ปีการศึกษา 2533

การเชื่อมต่อข้อมูลแบบขนาน

Parallel Interfacing Bus

โดย

นาย บุญยัง เอี่ยมกมล

นาย ปริญญา วิจิตรกาญจน์

อาจารย์ที่ปรึกษา

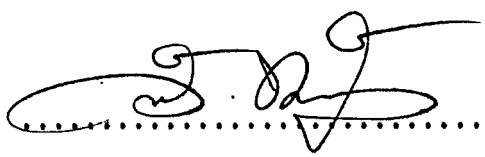
อาจารย์ วิทยา ทิพย์สุวรรณพร

ปริญญานิพนธ์ปีการศึกษา 2533

เรื่อง การเชื่อมต่อข้อมูลแบบขนาน (Parallel Interfacing Bus)

ผู้จัดทำ

1. นาย บุญยัง เอี่ยมกมล
2. นาย ปริญญา วิจิตรกาญจน์



..... อาจารย์ที่ปรึกษา

(อาจารย์ วิทยา ทิพย์สุวรรณพร)

ระบบบัสเชื่อมต่อข้อมูลแบบขนาน

โดย

นาย บุญยัง เอี่ยมกมล

นาย ปริญญา วิจิตรกาญจน์

อาจารย์ที่ปรึกษา

อาจารย์ วิทยา ทิพย์สุวรรณพร

ปีการศึกษา 2533

บทคัดย่อ

ในการติดต่อส่งข้อมูลเรามักจะคุ้นเคยกับระบบ การเชื่อมโยงแบบอนุกรม หรือ การเชื่อมโยงแบบขนาน แต่เมื่อไรก็ตามที่เราต้องใช้ระบบในการควบคุมเครื่องมือ เครื่องใช้ทางอิเลคทรอนิคส์แล้วละก็ เราจะพบว่าระบบบัสเชื่อมโยงเอนกประสงค์ (GPIB = General Purpose Interface Bus) เป็นระบบที่นิยมใช้กันมากในเครื่องมือวัดทางอิเลคทรอนิคส์ แต่ส่วนใหญ่แล้วเราไม่คุ้นเคยกับระบบนี้ดี ก็เป็นเพราะส่วนใหญ่จะถูกใช้ในเครื่องมือวัดที่มีราคาแพง หรือถูกใช้ในโรงงานอุตสาหกรรมเป็นส่วนใหญ่ ดังนั้นงานชิ้นนี้จึงถูกเสนอขึ้นมาเพื่อศึกษาถึงระบบการเชื่อมโยงต่อข้อมูล และการคอนโทรนนี้อย่างจริงจัง

ระบบบัสเชื่อมต่อข้อมูลแบบขนาน

โดย

นาย บุญยัง เอี่ยมกมล

นาย ปริญญา วิจิตรกาญจน์

อาจารย์ที่ปรึกษา

อาจารย์ วิทยา ทิพย์สุวรรณพร

ปีการศึกษา 2533

Preface

In communication system, almost we must say to " Serial communication system and Parallel communication system " When ever if you want to control electronice instrument. We must say to " GPIB = General Purpose Interface Bus " It is similar to parallel bus more trouble. The almost expensive instrument must be installed GPIB because It's high speed communication and GPIB standard specifices that maximum of 15 revices including instrument. We think electronic peoples in Thailand don't know it well. So this project was studied by my group.

สารบัญ

บทที่ 1

บทนำ

ระบบบัสเชื่อมต่อข้อมูลเอนกประสงค์เป็น ระบบบัสมาตรฐานที่ใช้ในการติดต่อรับส่งข้อมูลระหว่างอุปกรณ์ มาตรฐานนี้ได้ผ่านการรับรองจากสถาบันวิศวกรไฟฟ้า และอิเล็คทรอนิคส์ (Institute of Electrical and Electronics Engineers : IEEE) แห่งประเทศสหรัฐอเมริกา และถูกจัดเข้าเป็นลำดับที่ IEEE-488 ในปี ค.ศ. 1975 และได้มีการปรับปรุงมาตรฐานให้ดีขึ้นในภายหลังเป็น IEEE-488 (1978) ซึ่งได้ใช้มาจนถึงปัจจุบันนี้ แต่เนื่องจากในคุณลักษณะสมบัติของ IEEE-488 นี้ มีความหมายถึงบัสเชื่อมต่อข้อมูลที่ใช้งานได้ทั่วไปจึงนิยมใช้คำว่า GPIB (General Purpose Interface Bus)

ประวัติความเป็นมา IEEE-488

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

บริษัทฮิวเล็ตแพกการ์ด ผู้ผลิตเครื่องมือวัดรายใหญ่รายหนึ่งในอเมริกา ได้ทำการพัฒนาระบบบัสมาตรฐานของตัวเองอยู่ก่อนแล้วชื่อว่า HPIB (Hewlett Packard Interface Bus) จึงได้เสนอระบบบัสของตัวเองให้ IEEE พิจารณาและได้รับการยอมรับในที่สุดในปี 1975 เรียกว่ามาตรฐาน IEEE Std 488-1975 และมีการปรับปรุงต่อมาเป็น IEEE Std 488-1978 ระบบบัสนี้ก็คือ IEEE 488 ที่กล่าวถึงนั่นเอง สำหรับของ IEC ก็ได้มีการกำหนดมาตรฐานขึ้นมาอีกอันหนึ่งเรียกว่า IEC 625-1 ตั้งใจที่จะให้เป็นมาตรฐานสากล โดยมีรายละเอียดทางเทคนิคทุกประการเหมือนกับ IEEE Std 488-1978 เพียงแต่การวางตำแหน่งของสัญญาณต่าง ๆ ในขั้วต่อต่างกันเล็กน้อยเท่านั้นเอง คำว่า GPIB จึงหมายรวมถึงทั้ง 2 มาตรฐานดังกล่าว

จุดประสงค์ของบัสก็คือ การส่งข่าวสารข้อมูลถ่ายทอดกันระหว่างเครื่องข่าวสารที่ว่านี้ก็มีทั้งข้อมูลที่เป็นผลลัพธ์จากการวัดค่าหรือจากกระบวนการต่าง ๆ และรวมถึงคำสั่งที่ใช้กำหนดสภาวะทำงานของอุปกรณ์ต่าง ๆ ในระบบ

เมื่อมีการต่อเครื่องมือวัดหรืออุปกรณ์ที่ต้องการใช้ร่วมกันเข้ากับระบบบัสแล้ว จำเป็นต้องมีแบบแผนในการทำงานร่วมกันให้สอดคล้องกัน จึงต้องมีสัญญาณสำหรับควบคุมระบบ นอกเหนือจากข่าวสารข้อมูลที่ต้องมีการติดต่อกัน (สัญญาณควบคุมนี้ ใช้สำหรับควบคุมบัส และจัดระบบสำหรับการอินเตอร์เฟส อาจมองได้ว่าเป็นความเกี่ยวข้องทางฮาร์ดแวร์ไม่เกี่ยวข้องกับผู้ใช้งานภายนอก ส่วน คำสั่ง สำหรับกำหนดการทำงานของอุปกรณ์ในระบบเป็นงานทางด้านซอฟต์แวร์ที่เกี่ยวข้องกับการใช้งานโดยตรง)

บทที่ 2

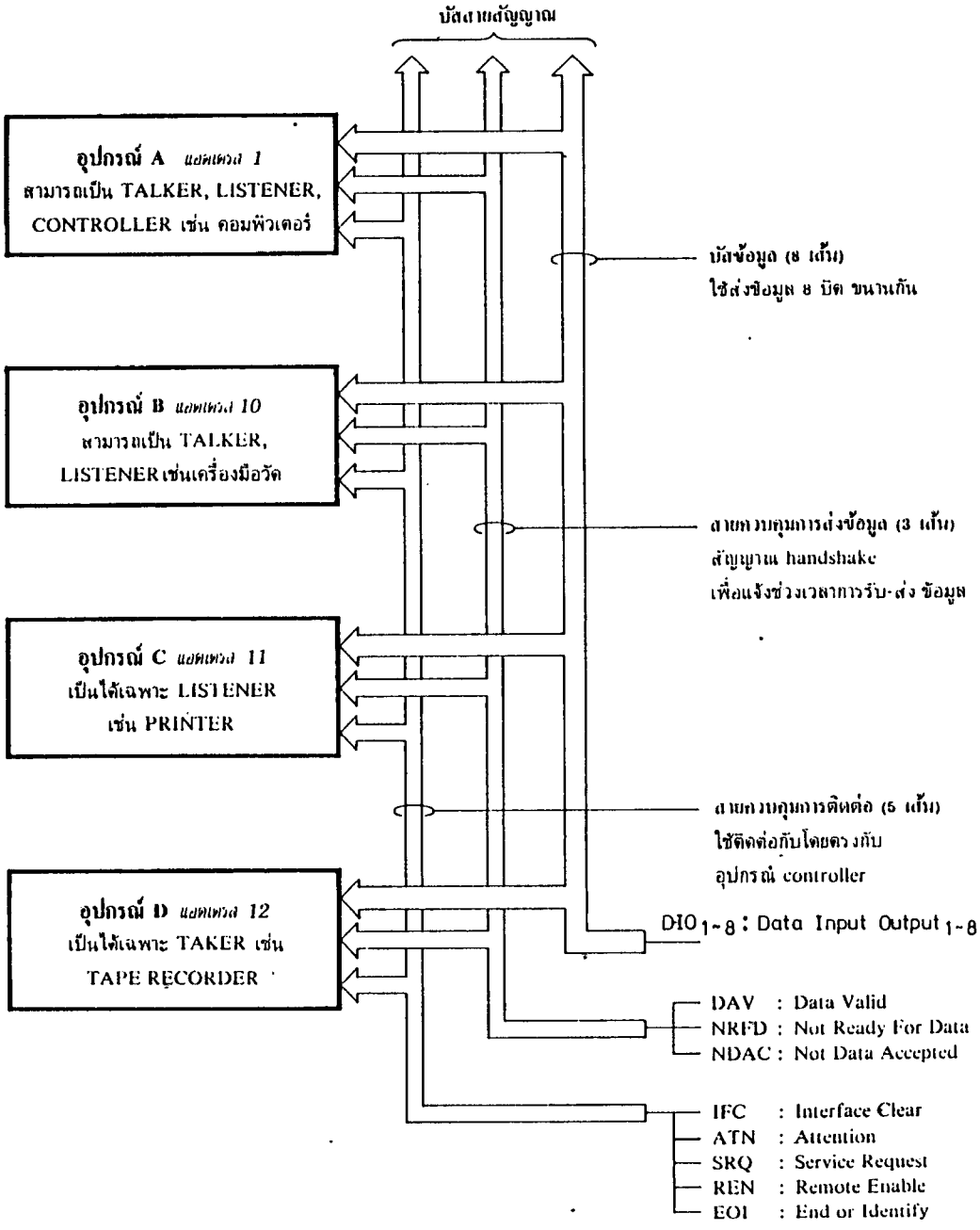
สัญญาณต่าง ๆ ในระบบบัสแบบขนาน

ที่มาของ GPIB นั้นจริง ๆ แล้วเกิดจากการรับรองของสถาบันวิศวกรไฟฟ้าและอิเล็กทรอนิกส์ของอเมริกา (Institute of Electrical and Electronics Engineers : IEEE) ซึ่งยอมรับเข้าเป็นมาตรฐานของสถาบัน และจัดเข้าเป็นลำดับที่ IEEE488 แต่เนื่องจากในคุณลักษณะสมบัติของ IEEE488 นี้ มีความหมายถึงบัสอินเตอร์เฟสที่ใช้งานได้ทั่วไปจึงมีผู้นิยมใช้คำว่า GPIB

GPIB มีชื่อเต็มว่า General Purpose Interface Bus เป็นบัสสัญญาณมาตรฐานที่ใช้ในการติดต่อรับส่งข้อมูลระหว่างอุปกรณ์ได้หลายเครื่อง โดยมีข้อพิเศษสำหรับผู้ใช้ก็คือในกรณีที่ต้องการขยายอุปกรณ์เพิ่มเติมเข้ามาในระบบผู้ใช้ไม่จำเป็นต้องเพิ่มเติมส่วนวงจร หรืออุปกรณ์อื่น ๆ อีกเลย เพียงแต่ผู้ใช้เพิ่มสายเคเบิ้ลเชื่อมต่อเข้ามาขนานกับสายเคเบิ้ลหรือขั้วต่อเดิมเท่านั้น โดยใช้การแก้ไขเฉพาะส่วนของซอฟแวร์เท่านั้น

ในระบบที่ไม่ใหญ่โตนัก GPIB นั้นก็สามารถต่อเข้ากับอุปกรณ์หรือเครื่องมืออื่น ๆ ได้สูงสุด 15 เครื่องโดยใช้สายสัญญาณเพียง 1 เส้นต่อขนานกันไปเรื่อย ๆ ดังนั้นบัสสัญญาณของ GPIB จึงเป็นบัสแบบขนานโดยมีสัญญาณควบคุมร่วมด้วยเพื่อควบคุมทิศทางและเลือกตัวที่ต้องการติดต่อ

หากจะเปรียบเทียบกับมาตรฐานการรับ-ส่งสัญญาณของ RS-232C หรือ Centronics Interface แล้ว GPIB มีข้อยุ่งยากและซับซ้อนกว่าในแง่การใช้งาน เพราะต้องมีการใช้คำสั่งควบคุมอุปกรณ์แต่ละตัวก่อนที่จะรับ-ส่งข้อมูลกันได้



รูปที่ 2.1 แผนผังแสดงอุปกรณ์ GPIB และการต่อสายสัญญาณต่าง ๆ

โครงสร้างของ GPIB

ส่วนประกอบพื้นฐานของ GPIB แสดงไว้ในรูปที่ 2.1 กล่าวคือ GPIB จะประกอบไปด้วยผู้ส่ง (talker), ผู้รับ (listener) และผู้ควบคุม (controller)

- talker ทำหน้าที่ในการส่งข้อมูลโดยสามารถที่จะนำ talker จำนวนมาก ๆ ใส่ไว้ในระบบแต่จะมี talker เพียงตัวเดียวเท่านั้นที่กำลังทำงานอยู่
- listener ทำหน้าที่รับข้อมูล listener ก็เช่นเดียวกับ talker คือ สามารถนำไปใส่ไว้ในระบบได้จำนวนมากและ listener ยังสามารถทำงานพร้อมกันในเวลาเดียวกันได้จำนวนมาก ดังนั้นจึงมีระบบที่ประกอบด้วย talker 1 ตัวและ listener หลายตัว
- controller เป็นตัวควบคุมสัญญาณต่าง ๆ บนบัส โดยรับความต้องการของอุปกรณ์ที่จะส่งข้อมูล หรือกำหนด talker ให้ทำการส่งหรือกำหนด listener ทำการรับข้อมูล

อุปกรณ์ที่มี GPIB

อุปกรณ์หรือเครื่องมือที่มี GPIB นั้น แบ่งตามหน้าที่การทำงานได้ดังนี้

1. ทำหน้าที่เป็น talker เท่านั้น เช่น เครื่องมือวัด เป็นต้น
2. ทำหน้าที่เป็น listener เท่านั้น เช่น เครื่องพิมพ์ (printer), เครื่องบันทึก (recorder) เป็นต้น
3. ทำหน้าที่เป็น talker และ listener เช่น คอมพิวเตอร์, เครื่องมือวัดที่สามารถควบคุมได้จากภายนอก เป็นต้น
4. ทำหน้าที่เป็น talker, listener และ controller เช่น คอมพิวเตอร์ที่ทำหน้า ที่ควบคุมระบบ

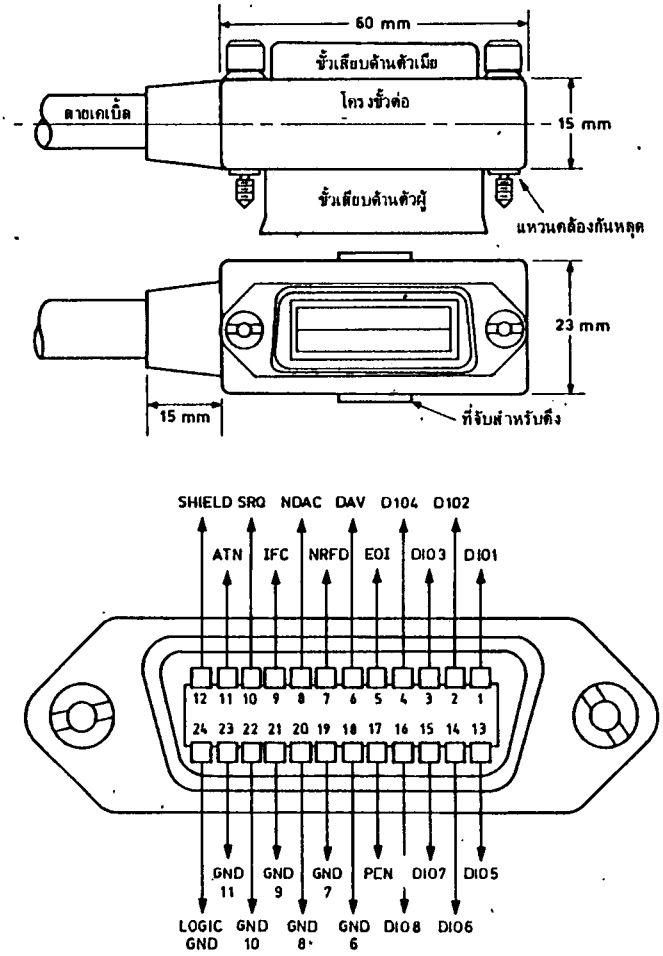
คุณลักษณะทางไฟฟ้าของ GPIB

คุณสมบัติทางไฟฟ้าซึ่งจะเป็นตัวกำหนดขีดจำกัดของ GPIB นั้นมีดังนี้

1. จำนวนของอุปกรณ์ (talker, listener, controller) ที่ต่อกับสายสัญญาณ (bus) 1 เส้น จะต้องมีไม่เกิน 15 เครื่อง
2. สายเคเบิ้ลที่ใช้ต่อระหว่างอุปกรณ์แต่ละตัวจะต้องยาวไม่เกิน 4 เมตร และความยาวรวมของสายเคเบิ้ลทั้งหมดต้องไม่เกิน 20 เมตร
3. ความเร็วในการส่งข้อมูลต้องไม่สูงเกิน 1 Mb/Sec (1 ล้านไบต์ต่อวินาที)
4. จำนวนของอุปกรณ์หรือเครื่องมือมากกว่าครึ่งหนึ่งต้องเปิดให้ทำงาน (จ่ายไฟ)

ขั้วต่อ GPIB

ขั้วต่อมาตรฐานของ GPIB เป็นแบบ Amphenal ขนาด 24 ขา และมีการจัดตำแหน่งของสัญญาณที่ขาต่าง ๆ ดังรูปที่ 2.2



รูปที่ 2.2 ขั้วต่อของ GBIP และการจัดขาของสัญญาณต่าง ๆ

สัญญาณควบคุมที่สำคัญ

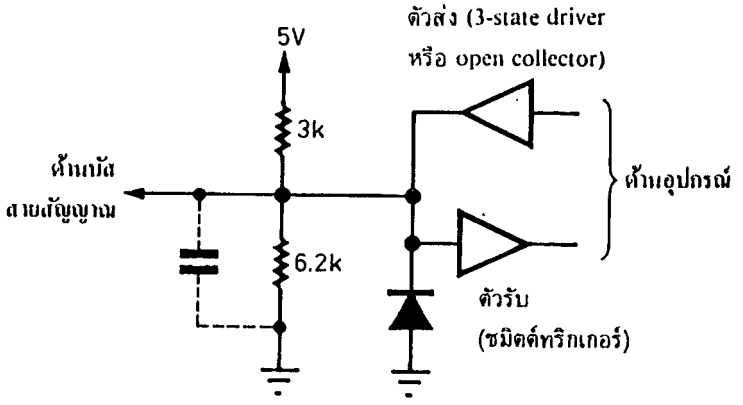
สัญญาณ SQR และ ATN มีรายละเอียดดังนี้

- SRQ (Serial Service Request) เป็นสัญญาณอินพุตที่ป้อนให้กับคอมพิวเตอร์ ซึ่งอุปกรณ์ทุกตัวสามารถดึงให้สัญญาณเป็นลอจิก "ต่ำ" เพื่อแจ้งให้แก่คอมพิวเตอร์ทราบว่า อุปกรณ์ตัวนั้นต้องการขอบริการ
- ATN (Attention) ใช้สำหรับเริ่มต้นการส่งคำสั่ง โดยคอมพิวเตอร์จะทำให้สัญญาณนี้เป็นลอจิก "ต่ำ" เพื่อกำหนดให้อุปกรณ์ทุกตัวที่ต่ออยู่กับสายสัญญาณทำตัวเป็น listener แล้วคอมพิวเตอร์จึงส่งสัญญาณแอดเดรสเพื่อกำ

หนดตำแหน่งของอุปกรณ์แต่ละตัว ถ้าหากอุปกรณ์ตัวที่ถูกเลือกไม่สามารถส่งสัญญาณกลับไปให้คอมพิวเตอร์ภายในช่วงเวลาที่กำหนด ตัวคอมพิวเตอร์จะสมมติว่าไม่มีอุปกรณ์ที่ตำแหน่งแอดเดรสนั้นอยู่ แล้วคอมพิวเตอร์จะแจ้งให้ผู้ใช้ทราบด้วย

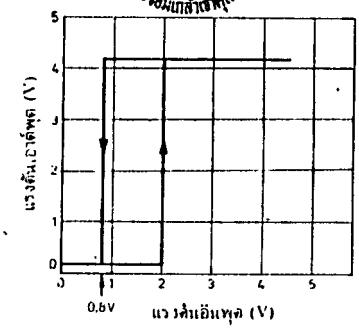
พิจารณาถึงวงจรไฟฟ้า

เมื่อพิจารณาถึงวงจรไฟฟ้าเมื่อมองย้อนเข้าไปยังจุดต่อสัญญาณบัสข้อมูล จะเห็นโหลดของ GPIB เป็นตัวต้านทานแบ่งแรงดัน 2 ตัวคือค่า 6.2 k และ 3 k ดังในรูปที่ 2.3



รูปที่ 2.3 วงจรภายในจุดต่อบัสข้อมูลที่ใช้รับ-ส่งข้อมูล

ทางด้านเกตตัวส่ง (driver) เป็นแบบ open collector หรือแบบเกต 3 สถานะ (3 state driver) โดยการขับด้วยกระแสขนาด 48 mA ส่วนทางด้านเกตตัวรับ (receiver) ใช้แบบชมิตต์ทริกเกอร์ (schmitt trigger) เพื่อกำจัดสัญญาณรบกวน โดยมีคุณสมบัติการให้สัญญาณเอาต์พุต ดังรูปที่ 2.4

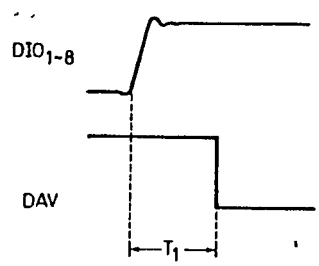


รูปที่ 2.4 คุณสมบัติฮิสเตอริซิสของตัว receiver แบบชมิตต์ทริกเกอร์

จากรูปที่ 2.4 ที่ระดับแรงดันอินพุตมีค่าต่ำ คือ น้อยกว่า 0.8 v เอาต์พุตจะมีค่าต่ำด้วย แต่ถ้าเพิ่มแรงดันอินพุตขึ้นไปเรื่อย ๆ จนมีค่ามากกว่า 2 v เอาต์พุตก็ยังคงมีค่าสูงอยู่ แรงดันอินพุทต้องลดลงจนกระทั่งต่ำกว่า 0.8 v เอาต์พุตจึงกลับมามีค่าต่ำ ลักษณะเช่นนี้เป็นลักษณะของ "ฮิสเตอริซิส"

แต่อย่างไรก็ตามตัวเกต driver/receiver ที่ใช้กับ GPIB และมีจำหน่ายอยู่นั้น อาจมีคุณสมบัติแตกต่างไปจากที่กล่าวมาก็ได้ เช่นกรณีที่ใช้เป็นแบบอุปกรณ์แยกชิ้น (discrete เช่น ทรานซิสเตอร์ทำหน้าที่เป็นสวิตช์) แล้วต่อปลาย (โหลด) ด้วยตัวต้านทาน เมื่อปิดแหล่งจ่ายไฟจะทำให้แหล่งจ่ายไฟนั้นมีค่าเป็น 0 v. (ลัดวงจร) แต่ที่สัญญาณบัสข้อมูลยังมีตัวต้านทาน 6.2 k และ 3 k ต่อขนานกันอยู่ ดังนั้นจึงทำให้มีค่าโหลดประมาณ 2 k ต่ออยู่ สัญญาณบัสข้อมูลกับกราวด์ ซึ่งจะเป็นผลเสียทางช่วงของสัญญาณรบกวน (noise margin)

กรณีที่ใช้เกต driver/receiver เฉพาะของ GPIB เช่น ไอซีเบอร์ SN75160,MC6488A เป็นต้น ซึ่งเป็นโหลดประเภทแอคตีฟ เมื่อปิดแหล่งจ่ายไฟ จะทำให้ไม่มีโหลดเข้าไปเกี่ยวพันด้วยทำให้เกิดผลดีบางด้าน noise margin คือข้อมูลไม่ผิดพลาดได้ง่าย



รูปที่ 2.5 แสดงการเกิดสัญญาณ DIO_{1}-DIO_{8} และ DAV โดยมีค่าเวลาหน่วง T_{1}

นอกจากนั้นความเชื่อถือได้ของการส่งข้อมูลยังขึ้นอยู่กับความยาวของสายเคเบิ้ล ในกรณีที่อุปกรณ์มีจำนวนมากกว่า 11 เครื่องขึ้นไป ความยาวของสายเคเบิ้ลจะต้องไม่เกิน 20 เมตร สำหรับกรณีที่อุปกรณ์มีจำนวนไม่เกิน 10 เครื่องนั้น ความยาวของสายเคเบิ้ล (L) จะมากกว่า 20 เมตร จึงควรจำกัดความยาวของสายเคเบิ้ลด้วยสูตร

$$L < 2N \ldots \text{เมตร}$$

โดย N เป็นจำนวนอุปกรณ์

ดังนั้นความยาวของสายเคเบิ้ลระหว่างอุปกรณ์ก็เช่นกันจะมีค่ามากที่สุด 4 เมตร ในทางปฏิบัติจริง ๆ แล้ว แม้ความยาวจะเกินกว่า 4 เมตรก็ตามยังสามารถทำงานได้โดยไม่มีความผิดพลาด แต่ควรจะปฏิบัติตามมาตรฐานจะปลอดภัยกว่า

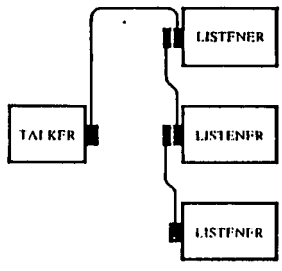
ทางด้านความเร็วในการส่งข้อมูล ซึ่งมีค่าสูงสุด 1Mb/Sec แต่ก็ยังมีขีดจำกัดอยู่อีกเช่นกัน กล่าวคือ หลังจากที่ป้อนข้อมูลออกทางสายสัญญาณ DIO_1-DIO_8 แล้ว สายสัญญาณ DAV จะต้องแอคตีฟโดยมีเวลาหน่วง T_1 ตามรูปที่ 2.5 ซึ่งค่าเวลา T1 จะเป็นตัวกำหนดความเร็วในการส่งข้อมูลดังนี้

1. กรณีที่ใช้ driver เป็นแบบ open collector ขนาดกระแส 48 mA เวลาหน่วง T_1 จะมีค่า > 2μS และทุก ๆ ระยะ 2 เมตร ติดตั้งโหลดมาตรฐานเข้าไปแล้วละก็ ความเร็วในการส่งในระยะ 20 เมตร จะต้องมีค่าต่ำกว่า 250 กิโลไบต์/วินาที
2. กรณีที่ใช้ driver เป็นแบบ 3-state driver ขนาดกระแส 48 mA และติดตั้ง โหลดมาตรฐานทุก ๆ ระยะ 2 เมตร T_1 มีค่า 500 nS และความเร็วใน การส่งในระยะ 20 เมตร จะต้องมีค่าต่ำกว่า 500 กิโลไบต์ /วินาที
3. กรณีที่ใช้ 3-state driver, T_1 = 350 nS และติดตั้งโหลดมาตรฐานทุก ๆ 1 เมตร ความเร็วในการส่งในระยะ 15 เมตร จะต้องมีค่าต่ำกว่า 1 เมกกะไบต์ /วินาที

ข้อควรระวังในการใช้ GPIB

กรณีที่ต้องการออกแบบหรือติดตั้งอุปกรณ์GPIBในระบบนั้น จำเป็นต้องระมัดระวังสิ่งต่าง ๆ ดังต่อไปนี้

1. จะต้องไม่ยืดการเดินสายระหว่างขั้วคอนเน็คเตอร์ให้ยาวออกไป ซึ่งจะทำให้เกิด Crosstalk และค่าตัวเก็บประจุในสายสัญญาณเพิ่มขึ้น
2. ปลายของสายชีลด์ (ขาต่อเบอร์ 12) ต่อเข้ากับเฟรมตรงใกล้ ๆ กับขั้วต่อคอนเน็คเตอร์
3. เมื่อทำการต่อระหว่างอุปกรณ์ต่าง ๆ ด้วยสายเคเบิ้ล ต้องระมัดระวังอย่าให้เกิดเป็น ลูป (loop) ซึ่งในกรณีที่อุปกรณ์ใช้ภาคจ่ายไฟแบบ switching อาจจะเกิดสัญญาณรบกวนเข้าไปในบัสสัญญาณได้ อาจทำให้เกิดการทำงานผิดพลาดได้
4. จะต้องไม่ให้มีการแพร่กระจายของสัญญาณรบกวนที่ไม่ต้องการออกไปจาก สายเคเบิ้ล ซึ่งในกรณีที่อุปกรณ์เป็นไมโครคอมพิวเตอร์จะเกิดความถี่ฮาร์โมนิคประมาณ 100 MHz จากความถี่นาฬิกาของระบบ ทำให้ไปรบกวนเครื่องรับวิทยุโทรทัศน์ได้ ซึ่งจะป้องกัน ได้โดยการใช้ขั้วต่อคอนเน็คเตอร์ที่มีการชีลด์ด้วยโครงโลหะอย่างดีตามมาตรฐาน ของ FCC



รูปที่ 2.6 ระบบการต่อ talk only และ listen only แบบง่าย ๆ

ระบบ TALK ONLY และ LISTEN ONLY

ในระบบ GPIB แบบง่าย ๆ ที่ไม่สลับซับซ้อน จะประกอบไปด้วย talker และ listener ดังแสดงในรูปที่ 2.6 ซึ่งวิธีการใช้งานอย่างนี้เรียกว่า talk only, listen only.

talk only หมายความว่าอุปกรณ์ตัวหนึ่งถูกกำหนดให้เป็น talker (ส่งข้อมูล) ส่วน listen only หมายความว่า อุปกรณ์อีกตัวหนึ่งกำหนดให้เป็น listener (รับข้อมูล) ลักษณะเช่นนี้ใช้ต่อเมื่อต้องการส่งข้อมูลระหว่าง talker กับ listener โดยไม่มี controller

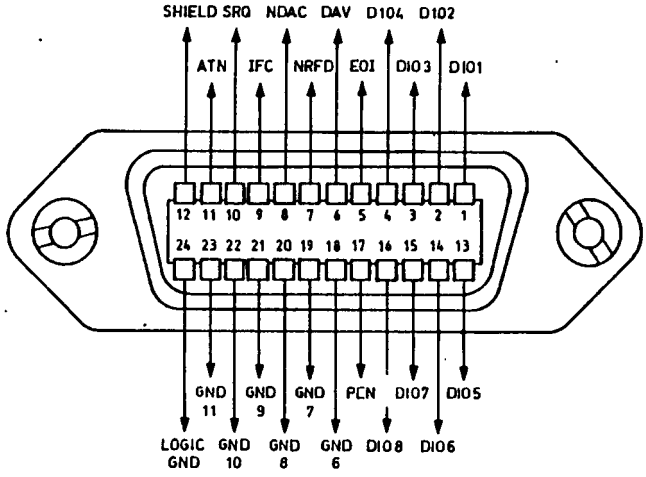
สำหรับระบบที่ทำงานโดยใช้ controller นั้น โดยทั่วไปแล้วไม่นิยมใช้ จะใช้เฉพาะกับกรณีที่ทำการตรวจสอบการส่งข้อมูลและเมื่อกำหนดให้เครื่องพิมพ์ (printer) เป็น listener เท่านั้น ดังนั้นในระบบที่มี controller จะไม่มีอุปกรณ์ talk only ต่อเข้ากับบัสสัญญาณ ถ้าหากมีอุปกรณ์ talk only จะทำให้ไม่สามารถส่งข้อมูลจาก controller ได้ด้วยเหตุนี้ สำหรับอุปกรณ์ที่มีการทำงานเป็น talk only หรือ listen only จำเป็นต้องมีสวิตช์ ON/OFF การทำงานติดมากับเครื่องด้วย

อุปกรณ์ที่มีระบบบัสเชื่อมต่อข้อมูลเอนกประสงค์

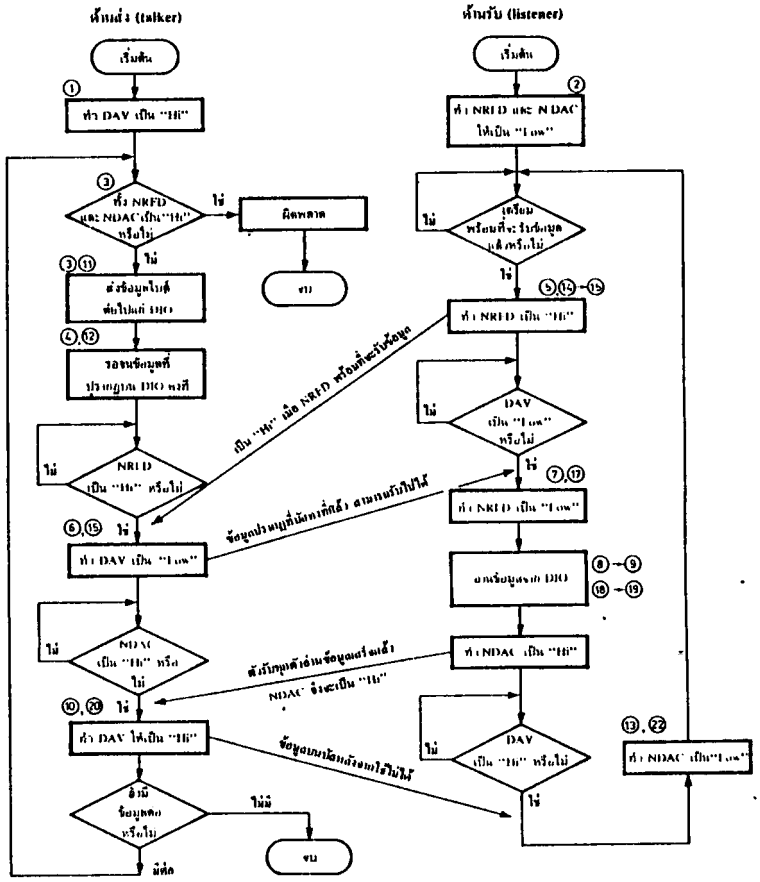
อุปกรณ์ที่มี GPIB แบ่งตามหน้าที่การทำงานได้ดังนี้

1. อุปกรณ์ที่ทำหน้าที่เป็นตัวส่งข้อมูล (Talker) เท่านั้น เช่น เครื่องมือวัด เป็นต้น
2. อุปกรณ์ที่ทำหน้าที่เป็นตัวรับข้อมูล (Listener) เท่านั้น เช่น เครื่องพิมพ์ (Printer), เครื่องบันทึก (Recorder) เป็นต้น
3. อุปกรณ์ที่ทำหน้าที่เป็นตัวรับข้อมูลและส่งข้อมูล (Listener and Talker) เช่น คอมพิวเตอร์, เครื่องมือวัดที่สามารถควบคุมได้จากภายนอก เป็นต้น
4. อุปกรณ์ที่ทำหน้าที่เป็นตัวรับข้อมูล, ตัวส่งข้อมูล และตัวควบคุม (Listener,

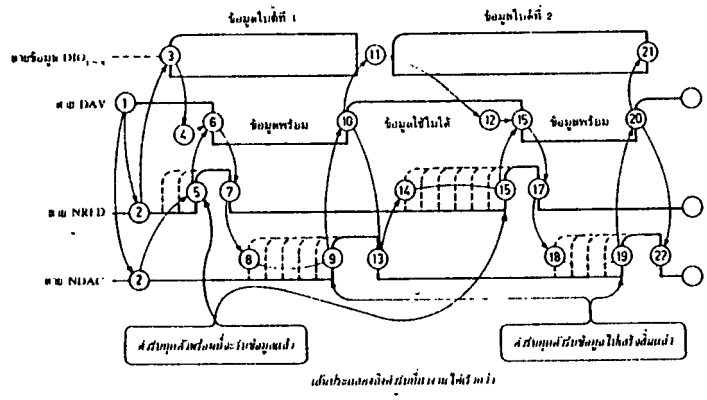
Talker and Controller) เช่น คอมพิวเตอร์ที่ทำหน้าที่ควบคุมระบบ



รูปที่ 2.7 ขั้วต่อของ GPIB และการจัดขาของสัญญาแต่ละ ๆ



รูปที่ 2.8 ผังงานของการรับส่งข้อมูลแต่ละครั้งแสดงการโต้ตอบของสัญญาณ DAV, NRFD และ NDAC



รูปที่ 2.9 ผังเวลาแสดงความสัมพันธ์ของสัญญาณโต้ตอบในการรับส่งข้อมูล

ความหมายของสัญญาณต่าง ๆ ในบัส

GPIB เป็นระบบบัสแบบขนาด ดังรูป 2.7 ดังนั้นอุปกรณ์ทุกตัวที่ต่อร่วมบัสกันอยู่จึงต่อขนานกันหมด สัญญาณต่าง ๆ จากระบบบัสจึงปรากฏต่ออุปกรณ์ทุกตัวในจำนวนสายต่อทั้ง 24 เส้นของบัสสามารถแบ่งออกได้เป็น 3 กลุ่มสัญญาณ คือ

กลุ่มสัญญาณควบคุมการรับส่งข้อมูลประกอบด้วย

- DAV (data valid) เมื่อถูกดึงเป็นลอจิก "Low" โดยอุปกรณ์ที่เป็นตัวส่ง (talker) เป็นการแจ้งแก่ระบบบัสว่า ตอนนี้ตัวส่งได้ทำการส่งข้อมูลลงไปที่สาย สัญญาณข้อมูลเรียบร้อยแล้ว
- NRFD (not ready for data) เมื่อมีลอจิก "Low" เป็นการแสดงว่าตอนนี้ระบบบัสยังไม่พร้อมที่จะรับข้อมูล เนื่องจากอุปกรณ์ในระบบยังพร้อมไม่ หมดทุกตัว ซึ่งสัญญาณเส้นนี้จะไม่เป็น "Hi" จนกว่าอุปกรณ์ทุกตัวให้ลอจิก "Hi" ครบถ้วนแล้ว ใช้ประโยชน์สำหรับกรณีที่อุปกรณ์ที่ใช้ร่วมกันมีความเร็วในการทำงานแตกต่างกัน
- NDAC (not data accepted) สัญญาณเส้นนี้ ควบคุมโดย

อุปกรณ์ตัวรับ (listener) จะมีลอจิก "Low" ในขณะที่ตัวรับกำลังเก็บข้อมูลจากสายข้อมูล และ จะเป็น "Hi" เมื่ออ่านข้อมูลเสร็จเรียบร้อยแล้ว

กลุ่มสัญญาณควบคุมการอินเตอร์เฟสประกอบด้วย

- ATN (attention) เป็นสัญญาณจากอุปกรณ์ที่เป็นตัวควบคุม (controller) ใช้ในการสั่งให้อุปกรณ์ทุกตัวในระบบเตรียมพร้อมเพื่อรอรับคำสั่งต่อไป
- IFC (interface clear) เป็นสัญญาณรีเซ็ตหรือเคลียร์ระบบ ก้าเนิดได้ โดยตัวควบคุมเท่านั้น เมื่ออุปกรณ์ในบัสได้รับสัญญาณเคลียร์นี้ จะกลับคืนสู่สภาวะเริ่มต้นใหม่ เป็นสภาวะแรกเริ่มก่อนการกำหนดฟังก์ชันเหมือนเมื่อแรกเปิดสวิตช์
- REN (remote enable) สัญญาณเส้นนี้ควบคุมโดยอุปกรณ์ตัวควบคุมตัวเดียวเท่านั้นเช่นกัน ใช้สั่งให้อุปกรณ์เปลี่ยนจากโหมดที่ใช้งานปกติด้วยมือมาเป็น การควบคุมโดยตัวควบคุมแทน
- SRQ (service request) เป็นสายสัญญาณอินเตอร์รัพต์ เพื่อเป็นการบอกแก่ระบบว่า ขณะนี้อุปกรณ์ต้องการการติดต่อจากตัวควบคุมยกตัวอย่างเช่น โวลต์มิเตอร์ที่มีค่าที่อ่านได้พร้อมแล้วที่จะส่งให้แก่ระบบเพื่อไปยังอุปกรณ์ที่ต้องการข้อมูลนี้ หรือเมื่อ เกิดมีการพบข้อผิดพลาดบางอย่าง
- EOI (end or Identify) สัญญาณที่ควบคุมได้ทั้งโดยอุปกรณ์ที่เป็นตัวควบคุมหรือตัวส่งก็ได้ ใช้แสดงว่าข่าวสารข้อมูลที่ส่งเป็นชุดนั้นสิ้นสุดลงแล้ว

กลุ่มสัญญาณข้อมูล (Data Line)

ประกอบด้วยสายสัญญาณจำนวน 8 เส้น สำหรับเป็นทางผ่านของข่าวสารข้อมูลของระบบ

สัญญาณลอจิกที่ใช้ใน GPIB บัสนี้มีลักษณะเป็นคอมพลีเมนต์ทั้งหมด คือ "1" เท่า กับ "Low" และ "0" เท่ากับ "Hi" ซึ่งตรงกันข้ามกับที่เราคุ้นเคยกัน

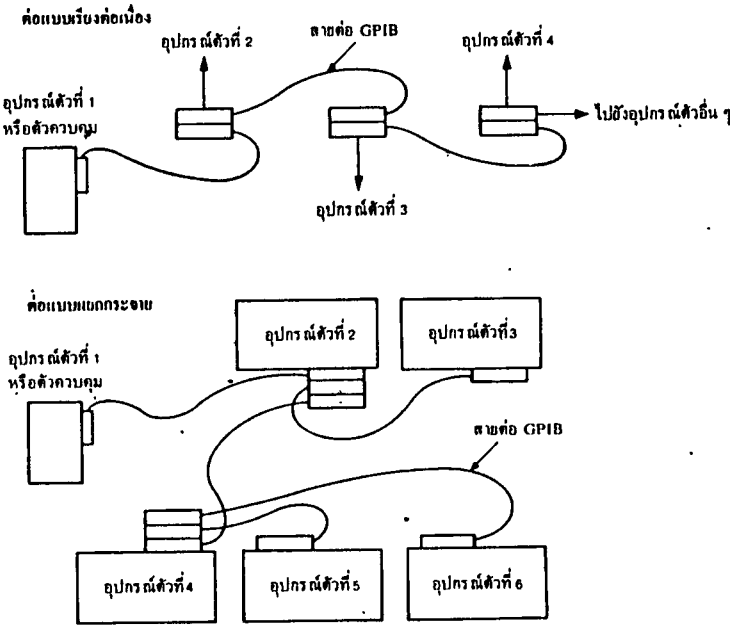
สัญญาณควบคุมที่มีอยู่อาจไม่ได้ใช้ทั้งหมดพร้อมกัน บางสายอาจไม่ได้ใช้งานในอุปกรณ์บาง เครื่องก็ได้ แล้วแต่ความจำเป็นในการติดต่อสำหรับสายที่เหลืออีก 7 เส้นนั้นเป็นสายกราวด์

ในเรื่องของรายละเอียดทางด้านสัญญาณคงจะไม่กล่าวลึกไปกว่านี้ ซึ่งเป็นส่วนของผู้ออก แบบระบบที่จะใช้งานมากกว่า ในส่วนผู้ใช้งานคงไม่จำเป็นนัก

สำหรับการต่ออุปกรณ์ต่าง ๆ ในระบบ GPIB อินเตอร์เฟสบัส มีอยู่ 2 วิธีคือ แบบต่อเนื่องกันไปเรื่อย ๆ จากเครื่องหนึ่งไปยังอีกเครื่องหนึ่ง และอีกวิธีก็คือ ต่อแบบกระจาย ดังแสดงในรูปที่ 2.10

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 (สายกราวด์ของ)	20	GND 7
21	GND 9 (สายที่ 6-11)	21	GND 8 (สายกราวด์ของ)
22	GND 10	22	GND 9 (สายที่ 6-12)
23	GND 11	23	GND 10
24	LOGIC GND	24	GND 11
		25	GND 12

ตารางที่ 1 เปรียบเทียบการจัดตำแหน่งสัญญาณระหว่าง IEEE Std488-1978 กับ IEC625-1



รูปที่ 2.10 ลักษณะการต่ออุปกรณ์เข้ากับระบบบัส GPIB ทั้ง 2 วิธี

คำสั่งใช้งานของ GPIB

เมื่อมีการเชื่อมต่อ หรืออินเตอร์เฟสอุปกรณ์เครื่องมือต่าง ๆ เข้าด้วยกันด้วย GPIB บัสแล้ว ตัวอย่างเช่น อาจจะเป็นการต่อคอมพิวเตอร์กับเครื่องวัดความถี่, ปริ้นเตอร์ และดิสค์ไดรฟ์ ตัวคอมพิวเตอร์ก็จะทำหน้าที่เป็นตัวควบคุมเพื่อบริหารอุปกรณ์อื่น ๆ และเป็นตัวรับเพื่อรับข้อมูลจากเครื่องวัดความถี่ ขณะเดียวกันก็อาจจะเป็นตัวส่งค่าความถี่ที่วัดได้ให้แก่ตัวอื่น และปริ้นเตอร์เป็นตัวรับข้อมูลมาพิมพ์ ส่วนดิสค์ไดรฟ์ก็เป็นได้ทั้งตัวรับและตัวส่งข้อมูล

จะเห็นว่าอุปกรณ์แต่ละตัวสามารถเป็นได้หลายอย่าง ขึ้นอยู่กับว่าในการทำงานขณะหนึ่ง ๆ ต้องการให้เป็นอะไรซึ่งอุปกรณ์ทุกตัวที่ต่ออยู่ในบัสจะมีหมายเลขรหัสประจำตัวสำหรับการอ้างถึง (เหมือนกับบ้านที่ต้องมีเลขที่) เพื่อให้ตัวควบคุมสั่งการบังคับหน้าที่ต่าง ๆ ได้ และใช้ในการอ้างอิงจุดหมายปลายทางของข้อมูล

การสั่งการต่าง ๆ เพื่อกำหนดหน้าที่การทำงาน และกำหนดฟังก์ชัน เช่น กำหนดช่วงการวัด โหมดการวัด หรืออื่น ๆ แก่เครื่องวัดที่ต่ออยู่ เหล่านี้นั้นตัวควบคุมจะเป็นตัวกำ

หนด โดยการส่งรหัสคำสั่งไปที่อุปกรณ์ โดยผ่านสายสัญญาณข้อมูล DI_1-DI_8 รหัสคำสั่งนี้จะถูกส่งไปในช่วงที่สายสัญญาณ ATN เป็น "Low" (ในระบบง่าย ๆ ที่ไม่ซับซ้อน ซึ่งไม่จำเป็นต้องใช้ตัวควบคุม อุปกรณ์จะถูกกำหนดไว้ตายตัว เช่นให้เป็นตัวส่งอย่างเดียว (talk only) หรือรับอย่างเดียว (listen only))

คำสั่งสำหรับกำหนดหน้าที่การทำงานแบบต่าง ๆ ตามมาตรฐานของ GPIB มีอยู่ด้วยกัน 128 รหัสคำสั่ง ดังแสดงในตารางที่ 2 โดยแบ่งได้เป็น 5 กลุ่มคำสั่ง

รหัสที่ใช้ในระบบ GPIB บัสนั้น ใช้ร่วมกันทั้งรหัสข้อมูล และรหัสคำสั่ง นั่นคือรหัสเดียวกันมีความหมายได้ 2 อย่าง คือเมื่อสายสัญญาณ ATN เป็น "Low" จะหมายถึงรหัสคำสั่ง แต่ถ้า ATN เป็น "Hi" รหัสนี้จะแทนข้อมูลที่เป็น ASCII แทน ซึ่งในตารางที่ 2 ก็ได้แบ่งความหมายออกเป็น 2 คอลัมน์ให้เห็น รหัสคำสั่งต่าง ๆ มีดังต่อไปนี้

1. กลุ่มคำสั่งเจาะจงจุดหมาย (addressed command group) เป็นคำสั่งที่ส่งไปยังอุปกรณ์ที่เป็นตัวส่งหรือตัวรับที่กำหนดไว้ล่วงหน้าแล้ว ให้มีสภาพการทำงานตามที่ต้องการ คำสั่งในกลุ่มนี้ประกอบด้วย

- GTL (go to local) สั่งให้อุปกรณ์กลับสู่สภาพการควบคุมปกติด้วยมือ
- SDC (selected divice clear) สั่งให้อุปกรณ์เคลียร์ตัวเองสู่สภาพเริ่มต้นใหม่
- PPC (parallel poll configure) เป็นคำสั่งสำหรับทำการจัดสรรสายสัญญาณของการทำกระบวนการตรวจสอบสภาพอุปกรณ์โดยวิธี ขนาน หรือ parallel poll (ซึ่งจะอธิบายภายหลัง) โดยใช้งานร่วมกับคำสั่งในกลุ่มรอง
- GET (group excute trigger) ใช้สั่งเริ่มต้นการทำงานของอุปกรณ์ทีละหลายตัว
- TCT (take control) เป็นการกำหนดให้อุปกรณ์ตัวส่งทำหน้าที่เป็นตัวควบคุม

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

MSD / LSD	0		1		2		3		4		5		6		7	
	ASCII	MSG	ASCII	MSG	ASCII	MSG¹	ASCII	MSG¹	ASCII	MSG¹	ASCII	MSG¹	ASCII	MSG	ASCII	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	MEANING DEFINED BY PCG CODE	q	MEANING DEFINED BY PCG CODE
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	↓

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

PRIMARY COMMAND GROUP (PCG)

ตารางที่ 2 รหัสข้อมูลข่าวสารของระบบ GPIB บัส

2. กลุ่มคำสั่งควบคุม (universal command group) เป็นคำสั่งที่ส่งไปยังอุปกรณ์ทุกตัวที่ต่ออยู่ในบัส ประกอบด้วย

- LLO (local lockout) เป็นการสั่งให้อุปกรณ์ล็อคอยู่ที่สภาวะควบคุมโดยปุ่มปรับที่หน้าปัดตามปกติ
- DCL (device clear) สั่งให้อุปกรณ์ทุกตัวกลับไปสู่สภาวะเริ่มต้น
- PPU (parallel poll unconfigure) ใช้ยกเลิกกระบวนการตรวจสอบสภาพแบบขนานทั้งหมด
- SPE (serial poll enable) เปลี่ยนโหมดของการตรวจสอบสภาพเป็นแบบอนุกรม ในโหมดนี้จะเป็นการส่งสถานะของเครื่องแทนการส่งข้อมูล
- SPD (serial poll disable) ยกเลิกโหมดการตรวจสอบแบบอนุกรม

3. กลุ่มคำสั่งกำหนดอุปกรณ์ตัวรับ (listener address group) เป็นคำสั่งสำหรับกำหนดให้อุปกรณ์เป็นตัวรับตามรหัสหมายเลขจาก 0 ถึง 30 และมีคำสั่ง UNL (unlistener) สำหรับยกเลิก

4. กลุ่มคำสั่งกำหนดอุปกรณ์ตัวส่ง (talker address group) สำหรับกำหนดให้อุปกรณ์เป็นตัวส่ง ตามรหัสหมายเลขจาก 0 ถึง 30 และมีคำสั่ง UNL (unlistener) สำหรับยกเลิกเช่นกัน

คำสั่งในกลุ่มที่ 1 ถึง 5 นั้น จัดเป็น *กลุ่มคำสั่งหลัก (primary command group)* ที่มีความหมายตายตัว ยังมีคำสั่งอีกกลุ่มที่ขึ้นอยู่กับการกำหนดภายหลังนั่นคือ กลุ่มคำสั่งรอง

5. กลุ่มคำสั่งรอง (secondary command group) เป็นคำสั่งที่กำหนดรายละเอียดย่อยของอุปกรณ์แต่ละตัวที่ต่ออยู่ในระบบ ให้มีการทำงานอย่างไรตามจุดประสงค์ใช้งานของเครื่องมือนั้นเช่นเดียวกับการปรับปุ่มปรับต่าง ๆ ด้วยมือนั่นเอง คำสั่งรองนี้จะตามหลังคำสั่งหลัก คือ จะใช้หลังจากอุปกรณ์ต่าง ๆ ถูกกำหนดวางตัวในระบบเรียบร้อยแล้ว

คำสั่งต่าง ๆ ที่กล่าวไป ซึ่งใช้ในการกำหนดสภาวะการทำงานของอุปกรณ์ แต่ละสภาวะที่กำหนดไปนั้นเป็นอย่างไร และมีจุดประสงค์เพื่ออะไร เรามาดูกันต่อ

Device Clear/Interface Clear

ดีไวซ์เคลียร์ (device clear) ใช้ในการทำให้อุปกรณ์ที่ต่ออยู่ในบัสกลับไปอยู่ในสภาวะเหมือนเมื่อแรกเริ่มที่เปิดไฟเข้าเครื่อง อันเป็นสภาวะเริ่มต้น ยังไม่มีการกำหนดฟังก์ชันใด ๆ สภาวะเริ่มต้นนี้จะแตกต่างกันไปแล้วแต่ว่าอุปกรณ์นั้นออกแบบมาไว้อย่างไร ดีไวซ์เคลียรมีอยู่ 2 ลักษณะ คือ เคลียร์หมดทุกตัวที่ต่ออยู่ (DCL) กับเคลียร์เจาะจงอุปกรณ์ตัวใดตัวหนึ่ง (SDC)

แต่ว่าในการเคลียร์อุปกรณ์ให้อยู่ในสภาวะเริ่มต้นนั้น ไม่ได้หมายความว่าอินเตอร์เฟสฟังก์ชันของ GPIB จะถูกเคลียร์ให้ไปอยู่ในสภาวะเริ่มต้นด้วยแต่อย่างใด เป็นแต่การเคลียร์เฉพาะตัวอุปกรณ์เท่านั้น อินเตอร์เฟสฟังก์ชันก็คือ สภาพการอินเตอร์เฟสที่ได้กำหนดเอา

ไว้ในระบบ ประกอบด้วยฟังก์ชันต่าง ๆ ดังแสดงในตารางที่ 3

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	✓

ตารางที่ 3 ฟังก์ชันการอินเตอร์เฟสในระบบ GPIB บัส

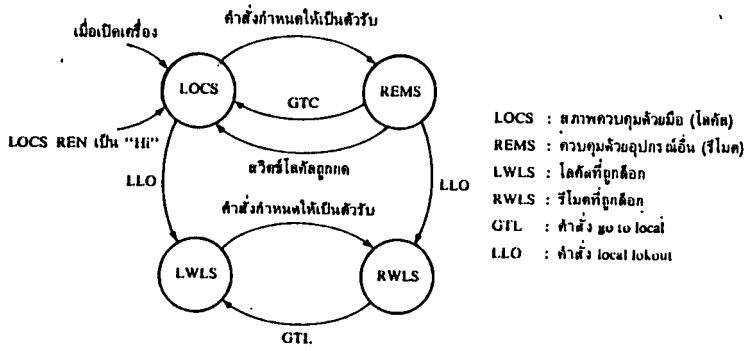
การเคลียร์สภาพการอินเตอร์เฟสไปอยู่ในสภาวะเริ่มต้นต้องใช้คำสั่ง *อินเตอร์เฟสเคลียร์* (interface clear) แทน ซึ่งจะทำให้ทุกฟังก์ชันถูกยกเลิกไป ยกเว้น SR (service request), RL (remote/local) และ PP (parallel poll) นั่นก็คือถึงจะยกเลิกสภาพอินเตอร์เฟส แต่จะยังคงรักษาลักษณะการจัดของ SR, RL และ PP เอาไว้

Remote/Local

รีโมต (remote) เป็นการกำหนดให้อุปกรณ์ที่ต่ออยู่ในระบบ เช่น เครื่องมือวัดให้อยู่ในการควบคุมของอุปกรณ์ตัวอื่นแทน ซึ่งปุ่มปรับต่าง ๆ บนหน้าปัดเครื่องจะไม่มีผลต่อการทำงาน ส่วน *โลคัล* (local) เป็นการควบคุมการทำงานของเครื่องมือวัดด้วยปุ่มปรับบนหน้าปัดตามปกติ หรือควบคุมด้วยมือนั่นเอง

การใช้รีโมตมีประโยชน์ในแง่ที่ขณะที่ตัวควบคุม เช่น คอมพิวเตอร์กำลังติดต่อกับอุปกรณ์ตัวนั้นอยู่ หากไม่มีการตัดการควบคุมโดยปุ่มปรับบนหน้าปัดออก ถ้ามีใครมาปรับแต่งก็จะทำให้การทำงานผิดพลาดไปได้ ในทางกลับกัน ขณะที่ใช้งานแบบปกติในโหมดของโลคัลก็จะไม่ให้คอมพิวเตอร์มายุ่งเกี่ยวได้ การทำงานของ GPIB ในรีโมตและโลคัลมี 4 ลักษณะดังนี้

1. LOCS ก็คือโลคัลนั่นเอง อันเป็นสภาพการควบคุมที่ปุ่มปรับตามปกติ ซึ่งอุปกรณ์จะอยู่ในสภาพนี้เมื่อตอนเปิดไฟเข้าเครื่อง หรือ REN. (remote enable) มีลอจิก "Hi" หรือเมื่อได้รับคำสั่ง GTL (go to local)
2. REMS คือรีโมต หมายถึงการตัดการควบคุมโดยปุ่มบนหน้าปัดออกกลายเป็นการควบคุมการทำงานโดยอุปกรณ์อื่นแทน สภาพรีโมตจะเกิดขึ้นเมื่อ REN มีลอจิก "Low" และจะถูกล็อคเอกไว้ เว้นแต่ว่าสวิตช์โลคัลที่ตัวอุปกรณ์จะถูกเปลี่ยนไปตำแหน่ง local
3. RWLS เป็นสภาพรีโมตที่ถูกล็อคเอาไว้เช่นกัน แต่ว่าจะตัดการควบคุมของสวิตช์โลคัลที่ตัวอุปกรณ์ออกไป ฉะนั้น สภาพรีโมตโดย RWLS จึงมีนัยสำคัญสูงกว่า REMS อย่างไรก็ตามก็ยังถูกยกเลิกได้ด้วยคำสั่ง LLO (Local lockout)
4. LWLS มีสภาพเช่นเดียวกับโลคัล แต่จะแตกต่างกันตรงที่สภาพโลคัลโดย LWLS นี้ เมื่อได้รับคำสั่งกำหนดอุปกรณ์ตัวรับ จะเปลี่ยนไปอยู่ในสภาพรีโมต แบบล็อกหรือ RWLS ทันที ในการที่จะมาที่สภาพ LWLS นี้ได้ก็มี 2 กรณี คือ เมื่ออยู่ ในสภาพโลคัลธรรมดา (LOCS) แล้วได้รับคำสั่ง LLO (local lockout) หรือ เมื่ออยู่ใน RWLS แล้ว ได้รับคำสั่ง GTL (go to local)



รูปที่ 2.11 สภาพโลคัลและรีโมตลักษณะต่าง ๆ และแผนภูมิแสดงการเปลี่ยนสภาวะ

ในรูปที่ 2.11 เป็นการแสดงการเปลี่ยนสภาวะต่าง ๆ ของโลคัลและรีโมตทั้ง 4 ลักษณะตามที่กล่าวมา ซึ่งเมื่อ REN มีลอจิก "Hi" ก็จะกลายเป็นโลคัลทันทีไม่ว่าเดิมจะมีในสภาพใด และเมื่อ REN เป็น "Low" แล้ว หากไม่มีคำสั่งกำหนดอุปกรณ์ตัวรับหรือคำสั่ง LLO เข้ามา ก็จะยังคงอยู่ในสภาพโลคัลเช่นนั้น

เมื่อเครื่องถูกกำหนดให้อยู่ในสภาพโลคัล หรือควบคุมด้วยมือแล้วหากคอมพิวเตอร์ส่งคำสั่งเข้ามาเพื่อกำหนดให้เป็นอุปกรณ์ตัวรับ เครื่องจะไม่สนใจคำสั่งดังกล่าว และหากถูกสั่งให้เป็นอุปกรณ์ตัวส่ง เครื่องก็จะส่งข้อมูลอย่างใดอย่างหนึ่งออกไปให้ นั่นคือ ระบบจะให้ความสำคัญต่อสภาพโลคัลมากกว่า

บทที่ 3

แผ่นวงจรพิมพ์ของโครงงาน

แผ่นวงจรพิมพ์นี้ได้ออกแบบสำหรับต่อกับเครื่อง IBM PC ได้ออกแบบและสายพัน (Wirewarp) เสร็จเรียบร้อย สามารถที่จะเสียบลงในสล็อตของเครื่อง IBM PC ใช้ได้ทันทีโดยมีโปรแกรมสนับสนุนแผ่นวงจรพิมพ์ควบคุมการทำงานของแผ่นวงจรพิมพ์ รายละเอียดของแผนวงจรพิมพ์ในที่นี้จะอธิบายเพียงคราว ๆ ซึ่งรายละเอียดทั้งหมดสามารถอ่านได้จาก หนังสือโครงงานแผ่นวงจรเชื่อมต่อข้อมูลเอนกประสงค์ตามมาตรฐาน IEEE-488

3.1 ส่วนประกอบของแผ่นวงจรพิมพ์

ประกอบด้วย วงจรรวม (IC) 8 ตัวดังนี้

1.	วงจรรวมเบอร์ 74LS04 (HEX INVERTER)	2 ตัว
2.	วงจรรวมเบอร์ 74LS133 (13 INPUT NAND)	2 ตัว
3.	วงจรรวมเบอร์ 8255 (PROGRAMMABLE PERIPHERAL INTERFACE)	2 ตัว
4.	วงจรรวมเบอร์ SN75160A (OCTAL GENERAL-PURPOSE INTERFACE BUS TRANSCEIVER)	1 ตัว
5.	วงจรรวมเบอร์ SN75161A (OCTAL GENERAL-PURPOSE INTERFACE BUS TRANSCEIVER)	1 ตัว

3.2 การถอดรหัสเรียกใช้แผ่นวงจร

วงจรรวมเบอร์ 74LS04 และ 74LS133 จะทำหน้าที่ถอดรหัสเพื่อใช้ 8255 แต่ละตัว โดยจะถอดรหัสพอร์ทแอดเดรส (Port Address) ดังนี้

พอร์ทแอดเดรส 0310H ถึง 0313H จะถอดรหัสและเพื่อใช้ 8255(1)

พอร์ทแอดเดรส 0314H ถึง 0317H จะถอดรหัสและเรียกใช้ 8255(2)

3.3 การทำงานของ 8255

จะทำงานอยู่ในโหมดศูนย์ โดยกำหนดให้

พอร์ท A เป็นพอร์ทของสัญญาณข้อมูล 8 บิท โดยกำหนดให้เป็นได้ทั้งอินพุทพอร์ท และเอาท์พุทพอร์ท

พอร์ท B เป็นพอร์ทที่ใช้สำหรับกำหนดทิศทางของสัญญาณข้อมูล และสัญญาณควบคุม โดยจะกำหนดให้เป็น เอาท์พุทพอร์ทเท่านั้น

พอร์ท C ใช้ทั้งพอร์ท C บนและพอร์ท C ล่าง เป็นได้ทั้ง อินพุทพอร์ท และเอาพุทพอร์ท โดยจะใช้ 8255 2 ตัว เพื่อความสะดวกในการควบคุมการทำงาน

3.4 การจัดกลุ่มสัญญาณ

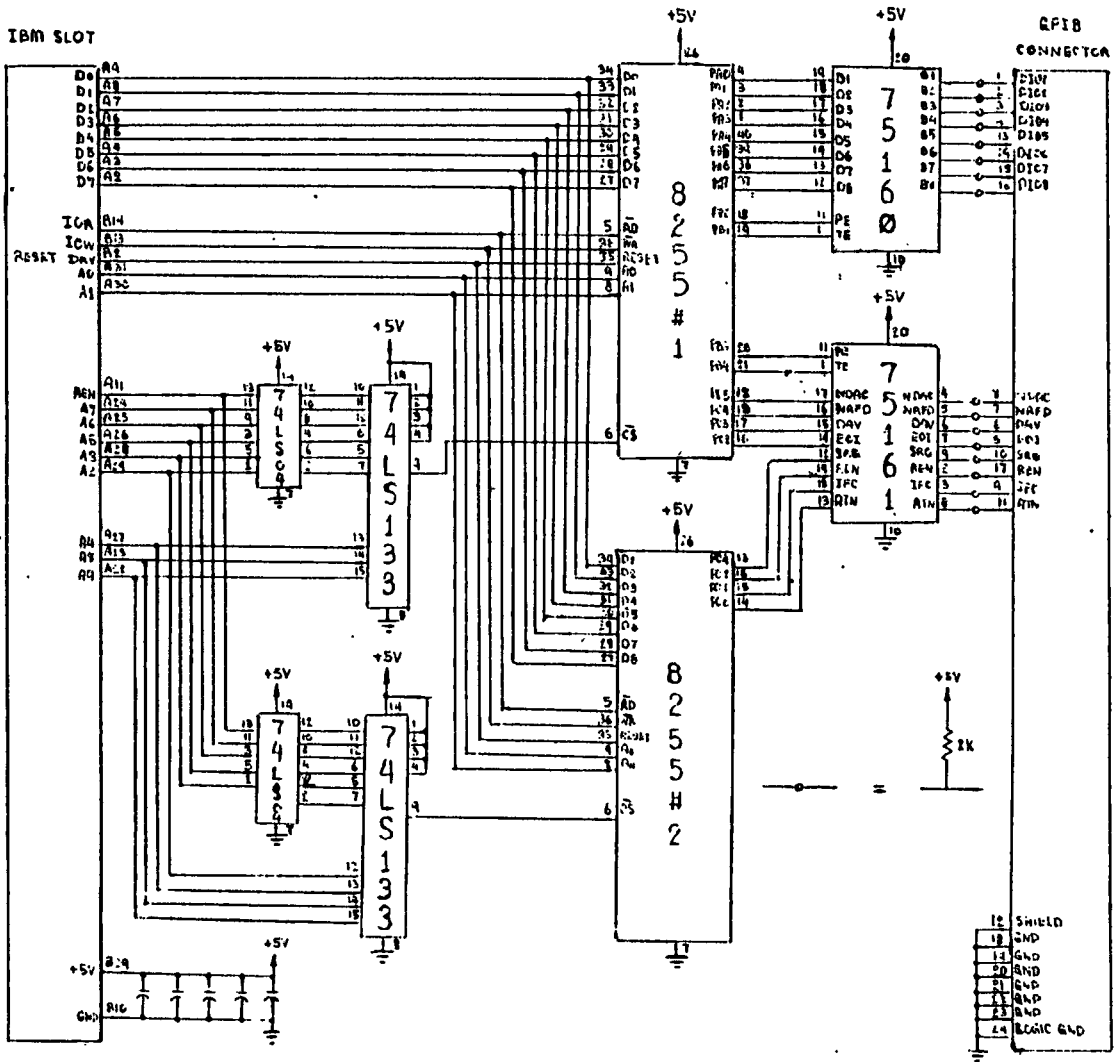
การจัดกลุ่มสัญญาณ จะพิจารณาจากทิศทางของสัญญาณที่กำหนดไว้ในวงจรรวมเบอร์ SN75160A และ SN75161A ซึ่งทำหน้าที่เป็นบัฟเฟอร์สำหรับการรับส่งข้อมูล และสัญญาณควบคุมสำหรับระบบ GPIB โดยเฉพาะ และพิจารณาถึงความสะดวกในการควบคุมการทำงานของ 8255

กลุ่มของสัญญาณต่าง ๆ

1. DIO1-DIO8 ต่อกับพอร์ท A ของ 8255(1) จะต่อผ่านวงจรรวมบัฟเฟอร์เบอร์ SN75160
2. NDAC, NRFD ต่อกับพอร์ท C บนของ 8255(1)
3. DAV, EOI ต่อกับพอร์ท C ล่างของ 8255(1)
4. SRQ ต่อกับพอร์ท C บนของ 8255(2)
5. REN, IFC, ANT ต่อกับพอร์ท C ล่างของ 8255(2)

สัญญาณกลุ่มที่ 2-5 เป็นสัญญาณควบคุมการติดต่อและการรับส่งข้อมูล โดยจะต่อผ่านวงจรรวมบัฟเฟอร์ เบอร์ SN75161A

วงจรของแผ่นวงจรเชื่อมต่อข้อมูลเอนกประสงค์นี้แสดงดังรูป 3-1



รูปที่ 3-1 แสดงวงจรไฟฟ้าของแผ่นวงจรเชื่อมต่อข้อมูลเอนกประสงค์

บทที่ 4

โปรแกรมสนับสนุนของแผ่นวงจรพิมพ์

ภาษาที่ใช้ในการเขียนโปรแกรมได้ใช้เทอร์โบปาสคาล เวอร์ชัน 5.5 โปรแกรมสนับสนุนที่เขียนไว้ทั้งหมดมี 4 โปรแกรม คือ

1. โปรแกรม GPIB1.PAS เป็นโปรแกรมจัดการระบบที่ต้องใช้ร่วมกับโปรแกรม USER.INC เวลาใช้งานต้องใช้โปรแกรม TURBO PASCAL VERSION 5.5 ชื่อโปรแกรม TURBO.EXE มาเป็นตัวคอมไพล์โปรแกรม GPIB1.PAS ก่อนใช้งานทุกครั้ง
2. โปรแกรม GPIB2.PAS เป็นโปรแกรมที่ทำงานเหมือน GPIB1.PAS แต่จะแสดงสภาวะการทำงานต่าง ๆ บันทึกขั้นตอนแต่ละคำสั่งที่เขียนใน USER.INC
3. โปรแกรม GPIB3.EXE เป็นโปรแกรมที่จะเขียนเป็นระบบเมนู เวลาใช้งานไม่ต้องใช้โปรแกรม TURBO.EXE มาคอมไพล์ใหม่เหมือนกับสองโปรแกรมแรกที่กล่าวมาแล้ว ที่โปรแกรมนี้จะมีการแสดงสภาวะการทำงานต่าง ๆ ตามเมนูที่เราเลือก
4. โปรแกรม GPIB4.EXE เป็นโปรแกรมที่สร้างขึ้นเหมือนกับโปรแกรม GPIB3.EXE แต่จะไม่แสดงรายละเอียดของสภาวะการทำงานต่าง ๆ ทำให้ทำงานได้รวดเร็วขึ้นมาก

ในบทนี้จะอธิบายการทำงานและการใช้งานของโปรแกรม GPIB1.PAS (GPIB2.PAS มีการทำงานและการใช้งานเหมือน GPIB1.PAS จึงไม่อธิบายอีก) ส่วน GPIB4.EXE จะอธิบายการใช้งานในบทที่ 5 (GPIB3.EXE มีการทำงานและการใช้งานคล้าย GPIB4.EXE ดังนั้นจึงไม่อธิบาย)

โปรแกรมสนับสนุน GPIB1.PAS นี้แบ่งออกเป็น 2 ส่วน คือ

4.1 ส่วนของโปรแกรมจัดการระบบ (System Program)

4.2 ส่วนของโปรแกรมของผู้ใช้งาน (User Program)

รายละเอียดมีดังต่อไปนี้

4.1 ส่วนของโปรแกรมจัดการระบบ (System Program)

ส่วนของโปรแกรมจัดการระบบเขียนเก็บไว้ในไฟล์ชื่อ GPIB1.PAS ซึ่งเป็นส่วนที่ใช้ในการควบคุมการทำงานของแผ่นวงจรให้เป็นไปตามมาตรฐาน GPIB โดยผู้ใช้ไม่จำเป็นต้องเข้ามาแก้ไขหรือเพิ่มเติมโปรแกรมในส่วนนี้ เพียงแต่ศึกษาการใช้แต่ละ Procedure ว่าใช้ทำอะไรและเรียกใช้ให้ถูกต้องก็พอ

รายละเอียดของโปรแกรมจัดการระบบ มีดังนี้ คือ

ส่วนข้อกำหนด (Declaration Part) จะเป็นการกำหนดค่าต่าง ๆ และตัวแปรสำหรับการนำไปใช้ในส่วนอื่นของโปรแกรม

ส่วนถ้อยแถลง (Statement Part) จะแบ่งออกเป็นหลาย ๆ Procedure ทำงานในหน้าที่ต่าง ๆ กัน เพื่อความสะดวกในการใช้งาน ดังมีรายละเอียดการทำงานของแต่ละ Procedure มีดังต่อไปนี้

```
Procedure Address (Addr : integer ; var MLA, MTA : byte) ;
```

ใน Procedure นี้จะเป็นการแปลงค่าแอดเดรสของอุปกรณ์ที่ตั้งค่าไว้ในระบบจากเลขฐานสิบ (เพื่อความสะดวกของผู้ใช้) ไปเป็นค่า MLA และ MTA (ซึ่งเป็นค่าที่กำหนดไว้ในมาตรฐาน) แล้วส่งค่าที่ได้คือ MLA (My Listen Address) และ MTA (My Takl Address) กลับไปยัง Procedure ต่าง ๆ ที่เรียกใช้มันมี Flow Chart กลับไปยัง Procedure ต่าง ๆ ที่เรียกใช้มัน มี Flow Chart การทำงานดังรูปที่ 4-1

```
Procedure Source_Operation (Data : byte) ;
```

ใน Procedure นี้ เป็นการทำขบวนการ Source Operation สำหรับการส่งข้อความคือ Data (ข้อความ 1 ไบท์) มี Flow Chart การทำงานดังรูปที่ 4-2

Procedure Init_Source ;

ใน Procedure นี้จะทำการกำหนดโหมดการทำงานของ 8255, 75160A และ 75161A ในขณะเริ่มต้น ซึ่งเป็นส่วนที่ผู้ใช้จำเป็นจะต้องเรียกใช้เป็นอันดับแรกในโปรแกรมของผู้ใช้งาน มี Flow Chart การทำงานดังรูปที่ 4-3

Procedure Send_Command (Addr : Integer ; Command : byte) ;

ใน Procedure นี้จะใช้สำหรับการส่งคำสั่งเพื่อควบคุมอุปกรณ์ให้เป็นไปตามต้องการ มี Flow Chart การทำงานดังรูปที่ 4-4

Procedure Send_Message (Addr : Integer ; M : DataFormat) ;

ใน Procedure จะใช้สำหรับการส่งข้อความ (หลาย ๆ ไบท์) ที่ใช้ในการควบคุมอุปกรณ์ให้เป็นไปตามต้องการ มี Flow Chart การทำงานดังรูปที่ 4-5

Procedure Acceptor_Operation (var DByte : byte) ;

ใน Procedure เป็นการทำขบวนการ Acceptor Operation สำหรับการรับข้อความทีละ 1 ไบท์ มี Flow Chart การทำงานดังรูปที่ 4-6

Procedure Get_Data (Addr:Inter;var D:DataFormat;Num:integer);

ใน Procedure นี้ใช้ในการรับข้อมูลหลาย ๆ ไบท์กลับสู่คอมพิวเตอร์ (ทำหน้าที่เป็น Controller) และอ่านค่าเวลาขณะนั้นเข้ามาด้วย มี Flow Chart การทำงานดังรูปที่ 4-7

Procedure IFC ;

ใน Procedure นี้ใช้สำหรับเคลียร์สัญญาณควบคุมทั้งหมด ให้กลับสู่สภาวะเริ่มต้น มี Flow Chart การทำงานดังรูปที่ 4-8

Procedure DCL ;

ใน Procedure นี้ใช้สำหรับทำให้อุปกรณ์ทุกตัวที่ต่ออยู่ในระบบ GPIB กลับสู่

สภาวะเริ่มต้น มี Flow Cahrt การทำงานดังรูปที่ 4-9

Procedure SPolling (Addr : interger ; var DByte : byte) ;

ใน Procewdure นี้ใช้สำหรับการทำขบวนการตรวจสอบแบบอนุกรม (Serial Polling) มี Flow Chart การทำงานดังรูปที่ 4-10

Procedure Save_Data (D : DataFormat) ;

ใน Procedure นี้ใช้สำหรับการนำข้อมูลที่ได้จากอุปกรณ์และเวลาขณะที่รับข้อมูลเก็บเข้าไฟล์ที่ต้องการไว้ในแผ่นดิส์เกต์ โดยไฟล์ข้อมูลเหล่านี้จะมีชื่อตามการตั้งชื่อของผู้ใช้มี Flow Chart การทำงานดังรูปที่ 4-11

($I USER.inc)

ในส่วนนี้ จะเป็นส่วนที่ใช้ในการติดต่อกับโปรแกรมที่ผู้ใช้เขียนขึ้นโดยเก็บไว้ในไฟล์ชื่อ USER.INC

4.2 ส่วนของโปรแกรมของผู้ใช้งาน (User Program)

ส่วนของโปรแกรมของผู้ใช้งานนี้ ผู้ใช้งานจะต้องเขียนโปรแกรมขึ้นมาเองโดยเขียนอยู่ในไฟล์ชื่อ USER.INC เท่านั้น ในการเขียนโปรแกรมส่วนนี้ ผู้ใช้จะต้องศึกษา และจะต้องเข้าใจระบบบัสมาตรฐาน GPIB นี้พอสมควร

รูปแบบคำสั่งที่ได้กำหนดไว้ในส่วนของโปรแกรมจัดการระบบ ผู้ใช้สามารถเรียกใช้ได้โดยอาศัยรูปแบบต่อไปนี้

Device Clear

รูปแบบ DCL;

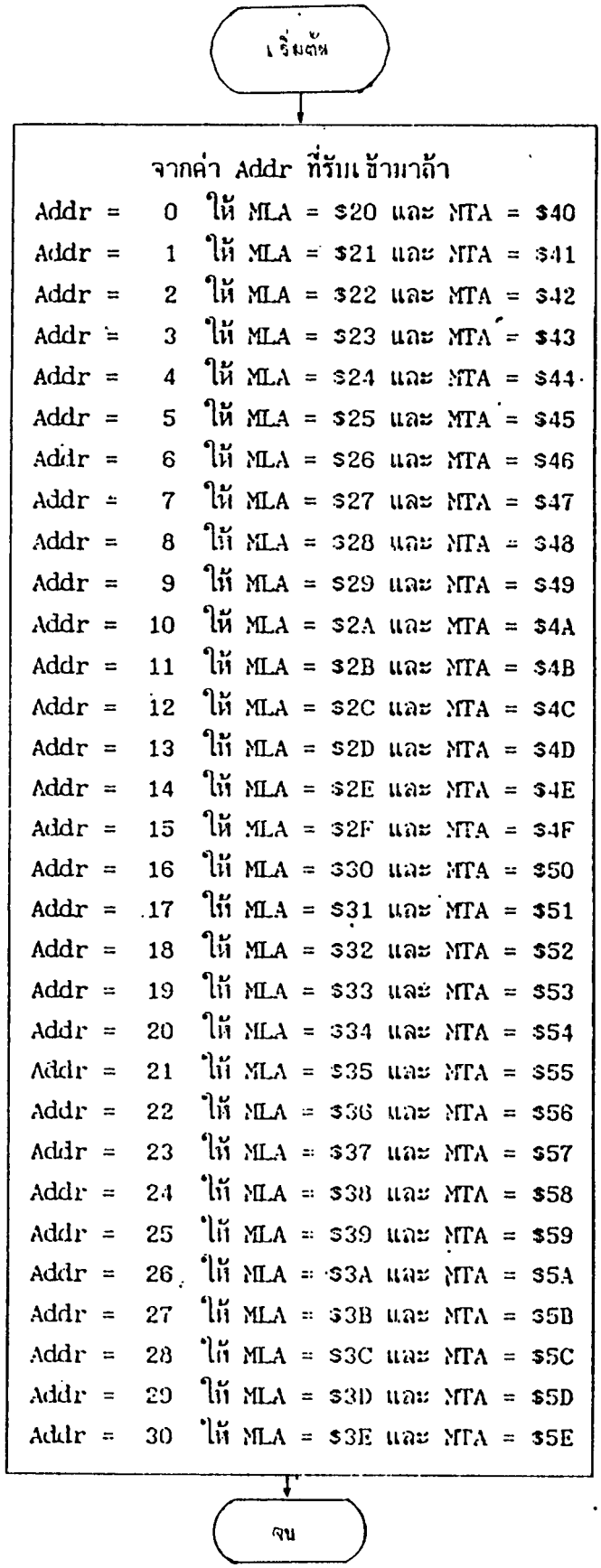
จุดประสงค์ คำสั่งนี้ใช้การสั่งให้อุปกรณ์ทุกตัวในระบบกลับสู่สภาวะเริ่มต้น

สัญญาณในบัส REN = ROW

IFC = HIGH

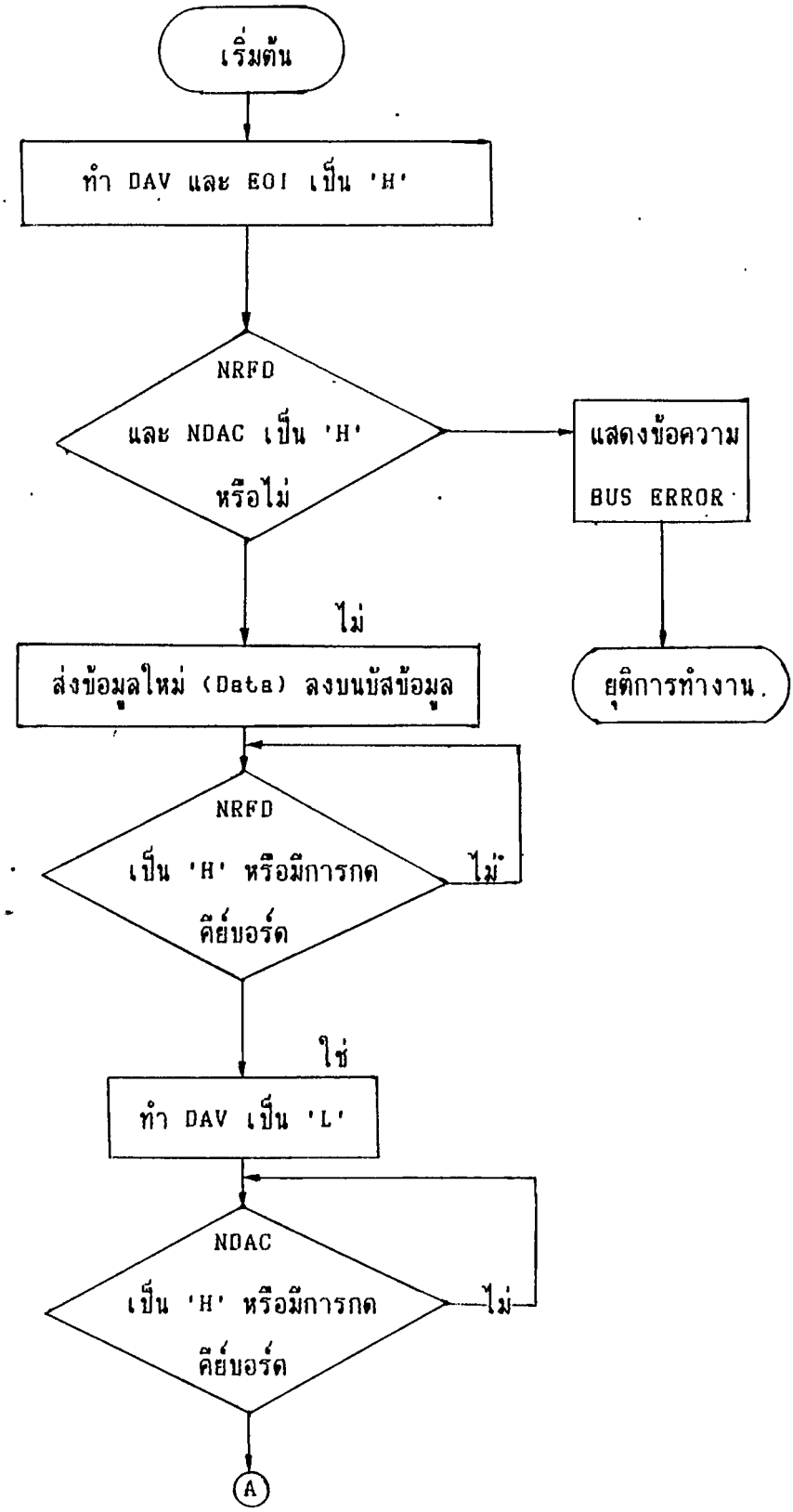
รูปที่ 4-1 แสดง Flow Chart และ Procedure Address

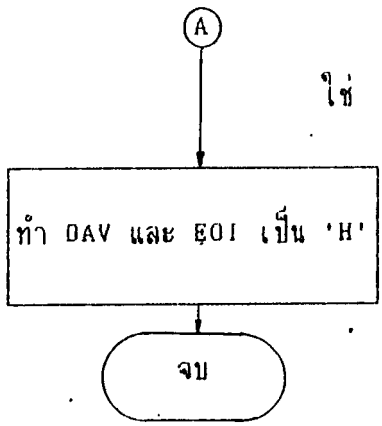
```
Procedure Address (Addr:interger;var MLA,MTA:byte);
```



รูปที่ 4-2 แสดง Flow Chart และ Procedure Source_Operation

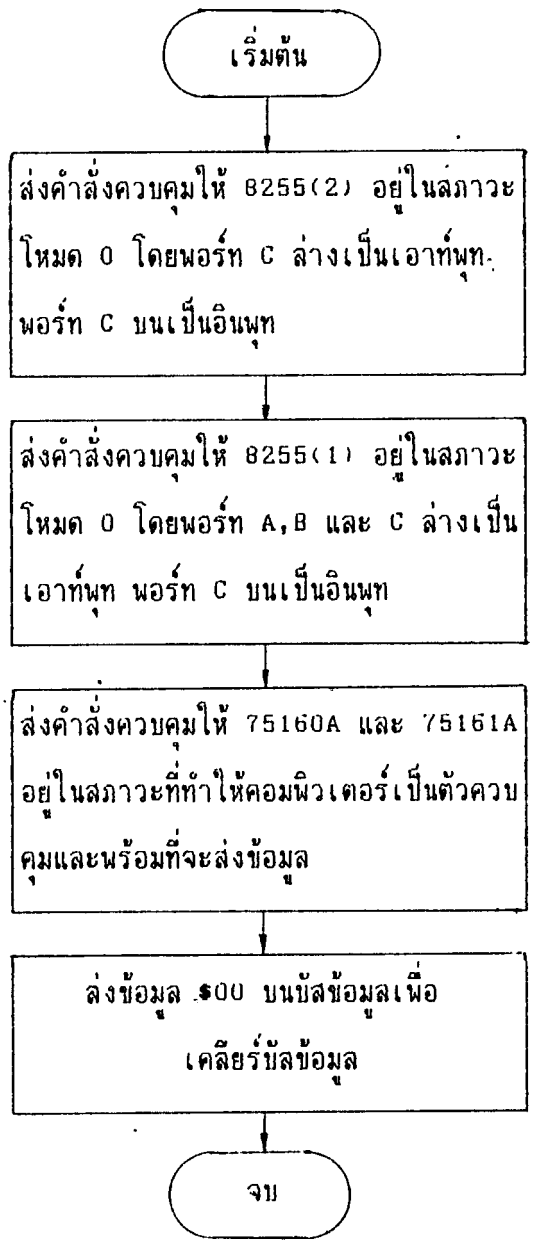
```
Procedure Source_Operation (Data:byte);
```





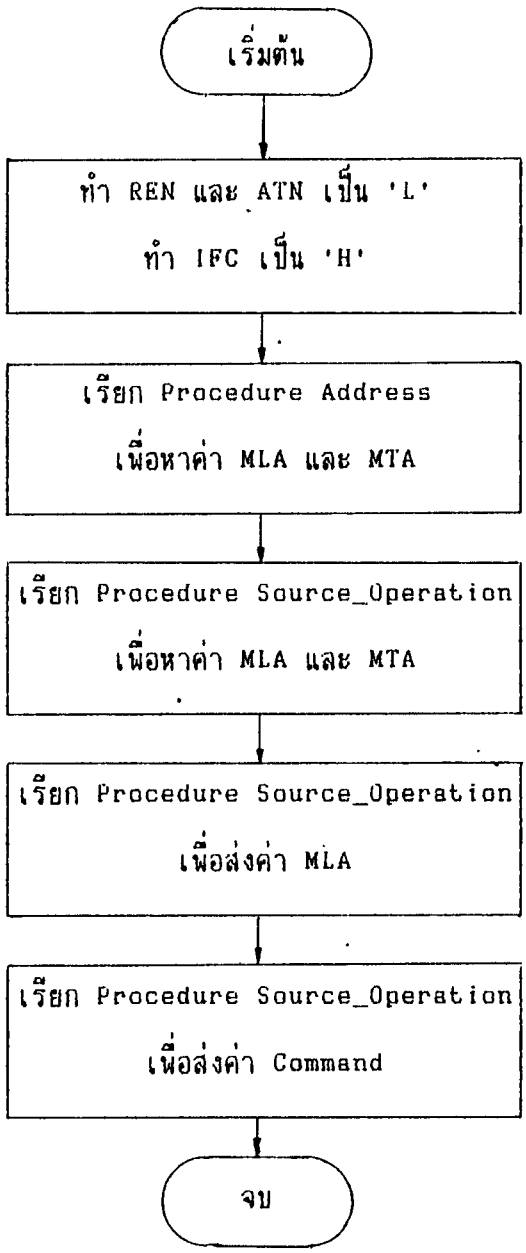
รูปที่ 4-3 แสดง Flow Chart และ Procedure Init_Source

```
Procedure Init_source;
```



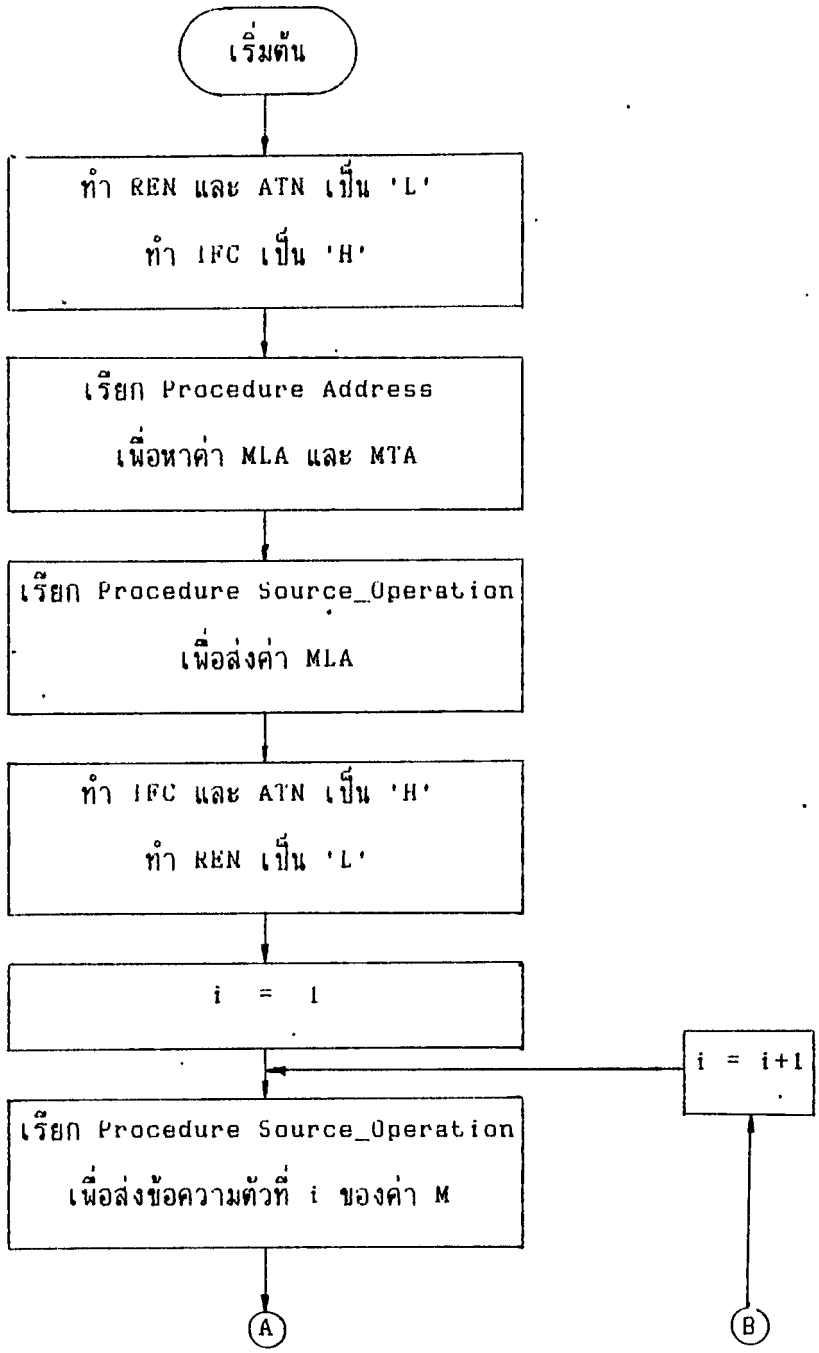
รูปที่ 4-4 แสดง Flow Chart ของ Procedure Send_Command

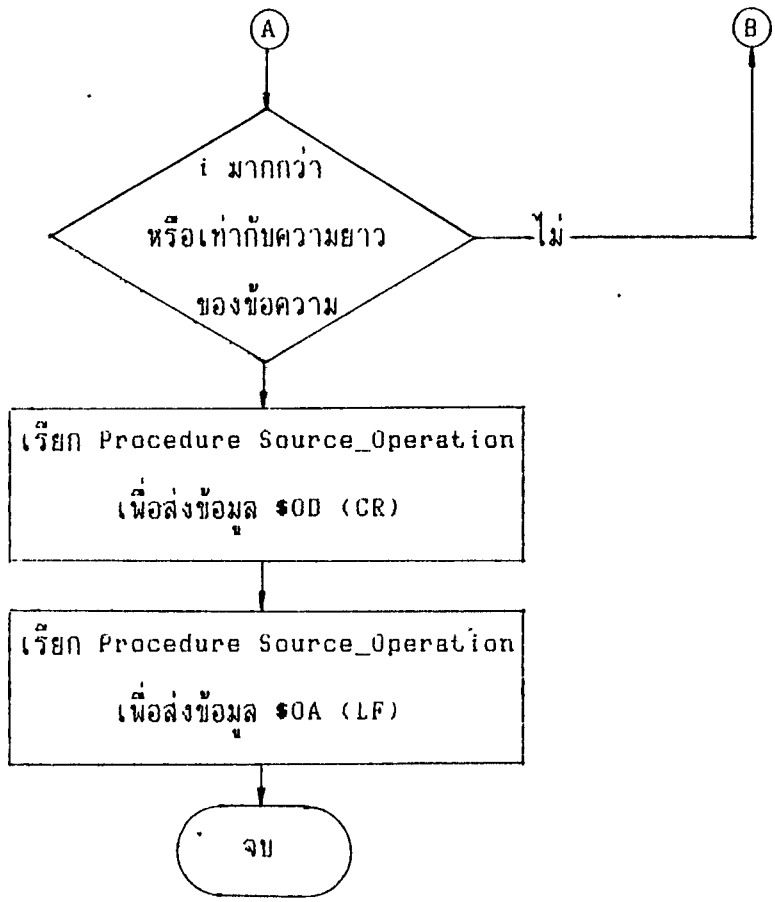
```
Procedure Send_Command (Addr:integer;Command:byte);
```



รูปที่ 4-5 แสดง Flow Chart และ Procedure Send_Message

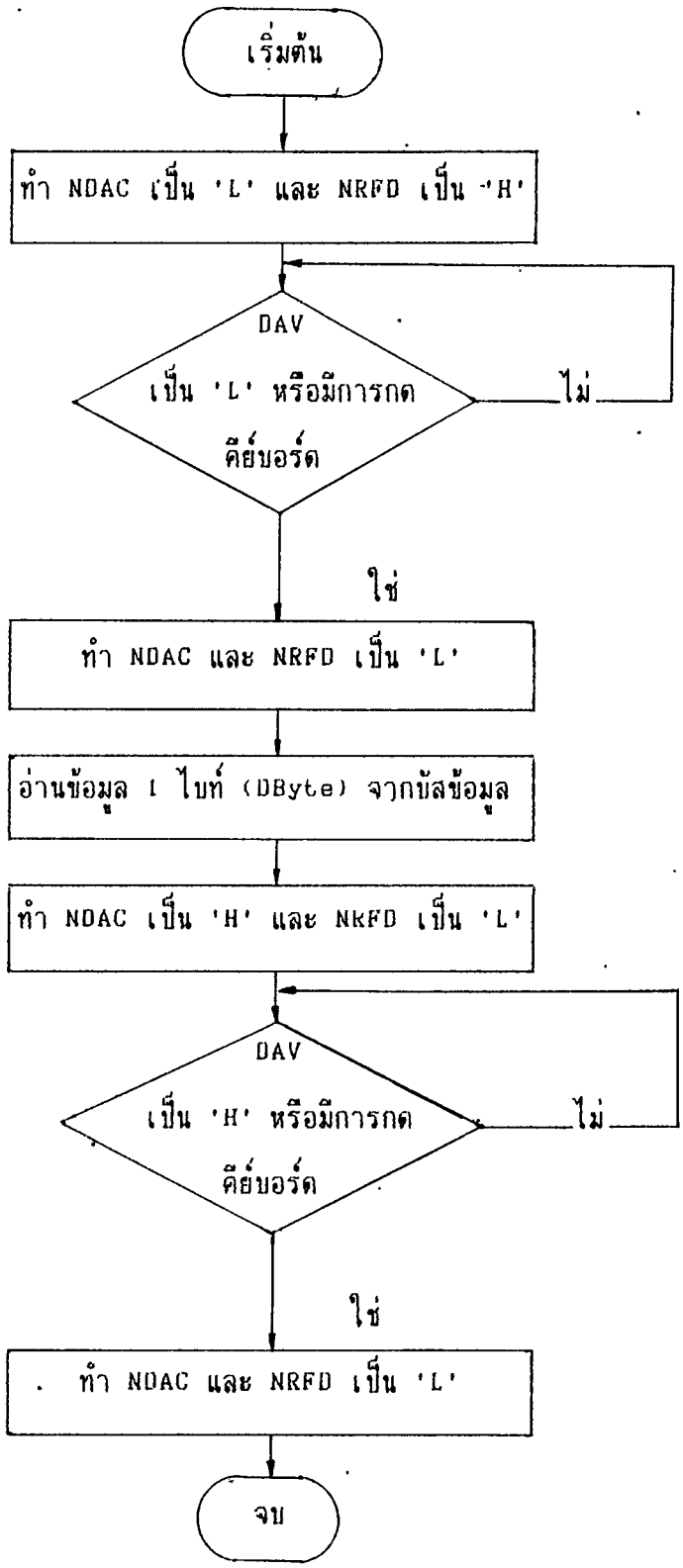
```
Procedure Send_message (Addr:integer;M:DataFormat);
```




A
B
i มากกว่า
หรือเท่ากับความยาว
ของข้อความ
ไม่
เรียก Procedure Source_Operation
เพื่อส่งข้อมูล $0D (CR)
เรียก Procedure Source_Operation
เพื่อส่งข้อมูล $0A (LF)
จบ

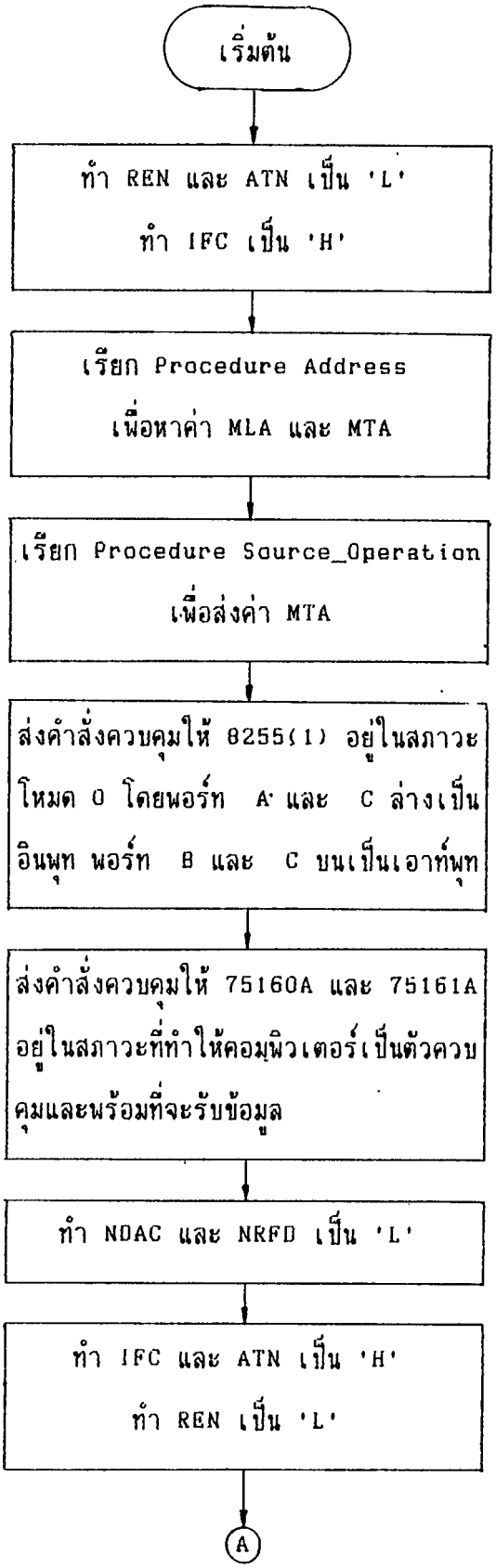
รูปที่ 4-6 แสดง Flow Chart และ Procedure Acceptor_Operation

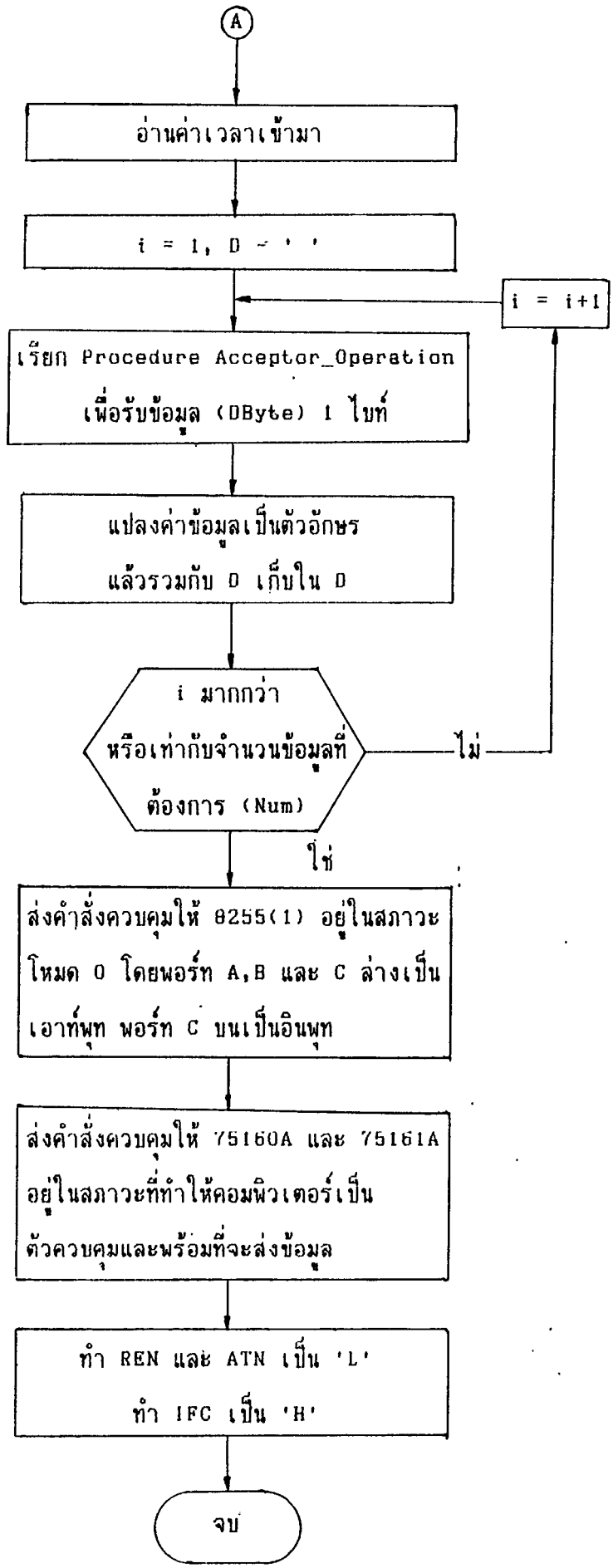
```
Procedure Acceptor_Operation (var DByte:byte);
```



รูปที่ 4-7 แสดง Flow Chart ของ Procedure Get_Data

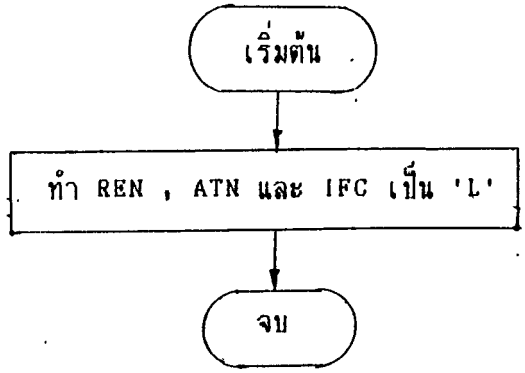
```
Procedure Get_Data (Addr:integer;var  D:DataFormat;Num:integer);
```




A
อ่านค่าเวลาเข้ามา
i = 1, D = ' '
i = i+1
เรียก Procedure Acceptor_Operation
เพื่อรับข้อมูล (DByte) 1 ไบท์
แปลงค่าข้อมูลเป็นตัวอักษร
แล้วรวมกับ D เก็บใน D
i มากกว่า
หรือเท่ากับจำนวนข้อมูลที่
ต้องการ (Num)
ไม่
ใช่
ส่งคำสั่งควบคุมให้ 8255(1) อยู่ในสภาวะ
โหมด 0 โดยพอร์ท A,B และ C ล่างเป็น
เอาท์พุท พอร์ท C บนเป็นอินพุท
ส่งคำสั่งควบคุมให้ 75160A และ 75161A
อยู่ในสภาวะที่ทำให้คอมพิวเตอร์เป็น
ตัวควบคุมและพร้อมที่จะส่งข้อมูล
ทำ REN และ ATN เป็น 'L'
ทำ IFC เป็น 'H'
จบ

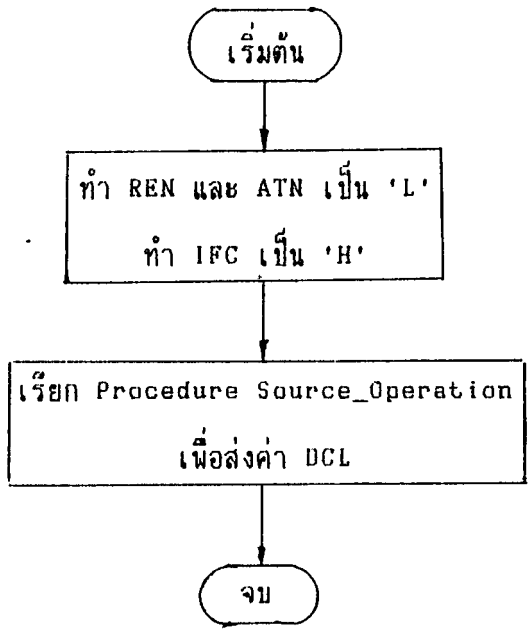
รูปที่ 4-8 แสดง Flow Chart ของ Procedure IFC

Procedure IFC;



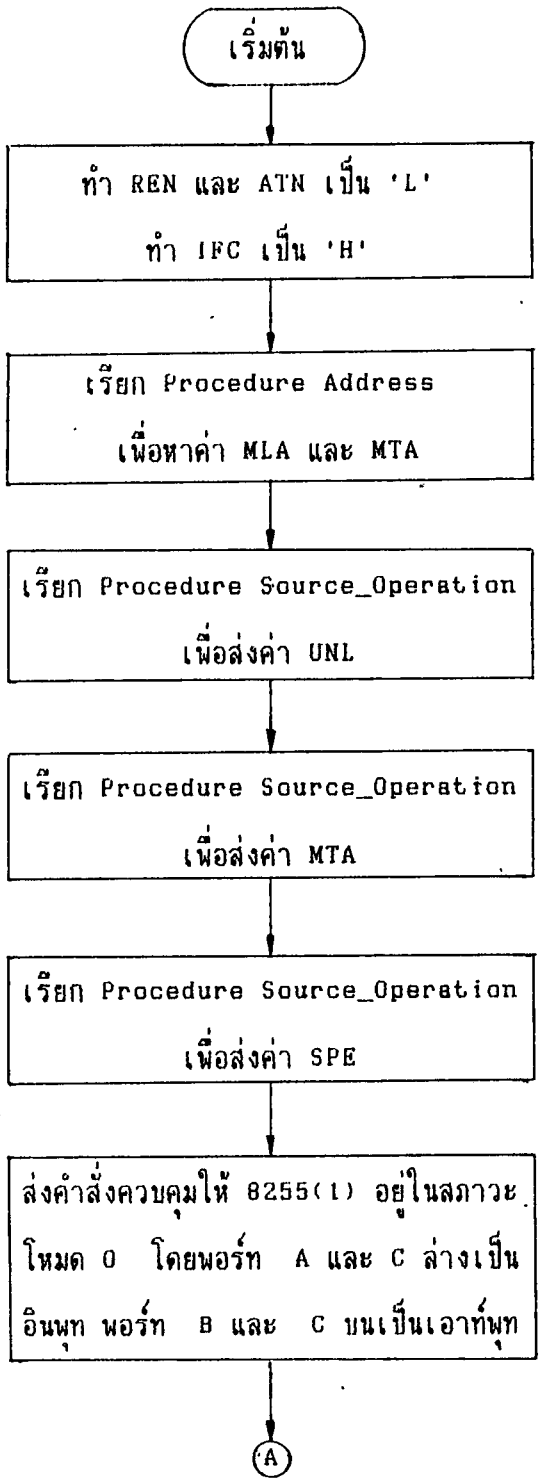
รูปที่ 4-9 แสดง Flow Chart ของ Procedure DCL;

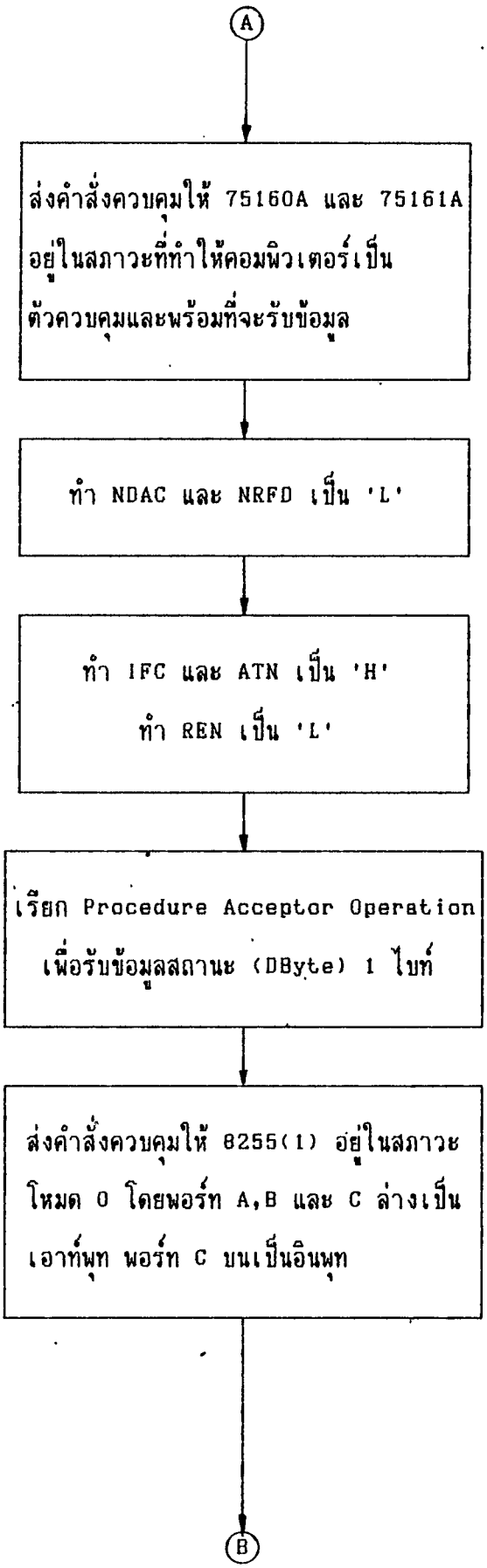
Procedure DCL;

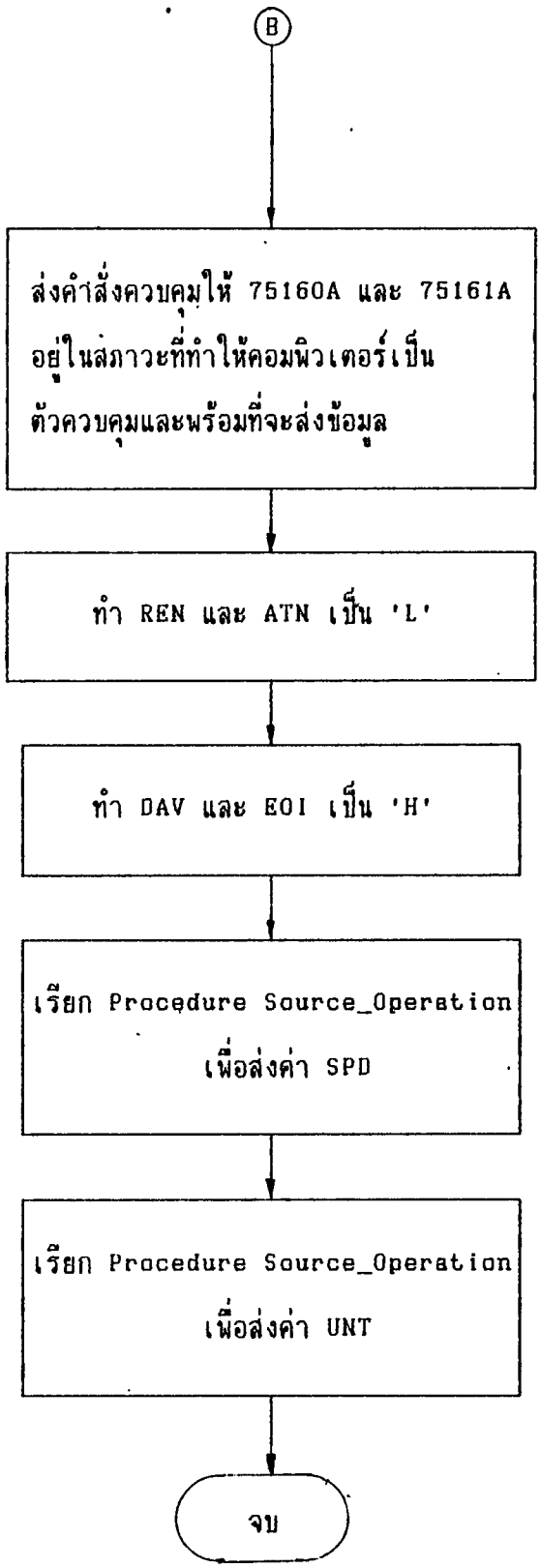


รูปที่ 4-10 แสดง Flow Chart ของ Procedure Spolling

Procedure Spolling (Addr:integer;var DByte:byte);

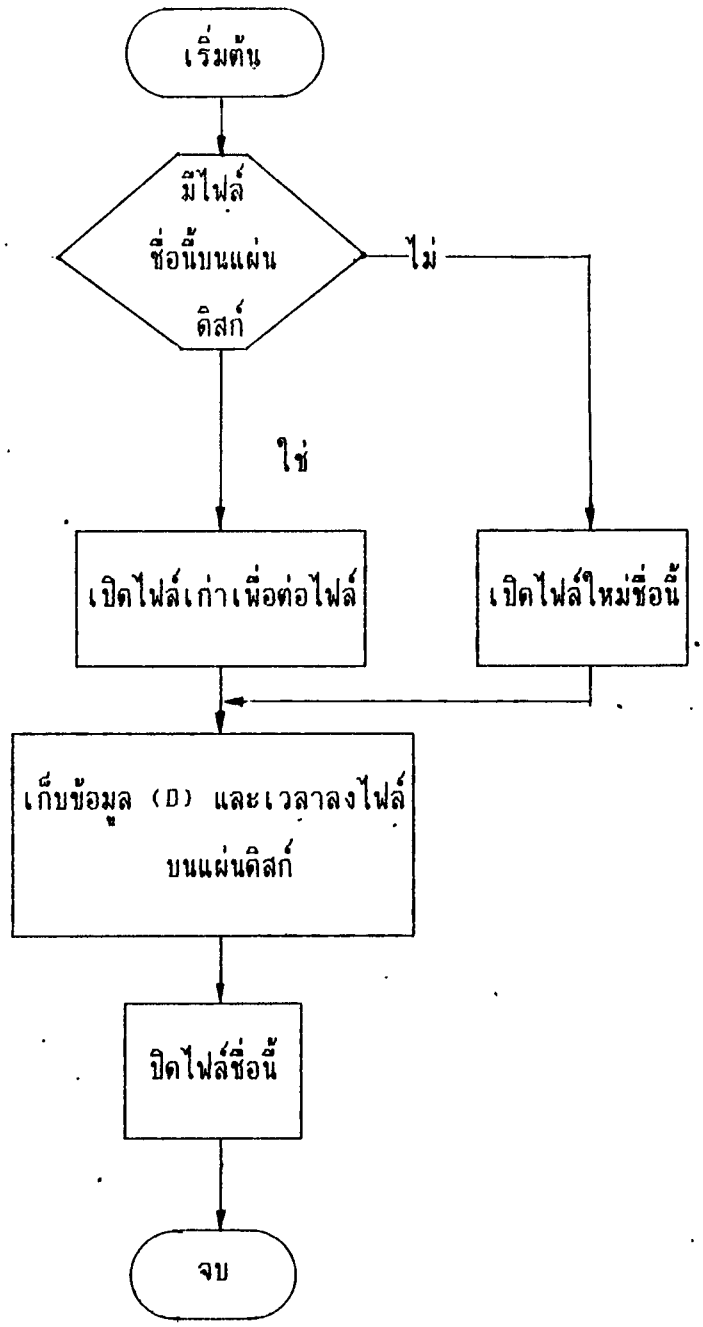



A
ส่งคำสั่งควบคุมให้ 75160A และ 75161A อยู่ในสภาวะที่ทำให้คอมพิวเตอร์เป็นตัวควบคุมและพร้อมที่จะรับข้อมูล
ทำ NDAC และ NRFD เป็น 'L'
ทำ IFC และ ATN เป็น 'H'
ทำ REN เป็น 'L'
เรียก Procedure Acceptor Operation เพื่อรับข้อมูลสถานะ (DByte) 1 ไบท์
ส่งคำสั่งควบคุมให้ 8255(1) อยู่ในสภาวะโหมด 0 โดยพอร์ท A,B และ C ล่างเป็นเอาท์พุท พอร์ท C บนเป็นอินพุท
B


B
ส่งคำสั่งควบคุมให้ 75160A และ 75161A อยู่ในสภาวะที่ทำให้คอมพิวเตอร์เป็นตัวควบคุมและพร้อมที่จะส่งข้อมูล
ทำ REN และ ATN เป็น 'L'
ทำ DAV และ EOI เป็น 'H'
เรียก Procedure Source_Operation เพื่อส่งค่า SPD
เรียก Procedure Source_Operation เพื่อส่งค่า UNT
จบ

รูปที่ 4-11 แสดง Flow Chart ของ Procedure Save_Data

```
Procedure Save_Data(D:DataFormat;Filename:str30);
```



บทที่ 5

วิธีการใช้โปรแกรม GPIB4.EXE

เนื่องด้วยทางผู้จัดทำเห็นว่าโปรแกรม GPIB1 มีความยุ่งยากแก่การใช้งานเพราะการใช้งานในแต่ละครั้งต้องให้ผู้ใช้เขียนโปรแกรม USER.INC พร้อมทั้งต้องคอมไพล์ใหม่ทั้งโปรแกรม ถ้าผู้ใช้ไม่มีความรู้ภาษาปาสคาลก็ไม่สามารถใช้งานได้ ดังนั้นเพื่อความสะดวกแก่ผู้ที่นำไปใช้งาน ทางผู้จัดทำจึงได้คิดทำเป็นโปรแกรมสำเร็จรูปขึ้นมาซึ่งได้ตั้งชื่อโปรแกรมนี้ว่า GPIB4.EXE

โปรแกรม GPIB4 ประกอบ 4 เมนูหลัก (MAIN MENU) ดังนี้คือ

1. GPIB MENU
2. FILE MENU
3. OPTION MENU
4. QUIT MENU

การเลือกใช้แต่ละเมนูหลัก ทำให้โดยกดคีย์ลูกศรซ้ายหรือขวาเพื่อเลื่อนแถบสีไปยังเมนูหลักที่ต้องการ (ซึ่งในแต่ละเมนูหลักที่แถบสีปรากฏอยู่นั้น จะมีรายละเอียดแสดงอยู่ที่บรรทัดล่างสุดของจอ) แล้วกดคีย์ ENTER หรืออีกวิธีหนึ่งคือ กดตัวอักษรตัวแรกของแต่ละเมนูหลักที่ต้องการเลือกใช้ เมื่อขณะที่เลือกใช้เมนูหลักใดแล้ว ก็สามารถเลือกใช้เมนูหลักอื่น ๆ ได้โดยกดคีย์ลูกศรซ้ายหรือขวา และขณะที่กำลังใช้เมนูหลักหรือเมนูย่อยใด ๆ ผู้ใช้สามารถกดคีย์ ESC เพื่อออกจากเมนูหลักหรือเมนูย่อยใด ๆ ได้ทุกเวลาที่ต้องการ ในเมนูย่อยนั้นบางครั้งจะมีการให้ใส่ข้อมูลลงไป ผู้ใช้สามารถใส่ข้อมูลลงไปได้เลย และถ้าต้องการแก้ไขก็สามารถแก้ไขได้โดย กดคีย์ Ctrl และอักษร Y พร้อมกันเพื่อลบทั้งบรรทัด หรือกดคีย์ลูกศรขึ้นหรือลง คีย์ใดก็ได้ 1 คีย์ ตัวอักษรที่อยู่บรรทัดนั้นจะกระพริบ ผู้ใช้สามารถใส่ข้อมูลลงไปใหม่ได้เลยโดยมันจะลบตัวอักษรเก่าออกจากบรรทัดนั้นเอง ถ้าผู้ใช้ต้องการลบอักษรตัวสุดท้ายสามารถทำได้โดยกดคีย์ Ctrl และอักษร H พร้อมกัน หรือคีย์ Backspace และถ้าผู้ใช้ต้องการแทรกข้อ

ความสามารถทำได้ โดยเลื่อนคีย์ลูกศรซ้ายหรือขวาไปยังตำแหน่งที่ต้องการแทรก แล้วกดตัวอักษรที่ต้องการแทรกลงไปได้เลย เมื่อผู้ใช้กรอกข้อมูลเสร็จก็ให้กดคีย์ ENTER ทุกครั้ง เพื่อให้เครื่องรับข้อมูลเข้าไป

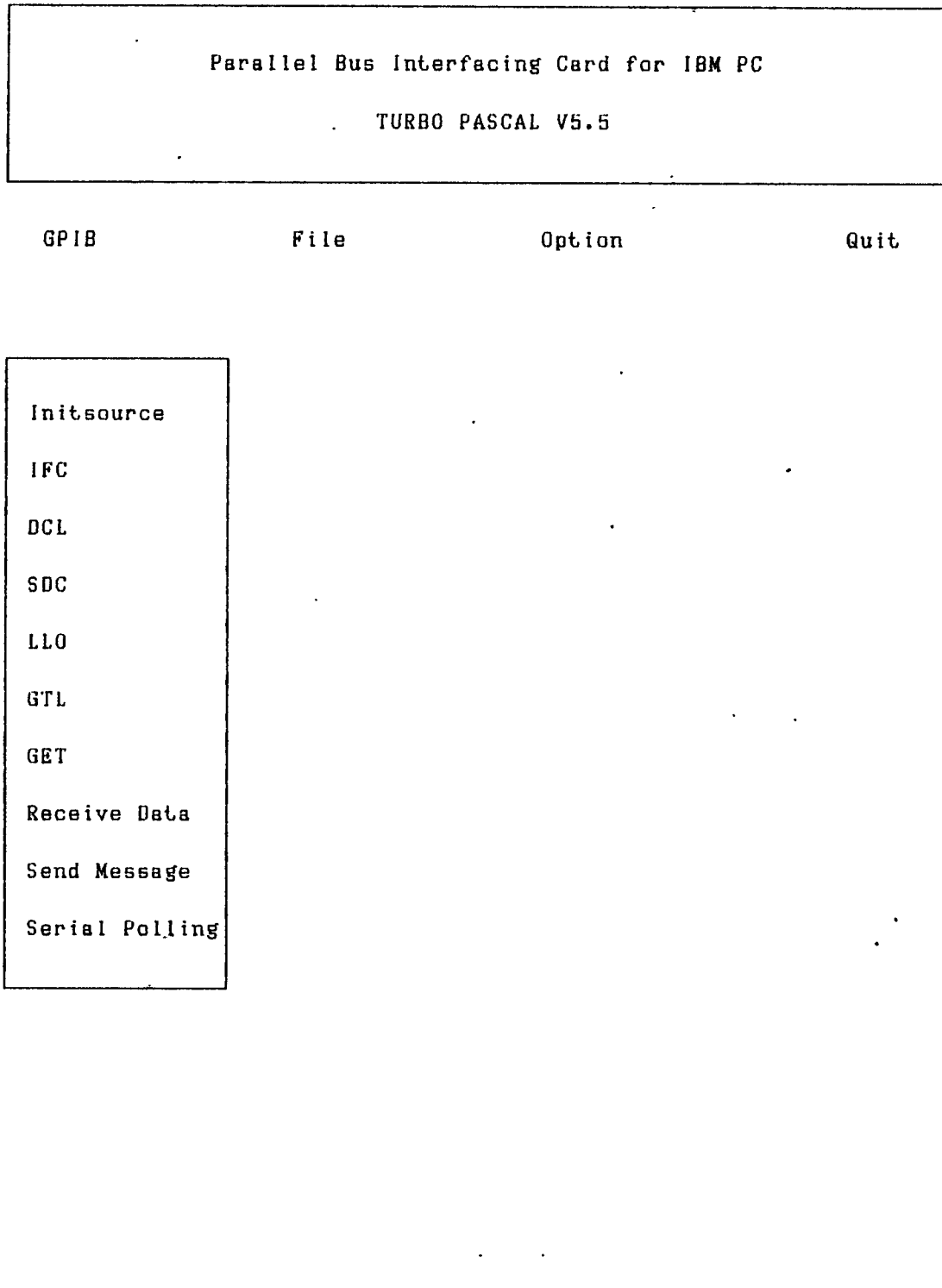
รายละเอียดของแต่ละเมนูหลักมีดังต่อไปนี้

5.1 GPIB MENU เมื่อเลือก GPIB MENU จะมีหน้าต่าง (WINDOW) เกิดขึ้นใหม่ใต้คำว่า GPIB ดังรูป 5-1 ภายใน GPIB MENU จะประกอบด้วยคำสั่งต่าง ๆ ดังนี้คือ

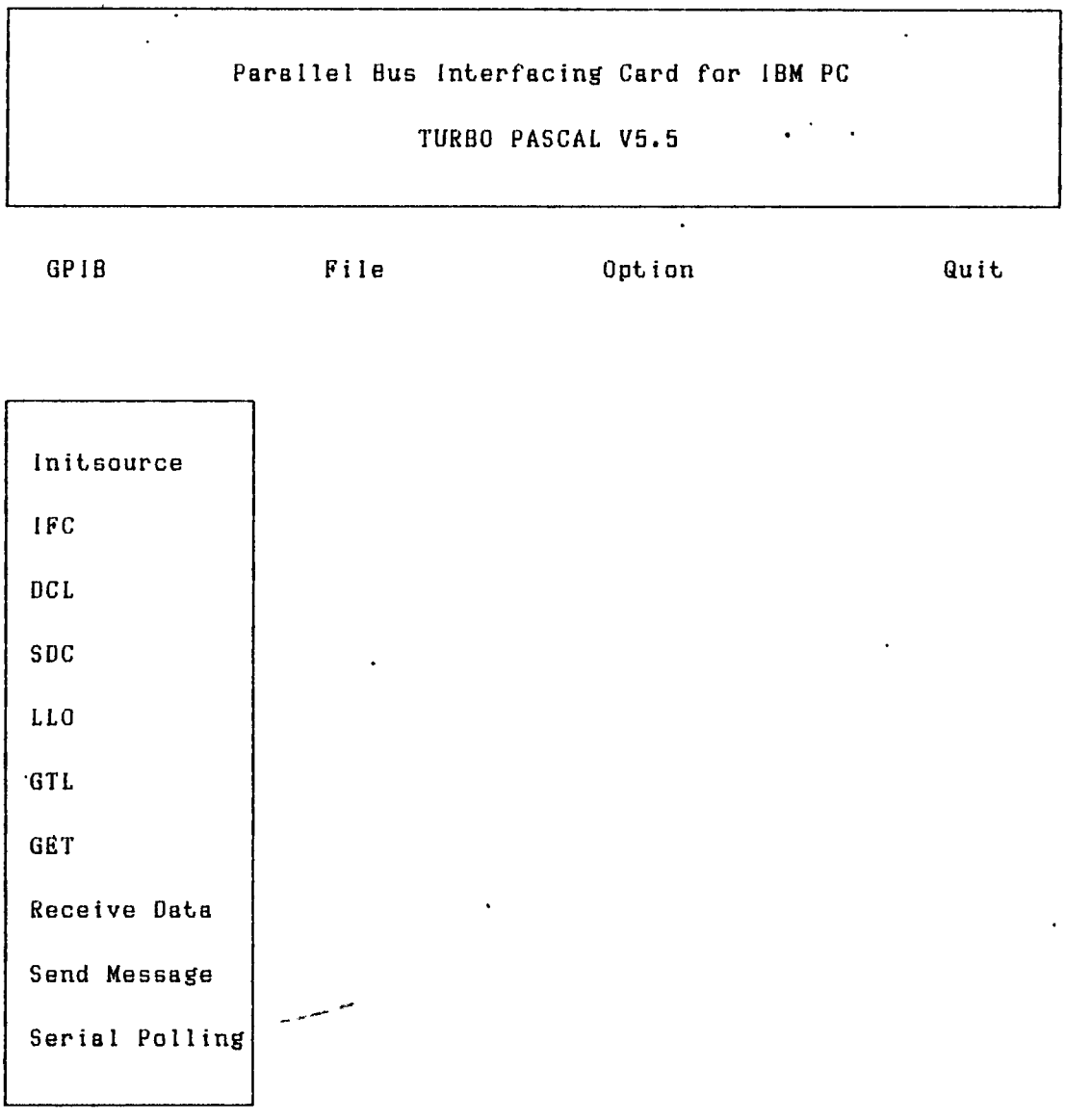
5.1.1 Initsource สามารถเลือกใช้โดยการ เลื่อนลูกศรขึ้นหรือลงหรือกดอักษร I ก็ได้ เพื่อให้แถบสีมาที่คำสั่งนี้ แล้วกดคีย์ ENTER ซึ่งที่คำสั่งนี้จะไปเรียกโปรแกรมย่อย Init_Source ให้ทำงาน

5.1.2 IFC สามารถเลือกใช้โดยการ เลื่อนลูกศรขึ้นหรือลง หรือกดอักษร I ก็ได้เพื่อให้แถบสีมาที่คำสั่งนี้ แล้วกดคีย์ ENTER ซึ่งที่คำสั่งนี้จะไปเรียกโปรแกรมย่อย IFC ให้ทำงาน

5.1.3 DCL สามารถเลือกใช้โดยการ เลื่อนลูกศรขึ้นหรือลง หรือกดอักษร D ก็ได้เพื่อให้แถบสีมาที่คำสั่งนี้ แล้วกดคีย์ ENTER ซึ่งที่คำสั่งนี้จะไปเรียกโปรแกรมย่อย DCL ให้ทำงาน ถ้ามีการเกิด BUS ERROR ขึ้น จะมีเสียงเตือนพร้อมทั้งมีตัวอักษร "BUS ERROR !" ปรากฏที่มุมล่างทางขวาของจอ ดังรูปที่ 5-2 และจะยกเลิกคำสั่ง DCL

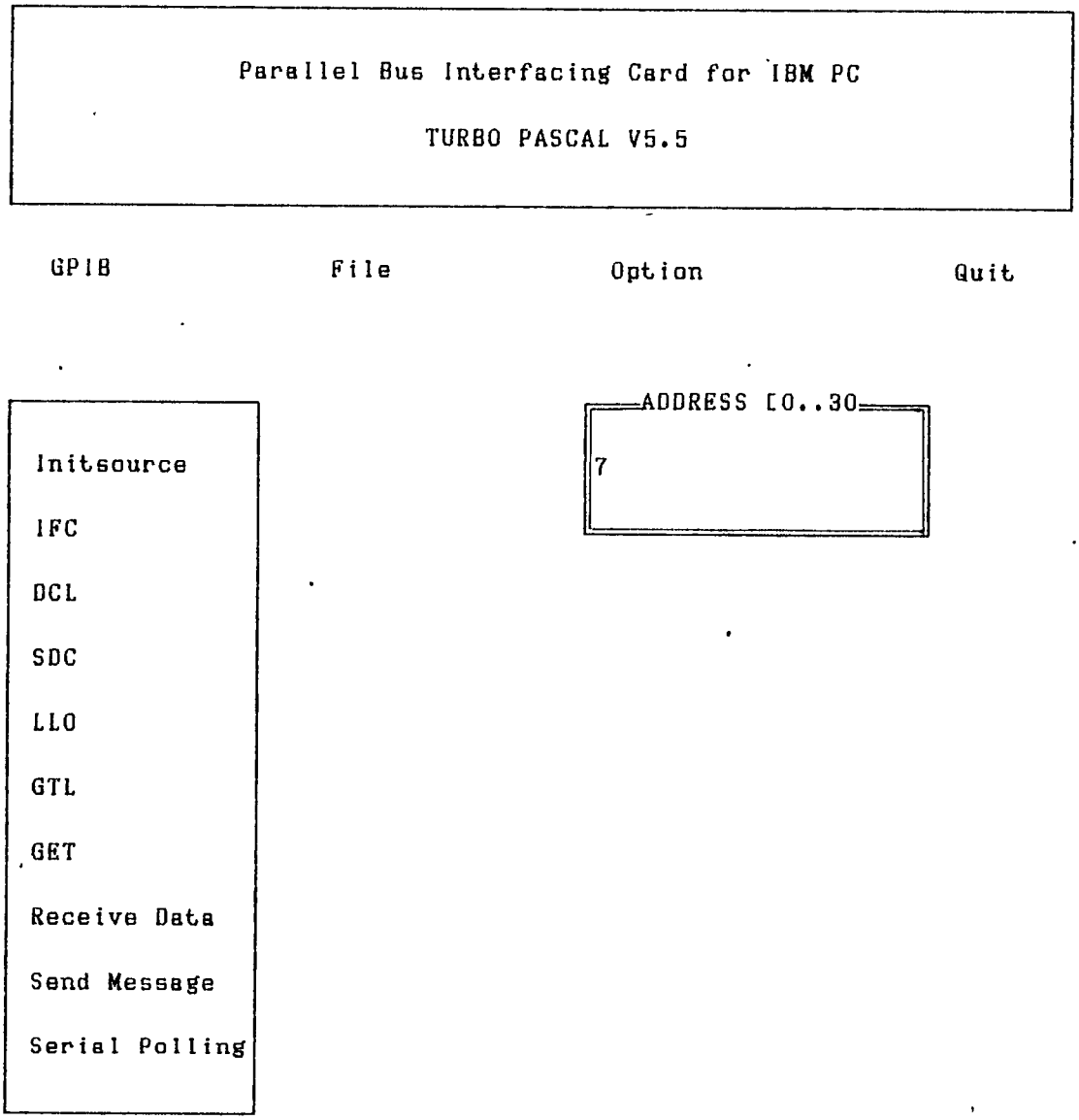


รูปที่ 5.1 แสดงภาพบนจอเมื่อเลือก GPIB MENU



รูปที่ 5.2 แสดงภาพบนจอกรณีเกิด BUS ERROR

5.1.4 SCD สามารถเลือกใช้โดยการ เลื่อนลูกศรขึ้นหรือลง หรือกดอักษร S ก็ได้เพื่อให้แถบสีมาที่คำสั่งนี้ แล้วกดคีย์ ENTER ต่อมาจะแสดงหน้าต่างขึ้นมาใหม่ให้ใส่ค่าแอดเดรส (Address) ดังรูปที่ 5-3 เมื่อใส่ค่าแอดเดรสแล้ว ก็จะไปเรียกโปรแกรมย่อย Send_Command เพื่อให้ส่งคำสั่ง SDC ออกไป



รูปที่ 5.3 แสดงหน้าต่างที่ให้ใส่ค่าแอดเดรส

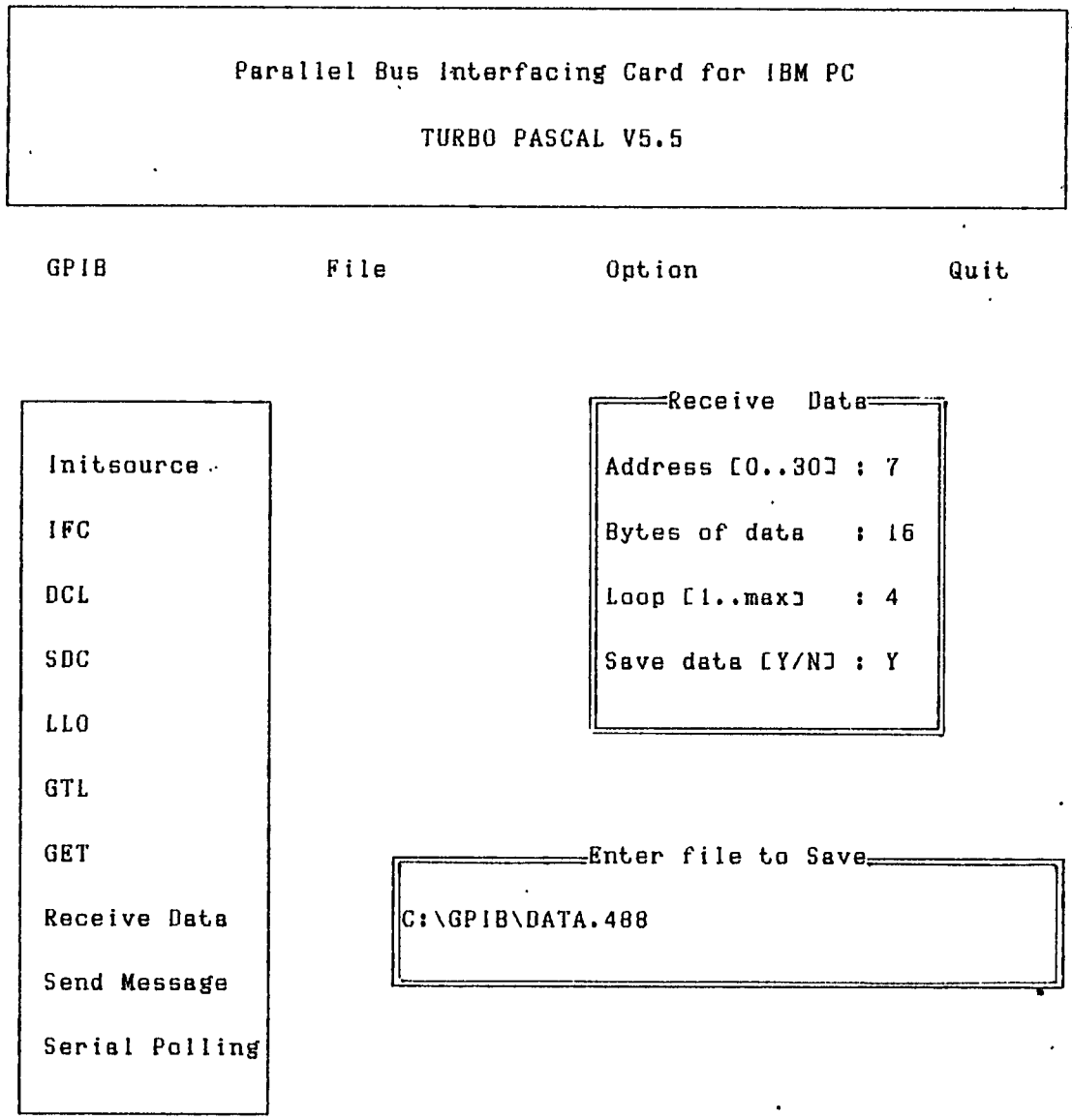
5.1.5 LLO สามารถเลือกใช้โดยการ เลื่อนลูกศรขึ้นหรือลง หรือกดอักษร L ก็ได้เพื่อให้แถบสีมาที่คำสั่งนี้ แล้วกดคีย์ ENTER ต่อมาจะแสดงหน้าต่างขึ้นมาใหม่ให้ใส่ค่าแอดเดรส (Address) เมื่อใส่ค่าแอดเดรสแล้ว ก็จะไปเรียกโปรแกรมย่อย Send_Commadn เพื่อให้ส่งคำสั่ง LLO ออกไป

5.1.6 GTL สามารถเลือกใช้โดยการ เลื่อนลูกศรขึ้นหรือลง หรือกดอักษร G ก็ได้เพื่อให้แถบสีมาที่คำสั่งนี้ แล้วกดคีย์ ENTER ต่อมาจะแสดงหน้าต่างขึ้นมาใหม่ให้ใส่ค่าแอดเดรส (Address) เมื่อใส่ค่าแอดเดรสแล้ว ก็จะไปเรียกโปรแกรมย่อย Send_Command เพื่อให้ส่งคำสั่ง GTL ออกไป

5.1.7 GET สามารถเลือกใช้โดยการ เลื่อนลูกศรขึ้นหรือลง หรือกดอักษร G ก็ได้เพื่อให้แถบสีมาที่คำสั่งนี้ แล้วกดคีย์ ENTER ต่อมาจะแสดงหน้าต่างขึ้นมาใหม่ให้ใส่ค่าแอดเดรส (Address) เมื่อใส่ค่าแอดเดรสแล้ว ก็จะไปเรียกโปรแกรมย่อย Send_Command เพื่อให้ส่งคำสั่ง GET ออกไป

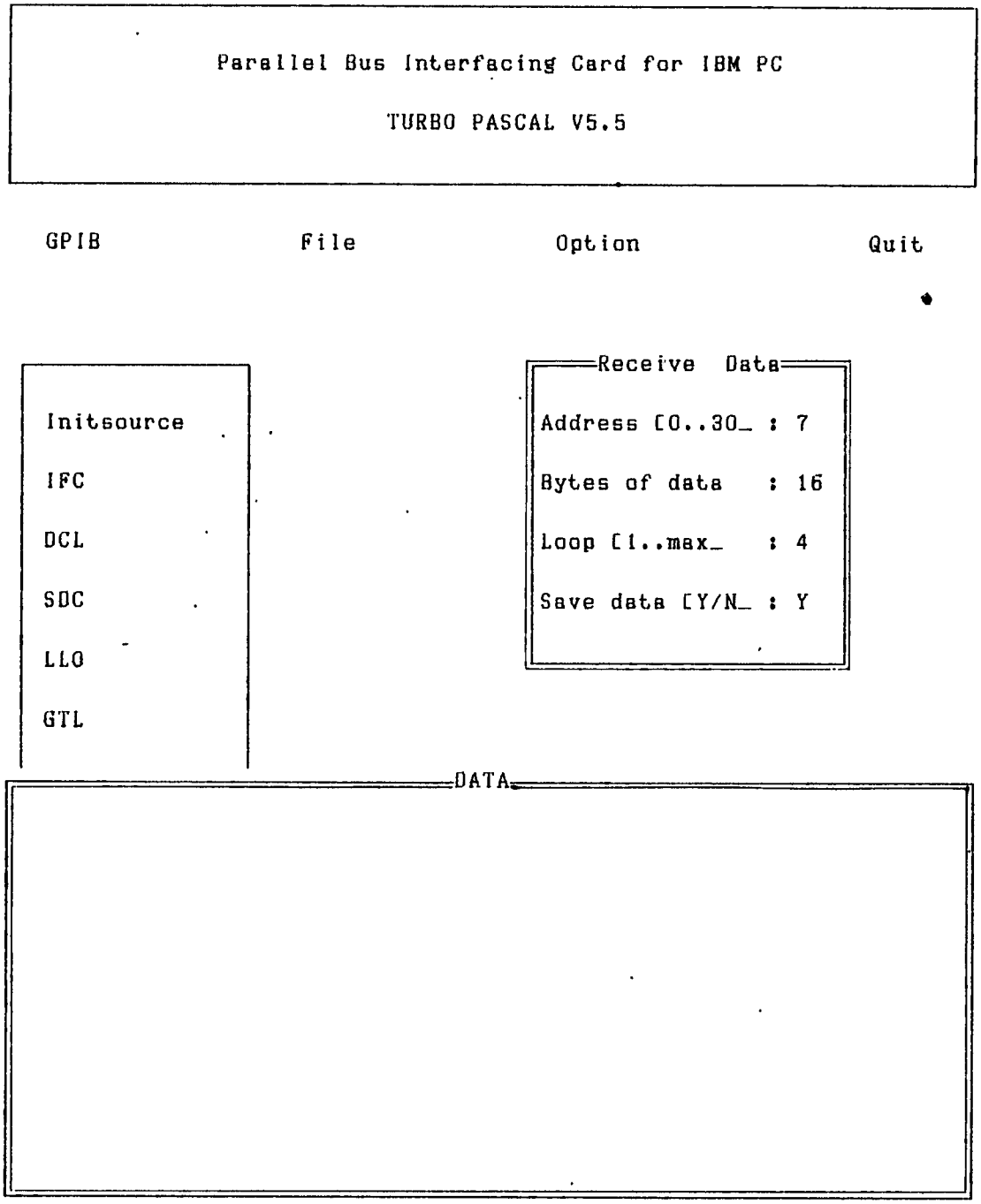
5.1.8 Receive Data สามารถเลือกใช้โดยการ เลื่อนลูกศรขึ้นหรือลง หรือกดอักษร R ก็ได้ เพื่อให้แถบสีมาที่คำสั่งนี้ แล้วกดคีย์ ENTER ต่อมาจะแสดงหน้าต่างขึ้นมาใหม่ให้ใส่ค่าแอดเดรส (Address) , จำนวนข้อมูล (Bytes of data) จำนวนรอบ (Loop) เมื่อใส่ค่าครบแล้ว ก็จะมีการถามว่าต้องการเก็บข้อมูลลงแผ่นดิสก์หรือไม่ ถ้าต้องการเก็บข้อมูลลงแผ่นดิสก์ ก็ให้กดตัวอักษร Y แล้วจะแสดงหน้าต่างขึ้นมาใหม่ให้ใส่ชื่อไฟล์ ดังรูปที่ 5-4 หรือถ้าไม่ต้องการเก็บข้อมูลลงแผ่นดิสก์ ก็ให้กดตัวอักษร N ต่อมาก็จะไปเรียกโปรแกรมย่อย Get_Data ให้ทำงานพร้อมทั้งอ่านเวลาขณะนั้นแล้ว จะแสดงหน้าต่างขึ้นมาใหม่เพื่อแสดงข้อ

มูลที่ได้รับพร้อมทั้งเวลา ดังรูปที่ 5-5



Receive data from device.

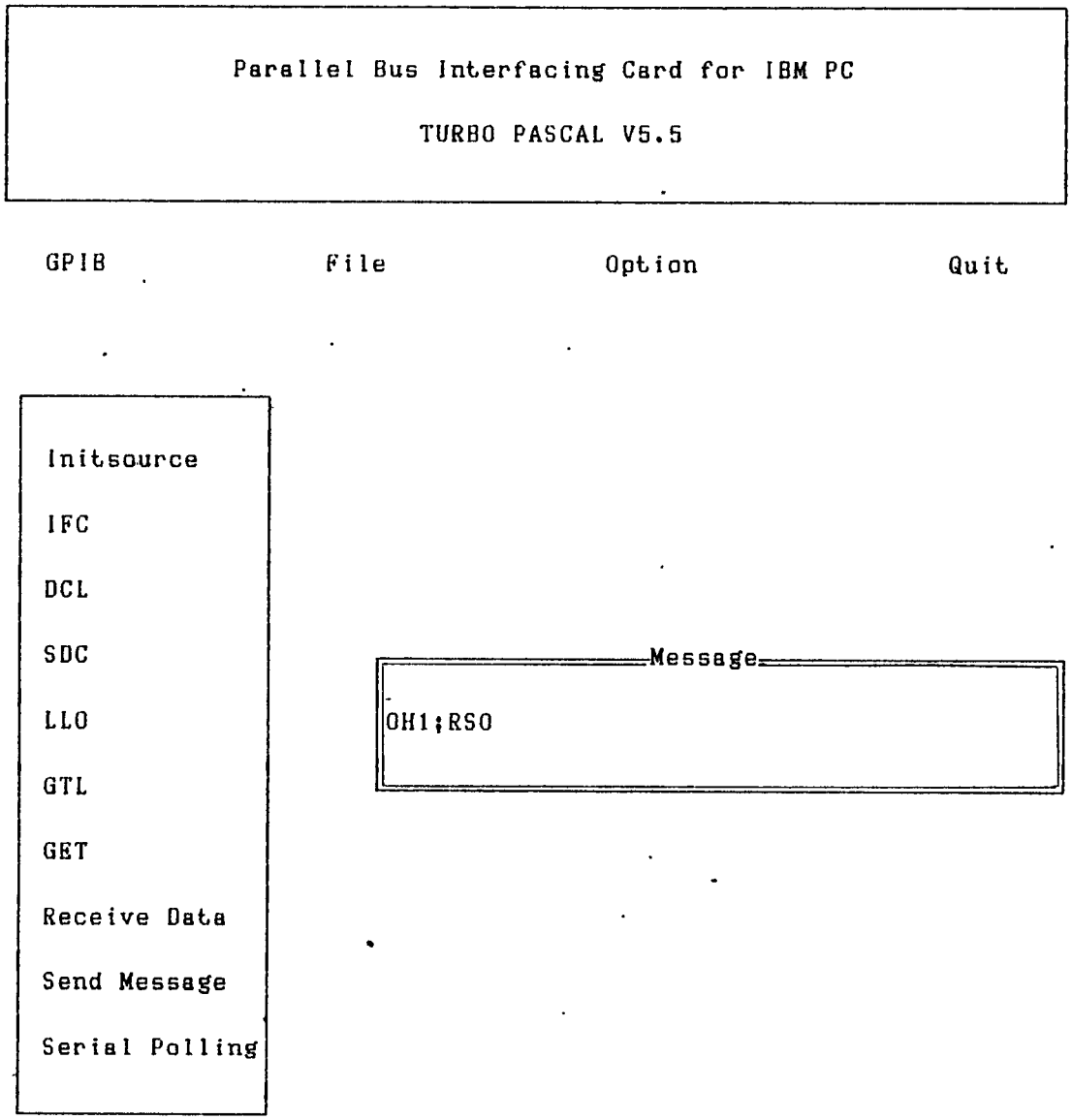
รูปที่ 5-4 แสดงหน้าต่างที่เปิดขึ้นมาเมื่อเลือกคำสั่ง Receive Data



รูปที่ 5-5 แสดงหน้าต่างที่เปิดขึ้นมาเพื่อแสดงข้อมูลที่ได้รับเข้ามา

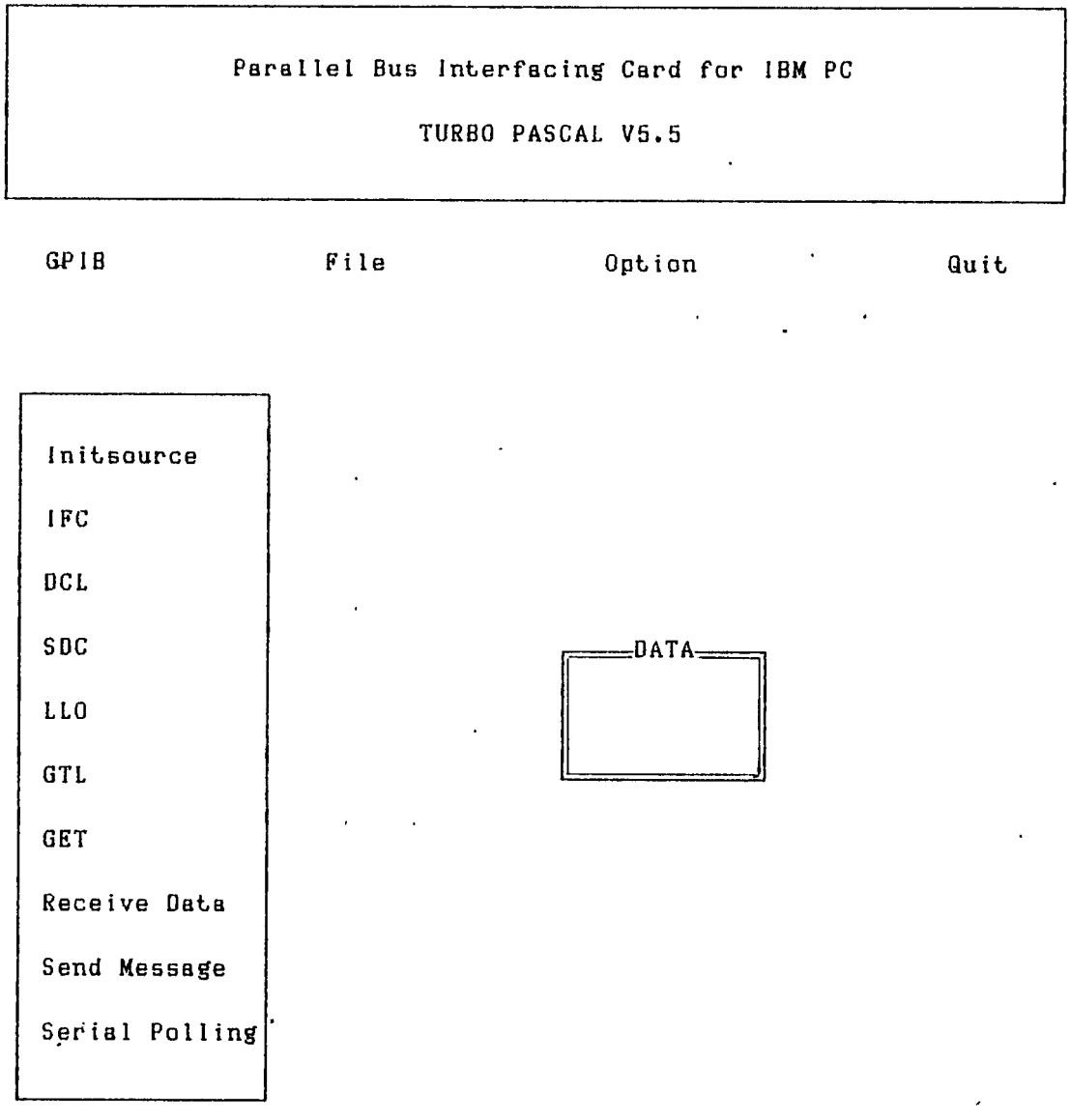
5.1.9 Send Message สามารถเลือกใช้โดยการ เลื่อนลูกศรขึ้นหรือลง หรือกดอักษร S ก็ได้ เพื่อให้แถบสีมาที่คำสั่งนี้ แล้วกดคีย์ ENTER ต่อมาจะแสดงหน้าต่างขึ้นมาใหม่ให้ใส่ค่าแอดเดรส

(Address) เมื่อใส่ค่าแอดเดรสแล้ว จะแสดงหน้าต่างขึ้นมาใหม่ให้ใส่ข้อความ (Message) ไม่เกิน 50 ตัวอักษรที่ต้องการส่งไปตั้ง (Set) การทำงานของเครื่องที่อยู่ที่แอดเดรสนั้น ดังรูปที่ 5-6 ต่อมาจะไปเรียกโปรแกรมย่อย Send_Message ให้ทำงาน



รูปที่ 5-6 แสดงหน้าต่างที่เปิดขึ้นมาเพื่อให้ใส่ข้อมูลที่ต้องการส่ง

5.1.10 Serial Polling สามารถเลือกใช้โดยการ เลื่อนลูกศรขึ้นหรือลง หรือกดอักษร S ก็ได้ เพื่อให้แถบสีมาที่คำสั่งนี้ แล้วกดคีย์ ENTER ต่อมาจะแสดงหน้าต่างขึ้นมาใหม่ให้ใส่ค่าแอดเดรส (Address) เมื่อใส่ค่าแอดเดรสแล้ว จะเรียกโปรแกรมย่อย Spolling ให้ทำงาน หลังจากนั้นจะแสดงหน้าต่างขึ้นมาใหม่เพื่อแสดงค่าข้อมูล (Data) ที่ได้รับจากเครื่องที่แอดเดรสนั้น ดังแสดงในรูปที่ 5-7



Serial polling.

รูปที่ 5-7 แสดงหน้าต่างที่เปิดขึ้นมาเพื่อแสดงข้อมูลที่ได้รับในกรณี Serial Polling

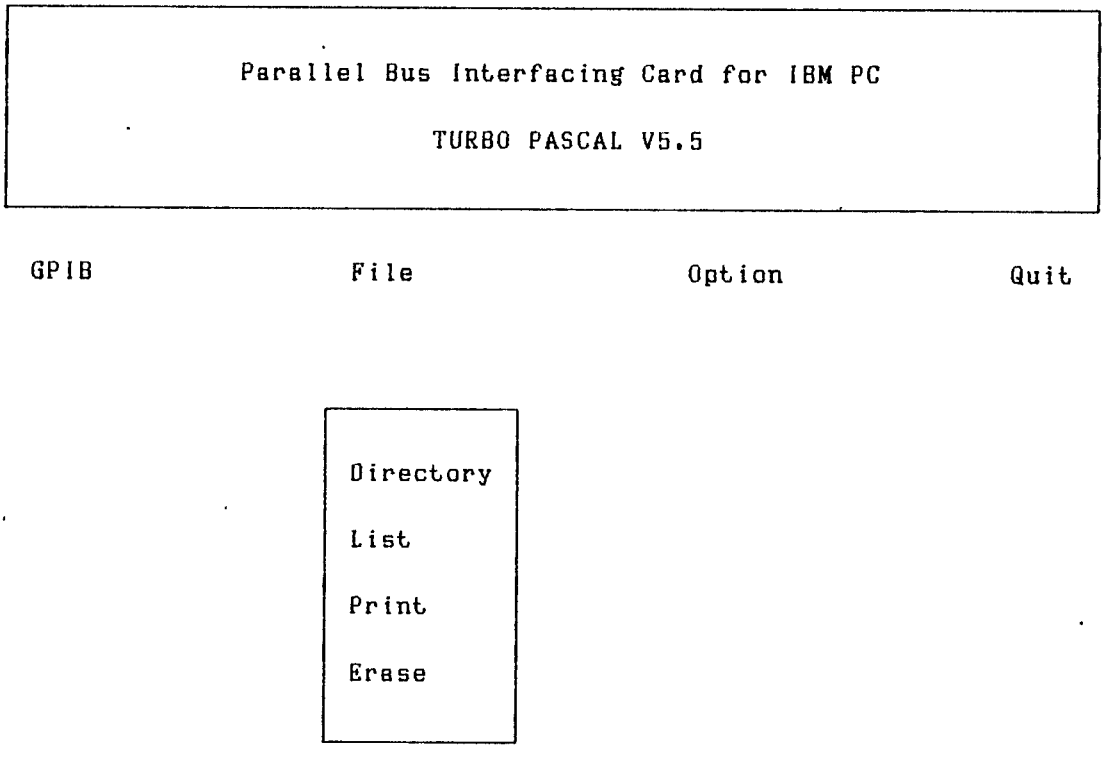
5.2 FILE MENU เมื่อเลือก FILE MENU จะมีหน้าต่าง (WINDOW) เกิดขึ้นใหม่ใต้คำว่า FILE ดังรูป 5-8 ภายใน FILE MENU จะประกอบด้วยคำสั่งต่าง ๆ ดังนี้คือ

5.2.1 Directory สามารถเลือกใช้โดยการ เลื่อนลูกศรขึ้นหรือลงเพื่อให้แถบสีมาที่คำสั่งนี้ แล้วกดคีย์ ENTER หรือกดอักษร D ก็ได้ แล้วจะแสดงหน้าต่างขึ้นมาใหม่ให้ใส่ช่องเก็บข้อมูลพร้อมทั้งทางเดิน (Drive and Path) แล้วจะใส่หรือไม่ใส่ชื่อไฟล์ก็ได้ โดยถ้าไม่ใส่อะไรเลยจะถือว่าเลือกช่องเก็บข้อมูล และทางเดินขณะนั้น และจะแสดงทุก ๆ ไฟล์ (*.*) ดังรูปที่ 5-6 และรูปที่ 5-9 เมื่อขณะที่แสดงไฟล์ผู้ใช้สามารถเลื่อนลูกศรขึ้น-ลง-ซ้าย-ขวา เพื่อเลื่อนแถบสีไปยังไฟล์ที่ต้องการหรือกดตัวอักษรตัวแรกของไฟล์นั้น เพื่อดูขนาดของไฟล์ที่เลือกได้ แล้วกดคีย์ ENTER หรือ ESC เพื่อออกจากคำสั่ง Directory

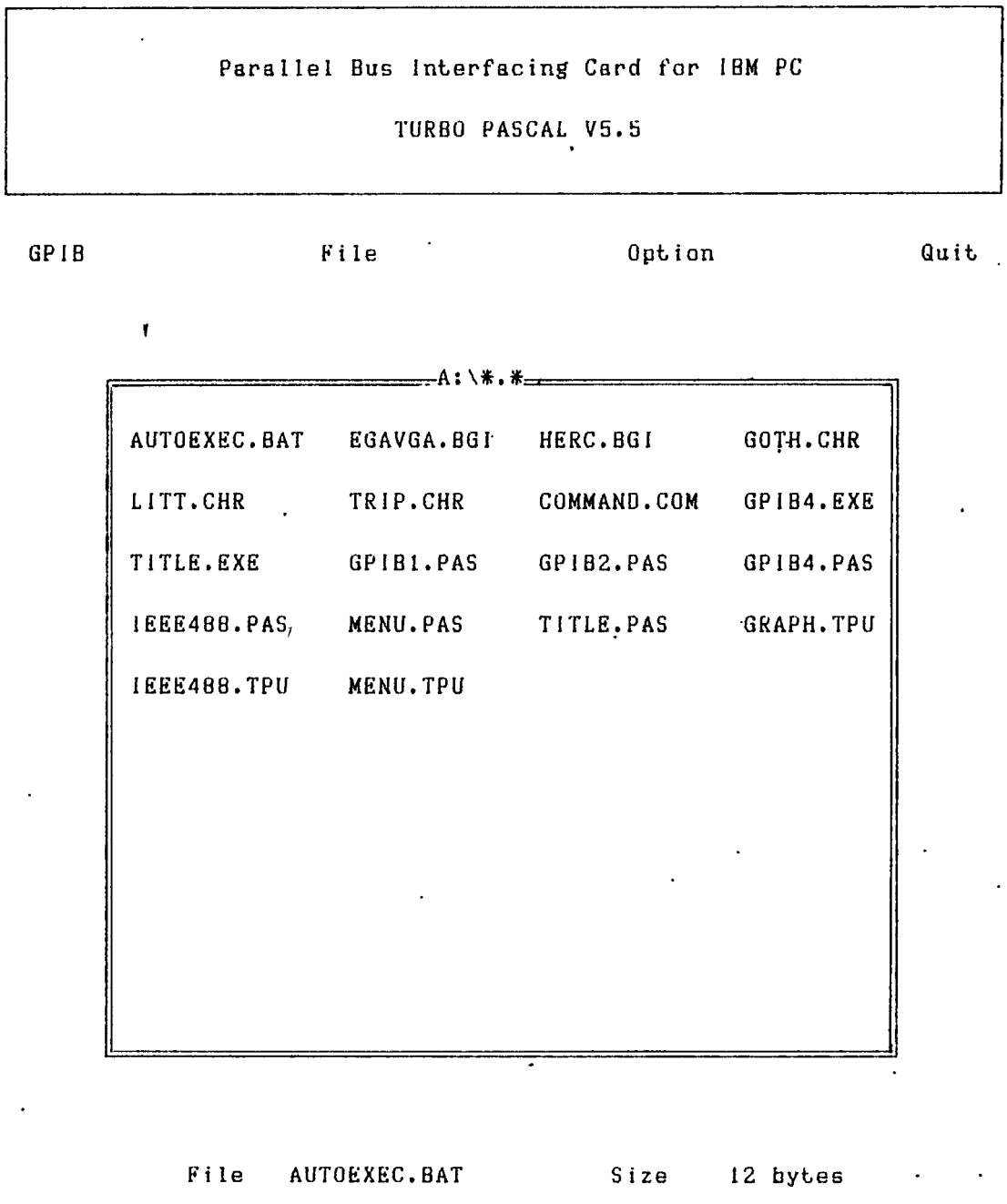
5.2.2 List สามารถเลือกใช้โดยการ เลื่อนลูกศรขึ้นหรือลงเพื่อให้แถบสีมาที่คำสั่งนี้ แล้วกดคีย์ ENTER หรือกดอักษร L ก็ได้ คำสั่งนี้จะแสดงข้อมูลที่อยู่ในไฟล์ที่เลือก โดยจะมีการถามชื่อไฟล์ก่อน ถ้าต้องการเลือกไฟล์ก็ให้กดคีย์ ENTER ก็จะแสดงไฟล์ต่าง ๆ เหมือนตอนเลือก Directory ซึ่งสามารถเลือกไฟล์ที่ต้องการแสดงโดยเลื่อนแถบสีไปยังไฟล์ที่ต้องการ หรือกดอักษรตัวแรกของไฟล์ที่ต้องการแสดงก็ได้แล้วกดคีย์ ENTER แต่ถ้าไม่ต้องการเลือกไฟล์ก็ให้ใส่ชื่อไฟล์ที่ต้องการแสดงแล้วกดคีย์ ENTER เมื่อข้อมูลที่อยู่ในไฟล์ได้แสดงที่จอ ผู้ใช้สามารถกดคีย์ต่าง ๆ ดังนี้

คีย์	เพื่อเลื่อนหน้า (Page) ขึ้น 1 บรรทัด
คีย์	เพื่อเลื่อนหน้า (Page) ลง 1 บรรทัด
คีย์ PgUp	เพื่อเลื่อนหน้า (Page) ขึ้น 1 หน้า

คีย์ PgDn	เพื่อเลื่อนหน้า (Page) ลง 1 หน้า
คีย์ Home	เพื่อเลื่อนไปยังต้นไฟล์
คีย์ End	เพื่อเลื่อนไปยังท้ายไฟล์
อักษร Q	เพื่อออกจาก List (หรือคีย์ ESC ก็ได้)



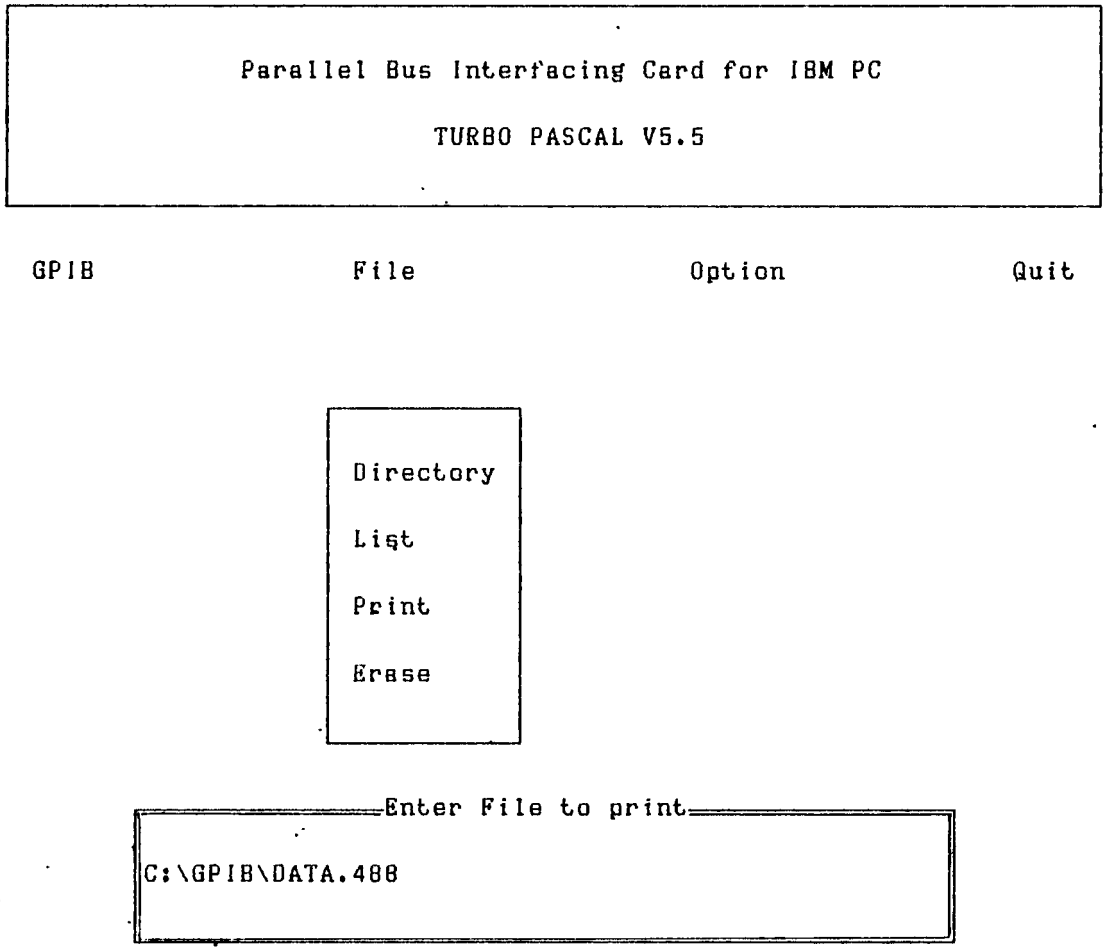
รูปที่ 5-8 แสดงภาพบนจอเมื่อเลือก FILE MENU



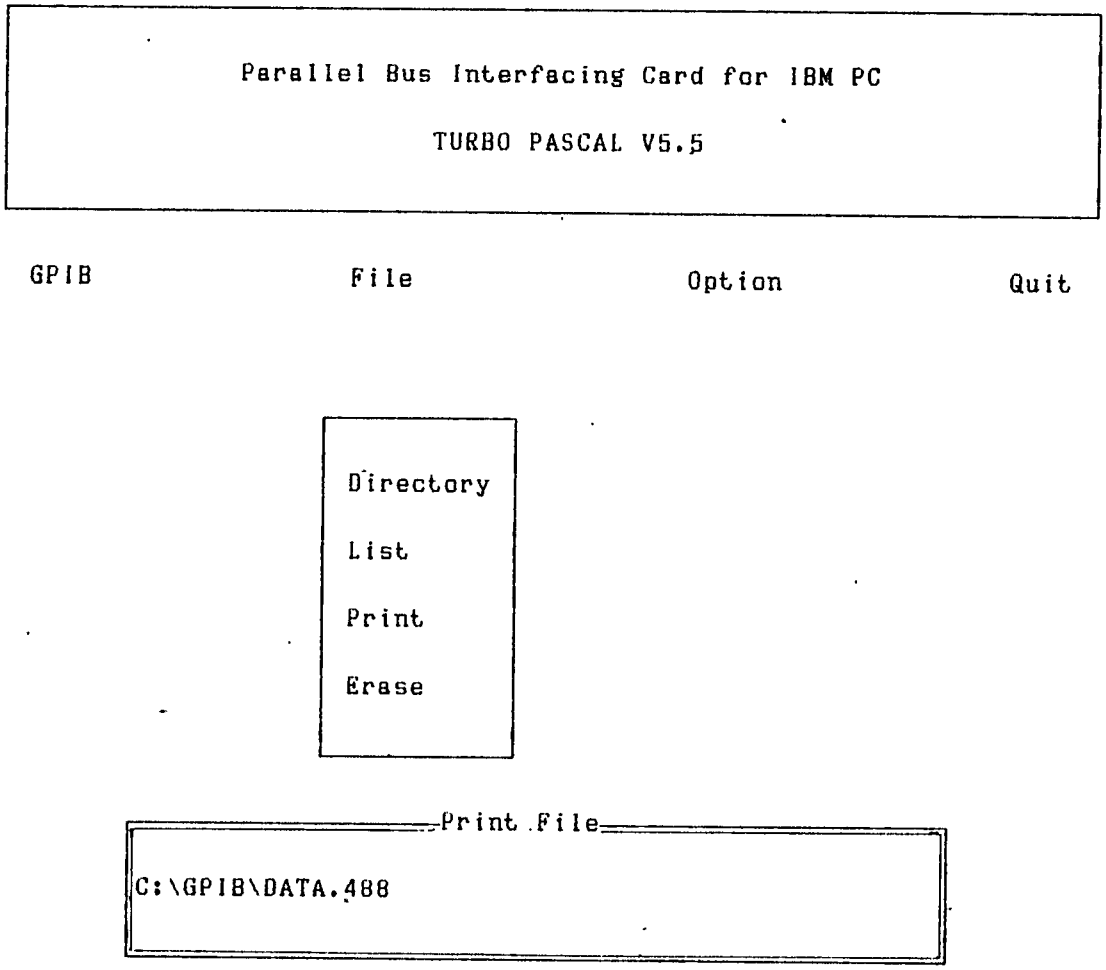
รูปที่ 5-9 แสดงหน้าต่างเมื่อเลือก Directory ให้แสดงทุก ๆ ไฟล์

5.2.3 Print สามารถเลือกใช้โดยการ เลื่อนลูกศรขึ้นหรือลงเพื่อให้แถบสีมาที่คำสั่งนี้แล้วกดคีย์ ENTER หรือกดอักษร P ก็ได้ คำสั่งนี้จะพิมพ์ข้อมูลที่อยู่ในไฟล์ที่เลือกออกทางเครื่องพิมพ์ (Printer) โดยจะแสดงหน้าต่างใหม่เพื่อถามชื่อไฟล์ก่อน ถ้าต้องการเลือกไฟล์ก็ให้กดคีย์ ENTER ก็จะแสดงไฟล์ต่างๆเหมือนตอนเลือก

Directory ซึ่งสามารถเลือกไฟล์ที่ต้องการพิมพ์โดยเลื่อนแถบสีไปยังไฟล์ที่ต้องการ หรือกดอักษรตัวแรกของไฟล์ที่ต้องการพิมพ์ แล้วกดคีย์ ENTER ดังรูปที่ 5-10 ต่อมาก็จะแสดงหน้าต่างใหม่เพื่อถามว่าเครื่องพิมพ์พร้อมหรือยัง ถ้าเครื่องพิมพ์พร้อมแล้วให้กดอักษร Y ก็จะแสดงหน้าต่างใหม่ . เพื่อบอกชื่อไฟล์ที่กำลังพิมพ์พร้อมทั้งพิมพ์ไฟล์นั้น ดังรูปที่ 5-11 แต่ถ้าเครื่องพิมพ์ยังไม่พร้อมให้กดอักษร N ก็จะเป็นการยกเลิกคำสั่ง Print

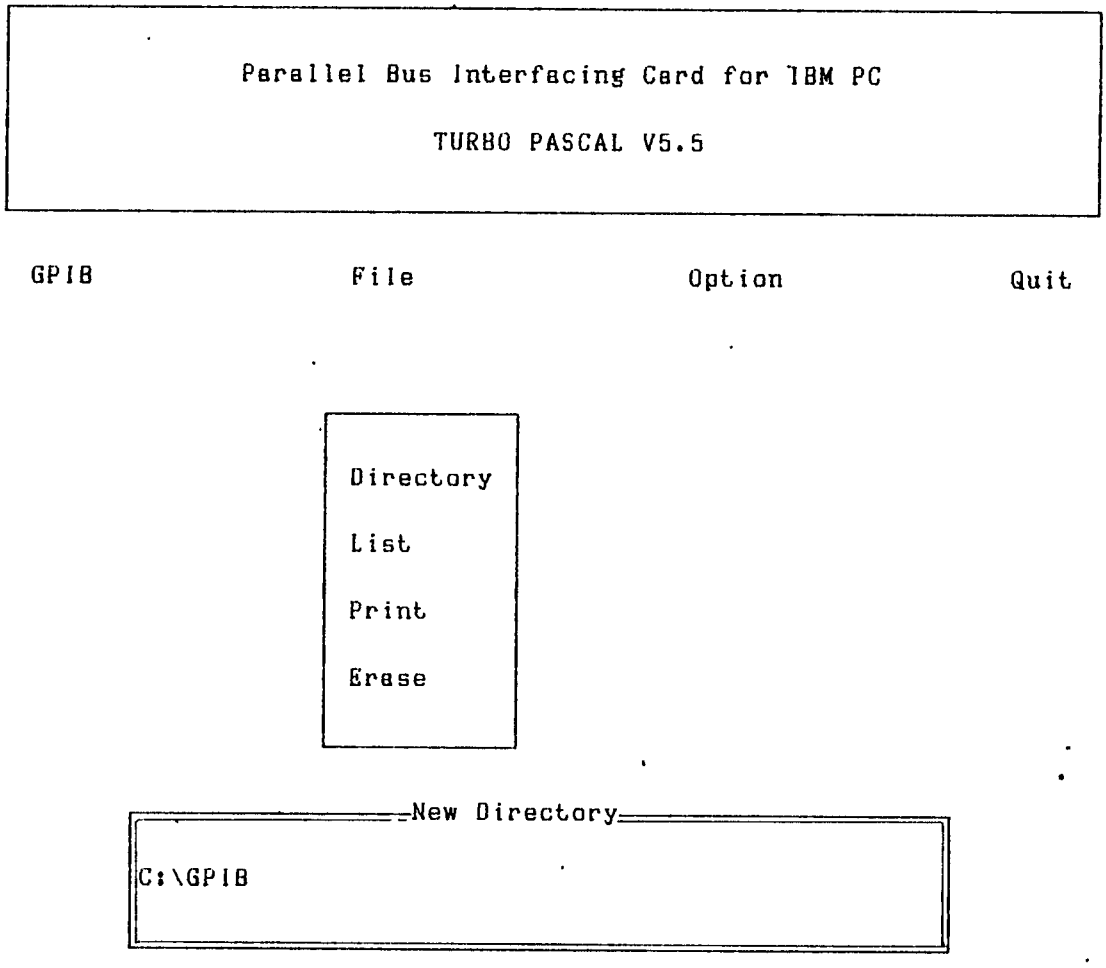


รูปที่ 5-10 แสดงหน้าต่างเมื่อเลือก Print กรณีให้ใส่ชื่อไฟล์



รูปที่ 5-11 แสดงหน้าต่างเมื่อเลือก Print กรณีแสดงชื่อไฟล์ที่กำลังพิมพ์

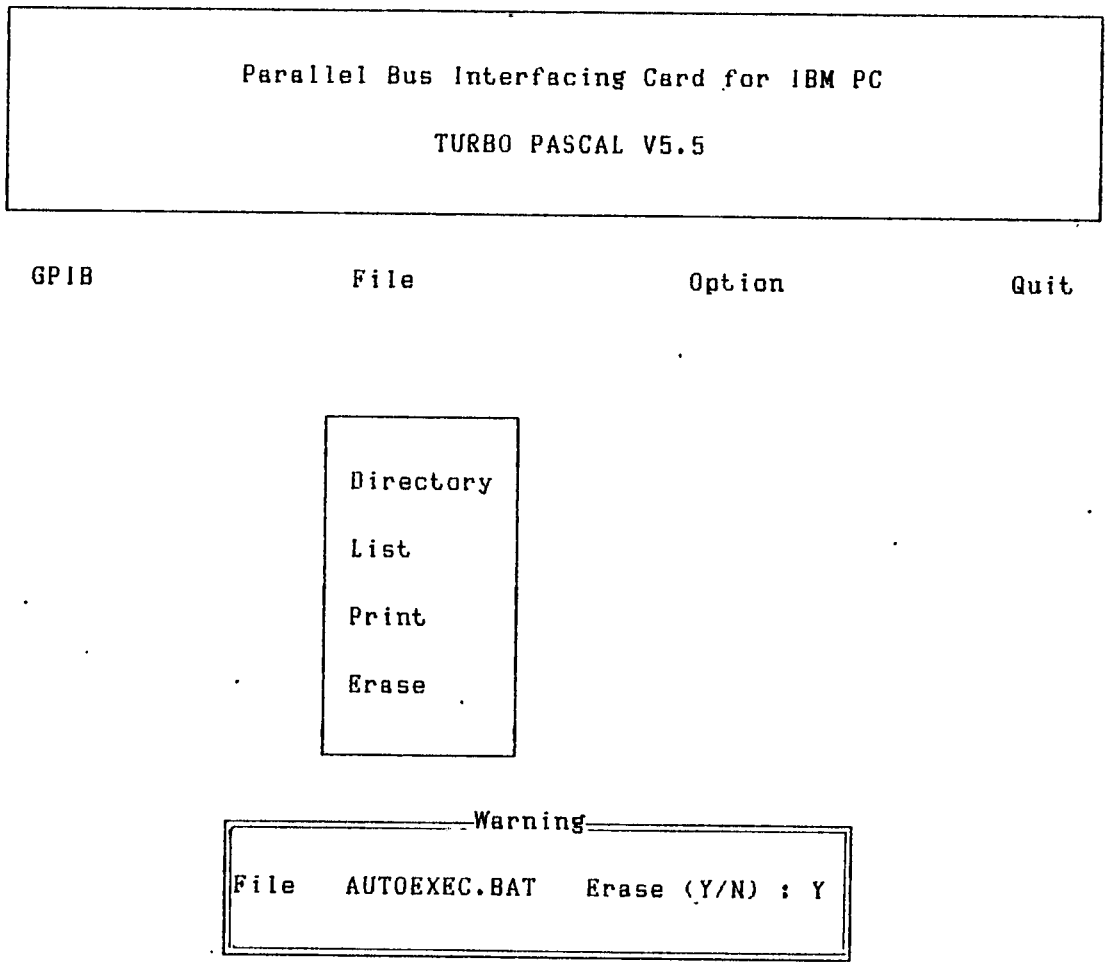
5.2.4 ChDir สามารถเลือกใช้โดยการ เลื่อนลูกศรขึ้นหรือลงเพื่อให้แถบสีมาที่คำสั่งนี้ แล้วกดคีย์ ENTER หรือกดอักษร C ก็ได้ คำสั่งนี้ใช้สำหรับเปลี่ยนช่องเก็บข้อมูล และทางเดิน (Drive and Path) โดยจะแสดงหน้าต่างขึ้นมาเพื่อให้ใส่ช่องเก็บข้อมูล และทางเดินใหม่ (Drive and Path หรือ New Directory) ดังรูปที่ 5-12



รูปที่ 5-12 แสดงหน้าต่างเมื่อเลือก ChDir กรณีให้ใส่ New Directory

5.2.5 Erase สามารถเลือกใช้โดยการ เลื่อนลูกศรขึ้นหรือลงเพื่อให้แถบสีมาที่คำสั่งนี้ แล้วกดคีย์ ENTER หรือกดอักษร E ก็ได้ คำสั่งนี้ใช้สำหรับลบไฟล์ที่ไม่ต้องการออกจากแผ่นดิสก์ โดยจะแสดงหน้าต่างเพื่อให้ใส่ชื่อไฟล์ที่ต้องการลบ ถ้าต้องการเลือกไฟล์ก็ให้กดคีย์ ENTER ก็จะแสดงไฟล์ต่าง ๆ เหมือนตอนเลือก Directory ซึ่งสามารถเลือกไฟล์ที่ต้องการลบโดยเลื่อนแถบสี

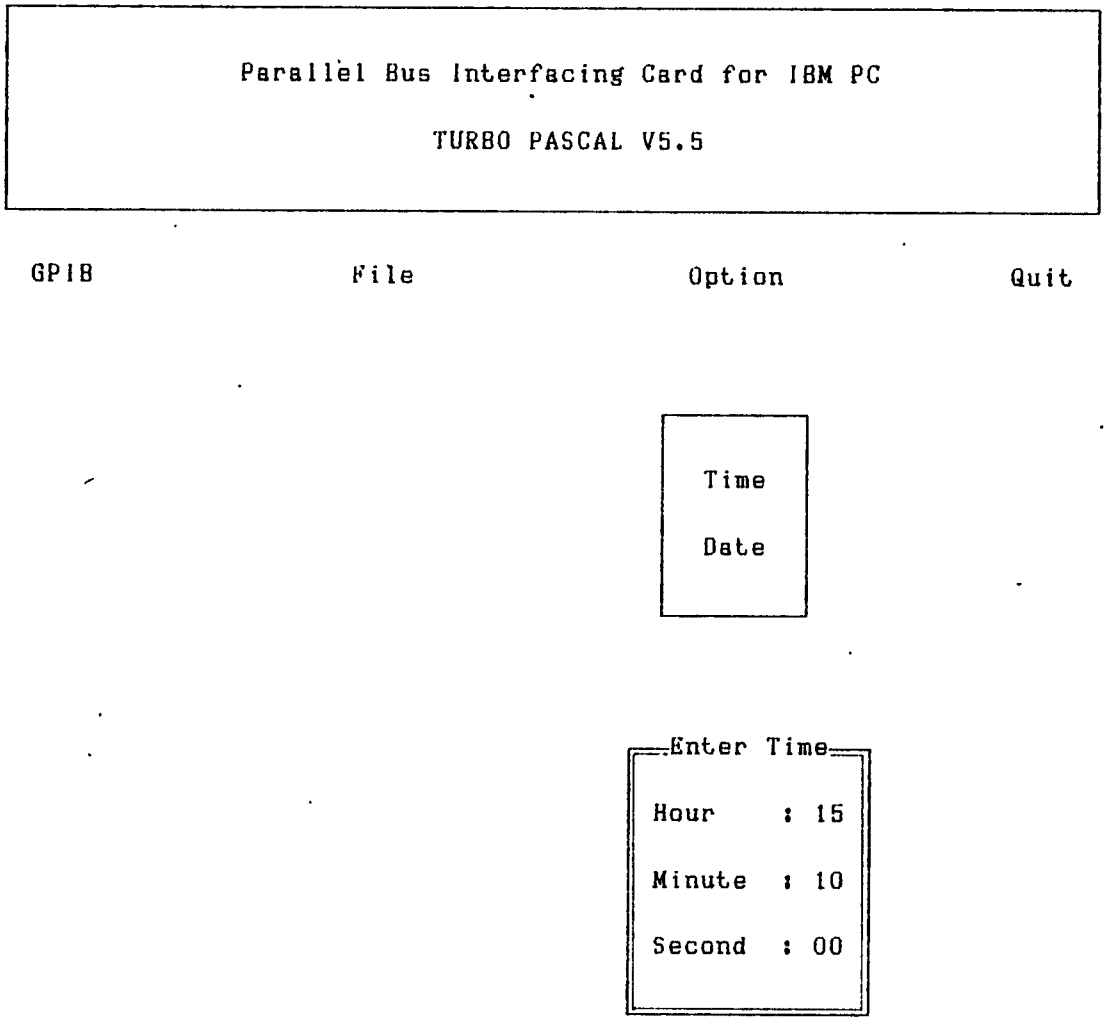
ไปยังไฟล์ที่ต้องการ หรือกดอักษรตัวแรกของไฟล์ที่ต้องการลบก็ได้แล้วกดคีย์ ENTER แต่ถ้าไม่ต้องการเลือกไฟล์ ก็ให้ใส่ชื่อไฟล์ที่ต้องการลบแล้วกดคีย์ ENTER ต่อมาก็จะแสดงหน้าต่างใหม่เพื่อถามว่า ไฟล์ที่ได้เลือกไว้ต้องการลบใช่หรือไม่ ดังแสดงในรูปที่ 5-13 ถ้าต้องการลบให้กดอักษร Y แต่ถ้าไม่ต้องการลบให้กดอักษร N ก็จะยกเลิกการลบและออกจาก Erase



รูปที่ 5-13 แสดงหน้าต่างเมื่อเลือก Erase กรณีถามว่าต้องการลบไฟล์ใช่หรือไม่

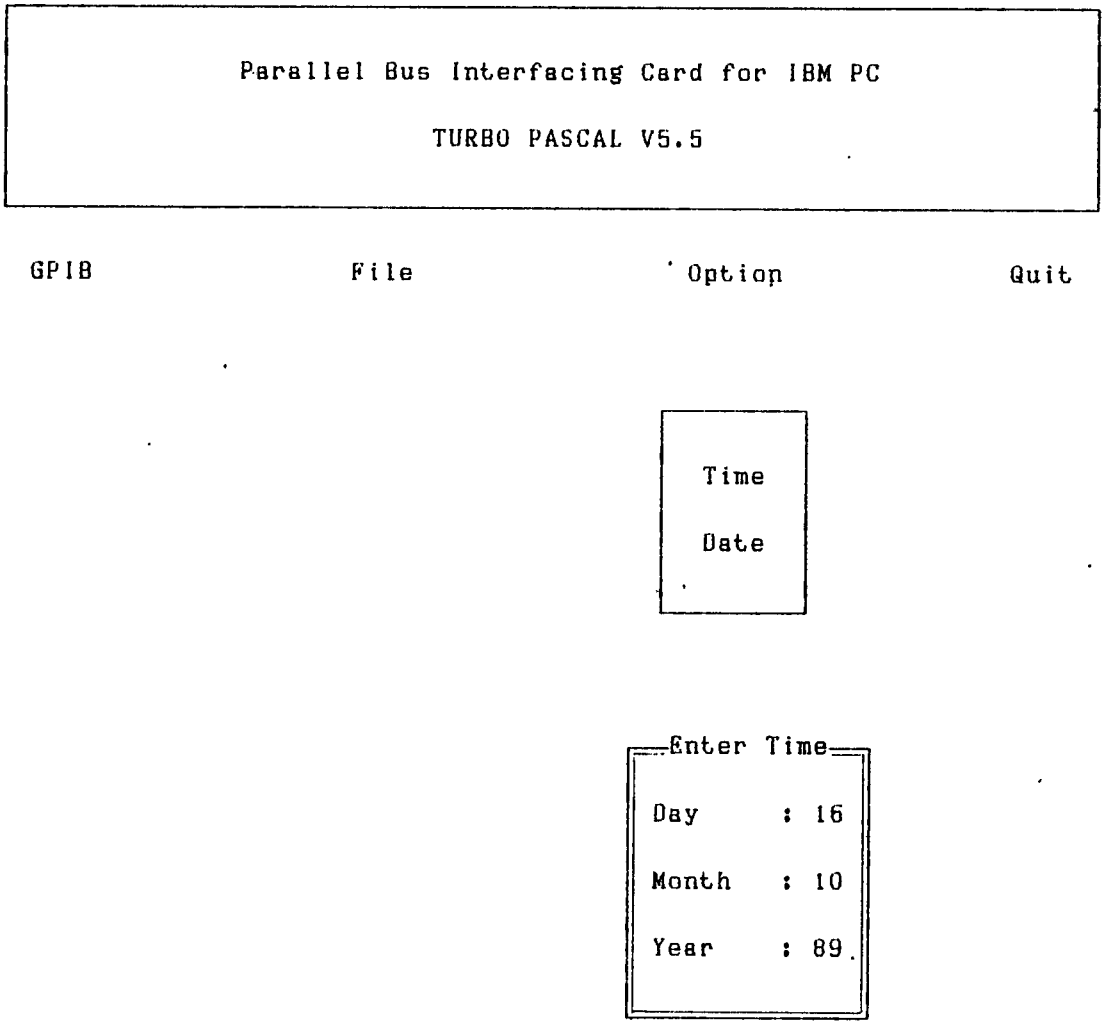
5.3 OPTION MENU เมื่อเลือก OPTION MENU จะมีหน้าต่าง (WINDOW) เกิดขึ้นใหม่ใต้คำว่า OPTION ดังรูป 5-14 ภายใน OPTION MENU จะประกอบด้วยคำสั่งต่าง ๆ ดังนี้คือ

5.3.1 Time สามารถเลือกใช้โดยการ เลื่อนลูกศรขึ้นหรือลงเพื่อให้แถบสีมาที่คำสั่งนี้ แล้วกดคีย์ ENTER หรือกดอักษร T ก็ได้ คำสั่งนี้ใช้สำหรับตั้งเวลาให้แก่ระบบจัดการ (เหมือนป้อนเวลาตอนเริ่มเดินเครื่อง) โดยจะแสดงหน้าต่างเพื่อให้ใส่เวลาเป็นชั่วโมง, นาที, วินาที ดังแสดงรูปที่ 5-14



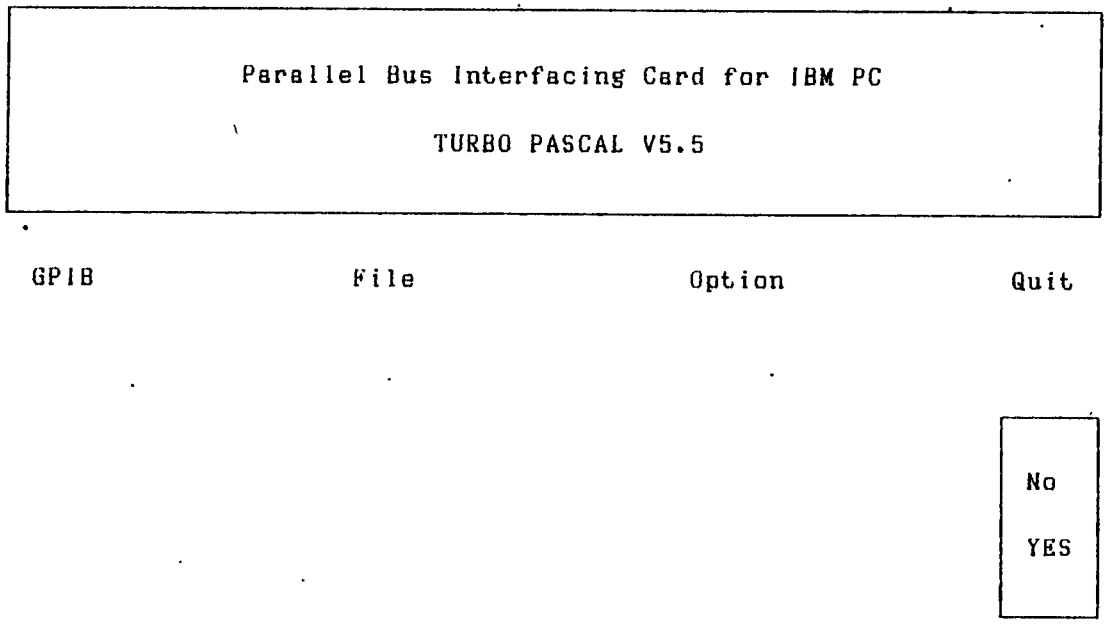
รูปที่ 5-14 แสดงหน้าต่างเมื่อเลือก Option กรณีให้ใส่เวลาที่ตั้งให้แก่ระบบ

5.3.2 Date สามารถเลือกใช้โดยการ เลื่อนลูกศรขึ้นหรือลงเพื่อให้แถบสีมาที่คำสั่งนี้ แล้วกดคีย์ ENTER หรือกดอักษร D ก็ได้ คำสั่งนี้ใช้สำหรับตั้งวันให้แก่ระบบจัดการ (เหมือนป้อนวันตอนเริ่มเดินเครื่อง) โดยจะแสดงหน้าต่างเพื่อให้ใส่วันเป็นวันที่, เดือน, ปี ดังแสดงรูปที่ 5-15



รูปที่ 5-15 แสดงหน้าต่างเมื่อเลือก Option กรณีให้ใส่วันที่ตั้งให้แก่ระบบ

5.4 QUIT MENU เมื่อเลือก QUIT MENU จะมีหน้าต่าง (WINDOW) เกิดขึ้นใหม่ใต้คำว่า QUIT ดังรูป 5-16 ถ้าไม่ต้องการออกจากโปรแกรมก็



รูปที่ 5-16 แสดงหน้าต่างเมื่อเลือก Quit

บทที่ 6

การทดลองและผลการทดลอง

ในการทดลองนี้ เทอมแรกคณะผู้จัดทำได้ทำการเขียนในส่วนของ Software ไปก่อนบางส่วน และส่วนของ Hardware นั้นได้ทำการออกแบบ Circuit ไว้แล้ว แต่ไอซีบางเบอร์หาในเมืองไทยไม่ได้คือ เบอร์ DS75161N จนก่อนสอบ Final เพียง 5 วันจึงได้มา เพราะฉะนั้นคณะผู้จัดทำจึงไม่กล้าออกแบบ Print อย่างถาวรได้ เนื่องจากอาจจะมีการเปลี่ยนแปลงไอซีเบอร์ดังกล่าวเป็นเบอร์อื่นๆ เพราะหาไม่ได้ในเมืองไทย หลังจากได้ไอซีมาแล้ว จำเป็นต้องใช้วิธีบัดกรีบนแผ่น Print เอนกประสงค์ซึ่งได้จาก ม.บ้านหม้อ ฉะนั้นจึงมีลายไฟค่อนข้างมากซึ่งทำให้เกิด Noise รบกวนใน Hardware ถึงแม้ว่าจะทำการแก้ไข Software ช่วยก็ตาม อาการรบกวนของ Noise ต้องเกิดขึ้นในโหมดของ Talker อยู่ดี

ในการเกิด Noise ต่าง ๆ จะเกิดจากสายต่อของ GPIB ไม่เป็นสายมาตรฐานของ HP (Hewlett Packard) ซึ่งจะเกิดขึ้นในกรณีสายต่อ GPIB นั้นยาวมาก ๆ แต่เมื่อใดถ้าเป็นสายต่อ GPIB แบบมาตรฐาน การเกิด Noise จะน้อยลงมาก ๆ หรือแทบจะไม่มี Noise เกิดขึ้นเลย แต่สายสัญญานมาตรฐานของ HP นี้จะมีราคาแพงมาก ส่วนอีกอย่างหนึ่งที่เกิด Noise ก็คือ wire wap ที่ลงบนแผ่น Print ซึ่งได้ทำการแก้ไขโดยการทำแผ่นวงจรลาย Print เรียบร้อย ส่วนในโหมดอื่นๆ ยังทำงานได้ตามปกติ

ในการทดลองนั้นควรจะมีเครื่องมือวัดหลายๆประเภท มาช่วยประกอบในการทดลองจึงจะสามารถสรุปได้ว่า ในระบบของ GPIB สามารถติดต่อกับอุปกรณ์เครื่องมือวัดต่างๆ ได้ 1 ถึง 14 ตัวจริงๆ

บทที่ 7

บทวิจารณ์และสรุป

จากผลการทดลองที่ได้นี้ยังไม่เป็นที่พอใจสำหรับคณะผู้จัดทำ สาเหตุเนื่องจากการได้ไอซีเบอร์ DS75161N มาช้าจึงทำให้การทดลองและการพัฒนาในช่วงสั้นๆ นั้น ไม่สามารถทำได้ดียิ่งขึ้น สิ่งที่ต้องกระทำเพื่อดำเนินงานต่อไปหรือสำหรับผู้ที่สนใจคือ ทำการพัฒนาทางด้าน Software ให้เป็นลักษณะของข้อมูลที่มีความละเอียดหรือความถูกต้องและผลที่ได้ให้มาเป็นในรูปของ Real Time Graphics

กิตติกรรมประกาศ

ในการทำโปรเจคเรื่อง การเชื่อมต่อข้อมูลทั่วไปแบบขนาน ในครั้งนี้ซึ่งได้บรรลุเป็นผลสำเร็จเป็นอย่างดี ก็เพราะได้รับการสนับสนุนเป็นอย่างดีจากผู้ที่มีรายชื่อดังต่อไปนี้

1. อาจารย์ วิทยา ทิพย์สุวรรณพร ผู้ซึ่งเป็นอาจารย์ที่ปรึกษาและผู้สนับสนุนทั้งทางด้านตำราและHardware ที่ใช้ประกอบการทดลอง

2. คุณ วิชัย นอกเมือง ตำแหน่ง " Technical Support Manager " ของ National Semiconductor Bangkok LTD. ซึ่งเป็นผู้อนุญาติให้คณะผู้จัดทำนำเครื่องมือ (Digital Multimeter) ใช้ประกอบการทดลองโปรเจค

3. คุณ วิบูลย์ พิชัยยา ตำแหน่ง " Section Head Engineer " ของ National Semiconductor Bangkok LTD. เป็นผู้อำนวยความสะดวกและดำเนินการในการขอยืมเครื่องมือ (Digital Multimeter)

ขอขอบพระคุณเป็นอย่างยิ่ง

หนังสืออ้างอิง

1. Borland International, " Reference Guide and User's Guide ", TURBO PASCAL, (1988)

2. Peter R. Rony, Microcomputer Interfacing with 8255 PPI chip, (1980)

3. Eugone Fisher and c.w. Jensen, PET/CMB and IEEE488 BUS(GPIB) 2 nd Edition

4. บุญเลิศ เอี่ยมทัศนา, " คู่มือเทอร์โบปาสคาล รุ่น 4.0-5.0 ", ISBN, 974-509-009-3, 2521

5. บุญเลิศ เอี่ยมทัศนา, " เรียนรู้ภาษาปาสคาลด้วยเทอร์โบปาสคาล 4.0-5.0 ", ISBN, 974-509-020-4

6. กนก เจนจิรยงค์เวช, " บัสอินเตอร์เฟส ", วารสารเซมิคอนดักเตอร์อิเลคทรอนิคส์, ฉบับที่ 78-79, หน้าที่ 214-218 และ 196-201, 2530

ภาคผนวก

```
Procedure UserProgram;
begin
   { Init source;              SET INTERFACE CARD }
    DCL;                     { CLEAR DEVICE }
    IFC;                     { CLEAR INTERFACE }
    { 7 = ADDRESS OF 2501A PRECISION DIGITAL MULTIMETER }
    Send_Message (1, F3,R5');   {  SET OHM AND AUTO RANGE }
    Get_data (1,D,16);           {  RECEIVE DATA 16 BYTE AND TIME }
    Save_data (D);              {  SAVE DATA AND TIME }
end;
```

```
program SYSTEM_PROGRAM;

{ IEEE 488 Interfaceing Card for IBM PC version 1.0 }
{         Adviser.....Vithaya                       }
{         By.....Boonyoung Iamkamol                 }


Uses       Crt,DOS;

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

type       DataFormat = string[50];

var        BusError     : Boolean;
           i,Addr,num   : integer;
           MLA          : byte;       { My Listen Address       }
           MTA          : byte;       { My Talk Address         }
           M            : DataFormat; { Remote message          }
           D            : DataFormat; { Data                    }
           DByte        : byte;
           Status       : byte;
           nr,mr,sr,fr  : word;
           ch           : char;

         { portA    = $310
           portB    = $311
           portC    = $312
           control1 = $313
           portC2   = $316
           control2 = $317 }


Procedure Address (Addr :integer ; var MLA,MTA : byte);
  begin
    case Addr of
      0 : begin MLA := $20 ; MTA := $40 ; end;
      1 : begin MLA := $21 ; MTA := $41 ; end;
      2 : begin MLA := $22 ; MTA := $42 ; end;
      3 : begin MLA := $23 ; MTA := $43 ; end;
      4 : begin MLA := $24 ; MTA := $44 ; end;
      5 : begin MLA := $25 ; MTA := $45 ; end;
      6 : begin MLA := $26 ; MTA := $46 ; end;
```

```
      7 : begin MLA := $27 ; MTA := $47 ; end;
      8 : begin MLA := $28 ; MTA := $48 ; end;
      9 : begin MLA := $29 ; MTA := $49 ; end;
     10 : begin MLA := $2A ; MTA := $4A ; end;
     11 : begin MLA := $2B ; MTA := $4B ; end;
     12 : begin MLA := $2C ; MTA := $4C ; end;
     13 : begin MLA := $2D ; MTA := $4D ; end;
     14 : begin MLA := $2E ; MTA := $4E ; end;
     15 : begin MLA := $2F ; MTA := $4F ; end;
     16 : begin MLA := $30 ; MTA := $50 ; end;
     17 : begin MLA := $31 ; MTA := $51 ; end;
     18 : begin MLA := $32 ; MTA := $52 ; end;
     19 : begin MLA := $33 ; MTA := $53 ; end;
     20 : begin MLA := $34 ; MTA := $54 ; end;
     21 : begin MLA := $35 ; MTA := $55 ; end;
     22 : begin MLA := $36 ; MTA := $56 ; end;
     23 : begin MLA := $37 ; MTA := $57 ; end;
     24 : begin MLA := $38 ; MTA := $58 ; end;
     25 : begin MLA := $39 ; MTA := $59 ; end;
     26 : begin MLA := $3A ; MTA := $5A ; end;
     27 : begin MLA := $3B ; MTA := $5B ; end;
     28 : begin MLA := $3C ; MTA := $5C ; end;
     29 : begin MLA := $3D ; MTA := $5D ; end;
     30 : begin MLA := $3E ; MTA := $5E ; end;
   end;
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
        BusError := true;
        writeln ('BUS ERROR !');
        delay(2000);
        halt;
      end;
   BusError := false;
   Port[$310] := $FF - Data;          { put Data on DIO lines       }
     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 NDAC=H }
   Port[$312] := (Port[$312] or $0C);   { make DAV=H EOI=H           }
end;

procedure Init_Source;
begin
    writeln( Init Source );
    Port[$317] := $88;                { CU2i CL20           }
    Port[$313] := $88;                { Alo  Blo  Culi  Clio }
    Port[$311] := $0B;                { 160=T;DAV=T,NRFD=NDAC=R }
    Port[$310] := $FF;                { clear data }
    writeln;
```

```
end;


procedure Send_Command (Addr : integer ; Command : byte);
begin
    writeln('Send Command');
    Port[$316] := $02;                  { make REN=L IFC=H ATN=L }
    Address (Addr,MLA,MTA);
    Source_Operation (MLA);
    Source_Operation (Command);
    writeln;
end;

procedure Send_Message (Addr : integer ; M : DataFormat);
var     message : char;
begin
    Port[$316] := $02;                  { make PEN=L  IFC=H  ATN=L}
    Address (Addr,MLA,MTA);
    Source_Operation (MLA);
    Port[$316] := $03;                 { make PEN=L  IFC=H  ATN=H}
    writeln('Send message  = ',M);
    for i :=1 to length (M) do          { send messages }
      begin
       message := M[i];
       Source_Operation (integer (message));
      end;
    Source_OPeration ($0D);            { send CR delimiter }
    Source_Operation ($0A);            { send LF delimiter }
    writeln;
end;

procedure Acceptor_Operation (var DByte : 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 }
    DByte      := $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 : integer ; var D : DataFormat ; Num : integer);
begin
    Port[$316] := $02;                 { make REN=L IFC=H ATN=L }
    Address (Addr,MLA,MTA);
    Source_Operation (MTA);
    Port[$313] := $91;                 { ALi Bio CUo CLio          }
    Port[$311] := $00;                 { 160=R;DAV=R,NRFD=NDAC=T   }
    Port[$312] := $C0;                 { make RDAC=L NRFD=L        }
    Port[$316] := $03;                 { make REN=L  IFC=H  ATN=H  }
    D := '';                           { clear Data  }
```

```
    GetTime(hr,mr,sr,fr);
    for i := 1 to num do
       begin
          Acceptor_Operation (DByte);
          D := D+Char(DByte);
       end;
    writeln('Receive  Data = ',D);
    writeln;
    writeln('Real Time Measure = ',hr,':',mr:2,':',fr:2);
    Port[$313] := $88;              { Ato  Bto  CUti  CLto     }
    Port[$311] := $0B;              { 160=T;DAV=T,NRFD=NDAC=R  }
    Port[$316] := $02;              { make REN=L  IFC=H  ATN=L }
    writeln;
end;

procedure IFC;
begin
    writeln('Interface Clear');
    Port[$316] := $00;              { make REN=L  IFC=L  ATN=L }
    writeln;
end;

procedure DCL;
begin
    writeln('Device Clear');
    Port[$316] := $02;              { make REN=L  IFC=H  ATN=L }
    Source_Operation ($14);
    writeln;
end;

procedure SPolling (Addr : integer ; var Dbyte : byte);
begin
    writeln('Serial Polling');
    Port[$316] := $02;              { make REN=L  IFC=H  ATN=L }
    Address (Addr,MLA,MTA);
    Source_Operation (UNL);
    Source_Operation (MTA);
    Source_Operation (SPE);
    Port[$313] := $91;              { Ati  Bto  CUto  CLti     }
    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 (DByte);
    ch := char(DByte);
    if (DByte = $0D) or (DByte = $0A) or (DByte = $08) then ch :=' ';
    writeln(' Receive STATUS ; ',DByte:3,'(',ch:1,')');
    Port[$313] := $88;              { Ato  Bto  CUti  CLto     }
    Port[$311] := $0B;              { 160=T;DAV=T,NRFD=NDAC=R  }
    Port[$316] := $02;              { make REN=L  IFC=H  ATN=L }
    Port[$312] := $0C;              { make DVA=H  EOI=H        }
    Source_Operation (SPD);
    source_Operation (UNT);
    writeln;
end;
```

```
Function Exist (FileName : string) : boolean;
var f : file;
begin
     assign(f,FileName);
     {$I-} reset(f); close(f); {$I+}
     Exist := (IOresult = 0);
end;

Procedure Save_Data (D : DataFormat);
var      FileSave : Text;
        FileData : string[30];
begin
    writeln('Save Data');
    FileData := '';
     repeat
        begin
          write('  Enter FileName to save :     ');
          readln(FileData);
    end;
      until FileData <> '';
      Assign(FileSave,FileData);
      if Exist(FileData) then append(FileSave)
       else
         begin
           writeln (   new file.............');
           rewrite(FileSave);
        end;
    writeln(FileSave,0,' ','Time =',hr:2,':',mr:2,':',sr:2,':',fr:2);
    close(FileSave);
    writeln('###Save data finish####');
    writeln;
    delay(1000);
end;

{$I User.inc}

begin
    ClrScr;
    writeln('  IEEE488 Interface Card for IBM ');
    writeln;
    UserProgram;
    delay(3000);
end. { IEEE488 Unit }
```

```
Unit IEEE488;

{ IEEE 488 Interfaceing Card for IBM PC version 1.0 }
{         Adviser......Vithaya                      }
{         By...........Boonyoung Iamkamol           }

Interface

Uses       Crt,Menu;

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

type       DataFormat = string[50];

var        BusError     : Boolean;
           i            : integer;
           Addr         : integer;
           MLA          : byte;        { My Listen Address      }
           MTA          : byte;        { My Talk Address        }
           M            : DataFormat;  { Remote message         }
           D            : DataFormat;  { Data                   }
           num          : integer;
           DByte        : byte;
           option       : byte;

         { portA     = $310
           portB     = $311
           portC1    = $312
           control1  = $313
           portC2    = $316
           control2  = $317 }

procedure Address (Addr :integer ; var MLA,MTA : byte);
procedure Source_Operation (Data : byte);
procedure Init_Source;
procedure Send_Command (Addr : integer ; Command : byte);
procedure Send_Message (Addr : integer ; M : DataFormat);
procedure Acceptor_Operation (var DByte : byte);
procedure Get_Data (Addr : integer ; var D : DataFormat ; Num : integer);
procedure IFC;
procedure DCL;
procedure SPolling (Addr : integer ; var Dbyte : byte);

Implementation
```

```
procedure Address (Addr :integer ; var MLA,MTA : byte);
 begin
  case Addr of
    0 : begin MLA := $20 ; MTA := $40 ; end;
    1 : begin MLA := $21 ; MTA := $41 ; end;
    2 : begin MLA := $22 ; MTA := $42 ; end;
    3 : begin MLA := $23 ; MTA := $43 ; end;
    4 : begin MLA := $24 ; MTA := $44 ; end;
    5 : begin MLA := $25 ; MTA := $45 ; end;
    6 : begin MLA := $26 ; MTA := $46 ; end;
    7 : begin MLA := $27 ; MTA := $47 ; end;
    8 : begin MLA := $28 ; MTA := $48 ; end;
    9 : begin MLA := $29 ; MTA := $49 ; end;
   10 : begin MLA := $2A ; MTA := $4A ; end;
   11 : begin MLA := $2B ; MTA := $4B ; end;
   12 : begin MLA := $2C ; MTA := $4C ; end;
   13 : begin MLA := $2D ; MTA := $4D ; end;
   14 : begin MLA := $2E ; MTA := $4E ; end;
   15 : begin MLA := $2F ; MTA := $4F ; end;
   16 : begin MLA := $30 ; MTA := $50 ; end;
   17 : begin MLA := $31 ; MTA := $51 ; end;
   18 : begin MLA := $32 ; MTA := $52 ; end;
   19 : begin MLA := $33 ; MTA := $53 ; end;
   20 : begin MLA := $34 ; MTA := $54 ; end;
   21 : begin MLA := $35 ; MTA := $55 ; end;
   22 : begin MLA := $36 ; MTA := $56 ; end;
   23 : begin MLA := $37 ; MTA := $57 ; end;
   24 : begin MLA := $38 ; MTA := $58 ; end;
   25 : begin MLA := $39 ; MTA := $59 ; end;
   26 : begin MLA := $3A ; MTA := $5A ; end;
   27 : begin MLA := $3B ; MTA := $5B ; end;
   28 : begin MLA := $3C ; MTA := $5C ; end;
   29 : begin MLA := $3D ; MTA := $5D ; end;
   30 : begin MLA := $3E ; MTA := $5E ; end;
  end;
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
      BusError := true;
      tone(300,300);
      wdrt (68,25, BUS ERROR !');
      delay(1000);
      exit;
    end;
  BusError := false;
  Port[$310] := $FF - Data;           { put Data on DIO lines       }
    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 NDAC=H }
  Port[$312] := (Port[$312] or $0C);   { make DAV=H EOI=H          }
end;

procedure Init_Source;
begin
    Port[$317] := $88;                { CU2i CL20           }
    Port[$313] := $88;                { Alo  Blo  Culi  Cllo }
    Port[$311] := $0B;                { 160=T;DAV=T,NRFD=NDAC=R }
    Port[$310] := $FF;                { clear data }
end;


procedure Send_Command (Addr : integer ; Command : byte);
begin
    Port[$316] := $02;                 { make REN=L IFC=H ATN=L }
    Address (Addr,MLA,MTA);
    Source_Operation (MLA);
    if BusError then exit;
    Source_Operation (Command);
end;

procedure Send_Message (Addr : integer ; M : DataFormat);
var    message : char;
begin
    Port[$316] := $02;                { make PEN=L  IFC=H  ATN=L}
    Address (Addr,MLA,MTA);
    Source_Operation (MLA);
    if BusError then exit;
    Port[$316] := $03;                { make PEN=L  IFC=H · ATN=H}
    for i :=1 to length(M) do          { send messages }
     begin
      message := M[i];
      Source_Operation(integer (message));
     end;
    Source_OPeration ($0D);           { send CR delimiter }
    Source_Operation ($0A);           { send LF delimiter }
end;

procedure Acceptor_Operation (var DByte : 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 }
    DByte      := $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 : integer ; var D : DataFormat ; Num : integer);
begin
    Port[$316] := $02;                { make REN=L IFC=H ATN=L }
```

```
    Address (Addr,MLA,MTA);
    Source_Operation (MTA);
    if BusError then exit;
    Port[$313] := $91;              { ALi Bio CUo CLIo          }
    Port[$311] := $00;              { 160=R;DAV=R,NRFD=NDAC=T   }
    Port[$312] := $C0;              { make RDAC=L NRFD=L        }
    Port[$316] := $03;              { make REN=L  IFC=H  ATN=H  }
    D := '';                        { clear Data  }
    for i := 1 to num do
       begin
          Acceptor_Operation (DByte);
          D := D+Char(DByte);
       end;
    Port[$313] := $88;              { Alo  Bio  CUIi  CLIo      }
    Port[$311] := $08;              { 160=T;DAV=T,NRFD=NDAC=R   }
    Port[$316] := $02;              { make REN=L  IFC=H  ATN=L  }
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 SPolling (Addr : integer ; var Dbyte : byte);
begin
    Port[$316] := $02;              { make REN=L  IFC=H  ATN=L  }
    Address (Addr,MLA,MTA);
    Source_Operation (UNL);
    if BusError then exit;
    Source_Operation (MTA);
    Source_Operation (SPE);
    Port[$313] := $91;              { Ali  Bio  CUIo  CLIi      }
    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 (DByte);
    Port[$313] := $88;              { Alo  Bio  CUIi  CLIo      }
    Port[$311] := $0B;              { 160=T;DAV=T,NRFD=NDAC=R   }
    Port[$316] := $02;              { make REN=L  IFC=H  ATN=L  }
    Port[$312] := $0C;              { make DVA=H  EOI=H         }
    Source_Operation (SPD);
    source_Operation (UNT);
end;

begin
    BusError   := false;
    Port[$317] := $88;
    Init_source;
    IFC;
```

```
    DCL;
end. { IEEE488 Unit }
```

```
Unit Menu;

Interface

Uses   Crt,Dos;

Const  single   ='┌┐└┘─│';
       double   ='╔╗╚╝═║';

Type   str8     = string[8];
       str12    = string[12];
       str30    = string[30];
       str80    = string[80];
       switch   = (on,off);
       CharSet  = set of char;
       OneScreen = array[1..25,1..80,1..2] of byte;
       Screen   = record
                    ScreenData : OneScreen;
                    row,col    : byte;
                  end;

var    IsColorCard        : boolean;
       LastActive         : byte;
       VideoSegment       : word;
       Firstkey,Lastkey   : char;

Function  DirFiName(fname : str30) : str30;
Function  Upper(strg : string) : string;
Function  Space(num : byte ) : string;
Function  Ctoi(strg : string ) : integer;
Function  Exist(FileName : string ) : boolean;
Procedure Tone(freq,time:word);
Procedure Color(T,B : byte );
Procedure Say(x,y : byte; message : str80);
Procedure Wdrt(x,y : byte; message : str80);
Procedure Shs(SW : switch);
Procedure Ssl(SW : switch);
Procedure Scur(SW : switch);
Procedure Suda(SW : switch);
Procedure Svcr;
Procedure Rscr;
Procedure SvcrT(var scr : Screen);
Procedure RscrF(scr : Screen);
Procedure Frame(border  : str8; tx,ty,bx,by : byte; title : str80);
Procedure Sftc(T,B : byte);
Procedure Prompt(col,row : byte; Menudata,Pmsg : str80);
Procedure Menuto(var option : byte);
Procedure Let(var strg; strglen : byte; Valid : CharSet);
Procedure ReadLet;

Implementation

Type   GetPtr    = ^GetRec;
       GetRec    = record
```

```
                data            : string;
                DataSeg,DataOfs : word;
                row,col,datalen : byte;
                ValidSet        : CharSet;
                pre,next        : GetPtr;
        end;

        PromptPtr  =  ^PromptRec;
        PromptRec  =   record
                       x,y,Fntspc   : byte;
                       Hilgt        : str80;
                       msg          : string;
                       PromptNo     : byte;
                       pre,next     : PromptPtr;
                    end;
        ScreenPtr  =  ^ScreenRec;
        ScreenRec  =   record
                       ScrData : Screen;
                       next    : ScreenPtr;
                    end;
Const   BiosSeg       = $40;
        Pcnt          : byte = 0;
        Udsl          : byte = 0;
        Hslt          : boolean = true;
        Uuda          : boolean = true;
        Slide         : boolean = false;
        FirstGet      : GetPtr  = nil;
        LastGet       : GetPtr  = nil;
        FirstPrompt   : PromptPtr  = nil;
        LastPrompt    : PromptPtr  = nil;
        LastScr       : ScreenPtr  = nil;

var     NowGet        : GetPtr;
        NowPrompt     : PromptPtr;
        NowScr        : ScreenPtr;
        ch            : char;
        Option,Hpc,Tpc,Bpc,Tmc,Bmc,Tlc,Blc,Tftc,Bftc : byte;
        bufhead       : word absolute BiosSeg:$1A;
        buftail       : word absolute BiosSeg:$1C;
        bufstart      : word absolute BiosSeg:$80;
        bufend        : word absolute BiosSeg:$82;

Function HeapAvail : boolean;
begin
    HeapAvail := (memavail > 4096);
end;

Procedure Tone(freq,time : word);
begin
    sound(freq);
    delay(time);
    nosound;
end;

Procedure Color(T,B : byte);
```

```
begin
    Textcolor(T);
    Textbackground(B);
end;

Procedure Say(x,y : byte; message : str80);
begin
    gotoXY(x,y);
    write(message);
end;

Procedure Wdrt( x,y : byte; message : str80);
var  i        : byte;
    location  : word;
begin
    dec(x); dec(y);
    location := (2*x) + (y*160);
    for i := 1 to length(message) do
      begin
       mem [VideoSegment:location] := ord(message[i]);
       inc(location,2);
      end;
end;


Procedure Wver(x,y : byte; message : str80);
var  i        : byte;
    location  : word;
begin
    dec(x); dec(y);
    location := (2*x) + (y*160);
    for i := 1 to length(message) do
      begin
       mem [VideoSegment:location] := ord(message[i]);
       inc(location,160);
      end;
end;


Procedure Wkbd(strg : string);
var      i,key : byte;
        tail  : word;
begin
    tail := buftail;
    if length(strg) > 0 then
    for i := 1 to length(strg) do
       begin
           inc(tail,2);
           if tail = 62 then tail := 30;
           if tail <> bufhead then
             begin
                 key := ord(strg[i]);
                 mem[BiosSeg:buftail] := key;
                 buftail := tail;
             end
```

```
            else begin tone(500,100); exit; end;
          end;
    end;

Procedure Wfnk(strg : string);
var     i,key,tail:word;
begin
    tail := buftail;
    if length(strg) > 0 then
    for i := 1 to length(strg) do
       begin
           inc(tail,2);
           if tail = 62 then tail :=30;
           if tail <> bufhead then
             begin
                 key := ord(strg[i]);
                 key := swap(key);
                 move(key,mem[BiosSeg:buftail],2);
                 buftail := tail;
             end
             else begin tone(500,100); exit; end;
             end;
       end;

Function upper(strg : string) : string;
var     i : byte;
begin
    if length(strg) > 0 then
    for i := 1 to length(strg) do strg[i] := upcase(strg[i]);
    upper := strg;
end;


Function rtrim(strg : string) : string;
var     len : byte absolute strg;
begin
    while (len > 0) and (strg[length(strg)] = #32) do
         delete(strg,length(strg),1);
    rtrim := strg;
end;

Function ltrim(strg : string) : string;
var     len : byte absolute strg;
begin
    while (len > 0) and (strg[1] = #32) do
         delete(strg,1,1);
    ltrim := strg;
end;

Function alltrim(strg : string) : string;
begin
    alltrim := rtrim(ltrim(strg)) ;
end;

Function Replicate(ch : char; num : byte) : string;
```

```
var strg : string;
    len  : byte absolute strg;
begin
     fillchar(strg[1],num,ch);
     len := num;
     Replicate := strg;
end;


Function Space(num : byte) : string;
begin
     Space := Replicate(#32,num);
end;

Function CtoI(strg : string) : integer;
var      code,number : integer;
begin
     val(strg,number,code);
     if code = 0 then CtoI := number
     else CtoI := 0;
end;

Function Exist(FileName : string) : boolean;
var      f : file;
begin
     assign(f,FileName);
     {$I-} reset(f); close(f); {$F+}
     Exist := (IOresult = 0);
end;

Procedure Shs(SW : switch);
begin
     if SW = on then Hslt := True
     else Hslt := False;
end;

Procedure Spc(H,T,B : byte);
begin
     Hpc := H;
     Tpc := T;
     Bpc := B;
end;

Procedure Ssl(SW : switch);
begin
     if SW = ON then Slide := True
     else Slide := False;
end;


Procedure Sct(Pch,Pcl : byte);
var      reg : registers;
begin
    with reg do
        begin
```

```
            ah := $01;
            ch := Pch;
            cl := Pcl;
        end;
        intr($10,reg);
end;


Procedure Scur(SW : switch);
begin
    if SW = off then Sct(13,0)
    else if IsColorCard then Sct(7,7)
    else Sct(13,13);
end;

Procedure Smc(T,B : byte);
begin
    Tmc := T; Bmc := B;
end;


Procedure Intp;
begin
    Slide := false;
    Uuda  := True;
end;


Procedure Suda(SW : switch);
begin
    if SW = ON then Uuda := True
    else Uuda := False;
end;

Procedure SvcrT(var scr : Screen);
begin
    with scr do begin
    move(mem[VideoSegment:$0000],ScreenData,SizeOF(ScreenData));
    row := whereY; col := whereX;
    end;
end;

Procedure RscrF(scr : Screen);
begin
    with scr do begin
    move(ScreenData,mem[VideoSegment:$0000],SizeOf(ScreenData));
    GotoXY(col,row);
    end;
end;


Procedure Svcr;
begin
    if HeapAvail then
      begin
```

```
          new(NowScr);
          with NowScr^ do begin
          SvcrT(ScrData);
          NowScr^ .next := LastScr;
          LastScr := NowScr;
        end;
      end
     else write('Heap overflow');
end;


Procedure Rscr;
begin
    if LastScr <> nil then
    with LastScr^ do begin
    RscrF(ScrData);
    NowScr := LastScr;
    LastScr := LastScr^.next;
    dispose(NowScr);
  end
 else write('No screen saved');
end;

Procedure Sftc(T,B : byte);
begin
    Tftc := T;
    Bftc := B;
end;


Procedure Frame( border  : str8; tx,ty,bx,by : byte; title : str80);
var     i,OWMn,OWMx,OMN,OMX : Word;
        OldAttr,start,len : byte;
        strg : str80;
begin
    OWMn := WindMin;
    OWMx := WindMax;
    window(tx,ty,bx,by);
    clrscr;
    len := bx-tx-1;
    strg := border[1] + Replicate(border[5],len) + border[2];
    Wdrt(tx,ty,strg);
    if Title <> #00 then
      begin
          OldAttr := TextAttr;
          color(Tftc,Bftc);
          start := (length(strg) div 2) - (length(title) div 2);
          OMN := WindMin;
          OMX := WindMax;
          window(1,1,80,25);
          gotoXY(tx+start,ty);
          write(title);
          WindMin := OMN;
          WindMax := OMX;
          TextAttr := OldAttr;
```

```
      end;
    strg := border[3] + replicate(border[6],len) + border[4];
    Wdrt(tx,by,strg);
    len := by-ty-1;
    strg := Replicate(border[7],len);
    Wver(tx,ty+1,strg);
    len := by-ty-1;
    strg := Replicate(border[8],len);
    Wver(bx,ty+1,strg);
    WindMin := OWMn;
    WindMax := OWMx;
end;

Procedure Cprompt;
begin
    while FirstPrompt <> nil do begin
        NowPrompt := FirstPrompt;
        FirstPrompt := FirstPrompt^.next;
        dispose(NowPrompt);
    end;
    FirstPrompt := nil;
    LastPrompt := nil;
    Pcnt := 0;
end;


Procedure WrtNPrompt(NowPrompt : PromptPtr);
var     OldX,OldY,OldAttr : byte;
begin
    OldAttr := TextAttr;
    with NowPrompt^ do
        begin
            say(x,y,Hilgt);
            OldX := whereX; OldY := whereY;
            TextColor(Hpc);
            say(x+Fntspc,y,Hilgt[Fntspc+1]);
            say(OldX,OldY,'');
        end;
    TextAttr := OldAttr;
end;


Procedure Prompt(col,row : byte; Menudata, Pmsg : str80);
begin
    if HeapAvail then
      begin
        new(NowPrompt);
        inc(Pcnt);
        with NowPrompt^ do
            begin
              msg := Pmsg;
              X := col; Y := row;
              PromptNo := Pcnt;
              Hilgt := Menudata;
              Fntspc := 0;
```

```
        while (Hilgt[Fntspc+1] = #32) and (Fntspc <= length(Hilgt)) do
        Inc(Fntspc);
        WrtNPrompt(NowPrompt);
        pre := LastPrompt;
      end;
    if FirstPrompt = nil then FirstPrompt := NowPrompt;
    if LastPrompt <> nil then LastPrompt^ .next := NowPrompt;
    LastPrompt := NowPrompt;
    LastPrompt^.next := nil;
    end
    else write('Heap overflow');
end;

Procedure Lkhl;
begin
    Wkbd(#27);
    Wfnk(#75);
    Wkbd(#13);
end;

Procedure Rkhl;
begin
    Wkbd(#27);
    Wfnk(#77);
    Wkbd(#13);
end;

Procedure Ukhl;
begin
    Wfnk(replicate(#75,Udsl));
end;

Procedure Dkhl;
begin
    Wfnk(replicate (#77,Udsl));
end;

Procedure Suds(num : byte);
begin
    Udsl := num;
    Suda(off);
end;


Procedure Menuto(var option : byte);

Procedure WrtPrompt (NowPrompt : PromptPtr);
var     OldAttr,OldWindMin,OldWindMax : word;
begin
    OldAttr := TextAttr;
    with NowPrompt^ do
        begin
          Color(Tpc,Bpc);
          say(x,y,Hilgt);
          color(Tmc,Bmc);
```

```
          OldWindMin := WindMin;
          OldWindMax := WindMax;
          window(1,1,80,25);
          gotoXY(1,25); clreol;
          write(msg+space(79-length(msg)));
          WindMin := OldWindMin;
          WindMax := OldWindMax;
          gotoXY(Fntspc+x,y);
          end;
      TextAttr := OldAttr;
end;

Procedure UpMove;
begin
    with NowPrompt^ do
    if pre = nil then
       NowPrompt := LastPrompt
    else NowPrompt := pre;
end;

Procedure DownMove;
begin
    with NOwPrompt^ do
      if next = nil then
        NowPrompt := FirstPrompt
      else Nowprompt := next;
end;


Procedure HotKey(var FirstKey : char; var NowPrompt : PromptPtr);
var      FirstCh           : char;
        OldPrompt          : PromptPtr;
        EndSearch,FirstTime : boolean;
begin
    OldPrompt := NowPrompt;
    NowPrompt := NowPrompt^.next;
    if NowPrompt = nil then NowPrompt := FirstPrompt;
    EndSearch := false;
    FirstTime := true;
    while (NowPrompt <> nil) and not EndSearch do
      with NowPrompt^ do begin
      FirstCh := Hilgt[Fntspc+1];
      if UpCase(FirstKey) = UpCase(FirstCh) then
        begin
          WrtNPrompt(OldPrompt);
          WrtPrompt(NowPrompt);
          if Hslt then FirstKey := #13;
          EndSearch := true;
        end
      else NowPrompt := next;
      if FirstTime and (not EndSearch) and (NowPrompt = nil) then
        begin
          NowPrompt := FirstPrompt;
          FirstTime := false;
        end;
```

```
      end;
      if Not EndSearch then NowPrompt := OldPrompt;
end;


begin { MenuTo }
     if FirstPrompt = nil then begin option := 0; exit; end;
     NowPrompt := FirstPrompt;
     if (option > LastPrompt^.PromptNo) or (option <= 0) then option :=1;
     while (NowPrompt^.PromptNo <> option) and (NOwprompt <> nil) do
       NowPrompt := Nowprompt^.next;
     if NowPrompt^.PromptNo <> option then
       NowPrompt := FirstPrompt;
     WrtPrompt(NowPrompt);
     firstKey := #00;
     while not (firstkey in [#13,#27]) do
       begin
         firstkey := readkey;
         lastkey := firstkey;
         if (firstkey = #00) and keypressed then
          begin
           WrtNPrompt(NowPrompt);
           if keypressed then begin
           lastkey := readkey;
           case lastkey of
             #75 : if Slide then Lkhl else UpMove;
             #77 : if Slide then Rkhl else DownMove;
             #72 : if Uuda then UpMove else Ukhl;
             #80 : if Uuda then DownMove else Dkhl;
             #71 : NowPrompt := FirstPrompt;
             #79 : NowPrompt := LastPrompt;
           end;
          end;
          if not keypressed then WrtPrompt(NowPrompt);
         end
         else if not (FirstKey in [#13,#27]) then
           HotKey(FirstKey,NowPrompt);
    end;
    WrtPrompt(NowPrompt);
    LastActive := NowPrompt^.PromptNo;
    if firstkey = #27 then option := 0
    else option := NowPrompt^.PromptNo;
    Cprompt;
    Intp;
    Hslt := true;
    Udsl := 0;
end; { MenuTo }


Procedure Clet;
begin
    while Firstget <> nil do
    begin
      NowGet   := FirstGet;
      FirstGet  := FirstGet^.next;
```

```
      dispose(NowGet);
    end;
  FirstGet := nil;
  LastGet  := nil;
end;


Procedure Slc(tcolor,bcolor : byte);
begin
    Tlc := tcolor;
    Blc := bcolor;
end;

Procedure Let(var strg; strglen : byte; Valid : CharSet);
var OldAttr  : byte;
   len      : byte absolute strg;
begin
    OldAttr := TextAttr;
    if HeapAvail then begin
      new(NowGet);
      with NowGet^ do begin
        row      := whereY;     col      := whereX;
        DataSeg  := Seg(strg);  DataOfs  := ofs(strg);
        datalen  := strglen;    ValidSet := Valid;
        move(strg,data,len+1);
        if length(data) > strglen then data := copy(data,1,strglen);
        pre := LastGet;
        color(Tlc,Blc);
        say(col,row,data+space(datalen-length(data)));
      end;
       if FirstGet = nil then FirstGet := NowGet;
       if LastGet <> nil then LastGet^.next := NowGet;
       LastGet := NowGet;
       LastGet^.next := nil;
      end;
      TextAttr := OldAttr;
end;


Procedure ReadLet;


Procedure ReadOneGet(x,y : byte; var strg : string;len : byte);
var      FirstTime : boolean;
        i,OldAttr : byte;

Procedure FunctionKey(var i : byte);
begin
    if keypressed then
      begin
        lastkey := readkey;
        case lastkey of
        #83 : delete(strg,i,1);
        #75 : if i > 1 then dec(i);
        #77 : if i <= length(strg) then inc(i);
```

```
        #71 : i := 1;
        #79 : begin i := length(strg); if i < len then inc(i); end;
        #72,#80 : ch := #13;
       end;
     end;
end;

begin { ReadOneGet }
     OLdAttr := TextAttr;
     FirstTime := True;
     color(Tlc,Blc);
     i := length(strg)+1;
     TextAttr := TextAttr + 128;
     repeat
       say(X,Y,strg+space(len-length(strg)));
       color(Tlc,Blc);
       gotoXY(x+i-1,y);
       ch := readkey;
       firstkey := ch;
       lastkey := firstkey;
       if FirstTime then
         begin
          if not (ch in [#0..#31]) then
            begin strg := '';i := 1; end;
          end;
        FirstTime := false;
        case ch of
          #00 : FunctionKey(i);
          ^H  : if i > 1 then begin dec(i); delete(strg,i,1); end;
          ^Y  : begin strg := '';i := 1; end;
          #1..#31:; { no action }
          else begin
            if ch in NowGet^.ValidSet then
              begin insert(ch,strg,i); inc(i); end
           else tone(500,100); end;
        end; { case }
        if i > len then begin tone(500,100); ch := #13; end;
        if length(strg) > len then delete(strg,len+1,1);
       until (ch in [#13,#27]) or (i > len);
       strg := Alltrim(strg);
       with NowGet^ do move (strg,mem[DataSeg:DataOfs],length(strg)+1);
       say(x,y,strg+space(len-length(strg)));
       TextAttr := OldAttr;
end;


begin { ReadGet }
     lastkey := #00;
     NowGet := FirstGet;
     while not (lastkey = #27) and (NowGet <> nil) do
      with NowGet^ do
        begin
          ReadOneGet(Col,row,data,datalen);
          case lastkey of
          #72 : begin
```

```
                NowGet := pre;
                if NowGet = nil then
                NowGet := LastGet;
               end;
          #80 : begin
                NowGet := next;
                if NowGet = nil then NowGet := FirstGet;
               end;
          else NowGet := next;
        end;
      end;
     Clet;
end;

Function DirFiName(fname : str30) : str30;
label   fin;

const   Lx = 9;
        Uy = 7;
        Rx = 71;
        Ly = 23;
        MaxFile = (Ly-Uy-1) * ((Rx-Lx) div 15);
var     Sr              : SearchRec;
        FiName          : str30;
        OLdWdMn,OldWdMx : word;
        option,fcount   : byte;
        SrSize,dirname  : string[40];
        dirmap          : array[1..MaxFile] of str12;
begin
    OLdWdMn := WindMin;
    OldWdMx := WindMax;
    Svcr;
    fcount :=0;
    if (fname ='') or (fname[length(fname)] in [':','\','.'])
      then fname := fname+'*.*';
    dirname := upper(FExpand (fname));
    frame(double,Lx,Uy,Rx,Ly,#32+dirname+#32);
    window(Lx+1,Uy+1,Rx-1,Ly-1);
    findfirst(fname,directory+archive,Sr);
    if doserror <> 0 then
      begin
        Wdrt(32,20,'Can''t find file !');
        repeat until keypressed;
        goto fin;
      end;
    while (doserror = 0) and (fcount < MaxFile) do
      begin
        inc(fcount);
        if whereX >= Rx-17 then gotoXY(1,whereY+1);
        with Sr do begin
          dirmap[fcount] := name;
          str(size:6,SrSize);
          prompt(whereX+1,whereY,#32+name+space(13-length(name)),
              +space(16)+'File .' + name + space(19-length(name)) +
              'Size .' + SrSize + ' bytes');
```

```
        end;

          findnext(Sr);
        end;
      option := 1;
      Scur(off);
      Shs(off);
      Suds((Rx-Lx) div 15);
      menuto(option);
      if option <> 0 then
        begin
          FiName := dirmap[option];
          DirFiName := FExpand(FiName);
        end
      else DirFiName := '';
   fin: WindMin := OldWdMn;
        WindMax := OLdWdMx;
        Rscr;
end;

BEGIN { unit menu }
     IsColorCard := (mem[$40:$49] <> mono);
     if IsColorCard then
       begin
         VideoSegment := $B800;
         Color(15,0);
         Spc(15,15,1);
         Smc(15,1);
         Slc(15,0);
         Sftc(14,5);
        end
       else
        begin
          VideoSegment := $B000;
          Color(7,0);
          Spc(15,0,15);
          Smc(0,15);
          Slc(0,15);
          Sftc(15,0);
        end;
END. { unit menu }
```

```
Program Project_2533;

Uses Crt,Dos,Printer,Menu,IEEE488;

const  MainOption   : byte   = 1;
       Quit         :boolean = false;

var    OutPutScreen : Screen;
       heaptop      : ^integer;
       hr,mr,sr,fr  : word;
       FileSave     : Text;
       FileData     : str30;

Procedure ReadYN(var ans : char);

begin
   repeat
     ans := upcase(readkey);
     if ans = #27 then exit;
   until ans in ['Y','N'];
   write(ans);
end;

Procedure GPIBmenu;
const    col = 3;
       option : byte = 1;
var      GPIBmenuScr : Screen;
       a1,a2,a3 : string[3];

Procedure GetAddr(var addr : integer);
label roj;
begin
roj : Svcr;
     Scur(on);
     frame(double,39,7,61,9,' ADDRESS [0..30] ');
     a1 := ' ';
     gotoxy(40,8);
     let(a1,21,[#48..#57]);
     readlet;
     Scur(off);
     Rscr;
     if Lastkey = #27 then exit;
     addr := CtoI(a1);
     if (addr > 30) or (a1 = '') then
       begin
         tone(500,250);
         goto roj;
       end;
end;

Procedure InitSource;
begin
   INIT_SOURCE;
end;
```

```
Procedure InterfaceClear;
begin
    IFC;
end;

Procedure DeviceClear;
begin
    DCL;
end;

Procedure SelectDeviceClear;
begin
    GetAddr(addr);
    if Lastkey = #27 then exit;
    SEND_COMMAND (ADDR,SDC);
end;

Procedure LocalLockOut;
begin
    GetAddr(addr);
    if Lastkey = #27 then exit;
    SEND_COMMAND (ADDR,LLO);
end;

Procedure GotoLocal;
begin
    GetAddr(addr);
    if Lastkey = #27 then exit;
    SEND_COMMAND (ADDR,GTL);
end;

Procedure GroupExecuteTrigger;
begin
    GetAddr(addr);
    if Lastkey = #27 then exit;
    SEND_COMMAND (ADDR,GET);
end;

Procedure ReceiveData;
label    wi1,wi2,wi3,wn;
var      ans : char;
       L,Loop : integer;

begin
    Svcr;
    Scur(on);
wi1: a1 := ''; a2 := ''; a3 := '';
    frame(double,39,7,62,12,' Receive Data ');
    say(41,8,'Address [0..30] : '); let(a1,2,[#48..#57]);
    say(41,9,'Bytes of data   : '); let(a2,3,[#48..#57]);
    say(41,10,'Loop [1..max]  : '); let(a3,3,[#48..#57]);
    readlet;
    if Lastkey = #27 then goto wi3;
    addr := CtoI(a1);
```

```
    if (addr > 30) or (a1 = '') or (a2 = '') or (a3 = '') then
      begin
       tone(500,250);
       goto wi1;
      end;
    num := Ctoi(a2);
    loop := Ctoi(a3);
    say(41,11,'Save data [Y/N] : '); ReadYN(ans);
    if ans = #27 then goto wi3;
    if ans = 'Y' then
      begin
      wn: FileData := '';
          Frame(double,27,14,75,16,' Enter File to Save ');
          gotoxy(28,15);
          let(FileData,47,[#0..#255]);
          readlet;
          if Lastkey = #27 then goto wi3;
          if FileData = ' ' then goto wn;
          FileData := upper(FileData);
          Assign(FileSave,FileData);
      end;
   frame(double,1,14,80,24,' DATA ');
   window(2,15,79,23);
   clrscr;
   For L := 1 to Loop do
      begin
        GET_DATA (ADDR,D,num);
        GetTime(hr,mr,sr,fr);
        writeln(D);
        writeln('Time=',hr,':',mr:2,':',sr:2,':',fr:2);
        writeln;
        if BusError then goto wi2;
        if ans = 'Y' then begin
          if Exist(FileData) then append(FileSave)
          else rewrite(FileSave);
   writeln(FileSave,D,'','Time=',hr:2,':',mr:2,':',sr:2,':',fr:2);
        close(FileSave);
        end;
      end;
    repeat until keypressed;
   wi2: window(1,1,80,25);
   wi3: Rscr;
       Scur(off);
   end;

Procedure SendMessage;
begin
    GetAddr(addr);
    if Lastkey = #27 then exit;
    M := '';
    Svcr;
    Scur(on);
    frame(double,24,11,75,13,' Message ');
    gotoxy(25,12);
    let(M,50,[#0..#255]);
```

```
    readlet;
    Rscr;
    Scur(off);
    if Lastkey = #27 then exit;
    SEND_MESSAGE (ADDR,M);
end;

Procedure SerialPolling;
begin
    GetAddr(addr);
    if Lastkey = #27 then exit;
    SPOLLING (ADDR,DByte);
    if BusError then exit;
    Svcr;
    frame(double,44,11,57,13,' DATA ');
    window(45,12,56,12);
    write(DByte);
    repeat until keypressed;
    window(1,1,80,25);
    Rscr;
end;

begin { GPIBmenu }
    SvcrT(GPIBmenuScr);
    repeat;
    Shs(off);
    frame(single,2,7,19,18,#00);
    prompt(col,      8,' Initsource     ','Initial the interface card.');
    prompt(col,whereY+1,' IFC           ','Interface clear.');
    prompt(col,whereY+1,' DCL           ','Device clear.');
    prompt(col,whereY+1,' SDC           ','Select device clear');
    prompt(col,whereY+1,' LLO           ','Local look out.');
    prompt(col,whereY+1,' GTL           ','Go to local.');
    prompt(col,whereY+1,' GET           ','Group execute trigger.');
    prompt(col,whereY+1,' Receive Data  ','Receive data from device.');
    prompt(col,whereY+1,' Send Message  ','Send message to device.');
    prompt(col,whereY+1,' Serail Polling','Serail Polling.');
    Ssl(on);
    menuto(option);
    case option of
      1 : InitSource;
      2 : InterfaceClear;
      3 : DeviceClear;
      4 : SelectDeviceClear;
      5 : LocalLockOut;
      6 : GotoLocal;
      7 : GroupExecuteTrigger;
      8 : ReceiveData;
      9 : SendMessage;
     10 : SerialPolling;
      0 : Option := LastActive;
    end; { case }
    until (LastKey = #27) or (LastKey = #75) or (LastKey = #77);
    RscrF(GPIBmenuScr);
    Shs(on);
```

```
end;

Procedure FileMenu;
const col = 26;
      option : byte = 1;
var   fileMenuScr : Screen;

Procedure ReadDir;
var      fname : str30;
begin
    fname := '*.*';
    Scur(on);
    Frame(double,16,15,64,17,' Enter drive and path ');
    gotoxy(17,16);
    let(fname,47,[#0..#255]);
    readlet;
    Scur(off);
    if LastKey = #27 then exit;
    FileData := DirFiName(fname);
end;


Procedure List;
type     pter = ^dataP;
       dataP = record
                 LineNo    : word;
                 data      : str80;
                 pre,next  : Pter;
               end;

var  heaplist               : ^integer;
     firstP,lastP,nowP,topP : Pter;
     source                 : text;
     Sfile                  : SearchRec;
     buffer                 : str80;
     OldAttr                : word;


Procedure ReadSource(var source : text);
var      CountLine : word;
begin
    firstP := nil;
    CountLine := 0;
    while not eof(source) do
    begin
      readln(source,buffer);
      inc(CountLine);
      new(nowP);
      with nowP^ do begin
        LineNo := CountLine;
        Data   := buffer;
        end;
   if firstP = nil then
   begin
   firstP := nowP;
```

```
    firstP^ .pre := nil;
    lastp := firstP;
    lastP^.next := nil;
    end
  else
    begin
      lastp^.next := nowP;
      nowP^.pre := lastP;
      lastP := nowP;
      lastP^.next := nil;
      end;
    end;
    close(source);
end;


Procedure Display(FileData : str30);
var      count    : integer;
        ch      : char;
        OldMin,OldMax : word;

Procedure Up_one_line;
begin
    if topP^.pre <> nil then
    begin
      topP := topP^.next;
      nowP := nowP^.next;
      gotoXY(1,1); insline;
      write(topP^.data);
    end;
end;

Procedure Down_one_line;
begin
    if nowP^.next <> nil then
    begin
      nowP := nowP^.next;
      topP := topP^.next;
      gotoXY(1,23);
      writeln;
      write(nowP^.data);
      end;
end;


Procedure Down_one_page;
begin
    count := 0;
    if nowP^.next = nil then exit;
    topP := nowP;
    clrscr;
    write(nowP^.data);
    while  (count < 22) and (nowP^.next <> nil) do
    begin
      inc(count);
```

```
      nowP := nowP^.next;
      writeln;
      write(nowP^.data);
      end;
end;

Procedure Write_one_page;
begin
    count := 0;
    clrscr;
    write(nowP^.data);
    while (count < 22) and (nowP^.next <> nil) do
    begin
    inc(count);
    nowP := nowP^.next;
    writeln;
    write(nowP^.data);
    end;
end;


Procedure Up_one_page;
begin
    count := 0;
    while (count < 44) and (nowP^.pre <> nil) do
      begin
        inc(count);
        nowP := nowP^.pre;
      end;
    topP := nowP;
    Write_one_page;
end;

Procedure First_line;
begin
    nowP := FirstP;
    topP := FirstP;
    Write_one_page;
end;

Procedure Last_line;
begin
    count := 0;
    nowP := lastP;
    while (count < 22) and (nowP^.pre <> nil) do
     begin
       inc(count);
      nowP := nowP^.pre;
      end;
      topP := nowP;
      Write_one_page;
end;


Function FillSpc(no : word) : str80;
```

```
var     strg : str80;
begin
    str(no,strg);
    FillSpc := strg + space(4-length(strg));
end;


begin { display }
     nowP := firstP;
     topP := firstP;
     ch := #00;
     OldMin := WindMin;
     OldMax := WindMax;
     Window(1,2,80,24);
     Write_one_page;
     while not (ch in ['Q',#27]) do
     begin
      Wdrt(70,1, 'Line : '+ FillSpc(topP^.LineNo));
      Wdrt(70,25,'Line : '+ FillSpc(NowP^.LineNo));
      ch := upcase(readkey);
      if ch = ^C then Halt(1);
      if (ch = #00) and keypressed then
       begin
        lastkey := readkey;
        case lastkey of
          #72 : Up_one_line;
          #80 : Down_one_line;
          #73 : Up_one_page;
          #81 : Down_one_page;
          #71 : First_line;
          #79 : Last_line;
        end;
       end;
     end;
     WindMin := OldMin;
     WindMax := OldMax;
end;


Procedure AssignFile(FileData : str30);
var     OldAttr : word;
begin
    OldAttr := TextAttr;
    if IsColorCard then
    begin
        textcolor(15);
        textbackground(1);
    end
  else
     begin
       textcolor(0);
       textbackground(7);
     end;
  gotoXY(1,25); clreol;
 write ('Command >> | Up | Down | PgDn | Home | End | Q)uit |');
```

```
   gotoXY(1,1); clreol;
   write('Reading File << ',FExpand(FileData));
   assign(source,FileData);
   reset(source);
   mark(heaplist);
   readsource(source);
   say(1,1,'File  >>  '+FileData); clreol;
   TextAttr := OldAttr;
end;

begin { List }
     clrscr;
     OldAttr := TextAttr;
     textcolor(14);
     textbackground(0);
     Scur(on);
     FileData := '';
     gotoXY(1,1);
     write('Enter file to display >'); clreol;
     Wdrt(16,3,'Press ENTER to dislay files ESC to cancel.)');
     let(FileData,56,[#0..#255]);
     readlet;
     Wdrt(16,3,'                                                    ');
     Scur(off);
     TextAttr := OldAttr;
     if Lastkey = #27 then exit;
     if FileData = '  then FileData := DirFiName(FileData);
     if Lastkey = #27 then exit;
     FileData := upper(FileData);
     if Exist(FileData) then
      begin
        AssignFile(FileData);
        display(FileData);
        release(heaplist);
      end
      else
        begin
          tone(500,100);
          Wdrt(32,20,' File not found ! ');
          repeat until keypressed;
          end;
end;

Procedure Print;
var      PrintOut : Text;
        DataOut  : str80;
        OldAttr  : word;
        ans      : char;
begin
    Svcr;
    Scur(on);
    FileData := '';
    Frame(double,16,15,64,17,' Enter File to Print ');
    Wdrt(16,19,'(Press ENTER to display files or ESC to cancel.)');
    gotoxy(17,16);
```

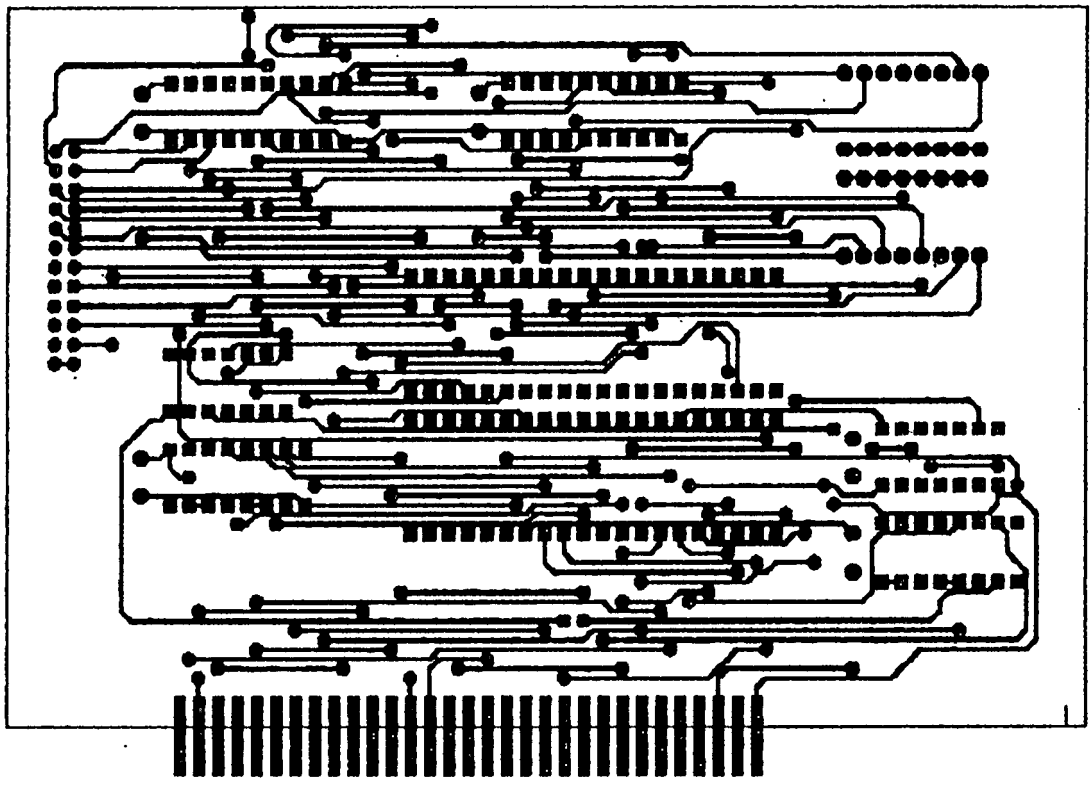
```
    let(FileData,47,[#0..#255]);
    readlet;
    Scur(off);
    Rscr;
    if Lastkey = #27 then exit;
    if FileData = '' then FileData := DirFiName(FileData);
    if Lastkey = #27 then exit;
    Frame(double,19,15,61,17,'');
    gotoxy(20,16);
    Scur(on);
    write(' Is your print ready? READY [Y/N] : ');
    ReadYN(ans);
    Scur(off);
    if ans = 'Y' then
      begin
       Frame(double,16,15,64,17,' Print File ');
       gotoxy(17,16);
       OldAttr := TextAttr;
       TextAttr := TextAttr+128;
       write(FExpand(FileData));
       TextAttr := OldAttr;
         if Exist(FileData) then
           begin
            assign(PrintOut,FileData);
            reset(PrintOut);
         while not EOF(PrintOut) do
           begin
            readln(PrintOut,DataOut);
            writeln(LST,DataOut);
         end;
       close(PrintOut);
      end
    else
      begin
       tone(500,100);
       Wdrt(32,20,'File not found !');
       repeat until keypressed;
      end;
   end;
end;


Procedure ChangeDir;
var      DirName : DirStr;
begin
    GetDir(0,DirName);
    Scur(on);
    Frame(double,16,15,64,17,' NEW DIRECTORY ');
    GotoXY(17,16); let(DirName,47,[#0..#255]);
    readlet;
    Scur(off);
    if Lastkey <> #27 then
      begin
       {$I-} ChDir(DirName); {$I+}
       if IOresult <> 0 then Wdrt(32,20,'Invalid directory');
```

```
      end;
end;

Procedure EraseFile;
var     DelName   : str30;
        f         : file;
        ans       : char;
        dir       : DirStr;
        name      : NameStr;
        ext       : ExtStr;
        OLdAttr   : word;

begin
    Svcr;
    Scur(on);
    DelName := '';
    Frame(double,16,15,64,17,'Enter File Name ');
    Wdrt(16,19,'(Press ENTER to display files or ESC to cancel.)');
    GotoXY(17,16); let(DelName,47,[#0..#255]);
    readlet;
    Scur(off);
    Rscr;
    if Lastkey <> #27 then
    begin
        if DelName = '' then DelName := DirFiName(DelName);
        if DelName <> '' then
              begin
                Sftc(142,4);
                Frame(double,20,15,60,17,' Warning ');
                FSplit(DelName,dir,name,ext);
                tone(900,400);
                Scur(on);
                Say(22,16, File >  ' + name + ext +' Erase (Y/N): ');
                ReadYN(ans);
                Scur(off);
                if ans = 'Y' then
                begin
                  assign(f,DelName);
                  {$I-} Erase(f); {$I+}
                  if IOresult <> 0 then
                  begin
                    tone(500,200);
                    Wdrt(30,23,'Can''t Erase This File');
                    repeat until keypressed;
                  end;
              end;
           end;
       end;
end;



begin { FileMenu }
     SvcrT(FileMenuScr);
     Frame(single,col-1,7,col+11,13,#00);
```

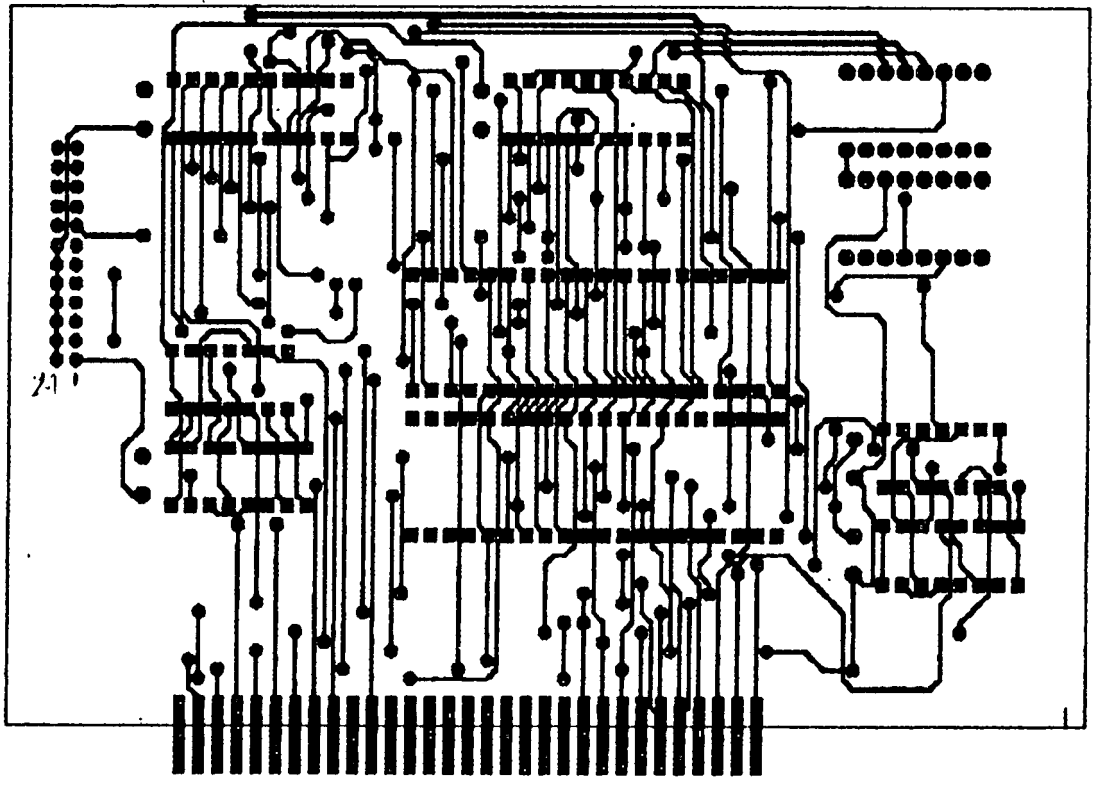
```
    prompt(col,      8,' Directory ','Display files.');
    prompt(col,whereY+1,' List      ','Display data from file.');
    prompt(col,whereY+1,' Print     ','Print data from file.');
    prompt(col,whereY+1,' ChDir     ','Change directory.');
    prompt(col,whereY+1,' Erase     ','Erase file.');
    Ssl(on);
    MenuTo(option);
    case option of
      1 : ReadDir;
      2 : List;
      3 : Print;
      4 : ChangeDir;
      5 : EraseFile;
      6 : option := LastActive;
    end;
    RscrF(FileMenuScr);
end;



Procedure OptionMenu;
const    col = 49;
       option : byte = 1;
var      OptionScr : Screen;


Procedure TimeSet;
label    tm;
var      hh,mm,ss :string[2];
begin
tm: Scur(on);
    hh := ''; mm := ''; ss :=  '';
    frame(double,45,13,60,17,' Enter Time ');
    say(47,14,'Hour    : '); let(hh,2,[#48..#57]);
    say(47,15,'Minute  : '); let(mm,2,[#48..#57]);
    say(47,16,'Second  : '); let(ss,2,[#48..#57]);
    readlet;
    Scur(off);
    if Lastkey = #27 then exit;
    hr := Ctoi(hh);
    mr := Ctoi(mm);
    sr := Ctoi(ss);
    if (hr > 23) or (mr > 59) or (sr > 59) then goto tm;
    SetTime(hr,mr,sr,0);
end;


Procedure DateSet;
var      yr,mn,dy : word;
       yi,mi,di : string[2];
begin
    Scur(on);
    di := '';mi := '';yi := '';
    frame(double,46,13,59,17, 'Enter Date ');
```

```
     say(48,14,'Day   : '); let(d1,2,[#48..#57]);
     say(48,15,'Month : '); let(m1,2,[#48..#57]);
     say(48,14,'Year  : '); let(y1,2,[#48..#57]);
     readlet;
     Scur(off);
     if Lastkey = #27 then exit;
     dy := Ctoi(d1);
     mn := Ctoi(m1);
     yr := Ctoi(y1);
     SetDate(yr,mn,dy);
end;

begin { OptionMenu }
     SvcrT(OptionScr);
     frame(single,col-1,7,col+8,10,#00);
     prompt(col,      8,' Time  ','Set real time to system.');
     prompt(col,whereY+1,' Date  ','Set real date to system.');
     Ssl(on);
     menuto(option);
     case option of
      1 : TimeSet;
      2 : DateSet;
      0 : option := LastActive;
  end;
  RscrF(OptionScr);
end;


Procedure QuitMenu;
const    col = 73;
        option : byte = 1;
var      QuitScr : Screen;
begin { QuitMenu }
     SvcrT(QuitScr);
     frame(single,col-1,7,col+6,10,#00);
     prompt(col,      8,' No  ', Cancel.');
     prompt(col,WhereY+1,' Yes  , Exit program.');
     Ssl(on);
     menuto(option);
     case option of
       1 : Quit := false;
       2 : Quit := true;
       0 : option := LastActive;
   end;
   RscrF(QuitScr);
end;

Procedure MainMenu;
begin { MAinmenu }
     Suda(off);
     prompt( 3,5, ' GPIB ','> General Purpose Interface Bus command <');
 prompt(26,5,' File ','> Directory . List . Print . Chdir . Erase <');
 prompt(49,5,' Option ','> Time . Date <');
 prompt(73,5,' Quit ','> Exit Program <');
 MenuTo(MainOption);
```

```
  case MainOption of
    1 : GPIBmenu;
    2 : Filemenu;
    3 : OptionMenu;
    4 : QuitMenu;
    0 : MainOption := LastActive;
    end;
  end;


BEGIN ( main )
      checksnow := true;
      directvideo := true;
      mark(heaptop);
      clrscr;
      Scur(off);
      frame(single,1,1,80,4,$00);
      say(22,2,'PARALLEL Interface Card for IBM PC');
      say(35,3, 'Version 0.1');
      if IsColorCard then color(0,7);
      repeat MainMenu;
      until Quit;
      release(heaptop);
      color(15,0);
      clrscr;
      frame(single,1,1,80,4,$00);
      say(23,2,'Computer Industrial');
      gotoxy(1,6);
      Scur(on);
END.
```



ด้าน Component



ด้าน Solder

Silicon Gate MOS 8255

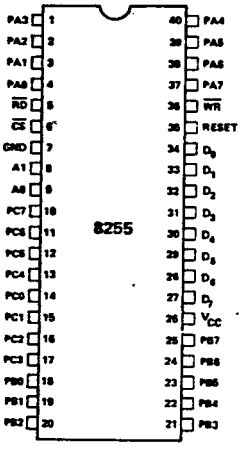
PROGRAMMABLE PERIPHERAL INTERFACE

- 24 Programmable I/O Pins
- Completely TTL Compatible
- Fully Compatible with MCS™-8 and MCS™-80 Microprocessor Families
- Direct Bit Set/Reset Capability Easing Control Application Interface
- 40 Pin Dual In-Line Package
- Reduces System Package Count

The 8255 is a general purpose programmable I/O device designed for use with both the 8008 and 8080 microprocessors. It has 24 I/O pins which may be individually programmed in two groups of twelve and used in three major modes of operation. In the first mode (Mode 0), each group of twelve I/O pins may be programmed in sets of 4 to be input or output. In Mode 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining four pins three are used for handshaking and interrupt control signals. The third mode of operation (Mode 2) is a Bidirectional Bus mode which uses 8 lines for a bidirectional bus, and five lines, borrowing one from the other group, for handshaking.

Other features of the 8255 include bit set and reset capability and the ability to source 1mA of current at 1.5 volts. This allows darlington transistors to be directly driven for applications such as printers and high voltage displays.

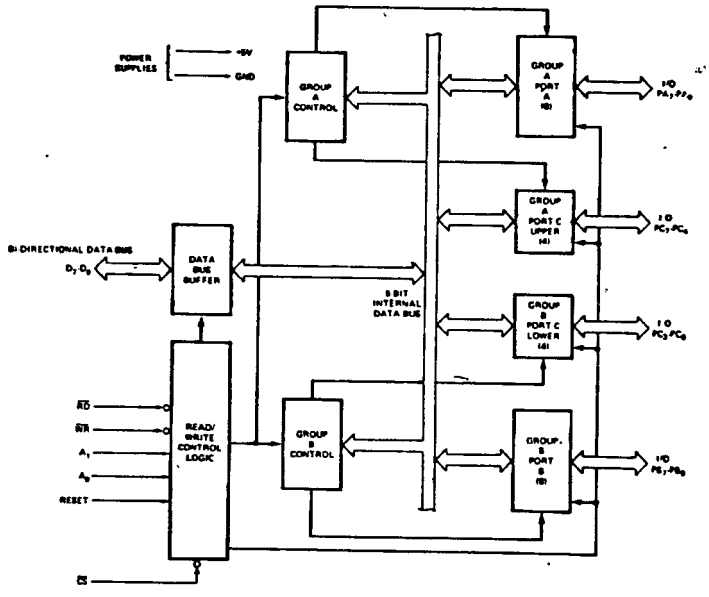
PIN CONFIGURATION



PIN NAMES

D_7–D_0	DATA BUS (BI-DIRECTIONAL)
RESET	RESET INPUT
$\overline{CS}$	CHIP SELECT
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A0, A1	PORT ADDRESS
PA7-PA0	PORT A (BIT)
PB7-PB0	PORT B (BIT)
PC7-PC0	PORT C (BIT)
V_{CC}	+5 VOLTS
GND	0 VOLTS

8255 BLOCK DIAGRAM



8255 BASIC FUNCTIONAL DESCRIPTION

General

The 8255 is a Programmable Peripheral Interface (PPI) device designed for use in 8080 Microcomputer Systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the 8080 system bus. The functional configuration of the 8255 is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

This 3-state, bi-directional, eight bit buffer is used to interface the 8255 to the 8080 system data bus. Data is transmitted or received by the buffer upon execution of INput or OUTput instructions by the 8080 CPU. Control Words and Status information are also transferred through the Data Bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the 8080 CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

($\overline{CS}$)

Chip Select: A "low" on this input pin enables the communication between the 8255 and the 8080 CPU.

($\overline{RD}$)

Read: A "low" on this input pin enables the 8255 to send the Data or Status information to the 8080 CPU on the Data Bus. In essence, it allows the 8080 CPU to "read from" the 8255.

($\overline{WR}$)

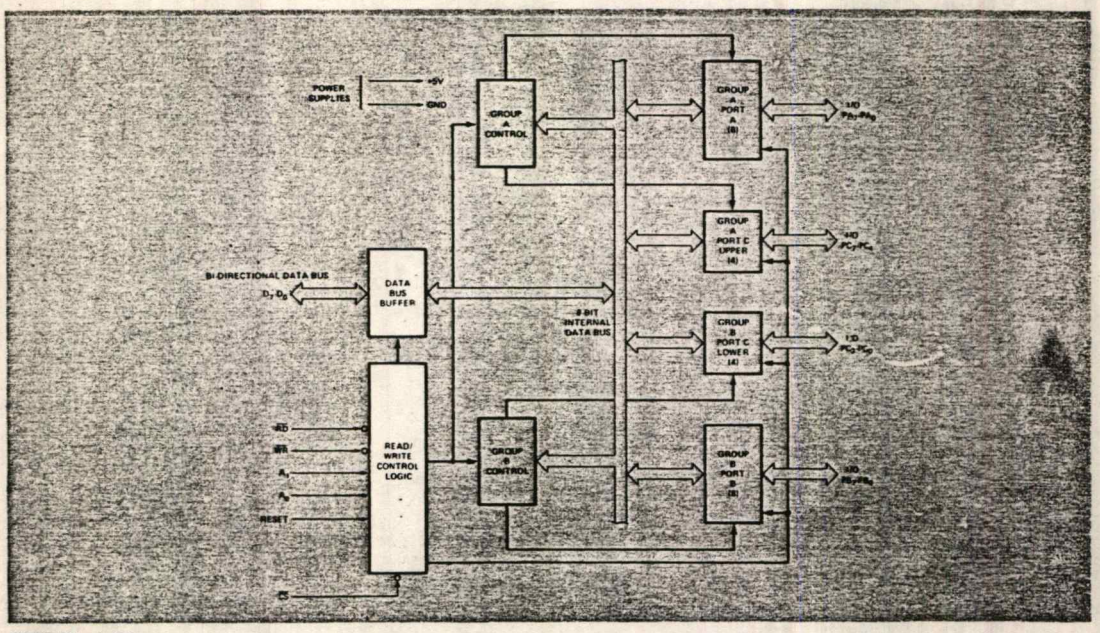
Write: A "low" on this input pin enables the 8080 CPU to write Data or Control words into the 8255.

(A_0 and A_1)

Port Select 0 and Port Select 1: These input signals, in conjunction with the $\overline{RD}$ and $\overline{WR}$ inputs, control the selection of one of the three ports or the Control Word Register. They are normally connected to the least significant bits of the Address Bus (A_0 and A_1).

8255 BASIC OPERATION

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	INPUT OPERATION (READ)
0	0	0	1	0	PORT A → DATA BUS
0	1	0	1	0	PORT B → DATA BUS
1	0	0	1	0	PORT C → DATA BUS
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	DATA BUS → PORT A
0	1	1	0	0	DATA BUS → PORT B
1	0	1	0	0	DATA BUS → PORT C
1	1	1	0	0	DATA BUS → CONTROL
					DISABLE FUNCTION
X	X	X	X	1	DATA BUS → 3-STATE
1	1	0	1	0	ILLEGAL CONDITION



8255 Block Diagram

(RESET)

Reset: A "high" on this input clears all internal registers including the Control Register and all ports (A, B, C) are set to the input mode.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the 8080 CPU "outputs" a control word to the 8255. The control word contains information such as "mode", "bit set", "bit reset" etc. that initializes the functional configuration of the 8255.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control Logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Control Group A – Port A and Port C upper (C7-C4)
Control Group B – Port B and Port C lower (C3-C0)

The Control Word Register can Only be written into. No Read operation of the Control Word Register is allowed.

Ports A, B, and C

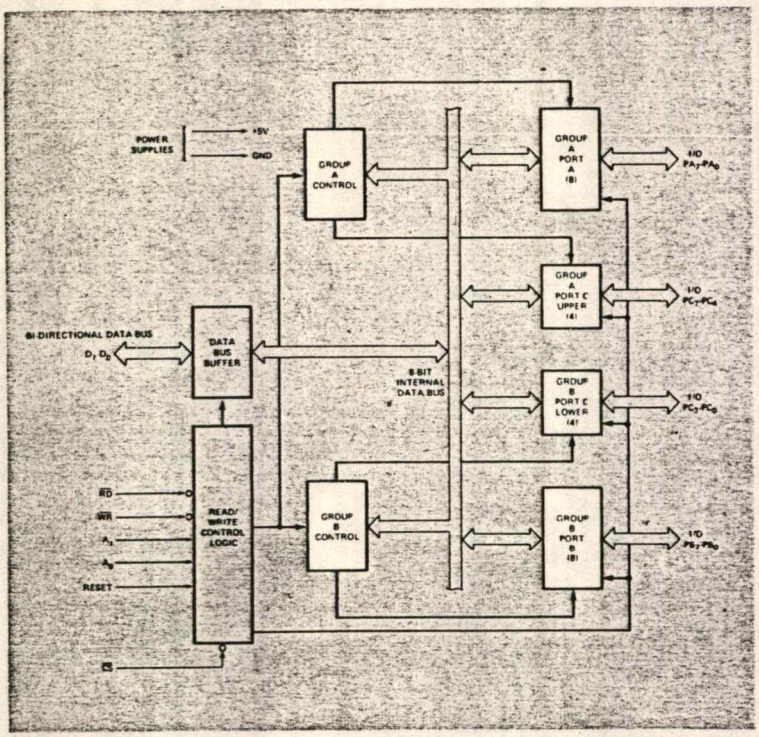
The 8255 contains three 8-bit ports (A, B, and C). All can be configured in a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255.

Port A: One 8-bit data output latch/buffer and one 8-bit data input latch.

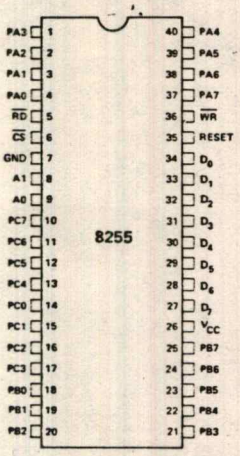
Port B: One 8-bit data input/output latch/buffer and one 8-bit data input buffer.

Port C: One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with Ports A and B.

8255 BLOCK DIAGRAM



PIN CONFIGURATION



PIN NAMES

D_7–D_0	DATA BUS (BI-DIRECTIONAL)
RESET	RESET INPUT
$\overline{CS}$	CHIP SELECT
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A0, A1	PORT ADDRESS
PA7-PA0	PORT A (BIT)
PB7-PB0	PORT B (BIT)
PC7-PC0	PORT C (BIT)
V_{CC}	+5 VOLTS
GND	0 VOLTS

8255 DETAILED OPERATIONAL DESCRIPTION

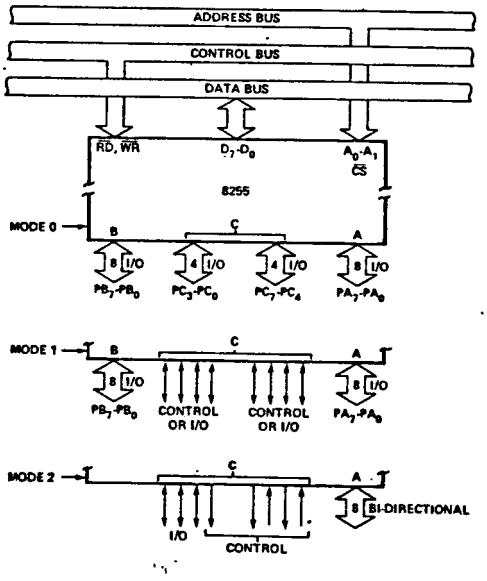
Mode Selection

There are three basic modes of operation that can be selected by the system software:

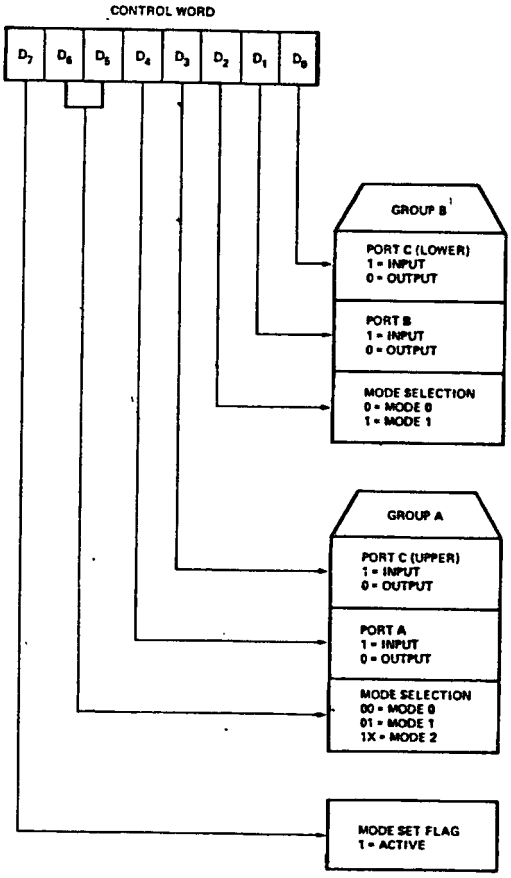
Mode 0 – Basic Input/Output
Mode 1 – Strobed Input/Output
Mode 2 – Bi-Directional Bus

When the RESET input goes "high" all ports will be set to the Input mode (i.e., all 24 lines will be in the high impedance state). After the RESET is removed the 8255 can remain in the Input mode with no additional initialization required. During the execution of the system program any of the other modes may be selected using a single OUTput instruction. This allows a single 8255 to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be "tailored" to almost any I/O structure. For instance; Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.



Basic Mode Definitions and Bus Interface

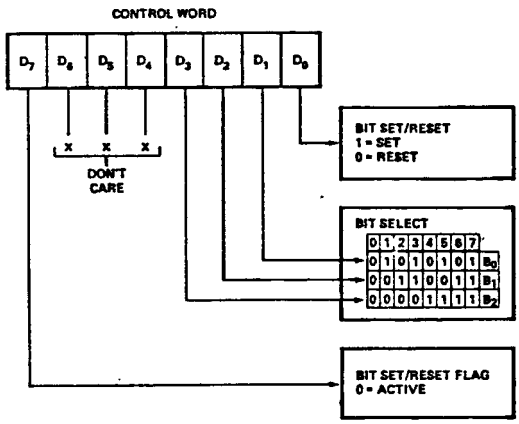


Mode Definition Format

The Mode definitions and possible Mode combinations may seem confusing at first but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 8255 has taken into account things such as efficient PC board layout, control signal definition vs PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature

Any of the eight bits of Port C can be Set or Reset using a single OUTput instruction. This feature reduces software requirements in Control-based applications.



Bit Set/Reset Format

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were data output ports.

Interrupt Control Functions

When the 8255 is programmed to operate in Mode 1 or Mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from Port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the Bit set/reset function of Port C.

This function allows the Programmer to disallow or allow a specific I/O device to interrupt the CPU without effecting any other device in the interrupt structure.

INTE flip-flop definition:

(BIT-SET) – INTE is SET – Interrupt enable
(BIT-RESET) – INTE is RESET – Interrupt disable

Note: All Mask flip-flops are automatically reset during mode selection and device Reset.

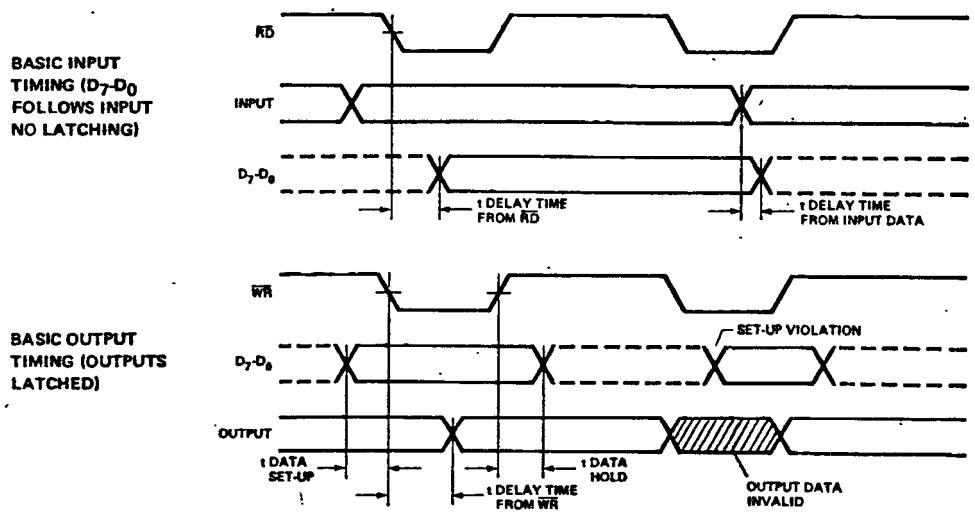
Operating Modes

Mode 0 (Basic Input/Output)

This functional configuration provides simple Input and Output operations for each of the three ports. No "handshaking" is required, data is simply written to or read from a specified port.

Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports.
- Any port can be input or output.
- Outputs are latched.
- Inputs are not latched.
- 16 different Input/Output configurations are possible in this Mode.



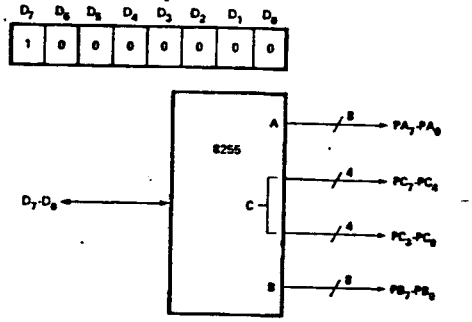
Mode 0 Timing

MODE 0 PORT DEFINITION CHART

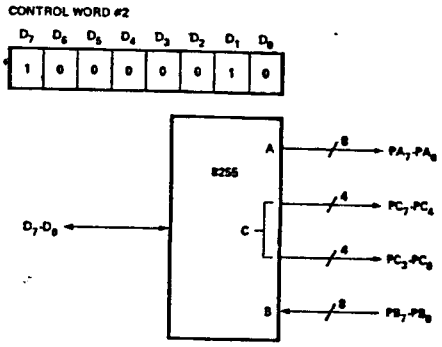
A		B		GROUP A			GROUP B	
D_4	D_3	D_1	D_0	PORT A	PORT C (UPPER)	#	PORT B	PORT C (LOWER)
0	0	0	0	OUTPUT	OUTPUT	0	OUTPUT	OUTPUT
0	0	0	1	OUTPUT	OUTPUT	1	OUTPUT	INPUT
0	0	1	0	OUTPUT	OUTPUT	2	INPUT	OUTPUT
0	0	1	1	OUTPUT	OUTPUT	3	INPUT	INPUT
0	1	0	0	OUTPUT	INPUT	4	OUTPUT	OUTPUT
0	1	0	1	OUTPUT	INPUT	5	OUTPUT	INPUT
0	1	1	0	OUTPUT	INPUT	6	INPUT	OUTPUT
0	1	1	1	OUTPUT	INPUT	7	INPUT	INPUT
1	0	0	0	INPUT	OUTPUT	8	OUTPUT	OUTPUT
1	0	0	1	INPUT	OUTPUT	9	OUTPUT	INPUT
1	0	1	0	INPUT	OUTPUT	10	INPUT	OUTPUT
1	0	1	1	INPUT	OUTPUT	11	INPUT	INPUT
1	1	0	0	INPUT	INPUT	12	OUTPUT	OUTPUT
1	1	0	1	INPUT	INPUT	13	OUTPUT	INPUT
1	1	1	0	INPUT	INPUT	14	INPUT	OUTPUT
1	1	1	1	INPUT	INPUT	15	INPUT	INPUT

MODE 0 CONFIGURATIONS

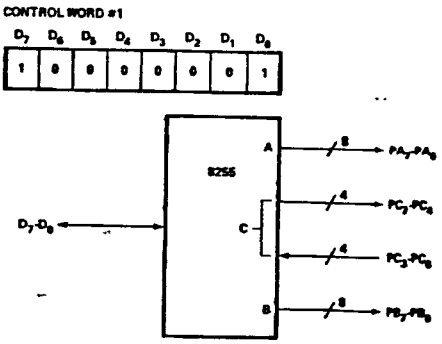
CONTROL WORD #0



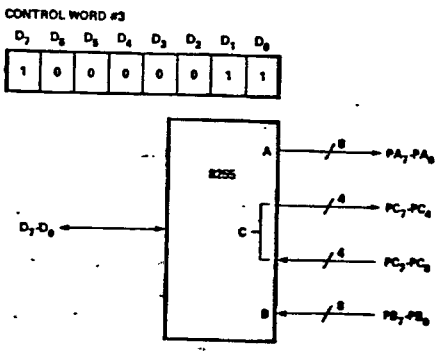
CONTROL WORD #2



CONTROL WORD #1

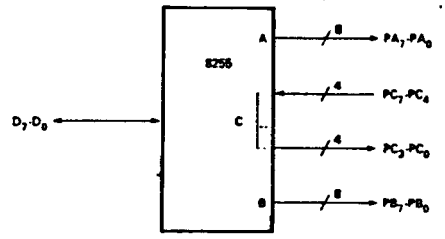


CONTROL WORD #3



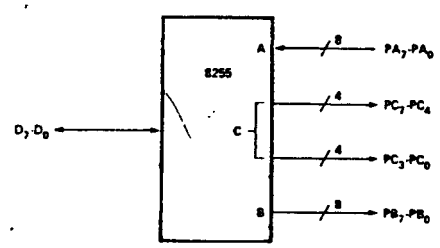
CONTROL WORD =4

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	0



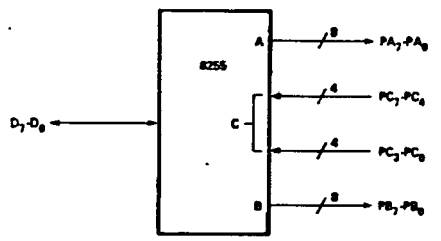
CONTROL WORD =8

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	0



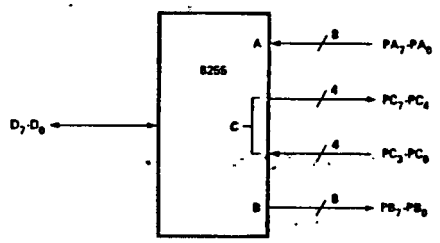
CONTROL WORD =5

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	0	1



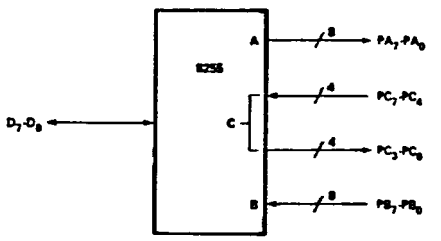
CONTROL WORD =9

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	1



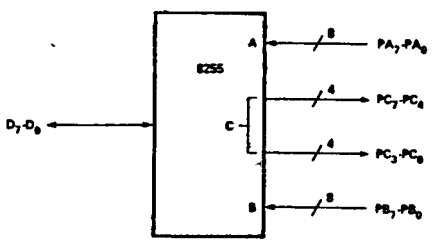
CONTROL WORD =6

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	0



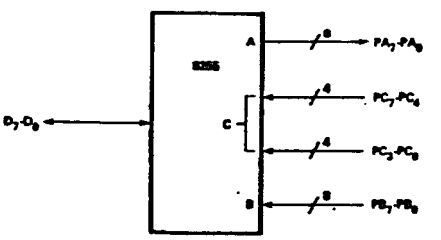
CONTROL WORD =10

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	0



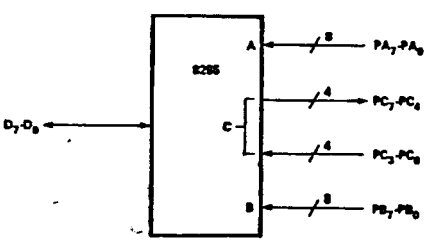
CONTROL WORD =7

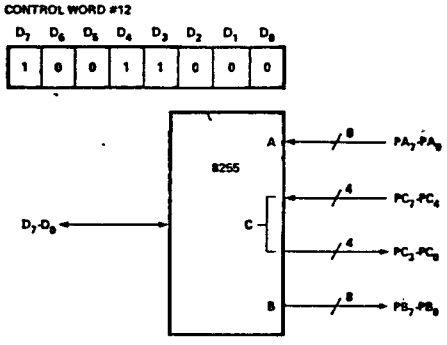
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	1	0	1	1

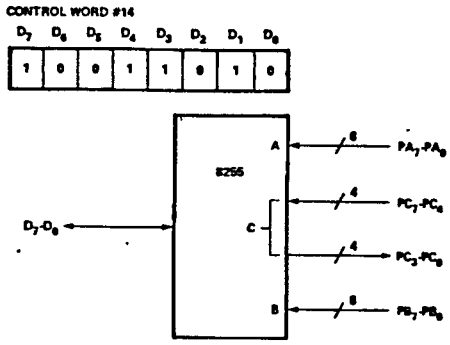


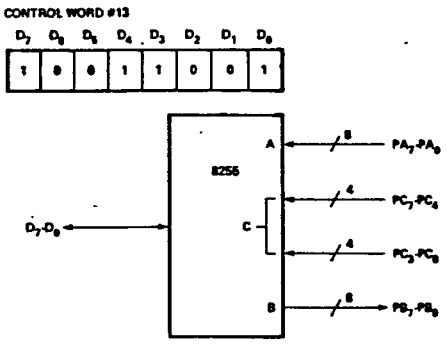
CONTROL WORD =11

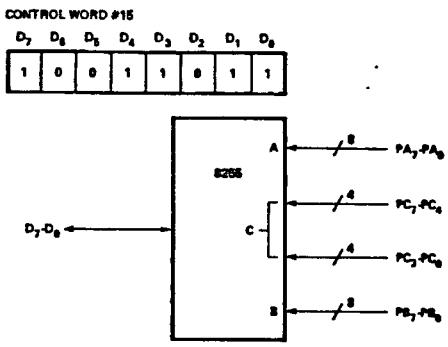
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	1	1











Operating Modes

Mode 1 (Strobed Input/Output)

This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or "handshaking" signals. In Mode 1, Port A and Port B use the lines on Port C to generate or accept these "handshaking" signals.

Mode 1 Basic Functional Definitions:

- Two Groups (Group A and Group B)
- Each group contains one 8-bit data port and one 4-bit control/data port.
- The 8-bit data port can be either input or output. Both inputs and outputs are latched.
- The 4-bit port is used for control and status of the 8-bit data port.

Input Control Signal Definition

$\overline{STB}$ (Strobe Input)

A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F)

A "high" on this output indicates that the data has been loaded into the input latch; in essence, an acknowledgement. IBF is set by the falling edge of the STB input and is reset by the rising edge of the RD input.

INTR (Interrupt Request)

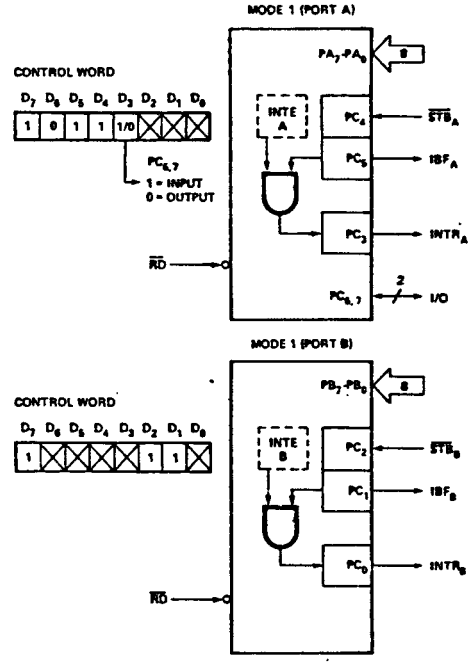
A "high" on this output can be used to interrupt the CPU when an input device is requesting service. INTR is set by the rising edge of $\overline{STB}$ if IBF is a "one" and INTE is a "one". It is reset by the falling edge of $\overline{RD}$. This procedure allows an input device to request service from the CPU by simply strobing its data into the port.

INTE A

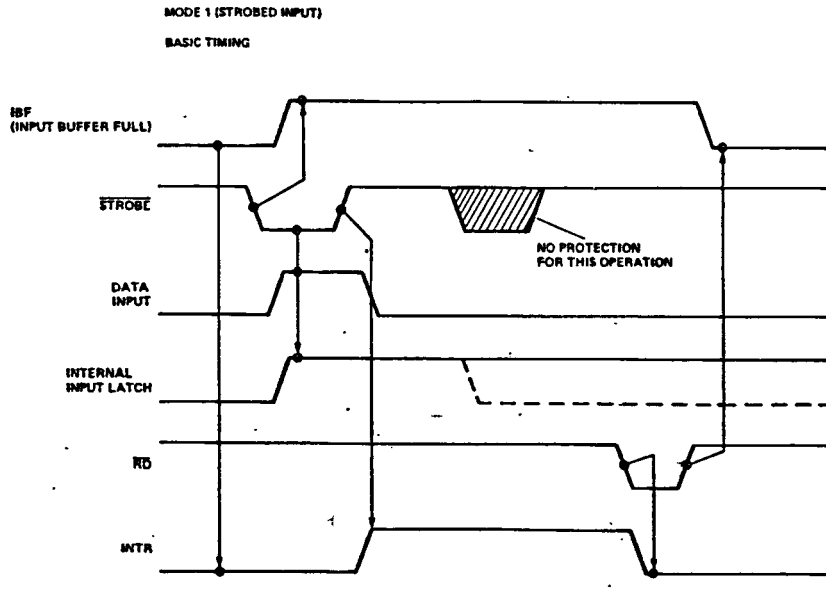
Controlled by bit set/reset of PC_4.

INTE B

Controlled by bit set/reset of PC_2.



Mode 1 Input



Basic Timing Input

Output Control Signal Definition

$\overline{OBF}$ (Output Buffer Full F/F)

The $\overline{OBF}$ output will go "low" to indicate that the CPU has written data out to the specified port. The OBF F/F will be set by the rising edge of the WR input and reset by the falling edge of the $\overline{ACK}$ input signal.

$\overline{ACK}$ (Acknowledge Input)

A "low" on this input informs the 8255 that the data from Port A or Port B has been accepted. In essence, a response from the peripheral device indicating that it has received the data output by the CPU.

INTR (Interrupt Request)

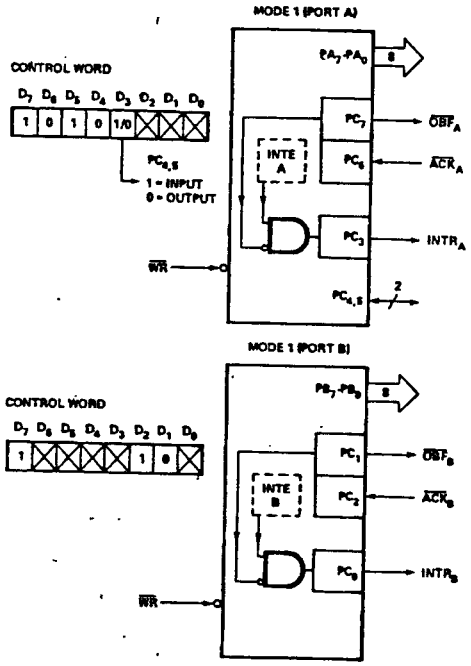
A "high" on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set by the rising edge of $\overline{ACK}$ if $\overline{OBF}$ is a "one" and INTE is a "one". It is reset by the falling edge of $\overline{WR}$.

INTE A

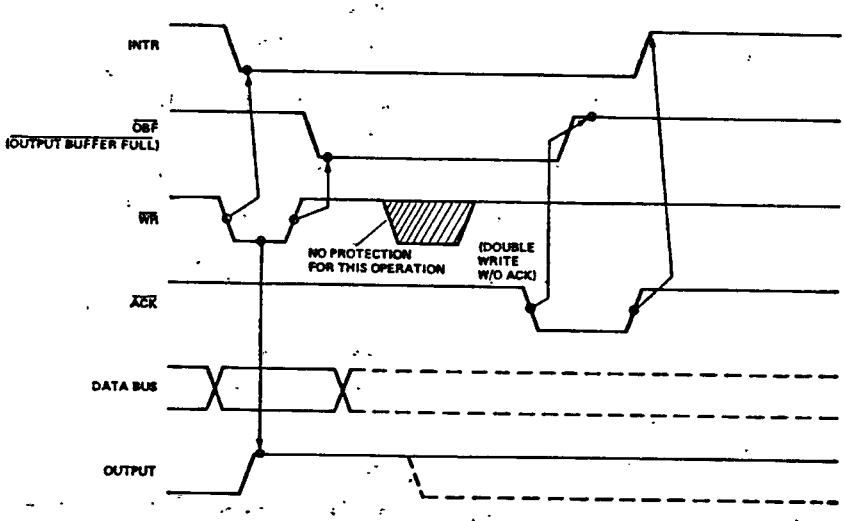
Controlled by bit set/reset of PC_6.

INTE B

Controlled by bit set/reset of PC_2.



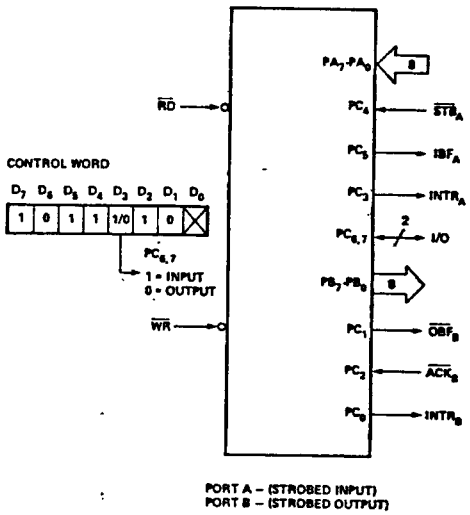
Mode 1 Output

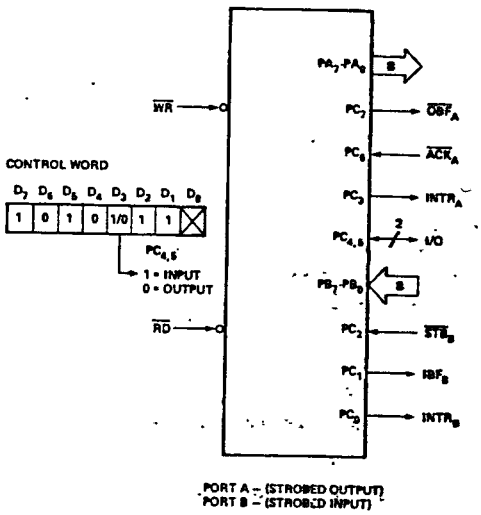


Basic Timing Output

Combinations of Mode 1

Port A and Port B can be individually defined as input or output in Mode 1 to support a wide variety of strobed I/O applications.





Operating Modes

Mode 2 (Strobed Bi-Directional Bus I/O)

This functional configuration provides a means for communicating with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bi-directional bus I/O). "Handshaking" signals are provided to maintain proper bus flow discipline in a similar manner to Mode 1. Interrupt generation and enable/disable functions are also available.

Mode 2 Basic Functional Definitions:

- Used in Group A only.
- One 8-bit, bi-directional bus Port (Port A) and a 5-bit control Port (Port C).
- Both inputs and outputs are latched.
- The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A).

Bi-Directional Bus I/O Control Signal Definition

INTR (Interrupt Request)

A high on this output can be used to interrupt the CPU for both input or output operations.

Output Operations

$\overline{OBF}$ (Output Buffer Full)

The $\overline{OBF}$ output will go "low" to indicate that the CPU has written data out to Port A.

$\overline{ACK}$ (Acknowledge)

A "low" on this input enables the tri-state output buffer of Port A to send out the data. Otherwise, the output buffer will be in the high-impedance state.

INTE 1 (The INTE Flip-Flop associated with $\overline{OBF}$)

Controlled by bit set/reset of PC_6.

Input Operations

$\overline{STB}$ (Strobe Input)

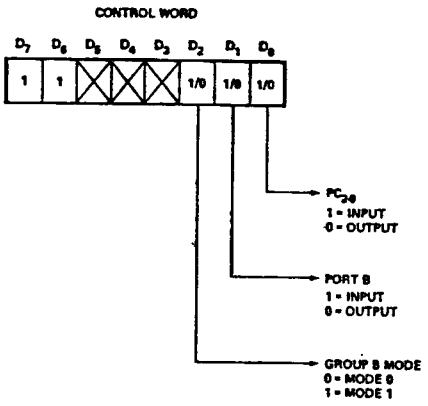
A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F)

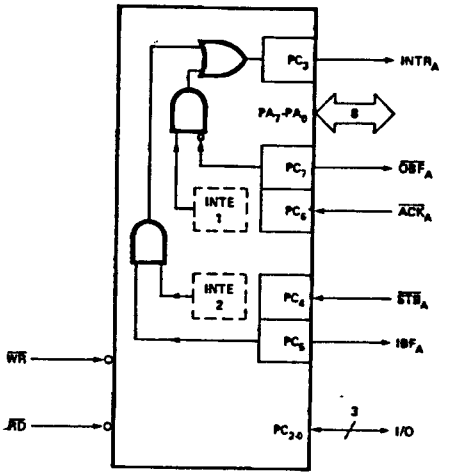
A "high" on this output indicates that data has been loaded into the input latch.

INTE 2 (The INTE Flip-Flop associated with IBF)

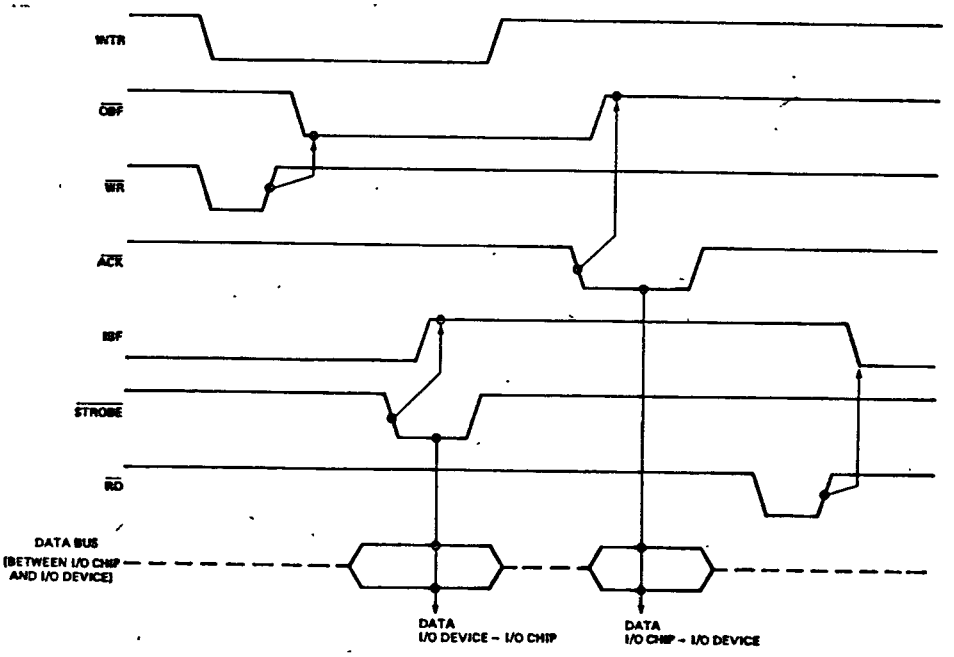
Controlled by bit set/reset of PC_4.



Mode 2 Control Word

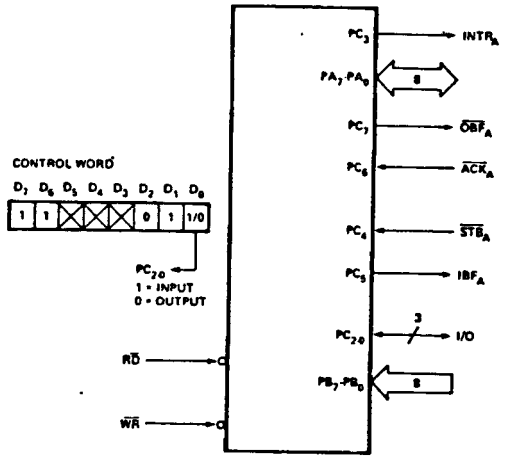


Mode 2

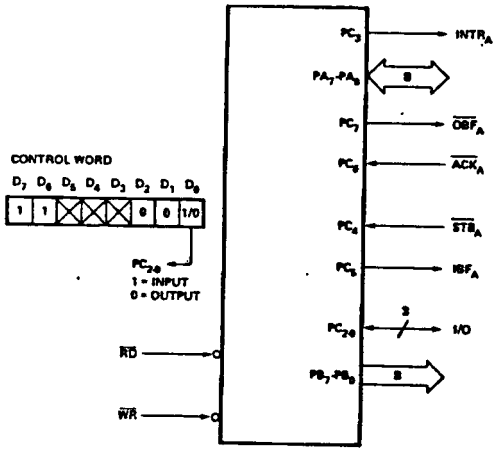


Mode 2 (Bi-directional) Timing

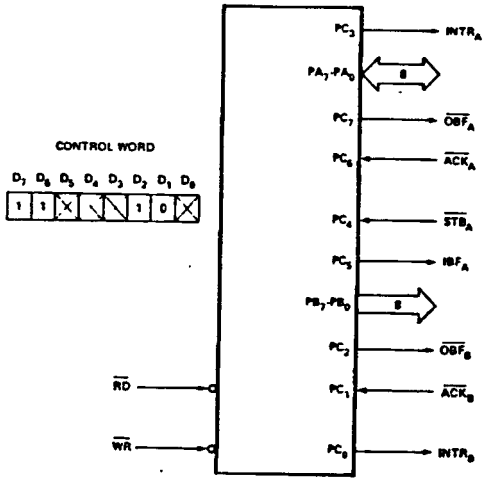
MODE 2 AND MODE 0 (INPUT)



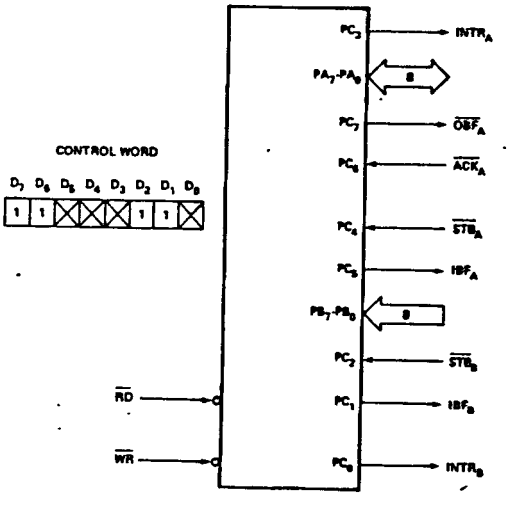
MODE 2 AND MODE 0 (OUTPUT)



MODE 2 AND MODE 1 (OUTPUT)



MODE 2 AND MODE 1 (INPUT)



Mode 2 Combinations

MODE DEFINITION SUMMARY TABLE

	MODE 0		MODE 1		MODE 2	
	IN	OUT	IN	OUT	GROUP A ONLY	
PA_0	IN	OUT	IN	OUT	↔	
PA_1	IN	OUT	IN	OUT	↔	
PA_2	IN	OUT	IN	OUT	↔	
PA_3	IN	OUT	IN	OUT	↔	
PA_4	IN	OUT	IN	OUT	↔	
PA_5	IN	OUT	IN	OUT	↔	
PA_6	IN	OUT	IN	OUT	↔	
PA_7	IN	OUT	IN	OUT	↔	
PB_0	IN	OUT	IN	OUT	—	MODE 0 OR MODE 1 ONLY
PB_1	IN	OUT	IN	OUT	—	
PB_2	IN	OUT	IN	OUT	—	
PB_3	IN	OUT	IN	OUT	—	
PB_4	IN	OUT	IN	OUT	—	
PB_5	IN	OUT	IN	OUT	—	
PB_6	IN	OUT	IN	OUT	—	
PB_7	IN	OUT	IN	OUT	—	
PC_0	IN	OUT	$INTR_B$	$INTR_B$	I/O	
PC_1	IN	OUT	IBF_B	$\overline{OBF}_B$	I/O	
PC_2	IN	OUT	$\overline{STB}_B$	$\overline{ACK}_B$	I/O	
PC_3	IN	OUT	$INTR_A$	$INTR_A$	$INTR_A$	
PC_4	IN	OUT	$\overline{STB}_A$	I/O	$\overline{STB}_A$	
PC_5	IN	OUT	IBF_A	I/O	IBF_A	
PC_6	IN	OUT	I/O	$\overline{ACK}_A$	$\overline{ACK}_A$	
PC_7	IN	OUT	I/O	$\overline{OBF}_A$	$\overline{OBF}_A$	

Special Mode Combination Considerations

There are several combinations of modes when not all of the bits in Port C are used for control or status. The remaining bits can be used as follows:

If Programmed as Inputs –
All input lines can be accessed during a normal Port C read.

If Programmed as Outputs –
Bits in C upper (PC_7-PC_4) must be individually accessed using the bit set/reset function.

Bits in C lower (PC_3-PC_0) can be accessed using the bit set/reset function or accessed as a threesome by writing into Port C.

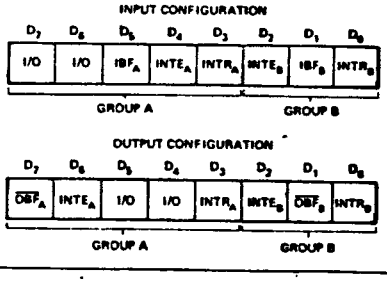
Source Current Capability on Port B and Port C

Any set of eight output buffers, selected randomly from Ports B and C can source 1mA at 1.5 volts. This feature allows the 8255 to directly drive Darlington type drivers and high-voltage displays that require such source current.

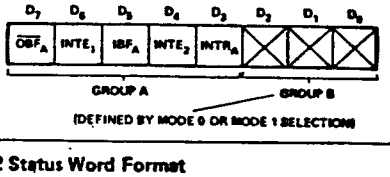
Reading Port C Status

In Mode 0, Port C transfers data to or from the peripheral device. When the 8255 is programmed to function in Modes 1 or 2, Port C generates or accepts "hand-shaking" signals with the peripheral device. Reading the contents of Port C allows the programmer to test or verify the "status" of each peripheral device and change the program flow accordingly.

There is no special instruction to read the status information from Port C. A normal read operation of Port C is executed to perform this function.



Mode 1 Status Word Format

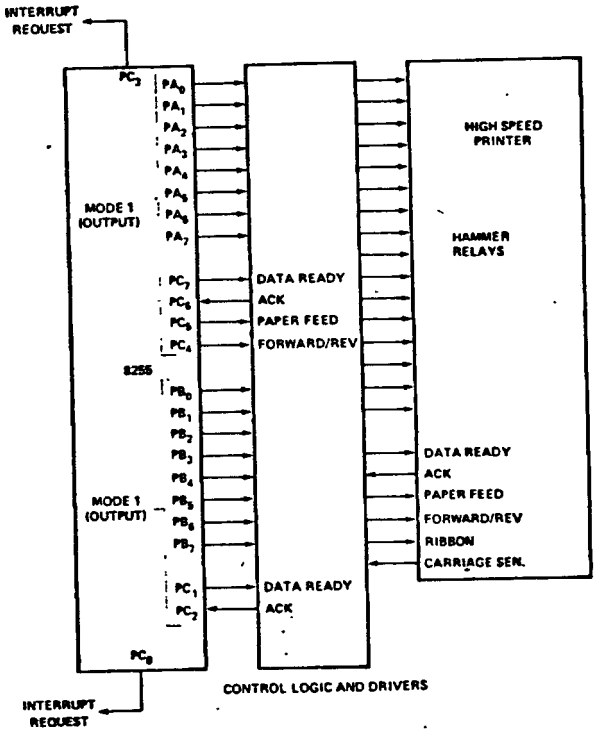


Mode 2 Status Word Format

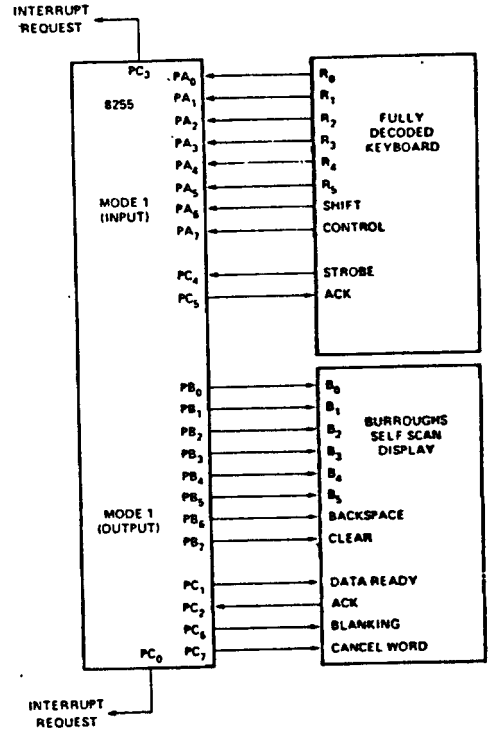
APPLICATIONS OF THE 8255

The 8255 is a very powerful tool for interfacing peripheral equipment to the 8080 microcomputer system. It represents the optimum use of available pins and is flexible enough to interface almost any I/O device without the need for additional external logic.

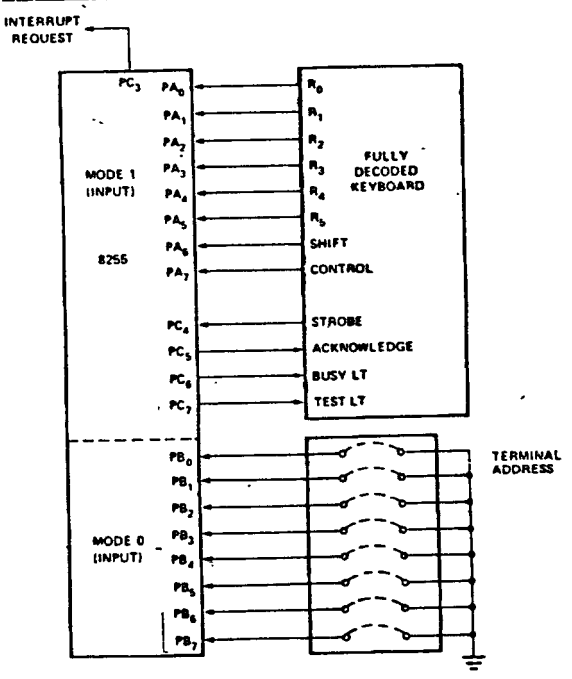
Each peripheral device in a Microcomputer system usually has a "service routine" associated with it. The routine manages the software interface between the device and the CPU. The functional definition of the 8255 is programmed by the I/O service routine and becomes an extension of the systems software. By examining the I/O devices interface characteristics for both data transfer and timing, and matching this information to the examples and tables in the Detailed Operational Description, a control word can easily be developed to initialize the 8255 to exactly "fit" the application. Here are a few examples of typical applications of the 8255.



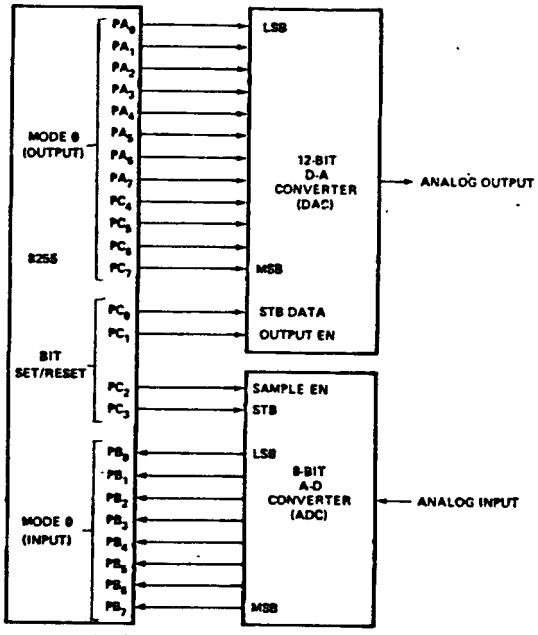
Printer Interface



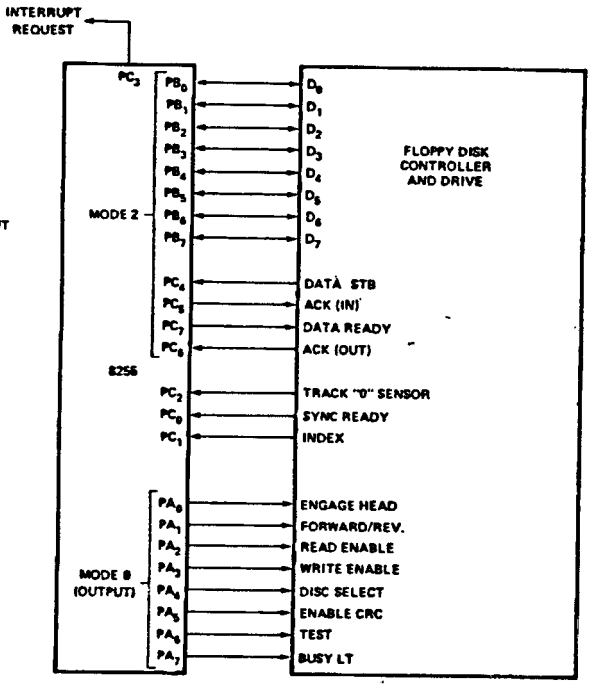
Keyboard and Display Interface



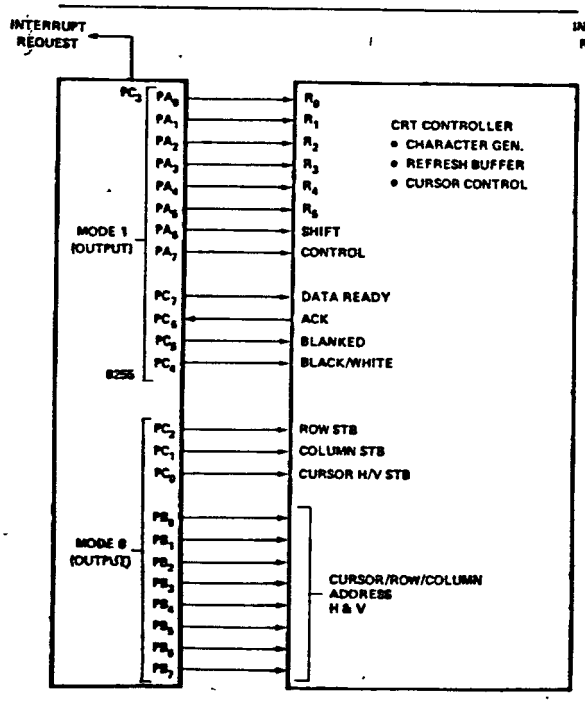
Keyboard and Terminal Address Interface



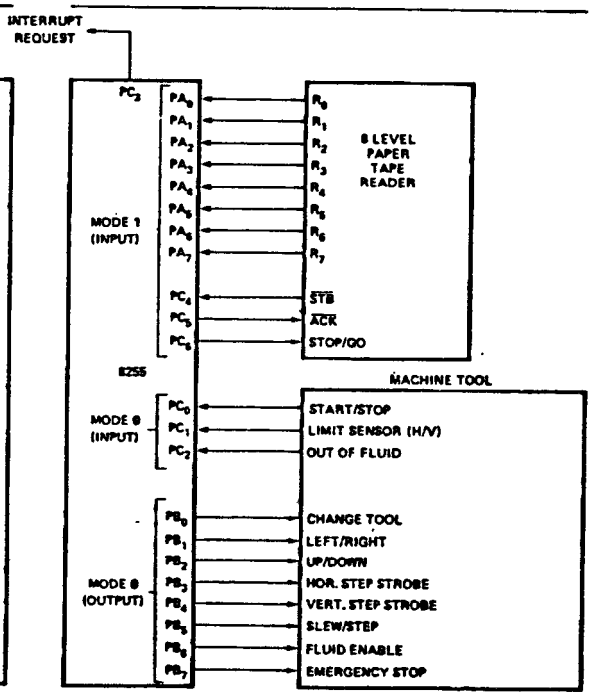
Digital to Analog, Analog to Digital



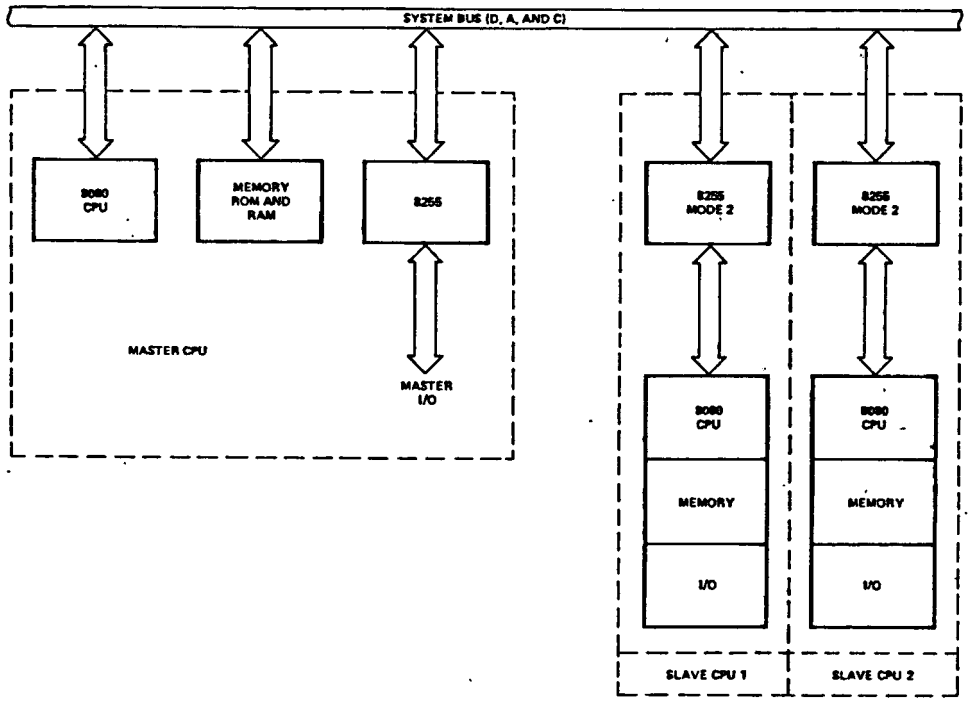
Basic Floppy Disc Interface



Basic CRT Controller Interface



Machine Tool Controller Interface



Distributed Intelligence Multi-Processor Interface

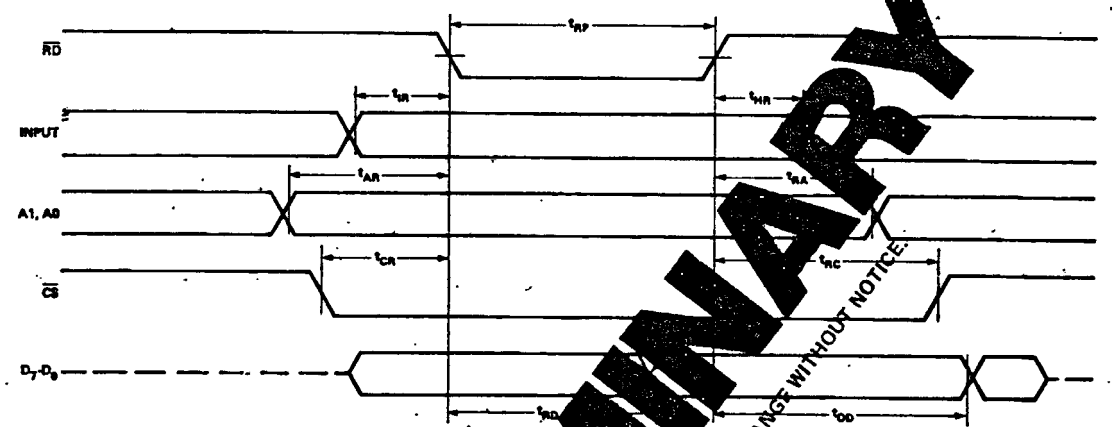
D.C. CHARACTERISTICS $T_A = 0°C$ to $70°C$; $V_{CC} = +5V \pm 5\%$; $V_{SS} = 0V$

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage			.8	V	
V_{IH}	Input High Voltage	2.0			V	
V_{OL}	Output Low Voltage			.4	V	$I_{OL} = 1.6mA$
V_{OH}	Output High Voltage	2.4			V	$I_{OH} = -50\mu A$ (-100μA for D.B. Port)
I_{OH}[1]	Darlington Drive Current		2.0		mA	$V_{OH} = 1.5V$, $R_{EXT} = 390\Omega$
I_{CC}	Power Supply Current		40		mA	

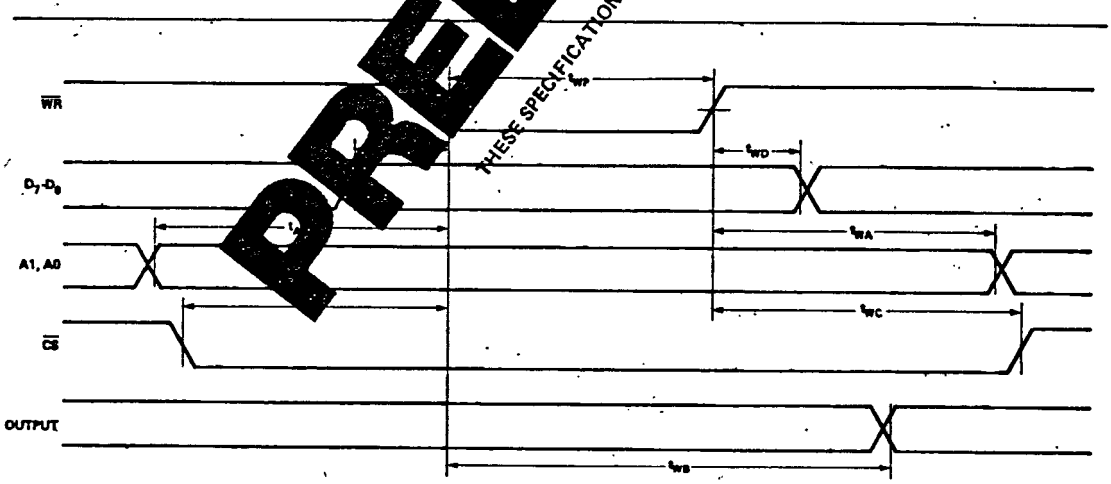
NOTE:
1. Available on 8 pins only.

A.C. CHARACTERISTICS $T_A = 0°C$ to $70°C$; $V_{CC} = +5V \pm 5\%$; $V_{SS} = 0V$

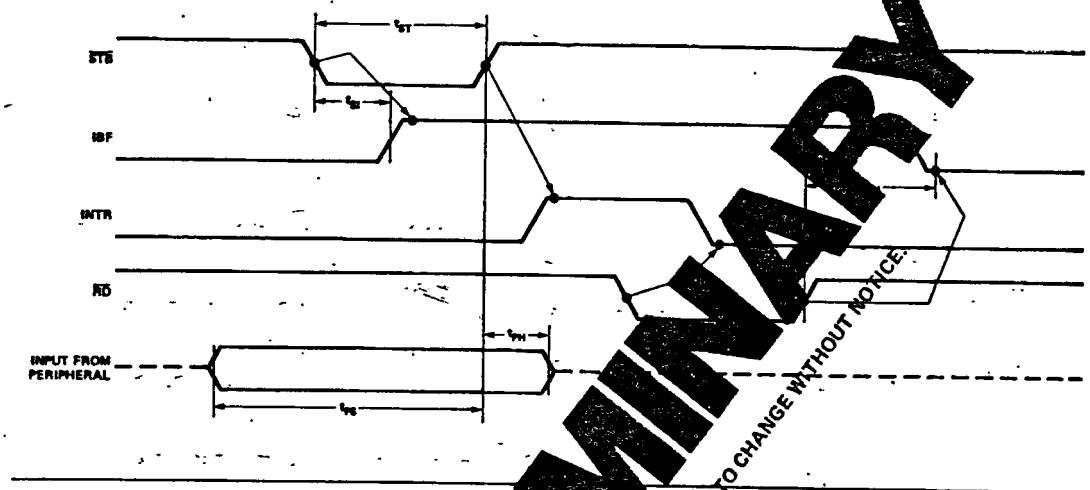
Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
t_{WP}	Pulse Width of $\overline{WR}$		250		ns	
t_{DW}	Time D.B. Stable Before $\overline{WR}$		10			
t_{WD}	Time D.B. Stable After $\overline{WR}$		10			
t_{AW}	Time Address Stable Before $\overline{WR}$				ns	
t_{WA}	Time Address Stable After $\overline{WR}$				ns	
t_{CW}	Time CS Stable Before $\overline{WR}$				ns	
t_{WC}	Time CS Stable After $\overline{WR}$				ns	
t_{WB}	Delay From $\overline{WR}$ To Output				ns	
t_{RP}	Pulse Width of $\overline{RD}$				ns	
t_{IR}	$\overline{RD}$ Set-Up Time		50		ns	
t_{HR}	Input Hold Time		10		ns	
t_{RD}	Delay From $\overline{RD}$ = 0 To Sy[illegible]		200		ns	
t_{OD}	Delay From $\overline{RD}$ = 1 To S[illegible]		100		ns	
t_{AR}	Time Address Stable [illegible]		25		ns	
t_{CR}	Time $\overline{CS}$ Stable Be[illegible]		25		ns	
t_{AK}	Width Of $\overline{ACK}$ Pu[illegible]		100		ns	
t_{ST}	Width Of $\overline{STB}$ [illegible]		100		ns	
t_{PS}	Set-Up Ti[illegible] For [illegible]pheral		200		ns	
t_{PH}	Hold Time [illegible] [illegible]pheral		10		ns	
t_{RA}	Hold Time for $A_{1,0}$ After $\overline{RD}$ = 1		10		ns	
t_{RC}	Hold Time For CS After $\overline{RD}$ = 1.		10		ns	
t_{AD}	Time From $\overline{ACK}$ = 0 To Output (Mode 2)		200		ns	
t_{KD}	Time From $\overline{ACK}$ = 1 To Output Floating		250		ns	
t_{WO}	Time From $\overline{WR}$ = 1 To $\overline{OBF}$ = 0		50		ns	
t_{AO}	Time From $\overline{ACK}$ = 0 To $\overline{OBF}$ = 1		200		ns	
t_{SI}	Time From $\overline{STB}$ = 0 To IBF		200		ns	
t_{RI}	Time From $\overline{RD}$ = 1 To IBF = 0		200		ns	



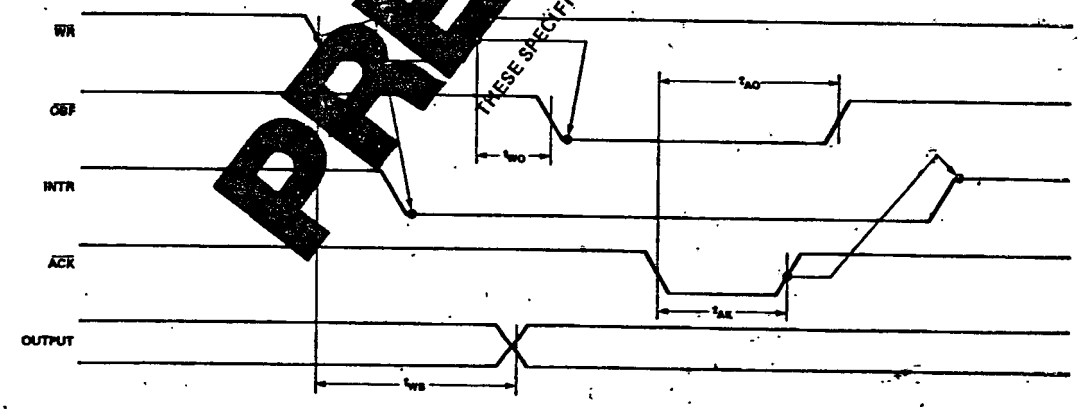
Mode 0 (Basic Input)



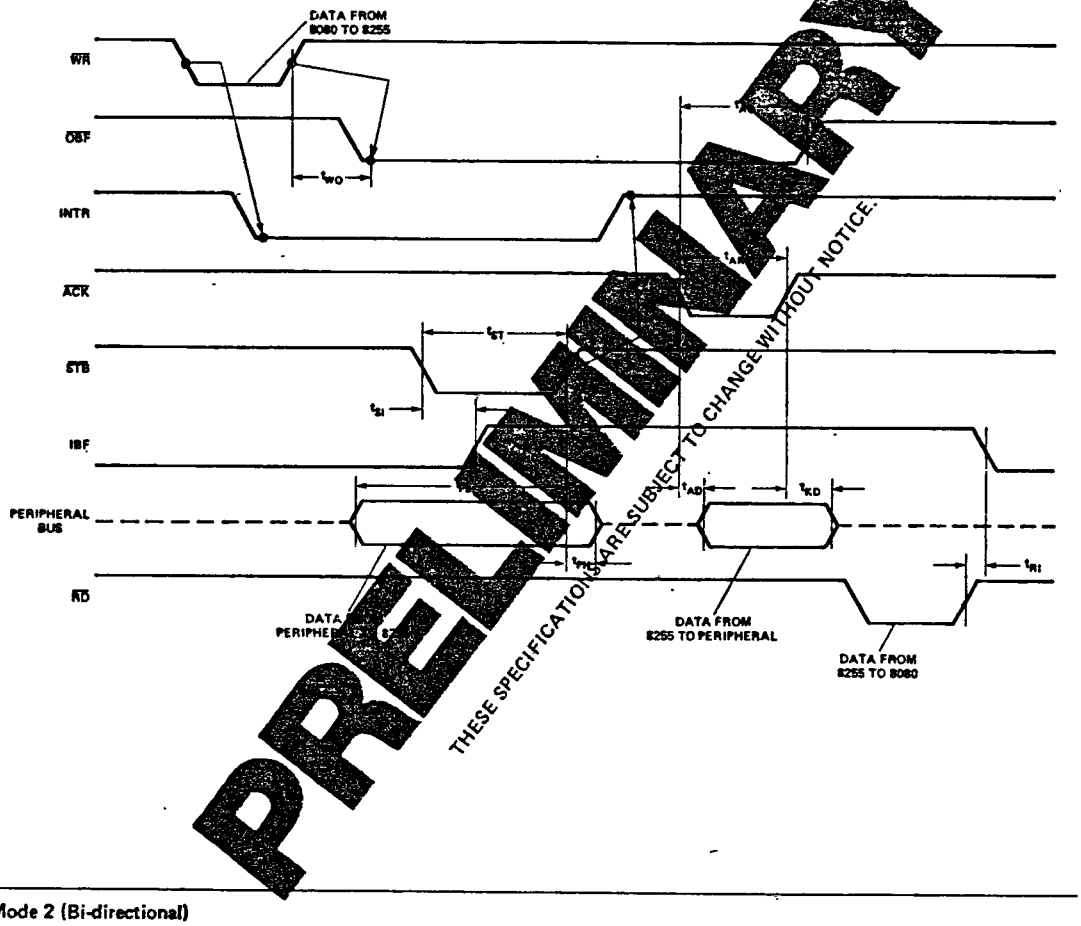
Mode 0 (Basic Output)



Mode 1 (Strobed Input)



Mode 1 (Strobed Output)



Mode 2 (Bi-directional)

National Semiconductor

DS75160A/DS75161A/DS75162A IEEE-488 GPIB Transceivers

General Description

This family of high-speed-Schottky 8-channel bi-directional transceivers is designed to interface TTL/MOS logic to the IEEE Standard 488-1978 General Purpose Interface Bus (GPIB). PNP inputs are used at all driver inputs for minimum loading, and hysteresis is provided at all receiver inputs for added noise margin. The IEEE-488 required bus termination is provided internally with an active turn-off feature which disconnects the termination from the bus when V_{CC} is removed. A power up/down protection circuit is included at all bus outputs to provide glitch-free operation during V_{CC} power up or down.

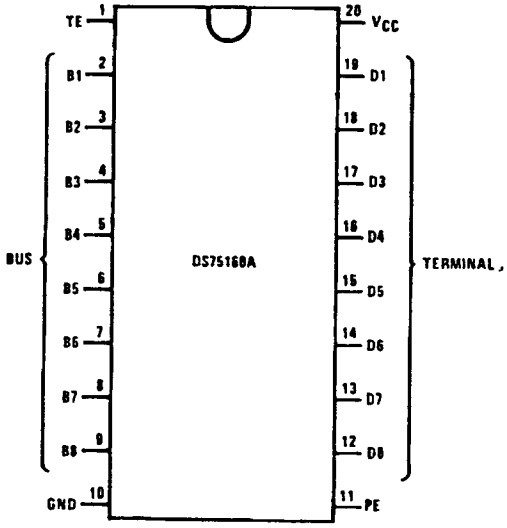
The General Purpose Interface Bus is comprised of 16 signal lines — 8 for data and 8 for interface management. The data lines are always implemented with DS75160A, and the management lines are either implemented with DS75161A in a single-controller system, or with DS75162A in a multi-controller system.

Features

- 8-channel bi-directional non-inverting transceivers
- Bi-directional control implemented with TRI-STATE® output design
- Meets IEEE Standard 488-1978
- High-speed Schottky design
- Low power consumption
- High impedance PNP inputs (drivers)
- 500 mV (typ) input hysteresis (receivers)
- On-chip bus terminators
- No bus loading when V_{CC} is removed
- Power up/down protection (glitch-free)
- Pin selectable open collector mode on DS75160A driver outputs
- Accommodates multi-controller systems

Connection Diagrams

Dual-In-Line Package



TL/F/5804-1

Top View

Order Number DS75160AN
See NS Package Number N20A

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Supply Voltage, V_{CC}	7.0V
Input Voltage	5.5V
Storage Temperature Range	−65°C to +150°C
Lead Temperature (Soldering, 4 sec.)	260°C
Maximum Power Dissipation* at 25°C Molded Package	1897 mW

*Derate molded package 15.2 mW/°C above 25°C.

Operating Conditions

	Min	Max	Units
V_{CC}, Supply Voltage	4.75	5.25	V
T_A, Ambient Temperature	0	70	°C
I_{OL}, Output Low Current			
Bus		48	mA
Terminal		16	mA

Electrical Characteristics (Notes 2 and 3)

Symbol	Parameter		Conditions		Min	Typ	Max	Units
V_{IH}	High-Level Input Voltage				2			V
V_{IL}	Low-Level Input Voltage						0.8	V
V_{IK}	Input Clamp Voltage		I_I = −18 mA			−0.8	−1.5	V
V_{HYS}	Input Hysteresis	Bus			400	500		mV
V_{OH}	High-Level Output Voltage	Terminal	I_{OH} = −800 μA		2.7	3.5		V
		Bus (Note 5)	I_{OH} = −5.2 mA		2.5	3.4		
V_{OL}	Low-Level Output Voltage	Terminal	I_{OL} = 16 mA			0.3	0.5	V
		Bus	I_{OH} = 48 mA			0.4	0.5	
I_{IH}	High-Level Input Current	Terminal and TE, PE, DC, SC Inputs	V_I = 5.5V			0.2	100	μA
			V_I = 2.7V			0.1	20	
I_{IL}	Low-Level Input Current		V_I = 0.5V			−10	−100	μA
V_{BIAS}	Terminator Bias Voltage at Bus Port	Bus	Driver Disabled	$I_{I(bus)}$ = 0 (No Load)	2.5	3.0	3.7	V
I_{LOAD}	Terminator Bus Loading Current		Driver Disabled	$V_{I(bus)}$ = −1.5V to 0.4V	−1.3			mA
				$V_{I(bus)}$ = 0.4V to 2.5V	0		−3.2	
				$V_{I(bus)}$ = 2.5V to 3.7V			2.5 −3.2	
				$V_{I(bus)}$ = 3.7V to 5V	0		2.5	
				$V_{I(bus)}$ = 5V to 5.5V	0.7		2.5	
			V_{CC} = 0V, $V_{I(bus)}$ = 0V to 2.5V				40	μA
I_{OS}	Short-Circuit Output Current	Terminal	V_I = 2V, V_O = 0V (Note 4)		−15	−35	−75	mA
		Bus (Note 5)			−35	−75	−150	
I_{CC}	Supply Current	DS75160A	Transmit, TE = 2V, PE = 2V, V_I = 0.8V			85	125	mA
			Receive, TE = 0.8V, PE = 2V, V_I = 0.8V			70	100	
		DS75161A	TE = 0.8V, DC = 0.8V, V_I = 0.8V			84	125	
		DS75162A	TE = 0.8V, DC = 0.8V, SC = 2V, V_I = 0.8V			85	125	
C_{IN}	Bus-Port Capacitance	Bus	V_{CC} = 5V or 0V, V_I = 0V to 2V, f = 1 MHz			20	30	pF

Note 1: "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. They are not meant to imply that the device should be operated at these limits. The table of "Electrical Characteristics" provides conditions for actual device operation.

Note 2: Unless otherwise specified, min/max limits apply across the 0°C to +70°C temperature range and the 4.75V to 5.25V power supply range. All typical values are for T_A = 25°C and V_{CC} = 5.0V.

Note 3: All currents into device pins are shown as positive; all currents out of device pins are shown as negative; all voltages are referenced to ground, unless otherwise specified. All values shown as max or min are so classified on absolute value basis.

Note 4: Only one output at a time should be shorted.

Note 5: This characteristic does not apply to outputs on DS75161A and DS75162A that are open collector.

Switching Characteristics V_{CC} = 5.0V ±5%, T_A = 0°C to 70°C (Note 1)

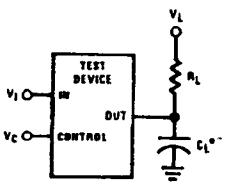
Symbol	Parameter	From	To	Conditions	DS75160A			DS75161A			DS75162A			Units
					Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
t_{PLH}	Propagation Delay Time, Low to High Level Output	Terminal	Bus	V_L = 2.3V, R_L = 38.3Ω, C_L = 30 pF *Figure 1*		10	20		10	20		10	20	ns
t_{PHL}	Propagation Delay Time, High to Low Level Output					14	20		14	20		14	20	ns
t_{PLH}	Propagation Delay Time, Low to High Level Output	Bus	Terminal	V_L = 5.0V, R_L = 240Ω, C_L = 30 pF *Figure 2*		14	20		14	20		14	20	ns
t_{PHL}	Propagation Delay Time, High to Low Level Output					10	20		10	20		10	20	ns
t_{PZH}	Output Enable Time to High Level	TE, DC, or SC (Note 2) (Note 3)	Bus	V_I = 3.0V, V_L = 0V, R_L = 480Ω, C_L = 15 pF *Figure 1*		19	32		23	40		23	40	ns
t_{PHZ}	Output Disable Time From High Level					15	22		15	25		15	25	ns
t_{PZL}	Output Enable Time to Low Level			V_I = 0V, V_L = 2.3V, R_L = 38.3Ω, C_L = 15 pF *Figure 1*		24	35		28	48		28	48	ns
t_{PLZ}	Output Disable Time From Low Level					17	25		17	27		17	27	ns
t_{PZH}	Output Enable Time to High Level	TE, DC, or SC (Note 2) (Note 3)	Terminal	V_I = 3.0V, V_L = 0V, R_L = 3 kΩ, C_L = 15 pF *Figure 1*		17	33		18	40		18	40	ns
t_{PHZ}	Output Disable Time From High Level					15	25		22	33		22	33	ns
t_{PZL}	Output Enable Time to Low Level			V_I = 0V, V_L = 5V, R_L = 280Ω, C_L = 15 pF *Figure 1*		25	39		28	52		28	52	ns
t_{PLZ}	Output Disable Time From Low Level					15	27		20	35		20	35	ns
t_{PZH}	Output Pull-Up Enable Time (DS75160A Only)	PE (Note 2)	Bus	V_I = 3V, V_L = 0V, R_L = 480Ω, C_L = 15 pF *Figure 1*		10	17		NA			NA		ns
t_{PHZ}	Output Pull-UP Disable Time (DS75160A Only)					10	15		NA			NA		ns

Note 1: Typical values are for V_{CC} = 5.0V and T_A = 25°C and are meant for reference only.

Note 2: Refer to Functional Truth Tables for control input definition.

Note 3: Test configuration should be connected to only one transceiver at a time due to the high current stress caused by the V_I voltage source when the output connected to that input becomes active.

Switching Load Configurations

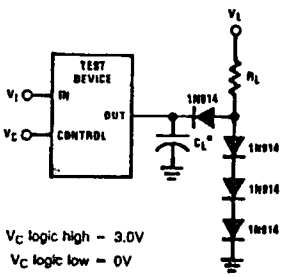


TL/F/5804–8

V_C logic high = 3.0V
V_C logic low = 0V

*C_L includes jig and probe capacitance

FIGURE 1



V_C logic high = 3.0V
V_C logic low = 0V

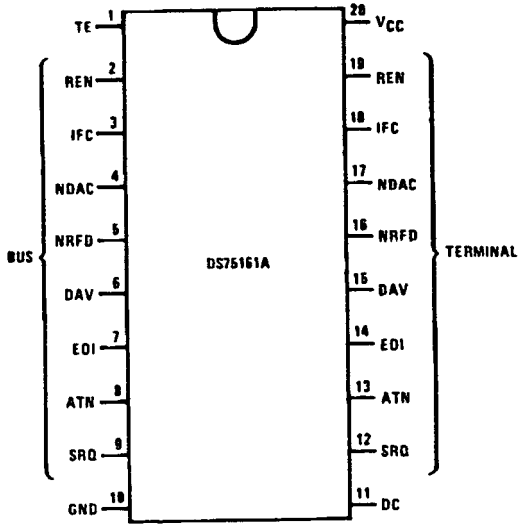
TL/F/5804–9

*C_L includes jig and probe capacitance

FIGURE 2

Connection Diagrams (Continued)

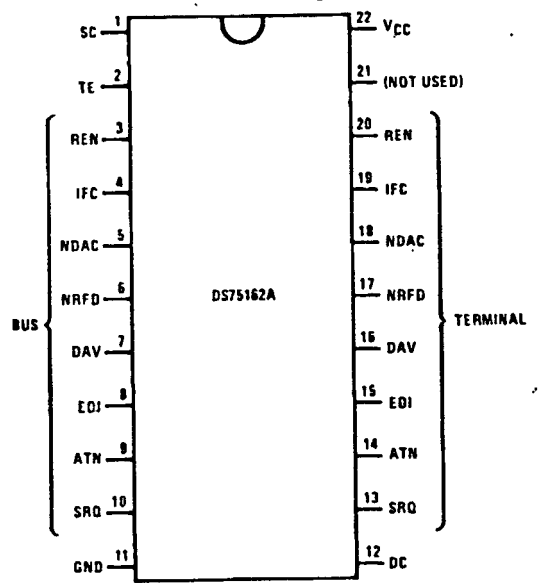
Dual-In-Line Package



TL/F/5804-2

Top View

Dual-In-Line Package



TL/F/5804-3

Top View

Order Number DS75161AN or DS75162AN
See NS Package Number N20A or N22A

Functional Description

DS75160A

This device is an 8-channel bi-directional transceiver with one common direction control input, denoted TE. When used to implement the IEEE-488 bus, this device is connected to the eight data bus lines, designated DIO_1–DIO_8. The port connections to the bus lines have internal terminators, in accordance with the IEEE-488 Standard, that are deactivated when the device is powered down. This feature guarantees no bus loading when V_{CC} = 0V. The bus port outputs also have a control mode that either enables or disables the active upper stage of the totem-pole configuration. When this control input, denoted PE, is in the high state, the bus outputs operate in the high-speed totem-pole mode. When PE is in the low state, the bus outputs operate as open collector outputs which are necessary for parallel polling.

DS75161A

This device is also an 8-channel bi-directional transceiver which is specifically configured to implement the eight management signal lines of the IEEE-488 bus. This device, paired with the DS75160A, forms the complete 16-line interface between the IEEE-488 bus and a single controller instrumentation system. In compliance with the system organization of the management signal lines, the SRQ, NDAC, and NRFD bus port outputs are open collector. In contrast to the DS75160A, these open collector outputs are a fixed configuration. The direction control is divided into three groups. The DAV, NDAC, and NRFD transceiver directions are controlled by the TE input. The ATN, SRQ, REN, and IFC transceiver directions are controlled by the DC input. The EOI transceiver direction is a function of both the TE and DC inputs, as well as the logic level present on the ATN channel. The port connections to the bus lines have internal terminators identical to the DS75160A.

DS75162A

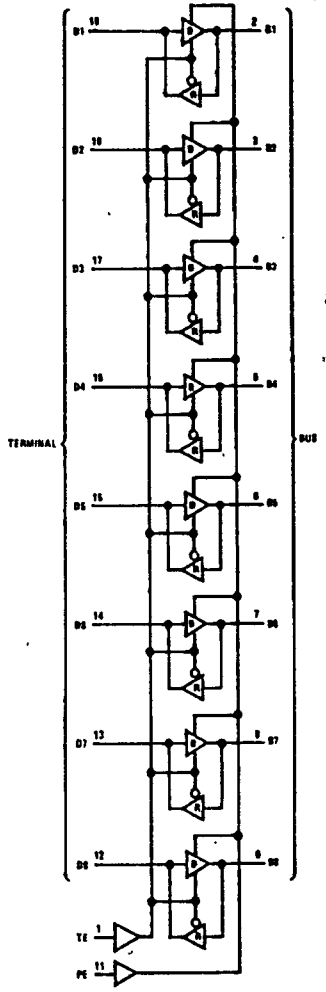
This device is identical to the DS75161A, except that an additional direction control input is provided, denoted SC. The SC input controls the direction of the REN and IFC transceivers that are normally controlled by the DC input on the DS75161A. This additional control function is instrumental in implementing multiple controller systems.

Table of Signal Line Abbreviations

Signal Line Classification	Mnemonic	Definition	Device
Control Signals	DC	Direction Control	DS75161A/ DS75162A
	PE	Pull-Up Enable	DS75160A
	TE	Talk Enable	All
	SC	System Controller	DS75162A
Data I/O Ports	B1–B8	Bus Side of Device	DS75160A
	D1–D8	Terminal Side of Device	
Management Signals	ATN	Attention	DS75161A/ DS75162A
	DAV	Data Valid	
	EOI	End or Identify	
	IFC	Interface Clear	
	NDAC	Not Data Accepted	
	NRFD	Not Ready for Data	
	REN	Remote Enable	
	SRQ	Service Request	

Logic Diagrams

DS75160A



Note 1: Denotes driver

Note 2: Denotes receiver

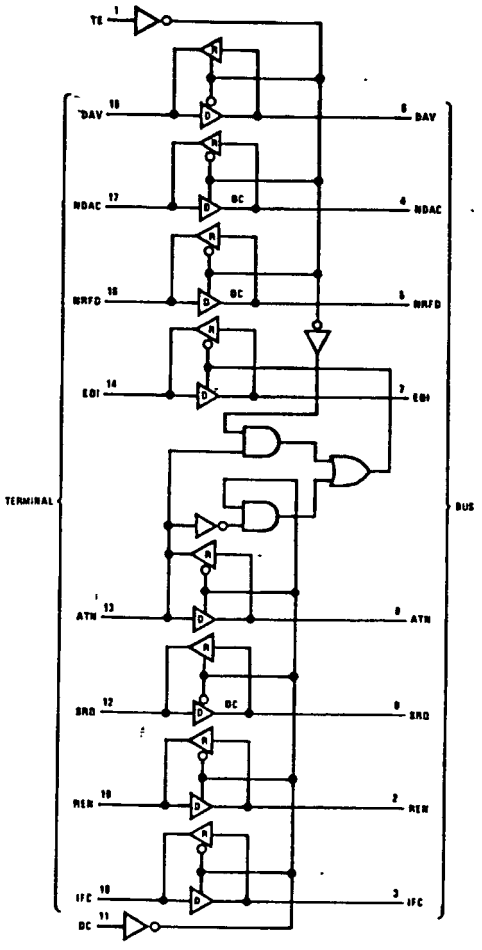
Note 3: Driver and receiver outputs are totem-pole configurations

Note 4: The driver outputs of DS75160A can have their active pull-ups disabled by switching the PE input (pin 11) to the logic low state. This mode configures the outputs as open collector.

TL/F/5

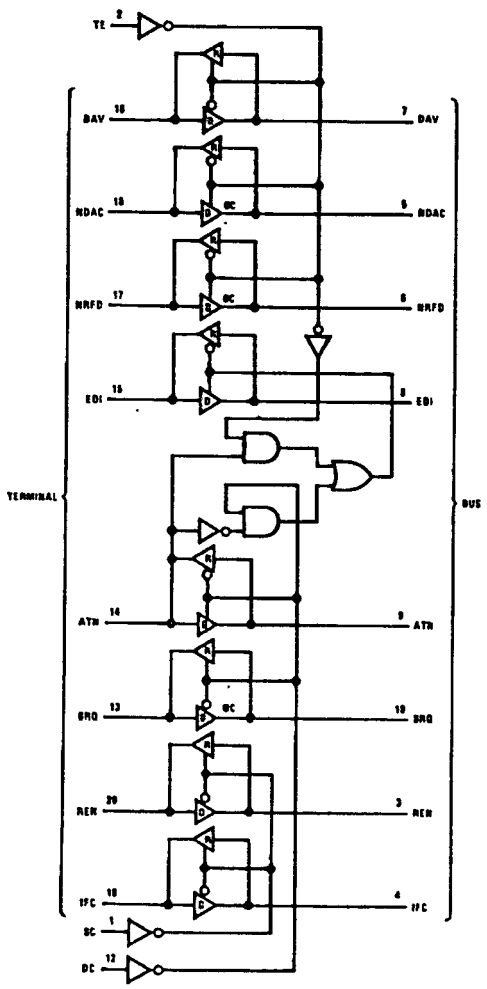
Logic Diagrams (Continued)

DS75161A



TL/F/5804–5

DS75162A



TL/F/5804–6

Note 1: Denotes driver

Note 2: Denotes receiver

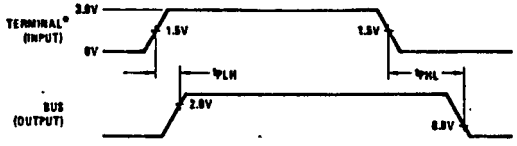
Note 3: Symbol "OC" specifies open collector output

Note 4: Driver and receiver outputs that are not specified "OC" are totem-pole configurations

TL/F/5804–7

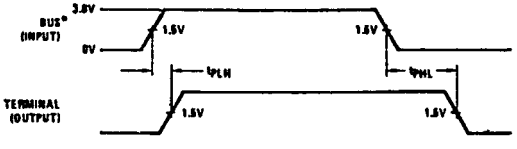
Switching Waveforms

Transmit Propagation Delays



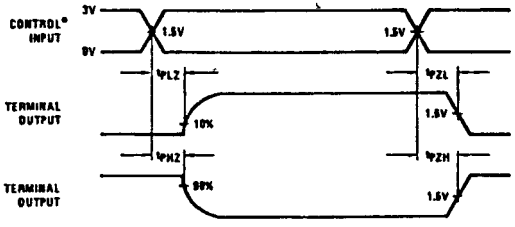
TL/F/5804–10

Receive Propagation Delays



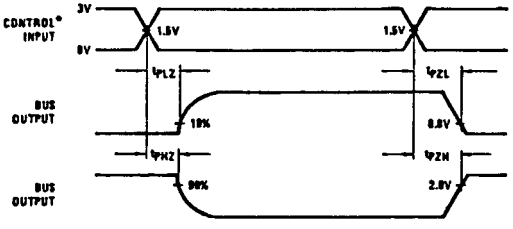
TL/F/5804–11

Terminal Enable/Disable Times



TL/F/5804–12

Bus Enable/Disable Times

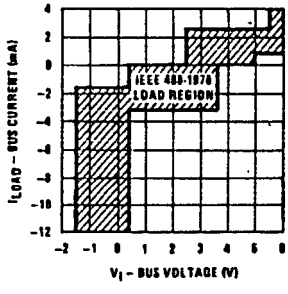


TL/F/5804–13

*Input signal: f = 1.0 MHz, 50% duty cycle, $t_r = t_f \leq 5$ ns

Performance Characteristics

Bus Port Load Characteristics



TL/F/5804–14

Refer to Electrical Characteristics table

Functional Truth Tables

DS75160A

Control Input Level		Data Transceivers	
TE	PE	Direction	Bus Port Configuration
H	H	T	Totem-Pole Output
H	L	T	Open Collector Output
L	X	R	Input

DS75161A

Control Input Level			Transceiver Signal Direction							
TE	DC	ATN*		EOI	REN	IFC	SRQ	NRFD	NDAC	DAV
H	H		R		R	R	T	R	R	T
H	L		T		T	T	R	R	R	T
L	H		R		R	R	T	T	T	R
L	L		T		T	T	R	T	T	R
H	X	H		T						
L	X	H		R						
X	H	L		R						
X	L	L		T						

DS75162A

Control Input Level				Transceiver Signal Direction							
SC	TE	DC	ATN*	EOI	REN	IFC	SRQ	NRFD	NDAC	DAV	
H	H	H		R		T	T	T	R	R	T
H	H	L		T		T	T	R	R	R	T
H	L	H		R		T	T	T	T	T	R
H	L	L		T		T	T	R	T	T	R
L	H	H		R		R	R	T	R	R	T
L	H	L		T		R	R	R	R	R	T
L	L	H		R		R	R	T	T	T	R
L	L	L		T		R	R	R	T	T	R
X	H	X	H		T						
X	L	X	H		R						
X	X	H	L		R						
X	X	L	L		T						

H = High level input

L = Low level input

X = Don't care

T = Transmit, i.e., signal outputted to bus

R = Receive, i.e., signal outputted to terminal

*The ATN signal level is sensed for internal multiplex control of EOI transmission direction logic.