสามารถของระบบ PC-Network System ที่สำคัญคือ

- การสื่อสารข้อมูลกันภายในเครือข่าย
- การโอนย้ายแฟ้มข้อมูล
- การใช้เครื่องพิมพ์ร่วมกัน
- การ run โปรแกรมจากฮาร์ดดิสก์ของ Server

ระบบ PC-Network System เป็นโปรแกรมประเภทฝังตัวในหน่วยความจำ ดังนั้นเราสามารถใช้งานได้ทันทีที่ต้องการเนื่องใช้ Hot-key ตามคำแนะนำที่ปรากฏขึ้นหลังจากเสร็จการติดตั้งโปรแกรมลงในหน่วยความจำ ทำให้ใช้งานได้สะดวก รวดเร็ว ระบบ PC-Network ยังรวมการทำงานหลายๆอย่างเข้าด้วยเป็นโปรแกรมอรรถประโยชน์แบบหน้าต่างให้อีกด้วย

PC-Network System

By

Teekayu Chaiburee 31.1072

Wisan Akarawinake 31.1262

Advisor

Banjong Piyathamrong

Pratheep Banyatnoparat

ABSTACT

In present,Microcomputers are more effect to human activity. There are many office work in calculate ,design and database with microcomputers which are stand alone for print document or file personal file on database. In truth,if we have communicate in network system then we could work comfortable ,standard and save storage media such as floppy-diskett fixdisk. PC-Network System support incordingly theory. PC-Network System is software for light load office ;because of it costs less than LAN ,and indepent on hardware in working. The capability of PC-Network System are

- Internal Communication
- Copy and Delete between 2 PC
- Shared Printer
- Load and Run

PC-Network System is residant program that instantly run if Hot-key is avtive. Hot-key were display after completely install program so we comfort quickly to running program. In other ,PC-Netwrok includ that utility program in window pop-up menu.

สารบัญ

หน้า

บทคัดย่อ

บทนำ

- อะไรคือ Network 1
- LAN ทำงานอย่างไร 2
- ความสามารถของ LAN 2
- การประยุกต์ใช้งานของระบบเครือข่ายท้องถิ่น 4
- การเลือกฮาร์ดแวร์สำหรับระบบเครือข่ายท้องถิ่น 5

บทที่ 1 ตัวกลางและเทคนิคในการติดต่อ

1.1 เทคนิคการส่งข้อมูล 10
1.1.1 Baseband Transmittion 11
1.1.2 Broadband Transmittion 11
1.2 ชนิดของตัวกลาง 12
1.2.1 Twisted pair cable 12
1.2.2 Coaxial cable 13
1.2.3 Optical fiber 13
1.3 ปัญหาที่เกิดขึ้นกับสายส่งข้อมูล 14
1.4 การเลือกสายส่งข้อมูล 15

บทที่ 2 รูปแบบและการพิจารณาการเชื่อมต่อ

2.1 The Star หรือ Radial Topology 17
2.2 The Bus Topology 18

หน้า

2.3 The Ring Topology 20

2.4 The Hybrid Topology 21

2.4.1 The Tree Topology 21

2.4.2 The Star-ring Topology 22

2.5 บทสรุป 23

บทที่ 3 การควบคุมการทำงานของระบบ

3.1 รูปแบบของ Packet 25

3.2 การใช้สายส่งร่วมกัน 26

3.3 Contention-Based Access Method 28

3.3.1 Multiple Access 28

3.3.2 Carrier Sense Multiple Access (CSMA) 28

3.3.3 Carrier Sense Multiple Access With Collision Detect (CSMA/CD) 28

3.3.4 Register Insertion 29

3.4 Noncontention Access Method 30

3.4.1 Sloted Rings 30

3.4.2 Token Passing 31

3.4.2.1 Token-passing Ring 31

3.4.2.2 Token-passing Bus 32

3.5 มาตรฐานสำหรับอนาคต 33

บทที่ 4 การใช้งานระบบเครือข่ายแบบต่างๆ

4.1 การใช้งานระบบเครือข่ายแบบ Ethernet 35

- การเชื่อมต่อทางกายภาพของ Ethernet 38

หน้า

- ระบบเครือข่าย Ethernet ที่ใช้สายส่งข้อมูลแบบบาง 38
- ระบบเครือข่าย Ethernet ที่ใช้สายส่งข้อมูลแบบหนา 39
- - ระบบเครือข่าย Ethernet ที่ใช้สายส่งข้อมูลทั้งสองแบบ 42

4.2 การใช้งานระบบเครือข่ายแบบ ARCnet 43

4.3 การใช้งานระบบเครือข่ายแบบ Token-Ring 47

4.4 การใช้งานระบบเครือข่ายแบบ StarLAN 50

4.5 การใช้งานระบบเครือข่ายแบบ Token Buses 53

บทที่ 5 โครงข่ายการสื่อสารข้อมูล

5.1 การส่งข้อมูล 55

5.1.1 การจำแนกวิธีการส่งข่าวสารตามทิศทางการส่งภายในสาย 55

5.1.2 การจำแนกวิธีการส่งตามความสัมพันธ์ของข้อมูล 56

- การส่งข้อมูลแบบสัมพันธ์ 57
- การส่งข้อมูลแบบไม่สัมพันธ์ 63

5.1.3 การจำแนกวิธีการส่งตามลักษณะการจัดข้อมูล 65

- การส่งข้อมูลแบบขนาน 65
- การส่งข้อมูลแบบอนุกรม 66

5.2 โปรโตคอลของการสื่อสารแบบอนุกรม 67

5.3 OSI มาตรฐานสำหรับซอฟท์แวร์สื่อสาร 68

5.4 มาตรฐาน EIA 69

5.5 CCITT 71

5.6 ANSI และ ISO 71

5.7 Bell Lab 74

บทที่ 6 พอร์ทอนุกรมของเครื่องคอมพิวเตอร์ไอบีเอ็มพีซี

หน้า

6.1 หน้าที่ของพอร์ทอนุกรม 75

6.2 โครงสร้างของพอร์ทอนุกรมบนไอบีเอ็มพีซี 75

6.3 8250 พอร์ทอนุกรมสำเร็จรูป 76

- รีจิสเตอร์ของ 8250 76

- Line Control Register (LCR) 76

- Divisor Latch Least Significant (DLL) 79

- Divisor Latch Most Significant (DLM) 79

- Line Status Register (LSR) 80

- Interrupt Identificantion Register (IIR) 81

- Interrupt Enable Register (IER) 82

- Modem Control Register (MCR) 83

- Modem Status Register (MSR) 84

- Receiver Buffer Register & Transmitter Holding Register 84

6.4 การใช้งานพอร์ทผ่านทาง BIOS 85

บทที่ 7 PC-Network System

7.1 ลักษณะของเครือข่าย PC-Network 86

7.2 โปรแกรมควบคุมการสื่อสารข้อมูล 87

- การทำงานของ PC-Network 88

7.3 ความสามารถ และการใช้งาน PC-Network 89

- การส่งข้อความสื่อสารกันภายในเครือข่าย 90

- การโอนย้ายไฟล์ข้อมูล 92

- โปรแกรมอรรถประโยชน์ 93

- The Remote Terminal Manager (RTM) 93

- Files Transfer MAnager (FTM) 99

7.4 การยกเลิกการติดตั้ง PC-Network 104

7.5 Concept ของการพัฒนา PC-Network 105

- SERVER Shell (SERVER.COM) 105

- LOGIN Shell (NLOGIN.COM) 106

- Program Command Line 106

- NSEND 106

- NCOPY 106

- NRUN 106

- REMOTE2 108

บทที่ 8 บทสรุปและแนวทางในการพัฒนาต่อ 113

บรรณานุกรม

ภาคผนวก

- Source Program Listing

บทนำ

LAN คืออะไร เราคงจะให้คำจำกัดความได้ยากแต่เมื่อเรานึกถึง Computer Network LAN คงเป็นหนึ่งในหลายๆรูปแบบของ Computer Network ซึ่งมีการติดต่อกันระหว่างเครื่องคอมพิวเตอร์ ปัจจุบันมีการแบ่ง Network ไว้ตามลักษณะของเนื้อที่ใช้สอยของการติดต่อไว้ 3 แบบ คือ

- ระบบเครือข่ายระดับประเทศ (WAN หรือ Wide Area Network) เป็นระบบเครือข่ายที่ติดตั้งใช้งานในบริเวณกว้าง เช่น ระบบเครือข่ายซึ่งทำงานครอบคลุมอาณาบริเวณทั่วโลก โดยปกติมีอัตราการส่งข้อมูลที่ต่ำและมีโอกาสเกิดข้อผิดพลาดได้สูง การส่งข้อมูลอาจใช้อุปกรณ์ในการสื่อสาร เช่น โมเด็มมาช่วย
- ระบบเครือข่ายท้องถิ่น (LAN หรือ Local Area Network) เป็นระบบเครือข่ายที่ทำงานครอบคลุมอาณาบริเวณที่เล็กลงมา อาจใช้อยู่ในอาคารเดียวกัน หรืออาคารที่อยู่ใกล้ๆกัน เช่น ภายในมหาวิทยาลัย ภายในอาคารสำนักงาน ในคลังสินค้า หรือโรงงาน เป็นต้น การส่งข้อมูลทำได้ด้วยความเร็วสูง และมีความผิดพลาดเกิดขึ้นน้อย ระบบเครือข่ายท้องถิ่นจึงออกแบบมาเพื่อให้ช่วยลดต้นทุน และเพิ่มประสิทธิภาพในการใช้งานอุปกรณ์ต่างๆร่วมกัน
- ระบบเครือข่ายระดับเมือง (MAN หรือ Metropolitan Area Network) ปัจจุบันกำลังมีการศึกษาวิจัยกันอยู่ การติดต่อสามารถกระทำได้ภายในเมืองเล็กๆ

Network ทั้งสามแบบตั้งชื่อให้สอดคล้องกับ ขนาดของพื้นที่ที่ครอบคลุมการติดต่อของ Computer ซึ่งขนาดของพื้นที่ของการติดต่อ เป็นปัจจัยสำคัญต่อการพัฒนา เทคโนโลยีที่จะใช้ใน Network

อะไรคือ Network

Tanenbaum ให้คำจำกัดความของ Network ไว้ในหนังสือ Computer Networks ซึ่งตีพิมพ์โดย Prentice Hall Inc. ในปี คศ. 1981 ว่า " an interconnected collection of autonomous computers " ซึ่งหมายถึงคอมพิวเตอร์ 2 เครื่อง หรือมากกว่า สามารถติดต่อสื่อสารหรือแลกเปลี่ยนข้อมูลกันได้ จุดสำคัญอยู่ที่คำว่า autonomous ที่มีความหมายว่าไม่มีคอมพิวเตอร์ใน Network สามารถเริ่ม ,หยุด หรือควบคุมคอมพิวเตอร์ตัวอื่นๆใน Network ได้ ซึ่งแตกต่างจากระบบในปัจจุบัน เช่น IBM 370 ประกอบด้วย Terminal หมายเลข 3270 IBM 370 ทำหน้าที่

เป็น Central Host Computer ดังนั้นคำจำกัดความดังกล่าวจึงไม่สามารถใช้กับระบบคอมพิวเตอร์ทั้งหมดในปัจจุบันได้

LAN ทำงานอย่างไร

หลังจากเราทราบถึงคำจำกัดความของ LAN แล้ว ก็ต้องรู้ว่า LAN มีลักษณะการทำงานภายนอกเป็นอย่างไร

- ระหว่าง node ที่ใช้เชื่อมต่อมีระยะทางจำกัด นั่นคือสูงสุดได้ไม่เกิน 10 กม. ต่ำสุดได้ไม่น้อยกว่า 1 ม.
- ปกติทำงานด้วยความเร็ว 1-10 Mbps. หากใช้เส้นใยนำแสงก็จะทำให้ความเร็วเพิ่มขึ้นเป็นระดับร้อย Mbps.
- เนื่องจาก LAN ใช้งานในระยะทางสั้นๆ อัตราความผิดพลาดจึงมีค่าน้อยกว่า WAN คุณสมบัติข้อนี้ทำให้ใช้ protocol และพัฒนางานได้ง่ายขึ้น
- LAN ใช้งานภายในองค์กรเล็กๆ เป็นการติดต่อของ user โดยตรง ต่างจาก WAN ซึ่งเป็นการติดต่อระหว่างเครือข่ายที่ใหญ่ขึ้น ต้องอาศัยศูนย์โทรคมนาคมของประเทศ LAN จึงใช้งานได้สะดวกกว่ามาก

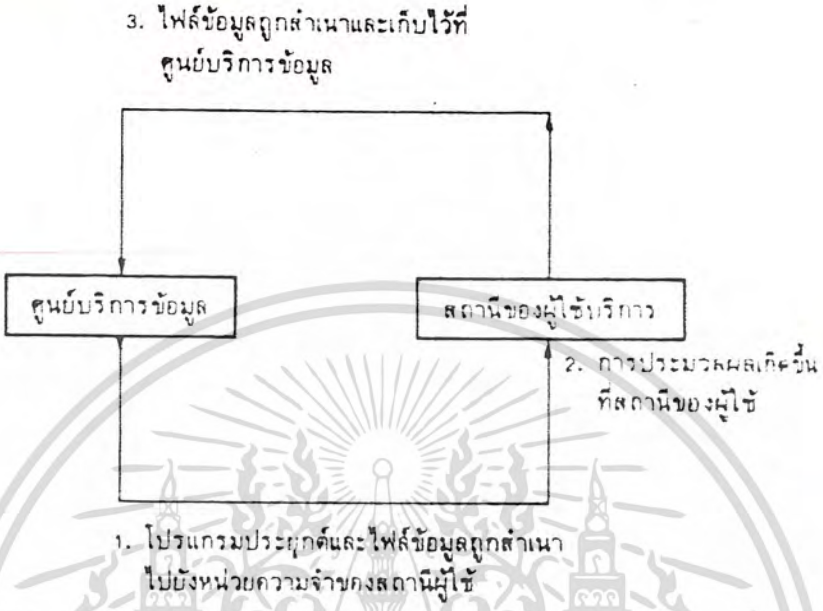
เราจึงสามารถสรุปได้ว่า LAN แตกต่างจากระบบทั่วๆไปตรงที่ ขนาดของพื้นที่มีขีดจำกัด นั่นหมายความว่า มันสามารถทำงานได้ด้วยความเร็วสูง และมีอัตราความผิดพลาดค่อนข้างต่ำ ซึ่งขึ้นอยู่กับการออกแบบ protocol อย่างไรก็ตาม protocol ที่ใช้ในระบบ LAN สามารถออกแบบได้ง่ายกว่าระบบทั่วๆไป นี่คือคุณสมบัติเฉพาะของ LAN

ความสามารถของ LAN

ระบบเครือข่ายท้องถิ่น (LAN) มีข้อดีเหนือมินิคอมพิวเตอร์ (Minicomputer) หรือเครื่องคอมพิวเตอร์เมนเฟรม (Mainframe) อยู่หลายประการกล่าวคือ มีการประมวลผลแบบกระจายงาน (distributed processing) ความรวดเร็วในการติดต่อสื่อสาร การติดต่อระหว่างสถานีผู้ใช้งาน (interconnected stations) ใช้โปรแกรมและข้อมูลร่วมกันได้ ใช้ฮาร์ดแวร์ และทรัพยากรที่มีอยู่ร่วมกันได้

การประมวลผลแบบกระจายงาน ในการประมวลผลแบบกระจาย เมื่อสถานีของผู้ใช้บริการขอโปรแกรม และข้อมูลจาก *ศูนย์บริการข้อมูล (file server)* โปรแกรมคำสั่งจะถูกสำ

เนาจากศูนย์บริการข้อมูลไปยังหน่วยความจำที่สถานีของผู้ใช้ เมื่อเสร็จสิ้นการใช้งานโปรแกรมและไฟล์ข้อมูลแล้ว สถานีผู้ใช้จะส่งข้อมูลกลับไปเก็บยังศูนย์บริการข้อมูล เพื่อใช้งานต่อไป



รูป 1 ลักษณะการประมวลผลแบบกระจายงานให้แต่ละสถานีของผู้ใช้บริการ

การประมวลผลแบบกระจายงานจะช่วยให้สถานีหลายๆสถานีใช้งานร่วมกันในระบบได้ โดยไม่ไปลดความสามารถในการประมวลผลข้อมูลของแต่ละสถานี ในเครื่องคอมพิวเตอร์แบบเมนเฟรม ความสามารถในการประมวลผลจะถูกแบ่งออกไปในแต่ละสถานี ถ้ายิ่งมีสถานีมากก็จะทำให้ประสิทธิภาพในการทำงานลดลง ในการกระจายการประมวลผลข้อมูลจะทำให้เพิ่มความเร็วและประสิทธิภาพของระบบได้

ความรวดเร็วในการติดต่อสื่อสาร ระบบเครือข่ายท้องถิ่นมีการใช้สื่อการส่งข้อมูลของตนเอง (โดยปกติจะเป็นสายส่งข้อมูล) มากกว่าที่จะใช้สื่อสาธารณะเช่น สายโทรศัพท์ ทำให้การติดต่อสื่อสารในระบบเครือข่ายมีความเร็วสูงมากกว่า 1 ล้านบิทต่อวินาทีขึ้นไปในสายส่งข้อมูลความเร็วสูง นอกจากนี้ระบบเครือข่ายท้องถิ่นยังมีช่องทางที่เรียกว่า *ประตูสื่อสาร'' (gateways)* เชื่อมต่อกับเครือข่ายแบบ WAN ในการติดต่อสื่อสารอื่นๆอีกด้วย

การสื่อสารที่มีความรวดเร็ว ช่วยให้ได้ข้อมูลทันต่อเหตุการณ์ ลดความซ้ำซ้อนของข้อมูล และเพิ่มความถูกต้องแม่นยำของข้อมูล

การติดต่อระหว่างสถานีผู้ใช้งาน เมื่อแต่ละสถานีเชื่อมต่อกันเข้าด้วยกันแล้วจะทำให้ผู้ใช้ติดต่อหรือส่งข้อมูลให้แก่กันได้ เช่นการใช้จดหมายอิเล็กทรอนิกส์ (electronic mail) และช่วยให้ผู้ใช้จัดการข้อมูลจากที่อื่นได้

การใช้โปรแกรมร่วมกัน ในระบบเครือข่ายผู้ใช้สามารถจะใช้ข้อมูลและซอฟแวร์ร่วมกันได้ เพื่อลดความซ้ำซ้อนของข้อมูล สื่อบันทึกข้อมูล ตลอดจนการค้นหาตรวจสอบข้อมูลได้รวดเร็วและถูกต้อง การปรับปรุงโปรแกรมที่ใช้งานร่วมกันก็สามารถกระทำ ณ. จุดเดียว

การใช้ทรัพยากรร่วมกัน เนื่องจากระบบเครือข่ายท้องถิ่นอนุญาตให้ผู้ใช้ระบบเครือข่ายใช้อุปกรณ์ร่วมกัน (เช่น เครื่องพิมพ์ โมเด็ม ประตูสื่อสาร อุปกรณ์ด้านกราฟฟิค และอุปกรณ์การเก็บข้อมูลเป็นต้น) ซึ่งจะช่วยลดความซ้ำซ้อนของอุปกรณ์ทำให้ประหยัดค่าใช้จ่าย นอกจากนี้ระบบเครือข่ายท้องถิ่น ยังช่วยให้ผู้จัดการระบบสามารถตรวจสอบการใช้อุปกรณ์ต่างๆ เหล่านี้จากผู้ใช้ในระบบได้อีกด้วย

เมื่อระบบการประมวลผลแบบกระจายได้เริ่มขยายตัวไปมากขึ้น ประกอบกับราคาของเครื่องไมโครคอมพิวเตอร์ถูกลงขณะที่ประสิทธิภาพของเครื่องเพิ่มขึ้นด้วย ดังนั้นการเชื่อมต่อไมโครคอมพิวเตอร์ให้เป็นระบบเครือข่ายท้องถิ่น จะช่วยเพิ่มประสิทธิภาพของการใช้อุปกรณ์ต่างๆในสำนักงานได้ดียิ่งขึ้น

การประยุกต์ใช้งานของระบบเครือข่ายท้องถิ่น

ระบบเครือข่ายท้องถิ่น อาจนำมาประยุกต์ใช้งานได้หลายแบบ เช่น

1. นำมาใช้ควบคุมการผลิตในโรงงานอุตสาหกรรม
2. ใช้เป็นเทอร์มินัลในการบริการ ณ. จุดขายในซุปเปอร์มาเก็ต กับเครื่องควบคุมสต๊อกสินค้า และระบบบัญชีบนเครื่องเมนเฟรม
3. เชื่อมต่อไมโครคอมพิวเตอร์ต่างๆและเครื่องพิมพ์ในสำนักงาน เพื่อจัดทำงานด้านปฏิบัติงานได้รวดเร็วยิ่งขึ้น เช่น ระบบบัญชีพื้นฐานซึ่งรวมบัญชีแยกประเภท บัญชีเจ้าหนี้ บัญชีลูกหนี้ บัญชีสินค้าบัญชีเงินเดือนค่าจ้างเข้าด้วยกัน ระบบทะเบียน และระบบวัดผลนักศึกษาในสถาบันการศึกษา เป็นต้น
4. เชื่อมต่อไมโครคอมพิวเตอร์ต่างๆและเครื่องพิมพ์ในสำนักงาน เพื่อทำเป็นสำนักงานอิเล็กทรอนิกส์ (electronic office) หรือตึกอัตโนมัติ (intelligent building)

ระบบเครือข่ายท้องถิ่นได้พัฒนาขึ้นมาหลายแบบ แต่ที่นิยมใช้กันกว้างขวางเป็นระบบเครือข่ายท้องถิ่นแบบ ISO-OSI (Open System Interconnection) ซึ่งมีคุณสมบัติต่างๆดังนี้

1. มีความยืดหยุ่นสูง ง่ายต่อการเปลี่ยนแปลงโครงสร้าง ในการเพิ่มหรือลดอุปกรณ์ต่างๆ ในระบบเครือข่าย
2. เป็นโครงสร้างที่ขยายสาขาพิเศษออกไปได้เมื่อต้องการ
3. ตัวกลางในการส่งข้อมูล และระบบควบคุม สามารถขยายออกไปได้ง่าย
4. เป็นระบบที่มีความเชื่อถือได้สูง ถ้าสายส่งข้อมูลหรือเครื่องใดเสียหายจะไม่ทำให้การส่งหรือรับข้อมูลของอุปกรณ์อื่นเสียหายไปด้วย
5. เป็นระบบที่กระจายการควบคุม (distributed control) เพื่อเพิ่มความเชื่อถือได้ของข้อมูล

การเลือกฮาร์ดแวร์สำหรับระบบเครือข่ายท้องถิ่น

เทคโนโลยีของระบบเครือข่ายท้องถิ่น ในปัจจุบันมีการเปลี่ยนแปลงไปอย่างมาก ทั้งทางด้านซอฟท์แวร์และฮาร์ดแวร์ ฮาร์ดแวร์ของ LAN มีความสามารถสูงขึ้น รวมทั้งระบบปฏิบัติการของ LAN ก็มีความซับซ้อนขึ้นด้วย นอกจากนี้การเชื่อมต่อระหว่างเครื่องต่างขนาด และต่างระบบกันก็มีความเป็นไปได้มากขึ้น รวมถึงการใช้งานข้อมูลต่างๆ ที่เก็บอยู่บนระบบเหล่านี้ด้วย ในปัจจุบันผู้ใช้เครื่อง PC สามารถใช้งานไฟล์ของตัวเอง (local files) ไฟล์จากศูนย์บริการข้อมูลศูนย์กลางของระบบเครือข่ายท้องถิ่น (central LAN server) ไฟล์จากศูนย์บริการข้อมูลที่อยู่ห่างไกล (remote LAN server) หรือแม้แต่ไฟล์จากคอมพิวเตอร์ศูนย์ (host computer) และนอกจากนี้ผู้ใช้ยังสามารถเชื่อมต่อ (bridging) ระหว่างระบบเครือข่ายท้องถิ่นที่ใช้ ฮาร์ดแวร์ต่างชนิดกันแต่มีโปรโตคอลแบบเดียวกันได้อีกด้วย

ในปัจจุบันผู้ผลิตฮาร์ดแวร์ของระบบ LAN จะทำให้ผลิตภัณฑ์ของตนใช้ได้กับโปรโตคอลของระบบเครือข่ายท้องถิ่นและระบบปฏิบัติการหลายแบบ รวมทั้งยังสนับสนุนการเชื่อมต่อสะพานสื่อสาร และประตูสื่อสาร ที่ช่วยให้ผู้ใช้สามารถเชื่อมต่ออุปกรณ์ชนิดต่างๆเข้าด้วยกันได้ ในทางกลับกัน ผู้ผลิตระบบปฏิบัติการของระบบเครือข่ายท้องถิ่นเอง ก็พยายามปรับปรุงซอฟท์แวร์ของตนให้ใช้งานได้กับฮาร์ดแวร์หลายๆ แบบตัวอย่างเช่น ฮาร์ดแวร์ของระบบเครือข่ายท้องถิ่นของหลายบริษัทสามารถใช้งานได้กับ Novell's NetWare,โปรแกรมของ IBM PC Network, NETBIOS, และ TCP/IP (Transmission Control Protocol/Internet Protocol) ส่วนทางด้านของซอฟท์แวร์ NetWare ของบริษัท Novell, VINES ของบริษัท Banyan และ Network OS ของบริษัท CBIS ก็สนับสนุนการเชื่อมต่อระ

หว่างฮาร์ดแวร์ของระบบเครือข่ายท้องถิ่นหลายแบบ และยังมีประตูสื่อสาร (ที่ใช้เชื่อมต่อระบบเครือข่าย 2 ระบบ ที่ใช้โปรโตคอลในชั้น physical ต่างกัน) ไปยังเครื่องเมนเฟรมและมินิคอมพิวเตอร์ คอมพิวเตอร์ที่ห่างไกล ระบบเครือข่ายท้องถิ่น และระบบเครือข่ายแบบ WANs ในปัจจุบันระบบเครือข่ายท้องถิ่นสามารถที่จะใช้สายส่งข้อมูล และ Topology ได้หลายแบบ

ผู้ผลิตฮาร์ดแวร์ส่วนใหญ่มักจะผลิตฮาร์ดแวร์ตามมาตรฐานที่กำหนดโดย IEEE/ANSI เช่น CSMA/CD token-passing bus และ token-passing ring การเลือกมาตรฐานของฮาร์ดแวร์ขึ้นอยู่กับ

1. ชนิดของสายส่งข้อมูล (cable type)
2. โทโพโลยี (topology)
3. วิธีใช้สายส่งข้อมูล (line access method)
4. เทคนิคที่ใช้ส่งข้อมูล (transmission technique)
5. ความเร็วในการส่งข้อมูล (transmission rate)
6. การขยายระบบทางภูมิศาสตร์ (geographic span)
7. ขนาดของตัวบอกตำแหน่ง (address size)
8. ระบบปฏิบัติการที่สามารถใช้กับระบบเครือข่ายท้องถิ่นได้
9. ชนิดของ CPU ที่สามารถใช้ได้ (supported CPU)
10. ประสิทธิภาพของระบบ (performance and throughput)

ชนิดรองสายส่งข้อมูล ชนิดของสายส่งข้อมูลที่ใช้ก็มีความสำคัญ ปัจจุบันมีสายข้อมูลอยู่หลายแบบ เช่น สายคู่บิดเกลียวแบบหุ้มฉนวน และสายคู่บิดเกลียวแบบไม่หุ้มฉนวน สำหรับการเดินสายระยะสั้น และแบบเส้นใยนำแสงสำหรับการเดินสายระยะทางไกล นอกจากนี้ในระบบเครือข่ายท้องถิ่นบางระบบยังสามารถใช้แสงอินฟราเรด ไมโครเวฟ คลื่นวิทยุ และแม้แต่สายไฟฟ้าได้อีกด้วย ความสามารถอันนี้จะช่วยให้ระบบเครือข่ายท้องถิ่นมีราคาถูก มีความยืดหยุ่นสูง ทำให้ง่ายต่อการติดตั้งและการขยายระบบในอนาคต

โทโพโลยี การเดินสายสำหรับระบบเครือข่ายท้องถิ่นก็เป็นข้อที่ควรคำนึงถึง การเลือกโทโพโลยีของระบบเครือข่ายท้องถิ่นให้เหมาะสมกับพื้นที่ใช้งานในปัจจุบันและอนาคต ควรมีการวางแผนให้ดี เพื่อที่จะประหยัดค่าใช้จ่ายในการเดินสาย ง่ายต่อการติดตั้งและให้ประสิทธิภาพที่ดีแก่ระบบ

วิธีใช้สายส่งข้อมูล เป็นชนิดของสัญญาณที่ใช้บนสายส่งข้อมูลที่ยอมให้อุปกรณ์หลายอันสามารถใช้สายส่งข้อมูลสายเดียวกันได้ โดย IEEE ได้กำหนดวิธีการส่งที่ใช้แยกระบบเครือข่ายท้องถิ่นแบบบัสไว้ 2 วิธี คือ CSMA/CD และ Token-passing

ในการใช้งานทั่วไป วิธีควบคุมการใช้สายส่งข้อมูลดูจะไม่ค่อยมีความจำเป็นนัก แต่ในสภาพแวดล้อมบางอย่างการใช้งานสายส่งข้อมูลก็มีความจำเป็น เช่นในระบบการควบคุมการผลิต ควรจะใช้ระบบเครือข่ายแบบ token passing (802.5 Token Rings) เช่น Pronet, ARCnet และอื่นๆ ซึ่งมีการใช้สายส่งข้อมูลแบบ "deterministic" มากกว่าที่จะใช้วิธีแบบ "probabilistic" ของระบบเครือข่ายแบบ contention (เช่น Ethernet ,G-Net) Token passing จะรับรองว่ากลุ่มข้อมูลสื่อสารของข้อมูลจะถูกส่ง ในขณะที่แบบหลังจะมีความน่าจะเป็นสูงที่ข้อมูลจะถูกส่งแน่นอน

เทคนิคที่ใช้ส่งข้อมูล การเลือกการส่งข้อมูลระหว่างเบสแบนด์และบรอดแบนด์ก็เป็นสิ่งที่ควรคำนึงถึงเช่นเดียวกัน ระบบแบบเบสแบนด์จะส่งสัญญาณเดี่ยวบนสายส่งข้อมูลที่เวลาใดเวลาหนึ่ง ในขณะที่ระบบแบบบรอดแบนด์จะส่งหลายๆสัญญาณบนสายส่งเส้นเดียวกันโดยใช้ความถี่ที่ต่างกัน ในการใช้งานระบบที่เป็นแบบเบสแบนด์จะมีราคาถูกกว่า ส่วนในระบบแบบบรอดแบนด์จะมีความซับซ้อนและมีความยืดหยุ่นสูงแต่จะมีราคาแพงกว่า

ความเร็วในการส่งข้อมูล ในระหว่างระบบเครือข่ายแบบมาตรฐานนั้น อัตราการส่งผ่านของกลุ่มข้อมูลสื่อสารบนระบบเครือข่ายจะแตกต่างกัน เช่น StarLAN มีความเร็ว 1 ล้านบิทต่อวินาที Token Ring มีความเร็ว 4 ล้านบิทต่อวินาที Token Bus มีความเร็ว 1 ถึง 10 ล้านบิทต่อวินาที Ethernet มีความเร็ว 10 ล้านบิทต่อวินาที ซึ่งความเร็วในการส่งข้อมูลต่างๆเหล่านี้ จะเป็นเพียงองค์ประกอบอันหนึ่งเท่านั้นในองค์ประกอบอื่นอีกหลายอัน ที่จะมีผลกระทบต่อประสิทธิภาพรวมของระบบ

การขยายระบบทางภูมิศาสตร์ ระยะทางทั้งระบบเครือข่าย และระยะทางระหว่างจุดที่ไกลที่สุด จะเป็นตัวจำกัดการขยายระบบทางด้านภูมิศาสตร์ ระบบเครือข่ายแบบ IEEE 802 สามารถใช้ตัวทวนซ้ำสัญญาณได้ ถ้าหากไม่มีตัวทวนซ้ำสัญญาณแล้ว ระบบเครือข่ายแบบ CSMA/CD 802.3 ที่สายโคแอกเชียลแบบบางจะขยายระบบได้ 305 เมตร แต่ถ้าเปลี่ยนมาใช้สายโคแอกเชียลแบบหนาแล้ว จะขยายได้ถึง 1000 เมตร ส่วนระบบแบบ token ring สายส่งข้อมูลจะยาวได้ถึง 100 เมตร จากฮับ (hub) และขยายได้ 300 ถึง 600 เมตร ซึ่งขึ้นอยู่กับการจัดการของวงแหวน

สะพานสื่อสารหรือ selective repeaters เป็นอีกวิธีหนึ่งที่ใช้เพื่อขยายระบบเครือข่าย โดยจะใช้วิธีการแบบ store-and-forward เพื่อทวนซ้ำสัญญาณของกลุ่มข้อมูลสื่อสารในเซกเมนต์ที่ต่ออยู่กับตัวทวนซ้ำสัญญาณ สะพานสื่อสารจะช่วยเพิ่มประสิทธิภาพของระบบเครือข่ายที่มีหลายเซกเมนต์ได้

ขนาดรองด้วยบอกตำแหน่ง ขนาดของตัวบอกตำแหน่งจะเป็นตัวที่บอกว่าจำนวนอุปกรณ์ที่ต่อได้ในระบบมีได้เท่าใด ในการนำโปรโตคอล CSMA/CD แบบใหม่ไปใช้งานจะใช้ตัวบอกตำแหน่งแบบ 48 บิท ส่วนแบบเก่าจะใช้เพียง 16 บิท token buses และ rings ใช้ทั้งแบบ 16 บิท หรือ 48 บิท ขึ้นอยู่กับการใช้งาน ระบบการใช้ตัวบอกตำแหน่งแบบ 48 บิท โดยปกติแล้วประกอบด้วยรหัสของผู้ผลิต และชุดของตัวเลขที่เป็นตัวบอกหมายเลขอุปกรณ์ที่ใช้ในระบบเครือข่าย

ระบบปฏิบัติการที่สามารถใช้กับระบบเครือข่ายท้องถิ่นได้ ในระบบเครือข่ายท้องถิ่นของเครื่องไมโครคอมพิวเตอร์ องค์ประกอบที่สำคัญอันหนึ่งก็คือระบบปฏิบัติการที่ใช้ ดังนั้นระบบเครือข่ายท้องถิ่นที่เลือกต้องสามารถใช้กับระบบปฏิบัติการเหล่านี้ได้ อาร์ดแวร์ของระบบเครือข่ายท้องถิ่นของผู้ผลิตหลายรายได้สนับสนุนการใช้ระบบปฏิบัติการของระบบเครือข่ายหลายแบบด้วย รวมทั้ง NetWare ของบริษัท Novell, VINES ของ Banyan และโปรแกรมสำหรับระบบเครือข่ายท้องถิ่นของ IBM PC ทำให้เกิดความยืดหยุ่นในการเปลี่ยนแปลงไปใช้ระบบปฏิบัติการอื่นๆภายหลัง นอกจากนี้ อาร์ดแวร์ของระบบเครือข่ายท้องถิ่นจากผู้ผลิตบางราย ยังสามารถเชื่อมต่อกับอุปกรณ์ที่ไม่ใช่เป็นแบบของไมโครคอมพิวเตอร์ เช่น DEC VAXes, Apple Macintoshes, Sun wokstation เข้ากับระบบเครือข่ายได้อีกด้วย

ชนิดรอง CPU ที่สามารถใช้ได้ ถ้ามีการวางแผนเชื่อมต่อระบบเครือข่ายท้องถิ่น เข้ากับอุปกรณ์ที่ไม่ใช่เป็นไมโครคอมพิวเตอร์ ควรคำนึงด้วยว่ามีอุปกรณ์เชื่อมต่อเชื่อมต่อ (interface) สำหรับระบบเครือข่ายท้องถิ่นแบบนั้นอยู่หรือไม่ เช่นอุปกรณ์เชื่อมต่อของ Ethernet จะมีอยู่บนทั้งไมโครคอมพิวเตอร์ มินิคอมพิวเตอร์ และเมนเฟรม ระบบเครือข่ายแบบ Token-Ring Pronet 10 ของบริษัท Proteon จะมีการติดต่อได้หลายแบบ ซอฟท์แวร์ TCP/IP ใช้ได้กับ Pronet 10 และ Ethernet ได้หลายแบบ TCP/IP ช่วยให้สามารถส่งไฟล์ และจำลองเทอร์มินัลของระบบที่ต่างกันให้เป็นเทอร์มินัลของอีกระบบหนึ่งได้

ประสิทธิภาพของระบบ ประสิทธิภาพของระบบเครือข่ายท้องถิ่น มีองค์ประกอบ

หลายอย่างด้วยกัน อย่างแรกคืออัตราการส่งข้อมูล ซึ่งเป็นความเร็วที่ข้อมูลเดินทางผ่านระบบเครือข่ายท้องถิ่น อย่างไรก็ตามองค์ประกอบอื่นเช่น วิธีใช้สายส่งข้อมูล ประสิทธิภาพในการเชื่อมต่อระบบ และอัตราการส่งถ่ายข้อมูล ก็มีผลต่อประสิทธิภาพรวมของระบบ

ในบทต่อๆไปจะกล่าวถึงทฤษฎีที่เกี่ยวข้องกับการออกแบบ และสร้างระบบเครือข่ายท้องถิ่นอย่างง่ายๆขึ้น ซึ่งในระบบเครือข่ายโดยทั่วไปจะต้องใช้กัน เช่น สายตัวนำหรือสายสื่อสารที่ใช้ในการติดต่อ รูปแบบและโครงสร้างของการเชื่อมต่อ ลักษณะการส่งข้อมูล และลักษณะของข้อมูล เป็นต้น เพื่อใช้เป็นพื้นฐาน และอ้างอิงการออกแบบโครงงานชิ้นนี้ขึ้นมา ส่วนสุดท้ายก็จะเป็นการใช้งานระบบเครือข่ายไมโครคอมพิวเตอร์ที่สร้างขึ้น

บทที่ 1
ตัวกลางและเทคนิคในการติดต่อ

ในทุกๆเครือข่ายจะต้องมีการเชื่อมต่อตัวกลางเพื่อส่งผ่านข้อมูลจากจุดหนึ่งไปยังอีกจุดหนึ่ง ตัวกลางใดๆจะต้องมีความแข็งแรง และทนทานต่อสภาพแวดล้อมได้ดี เช่น กระแสไฟฟ้า ,แสง หรือสิ่งรบกวนทางคลื่นแม่เหล็กไฟฟ้า *(EMI : Electromagnetic Interference)* นอกจากนั้นจะต้องปรับปรุงหรือซ่อมแซมเครือข่ายได้ง่าย ดังนั้นต้นทุนในการติดตั้งระบบจึงขึ้นอยู่กับ ราคาของตัวกลางและการติดตั้งตัวกลางในแต่ละจุดอีกด้วย การติดต่อระหว่างจุดต่างๆใน Network จะต้องมีความเร็วเหมาะสมในช่วงการใช้งาน เราเรียกช่วงนั้นว่า *Terminal-to-Host Traffic* สำหรับตัวกลางที่เราเลือกใช้จะต้องมีความเร็วในการทำงานเพียงพอกับงานที่ใช้ในระบบ เพื่อจะได้เลือกใช้ตัวกลางได้เหมาะสม เราจึงต้องศึกษาลักษณะและปัญหาของตัวกลางแต่ละประเภทพร้อมทั้งวิธีการส่งข่าวสารข้อมูลในรูปแบบต่างๆกัน

1.1 เทคนิคการส่งข้อมูล

วิธีการส่งข้อมูลทางดิจิตอลผ่านตัวกลางนั้นมีหลายวิธีด้วยกัน การเลือกวิธีใดวิธีหนึ่งนั้น ข้อสำคัญอยู่ที่ ความเร็ว ,ความเที่ยงตรงในการส่งข้อมูล และราคา บางครั้งตัวแปรในระบบก็มีอิทธิพลต่อวิธีการส่งข้อมูล วิธีง่ายๆที่ใช้กันคือ ปรับเปลี่ยนสัญญาณทางไฟฟ้าให้มีความหมายสอดคล้องกับข้อมูล และเปลี่ยนสัญญาณเหล่านี้กลับเป็นข้อมูลทางด้านเครื่องรับ อุปสรรคสำคัญของวิธีนี้คือ *การลดทอนของสัญญาณ (attenuation)* และ *คลื่นรบกวน (noise)* noise เกิดจากแหล่งกำเนิดต่างๆมากมายในสภาพแวดล้อมและมันจะไปรบกวนสัญญาณเดิมให้ผิดเพี้ยนจากปกติ ส่วน attenuation เป็นการทำให้สัญญาณมีความแรงหรือกำลังของสัญญาณลดลง ในขณะส่งผ่านตัวกลางในระยะทางต่างๆกัน นอกจากนั้นการลดทอนของสัญญาณยังขึ้นอยู่กับเฟสของความถี่ของสัญญาณ ซึ่งมีผลทำให้สัญญาณของข้อมูลถูกรบกวน

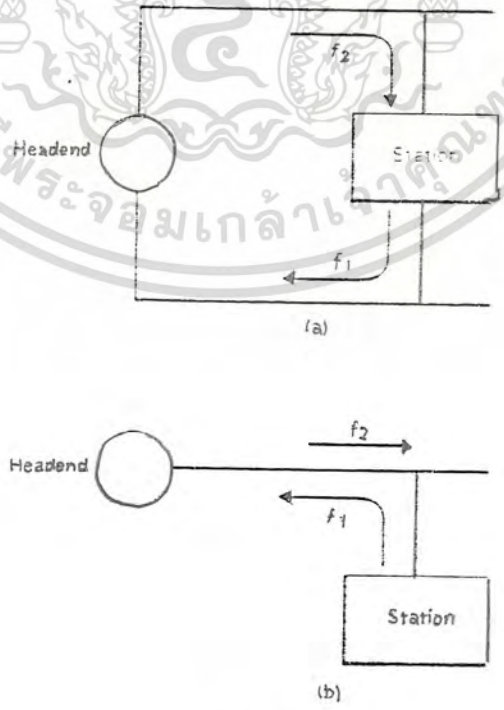
สำหรับตัวกลางชนิดต่างๆกัน สามารถใช้งานในช่วงความถี่ที่แตกต่างกัน ทำให้ส่งข้อมูลโดยไม่เกิดการลดทอน หรือเกิดน้อยที่สุด ถ้าหากเราใช้ความถี่นอกเหนือจากช่วงความถี่การใช้งานดังกล่าว การส่งข้อมูลก็จะมีโอกาสเกิดความผิดพลาดขึ้นมาก ในการวัดขนาดของข้อมูลที่ส่งผ่านตัวกลางนั้น เราพิจารณาจากความกว้างของช่วงความถี่ที่เรียกว่า *Bandwidth* ของตัวกลางที่มีขนาดต่างๆกันไปตามลักษณะเฉพาะของตัวกลางแต่ละชนิด ซึ่งการส่งข้อมูลในระบบ LAN นี้จะแบ่งได้ใน 2 ลักษณะใหญ่ๆ คือ *Baseband* และ *Broadband* ดังรายละเอียดที่จะกล่าวต่อไป

1.1.1 Baseband Transmittion

ในการส่งข้อมูลทางดิจิตอล มีการแทนระดับของแรงดันไฟฟ้า (Voltage) เป็นสองระดับคือ '0' และ '1' โดย '0' แทนระดับแรงดันไฟฟ้าค่าต่ำ และ '1' แทนระดับแรงดันไฟฟ้าค่าสูง การส่งข้อมูลแบบ baseband transmittion นี้จะทำให้สัญญาณทางไฟฟ้าที่เข้ารหัสไว้มีความแรงน้อยลงตามระยะทางที่เคลื่อนที่ได้ ดังนั้นในระบบที่ใช้การส่งข้อมูลแบบ baseband transmittion จะต้องมีความเร็วในการส่งข้อมูลต่ำ และในระยะทางการติดต่อที่ไกลมากๆ ต้องมี *เครื่องทวนสัญญาณ (Repeater)* เพื่อเพิ่มแรงดันไฟฟ้าในสายตัวกลาง

1.1.2 Broadband Transmittion

การส่งสัญญาณโดยวิธีของ broadband เป็นหนึ่งในรูปแบบของ *FDM (Frequency Division Multiplex)* ซึ่งวิธีของ broadband เป็นวิธีสร้างสัญญาณในการส่งข้อมูลผ่านสายส่งหลายๆช่องสัญญาณนั่นเอง ช่องสัญญาณจะถูกสร้างขึ้นมาด้วยสัญญาณที่ถูก modulate กับความถี่ต่างๆใน Bandwidth ที่ต่างกัน ทำให้สามารถแบ่งช่องสัญญาณได้เป็นหลายช่องสัญญาณ คุณสมบัติที่แตกต่างจากการส่งสัญญาณแบบ baseband คือ สำหรับการส่งสัญญาณแบบ broadband นั้น สัญญาณถูกส่งไปในทิศทางเดียวเสมอ ขึ้นอยู่กับเครื่องส่งและเครื่องรับ ต่างจาก baseband ที่สัญญาณอาจมีการเคลื่อนที่สองทิศทางในช่องสัญญาณเดียวกัน



รูป 1.1

การออกแบบเพื่อให้แต่ละ node ใน broadband transmittion สามารถเป็นได้ทั้งเครื่องรับและเครื่องส่ง จึงต้องมีอุปกรณ์ที่เรียกว่า *Head-end* ช่วยในการรับส่งสัญญาณในช่องสัญญาณเดียวกัน ดังรูป 1.1 f_1 เป็นคลื่นพาหะที่ใช้ในการส่งสัญญาณ f_2 เป็นคลื่นพาหะที่ใช้ในการรับสัญญาณ Head-end จะรับสัญญาณจาก node ด้วยความถี่ f_1 แล้วส่งสัญญาณไปยัง node ต่างๆด้วยความถี่ f_2 ซึ่งวิธีดังกล่าวเป็นการส่งสัญญาณด้วยช่องสัญญาณต่างกัน แต่มีทิศทางเดียวกันในช่องสัญญาณเดียวกัน

1.2 ชนิดของตัวกลาง

ตัวกลางในระบบ LAN นั้นมีให้เลือกมากมาย ทั้งตัวนำที่เป็นโลหะ เช่น ทองแดง หรือตัวนำทางด้านแสง เช่น optic fiber หรืออื่นๆ เช่น ไมโครเวฟ ,อินฟราเรด เป็นต้น ล้วนแต่มีข้อดีและข้อเสีย ในหัวข้อนี้จะเปรียบเทียบและแนะนำตัวกลางแต่ละชนิดว่า มีข้อดีและข้อเสียอย่างไรเพื่อประโยชน์ในการเลือกใช้งานได้เหมาะสม

1.2.1 Twisted pair cable

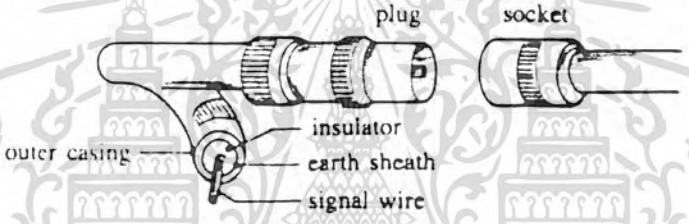
Twisted pair cable เป็นอุปกรณ์การสื่อสารข้อมูลที่นิยมมาก ประกอบด้วยสายทองแดง 2 เส้นที่พันกันอยู่ มี impedance เท่ากัน คุณสมบัติข้อนี้ทำให้สาย cable ถูกรบกวนจาก noise ได้น้อยลง ปัญหาที่เกิดขึ้นจากคุณสมบัติทางไฟฟ้า เช่นความยาว หรือการวางสายที่มีลักษณะเหมือนตัวเก็บประจุไฟฟ้า เป็นต้น จากปัญหาดังกล่าวนี้ ในระบบ LAN เราจึงใช้สายที่มีคุณภาพดีขึ้นเรียกสายชนิดนี้ว่า *data grade medium (DGM)* ทำให้มีข้อมูลที่ถูกต้องแน่นอนขึ้น เพราะคุณสมบัติที่มีลักษณะเหมือนตัวเก็บประจุลดลงจากการใช้วัสดุคุณภาพสูง และ noise ลดลงเพราะมี shield ที่มีลักษณะเป็นทองแดงที่ถักเป็นตาข่ายป้องกันไว้

ข้อดีของ Twisted pair cable ก็คือติดตั้งได้ง่าย เพราะสมบัติทางกายภาพเช่น น้ำหนักเบา และ ยังเชื่อมต่อได้ง่าย แต่การส่งข้อมูลยังไม่ดีนัก เพราะมีการลดทอนของสัญญาณ ในการใช้งานในระยะทางไกลๆ จึงต้องมีเครื่องทวนสัญญาณ นอกจากนี้ Bandwidth ของ Twisted pair cable ก็มีค่าต่ำ ทำให้ใช้งานในแบบ broadband ไม่ได้ อีกทั้งยังเกิด noise ได้ง่าย

รูป 1.2 Twisted pair cable

1.2.2 Coaxial cable

Coaxial cable ประกอบด้วยสายแข็งห่อหุ้มด้วย shield ทองแดง 1 ชั้นหรือมากกว่า ทั้งสายทองแดงและ shield แยกจากกันด้วยฉนวนไฟฟ้า สายด้านในทำหน้าที่นำสัญญาณ shield จะถูกต่อลงกราวน์ด Coaxial cable นิยมใช้ในวงการสื่อสารด้านโทรทัศน์ *(CATV : Community Attenna Television)* เนื่องจากเราสามารถส่งคลื่นโทรทัศน์ได้ทีละหลายๆช่องพร้อมกัน เพราะมี Bandwidth กว้างถึง 300 MHz. จึงใช้เครือข่ายระบบ *Shared Cable Network* ได้ นอกจากนั้น Coaxial cable มีการสูญเสียพลังงานในสายส่งต่ำกว่า Twisted pair โดยเฉพาะที่ความถี่สูงๆ เราจึงใช้เครื่องทวนสัญญาณน้อยลง ส่วน shield ที่ต่อลงกราวน์ดไว้ นอกจากจะช่วยป้องกัน noise แล้วยังช่วยส่งสัญญาณในขณะเดียวกันอีกด้วย

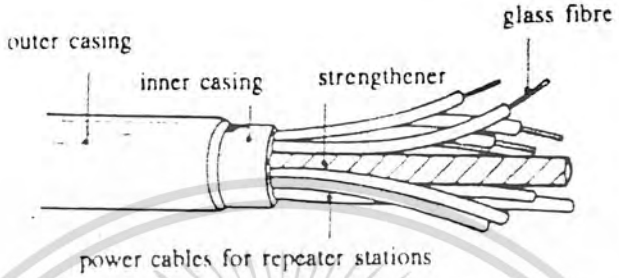


รูป 1.3 Coaxial cable

1.2.3 Optical fiber

Optic fiber ทำจากเส้นใยที่มีลักษณะคล้ายแก้ว มีโครงสร้างที่สามารถนำแสงจากแหล่งกำเนิดที่ปลายด้านหนึ่ง ไปยังอีกด้านหนึ่ง แหล่งกำเนิดแสงดังกล่าวอาจเป็น *LED* หรือ *LDs (Laser diodes)* ข้อมูลจะถูกส่งผ่านโดยการ modulate เข้ากับความถี่ของแสง Bandwidth ในการส่งสัญญาณของตัวกลางสูงมากกว่า 5 GHz. และอัตราการ modulate ข้อมูลขึ้นอยู่กับแหล่งกำเนิดแสงแต่ละชนิด เช่น LED จะอยู่ระหว่าง 20-150 Mbps. และจะสูงขึ้นกว่านี้ถ้าใช้ LDs เป็นแหล่งกำเนิดแสง แต่ปัญหาที่สำคัญของ Optic fiber คือ แตกหักง่าย นอกจากนั้นการเชื่อมต่อของแหล่งกำเนิดแสงกับ cable เองก็เป็นเรื่องที่ยุ่งยากซับซ้อน หากไม่ดีพอก็จะเกิดการสูญเสียพลังงานไปในสายส่ง ข้อได้เปรียบของ Optic fiber ที่เห็นได้ชัดคือ มันสามารถป้องกันการรบกวนได้สมบูรณ์แบบเพราะว่า ข้อมูลเคลื่อนที่ไปพร้อมกับแสงที่มีความถี่สูงมาก สิ่งรบกวนทางคลื่นแม่เหล็กไม่มีผลต่อข้อมูลในสายส่งเลย

นอกจากนี้ยังไม่สามารถดักจับสัญญาณได้จากสายส่ง การจับสัญญาณจะต้องตัดสายออกและต่อเข้ากับตัวรับสัญญาณเท่านั้น ซึ่งจะมีประโยชน์ในการรักษาความปลอดภัยของข้อมูล



รูป 1.4 Optical fiber

1.8 ปัญหาที่เกิดขึ้นกับสายส่งข้อมูล

ปัญหาสำคัญในการใช้สายส่งข้อมูล ก็คือสัญญาณรบกวน ซึ่งเกิดขึ้นได้ 2 กรณีคือ

1. สัญญาณรบกวนที่เกิดจากสภาพแวดล้อมภายนอก เช่นสายส่งข้อมูลผ่านอุปกรณ์ไฟฟ้าแรงสูงหรืออาจเกิดจากสัญญาณที่แผ่ออกมาจากสายส่งที่อยู่ใกล้กันก็ได้
2. สัญญาณรบกวนที่มาจากตัวสายส่งข้อมูลเอง ไปรบกวนการทำงานของอุปกรณ์อื่นได้ สัญญาณที่แผ่ออกมาจากสายส่งข้อมูลนี้เอง ทำให้เกิดปัญหาในด้านการรักษาความปลอดภัยของข้อมูลตามมา เพราะสามารถตรวจจับสัญญาณนี้ได้โดยใช้ตัวตรวจจับ และแปลงสัญญาณนี้ออกมาเป็นข้อมูลได้ ดังนั้นการลดการแผ่สัญญาณของสายส่งสัญญาณก็ช่วยเพิ่มความปลอดภัยของข้อมูลได้
3. สายทองแดงที่ใช้ส่งข้อมูลมีคุณสมบัติเหมือนเสาอากาศวิทยุ ดังนั้นจึงรับสัญญาณรบกวนจากภายนอกได้เป็นอย่างดี การหลีกเลี่ยงปัญหานี้ทำได้โดยการใช้ทองแดงหุ้มรอบนอกของสายส่งอีกทีหนึ่ง แต่ก็จะเพิ่มราคาของสายส่งข้อมูลนั้น

สายส่งข้อมูลแบบเส้นใยนำแสงใช้แสงเป็นตัวส่งข้อมูล ทำให้ไม่มีปัญหาด้านสัญญาณไฟฟ้าที่รบกวน ทำให้เหมาะสมที่จะใช้ติดตั้งในสภาพแวดล้อมในโรงงานซึ่งมีการใช้อุปกรณ์ที่ใช้ไฟฟ้าแรงสูง

1.4 การเลือกสายส่งข้อมูล

ในอดีตสายส่งข้อมูลสำหรับแต่ละโปรโตคอลของระบบเครือข่าย จะเป็นสายแบบโคแอกเชียล และสายโทรศัพท์ เช่น ระบบ 802.3 จะใช้ได้เฉพาะสายแบบโคแอกเชียล เมื่อไม่นานมานี้ได้มีการแนะนำสายคู่บิดเกลียวแบบไม่หุ้มฉนวนสำหรับระบบเครือข่ายแบบ Ethernet ความเร็ว 10 เมกกะบิทต่อวินาที และ IBM ได้สนับสนุนสื่อ 3 ชนิดในระบบเครือข่าย token-ring สายโทรศัพท์ธรรมดา (24 gauge unshielded) แบบคู่บิดเกลียวได้กลายมาเป็นมาตรฐานของระบบเครือข่ายท้องถิ่นขนาดเล็ก ซึ่งสายส่งข้อมูลเหล่านี้จะมีราคาถูก และติดตั้งได้ง่าย

สำหรับระบบเครือข่ายที่มีขนาดใหญ่ขึ้นมา เช่น เดินสายหลายๆชั้นภายในอาคารเดียวกัน นิยมใช้สายโคแอกเชียล และสายคู่บิดเกลียว ส่วนระบบเครือข่ายที่มีการใช้งานระหว่างอาคาร ควรใช้สื่อแบบเส้นใยนำแสงมากกว่า

โดยทั่วไปแล้วมีแนวโน้มว่าจะมีการใช้สื่อแบบเส้นใยนำแสงกันมาก เพราะสื่อดังกล่าวส่งข้อมูลด้วยแสง ดังนั้นมันจึงมีความเร็วในการส่งข้อมูลสูงกว่าสายส่งข้อมูลแบบทองแดง 10 ถึง 100 เท่า

เทคโนโลยีของเส้นใยนำแสงเพิ่งจะเริ่มต้นขึ้น การปรับปรุงทางด้านตัวส่งข้อมูลและตัวรับข้อมูล ส่วนใหญ่จะเป็นการเพิ่มช่วงสัญญาณของสื่อข้อมูลแบบนี้ ข้อเสียที่สำคัญของสายส่งข้อมูลแบบนี้ คือ ราคา ทั้งอุปกรณ์ และค่าติดตั้งสูง

คณะกรรมการ ANSI (X3T.9) ได้พิมพ์มาตรฐานสำหรับ token ring 802.5 ที่ใช้งานกับสื่อข้อมูลแบบเส้นใยนำแสงความเร็วสูงขึ้นมา คือมาตรฐาน Fiber Distributed Data Interface (FDDI) กำหนดระบบเครือข่าย token ring ที่ติดต่อกันผ่านเส้นใยที่ความเร็ว 100 เมกกะบิทต่อวินาที และสามารถต่อได้สูงถึง 500 จุด ในระบบเครือข่าย ซึ่งมีขนาดเส้นรอบวงได้ถึง 100 กิโลเมตร ความสามารถอีกอันหนึ่งของ FDDI ก็คือคุณสมบัติ fault-tolerance โดยทั่วแหวนจะสามารถปรับปรุงจากสายส่งข้อมูลขาด หรือสถานีเสียโดยไม่ต้องถอดสถานีผู้ใช้ออกหรือต่อสายใหม่ ความเชื่อถือได้อันนี้ทำงานโดยใช้วงแหวนที่สองที่เป็นตัวตรวจสอบการสับเปลี่ยน เมื่อสถานีใดๆ ตรวจสอบได้ว่ามีสายส่งข้อมูลขาดอยู่ระหว่างกัน มันจะเปลี่ยนทิศทางการส่งข้อมูลไปยังสายส่งข้อมูลของวงแหวนที่สอง และจัดการติดต่อขึ้นใหม่

บทที่ 2
รูปแบบและการพิจารณาการเชื่อมต่อ

จากบทที่ 1 เราพิจารณาตัวกลางในการส่งข้อมูลแล้ว ต่อไปก็ต้องตัดสินใจว่า เราจะเชื่อมต่อจุดต่างๆใน Network เข้าด้วยกันได้อย่างไร รูปแบบของการเชื่อมต่อจุดต่างๆใน Network เข้าด้วยกันนั้นเราเรียกว่า Topology การเลือกเอา topology ใดมาใช้งานนั้น เราต้องคำนึงถึงตัวกลางและทฤษฎีในการติดต่อด้วย เพราะการวางสายตัวกลางในระบบ LAN สิ่งสำคัญคือราคา นอกจากนี้ยังมีปัจจัยอีกหลายประการด้วยกันคือ

- ต้นทุน (Cost) เมื่อตัวกลางใดๆถูกเลือกใช้ในระบบ การติดตั้งในแง่ของความยาวของสายส่ง ซึ่งมีอิทธิพลต่อราคาของระบบ ในแต่ละ Network ย่อมต้องการต้นทุนในการติดตั้งต่ำทั้งสิ้น วิธีที่ดีที่สุดในการลดต้นทุนของการติดตั้งระบบคือ ใช้ตัวกลางที่เหมาะสมและใช้ ระยะทางในการติดตั้งต่ำที่สุด
- ความยืดหยุ่น (Flexibility) เนื่องจาก LAN สามารถขยายจุดเชื่อมต่อได้อีกมากมาย จุดเชื่อมต่อเหล่านี้จะต้องใช้งานได้สะดวก ดังนั้นหากมีการเปลี่ยนแปลง หรือเคลื่อนย้าย อุปกรณ์ในสำนักงาน หรืออาคาร รูปแบบในการเชื่อมต่อจะต้องถูกเปลี่ยนหรือดัดแปลงให้เข้ากับโครงสร้างใหม่ได้ง่าย ความยืดหยุ่นในการเชื่อมต่อ จึงหมายถึงการเพิ่ม หรือลดหรือเคลื่อนย้ายจุดเชื่อมต่อได้ง่ายนั่นเอง
- ความน่าเชื่อถือ (Reliability) ความผิดปกติส่วนใหญ่ในระบบ แบ่งเป็นสาเหตุใหญ่ๆได้ 2 ประการ คือ
  1. ความผิดปกติที่จุดเชื่อมต่อ อาจเกิดจากการเชื่อมต่อระหว่าง terminal กับสายส่ง เป็นต้น ความผิดปกตินี้จะมีอันตรายน้อย
  2. ความผิดปกติของโปรแกรมที่ใช้ในระบบ

ดังนั้น Topology ที่ดีจะต้องมีความเที่ยงตรง และเชื่อถือได้

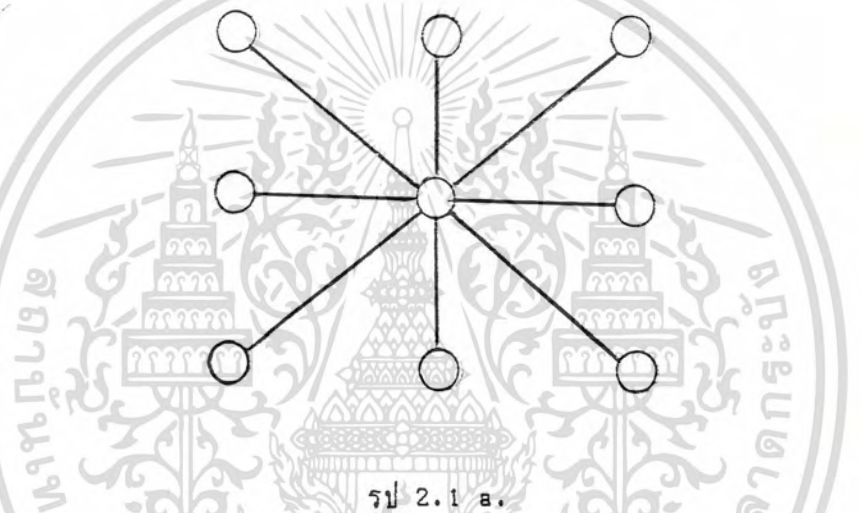
เราสามารถแบ่งการเชื่อมต่อระบบเป็นรูปแบบหลักที่สำคัญได้ เป็น 3 แบบ คือ

- star หรือ radial topology
- bus topology
- ring หรือ loop topology

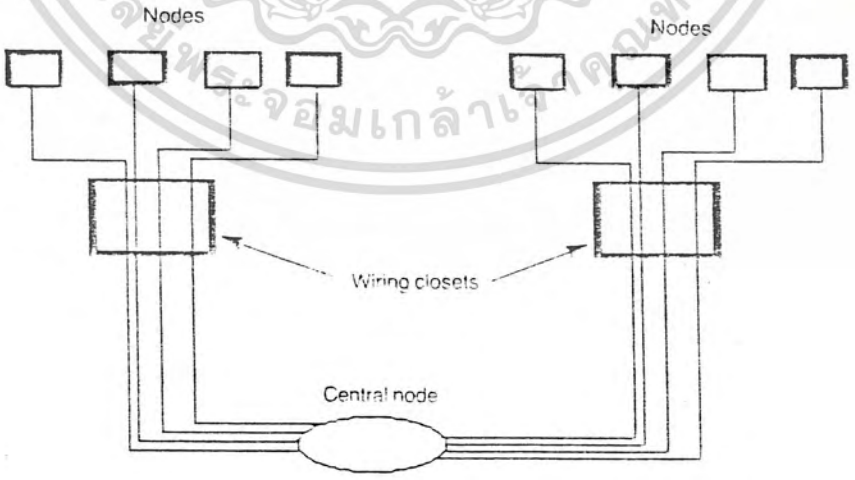
นอกจากนี้ยังมีการเชื่อมต่อแบบ hybrid topology ที่เกิดจากการผสมผสานรูปแบบทั้ง 3 แบบข้างต้น ดังจะกล่าวรายละเอียดต่อไป

2.1 The Star หรือ Redial Topology

star topology จะประกอบด้วย central node ซึ่งมี node อื่นเชื่อมต่ออยู่เป็นรูปดาว ดังรูป 2.1a Topology แบบนี้นิยมใช้กันมากใน Network ที่เกี่ยวกับ data processing และ voice communication ตัวอย่างที่พบกันบ่อยๆ คือ PBX ในสำนักงานต่างๆ เป็นต้น star topology นี้มักจะมีจุดรวมของสายสัญญาณที่เรียกว่า *ศูนย์กลางสายส่งข้อมูล (wiring closet)* ดังรูป 2.1b ไว้สำหรับรวมสายสัญญาณ เนื่องแบ่งการใช้งานสำหรับพื้นที่ย่อยๆ เช่น ชั้นต่างๆ ของอาคารสำนักงาน เป็นต้น



รูป 2.1 a.



รูป 2.1 b.

ข้อดีของ star topology

- ง่ายในการให้บริการหรือติดตั้งระบบ เพราะมีจุดศูนย์กลางสายส่งข้อมูลที่จุดเดียวกัน
- มีอุปกรณ์เพียง 1 ตัว ต่อจุดเชื่อมต่อ 1 จุด ทำให้แต่ละอุปกรณ์มีอิสระต่อกัน เมื่อเกิดความผิดปกติก็ไม่ทำให้ทั้งระบบเสียหาย
- แก้ไข ซ่อมแซม ปัญหาที่เกิดขึ้นที่ node ได้อย่างสะดวกเพราะแต่ละ node มีอิสระต่อกัน
- ใช้ protocol อย่างง่ายๆในการติดต่อสื่อสารข้อมูล ระหว่างจุดต่างๆในระบบ การเชื่อมต่อในระบบที่ใช้ Topology แบบดาวจะเกี่ยวข้องกันระหว่างศูนย์กลางและอุปกรณ์ที่อีกจุดหนึ่งเท่านั้น ทำให้การควบคุมการส่งข้อมูลทำได้ง่าย

ข้อเสียของ star topology

- ต้องใช้สายส่ง หรือตัวกลางในการติดต่อยาวมาก เนื่องจากแต่ละจุดจะต่อโดยตรงกับศูนย์กลาง ทำให้ต้องเสียค่าใช้จ่ายเพิ่มขึ้นในการติดตั้งและบำรุงรักษา
- การขยายระบบ หรือเพิ่มจำนวนจุดเชื่อมต่อในระบบทำได้ลำบาก
- การทำงานจะขึ้นอยู่กับ central node หาก central node เกิดความผิดปกติ จะมีผลทำให้ node อื่นๆทั้งหมดทำงานผิดปกติ ด้วย

2.2 The Bus Topology

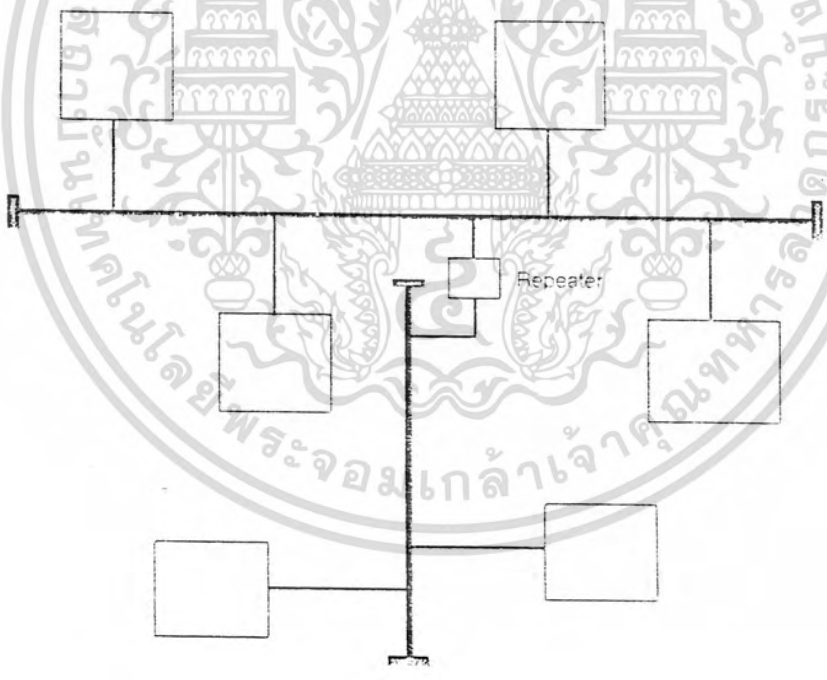
bus topology ประกอบด้วยสายส่งสัญญาณเพียงเส้นเดียว (มักเป็น coaxial cable) สำหรับการติดต่อของ node ในระบบ node ต่างๆจะเรียงกันตามความยาวของสายส่ง ดังรูป 2.2 Topology แบบนี้ Computer หลัก หรือ host computer จะอยู่ตรงส่วนปลายของสายส่ง และมี Terminal ติดต่อกันไปตามยาวของสายส่ง รูปแบบของสายส่งนี้เราเรียกว่า *multidrop line*

ข้อดีของ bus topology

- สายส่งสั้น ทำให้สามารถจัดวางสายส่งได้ง่าย สะดวกต่อการบำรุงรักษา
- ขยายหรือเพิ่มเติม node ได้ง่าย เพียงแต่ใช้เครื่องทวนสัญญาณ (repeater) สำหรับสายส่งเส้นอื่นๆที่ต้องการเชื่อมต่อด้วยเท่านั้น
- มีโครงสร้างที่ง่าย สถาปัตยกรรมแบบบัสมีโครงสร้างที่ง่ายและมีความเชื่อถือได้ เพราะใช้สายส่งเพียงเส้นเดียว

ข้อเสียของ bus topology

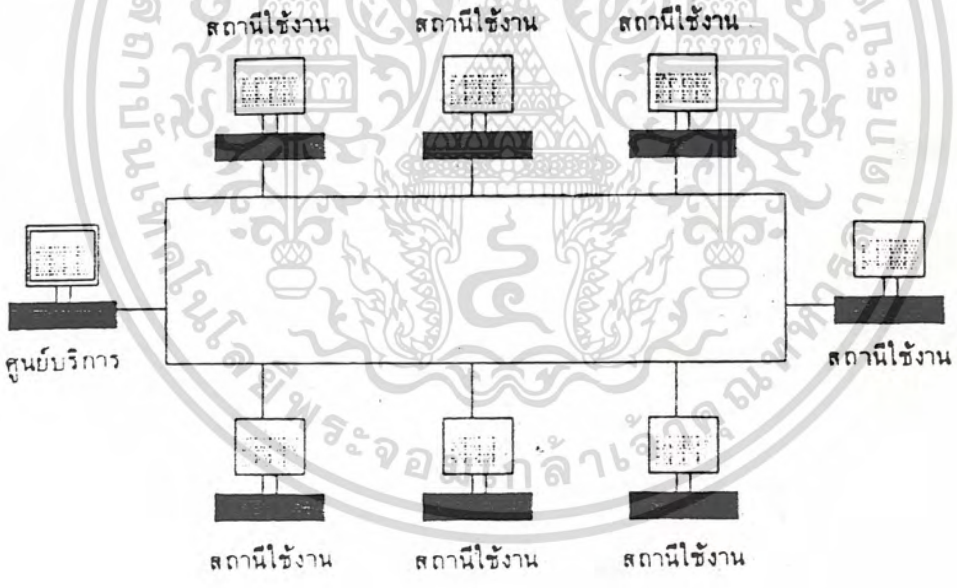
- ความผิดปกติในระบบ ยากต่อการตรวจสอบ เพราะทุกๆ node ในสายส่งทั้งหมดมีโอกาสเกิดความผิดปกติเท่ากัน จึงต้องตรวจสอบไปทุก node จนกว่าจะพบสาเหตุ
- ถ้ามีการใช้ repeater ในระบบ จะต้องกำหนดลักษณะเฉพาะของ repeater บางครั้ง จะต้องมีการปรับแต่ง Terminal ของระบบใหม่
- จุดในระบบต้องฉลาดพอ เนื่องจากแต่ละจุดในระบบเครือข่ายต่อโดยตรงกับบัส ซึ่งหมายความว่าการตัดสินใจว่าใครจะใช้งานสายส่งข้อมูล จะเป็นหน้าที่ของแต่ละจุด จะต้องแบ่งการใช้งานของ central line ให้กับ node อย่างถูกต้องเหมาะสม จึงเป็นไปได้ว่าราคาต้นทุนของระบบจะเพิ่มขึ้นทั้งส่วนของ hardware และ software



รูป 2.2

2.3 The Ring Topology

การเชื่อมต่อสายส่งแบบ ring topology นี้ แต่ละ node จะเชื่อมต่ออยู่กับ node ข้างๆ เสมอ ดังนั้นแต่ละ Terminal จะต่ออยู่กับ node 2 node ข้อมูลจะถูกส่งผ่านจาก node หนึ่งไปยังอีก node อื่นๆเรื่อยไปในทิศทางเดิมและทิศทางเดียวกัน ดังรูป 2.3 จะเห็นได้ว่า Topology แบบ ring นี้มีลักษณะการเชื่อมต่อเป็นวงแหวนตามชื่อ ข้อสำคัญคือข้อมูลจะถูกส่งผ่านจาก node หนึ่งไปยังอีก node หนึ่ง ไม่ใช่เคลื่อนที่ผ่าน นั่นหมายความว่าสัญญาณจะถูกขยายก่อนทุกครั้ง ที่จะส่งผ่านไปยัง node อื่น เมื่อ Terminal จุดหมายได้รับข้อมูลแล้ว จะต้องกำหนดรหัสที่ Terminal ที่ส่งข้อมูลมาให้รับรู้ว่า ข้อมูลถึงจุดหมายแล้ว การกำหนดรหัสดังกล่าวจะทำก่อนที่ข้อมูลจะถูกส่งต่อไป



รูป 2.3

ข้อดีของ ring topology

- ใช้สายส่งสั้น นั่นคือสั้นกว่า แบบ star แต่อาจจะยาวกว่าหรือใกล้เคียงกับแบบ bus ทำให้ช่วยเพิ่มความน่าเชื่อถือของระบบได้มากขึ้น
- ไม่จำเป็นต้องใช้ wiring closet เพราะสายส่งแต่ละเส้นจะต่ออยู่กับ node เสมอ
- สามารถดัดแปลงสายส่งให้ใช้กับเส้นใยนำแสงได้ เพราะการใช้สายส่งข้อมูลแบบเส้นใยนำแสงจะช่วยให้ส่งข้อมูลได้ด้วยความเร็วสูง โดยสัญญาณข้อมูลจะเคลื่อนที่ไปในทิศทางเดียวกันเสมอ ซึ่งเส้นใยนำแสงก็มีคุณสมบัติที่สัญญาณเคลื่อนที่ในทิศทางเดียว ในการส่งแต่ละจุดจะเชื่อมกับจุดติดกันด้วยสายส่งข้อมูลทำให้สามารถเลือกได้ว่า จะใช้สายส่งแบบไหนในแต่ละส่วนของระบบ เช่น เลือกใช้สายแบบเส้นใยนำแสงในส่วนที่ใช้ในโรงงานซึ่งมีปัญหาด้านสัญญาณไฟฟ้ารบกวนมาก

ข้อเสียของ ring topology

- เมื่อมี node ใดผิดปกติ หรือไม่ทำงานจะมีผลทำให้เกิดความผิดปกติทั้งระบบ
- ยากต่อการตรวจสอบความผิดปกติ เพราะความผิดปกติดังกล่าวมีผลทำให้ node อื่นๆ หยุดทำงาน
- จัดโครงสร้างของระบบใหม่ได้ยาก การเพิ่มเติม Terminal จะทำให้ระบบทั้งหมดต้องหยุดทำงานนั่นคือ ไม่สามารถแยกส่วนเพื่อเพิ่มขยาย Terminal ได้
- Topology นี้มีผลต่อรูปแบบของการส่งสัญญาณ เพราะว่าก่อนการส่งสัญญาณข้อมูลทุกครั้งต้องแน่ใจว่าสายส่งว่างที่จะส่งข้อมูลได้

2.4 The Hybrid Topology

hybrid topology เกิดจากการนำเอารูปแบบทั้ง 3 แบบข้างต้น มารวมกันเพื่อเพิ่มประสิทธิภาพในการใช้งาน ให้ได้ประโยชน์สูงสุด

2.4.1 The Tree Topology

เป็น Topology ที่ใช้กันใน IBM PC-Network ซึ่งเลียนแบบมาจากต้นไม้ที่มีรากแก้ว และแตกกิ่งก้านสาขาออกไป สายส่งที่ใช้มักเป็น coaxial cable และใช้เทคนิคการส่งข้อมูลแบบ broadband Topology รูปแบบนี้พัฒนามาจาก bus topology ข้อแตกต่างระหว่าง tree topology และ bus topology คือ tree topology จะมีส่วนประกอบที่เรียกว่า root หรืออีกชื่อ

หนึ่งว่า *Head-end* ทำหน้าที่รับและส่งสัญญาณจากทุกๆ node หมายความว่า ข้อมูลจากทุกๆ node ในระบบจะต้องส่งผ่าน Head-end เสมอ ด้วยวิธีนี้จะทำให้ใช้เครื่องทวนสัญญาณน้อยลง

ข้อดีของ tree topology

- เพิ่ม หรือขยายจุดเชื่อมต่อได้ง่าย เพียงแค่เพิ่มจุดเชื่อมต่อเข้าไป ก็สามารถทำงานได้ทันที เพราะแต่ละจุดมีอิสระต่อกัน
- สามารถแยกจุดเชื่อมต่อแต่ละจุดออกจากระบบได้ทันที หากเกิดความผิดปกติ

ข้อเสียของ tree topology

- การทำงานของทุกๆ node จะขึ้นอยู่กับ Head-end หาก Head-end ไม่ทำงาน node ทุกๆ node ในระบบก็จะไม่ทำงาน

2.4.2 The Star-ring Topology

เป็นการผสมผสานระหว่าง star topology และ ring topology เพื่อเพิ่มประสิทธิภาพในการทำงาน รูปแบบของมันประกอบไปด้วยจุดรวม wiring closet ที่เชื่อมต่อกันแบบ ring topology โดยที่แต่ละ wiring closet อาจอยู่แยกกันบนชั้นต่างๆของอาคาร node ใน wiring closet จะถูกใช้งานแบบ star topology การทำงานของมันคล้ายการทำงานแบบ ring topology แต่จะมีการจัดวางสายเป็นอนุกรมของ star topology บางครั้งเราจึงเรียกการเชื่อมต่อสายรูปแบบนี้ว่า *star-shaped ring*

ข้อดีของ star-ring topology

- สามารถแยกจุดเชื่อมต่อต่างๆในระบบเพื่อตรวจสอบได้ง่าย
- เพิ่มเติม หรือขยาย node ได้โดยไม่กระทบกระเทือนต่อส่วนอื่นๆในระบบ
- เชื่อมต่อสายส่งได้สะดวก และสั้นลง

ข้อเสียของ star-ring topology

- แต่ละ wiring closet จะต้องมีการตรวจสอบตัวเอง และทราบว่ามีการส่งข้อมูลผ่านมายังตัวของมันและยังต้องแยกส่งไปยัง node ที่เชื่อมต่ออยู่กับตัวมันได้ถูกต้อง
- มีข้อจำกัดของการเชื่อมต่อสายระหว่างจุดรวม wiring closet เพราะเป็นการเชื่อมต่อสายแบบ star topology

2.5 บทสรุป

การเลือก Topology ที่ใช้ในระบบต่างๆนั้น เราต้องคำนึงถึงปัจจัยหลายประการ เช่น การติดตั้ง การเพิ่มขยายให้ได้เนียงพอกับความต้องการ เป็นต้น ซึ่งปัจจัยด้านภูมิประเทศมักมีการเปลี่ยนแปลงเสมอให้ทันกับความต้องการของผู้ใช้ นอกจากนี้การซ่อมบำรุงก็มีส่วนสำคัญ เพราะผู้ใช้ต้องการให้ระบบกลับคืนสู่สภาพเดิมโดยเร็วที่สุด สิ่งเหล่านี้จึงเป็นปัจจัยสำคัญในการเลือกรูปแบบของ Topology ที่จะใช้งานให้ได้ประโยชน์สูงสุด

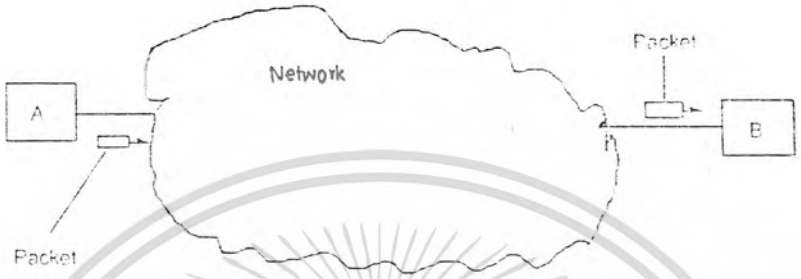
บทที่ 3
การควบคุมการทำงานของระบบ

จากบทที่ 2 Topology ทุกแบบ ยกเว้นแบบ star topology สายส่งสัญญาณจะถูกแบ่งให้ node ได้ใช้งานร่วมกัน ปัญหาที่เกิดขึ้นจากการที่ node ใช้งานสายส่งร่วมกัน มี 2 ลักษณะ คือ อย่างแรก ต้องแน่ใจว่ามี node เพียง node เดียวเท่านั้นที่กำลังใช้งานสายส่งอยู่ node อื่นๆจะส่งข้อมูลพร้อมกันไม่ได้โดยเด็ดขาด อย่างที่สอง เราจะต้องสร้างรูปแบบการส่งข้อมูลระหว่าง node ที่ทำให้สามารถส่งข้อมูลถึงกันได้อย่างถูกต้อง

วิธีการสื่อสารข้อมูลในระบบมีอยู่ด้วยกันหลายวิธี วิธีง่ายๆ คือ การกำหนดสายส่งให้สื่อสารถึงกันได้โดยตรง เราเรียกวิธีนี้ว่า *circuit switching* เป็นวิธีการเชื่อมต่อสายสัญญาณเส้นใดๆที่ต้องการสื่อสารข้อมูลถึงกันโดยตรง ดังรูป 3.1 circuit switching ใช้กันมากในระบบโทรศัพท์ ซึ่งจะติดต่อถึงกันโดยผ่านเครื่องสลับสายคู่สายอัตโนมัติ ในชุมสายโทรศัพท์

รูป 3.1

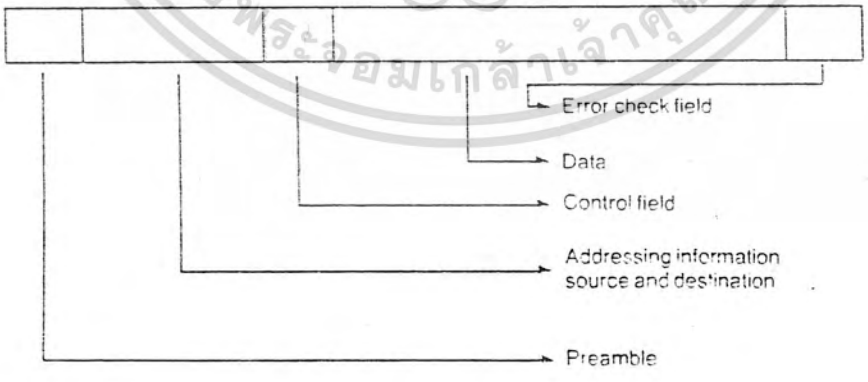
แต่ในการสื่อสารข้อมูลที่ไม่ใช่สัญญาณเสียง มักมีระยะเวลาในการติดต่อนานมาก ดังนั้นในช่วงเวลาที่รอการติดต่อ เราจะเสียเวลาช่วงหนึ่งไปโดยเปล่าประโยชน์ วิธีการที่ทำให้การติดต่อได้ประโยชน์สูงสุด คือ วิธีที่เรียกว่า *packet switching* วิธีนี้จะไม่มีทางเฉพาะที่จะติดต่อระหว่างต้นทางกับปลายทาง ข้อมูลจะถูกเก็บไว้ในลักษณะที่เรียกว่า Packet และถูกส่งเข้าไปใน Network ดังรูป 3.2 แต่ละ node เพียงแต่ทีการกำหนดลักษณะพิเศษให้กับตัวกลางขณะที่ส่งข้อมูล ด้วยวิธีนี้ทุกๆ node ในระบบก็จะมีโอกาสส่งข้อมูลได้มากขึ้น



รูป 3.2

3.1 รูปแบบของ Packet

รูปแบบของ Packet ที่ใช้ในระบบ LAN นั้นมีอยู่มากมาย แต่โดยทั่วไปจะประกอบด้วยส่วนต่างๆ ดังรูป 3.3



รูป 3.3

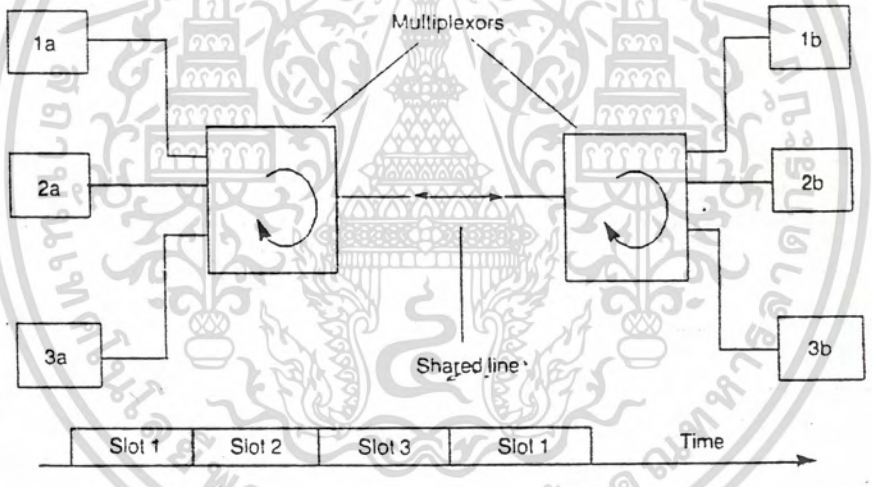
- *Preamble* เป็นส่วนหัวของ Packet ปกติประกอบด้วยรหัส หรือข้อมูลที่เป็นพิเศษ ในบางระบบอาจหมายถึง ความถี่ของสัญญาณนาฬิกาที่ใช้ในการส่งข้อมูล
- *Source and Destination address* node แต่ละ node ใน Network จะถูกกำหนดชื่อ หรือตำแหน่ง (address) ไว้ address เหล่านี้จะไม่ซ้ำกัน และกำหนดโดยรูปลักษณะทางกายภาพ เพื่อใช้อ้างอิงถึงตำแหน่งของผู้ส่ง และผู้รับ โดยเฉพาะในระบบ LAN จะเปรียบเสมือนว่ามีข้อมูลทั้งหมดส่งผ่านตลอด แต่จะเก็บเอาข้อมูลที่มี address ของ Packet ตรงกับ address ของ node เท่านั้น address ของผู้ส่งก็มีความสำคัญเพราะ node ปลายทางจะสามารถส่งสัญญาณตอบรับกลับมายังผู้ส่งได้ถูกต้อง ขณะที่ผู้ส่งก็จะได้ยกเลิกการส่ง Packet ที่ถึงปลายทางแล้ว ทำให้ลดความหนาแน่นของข้อมูลในสายส่งลง บางครั้ง Network อาจกำหนด address พิเศษเพื่อจุดประสงค์บางอย่าง เช่น อาจใช้เป็น address สาธารณะที่ทุกๆ node สามารถตอบรับข้อมูลได้
- *Type* เป็นส่วนที่ใช้ในการควบคุม และแสดงชนิดของ Packet ข้อมูลส่วนนี้จะอยู่หน้าข้อมูลหลัก เพื่อแสดงชนิดของข้อมูลที่ตามมาข้างหลัง
- *Data* เป็นข้อมูลที่ต้องการติดต่อส่งผ่านระหว่าง node จริงๆ อาจจะกำหนดความยาวของข้อมูลได้ทั้งแบบตายตัว หรือเปลี่ยนแปลงได้
- *ส่วนตรวจสอบความผิดพลาด* เนื่องจากความผิดพลาดในระบบ LAN มีโอกาสเกิดขึ้นน้อยมาก ด้วยเหตุผลนี้การตรวจสอบจึงใช้วิธีการที่ไม่ยุ่งยากซับซ้อนเท่าใดนัก เช่น *Cyclic redundancy checking (CRC)* หรือวิธี *check parity bit*

3.2 การใช้สายส่งร่วมกัน

การทำให้ข้อมูลส่งผ่านสายส่งตามลำดับ ต้องมีวิธีการควบคุมการทำงานของสายส่งให้ใช้งานระหว่าง node เพียงคู่เดียว เครื่องมือที่ใช้ควบคุมการทำงานของสายส่งเราเรียกว่า *Time Division Multiplexor* ดังรูป 3.4 เครื่องมือนี้จะกำหนดคู่ของสายส่ง ณ.เวลาหนึ่งๆ ดังนั้นคู่ที่ 1a และ 1b จะใช้สายส่งในช่วงเวลา slot 1 คู่ที่ 2 ก็จะใช้สายส่งในช่วงเวลา slot 2 ข้อเสียข้อหนึ่งของวิธีนี้คือ ขณะที่คู่ใดคู่หนึ่งได้ใช้สายส่งอาจไม่มีการส่งข้อมูลก็ได้ และหากมีการส่งข้อมูลระหว่างที่ไม่ได้ใช้สายส่ง ข้อมูลอาจสูญหายไปได้

ในระบบ LAN นั้นทุกๆ node จะต้องมีวิธีการส่งข้อมูลเหมือนกัน วิธีการหรือทฤษฎีแบ่งได้เป็น 2 ชนิด คือ *Contention* และ *Noncontention* ในกรณีของ Contention node จะคอยเวลาที่ Network ว่างลงจึงจะเริ่มทำการส่งข้อมูล หากมี node มากกว่า 1 node ต้องการส่งข้อมูลในเวลา

เดียวกัน เราต้องใช้เทคนิคอื่นช่วยเพื่อให้สามารถส่งข้อมูลได้ถูกต้อง ส่วนกรณีของ noncontention ถ้า node ใด node หนึ่งต้องการส่งข้อมูล จะต้องมีสัญญาณการตอบรับจาก Network ก่อน



รูป 3.4

3.3 Contention-Based Access Method

3.3.1 Multiple Access

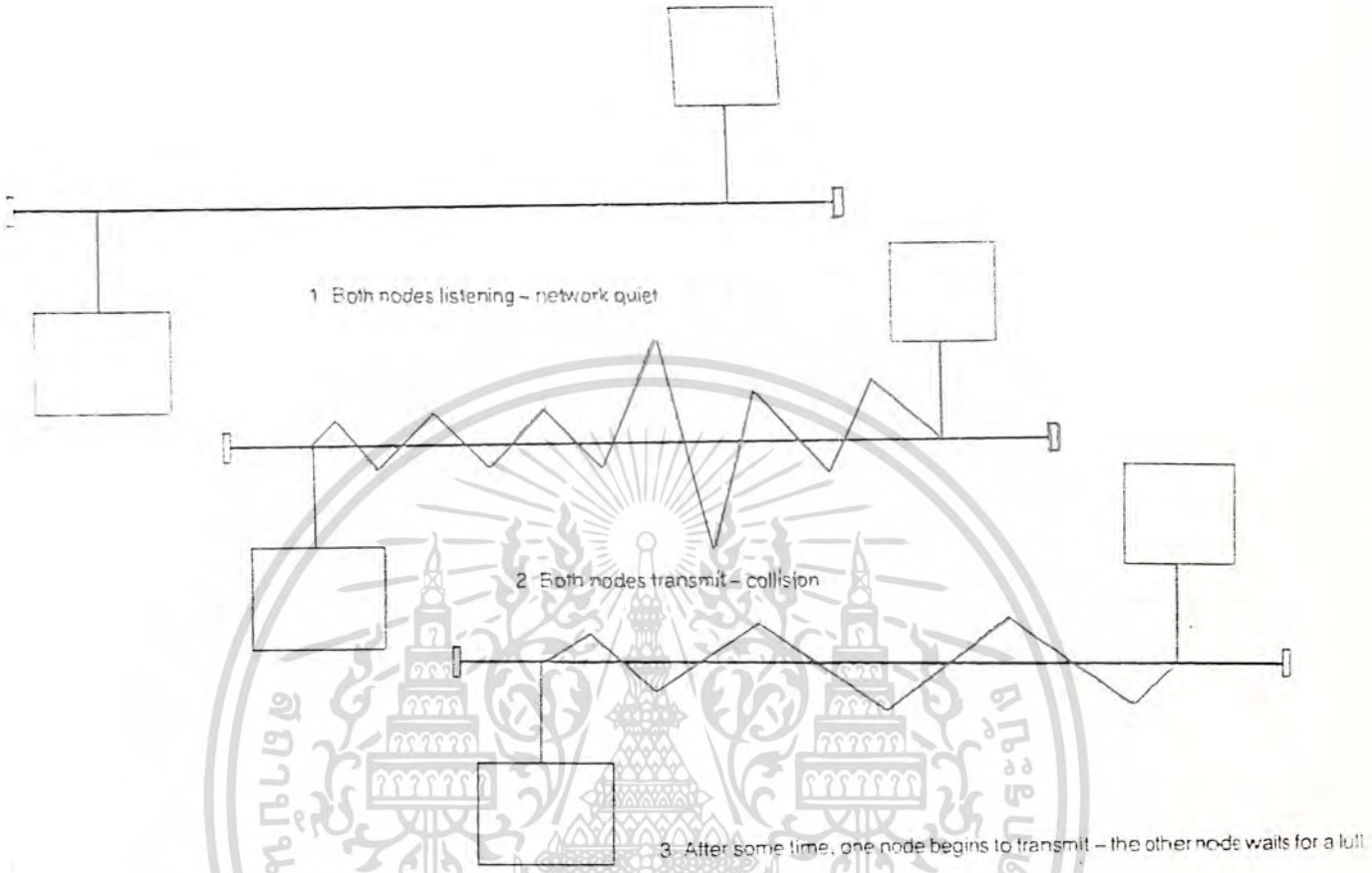
Contention-based Network เกิดขึ้นครั้งแรกโดยมหาวิทยาลัย อาวาย เราเรียกระบบนี้ว่า *Aloha System* เป็นระบบที่ใช้เชื่อมโยงวิทยาเขตทั้ง 7 แห่งใน 4 เกาะ ของมหาวิทยาลัยเข้าด้วยกันโดยมีศูนย์กลางอยู่ที่เกาะ Oahu การส่งข้อมูล จะใช้สัญญาณวิทยุ 2 ช่อง ช่องหนึ่งสำหรับการส่งจาก Terminal ไปยัง host และอีกช่องหนึ่งสำหรับส่งจาก host ไปยัง Terminal กรณีที่ host ส่งข้อมูลไปยัง Terminal Terminal ใดที่มี address ตรงตามที่ต้องการเท่านั้นจะเก็บเอาข้อมูลไว้ ไม่มีปัญหาของการใช้ตัวกลางร่วมกัน แต่หากเป็นกรณีที่ Terminal ต้องการส่งข้อมูลไปยัง host เป็นไปได้ที่ Terminal มากกว่า 1 แห่งส่งข้อมูลไปพร้อมกัน ข้อมูลจะถูกทำลายไปเราเรียกปรากฏการณ์นี้ว่า *การชนกันของข้อมูล (Collision)* ปัญหานี้แก้ไขได้โดยให้ Terminal รอรับสัญญาณตอบรับจาก host หลังการส่งข้อมูลทุกครั้ง ถ้าไม่ได้รับสัญญาณตอบรับภายในเวลาที่กำหนดจาก host แสดงว่าการส่งข้อมูลเกิดความผิดพลาด ต้องรอเวลาการส่งสัญญาณครั้งใหม่

3.3.2 Carrier Sense Multiple Access (CSMA)

carrier sense คือการตรวจสอบสถานะของ Network ก่อนการส่งข้อมูล ในกรณีนี้จะเกิด Collision เฉพาะกรณีที่มี node มากกว่า 1 node เริ่มส่งข้อมูลพร้อมกันเท่านั้น

3.3.3 Carrier Sense Multiple Access With Collision Detect (CSMA/CD)

CSMA/CD เป็นมาตรฐานการส่งข้อมูลแบบหนึ่งที่กำหนดโดย IEEE มาตรฐานนี้คือ **IEEE Standard 802.3** CSMA/CD ทำงานได้โดยการให้ node ตรวจสอบสถานะของ Network ขณะที่มีการส่งข้อมูลได้ ถ้าหากมี Collision เกิดขึ้น node ที่กำลังติดต่อกันจะป้องกันสัญญาณที่ผิดไปจากเดิมใน Network ถ้า Collision เกิดในช่วงเวลาสั้นๆ เป็นไปได้ว่า node บาง node ไม่สามารถป้องกันการเกิด Collision ได้ เพื่อลด Collision node ที่เกิด Collision จะมีวิธีการที่เรียกว่า *collision consensus enforcement* เป็นการสุ่มข้อมูลเป็นจำนวน byte แล้วส่งการติดต่อถึงกัน เราเรียกข้อมูลที่สุ่มขึ้นด้วยวิธีการนี้ว่า *jam* ข้อมูลที่ว่านี้จะต้องเกิดขึ้นนานเพียงพอจะทำให้ Collision หายไป ระยะเวลาดังกล่าวขึ้นอยู่กับระยะเวลาในการส่งสัญญาณจากปลายข้างหนึ่งของ Network ไปยังปลายอีกข้างหนึ่ง (เรียกว่า *end-to-end propagation delay*) เพื่อลดความสิ้นเปลืองของ Bandwidth ระยะเวลาดังกล่าวจะต้องสั้นที่สุดเท่าที่เป็นไปได้

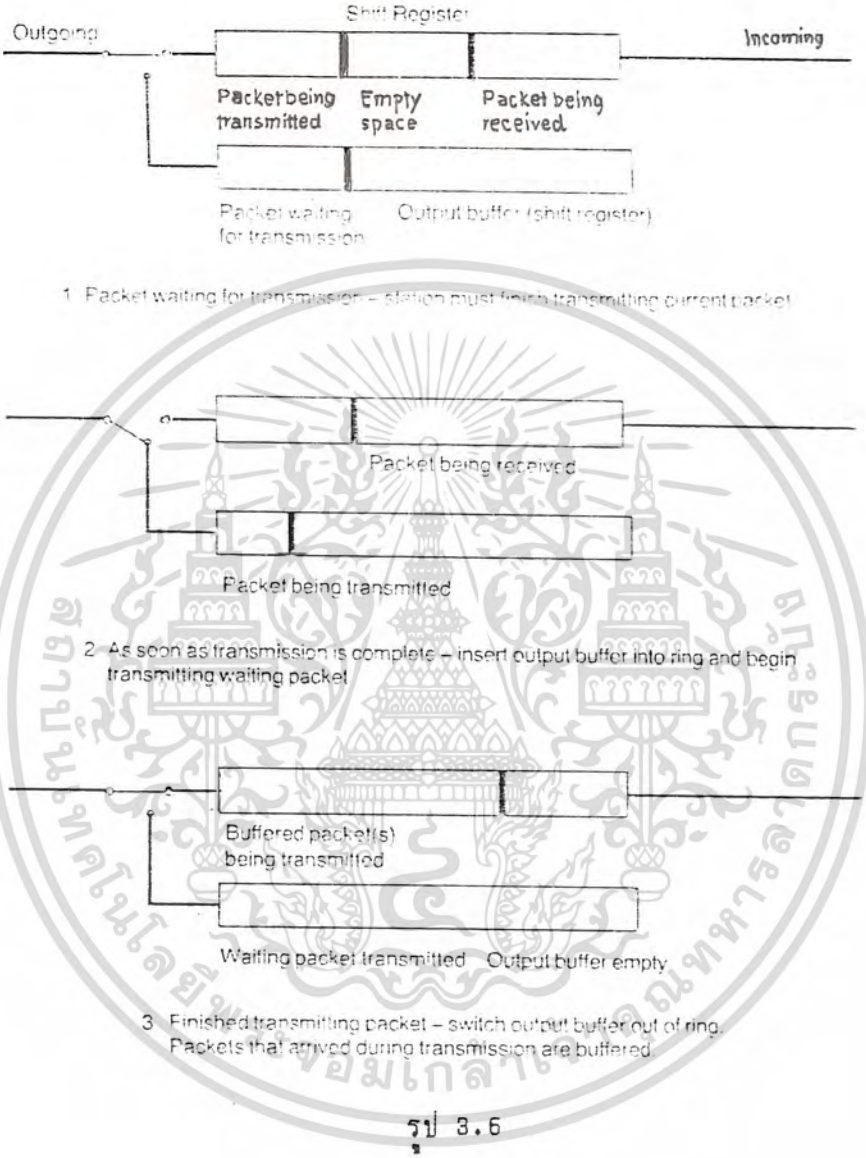


รูป 3.5

3.3.4 Register Insertion

register insertion เป็นเทคนิคที่ใช้กันใน Topology แบบ ring โดยแต่ละ node จะประกอบด้วย register 2 ชุด ดังรูป 3.6 ชุดแรกต่ออนุกรมเข้ากับวงแหวน เมื่อข้อมูลส่งผ่านเข้ามาก็จะถูกส่งเข้ามาใน register นี้ address ในข้อมูลจะถูกเปรียบเทียบกับ address ของ node ถ้าตรงกันก็จะทำการเก็บข้อมูลไว้ ถ้าไม่ตรงก็จะส่งต่อไปยัง node ต่อไปตามลำดับ register ชุดที่สองเรียกว่า *output buffer* ใช้ในการส่งข้อมูลของแต่ละ node ข้อมูลที่ต้องการส่งจะถูก load เก็บไว้ใน register ชุดนี้ และจะรอจนกว่าเงื่อนไขในการส่งข้อมูลทั้งหมดเป็นจริง ประการแรกคือ *shift register* จะต้องส่งข้อมูลที่มีอยู่เสร็จเรียบร้อย ประการที่สองคือ จะต้องมีเนื้อที่ว่างพอเพียงสำหรับข้อมูลที่เข้ามาใหม่ใน *input register* ขณะที่ *output buffer* กำลังส่งข้อมูล เมื่อเงื่อนไขเป็นจริง *output line* จะถูกสลับจาก input register เป็น output register จากนั้นจึง

แทรกข้อมูลเหล่านี้ลงในวงแหวน ถ้ามีข้อมูลใน input buffer ที่ต้องการส่งออก output line จะถูกสลับกลับมาที่ input buffer วิธีการนี้มีข้อดีเหมือนกับวิธีการของ CSMA/CD คือ ส่งข้อมูลได้เร็วในขณะที่มีข้อมูลในการส่งน้อยๆเท่านั้น



รูป 3.6

3.4 Noncontention Access Method

3.4.1 Sloted Rings

sloted rings เป็น Topology แบบ ring มีลักษณะของการส่งข้อมูลเป็น slot ที่กำหนดไว้ว่า ว่าง หรือไม่ว่าง slot ที่ไม่ว่างจะประกอบไปด้วย Packet ของข้อมูลที่ส่งจาก node หนึ่งไปยังอีก node หนึ่ง ขณะเริ่มต้นระบบ Terminal หนึ่งในวงแหวนจะสร้าง slot ว่างขึ้นมาอย่างน้อย

1 slot slot นี้จะถูกส่งไปรอบๆวงแหวนจนกว่าจะถึง node ที่ต้องการส่งข้อมูล จากนั้น node ที่ส่งข้อมูลจะทำเครื่องหมายแสดงว่า slot ไม่ว่าง แล้วใส่ข้อมูลไว้ Packet ของข้อมูลจะถูกส่งไปยังปลายทาง node ปลายทาง เมื่อรับข้อมูลเรียบร้อยจะแสดงเครื่องหมายไว้ที่ปลายของ Packet ว่าข้อมูลได้รับเรียบร้อยแล้ว เมื่อข้อมูลกลับมายังผู้ส่งอีกครั้ง node ต้นทางจะ set สถานะของ slot ให้เป็น slot ว่าง และส่งต่อไปตามทางเพื่อให้ node อื่นใช้งานต่อไป ในการส่งข้อมูล แต่ละ node จะทำการหน่วงเวลาในการส่งข้อมูล โดยที่เวลารวมของการหน่วงจะต้องนานอย่างน้อยเท่ากับเวลาในการส่งข้อมูล มิฉะนั้น ส่วนหัวของ Packet จะกลับมายัง node ต้นทางก่อนที่การรับส่งข้อมูลจะเสร็จสมบูรณ์ ตามปกติจะมี *delay buffer* อยู่ในวงแหวนเพื่อป้องกันเหตุการณ์นี้

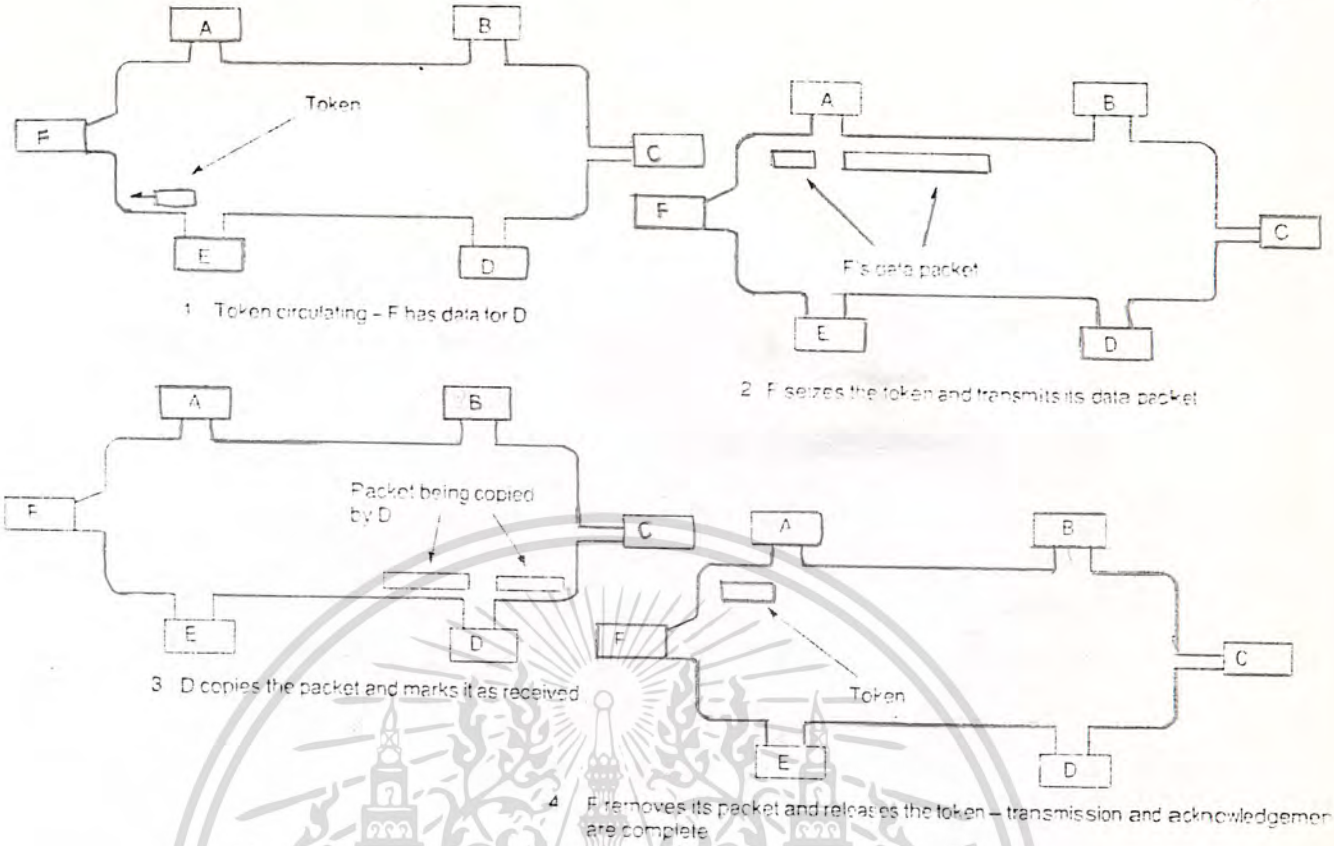
ปัญหาที่เกิดขึ้นกับ slot ring คือ อาจจะไม่มี slot ว่างในระบบเลยตลอดเวลา เกิดขึ้นจากการที่ เมื่อ node ได้รับข้อมูลแล้วไม่ได้ set ให้ slot เป็น slot ว่าง กรณีนี้แก้ไขได้โดยการกำหนดให้ node ใด node หนึ่งในระบบ แก้ไข slot ให้เป็น slot ว่าง หาก slot นั้นเคลื่อนที่ผ่าน node ดังกล่าวมากกว่า 1 ครั้ง

3.4.2 Token Passing

token passing อาศัยหลักการส่งข้อมูลลักษณะพิเศษไปตามลำดับในวงแหวน เราเรียกการส่งข้อมูลดังกล่าวว่า *Token* หากมี node ที่ต้องการส่งข้อมูลก็จะส่งสัญญาณตอบรับสัญญาณ token แต่ถ้าไม่ต้องการส่งข้อมูลก็จะผ่าน token ไปให้ node ถัดไป ทฤษฎีนี้สามารถใช้กับ Topology ได้หลายแบบ แต่ในที่นี้จะกล่าวถึงเฉพาะแบบ ring และ แบบ bus เท่านั้น

3.4.2.1 Token-passing Ring

token-passing ring เป็นการส่งข้อมูลตามมาตรฐาน IEEE 802.5 มีหลักการทำงานดังนี้คือ token จะถูกส่งไปรอบๆวงแหวน ถ้ามี node ใดต้องการส่งข้อมูลก็จะรับ token ไว้ แล้วส่งข้อมูลไปตามสายส่ง รูป 3.7 node F ต้องการส่งข้อมูลไปยัง node D node F จะรับ token ไว้ token จะหายไป จากนั้นจึงส่งข้อมูลผ่าน node A ,B ,C ไปยัง node D node D จะเก็บข้อมูลไว้ และตอบรับการรับข้อมูลแล้วส่งกลับไปยัง node F จากนั้น node F ก็จะสร้าง token ใหม่ขึ้นมา ถ้า node ใดมีข้อมูลที่ต้องการส่ง ก็จะต้องรอจนกว่า token จะมาถึง ถ้ามีการส่งข้อมูลในระบบมากๆ node อาจต้องรอ token นาน ดังนั้น token ต้องให้โอกาสที่ทุกๆ node ได้ส่งข้อมูลเท่าๆกัน

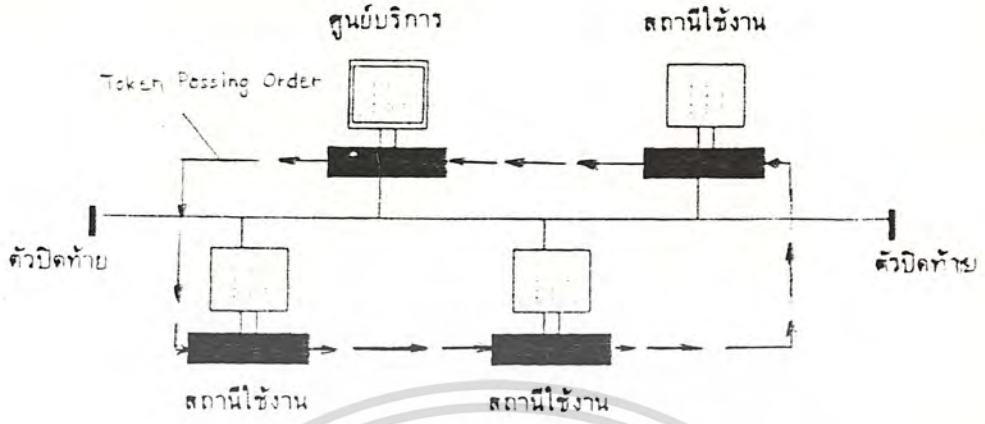


รูป 3.7

3.4.2.2 Token-passing Bus

token-passing bus เป็นการส่งข้อมูลตามมาตรฐาน IEEE 802.4 แต่ละ node เรียงกันเป็นเส้นตรง token จะถูกเจาะจงส่งไปยัง node ที่ต้องการเราเรียก token แบบนี้ว่า *explicit token* ซึ่งต้องเพิ่มการกำหนด address ก่อนการส่งข้อมูล node ที่เกี่ยวข้องต้องถูกกำหนด address โดย node ก่อนหน้าในลำดับ ด้วยวิธีนี้ node ใน token bus จะถูกเรียงลำดับเสมือนกับเป็นวงแหวน (*logical ring*) ดังรูป 3.8 การทำงานในแบบ token bus คล้ายกับการทำงานในแบบ token ring แตกต่างกันตรงที่ว่าการเรียงลำดับของ node ใน logical ring ไม่จำเป็นต้องเป็นไปตามการเรียงลำดับใน bus หมายความว่า node ที่เกี่ยวข้องกับ bus อาจไม่เป็นส่วนหนึ่งของ logical ring ก็ได้ คุณสมบัติข้อนี้ทำให้สามารถประยุกต์เพื่อใช้งานในลักษณะพิเศษได้ แต่จะเป็นปัญหาในการลดเพิ่ม node ในระบบ

เมื่อระบบเริ่มทำงาน จะไม่มี token อยู่ในระบบ node ใด node หนึ่งในระบบจะต้องสร้าง token ขึ้นมา การเลือกว่า node ใดจะได้สร้าง token ขึ้นอยู่กับ address ของ node นั้น node ที่มี address สูงสุดจะสร้าง token ได้ก่อน ช่วงเวลานี้ logical ring จะประกอบด้วย node เพียง node เดียว node อื่นๆจะเริ่มเข้ามาเป็นส่วนหนึ่งของ logical ring เมื่อไม่มี node ใดใน logical ring ใช้งาน



รูป 3.8

การทำงานของ token-passing bus บางครั้งก็มีการใช้เทคนิค CSMA/CD เข้ามาช่วย อย่างไรก็ตาม วิธีของ explicit token-passing จะทำให้เกิดความลื่นเปลืองของ *delay transmittion* และทำให้เกิด *throughput* ได้

3.5 มาตรฐานสำหรับอนาคต

มาตรฐานของฮาร์ดแวร์บนระบบเครือข่ายท้องถิ่นแต่ละแบบ มีคุณสมบัติที่เหมาะสมตามสภาพแวดล้อมที่ต่างกันออกไป โปรโตคอล CSMA/CD เป็นแบบที่ค่อนข้างง่าย ทำงานได้ดีที่ปริมาณการใช้งานของระบบเครือข่ายต่ำถึงปานกลาง และได้ถูกนำไปใช้อย่างกว้างขวาง โดยผู้ผลิตหลายรายเป็นสื่อข้อมูลหลายๆแบบ

ค่าใช้จ่ายในการติดตั้งที่ต่ำของระบบเครือข่ายแบบ StarLAN ก็เป็นข้อหนึ่งในการเลือกใช้กับระบบเครือข่ายขนาดเล็ก แต่ถ้ามีการใช้งานในระบบมาก ระบบเครือข่ายแบบ token จะมีประสิทธิภาพสูงกว่า ข้อดีของระบบเครือข่ายแบบ token ก็คือการตอบรับของสถานีเป็นแบบ deterministic ซึ่งสถานีส่งข้อมูลจะรอเป็นเวลาหนึ่งที่แน่นอนก่อนที่จะมีการส่งข้อมูล ในทางกลับกันเวลาในการรอของสถานีใน CSMA/CD จะเป็นแบบ statistically เนื่องจากทุกครั้งที่มีความพยายามส่งข้อมูลอาจเกิดการชนกันได้ ในกรณีที่เลวร้ายที่สุดสถานีจะถูกบล็อกอย่างไม่มีกำหนด สถาปัตยกรรมแบบ token จะอนุญาตให้สถานีกำหนดระดับในการใช้สายส่งข้อมูลได้ นอกจากนี้ระบบเครือข่ายแบบ token ยังไม่จำกัดขนาดเล็กที่สุดของกลุ่มข้อมูลสื่อสารอีกด้วย ข้อเสียของระบบเครือ

ข่ายแบบ token ก็คือ ความซับซ้อนของมันเช่น การกู้คืนจากข้อผิดพลาดหรือการจัดกระบวนการเริ่มต้นของวงแหวน เป็นต้น

การขยายการสนับสนุนในมาตรฐานของฮาร์ดแวร์ ทำให้ผู้ผลิตซอฟท์แวร์ระบบเครือข่ายท้องถิ่นได้สร้างระบบปฏิบัติการ ที่สามารถทำงานได้บนระบบเครือข่ายมาตรฐาน เช่น NetWare ของ Novell สนับสนุน ARCnet และมาตรฐาน 802 ทั้งหมด, 3Plus ของ 3Com สนับสนุนทั้ง Ethernet และ Token-Ring ของ IBM ในทางตรงกันข้าม PC LAN ของ IBM ทำงานได้เฉพาะบน Token-Ring และระบบเครือข่ายของ IBM เท่านั้น การเลือกระบบปฏิบัติการของระบบเครือข่ายท้องถิ่นที่สนับสนุนหลายๆมาตรฐาน จะช่วยให้สามารถใช้ซอฟท์แวร์ตัวเดียวกันในระบบเครือข่ายหลายๆแบบได้ และยังช่วยให้สามารถเลือกฮาร์ดแวร์ที่เหมาะสมได้อีกด้วย

บทที่ 4
การใช้งานระบบเครือข่ายแบบต่างๆ

4.1 การใช้งานระบบเครือข่ายแบบ Ethernet

ระบบเครือข่ายแบบ Ethernet เป็นการใช้งานระบบเครือข่ายตามมาตรฐาน IEEE 802.3 ที่มีการส่งข้อมูลแบบเบสแบนด์ความเร็ว 10 เมกกะบิทต่อวินาที มี Topology แบบบัส และใช้โปรโตคอลแบบ CSMA/CD ข้อมูลที่ถูกส่งในระบบเครือข่าย Ethernet จะส่งเป็นกลุ่มข้อมูลสื่อสารของเฟรม ซึ่งรูปแบบของเฟรมมีตามรูปที่ 4.1

Preamble	Destn. Address	Source Address	Type Field	Data Field	Frame Check Sequence
8 bytes	6 bytes	6 bytes	2 bytes	46-1500 bytes	4 bytes

Ethernet Frame Format

Preamble	Start Delm.	Destn. Address	Source Address	Type Field	Data Field	Pad Byte	Frame Check Sequence
7 bytes	1 bytes	2 or 6 bytes	2 or 6 byter	2 bytes	0-n bytes	0-p bytes	4 bytes

IEEE 802.3 Frame Format

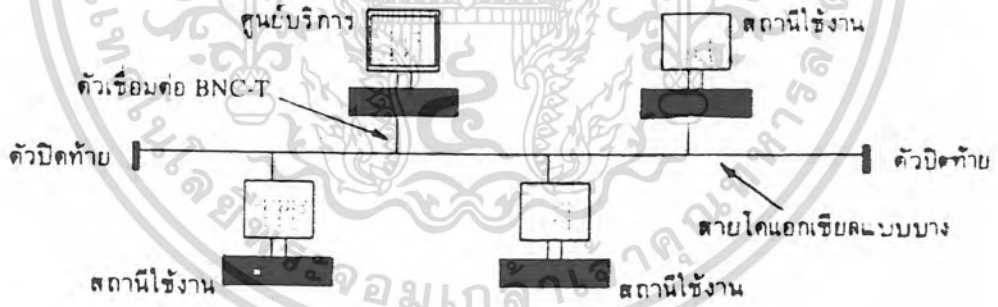
รูป 4.1

อุปกรณ์แต่ละตัวในระบบเครือข่าย Ethernet จะมีตำแหน่งของอุปกรณ์ขนาด 48 บิท ซึ่งตำแหน่งของอุปกรณ์นี้สร้างมาจากโรงงานเลย โดย IEEE ในนิวยอร์ค สหรัฐอเมริกาจะควบคุมการกำหนดตำแหน่งเหล่านี้

เมื่ออุปกรณ์นี้ได้ส่งกลุ่มข้อมูลสื่อสารไปในระบบเครือข่ายแล้ว จะรออย่างน้อย 9.6 ไมโครวินาที ก่อนจะส่งข้อมูลอีก ซึ่งช่วงเวลานี้เรียกว่าเป็น Inter-Packet-Gab (IPG) ซึ่ง IPG จะช่วยให้อุปกรณ์รับส่งมีเวลาเตรียมพร้อมที่จะรับกลุ่มข้อมูลสื่อสารโดยที่ไม่เกิดปัญหา ถ้าอุปกรณ์พยายามส่งข้อมูลและพบว่าสายไม่ว่างมันจะรอ ถ้าหลังจากที่ส่งกลุ่มข้อมูลสื่อสารไปแล้วเกิดการชนกันขึ้น อุปกรณ์จะรอเป็นช่วงเวลาแบบสุ่มก่อนที่เริ่มการส่งข้อมูลสื่อสารที่ชนกันนั้นใหม่ อุปกรณ์รับส่งจะพยายามส่งสูงสุด 16 ครั้ง ถ้ายังส่งไม่ได้จะรายงานว่าเกิดข้อผิดพลาดขึ้น

ความต้องการพื้นฐานของระบบเครือข่าย Ethernet ก็คือเมื่อเกิดการชนกันของข้อมูลขึ้น ทุกๆสถานีจะต้องรู้ถึงการชนกันนั้น ก่อนที่สถานีที่เกี่ยวข้องกับการชนนั้นจะส่งเฟรมขนาดต่ำสุดคือ 572 บิทออกมา ถ้ากฎเกณฑ์เหล่านี้ไม่ดีพอแล้ว อาจทำให้บางสถานีรับข้อมูลได้แต่บางสถานีไม่อาจรับได้ ข้อมูล LLC จะมีความยาว 368 บิท (46 ไบท์) และ 12,000 บิท (1,500 ไบท์) ถ้าข้อมูลมีขนาดสั้นเกินไปจะถูกเติมให้ครบ 368 บิท ด้วย 0 ซึ่งจะหมายความว่า ถ้าทุกกลุ่มข้อมูลสื่อสารใช้ขนาดต่ำสุดในการส่งแล้วจะสามารถส่งได้ด้วยความเร็ว 7,619,047 บิทต่อวินาที หรือส่งด้วย utilization 76% ส่วนที่เหลือในกลุ่มข้อมูลสื่อสารของ Ethernet ก็คือ IPG และ overhead อื่นๆ ถ้ากลุ่มข้อมูลสื่อสารใช้ขนาดสูงสุดในการส่งแล้วจะสามารถส่งได้ด้วยความเร็ว 9,523,212 บิทต่อวินาที หรือมี utilization 95.25%

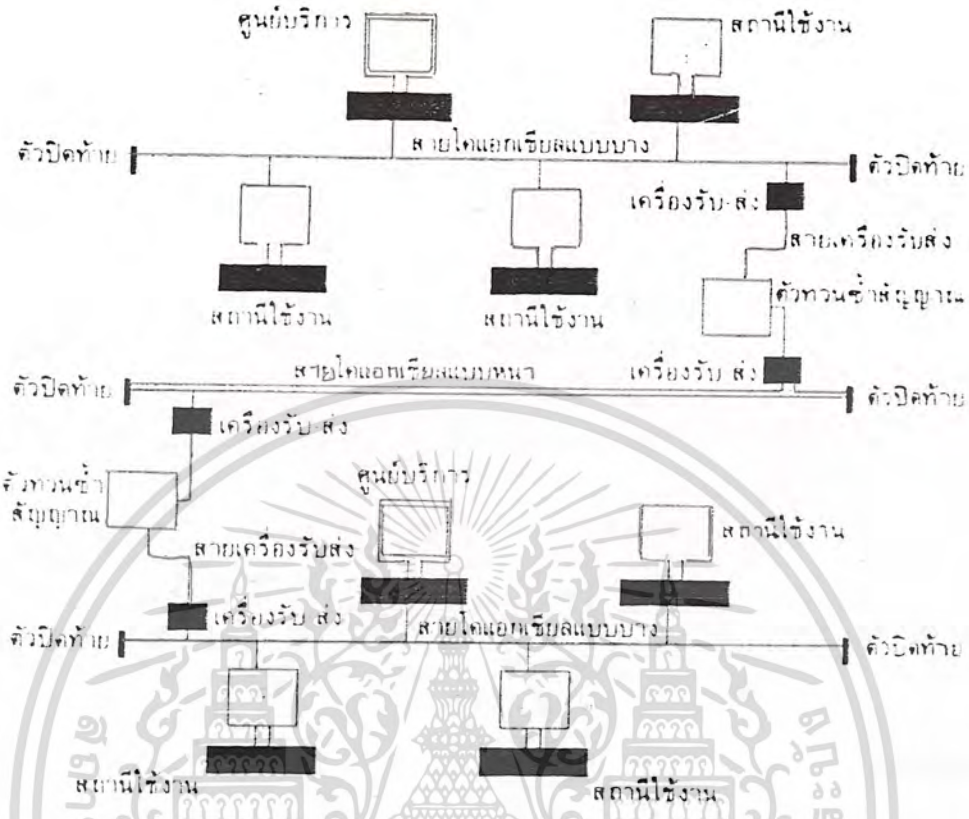
ระบบเครือข่ายแบบ Ethernet ใช้สายส่งข้อมูลแบบโคแอกเชียลได้ 2 แบบ คือแบบบาง (RG-58, 50 โอห์ม) ใช้กับเซกเมนต์ที่มีความยาวน้อยกว่า 185 เมตร และแบบหนา (RG-11, 50 โอห์ม) ซึ่งสามารถติดตั้งในความยาวถึง 500 เมตร ถ้าใช้อุปกรณ์เช่น ตัวทวนซ้ำสัญญาณ (repeater) ก็สามารถจะเพิ่มความยาวในการใช้งานของสายโคแอกเชียลออกไปได้ถึง 3000 เมตร



รูป 4.2 การใช้งานระบบ Ethernet โดยใช้สายโคแอกเชียลแบบบาง

สายโคแอกเชียลแบบบางถูกออกแบบมา สำหรับใช้กับระบบเครือข่ายท้องถิ่นภายในสำนักงาน อุปกรณ์ต่างๆในระบบจะต่อเข้ากับสายโคแอกเชียลแบบนี้ โดยใช้อุปกรณ์ BNC T-connector แต่ละสถานีจะต้องห่างกันอย่างน้อย 1 เมตร เพื่อป้องกันไม่ให้สัญญาณรบกวนกันเอง และปลายทางเซกเมนต์จะต้องปิดด้วยตัวเทอร์มิเนเตอร์ BNC แบบ 50 โอห์ม ถ้าใช้โคแอกเชียลแบบบางจะต่อสถานีได้

สูงสุด 30 สถานี ดังรูป 4.2



รูป 4.3 การใช้งานระบบ Ethernet โดยใช้สายโคแอกเชียลแบบหนา

สายโคแอกเชียลแบบหนาหรือที่เรียกว่า backbone Ethernet ใช้งานได้ในระยะไกลกว่าสายโคแอกเชียลแบบบาง และใช้ร่วมกับสายโคแอกเชียลแบบบางได้ด้วย ระยะห่างระหว่างแต่ละสถานีต้องไม่ใกล้กันจนเกินไป เพื่อป้องกันการเกิดการรบกวนกันเองของสัญญาณ สถานีแต่ละสถานีจะต่อกับสายโคแอกเชียลแบบหนาโดยใช้เครื่องรับ-ส่ง (transceiver) ซึ่งระยะห่างระหว่างเครื่องรับ-ส่งแต่ละตัวจะต้องไม่ต่ำกว่า 2.6 เมตร ระยะทางจากเครื่องรับ-ส่งถึงสถานีแต่ละสถานีจะต่อโดยใช้สายของเครื่องรับ-ส่งหรือที่เรียกว่าสาย "AUI" (Attachment Unit Interface) และต้องยาวไม่เกิน 50 เมตร สายโคแอกเชียลแบบหนาจะต่อเครื่องรับส่งได้สูงสุดถึง 100 ตัว (ดูรูป 4.3)

เราสามารถขยายระบบเครือข่ายแบบ Ethernet โดยใช้ตัวทวนซ้ำสัญญาณได้ แต่จำนวนตัวทวนซ้ำสัญญาณระหว่างสองสถานีใดๆจะต้องไม่เกิน 2 ตัว (หรือ 4 ตัว ถ้าใช้ร่วมกับเซกเมนต์ของเครือข่ายแบบ Inter-Repeater-Link) ตัวทวนซ้ำสัญญาณจะถูกต่อเข้ากับสายโคแอกเชียลโดยผ่านทางเครื่องรับ-ส่ง และต่อที่จุดใดก็ได้ที่ต่อเครื่องรับ-ส่งได้ ตัวทวนซ้ำสัญญาณแบบหลายช่องทาง (Multiport Repeater) เช่น DEMPR ของบริษัท Digital Equipment จะต่อสายโคแอกเชียล

แบบบางได้ถึง 8 เซกเมนต์ด้วยกันใน Topology แบบดาว และถ้าใช้ตัวทวนซ้ำสัญญาณร่วมกับสายแบบเส้นใยนำแสงทำให้ความยาวของแต่ละเซกเมนต์เพิ่มเป็น 1000 เมตร

การเชื่อมต่อทางกายภาพของ Ethernet

1. *ระบบเครือข่าย Ethernet ที่ใช้สายส่งข้อมูลแบบบาง*

อาร์ดแวร์ที่ใช้ในระบบเครือข่ายที่ใช้สายส่งข้อมูลแบบบางมีดังนี้

Network Interface Board (NIB) เป็นแผงวงจรที่สถานีใช้งานแต่ละตัว ที่ต่อกับสายส่งข้อมูลเพื่อส่งข้อมูลในระบบ

BNC Connectors ตัวเชื่อมระหว่าง NIB กับสายส่งข้อมูล

Thin Ethernet Cable สาย Ethernet แบบบางเป็นสายส่งข้อมูลแบบ 0.2 นิ้ว RG-58 A/U 50 โอห์ม

BNC Barrel Connectors ใช้ต่อสาย Ethernet แบบบาง 2 เส้นเข้าด้วยกัน

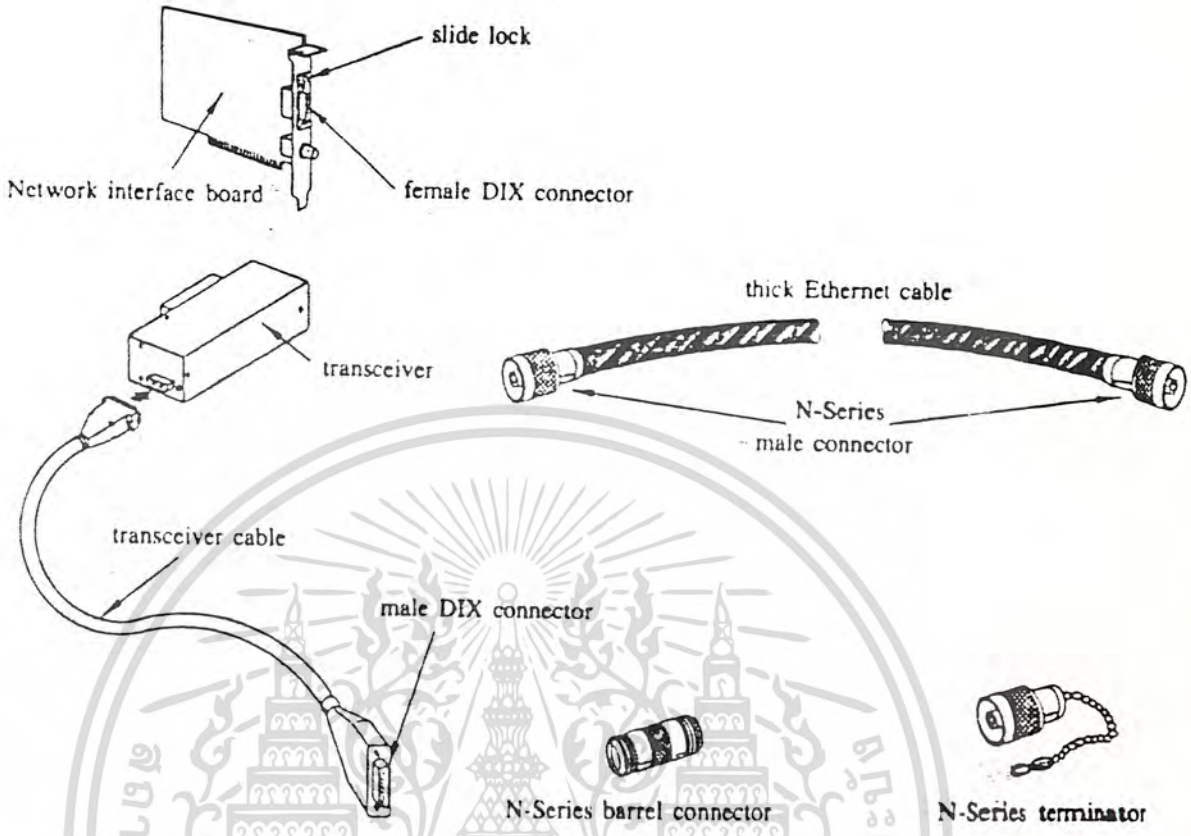
BNC T-Connectors คล้ายกับ BNC Barrel Connectors โดยที่ปลาย 2 ด้านใช้ต่อสาย Ethernet แบบบางสองเส้นเข้าด้วยกัน และปลายอีกด้านหนึ่งต่อเข้ากับ NIB

BNC Terminator เป็นเทอร์มิเนเตอร์แบบ 50 โอห์ม ที่ใช้ป้องกันการรบกวนของสัญญาณไฟฟ้า ใช้ที่ปลายทั้งสองด้านของสายส่งข้อมูล

ตัวเครื่องรับ-ส่งของ Ethernet แบบบางจะอยู่ในตัวเชื่อมแบบ BNC ถ้าใช้สายส่งข้อมูลแบบ 10 BASE 2 จะต้องไม่มีสายต่อระหว่างตัวเครื่องรับ-ส่งและเซกเมนต์ของระบบเครือข่าย นอกจากนี้ความยาวสุงสุดของ BNC แบบ T ต้องไม่เกิน 40 มิลลิเมตร

ข้อจำกัด

- จำนวนสูงสุดของทรังก์เซกเมนต์ : 5
  (3 เซกเมนต์ + inter-repeater link อีก 2 เซกเมนต์)
- ความยาวสูงสุดของทรังก์เซกเมนต์ : 185 เมตร
- ความยาวสูงสุดของทั้งระบบ : 925 เมตร
- จำนวนสถานีใช้งานสูงสุดในหนึ่งทรังก์เซกเมนต์ : 30
- ระยะทางต่ำสุดระหว่าง T-Connector : 0.5 เมตร



รูป 4.4 อาร์ดแวร์ที่ใช้ในระบบเครือข่ายที่ใช้สายส่งข้อมูลแบบบาง

2. *ระบบเครือข่าย Ethernet ที่ใช้สายส่งข้อมูลแบบหนา*

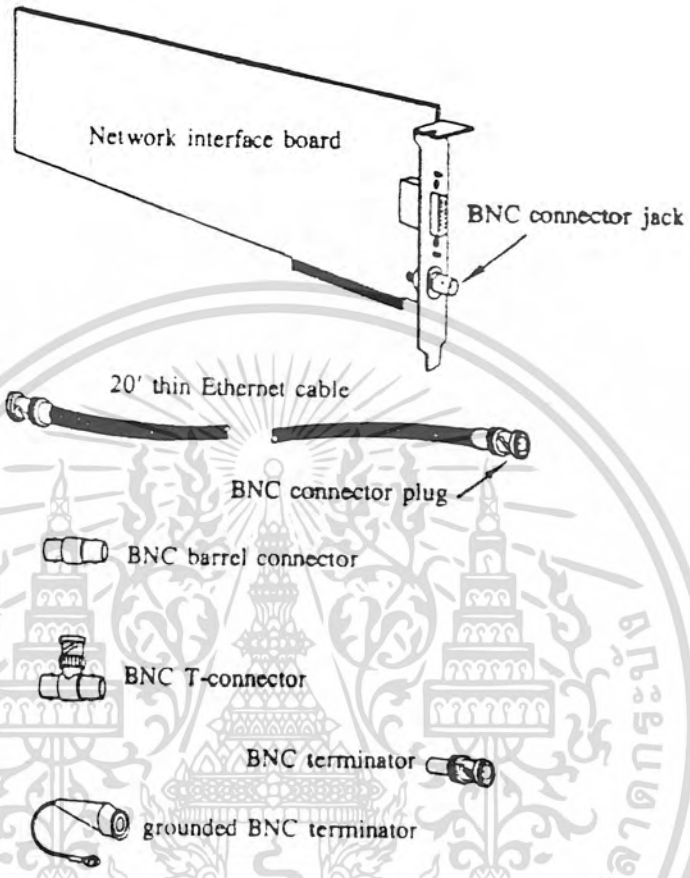
อาร์ดแวร์ที่ใช้ในระบบเครือข่ายที่ใช้สายส่งข้อมูลแบบบางมีดังนี้

Network Interface Board (NIB) เป็นแผงวงจรที่สถานีใช้งานแต่ละตัว ที่ต่อกับสายส่งข้อมูลเพื่อส่งข้อมูลในระบบ

Transceivers Medium Attached Unit (MAU) สถานีใช้งานในระบบที่ใช้สายส่งข้อมูลเป็นแบบหนา จะติดต่อกับระบบเครือข่ายผ่าน เครื่องรับ-ส่งที่ต่ออยู่กับสายส่งข้อมูลหลักของระบบ

Transceiver Cable หรือ *Attached Unit Interface (AUI)* เป็นสายคู่บิดเกลียว 4 เส้น หุ้มด้วยฉนวนอีกชั้นหนึ่ง ใช้ในการต่ออุปกรณ์ของระบบเครือข่าย เข้ากับเครื่องรับ-ส่ง

DIX Connector DIX Connector ใช้ต่อระหว่าง NIB กับสายเครื่องรับ-ส่งสัญญาณ

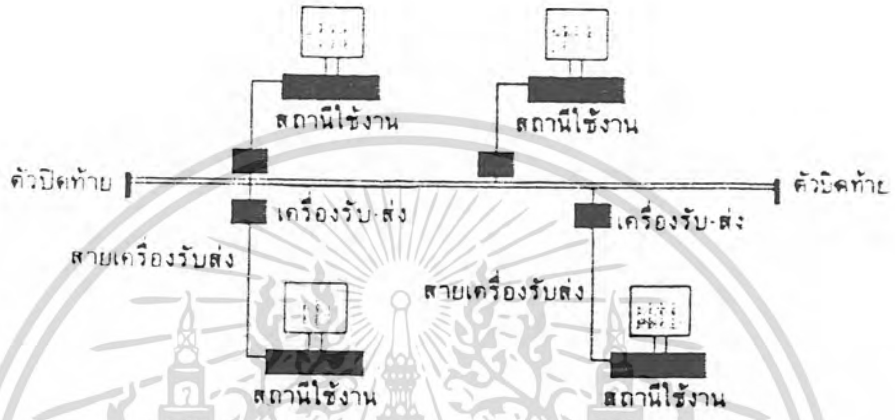


รูป 4.5 อาร์ดแวร์ที่ใช้ในระบบเครือข่ายที่ใช้สายส่งข้อมูลแบบหนา

Thick' Ethernet' Cable	สาย Ethernet แบบหนาเป็นสายส่งข้อมูลโคแอกเชียล ชนิด 50 โอห์ม ขนาดเส้นผ่านศูนย์กลาง 0.4 นิ้ว
N-Series Male Connectors	เป็นตัวเชื่อมต่อที่อยู่ด้านปลายสายส่งข้อมูล
N-Series Barrel Connector	ใช้ต่อสาย Ethernet แบบหนาสองเส้นเข้าด้วยกัน
N-Series Terminators	เป็นเทอร์มิเนเตอร์ชนิด 50 โอห์ม ที่ใช้ปิดปลายทั้งสองด้านของสายส่งข้อมูลเพื่อป้องกันสัญญาณรบกวน

Fan-Out Unit เป็นอุปกรณ์ที่ใช้ต่อกับตัวเครื่องรับ-ส่ง เพื่อช่วยให้ต่ออุปกรณ์ของระบบเครือข่ายหลายๆตัว (8 ตัว) เข้ากับเครื่องรับ-ส่งตัวเดียวได้

Inter-Repeater Link (IRL) เป็นสายส่งข้อมูลที่ไม่มีอุปกรณ์ของระบบเครือข่ายต่ออยู่



รูป 4.6 การเชื่อมต่อทางกายภาพของ Ethernet ที่ใช้สายส่งข้อมูลแบบหนา

ระบบเครือข่ายแบบนี้สถานีใช้งานจะต่อกับตัวเครื่องรับ-ส่ง โดยผ่านสาย AUI ซึ่งสาย AUI จะต้องมีความยาวไม่เกิน 50 เมตร และตัวเครื่องรับ-ส่งจะต่อกับสายส่งข้อมูลหลักโดยผ่านทาง Physical Medium Attachment

ข้อจำกัด
- จำนวนสูงสุดของทรังก์เซกเมนต์ : 5
- ความยาวสูงสุดของทรังก์เซกเมนต์ : 500 เมตร
- ความยาวสูงสุดของทั้งระบบ : 3,000 เมตร
  (3 เซกเมนต์ (3*500) = 1500 เมตร
    2 inter-repeater links (2*500) = 1000 เมตร
    สาย AUI 10 เส้น (10*50) = 500 เมตร)
- จำนวนสถานีใช้งานสูงสุดในหนึ่งทรังก์เซกเมนต์ : 100
- จำนวนสถานีใช้งานสูงสุดในทั้งระบบ : 1024
- ระยะทางต่ำสุดระหว่าง transceivers : 2.5 เมตร

- ความยาวสูงสุดของสายเครื่องรับ-ส่ง : 50 เมตร

3. *ระบบเครือข่าย Ethernet ที่ใช้สายส่งข้อมูลทั้งสองแบบ*

ระบบเครือข่ายที่ใช้สายส่งข้อมูลทั้งแบบบางและแบบหนาร่วมกัน ใช้อาร์ดแวร์เหมือนกับที่ใช้ในระบบเครือข่ายที่ใช้สายส่งข้อมูลแบบบางและแบบหนา นอกจากนี้ต้องเพิ่มตัวอะแดปเตอร์เพื่อต่อทรังก์เซกเมนต์ของทั้งสองแบบเข้าด้วยกัน อะแดปเตอร์มี 2 แบบ คือ BNC female to N-series female adaptor และ BNC female to N-series male adaptor

BNC female to N-Series female adapter

BNC female to N-Series male adapter

รูป 4.7

การใช้สายโคแอกเชียลแบบบางและแบบหนาร่วมกันในระบบเครือข่ายแบบ Ethernet นี้ ความยาวสูงสุดของแต่ละเซกเมนต์จะอยู่ระหว่าง ความยาวสูงสุดของสายส่งข้อมูลแบบหนาและแบบบาง (185 เมตร และ 500 เมตร ตามลำดับ) ความยาวสูงสุดของระบบจะลดลงเป็นสัดส่วนตามความยาวของสายส่งข้อมูลแบบบางที่ใช้ มีสัดส่วนดังนี้

(3.28 X thin) + thick < 500 เมตร

ความยาวของสายส่งข้อมูลแบบบางและที่สามารถใช้ในเซกเมนต์ได้ สามารถคำนวณได้ดังนี้

Max_Length = (500 - L) / 3.28

โดยที่ Max_Length เป็นความยาวสูงสุดของสายแบบบาง, L เป็นความยาวของทรังก์

เซกเมนต์ เช่น ถ้าต้องการต่อทรังก์เซกเมนต์ที่ยาว 300 เมตร และต้องการลดค่าใช้จ่ายในเรื่องฮาร์ดแวร์ จากสูตรสามารถคำนวณได้ดังนี้

(500 - 300) / 3.28 = 60.97 เมตร

ดังนั้นสามารถใช้สายแบบบางได้ยาว 60 เมตร ส่วนที่เหลืออีก 240 เมตร ต้องเป็นสายแบบหนา

4.2 การใช้งานระบบเครือข่ายแบบ ARCnet

ARCnet เป็นระบบเครือข่ายแบบ token-passing ที่พัฒนาขึ้นโดยบริษัท Datapoint กับ ระบบคอมพิวเตอร์ ARC ARCnet โดยมีความสัมพันธ์กับมาตรฐาน 802.4 token-passing bus แต่มาตรฐานของ ARCnet จะขึ้นกับบริษัทผู้ผลิตฮาร์ดแวร์ต่างๆ ARCnet ได้กลายมาเป็นการเชื่อมต่อของระบบเครือข่ายท้องถิ่นที่ได้รับความนิยมอย่างมากแบบหนึ่ง บอร์ดของ ARCnet จะใช้ไมโครชิฟที่ผลิตโดย Standard Microsystem Corporation (SMC) ARCnet พัฒนาขึ้นโดยใช้สายส่งข้อมูลแบบโคแอกเชียล และต่อมาสามารถใช้สายเส้นใยนำแสงในการส่งข้อมูลได้ ARCnet มีระบบปฏิบัติการสำหรับระบบเครือข่ายท้องถิ่นหลายอันที่สนับสนุนเช่น Advanced NetWare ของ Novell และ VINES ของ Banyan

ARCnet มีข้อดีหลายอย่างเช่น ง่ายต่อการติดตั้งและบำรุงรักษา และยังเป็นมาตรฐานที่ได้รับการสนับสนุนจากผู้ผลิตหลายราย

ตอนแรก ARCnet ออกแบบมาให้มีการต่อระบบเครือข่ายแบบโมดูลาร์โดยใช้อับ และสายส่งข้อมูลแบบโคแอกเชียล ซึ่งสามารถต่อเครื่องคอมพิวเตอร์ได้สูงสุด 255 เครื่องด้วยกัน ข้อมูลส่งที่อัตรา 2.5 เมกกะบิทต่อวินาที ในกลุ่มข้อมูลสื่อสารที่มีขนาดไม่เกิน 512 ไบท์ โดยใช้โปรโตคอลแบบ token-passing เหมือนกับ Token Ring ของ IBM และมีสถาปัตยกรรมแบบดาวกระจาย (distributed star) ในกลุ่มข้อมูลสื่อสารที่ส่งจะมี overhead อยู่ 4 ไบท์ ดังนั้นขนาดของข้อมูลสูงสุดที่จะส่งได้ในกลุ่มข้อมูลสื่อสารจะไม่เกิน 508 ไบท์ แต่ NetWare ของ Novell จะส่งกลุ่มข้อมูลสื่อสารแบบอ่านและเขียนขนาด 560 ไบท์ ไปยังศูนย์บริการข้อมูล ดังนั้นมันจะต้องส่งกลุ่มข้อมูลสื่อสารถึง 2 กลุ่มซึ่งในกลุ่มข้อมูลสื่อสารที่ 2 จะเพิ่มช่วงเวลาของ token แต่อย่างไรก็ตามระยะเวลาการตอบสนอง (response time) ก็ยังไม่สูงเกินข้อกำหนด ขนาดของกลุ่มข้อมูลสื่อสารที่ส่งจะถูกปรับให้เหมาะสมกับการใช้งานได้ นอกจากนี้ ARCnet ยังปรับตัวมันเองได้โดยอัตโนมัติ ถ้ามี

สถานีใดต่อเพิ่มหรือปลดออก ปิดหรือเปิดสถานีใหม่

แม้ว่า ARCnet จะมีการส่งข้อมูลที่ช้ากว่า Ethernet ซึ่งมีความเร็วถึง 10 เมกกะบิทต่อวินาที หรือระบบเครือข่ายแบบเส้นใยนำแสงซึ่งมีความเร็วถึง 100 เมกกะบิทต่อวินาที แต่ ARCnet ก็ยังคงอยู่ได้และในหลายๆกรณี ARCnet ก็ทำงานได้ดีกว่าระบบเครือข่ายแบบอื่นอีกด้วย

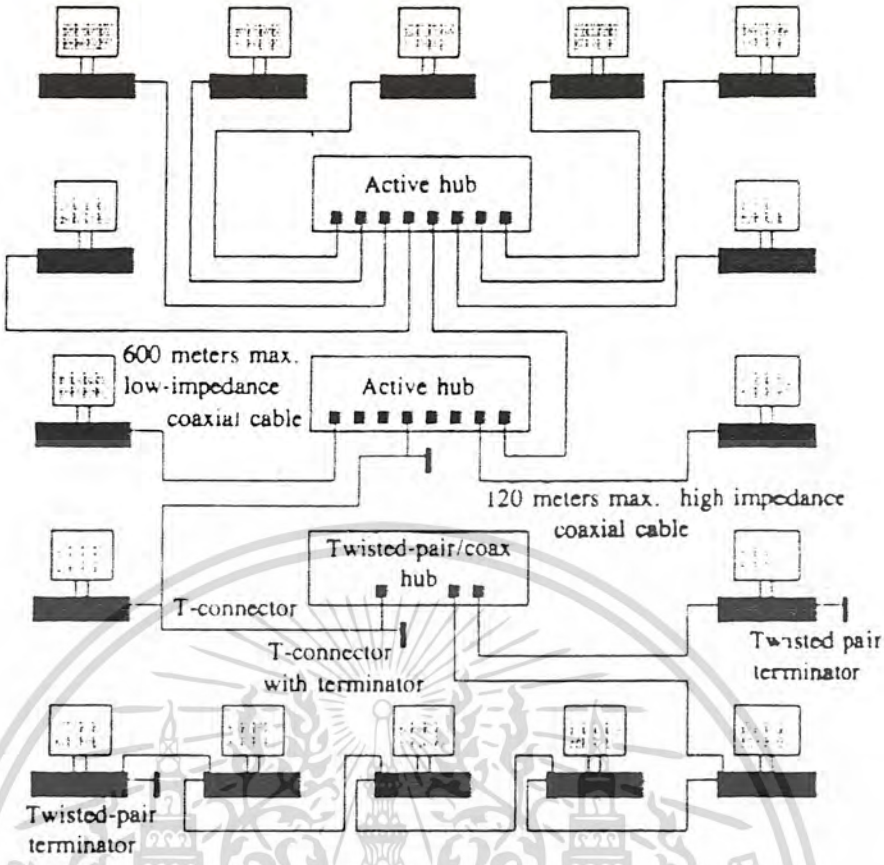
ARCnet อาจใช้ในระบบเครือข่ายท้องถิ่นขนาดใหญ่ที่มีเครื่องคอมพิวเตอร์มากกว่า 255 ตัวได้ ถ้าศูนย์บริการข้อมูลทำงานเป็นสะพานสื่อสารระหว่างกลุ่มทำงาน (workgroup) ซึ่ง ARCnet ยังใช้ได้ดีกับระบบเครือข่ายขนาดเล็กที่มีเครื่องคอมพิวเตอร์เพียง 2 ถึง 3 ตัว

ประสิทธิภาพรวมของ ARCnet มากกว่าของ StarLAN ซึ่งมีความเร็วเพียง 1 เมกกะบิทต่อวินาที แต่น้อยกว่า token ring ของ IBM ที่มีความเร็ว 4 เมกกะบิทต่อวินาที แต่ความกว้างแถบข้อมูลเป็นอีกเรื่องหนึ่ง Ethernet และ StarLAN ใช้โปรโตคอลแบบ CSMA/CD ซึ่งต่างกับวิธีการแบบ token-passing ที่ ARCnet และ token ring ใช้เป็นโปรโตคอล

โปรโตคอลแบบ token-passing จะมีประสิทธิภาพมากภายใต้การใช้งานหนัก หรือ heavy load

โครงสร้างของระบบ ARCnet จะเป็นต้นไม้ที่ไม่มีราก (ไม่มีลูป) แต่ละกิ่งก้านสาขาหรือเซกเมนต์ของ tree จะต่ออยู่กับฮับ ซึ่งฮับที่ใช้อาจเป็นได้ทั้ง active hub หรือ passive hub ดังรูป 4.8 active hub เป็นอุปกรณ์ซึ่งต้องมีไฟเลี้ยงที่ใช้ในการสร้างสัญญาณใหม่ และขยายสัญญาณเพื่อช่วยให้ฮับสามารถต่อกับฮับอื่นหรือเครื่องคอมพิวเตอร์อื่นได้ active hub โดยปกติจะมีตัวเชื่อมต่อ 4 ถึง 16 ตัว passive hub เป็นอุปกรณ์ที่ไม่ต้องมีไฟเลี้ยง และจะไม่สร้างสัญญาณใหม่หรือขยายสัญญาณ passive จะจำกัดตัวเชื่อมต่อไม่เกิน 4 ตัว ตัวเชื่อมต่อที่ใช้จะเป็นสายโคแอกเชียลแบบ RG-62 93 โอห์ม

ระบบเครือข่ายแบบ ARCnet ต้องมีความยาวไม่เกิน 6000 เมตร ระยะสูงสุดระหว่าง active hub อื่นจะต้องไม่เกิน 600 เมตร ระยะสูงสุดระหว่าง active hub และ passive hub หรือ passive hub กับสถานีผู้ใช้ต้องไม่เกิน 30 เมตร passive hub ไม่สามารถต่อโดยตรงได้ และไม่สามารถต่อเป็นลูปได้ สายส่งข้อมูลที่ใช้จะเป็นสายโคแอกเชียล RG-62 แบบ 93 โอห์ม โดยใช้ตัวต่อแบบ BNC และเซกเมนต์ที่ต่ออยู่กับจุดแบบ active มีความยาวได้สูงสุด 2000 ฟุต แต่เซกเมนต์ที่ต่อกับจุดแบบ passive ต้องมีความยาวไม่เกิน 100 ฟุต จุดแบบ active สามารถต่อกับ active hub หรืออะแดปเตอร์ของ ARCnet ได้เท่านั้น ตัวเชื่อมต่อบนจุดแบบ active ที่ไม่ได้ใช้งานไม่จำเป็นต้องมีตัวเทอร์มิเนเตอร์ และจุดแบบ passive จะต้องมีระยะทางระหว่างอะแดปเตอร์ไม่น้อยกว่า 1.8 เมตร



รูป 4.8 การใช้งานระบบเครือข่าย ARCnet

เวอร์ชันแรกของ ARCnet จะเป็นไปตามที่กล่าวมาข้างต้น ซึ่งมีการสนับสนุนการใช้งานโดยผู้ผลิตหลายราย ส่วนเวอร์ชันที่ได้ใช้งานใหม่จะใช้งานกับสายส่งข้อมูลแบบเส้นใยนำแสงและแบบคู่บิดเกลียว ทุกเวอร์ชันของ ARCnet จะทำงานที่ความเร็ว 2.5 เมกกะบิทต่อวินาที โดยใช้โปรโตคอลแบบเดียวกัน

เครื่องไมโครคอมพิวเตอร์สามารถต่อกับเซกเมนต์ daisy-chain แบบโคแอกเชียลหรือสายคู่บิดเกลียวได้สูงสุดถึง 10 ตัวด้วยกัน อะแดปเตอร์ทุกตัวที่ต่อเชื่อมกับเซกเมนต์ daisy-chain จะต้องใช้อะแดปเตอร์แบบ high-impedance adaptor และ อะแดปเตอร์แบบ high-impedance ชนิดโคแอกเชียลจะต่อโดยใช้ตัวเชื่อมแบบ BNC ชนิด T โดยสายจะถูก daisy-chain ผ่านอะแดปเตอร์ที่ตัวเชื่อมต่อเป็นแบบ RJ-11 ที่ใช้กับโทรศัพท์แบบโมดุลาร์ และยังสามารถใช้กับสายโทรศัพท์แบบธรรมดาได้ด้วยถ้ามีระยะไม่ไกลนัก แต่สายแบบคู่บิดเกลียวต้องใช้ในระยะทางไกลเท่านั้น และปลายของ chain แบบคู่บิดเกลียวจะต้องปิดด้วยตัวเทอร์มิเนเตอร์

ตัวอะแดปเตอร์ของ ARCnet ใช้ความเร็วในการส่งข้อมูล 2.5 เมกกะบิทต่อวินาที โดยใช้โปรโตคอลแบบเดียวกัน ทำให้ระบบเครือข่ายท้องถิ่นแบบ ARCnet สามารถใช้อะแดปเตอร์ และสายส่งข้อมูลได้หลายแบบ การแปลงระหว่างชนิดของสายส่งข้อมูลจะทำโดยใช้อับพิเศษ เช่น SMC

two-port, twisted-pair, high-impedance coaxial hub

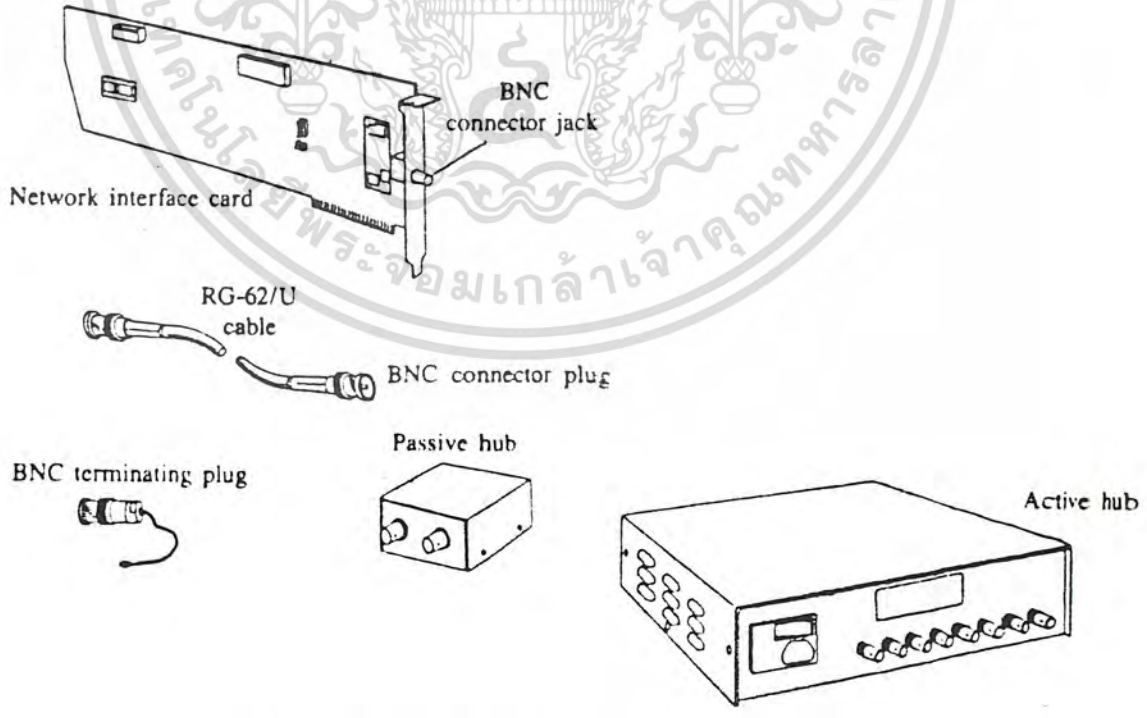
ตัวอะแดปเตอร์แต่ละตัวมีบัฟเฟอร์ขนาด 2 กิโลไบท์และอาจจะมี boot ROM ขนาด 16 K ได้ อินเทอร์รัพท์และพอร์ท I/O ก็สามารถเลือกได้ ซึ่งทำให้อะแดปเตอร์ของผู้ผลิตแต่ละรายสามารถใช้งานร่วมกันได้

มีข้อควรคำนึงถึงข้อหนึ่งก็คือ ตำแหน่งบัฟเฟอร์ขนาด 2 กิโลไบท์ ตำแหน่งของหน่วยความจำที่ซอฟท์แวร์ระบบเครือข่ายที่ใช้งานกับ ARCnet จะอยู่ที่ D000h ซึ่งอาจจะเป็นปัญหา ถ้าต้องการใช้ EMS หรือ บอร์ด EGA ซึ่งมีการใช้หน่วยความจำที่ตำแหน่ง D000h เหมือนกัน

อะแดปเตอร์ของผู้ผลิตหลายราย สามารถปรับได้เพียงตำแหน่งในหน่วยความจำแค่ 4 บิทบน ดังนั้นจะกำหนดตำแน่งของบัฟเฟอร์ได้ที่ B000, C000, D000 และต่อๆไป ซึ่งทำให้มีทางเลือก 2 ทางคือ ไม่ใช้ ARCnet, EGA หรือ EMS อย่างใดอย่างหนึ่ง หรือใช้หน่วยความจำหลักแค่ 512 กิโลไบท์ และใช้ตำแหน่งของบัฟเฟอร์ที่ A000h

การใช้ตัวอะแดปเตอร์หลายชนิดจากผู้ผลิตหลายรายร่วมกันได้ ในระบบเครือข่ายท้องถิ่นเดียวกัน เป็นจุดเด่นของ ARCnet โดยเฉพาะอย่างยิ่งในระบบเครือข่ายที่มีการเติบโตและมีการเปลี่ยนแปลงอะแดปเตอร์ และสายส่งข้อมูลที่เหมาะสมจะสามารถนำมาใช้ได้

อาร์ดแวร์ที่ใช้ในระบบเครือข่าย ARCnet มีดังนี้



รูป 4.9 อาร์ดแวร์ที่ใช้ในระบบเครือข่าย ARCnet

Passive Hub อุปกรณ์ที่ใช้แบ่งสัญญาณของระบบเครือข่าย โดยปกติ 1 ตัวจะมี 4 ช่องทาง ช่องทางที่ไม่ได้ติดต่อกับสถานีใช้งานต้องปิดด้วยเทอร์มิเนเตอร์ชนิด 91โอห์ม

Active Hub อุปกรณ์ที่ใช้เพื่อขยายสัญญาณ (condition, boost และ relay) ของระบบเครือข่าย โดยปกติ 1 ตัว จะมีอยู่ 8 ช่องทาง

สายส่งข้อมูล ระบบเครือข่ายแบบ ARCnet ใช้สายส่งข้อมูลโคแอกเชียล RG-62 /U 93 โอห์ม โดยปลายทั้งสองด้านเป็นตัวเชื่อมต่อ BNC ตัวเมีย

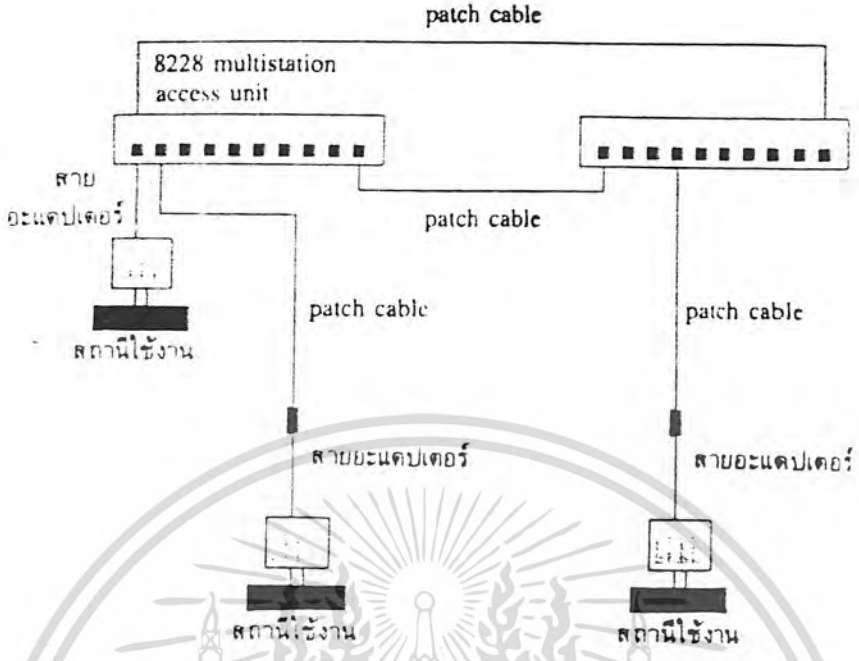
ข้อจำกัด

- ความยาวสูงสุดของสายส่งสัญญาณระหว่างสถานีที่ไกลที่สุดของระบบเครือข่าย : 6000 เมตร
- ระยะสูงสุดระหว่าง active hub และสถานีใช้งาน หรือระหว่าง active hub กับ active hub : 600 เมตร
- ระยะทางสูงสุดระหว่าง active hub กับ passive hub : 30 เมตร
- ระยะทางสูงสุดระหว่าง passive hub กับสถานีใช้งาน : 30 เมตร
- passive hub ไม่สามารถต่อกับ passive hub อื่นได้
- ช่องทางของ passive hub ที่ไม่ได้ใช้ต้องปิดด้วยเทอร์มิเนเตอร์ชนิด 91โอห์ม

4.3 การใช้งานระบบเครือข่ายแบบ Token-Ring

ระบบเครือข่ายแบบ Token-Ring เป็นระบบเครือข่ายที่มีความสัมพันธ์กับมาตรฐาน IEEE 802.5 Token-passing Ring ที่มี Topology แบบวงแหวน และมีโปรโตคอลแบบ token-passing หลังจากได้มีการทดลองใช้งานวิธีการแบบ Token-Ring อยู่หลายครั้ง บริษัท Prime Computer และ Apollo ก็ได้ขายระบบเครือข่ายในต้นทศวรรษที่ 80 หลังจากนั้นไม่นานทาง IBM ก็ได้ประกาศใช้เวอร์ชันของระบบเครือข่ายสำหรับเครื่องไมโครคอมพิวเตอร์ที่เป็นแบบ Token-Ring เช่นกัน และระบบเครือข่ายของ IBM ยังสามารถขยายเพื่อต่อกับเครื่องเมนเฟรม IBM 3725 communication controller เพื่อทำให้เครื่องไมโครคอมพิวเตอร์ทำงานเหมือนเป็นเทอร์มินัลหรือประตูสื่อสารได้

จากมาตรฐานแบบ nonproprietary ที่มีอยู่ร่วมกับการสนับสนุนของ IBM ทำให้ IEEE 802.5 ได้รับความนิยมอย่างมาก ผู้ผลิตหลายรายได้เสนอผลิตภัณฑ์ของตนที่มีความเข้ากันได้กับ 802.5 รวมทั้ง 3Com, Proteon, General Instrument และ Ungermann-Bass และยังมีเครื่องมือในการตรวจสอบการทำงาน (Diagnostic tools) เช่น Sniffer protocol analyzer ของ Network General



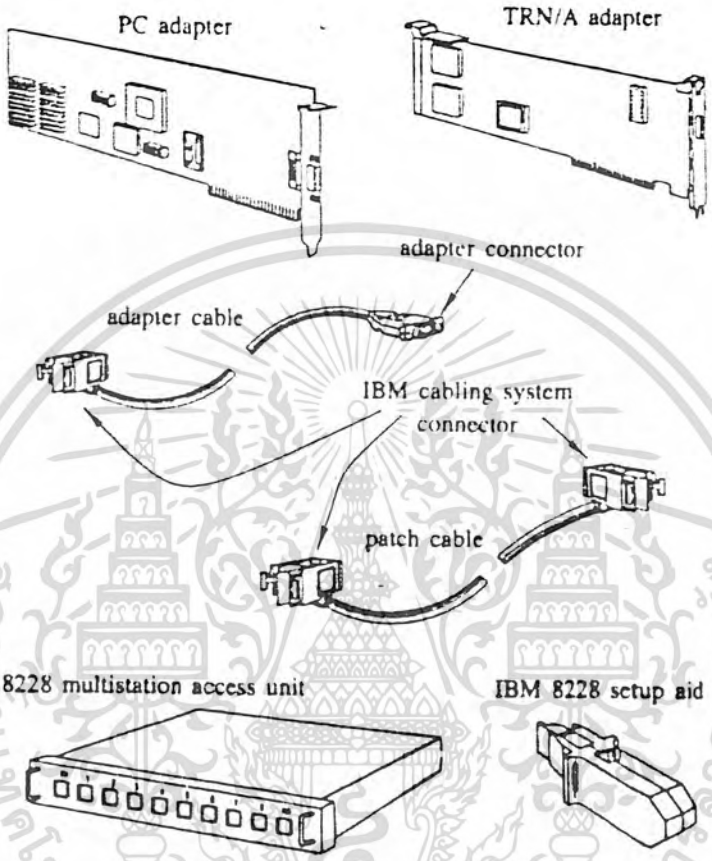
รูป 4.10 การใช้งานระบบเครือข่าย IBM Token-Ring

ระบบเครือข่ายแบบ token ring ประกอบด้วยชุดของสถานีผู้ใช้ที่ต่อกันโดยมีสายส่งข้อมูลเป็นตัวส่งข้อมูล และเป็นการส่งข้อมูลแบบเรียงลำดับ (sequential) จากสถานีหนึ่งไปยังสถานีถัดไป สถานีที่รับจะต้องทวนซ้ำข้อมูล และสร้างสัญญาณใหม่ไปยังสถานีถัดไปในวงแหวน สถานีรับที่มีตำแหน่งตรงกับฟิลด์ของตำแหน่งปลายทางในกลุ่มข้อมูลสื่อสารจะรับข้อมูลนั้นไว้ และประมวลผลตามฟังก์ชัน MAC ของ ชั้นที่ 2 ใน 802.5 สถานีที่มีตำแหน่งตรงกับตำแหน่งปลายทางจะยังคงส่งข้อมูลนี้ไปยังสถานีถัดไปในวงแหวน สถานีที่เป็นตัวส่งข้อมูลเริ่มแรกจะเป็นตัวลบมันออกจากวงแหวน

สถานีแต่ละสถานีจะสามารถส่งข้อมูลได้ก็ต่อเมื่อมันได้รับ token เมื่อสถานีได้รับ token สถานีนั้นจะปรับตัวเพื่อสร้างเฟรมที่ประกอบด้วย start-of-frame sequence ฟิลด์ควบคุมและฟิลด์สถานะ ฟิลด์ตำแหน่ง ฟิลด์ข้อมูล frame check sequence และ end-of-frame sequence หลังจากที่เฟรมใหม่ถูกสร้างขึ้นมาแล้ว สถานีจะส่ง initiate token ใหม่ ซึ่งจะทำให้สถานีอื่นในวงแหวนใช้งานมันได้ token จะควบคุมเวลา และจำนวนเวลาสูงสุดที่สถานีจะสามารถใช้ตัวกลางได้ ทั้งการส่งเฟรมใหม่หรือไม่ส่ง ก่อนที่จะมีการส่ง token ออกไป

มาตรฐาน 802.5 รวมถึงระบบของลำดับความสำคัญ (priority) ที่ถูกกำหนดโดยชนิดของข้อความ เช่น synchronous, asynchronous และ immediate (network recovery) อย่างไรก็ตาม token ring ของ IBM ที่ใช้งานกับเครื่องไมโครคอมพิวเตอร์ไม่สนับสนุนลำ

ดับความสำคัญ



รูป 4.11 อาร์ดแวร์ที่ใช้ในระบบเครือข่าย IBM Token-Ring

ระบบเครือข่ายแบบ token-ring ของ IBM ใช้สายส่งข้อมูลได้สองแบบคือ shielded twisted pair (type 1, 2 and 9) และ unshielded telephone-style wire (type 3) สายส่งข้อมูลแบบ type 1 นั้นมีความเชื่อถือได้สูงกว่า type 3 และสามารถต่อได้ยาวกว่าด้วย

จำนวนอุปกรณ์ที่ต่อได้สูงสุดในวงแหวนคือ 260 ตัว รวม IBM Multistation Access Unit (MAU) 33 ตัวด้วย สายส่งข้อมูลจาก MAU ถึงสถานีผู้ใช้ต้องน้อยกว่า 100 เมตร เซกเมนต์ที่ยาวที่สุดจะเป็นไปได้เมื่อ MAU อยู่ใน wiring closet เดียวกัน ตัวทวนซ้ำสัญญาณหรือ IBM Copper Repeater จะช่วยให้วงแหวนทำงานได้ไกลถึง 775 เมตร อาจใช้ตัวทวนซ้ำสัญญาณแบบ

เส้นใยนำแสงในสถานที่ที่มีการรบกวนของสัญญาณแม่เหล็กไฟฟ้าสูง หรืออาจใช้ตัวทวนซ้ำสัญญาณแบบนี้เพื่อช่วยเพิ่มระยะการใช้งานได้ โดยที่ตัวทวนซ้ำสัญญาณ 1 ตัวจะขยายระยะทางได้ 2 กิโลเมตร

ข้อจำกัด
- จำนวนสถานีใช้งานสูงสุด : 260
- จำนวนของ 8228 units : 33
- ความยาวสูงสุดของ patch cable ระหว่าง 8228 units กับสถานีใช้งาน (ไม่รวมสายอะแดปเตอร์) : 100 เมตร
- ความยาวของ patch cable ทั้งหมดที่ต่ออยู่กับ 8228 units : 775 เมตร

4.4 การใช้งานระบบเครือข่ายแบบ StarLAN

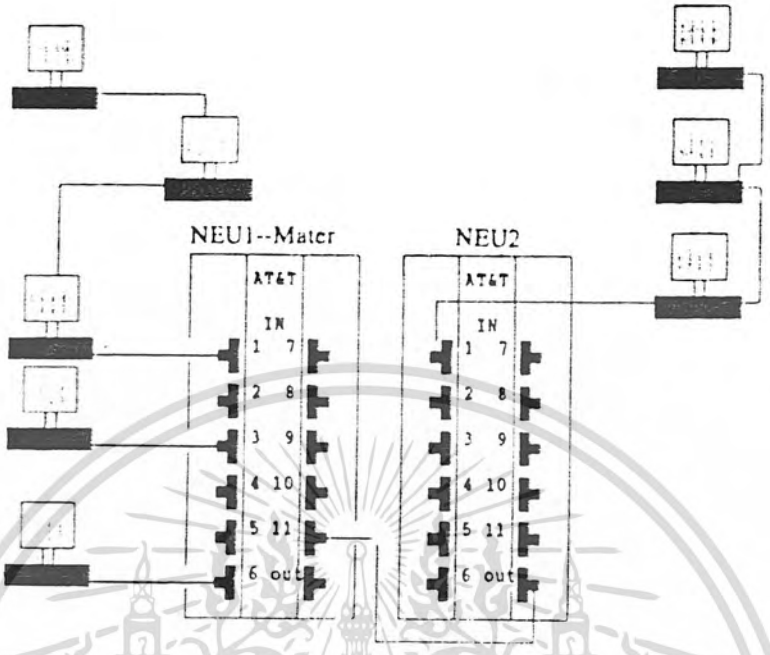
StarLAN เป็นการใช้งานระบบเครือข่ายที่ใช้สายคู่บิดเกลียวของ AT&T โดยใช้โปรโตคอล CSMA/CD ถึงแม้ว่าความเร็วในระบบแบบ StarLAN จะแค่ 1 เมกกะบิทต่อวินาที เมื่อเทียบกับระบบ Ethernet ที่มีความเร็วถึง 10 เมกกะบิทต่อวินาทีก็ตาม แต่ก็มีข้อดีตรงที่การเดินสายจะง่ายกว่า เมื่อเทียบกับสายแบบโคแอกเชียล

สายที่ใช้จะเป็นแบบ four-pair modular ซึ่งจะคล้ายกันกับสายโทรศัพท์แบบโมดูลาร์ แต่ไม่สามารถใช้แทนกันได้ เนื่องจากโครงสร้าง (configuration) ของ pin บน jack แตกต่างกัน ถ้าจำเป็นอาจใช้ AT&T DEBA-DE หรือสายโมดูลาร์และตัวเชื่อมต่อ 451-A ของ AT&T ก็ได้

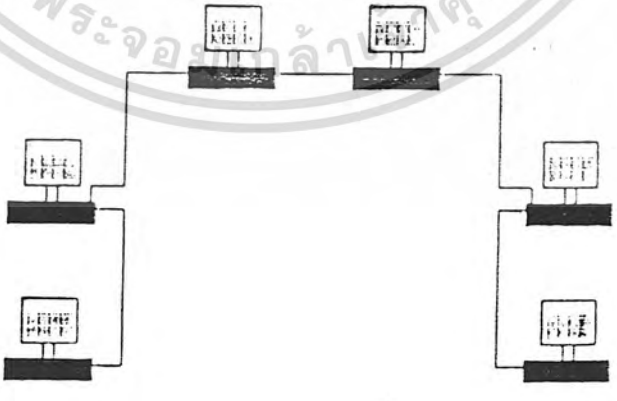
ระบบเครือข่ายท้องถิ่นแบบ StarLAN ใช้ Topology แบบดาว, daisy chain หรือทั้งสองแบบร่วมกันได้ ในการต่อแบบ daisy chain มีข้อดีคืออุปกรณ์แต่ละตัวจะเชื่อมต่อกับตัวต่อไปแบบอนุกรมด้วยอุปกรณ์เชื่อมต่อแบบตัว T ซึ่งเป็นตัวให้สัญญาณออกของสถานีหนึ่ง จะเป็นสัญญาณเข้าของอีกสถานีหนึ่งดังรูป 4.12

การต่อแบบ daisy chain เหมาะสำหรับระบบเครือข่ายที่มีไม่เกิน 10 สถานี และระยะห่างระหว่างปลายทั้งสองด้านต้องน้อยกว่า 122 เมตร

ถ้าเป็นระบบเครือข่ายที่มีความยาวเกินกว่า 122 เมตร จะต้องต่อโดยใช้ Topology แบบดาวแทน ซึ่งจะเป็นการต่อแต่ละอุปกรณ์ ออกจากจุดศูนย์กลางที่เรียกว่า Network Extension Unit (NEU) ใน NEU แต่ละตัวจะต่อสถานีออกมาได้สูงสุด 11 สถานี และสายส่งข้อมูลของแต่ละสถานีจาก NEU จะยาวได้สูงสุด 245 เมตร ถ้าระบบมีสถานีมากกว่า 11 สถานี ก็ต้องใช้ NEU หลายตัว ซึ่งจะมี NEU ตัวหนึ่งเป็นตัวหลัก NEU ตัวอื่นจะต่ออยู่กับ NEU ตัวหลัก ด้วยสายส่งข้อมูลที่มีความยาวไม่เกิน 3 เมตร



รูป 4.12 การใช้งานระบบเครือข่าย StarLAN



รูป 4.13 การใช้งานระบบเครือข่าย StarLAN ขนาดเล็กโดยใช้ daisy chain

StarLAN มีความยืดหยุ่นที่ยอมให้ทั้ง daisyy chain และ Topology แบบดาวต่ออยู่ในระบบเครือข่ายเดียวกันได้ รูป 4.13 แสดงการต่อแบบ daisy chain ที่มี 2 ถึง 10 สถานี สามารถต่อออกจากพอร์ทของ NEU ตัวเดียวได้ระยะทางสูงสุดที่สามารถจะต่อ daisy chain ออกจาก NEU ได้ดังตาราง 4.1

จำนวนสถานี	ระยะทาง (เมตร)
2 ถึง 5	245
6	225
7	190
8	170
9	140
10	125

ตาราง 4.1

เราสามารถขยายระยะทางได้ โดยใช้โมดูลการเชื่อมต่อเครือข่ายต่อระหว่างระบบเครือข่าย StarLAN กับ Information Systems Network (ISN) data switch ของ AT&T ISN จะสนับสนุนการต่อหลายแบบ ซึ่งเพิ่มระยะทางได้เป็นหลายพันเมตร แต่การแปลงโปรโตคอลที่จำเป็นยังไม่มีซอฟท์แวร์สนับสนุนมากนัก เช่น NetWare ของ Novell ก็ไม่สนับสนุนการแปลงโปรโตคอลเพื่อใช้กับ ISN

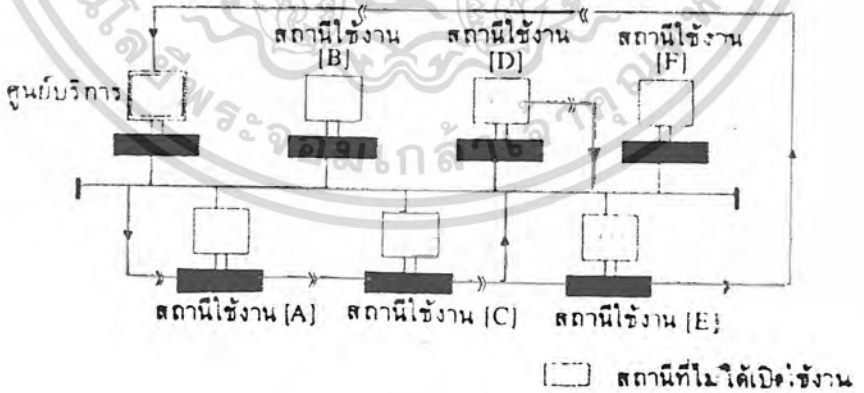
ข้อจำกัด
- การต่อแบบ daisy chain เหมาะสำหรับระบบเครือข่ายที่มีไม่เกิน 10 สถานี และระยะทางระหว่างปลายทั้งสองด้านต้องน้อยกว่า 122 เมตร
- การต่อแบบดาวเป็นการต่ออุปกรณ์แต่ละตัวออกมาจาก NEU ซึ่ง NEU แต่ละตัวต่อสถานีใช้งานออกไปได้ 11 สถานี
- สายส่งข้อมูลจาก NEU ถึงสถานีผู้ใช้ยาวได้ไม่เกิน 245 เมตร
- NEU ที่ต่อกับ NEU หลักต้องมีความยาวไม่เกิน 1 เมตร

4.5 การใช้งานระบบเครือข่ายแบบ Token Buses

Topology ของระบบเครือข่ายที่จะใช้งานในโรงงาน ต้องมีความยืดหยุ่นสูงในด้านของความยาวสายส่งข้อมูล และความสามารถในการใช้ Topology แบบต้นไม้ หรือแบบดาว รวมทั้งการใช้ตัวทวนซ้ำสัญญาณ ระบบเครือข่ายที่ใช้ต้องมีความเชื่อถือได้สูง มีระยะเวลาการตอบสนองต่ำ บัสแบบ token-passing จะช่วยในด้านเหล่านี้ General Motors and Society of Manufacturing Engineers ได้พัฒนารูปแบบของ manufacturing automation protocol (MAP) ขึ้นมาเพื่อใช้งานระบบเครือข่ายแบบ token-passing bus ซึ่งแนวคิดพื้นฐานเหล่านี้ถูกนำไปใช้กับระบบเครือข่าย ARCnet

มาตรฐาน IEEE 802.4 ได้กำหนด token bus ที่สามารถนำไปใช้งานโดยใช้เบสแบนด์ บรอดแบนด์ หรือ hybrid application ได้ บัสจะมีการทำงานคล้ายกับ token ring ตรงที่ algorithm ในการส่ง token จะเป็นแบบเรียงลำดับ จากสถานีหนึ่งไปยังอีกสถานีหนึ่ง

token bus จะมีการทำงานเชิงตรรกเป็นระบบเครือข่ายแบบวงแหวน (4.4.2.2 Token-passing Buses) ดังรูป 4.14



รูป 4.14 การทำงานของระบบเครือข่ายแบบ Token-Passing Bus

หลังจากที่สถานีได้ส่งเฟรมของข้อมูลใดๆ เรียบร้อยแล้ว สถานีจะส่งเฟรม token MAC ไปให้ตัวต่อไปจากมัน แล้วคอยฟังการตอบรับที่ตัวต่อจากมันว่าได้รับเฟรมที่ส่งไปให้หรือไม่ แล้วตัวนั้นก็จะทำงาน (active) ถ้าผู้ส่ง token ไม่ได้รับเฟรมที่ถูกต้องเมื่อคอยฟังครบ 4 ครั้งแล้ว มันจะส่งข้อมูลใหม่อีกครั้ง และถ้าตัวต่อไปจากมันยังไม่ส่งเฟรมที่ถูกต้องมาอีก โดยจะถือว่าตัวต่อไปจากมันเสีย ผู้ส่งจะส่งเฟรม who follows ที่จะไปถามว่าใครเป็นสถานีที่อยู่ต่อจากสถานีที่เสีย เพื่อช่วยให้ผู้ส่งสามารถกำหนดสถานีถัดไปที่จะส่งข้อมูลไปให้ และส่ง token ไปให้

Response windows จะช่วยให้สถานีใหม่ต่อเข้าทางตรรกกับวงแหวนได้ เฟรม solicit successor จะระบุช่วงของตำแหน่งของสถานีซึ่งสถานีที่เข้ามาจะตอบรับถ้ามันอยู่ในตำแหน่งช่วงนี้ สถานี solicite จะต้องกำหนดสถานีต่อไปถ้ามันตอบรับ

มิลิอการส่งข้อมูล 3 แบบ และวิธีการส่งที่มีอยู่ในมาตรฐาน 802.4 สำหรับเวอร์ชัน 1 เมกกะบิทต่อวินาที จะใช้บัสแบบทางเดียว (omnidirectional) และสายส่งข้อมูลแบบโคแอกเชียล 75 โอห์ม เช่น RG-6, RG-11 และ semirigid (CATV) สายที่ต่อออกไปจะเป็น stub แบบ 25 ถึง 50 โอห์ม ที่ยาวไม่เกิน 3.5 เมตร และสายส่งข้อมูลหลักจะต้องไม่มีการแยกแขนง (branch) ถ้าต้องการขยายระบบอาจจะใช้ active regenerative repeater ก็ได้ สำหรับการแยกแขนงออกไป ระบบสัญญาณที่เดินในสายจะถูกเข้ารหัสโดยวิธีแมนเชสเตอร์

สำหรับเวอร์ชันที่เป็นแบบเบสแบนด์ 5 ถึง 10 เมกกะบิทต่อวินาที จะใช้บัสแบบ omnidirection และ สายส่งข้อมูลแบบ 75 โอห์ม หรืออาจใช้สายแบบ semirigid และต่อมายังสถานีด้วย RG-6 ก็ได้ อุปกรณ์ทวนซ้ำสัญญาณแบบ active (active repeater) อาจจะถูกใช้เพื่อขยายระยะทาง และการแยกแขนงระบบบัส สัญญาณในสายจะเป็นแบบ frequency shift keying (FSK) ที่มีการเข้ารหัสโดยตรงที่ frequency shift โดยเฉพาะ

ในเวอร์ชันที่เป็นแบบบรอดแบนด์จะใช้สายทรังก์แบบ semirigid ที่คล้ายกับสาย CATV ตัวขยาย (amplifier) มาตรฐานสำหรับ CATV จะใช้เพียงเป็น head-end regenerative repeaters ที่จะจัดการสัญญาณนาฬิกา สามารถใช้ความเร็วได้ 1, 5 และ 10 เมกกะบิทต่อวินาที กับช่องทางบรอดแบนด์ 1.5, 6 และ 12 เมกะเฮิร์ตซ์ตามลำดับ สัญญาณจะเป็นแบบ amplitude modulation ของสัญญาณความถี่วิทยุ 3 ระดับ คือ ศูนย์, nondata และหนึ่ง ระดับ nondata ใช้เพื่อให้มั่นใจในการ synchronization

บทที่ 5
โครงข่ายการสื่อสารข้อมูล

โครงข่ายการสื่อสารข้อมูล *(Data Communication Network)* นั้น เป็นโครงข่ายการติดต่อของระบบการสื่อสารจำนวนมากรวมกัน ตามปกติมักจะประกอบด้วยคอมพิวเตอร์ที่มีอุปกรณ์ปลาย *(Terminal)* ต่ออยู่โดยใช้สายสื่อสาร *(Communication line)* สายสื่อสารทำหน้าที่เป็นตัวกลางในการนำข่าวสารระหว่างคอมพิวเตอร์กับอุปกรณ์ปลาย อุปกรณ์ปลายนั้นมีมากมายหลายชนิดตั้งแต่ เครื่องพิมพ์ดีดข้อมูล *(Datatypewriter)* แป้นพิมพ์ *(Keyboard)* เครื่องพิมพ์ความเร็วสูง *(High-Speed Printer)* จอภาพแสดงผล *(Visual Display Terminal)* หรือแม้กระทั่งเครื่องคอมพิวเตอร์เองก็เป็นอุปกรณ์ปลายได้

ในการใช้ตัวกลางในการต่อเป็นโครงข่ายสื่อสารนั้น มีรูปแบบในการต่อโครงข่ายหลายๆรูปแบบ เช่น การต่อโครงข่ายแบบดาว *(Star Network)* โครงข่ายแบบวงแหวน *(Ring Network)* โครงข่ายแบบตาข่าย *(Mesh Network)* โครงข่ายแบบลำดับขั้น *(Hierachical Network)* เป็นต้น สำหรับการติดต่อระหว่างอุปกรณ์ปลายในแต่ละด้านนั้น อาจจะมีอุปกรณ์ปลายตัวหนึ่งเป็นตัวควบคุมตัวอื่นๆ หรือแต่ละตัวต่างเป็นอิสระต่อกันก็ได้ ในการติดต่อนั้นจำเป็นต้องมีข้อกำหนดในการติดต่อ เพื่อแสดงการเริ่มต้นการส่ง การสิ้นสุด ความถูกต้องของข้อมูล ตลอดจนรายละเอียดอื่นๆที่จำเป็นต่อการติดต่อสื่อสารกันซึ่งได้มีการกำหนดเป็นมาตรฐานในการติดต่อ จะได้กล่าวรายละเอียดต่อไป

5.1 การส่งข้อมูล

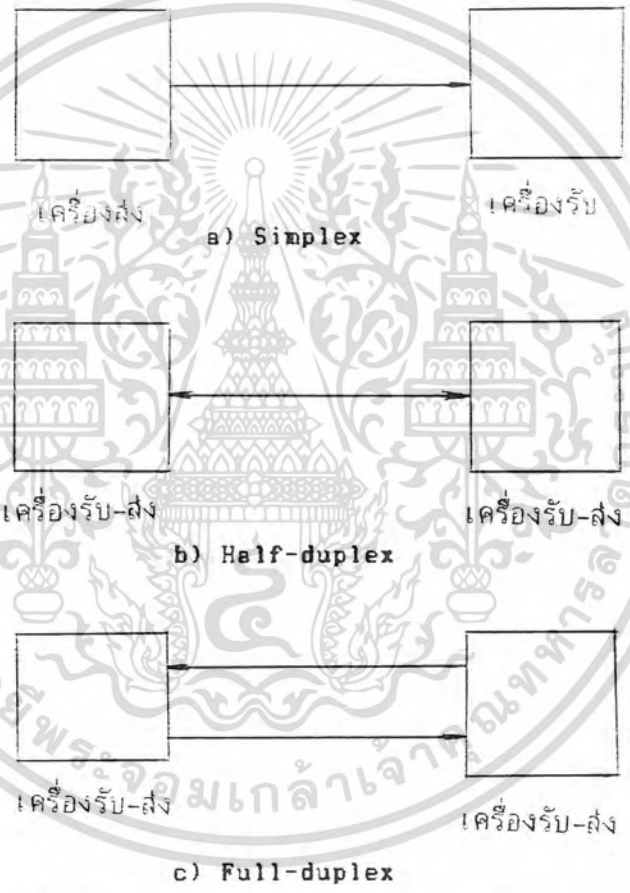
ในการส่งข้อมูลไปบนตัวกลางการสื่อสารนั้น ข้อมูลที่อยู่ในรูปของเลขฐานสองจะถูกเปลี่ยนให้เป็นสัญญาณทางไฟฟ้าก่อนแล้วจึงทำการส่งไปยังจุดรับ สำหรับวิธีการในการส่งข้อมูลนั้นสามารถจำแนกออกตามคุณสมบัติต่างๆได้หลายวิธีด้วยกัน ในที่นี้จะกล่าวถึงลักษณะทางไฟฟ้าที่ใช้แทนข้อมูลสำหรับการส่ง วิธีการส่งข้อมูลจำแนกตามลักษณะทิศทางการส่ง จำแนกตามลักษณะการจัดข้อมูลในการส่ง จำแนกตามความสัมพันธ์ของข้อมูล เป็นต้น

5.1.1 การจำแนกวิธีการส่งข่าวสารตามทิศทางการส่งภายในสาย

ตัวกลางที่ใช้ในการสื่อสารข้อมูลอาจมีได้หลายชนิดแต่ตัวกลางที่นิยมใช้กันมากก็คือ สายสื่อสาร *(Communication Line)* ซึ่งตามปกติแล้วก็คือสายโทรศัพท์ *(Telephone Line)* โดยการส่งข้อมูลออกไปตามสาย เราสามารถพิจารณาตามทิศทางการส่งข้อมูลภายในสาย แล้วแบ่งการส่งข้อ

มูลออกเป็น 3 ชนิดคือ

1. การส่งแบบทิศทางเดียว *(One-way Transmission or Simplex Tranmission)*
2. การส่งแบบทิศทางใดทิศทางหนึ่ง *(Either-way transmission or Half-duplex Transmission)*
3. การส่งแบบสองทิศทาง *(Both-way Transmission or Full-duplex Transmission)*



รูปที่ 5.1 การส่งข้อมูลแบบต่างๆจำแนกตามทิศทางการส่งภายในสายส่ง

5.1.2 การจำแนกวิธีการส่งตามความสัมพันธ์ของข้อมูล

สำหรับวิธีการส่งข้อมูลที่จำแนกตามความสัมพันธ์ระหว่างข้อมูลที่ส่งนั้น เราอาจแบ่งออกได้เป็น 2 ชนิด ดังนี้คือ

1) การส่งข้อมูลแบบสัมพันธ์ *(Synchronous Transmission)*

2) การส่งข้อมูลแบบไม่สัมพันธ์ *(Asynchronous Transmission)*

การส่งข้อมูลแบบสัมพันธ์

สำหรับความสัมพันธ์ระหว่างข้อมูลที่เราส่งในระบบการสื่อสารข้อมูลนั้น เราจะต้องพิจารณาถึงความสัมพันธ์สองชนิดคือความสัมพันธ์ของบิท (Bit Synchronization) ที่ประกอบกันเป็นอักขระหนึ่งตัว และความสัมพันธ์ของอักขระ (Character Synchronization) ที่ประกอบกันเป็นบล็อคในการส่งที่เราจะพิจารณาในตอนนี้

ความสัมพันธ์ของบิท

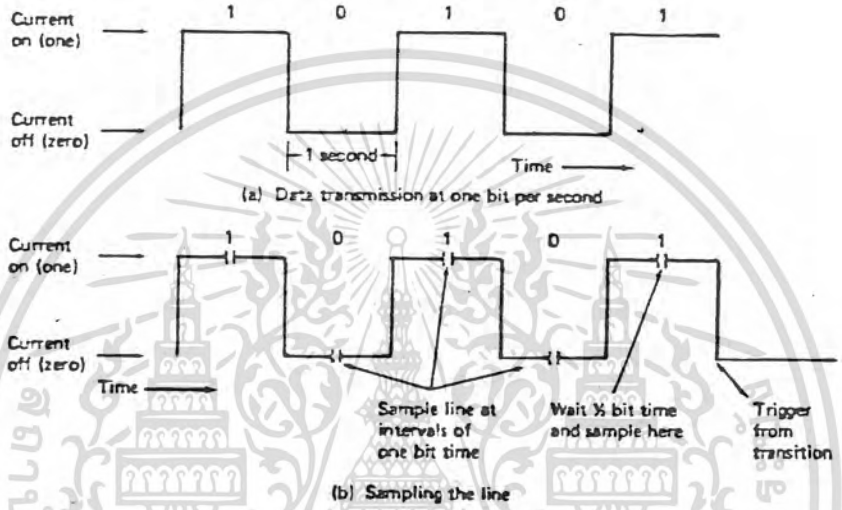
สำหรับความสัมพันธ์ของบิท หมายถึงว่าทางด้านรับจะต้องรับบิทต่างๆที่ทางด้านส่งทำการส่งมาได้อย่างถูกต้อง ซึ่งทางด้านรับจะต้องทราบว่าจะรับบิทจากสายส่งเมื่อใด หลังจากรับมาแล้วจะรับตัวที่ 2, 3 และตัวต่อๆไปเมื่อไร ซึ่งสามารถทำได้โดยการเพิ่มสัญญาณนาฬิกาเข้าไปที่จุดปลายของระบบทั้งสองด้าน ดังรูป 5.2

รูป 5.2 การเพิ่มคล็อคเพื่อความสัมพันธ์ของบิท

สัญญาณคล็อคทางด้านส่งจะเป็นตัวกำหนดเวลาว่า เมื่อใดจะส่งบิทข้อมูลออกไปตามแช็นแนล และสัญญาณคล็อคทางด้านรับจะเป็นตัวกำหนดว่า จะรับข้อมูลมาจากแช็นแนลด้วยอัตราความถี่เท่าใด เช่น ถ้าเราต้องการส่งข้อมูลด้วยความเร็ว 100 ต่อวินาที เราก็ใช้สัญญาณคล็อคที่มีความถี่ 100 บิทต่อวินาทีสำหรับด้านส่ง ทางด้านส่งก็จะส่งบิทออกไปตามสายส่ง 100 บิท ใน 1 วินาที ทางด้านรับที่รับสัญญาณที่ส่งมาจะต้องรับบิททุกๆ 1/100 วินาที ดังนั้นเราต้องสร้างสัญญาณคล็อคทางด้านรับให้มีค่า 100 บิทต่อวินาที เพื่อที่ทางด้านรับจะได้ทำการรับข้อมูลจากสายจำนวน 100 ครั้งใน 1 วินาที

สำหรับวิธีการในการ "ส่งบิทออกมาตามสาย" และ "การสุ่มตัวอย่างบิทในทางด้านรับ" นั้นจะแตกต่างกันไปสำหรับระบบแต่ละระบบ เช่นในวงจรไฟฟ้าเราอาจแทนค่า 1 ด้วยการทำให้กระแส

ไหลในวงจร (ดวงไฟ "ติด") และแทนค่า 0 ด้วยการตัดกระแสออกจากวงจร (ดวงไฟ "ดับ") ถ้าหากเราต้องการส่งแถวของบิทที่ประกอบด้วย 0 และ 1 ในอัตราความเร็ว 1 บิทต่อวินาที เรากระทำดังแสดงในรูป 5.3 ซึ่งเป็นตัวอย่างของค่าข้อมูลในช่วงเวลาใดๆช่วงหนึ่ง ขนาดความกว้างของแต่ละบิทเท่ากันเพราะสัญญาณคล็อคเป็นตัวควบคุมความกว้างของแต่ละบิท



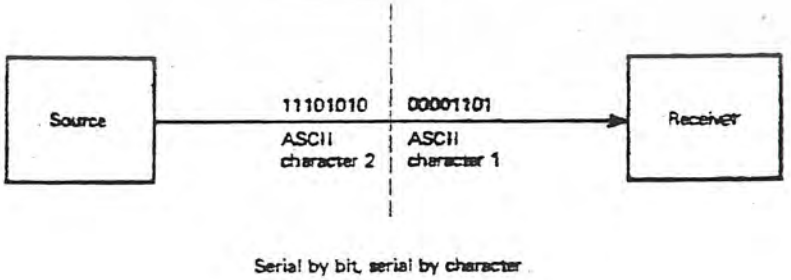
รูป 5.3 ตัวอย่างเกี่ยวกับการรับส่งแบบง่ายๆ

ทางด้านรับนั้นหากเป็นมนุษย์ก็สามารถมองเห็นการติดดับของดวงไฟ จึงสามารถแปลชุดของบิทที่เข้ามาได้ แต่หากเราใช้อุปกรณ์ปลายทางอิเลคทรอนิคส์แทนดวงไฟและโอเปอเรเตอร์ เราจะต้องแน่ใจว่าอุปกรณ์อิเลคทรอนิคส์นั้นสามารถรับข่าวสารมาได้อย่างถูกต้อง ตามปกติแล้วอุปกรณ์ปลายทางอิเลคทรอนิคส์จะสุ่มตัวอย่างโดยการตรวจดูสถานะของสาย (0 หรือ 1) ณ เวลาใดๆในช่วงสั้นๆ ว่า ค่าของสายเป็น 0 หรือ 1 ซึ่งเวลาที่สุ่มตัวอย่างนี้ควรจะเป็นช่วงกลางของบิทแต่ละบิท หากสุ่มตัวอย่างในช่วงเวลาที่กำหนดไว้สำหรับเป็นช่วงการเปลี่ยนแปลงระหว่าง 0 และ 1 นั้นอาจจะทำให้ได้ค่าที่ไม่ถูกต้อง ปกติแล้วการสุ่มตัวอย่างจะอยู่ที่จุดกลางของความกว้างของบิทพอดี ซึ่งสามารถกระทำได้โดยการใช้จุดของการเปลี่ยนแปลงจาก 1 ไป 0 หรือ 0 ไป 1 เป็นจุดอ้างอิงในตอนเริ่มแรก โดยการรอเวลาผ่านไปเท่ากับ 1/2 ของเวลาของความกว้างหนึ่งบิท แล้วจึงทำการสุ่มตัวอย่าง ณ จุดนั้นๆ หลังจากการสุ่มค่าของบิทแรกแล้วก็รอเวลาผ่านไปเท่ากับความกว้างของบิทหนึ่งบิท จึงทำการสุ่มค่า

ของบิทที่สอง สำหรับบิทที่สามและบิทต่อๆไปก็ทำในทำนองเดียวกัน ดังรูป 5.3b ซึ่งการสุ่มตัวอย่างค่าบิทแต่ละบิทเกิดที่กึ่งกลางของความกว้างของบิท ถ้าหากความเร็วของคล็อคทั้งทางด้านรับและส่งมีความเร็วเท่ากัน การสุ่มตัวอย่างค่าบิทก็จะกระทำได้อย่างถูกต้อง ในทางปฏิบัติแล้วคล็อคของทางด้านรับและส่งจะเป็นอิสระจากกัน ค่าความเร็วจิงอาจแตกต่างกันบ้างเล็กน้อย แต่ความเร็วที่แตกต่างกันแม้เพียงเล็กน้อย จะเพิ่มขึ้นเรื่อยๆเพราะสัญญาณนี้เกิดติดต่อกันเป็นวัฏจักร จิงทำให้เกิดความผิดพลาดในการรับสัญญาณของทางด้านรับได้ จำเป็นต้องมีการปรับความเร็วของคล็อค ทั้งทางด้านรับและส่งให้สัมพันธ์กันที่เรียกว่า ริซิงโครไนเซชั่น (Resynchronization)

ความสัมพันธ์ของตัวอักขระ

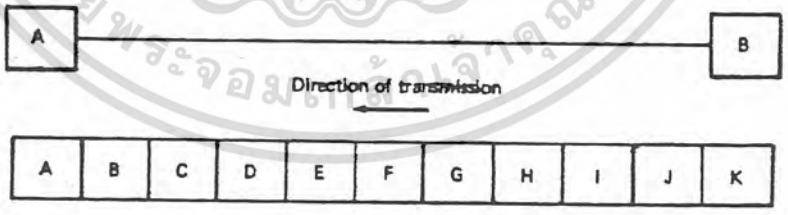
ในการรับข่าวสารตามสายนั้น แม้ว่าเราจะมีการจัดเกี่ยวกับความสัมพันธ์ของบิทแล้วก็ตาม ยังมีปัญหาที่ตามมาอีกก็คือ ในการส่งนั้นเรานำเอาบิทของอักขระหลายๆตัวมารวมกันเป็นบล็อค ฉนั้นแม้ว่าบิทต่างๆได้รับมาถูกต้องก็ตาม เรายังต้องทราบว่ากลุ่มของบิทที่แสดงถึงตัวอักขระต่างๆนั้นเริ่มต้นที่บิทใด รูป 5.4 แสดงถึงการส่งชุดของบิทของอักษรในรหัสแบบอัสกีย์ 2 ตัวที่ส่งไปอย่างอนุกรมตามสายสื่อสาร โดยมีข้อมูลขนาด 8 บิท จำนวน 2 ชุด สำหรับตัวอักษรตัวที่ 1 และตัวที่ 2 ตามลำดับเมื่อบิทของข้อมูลทั้งสองชุดนี้ถูกส่งไปติดต่อกันทางด้านรับเมื่อรับบิทดังกล่าวมาแล้ว จะทราบได้อย่างไรว่าชุดของบิทจำนวน 8 บิทของตัวอักษรแต่ละตัวเริ่มต้นที่บิทใด วิธีแก้ปัญหานี้กระทำได้ถ้าหากว่าเราทราบว่าบิทใดเป็นบิทเริ่มต้นของตัวอักษร และถ้าหากทราบว่า 1) ในตัวอักษรหนึ่งตัวนั้นมีกี่บิท และทราบว่า 2) ความเร็วของการส่งบิทต่างๆมาตามสาย โดยการนับจำนวนบิทที่ได้รับมาตามสายหลังจากทราบบิทแรกก็จะสามารถแยกตัวอักษรออกจากกันได้



รูป 5.4 การส่งข้อมูลเป็นตัวอักษร 2 ตัวแบบอนุกรม

เทคนิคการส่งแบบสัมพันธ์นั้นใช้สำหรับส่งข้อมูลทั้งชุดไปครั้งเดียว ในการส่งแบบนี้นั้นช่วงความกว้าง (เวลา) ระหว่างบิทแต่ละบิทจะมีค่าเท่ากัน และสำหรับระบบที่มีการส่งครั้งละ 1 ตัวอักษรนั้น ช่วงเวลาระหว่างการสิ้นสุดของบิทสุดท้ายของตัวอักษรตัวหนึ่งกับการเริ่มต้นของบิทแรกของตัวอักษรตัวถัดไปจะมีค่า 0 หรือมีเวลาเท่ากับเวลาทั้งหมดสำหรับการส่งหนึ่งตัวอักษรทำให้การส่งมีลักษณะคล้ายกับการส่งข่าวสารในรูปของเลขฐานลองที่มีจำนวนบิทติดต่อกันไป โดยไม่ได้แยกว่าความยาวใดเป็นของช่วงตัวอักษรใด ในระบบเช่นนี้บิทแต่ละบิทจะมีความยาวเท่ากัน

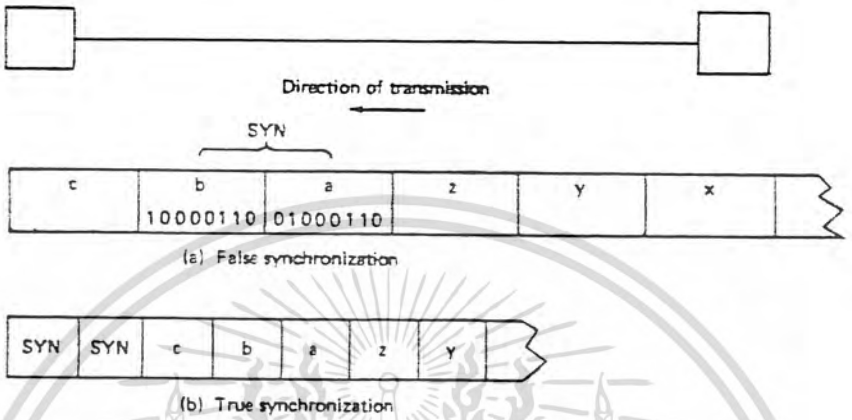
ในรูป 5.5 แสดงให้เห็นถึงการส่งอักษรต่างๆไปแบบการส่งสัมพันธ์ โดยตัวอักษรแต่ละตัวมีช่วงเวลาห่างกันเท่ากับศูนย์ ทางด้านรับนั้นเพียงหาว่า บิทแรกของตัวอักษรตัวแรกคือบิทใด และทราบขนาดหรือจำนวนบิทในหนึ่งตัวอักษร พร้อมทั้งความเร็วในการส่งก็จะสามารถแยกข่าวสารของแต่ละอักษรออกมาได้ ในกรณีที่ข้อมูลเป็นรหัสแบบอัลกีย์ ตัวอักษรแต่ละตัวจะประกอบด้วยบิท 8 บิท หากทราบว่าบิทใดคือบิทเริ่มต้นของตัวอักษร โดยวิธีการนับมากลุ่มละ 8 บิท เราก็สามารถแยกตัวอักษรต่างๆได้ เพื่อให้การหาบิทแรกของตัวอักษรตัวแรกเป็นไปอย่างถูกต้องจึงมักส่งชุดของข้อมูลชุดหนึ่งก่อนหน้าการส่งข้อมูลตัวอักษร โดยการส่งตัวอักษรควบคุมความสัมพันธ์ (SYN Transmission Control Character : TC) ชุดของข่าวสารดังกล่าวนี้ประกอบด้วยบิทขนาด 8 บิท คือ 00010110 (มีพาริตีคี่) สำหรับทางด้านรับนั้นจะถูกออกแบบมาให้นำบิทที่รับมาเปรียบเทียบกับชุดของบิทของตัวอักษรควบคุมความสัมพันธ์ (Looking for SYNC) โดยกระทำทุกครั้งที่รับบิทใหม่เข้ามาจนกว่าจะได้ชุดของบิทที่ต้องการดังกล่าว



รูป 5.5 การส่งแบบสัมพันธ์

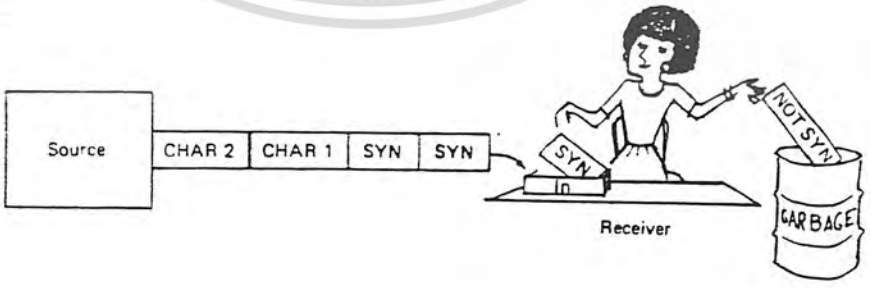
แต่วิธีการดังกล่าวนี้อาจจะเกิดผิดพลาดได้ ถ้าหากชุดของบิทของตัวอักษรที่ตามกันมามีลักษณะของบิทเหมือนกับชุดของบิทของตัวอักษรควบคุมความสัมพันธ์ ทำให้ความสัมพันธ์ที่ได้เกิดผิดพลาดไป

กรณีดังกล่าวแสดงในรูป 5.6a เพื่อแก้ไขความผิดพลาดดังกล่าวนี้เราจึงส่งตัวอักษร SYN จำนวน 2 ตัว ไปก่อนข้อมูลดังรูป 5.6b

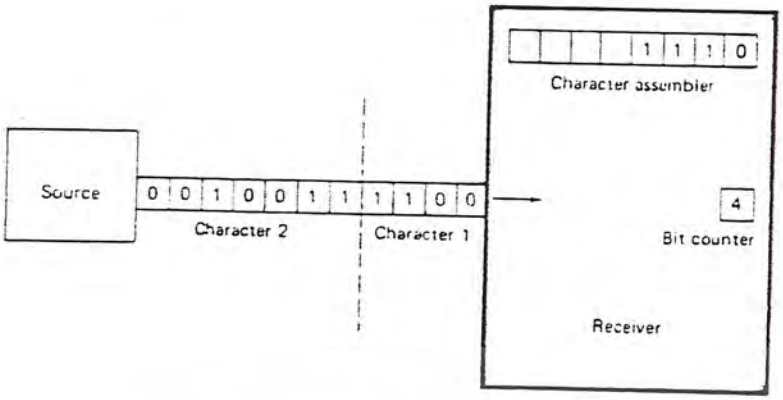


รูป 5.6 การส่งตัวอักษร SYN ไป 2 ตัว เพื่อป้องกันความผิดพลาด

สำหรับทางด้านรับเมื่อรับข่าวสารและตรวจสอบได้ว่าเป็นตัวอักษร SYN แล้ว จะต้องนำชุดของบิทจำนวน 8 บิทต่อไปมาตรวจสอบอีกว่าเป็นอักษร SYN หรือไม่ หากชุดที่ 2 เป็นอักษร SYN ก็แสดงว่าชุด 8 บิทที่นำมาคิดนั้นถูกต้อง ถ้าหากว่า 8 บิทชุดแรกเป็นอักษร SYN แต่ชุดที่ 2 ไม่ใช่ก็จะต้องตรวจสอบต่อไป รูป 5.7 และ รูป 5.8 แสดงถึงวิธีการดังกล่าว



รูป 5.7 การส่งแบบล้มพันธ์ที่มี SYN หลายตัว

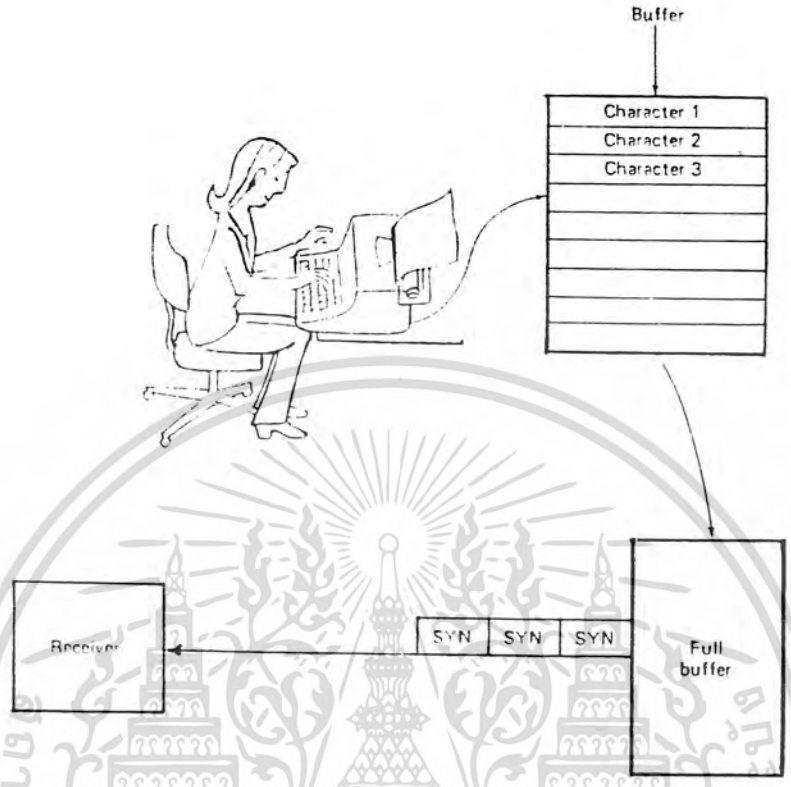


รูป 5.8 การส่งแบบสัมพันธ์

ในระบบส่วนมากมักจะมีตัวอักษร SYN นำหน้าข้อมูล 3-4 ตัว เพื่อให้แน่ใจได้ว่าชุดของบิทสำหรับอักขระที่รับต่อมาคือข้อมูลที่ถูกต้อง

นอกจากวิธีการส่งตัวอักษร SYN หลายตัวติดต่อกันสำหรับการสร้างความสัมพันธ์ของตัวอักษรดังได้กล่าวมาแล้วนั้น ยังมีวิธีการบางชนิดที่ส่งตัวอักษรสองชนิดที่แตกต่างกันไปเพื่อสร้างความสัมพันธ์ โดยทางด้านรับ เมื่อตรวจพบตัวชุดของบิทที่ตรงกับอักษรตัวแรกที่กำหนดแล้วจะต้องตรวจสอบต่อไปว่าอักษรตัวที่สองนั้นตรงกับที่กำหนดไว้หรือไม่

สำหรับการส่งแบบสัมพันธ์นี้ ส่วนมากแล้วข่าวสารที่จะส่งนั้นมักจะถูกรวบรวมหรือเก็บรักษาไว้ในอุปกรณ์ปลายก่อนที่จะส่งออกไป ทั้งนี้เพราะในการส่งแบบสัมพันธ์นั้น อักษรทุกตัวจะต้องถูกส่งออกไปด้วยเวลาที่คงที่ ซึ่งเป็นการยากที่โอเปอเรเตอร์ที่ป้อนข่าวสารจะกระทำได้ และนอกจากนี้ความเร็วในการส่งของสายนั้นมีความเร็วสูงกว่าการทำงานของโอเปอเรเตอร์มาก ฉนั้นอุปกรณ์ปลายโดยทั่วไปจึงมักจะมีส่วนสำหรับเก็บข่าวสารที่เรียกว่า บัฟเฟอร์ (Buffer) สำหรับเก็บรวบรวมข่าวสารที่โอเปอเรเตอร์ป้อนเข้าไป แล้วส่งไปด้วยความเร็วการทำงานของอุปกรณ์ปลายเอง อุปกรณ์เก็บรวบรวมข่าวสารนี้ก็คืออุปกรณ์ทางดิจิตอลที่เรียกว่า เมมโมรี่ (Memory) นั่นเอง สำหรับการใส่ตัวอักษร SYN นั้น โอเปอเรเตอร์ไม่จำเป็นต้องใส่เข้าไป แต่อุปกรณ์ปลายจะใส่เข้าไปให้เองก่อนการส่งข้อมูลไปตามสาย สำหรับปลายอีกด้านหนึ่งนั้นก็ต้องมีอุปกรณ์ที่เรียกว่าบัฟเฟอร์ เพื่อรับข้อมูลที่ส่งมาจากทางด้านส่ง วิธีการส่งแบบสัมพันธ์นี้ทำให้เราสามารถใช้สายสื่อสารได้อย่างเต็มที่ รูป 5.9 แสดงถึงการส่งแบบสัมพันธ์ที่มีบัฟเฟอร์

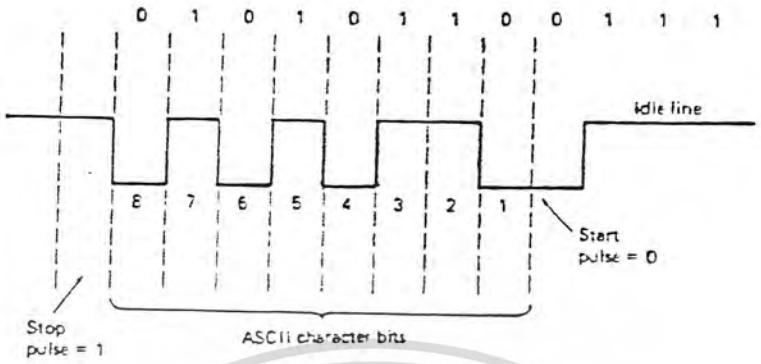


รูป 5.9 การใช้บัฟเฟอร์ในการส่งแบบสัมพันธ์

การส่งข้อมูลแบบไม่สัมพันธ์

การส่งข้อมูลแบบนี้ เราสามารถส่งข้อมูลออกไป โดยไม่จำเป็นต้องมีความสัมพันธ์ระหว่างตัวอักษรว่าจะต้องมีเวลาที่แน่นอนอย่างไร โดยเราอาจจะส่งตัวอักษรติดต่อกันไป หรืออาจจะเว้นช่วงเวลาสำหรับการส่งตัวอักษรเป็นเวลาเท่าใดก็ได้ ในกรณีเช่นนี้ ทางด้านรับจะต้องสร้างความสัมพันธ์ขึ้นใหม่สำหรับอักษรแต่ละตัวที่รับมา ฉนั้นทางด้านรับจะต้องทราบถึงบิทเริ่มต้นของอักขระแต่ละตัวว่าเริ่มต้นที่ใด วิธีการดังกล่าวสามารถกระทำได้โดยการเพิ่มบิทที่เรียกว่า *Start Pulse* โดยเติมเข้าไปข้างหน้าชุดของบิททุกๆตัวอักขระ สำหรับบอกให้ด้านรับทราบถึงบิทเริ่มต้นของอักขระใดๆ ในทางปฏิบัติด้านรับสามารถทราบว่ามี Start Pulse หรือยังโดยการตรวจสอบสถานะของสายซึ่งจะมีสถานะเป็น 0 หรือ 1 ก็ได้ โดยทั่วไปเมื่อไม่มีการส่งข่าวสารมักกำหนดให้สายมีสถานะเป็น 1 เรียกว่า *MARK* สถานะตรงข้ามคือ 0 เรียกว่า *SPACE* เมื่อต้องการส่งตัวอักษร 1 ตัว เครื่องส่งจะต้องส่งบิทที่บอกถึงการเริ่มต้นเรียกว่า บิทเริ่มต้น (*Start Bit*) หรือ *Start Pulse* โดยการส่ง 0

ออกไปตามลาย ด้านรับจะทราบว่าบิทที่ตามมาเป็นข่าวสาร แสดงดังรูปที่ 5.10



รูปที่ 5.10 การส่งข้อมูลแบบไม่สัมพันธ์

ทางด้านรับเมื่อรับทราบถึงการเปลี่ยนสถานะของสายการเริ่มต้น การทำงานจะเริ่มภายหลังเวลาผ่านไปเท่ากับครึ่งเวลาของความกว้างขนาด 1 บิท โดยจะสุ่มตัวอย่าง (ตรวจสอบ) สายว่าขณะนั้นมีสถานะ 0 หรือ 1 ถ้าเป็น 0 แสดงว่ามีการส่งบิทเริ่มต้น หลังจากนั้นจะสุ่มตัวอย่างสถานะของสายทุกๆช่วงเวลา 1 บิทเพื่อหาค่าเวลารหัสของตัวอักขระที่ส่งมา ในการสุ่มตัวอย่างครั้งแรกนั้นหลังจากได้รับการเปลี่ยนแปลงจาก 1 ไป 0 เป็นเวลา 1 บิท หากค่าสถานะของสายที่รับมามีค่า 1 ก็แสดงว่าบิทที่รับมาไม่ใช่บิทเริ่มต้น อาจเป็นเพียงสัญญาณรบกวนที่เรียกว่า *สัญญาณรบกวนอิมพัลส์ (Noise Impulse)* ในกรณีนี้ทางด้านรับจะทิ้งสัญญาณนี้ไป การตรวจสอบบิทเริ่มต้นนี้กระทำทุกครั้งที่มีอักขระตัวใหม่เข้ามา เราจึงสามารถเว้นระยะห่างระหว่างตัวอักขระให้มีเวลาเท่าใดก็ได้ และเมื่อส่งครบบิทก็จะมีบิทสำหรับบอกการสิ้นสุดเรียกว่า *พัลส์การสิ้นสุด'(Stop'Pulse)'* เพื่อให้ด้านรับมีเวลาสำหรับการเตรียมรับข้อมูลตัวต่อไป ในรูปที่ 5.10 จะพบว่าบิทของข้อมูลนั้นมีบิทเริ่มต้นนำหน้าและปิดท้ายด้วยบิทหยุดทุกๆครั้งตามขนาดของบิทที่แทนตัวอักษรหนึ่งตัว พัลส์การหยุดมีค่าเป็น 1 ส่วนระยะเวลาของพัลส์ดังกล่าวนี้จะแตกต่างออกไปสำหรับแต่ละระบบสำหรับรหัสอัลกิย์ พัลส์การหยุดมีขนาดความกว้างเท่ากับขนาดความกว้างของบิทข้อมูลหนึ่งหรือสองบิท ความจริงแล้วช่วงเวลาของบิทการหยุดนั้นเป็นช่วงเวลาสำหรับทางด้านรับจะทำงานหลังจากรับข้อมูลมาครบหนึ่งตัวอักขระ เช่นการพิมพ์ในเครื่องพิมพ์การเจาะรูบนเทปของเครื่องเจาะเทป เป็นต้น ในการทำงานของอุปกรณ์แบบจักรกลนี้มีการทำงานที่ช้ามาก จึงอาจต้องมีเวลาของพัลส์การหยุดเท่ากับเวลาของ 2 บิท แต่อุปกรณ์ประเภทเครื่องจักรไฟฟ้ามีความเร็วในการทำงานมากกว่า จึงมีพัลส์การหยุดเพียง 1 บิท

เนื่องจากอักขระทุกตัวที่ส่งออกมามีความเป็นอิสระต่อกัน และทุกๆอักขระจะมีบิทเริ่มต้น

และบิทสิ้นสุดอย่างน้อยอักขระละ 1 บิท หากเกิดสัญญาณรบกวนอาจทำให้เกิดความผิดพลาดกับข้อมูลได้ ความผิดพลาดนี้จะเกิดขึ้นกับข้อมูลเพียงอักขระเดียว เพราะอักขระแต่ละตัวมีความสัมพันธ์กับบิทเริ่มต้นกับบิทสิ้นสุดของตัวมันเองเท่านั้น

5.1.3 การจำแนกวิธีการส่งตามลักษณะการจัดข้อมูล

นอกจากการจำแนกวิธีการส่งตามลักษณะทิศทางการส่งแล้ว ยังสามารถจำแนกวิธีการส่งออกตามลักษณะการจัดข้อมูลที่ใช้ส่งได้อีกเป็น 2 วิธีคือ

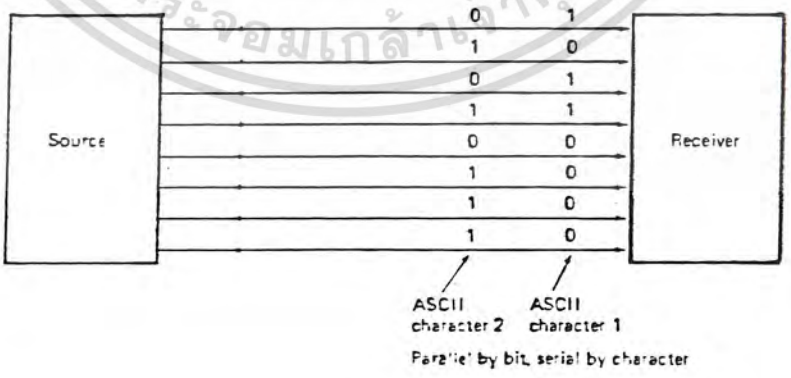
1) การส่งแบบขนาน *(Parallel Transmission)*

2) การส่งแบบอนุกรม *(Serial Transmission)*

การส่งข้อมูลแบบขนาน

วิธีการส่งแบบขนานนั้น เราจะนำเอาทุกๆบิทของรหัสที่ประกอบเป็นอักษรหนึ่งตัวส่งออกไปพร้อมๆกันในเวลาเดียวกัน ซึ่งจะต้องใช้แชนแนลหรือสายส่งเท่ากับจำนวนของบิทที่ประกอบเป็นอักษรหนึ่งตัว นั่นหมายความว่าหากเรามีรหัสขนาด 8 บิทก็ต้องมีแชนแนลสำหรับการส่งจำนวนเท่ากับ 8 แชนแนล ตัวอย่างเช่น โนรหัส *ASCII* มีจำนวนบิทแทนตัวอักษรแต่ละตัวจำนวน 8 บิท เมื่อเราใช้วิธีการส่งแบบขนานต้องมีสายส่งทั้งหมด 8 เส้นทั้งทางด้านส่งและทางด้านรับ ดังรูปที่ 5.11

ในการส่งแบบขนานนั้น ทุกๆบิทของข่าวสารในหนึ่งตัวอักษร จะถูกส่งไปในลักษณะขนานกัน แต่การส่งระหว่างตัวอักษรจะส่งเป็นแบบอนุกรมกัน คือส่งตัวอักษรตัวที่ 1 ที่ 2 ที่ 3 ไปเรื่อยๆจนหมด



รูปที่ 5.11 การส่งข้อมูลแบบขนาน

การส่งแบบขนานนั้นมักใช้ในระบบสื่อสารที่มีระยะทางในการติดต่อไม่ไกลมากนัก และโดยเฉพาะอย่างยิ่งในการส่งข้อมูลระหว่างคอมพิวเตอร์กับอุปกรณ์ประกอบของคอมพิวเตอร์ เช่น จอภาพ เครื่องอ่านเทปแม่เหล็ก เครื่องอ่านจานแม่เหล็ก ตลอดจนเครื่องพิมพ์ และอุปกรณ์อื่นๆ การส่งข้อมูลแบบขนานจะทำให้เราได้ระบบการสื่อสารที่มีอัตราการส่งข้อมูลสูงมาก แต่ในกรณีที่มีระยะทางไกล การส่งข้อมูลแบบขนานจะไม่นิยมเพราะราคาของการวางแชนแนลแบบขนานจะสิ้นเปลืองค่าใช้จ่ายสูงมาก

การส่งข้อมูลแบบอนุกรม

วิธีการส่งข้อมูลที่นิยมแพร่หลายมากที่สุดคือ การส่งข้อมูลแบบอนุกรม ในการส่งแบบอนุกรมนั้น บิททั้งหมดของตัวอักษรหนึ่งตัวจะถูกนำมาส่งออกไปทีละบิท ติดต่อกันไปตามแชนแนลซึ่งมีอยู่เพียงแชนแนลเดียว ดังรูปที่ 5.12 ทางด้านเครื่องรับเมื่อรับข้อมูลมาแล้วก็จะนำมาจัดเป็นตัวอักษรขึ้นใหม่ให้ตรงกับชุดของตัวอักษรที่ทางด้านส่งส่งมา ซึ่งวิธีการดังกล่าวนี้จะต้องประกอบด้วยความสัมพันธ์ในการทำงานระหว่างด้านรับและด้านส่ง เพื่อให้สามารถได้ตัวอักษรที่ถูกต้องกลับมาใช้งาน

รูปที่ 5.12 การส่งข้อมูลแบบอนุกรม

สำหรับการส่งข้อมูลในระบบการสื่อสารนั้น จะใช้วิธีการส่งแบบอนุกรมเพราะ การใช้งานหลักของการสื่อสารคือการส่งข้อมูลในระยะทางไกลๆ

สำหรับการส่งข้อมูลแบบอนุกรมนั้น สิ่งหนึ่งที่จะต้องระมัดระวังคือ ความผิดพลาดเกี่ยวกับความสัมพันธ์ของบิทและของตัวอักขระที่ส่ง เพราะเรานำเอาบิทของอักขระหลายๆตัวมาส่งเรียงกันมาออกทางด้านรับจะต้องสามารถรับบิทได้อย่างถูกต้อง และจะต้องแยกอักขระที่รับมาเป็นอักขระเดิมที่ทางด้านส่งส่งออกมาได้การส่งข้อมูลแบบอนุกรมนี้จะใช้พอร์ทอนุกรม COM 1 และ COM 2 ในการติดต่อ ซึ่งมีมาตราฐานการติดต่อหลายรูปแบบ เช่น CCITT V.28 ,CCITT V.24 หรือ RS-232 แต่ที่นิยมแพร่

หลายมากคือ *RS-232*

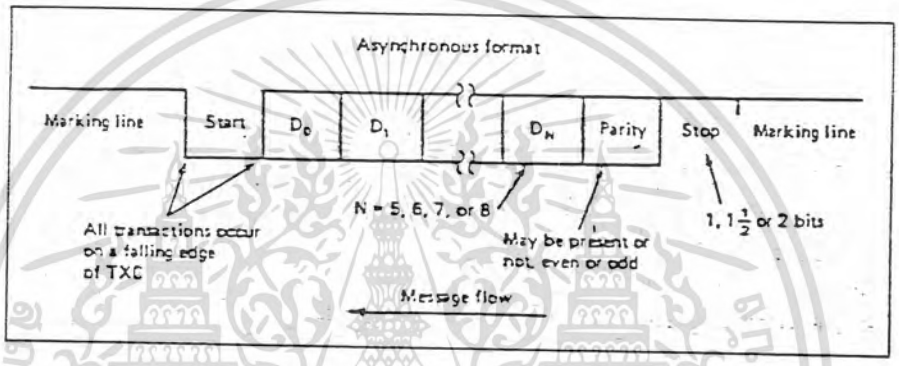
5.2 โปรโตคอลของการสื่อสารแบบอนุกรม

เมื่อพิจารณาการส่งข้อมูลในแบบอนุกรมจะพบว่า ปัญหาหนึ่งที่เกิดขึ้นอยู่เสมอก็คือ การตัดสินว่าข้อมูลที่ได้รับนั้นมีจุดเริ่มต้นที่ใด ดังนั้นจึงมีการกำหนดข้อตกลงในการสื่อสารขึ้นเพื่อแก้ปัญหานี้ ข้อตกลงดังกล่าวเราเรียกว่า *โปรโตคอล (Protocol) ของการสื่อสารข้อมูลแบบอนุกรม* สามารถแบ่งออกได้เป็น 2 ประเภทใหญ่ คือ โปรโตคอลสำหรับการสื่อสารข้อมูลอนุกรมแบบซิงโครนัส (Synchronous) และโปรโตคอลสำหรับการสื่อสารข้อมูลอนุกรมแบบอซิงโครนัส (Asynchronous) การสื่อสารแบบซิงโครนัสนั้น ข้อมูลจะถูกส่งออกไปอย่างสม่ำเสมอ ช่วงเวลาระหว่างบิทและระหว่างเวิร์ดจะมีค่าเท่ากันเสมอ ดังนั้นในการสื่อสารข้อมูลอนุกรมแบบซิงโครนัสจึงต้องมีสายสัญญาณเพิ่มเติมเพื่อกำกับการส่งว่าควรจะส่งเมื่อใด และควรจะหยุดเมื่อใด ระบบที่เป็นซิงโครนัสจะเป็นระบบที่มีความเร็วสูง แต่ก็ยังต่ำกว่าการสื่อสารแบบขนาน

การสื่อสารแบบอซิงโครนัส เป็นหัวใจของการสื่อสารข้อมูลผ่านทางสายโทรศัพท์ในปัจจุบัน การสื่อสารแบบนี้ช่วงเวลาระหว่างบิทจะมีค่าเช่นเดียวกับซิงโครนัส แต่จะแตกต่างกันที่ระยะห่างระหว่างเวิร์ด ซึ่งไม่มีข้อกำหนดว่าจะห่างกันเป็นระยะเวลานานเท่าไร กล่าวคือ ระหว่างเวิร์ดนั้นจะห่างกันกี่วินาที กี่นาที กี่ชั่วโมง กี่วัน กี่เดือน หรือกี่ปีก็ได้ทั้งสิ้น ขึ้นอยู่กับว่าฝ่ายรับจะรอรับได้หรือไม่

เมื่อไม่มีข้อกำหนดทางด้านระยะเวลาระหว่างเวิร์ดแล้ว ทางผู้ส่งและผู้รับจะเข้าใจตรงกันได้อย่างไรว่าที่ใดคือจุดเริ่มต้นและจุดสิ้นสุดของแต่ละเวิร์ด เพื่อแก้ปัญหานี้จึงมีการกำหนดข้อตกลงเกี่ยวกับฟอร์แมตของข้อมูลที่จะส่งให้ทางผู้รับสามารถเข้าใจว่า จุดใดเป็นจุดเริ่มต้นของเวิร์ด ข้อกำหนดดังกล่าวจะกำหนดให้แต่ละเวิร์ดต้องขึ้นต้นด้วยบิทที่เรียกว่า *start bit* หรือ *บิทเริ่มต้น* ซึ่งจะต้องเป็นลอจิกศูนย์เสมอ จากนั้นตามด้วยบิทข้อมูลที่ต้องการส่ง ซึ่งมีความยาว 5 ถึง 8 บิท ถัดจากบิทข้อมูลก็จะเป็นพาริตีบิทซึ่งจะทำหน้าที่เป็นบิทสำหรับตรวจสอบความถูกต้องหรือไม่ บิทพาริตีนี้มี 2 ประเภทคือ *even parity* ซึ่งจะเซ็ต (มีค่าเป็น 1) เมื่อจำนวนบิทที่เป็นลอจิก 1 ในบิทที่เป็นข้อมูลมีจำนวนเป็นคี่ และ *odd parity* ซึ่งจะเซ็ตเมื่อจำนวนบิทที่เป็นลอจิก 1 ในบิทที่เป็นข้อมูลมีจำนวนเป็นคู่ ในการส่งข้อมูลบางครั้งอาจจะไม่มีการใช้บิทพาริตีก็ได้ ถ้าหากการสื่อสารในครั้งนั้นมีความน่าเชื่อถือสูง คือ มีสัญญาณรบกวนต่ำเป็นการเพิ่มความเร็วในการส่งข้อมูลด้วย บิทสุดท้ายในฟอร์แมตก็คือ *stop bit* ทำหน้าที่บอกทางผู้รับว่าขณะนี้ข้อมูลที่ทางผู้รับได้รับนั้นครบเวิร์ดแล้วขอให้เตรียมตัวรับเวิร์ดต่อไป stop bit นี้ถูกกำหนดให้เป็นลอจิก 1 เสมอ ทั้งนี้เพื่อให้ระบบสามารถตรวจสอบบิทเริ่มต้นได้ (บิทเริ่มต้นมีลอจิกเป็นศูนย์) stop bit อาจมีความยาวเป็น 1 บิท 1.5 บิท หรือ 2 บิทก็ได้

จากฟอร์แมตดังกล่าว จะเห็นว่าเรามีฟอร์แมตสำหรับการสื่อสารมากมายไปหมด เช่น 5E1 (5Data bit, Even parity, 1 Stop bit), 7E1 (7Data bit, Even parity, 1 Stop bit) และ 8N1 (8Data bit, No parity, 1 Stop bit) เป็นต้น ในการใช้งานทั่วไปเรานิยมใช้กันอยู่เพียง 2 ฟอร์แมตคือ 7E1 และ 8N1 จะเลือกใช้ฟอร์แมตใดขึ้นอยู่กับสภาพของสายส่งสัญญาณว่ามีสัญญาณรบกวนมากเพียงใด ถ้าหากสายส่งมีสัญญาณรบกวนมากควรจะใช้ 7E1 แต่ถ้าสายส่งมีสภาพดีสัญญาณรบกวนต่ำควรใช้ 8N1 เพราะความเร็วมากกว่า ทั้งนี้ต้องมีการตกลงกันล่วงหน้าระหว่างผู้ส่งกับผู้รับว่าจะใช้ฟอร์แมตใดในการสื่อสารลักษณะของข้อมูลที่ถูกส่งออกไปจะมีลักษณะดังรูป 5.13



รูป 5.13 ตัวอย่างของฟอร์แมตข้อมูล

5.3 OSI มาตรฐานสำหรับซอฟท์แวร์สื่อสาร

แต่เดิมการพัฒนาซอฟท์แวร์สำหรับการสื่อสารนั้นมักจะกระทำกันอย่างไร้ทิศทาง ขนาด ข้อกำหนดที่แน่นอน ทำให้เกิดปัญหาการเข้ากันไม่ได้ของซอฟท์แวร์ กล่าวคือ ในกรณีที่มีการสื่อสารระหว่างเครื่องคอมพิวเตอร์ 2 เครื่อง ทั้ง 2 เครื่องจะต้องวิ่งโปรแกรมสำหรับการสื่อสารโปรแกรมเดียวกันเท่านั้น ทำให้เกิดความไม่สะดวกและเกิดความอ่อนตัวเป็นอย่างมาก ภายหลังจึงมีการกำหนดมาตรฐานร่วมสำหรับซอฟท์แวร์สื่อสารขึ้นมาชุดหนึ่งเรียกว่า *Open System Interfacing* หรือ *OSI* เป็นการกำหนดระดับหรือเลเยอร์ของซอฟท์แวร์สำหรับการสื่อสารขึ้น โดยมีการแบ่งออกเป็น 7 เลเยอร์มีหน้าที่ต่างๆกัน เริ่มจาก *Physical' Layer* เป็นเลเยอร์ที่อยู่ล่างสุดและมักจะเป็นฮาร์ดแวร์ทั้งหมด เป็นข้อกำหนดเกี่ยวกับการเชื่อมต่อของสัญญาณต่างๆ และการรับส่งข้อมูลในระดับสัญญาณไฟฟ้า เช่น ระดับลอจิก ขนาดของกระแส เป็นต้น ถัดจากชั้น Physical Layer ขึ้นมาจะเป็น *Data Link Layer'* เป็นการกำหนดฟอร์แมตของการสื่อสาร เช่น 8N1 หรือ 7E1 เป็นต้น ในการสื่อสารข้อมูลนั้น

อย่างน้อยจะต้องมีสองระดับนี้จึงจะสามารถทำการสื่อสารได้

อีก 5 ชั้นที่เหลือนั้น มักจะพบในการสื่อสารระดับสูงและซับซ้อนมากๆ เช่น ในระบบสื่อสารระดับประเทศ หรือในเครื่องคอมพิวเตอร์ระดับมินิหรือเมนเฟรม เป็นต้น คือ *Network Level* จะเป็นส่วนที่ทำหน้าที่ควบคุมการไหลของข้อมูลให้ไปในทิศทางที่ถูกต้อง ชั้น *Transfer Layer* ทำหน้าที่การควบคุมความน่าเชื่อถือ (Reliability) ของข้อมูล มีการตรวจสอบข้อผิดพลาดในระดับสูง เช่นการใช้ check sum หรือ CRC การทำ error correction เป็นต้น ชั้น *Session Layer* ทำหน้าที่เริ่มและสิ้นสุดการติดต่อข้อมูล *Presentation Layer* ทำหน้าที่กำหนดโครงสร้างของข้อมูลทั้งบล็อค และชั้นสุดท้ายซึ่งอยู่สูงสุดคือ *Application Layer* ซึ่งเป็นส่วนของโปรแกรมของผู้ใช้

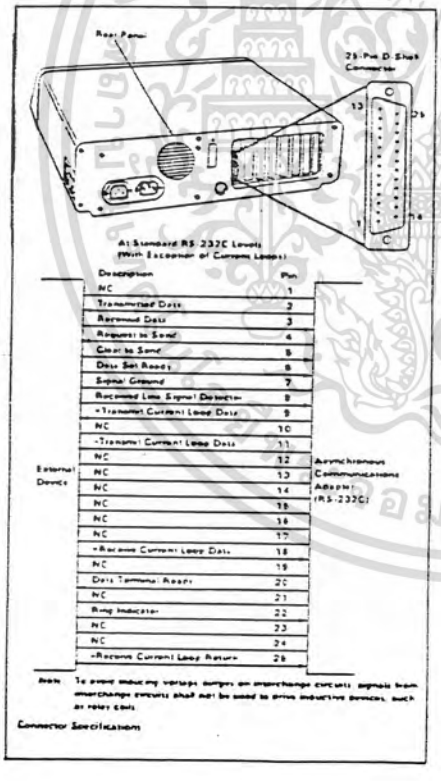
5.4 มาตรฐาน EIA

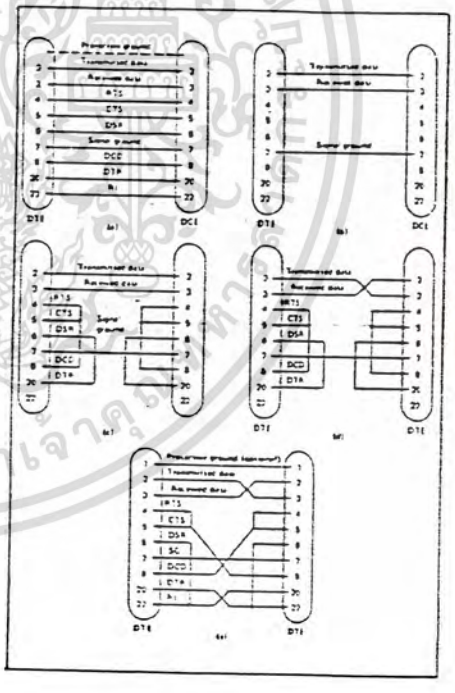
เป็นมาตรฐานที่คุ้นหูคุ้นตากันมากสำหรับประเทศไทย มาตรฐานของ EIA หรือมาตรฐานที่ขึ้นต้นด้วย RS เป็นของสมาคมอุตสาหกรรมอิเล็กทรอนิกส์ (The Electronics Industries Association) สหรัฐอเมริกา ได้กำหนดมาตรฐานต่างๆสำหรับใช้กับอุปกรณ์ทางด้านอิเล็กทรอนิกส์ ซึ่งรวมทั้งอุปกรณ์ทางด้านการสื่อสารข้อมูลด้วยมากมาย ที่รู้จักกันดีที่สุดในวงการคอมพิวเตอร์เมืองไทยคือ RS-232 (RS ย่อมาจาก *Recommended Standard*) หรือบางทีเรียกกันว่า EIA-232 เป็นมาตรฐานที่กำหนดสัญญาณและการเชื่อมต่อของสัญญาณสำหรับการสื่อสารข้อมูลแบบอนุกรม

ข้อกำหนดของ EIA กำหนดให้อุปกรณ์ในการสื่อสารข้อมูลมี 2 ชนิดคือ *DTE* หรือ *Data Terminal' Equipment'* และ *DCE* หรือ *Data Communication Equipment* ซึ่งต่อมาได้เปลี่ยนเป็น *Data' Circuit' Terminating Equipment* อุปกรณ์ที่เป็น DTE นั้นจะอยู่ที่ปลายของสายสัญญาณสำหรับการสื่อสารข้อมูล ตัวอย่างของ DTE ที่เห็นได้ชัดคือเครื่องคอมพิวเตอร์นั่นเอง ส่วน DCE นั้นจะทำหน้าที่แปลงสัญญาณที่ได้จาก DTE ให้อยู่ในสภาพที่เหมาะสมที่จะส่งไปในสายสัญญาณ และทำหน้าที่แปลงสัญญาณที่ได้รับจากสายสัญญาณให้เป็นข้อมูลเพื่อจะส่งให้ DTE ต่อไป

EIA-232 กำหนดให้การเชื่อมต่อระหว่าง DTE และ DCE จะต้องใช้สัญญาณชุดหนึ่งประกอบด้วย สัญญาณต่างๆ 21 เส้น แต่ในการใช้งานมักจะใช้เพียง 9 เส้นเท่านั้น ขาสัญญาณที่ใช้งานอยู่เป็นประจำจะมีลักษณะดังรูป 5.14 ในการเชื่อมต่อระหว่างอุปกรณ์โดยใช้ชุดสัญญาณของ RS-232 นั้นสามารถทำได้หลายวิธีขึ้นอยู่กับว่าเป็นการเชื่อมต่อระหว่างอุปกรณ์ชนิดใด และความต้องการของผู้ใช้ที่ว่าต้องการให้มีการตรวจสอบฝ่ายตรงข้าม หรือที่เรียกว่าเป็นการทำ *Hand Shake* หรือไม่ การเชื่อมต่อดังกล่าวแสดงไว้ในรูป 5.15a เป็นการเชื่อมต่อระหว่าง DTE กับ DCE ในแบบ FDX มาตรฐาน รูป b เป็นการเชื่อมต่อระหว่าง DTE กับ DCE ในลักษณะของ FDX แบบประหยัด คือใช้สาย

สัญญาณเพียง 3 เส้นเท่านั้น รูป c เป็นการเชื่อมต่อระหว่าง DTE กับ DCE ในแบบ Loop Back วิธีการนี้จะไม่มีการทำ Hand Shake ระหว่างอุปกรณ์ทั้งสองตัว แต่จะเป็นการหลอกตัวเองโดยใช้การป้อนกลับของสัญญาณเอาท์พุทกลับไปให้อินพุทของตัวเอง รูป d เป็นการเชื่อมต่อระหว่าง DTE กับ DCE ด้วยกัน (อาจจะต่อระหว่าง DTE กับ DCE ด้วยกันก็ได้) และมีการทำ Loop Back ลังเกตุที่สัญญาณ Txd และ Rxd ของแต่ละตัวจะมีการสลับกัน ลักษณะนี้เรียกว่า *Null Modem* รูป d เป็นการต่อแบบ Null Modem พร้อมกับมีการทำ Hand Shake ระหว่างอุปกรณ์ทั้งลองเรียกว่า เป็น





รูป 5.14

รูป 5.15

การทำ Double-Cross ในการเชื่อมต่อระหว่างคอมพิวเตอร์ซึ่งเป็น DTE กับโมเด็มซึ่งเป็น DCE จะใช้การเชื่อมต่อในแบบแรก (แบบ a) ส่วนการเชื่อมต่อระหว่างคอมพิวเตอร์ด้วยกัน อาจจะใช้แบบ d หรือแบบ e ก็ได้ ขึ้นอยู่กับว่าต้องการให้มี Hand Shake หรือไม่ การทำ Hand Shake จะได้ประโยชน์ในแง่ที่สามารถรับประกันได้ว่า ข้อมูลทางด้านผู้ส่งจะถูกส่งออกไปเมื่อทางผู้รับพร้อมที่จะรับเท่านั้น เท่ากับเป็นการป้องกันความผิดพลาดที่อาจเกิดขึ้นได้ อย่างไรก็ตาม การเขียนโปรแกรมเพื่อควบคุมก็จะยุ่งยากมากขึ้น

5.5 CCITT

นอกจากมาตรฐานของ EIA แล้ว ในด้านการสื่อสารยังมีอีกองค์กรหนึ่งที่จะต้องกล่าวถึงนั่นคือ *CCITT' หรือ The' Consultative' Commitee In International Telegraphy and telephony'* หน่วยงานหนึ่งของ *ITU' (International' Telecommunication Union)* องค์กรระหว่างประเทศที่ทำหน้าที่ กำหนดมาตรฐานของการสื่อสารระหว่างประเทศทั้งประเภทการสื่อสารด้วยสายสัญญาณ และการสื่อสารแบบไร้สาย CCITT จะเป็นกลุ่มที่ทำหน้าที่ออกมาตรฐานเกี่ยวกับการสื่อสารข้อมูลออกมามากมาย ที่สำคัญคือ V.28 ที่สามารถใช้แทน RS-232 ได้ และ X.25 ซึ่งเป็นโปรโตคอลการสื่อสารข้อมูลที่ใช้กันแพร่หลายทั่วโลก เป็นต้น รูป 5.16 เป็นตัวอย่างของมาตรฐานที่ทาง CCITT ได้นำออกมาให้ใช้ เป็นที่น่าสังเกตุอยู่ว่า CCITT ใช้คำว่า Recommendation ซึ่งหมายถึงข้อแนะนำ แทนที่จะเป็น Standard ที่แปลว่า มาตรฐาน ทั้งที่อาจเนื่องจาก CCITT ไม่มีอำนาจในการบังคับให้ประเทศต่างๆใช้ข้อกำหนดดังกล่าว แต่ถ้าไม่ใช้ก็จะติดต่อกับประเทศอื่นๆไม่ได้

มาตรฐานหรือข้อแนะนำที่ทาง CCITT กำหนดออกมานั้น จะมีเป็นฉบับต่างๆกันทุกปี โดยจะมีการประชุมกันระหว่างประเทศสมาชิก เพื่อหาริอและกำหนดข้อแนะนำใหม่ๆออกมาให้ใช้กัน การออกข้อแนะนำ หรือความจริงก็คือข้อบังคับนั่นเอง จะออกกันเป็น series ตั้งแต่ A ถึง Z แต่ละตัวอักษรจะเป็นเรื่องย่อยๆต่างๆกันมากมาย นอกจากนี้ยังมีการกำหนดสีและปีอีกด้วย ดังนั้นการอ้างอิงถึงมาตรฐานของ CCITT จึงต้องระบุเป็นปีและสี เช่น CCITT Recommendation Orange Book Series X จะเป็นการกล่าวถึง Public Data Network Data Transittion ดังรูป 5.16

5.6 ANSI และ ISO

ชื่อ ISO นี้น่าจะเป็นที่คุ้นหูของผู้ที่อยู่ในวงการอุตสาหกรรมเป็นอย่างมาก เพราะ ISO เป็นองค์การระหว่างประเทศที่ทำหน้าที่กำหนดมาตรฐานต่างๆทางด้านอุตสาหกรรม ชื่อเต็มๆของ ISO คือ *International' Standard' Organization* มีที่ทำการอยู่ที่เจนีวา ประเทศสวิตเซอร์แลนด์ ISO

จะประสานงานกับ CCITT อย่างใกล้ชิด ส่วน ANSI นั้นก็คือองค์การที่ทำหน้าที่กำหนดมาตรฐานผลิตภัณฑ์อุตสาหกรรมของสหรัฐอเมริกา ชื่อของ ANSI นั้นจะเป็นที่รู้จักกันมากในวงการ BBS (Bulletin Board System) ในฐานะของเจ้าของรหัสมหัศจรรย์ที่สามารถทำให้เราควบคุมการทำงานของจอภาพทางด้านการแสดงสี และการเลื่อนเคอร์เซอร์ตลอดจนการควบคุมการใช้งานคีย์บอร์ดจากคอมพิวเตอร์ที่อยู่อีกแห่งได้ ทั้งนี้จุดประสงค์เดิมของการกำหนดชุดรหัสดังกล่าวนั้น มีไว้สำหรับใช้ควบคุมจอเทอร์มินัล สำหรับเครื่องคอมพิวเตอร์ในระดับเมนเฟรม มีชื่อเรียกกันว่า ANSI Terminal ต่อมาเมื่อคอมพิวเตอร์มีราคาถูกลง มีการนำเอาไมโครคอมพิวเตอร์มาใช้เป็นเทอร์มินัลแทนเทอร์มินัลแบบเดิมจึงจำเป็นต้องทำให้ไมโครคอมพิวเตอร์สามารถรับรหัสของANSI ได้ สำหรับเครื่องในตระกูล IBM นั้น จะสามารถใช้รหัสของ ANSI ได้โดยการติดตั้งดีไวซ์ไดร์เวอร์ที่ชื่อว่า *ANSI.SYS* ซึ่งมากับ DOS ทุกเวอร์ชันอยู่แล้ว

Data Transmission Recommendations of the CCITT

The X Series of CCITT for data transmission over public data networks.

Number	*Title*
X.1	International user classes of service in public data networks
X.2	International user facilities in public data networks
X.4	General structure of signals of International Alphabet No. 5 code for data transmission over public data networks
X.20	Interface between data terminal equipment and data circuit-terminating equipment for start-stop transmission services on public data networks
X.21	General purpose interface between data terminal equipment and data circuit-terminating equipment for synchronous operation on public data networks
X.24	List of definitions of interchange circuits between data terminal equipment and data circuit-terminating equipment on public data networks
X.25	Interface between data terminal equipment and data circuit-terminating equipment for terminals operating in the packet mode on public data networks
X.26	Electrical characteristics for unbalanced double-current interchange circuits for general use with integrated circuit equipment in the field of data communications
X.27	Electrical characteristics for balanced double-current interchange circuits for general use with integrated circuit equipment in the field of data communications
X.28	Standardization of basic model page printing machine in accordance with International Alphabet No. 5
X.29	Characteristics for start-stop data terminal equipment using International Alphabet No. 5
X.32	Answer back units for 200 baud start-stop machines in accordance with International Alphabet No. 5
X.33	Standardization of an international test for the measurement of the margin of start-stop machines in accordance with International Alphabet No. 5
X.40	Standardization of frequency-shift modulated transmission systems for the provision of telegraph and data channels by frequency division of a primary group
X.50	Fundamental parameters of a multiplexing scheme for the international interface between synchronous data networks
X.51	Fundamental parameters of a multiplexing scheme for the international interface between synchronous data networks using 10-bit envelope structure
X.60	Common channel signaling for synchronous data applications-data user part
X.70	Terminal and transit control signaling for start-stop services on international circuits between asynchronous data networks
X.71	Decentralized terminal and transit control signaling system on international circuits between synchronous data networks
X.92	Hypothetical reference connections for public synchronous data networks
X.95	Network parameters in public data networks
X.96	Call progress signals in public data networks

The V Series of CCITT for data transmission over the voice circuit or analog networks

V.1	Equivalence between binary notation symbols and the significant conditions of a two-condition code
V.2	Power levels for data transmission over telephone lines
V.3	International Alphabet No. 5 for transmission of data and messages
V.4	General structure of signals of the 5-unit code for data and message transmission
V.10	Use of the telex network for data transmission at the modulation rate of 50 baud
V.11	Automatic calling and or answering on the telex network
V.13	Answer-back unit simulators
V.15	Use of acoustic couplers for data transmission
V.21	200 baud modem standardized for use in the general switched telephone network
V.22	Standardization of modulation rates and data signaling rates for synchronous data transmission in the general switched telephone network
V.22B	Standardization of modulation rates and data signaling rates on leased telephone circuits
V.23	600/1200 baud modem standardized for use on the general switched telephone networks
V.24	Functions and electrical characteristics of circuits at the interface between data terminal equipment and data circuit-terminating equipment
V.25	Automatic calling and/or answering on the general switched telephone network
V.26	2400 bits/second modem for use on four-wire leased point-to-point circuits
V.26B	2400 bits/second modem for use on the general switched telephone network
V.27	Modem for data signaling rates up to 4800 bits/second over leased circuits
V.28	Electrical characteristics for interface circuits
V.30	Parallel data transmission system for universal use on the general switched telephone network
V.31	Electrical characteristics for contact closure-type interface circuits
V.35	Transmission of 48 kilobits/second data using 60 to 108 kHz group bank circuits
V.40	Error indication with electromechanical equipment
V.41	Code-independent error control system
V.50	Standard limits for transmission quality of data transmission
V.51	Organization of the maintenance of international telephone-type circuits used for data transmission
V.52	Characteristics of distortion and error rate measuring apparatus for data transmission
V.53	Limits for the maintenance of telephone-type circuits used for data transmission
V.56	Comprehensive tests for modems that use their own interface circuits
V.57	Comprehensive test set for high transmission rates

รูป 5.16 ตัวอย่างของ CCITT Recommendation Orange Book Series X ที่กล่าวถึงการถ่ายทอดข้อมูลในเครือข่ายข้อมูลสาธารณะ

5.7 Bell Lab

Bell เป็นองค์การของเอกชนที่ควบคุมทางด้านการสื่อสารของสหรัฐอเมริกา เนื่องจากระบบโทรศัพท์ของสหรัฐอเมริกาส่วนใหญ่เป็นของ Bell ดังนั้นมาตรฐานที่กำหนดโดย Bell จึงเปรียบเสมือนกับเป็นมาตรฐานของระบบโทรศัพท์ของสหรัฐอเมริกา อย่างไรก็ตามอุปกรณ์ที่ตรงตามมาตรฐานของ Bell ส่วนใหญ่จะสามารถใช้งานร่วมกับอุปกรณ์ที่ตรงตามมาตรฐาน CCITT เช่น Bell 212A จะเทียบเท่ากับ CCITT V.22 ซึ่งเป็นมาตรฐานสำหรับโมเด็มขนาด 1200 bps เป็นต้น

บทที่ 6
พอร์ทอนุกรมของเครื่องคอมพิวเตอร์ไอบีเอ็มพีซี

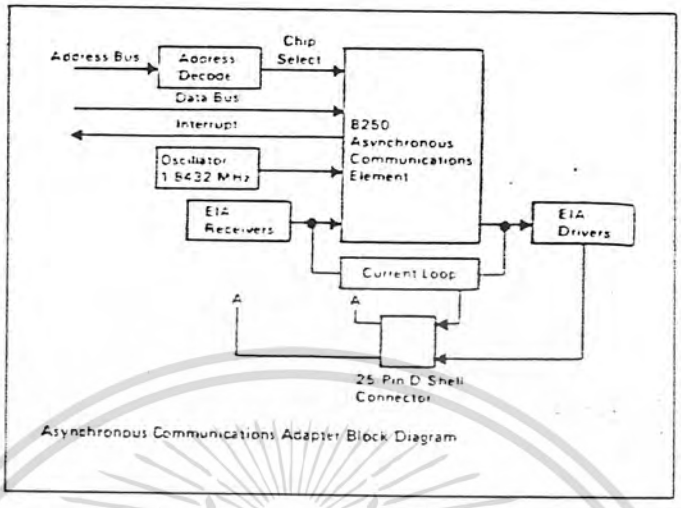
พอร์ทเป็นสิ่งหนึ่งที่ทางผู้ผลิตคอมพิวเตอร์มักจะบรรจุลงในสินค้า บนเครื่องคอมพิวเตอร์ในตระกูลไอบีเอ็มพีซีนั้น เรามักจะได้ยินชื่อพอร์ทมากมายหลายชื่อ เช่น เกมพอร์ท พอร์ทเครื่องพิมพ์ พอร์ทขนาน เม้าส์พอร์ท พอร์ทอนุกรม พอร์ท RS-232 พอร์ทโมเด็ม เป็นต้น ทั้งนี้พอร์ทที่เราคุ้นเคยและมักจะถามหาก็มักจะมีเพียงพอร์ทสำหรับต่อกับจอยสติก หรือเกมพอร์ท และพอร์ทสำหรับต่อกับเครื่องพิมพ์หรือพอร์ทขนานเท่านั้น อาจจะมีอยู่อีกส่วนหนึ่งที่มักจะถามหาพอร์ทอนุกรม (เพื่อจะได้มีพอร์ทครบทุกแบบ) ในจำนวนนี้จะมีเพียงประมาณครึ่งเดียวที่รู้ว่าพอร์ทอนุกรมมีไว้ใช้ทำอะไร และในจำนวนผู้รู้ว่าจะเอาพอร์ทไปใช้ทำอะไรนั้น จะมีอยู่ไม่ถึงครึ่งที่สามารถเขียนโปรแกรมใช้งานพอร์ทได้ทั้งๆที่พอร์ทอนุกรมที่มีความสำคัญไม่แพ้พอร์ทอื่นๆเลย

6.1 หน้าที่ของพอร์ทอนุกรม

จุดประสงค์แรกเริ่มที่มีการประดิษฐ์พอร์ทอนุกรมขึ้นมานั้น ก็เพื่อใช้เป็นพอร์ทสำหรับสื่อสารข้อมูลระหว่างเครื่องคอมพิวเตอร์ด้วยกัน ดังจะเห็นได้จากที่บางครั้งจะมีคนเรียกพอร์ทนี้ ว่า Communication Port แต่เนื่องจากในช่วงแรกๆ ที่เรารู้จักพอร์ทอนุกรมจากเครื่องคอมพิวเตอร์ในระดับไมโครคอมพิวเตอร์นั้น การสื่อสารข้อมูลระหว่างเครื่องคอมพิวเตอร์ไม่ใช่เรื่องที่น่าสนใจนัก จึงทำให้ความหมายของพอร์ทชนิดนี้เปลี่ยนไป จนกระทั่งมีผู้ประยุกต์นำเอาพอร์ทอนุกรมมาใช้ต่อกับเม้าส์ จึงทำให้มันเริ่มมีความสำคัญมากขึ้นอีกครั้ง

6.2 โครงสร้างของพอร์ตอนุกรมบนไอบีเอ็มพีซี

รูป 6.1 เป็นการแสดงถึงส่วนประกอบต่างๆของพอร์ทอนุกรม จะเห็นว่าพอร์ทอนุกรมบนไอบีเอ็มพีซีมีชิป 8250 เป็นหัวใจ ทุกๆส่วนที่เกี่ยวข้องกับพอร์ทอนุกรมจะต้องมีการติดต่อกับชิปนี้ ไม่ทางตรงก็ทางอ้อม ส่วนอื่นๆนอกเหนือจากนี้ก็มีการติดต่อกับชิพยู จะมีการ decode ค่าของ address bus ตามโครงสร้างทั่วไปของระบบคอมพิวเตอร์ ส่วนที่เป็น oscillator จะทำหน้าที่สร้างสัญญาณนาฬิกาเพื่อใช้เป็นเวลาอ้างอิงสำหรับชิป 8250 ในการทำงานต่างๆ เช่น การรับส่งข้อมูล เป็นต้น ส่วนไดร์เวอร์ทำหน้าที่ขับสัญญาณให้มีคุณสมบัติตามมาตรฐาน EIA-232 ทั้งนี้การเชื่อมต่อระหว่างอุปกรณ์ผ่านพอร์ทอนุกรมของไอบีเซ็มพีซี จะใช้มาตรฐานของ EIA-232 (หรือ RS-232) เป็นหลัก



รูป 6.1 โครงสร้างของพอร์ทอนุกรมของ IBM PC

6.3 8250 พอร์ทอนุกรมลำเร็จรูป

ชิป 8250 ผลิตโดยบริษัท Intel เป็นชิปที่ทำหน้าที่สำหรับการสื่อสารแบบ SerialAsynchronous โดยเฉพาะสัญญาณต่างๆจะคอมแพทกับมาตรฐานของ EIA-232 โดยจะต้องใช้ LineDriver เป็นตัวขับสัญญาณ การใช้งาน 8250 นั้น จะกระทำโดยการโปรแกรมค่าของริจิสเตอร์ต่างๆภายในชิปให้เป็นไปตามต้องการ ริจิสเตอร์ต่างๆภายใน 8250 จะมีแอดเดรสดังรูป 6.2 โดย primary address จะเป็นริจิสเตอร์สำหรับ COM1 และ secondary address จะเป็นริจิสเตอร์สำหรับ COM2

ริจิสเตอร์ของ 8250

ริจิสเตอร์ของ 8250 มีหน้าที่ดังต่อไปนี้

- Line Control Register (LCR)

LCR จะอยู่ที่ Offset ที่ 3 นับจาก Base Address เช่นพอร์ท COM1 มี Base Address ที่ 03F8 เราจะได้ตำแหน่งของ LCR สำหรับ COM1 ที่ 03F8 + 3 คือที่ 03FB นั่นเอง LCR เป็นริจิสเตอร์สำหรับควบคุมฟอร์แมตของข้อมูลที่ใช้ในการสื่อสาร เช่น ความยาวของข้อมูล จำนวน Stop Bit ชนิดของ Parity Bit และอื่นๆ ดังรูป 6.3

I/O Decode (in Hex)			
Primary Adapter	Alternate Adapter	Register Selected	DLAB State
3F8	2F8	TX Buffer	DLAB=0 (Write)
3F8	2F8	RX Buffer	DLAB=0 (Read)
3F8	2F8	Divisor Latch LSB	DLAB=1
3F9	2F9	Divisor Latch MSB	DLAB=1
3F9	2F9	Interrupt Enable Register	
3FA	3FA	Interrupt Identification Registers	
3FB	2FB	Line Control Register	
3FC	2FC	Modem Control Register	
3FD	2FD	Line Status Register	
3FE	2FE	Modem Status Register	

I/O Decodes

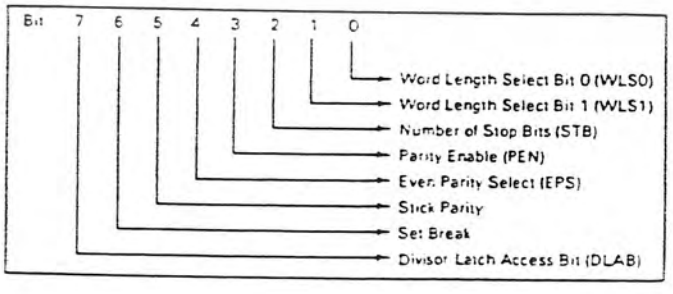
Hex Address 3F8 to 3FF and 2F8 to 2FF											
A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	DLAB	Register
1	1/0	1	1	1	1	1	x	x	x		
							0	0	0	0	Receive Buffer (read), Transmit Holding Reg (write)
							0	0	1	0	Interrupt Enable
							0	1	0	x	Interrupt Identification
							0	1	1	x	Line Control
							1	0	0	x	Modem Control
							1	0	1	x	Line Status
							1	1	0	x	Modem Status
							1	1	1	x	None
							0	0	0	1	Divisor Latch (LSB)
							0	0	1	1	Divisor Latch (MSB)

Note: Bit 8 will be logical 1 for the adapter designated as primary or a logical 0 for the adapter designated as alternate (as defined by the address jumper module on the adapter).

A2, A1 and A0 bits are "don't cares" and are used to select the different register of the communications chip.

Address Bits

รูป 6.2 แอดเดรสของรีจิสเตอร์ต่างๆของ 8250 primary address จะเป็นรีจิสเตอร์สำหรับ COM1 และ secondary address จะเป็นรีจิสเตอร์สำหรับ COM2



รูป 6.3 หน้าที่ของบิทต่างๆใน LCR

บิทที่ 0 และ 1 เป็นบิทที่ใช้ควบคุมความยาวของข้อมูล สามารถกำหนดได้ตั้งแต่ 5 ถึง 8 บิท ดังตาราง 6.1

Bit 0	Bit 1	Word Length
0	0	5 bits
0	1	6 bits
1	0	7 bits
1	1	8 bits

ตาราง 6.1

บิทที่ 2 จะใช้กำหนดความยาวของ Stop Bit โดยถ้าบิท 2 เป็น 0 เป็นการเลือกใช้ Stop Bit เพียงบิทเดียว แต่ถ้าบิท 2 เป็น 1 และเลือกใช้ความยาวของเวิร์ดเป็น 5 บิท เป็นการเลือกใช้ 1.5 Stop Bit และถ้าใช้ความยาวของเวิร์ดเป็น 6 ถึง 8 บิท จะเป็นการเลือกใช้ 2 Stop Bit

ถ้าข้อมูลในบิทที่ 3 เป็น 1 หมายความว่าใช้ Parity Bit ในฟอร์แมต แต่ถ้าเป็น 0 หมายความว่าไม่ใช้ Parity Bit ในการสื่อสาร

บิทที่ 4 จะเป็นการเลือกใช้ว่า ถ้าหากเลือกใช้ Parity Bit แล้ว จะเลือกใช้แบบใด ในกรณีที่บิทนี้มีค่าเป็น 1 จะเป็นการเลือกใช้ Even Parity และถ้าเป็น 0 จะใช้ Odd Parity บิทนี้จะไม่มีความหมายถ้าบิทที่ 3 เป็น 0

บิทที่ 5 จะเป็นบิทที่มีความซับซ้อนเล็กน้อย และไม่ค่อยใช้ในการสื่อสารทั่วไป มักใช้ในการ Initial การสื่อสาร ถ้าบิทที่ 5 เป็น 1 พร้อมกับบิทที่ 3 Parity Bit ที่ส่งออกไปจะมีค่าตรงข้ามกับค่าในบิทที่ 4 เช่นถ้าบิทที่ 4 เป็น 0 ในขณะที่บิทที่ 3 และ 5 เป็น 1 แล้ว ค่า 1 จะถูกส่งออกไปในตำแหน่งของ Parity Bit ของข้อมูล แต่ถ้าบิทที่ 4 เป็น 1 ข้อมูลที่ถูกส่งออกไปจะมีค่าเป็น 0 แทน

เมื่อใดก็ตามที่ข้อมูลในบิทที่ 6 มีค่าเป็น 1 ระดับสัญญาณที่ขา SOUT ซึ่งเป็นเอาท์พุตของข้อมูลที่จะส่งออกไปยังจุดหมายปลายทาง จะถูกกดให้เป็น 0 ตลอดเวลา ไม่ว่าขณะนั้นจะมีการส่งข้อมูลหรือไม่ก็ตาม จนกว่าจะมีการเคลียร์ค่าในบิทที่ 6 นี้ให้เป็นเป็น 0 ค่าของเอาท์พุตดังกล่าวจึงจะขึ้นอยู่กับข้อมูลที่ส่งออกไปอีกครั้ง ปกติบิทนี้จะไม่ใช้งาน

บิทที่ 7 เป็นบิทพิเศษที่ใช้งานใน 8250 เอง เรียกว่า Divisor Latch Access Bit หรือ DLAB ในกรณีที่ DLAB มีค่าเป็น 1 จะเป็นการบอก 8250 ว่าเราต้องการโปรแกรมค่าของ Bit Rate และถ้าหากเป็น 0 จะเป็นการใช้งานปกติ

- *Divisor Latch Least Significant (DLL)*

DLL จะอยู่ที่ตำแหน่ง Base Address ของพอร์ท เช่น 03F8 ของ COM1 หรือ 02F8 สำหรับ COM2 ทั้งนี้ตำแหน่งนี้จะเป็น DLL ก็ต่อเมื่อ DLAB มีค่าเป็น 1 ค่าที่เราจะใส่ใน DLL จะเป็นไบต์ต่ำของตัวหาร ที่จะนำไปใช้ในการหารสัญญาณนาฬิกา 1.8432 MHz. ให้เป็น Bit Rate สำหรับการส่งข้อมูล ในกรณีที่ DLAB เป็น 0 รีจิสเตอร์ DLL ก็จะกลายเป็น RBR หรือ THR ซึ่งจะกล่าวต่อไปภายหลัง

- *Divisor Latch Most Significant (DLM)*

DLM จะอยู่ที่ Offset ที่ 1 นับจาก Base Address เช่น 03F8 + 1 คือ 03F9 สำหรับ COM1 เป็นต้น เป็นรีจิสเตอร์สำหรับใส่ค่าไบต์สูงของตัวหารสัญญาณนาฬิกา เพื่อให้ได้ Bit Rate ที่ต้องการ เช่นเดียวกับ DLL ตำแหน่งที่ DLM อยู่จะทำหน้าที่เป็น DLM ก็ต่อเมื่อ DLAB เป็น 1 ถ้าหากเป็น 0 ก็จะกลายเป็น IER ซึ่งจะกล่าวต่อไปนี้

การโปรแกรมค่า Bit Rate นั้น จะคำนวณโดยใช้สูตร

$$\text{Divisor} = \frac{1.8432\ \text{MHz}}{\text{Bit Rate} \times 16}$$

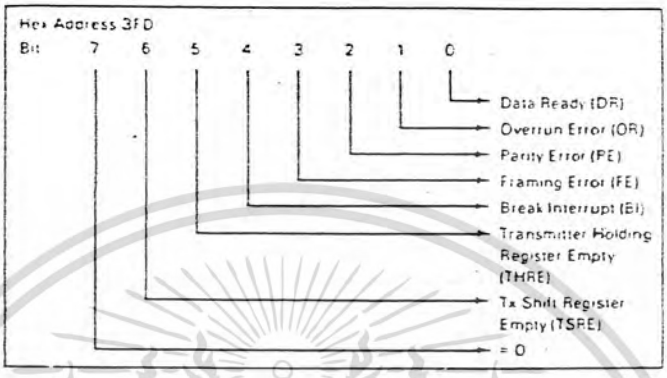
โดยค่า Divisor จะต้องปัดให้เป็นเลขจำนวนเต็ม และจะต้องมีค่าไม่เกิน 65535 เราจะนำค่านี้ไปใส่ใน DLL และ DLM เพื่อให้ได้ Bit Rate ที่ต้องการ การปัดให้เป็นจำนวนเต็มนี้อาจจะทำให้เกิดค่าความผิดพลาดขึ้น เราจำเป็นจะต้องคำนวณค่าความผิดพลาดแล้วดูว่า มีค่ามากเกินไปหรือไม่ ค่าผิดพลาดที่ดีไม่ควรเกิน 0.5 เปอร์เซ็นต์ การคำนวณหาค่าผิดพลาด คำนวณได้จากสูตร

$$\text{Error (\%)} = \left| \frac{\text{Bit Rate - Actual Bit Rate}}{\text{Bit Rate}} \right| \times 100$$

โดยค่า Bit Rate ก็คือค่า Bit Rate ที่เราต้องการ ส่วน Actual Bit Rate ก็คือ Bit Rate ที่ได้จากการปัดเศษตัวหารแล้วคำนวณกลับ

- *Line Status Register (LSR)*

LSR เป็นรีจิสเตอร์ที่อยู่ที่ Offset ที่ 5 นับจาก Base Address ทำหน้าที่เก็บสถานะต่างๆ ในขณะนั้นไว้ โดยมีฟอร์แมตดังรูป 6.4



รูป 6.4 ตำแหน่งและหน้าที่ของบิทต่างๆใน LSR

บิทที่ 0 หรือ DR (Data Ready) เมื่อมีค่าเป็น 1 จะหมายความว่าได้รับข้อมูลจากทางผู้ส่งและข้อมูลได้รออยู่ในบัฟเฟอร์เรียบร้อยแล้ว

บิทที่ 1 หรือ OR (Overrun Error) เมื่อมีค่าเป็น 1 จะแสดงว่า เกิดการ Overrun คือ ข้อมูลที่ได้รับในบัฟเฟอร์ได้ถูกเขียนทับโดยข้อมูลที่ได้รับเข้ามาใหม่เรียบร้อยแล้ว นั่นคือ ข้อมูลที่อยู่ในขณะนี้เป็นข้อมูลที่เกิดการสูญหาย อาจจะนำไปใช้งานไม่ได้อีกต่อไป

บิทที่ 2 หรือ PE (Parity Error) เมื่อมีค่าเป็น 1 แสดงว่าข้อมูลที่ได้รับมี Parity ที่ได้รับไม่ตรงกัน หรือพูดง่ายๆก็คือข้อมูลที่ได้รับอาจผิดพลาดนั่นเอง

บิทที่ 3 หรือ FE (Framing Error) เมื่อมีค่าเป็น 1 แสดงว่า Stop Bit ที่ได้รับมีข้อผิดพลาดเกิดขึ้น อาจจะเกิดจากความยาวของ Stop Bit ผิดพลาด หรือสูญหายก็ได้

บิทที่ 4 หรือ BI (Break Interrupt) จะมีค่าเป็น 1 เมื่อ 8250 ได้รับลอจิก 0 จากทางผู้ส่ง เป็นระยะเวลานานเกินกว่าความยาวของแต่ละเวิร์ด แสดงว่าต้นทางได้ทำการ Break โดยการเซ็ตบิทที่ 6 ของ LCR ให้เป็น 1

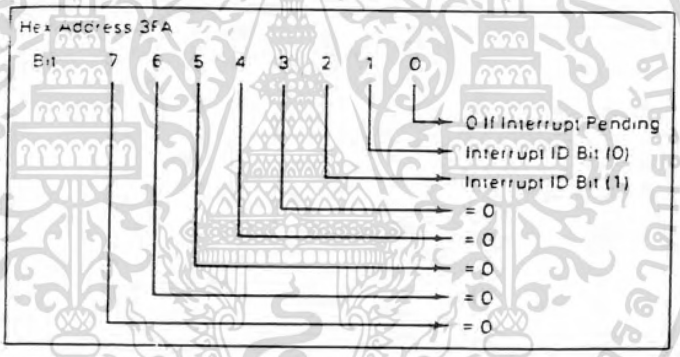
บิทที่ 5 หรือ THRE (Transmitter Holding Register Empty) เป็นบิทที่บอกว่าบัฟเฟอร์สำหรับการส่งข้อมูลออกนั้นได้ว่างแล้ว โดยจะมีค่าเป็น 1 ถ้าหากบัฟเฟอร์ดังกล่าวว่าง

บิทที่ 6 หรือ TSRE (Transmitter Shift Register Empty) เป็นบิทที่เมื่อเซ็ทเป็น 1 แล้ว จะหมายความว่าขณะนี้บัฟเฟอร์ที่ทำหน้าที่แปลงข้อมูลจากขนานไปเป็นอนุกรมได้ว่างลงแล้ว พร้อมที่จะรับข้อมูลตัวต่อไป รีจิสเตอร์ตัวนี้จะรับข้อมูลโดยตรงจาก THR หรือ Transmitter Holding Register อีกทีหนึ่งไม่สามารถเรียกใช้งานโดยตรงได้

บิทที่ 7 ไม่มีความหมายใดๆ แต่จะมีค่าเป็น 0 ตลอดเวลา เมื่อใดที่อ่านแล้วได้ข้อมูลเป็น 1 อาจเป็นไปได้ 2 กรณีคือ อ่านผิดที่ หรือ 8250 เสีย

- Interreupt Identification Register (IIR)

หน้าที่ของ IIR ก็คือ รายงานประเภทของการอินเทอร์รัพท์ที่เกิดขึ้น มีที่ใช้น้อยมาก เพราะการเขียนโปรแกรมให้ใช้การอินเทอร์รัพท์นั้นยุ่งยากมาก หน้าที่และตำแหน่งของบิทต่างๆใน IIR แสดงไว้ในรูป 6.5 IIR จะอยู่ที่ Offset ที่ 2 จาก Base Address



รูป 6.5 หน้าที่และตำแหน่งของบิทต่างๆใน IIR

บิทที่ 0 ของ IIR จะเป็นบิทสำหรับตรวจสอบว่ามีการอินเทอร์รัพท์เกิดขึ้นหรือไม่ ถ้าหากบิทนี้ถูกเซ็ตให้เป็น 1 หมายความว่าไม่มีการใช้อินเทอร์รัพท์ สำหรับการสื่อสาร ในกรณีที่ต้องการใช้พอร์ทอนุกรมนี้ในลักษณะของการอินเทอร์รัพท์ เราจะต้องมีการโปรแกรมชิป 8259 อีกขั้นหนึ่ง โดยกำหนดค่าต่างๆของรีจิสเตอร์ใน 8259 ให้อนุญาตให้พอร์ทอนุกรมสามารถร้องขอการอินเทอร์รัพท์ได้ ในกรณีที่เราไม่ได้ทำการเซ็ทค่าของ 8259 เพื่อใช้งานพอร์ทในลักษณะของการอินเทอร์รัพท์ เราอาจสามารถใช้งานในลักษณะกึ่งอินเทอร์รัพท์ได้โดยการเซ็ทค่าต่างๆ ในรีจิสเตอร์ IER ที่จะกล่าวต่อไป และเขียนโปรแกรมทำการตรวจสอบค่าของ IIR เพื่อตรวจสอบการอินเทอร์รัพท์ด้วยตนเองอีกทีหนึ่ง

บิทที่ 1 และ บิทที่ 2 จะเป็นตัวบอกว่าอินเทอร์รัพท์ที่เกิดขึ้นเป็น อินเทอร์รัพท์ประเภทใดซึ่งมีรายละเอียดดังตาราง 6.2

บิทที่ 3 ถึง 7 จะมีค่าเป็น 0 ตลอด และไม่มีความหมายใดๆ

Interrupt ID Register				Interrupt Set and Reset Functions		
Bit 2	Bit 1	Bit 0	Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset Control
0	0	1		None	None	
1	1	0	Highest	Receiver Line Status	Overrun Error or Parity Error or Framing Error or Break Interrupt	Reading the Line Status Register
1	0	0	Second	Received Data Available	Receiver Data Available	Reading the Receiver Buffer Register
0	1	0	Third	Transmitter Holding Register Empty	Transmitter Holding Register Empty	Reading the IIR Register (if source of interrupt) or Writing into the Transmitter Holding Register
0	0	0	Fourth	Modem Status	Clear to Send or Data Set Ready or Ring Indicator or Received Line Signal Detect	Reading the Modem Status Register

ตาราง 6.2 แสดงประเภทของการอินเทอร์รัพท์ที่เกิดขึ้น โดย IIR

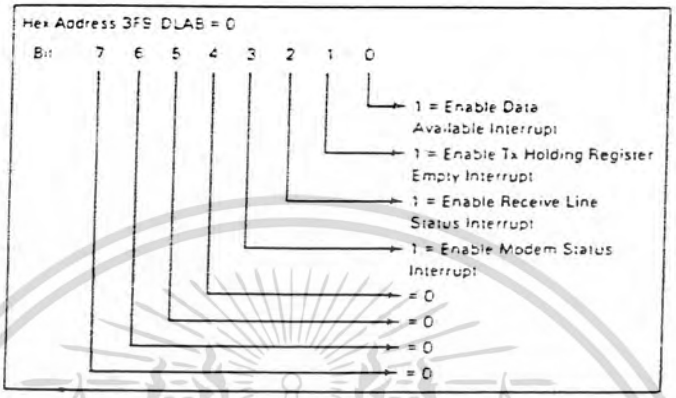
- *Interrupt Enable Register (IER)*

รีจิสเตอร์ตัวนี้มักใช้คู่กันกับ IIR เพราะเป็นรีจิสเตอร์ที่ใช้ในการอินเทอร์รัพท์เหมือนกัน ต่างกันที่ IIR ใช้เป็นรีจิสเตอร์สำหรับตรวจสอบสถานะ แต่ IER เป็นรีจิสเตอร์สำหรับกำหนดสถานะการอินเทอร์รัพท์ต่างๆมีตำแหน่งดังรูป 6.6 รีจิสเตอร์ตัวนี้จะอยู่ที่ Offset ที่ 1 และ DLAB จะต้องเป็น 0

บิทที่ 0 ถ้าเซ็ทให้เป็น 1 จะเป็นการอนุญาตให้มีการอินเทอร์รัพท์ เมื่อมีข้อมูลมารออยู่ในบัฟเฟอร์ข้อมูล

บิทที่ 1 จะเป็นการอนุญาตให้มีการอินเทอร์รัพท์ เมื่อบัฟเฟอร์สำหรับข้อมูลขาออกว่างลง ถ้าเซ็ทให้มีค่าเป็น 1

บิทที่ 2 เมื่อมีค่าเป็น 1 จะเป็นการอนุญาตให้อินเทอร์รัพท์ ถ้ามีการเปลี่ยนแปลงของข้อมูลใน LSR หรือมีการเปลี่ยนแปลงของสถานะของระบบนั่นเอง



รูป 6.6 หน้าที่และตำแหน่งของบิทต่างๆใน IER

บิทที่ 3 เมื่อค่าเป็น 1 จะเป็นการอนุญาตให้มีการอินเทอร์รัพท์เมื่อมีการเปลี่ยนแปลงสถานะของสัญญาณควบคุมโมเด็ม

บิทที่เหลือจะมีค่าเป็น 0 ตลอดเวลา

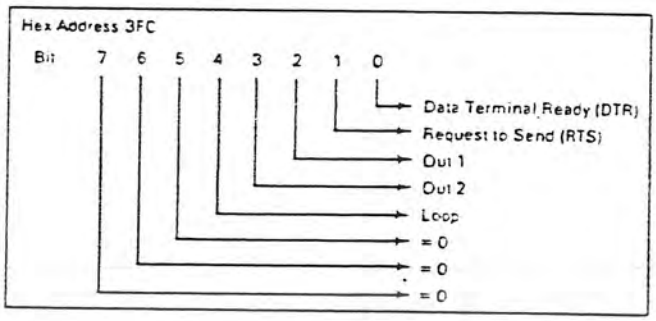
- Modem Control Register (MCR)

MCR เป็นริจิสเตอร์สำหรับให้เราแก้ไขเปลี่ยนแปลงสถานะของสัญญาณควบคุมโมเด็มตามมาตรฐาน EIA-232 เช่น DTR หรือ RTS โดย MCR จะอยู่ที่ตำแหน่ง Offset ที่ 4 จาก Base Address มีตำแหน่งและหน้าที่ต่างๆดังรูป 6.7

บิทที่ 0 ใน MCR จะทำหน้าที่เป็นสัญญาณ DTR ตามมาตรฐาน EIA-232 แต่จะมีสถานะตรงข้ามกับที่เอาท์พุท DTR คือ ถ้าเซ็ทเป็น 1 จะทำให้สัญญาณ DTR เป็น 0 และถ้าเคลียร์ให้เป็น 0 จะทำให้ DTR เป็น 1 ทั้งนี้เพราะ DTR จะแอคทีฟที่ลอจิก 0

บิทที่ 1 ทำหน้าที่เป็น RTS ตามมาตรฐาน EIA-232 จะเป็น Negative Logic คือให้เอาท์พุทตรงข้ามกับค่าที่ป้อนให้เช่นเดียวกับบิท 0

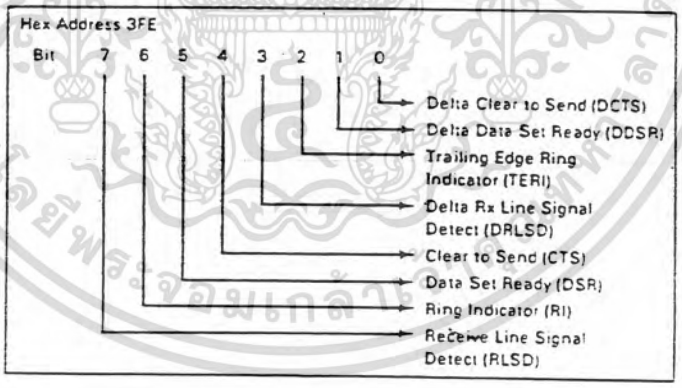
บิทที่เหลือไม่ได้ใช้งานใน ไอบีเอ็มพีซี



รูป 6.7 หน้าที่และตำแหน่งของบิทต่างๆใน MCR

- *Modem Status Register*

เป็นรีจิสเตอร์ที่แสดงสถานะต่างๆของสัญญาณ EIA-232 โดยจะอยู่ที่ Offset ที่ 6 จาก Base Address และจะมีหน้าที่และตำแหน่งของบิทต่างๆเป็นดังรูป 6.8



รูป 6.8 หน้าที่และตำแหน่งของบิทต่างๆใน MSR

- *Receiver Buffer Register และ Transmitter Holding Register*

รีจิสเตอร์สองตัวนี้เป็นรีจิสเตอร์ที่ใช้เก็บข้อมูลที่ได้รับจากสายส่ง และเป็นที่เก็บข้อมูลที่ต้องการส่งก่อนที่จะทำการส่ง RBR และ THR จะอยู่ที่ตำแหน่ง Base Address และจะเป็น RBR และ

THR ก็ต่อเมื่อ DLAB มีค่าเป็น 0 เท่านั้น ข้อแตกต่างของ RBR และ THR ก็คือเมื่ออ่านจะเป็นการอ่านจาก RBR และเมื่อเขียนจะเป็นการเขียนลง THR

6.4 การใช้งานพอร์ทผ่านทาง BIOS

ในการใช้งานพอร์ทอนุกรมนี้ เราสามารถทำได้หลายทาง วิธีแรกก็คือเขียนโปรแกรมเพื่อควบคุมการทำงานที่ชิป 8250 โดยการโปรแกรมค่าให้แก่รีจิสเตอร์ต่างๆภายในชิปโดยตรง วิธีนี้จะยุ่งยากมาก แต่จะมีความเร็วสูง เพราะเป็นการเข้าถึงระดับฮาร์ดแวร์โดยตรง อีกวิธีหนึ่งที่อาจจะช้าไปบ้างแต่ก็สะดวกมากกว่าก็คือการเรียกใช้บริการของ BIOS ผ่านซอฟท์แวร์อินเทอร์รัพต์หมายเลข 14h

อินเทอร์รัพต์หมายเลขนี้มีฟังก์ชันให้ใช้งานที่จำเป็นไว้อย่างครบครัน การใช้งานเพียงแต่โหลดค่าฟังก์ชันที่ต้องการไว้ในรีจิสเตอร์ AH และหมายเลขพอร์ทไว้ใน DX โดยหมายเลขพอร์ทนั้นจะมีค่าน้อยกว่าหมายเลขพอร์ทที่แท้จริงอยู่ 1 เช่นหมายเลขพอร์ทของ COM1 จะเป็น 0 เป็นต้น แล้วทำการเรียกอินเตอร์รันท์หมายเลข 14h บางครั้งอาจจะต้องมีการโหลดข้อมูลเพิ่มเติมไว้ใน AL รายละเอียดของฟังก์ชันต่างๆ ศึกษาได้เพิ่มเติมจากหนังสือ Advance MSDOS

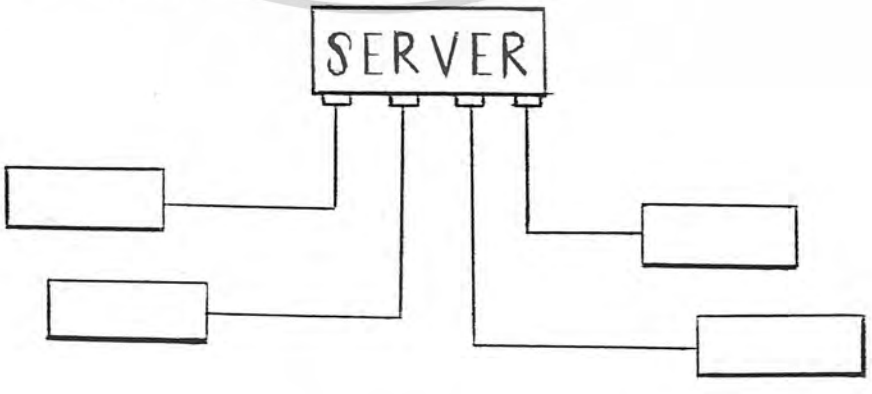
บทที่ 7

PC-Network System

โครงงาน PC-Network เป็นรูปแบบหนึ่งของระบบเครือข่ายไมโครคอมพิวเตอร์ราคาถูก โดยระบบนี้เป็นการเชื่อมต่อเครื่องไมโครคอมพิวเตอร์เข้าด้วยกัน ผ่านทางพอร์ทอนุกรมเป็นระบบเครือข่ายเล็กๆ ตั้งแต่ 2-5 เครื่อง เพื่อให้สามารถสื่อสารข้อมูล,ใช้ข้อมูล หรืออุปกรณ์ต่อพ่วงภายในเครือข่ายร่วมกัน ซึ่งถือว่าเป็นระบบที่เลียนแบบระบบ LAN (Local Area Network) แต่มีขนาดเล็กกว่าก็ย่อมได้ ดังนั้นความสามารถหรือประสิทธิภาพย่อมจะไม่อาจเปรียบเทียบกับระบบ LAN ในท้องตลาดได้ เพราะถือว่าเป็นคนละลักษณะกัน ระบบ PC-Network จะเป็นเครือข่ายสำหรับหน่วยงานเล็กๆที่มีความต้องการในการใช้ข้อมูลหรืออุปกรณ์บางอย่างร่วมกัน ระหว่างเครื่องไมโครคอมพิวเตอร์ไม่กี่เครื่อง และปริมาณงานก็ไม่มากนัก หากจะติดตั้งระบบ LAN ก็จะมีราคาสูงมาก ซึ่งไม่คุ้มค่าต่อการลงทุน การใช้ระบบเครือข่ายของ PC-Network แทน นั้นไม่ต้องเพิ่มเติมฮาร์ดแวร์พิเศษมากมาย ต้นทุนต่ำ จึงดูจะเป็นทางเลือกที่เหมาะสมกว่ามาก

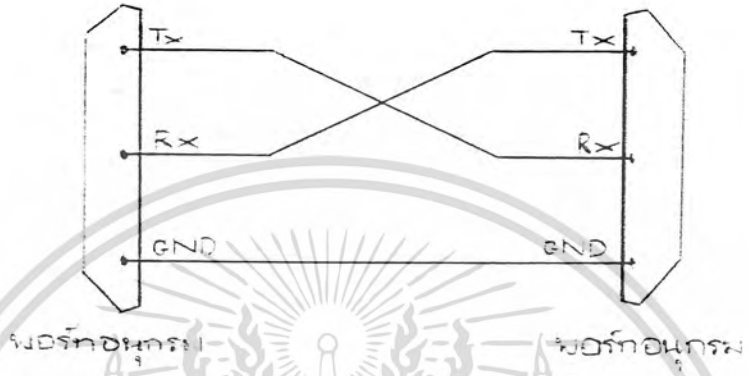
7.1 ลักษณะของเครือข่าย PC-Network

เครื่องไมโครคอมพิวเตอร์ที่จะเชื่อมโยงเข้าเป็นเครือข่ายของ PC-Network จะต้องมีพอร์ทอนุกรมอย่างน้อย 1 พอร์ท โดยจะแบ่งเครื่องภายในเครือข่ายเป็น 2 ประเภท คือ เครื่องที่ทำหน้าที่เป็น server และเครื่องที่เป็น Terminal โดยเครื่อง server จะเป็นตัวกลางในการสื่อสารข้อมูล หากมีพอร์ทอนุกรมมากเท่าใดก็สามารถต่อเชื่อมเครื่องลูกได้มากเท่านั้น เพราะใช้ Topology ในการเชื่อมต่อแบบดาว ดังนั้น หากเครื่อง server มีพอร์ทอนุกรมแบบ 4 พอร์ท ก็จะมีการเชื่อมต่อเครือข่ายดังรูป 7.1



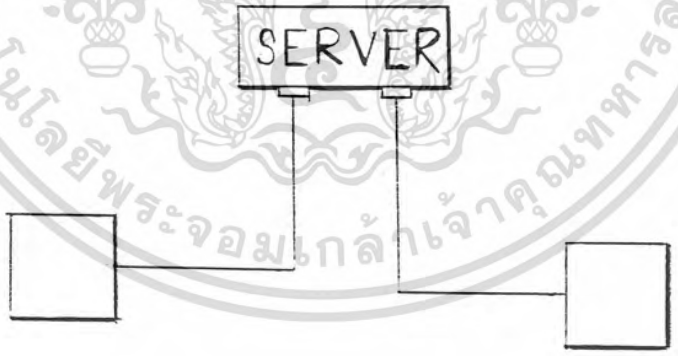
รูป 7.1 รูปแบบในการเชื่อมโยงเครือข่าย

สายสื่อสารที่ใช้ในการเชื่อมต่อระหว่างเครื่อง server และเครื่อง Terminal จะใช้สายโทรศัพท์ หรือสายไฟธรรมดาก็ได้ จะมีขาสัญญาณที่จะต้องใช้อยู่ 3 ขาคือ Tx, Rx และ กราวด์ ส่วนขาสัญญาณที่เหลือของพอร์ทอนุกรมเราจะไม่นำมาใช้งาน การต่อสายสัญญาณระหว่างแต่ละ node จะมีลักษณะดังรูป 7.2



รูป 7.2 การเชื่อมต่อสายสัญญาณระหว่าง node

ส่วนเครื่องไมโครคอมพิวเตอร์ที่มีพอร์ทอนุกรมเพียง 2 พอร์ทคือ COM1, COM2 ก็จะได้เครื่อง server ที่สามารถต่อเครื่องลูก หรือเครื่อง Terminal ได้ 2 เครื่องดังรูป 7.3



รูป 7.3 การเชื่อมโยงโดย server มีพอร์ทอนุกรม 2 พอร์ท

7.2 โปรแกรมควบคุมการสื่อสารข้อมูล

หัวใจสำคัญของระบบ PC-Network ก็คือโปรแกรมสำหรับควบคุมและจัดการการสื่อสารข้อมูลระหว่างเครื่องคอมพิวเตอร์ภายใน Network ให้สามารถสื่อสารกันได้อย่างถูกต้อง โดยระบบควบ

คุมนี้จะต้องทำตัวเสมือนระบบปฏิบัติการของ Network ที่คลุมอยู่บนระบบปฏิบัติการ MS-DOS อีกทีหนึ่ง คอยควบคุมดูแล และให้บริการสำหรับการทำงานที่เกี่ยวข้องกับ Network ต่างๆ

การเขียนโปรแกรมสำหรับควบคุมการสื่อสารข้อมูลโดยผ่านทางพอร์ทอนุกรม มีวิธีการที่นิยมใช้กันโดยทั่วไปอยู่ 2 วิธีคือการใช้บริการของ BIOS โดยใช้งานและควบคุมพอร์ทอนุกรมผ่านทาง int.14h ซึ่งมีฟังก์ชันย่อยสำหรับการ initialize port การส่ง, รับ character ผ่านพอร์ทอนุกรมสามารถเรียกใช้งานได้โดยง่าย อย่างไรก็ตามการเขียนโปรแกรมควบคุมการสื่อสารข้อมูลโดยใช้บริการ int. 14h ก็มีจุดอ่อนตรงที่ว่าไม่สามารถทำการรับส่งข้อมูลด้วยอัตราเร็วสูงได้ การจัดการกับข้อมูลที่ไหลเข้าออกผ่านทางพอร์ทอนุกรมก็ช้าเกินกว่า จะทำเป็นระบบ Network ที่มีประสิทธิภาพได้ ดังนั้น PC-Network Project จึงเลือกใช้วิธีการที่ 2 คือการเข้าไปเขียนโปรแกรมควบคุมฮาร์ดแวร์ ของพอร์ทอนุกรมโดยตรง และจัดการกับ interrupt service routine ของพอร์ทอนุกรมเอง ทำให้สามารถเขียนโปรแกรมควบคุมการสื่อสารข้อมูลให้มีอัตราการรับส่งข้อมูลด้วยอัตราเร็วสูงถึง 115,200 bps. ซึ่งเป็นความเร็วสูงที่สุดเท่าที่ฮาร์ดแวร์ของพอร์ทอนุกรมจะสามารถทำได้ (อัตราเร็วที่ int. 14h สามารถทำได้คือ 9,600 bps.)

การทำงานของ PC-Network

การทำงานของเครื่องไมโครคอมพิวเตอร์ภายในระบบเครือข่าย สามารถทำงานเดิมของตนเองได้ตามปกติทุกอย่างรวมทั้งเครื่องที่ทำหน้าที่เป็น server ด้วย ส่วนการใช้งานในลักษณะของเครือข่าย ความสามารถหลักๆก็มีดังต่อไปนี้

- *Network Send Message*

เป็นการส่งข้อความสื่อสารตอบโต้กัน ระหว่างเครื่องคอมพิวเตอร์ภายในเครือข่าย โดยสามารถส่งไปให้เครื่องใดๆภายในเครือข่ายก็ได้ ซึ่งความสามารถนี้มีทั้งแบบส่งข้อความโต้ตอบกันทีละบรรทัดที่ DOS Prompt และแบบ pop-up full screen สำหรับการส่งข้อความถึงกันเป็นจำนวนมาก ซึ่งการใช้งานจะได้กล่าวในหัวข้อถัดไป

- *Network Copy Files*

เป็นการโอนย้ายข้อมูลระหว่างเครื่องไมโครคอมพิวเตอร์ภายในเครือข่าย ผ่านสายสื่อสารข้อมูล ซึ่งความสามารถนี้ทำได้ทั้งส่งไฟล์ไปยังเครื่องอื่น หรือขอ copy ไฟล์จากเครื่องอื่น สามารถใช้งานได้ทั้งแบบ command line ที่ DOS Prompt หรือแบบ Full Screen Directory ก็ได้

- *Network Run Files*

เป็นการนำเอาโปรแกรมจากเครื่องอื่นมา run ที่เครื่องของตัวเอง ซึ่งจะเป็นลักษณะ

"load and run" ดังเช่นระบบ LAN ของ Novell NetWare ซึ่งจะต่างจากการ copy ไฟล์มาแล้วค่อยสั่ง run คือเป็นความสามารถในการใช้งานฮาร์ดดิสค์ร่วมกันภายในระบบเครือข่าย

- *Network Remote Access*

เป็นการเข้าไปควบคุมการทำงานของเครื่องอื่นภายในเครือข่าย จากเครื่องที่เรากำลังทำงานอยู่ ซึ่งจะเป็นการควบคุมการใช้งานระยะไกล ต่างไปจากการ "load and run" มาก

- *Network Share Printer*

เป็นการใช้งานเครื่องพิมพ์ร่วมกันระหว่างเครื่องไมโครคอมพิวเตอร์ภายในระบบเครือข่าย

7.3 ความสามารถ และการใช้งาน PC-Network

ระบบ PC-Network ประกอบไปด้วยโปรแกรมควบคุมและโปรแกรมอรรถประโยชน์มากมายหลายโปรแกรม โปรแกรมควบคุมที่ถือเป็น Network-OS ของระบบ PC-Network ก็คือโปรแกรม SERVER ซึ่งเป็น resident program จะต้อง run ทิ้งไว้ที่เครื่อง server ซึ่งการใช้งานในครั้งแรก เมื่อเริ่มมีการติดตั้งระบบ PC-Network จากการ run โปรแกรม SERVER ที่เครื่อง server จะเป็นดังรูป 7.4

```
F:\USER\JUB\TEST >server1

        The PC-Network Server Manager
            Version 1.00 (3 user)

            Written by  C.Teekayu

        Copyright (c) 1992  PC-Network Project.

F:\USER\JUB\TEST >
```

รูป 7.4 การติดตั้ง server

ส่วนที่เครื่องลูกทั้งหมดในเครือข่าย ก็จะมี resident program อีกตัวหนึ่ง ทำหน้าที่เป็นผู้จัดการในการติดต่อกับ server และเครื่องอื่นๆภายในเครือข่าย คือโปรแกรม NLOGIN (Netwrok Login) ซึ่งการใช้งานจะต้องแจ้งพอร์ทอนุกรมที่ใช้และชื่อ user และต้องติดตั้งหลังจากที่เครื่อง server ติดตั้งโปรแกรม SERVER แล้วเท่านั้น ซึ่งถ้ามีการเรียกใช้งานผิดพลาดจะมีการแจ้งเตือนดังรูป 7.5

```
F:\USER\JUB\TEST >nlogin

 PCNet V1.00 - Workstation Shell for MS-DOS User login
 (C) Copyright 1992 PC-Network Project. All Rights Reserved.

Enter your COM port: com1
Enter your login name: teekayu

     PCNet SERVER not found,check and try again!

F:\USER\JUB\TEST >
```

รูป 7.5 การ Login ในขณะที่ server ยังไม่พร้อม

ซึ่งถ้ามีการติดตั้งได้ถูกต้อง ระบบจะแจ้งให้ทราบดังรูป 7.6

```
F:\USER\JUB\TEST >nlogin

 PCNet V1.00 - Workstation Shell for MS-DOS User login
 (C) Copyright 1992 PC-Network Project. All Rights Reserved.

Enter your COM port: com1
Enter your login name: teekayu

     Attached to PCNet SERVER O.K.

F:\USER\JUB\TEST >
```

รูป 7.6 Login พร้อมที่จะใช้งาน

เมื่อทำการติดตั้งระบบ PC-Network ภายในเครือข่ายเรียบร้อยแล้ว จากนั้นก็จะสามารถใช้งานเครื่องคอมพิวเตอร์ได้ตามปกติ และสามารถเรียกใช้งานความสามารถต่างๆของระบบเครือข่ายได้ทันที

การส่งข้อความสื่อสารกันภายในเครือข่าย

สามารถทำได้โดยใช้คำสั่ง NSEND ตามด้วยหมายเลข station และข้อความที่ต้องการจะส่ง ซึ่งถ้าใส่พารามิเตอร์ผิด ระบบจะแจ้งวิธีการที่ถูกต้องดังรูป 7.7

```
F:\USER\JUB\TEST >nsend

PCNet V1.00 - Send Message to Workstation
(C) Copyright 1992 PC-Network Project. All Rights Reserved.

Usage:
       NSEND  <station>,"message"

    Where <station> : 0 = ALL
                      1 = USER#1
                      2 = USER#2

F:\USER\JUB\TEST >
```

รูป 7.7 Help ของ NSEND

เมื่อป้อนพารามิเตอร์ถูกต้องแล้ว ระบบก็จะทำการส่งข้อความไปให้ station ที่กำหนด พร้อมทั้งแจ้งผลการส่งว่าสำเร็จหรือไม่ดังรูป 7.8

```
F:\USER\JUB\TEST >nsend 0 "Hello,how are you,today?"

PCNet V1.00 - Send Message to Workstation
(C) Copyright 1992 PC-Network Project. All Rights Reserved.

      Message sent to Workstation 1
      Message NOT sent to Workstation 2

F:\USER\JUB\TEST >
```

รูป 7.8 ผลการใช้ NSEND

ซึ่งข้อความที่ส่งไปนี้จะไปปรากฏที่บรรทัดล่างสุดของจอ station ที่กำหนด ซึ่งเขาสามารถส่งข้อความตอบกลับมาได้ ข้อความก็จะมาปรากฏที่บรรทัดล่างสุดของหน้าจอของเราเช่นกันดังรูป 7.9

```
F:\USER\JUB\TEST >nsend 0 "Hello,how are you,today?"

 PCNet V1.00 - Send Message to Workstation
 (C) Copyright 1992 PC-Network Project. All Rights Reserved.

       Message sent to Workstation 1
       Message NOT sent to Workstation 2

F:\USER\JUB\TEST >
```

```
•>> From [1] : I'm fine, thank you.
```

รูป 7.9

การโอนย้ายไฟล์ข้อมูล

การแลกเปลี่ยนข้อมูลระหว่างเครื่องคอมพิวเตอร์ภายในเครือข่าย สามารถกระทำได้ด้วยคำสั่ง NCOPY ซึ่งมีการใช้งานดังรูป 7.10

```
F:\USER\JUB\TEST >ncopy

 PCNet V1.00 - Files Transfer from/to Workstation
 (C) Copyright 1992 PC-Network Project. All Rights Reserved.

Usage:
       NCOPY  filename  FROM/TO  station

    Where <station> : 1 = USER#1
                      2 = USER#2

F:\USER\JUB\TEST >
```

รูป 7.10 Help ของ NCOPY

การ run โปรแกรมของเครื่อง Terminal อื่นในระบบ

การ load and run สามารถกระทำได้ด้วยคำสั่ง NRUN ซึ่งจะมีวิธีการใช้ดังนี้ คือ NRUN [port,_ filename ระบบจะทำการ load โปรแกรมจากเครื่องอื่นมา run ตามต้องการ

โปรแกรมอรรถประโยชน์

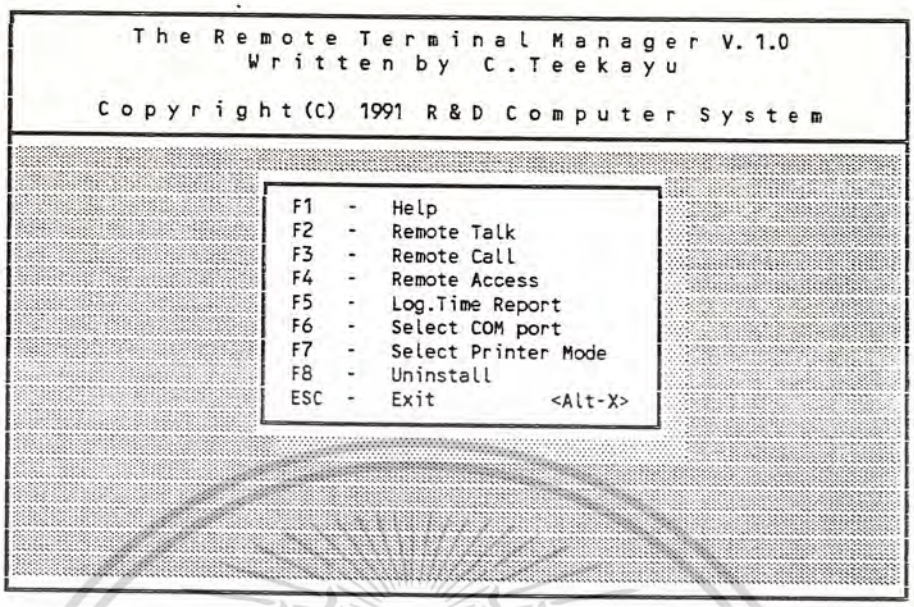
ระบบ PC-Network มีโปรแกรมที่ช่วยเสริมความสามารถของระบบเครือข่ายอีกหลายตัว ที่สำคัญๆก็คือ The Remote Terminal Manager และ File Transfer Manager ดังจะได้กล่าวถึงรายละเอียดและวิธีใช้งานดังต่อไปนี้

The Remote Terminal Manager (RTM)

เป็นโปรแกรม resident อีกตัวหนึ่งที่ทำหน้าที่เป็นทั้งตัวส่งและตัวรับ การใช้งานจะต้อง run ทิ้งไว้ทั้ง 2 เครื่องที่ต้องการติดต่อกัน การเริ่มใช้งานครั้งแรกโปรแกรมจะแสดงดังรูป 7.11

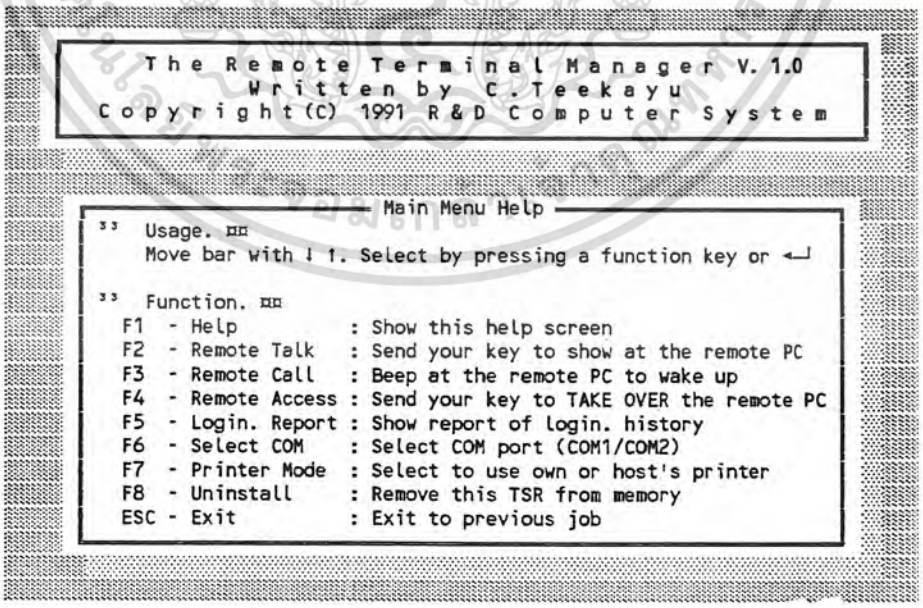
รูป 7.11 การติดตั้ง The Remote Terminal Manager

เมื่อต้องการใช้งานไม่ว่าจะทำงานอะไรอยู่ก็ตาม สามารถเรียกใช้งานโปรแกรม RTM ได้ โดยการกด Hot-Key คือ Alt-Backspace โปรแกรม RTM ก็จะ pop-up ขึ้นมาให้ใช้งานดังรูป 7.12



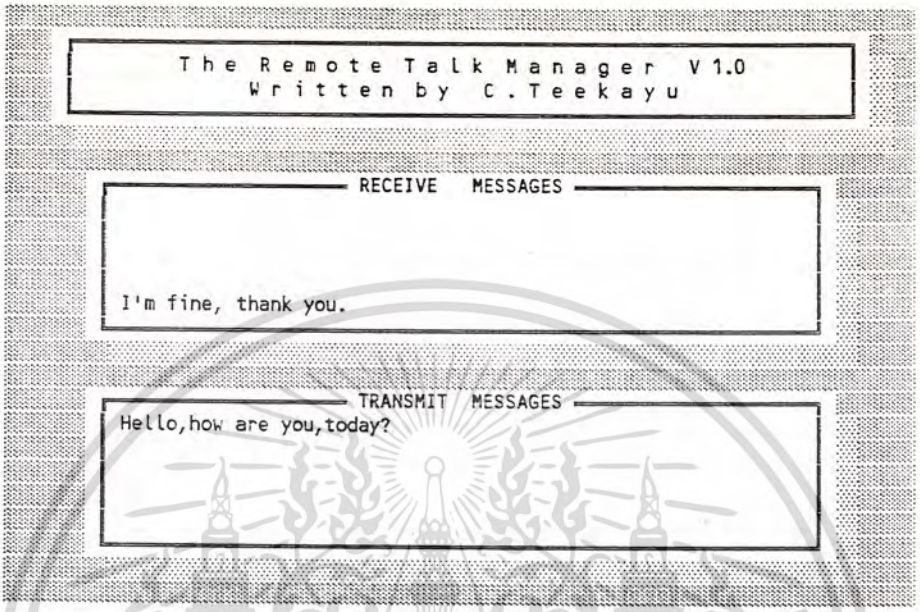
รูป 7.12 pop-up main menu ของ RTM

ซึ่งการใช้งานที่ main menu ของโปรแกรม RTM สามารถกระทำได้ทั้งแบบใช้ลูกศรเลื่อนแถบสว่างขึ้นลง เพื่อเลือกการทำงาน หรือจะใช้วิธีกด Hot-Key ตามที่แจ้งไว้บนหน้าจอก็ได้ การเลือกขอดูความช่วยเหลือ จะได้ดังรูป 7.13



รูป 7.13 Help ของ RTM

ส่วนการใช้งานในการส่งข้อความโต้ตอบกันนั้น RTM จะสามารถทำได้ดีกว่าคำสั่ง NSEND โดย RTM จะมีการสื่อสารกันแบบ Full Screen ซึ่งมีประสิทธิภาพมากกว่าดังแสดงในรูป 7.14



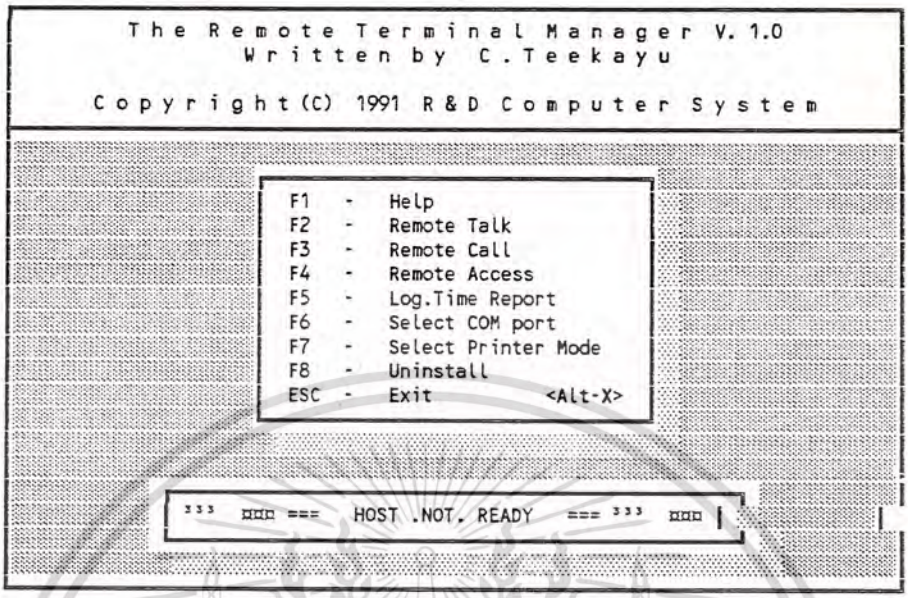
รูป 7.14 TALK ของ RTM

เมื่อพิมพ์ข้อความหรือรับข้อความจนเต็มหน้าต่าง หน้าต่างข้อความก็จะถูกเลื่อนขึ้นทีละบรรทัด ซึ่งทำให้เราพิมพ์ข้อความคุยกันได้เป็นเรื่องเป็นราวมากกว่าคำสั่ง NSEND ซึ่งส่งข้อความได้ทีละ 1 บรรทัดเท่านั้น

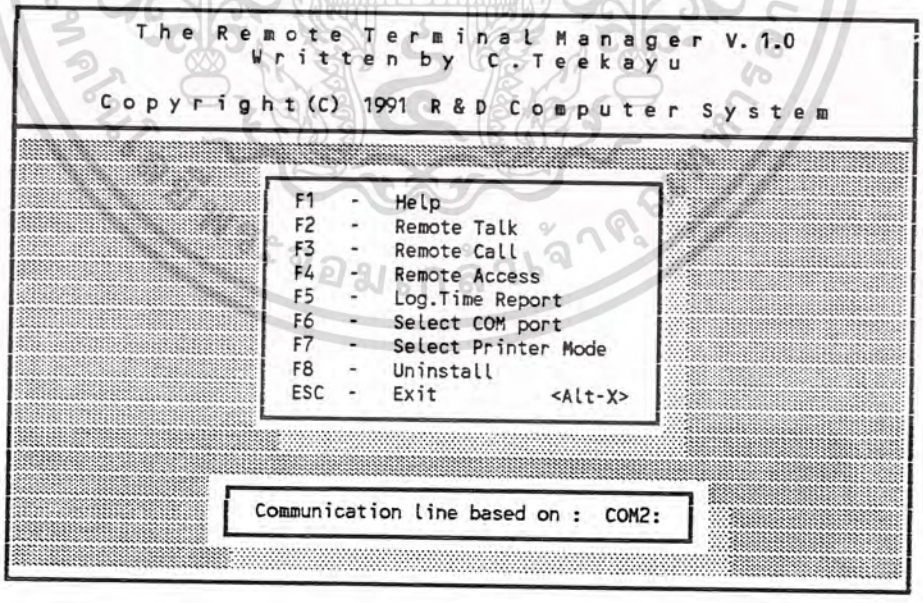
การเลือกใช้งานฟังก์ชันต่างๆที่กำหนดไว้ใน main menu ถ้าหากฝ่ายตรงข้ามไม่พร้อมที่จะทำการสื่อสารด้วย โปรแกรม RTM จะแจ้งข้อความเตือนดังรูป 7.15

โปรแกรม RTM สามารถเปลี่ยนพอร์ทอนุกรมที่ใช้งานได้ด้วย โดยเมื่อเลือกคำสั่งเปลี่ยนพอร์ท โปรแกรมจะถามพอร์ทที่ต้องการใช้แบบ toggle select ดังรูป 7.16

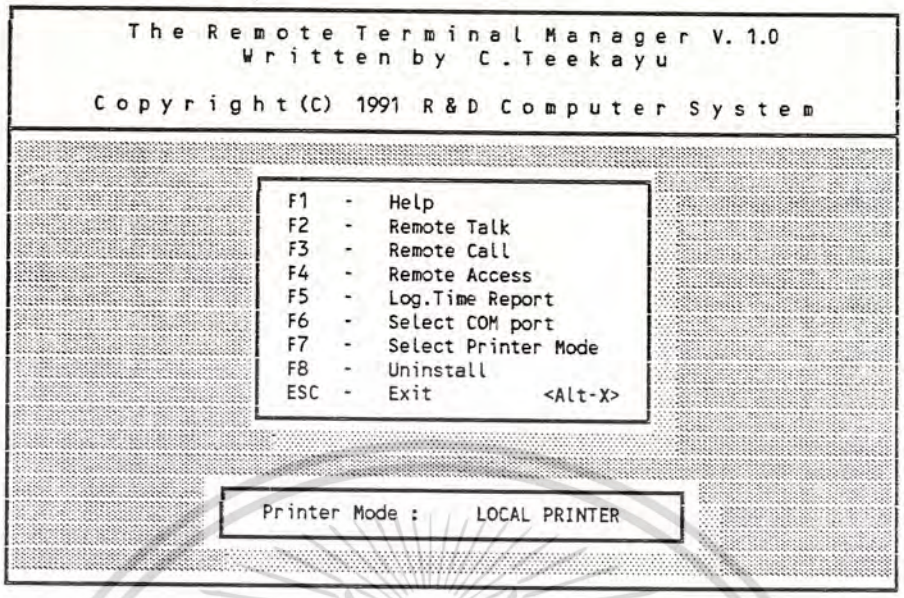
การใช้งานเครื่องพิมพ์สามารถเลือกได้ 2 แบบคือ Local Printer และ Remote Printer ดังรูป 7.17 และ 7.18 โดย Local Printer จะหมายถึงการใช้งาน Printer ของเครื่องตัวเอง ส่วน Remote Printer เป็นการใช้งาน Printer ของเครื่องอื่น ดังนั้นต่อไปไม่ว่าจะเกิดการพิมพ์จากโปรแกรม application ใดๆ งานพิมพ์นั้นก็จะถูกส่งไปพิมพ์ที่เครื่องพิมพ์ของเครื่องอื่นทันที โดยมองเสมือนว่าเป็นเครื่องพิมพ์ของตัวเอง



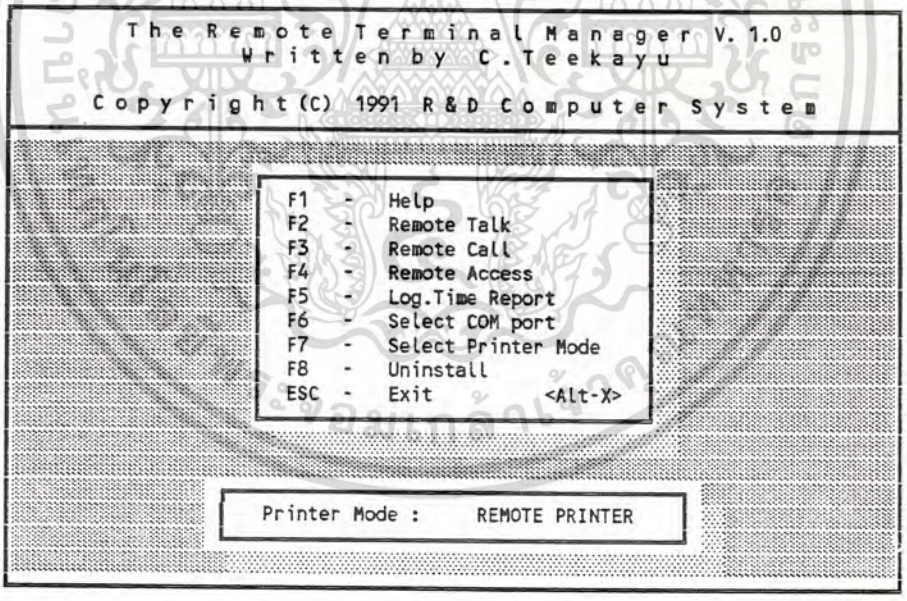
รูป 7.15 ระบบจะเตือนเมื่อคลื่อสารไม่พร้อม



รูป 7.16 การเลือกพอร์ทสื่อสาร



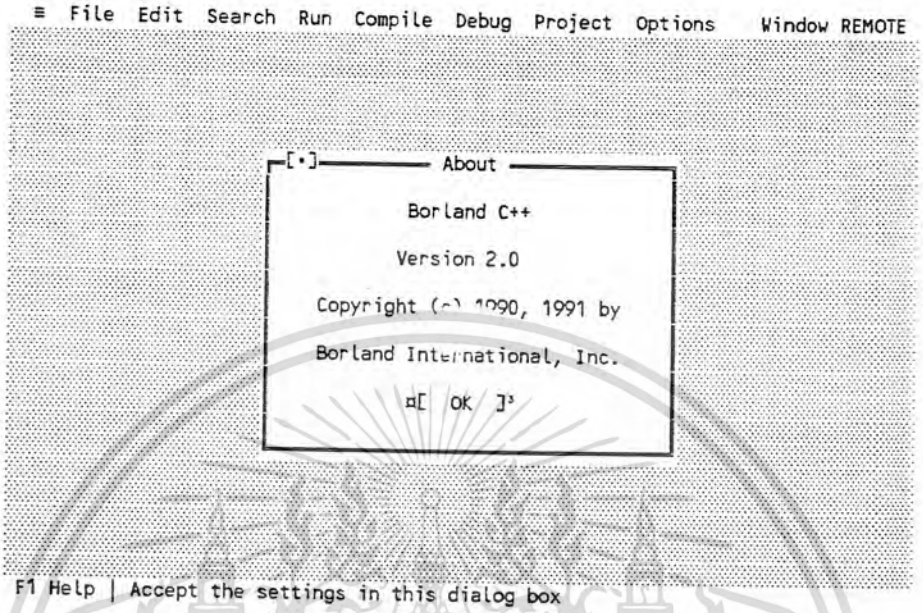
รูป 7.17 Local Printer Mode



รูป 7.18 Remote Printer Mode

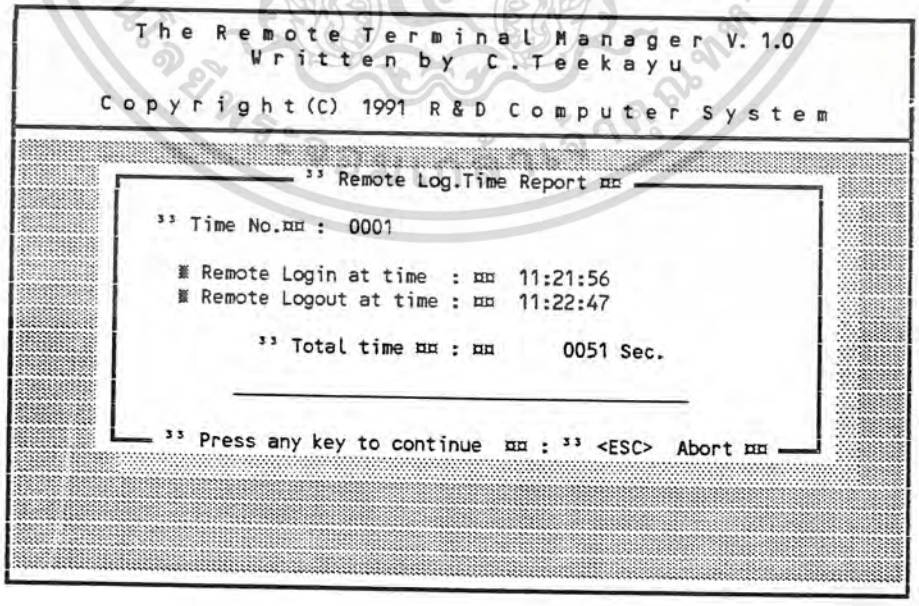
การทำ remote access สามารถกระทำได้ โดยเราจะสามารถควบคุมเครื่องอื่นได้จากคีย์บอร์ดของเครื่องที่เราทำงานอยู่ และผลการทำงานจะถูกส่งมายังหน้าจอของเราด้วย โดยมีคำ

ว่า REMOTE กระพริบอยู่ที่มุมขวาบนสุดของหน้าจอเรา ดังแสดงการทำ Remote Access ไป run โปรแกรม Borland C^{++} ดังรูป 7.19



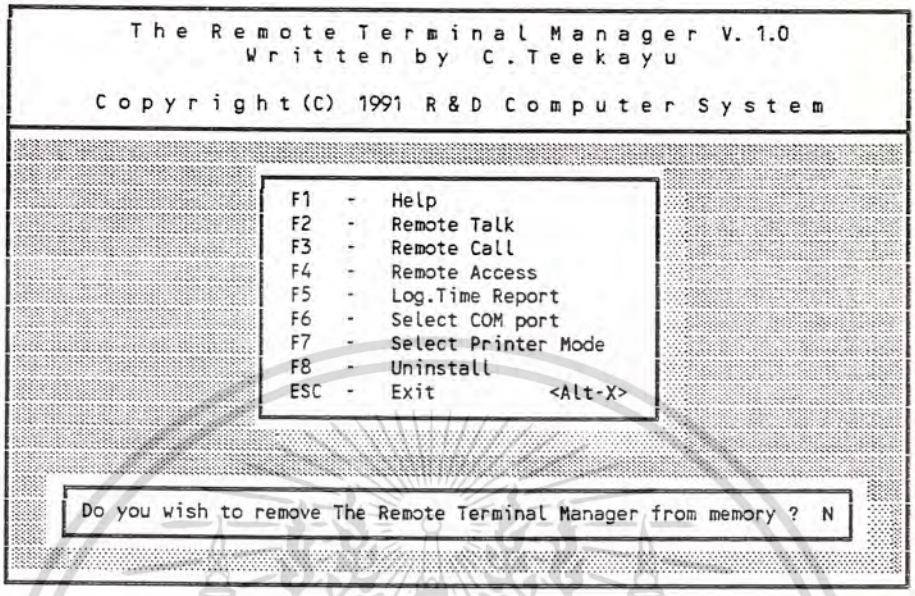
รูป 7.19 การ run Borland C^{++} แบบ Remote

และการทำ Remote Access ยังมีการบันทึกสถิติการถูกควบคุมระยะไกลเก็บไว้เป็น history ให้เรียกดูได้ดังรูป 7.20



รูป 7.20 history ของการถูกควบคุมระยะไกล

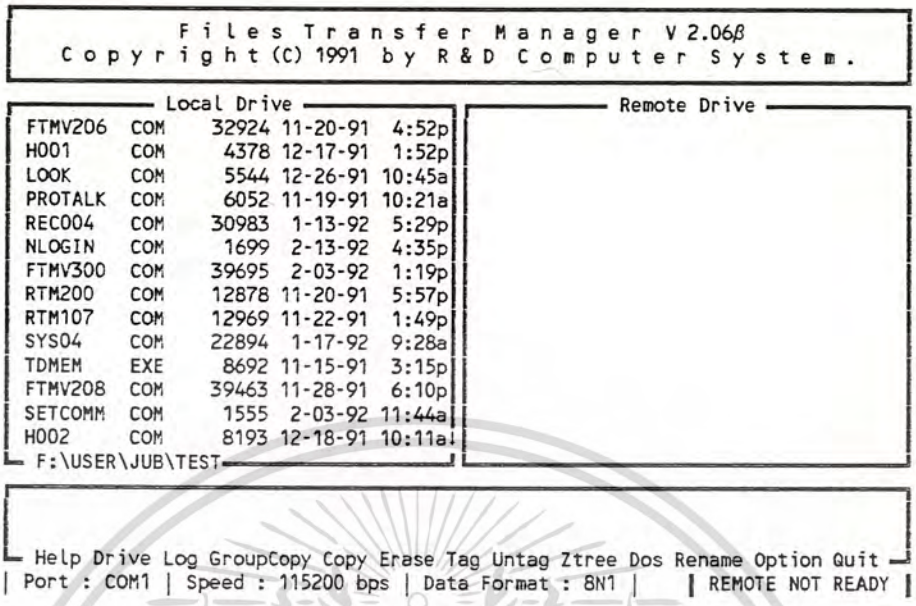
นอกจากนี้เมื่อต้องการเลิกใช้งานโปรแกรม RTM ก็สามารถเลือกคำสั่ง Uninstall ดังรูป 7.21 เพื่อยกเลิกโปรแกรมที่ resisdent อยู่คืน memory ให้กับระบบได้อีกด้วย



รูป 7.21 การเลือก Uninstall

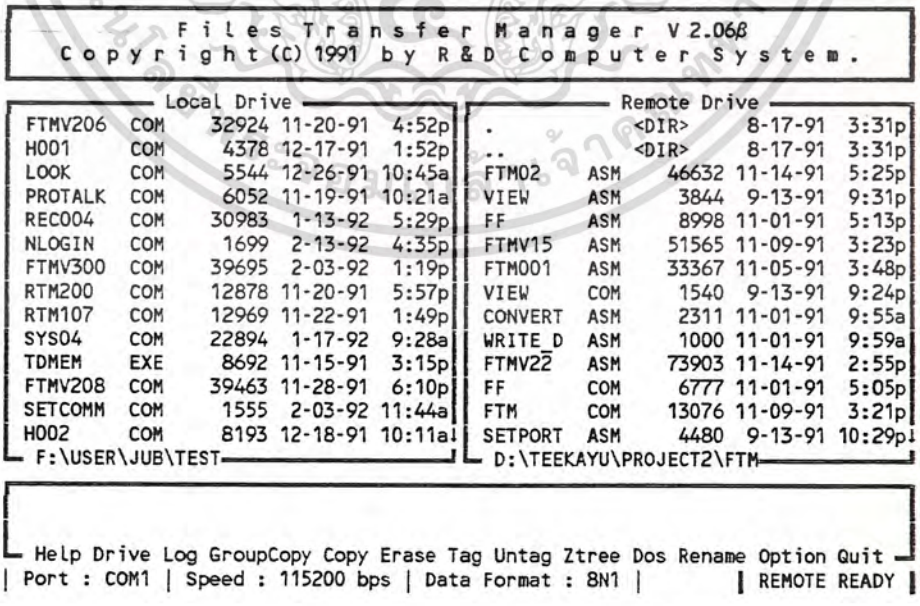
Files Transfer Manager (FTM)

เป็นโปรแกรมควบคุมการโอนย้ายไฟล์ข้อมูลและจัดการไฟล์ แบบ Full Screen Directory ซึ่งจะมีความสามารถและประสิทธิภาพสูงกว่าคำสั่ง NCOPY มาก เมื่อต้องการใช้งานก็เรียกโปรแกรม FTM ซึ่งจะปรากฏผลบนจอดังรูป 7.22 โดยถ้าเครื่องที่ต้องการติดต่อด้วยยังไม่ได้ run โปรแกรม FTM หน้าจอจะมีเฉพาะรายชื่อไฟล์ในหน้าต่าง Local Drive ซึ่งเป็นรายชื่อไฟล์ในเครื่องของตัวเองเท่านั้น ส่วนที่ช่อง Remote Drive จะยังคงว่างเปล่า พร้อมกับที่มุมซ้ายล่างสุดจะมีข้อความแจ้งว่า "Remote Not Ready"



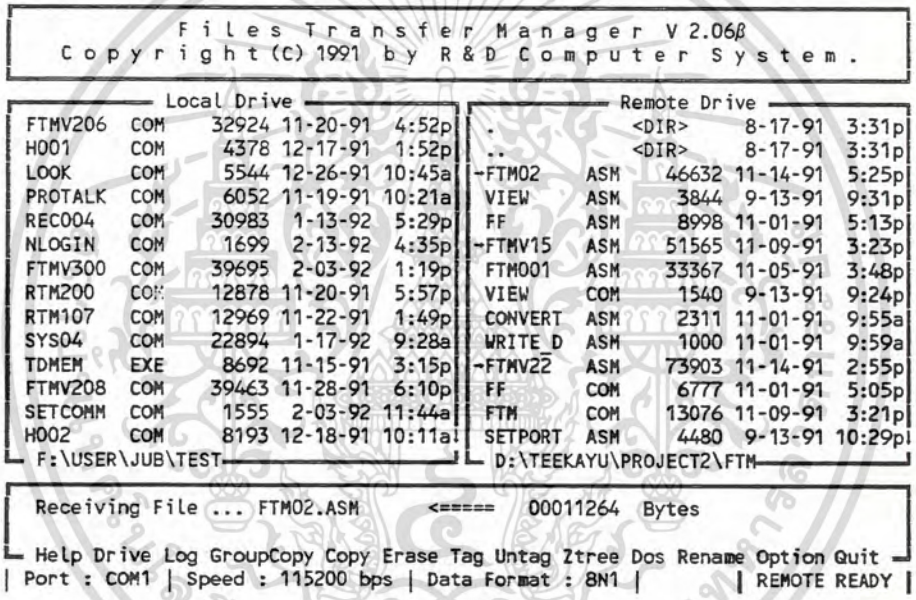
รูป 7.22 ขณะเริ่มใช้งาน FTM

เมื่อฝ่ายตรงข้าม run FTM พร้อมที่จะสื่อสารข้อมูล โปรแกรมก็จะทราบเองโดยอัตโนมัติ โดยส่งรายชื่อไฟล์ของเราไปให้ฝ่ายตรงข้าม และรับรายชื่อไฟล์ของฝ่ายตรงข้ามมาแสดงที่หน้าจอในช่อง Remote Drive ดังรูป 7.23

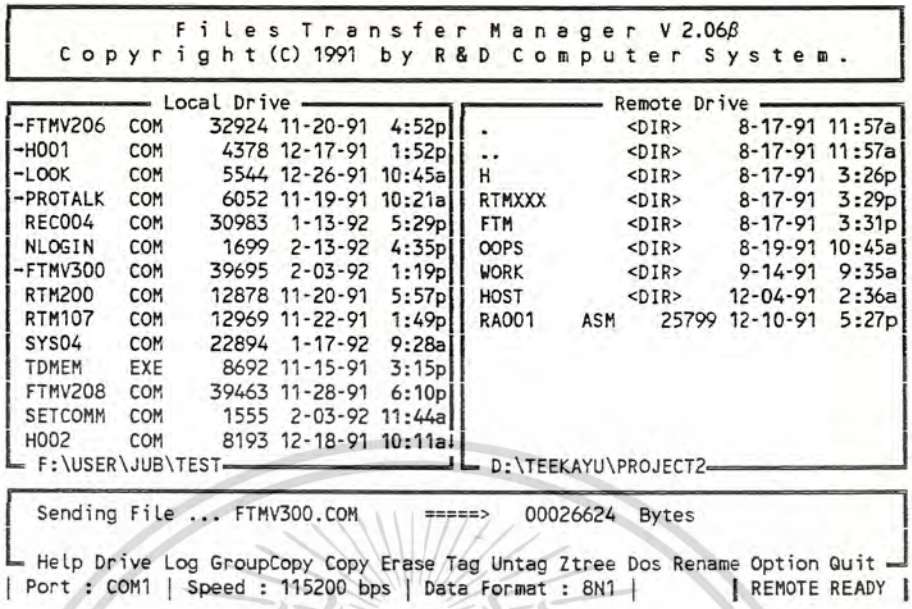


รูป 7.23 FTM เมื่อพร้อมที่จะใช้งาน

จากนั้นก็จะสามารถใช้งานฟังก์ชันต่างๆได้ ไม่ว่าจะ copy ไฟล์ไป-มา ,ลบไฟล์ หรือเปลี่ยน directory โดยสามารถจัดการได้ทั้ง Local Drive และ Remote Drive สามารถใช้คีย์ลูกศรเลื่อนแถบสว่างเพื่อเลือกไฟล์ และเลือกการทำงานต่างๆได้โดยสะดวก ซึ่งความสามารถและลักษณะการใช้งานจะคล้ายคลึงกับโปรแกรม LAPLINK ที่มีชื่อเสียงของต่างประเทศนั่นเอง แต่การทำงานต่างๆเช่นการเลื่อนแถบสว่าง การเลือกไฟล์ การเปลี่ยน directory โปรแกรม FTM จะสามารถทำได้เร็วกว่าโปรแกรม LAPLINK มาก การ copy ไฟล์จาก Remote Drive มายัง Local Drive จะแสดงผลดังรูป 7.24 และการ copy ไฟล์ จาก Local Drive ไปยัง Remote Drive แสดงผลดังรูป 7.25

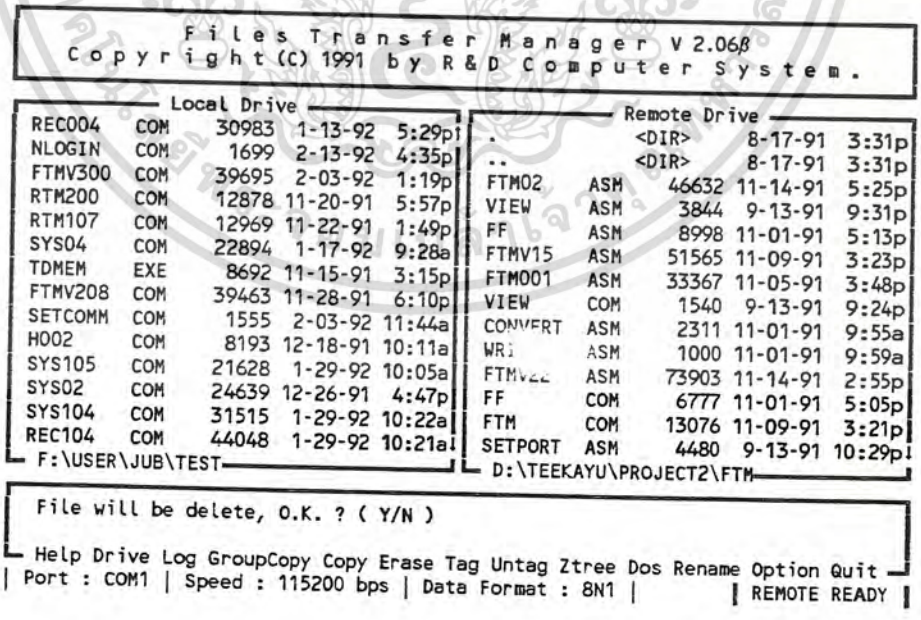


รูป 7.24 การรับไฟล์



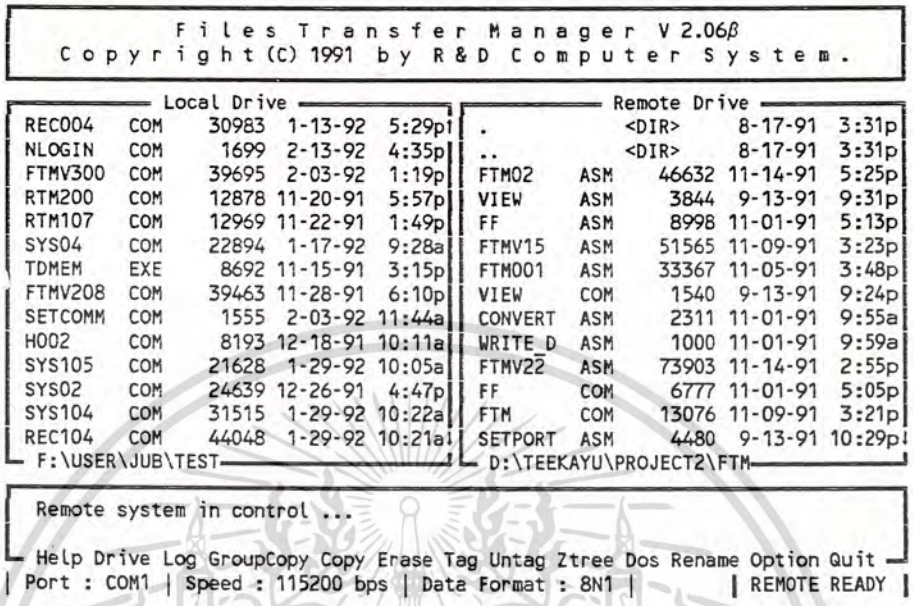
รูป 7.25 การส่งไฟล์

การจัดการกับไฟล์เช่น ลบไฟล์ สามารถทำได้ทั้ง Local Drive และ Remote Drive โดยโปรแกรมจะถามให้มีการ confirm ก่อนดังรูป 7.26



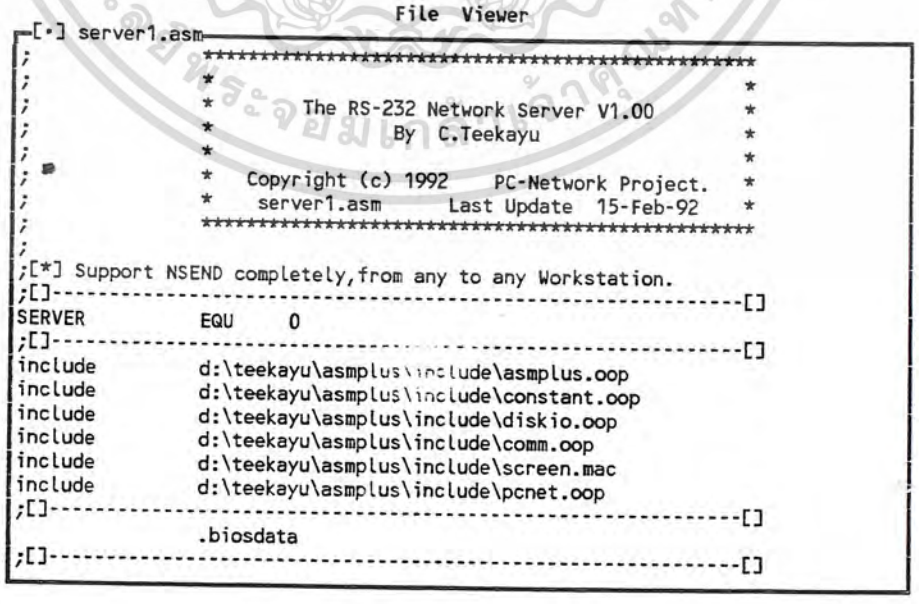
รูป 7.26 การสั่งให้ลบไฟล์

ในขณะที่ฝ่ายหนึ่งฝ่ายใดกำลังจัดการกับไฟล์อยู่นั้น ฝ่ายตรงข้ามจะถูกห้ามการทำงานชั่วขณะ พร้อมกับบอกสถานะดังแสดงในรูป 7.27



รูป 7.27 แสดงฝ่ายตรงข้ามกำลังทำงานที่สำคัญอยู่

นอกจากนี้ FTM ยังสามารถ view file ได้อีกด้วย เพื่อขอดู text file แต่แก้ไขไม่ได้ ผลของการเลือกคำสั่ง view แสดงดังรูป 7.28



รูป 7.28 View file

ในขณะที่ฝ่ายหนึ่ง view file อยู่นั้น ฝ่ายตรงข้ามจะถูกห้ามการทำงานดังรูป 7.27 จนกว่าการ view file จะสำเร็จ ฝ่ายตรงข้ามจึงจะสามารถใช้งานได้ตามปกติ เมื่อมีฝ่ายหนึ่งฝ่ายใดเลิกการทำงาน ด้วยการออกจากโปรแกรม ฝ่ายตรงข้ามก็จะรู้ได้โดยอัตโนมัติ หน้าจอจะกลับไปเป็นเหมือนขณะเริ่มใช้งานดังแสดงในรูป 7.22

7.4 การยกเลิกการติดตั้ง PC-Network

ระบบ PC-Network สามารถยกเลิกการติดตั้งใช้งานได้ เมื่อไม่ต้องการใช้งานแบบเครือข่ายอีกแล้ว โดยที่เครื่อง server ใช้คำสั่ง SERVER /U ก็จะเป็นการถอน resident ออกจากระบบ และที่เครื่อง Terminal ก็ใช้คำสั่ง NLOGOUT จะได้ผลดังแสดงในรูป 7.29 เป็นการออกจากเครือข่าย นอกจากนี้เมื่อต้องการติดตั้งใช้งานใหม่ ก็สามารถทำได้อีกโดยง่ายตามขั้นตอนตามที่กล่าวมาตั้งแต่ต้น จะเห็นว่าระบบ PC-Network ใช้งานได้ง่าย ความสามารถและประสิทธิภาพก็เหมาะสมกับค่าใช้จ่าย ซึ่งก็เป็นอีกทางเลือกหนึ่งของหน่วยงานที่ต้องการใช้ข้อมูลและอุปกรณ์

```
F:\USER\JUB\TEST >nlogout

PCNet V1.00 - Workstation user logout
(C) Copyright 1992 PC-Network Project. All Rights Reserved.

     PCNet Workstation logged out.

F:\USER\JUB\TEST >
```

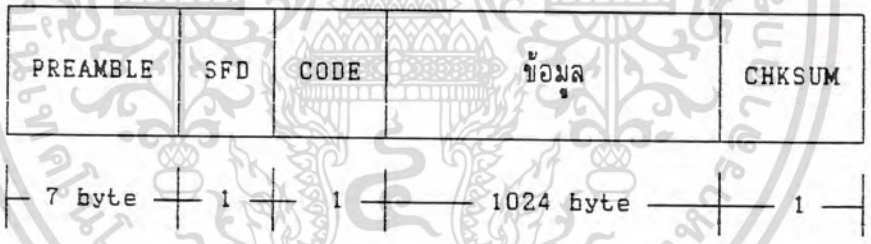
รูป 7.29 การ Logout ออกจากระบบ PC-Network

ร่วมกันเป็นระบบเครือข่าย โดยที่มีปริมาณงานไม่มากนัก และยังไม่พร้อมที่จะติดตั้งระบบ LAN ขนาดใหญ่

7.5 Concept ของการพัฒนา PC-Network

เนื่องจากต้องการให้เครื่องคอมพิวเตอร์ที่ต่อเชื่อมอยู่ใน Network สามารถติดต่อสื่อสารข้อมูลกันได้ตลอดเวลา ดังนั้นจึงต้องมีการพัฒนา Network Shell เป็น resident ฝังตัวอยู่ในหน่วยความจำของทุกเครื่องอยู่ตลอดเวลา เพื่อคอยจัดการควบคุมการรับ-ส่งข้อมูล ซึ่งการทำงานจะอาศัยการ interrupt ของเหตุการณ์ต่างๆมากระตุ้นให้กระทำต่างๆ เช่น interrupt จากคีย์บอร์ด ,จากพอร์ทสื่อสาร ,จากสัญญาณนาฬิการของเครื่อง เป็นต้น หลักการสำคัญก็จะต้องมี SERVER Shell ฝังตัวอยู่ที่เครื่อง SERVER และ LOGIN Shell ฝังตัวอยู่ที่เครื่อง Client ทุกเครื่อง จากนั้นเมื่อจะใช้งานผ่าน Network ก็เรียก Command Program ซึ่งจะไปเรียกใช้ Shell ให้บริการอีกทีหนึ่ง เช่น คำสั่ง transfer file ระหว่างเครื่อง ,คำสั่งส่งข้อความโต้ตอบกัน เป็นต้น

SERVER' Shell (SERVER.COM) จะดักการ interrupt จากพอร์ท COM1 และ COM2 เมื่อมีข้อมูลถูกส่งเข้ามาที่พอร์ท ก็จะเกิดการ interrupt กระโดดมายัง SERVER Shell SERVER Shell ก็จะตรวจสอบข้อมูล และทำงานตามชนิดของข้อมูลที่ได้รับนั้น โดยรูปแบบข้อมูลที่ได้รับจะอยู่ในรูป block format ดังรูป 7.30



รูป 7.30 block format ของข้อมูลที่ใช้ในการสื่อสาร

SERVER Shell จะตรวจสอบตาม block format ของข้อมูลถูกต้อง ก็จะทำงานตาม CODE ของข้อมูลนั้น ซึ่งอาจจะเป็นการส่ง block ของไฟล์ หรือข้อความสำหรับสื่อสารกัน เป็นต้น นอกจากนี้ SERVER Shell ยังจัดการ Network interrupt ที่ define ขึ้นมาเองคือ int.62h โดย Network interrupt นี้จะให้บริการแก่ Program Command Line ต่างๆ ที่เรียกใช้เข้ามาเมื่อ USER เรียกใช้งานการสื่อสารข้อมูลภายใน Network

ขณะที่ SERVER Shell ให้บริการแก่เครื่องใดเครื่องหนึ่งภายใน Network เครื่องอื่นๆที่ต้องการจะใช้งานก็จะต้องรอไปชั่งขณะหนึ่ง โดยภายในระบบการติดต่อสื่อสารของ PC-Network นี้ จะเป็นการส่ง Command Code Block ไปก่อน แล้วรอการตอบรับ ถ้ามีการ ACK การทำงานนั้นๆ ก็จะดำ

เนินต่อไป แต่ถ้ามีการตอบ NAK หรือไม่มีการตอบรับภายในระยะเวลาชั่วขณะหนึ่ง (ประมาณ 1-2 sec.) ก็จะยกเลิกการทำงานนั้น พร้อมกับแจ้งให้ USER ทราบ

LOGIN Shell (NLOGIN.COM) จะคอยจัการ Network interrupt (int. 62h) ซึ่งจะถูกเรียกใช้บริการจาก program command line ต่างๆ เมื่อ USER ต้องการติดต่อกับ Network เช่นการ tranfer file ,การส่งข้อความสื่อสาร เป็นต้น และ LOGIN Shell จะดักการ interrupt จากพอร์ทสื่อสารที่ใช้ (COM1 หรือ COM2) เพื่อรับข้อมูลที่ส่งมาจาก SERVER ซึ่งก็จะมี format ดังแสดงในรูปที่ 7.30 เช่นเดียวกัน จากนั้น LOGIN Shell ก็จะทำงานตาม CODE ของข้อมูลนั้นๆ เช่น รับ block ของข้อมูล ,ส่ง block ของข้อมูล ,อ่าน-เขียน ไฟล์ข้อมูล เป็นต้น

PROGRAM' COMMAND LINE เป็นโปรแกรมเรียกใช้งานภายใน Network โดยต้องเรียกใช้เมื่อ USER ต้องการใช้งานในแต่ละครั้ง ไม่ได้ฝังตัวไว้ในหน่วยความจำ ซึ่งโปรแกรมเหล่านี้ ก็จะไปเรียกใช้บริการบางอย่างที่จำเป็นในการติดต่อสื่อสารผ่าน Network จาก Shell ที่อยู่ในเครื่องนั้น ซึ่งอาจเป็นได้ทั้ง SERVER Shell หรือ LOGIN Shell โปรแกรมเหล่านี้ ได้แก่ NSEND ,NCOPY , NRUN ,REMOTE2 เป็นต้น

NSEND' ลำหรับใช้ในการส่งข้อความไปยัง station ต่างๆภายใน Network โดย NSEND จะเรียกบริการจาก Shell เพื่อส่ง Command block ไปยัง SERVER ถ้ามีการตอบรับเป็น ACK ก็แสดงว่าการส่งข้อความนั้นสำเร็จ ถ้าตอบเป็น NAK หรือไม่ตอบ ก็แสดงว่า การส่งข้อความไม่สำเร็จ เมื่อทำงานไปตามขั้นตอนนั้นแล้ว ก็จะแจ้งผลให้ USER ทราบ

NCOPY' ลำหรับใช้ในการ transfer file ระหว่างเครื่อง SERVER กับ Client โดย NCOPY ก็จะไปเรียกใช้บริการจาก Shell ทำการส่ง Command block ไปก่อน แล้วรอการตอบรับ ถ้ามีการตอบ ACK NCOPY ก็จะทำการส่งไฟล์ทีละ block แล้วรอการตอบรับเช่นเดิม ไปเรื่อยๆจนครบไฟล์ จึงจะเสร็จสิ้นการทำงาน แต่ถ้ามีการตอบ NAK NCOPY ก็จะทำการส่ง block เดิมไปให้ใหม่ ส่วนถ้ามิการตอบ CAN หรือไม่มีการตอบรับ NCOPY ก็จะยกเลิกการ transfer file

NRUN' ลำหรับใช้ในการ load file จากฮาร์ดดิสก์ ของเครื่อง SERVER ให้มาทำงานที่เครื่อง Client โดย NRUN จะส่ง Command block ไปยัง SERVER เมื่อ SERVER ตอบรับพร้อม

กับส่ง program นั้นๆมาให้ NRUN ก็จะรับมาไว้ในหน่วยความจำของเครื่องจนครบไฟล์ จากนั้น NRUN ก็จะจำลองตัวเองให้ทำงานเหมือน DOS ในการสั่ง RUN โปรแกรมต่างๆ โดย NRUN จะสร้าง PSP ให้กับโปรแกรมนั้นๆ ,เตรียมทางกลับ ,เตรียมรีจิสเตอร์ และสถานะต่างๆให้พร้อมที่จะทำงานกับโปรแกรมได้ เมื่อพร้อมแล้วก็จะโอนการควบคุมไปให้โปรแกรมนั้นทำงานจนเสร็จสิ้น การควบคุมจึงจะ return กลับมาที่ NRUN พร้อมกับ return code NRUN ก็จะรับมาพร้อมกับส่งต่อให้ DOS อีกทีหนึ่ง เป็นอันเสร็จสิ้นการทำงานของ NRUN

ในการโอนการทำงานไปให้ application program ทำงาน มีข้อกำหนดต่างๆที่จะต้องเตรียมการให้ตามลำดับดังนี้

1. จองหน่วยความจำให้กับโปรแกรม
2. รับข้อมูลจาก remote PC มาลงหน่วยความจำที่จองไว้
3. สร้าง PSP ให้กับโปรแกรม โดยข้อมูลใน PSP จะต้องเซ็ท DOS Environment ให้ถูกต้อง และเตรียมทางกลับโดยเซ็ท interrupt 22h ,23h ,24h ให้มาที่ routine ใน NRUN ซึ่งจะรอรับการ return กลับ เมื่อโปรแกรม terminated นอกจากนี้ต้องเซ็ทค่า default ใน PSP ดังนี้

- เซ็ท DTA ไปที่ PSP:0080h
- เซ็ท AX = drive ID

สำหรับการทำงานในขั้นตอนต่อไปนั้น จะแตกต่างกันไปตามชนิดของโปรแกรมว่าเป็น .COM หรือ .EXE โดยมีขั้นตอนแต่ละชนิดดังนี้

EXE Programs

จะต้องมีการ relocate ตำแหน่งโปรแกรมในหน่วยความจำ โดยเอาข้อมูลมาจาก header ของโปรแกรม หลังจากการทำ relocate เรียบร้อยแล้ว ก็จะมีขั้นตอนต่อไปดังนี้

- เซ็ท DS และ ES ให้เท่ากับ PSP segment
- เซ็ท CS ,IP ,SS และ SP ตามค่าที่กำหนดที่กำหนดไว้ใน header

จากนั้นก็ส่งการควบคุมไปให้โปรแกรมโดยการทำ FAR jump to the CS:IP

COM Programs

จะต้องเซ็ทดังต่อไปนี้

- เซ็ท CD ,DS ,ES และ SS ให้เท่ากับ PSP
- เซ็ท SP ไปยังท้ายสุด segment ของ PSP
- PUSH 00h ลง stack 1 word

จากนั้นก็ส่งการควบคุมไปให้โดยการ JMP to PSP:100

เมื่อการทำงานของโปรแกรมเสร็จสิ้นลง การควบคุมจะถูกส่งคืนให้ NRUN ผ่าน int. 22h ซึ่ง NRUN จะรับช่วงต่อ และส่งให้ DOS ต่อไป

REMOTE2 สำหรับใช้ควบคุมระยะไกลจาก SERVER ไปเครื่อง Client ต่างๆ โดย REMOTE2 จะส่ง Command block ไปยัง LOGIN Shell เพื่อขอเข้าควบคุมการใช้งานเครื่อง เมื่อ LOGIN Shell ตอบรับเรียบร้อย ต่อไปการกดคีย์บอร์ดต่างๆจากเครื่อง SERVER ก็จะถูกส่งไปควบคุมเครื่อง Client พร้อมกับรับหน้าจอมาจาก LOGIN Shell ซึ่งจะมีผลทำให้ SERVER สามารถควบคุม , ใช้งาน ,ติดตามผล เครื่อง Client ได้เสมือนเป็นเครื่องของ SERVER เอง

more ↑ EXE File Header Layout more ↓

Offset	Size	Contents	
+0	2	4Dh 5aH	.EXE file signature ('MZ')
+2	2	PartPag	length of partial page at end (generally ignored)
+4	2	PageCnt	length of image in 512-byte pages, including the header
+6	2	ReloCnt	number of items in relocation table
+8	2	HdrSize	size of header in 16-byte paragraphs
+0aH	2	MinMem	minimum memory needed above end of program (in paragrap
+0cH	2	MaxMem	maximum memory needed above end of program (in paragrap
+0eH	2	ReloSS	segment offset of stack segment (for setting SS)
+10H	2	ExeSP	value for SP register (stack pointer) when started
+12H	2	ChkSum	file checksum (negative sum of all words in file)
+14H	2	ExeIP	value for IP register (instruction pointer) when starte
+16H	2	ReloCS	segment offset of code segment (for setting CS)
+18H	2	TablOff	file-offset of first relocation item (often 001cH)
+1aH	2	Overlay	overlay number (0 for base module)
1cH		size of formatted portion of EXE header	
+ ?	4*?	offset segment | |offset segment	Relocation table. Starts file offset [EXE+18H]. Has [EXE+6] 4-byte entries.
+ ?	?	filler to a paragraph boundary	
+ ?	?	start of program image	

Since an EXE file may be loaded on any segment, all absolute segment referenc (such as FAR CALLs, long address pointers, and refs like MOV AX,data_seg) mus be adjusted to work for the memory location in which they are loaded. Here a the steps used by the DOS loader program (Fn 4bH) to load an EXE file:

1. create a PSP via DOS Fn 26H
2. read 1cH bytes of the EXE file (the formatted portion of the EXE header) into a local memory area.
3. determine the load module size = ((PageCnt*512)-(HdrSize*16)) - PartPag
4. determine file offset of load module = (HdrSize * 16)
5. select a segment address, START_SEG, for loading (usually PSP + 10H)
6. read the load module into memory starting at START_SEG:0000
7. LSEEK (set the file pointer) to the start of the relocation table (TablOf
8. for each relocation item (ReloCnt):
   a. read the item as two 16-bit words (I_OFF,I_SEG)
   b. add RELO_SEG=(START_SEG+I_SEG) (find the address of relocation ref)
   c. fetch the word at RELO_SEG:I_OFF (read current value)
   d. add START_SEG to that word (perform the segment fixup)
   e. store the sum back at its original address (RELO_SEG:I_OFF)
9. Allocate memory for the program according to MaxMem and MinMem
10. Initialize registers and execute the program:
    a. ES = DS = PSP
    b. Set AX to indicate the validity of drive IDs in command line
    c. SS = START_SEG+ReloSS, SP = ExeSP
    d. CS = START_SEG+ReloCS, IP = ExeIP (use: PUSH seg; PUSH offset; RETF)

Note: Recent additions to the EXE format, specifically the "CodeView" and "Windows" versions of the EXE file contain additional information embedded in the executable file. These additions are not covered in this version of TECH Help!

more ↑ Program Segment Prefix (PSP) more ↓

Offset	Size	Contents	
+0	2	INT 20H	EXE programs may JMP or RET here (PSP:0) to exit
+2	2	MemTop	top of available system memory in paragraphs
+4	1		(reserved)
+5	5	CALL offset segment	FAR CALL to DOS function dispatcher
+6		Avail	bytes available in Program Segment (for COM file only)
+0aH	4	offset segment	Terminate address. See INT 22H
+0eH	4	offset segment	Ctrl-Break handler address. See INT 23
+12H	4	offset segment	Critical Error handler addr. See INT 2
+16H	16H	DOS reserved area	
+2cH	2	EnvSeg	Segment address of DOS environment
+2eH	2eH	DOS reserved area	
+5cH	10H	formatted parm area 1	setup as an FCB for 1st cmd parameter
+6cH	14H	formatted parm area 2	setup as an FCB for 2nd cmd parameter
+80H	1	len	count of characters in UPA at 81H also offset of default DTA
+81H	7fH	Unformatted Parm Area	characters from DOS command line (except any redirection directives)
100H			Program Segment Prefix size

DOS Fn 26H Create PSP 4bH EXEC Program Startup & Exit Control Block Ind

รูป 7.32 Program Segment Prefix (PSP)

INT 22H: Terminate Address

The address at this vector (0000:0088) contains the address that will receive control when the currently-executing program terminates via any of:

```
        INT 20H   (traditional exit to DOS)
DOS Fn 00H
DOS Fn 4cH  (EXIT)

        INT 27H  (TSR: Terminate but Stay Resident)
DOS Fn 31H  (KEEP)
```

DOS takes this vector upon an Abort request during INT 24H (critical error).

The address at this vector is copied into the PSP Terminate Address field by DOS Fn 26H (create PSP) and Fn 4bH (EXEC).

Do not invoke INT 22H or call its address directly.

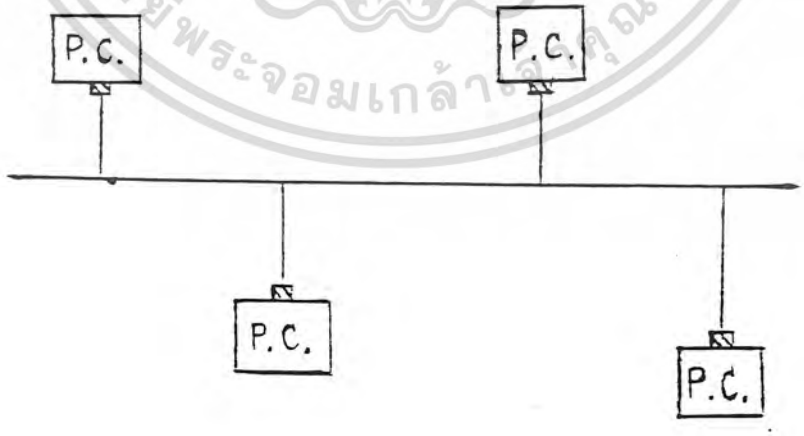
Program Startup & Exit ROM-BIOS Functions DOS Functio

รูป 7.33 INT 22H: Terminate Address

บทที่ 8
บทสรุปและแนวทางการพัฒนาต่อ

การวิจัยและพัฒนาโครงงาน PC-Network System สำเร็จตามจุดมุ่งหมายในขั้นหนึ่ง โดยสามารถเชื่อมโยงเครื่องไมโครคอมพิวเตอร์ ให้ติดต่อสื่อสาร ,แลกเปลี่ยนข้อมูล และใช้อุปกรณ์ต่อพ่วงภายในระบบ เช่น เครื่องพิมพ์ ,ฮาร์ดดิสก์ ร่วมกันได้ โดยมีประสิทธิภาพในการทำงานอยู่ในขั้นที่น่าพอใจในระดับหนึ่ง สามารถนำไปประยุกต์ใช้งานจริงกับระบบงานที่มีปริมาณงานไม่สูงมากนักได้ ซึ่งแนวทางในการพัฒนาต่อไปในอนาคต ควรมีการพัฒนาเพิ่มเติมในเรื่องของจำนวนเครื่องไมโครคอมพิวเตอร์ที่สามารถเชื่อมโยงกันในระบบให้มีจำนวนมากขึ้น และปรับปรุงประสิทธิภาพระบบควบคุมเครือข่าย (Network OS) ให้มีความสามารถมากขึ้น เช่น จัดสรรทรัพยากรภายในระบบให้แก่ USER ต่างๆใช้งานได้อย่างทั่วถึง ,ตอบสนองต่อการเรียกใช้งานของ USER ได้รวดเร็วยิ่งขึ้น เป็นต้น นอกจากนี้ ควรจะมีการเพิ่มเติมระบบรักษาความปลอดภัยของระบบเครือข่าย เช่น ระดับของการอนุญาตในการเข้าถึงข้อมูลตามความสำคัญ (Piority) ของ USER โดยการใช้รหัสผ่าน เป็นต้น

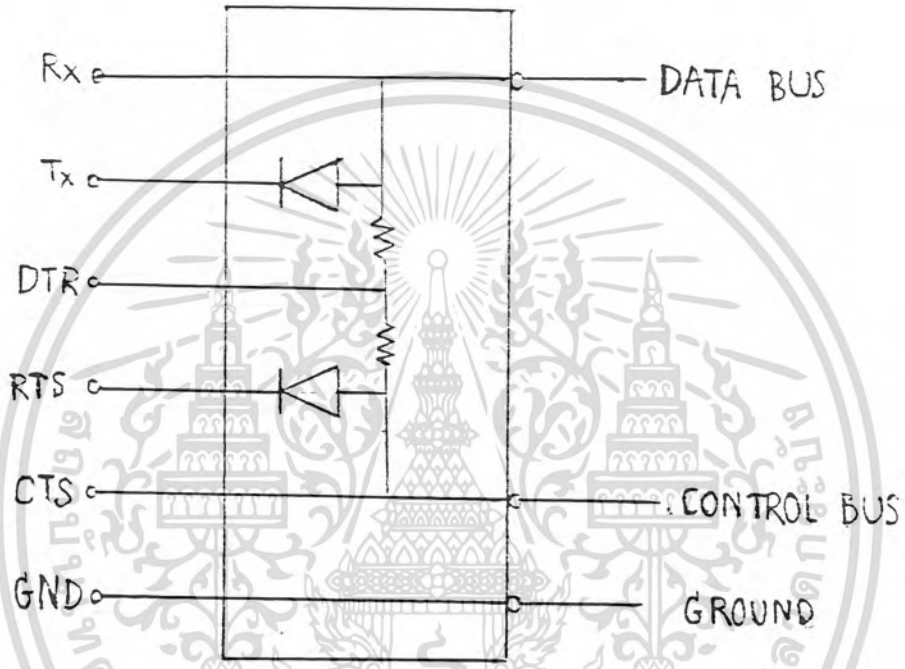
เครือข่ายนี้สามารถเชื่อมโยงเครื่องไมโครคอมพิวเตอร์ภายในระบบ ด้วยโทโพโลยีแบบอื่นนอกเหนือจากโทโพโลยีแบบดาวที่พัฒนาไว้นี้ได้ เช่น โทโพโลยีแบบบัส ที่เราได้ออกแบบไว้เพื่อเป็นแนวทางในการพัฒนาต่อไปในอนาคต มีการเชื่อมต่อดังรูป 8.1



รูป 8.1 การเชื่อมต่อเครือข่ายด้วย Bus Topology

โดยจะต้องมีวงจรอินเตอร์เฟส เพิ่มเติมที่พอร์ทอนุกรม เพื่อควบคุมทิศทางการไหลของข้อมูลภายในเครือข่าย ไม่ให้เกิดการชนกันของข้อมูล ซึ่งการชนกันนี้จะทำให้การรับส่งข้อมูลผิดพลาดได้

สำหรับวงจรอินเตอร์เฟส นอกเหนือจากขาสัญญาณ Rx และ Tx ที่ใช้สำหรับรับส่งข้อมูลแล้ว จะมีการใช้ขาสัญญาณ DTR ,RTS และ CTS ซึ่งตามมาตรฐาน RS-232C มีไว้สำหรับควบคุม modem มาใช้ในการควบคุมเครือข่ายแทน ลักษณะการต่อขาสัญญาณดังรูป 8.2



รูป 8.2 วงจรอินเตอร์เฟส

สำหรับสายสื่อสารข้อมูลจะประกอบไปด้วยสายสัญญาณ 3 เส้นคือ

1. DATA BUS ใช้ในการรับส่งข้อมูลระหว่างเครื่องไมโครคอมพิวเตอร์ภายในเครือข่าย ซึ่งเป็นการส่งข้อมูลในแบบ Haft-Duplex โดยในขณะใดขณะหนึ่ง จะมี node เพียง node เดียวเท่านั้น ที่สามารถใช้สายส่งข้อมูลนี้ได้

2. CONTROL BUS ใช้สำหรับควบคุมการจราจรของข้อมูลภายใน DATA BUS โดยจะเป็นตัวควบคุมสถานะของ DATA BUS ว่าว่างพร้อมที่จะใช้งานได้หรือไม่

3. GROUND ใช้สำหรับเทียบสัญญาณ

ซึ่งการพัฒนาระบบบัสนี้ต่อไป จะต้องมีการพัฒนาระบบควบคุมเครือข่าย (Network OS) ให้ควบคุมระบบบัสนี้ให้ทำงานได้ถูกต้อง โดยก่อนการส่งข้อมูลแต่ละครั้งจะต้องมีการตรวจสอบ CONTROL BUS ด้วยการอ่านสัญญาณจากขา CTS ว่าสาย DATA BUS ว่างหรือไม่ ถ้าว่างก็ทำการจอง DATA BUS ด้วยการส่งสัญญาณ RTS ออกไป จากนั้นจึงทำการส่งข้อมูลได้ เมื่อส่งข้อมูลเสร็จแล้วก็ต้องทำการคืนสาย DATA BUS ให้กับระบบ ด้วยการ off สัญญาณ RTS จะมีผลทำให้ node อื่นๆ สามารถเข้ามาใช้งาน DATA BUS ได้ด้วยวิธีการเดียวกัน

บรรณานุกรม

รศ. ประทีป บัญญัติชนรัตน์ ,การสื่อสารข้อมูล ,ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหาร ลาดกระบัง.

นิตยสารคอมพิวเตอร์รีวิว ฉบับ BBS ประจำเดือนกรกฎาคม พ.ศ. 2534.

Michael Hyman ,Advanced DOS ,management Information Source ,Inc. 3Rd. Edition 1989.

Martin D. Sayer ,Computer Guide to RS232 and Parallel connections , Prentice-Hall ,Englewoof Cliffs ,New Jersy. 1988.

Uyless D. Black ,Computer Networks ;Protocols ,Standards and Interfaces ,Prentice-Hall. 1987.

Uyless D. Black ,Data Communications and Distributed Networks , Prentice-Hall. 1987.

Douglas E. Comer ,Internetworking With TCP/IP ;Principles , Protocols and Architecture Prentice-Hall ,Englewoof Cliffs , New Jersy. 1988.

Tangney ,Brenden ,Local Area Network ,Prentice-Hall. 1988.

Tom Sheldon ,Osborne ,Novell Netware 386 :The Complete Reference , /McGraw-Hill ,Berkeley. CA. 1990.

PC/AT Technical Reference ,International Business Machines Corporation. 1St. Edition 1984.

Ray Duncan ,The MS-DOS Encyclopedia ,:Microsoft Press. 1990.

Dare Williams,The Programmer's technical reference :MS-DOS ,IBM PC & Compatibles :John Wiley/Sigma Press. Cheshire, England. 1991.

Turbo Assembler :Reference Guide & User's Guide Version 2.0 BORLAND internation ,Inc. SCOTTS VALLEY CA. 1988.

ภาคผนวก

```
;                ************************************************
;                *                                              *
;                *        The RS-232 Network Server V1.00       *
;                *                By  C.Teekayu                 *
;                *                                              *
;                *   Copyright (c) 1992    PC-Network Project.  *
;                *    server2.asm      Last Update '19-Feb-92    *
;                ************************************************
;
;[*] Add Support NCOPY from Workstation to Workstation.
;[]-------------------------------------------------------------[]
SERVER          EQU     0
;[]-------------------------------------------------------------[]
include         d:\teekayu\asmplus\include\asmplus.oop
include         d:\teekayu\asmplus\include\constant.oop
include         d:\teekayu\asmplus\include\diskio.oop
include         d:\teekayu\asmplus\include\comm.oop
include         d:\teekayu\asmplus\include\screen.mac
include         d:\teekayu\asmplus\include\pcnet.oop
;[]-------------------------------------------------------------[]
                .biosdata
;[]-------------------------------------------------------------[]
code            segment byte public 'CODE'
                assume cs:code,ds:code
                org 100h
start:
                jmp install
;[]-------------- INTERRUPT SERVICE ROUTINE --------------------[]
Reference_point label   byte
;[]-------------------------------------------------------[]
; network interrupt service routine
;[]-------------------------------------------------------[]
int62h          proc    far
                cli
                push ds
                mov bx,cs
                mov ds,bx
                assume ds:code
                call NetISR
                pop ds
                sti
                iret
int62h          endp
;
;[]-------------------------------------------------------[]
; disk interrupt service routine
;[]-------------------------------------------------------[]
int13h          proc    far
                inc byte ptr cs:inisr13h
                pushf
                cli
                call cs:oldint13h
                pushf
                dec byte ptr cs:inisr13h
                popf
                sti
                ret 2                   ; simurate IRET without poping
int13h          endp
;
;[]-------------------------------------------------------[]
; timer tick interrupt service routine
;[]-------------------------------------------------------[]
int1ch          proc    far
                pushf
                cli
                call cs:oldint1ch
                sti
                iret
```

```
int1ch          endp
;
;[]-------------------------------------------------------[]
; dos idle interrupt service routine
;[]-------------------------------------------------------[]
int28h          proc    far
                inc byte ptr cs:inisr28h
                pushf
                cli
                call cs:oldint28h
                dec byte ptr cs:inisr28h
                sti
                iret
int28h          endp
;
;[]-------------------------------------------------------[]
; com1 interrupt service routine
;[]-------------------------------------------------------[]
intcom1         proc    far
                cli
                pushaa
                commOFF
                mov ax,cs
                mov ds,ax
                mov es,ax
                assume ds:code,es:code
                mov dx,COM1ADDR
                in al,dx
                mov _baseaddr,dx
                mov di,offset User1name
                call ComISR
                eoi
                commON
                popaa
                sti
                iret
intcom1         endp
;
;[]-------------------------------------------------------[]
; com2 interrupt service routine
;[]-------------------------------------------------------[]
intcom2         proc    far
                cli
                pushaa
                commOFF
                mov ax,cs
                mov ds,ax
                mov es,ax
                assume ds:code,es:code
                mov dx,COM2ADDR
                in al,dx
                mov _baseaddr,dx
                mov di,offset User2name
                call ComISR
                eoi
                commON
                popaa
                sti
                iret
intcom2         endp
;
;[]------------ data ----------------------------------------[]
Stat            db SERVER
TSRPSP          dw ?                            ; save tsr segment
InDOSAddr       dd ?                            ; address of MS-DOS In
oldint13h       dd ?
oldint28h       dd ?
oldint1ch       dd ?
```

```
oldint62h       dd ?
oldintcom1      dd ?
oldintcom2      dd ?
inisr13h        db 0
inisr28h        db 0
_baseaddr       dw COM1ADDR                     ; com1
_Divisor        db 1                            ; 115200
_DataFormat     db 00000011b                    ; 8N1
_videoaddr      dw MONO
;
;[]-------------------------------------------------------[]
include         netbios.oop
;[]-------------------------------------------------------[]
;
;[]-------------------------------------------------------[]
; table of routines within communicate interrupt  COM1
;[]-------------------------------------------------------[]
ComISRtbl       label   word
                dw offset UserLogin                     ;80h
                dw offset UserLogout                    ;81h
                dw offset Msg2ALL                       ;82h
                dw offset Msg2Server                    ;83h
                dw offset Msg2Client                    ;84h
                dw offset Ack2Another                   ;85h
                dw offset Nak2Another                   ;86h
                dw offset File2Server                   ;87h
                dw offset File2Client                   ;88h
                dw offset FileFromServer                ;89h
                dw offset FileFromClient                ;8Ah
                dw offset RemoteRun                     ;8Bh
EndComtbl       equ   $-ComISRtbl
;
;[]-------------------------------------------------------[]
;dispatch routine - al=code,dx=_baseaddr
;[]-------------------------------------------------------[]
ComISR          proc    near
                sub al,80h
                jc  comisr00
                xor bx,bx
                mov bl,al
                shl bx,1
                cmp bx,EndComtbl
                jb  comisr01
comisr00:
                ret
comisr01:
                jmp word ptr [bx+offset ComISRtbl]
ComISR          endp
;
;[]-------------------------------------------------------[]
; data for communication routine
;[]-------------------------------------------------------[]
UserName        label   byte
User2name       db 10 dup (0)
User1name       db 10 dup (0)
;
headmsg         db '•>> From ['          ; 10
station         db '1] : '               ; 5
MsgBuffer       db 65 dup (0)            ; 65
line24          dw 80 dup (0)
;
FileType        db 00h                   ; COM type
filespec        db 128 dup (0)
_Filehandle     dw 0
CopyBuffer      db 1024 dup (0)
;[]-------------------------------------------------------[]
; Routine support communication ISR
```

```
;[]-------------------------------------------------------[]
DoNothing       proc    near
                ret
DoNothing       endp
;
;[]-------------------------------------------------------[]
; Registation user login
; input  -  si -> username buffer
;[]-------------------------------------------------------[]
UserLogin       proc    near
login01:
                getcom
                stosb
                or al,al
                jnz login01
                xsend ACK
                ret
UserLogin       endp
;
;[]-------------------------------------------------------[]
; Registation user logout
; input  -  si -> username buffer
;[]-------------------------------------------------------[]
UserLogout      proc    near
                xor ax,ax                       ; mark zero
                stosb
                ret
UserLogout      endp
;
;[]-------------------------------------------------------[]
; Receive msg, send to another client and show on own screen
;[]-------------------------------------------------------[]
Msg2ALL         proc    near
                xor dh,1
                xsend 83h                       ; to wake up another
                xor dh,1
                mov di,offset MsgBuffer
                cld
msg2A01:                                        ; receive msg
                getcom
                stosb
                xor dh,1
                xsend
                xor dh,1
                or al,al
                jnz msg2A01
msg2A02:
                xor dh,1
                call GetResponse
                xor dh,1
                jc  msg2A03
                cmp al,ACK
                je  msg2A04
msg2A03:
                mov al,NAK
msg2A04:
                xsend                           ; acknowledge
                call ShowMsg                    ; another client will
                ret
Msg2ALL         endp
;
;[]-------------------------------------------------------[]
; Receive msg, show on own screen
;[]-------------------------------------------------------[]
Msg2Server      proc    near
                mov di,offset MsgBuffer
                cld
msg2S01:                                        ; receive msg
```

```
                getcom
                stosb
                or al,al
                jnz msg2S01
msg2S02:
                xsend ACK                       ; acknowledge
                call ShowMsg
                ret
Msg2Server      endp
;
; show msg on own screen
ShowMsg         proc    near
                mov al,'1'
                cmp dh,03h
                je  get_source
                inc al
get_source:
                mov station,al
                mov di,offset line24            ; save line 24
                mov si,row24
                mov ax,_videoaddr
                mov ds,ax
                assume ds:nothing
                mov cx,80
                rep movsw
                mov es,ax
                assume es:nothing
                mov ax,cs
                mov ds,ax
                assume ds:code
                mov ah,03h                      ; read cursor position
                xor bx,bx
                int 10h
                push dx                         ; save cursor
                mov ah,02h                      ; set cursor hide
                mov dx,1900h                    ; to row 25
                int 10h
                mov di,row24
                mov si,offset headmsg
                mov ah,70h
                mov cx,80
msg2S03:                                        ; show msg
                lodsb
                or al,al
                jz msg2S04
                stosw
                loop msg2S03
msg2S04:
                mov al,20h
                rep stosw
                beep
                readkey                         ; wait anykey
                mov di,row24                    ; restore line 24
                mov si,offset line24
                mov cx,80
                rep movsw
                pop dx                          ; restore cursor
                mov ah,02h
                xor bx,bx
                int 10h
                ret
ShowMsg         endp
;
;[]------------------------------------------------------[]
; Receive msg, send to another client
;[]------------------------------------------------------[]
Msg2Client      proc    near
                xor dh,1
```

```
                xsend 83h                       ; wake up another
                xor dh,1
msg2C01:
                getcom
                xor dh,1
                xsend
                xor dh,1
                or al,al
                jnz msg2C01
msg2C02:                                        ; another client will
                xor dh,1
                call GetResponse
                xor dh,1
                jc  msg2C03
                cmp al,ACK
                je  msg2C04
msg2C03:
                mov al,NAK
msg2C04:
                xsend
                ret
Msg2Client      endp
;
;[]-------------------------------------------------------[]
; Read file and send block to client
;[]-------------------------------------------------------[]
FileFromServer  proc    near
                mov di,offset filespec
ffs01:
                getcom
                stosb
                or al,al
                jnz ffs01
                fopen filespec,0
                jc ffsErr01
                mov _Filehandle,ax
                send ACK
                call SendFile
                jnc  ffs02
                send CAN
ffs02:
                fclose
                jmp short ffsEnd
ffsErr01:
                send NAK
ffsEnd:
                ret
FileFromServer  endp
;
;[]-------------------------------------------------------[]
; Send Request,Receive block from another and send to client
;[]-------------------------------------------------------[]
FileFromClient  proc    near
                ret
FileFromClient  endp
;
;[]-------------------------------------------------------[]
; Send Acknowledge to another client
;[]-------------------------------------------------------[]
Ack2Another     proc    near
                xor dh,1
                xsend ACK
                xor dh,1
                ret
Ack2Another     endp
;
;[]-------------------------------------------------------[]
; Send Negative Acknowledge to another client
```

```
;[]-------------------------------------------------------[]
Nak2Another     proc    near
                xor dh,1
                xsend NAK
                xor dh,1
                ret
Nak2Another     endp
;
;[]-------------------------------------------------------[]
; Receive file from client
;[]-------------------------------------------------------[]
File2Server     proc    near
                xsend ACK
                call ReceiveFile
                ret
File2Server     endp
;
;[]-------------------------------------------------------[]
; Transmit file to client
;[]-------------------------------------------------------[]
File2Client     proc    near
                ret
File2Client     endp
;
;[]-------------------------------------------------------[]
; Receive file from client
;[]-------------------------------------------------------[]
ReceiveFile     proc    near
                cli
RecFile01:
                receive
                cmp al,SOH
                jne RecFile02
                mov si,offset filespec
                call RecString
                jnz RecNAK
                fcreate filespec,0      ; normal
                jc RecCANCEL
                mov _Filehandle,ax
RecACK:
                send ACK
                jmp short RecFile01
RecNAK:
                send NAK
                jmp  RecFile01
RecCANCEL:
                send CAN
                stc
                jmp RecFile06
RecFile02:
                cmp al,STX
                jne RecFile03
                call ReceiveBlk
                jnz RecNAK
                fwrite CopyBuffer,1024
                jc RecError01
                jmp RecACK
RecFile03:
                cmp al,ETX
                jne RecFile04
                call ReceiveByCX
                jnz RecNAK
                or cx,cx
                jz RecFile05
                fwrite CopyBuffer,cx
                jnc RecFile05
RecError01:
                send CAN
```

```
                jmp short RecError02
RecFile04:
                cmp al,CAN
                je  RecError02
                jmp RecFile01
RecError02:
                fclose
                stc
                jmp short RecFile06
RecFile05:
                send ACK
                fclose
                clc
RecFile06:
                sti
                ret
ReceiveFile     endp
;
; si -> buffer
RecString       proc    near
                push si
                xor bx,bx
RecString01:
                receive
                add bl,al
                mov [si],al
                inc si
                or al,al
                jnz RecString01
                receive
                cmp bl,al
                pop si
                ret
RecString       endp
;
ReceiveBlk      proc    near
                mov si,offset CopyBuffer
                xor bx,bx
                mov cx,1024
ReceiveBlk01:
                receive
                add bl,al
                mov [si],al
                inc si
                loop ReceiveBlk01
                receive                 ; bl=own chksum, al=remote chk
                cmp al,bl               ; return ZF flag
                ret
ReceiveBlk      endp
;
ReceiveByCX     proc    near
                receive
                mov ch,al
                receive
                mov cl,al
                or cx,cx
                jz ReceiveByCX02
                push cx                 ; save total byte to receive
                mov si,offset CopyBuffer
                xor bx,bx
ReceiveByCX01:
                receive
                add bl,al
                mov [si],al
                inc si
                loop ReceiveByCX01
                pop cx                  ; restore total Byte received
                receive
```

```
                cmp al,bl
ReceiveByCX02:
                ret
ReceiveByCX     endp
;
;[]-------------------------------------------------------[]
; Transmit file to client
;[]-------------------------------------------------------[]
SendFile        proc    near
SendFile01:
                send SOH
                receive
                cmp al,NAK
                je SendFile01
                cmp al,CAN
                je SendError02
                cmp al,ACK
                je SendFile02
SendError01:
                send CAN
SendError02:
                fclose
                stc
                jmp SendFile07
SendFile02:
                fread CopyBuffer,1024
                jc SendError01
                cmp ax,1024
                jb SendFile04
SendFile1024:
                call SendBlk                    ; STX
                receive
                cmp al,NAK
                je SendFile1024
                cmp al,CAN
                je SendError02
                cmp al,ACK
                jne SendError01
                jmp short SendFile02
SendFile04:
                mov cx,ax
SendFile05:
                call SendByCX                   ; CX not changed
                receive
                cmp al,NAK
                je SendFile05
                cmp al,CAN
                jne SendFile06
                jmp SendError02
SendFile06:
                fclose
                clc
SendFile07:
                ret
SendFile        endp
;
SendBlk         proc    near
                send STX
                mov si,offset CopyBuffer
                xor bx,bx                       ; clear
                mov cx,1024
SendBlk01:
                lodsb                           ; get data
                mov ah,al                       ; save data
                add bl,al                       ;
                send ah                         ;
                loop SendBlk01
                send bl
```

```
                ret
SendBlk         endp
;
SendByCx        proc    near
                send ETX
                send ch                 ; size in dx
                send cl
                or cx,cx
                jz SendByCX02
                push cx
                mov si,offset CopyBuffer
                xor bx,bx
SendByCX01:
                lodsb                   ; get data
                mov ah,al               ; save data
                add bl,al               ;
                send ah                 ;
                loop  SendByCX01
                send bl
                pop cx
SendByCX02:
                ret
SendByCX        endp
;
RemoteRun       proc    near
                mov di,offset filespec
rr01:
                getcom
                stosb
                or al,al
                jnz rr01
                dec di
                mov al,'.'
                stosb
                mov al,'c'
                mov ah,'o'
                stosw
                xor ax,ax
                mov al,'m'
                stosw
                mov byte ptr FileType,00h       ; COM type
                fopen filespec,0
                jnc rr02
                sub di,4
                mov al,'e'
                mov ah,'x'
                stosw
                xor ax,ax
                mov al,'e'
                stosw
                fopen filespec,0
                jc  rrErr01
                mov byte ptr FileType,0FFh      ; EXE type
rr02:
                mov _Filehandle,ax
                xsend FileType
                call SendFile
                jnc rr03
                send CAN
rr03:
                fclose
                jmp short rrEnd
rrErr01:
                mov al,4                ; file not found
                xsend
rrEnd:          ret
RemoteRun       endp
```

```
;
GetResponse     proc    near
                mov bx,2
GetResponse0:
                mov cx,0ffffh
                mov dl,LSR
GetResponse1:
                in al,dx
                jmp short $+2
                test al,1
                jnz  GetResponse2
                loop GetResponse1
                dec bx
                jnz GetResponse0
                mov dl,RXBUF
                stc
                ret
GetResponse2:
                mov dl,RXBUF
                in al,dx
                clc
                ret
GetResponse     endp
;
;-----------------------------------------------
; PROC      VerifyDOSState
; DESCRIPT  Verify Status of DOS
; RETURN:   carry flag set if MS_DOS is busy
;-----------------------------------------------
VerifyDOSState  PROC    near
                cmp byte ptr cs:inisr13h,0
                jne dos_busy
                cmp byte ptr cs:inisr28h,0
                jz  dos_ok
                push ds
                lds bx,cs:InDOSAddr     ;ds:bx = InDOSAddr
                mov al,[bx]             ;al = InDOS flag
                pop ds
                or  al,al
                jz  dos_ok
dos_busy:
                stc
                ret
dos_ok:
                clc
                ret
VerifyDOSState  ENDP
;
;[]------------ INSTALL SECTION ---------------------------------
;[]----------------------------------------------------------[]
; install TSR  ds -> cs ->es -> data
;[]----------------------------------------------------------[]
install         proc    near
                mov TSRPSP,es           ; save TSR PSP
                mov ax,3562h            ; get func. 62h
                int 21h                 ;
                cmp bx,offset int62h    ; is own interrupt routine ?
                jne install1            ; no,to install
                mov si,81h              ; ds:si
                cld
uninstall0:
                lodsb
                cmp al,20h
                je  uninstall0
                cmp al,0dh
                je  uninstall1
                cmp al,'/'
                jne uninstall1
```

```
                lodsb
                or al,20h
                cmp al,'u'
                jne uninstall1
                xor ax,ax                       ; code al=00h (uninstall)
                int 62h                         ; return  ax=TSRPSP
                mov cs:TSRPSP,ax
                call freemem
                jmp short install0
uninstall1:
                mov ah,09h                      ; already installed !
                mov dx,offset already           ;
                int 21h
install0:
                mov ax,4c00h                    ; terminate
                int 21h                         ;
install1:
                cli
                mov word ptr oldint62h,bx       ;
                mov word ptr oldint62h+2,es     ;
                mov ax,2562h                    ; set interrupt vector 62h
                mov dx,offset int62h            ;
                int 21h                         ;
; Get InDOS address from MS-DOS
                mov ah,34h                      ;
                int 21h                         ; es:bx -> InDOS
                mov word ptr InDOSAddr,bx       ;
                mov word ptr InDOSAddr+2,es;
; install int. 13h handler ( disk i/o )
                mov ax,3513h                    ; get interrupt vector 13h
                int 21h                         ;
                mov word ptr oldint13h,bx       ;
                mov word ptr oldint13h+2,es     ;
                mov ax,2513h                    ; set interrupt vector 13h
                mov dx,offset int13h            ;
                int 21h                         ;
; install int. 1ch handler ( Timer tick )
                mov ax,351ch                    ; get 1ch
                int 21h                         ;
                mov word ptr oldint1ch,bx       ;
                mov word ptr oldint1ch+2,es;
                mov ax,251ch                    ; set 1ch
                mov dx,offset int1ch            ;
                int 21h                         ;
; install int. 28h handler ( DOS Idle )
                mov ax,3528h                    ; get interrupt vector 28h
                int 21h                         ;
                mov word ptr oldint28h,bx       ;
                mov word ptr oldint28h+2,es     ;
                mov ax,2528h                    ; set interrupt vector 28h
                mov dx,offset int28h            ;
                int 21h                         ;
; install int. communication handle (COM1/COM2)
                mov ax,350bh                    ; com2
                int 21h                         ;
                mov word ptr oldintcom2,bx      ;
                mov word ptr oldintcom2+2,es;
                mov ax,250bh                    ; set interrupt vector com2
                mov dx,offset intcom2           ;
                int 21h                         ;
                mov ax,350ch                    ; com1
                int 21h
                mov word ptr oldintcom1,bx      ;
                mov word ptr oldintcom1+2,es;
                mov ax,250ch                    ; set interrupt vector com1
                mov dx,offset intcom1           ;
                int 21h                         ;
                mov _baseaddr,COM2ADDR          ;
```

```
                call commset
                mov _baseaddr,COM1ADDR
                call commset
; ready to make resident
                mov dx,offset showmsg1  ;
                mov ah,9                ;
                int 21h                 ;
                call initvideoram       ;
                commON                  ;
; Terminate and stay resident
                mov dx,offset install   ;
                int 27h                 ;
install         endp
;
;[]-------------------------------------------------------[]
; set video ram buffer address
;[]-------------------------------------------------------[]
initvideoram    proc    near
                mov ah,0fh              ; get video mode
                int 10h                 ;
                cmp al,7                ; monochrome ?
                jz is001                ;
                mov _videoaddr,COLOR    ; no,set to video ram addr. of
is001:
                mov ax,_videoaddr       ;
                mov es,ax               ;
                ret                     ;
initvideoram    endp
;
;[]-------------------------------------------------------[]
; release memory
;[]-------------------------------------------------------[]
freemem         proc    near
                push es
                push ax
                mov es,cs:TSRPSP
                mov es,es:[2ch]
                mov ah,49h
                int 21h
                mov es,cs:TSRPSP
                mov ah,49h
                int 21h
                pop ax
                pop es
                ret
freemem         endp
;
;[]-------------------------------------------------------[]
; Proc          COMMSet
; Descript      Initialize the serial port by setting 8250 ( UART )
;[]-------------------------------------------------------[]
COMMSet         Proc    near
                cli                     ;
                mov dx,_baseaddr        ;
                inc dx                  ; point to interrupt enable re
                sub al,al               ; reset interrupt enables
                out dx,al               ;
;--Initialize the baud rate divisor registers for parameter (baud)
                add dx,2                ; point to line control regist
                mov al,10000000b        ; turn on bit 7
                out dx,al               ;
                dec dx                  ; point to buad rate divisor(h
                dec dx                  ;
                xor ax,ax               ; divisor = 0
                out dx,al               ;
                dec dx                  ; point to buad rate divisor(l
                mov al,_Divisor         ;
                out dx,al               ;
```

```
;--Initialize the line control register
                add dx,3                ; point to line control regist
                xor ax,ax               ; turn off bit 7
                or al,_Dataformat       ; defalt=8 bit data,1 stop bit
                out dx,al               ;
;--Initialize modem control register for enable interrupt
                inc dx                  ; point to modem control regis
                mov al,00001011b        ; turn on bit 3 (OUT2) for ena
                                        ; and turn on DTR(bit 0) ,RTS(
                out dx,al               ;
;--Read to clear possible old interrupt
                inc dx                  ; point to line status reg.
                in al,dx                ;
                jmp short $+2           ;
                inc dx                  ; point to modem status reg.
                in al,dx                ;
;--Initialize the interrupt enable register:
                sub dx,5                ; point to interrupt enable re
                mov al,00000001b        ; enable data rec.
                out dx,al               ;
;--Clear register
ClRgs:
                dec dx                  ;
                in al,dx                ; clear possible old data
                sti
                ret
COMMSet         Endp
;
;[]------------ TRANSIENT DATA ------------------------------------
;
showmsg1        db cr,lf,'
                db cr,lf,'
                db cr,lf,'
                db cr,lf,'
                db cr,lf,'
                db cr,lf,'
                db cr,lf,'
                db cr,lf,'
                db cr,lf,'$'
;
already         db cr,lf,'
                db cr,lf,'
                db cr,lf,'
                db cr,lf,bell,'$'
;
;[]------------ END SECTION ---------------------------------------
;
code            ends
                end     start
```

```
┌──────────────────────────────────────────────────┐
│        The PC-Network Server Manager             │
│            Version 1.00 (3 user)                 │
│                                                  │
│          Written by   C.Teekayu                  │
│                                                  │
│  Copyright (c) 1992  PC-Network Project.         │
└──────────────────────────────────────────────────┘
```

```
┌──────────────────────────────────────────────────┐
│  The PC-Network Server Manager already ins       │
└──────────────────────────────────────────────────┘
```

```
;                ************************************************
;                *                                              *
;                *    The PC-Network Workstation Login  V1.00   *
;                *                  By   C.Teekayu              *
;                *                                              *
;                *   Copyright (c) 1992    PC-Network Project.  *
;                *    nlogin.asm       Last Update  19-Feb-92   *
;                ************************************************
;
;[]-------------------------------------------------------------[]
CLIENT          EQU     -1
;[]-------------------------------------------------------------[]
include         d:\teekayu\asmplus\include\asmplus.oop
include         d:\teekayu\asmplus\include\constant.oop
include         d:\teekayu\asmplus\include\diskio.oop
include         d:\teekayu\asmplus\include\comm.oop
include         d:\teekayu\asmplus\include\screen.mac
include         d:\teekayu\asmplus\include\pcnet.oop
;[]-------------------------------------------------------------[]
Cut00FFh        macro
                LOCAL Cut,NotCut
                or al,al
                jz Cut
                cmp al,0ffh
                jz Cut
                jmp short NotCut
Cut:
                mov al,20h
NotCut:
                endm
;[]-------------------------------------------------------------[]
                .biosdata
;[]-------------------------------------------------------------[]
code            segment byte public 'CODE'
                assume cs:code,ds:code,es:code
                org 100h
start:
                jmp install
;[]------------- INTERRUPT SERVICE ROUTINE --------------------[]
Reference_point label   byte
;[]-------------------------------------------------------------[]
; network interrupt service routine
;[]-------------------------------------------------------------[]
int62h          proc    far
                cli
                push ds
                mov bx,cs
                mov ds,bx
                assume ds:code
                call NetISR
                pop ds
                sti
                iret
int62h          endp
;
;[]-------------------------------------------------------------[]
; disk interrupt service routine
;[]-------------------------------------------------------------[]
int13h          proc    far
                inc byte ptr cs:inisr13h
                pushf
                cli
                call cs:oldint13h
                pushf
                dec byte ptr cs:inisr13h
                popf
                sti
                ret 2                           ; simurate IRET without poping
```

```
int13h          endp
;
;[]-------------------------------------------------------[]
; dos idle interrupt service routine
;[]-------------------------------------------------------[]
int28h          proc    far
                inc byte ptr cs:inisr28h
                pushf
                cli
                call cs:oldint28h
                dec byte ptr cs:inisr28h
                sti
                iret
int28h          endp
;
SCREENON        db      FALSE
TickTack        db      0
;[]-------------------------------------------------------[]
; timer tick interrupt service routine
;[]-------------------------------------------------------[]
int1ch          proc    far
                pushf
                cli
                call cs:oldint1ch
                cli
                cmp byte ptr cs:SCREENON,FALSE
                jz  exit_isr1ch
                push ds
                push ax
                push bx
                .dataseg
                mov al,TickTack
                or  al,al
                jz  isr1ch00
                dec ax
                jmp short isr1ch02
isr1ch00:
                ;cmp byte ptr inisr13h,0
                ;jnz end_isr1ch
                cmp byte ptr inisr28h,0
                jz  isr1ch01
                lds bx,cs:InDOSAddr     ;ds:bx = InDOSAddr
                mov al,[bx]             ;al = InDOS flag
                or  al,al
                jnz end_isr1ch
isr1ch01:
                call ScreenManager
                mov al,2
isr1ch02:
                mov TickTack,al
end_isr1ch:
                pop bx
                pop ax
                pop ds
exit_isr1ch:
                sti
                iret
int1ch          endp
;
;[]-------------------------------------------------------[]
; com interrupt service routine
;[]-------------------------------------------------------[]
intcomm         proc    far
                cli
                pushaa
                .dataseg
                mov dx, baseaddr
                in al,dx
```

```
                call ComISR
                eoi
                popaa
                sti
                iret
intcomm         endp
;
;[]------------ data ------------------------------------------[]
Stat            db CLIENT
TSRPSP          dw ?                            ; save tsr segment
InDOSAddr       dd ?                            ; address of MS-DOS In
oldint13h       dd ?
oldint28h       dd ?
oldint1ch       dd ?
oldint62h       dd ?
oldintcomm      dd ?
;
inisr13h        db 0
inisr28h        db 0
_baseaddr       dw COM1ADDR                     ; com1
_Divisor        db 1                            ; 115200
_DataFormat     db 00000011b                    ; 8N1
_intcom         db 0ch
_videoaddr      dw MONO
;
_Filehandle     dw 0
FileType        db 0
filespec        db 256 dup (0)
CopyBuffer      db 1024 dup (0)
;
;[]--------------------------------------------------------[]
include         netbios.oop
;[]--------------------------------------------------------[]
;
Screen          db 1024*2 dup (0)
EndScreen       dw ?
;
; Re-arrange screen use ds,es,ax,cx,dx,si,di
GetScreen       proc    near
                mov ax,cs:_videoaddr
                mov ds,ax
                assume ds:nothing
                mov ax,cs
                mov es,ax
                assume es:code
                xor si,si                       ; ds:si -> screen
                mov di,offset cs:Screen         ; es:di -> buffer
                lodsw                           ; get first
                Cut00FFh
                mov dx,ax                       ; save 1st
                xor ax,ax                       ; keep 0
                mov ah,dh                       ; keep attr.
                stosw                           ; start command code
GetScreen00:
                cmp si,row24+160                ; is end of screen ?
                jae GetScreen06
                cmp di,offset cs:EndScreen      ; is buffer full ?
                jae GetScreen06
                lodsw                           ; 2nd
                Cut00FFh
                cmp ax,dx
                jne Getscreen01
                lodsw                           ; 3rd
                Cut00FFh
                cmp ax,dx
                jne  GetScreen02
                lodsw                           ; 4th
                Cut00FFh
```

```
                cmp ax,dx
                je GetScreen03
                xchg ax,dx              ; save as 1st (dx)
                stosb                   ; store 3 prev char.
                stosb                   ;
                stosb                   ;
GetScreen05:
                cmp ah,dh               ; attr. changed ?
                je  GetScreen00
                xor al,al               ; keep 0 in al, ah=attr.
                mov ah,dh               ; get current attr.
                stosw                   ; store attr. code
                jmp short GetScreen00
GetScreen01:
                xchg ax,dx              ; save as 1st (dx)
                stosb                   ; store prev char.
                jmp short GetScreen05
GetScreen02:
                xchg ax,dx              ; save as 1st (dx)
                stosb                   ; store 2 prev char.
                stosb                   ;
                jmp short GetScreen05
GetScreen03:
                mov al,0ffh             ; keep FFH
                mov ah,dl               ; keep char.
                stosw
                mov cx,3                ; reentant counter
GetScreen04:
                inc cx
                lodsw
                Cut00FFh
                cmp ax,dx
                je GetScreen04
                xchg ax,cx
                stosw                   ; store reentant counter
                mov ax,dx               ; prev. char
                mov dx,cx               ; next char
                jmp short GetScreen05
GetScreen06:
                mov cs:EndScreen,di
                ret
GetScreen       endp
;
ScreenManager   proc    near
                push cx
                push dx
                push es
                push si
                push di
                call GetScreen          ; packed screen
;-- SendScreen
                send SPACKED            ; send screen
                .dataseg
                mov di,EndScreen
                mov si,offset Screen
SendScreen01:
                lodsb
                send
                cmp si,di
                jb SendScreen01
                send 0ffh
                send 0ffh
;--SendScreen
                pop di
                pop si
                pop es
                pop dx
                pop cx
```

```
                ret
ScreenManager   endp
;
;[]-------------------------------------------------------[]
; table of routines within communicate interrupt
;[]-------------------------------------------------------[]
ComISRtbl       label   word
                dw offset DoNothing                     ;80h login
                dw offset DoNothing                     ;81h logout
                dw offset MsgFromServer                 ;82h msg
                dw offset MsgFromClient                 ;83h msg
                dw offset File2server                   ;84h
                dw offset File2Client                   ;85h
                dw offset FileFromServer                ;86h
                dw offset FileFromClient                ;87h
                dw offset RemoteRun                     ;88h
                dw offset RemoteKey                     ;89h
                dw offset RemoteON                      ;8Ah
                dw offset RemoteOFF                     ;8Bh
EndComtbl       equ     $-ComISRtbl
;
;[]-------------------------------------------------------[]
;dispatch routine - al=code,dx=_baseaddr
;[]-------------------------------------------------------[]
ComISR          proc    near
                sub al,80h
                jc  comisr00
                xor bx,bx
                mov bl,al
                shl bx,1
                cmp bx,EndComtbl
                jb  comisr01
comisr00:
                ret
comisr01:
                jmp word ptr [bx+offset ComISRtbl]
ComISR          endp
;
;[]-------------------------------------------------------[]
;Routine support communication ISR
;[]-------------------------------------------------------[]
DoNothing       proc    near
                ret
DoNothing       endp
;
headmsg         db '->> From ['         ; 10
station         db '1] : '              ; 5
MsgBuffer       db 65 dup (0)           ; 65
line24          dw 80 dup (0)
;
MsgFromServer   proc    near
                mov ax,cs
                mov es,ax
                assume es:code
                mov di,offset MsgBuffer
                cld
MsgFromServer01:                                        ; receive msg
                getcom
                stosb
                or al,al
                jnz MsgFromServer01
MsgFromServer02:
                xsend ACK                               ; acknowledge
                mov station,'1'
                call ShowMsg
                ret
MsgFromServer   endp
;
```

```
MsgFromClient   proc    near
                mov ax,cs
                mov es,ax
                assume es:code
                mov di,offset MsgBuffer
                cld
MsgFromClient01:                                ; receive msg
                getcom
                stosb
                or al,al
                jnz MsgFromClient01
MsgFromClient02:
                xsend ACK                       ; acknowledge
                mov station,'2'
                call ShowMsg
                ret
MsgFromClient   endp
;
; show msg on own screen
ShowMsg         proc    near
                mov di,offset line24            ; save line 24
                mov si,row24
                mov ax,_videoaddr
                mov ds,ax
                assume ds:nothing
                mov cx,80
                rep movsw
                mov es,ax
                assume es:nothing
                mov ax,cs
                mov ds,ax
                assume ds:code
                mov ah,03h                      ; read cursor position
                xor bx,bx
                int 10h
                push dx                         ; save cursor
                mov ah,02h                      ; set cursor hide
                mov dx,1900h                    ; to row 25
                int 10h
                mov di,row24
                mov si,offset headmsg
                mov ah,70h
                mov cx,80
msg2S03:                                        ; show msg
                lodsb
                or al,al
                jz msg2S04
                stosw
                loop msg2S03
msg2S04:
                mov al,20h
                rep stosw
                beep
                readkey                         ; wait anykey
                mov di,row24                    ; restore line 24
                mov si,offset line24
                mov cx,80
                rep movsw
                pop dx                          ; restore cursor
                mov ah,02h
                xor bx,bx
                int 10h
                ret
ShowMsg         endp
;
File2Server     proc    near
                mov si,offset filespec
ffs01:
```

```
                getcom
                mov [si],al
                inc si
                or al,al
                jnz ffs01
                fopen filespec,0
                jc ffsErr01
                mov _Filehandle,ax
                send ACK
                call SendFile
                jnc  ffs02
                send CAN
ffs02:
                fclose
                jmp short ffsEnd
ffsErr01:
                send NAK
ffsEnd:
                ret
File2Server     endp
;
File2Client     proc    near
                ret
File2Client     endp
;
FileFromServer  proc    near
                xsend ACK
                call ReceiveFile
                ret
FileFromServer  endp
;
FileFromClient  proc    near
                ret
FileFromClient  endp
;
;[]------------------------------------------------------[]
; Receive file from server
;[]------------------------------------------------------[]
ReceiveFile     proc    near
                cli
RecFile01:
                receive
                cmp al,SOH
                jne RecFile02
                mov si,offset filespec
                call RecString
                jnz RecNAK
                fcreate filespec,0          ; normal
                jc RecCANCEL
                mov _Filehandle,ax
RecACK:
                send ACK
                jmp short RecFile01
RecNAK:
                send NAK
                jmp  RecFile01
RecCANCEL:
                send CAN
                stc
                jmp RecFile06
RecFile02:
                cmp al,STX
                jne RecFile03
                call ReceiveBlk
                jnz RecNAK
                fwrite CopyBuffer,1024
                jc RecError01
                jmp RecACK
```

```
RecFile03:
                cmp al,ETX
                jne RecFile04
                call ReceiveByCX
                jnz RecNAK
                or cx,cx
                jz RecFile05
                fwrite CopyBuffer,cx
                jnc RecFile05
RecError01:
                send CAN
                jmp short RecError02
RecFile04:
                cmp al,CAN
                je  RecError02
                jmp RecFile01
RecError02:
                fclose
                stc
                jmp short RecFile06
RecFile05:
                send ACK
                fclose
                clc
RecFile06:
                sti
                ret
ReceiveFile     endp
;
; si -> buffer
RecString       proc    near
                push si
                xor bx,bx
RecString01:
                receive
                add bl,al
                mov [si],al
                inc si
                or al,al
                jnz RecString01
                receive
                cmp bl,al
                pop si
                ret
RecString       endp
;
ReceiveBlk      proc    near
                mov si,offset CopyBuffer
                xor bx,bx
                mov cx,1024
ReceiveBlk01:
                receive
                add bl,al
                mov [si],al
                inc si
                loop ReceiveBlk01
                receive                 ; bl=own chksum, al=remote chk
                cmp al,bl               ; return ZF flag
                ret
ReceiveBlk      endp
;
ReceiveByCX     proc    near
                receive
                mov ch,al
                receive
                mov cl,al
                or cx,cx
                jz ReceiveByCX02
```

```
                push cx                 ; save total byte to receive
                mov si,offset CopyBuffer
                xor bx,bx
ReceiveByCX01:
                receive
                add bl,al
                mov [si],al
                inc si
                loop ReceiveByCX01
                pop cx                  ; restore total Byte received
                receive
                cmp al,bl
ReceiveByCX02:
                ret
ReceiveByCX     endp
;
;[]------------------------------------------------------[]
; Transmit file to client
;[]------------------------------------------------------[]
SendFile        proc    near
SendFile01:
                send SOH
                receive
                cmp al,NAK
                je SendFile01
                cmp al,CAN
                je SendError02
                cmp al,ACK
                je SendFile02
SendError01:
                send CAN
SendError02:
                fclose
                stc
                jmp SendFile07
SendFile02:
                ifnotkeypressed SendFile03
                ife_esc SendError01
SendFile03:
                fread CopyBuffer,1024
                jc SendError01
                cmp ax,1024
                jb SendFile04
SendFile1024:
                call SendBlk                    ; STX
                receive
                cmp al,NAK
                je SendFile1024
                cmp al,CAN
                je SendError02
                cmp al,ACK
                jne SendError01
                jmp short SendFile02
SendFile04:
                mov cx,ax
SendFile05:
                call SendByCX           ; CX not changed
                receive
                cmp al,NAK
                je SendFile05
                cmp al,CAN
                jne SendFile06
                jmp SendError02
SendFile06:
                fclose
                clc
SendFile07:
                ret
```

```
SendFile        endp
;
SendBlk         proc    near
                send STX
                mov si,offset CopyBuffer
                xor bx,bx               ; clear
                mov cx,1024
SendBlk01:
                lodsb                   ; get data
                mov ah,al               ; save data
                add bl,al               ;
                send ah                 ;
                loop  SendBlk01
                send bl
                ret
SendBlk         endp
;
SendByCx        proc    near
                send ETX
                send ch                 ; size in dx
                send cl
                or cx,cx
                jz SendByCX02
                push cx
                mov si,offset CopyBuffer
                xor bx,bx
SendByCX01:
                lodsb                   ; get data
                mov ah,al               ; save data
                add bl,al               ;
                send ah                 ;
                loop  SendByCX01
                send bl
                pop cx
SendByCX02:
                ret
SendByCX        endp
;
RemoteRun       proc    near
                ret
RemoteRun       endp
;
RemoteKey       proc    near            ; dx -> _baseaddr
                cli
                .biosseg
                mov bx,_tail
host01:
                getcom                  ; get key
                mov [bx],al             ; put key
                inc bx                  ; adjust pointer
                getcom
                mov [bx],al
                inc bx                  ; advance tail pointer (round
                cmp bx,offset _endbuffer;
                jne host02              ;
                mov bx,offset _kbbuffer ;
host02:
                mov _tail,bx            ;
endhost:
                ret
RemoteKey       endp
;
RemoteON        proc    near
                cli
                mov byte ptr SCREENON,TRUE
                ret
RemoteON        endp
;
```

```
RemoteOFF       proc    near
                mov byte ptr SCREENON,FALSE
                ret
RemoteOFF       endp
;
;[]------------ INSTALL SECTION -----------------------------------
;[]-------------------------------------------------------[]
; install TSR  ds -> cs ->es -> data
;[]-------------------------------------------------------[]
install         proc    near
                mov TSRPSP,es           ; save TSR PSP
                printstr header
                call initparameter
                mov ax,3562h            ; get func. 62h
                int 21h                 ;
                cmp bx,offset int62h    ; is own interrupt routine ?
                jne install1
                mov dx,offset alreadylogin
install0:
                mov ah,9
                int 21h
                mov ax,4c00h            ; terminate
                int 21h                 ;
install1:
; save address of old int 62h
                mov word ptr oldint62h,bx   ;
                mov word ptr oldint62h+2,es ;
; send user name to registation
                commOFF
                call commset
                call userlogin
                jc install0
; install int. 62h (Network service routine define)
                cli
                mov ax,2562h            ; set interrupt vector 62h
                mov dx,offset int62h    ;
                int 21h                 ;
; Get InDOS address from MS-DOS
                mov ah,34h              ;
                int 21h                 ; es:bx -> InDOS
                mov word ptr InDOSAddr,bx  ;
                mov word ptr InDOSAddr+2,es;
; install int. 13h handler ( disk i/o )
                mov ax,3513h            ; get interrupt vector 13h
                int 21h                 ;
                mov word ptr oldint13h,bx   ;
                mov word ptr oldint13h+2,es ;
                mov ax,2513h            ; set interrupt vector 13h
                mov dx,offset int13h    ;
                int 21h                 ;
; install int. 1ch handler ( Timer tick )
                mov ax,351ch            ; get 1ch
                int 21h                 ;
                mov word ptr oldint1ch,bx  ;
                mov word ptr oldint1ch+2,es;
                mov ax,251ch            ; set 1ch
                mov dx,offset int1ch    ;
                int 21h                 ;
; install int. 28h handler ( DOS Idle )
                mov ax,3528h            ; get interrupt vector 28h
                int 21h                 ;
                mov word ptr oldint28h,bx   ;
                mov word ptr oldint28h+2,es ;
                mov ax,2528h            ; set interrupt vector 28h
                mov dx,offset int28h    ;
                int 21h                 ;
; install int. communication handle (COM1/COM2)
                mov ah,35h
```

```
                mov al,_intcom
                int 21h                 ;
                mov word ptr oldintcomm,bx  ;
                mov word ptr oldintcomm+2,es;
                mov ah,25h
                mov al,_intcom
                mov dx,offset intcomm
                int 21h                 ;
; ready to make resident
                printstr install_ok
                call initvideoram
                commON
; Terminate and stay resident
                mov dx,offset install   ;
                int 27h                 ;
install         endp
;
;[]------------ DATA ----------------------------------------[]
header db '
       db '  PCNet V1.00 - Workstation Shell for MS-DOS User login
       db '  (C) Copyright 1992 PC-Network Project. All Rights Reserve
       db cr,lf,'$'
;
comportmsg      db 'Enter your COM port: $'
loginmsg        db 'Enter your login name: $'
alreadylogin    db '    PCNet Workstation login shell already exist.'
notready        db '    PCNet SERVER not found,check and try again!',
install_ok      db '    Attached to PCNet SERVER O.K.',cr,lf,'$'
newlinemsg      db cr,lf,'$'
;
comport         db 5
cport           db 0
                db 5 dup (0)
;
login_name      db 9
cname           db 0
                db 9 dup (0)
;
com1str         db 'com1',0
com2str         db 'com2',0
;
;[]-------------------------------------------------------[]
; set video ram buffer address
;[]-------------------------------------------------------[]
initvideoram    proc   near
                mov ah,0fh              ; get video mode
                int 10h                 ;
                cmp al,7                ; monochrome ?
                jz is001                ;
                mov _videoaddr,COLOR    ; no,set to video ram addr. of
is001:
                mov ax,_videoaddr       ;
                mov es,ax               ;
                ret                     ;
initvideoram    endp
;
; if identical then CF=0 and si = next parameter
; if CF set si = old parameter
CompareStr      proc    near
                push si
cmpstr01:
                lodsb                   ; ds:si
                or al,20h               ; make lower case
                cmp al,[di]
                jnz cmpstr02
                inc di
                cmp [di],byte ptr 0
```

```
                jnz cmpstr01
                pop ax                  ; pop old SI
                clc
                ret
cmpstr02:
                pop si
                stc
                ret
CompareStr      endp
;
initparameter   proc    near
                mov si,80h
                lodsb
                or  al,al
                jz  initpara_port
initpara01:
                lodsb
                cmp al,20h
                jz  initpara01
                dec si
                cmp byte ptr [si],'2'
                jz  init_port2
                cmp byte ptr [si],'1'
                jz  initpara02
                mov di,offset com1str
                call CompareStr
                jnc initpara02
                mov di,offset com2str
                call CompareStr
                jc initpara_port
init_port2:
                mov byte ptr _intcom,0bh
                mov word ptr _baseaddr,COM2ADDR
initpara02:
                lodsb
                cmp al,20h
                je  initpara02
                cmp al,':'
                je  initpara02
                cmp al,'/'
                je  initpara02
                cmp al,','
                je  initpara02
                cmp al,';'
                je  initpara02
                cmp al,0dh
                je  initpara_name
                dec si
                mov cx,8
                mov di,offset login_name+2
initpara03:
                lodsb
                cmp al,0dh
                je  initpara_password
                stosb
                inc byte ptr cs:cname
                loop initpara03
                jmp short initpara_password
initpara_port:
                call ask_port
initpara_name:
                call ask_name
initpara_password:
;                call ask_password
                ret
initparameter   endp
;
ask_port        proc    near
```

```
askport01:
                printstr comportmsg
                mov ah,0ah
                mov dx,offset comport
                int 21h
                printstr newlinemsg
                mov si,offset comport+2
                cmp byte ptr cport,1
                ja  askport02
                cmp byte ptr [si],'2'
                jz  askport03
                cmp byte ptr [si],'1'
                jz  askportEnd
                jmp short askport01
askport02:
                mov di,offset com1str
                call CompareStr
                jnc askportEnd
                mov di,offset com2str
                call CompareStr
                jc  askport01
askport03:
                mov byte ptr _intcom,0bh
                mov word ptr _baseaddr,COM2ADDR
askportEnd:
                ret
ask_port        endp
;
ask_name        proc    near
askname01:
                printstr loginmsg
                mov ah,0ah
                mov dx,offset login_name
                int 21h
                printstr newlinemsg
                mov si,offset cname
                lodsb
                or al,al
                jz askname01
                printstr newlinemsg
                ret
ask_name        endp
;
userlogin       proc    near
                xor cx,cx
                mov si,offset cname
                lodsb
                mov cl,al
                send 80h                        ; login code
sendname01:
                lodsb
                send
                loop sendname01
                xor ax,ax                       ; terminate_code for user name
                send
                mov bx,10
_getserver:
                mov cx,0ffffh
                mov dl,LSR
_getserver00:
                in al,dx
                jmp short $+2
                test al,1
                jnz  _getserver01
                loop _getserver00
                dec bx
                jnz _getserver
;
userloginExit:
```

```
                mov dx,offset notready
                stc
                ret
_getserver01:
                mov dl,RXBUF
                in al,dx
                jmp short $+2
                cmp al,ACK
                jne  userloginExit
                clc
                ret
userlogin       endp
;
;[]-------------------------------------------------[]
; Proc           COMMSet
; Descript       Initialize the serial port by setting 8250 ( UART )
;[]-------------------------------------------------[]
COMMSet         Proc    near
                cli                             ;
                mov dx,_baseaddr                ;
                inc dx                          ; point to interrupt enable re
                sub al,al                       ; reset interrupt enables
                out dx,al                       ;
;--Initialize the baud rate divisor registers for parameter (baud)
                add dx,2                        ; point to line control regist
                mov al,10000000b                ; turn on bit 7
                out dx,al                       ;
                dec dx                          ; point to buad rate divisor(h
                dec dx                          ;
                xor ax,ax                       ; divisor = 0
                out dx,al                       ;
                dec dx                          ; point to buad rate divisor(l
                mov al, Divisor                 ;
                out dx,al                       ;
;--Initialize the line control register
                add dx,3                        ; point to line control regist
                xor ax,ax                       ; turn off bit 7
                or al,_Dataformat               ; defalt=8 bit data,1 stop bit
                out dx,al                       ;
;--Initialize modem control register for enable interrupt
                inc dx                          ; point to modem control regis
                mov al,00001011b                ; turn on bit 3 (OUT2) for ena
                                                ; and turn on DTR(bit 0) ,RTS(
                out dx,al                       ;
;--Read to clear possible old interrupt
                inc dx                          ; point to line status reg.
                in al,dx                        ;
                jmp short $+2                   ;
                inc dx                          ; point to modem status reg.
                in al,dx                        ;
;--Initialize the interrupt enable register:
                sub dx,5                        ; point to interrupt enable re
                mov al,00000001b                ; enable data rec.
                out dx,al                       ;
;--Clear register
ClRgs:
                dec dx                          ;
                in al,dx                        ; clear possible old data
                sti
                ret
COMMSet         Endp
;
code            ends
                end     start
```

```
;               ************************************************
;               *                                              *
;               *        The RS-232 Network Logout V1.00       *
;               *                By   TEEKAYU C.               *
;               *                                              *
;               *  Copyright (c) 1992    R & D Computer System *
;               *    nlogout.asm       Last Update  06-Feb-92   *
;               ************************************************
;
;[]------------------------------------------------------------[]
include         d:\teekayu\asmplus\include\asmplus.oop
include         d:\teekayu\asmplus\include\constant.oop
include         d:\teekayu\asmplus\include\diskio.oop
include         d:\teekayu\asmplus\include\comm.oop
include         d:\teekayu\asmplus\include\screen.oop
include         d:\teekayu\asmplus\include\pcnet.oop
;[]------------------------------------------------------------[]
                .biosdata
;[]------------------------------------------------------------[]
code            segment byte public 'CODE'
                assume cs:code,ds:code
                org 100h
start:
                jmp install
;
Reference_point label   byte
;
;[]------------ DATA ----------------------------------------[]
header db '
       db '  PCNet V1.00 - Workstation user logout
       db '  (C) Copyright 1992 PC-Network Project. All Rights Reserve
       db '
       db cr,lf,'$'
;
notready        db '    PCNet Workstation shell does not exist.',be
ready           db '    PCNet Workstation logged out.',cr,lf,'$'
;
;[]------------ code ----------------------------------------[]
install:
                printstr header
                mov ax,3562h
                int 21h
                cmp bx,offset Reference_point
                je  main01
                printstr notready
                jmp short mainEnd
main01:
                mov ah,SENDCODE
                mov al,LOGOUT_CODE
                xor dx,dx                       ; default port
                int 62h
                mov ah,UNINSTALL_
                int 62h                         ; return  ax=TSRPSP
                call freemem
                printstr ready
mainEnd:
                terminate
;
;[]------------------------------------------------------------[]
; release memory
;[]------------------------------------------------------------[]
freemem         proc    near
                push es
                mov es,ax
                push es
                mov es,es:[2ch]
                mov ah,49h
                int 21h
```

```
                pop es
                mov ah,49h
                int 21h
                pop es
                ret
freemem         endp
;
code            ends
                end     start
```

```
;               ************************************************
;               *                                              *
;               *       The Network Send Messages V1.00        *
;               *                By   TEEKAYU C.               *
;               *                                              *
;               *  Copyright (c) 1992    R & D Computer System *
;               *    nsend.asm        Last Update  13-Feb-92    *
;               ************************************************
;
;[]-------------------------------------------------------------[]
SERVER          EQU     0
CLIENT          EQU     -1
;[]-------------------------------------------------------------[]
include         d:\teekayu\asmplus\include\asmplus.oop
include         d:\teekayu\asmplus\include\constant.oop
include         d:\teekayu\asmplus\include\diskio.oop
include         d:\teekayu\asmplus\include\comm.oop
include         d:\teekayu\asmplus\include\screen.oop
include         d:\teekayu\asmplus\include\pcnet.oop
;[]-------------------------------------------------------------[]
                .biosdata
;[]-------------------------------------------------------------[]
code            segment byte public 'CODE'
                assume cs:code,ds:code,es:code
                org 100h
start:
                jmp install
;[]-------------------------------------------------------------[]
Reference_point label   byte
;[]-------------------------------------------------------------[]
;[]------------ data -------------------------------------------[]
header db '
       db '  PCNet V1.00 - Send Message to Workstation
       db '  (C) Copyright 1992 PC-Network Project. All Rights Reserve
       db '
       db cr,lf,'$'
;
notready        db '      PCNet SERVER/Workstation shell does not exist
                db bell,cr,lf,'$'
;
Syntax          db 'Usage:',cr,lf
                db '        NSEND  <station>,"message"',cr,lf,lf
                db '    Where <station> : 0 = ALL',cr,lf
                db '                      1 = USER#1',cr,lf
                db '                      2 = USER#2',cr,lf,'$'
;
Ssyntax         db 'Usage:',cr,lf
                db '        NSEND "message"  [TO]  USER1/USER2/ALL',cr,
;
Wsyntax         db 'Usage:',cr,lf
                db '        NSEND "message"  [TO]  SERVER/ANOTHER/ALL',
;
send_err        db '        Message NOT sent to Workstation ',bell,'$'
;
send_ok         db '        Message sent to Workstation '
;
WhereCode       db '0'
                db cr,lf,'$'
;
_baseaddr       dw COM1ADDR
;
;[]------------ code -------------------------------------------[]
install:
                printstr header
                mov ax,3562h
                int 21h
                cmp bx,offset Reference_point
                je  main01
```

```
                mov dx,offset notready
                jmp short mainPrint
main01:
                mov dx,offset Syntax
                xor ax,ax
                mov si,80h
                lodsb
                or ax,ax
                jz mainPrint
main02:
                lodsb
                cmp al,20h
                je main02
                cmp al,'0'
                jb mainPrint
                cmp al,'2'
                ja mainPrint
                mov WhereCode,al
main03:
                lodsb
                cmp al,20h
                je main03
                cmp al,','
                je main03
                cmp al,':'
                je main03
                cmp al,';'
                je main03
                cmp al,'/'
                je main03
                cmp al,'"'
                jne mainPrint
                mov ah,03h              ; get stat
                int 62h
                cmp al,SERVER
                je  main04
                call SendToServer
                jmp short mainPrint
main04:
                call SendFromServer
mainPrint:
                mov ah,9
                int 21h
mainEnd:
                terminate
;
SendToServer    proc    near
                mov al,WhereCode
                add al,(NSENDALL_CODE - '0')
                mov ah,01h
                int 62h
                mov  baseaddr,dx
                cmp byte ptr WhereCode,'0'
                jne sts01
                call SendALL
                ret
sts01:
                call SendMsg
                ret
SendToServer    endp
;
SendALL         proc    near
                cli
                mov cx,65
sendall01:
                lodsb
                cmp al,'"'
                je sendall02
```

```
                cmp al,0dh
                je sendall02
                xsend
                loop sendall01
sendall02:
                xor ax,ax                               ; send 0
                xsend
                mov byte ptr WhereCode,'1'
                call GetResponse
                jc sendall04
                cmp al,ACK
                je sendall03
                cmp al,NAK
                jne sendall04
                printstr send_ok
                jmp short sendall05
sendall03:
                printstr send_ok
                mov byte ptr WhereCode,'2'
                mov dx,offset send_ok
                clc
                jmp short sendall06
sendall04:
                printstr send_err
                printstr WhereCode
sendall05:
                mov byte ptr WhereCode,'2'
                printstr send_err
                mov dx,offset WhereCode
                stc
sendall06:
                sti
                ret
SendALL         endp
;
SendMsg         proc    near
                cli
                mov cx,65
sendmsg01:
                lodsb
                cmp al,'"'
                je sendmsg02
                cmp al,0dh
                je sendmsg02
                xsend
                loop sendmsg01
sendmsg02:
                xor ax,ax                               ; send 0
                xsend
                call GetResponse
                sti
                jnc sendmsg03
sendmsgExit:
                printstr send_err
                mov dx,offset WhereCode
                stc
                ret
sendmsg03:
                cmp al,ACK
                jne sendmsgExit
                mov dx,offset send_ok
                clc
                ret
SendMsg         endp
;
SendFromServer  proc    near
                mov dx,COM1ADDR
                mov al,WhereCode
```

```
                cmp al,'1'
                je  sfs01
                cmp al,'2'
                je  sfs00
                call SendMsgALL                 ; dx = com1addr
                ret
sfs00:
                mov dx,COM2ADDR
sfs01:
                mov _baseaddr,dx
                xsend 82h                       ; msg code
                call SendMsg
                ret
SendFromServer  endp
;
SendMsgALL      proc    near
                cli
                xsend 82h
                xor dh,1
                xsend 82h
                xor dh,1
                mov cx,65
sendmsgall01:
                lodsb
                cmp al,'"'
                je sendmsgall02
                cmp al,0dh
                je sendmsgall02
                xsend
                xor dh,1
                xsend
                xor dh,1
                loop sendmsgall01
sendmsgall02:
                xor ax,ax                       ; send 0
                xsend
                xor dh,1
                xsend
                xor dh,1
                xor si,si                       ; initial mark com1
                xor di,di                       ; initial mark com2
                cli
                mov bx,5
GetResponseAll1:
                mov cx,0ffffh
                mov dl,LSR
GetResponseAll2:
                or  si,si
                jz  GetResponseAll4
                or  di,di
                jnz GetResponseAll5
GetResponseAll4:
                in  al,dx
                jmp short $+2
                test al,1
                jz  GetResponseAll3
                mov si,0fh                      ; mark com1 o.k.
                mov dx,COM1ADDR
                in al,dx
                mov dl,LSR
                cmp al,ACK
                je  GetResponseAll3
                inc si                          ; mark com1 not o.k.
GetResponseAll3:
                xor dh,1
                in al,dx
                xor dh,1
                test al,1
```

```
                jz  GetResponseAll6
                mov di,0fh                      ; mark com2 o.k.
                mov dx,COM2ADDR
                in al,dx
                mov dx,COM1ADDR
                cmp al,ACK
                je  GetResponseAll6
                inc di                          ; mark com2 not o.k.
GetResponseAll6:
                loop GetResponseAll2
                dec bx
                jnz  GetResponseAll1
GetResponseAll5:
                mov byte ptr WhereCode,'1'
                cmp si,0fh
                je  sendmsgall03
                printstr send_err
                printstr WhereCode
                jmp short sendmsgall04
sendmsgall03:
                printstr send_ok
sendmsgall04:
                mov byte ptr WhereCode,'2'
                cmp di,0fh
                je  sendmsgall05
                printstr send_err
                mov dx,offset WhereCode
                stc
                jmp short sendmsgall06
sendmsgall05:
                mov dx,offset send_ok
                clc
sendmsgall06:
                sti
                ret
SendMsgALL      endp
;
GetResponse     proc    near
                mov bx,5
GetResponse0:
                mov cx,0ffffh
                mov dl,LSR
GetResponse1:
                in al,dx
                jmp short $+2
                test al,1
                jnz  GetResponse2
                loop GetResponse1
                dec bx
                jnz GetResponse0
                mov dl,RXBUF
                stc
                ret
GetResponse2:
                mov dl,RXBUF
                in al,dx
                clc
                ret
GetResponse     endp
;
code            ends
                end     start
```

```
;               ************************************************
;               *                                              *
;               *      The Network Files Transfer Manager      *
;               *                  By   TEEKAYU C.             *
;               *                                              *
;               *  Copyright (c) 1992    R & D Computer System *
;               *    ncopy.asm         Last Update  17-Feb-92   *
;               ************************************************
;
;[]-------------------------------------------------------------[]
SERVER          EQU     0
CLIENT          EQU     -1
_TO_            EQU     0
_FROM_          EQU     1
;[]-------------------------------------------------------------[]
include         d:\teekayu\asmplus\include\asmplus.oop
include         d:\teekayu\asmplus\include\constant.oop
include         d:\teekayu\asmplus\include\diskio.oop
include         d:\teekayu\asmplus\include\comm.oop
include         d:\teekayu\asmplus\include\screen.oop
include         d:\teekayu\asmplus\include\pcnet.oop
;[]-------------------------------------------------------------[]
                .biosdata
;[]-------------------------------------------------------------[]
code            segment byte public 'CODE'
                assume cs:code,ds:code,es:code
                org 100h
start:
                jmp install
;[]-------------------------------------------------------------[]
Reference_point label   byte
;[]-------------------------------------------------------------[]
;[]------------ data -------------------------------------------[]
EXTRN   _Filehandle:word
;
header db '
       db '  PCNet V1.00 - Files Transfer from/to Workstation
       db '  (C) Copyright 1992 PC-Network Project. All Rights Reserve
       db '
       db cr,lf,'$'
;
notready        db '      PCNet SERVER/Workstation shell does not exist
                db bell,cr,lf,'$'
;
Syntax          db 'Usage:',cr,lf
                db '       NCOPY  filename  FROM/TO  station',cr,lf,lf
                db '    Where <station> : 1 = USER#1',cr,lf
                db '                      2 = USER#2',cr,lf,'$'
;
TOstr           db 'to',0
FROMstr         db 'from',0
Direction       db  TO
WhereCode       db 0
 baseaddr       dw COM1ADDR
filespec        db 128 dup (0)
fnameptr        dw offset filespec
filenotfound    db '      File not found !',bell,cr,lf,'$'
readfileerror   db '      Read file error !',bell,cr,lf,'$'
notresponse     db '      Not response from Workstation !',bell,cr,lf,
transmiterr     db '      Transmitting file error !',bell,cr,lf,'$'
okmsg           db '      Transfer completely ',cr,lf,'$'
CopyBuffer      db 1024 dup (0)
;
;[]------------ code -------------------------------------------[]
install:
                commOFF
                setDTA  DTAbuffer
                printstr header
```

```
                mov ax,3562h
                int 21h
                cmp bx,offset Reference_point
                je  main01
                mov dx,offset notready
                jmp short mainPrint
main01:
                mov dx,offset Syntax
                xor ax,ax
                mov si,80h
                cld
                lodsb
                or ax,ax
                jz mainPrint
main02:
                lodsb
                cmp al,20h
                je main02
                dec si
                call DoCommand
mainPrint:
                mov ah,9
                int 21h
                commON
                terminate
;
; get fname from command line : return al=next char
DoCommand       proc    near
                mov ax,cs
                mov es,ax
                assume es:code
                mov di,offset filespec
docmd01:
                lodsb
                cmp al,20h
                je docmd02
                cmp al,0dh
                je docmdEnd
                cmp al,'\'
                jne docmd011
                inc di
                mov fnameptr,di
                dec di
docmd011:
                stosb
                jmp short docmd01
docmd02:
                lodsb
                cmp al,20h
                je docmd02
                cmp al,0dh
                je docmdEnd
                dec si
                mov di,offset TOstr
                call CompareStr
                jnc docmd03
                mov di,offset FROMstr
                call CompareStr
                jc docmdEnd
                mov byte ptr Direction,_FROM_
docmd03:
                lodsb
                cmp al,20h
                je docmd03
                cmp al,'1'
                jb docmdEnd
                cmp al,'2'
                ja docmdEnd
```

```
                mov WhereCode,al
docmd04:
                lodsb
                cmp al,20h
                je docmd04
                cmp al,0dh
                jne docmdEnd
                call Ncopy
docmdEnd:
                ret
DoCommand       endp
;
; input - si->source, di->CONSTANT str
; if identical then CF=0 and si = next parameter
; if CF set si = old parameter
CompareStr      proc    near
                push si
cmpstr01:
                lodsb                           ; ds:si
                or al,20h                       ; make lower case
                cmp al,[di]
                jnz cmpstr02
                inc di
                cmp [di],byte ptr 0
                jnz cmpstr01
                pop ax                          ; pop old SI
                clc
                ret
cmpstr02:
                pop si
                stc
                ret
CompareStr      endp
;
GetResponse     proc    near
                push dx
                mov bx,5
GetResponse0:
                mov cx,0ffffh
                mov dl,LSR
GetResponse1:
                in al,dx
                jmp short $+2
                test al,1
                jnz GetResponse2
                loop GetResponse1
                dec bx
                jnz GetResponse0
                stc
                jmp short GetResponse3
GetResponse2:
                mov dl,RXBUF
                in al,dx
                clc
GetResponse3:
                pop dx
                ret
GetResponse     endp
;
Ncopy           proc    near
                mov ah,03h                      ; get stat
                int 62h
                mov dx,offset Syntax
                cmp al,SERVER
                jne ncopy01
                call ServerNcopy
                jmp short ncopyEnd
ncopy01:
```

```
                call ClientNcopy
ncopyEnd:
                ret
Ncopy           endp
;
ServerNcopy     proc    near
                cmp byte ptr Direction,_FROM_
                je  serverncopy01
                call CopyToClient
                jmp short serverncopyEnd
serverncopy01:
                call CopyFromClient
serverncopyEnd:
                ret
ServerNcopy     endp
;
ClientNcopy     proc    near
                cmp byte ptr Direction,_FROM_
                je  clientncopy01
                call CopyToServer
                jmp short clientncopyEnd
clientncopy01:
                call CopyFromServer
clientncopyEnd:
                ret
ClientNcopy     endp
;
CopyToServer    proc    near
                findfirst filespec
                jc  ctsErr01
                fopen filespec,0
                jc  ctsErr02
                mov _Filehandle,ax
                mov al,WhereCode
                add al,(87h - '1')              ; 87h - code File2Serv
                mov ah,01h                      ; send byte
                int 62h
                mov _baseaddr,dx
                call GetResponse
                jc  ctsErr03
                cmp al,ACK
                jne ctsErr01
                call SendFile
                jnc cts01
                mov dx,offset transmiterr
                jmp short ctsEnd
ctsErr01:
                mov dx,offset filenotfound
                jmp short ctsEnd
ctsErr02:
                mov dx,offset readfileerror
                jmp short ctsEnd
ctsErr03:
                mov dx,offset notresponse
                jmp short ctsEnd
cts01:
                mov dx,offset okmsg
ctsEnd:
                ret
CopyToServer    endp
;
SendFile        proc    near
SendFile01:
                call SendFname          ; si -> filename
                receive
                cmp al,NAK
                je SendFile01
                cmp al,CAN
```

```
                je SendError02
                cmp al,ACK
                je SendFile02
SendError01:
                send CAN
SendError02:
                fclose
                stc
                jmp SendFile07
SendFile02:
                ifnotkeypressed SendFile03
                ife_esc SendError01
SendFile03:
                fread CopyBuffer,1024
                jc SendError01
                cmp ax,1024
                jb SendFile04
SendFile1024:
                call SendBlk                    ; STX
                receive
                cmp al,NAK
                je SendFile1024
                cmp al,CAN
                je SendError02
                cmp al,ACK
                jne SendError01
                jmp short SendFile02
SendFile04:
                mov cx,ax
SendFile05:
                call SendByCX           ; CX not changed
                receive
                cmp al,NAK
                je SendFile05
                cmp al,CAN
                jne SendFile06
                jmp SendError02
SendFile06:
                fclose
                clc
SendFile07:
                ret
SendFile        endp
;
;-----------------------------------------------
; Proc          SendFname
; Entry         lea si,buffer
;-----------------------------------------------
SendFname       Proc    near
                send SOH
                mov si,offset _filename
                xor bx,bx
SM001:
                lodsb                   ; get data
                mov ah,al
                add bl,al
                send ah
                or ah,ah
                jnz SM001
                send bl
                ret
SendFname       Endp
;
SendBlk         proc    near
                send STX
                mov si,offset CopyBuffer
                xor bx,bx               ; clear
                mov cx,1024
```

```
SendBlk01:
                lodsb                           ; get data
                mov ah,al                       ; save data
                add bl,al                       ;
                send ah                         ;
                loop  SendBlk01
                send bl
                ret
SendBlk         endp
;
SendByCx        proc    near
                send ETX
                send ch                         ; size in dx
                send cl
                or cx,cx
                jz SendByCX02
                push cx
                mov si,offset CopyBuffer
                xor bx,bx
SendByCX01:
                lodsb                           ; get data
                mov ah,al                       ; save data
                add bl,al                       ;
                send ah                         ;
                loop  SendByCX01
                send bl
                pop cx
SendByCX02:
                ret
SendByCX        endp
;
CopyFromServer  proc    near
                mov al,WhereCode
                add al,(89h - '1')                      ; 89h - code FileFromS
                mov ah,01h
                int 62h
                mov _baseaddr,dx
                mov si,offset filespec
cfs01:
                lodsb
                xsend
                or al,al
                jnz cfs01
                call GetResponse
                jc  cfsErr01
                cmp al,NAK
                je  cfsErr02
                cmp al,ACK
                jne cfsErr03
                call ReceiveFile
                jnc cfs02
cfsErr03:
                mov dx,offset transmiterr
                jmp short cfsEnd
cfsErr02:
                mov dx,offset filenotfound
                jmp short cfsEnd
cfsErr01:
                mov dx,offset notresponse
                jmp short cfsEnd
cfs02:
                mov dx,offset okmsg
cfsEnd:
                ret
CopyFromServer  endp
;
ReceiveFile     proc    near
                cli
```

```
RecFile01:
                receive
                cmp al,SOH
                jne RecFile02
                mov dx,fnameptr
                xor cx,cx
                mov ah,3ch                          ; create
                int 21h
                jc RecCANCEL
                mov _Filehandle,ax
RecACK:
                send ACK
                jmp short RecFile01
RecNAK:
                send NAK
                jmp  RecFile01
RecCANCEL:
                send CAN
                stc
                jmp RecFile06
RecFile02:
                cmp al,STX
                jne RecFile03
                call ReceiveBlk
                jnz RecNAK
                fwrite CopyBuffer,1024
                jc RecError01
                jmp RecACK
RecFile03:
                cmp al,ETX
                jne RecFile04
                call ReceiveByCX
                jnz RecNAK
                or cx,cx
                jz RecFile05
                fwrite CopyBuffer,cx
                jnc RecFile05
RecError01:
                send CAN
                jmp short RecError02
RecFile04:
                cmp al,CAN
                je  RecError02
                jmp RecFile01
RecError02:
                fclose
                stc
                jmp short RecFile06
RecFile05:
                send ACK
                fclose
                clc
RecFile06:
                sti
                ret
ReceiveFile     endp
;
ReceiveBlk      proc    near
                mov si,offset CopyBuffer
                xor bx,bx
                mov cx,1024
ReceiveBlk01:
                receive
                add bl,al
                mov [si],al
                inc si
                loop ReceiveBlk01
                receive                        ; bl=own chksum, al=remote chk
```

```
                cmp al,bl               ; return ZF flag
                ret
ReceiveBlk      endp
;
ReceiveByCX     proc    near
                receive
                mov ch,al
                receive
                mov cl,al
                or cx,cx
                jz ReceiveByCX02
                push cx                 ; save total byte to receive
                mov si,offset CopyBuffer
                xor bx,bx
ReceiveByCX01:
                receive
                add bl,al
                mov [si],al
                inc si
                loop ReceiveByCX01
                pop cx                  ; restore total Byte received
                receive
                cmp al,bl
ReceiveByCX02:
                ret
ReceiveByCX     endp
;
CopyToClient    proc    near
                findfirst filespec
                jc  ctcErr01
                fopen filespec,0
                jc  ctcErr02
                mov  Filehandle,ax
                mov dx,COM1ADDR
                mov al,WhereCode
                cmp al,'2'
                jne ctc00
                mov dx,COM2ADDR
ctc00:
                mov  baseaddr,dx
                xsend 86h                       ; code copyto to login
                call GetResponse
                jc  ctcErr03
                cmp al,ACK
                jne ctcErr01
                call SendFile
                jnc ctc01
                mov dx,offset transmiterr
                jmp short ctcEnd
ctcErr01:
                mov dx,offset filenotfound
                jmp short ctcEnd
ctcErr02:
                mov dx,offset readfileerror
                jmp short ctcEnd
ctcErr03:
                mov dx,offset notresponse
                jmp short ctcEnd
ctc01:
                mov dx,offset okmsg
ctcEnd:
                ret
CopyToClient    endp
;
CopyFromClient  proc    near
                mov dx,COM1ADDR
                mov al,WhereCode
                cmp al,'2'
```

```
                jne cfc00
                mov dx,COM2ADDR
cfc00:
                mov _baseaddr,dx
                xsend 84h                              ; code copyfrom to log
                mov si,offset filespec
cfc01:
                lodsb
                xsend
                or al,al
                jnz cfc01
                call GetResponse
                jc  cfcErr01
                cmp al,NAK
                je  cfcErr02
                cmp al,ACK
                jne cfcErr03
                call ReceiveFile
                jnc cfc02
cfcErr03:
                mov dx,offset transmiterr
                jmp short cfcEnd
cfcErr02:
                mov dx,offset filenotfound
                jmp short cfcEnd
cfcErr01:
                mov dx,offset notresponse
                jmp short cfcEnd
cfc02:
                mov dx,offset okmsg
cfcEnd:
                ret
CopyFromClient  endp
;
code            ends
                end     start
```

```
;                ************************************************
;                *                                              *
;                *       The Network File Execute Manager       *
;                *                  By   TEEKAYU C.             *
;                *                                              *
;                *  Copyright (c) 1992    R & D Computer System  *
;                *    nrun.asm       Last Update  Wed 26Feb1992  *
;                ************************************************
;
;[]-------------------------------------------------------------[]
SERVER          EQU     0
CLIENT          EQU     -1
;[]-------------------------------------------------------------[]
include         d:\teekayu\asmplus\include\asmplus.oop
include         d:\teekayu\asmplus\include\constant.oop
include         d:\teekayu\asmplus\include\diskio.oop
include         d:\teekayu\asmplus\include\comm.oop
include         d:\teekayu\asmplus\include\screen.oop
include         d:\teekayu\asmplus\include\pcnet.oop
include         d:\teekayu\asmplus\include\exe.h
;[]-------------------------------------------------------------[]
                .biosdata
;[]-------------------------------------------------------------[]
code            segment byte public 'CODE'
                assume cs:code,ds:code,es:code,ss:code
                org 100h
start:
                jmp install
;[]-------------------------------------------------------------[]
Reference_point label   byte
;[]-------------------------------------------------------------[]
;[]------------ DATA -------------------------------------------[]
EXTRN   _Filehandle:word
;
header  db '
        db '  PCNet V1.00 - Load and Run remote file on Workstation
        db '  (C) Copyright 1992 PC-Network Project. All Rights Reserv
        db '
        db cr,lf,'$'
;
msg0    db 'Usage:',cr,lf,'         NRUN  [port,] filename',cr,lf,lf,'$'
msg1    db '    PCNet SERVER/Workstation shell does not exist !',bell
msg2    db '     Not response from Workstation !',bell,cr,lf,'$'
msg3    db '     File access error !',bell,cr,lf,'$'
msg4    db '     File not found !',bell,cr,lf,'$'
msg5    db '     Path not found !',bell,cr,lf,'$'
msg6    db '     Transmit file completely',cr,lf,'$'
msg7    db '     Remote Workstation not ready !',bell,cr,lf,'$'
msg8    db '     Not enough memory !',bell,cr,lf,'$'
msg9    db '     Can not run this file !',bell,cr,lf,'$'
msg10   db '     Memory resize failed !',bell,cr,lf,'$'
msg11   db '     .EXE file type',cr,lf,'$'
msg12   db '     .COM file type',cr,lf,'$'
msg13   db '     Allocate memory error !',bell,cr,lf,'$'
;
msgtbl          dw msg0
                dw msg1
                dw msg2
                dw msg3
                dw msg4
                dw msg5
                dw msg6
                dw msg7
                dw msg8
                dw msg9
                dw msg10
                dw msg11
                dw msg12
```

```
                dw msg13
;
com1msg         db 'com1',0
com2msg         db 'com2',0
_baseaddr       dw COM1ADDR
CommandLength   db 0
CommandLine     db 128 dup (0)
filespec        db 128 dup (0)
;
oldint22h       dd ?
UserIP          dw 100h
XUserSeg        dw 0
UserProgSeg     dw 0
UserCodeSeg     dw 0
MemSize         dw 0ffffh
SPsave          dw ?
RegBX           dw 0
RegCX           dw 0
RegCS           dw 0
RegIP           dw 0
RegSS           dw 0
RegSP           dw 0
;
;[]------------ CODE ----------------------------------------[]
install:
                cli
                mov sp,offset TopOfStack
                printstr header
                mov bx,offset TopOfStack
                mov cl,4
                shr bx,cl
                inc bx
                mov ah,4ah
                int 21h
                jnc main00
                mov al,10
                jmp short mainPrint
main00:
                commOFF
                sti
                mov ax,3562h
                int 21h
                cmp bx,offset Reference_point
                je  main01
                mov al,1
                jmp short mainPrint
main01:
                xor ax,ax
                mov si,80h
                cld
                lodsb
                or ax,ax
                jz mainPrint
main02:
                lodsb
                cmp al,20h
                je main02
                dec si
                call DoCommand
mainPrint:
                call PrintMsg
                call Freemem
                commON
                terminate
;
PrintMsg        proc  near
                push bx
                push dx
```

```
                mov bx,offset msgtbl
                xor ah,ah
                shl ax,1
                add bx,ax
                mov dx,[bx]
                mov ah,09h
                int 21h
                pop dx
                pop bx
                ret
PrintMsg        endp
;
; input - si->source, di->CONSTANT str
; if identical then CF=0 and si = next parameter
; if CF set si = old parameter
CompareStr      proc    near
                push si
cmpstr01:
                lodsb                           ; ds:si
                or al,20h                       ; make lower case
                cmp al,[di]
                jnz cmpstr02
                inc di
                cmp [di],byte ptr 0
                jnz cmpstr01
                pop ax                          ; pop old SI
                clc
                ret
cmpstr02:
                pop si
                stc
                ret
CompareStr      endp
;
GetResponse     proc    near
                push dx
                mov bx,5
GetResponse0:
                mov cx,0ffffh
                mov dl,LSR
GetResponse1:
                in al,dx
                jmp short $+2
                test al,1
                jnz GetResponse2
                loop GetResponse1
                dec bx
                jnz GetResponse0
                mov al,02h                      ; not response from remote
                stc
                jmp short GetResponse3
GetResponse2:
                mov dl,RXBUF
                in al,dx
                clc
GetResponse3:
                pop dx
                ret
GetResponse     endp
;
; get fname from command line : return al=next char
DoCommand       proc    near
                mov ax,cs
                mov es,ax
                assume es:code
                mov di,offset com1msg
                call CompareStr
                jnc docmd_port1
```

```
                mov di,offset com2msg
                call CompareStr
                jnc docmd_port2
                lodsb
                cmp al,'1'
                je  docmd_port1
                cmp al,'2'
                jne  docmd00
docmd_port2:
                mov _baseaddr,COM2ADDR
docmd_port1:
                lodsb
                cmp al,20h
                je docmd_port1
                cmp al,','
                je docmd_port1
                cmp al,':'
                je docmd_port1
                cmp al,0dh
                jne docmd00
                xor ax,ax
                jmp short docmdEnd
docmd00:
                dec si
                mov di,offset filespec
docmd01:
                lodsb
                cmp al,20h
                je  docmd02
                cmp al,0dh
                je  docmd02
                stosb
                jmp short docmd01
docmd02:
                mov di,offset CommandLine
docmd021:
                stosb
                cmp al,0dh
                je  docmd03
                inc byte ptr CommandLength
                lodsb
                jmp short docmd021
docmd03:
                call NetworkRun
docmdEnd:
                ret
DoCommand       endp
;
SendFname       proc    near
                mov si,offset filespec
sfname01:
                lodsb
                xsend
                or al,al
                jnz sfname01
                ret
SendFname       endp
;
Freemem         proc    near
                push es
                push ax
                mov ax,XUserSeg
                mov es,ax
                mov ah,49h
                int 21h
                pop ax
                pop es
                ret
```

```
Freemem         endp
;
NewTerminate:
                cli
                lds dx,cs:[oldint22h]
                mov ax,2522h
                int 21h
                mov ax,cs
                mov ds,ax
                mov es,ax
                mov ss,ax
                assume ds:code,es:code,ss:code
                mov sp,[SPsave]
                mov bx,cs
                mov ah,50h
                int 21h
                .dataseg
                commON
                sti
                mov ax,4d00h
                int 21h
                mov ah,4ch
                int 21h
;
RemoteRun       proc    far

RemoteRun       endp
;
NetworkRun      proc    near
                mov bx,0ffffh           ; allocate memory
                mov ah,48h              ;
                int 21h                 ; BX=the largest available blo
                jnc malloc_err
                mov ah,48h
                int 21h
                jc  malloc_err
                mov XUserSeg,ax
                mov ah,03h              ; get stat
                int 62h
                cmp al,SERVER
                je  server_run
client_run:
                mov al,8Bh              ; code remote run for server s
                mov ah,01h
                int 62h
                mov _baseaddr,dx
                jmp short nrun01
malloc_err:
                mov al,13
                jmp short nrun_exit
server_run:
                mov dx,_baseaddr
                xsend 88h               ; code remote run for login sh
nrun01:
                call SendFname
                call GetResponse        ; 00h -.COM, FFh -.EXE
                jc  nrun_exit
                or  al,al
                jnz nrun02
                jmp RunCOM
nrun02:
                cmp al,0FFh
                jnz nrun_exit
;###############################################################
                mov al,11
                ret
;               jmp RunEXE
;###############################################################
```

```
nrun_exit:
                ret                             ; return error code to print
NetworkRun      endp
;
RunCOM:
                call RecFile
                mov ax,es
                mov cx,XUserSeg
                add cx,10h
                sub ax,cx
                mov cl,4
                shl ax,cl
                add ax,di
                mov cx,ax
                mov word ptr RegCX,cx
                call RunSetup
                xor bx,bx
                xor dx,dx
                xor si,si
                xor di,di
                mov bp,dx
                mov cx,[RegCX]
                cli
                mov SPsave,sp
                mov ax,XUserSeg
                mov es,ax
                mov ss,ax
                mov ds,ax
                assume ds:nothing,es:nothing,ss:nothing
                mov sp,0fffeh
                mov ax,0302h                    ; parameter drive ID
                push bx                         ; push word 00h
                sti
                jmp dword ptr cs:[UserIP]
;
RunEXE          proc    far
                call RecFile
                call DecodeHeader
                call RunSetup
                cli
                mov SPsave,sp
                xor dx,dx
                xor si,si
                xor di,di
                mov bp,dx
                mov bx,[RegBX]
                mov cx,[RegCX]
                mov ss,[RegSS]
                mov sp,[RegSP]
                mov ax,XUserSeg
                mov es,ax
                mov ds,ax
                assume ds:nothing,es:nothing
                mov ax,0302h                    ; parameter drive ID
                push cs:[RegCS]
                push cs:[RegIP]
                sti
                ret                             ; go execute
RunEXE          endp
;
RunSetUp        proc    near
                cli
                mov ax,3522h
                int 21h
                mov word ptr oldint22h,bx
                mov word ptr oldint22h+2,es
                mov dx,offset NewTerminate
                mov ax,2522h
```

```
                int 21h
                mov dx,XUserSeg
                mov ah,26h
                int 21h
                mov bx,XUserSeg
                mov ah,50h
                int 21h
                mov ax,XUserSeg
                mov ds,ax
                assume ds:nothing
                mov dx,80h                      ; ds:dx
                mov ah,1ah
                int 21h
                .dataseg
                mov ax,XUserSeg
                mov es,ax
                assume es:nothing
                mov di,80h
                mov al,CommandLength
                stosb                           ; command line length
                mov si,offset CommandLine
                mov cx,100                      ; command tail 128
                cld
                rep movsb                       ; ds:si -> es:di
                sti
                ret
RunSetup        endp
;
DecodeHeader    proc    near
                mov ax,cs:[XUserSeg]
                mov ds,ax
                assume ds:nothing
                mov si,100h
                mov dx,[si+HeadSize]
                add dx,ax
                mov cs:[UserCodeSeg],dx
;
                mov ax,[si+LastPage] ;convert 512 bytes
                mov cx,[si+NumPage]
                jcxz No_ofs
                dec ax
No_ofs:
                cwd
                mov dx,512
                mul dx
                xor bx,bx
                add ax,cx
                adc dx,bx
                mov cs:[RegBX],dx
                mov cs:[RegCX],ax
; Relocate segment
                mov cx,[si+NumRelo]    ;number of rel . entry
                or  cx,cx
                jz  MainCont020
;
                mov bx,[si+ReloBeg]    ;begin of rel. table
                mov dx,cs:[UserCodeSeg]
MainCont015:
                les di,[bx]            ;get relocation ptr.
                mov ax,es
                add ax,dx
                mov es,ax              ;adjust pointer to actual code
                add es:[di],dx         ;relocating
                add bx,4               ;next entry
                loop MainCont015
MainCont020:
                .dataseg
                mov ax,XUserSeg
```

```
                mov es,ax
                assume es:nothing
                mov bx,UserCodeSeg
                mov ax,es:[StackSeg]
                add ax,bx
                mov RegSS,ax
                mov ax,es:[ProgSeg]
                add ax,bx
                mov RegCS,ax
                mov ax,es:[StackOfs]
                mov RegSP,ax
                mov ax,es:[ProgOfs]
                mov RegIP,ax
                sub bx,10h                      ; 10 para for PSP
                mov XUserSeg,bx
                ret
DecodeHeader    endp
;
RecFile         proc    near
                cli
                mov ax,XUserSeg
                add ax,10h
                mov es,ax
                assume es:nothing
                xor di,di                       ; mov di,100h
recfile01:
                receive
                cmp al,SOH
                jne recfile02
                send ACK
                jmp short recfile01
recfile02:
                cmp al,STX
                jne recfile03
                call RecBlk
                jmp short recfile01
recfile03:
                cmp al,ETX
                jne recfile04
                call RecByCx
                jc  recfile01
                jmp short recfileEnd
recfile04:
                cmp al,CAN
                jne recfile01
recfileEnd:
                sti
                ret
RecFile         endp
;
OldES           dw  0
;
RecBlk          proc    near
                xor di,di
                mov OldES,es
                mov dx, baseaddr
                xor bx,bx
                mov si,64                       ; total 1024 (64x16)
recblk01:
                mov cx,16
recblk02:
                getcom
                stosb                           ; es:di
                add bl,al
                loop recblk02
                mov ax,es                       ; set new paragraph
                inc ax                          ;
                mov es,ax                       ;
```

```
                xor di,di                       ; para:00
                dec si
                jnz recblk01
                getcom
                cmp al,bl
                je  recblk_ok
                mov es,OldES                    ; restore if ChkSum fail
                mov al,NAK
                jmp short recblkEnd
recblk_ok:
                mov al,ACK
recblkEnd:
                xsend
                ret
RecBlk          endp
;
RecByCx         proc    near
                xor di,di
                mov OldES,es
                mov dx,_baseaddr
                getcom
                mov ch,al
                getcom
                mov cl,al
                or cx,cx
                jz recbycx_ok
                xor bx,bx
recbycx01:
                getcom
                stosb
                add bl,al
                cmp di,0Fh
                jbe recbycx02
                mov ax,es
                inc ax
                mov es,ax
                xor di,di
recbycx02:
                loop recbycx01
                getcom
                cmp al,bl
                je  recbycx_ok
                mov es,OldES
                mov al,NAK
                stc
                jmp short recbycxEnd
recbycx_ok:
                mov al,ACK
                clc
recbycxEnd:
                pushf
                xsend
                popf
                ret
RecByCx         endp
;
;[]------------ STACK ----------------------------------------[]
                dw      64 dup (?)
TopOfStack      equ     $
;
;[]------------ END ------------------------------------------[]
code            ends
                end     start
```

```
;               ************************************************
;               *                                              *
;               *         The Remote Access Manager V1.0       *
;               *              By    C . Teekayu               *
;               *                                              *
;               *  Copyright ( C ) 1991  R & D Computer System *
;               *   Remote2.asm          Last update 22-Feb-92  *
;               ************************************************
;
;[]-------------------------------------------------------------[]
SERVER          EQU     0
CLIENT          EQU     -1
;[]-------------------------------------------------------------[]
include         d:\teekayu\asmplus\include\asmplus.oop
include         d:\teekayu\asmplus\include\constant.oop
include         d:\teekayu\asmplus\include\diskio.oop
include         d:\teekayu\asmplus\include\comm.oop
include         d:\teekayu\asmplus\include\screen.oop
include         d:\teekayu\asmplus\include\pcnet.oop
;[]-------------------------------------------------------------[]
                .biosdata
;[]-------------------------------------------------------------[]
code            segment byte public 'CODE'
                assume cs:code,ds:code,es:code
                org 100h
start:
                jmp install
;[]-------------------------------------------------------------[]
Reference_point label   byte
;[]-------------------------------------------------------------[]
;[]------------ data -------------------------------------------[]
EXTRN   _baseaddr:word,_dataformat:byte,_divisor:byte
;
header  db '
        db '  PCNet V1.00 - Remote Access Manager
        db '  (C) Copyright 1992 PC-Network Project. All Rights Reserv
        db '
        db 'Use HotKey  Ctrl-LeftShift to EXIT ...   ',cr,lf,lf
        db '          ... Press anykey to remote control to ...',cr,lf,lf
;
bye_    db '
        db '  Remote Access Manager Version 1.00   Copyright (c) 1992
        db '
        db '$'
;
Syntax  db 'Usage:',cr,lf
        db '         REMOTE2  [port] [/X]',cr,lf
        db '          /X    :   S = Screen OFF',cr,lf,'$'
;
notready        db '    PCNet SERVER shell does not exist !',bell,cr,
;
notserver       db '    Remote2 can work on SERVER only !',bell,cr,lf
;
intcom          dw 0b0ch
oldintcom1      dd ?
oldintcom2      dd ?
;
SCREENOFF       db FALSE
;[]------------ code -------------------------------------------[]
install:
                printstr header
                mov ax,3562h
                int 21h
                cmp bx,offset Reference_point
                je  main01
                mov dx,offset notready
mainPrint:
                mov ah,9
```

```
                int 21h
                terminate
;
main01:
                mov ah,03h                      ; get stat
                int 62h
                cmp al,SERVER
                je  main02
                mov dx,offset notserver
                jmp short mainPrint
main02:
                call initparameter
                jc  mainPrint
                cli
                mov ax,350ch
                int 21h
                mov word ptr oldintcom1,bx
                mov word ptr oldintcom1+2,es
                mov ax,350bh
                int 21h
                mov word ptr oldintcom2,bx
                mov word ptr oldintcom2+2,es
                mov ax,intcom
                mov ah,25h
                mov dx,offset isrcomm
                int 21h
                mov ax,intcom
                mov al,ah
                mov ah,25h
                mov dx,offset combusy
                int 21h
                initscreen
                commON
                readkey
                call manager
                cli
                Scroll UP,0,0,24,79,0,07h
                setcursor 0,0
                mov ah,09h
                mov dx,offset bye_
                int 21h
                mov ax,250ch
                lds dx,cs:oldintcom1
                int 21h
                mov ax,250bh
                lds dx,cs:oldintcom2
                int 21h
                sti
                terminate
;
manager         proc    near                    ; ds -> biosdata , es -> video
                cmp byte ptr SCREENOFF,TRUE
                je  manager0
                send 8Ah
manager0:
                .biosseg
manager1:
                mov al,_shift
                and al,00000110b
                cmp al,00000110b
                je  endmanager
                mov bx,_head
                cmp bx,_tail
                je manager1
                call sendkeycode
                jmp short manager1
endmanager:
                .dataseg
```

```
                cmp byte ptr SCREENOFF,TRUE
                je  manager2
                send 8Bh
manager2:
                ret
manager         endp
;
; get key code send to host and save key in CX
sendkeycode     proc    near
                cli
                send 89h                ; code RemoteKey
                assume ds: biosdata
                mov al,[bx]             ; get key code
                inc bx
                send
                mov al,[bx]
                inc bx
                send
                cmp bx,offset _endbuffer;
                jne keycode1            ;
                mov bx,offset _kbbuffer ;
keycode1:
                mov _head,bx            ; adjust head pointer
                sti                     ;
                ret                     ;
sendkeycode     endp
;
combusy         proc    far
                cli
                push ax
                push dx
                mov dx,cs:_baseaddr
                in al,dx
                jmp short $+2
                xsend NAK
                eoi
                pop dx
                pop ax
                sti
                iret
combusy         endp
;
;[]------------------------------------------------------[]
; int. communication service
;[]------------------------------------------------------[]
isrcomm         proc    far
                cli
                push ds
                push ax
                push dx
                .dataseg
                mov dx, BaseAddr
                in al,dx
                jmp short $+2
                cmp al,SPACKED
                jne exit_isrcomm
                call RecScreen
exit_isrcomm:
                mov al,20h
                out 20h,al
                jmp short $+2
                pop dx
                pop ax
                pop ds
                sti
                iret
isrcomm         endp
;
```

```
Screen          db 1024*2 dup (0)
RecScreen       proc    near
                push es
                push di
                mov ax,cs
                mov es,ax
                assume es:code
                mov di,offset Screen    ; es:di -> buffer
                cld
RecScreen01:
                receive
                stosb
                cmp al,0ffh
                jne RecScreen01
                receive
                stosb
                cmp al,0ffh
                jne RecScreen01
                call PutScreen
                pop di
                pop es
                ret
RecScreen       endp
;
PutScreen       proc    near
                push cx
                push si
                mov ax,_videoaddr
                mov es,ax
                assume es:nothing
                mov si,offset Screen
                xor di,di
PutScreen01:
                lodsb
                cmp al,0
                jne PutScreen02
                lodsb
                mov ah,al               ; save attr.
                jmp short PutScreen01
PutScreen02:
                cmp al,0ffh
                jne PutScreen03
                lodsb                   ; now ax = char. with attr.
                cmp al,0ffh
                je exitPutScreen
                mov cx,ax               ; save
                lodsw                   ; get counter
                xchg ax,cx
                rep stosw
                jmp short PutScreen01
PutScreen03:
                stosw
                jmp short PutScreen01
exitPutScreen:
                pop si
                pop cx
                ret
PutScreen       endp
;
;[]------------------------------------------------[]
; initialize parameter from command line
;[]------------------------------------------------[]
initparameter   proc    near
                mov _baseaddr,COM1ADDR
                mov si,81h              ; ds:si
                cld
cut_blank:
                lodsb
```

```
                cmp al,20h
                je cut_blank
                cmp al,0dh
                je initpara_end
                cmp al,'2'
                je  para02
                cmp al,'1'
                je  para03
                cmp al,'/'
                je  para04
                jmp short initpara_usage
para02:
                mov _baseaddr,COM2ADDR
                mov intcom,0c0bh
para03:
                lodsb
                cmp al,20h
                je  para03
                cmp al,0dh
                je  initpara_end
                cmp al,'/'
                jne initpara_usage
para04:
                lodsb
                or al,20h
                cmp al,'s'
                jne initpara_usage
                mov byte ptr SCREENOFF,TRUE
                clc
                jmp short initpara_end
initpara_usage:
                mov dx,offset Syntax
                stc
initpara_end:
                ret
initparameter   endp
;
code            ends
                end start
```