การควบคุมอุปกรณ์ไฟฟ้าผ่านเครือข่ายอินเตอร์เน็ต

Office Automation Via Internet

นายวุฒิศักดิ์ องค์พัฒนากุล
Mr. Wutisak Ongpatanakul
นายอัครเดช คฤสาร
Mr. Akkaradech Karersan
นายอันตกร วางวรรณ์
Mr. Andakorn Wangwan

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิศวกรรมอุตสาหการ
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2545

หัวข้อวิทยานิพนธ์	การควบคุมอุปกรณ์ไฟฟ้าผ่านเครือข่ายอินเตอร์เน็ต	
	Office Automation Via Internet	
นักศึกษา	นายวุฒิศักดิ์ องค์พัฒนากุล	รหัสนักศึกษา 42010637
	นายอัครเดช คฤสาร	รหัสนักศึกษา 42010690
	นายอันตกร วางวรรณ์	รหัสนักศึกษา 42010692
ปริญญา	วิศวกรรมศาสตรบัณฑิต	
สาขาวิชา	วิศวกรรมอุตสาหการ	
ปีการศึกษา	2545	

อาจารย์ผู้ควบคุมปริญญานิพนธ์

(ผศ.ดร.สรรพสิทธิ์ ลิ่มนรรัตน์)

(อ.พลชัย โชติปรายนกุล)

(อ.อุดม จันทร์จรัสสุข)

หัวข้อวิทยานิพนธ์	การควบคุมอุปกรณ์ไฟฟ้าผ่านเครือข่ายอินเตอร์เน็ต
นักศึกษา	นายวุฒิศักดิ์ องค์พัฒนากุล
นักศึกษา	นายอัครเดช คฤสาร
นักศึกษา	นายอันตกร วางวรรณ์
ระดับการศึกษา	วิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมอุตสาหการ
	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา	2545
อาจารย์ผู้ควบคุมปริญญานิพนธ์	ผศ.ดร.สรรพสิทธิ์ ลิ่มนรรัตน์
	อ.พลชัย โชติปรายนกุล
	อ.อุดม จันทร์จรัสสุข

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้เป็นการออกแบบ และสร้างระบบควบคุมอุปกรณ์ทางไฟฟ้าในระยะไกลผ่านระบบเครือข่ายอินเตอร์เน็ตโดยใช้ไมโครคอนโทรลเลอร์ MCS-51 เป็นตัวควบคุมและใช้ภาษา C ในการเขียนสั่งงานกับไมโครคอนโทรลเลอร์ในการสั่งงานรีเลย์เพื่อควบคุมอุปกรณ์ไฟฟ้า โดยแบ่งวงจรหลักเป็น 4 ส่วนคือ วงจรไมโครคอนโทรลเลอร์ วงจรรีเลย์ วงจร ULN-OPTO และวงจรเซนเซอร์ สำหรับในส่วนโปรแกรมจะมีการแสดงสถานะของอุปกรณ์และส่วนควบคุมแก่ผู้ใช้งาน โดยใช้โปรแกรม Delphi 6 ในการพัฒนา เพื่อเป็นแนวทางในการเรียนรู้การเขียนโปรแกรมระหว่าง Delphi 6 และอุปกรณ์ไฟฟ้า โดยสามารถนำไปใช้งานกับอุปกรณ์ไฟฟ้าได้จริง และสามารถนำไปประยุกต์ใช้ได้กับการควบคุมอุปกรณ์ต่าง ๆ ในอุตสาหกรรม โดยหวังเป็นอย่างยิ่งว่าปริญญานิพนธ์ฉบับนี้จะเป็นประโยชน์ต่อผู้ที่สนใจ ต่อไปในอนาคต

Thesis Title	OA Via Internet
Student	Mr. Wutisak Ongpatanakul
	Mr. Akkaradech Karersan
	Mr. Andakorn Wangwan
Degree	Bachelor of Engineering in Industrial Engineering
	King Mongkut's Institute of Technology Ladkrabang
Academic Year	2002
Advisor	Asst.Prof.Dr. Sunpasit Limnararat
	Pholchai Chotiprayanakul
	Udom Chanjaratsuk

ABSTRACT

This thesis is concerning about designing and constructing an equipment which is used for controlling electrical remote instrument via internet. This equipment works by using Microcontroller MCS-51 as a controller and C language in operating the system. By deviding the main circuit into 4 categories which are microcontroller circuit, relay circuit, ULN-OPTO circuit and sensor circuit. The computer programming is designed to show the status of the equipment and the part of controller. We use programme "Delphi6" in developing and this can help people to learn how to write a joint programme between Delphi6 and electric component and can be adapted to use in operating things in various industries.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงด้วยดีเพราะคำปรึกษาและการให้กำลังใจจาก ผศ.พรศักดิ์ อรรถวานิช ที่เคารพนับถือของคณะผู้จัดทำ ผศ.ดร.สรรพสิทธิ์ ลิ่มนรรัตน์ ที่คอยเป็นกำลังใจ ดูแลเอาใจใส่พวกเราอยู่ตลอดเวลา คอยให้คำปรึกษาในเรื่องต่างๆ พร้อมทั้งการอำนวยความสะดวกทุกๆอย่าง ทั้งการอำนวยความสะดวกในด้านการทำโครงงานจนโครงงานนี้สำเร็จลุล่วงมาได้ ขอขอบพระคุณ อ.พลชัย โชติปรายนกุล ที่กรุณาดูแลให้คำปรึกษาในการทำโครงงาน คอยแก้ปัญหาต่างๆ และดูแลตลอดทั้งโครงงาน อ.อุดม จันทร์จรัสสุข ที่คอยแนะนำให้คำปรึกษา และช่วยแก้ปัญหาต่างๆ ขอขอบคุณ ดร.สิทธิพร พิมพ์สกุล ที่ช่วยเป็นกำลังใจให้พวกเราด้วยดีเสมอมา อ.เอกพจน์ ตันตราภิวัฒน์ ที่ให้คำแนะนำในตอนเริ่มทำโครงงาน ซึ่งโครงงานจะสำเร็จลุล่วงมิได้ ถ้าขาดอาจารย์ทุกท่านที่กล่าวมาข้างต้น ซึ่งทางคณะผู้จัดทำใคร่ขอขอบพระคุณเป็นอย่างยิ่ง

ขอขอบพระคุณ คุณพ่อ คุณแม่ และญาติพี่น้องของพวกเราทุกคน ที่คอยสนับสนุนและคอยให้กำลังใจในทุกๆเรื่องตลอดมา ถ้าไม่มีท่านวันนี้คงไม่มีพวกเรา และไม่มีโครงงานนี้

ขอขอบคุณภาควิชาวิศวกรรมอุตสาหการ ที่ทำให้เราเป็น IE ไม่มีที่ไหนเรียนแล้วมีความสุขเท่านี้อีกแล้ว

ขอขอบคุณคณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ขอขอบคุณเพื่อนๆทุกคนที่อดทนลำบากมาด้วยกัน และเป็นกำลังใจให้พวกเรามาโดยตลอด

ขอขอบคุณนายกฤดา ชื่นงูเหลือม นักศึกษาปริญญาโท ภาควิชาวิศวกรรมโทรคมนาคม ที่คอยให้คำปรึกษามาโดยตลอด

ขอขอบคุณกำลังใจจากคนพิเศษของพวกเรา Jiu Peak Tak สำหรับกำลังใจที่มีให้อยู่ตลอดเวลา

ขอขอบคุณทุกๆคน ที่ทำให้มีวันนี้แต่พวกเราไม่ได้กล่าวถึง

คณะผู้จัดทำ

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

หน้า

บทที่ 1

บทนำ

1.1 ความสำคัญของโครงงาน

ในปัจจุบันข้อมูลข่าวสารต่างๆ นั้นได้มีบทบาทที่สำคัญในชีวิตประจำวัน การที่สามารถรับรู้ข้อมูลข่าวสารจากที่ที่หนึ่งได้อย่างรวดเร็วจึงเป็นสิ่งสำคัญ อีกทั้งในปัจจุบันการติดต่อสื่อสารผ่านระบบเครือข่าย Internet นั้นได้เพิ่มความสำคัญและสามารถใช้งานได้ครอบคลุมและกว้างขวางมากยิ่งขึ้น การควบคุมระยะไกลก็ได้มีบทบาทสำคัญทางด้านงานควบคุมต่างๆ เพราะไม่จำเป็นที่ผู้ควบคุมต้องทำงานในสถานที่ ที่เครื่องมือตั้งอยู่ ถ้าเราสามารถนำคุณสมบัติของ Internet มาใช้จะทำให้เราสามารถลดเวลาในการสั่งงาน และช่วยให้การสั่งงานเป็นไปอย่างรวดเร็ว

1.2 จุดประสงค์และวัตถุประสงค์ของโครงงาน

1. ศึกษาการส่งผ่านข้อมูลในระยะไกล ผ่านระบบเครือข่าย Internet (TCP / IP)
2. นำไมโครคอนโทรลเลอร์ไปประยุกต์ เพื่อควบคุมอุปกรณ์ทางอิเล็กทรอนิกส์ และติดต่อการทำงานกับเครื่องคอมพิวเตอร์โดยผ่าน Serial port (RS - 232 C)
3. ศึกษากระบวนการเขียนภาษา Object Pascal เพื่อใช้การพัฒนาการเขียนโปรแกรม Delphi สำหรับการติดต่อกับ ไมโครคอนโทรลเลอร์ และการควบคุมระยะไกลผ่านระบบเครือข่ายอินเตอร์เน็ต
4. ทำการสร้างโปรแกรมที่สามารถควบคุม และแสดงสถานะ การทำงาน
5. ศึกษาการเขียนโปรแกรมภาษา C เพื่อควบคุมไมโครคอนโทรลเลอร์

1.3 ขอบเขตของโครงงาน

1. ใช้ไมโครคอนโทรลเลอร์ (89C52) ควบคุมอุปกรณ์ทาง อิเล็กทรอนิกส์ และติดต่อการทำงานกับคอมพิวเตอร์ผ่าน Port RS – 232C
2. ใช้ภาษา Delphi ในการเขียนโปรแกรมเพื่อควบคุมไมโครคอนโทรลเลอร์ และส่งผ่านข้อมูลระยะไกล
3. โปรแกรมสามารถควบคุมและแสดงสถานะ การทำงานทั้งที่ Server และ Client
4. ใช้ภาษา C ในการเขียนโปรแกรม สำหรับไมโครคอนโทรลเลอร์
5. สร้างชุดควบคุมเปิด-ปิดไฟผ่านระบบเครือข่ายอินเตอร์เน็ต

1.4 ประโยชน์ที่คาดว่าจะได้รับจากโครงงาน

1. เพิ่มความสะดวกสบายแก่ผู้ใช้งานในการควบคุมอุปกรณ์ไฟฟ้า
2. เป็นแนวทางในการศึกษาและการพัฒนาต่อเกี่ยวกับการควบคุมการทำงานของเครื่องจักรอุตสาหกรรม
3. เพื่อนำความรู้ที่ได้จากโครงงานมาประยุกต์ใช้ในงานด้านวิศวกรรม
4. เพื่อสร้างทักษะการทำงานเป็นหมู่คณะ
5. เพื่อทำให้รู้จักการวางแผนการทำงานและดำเนินงาน
6. เพื่อสร้างทักษะทางด้านเศรษฐศาสตร์เพื่อใช้ในการวิเคราะห์การใช้งบประมาณในการสร้าง

บทที่ 2
ทฤษฎีที่เกี่ยวข้อง

โครงงานการควบคุมอุปกรณ์ไฟฟ้าผ่านเครือข่ายอินเตอร์เน็ตได้ใช้ทฤษฎีต่างๆ ที่เกี่ยวเนื่องกับวงจรไฟฟ้าเป็นส่วนใหญ่ และได้เขียนโปรแกรม Delphi ขึ้นมาเพื่อใช้ในการควบคุมอุปกรณ์ไฟฟ้าต่างๆ ผ่านไมโครคอนโทรลเลอร์ MCS-51 โดยมีวงจรในส่วนแสดงสถานะย้อนกลับด้วย โดยทฤษฎีต่างๆที่ใช้ในโครงงานนี้มีดังนี้

2.1 ไมโครคอนโทรลเลอร์ MCS-51

MCS-51 เป็นไมโครคอนโทรลเลอร์แบบชิปเดี่ยวที่มีข้อดีต่างๆดังนี้

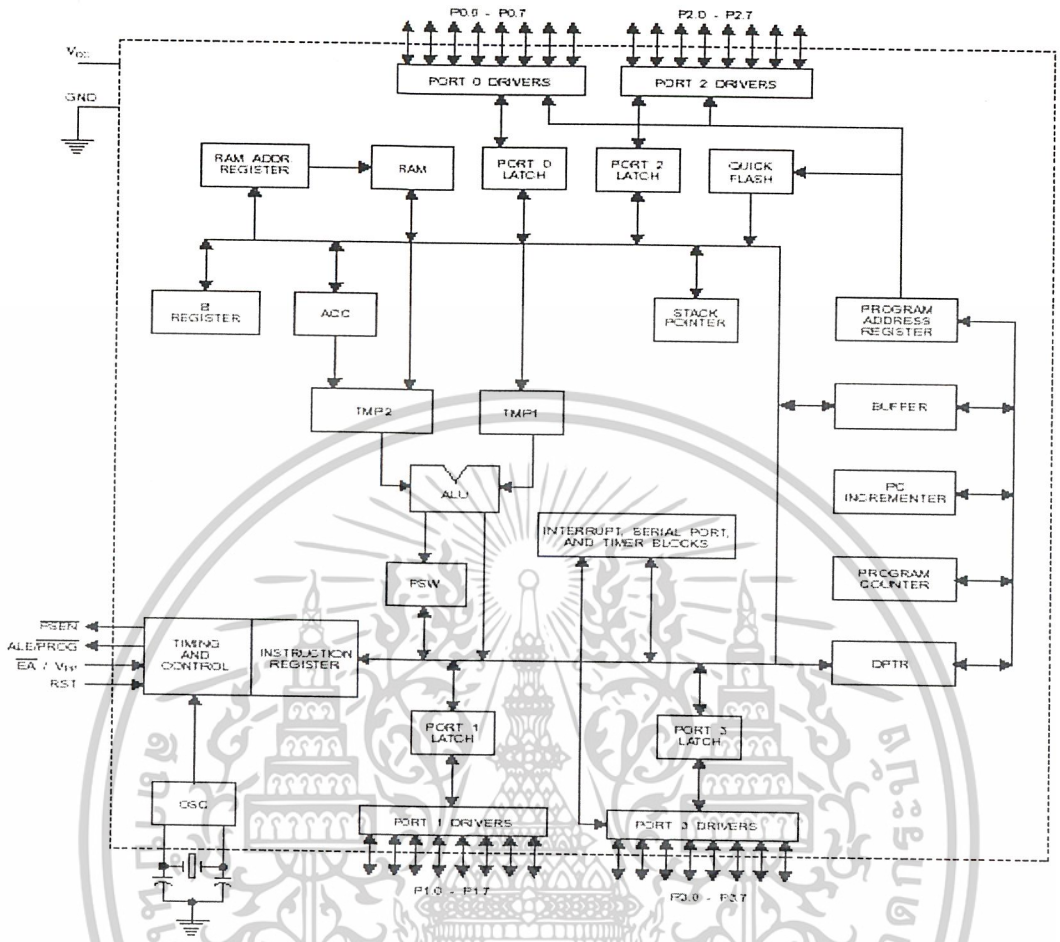
- มีหน่วยความจำสำหรับเก็บข้อมูลทั่วไป (RAM) บรรจุไว้ภายใน 128 – 256 ไบต์
- มีหน่วยความจำสำหรับเก็บโปรแกรมควบคุมการทำงานอยู่ภายในจำนวน 4 กิโลไบต์
- มีวงจรตั้งเวลาวงจรนับขนาด 16 บิต 2 ตัว อยู่ภายใน
- มีวงจรรับส่งข้อมูลอนุกรมได้ 2 ทิศทาง
- มีสัญญาณนาฬิกาภายในตัว
- มีพอร์ตที่สามารถรับหรือส่งข้อมูลได้ 2 ทิศทางจำนวน 4 พอร์ต ๆ ละ 8 บิต

นอกจากนี้MCS-51 ยังมีคุณสมบัติอื่นๆที่น่าสนใจ คือ

- ต้องการแหล่งจ่ายไฟ 5 โวลต์เพียงชุดเดียว
- มีหน่วยความจำสำหรับเก็บโปรแกรมควบคุมการทำงานอยู่ภายในชิป
- สามารถใช้หน่วยความจำสำหรับโปรแกรมควบคุมการทำงานอยู่ภายในชิป
- สามารถใช้หน่วยความจำสำหรับโปรแกรมและข้อมูลที่อยู่ภายนอกชิปได้อย่างละ 64 กิโลไบต์
- มีคำสั่งคูณและหารเลขขนาด 8 บิตในตัวเอง
- จัดลำดับความสำคัญของสัญญาณอินเตอร์รัปต์ได้ 2 ระดับ
- รับและส่งข้อมูลแบบอนุกรมได้ในตัว โดยสามารถกำหนดอัตราเร็วในการรับส่งข้อมูล ได้ตั้งแต่ 300 ถึง 375 กิโลบิตต่อวินาที
- สามารถประมวลผลแบบบูลีนเพื่อใช้ในงานควบคุมโดยเฉพาะ
- มีรีจิสเตอร์สำหรับใช้งานเป็นไทม์เมอร์หรือเคาน์เตอร์ เพื่อนับจำนวนสัญญาณนาฬิกา ภายในชิปหรือ นับการเปลี่ยนสถานะของสัญญาณภายนอกขนาด 16 บิต จำนวน 2 ตัว เพื่อใช้สำหรับนับจำนวนพัลส์ วัดความกว้างหรือพัลส์ วัดความกว้างของพัลส์หรือใช้วัดช่วงเวลา
- หน่วยความจำสำหรับเก็บข้อมูลภายในบางส่วนสามารถเข้าถึงข้อมูลได้ทั้งระดับไบต์และระดับบิตเพื่อให้การออกแบบโปรแกรมและการควบคุมระบบงานทำได้ง่ายขึ้น

2.1.1 สถาปัตยกรรมของไมโครคอนโทรลเลอร์

โครงสร้างภายในชิปไมโครคอนโทรลเลอร์ ชิปเดี่ยวแสดงดังรูป 2.1



รูปที่ 2.1 แสดงโครงสร้างภายในชิปไมโครคอนโทรลเลอร์

ขาการใช้งานต่างๆของไมโครคอนโทรลเลอร์มีดังนี้

- V_{CC} ขา 40 เป็นขาที่ต้องป้อนไฟเลี้ยง +5 โวลต์เข้าไปเพื่อให้วงจรสามารถทำงานได้
- V_{SS} ขา 20 เป็นขาที่ต้องต่อกับกราวด์ของแหล่งจ่ายไฟ
- RST ขา 9 ขารีเซตนี้จะรีเซตการทำงานของ 8051 ถ้าป้อนสัญญาณที่มีสภาวะลอจิก 1 ที่ขานี้จะเป็นการรีเซตการทำงาน กลับไปเริ่มการทำงานจากคำสั่งที่อยู่ในหน่วยความจำตำแหน่ง 0000
- ALE ขา 30 ใช้เป็นขาส่งสัญญาณออกไปภายนอก เพื่อควบคุมการแลตช์ค่าตำแหน่งไบต์ต่ำจากพอร์ต 0 ในระหว่างการติดต่อหน่วยความจำสำหรับเก็บโปรแกรมหรือข้อมูลภายนอก
- PSEN ขา 29 ใช้ส่งสัญญาณเพื่ออ่านคำสั่งจากโปรแกรมที่เก็บไว้ในหน่วยความจำภายนอกชิป
- XTALI ขา 19 ใช้ต่อคริสตอลภายนอก โดยเป็นอินพุทเข้าสู่วงจรออสซิเลเตอร์
- XTALI ขา 18 ใช้ต่อคริสตอลภายนอก โดยเป็นเอาท์พุทออกจากวงจรออสซิเลเตอร์
- PORT 0 เป็นพอร์ตขนานขนาด 8 บิตอยู่ที่ขา 39 ถึง 32 เริ่มจากบิต 0 ถึงบิต 7 ตามลำดับ พอร์ต 0 นี้ใช้ได้ทั้งการรับส่งตำแหน่งและข้อมูลกับหน่วยความจำหรือใช้รับส่งข้อมูลก็ได้ นอกจากนี้ยังใช้งานได้หลายอย่างดังนี้

1. ใช้สำหรับส่งค่าตำแหน่งหน่วยความจำที่ต้องการติดต่อด้วย โดย 8 บิตล่างถูกส่งออกไปทางพอร์ต 0 และ 8 บิตบนถูกส่งออกทางพอร์ต 2
2. ใช้รับส่งข้อมูลกับ Data Memory หรือใช้รับข้อมูลจาก Program Memory
3. ใช้รับส่งข้อมูลออกจากพอร์ตโดยตรง

- PORT 1 เป็นพอร์ตขนานขนาด 8 บิตอยู่ที่ขา 1 ถึง 8 เริ่มจากบิต 0 ถึงบิต 7 ตามลำดับใช้ทำหน้าที่เป็นตัวรับส่งข้อมูลเท่านั้น ไม่สามารถส่งตำแหน่งได้
- PORT 2 เป็นพอร์ตขนานขนาด 8 บิตอยู่ที่ขา 21 ถึง 28 เริ่มจากบิต 0 ถึงบิต 7 ตามลำดับใช้งานเพียง 2 ลักษณะ คือ
    1. ใช้ส่งค่าตำแหน่งหน่วยความจำภายนอกที่ต้องการติดต่อทำงานร่วมกับพอร์ต 0
    2. ใช้เป็นพอร์ตรับส่งข้อมูลกับภายนอก
- PORT 3 เป็นพอร์ตขนานขนาด 8 บิตอยู่ที่ขา 10 ถึง 17 เริ่มจากบิต 0 ถึงบิต 7 ตามลำดับ นอกจากจะใช้งานเหมือนกับพอร์ตอื่นๆ แล้วยังใช้งานอื่นโดยใช้คำสั่งควบคุม ดังนี้

P3.0 (RxD)	เป็นขาที่ใช้รับข้อมูลแบบอนุกรม
P3.1(TxD)	เป็นขาที่ใช้ส่งข้อมูลแบบอนุกรม
P3.2(INTO)	ใช้รับสัญญาณขัดจังหวะจากภายนอก
P3.3(INT1)	ใช้รับสัญญาณขัดจังหวะจากภายใน
P3.4(T0)	ใช้เป็นขารับสัญญาณให้เคาน์เตอร์ของไทม์เมอร์ 0
P3.5(T1)	ใช้เป็นขารับสัญญาณให้เคาน์เตอร์ของไทม์เมอร์ 1
P3.6(WR)	ใช้เป็นขาสัญญาณควบคุมการเขียนข้อมูลไปยังหน่วยความจำสำหรับข้อมูลภายนอก
P3.7(RD)	ใช้เป็นขาสัญญาณควบคุมการอ่านข้อมูลไปยังหน่วยความจำสำหรับข้อมูลภายนอก

พอร์ต 3 ของ MCS-51 ถูกใช้เป็นพอร์ตอนุกรม จะใช้ขา TxD และ RxD ในการรับส่งข้อมูล โดยขาทั้งสอง จะอยู่ในพอร์ต 3 คือ P3.1 หรือขา 11 เป็น TxD และ P3.0 หรือขา 10 เป็น RxD พอร์ตอนุกรม MCS-51 สามารถทำงานเป็นแบบ Full Duplex ได้ คือสามารถรับและส่งข้อมูลในเวลาเดียวกันได้ โดยมีการรับส่งข้อมูลจะมีบัฟเฟอร์ สำหรับเก็บข้อมูลให้ใช้

รีจิสเตอร์ที่สำคัญ ในการรับส่งข้อมูลคือ SBUF และ SCON ซึ่งเป็นรีจิสเตอร์ ที่อยู่ใน Special Function Register โดยรีจิสเตอร์ Serial Port Buffer (SBUF) จะอยู่ในตำแหน่ง 99H ถ้าเขียนข้อมูลลงไปในตำแหน่งนี้จะเป็นการส่งข้อมูลออกทางพอร์ตอนุกรม และถ้าอ่านข้อมูลจากตำแหน่งนี้จะเป็นการรับข้อมูลจากพอร์ทอนุกรม โดยในSBUF จะประกอบด้วยบัฟเฟอร์ 2 ตัวสำหรับส่งและรับข้อมูล

สำหรับ Serial Port Control Resistor (SCON) ซึ่งอยู่ที่ตำแหน่ง 98H จะเป็นรีจิสเตอร์ที่สามารถเข้าถึงข้อมูลระดับบิตได้ รีจิสเตอร์นี้จะทำหน้าที่ควบคุม และบอกสถานะต่างๆ ของการรับส่งข้อมูลแบบอนุกรม

สำหรับความเร็วของการส่งข้อมูล (Baud Rate) สามารถหาได้จากการหาสัญญาณนาฬิกาที่ใช้กับ MCS-51

ตารางที่ 2-1 Serial Port Control Resistor

บิต	ชื่อ	ตำแหน่ง	ความหมาย
SCON.7	SM0	9FH	บิตเลือกโหมดการทำงานบิต 0
SCON.6	SM1	9EH	บิตเลือกโหมดการทำงานบิต 1
SCON.5	SM2	9DH	บิตเลือกโหมดการทำงานบิต 2
SCON.4	REN	9CH	บิตแฟลกกำหนดยอมให้มีการรับข้อมูล
SCON.3	TB8	9BH	ค่าของบิต 9 สำหรับการส่งข้อมูลในโหมด 2 และ 3 สามารถเซต และเคลียร์ได้โดยซอฟท์แวร์
SCON.2	RB8	9AH	ค่าของบิต 9 เมื่อรับข้อมูลเข้ามา
SCON.1	TI	99H	บิตแฟลกแสดงการอินเทอร์รัพท์ภายหลังการส่งข้อมูลออกไป โดยจะเซตเมื่อส่งข้อมูลออกไปหมดแล้ว และสามารถเคลียร์ได้โดยซอฟท์แวร์
SCON.0	RI	98H	แฟลกแสดงการอินเทอร์รัพท์ภายหลังรับข้อมูลเข้ามาสามารถเคลียร์ได้โดยซอฟท์แวร์

ตารางที่ 2-2 แสดงโหมดต่าง ๆ ของการรับข้อมูลแบบอนุกรม

SM0	SM1	MODE	ความหมาย	Baud Rate
0	0	0	Shift Register	เปลี่ยนแปลงไม่ได้ (Oscillator Frequency / 12)
0	1	1	8 – bit UART	สามารถเปลี่ยนแปลงได้โดยกำหนดจากไทม์เมอร์
1	0	2	9 – bit UART	เปลี่ยนแปลงไม่ได้ ได้ (Oscillator Frequency / 12 หรือ /64)
1	1	3	9 – bit UART	สามารถเปลี่ยนได้จากไทม์เมอร์

2.1.2 ไทม์เมอร์ / เคาน์เตอร์

ไมโครคอนโทรลเลอร์ในตระกูล มีรีจิสเตอร์พิเศษที่สามารถเลือกใช้งานเป็นไทม์เมอร์ หรือเคาน์เตอร์อย่างใดอย่างหนึ่ง รีจิสเตอร์ประเภทนี้มีอยู่ด้วยกัน 2 ตัว แต่ละตัวขนาด 16 บิต เรียกไทม์เมอร์ 0 และไทม์เมอร์ 1 ตามลำดับ

- ไทม์เมอร์นั้นค่ามนรีจิสเตอร์ที่ใช้เป็นไทม์เมอร์ที่ถูกเลือกใช้งานจะถูกเพิ่มขึ้นทุกแมชชีนไซเคิล
- เคาน์เตอร์นั้นค่าในรีจิสเตอร์ที่ใช้เป็นเคาน์เตอร์ที่ถูกเลือกใช้งานจะถูกเพิ่มค่าทีละ 1 เมื่อมีการเปลี่ยนสถานะ

ไทม์เมอร์ 0 และไทม์เมอร์ 1

สามารถเลือกการทำงานให้เป็นไทม์เมอร์หรือเคาน์เตอร์ได้โดยการกำหนดค่าบิตในรีจิสเตอร์ใช้งานเฉพาะ โดยหากค่าบิตนี้มีค่าเป็น 0 หมายถึงเลือกใช้งานเป็นไทม์เมอร์ ถ้าบิตนี้มีค่าเป็น 1 หมายถึงเลือกใช้งานเป็นเคาน์เตอร์

นอกจากจะเลือกการทำงานของรีจิสเตอร์ให้เป็นไทม์เมอร์ หรือ เคาน์เตอร์ได้แล้วในแต่ละการทำงานยังมีการทำงานย่อยอยู่อีก 4 แบบ ตามความเหมาะสมของการใช้งาน โหมด 0 จะใช้รีจิสเตอร์ขนาด 8 บิตเป็นตัวนับโดยมีการเพิ่ม

ค่าครั้งละ 1 ทุกครั้งนับสัญญาณได้ครบ 32 ครั้ง โดยในโหมดนี้รีจิสเตอร์ที่ใช้นับเพียง 13 บิต (8 บิตในรีจิสเตอร์ TLx รวมกับ 5 บิตใน THx)

โหมด 1 การทำงานเหมือนโหมด 0 เว้นแต่ค่าในรีจิสเตอร์ถูกใช้งานครบทั้ง 16 บิตนั่นเอง คือไทม์เมอร์ หรือเคาน์เตอร์ในโหมดนี้มีขนาด 16 บิต

โหมด 2 ในโหมดนี้จะกำหนดรีจิสเตอร์ใช้งานในการนับเพียง 8 บิต (จากรีจิสเตอร์ TLx) ที่มีการโหลดค่าด้วยค่าในรีจิสเตอร์ THx การใช้งานโหมดนี้มีไว้เพื่อสร้างสัญญาณอินเตอร์รัปต์ที่มีคาบเวลาคงที่

โหมด 3 ในโหมดนี้ไทม์เมอร์ 1 จะไม่มีการนับแต่ไทม์เมอร์จะบังคับให้รีจิสเตอร์อ TL0 ของไทม์เมอร์ 0 ถูกใช้เป็นไทม์เมอร์เพียงอย่างเดียว การทำงานโหมด 3 มีไว้เพื่อใช้งานที่ต้องการไทม์เมอร์หรือเคาน์เตอร์ขนาด 8 บิตเพิ่มขึ้น

ตารางที่ 2-3 แสดงค่าในรีจิสเตอร์ TMOD ค่าต่างๆสำหรับ Timer 0

โหมด	ฟังก์ชั่นไทม์เมอร์ 0	ควบคุมจากโปรแกรม	TMOD ควบคุมจากฮาร์ดแวร์ภายนอก
0	13 bit Timer	00H	08H
1	16 bit Timer	01H	09H
2	8 bit Auto Reload	02H	0AH
3	two 8 bit Timer	03H	0BH

ตารางที่ 2-4 แสดงค่าในรีจิสเตอร์ TMOD ค่าต่างๆสำหรับ Counter 0

โหมด	ฟังก์ชั่นไทม์เมอร์ 0	ควบคุมจากโปรแกรม	TMOD ควบคุมจากฮาร์ดแวร์ภายนอก
0	13 bit Timer	04H	0CH
1	16 bit Timer	05H	0DH
2	8 bit Auto Reload	06H	0EH
3	two 8 bit Timer	07H	0FH

ตารางที่ 2-5 แสดงค่าในรีจิสเตอร์ TMOD ค่าต่างๆสำหรับ Timer 1

โหมด	ฟังก์ชั่นไทม์เมอร์ 0	ควบคุมจากโปรแกรม	TMOD ควบคุมจากฮาร์ดแวร์ภายนอก
0	13 bit Timer	00H	80H
1	16 bit Timer	10H	90H
2	8 bit Auto Reload	20H	A0H
3	does not run	30H	BoH

ตารางที่ 2-6 แสดงค่าในรีจิสเตอร์ TMOD ค่าต่างๆสำหรับ Counter 1

โหมด	ฟังก์ชั่นไทม์เมอร์ 0	ควบคุมจากโปรแกรม	TMOD ควบคุมจากฮาร์ดแวร์ภายนอก
0	13 bit Timer	40H	C0H
1	16 bit Timer	50H	D0H
2	8 bit Auto Reload	60H	E0H
3	not available	--	--

2.1.3 รีจิสเตอร์สำหรับใช้งานทั่วไปใน MCS-51

รีจิสเตอร์ A,B และรีจิสเตอร์ใช้งานทั่วไป R0-R7 ซึ่งอยู่ในหน่วยความจำสำหรับเก็บข้อมูลทั่วไปภายในชิฟ บริเวณ 128 ไบต์แรก รีจิสเตอร์ใช้งานทั่วไป R0-R7 ใน MCS-51 มีอยู่ด้วยกันทั้งหมด 4 กลุ่ม แต่ละกลุ่มประกอบด้วยรีจิสเตอร์จำนวน 8 ตัว (R0 – R7) ซึ่งมีชื่อเรียกเหมือนกัน ดังนั้นรีจิสเตอร์ใช้งานทั่วไป R0-R7 ใน MCS-51 จึงมีทั้งหมด 32 ตัว ในการทำงานขณะใดๆ รีจิสเตอร์ทั้ง 4 กลุ่ม (R0 – R7) จะถูกเลือกใช้งานเพียงกลุ่มเดียวเท่านั้น การเลือกใช้งานรีจิสเตอร์ R0-R7 กลุ่มใดกลุ่มหนึ่งใน 4 กลุ่มกระทำโดยการเซตหรือเคบียร์บิต RS0 , RS1 ในรีจิสเตอร์ใช้งานเฉพาะ PSW

2.1.4 รีจิสเตอร์ฟังก์ชั่นพิเศษใน MCS – 51

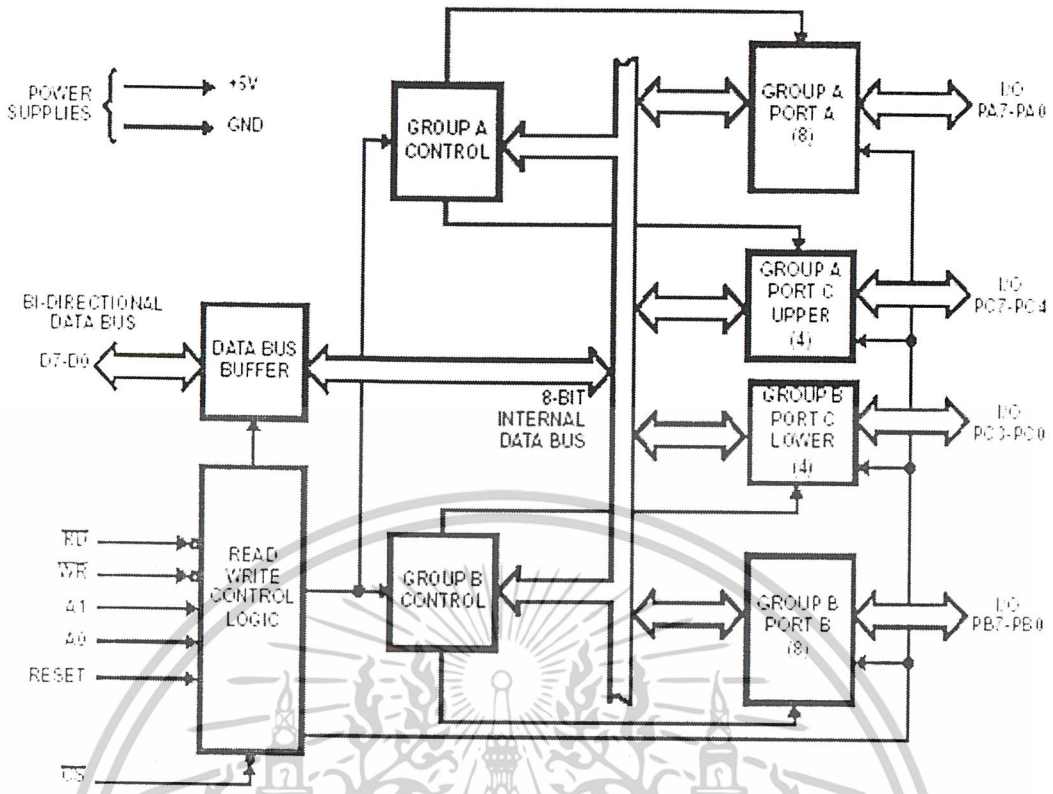
ใน MCS-51 รีจิสเตอร์จะใช้หน่วยความจำ RAM ภายในชิป โดยส่วนหนึ่งเป็นรีจิสเตอร์พิเศษ (Special Function Register : SFR) ซึ่งมีทั้งหมด 21 ตัว โดยรีจิสเตอร์พิเศษต่างๆ จะเริ่มที่หน่วยความจำตั้งแต่ 80H ถึง FFH ซึ่งมีทั้งหมด 128 ตำแหน่ง แต่จะเป็นรีจิสเตอร์ฟังก์ชั่นพิเศษเพียง 21 ตำแหน่ง แต่ถ้าเป็น 8032 / 8051 จะใช้ 26 ตำแหน่งหรือมี SFR 26 ตัว

ตารางที่ 2-7 รีจิสเตอร์ใช้งานเฉพาะ PSW (Program Status Word) เข้าถึงข้อมูลได้ในระดับบิต

บิต	ชื่อบิต	ตำแหน่ง	ความหมาย
PSW.7	CY	D7H	Carry Flag
PSW.6	AC	D6H	Auxiliary Carry Flag
PSW.5	F0	D5H	Flag 0
PSW.4	RS1	D4H	บิตสำหรับเลือกรีจิสเตอร์ แบงค์ 1
PSW.3	RS0	D3H	บิตสำหรับเลือกรีจิสเตอร์ แบงค์ 0 00 = Bank 0 ; Address 00H – 07H 01 = Bank 1 ; Address 08H – 0FH 10 = Bank 2 ; Address 10H – 17H 11 = Bank 3 ; Address 18H – 1FH
PSW.2	OV	D2H	Overflow Flag
PSW.1	-	D1H	Reserved
PSW.0	P	D0H	Even Parity Flag

2.2 ลักษณะพื้นฐานของ 8255

ไอซีเบอร์ 8255 ได้รับการออกแบบมาเพื่อทำหน้าที่เป็นพอร์ตสำหรับการรับส่งข้อมูลแบบขนานระหว่างอุปกรณ์ภายนอกกับไมโครคอนโทรลเลอร์ จากแผนภาพรูปที่ 2.2 จะพบว่า 8255 ประกอบด้วยบล็อก ของหน่วยการทำงานหลายส่วน ภายในบล็อกทางด้านขวามือจำนวน 4 บล็อก เป็นส่วนที่เชื่อมต่อกับอุปกรณ์ภายนอกโดยตรงผ่านทางสัญญาณที่ระบุชื่อว่า PA0 – PA7 , PB0 – PB7 และ PC0 – PC7



รูปที่ 2.2 แสดงแผนภาพแบบบล็อกภายในและขาสัญญาณของไอซีเบอร์ 8255

กลุ่มของสัญญาณเหล่านี้จำแนกออกเป็น 3 กลุ่ม คือ พอร์ต A (PA) , พอร์ต B (PB) และพอร์ต C (PC) สำหรับบล็อกถัดเข้ามาบริเวณส่วนกลาง มีชื่อว่า GROUP A CONTROL และ GROUP B CONTROL ทำหน้าที่กำหนดการทำงานของพอร์ตทั้งสาม บล็อกทั้งสองนี้เชื่อมต่อกับบล็อกอื่น ๆ ผ่านทางบัสข้อมูลภายใน 8255 เอง สำหรับบล็อกการทำงานทางด้านซ้าย ที่มีชื่อว่า ดาต้าบัสบัฟเฟอร์ (Data Bus Buffer) และ อ่าน/เขียน คอนโทรลลอจิก (Read/Write Control Logic) ทำหน้าที่เชื่อมต่อระหว่างระบบบัสของไมโครคอนโทรลเลอร์กับ 8255 เพื่อรับหรือส่งข้อมูลระหว่างกันตามระดับลอจิกของขาสัญญาณ RD\ และ WR\ ตามลำดับ

2.2.1 การจำแนกกลุ่มของพอร์ต 8255

พอร์ต 8255 คือ พอร์ต A พอร์ต B และ พอร์ต C โดยพื้นฐานจะเป็นพอร์ตแบบขนานที่ประกอบด้วยสัญญาณ 8 เส้น ซึ่งแต่ละเส้นจะแทนบิตของข้อมูลพอร์ต (เป็นพอร์ตแบบ 8 บิต) นอกจากนี้ยังสามารถอ้างถึงแต่ละบิตของเส้นสัญญาณพอร์ตนี้ได้โดยอิสระ แต่อย่างไรก็ตาม 8255 ได้จัดกลุ่มของพอร์ตเหล่านี้ออกเป็น 2 กลุ่ม คือ กลุ่ม A และกลุ่ม B เพื่อประโยชน์ในการกำหนดรูปแบบการทำงานของพอร์ต

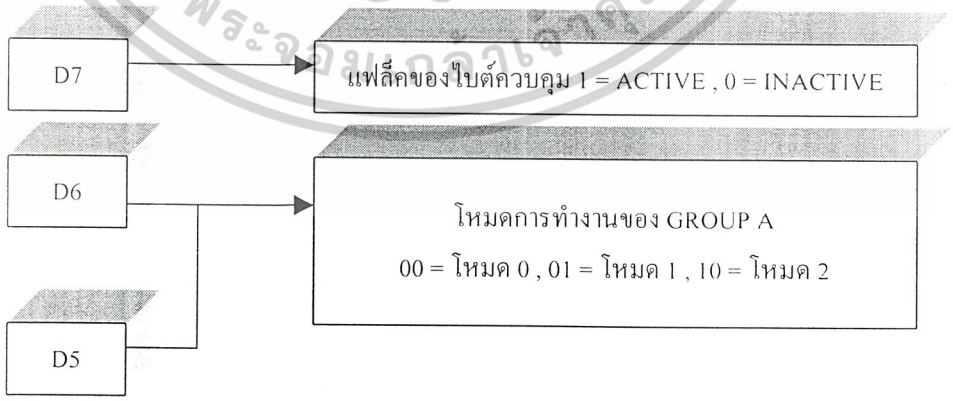
ตารางที่ 2-8 ตารางแสดงการจัดกลุ่มของพอร์ตของ 8255

ชื่อกลุ่ม	ลักษณะ
กลุ่ม A	พอร์ต A จำนวน 8 บิต (ทุกบิตของพอร์ต) พอร์ต C จำนวน 4 บิต (เฉพาะ 4 บิตบนของพอร์ต)
กลุ่ม B	พอร์ต B จำนวน 8 บิต (ทุกบิตของพอร์ต) พอร์ต C จำนวน 4 บิต (เฉพาะ 4 บิตล่างของพอร์ต)

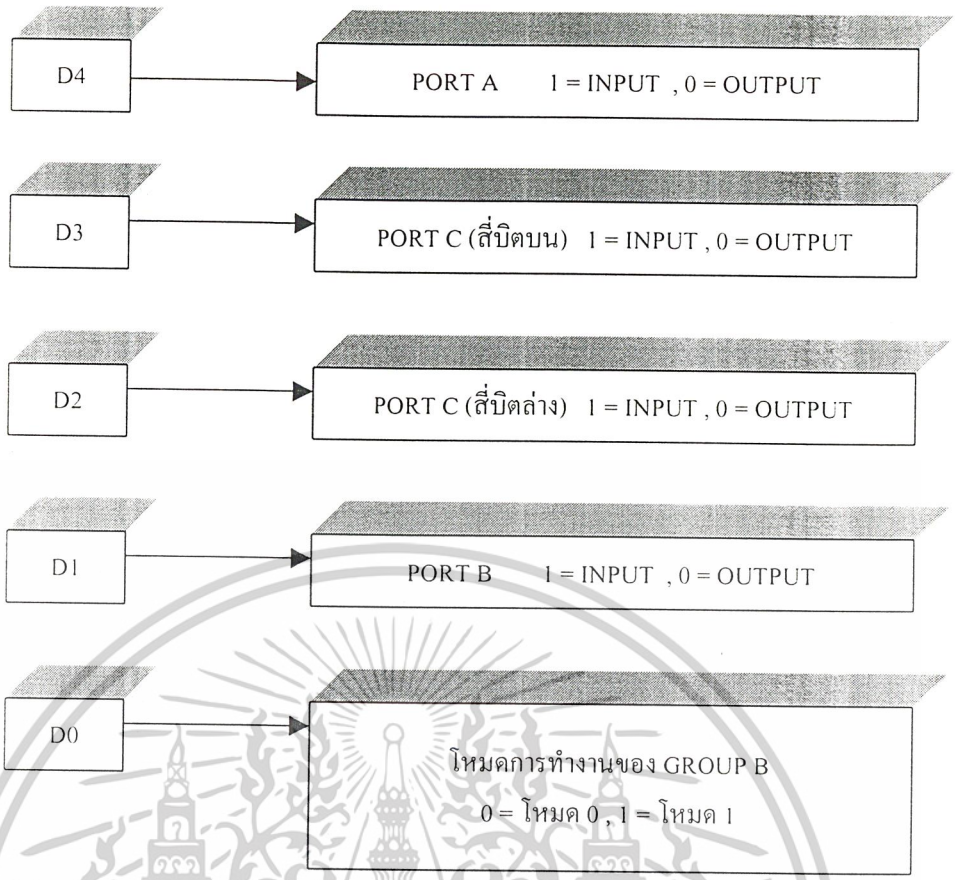
จากตารางการทำงานข้างต้นจะเห็นว่า จำนวนเส้นสัญญาณทั้งหมดของพอร์ต C ได้ถูกแยกออกเป็นกลุ่ม คือ กลุ่มของ 4 บิตล่างจาก PC0 – PC3 และกลุ่มของ 4 บิตบนจาก PC4 – PC7 ดังนั้น กลุ่ม A และ กลุ่ม B ของ 8255 จึงมักมีจำนวนบิตในแต่ละกลุ่มเป็นจำนวนถึง 12 บิต

ตารางที่ 2-9 ตารางหน้าที่การทำงานของขาสัญญาณไอซี 8255

สัญญาณ	ความหมาย
D0 – D7	กลุ่มของเส้นสัญญาณข้อมูลของ 8255 เมื่อมีการเขียนหรืออ่าน
CS\	สัญญาณเลือกอุปกรณ์ เมื่อขาสัญญาณนี้เป็นระดับลอจิกต่ำ ซีพียู ก็สามารถ เขียนหรืออ่านข้อมูลจาก 8255 ได้
RD\	สัญญาณบอกสถานะต้องการอ่านข้อมูลจากรีจิสเตอร์ของ 8255
WR\	สัญญาณบอกสถานะต้องการเขียนข้อมูลจากรีจิสเตอร์ของ 8255
A0 – A1	สัญญาณระบุตำแหน่งรีจิสเตอร์ภายใน 8255 ที่ต้องการ
RESET	สัญญาณการรีเซตวงจรทำงานภายใน 8255 เพื่อเริ่มต้นใหม่
PA0 – PA7	กลุ่มของสัญญาณ 8 เส้น เมื่อทำการติดต่อกับพอร์ต A ของ 8255
PB0 – PB7	กลุ่มของสัญญาณ 8 เส้น เมื่อทำการติดต่อกับพอร์ต B ของ 8255
PC0 – PC7	กลุ่มของสัญญาณ 8 เส้น เมื่อทำการติดต่อกับพอร์ต C ของ 8255



รูปที่ 2.3 แสดงความหมายของบิตภายในข้อมูลควบคุมสำหรับ 8255



รูปที่ 2.3 (ต่อ) แสดงความหมายของบิตภายในข้อมูลควบคุมสำหรับ 8255

2.2.2 รูปแบบคำสั่งเพื่อกำหนดการทำงานของ 8255

การกำหนดให้พอร์ตทั้งสามของ 8255 ทำงานในลักษณะต่าง ๆ กันหรือที่เรียกว่า โหมดการทำงาน จะเริ่มด้วยการส่งค่าข้อมูลไบต์หนึ่งให้กับรีจิสเตอร์ควบคุมการทำงานภายในตัว 8255 ข้อมูลนี้จะเรียกว่า ไบต์ข้อมูลควบคุม โดยแต่ละบิตข้อมูลนี้จะมีความหมายที่ระบุถึงความต้องการต่าง ๆ ไปดังแสดงในรูปที่ 2.3 การส่งข้อมูลไบต์นี้จะต้องเริ่มต้นเป็นลำดับแรกก่อนที่จะได้มีการดำเนินการใด ๆ กับ 8255

ตามความหมายของบิตภายในตารางของรูป 2.3 จะเห็นว่าการเลือกให้พอร์ตใดทำหน้าที่เป็นพอร์ตอินพุท ก็กำหนดค่าข้อมูล 1 ให้กับบิตที่เกี่ยวข้องกับพอร์ตนั้น หรือกรณีตรงข้ามสำหรับการเอาท์พุทก็เพียงการกำหนดค่าข้อมูล 0 เท่านั้น อย่างไรก็ตามการกำหนดให้ไบต์ข้อมูลควบคุมนี้มีผลอย่างถูกต้อง ก็จะต้องทำการกำหนดให้บิต D7 มีค่าเป็น 1 เสมอ

2.2.3 การเชื่อมต่อ 8255 กับ ไมโครคอนโทรลเลอร์

เมื่อพิจารณาแผนภาพของ 8255 จะเห็นว่ามีขาสัญญาณแอดเดรสจำนวน 2 เส้น คือ A0 และ A1 ทำให้ตำแหน่งของแอดเดรสที่จะอ้างถึงได้มีค่าเป็น 2^2 หรือเท่ากับ 4 ตำแหน่งซึ่งแต่ละตำแหน่งจะมีความหมายถึงการระบุรีจิสเตอร์หรือพอร์ตภายใน 8255

ตารางที่ 2-10 ตารางแสดงการระบุถึงรีจิสเตอร์หรือพอร์ตภายใน 8255

A1	A0	ชื่อของรีจิสเตอร์
0	0	พอร์ต A
0	1	พอร์ต B
1	0	พอร์ต C
1	1	รีจิสเตอร์ควบคุม

เมื่อพิจารณาค่าของแอดเดรสเหล่านี้ร่วมกับระดับลอจิกของขาสัญญาณ RD\ และ WR\ จะเป็นการอ่านหรือเขียนข้อมูลทางขาสัญญาณ D0 – D7 ให้กับรีจิสเตอร์นั้นตามลำดับ ดังตารางต่อไปนี้

ตารางที่ 2-11 ตารางแสดงความหมายของระดับของขาสัญญาณ RD\ และ WR\

RD\	WR\	A1	A0	ความหมาย
0	1	0	0	ส่ง (หรือเขียน) ข้อมูลให้กับพอร์ต A
1	0	0	0	รับ (หรืออ่าน) ข้อมูลให้กับพอร์ต A
0	1	0	1	ส่ง (หรือเขียน) ข้อมูลให้กับพอร์ต B
1	0	0	1	รับ (หรืออ่าน) ข้อมูลให้กับพอร์ต B
0	1	1	0	ส่ง (หรือเขียน) ข้อมูลให้กับพอร์ต C

ตารางที่ 2-11 (ต่อ) ตารางแสดงความหมายของระดับของขาสัญญาณ RD\ และ WR\

RD\	WR\	A1	A0	ความหมาย
1	0	1	0	รับ (หรืออ่าน) ข้อมูลให้กับพอร์ต C
0	1	1	1	ส่ง(หรือเขียน)ข้อมูลให้กับรีจิสเตอร์ควบคุม
1	0	1	1	เป็นสภาวะที่ไม่ถูกต้อง

ขาสัญญาณควบคุม RD\ และ WR\ มักจะเชื่อมต่อเข้ากับขาสัญญาณชื่อเดียวกับของไมโครคอนโทรลเลอร์ ได้โดยตรง ซึ่งทำให้แอดเดรสพอร์ตของ 8255 อยู่ในพื้นที่ของหน่วยความจำข้อมูลของ ไมโครคอนโทรลเลอร์ สำหรับขารีเซตของ 8255 ซึ่งจะมีผลทำให้เกิดการรีเซตหรือเริ่มสภาวะการทำงานใหม่เมื่อระดับของขาสัญญาณเป็นลอจิกสูง ดังนั้นหากว่าจะใช้สัญญาณการรีเซตเดียวกับของ ไมโครคอนโทรลเลอร์ เพื่อที่จะรีเซต 8255 ด้วยก็สามารถต่อได้โดยตรง ส่วนขาสัญญาณ D0 – D7 ก็สามารถนำไปเชื่อมต่อโดยตรงเข้ากับบัสของไมโครคอนโทรลเลอร์ ได้เช่นกัน

2.3 การเชื่อมต่อทางแสง

ตัวเชื่อมโยงทางแสง (Optocoupler) หรือตัวแยกโดยใช้แสง (Opto Isolator) เป็นอุปกรณ์ที่มีคุณสมบัติพิเศษหลายประการ เช่น คุณสมบัติในการไอโซเลททำให้สามารถนำมาใช้ในการเชื่อมโยงสัญญาณต่าง ๆ ของวงจรที่มีกราวด์ต่างกัน สามารถป้องกันการรบกวนซึ่งกันและกัน ระหว่างภาคอินพุทกับภาคเอาท์พุทได้อย่างเด็ดขาด ซึ่งการคัพปลิ้งด้วยวิธีอื่น ๆ จะทำไม่ได้ จึงได้นำเอาออปโต้คัพเปลอร์มาประยุกต์ใช้ในวงจร เพื่อประสิทธิภาพและความน่าเชื่อถือของวงจร

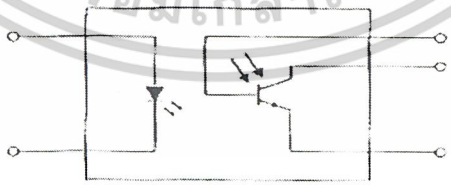
ออปโต้คัพเปลอร์ เป็นอุปกรณ์เดียวที่ประกอบด้วยแหล่งกำเนิดแสงและตัวตรวจรับแสงโดยที่ทั้งสองส่วนนี้จะแยกจากกัน โดยมีฉนวนที่โปร่งใส เช่น กระจกบาง ๆ คั่นกลางและชิ้นส่วนทั้งหมดจะถูกบรรจุอยู่ในตัวถังทึบแสง รูปร่างภายนอกมีอยู่หลายแบบ แต่ที่พบเห็นบ่อย ๆ ส่วนมากเป็นแบบดิพ (DIP : Dual In-Line Package) มีลักษณะเหมือนไอซี แต่มี 6 ขา แหล่งกำเนิดแสงส่วนใหญ่จะใช้ไดโอดเปล่งแสงอินฟาเรด (IRED : Infrared Emitter Diode) ทำจากสารแกลเลี่ยมอาร์เซไนด์ (GaAs) ส่วนตัวตรวจรับหรืออุปกรณ์ภาคเอาท์พุทนั้น อาจจะเป็นโฟโต้ดาร์ลิงตัน สวิตช์ 2 ทิศทาง (Triac) ซึ่งทำงานเมื่อมีแสงมากระตุ้น และเอสซีอาร์ (SCR) ที่ถูกกระตุ้นด้วยแสง เป็นต้น ออปโต้คัพเปลอร์ หรือออปโต้ไอโซเลเตอร์ได้รับการออกแบบไว้ให้ทำการป้องกันอุปกรณ์อิเล็คทรอนิกส์ ไม่ให้ได้รับแรงกระชากสูง ๆ หรือคุ้มครองระดับนอยส์ต่ำ ๆ ซึ่งเป็นต้นเหตุให้เกิดเอาท์พุทไม่ถูกต้องหรือทำให้เกิด คลื่นผิดพลาดขึ้นมา ออปโต้คัพเปลอร์เป็นอุปกรณ์ที่ทำให้สามารถเชื่อมต่อกับอุปกรณ์ตัวอื่น ๆ ที่มีระดับลอจิกแตกต่างกัน ในออปโต้คัพเปลอร์สัญญาณอินพุทจะถูกเปลี่ยนเป็นพลังงานแสงเพราะมี LED ที่อยู่ภายใน พลังงานจึงถูกส่งไปยังโฟโต้ดีเทคเตอร์ ดังนั้นมันจึงทำงานตรงกับพลังงานของแสงที่ได้จาก LED และมีสเปคตามอัตราส่วนการส่งผ่านกระแส (CTR) กับ Isolator Voltage CTR เป็นอัตราส่วนระหว่างกระแสอินพุทต่อกระแสเอาท์พุท ซึ่งเป็นการวัดความสามารถของออปโต้คัพเปลอร์ในเรื่องความสามารถให้สัญญาณอินพุทถูกส่งไปยังเอาท์พุทอย่างมีประสิทธิภาพ ซึ่งจะขึ้นอยู่กับประสิทธิภาพของ IRED ช่องว่างระหว่างชิ้นส่วนทางอินพุทและเอาท์พุทรวมทั้งพื้นที่ ความไว (Sensitivity) และอัตราการขยายตัวตรวจจับสำหรับ Isolation Voltage ของออปโต้คัพเปลอร์ คือ ปริมาณแรงดันที่ออปโต้คัพเปลอร์สามารถทำงานได้อย่างปลอดภัย

ตัวแปรอินพุททางด้านไฟฟ้ากระแสตรงเป็นตัวกำหนดตัวแปรทางด้านความไวของไดโอดเปล่งแสงอินฟาเรด (IRED) ได้แก่ กระแสของไดโอดเมื่อได้รับการไบอัสตรง (If) แรงดันตกคร่อมไดโอดเมื่อได้รับการไบอัสตรง (Vf) และแรงดันสูงสุดที่ทนได้เมื่อได้รับการไบอัสกลับ (Vr)

เนื่องจากตัวแปรเอาท์พุททางด้านไฟฟ้ากระแสตรงและตัวแปรส่งถ่าย (Transfer Parameter) จะแตกต่างกันโดยขึ้นอยู่กับชนิดของชิ้นส่วนที่เป็นตัวตรวจจับที่ใช้ในออปโต้คัพเปลอร์ ซึ่งจะมีรายละเอียดแตกต่างกันขึ้นอยู่กับตัวตรวจจับนั้น ๆ ตัวอย่าง เช่น

2.3.1 ทรานซิสเตอร์คัพเปลอร์ (Transistor Coupler)

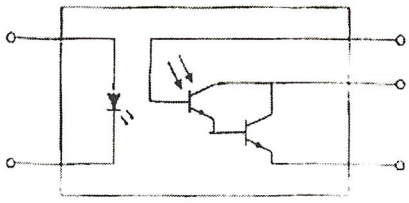
อุปกรณ์ประเภทนี้ได้รับความนิยมมากที่สุด มีความไวระดับกลาง มีราคาถูก ตรงจุดเชื่อมต่อ (Junction) ภายในระหว่างคอลเลคเตอร์ – เบส ของทรานซิสเตอร์สามารถเอาสายมาต่อข้างนอกให้ทำหน้าที่เป็นโฟโต้ไดโอด ซึ่งมีความเร็วในการทำงานสูงยิ่งไปกว่าเดิม



รูปที่ 2.4 ออปโต้แบบทรานซิสเตอร์คัพเปลอร์

2.3.2 ดาร์ลิงตันทรานซิสเตอร์คัพเปลอร์ (Darlington Transistor Coupler)

อุปกรณ์ประเภทนี้ให้อัตราส่วนการส่งกระแส หรือมีอัตราการขยายสูงสามารถให้กระแสเอาท์พุทเพิ่มขึ้น ซึ่งจะได้อัตราขยายสูงเป็น 10 เท่า แต่ความเร็วในการทำงานจะช้ากว่า 10 เท่า ของการใช้ทรานซิสเตอร์ตัวเดียว

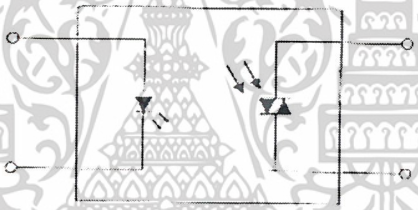


รูป 2.5 ออปโต้แบบดาร์ลิงตันทรานซิสเตอร์คัพเปลอร์

ออปโต้ทรานซิสเตอร์คัพเปลอร์ และแบบดาร์ลิงตันทรานซิสเตอร์คัพเปลอร์ นั้นมีหลักการทำงานเหมือนกัน รอยต่อระหว่างขาคอลเลคเตอร์กับขาเบสถูกทำให้กว้างขึ้น แสงที่ตกกระทบรอยต่อจะทำให้เกิดคู่ของอิเล็คตรอนและโฮลขึ้นมาเกิดการนำกระแสได้

2.3.3 ออปโต้คัพเปลอร์ที่ใช้สวิตซ์สองทิศทางหรือไตรแอค (TRIAC)

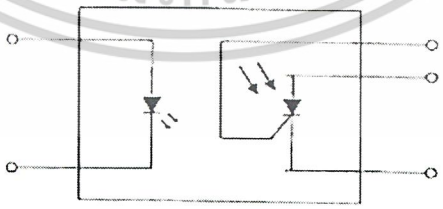
อุปกรณ์ประเภทนี้จะทำงานเมื่อมีแสงมากระตุ้นเป็นภาคเอาท์พุท ถูกออกแบบมาสำหรับใช้งานซึ่งต้องการแยกการทริก หรือ การกระตุ้นตัวไตรแอค การแยกสวิตซ์ทางด้านไฟฟ้ากระแสสลับที่มีขนาดกระแสต่ำ และการแยกกันทางไฟฟ้ามีค่าสูง



รูป 2.6 ออปโต้แบบไตรแอคคัพเปลอร์

2.3.4 ออปโต้คัพเปลอร์ที่ใช้เอสซีอาร์ (SCR) ที่ถูกกระตุ้นด้วยแสง

อุปกรณ์ประเภทนี้ ถูกออกแบบมาสำหรับใช้ในงานที่ต้องการการแยกกันทางไฟฟ้าที่มีค่ากระแสสูงระหว่างวงจรทางด้านแรงดันต่ำ (ซึ่งใช้ไอซี) และทางด้านไฟฟ้ากระแสสลับแรงดันสูง



รูป 2.7 ออปโต้แบบเอสซีอาร์คัพเปลอร์

2.4 การส่งผ่านข้อมูลผ่านทางพอร์ตอนุกรม

2.4.1 รูปแบบข้อมูลในคอมพิวเตอร์

การที่จะทำความเข้าใจการส่งผ่านข้อมูล สิ่งแรกคือต้องทำความเข้าใจกับวิธีที่ข้อมูลถูกเก็บไว้ภายในคอมพิวเตอร์ก่อน

2.4.2 บิตและไบต์

ในเลขฐานสิบ มีตัวเลขอยู่สิบตัว คือ 0-9 การเพิ่ม 0 หนึ่งตัวเข้าทางท้ายเป็นการคูณจำนวนด้วย 10 ในเลขฐาน 2 มีเพียงตัวเลข 2 ตัวคือ 0 และ 1 การเพิ่ม 0 เข้าทางท้ายจำนวนเป็นการคูณด้วย 2 ตัวเลข 0 หรือ 1 แต่ละตัวในเลขฐาน 2 เรียกว่า บิต 8 บิตเป็นหนึ่งไบต์ ผลที่ตามมาคือ ค่าของ 1 ไบต์จึงเป็นได้ตั้งแต่ 00000000 ถึง 11111111 หรือ 0-255 ในเลขฐาน 10 บิตที่อยู่ทางขวาสุดของไบต์เรียกว่า บิต 0 บิตที่อยู่ทางซ้ายสุดเรียกว่า บิต 7 บิต 0 เรียกว่าบิตที่มีนัยสำคัญต่ำสุด (least significant bit) และบิต 7 เรียกว่าบิตที่มีนัยสำคัญสูงสุด (most significant bit)

หมายเลขบิต	7	6	5	4	3	2	1	0
ค่าถ้าถูกเซต	128	64	32	16	8	4	2	1
การเซต	0	0	1	0	0	0	1	1
ค่าตามที่เซต	0	0	32	0	0	0	2	1

รูปที่ 2.8 แสดงจำนวน 35 ในฐานสอง

คอมพิวเตอร์เกือบทั้งหมดทำงานในระบบเลขฐานสอง เพราว่ามันเป็นการง่ายที่จะแปลงรหัส 0 และ 1 เป็นแรงดันไฟฟ้าบวกและลบ ในคอมพิวเตอร์ส่วนใหญ่ หน่วยเล็กที่สุดของหน่วยความจำที่อ้างถึงได้โดยการแอดเดรสคือไบต์ ดังนั้นเมื่อข้อมูลถูกเก็บและจัดการในคอมพิวเตอร์ตามปกติมันจึงถูกแปลงให้เป็นไบต์ที่เรียงลำดับกัน

2.4.3 การเข้ารหัสข้อความ

เมื่อข้อความ (อักขระ เครื่องหมายวรรคตอน และอื่นๆ) ถูกเก็บในคอมพิวเตอร์ แต่ละตัวอักษรที่แตกต่างกันจะถูกแทนด้วยจำนวนที่ต่างกัน จำนวนเหล่านี้โดยปกติมีค่าจาก 0 ถึง 127 หรือจาก 0 ถึง 255 เนื่องจากไบต์หนึ่งสามารถมีค่าจาก 0 ถึง 255 มันจึงเป็นธรรมชาติที่จะให้หนึ่งไบต์แทนตัวอักษรหรือเครื่องหมายวรรคตอนแต่ละตัวในข้อมูลที่เป็นข้อความ

มีสองวิธีที่ต่างกันสำหรับการจับคู่ตัวอักษรกับจำนวน คือ EBCDIC (Extended Binary Coded Decimal Interchange code) ซึ่งถูกใช้ในคอมพิวเตอร์ชนิดอื่นของไอบีเอ็มยกเว้นไอบีเอ็มพีซี และ ASII (American Standard Code for Information Interchange) ซึ่งถูกใช้ในคอมพิวเตอร์อื่นส่วนใหญ่ เราจะเกี่ยวข้องกับ ASCII เท่านั้นในหนังสือเล่มนี้เท่านั้น

ตาราง ASCII อย่างเป็นทางการให้จำนวนระหว่าง 32 ถึง 126 แทนตัวเลข ตัวอักษร เครื่องหมายวรรคตอนและสัญลักษณ์ที่ใช้กันทั่วไปอื่นๆ จำนวนจาก 0 ถึง 31 และ 127 มีความหมายพิเศษเช่น Carriage return , Line feed และตัวอักษรที่ไม่สามารถแสดงผลได้อื่นๆ

ตัวอย่างเช่น ตัว A ถูกเก็บเป็นเลขฐานสิบ 65 ในฐานสอง คือ 01000001 คอมมาถูกเก็บในเลขฐานสิบ 44 ซึ่งเป็น 00101100 ในฐานสอง

เนื่องจากจำนวน 127 ในฐานสองใช้เพียงเจ็ดบิต ตัวอักษรทั้งหมดถูแทนด้วย 0 ถึง 127 สามารถถูกเก็บในหนึ่งไบต์ โดยจะเหลืออีกหนึ่งบิต เนื่องจากเราใช้ชื่อบิตในไบต์ หนึ่งตั้งแต่ศูนย์ถึงเจ็ด จะเห็นได้ว่ารหัส ASCII ใช้เพียงบิตศูนย์ถึงหก บิตเจ็ดถูกสำรองไว้

คอมพิวเตอร์หลายชนิดใช้เต็มที่ทั้งแปดบิตสำหรับการเข้ารหัส ทำให้มีรหัสที่แตกต่างกัน 256 ตัว 128 ตัวแรกเป็นไปตาม ASCII และส่วนที่เหลือถูกใช้สำหรับอักขระต่างชาติ สัญลักษณ์ทางคณิตศาสตร์ อักขระ กราฟิก และอื่นๆ ตามแต่การออกแบบ โชคไม่ดีที่ไม่มีมาตรฐานสำหรับอักขระเพิ่มเติม (extend character) เหล่านี้ ซึ่งมักจะมีความหมายแตกต่างกันบนคอมพิวเตอร์ คนละชนิด

2.4.4 รหัสASCII ชนิดพิเศษ

รหัส 32 ตัวแรกในตาราง ASCII มีความหมายพิเศษ ดังในตารางที่2-12 มีหลายตัวที่ได้รับการออกแบบเพื่อวัตถุประสงค์ทางการสื่อสาร โดยเฉพาะ

ตาราง 2-12 รหัส ACSII พิเศษ

รหัส	อักขระ	ความหมาย
0	NULL	วิธีหนึ่งที่จะทำให้เกิดการหน่วงเวลาอย่างจงใจ ในอดีตมันมีความจำเป็นที่จะส่ง NULL หลังจากCarriage Return เพื่อให้เครื่องพิมพ์ปัดแคร่ไปทางซ้ายสุดของหน้ากระดาษ ปัจจุบันเครื่องพิมพ์ทำงานได้เร็วขึ้น NULL จึงถูกใช้สำหรับจุดประสงค์อื่นหลายอย่าง
1	SOH	Start of heading แสดงว่าข้อความที่ตามมาเป็นส่วนหนึ่งของหัวข้อ
2	STX	Start of text แสดงจุดสิ้นสุดของข้อความจริงของข่าวสาร
3	ETX	End of text แสดงจุดสิ้นสุดของข้อความ
4	EQT	End of transmission แสดงการสิ้นสุดของการส่ง
5	ENQ	Enquiry โดยปกติถูกใช้เป็นส่วนหนึ่งของ software แฮนด์เช็คกิ้ง ในการขอให้คอมพิวเตอร์ ฝ่ายรับตอบการได้รับข่าวสาร
6	ACK	Acknowledge การตอบรับการได้รับข่าวสาร
7	BEL	ส่งเสียงออกทางเทอร์มินัล
8	BS	Backspace
		Horizontal tab
9	HT	Horizontal tab
10	LF	Line feed ทำให้ขึ้นบรรทัดใหม่ในตำแหน่งเดิม
11	VT	Vertical tab
12	FF	Form feed เลื่อนหน้ากระดาษไปหนึ่งหน้า
13	CR	Carriage return เลื่อนไปที่ต้นบรรทัด บางครั้งทำให้เกิด Line feed ด้วยเช่นกัน
14	SO	Shift out กำหนดจุดเริ่มต้นของรหัสควบคุมพิเศษบ่อยครั้งที่ใช้ ESC แทน
15	SI	Switch in กำหนดจุดสิ้นสุดของรหัสควบคุมที่เริ่มต้นโดย SO

ตาราง 2-12 (ต่อ) รหัส ACSII พิเศษ

รหัส	อักขระ	ความหมาย
16	DLE	Data link escape เหมือนกับ ESC
17	DC1	
18	DC2	Device control 1 ถึง 4 รหัสที่สำรองไว้ให้ใช้ตามความต้องการบางครั้งใช้ใน software แฮนด์เช็คกิ้ง
19	DC3	
20	DC4	
21	NAK	Negative acknowledgement บ่งชี้ว่าข้อมูลที่ส่งนั้นไม่ได้ถูกรับอย่างถูกต้อง ตัวอย่าง เช่น พบความผิดพลาดทางพาริตี้
22	SYN	Synchronous idle เหมือนกับ NULL แต่ถูกใช้ในการสื่อสารแบบซิงโครนัส เพื่อดูแลให้อุปกรณ์สองตัว ซิงโครไนซ์กันระหว่างส่ง
23	ETB	End of transmission block ถูกใช้ในที่ซึ่งการส่งข้อมูลถูกแบ่งเป็นบล็อค เพื่อวัตถุประสงค์ในการตรวจข้อผิดพลาด
24	CAN	Cancel บ่งชี้ว่า ข้อมูลถูกส่งไปควรถูกทิ้งไป
25	EM	End of medium บ่งชี้ว่ามาถึงปลายของเทปกระดาษ
26	SUB	Substitute แก้ไขตัวอักษรที่ถูกส่งมาผิดพลาด ถูกใช้เพื่อบ่งชี้จุดสิ้นสุดของการส่งด้วยเช่นกัน
27	Esc	Escape บ่งชี้จุดเริ่มต้นของตัวอักษรที่ติดตามมาว่ามีความพิเศษ
28	FS	
29	GS	File group , Record และ Unit separator ตามลำดับ ใช้เพื่อกำหนดขอบเขตระหว่างส่วนของข้อความ
30	RS	
31	US	
32	DEL	บ่งชี้ว่า ตัวอักษรที่มาก่อนมันควรถูกลบ

รหัส 1 ถึง 26 ถูกอ้างถึงเป็น Ctrl – A ถึง Ctrl – Z ด้วยเช่นกัน และพวกมันสามารถถูกสร้างด้วยแป้นพิมพ์ของคอมพิวเตอร์ โดยการกดปุ่ม Ctrl ค้างไว้ และกดปุ่มตัวอักษรที่เหมาะสมพร้อมกัน (ดังนั้น 1 = Ctrl – A , 2 = Ctrl – B เป็นต้น) บางรหัสสามารถถูกป้อนเข้าโดยการกดปุ่มเฉพาะ เช่น TAB สำหรับ รหัส 9 หรือ Return สำหรับรหัส 13

2.4.5 การเข้ารหัสข้อมูลที่ไม่ใช่ข้อความ

แน่นอนว่าทุกอย่างที่ถูกเก็บในคอมพิวเตอร์ไม่ได้อยู่ในรูปของข้อความเสมอไป คำสั่งของโปรแกรม ข้อมูล และกราฟฟิกอิมเมจ เป็นตัวอย่างข้อมูลที่ไม่ได้ถูกเก็บในรูปแบบ ASCII

ข้อมูลประเภทนี้โดยปกติถูกเข้ารหัสให้ใช้ทุกค่าที่เป็นไปได้ของหนึ่งไบต์ จำนวนถูกเก็บในรูปแบบไบนารี และสามารถขยายไปเป็นหลายไบต์ คำสั่งของโปรแกรมมักจะประกอบด้วยหนึ่งหรือสองไบต์ เราเรียกข้อมูลประเภทนี้ว่า ข้อมูลไบนารี (Binary Data) แม้ว่าข้อความจะถูกเก็บในรูปแบบไบนารีเช่นกัน

เนื่องจากไบต์ที่เก็บข้อมูลซึ่งไม่ใช่ข้อความสามารถเป็นค่าใด ๆ ก็ได้ ในเวลาที่มันตรงกับค่าที่มีความหมายพิเศษในตาราง ASCII ทำให้เกิดความยุ่งยากในการส่งข้อมูล ถ้าอุปกรณ์ฝ่ายรับเกิดแปลไบต์ที่ไม่ใช่ข้อความว่าหมายถึงสิ้นสุดข่าวสาร ในกรณีนี้ข้อมูลไม่สามารถถูกส่งในรูปแบบข้อมูลดิบ เพราะว่าไบต์ที่อยู่กลางข่าวสารอาจตรงกับสัญลักษณ์สิ้นสุดข่าวสารโดยบังเอิญและทำให้อุปกรณ์ฝ่ายรับหยุดรับส่งข้อมูล

2.4.6 การสื่อสารแบบอนุกรม

คอมพิวเตอร์เกือบทั้งหมดเก็บและจัดการข้อมูลในแบบขนาน หมายความว่าเมื่อไบต์หนึ่งถูกส่งจากส่วนหนึ่งของคอมพิวเตอร์ไปยังส่วนอื่นมันไม่ได้ถูกส่งไปครั้งละหนึ่งบิต แต่จะถูกส่งไปหลายบิตพร้อมกันผ่านตัวนำในแบบขนาน จำนวนบิตที่ถูกส่งในครั้งหนึ่งแปรผันไปตามเครื่อง แต่โดยปกติจะเป็นแปดหรือทวีคูณของแปด เพราะฉะนั้นคอมพิวเตอร์สามารถทำงานกับหนึ่งไบต์เป็นอย่างน้อยนครั้งหนึ่ง

เนื่องจากการสื่อสารจากคอมพิวเตอร์ไปยังอุปกรณ์อื่นหลายชนิดเป็นแบบอนุกรม หมายความว่าข้อมูลถูกส่งไปทีละหนึ่งบิต ตัวเชื่อมต่อการสื่อสารต้องสามารถนำไบต์ที่รับมาแบบขนานส่งออกไปทีละบิตได้

จากที่กล่าวมาแล้วว่าสายข้อมูลในการสื่อสารแบบอนุกรม มีเพียงสภาวะ MARK และ SPACE ซึ่งในกรณีของการเชื่อมต่อโดยตรงเท่ากับแรงดันไฟฟ้าลบหรือบวกตามลำดับ ข้อมูลใด ๆที่ถูกส่งต้องถูกแปลงให้เป็นลำดับของ MARK และ SPACE ก่อน สำหรับการส่งข้อมูล MARK แทนค่าหนึ่ง และ SPACE แทนค่าศูนย์

2.4.6.1 การสื่อสารแบบซิงโครนัสและอะซิงโครนัส

เมื่อข้อมูลถูกแปลงให้เป็นรูปแบบอนุกรมแล้ว มีวิธีในการส่งข้อมูลอยู่ 2 อย่าง คือ ซิงโครนัส (Synchronous) และอะซิงโครนัส (Asynchronous)

เมื่อข้อมูลถูกส่งมาจากการพิมพ์ที่แป้นพิมพ์ การส่งและรับจะเป็นแบบอะซิงโครนัส คือคนที่พิมพ์ไม่สามารถที่จะพิมพ์ได้อย่างต่อเนื่อง ดังนั้นเมื่อคอมพิวเตอร์รับตัวอักษรแต่ละตัวจะมีช่องว่างระหว่างตัวอักษรที่ไม่สม่ำเสมอ ทำให้อุปกรณ์ฝ่ายรับไม่อาจคาดหมายได้ว่า ตัวอักษรต่อไปจะมาถึงเมื่อใด จากการขาดความต่อเนื่องนี้จึงมีความจำเป็นต้องใส่บิตพิเศษก่อนและหลังตัวอักษรแต่ละตัวเพื่อบ่งบอกจุดเริ่มต้นและสิ้นสุดของตัวอักษรบิตพิเศษนี้ว่า บิตเริ่มต้น (Start Bit) นอกจากนี้ยังมีอีกหนึ่งคือ บิตพาริตี้ (Parity Bit) ที่มักจะถูกใส่เพิ่มเข้าไปเพื่อใช้ตรวจสอบความผิดพลาด วิธีนี้เรียกว่า การสื่อสารแบบอะซิงโครนัส (Asynchronous Communication)

เมื่อตัวอักษรถูกส่งไปเป็นกลุ่มตามความเร็วของเครื่อง ช่วงห่างระหว่างกันก็จะสม่ำเสมอจึงไม่มีความจำเป็นต้องมีบิตเริ่มต้นและบิตจบสำหรับตัวอัษรแต่ละตัว เพราะว่าเมื่อตัวอักษรแรกถูกรับไป อุปกรณ์ฝ่ายรับสามารถคาดหมายการมาถึงของตัวอักษรถัดไปได้ กล่าวอีกหนึ่งคือ มันสามารถเข้าจังหวะตัวมันเองกับคอมพิวเตอร์ฝ่ายส่งได้ วิธีแบบนี้เรียกว่า การสื่อสารแบบซิงโครนัส (Synchronous Communication)

เนื่องจากการสื่อสารแบบอะซิงโครนัสต้องการบิตเริ่มต้นและบิตจบเพิ่มเข้าไปในแต่ละตัวอักษร จึงมีความยาวในการส่งไฟล์มากกว่าการสื่อสารแบบซิงโครนัส ประมาณ 20 เปอร์เซ็นต์ ความแตกต่างนี้อาจสังเกตไม่เห็นเมื่อแหล่งข้อมูลที่ส่งมาจากการพิมพ์ที่เทอร์มินอล

นอกจากในโลกของไอบีเอ็มเมนเฟรมซึ่งเทอร์มินอลแบบซิงโครนัสเป็นอุปกรณ์สามัญ การสื่อสารแบบอนุกรมส่วนใหญ่เกิดขึ้นในแบบอะซิงโครนัส ซึ่งประยุกต์เข้ากับการสื่อสารเกือบทั้งหมดระหว่างไมโครคอมพิ

การจัดเฟรม

ในกรณีการสื่อสารแบบอะซิงโครนัส บิตที่เป็นตัวแทนของหนึ่งไบต์ ซึ่งเรียกว่า บิตข้อมูล (Data Bit) จะถูกนำและตามด้วยบิตเริ่มต้น บิตจบและบิตพาริตี้ กระบวนการนี้เรียกว่า การจัดเฟรม (Framing)

จำนวนของบิตที่แทนหนึ่งตัวอักษรแปรผันไปตามโพรโตคอลสื่อสารที่ใช้ จำนวนที่ว่าหมายถึงจำนวนของบิตของข้อมูล หรือความยาวเวิร์ด (Word length) โดยปกติจะเป็นเจ็ดหรือแปดบิต แต่ละตัวอักษรจะถูกส่งออกไปเป็นกลุ่มที่ประกอบด้วยบิตเริ่มต้น ตัวอักษร (บิตข้อมูล) บิตพาริตี้ซึ่งสามารถเลือกได้และบิตจบหนึ่งถึงสองบิตเพื่อความชัดเจน เราจะเรียกกลุ่มของตัวอักษรและบิตเหล่านี้ว่า เฟรม (Frame) เพื่อหลีกเลี่ยง ความสับสนกับคำว่าตัวอักษร ที่บางครั้งอ้างถึงบิตข้อมูลและบางครั้งอ้างถึงบิตข้อมูลและบางครั้งอ้างถึงทั้งกลุ่มพร้อมด้วยบิตเริ่มต้น บิตจบ และบิตพาริตี้ ตัวอย่างของเฟรมที่ถูกส่งแสดงไว้ในรูป

รูปที่ 2.9 ตัวอย่างการส่งตัวอักษร A สองแบบ

บิตเริ่มต้น (Start Bit)

บิตเริ่มต้นถูกใส่เพิ่มที่จุดเริ่มต้นของเฟรมเสมอ เพื่อเตือนอุปกรณ์ฝ่ายรับว่าข้อมูลกำลังมาถึง และเพื่อเข้าจังหวะกลไกที่แยกแต่ละบิต บิตเริ่มต้นคือ SPACE หรือ ไบนารี 0

ในการเชื่อมต่อโดยตรง SPACE หรือ 0 ถูกส่งเป็นแรงดันไฟฟ้าบวก แรงดันไฟฟ้าระหว่างเฟรมจะเป็นลบ ดังนั้นที่จุดเริ่มต้นของแต่ละเฟรมแรงดันไฟฟ้าจะเปลี่ยนจากลบเป็นบวก

บิตข้อมูล (Data Bit)

มาตรฐานหรือโพรโตคอลการสื่อสารแบบอนุกรม ทำให้เกิดการส่งตัวอักษรที่ยาวต่างกันเมื่อซอฟต์แวร์สื่อสารให้คุณเลือกความยาวเวิร์ด มันกำลังถามว่าคุณต้องการส่งตัวอักษรเจ็ดบิตหรือแปดบิต (บางครั้งความยาวอื่นก็ถูกใช้

แต่แทบจะไม่ค่อยมี) ถ้าข้อมูลทั้งหมดถูกส่งในรูปแบบ ASCII เวิร์ดขนาดเจ็ดบิตก็เพียงพอ จำไว้ว่าตาราง ASCII กำหนดจำนวนจาก 0 ถึง 127 ซึ่งทั้งหมดสามารถแทนได้ด้วยเจ็ดบิต

ถ้าข้อมูลที่ส่งไม่ใช่ ASCII (เช่น ข้อความที่ใช้ชุดตัวอักขระเพิ่มเติมหรือข้อมูลไบนารี) ทั้งแปดบิตของแต่ละไบต์จึงมีความจำเป็น คุณไม่สามารถใช้โพรโตคอลเจ็ดบิตได้ ถ้าข้อมูลไม่ถูกแปลงเป็นรูปแบบเจ็ดบิตเสียก่อน

บิตพาริตี้ (Parity Bit)

การตรวจสอบพาริตี้เป็นวิธีหนึ่งในการทดสอบว่า ข้อมูลที่ส่งได้ถูกรับไปอย่างถูกต้องหรือไม่ อุปกรณ์ฝ่ายส่งจะเพิ่มบิตพาริตี้อีกหนึ่งบิตเป็นค่า 0 หรือ 1 ขึ้นอยู่กับบิตข้อมูล อุปกรณ์ฝ่ายรับจะตรวจสอบว่าบิตพาริตี้มีความสัมพันธ์ที่ถูกต้องกับบิตอื่นหรือไม่ ถ้าไม่แสดงว่าบางสิ่งต้องผิดพลาดในระหว่างการส่ง พาริตี้สามารถคำนวณได้จากวิธีต่อไปนี้

พาริตี้คู่ (Even Parity) หมายความว่า จำนวนของบิตข้อมูลที่เป็น 1 และค่าของบิตพาริตี้รวมกัน เป็นจำนวนคู่ เช่น ตัว A ในฐานสองคือ 01000001 เมื่อนับจำนวนของบิตที่ได้เป็น 1 จะได้ 2 ซึ่งเป็นเลขคู่ ดังนั้นบิตพาริตี้ต้องเป็น 0 ถ้าตัวอักษร A ที่รับได้มีพาริตี้เป็น 1 แสดงว่าเกิดความผิดพลาดในระหว่างการส่ง

พาริตี้คี่ (Odd Parity) หมายความว่า จำนวนทั้งหมดของบิตข้อมูลที่เป็น 1 บวกกับค่าของบิตพาริตี้เป็นจำนวนคี่ ดังนั้นสำหรับตัวอักษร A บิตพาริตี้ควรถูกเซตเป็น 1 เพื่อให้จำนวนของบิตที่เป็น 1 ทั้งหมดเป็น 3 ซึ่งเป็นจำนวนคี่

ไม่มีพาริตี้ (Null Parity) หมายถึง ไม่มีบิตพาริตี้

SPACE (บางครั้งเรียกว่า Bit Trimming) คือบิตพาริตี้ที่เป็น 0 เสมอ มีประโยชน์ในการตรวจสอบข้อผิดพลาดบางอย่าง เมื่อการส่งข้อมูลเป็นขยะมาก บางครั้งบิตพาริตี้อาจกลายเป็น 1 แสดงว่า เกิดข้อผิดพลาด พาริตี้แบบนี้สามารถใช้เพื่อส่งอักษรเจ็ดบิตให้กับอุปกรณ์ที่ต้องการตัวอักษรแปดบิตได้เช่นกัน อุปกรณ์ฝ่ายรับจะถือว่าบิตพาริตี้เป็นบิตสุดท้ายของข้อมูล

MARK (บางครั้งเรียกว่า Bit Forcing) ทำงานเหมือนกับพาริตี้แบบ SPACE ยกเว้นแต่บิทพาริตี้จะเป็น 1 เสมอ เนื่องจาก 1 ในตำแหน่งนั้นสามารที่จะถูกตีความรวมเข้ากับค่าของจำนวนได้ อุปกรณ์ หรือ คอมพิวเตอร์ฝ่ายรับต้องถูกโปรแกรมไม่ให้สนใจมัน

บิตจบ (Stop Bit)

ที่ท้ายของแต่ละเฟรม บิตจบจะถูกส่งออกมา บิตจบมีทั้งแบบหนึ่งบิต หนึ่งบิตครึ่ง หรือสองบิต อย่างน้อยต้องมี 1 บิตเสมอ เพื่อประกันว่ามีแรงดันไฟฟ้าลบอย่างน้อยเป็นช่วงเวลาหนึ่งก่อนที่เฟรมถัดไปจะมาถึง เพื่อที่จะสามารถแยกแยะเฟรมถัดไปได้จากบิตเริ่มต้นที่เป็นบวกของมัน บิตจบมากกว่า1 บิตโดยทั่วไปจะใช้เมื่ออุปกรณ์ฝ่ายรับต้องการเวลาเพิ่มขึ้นก่อนที่มันจะสามารถจัดการกับตัวอักษรที่เข้ามาตัวถัดไปได้

หนึ่งบิตครึ่ง หมายความว่า ความยาวของบิตนั้นมากกว่าบิตปกติ บิตจบบังคับให้มีช่องว่างอย่างน้อยระหว่างเฟรม พวกมันถูกส่งเป็นไบนารีหนึ่งซึ่งในการเชื่อมต่อโดยตรงจะเป็นแรงดันไฟฟ้าลบ

บิตจบสองบิตมักจะถูกใช้ที่อัตราบอด 110 ซึ่งเป็นอัตราการส่งข้อมูลต่ำสุดที่ใช้กันทั่วไป เพื่อให้สอดคล้องกับความต้องการของเครื่องโทรพิมพ์รุ่นเก่า ซึ่งใช้อัตราบอดต่ำและต้องการเวลาพิเศษเพื่อประมวลตัวอักษร

เบรก

ดังที่อธิบายมาก่อนเมื่อกล่าวถึงบิตเริ่มต้นว่า ระหว่างตัวอักษร สายข้อมูลโดยปกติอยู่ในสภาวะ MARK (แรงดันไฟฟ้าลบ , ไบนารีหนึ่ง) ถ้าตัวอักษรประกอบด้วยศูนย์ทั้งหมด พร้อมด้วยแปดบิตข้อมูลและพาริตี้คู่ สภาวะ SPACE จะปรากฏอยู่ สิบบิตคือ บิตเริ่มต้น บิตข้อมูลทั้งแปดและบิตพาริตี้ ซึ่งเป็นสภาวะ SPACE ที่ยาวที่สุด ก่อนจะสิ้นสุดเมื่อถึงบิตจบ ดังนั้นที่อัตรา 150 บิตต่อวินาที สภาวะ SPACE ตามปกติจะไม่มากไปกว่า 1/15 วินาที หรือ 66.67 มิลลิวินาที

สภาวะ SPACE ที่นานกว่านี้ โดยปกติเป็น 100 ถึง 600 มิลลิวินาที ถูกใช้เป็นสัญญาณพิเศษเรียกว่า เบรก (Break) เบรกบางครั้งถูกใช้เสมือนกับ Ctrl-C ของเมนเฟรมบนพีซี มันจะขัดจังหวะไม่ว่าโปรแกรมอะไรกำลังทำงานอยู่และกลับคืนสู่ระบบปฏิบัติการ หรือ เมนูที่อยู่ในระบบบนภายในโปรแกรม เช่นเดียวกับ Ctrl – C หรือ Break มันมีประโยชน์สำหรับการหนีออกจากโปรแกรมที่เข้ามาลูปไม่รู้จบ

อัตราบอด

อัตราบอด (Baud Rate) แสดงจำนวนของสัญญาณแต่ละหน่วยในหนึ่งหน่วยวินาที มันถูกตั้งชื่อตาม Baudot ซึ่งเป็นผู้บุกเบิกการสื่อสารชาวฝรั่งเศส ในการส่งแบบไบนารีมันเป็นสิ่งเดียวกับบิตต่อวินาที (bps) หรือ จำนวนของเลขฐานสองที่ถูกส่งในหนึ่งวินาที ทั้งสองคำนี้มีความแตกต่างกัน แต่มักจะทำให้สับสน ผู้คน 200,000 คน อาจบอกว่าพวกเขามีโมเด็ม 1200 บอด และไม่มีสักคนที่มีจริง ๆ ที่จริงแล้วพวกเขามีโมเด็ม 1200 bps

ในการเชื่อมต่อ RS-232 โดยตรง สัญญาณจะเป็นหนึ่งในสองสถานะ ในเวลาขณะใดขณะหนึ่ง อัตราบอดและ bps จึงเท่ากัน อย่างไรก็ตาม จะเห็นได้ว่าเมื่อสัญญาณหนึ่งถูกส่งผ่านระหว่างโมเด็ม มันสามารถเป็นหนึ่งในหลายสถานะ ความยาวของสัญญาณอาจเป็น 1/600 วินาที (600 บอด) แต่เนื่องจากมากกว่าสองบิตของข้อมูลสามารถถูกส่งไปพร้อมกับการเปลี่ยนแปลงสภาวะ อัตราบิตต่อวินาทีจะสูงกว่าอัตราบอด

มีจุดที่น่าสังเกตคือทั้งอัตราบอด และ bps อ้างถึงอัตราที่บิตภายในหนึ่งเฟรมถูกส่ง ช่องว่างระหว่างเฟรมอาจมีความยาวแปรเปลี่ยนได้ เช่น จากการพิมพ์ตัวอักษรด้วยอัตราแตกต่างกัน ดังนั้นทั้งอัตราบอดและ bps จึงไม่ได้หมายถึงอัตราที่ข้อมูลถูกส่งไปจริง ๆ

2.4.6.2 ข้อจำกัดในการส่งข้อมูลผ่านพอร์ตอนุกรม

เมื่ออุปกรณ์สองตัวสื่อสารซึ่งกันและกัน พวกมันต้องตกลงกันในเรื่องของอัตราบอด ความยาวเวิร์ด จำนวนบิตจบ และพาริตี้ ถ้าพบว่าไม่ได้รับอะไรเลย ความผิดพลาดอาจอยู่ที่การเชื่อมต่อทางกายภาพ เช่น ข้อมูล กำลังถูกส่งบนสายผิดเส้น สายขาด หรือไม่ได้รับสัญญาณแฮนด์เช็คกิ้งที่ถูกต้อง ถ้าได้รับขยะ ความผิดพลาดอาจอยู่ในหัวข้อที่จะกล่าวต่อไปนี้

อัตราบอดไม่ตรงกัน

ถ้าอุปกรณ์ 2 ตัวถูกตั้งอัตราบอดต่างกัน อุปกรณ์ฝ่ายรับอาจพยายามที่จะแปรข้อมูล (ถ้ามันไม่ได้ถูกโปรแกรมให้รายงานข้อผิดพลาดทางพาริตี้และทางเฟรม) โดยปกติคุณจะเห็นว่าจำนวนข้อมูลที่รับได้แตกต่างจากที่ส่งมา

ความผิดพลาดทางพาริตี้

ความผิดพลาดทางพาริตี้ (Parity Error) บ่งบอกว่าข้อมูลถูกทำลายในระหว่างการส่ง อย่างไรก็ตาม มันอาจหมายความว่าอุปกรณ์ทั้งสองไม่ได้ถูกตั้งให้มีพาริตี้ (คู่ , คี่ หรือ ไม่มี) หรือความยาวเวิร์ดตรงกัน

ความยาวเวิร์ดไม่ตรงกัน

ถ้าเวิร์ดขนาดแปดบิต กำลังถูกส่งและอุปกรณ์ฝ่ายรับคาดหวังที่จะรับเวิร์ดขนาดเจ็ดบิต คุณอาจไม่พบความแตกต่างในการส่งข้อความ เพราะว่าเพียงแค่เจ็ดบิตแรกที่มีนัยสำคัญ เนื่องจากบิตศูนย์ถูกส่งก่อนและบิตเจ็ดไม่ถูกใช้ในการส่ง ASCII ปกติการขาดหายไปของมันจึงไม่มีความสำคัญอย่างไรก็ตาม อุปกรณ์ฝ่ายรับอาจพยายามแปลความหมายบิตที่เกินมาเป็นบิตพาริตี้คี่ และรายงานข้อผิดพลาด ดังนั้นข้อผิดพลาดทางพาริตี้จึงไม่จำเป็นที่จะต้องหมายถึงข้อมูลถูกทำลายในการส่ง มันอาจบอกถึงความยาวเวิร์ดไม่ตรงกันก็ได้

ถ้าส่งเวิร์ดขนาด 7 บิตโดยที่อุปกรณ์ฝ่ายรับต้องการเวิร์ดขนาด 8 บิต บิตพาริตี้อาจถูกนำไปรวมเป็นบิตที่ 8 เนื่องจากบิตพาริตี้อาจเป็น 1 สำหรับตัวอักษรครึ่งหนึ่ง และเป็น 0 สำหรับครึ่งหนึ่ง จึงพบได้บ่อยครั้งว่าอุปกรณ์ฝ่ายรับจะแสดงอักขระเพิ่มเติม เช่น อักขระกราฟิกในจำนวนครึ่งหนึ่งของตัวอักษรที่รับได้

บิตจบ

ไม่ควรจะมีปัญหาถ้าบิตจบสองบิตถูกส่งมา แม้มีเพียงบิตเดียวที่ต้องการ บิตจบที่เกินมาเพียงแต่รวมเข้าในช่องว่างระหว่างตัวอักษร อย่างไรก็ตามการส่ง 1 บิตจบ เมื่อต้องการ 2 บิต อาจทำให้เกิดปัญหาขึ้นอยู่กับคุณลักษณะของอุปกรณ์ฝ่ายรับ เรื่องนี้ไม่เป็นปัญหากับอุปกรณ์โมเด็ม

ความผิดพลาดทางเฟรม

ความผิดพลาดทางเฟรม บ่งบอกความไม่ตรงกันของจำนวนบิต ซึ่งมักจะถูกรายงาน เมื่อไม่ได้รับบิตจบตามที่คาดหวัง

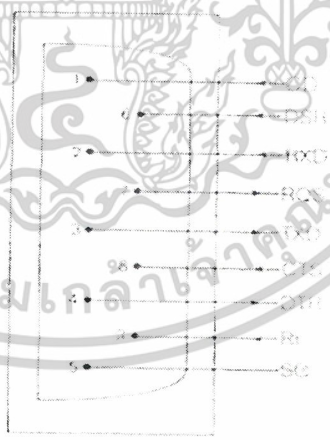
2.4.7 สัญญาณทางไฟฟ้า

มาตรฐาน RS-232C กำหนดคุณลักษณะของสัญญาณทางไฟฟ้าที่ถูกใช้ในการเชื่อมต่ออนุกรมโดยตรง มีเพียง 2 ลักษณะคือ SPACE แสดงถึง ไบนารี 0 หรือแรงดันไฟฟ้าบวกและ MARK แสดงถึงไบนารี 1 หรือแรงดันไฟฟ้าลบ

บนสายข้อมูล (เช่น สาย 2 และ 3) แรงดันไฟฟ้าบวกแสดงถึงค่าลอจิก 0 และแรงดันไฟฟ้าลบแสดงถึงค่าลอจิก 1 บนสายแฮนด์เช็คกิ้ง (เช่น DTR และ DRS)แรงดันไฟฟ้าบวกแสดงว่า ส่งข้อมูลได้ส่วนแรงดันไฟฟ้าลบหมายถึงหยุดส่งข้อมูล

แรงดันไฟฟ้าบวก (สถานะ SPACE) อยู่ระหว่าง +5 ถึง –5 V สำหรับเอาท์พุท และระหว่าง +3 ถึง +15 V สำหรับอินพุท ความแตกต่างมีไว้เผื่อกรณีที่แรงดันไฟฟ้าสูญหายเนื่องจากความยาวของสายสัญญาณ ในทำนองเดียวกันแรงดันไฟฟ้าลบ (สถานะ MARK) ถูกกำหนดไว้ระหว่าง –5 ถึง –15 สำหรับเอาท์พุท และ –3 ถึง –15 สำหรับ อินพุท

สังเกตว่า ถ้าสัญญาณยาวเกิน ไประดับแรงดันไฟฟ้าจะตกลงเกินขอบเขตที่ยอมรับได้นอกจากนี้ ความจุไฟฟ้าที่เกิดขึ้นจะมีผลกับคุณภาพของสัญญาณ โดยทำการเปลี่ยนสถานะจากแรงดันไฟฟ้าบอกไปลบไม่ชัดเจน RS-232-C ไม่ได้มุ่งหวังนำไปใช้กับระยะไกล และโดยทั่วไป 50 ฟุตเป็นระยะทางไกลที่สุด ในการใช้สัญญาณปกติที่อัตราส่งข้อมูลปกติ ถ้าอุปกรณ์อยู่ห่างกันมาก อาจจำเป็นต้องใช้โมเดม หรือวิธีการอื่น



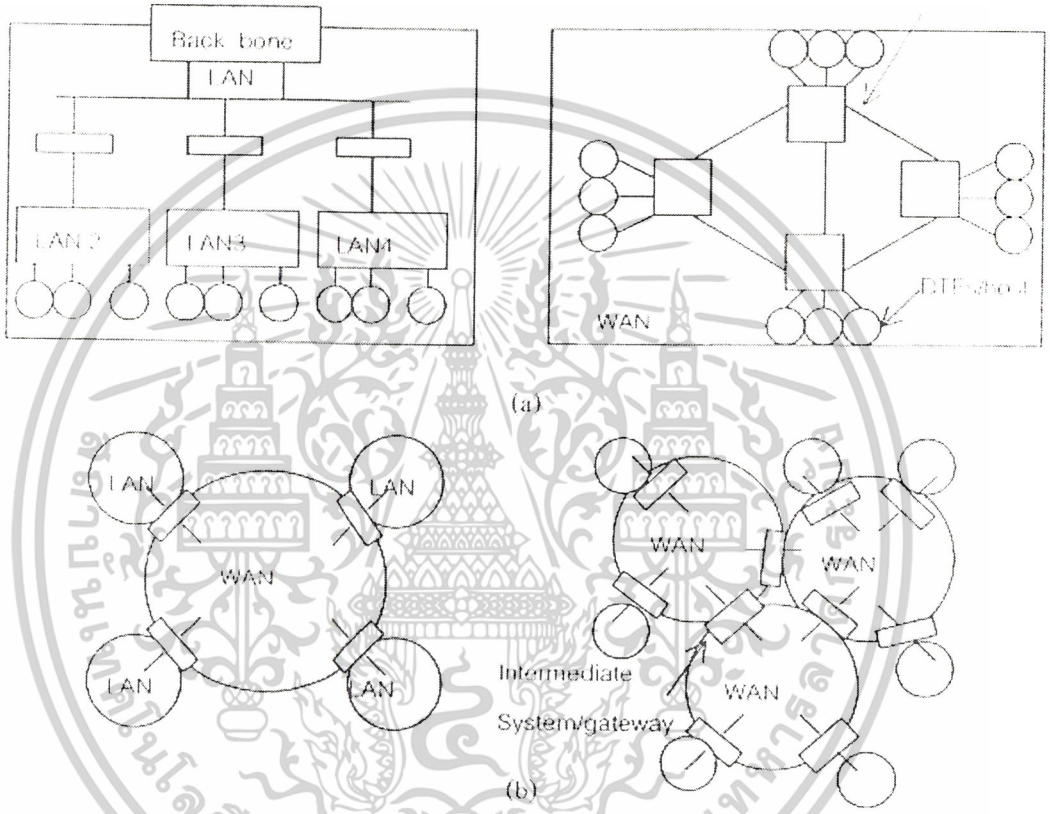
รูปที่ 2.10 หัวต่อแบบ DB-9 ในไอบีเอ็มพีซีเอที

2.5 เครือข่ายอินเตอร์เน็ต

Internet คือ การที่เครือข่าย 2 หรือมากกว่า เชื่อมต่อเข้าด้วยกัน และทำงานเสมือนเป็นเครือข่ายเดียวกัน โดย network ที่เป็นส่วนประกอบของ internet คือ Subnetwork (Subnet) ซึ่งอาจจะเป็นเครือข่าย Local area network (LAN) หรือ Wide area network (WAN) อุปกรณ์ที่ใช้ในการเชื่อมต่อ 2 เครือข่าย เข้าด้วยกันก็คือ intermediate system (IS) หรือ internetworking unit (IWU) การเชื่อมโยงระหว่างระบบที่แตกต่างกัน จำเป็นต้องมีมาตรฐานในการติดต่อกัน ซึ่งเรียกเป็นศัพท์เฉพาะว่า โพรโตคอล (Protocal)

2.5.1 สถาปัตยกรรมอินเตอร์เน็ต (Internet Architectures)

สถาปัตยกรรมพื้นฐานของ Internet แสดงดังรูป

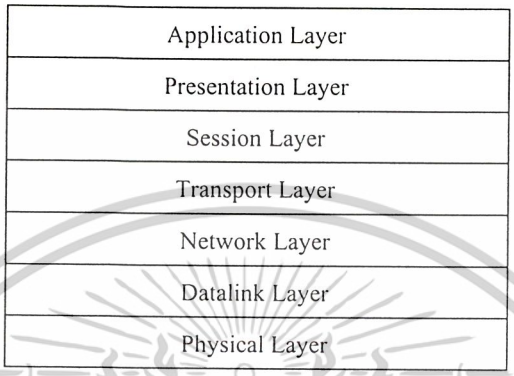


รูปที่ 2.11 แสดงสถาปัตยกรรมของ Inernet (a) Single LAN and WAN (b) Interconnected LAN / WAN

ในรูป (a) แสดงตัวอย่าง 2 ตัวอย่างของเครือข่ายเดี่ยว (Single Network) ซึ่งอย่างแรกเป็น Site-Wide LAN ซึ่งประกอบขึ้นมาจากชุดของ LANS ซึ่งถูกต่อเข้ากับเครือข่ายหลัก (Backbone) ซึ่งอุปกรณ์ที่ใช้ต่อ LAN เข้ากับเครือข่ายหลัก ถ้า LAN ทุกเครือข่ายมีระบบเดียวกันก็จะใช้ Bridge ถ้าเป็น LAN ที่แตกต่างกันก็จะใช้ Router ตัวอย่างที่ 2 เป็นตัวอย่างของ WAN เดี่ยว ๆ ในรูปของ (b) แสดงถึงเครือข่ายอินเตอร์เน็ต ซึ่งประกอบด้วย Network ทั้ง 2 ชนิดข้างต้น

2.5.2 OSI โมเดล

องค์กรมาตรฐานสากล ISO (International Organization for Standardization) ได้กำหนดมาตรฐานของเครือข่าย โดยจัดแบ่งกิจกรรมของเครือข่าย ออกเป็นงานย่อย ๆ และกำหนดโมเดลแบ่งเป็นชั้น ๆ ตามลำดับเรียกว่ามาตรฐาน OSI (Open System Interconnection) โดยที่จะแบ่งกิจกรรมที่ซับซ้อนในเครือข่าย ออกเป็นงานย่อย ๆ ก็จะช่วยในการออกแบบ และการใช้งานเครือข่ายรวมถึงการเชื่อมโยงกัน เป็นไปได้ด้วยความสะดวก และวิธีการทำงานอยู่ในกรอบเดียวกัน ดังในรูปที่ 2.11



รูปที่ 2.12 แสดงการแบ่งการทำงานของเครือข่ายออกเป็น OSI Model

ในแต่ละชั้นของ OSI Model จะมีการติดต่อสื่อสารกันเป็นชั้น ๆ ตามลำดับลงมาเช่น Application Layer ก็จะติดต่อสื่อสารกับ Presentation Layer ตามลำดับไปจนถึงชั้นแรกสุดคือ Physical Layer

Application Layer เป็นชั้นบนสุดของโมเดล เป็นส่วนที่จะทำให้การติดต่อระหว่างเครือข่ายกับผู้ใช้ เป็นไปได้ตามต้องการ ตัวอย่างแอปพลิเคชั่นของเครือข่าย เช่น ระบบ E-mail , การโอนถ่ายข้อมูล (File Transfer) , การขอเข้าใช้ระบบคอมพิวเตอร์ในเครือข่าย เป็นต้น

Presentation Layer มีการกำหนดหน้าที่ไม่ชัดเจนนักและมีการนำไปใช้ไม่มาก ซึ่งหน้าที่หลักก็คือ เป็นส่วนที่จัดรูปแบบและนำเสนอข้อมูล ให้เป็นไปตามต้องการ รวมไปถึงการจัดแปลงข้อมูล ในรูปมาตรฐาน ASCII หรือ EBCDIC ,การลดขนาดข้อมูล (Data Compression) การเข้ารหัส หรือถอดรหัสของข้อมูล แต่ส่วนใหญ่แล้วแอปพลิเคชั่นจะจัดการแทนได้

Session Layer เป็นชั้นที่จัดการในเรื่อง การติดต่อแต่ละครั้ง หรือ Session ให้ระบบคอมพิวเตอร์ทั้ง 2 ฝั่ง โดยทำหน้าที่ตั้งแต่เริ่มการติดต่อ ดูแลในการส่งผ่านข้อมูล ในการติดต่อครั้งนั้น ๆ เป็นไปได้โดยไม่มีปัญหา จนถึงเลิกการติดต่อเมื่อเสร็จงาน

Transport Layer ทำหน้าที่ควบคุมปริมาณ และรายละเอียดวิธีการรับส่งข้อมูล ให้เป็นไปตามที่กำหนดไว้ และจัดการให้การเชื่อมโยงเครือข่ายเป็นไปด้วยความราบรื่น Transport Layer จะเป็นชั้นสุดท้ายที่จัดการเรื่องเส้นทางในการส่งข้อมูล และจัดการตรวจสอบความผิดพลาดของข้อมูลซึ่งส่วนของ TCP (Transmission Control Protocol) ในโพรโตคอล TCP/IP ทำงานที่ระดับนี้

Network Layer ทำหน้าที่ควบคุมวิธีการส่งผ่านข้อมูลระหว่างเครือข่ายให้ถูกต้อง และเป็นไปตามเส้นทางที่กำหนด โดยจะจัดการส่งผ่าน Packet ข้อมูล ผ่านอุปกรณ์ต่าง ๆ ไปยังเครือข่ายย่อยได้อย่างถูกต้องตามที่ต้องการ นอกจากนี้ยังจัดการดูแลเส้นทางในการส่งข้อมูล (Routing Table) และ กลั่นกรอง Packet ข้อมูลที่ส่งไปยังเครือข่ายเดียวกัน ไม่ให้ข้ามไปยังเครือข่ายอื่น ซึ่งจะช่วยลดปริมาณข้อมูลที่วิ่งบนเครือข่ายได้ส่วนหนึ่ง โพรโตคอล IP , TCP/IP และ IP x เป็นโพรโตคอลที่ทำงานอยู่ใน Layer นี้

Data Link Layer ทำหน้าที่เรียกใช้หรือกำหนดช่องทาง ในการส่งข้อมูลที่ถูกต้อง เช่น Ethernet , Tokenring หรือ FDDI เป็นต้น รวมถึงการลำดับและอัตราการรับส่งข้อมูลหรือ Flow Control และสถานที่ที่จะส่งข้อมูลไป (Address) ทั้งนี้ Data Link Layer จะเป็นชั้นแรกที่จัดการแปลงข้อมูลจาก Bit ให้เป็น Packet โดยจะมีการเพิ่มข้อมูลเพื่อตรวจสอบความถูกต้อง ในกรณีที่ส่งข้อมูลออกไป หรือ กรณีที่อ่านข้อมูลเข้ามา ก็จะตรวจสอบผ่าน Checksum เพื่อดูว่าข้อมูลที่ได้รับมาถูกต้องครบถ้วน และถ้าได้รับ Package ข้อมูลที่ไม่ถูกต้องก็จะไม่เอาข้อมูลนั้นไปใช้งาน และจะบอกให้ต้นทางส่งข้อมูลเดิมมาใหม่

Physical Layer รับผิดชอบดูแลในรายละเอียดในการส่งข้อมูลในด้าน Hardware เช่น การควบคุม Network Interface Card การส่งสัญญาณผ่านสายสัญญาณแบบต่าง ๆ การเชื่อมต่อเข้ากับเครือข่ายแบบต่าง ๆ โดยใช้ Physical Layer จะจัดสร้างสัญญาณทางไฟฟ้า , สัญญาณเสียง หรือ สัญญาณที่จำเป็นในการสื่อสารโดยตรง

2.5.3 โครงสร้างชั้นเน็ตเวิร์ค (Network Layer Structure)

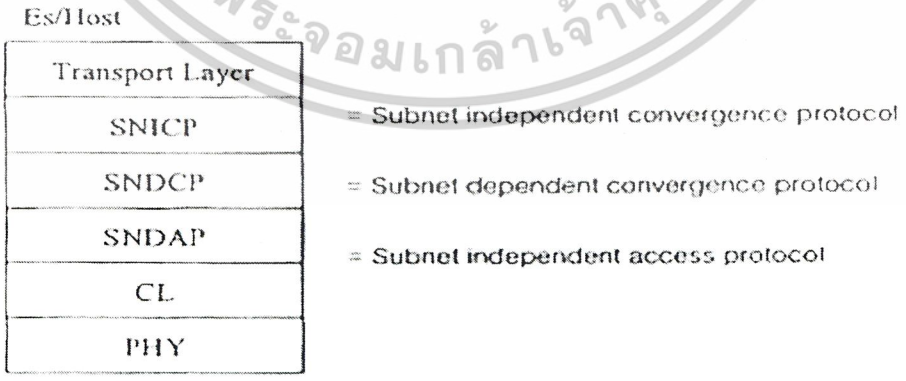
หน้าที่ของ Network Layer ในแต่ละ End System (ES) จะเป็นตัวจัดการการติดต่อแบบ End-to-End ของการบริการ Internetwide ไปยังผู้ใช้บริการ (NS-User)

โดย ISO ได้จัด Network Layer เป็น 3 (Sublayer) Protocol ซึ่งจะทำงานร่วมกัน เพื่อให้บริการใน Network Layer ได้แก่

Subnetwork independent convergence protocol (SNICP)

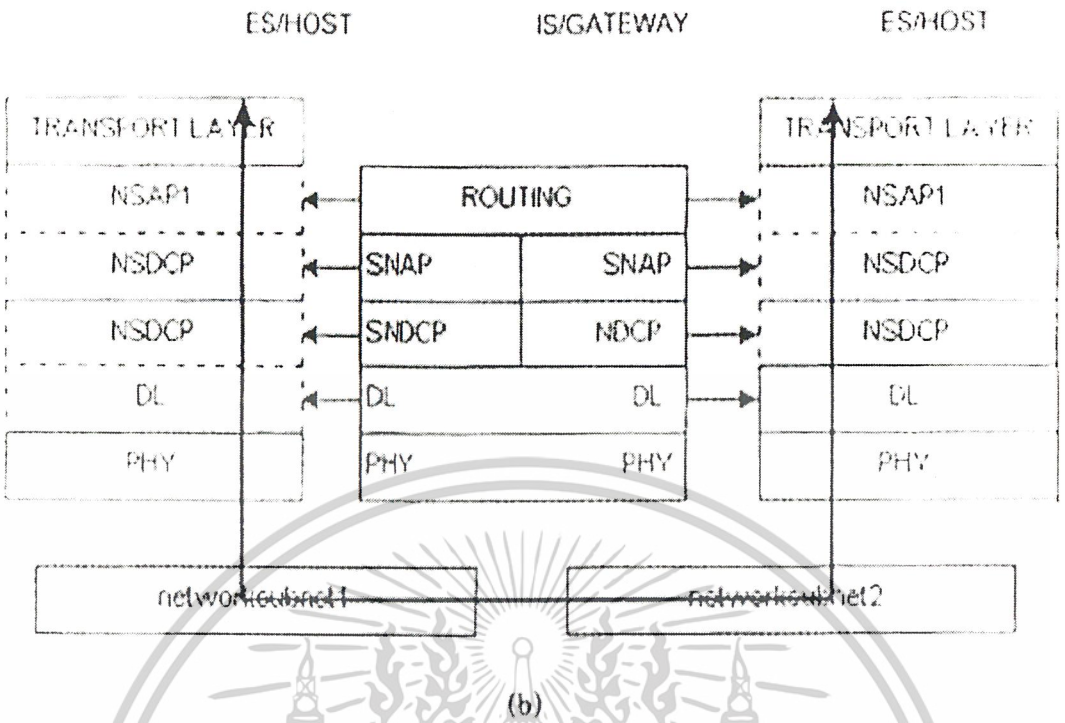
Subnetwork dependent convergence protocol (SNDCP)

Subnetwork dependent access protocol (SNDAP)



(a)

รูปที่ 2.13 Network Layer Structure (a) Sublayer Protocol ; (b) IS Structure



(b)

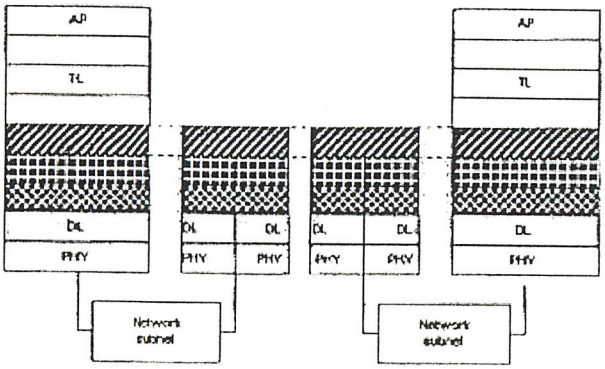
รูปที่ 2.13(ต่อ) **Network Layer Structure (a) Sublayer Protocol ; (b) IS Structure**

โดยที่ SNICP จะเป็นตัวสนับสนุนจัดการให้ผู้ใช้บริการ (NS-User) สามารถ Interface กับ Inernet ซึ่งมันจะมีหน้าที่ เป็นตัวประสานฟังก์ชันต่าง ๆ ที่จำเป็นในการเลือกเส้นทางและถ่ายทอดข้อมูลของผู้ใช้ข้าม Internet ซึ่งการทำงานของมันไม่ขึ้นอยู่กับคุณสมบัติเฉพาะของ เครือข่ายย่อย (subnet)

SNDAP จะเป็นตัวโพรโตคอลที่ติดต่อกับเครือข่ายย่อย (Subnet) ที่มีลักษณะเฉพาะใน Internet เช่น X.25 Packet Layer Protocol สำหรับเครือข่าย X.25 ซึ่งใช้บ่อยใน LAN เพราะว่าการบริการและการทำงานของ SNDAP แตกต่างจาก Network แบบอื่น ๆ Sublayer ที่อยู่ตรงกลางคือ SNDCP จะเป็นตัวจัดการระหว่าง SNICP และ SNDAP

2.5.4 รูปแบบมาตรฐานโพรโตคอลของอินเตอร์เน็ต (Internet Protocol Standards)

อินเตอร์เน็ตโพรโตคอลซึ่งถูกใช้ในอินเตอร์เน็ตมี โพรโตคอลก็คือ TCP/IP (Transfer Control Protocol / Internet Protocol) ซึ่งรวมถึง Transport และ Application โพรโตคอล ซึ่งทั้งหมดของ TCP/IP จะกำหนด ให้เหมาะกับการใช้ในเชิงสาธารณะ ซึ่งรูปแบบโดยทั่วไปแสดงในรูปที่ 2.14



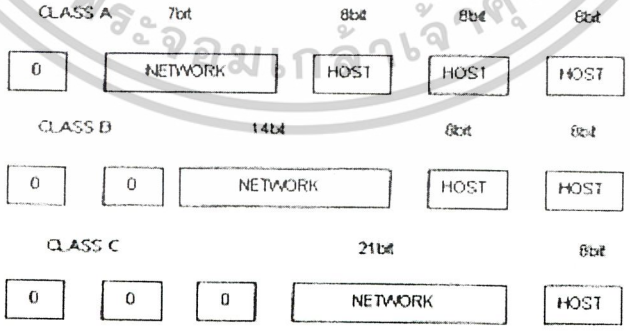
รูปที่ 2.14 Internetwide IP Schematic

IP เป็น Internetwide Protocol ซึ่งทำให้สอง Transport Protocol ที่ต่างสถานที่และต่าง ESS / Hosts กันสามารถแลกเปลี่ยนหน่วยข้อมูล (NSDUS) กันได้ ซึ่งหมายถึงว่า หลาย ๆ Network / Subnet และ Iss / Gateways ที่แตกต่างกันสามารถติดต่อสื่อสารกันได้อย่างสมบูรณ์

2.5.5 Internet IP

Address Structure

ในศัพท์ของ ISO เมื่อ 2 Network ติดต่อกันด้วย Host/ES ที่ต่อกับอินเตอร์เน็ต Network เหล่านี้ ติดต่อกันได้โดยใช้ Network Service Access Point (NSAP) Address และ Subnet Point of Attachment (SNPA) สำหรับใน TCP/IP ก็จะมี IP Address และ NPA Address ตามลำดับ โดย NPA Address จะแตกต่างกันในแต่ละชนิดของ Network / Subnet ขณะที่ IP Address จะเป็นรูปแบบเดียวกัน โครงสร้างของ IP Address แสดงในรูปที่ 2.15



รูปที่ 2.15 โครงสร้างของ Address ที่ใช้ใน Class ต่าง ๆ ของเครือข่ายทั้งหมดความยาว 32 Bit

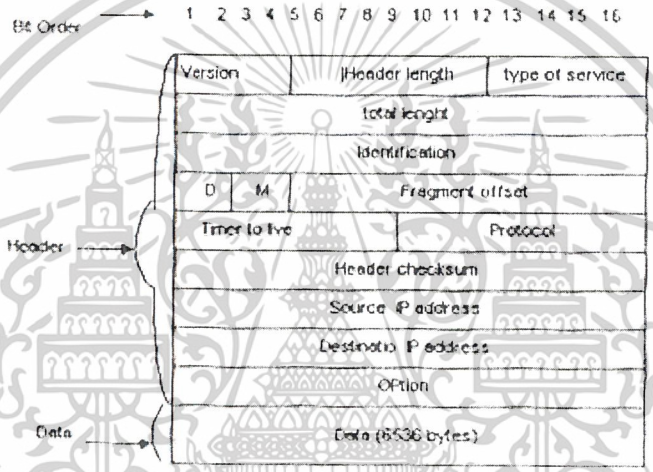
IP address นี้มีการจัดแบ่งเป็นทั้งหมด 5 ระดับ (class) แต่ที่ใช้ทั่วไปจะมีเพียง 3 ระดับคือ Class A, Class B และ Class C ซึ่งจะแบ่งตามขนาดความใหญ่ของเครือข่าย ถ้าเครือข่ายใดมีจำนวนเครื่องคอมพิวเตอร์ที่เชื่อมต่ออยู่มาก จะมีหมายเลขอยู่ใน Class A และลดหลั่นกันมาใน Class B และ Class C ตามลำดับ

จากรูปจะเห็นว่าหมายเลข IP ของ Class A มีตัวแรกเป็น 0 และหมายเลขของเครือ (network number) ขนาด 7 bit และมีหมายเลขเครื่องคอมพิวเตอร์ (Host number) ขนาด 24 bit ทำให้ในหนึ่งเครือข่ายของ Class A สามารถมีคอมพิวเตอร์เชื่อมต่ออยู่ในเครือข่ายได้ถึง 2^{24} = กว่า 65,000 เครื่อง และสุดท้ายคือ Class C ซึ่งมีหมายเลขคอมพิวเตอร์แบบ 8 bit และมีหมายเลขเครือข่ายแบบ 21 bit ส่วน 3 bit แรกบังคับว่าต้องเป็น 110_2 ดังนั้นในแต่ละเครือข่าย Class C จะมีจำนวนเครื่องต่อเชื่อมได้เพียงไม่เกิน 254 เครื่องในแต่ละเครือข่าย (2^8 = 256 แต่หมายเลข 0 และ 255 จะไม่ถูกใช้งาน จึงเหลือเพียง 254)

จะเห็นได้ว่าเมื่อเครือข่ายและเครื่องคอมพิวเตอร์ ที่ต่ออยู่ในอินเตอร์เน็ตมีหมายเลข IP address ให้ใช้อ้างอิงได้ไม่ซ้ำกัน และมีความหมายให้ทราบถึงขนาดเครือข่ายแล้ว การติดต่อส่งผ่านข้อมูลจึงกระทำได้ไม่สับสน

รูปแบบของข้อมูล (Datagrams)

รูปแบบของ IP data unit ก็คือ datagrams ซึ่งโครงการของ datagrams เป็นดังรูปที่ 2.16



รูปที่ 2.16 Internet datagrams format and contents

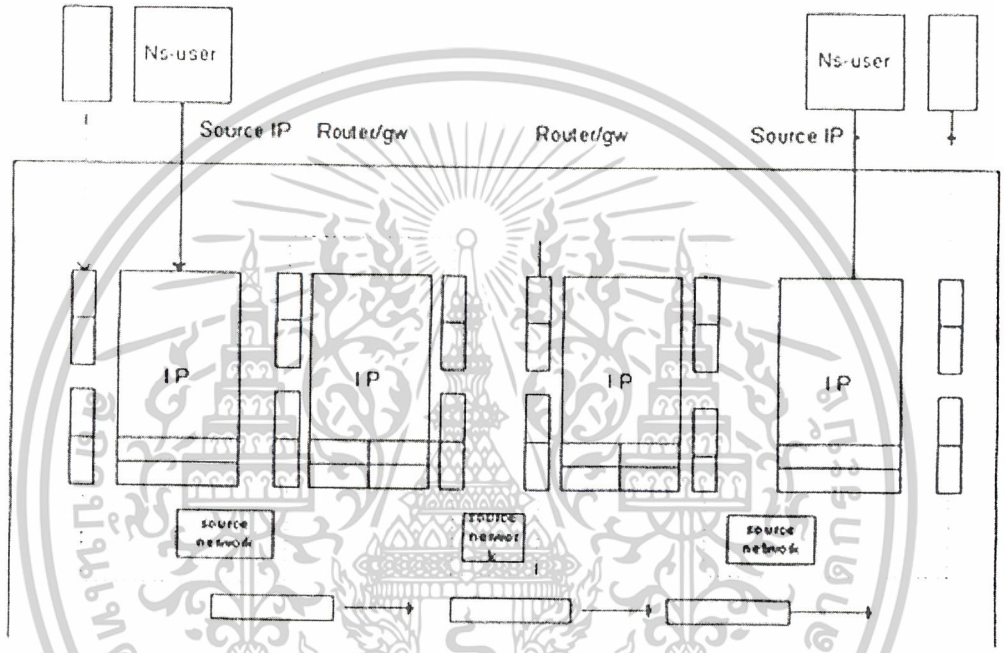
หน่วยข้อมูล IP (IP datagrams) แต่ละหน่วยจะประกอบด้วย ส่วนของข้อมูลที่รับมาจากส่วนของงาน TCP หรือ UDP และส่วนของข้อมูลนำทาง (Header) ซึ่งมีรายละเอียดดังนี้

- Version หมายเลขรุ่นของข้อกำหนด IP
- Haeder Length ความยาวของข้อมูลนำทาง
- Type of service วิธีการจัดการกับข้อมูล
- Total Length ความยาวของหน่วยข้อมูล
- Identification, Flags และ Fragment offset รายละเอียดที่เกี่ยวกับการแบ่งย่อยข้อมูล ซึ่งจะถูกนำมาใช้ในการรวบรวมข้อมูล
- Time to live เวลาสูงสุดที่ใช้ในการเดินทาง ซึ่งกำหนดมาจากต้นทาง เวลานี้จะลดลงเรื่อยๆ ในระหว่างทาง ถ้าลดลงไปถึงศูนย์ หน่วยข้อมูลนั้นจะถูกกำจัดไป
- Protocal ชนิดของข้อมูลเป็น UDP หรือ TCP
- Header Checksum ค่าตรวจสอบข้อมูลนำทาง
- IP address หมายเลข internetwide IP (NSAP) ของเครื่องต้นทางและปลายทาง

Option ข้อมูลอื่นๆ เช่น ข้อมูลเกี่ยวกับการรักษาความปลอดภัย บันทึกเส้นทางเดินของข้อมูล และเวลาที่ข้อมูล และเวลาที่ข้อมูลเดินทางมาถึง เป็นต้น

การแบ่งส่วนย่อยของข้อมูลและการประกอบขึ้นใหม่ (Fragmentation and Reassembly)

ขนาดของข้อมูลของผู้ใช้ซึ่งอ้างอิงกับ NSDU มีความจุได้ถึง 64k หรือ 65,536 bytes แต่ขนาดของหน่วยข้อมูล (packet size) ที่สามารถติดต่อกันในระบบที่ต่างกัน สามารถมีได้ตั้งแต่ 128 bytes สำหรับระบบ X.25 packet switching จนถึง 8000 bytes สำหรับบาง LAN ดังนั้นกระบวนการ Fragmentation และ Reassembly ซึ่งถูกนำมาใช้เพื่อ ทำให้ขนาดของข้อมูลเล็กลง และสามารถส่งไปในระบบได้ และเมื่อถึงปลายทาง IP ก็จำทำหารประกอบข้อมูล (reassembly) ขึ้นมาใหม่ก่อนที่จะส่งผ่านไปยังผู้ใช้ปลายทาง ดังรูปที่2.17



รูปที่ 2.17 internet fragmentation and reassembly

อันดับแรก IP ใน Host ต้นทางจะแยกข้อมูลของผู้ใช้ (NS – User), NSDU เป็น Datagram ซึ่งมี Address กำกับเป็นเฉพาะส่วนๆไป ซึ่งจะถูกออกคำสั่งโดย Network ที่มันติดต่ออยู่ด้วยและส่ง Datagram ไปยัง IP ใน Gateway ตัวแรก โดยที่ IP ใน Gateway จะไม่ Reassemble NSDU แต่จะปรับปรุงในขอบเขตที่เหมาะสม และส่ง Datagram ที่ได้รับตรงไปยัง Network ที่สอง (ถ้า Network ที่สองสามารถรองรับขนาด Datagram นี้) หรือทำการ Fragment datagram ให้มีขนาดเล็กลง ซึ่งขั้นตอนนี้จะถูกทำซ้ำที่ Gateway ตัวต่อไป โดยในรูปที่ 2.17 Network หลังสุดสามารถรองรับขนาดของ Packet ได้มากกว่า Packet ที่มันได้รับข้อมูลจึงถูกส่งได้โดยตรง โดยที่จะมีการปรับปรุงในบางส่วน Header ของ Datagram เท่านั้น จากนั้น IP ใน Host ปลายทางจะทำการประกอบข้อมูล (Reassembly) ที่มันได้รับขึ้นมาใหม่ และส่งผลที่ได้รับก็คือ NSDU ไปยังผู้ใช้ (NS-user)

ในการคิดค่าเวลาสูงสุดที่ Host ต้นทางกำหนดให้ Gateway รอ Datagram (NSDU) ระหว่างแต่ละการ Assembly ซึ่งก็คือ time-to-live ซึ่งจะถูกนำติดไปที่ส่วน Header ของ Diagram ซึ่งจะถูกตั้งค่าโดย IP ใน Host ต้นทาง ซึ่งจะมีค่าลดลงเรื่อยๆในแต่ละขั้นตอนของการ Process datagram ถูก Fragment ค่าปัจจุบันจะถูกนำไปใส่ในส่วน Header

ของ Diagram ตัวใหม่ ถ้ามันถึงค่า 0 ที่จุดใดๆ ระหว่างการ Process ใน Gateway (หรือ Host) การ Reassembly ก็จะล้มเหลวและทุกๆการ Fragment ที่เกี่ยวกับ NSDU ก็จะถูกตัดทิ้ง

ค่า Time-to-live ในแต่ละ Datagram จะเป็นจำนวนเท่าของ 1 วินาที โดยที่จำนวนของมันจะถูกลดลงโดIP ซึ่งจะเปลี่ยนแปลงไปตามค่าเวลาจริงในการส่งถ่ายข้อมูลของ Network ที่ติดต่อด้วย

การเลือกเส้นทาง (Routing)

แต่ละ Network (หรือ subnet) ใน Internet จะมีชนิดของ PA address ที่แตกต่างกัน ซึ่งระบบ (system)-host หรือ gateway ที่ถูกต่อเข้ากับ network จะสามารถส่ง datagram ไประบบอื่นได้โดยตรงเฉพาะ network ที่เหมือนกันเท่านั้น ในการเลือกเส้นทาง (routing) ให้ datagram ข้ามไปยังหลายๆ network IP ในแต่ละ internetwork gateway ต้องรู้ PA address ของ host ปลายทาง

ซึ่งมี 2 วิธีการพื้นฐานที่ถูกใช้ในการหาเส้นทางภายใน Internet คือ centralized และ distributed ด้วยวิธีการ centralized routing ข้อมูลเกี่ยวกับการเลือกเส้นทาง ที่เกี่ยวข้องกับแต่ละ gateway จะถูก download จาก site ส่วนกลาง โดยใช้ข้อมูล network และ special network management โดย network management จะพยายามตรวจสอบ network และ host ที่ถูกเพิ่มเข้าและถอดออก และข้อบกพร่องที่จะถูกวินิจฉัยและตรวจซ่อม

ด้วยวิธีการ Distributrd routing ทุกๆ host และ gateway จะร่วมกันในการแบ่งปัน วิธีการในการรับประกันว่าข้อมูลเกี่ยวกับการเลือกเส้นทางในแต่ละ system, host และ gateway จะถูกทำให้ทันสมัย และสอดคล้องกัน ข้อมูลเกี่ยวกับการเลือกเส้นทางจะถูกจดจำไว้โดยแต่ละระบบ ในรูปของ routing table ซึ่งจะมี NPA address ไว้ได้ในกรส่งแต่ละ diagram ซึ่ง Internet จะใช้วิธีการแบบนี้

ขั้นตอนการ Routing ที่เกี่ยวกับ IP ขั้นตอนแรกจะอ่าน IP address (NSAP) ปลายทางจากภายใน datagram และใช้มันในการหาการตอบสนอง PA address ของ host หรือ gateway จาก routing table ในส่วนที่เพิ่มเติมชุดของ routing protocal จะถูกใช้เพิ่มเติมชุดของ routing protocal จะถูกใช้เพิ่มและรักษาส่วนที่อยู่ในแต่ละ routing table ในแบบของ distributed ซึ่งรูปแบบทั่วไปถูกใช้ภายใน host IP แสดงดังรูปที่ 2.18

2.6 โปรแกรม Delphi

2.6.1 แนะนำ Delphi

การสร้างแอพพลิเคชันบน Windows ในยุคปัจจุบัน สามารถทำได้ง่ายและสะดวกกว่ายุคแรกๆ ที่จะต้องเขียนโค้ดใส่ไว้ในไฟล์เป็นจำนวนมาก แล้วจึงค่อยสั่งให้แปลภาษา และยุ่งยากในการแก้ไขข้อผิดพลาดของโปรแกรม (Debug) อีกทั้งยากที่จะเห็นผลลัพธ์ที่ได้ว่าจะเป็นไปตามที่ต้องการหรือไม่ แต่ในยุคนี้มีเครื่องมือในการสร้างแอพพลิเคชั่นมากมาย ที่มารถเห็นผลได้ตั้งแต่ในขณะที่กำลังสร้าง ทีใช้มีทั้งภาษา C ภาษา Basic และภาษา Pascal ซึ่งก็คือ Delphi นั่นเอง

Delphi เป็นเครื่องมือในการพัฒนาแอพพลิเคชั่น ที่นอกจากจะใช้ง่ายและสามารถเห็นผลลัพธ์ที่ที่จะได้ตั้งแต่ในขณะที่ทำการสร้างแล้ว ยังมีข้อดีที่เหนือกว่าภาษาอื่นๆคือ ในแง่ของการเขียนโค้ดทำได้ง่ายกว่าภาษา C และแอพพลิเคชั่นที่ได้จะสามารถทำงานได้ประสิทธิภาพสูงเกือบเท่าภาษา C ซึ่งเร็วกว่าภาษา Basic มาก

Delphi สามารถใช้พัฒนาแอพพลิเคชั่นได้ทุกระดับทุกประเภทครบถ้วนในตัวเดียว โดยไม่ต้องใช้ภาษาอื่นเพิ่มเติมอีกเลย ไม่ว่าจะเป็นการสร้างโปรแกรมง่ายๆสำหรับมือสมัครเล่น หรือใช้ทำงานที่สลับซับซ้อนสำหรับมืออาชีพ อีกทั้งยังรองรับระบบปฏิบัติการทั้ง Windows 95, 98 และแม้แต่ Windows NT

สำหรับเนื้อหาในบทนี้จะได้แนะนำให้รู้จักว่า Delphi คืออะไร มีคุณสมบัติหลักๆอะไรบ้าง ความต้องการของเครื่องที่จะใช้ Delphi เป็นอย่างไร ควรมีคุณสมบัติอะไรบ้าง และ Delphi มีเครื่องมืออะไรให้ใช้ได้บ้าง

Delphi คืออะไร

Delphi เป็นเครื่องมือที่ใช้ในการพัฒนาโปรแกรมบน Windows โดยใช้ภาษาปาสคาล (Pascal) เป็นหลักในการพัฒนาโปรแกรม ซึ่งเป็นโครงสร้างภาษาที่เขียนง่าย และถึงแม้ว่าผู้ใช้จะไม่มีความรู้เกี่ยวกับภาษาที่ใช้เลยก็ไม่ได้เป็นอุปสรรคต่อการพัฒนาโปรแกรมแต่อย่างใด เนื่องจาก Delphi มีเครื่องมือในการช่วยเหลือในการนำคำสั่งต่างๆมาใช้งานได้อย่างสะดวกและรวดเร็ว

Delphi เป็นผลิตภัณฑ์ของบริษัท Borland (ต่อมาได้เปลี่ยนชื่อเป็น Inprise โดยยังคงใช้ชื่อผลิตภัณฑ์ว่า Borland Delphi เช่นเดิม) ซึ่งประสบความสำเร็จจากการพัฒนา Turbo Pascal ที่มีชื่อเสียงโด่งดัง DOS จนกระทั่งมาเป็น Delphi ในปัจจุบัน

Delphi เป็นเครื่องมือสำหรับการพัฒนาโปรแกรม ที่มีสิ่งอำนวยความสะดวกไว้อย่างครบครัน โดยมีสภาพแวดล้อมในการทำงาน (Development Environment) ที่ช่วยให้สามารถทำทุกอย่างได้จากใน Delphi เอง มีเครื่องมือทุกชนิดที่จำเป็นสำหรับการสร้างแอพพลิเคชั่นบน Windows ทั้งในส่วนของการติดต่อกับผู้ใช้ การแสดงผลกราฟิก การติดต่อกับฐานข้อมูล การจัดการระบบ ตลอดจนการพัฒนาแอพพลิเคชั่นเพื่อทำงานบนอินเตอร์เน็ต และด้วยคุณสมบัติที่มีอยู่มากมายของ Delphi พอจะแยกได้เป็นข้อๆดังนี้

วิชวลโปรแกรมมิ่ง (Visual Programming)

การพัฒนาโปรแกรมแบบวิชวล คือการพัฒนาโดยเห็นผลที่จะเกิดขึ้นเมื่อรันโปรแกรมได้ตั้งแต่ในขณะที่กำลังสร้าง โดยการนำชิ้นส่วนต่างๆที่ต้องการ ได้แก่ ปุ่ม (Button), ข้อความ (Label), รูปภาพ (Image)ฯลฯ ซึ่งเหล่านี้เรียกโดยรวมๆว่า คอมโพเนนต์ (Component) นำมาวางบนวินโดว์ที่เรียกว่า ฟอร์ม (Form) ปรับขนาดและตำแหน่ง รวมทั้งคุณสมบัติต่างๆของคอมโพเนนต์ และแม้แต่ฟอร์มเองให้ได้ผลตามที่ต้องการ โดยการเปลี่ยนคุณสมบัติเหล่านี้ จะมีผลตั้งแต่ในขณะที่กำลังออกแบบ และเมื่อรันโปรแกรมก็จะได้ผลลัพธ์เหมือนกับที่เห็นในขณะที่ออกแบบ

การโปรแกรมเชิงวัตถุ (Object Oriented Programming – OOP)

การโปรแกรมเชิงวัตถุเป็นการพัฒนาโปรแกรมโดยการสร้าง วัตถุ หรือ ออบเจ็ค (Object)ที่ต้องการ ในมุมมองของตัววัตถุเองว่าต้องการให้มีลักษณะเป็นอย่างไรและสามารถทำอะไรได้บ้าง แทนที่จะมองที่การสร้างรูทีน (Routine) หรือโพรซีเยอร์ (Procedure) เป็นหลักเช่นดังก่อนๆนี้ ประโยชน์ที่ได้ก็คือ เราสามารถสร้างวัตถุโดยเริ่มจากวัตถุที่ง่ายๆไม่ซับซ้อนเป็นพื้นฐานขึ้นมาเสียก่อน จากนั้นจึงนำวัตถุเหล่านี้มาตกแต่งปรับปรุงให้มีความสามารถมากขึ้นหรือทำงานได้หลากหลายขึ้น โดยนำสิ่งที่เหมือนกันหรือใช้ร่วมกันมาไว้ในวัตถุ ซึ่งจะเรียกว่า BaseObject หรือ Base Class จากนั้นจึงแต่งเติม Base Object นี้ให้กลายเป็นออบเจ็คอื่นๆ ตามที่ต้องการ เราสามารถนำวัตถุที่ได้นี้กลับมาใช้ใหม่ได้เรื่อยๆ ในแอพพลิเคชั่นต่างๆ และแต่งเติมต่อไปได้เรื่อยๆ เช่นกัน

สิ่งสำคัญในการเขียนโปรแกรมเชิงวัตถุคือ จะต้องมีความเข้าใจเกี่ยวกับโครงสร้างของออบเจ็คและหลักการของ OOP ดังนี้

โครงสร้างของออบเจ็ค

ออบเจ็คทุกๆ ออบเจ็คจะต้องมีโครงสร้างดังต่อไปนี้

ชนิดของออบเจ็ค	ออบเจ็คแต่ละออบเจ็คจะถือว่าเป็นคนละชนิด (Type) กัน เมื่อนำออบเจ็คไปสร้างต่อให้เป็นออบเจ็คใหม่เสมอ หรือเรียกได้ว่าเป็นคนละคลาส (Class) กันนั่นเอง ชนิดของออบเจ็คได้แก่ ออบเจ็คชนิดปุ่ม และชนิดข้อความ
คุณสมบัติ	หรือเรียกว่า “พร็อพเพอร์ตี้” (Property) หมายถึงคุณลักษณะของออบเจ็คแต่ละตัวที่สามารถกำหนดให้แตกต่างกันไปตามความต้องการที่ต่างกัน เช่น ขนาดและสีของปุ่มหรือข้อความของออบเจ็คที่แสดงอยู่บนปุ่ม เป็นต้น
พฤติกรรม	หรือเมธอด (Method) คือความสามารถในการทำงานของออบเจ็ค ตัวอย่างเช่น การแสดงปุ่ม (Show) หรือการซ่อนปุ่ม (Hide) เป็นต้น

หลักการของ OOP

คุณสมบัติของการโปรแกรมที่ถือว่าเป็นเชิงวัตถุได้ จะต้องมีคุณสมบัติในการซ่อนเร้นการทำงานนี้ไว้ภายในสามารถนำคุณสมบัติที่มีไปใช้งานต่อๆไปได้ และอาจจะแปลงเป็นออบเจ็คใหม่ได้ ดังนี้

Encapsulation	เป็นการซ่อนเร้นส่วนการทำงานภายในออบเจ็ค ที่ไม่เกี่ยวข้องกับภายนอกไว้ ไม่ให้เห็นและไม่ให้แก้ไขเปลี่ยนแปลงส่วนที่ซ่อนไว้นี้ ซึ่งเราจะนำออบเจ็คไปใช้หรือดัดแปลงได้เฉพาะส่วนที่ออบเจ็คนั้นยอมให้เท่านั้น
Inheritance	เป็นการสืบทอดคุณสมบัติของออบเจ็ค เมื่อนำออบเจ็คใดๆไปสร้างเป็นออบเจ็คใหม่ คุณสมบัติของออบเจ็คเดิมจะยังคงมีอยู่ และสามารถเรียกใช้และทำงานได้อย่างครบครัน
Polymorphism	คือลักษณะการทำงานที่แตกต่างกันของคุณสมบัติหรือพฤติกรรมเดียวกัน แต่เป็นของออบเจ็คคนละชนิดกัน ตัวอย่างเช่น เมธอด Save To File ของเมโมจะได้ text file ที่เก็บข้อความนั้น ในขณะที่ Save To File ของอิมเมจจะได้ไฟล์รูปภาพ ซึ่งแม้จะเป็นการบันทึกข้อมูลไว้เป็นไฟล์เหมือนกัน แต่วิธีการบันทึกก็ต่างกัน

คอมไพเลอร์

Deiphi เป็นคอมไพเลอร์ที่ใช้แปลภาษาโปรแกรมเป็นภาษาเครื่อง ซึ่งหลังจากการคอมไพล์โปรแกรม เราจะได้ไฟล์ .exe ซึ่งเก็บภาษาของเรื่องนั้นๆ และสามารถทำงานได้เลยโดยไม่ต้องขั้นตอนการแปลภาษาในระหว่างการทำงานอีก ทำให้สามารถทำงานได้เร็วและไม่มีขีดจำกัด

คอมโพเนนต์ไลบรารี (Component Libraly)

คอมโพเนนต์เป็นส่วนประกอบย่อยๆ ที่เราจะนำมาใช้ในการสร้างแอพพลิเคชั่น ซึ่งใน Delphi มีคอมโพเนนต์ให้เลือกใช้อยู่เป็นจำนวนมาก โดยจะเก็บอยู่ใน " ห้องสมุด " หรือ คอมโพเนนต์ไลบรารี (component library) และจัดแยกเป็นกลุ่มๆ ตามประเภทของการใช้งาน

OCX และ ActiveX

OCX คือออบเจ็คที่ใช้ใน Visual Basic ของไมโครซอฟท์ ซึ่งมีลักษณะคล้ายกับคอมโพเนนต์ของ Delphi ส่วน ActiveX ก็เป็นออบเจ็คที่ถูกพัฒนามาในรูปแบบเดียวกับ OCX แต่มีวัตถุประสงค์เพื่อใช้งานบนอินเตอร์เน็ต ซึ่งทั้ง OCX และ Active ต่างก็เป็นออบเจ็คที่นำมาใช้กับ Delphi ได้

Wizard และ Object Repository

Delphi มี Wizard ที่ใช้ในการสร้างฟอร์มและออบเจ็คที่ต้องใช้บ่อยๆ หรือสร้างได้ยาก ไว้ให้ใช้เป็นจำนวนมาก ตัวอย่างเช่น Wizard ในการสร้างรายงานแบบต่างๆ หรือฟอร์ม เช่น About Box และ Dialog Box ซึ่งจะเรียกอีกอย่างหนึ่งว่า เทมเพลต (Template) ก็ได้ เหล่านี้ล้วนช่วยให้การสร้างแอพพลิชันเป็นไปอย่างรวดเร็วและมีความกลมกลืนมากขึ้น นอกจากนี้ยังสามารถเก็บฟอร์มหรือเก้บฟอร์มหรือออบเจ็คที่คิดว่าต้องใช้บ่อยๆ เพิ่มเข้าไปไว้ให้เรียกใช้ในแอพลิเคชั่นอื่นๆได้อีกด้วย โดยการบันทึกไว้ใน Object Repository จากนั้นเมื่อต้องการใช้งานก็เรียกใช้ได้เหมือนกับการใช้ Wizard ทั่วๆไป

การติดต่อฐานข้อมูล

การติดต่อกับฐานข้อมูลนั้น Delphi มีคอมโพเนนต์ที่สามารถเชื่อมต่อ เพื่อจัดการกับข้อมูลที่เก็บอยู่ในฐานข้อมูลทุกประเภท ไม่ว่าจะเป็นการเพิ่ม ลบ แก้ไข หรือการเรียกดูข้อมูล โดยผู้ใช้ไม่ต้องเขียนชุดคำสั่งใดๆ ในโปรแกรมเลยก็สามารถสร้างแอพพลิเคชั่นอย่างง่ายๆ ที่ทำงานกับฐานข้อมูลขึ้นมาได้แล้ว

เราสามารถใช้ Delphi จัดการกับฐานข้อมูลที่เป็นแบบง่ายๆ ซึ่งได้แก่ dBase หรือ Paradox และ MS Access ซึ่งพวกนี้จะเรียกว่าเป็น Local Database คือ Database ที่ทำงานในเครื่องนั้นเอง หรือเรียกอีกอย่างหนึ่งว่า เป็นแบบ File_oriented Database ก็ได้ เพราะเป็นการเก็บข้อมูลไว้ในโครงสร้างของไฟล์ นอกจากนี้ยังสามารถใช้งานกับระบบฐานข้อมูลที่เป็น Database Server ต่างๆ เช่น SQL Server หรือ InterBase ซึ่งอาจจะทำงานอยู่ในเครื่องเดียวกันกับแอพพลิเคชั่นในกรณีของ Local InterBase หรืออาจจะทำงานอยู่บนเครื่อง Server เครื่องอื่น ซึ่งเรียกว่าเป็นการใช้ฐานข้อมูลในแบบ Client/Server ก็ยังได้

นอกจากนี้ใน Delphi ยังมีคุณสมบัติในการติดต่อกับฐานข้อมูลด้วย ADO ซึ่งไมโครซอฟท์ได้พัฒนาขึ้นมาใหม่เพื่อให้สามารถติดต่อกับแหล่งข้อมูลได้หลายประเภทโดยไม่ต้องใช้ Borland Database Engine (BDE)

2.6.2 การทำงานกับสภาพแวดล้อมของ Delphi

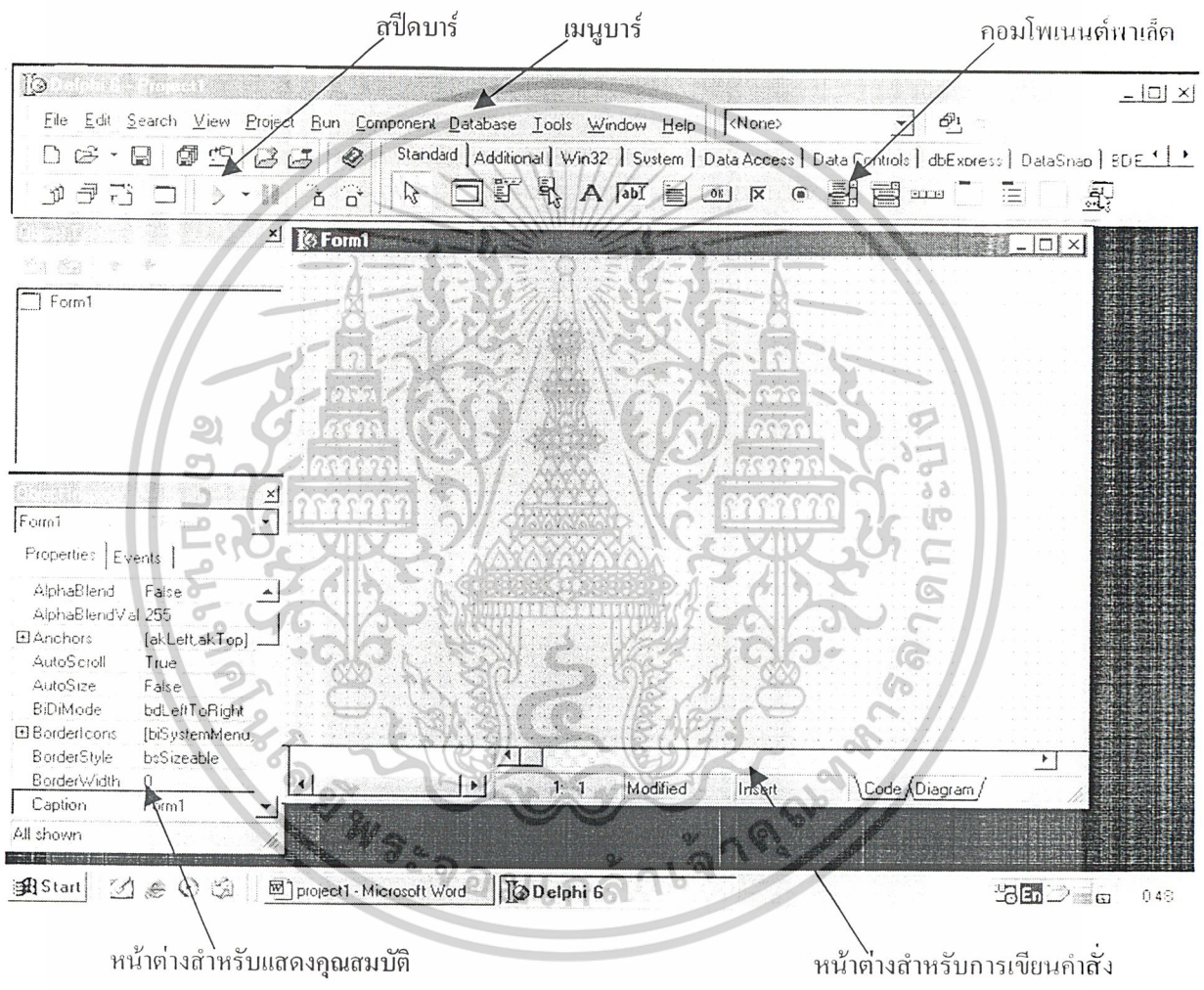
หลังจากที่ติดตั้ง Delphi และพร้อมใช้งานแล้ว สิ่งที่จะต้องทำความรู้จักก่อนที่จะเริ่มพัฒนาแอพพลิเคชั่นก็คือ สภาพแวดล้อมของโปรแกรม Delphi เพราะเราคงจะทำงานไม่ได้ถ้าไม่รู้ว่าแต่ละส่วนใน Delphi หมายถึงอะไร และ

ใช้งานอย่างไร ในบทนี้ผู้อ่านจะได้รู้จักหน้าตาและการทำงานกับสภาพแวดล้อมของ Delphi ตลอดจนการปรับแต่งสภาพแวดล้อมเพื่อใช้ในการพัฒนาแอพพลิเคชั่น

สภาพแวดล้อมของ **Delphi**

การสร้างโปรแกรมใน Delphi 7 มีรูปแบบที่เรียกว่า Integrated Development Environment (IDE) คือเป็นการสร้างโปรแกรมโดยการทำงานทุกอย่างสามารถทำได้จากในที่แห่งเดียวโดยไม่จำเป็นต้องใช้ Editor ตัวหนึ่งเขียนโปรแกรม และคอมไพล์ด้วยคำสั่งที่เป็น Command Line (แต่ถ้าต้องการก็ยังสามารถทำได้ ซึ่งนอกเหนือจากขอบเขตของหนังสือเล่มนี้)

หลังจากที่รันโปรแกรม Delphi เราจะเห็นหน้าตาของ Delphi ซึ่งภายในประกอบด้วยวินโดว์ต่างๆดังนี้



รูปที่ 2.19 แสดงหน้าตาโปรแกรม Delphi

วินโดว์หลัก (**Main Window**)

วินโดว์หลักทำหน้าที่เป็นตัวกลางสำหรับควบคุมประบวนการในการพัฒนาโปรแกรม ไม่ว่าจะเป็นการจัดการเกี่ยวกับไฟล์ภายในแอพพลิเคชัน การคอมไพล์ การตรวจสอบข้อผิดพลาด เมื่อรันโปรแกรม Delphi จะเห็นวินโดว์หลักเป็นวินโดว์ที่อยู่บนสุด โดยภายในวินโดว์หลักจะแบ่งออกเป็น 5 ส่วน คือไตเติลบาร์ เมนูบาร์ เดสก์ท็อปทูลบาร์ สปีดบาร์ และคอมโพเนนต์พาเล็ต

ไตเติ้ลบาร์ (Title Bar)

เป็นส่วนที่อยู่บนสุด แสดงชื่อโปรแกรม Delphi และชื่อโปรเจ็คที่ทำงานอยู่ในขณะนั้น

เมนูบาร์ (Menu Bar)

เป็นส่วนประกอบที่จะพบได้ในแอพพลิเคชั่นบน Windows ทั่วไป โดยจะอยู่ใต้ไตเติ้ลบาร์ ภายในเมนูบาร์จะเป็นส่วนที่แสดงเมนูคำสั่งของ Delphi ซึ่งใช้ในการทำงานต่างๆ เกือบทั้งหมด เช่น เมนู File ใช้ทำงานกับแฟ้มข้อมูล ไม่ว่าจะเป็นการเปิด การปิด หรือการบันทึกแฟ้มข้อมูล เป็นต้น

เดสก์ท็อปทูลบาร์ (Desktops toolbar)

เดสก์ท็อปทูลบาร์เป็นคุณสมบัติใหม่ใน Delphi ซึ่งจะใช้สำหรับการบันทึกเลย์เอาต์ของเดสก์ท็อปทั้งในขณะออกแบบและขณะดีบักโปรแกรม ซึ่งก็คือตำแหน่งของวินโดว์ต่างๆ รวมทั้งไดอะล็อกบ็อกซ์ที่เปิดขึ้นมาภายใน Delphi โดยในส่วนของเดสก์ท็อปทูลบาร์นี้จะประกอบด้วย

Desktop speed settings	เป็นคอมโบบ็อกซ์ใช้สำหรับเลือกเลย์เอาต์ (layout) ของเดสก์ท็อปแบบต่างๆ ที่ต้องการใช้งาน
Save current desktop	เป็นไอคอนสำหรับบันทึกเดสก์ท็อปในขณะนั้น
Set debug desktop	เป็นไอคอนสำหรับกำหนดให้ใช้เดสก์ท็อปขณะนั้นเป็นเดสก์ท็อปที่จะปรากฏในขณะดีบัก ซึ่งจะแสดงโดยอัตโนมัติขณะรันไทม์

การปรับแต่งเลย์เอาต์

เราสามารถจัดวางวินโดว์ต่างๆ ที่เปิดขึ้นมาใน Delphi ไว้ที่ตำแหน่งต่างๆ และกำหนดให้มีขนาดตามที่ต้องการ ตัวอย่างเช่น ใช้เลย์เอาต์ตามดีฟอลต์ที่เปิด Delphi ขึ้นมา แล้วเลื่อนวินโดว์ออบเจ็คอินสเป็คเตอร์เข้าไปไว้ในวินโดว์โค้ดเอดิเตอร์ หลังจากนั้นจึงทำการบันทึกเดสก์ท็อปไว้ เพื่อให้สามารถดึงมาใช้งานได้สะดวกและรวดเร็ว ดังที่จะแสดงดังต่อไปนี้

1. จัดวางเดสก์ท็อปตามตำแหน่งที่ต้องการ
2. คลิกไอคอน **Save current desktop** จะปรากฏไดอะล็อกบ็อกซ์ Save Desktop
3. คลิกปุ่ม **OK**

สำหรับเลย์เอาต์ที่เลือกไว้จะมีผลต่อทุกโปรเจ็คที่เปิดขึ้นมาภายใต้เลย์เอาต์นั้นและจะถูกใช้ในครั้งต่อๆไปที่เข้าสู่ Delphi ในการบันทึกทับที่เลย์เอาต์เดิมหรือไม่ แต่ถ้าตั้งชื่อก็จะเป็น การบันทึกไว้เป็นเลย์เอาต์ใหม่สำหรับการใช้งานในกรณีต่างๆ

การสร้างเดสก์ท็อปสำหรับดีบัก

เราสามารถกำหนดเดสก์ท็อปไว้เพื่อใช้ในขณะดีบักโปรแกรมได้ โดยบันทึกเดสก์ท็อปเลย์เอาต์สำหรับดีบักไว้ก่อน ตัวอย่างเช่น เตรียมเดสก์ท็อปสำหรับดีบักไว้โดยการนำวินโดว์ Watches เข้ามาใส่ไว้ในโค้ดเอดิเตอร์ (ดูรายละเอียดเพิ่มเติมในหัวข้อ “การปรับสภาพแวดล้อมด้วย Docking Window”) แล้วจึงบันทึกไว้และกำหนดให้เป็นเดสก์ท็อปสำหรับดีบักดังนี้

1. คลิกเมนู **View => Debug Windows => Watches**
2. คลิกที่วินโดว์ Watch List แล้วลากไปปล่อยบริเวณขอบของโค้ดเอดิเตอร์
3. คลิกไอคอน **Save current desktop** แล้วบันทึกชื่อเดสก์ท็อปเป็น Debugging
4. คลิกไอคอน **Set debug desktop** หรือเลือก **View => Desktops => Set Debug Desktop**
5. เลือกเดสก์ท็อปเลย์เอาต์ที่ต้องการใช้ขณะรันแอพพลิเคชัน ในตัวอย่างนี้คือ Debugging

หลังจากที่เรากำหนดเดสก์ท็อปสำหรับดีบักแล้ว เดสก์ท็อปนี้จะทำงานโดยอัตโนมัติเมื่อเรารันแอพพลิเคชัน ในการทดสอบเพื่อให้เห็นการเรียกใช้เดสก์ท็อปสำหรับการดีบักเราจะเปลี่ยนเดสก์ท็อปเป็น My Desktop เสียก่อนแล้วทดลองรันโปรแกรมเพื่อดูหน้าตาเดสก์ท็อปขณะที่ดีบักโปรแกรมดังนี้

1. คลิกที่ปุ่ม Run
2. คลิกที่วินโดว์เอดิเตอร์ เราจะเห็นวินโดว์ Watches แสดงขึ้นมา

การลบเดสก์ท็อปเลย์เอาต์

เราสามารถลบเดสก์ท็อปที่กำหนดไว้ออกได้ดังนี้

1. เลือกเมนู **View => Desktop => Delete** จะปรากฏไดอะล็อกบ็อกซ์ Delete Saved Desktop
2. เลือกเลย์เอาต์ที่ต้องการลบ ตัวอย่าง Debugging
3. คลิกปุ่ม **Delete**

สปีดบาร์ (Speed Bar)

คือกลุ่มของคำสั่งที่ใช้บ่อยซึ่งจะแสดงอยู่ใต้เมนูบาร์ทางด้านซ้ายของวินโดว์หลัก ภายในสปีดบาร์ประกอบด้วยปุ่มแทนรายการต่างๆ ของเมนู เมื่อคลิกเมาส์ที่ปุ่มใดก็จะเหมือนกับการเลือกคำสั่งจากเมนู และเมื่อเลื่อนเมาส์ไปหยุดบนปุ่มใดก็จะเห็น กรอบข้อความ (Tooltips) แสดงชื่อเมนู และฟังก์ชั่นคีย์ซึ่งเป็นคีย์ลัดสำหรับใช้แทนปุ่มนั้น

การปรับแต่งสปีดบาร์

เราสามารถปรับแต่งปุ่มต่างๆที่ปรากฏบนสปีดบาร์เพื่อความสะดวกในการใช้งาน โดยอาจจะเพิ่มหรือลดได้ ตัวอย่างเช่น ถ้าเรามีการใช้งานคำสั่ง **Print** ในเมนู File บ่อยๆ ซึ่งแทนที่จะเลือกจากเมนู ก็สามารถเพิ่มปุ่ม Print ลงในสปีดบาร์ได้ดังนี้

1. คลิกเมาส์ขวาบนสปีดบาร์ จะปรากฏพ็อพอัพเมนูหรือเมนูลัดขึ้น
2. เลือกเมนู **Customize** จะปรากฏไดอะล็อกบ็อกซ์ Customize
3. คลิกแท็บ **Commands**
4. คลิกที่หัวข้อ File ในกรอบ **Categories** (กรอบทางซ้ายมือ)
5. เลื่อนสโครลบาร์ในกรอบของ Commands ลงมา จากนั้นคลิกที่หัวข้อ **Print** เมาส์พอยเตอร์จะเปลี่ยนจะเปลี่ยนเป็นรูปปุ่ม
6. คลิกที่สปีดบาร์ในตำแหน่งที่ต้องการให้ปรากฏปุ่ม Print
7. คลิกปุ่ม **Close** บนไดอะล็อกบ็อกซ์ Customize

ในกรณีที่ต้องการลบปุ่มออกจากสปีดบาร์ก็สามารถทำได้ง่าย โดยเปิดไดอะล็อกบ็อกซ์ Customize จากนั้นคลิกและลากปุ่มที่ต้องการลบไปปล่อยทิ้งนอกสปีดบาร์

นอกจากการเพิ่มหรือลบไอคอนบนสปีดบาร์โดยทำผ่านไดอะล็อกบ็อกซ์ Customize แล้ว ภายในไดอะล็อกบ็อกซ์ Customize ยังมีการทำงานในอีก 2 ส่วนแยกตามแท็บได้ดังนี้

Toolbars (ทูลบาร์) ใช้กำหนดให้แสดงหรือไม่แสดงกลุ่มไอคอนบนทูลบาร์ โดยคลิกเลือกเช็คบ็อกซ์ที่ต้องการให้ปรากฏและไม่เลือกในกลุ่มที่ไม่ต้องการให้ปรากฏ ซึ่งมีลักษณะการทำงานเหมือนการเลือกจากพ็อพอัพเมนูของสปีดบาร์

Options ใช้สำหรับกำหนดให้แสดงหรือไม่แสดงกรอบข้อความ (tooltips) และคีย์ลัดบนปุ่มของสปีดบาร์ โดยคลิกให้มีเครื่องหมายถูกหน้า Show tooltips เพื่อเลือกให้แสดงกรอบข้อความแนะนำการใช้งาน และคลิกเลือกเช็คบ็อกซ์ Show shortcut keys on tooltips เพื่อแสดงชื่อคีย์ลัดในกรอบข้อความด้วย

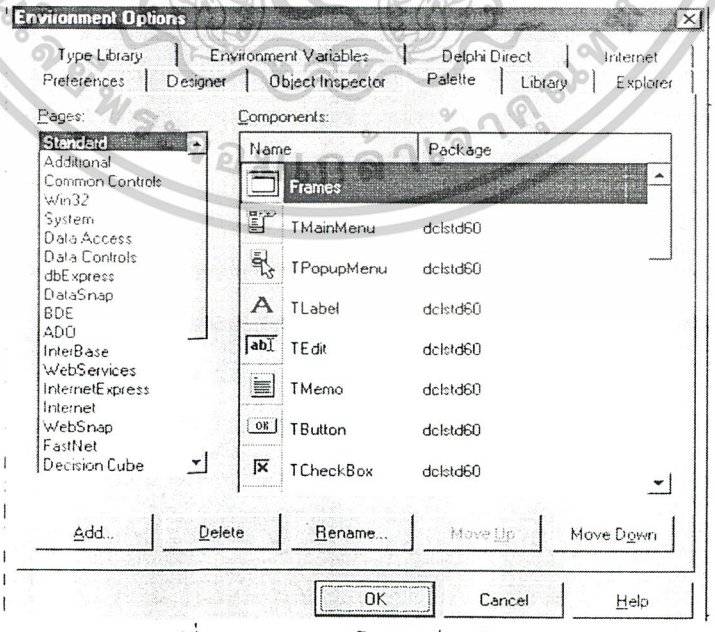
คอมโพเนนต์พาเล็ต (Component Palette)

คอมโพเนนต์พาเล็ตคือส่วนที่อยู่ถัดจากสปีดบาร์ไปทางด้านขวา ประกอบด้วยคอมโพเนนต์สำหรับสร้างแอพพลิเคชัน โดยจัดกลุ่มไว้ในแท็บต่างๆ คลิกที่แท็บเพื่อแสดงคอมโพเนนต์ในแต่ละกลุ่ม และเมื่อเลื่อนเมาส์ไปหยุดบนคอมโพเนนต์ใดก็จะปรากฏกรอบข้อความแสดงชื่อคอมโพเนนต์นั้นขึ้น

การปรับแต่งคอมโพเนนต์พาเล็ต

ถ้าต้องการจัดกลุ่มหรือลำดับของกลุ่มใหม่ให้สะดวกต่อการใช้งาน เราก็สามารถปรับแต่งคอมโพเนนต์พาเล็ตได้ โดยมีวิธีการดังนี้

1. คลิกเมนู **Tools => Environment Options**
2. คลิกแท็บ **Palette**



รูปที่ 2.20 แสดงคอมโพเนนต์พาเล็ต

จากรูป ชื่อกลุ่มต่างๆจะแสดงอยู่ในกรอบ Pages ที่อยู่ทางซ้าย และคอมโพเนนต์ของแต่ละกลุ่มจะแสดงในกรอบ Components ทางด้านขวา ส่วนด้านล่างจะแสดงปุ่มต่างๆที่ใช้สำหรับปรับแต่งคอมโพเนนต์พาเล็ต โดยปุ่มเหล่านี้จะทำหน้าที่แตกต่างกันแล้วแต่ว่ากำลังเลือกอยู่ที่ใดดังนี้

Add...	สำหรับเพิ่มกลุ่มใหม่
Delete...	สำหรับลบกลุ่มที่ไม่ต้องการ โยกลุ่มที่ลบต้องเป็นกลุ่มที่ว่างอยู่ คือไม่มีคอมโพเนนต์ใดๆ อยู่ในนั้น
Rename...	สำหรับเปลี่ยนชื่อกลุ่ม
Move Up...	สำหรับเลื่อนลำดับของกลุ่ม หรือคอมโพเนนต์ขึ้น
Move Down...	สำหรับเลื่อนลำดับของกลุ่ม หรือคอมโพเนนต์ลง
Hide...	สำหรับลบคอมโพเนนต์ออกจากกลุ่ม โดยปุ่มนี้จะปรากฏให้ใช้งานเมื่อคลิกที่คอมโพเนนต์ในกรอบ Components
Default Pages	สำหรับกำหนดให้คอมโพเนนต์พาเล็ตแสดงกลุ่มและคอมโพเนนต์ตามดีฟอลต์เหมือนตอนเริ่มต้นทำงานกับ Delphi โดยปุ่มนี้จะปรากฏให้ใช้งานเมื่อคลิกที่ [All] ในกรอบ Pages ทางซ้ายเท่านั้น

เมื่อเราได้รู้จักหน้าที่ของแต่ละปุ่มแล้ว ต่อไปจะแสดงตัวอย่างการปรับแต่งคอมโพเนนต์พาเล็ตโดยจะสร้างกลุ่มใหม่ชื่อ My Components และย้ายคอมโพเนนต์ Edit จากกลุ่ม Standard ไปยังกลุ่มใหม่ที่สร้างขึ้น การเพิ่มกลุ่มและย้ายคอมโพเนนต์มีขั้นตอนดังนี้

1. คลิกปุ่ม **Add** บนแท็บ **Palette** จะปรากฏไดอะล็อกบ็อกซ์ **Add page**
2. ใส่ชื่อกลุ่มใหม่ลงในช่อง **Page name** เช่นในตัวอย่างใส่ชื่อ My Computer
3. คลิกปุ่ม **OK** ในไดอะล็อกบ็อกซ์ Add page
4. ชื่อของกลุ่มที่สร้างขึ้นจะปรากฏในกรอบ Pages
5. คลิกกลุ่ม **System**
6. คลิกและลากคอมโพเนนต์ **TTimer** ไปปล่อยบน My Component
7. คลิกปุ่ม **OK** จะเห็นแท็บ My Component ปรากฏบนคอมโพเนนต์พาเล็ตดังรูป

วินโดว์ออบเจ็กอินสเป็คเตอร์ (Object Inspector Window)

วินโดว์ออบเจ็กอินสเป็คเตอร์ เป็นวินโดว์ซึ่งอยู่วินโดว์หลักทางด้านซ้าย ใช้สำหรับแสดงและปรับแต่งค่าพร็อพเพอร์ตี้ (property) และอีเวนต์ (event) ของคอมโพเนนต์ จากรูปจะเห็นว่าใต้ไตเติลบาร์ของวินโดว์ออบเจ็คอินสเป็คเตอร์ จะเป็นคอมโบบ็อกซ์สำหรับแสดงชื่อและชนิดของคอมโพเนนต์ที่ทำงานอยู่ในขณะนั้น ในรูปเป็น "Form1: TForm1" แสดงว่าเรากำลังทำงานกับคอมโพเนนต์ชื่อ Form1 และมีชนิดของคอมโพเนนต์เป็น TForm1 ถัดลงมาจะมีแท็บ 2 แท็บคือ แท็บ Properties และแท็บ Events ส่วนทางด้านล่างที่เป็น Status Bar จะแสดงจำนวนพร็อพเพอร์ตี้ หรืออีเวนต์ที่ซ่อนไว้

แท็บ Properties

ใช้สำหรับแสดงและกำหนดค่าคุณสมบัติหรือพร็อพเพอตี้ (property) ของคอมโพเนนต์ ภายในแท็บแบ่งออกเป็น 2 คอลัมน์ โดยคอลัมน์ทางซ้ายแสดงชื่อพร็อพเพอร์ตี้ ส่วนคอลัมน์ทางขวามีไว้สำหรับแสดงและแก้ไขค่าของพร็อพเพอร์ตี้

แท็บ Events

ใช้สำหรับกำหนดการทำงานเมื่อมีเหตุการณ์หรือการกระทำใดๆ เกิดขึ้นกับคอมโพเนนต์ ซึ่งอาจเกิดจากการกระทำของผู้ใช้หรือตัวโปรแกรมเอง ตัวอย่างเช่น การใช้เมาส์คลิกที่ปุ่มจะเกิดอีเวนต์ OnClick ขึ้นอยู่กับปุ่มนั้นหรือการกด Enter บน EditBox ก็จะเกิดอีเวนต์ OnKeyDown, OnKeyUp และOnKeyPress ขึ้นมาเมื่อใช้เมาส์คลิกที่แท็บอีเวนต์ออบเจ็คอินสเป็คเตอร์ ดังที่แสดงในรูปจะเห็นว่าภายในแบ่งออกเป็น 2 คอลัมน์ คอลัมน์ทางซ้ายแสดงชื่ออีเวนต์ของคอมโพเนนต์ ส่วนคอลัมน์ทางขวาจะแสดงชื่อโพรซีเยอร์ที่จะทำงานเมื่อเกิดอีเวนต์ขึ้น

การปรับแต่งออบเจ็คอินสเป็คเตอร์

ใน Delphi ได้มีการจัดแบ่งพร็อพเพอร์ตี้ และอีเวนต์ไว้เป็นกลุ่มๆ (Cate Gory) ซึ่งจะช่วยลดจำนวนพร็อพเพอร์ตี้และอีเวนต์ที่ปรากฏในออบเจ็คอินสเปคเตอร์ได้ อีกทั้งยังช่วยให้หาพร็อพเพอร์ตี้ที่มีความสัมพันธ์กันได้ง่ายขึ้นด้วย โดยเราสามารถเลือกดูพร็อพเพอร์ตี้ หรืออีเวนต์เฉพาะกลุ่มที่สนใจ และยังสามารถเลือกจัดลำดับได้อีกด้วย

การเรียงลำดับพร็อพเพอร์ตี้และอีเวนต์

เราสามารถจัดเรียงพร็อพเพอร์ตี้และอีเวนต์ที่ปรากฏในวินโดว์ออบเจ็คอินสเปคเตอร์ได้ 2 ลักษณะคือเรียงตามชื่อและเรียงตามกลุ่ม โดยการคลิกเมาส์ขวาบนออบเจ็คอินสเป็คเตอร์แล้วเลือกเมนู Arrange => by Category ถ้าต้องการเรียงตามกลุ่ม และหากต้องการให้เรียงตามชื่อให้เลือกเมนู Arrange => by Name

การกรองพร็อพเพอร์ตี้และอีเวนต์

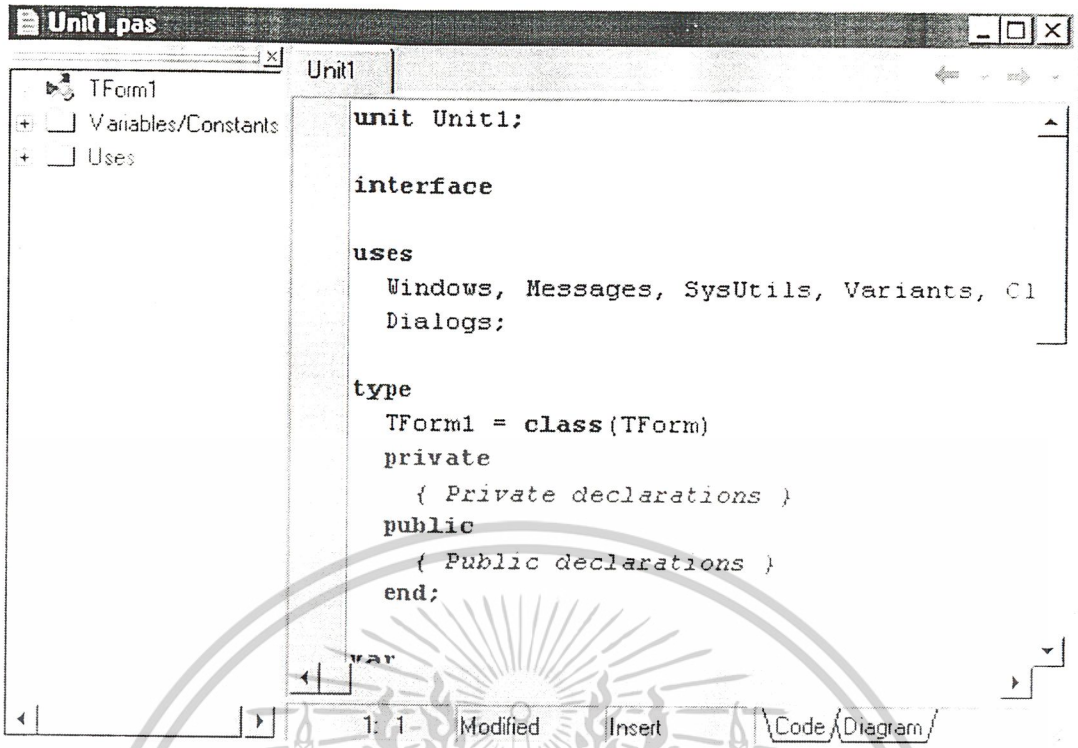
นอกจากการเรียงลำดับพร็อพเพอร์ตี้และอีเวนต์แล้วเรายังกำหนดให้แสดงหรือไม่แสดงกลุ่มของพร็อพเพอร์ตี้และอีเวนต์ได้ด้วย โดยคลิกเมาส์ขวาบนออบเจ็คอินสเปคเตอร์เลือกเมนู View หลังจากนั้นเลือกกลุ่มที่ต้องการให้ปรากฏหรือไม่ปรากฏ ตัวอย่างเช่นถ้าไม่ต้องการแสดงพร็อพเพอร์ตี้กลุ่ม Help and Hints ให้เลือกที่ Help and Hints เพื่อให้เครื่องหมายถูกหายไป พร็อพเพอร์ตี้ในกลุ่มนี้จะไม่ปรากฏบนออบเจ็คอินสเป็คเตอร์

วินโดว์ (Form Window)

ฟอร์มเป็นวินโดว์ที่ใช้สำหรับออกแบบส่วนที่ติดต่อกับผู้ใช้ โดยการนำคอมโพเนนต์ต่างๆ คอมโพเนนต์พาเล็ตมาวางลงบนฟอร์มเมื่อเปิดโปรแกรม Delphi ขึ้นมาหรือสร้างแอพพลิเคชั่นใหม่ Delphi จะสร้างฟอร์มให้หนึ่งฟอร์มเสมอ ในแอพพลิเคชั่นหนึ่งๆจะประกอบด้วยฟอร์มอย่างน้อยหนึ่งฟอร์มเสมอ

วินโดว์โค้ดเอดิเตอร์ (Code Editor Window)

วินโดว์โค้ดเอดิเตอร์มีไว้สำหรับเขียนโปรแกรม โดยอาจจะถูกบังอยู่ใต้ฟอร์ม เมื่อเลื่อนวินโดว์ฟอร์มออกไปหรือคลิกที่ปุ่ม minimize ของฟอร์ม จะเห็นวินโดว์เอดิเตอร์ปรากฏขึ้นดังรูป



รูปที่ 2.21 แสดงวินโดว์โค้ดเอดิเตอร์

ภายในวินโดว์โค้ดเอดิเตอร์ จะประกอบด้วยส่วนต่างๆ ดังต่อไปนี้

Unit Explorer เป็นกรอบที่อยู่ทางด้านซ้าย ซึ่งใช้สำหรับแสดงสิ่งต่างๆที่มีอยู่ในยูนิตนั้น เช่น ออบเจ็ค ตัวแปร ค่าคงที่ และชื่อของยูนิตต่างๆ ที่ถูกอ้างอิงถึงในยูนิต ลักษณะการแสดงรายละเอียดจะอยู่ในรูปแบบของโครงสร้างต้นไม้ (Tree View)

Code editor โค้ดเอดิเตอร์เป็นส่วนที่ใช้เขียนคำสั่ง โดยมีแท็บของชื่อไฟล์ที่ใช้เก็บยูนิตนั้นสำหรับเลือกยูนิตที่ต้องการ

สถานะการทำงาน อยู่ที่ด้านล่างของวินโดว์โค้ดเอดิเตอร์ ซึ่งประกอบด้วยส่วนต่างๆ ดังนี้

- ตำแหน่งของเคอร์เซอร์ ประกอบด้วยหมายเลข 2 ตัวคั่นด้วยเครื่องหมาย : ตัวแรกแสดงหมายเลขบรรทัดและตัวที่สองแสดงหมายเลขคอลัมน์ที่เคอร์เซอร์อยู่บนโค้ดเอดิเตอร์
- สถานะการแก้ไข ใช้บอกว่ายูนิตนี้มีการแก้ไขหรือไม่ กรณีที่แก้ไขแล้วไม่มีการบันทึก ในช่องนี้จะแสดงคำว่า Modified แต่ถ้ายังไม่มีการแก้ไขหรือบันทึกการแก้ไขหรือบันทึกการแก้ไขไปแล้วก็จะไม่มีข้อความใดๆ ปรากฏในช่องนี้
- สถานะการพิมพ์ ใช้บอกสถานะการพิมพ์ในขณะนั้น เช่น เป็นการพิมพ์แทรก (Insert) หรือพิมพ์ทับ (Overwrite)

Code Insight

เพื่อลดข้อผิดพลาดและเพิ่มอย่างรวดเร็วในการเขียนโปรแกรม Delphi ก็มีเครื่องมือที่เรียกว่า *Code Insight* ซึ่งจะช่วยอำนวยความสะดวกให้ โดยมีตั้งแต่เวอร์ชัน 3 ประกอบด้วย

Code completion

ใช้สำหรับแสดงพร็อพเพอร์ตี้, เมธอด หรืออีเวนต์ของออบเจ็คแล้วตามด้วยจุด เช่น ถ้าโปรแกรมมีออบเจ็คชื่อ Form1 ซึ่งเป็นออบเจ็คประเภท TForm1 ขึ้นมาให้เลือก เมื่อเลือกค่าที่ต้องการแล้วกด Enter ค่าที่เลือกจะปรากฏในโค้ดเอดิเตอร์ให้ทันที

Code parameters

เป็นการแสดงพารามิเตอร์ของโพรซิเยอร์หรือฟังค์ชันตามด้วยวงเล็บเปิด จะมีข้อความเล็กๆ แสดงชื่อพารามิเตอร์ให้เห็นตัวอย่างเช่น คีย์ ShowMessage (จะปรากฏพารามิเตอร์ของโพรซิเยอร์ ShowMessage ขึ้นมาเพื่อช่วยให้ผู้ใช้ใส่ค่าพารามิเตอร์ได้อย่างถูกต้อง

Code templates

เป็นการเรียกใช้คำสั่งจากเทมเพลต ซึ่ง Delphi จะเก็บโครงสร้างของคำสั่งต่างๆ ไว้ โดยที่เราสามารถเพิ่มเติมหรือลบคำสั่งในเทมเพลตได้ การใช้งาน Code templates ก็เพียงแต่คีย์ตัวอักษรชุดแรกของคำสั่ง จากนั้นกด Ctrl + J จะปรากฏรายการของคำสั่งในโค้ดเทมเพลตให้เลือก ตัวอย่างเช่น คีย์ case แล้วกด Ctrl + J ก็จะเห็นรายการของคำสั่ง case ขึ้นมาให้เลือก จากนั้นเลือกคำสั่งที่ต้องการแล้วกด Enterคำสั่งที่เลือกก็จะเข้ามาในโค้ดเอดิเตอร์ทันที

จากคุณสมบัติต่างๆ ของ Code template เราสามารถกำหนดให้ทำงานโดยอัตโนมัติหรือไม่ก็ได้ หรือหากต้องการเพิ่มคำสั่งลงใน Code templates ก็สามารถทำได้ โดยกำหนดในไดอะล็อกบ๊อกซ์ Editor Properties ตามขั้นตอนดังนี้

1. คลิกเมนู Tools => Editor Options จะปรากฏไดอะล็อกบ๊อกซ์ Editor Properties
2. คลิกแท็บ Code Insight

ภายใน code Insight มีรายละเอียดดังนี้

Automatic features เป็นการกำหนดให้ Code Insight แต่ละประเภททำงานหรือไม่ โดยเลือกที่เช็คบ๊อกซ์ของหัวข้อต่างๆ และยังสามารถกำหนดเวลาที่จะให้ Code complement และ Code parameters ทำงานได้โดยเลื่อนสไลด์บาร์ที่ใต้ข้อความ Delay ซึ่งจะแสดงเวลาเป็นหน่วยวินาที

Code templates สำหรับเพิ่มเติม แก้ไข และลบคำสั่งที่เก็บไว้ใน Code templates ซึ่งมีการทำงานต่างๆ ดังนี้

Add	สำหรับเพิ่มคำสั่ง
Edit	สำหรับแก้ไขคำสั่ง
Delete	สำหรับลบคำสั่งออกจาก Code templates

บทที่ 3
การออกแบบและการดำเนินงาน

3.1 การวางแผนการดำเนินงาน

ตารางที่ 3-1 ตารางแสดงแผนการดำเนินงาน

รายละเอียดการดำเนินงาน	มิ.ย.	ก.ค.	ส.ค.	ก.ย.	ต.ค.	พ.ย.	ธ.ค.	ม.ค.	ก.พ.	มี.ค.
1.การศึกษาและออกแบบการทำงานของระบบ										
2. การออกแบบในส่วนวงจรควบคุม										
3. การต่อวงจรต่างๆ										
4. การเขียนโปรแกรม (Delphi)										
5. ทดสอบและปรับปรุงแก้ไขโปรแกรม										
6. การเชื่อมต่อเพื่อการนำไปใช้งานจริง										
7. ทดสอบและปรับปรุงครั้งสุดท้าย										
8. สรุปผลและจัดทำปริญญานิพนธ์										

3.2 การออกแบบและส่วนประกอบ

3.2.1 ส่วนประกอบวงจร

ได้ทำการออกแบบชุดควบคุมอุปกรณ์ไฟฟ้าโดยมีลักษณะเป็นกล่องควบคุม (Control Box) ซึ่งภายในชุดควบคุมนี้จะประกอบไปด้วยวงจร 4 ส่วน มีรายละเอียดต่างๆดังต่อไปนี้

Board Controller (MCS-51)

ภายในบอร์ด Controller นี้ประกอบด้วย ไมโครคอนโทรลเลอร์ (89C52) ซึ่งใช้เป็นตัวรับค่าจากคอมพิวเตอร์แล้วจึงนำไปประมวลผลสั่งงานอุปกรณ์ทางอิเล็กทรอนิกส์ โดยก่อนที่ไมโครคอนโทรลเลอร์จะรับค่าหรือส่งค่าไปยังคอมพิวเตอร์นั้นจะต้องผ่านไอซี DS-275 ก่อนเพื่อปรับมาตรฐานการสื่อสารของ RS-232 ให้ตรงกันโดยจะต่อเข้ากับขา RXD และ TXD ของไมโครคอนโทรลเลอร์ และขา WR, RD,T0,T1 และพอร์ต 0 จะต่อเข้ากับไอซี 8255 เพื่อเป็นการขยายพอร์ตเพื่อรองรับอุปกรณ์เอาท์พุตและอินพุตที่เพื่อขึ้น และจะต่อสวิตซ์รีเซตเข้ากับ ไมโครคอนโทรลเลอร์ เพื่อเป็นการรีเซตโปรแกรมหากเกิดการผิดพลาด โดยสาเหตุที่ใช้ไมโครคอนโทรลเลอร์เนื่องจากได้ออกแบบไว้สำหรับเครื่องคอมพิวเตอร์ที่เป็น Server ที่ไม่มีช่องเสียบการ์ด PCI ของไอซี 8255 และการใช้ไมโครคอนโทรลเลอร์นั้นจะทำให้ให้คอมพิวเตอร์ซึ่งเป็นตัวสั่งงานไม่ต้องทำงานหนักเนื่องจากการประมวลผลตลอดเวลา และไมโครคอนโทรเลอร์ เบอร์ 89C52 นั้นมีข้อดีคือเป็นไอซีแบบ 40 ขาซึ่งมีจำนวนพอร์ตมากกว่าแบบ 20 ขาเพื่อที่จะสามารถรองรับการใช้งาน เช่น การต่อกับอุปกรณ์ภายนอกเพื่อสั่งงานและรับค่าได้มากขึ้น และยังมีหน่วยความจำและหน่วยความจำสำรองที่มากกว่า

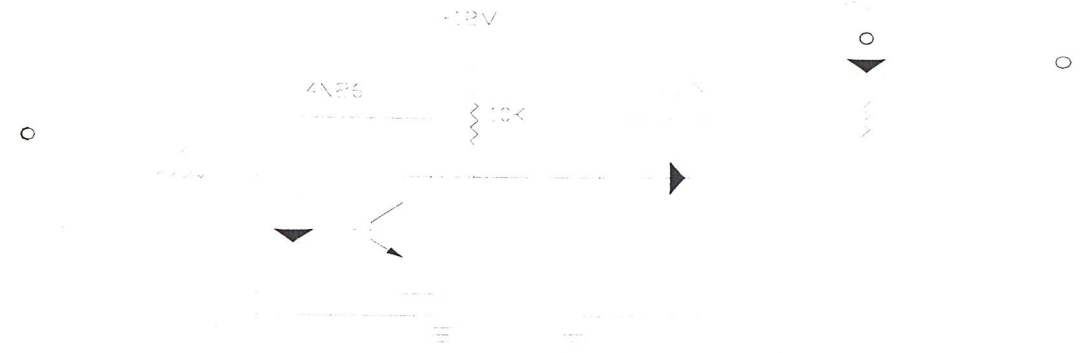
วงจร 8255 (IO Port)

สำหรับ IC 8255 นั้นจะใช้สำหรับเพิ่มจำนวนพอร์ตของ ไมโครคอนโทรลเลอร์ เพื่อรองรับการเพิ่มขึ้นของจำนวนอุปกรณ์ Input และ Output โดยจากวงจรในรูปที่ 3.1 พบว่าจะต่อไอซี 8255 สองตัวเข้ากับไมโครคอนโทรลเลอร์ เพื่อเป็นการขยายพอร์ตที่มีจำนวนจำกัดของไมโครคอนโทรลเลอร์ให้มีขนาดมากขึ้น โดยจะต่อบอร์ด ULN เข้าที่พอร์ต A0 ถึง A6 ของไอซี 8255 ตัวที่ 1 และต่อบอร์ดวงจรเซ็นเซอร์ เข้าที่พอร์ต B0 ถึง B7 ของไอซี 8255 ของไอซีตัวที่ 1 ส่วนพอร์ตที่เหลือของไอซี 8255 ตัวที่ 1 และตัวที่ 2 จะใช้สำหรับรองรับการเพิ่มของอุปกรณ์

รูปที่ 3.1 แสดงการออกแบบวงจร Controller

วงจร ULN – OPTO

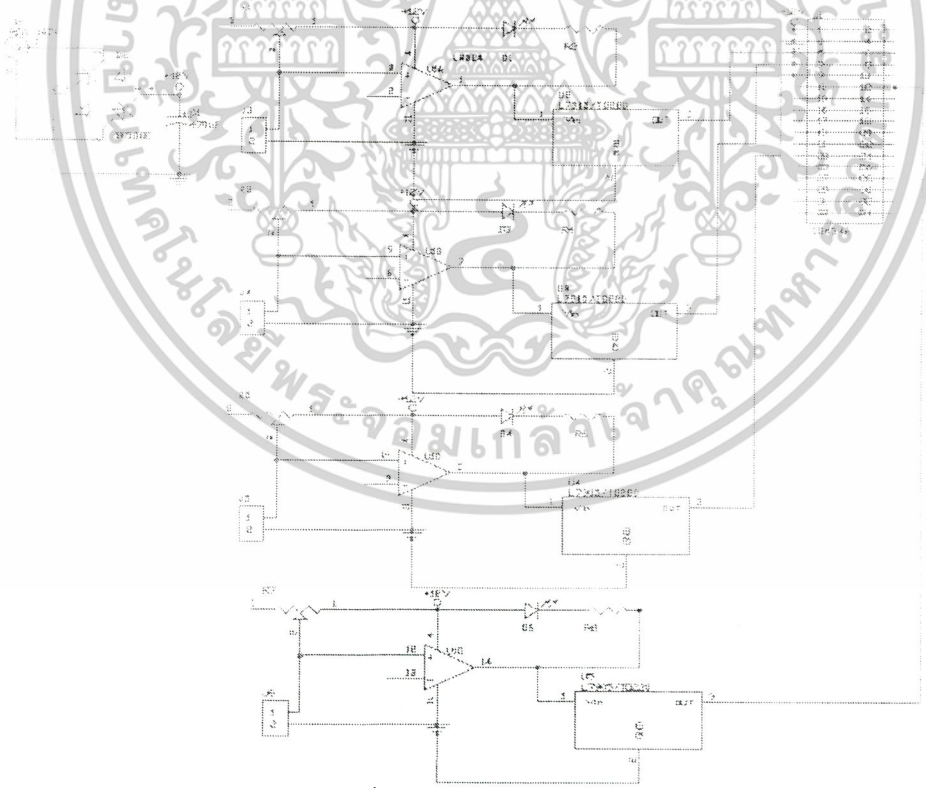
ประกอบไปด้วย IC 4N26 เป็น IC ที่มีความสามารถเหมือนกับสวิตช์ทางแสงตัวหนึ่งโดยภายในวงจรนี้จะใช้สำหรับป้องกันวงจรไมโครคอนโทรลเลอร์หากเกิดการลัดวงจรของอุปกรณ์ไฟฟ้า โดยจะใช้ไอซี 4N26 จำนวน 6 ตัว เมื่อกดปุ่ม Reset ของไมโครคอนโทรลเลอร์จะทำให้บิตต่าง ๆ เป็น 0 ทำให้ Relay จะยังไม่ทำงาน อุปกรณ์ชิ้นต่อไปคือ Relay นั้นจะใช้สำหรับเป็นสวิตช์สำหรับการควบคุมการเปิด-ปิดอุปกรณ์ไฟฟ้าโดย Relay จะทำงานได้ก็ต่อเมื่อมีสัญญาณจาก ไมโครคอนโทรลเลอร์สั่งงาน โดยในวงจรจะใช้ Relay จำนวน 6 ตัว ส่วนประกอบต่อไปคือ IC ULN2003 ใช้สำหรับขับตัว Relay



รูปที่ 3.2 แสดงการออกแบบวงจร ULN - OPTO

วงจร SENSOR

เป็นวงจรที่ใช้สำหรับการตรวจสอบอุปกรณ์ที่เราสั่งงานว่าสามารถทำงานได้หรือไม่ รวมทั้งเป็นวงจรที่ใช้แสดงสถานะแก่ไมโครคอนโทรลเลอร์ว่าอุปกรณ์นั้น ๆ อยู่ในสถานะใด เช่น หลอดไฟเปิดหรือปิดอยู่ เป็นต้น โดยใช้ตัวต้านทานที่ปรับค่าตามแสงสว่าง (LDR) เป็นอุปกรณ์ในการตรวจจับว่าหลอดไฟที่เราสั่งงานไปนั้นได้ทำงานเป็นปกติหรือไม่ และใช้ ไอซี LM 324 เป็นตัวขยายสัญญาณ โดยเอาท์พุตที่ได้จะถูกแปลงเป็นสัญญาณที่มีแรงดันไฟฟ้า 5 โวลต์ โดยผ่านไอซี 7805 เพื่อเป็นสัญญาณทางไฟฟ้าเพื่อส่งไปยังไอซี 8255 ในพอร์ต B



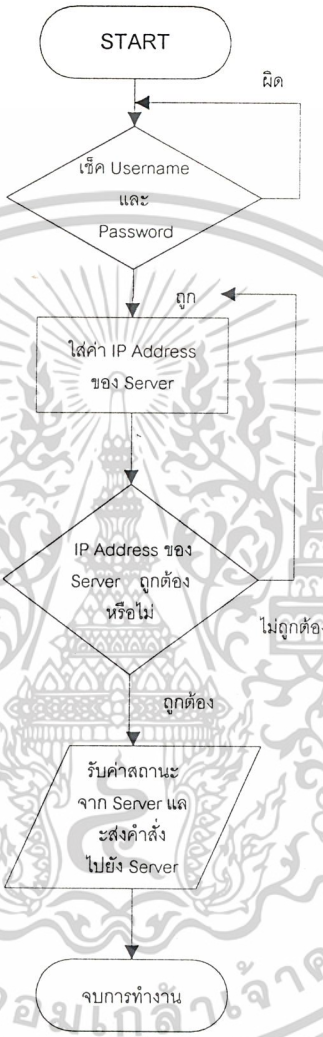
รูปที่ 3.3 แสดงการออกแบบวงจร SENSOR

3.2.2 การออกแบบการทำงานชุดควบคุม

ได้ออกแบบการทำงานของชุดควบคุมอุปกรณ์ไฟฟ้านี้เป็น 3 ส่วน โดยแต่ละส่วนมีรายละเอียดต่างๆดังนี้

- ส่วนของโปรแกรม Client Control

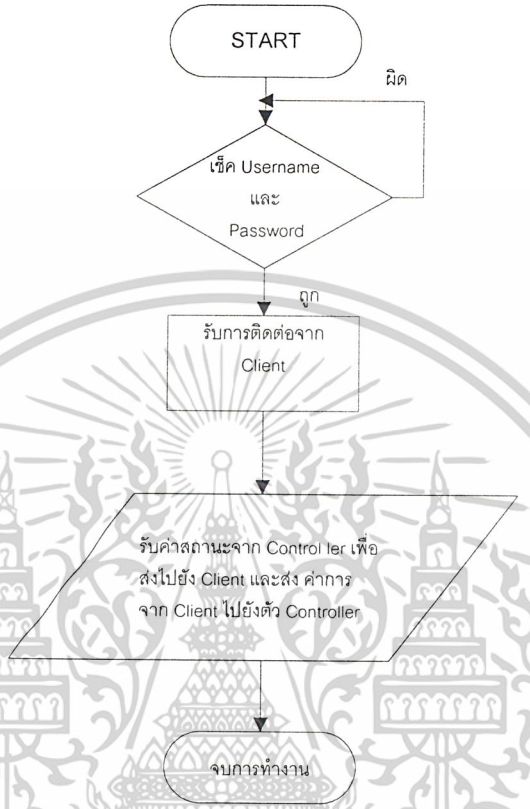
ชุดโปรแกรมนี้สำหรับผู้ใช้ทั่วไปเข้ามาสั่งงานเปิด ปิดไฟผ่านระบบเครือข่ายอินเตอร์เน็ต โดยมีการแสดงสถานะของหลอดไฟในขณะนั้นว่าเสียหรือไม่ให้ผู้ใช้งานทราบด้วย ซึ่งมีหลักการทำงานดังนี้



รูปที่ 3.4 แสดงแผนภูมิการทำงานของโปรแกรมในส่วนของ Client

- ส่วนของโปรแกรม Server Control

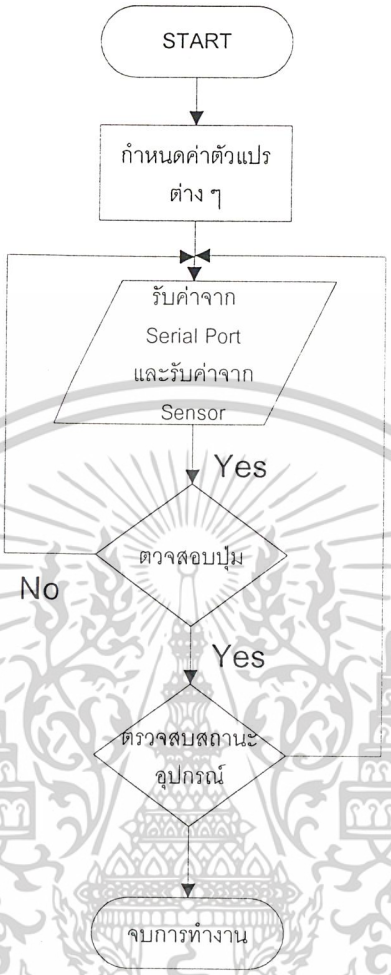
  ชุดโปรแกรม Server Control นี้จะทำการติดตั้งภายในเครื่องคอมพิวเตอร์ซึ่งได้ทำการเชื่อมต่อกับตู้คอนโทรลเลอร์แล้ว โดยชุดโปรแกรมนี้จะให้ผู้ใช้สามารถสั่งงานได้เลยโดยไม่ต้องผ่านระบบเครือข่ายอินเตอร์เน็ต และมีการแสดงสถานะของหลอดไฟในขณะนั้นด้วย



รูปที่ 3.5 แสดงแผนภูมิการทำงานของโปรแกรมในส่วนของ Server

- ส่วนของไมโครคอนโทรลเลอร์

  ในส่วนนี้จะเป็นหลักการทำงานของตัวไมโครคอนโทรลเลอร์



รูปที่ 3.6 แสดงแผนภูมิการทำงานของคอนโทรลเลอร์

บทที่ 4

ผลการทดลอง

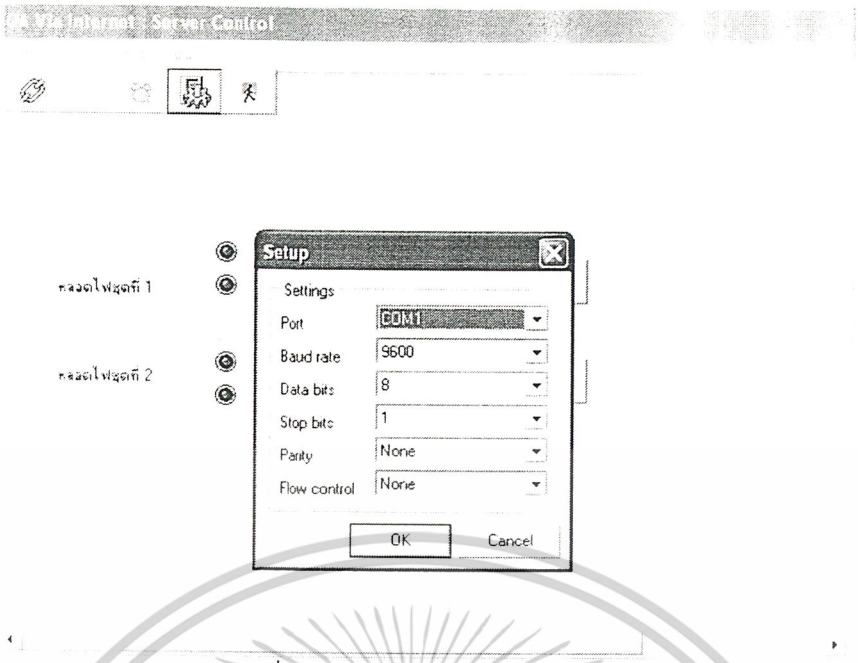
ได้ทำการทดลองเลือกคำสั่งการทำงานโดยสั่งงานทางคอมพิวเตอร์ผ่านระบบเครือข่ายอินเตอร์เน็ต ไปสู่ชุดควบคุมการทำงานเพื่อดูผลการทำงานของชุดควบคุมอุปกรณ์ไฟฟ้าผ่านระบบเครือข่ายอินเตอร์เน็ต ในทดลองนั้นจะทำการแบ่งการทดลองออกเป็นสองส่วน คือ ส่วนควบคุมผ่านserver โดยตรง และ ส่วนควบคุมผ่านเครือข่ายอินเตอร์เน็ต โดยแต่ละชุดคำสั่งจะมีรายละเอียดการทำงานดังต่อไปนี้

4.1 ทดลองสั่งงานผ่านโปรแกรม Server Control

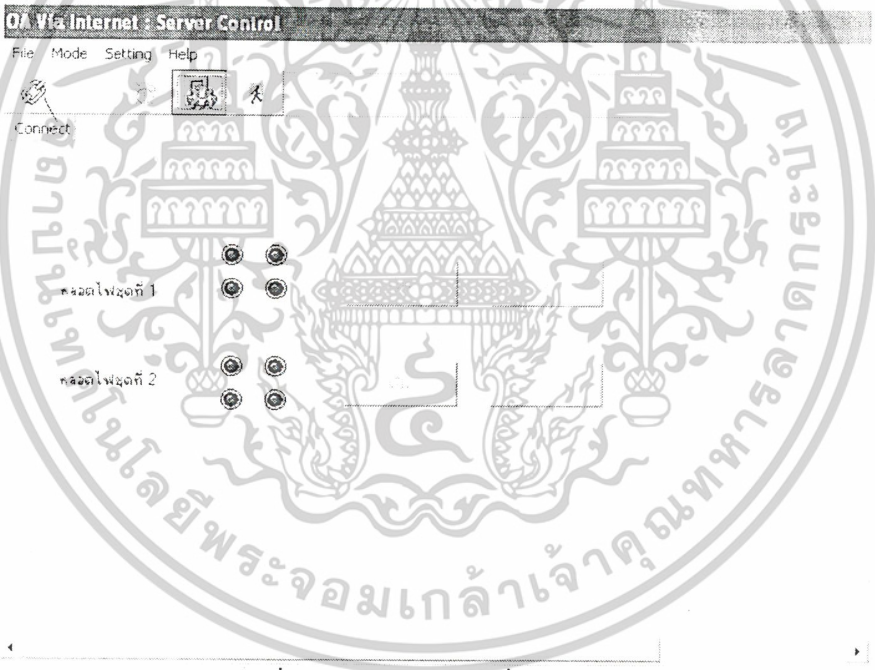
ได้ทำการทดลองสั่งงานเปิดปิดไฟผ่านตัวโปรแกรม Server Control เพื่อทดสอบดูว่าสามารถเปิดปิดไฟได้อย่างแม่นยำ รวมถึงสามารถแสดงสถานะได้ถูกต้องภายในโปรแกรมด้วย ในการทดลองนั้นจะทดสอบเปิดและปิดหลอดไฟชุดที่ 1 และหลอดไฟชุดที่ 2

ขั้นตอนการทำงาน

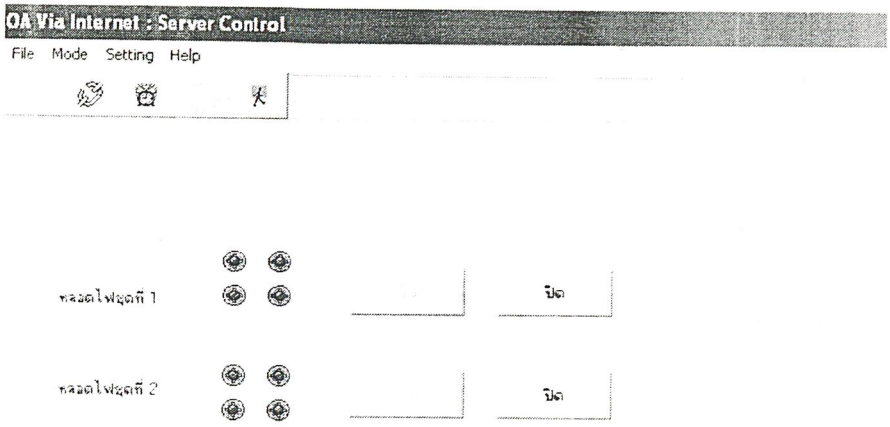
1. เลือก setting เพื่อเซตค่าคอมพอร์ต และ เลือกความเร็วในการส่ง ยืนยันคำสั่ง (OK) (ดังรูปที่ 4.1)
2. กด Connect เพื่อเปิดพอร์ต (ดังรูปที่ 4.2)
3. หลอดไฟทั้งชุดที่ 1 และ 2 จะแสดงสถานะจริงของหลอดไฟในขณะนั้นว่ามีหลอดใดติดบ้าง (ดังรูปที่ 4.3)
4. กดปุ่มเปิด ปิด จากชุดหลอดไฟ (ดังรูปที่ 4.4)
5. หลอดไฟต่างๆจะแสดงสถานะจริงในขณะนั้นว่ามีหลอดใดเสียอยู่บ้าง
6. เลือกโหมด Auto switch ON & OFF เพื่อเลือกสั่งเปิดปิดไฟ โดยการตั้งเวลา (ดังรูปที่ 4.5)
7. เลือกเปิดใช้ระบบอัตโนมัติ และเวลาจริงในขณะนั้นจะแสดงขึ้นมา (ดังรูปที่ 4.6)
8. คลิกเลือกตั้งเวลาเปิดหรือปิด
9. กรอกเวลาที่ต้องการ และเลือกชุดหลอดไฟ (ดังรูปที่ 4.7)
10. เก็บค่าผลการทดลองที่ได้



รูปที่ 4.1 ตัวอย่างการเซ็ตค่าคอมพอร์ต

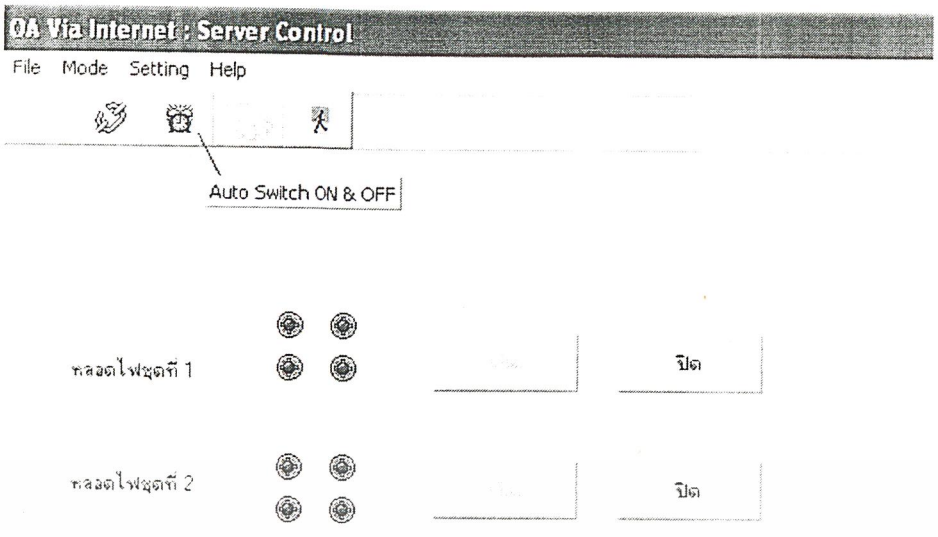


รูปที่ 4.2 กดปุ่ม Connect เพื่อเปิดพอร์ต



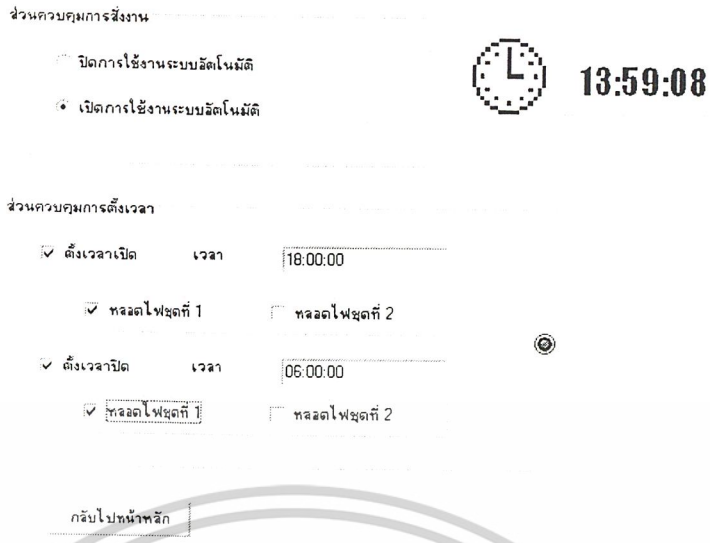
รูปที่ 4.3 ตัวอย่างโปรแกรมเมื่อต่อพอร์ตได้และมีไฟแสดงสถานะขึ้น

รูปที่ 4.4 ตัวอย่างการแสดงสถานะหลังจากลองปิดไฟ



รูปที่ 4.5 กดเพื่อเข้าโหมดตั้งเวลาเปิดปิดอัตโนมัติ

รูปที่ 4.6 ตัวอย่างโหมดตั้งเวลาเปิดปิดอัตโนมัติ



รูปที่ 4.7 ตัวอย่างการตั้งเวลาเปิดปิดอัตโนมัติ

4.2 สั่งงานผ่านระบบเครือข่ายอินเตอร์เน็ต

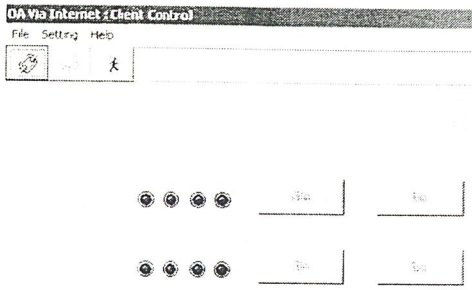
ในส่วนนี้จะแบ่งการทำงานเป็น 2 ส่วนคือ ส่วนของ Client Control กับ Server Standby โดยจะมีชุดควบคุมการทำงานตั้งอยู่ที่ห้อง โดยต่อระบบ Lan อยู่ ส่วนผู้ใช้จะติดต่อเข้ามาสั่งงานผ่าน โปรแกรม Client Control ซึ่งรายละเอียดการทำงานจะมีดังต่อไปนี้

4.2.1 ผู้ใช้ติดต่อเข้ามาผ่านโปรแกรม Client Control

ได้ทำการทดลองสั่งงานด้วยโปรแกรม Client Control โดยจะทำการสั่งงานผ่านระบบเครือข่ายอินเตอร์เน็ต เพื่อดูว่าสามารถเปิดปิดไฟได้และแสดงสถานะในโปรแกรมได้อย่างถูกต้องด้วย ในการทดลองนั้นจะทดสอบเปิดและปิด หลอดไฟชุดที่ 1 และหลอดไฟชุดที่ 2

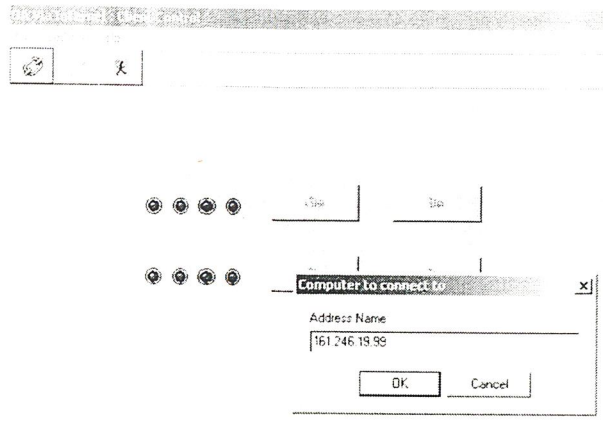
ขั้นตอนการทำงาน

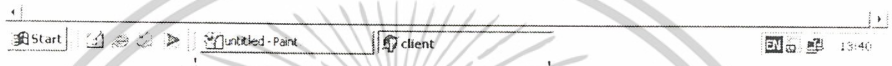
1. กด Connect เพื่อติดต่อไปยัง Server (ดังรูปที่ 4.8)
2. ใส่ IP Address ของเครื่อง Server (ดังรูปที่ 4.9 – 4.11)
3. กด OK เพื่อยืนยันคำสั่ง
4. เมื่อติดต่อได้แล้ว หลอดไฟของชุดที่ 1 และ 2 จะแสดงสถานะจริงในขณะนั้น (ดังรูปที่ 4.12)
5. กดปุ่มเปิดและปิด เพื่อสั่งงานผ่านระบบเครือข่ายอินเตอร์เน็ตไปยังเครื่อง Server
6. หลอดไฟในโปรแกรมจะแสดงสถานะจริงที่เป็นอยู่ของหลอดไฟ เพื่อให้ทราบว่าหลอดใดเสียบ้าง(ดังรูปที่ 4.13)
7. เก็บค่าผลการทดลอง



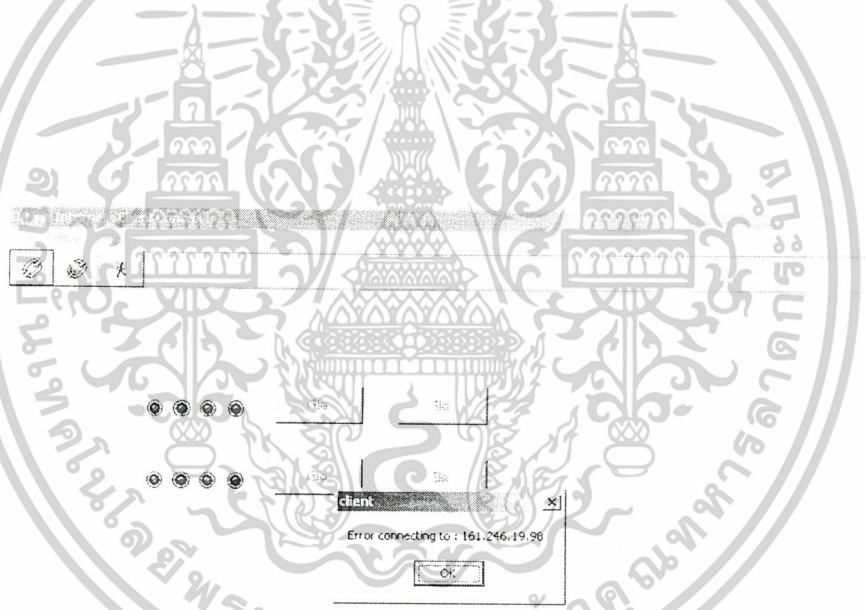
รูปที่ 4.8 กด Connect เพื่อติดต่อไปยัง Server

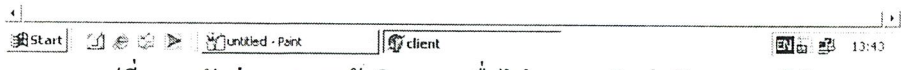
รูปที่ 4.9 ตัวอย่างการใส่ IP Address เพื่อติดต่อกับ Server



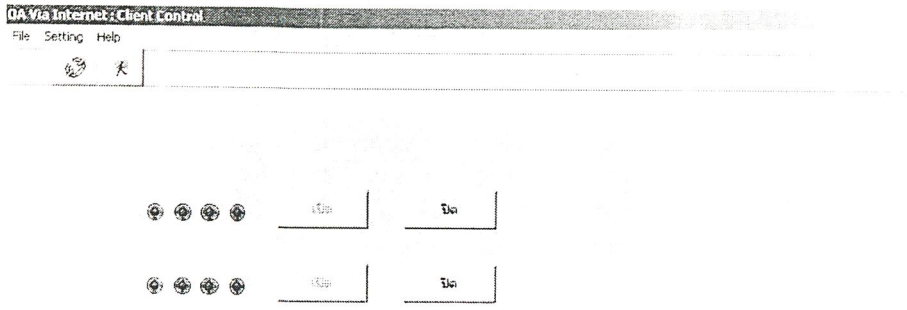


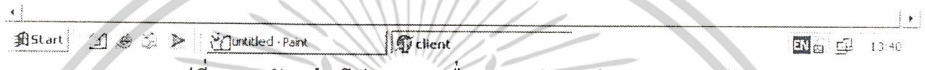
รูปที่ 4.10 ตัวอย่างการใส่ IP Address เพื่อติดต่อกับ Server





รูปที่ 4.11 ตัวอย่างการบอกข้อผิดพลาดเมื่อไม่สามารถติดต่อกับ Server ได้





รูปที่ 4.12 ตัวอย่างโปรแกรมเมื่อสามารถติดต่อกับ Server ได้

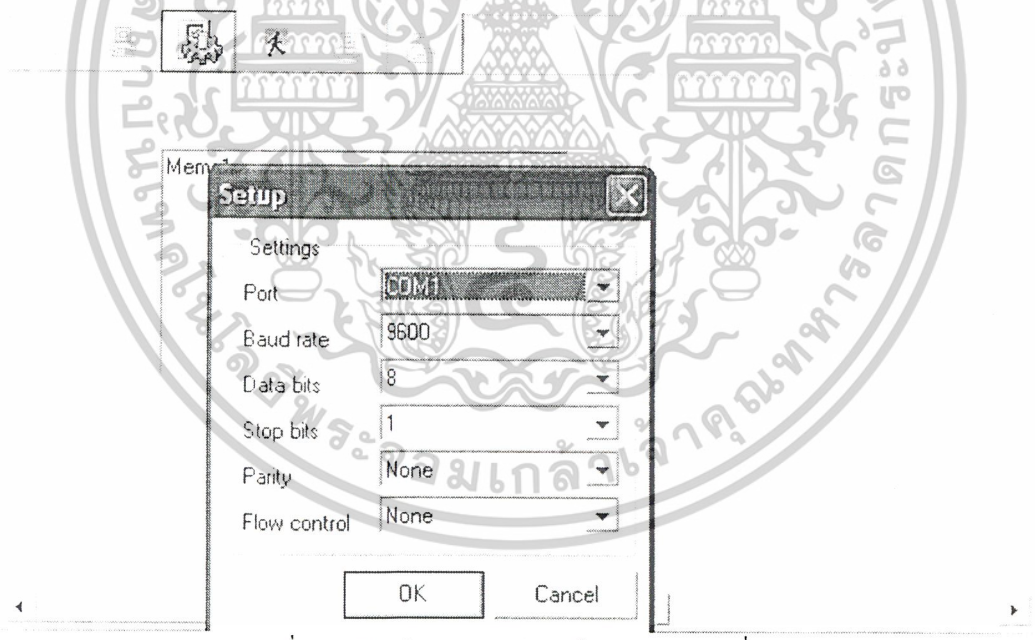
รูปที่ 4.13 ตัวอย่างการเปิดปิดไฟและการแสดงสถานะ

4.2.2 ชุดควบคุมการทำงานด้วยโปรแกรม Server Standby

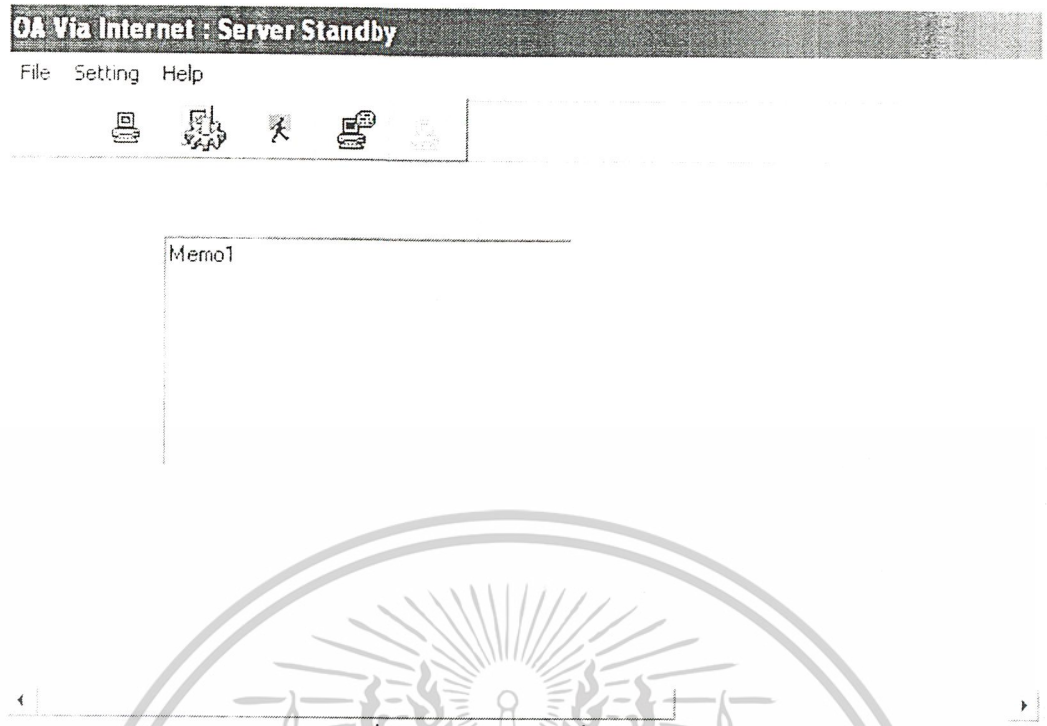
โปรแกรม Server Standby จะทำการติดตั้งไว้ภายในคอมพิวเตอร์ โดยคอมพิวเตอร์เครื่องนั้นจะทำการติดต่อกับตู้คอนโทรลเลอร์ และเชื่อมต่ออินเตอร์เน็ตไว้เรียบร้อยแล้ว โดยการทดลองจะรับการสั่งงานมาจากโปรแกรม Client Control เพื่อไปสั่งเปิด ปิดไฟและแสดงค่าขึ้นมาใน memo ในตัวโปรแกรม โดยเลข 1 จะหมายถึง หลอดไฟชุดที่ 1 และเลข 2 จะหมายถึงหลอดไฟชุดที่ 2 ด้วย ในการทดลองนั้นจะทดสอบเปิดและปิด หลอดไฟชุดที่ 1 และหลอดไฟชุดที่ 2

ขั้นตอนการทำงาน

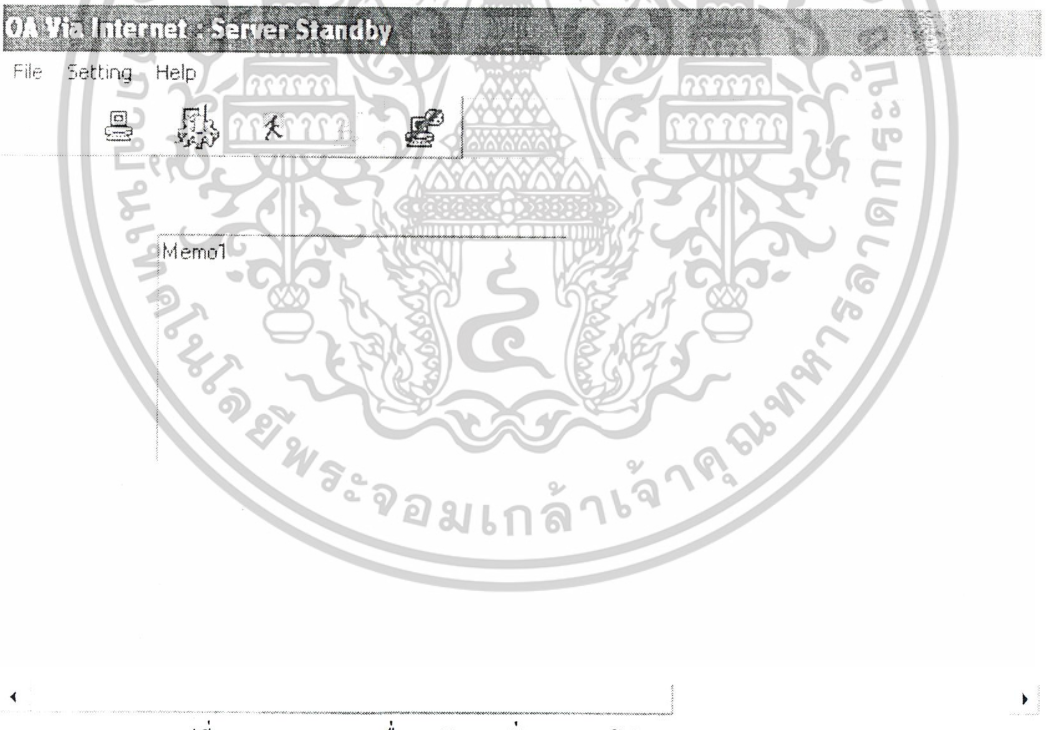
1. เลือก setting เพื่อเซ็ตค่าคอมพอร์ต และ เลือกความเร็วในการส่ง ยืนยันคำสั่ง (OK) (ดังรูปที่ 4.14)
2. กด Connect เพื่อเปิดพอร์ต (ดังรูปที่ 4.15)
3. กด Listen เพื่อรอรับการสั่งงานจากโปรแกรม Client control (ดังรูปที่ 4.16)
4. เมื่อมีผู้ใช้ติดต่อเข้ามาได้แล้ว ใน Memo จะขึ้นคำว่า Connect (ดังรูปที่ 4.17)
5. จากนั้นเมื่อมีการสั่งงานมาแต่ละครั้งจากผู้ใช้ Memo ก็จะทำการบันทึกค่าเอาไว้ (ดังรูปที่ 4.18)
6. เก็บค่าผลการทดลอง



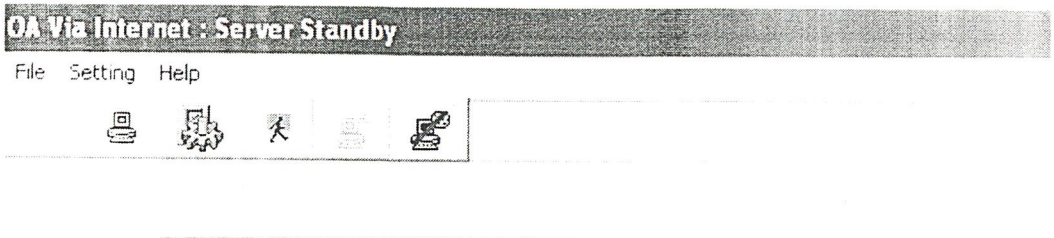
รูปที่ 4.14 ตัวอย่างการเซ็ตค่าพอร์ต และความเร็วที่ส่ง



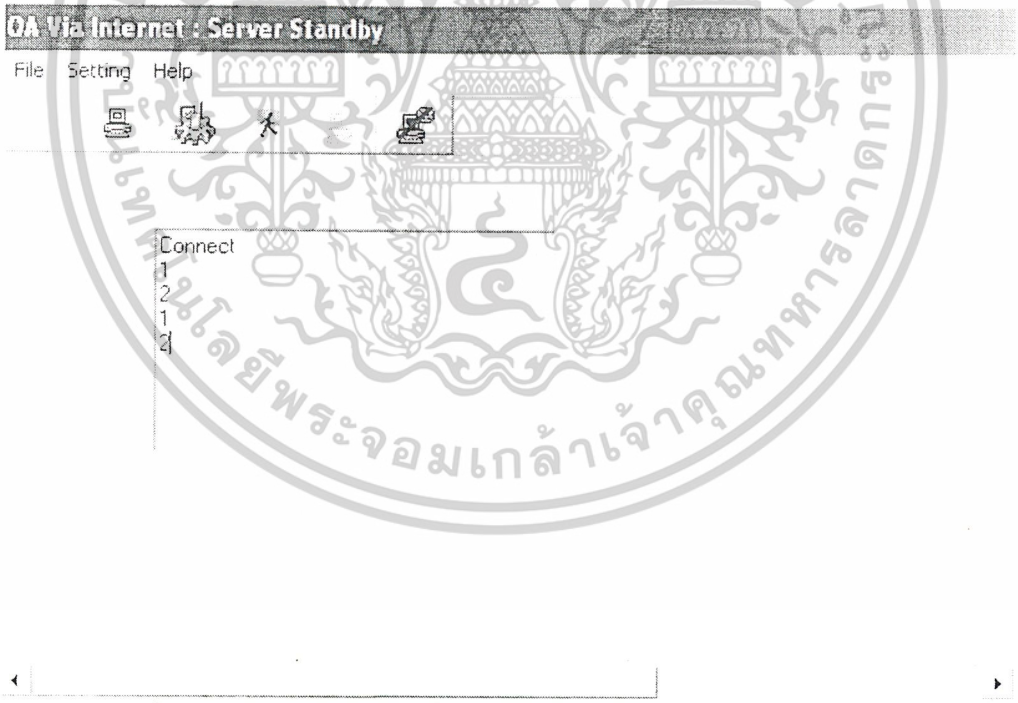
รูปที่ 4.15 กด Connect เพื่อเปิดพอร์ต



รูปที่ 4.16 กด Listen เพื่อรอรับการสั่งงานจากโปรแกรม Client Control



รูปที่ 4.17 Memo จะแสดงคำว่า Connect เมื่อสามารถติดต่อได้แล้ว



รูปที่ 4.18 ภายใน Memo จะบันทึกค่าที่มีการสั่งมาจากโปรแกรม Client Control

4.3 ผลการทดลอง

4.3.1 ทำการสั่งงานผ่านโปรแกรม Server Control

จากการทดลองในห้องที่ติดตั้งกล่องควบคุมโดยสั่งงานผ่านโปรแกรม Server Control สามารถสั่งงานหลอดไฟชุดที่ 1 และหลอดไฟชุดที่ 2 ได้ถูกต้องทั้งในโหมดปกติและโหมดตั้งเวลาเปิดและปิดอัตโนมัติ โดยรวมถึงส่วนแสดงสถานะของโปรแกรม สามารถบอกจำนวนหลอดไฟที่ติดและไม่ติดได้อย่างถูกต้องทุกครั้ง

4.3.2 ทำการสั่งงานผ่านระบบเครือข่ายอินเตอร์เน็ต

จากการสั่งงานจากโปรแกรม Client Control ไปยัง Serverstandby ผ่านระบบเครือข่ายอินเตอร์เน็ต สามารถสั่งงานหลอดไฟชุดที่ 1 และ 2 ได้ ทุกครั้งและไฟแสดงสถานะก็สามารถแสดงได้อย่างถูกต้อง

บทที่ 5

สรุปผลการทดลองและวิธีการแก้ไข

5.1 สรุปผลการทดลอง

จากการทดลองตามที่ได้วางแผนไว้จะได้ว่าชุดควบคุมอุปกรณ์ไฟฟ้าผ่านระบบเครือข่ายอินเตอร์เน็ตสามารถทำงานได้ตามที่ออกแบบไว้ คือ

1. ควบคุมการเปิด – ปิดไฟผ่าน Server
2. ควบคุมการเปิด – ปิดไฟผ่าน Client โดยการสั่งงานผ่านระบบเครือข่ายอินเตอร์เน็ต
3. Mode ต่างๆที่ได้กำหนดเอาไว้ ได้แก่
   3.1 การแสดงสถานะ การเปิด – ปิดไฟ
   3.2 การตั้งเวลาในการเปิด – ปิดไฟ

ซึ่งทำงานได้ตามที่ได้กำหนดในโปรแกรมการทำงานทุกส่วนควบคุม

5.2 ปัญหาที่เกิดขึ้นรวมถึงขีดจำกัด

- โปรแกรม Delphi ไม่มี Component สำหรับการสั่งงานผ่าน Serial Port

ในส่วนของ โปรแกรม Delphi นั้น ไม่มีส่วนของ Component ที่สามารถติดต่อและสั่งงาน Serial Port ของคอมพิวเตอร์ได้โดยตรง ต้องเขียนโปรแกรมที่ติดต่อกับ Serial Port ขึ้นมาเอง หรือ ต้องโหลดมาจาก Website ที่เกี่ยวข้องและให้ความรู้

- ต้องเปลี่ยนวงจรจาก Opto เป็นวงจร Sensor แสง

เนื่องจากวงจร Opto ในการตรวจสอบสถานะนั้นมีข้อจำกัดเมื่อใช้กับสวิตซ์ 3 ทางเพราะจะสามารถตรวจสอบสถานะได้แค่ทิศทางเดียว เฉพาะกับสถานะของไฟ LINE หรือ NEUTRON เท่านั้น เพราะฉะนั้นจึงต้องใช้วงจรเซ็นเซอร์ตรวจจับแสงจากหลอดไฟโดยตรงโดยใช้ IC LM324 และ LDR ในการตรวจจับแสง

- ลายวงจร บางเกินไป

จากการทดสอบพบว่าเมื่อใช้ไฟ 220 V. จ่ายไฟและลองเปิดหรือปิดไฟดูพบว่าลายวงจรจะขาดเนื่องจากลายทองแดงนั้นจะไม่สามารถทนกระแสไฟฟ้าที่มีค่ามาก ๆ ได้จึงต้องทำการแก้ไขโดยการใช้สายไฟติดต่อกันระหว่างอุปกรณ์แทนลายวงจรเพื่อป้องกันความเสียหายที่จะเกิดขึ้นกับลายวงจรในส่วนอื่น

- บัลลาสต์

จากการทดสอบพบว่าหากใช้บัลลาสต์ขดลวดทองแดงธรรมดาพบว่าจะสามารถใช้เปิดหรือปิดได้เพียง 3 ถึง 4 ครั้งเท่านั้น และในครั้งต่อไปก็จะเกิดการลัดวงจรเกิดขึ้นทำให้ตัวรีเลย์เกิดความเสียหายเนื่องจาก ในการปิดสวิตซ์ของหลอดไฟฟ้านั้นจะเกิดประจุไฟฟ้าส่วนหนึ่งจากหลอดไฟฟ้าไหลไปที่บัลลาสต์ในเวลาสั้น ๆ ซึ่งจะทำให้เกิดกระแสไฟฟ้ามากขึ้นมาก ๆ ในขณะนั้นทำให้กระแสในส่วนนั้นไหลไปที่ตัวรีเลย์ซึ่งสามารถทนกระแสได้เพียงค่าหนึ่ง ๆ เท่านั้นจึงทำให้เกิดการลัดวงจร โดยวิธีการแก้ไขคือการเปลี่ยนบัลลาสต์จากแบบธรรมดาเป็นแบบอิเล็คทรอนิกส์ซึ่งพบว่าจะไม่เกิดปัญหานี้ขึ้น

5.3 ประโยชน์ที่ได้รับจากการทำโครงงาน

1.นำความรู้ทางวิศวกรรมที่ได้ศึกษามาใช้ในการออกแบบ และดำเนินการสร้างโครงงานโดยมีการใช้แนวความคิด ทางด้าน Electronics, Programming, Information Technology และความรู้ด้านอื่นๆ ไปประยุกต์ และใช้งานได้จริงตามการทำงานในวงการด้านอุตสาหกรรม

2.เป็นแนวทางในการศึกษาและพัฒนาต่อเกี่ยวกับการควบคุมการทำงานของเครื่องจักรในอุตสาหกรรม รวมถึงการนำไปใช้งานจริงในระบบงานที่ซับซ้อนขึ้นไป

3.การเขียนโปรแกรมคอมพิวเตอร์ เพื่อใช้ในการควบคุมอุปกรณ์ไฟฟ้าให้มีการทำงานได้จริง

4.ทำให้มีความรู้ความเข้าใจในวงจรไฟฟ้า, ระบบเครือข่ายอินเตอร์เน็ต และระบบการส่งถ่ายข้อมูล (Interface) มากขึ้น

5.เพิ่มทักษะในการเขียนโปรแกรมคอมพิวเตอร์ ทั้ง Delphi และภาษา C สำหรับ MCS-51

6. สะดวกต่อการใช้งานและการปฏิบัติการ เพราะว่าใช้คอมพิวเตอร์ในการสั่งงาน ช่วยลดเวลาในการที่คนจะไปสั่งงานที่อุปกรณ์นั้นๆ

7.ทำให้รู้จักการวางแผนและการดำเนินงาน

8. ทำให้รู้จักการทำงานเป็นทีมร่วมกับผู้อื่น (Teamwork)

5.4 ข้อเสนอแนะและการพัฒนาต่อไป

1. สามารถประยุกต์การสั่งงานผ่านระบบเครือข่ายอินเตอร์เน็ตกับส่วนต่างๆ ในวงการอุตสาหกรรมได้ โดยจะต้องประยุกต์วงจรไฟฟ้า โดยดูที่ขนาดกำลังไฟฟ้าของอุปกรณ์นั้น ยกตัวอย่างเช่น หลอดไฟ 1 หลอดใช้กระแสไฟฟ้า 0.43 A รีเลย์ขนาด 10 A จะควบคุมได้ประมาณ 23 หลอด โดยวงจรและขนาดของรีเลย์จะต้องขึ้นอยู่กับกำลังไฟฟ้า และกระแสไฟฟ้าโดยจะต้องสามารถทนแรงดันและกระแสไฟฟ้าได้

2. สามารถพัฒนาให้ใช้กับอุปกรณ์ไฟฟ้าทุกชนิด เพื่อความสะดวกสบายในการใช้ชีวิตประจำวัน

3. เป็นแนวทางในการประยุกต์ Information Technology เข้ามาใช้ในวงการอุตสาหกรรมมากขึ้น

4. พัฒนาในส่วนของวงจรควบคุมให้มีขนาดเล็กลงเพื่อให้ประหยัดเนื้อที่

5. ข้อควรระวังในการใช้บัลลาสต์ขดลวดทองแดงไม่สามารถใช้กับรีเลย์ไฟ 220 V ได้ โดยจะต้องเปลี่ยนมาใช้บัลลาสต์อิเล็กทรอนิกส์

บรรณานุกรม

1) ชัยวัฒน์ ลิ้มพรจิตรวิไล ; "คู่มือนักอิเล็กทรอนิกส์" , ซีเอ็ดยูเคชั่น , 2538

2) ธีรวัฒน์ ประกอบผล ; "การประยุกต์ใช้งานไมโครคอนโทรลเลอร์" , สมาคมส่งเสริมเทคโนโลยี (ไทย – ญี่ปุ่น) , 2542

3) David W. Pessen ; " Industrial Automation (Circuit Design and Components)" , A Wiley-Interscience Publication , 1989

4) อัครเดช สมุทรผ่อง , จักร พิชัยศรทัต ; "ระบบเครือข่ายคอมพิวเตอร์" , ซีเอ็ดยูเคชั่น ,2535

5) กมลมาศ กำธรกิจการ ; "คู่มือ Borland Delphi 5 ฉบับสมบูรณ์" , โปรวิชั่น , 2543

ภาคผนวก ก.

โปรแกรมที่ใช้ในการควบคุมการทำงานผ่าน Server

Source Code ของโปรแกรม Server Control

```
program Project1;
uses
  Forms,
  Unit1 in '..\Project_oa\Project_oa\program\server control\Unit1.pas' {Form1},
  Unit2 in '..\Project_oa\Project_oa\program\server control\Unit2.pas' {Form2},
  Unit3 in '..\Project_oa\Project_oa\program\server control\Unit3.pas' {Form3},
  Unit4 in 'Unit4.pas' {Form4};
{$R *.res}
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.CreateForm(TForm3, Form3);
  Application.CreateForm(TForm4, Form4);
  Application.Run;
end.

unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, jpeg, ExtCtrls;
type
  TForm1 = class(TForm)
    Image1: TImage;
    Timer1: TTimer;
    procedure Timer1Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
```

```
  Form1: TForm1;
implementation
uses Unit2;
{SR *.dfm}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
form1.Hide;
form1.Timer1.Enabled:=false;
form2.show;
end;
end.

unit Unit2;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, StdCtrls, CPort, ExtCtrls, Buttons, CPortCtl;
type
  TForm2 = class(TForm)
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    Connect1: TMenuItem;
    Disconnect1: TMenuItem;
    N1: TMenuItem;
    PortCOM11: TMenuItem;
    PortCOM21: TMenuItem;
    N2: TMenuItem;
    Exit1: TMenuItem;
    Mode1: TMenuItem;
    AutoSwitchON1: TMenuItem;
    AutoSwitchOFF1: TMenuItem;
    Setting1: TMenuItem;
    ComportSetting1: TMenuItem;
    Help1: TMenuItem;
```

```
    About1: TMenuItem;
    ComPort1: TComPort;
    Panel1: TPanel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    ComLed1: TComLed;
    ComLed2: TComLed;
    ComLed3: TComLed;
    ComLed4: TComLed;
    ComLed5: TComLed;
    ComLed6: TComLed;
    ComLed7: TComLed;
    ComLed8: TComLed;
    ComLed9: TComLed;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Button4: TButton;
    Button5: TButton;
    Button6: TButton;
    procedure Button1Click(Sender: TObject);
    procedure About1Click(Sender: TObject);
    procedure Exit1Click(Sender: TObject);
    procedure ComportSetting1Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure Connect1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure Disconnect1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
```

```
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form2: TForm2;
implementation
uses Unit1, Unit3;
{$R *.dfm}
procedure TForm2.Button1Click(Sender: TObject);
begin
comport1.WriteStr('a');
comled1.State:=lson;
comled2.State:=lson;
comled3.State:=lson;
comled4.State:=lson;
button1.enabled:=false;
button2.enabled:=true;
end;
procedure TForm2.About1Click(Sender: TObject);
begin
form3.show;
end;
procedure TForm2.Exit1Click(Sender: TObject);
begin
form1.close;
end;
procedure TForm2.ComportSetting1Click(Sender: TObject);
begin
```

```
comport1.ShowSetupDialog;
bitbtn1.Enabled:=true;
connect1.Enabled:=true;
if comport1.Port = 'COM1' then
      begin
      portcom11.Enabled:=true;
      portcom11.Checked:=true;
      portcom21.Enabled:=false;
      portcom21.Checked:=false;
      end;
if comport1.Port = 'COM2' then
      begin
      portcom21.Enabled:=true;
      portcom21.Checked:=true;
      portcom11.Enabled:=false;
      portcom11.Checked:=false;
      end;
end;
procedure TForm2.BitBtn5Click(Sender: TObject);
begin
form1.Close;
comport1.Close;
end;
procedure TForm2.BitBtn4Click(Sender: TObject);
begin
comport1.ShowSetupDialog;
bitbtn1.Enabled:=true;
connect1.Enabled:=true;
if comport1.Port = 'COM1' then
      begin
      portcom11.Enabled:=true;
      portcom11.Checked:=true;
      portcom21.Enabled:=false;
      portcom21.Checked:=false;
```

```
        end;
    if comport1.Port = 'COM2' then
        begin
        portcom21.Enabled:=true;
        portcom21.Checked:=true;
        portcom11.Enabled:=false;
        portcom11.Checked:=false;
        end;
    end;
    procedure TForm2.BitBtn1Click(Sender: TObject);
    begin
    bitbtn2.Enabled:=true;
    bitbtn3.Enabled:=true;
    disconnect1.Enabled:=true;
    autoswitchon1.Enabled:=true;
    autoswitchoff1.Enabled:=true;
    bitbtn1.Enabled:=false;
    connect1.Enabled:=false;
    bitbtn4.Enabled:=false;
    comportsetting1.Enabled:=false;
    button1.Enabled:=true;
    button3.Enabled:=true;
    button5.Enabled:=true;
    comport1.Open;
    end;
    procedure TForm2.Connect1Click(Sender: TObject);
    begin
    bitbtn2.Enabled:=true;
    bitbtn3.Enabled:=true;
    disconnect1.Enabled:=true;
    autoswitchon1.Enabled:=true;
    autoswitchoff1.Enabled:=true;
    bitbtn1.Enabled:=false;
    connect1.Enabled:=false;
```

```
bitbtn4.Enabled:=false;
comportsetting1.Enabled:=false;
end;
procedure TForm2.BitBtn2Click(Sender: TObject);
begin
bitbtn1.Enabled:=true;
connect1.Enabled:=true;
bitbtn2.Enabled:=false;
disconnect1.Enabled:=false;
bitbtn3.Enabled:=false;
autoswitchon1.Enabled:=false;
autoswitchoff1.Enabled:=false;
bitbtn4.Enabled:=true;
comportsetting1.Enabled:=true;
comport1.Close;
button1.enabled:=false;
button2.enabled:=false;
button3.enabled:=false;
button4.enabled:=false;
button5.Enabled:=false;
button6.Enabled:=false;
end;
procedure TForm2.Disconnect1Click(Sender: TObject);
begin
bitbtn1.Enabled:=true;
connect1.Enabled:=true;
bitbtn2.Enabled:=false;
disconnect1.Enabled:=false;
bitbtn3.Enabled:=false;
autoswitchon1.Enabled:=false;
autoswitchoff1.Enabled:=false;
bitbtn4.Enabled:=true;
comportsetting1.Enabled:=true;
end;
```

```
procedure TForm2.Button2Click(Sender: TObject);
begin
comport1.WriteStr('a');
comled1.State:=lsoff;
comled2.State:=lsoff;
comled3.State:=lsoff;
comled4.State:=lsoff;
button1.enabled:=true;
button2.enabled:=false;
end;
procedure TForm2.Button3Click(Sender: TObject);
begin
comport1.WriteStr('b');
comled5.State:=lson;
comled6.State:=lson;
comled7.State:=lson;
comled8.State:=lson;
button3.enabled:=false;
button4.enabled:=true;
end;
procedure TForm2.Button4Click(Sender: TObject);
begin
comport1.WriteStr('b');
comled5.State:=lsoff;
comled6.State:=lsoff;
comled7.State:=lsoff;
comled8.State:=lsoff;
button3.enabled:=true;
button4.enabled:=false;
end;

procedure TForm2.Button5Click(Sender: TObject);
begin
```

```
comport1.WriteStr('c');
comled9.State:=lson;
button5.enabled:=false;
button6.enabled:=true;
end;
procedure TForm2.Button6Click(Sender: TObject);
begin
comport1.WriteStr('c');
comled9.State:=lsoff;
button5.enabled:=true;
button6.enabled:=false;
end;
end.

unit Unit3;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, Buttons;
type
  TForm3 = class(TForm)
    Image1: TImage;
    Label1: TLabel;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form3: TForm3;
implementation
uses Unit1;
```

```
{$R *.dfm}
procedure TForm3.Button1Click(Sender: TObject);
begin
form3.close;
end;
end.

unit Unit4;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, ExtCtrls, StdCtrls, CPortCtl, Mask, CPort;
type
  TForm4 = class(TForm)
    RadioGroup1: TRadioGroup;
    RadioButton1: TRadioButton;
    RadioButton2: TRadioButton;
    Panel1: TPanel;
    ComPort1: TComPort;
    GroupBox1: TGroupBox;
    MaskEdit1: TMaskEdit;
    MaskEdit2: TMaskEdit;
    ComLed1: TComLed;
    Label1: TLabel;
    Label2: TLabel;
    Timer3: TTimer;
    Timer4: TTimer;
    Button1: TButton;
    Image1: TImage;
    CheckBox1: TCheckBox;
    CheckBox2: TCheckBox;
    GroupBox2: TGroupBox;
    GroupBox3: TGroupBox;
    CheckBox3: TCheckBox;
```

```
    CheckBox4: TCheckBox;
    CheckBox5: TCheckBox;
    CheckBox6: TCheckBox;
    procedure RadioButton1Click(Sender: TObject);
    procedure Timer4Timer(Sender: TObject);
    procedure RadioButton2Click(Sender: TObject);
    procedure Timer3Timer(Sender: TObject);
    procedure RadioButton3Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure CheckBox1Click(Sender: TObject);
    procedure CheckBox2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form4: TForm4;
  DateTime : TDateTime;
  str : string;
implementation
Uses  unit2, Unit1;
{$R *.dfm}
procedure TForm4.RadioButton1Click(Sender: TObject);
begin
timer4.Enabled := false;
panel1.Caption := 'TIME' ;
groupbox1.Enabled := false;
   checkbox1.Enabled := false;
   checkbox2.Enabled := false;
     label1.Enabled    := false;
   label2.Enabled    := false;
   maskedit1.Enabled  := false;
   maskedit2.Enabled  :=false;
```

```
end;
procedure TForm4.Timer4Timer(Sender: TObject);
begin
 DateTime := time;
 str := TimeToStr(DateTime);
 if length(str) = 7 then
 str := ' ' + str;
 panel1.Caption := str;
 end;
procedure TForm4.RadioButton2Click(Sender: TObject);
begin
 if ComPort1.Connected then
  ComPort1.Close
  else
  ComPort1.Open;
  groupbox1.Enabled := true;
  checkbox1.Enabled := true;
  checkbox2.Enabled := true;
    label1.Enabled  := true;
  label2.Enabled   := true;
  maskedit1.Enabled  := true;
  maskedit2.Enabled  := true;
 timer3.Enabled := true;
 timer4.Enabled := true;
 if count1 = 4 then
 checkbox5.Enabled := false
 else
 checkbox3.Enabled := true;
 if count2 = 4 then
 checkbox6.Enabled := false
 else
 checkbox4.Enabled := true;
end;
procedure TForm4.Timer3Timer(Sender: TObject);
```

```
 begin
  if str = maskedit1.text
  then
  begin
  comled1.State := lson;
  if checkbox3.Checked then
  comport1.WriteStr('2') ;
  if checkbox4.Checked then
  comport1.WriteStr('3');
  //timer3.Enabled := false;
  maskedit1.Enabled := false;
  maskedit2.Enabled := true;
   end;
 if str = maskedit2.text
  then
  begin
  comled1.State:=lsoff;
  if checkbox5.Checked then
  comport1.WriteStr('2') ;
  if checkbox6.Checked then
  comport1.WriteStr('3');
  //timer3.Enabled := false;
  maskedit1.Enabled := true;
  maskedit2.Enabled := false;
  end;
end;
procedure TForm4.RadioButton3Click(Sender: TObject);
begin
 timer3.Enabled := true;
end;
procedure TForm4.Button1Click(Sender: TObject);
begin
form2.Show;
comport1.Close;
```

```
timer3.Enabled := false;
timer4.Enabled := false;
end;
procedure TForm4.CheckBox1Click(Sender: TObject);
begin
 if checkbox1.Checked then
 begin
 maskedit1.Enabled := true;
 checkbox3.Enabled := true;
   checkbox4.Enabled := true;
 end
 else
 begin
 maskedit1.Enabled := false;
 checkbox3.Enabled := false;
   checkbox4.Enabled := false;
 end;
end;
procedure TForm4.CheckBox2Click(Sender: TObject);
begin
if checkbox2.Checked then
 begin
 maskedit2.Enabled := true;
   checkbox5.Enabled := true;
   checkbox6.Enabled := true;
 end
 else
 begin
 maskedit2.Enabled := false;
   checkbox5.Enabled := false;
   checkbox6.Enabled := false;
 end;
end;
end.
```

ภาคผนวก ข.

โปรแกรมที่ใช้ในการรับคำสั่งจาก Client

Source Code ของโปรแกรม Server Stand by

```
program Project1;
uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1},
  Unit2 in 'Unit2.pas' {Form2},
  Unit3 in 'Unit3.pas' {Form3};
{SR *.res}
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.CreateForm(TForm3, Form3);
  Application.Run;
end.

unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, jpeg, ExtCtrls;
type
  TForm1 = class(TForm)
    Image1: TImage;
    Timer1: TTimer;
    procedure Timer1Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
Implementation
```

```
uses Unit2;
{$R *.dfm}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
form1.Hide;
form1.Timer1.Enabled:=false;
form2.show;
end;
end.

unit Unit2;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, StdCtrls, CPort, ExtCtrls, Buttons, CPortCtl, ScktComp;
type
  TForm2 = class(TForm)
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    Con1: TMenuItem;
    Dis1: TMenuItem;
    N1: TMenuItem;
    N2: TMenuItem;
    Exit1: TMenuItem;
    Setting1: TMenuItem;
    ComportSetting1: TMenuItem;
    Help1: TMenuItem;
    About1: TMenuItem;
    ComPort1: TComPort;
    Panel1: TPanel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
```

```
    Memo1: TMemo;
    ServerSocket1: TServerSocket;
    listen1: TMenuItem;
    dis2: TMenuItem;
    BitBtn3: TBitBtn;
    BitBtn6: TBitBtn;
    Timer1: TTimer;
    procedure About1Click(Sender: TObject);
    procedure Exit1Click(Sender: TObject);
    procedure ComportSetting1Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure Con1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure Dis1Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure ServerSocket1ClientConnect(Sender: TObject;
      Socket: TCustomWinSocket);
    procedure ServerSocket1ClientRead(Sender: TObject;
      Socket: TCustomWinSocket);
    procedure Timer1Timer(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure ServerSocket1Accept(Sender: TObject;
      Socket: TCustomWinSocket);
  private
    { Private declarations }
  public
    { Public declarations }
  Protected
  Isserver :Boolean;
  end;
```

```
var
  Form2: TForm2;
  server,z: string;
implementation
uses Unit1, Unit3;
{$R *.dfm}
procedure TForm2.About1Click(Sender: TObject);
begin
form3.show;
end;
procedure TForm2.Exit1Click(Sender: TObject);
begin
form1.close;
end;
procedure TForm2.ComportSetting1Click(Sender: TObject);
begin
comport1.ShowSetupDialog;
bitbtn1.Enabled:=true;
con1.Enabled:=true;
end;
procedure TForm2.BitBtn5Click(Sender: TObject);
begin
form1.Close;
comport1.Close;
end;
procedure TForm2.BitBtn4Click(Sender: TObject);
begin
comport1.ShowSetupDialog;
bitbtn1.Enabled:=true;
con1.Enabled:=true;
end;
procedure TForm2.BitBtn1Click(Sender: TObject);
begin
bitbtn1.enabled:=false;
```

```
bitbtn2.Enabled:=true;
bitbtn3.Enabled:=true;
if ComPort1.Connected then
   ComPort1.Close
 else
   ComPort1.Open;
bitbtn4.Enabled:=true;
con1.Enabled:=false;
dis1.Enabled:=true;
listen1.Enabled:=true;
dis2.Enabled:=false;
end;
procedure TForm2.Con1Click(Sender: TObject);
begin
bitbtn2.Enabled:=true;
bitbtn3.Enabled:=true;
dis1.Enabled:=true;
comport1.Open;
bitbtn1.Enabled:=false;
con1.Enabled:=false;
bitbtn4.Enabled:=false;
comportsetting1.Enabled:=false;
end;
procedure TForm2.BitBtn2Click(Sender: TObject);
begin
bitbtn1.Enabled:=true;
bitbtn2.Enabled:=false;
bitbtn3.Enabled:=false;
bitbtn6.Enabled:=false;
con1.Enabled:=true;
dis1.Enabled:=false;
listen1.Enabled:=false;
dis2.Enabled:=false;
 comport1.Connected:=false;
```

```
end;
procedure TForm2.Dis1Click(Sender: TObject);
begin
bitbtn1.Enabled:=true;
con1.Enabled:=true;
bitbtn2.Enabled:=false;
dis1.Enabled:=false;
bitbtn3.Enabled:=false;
bitbtn4.Enabled:=true;
comportsetting1.Enabled:=true;
end;
procedure TForm2.BitBtn3Click(Sender: TObject);
begin
bitbtn6.enabled:=true;
bitbtn3.enabled:=false;
listen1.Enabled:=false;
dis2.Enabled:=true;
listen1.Checked := not listen1.Checked;
 if listen1.Checked then
 begin
  ServerSocket1.Active := True;
 end
 else
 begin
  if ServerSocket1.Active then
   ServerSocket1.Active := False;
 end;
end;
procedure TForm2.BitBtn6Click(Sender: TObject);
begin
bitbtn3.Enabled:=true;
bitbtn6.Enabled:=false;
listen1.Enabled:=true;
dis2.Enabled:=false;
```

```
end;
procedure TForm2.ServerSocket1ClientConnect(Sender: TObject;
  Socket: TCustomWinSocket);
begin
memo1.Lines.Clear;
memo1.Lines.Add('Connect');
end;
procedure TForm2.ServerSocket1ClientRead(Sender: TObject;
  Socket: TCustomWinSocket);
begin
memo1.Lines.Add(socket.ReceiveText);
comport1.WriteStr(memo1.Lines[memo1.Lines.count-1])
end;
procedure TForm2.Timer1Timer(Sender: TObject);
 var  i,j,k :integer;
    b :array[1..8] of integer;
    c :array[1..8] of string;
    x,y : char;
    p:string;
begin
     comport1.Connected:=false;
     comport1.Close;
     comport1.Connected:=true;
     repeat
     comport1.WriteStr('a');
     k:=comport1.ReadStr(z,1);
     until k <>  0;
     i:=ord(z[1]);
     p:=inttostr(i);
     serversocket1.Socket.Connections[0].SendText(p);
     {j:=1;
     while j<= 8 do
     begin
     b[j]:=i mod 2;
```

```
      i:= i div 2 ;
      j:=j+1;
      end;
     for i:= 1 to 8 do
     begin
      c[i] := inttostr(b[i]);
     end;  }
     end;
procedure TForm2.Button1Click(Sender: TObject);
 var i,j,k :integer;
    b :array[1..8] of integer;
    c :array[1..8] of string;
    x,y : char;
    p:string;
begin
     comport1.Connected:=false;
     comport1.Close;
     comport1.Connected:=true;
     repeat
     comport1.WriteStr('a');
     k:=comport1.ReadStr(z,1);
     until k <>  0;
     i:=ord(z[1]);
     p:=inttostr(i);
     serversocket1.Socket.Connections[0].SendText(p);
end;
procedure TForm2.Button2Click(Sender: TObject);
var z:string;k,i:integer;
begin
comport1.Connected:=false;
comport1.Close;
comport1.Connected:=true;
repeat
comport1.WriteStr('a');
```

```
k:=comport1.ReadStr(z,1);
until k <> 0;
serversocket1.Socket.Connections[0].SendText(z[1]);
end;
procedure TForm2.ServerSocket1Accept(Sender: TObject;
 Socket: TCustomWinSocket);
begin
timer1.Enabled := true;
end;
end.

unit Unit3;
interface
uses
 Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
 Dialogs, StdCtrls, ExtCtrls, Buttons;
type
 TForm3 = class(TForm)
  Image1: TImage;
  Label1: TLabel;
  Button1: TButton;
  procedure Button1Click(Sender: TObject);
 private
  { Private declarations }
 public
  { Public declarations }
 end;
var
 Form3: TForm3;
implementation
uses Unit1;
{$R *.dfm}
procedure TForm3.Button1Click(Sender: TObject);
begin
```

```
form3.close;
end;
end.
```

ภาคผนวก ค.

โปรแกรมที่ใช้ในการควบคุมการทำงานผ่าน Client

Source Code ของโปรแกรม Client Control

```
program Project1;
uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1},
  Unit2 in 'Unit2.pas' {Form2},
  Unit3 in 'Unit3.pas' {Form3};
{SR *.res}
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.CreateForm(TForm2, Form2);
  Application.CreateForm(TForm3, Form3);
  Application.Run;
end.

unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, jpeg, ExtCtrls;
type
  TForm1 = class(TForm)
    Image1: TImage;
    Timer1: TTimer;
    procedure Timer1Timer(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
```

```
implementation
uses Unit2;
{SR *.dfm}
procedure TForm1.Timer1Timer(Sender: TObject);
begin
form1.Hide;
form1.Timer1.Enabled:=false;
form2.show;
end;
end.

unit Unit2;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, Menus, StdCtrls, CPort, ExtCtrls, Buttons, CPortCtl, SckComp;
type
  TForm2 = class(TForm)
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    Connect1: TMenuItem;
    Disconnect1: TMenuItem;
    N1: TMenuItem;
    PortCOM11: TMenuItem;
    PortCOM21: TMenuItem;
    N2: TMenuItem;
    Exit1: TMenuItem;
    Setting1: TMenuItem;
    ComportSetting1: TMenuItem;
    Help1: TMenuItem;
    About1: TMenuItem;
    ComPort1: TComPort;
    Panel1: TPanel;
    BitBtn1: TBitBtn;
```

```
BitBtn2: TBitBtn;
BitBtn4: TBitBtn;
BitBtn5: TBitBtn;
ComLed1: TComLed;
ComLed2: TComLed;
ComLed3: TComLed;
ComLed4: TComLed;
ComLed5: TComLed;
ComLed6: TComLed;
ComLed7: TComLed;
ComLed8: TComLed;
Button1: TButton;
Button2: TButton;
Button3: TButton;
Button4: TButton;
ClientSocket1: TClientSocket;
Timer1: TTimer;
Memo1: TMemo;
procedure Button1Click(Sender: TObject);
procedure About1Click(Sender: TObject);
procedure Exit1Click(Sender: TObject);
procedure ComportSetting1Click(Sender: TObject);
procedure BitBtn5Click(Sender: TObject);
procedure BitBtn4Click(Sender: TObject);
procedure BitBtn1Click(Sender: TObject);
procedure Connect1Click(Sender: TObject);
procedure BitBtn2Click(Sender: TObject);
procedure Disconnect1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure Button4Click(Sender: TObject);
procedure Button5Click(Sender: TObject);
procedure Button6Click(Sender: TObject);
procedure ClientSocket1Connect(Sender: TObject;
```

```
    Socket: TCustomWinSocket);
  procedure ClientSocket1Disconnect(Sender: TObject;
    Socket: TCustomWinSocket);
  procedure ClientSocket1Error(Sender: TObject; Socket: TCustomWinSocket;
    ErrorEvent: TErrorEvent; var ErrorCode: Integer);
  procedure Timer1Timer(Sender: TObject);
  { procedure Button7Click(Sender: TObject);}
  { procedure ServerSocket1ClientRead(Sender: TObject;
    Socket: TCustomWinSocket); }
  procedure ClientSocket1Read(Sender: TObject; Socket: TCustomWinSocket);
 private
  { Private declarations }
 public
  { Public declarations }
 Protected
 Isserver: Boolean;
 end;
var
 Form2: TForm2;
 Server: String;
 s :integer;
 p:string;
implementation
uses Unit1, Unit3;
{$R *.dfm}
procedure TForm2.Button1Click(Sender: TObject);
begin
clientsocket1.Socket.SendText('1');
button1.enabled:=false;
button2.enabled:=true;
end;
procedure TForm2.About1Click(Sender: TObject);
begin
form3.show;
```

```
end;
procedure TForm2.Exit1Click(Sender: TObject);
begin
form1.close;
end;
procedure TForm2.ComportSetting1Click(Sender: TObject);
begin
comport1.ShowSetupDialog;
bitbtn1.Enabled:=true;
connect1.Enabled:=true;
if comport1.Port = 'COM1' then
      begin
      portcom11.Enabled:=true;
      portcom11.Checked:=true;
      portcom21.Enabled:=false;
      portcom21.Checked:=false;
      end;
if comport1.Port = 'COM2' then
      begin
      portcom21.Enabled:=true;
      portcom21.Checked:=true;
      portcom11.Enabled:=false;
      portcom11.Checked:=false;
      end;
end;
procedure TForm2.BitBtn5Click(Sender: TObject);
begin
form1.Close;
end;
procedure TForm2.BitBtn4Click(Sender: TObject);
begin
comport1.ShowSetupDialog;
bitbtn1.Enabled:=true;
connect1.Enabled:=true;
```

```
if comport1.Port = 'COM1' then
      begin
      portcom11.Enabled:=true;
      portcom11.Checked:=true;
      portcom21.Enabled:=false;
      portcom21.Checked:=false;
      end;
if comport1.Port = 'COM2' then
      begin
      portcom21.Enabled:=true;
      portcom21.Checked:=true;
      portcom11.Enabled:=false;
      portcom11.Checked:=false;
      end;
end;
procedure TForm2.BitBtn1Click(Sender: TObject);
begin
if ClientSocket1.Active then ClientSocket1.Active := False;
 if InputQuery('Computer to connect to', 'Address Name:', Server) then
   if Length(Server) > 0 then
     with ClientSocket1 do
     begin
       Host := Server;
       Active := True;
 BitBtn2.Enabled:=true;
end;
end;
procedure TForm2.Connect1Click(Sender: TObject);
begin
bitbtn2.Enabled:=true;
disconnect1.Enabled:=true;
bitbtn1.Enabled:=false;
connect1.Enabled:=false;
bitbtn4.Enabled:=false;
```

```
comportsetting1.Enabled:=false;
end;
procedure TForm2.BitBtn2Click(Sender: TObject);
begin
clientsocket1.Active :=false;
end;
procedure TForm2.Disconnect1Click(Sender: TObject);
begin
bitbtn1.Enabled:=true;
connect1.Enabled:=true;
bitbtn2.Enabled:=false;
disconnect1.Enabled:=false;
bitbtn4.Enabled:=true;
comportsetting1.Enabled:=true;
end;
procedure TForm2.Button2Click(Sender: TObject);
begin
clientsocket1.Socket.SendText('1');
button1.enabled:=true;
button2.enabled:=false;
end;
procedure TForm2.Button3Click(Sender: TObject);
begin
clientsocket1.Socket.SendText('2');
button3.enabled:=false;
button4.enabled:=true;
end;
procedure TForm2.Button4Click(Sender: TObject);
begin
clientsocket1.Socket.SendText('2');
button3.enabled:=true;
button4.enabled:=false;
end;
procedure TForm2.Button5Click(Sender: TObject);
```

```
 begin
 clientsocket1.Socket.SendText('c');
 end;
 procedure TForm2.Button6Click(Sender: TObject);
 begin
 clientsocket1.Socket.SendText('c');
 end;
 procedure TForm2.ClientSocket1Connect(Sender: TObject;
  Socket: TCustomWinSocket);
 begin
  BitBtn2.Visible := true;
  BitBtn1.Visible := false;
  connect1.Enabled := false;
  disconnect1.Enabled := true;
  button1.Enabled:=true;
  button3.Enabled:=true;
   memo1.Clear;
 end;
 procedure TForm2.ClientSocket1Disconnect(Sender: TObject;
  Socket: TCustomWinSocket);
 begin
  bitbtn1.Visible := true;
  bitbtn2.Visible := false;
  connect1.Enabled := true;
  disconnect1.Enabled := false;
 end;
 procedure TForm2.ClientSocket1Error(Sender: TObject;
  Socket: TCustomWinSocket; ErrorEvent: TErrorEvent;
  var ErrorCode: Integer);
 begin
 showmessage('Error connecting to : ' + server);
 errorcode :=0;
 end;
 procedure TForm2.Timer1Timer(Sender: TObject);
```

```
var
    i,j,k,l,count1,count2 :integer;
   b :array[1..8] of integer;
   c :array[1..8] of string;
begin
//timer1.Enabled  := true;
j:=1;
    while j<= 8 do
    begin
    b[j]:=s mod 2;
    s:= s div 2 ;
    j:=j+1;
    end;
   for l:= 1 to 8 do
   begin
    c[l] := inttostr(b[l]);
   end;
   {showmessage(c[1]+c[2]+c[3]+c[4]+ c[5]+c[6]+c[7]+c[8]);}
   if c[1] = '1' then comled1.State := lsoff
   else comled1.State := lson;
   if c[2] = '1' then comled2.State := lsoff
   else comled2.State := lson;
   if c[3] = '1' then comled3.State := lsoff
   else comled3.State := lson;
   if c[4] = '1' then comled4.State := lsoff
   else comled4.State := lson;
   if c[5] = '1' then comled5.State := lsoff
   else comled5.State := lson;
   if c[6] = '1' then comled6.State := lsoff
   else comled6.State := lson;
   if c[7] = '1' then comled7.State := lsoff
   else comled7.State := lson;
   if c[8] = '1' then comled8.State := lsoff
   else comled8.State := lson;
```

```
count1:=b[1]+b[2]+b[3]+b[4];
count2:=b[5]+b[6]+b[7]+b[8];
if count1 = 4 then
   begin
   button1.Enabled:=true;
   button2.Enabled:=false;
   end
else
   begin
   button1.Enabled:=false;
   button2.Enabled:=true;
   end;
if count2 = 4 then
   begin
   button3.Enabled:=true;
   button4.Enabled:=false;
   end
else
   begin
   button3.Enabled:=false;
   button4.Enabled:=true;
   end;
{comport1.WriteStr('a');
comport1.ReadStr(z,1);
showmessage(z);
timer1.Enabled := false;
y:=z[1];
i:=ord(y);
j:=1;
while j<= 8 do
begin
x[j]:=i mod 2;
j:=j+1;
end;}
```

```
end;


{procedure TForm2.Button7Click(Sender: TObject);
var
   i,j,k,l :integer;
  b :array[1..8] of integer;
  c :array[1..8] of string;
begin
//timer1.Enabled  := true;
j:=1;
   while j<= 8 do
   begin
   b[j]:=s mod 2;
   s:= s div 2 ;
   j:=j+1;
   end;
  for l:= 1 to 8 do
  begin
   c[l] := inttostr(b[l]);
  end;
  showmessage(c[1]+c[2]+c[3]+c[4]+ c[5]+c[6]+c[7]+c[8]);
  if c[1] = '1' then comled1.State := lsoff
  else comled1.State := lson;
  if c[2] = '1' then comled2.State := lsoff
  else comled2.State := lson;
  if c[3] = '1' then comled3.State := lsoff
  else comled3.State := lson;
  if c[4] = '1' then comled4.State := lsoff
  else comled4.State := lson;
  if c[5] = '1' then comled5.State := lsoff
  else comled5.State := lson;
  if c[6] = '1' then comled6.State := lsoff
  else comled6.State := lson;
  if c[7] = '1' then comled7.State := lsoff
```

```
    else comled7.State := lson;
    if c[8] = '1' then comled8.State := lsoff
    else comled8.State := lson;
end; }
{procedure TForm2.ServerSocket1ClientRead(Sender: TObject;
 Socket: TCustomWinSocket);
   var  p : string;
    i,j,k :integer;
   b :array[1..8] of integer;
   c :array[1..8] of string;
begin
 p:= clientsocket1.socket.ReceiveText;
 memo1.Lines.Add(socket.ReceiveText);
 i:= strtoint(p);
j:=1;
    while j<= 8 do
    begin
    b[j]:=i mod 2;
    i:= i div 2 ;
    j:=j+1;
    end;
   for i:= 1 to 8 do
   begin
    c[i] := inttostr(b[i]);
   end;
   if c[1] = '1' then comled1.State := lsoff
   else comled1.State := lson;
   if c[2] = '1' then comled2.State := lsoff
   else comled2.State := lson;
   if c[3] = '1' then comled3.State := lsoff
   else comled3.State := lson;
   if c[4] = '1' then comled4.State := lsoff
   else comled4.State := lson;
   if c[5] = '1' then comled5.State := lsoff
```

```
    else comled5.State := lson;
    if c[6] = '1' then comled6.State := lsoff
    else comled6.State := lson;
    if c[7] = '1' then comled7.State := lsoff
    else comled7.State := lson;
    if c[8] = '1' then comled8.State := lsoff
    else comled8.State := lson;
end; }

procedure TForm2.ClientSocket1Read(Sender: TObject;
  Socket: TCustomWinSocket);
 var
     p: string;
begin
memo1.Lines.Add(socket.ReceiveText);
p:=memo1.Lines[memo1.Lines.count-1];
s:= strtoint(p);
timer1.Enabled := true;
end;
end.
```