ระบบควบคุมไฟฟ้าในห้องพัก

(DISTRIBUTE CONTROL SYSTEM)

โดย

1. นาย ธวัชชัย รัตนอำนวยศิริ 34.131111
2. นาย สมศักดิ์ แซ่ตั้ง 34.131135
3. นาย รุสภณ ดุรุรัตน์ 34.131138
4. นาย อนุชิต จิตต์ประเสริฐ 34.131139

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตร์บัณฑิต

สาขาเทคโนโลยีอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2535

ระบบควบคุมไฟฟ้าในห้องพัก

(DISTRIBUTE CONTROL SYSTEM)

โดย

1. นาย ธวัชชัย	รัตน์อำนวยศิริ	34.131111
2. นาย สมศักดิ์	แซ่ตั้ง	34.131135
3. นาย โสภณ	คุรุรัตน์	34.131138
4. นาย อนุชิต	จิตต์ประเสริฐ	34.131139

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2535

ระบบควบคุมไฟฟ้าในห้องพัก

(DISTRIBUTE CONTROL SYSTEM)

ได้รับอนุมัติให้เป็นส่วนหนึ่งของ การศึกษาตามหลักสูตรอุตสาหกรรมศาสตร์บัณฑิต

ปีการศึกษา 2535

คณะวิศวกรรมศาสตร์ ภาควิชาเทคนิคอุตสาหกรรม สาขาวิชาเทคโนโลยีอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง

คณะกรรมการตรวจสอบปริญญานิพนธ์

..... ประธานกรรมการ

(..นายอรรถสิทธิ์ หล่าสกุล.....)

..... กรรมการ

(..นายอุทัย กรีธีระวิโรจน์ .....)

..... กรรมการ

(..นายไพศาล สิทธิโยภาสกุล.....)

..... กรรมการ

(.....)

โครงงาน

ชื่อปริญญานิพนธ์ : ระบบควบคุมไฟฟ้าในห้องพัก

ผู้จัดทำ

1. นาย ธวัชชัย	รัตนอำนวยศิริ	34.131111
2. นาย สมศักดิ์	แซ่ตั้ง	34.131135
3. นาย โสภณ	คุรุรัตน์	34.131138
4. นาย อนุชิต	จิตต์ประเสริฐ	34.131139

อาจารย์ที่ปรึกษา

อาจารย์ ไพศาล สิทธิรัยภาสกุล

คณะวิศวกรรมศาสตร์

ภาควิชาเทคนิคอุตสาหกรรม

สาขาเทคโนโลยีอิเล็กทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง

PROJECT

DISTRIBUTE CONTROL SYSTEM

1. Tawatchai Ratanaumnuysiri

2. Somsak Tang

3. Sophon Kururat

4. Anuchit Jitprasert

Faculty of Engineering

Department of Industrial Technic

Major in Electronic Technology

King Mongkut's Institute of Technology Ladkrabang

วัตถุประสงค์

1. เพื่อศึกษาทฤษฎีการทำงานและการประยุกต์ใช้งานของ Z80180
2. เพื่อศึกษาวิธีการออกแบบวงจรสนับสนุนต่างๆให้ CPU ทำการควบคุมได้ตามต้องการ
3. เพื่อศึกษาวิธีการออกแบบโปรแกรมควบคุมการทำงานของ Z80180 โดยใช้ภาษา Assembly
4. เพื่อศึกษาการติดต่อสื่อสารข้อมูลระหว่าง Microcomputer กับ Z80180 โดยผ่าน Port RS 232
5. เพื่อเป็นแนวทางในการนำเทคโนโลยีทางด้านอิเล็กทรอนิกส์มาประยุกต์ช่วยงานทางด้านธุรกิจ

สารบัญ

เครื่องควบคุมอุปกรณ์ไฟฟ้าในอาคาร

นายธวัชชัย	รัตนอำนวยศิริ	34.131111
นายสมศักดิ์	แซ่ตั้ง	34.131135
นายโสภณ	ดุรุรัตน์	34.131138
นายอนุชิต	จิตต์ประเสริฐ	34.131139

อาจารย์ที่ปรึกษา

อาจารย์ไพศาล สิทธิริยภาสกุล

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้กล่าวถึงการออกแบบเครื่องควบคุมอปกรณ์ไฟฟ้าภายในอาคาร ซึ่งสามารถควบคุมอุปกรณ์ไฟฟ้าได้ 512 ห้อง โดยแต่ละห้องจะมีอุปกรณ์ไฟฟ้าได้ 8 ชิ้น และมีตัวตรวจจับพร้อมทั้งอุปกรณ์ฉุกเฉิน 4 ชุดต่อ 1 ห้อง การควบคุมทั้งหมดสามารถสั่งงานได้ทาง PC และแสดงสถานะของอุปกรณ์ทุกตัวได้ทาง MONITOR ของ PC ได้เช่นกัน โดยใช้ Z80180 CONTROL BOARD เป็นตัวควบคุมอุปกรณ์ต้นทางและปลายทาง และยังทำหน้าที่ INTERFACE ระหว่าง PC กับอุปกรณ์ต้นทางปลายทาง Z80180 CONTROL BOARD นี้สามารถทำงานได้อย่างอิสระทำให้เราสามารถปิดเครื่อง PC หรือนำ PC ไปใช้งานอย่างอื่นได้โดยไม่กระทบกระเทือนต่อการทำงานของวงจร

DISTRIBUTE CONTROL SYSTEM

Mr. Tawatchai Ratanaumnuysiri

Mr. Somsak Tang

Mr. Sophon Kururat

Mr. Anuchit Jitprasert

Advisor:

Phaisarn Suthiyophakul

Abstrac

THIS THESIS IS THE MENTION OF THE CONTROL FOR THE ELECTRICAL APPLIANCE INTHE BUILDING, WHICH CAN BE ABLE TO CONTROL THE EQUIPMENT UP TO 512 ROOMS, BY THE EACH ROOME CAN BE CONTROL FOR 8 UNITS FOR ONE ROOM, AND CAN PROVIDE THE UNIT SENSOR AND EMERGENCY CONTROL FOR 4 UNITS PER ONE ROOM. FOR THE ALL CONTROL UNITS CAN BE PROCEED AND SHOW THE ALL FUNCTION ON THE PC. IT USED THE Z80180 CONTROL BOARD TO BE CONTROL OF THE ORDER OF THE START AND END EQUIPMENT.

กิตติกรรมประกาศ

ในการสร้างโครงงานปริญญานิพนธ์นี้ ทางผู้จัดทำได้รับความอนุเคราะห์ช่วยเหลือให้คำปรึกษา แนะนำแนวทางในการดำเนินงาน ตลอดจนความช่วยเหลือในเรื่องเครื่องมือที่ใช้ในการวิจัยต่างๆ จากท่านอาจารย์ไพศาล สิทธิรัยภาสกุล จนกระทั่งสำเร็จเป็นโครงงานนี้

ทีมงานผู้จัดทำโครงงานขอกราบขอบพระคุณท่านอาจารย์ ที่ให้ความกรุณามา ณ ที่นี้

คณะผู้จัดทำโครงงาน

บทที่ 1

บทนำ

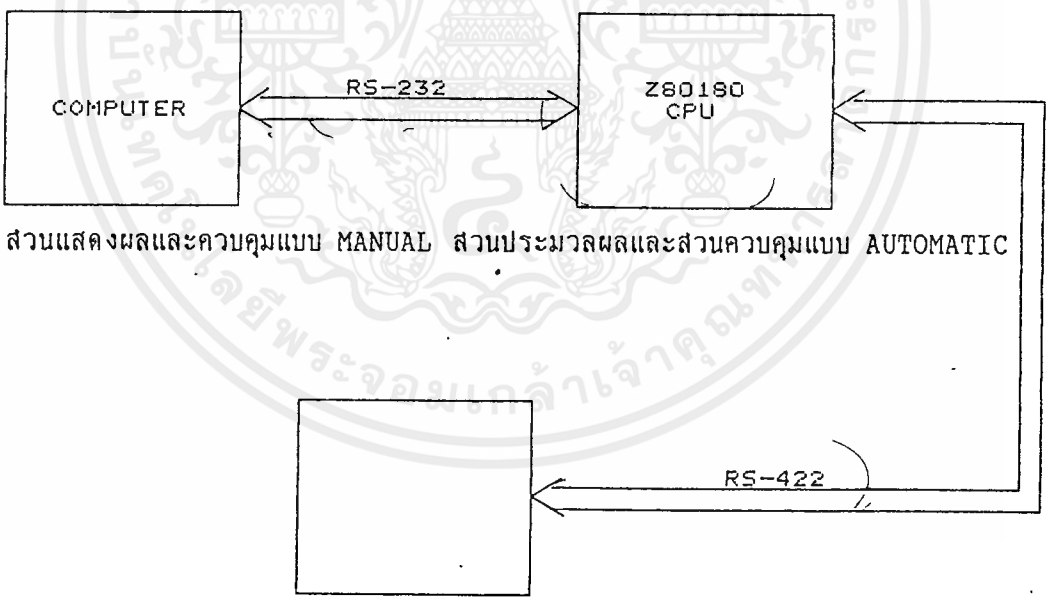
ระบบควบคุมไฟฟ้าในห้องพัก

ระบบนี้จะทำหน้าที่ควบคุมอุปกรณ์ไฟฟ้าในห้องพัก และรับสัญญาณจากอุปกรณ์ตรวจจับ เพื่อนำไปประมวลผลในการทำงานและส่งสัญญาณไปควบคุมอุปกรณ์ไฟฟ้าในห้องนั้น อีกที สำหรับผลการทำงานการจัดการเกี่ยวกับห้องนั้น จะนำมาแสดงผลออกทางจอ computer เพื่อให้ผู้ใช้ได้รับรู้สถานะต่างๆ ของอุปกรณ์ไฟฟ้าในห้องพักอีกด้วย โดยระบบนี้สามารถควบคุมอุปกรณ์ไฟฟ้าในห้องพักในแต่ละห้องได้ 8 ชิ้น รับสัญญาณตรวจจับได้ 4 อินพุท และควบคุมสวิทช์ฉุกเฉินได้ 4 สวิทช์ สามารถควบคุมห้องพักได้สูงสุด 512 ห้อง แบ่งเป็น 16 ชั้นๆ ละ 32 ห้องพัก ระบบควบคุมไฟฟ้าในห้องพักนี้จะประกอบด้วยส่วนต่างๆ 3 ส่วนทำงานร่วมกันคือ

1.ส่วนแสดงผลและควบคุมแบบ manual

2.ส่วนประมวลผลและควบคุมแบบ automatic

3.ส่วนควบคุมอุปกรณ์ปลายทาง



ส่วนแสดงผลและควบคุมแบบ MANUAL ส่วนประมวลผลและส่วนควบคุมแบบ AUTOMATIC

ส่วนควบคุมปลายทาง

ระบบควบคุมไฟฟ้าในห้องพัก

1.ส่วนแสดงผลและควบคุมแบบ manual ส่วนนี้เราใช้ computer หนึ่งชุดซึ่งจะทำหน้าที่แสดงผลออกทางจอภาพและสามารถสั่งงาน หรือทำการควบคุมอุปกรณ์ปลายทางได้ทางด้าน keyboard การทำงานนี้จะต้องมีตัวโปรแกรมของระบบเข้ามาทำงานก่อน ส่วนอุปกรณ์ทางด้านฮาร์ดแวร์ในส่วนนี้ไม่มีอะไรต้องเพิ่มเติม ส่วนนี้จะต้องติดต่อกับส่วนที่สอง คือส่วนประมวลผลและควบคุมแบบ automatic โดยผ่านทางพอร์ต RS 232 เพื่อนำข้อมูลเข้ามาแสดงผล และเพื่อส่งข้อมูลออกไปเมื่อมีการควบคุมแบบ manual

2.ส่วนประมวลผลและควบคุมแบบ automatic ส่วนนี้ประกอบด้วย CPU , I/O PORT,DECODER,ENCODER,RS-422 LINE DRIVER/RECEIVER ประกอบกันขึ้นมาส่วนนี้ทำหน้าที่ประมวลผลข้อมูล และทำหน้าที่ติดต่อกับทั้งส่วนที่หนึ่งและส่วนที่สอง คือส่วนแสดงผลและควบคุมแบบ manual และ ส่วนควบคุมอุปกรณ์ปลายทาง โดยติดต่อกับส่วนควบคุมปลายทางเพื่อส่งสัญญาณไปควบคุม และนำข้อมูลจากทางด้านส่วนควบคุมอุปกรณ์ปลายทางมาประมวลผลแล้วเก็บไว้เพื่อส่งให้ส่วนแสดงผลอีกที หรือจะส่งกลับไปควบคุมอุปกรณ์ปลายทางอีกทีแล้วแต่การประมวลผลของ CPU และโปรแกรมที่สั่งให้ CPU ทำงาน สำหรับการติดต่อกับส่วนควบคุมปลายทางจะติดต่อออกทาง RS-422 เพื่อให้สามารถส่งได้ไกล ๆ สำหรับการติดต่อกับส่วนแสดงผลและควบคุมแบบ manual นั้นจะติดต่อเพื่อส่งข้อมูลออกไปแสดงผลที่จอภาพหรือนำคำสั่งการควบคุมแบบ manual มาประมวลผลเพื่อควบคุมต่อไป

3.ส่วนควบคุมอุปกรณ์ปลายทาง จะทำหน้าที่รับส่งข้อมูลกับส่วนประมวลผลและควบคุมแบบ automaticเพื่อนำข้อมูลมาควบคุมอุปกรณ์ไฟฟ้า หรือนำข้อมูลสถานะต่าง ๆ ของอุปกรณ์ไฟฟ้าและตัวตรวจจับส่งไปให้ประมวลผล ส่วนนี้ประกอบด้วย DECODER, T FLIP-FLOP,RS-422 LINE DRIVER/RECEIVER

รายละเอียดทางด้านฮาร์ดแวร์

1.ส่วนแสดงผลและควบคุมแบบ manual ใช้ computer อย่างเดียวไม่มีฮาร์ดแวร์อย่างอื่นเพิ่มเติมจึงไม่ขอกล่าวถึง

2.ส่วนประมวลผลและควบคุมแบบ automatic นี้ประกอบด้วย CPU ,I/O PORT,DECODER,ENCODER,RS-422 LINE DRIVER/RECEIVER โดยที่ CPU,I/O PORT จะรวมอยู่ในส่วนเดียวกันเป็น Z180 CONTROL PACK โดยตัว CPU เราใช้ไอซีเบอร์ Z80180 และตัว I/O PORT ใช้ไอซีเบอร์ 8255 ส่วนทางด้าน DECODER,ENCODER, RS-422 LINE DRIVER/RECEIVER จะอยู่อีกคนละส่วนแยกจากกัน โดยส่วน DECODER จะอยู่บนบอร์ดอีกบอร์ดหนึ่งและส่วน ENCODER, RS-422 LINE DRIVER/RECEIVER จะอยู่อีกบอร์ดหนึ่งซึ่งมีวงจรทางด้าน DECODER ขนาดเล็กประกอบอยู่ด้วยอีกรวมแล้วในส่วนที่สองนี้ประกอบด้วยบอร์ดถึง 3 บอร์ดและในส่วนนี้ยังต้องมีส่วนโปรแกรมที่ใช้สำหรับตัว CPU ที่อยู่ในส่วน Z180 CONTROL PACK เพื่อให้ทำงานประสานระหว่างบอร์ดต่าง ๆ และส่วนประกอบต่าง ๆ เป็นไปตามต้องการ

3.ส่วนควบคุมอุปกรณ์ปลายทาง ส่วนนี้มีบอร์ดเพียงหนึ่งบอร์ดซึ่งประกอบด้วยวงจร DECODER, ENCODER, RS-422 LINE DRIVER/RECEIVER,T FLIP-FLOP ทั้งหมดประกอบกันเพื่อทำหน้าที่รับข้อมูลมาใช้ทำเป็นสัญญาณควบคุมอุปกรณ์ไฟฟ้าต่างๆ และรับสัญญาณจากตัวตรวจจับมาทำการ ENCODER แล้วส่งไปให้ส่วนประมวลผลและควบคุมแบบ automatic เพื่อใช้ในการประมวลผลต่อไป

บทที่ 2

ขั้นตอนการออกแบบและสร้าง

ขั้นตอนการออกแบบและการสร้าง

เนื่องจากส่วนฮาร์ดแวร์มีอยู่ 2 ส่วนเราจึงแยกกล่าวทีละส่วนดังนี้

1.ส่วนประมวลผลและควบคุมแบบ automatic ส่วนนี้มีทั้งหมด 3 บอร์ด คือ บอร์ด CONTROLPACK บอร์ด DECODER บอร์ด ENCODER,RS-422 DRIVER/ RECEIVER ในที่นี้ บอร์ด CONTROL PACK ไม่ต้องออกแบบเนื่องจากเป็นบอร์ดที่จัดเป็นชุดสำเร็จไว้แล้วสำหรับนำมาใช้พัฒนา ดังนั้นในส่วนนี้เราต้องออกแบบและสร้างบอร์ด 2 บอร์ดเท่านั้น ซึ่งทั้ง 2 บอร์ดจะทำงานร่วมกันและคล้าย ๆกันคือจะนำข้อมูลที่ได้จาก I/O PORT มาแบ่งให้ส่วนหนึ่งเป็นข้อมูลสำหรับการควบคุมอุปกรณ์ไฟฟ้าภายในห้องพัก อีกส่วนหนึ่งเป็นข้อมูลในการเลือกห้องพักเลือกชั้น ซึ่งจะกล่าวถึงดังนี้

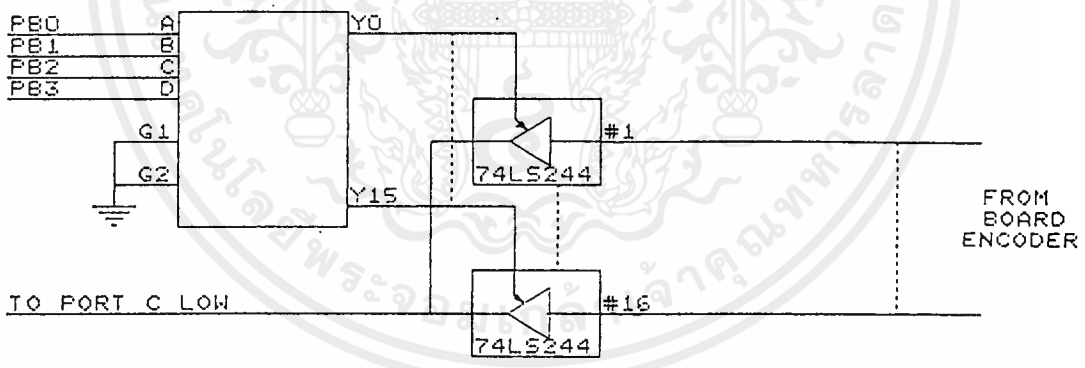
1.1 บอร์ด DECODER ส่วนนี้จะนำข้อมูลจากทางด้าน I/O PORT ของ CONTROL PACK มาทำการ DECODER ให้ได้เป็นสัญญาณเลือกชั้นในการจะส่งข้อมูลและเลือกชั้นที่จะรับข้อมูล ที่ต้องแยกวงจรกันนั้นเนื่องจากการส่งและการรับข้อมูลจะทำคนละช่วงเวลากันและใช้เวลาไม่เท่ากัน ดังนั้นจึงต้องแยกวงจรกัน จากการที่ระบบสามารถควบคุมได้ 16 ชั้น ดังนั้นบอร์ดนี้จึงต้อง DECODER สัญญาณออกมาทั้งหมด 16 สัญญาณ ซึ่งสามารถใช้แค่ข้อมูลจาก I/O PORT เพียง 4 BIT ก็พอ แต่เนื่องจากการที่แยกวงจรออกเป็นสองส่วนคือส่วนเลือกชั้นในการรับข้อมูลกับส่วนเลือกชั้นในการส่งข้อมูล ดังนั้นต้องเพิ่มจำนวนข้อมูลที่จะใช้ในการ DECODER อีกเพื่อให้เกิดการรัดกุมยิ่งขึ้นจึงใช้ข้อมูลจำนวน 8 BIT หรือ 1 PORT นั้นเองจากทั้งหมดที่ไอซีเบอร์ 8255 มีให้ใช้ 3 PORT คือ PORT A,PORT B,PORT C ในที่นี้ใช้ PORT B ส่วนที่เหลือจะใช้ในบอร์ดต่อไป

1.2 บอร์ด ENCODER,RS-422 DRIVER/RECEIVER ส่วนนี้จะนำข้อมูลทาง PORT มาทำเป็นข้อมูลในการควบคุมอุปกรณ์ไฟฟ้าและสวิทช์รวมกันทั้งหมด 12 สัญญาณ จึงใช้ข้อมูลแค่ 4 BIT เท่านั้นโดยข้อมูล 4 BIT นี้จะถูก DECODER ออกเป็นสัญญาณ 16 สัญญาณที่ส่วนควบคุมปลายทาง แต่จะถูกแยกออกไว้ต่างหากเพื่อให้รู้ว่าเป็นสัญญาณควบคุมอุปกรณ์ไฟฟ้าตั้งแต่ที่บอร์ดนี้แล้ว จะเห็นว่าที่กล่าวมาข้างต้นเราจัดแบ่งข้อมูลจาก PORT มาใช้ในการเลือกชั้น และเป็นข้อมูลสำหรับการควบคุมแล้วแต่ยังไม่มีสัญญาณเลือกห้องซึ่งอยู่ในแต่ละชั้นซึ่งจากข้อกำหนดในแต่ละชั้นมีจำนวนห้องสูงสุดได้ 32 ห้องจึงต้องใช้ข้อมูลจาก PORT มาทำการ DECODER จำนวน 5 BIT จากนั้นยังจะต้องมี PORT ไว้สำหรับรับข้อมูลที่ส่งมาจากส่วนควบคุมปลายทางที่จะส่งข้อมูลของสถานะอุปกรณ์ไฟฟ้า และตัวตรวจจับมาให้

อีกรวมทั้งหมด 16 สัญญาณ แต่สัญญาณทั้ง 16 นี้เป็นข้อมูลทั้งหมดทุกสัญญาณไม่ใช่การ DECODER ขึ้นมาจากข้อมูลจำนวน 4 BIT ดังนั้นเราจะทำการ ENCODER ไม่ได้ต้องส่งมาตรง ๆ ทั้ง 16 สัญญาณ ซึ่งจะเสีย PORT ที่จะมารับข้อมูลถึง 2 PORT เต็ม ๆ เราจึงแก้ไขเป็นการใช้ PORT ข้อมูลเพียง 4 BIT มารับข้อมูลส่วนนี้แต่ให้ทำการรับ 4 ครั้ง ๆ ละ 4 สัญญาณ รวมเป็น 16 สัญญาณ จะเห็นว่า เราต้องแบ่ง PORT ข้อมูลมาใช้อีก 3 ส่วนแต่เราพบว่า ไอซี 8255 มี 3 PORT ถูกใช้ไปแล้วหนึ่ง ยังเหลืออีกสองคือ PORT B, PORT C ต้องมาทำเป็น 3 ส่วนคือ ส่วนที่ทำหน้าที่เป็น เอาท์พุท 2 ส่วน ส่วนที่ทำหน้าที่เป็นอินพุท 1 ส่วน เราจึงแยก PORT C ออกเป็นสองส่วนเพราะ PORT นี้สามารถแยกได้ โดยให้ส่วนหนึ่งเป็นส่วนเอาท์พุท 4 BIT ในที่นี้ใช้ POTR C บน ส่วนอีก 4 BIT ที่เหลือใช้เป็นส่วนอินพุทคือ PORT C ล่าง จากนั้นจะเหลืออีก 1 PORT คือ PORT A ใช้สำหรับเป็นข้อมูลในการ DECODER ห้องซึ่งจะใช้เพียง 5 BIT เมื่อเราจัดแบ่ง PORT ข้อมูลกันเรียบร้อยแล้วขั้นตอนต่อไปเป็นการออกแบบวงจรกันก็จะกล่าวถึงเป็นบอร์ด ๆ ไปดังนี้

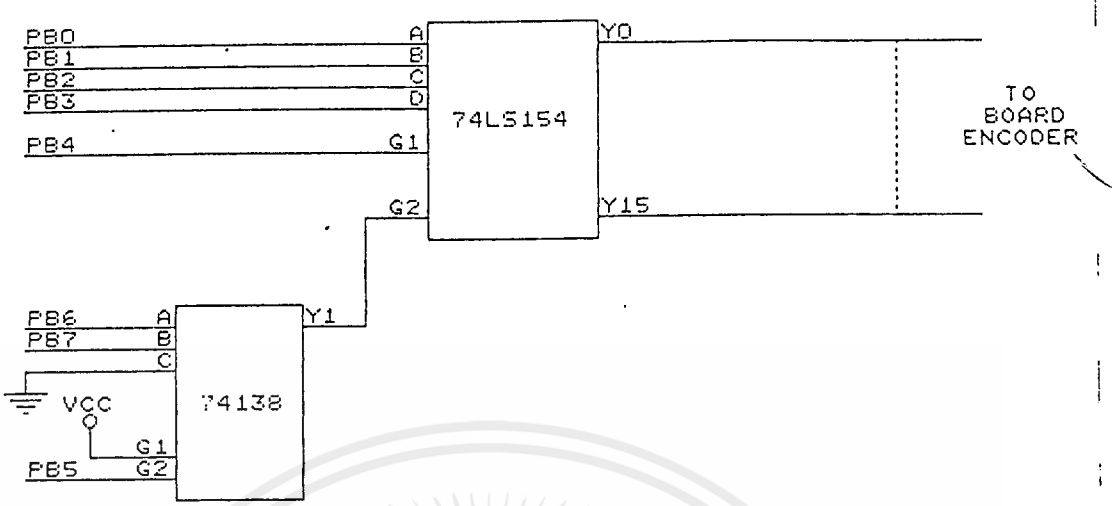
1.1 บอร์ด DECODER ซึ่งทำหน้าที่เลือกชั้นที่จะรับส่งข้อมูล มีทั้งหมด 16 ชั้น ในส่วนรับข้อมูลนั้นเป็นส่วนที่จะถูก DECODER ตลอดเวลาเราจึงใช้ข้อมูลจาก PORT B มา 4 BIT ส่งมาทำการ DECODER ด้วยไอซีเบอร์ 74LS154 ซึ่งเป็น 4 TO 16 DECODER ที่ถูกต่อให้ ENABLE อยู่ตลอดเพื่อให้ข้อมูลของแต่ละชั้นที่ทำการติดต่ออยู่นั้นถูกเลือกอยู่ตลอดเวลาคือพร้อมที่จะถูกอ่านเข้าไปทาง PORT อินพุท คือ PORT C ล่างได้ทุกเมื่อไม่ต้องไปเสียเวลา ENABLE ตัว DECODER อีกที ในที่นี้จะใช้ 4 BIT ล่างของ PORT B คือ PB 0 ถึง PB 3 มาทำการ DECODER ส่วนวงจรที่ทำหน้าที่เลือกชั้นที่จะส่งข้อมูลนั้นก็ใช้ 4 BIT เดียวกันนี้แต่จะมีวงจรเพิ่มเติมเพื่อให้แตกต่างกันบ้างเล็กน้อยเพื่อที่ขณะที่วงจร DECODER ในส่วนที่รับข้อมูลทำงานตลอดเวลานั้นวงจร DECODER ทางด้านส่งข้อมูลจะยังไม่ทำงานจนกว่าจะมีการสั่งให้ทำงานเท่านั้น ทั้งนี้เพราะว่าในส่วนเลือกชั้นที่จะส่งข้อมูลนี้จะทำหน้าที่ในการเลือกชั้นและทำการทริกให้ทำการส่งข้อมูลเลย ดังนั้นวงจรในส่วนนี้จะถูกสั่งให้ทำงานก็ต่อเมื่อมีการเลือกห้อง และกำหนดข้อมูลที่จะทำการติดต่ออะไรกันเรียบร้อยแล้วเท่านั้นจึ้งจะสั่งให้วงจรส่วนนี้ทำงานได้ และในการทริกให้ส่งข้อมูลนี้จะ

ทริกเพียงชั่วเวลาขณะหนึ่งเท่านั้น ดังนั้นจึงต้องมีการสั่งให้หยุดทริกอีกด้วย เมื่อเป็นเช่นนี้การออกแบบวงจรจึงต้องมีส่วนเพิ่มเติมจากวงจรด้านรับคือ จะต้องมีการ DISENABLE ส่วนของวงจร DECODER ด้วย ดังนั้นเราก็ใช้วงจร 4 TO 16 DECODER แบบเดียวกับส่วนเลือกชั้นที่จะรับข้อมูล แต่ที่ขา ENABLE จะไม่ต่อให้ ENABLE ตลอดเวลาแต่จะมีวงจรมาต่อเข้าไปอีกเป็นส่วน DECODER ข้อมูลที่จะมาทริกให้ทำการส่งข้อมูลไปยังชั้นที่ได้ถูกตัว 4 TO 16 DECODER ทำการเลือกไว้แล้วข้อมูลที่จะนำมาใช้ในการ DECODER เพื่อทริกให้ส่งหรือไม่ส่งข้อมูลนี้ ก็ใช้จาก 4 BIT ที่เหลือของ PORT B นำ 4 BIT นี้มาผ่าน 3 LINE TO 8 LINE DECODER เป็นไอซีเบอร์ 74LS138 เพื่อนำสัญญาณที่ได้จาก 1 ใน 8 line มาใช้ ENABLE ให้ตัวไอซี 4 TO 16 DECODER ทำงานได้ สัญญาณที่เป็นทั้งตัวทริกให้ส่งข้อมูลและเลือกชั้นที่จะส่งข้อมูลซึ่งเป็นสัญญาณเส้นเดียวกันนี้จะถูกส่งไปให้ส่วนที่ 1.2 ต่อไป สำหรับสัญญาณ DECODER เลือกชั้นที่จะรับข้อมูลนั้นจะถูกต่อไปใช้สำหรับ ENABLE BUFFER ข้อมูลของแต่ละชั้นซึ่งมีทั้งหมด 16 ชั้น BUFFER ของแต่ละชั้นจะถูกควบคุมด้วยสัญญาณที่ได้จากวงจร DECODER ที่ว่านี้เพื่อให้ข้อมูลของแต่ละชั้นไม่ชนกัน รายละเอียดของวงจรที่ได้จากการออกแบบมาเป็นดังรูปข้างล่างนี้



1.2 บอร์ด ENCODER,RS-422 DRIVER/RECEIVER บอร์ดนี้จะมีวงจร 3 วงจรคือ วงจร ENCODER,วงจร DECODER และวงจร RS-422 DRIVER/RECEIVER จะแยกกล่าวเป็นวงจร ๆ ไปในที่นี้เรายังต้องใช้ข้อมูลอีกสองอย่างที่ยังไม่ได้ใช้เลยคือ ข้อมูลในการใช้เลือกห้องกับข้อมูลในการใช้เพื่อควบคุมอุปกรณ์ไฟฟ้าที่ปลายทาง ข้อมูลที่กล่าวมาจะใช้กับวงจรดังนี้คือ วงจร ENCODER จะเป็นวงจรที่นำข้อมูลที่จะส่งไปควบคุมอุปกรณ์ปลายทางกับข้อมูลในการเลือกห้องมาเข้ารหัสกันนั่นคือ เอาข้อมูลสำหรับการควบคุม

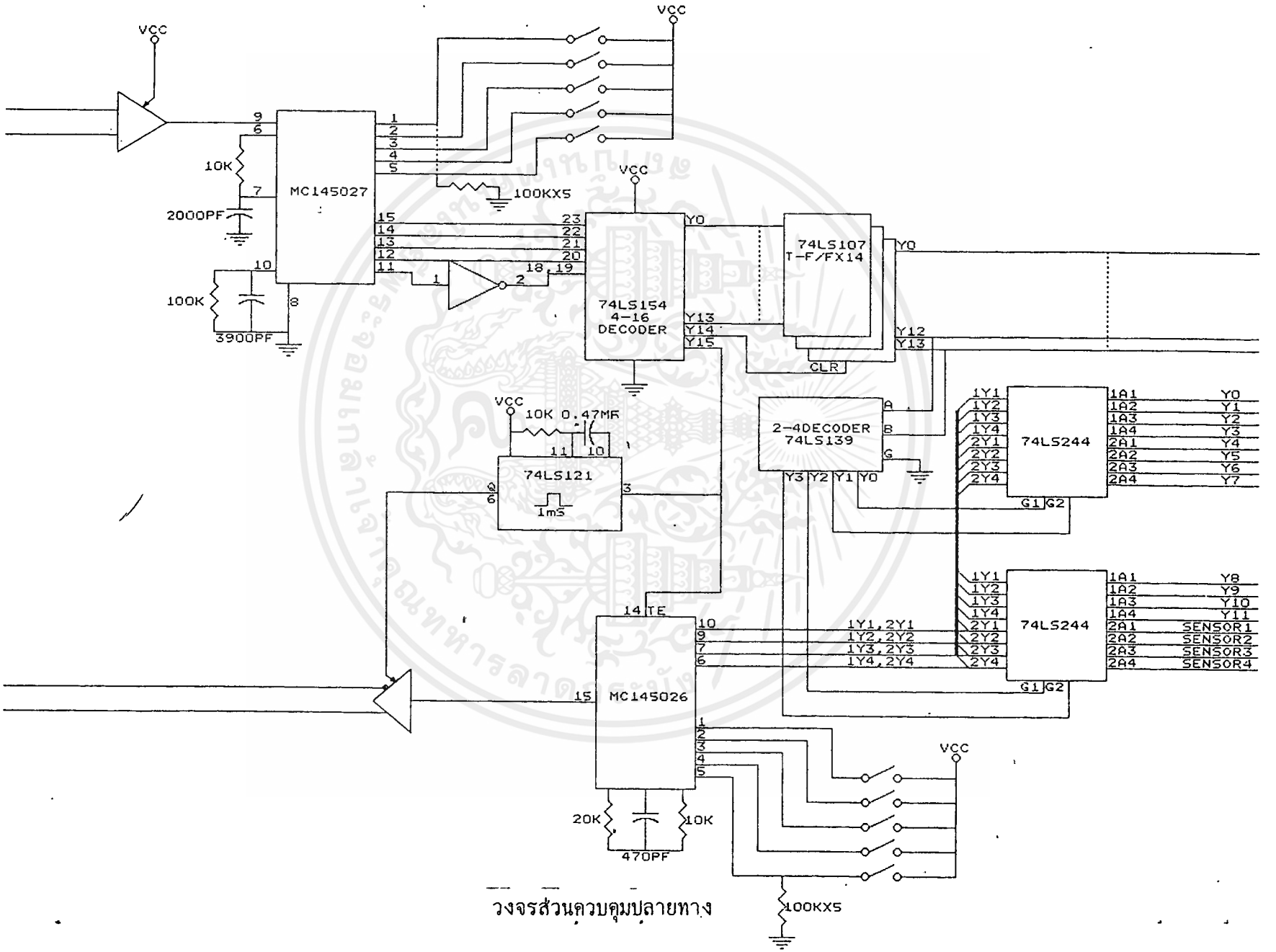
อุปกรณ์ปลายทางทั้งหมด 4 เส้นสัญญาณ กับข้อมูลที่ใช้ในการเลือกห้องที่จะส่งข้อมูลอีก 5 เส้นสัญญาณรวมเป็นทั้งหมด 9 เส้น มาทำการเข้ารหัสออกมาให้เหลือเพียง 1 เส้นสัญญาณ เพื่อใช้ส่งไปยังส่วนควบคุมปลายทาง ซึ่งในส่วนที่กล่าวมานี้คือส่วน ENCODER นี้เราใช้ไอซีสำเร็จรูปเพียงตัวเดียวคือไอซีเบอร์ MC145026 เป็นไอซี ENCODER โดยเฉพาะ สำหรับสัญญาณที่ได้จากการ ENCODER นี้ยังไม่พร้อมที่จะใช้ในการส่งไปไกล ๆ เพราะมีกำลังไม่พอและเนื่อง จากเป็นสัญญาณแบบ UNBALANCE ซึ่งจะถูกรบกวนตามสายส่งสูงมาก จนไม่อาจใช้ส่งไกล ๆ ได้ ดังนั้นเราต้องทำการเพิ่มกำลังในการส่งและแปลงเป็นสัญญาณแบบ BALANCE เพื่อจะได้ส่งไปได้ไกล ๆซึ่งเราใช้ไอซีที่ทำหน้าที่นี้เบอร์ XC3487 เป็น RS-422 LINE DRIVER สัญญาณที่ได้จากไอซีเบอร์นี้ก็พร้อมที่จะส่งไปได้ไกล ๆ เพื่อให้ส่วนควบคุมปลายทางรับไปทำงานต่อ จากนั้นมาดูวงจร RS-422 RECEIVER จะทำหน้าที่รับสัญญาณจากส่วนควบคุมปลายทางที่ส่งมาในลักษณะแบบ BALANCE มาแปลงเป็นแบบ UNBALANCE เพื่อใช้ในวงจร DECODER โดยไอซีที่ทำหน้าที่แปลงสัญญาณนี้คือ ไอซีเบอร์ MC3486 LINE RECEIVER สัญญาณที่ได้จากไอซีตัวนี้จะส่งไปให้วงจร DECODER ที่ทำหน้าที่ DECODER สัญญาณ ไอซีที่รับหน้าที่นี้ก็คือ ไอซีเบอร์ MC145027 ซึ่งเป็นไอซี DECODER ที่จะทำการ DECODER ข้อมูลที่ส่งมาในรูปการเข้ารหัสระหว่าง สัญญาณข้อมูลและสัญญาณที่บอกหมายเลขห้องเพื่อแยกข้อมูลที่ส่งมาออกจากข้อมูลหมายเลขห้อง ซึ่งในการที่จะสามารถ DECODER ข้อมูลออกมาได้นั้นทางตัว DECODER ต้องกำหนดหมายเลขห้องให้ตรงกับของสัญญาณที่ส่งมาไม่อย่างนั้นแล้วจะไม่สามารถทำการ DECODER ได้เลย นั่นคือในขณะที่เราทำการส่งข้อมูลไปยังส่วนควบคุมปลายทางซึ่งเราต้องกำหนด ข้อมูลที่จะส่งข้อมูลหมายเลขห้อง ข้อมูลหมายเลขชั้นแล้วเราจะสามารถส่งข้อมูลไปได้และยังจะสามารถรับข้อมูลของห้องนั้นมาได้ด้วยเพราะหมายเลขห้องที่ปลายทางกับที่ส่วน DECODER แบบเปรียบเทียบหมายเลขห้องกันนี้จะตรงกันทำให้สามารถ DECODER เอาข้อมูลที่ส่งมาได้ สำหรับรายละเอียดของวงจรในส่วนนี้ แสดงดังรูปในหน้าถัดไป



2.ส่วนควบคุมอุปกรณ์ปลายทาง ส่วนนี้ประกอบไปด้วยวงจร RS-422 DRIVER/RECEIVER,ENCODER,DECODER,T FLIPFLOP ซึ่งจะเห็นว่ามีหลายส่วนเพื่อทำงานในการรับข้อมูลมาควบคุมอุปกรณ์ไฟฟ้า และเพื่อส่งข้อมูลสถานะของอุปกรณ์ไฟฟ้าและตัวตรวจจับไปให้ส่วนประมวลผลข้อมูลการทำงานของวงจร วงจร RS-422 DRIVER/RECEIVER จะเป็นส่วนที่ใช้ในการรับข้อมูลและส่งข้อมูลไปให้ส่วนประมวลผลและควบคุมแบบ AUTOMATIC ซึ่งข้อมูลที่ได้มาตามสายส่งเป็นแบบ RS-422 ซึ่งเป็นแบบ BALANCE เราต้องมาแปลงกลับเป็นแบบ UNBALANCE โดยใช้ไอซีเบอร์ MC3486 RS-422 LINE RECEIVER เมื่อแปลงกลับมาแล้วต่อไปสัญญาณนี้จะส่งมาให้ส่วนวงจร DECODER เพื่อทำการถอดรหัส เพราะสัญญาณที่ส่งมาเป็นสัญญาณที่ถูกเข้ากันระหว่างข้อมูลที่ใช้ควบคุมอุปกรณ์ไฟฟ้ากับข้อมูลที่เป็นหมายเลขห้อง ตัว DECODER จะทำการถอดรหัสข้อมูลออกมาจากรหัสหมายเลขห้องโดยที่ตัว DECODER นี้จะสามารถถอดรหัสได้ก็ต่อเมื่อรหัสหมายเลขห้องที่ส่งมาตรงกับรหัสหมายเลขห้องที่ตัว DECODER อยู่ ตัว DECODER นี้เราใช้ไอซีเบอร์ MC145027 DECODER ซึ่งเป็นคู่กับไอซี ENCODER เบอร์ MC145026 ENCODER ที่ทางส่วนประมวลผลใช้อยู่ เมื่อเราได้ข้อมูลที่ใช้ในการควบคุมอุปกรณ์ไฟฟ้าแล้วเราก็ส่งมาให้วงจรส่วนต่อไปเนื่องจากข้อมูลที่ได้ในขณะนี้มีเพียง 4 BIT แต่เราใช้ควบคุมอุปกรณ์ไฟฟ้าต่าง ๆ มากกว่านี้ เราจึงต้องส่งมาให้ส่วน DECODER แบบ 4 TO 16 DECODER คือใช้ไอซีเบอร์ 74LS154 มาถอดรหัสตัว DECODER นี้จะทำการถอดรหัสก็ต่อเมื่อมีการ

ENABLE มันในที่นี้เราใช้สัญญาณที่ได้มาจากตัว DECODER ที่อยู่ในส่วนข้างหน้าที่กล่าวมาแล้วนั้น ซึ่งสัญญาณนี้จะเกิดขึ้นทุกครั้งที่ตัว DECODER ทำการ DECODER รหัสข้อมูลออกมาเราใช้สัญญาณนี้ในการทริกให้ 4 TO 16 DECODER ทำการ DECODER ข้อมูลออกมาเมื่อไม่มีสัญญาณนี้เกิดขึ้นแล้ว ตัว 4 TO 16 DECODER นี้จะไม่ทำงาน นั่นคือเอาท์พุทที่ได้จะเกิดขึ้นทุกครั้งที่มีการถอดรหัสจากส่วนแรกก่อนเท่านั้น ดังนั้นสัญญาณเอาท์พุทที่ได้จะเป็นลักษณะสัญญาณสำหรับใช้ทริกคือเป็น พัลส์ช่วงสั้น ๆ สัญญาณพัลส์สั้น ๆ นี้จะใช้ในการทริก T FLIP-FLOP ให้เกิดการ ON-OFF เพื่อใช้ปิด - เปิดอุปกรณ์ไฟฟ้าต่าง ๆ ที่ควบคุมอยู่ นั่นคือส่วนวงจรต่อไปจะต้องเป็นวงจร T FLIP-FLOP สำหรับในวงจรนี้เราใช้ T FLIP-FLOP จำนวนทั้งหมด 14 ตัวคือใช้ในการควบคุมอุปกรณ์ไฟฟ้าปกติจำนวน 8 ตัว ใช้ควบคุมอุปกรณ์ไฟฟ้ายามฉุกเฉิน คือยามที่มีการตรวจจับได้สิ่งผิดปกติจะสั่งให้อุปกรณ์ไฟฟ้าบางตัวทำงานเพื่อแก้ไขสิ่งผิดปกตินั้น จำนวน 4 ตัว ที่เหลืออีก 2 ตัวจะใช้ในการจำสถานะของการส่งข้อมูลกลับไปให้ส่วนประมวลว่า ขณะนี้เราส่งข้อมูลชุดไหนอยู่เพราะเราใช้วิธีส่ง 4 ครั้งในการส่งข้อมูลทั้งหมดในหนึ่งห้อง เมื่อเราใช้ T FLIP-FLOP 2 ตัวในการจำข้อมูลที่ใช้ในส่วนนี้เราก็นำมา DECODER อีกทีจะได้ 4 สัญญาณพอดี ซึ่งสัญญาณทั้ง 4 นี้จะใช้ในการทริกให้ตัวส่งข้อมูลกลับว่าจะส่งข้อมูลชุดไหนกลับไป ในวงจรนี้เราใช้ไอซี 74LS139 2 TO 4 LINE DECODER ซึ่งจะทำการ DECODER สัญญาณทั้งสองที่มาจาก T FLIP-FLOP เมื่อ DECODER ออกมาได้แล้วจะนำมาใช้ในการ ENABLE BUFFER ขนาด 4 BIT ที่มีทั้งหมด 4 ชุด รวมแล้วจะมีสัญญาณที่จะส่งกลับให้ส่วนประมวลผลทั้งหมด 16 BIT พอดี ในวงจร BUFFER นี้เราใช้ไดซีเบอร์ 74LS244 ซึ่งภายในเป็น BUFFER ขนาด 4 BIT 2 ชุด เราจึงใช้ไอซีเบอร์นี้ 2 ตัวเท่ากับ 4 ชุดพอดี เมื่อเราได้วงจรที่ใช้กำหนดชุด BUFFER ที่จะใช้ในการส่งข้อมูลกลับและได้วงจรส่วน BUFFER แล้วต่อจากนั้นเราก็มาดูที่วงจรที่จะทำการเข้ารหัสข้อมูลทั้ง 4 BIT ที่จะทำการส่งกลับกับรหัสข้อมูลหมายเลขห้องเพื่อที่ทางด้านตัวรับจะได้ตั้งรหัสหมายเลขห้องให้ตรงกันจะได้ทำการถอดรหัสเอาข้อมูล 4 BIT ไปได้อย่างถูกต้อง ในวงจรนี้ใช้ไอซีเบอร์ MC145026 ซึ่งเป็นแบบเดียวกันกับทางด้านส่วนประมวลผลที่ใช้ในการเข้ารหัสข้อมูลส่งมาให้ แต่ว่าที่วงจรนี้จะถูกควบคุมให้ทำการเข้ารหัสโดย ไอซี 4 TO 16 DECODER ที่อยู่ในส่วนที่เราได้กล่าวมาแล้วคือตัว DECODER นี้มีสัญญาณเอาท์พุท 16 เส้น ใช้ในการควบคุมอุปกรณ์ต่าง ๆ ไปแล้ว 14 เส้น เหลืออีก 2 เส้นเราจึงนำมาใช้ควบคุมไอซี MC145026 ให้ทำการเข้ารหัสนี้ 1 เส้นที่

เหลือ 1เส้นจะใช้ในการ RESET T FLIP-FLOP ทุกตัว คือทำการ CLEAR อุปกรณ์ไพฟ้าทุกตัวให้ปิดหมด และ ลบค่าที่จำสถานะของการส่งข้อมูลกลับให้เป็นข้อมูลชุดแรกจากทั้งหมด 4 ชุด จากนั้นเมื่อเราควบคุมให้ไอซี MC145026 เข้ารหัสได้แล้วต่อไปเราก็จะนำเอาสัญญาณที่ผ่านการเข้ารหัสคือสัญญาณหนึ่งเส้นแบบ UNBALANCE ไปทำเป็นสัญญาณแบบBALANCE เพื่อให้ส่งไปได้ไกลขึ้นซึ่งเราก็ใช้ไอซีเบอร์เดียวกับที่ทางส่วนประมวลผลใช้ในการส่งข้อมูล คือเบอร์ XC3487 โดยที่ไอซีตัวนี้สามารถควบคุมทางด้านเอาท์พุทได้ว่าจะให้แปลงสัญญาณออกมาเป็นแบบBALANCE หรือไม่ให้ทำการแปลงโดยให้เป็นสถานะHIGH IMPEDANCE โดยจะมีขาที่ใช้สำหรับควบคุมต่างหากซึ่งที่ผ่านมาเราใช้ไอซีเบอร์นี้ในลักษณะที่ให้ทำงานตลอดคือแปลงสัญญาณตลอด แต่ที่วงจรส่วนนี้เราต้องควบคุมให้ทำงานและไม่ทำงานเป็นบางช่วงคือจะให้ทำงานในช่วงที่มีการควบคุมให้ส่งข้อมูลกลับ นอกจากนั้นจะไม่ให้ทำงานคือมีสถานะทางเอาท์พุทเป็น HIGH IMPEDANCE ทั้งนี้เนื่องจากสายสัญญาณที่ใช้ในการส่งข้อมูลกลับนี้ใช้ร่วมกับห้องอื่น ๆ ในชั้นเดียวกันดังนั้นจึงต้องป้องกันไม่ให้ข้อมูลของแต่ละห้องชนกันโดยใช้วิธีนี้ จะเห็นว่าไอซีตัวนี้จะถูกควบคุมให้ทำงานเพียงช่วงเวลาหนึ่งเราจึงนำสัญญาณที่ใช้ในการควบคุมไอซีMC145026ให้ทำการเข้ารหัสข้อมูลมาทำการหน่วงเวลาออกเพื่อเอาชั้มาควบคุมไอซี XC3487 ให้ทำงานเพียงช่วงเวลาสั้น ๆ ที่ได้หน่วงออกมานั้น ซึ่งจะต้องให้เพียงพอกับที่สัญญาณที่ถูกเข้ารหัสโดยไอซีเบอร์ MC145026 จะผ่านออกไปได้หมดคือถูกแปลงเป็นแบบ BALANCE ได้หมดนั้นเอง วงจรที่ใช้หน่วงเวลานี้เราใช้ไอซีเบอร์ 74LS121 MONOSTABLE ซึ่งเรากำหนดเวลาที่หน่วงไว้ประมาณ 3 mS จากการคำนวณที่พอเพียงที่จะให้ข้อมูลที่เข้ารหัสผ่านออกไปได้หมดพอดี สำหรับวงจรทั้งหมดของส่วนควบคุมปลายทางนี้แสดงดังรูปในหน้าถัดไป



วงจรส่วนควบคุมปลายทาง

บทที่ 3

การทำงานของวงจร

การทำงานของแต่ละวงจร

การทำงานของแต่ละวงจรจะขอกล่าวถึงเป็นส่วน ๆ แบบเดียวกับส่วนฮาร์ดแวร์ดังนี้

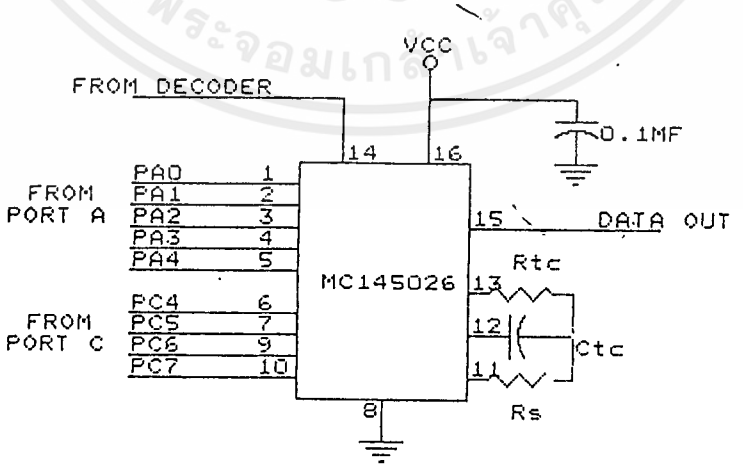
1 ส่วนประมวลผลและควบคุมแบบ AUTOMATIC จะมีวงจร DECODER,ENCODER,RS-422 DRIVER/RECEIVER ซึ่งจะต่อใช้งานร่วมกับ Z180 CONTROL PACK ซึ่งการทำงานมีดังนี้ ทางบอร์ด CONTROL PACK จะส่งข้อมูลออกมาทาง PORT ทั้ง 3 มาให้วงจร DECODER และวงจร ENCODER ทำงานดังนี้ PORT C บน ที่ใช้เป็นข้อมูลสำหรับควบคุมอุปกรณ์ปลายทางและ PORT A ที่ใช้เป็นหมายเลขห้อง จะถูกส่งมาให้ตัว ENCODER ของทุกชั้นเพื่อให้ทำการเข้ารหัสจากข้อมูลแบบขนานมาเป็นแบบอนุกรมแต่ยังไม่สั่งให้ทำงานดังรูปวงจร ENCODER ใช้ไอซี MC145026 ในการทำงานเข้ารหัสข้อมูลส่งออกไปจะเห็นว่ามีอุปกรณ์ R, C ต่อร่วมเพียง 3 ตัวใช้กำหนดความถี่ออสซิเลเตอร์ ในวงจรนี้ใช้ความถี่ 88.7 kH_Z ซึ่งกำหนดโดยสูตร

$$f_{osc} = \frac{1}{2.3\ R_{TC}\ C_{TC}'}$$

$$C_{TC}' = C_{TC} + C_{layout} + 12pF$$

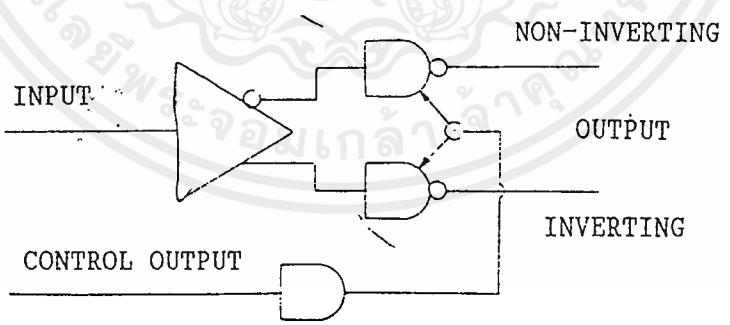
$$100pF <= C_{TC} <= 15\mu F$$

$$R_{TC} >= 10\ K,\ R_S = 2\ R_{TC}$$



รูปวงจรส่วน ENCODER

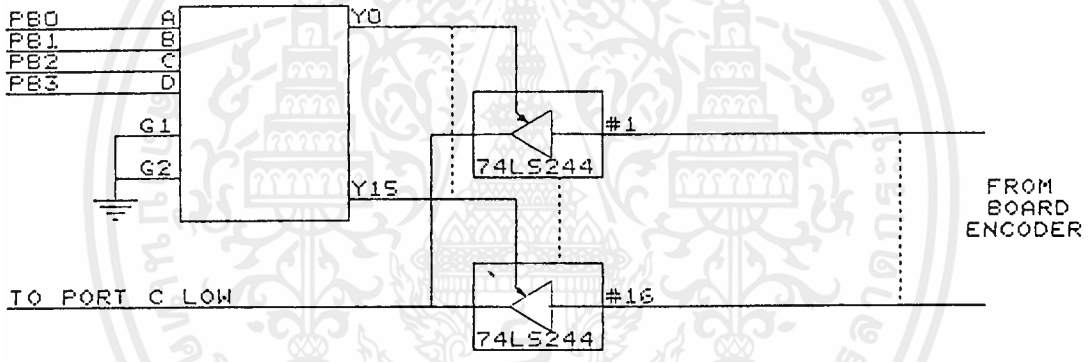
การทำงานของวงจรส่วน ENCODER มีดังนี้ ตัวไอซีจะมีวงจรส่วนออสซิเลเตอร์อยู่ภายในเพียงต่ออุปกรณ์ R,C ภายนอกในการกำหนดความถี่เท่านั้นส่วนการเข้ารหัสจะทำภายในตัวไอซีคือเมื่อมีการทริกที่ขา TE เพื่อให้ไอซีทำงานแล้วไอซีตัวนี้จะนำเอาข้อมูลทางแอดเดรสหรือหมายเลขห้องกับข้อมูลที่จะส่งไปควบคุมอุปกรณ์ในห้องนั้นซึ่งรวมแล้วเป็น 9 BIT มาเข้ารหัสกันตามรูปแบบที่กำหนดของตัวไอซีเอง เมื่อเข้ารหัสแล้วจะส่งข้อมูลที่เข้ารหัสแล้ว ออกมาทางขา DATA OUT ที่มีเพียงหนึ่งขาจะเห็นว่าการเข้ารหัสนี้เป็นการแปลงจากสัญญาณข้อมูลแบบขนานไปเป็นแบบอนุกรมไปในตัวด้วยเพื่อที่จะได้ส่งไปที่ต่าง ๆ โดยประหยัดสายข้อมูล ในการเข้ารหัสและส่งออกไปนี้ตัวไอซีจะทำการส่งออกไป 2 WORD คือข้อมูล 9 BIT ที่นำมาเข้ารหัสในรูปแบบที่กำหนดจะส่งออกมา 2 ครั้งทางขา DATA OUT ในการเข้ารหัสหนึ่งครั้งทั้งนี้เพื่อให้ทางด้านส่วนรับปลายทางสามารถตรวจสอบความถูกต้องได้คือจะนำข้อมูลที่ส่งมาทั้งสองครั้งมาเปรียบเทียบกันเองซึ่งจะกล่าวถึงภายหลัง เมื่อเราได้ข้อมูลที่เข้ารหัสเรียบร้อยทางขา DATA OUT ของไอซีแล้ว เราจะส่งไปให้วงจรที่จะแปลงสัญญาณจาก UNBALANCE ไปเป็นแบบ BALANCE ซึ่งก็คือวงจร RS-422 LINE DRIVER ที่เราต้องทำแบบนี้เพราะว่าจะป้องกันสัญญาณรบกวนในสายส่งจะได้ส่งไปได้ไกลยิ่งขึ้น สำหรับวงจร RS-422 LINE DRIVER แสดงดังรูปข้างล่าง



วงจร RS-422 LINE DRIVER

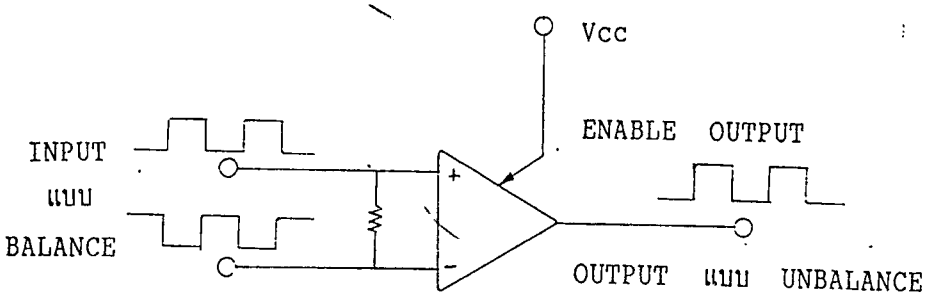
จะเห็นว่าทั้งสองวงจรเมื่อต่อรวมกันแล้วก็พร้อมที่จะส่งข้อมูลออกไปให้อุปกรณ์ควบคุมปลายทางแล้วเพียงแต่รอสัญญาณทริกให้ทำการส่งเท่านั้น สำหรับวงจรทั้งสองที่ต่อรวมกันนี้ใช้สำหรับส่งข้อมูลให้อุปกรณ์ปลายทางเพียงหนึ่งชั้นเท่านั้น ดังนั้นเพื่อที่จะใช้ควบคุมได้ 16 ชั้นเราต้องมีวงจรทั้งสองเท่ากับ 16 ชุด โดยที่ทางด้าอินพุททั้งหมดขนานกันหมดแต่จะยังไม่มีชุดใดทำงานคือจะรอสัญญาณทริกอยู่และจะมีสัญญาณมาทริกให้ชุดใดชุดหนึ่งทำงานเพียงชุดเดียวเท่านั้นในการส่งหนึ่งครั้ง สำหรับสัญญาณทริกนี้ก็คือสัญญาณเลือกชั้นนั่นเองซึ่งจะได้มาจากวงจร DECODER ซึ่งการทำงานมีดังนี้ จากรูป ข้อมูลจาก CONTROL PACK ที่ใช้ในการเลือกชั้นจะส่งออกมาทาง PORT B ในการเลือกชั้นนี้จะมีทั้งเลือกชั้นที่จะส่งข้อมูลและเลือกชั้นที่จะรับข้อมูล ซึ่งจะต้องแยกกันเราจะเริ่มกล่าวถึงส่วนที่จะใช้เลือกชั้นในการส่งข้อมูลก่อนดังนี้ เราเอาข้อมูลจาก PORT B มาแบ่งเป็นส่วนที่จะใช้เลือกชั้นกับส่วนที่จะ ENABLE ตัว DECODER เพื่อให้ทำการเลือกชั้นโดยเราใช้ PORT B ส่างคือ PB0 ถึง PB3 มาใช้ในการ DECODER เลือกชั้นจะได้ 16 ชั้น พอดีเราใช้ไอซีเบอร์ 74LS154 ซึ่งเป็น 4 TO 16 LINE DECODER สำหรับสัญญาณที่จะมาทำให้ไอซีตัวนี้ทำงานหรือ ENABLE ก็จะใช้ข้อมูลที่เหลือของ PORT B คือ PB4 ถึง PB7 ซึ่งขา ENABLE ของไอซี 74LS154 มี 2 ขาจะ ENABLE ที่สถานะ LOW เราก็เอา PB7 ป้อนเข้าตรง ๆ หนึ่งขา ที่เหลืออีกหนึ่งขาเราใช้การ DECODER ข้อมูลของ PB4 ถึง PB6 มาป้อนให้คือเราใช้ไอซี DECODER อีกตัวมาทำการ DECODER คือไอซีเบอร์ 74LS138 3 TO 8 LINE DECODER ซึ่งจะมีขาเอาท์พุท 8 ขาเราใช้เพียงหนึ่งขามาป้อนให้กับขา ENABLE อีขาที่เหลือของไอซี 74LS154 ในที่นี้เราใช้ขา Y1 ของ 74LS138 ซึ่งเมื่อเราจัดขาสัญญาณเรียบร้อยแล้วเราจะได้ว่าข้อมูลจาก PORT B ที่ PB4 ถึง PB7 จะมีค่าเท่า 0100b ในเลขฐานสองซึ่งเท่ากับ 4 ในเลขฐานสิบ ส่วน PB0 ถึง PB3 จะมีค่าเท่าใดขึ้นอยู่กับหมายเลขชั้น สำหรับการเลือกชั้นในการส่งนี้จะมีสัญญาณทริกให้ส่งข้อมูลออกไปด้วยดังนั้นเมื่อทำการส่งข้อมูลเรียบร้อยแล้วเราจะต้องหยุดทริกให้ส่งข้อมูลด้วยนั้นคือต้อง DISENABLE ไอซีที่ทำการเลือกชั้นที่จะส่งข้อมูลโดยการเปลี่ยนค่าข้อมูลที่ทำการ ENABLE มันซึ่งก็คือข้อมูลที่ PORT Bบน หรือ PB4 ถึง PB7 จะต้องไม่ใช่ค่า 0100b ในเลขฐานสอง ก็จะเป็นการDISENABLE มันแล้วซึ่งตัวที่ควบคุมให้มันทำงานหรือไม่ทำงานนี้ก็คือ CPU ที่อยู่ใน CONTROL PACK นั่นเอง สำหรับการเลือกชั้นที่จะรับข้อมูลนี้จะไม่ยุ่งยากเหมือนการเลือกชั้นที่จะทำการส่ง เพราะว่าเราไม่ต้องควบคุมให้มันทำงานหรือหยุดทำงานแต่เราจะให้มันทำงานตลอดคือจะสามารถรับ

ข้อมูลจากชั้นที่เราเลือกได้ทันทีที่มีการส่งข้อมูลไปนั้นคือเราใช้ข้อมูลชุดเดียวกับที่เลือกชั้นในการส่งข้อมูลแต่จะให้ตัว DECODER นี้ถูก ENABLE ให้ทำงานตลอดไม่มีการควบคุมนั้นคือจะใช้ข้อมูลจาก PORT B คือ PB0 PB3 ป้อนให้กับตัว 74LS154 4 TO 16 LINE DECODER ที่ถูกทำให้ ENABLE อยู่แล้ว นั่นก็คือว่าเมื่อเลือกชั้นที่จะส่งข้อมูลก็จะเป็นการเลือกชั้นที่จะรับข้อมูลไปในตัว และสัญญาณที่ได้จากการ DECODER ชั้นที่จะรับข้อมูลนี้จะส่งไปควบคุมชุด BUFFER ที่เป็นตัวเปิดให้ข้อมูลจากแต่ละชั้นผ่านเข้าไปยัง PORT C ล่าง ซึ่งจะเป็น BUFFER ขนาด 4 BIT เมื่อเลือกชั้นไหน BUFFER ของชั้นนั้นก็จะเปิดให้ข้อมูลของชั้นนั้นผ่านเข้ามารอที่ PORT C ล่างเพื่อรอให้ CPU มาอ่านเข้าไปใช้งานสำหรับ BUFFER นี้จะใช้เป็นไอซีเบอร์ 74LS244 BUFFER DRIVER สำหรับวงจรของส่วน DECODER เพื่อเลือกชั้นและวงจร BUFFER แสดงดังในรูป



วงจร DECODER เพื่อเลือกชั้นและวงจร BUFFER ของข้อมูลในแต่ละชั้น

เมื่อเรามีวงจร BUFFER สำหรับข้อมูลแต่ละชั้นแล้วต่อไปจะเป็นส่วนที่จะรับข้อมูลที่ส่งมาจากส่วนควบคุมปลายทาง ซึ่งจะประกอบด้วยส่วนที่แปลงสัญญาณจากแบบ BALANCE ที่ใช้ส่งมาตามสายส่งมาเป็นแบบ UNBALANCE ที่ใช้สำหรับถอดรหัสต่อไป วงจรที่แปลงสัญญาณนี้คือวงจร RS-422 LINE RECEIVER ซึ่งใช้ไอซีเบอร์ MC3486 ทำงานเพียงตัวเดียวโดยมีความต้านทานอีกหนึ่งตัวทำหน้าที่เป็นความต้านทานทางอินพุทแสดงดังรูป



รูปวงจร RS-422 LINE RECEIVER

เมื่อเราได้สัญญาณข้อมูลที่ส่งมาจากทางส่วนควบคุมปลายทางที่แปลงเป็นแบบ BALANCE แล้วต่อไปเราก็นำมาทำการถอดรหัสเอาข้อมูลสถานะของอุปกรณ์ไฟฟ้าออกจากข้อมูลที่บอกหมายเลขห้องโดยใช้ไอซี DECODER เบอร์ MC145027 ทำการถอดรหัสเพราะเป็นคู่กับไอซีเข้ารหัสของทางส่วนควบคุมอุปกรณ์ปลายทาง ไอซี DECODER จะทำการถอดรหัสได้โดยการป้อนรหัสห้องให้ตรงกับรหัสห้องที่อยู่ในสัญญาณที่จะถอดรหัสนั้นเมื่อรหัสห้องตรงกันแล้วก็จะได้ข้อมูลของอุปกรณ์ไฟฟ้าที่ปลายทางที่ส่งมา ซึ่งจะส่งไปให้วงจร BUFFER ต่อไป สำหรับวงจรที่ใช้ในส่วนนี้แสดงดังในรูปจะเห็นว่าข้อมูลที่ใช้เป็นหมายเลขห้องก็ใช้ข้อมูลจาก PORT A ที่ใช้เป็นหมายเลขห้องในส่วนวงจร ENCODER ที่ใช้เข้ารหัสข้อมูลสำหรับส่งไปควบคุมอุปกรณ์ปลายทางนั้นเองสำหรับ R,C ที่ใช้ในวงจรใช้สำหรับกำหนดความถี่ออสซิเลเตอร์ในตัวไอซี DECODER เองเพื่อให้ตรงกันกับความถี่ของไอซี ENCODER ที่ใช้เข้ารหัส

R_1 >= 10 K

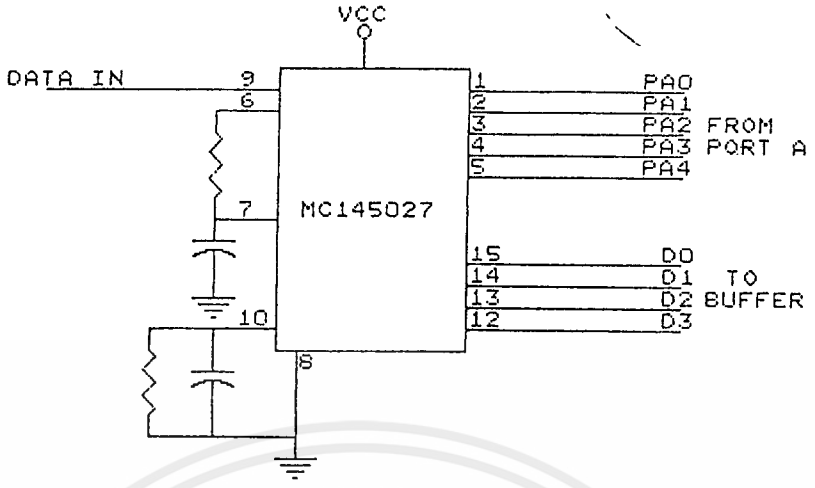
C_1 >= 400 pF

R_2 >= 100 K

C_2 >= 700 pF

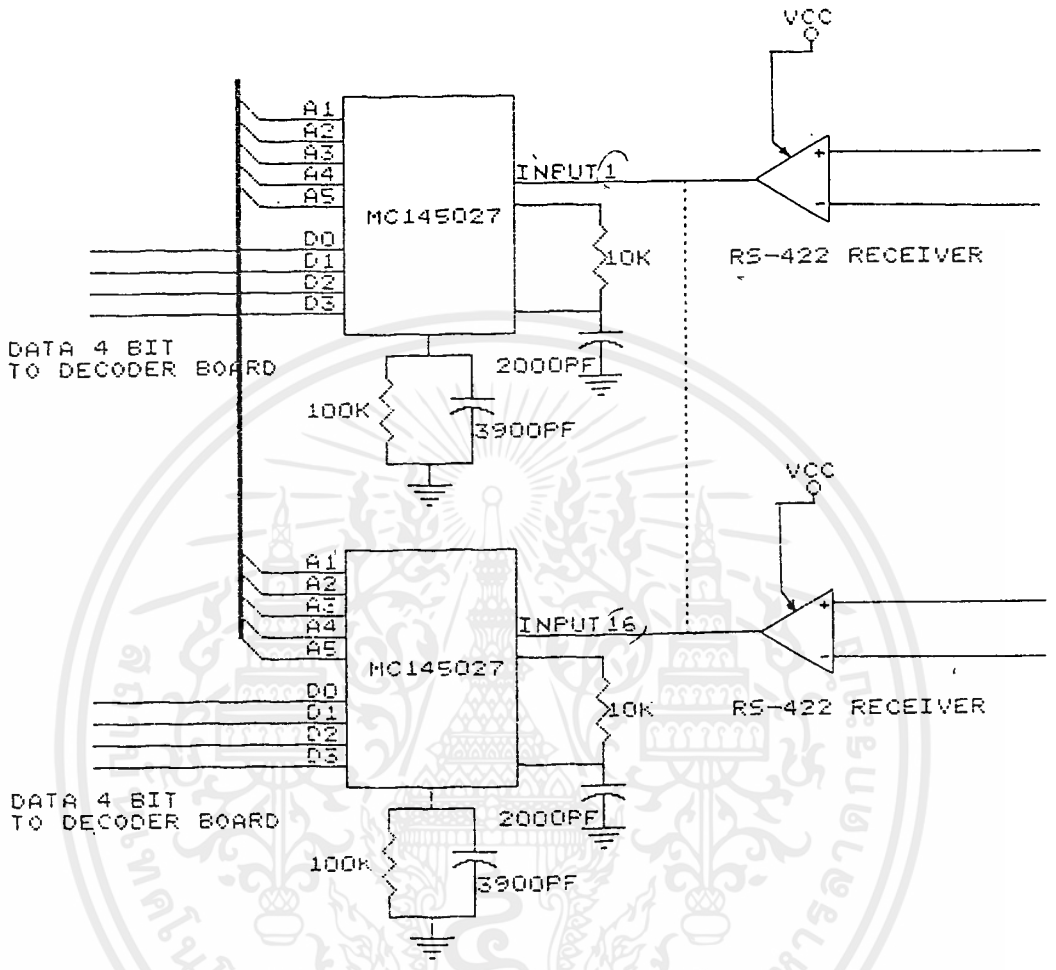
$R_1C_1 = 3.95\ R_{TC}\ C_{TC}$

$R_2C_2 = 77\ R_{TC}\ C_{TC}$



วงจรส่วน DECODER ข้อมูลออกจากหมายเลขห้อง

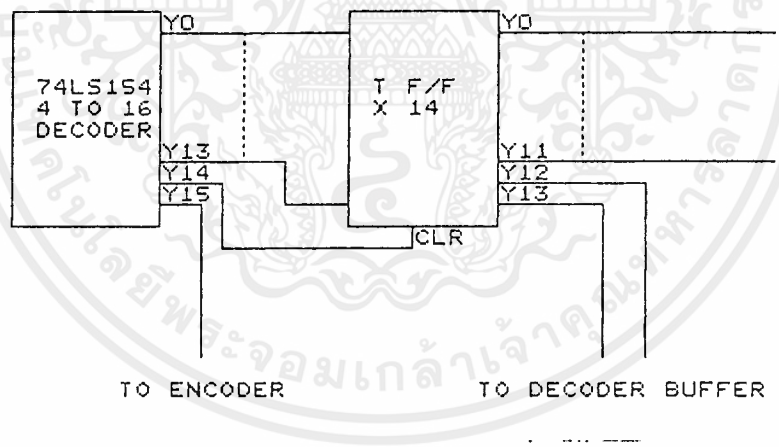
2.ส่วนควบคุมปลายทาง ส่วนนี้จะประกอบด้วยวงจร RS-422 LINE DRIVER/RECEIVER ,DECODER,ENCODER,BUFFER,T FLIP-FLOP วงจรที่ใช้ในส่วนนี้โดยส่วนมากจะใช้วงจรเหมือนกับส่วนประมวลผลและควบคุมแบบ AUTOMATIC เช่น วงจร RS-422, ENCODER-ENCODER ที่ใช้ในการรับส่งข้อมูลและที่ใช้ในการเข้ารหัส-ถอดรหัสข้อมูลของอุปกรณ์ไฟฟ้ากับหมายเลขห้อง การทำงานของวงจรเมื่อรับสัญญาณที่ส่งมาได้แล้วจะทำการแปลงสัญญาณแบบ BALANCE กลับเป็นแบบ UNBALANCE เพื่อใช้ถอดรหัสโดยไอซีถอดรหัส MC145027 ที่เป็นคู่กับไอซีที่เข้ารหัสข้อมูลที่ส่งมาจากส่วนประมวลผลนั่นเองสำหรับวงจรในทั้งสองส่วนนี้แสดงดังรูป สำหรับ R,C ที่ประกอบในวงจร DECODER นั้นใช้กำหนดความถี่ออสซิเลเตอร์ให้เหมือนกันกับของไอซีที่ส่งมา



รูปแสดงวงจรส่วน RS-422 RECEIVER และ DECODER

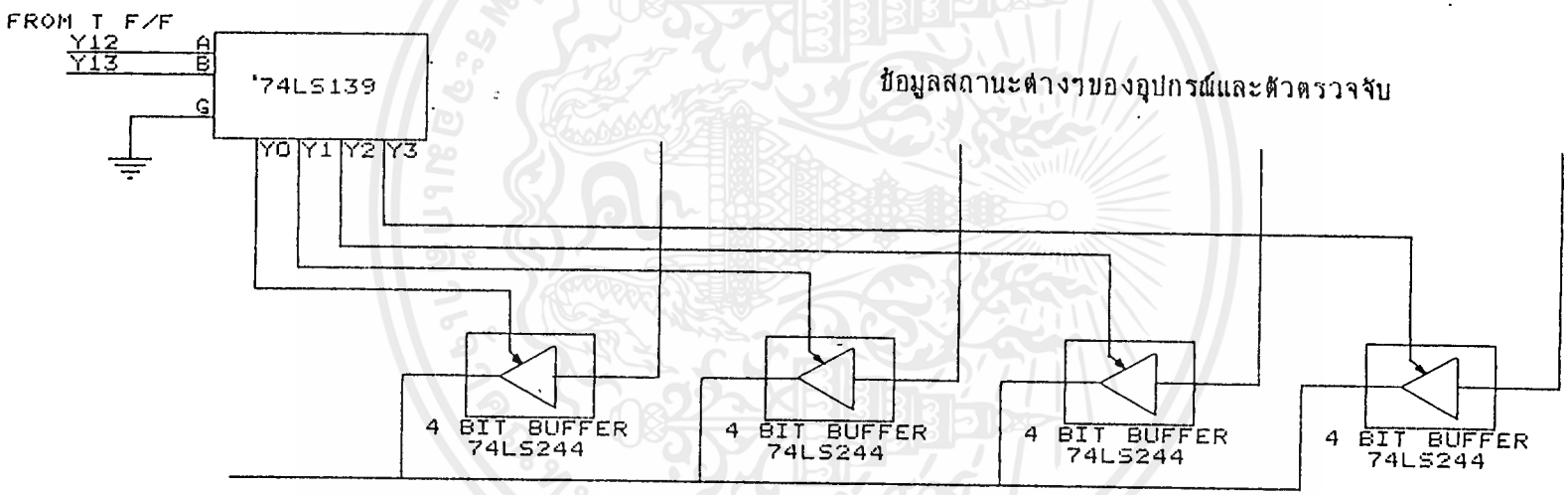
เมื่อเราได้ข้อมูลที่แยกออกจากหมายเลขห้องแล้วเราจะนำข้อมูลนี้มาทำการ DECODER อีก เพื่อให้ได้ข้อมูล 16 เส้นรดยใช้ไอซี 74LS154 ซึ่งเป็น 4 TO 16 LINE DECODER ซึ่งจะถูกควบคุมให้ทำงานรดยสัญญาณที่ป้อนให้ขา ENABLE ซึ่งจะนำมาจากไอซีที่ถอดรหัสข้อมูลออกจากรหัสหมายเลขห้อง เพราะไอซีตัวนี้เวลาที่ถอดรหัสได้ข้อมูลออกมาแล้วจะมีสัญ

ญาณออกมาที่ขา VT ของมันเพื่อเป็นการบอกว่าได้ทำการถอดรหัสข้อมูลออกมาแล้วซึ่งสัญญาณที่ขา VT นี้จะเป็นพัลส์บวกคาบเวลาสั้น ๆ ซึ่งเหมาะกับการใช้ ENABLE ไอซี 4 TO 16 DECODER พอดี ดังนั้นไอซี DECODER นี้ก็จะให้เอาท์พุทออกมาเป็นพัลส์สั้น ๆ ไปด้วยตามคาบเวลาที่มา ENABLE มัน ซึ่งเอาท์พุทนี้จะนำไปใช้ทริก T FLIP-FLOP ให้ทำการ ON หรือ OFF อุปกรณ์ไฟฟ้าต่อไป วงจรแสดงดังรูป จะเห็นว่าเราใช้เอาท์พุทของT FLIP-FLOP ไปควบคุมอุปกรณ์ไฟฟ้าและอุปกรณ์ฉุกเฉินเพียง 12 เส้นที่เหลือ 2 เส้นเราใช้สำหรับจำสถานะให้กับวงจรส่วน DECODER ที่ใช้เลือก BUFFER ที่จะส่งข้อมูลชุดที่จะส่งกลับไปให้ส่วนประมวลผลได้ผ่านเข้าสู่วงจรเข้ารหัสข้อมูลกับหมายเลขห้องหรือ ENCODER นั่นเองตอนนี้เราใช้เอาท์พุทของ 4 TO 16 DECODER ไปแล้ว 14 เส้นเหลืออีก 2 เส้นเราแยกมา 1 เส้นมาใช้สำหรับ CLEAR T FLIP-FLOP เพื่อ OFF อุปกรณ์ทุกชิ้นที่ควบคุมอยู่ส่วนอีกหนึ่งเส้นที่เหลือเราใช้ในการทริกให้วงจรเข้ารหัสข้อมูลทำงานและใช้ทริกให้วงจร RS-422 DRIVER ทำงานด้วย ซึ่งจะได้กล่าวถึงต่อไป



รูปวงจร DECODER และ T FLIP-FLOP

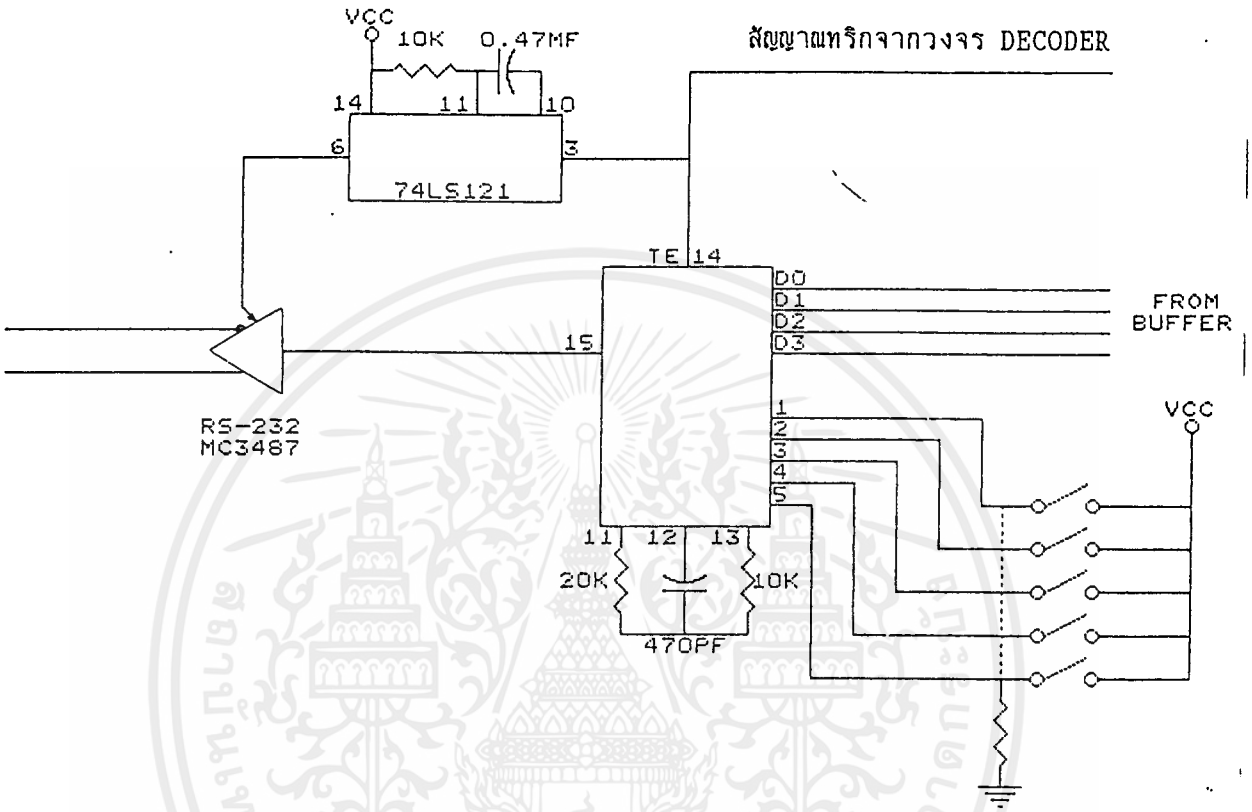
เมื่อเราได้สัญญาณสำหรับจะนำมาเลือก BUFFER แล้วเราต้องทำการ DECODER ก่อนจึงจะนำไปใช้ได้เราจึงนำสัญญาณทั้งสองเส้นจาก T FLIP-FLOP ที่แบ่งไว้แล้วมาเข้าไอซี DECODER แบบ 2 TO 4 LINE เบอร์ 74LS139 ซึ่งจะถูกต่อแบบ ENABLE ตลอด เพื่อที่จะเลือกBUFFERตลอดเวลา เมื่อได้เอาท์พุทออกมาเป็น 4เส้นก็จะนำไปควบคุม BUFFER



รูปแสดงวงจร BUFFER DECODER และวงจร BUFFER

4 ชุด ซึ่ง BUFFER แต่ละชุดจะมีขนาด 4 BIT เท่ากับ ข้อมูลเกี่ยวกับสถานะของอุปกรณ์ไฟฟ้าที่ควบคุมอยู่ 8 BIT อุปกรณ์ไฟฟ้าฉุกเฉิน 4 BIT และสัญญาณจากตัวตรวจจับอีก 4 BIT รวมเป็น 16 BIT พอดี ซึ่งข้อมูลทั้งหมดจะต้องทำการส่ง 4 ครั้ง ๆ ละ 4 BIT จึงต้องใช้ BUFFER 4 ชุดสำหรับวงจร BUFFER DECODER และ BUFFER แสดงดังรูป

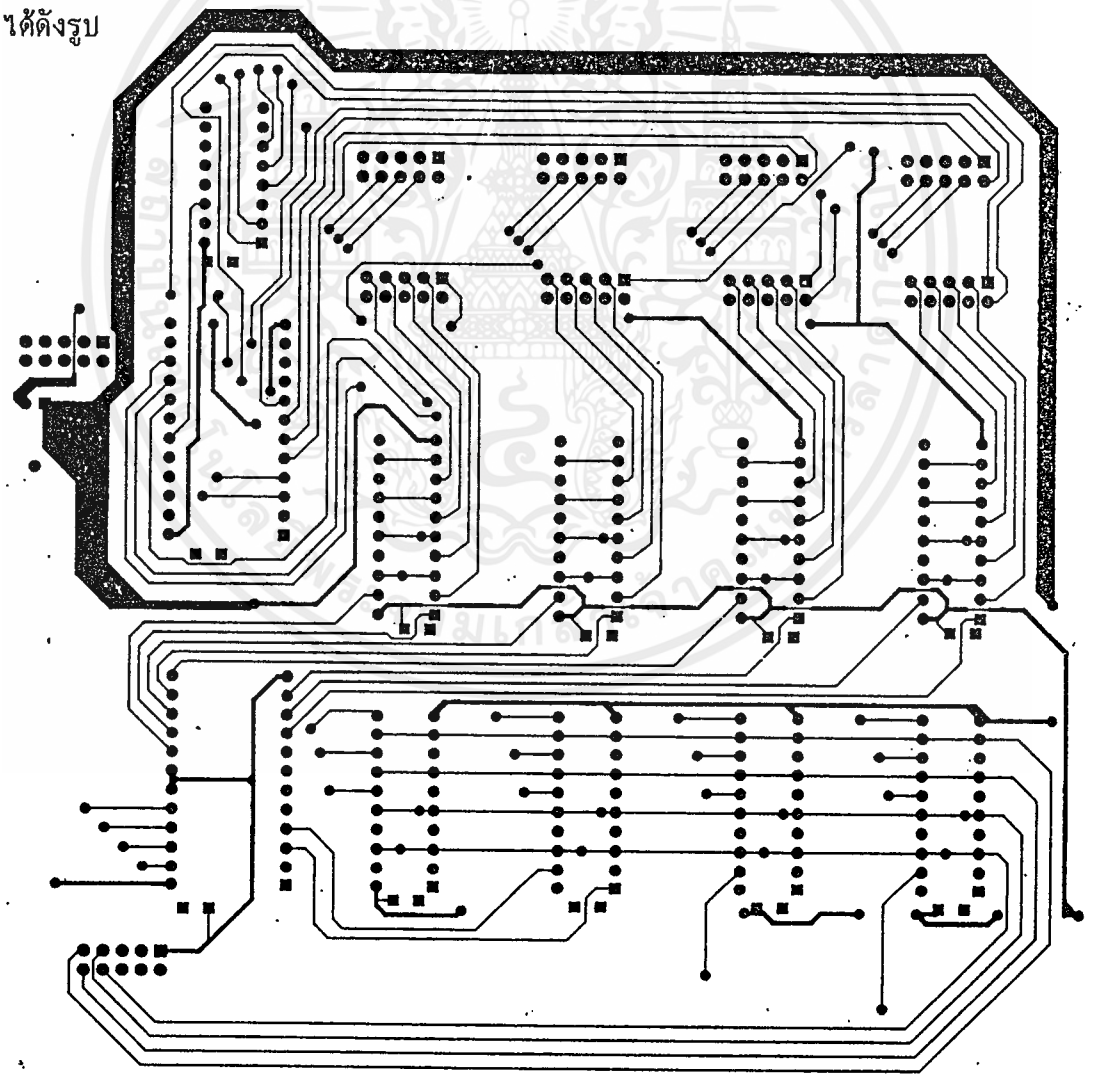
เมื่อเราได้วงจรเลือกข้อมูลที่จะส่งกลับไปให้ส่วนประมวลผลแล้วต่อไปจะเป็นส่วนของวงจร ENCODER ที่ใช้เข้ารหัสข้อมูลเพื่อส่งออกไปเราใช้ไอซีเบอร์ MC145026 เหมือนกับที่ทางส่วนประมวลผลใช้ส่งข้อมูลมาแต่ในวงจรนี้ทางข้อมูลหมายเลขห้องจะถูกกำหนดค่าให้คงที่ตามห้องที่ส่วนควบคุมปลายทางติดตั้งอยู่สำหรับสัญญาณทริกที่จะให้ส่วนเข้ารหัสนี้ทำงานเราได้มาจากวงจร DECODER ที่กล่าวมาแล้วเมื่อเราเข้ารหัสข้อมูลเรียบร้อยแล้วเราก็ได้สัญญาณที่จะทำการส่งออกไปให้ส่วนประมวลผลแล้วแต่ยังเป็นแบบ UNBALANCE ที่มีความต้านทานสัญญาณรบกวนต่ำทำให้ส่งได้ไม่ไกลมากนักเราจึงต้องแปลงเป็นสัญญาณแบบ BALANCE ซึ่งมีความต้านทานต่อสัญญาณรบกวนสูงกว่าและยังส่งได้เป็นระยะทางไกล ๆ ซึ่งวงจรแปลงนี้เราใช้ไอซีเบอร์ MC 3487 RS-422 DRIVER ซึ่งเป็นไอซีที่ควบคุมได้ว่าจะให้ทำการแปลงสัญญาณออกทางเอาท์พุทหรือจะให้สถานะทางเอาท์พุทเป็น HIGH-IMPEDANCE ก็ได้ที่เราต้องใช้การควบคุมสถานะทางเอาท์พุทของไอซีตัวนี้ด้วยเพราะว่าสายส่งข้อมูลที่ทำการส่งข้อมูลกลับไปให้ส่วนประมวลผลนี้เราใช้ร่วมกันกับห้องอื่นด้วยดังนั้นข้อมูลจากแต่ละห้องอาจชนกันได้ สำหรับการควบคุมสถานะทางเอาท์พุทไอซีเราต้องควบคุมให้พอเหมาะคือถ้าสั้นไปข้อมูลที่จะส่งออกก็จะออกไม่หมดเพราะสถานะทางเอาท์พุทเป็น HIGH IMPEDANCE ไปก่อน แต่ถ้าเราควบคุมให้ส่งนานไปแม้ว่าข้อมูลที่จะส่งได้ส่งไปเรียบร้อยแล้วเรายังส่งต่อไปอีกก็จะทำให้ไปเสียเวลาของห้องอื่นที่จะส่งบ้าง สำหรับการคำนวณนี้เราเอาค่าความถี่ที่ใช้ทำงานของไอซีเข้ารหัสข้อมูลมาหาค่าเวลาใน 1 CYCLEของความถี่นั้นจากนั้นเอามาคูณกับ 185 ซึ่งเป็นจำนวน CYCLE ที่ใช้ในการเข้ารหัสข้อมูลหนึ่งครั้งเสร็จแล้วค่าที่ได้จะเป็นค่าเวลาที่ไอซีเข้ารหัสใช้ดังนั้น ไอซี RS-422 DRIVER จะต้องทำงานนานกว่าเวลานี้เพียงเล็กน้อย ในวงจรนี้ใช้ประมาณ 3.5 mS เมื่อเราได้เวลามาแล้วเราต้องสร้างวงจรหน่วงเวลาไป 3.5 mS เราใช้ไอซี MONOSTABLE เบอร์ 74LS121 ที่มีความต้านทานและตัวเก็บประจุสำหรับกำหนดค่าเวลาต่อรวมด้วยซึ่งค่าต่าง ๆ คำนวณจากคู่มือของไอซีเบอร์นี้ สำหรับสัญญาณที่จะใช้หน่วงคือสัญญาณที่มาทริกให้ไอซีเข้ารหัสทำงานนั่นเอง สำหรับวงจรส่วนนี้แสดงดังรูปในหน้าถัดไป



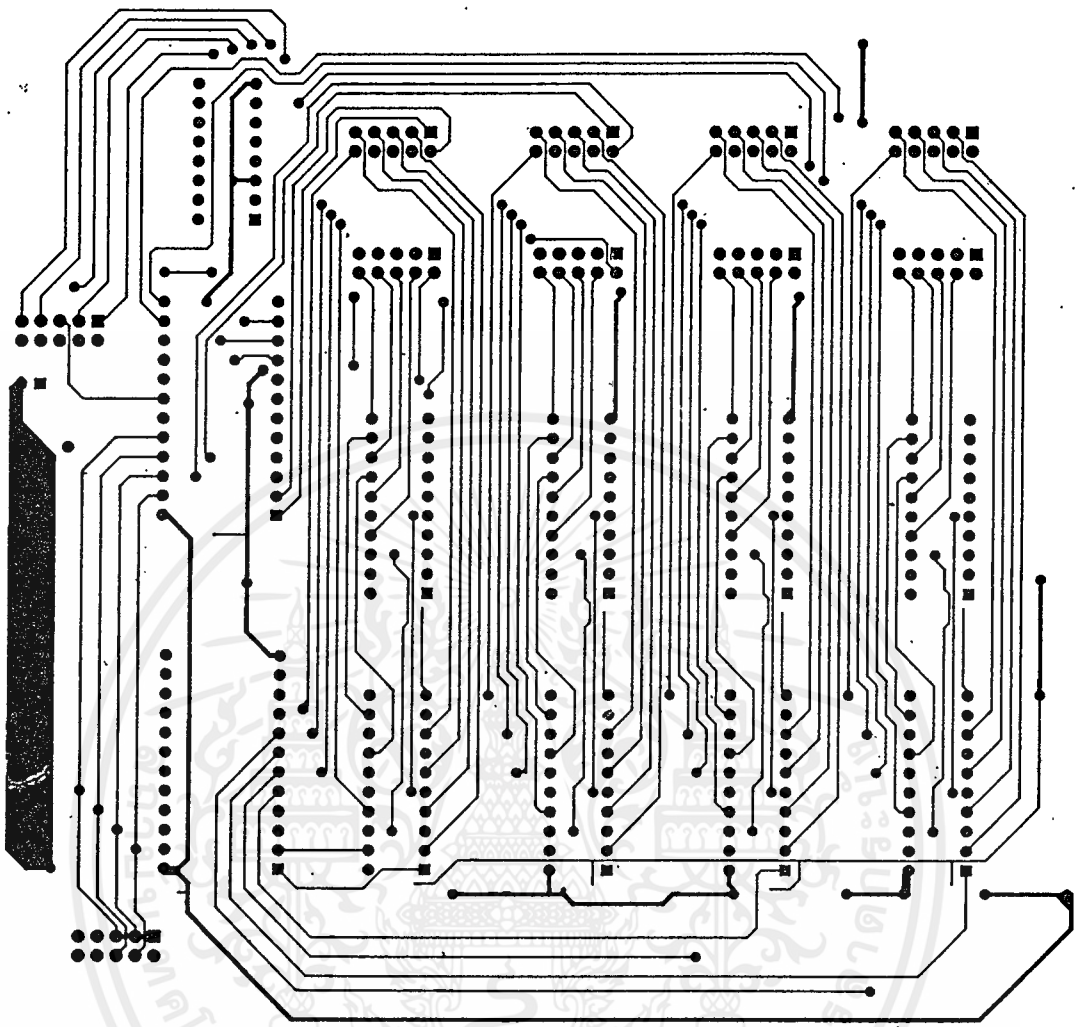
วงจรส่วน ENCODER,MONOSTABLE,RS-422 LINE DRIVER

การออกแบบขั้นสุดท้าย

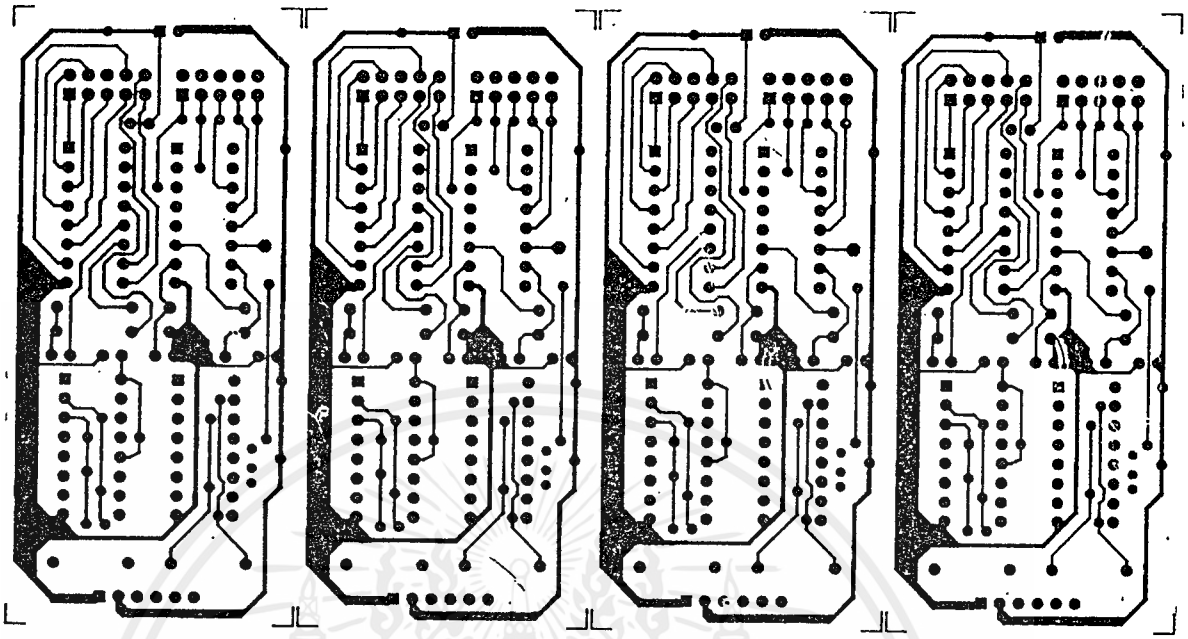
การออกแบบขั้นสุดท้ายหลังจากที่เราได้ทำการทดลองวงจรมาเรียบร้อยแล้วการออกแบบขั้นสุดท้ายก็คือการออกแบบแผ่นวงจรพิมพ์ ซึ่งในที่นี้เราใช้คอมพิวเตอร์ช่วยในการออกแบบซึ่งบางวงจรเราต้องใช้แผ่นวงจรแบบสองหน้าด้วยเมื่อออกแบบเสร็จแล้วจากนั้นเราก็นำเอาวงจรที่ออกแบบโดยคอมพิวเตอร์มา PLOT ลงกระดาษแล้วนำไปถ่ายฟิล์มเมื่อได้ฟิล์มมาแล้วก็นำแผ่นปรินซ์มาทำการเคลือบน้ำยาไวแสงเพื่อจะฉายแสงผ่านฟิล์มมาตกที่แผ่นปรินซ์จะได้ลายวงจรที่ต้องการจากนั้นนำไปล้างน้ำยาส่วนที่ไม่ต้องการออกแล้วนำแผ่นปรินซ์นั้นไปกัดลายทองแดงที่ไม่ได้ถูกน้ำยาเคลือบไว้ออกก็จะได้แผ่นปรินซ์ที่มีลายวงจรตามต้องการซึ่งเราต้องออกแบบทั้งหมด 3 วงจร คือ วงจร DECODER วงจร RS-422 LINE DRIVER /RECEIVER & ENCODER วงจร ส่วนควบคุมปลายทาง ซึ่งลายปรินซ์ที่ออกแบบแล้วแสดงได้ดังรูป



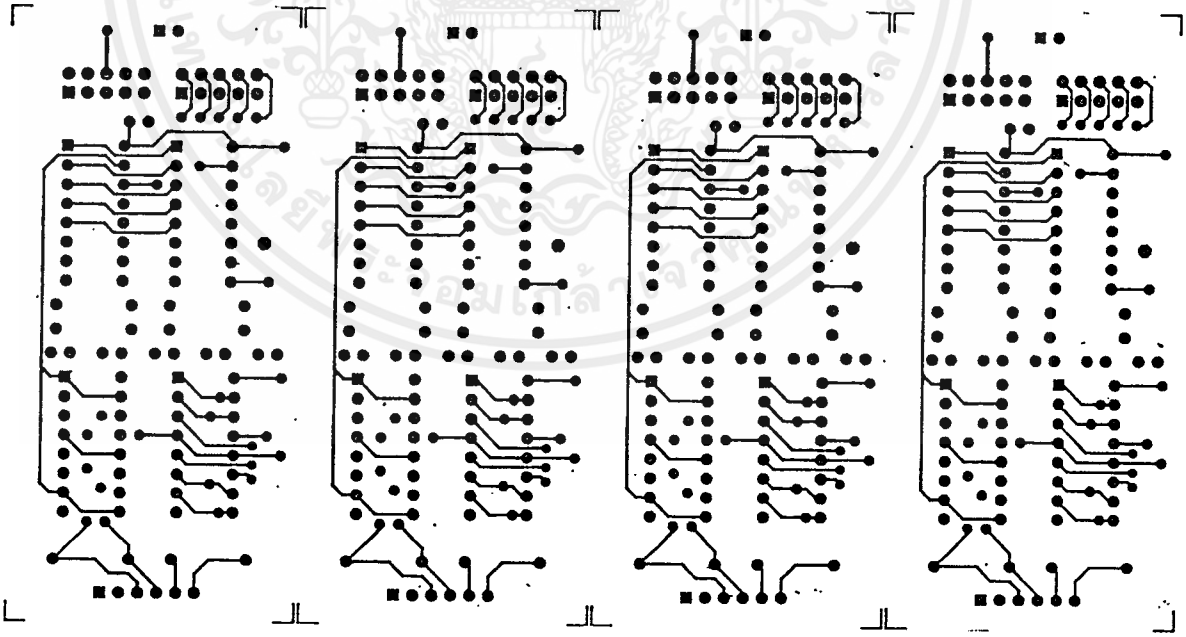
รูปลายวงจร DECODER หน้าที่ 1



รูปลายวงจร DECODER หน้าที่ 2



รูปลายวงจร RS-422 DRIVER/RECEIVER,ENCODER หน้าที่ 1



รูปลายวงจร RS-422 DRIVER/RECEIVER,ENCODER หน้าที่ 2

ปัญหาที่เกิดขึ้นและการแก้ไข

ปัญหาที่เกิดขึ้นมีดังนี้

การออกแบบวงจรผิดพลาด

การคำนวณคาบเวลาที่วงจรหน่วงเวลาในส่วนควบคุมปลายทางส่งข้อมูลกลับผิด

รูปรแกรมที่ทำงานกับส่วนประมวลผลทำงานเร็วเกินกว่าที่ส่วนส่งและรับข้อมูลจะทำงานทัน

การแก้ไข

ออกแบบวงจรและทดลองใหม่

คำนวณคาบเวลาใหม่ให้ถูกต้องและเผื่อค่าผิดพลาดจากอุปกรณ์ที่ใช้ด้วย

ทำการเขียนรูปรแกรมให้มีการหน่วงเวลาไว้ด้วย

สรุปผลการปฏิบัติงาน

จากการที่ได้ทดลองให้ระบบได้ทำงานเต็มที่พบว่าการทำงานได้ตามจุดประสงค์ที่ต้องการ อาจจะมีปัญหาบ้างทางด้านฮาร์ดแวร์หรือซอฟต์แวร์แต่ก็ได้ทำการแก้ไขปัญหาไปเรียบร้อยแล้ว ทำให้ระบบสามารถใช้ควบคุมอุปกรณ์ไฟฟ้าตามห้องพักได้ตามวัตถุประสงค์ซึ่งเป็นผลที่น่าพอใจ นอกจากนี้ทางด้านฮาร์ดแวร์ยังได้ออกแบบให้สามารถเขียนโปรแกรมควบคุมเพิ่มเติมหรือเปลี่ยนแปลงให้เข้ากับงานที่จะประยุกต์ใช้ในอนาคตต่อไป

บทที่ 4

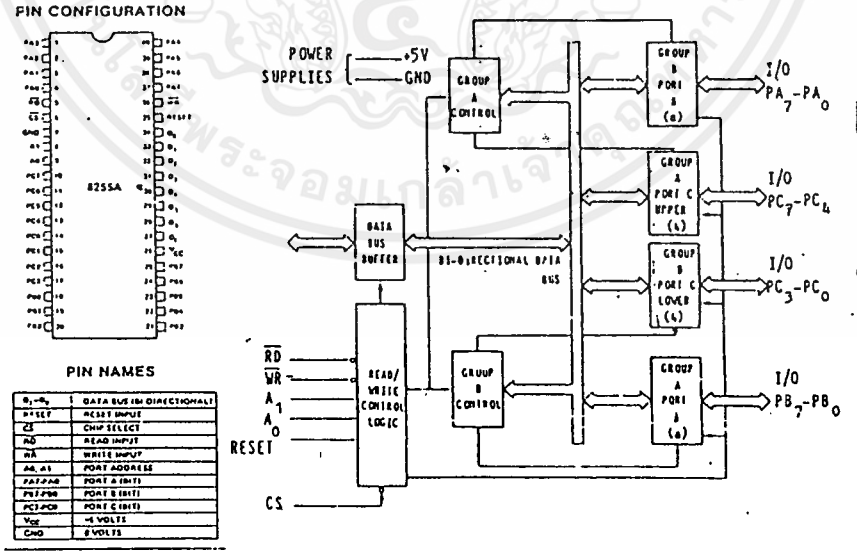
การใช้งาน 8255 PIA

การใช้งาน 8255 PIA

IC 8255 PIA (Programable Interface Adapter) เป็นอุปกรณ์ LSI (Large Scale Integrated Circuit) บรรจุอยู่ใน Packet แบบ Dip 40 ขา ผู้ผลิตมีจุดประสงค์ที่จะใช้งานร่วมกันกับ CPU Z80 โดยเฉพาะ 8255 PIA สามารถแบ่งเป็น BLOCK ได้ 4 Block โดยมี Port ทั้งหมด 3 Port คือ Port A(PA),Port B(PB) และ Port C(PC) ทั้ง 3 Port สามารถทำงานได้ทั้ง Input Portและ Output Port โดย Port A และ Port B จะมี Port ละ 8 bits ส่วน Port C upper และ Port C lower มี Port ละ 4 bits ส่วนนี้คือ Block ที่ 1

ส่วน Block ที่ 2 ได้แก่ Group A control และ Group B control เป็นตัวกำหนดการทำงานให้แก่ Port ทั้ง 3 ทำงานต่างกันอยู่ 3 Mode โดยจะใช้ Control word เป็นตัวสั่งงาน โดย Group A control จะควบคุมเฉพาะ Port C และ Group B control จะควบคุม Port A และ Port B

Block ที่ 3 ได้แก่ Data Bus Buffer และ Read/Write Buffer นี้จะเป็น Buffer ให้กับข้อมูลของ CPU ส่วน Read/Write Control logic เป็นส่วนที่ควบคุมให้ข้อมูลเข้าออกจาก รีจีสเตอร์ภายใน ในเวลาที่เหมาะสม



รูปที่ 4.1 แสดง Block Diagram ของ 8255

จากรูป 4.1 จะเห็นว่าตัว IC มีอยู่ 3 Port ที่ใช้งานคือ Port A,Port B มีขนาด 8 bits และ Port C โดยที่ Port C นั้นยังสามารถแบ่งกลุ่มออกเป็น Port ละ 4 bits จึงทำให้มี Port C บน และ Port C ล่าง คราวนี้การต่อใช้งานก็จะมีขาที่เป็น input ให้กับตัว IC ก็มี D0-D7 ต่อเข้ากับ Data Bus ของ CPU เพื่อใช้สำหรับรับส่งข้อมูลกันระหว่าง Port กับ CPU A0-A1 ขา Address ซึ่งเป็นตัวสำคัญในการกำหนด Port ว่าเรียก Port อะไรเป็น Port A,B หรือ C จากที่กล่าวมาแล้วสถานะที่เราคิดมีเพียงONกับ OFF ดังนั้น IC ตัวนี้จึงมีเบอร์ Port ในตัวมัน 4 Port เพราะมีสาย Address 2 เส้น = สองยกกำลังสอง และเรากล่าวมาแล้ว 3 Port ดังนั้นจึงเหลืออีก Port หนึ่งซึ่ง Port เบอร์นี้เป็นตัวที่สำคัญที่สุดในการทำงานของไอซีตัวนี้ ซึ่งก่อนที่จะให้ไอซีตัวนี้มีหน้าที่ทำอะไรนั้น จะต้องมาการสั่งหน้าที่ของไอซีให้กับ Port นี้เสียก่อน เรียก Port นี้ว่า Control Port ซึ่งจะมีการเรียงลำดับดังนี้

A7	A6	A5	A4	A3	A2	A1	A0	
ไม่สนใจ						0	0	Port A
ไม่สนใจ						0	1	Port B
ไม่สนใจ						1	0	Port C
ไม่สนใจ						1	1	Port Control

ดังนั้นเวลาเราเรียก Port ซึ่งในหนึ่งคำสั่งนั้นจะต้องเรียกเบอร์ Port เป็น 8 bit คือเลข HEX 2 หลักแต่ IC จะให้ Port ไหนทำงานจะมีสำคัญเพียงแค่หลักหลังคือ A1 กับ A0 ว่ามีค่าเป็นอะไร อย่างเช่น บน ET-BOARD ที่ Decode ที่ 8255 ว่างไว้คือ Port 20H ถ้าเราเรียก Port เบอร์นี้ก็จะให้ผล คือ Port A เพราะ A1 กับ A0 เป็น 0 แต่ถ้าเรียก Port 21H บ้างก็เป็น Port B เพราะ A1 เป็น 0 และ A0 เป็น 1 หรือถ้าเราเรียก Port เบอร์ 24H บ้าง Port ที่จะทำงานก็คือ Port A เพราะ A1 กับ A0 เป็น 0 ดังนั้นเวลาเรียก Port 20H กับ 24H จึงเป็น Port เดียวกันบน ET-BOARD เพราะ 1 จุดอ้างถึง 32 Port คือ 20H-3FH หรือสังเกตุอีกอย่างคือ มันจะทับซ้อนตัวเอง เพราะตัว IC อ้างได้ 4 Port พอเกิน 4 Port มันก็จะกลับไปเริ่มต้นใหม่

อย่าง 20H อ้างถึง 4 Port ก็จะถึง 23H พอเพิ่มอีก 1 มันก็จะเพิ่ม 4 Port คือ Port 24H ก็เป็น 20H นั่นเองอย่างเช่น Port 20H, 24H, 28H 2CH, 30H ทั้งหมดนี้ก็คือ Port เดียวกัน(Port A) ในจุด Decode 1 จุดบน ET-BOARD CS เป็นขาเลือก IC Port ให้ทำงานขานี้จะต้องต่อเข้ากับ IC ที่ Decode เบอร์ Port ไว้โดยการเรียกเบอร์ Port นี้ จะรวมเข้ากับ Address 2 เส้นที่ต่อเข้ากับ Port ด้วยคือ A0 กับ A1 เพราะเวลาที่เรียกเบอร์ Port ต้องใช้คำสั่งซึ่งเป็น 8 bit ตาม CPU ดังนั้น 1 คำสั่งจึงรวมสาย Address 2 เส้นนี้ด้วยเพราะอยู่ Address ต่ำบน ET-BOARD 8255 ที่ว่างขา CS จะต่อเข้ากับ 74LS138 ขา 14 คือจุดที่ Decode เบอร์ Port ตั้งแต่ 20H ไว้นั้นเอง(Port A) RD ใช้ขบวนการ input เมื่อ CS และ RD Active เป็น 0 WR ใช้ขบวนการ output เมื่อ CS และ RD Active เป็น 0, Reset เป็น 1 ใช้ Clear สถานะต่างๆของ 8255

PORT ที่ใช้สำหรับ CONTROL

การใช้ 8255 จะต้องสั่งรหัสควบคุม(Control Byte) เข้าไปยัง Port ข้อมูลควบคุม(Port สุดท้ายใน 4 Port คือที่ A1 กับ A0 เป็น 1 เช่น Port 23H บน ET-BOARD ที่ 8255 ว่าง) เพื่อควบคุมการทำงานของ 8255 ว่าให้ทำงานใน Mode ไหนและให้แต่ละ Port เป็น input หรือ output ความหมายของ bit ต่างๆของรหัสควบคุม (Control Byte) หรือรหัสสั่งงาน 8255 ในตอนเริ่มแรก

D7 แสดงถึงรหัสควบคุมให้เริ่มทำงาน(1=ทำงาน) คือจะมีผลให้ 8255 รับรู้ต่อไปใน bit ต่างๆที่กำหนดให้เพราะฉะนั้นเวลาจะสั่งงานหรือกำหนดหน้าที่ให้กับ 8255 bit นี้จะเป็น 1 เสมอ

D6,D5 เป็นการเลือก Mode ในการทำงานของ Port A ซึ่งมี 3 Mode ใน 8255 จะได้กล่าวต่อไป

D4 กำหนดให้ Port A เป็น input หรือ output โดย

0 = output Port

1 = input Port

D3 กำหนดให้ Port C เป็น input หรือ output โดย

0 = output Port

1 = input Port

D2 เป็นการเลือก Mode ให้กับ Port B โดย

0 = Mode 0

1 = Mode 1

D1 กำหนดให้ Port B เป็น input หรือ output โดย

0 = output

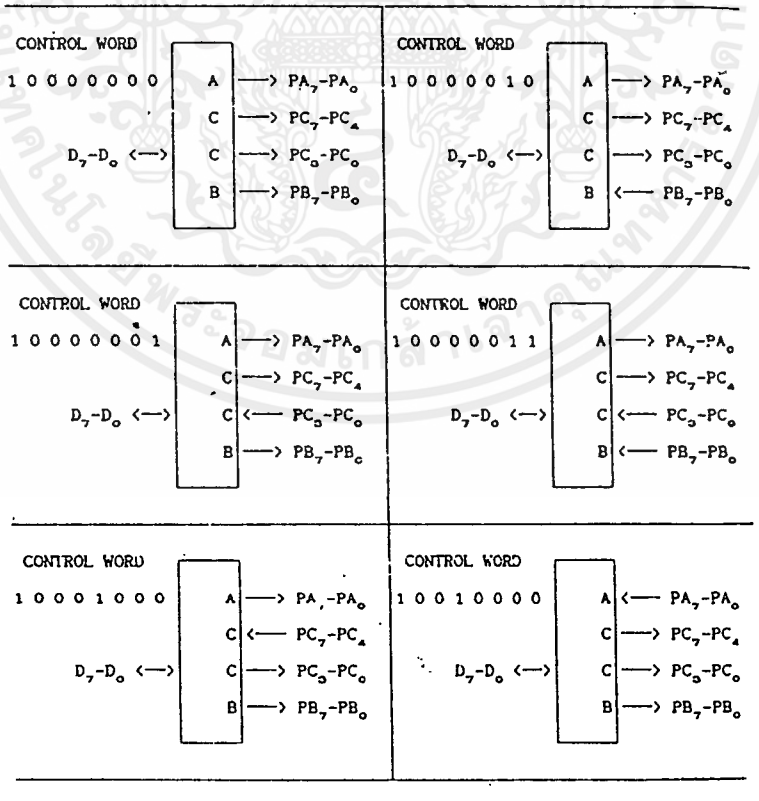
1 = input

D0 กำหนดให้ Port C ล่าง (PC0-PC3) เป็น input หรือ output โดย

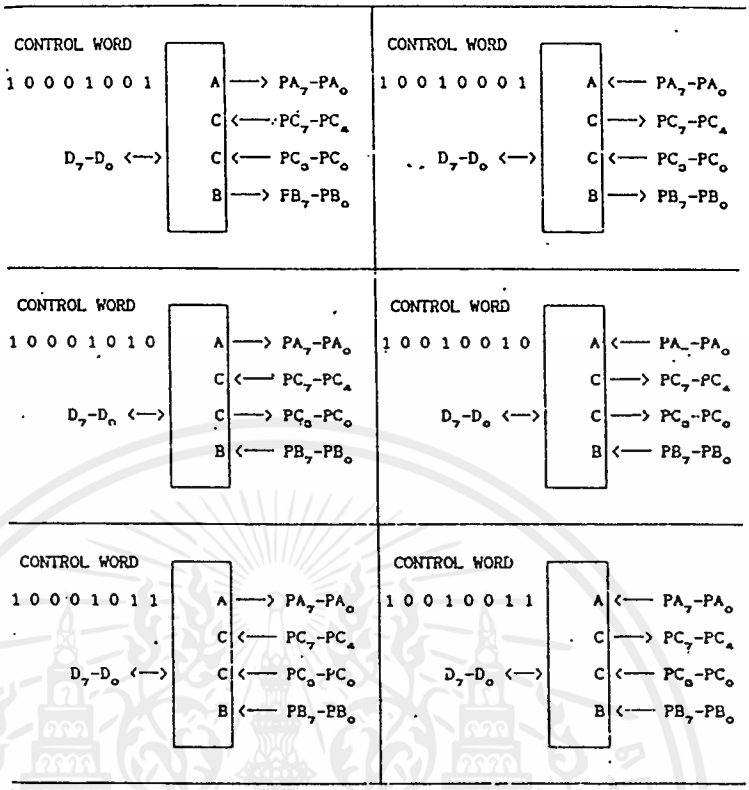
0 = output

1 = input

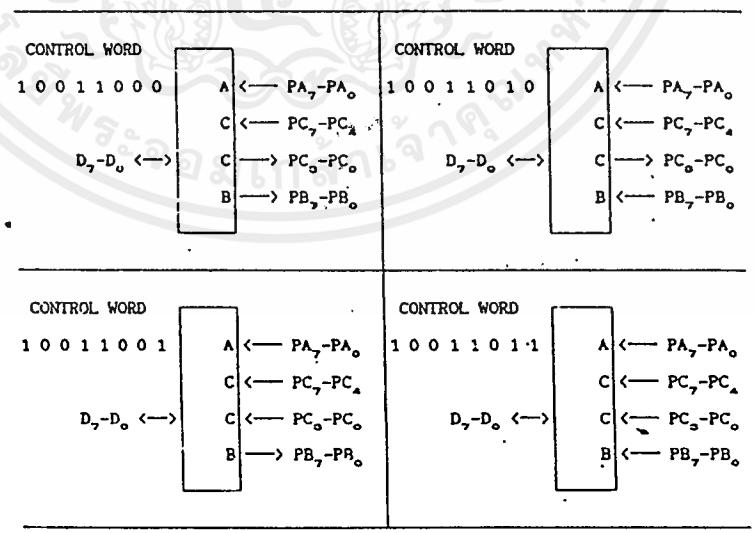
หลักการทำงานของ Port ต่างๆ ที่สามารถกำหนดได้ Mode 0 แสดงไว้ดังรูป 4.2



รูปที่ 4.2 ก



รูปที่ 4.2 ข



รูปที่ 4.2 ค

รูปที่ 4.2 การทำงานของ 8255 แบบต่าง ๆ ในโหมด 0

ตัวอย่างเช่น กำหนดให้ 8255 เป็น Port input output แบบธรรมดาใน Mode 0 โดย Port A เป็น output Port Bเป็น output Port C บนเป็น input และ Port C ล่างเป็น output นี้คือการทำงานของ 8255 บน ET-BOARD เราใส่ code ให้รหัสควบคุมดังนี้

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	0	0	0
8H				8H			

จากนั้นเราก็นำค่านี้ออกไปที่ Port ควบคุมในกรณีของ ET-BOARD Port เริ่มแรกที่ Decode อยู่ที่ 00H จึงนำค่านี้ออก Port สุดท้าย(Port ที่ 4) คือ 03H ดังนี้

```
LD   A,88
OUT  (03H),A
```

เมื่อทำ 2 คำสั่งนี้เสร็จ 8255 ก็จะทำหน้าที่ดังที่เราต้องการเราก็สามารถ นำข้อมูลออก Port ได้แล้วเช่นที่ Port A ต่อ LED Active ที่ Logic 1 เมื่อเราจะให้ LED ติดหมดเราก็สามารถใช้คำสั่งส่งข้อมูลออกไปยัง Port A ได้เลย

```
LD   A,0FFH
OUT  (03H),A      ;ข้อมูลออกที่ Port A
```

ดังนั้นทุกครั้งก่อนใช้ 8255 ต้อง Control Code ก่อน

MODE ต่างๆ ของ 8255

MODE 0

ใช้เป็น Port พื้นฐานทั่วไปคือ กำหนดให้ Port A,B และ C เป็น input หรือ output

MODE 1

เป็น Port input-output ที่มีการตรวจสอบสัญญาณ(Hand shake) โดยใช้ Port C เป็นตัวตรวจสอบ Port A ส่วน C ล่างจะตรวจ Port B แนวคิดของ

Hand Shake คือ ให้มีการทำงานที่พอเหมาะระหว่างอุปกรณ์ที่ทำงานเร็วกับช้า เช่น Computer กับเครื่องพิมพ์โดยที่เมื่อ Computer ส่งตัวอักษรตัวที่ 1 ไปเครื่องพิมพ์รับและกำลังจะพิมพ์ในขณะเดียวกัน Computer ส่งตัวที่ 2 ที่ 3 ออกมาเครื่องพิมพ์ก็ทำงานไม่ทำให้ข้อมูลที่ส่งมาสูญหายจึงต้องมีสัญญาณจากเครื่องพิมพ์ออกไปบอก Computer ว่า ขณะนี้กำลังรับข้อมูลอย่าพึ่งส่งข้อมูลมา เมื่อพิมพ์เสร็จจึงส่งสัญญาณออกไปบอกกับ Computer ว่าให้ส่งตัวอักษรชุดต่อไปมาได้

MODE 3

จะทำได้เฉพาะ Port A ซึ่งจะทำให้ Port A เป็น Port แบบ 2 ทิศทาง คือ เป็นได้ทั้ง input output และยังมี Hand Shake อีกด้วย โดยใช้ Port C ส่วน Port B ก็สามารถใช้เป็น input หรือ output แบบธรรมดาได้อย่างอิสระ

การต่อ Port เข้ากับระบบ Z80

สัญญาณทางไฟฟ้าที่ใช้ในระบบจะประกอบไปด้วย

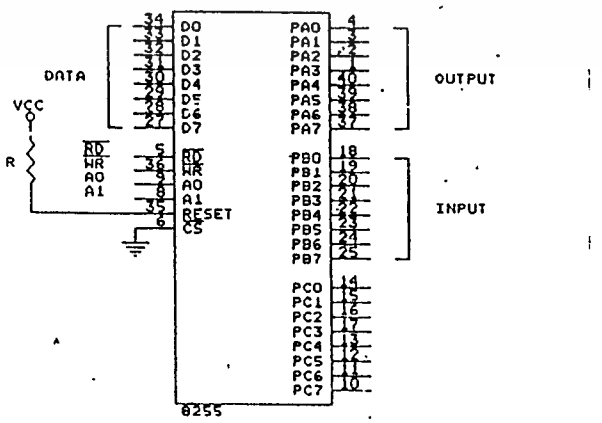
บัสข้อมูล (Data Bus)

บัสแอดเดรส (Address Bus)

บัสควบคุม (Control Bus)

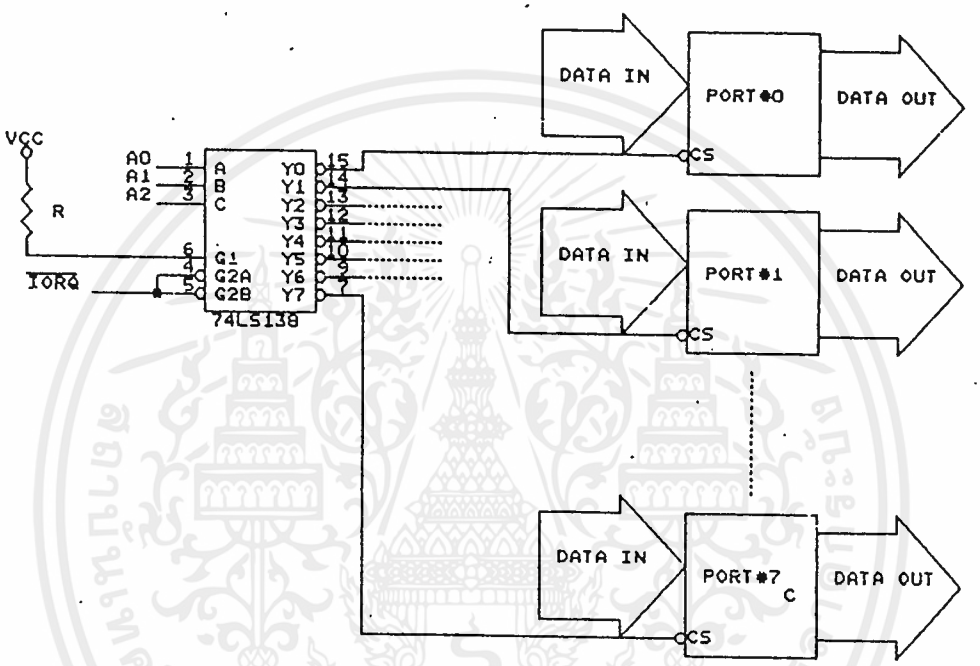
CPU จะส่งข้อมูลที่จะนำไป ยังอุปกรณ์ภายนอกมายัง Data bus ที่ต่อเข้าด้วยกันกับ Port จากนั้นซีพียูก็จะอนุญาติให้ Port ติดต่อได้ด้วยการที่ Port จะติดต่อกับ ซีพียูได้นั้นก็จะใช้สาย Address และสายควบคุม ซึ่งสาย Address จะเป็นตัวกำหนดเบอร์ Port ว่า Port แต่ละตัวอยู่ตำแหน่งไหนและ CPU จะติดต่อกับตำแหน่งใด ส่วนสายควบคุมจะเป็นตัวบอกว่าตอนนี้ซีพียูต้องการติดต่อกับ Port จริงๆ และกำหนดให้เป็น Port input หรือ Port output

Z80 ใช้สาย Address ได้ตั้งแต่ A0-A7 เป็นตัวเลือกเบอร์ Port ด้วยจึงทำให้สามารถเลือก Port ได้ถึง 256 Port คือตั้งแต่ Port 00H-FFH การต่อจะให้เป็น Port input หรือ Output นั้นก็จะใช้สาย RD และ WR จาก CPU ต่อด้วยดังนั้นการต่อ Port เข้ากับระบบจึงมีสาย สำคัญ คือ กลุ่มสาย address ที่สามารถใช้ได้ตั้งแต่ A0-A7 เป็นตัวกำหนดตำแหน่งหรือเบอร์ Port จากนั้นก็สาย IORQ เพื่อแยกการติดต่อระหว่าง Memory และ Port ออกจากกันแล้วตามด้วย WR หรือ RD โดยถ้าใช้ WR ก็เป็น Output Port คือ CPU เขียนข้อมูลหรือส่งข้อมูลที่ต้องการออกไป ยังอุปกรณ์ภายนอก ส่วน RD ก็จะเป็น Input Port คือ อุปกรณ์ภายนอกส่งเข้ามา CPU ทำการอ่านข้อมูลนั้น RD จึงทำงานดังรูป 4.3



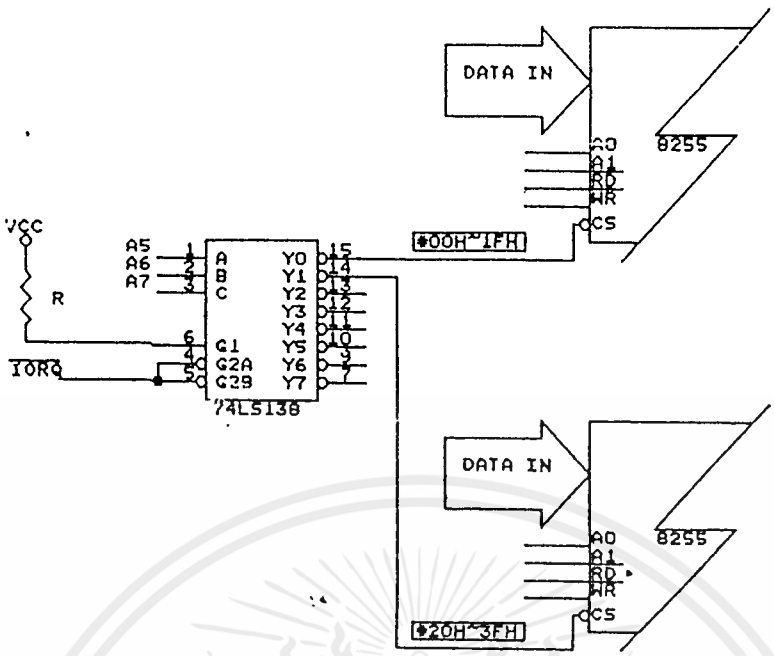
รูปที่ 4.3 แสดงตำแหน่งของ Port จาก 8255

ถ้าเรามีความต้องการใช้ Port มากขึ้นเราก็สามารถใช้ IC Decoder สำเร็จรูปมาใช้เลยจะสะดวกและรวดเร็วกว่าอย่างเช่น 74LS138 ซึ่ง Decode Port ได้ 8 เบอร์ การต่อเราก็ต่อ input ให้กับขาที่จะรับข้อมูลนั้นๆ ให้กับ 74LS138 ดังรูปที่ 4.4



รูปที่ 4.4 แสดงการใช้ IC 74LS138 มาเป็นตัว Decode Port

เช่นต้องการ Decode Port ตั้งแต่เบอร์ 00H-07H เราจะใช้สาย Address 3 เส้นเพราะระบบดิจิตอลจะมี 2 ลักษณะคือ 0 กับ 1 ดังนั้นจึงเท่ากับ 2 จะทำให้ได้ Port 8 เบอร์เลข 3 ที่ยกกำลังนั้นก็คือ จำนวนสาย Address ที่ต่อนั้นเองและเริ่มที่สาย Address ต่ำ คือ A2 A1 A0 เพราะ Port ทั้ง 8 เบอร์นี้ต้องการเบอร์ต่ำ

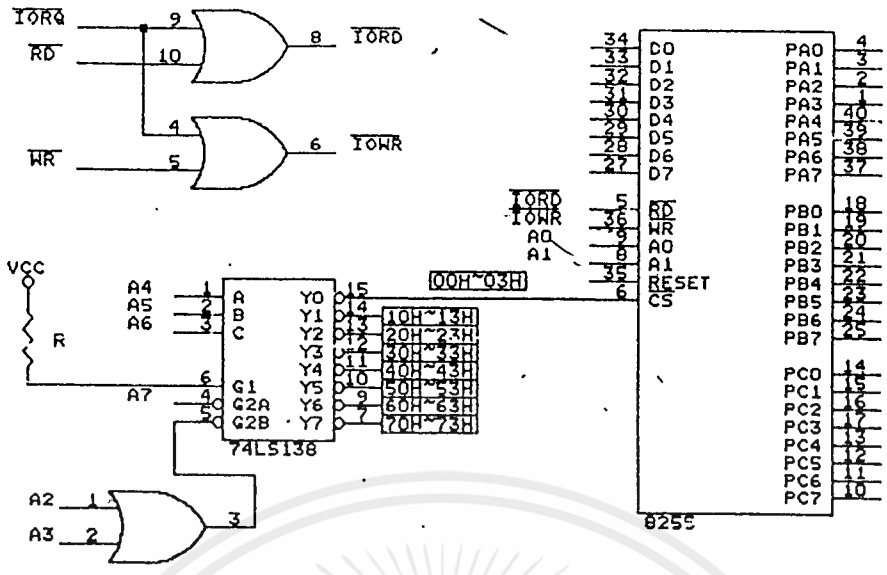


รูปที่ 4.5 การต่อ G2A กับ G2B ต่อกับ IORQ เพื่อบอกว่าต้องการติดต่อ Port

ข้อสังเกตุเมื่อเราเรียก Port เบอร์ 00H กับ 08H จะมีการติดต่อหรือ Port มีการทำงานเป็นตัวเดียวกัน เพราะว่าสายที่เราใช้ถอดรหัสนั้นมีเพียง 3 เส้นใน 8 เส้นที่ใช้ได้ดังนั้นเส้นที่ 4 เป็นต้นไปจึงเหมือนไม่มีค่าอะไรที่น่าสนใจจะคิดเฉพาะ 3 เส้นที่ต่อเท่านั้น

A7 A6 A5 A4	A3 A2 A1 A0	A7 A6 A5 A4	A3 A2 A1 A0
ไม่สนใจ	ที่สนใจ	ไม่สนใจ	ที่สนใจ
0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 0
Port -	0	Port -	8

จากรูปจะเป็นว่าระหว่าง Port 00H กับ 08H 3 bit หลังเหมือนกันเลย เมื่อเราเรียกใช้เป็นตัวเดียวกัน แต่ถ้าจะ Decode ให้ได้เฉพาะที่ต้องการต้องใช้สาย Address ทั้ง 8 เส้นมา Decode ดังรูป 4.6



รูปที่ 4.6 แสดงการใช้สาย Address ทั้ง 8 เส้น มา Decode Port

	A7	A6	A5	A4	A3	A2	A1	A0
	ที่ใช้				ไม่สนใจ			
เริ่มแรก	0	0	0	0	0	0	0	0
Port	0				0			

ให้สาย Address ที่ต่อ Decode มีค่าต่ำสุด คือ ให้เป็น 0 ทุก bit ที่ไม่ได้ต่อก็ให้มีค่าต่ำที่สุดเช่นเดียวกันจะทำให้รู้เบอร์ Port ของ output ที่จุดแรกว่าเป็นอะไรจากนั้นให้ bit ที่ไม่ได้ต่อมีค่าสูงสุดคือเป็น 1 ทุก bit จะทำให้รู้ว่า outputที่ Decode 1 จุดนั้นสามารถอ้าง Port ได้จากเบอร์ไหนถึงไหน

	A7	A6	A5	A4	A3	A2	A1	A0
	0	0	0	1	1	1	1	1
Port	1				F			

ดังนั้นจะได้จุดแรก Y0 เป็น Port เบอร์ 00-1F จากนั้นถ้า A5 เปลี่ยนจาก 0 เป็น 1 ก็คิดเช่นเดียวกัน คือที่ไม่สนใจให้เป็น 0 ทั้งหมดจากนั้นก็ให้เป็น 1 ทั้งหมดดังนั้นที่ Y1 หรือ output ขาต่อไปก็จะเป็น Port 20-3F นั่นเอง และ Port 8255 ที่ว่าก็จะใช้จุดนี้เอง

สรุปขบวนการ PORT INPUT/OUTPUT

เหตุการณ์ในขบวนการ OUTPUT

1. Bus Address A0-A7 จะส่งตำแหน่ง output ภายใต้การควบคุมของซีพียูซึ่งขณะเดียวกัน Bus Address จะถูกถอดรหัสโดย Hardware ส่วน Decode Port

2. Z80 จะส่งข้อมูลที่ต้องการ ส่งออกสู่ Bus ข้อมูล D0-D7 ข้อมูลจะรออยู่ที่ขา input D0-D7 ของ Port

3. ขา IORQ จะ Active Low เพื่อบอกว่า CPU ต้องการจะติดต่อกับ Port และสัญญาณนี้จะค้างอยู่จนกว่าจะสิ้นขบวนการ

5. ขา WR จะ Active Low ทำให้เกิดการเปลี่ยนแปลงข้อมูลออกไปอุปกรณ์ภายนอก และจากนั้น WR ก็จะกลับมาเป็น Logic 1 อีกครั้ง

เหตุการณ์ในขบวนการ INPUT

1. Bus Address A0-A7 ส่งตำแหน่ง Input Port ภายใต้การควบคุมของ CPU ในขณะเดียวกัน Bus Address จะถูกถอดรหัสโดย Hardware ส่วน Decode Port

2. ข้อมูลจากอุปกรณ์ภายนอกจะมารออยู่ที่ input ของ Port เพื่อรอการส่งไปยัง Data Bus ของ CPU จากนั้น CPU ก็จะอ่านข้อมูลจาก Data Bus นั้น

3. ขา IORQ จะ Active Low

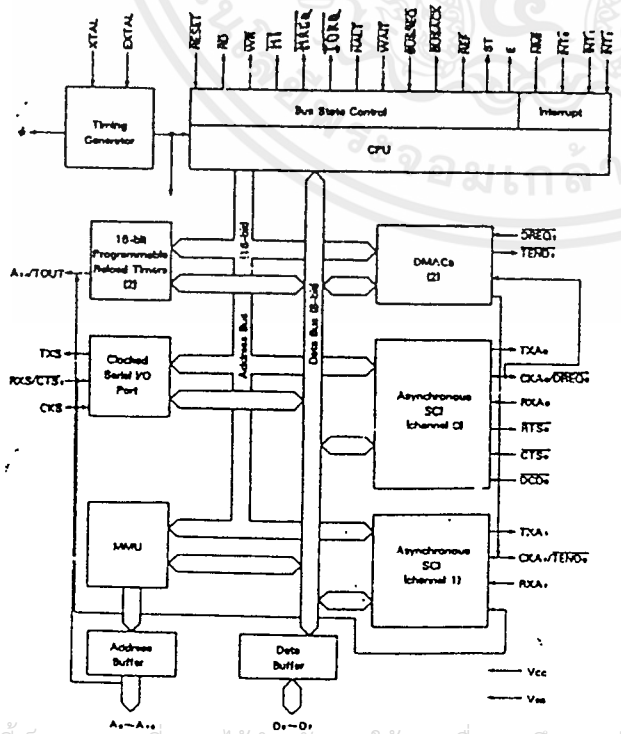
4. ขา RD จะ Active Low จะทำให้ข้อมูลจาก Port เข้าสู่ Data Bus และ RD ก็จะกลับมาเป็น Logic 1 อีก

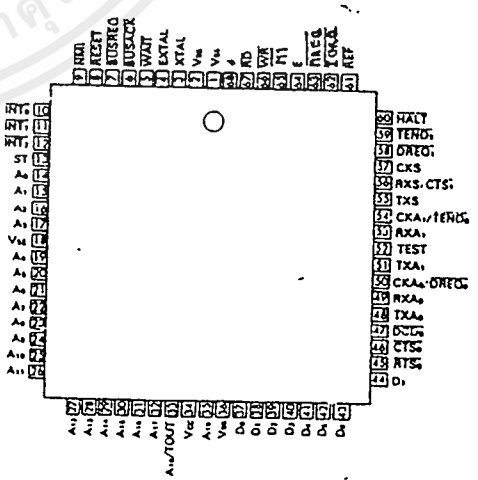
บทที่ 5

Z80180

Z80180 เป็น CPU ที่มีความสามารถสูงที่ได้รวม CHIP สำคัญอื่นๆไว้ใน CPU CHIP เดียว จึงทำให้มีลักษณะคล้ายกับ CPU ที่ใช้ในงาน CONTROL ในจำพวก " SINGLE CHIP " แต่เนื่องจาก SINGLE CHIP มีข้อดี คือ เป็นระบบเล็กราคาถูก แต่ข้อเสีย คือ การโปรแกรม CONTROL ค่อนข้างยากในตอนเริ่มต้นและกับระบบงานที่ใหญ่ขึ้น แต่ Z80180 ทางด้านโปรแกรมจะสะดวกอย่างมาก เพราะคำสั่งที่ใช้มีมาก และตรงไปตรงมาทั้งคู่มือภาษาไทยและตัวอย่างการใช้งานอย่างมากมาย เพราะ CPU Z80180 นี้เป็น SUPPER COMPAT Z80 คือ คำสั่งทั้งหมดยังเป็น Z80 และได้เพิ่มชุดคำสั่งขึ้นมาเพื่อเพิ่มความสะดวกในการใช้งานขึ้นอีก

เมื่อมองดูระบบ MICRO CONTROLLER " SINGLE CHIP " แล้ว Z80180 จะด้อยกว่าตรงที่ไม่มี ROM , RAM และ PORT แต่ถ้าเป็นในระดับงานอุตสาหกรรมแล้วระบบของ Z80180 กับ CHIP MICRO CONTROLLER แล้วจะไม่ต่างกันเลยเพราะความต้องการเนื้อที่ในการเก็บข้อมูลมากและ PORT มากตามจึงทำให้ต้องต่อเพิ่มภายนอกขึ้น จึงทำให้ Z80180 ในระดับงาน CONTROL อุตสาหกรรมคล่องตัวกว่ามากเพราะภายใน Z80180 ประกอบด้วย เป็น CMOS , OSCILATOR ในตัว RUN ที่ 10 MHz , MMU CHIP อ้าง MEMCRY ได้ 1 MBYTE , DMA 2 CHANEL , PORT สื่อสาร UART 2 CHANEL , CLOCK SERIAL I/O , 16 BIT TIMER COUNTER และเกี่ยวกับ PORT สื่อสารสามารถทำ MULTI PROCESSOR COMMUNICATION ซึ่งโครงสร้างของ CHIP นี้จะเป็นดังรูป :-





ขาการใช้งาน

A0–A19	ADDRESS BUS ระหว่าง RESET จะเป็น HIGH IMPEDANCE
$\overline{\text{BUSAK}}$	BUS ACKNOWLEDGE เป็นขา OUTPUT ACTIVE LOW ทำงานก็ต่อเมื่อ Z80180 ตอบสนองต่อการขอ BUS ของ $\overline{\text{BUSRQ}}$ และจะทำให้ BUS ข้อมูล BUS ADDRESS และสัญญาณ CONTROL บางเส้นเป็น HIGH IMPEDANCE
$\overline{\text{BUSRQ}}$	BUS REQUEST เป็นขา INPUT ACTIVE LOW ซึ่งจะมีความสำคัญสูงกว่า NMI โดยจะมีการตรวจสอบสัญญาณนี้ทุกๆการสิ้นสุดของ MACHINE CYCLE
CKA0 , CKA1	ASYNCHRONOUS CLOCK 0 และ 1 เป็นขาสัญญาณ CLOCK แบบ 2 ทิศทาง คือ จะใช้เป็นขา INPUT หรือ OUTPUT ก็ได้
CKS	SERIAL CLOCK เป็นขา CLOCK 2 ทิศทางของ CSI/O
CLOCK	เป็นขา OUTPUT โดยจะเป็นครึ่งหนึ่งของ X'TAL หรือ CLOCK OUT เช่น X'TAL 12 MHz Z80180 จะ RUN ที่ 6 MHz
$\overline{\text{CTS0}}$–$\overline{\text{CTS1}}$	CLEAR TO SEND 0 และ 1 เป็นขา INPUT ACTIVE LOW ใช้ในการควบคุม MODEM
D0–D7	DATA BUS เป็นแบบ 2 ทิศทาง
$\overline{\text{DCD0}}$	DATA CARRIER DETECT 0 เป็นขา INPUT ACTIVE LOW ใช้ควบคุมในการติดต่อกับ MODEM ของ ASCI CHANEL 0
$\overline{\text{DREQ0}}$–$\overline{\text{DREQ1}}$	DMA REQUEST 0 และ 1 เป็นขา INPUT ACTIVE LOW ใช้ในการขอ DMA และขานี้จะโปรแกรมได้ว่าให้ตรวจสัญญาณที่ขอบหรือระดับได้
E	ENABLE CLOCK เป็นขา OUTPUT ACTIVE HIGH ซึ่งใช้ซิงค์การทำงานกับอุปกรณ์ภายนอกระหว่างการทำงานเกี่ยวกับ BUS และใช้เชื่อมต่อกับอุปกรณ์ในตระกูล 68XX และ 80XX
$\overline{\text{HALT}}$	เป็นขา OUTPUT ACTIVE LOW จะทำงานเมื่อทำคำสั่ง HALT หรือ SLP
$\overline{\text{INT0}}$	MASKABLE INTERRUPT 0 เป็นขา INPUT ACTIVE LOW สัญญาณที่ขานี้จะถูกตรวจทุกๆการสิ้นสุดของคำสั่ง
$\overline{\text{INT1}}$, $\overline{\text{INT2}}$	เช่นเดียวกับ INT0 แต่มีระดับความสำคัญรองลงมาตามลำดับ
$\overline{\text{IORQ}}$	เป็นขา OUTPUT เพื่อบอกว่ากำลังติดต่อกับ I/O หรือขา $\overline{\text{IOE}}$ ใน 64180

$\overline{M1}$ MACHINE CYCLE 1 เป็นขา OUTPUT ACTIVE LOW จะทำงานเมื่อ FETCH OP-CODE หรือเป็นขา LIR ของ 64180

$\overline{NMI}$ NON MASKABLE INTERRUPT เป็นขา INPUT ACTIVE LOW ขานี้จะตอบรับการ INTERRUPT เสมอ โดยไม่สามารถหยุดด้วย SOFTWARE

$\overline{RD}$ เป็นขาที่ใช้ทำการอ่านข้อมูลจาก MEMORY หรือ I/O

$\overline{RFSH}$ เป็นขาที่ให้ ADDRESS LOW (A0-A7) ไป REFRESH DYNAMIC RAM หรือ ขา $\overline{REF}$ ของ 64180

$\overline{RTSD}$ REQUEST TO SEND เป็นขา OUTPUT ACTIVE LOW ขานี้ใช้โปรแกรมสัญญาณควบคุมโมเด็มของ ASCI CANEL 0

RXA0 , RXA1 RECEIVE DATA 0 และ 1 เป็นขารับสัญญาณจาก SERIAL PORT ของ ASCI

RXS CLOCK SERIAL RECEIVE DATA เป็นขารับสัญญาณ SERIAL ของ CSIO

ST STATUS เป็นขา OUTPUT ACTIVE HIGH ใช้แสดงสถานะการทำงานของ CPU โดยร่วมกับ $\overline{M1}$ และ $\overline{HALT}$ ดังตาราง :-

ST	$\overline{HALT}$	$\overline{M1}$	Operation
0	1	0	CPU operation (1st op-code fetch)
1	1	0	CPU operation (2nd op-code and 3rd op-code fetch)
1	1	1	CPU operation (MC except for op-code fetch)
0	X	1	DMA operation
0	0	0	HALT mode
1	0	1	SLEEP mode (including SYSTEM STOP mode)

NOTE X: Don't care
MC: Machine cycle

$\overline{\text{TEND0}}$-$\overline{\text{TEND1}}$	TRANSFER END 0 และ 1 เป็นขา OUTPUT ACTIVE LOW ใช้แสดงถึงว่าทำ DMA สิ้นสุดลงแล้ว
TOUT	TIMER OUT ใช้กำเนิดพัลส์จาก PRT CHANEL 1
TXA0 , TXA1	TRANSMIT DATA 0 และ 1 เป็นขาส่งข้อมูล SERIAL ของ ASCI
TXS	CLOCK SERIAL TRANSMIT DATA เป็นขาส่งข้อมูล SERIAL ของ CSIO
$\overline{\text{WAIT}}$	ขา INPUT ACTIVE LOW จะถูกตรวจที่ขอบขาลงของ CLOCK ลูกที่ 2 ของทุกๆ MACHINE เพื่อเป็นการรอให้อุปกรณ์ภายนอกทำงานให้ทันกับการทำงานของ CPU
WR	ใช้สำหรับการส่งข้อมูลไปยัง I/O หรือ MEMORY
X'TAL	เป็นขาที่ใช้ต่อกับ X'TAL

ขาที่ MULTIPLEX

A18/TOUT	ระหว่าง RESET จะเป็น A18 แต่ถ้ามีการเลือก SET BIT TOC1 หรือ TOC0 ใน TIMER CONTROL REGISTOR (TCR) ก็จะทำหน้าที่เป็น TOUT
CKA0/$\overline{\text{DREQ0}}$	ระหว่าง RESET ขานี้จะเป็น CKA0 แต่ถ้า DM1 หรือ SM1 ใน DMA MODE REGISTOR (DMODE) ถูก SET เป็น 1 จะเป็นขา DREQ0
CKA1/$\overline{\text{TEND0}}$	ระหว่าง RESET จะเป็นขา CKA1 แต่ถ้า BIT CKA1D ใน ASCI ถูก SET จะเป็นขา $\overline{\text{TEND0}}$
RXS/$\overline{\text{CTS1}}$	ระหว่าง RESET ขานี้จะเป็นขา RXS ถ้า BIT CTS1E ใน ASCI ถูก SET จะเป็นขา $\overline{\text{CTS1}}$

INTERNAL I/O REGISTOR

ซึ่งมีด้วยกัน 64 I/O ADDRESS ดังแสดงในรูป :-

	Register	Mnemonic	Address	
			Binary	Hexadecimal
ASCI	ASCI Control Register A Ch 0	CNTLA0	XX000000	00H
	ASCI Control Register A Ch 1	CNTLA1	XX000001	01H
	ASCI Control Register B Ch 0	CNTLB0	XX000010	02H
	ASCI Control Register B Ch 1	CNTLB1	XX000011	03H
	ASCI Status Register Ch 0	STAT0	XX000100	04H
	ASCI Status Register Ch 1	STAT1	XX000101	05H
	ASCI Transmit Data Register Ch 0	TDR0	XX000110	06H
	ASCI Transmit Data Register Ch 1	TDR1	XX000111	07H
	ASCI Receive Data Register Ch 0	RDR0	XX001000	08H
	ASCI Receive Data Register Ch 1	RDR1	XX001001	09H
CSI/O	CSI/O Control Register	CNTR	XX001010	0AH
	CSI/O Transmit/Receive Data Register	TRDR	XX001011	0BH
Timer	Timer Data Register Ch 0L	TMDR0L	XX001100	0CH
	Timer Data Register Ch 0H	TMDR0H	XX001101	0DH
	Reload Register Ch 0L	RLDR0L	XX001110	0EH
	Reload Register Ch 0H	RLDR0H	XX001111	0FH
	Timer Control Register	TCR	XX010000	10H
	Reserved		XX010001 ⟩ XX010011	11H ⟩ 13H
	Timer Data Register Ch 1L	TMDR1L	XX010100	14H
	Timer Data Register Ch 1H	TMDR1H	XX010101	15H
	Reload Register Ch 1L	RLDR1L	XX010110	16H
	Reload Register Ch 1H	RLDR1H	XX010111	17H
Others	Free Running Counter	FRC	XX011000	18H
	Reserved		XX011001 ⟩ XX011111	19H ⟩ 1FH
DMA	DMA Source Address Register Ch 0L	SAR0L	XX100000	20H
	DMA Source Address Register Ch 0H	SAR0H	XX100001	21H
	DMA Source Address Register Ch 0B	SAR0B	XX100010	22H
	DMA Destination Address Register Ch 0L	DAR0L	XX100011	23H
	DMA Destination Address Register Ch 0H	DAR0H	XX100100	24H
	DMA Destination Address Register Ch 0B	DAR0B	XX100101	25H
	DMA Byte Count Register Ch 0L	BCR0L	XX100110	26H
	DMA Byte Count Register Ch 0H	BCR0H	XX100111	27H
	DMA Memory Address Register Ch 1L	MAR1L	XX101000	28H
	DMA Memory Address Register Ch 1H	MAR1H	XX101001	29H
	DMA Memory Address Register Ch 1B	MAR1B	XX101010	2AH
	DMA I/O Address Register Ch 1L	IAR1L	XX101011	2BH
	DMA I/O Address Register Ch 1H	IAR1H	XX101100	2CH
	Reserved		XX101101	2DH
	DMA Byte Count Register Ch 1L	BCR1L	XX101110	2EH
	DMA Byte Count Register Ch 1H	BCR1H	XX101111	2FH
	DMA Status Register	DSTAT	XX110000	30H
	DMA Mode Register	DMODE	XX110001	31H
	DMA/WAIT Control Register	DCNTL	XX110010	32H
INT	IL Register (Interrupt Vector Low Register)	IL	XX110011	33H
	INT/TRAP Control Register	ITC	XX110100	34H
	Reserved		XX110101	35H
Refresh	Refresh Control Register	RCR	XX110110	36H
	Reserved		XX110111	37H
MMU	MMU Common Base Register	CBR	XX111000	38H
	MMU Bank Base Register	BBR	XX111001	39H
	MMU Common/Bank Area Register	CBAR	XX111010	3AH
I/O	Reserved		XX111011 ⟩ XX111101	3BH ⟩ 3DH
	Operation Mode Control Register	OMCR	XX111110	3EH
	---	---	---	---

จาก MAP I/O ภายในจะเห็นว่าโปรแกรม Z80 เก่าที่เรามีอยู่อาจจะมีการส่ง PORT ซ้ำกับ I/O ภายใน ทำให้โปรแกรมเดิมทำงานไม่ได้ สามารถแก้ไขได้โดยการโปรแกรมย้าย MAP I/O ภายใน โดยการ CONTROL BIT ใน REGISTOR I/O ICR ADDRESS 3FH ซึ่งสามารถย้ายไปที่ใดก็ได้ภายใน 256 ตำแหน่ง ดังนี้ :-

BIT	7	6	5	4	3	2	1	0
	IOA7	IOA6	IOSTP					

และการโปรแกรมจะเป็นดังนี้ :-

IOA7	IOA6	ช่วง ADDRESS I/O
0	0	0000 - 003FH
0	1	0040 - 007FH
1	0	0080 - 00BFH
1	1	00C0 - 00FFH

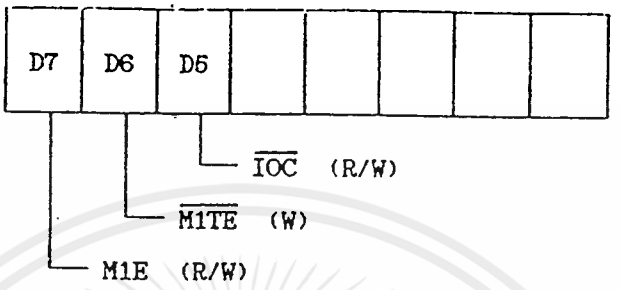
เช่น ต้องการย้าย I/O ภายใน ไป I/O ADDRESS 80H เป็นต้นไป จะโปรแกรมได้เป็น

```
LD   A , 80H
OUTO (3FH) , A
```

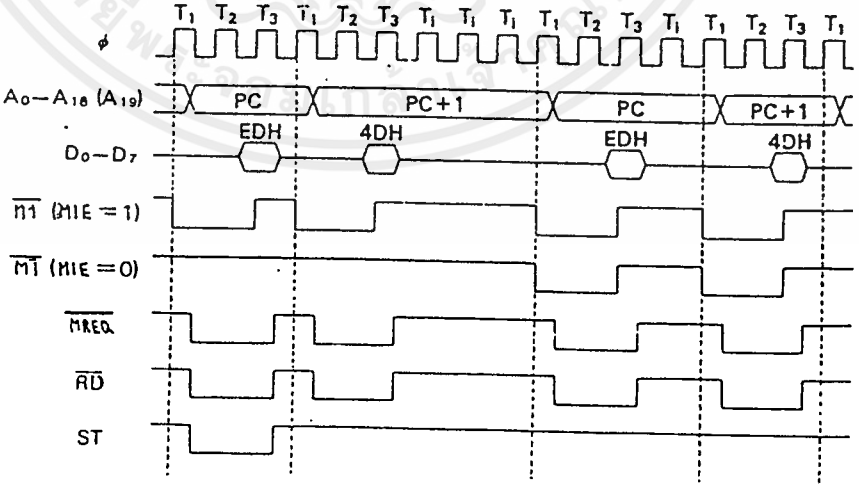
ส่วน IOSTP : IOSTOP MODE BIT 5 เป็น 1 จะทำให้ I/O ภายในหยุดทำงาน เมื่อ RESET BIT นี้จะเป็น 0

OPERATION MODE

Z80180 สามารถกำหนดการทำงานให้เหมือน 64180 ได้ โดยการ SET BIT CONTROL MODE CONTROL REGISTOR (OMCR I/O ADDRESS 3EH)

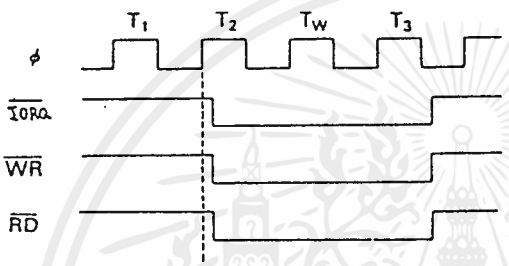


M1E (M1 ENABLE) ระหว่าง RESET BIT นี้จะเป็น 1 $\overline{M1}$ OUTPUT จะเป็น LOW เมื่อ FETCH OPCODE และเนื่องจากการทำคำสั่ง RETI ของ Z80180 จะถูกกระทำ 2 ครั้ง ใน 1 คำสั่ง จึงทำให้เกิด $\overline{M1}$ ขึ้น 2 ครั้งด้วย อันอาจจะทำให้เกิด INTERRUPT เข้ามาได้เมื่อยังทำไม่หมดคำสั่ง ด้วยเหตุนี้ BIT M1E จะถูก SET เป็น 0 สำหรับ Z80180 เพื่อให้ $\overline{M1}$ ถูกทำงานปกติ คือ เมื่อทำคำสั่ง RETI จะมี $\overline{M1}$ เพียงครั้งเดียว ดังรูป :-

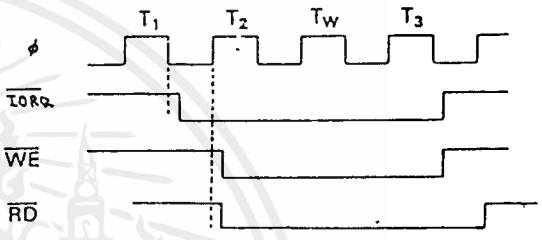


$\overline{M1TE}$ (M1 TEMPORARY ENABLE) ใช้กับการต่อ INTERFACE กับ Z80 PIO

$\overline{IOC}$ เป็น BIT ใช้ควบคุม TIMING ของ $\overline{IORQ}$ และ $\overline{RD}$ ให้เหมือน Z80 หรือ 64180 โดยถ้า BIT นี้ถูก SET เป็น 1 TIMING จะเป็นของ 64180 คือ $\overline{IORQ}$ และ $\overline{RD}$ จะ ACTIVE ที่ขอบขาลงของ T1 แต่ถ้า BIT นี้เป็น 0 TIMING จะเป็นของ Z80 คือ จะ ACTIVE ที่ขอบขาขึ้นของ T2 เพื่อให้ใช้อุปกรณ์สนับสนุนของ Z80 ได้ ระหว่าง RESET BIT นี้จะเป็น 1 ดังรูป :-



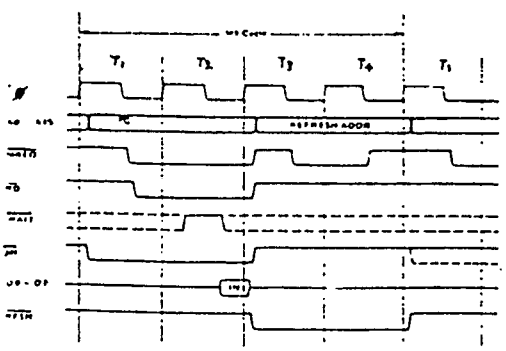
I/O Read Write Cycle When $\overline{IOC}$=0

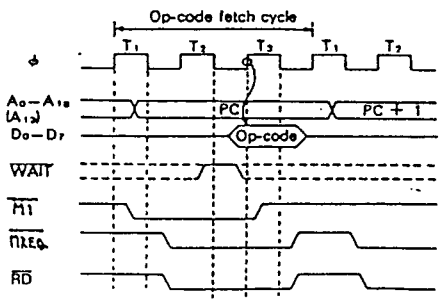


I/O Read Write Cycle When $\overline{IOC}$=1

เกี่ยวกับ TIMING

ให้ดูรายละเอียดในคู่มือฉบับภาษาอังกฤษ แต่กล่าวสรุปได้ว่า Z80180 ใช้เวลาในการทำคำสั่งใน 1 MACHINE CYCLE น้อยกว่า Z80 อยู่ 1 T STATE คือ ใช้เวลาใน 1 MACHINE CYCLE เพียง 3 T STATE ในขณะที่ Z80 ใช้ 4 T STATE จะเห็นได้ว่าในขณะที่ให้ Z80180 RUN ความถี่เดียวกันกับ Z80 CPU Z80180 ก็ยังให้ความเร็วกว่า Z80 ถึงอีก 25% แต่ในขณะเดียวกัน Z80180 ยังสามารถต่อ CLOCK สูงกว่า Z80 ได้มากกว่า 1 เท่า จึงทำให้ความเร็วในการทำงานของ Z80180 ดีกว่ามาก ดูรูปเปรียบเทียบ T STATE ของ Z80 กับ Z80180





WAIT STATE GENERATOR

Z80180 ทำงานด้วยความถี่ที่สูงขึ้นจึงอาจทำให้ MEMORY หรือ I/O ทำงานไม่ทันจึงต้องมีสัญญาณมาเป็นตัวช่วยกำหนดความพร้อมระหว่าง CPU กับอุปกรณ์ภายนอกนั่นก็คือ สัญญาณ WAIT ซึ่ง Z80 นั้นจะต้องให้อุปกรณ์ภายนอกส่งสัญญาณนี้มาให้แต่ Z80180 ยังสามารถให้โปรแกรมจำนวน WAIT STATE เพื่อเพิ่มเข้าไปในขณะที่ CPU ปฏิบัติคำสั่งหรือทำ DMA ด้วย

การโปรแกรมจะใช้ 4 BIT ของ DMA/WAIT CONTROL REGISTOR (DCNTL I/O ADDRESS 32H)

BIT 7 6 5 4

MWI1	MWI0	IWI1	IWI0	

BIT 7 , 6 MWI1 , MWI0 (MEMORY WAIT INSERTION)
จะทำการเพิ่มจาก 0-3 WAIT STATE ของการเข้าถึง MEMORY โดยการโปรแกรม

MWI1	MWI0	จำนวน WAIT STATE
0	0	0
0	1	1
1	0	2
1	1	3

BIT 5 , 4 IWI1 , IWI0 (I/O WAIT INSERTION)
จะทำการเพิ่ม WAIT STATE ให้กับ I/O ภายนอกจาก 1-6 ดังตาราง

IWI1	IWI0	I/O ภายนอก	INT0
0	0	1	2
0	1	2	4
1	0	3	5
1	1	4	6

จะเห็นว่า WAIT STATE ของ I/O มากกว่า MEMORY อยู่หนึ่ง T STATE เพราะขณะเข้าถึง I/O ปกติ WAIT STATE จะถูกเพิ่มขึ้น 1 อยู่แล้ว ดังนั้นเมื่อเพิ่ม WAIT STATE เข้าไปก็จะรวมกับที่มีอยู่ปกติ และส่วน $\overline{INTO}$ ก็เช่นเดียวกัน ขณะเกิด $\overline{INTO}$ ปกติ จะมี WAIT STATE อยู่ 2 WAIT STATE อยู่แล้ว และขณะที่ RESET BIT CONTROL WAIT STATE ทั้ง 4 จะเป็น 1 ทั้งหมด คือ อยู่ใน MODE ของ MAX WAIT STATE

ตัวอย่างเช่น เราต้องการเพิ่ม WAIT STATE ในการเข้าถึง MEMORY 2 WAIT STATE จะโปรแกรม ดังนี้ :-

```
INO   A , (32H)     ;   IN ค่าใน REGISTOR DMA/WAIT
AND   0BFH          ;   FILL เฉพาะ BIT 7 และ 6 เท่านั้น
OUTO  (32H) , A     ;   ที่ใช้ IN แล้ว AND ก็เพราะว่า REGISTOR นี้
                    ;   มีการกำหนดเกี่ยวกับ DMA ดังนั้นเราจึง FILL
                    ;   เฉพาะ BIT ที่จะต้องการโปรแกรม
```

ดูรูปขณะ WAIT STATE ปกติ

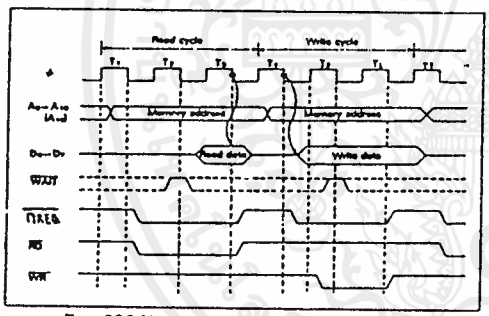


Figure 2.2.3 Memory Read/Write Timing (without wait state)

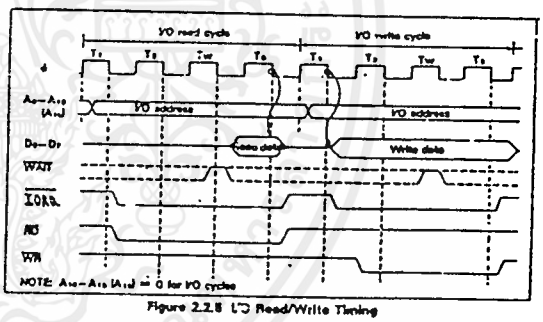


Figure 2.2.6 I/O Read/Write Timing

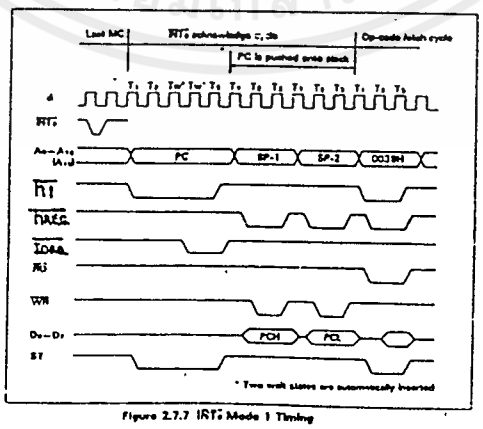


Figure 2.7.7 IRT: Mode 1 Timing

HALT และ LOW POWER MODE

มีด้วยกัน 4 MODE คือ

HALT MODE — โดยทำคำสั่ง 76H จะทำให้ CPU หยุดทำคำสั่ง แต่การทำงานต่างๆของ CPU ยังทำปกติ การออกจาก HALT โดย RESET หรือ INTERRUPT

SLEEP MODE — โดยการทำคำสั่ง SLP ซึ่ง CPU จะหยุด CLOCK ภายในทำให้ ADDRESS เป็น HIGH , DATA BUS เป็น TRISTATE , DRAM REFRESH , INTERNAL DMAC หยุดทำงานการออกจาก SLEEP MODE โดยการ RESET หรือ INTERRUPT

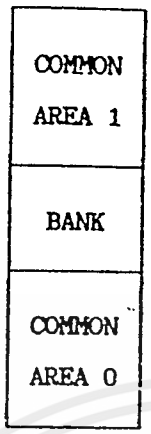
IOSTOP MODE — ใช้หยุดการทำงานของ CHIP ภายในคือ ASCI , CSI/O และ PRT โดยการ SET BIT ใน I/O CONTROL REGISTOR (ICR I/O ADDRESS 3FH) เป็น 1 และจะให้ทำงานต่อก็ RESET หรือโปรแกรมให้ BIT ใน ICR เป็น 0

SYSTEM STOP MODE — เป็นการรวมกันของ IOSTOP กับ SLEEP MODE โดยการ SET BIT ใน ICR แล้วตามด้วยคำสั่ง SLP จะทำให้ IO ภายในหยุดทำงานและ CPU หยุดทำงานเพื่อเป็นการประหยัดพลังงาน ซึ่งใน MODE นี้ CPU จะกินกระแสเพียง 7.5 MA ในขณะปกติจะกินกระแสประมาณ 35 MA เมื่อจะออกจาก SYSTEM STOP MODE ก็โดยการ RESET หรือ INTERRUPT จากภายนอก

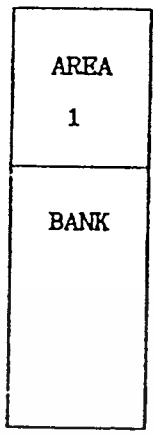
MEMORY MANAGEMENT UNIT (MMU)

ใช้เป็นตัวขยาย MEMORY จาก 64 K (LOGICAL) เป็น 1 MBYTE (PHYSICAL) โดยการแบ่ง 64 K BYTE LOGICAL (คือ ADDRESS ปกติที่ใช้เช่นเดียวกับ Z80) เป็น 3 ส่วนในการใช้งานด้วยกัน คือ COMMON AREA 0 , BANK AREA และ COMMON AREA 1 โดยการกำหนดโปรแกรมจัด MAP LOGICAL ใน REGISTOR I/O CBAR (ADDRESS 3AH) ซึ่งใน REGISTOR นี้จะถูกแบ่งเป็น 2 นิบเบิ้ล คือ 4 BIT สูง และ 4 BIT ต่ำ โดย 4 BIT สูงใช้โปรแกรมพื้นที่ของ COMMON AREA 1 และ 4 BIT ต่ำใช้โปรแกรมส่วน BANK AREA ดังนั้นการโปรแกรม REGISTOR CBAR นี้ก็จะจัด MAP ได้เป็น 2^4 คือ 4 รูปแบบ ดังรูป :-

1



2



เช่น A1> BANK > A0

CBAR = D4H

A1> BANK = A0 (0H)

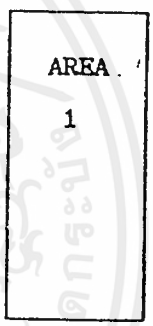
CBAR = F0 (ตอน RESET เป็นเช่นนี้

3

A1 = BANK > A0

CBAR = FFH

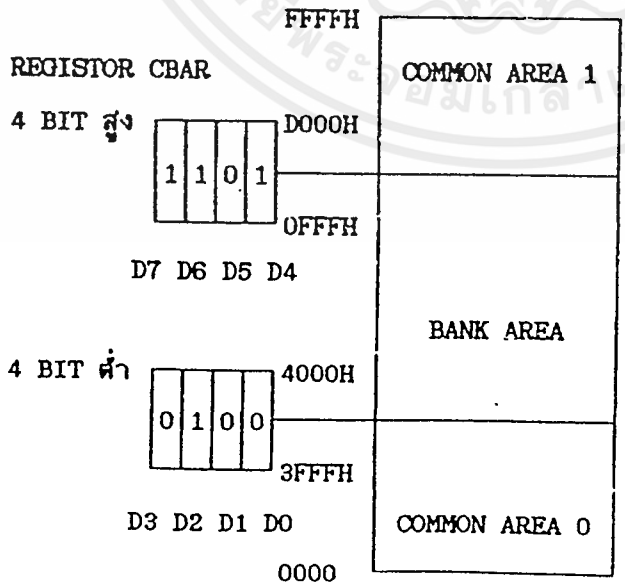
4



A1 = BANK = A0 (0H)

CBAR = 00H

MAP LOGICAL เกิดที่



จากรูปเป็นการโปรแกรม REGISTOR CBAR ให้ MAP LOGICAL เป็น COMMON AREA 0 ตั้งแต่ ADDRESS 0000-3FFFH , BANK AREA ตั้งแต่ 4000H-CFFFH และ COMMON AREA 1 ตั้งแต่ D000-FFFF ทั้งนี้เป็นไปตามค่าใน CBAR ทั้ง 2 นิบเบิ้ล เพราะนิบเบิ้ลสูงเป็นของ AREA 1 ซึ่งคือ 0DH ก็คือ AREA1 เริ่มแต่ D000H-FFFFH และนิบเบิ้ลต่ำจะเป็นจุดสิ้นสุดของ BANK AREA ซึ่ง 04H ก็คือ ถัดจาก AREA 1 เป็นต้นไปจนถึง 4000H เป็น BANK ที่เหลือจึงเป็น AREA 0 นั่นเอง

จากค่าที่โปรแกรมใน CBAR จึงทำให้โปรแกรม COMMON AREA ทั้ง 2 และ BANK ได้ตั้งแต่ 4 K BYTE ขึ้นไป เช่น ให้นิบเบิ้ลสูงของ CBAR = 0FH ก็คือ AREA 1 มีค่าตั้งแต่ F000-FFFFH (คือ 4 K อย่างต่ำนั่นเอง) และจุดที่น่าสังเกตุจากการจัด MAP ทั้ง 4 รูปแบบนั่นก็คือ COMMON 0 และ BANK สามารถมีตำแหน่งที่ทับซ้อนกันได้ (ตำแหน่งเดียวกัน) และ COMMON AREA 1 กับ BANK ก็สามารถโปรแกรมให้อยู่ที่ใดก็ได้อย่างอิสระตั้งแต่ 4 K BYTE ขึ้นไปของส่วน PHYSICAL ADDRESS (1 MBYTE โดยใช้ร่วมกับ REGISTOR อีก 2 ตัว) แต่ส่วน COMMON AREA 0 แล้วจะเป็น BASED หรือ MONITOR ของระบบนั่นเอง

จากที่กล่าวมาเรายังไม่พูดถึงการขยาย MEMORY ออกไปมากกว่า 64 K เพราะว่าการที่จะย้าย MEMORY เกินกว่า 64 K นั้น จะต้องอิงส่วนของ LOGICAL ด้วย เนื่องด้วยคำสั่งของ Z80 ไม่สามารถอ้าง MEMORY เกินนี้ได้ ดังนั้นการอ้างถึง MEMORY ทั้งหมดจึงยังเป็นส่วนของ LOGICAL แต่ข้อมูลที่ถูกกระทำจริงจะเป็นส่วนของ PHYSICAL เช่น ในคำสั่งอาจเป็นดังนี้

LD A , (8000H)

ซึ่งดูจากคำสั่งนี้จะเป็นการทำกับตำแหน่ง 8000H (สมมุติในส่วน BANK AREA)แต่เรา SET PHYSICAL AREA ไว้ที่ 10000H นั่นก็หมายความว่าการทำคำสั่งข้างบนนี้ข้อมูลจะถูกกระทำที่ ADDRESS 10000H นั่นเอง

การคิด PHYSICAL ADDRESS

1) จะกระทำในส่วนของ BANK และ COMMON AREA 1 โดยผ่านทาง REGISTOR I/O CBR และ BBR คูณด้วย 1000H แล้วนำค่าที่ได้บวกกับ LOGICAL ADDRESS ของส่วนนั้นๆ (BANK หรือ COMMON AREA 1)
2) การกระทำทั้งหมดเกิดขึ้นภายใน CPU เอง ดังนั้นการอ้าง ADDRESS ในโปรแกรมก็ยังเป็น 64 K คือตาม LOGICAL ที่กำหนดใน CBAR นั่นเอง

REGISTOR CONTROL

CBAR : COMMON/BANK AREA REGISTOR (I/O ADDRESS 3AH) ใช้กำหนดพื้นที่ของ LOGICAL ที่เป็น COMMON AREA 0 , BANK AREA และ COMMON AREA 1

BIT	7	6	5	4	3	2	1	0
	CA 3	CA 2	CA 1	CA 0	BA 3	BA 2	BA 1	BA 0

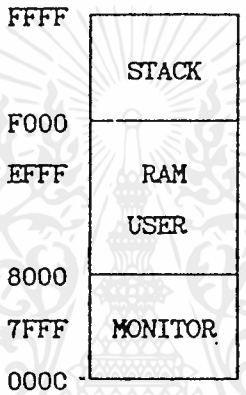
CA 3 - CA 0 เป็นตัวกำหนด ADDRESS เริ่มต้นของ COMMON AREA 1

BA 3 - BA 0 เป็นตัวกำหนดจุด ADDRESS สุดท้ายของพื้นที่ BANK AREA ที่ต่อจากจุดเริ่มต้นของ COMMON AREA 1

CBR : COMMON BASE REGISTOR (I/O ADDRESS 38H) เป็น REGISTOR I/O 8 BIT เพื่อใช้กำเนิด PHYSICAL COMMON AREA 1

BBR : BANK BASE REGISTOR (I/O ADDRESS 39H) ใช้กำเนิด PHYSICAL BANK AREA

ตัวอย่าง กำหนดให้ MONITOR ที่ 0000H-7FFFH และ RAM ใช้งานที่ 10000H โดยมีพื้นที่ STACK ที่ 18000H จากนั้นก็ต้องกำหนด MAP ใน LOGICAL โดยสมมุติให้ STACK มีเนื้อที่ 4 K นอกนั้นเป็น BANK จากนั้นก็หาค่าให้กับ BANK และ AREA 1 เช่น MAP LOGICAL กำหนดได้เป็นดังนี้ :-



หาค่าใส่ให้กับ BBR และ CBR STACK ที่ 18000 H (PHYSICAL) ที่ LOGICAL เป็น F000 H ดังนั้นค่าที่ให้กับ CBR เป็น

```
  18000
- F000
  09000
```

จากที่ทราบแล้วว่า ค่าใน CBR จะคูณด้วย 1000 H ดังนั้นในทางกลับกันเมื่อจะนำค่ามาให้กับ CBR ก็ต้องทำการหารค่าผลต่างนั้นด้วย 1000 H ก็จะได้ค่าใน CBR = 09 H ส่วน RAMUSER (BANK) ก็เช่นเดียวกัน

```
  10000
- 08000
  08000
```

ที่ BBR = 08 H

ดังนั้นการโปรแกรมจากโจทย์ตัวอย่างก็จะเป็น

CBAR = 0F8H , BBR = 08 H และ CBR = 09 H

เมื่อคำนวณกลับจะได้ CBAR นิบเบิ้ลต่ำ ADDRESS สุดท้ายของ BANK เป็น

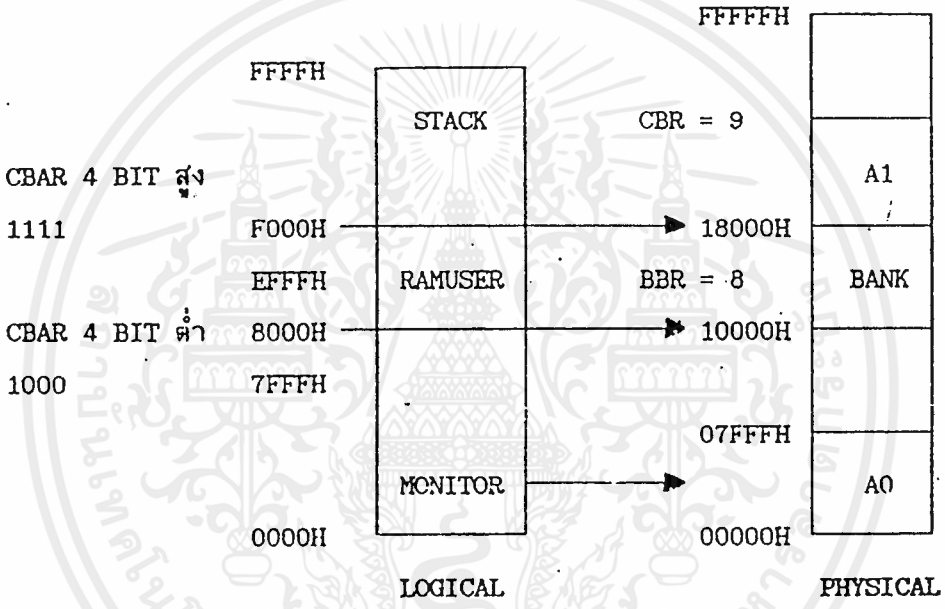
8000 + (BBR = (8) X 1000 H) = 10000 H

CBAR นิบเบิ้ลสูง ADDRESS เริ่มต้นของ AREA 1 เป็น

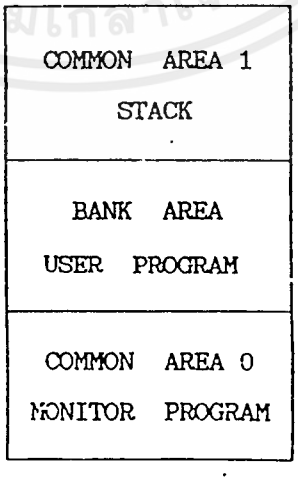
0F000 + (CBR = (9) X 1000 H) = 18000 H

การโปรแกรม

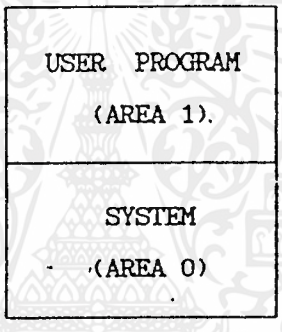
```
LD  A , 0F8H
OUTO (CBAR) , A
LD  A , 8
OUTO (BBR) , A
LD  A , 9
OUTO (CBR) , A
```



ดังนั้นเราอาจจะกล่าวได้ว่าใน LOGICAL ส่วนมากจะถูกจัดเป็น



โดยให้ส่วนของ AREA 1 และ MONITOR คงที่ส่วนของ BANK ให้ย้ายไปที่ใดๆก็ได้ใน 1 MBYTE จะเป็นการขยายพื้นที่ของการใช้งานโดยใช้เนื้อที่ของ STACK เป็นการกระทำกับตัวแปรหรือ DATA อื่นใน 64K อื่นๆ เมื่อเรามอง LOGICAL 64K ออกเป็น PAGE ๆ ใน 1 MBYTE แต่ข้อเสียในการจัดแบบนี้จะทำให้ใช้ BANK AREA ได้ไม่เต็มที่ เช่น เราต้องการใช้ RAM ถึง 32K เต็ม เช่น ให้ ROM MONITOR อยู่ที่ 0000-7FFFH และ RAM เริ่มแต่ 8000-FFFFH ซึ่งจะเห็นว่า RAM ในส่วนนี้จะต้องเป็น STACK ด้วยเมื่อเรายึด BANK ออกไปที่ PHYSICAL อื่นก็จะไม่สามารถใช้ได้ถึง 32K เช่นมี RAM ที่ตำแหน่ง 18000-1FFFFH อีกเราจะใช้ได้แค่ 24K เพราะพอเราอ้างที่ 0F000 แทนที่ข้อมูลจะถูกกระทำที่ 1F000H ก็จะมากระทำที่ 0F000H แทนตามที่กำหนด AREA 1 ไว้ใน LOGICAL เราจึงอาจแบ่ง MAP เป็นลักษณะกว้างๆดังนี้ :-



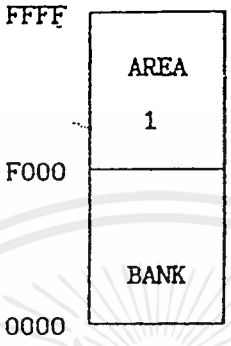
CBAR = 80 H

โดยกำหนดให้ AREA 1 เป็นส่วน USER PROGRAM ส่วน SYSTEM เป็นของ COMMON AREA 0 ดังนั้นเมื่อเราให้ AREA 1 เริ่มที่ 8000H ก็จะใช้ RAM ได้ถึง 32 K เต็ม ส่วน SYSTEM ก็คือ ของ AREA 0 ซึ่งเป็นส่วน MONITOR แต่ในส่วนนี้เราได้กำหนดไว้ถึง 32 K คือ จาก 7FFF ลงไปถึง 0000H ซึ่งใน SYSTEM เราอาจจะใส่ RAM ไว้ใน ADDRESS ช่วงนี้เพื่อเป็นเนื้อที่ของ STACK ก็จะทำให้เราย้ายเนื้อที่ของการใช้งานได้เต็ม

<u>สรุป</u>

1) ระหว่าง RESET LOGICAL ใน CBAR จะถูกกำหนดด้วยค่า 0F0H
2) ให้กำหนด MAP ADDRESS ของ LOGICAL ก่อนที่ CBAR (3AH)
3) BBR และ CBR จะเป็นตัวกำหนดตำแหน่งของข้อมูลในการใช้งานจริงในพื้นที่ 1 MBYTE (PHYSICAL ADDRESS)
4) การคิดค่า PHYSICAL ADDRESS คือ นำค่าใน BBR หรือ CBR คูณด้วย 1000H แล้วบวกด้วย LOGICAL ของพื้นที่นั้นๆ

ข้อสังเกตุ ในการจัดรูปแบบการ SET MAP ทั้ง 4 อย่าง ที่กล่าวในตอนต้น เช่น ตอน RESET CBAR = F0 จึงเป็นดังรูป :-



ถ้าระบบของเรามี MONITOR ที่ 0-7FFFH RAM ที่ 8000-FFFFH โดยมี STACK ที่ FE00H ถ้าไม่มีการเข้าไป CONTROL REGISTOR ของ MMU ทั้ง 3 ตัวนี้ระบบของเราก็ยังทำงานได้อยู่ แต่ถ้าเกิดเราโปรแกรมให้ BBR = 10 H ตอนนี้ระบบจะทำงานไม่ได้ทำไมถึงเป็นเช่นนั้น คำตอบก็คือ เราไม่ได้โปรแกรม LOGICAL ให้มี AREA 0 แต่ที่ตอนแรกใช้งานได้เพราะ BANK AREA ทับซ้อน COMAREA 0 อยู่ แต่เมื่อโปรแกรม BBR = 10 H แล้ว BANK AREA เลยกลายเป็น 10000H จึงทำให้ COMMON AREA 0 ไม่มีจึงทำให้ระบบทำงานไม่ได้

INTERRUPT

มีด้วยกัน 12 INTERRUPT แบ่งเป็น 4 INTERRUPT ภายนอก และ 8 INTERRUPT ภายใน โดยมีลำดับความสำคัญจากมากไปหาน้อย ดังนี้ TRAP (ภายใน) , (ภายนอก) NMI , $\overline{\text{INT0}}$, $\overline{\text{INT1}}$, $\overline{\text{INT2}}$, (ภายใน) TIMER 0 , TIMER 1 , DMA CHANEL 0 , DMA CHANEL 1 CLOCK SERIAL , ASCI CHANEL 0 และ ASCI CHANEL 1

REGISTOR และ FLAG ที่ใช้ควบคุมการ INTERRUPT

INTERRUPT VECTOR LOW (IL) , INTERRUPT VECTOR HIGH (I) , INTERRUPT TRAP CONTROL (ITC) และ FLAG IEF1 , IEF2 โดยที่ FLAG IEF1 จะใช้ในการ ENABLE INTERRUPT ภายในทั้งหมดยกเว้น TRAP

INTERRUPT VECTOR LOW REGISTOR (IL I/O ADDRESS 33H)

ใช้เป็น VECTOR TABLE BYTE ต่ำ ของ INTERRUPT ภายนอก $\overline{INT1}$, $\overline{INT2}$ และ INTERRUPT ภายในทั้งหมดยกเว้น " TRAP " โดย 3 BIT สูงของ IL สามารถโปรแกรมได้ แต่ 5 BIT หลังจะถูก FIX ดังรูป :-

Interrupt Source	Priority	IL			Fixed Code				
		b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
$\overline{INT_0}$	Highest	•	•	•	0	0	0	0	0
$\overline{INT_2}$		•	•	•	0	0	0	1	0
PRT channel 0		•	•	•	0	0	1	0	0
PRT channel 1		•	•	•	0	0	1	1	0
DMA channel 0		•	•	•	0	1	0	0	0
DMA channel 1		•	•	•	0	1	0	1	0
CSI/O		•	•	•	0	1	1	0	0
ASCI channel 0		•	•	•	0	1	1	1	0
ASCI channel 1	Lowest	•	•	•	1	0	0	0	0

• Programmable

ดังนั้นการ INTERRUPT ส่วนใหญ่จะเป็น MODE 2 คือนำค่าใน I และ IL หรือจากอุปกรณ์ที่ขอ INTERRUPT ในกรณี INTO มาประกอบกันเป็น ADDRESS ที่จะเก็บข้อมูลที่จะกระโดดไป เช่น I = 10 H และ IL = 40 H และใน ADDRESS 1040H มีข้อมูล 00H , 60H ตามลำดับ เมื่อเกิด INTERRUPT ขึ้น ก็จะกระโดดไปทำโปรแกรมที่ตำแหน่ง 6000H นั่นเอง

INT/TRAP CONTROL REGISTOR (ITC ADDRESS I/O 34H)

BIT	7	6	5	4	3	2	1	0
	TRAP	UFO				ITE2	ITE1	ITE0

ITE2 , 1 , 0 INTERRUPT ENABLE 2 , 1 , 0 ใช้ ENABLE และ DISABLE INTERRUPT ภายนอก ถ้าเป็น 0 จะ DISABLE แต่ BIT นี้จะไม่ทำให้เกิด INTERRUPT ขึ้นทันทีจนกว่าจะทำคำสั่ง EI ดังนั้น $\overline{INT0}$ จะต่างกับ Z80 ตรงที่มีส่วนนี้ แต่เมื่อเกิด RESET ITE0 จะถูก SET เป็น 1 โดยอัตโนมัติเพื่อให้ขึ้นกับคำสั่ง EI หรือ DI อย่างเดียว เช่น Z80 แต่ ITE1 และ ITE2 จะเป็น 0

TRAP จะเป็น 1 เมื่อทำคำสั่งที่ไม่มีใน Z80180 TRAP สามารถ RESET ภายใต้โปรแกรมควบคุมได้ แต่ไม่สามารถเขียน 1 เข้าไปได้ระหว่าง RESET จะถูก CLEAR

UFO : UNDEFIND FETCH OBJECT เมื่อ TRAP เกิดขึ้น UFO จะให้ค่าของตำแหน่งที่ผิดในคำสั่งนั้นไว้ใน STACK เนื่องจาก TRAP อาจเกิดขึ้นจาก OPCODE 2 หรือ 3 BYTE UFO จะปรับค่า PC ให้ คือ ถ้าเป็นคำสั่ง OPCODE 2 BYTE UFO จะเป็น 0 และจะทำให้ PC ของคำสั่งถัดไปจากคำสั่งที่ไม่ใช่ของ Z80180 ถูกลดลง 1 แต่ถ้า UFO = 1 คำสั่งที่ผิดจะมี OPCODE 3 BYTE และ PC จะถูกลดลง 2 ตำแหน่ง และค่า PC นี้จะถูกเก็บไว้ใน STACK เช่น

2000 ED 99

2002 ← PC ชี้คำสั่งถัดไป

เมื่อ CPU RUN มาพบข้อมูลที่ตำแหน่ง 2000 ก็จะเกิด INTERRUPT TRAP ขึ้น และรู้ด้วยว่าเป็นคำสั่ง 2 BYTE และ PC ก็ชี้คำสั่งถัดไปคือ ADDRESS 2002 แต่ FLAG UFO จะถูกทำให้เป็น 0 เพื่อปรับค่า PC นั้นด้วยการลดลง 1 เช่น ADDRESS 2001 ซึ่งก็คือตำแหน่งข้อมูลที่ผิดนั่นเอง

TRAP INTERRUPT เป็นเหมือน NMI คือ ไม่สามารถหยุดได้เมื่อเกิดกระทำคำสั่งผิดขึ้นซึ่งเป็นตัวช่วยให้เกิดความน่าเชื่อถือทางด้าน SOFTWARE และอาจใช้เพิ่มคำสั่งได้อีกด้วย BIT TRAP ใน ITC จะถูก SET เป็น 1 และ UFO จะ SET หรือไม่ SET ขึ้นอยู่กับว่าเป็นคำสั่ง 2 หรือ 3 BYTE และ FLAG UFO นี้ก็จะไปปรับ PC ให้ถูกต้อง และเก็บไว้ใน STACK แล้วกระโดดไป RUN ที่ ADDRESS 0000 H

DYNAMIC RAM REFRESH CONTROL

Z80180 ให้ ADDRESS A0-A7 สำหรับ DYNAMIC RAM และยังสามารถโปรแกรมเวลาในการ REFRESH โดยการโปรแกรมที่ RCR

REFRESH CONTROL REGISTOR (RCR ADDRESS I/O 36 H)

BIT	7	6	5	4	3	2	1	0
	REFE	REFW	-	-	-	-	CYC1	CYC0

REFE : REFRESH ENABLE เมื่อเป็น 0 จะ DISABLE แต่ถ้าเป็น 1 จะให้สัญญาณ REFRESH ระหว่าง RESET จะเป็น 1

REFW : REFRESH WAIT เป็น 0 จะให้สัญญาณ REFRESH ทุกๆ 2 CLOCK ถ้าเป็น 1 จะเพิ่ม REFRESH WAIT เข้าอีก 1 ระหว่าง RESET จะเป็น 1

CYC1 , CYC0 : CYCLE INTERVAL ใช้กำหนดช่วงเวลาในการ REFRESH เช่นกรณี DYNAMIC RAM จะต้อง REFRESH 128 ครั้ง ทุกๆ 2 ms (หรือ 256 ครั้ง ทุกๆ 4 ms) เพราะฉะนั้นสัญญาณ REFRESH แต่ละครั้งจะต้องไม่น้อยกว่าหรือเท่ากับ 15.625 us จากตาราง ค่าที่ขีดเส้นใต้เป็นค่าโปรแกรมที่เหมาะสมกับ CLOCK ที่ใช้ในระบบ

CYC1	CYC0	Insertion interval	Time interval			
			ϕ: 8 MHz	6 MHz	4 MHz	2.5 MHz
0	0	10 states	1.25 μs	1.66 μs	2.5 μs	4.0 μs
0	1	20 states	2.5 μs	3.3 μs	5.0 μs	8.0 μs
1	0	40 states	5.0 μs	6.6 μs	10.0 μs	16.0 μs
1	1	80 states	10.0 μs	13.3 μs	20.0 μs	32.0 μs

DMA CONTROLLER (DMAC)

มีด้วยกัน 2 CHANEL เพื่อเป็นการเพิ่มความเร็วในการ TRANFER ข้อมูล โดยการกระทำไม่ต้องผ่าน CPU โดยมีความสามารถดังนี้

MEMORY ADDRESS SPACE โดยสามารถกำหนดตำแหน่ง SOURCE และ DESTINATION ที่ใดก็ได้ใน 1024 K BYTE

I/O ADDRESS SPACE กำหนดที่ใดก็ได้ใน 64 K BYTE ทั้ง SOURCE และ DESTINATION

TRANSFER LENGTH ใช้เป็น COUNTER ในการ TRANSFER ได้เป็น BLOCK ๆละ 64 K BYTE

$\overline{\text{DREQ}}$ เป็นขา INPUT จะตรวจจับที่ระดับหรือขอบของสัญญาณ

$\overline{\text{TEND}}$ เป็นขา OUTPUT เพื่อบอกกับอุปกรณ์ภายนอกว่าทำ DMA หมด BLOCK แล้ว

TRANSFER RATE การ TRANSFERแต่ละครั้งจะเกิดทุกๆ 6 CLOCK และ WAIT STATE สามารถเพิ่มเข้าไปใน DMA ได้ สำหรับ MEMORY หรือ I/O ที่ทำงานช้าที่ระบบ SYSTEM CLOCK (Ø) = 6 MHz อัตราการ TRANSFER จะสูงถึง 1 M BYTE ใน 1 วินาที (ไม่มี WAIT STATE)

ความสามารถของแต่ละ CHANEL

CHANEL 0 สามารถ TRANSFER MEMORY ↔ MEMORY , MEMORY ↔ I/O , MEMORY ↔ MEMORY I/O MAP และสามารถให้ ADDRESS ในการ TRANSFER เพิ่ม , ลด หรือให้คงที่ได้ การ TRANSFER จะให้เป็นแบบ CYCLE STEAL (ขโมยเวลาเป็นช่วง) คือ เมื่อ TRANSFER ครบ 1 หรือ 2 BYTE ก็จะคืน BUS ให้ UP, จนอุปกรณ์พร้อมก็จะ TRANSFER ข้อมูลต่อ ทำอย่างนี้สลับกันไปจนหมด BLOCK ใช้สำหรับอุปกรณ์ที่ทำงานช้า และ BURST (TRANSFER แบบต่อเนื่อง) คือ ทำจนจบ BLOCK จึงจะคืน BUS ให้ UP

CHANEL 1 จะใช้กับ MEMORY ↔ I/O โดย MEMORY ADDRESS เพิ่มหรือลดได้

extend instruction to condition

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	No.of Byte	No.of M Machine	No.of T State
SLP	Sleep	▪	▪	▪	▪	▪	▪	2	2	8
MLT	uuHr←uuLr→uur	▪	▪	▪	▪	▪	▪	2	13	17
INO g,(m)	(00m)→gr g=110:only the flags will change. m→A0~A7 00→A8~A15	↑	↑	R	P	R	▪	3	4	12
OUTO (m),g	gr→(00m) m→A0~A7 00→A8~A15	▪	▪	▪	▪	▪	▪	3	5	13
OTIM	(HL)→(00C) HL+1→HL C+1→C B-1→B C→A0~A7 00→A8~A15	↑	↑[5]	↑	P	↑[6]	↑	2	6	14
OTIMR	(HL)→(00C) HL+1→HL C+1→C B-1→B Repeat 9 until B=0 C→A0~A7 00→A8~A15	R	S	R	S	↑[6]	R	2	8	16 (if B=0) 14 (if B=0)
OTDM	(HL)→(00C) HL-1→HL C-1→C B-1→B C→A0~A7 00→A8~A15	↑	↑[5]	↑	P	↑[6]	↑	2	6	14
OTDMR	(HL)→(00C) HL-1→HL C-1→C B-1→B Repeat 9 until B=0 C→A0~A7 00→A8~A15	R	S	R	S	↑[6]	R	2	8	16 (if B=0) 14 (if B=0)
TSTIO m	(00C)∧m C→A0~A7 00→A8~A15	↑	↑	S	P	R	R	3	4	12
TST g	Ar∧gr	↑	↑	S	P	R	R	2	3	7
TST (HL)	Ar∧(HL)	↑	↑	S	P	R	R	2	4	10
TST m	Ar∧m	↑	↑	S	P	R	R	2	3	9

g = reg ABCDEHL

uu = BC,DE,HL,SP

m = data 8 bit

flag

▪ : not effect

↑ : effect

S : set to 1

R : reset to 0

P : parity

[5] ↑ Z=1:B-1=0
Z=0:B-1=0

[6] ↑ N=1:MSB of data=1
N=0:MSB of data=0

```
000002 0000                    ;************************
000003 0000                    ;DEMO SOFT-WARE Z80180 *
000004 0000                    ;************************
000005 0000                    ;
000006 0000                    ; ROM monitor program 00000-07FFFH
000007 0000                    ; RAM for use 18000,1EFFFH
000008 0000                    ; RAM stack for user&system 1F000-1FFFFH
000009 0000
000010 0000                    ; example flash LED at 8255 port 0C082H
000011 0000                    ; and  receive  the key board OF (serial port IBM PC)
000012 0000                    ; to Z80180 ASCI chanel 1 interrupt and send charecter
000013 0000                    ; from keyboard to display (ehco).
000014 0000
000015 0000                    ; timer reload chanel 0 interrup every 100 us for flash LED
000016 0000
000017 0000                    ;         main program is sleep mode
000018 0000                    ;
000019 0000
000020 0000                                .ORG 0
000021 0000                    ;
000022 0000                    ;*** REGTER I/O Z80180 ***
000023 0000                    ;
000024 0007                                .EQU TDR1,7         ;ASCI CHANEL1 TX
000025 0009                                .EQU RDR1,9         ;RX
000026 0005                                .EQU STAT1,5        ;STATUS
000027 0001                                .EQU CNTLA1,1       ;DATA FORMAT
000028 0003                                .EQU CNTLB1,3       ;BAUD RATE
000029 0000                    ;
000030 000C                                .EQU TMDR0L,0CH     ;TIMER0 BYTE LOW
000031 000D                                .EQU TMDR0H,0DH     ;TIMER0 BYTE HIGH
000032 000E                                .EQU RLDR0L,0EH     ;RELOAD BYTE LOW
000033 000F                                .EQU RLDR0H,0FH     ;RELOAD BYTE HIGH
000034 0010                                .EQU TCR,10H        ;TIMER CONTROL
000035 0000                    ;
000036 0033                                .EQU IL,33H         ;VECTOR LOW
000037 003E                                .EQU OMCR,3EH       ;OPERATION CONTROL
000038 0000                    ;
000039 003A                                .EQU CBAR,3AH       ;SET LOGICAL
000040 0039                                .EQU BBR,39H        ;BANK BASE
000041 0038                                .EQU CBR,38H        ;COMMON AREA 1
000042 0000                    ;
000043 FF00                                .EQU STACK,0FF00H
000044 0080                                .EQU VECLOW,80H     ;VECTOR LOW
000045 000A                                .EQU LF,0AH         ;LINE FEED
000046 000D                                .EQU CR,0DH         ;CARRIER
000047 0007                                .EQU BEL,7          ;BELL
000048 0000                                .EQU TIMEL,0        ;TIMER 100 MS
000049 0078                                .EQU TIMEH,78H      ;
000050 C083                                .EQU PCTRL,0C083H   ;PORT CONTROL 8255
000051 C082                                .EQU PDATA,0C082H   ;PORT C OF 8255
000052 0000                    ;
000053 0000                    ;************************
000054 0000                    ;     POWER UP DELAY    *
000055 0000                    ;************************
000056 0000                    ;
000057 0000 AF                 START:      XOR  A              ;POWER UP
000058 0001 00                 START1:     NOP
000059 0002 3D                             DEC  A
000060 0003 20FC                           JR   NZ,START1
000061 0005 C39400                         JP   INIT
000062 0008                    ;
000063 0080                                .ORG VECLOW
000064 0080                    ;
000065 0080                    ;*** PRIORITY INTERRUPT ***
000066 0080                    ;
000067 0080 9200               INT1:       .DRW EMPTY          ;EMPTY FOR NOT USE IN PROGRAM
000068 0082 9200               INT2:       .DRW EMPTY
000069 0084 1C01               PRT0:       .DRW PRTINT         ;TIMER INT CHANEL0
000070 0086 9200               PRT1:       .DRW EMPTY
000071 0088 9200               DMA0:       .DRW EMPTY
000072 008A 9200               DMA1:       .DRW EMPTY
000073 008C 9200               CSIO:       .DRW EMPTY
000074 008E 9200               ASCI0:      .DRW EMPTY
000075 0090 0401               ASCI1:      .DRW TRXMIT         ;ASCI CHANEL 1 INT
```

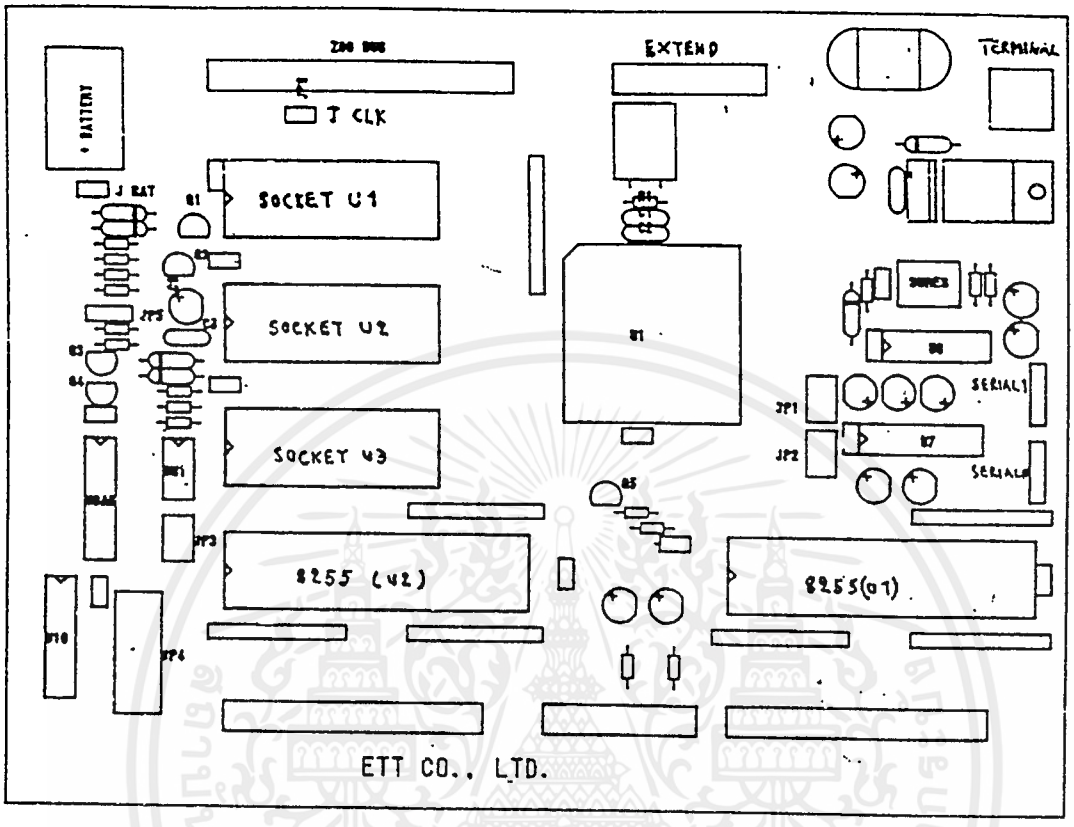
```
000076 0092                     ;
000077 0092 ED76        EMPTY:  SLP             ;DUMY
000078 0094                     ;
000079 0094                     ;************************
000080 0094                     ;   INITIAL PARAMETER  *
000081 0094                     ;************************
000082 0094                     ;
000083 0094 3E00        INIT:   LD   A,0        ;for Z80180
000084 0096 ED393E              OUT0 (OMCR),A
000085 0099 3EF8                LD   A,0F8H     ;logical address
000086 009B ED393A              OUT0 (CBAR),A
000087 009E 3E10                LD   A,10H
000088 00A0 ED3939              OUT0 (BBR),A    ;psysical bank 18000H
000089 00A3 ED3938              OUT0 (CBR),A    ;psyical common areal 1F000H
000090 00A6                     ;
000091 00A6 3100FF              LD   SP,STACK   ;load stack for call program
000092 00A9 3E80                LD   A,VECLOW   ;set low vector
000093 00AB ED3933              OUT0 (IL),A
000094 00AE CDDE00              CALL ASCSET     ;set serial port 1
000095 00B1 CDEE00              CALL SETPRT     ;set timer chanel 0
000096 00B4 213601              LD   HL,TABLE
000097 00B7 CDC800              CALL PRINT      ;display title
000098 00BA                     ;
000099 00BA 0183C0              LD   BC,PCTRL   ;set 8255. A,B,C (output)
000100 00BD 3E80                LD   A,80H
000101 00BF ED79                OUT  (C),A
000102 00C1 2E00                LD   L,0        ;data LED flash
000103 00C3 FB                  EI              ;enabel INT
000104 00C4                     ;
000105 00C4                     ;************************
000106 00C4                     ;     MAIN PROGRAM      *
000107 00C4                     ;************************
000108 00C4                     ;
000109 00C4 ED76        MAIN:   SLP             ;sleep
000110 00C6 18FC                JR   MAIN
000111 00C8                     ;
000112 00C8                     ;*** PRINT TO CONSOLE ***
000113 00C8                     ;
000114 00C8 7E          PRINT:  LD   A,(HL)     ;load charecter
000115 00C9 CDD300      PRINT0: CALL CONOUT     ;send to display
000116 00CC 23                  INC  HL
000117 00CD 7E                  LD   A,(HL)
000118 00CE FE00                CP   0
000119 00D0 20F7                JR   NZ,PRINT0  ;end data ?
000120 00D2 C9                  RET             ;yes
000121 00D3                     ;
000122 00D3                     ;*** SEND 1 CHARECTER TO CONSOLE ***
000123 00D3                     ;
000124 00D3 ED1805      CONOUT: IN0  E,(STAT1)  ;check flag send
000125 00D6 CB4B                BIT  1,E
000126 00D8 28F9                JR   Z,CONOUT   ;flag TDRE ASCI1 =1 ?
000127 00DA ED3907              OUT0 (TDR1),A   ;yes, send to display
000128 00DD C9                  RET
000129 00DE                     ;
000130 00DE                     ;*** SET ASCI CHANEL1 TX,RX,8,N,1,9600 AT X'TAL 12.488 ***
000131 00DE                     ;
000132 00DE 3E64        ASCSET: LD   A,64H
000133 00E0 ED3901              OUT0 (CNTLA1),A ;TX,RX,8BIT,1 STOP
000134 00E3 3E02                LD   A,2        ;9600 BAUD AT X'TAL 12.488
000135 00E5 ED3903              OUT0 (CNTLB1),A
000136 00E8 3E08                LD   A,8        ;RIE enable to keyboard
000137 00EA ED3905              OUT0 (STAT1),A
000138 00ED C9                  RET
000139 00EE                     ;
000140 00EE                     ; *** SET TIMER COUNTER CHANEL 0 ***
000141 00EE                     ;
000142 00EE 3E00        SETPRT: LD   A,TIMEL
000143 00F0 ED390C              OUT0 (TMDR0L),A ;TIMER byte low
000144 00F3 ED390E              OUT0 (RLDR0L),A ;RELOAD byte low
000145 00F6 3E78                LD   A,TIMEH
000146 00F8 ED390D              OUT0 (TMDR0H),A ;TIMER byte high
000147 00FB ED390F              OUT0 (RLDR0H),A ;RELOAD byte high
000148 00FE 3E11                LD   A,11H
000149 0100 ED3910              OUT0 (TCR),A    ;enable timer flag INT&
000150 0103 C9                  RET
```

```
000151 0104                    ;
000152 0104                    ;*** ASCI INTERRUPT EHCO TO CONSOLE ***
000153 0104                    ;
000154 0104 F3                 TRXMIT:     DI                  ;disable interrupt
000155 0105 F5                             PUSH AF
000156 0106 ED3809                         IN0  A,(RDR1)       ;clear RDRF & read keyboard
000157 0109 CDD300                         CALL CONOUT         ;EHCO
000158 010C FE0D                           CP   CR             ;check enter
000159 010E 3E0A                           LD   A,LF           ;line feed
000160 0110 CCD300                         CALL Z,CONOUT       ;y,out line feed
000161 0113 3E07                           LD   A,BEL          ;bell
000162 0115 CDD300                         CALL CONOUT
000163 0118 F1                             POP  AF
000164 0119 FB                             EI                  ;enable
000165 011A ED4D                           RETI
000166 011C                    ;
000167 011C                    ;*** TIMER CHANEL1 INTERRUPT 100 ms ***
000168 011C                    ;
000169 011C F3                 PRTINT:     DI
000170 011D F5                             PUSH AF             ;save reg
000171 011E C5                             PUSH BC
000172 011F                    ;
000173 011F ED3810             PRTINT1:    IN0  A,(TCR)        ;TIF flag enable ?
000174 0122 CB77                           BIT  6,A
000175 0124 28F9                           JR   Z,PRTINT1
000176 0126 ED380C                         IN0  A,(TMDR0L)     ;yes,clear TIF0
000177 0129                    ;
000178 0129 0182C0                         LD   BC,PDATA
000179 012C ED69                           OUT  (C),L          ;OUT DATA
000180 012E 7D                             LD   A,L
000181 012F 2F                             CPL                 ;toggle data flash LED
000182 0130 6F                             LD   L,A
000183 0131                    ;
000184 0131 C1                             POP  BC             ;restroe reg
000185 0132 F1                             POP  AF
000186 0133 FB                             EI
000187 0134 ED4D                           RETI
000188 0136                    ;
000189 0136 44454D4F20434F4E   TABLE:      .DB  "DEMO CONTROL Z80180",CR,LF,0
       013E 54524F4C205A3830
       0146 3138300D0A00
000190 014C                                .END
```

บทที่ 6

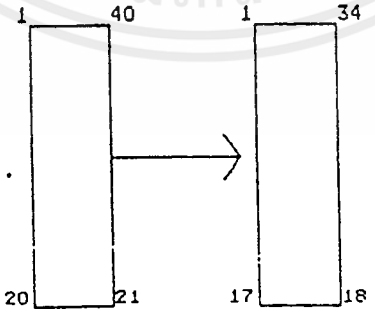
CP-180 CONTROL PACK

ภาพแสดงลักษณะ CP-180



จุดที่ SCREEN ผิดบน BOARD

1) I/O COUNECTOR 1 และ 2 ต้องเป็น 34P
และที่ SCREEN ใน PRINT CONNECTOR I/O 1,2 จะสลับกัน

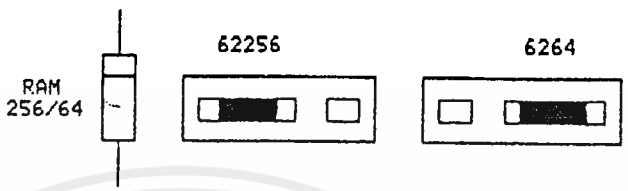


2) LED ของ 8255 จะสลับกับความเป็นจริงคือ ที่ SCREEN PC71 เป็น PC72 และ PC72 เป็น PC71

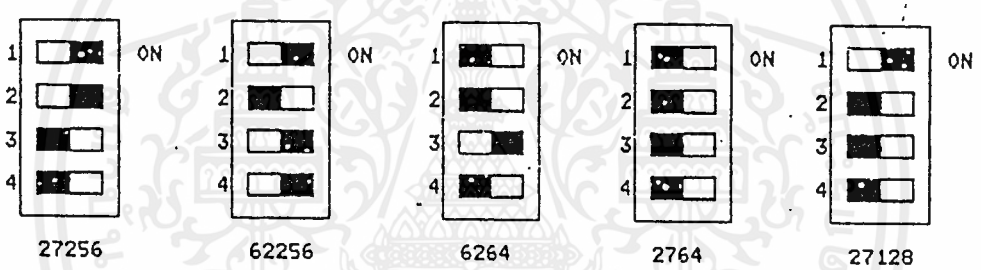
ข้อกำหนดบน BOARD

SOCKET 1 ใช้กับ ROM หรือ EPROM 27256 อย่างเดียว

SOCKET 2 ใช้กับ RAM 6264 (8K) หรือ 62256 (32K) โดยการเลือก JUMPER (JP5)



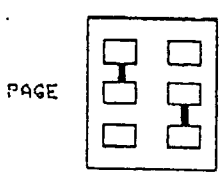
SOCKET 3 สามารถใส่ IC ได้หลายเบอร์ โดยการ SET DIP SW 4 P ดังรูป



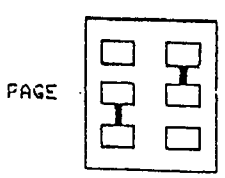
JP SP อยู่ใกล้กับ Q5 และ LED PC72 ซึ่งใช้ต่อกับลำโพง

JP RES ใกล้กับ SWITCH RESET สำหรับต่อ SWITCH RESET ไว้ที่อื่น

JP PAGE (JP 3) ใกล้ DIP SW 4 PIN ใช้สำหรับเลือก PAGE ของ PORT ว่าให้อยู่ใน DECODE แบบ 256 PORT หรือ 64 K PORT โดยมีรูปแบบดังนี้ :-



8255 ทั้ง 2 ตัวบน BOARD จะอ้างที่ 0C0XXH

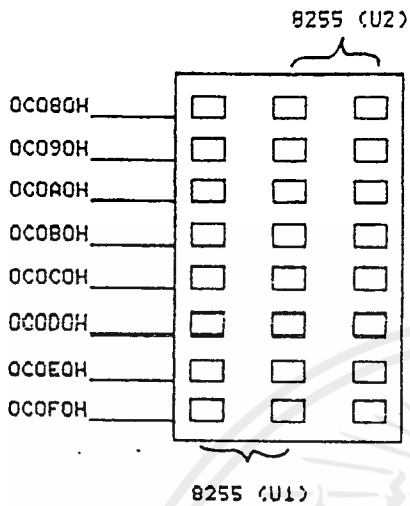


8255 ทั้ง 2 ตัวบน BOARD จะอ้างที่ XCH - XFH

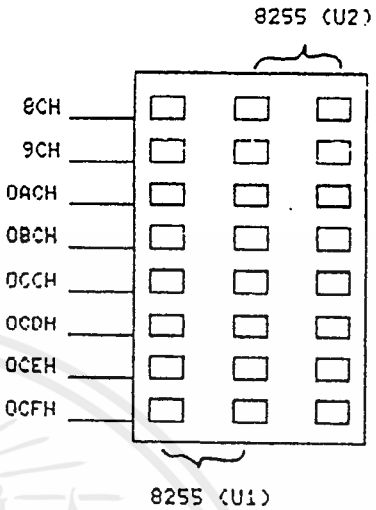
ซึ่งค่า X นั้นยังขึ้นอยู่กับการเลือก JUMPER PORT อีกครั้งหนึ่ง

JP PORT (JP 4) ใช้เลือก NUMBER PORT โดยสามารถเลือกได้ดังรูป

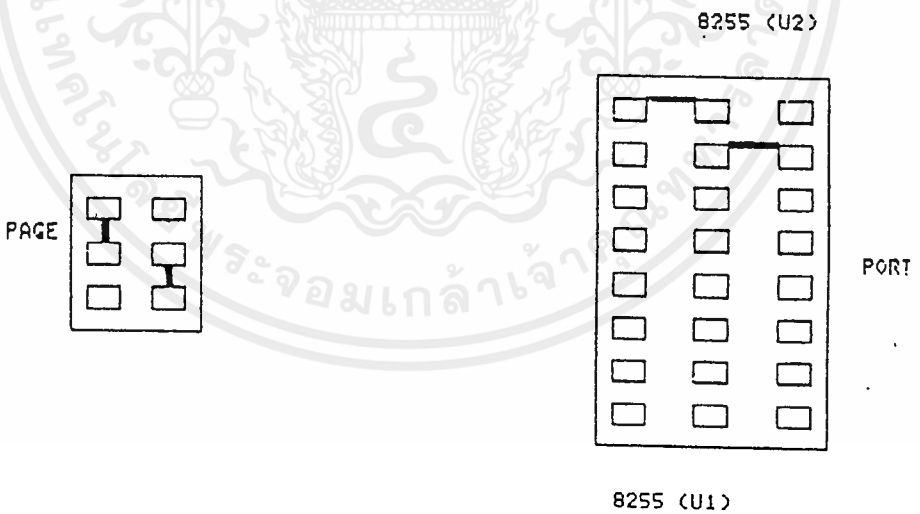
เมื่อเลือก PAGE เป็น 0C0XXH



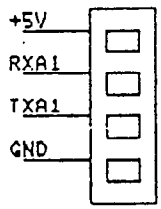
เมื่อเลือก PAGE เป็น XCH – XFH



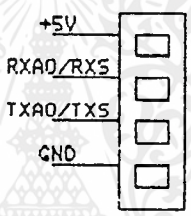
ซึ่ง BOARD สำเร็จที่ทำขึ้น จะถูกกำหนดเป็น 0C080H ของ 8255 (U1) และ 0C090H ของ 8255 (2) ซึ่งเป็นของ PRINTER PORT ด้วย ซึ่งผู้ใช้สามารถเปลี่ยนแปลงได้ แต่เมื่อใช้กับ SOFT WARE DEBUG ต้องกำหนดตามนี้



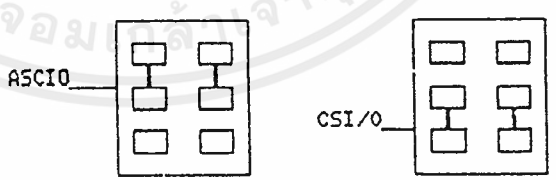
SERIAL 1 เป็น CONNECTOR SERIAL PORT ASCI CHANEL 1 โดยขาสัญญาณจะเป็นดังรูป :-



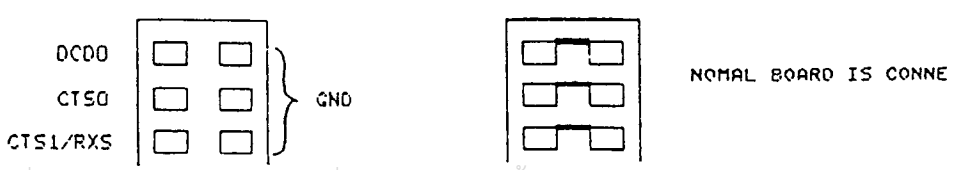
SERIAL 0 เป็น CONNECTOR SERIAL PORT ของ ASCI CHANEL 0 และยังเลือกได้ว่าให้เป็นขั้วต่อของ CLOCK SERIAL I/O ได้ โดยการ SET JUMPER ที่ ASCI/CSIO



JP ASCI/CSIO (JP1) ใช้สำหรับเลือกว่าต้องการใช้ SERIAL PORT CHANEL 0 หรือ CLOCK SERIAL I/O โดย SET ดังรูป และเมื่อใช้ CSI/O ต้องนำ JUMPER CTS1/RXS ใน JP MODEM ออกด้วย



JP MODEM (JP2) ใช้สำหรับเลือกว่าต้องการนำสัญญาณ MODEM ไปใช้หรือไม่สำหรับการต่อ SERIAL PORT ใช้เองในกรณีสัญญาณคราปกติบน BOARD จะถูก JUMP ไว้ให้



JP BAT — อยู่ใกล้กับ BATTERY (J BAT) โดยเมื่อใส่ BATTERY แล้ว JUMP ที่ JUMPER นี้ จะเป็นการ BACK UP RAM ของ SOCKET U2

JP CLK — เป็น JUMPER เลือก CLOCK ว่าให้ CLOCK จาก CPU ออกไปที่ BUS Z80 40 PIN ประโยชน์ เมื่อต้องการนำ BOARD ไปแทนระบบ Z80 เดิม เพื่อไม่ให้ CLOCK ชนกัน

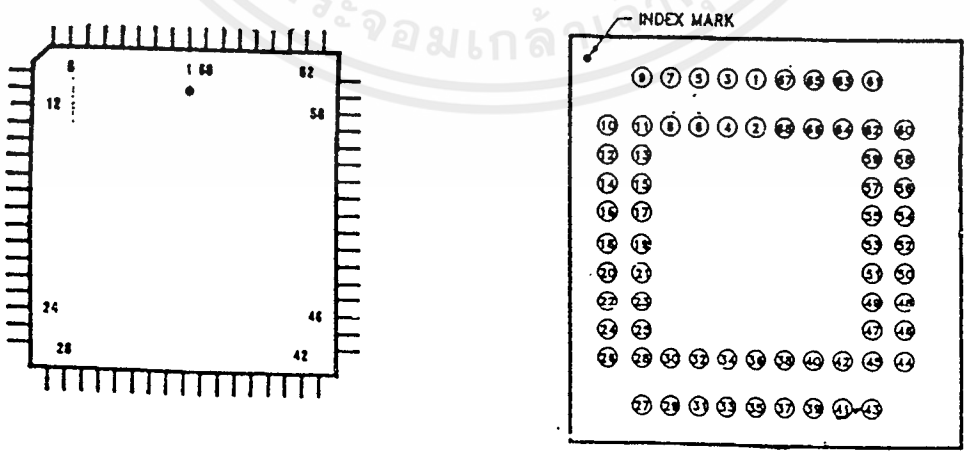
TERMINAL 2 P — สำหรับต่อ SUPPLY DC 5 V ให้กับ BOARD โดยตรง

CONNECTOR Z80 BUS — สำหรับ INTERFACE กับอุปกรณ์ภายนอก

CONNECTOR EXTEND — เป็นส่วนขาสัญญาณที่เพิ่มจาก Z80 สำหรับ INTERFACE กับอุปกรณ์ภายนอก

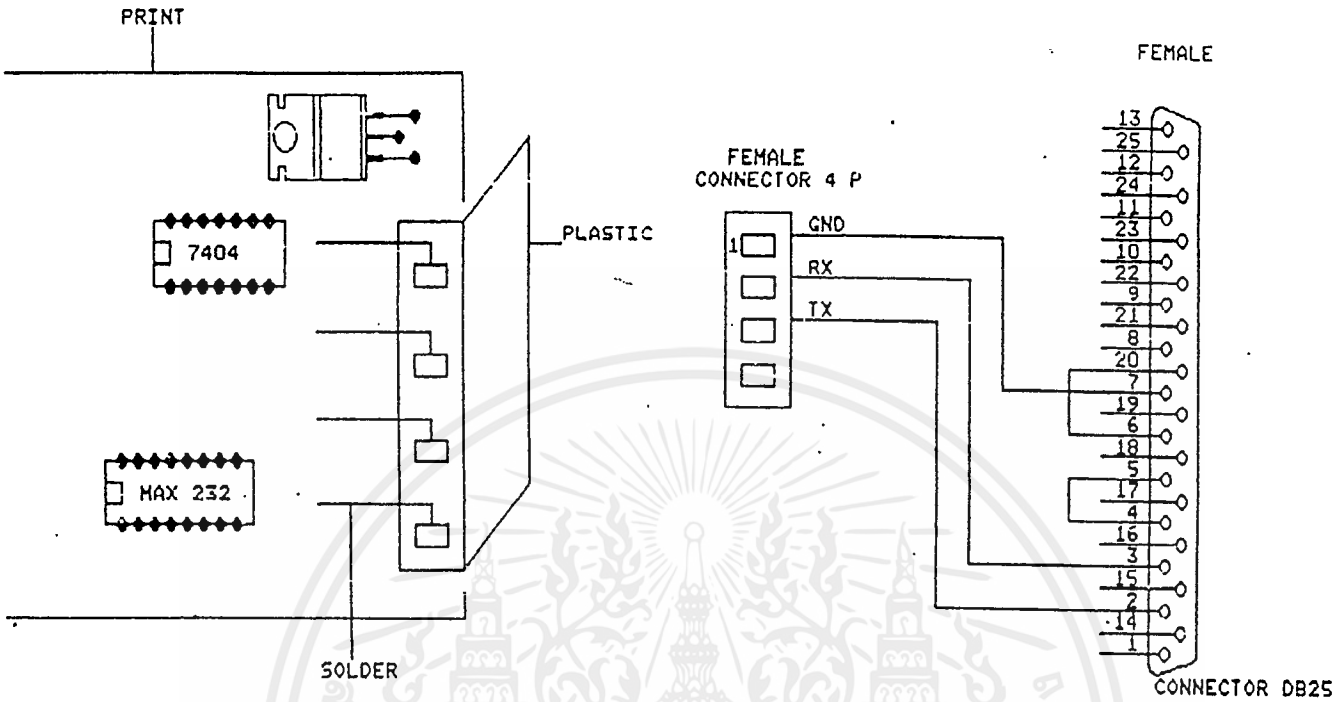
<u>คำแนะนำในการประกอบ</u>

1) ควรใส่อุปกรณ์ตามลำดับ คือ ตัวต้านทาน , ไดโอด , ตัวเก็บประจุ , SOCKET
2) ตัวเก็บประจุและไดโอดต้องใส่ให้ถูกขั้วรวมทั้ง TRANSISTOR ด้วย
3) LED ขายาว คือ A ขาสั้น คือ K
4) การใส่ SOCKET 68 PIN PLCC ให้ดูรอยบากริมซ้ายจะเป็นตัวแสดงว่าเป็นด้านบนที่จะใส่ IC ด้านขา 1 และเมื่อจะใส่ IC Z80180 ก็ให้ดูรอยบากเช่นเดียวกัน และใส่ให้ตรงกับ SOCKET โดยตัว IC จะมีจุดวงกลมตรงกลางแสดงถึงตำแหน่งขา 1 ของ IC ดังรูป :-



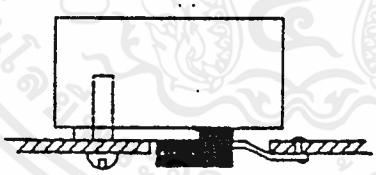
การนับขา SOCKET 68P PLCC (TOP VIEW)

5) การใส่ขั้ว CONNECTOR 4 PIN ของ SERIAL PORT ทั้ง 2 CHANEL ต้องหันด้านที่มี พลาสติก ให้อยู่ชิดแผ่น PRINT และด้านที่เป็น ขั้วทองแดง ให้หันเข้ามาด้านใน และการต่อสาย SERIAL PORT กับ PC เป็นดังรูป



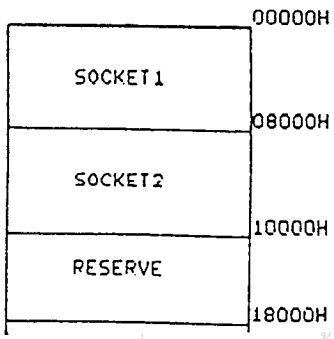
5) การบัดกรีให้กระทำเฉพาะด้านล่างเพียงด้านเดียวก็พอ เพราะ PRINT เป็นแบบ PLATE THROUGH HOLES

6) การติดตั้งและบัดกรี RECULATOR IC ให้ดูดังรูป :-



7) เมื่อทุกอย่างเรียบร้อยแล้วจึงค่อยใส่ IC ลงใน SOCKET ระวังอย่าให้ผิดเบอร์โดยเฉพาะ Z80180 ให้ดูขาในการใส่ให้ดี

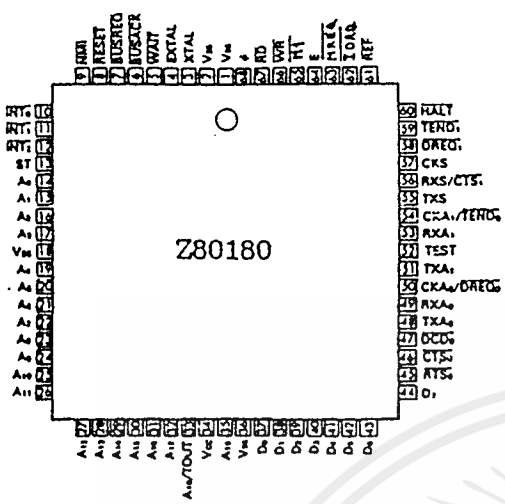
รายละเอียดการจัดหน่วยความจำ

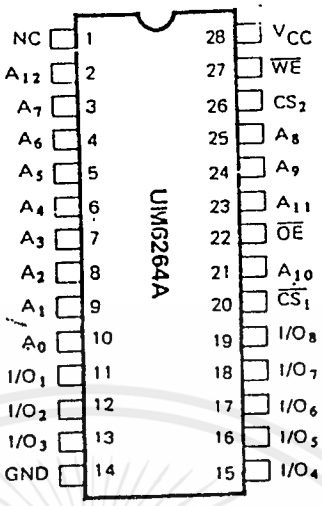


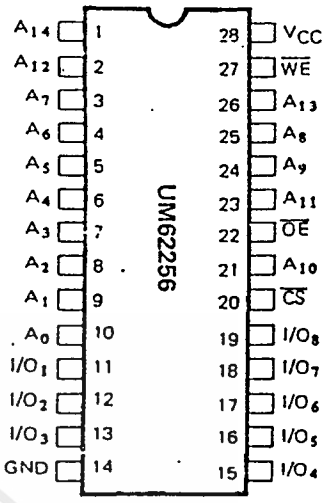
Specification

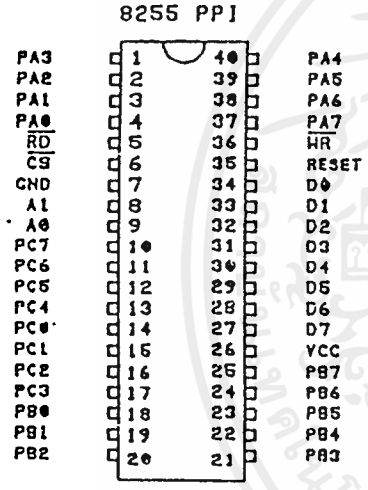
CPU	Z80180
Memory	27256 Rom (32 Kbyte at 00000H) 6264/62256 (8 Kbyte on board,expand 32 kbyte) & Back up 27256/27128/2764/6264/62256 Socket expand (Max 32 K byte)
Port	8255 I/O Port 48 Bit
Printer Port	1 Port
Serial Port	2 Chanel RS 232 & 1 Chanel can select clock serial I/O
Clock Rate	6.144 Mhz
Power supply	Comsumption 5V DC & Terminal 5V DC Main Input 9-12 V DC
Connector	1 40-Pin Expansion Header-Strip (Z80 Pin) 1 20-Pin Extend (Z80180 Pin) Header-Strip for (Modem,DMA,Expand memory 1Mbyte) 2 34-Pin Peripheral Header-Strip (8255 Port) 1 24-Pin Seclect Decode port 1 6-Pin Select Page of Port 1 6-Pin Select Modem control 1 2-Pin for battery back up 1 2-Pin Clock Select for connector old Z80 system 1 2-Pin Jumper SW reset 1 2-Pin Speaker
LED	1 Power Red LED 2 PC7 Port 8255*2 Orange LED 1 Halt Green LED
PCB Size	13 * 17.5 cm

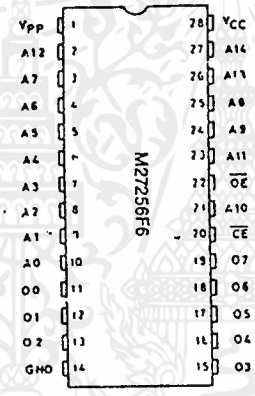
รายละเอียดขา IC

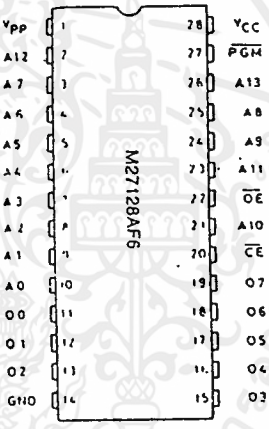


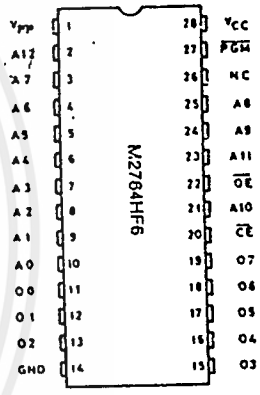


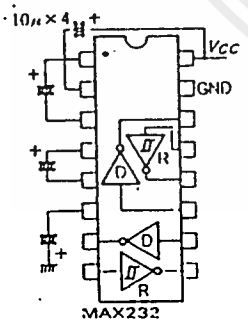


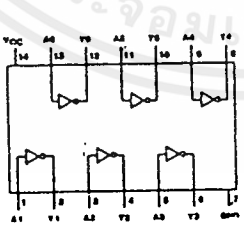


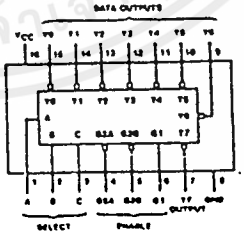


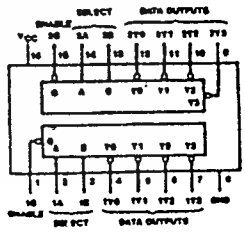






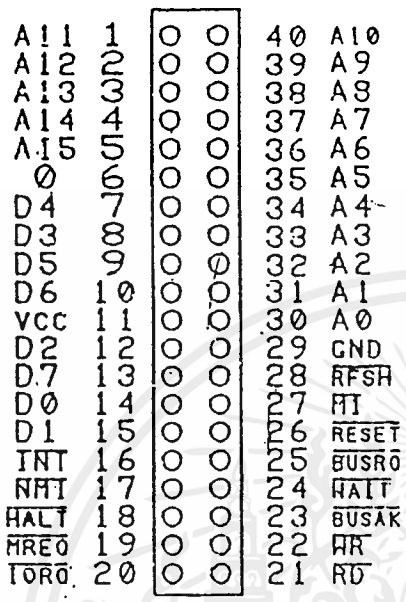




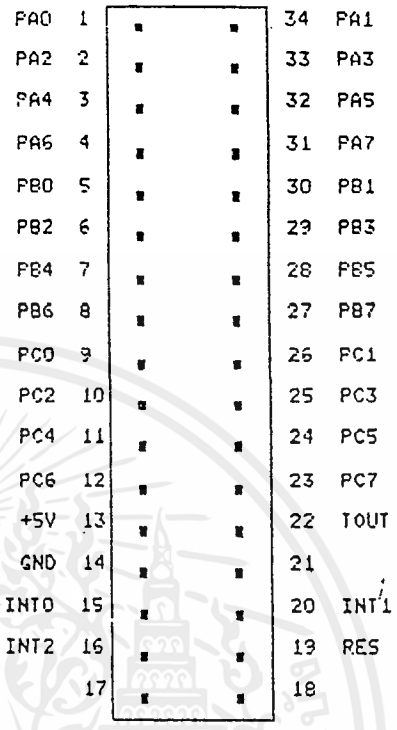


รายละเอียดขา CONNECTOR

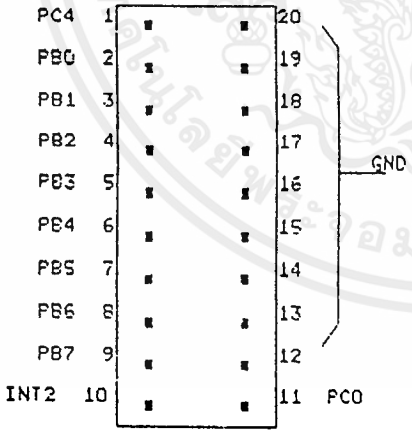
Z80 CPU CONNECTOR



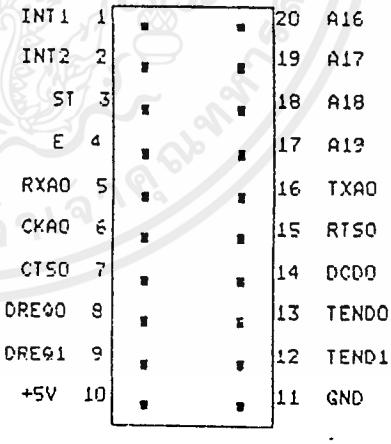
I/O CONNECTOR



PRINTER



EXTEND



บทที่ 7

CP-180 DEBUGGER

ข้อกำหนดต่างๆในการใช้งาน

1. การป้อนคำสั่งจะผ่านทาง KEYBOARD ของเครื่อง PC โดยแต่ละคำสั่งจะต้อง KEY ให้ถูกต้องตามรูปแบบที่กำหนด และคำสั่งเหล่านั้นจะถูกกระทำเมื่อกด ENTER

2. ผู้ใช้สามารถใช้อักษรตัวเล็กหรือตัวใหญ่ได้ตามต้องการ

3. จำนวนตัวเลขต่างๆที่ป้อนเข้าไปจะถือเป็นเลขฐาน 16 (HEX) โดยไม่จำเป็นต้องใส่ H ต่อท้าย และถ้าตัวเลขเป็นอักษร A-F นำหน้าก็ไม่จำเป็นต้องใส่ 0 นำหน้าก่อน * ยกเว้นใน MODE ของ MINI ASSEMBLE * และบาง FUNCTION อาจให้ใช้เลขฐาน 10 ในหัวข้อของ FUNCTION นั้นจะบอกไว้ถ้า KEY ผิดจะมีข้อความแสดงให้ทราบ

4. ในกรณีป้อนคำสั่งไม่ถูกรูปแบบเครื่องจะแสดงคำว่า " SYNTAX ERROR "

5. FUNCTION ที่มีการตั้ง LENGTH ADDRESS ถ้าตั้ง START มากกว่า FINAL เครื่องจะแสดงข้อความให้ทราบ

6. การออกจากสภาวะต่างๆใช้ KEY ESC จะทำให้กลับสู่เครื่องหมาย PRMPT ตามเดิม

7. การกระทำกับข้อมูลที่ต้องเป็น RAM หากมีการกระทำในพื้นที่ที่เป็น ROM หรือ RAM ที่ถูก PROTECT ไว้ FUNCTION ที่ใช้นั้นจะไม่ถูกกระทำและจะมีข้อความบอก

8. เมื่อมีการทำคำสั่งที่ไม่มีในชุดคำสั่งของ Z80180 จะเกิด TRAP INTERRUPT ขึ้น และจะแสดงค่าตำแหน่งที่ผิดนั้นบน DISPLAY

9) RAM บน BOARD ที่ได้ไปกับเครื่องจะเป็น 8 K BYTE แต่การอ้าง RAM บน DEBUG 180 จะเป็นของ ขนาด 32 KBYTE คือ ของUSER ตั้งแต่ 08000H-0EFFFH และ 0F000H-0FFFFH จะเป็นที่เก็บ พารามิเตอร์ร่วมของ ทุกๆPAGE รวมทั้งเป็นส่วนของ STACK ทั้ง USER และ SYSTEM ด้วย ดังนั้น จะเกิดการทับซ้อนกันขึ้น โดยที่ 4 KBYTE หลังของ RAM 8 KBYTE คือตำแหน่ง ADDRESS 09000H-09FFFH จะเป็นเช่นเดียวกับตำแหน่ง 0F000H-0FFFFH นั่นเอง (คือที่เก็บ PARAMETER และ STACK) ดังนั้น เมื่อใช้ RAM 8 K ต้องระวังส่วนนี้ด้วย แต่ถ้าจะให้ง่ายโดยการอ้างตรงกับ 32 K ที่กำหนดไว้ ให้แบ่ง 32K ออก เป็น 8 K 4 PAGE ก็จะได้เป็น

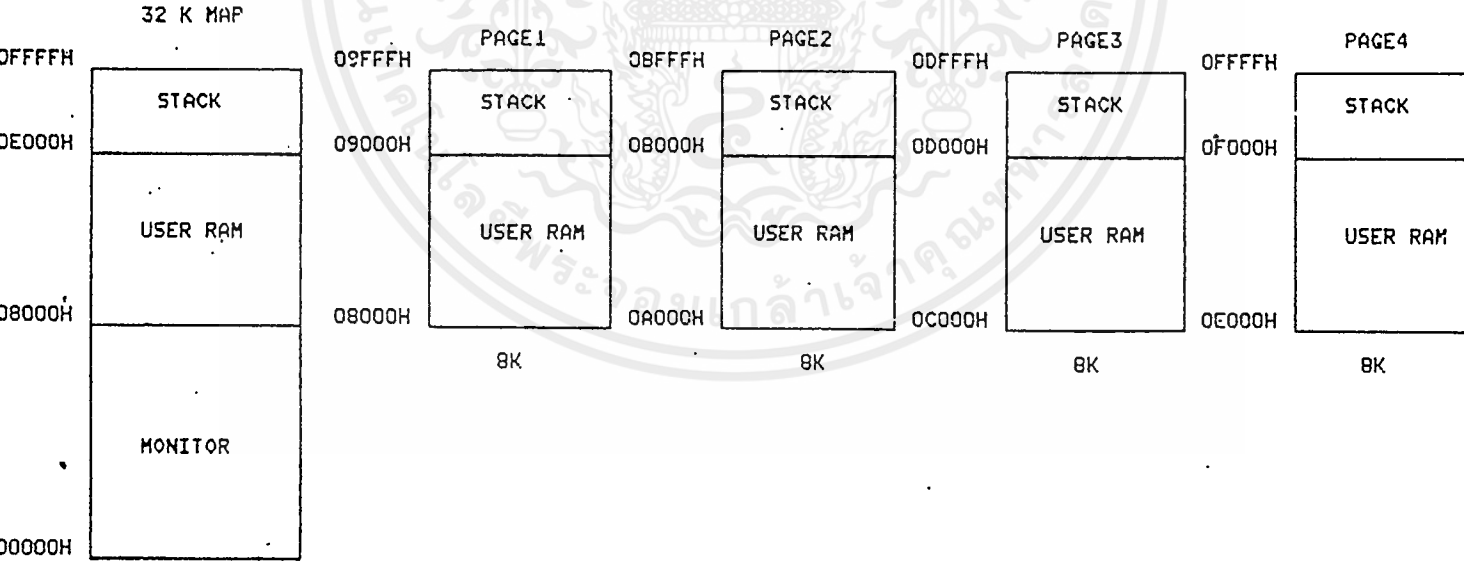
08000H-09FFFH (PAGE 1)

0A000H-0BFFFH (PAGE 2)

0C000H-0DFFFH (PAGE 3)

0E000H-0FFFFH (PAGE 4)

เพราะฉะนั้น เราจะอ้างที่ไหนก็ได้ใน 4 PAGE เมื่อใส่ RAM บน BOARD 8K เพราะเมื่อเราเขียนข้อมูลเกิน ที่เก็บภายในตัว RAM ก็จะทำให้กลับมาเริ่ม ที่ตำแหน่งเริ่ม ต้นของตัวเองอีก (เมื่อ วงจรเลือก RAM ที่ให้ใส่ มากกว่าตัว RAM) ดังนั้น เมื่อเราอ้างที่ PAGE 4 ก็จะทำให้ตำแหน่งที่อ้างตรงกับที่อ้างไว้ที่ 32 K ของ DEBUG ดูรูป การแยก

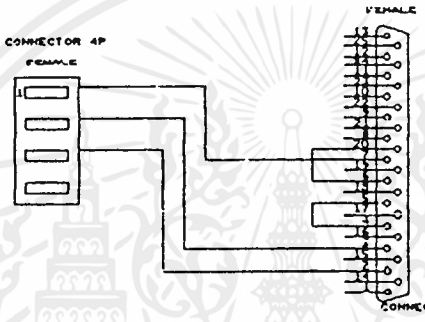


เมื่อเริ่มใช้ DEBUGGER

1. นำ EPROM DEBUGGER ซึ่งเป็นขนาด 32K ที่ ADDRESS 00000H.

2. นำ RAM ซึ่งจะเป็น 8K (6264) หรือ 32K (62256) ที่ ADDRESS 18000H

3. นำสายเชื่อมต่อกับ SERIAL PORT โดยต่อที่ CONNECTOR 4 P ของ ASCI CHANEL 1 บน BOARD เข้ากับ SERIAL PORT ของ IBM PC ซึ่งจะเป็นเครื่อง XT หรือ AT ก็ได้ โดยสายเชื่อมต่อจะมีการต่อดังรูป



4. จากนั้นนำแผ่น SOFT WARE PROCOM ใส่ใน DISK DRIVE แล้วเรียก Z180 ก็จะเข้าสู่การทำงานของ PROCOM เพื่อใช้เป็นตัวติดต่อกับ BOARD Z80180 ที่ DISPLAY จะมีข้อความดังรูป A>Z180

PROCOMM+ (R)
Intuitive Communications (tm)

» PROCOMM PLUS - Version 1.1B «
Copyright (C) 1987, 1988 DATASTORM TECHNOLOGIES, INC.
All Rights Reserved
UNAUTHORIZED DUPLICATION PROHIBITED

PRESS ANY KEY TO ENTER TERMINAL MODE

รายละเอียดคร่าวๆด้านล่างของจอจะบอกสถานะต่างๆไว้โดยเมื่อ KEY ALT Z จะเป็น HELP MENU ดังรูป :-

PROCOMM PLUS Ready!

```
                P R O C O M M   P L U S   C O M M A N D   M E N U

COMMUNICATIONS
BEFORE                                  AFTER
Dialing Directory Alt-D     Hang Up ..... Alt-H
                            Exit ........ Alt-X
DURING
Script Files ... Alt-F5     Send Files ..... PgUp
Keyboard Macros . Alt-M     Receive Files .... PgDn
Redisplay ..... Alt-F6      Log File On/Off  Alt-F1
Clear Screen .... Alt-C     Log File Pause . Alt-F2
Break Key ..... Alt-B       Screen Snapshot . Alt-G
Elapsed Time .... Alt-T     Printer On/Off .. Alt-L
OTHER
Chat Mode ..... Alt-O       Record Mode ..... Alt-R
Host Mode ..... Alt-Q       Duplex Toggle ... Alt-E
Auto Answer ..... Alt-Y     CR-CR/LF Toggle  Alt-F3
User Hot Key 1 .. Alt-J     Kermit Server Cmd Alt-K
User Hot Key 2 .. Alt-U     Screen Pause .... Alt-N

SET UP
Setup Facility .. Alt-S
Line/Port Setup . Alt-P
Translate Table . Alt-W
Key Mapping .... Alt-F8

OTHER FUNCTIONS
File Directory .. Alt-F
Change Directory Alt-F7
View a File ..... Alt-V
Editor ..... Alt-A
DOS Gateway .... Alt-F4
Program Info .... Alt-I
Menu Line Key .......

Press Alt-Z for extended help
```

การตั้ง BAUD RATE ใช้ ALT P และเมื่อเลือกแล้วต้องการ SAVE ไว้อย่างถาวรก็ใช้ ALT S โดย DISK PROCOMM นี้ได้ DEFLUAT ไว้ที่ 9600

การ LOAD ข้อมูลจาก PC ไปยัง BOARD Z80180 ใช้ PAGE UP

การ LOAD ข้อมูลจาก BOARD Z80180 ใช้ PAGE POWN

การออกจาก PROCOMM และเมื่อเข้ามาใหม่ใช้ " EXIT " ใช้ ALT F4

การ VIEW ดูโปรแกรม ใช้ ALT V

การเปิดหน้าจอเพื่อเก็บข้อมูล ALT F1

การเปลี่ยน PATH ของ SUB DIRECTORY ALT F7

การ CLEAR หน้าจอ ALT C

การเปิด , ปิด PRINTER เพื่อนำสิ่งที่อยู่บนจอ DISPLAY ออกสู่ PRINTER ALT L และรายละเอียดอีกหลายอย่าง โดยสามารถหาได้จากคู่มือของ PROCOMM ตามร้านขายหนังสือ COMPUTER ทั่วๆไป

5. เมื่อเริ่มจ่ายไฟให้ BOARD SOFT WARE ใน DEBUG จะทำ AUTO BAUD RATE ซึ่งมี BAUD CENTER ที่ 9600 BAUD ถ้าบน PC ตั้ง BAUD RATE ที่ CENTER ก็จะมีข้อความ Z180 ขึ้น จากนั้นก็จะรอการกด KEY " ENTER " เพียงอย่างเดียว ถ้าเป็น KEY อื่นเครื่องจะไม่สามารถเข้าสู่ DUBUG ได้ ต้อง RESET BOARD Z80180 ใหม่ แล้วทำขบวนการเดิมถูกต้องก็จะมีข้อความ

```
ET-CP180  DEBUGGER  Z80180 (64180)  V1.0
BY  ETT  CO., LTD.
(HELP  MANU  KEY  ?)
ET01>
```

ตอนนี้ก็จะใช้คำสั่งต่างๆที่มีใน DEBUG ได้แล้ว แต่ถ้าการตั้ง BAUD RATE บน PC ไม่ได้อยู่ที่ CENTER เมื่อเริ่มจ่ายไฟหรือกด RESET จะไม่มีคำ Z180 ขึ้นมาแสดง ซึ่งอาจจะเป็นอักษรที่ไม่สามารถสื่อความหมายได้หรืออาจไม่ขึ้นอะไรเลยก็ให้กด ENTER 1 หรือ 2 ครั้ง ก็จะมีข้อความและเครื่องหมาย PRMPT อย่างเดียวกับที่ 9600 BAUD ซึ่ง BAUD RATE ที่เลือกบน PC ได้ตั้งแต่ 300 , 1200 , 2400 , 4800 , 9600 , 19200 , 38400 BAUD RATE

ก่อนเข้าสู่ FUNCTION KEY

1. เมื่อจะดูรูปแบบของ FUNCTION KEY ต่างๆก็โดยการกด KEY " ? " แล้วกด ENTER ก็จะขึ้น HELP MANU ซึ่งแบ่งออกเป็น 3 PAGE โดยแต่ละ PAGE จะรอการกด KEY ใดๆก็ได้ก็จะแสดง PAGE ต่อไปดังรูป

```
HELP FOR ET-RS232 DEBUGER Z80 V1.0
?                                   ;help
A <adr>                             ;assembler
A                                   ;ascii table
B                                   ;break display
B <adr>                             ;set break
B (-)<adr>                          ;del addr break
B (-)                               ;del break all
B <adr>,<count>                     ;breakloop
C <adr1>,<adr2>,<hex>               ;compare hex
C <adr1>,<adr2>,<adr3>              ;compare block
C <adr1>,<adr2>                     ;test ram
C <adr1>,<adr2>,<byte>,<data> <data change>
                                    ;change data max 9 byte
D                                   ;dump current addr
D <adr>                             ;dump fix addr
D ,<final adr>                      ;dump current addr-addr final
D <adr1>,<adr2>                     ;dump addr1-addr2
E                                   ;get current data
E <adr>                             ;get data fix addr
E <adr1>,<adr2>,<+or->(byte>        ;edit byte=1-99

F <adr1>,<adr2>,<hex>               ;fill
F <adr1>,<adr2> <byte>,<data find>
                                    ;find max 9 byte
G                                   ;run pc
G <adr>                             ;run addr
G ,<adr final>                      ;run pc to addr
G <adr1>,<adr2>                     ;run addr1 to addr2
H <hex>,<hex>                       ;hex math +,-,*,/
H <$ hex number>                    ;hex to decimal
H <decimal number>                  ;decimal to hex
I <port>                            ;in port
I <port>,L                          ;in port latch
K <adr>                             ;keep text file to ram
L                                   ;down load
L <offset>                          ;down load + offset
M <adr1>,<adr2>,<adr3>              ;move memory
N                                   ;clear reg
N  <adr1>,<adr2>                    ;clear memory
O <port>,<hex>                      ;out port
P <adr1>,<adr2>                     ;up load
P (F)<HEX NUMBER>                   ;select page

Q                                   ;quit
R                                   ;display reg
R <XX or XX'),<data>                ;get reg pair
R <X or x'),<data>                  ;get 1 reg
R <XF),<1 or 0>                     ;get flag (X= SZHPNC)
RS                                  ;load example program to ram user
T                                   ;step pc
T <adr>                             ;step addr
T <adr>,<count>                     ;nstep
T ,<count>                          ;nstep pc
U                                   ;disassembler
U <adr>                             ;unasm to addr
U <N> OR U <N> <adr>                ;unasm no display addr&code
V                                   ;display RST
V (-)                               ;del variable all
V (-)<RST>                          ;del RST
V <RST),<adr>                       ;get RST=XX
```

2. เนื่องจาก Z80180 อ้าง MEMORY ได้ 1 MBYTE แต่คำสั่งต่างๆที่มีให้ไม่สามารถอ้างเกิน 64 KBYTE จึงใช้เทคนิคในการแบ่งเป็น PAGE ๆ ละ 64K โดย SOFT WARE ของ DEBUG จะ PROGRAM แบ่งเป็น ROM 0-7FFFH และ RAM 8000H-EFFFH ซึ่งเป็นของ USER ในการใช้งาน และส่วน F000-FFFFH จะใช้เป็นที่เก็บ PARAMETER ต่างๆและเป็นส่วน STACK ที่ใช้ร่วมกันทุกๆ PAGE เช่นเดียวกับตำแหน่ง ROM ดังนั้น ข้อมูลใน PAGE อื่นจะเชื่อมต่อกันโดยส่วน STACK นี้

3. เนื่องจาก SOFT WARE ของ DEBUG ใช้ INTERRUPT ของ SERIAL CHANEL 1 ทาง RECEIVE จึงทำให้

3.1 เมื่อ DUMP MEMORY หรือ ดูการ DISASSEMBLER เป็นจำนวนมากๆสามารถหยุดการแสดงผลโดย " CONTROL S " และจะให้แสดงผลต่อก็กด KEY ใดๆก็ได้

3.2 เมื่อกด KEY ESC จะทำให้กลับสู่ MAIN PROGRAM คือกลับสู่เครื่องหมาย PROMPT ยกเว้นในขณะใช้ " ? " HELP MANU

3.3 เมื่อกระทำการ RUN โปรแกรมของ USER สามารถทำงานใน FUNCTION อื่นได้อีก คือ

FUNCTION IN (I)
FUNCTION OUT (O)
FUNCTION REGISTOR (R)
FUNCTION YANK I/O (Y)
FUNCTION DISPLAY MEMORY (D)

3.4 ขณะทำ SINGLE STEP สามารถเปลี่ยนแปลงค่า REGISTOR ได้

4. เมื่อมีการตั้งการทำงานของผู้ใช้ โดยให้ RUN โปรแกรมของผู้ใช้ที่เขียนขึ้นเลยเมื่อมีการเปิดเครื่อง (AUTO START) และสามารถออกจาก AUTO START ได้ภายใน 5 วินาที ที่เปิดเครื่อง โดยการกด KEY ใดๆก็ได้บน PC จากนั้นก็ให้เข้าไป CLEAR ADDRESS EFFFH ถ้าไม่ต้องการให้เกิด AUTO START อีกเมื่อเปิดเครื่อง

5. เมื่อต้องการ RUN PROGRAM ที่เขียนขึ้น โดยโปรแกรมที่เขียนขึ้นไม่ให้วน LOOP ต้องมีการหยุดโปรแกรมอาจจะเขียนด้วย HALT หรือ RST 18 H ซึ่ง RST 18 H จะหยุดโปรแกรม โดยการแสดงค่า REGISTOR ส่วน HALT จะทำให้ CPU หยุดทำคำสั่ง และจะให้ทำงานต่อก็โดยการกด ESC เมื่อกลับสู่ PROMPT

คำสั่ง A

ทำหน้าที่ 2 อย่างด้วยกันคือ ตาราง ASCII และ MINI ASSEMBLE

รูปแบบ A , A addr

คำอธิบาย A แล้ว ENTER จะแสดงตาราง ASCII

A แล้วตามด้วย ADDRESS จะเป็น MINI ASSEMBLE คือเป็นการใส่ข้อมูลใน RAM ด้วยการเขียนเป็นภาษา ASSEMBLY ซึ่งสะดวกกว่าการใส่ OP CODE ทำให้สะดวกในการทดสอบและลองโปรแกรมไม่ยาวมากนัก เนื่องจากไม่ต้อง COMPLIER เพราะเป็น ONE LINE ASSEMBLER คือ เมื่อ KEY 1 คำสั่ง เมื่อกด ENTER ก็จะถูกแปลเป็น MACHINE CODE ลงใน RAM ตามตำแหน่งที่กำหนด และค่าตำแหน่งจะเพิ่มขึ้นเพื่อเก็บ CODE ของคำสั่งถัดไป ข้อเสีย คือ ไม่สามารถกำหนดค่าหรืออ้าง LABEL ต่างๆที่เป็นตัวแปลได้ เนื่องจากการกระทำทั้งหมดบน PC จึงสามารถไปเขียน ASSEMBLY แบบเต็มรูปแบบใน EDITOR ต่างๆ แล้ว COMPLIER เสร็จแล้วจึง LOAD โปรแกรมนั้นมา RUN ได้ ดังนั้นการ JUMP ต่างๆต้องกำหนด ADDRESS และเมื่อจะออกจากคำสั่งนี้ก็กด ENT หรือ ESC

ตัวอย่าง

```
ET01>  A  8000
8000 : LD  A , 40
8002 : DEC  A
8003 : JR  NL , 8002
8005 : <ENT>
ET01>  -
```

คำสั่ง B (BREAK)

ทำหน้าที่ หยุดโปรแกรมเพื่อ CHECK ค่า REGISTOR ต่างๆซึ่งมีรูปแบบหลายอย่างดังนี้

```
B                ;   DISPLAY ค่า BREAK
B  addr          ;   ตั้ง ADDRESS ที่จะ BREAK
B  addr , loop   ;   ตั้ง BREAK พร้อมค่าการวน LOOP ของโปรแกรม
B  - addr        ;   CLEAR ADDRESS BREAK
B  -             ;   CLEAK ADDRESS BREAK ทั้งหมด
```

คำอธิบาย ใช้สำหรับการ TEST โปรแกรมเป็นส่วนๆซึ่งสามารถตั้งการ CHECK ได้ถึง 9 จุด แต่ที่ถาวร คือ การใช้คำสั่ง B ซึ่งสูงสุดคือตั้งจุด BREAK ได้ 8 ตำแหน่ง และถ้ามีการตั้ง BREAK ซ้ำจะมีข้อความบอก ซึ่งการตั้ง BREAK เครื่องจะทำการ SORTING ให้ และจะคงอยู่แม้จะ RESET ก็ตาม ถ้าจะ CLEAR BREAK จะต้องใช้คำสั่งในการ CLEAR ส่วน LOOP BREAK จะเป็นเลข HEX 16 BIT ใช้ TEST โปรแกรมที่ติด LOOP ว่าถูกต้องตามที่กำหนดไหมโดยตำแหน่งที่ตั้ง BREAK LOOP ต้องอยู่ก่อนคำสั่งวน LOOP (JP , JR) ถ้าอยู่ที่คำสั่งวน LOOP (JP , JR) จะไม่สามารถ CHECK LOOP ได้ และเมื่อมีการตั้ง BREAK LOOP แต่จำนวน LOOP ที่ตั้งน้อยกว่าโปรแกรมวน เมื่อกดให้ RUN ต่อจะติด LOOP ไปเรื่อยจนกว่าการ TEST LOOP ของโปรแกรมจริง จะหมดดังนั้นเมื่อตั้ง LOOP โปรแกรมไม่ครบ เมื่อโปรแกรม BREAK ให้ดูค่าแล้ว เมื่อจะ RUN ต่อให้ CLEAR BREAK LOOP นี้ก่อน เมื่อโปรแกรม BREAK แล้วจะ RUN ต่อให้กด <ENT> ส่วน BREAK ในจุดที่ 9 จะอยู่ในส่วนของการ RUN

ตัวอย่าง

```
8000      ;    LD  H , 8
8002      ;    LD  L , 2
8004      ;    MLT  HL
8006      ;    LD  A , 80
8008      ;    DEC  A
8009      ;    JR  NL , 8008

ET01>     B  8002          <ENT>
ET01>     B  8008 , 60     <ENT>
ET01>     B  8004          <ENT>
ET01>     B                <ENT>
```

```
00     8002     0001
01     8004     0001
02     8006     0060

ET01>      G    8000          <ENT>

  PC  SP  (SP)  IF   IX   IY   AF   BC   DE   HL   AF   BC   DE   HL  SZHPNC
8002 FEA0 0000 0004 0000 0000 0000 0000 0000 0800 0000 0000 0000 0000  000000
                                  <ENT>
  PC  SP  (SP)  IF   IX   IY   AF   BC   DE   HL   AF   BC   DE   HL  SZHPNC
8004 FEA0 0000 0000 0000 0000 0000 0000 0000 0802 0000 0000 0000 0000  000000
                                  <ENT>
  PC  SP  (SP)  IF   IX   IY   AF   BC   DE   HL   AF   BC   DE   HL  SZHPNC
8008 FFEA 0000 0000 0000 0000 2102 0000 0000 0010 0000 0000 0000 0000  000010
```

คำสั่ง C

ทำหน้าที่ ใช้ใน 3 หน้าที่ คือ COMPARE , CHANGE และ CHECK RAM

รูปแบบ COMPARE ประกอบไปด้วย COMPARE HEX และ COMPARE BLOCK
โดยมีรูปแบบดังนี้ :-

C start addr , final addr , data ; COMPARE HEX

C start addr , final addr , destination ; COMPARE BLOCK

และเมื่อการ COMPARE ข้อมูลที่ไม่ตรงกันเครื่องก็จะแสดง ADDRESS นั้นให้ทราบ

CHANCE

C start addr , final addr , byte data <space> data change

จะประกอบไปด้วย ADDRESS ที่จะหาข้อมูลที่จะเปลี่ยน BYTE คือ จำนวนข้อมูลที่จะทำการเปลี่ยน DATA คือ ชุดข้อมูลที่จะเปลี่ยนและ DATA CHANGE คือ ข้อมูลที่มาแทนข้อมูลเดิม เมื่อกด ENT เครื่องจะทำการหาและเปลี่ยนให้จากนั้นก็จะบอกจำนวนการเปลี่ยนค่าของชุดข้อมูลในโปรแกรมช่วงนั้นว่าได้เปลี่ยนไปกี่จุด

CHECK RAM

C start addr , final addr

จะทำการ CHECK RAM ตาม ADDRESS ที่กำหนด ถ้า RAM บกพร่องที่ ADDRESS ใด ADDRESS นั้นก็จะค้างที่ DISPLAY รอการกด KEY ใดๆก็ได้ก็จะ CHECK ต่อ และถ้า CHECK เรียบร้อยแล้วค่าใน RAM ตำแหน่งที่ทำการ CHECK จะมีค่าเป็น OOH

ตัวอย่าง COMPARE

ET01> C 8000 , 8200 , 33 <ENT>

เป็นการเปรียบเทียบค่าตั้งแต่ ADDRESS 8000 ถึง 8200 มีค่าเท่ากับ 33 ทั้งหมดไหม

ET01> 8000 , 8200 , 0000 <ENT>

เปรียบเทียบข้อมูลที่อยู่ในช่วง 8000 ถึง 8200 กับข้อมูลใน ADDRESS 0000

CHANGE

```
ADDRESS  8000  :  00  00  33  33  00  00  00  00
         8008  :  33  33  00  55  33  44  00  00
         8010  :  00
```

คำสั่ง D (DISPLAY)

ทำหน้าที่ D

D addr

D , addr

D start addr , final addr

คำอธิบาย ในกรณีกด D อย่างเดียวเครื่องจะแสดงข้อมูลโดยใช้ ADDRESS ล่าสุดที่จำเอาไว้หรือถ้าใส่ ADDRESS ก็จะแสดงข้อมูลที่ ADDRESS นั้นเป็นจำนวน 128 BYTE หรือถ้ากำหนด START , FINAL ก็จะ DISPLAY ตามช่วงที่กำหนด ซึ่งในกรณีที่ DUMP ข้อมูลมากสามารถหยุดการ DISPLAY ได้โดยกด CONTROL S และจะดูต่อก็กด KEY ใดๆก็ได้

ตัวอย่าง

```
ET01>    D  0000

0000 AF 3D 00 20 FC C3 F7 00 - C3 F8 03 00 00 00 00 00 .=. ..............
0010 FE 80 D8 FE A0 D0 18 58 - C3 EB 03 00 00 00 00 00 .......X..........
0020 E5 2A BF FF C3 95 03 00 - E5 2A C1 FF C3 95 03 00 .*.......*.......
0030 E5 2A C3 FF C3 95 03 00 - E5 2A C5 FF C3 87 00 00 .*.......*.......
0040 8F 00 9C 00 A9 00 B6 00 - C3 00 D0 00 DD 00 EA 00 ................
0050 16 38 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00 .8..............
0060 00 00 00 00 00 00 E5 2A - BD FF C3 95 03 00 00 00 .......*........
0070 E5 F5 C5 D6 80 47 0E 02 - 21 C8 43 CD 0A 36 4E 23 .....G..!.C..6N#
ET01>
```

```
ET01>    C   8000 , 800F , 2   33 , 33 , 56 , 78     <ENT>
         02
ET01>    -
```

ตอนนี้ที่ ADDRESS 8002 จะมีค่า 56 , 8003 จะมีค่า 78 และ 8008 , 8009 จะเป็น 56 , 78

คำสั่ง E

ทำหน้าที่ แบ่งเป็น ENTER และ EDIT ส่วนของ ENTER ใช้สำหรับป้อนข้อมูลลงใน MEMORY EDIT ใช้ในการเพิ่มเติมโปรแกรมหรือลดตำแหน่งโปรแกรม ซึ่งจะมีประโยชน์มาก เช่น เราเขียนโปรแกรมโดย ASSEMBLE จนเสร็จแล้วเกิดตกคำสั่งไป 1 BYTE EDIT จะช่วยให้ไม่ต้องเขียนโปรแกรมใหม่

รูปแบบ
```
E       ;   ENTER ตามค่า ADDRESS ล่าสุดที่จำเอาไว้
E addr  ;
E start addr , final addr , + byte
```

คำอธิบาย ENTER ป้อนข้อมูลเป็นเลข HEX ลงใน ADDRESS ที่กำหนด เมื่อใช้คำสั่งนี้เครื่องจะแสดงค่า ADDRESS พร้อมทั้งค่าข้อมูลเดิมที่มีอยู่และเมื่อใช้ป้อนข้อมูลยังไม่ครบ BYTE เกิดผิดผลาดสามารถใช้ BLACK SKIP ไปลบได้และถ้าเกิดป้อนข้อมูล BYTE นั้นผิดผลาดก็สามารถ DECRESS MENT ไปแก้ไขได้โดยกด - และถ้าไม่ป้อนข้อมูลกด ENT ก็จะเป็นการ INC ADDRESS

EDIT จำนวน BYTE ที่จะ INSERT หรือ DELETE โปรแกรมในช่วงนั้นใช้เป็นเลขฐาน 10 คือ 1-99 BYTE โดยเครื่องหมาย + คือ การ INSERT ส่วน - หมายถึง DELETE

ตัวอย่าง
```
ET01>   E 8000
        8000   3E   06
        8001   75   75
        กด ESC
ET01>
ADDRESS   8000 :  3E  06  03  02  3E  01  D3  01
          8008 :  C3  00  80  11  11  11  11  11
ต้องการ INSERT ข้อมูลจำนวน 2 BYTE ที่ ADDRESS 8002
ET01>     E  8002 , 800A , +2        <ENT>
ET01>
```

เมื่อ DISPLAY ข้อมูลจะได้เป็น

```
8000 : 3F 06 00 00 D3 02 3E 01
8008 : D3 01 C3 00 80 11 11 11
```

และเมื่อจะ DELETE ADDRESS 8002 และ 8003

```
ET01>     E 8003 , 800C , -2
```

ข้อมูลจะเป็นเมื่อตอนเริ่มแรก

คำสั่ง F

ทำหน้าที่ แบ่งเป็น FILL และ FIND โดย FILL สำหรับใส่ข้อมูลลงใน MEMORY ส่วน FIND หาข้อมูลใน MEMORY

รูปแบบ
```
F  start addr , final addr , HEX                    ;  FILL
F  start addr , final addr <space> byte , data      ;  FIND
```

คำอธิบาย FILL นำข้อมูล HEX ใดๆ 1 BYTE ใส่ลงใน MEMORY ตามช่วงที่กำหนด

FIND ทำการหาข้อมูลในช่วง ADDRESS ที่กำหนดโดยกำหนดความยาวของข้อมูลที่จะหาได้สูงสุด 9 BYTE

ตัวอย่าง FILL

```
ET01>    F  8000 , 8100 , 77       <ENT>
```

ตอนนี้ในตำแหน่ง 8000 ถึง 8100 จะมีค่า 77

FIND

สมมุติว่าตอนนี้เราป้อนข้อมูลในตำแหน่ง 8005 เป็น 66 และ 8010 เป็น 66

```
ET01>    F  8000 , 8100  2 , 66 , 77    <ENT>
         8005  8010
ET01>
```

เครื่องจะแสดง ADDRESS ที่ข้อมูลตรงกับ DATA ที่หาให้

คำสั่ง G (GO)

ทำหน้าที่ RUN โปรแกรมที่ผู้ใช้เขียนขึ้น ซึ่งมีรูปแบบ ดังนี้

G <ENT>	;	RUN ตาม PC ที่ชี้อยู่ขณะนั้น
G addr <ENT>	;	RUN จาก ADDRESS ที่กำหนด
G , addr	;	RUN จาก PC ถึง ADDRESS ที่กำหนด
G addr1 , addr2	;	RUN จาก ADDRESS1 ถึง ADDRESS2

คำอธิบาย ใช้ในการกระโดดไปทำงานตามโปรแกรมที่กำหนดโดย PC หรือ ADDRESS และสามารถกำหนดว่าจะให้ RUN โปรแกรมจากไหนถึงไหนนั่นก็คือ การ BREAK ต่อจากการตั้ง BREAK ที่มีทั้งหมด 8 จุด ที่เคยกล่าวถึงในคำสั่ง B เมื่อรวมกับ RUN จึงกล่าวได้ว่าตั้ง BREAK สูงสุดได้ 9 จุด แต่การตั้งการ RUN จาก ADDRESS ที่ 1 ถึง ADDRESS ที่ 2 เป็นการตั้ง BREAK เมื่อใช้คำสั่ง RUN เท่านั้น เมื่อออกจาก RUN ค่า BREAK นี้จะถูก CLEAR ซึ่งต่างจากคำสั่ง BREAK ที่จะฝังตัวจนกว่าจะใช้คำสั่ง CLEAR ในการกระโดดไปทำงานตามโปรแกรมนั้น เครื่องจะทำการ LOAD เอาค่า REGISTOR ต่างๆที่จำไว้เข้าในตัว CPU และเมื่อโปรแกรมที่ RUN เกิดถูกการ BREAK ขึ้น เมื่อจะให้ RUN ต่อก็ให้กด KEY <ENT> และเมื่อการ RUN ๆมาเจอคำสั่ง RST ต่างๆ 20H , 28H , 30H และการ INTERRUPT จาก CHIP ภายในต่างๆถ้าไม่มีการกำหนด ADDRESS เครื่องจะไม่ทำการ RUN ให้ และจะแจ้งข้อความให้ทราบ

ในขณะ RUN อยู่สามารถใช้คำสั่งต่างๆได้อีก 4 คำสั่ง คือ

- คำสั่ง I (INPUT) ตามรูปแบบคำสั่ง I
- คำสั่ง O (OUTPUT) ตามรูปแบบคำสั่ง O
- คำสั่ง R (REGISTOR) ใช้ดู REGISTOR ได้อย่างเดียว โดยกด R อย่างเดียวไม่ต้องมี ENTER
- คำสั่ง Y (YANK I/O) ใช้ดูค่า I/O ภายในโดยไม่ต้องกด <ENT>
- คำสั่ง D (DISPLAY) ตามรูปแบบคำสั่ง D แต่เมื่อกด ESC จะออกไปเลย คือ ไม่สามารถกลับมา RUN ต่อได้ทุกอย่างยังเหมือนใช้คำสั่ง D คือ กด CONTROL S ได้

ซึ่งทั้ง 4 คำสั่ง จะไม่ทำให้โปรแกรมที่ RUN อยู่เสีย เว้นแต่คำสั่ง D เมื่อใช้ ESC จะทำให้โปรแกรมที่กำลัง RUN อยู่เสียเพราะจะกลับสู่ MAIN PROGRAM เลย

คำสั่ง H (HEX MATCH)

ทำหน้าที่ ใช้สำหรับคำนวณค่าต่างๆแปลงเลขฐาน 10 และ 16 (HEX)

รูปแบบ

```
H   NUMBER (HEX) , NUMBER (HEX)   ;   ใช้คำนวณ + - x และ /
H   $ NUMBER (HEX)                ;   เปลี่ยนเลข HEX เป็น DECIMAL
H   NUMBER (DECIMAL)              ;   เปลี่ยนเลข DECIMAL เป็น HEX
```

คำอธิบาย ในกรณีใช้ในการคำนวณ NUMBER (HEX) ค่าแรกควรมากกว่า NUMBER (HEX) ค่าสองและในทางกลับกันผู้ใช้ต้องเป็นผู้รู้เองในกรณีให้ค่าแรกน้อยกว่าค่าสอง ซึ่งจริงแล้วการกระทำนี้จะเป็นการดูค่าลบ ซึ่งการคำนวณนี้จะเป็นการคำนวณแบบ 20 BIT

เช่น

```
H   FFFFF , 8          <ENT>
SUM     DIFF    PRODUCT     QUOTIENT
100007  0FFFF7  7FFFF8     01FFFF , 000007
```

และถ้าเป็น

```
H   1 , 2              <ENT>
SUM     DIFF        PRODUCT     QUOTIENT
03      01FFFFFF      02        00.000001
```

ในกรณีเราต้องการดูค่าของ -1 ก็ดูที่ D1FF และถ้าเป็น -3 ก็เพียงเปลี่ยนจาก 2 เป็น 4

ส่วน H $ 1234 จะเปลี่ยนเลข HEX 20 BIT เป็นเลข DECIMAL

```
HEX TO DEC
4660
```

และ H 4660 จะเปลี่ยนเลข DECIMAL เป็น HEX 20 BIT

```
DEC TO HEX
1234H
```

คำสั่ง I (INPUT)

ทำหน้าที่ สำหรับอ่านข้อมูลจาก PORT

รูปแบบ I PORT

I PORT , L

คำอธิบาย ใช้ในการรับข้อมูลจาก PORT ตามหมายเลข PORT ที่กำหนด โดยเครื่องจะแสดงค่าที่รับได้ในขณะนั้นแสดงที่ DISPLAY และเมื่อมี L ต่อท้ายจะเป็นการ INPORT แบบ LATCH คือ จะแสดงค่า INPUT ที่รับเข้ามาแสดงที่ DISPLAY ค้างไว้จนกว่าข้อมูลที่ IN เข้ามาเปลี่ยนก็แสดงค่าใหม่นั้น การออกจาก IN แบบ LATCH นี้ให้กด ESC ในกรณีที่เราจะ CHECK CODE KEY ที่เป็นรหัส ASCII ก็ได้ โดย IN LATCH ที่ PORT DATA ของ SERIAL PORT CHANEL 1 (09H) ซึ่งจะทำให้ได้ตัวอักษรที่กดและค่า ASCII โดยการกดซ้ำกัน 2 ครั้ง ซึ่งครั้งแรกจะได้ตัวอักษรและครั้งที่ 2 จะได้รหัส ASCII

ตัวอย่าง

```
ET01>   I  1080      <ENT>
        DF

ET01>
```

นั่นก็หมายถึง ทำการอ่านข้อมูลจาก PORT 1080 ซึ่งได้ค่า DF หรือ

```
ET01>   I   9 , L    <ENT>
        OD  A41
```

และนี่ก็คือการ INPORT แบบ LATCH ที่ ASCI CHANEL 1 พอเรากด <ENT> ก็จะได้ค่า OD และเมื่อกด A ครั้งแรกจะได้อักษร A และเมื่อกด A อีกครั้งก็จะได้ 41

คำสั่ง K (KEEP TEXT FILE)

ทำหน้าที่ รับข้อมูลที่เป็น TEXT FILE

รูปแบบ K addr

คำอธิบาย จะใช้รับข้อมูลที่เป็น TEXT FILE มาเก็บไว้ใน RAM บน BOARD Z80180 โดย TEXT FILE หรือข้อมูลที่เป็นอักษรข้อความต่างๆซึ่งอาจจะเขียนจาก EDITOR ใดๆก็ได้ เช่น SIDE KIK , TURBO PASCAL DEBASE ฯลฯ

ตัวอย่าง ET01> K 9000 <ENT>

LOADING ACTIVE WHEN PG UP , 4 (ASCII) , TEXT FILE

ซึ่งเมื่อขึ้นข้อความนี้แล้วก็ให้ใช้ FUNCTION ของ PROCOM ส่งข้อมูลลงมาบน BOARD โดยการกด PG UP และ 4 ซึ่งเป็น ASCII จากนั้น PROCOM ก็จะให้ใส่ชื่อ FILE ที่จะส่ง เมื่อเรียบร้อยแล้วกด <ENT> TEXT FILE ก็จะถูก LOAD ลงมายัง RAM แต่ถ้าจะยกเลิกให้กด KEY ใดก็ได้หรือ ESC รอสักครู่ก็จะมีเครื่องหมาย PRMPT ขึ้นมาแสดงให้

ET01>

คำสั่ง L (LOAD)

ทำหน้าที่ สำหรับการ DOWN LOAD ข้อมูลจาก PC ลงบน BOARD

รูปแบบ L

L addr (offset)

คำอธิบาย ข้อมูลที่จะ DOWM LOAD ลงมาจะต้องเป็น INTEL HEX FORMAT ดังรูป

INTEL HEX FORMAT

INTEL 8 BINARY (8 BIT FORMAT)

: NNAAAARRHHHHHHHHHHCC TT

: = RECORD START CHARACTER

NN = BYTE COUNT(HEX)

AAAA = ADDRESS OF FIRST BYTE (HEX)

RR = RECORD TYPE (HEX,00 EXCEPT FOR LAST RECORD WHICH IS 01)

CC = CHECK SUM (HEX)

TT = LINE TERMINATOR (CARRIAGE RETURN, LINE FEED)

SUM = BYTE_COUNT + ADDRESS_HI + ADDRESS_LO + RECORD_BYTE
+ (SUM OF DATA BYTE)

CHECKSUM = ((- SUM) AND FFH))

ในกรณีกด L อย่างเดียว ข้อมูลที่ DOWN LOAD ลงมาจะมาลง RAM ตามที่ ORIGIN ไว้ที่โปรแกรม ในกรณีที่ใส่ ADDRESS ก็จะเป็นการ OFFSET ค่าที่จะให้โปรแกรมที่ DOWN LOAD ลงมาที่ RAM เช่น ORIGIN PROGRAM ที่ 8000H

ตัวอย่าง ET01> L 1000 <ENT>

LOADING ACTIVE WHEN PG UP , 4 (ASCII) , INTEL HEX FILE

เมื่อหน้าจอแสดงข้อความให้กด PAGE UP ซึ่งเป็น FUNCTION ของโปรแกรม PROCOMM จากนั้นให้เลือกเบอร์ 4 (ASCII) แล้วก็ใส่ชื่อ INTEL HEX FILE เมื่อเรียบร้อยกด <ENT> ข้อมูลจะถูก LOAD จนจบ FILE และขึ้นคำว่า " 9000 OK " นั่นก็หมายความว่าข้อมูลได้ถูกถ่ายลงมาบน BOARD เรียบร้อยแล้วที่ ADDRESS 9000H

คำสั่ง M (MOVE)

ทำหน้าที่ สำหรับการ COPY ข้อมูลเป็น BLOCK ๆ

รูปแบบ M start addr , final addr , destination

คำอธิบาย ใช้ในการ COPY ข้อมูลจากส่วนหนึ่งไปอีกส่วนหนึ่ง โดยสามารถ COPY เป็น BOLCK ๆ ละ 64K BYTE ภายในพื้นที่ 1 MBYTE ตำแหน่งที่อ้างถึงในการ COPY จาก SOURCE ไปยัง DESTINATION ด้วยตำแหน่งจริงทาง HARD WARE ลักษณะการย้ายข้อมูลใช้วิธีการ DMA จึงมีความเร็วมาก

ตัวอย่าง ET01> M 18000 , 19000 , 8000 <ENT>
ก็จะเป็นการ COPY ข้อมูลในตำแหน่ง 18000 (8000H ใน PAGE 3 " ET03> ") ถึง 19000H ไปยัง ADDRESS 8000H

หมายเหตุ การใช้คำสั่ง MOVE นี้ จะไม่คำนึงถึง PAGE เพราะฉะนั้นเวลาทำการ COPY ข้อมูลต้องคำนึงถึง ADDRESS จริง ที่กำลังทำอยู่

คำสั่ง N (NEW)

ทำหน้าที่ สำหรับ CLEAR ข้อมูล คือ REGISTOR หรือ RAM ก็ได้

รูปแบบ
```
N                          ;    CLEAR REGISTOR
N START , FINAL ADDR       ;    CLEAR RAM
```

คำอธิบาย สำหรับการ CLEAR ค่า REGISTOR ทั้งหมดให้เป็น 0 และค่าใน PC กับ SP จะเท่ากับค่า DEFAULT คือ

PC = 8000H

SP = FEA0H

ส่วนการ CLEAR RAM ก็จะใส่ 0 ไปยังช่วง ADDRESS ที่กำหนด

ตัวอย่าง
```
ET01>  N
ET01>  N 8000 , 8200
```

คำสั่ง O (OUTPUT)

ทำหน้าที่ สำหรับการส่งข้อมูลไปยัง PORT

รูปแบบ O PORT , HEX

คำอธิบาย PORT คือ เบอร์ PORT ที่กำหนด และ HEX คือ ข้อมูลที่ต้องการส่งออกไป

ตัวอย่าง ET01> O 2 , FF <ENT>
ET01>
หมายถึง ส่งค่าข้อมูล FF ไปยัง PORT 2 ดูผลที่ PORT

คำสั่ง P (PUNCH FOR UP LOAD) , (PAGE SELECT)

ทำหน้าที่ สำหรับการ UP LOAD ข้อมูลจาก BOARD ไปยัง PC และเลือก PAGE ในการใช้งาน

รูปแบบ P start addr , final addr ; UP LOAD

P # NUMBER (HEX) ; PAGE SELECT

คำอธิบาย จะทำการนำข้อมูลที่กำหนดจาก START ADDRESS ถึง FINAL ADDRESS ขึ้นไปเก็บเป็น FILE บน PC ในรูปแบบของ INTEL HEX FORMAT ในกรณี SAVE เป็น FILE ต้องใช้ FUNCTION บน PROCOMM ด้วย

ตัวอย่าง ET01> P 8000 , 8100 <ENT>

ACTIVE WHEN PG DN , 4 (ASCII) , FILE SAVE , STRICK ANY KEY

เมื่อมีข้อความขึ้นก็จะเป็นการรอการกด KEY ซึ่งถ้า KEY ที่กดไม่ใช่ FUNCTION ของ PROCOMM ข้อมูลในตำแหน่ง 8000 , 8100 (ใน PAGE 1) ก็จะแสดงที่ DISPLAY อย่างเดียว ซึ่งอาจจะออกได้ด้วย ESC หรือหยุดดูได้ " CONTROL S " แต่ถ้าเป็น FUNCTION ของ PROCOMM ก็จะเข้าสู่การเก็บข้อมูลในตำแหน่งที่กำหนดลงเป็น FILE โดย KEY PG DN จากนั้นก็เลือกเบอร์ 4 (ASCII) ซึ่งตอนนี้ PROCOMM ก็จะให้ใส่ชื่อ FILE สำหรับที่จะ SAVE เมื่อกด ENTER PROGRAM ในการ UP LOAD ก็รอให้กด KEY ใดๆก็ได้ เมื่อนั้นข้อมูลใน ADDRESS 8000H ถึง 81C0 จะถูกเก็บลงเป็น FILE จนเครื่องแสดงเครื่องหมาย PROMPT อีกครั้ง หมายความว่า ข้อมูลถูกส่งไปยัง PC เรียบร้อยแล้ว จากนั้นให้กด KEY " ESC " เพื่อไม่ให้มีการเก็บข้อมูลลง FILE อีก

คำอธิบาย การ SELECT PAGE เป็นการเลือก RAM ของ USER ว่าจะใช้ RAM เก็บข้อมูลใน ADDRESS ไหน ในกรณีมีโปรแกรมหรือข้อมูลมากๆและส่วนที่ใช้ในการส่งผ่าน PARAMETER F000-FFFFH ซึ่งเป็นส่วน STACK และพื้นที่ว่าง ส่วน RAM จะกำหนดไว้ที่ 8000-EFFF ที่เหลือจะเป็นของ ROM MONITOR 0000-7FFFH และเมื่อมีการย้ายไปที่ PAGE อื่นๆก็จะมี 2 ส่วนที่ตามไปด้วย คือ ส่วน ROM และ STACK ซึ่งคงที่ ส่วนที่เปลี่ยน คือ ของ USER (8000-EFFF) ซึ่งเวลาเรียกใช้งานยังเป็นตำแหน่งเดียวกัน คือ 8000-EFFF แต่ข้อมูลที่กระทำจริงจะลงใน RAM ตามตำแหน่ง PAGE นั้น คือ ตำแหน่ง PAGE X 8000H เช่น

PAGE 0 เพราะฉะนั้นข้อมูล USER คือ ADDRESS 0

PAGE 1 ข้อมูล USER คือ ADDRESS 8000H

ตัวอย่าง	ET01>	;	ตอนนี้ RAM ที่ใช้งาน คือ 8000H
	ET01> P # 4		<ENT>
	ET04>	;	RAM ที่อ้างถึงตอนนี้ คือ 20000H

คำสั่ง . Q (QUIT)

ทำหน้าที่ สำหรับการออกจากโปรแกรม DEBUG

รูปแบบ Q <ENT>

คำอธิบาย ใช้ในการออกจากโปรแกรม DEBUG ซึ่งจะทำให้ CPU อยู่ใน SYSTEM STOP MODE คือ หยุดทำคำสั่งและ I/Q ภายในหยุดทำงานทั้งหมด และยังทำให้ CPU ใช้กำลังงานน้อยที่สุดอีกด้วย จึงทำให้ ถ้าต่อ BATTARY BACK UP RAM ไว้ จะทำให้ข้อมูลไม่สูญหายในขณะนำ SUPPLY ออกจาก BOARD ซึ่งในกรณี BOARD ทดลองนี้จะมี BACK UP RAM ที่ PAGE 1 เพราะ BOARD มีที่ให้ต่อ BATTARY BACK UP ไว้ และเมื่อจ่ายไฟให้ใหม่ก็จะเหมือนกับการเปิดเครื่องครั้งแรก และจะ CHECK ว่าข้อมูลยังถูกต้องอยู่หรือเปล่า ถ้าข้อมูลถูกต้องไม่สูญหายก็จะขึ้นคำว่า CHECK SUM OK พร้อมเสียง BEEP ถึงแม้ว่าถ้ามีการต่อ BATTARY BACK UP RAM ไว้แล้วดึงแหล่งจ่ายไฟออกโดยไม่ผ่านคำสั่ง Q โอกาสที่ข้อมูลจะสูญหายเป็นบาง BYTE ก็ยังมี เพราะ CPU ยังทำงานและ RUN คำสั่งใน MONITOR อยู่ และเนื่องด้วย RAM ไม่มีการต่อ WRITE PROTECT ไว้ ดังนั้นจึงควรออกจาก DEBUG ด้วยคำสั่ง Q ถ้าต้อง BACK UP ข้อมูลของ RAM

คำสั่ง R (REGISTOR)

ทำหน้าที่ สำหรับการกำหนดและดูค่า REGISTOR

รูปแบบ R REG , VALUE

คำอธิบาย ในกรณีกด R อย่างเดียวจะเป็นการดูค่า REGISTOR แต่ถ้าใส่ค่า REG จะเป็นการกำหนดค่า REGISTOR โดยค่าใน VALUE ค่า REGISTOR จะแบ่งได้เป็น

1) ประเภท 16 BIT

PC , SP , IF (ค่า I และ IFF โดย IFF จะสนใจเฉพาะ BIT ที่ 2 เท่านั้น)

IX , IY , AF , BC , DE , HL , AF , BC , DE , HL

2) ประเภท 8 BIT

I , F , A , B , C , D , E , H , L , F , A , B , C , D E , H , L

3) ประเภท FLAG

SF , ZF , HF , PF , NF , CF

(ค่า VALUE ของ FLAG จะเป็นได้เฉพาะ 0 กับ 1 เท่านั้น)

ตัวอย่าง

```
ET01>   R   HL , 1234      ;   กำหนดให้ HL = 1234
ET01>   R   CF , 1         ;   กำหนดให้ CARRY FLAG = 1
ET01>   R   A , 3          ;   กำหนดให้ A = 3
ET01>   R
```

```
 PC   SP   (SP)  IF   IX   IY   AF   BC   DE   HL  AF'  BC'  DE'  HL'  SZHPNC
8000 FEA0 0000 0000 0000 0000 0301 0000 0000 1234 FFFF 00FF 0006 FF6A 000001
```

คำสั่ง S (SHOW EXAMPLE PROGRAM)

ทำหน้าที่ โหลดโปรแกรมตัวอย่างใน MONITOR มาไว้ยัง RAM USER

คำอธิบาย ใช้สำหรับเลือกโปรแกรมตัวอย่าง 1 โปรแกรมในตัวอย่างทั้งหมด 8 โปรแกรมที่อยู่ในส่วนของ ROM MONITOR มายัง RAM USER ในตำแหน่ง 8000H ซึ่งทำให้ผู้ใช้สะดวกขึ้นไม่ต้องเสียเวลาในการ KEY โปรแกรมทดลอง โดยโปรแกรมจะขึ้น MENU ให้ผู้ใช้เลือก NUMBER ของโปรแกรมที่ต้องการ

คำสั่ง T (TRACE)

ทำหน้าที่ สำหรับกำหนดค่าการ SINGLE STEP

รูปแบบ

```
T                ; ONE STEP ตาม PC
T addr           ; ONE STEP ตาม ADDRESS
T , count        ; STEP เริ่มต้นตามตำแหน่ง PC เป็นจำนวนครั้งตามค่า COUNT
T addr , count   ; STEP ตาม ADDRESS ที่ตั้งเป็นจำนวนครั้งของคำสั่งตามค่า
                   COUNT
```

คำอธิบาย ใช้สำหรับทดสอบโปรแกรม โดยสามารถตั้งจำนวนการ STEP ใน 1 ครั้งว่าจะให้ทำได้ครั้งละกี่คำสั่ง คือ จาก 1-0FFFFH คำสั่ง โดยการเลือก T ตาม FORMAT ซึ่งจะแสดงคำสั่งและค่า REGISTOR ที่มีผลต่อคำสั่งที่กระทำนั้นว่ามีการเปลี่ยนแปลงอย่างไรบ้าง และเมื่อกระทำต่อให้กด ENTER และถ้าเป็นโปรแกรมที่ติด LOOP สามารถใช้คำสั่ง R แก้ไขค่า REGISTOR ในการ TEST โปรแกรมได้ โดยคำสั่ง R จะไม่สามารถใช้คำสั่งดู REGISTOR ได้ นอกจากแก้ไขและถ้าโปรแกรมที่ทำ STEP เริ่มแรกเป็น ROM เครื่องจะไม่ทำ STEP ให้ และมีข้อความ MEMORY PROTECT แต่ถ้าโปรแกรมใน RAM ที่เราทำ STEP มีการ CALL เข้าไปในโปรแกรมส่วนที่เป็น ROM หรือ RAM ที่ PROTECT ไว้ เครื่องจะไม่เข้าไป STEP ในจุดนั้นๆ แต่จะไป RUN ผลในโปรแกรมนั้นเลย และจะมาหยุดที่คำสั่งถัดไปใน RAM (ถัดจากคำสั่ง CALL) และถ้าเป็นการ STEP ในคำสั่ง RST ต่างๆเนื่องจากตำแหน่ง RESTART จะอยู่ใน ROM แต่ถ้าเราตั้งตำแหน่ง RST ไว้ใน RAM การ STEP จะตามเข้าไปยัง ADDRESS นั้นๆด้วย แต่ถ้าเป็นใน ROM ก็จะไป RUN ผลเช่นเดียวกับที่กล่าวมา

ตัวอย่าง

```
ET01>T 8000
8000 26 05          LD   H,05H

 PC   SP  (SP)  IF   IX   IY   AF   BC   DE   HL  AF'  BC'  DE'  HL'  SZHPNC
8002 FEA0 0000 0004 0000 0000 0500 0000 0000 050A 0000 0000 0000 0000 000000
>STEP

8002 2E 02          LD   L,02H

 PC   SP  (SP)  IF   IX   IY   AF   BC   DE   HL  AF'  BC'  DE'  HL'  SZHPNC
8004 FEA0 0000 0000 0000 0000 0500 0000 0000 0502 0000 0000 0000 0000 000000
>STEP

8004 ED 6C          MLT  HL

 PC   SP  (SP)  IF   IX   IY   AF   BC   DE   HL  AF'  BC'  DE'  HL'  SZHPNC
8006 FEA0 0000 0000 0000 0000 0500 0000 0000 000A 0000 0000 0000 0000 000000
```

คำสั่ง U (UNASSEMBLER)

ทำหน้าที่ สำหรับทำการ UNASSEMBLER หรือ DISASSEMBLER

รูปแบบ

```
U                              ; DISASSEM ตาม ADDRESS ล่าสุดที่จำเอาไว้ 16 BYTE
U addr.                        ; DISASSEM ตาม ADDRESS ที่กำหนด
U N                            ; DISASSEM ตาม ADDRESS ล่าสุดที่จำเอาไว้เฉพาะ MNEMONIC 16 BYTE
U start addr , final addr      ; DISASSEM จาก ADDRESS START ถึง FINAL
U N , start addr , final addr  ; DISASSEM จาก ADDRESS START ถึง FINAL เฉพาะ MNEMONIC
```

คำอธิบาย ในกรณีกด U อย่างเดียวจะ DISASSEMBLER โดยแสดง ADDRESS OPCODE และ MNEMONIC เป็นจำนวน 16 BYTE และถ้ามี N ต่อจะเป็นการไม่แสดง ADDRESS และ OPCODE และถ้ากำหนดช่วง DISASSEMBLER ก็จะแสดงจนจบช่วง ซึ่งถ้ายาวๆเราสามารถหยุดดูได้โดยกด CONTROL S และจะให้แสดงผลต่อก็กด KEY ใดๆก็ได้ ในกรณีที่จะเก็บการ DISASSEM ลง FILE เพื่อใช้แก้ไขปรับปรุงโปรแกรมก็สามารถทำได้ด้วย FUNCTION ของ PROCOM ดังนี้

ตัวอย่าง

```
ET01> U N , 0000 , 0100 ;
```

ก่อนกด ENTER ให้กด ALT F1 เพื่อเป็นการเก็บสิ่งที่จะเกิดขึ้นบนจอต่อไปลง FILE PROCOM ก็จะให้ใส่ชื่อ FILE เมื่อเรียบร้อยกด ENTER ตัวแสดงสถานะบรรทัดล่างจะบอกว่า LOGON จากนั้นกด ENTER อีกทีตอนนี้โปรแกรมจะ DISASSEM ตำแหน่ง 0000 ถึง 0100 DISPLAY ที่จอพร้อมกับ SAVE สิ่งเหล่านั้นลง FILE จนกระทั่งเครื่องแสดงเครื่องหมาย PROMPT อีกที ให้กด ALT F1 อีกที เพื่อ LOG CLOSED ไม่ให้สิ่งต่างๆที่เกิดขึ้นลง FILE ได้อีก

```
ET01>
```

คำสั่ง V (VARIABLE)

ทำหน้าที่ สำหรับกำหนด RST , INTERRUPT ภายในและภายนอก

รูปแบบ V VAR , VALUE

คำอธิบาย ใช้กำหนดค่า ADDRESS สำหรับการ RESTART , INTERRUPT ทั้งภายในและภายนอก กรณีที่ไม่ใส่ค่า VAR และ VALUE จะเป็นการแสดงค่าที่กำหนดไว้แล้ว แต่ถ้ากำหนด VAR ก็จะเป็นการ SET ค่าใหม่ โดยค่า VAR จะเป็นได้ คือ 20 , 28 , 30 , 38 , 66 , INT 1 , INT 2 , PRT 0 , PRT 1 , DMA 0 , DMA 1 , CSIO ASCI (CHANEL 0) ส่วน VALUE ก็คือ ค่า ADDRESS ที่มีโปรแกรมที่ใช้ RST หรือ INTERRUPT และถ้ามีเครื่องหมาย - ก็จะเป็นการ CLEAR ADDRESS ที่ใช้ RST หรือ INTERRUPT ด้วยรูปแบบ ดังนี้ :-

```
V                   ; DISPLAY
V  var , value      ; SET ADDRESS ของ RST หรือ INTERRUPT
V  - var            ; CLEAR ADDRESS ที่ใช้ RST หรือ INTERRUPT
V  -                ; CLEAR ADDRESS ที่ตั้ง RST กับ INT ทั้งหมด
```

ตัวอย่าง

```
ET01>   V   28 , 8000          <ENT>
ET01>   V   PRT 0 , 8300       <ENT>
ET01>   V                      <ENT>
        28H     8000
        PRT 0   8300
ET01>   V   - 28               <ENT>
ET01>   V                      <ENT>
        PRT 0   8300
ET01>
```

จากตัวอย่างนั่นก็คือ เมื่อ TIMER CHANEL 0 เกิด INTERRUPT ขึ้นก็จะกระโดดไปทำโปรแกรมในตำแหน่ง 8300 นั่นเอง

ในกรณี ผู้ใช้จะใส่ตำแหน่ง ADDRESS ของการ RESTART หรือ INTERRUPT ไว้ในการเขียนโปรแกรมเลย โดยไม่ต้องใช้ FUNCTION V ก็ให้ใส่ค่า ADDRESS ในตำแหน่งดังต่อไปนี้ :-

	ADDRESS
RST 20 H	OFFBFH
RST 28 H	OFFC1H
RST 30 H	OFFC3H
RST 38 H	OFFC5H
NMI 66 H	OFFBDH
INT 1	OFFC7H
INT 2	OFFC9H
PRT 0	OFFCBH
PRT 1	OFFCDH
DMA 1	OFFCFH
DMA 0	OFFD1H
CSIO	OFFD3H
ASCI	OFFD5H

สรุปส่วนสำคัญ

1) การกำหนดตำแหน่งสำหรับการ INTERRUPT จากส่วนต่างๆอาทิ เช่น INTERRUPT 38 TIMER นอกจากใช้ FUNCTION V แล้วเมื่อเราเขียนโปรแกรม อาจจะไม่สะดวกสำหรับการที่ต้องจำตำแหน่งที่จะต้อง INTERRUPT เราสามารถใส่ตำแหน่งที่จะให้ไปทำโปรแกรม INTERRUPT ไว้ยังตำแหน่ง ADDRESS ที่กำหนดไว้ให้ ดังค่า ADDRESS ในหน้าการใช้ FUNCTION V

2) เมื่อต้องการให้เมื่อมีการเปิดเครื่องแล้วไป RUN โปรแกรมที่เขียนขึ้นเลย โดยไม่ต้องมารอการรับ KEY หรือ DISPLAY บนเครื่องคอมพิวเตอร์ IBM ก็โดยการใส่ CODE 0A3H ที่ตำแหน่ง 0EFFFH (เรียน AUTO RUN)

3) เมื่อต้องการออกจาก AUTO RUN ก็เพียงแต่ต่อสาย SERIAL CHANEL 1 เข้ากับเครื่อง COMPUTER ถ้าโปรแกรมของผู้ใช้ที่เขียนขึ้นไม่มีการ DI หรือ DISABLE INTERRUPT FLAG RECEIVE ของ ASCI CHANEL 1 ก็กด KEY " ESC " หรือ KEY ใดๆก็ได้ แต่ถ้ามีการ DI ดังที่กล่าวมาแล้ว ก็ทำได้โดยต่อสาย SERIAL ให้เรียบร้อย เมื่อเริ่มจ่ายไฟเข้าเครื่องภายใน 5 วินาที ให้กด KEY " ESC " บนเครื่อง PC หรือ KEY ใดๆก็ได้ ก็จะเข้าสู่การทำงานของ MONITOR และเมื่อไม่ต้องการให้เกิด AUTO RUN อีกให้ CLEAR ตำแหน่ง 0EFFFH โดยใส่ค่า 00H ไปที่ตำแหน่งนี้

4) การทำ AUTO RUN จะมีการเก็บ PARAMETER ต่างๆนั้นก็คือ PAGE ที่ใส่ AUTO RUN ไว้ ค่า BAUD RATE ที่ติดต่ออยู่ขณะนั้น ถ้ามีการตั้ง AUTO RUN ไว้หลาย PAGE โปรแกรมจะทำการ RUN AUTO RUN NUMBER น้อยก่อน และถ้าผู้ใช้เกิดไปเปลี่ยน BAUD RATE บน PC ทำให้ เมื่อต้องการออกจาก AUTO RUN ทำไม่ได้วิธีแก้ก็คือ OFF SUPPLY ที่เลี้ยง RAM ออก แต่จะทำให้ข้อมูลที่มีอยู่หายไปด้วย หรือค่อยๆเปลี่ยน BAUD RATE บน PC แล้วค่อยไล่กด ESC ก็จะทำให้มีครั้งหนึ่งที่ BAUD RATE ทั้ง 2 เครื่องตรงกัน

5) การใช้ FUNCTION ของ PROCOMM

ALT P	ตั้ง BAUD RATE
PAGE UP	DOWN LOAD
PAGE DOWN	UP LOAD
ALT F4	EXIT
ALT V	VIEW PROGRAM
ALT F1	เปิดหน้าจอเพื่อเก็บข้อมูล
ALT F7	เปลี่ยน PATH

ALT C CLEAR หน้าจอ

ALT L เปิดปิด PRINTER

ในส่วนของการเปิดปิด PRINTER นี้จะมีประโยชน์มาก ในกรณีที่เรา RUN โปรแกรมแล้ว BREAK โปรแกรม ให้แสดงค่าสถานะต่างๆออกที่ PRINTER เพื่อ CHECK ดูความถูกต้องของโปรแกรมจะทำให้สะดวกและรวดเร็วขึ้น เช่น ในกรณี STEP , TRACE , BREAK , DISASSEM เป็นต้น

ภาคผนวก

SOFTWARE

โปรแกรมในส่วนระบบควบคุม

โปรแกรมที่ใช้ในการควบคุมเขียนด้วยภาษา ASSEMBLY Z180 โดยในการเขียนนั้นจะควบคุมโดยเมื่อเปิดเครื่องครั้งแรกจะมีการ CHECK ว่าเปิดเครื่องครั้งแรกหรือเปล่า โดยการ CHECK ค่าใน RAM ที่ตำแหน่ง POWER_ON มีค่า 03FH อยู่หรือเปล่า ถ้าไม่มีค่า 03FH อยู่ในตำแหน่ง POWER_ON นั้นจะทำการเริ่ม CLEAN RAM ที่ใช้เก็บข้อมูลโดยให้มีค่าเป็น 00H เพื่อให้ค่าข้อมูลสวิทช์ทุกห้องทุกชั้นเริ่มต้นที่สถานะ OFF โดยตำแหน่งเริ่มต้นอยู่ที่ตำแหน่ง DATA1 และถ้ามีค่า 03FH อยู่ในตำแหน่ง POWER_ON นั้น ก็จะมีการ CHECK ว่า มีการให้เปิด,ปิด SWITCH อยู่หรือไม่ โดยมีการ CHECK ค่า 0F3H ใน RAM ที่ตำแหน่ง CHECK ถ้าไม่มีทำการ CHECK สถานะ SWITCH ทั้ง 8 ตัวและตัว SENSOR อีก 4 ตัว โดยโปรแกรมที่ทำการที่บอกมานั้นเป็นโปรแกรมย่อยที่ชื่อว่า CHECK_SW และ หลังจากนั้นจะทำการ CHECK ตัว SENSOR อีก 4 ตัว โดยโปรแกรมที่ใช้ CHECK นั้นเป็นโปรแกรมย่อยที่มีชื่อว่า CHECK_ERR เมื่อโปรแกรมย่อยที่ชื่อว่า CHECK_ERR ทำไปแล้วก็จะกลับมาเริ่มที่จะมีการ CHECK ค่า 0F3H ในตำแหน่งที่มีชื่อว่า CHECK อีกแล้วจะทำงานวนไปวนมาอย่างนี้เรื่อยๆ และถ้ามีการสั่งจาก PC ให้มีการเปิด,ปิด SWITCH โดย PC จะส่งมาทาง SERIES PORT โดยการใช้การ INTERUPT ผ่านทางของ SERIES PORT โดยจะส่งค่าที่ใช้ในการเปิด,ปิด SWITCH นั้นมาให้ระบบควบคุมโดยระบบควบคุมจะนำค่าที่ PC ส่งมานั้นมาใส่ไว้ใน RAM ในตำแหน่ง DATAL และหลังจากนั้นก็จะนำค่า 0F3H นั้นไปใส่ไว้ในตำแหน่งของ CHECK หลังจากนั้นก็จะทำการทำการ RET กลับไปยังส่วนที่วน LOOP อยู่ แต่โปรแกรมที่อยู่ในส่วนของวน LOOP จะมีการ CHECK ว่า ค่าใน RAM ที่ตำแหน่งที่มีชื่อว่า CHECK ว่า มีค่า 0F3H อยู่หรือเปล่า ถ้ามีอยู่ก็จะนำค่าในตำแหน่ง DATAL ซึ่งเป็นที่เก็บค่าต่างๆ ที่ PC ส่งมาให้โดยค่าต่างๆ นั้นจะเก็บสถานะทั้งหมดที่เก็บโดยในส่วนของ RAM จะเก็บค่า CHECK IN หรือ CHECK OUT , POWER ON หรือ POWER OFF , CONTINUE หรือ NO CONTINUE , ตำแหน่งของห้องโดยใน BYTE ต่อมา จะเก็บค่าของชั้นและ SWITCH โดยจะแสดงรูปแบบของค่าต่างๆดังนี้

DATA1

SW 1	SW 2	SW 3	SW 4	SW 5	SW 6	SW 7	SW 8

DATA1 + 1

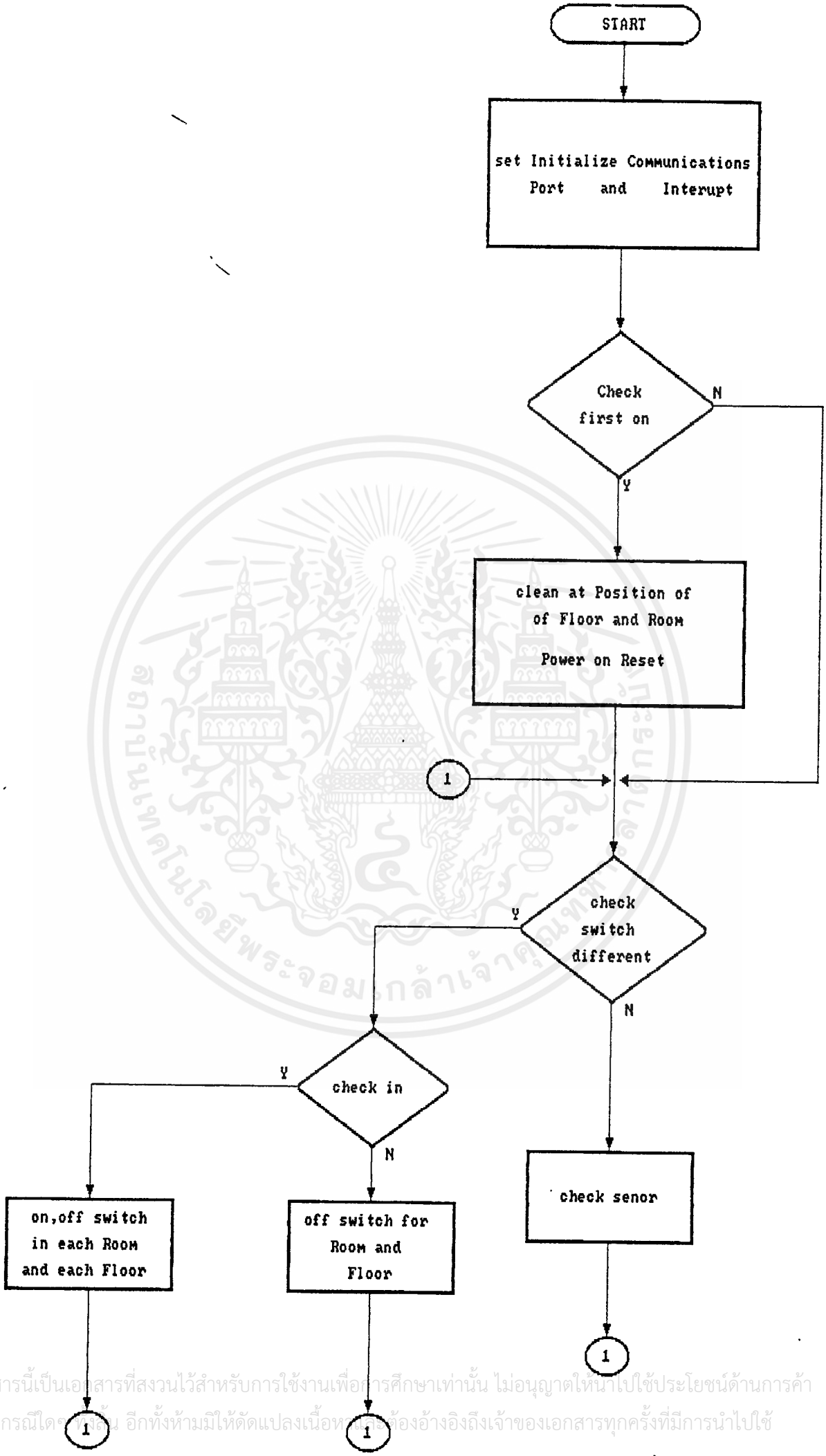
SW 9	SW 10	SW 11	SW 12	SW 13	SW 14	SW 15	SW 16

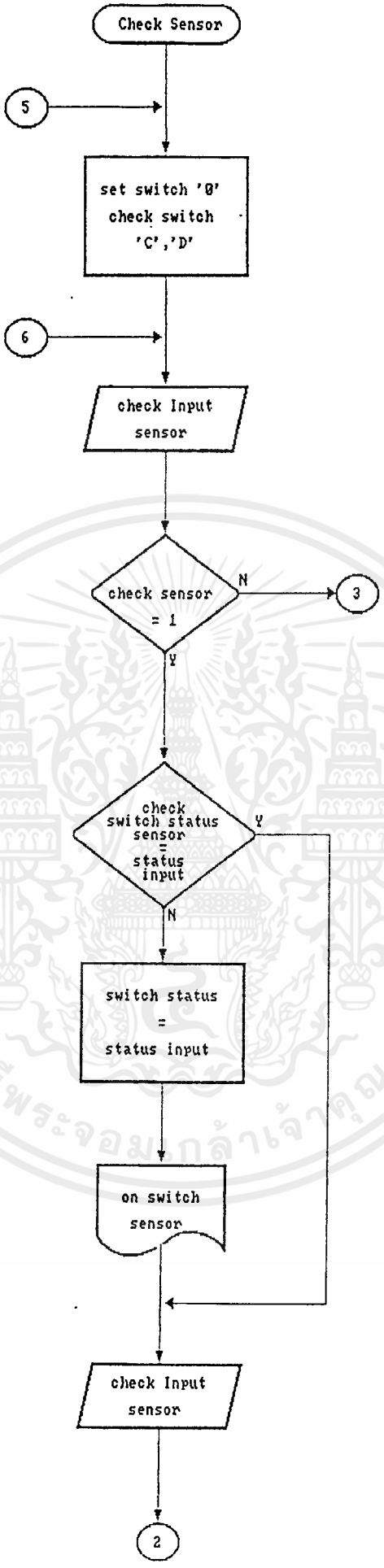
DATAL

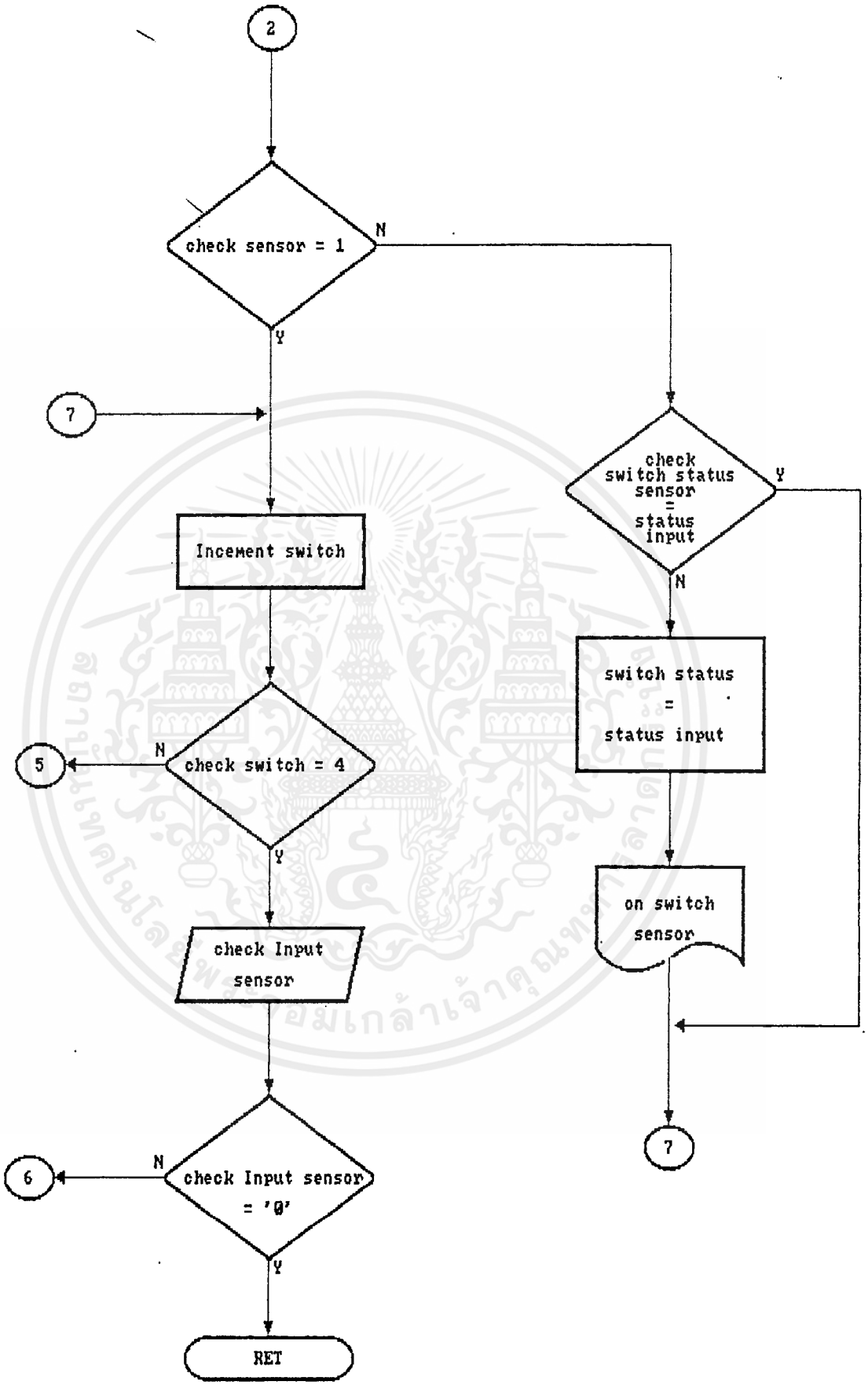
CHECK IN/OUT	POWER	CONTINUE	ROOM

DATAL +1

FLOOR	SWITCH







```
                .ORG   8000H
                ;
                .EQU   NUMBER     ,  600H
                .EQU   MAX_FLOOR  ,  10H
                .EQU   MAX_ROOM   ,  20H
                .EQU   NO_FLOOR   ,  00H
                .EQU   TMP_BC     ,  9000H
                .EQU   TMP_DE     ,  9002H
                .EQU   ASCOINT    ,  0FFD5H
                .EQU   STAT0      ,  04H
                .EQU   CNTLA0     ,  00H
                .EQU   CNTLB0     ,  02H
                .EQU   RDR0       ,  08H
                .EQU   TDR0       ,  06H
                .EQU   P1CNTL     ,  8FH
                .EQU   P1ROOM     ,  8CH
                .EQU   P1FLOOR    ,  8DH
                .EQU   P1SWITCH   ,  8EH
                ;
                ;
START:          LD     A,81H
                OUT    (P1CNTL),A          ; SET STATUS PORT 8255
                CALL   DELAY
                LD     HL,RT_INT
                LD     (ASCOINT),HL        ; SET ADDRESS INTERUPT
                LD     A,02H
                OUT0   (CNTLB0),A
                LD     A,64H
                OUT0   (CNTLA0),A
                LD     A,08H
```

```
                OUT0  (STAT0),A
                ;
WORK:           LD    A,(POWER_ON)        ; CHECK FIRST ON
                CP    3FH
                JP    Z,WORKE
                ;
                LD    HL,DATAL            ; CLEAN STATUS EACH ROOM
                LD    BC,NUMBER           ; AND CLEAN RAM KEEP STATUS
                ;                         ; SWITCH
LOOP:           XOR   A
                LD    (HL),A
                INC   HL
                DEC   BC
                LD    A,B
                OR    C
                JP    NZ,LOOP
                ;
                XOR   A
                LD    (STFLOOR),A
CEL_FLOOR:      XOR   A
                LD    (STROOM),A
CEL_ROOM:       LD    A,0EH
                LD    (SWITCH),A
                LD    A,(STROOM)
                LD    (ROOM),A
                LD    A,(STFLOOR)
                LD    (FLOOR),A
                CALL  OUTP
                LD    A,(STROOM)
                INC   A
```

```
                LD    (STROOM),A
                CP    MAX_ROOM
                JP    NZ,CEL_ROOM
                LD    A,(STFLOOR)
                INC   A
                LD    (STFLOOR),A
                CP    MAX_FLOOR
                JP    NZ,CEL_FLOOR
                ;
                XOR   A
                LD    (STROOM),A
                LD    (STFLOOR),A
                ;
                LD    A,03FH
                LD    (POWER_ON),A
                ;
WORKE:          LD    A,(CHECK)          ; CHECK DIFFERENT
                CP    0F3H               ; SWITCH
                JP    Z,WORK1
                ;
                LD    A,(CFRS)
                CP    NO_FLOOR
                JP    Z,WORK
                ;
CHK:            LD    A,(STFLOOR)
                LD    (FLOOR),A
CHK1:           LD    A,(STROOM)
                LD    (ROOM),A
                ;
                CALL  AFR1
```

```
                CALL CHECK_SW                   ; CHECK STATUS SWITCH
                CALL CHECKERR                   ; CHECK SENSOR
                ;
                LD   A,(STROOM)
                INC  A
                LD   (STROOM),A
                ;
                LD   A,(CHECK)
                CP   0F3H
                JP   Z,WORK1
                LD   A,(FLOOR)
                INC  A
                LD   B,A
                LD   HL,CFRS
                PUSH HL
DRE:            INC  HL
                DEC  B
                JP   NZ,DRE
                LD   A,(HL)
                LD   B,A
                LD   A,(STROOM)
                POP  HL
                CP   B
                JP   C,CHK1
                XOR  A
                LD   (STROOM),A
                LD   A,(CFRS)
                LD   B,A
                LD   A,(STFLOOR)
                INC  A
```

```
                    LD    (STFLOOR),A
                    CP    B
                    JP    C,CHK
                    XOR   A
                    LD    (STROOM),A
                    LD    (STFLOOR),A
                    ;
                    CALL  DELAY
                    ;
                    JP    WORK
                    ;
WORK1:              LD    DE,DATA1
                    ;
WORK2:              LD    A,(DE)
                    BIT   7,A
                    JP    NZ,WORK3

;/* LOOP CHK_OUT */
                    ;
                    CALL  AFR
                    XOR   A
                    LD    (SWITCH),A
                    LD    B,08H
                    LD    A,(HL)
                    PUSH  AF
                    XOR   A
                    LD    (HL),A
                    POP   AF
                    ;
LOOP1:              RLC   A
```

```
                JP    NC,EWORK          ; CY = 0   NO OUTPORT
                PUSH  AF
                XOR   A
                LD    (POWER),A
                PUSH  BC
                CALL  OUTP
                POP   BC
                POP   AF
                ;
EWORK:          PUSH  AF
                LD    A,(SWITCH)
                INC   A
                LD    (SWITCH),A
                POP   AF
                DEC   B
                JP    NZ,LOOP1
                ;
                LD    A,(CONTINUE)
                CP    01H
                JP    NZ,CLEAN
                ;
                INC   DE
                INC   DE
                ;
                JP    WORK2
                ;
;/* LOOP CHK_IN */
                ;
WORK3:          PUSH  DE
                CALL  AFR
```

```
                LD    A,(SWITCH)
                CALL  SCANR
                LD    A,(POWER)
                CP    B
                JP    Z,CHK2
                LD    A,(SWITCH)
                AND   08H
                JP    Z,EQ
                PUSH  HL
                INC   HL
                CALL  EQR
                POP   HL
                JP    OPT
EQ:             CALL  EQR
OPT:            CALL  OUTP
                ;
CHK2:           POP   DE
                LD    A,(CONTINUE)
                CP    01H
                JP    NZ,CLEAN
                ;
                INC   DE
                INC   DE
                ;
                JP    WORK2
                ;
CLEAN:          XOR   A
                LD    (CHECK),A
                CALL  DELAY
                JP    WORK
```

```
                ;
                ;
;/* FIND  FLOOR,ROOM,SWITCH,STATUS,
;   CONTINUE,POWER,POINTER ADDRESS DATA SWITCH
;   I/P REG A
;   O/P REG HL,(FLOOR),(ROOM),(SWITCH),(STATUS)....
                ;
AFR:            LD   IX,TEMP
                BIT  6,A
                JP   NZ,L00
                EX   AF,AF'
                XOR  A
                LD   (POWER),A
                JP   L06
L00:            EX   AF,AF'
                LD   A,01H
                LD   (POWER),A
L06:            EX   AF,AF'
                BIT  5,A
                JP   NZ,L000
                EX   AF,AF'
                XOR  A
                LD   (CONTINUE),A
                JP   L006
L000:           EX   AF,AF'
                LD   A,01H
                LD   (CONTINUE),A
                ;
L006:           EX   AF,AF'
                AND  1FH
```

```
                LD    (ROOM),A
                SLA   A
                LD    (IX+00),A
                PUSH  DE
                POP   HL
                INC   HL
                RRD
                AND   0FH
                LD    (SWITCH),A
                AND   0CH
                SRL   A
                SRL   A
                LD    (STATUS),A
                LD    A,(HL)
                AND   0FH
                LD    (FLOOR),A
                LD    B,06H
                LD    H,00H
                LD    L,A
                ;
MULI:           SLA   L
                RL    H
                DEC   B
                JP    NZ,MULI
                LD    C,(IX+00)
                ADD   HL,BC
                LD    BC,DATAL
                ADD   HL,BC
                RET
                ;
```

```
;/* FIND POINTER ADDRESS DATA SWITCH
;    I/P (ROOM),(FLOOR)
;    O/P REG HL
                ;
                ;
AFR1:           LD    IX,TEMP
                LD    A,(ROOM)
                SLA   A
                LD    (IX+00),A
                LD    A,(FLOOR)
                LD    B,06H
                LD    H,00H
                LD    L,A
                ;
MULI1:          SLA   L
                RL    H
                DEC   B
                JP    NZ,MULI1
                LD    C,(IX+00)
                ADD   HL,BC
                LD    BC,DATAL
                ADD   HL,BC
                RET
                ;
;/* DISPLAY SWITCH EACH ROOM IN FLOOR
;    I/P   (ROOM),(FLOOR),(SWITCH)
;    O/P   NONE
                ;
                ;
OUTP:           LD    A,(ROOM)
```

```
                OUT   (P1ROOM),A
                CALL DELAY
                LD    A,(SWITCH)
                SLA   A
                SLA   A
                SLA   A
                SLA   A
                OUT   (P1SWITCH),A
                CALL DELAY
                LD    A,(FLOOR)
                OR    40H
                OUT   (P1FLOOR),A
                CALL DELAY
                LD    A,(FLOOR)
                OUT   (P1FLOOR),A
                CALL DELAY
                LD    A,(SWITCH)
                CP    0CH
                JP    P,FAS
                ;
                CALL PUTSW1
                ;
SEET:              LD    A,0F0H
                OUT   (P1SWITCH),A
                CALL DELAY
                LD    A,(ROOM)
                OUT   (P1ROOM),A
                CALL DELAY
                LD    A,(FLOOR)
                OR    40H
```

```
                OUT   (P1FLOOR),A
                CALL  DELAY
                LD    A,(FLOOR)
                OUT   (P1FLOOR),A
                CALL  DELAY
                IN    A,(P1SWITCH)          ; CHECK STATUS
                AND   0FH                   ; SWITCH IN ROOM
                LD    (ERROR),A
                LD    A,(SWITCH)
                CPL
                AND   03H
                CP    00H
                JP    Z,DWS
                LD    B,A
                LD    A,(ERROR)
DDD:            SRL   A
                DEC   B
                JP    NZ,DDD
                JP    DSK
DWS:            LD    A,(ERROR)
DSK:            AND   01H
                LD    B,A
                PUSH  BC
                LD    A,(SWITCH)
                CALL  SCANR
                LD    A,B
                POP   BC
                CP    B
                JP    Z,FAS
                LD    A,(ROOM)
```

```
                OUT   (P1ROOM),A
                CALL DELAY
                LD    A,(SWITCH)
                SLA   A
                SLA   A
                SLA   A
                SLA   A
                OUT   (P1SWITCH),A
                CALL DELAY
                LD    A,(FLOOR)
                OR    40H
                OUT   (P1FLOOR),A
                CALL DELAY
                LD    A,(FLOOR)
                OUT   (P1FLOOR),A
                JP    SEET
FAS:            RET
                ;
;/* CHECK STATUS SWITCH SENSOR
;   AND  DISPLAY STATUS SENSOR
                ;
                ;
CHECKERR:
                LD    (TMP_BC),BC
                LD    (TMP_DE),DE
                ;
CHKERR1:        LD    BC,0001H
                LD    D,0BH
                ;
INA:            PUSH DE
```

```
                PUSH BC
                CALL PUTSW
                ;
SEET1:          LD   A,0F0H
                OUT  (P1SWITCH),A
                CALL DELAY
                LD   A,(ROOM)
                OUT  (P1ROOM),A
                CALL DELAY
                LD   A,(FLOOR)
                OR   40H
                OUT  (P1FLOOR),A
                CALL DELAY
                LD   A,(FLOOR)
                OUT  (P1FLOOR),A
                POP  BC
                POP  DE
                CALL DELAY
                IN   A,(P1SWITCH)
                AND  C
                LD   (ERROR),A
                CP   00H
                JP   Z,S
                LD   A,01H
S:              LD   (POWER),A
                CP   01H
                JP   NZ,OOP
RPK:            PUSH DE
                PUSH BC
                LD   A,D
```

```
                CALL SCANR
                LD   A,(POWER)
                CP   B
                JP   Z,IN2
                LD   A,D
                AND  07H
                INC  A
                LD   (POSITION),A
                PUSH HL
                INC  HL
                CALL EQR
                POP  HL
                LD   A,D
                LD   (SWITCH),A
                CALL OUTP
                ;
IN2:            CALL PUTSW
SEET2:          LD   A,0FH
                LD   (SWITCH),A
                CALL OUTP
                CALL DELAY
                POP  BC
                POP  DE
                IN   A,(P1SWITCH)
                AND  C
                CP   00H
                JP   Z,S1
                LD   A,01H
S1:             LD   (POWER),A
                CP   01H
```

```
                JP    NZ,OOP
KPP1:           DEC   D
                SLA   C
                INC   B
                LD    A,B
                CP    04H
                JP    NZ,INA
                ;
                CALL DELAY
                ;
                LD    BC,(TMP_BC)
                LD    DE,(TMP_DE)
                ;
                RET
                ;
OOP:            PUSH DE
                PUSH BC
                LD    A,D
                CALL SCANR
                LD    A,(POWER)
                CP    B
                JP    Z,OP1
                LD    A,D
                AND   07H
                INC   A
                LD    (POSITION),A
                PUSH HL
                INC   HL
                CALL EQR
                POP   HL
```

```
                LD    A,D
                LD    (SWITCH),A
                CALL  OUTP
OP1:            POP   BC
                POP   DE
                DEC   D
                SLA   C
                INC   B
                LD    A,B
                CP    04H
                JP    NZ,INA
                ;
                CALL  DELAY
                ;
                LD    BC,(TMP_BC)
                LD    DE,(TMP_DE)
                ;
                RET
                ;
                ;
;/*  FIND  STATUS SWITCH 12,13
                ;
PUTSW:          LD    A,0CH
                LD    (SWITCH),A
                CALL  AFR1
                LD    A,0CH
                CALL  SCANR
                LD    A,B
                LD    (STATUS12),A
                LD    A,0DH             ;HAVE  HL IN CALL AFR1
```

```
CALL SCANR
LD   A,B
LD   (STATUS13),A
LD   A,01H
LD   (TEMP12),A
LD   (TEMP13),A
INC  A
LD   (STATUS),A
LD   A,(STATUS12)
LD   B,A
LD   A,(TEMP12)
CP   B
JP   Z,ASA1
LD   (POWER),A
LD   A,05H
LD   (POSITION),A
PUSH HL
INC  HL
CALL EQR
POP  HL
LD   A,0C0H
OUT  (P1SWITCH),A
CALL DELAY
LD   A,(ROOM)
OUT  (P1ROOM),A
CALL DELAY
LD   A,(FLOOR)
OR   40H
OUT  (P1FLOOR),A
CALL DELAY
```

```
                LD    A,(FLOOR)
                OUT   (P1FLOOR),A
ASA1:           LD    A,(STATUS13)
                LD    B,A
                LD    A,(TEMP13)
                CP    B
                JP    Z,ST1
                LD    (POWER),A
                LD    A,06H
                LD    (POSITION),A
                PUSH HL
                INC   HL
                CALL EQR
                POP   HL
                LD    A,0D0H
                OUT   (P1SWITCH),A
                CALL DELAY
                LD    A,(ROOM)
                OUT   (P1ROOM),A
                CALL DELAY
                LD    A,(FLOOR)
                OR    40H
                OUT   (P1FLOOR),A
                CALL DELAY
                LD    A,(FLOOR)
                OUT   (P1FLOOR),A
ST1:            RET
                ;
;/* FIND   POWER OF SWITCH
                ;
```

```
SCANR:          PUSH AF
                AND  07H
                INC  A
                LD   (POSITION),A
                LD   B,A
                POP  AF
                AND  8H
                JP   Z,JER
                PUSH HL
                INC  HL
                LD   C,(HL)
                POP  HL
                JP   L2
JER:            LD   C,(HL)
L2:             SLA  C
                DEC  B
                JP   NZ,L2
                RL   B
                RET
                ;
;/*  SET STATUS IN EACH SWITCH
; I/P  (POSITION),(POWER),POINTER ADDRESS DATA SWITCH
                ;
EQR:            LD   A,(POSITION)
                LD   B,08H
                LD   C,(HL)
L3:             CP   B
                JP   NZ,LL5
                PUSH AF
                LD   A,(POWER)
```

```
                CP    00H
                JP    NZ,L4
                RES   0,C
L4:             OR    C
                LD    C,A
                POP   AF
LL5:            RRC   C
                DEC   B
                JP    NZ,L3
                LD    (HL),C
                RET
                ;
CHECK_SW:       LD    D,00H
                XOR   A
STAT:           LD    (STATUS),A
                ;
                PUSH  AF
                PUSH  DE
                CALL  PUTSW1
                ;
SEET3:          LD    A,0F0H
                OUT   (P1SWITCH),A
                CALL  DELAY
                LD    A,(ROOM)
                OUT   (P1ROOM),A
                CALL  DELAY
                LD    A,(FLOOR)
                OR    40H
                OUT   (P1FLOOR),A
                CALL  DELAY
```

```
                LD    A,(FLOOR)
                OUT   (P1FLOOR),A
                CALL  DELAY
                ;
                IN    A,(P1SWITCH)
                AND   0FH
                RLC   A
                RLC   A
                RLC   A
                RLC   A
                LD    (ERROR),A
                POP   DE
                POP   AF
                ;
                LD    (TMP_BC),BC
                LD    C,04
                LD    A,(ERROR)
KU1:            RLC   A
                JP    NC,POW_RES
                ;
                PUSH  AF
                LD    A,01H
                LD    (POWER),A
                POP   AF
                ;
                PUSH  BC
                CALL  PUT_SW
                POP   BC
                ;
                JP    SCHD
```

```
                LD    A,(FLOOR)
                OUT   (P1FLOOR),A
                CALL  DELAY
                ;
                IN    A,(P1SWITCH)
                AND   0FH
                RLC   A
                RLC   A
                RLC   A
                RLC   A
                LD    (ERROR),A
                POP   DE
                POP   AF
                ;
                LD    (TMP_BC),BC
                LD    C,04
                LD    A,(ERROR)
KU1:            RLC   A
                JP    NC,POW_RES
                ;
                PUSH  AF
                LD    A,01H
                LD    (POWER),A
                POP   AF
                ;
                PUSH  BC
                CALL  PUT_SW
                POP   BC
                ;
                JP    SCHD
```

```
                    ;
POW_RES:            PUSH AF
                    XOR  A
                    LD   (POWER),A
                    POP  AF
                    ;
                    PUSH BC
                    CALL PUT_SW
                    POP  BC
                    ;
SCHD:               DEC C
                    JP NZ,KU1
                    ;
                    LD A,(STATUS)
                    INC A
                    LD (STATUS),A
                    ;
                    CP  02H
                    JP  M,STAT
                    LD  D,00H
                    CP  03H
                    JP  NZ,STAT
                    LD  BC,(TMP_BC)
                    RET
                    ;
PUT_SW:             PUSH AF
                    ;
                    LD   A,D
                    AND  07H
                    INC  A
```

```
                LD    (POSITION),A
                LD    A,(STATUS)
                CP    02H
                JP    Z,SWERR
                ;
                PUSH DE
                CALL EQR
                POP   DE
                JP    SWERR1
                ;
SWERR:          PUSH DE
                PUSH HL
                INC   HL
                CALL EQR
                POP   HL
                POP   DE
                ;
SWERR1:         INC   D
                POP   AF
                RET
                ;
PUTSW1:         LD    A,0CH
                CALL SCANR
                LD    A,B
                LD    (STATUS12),A
                LD    A,0DH
                CALL SCANR
                LD    A,B
                LD    (STATUS13),A
                ;
```

```
                LD    A,(STATUS)
                CP    02H
                JP    NZ,SE
                ;
                EX    AF,AF'
                XOR   A
                LD    (TEMP12),A
                INC   A
                LD    (TEMP13),A
                EX    AF,AF'
                ;
SE:             CP    01H
                JP    NZ,SE1
                ;
                EX    AF,AF'
                LD    A,01H
                LD    (TEMP12),A
                DEC   A
                LD    (TEMP13),A
                EX    AF,AF'
                ;
SE1:            CP    00H
                JP    NZ,SE2
                ;
                EX    AF,AF'
                XOR   A
                LD    (TEMP12),A
                LD    (TEMP13),A
                EX    AF,AF'
                ;
```

```
SE2:    LD   A,(STATUS12)
        LD   B,A
        LD   A,(TEMP12)
        CP   B
        JP   Z,ASA
        ;
        LD   (POWER),A
        LD   A,05H
        LD   (POSITION),A
        PUSH HL
        INC  HL
        CALL EQR
        POP  HL
        ;
        LD   A,0C0H
        OUT  (P1SWITCH),A
        CALL DELAY
        LD   A,(ROOM)
        OUT  (P1ROOM),A
        CALL DELAY
        LD   A,(FLOOR)
        OR   40H
        OUT  (P1FLOOR),A
        CALL DELAY
        LD   A,(FLOOR)
        OUT  (P1FLOOR),A
        ;
ASA:    LD   A,(STATUS13)
        LD   B,A
        LD   A,(TEMP13)
```

```
                CP    B
                JP    Z,SERT
                ;
                LD    (POWER),A
                LD    A,06H
                LD    (POSITION),A
                PUSH HL
                INC   HL
                CALL EQR
                POP   HL
                ;
                LD    A,0D0H
                OUT   (P1SWITCH),A
                CALL DELAY
                LD    A,(ROOM)
                OUT   (P1ROOM),A
                CALL DELAY
                LD    A,(FLOOR)
                OR    40H
                OUT   (P1FLOOR),A
                CALL DELAY
                LD    A,(FLOOR)
                OUT   (P1FLOOR),A
                CALL DELAY
SERT:           RET
                ;
;/* ADDRESS INTERUPT
                ;
RT_INT:         PUSH AF
                INO   A,(RDR0)
```

```
                CP    'U'
                JP    Z,RT_INT1
                CP    'E'
                JP    Z,RT_INT2
                CP    'S'
                JP    Z,RT_INT3
                POP   AF
                EI
                RET
                ;
RT_INT1:        PUSH HL
                PUSH DE
                PUSH BC
                ;
                LD    DE,1024
                LD    HL,DATAL
LOATY:          LD    C,(HL)
                CALL TX_ASCII
                INC   HL
                DEC   DE
                LD    A,E
                OR    D
                JP    NZ,LOATY
                ;
                POP   BC
                POP   DE
                POP   HL
                POP   AF
                EI
                RETI
```

```
                ;
RT_INT2:        PUSH HL
                PUSH BC
                LD   A,0
                OUTO (STATO),A
                LD   HL,DATA1
                LD   B,4
                ;
RT_INT21:
                INO  A,(STATO)
                BIT  7,A
                JP   Z,RT_INT21
                INO  A,(RDRO)
                LD   (HL),A
                INC  HL
                DEC  B
                JP   NZ,RT_INT21
                ;
                LD   A,0F3H
                LD   (CHECK),A
                LD   A,8
                OUTO (STATO),A
                ;
                CALL DELAY
                POP  BC
                POP  HL
                POP  AF
                EI
                RETI
                ;
```

```
                ;
RT_INT3:        PUSH HL
                PUSH BC
                LD   A,0
                OUTO (STATO),A
                LD   HL,CFRS
                ;
RT_INT31:       INO  A,(STATO)
                BIT  7,A
                JP   Z,RT_INT31
                INO  A,(RDRO)
                LD   (HL),A
                INC  HL
                LD   B,A
                ;
RT_INT32:       INO  A,(STATO)
                BIT  7,A
                JP   Z,RT_INT32
                INO  A,(RDRO)
                LD   (HL),A
                INC  HL
                DEC   B
                JP    NZ,RT_INT32
                ;
                LD    A,8
                OUTO  (STATO),A
                ;
                CALL DELAY
                POP   BC
                POP   HL
```

```
                POP   AF
                EI
                RETI
                ;
                ;
;************************
;       SEND ASCII       *
;************************
; I/P = C
; REG = NONE
                ;
TX_ASCII:       PUSH AF
TX_ASC1:        INO  A,(STAT0)
                BIT  1,A
                JP   Z,TX_ASC1
                OUT0 (TDR0),C
                POP  AF
                RETI
                ;
DELAY:          PUSH AF
                PUSH HL
                PUSH BC
                LD   HL,029H
SSSS:           LD   B,L
DEL:            DEC  B
                JP   NZ,DEL
                DEC  HL
                LD   A,L
                OR   H
                JP   NZ,SSSS
```

```
                POP   BC
                POP   HL
                POP   AF
                RET
                ;
DATAL:          .RS   1025
                ;
DATA1:          .RS   10H
CFRS:           .RS   20H
STFLOOR:        .DB   0
STROOM:         .DB   0
ERROR:          .DB   0
TEMP:           .DB   0
TEMP12:         .DB   0
TEMP13:         .DB   0
CONTINUE:       .DB   0
POWER:          .DB   0
POSITION:       .DB   0
STATUS:         .DB   0
STATUS12:       .DB   0
STATUS13:       .DB   0
SWITCH:         .DB   0
ROOM:           .DB   0
FLOOR:          .DB   0
POWER_ON:       .DB   0
CHECK:          .DB   0
                ;
                .ORG  0EFFEH
                .DB   096H
                .DB   0A3H
```

โปรแกรมในส่วนของ PC

ส่วนของโปรแกรม PCOM1.EXE หรือ PCOM2.EXE เมื่อทำการ RUN โปรแกรมจะต้องรอเวลาระยะหนึ่งเพื่อทำการ LOAD DATA จาก ส่วนควบคุมระบบ จากนั้นก็จะแสดง VERSION ของโปรแกรม แล้วหยุดรอรับ KEY ใดๆ เมื่อกด KEY ใดๆ แล้วก็จะแสดง MENU ในของโปรแกรมโดยมีดังนี้คือ FILE , EDIT , DIS-PLAY , SETUP , VERSION , QUIT โดยจะมี BAR อยู่ที่ FILE ถ้าต้องการเลือก MENU ใดๆ ก็เลื่อน BAR ไปในตำแหน่งต้องการเลือกโดยถ้าเลือก EDIT ก็ต้องเลื่อน BAR ไปทางขวาโดยกดลูกศรที่ชี้ไปทางขวาหนึ่งครั้งโดยที่ BAR อยู่ที่ตำแหน่งของ FILE และสามารถเลื่อนไปทางซ้ายหรือทางขวาได้โดยการกดลูกศรที่ชี้ไปทางซ้ายหรือทางขวา ถ้าเลือก FILE ก็ต้องเลื่อน BAR ไปในตำแหน่งของ FILE แล้วกด KEY <ENTER> ก็จะเข้าส่วนของ FILE ถ้าเลือก EDIT ก็ต้องเลื่อน BAR ไปในตำแหน่งของ EDIT แล้วกด KEY <ENTER> ก็จะเข้าส่วนของ EDIT ถ้าเลือก DISPLAY ก็ต้องเลื่อน BAR ไปในตำแหน่งของ DISPLAY แล้วกด KEY <ENTER> ก็จะเข้าส่วนของ DISPLAY ถ้าเลือก SETUP ก็ต้องเลื่อน BAR ไปในตำแหน่งของ SETUP แล้วกด KEY <ENTER> ก็จะเข้าส่วนของ SETUP ถ้าเลือก VERSION ก็ต้องเลื่อน BAR ไปในตำแหน่งของ SETUP แล้วกด KEY <ENTER> ก็จะเข้าส่วนของ SETUP ถ้าเลือก QUIT ก็ต้องเลื่อน BAR ไปในตำแหน่งของ QUIT แล้วกด KEY <ENTER> ก็จะเข้าส่วนของ QUIT

ส่วนของ FILE ก็จะมีส่วนของ MENU ในส่วนของ FILE โดยมี MENU ดังนี้คือ UPDATE DATA , LOAD FILE , SAVE FILE , DIRECTORY , CHANGE DRIVE และ OS SHELL โดยในการเลือก MENU ต่างๆ ก็จะใช้การเลื่อน BAR ไปในตำแหน่งที่ต้องการเลือก โดยจะใช้การกด KEY ลูกศรที่ชี้ขึ้นหรือลงเพื่อเลือก MENU ที่ต้องการเมื่อ BAR อยู่ที่ตำแหน่งที่ต้องการเลือกแล้วกด KEY <ENTER> เพื่อเข้าไปใน MENU นั้นๆ

ส่วนของ UPDATE DATA นั้นเป็นส่วนที่ต้องนำ DATA จากส่วนควบคุมซึ่งต้องมีการรอเวลาชั่วระยะเวลาหนึ่ง เพื่อนำข้อมูลมาประมวลผลในส่วนของที่แสดงผลคือส่วนของ DISPLAY ซึ่งมีข้อความแสดงในส่วนข้างล่างของจอว่า PLEASE WAIT . . .

ส่วนของ SAVE FILE นั้นจะมี MENU อีก 2 อย่าง คือ SAVE CHECK , SAVE STATUS ในส่วนของ SAVE CHECK นั้นจะเป็นส่วนเก็บสถานะของการ CHECK IN หรือ CHECK OUT ไว้ใน FILE ชื่อ CHECK.CFG ถ้าไม่สามารถเก็บข้อมูลลงไปได้ จะมีข้อความแสดงไม่สามารถเก็บข้อมูลลงใน FILE ชื่อ CHECK .CFG ได้ และในส่วนของ SAVE STATUS นั้นจะเป็นส่วนเก็บสถานะของการ ON , OFF SWITCH ในแต่ละห้อง ถ้าไม่สามารถเก็บข้อมูลลงไปได้ จะมีข้อความแสดงว่าไม่สามารถเก็บข้อมูลลงใน FILE ชื่อ STATUS.CFG ได้ เมื่อสามารถเก็บได้ ก็จะมีข้อความว่ากำลังเก็บข้อมูลอยู่ในขณะนั้นแล้วจะแสดง MENU ของ SAVE FILE ใหม่ ถ้าต้องการออกจาก MENU SAVE FILE จะต้องกด KEY <ESC> เพื่อเป็นการบอกว่าต้องการจะออกจากส่วนของ SAVE FILE ไปในส่วนของ MENU FILE

ส่วนของ LOAD FILE นั้นจะมี MENU อีก 2 อย่าง คือ LOAD CHECK , LOAD STATUS ในส่วนของ LOAD CHECK นั้นจะเป็นส่วนนำสถานะของการ CHECK IN หรือ CHECK OUT จาก FILE ชื่อ CHECK.CFG ออกมาให้ตัวแปร ถ้าไม่สามารถนำข้อมูลออกจาก FILE ชื่อ CHECK.CFG มาให้ตัวแปรได้ จะมีข้อความแสดงไม่สามารถนำข้อมูลออกมาจาก FILE ชื่อ CHECK.CFG มาให้ตัวแปร และในส่วนของ LOAD STATUS นั้นจะเป็นส่วนนำสถานะของการ ON , OFF SWITCH ในแต่ละห้อง ถ้าไม่สามารถนำข้อมูลออกมาจาก FILE ชื่อ STATUS .CFG มาให้ตัวแปรได้ จะมีข้อความแสดงว่าไม่สามารถนำข้อมูลออกมาจาก FILE ชื่อ STATUS.CFG มาให้ตัวแปรได้ เมื่อสามารถนำข้อมูลออกมาได้ ก็จะมีข้อความว่ากำลังนำข้อมูลออกมาอยู่ในขณะนั้นแล้วจะแสดง MENU ของ LOAD FILE ใหม่ ถ้าต้องการออกจาก MENU LOAD FILE จะต้องกด KEY <ESC> เพื่อเป็นการบอกว่าต้องการจะออกจากส่วนของ LOAD FILE ไปในส่วนของ MENU FILE

ส่วนของ DIRECTORY นั้นเป็นส่วนแสดงรายชื่อ FILE ทั้งหมดที่มีอยู่ใน DISK โดยสามารถดูรายชื่อ FILE ใน DISK ไหนก็ได้ ถ้าไม่สามารถดูรายชื่อได้จะมีข้อความแสดงออกมาว่าไม่สามารถแสดงรายชื่อได้

ส่วน CHANGE DRIVE นั้น เป็นส่วนของทำการเปลี่ยน DRIVE โดย

สามารถเลือกได้สูงสุด 5 DRIVE แต่จะมีตัว CHECK ว่าในคอมพิวเตอร์มี DRIVE สูงสุดกี่ DRIVE ก็แสดงแต่ละ DRIVE ของคอมพิวเตอร์ โดยสามารถเปลี่ยน DRIVE โดยการเลื่อน KEY ลูกศรไปทางซ้ายหรือทางขวาได้ โดยให้ BAR นั้นไปอยู่ตำแหน่งของ DRIVE ที่ต้องการแล้ว กด KEY <ENTER> แล้ว ก็จะเป็นการเปลี่ยน DRIVE ที่ต้องการแล้ว โดยสามารถดูรายชื่อ FILE ที่ต้องการตาม DRIVE ที่เราเปลี่ยนไป

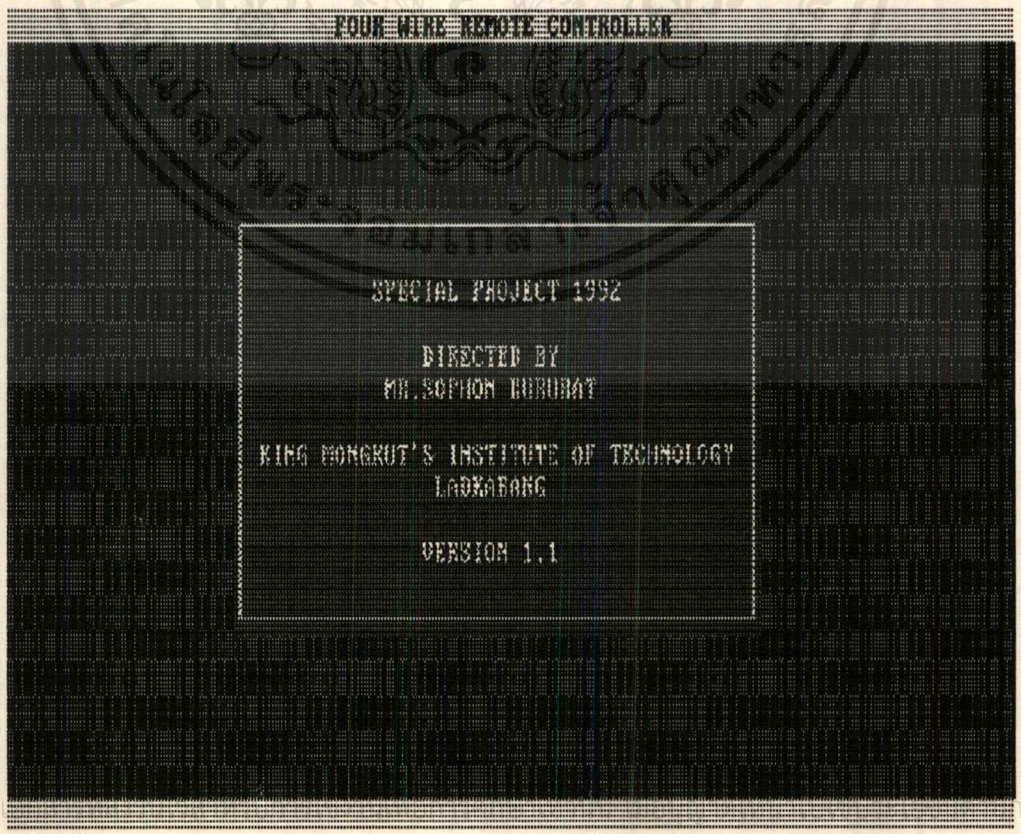
ส่วน OS SHELL จะเป็นส่วนที่ออกจากโปรปกรมชั่วคราวเพื่อไประบบของ DOS โดยจะมีข้อความแสดงว่าขณะนั้นออกจากโปรแกรมชั่วคราวแล้ว จะกลับเข้ามาใหม่ยังโปรแกรมก็ทำการกด KEY คำว่า'EXIT'แล้วกด KEY <ENTER>ก็จะกลับมายังโปรแกรม

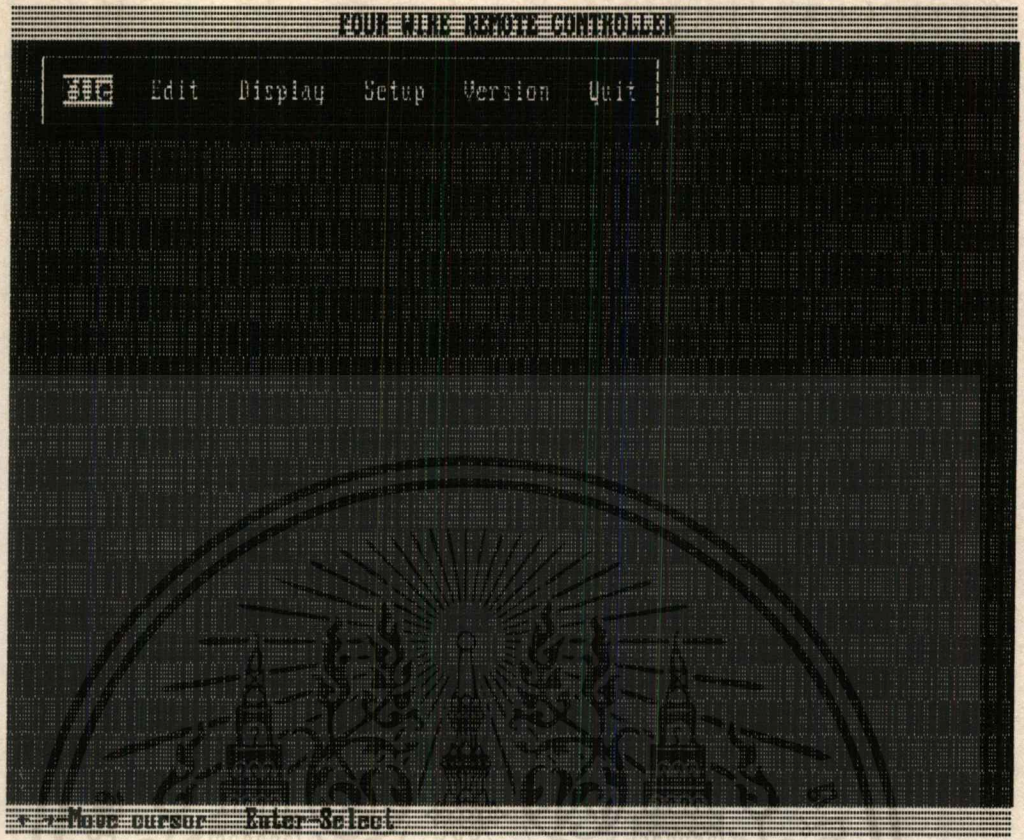
ส่วนของ EDIT นั้นเป็นส่วนที่ใช้ในการเปิด,ปิด SWITCH ภายในแต่ละห้อง โดยต้องมีการบอกว่ามีการ CHECK IN หรือการ CHECK OUT ก่อน ถ้าเป็นการ CHECK IN นั้นจะสามารถเปิด,ปิด SWITCHS ได้ในแต่ละ SWITCH โดยต้องมีการเลือกชั้นก่อนแล้วเลือกห้องและหลังจากนั้นก็จะเป็นการเลือก SWITCH ที่จะเปิด,ปิด ถ้าเป็นการ CHECK OUT เป็นการปิด SWITCH ในแต่ละห้องโดยต้องเลือกชั้นแล้วเลือกห้อง ก็จะเป็นการปิด SWITCH ทุกดวงของชั้นนั้นในห้องนั้น

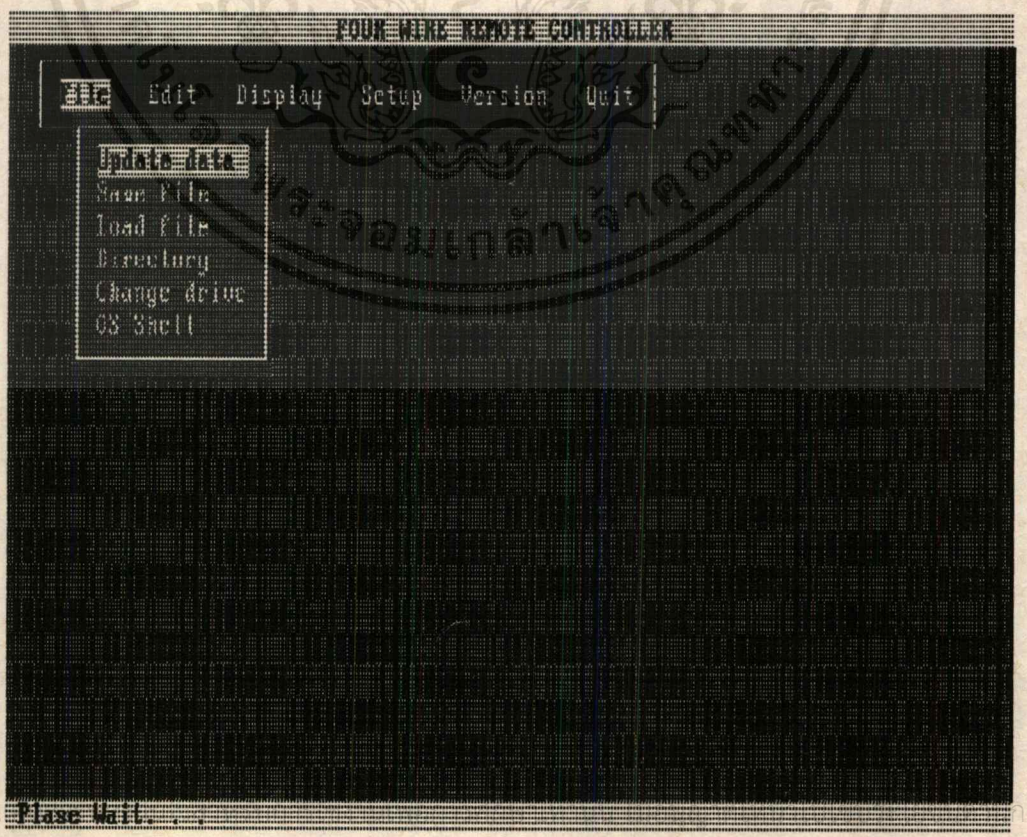
ส่วน DISPLAY นั้นจะเป็นส่วนแสดงว่ารายละเอียดของ SWITCH ในแต่ละห้องของชั้นนั้นว่ามี SWITCH ตัวไหนมีการเปิด , ปิด อยู่หรือเปล่าและยังแสดงว่ามีตัว SENSOR ว่าตอนนี้ทำงานอยู่หรือเปล่าด้วย

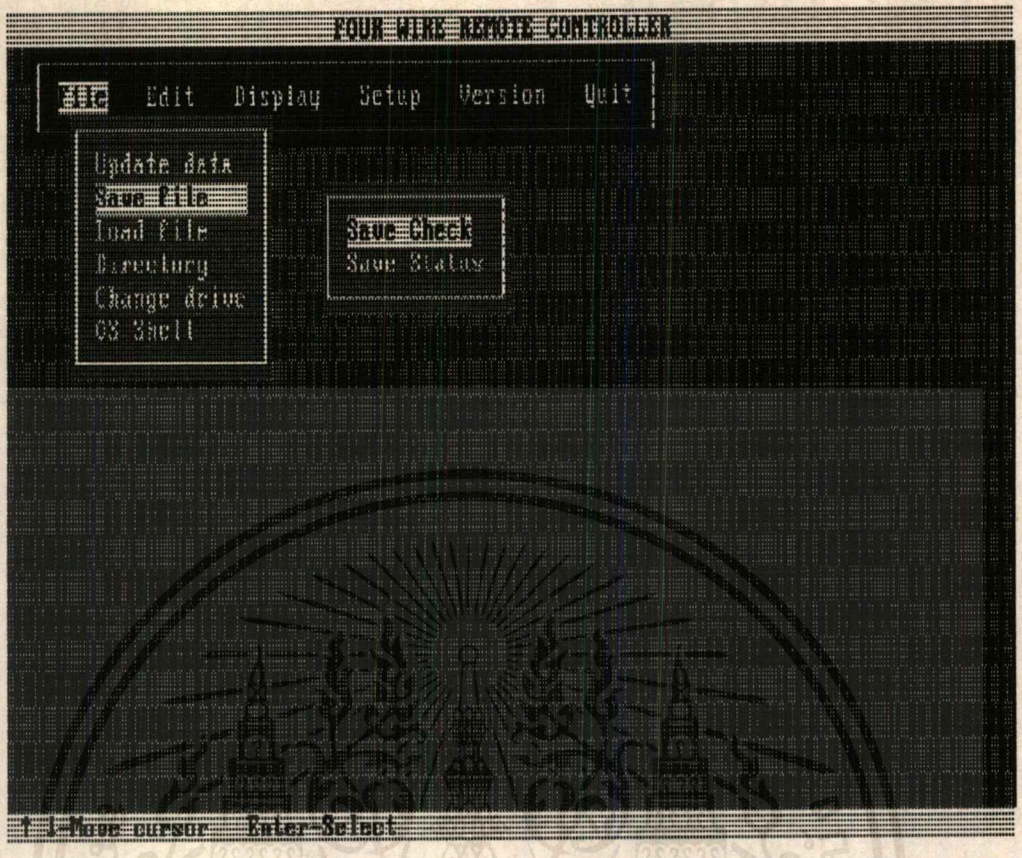
ส่วนของ SETUP จะเป็น SET จำนวนชั้นสูงสุด และจำนวนห้องสูงสุดในแต่ละชั้นที่ SET เอาไว้ เมื่อเปิดระบบควบคุมครั้งแรกทุกครั้ง จะมีการ SET จำนวนชั้นสูงสุดและจำนวนห้องสูงสุดในแต่ละชั้นก่อน เพราะฉะนั้นระบบควบคุมจะไม่ทำการตรวจระบบ SENSOR

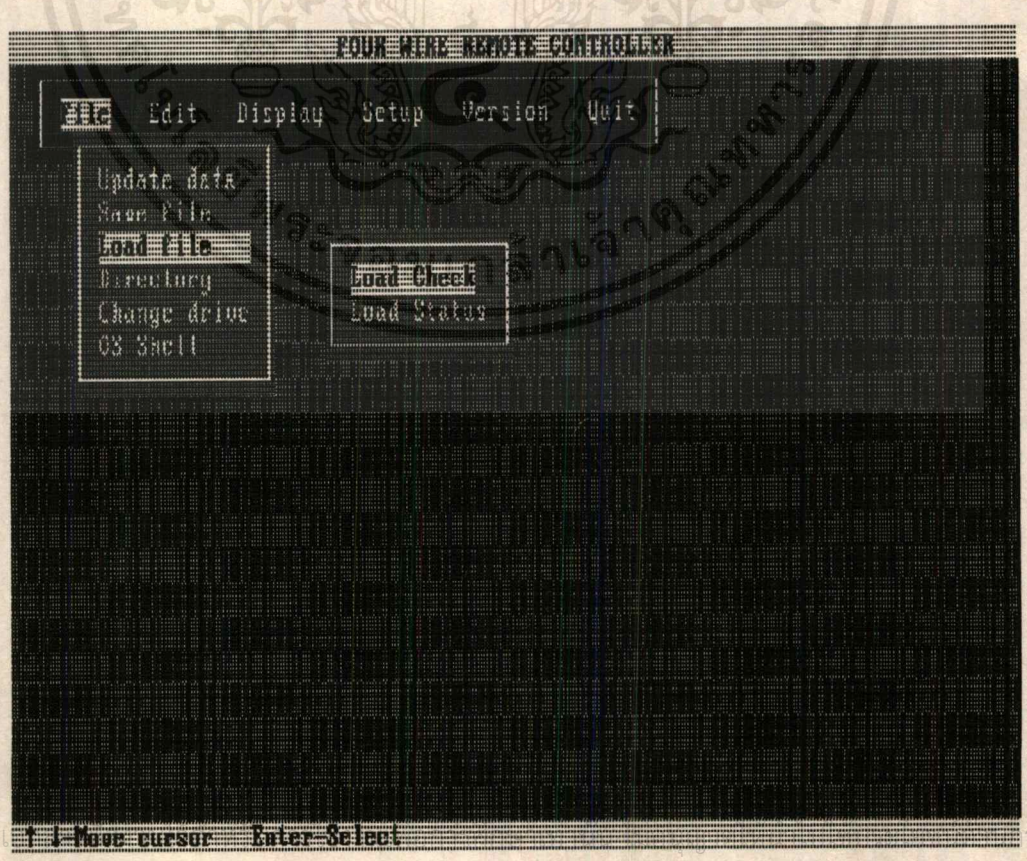
ส่วนของ QUIT นั้นจะเป็นการออกจากโปรแกรม โดยถ้าจะเข้ามายังโปรแกรมใหม่นั้นจะต้องมีการ LOAD FILE ชื่อ PCOM1.EXE หรือ PCOM2.EXE ใหม่

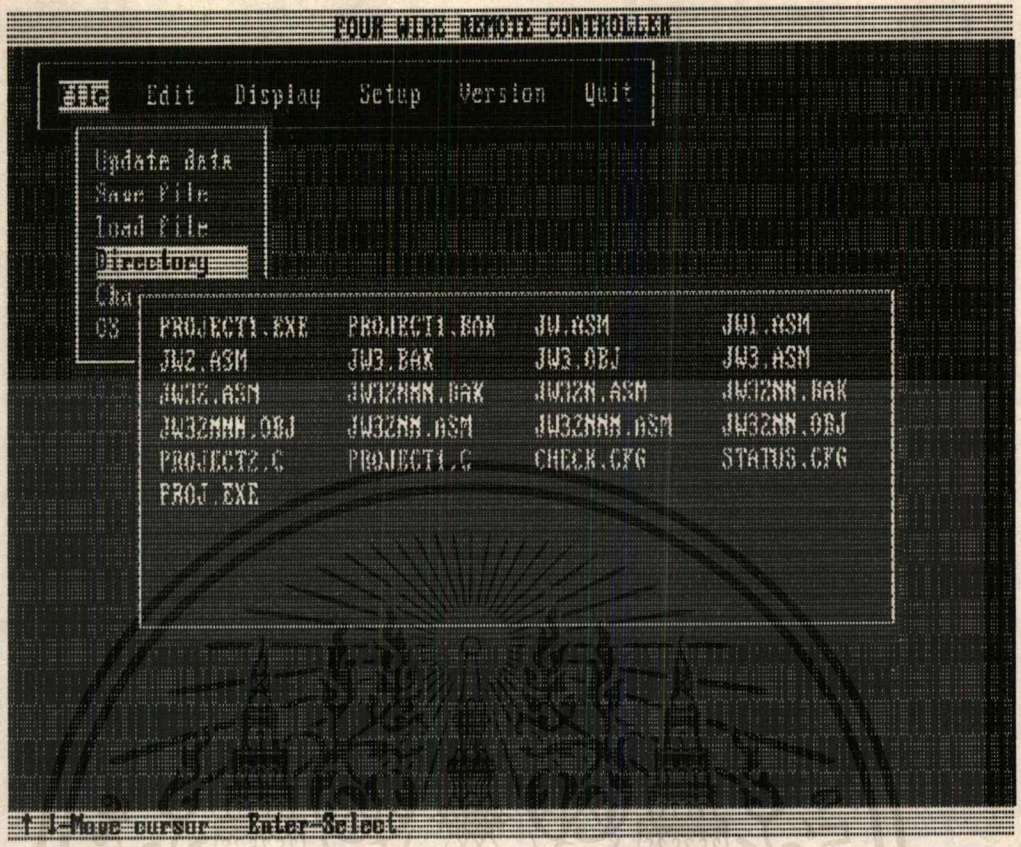


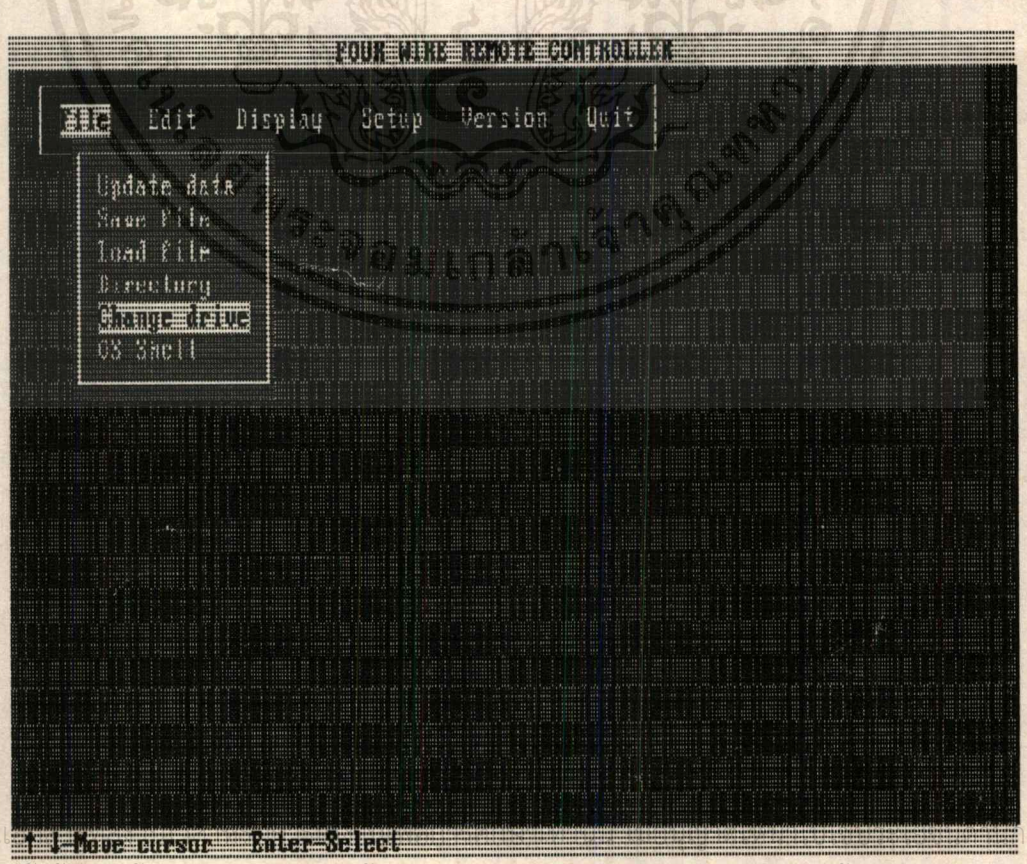


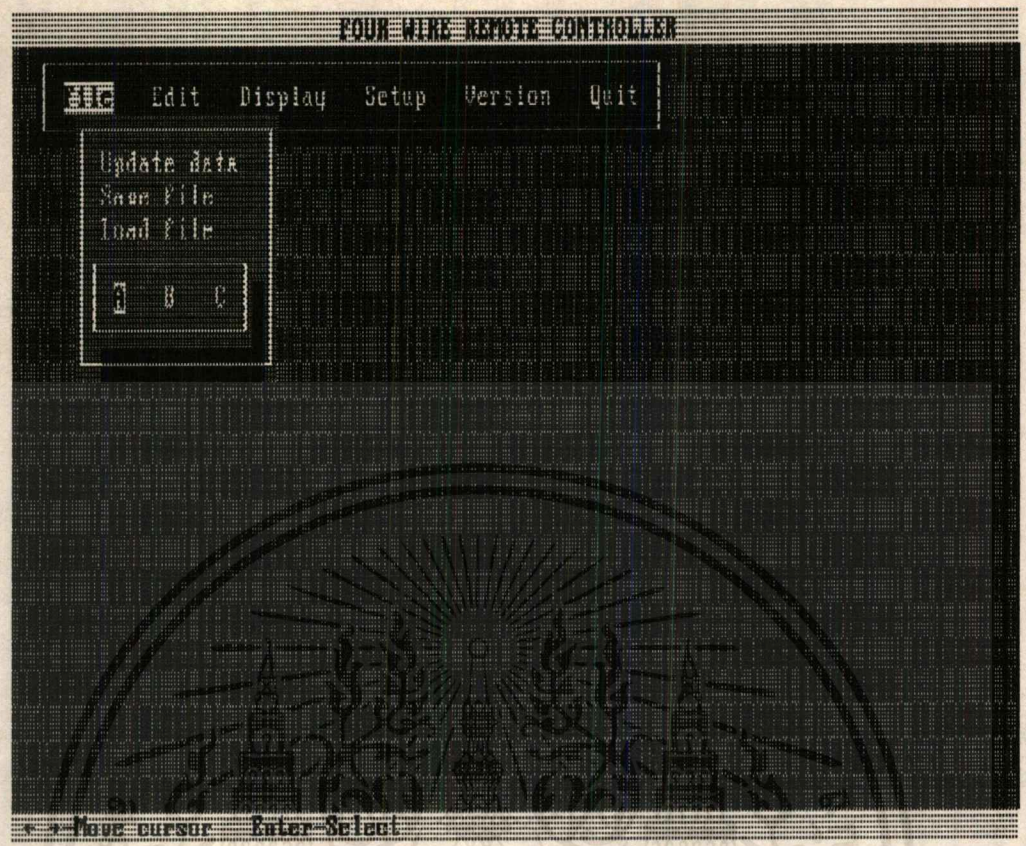


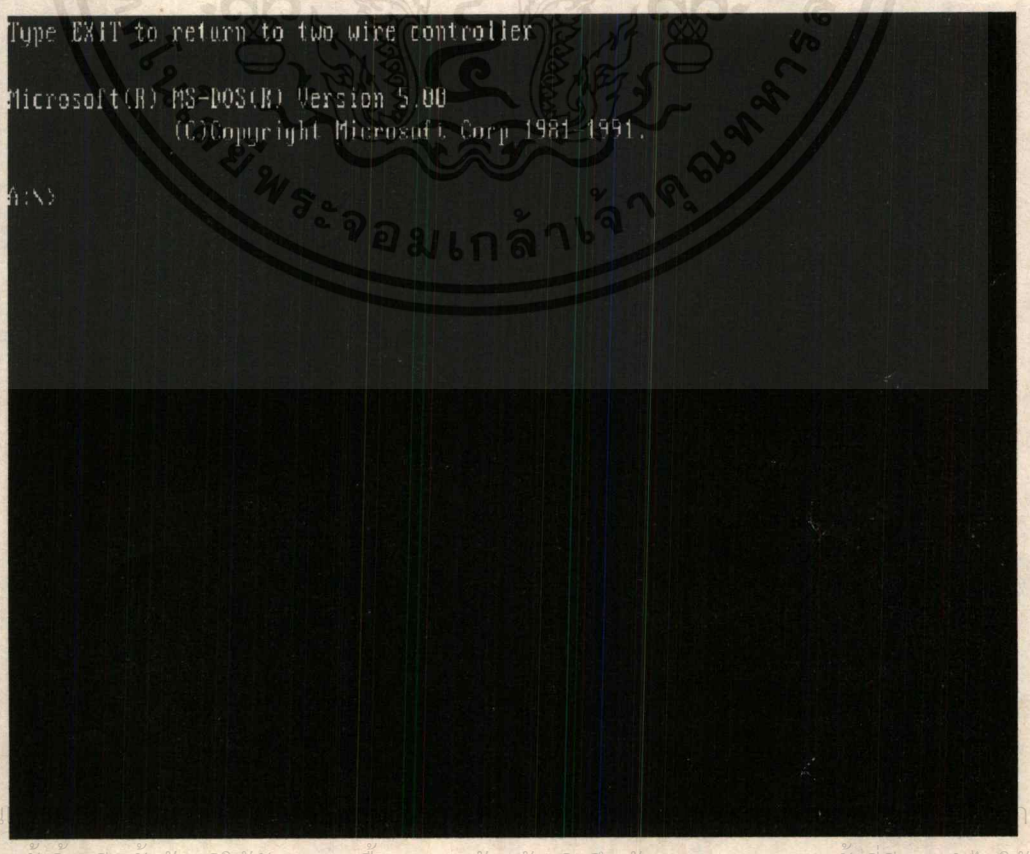


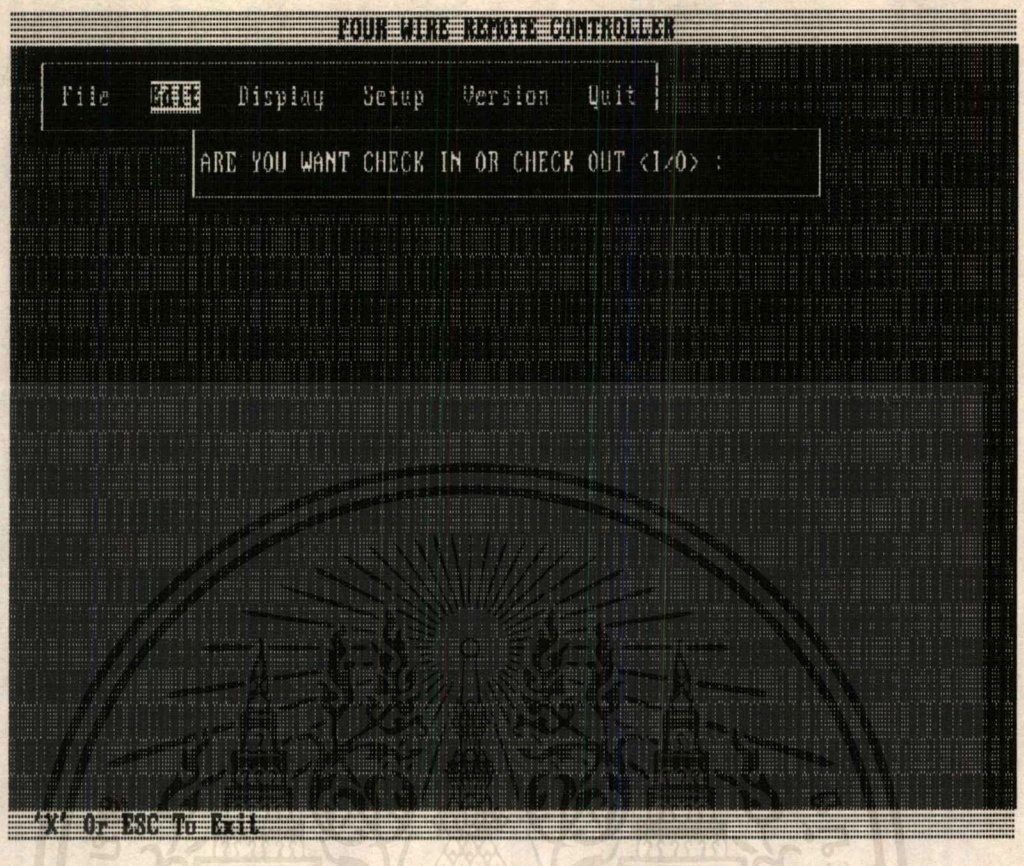


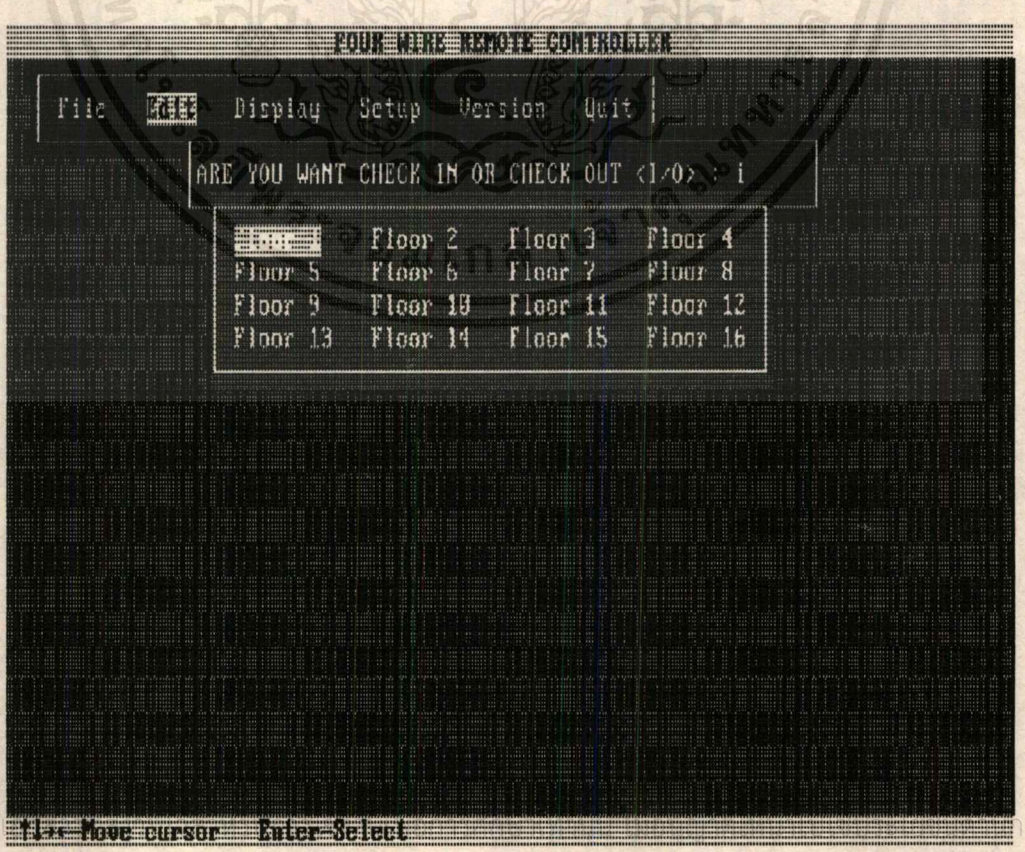


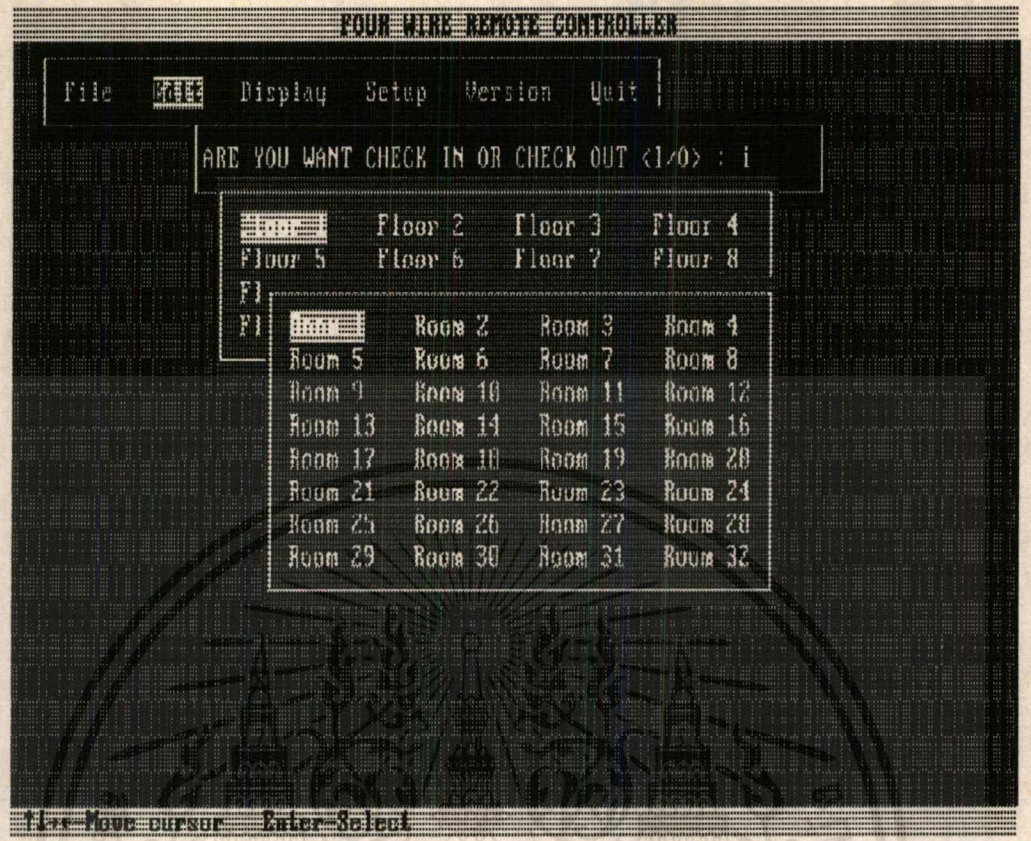


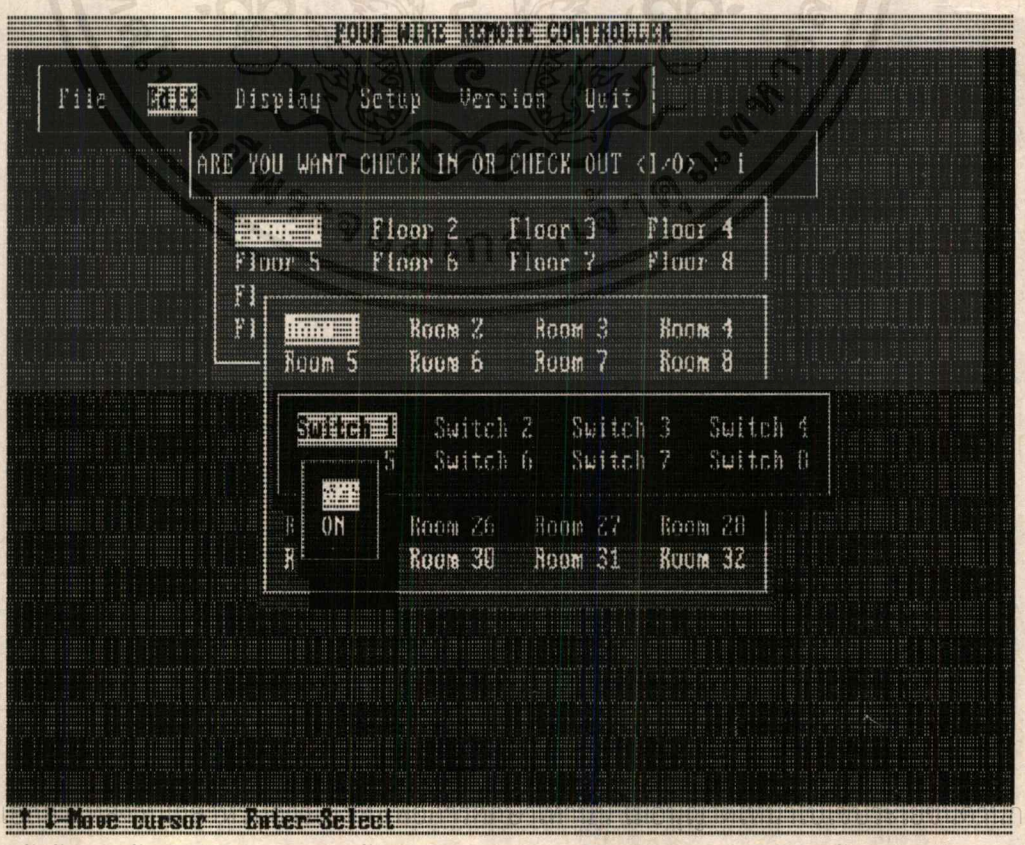


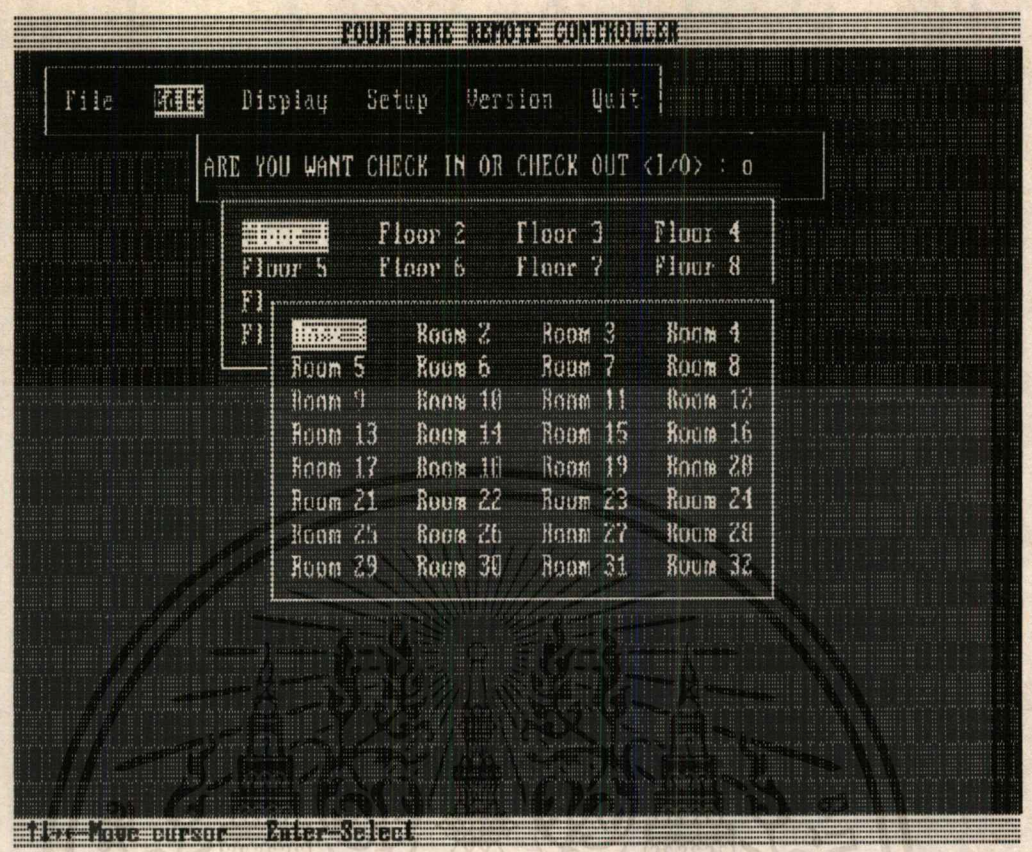


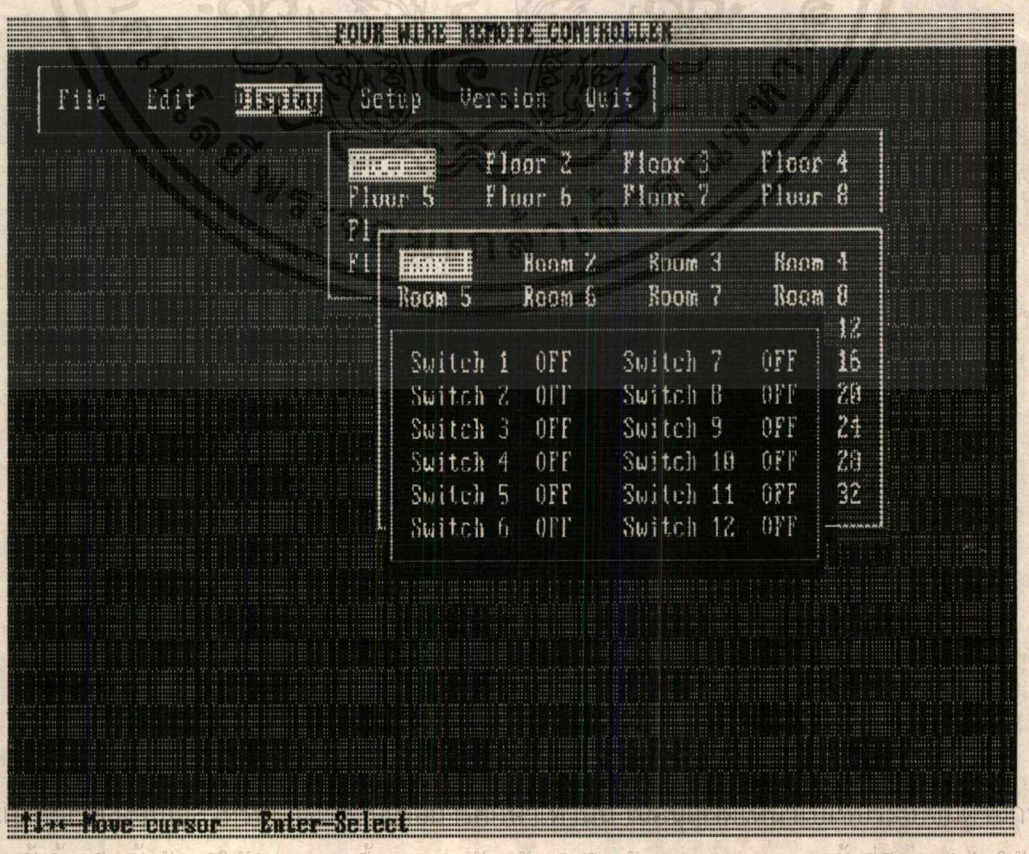


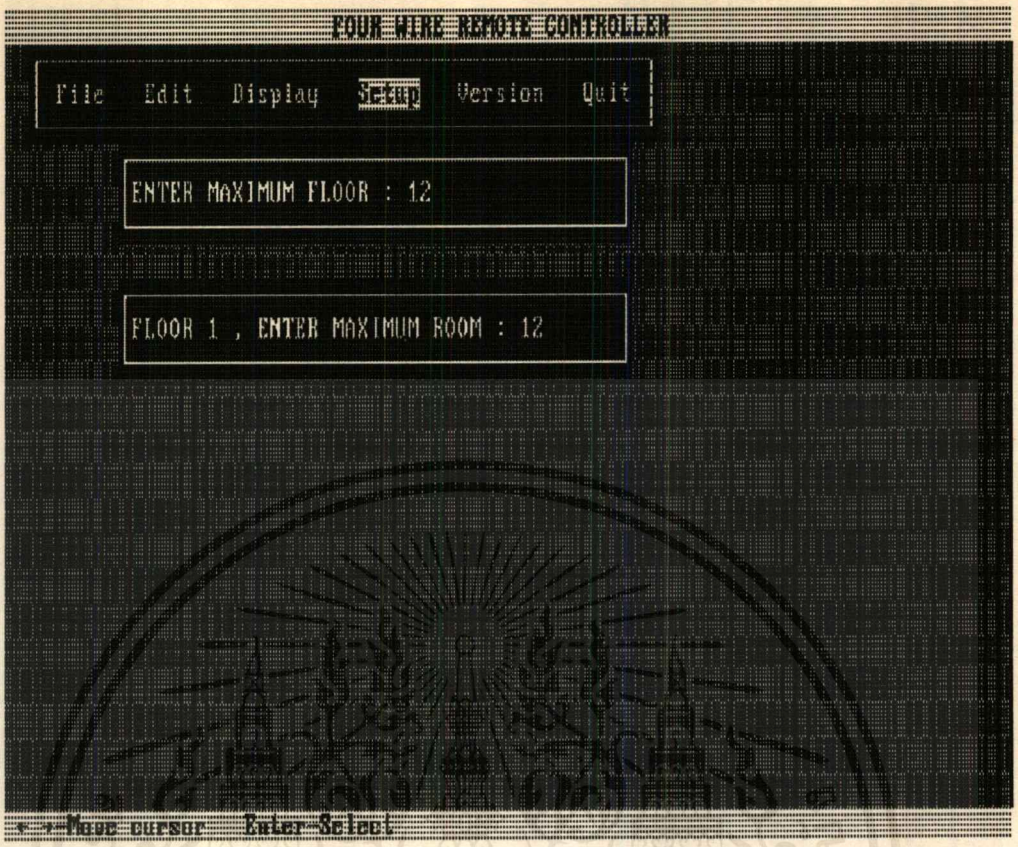


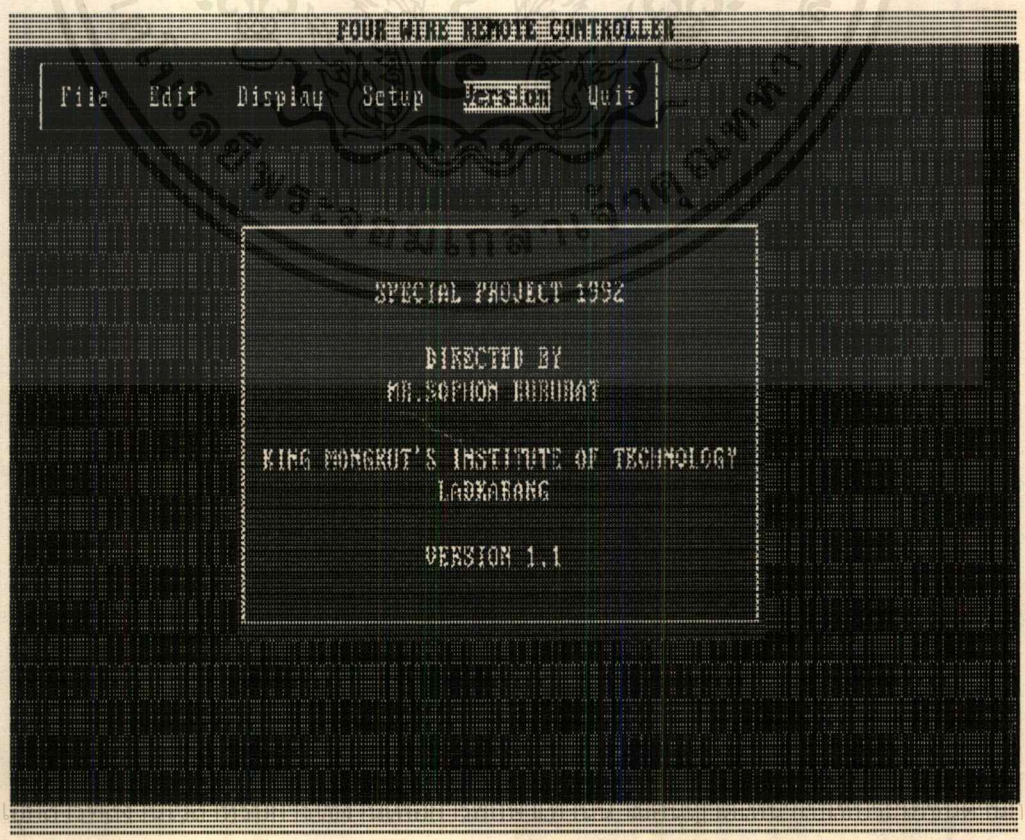












```
/* ------------------------------------------------------ *
 *            Two wire controller program                 *
 *              make for special project                  *
 *         of Industrial Electrical Technology            *
 *                  October.20,1992                       *
 *       King mongkut's institute of technology           *
 * ------------------------------------------------------ */

#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <conio.h>
#include <string.h>
#include <bios.h>
#include <biosarea.h>
#include <ctype.h>
#include <dos.h>
#include <dir.h>
#include <fcntl.h>
#include <c:\series.c>

/* ----- number of control room ----- */
#define FLOOR           16
#define ROOM            32
#define SWITCH          12

/* ------- setting delay ------- */
#define DELAY           0

/* ------ Cursor attrubute ----- */
#define _NOCURSOR       0x20
#define _NORMALCURSOR   0xDF

/* ------ Video attribute ------ */
#define REVERSE         0x70
#define NORMAL          0x07
#define HIGH            0x0F

/* ------- Maximum X,Y cordinate -------- */
#define MAXX            80
#define MAXY            25

/* ------- Shardow of window -------- */
#define FALSE           0
#define TRUE            1
#define BLANK           2

/* -------- ASCII code for control --------- */
#define BELL            0x07
#define ESC             0x1B
#define CR              0x0D
#define SPC             0x20

/* ------ arrow define ------ */
#define VER             1
#define HOR             2
#define BIDIR           3

/* -------- Number of window that it can opened --------- */
#define MAX_WINDOWS     9

/* ---------------- Address Cursor ----------------*/
BIOSDATA far *bios = MK_FP(0x0040,0);
/* --------- window definition structures ------------ */
struct text_info windows [MAX_WINDOWS];
```

```
struct text_info wkw;   /* working window */

/* ----------------- window dressing ----------------------- */
struct {
        int frame;       /* true if the window has a frame          */
        int shadow;      /* 0 if the window has no shadow           */
                         /* 1 if the window has a transparent shadow */
                         /* 2 if the window has an opaque shadow    */
        char *wsave;     /* points to the video memory save buffer  */
} dressing [MAX_WINDOWS];

/* ---------- stream of string that use for making menu ---------- */
typedef struct {
        int member;       /* number of string */
        int forg;         /* forground color */
        int backg;        /* background color */
        char *string[32]; /* maximum of string is 32 set */
} MENU;

struct ffblk *ffblk;

static char data[ROOM][SWITCH];  /* data memory use to hold configulation */

int flr = 16,
    rom[16] = {32,32,32,32,32,32,32,32,32,32,32,32,32,32,32,32};

int fl[16],r[16][32],sws[16][32][S][2];

int chk[16][32] ;
int te[1024] ;

int curr_wnd = 1 ;                  /* current window */

/* ----------------- STARTING THE PROGRAM ------------------ */
main( int argc , char *argv[])
{
        MENU menu = {6,YELLOW,BLUE,
        "File","Edit","Display","Setup","Version","Quit"}; /* main menu */
        static int x,y;             /* hold the current position */
        static int pos = 0;   /* hold the current menu */
        char ch ;
        int i;
        _setcursortype(_NOCURSOR); /* hidden cursor */
        textattr(NORMAL);
        clrscr();
        make_scrn();                /* make background screen */
           writeport("U");          /* loading data in borard z180 */
        for(i=0;i<1024;i++)
             te[i]=readport();
        load();                     /* loading configulation file to memory */
        cfrs();                     /* loading status each switch of room */
        version();                  /* display version of program */
        bottom("-Move cursor   Enter-Select",FALSE,HOR);
        if(argc == 2)
         if(strcmp(strupr(argv[1]),"/SETUP") == 0)
          setup();                  /* edit mumber of floor and room */
         do
         {
           ch = move_cursor(3,2,menu.member,1,menu,&pos,&x,&y);
           if(ch == CR)
           {
           /* checking which menu was select */
             switch(pos)
             {
                case 0 :
                        file();     /* working with file system */
```

```
                        break;
                case 1 :
                        edit();     /* edit switch in any room */
                        break;
                case 2 :
                        display();  /* display switch each room in floor */
                        break;
                case 3 :
                        setup();    /* edit mumber of floor and room */
                        break;
                case 4 :
                        version();  /* display version of program */
                        break;
              }
            }
          close_window();
          } while (pos != menu.member-1);  /* if selected QUIT then save */
          save();                          /* and exit to DOS */
          _setcursortype(_NORMALCURSOR);
          textattr(NORMAL);
          clrscr();
}

/* ----- display submenu of file and select it ----- */
file(void)
{
       /* submenu of file system */
        MENU files = {6,YELLOW,MAGENTA,
        "Update data ",
        "Save file   ",
        "Load file   ",
        "Directory   ",
        "Change drive",
        "OS Shell    "};
        static int current = 0;    /* hold current position of submenu */
        static int x,y;
        char ch;
        int  i;
        bottom("-Move cursor   Enter-Select",FALSE,VER);
        do
        {
           ch = move_cursor(6,4,1,files.member,files,&current,&x,&y);
           if (ch == CR)
           {
             switch (current)
             {
                case 0 :
                        bottom("Plase Wait. . .");
                        writeport("U");    /* loading data in borard z180 */
                        for(i=0;i<1024;i++)
                            te[i]=readport();
                        bottom("-Move cursor   Enter-Select",FALSE,VER);
                        break;
                case 1 :
                         save();           /* save configulation file */
                         break;
                case 2 :
                         load();           /* load configulation file */
                         break;
                case 3 :
                         dir();            /* display file on disk */
                         break;
                case 4 :
                         change();         /* change working drive */
                         break;
                case 5 :
```

```
                        shell();          /* exit to OS shell */
                        break;
                    }
                }
                close_window();
            } while (ch != ESC);
            bottom("-Move cursor   Enter-Select",FALSE,HOR);
}

/* ----- saving configulation file ----- */
save()
{
        FILE *out;
        MENU sav = {2,YELLOW,MAGENTA,"Save Status","Save Check"};
        char c,ch;
        int i,j,sa,x,y;
           ch = move_cursor(20,6,1,2,sav,&sa,&x,&y);
           if (ch==CR)
           {
                switch(sa)
                {
                  case 0 :
                        bottom("Saving Check Config File . . ."
                                                ,FALSE,FALSE);
                        out = fopen("c:CHECK.CFG","w");
                        if (out == NULL)
                        {
                          open_window(26,12,53,14,YELLOW,BROWN,1,TRUE);
                          cprintf(" Can not save config file");
                          for (i=400; i<=1000; i+=100) /* making sound */
                          {
                             sound(i);
                             delay(30);
                          }
                          nosound();
                          delay(1000);
                          close_window();
                          return;
                        }
                        for(i=0;i<FLOOR;i++)
                           for(j=0;j<ROOM;j++);
                              fputc(chk[i][j],out);
                        break;
                   case 1 :
                        bottom("Saving Status Config File . . ."
                                                ,FALSE,FALSE);
                        out = fopen("c:STATUS.CFG","w");
                        if (out == NULL)
                        {
                           open_window(26,12,53,14,YELLOW,BROWN,1,TRUE);
                           cprintf(" Can not save config file");
                           for (i=400; i<=1000; i+=100)
                           {                          /* making sound */
                            sound(i);
                            delay(30);
                           }
                           nosound();
                           delay(1000);
                           close_window();
                           return;
                        }
                        for(i=0;i<1024;i++)
                           fputc(te[i],out);
                        break;
                   }
           }
```

```
            close_window();
        bottom("-Move cursor   Enter-Select",FALSE,VER);
        fclose(out);
}

/* ----- loading configulation file ----- */
load()
{
        FILE *in;
        MENU lod = {2,YELLOW,MAGENTA,"Load Status","Load Check"};
        char c,ch;
        int i,j,la,x,y;
           ch = move_cursor(15,7,1,2,lod,&la,&x,&y);
           if (ch==CR)
           {
                switch(la)
                {
                   case 0:
                            bottom("Loading Config File . . .",FALSE,FALSE);
                            in = fopen("c:CHECK.CFG","r");
                            if (in == NULL)
                            {
                                open_window(26,12,53,14,YELLOW,BROWN,1,TRUE);
                                cprintf(" Can not load config file");
                                for (i=400; i<=1000; i+=100)
                                                        /* making sound */
                            {
                            sound(i);
                            delay(30);
                        }
                            nosound();
                            delay(1000);
                            close_window();
                            return;
                    }
                    for(i=0;i<FLOOR;i++)
                       for(j=0;j<ROOM;j++)
                       {
                          c = getc(in);
                          chk[i][j] = c;
                       }

                case 1 :
                          bottom("Loading Config File . . .",FALSE,FALSE);
                          in = fopen("c:STATUS.CFG","r");
                          if (in == NULL)
                          {
                             open_window(26,12,53,14,YELLOW,BROWN,1,TRUE);
                             cprintf(" Can not load config file");
                             for (i=400; i<=1000; i+=100)    /* making sound */
                             {
                                  sound(i);
                                  delay(30);
                             }
                             nosound();
                             delay(1000);
                             close_window();
                             return;
                          }
                          for(i=0;i<1024;i++)
                          {
                                  c = getc(in);
                                  te[i] = c;
                          }
                }
           }
}
```

```
            close_window();
        bottom("-Move cursor   Enter-Select",FALSE.VER);
        fclose(in);
}

/* ----- exit to DOS shell ----- */
shell()
{
        _setcursortype(_NORMALCURSOR);
        open_window(1,1,80,25,YELLOW,BLACK.0,0);
        clrscr();
        cprintf("Type EXIT to return to two wire controller");
        system("\command");
        close_window();
        _setcursortype(_NOCURSOR);
}

/* ----- change working drive ----- */
change()
{
        int drive;
        static int x,y;
        static int current = 0;
        MENU drv = {7,YELLOW,LIGHTRED,
        "A","B","C","D","E","F","G"};
        bottom("-Move cursor   Enter-Select",FALSE,HOR);
        current = getdisk();
        drv.member = getdrive() + 1;
        move_cursor(7,8,drv.member,1,drv,&current,&x,&y);
        setdisk(current);  /* set working drive */
        close_window();
        bottom("-Move cursor   Enter-Select",FALSE,VER);
}

/* ----- list directory ----- */
dir(void)
{
        int fi,x,y;
        char ch;
        x = 2; y = 1;
        window(0,0,79,24);
        open_window(11,9,71,19,YELLOW,MAGENTA,1,TRUE);
        /* --- find directory --- */
        fi = findfirst("*.*",ffblk,0x01);
        /* --- display file on disk --- */
        do
        {
              if (fi == FALSE)
              {
                 gotoxy(x,y);
                 cprintf("%s",ffblk->ff_name);
                 x += 15;
                 if (x > 61)
                 {
                    y += 1;
                    if (y == 10)
                    {
                       ch = getch();
                       if (ch ==0)
                          ch = getch();
                       clrscr();
                       y = 1;
                    }
                    x = 2;
                 }
              }
```

```
            fi = findnext(ffblk);
        } while (fi == FALSE);
        /* --- press any key to exit --- */
        ch = getch();
        if (ch == 0)
        ch = getch();
        close_window();
}

/* ----- set switch of each room ----- */
edit(void)
{
        /*number of floor */
        MENU floor = {16,YELLOW,MAGENTA,
        "Floor 1","Floor 2","Floor 3","Floor 4",
        "Floor 5","Floor 6","Floor 7","Floor 8",
        "Floor 9","Floor 10","Floor 11","Floor 12",
        "Floor 13","Floor 14","Floor 15","Floor 16"};
        /* number of floor */

        /* number of room */
        MENU room   = {32,YELLOW,MAGENTA,
        "Room 1","Room 2","Room 3","Room 4",
        "Room 5","Room 6","Room 7","Room 8",
        "Room 9","Room 10","Room 11","Room 12",
        "Room 13","Room 14","Room 15","Room 16",
        "Room 17","Room 18","Room 19","Room 20",
        "Room 21","Room 22","Room 23","Room 24",
        "Room 25","Room 26","Room 27","Room 28",
        "Room 29","Room 30","Room 31","Room 32"};
        /* number of switch */

        MENU sw = {12,LIGHTRED,BLUE,
        "Switch 1","Switch 2","Switch 3","Switch 4",
        "Switch 5","Switch 6","Switch 7","Switch 8",
        "Switch 9","Switch 10","Switch 11","Switch 12"};
        /* switch status */
        MENU status = {2,LIGHTRED,BLUE,"OFF","ON "};
        char ch1,ch2,ch3,ch4; /* value from keyboard */
        int temp; /* use for check which bit is on/off */
        int py,check,aress,poer,frs[2];
        int flor,rm,swit,sta,i=0;
        int x,y,x1,y1,x2,y2,x3,y3,x4,y4;
        do
        {
        bottom(" 'X' Or ESC To Exit ",FALSE,FALSE);
        open_window(15,4,65,6,WHITE,BLUE,1,FALSE);
        cprintf("ARE YOU WANT CHECK IN OR CHECK OUT <I/O> : ");
        check = getche();
        if(check == 'I' || check == 'i' || check == 'O' || check == 'o')
        {
        if(check == 'I' || check == 'i')
           poer=1;
        else
            poer=0;
        bottom("-Move cursor   Enter-Select",FALSE,BIDIR);
        x1 = 0;
        y1 = 0;
        flor = 0;
        do
        {
        if(flr % 4)
            py = flr / 4 + 1;
        else
            py = flr / 4 ;
        ch1 = move_cursors(flr,fl,17,6,4,py,floor,&flor,&x1,&y1);
```

```
if (ch1 == CR)
{
 x2 = 0;
 y2 = 0;
 rm = 0;
 do
 {
  cfrs();
  if(rom[flor] % 4)
     py = rom[flor] / 4 + 1 ;
  else
     py = rom[flor] / 4 ;
  ch2 = move_cursors(rom[flor],r[flor],21,9,4,py,room,&rm,&x2,&y2);
  if(ch2 == CR)
  {
   if (check == 'O' || check == 'o')
   {
       chk[flor][rm] = WHITE;
       r[flor][rm]   = WHITE;
   }
   else
   {
       chk[flor][rm] = YELLOW;
       r[flor][rm]   = YELLOW;
   }
   x3 = 0;
   y3 = 0;
   swit = 0;
   do
   {
    ch3 = move_cursor(22,12,4,3,sw,&swit,&x3,&y3);
    if(ch3 == CR)
    {
     x4=0;
     y4=0;
     sta = 0;
     do
     {
      bottom("-Move cursor   Enter-Select",FALSE,VER);
      x =  (x3*12)+16+strlen(sw.string[0]);
      y =  y3+14;
      ch4 = move_cursors(2,sws[flor][rm][swit],x,y,1,2,status,&sta,&x4,&
      if (ch4 == CR)
      {
        sws[flor][rm][swit][sta]    = YELLOW;
        sws[flor][rm][swit][abs(sta-1)] = WHITE;
        aress= flor*0x40+rm*2+swit/8;
        temp = (int)((te[aress]>>(7-(swit&7)))&1);
        if(temp != sta)
         if(sta == 1)
           te[aress]|=power(2,7-swit&7);
         else
              te[aress]&=((~power(2,7-swit&7))&0xff);
        frs[0]=((((poer<<1|sta)<<1)|1)<<5)|rm;
        frs[1]=(flor<<4)|swit;
        writeport("E%c%c%c%c",frs[0],frs[1],frs[0]&0xdf,frs[1]);
        i++;
      }
      close_window();
      bottom("-Move cursor   Enter-Select",FALSE,BIDIR);
     } while ( ch4 != ESC);
    }
    close_window();
   } while (ch3 != ESC);
   bottom("-Move cursor   Enter-Select",FALSE,BIDIR);
}
```

```
            close_window();
            bottom("-Move cursor   Enter-Select",FALSE,BIDIR);
           } while (ch2 != ESC);
           bottom("-Move cursor   Enter-Select",FALSE,BIDIR);
          }
          close_window();
          bottom("-Move cursor   Enter-Select",FALSE,BIDIR);
         }while(ch1 != ESC);
         }
         close_window();
         bottom(" 'X' Or ESC To Exit ",FALSE,FALSE);
         }while(check != 'X'&& check != 'x'&& check != ESC);
         bottom("-Move cursor   Enter-Select",FALSE,HOR);
}
/* ----- display switch of each room ----- */
display(void)
{
        /*number of floor */
        MENU floor = {16,YELLOW,MAGENTA,
        "Floor 1","Floor 2","Floor 3","Floor 4",
        "Floor 5","Floor 6","Floor 7","Floor 8",
        "Floor 9","Floor 10","Floor 11","Floor 12",
        "Floor 13","Floor 14","Floor 15","Floor 16"};
        /* number of floor */

        /* number of room */
        MENU room   = {32,YELLOW,MAGENTA,
        "Room 1","Room 2","Room 3","Room 4",
        "Room 5","Room 6","Room 7","Room 8",
        "Room 9","Room 10","Room 11","Room 12",
        "Room 13","Room 14","Room 15","Room 16",
        "Room 17","Room 18","Room 19","Room 20",
        "Room 21","Room 22","Room 23","Room 24",
        "Room 25","Room 26","Room 27","Room 28",
        "Room 29","Room 30","Room 31","Room 32"};
        /* number of switch */

        MENU sw = {12,YELLOW,BLUE,
        "Switch 1","Switch 2","Switch 3","Switch 4",
        "Switch 5","Switch 6","Switch 7","Switch 8",
        "Switch 9","Switch 10","Switch 11","Switch 12"};
        /* switch status */
        char *status[] = {"ON ","OFF"};
        char ch1,ch2,ch3; /* value from keyboard */
        int temp; /* use for check which bit is on/off */
        int py,count;
        int flor,rm,aress;
        int x1,y1,x2,y2;
        bottom("-Move cursor   Enter-Select",FALSE,BIDIR);
        x1 = 0;
        y1 = 0;
        flor = 0;
        do
        {
         if(flr % 4)
            py = flr / 4 + 1 ;
         else
            py = flr / 4 ;
         ch1 = move_cursors(flr,fl,26,4,4,py,floor,&flor,&x1,&y1);
         if (ch1 == CR)
         {
         x2 = 0;
         y2 = 0;
         rm = 0;
          do
          {
```

```
            if(rom[flor] % 4)
               py = rom[flor] / 4 + 1 ;
            else
               py = rom[flor] / 4 ;
            ch2 = move_cursors(rom[flor],r[flor],30,7,4,py,room,&rm,&x2,&y2);
            if(ch2 == CR)
            {
              open_window(31,10,65,17,LIGHTRED,BLUE,1,TRUE);
              do
              {
                /* printing switch status */
                for(count=0;count<sw.member/2;count++)
                {
                 aress= flor*0x40+rm*2;
                 temp = (int)((te[aress]>>(7-count))&1);
                 gotoxy(2,count+1);
                 textcolor(sw.forg);
                 textbackground(sw.backg);
                 cprintf("%s  %s    %s",sw.string[count],temp ?
                                        status[0]:status[1],
                                        sw.string[count+6]);
                 aress = flor*0x40+rm*2+(count+6)/8;
                 if(count > 1)
                    temp = (int)((te[aress]>>(9-count))&1);
                 else
                    temp = (int)((te[aress]>>(1-count))&1);
                 gotoxy(30,count+1);
                 cprintf("%s",temp ? status[0]:status[1]);
                }
                ch3 =getch();
              } while (ch3 != ESC);
              close_window();
              bottom("-Move cursor   Enter-Select",FALSE,BIDIR);
            }
            close_window();
            bottom("-Move cursor   Enter-Select",FALSE,BIDIR);
           } while (ch2 != ESC);
           bottom("-Move cursor   Enter-Select",FALSE,BIDIR);
          }
          close_window();
          bottom("-Move cursor   Enter-Select",FALSE,BIDIR);
         }while(ch1 != ESC);
         bottom("-Move cursor   Enter-Select",FALSE,HOR);
}

/* edit mumber of floor and room */
setup(void)
{
    int i , ch;
    open_window(10,5,50,7,WHITE,BLUE,1,FALSE);
    ch = ipu(16,38,&flr,"ENTER MAXIMUM FLOOR : ");
    if ( ch == ESC)
         flr = 16 ;
    open_window(10,9,50,11,WHITE,BLUE,1,FALSE);
    for(i=0;i < flr; i++)
    {
        clrscr();
        ch = ipu(32,38,&rom[i],"FLOOR %d , ENTER MAXIMUM ROOM : ", i+1);
        if ( ch == ESC )
             rom[i] = 32 ;
    }
    close_window();
    close_window();
}

cfrs(void)
```

```
{
int tem,i=0,j,k,l,equ=0x80,sfl=1;

        for(j=0;j<16;j++)
        {
         for(k=0;k<32;k++)
         {
          for(l=0; l < 8 ;l++)
          {
                tem=te[i];
                tem=(tem&equ)>>(7-l);
                if(tem==0)
                {
                        sws[j][k][l][0]=YELLOW;
                        sws[j][k][l][1]=WHITE;
                }
                else
                {
                        sws[j][k][l][0]=WHITE;
                        sws[j][k][l][1]=YELLOW;
                }
                equ>>=1;
          }
          r[j][k] = chk[j][k];
          if(r[j][k]==YELLOW)
                sfl&=1;
          else
                sfl&=0;
          equ=0x80;
          i+=2;
         }
         if(sfl==1)
                fl[j]=YELLOW;
         else
                fl[j]=WHITE;
         sfl=1;
        }
}
/* ----- display version of program ----- */
version(void)
{
        int j = 0;
        int i,count;
        char *text[] = {"SPECIAL PROJECT 1992",
                        "DIRECTED BY",
                        "MR.SOPHON KURURAT",
                        "KING MONGKUT'S INSTITUTE OF TECHNOLOGY",
                        "LADKABANG",
                        "VERSION 1.1"};
        char *temp;
        bottom("                                                ",FALSE,FALSE);
        open_window(19,7,60,19,LIGHTGREEN,LIGHTRED,1,TRUE);
        for (i=2; i<=10; i++)
        {
                if (i%3)
                {
                        for (count=1; count<=strlen(text[j]); count++)
                        {
                            temp = slide(text[j],count);
                            sound(50);
                            delay(20);
                            nosound();
                            gotoxy((50-8)/2+strlen(text[j])/2-count,i);
                            cprintf("%s",temp);
                        }
                        j++;
```

```
                }
        }
        getch();
        close_window();
        bottom("-Move cursor   Enter-Select",FALSE,HOR);
}

/* -------------- display the pull down menu ----------------- */
int move_cursor(int startx,int starty.int dimx,int dimy,
                MENU menu,int *current,int *x,int *y)
{
        int i,j,len;
        int temp = 0;
        int width = 0;
        char ch = 0;
        int posx,posy;
        /* caculate length of string */
        for (i=0; i<menu.member; i++)
        {
            len = strlen(menu.string[i]);
            width += len;
            width += 3;
            if (len > temp)
               temp = len;
        }
        len = temp;
        if (dimy != 1)
              posx = startx+dimx*len+dimx*3;
        else
              posx = startx+width;
        posy = starty+dimy+1;
        open_window(startx,starty,posx,posy,menu.forg,menu.backg,1,TRUE);
        do
        {
           *current = *y * dimx + *x;     /* caculate position of cursor */
           posx = 2; posy = 1;
           textcolor(menu.forg);
           textbackground(menu.backg);
/* --------- Display bar menu ---------- */
           for (i=0; i<menu.member; i++)
           {
                textattr(NORMAL);
                textbackground(menu.backg);
                gotoxy(posx,posy);
                if (i == *current)
                {
                   textattr(REVERSE);
                }
                cprintf("%s",menu.string[i]);
                if (dimy != 1)
                        posx = posx+len+3;
                else
                        posx = posx+strlen(menu.string[i])+3;
                if (posx > dimx*len+dimx*3)
                {
                   posy++;
                   posx = 2;
                }
                if (posy > dimy)
                   posy = 1;
           }
           ch = getch();
           if (ch == 0)
                ch = getch();
           /* checking arrow key use for moving cursor */
           switch (ch)
```

```
            {
                case 0x4b :
                            if (dimx > 1)    /* move left */
                            {
                               (*x)--;
                               if ((*x) < 0)
                                  (*x) = dimx-1;
                            }
                            break;
                case 0x4d :
                            if (dimx > 1)    /* move right */
                            {
                               (*x)++;
                               if ((*x) > dimx-1)
                                  (*x) = 0;
                            }
                            break;
                case 0x50 :
                            if (dimx > 1)    /* move down */
                            {
                               (*y)++;
                               if ((*y) > dimy-1)
                                  (*y) = 0;
                            }
                            else
                            {
                               (*x)++;
                               if ((*x) > dimy-1)
                                  (*x) = 0;
                            }
                            break;
                case 0x48 :
                            if (dimx > 1)    /* move up */
                            {
                               (*y) -= 1;
                               if ((*y) < 0)
                                  (*y) = dimy-1;
                            }
                            else
                            {
                               (*x)--;
                               if ((*x) < 0)
                                  (*x) = dimy-1;
                            }
                            break;
            }
            if (*current > menu.member)
               *current = menu.member-1;
        } while (ch != ESC && ch != CR);
        return(ch);
}

/* ------ display the pull down menu for maximum floor and room ------ */
int move_cursors(int max,int st[],int startx,int starty,int dimx,int dimy,
                 MENU menu,int *current,int *x,int *y)
{
        int i,j,len,s;
        int temp = 0;
        int width = 0;
        char ch = 0;
        int posx,posy;
        /* caculate length of string */
     for (i = 0; i < max; i++)
        {
            len = strlen(menu.string[i]);
            width += len;
```

```
            width += 3;
            if (len > temp)
               temp = len;
         }
         len = temp;
         if (dimy != 1)
                posx = startx+dimx*len+dimx*3;
         else
                posx = startx+width;
         posy = starty+dimy+1;
         open_window(startx,starty,posx,posy,menu.forg,menu.backg,1,TRUE);
         do
         {
            *current = *y * dimx + *x;     /* caculate position of cursor */
            posx = 2; posy = 1;
            textcolor(menu.forg);
            textbackground(menu.backg);
/* --------- Display bar menu ---------- */
            for (i=0; i<max ; i++)
            {
                 textattr(st[i]);
                 textbackground(menu.backg);
                 gotoxy(posx,posy);
                 if (i == *current)
                 {
                    textattr(REVERSE | st[i]);
                 }
                 cprintf("%s",menu.string[i]);
                 if (dimy != 1)
                       posx = posx+len+3;
                 else
                       posx = posx+strlen(menu.string[i])+3;
                 if (posx > dimx*len+dimx*3)
                 {
                    posy++;
                    posx = 2;
                 }
                 if (posy > dimy)
                    posy = 1;
            }
            ch = getch();
            if (ch == 0)
                ch = getch();
            /* checking arrow key use for moving cursor */
            switch (ch)
            {
        case 0x4B :
                    if (dimx > 1)    /* move left */
                    {
                      (*x)--;
                      if ((*x) < 0 && (*y) == max / dimx && max % dimx)
                        (*x) = max % dimx - 1;
                      else if ((*x) < 0)
                        (*x) = dimx-1;
                           }
                           break;
        case 0x4D :
                    if (dimx > 1)    /* move right */
                    {
                      (*x)++;
                      if (((*x) > dimx-1) || ((*x) == max % dimx) && ((*y) == max
                        (*x) = 0;
                    }
                    break;

        case 0x48 :
```

```
                    if (dimx > 1)    /* move up */
                    {
                      (*y) -- ;
                      if ((*y) < 0 && ((*x) < max % dimx || max % dimx == 0))
                         (*y) = dimy-1;
                      else if((*y) < 0 && (*x) >= max % dimx)
                         (*y) = max / dimx - 1;
                    }
                    else
                    {
                      (*x)--;
                      if ((*x) < 0)
                         (*x) = dimy-1;
                    }
                    break;

          case 0x50 :
                    if (dimx > 1)    /* move down */
                    {
                      (*y)++;
                      if (((*y) > dimy-1) && ((*x) < max % dimx) ||
                          (((*y) >= max /dimx) && ((*x) >= max % dimx)))
                            (*y) = 0;
                    }
                    else
                    {
                      (*x)++;
                      if ((*x) > dimy-1)
                        (*x) = 0;
                    }
                    break;
                 }
               if (*current > max)
                  *current = max - 1;
               } while (ch != ESC && ch != CR);
               return(ch);
}

/* ---------- input maximum of floor and room -------- */
ipu(int mx,int mc,int *mum, char *fmt, ...)
{
    va_list  argptr ;
    int i = 0, again = TRUE,xi;
    char s[140],m[140],d;
    va_start( argptr , format ) ;
    vsprintf( s , fmt , argptr ) ;
    cprintf("%s" , s ) ;
    memset(m,'\0',140);
    xi = wherex();
    do
    {
        d = getch();
        switch(d)
        {
            case '0' :
            case '1' :
            case '2' :
            case '3' :
            case '4' :
            case '5' :
            case '6' :
            case '7' :
            case '8' :
            case '9' : if (wherex() <= mc)
                       {
                        cprintf("%c",d);
```

```
                        m[i] = d;
                        m[i+1] = '\0';
                        i++;
                       }
                        break;
              case CR :  *mum = atoi(m);
                        if(*mum > mx || *mum < 0)
                        {
                          clrscr();
                          cprintf("INVALIC NUMBER, PRESS ANY KEY TO AGAIN");
                          getch();
                          clrscr();
                          cprintf("%s",s);
                          i = 0;
                          memset(m,'\0',140);
                          again = TRUE;
                        }
                        else if ( *mum == 0 )
                        {
                          *mum = mx ;
                          again = FALSE;
                        }
                        else
                          again = FALSE;
                        break;
             case  0 :  d = getch();
                        if(d == 0x4B)
                        {
                         if(xi < wherex() - 1)
                         {
                            gotoxy(wherex() - 1,wherey());
                            cprintf("%c",' ');
                            gotoxy(wherex() - 1,wherey());
                         }
                         else
                         {
                            gotoxy(xi,wherey());
                            cprintf("%c",' ');
                            gotoxy(xi,wherey());
                         }
                         i--;
                         m[i] = '\0';
                        }
                        break;
        }
    }while(again && d != ESC);
    return(d);
}


/* ---------------- Make screen ----------------- */
make_scrn()
{
        char *bar = " TWO WIRE REMOTE CONTROLLER ";
        putasc(0xb1,NORMAL,LIGHTBLUE,(24*80));        /* Fill screen */
        gotoxy(1,1);
        top(bar);
}

/* --------- display at top of screen ---------- */
top(char *bar)
{
        int len;
        textattr(REVERSE);
        gotoxy(1,1);
```

```
        clreol();
        len = strlen(bar);
        gotoxy((MAXX/2-len/2),1);
        cprintf("%s",bar);                      /* -- display title -- */
}
/* ---------- display at bommtom of screen ------------ */
bottom(char *ex,int flg,int arrow)
{
        window(1,1,80,25);
        gotoxy(1,25);
        textattr(REVERSE);
        clreol();
        if (flg == TRUE)
                gotoxy((MAXX/2)-(strlen(ex)/2),25);
        else
                gotoxy(2,25);
        if (arrow == HOR)
        {
                putch(27);
                putch(0x20);
                putch(26);
        }
        if (arrow == VER)
        {
                putch(24);
                putch(0x20);
                putch(25);
        }
        if (arrow == BIDIR)
        {
                putch(24);
                putch(25);
                putch(26);
                putch(27);
        }
        cprintf("%s",ex);                       /* -- display help -- */
}

/* ----------- open a new window ------------- */
open_window(left,top,right,bottom,foreg,backg,frame,shadow)
{
        int bfsize;
        int sinc = 0;

        for (sinc=1500; sinc<=2000; sinc ++)
                sound(sinc);
        nosound();
        sinc = 0;
        if (shadow && right < 80 && bottom < 25)
                sinc = 1;
        bfsize = (bottom-top+1+sinc) * (right-left+1+sinc) * 2;
        if (curr_wnd < MAX_WINDOWS)
        {
                if (curr_wnd == 1)
                        gettextinfo(windows);
                else
                {
                        windows[curr_wnd-1].curx = wherex();
                        windows[curr_wnd-1].cury = wherey();
                }
                if ((dressing[curr_wnd].wsave=malloc(bfsize))!=NULL)
                {
                        gettext(left, top, right+sinc, bottom+sinc,
                                dressing[curr_wnd].wsave);
                        window(left,top,right,bottom);
```

```
                        textcolor(foreg);
                        textbackground(backg);
                        gettextinfo(&wkw);
                        dressing[curr_wnd].frame = frame;
                        dressing[curr_wnd].shadow = shadow;
                        clear_window();
                        windows[curr_wnd++] = wkw;
                }
        }
}

/* --------- window frame characters ---------------- */
#define NW   (dressing[curr_wnd].frame == 1 ? '\xda' : '\xc9')
#define NE   (dressing[curr_wnd].frame == 1 ? '\xbf' : '\xbb')
#define SE   (dressing[curr_wnd].frame == 1 ? '\xd9' : '\xbc')
#define SW   (dressing[curr_wnd].frame == 1 ? '\xc0' : '\xc8')
#define SIDE (dressing[curr_wnd].frame == 1 ? '\xb3' : '\xba')
#define LINE (dressing[curr_wnd].frame == 1 ? '\xc4' : '\xcd')

/* ------- blank the window and draw its frame -------- */
clear_window(void)
{
        int x, y;
        int ht = wkw.winbottom - wkw.wintop + 1;
        int wd = wkw.winright - wkw.winleft + 1;
        char line[81];

        clrscr();
        if (dressing[curr_wnd].shadow)
        {
                textcolor(LIGHTGRAY);
                textbackground(BLACK);
                window(wkw.winleft,wkw.wintop,wkw.winright+1,
                                        wkw.winbottom+2);
                for (y = 2; y < ht+1; y++)
                {
                        gotoxy(wd+1,y);
                        putch(dressing[curr_wnd].shadow == 2 ? ' ' :
                                *(dressing[curr_wnd].wsave+((wd+1)*y-1)*2));
                }
                gotoxy(2, ht+1);
                for (x = 0; x < wd; x++)
                        putch(dressing[curr_wnd].shadow == 2 ? ' ' :
                        *(dressing[curr_wnd].wsave+(((wd+1)*ht+1)+x)*2));
                window(wkw.winleft,wkw.wintop,wkw.winright,
                                        wkw.winbottom);
                textattr(wkw.attribute);
        }
        if (dressing[curr_wnd].frame)
        {
                window(wkw.winleft,wkw.wintop,wkw.winright,
                                        wkw.winbottom+1);
                memset(line+1,LINE,wd-2);
                line[0] = NW;
                line[wd-1] = NE;
                line[wd] = '\0';
                cputs(line);
                for (y = 2; y < ht; y++)
                {
                        gotoxy(1,y);
                        putch(SIDE);
                        gotoxy(wd,y);
                        putch(SIDE);
                }
                line[0] = SW;
                line[wd-1] = SE;
```

```
                cputs(line);
                window(wkw.winleft+1,wkw.wintop+1,wkw.winright-1,
                                        wkw.winbottom-1);
                wkw.curx = wkw.cury = 1;
                gotoxy(1, 1);
        }
}

/* ------------ close a window ------------------ */
close_window(void)
{
        int sinc = 0;

        if (dressing[curr_wnd-1].shadow)
                sinc = 1;
        if (curr_wnd > 1)
        {
                puttext(wkw.winleft,wkw.wintop,
                        wkw.winright+sinc,
                        wkw.winbottom+sinc,
                        dressing[curr_wnd-1].wsave);
                free(dressing[curr_wnd-1].wsave);
                wkw = windows[(--curr_wnd)-1];
                textattr(wkw.attribute);
                if (dressing[curr_wnd-1].frame)
                        window(wkw.winleft+1,wkw.wintop+1,
                                wkw.winright-1,wkw.winbottom-1);
                else
                        window(wkw.winleft,wkw.wintop,wkw.winright,
                                wkw.winbottom);
                gotoxy(wkw.curx, wkw.cury);
        }
}

/* ----------------- Put a character on screen ----------------- */
putasc(unsigned char ascii,unsigned char attr,
       unsigned char backg,unsigned int count)
{
        textcolor(backg);
        _AL = ascii;
        _BL = attr;
        _CX = count;
        _BH = 0;
        _AH = 9;
        geninterrupt(0x10);
}

getdrive()
{
    _CX = 6;
    do
    {
        _CX -= 1;
        _AH = 0x0E;
        _DX = _CX;
        geninterrupt(0x21);
        _AH = 0x19;
        geninterrupt(0x21);
        _AH = 0;
    } while (_AX != _CX);
    return(_AX);
}

slide(char *string,int len)
{
        char *str;
```

```
        int i;
        for (i=0; i<len; i++)
        {
            str[i] = *string++;
        }
        str[i] = '\0';
        return(str):
}

_setcursortype(int status)
{
  union REGS reg;
     if ( status == 0x20 )
          reg.h.ch = bios->cursTop | status ;
     else
          reg.h.ch = bios->cursTop & status ;

     reg.h.ah = 1 ;
     reg.h.cl = bios->cursBottom ;
     int86(0x10,&reg,&reg) ;
}

int power(int x,int y)
{
 int i;
 if(y==0)
  x=1;
 else
   for(i=0;i<y-1;i++)
       x+=x;
 return(x);
}
```

MOTOROLA
SEMICONDUCTOR
TECHNICAL DATA

MC145026
MC145027
MC145028
SC41342
SC41343
SC41344

Encoder and Decoder Pairs
CMOS

These devices are designed to be used as encoder/decoder pairs in remote control applications.

The MC145026 encodes nine lines of information and serially sends this information upon receipt of a transmit enable ($\overline{TE}$) signal. The nine lines may be encoded with trinary data (low, high, or open) or binary data (low or high). The words are transmitted twice per encoding sequence to increase security.

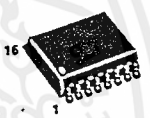
The MC145027 decoder receives the serial stream and interpets five of the trinary digits as an address code. Thus, 243 addresses are possible. If binary data is used at the encoder, 32 addresses are possible. The remaining serial information is interpeted as four bits of binary data. The valid transmission output (VT) goes high on the MC145027 when two conditions are met. First, two addresses must be consecutively received (in one encoding sequence) which both match the local address. Second, the 4-bits of data must match the last valid data received. The active VT indicates that the information at the data output pins has been updated.

The MC145028 decoder treats all nine trinary digits as an address which allows 19,683 codes. If binary data is encoded, 512 codes are possible. The valid transmission output (VT) goes high on the MC145028 when two addresses are consecutively received (in one encoding sequence) which both match the local address.

- Operating Temperature Range: –40° to 85°C
- Very-Low Standby Current for the Encoder: 300 nA Maximum @ 25°C
- Interfaces with RF, Ultrasonic, or Infrared Modulators and Demodulators
- RC Oscillator, No Crystal Required
- High External Component Tolerance; Can Use ±5% Components
- Internal Power-On Reset Forces All Decoder Outputs Low
- For Infrared Applications, See Applications Note AN1016
- Operating Voltage Range: 4.5 to 18 V
- Low-Voltage Versions Available —
  - SC41342: 2.5 to 18 V Version of the MC145026
  - SC41343: 2.8 to 10 V Version of the MC145027
  - SC41344: 2.8 to 10 V Version of the MC145028

P SUFFIX
PLASTIC DIP
CASE 648

D SUFFIX
SOG
CASE 751B

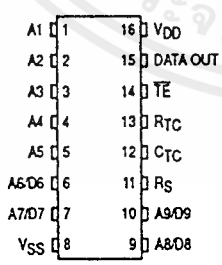


DW SUFFIX
SOG
CASE 751G

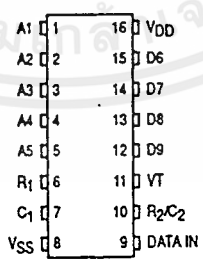
ORDERING INFORMATION

MC145026P, SC41342P	Plastic DIP
MC145026D, SC41342D	SOG Package
MC145027P, SC41343P	Plastic DIP
MC145027DW, SC41343DW	SOG Package
MC145028P, SC41344P	Plastic DIP
MC145028DW, SC41344DW	SOG Package

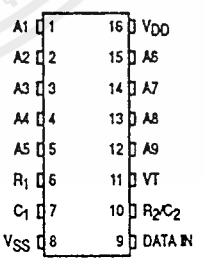
PIN ASSIGNMENTS



MC145026
SC41342
ENCODERS



MC145027
SC41343
DECODERS



MC145028
SC41344
DECODERS

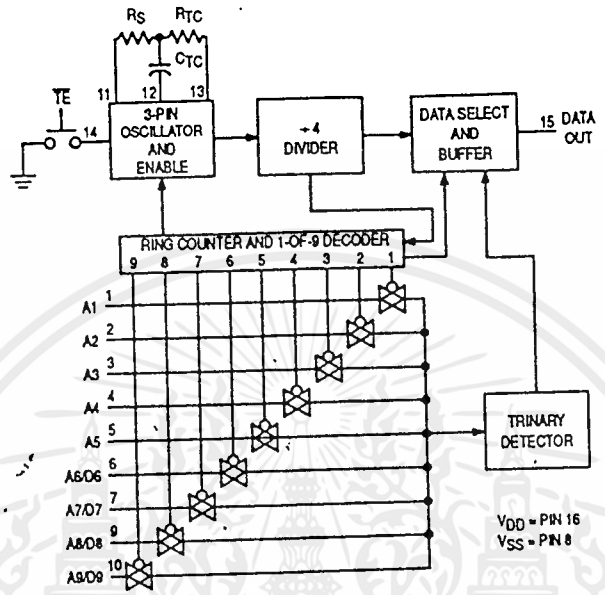


Figure 1. MC145026 Encoder Block Diagram

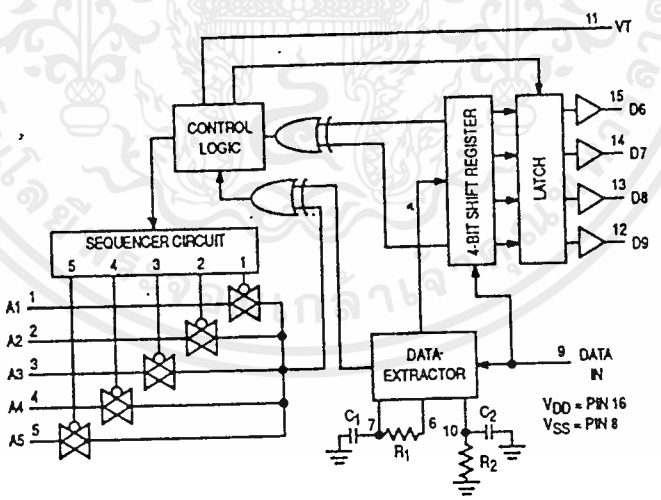


Figure 2. MC145027 Decoder Block Diagram

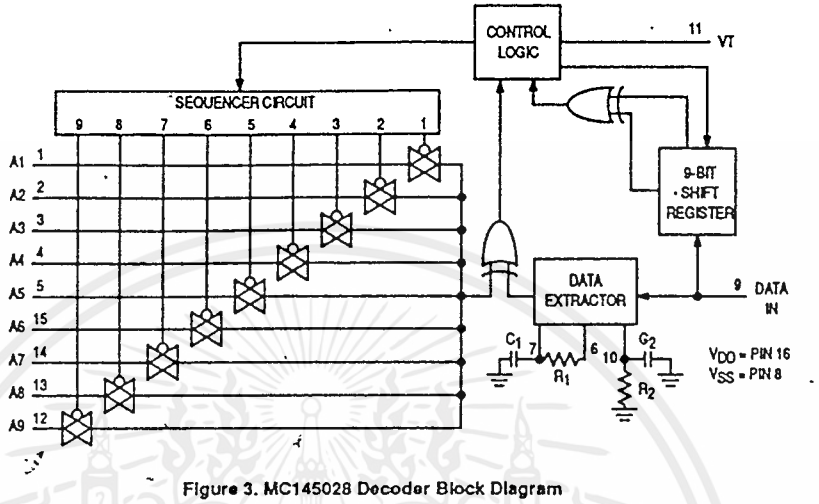


Figure 3. MC145028 Decoder Block Diagram

MAXIMUM RATINGS* (Voltage Referenced to V_{SS})

Symbol	Parameter	Value	Unit
V_{DD}	DC Supply Voltage (except SC41343, SC41344)	−0.5 to +18	V
V_{DD}	DC Supply Voltage (SC41343, SC41344 only)	−0.5 to +10	V
V_{in}	DC Input Voltage	−0.5 to V_{DD}+0.5	V
V_{out}	DC Output Voltage	−0.5 to V_{DD}+0.5	V
I_{in}	DC Input Current, per Pin	±10	mA
I_{out}	DC Output Current, per Pin	±10	mA
P_D	Power Dissipation, per Package	500	mW
T_{stg}	Storage Temperature	−65 to +150	°C
T_L	Lead Temperature, 1 mm from Case for 10 Seconds	260	°C

* Maximum Ratings are those values beyond which damage to the device may occur. Functional operation should be restricted to the limits in the Electrical Characteristics tables or Pin Descriptions section.

This device contains protection circuitry to guard against damage due to high static voltages or electric fields. However, precautions must be taken to avoid applications of any voltage higher than maximum rated voltages to this high-impedance circuit. For proper operation, V_{in} and V_{out} should be constrained to the range $V_{SS} \le (V_{in}$ or $V_{out}) \le V_{DD}$.

ELECTRICAL CHARACTERISTICS — MC145026, MC145027, MC145028, and SC41342* (Voltage Referenced to V_{SS})

Symbol	Characteristic	V_{DD} V	Guaranteed Limit						Unit
			−40°C		25°C		+85°C		
			Min	Max	Min	Max	Min	Max	
V_{OL}	Low-Level Output Voltage (V_{in} = V_{DD} or 0)	5.0	—	0.05	—	0.05	—	0.05	V
		10	—	0.05	—	0.05	—	0.05	
		15	—	0.05	—	0.05	—	0.05	
V_{OH}	High-Level Output Voltage (V_{in} = 0 or V_{DD})	5.0	4.95	—	4.95	—	4.95	—	V
		10	9.95	—	9.95	—	9.95	—	
		15	14.95	—	14.95	—	14.95	—	
V_{IL}	Low-Level Input Voltage								V
	(V_{out} = 4.5 or 0.5 V)	5.0	—	1.5	—	1.5	—	1.5	
	(V_{out} = 9.0 or 1.0 V)	10	—	3.0	—	3.0	—	3.0	
	(V_{out} = 13.5 or 1.5 V)	15	—	4.0	—	4.0	—	4.0	
V_{IH}	High-Level Input Voltage								V
	(V_{out} = 0.5 or 4.5 V)	5.0	3.5	—	3.5	—	3.5	—	
	(V_{out} = 1.0 or 9.0 V)	10	7.0	—	7.0	—	7.0	—	
	(V_{out} = 1.5 or 13.5 V)	15	11	—	11	—	11	—	
I_{OH}	High-Level Output Current								mA
	(V_{out} = 2.5 V)	5.0	−2.5	—	−2.1	—	−1.7	—	
	(V_{out} = 4.6 V)	5.0	−0.52	—	−0.44	—	−0.36	—	
	(V_{out} = 9.5 V)	10	−1.3	—	−1.1	—	−0.9	—	
	(V_{out} = 13.5 V)	15	−3.6	—	−3.0	—	−2.4	—	
I_{OL}	Low-Level Output Current								mA
	(V_{out} = 0.4 V)	5.0	0.52	—	0.44	—	0.36	—	
	(V_{out} = 0.5 V)	10	1.3	—	1.1	—	0.9	—	
	(V_{out} = 1.5 V)	15	3.6	—	3.0	—	2.4	—	
I_{in}	Input Current — $\overline{TE}$	5.0	—	—	3.0	11	—	—	µA
	(MC145026 and SC41342, Pullup Device)	10	—	—	16	60	—	—	
		15	—	—	35	120	—	—	
I_{in}	Input Current R_S (MC145026 and SC41342), Data In (MC145027, MC145028)	15	—	±0.3	—	±0.3	—	±1.0	µA
I_{in}	Input Current								µA
	A1-A5, A6/D6–A9/D9 (MC145026 and SC41342),	5.0	—	—	—	±110	—	—	
	A1-A5 (MC145027),	10	—	—	—	±500	—	—	
	A1-A9 (MC145028)	15	—	—	—	+1000	—	—	
C_{in}	Input Capacitance (V_{in} = 0)	—	—	—	—	7.5	—	—	pF
I_{DD}	Quiescent Current — MC145026 and SC41342	5.0	—	—	—	0.1	—	—	µA
		10	—	—	—	0.2	—	—	
		15	—	—	—	0.3	—	—	
I_{DD}	Quiescent Current — MC145027, MC145028	5.0	—	—	—	50	—	—	µA
		10	—	—	—	100	—	—	
		15	—	—	—	150	—	—	
I_{dd}	Dynamic Supply Current — MC145026 and SC41342 (f_C = 20 kHz)	5.0	—	—	—	200	—	—	µA
		10	—	—	—	400	—	—	
		15	—	—	—	600	—	—	
I_{dd}	Dynamic Supply Current — MC145027, MC145028 (f_C = 20 kHz)	5.0	—	—	—	400	—	—	µA
		10	—	—	—	800	—	—	
		15	—	—	—	1200	—	—	

*Also see next Electrical Characteristics table for 2.5 V specifications.

ELECTRICAL CHARACTERISTICS — SC41342 (Voltage Referenced to V_{SS})

Symbol	Characteristic	V_{DD} V	Guaranteed Limit						Unit
			−40°C		25°C		+85°C		
			Min	Max	Min	Max	Min	Max	
V_{OL}	Low-Level Output Voltage (V_{in} = 0 V or V_{DD})	2.5	—	0.05	—	0.05	—	0.05	V
V_{OH}	High-Level Output Voltage (V_{in} = 0 V or V_{DD})	2.5	2.45	—	2.45	—	2.45	—	V
V_{IL}	Low-Level Input Voltage (V_{out} = 0.5 V or 2.0 V)	2.5	—	0.3	—	0.3	—	0.3	V
V_{IH}	High-Level Input Voltage (V_{out} = 0.5 V or 2.0 V)	2.5	2.2	—	2.2	—	2.2	—	V
I_{OH}	High-Level Output Current (V_{out} = 1.25 V)	2.5	0.28	—	0.25	—	0.2	—	mA
I_{OL}	Low-Level Output Current (V_{out} = 0.4 V)	2.5	0.22	—	0.2	—	0.16	—	mA
I_{in}	Input Current ($\overline{TE}$ — Pullup Device)	2.5	—	—	0.09	1.8	—	—	µA
I_{in}	Input Current (A1-A5, A6/D6-A9/D9)	2.5	—	—	—	±25	—	—	µA
I_{DD}	Quiescent Current	2.5	—	—	—	0.05	—	—	µA
I_{dd}	Dynamic Supply Current (f_c = 20 kHz)	2.5	—	—	—	40	—	—	µA

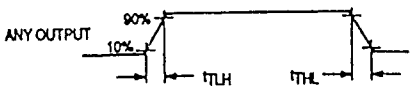


Figure 4.

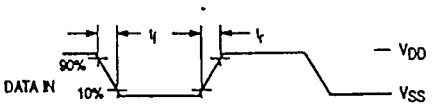


Figure 5.

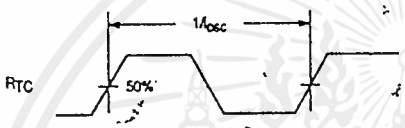
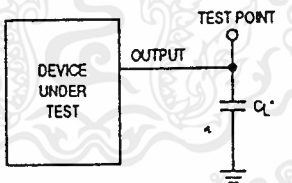


Figure 6.

Figure 7.



*INCLUDES ALL PROBE AND JIG CAPACITANCE.

Figure 8. Test Circuit

OPERATING CHARACTERISTICS

MC145026

The encoder serially transmits trinary data as defined by the state of the A1 through A5 and A6/D6 through A9/D9 input pins. These pins may be in either of three states (low, high, or open) allowing 19,683 possible codes. The transmit sequence is initiated by a low level on the $\overline{TE}$ input pin. Each time the $\overline{TE}$ input is forced low, the encoder outputs two identical data words. Between the two data words, no signal is sent for three data periods. If the $\overline{TE}$ input is kept low, the encoder continuously transmits the data word. See Figure 10.

Each transmitted trinary digit is encoded into pulses (See Figure 11). A logic zero (low) is encoded as two consecutive short pulses, a logic one (high) as two consecutive long pulses, and an open (high-impedance) as a long pulse followed by a short pulse. The input state is determined by using a weak "output" device to try to force each input first low, then high. If only a high state results from the two tests, the input is assumed to be hardwired to V_{DD}. If only a low state is obtained, the input is assumed to be hardwired to V_{SS}. If both a high and a low can be forced at an input, an open is assumed and is encoded as such. The "high" and "low" levels are 70% and 30% of the supply voltage as shown in the Electrical Characteristics Table. The weak "output" device sinks/sources up to 110 µA at a 5 V supply level, 500 µA at 10 V, and 1 mA at 15 V.

The $\overline{TE}$ input has an internal pullup device so that a simple switch may be used to force the input low. While $\overline{TE}$ is high, the encoder is completely disabled, the oscillator is inhibited, and the current drain is reduced to quiescent current. When $\overline{TE}$ is brought low, the oscillator is started, and the transmit sequence begins. The inputs are then sequentially selected, and determinations are made as to the input logic states. This information is serially transmitted via the Data Out pin.

MC145027

This decoder receives the serial data from the encoder and outputs the data, if it is valid. The transmitted data, consisting of two identical words, is examined bit by bit during reception. The first five trinary digits are assumed to be the address. If the received address matches the local address, next four (data) bits are internally stored, but are not transferred to the output data latch. As the second encoded word is received, the address must again match. If a match occurs, the new data bits are checked against the previously stored data bits. If the two nibbles of data (four bits each) match, the data is transferred to the output data latch by VT and remains until new data replaces it. At the same time, the VT output pin is brought high and remains high until an error is received or until no input signal is received for four data periods. See Figure 10.

Although the address information may be encoded in trinary, the data information must be either a one or a zero. A trinary (open) data line is decoded as a logic one.

MC145028

This decoder operates in the same manner as the MC145027 except that nine address lines are used and no data output is available. The VT output is used to indicate that a valid address has been received. For transmission security, two identical transmitted words must be consecutively received before a valid transmission output (VT) signal is issued.

The MC145028 allows 19,683 addresses when trinary levels are used. 512 addresses are possible when binary levels are used.

PIN DESCRIPTIONS

MC145026 ENCODER

A1 through A5, A6/D6 through A9/D9 (Pins 1 through 7, 9, and 10)

These address/data inputs are encoded and the data is sent serially from the encoder via the data out pin.

R_S, C_{TC}, R_{TC} (Pins 11, 12, and 13)

These pins are part of the oscillator section of the encoder. See Figure 9.

If an external signal source is used instead of the internal oscillator, it should be connected to the R_S input and the R_{TC} and C_{TC} pins should be left open.

$\overline{TE}$ (Pin 14)

This active-low transmit enable input initiates transmission when forced low. An internal pullup device keeps this input normally high. The pullup current is specified in the Electrical Characteristics table.

Data Out (Pin 15)

This is the output of the encoder that serially presents the encoded data word.

V_{SS} (Pin 8)

The most-negative supply potential. This pin is usually ground.

V_{DD} (Pin 16)

The most-positive power supply pin.

MC145027 AND MC145028 DECODERS

A1 through A5 (Pins 1 through 5) — MC145027
A1 through A9 (Pins 1 through 5, 15, 14, 13, and 12) — MC145028

These are the local address inputs. The states of these pins must match the appropriate encoder inputs for the VT pin to go high. The local address may be encoded with trinary or binary data.

D6 through D9 (Pins 15, 14, 13, and 12) – MC145027 ONLY

These outputs present the binary information that is on encoder inputs A6/D6 through A9/D9. Only binary data is acknowledged; a trinary open at the MC145026 encoder is decoded as a high level (logic 1).

R_1, C_1 (Pins 6, 7)

As shown in Figures 2 and 3, these pins accept a resistor and capacitor that are used to determine whether a narrow pulse or wide pulse has been received. The time constant $R_1 \times C_1$ should be set to 1/72 encoder clock periods:

$R_1 C_1 = 3.95 R_{TC} C_{TC}$.

R_2/C_2 (Pin 10)

As shown in Figures 2 and 3, this pin accepts a resistor and capacitor that are used to detect both the end of a received word and the end of a transmission. The time contant $R_2 \times C_2$ should be 33.5 encoder clock periods (four data periods per Figure 11): $R_2 C_2 = 77 R_{TC} C_{TC}$. This time contant is used to determine whether the data in pin has remained low for four data periods (end of transmission). A separate on-chip comparator looks at the voltage-equivalent two data periods ($0.4 R_2 C_2$) to detect the dead time between received words within a transmission.

VT (Pin 11)

This valid transmission output goes high after the second word of an encoding sequence when the following conditions are satisfied:

(1) the received addresses of both words match the local decoder address, and

(2) the received data bits of both words match.

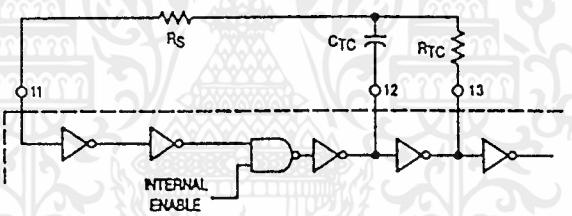
VT remains high until either a mismatch is received or no input signal is received for four data periods.

V_{SS} (Pin 8)

The most-negative supply potential. This pin is usually ground.

V_{DD} (Pin 16)

The most-positive power supply pin.



This oscillator operates at a frequency determined by the external RC network; i.e.,

$$f = \frac{1}{2.3\, R_{TC}\, C_{TC}'} \text{ (Hz)}$$

for $1\text{ kHz} \le f \le 400\text{ kHz}$

where: $C_{TC}' = C_{TC} + C_{layout} + 12\text{ pF}$

$R_S = 2 R_{TC}$

$R_S \ge 20\text{ k}$

$R_{TC} \ge 10\text{ k}$

$400\text{ pF} < C_{TC} < 15\ \mu\text{F}$

The value for R_S should be chosen to be ≥ 2 times R_{TC}. This range ensures that current through R_S is insignificant compared to current through R_{TC}. The upper limit for R_S must ensure that $R_S \times 5$ pF (input capacitance) is small compared to $R_{TC} \times C_{TC}$.

For frequencies outside the indicated range, the formula is less accurate. The minimum recommended oscillation frequency of this circuit is 1 kHz. Susceptibility to externally induced noise signals may occur for frequencies below 1 kHz and/or when resistors utilized are greater than 1 MΩ.

Figure 9. Encoder Oscillator Information

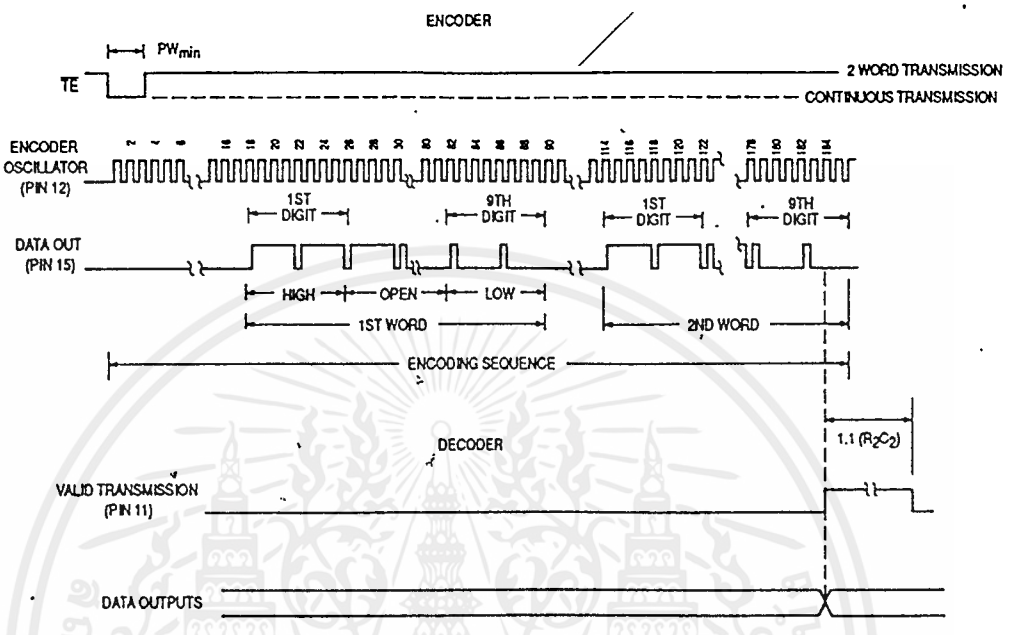


Figure 10. Timing Diagram

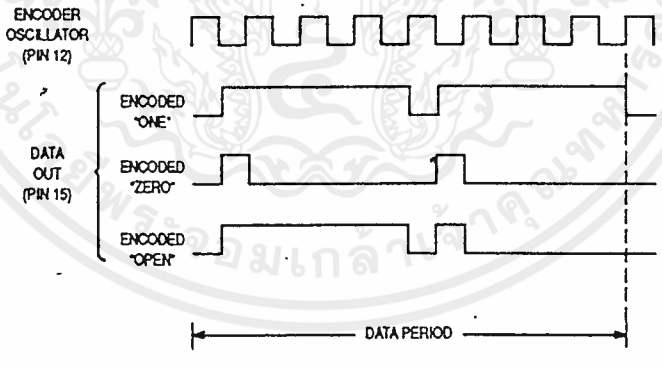


Figure 11. Encoder Data Waveforms

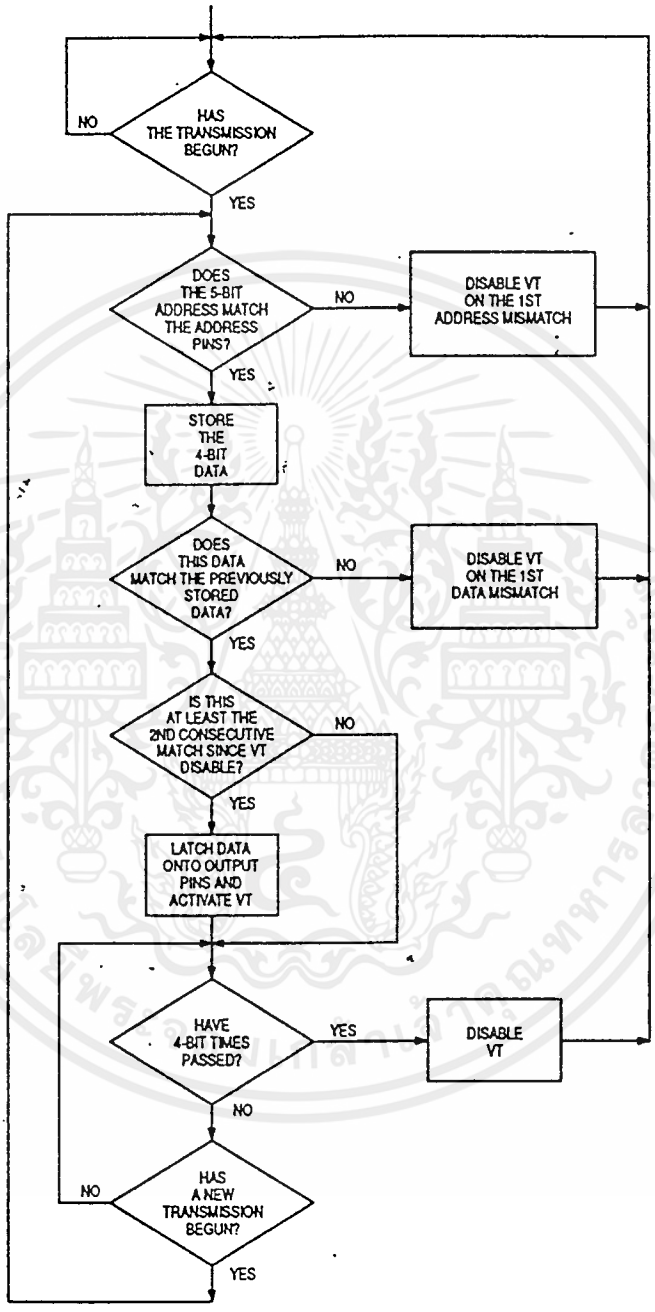


Figure 12. MC145027 Flowchart

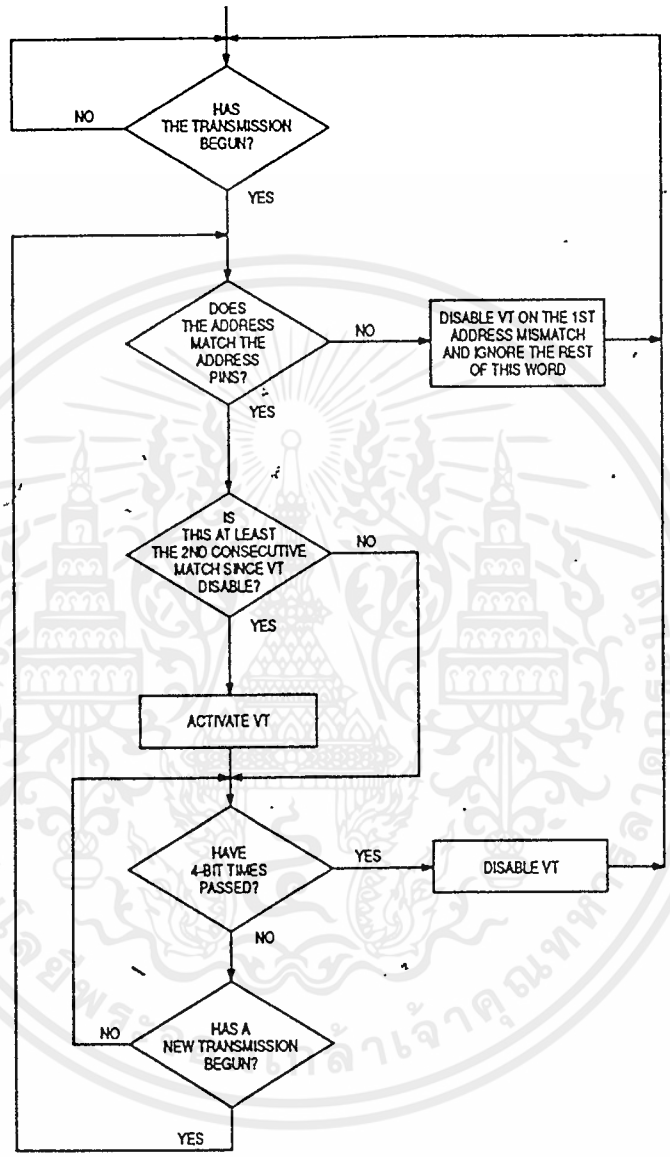


Figure 13. MC145028 Flowchart

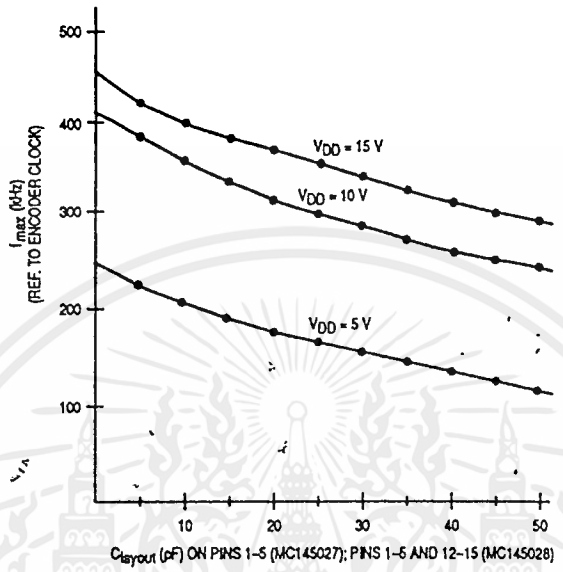
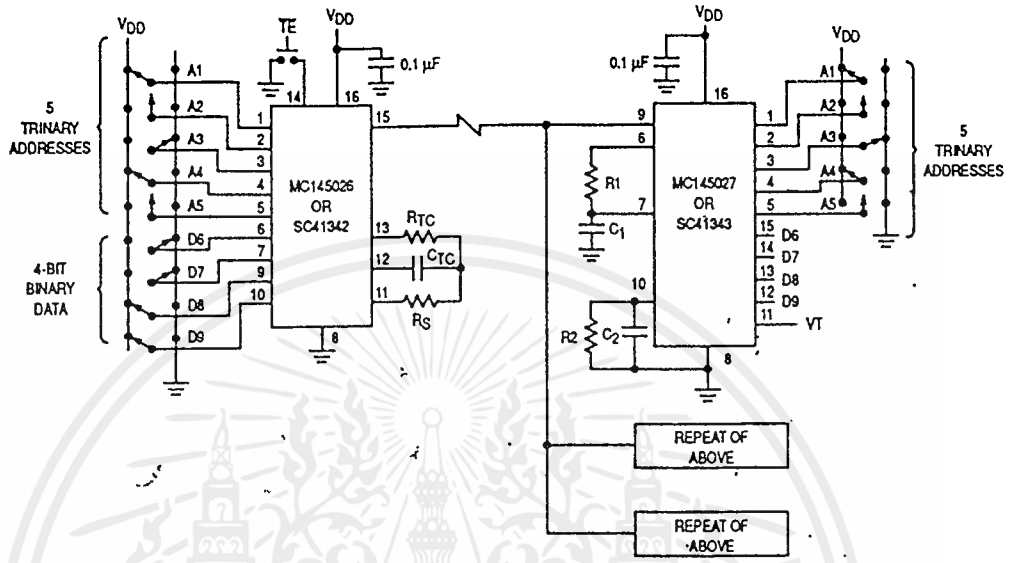


Figure 14. f_{max} vs C_{layout} — Decoders Only



$$f_{osc} = \frac{1}{2.3\,R_{TC}C_{TC}'}$$

$R_1C_1 = 3.95\,R_{TC}C_{TC}$

$R_2C_2 = 77\,R_{TC}C_{TC}$

$C_{TC}' = C_{TC} + C_{layout} + 12\ pF$
$100\ pF \le C_{TC} \le 15\ \mu F$
$R_{TC} \ge 10\ k;\ R_S = 2\,R_{TC}$
$R_1 \ge 10\ k$
$C_1 \ge 400\ pF$
$R_2 \ge 100\ k$
$C_2 \ge 700\ pF$

Example R/C Values

(All Resistors and Capacitors are ±5%)

($C_{TC}' = C_{TC} + 20\ pF$)

f_{osc} (kHz)	R_{TC}	C_{TC}'	R_S	R_1	C_1	R_2	C_2
362	10 k	120 pF	20 k	10 k	470 pF	100 k	910 pF
181	10 k	240 pF	20 k	10 k	910 pF	100 k	1800 pF
88.7	10 k	490 pF	20 k	10 k	2000 pF	100 k	3900 pF
42.6	10 k	1020 pF	20 k	10 k	3900 pF	100 k	7500 pF
21.5	10 k	2020 pF	20 k	10 k	8200 pF	100 k	0.015 µF
8.53	10 k	5100 pF	20 k	10 k	0.02 µF	200 k	0.02 µF
1.71	50 k	5100 pF	100 k	50 k	0.02 µF	200 k	0.1 µF

Figure 15. Typical Application

APPLICATIONS INFORMATION

Infrared Transmitter

In Figure 16, the MC145026 encoder is set to run at an oscillator frequency of about 4 kHz to 9 kHz. Thus, the time required for a complete two-word encoding sequence is about 20 ms to 40 ms. The data output from the encoder gates an RC oscillator running at 50 kHz; the oscillator shown starts rapidly enough to be used in this application. When the "send" button is not depressed, both the MC145026 and oscillator are in a low-power standby state. The RC oscillator has to be trimmed for 50 kHz and has some drawbacks for frequency stability. A superior system uses a ceramic resonator oscillator running at 400 kHz. This oscillator feeds a divider as shown in Figure 17. The unused inputs of the MC14011UB must be grounded.

The MLED81 IRED is driven with the 50 kHz square wave at about 200 mA to 300 mA to generate the carrier. If desired, 2 IREDs wired in series can be used. (See Application Note AN1016 for more information.) The bipolar IRED switch shown in Figure 16 offers two advantages over a FET. First, a logic FET has too much gate capacitance for the MC14011UB to drive without waveform distortion. Second, the bipolar drive permits lower supply voltages, which are an advantage in portable battery-powered applications.

The configuration shown in Figure 16 operates over a supply range of 4.5 V to 18 V. A low-voltage system which operates down to 2.5 V could be realized if the SC41342 (the low-voltage version of the MC145026) is used in lieu of the MC145026. The oscillator section of a MC74HC4060 is used in place of the MC14011UB. The data output of the SC41342 is inverted and fed to the reset pin of the MC74HC4060. Alternately, the MC74HCU04 could be used for the oscillator.

Information on the MC14011UB is in book number DL131/D. The MC74HCU04 and MC74HC4060 are found in book number DL129/D.

Infrared Receiver

The receiver in Figure 18 couples an IR-sensitive diode to input preamp A1, followed by bandpass amplifier A2 with a gain of about 10. Limiting stage A3 follows, with an output of about 800 mVp-p. The limited 50 kHz burst is detected by comparator A4 that passes only positive pulses, and peak-detected and filtered by a diode/RC network to extract the data envelope from the burst. Comparator A5 boosts the signal to logic levels compatible with the MC145027/8 data input. The data in pin of these decoders is a standard CMOS high-impedance input which must NOT be allowed to float. Therefore, direct coupling from A5 to the decoder input is utilized.

Shielding should be used on at least A1 and A2, with good ground and high-sensitivity circuit layout techniques applied.

For operation with supplies higher than +5 V, limiter A4's positive output swing needs to be limited to 3 V to 5 V. This is accomplished via adding a zener diode in the negative feedback path, thus avoiding excessive system noise. The biasing resistor stack should be adjusted such that V3 is 1.25 V to 1.5 V.

This system works up to a range of about 10 meters. The gains of the system may be adjusted to suit the individual design needs. The 100 Ω resistor in the emitter of the first 2N5088 and the 1 kΩ resistor feeding A2 may be altered if different gain is required. In general, more gain does not necessarily result in increased range. This is due to noise floor limitations. The designer should increase transmitter power and/or increase receiver aperature with fresnal lensing to greatly improve range. See applications note AN1016 for additional information.

Information on the MC34074 is in data book DL128/D.

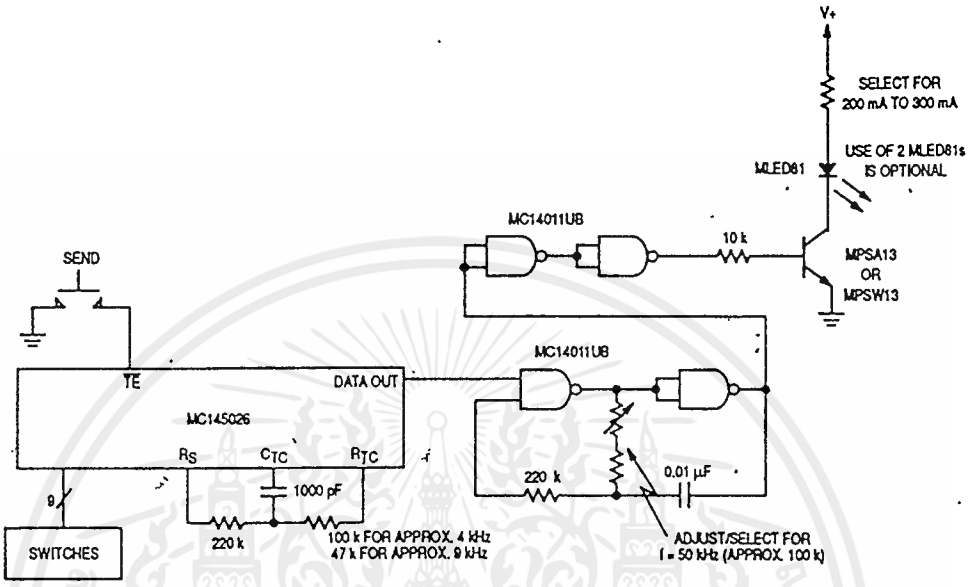


Figure 16. IRED Transmitter Using RC Oscillator to Generate Carrier Frequency

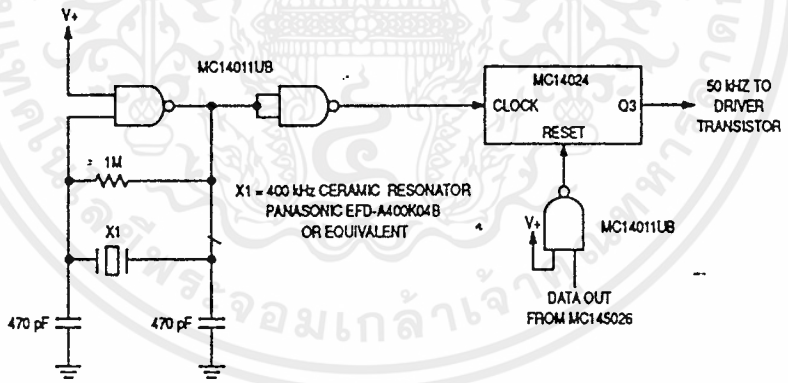


Figure 17. Using a Ceramic Resonator to Generate Carrier Frequency

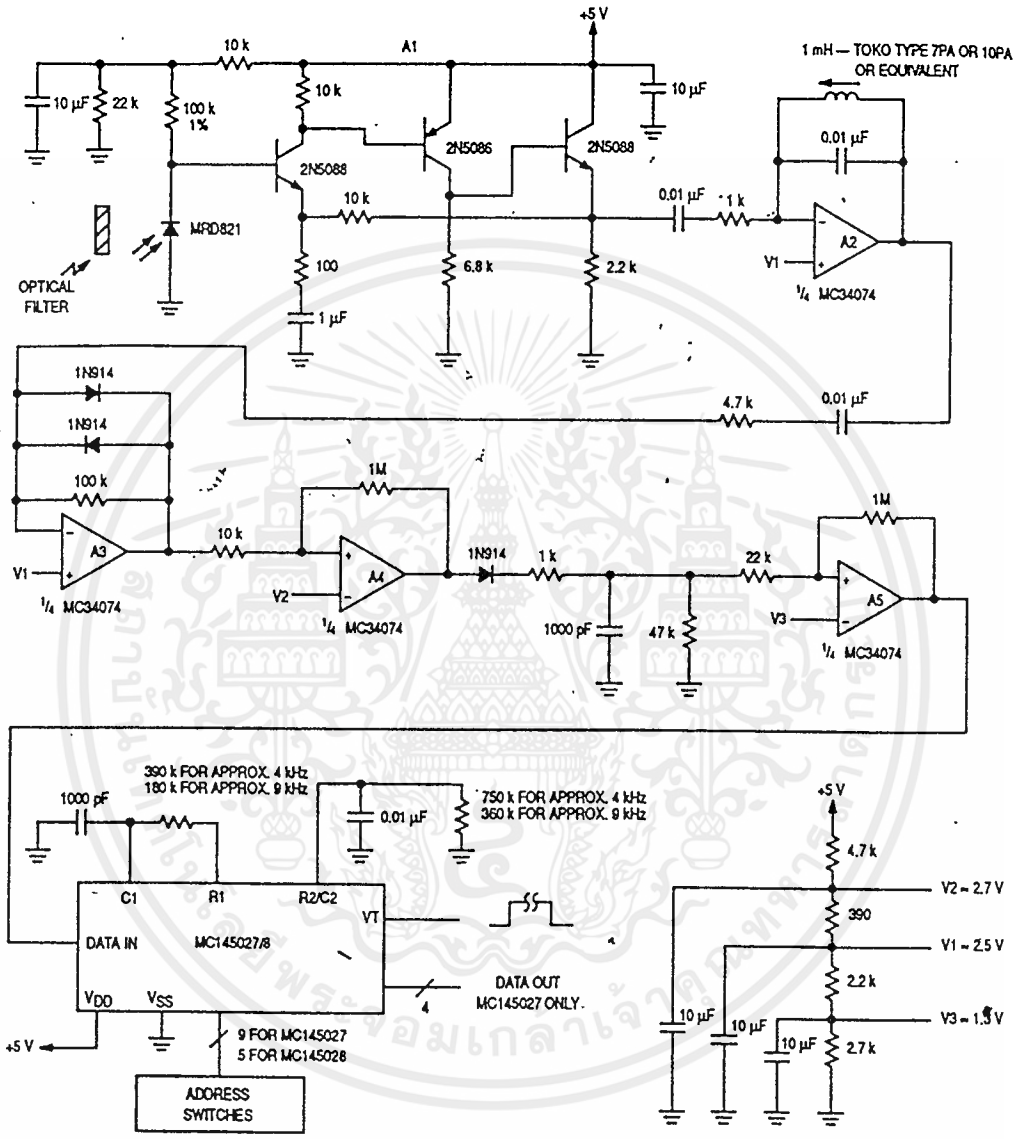


Figure 18. Infrared Receiver

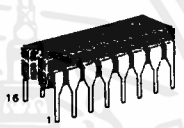
MC3486

QUAD RS-422/423 LINE RECEIVER

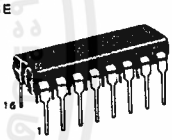
Motorola's Quad RS-422/3 Receiver features four independent receiver chains which comply with EIA Standards for the Electrical Characteristics of Balanced/Unbalanced Voltage Digital Interface Circuits. Receiver outputs are 74LS compatible, three-state structures which are forced to a high impedance state when the appropriate output control pin reaches a logic zero condition. A PNP device buffers each output control pin to assure minimum loading for either logic one or logic zero inputs. In addition, each receiver chain has internal hysteresis circuitry to improve noise margin and discourage output instability for slowly changing input waveforms. A summary of MC3486 features include:

- Four Independent Receiver Chains
- Three-State Outputs
- High Impedance Output Control Inputs (PIA Compatible)
- Internal Hysteresis – 100 mV (Typ)
- Fast Propagation Times – 25 ns (Typ)
- TTL Compatible
- Single 5 V Supply Voltage

QUAD RS-422/3 LINE RECEIVER WITH THREE-STATE OUTPUTS

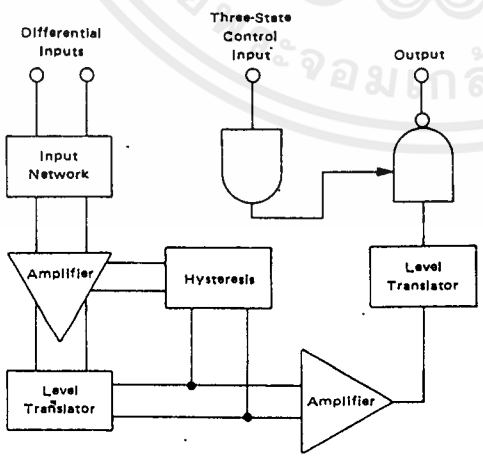


L SUFFIX
CERAMIC PACKAGE
CASE 620

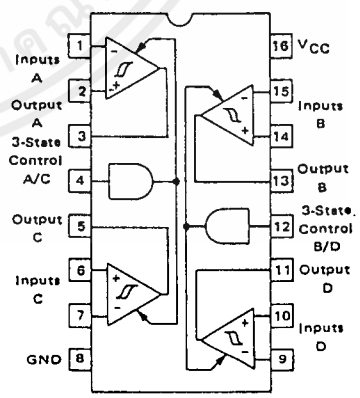


P SUFFIX
PLASTIC PACKAGE
CASE 648

RECEIVER CHAIN BLOCK DIAGRAM



PIN CONNECTIONS



ORDERING INFORMATION		
DEVICE	TEMPERATURE RANGE	PACKAGE
MC3486L	0 to +70°C	Ceramic DIP
MC3486P	0 to +70°C	Plastic DIP

ABSOLUTE MAXIMUM RATINGS (Note 1)

Rating	Symbol	Value	Unit
Power Supply Voltage	V_{CC}	8.0	Vdc
Input Common Mode Voltage	V_{ICM}	±15	Vdc
Input Differential Voltage	V_{ID}	±15	Vdc
Three-State Control Input Voltage	V_I	8.0	Vdc
Output Sink Current	I_O	50	mA
Storage Temperature	T_{stg}	-65 to +150	°C
Operating Junction Temperature	T_J		°C
Ceramic Package		+175	
Plastic Package		+150	

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 devices should be operated at these limits. The "Table of Electrical Characteristics" provides conditions for actual device operation.

RECOMMENDED OPERATING CONDITIONS

Rating	Symbol	Value	Unit
Power Supply Voltage	V_{CC}	4.75 to 5.25	Vdc
Operating Ambient Temperature	T_A	0 to +70	°C
Input Common Mode Voltage Range	V_{ICR}	-7.0 to +7.0	Vdc
Input Differential Voltage Range	V_{IDR}	6.0	Vdc

ELECTRICAL CHARACTERISTICS

(Unless otherwise noted minimum and maximum limits apply over recommended temperature and power supply voltage ranges. Typical values are for T_A = 25°C, V_{CC} = 5.0 V and V_{IC} = 0 V. See Note 1.)

Characteristic	Symbol	Min	Typ	Max	Unit
Input Voltage — High Logic State (Three-State Control)	V_{IH}	2.0	–	–	V
Input Voltage — Low Logic State (Three-State Control)	V_{IL}	–	–	0.8	V
Differential Input Threshold Voltage ($-7.0\ V \le V_{IC} \le 7.0\ V$, $V_{IH(3C)}$ = 2.0 V)	$V_{TH(D)}$				V
(I_O = 0.4 mA, $V_{OH} > 2.7$ V)		–	0.05	0.2	
(I_O = 8.0 mA, $V_{OL} > 0.5$ V)		–	-0.05	-0.2	
Input Bias Current (V_{CC} = 0 V or 5.25) (Other Inputs at 0 V)	$I_{IB(D)}$				mA
(V_I = −10 V)		–	–	-3.25	
(V_I = −3.0 V)		–	–	-1.50	
(V_I = +3.0 V)		–	–	+1.50	
(V_I = +10 V)		–	–	+3.25	
Input Balance ($-7.0\ V \le V_{IC} \le 7.0\ V$, $V_{IH(3C)}$ = 2.0 V, See Note 3)					V
(I_O = 0.4 mA, V_{ID} = 0.4 V)		2.7	–	–	
(I_O = 8.0 mA, V_{ID} = 0.4 V)		–	–	0.5	
Output Third State Leakage Current	I_{OZ}				μA
($V_{I(D)}$ = +3.0 V, $V_{IL(3C)}$ = 0.8 V, V_{OL} = 0.5 V)		–	–	-40	
($V_{I(D)}$ = −3.0 V, $V_{IL(3C)}$ = 0.8 V, V_{OH} = 2.7 V)		–	–	40	
Output Short-Circuit Current ($V_{I(D)}$ = 3.0 V, $V_{IH(3C)}$ = 2.0 V, V_O = 0 V, See Note 2)	I_{OS}	-15	–	-100	mA
Input Current — Low Logic State (Three-State Control) ($V_{IH(3S)}$ = 0.5 V)	I_{IL}	–	–	-100	μA
Input Current — High Logic State (Three-State Control)	I_{IH}				μA
($V_{IH(3S)}$ = 2.7 V)		–	–	20	
($V_{IH(3S)}$ = 5.25 V)		–	–	100	
Input Clamp Diode Voltage (Three-State Control) ($I_{IC(3S)}$ = -10 mA)	V_{IC}	–	–	-1.5	V
Power Supply Current ($V_{IL(3S)}$ = 0 V)	I_{CC}	–	–	80	mA

ELECTRICAL CHARACTERISTICS (continued)

SWITCHING CHARACTERISTICS (Unless otherwise noted, V_{CC} = 5.0 V and T_A = 25°C.)

Characteristic	Symbol	Min	Typ	Max	Unit
Propagation Delay Time — Differential Inputs to Output					ns
(Output High to Low)	$t_{PHL(D)}$	–	20	–	
(Output Low to High)	$t_{PLH(D)}$	–	25	–	
Propagation Delay time — Three-State Control to Output					ns
(Output Low to Third State)	t_{PLZ}	–	23	–	
(Output High to Third State)	t_{PHZ}	–	25	–	
(Output Third State to High)	t_{PZH}	–	18	–	
(Output Third State to Low)	t_{PZL}	–	20	–	

NOTES:
1. All currents into device pins are shown as positive, out of device pins are negative. All voltages referenced to ground unless otherwise noted.
2. Only one output at a time should be shorted.
3. Refer to EIA RS-422/3 for exact conditions.

FIGURE 1 — SWITCHING TEST CIRCUIT AND WAVEFORMS

Propagation Delay Differential Input to Output

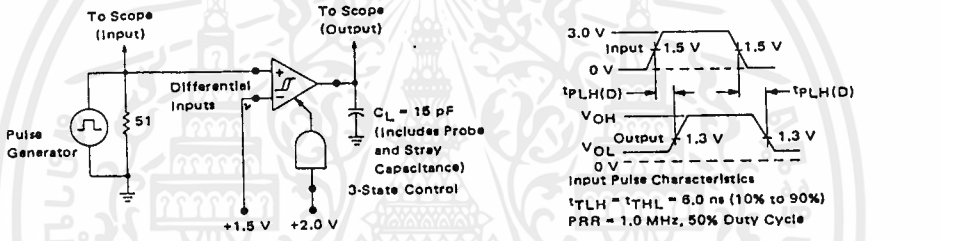
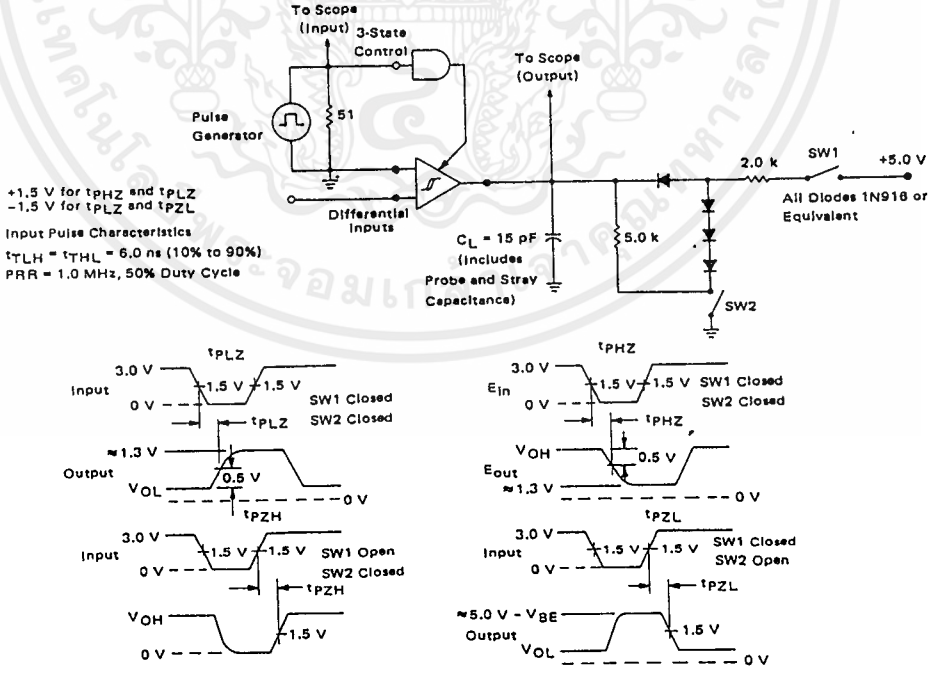


FIGURE 2 — PROPAGATION DELAY THREE-STATE CONTROL INPUT TO OUTPUT



XC3487

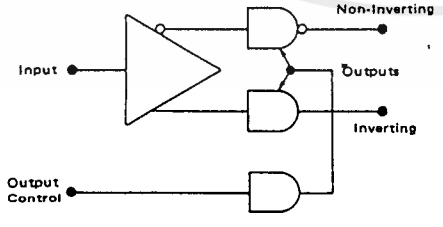
Product Preview

QUAD LINE DRIVER WITH THREE-STATE OUTPUTS

Motorola's Quad RS-422 Driver features four independent driver chains which comply with EIA Standards for the Electrical Characteristics of Balanced Voltage Digital Interface Circuits. The outputs are three-state structures which are forced to a high impedance state when the appropriate output control pin reaches a logic zero condition. All input pins are PNP buffered to minimize input loading for either logic one or logic zero inputs. In addition, internal circuitry assures a high impedance output state during the transition between power up and power down. A summary of MC3487 features include:

- Four Independent Driver Chains
- Three-State Outputs
- PNP High Impedance Inputs (PIA Compatible)
- Power Up/Down Protection
- Fast Propagation Times (Typ 15 ns)
- TTL Compatible
- Single 5 V Supply Voltage
- Output Rise and Fall Times Less Than 20 ns
- Equivalent to DS 3487

DRIVER BLOCK DIAGRAM

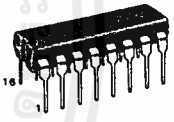


This is advance information and specifications are subject to change without notice.

QUAD RS-422 LINE DRIVER WITH THREE-STATE OUTPUTS

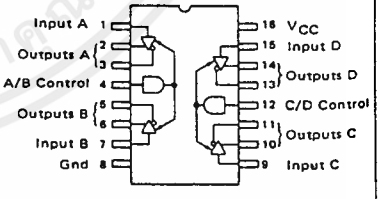
SILICON MONOLITHIC INTEGRATED CIRCUIT

L SUFFIX
CERAMIC PACKAGE
CASE 620



P SUFFIX
PLASTIC PACKAGE
CASE 648

PIN CONNECTIONS



TRUTH TABLE

Input	Control Input	Non-Inverter Output	Inverter Output
H	H	H	L
L	H	L	H
X	L	Z	Z

L = Low Logic State
H = High Logic State
X = Irrelevant
Z = Third-State (High Impedance)

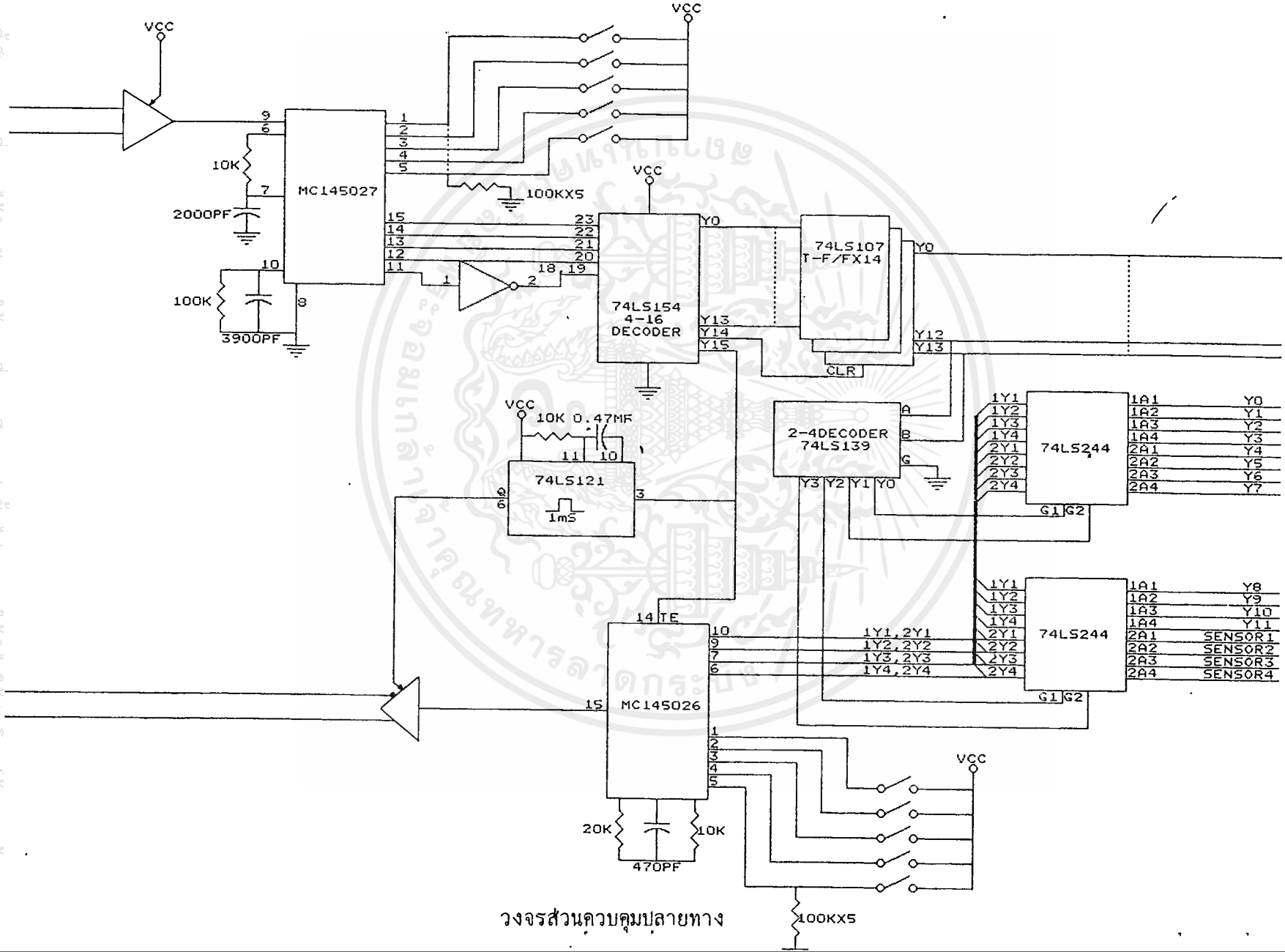
*ABSOLUTE MAXIMUM RATINGS			
Rating	**Symbol**	**Value**	**Unit**
Power Supply Voltage	V_{CC}	8.0	Vdc
Input Voltage	V_I	5.5	Vdc
Operating Ambient Temperature Range	T_A	0 to +70	°C
Operating Junction Temperature Range Ceramic Package Plastic Package	T_J	 175 150	°C
Storage Temperature Range	T_{stg}	-65 to +150	°C

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

ELECTRICAL CHARACTERISTICS (Unless otherwise noted specifications apply 4.75 V ≤ V_{CC} ≤ 5.25 V and 0°C ≤ T_A ≤ 70°C. Typical values measured at V_{CC} = 5.0 V, and T_A = 25°C.)

Characteristic	Symbol	Min	Typ	Max	Unit
Input Voltage — Low Logic State	V_{IL}	–	–	0.8	Vdc
Input Voltage — High Logic State	V_{IH}	2.0	–	–	Vdc
Input Current — Low Logic State (V_{IL} = 0.5 V)	I_{IL}	–	–	-200	μA
Input Current — High Logic State (V_{IH} = 2.4 V) (V_{IH} = 5.5 V)	I_{IH}	 – –	 – –	 -50 -50	μA
Input Clamp Voltage (I_{IC} = -12 mA)	V_{IC}	–	–	-1.5	V
Output Voltage — Low Logic State (I_{OL} = 48 mA)	V_{OL}	–	–	0.5	V
Output Voltage — High Logic State (I_{OH} = -10 mA) (I_{OH} = -50 mA)	V_{OH}	 2.5 2.0	 –	 – –	V
Output Short-Circuit Current (V_{IH} = 2.0 V) [2]	I_{OS}	-50	–	-150	mA
Output Leakage Current — Hi-Z State V_{IL} = 0.4 V, $V_{IL(Z)}$ = 0.8 V) V_{IH} = 2.4 V, $V_{IL(Z)}$ = 0.8 V)	$I_{OL(Z)}$	 – –	 – –	 ±100 ±100	μA
Output Leakage Current — Power OFF (V_{OH} = 6.0 V, V_{CC} = 0 V) (V_{OL} = -0.25 V, V_{CC} = 0 V)	$I_{OL(off)}$	 – –	 – –	 +100 -100	μA
Output Offset Voltage 1	V_{OS}	–	–	±0.4	V
Output Differential Voltage 1	V_T	2.0	–	–	V
Output Differential Voltage Difference 1	$V_T - \bar{V}_T$	–	–	±0.4	V
Power Supply Current	I_{CC}	–	–	95	mA

1. See EIA Specification RS-422 for exact test conditions.
2. Only one putput may be shorted at a time.



วงจรส่วนควบคุมปลายทาง