การ์ด IEEE 488 และการโปรแกรม
IEEE 488 AND PROGRAMMING

ผู้จัดทำ

นายอดิศักดิ์ สายอ้อง รหัส 33100475

นายอนุรักษ์ อ่อนรักษ์ รหัส 33100485

อาจารย์ที่ปรึกษา

อาจารย์ขนิษฐา แซ่ตั้ง

ปริญญานิพนธ์ฉบับนี้ เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหาร ลาดกระบัง

ปีการศึกษา 2536

ปริญญานิพนธ์ปีการศึกษา 2536

ภาควิชา อิเล็กทรอนิคส์

คณะ วิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหาร ลาดกระบัง

เรื่อง การ์ด IEEE 488 และการโปรแกรม (IEEE 488 CARD AND PROGRAMMING)

ผู้จัดทำ

1. นายอดิศักดิ์ สายอ๋อง รหัส 33100475
2. นายอนุรักษ์ อ่อนรักษ์ รหัส 33100485

_____อาจารย์ที่ปรึกษา

(อาจารย์ชนิษฐา แซ่ตั้ง)

การ์ด IEEE 488 และการโปรแกรม
IEEE 488 CARD AND PROGRAMMING

โดย	นายอติศักดิ์	สายอ่อง	33100475
	นายอนุรักษ์	อ่อนรักษ์	33100485
อาจารย์ที่ปรึกษา	อาจารย์ขนิษฐา		แซ่ตั้ง

บทคัดย่อ

โครงงานการ์ด IEEE 488 และการโปรแกรม นี้เป็นวิธีการนำคอมพิวเตอร์มาประยุกต์ใช้ให้ทำงานเป็นตัวควบคุมระบบ ในระบบที่มีมาตรฐานการอินเทอร์เฟสแบบ IEEE 488 โครงงานนี้แบ่งออกได้เป็น 2 ส่วนคือ ส่วนแรกเป็นส่วนของฮาดแวร์ซึ่งประกอบด้วยส่วนของอินพุท/เอ้าต์พุทพอร์ต ส่วนบัฟเฟอร์ และส่วนของวงจรเลือกแอดเดรส และอีกส่วนหนึ่งคือส่วนซอฟแวร์ที่สนับสนุนการ์ดนี้ ให้ทำงานได้ตามฟังก์ชั่นที่ใช้กันในมาตรฐาน IEEE 488 โดยได้แบ่งส่วนซอฟแวร์ออกเป็น 2 ส่วนคือ ในส่วนของโปรแกรมสำเร็จรูปซึ่งผู้ใช้สามารถใช้ในการติดต่อทั่ว ๆ ไปกับเครื่องมือวัดได้ทันที เช่นการอ่านข้อมูลจากเครื่องมือวัดในระบบแล้วนำมาทำการประมวลผลบนคอมพิวเตอร์ และอีกส่วนหนึ่งเป็นส่วนของโปรแกรมยูนิตที่ผู้ใช้สามารถนำไปใส่ไว้ในโปรแกรมที่เขียนขึ้นใช้ในงานเฉพาะอย่าง ซึ่งผู้ใช้จะต้องรวมเอายูนิต IEEE 488 เข้าไว้ในโปรแกรมด้วย

ABSTRACT

This IEEE 488 Card and programming project presents the mean in which the users can put this interface card into their computer to operate it as a controller in the IEEE 488 standard system. There are two parts in this project. One is about the hardware which composed of the input/output port circuit ,buffer and the decoder circuit. The other part is the software supporting this card,performs the function that used in general IEEE 488 standard systems. In addition,we divide this software into two sections of use,one is the ready for use program which the you can generally use it for communicating with your remoted instruments,reads data and manipulates them on your computer,the other one is a group of program units,you can put them into your program to perform any special works ,In this way ,you must also include IEEE488 unit into your program.

สารบัญ

บทที่ 1

GPIB (IEEE 488)
บัสอินเทอร์เฟสมาตรฐาน

กล่าวนำ

ความจำเป็นที่จะต้องมีระบบบัสมาตรฐานขึ้นมาใช้งานกัน ก็เนื่องมาจากเครื่องมือวัดที่ทันสมัยและมีความสามารถสูง อาศัยการทำงานในเชิงดิจิตอลเข้ามาเกี่ยวข้องด้วยและในการใช้งาน เมื่อจำเป็นต้องมีการต่อร่วมกันมากกว่า 1 เครื่อง แต่เดิมนั้นแต่ละผู้ผลิตจะมีระบบเชื่อมต่อของตนเอง ทำให้ไม่สะดวกต่อผู้ใช้งานที่มีเครื่องมือจากหลายบริษัท และเป็นผลเสียต่อผู้ผลิตเครื่องมือวัดเองด้วยในการพัฒนาเครื่องมือใหม่ ๆ ออกมา

บริษัทต่าง ๆ ที่เป็นผู้ผลิตเครื่องมือวัดในอเมริกา จึงได้ตกลงร่วมกันที่จะจัดหาระบบอินเทอร์เฟสมาตรฐานสำหรับที่จะใช้ร่วมกันขึ้นมา และในประเทศเยอรมันก็มีการพัฒนาระบบบัสมาตรฐานขึ้นมาเช่นกัน โดยความร่วมมือช่วยเหลือของ IEC (International Electrotechnical Commission) จนในปี 1972 อเมริกาโดย IEEE จึงได้มีการประชุมร่วมมือกันกับ IEC วางแผนพิจารณาระบบบัสมาตรฐานร่วมกัน

บริษัทฮิวเลตแพกการ์ด ผู้ผลิตเครื่องมือวัดรายใหญ่รายหนึ่งในอเมริกาได้ทำการพัฒนาระบบบัสมาตรฐานของตัวเองอยู่ก่อนแล้ว ชื่อว่า HPIB (Hewlett Packard Interface bus) จึงได้เสนอระบบบัสของตัวเองให้ IEEE พิจารณาและได้รับการยอมรับในในปี 1975 เรียกว่ามาตรฐาน IEEE Std 488-1975 และมีการปรับปรุงต่อมาเป็น IEEE Std 488-1978 ระบบบัสนี้ก็คือ IEEE 488 ที่กล่าวถึงนั่นเอง สำหรับของ IEC มีการกำหนดมาตรฐานขึ้นมาอีกอันหนึ่งเรียกว่า IEC625-1 ตั้งใจที่จะให้เป็นมาตรฐานสากล โดยมีรายละเอียดทางเทคนิคทุกประการเหมือนกับ IEEE Std 488-1978 เพียงแต่การวางตำแหน่งของสัญญาณต่าง ๆ ในขั้วต่อต่างกันเล็กน้อยเท่านั้นเอง คำว่า GPIBจึงหมายรวมถึงทั้ง 2 มาตรฐานดังกล่าว

จุดประสงค์ของบัสก็คือ การส่งข้อมูลข่าวสารถ่ายทอดกันระหว่างเครื่อง ข่าวสารที่ว่านี้ก็มีทั้งข้อมูลที่เป็นผลลัพธ์จากการวัดค่าหรือจากกระบวนการต่าง ๆ และรวมถึงคำสั่งที่ใช้กำหนดสภาวะทำงานของอุปกรณ์ต่าง ๆ ในระบบ

เมื่อมีการต่อเครื่องมือวัดหรืออุปกรณ์ที่ต้องการใช้งานร่วมกันเข้ากับระบบบัสแล้วจำเป็นต้องมีแบบแผนในการทำงานร่วมกันให้สอดคล้องกันจึงต้องมีสัญญาณสำหรับควบคุมระบบ นอกเหนือจากข่าวสารข้อมูลที่ต้องมีการติดต่อกัน (สัญญาณควบคุมนี้ ใช้สำหรับควบคุมบัสและจัดระบบสำหรับการอินเทอร์เฟส อาจมองได้ว่าเป็นความเกี่ยวข้องทางฮาดแวร์ ไม่เกี่ยวข้องกับผู้ใช้งานภายนอก ส่วนคำสั่งสำหรับกำหนดการทำงานของอุปกรณ์ในระบบเป็นงานทางด้านซอฟแวร์ที่เกี่ยวข้องกับการใช้งานโดยตรง)

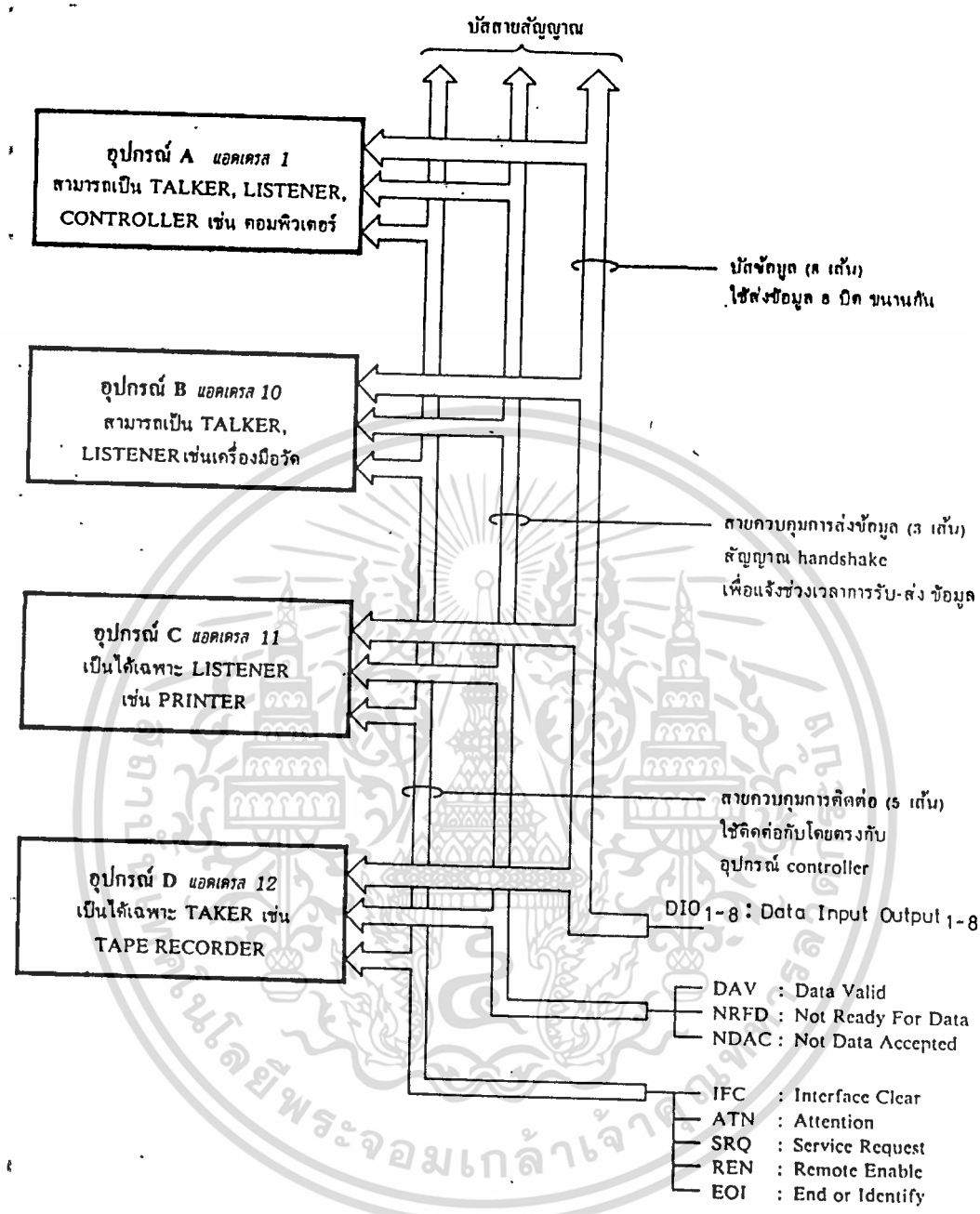
บทที่ 2

คุณสมบัติและการใช้งานทั่วไป

GPIB มีชื่อเต็มว่า General Purpose Interface Bus เป็นบัสมาตรฐานที่ใช้ในการติดต่อรับส่งข้อมูลระหว่างอุปกรณ์ได้หลายเครื่อง โดยมีข้อพิเศษสำหรับผู้ใช้ก็คือในกรณีที่ต้องการขยายอุปกรณ์เพิ่มเติมเข้ามาในระบบ ผู้ใช้ไม่จำเป็นต้องเพิ่มเติมส่วนวงจรหรืออุปกรณ์อื่น ๆ อีกเลย เพียงแต่ผู้ใช้เพิ่มสายเคเบิ้ลเชื่อมต่อเข้ามาขนานกับสายเคเบิ้ลหรือขั้วต่อเดิมเท่านั้น โดยใช้การแก้ไขเฉพาะส่วนของซอฟแวร์

ในระบบที่ไม่ใหญ่โตนัก GPIB นั้นก็สามารถต่อเข้ากับอุปกรณ์หรือเครื่องมืออื่น ๆ ได้สูงสุด 15 เครื่อง โดยใช้สายสัญญาณเพียง 1 เส้นต่อขนานกันไปเรื่อย ๆ ดังนั้นบัสสัญญาณของ GPIB จึงเป็นบัสแบบขนานโดยมีสัญญาณควบคุมร่วมด้วย เพื่อกำหนดทิศทางและเลือกตัวอุปกรณ์ที่ต้องการจะติดต่อ

หากจะเปรียบเทียบกับมาตรฐานการรับส่งของ RS-232 หรือ Centronics Interface แล้ว GPIB มีข้อยุ่งยากและซับซ้อนกว่าในแง่การใช้งาน เพราะต้องมีการใช้คำสั่งควบคุมอุปกรณ์แต่ละตัวก่อนที่จะรับส่งข้อมูลกันได้



รูปที่ 2.1 แผนผังแสดงอุปกรณ์ GPIB และสายสัญญาณต่าง ๆ

โครงสร้างของ GPIB

ส่วนประกอบพื้นฐานของ GPIB แสดงไว้ในรูปที่ 2.1 กล่าวคือ GPIB ประกอบไปด้วยอุปกรณ์ที่ทำหน้าที่เป็นตัวส่ง (Talker),ตัวรับ (Listener) และตัวควบคุม (Controller)

Talker ทำหน้าที่ในการส่งข้อมูลโดยสามารถที่จะนำ Talker จำนวนหลาย ๆ ตัว มาใส่ไว้ในระบบ แต่จะมี Talker เพียงตัวเดียวเท่านั้นที่กำลังทำงานอยู่

Listener ทำหน้าที่รับข้อมูล Listener ก็เช่นเดียวกับ Talker คือ สามารถนำไปใส่ไว้ในระบบได้หลาย ๆ ตัว ดังนั้น ในระบบหนึ่ง ๆ จึงประกอบด้วย Talker 1 ตัว และ Listener ตั้งแต่ 1 ตัวขึ้นไป

Controller เป็นตัวควบคุมสัญญาณต่าง ๆ บนบัส โดยรับความต้องการของอุปกรณ์ที่จะส่งข้อมูลหรือกำหนด Listener ที่จะทำการรับข้อมูล

อุปกรณ์ที่มีระบบ GPIB

อุปกรณ์หรือเครื่องมือที่มีระบบ GPIB นั้นแบ่งตามหน้าที่การทำงานได้ดังนี้

1. ทำหน้าที่เป็น Talker เท่านั้น เช่น เครื่องมือวัด เป็นต้น
2. ทำหน้าที่เป็น Listener เท่านั้น เช่น เครื่องพิมพ์ เครื่องบันทึก เป็นต้น
3. ทำหน้าที่เป็นได้ทั้ง Talker และ Listener เช่น Computer,เครื่องมือวัดที่สามารถควบคุมได้จากภายนอก เป็นต้น
4. ทำหน้าที่เป็นได้ทั้ง Listener,Talker และ Controller เช่น Computer ที่ทำหน้าที่ควบคุมระบบ

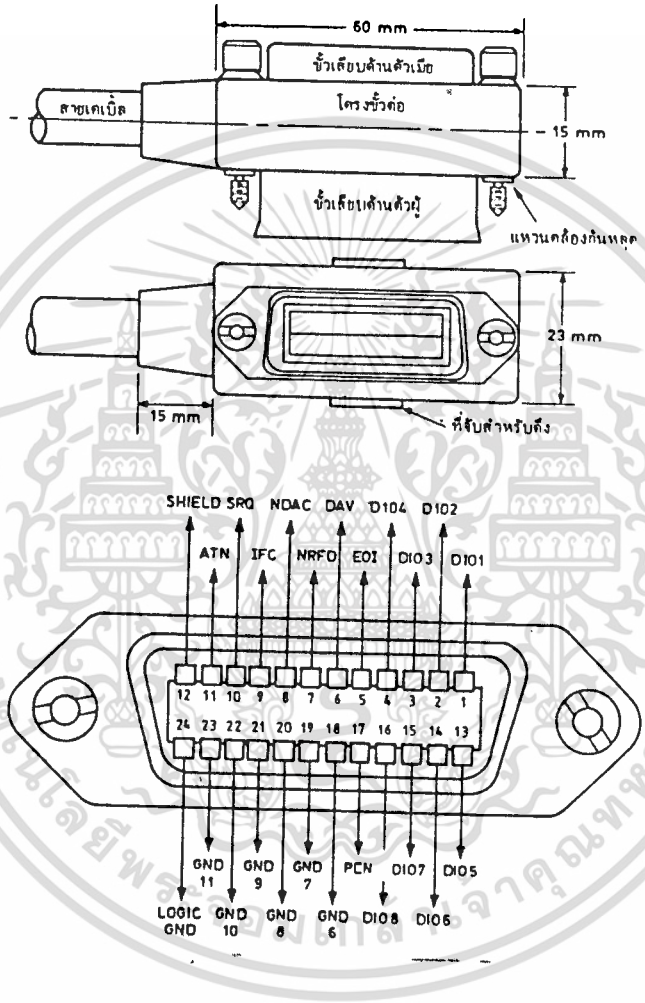
คุณสมบัติทางไฟฟ้าของ GPIB

คุณสมบัติทางไฟฟ้าซึ่งจะเป็นตัวกำหนดขีดจำกัดของ GPIB นั้น มีดังนี้

1. จำนวนอุปกรณ์ (Talker,Listener,Controller) ที่ต่ออยู่กับสายสัญญาณ 1 เส้น จะต้องไม่เกิน 15 เครื่อง
2. สายเคเบิ้ลที่ใช้ต่ออยู่ระหว่างอุปกรณ์แต่ละตัวต้องยาวไม่เกิน 4 เมตร และความยาวรวมของสายเคเบิ้ลต้องไม่เกิน 20 เมตร
3. ความเร็วในการส่งข้อมูลต้องไม่เกิน 1 Mb/Sec
4. จำนวนของอุปกรณ์หรือเครื่องมือมากกว่าครึ่งหนึ่งต้องเปิดให้ทำงาน (จ่ายไฟ)

ขั้วต่อของ GPIB

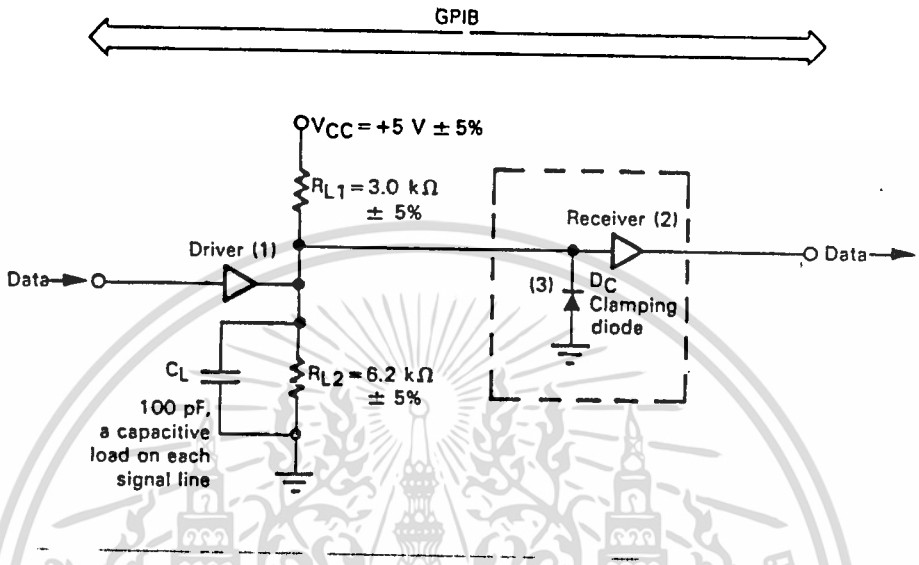
ขั้วต่อมาตรฐานของ GPIB มีทั้งหมด 24 ขา และมีการจัดตำแหน่งสัญญาณที่ขาต่าง ๆ ดังรูปที่ 2.2



รูปที่ 2.2 ขั้วต่อของ GPIB และการจัดขาของสัญญาณต่าง ๆ

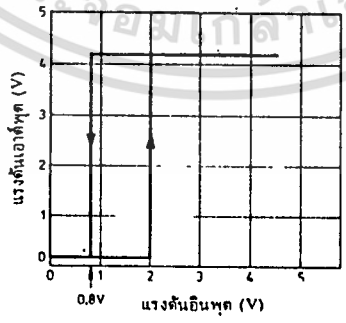
พิจารณาถึงวงจรทางไฟฟ้า

เมื่อพิจารณาถึงวงจรไฟฟ้าเมื่อมองย้อนเข้าไปยังจุดต่อสัญญาณบัสข้อมูล เราจะเห็นโหลดของ GPIB เป็นตัวต้านทานแบ่งแรงดัน 2 ค่าคือ 6.2 kOhm และ 3 kOhm ดังในรูปที่ 2.3



รูปที่ 2.3 วงจรภายในจุดต่อบัสข้อมูลที่ใช้รับส่งข้อมูล

ทางด้านเกตตัวส่ง (Driver) เป็นแบบ Open collector หรือ แบบเกต 3 สถานะ (3 State Driver) โดยขับด้วยกระแสขนาด 48 mA. ส่วนทางด้านเกตตัวรับ (Receiver) ใช้แบบ Schmitt trigger เพื่อกำจัดสัญญาณรบกวน โดยมีคุณสมบัติการให้สัญญาณเอ้าต์พุทดังรูปที่ 2.4



รูปที่ 2.4 คุณสมบัติฮิสเตอริซิสของตัว Receiver แบบ Schmitt trigger

จากรูปที่2.4 ที่ระดับแรงดันอินพุทมีค่าต่ำหรือน้อยกว่า 0.8 V เอ้าต์พุทจะมีค่าต่ำด้วย แต่ถ้าเพิ่มแรงดันอินพุทไปเรื่อย ๆ จนมีค่ามากกว่า 2 V เอ้าต์พุทจะมีค่าสูง หลังจากนั้นถ้าแรงดันมีค่าลดลงเรื่อย ๆ จนต่ำกว่า 2 V เอ้าต์พุทก็ยังค่าสูงอยู่ แรงดันอินพุทต้องลดลงจนกระทั่งต่ำกว่า 0.8 V เอ้าต์พุทจึงกลับมามีค่าต่ำ ลักษณะเช่นนี้เป็นลักษณะของ ฮิสเตอริซิส

แต่อย่างไรก็ตามตัวเกต Driver/Receiver ที่ใช้กับ GPIB และมีจำหน่ายอยู่นั้นอาจมีคุณสมบัติแตกต่างไปจากที่กล่าวมาก็ได้เช่น กรณีที่ใช้เป็นแบบอุปกรณ์แยกชิ้น (Discrete เช่นทรานซิสเตอร์ทำหน้าที่เป็นสวิตช์) แล้วต่อปลาย (โหลด) ด้วยตัวต้านทานเมื่อปิดแหล่งจ่ายไฟจะทำให้แหล่งจ่ายไฟมีค่าเป็น 0 V (ลัดวงจร) แต่ที่สัญญาณบัสข้อมูลยังมีตัวต้านทาน 6.2 kOhm และ 3 kOhm ต่อขนานกันอยู่ ดังนั้นจึงทำให้มีค่าโหลดประมาณ 2 kOhm ต่อสัญญาณบัสข้อมูลกับกราวน์ซึ่งจะเป็นผลเสียทางช่วงของสัญญาณรบกวน (Noise margin)

กรณีที่ใช้เกต Driver/Receiver เฉพาะของ GPIB เช่น ICเบอร์ SN75160,MC6488A เป็นต้น ซึ่งเป็นโหลดประเภทแอคทีฟ เมื่อปิดแหล่งจ่ายไฟจะทำให้ไม่มีโหลดเข้าไปเกี่ยวพันด้วยทำให้เกิดผลดีบางด้าน Noise margin คือข้อมูลไม่ผิดพลาดได้ง่าย

ความหมายของสัญญาณต่าง ๆ ในบัส

GPIB เป็นระบบบัสแบบขนาน ดังนั้นอุปกรณ์ทุกตัวที่ต่อร่วมบัสกันอยู่จึงต่อขนานกันหมด สัญญาณต่าง ๆ จากระบบบัสจึงปรากฏต่ออุปกรณ์ทุกตัวในจำนวนสายต่อทั้ง 24 เส้นของบัสสามารถแบ่งออกได้เป็น 3 กลุ่มสัญญาณคือ

กลุ่มสัญญาณควบคุมการรับส่งข้อมูล ประกอบด้วย

* **DAV** (Data Valid) เมื่อถูกดึงให้มีระดับแรงดันเป็น 'LOW' โดยอุปกรณ์ที่ทำหน้าที่เป็นตัวส่ง (talker) เป็นการแจ้งแก่ระบบบัสว่าตอนนี้ตัวส่งได้ทำการส่งข้อมูลลงไปที่สายสัญญาณข้อมูลเรียบร้อยแล้ว
* **NRFD** (Not Ready For Data) เมื่อถูกดึงให้มีระดับแรงดันเป็น 'LOW' เป็นการแสดงว่า ตอนนี้ระบบบัสยังไม่พร้อมที่จะรับข้อมูล เนื่องจากอุปกรณ์ในระบบยังพร้อมไม่หมดทุกตัว ซึ่งสัญญาณเส้นนี้จะไม่เป็น HIGH จนกว่าอุปกรณ์ทุกตัวให้ระดับแรงดัน 'HIGH' ครบถ้วนแล้ว สัญญาณนี้ใช้ประโยชน์สำหรับกรณีที่อุปกรณ์ที่ใช้ร่วมกันมีความเร็วในการทำงานต่างกัน
* **NDAC** (Not Data Accepted) สัญญาณเส้นนี้ควบคุมโดยอุปกรณ์ตัวรับ (Listener) จะให้ระดับแรงดันเป็น 'LOW' ในขณะที่ตัวรับกำลังเก็บข้อมูลจากสายข้อมูล และเป็น 'HIGH' เมื่ออ่านข้อมูลเรียบร้อยแล้ว

กลุ่มสัญญาณควบคุมการอินเทอร์เฟส ประกอบด้วย

* **ATN** (Attention) เป็นสัญญาณจากอุปกรณ์ที่เป็นตัวควบคุม (controller) ใช้ในการสั่งให้อุปกรณ์ทุกตัวในระบบเตรียมพร้อม เพื่อรอรับคำสั่งต่อไป
* **IFC** (Interface Clear) เป็นสัญญาณรีเซตหรือเคลียร์ระบบ ซึ่งกำเนิดโดยตัวควบคุมเท่านั้น เมื่ออุปกรณ์ในบัส

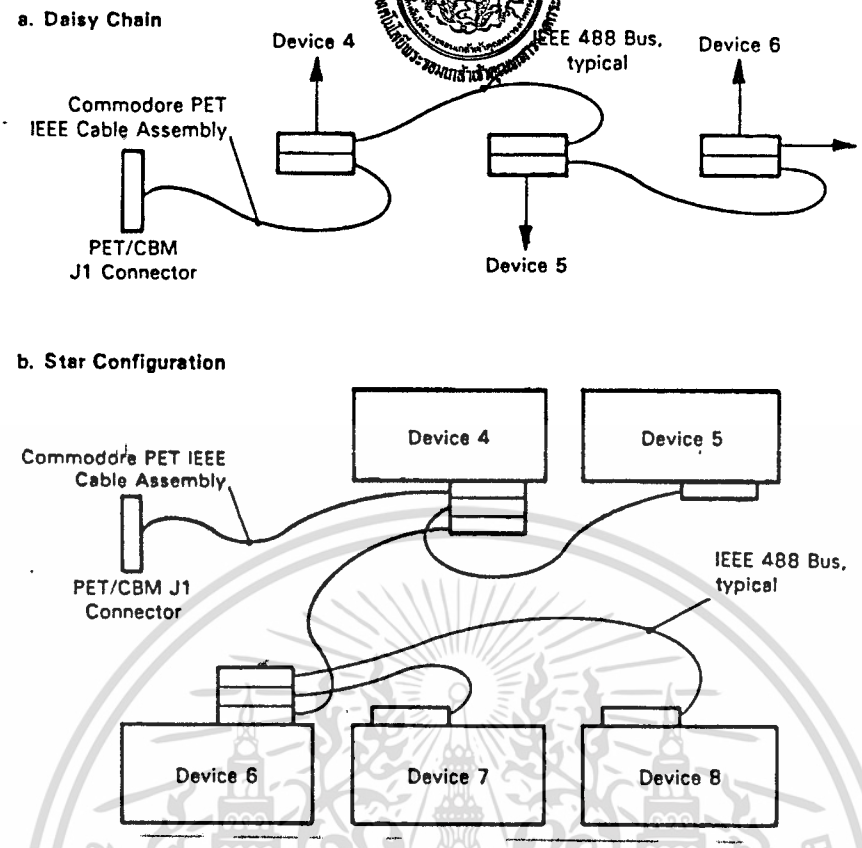
ได้รับสัญญาณเคลียร์นี้จะกลับคืนสู่สภาวะเริ่มต้นใหม่เป็นสภาวะแรกก่อนการกำหนดฟังก์ชั่น เหมือนเมื่อแรกเปิดสวิตช์

* **REN** (Remote Enable) สัญญาณเส้นนี้ควบคุมโดยอุปกรณ์ตัวควบคุมตัวเดียวเท่านั้นเช่นกัน ใช้สั่งให้อุปกรณ์เปลี่ยนจากโหมดที่ใช้งานปกติด้วยมือมาเป็นการควบคุมโดยตัวควบคุมแทน
* **SRQ** (Service Request) เป็นสายสัญญาณอินเทอร์รัพต์ เพื่อเป็นการบอกแก่ระบบว่าขณะ นี้อุปกรณ์ต้องการการติดต่อจากตัวควบคุม ยกตัวอย่างเช่น โวลต์มิเตอร์ที่มีค่าที่อ่านได้พร้อม แล้วที่จะส่งให้แก่ระบบเพื่อไปยังอุปกรณ์ที่ต้องการข้อมูลนี้ หรือเมื่อเกิดข้อผิดพลาดบางอย่าง
* **EOI** (End Or Identify) สัญญาณที่ควบคุมได้ทั้งโดยอุปกรณ์ที่เป็นตัวควบคุมหรือตัวส่งก็ได้ ใช้แสดงว่าข่าวสารข้อมูลที่ส่งเป็นชุดนั้นสิ้นสุดลงแล้ว

กลุ่มสัญญาณข้อมูล ประกอบด้วยสายสัญญาณจำนวน 8 เส้น สำหรับเป็นทางผ่านของข่าวสารข้อมูลของระบบ

สัญญาณลอจิกที่ใช้ใน GPIB บัสนี้มีลักษณะเป็นคอมพลีเมนต์ทั้งหมด คือ '1' เท่ากับ 'LOW' และ '0' เท่ากับ 'HI' ซึ่งตรงกันข้ามกับที่เราคุ้นเคย

สัญญาณควบคุมที่มีอยู่อาจไม่ได้ใช้ทั้งหมดพร้อมกัน บางสายอาจไม่ได้ใช้งานในอุปกรณ์บางเครื่องก็ได้ แล้วแต่ความจำเป็นในการติดต่อ สำหรับสายที่เหลืออีก 7 เส้นนั้น เป็นสายกราวน์ สำหรับการต่ออุปกรณ์ต่าง ๆ ในระบบ GPIB อินเทอร์เฟสบัส มีอยู่ 2 วิธี คือ แบบต่อเนื่องกันไปเรื่อย ๆ จากเครื่องหนึ่งไปยังอีกเครื่องหนึ่งและอีกวิธี คือ การต่อแบบกระจายดังแสดงในรูปที่ 2.5

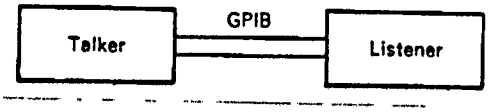


รูปที่ 2.5 ลักษณะการต่อเข้ากับระบบบัส GPIB ทั้ง 2 วิธี

กระบวนการ HANDSHAKE

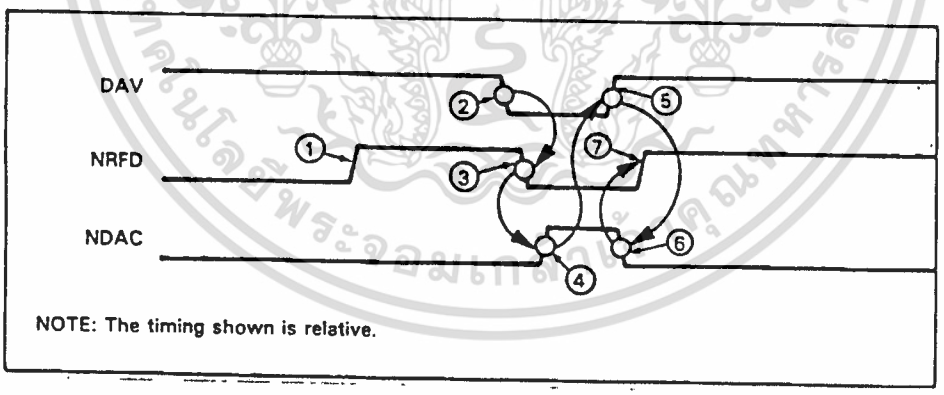
ในการติดต่อสื่อสารระหว่างอุปกรณ์ต่าง ๆ ภายในระบบบัสของ GPIB นั้น จะเป็นแบบอะซิงโครนัส ซึ่งเมื่อจะมีการรับส่งข้อมูลกันระหว่างตัวส่งและตัวรับ ฝ่ายส่งก็จะต้องทำให้ฝ่ายรับทราบว่าขณะนี้ข้อมูลพร้อมอยู่บนบัสแล้ว ให้มาอ่านเอาไปได้ ส่วนฝ่ายรับเมื่อทราบแล้ว ก็จะมาอ่านข้อมูลไป หลังจากนั้นก็จะต้องบอกให้ฝ่ายส่งทราบว่า ได้รับข้อมูลเรียบร้อยแล้ว เพื่อที่ว่าฝ่ายส่งจะส่งข้อมูลตัวใหม่หรือหยุดส่งได้ กระบวนการเหล่านี้จะเกิดขึ้นทุกครั้งเมื่อมีการรับส่งข้อมูล (ซึ่งอาจจะเป็นข้อมูลที่ได้จากการวัด หรือคำสั่งต่าง ๆ ที่ใช้ในการควบคุมอุปกรณ์ภายในระบบ) เราเรียกกระบวนการเหล่านี้ว่า "HANDSHAKE"

สำหรับระบบทั่ว ๆ ไปเราสามารถจำลองให้อยู่ในรูปอย่างง่ายได้คือ มีตัวส่ง 1 ตัวและตัวรับ 1 ตัว ดังในรูปที่ 2.6



รูปที่ 2.6 ระบบการอินเทอร์เฟสอย่างง่าย

ในระบบ GPIB จะมีสายสัญญาณที่ใช้ในกระบวนการ HANDSHAKE อยู่ 3 สัญญาณ คือ DAV (Data Valid) ,NRFD (Not Ready For Data) และ NDAC Not Data Accepted) สัญญาณ DAV ถูกควบคุมโดยตัวส่ง ใช้สำหรับแจ้งให้ตัวรับทราบว่าขณะนี้มีข้อมูลที่จะส่งให้ตัวรับอยู่บนบัสเรียบร้อยแล้ว ส่วนสัญญาณ NRFD และ NDAC ถูกควบคุมโดยตัวรับ ใช้สำหรับแจ้งให้ตัวส่งทราบว่า ขณะนี้ตัวรับพร้อมที่จะรับข้อมูลหรือได้รับข้อมูลเรียบร้อยแล้วหรือยัง ตามลำดับ โดยมีไดอะแกรมเวลาของกระบวนการ HANDSHAKE ดังแสดงในรูปที่ 2.7 ทั้งนี้จะขออธิบายกระบวนการไปตามลำดับหมายเลขในไดอะแกรมเวลาดังนี้ (ทั้งนี้เราจะสมมุติว่าได้มีการระบุอุปกรณ์ที่จะทำหน้าที่เป็นตัวรับและตัวส่งเรียบร้อยแล้ว)



รูปที่ 2.7 ไดอะแกรมเวลาของกระบวนการ HANDSHAKE

1 เมื่อตัวรับพร้อมที่จะรับข้อมูลจากตัวส่ง ตัวรับก็จะทำการแอคทีฟสัญญาณ NRFD ให้มีระดับแรงดันเป็น High เพื่อบอกให้ตัวส่ง ส่งข้อมูลลงไปบนบัสข้อมูลได้เลย

2 เมื่อตัวส่งจับสัญญาณ NRFD ได้ว่ามีระดับแรงดันเป็น High ซึ่งแสดงว่าตัวรับพร้อมที่จะรับข้อมูลแล้วนั้น ตัวส่งก็จะส่งข้อมูลลงไป หลังจากนั้นก็จะดึงสัญญาณ DAV ให้มีระดับแรงดันเป็น Low เพื่อบอกให้ตัวรับทราบ ว่าขณะนี้ข้อมูลได้อยู่บนบัสเรียบร้อยแล้ว ให้มาอ่านไปได้

3. เมื่อตัวรับทราบว่าข้อมูลที่ต้องการอยู่บนบัสแล้ว ตัวรับจะตอบรับให้ตัวส่งทราบโดยการดึงสัญญาณ NRFD ให้เป็น Low แล้วก็ทำการอ่านข้อมูลเข้าไป

4 หลังจากที่ตัวรับอ่านข้อมูลเสร็จแล้ว ก็จะบอกให้ตัวส่งทราบโดยการดึงสัญญาณ NDAC ให้มีระดับแรงดันเป็น High

5 เมื่อฝ่ายส่งจับได้ว่าสัญญาณ NDAC เป็น High อันแสดงว่าตัวรับอ่านข้อมูลที่ส่งไปเรียบร้อยแล้ว ตัวส่งก็จะแอคทีฟสัญญาณ DAV ให้เป็น High อีกครั้งหนึ่ง ซึ่งเป็นการบอกให้ฝ่ายรับทราบว่าข้อมูลตัวที่เพิ่งอ่านไปจะถูก Clear ออกไปจากบัสแล้ว

6. ตัวรับก็จะตอบรับจากขั้นตอนที่ 5 โดยการดึงสัญญาณ NDAC ให้กลับมาเป็น Low อีกครั้ง

7 หลังจากนั้นตัวรับก็จะดึงสัญญาณ NRFD ให้เป็น High อีก เพื่อบอกให้ตัวส่งทราบว่าพร้อมที่จะรับข้อมูลตัวต่อไปแล้ว

กระบวนการนี้ก็จะเสร็จสมบูรณ์ ตัวรับก็จะรอรับข้อมูลตัวต่อไป หากตัวส่งมีข้อมูลที่ต้องการส่งอีก กระบวนการนี้ก็จะเริ่มใหม่ตั้งแต่ขั้นตอนที่ 2 ลงมา จนกว่าข้อมูลจะรับส่งกันจนหมดกระบวนการ HANDSHAKE นี้จำเป็น สำหรับการติดต่อสื่อสารทุกครั้งที่ต้องการรับส่งข้อมูลกัน และยังก่อให้เกิดผลดีในแง่ที่ตัวรับและตัวส่งมีความเร็วในการทำงานที่ต่างกันมาก ๆ โดยฝ่ายที่ทำงานช้าก็จะมีเวลาเพียงพอที่จะทำงานให้เสร็จ โดยการให้ฝ่ายที่เร็วกว่ารอ

ตารางที่ 2.1 เปรียบเทียบการจัดตำแหน่งสัญญาณระหว่าง IEEE Std 488-1978 กับ IEC625-1

IEEE Standard		IEC Standard	
Pin	Designation	Pin	Designation
1	DIO1	1	DIO1
2	DIO2	2	DIO2
3	DIO3	3	DIO3
4	DIO4	4	DIO4
5	EOI	5	REN
6	DAV	6	EOI
7	NRFD	7	DAV
8	NDAC	8	NRFD
9	IFC	9	NDAC
10	SRQ	10	IFC
11	ATN	11	SRQ
12	SHIELD	12	ATN
13	DIO5	13	SHIELD
14	DIO6	14	DIO5
15	DIO7	15	DIO6
16	DIO8	16	DIO7
17	REN	17	DIO8
18	GND 6	18	GND 5
19	GND 7	19	GND 6
20	GND 8	20	GND 7
21	GND 9 (Ground pins for lines 6-11 inclusive)	21	GND 8 (Ground pins for lines 5-12 inclusive)
22	GND 10	22	GND 9
23	GND 11	23	GND 10
24	LOGIC GND	24	GND 11
		25	GND 12

คำสั่งใช้งานของ GPIB

เมื่อมีการเชื่อมต่อหรืออินเทอร์เฟสอุปกรณ์เครื่องมือต่าง ๆ เข้าด้วยกัน ด้วย GPIB บัสแล้ว ตัวอย่างเช่น อาจจะเป็นการต่อคอมพิวเตอร์กับเครื่องมือวัดความถี่,ปริ้นเตอร์และดิสไดรฟ์ ตัวคอมพิวเตอร์ก็จะทำหน้าที่เป็นตัวควบคุม เพื่อบริหารอุปกรณ์อื่น ๆ และเป็นตัวรับเพื่อรับข้อมูลจากเครื่องวัดความถี่ ขณะเดียวกันก็อาจจะเป็นตัวส่งด้วย โดยส่งข้อมูลไปยังปริ้นเตอร์และดิสไดรฟ์ ส่วนเครื่องวัดก็จะเป็นตัวส่งค่าความถี่ที่วัดได้ให้แก่ตัวอื่น และปริ้นเตอร์เป็นตัวรับข้อมูลมาพิมพ์ ส่วนดิสไดรฟ์ก็เป็นได้ทั้งตัวรับและตัวส่งข้อมูล

จะเห็นว่าอุปกรณ์แต่ละตัวสามารถเป็นได้หลายอย่าง ขึ้นอยู่กับว่าในการทำงานขณะหนึ่ง ๆต้องการให้เป็นอะไร ซึ่งอุปกรณ์ทุกตัวที่ต่ออยู่ในบัส จะมีหมายเลขรหัสประจำตัวสำหรับการอ้างถึง (เหมือนกับบ้านที่ต้องมีเลขที่) เพื่อให้ตัวควบคุมสั่งการบังคับหน้าที่ต่าง ๆ ได้และใช้ในการอ้างอิงจุดหมายปลายทางของข้อมูล

การสั่งการต่าง ๆ เพื่อกำหนดหน้าที่การทำงานและกำหนดฟังก์ชั่น เช่น กำหนดช่วงการวัด,โหมดการวัด แก่เครื่องมือวัดที่ต่ออยู่เหล่านี้นั้น ตัวควบคุมจะเป็นตัวกำหนดโดยการส่งรหัสคำสั่งไปที่อุปกรณ์ผ่านสายสัญญาณข้อมูล DI_1-DI_8 รหัสคำสั่งนี้จะถูกส่งไปในช่วงที่สายสัญญาณ ATN มีระดับแรงดันเป็น 'LOW' (ในระบบง่าย ๆ ที่ไม่ซับซ้อนซึ่งไม่จำเป็นต้องมีตัวควบคุม อุปกรณ์จะถูกกำหนดไว้ตายตัวเช่นเป็นตัวส่งได้อย่างเดียว (Talk only) หรือรับอย่างเดียว (Listen only)

คำสั่งสำหรับกำหนดหน้าที่การทำงานแบบต่าง ๆ ตามมาตรฐานของ GPIB มีอยู่ด้วยกัน 128 คำสั่ง ดังแสดงในตารางที่ 2.2 โดยแบ่งได้เป็น 5 กลุ่มคำสั่ง

ตารางที่ 2.2 รหัสข้อมูลข่าวสารของระบบ GPIB บัส

ASCII — IEEE 488 BUS MESSAGES (COMMANDS AND ADDRESSES) HEX CODES

MSD / LSD	0 ASCII	0 MSG	1 ASCII	1 MSG	2 ASCII	2 MSG¹	3 ASCII	3 MSG¹	4 ASCII	4 MSG¹	5 ASCII	5 MSG¹	6 ASCII	6 MSG	7 ASCII	7 MSG
0	NUL		DLE		SP	00	0	16	@	00	P	16	`		p	
1	SOH	GTL	DC1	LLO	!	01	1	17	A	01	Q	17	a		q	
2	STX		DC2		"	02	2	18	B	02	R	18	b		r	
3	ETX		DC3		#	03	3	19	C	03	S	19	c		s	
4	EOT	SDC	DC4	DCL	$	04	4	20	D	04	T	20	d		t	
5	ENQ	PPC	NAK	PPU	%	05	5	21	E	05	U	21	e		u	
6	ACK		SYN		&	06	6	22	F	06	V	22	f		v	
7	BEL		ETB		'	07	7	23	G	07	W	23	g		w	
8	BS	GET	CAN	SPE	(	08	8	24	H	08	X	24	h		x	
9	HT	TCT	EM	SPD	)	09	9	25	I	09	Y	25	i		y	
A	LF		SUB		*	10		26	J	10	Z	26	j		z	
B	VT		ESC		+	11		27	K	11	[	27	k		{	
C	FF		FS		,	12	<	28	L	12	\	28	l		|	
D	CR		GS		-	13	=	29	M	13	]	29	m		}	
E	SO		RS		.	14	>	30	N	14	^	30	n		~	
F	SI		US		/	15	?	UNL	O	15	_	UNT	o		DEL	

MEANING DEFINED BY PCG CODE

MEANING DEFINED BY PCG CODE

ADDRESSED COMMAND GROUP | UNIVERSAL COMMAND GROUP | LISTEN ADDRESS GROUP | TALK ADDRESS GROUP | SECONDARY COMMAND GROUP

PRIMARY COMMAND GROUP (PCG)

รหัสที่ใช้ในระบบGPIBบัสนั้น ใช้ร่วมกันทั้งรหัสข้อมูลและรหัสคำสั่ง นั่นคือรหัสเดียวกันมีความหมายได้ 2 อย่าง คือเมื่อสายสัญญาณ ATN เป็น 'LOW' จะหมายถึงรหัสคำสั่ง แต่ถ้า ATNเป็น'HI' รหัสนี้จะแทนข้อมูลที่เป็น ASCII แทน ซึ่งในตารางที่ 2.2 ได้แบ่งความหมายออกเป็น 2 คอลัมน์ให้เห็น รหัสคำสั่งมีดังนี้

1.**กลุ่มคำสั่งเจาะจงจุดหมาย**(AddressedCommandGroup) เป็นคำสั่งที่ส่งไปยังอุปกรณ์ที่เป็นตัวส่งหรือตัวรับที่กำหนดไว้ล่วงหน้าแล้ว ให้มีสภาพการทำงานตามที่ต้องการคำสั่งในกลุ่มนี้ประกอบด้วย

GTL (Go To Local)	สั่งให้อุปกรณ์กลับสู่สภาพการควบคุมปกติด้วยมือ
SDC (Selected Device Clear)	สั่งให้อุปกรณ์เคลียร์ตัวเองสู่สภาพเริ่มต้นใหม่
PPC (ParallelPollConfigure)	เป็นคำสั่งสำหรับทำการจัดสรรสายสัญญาณ ของการทำกระบวนการตรวจสอบสภาพอุปกรณ์โดยวิธีขนานคือ parallelpoll (ซึ่งจะอธิบายในภายหลัง) โดยใช้งานร่วมกับกลุ่มคำสั่งรอง
GET (Group Execute Trigger)	ใช้สั่งเริ่มต้นการทำงานของอุปกรณ์ทีละหลาย ๆ ตัว
TCT (Take Control)	เป็นการกำหนดให้อุปกรณ์ตัวส่งทำหน้าที่เป็นตัวควบคุม

2.**กลุ่มคำสั่งครอบคลุม** (Universal Command Group) เป็นคำสั่งที่ส่งไปยังอุปกรณ์ทุกตัวที่ต่ออยู่ในบัส ประกอบด้วย

LLO(Local Lockout)	เป็นการสั่งให้อุปกรณ์ล็อกอยู่ที่สภาวะควบคุม โดยจะไม่สามารถใช้ปุ่มปรับที่หน้าปัดได้
DCL (Device Clear)	สั่งให้อุปกรณ์ทุกตัวกลับไปสู่สภาวะเริ่มต้น
PPU(Parallel Poll Unconfigure)	ใช้ยกเลิกกระบวนการตรวจสอบสภาพแบบขนานทั้งหมด
SPE (SerialPoll Enable)	เปลี่ยนโหมดของการตรวจสอบสภาพเป็นแบบอนุกรม ในโหมดนี้จะเป็นการส่งสถานะของเครื่องแทนการส่งข้อมูล
SPD (Serial Poll Disable)	ยกเลิกโหมดการตรวจสอบแบบอนุกรม

3.**กลุ่มคำสั่งกำหนดอุปกรณ์ตัวรับ**(Listen Address Group) สำหรับกำหนดให้อุปกรณ์เป็นตัวรับตามรหัสหมายเลขจาก 0 ถึง 30 และมีคำสั่ง UNL (Unlisten) สำหรับยกเลิก

4. **กลุ่มคำสั่งกำหนดอุปกรณ์ตัวส่ง**(Talker Address Group) สำหรับกำหนดให้อุปกรณ์เป็นตัวส่งตามรหัสหมายเลขจาก 0 ถึง 30 และมีคำสั่ง UNT (Untalk) สำหรับยกเลิกเช่นกัน

คำสั่งในกลุ่มที่1ถึง 4 นั้นจัดเป็นกลุ่มคำสั่งหลัก (Primary Command Group) ที่มีความหมายตายตัว ยังมีคำสั่ง อีกกลุ่มหนึ่งที่ขึ้นอยู่กับการกำหนดภายหลังนั่นคือกลุ่มคำสั่งรอง

5.**กลุ่มคำสั่งรอง** (Secondary Command Group) เป็นคำสั่งสำหรับใช้กำหนดรายละเอียดย่อยของอุปกรณ์แต่ละตัวที่ต่ออยู่ในระบบ ให้มีการทำงานอย่างไรตามจุดประสงค์ใช้งานของเครื่องมือนั้นเช่นเดียวกับการปรับปุ่มปรับต่าง ๆ ด้วยมือนั่นเอง คำสั่งรองนี้จะตามหลังคำสั่งหลัก คือจะใช้หลังจากอุปกรณ์ต่าง ๆ ถูกกำหนดวางตัวในระบบเรียบร้อยแล้ว

คำสั่งต่าง ๆ ที่กล่าวไป ซึ่งใช้ในการกำหนดสภาวะการทำงานของอุปกรณ์ แต่ละสภาวะที่กำหนดไปนั้นเป็นอย่างไร และมีจุดประสงค์เพื่ออะไร มาดูกันต่อไป

Device Clear/Interface Clear

DeviceClear ใช้ในการทำให้อุปกรณ์ที่ต่ออยู่ในบัสกลับไปอยู่ในสภาวะเหมือนเมื่อแรกเริ่มที่เปิดไฟเข้าเครื่องอันเป็นสภาวะเริ่มต้น ยังไม่มีการกำหนดฟังก์ชั่นใด ๆ สภาวะเริ่มต้นนี้จะแตกต่างกันไปแล้วแต่ว่าอุปกรณ์นั้นถูกออกแบบมาไว้อย่างไร DeviceClear มีอยู่ด้วยกัน 2 ลักษณะคือ เคลียร์หมดทุกตัวที่ต่ออยู่ (DCL) กับเคลียร์เจาะจงอุปกรณ์ตัวใดตัวหนึ่ง (SDC) แต่ว่าการเคลียร์อุปกรณ์ให้อยู่ในสภาวะเริ่มต้นนั้น ไม่ได้หมายความว่าอินเทอร์เฟสฟังก์ชั่นของ GPIB จะถูกเคลียร์ให้ไปอยู่ในสภาวะเริ่มต้นด้วยแต่อย่างใด เป็นการเคลียร์เฉพาะตัวอุปกรณ์เท่านั้นอินเทอร์เฟสฟังก์ชั่นก็คือสภาพการอินเทอร์เฟสที่ได้กำหนดเอาไว้ในระบบประกอบด้วยฟังก์ชั่นต่าง ๆ ดังแสดงในตารางที่ 2.3

ตารางที่ 2.3 ฟังก์ชั่นการอินเทอร์เฟสในระบบ GPIB บัส

ฟังก์ชัน	สัญลักษณ์	การกลับสู่สภาพเริ่มต้นโดย IFC
source hand shake	SH	✓
acceptor hand shake	AH	✓
talker หรือ enlarge talker	T หรือ TE	✓
listener หรือ enlarge listener	L หรือ LT	✓
service request	SR	—
remote/local	RL	—
parallel poll	PP	—
device clear	DC	✓
device trigger	DT	✓
controller	C	✓

การเคลียร์สภาพการอินเทอร์เฟสให้อยู่ในสภาวะเริ่มต้น ต้องใช้คำสั่ง Interface clear แทนซึ่งจะทำให้ทุกฟังก์ชั่นถูกยกเลิกไปยกเว้น SR (Service Request),RL (Remote/Local) และ PP (Parallel Poll) นั่นก็คือถึงจะยกเลิกสภาพการอินเทอร์ เฟส แต่จะยังคงรักษาลักษณะการจัดของ SR, RL และ PP เอาไว้

Remote/Local

รีโมต (Remote) เป็นการกำหนดให้อุปกรณ์ที่ต่ออยู่ในระบบ เช่น เครื่องมือวัด ให้อยู่ในการควบคุมของอุปกรณ์ตัวอื่นแทนซึ่งปุ่มปรับต่าง ๆ บนหน้าปัดเครื่องจะไม่มีผลต่อการทำงาน ส่วนโลคัล (Local) เป็นการควบคุมเครื่องมือวัดด้วยปุ่มปรับบนหน้าปัดตามปกติ หรือควบคุมด้วยมือนั่นเอง

การใช้รีโมตมีประโยชน์ในแง่ที่ขณะที่ตัวควบคุม เช่นคอมพิวเตอร์ กำลังติดต่อกับอุปกรณ์ตัวนั้นอยู่ หากไม่มีการตัดการควบคุมโดยปุ่มปรับบนหน้าปัดออก ถ้ามีใครมาปรับแต่งก็จะทำให้การทำงานผิดพลาดไปได้ในทางกลับกันขณะที่ใช้งานปกติในโหมดของโลคัล ก็จะไม่ให้คอมพิวเตอร์มายุ่งเกี่ยวได้ การทำงานของ GPIB ในรีโมตและโลคัล มี 4 ลักษณะดังนี้

1 LOCS ก็คือโลคัลนั่นเอง อันเป็นสภาพการควบคุมที่ปุ่มปรับตามปกติซึ่งอุปกรณ์จะอยู่ในสภาพนี้เมื่อตอนเปิดไฟเข้าเครื่อง หรือ REN (Remote Enable) มีระดับแรงดันเป็น 'HIGH' หรือเมื่อได้รับคำสั่ง GTL (Go To Local)

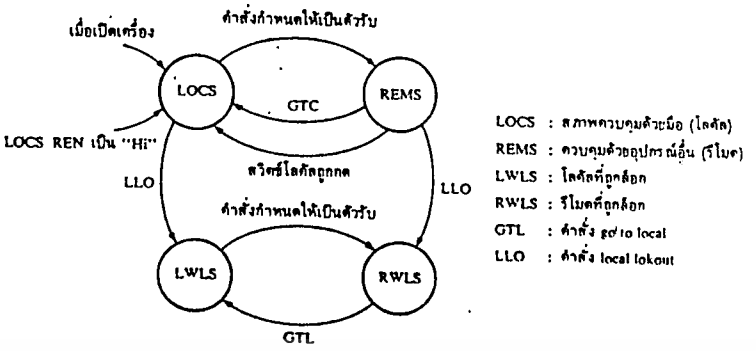
2. REMS คือรีโมต หมายถึงการตัดการควบคุมโดยปุ่มบนหน้าปัดออก กลายเป็นการควบคุมการทำงานโดยอุปกรณ์อื่นแทนสภาพรีโมตจะเกิดขึ้นเมื่อRENมีระดับแรงดันเป็น 'LOW' และจะถูกล็อกเอาไว้ เว้นแต่ว่าสวิตช์โลคัลที่ตัวอุปกรณ์จะถูกเปลี่ยนไปตำแหน่ง Local

3. RWLS เป็นสภาพรีโมตที่ถูกล็อกเอาไว้เช่นกัน แต่ว่าจะตัดการควบคุมของสวิตช์โลคัลที่ตัวอุปกรณ์ออกไป ฉะนั้นสภาพรีโมตโดย RWLS จึงมีนัยสำคัญสูงกว่า REMS อย่างไรก็ตามก็ยังถูกยกเลิกได้ด้วยคำสั่ง LLO (Local Lockout)

4. LWLS มีสภาพเช่นเดียวกันโลคัลแต่จะแตกต่างกันตรงที่สภาพโลคัลโดย LWLS นี้ เมื่อได้รับคำสั่งกำหนดอุปกรณ์ตัวรับ จะเปลี่ยนไปอยู่ในสภาพรีโมตแบบล็อคหรือ RWLS ทันที ในการจะมาที่สภาพ LWLS นี้ได้ ก็มี 2 กรณีคือ เมื่ออยู่ในสภาพโลคัลธรรมดา (LOCS) แล้วได้รับคำสั่ง LLO (Local Lockout) หรือเมื่ออยู่ใน RWLS แล้วได้รับคำสั่ง GTL (Go To local)

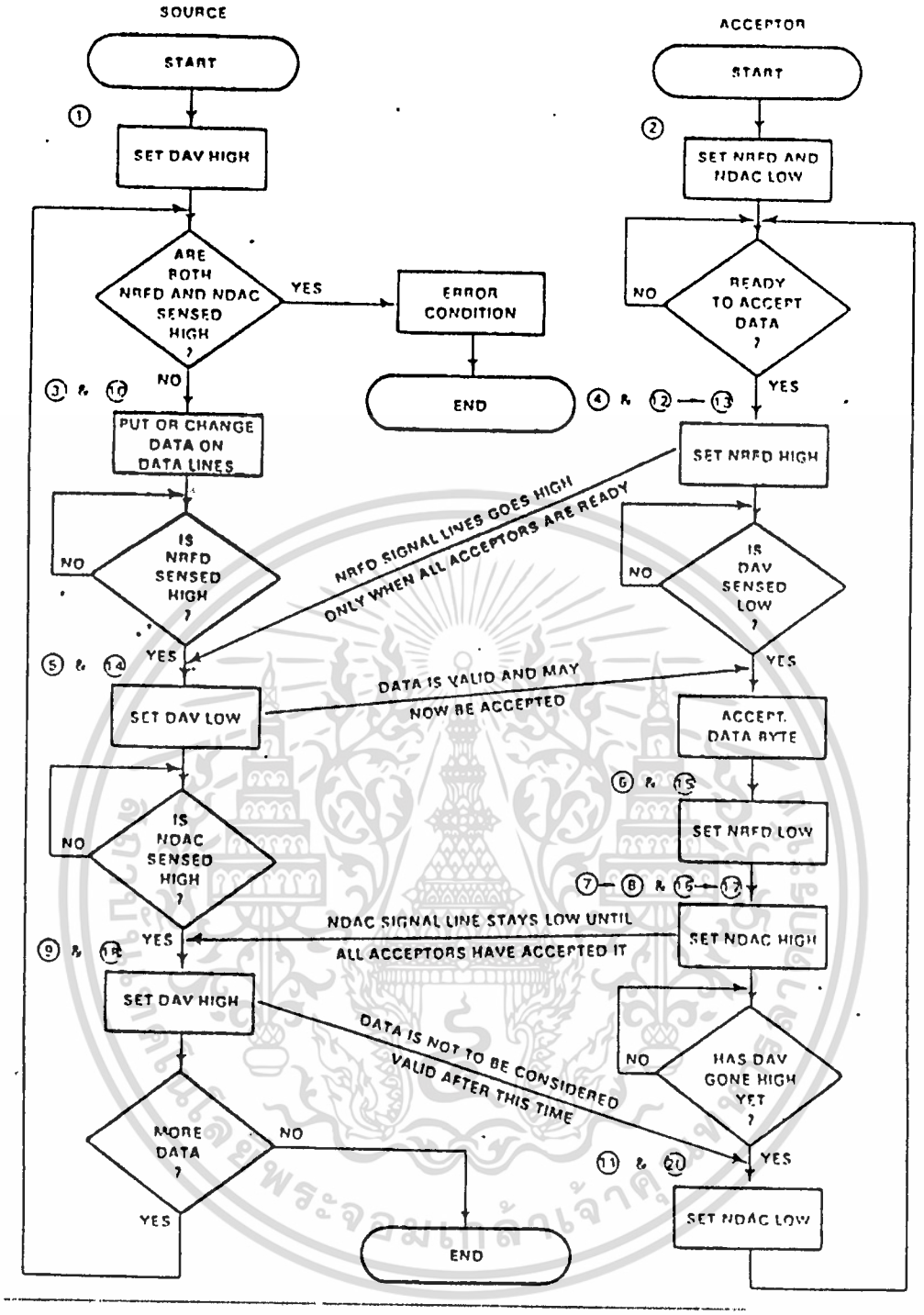
ในรูปที่ 2.8 เป็นการแสดงการเปลี่ยนสภาวะต่าง ๆ ของโลคัลและรีโมตทั้ง 4 ลักษณะตามที่กล่าวมา ซึ่งเมื่อ REN มีระดับแรงดัน 'HIGH' ก็จะกลายเป็นโลคัลทันที ไม่ว่าเดิมจะมีสภาพใด และเมื่อ RENเป็น'LOW'แล้ว หากไม่มีคำสั่งกำหนดอุปกรณ์ตัวรับหรือคำสั่ง LLO เข้ามาก็จะอยู่ในสภาพโลคัลเช่นนั้น

เมื่อเครื่องถูกกำหนดให้อยู่ในสภาพโลคัลหรือควบคุมด้วยมือแล้ว หากคอมพิวเตอร์ส่งคำสั่งเข้ามาเพื่อกำหนดให้เป็นอุปกรณ์ตัวรับเครื่องจะไม่สนใจคำสั่งดังกล่าวและหากถูกสั่งให้เป็นอุปกรณ์ตัวส่งเครื่องก็จะส่งข้อมูลอย่างใดอย่างหนึ่งออกไปให้ นั่นคือระบบจะให้ความสำคัญต่อสภาพโลคัลมากกว่า



รูปที่ 2.8 สภาพโลคัลและรีโมตลักษณะต่าง ๆ และแผนภูมิการเปลี่ยนสภาวะ

ในการรับส่งข้อมูลกันในระบบ GPIBจะมี Algorithm ดังแสดงเป็น Flow chart ในรูปที่ 2.9



รูปที่ 2.9 Flow chart แสดงขั้นตอนการรับส่งข้อมูลในระบบ GPIB

บทที่ 3

การ์ดอินเทอร์เฟส

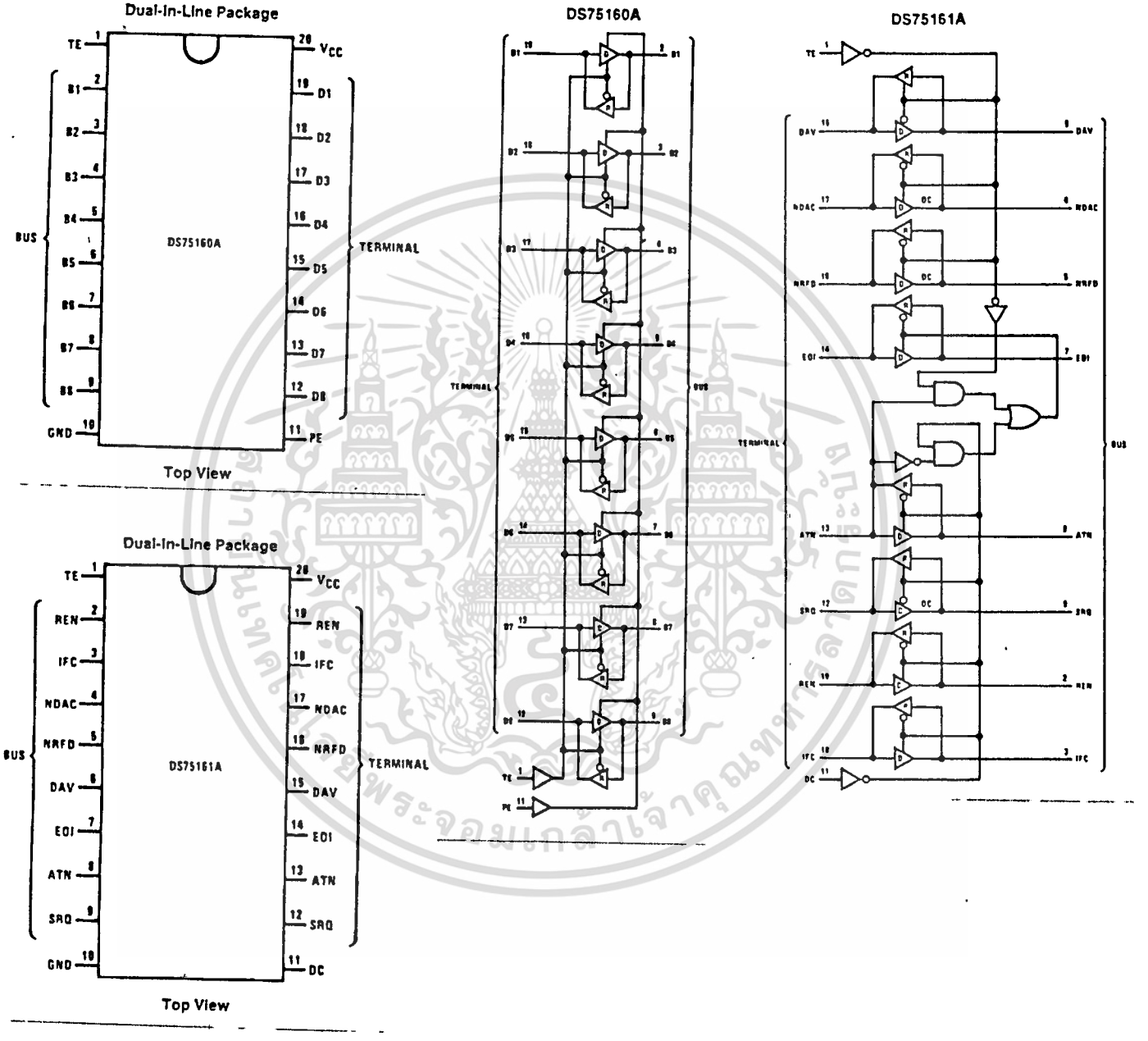
การ์ดอินเทอร์เฟส GPIB นี้ เป็นการ์ดที่ใช้เสียบลงในสล๊อตคอมพิวเตอร์ทั่ว ๆ ไป รายละเอียดต่างๆ เกี่ยวกับการอินเทอร์เฟสกับคอมพิวเตอร์ผ่านทางสล๊อตเหล่านี้ สามารถหาอ่านได้จากหนังสือ การอินเทอร์เฟส IBM PC รายการอุปกรณ์ในโครงงานการ์ดอินเทอร์เฟสนี้ประกอบไปด้วย

1. ไอซีเบอร์ 8255 (Programmable Peripheral Interface) 2 ตัว
2. ไอซีเบอร์ 74LS133 (13 Input NAND Gate) 2 ตัว
3. ไอซีเบอร์ 74LS04 (Hex Inverter) 2 ตัว
4. ไอซีเบอร์ SN75160 (Octal General Purpose Interface bus Transceiver) 1 ตัว
5. ไอซีเบอร์ SN75161 (Octal General Purpose Interface bus Transceiver) 1 ตัว
6. ตัวเก็บประจุชนิด Multilayer ขนาด 0.1 uF 8 ตัว
7. ความต้านทานแบบ 9 ขา ขนาด 2k 2 ตัว

การออกแบบวงจรบัฟเฟอร์

ในส่วนของวงจรบัฟเฟอร์ เราได้เลือกไอซีเบอร์SN75160 และ SN75161 ซึ่งเป็นไอซีที่ออกแบบมาให้ใช้กับระบบบัสของ GPIB โดยเฉพาะ ซึ่งมีรายละเอียดดังนี้

Connection Diagram



รูปที่ 3.1 แสดงตำแหน่งขาของ IC 75160 และ 75161

คุณสมบัติของไอซีทั้งสองเป็นดังนี้

SN75160

- เป็นอุปกรณ์ 8 Channel ที่มีการส่งผ่านสองทิศทาง (Bidirectional Transceiver) เป็นทั้ง driver และ receiver ในตัวเดียวกัน โดยอาศัยการควบคุมทิศทางอินพุทคือ สัญญาณ TE
- ต่ออยู่กับระบบบัสทั้ง 8 เส้นของ GPIB คือ DIO_1-DIO_8 ตามมาตรฐานของ IEEE 488
- เมื่อเกิด Power Down (vcc = 0) จะไม่เกิดการ Load
- นอกจากนั้นพอร์ตเอาต์พุทยังสามารถกำหนดโหมดให้ Enable หรือ Disable การต่อใช้งานในแบบ Totempole ได้อีกด้วย
- เมื่อสัญญาณควบคุมอินพุท PE อยู่ในสภาวะ High บัสเอ้าต์พุทจะอยู่ในสภาวะ High speed totempole mode และเมื่อ PE อยู่เป็น Low เอ้าต์พุทจะทำตัวเป็น Open Collector ซึ่งใช้ประโยชน์สำหรับการ Polling แบบขนาน

SN75161

- เป็นอุปกรณ์ที่มีการส่งผ่านได้สองทิศทางและมี 8 Channel เช่นเดียวกันกับ 75160 แต่มีโครงสร้างที่พิเศษเฉพาะ คือใช้ทำหน้าที่จัดการเกี่ยวกับสัญญาณควบคุมในระบบบัสของ IEEE 488
- เป็นอุปกรณ์ที่ออกแบบมาให้ใช้งานร่วมกับ 75160 ประกอบกันเป็นบัส 16 เส้น ต่ออินเทอร์เฟสระหว่าง IEEE 488 บัส กับ Single Controller Instrumentation System
- ในการใช้งานสัญญาณควบคุม SRQ , NDACและ NRFD ในระบบบัสมีเอ้าต์พุทเป็นแบบ Open Collector

การควบคุมทิศทางของสัญญาณควบคุมถูกแบ่งออกเป็น 3 กลุ่มด้วยกันคือ

1. DAV,NDAC และ NRFD เป็นกลุ่มสัญญาณรับส่งข้อมูลซึ่งถูกควบคุมโดย TE อินพุท
2. ATN,SRQ และ IFC เป็นกลุ่มสัญญาณควบคุมการอินเทอร์เฟส ถูกควบคุมโดย DC อินพุท
3. EOI ทิศทางการส่งผ่านข้อมูลจะถูกควบคุมทั้ง TE และ DC

การออกแบบส่วนของพอร์ตควบคุม

เราได้ทราบแล้วว่าแบบมาตรฐานของ IEEE 488 นั้นประกอบด้วยบัสต่าง ๆ จำนวน 16 เส้น ซึ่งสามารถแบ่งได้เป็น 3 กลุ่มด้วยกันคือ

1. กลุ่มสัญญาณควบคุมการรับส่งข้อมูล ประกอบด้วย DAC,NRFD และ NDAC ซึ่งถูกควบคุมโดย TE
2. กลุ่มสัญญาณควบคุมการอินเทอร์เฟสประกอบด้วย ATN,IFC,REN,SRQ ถูกควบคุมโดย DC และ EOI ควบคุมได้ทั้ง DC และ TE
3. กลุ่มสัญญาณข้อมูล เป็นเส้นทางผ่านข้อมูลของระบบ จำนวน 8 เส้น

นอกจากนั้นเรายังต้องกำหนดสัญญาณควบคุมของไอซี 75160 และ 75161 ด้วย สัญญาณนั้นคือสัญญาณ PE,TE และ DC ตามลำดับ ดังนั้นเราจึงต้องกำหนดพอร์ตทั้งสิ้นจำนวน 4 พอร์ตด้วยกัน และพอร์ตนี้จะต้องเป็นได้ทั้งอินพุทและเอ้าต์พุทพอร์ตซึ่งตรงกับคุณสมบัติของไอซี 8255 พอดี ซึ่งเราได้กล่าวในตอนแรกแล้วว่า ภายใน 8255 1 ตัวจะประกอบด้วยพอร์ต 3 พอร์ต อิสระต่อกัน

ดังนั้นจึงขอกำหนดการทำงานของ 8255 ดังต่อไปนี้

1. ให้ 8255 ทำงานในโหมดศูนย์
2. - กำหนดให้พอร์ต A เป็นพอร์ตของบัสข้อมูล 8 บิท โดยกำหนดให้เป็นได้ทั้งอินพุทและเอ้าท์พุท
   - ให้พอร์ต B เป็นพอร์ตที่ใช้สำหรับกำหนดทิศทางของสัญญาณข้อมูลและสัญญาณควบคุม โดยกำหนดให้เป็นเอ้าต์ พุทพอร์ตเท่านั้น
   - ให้พอร์ต C ซึ่งใช้ทั้งพอร์ต C บนและพอร์ต C ล่าง เป็นได้ทั้งอินพุทและเอ้าต์พุทพอร์ต โดยใช้ 8255 2 ตัว เพื่อความสะดวกในการควบคุมการทำงาน

การจัดกลุ่มสัญญาณ

การจัดกลุ่มสัญญาณจะพิจารณาจากทิศทางของสัญญาณที่กำหนดไว้ในไอซีเบอร์ SN75160 และ SN75161 ซึ่งทำหน้าที่เป็นบัฟเฟอร์สำหรับการรับส่งข้อมูลและสัญญาณควบคุมสำหรับระบบ GPIB โดยเฉพาะ และพิจารณาถึงความสะดวกในการควบคุมการทำงานของ 8255

กลุ่มสัญญาณต่าง ๆ

1. DIO_1-DIO_8 ต่อกับพอร์ต A ของ 8255 ตัวที่ 1
2. NDAC,NRFD ต่อกับพอร์ต C บนของ 8255 ตัวที่ 1
3. DAV,EOI ต่อกับพอร์ต C ล่างของ 8255 ตัวที่ 1
4. SRQ ต่อกับพอร์ต C บนของ 8255 ตัวที่ 2
5. REN,IFC,ATN ต่อกับพอร์ต C ล่างของ 8255 ที่ 2

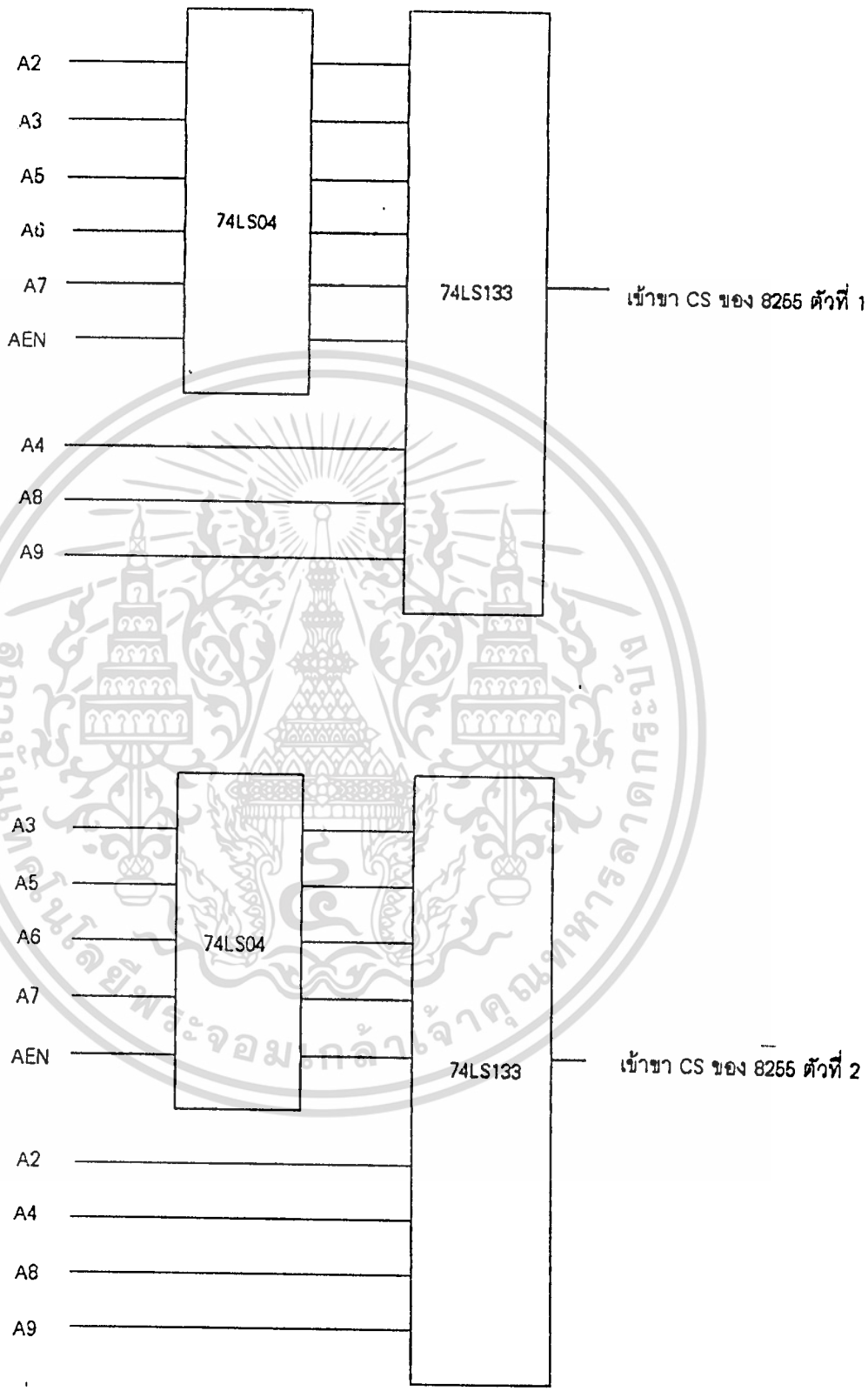
และกลุ่มสัญญาณควบคุมต่าง ๆ บน 8255ซึ่งประกอบไปด้วย RD,WR และ RESET สามารถต่อเข้าโดยตรงกับ IBM Slot ของคอมพิวเตอร์ได้เลย เพื่อใช้ในขบวนการอ่านเขียนและรีเซตตามลำดับ

การออกแบบวงจรถอดรหัส

ในการทำโครงงานนี้ได้เลือกใช้พอร์ตแอดเดรสที่ 310H-317H สำหรับการกำหนดตำแหน่งของไอซี 8255 ทั้งสองตัวโดยเรากำหนดให้ แอดเดรสที่ 0310H-0313H เป็นของไอซี 8255 ตัวที่ 1 และแอดเดรสที่ 0314H-0317H เป็นของไอซี 8255 ตัวที่ 2 ซึ่งในการเลือกอุปกรณ์ Gate มาสร้างวงจร Decoder เพื่อ Decode แอดเดรสของไอซี 8255 ทั้งสองตัว สามารถพิจารณาได้จากการแจกแจงค่าในแต่ละบิทของแอดเดรสต่าง ๆ จากช่องสล๊อตของคอมพิวเตอร์ ดังตารางในหน้าถัดไป

A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	Port Address	Port Name	8255 ตัวที่
1	1	0	0	0	1	0	0	0	0	310H	Port A	1
1	1	0	0	0	1	0	0	0	1	311H	Port B	1
1	1	0	0	0	1	0	0	1	0	312H	Port C	1
1	1	0	0	0	1	0	0	1	1	313H	Control	1
1	1	0	0	0	1	0	1	0	0	314H	Port A	2
1	1	0	0	0	1	0	1	0	1	315H	Port B	2
1	1	0	0	0	1	0	1	1	0	316H	Port C	2
1	1	0	0	0	1	0	1	1	1	317H	Control	2

จากตารางจะเห็นว่าสำหรับ 8255 ทั้งสองตัวนั้น มีเพียง A1 และ A0 เท่านั้นที่เปลี่ยนเมื่อแอดเดรสเปลี่ยน ดังนั้นเราจึงสามารถสร้างวงจร Decode แอดเดรสของ 8255 ทั้งสองตัวได้โดยการนำสัญญาณ A1 และ A0 ไปเข้าไอซีทั้งสองตัว โดยมีสัญญาณเลือกให้ไอซีตัวใดทำงานจากการนำเอาบิทที่เหลือของสัญญาณแอดเดรสนี้ไปเข้าวงจรเกตดังรูปที่ 3.2



รูปที่ 3.2 แสดงการเลือก Decode แอดเดรสของ 8255 ทั้งสองตัว

สำหรับ A0-A1 นั้นต่อเข้าโดยตรงกับ 8255 ทั้ง 2 ตัวเลย เพื่อใช้ถอดรหัสพอร์ตภายในซึ่งหมายเลขพอร์ตแสดงไว้ในตารางด้านบนแล้ว นอกจากนั้นเราจะต้องนำสัญญาณ AEN (Address Enable output) ซึ่งเป็นสัญญาณที่ใช้สำหรับแยกบัสแอดเดรสในการทำขบวนการ DMA เมื่อสัญญาณนี้แอคทีฟจะทำให้ DMA Controller สามารถควบคุมการทำงานแทนการควบคุมของ CPU มาต่อร่วมอยู่ด้วยเพื่อไม่ให้เกิดการชนกันของข้อมูลเมื่อ CPU ทำขบวนการ DMA นั่นเอง ขบวนการ DMA เป็นการเพิ่มความเร็วของการส่งผ่านข้อมูลระหว่างหน่วยความจำกับอุปกรณ์ I/O ได้ดียิ่งขึ้น

ในส่วนของ 8255 ยังไม่จบเพียงเท่านี้ เรายังต้องส่งรหัสควบคุม (Control byte) เข้าไปยังพอร์ตควบคุมข้อมูล (port สุดท้ายใน 4 พอร์ต คือ ที่ A1 กับ A0 เป็น 1 ซึ่งได้แก่ พอร์ตหมายเลข 310H และ 317H ของ 8255 ทั้งสองตัว) ให้ทำงานในโหมดไหนและให้เป็นอินพุทหรือเอ้าต์พุทพอร์ต ความหมายของบิทต่าง ๆ ของรหัสควบคุม (Control byte) หรือรหัสสั่งงาน ของ 8255 เป็นดังนี้คือ

D7 - แสดงถึงรหัสควบคุมให้เริ่มทำงาน (1 คือ ทำงาน) คือจะมีผลทำให้ 8255 รับรู้สิ่งต่อไปนี้ บิทต่าง ๆ ที่จะกำหนดให้ เพราะฉะนั้นเวลาสั่งงานหรือสั่งหน้าที่ให้กับ 8255 บิทนี้จะเป็นลอจิก 1 เสมอ

D6,D5 - เป็นการเลือกโหมดการทำงานของ พอร์ต A ซึ่งมีอยู่ 3 โหมด ใน 8255 จะได้กล่าวต่อไป

D4 - กำหนดให้พอร์ต A เป็นอินพุทหรือเอ้าต์พุท โดยที่

0 = Output port

1 = Input port

D3 - กำหนดให้ port C บนเป็นอินพุทหรือเอ้าต์พุทโดย

0 = Output port

1 = Input port

D2 - เป็นการเลือกโหมดให้กับพอร์ต B

0 = Mode 0

1 = Mode 1

D1 - กำหนดให้พอร์ต B เป็นอินพุทหรือเอ้าต์พุทโดย

0 = Output port

1 = Input port

D0 - กำหนดให้พอร์ต C ล่างเป็นอินพุทหรือเอ้าต์พุทพอร์ตโดย

0 = Output port

1 = Input port

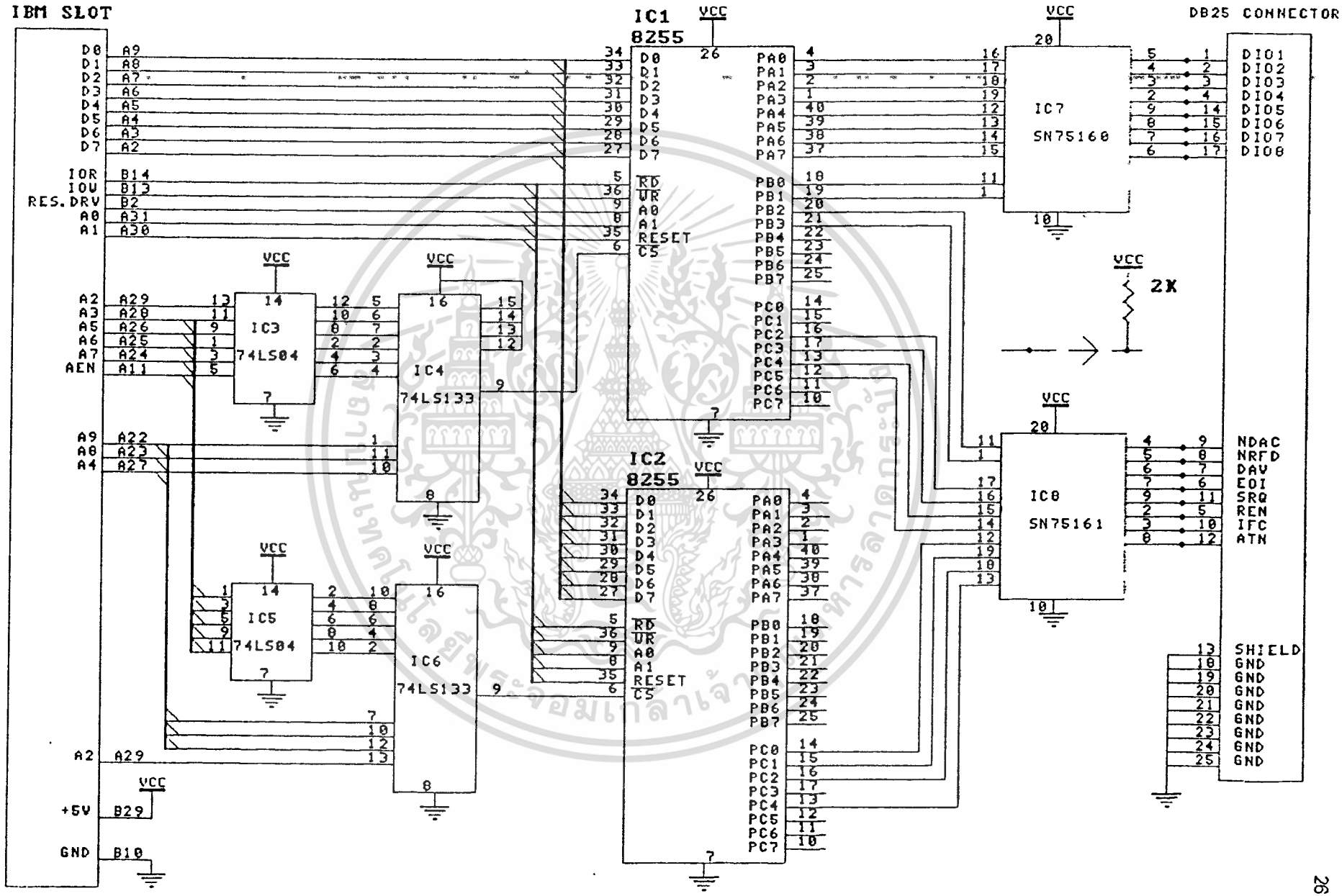
ซึ่งแสดงได้ดังต่อไปนี้

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	X	X	0	0	X

จะเห็นได้ว่าเรายังไม่สามารถระบุได้แน่ชัดว่ารหัสคำสั่งเป็นอะไรทั้งนี้เพราะพอร์ต A พอร์ต C บนและพอร์ต C ล่างสามารถเปลี่ยนแปลงได้ $2^3 = 8$ ค่าดังนี้

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	X	X	0	0	X
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	0	1	0	0	0	1
1	0	0	1	1	0	0	0
1	0	0	1	1	0	0	1

โดยเราจะได้รหัสควบคุมคือ 80H,81H,88H,89H,90H,91Hและ99H ตามลำดับ จากที่ผ่านมาเราสามารถออกแบบ Hardware ของระบบบัสของ GPIB ได้ตามกระบวนการที่ได้กล่าวมาแล้วในตอนต้นทั้งหมด ในตอนต่อไปนี้ก็จะเป็นการรวมวงจรทั้งหมดที่กล่าวถึงมารวมกันเป็นวงจรใช้งานจริงในหน้าถัดไป



รูปที่ 3.3 วงจร IEEE 488 บัส

บทที่ 4

การเขียนโปรแกรมควบคุม GPIB CARD

ภาษาที่ใช้ในการเขียนโปรแกรมสำหรับควบคุมการ์ดในโครงงานที่ใช้ คือ Turbo Pascal Version 5.5 ซึ่งรายละเอียดที่น่าสนใจสำหรับโปรแกรมนั้นจะอยู่ที่ Procedure ต่าง ๆ ที่ใช้สำหรับจัดการระบบ โดย Procedure เหล่านี้ได้เขียนรวมไว้ใน Unit IEEE488 รายละเอียดของแต่ละ Procedure มีดังนี้

Init_Card

รูปแบบ Init_card;

การทำงาน ทำการกำหนดโหมดการทำงานของ 8255,75160A และ 75161A ในขณะเริ่มต้น run program ซึ่งผู้ใช้จะต้องเรียกใช้เป็นอันดับแรก Flow Chart การทำงานแสดงดังรูปที่ 4.1

Init_Set_Device

รูปแบบ Init_Set_Device(addr : integer);

การทำงาน จะกำหนดให้ device ที่มี address เท่ากับ addr (เลขฐานสิบ) เป็น listener Flow Chart แสดงการทำงานคือรูปที่ 4.2

IFC

รูปแบบ IFC;

การทำงาน ทำการเคลียร์สัญญาณควบคุมทั้งหมด ให้กลับสู่สภาวะเริ่มต้น ด้วยการทำขา IFC ให้เป็น True (Low Level Voltage) Flow Chart การทำงานแสดงดังรูปที่ 4.3

DCL

รูปแบบ DCL;

การทำงาน ทำให้ device ทุกตัวที่ต่ออยู่ในระบบ กลับสู่สภาวะเริ่มต้น โดยการส่งคำสั่ง DCL ไปยัง device ทุกตัว Flow Chart การทำงานแสดงดังรูปที่ 4.4

Group_Execute_Trigger

รูปแบบ Group_Execute_Trigger(addr integer);

การทำงาน กำหนดให้ device ที่มี address เท่ากับ addr กลับมาอยู่ในสภาพเป็น Listener Flow Chart การทำงานแสดงดังรูปที่ 4.5

Go_To_Local

รูปแบบ Go_To_Local(addr : integer);

การทำงาน กำหนดให้ device ที่มี address เท่ากับ addr กลับไปสู่การควบคุมด้วยมือ โดยการส่งคำสั่ง GTL ไปยัง device Flow Chart การทำงานแสดงดังรูปที่ 4.6

Local_Lock_Out

รูปแบบ Local_Lock_Out;

การทำงาน กำหนดให้ การกดปุ่ม Local ที่หน้าปัดเครื่อง ใช้งานไม่ได้ โดยการส่งคำสั่ง LLO Flow Chart แสดงการทำงานดังรูปที่ 4.7

Select_Device_Clear

รูปแบบ Select_Device_Clear(addr : integer);

การทำงาน กำหนดให้ device ที่มี address เท่ากับ addr เคลียร์ตัวเองกลับสู่สภาวะเริ่มต้นเมื่อเริ่มเปิดเครื่อง โดยการส่งคำสั่ง SDC Flow Chart การทำงานแสดงดังรูปที่ 4.8

Send_Command

รูปแบบ Send_Command(addr : integer; Command : byte);

การทำงาน ทำการส่งคำสั่งควบคุมต่าง ๆ เพื่อควบคุม device ที่มี address ตรงกับ addr ให้เป็นไปตามที่ต้องการ Flow Chart การทำงานแสดงดังรูปที่ 4.9

Send_Message

รูปแบบ Send_Message(addr : integer; m : string; eoi : char);

การทำงาน ทำการส่งข้อความ m ไปยัง device ที่มี address ตรงกับ addr เพื่อควบคุมการทำงานของ device ซึ่งข้อความ m นี้ ดูได้จาก manual ของ device แต่ละเครื่องที่ต้องการจะควบคุม สำหรับ eoi ถ้าให้เป็น 'y' จะทำการยืนยันขา eoi ที่ขั้วต่อให้เป็น True เพื่อบอกถึงการจบข้อความที่จะส่ง แต่ถ้าเป็นอักขระตัวอื่นจะไม่ยืนยันขา eoi Flow Chart การทำงานแสดงดังรูปที่ 4.10

Get_Data1Byte

รูปแบบ Get_Data1Value(addr : byte; var d : string; eos : char);

การทำงาน ทำการรับข้อมูลมา 1 ค่า โดยค่าที่ได้จะยังอยู่ในรูปของ String ซึ่งอยู่ในตัวแปร d จาก device ที่มี address เท่ากับ addr สำหรับ eos เป็นอักขระ 1 ตัว ที่ผู้ใช้จะต้องบอก ในกรณีที่ device ตัวนั้นมีอักขระปิดท้ายข้อความเช่น LF แต่ถ้าไม่มีจะให้เป็นอักขระอะไรก็ได้ Flow Chart การทำงานแสดงดังรูปที่ 4.11

Spolling 1 Byte

รูปแบบ Spolling1Byte(addr : integer; var databyte : byte);

การทำงาน ทำการอ่านข้อมูล 1 ไบต์ จาก device ที่มี address เท่ากับ addr โดยข้อมูล 1 ไบต์นี้ แต่ละบิท จะบอกความหมายในการขอบริการของ device ตัวนั้น จากตัว Controller ซึ่งผู้ใช้ต้องนำข้อมูลนี้ซึ่งอยู่ในตัวแปร databyte ไปเปรียบเทียบกับ Serial Poll Byte ของ device ตัวนั้น ซึ่งดูได้จาก Manual Flow Chart การทำงานแสดงดังรูปที่ 4.12

Procedure บาง Procedure ถ้านำไปใช้แล้วเกิด error ขึ้นมาในกรณีที่ติดต่อกับ device ไม่ได้ จะมีการแสดงข้อความ "Bus Error" ซึ่งแสดงในโหมดกราฟิก ดังนั้นผู้ที่ต้องการจะเขียนโปรแกรมในเท็กซ์โหมด ขอให้เปลี่ยน Procedure BusError ให้แสดงข้อความในเท็กซ์โหมดด้วย

สำหรับ Procedure อีกบาง Procedure ที่ผู้ใช้ไม่สามารถเรียกใช้โดยตรงได้ แต่ Procedure ที่ผู้ใช้เรียกใช้ได้บางตัวจะไปเรียก Procedure เหล่านี้เอง เพื่อความเข้าใจจึงจะแสดง Flow Chart และการทำงานดังต่อไปนี้

Addr_to_Hex(addr : integer; var MLA,MTA : byte);

เป็น Procedure สำหรับหาค่า MLA (My Listen Address) และ MTA (My Talk Address) โดยโปรแกรมจะไปเปิด array ลำดับที่ addr ซึ่งกำหนดเป็นค่าคงที่ไว้แล้ว Flow Chart การทำงานเป็นดังรูปที่ 4.13

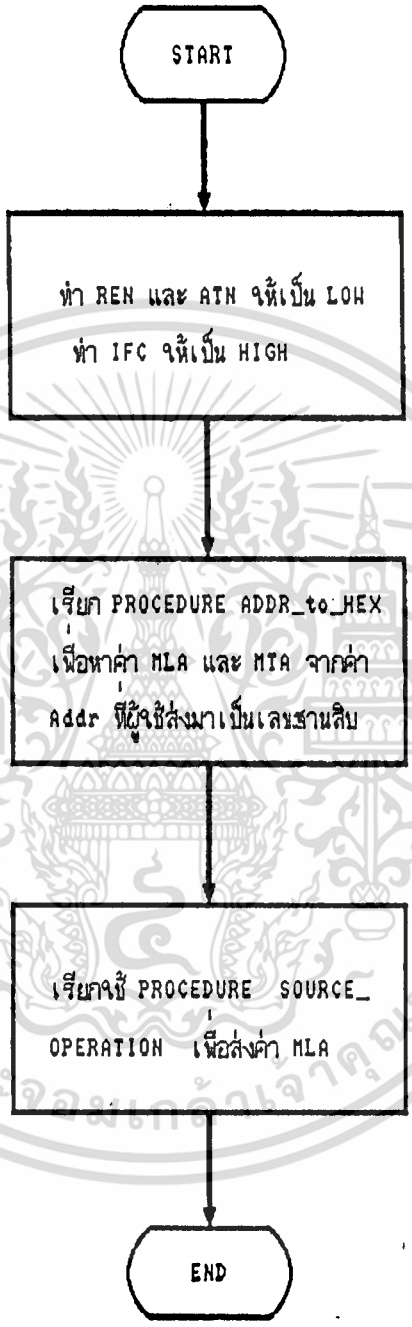
Source_Operation(data : byte);

เป็น Procedure สำหรับการส่งข้อความหรือข้อมูลขนาด 1 ไบต์ การทำงานแสดงดัง Flow Chart รูปที่ 4.14

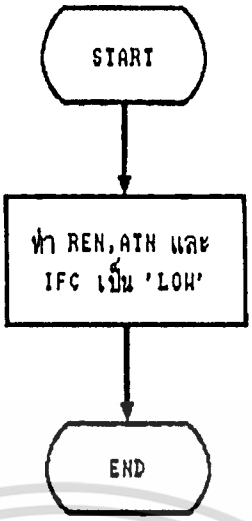
Acceptor_Operation(var databyte : byte);

ทำกระบวนการ Acceptor_Operation เพื่อการรับข้อความหรือข้อมูลขนาด 1 ไบต์ มี Flow Chart แสดงการทำงานดังรูปที่ 4.15

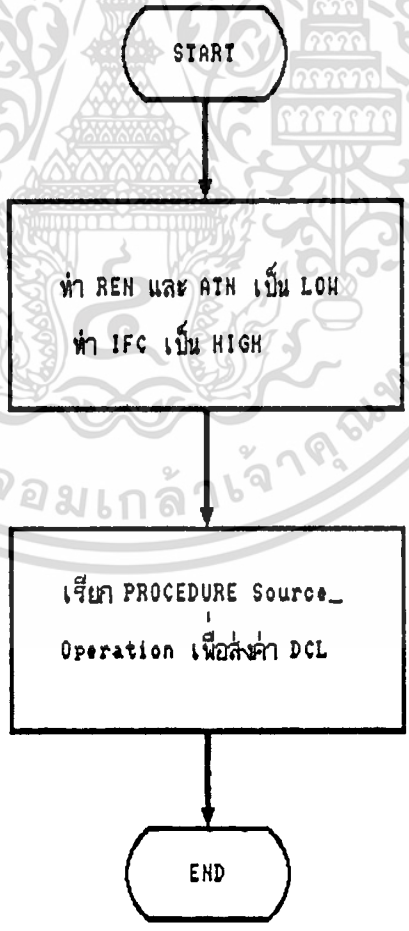
รูปที่ 4.1 Flow Chart แสดงการทำงานของ Procedure Init_Card



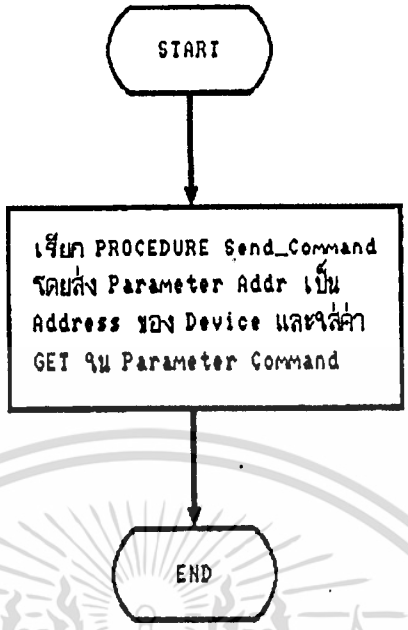
รูปที่ 4.2 Flow Chart แสดงการทำงานของ Procedure Init_Set_Device



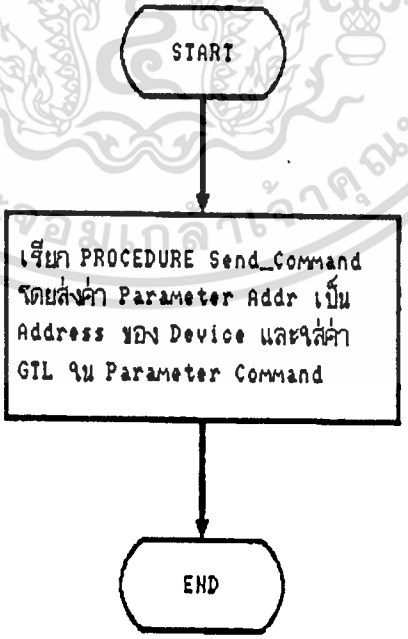
รูปที่ 4.3 Flow Chart แสดงการทำงานของ Procedure IFC



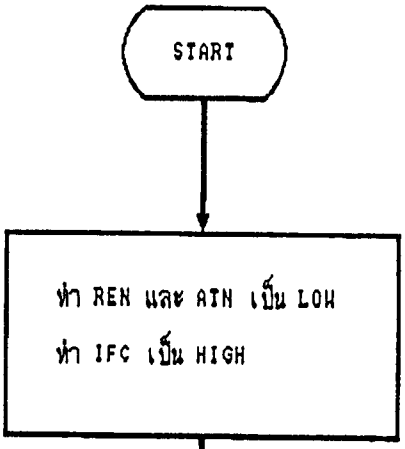
รูปที่ 4.4 Flow Chart แสดงการทำงานของ Procedure DCL



รูปที่ 4.5 Flow Chart แสดงการทำงานของ Procedure Group_Execute_Trigger

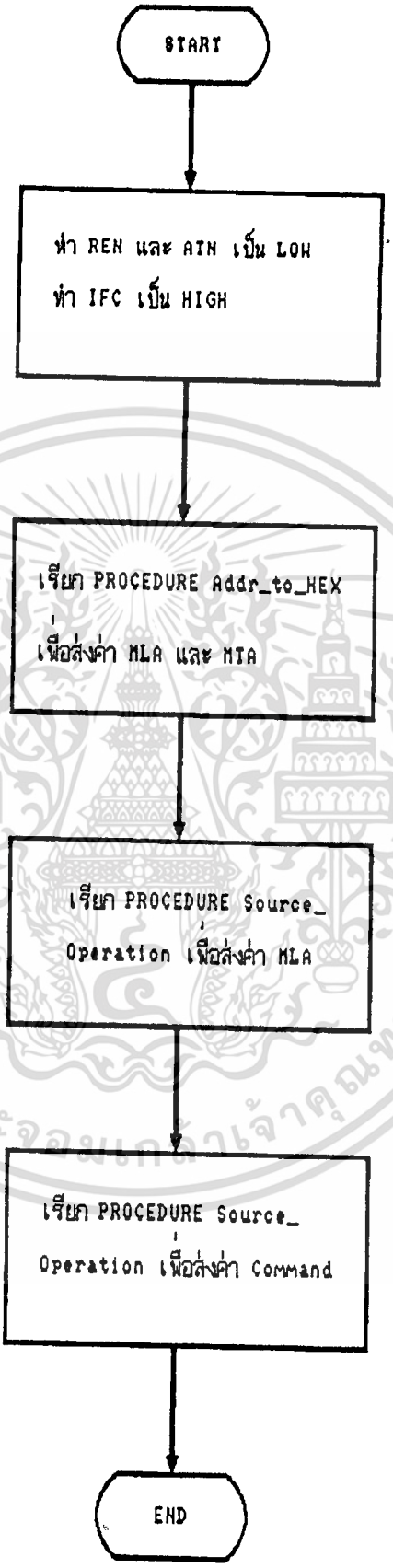


รูปที่ 4.6 Flow Chart แสดงการทำงานของ Procedure Go_to_Local

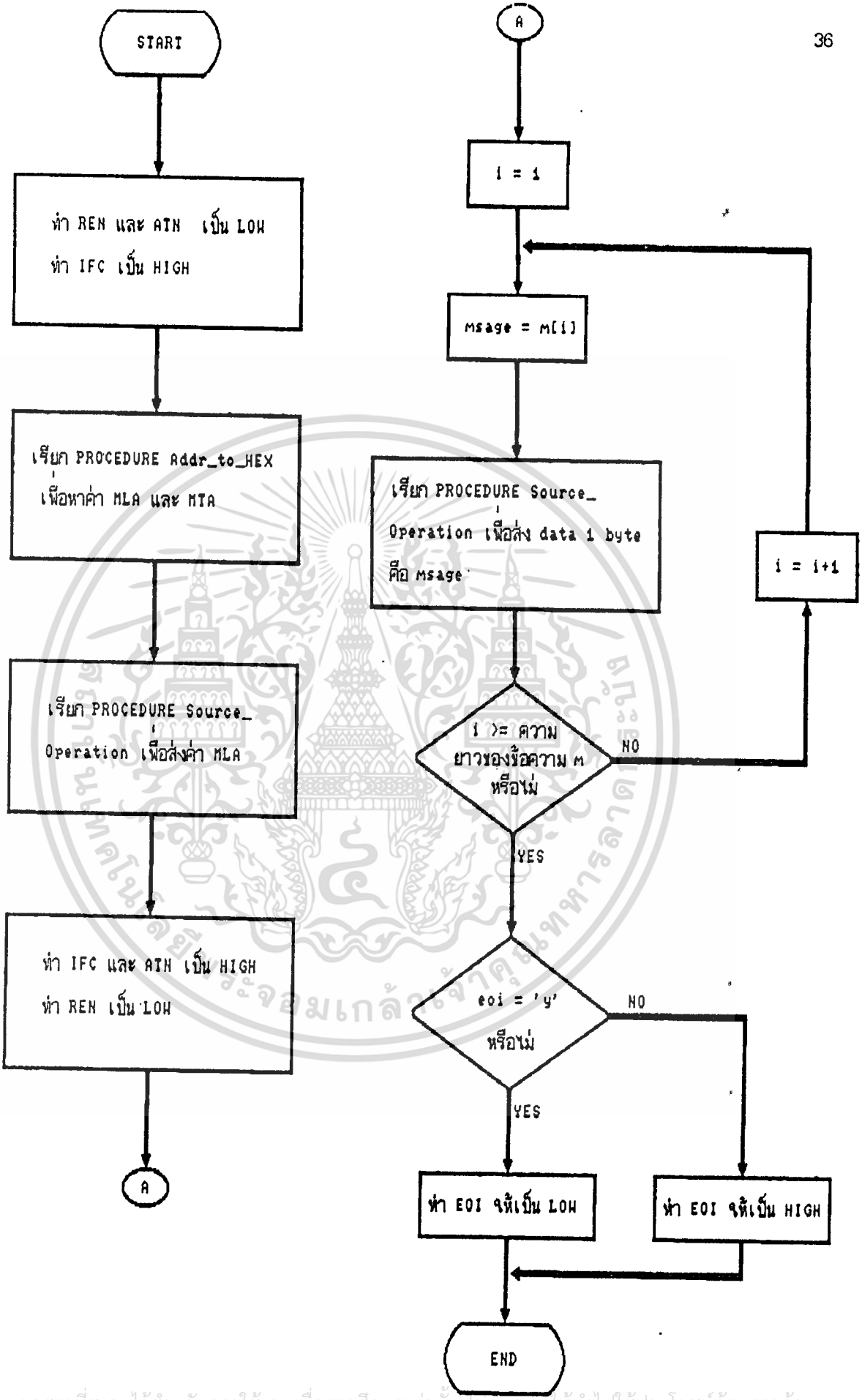


รูปที่ 4.7 Flow Chart แสดงการทำงานของ Procedure Local_Lock_Out

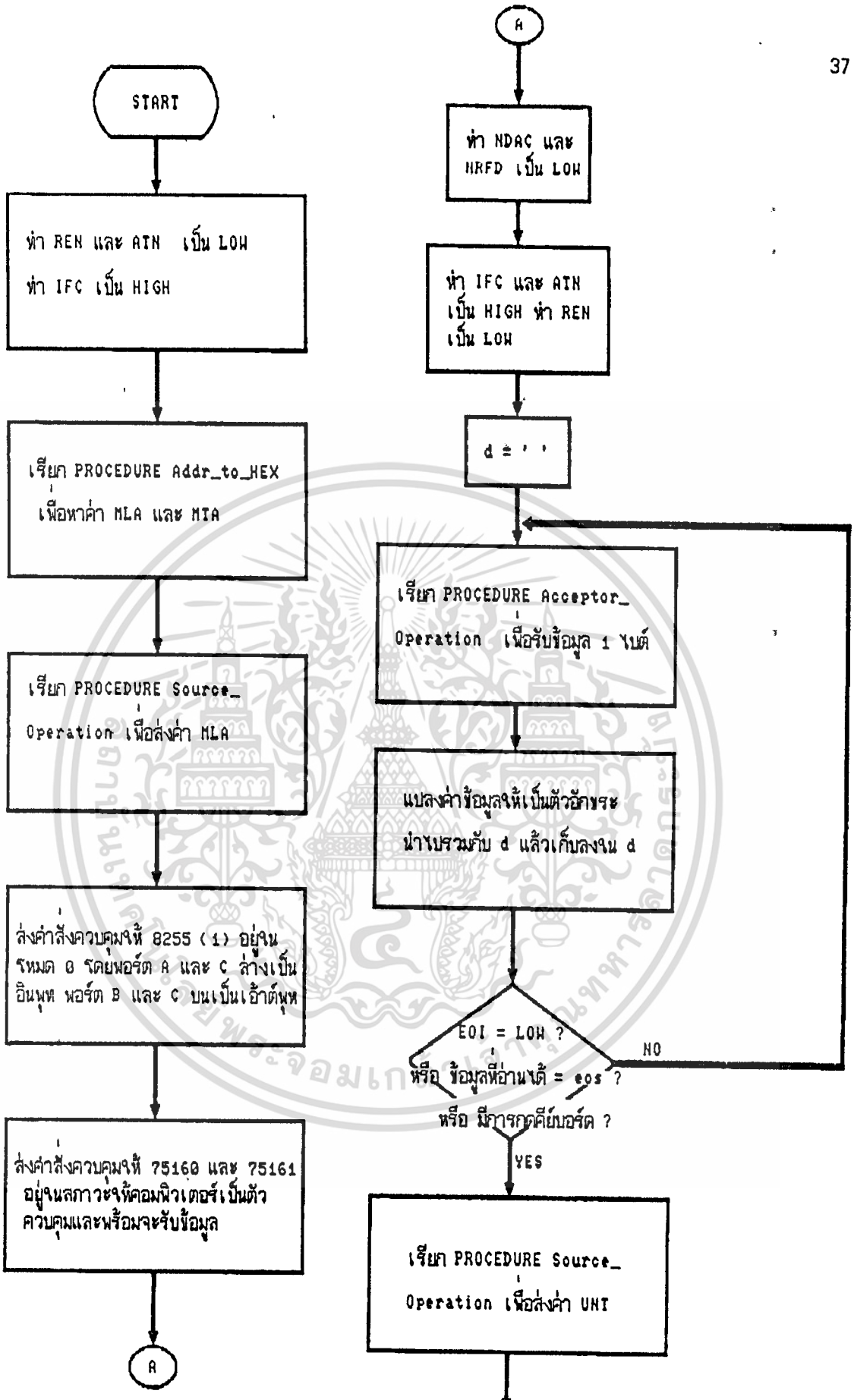
รูปที่ 4.8 Flow Chart แสดงการทำงานของ Procedure Select_Device_Clear



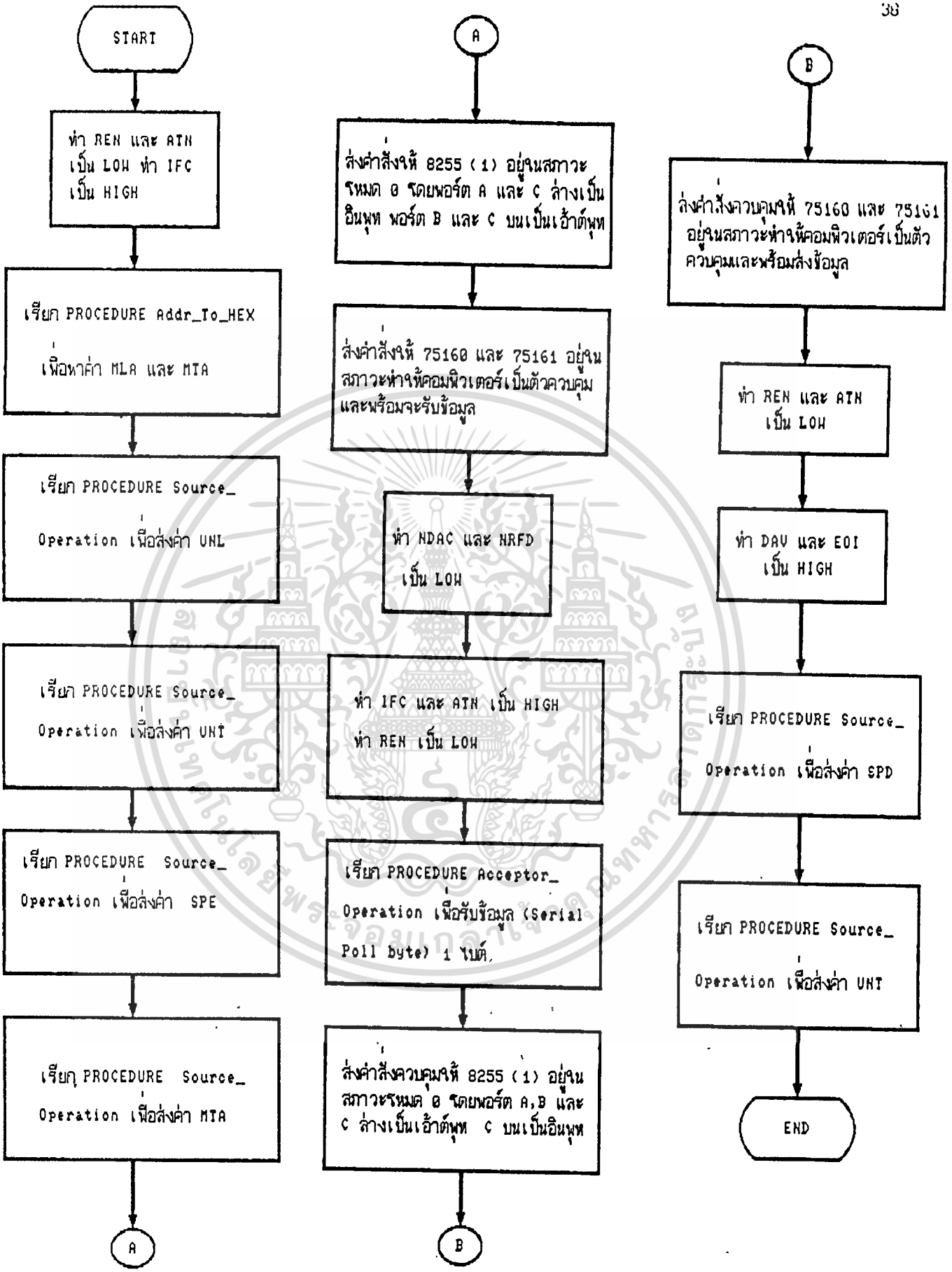
รูปที่ 4.9 Flow Chart แสดงการทำงานของ Procedure Send_Command



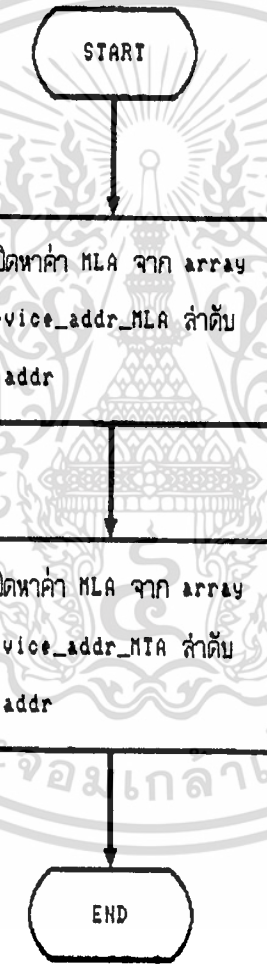
รูปที่ 4.10 Flow Chart แสดงการทำงานของ Procedure Send_Message



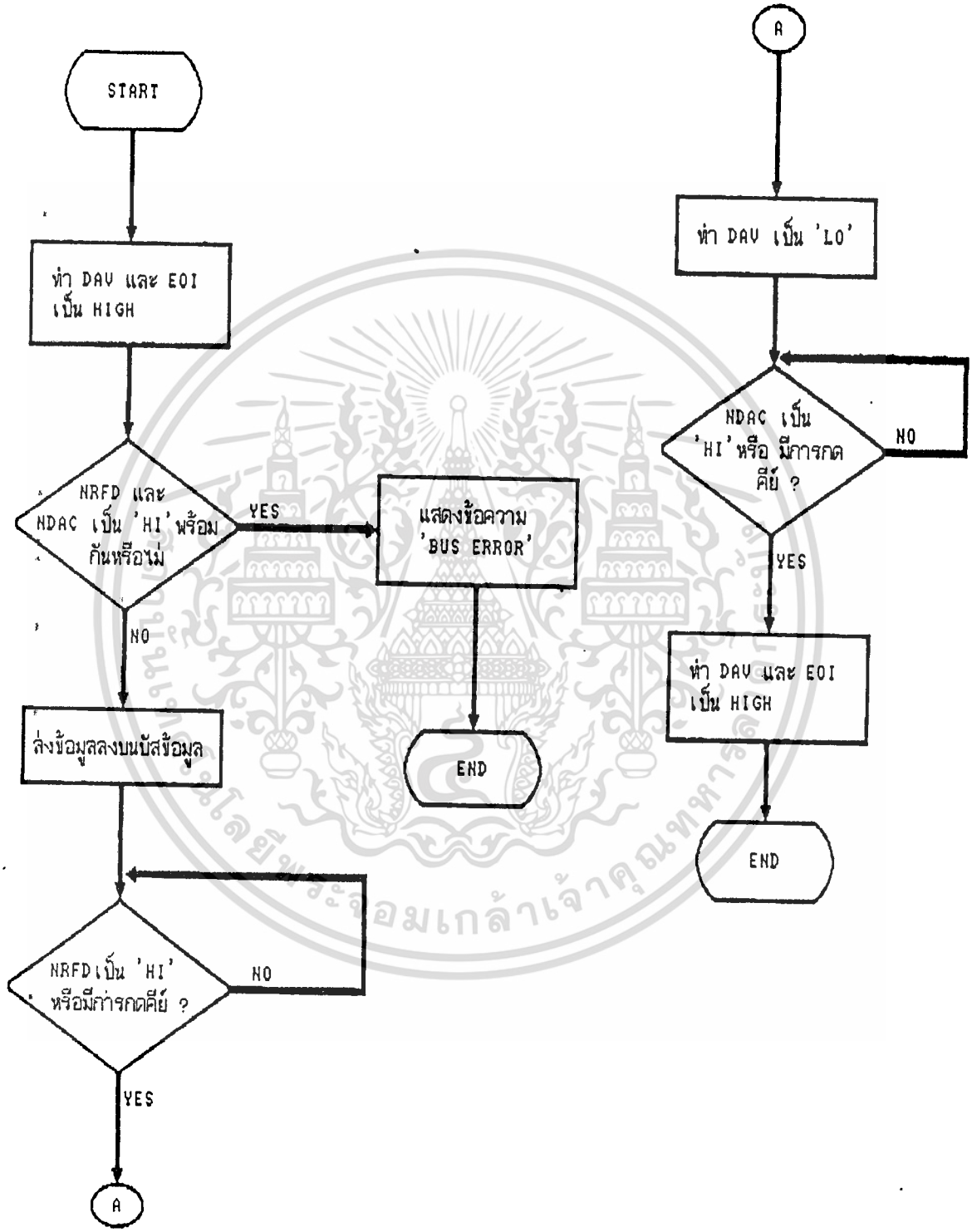
รูปที่ 4.11 Flow Chart แสดงการทำงานของ Procedure Get_Data1byte



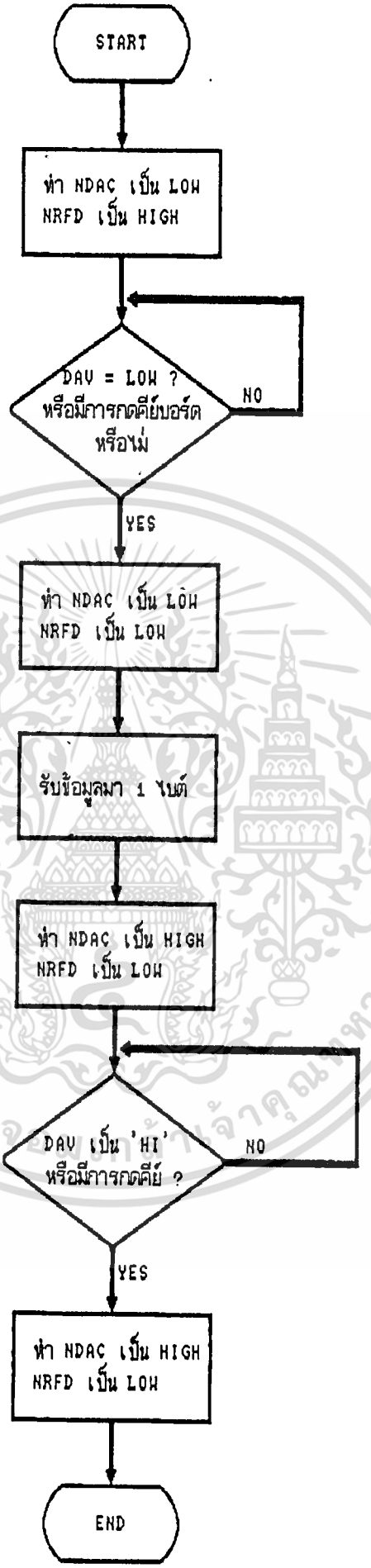
รูปที่ 4.12 Flow Chart แสดงการทำงานของ Procedure Spolling1byte



รูปที่ 4.13 Flow Chart แสดงการทำงานของ Procedure Addr_to_Hex



รูปที่ 4.14 Flow Chart แสดงการทำงานของ Procedure Source_Operation



รูปที่ 4.15 Flow Chart แสดงการทำงานของ Procedure Acceptor_Operation

บทที่ 5

การใช้โปรแกรมและการประยุกต์

จากที่ได้กล่าวไว้พอสังเขปบ้างแล้วในส่วนของบทคัดย่อว่าการใช้งานโปรแกรมนี้จะแบ่งออกได้เป็น 2 ส่วนใหญ่ ๆ คือ ในส่วนของโปรแกรมสำเร็จรูปที่เขียนขึ้นให้ผู้ใช้ ใช้งานทั่ว ๆ ไปเช่นใช้รับส่งข้อมูลกับเครื่องมือวัด แล้วนำมาเขียนกราฟเพื่อประโยชน์บางอย่างเช่น กราฟความสัมพันธ์ระหว่างโวลเตจกับกระแส ก็คือความต้านทาน ซึ่งมีประโยชน์อย่างมากในการหาความต้านทานของวงจรรวมใด ๆ

และในส่วนที่สองเป็นส่วนของโปรแกรมยูนิตซึ่งได้กล่าวถึงไว้ในบทที่ 4 โดยผู้ใช้สามารถนำยูนิตนี้ไปใช้ประกอบการเขียนโปรแกรมควบคุมการ์ดอินเทอร์เฟสนี้ เพื่องานเฉพาะอย่างได้ เช่นเขียนโปรแกรมวนลูปรับค่าของความถี่ที่ได้จากเครื่องวัดความถี่ เพื่อนำไปควบคุมเครื่องกำเนิดความถี่ให้จ่ายความถี่ค่าที่ต้องการคงที่ตลอดเวลา ซึ่งก็จะได้กล่าวในรายละเอียดของการนำไปประยุกต์ใช้ต่อไป

จากโปรแกรมสำเร็จรูปที่เขียนขึ้นมานั้น เราสามารถแบ่งเมนูใช้งานออกเป็นเมนูใหญ่ ๆ ได้ 5 เมนู คือ เมนู Quit,Option,GPIB,Print และ Plot โดยแต่ละเมนูใหญ่ ๆ นี้ ก็จะแบ่งเป็นเมนูย่อย ๆ ลงไปอีก ในที่นี้จะขออธิบายการใช้งานเมนูแต่ตัวไปตามลำดับคือ

1. **เมนู Quit** เป็นเมนูที่เราจะเลือกเมื่อต้องการเลิกการใช้งานโปรแกรม พร้อมกับออกไประบบปฏิบัติการทันที

2. **เมนู Option** เป็นเมนูที่อำนวยความสะดวกในการใช้โปรแกรมโดยประกอบไปด้วยเมนูย่อย ๆ คือ

- **เมนู Initial Time** ใช้สำหรับดูเวลา และตั้งค่าเวลาใหม่

-**เมนู Initial Date** ใช้สำหรับดูวัน เดือน ปี ปัจจุบัน และสามารถตั้งค่าเหล่านี้ได้ใหม่เช่นเดียวกับการตั้งค่าเวลา

- **เมนู Program Code** เมื่อเราเลือกเมนูนี้ จะเป็นการส่งข้อมูลซึ่งเป็นชุดคำสั่งสำหรับส่งให้เครื่องมือวัดทำตามคำสั่งที่ผู้ใช้ส่งไปเป็น Program Code นี้ โดยคำสั่งนี้โปรแกรมจะยอมให้ส่งได้ครั้งละไม่เกิน 45 ตัวอักษร โดยเราสามารถเก็บชุดคำสั่งนี้ไว้ในไฟล์ เพื่อนำออกมาใช้ซ้ำได้อีกตามที่ต้องการ

3. **เมนู GPIB** เป็นเมนูสำหรับใช้งานโดยเฉพาะสำหรับระบบ GPIB นี้ ใช้สำหรับการติดต่อสื่อสารกันระหว่างผู้ใช้กับเครื่องมือวัดต่าง ๆ ที่ต่ออยู่ในระบบ โดยจะประกอบไปด้วยเมนูย่อย ๆ ดังนี้คือ

- **เมนู Config Device** ใช้สำหรับติดตั้งอุปกรณ์ที่เพิ่งต่อเข้ามาในระบบ หรือใช้เลือกอุปกรณ์ที่มีอยู่ในระบบแล้วที่ผู้ใช้ต้องการจะติดต่อด้วย ซึ่งในเมนูนี้ก็จะประกอบด้วยเมนูย่อย 2 เมนูคือ

1 **เมนู New Device** ใช้สำหรับติดตั้งอุปกรณ์ตัวใหม่เข้ามาในระบบ เมื่อผู้ใช้เลือกเมนูนี้แล้ว โปรแกรมก็จะให้ผู้ใช้ใส่ชื่อของอุปกรณ์ตัวใหม่นี้ พร้อมกับค่าแอดเดรส (Address) ซึ่งต้องไม่ซ้ำกับค่าแอดเดรสของเครื่องอื่น ๆ ที่ต่ออยู่ก่อนแล้ว

2 **เมนู Exist Device** ใช้สำหรับเลือกตัวอุปกรณ์ซึ่งมีอยู่ในระบบแล้ว และผู้ใช้ต้องการจะติดต่อด้วย (เพราะในขณะหนึ่ง ๆ ผู้ใช้สามารถรับส่งข้อมูลกับเครื่องในระบบได้ครั้งละ 1 ตัวเท่านั้น) ผู้ใช้สามารถเลือก

ชื่อของอุปกรณ์ที่ต้องการได้ จากชื่ออุปกรณ์ในกรอบทางด้านขวาและใช้คีย์ลูกศรเลื่อนแถบสว่างไปยังชื่อที่ต้องการแล้วจึงกด Enter แต่ถ้าหากยังไม่มีชื่ออุปกรณ์ทางด้านนี้เลยโปรแกรมก็จะขึ้นข้อความ "Config New Device" เพื่อให้ผู้ใช้ป้อนชื่อของอุปกรณ์เข้าไปในระบบก่อน

- **เมนู Iteractive Mode** ใช้สำหรับส่งคำสั่งเพื่อเกิดการรับส่งข้อมูลกันระหว่างคำสั่งจากผู้ใช้ไปยังเครื่องมือวัด และข้อมูลต่าง ๆ ที่ได้จากการวัดจากเครื่องมือวัด ส่งมาให้ผู้ใช้ โดยในเมนูนี้จะประกอบไปด้วยเมนูย่อย ๆ คือ

1 **เมนู Basic GPIB Command** ซึ่งเป็นคำสั่งทั่ว ๆ ไปที่ใช้ในการติดตั้ง การเคลียร์ การรีโมตเครื่องมือวัดให้อยู่ภายใต้การควบคุมของผู้ใช้ ฯลฯ จะขออธิบายคร่าว ๆ ดังนี้คือ

- **คำสั่ง Initialize** ใช้สำหรับการเซ็ตระบบเมื่อจะเริ่มต้นใช้งาน

- **คำสั่ง Interface Clear** ใช้สำหรับการเคลียร์ให้อุปกรณ์ทุกตัวพร้อมที่จะรับคำสั่งใช้งาน

- **คำสั่ง Device Clear** ใช้สำหรับเลือกเคลียร์เฉพาะอุปกรณ์ตัวใดตัวหนึ่งที่เราต้องการ

- **คำสั่ง Group Exec Trig** เป็นคำสั่งสำหรับสั่งให้อุปกรณ์เฉพาะตัวใดตัวหนึ่งกลับมาสู่สภาพการทำงานตามปกติหลังจากที่เริ่มอยู่ในสภาพรีโมตอีกครั้งหนึ่ง

- **คำสั่ง Go To Local** เป็นคำสั่งที่ใช้สั่งให้อุปกรณ์ที่กำลังติดต่ออยู่กลับไปอยู่ในสภาพที่ สามารถบังคับได้จากปุ่มปรับที่หน้าปัดตามปกติ

- **คำสั่ง Local Lockout** เป็นคำสั่งสำหรับล็อคให้อุปกรณ์ที่กำลังใช้งานอยู่ อยู่ในสภาพรีโมตที่ไม่สามารถใช้ปุ่ม Local ที่หน้าปัดเครื่องทำให้กลับไปอยู่ในสภาพควบคุมด้วยมือได้

- **คำสั่ง Select Dev Clr** ใช้งานได้เช่นเดียวกับคำสั่ง Device Clear

- **คำสั่ง Set Device** ใช้สำหรับเซ็ตให้อุปกรณ์อยู่ในสภาวะที่ต้องการโดยเลือกชื่ออุปกรณ์ที่ต้องการจากชื่อในกรอบทางด้านขวามือ และใช้คีย์ลูกศรเลื่อนแถบสว่างไปยังชื่อที่ต้องการ

2. **เมนู Data Transfer Command** ใช้สำหรับการรับส่งข้อมูลระหว่างผู้ใช้และเครื่องมือวัดภายในระบบ โดยประกอบไปด้วยคำสั่งต่าง ๆ ดังนี้ คือ

- **คำสั่ง Listen** ใช้สำหรับรับข้อมูลที่ส่งมาจากเครื่องมือวัดที่เราต้องการโดยเราสามารถทำการรับข้อมูลได้ใน 4 ลักษณะคือ

1. นำข้อมูลที่ได้ไปเก็บไว้ในไฟล์ โดยการเลือกเมนูย่อย File โปรแกรมก็จะให้ผู้ใช้ใส่ชื่อไฟล์ที่ต้องเก็บข้อมูลไว้

2. นำข้อมูลที่รับได้มาแสดงบนจอภาพแสดงผล โดยการเลือกเมนูย่อย Display โดยโปรแกรมก็จะถามว่าข้อมูลที่ส่งมามีค่า EOS (เป็นตัวบอกถึงจุดสิ้นสุดของข้อความหนึ่ง ๆ) ส่งมาด้วยหรือไม่ ถ้ามี ค่า EOS นี้มีรหัสแอสกี้เป็นเท่าไรก็ให้ผู้ใช้ใส่ค่าลงไป จากนั้นโปรแกรมก็จะถามว่าต้องการข้อมูลทศนิยมกี่ตำแหน่ง ก็ให้ผู้ใช้ใส่จำนวนจุดทศนิยมที่ต้องการลงไป เมื่อรับข้อมูลมาเสร็จแล้ว โปรแกรมก็จะถามว่าต้องการอ่านข้อมูลตัวใหม่เข้ามาอีกหรือไม่ ถ้าต้องการก็ให้ตอบ Yes ถ้าไม่ ก็ให้ตอบ No

- **คำสั่ง Talk** ใช้สำหรับส่งคำสั่งควบคุมให้เครื่องมือวัดทำงานตามฟังก์ชั่นที่ผู้ใช้กำหนด โดยแต่ละ

คำสั่งก็จะขึ้นอยู่กับแต่ละเครื่องว่าใช้คำสั่งอย่างไร โดยเราสามารถส่งคำสั่งออกไปได้ใน 2 ลักษณะคือ

1. ส่งไปในรูปของลำดับคำสั่ง ต่อ ๆ กันไป 1 ลำดับ โดยการเลือกเมนู Literal ซึ่งโปรแกรมก็จะยอมให้ผู้ใช้ส่งคำสั่งได้ครั้งละไม่เกิน 45 ตัวอักษร เมื่อเสร็จสิ้นการส่งแต่ละครั้งโปรแกรมก็จะถามว่าต้องการส่งคำสั่งอื่น ๆ ไปอีกหรือไม่โดยการเลือกตอบ Yes หรือ No

2. ส่งคำสั่งที่เก็บอยู่ในไฟล์ออกไป โดยการเลือกเมนูย่อยนี้ โปรแกรมก็จะถามชื่อไฟล์ที่เก็บ Program Code ที่ผู้ใช้ต้องการส่งไป

- **คำสั่ง Serial Poll** ใช้สำหรับการตรวจสอบแบบอนุกรมเมื่อเกิดมีสัญญาณ Request มาจากเครื่องมือวัดภายในระบบซึ่งยังไม่สามารถบอกได้ว่าอุปกรณ์ตัวใดเป็นผู้ส่งสัญญาณมา จึงต้องการตรวจสอบแบบอนุกรมขึ้นมา

4. **เมนู Print** ซึ่งเป็นเมนูสำหรับการแสดงไฟล์ข้อมูลที่ได้จากเครื่องมือวัด โดยจะแสดงออกใน 2 รูปแบบคือ
- พิมพ์ออกมาทางเครื่องพิมพ์ โดยการเลือกเมนูย่อย Printer พร้อมทั้งต้องป้อนชื่อไฟล์ที่ต้องการเข้าไปด้วย
- แสดงออกมาทางจอภาพ ซึ่งผู้ใช้ก็ต้องป้อนชื่อไฟล์เข้าไปด้วยเช่นกัน

5. **เมนู Plot** ใช้สำหรับเขียนกราฟความสัมพันธ์ระหว่างตัวแปร 2 ชนิดที่ต้องการเช่น ระหว่างกระแสกับแรงดัน,ระหว่างโวลเตจกับความถี่ โดยโปรแกรมจะให้ผู้ใช้ป้อนชื่อแกน X ,แกน Y และชื่อของกราฟที่ต้องการและให้เลือกตัวแปรที่ต้องการนำมาเขียนกราฟซึ่งอยู่ทางด้านล่างของจอภาพเป็น A,B เป็นต้น

จากเมนูและคำสั่งใช้งานต่าง ๆ ก็พอจะอำนวยความสะดวกในการใช้งานของผู้ใช้พอสมควรโดยจะขอยกตัวอย่างการนำไปใช้งานคร่าว ๆ เช่นในกรณีของการหาความต้านทานอินพุทของวงจรใด ๆ เราสามารถทำได้โดยการป้อนโวลเตจอินพุทเข้าไปพร้อมกับวัดค่ากระแสอินพุท ณ ค่าโวลเตจนั้น ๆ โดยการเปลี่ยนค่าโวลเตจอินพุทไปหลาย ๆ ค่า นำค่าโวลเตจและกระแสที่จุดนั้น ๆ มาเขียนกราฟความสัมพันธ์เราก็จะสามารถหาค่าความต้านทานรวมทางด้านอินพุทของวงจรได้ โดยถ้าให้แกน X เป็นค่ากระแส และแกน Y เป็นค่าโวลเตจ ค่าความต้านอินพุทก็คือค่าความชันของกราฟนั่นเอง

ส่วนในกรณีของการใช้งานโปรแกรมยูนิตนั้นผู้ใช้ต้องมีพื้นฐานในการเขียนโปรแกรมภาษาปาสคาลได้พอสมควร โดยจะขอยกตัวอย่างโปรแกรมดังนี้ (ผู้เขียนโปรแกรมจะต้องนำโปรแกรมไปคอมไพล์กับ Turbo Pascal 5.5 สำหรับตัวอย่างนี้ จะทำการกำหนดช่วงการวัดและฟังก์ชั่นการวัดให้กับเครื่องดิจิตอลโวลต์มิเตอร์ 3455A แล้วทำการอ่านค่าการวัดมาแสดงผลบนจอ1 ค่า

```
uses    crt,IEEE488;
var     dvm_addr  :  integer;
        d             string;
begin
        clrscr,
        dvm_addr  :=  22,
```

```
        init_card;
        IFC;
        DCL;
        if bus_error then Halt
        else
        begin
                Send_message(dvm_addr,'F1','y');  {กำหนดฟังก์ชั่นดีซีโวลเตจ ให้ดิจิตอลโวลต์มิเตอร์}
                Send_message(dvm_addr,'R3','y');  {กำหนดช่วงการวัดให้ดิจิตอลโวลต์มิเตอร์เป็น 100}
                Get_data1byte(dvm_addr,d,#10);    {รับข้อมูลมา 1 ค่า ซึ่งค่าที่ได้จากโพรซีเดอร์นี้ จะเป็น
                                                  สตริง ถ้านำมาคำนวณต้องแปลงให้มีค่าเป็นตัวเลข
                                                  ก่อน}
                Writeln('Data from DVM = ',d,'Vdc');
        end;
end.
```

จากโปรแกรมนี้ มีส่วนที่ผู้ใช้ต้องแก้ไขคือ ในส่วนฟังก์ชั่น bus_error ในยูนิต IEEE488 ซึ่งมีการแสดงผลในโหมดกราฟิก ผู้ใช้จะต้องแก้ไขโปรแกรมในฟังก์ชั่น bus_error ให้แสดงผล ในเท็กซ์โหมด เสียก่อนจึงจะรันโปรแกรมนี้ได้

บทสรุปและวิจารณ์

จากการทดลองใช้โปรแกรมใช้งานอินเทอร์เฟสกับเครื่องมือวัด 2 ตัว คือ ดิจิตอลมัลติมิเตอร์ และดิจิตอลโวลต์มิเตอร์ ปรากฏผลเป็นที่น่าพอใจคือ ข้อมูลที่อ่านเข้ามาแสดงบนจอภาพกับตัวเลขที่แสดงบนหน้าปัดของเครื่องมือวัดนั้นมีค่าเท่ากัน

แต่ทั้งนี้ประสบปัญหาหลักที่สำคัญประการหนึ่งคือ เครื่องคอมพิวเตอร์ที่ใช้เป็นตัวคอนโทรลเลอร์ในการทดสอบเป็นเครื่องคอมพิวเตอร์ที่มีการทำงานที่ช้ามาก ดังนั้นทำให้เกิดการเสียเวลาเป็นอย่างมากในการอ่านข้อมูลครั้งละมาก ๆ เข้ามาแสดงผล เพราะในส่วนของโปรแกรมสำเร็จรูปจะมีส่วนของการนำข้อมูลมาเขียนกราฟ ซึ่งต้องอ่านข้อมูลเข้ามาครั้งละหลาย ๆ จุด ทำให้เสียเวลาในส่วนนี้ไป ทางแก้ไขที่จะทำได้โดยไม่ต้องมีการแก้ไขทางฮาดแวร์ของการ์ดก็คือต้องใช้คอมพิวเตอร์ที่มีความเร็วในการทำงานสูง ๆ ก็จะช่วยได้ ส่วนการแก้ไขในส่วนของฮาดแวร์นั้นก็จะช่วยให้สามารถทำงานได้เร็วขึ้นได้ ถ้าหากบนการ์ดเองมีการเพิ่มเติมในส่วนของไมโครโปรเซสเซอร์ที่จะมาทำหน้าแทนเครื่องคอมพิวเตอร์ในการทำกระบวนการแฮนด์เชค ซึ่งทางผู้จัดทำคิดว่าน่าจะมีส่วนช่วยให้การทำงานเร็วขึ้นได้เพราะในส่วนของโปรแกรมที่ทำขึ้นมา ตัวคอมพิวเตอร์จะต้องเป็นตัวทำกระบวนการแฮนด์เชคเอง ซึ่งต้องมีการทำงานในส่วนนี้ทุกครั้งก่อนที่จะมีการรับส่งข้อมูลกันซึ่งเป็นการเสียเวลา

และปัญหาที่ประสบอีกอย่างหนึ่งคือ ภาษาที่ใช้คือภาษาปาสคาลนี้จะมีข้อจำกัดในการหน่วยความจำของเครื่อง คือสามารถใช้หน่วยความจำได้ไม่มากนัก ทำให้ไม่สามารถเขียนโปรแกรมที่มีขนาดใหญ่ ๆ ได้ ซึ่งในส่วนนี้การใช้ภาษาซีจะสามารถแก้ปัญหานี้ได้

นอกจากนี้ทางผู้จัดทำได้ทดลองนำภาษาไทยมาใช้ในโปรแกรมดูแล้ว พบว่าทำได้ไม่ดีนัก เพราะจะทำให้การทำงานของโปรแกรมช้าลงไปมาก เพราะในส่วนของการแสดงผลภาษาไทยจำเป็นต้องเขียนโปรแกรมเฉพาะขึ้นมาใหม่ ซึ่งต่างกับการแสดงผลในภาษาอังกฤษซึ่งมีฟังก์ชั่นให้ใช้งานอยู่แล้วทำให้สามารถทำงานได้เร็วกว่า

แนวทางในการพัฒนาการ์ดอินเทอร์เฟสนี้ ถ้าหากไม่มีการเปลี่ยนแปลงทางด้านฮาดแวร์เลย ทางผู้จัดทำคิดว่าคงไม่สามารถเพิ่มประสิทธิภาพในการทำงานได้มากนัก ซึ่งอาจจะทำได้ในส่วนของการพัฒนาโปรแกรมให้มีฟังก์ชั่นการใช้งานให้มากขึ้น เพื่ออำนวยความสะดวกแก่ผู้ใช้ แต่ถ้าหากมีการเปลี่ยนแปลงทางด้านฮาดแวร์ก็จะทำให้สามารถพัฒนาโปรแกรมได้มากยิ่งขึ้น ซึ่งแนวทางในการพัฒนาทางด้านฮาดแวร์ก็ได้กล่าวไว้ในตอนต้นแล้วคือการนำไมโครโปรเซสเซอร์เข้ามาทำงานร่วมกันกับคอมพิวเตอร์ให้ไมโครโปรเซสเซอร์บนการ์ดทำหน้าที่ในการควบคุมสัญญาณต่าง ๆ บนบัส โดยมีคอมพิวเตอร์ซึ่งมีความสามารถในการประมวลผลดีกว่าเป็นตัวประมวลผลข้อมูลที่รับมา

กิตติกรรมประกาศ

สำหรับโปรเจ็คในครั้งนี้สามารถลุล่วงไปได้ด้วยดี ก็ด้วยความกรุณาของบุคคลหลาย ๆ ท่าน ที่ได้ให้ความช่วยเหลือ ซึ่งผู้จัดทำขอขอบคุณไว้ ณ โอกาสนี้ได้แก่

ขอบคุณอาจารย์ชนิษฐา ที่ให้คำปรึกษาตลอดการทำโปรเจ็ค ช่วยหยิบยืมอุปกรณ์ที่ใช้ในการทดลองตลอดจนอุปกรณ์ทุก ๆ อย่างในการทำโปรเจ็คนี้

ขอบคุณอาจารย์พลผดุง สำหรับเครื่องมือวัดที่ใช้ในการทดสอบโปรเจ็ค

ขอบคุณพี่อิ่ม ศูนย์วิจัยคอมพิวเตอร์ สำหรับเครื่องมือวัดที่ใช้ในการทดสอบโปรเจ็คเช่นกัน

ขอบคุณพี่อั๋น ศูนย์วิจัยและพัฒนาทางวิศวกรรม,ศูนย์วิจัยคอมพิวเตอร์ สำหรับคอมพิวเตอร์ที่ใช้พิมพ์รายงานครั้งนี้ ตลอดจนเพื่อน ๆ ที่คอยให้กำลังใจทุก ๆ คน

บรรณานุกรม

1. กนก . เจนจิระพงศ์เวช,วารสารเซมิคอนดัคเตอร์ฉบับที่ 78 บทความเรื่อง "GPIB [IEEE488] บัสอินเตอร์เฟส มาตรฐานและการใช้",บริษัทซีเอ็ดยูเคชั่น จำกัด

2. กนก เจนจิระพงศ์เวช,วารสารเซมิคอนดัคเตอร์ฉบับที่ 87 โครงงานเรื่อง "IEEE-Std 488 ผ่านพรินเตอร์การ์ด" ,บริษัทซีเอ็ดยูเคชั่น จำกัด

3. Asyst Software Technogies Inc, "ASYSTANT GPIB".

4. Eugene Fisher-C.W. Jensen, "PET and the IEEE 488 Bus (GPIB).

5. Eugene Fisher-C.W. Jensen, "PET/CBM and IEEE 488 Bus (GPIB) ,2nd Edition.

6. Hewlett Packard,"HP 3458A Multimeter",Operating,Programming and Configuration Manual.

7. Intel, "Micropressor and Peripheral Handbook" Vollume II Peripheral,2nd Edition.

ภาคผนวก

```
program test,
{$F+}
uses  overlay,tstinit,keyboard,menu,ieee488,dirtory,graph,crt,dos,printer
    ,thaitpu,mousetpu;
{$O DIRTORY}
{$O thaitpu}
{$O mousetpu}

const
  timer = $1C;
  Gray          FillPatternType = ($CA, $55, $CA,
                        $55, $C5, $5C, $CA, $55);
  menubar       : array[0..4] of string[6] = ('Quit','Option','GPIB  '
                                  ,'Print ','Plot  ');
  sub_menu_quit : array[0..1] of string[14] = ('Os-Shell      ','Quit          ');
  sub_menu_option   array[0..2] of string[16] = ('Initial Time    ',
                'Initial Date    ','Program Code    ');
  sub_menu_GPIB: array[0..1] of string[16] = ('Config Device   ',
                                  'Interactive Mode');
  sub_menu_print_scrn   : array[0 .1] of string[7] = ('Printer','Screen ');

var
  avar,bvar,cvar : array[0 .1023] of real;
  aindex,bindex,cindex,var_index,chksrq : integer;
  adt,bdt,cdt : real;
  c : char;
  i,device_num,de_num,curr_num,choice : byte;
  point,menu_num : integer;
  srq,quit,magnitude : boolean;
  device_name,temp_name,adr   array[1..30] of string;
  hh,mm,ss,hund        : word;
  command,talk_string,EOS_st,path,name;d : string;
```

```
  year, month,dat, dow : Word;
  vector : pointer;

function IntToStr(i. longint): string;
var s: string[11];
begin
  Str(i, s);
  IntToStr := s;
end;

procedure chk_SRQ(Flags,CS,IP,AX,BX,CX,DX,SI,DI,Ds,ES,BP : word); Interrupt;
begin
  Inline($FA); { CLI   disable interrupts }
  inc(chksrq);
  if chksrq=25 then
  begin
    chksrq := 0;
    if (((port[$316] and $10) <> $10) and(srq=false)) then
    begin
      write(#7);
      setcolor(4);
      outtextxy(580,29,'SRQ');
      srq := true;
    end;
    if (((port[$316] and $10) = $10) and (srq=true)) then
    begin
      write(#7);
      setcolor(15),
      outtextxy(580,29,'SRQ');
      srq := false,
    end;
  end;
```

```
  Inline($FB); { STI enable interrupts }
end;

procedure listen_data_var_par;
const varia : array[1..3] of string[1] = ('A','B','C');
      param : array[1..3] of string[1] = ('D','E','F');
var h : byte;
begin
  frame(8,310,429,471,4,0,15,0,'');
  frame(13,330,424,380,7,0,15,1,'');
  textbox(13,315,424,325,15,0,'Listen Data');
  textbox(13,385,214,395,15,0,'Variable');
  textbox(224,385,424,395,15,0,'Parameter');
  frame(13,400,214,465,7,0,15,1,'');
  frame(224,400,424,465,7,0,15,1,'');
  for h := 1 to 5 do
  begin
    if ((h mod 2) <> 0) then
    begin
      frame(h*32,422,31+h*32,442,7,4,0,0,varia[(h+1) div 2]);
      frame(211+h*32,422,242+h*32,442,7,4,0,0,param[(h+1) div 2]);
    end;
  end;
  frame(var_index*32,422,31+var_index*32,442,12,15,0,0,
        varia[(var_index+1) div 2]);

end;

procedure display_menubar(p : byte);
var i : integer;
begin
  setfillpattern(Gray,3);
```

```
  bar(7,41,429,309);
  for i := 0 to 4 do
    textbox(9+i*70,25,78+i*70,39,15,0,menubar[i]);
  textbox(9+p*70,25,78+p*70,39,3,15,menubar[p]);
  frame(430,41,631,471,4,0,15,0,'');
  frame(435,60,626,395,7,0,15,1,'');
  frame(435,400,626,465,7,0,15,1,'');
  textbox(480,45,581,55,15,0, 'GPIB Device');
  textbox(480,410,581,420,7,0,'Current Device :     ');
  textbox(480,445,581,455,7,0,'Address :           ');
  listen_data_var_par;
end;

procedure display_sub_quit;
var i: integer;
begin
  for i := 0 to 1 do
    textbox(10,42+i*14,159,55+i*14,15,0,sub_menu_quit[i]);
  setcolor(0);
  rectangle(9,41,160,70);
  textbox(10,42,159,55,3,15,sub_menu_quit[0]);
end;

procedure display_sub_GPIB;
var i: integer;
begin
  for i := 0 to 1 do
    textbox(150,42+i*14,299,55+i*14,15,0,sub_menu_GPIB[i]);
  setcolor(0);
  rectangle(149,41,300,70);
  textbox(150,42,299,55,3,15,sub_menu_GPIB[0]);
end;
```

```
procedure display_sub_ptr_scrn(p : byte);
var i . integer;
begin
 for i := 0 to 1 do
   textbox(220,42+i*14,369,55+i*14,15,0,sub_menu_print_scrn[i]);
 setcolor(0);
 rectangle(219,41,370,70);
 if p=0 then
    textbox(220,42,369,55,3,15,sub_menu_print_scrn[0])
 else
    textbox(220,56,369,69,3,15,sub_menu_print_scrn[1])
end;

procedure display_sub_option;
begin
 for i := 0 to 2 do
   textbox(80,42+i*14,229,55+i*14,15,0,sub_menu_option[i]);
 setcolor(0);
 rectangle(79,41,230,84);
 textbox(80,42,229,55,3,15,sub_menu_option[0]);
end;

procedure hide_sub_print;
begin
 setfillpattern(Gray,3);
 bar(219,41,370,70);
end;

procedure hide_sub_quit;
begin
 setfillpattern(Gray,3);
```

```
    bar(9,41,160,70);
  end;

  procedure hide_sub_option;
  begin
    setfillpattern(Gray,3);
    bar(79,41,230,84);
  end;

  procedure hide_sub_GPIB;
  begin
    setfillpattern(Gray,3);
    bar(149,41,300,70);
  end;

  procedure os_shell;
  var i : byte;
      cs:string;
  begin
    closegraph;
    clrscr;
    writeln('Type EXIT to return to GPIB Program');
    repeat
      getdir(0,cs);
      Write(cs,'>');
      ReadLn(command);
      for i := 1 to Length(command) do
        command[i] := UpCase(command[i]);
      if ((command <> '') and (command <> 'EXIT')) then
      begin
        command := '/C ' + command;
        SwapVectors;
```

```
      Exec(GetEnv('COMSPEC'),command);
      SwapVectors;
      if DosError <> 0 then
      begin
        WriteLn('Could not execute COMMAND.COM Error=',DosError);
        WriteLn(MemAvail, ' bytes available');
        WriteLn('Largest free block is ', MaxAvail, ' bytes');
      end;
    end;
  until command='EXIT';
  opengraph;
  win(0,0,getmaxx,getmaxy,7,'IEEE 488 GPIB');
  display_menubar(0);
  for i := 0 to 1 do
    textbox(10,42+i*14,159,55+i*14,15,0,sub_menu_quit[i]);
  setcolor(0);
  rectangle(9,41,160,70);
  textbox(10,42,159,55,3,15,sub_menu_quit[0]);
  if device_num > 0 then
  begin
    for i := 1 to de_num do
    begin
      textbox(436,70+20*i,625,89+20*i,
              7,0,device_name[i]);
    end;
    textbox(436,70+20*curr_num,625,89+20*curr_num,3,15,device_name[curr_num]),
    textbox(480,425,581,435,7,15,device_name[curr_num]),
    textbox(530,445,560,455,7,15,IntToStr(address[curr_num]));
  end,
end;
```

```
function exist(file_name  string) : boolean;
var f : file;
begin
  assign(f,file_name);
  {$I-} reset(f); {$I+}
  if ioresult=0 then
  begin
    close(f);
    exist := true;
  end
  else exist := false;
end;

procedure message_error;
begin
  frame(70,60,420,111,7,0,15,0,'');
  textbox(170,65,320,75,15,0,'ERROR !!!');
  frame(75,80,415,106,15,0,15,1,'File Not Found');
  read_func_key(c);
  setfillpattern(Gray,3);
  bar(70,60,420,111);
end;

procedure print_to_printer;
var pname,ppname,name : string;
    printout : text;
    dataout  : string;
    l : byte;

 procedure print_win;
 begin
    frame(70,60,420,111,7,0,15,0,'');
```

```
    textbox(170,65,320,75,15,0,'Print File Name');
    frame(75,80,415,106,15,0,15,1,'');
    readstring(78,82,pname,38,'s');
    setfillpattern(Gray,3);
    bar(70,60,420,111);
    display_sub_ptr_scrn(0);
  end;

 procedure select_file;
 begin
   readdir(50,100,pname,ppname,name);
   frame(430,41,631,471,4,0,15,0,'');
   frame(435,60,626,395,7,0,15,1,'');
   frame(435,400,626,465,7,0,15,1,'');
   textbox(480,45,581,55,15,0, 'GPIB Device');
   textbox(480,410,581,420,7,0,'Current Device :     ');
   textbox(480,445,581,455,7,0,'Address :          ');
   if device_num > 0 then
   begin
     for i := 1 to de_num do
     begin
       textbox(436,70+20*i,625,89+20*i,
            7,0,device_name[i]);
     end;
     textbox(436,70+20*curr_num,625,89+20*curr_num,3,15,device_name[curr_num]);
     textbox(480,425,581,435,7,15,device_name[curr_num]);
     textbox(530,445,560,455,7,15,IntToStr(address[curr_num]));
   end;
 end;

procedure print_data;
var q . boolean;
```

```
begin
 q := false;
 assign(printout,pname);
 reset(printout);
 frame(70,60,420,121,7,0,15,0,'');
 textbox(170,65,320,75,15,0,'Printing');
 frame(75,80,415,100,15,0,15,1,'Printer data from');
 frame(75,102,415,116,15,0,15,1,fexpand(pname));
repeat
  if not EOF(printout) then
  begin
      readln(printout,dataout);
{$I-} writeln(LST,dataout); {$i+}
      if ioresult<>0 then
      begin
       frame(70,60,420,121,7,0,15,0,'');
       textbox(170,65,320,75,15,0,'ERROR !!!');
       frame(75,80,415,100,15,0,15,1,'Printer Not Ready');
       frame(75,102,415,116,15,0,15,1,
            'Press R to retry : Any key to cancel');
      read_func_key(c);
      setfillpattern(Gray,3);
      bar(70,60,420,121);
      if upcase(c) <>'R' then q := true
      else
      begin
       frame(70,60,420,121,7,0,15,0,'');
       textbox(170,65,320,75,15,0,'Printing');
       frame(75,80,415,100,15,0,15,1,'Printer data from');
       frame(75,102,415,116,15,0,15,1,fexpand(pname));
      end;
    end;
```

```
      end;
    until (keypressed or (q) or (EOF(printout)));
    display_sub_ptr_scrn(0);
    c := ' ';
    setfillpattern(Gray,3);
    bar(70,60,420,121);
   'close(printout),
   end;

begin
   print_win;
   if ((pname <> Return_key) and (pname <> Esc_key)) then
   begin
     if exist(pname) then
       print_data
     else
     begin
        select_file;
        if name<>Esc_key then
        begin
          pname := ppname+name;
          if exist(pname) then print_data
          else
            if pname<>Esc_key then
               message_error;
        end;
     end,
   end;
  display_sub_ptr_scrn(0);
end;

{$I c:\tp55\pas\fiscrn.pas}
```

```
procedure print_to_screen,
var dname,name,ddname : string;

  procedure select_file;
  var i : integer;
  begin
    readdir(50,100,dname,ddname,name);
    frame(430,41,631,471,4,0,15,0,'');
    frame(435,60,626,395,7,0,15,1,'');
    frame(435,400,626,465,7,0,15,1,'');
    textbox(480,45,581,55,15,0, 'GPIB Device');
    textbox(480,410,581,420,7,0,'Current Device :      ');
    textbox(480,445,581,455,7,0,'Address :          ');
    if device_num > 0 then
    begin
      for i := 1 to de_num do
      begin
        textbox(436,70+20*i,625,89+20*i,
              7,0,device_name[i]);
      end;
      textbox(436,70+20*curr_num,625,89+20*curr_num,3,15,device_name[curr_num]),
      textbox(480,425,581,435,7,15,device_name[curr_num]);
      textbox(530,445,560,455,7,15,IntToStr(address[curr_num]));
    end;
  end;

procedure display_win;
begin
  frame(70,60,420,111,7,0,15,0,'');
  textbox(170,65,320,75,15,0,'Display File Name');
  frame(75,80,415,106,15,0,15,1,'');
```

```
    readstring(78,82,dname,38,'s');
    setfillpattern(Gray,3);
    bar(70,60,420,111);
    display_sub_ptr_scrn(1);
  end;

  procedure display_file;
  begin
    frame(7,41,631,471,3,0,15,0,'');
    frame(13,46,626,466,7,0,15,1,'');
    read_data_from_file(2,47,dname);
    frame(430,41,631,471,4,0,15,0,'');
    frame(435,60,626,395,7,0,15,1,'');
    frame(435,400,626,465,7,0,15,1,'');
    textbox(480,45,581,55,15,0, 'GPIB Device');
    textbox(480,410,581,420,7,0,'Current Device .    ');
    textbox(480,445,581,455,7,0,'Address .          ');
    listen_data_var_par;
  end;

  procedure redraw;
  var i : integer;
  begin
    display_menubar(3);
    if device_num > 0 then
    begin
      for i := 1 to de_num do
      begin
        textbox(436,70+20*i,625,89+20*i,
            7,0,device_name[i]);
      end;
      textbox(436,70+20*curr_num,625,89+20*curr_num,3,15,device_name[curr_num]),
```

```
    textbox(480,425,581,435,7,15,device_name[curr_num]);
    textbox(530,445,560,455,7,15,IntToStr(address[curr_num]));
  end;
  display_sub_ptr_scrn(1);
end;

begin
 'display_win;
 if ((dname <> Return_key) and (dname <> Esc_key)) then
 begin
  if exist(dname) then
  begin
    display_file;
    display_sub_ptr_scrn(1);
  end
  else
  begin
     select_file;
     if name<>Esc_key then
     begin
      dname := ddname+name;
      if exist(dname) then
      begin
        display_file;
        display_sub_ptr_scrn(1);
      end
      else
        if dname<>Esc_key then
        begin
          message_error;
          display_sub_ptr_scrn(1);
        end;
```

```
        end;
      end;
    end;
  end;

 procedure chk_sub_quit_no(no : integer);
 begin
   case no of
   0 : os_shell;
   1 : quit := true;
   end;
 end;

{$I c:\tp55\pas\gpib.pas}

{$I c:\tp55\pas\timedate.pas}

procedure program_code;
var pname,pgc : string;
    filedata : text;
begin
 frame(25,200,429,251,4,15,0,0,'');
 frame(30,220,424,246,15,15,0,1,'');
 textbox(65,205,399,215,15,0,'Program Code Sequence (max 45 char)');
 readstring(32,222,pgc,45,'s');
 if (pgc <> Return_key) then
 begin
  if (pgc <> Esc_key) then
  begin
    setfillpattern(Gray,3);
    bar(25,200,429,251),
    frame(70,160,420,211,7,0,15,0,'');
```

```
      textbox(100,165,390,175,15,0,'Save Program Code To File Name');
      frame(75,180,415,206,15,0,15,1,'');
      readstring(78,182,pname,38,'s');
      setfillpattern(Gray,3);
      bar(70,160,420,211);
      if (pname <> Esc_key) then
      begin
       if exist(pname) then
       begin
         assign(filedata,pname);
         reset(filedata);
         close(filedata);
         append(filedata);
       end
       else
       begin
         assign(filedata,pname);
         rewrite(filedata);
       end;
       write(filedata,pgc);
       close(filedata);
     end;
   end;
  end;
  setfillpattern(Gray,3);
  bar(25,200,429,251);
end;

procedure chk_sub_option_no(no  integer);
begin
 case no of
  0 : set_time(hh,mm,ss);
```

```
    1  set_date;
    2  program_code;
  end;
end;

procedure chk_sub_GPIB_no(no . integer);
begin
  case no of
   0 · configure;
   1   interac;
  end;
end;

procedure chk_sub_print_no(no : integer);
begin
  case no of
   0  print_to_printer;
   1  print_to_screen;
  end;
end;

{$I c:\tp55\pas\plotgph.pas}

procedure execute_command(menu_no,sub_no : integer),
begin
  case menu_no of
   0  chk_sub_quit_no(sub_no);
   1  chk_sub_option_no(sub_no);
   2  chk_sub_GPIB_no(sub_no);
   3  chk_sub_print_no(sub_no);
   4  plot,
  end,
```

```
end;

procedure check_sub_menu;
begin
 case menu_num of
 0 : begin
      display_sub_quit;
      choice := 1;
     end;
 1 : begin
      display_sub_option;
      choice := 2;
     end;
 2 : begin
      display_sub_GPIB;
      choice := 1;
     end;
 3 : begin
      display_sub_ptr_scrn(0);
      choice := 1;
     end;
 end;
end;


procedure select_sub_menu;
var i  integer;
   temp  string;
begin
 i := 0;
 repeat
```

```
repeat
 read_func_key(c);
  case c of
   Dn_key . begin
            if menu_num <> 4 then
            begin
             case menu_num of
             0 : temp := sub_menu_quit[i];
             1 : temp := sub_menu_option[i];
             2 : temp := sub_menu_GPIB[i];
             3 : temp := sub_menu_print_scrn[i];
            end;
            textbox(10+70*menu_num,42+i*14,159+70*menu_num,
                  55+i*14,15,0,temp);
           i := i+1;
           if i>choice then i := 0;
           case menu_num of
           0 . temp := sub_menu_quit[i];
           1 : temp := sub_menu_option[i];
           2 : temp := sub_menu_GPIB[i];
           3 : temp := sub_menu_print_scrn[i];
          end;
          textbox(10+70*menu_num,42+i*14,159+70*menu_num,
                55+i*14,3,15,temp);
         end;
        end;
  Up_key   begin
          if menu_num <> 4  then
          begin
           case menu_num of
           0   temp := sub_menu_quit[i],
           1   temp  = sub_menu_option[i];
```

```
        2  temp := sub_menu_GPIB[i];
        3 : temp := sub_menu_print_scrn[i];
       end;
       textbox(10+70*menu_num,42+i*14,159+70*menu_num,
            55+i*14,15,0,temp);
       i := i-1;
       if i<0 then i := choice;
       case menu_num of
        0  temp := sub_menu_quit[i];
        1  temp := sub_menu_option[i];
        2  temp := sub_menu_GPIB[i];
        3  temp := sub_menu_print_scrn[i];
       end;
       textbox(10+70*menu_num,42+i*14,159+70*menu_num,
            55+i*14,3,15,temp);
      end;
     end;

Rt_key  begin
       case menu_num of
        0 : hide_sub_quit;
        1  hide_sub_option;
        2 : hide_sub_GPIB;
        3  hide_sub_print;
       end;
       textbox(9+menu_num*70,25,78+menu_num*70,39,15,0
            ,menubar[menu_num]);
       menu_num := menu_num+1;
       if menu_num > 4 then menu_num := 0;
       textbox(9+menu_num*70,25,78+menu_num*70,39,3,15
            ,menubar[menu_num]),
       check_sub_menu,
```

```
            i := 0;
        end;
    Lt_key : begin
            case menu_num of
            0 : hide_sub_quit;
            1 : hide_sub_option;
            2 : hide_sub_GPIB;
            3 : hide_sub_print;
            end;
            textbox(9+menu_num*70,25,78+menu_num*70,39,15,0
                ,menubar[menu_num]);
            menu_num := menu_num-1;
            if (menu_num < 0) then menu_num := 4;
            textbox(9+menu_num*70,25,78+menu_num*70,39,3,15
                ,menubar[menu_num]);
            check_sub_menu;
            i := 0;
        end;
  end;
 until c = #13;
 execute_command(menu_num,i);
until quit;
init_card;
IFC;
DCL;
send_command(address[curr_num],UNL);
send_command(address[curr_num],GTL);
setintvec(timer,vector);
end;

procedure selectmenu;
var i : integer;
```

```
begin
 i := 0;
 menu_num := 0;
 repeat
  read_func_key(c);
  case c of
   #77 . begin
          textbox(9+i*70,25,78+i*70,39,15,0,menubar[i]);
          i := i+1;
          if i>4 then i := 0;
          textbox(9+i*70,25,78+i*70,39,3,15,menubar[i]);
          menu_num := i;
         end;
   #75  begin
          textbox(9+i*70,25,78+i*70,39,15,0,menubar[i]);
          i := i-1,
          if i<0 then i := 4;
          textbox(9+i*70,25,78+i*70,39,3,15,menubar[i]);
          menu_num := i;
         end;
  end;
 until c = #13;
 if menu_num <> 4 then
 begin
  check_sub_menu;
  select_sub_menu;
 end
 else
 begin
  plot,
  select_sub_menu;
 end.
```

```
end;

{$I c:\tp55\pas\space.pas }

procedure initial_program;
begin
  quit := false;
  srq := false;
  de_num := 0;
  device_num := 0;
  curr_num := 0;
  aindex := 0;
  bindex := 0;
  cindex := 0;
  var_index := 1;
  chksrq := 0;
  for i := 1 to 30 do
  begin
    device_name[i] := '';
    adr[i] := '';
  end;
  space;
  c := readkey;
  c := ' ';
  setfillstyle(0,0);
  bar(0,0,getmaxx,getmaxy);
{
  Init_card,
  IFC,
  DCL,
```

```
  if bus_error then
    begin
      closegraph;
      writeln('Please check bus now before run GPIB program !!!');
      halt;
    end; }
end;

begin {main}
  opengraph;
  initial_program;
  win(0,0,getmaxx,getmaxy,3,'IEEE 488 GPIB');
  display_menubar(0);
  getintvec(timer,vector);
  setintvec(timer,@chk_SRQ);
  selectmenu;
  closegraph;
end.
```

```
unit IEEE488;
{$O+,F+}
interface

uses crt,menu,graph;

const
    Gray  : FillPatternType = ($CA, $55, $CA,
          $55, $C5, $5C, $CA, $55);
    device_addr_MLA : array[0..30] of byte =
    ($20,$21,$22,$23,$24,$25,$26,$27,$28,$29,$2A,$2B,$2C,$2D,$2E,$2F,
     $30,$31,$32,$33,$34,$35,$36,$37,$38,$39,$3A,$3B,$3C,$3D,$3E);

    device_addr_MTA : array[0..30] of byte =
    ($40,$41,$42,$43,$44,$45,$46,$47,$48,$49,$4A,$4B,$4C,$4D,$4E,$4F,
    $50,$51,$52,$53,$54,$55,$56,$57,$58,$59,$5A,$5B,$5C,$5D,$5E);

    GTL = $01;   { Go To Local          }
    SDC = $04;   { Select Device Clear  }
    PPC = $05;   { Parallel Poll Configure }
    GET = $08;   { Group Execute Trigger  }
    TCT = $09,   { Take Control         }
    LLO = $11;   { Local Lock Out       }
    PPU = $15;   { Parallel Poll Unconfigure }
    SPE = $18;   { Serial Poll Enable   }
    SPD = $19,   { Serial Poll Disable  }
    UNL = $3F;   { Unlisten             }
    UNT = $5F;   { Untalk               }

type SPLdata = array[1..14] of byte,
     address_array = array[1..30] of byte;
```

```
var   MLA,MTA     byte, { My Listen Address and My Talk Address }
      bus_error . boolean;
      databyte    byte;
      data_temp · array[0..1023] of real;
      address     address_array;

      { Port A    =  $310
        Port B    =  $311
        Port C1   =  $312
        Control 1 =  $313
        Port C2   =  $316
        Control 2 =  $317 }

procedure Init_card;
procedure Init_set_device(addr · integer);
procedure IFC;
procedure DCL;
procedure Group_Execute_Trigger(addr : integer);
procedure Go_To_Local(addr : integer);
procedure Local_Lock_Out;
procedure Select_Device_Clear(addr : integer);
procedure send_command(addr . integer; command : byte);
procedure send_message(addr : integer; m · string;eoi : char);
procedure get_data(addr · byte,var d   string; point : integer;lim   byte;
                   eos   char; times   integer);
procedure Spolling(start_dev,stop_dev : integer; var one_byte : SPLdata);
procedure Spolling1byte(addr · integer; var databyte : byte);
procedure get_data1byte(addr : byte;var d : string; eos : char);

implementation

function string_data(dat . string) · string;
```

```
var i  byte,
begin
  i := 0;
  while ((i<length(dat)) and (dat[i]<>#10)) do
  begin
     inc(i);
  end;
  if dat[i]=#10 then delete(dat,i,1);
  i := 0;
  while ((i<length(dat)) and (dat[i]<>#13)) do
  begin
     inc(i);
  end;
  if dat[i]=#13 then delete(dat,i,1);
  string_data := dat;
end,

procedure buserror;
var i,co : integer;
   ch . char;
begin
  frame(90,210,410,290,7,15,0,0,' ');
  frame(100,220,400,280,7,15,0,1,' ');
  for i := 0 to 100 do
  begin
    if (i mod 2) = 0 then co := 0 else co := 15;
    putpixel(101+random(298),221+random(78),co);
  end;
  repeat
    textstyle(0,6,0,180,225,'BUS ERROR');
    delay(350),
    textstyle(3,6,0,180,225,'BUS ERROR');
```

```
    delay(350);
  until keypressed,
  ch := readkey;
  setfillpattern(Gray,3);
  bar(90,210,410,290);
  settextstyle(0,0,0);
end;

procedure addr_to_HEX (addr : integer ; var MLA,MTA : byte);
begin
    MLA := device_addr_MLA[addr];
    MTA := device_addr_MTA[addr],
end;

procedure source_operation(data : byte);
begin
  port[$312] := (port[$312] or $0C);    {make DAV = H EOI = H}
  if ((port[$312] or $CF) = $FF) then   {check NRFD = NDAC = H}
  begin
    bus_error := true;
    buserror,
  end
  else
  begin
    bus_error := false;
    port[$310] := $FF - data;    { put data on DIO line }
    repeat
    until ((port[$312] or $EF)=$FF) or keypressed; { check NRFD = H }
    port[$312] := (port[$312] and $F7); { make DAV = L }
    repeat
    until ((port[$312] or $DF) = $FF) or keypressed, { check NDAV = H }
    port[$312] := (port[$312] or $0C); { make DAV = H EOI = H }
```

```
  end.
end;

procedure Init_card;
begin
  port[$317] := $88;  { control up 2 = i/p, control low 2 = o/p}
  port[$313] := $88,  { A1 = output , B1 = o/p ,Cup 1 = i/p ,Clow 1 = o/p }
  port[$311] := $0B;
  port[$310] := $FF,  { clear data on DIO lines }
end;

procedure send_command(addr . integer; command . byte);
begin
  port[$316] := $02;  { make REN = L IFC = H ATN = L }
  addr_to_HEX(addr,MLA,MTA);
  source_operation(MLA);
  if bus_error then
  else source_operation(command);
end,

procedure send_message(addr : integer; m : string;eoi : char);
var msage : char;
    i  integer;
begin
  port[$316] := $02,   { make REN = L IFC = H ATN = L }
  addr_to_HEX(addr,MLA,MTA);
  source_operation(MLA);
  if bus_error then buserror
  else
  begin
    port[$316] := $03,   { make REN = L IFC = H ATN = H }
    for i := 1 to length(m) do
```

```
    begin
      msage := m[i];
      source_operation(integer (msage));
    end;
   { source_operation($0D); }{send CR}
   { source_operation($0A);  }{send LF}
    if eoi='y' then
    begin
      port[$312] := (port[$312] or $0C);
      port[$312] := (port[$312] and $0B);
    end
    else port[$312] := (port[$312] or $0C);
  end;
end;


procedure acceptor_operation(var databyte : byte);
begin
  port[$312] := $D0;     { make NDAC = L NRFD = H }
  repeat
  until ((port[$312] and $08) <> $08) or keypressed; { check DAV = L }
  port[$312] := $C0;     { make NDAC=L NRFD=L }
  databyte := $FF - port[$310]; { accept data 1 byte }
  port[$312] := $E0;     { make NDAC=H NRFD=L }
  repeat
  until ((port[$312] and $08) = $08) or keypressed; { check DAV = H }
  port[$312] := $C0;
end;

procedure get_data(addr . byte;var d : string; point . integer;lim : byte;
                   eos . char, times  integer);
var i,b,code : integer;
```

```
begin
  port[$316] := $02;       { make REN = L IFC = H ATN = L }
  addr_to_HEX(addr,MLA,MTA);
  source_operation(MTA);
  if bus_error then
  else
  begin
    port[$313] := $91;     { A1 i/p B1 o/p Cup o/p Clow o/p }
    port[$311] := $00;     { 160 = R DAV = R NRFD = NDAC = T }
    port[$312] := $C0;     { make NRFD = L NDAC = L }
    port[$316] := $03;     { make REN = L IFC = H ATN = H }
    d  = '',               { clear data }
    if lim=1 then
    begin
      for i := 1 to point do
      begin
        acceptor_operation(databyte),
        d := d+char(databyte),
      end;
      repeat
        acceptor_operation(databyte);
      until (((port[$312] and $04) <> $04) or ((char(databyte) = eos))
              or (char(databyte)=#10)),
    end,
    else,
    begin
      if times=0 then
      begin
        repeat
          acceptor_operation(databyte);
          d  = d+char(databyte),
        until(((port[$312] and $04) <> $04) or ((char(databyte) = eos)),
```

```
            or (char(databyte)=#10) or keypressed);
      end
      else
      begin
        b := 0;
        repeat
          d := '';
          repeat
            acceptor_operation(databyte);
            d := d+char(databyte);
          until(((port[$312] and $04) <> $04) or ((char(databyte) = eos))
                or (char(databyte)=#10) or keypressed);
          d := string_data(d);
          val(d,data_temp[b],code);
          inc(b);
        until b > times;
      end;
    end;
    port[$313] := $88;      { A1 o/p B1 o/p Cup1 i/p Clow1 o/p }
    port[$311] := $0B;      { 160 = T DAV = T NRFD = NDAC = R }
    port[$316] := $02;      { make REN = L IFC = H ATN = L }
    source_operation(UNT);
  end;
end;

procedure get_data1byte(addr : byte;var d . string; eos : char);
begin
  port[$316] := $02;        { make REN = L IFC = H ATN = L }
  addr_to_HEX(addr,MLA,MTA);
  source_operation(MTA);
  if bus_error then
  else
```

```
  begin
    port[$313] := $91,     { A1 i/p B1 o/p Cup o/p Clow o/p }
    port[$311] := $00;     { 160 = R DAV = R NRFD = NDAC = T }
    port[$312] := $C0;     { make NRFD = L NDAC = L }
    port[$316] := $03;     { make REN = L IFC = H ATN = H }
    d := '';               { clear data }
    repeat
      acceptor_operation(databyte);
      d := d+char(databyte);
    until(((port[$312] and $04) <> $04) or ((char(databyte) = eos))
          or (char(databyte)=#10) or keypressed);
    d := string_data(d);
    port[$313] := $88;     { A1 o/p B1 o/p Cup1 i/p Clow1 o/p }
    port[$311] := $0B;     { 160 = T DAV = T NRFD = NDAC = R }
    port[$316] := $02;     { make REN = L IFC = H ATN = L }
    source_operation(UNT);
  end,
end;

procedure IFC;
begin
  port[$316] := $00;       { make REN = L IFC = L ATN = L }
end;

procedure DCL,
begin
  port[$316] := $02;       { make REN = L IFC = H ATN = L }
  source_operation($14);
end,

procedure Group_Execute_Trigger(addr : integer),
begin
```

```
  send_command(addr,GET),
end,

procedure Go_To_Local(addr . integer);
begin
  send_command(addr,GTL),
end;

procedure Local_Lock_Out;
begin
  source_operation(LLO);
end,

procedure Select_Device_Clear(addr : integer);
begin
  send_command(addr,SDC);
end;

procedure Init_set_device(addr : integer);
begin
  addr_to_HEX(addr,MLA,MTA);
  source_operation(MLA);
end;

procedure Spolling(start_dev,stop_dev  integer; var one_byte . SPLdata),
var b . byte;
begin
  b := start_dev;
  port[$316] := $02;       { make REN = L IFC = H ATN = L }
  addr_to_HEX(address[b],MLA,MTA);
  source_operation(UNL);
  if bus_error then
```

```
  else
  begin
    source_operation(UNT);
    source_operation(SPE);
    repeat
      if b <> start_dev then
      begin
        addr_to_HEX(address[b],MLA,MTA);
        source_operation(UNL);
        source_operation(UNT);
      end;
      source_operation(MTA);
      port[$313] := $91;     { A1 i/p B1 o/p CU1 o/p CL1 i/p }
      port[$311] := $00;     { 160=R; DAV=R; NRFD=NDAC=T    }
      port[$312] := $C0;     { make NDAC = L NRFD = L       }
      port[$316] := $03;     { make REN = L IFC = H ATN = H  }
      acceptor_operation(one_byte[b]);
      port[$313] := $88;     { A1 o/p B1 o/p CU1 i/p CL1 o/p }
      port[$311] := $0B;     { 160=T; DAV=T; NRFD=NDAC=R }
      port[$316] := $02;     { make REN = L IFC = H ATN = L  }
      port[$312] := $0C;     { make DAV = H EOI = H         }
      inc(b);
    until b = stop_dev+1;
    source_operation(SPD);
    source_operation(UNT);
  end;
end;

procedure Spolling1byte(addr : integer; var databyte : byte);
begin
  port[$316] := $02;       { make REN = L IFC = H ATN = L }
  addr_to_HEX(addr,MLA,MTA);
```

```
    source_operation(UNL);
    if bus_error then
    else
    begin
     source_operation(UNT);
     source_operation(SPE);
     source_operation(MTA);
     port[$313] := $91;    { A1 i/p B1 o/p CU1 o/p CL1 i/p }
     port[$311] := $00;    { 160=R; DAV=R; NRFD=NDAC=T    }
     port[$312] := $C0;    { make NDAC = L NRFD = L       }
     port[$316] := $03;    { make REN = L IFC = H ATN = H  }
     acceptor_operation(databyte);
     port[$313] := $88;    { A1 o/p B1 o/p CU1 i/p CL1 o/p }
     port[$311] := $0B;    { 160=T; DAV=T; NRFD=NDAC=R }
     port[$316] := $02;    { make REN = L IFC = H ATN = L  }
     port[$312] := $0C;    { make DAV = H EOI = H         }
     source_operation(SPD);
     source_operation(UNT);
   end;
end;

begin
  bus_error := false;
  port[$317] := $88;
end;  { IEEE488 }
```