FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY

LADKRABANG

PROJECT

DISTRIBUTE CONTROL SYSTEM

PROJECT REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE BACHELOR'S DEGREE

DEPARTMENT OF INDUSTRIAL TECHNOLOGY

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

1993

ระบบควบคุมอุปกรณ์ไฟฟ้าภายในอาคาร

โดย

นายบัณฑิต	ตัญบุญยกิจ	35.102010
นายสมชาย	ศิริเศรษฐวงศ์	35.102029
นายสมศักดิ์	สมบุญจันทร์	35.102030
นายสัญชัย	พรหมเทพ	35.102032

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตร์บัณฑิต

สาขาเทคโนโลยีอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2536

หัวข้อปริญญานิพนธ์ ระบบควบคุมอุปกรณ์ไฟฟ้าภายในอาคาร (DISTRIBUTE CONTROL SYSTEM)

โดย

นายบัณฑิต	ตัญบุญธกิจ	35.102010
นายสมชาย	ศิริเศรษฐวงศ์	35.102029
นายสมศักดิ์	สมบุญจันทร์	35.102030
นายสัญชัย	พรหมเทพ	35.102032

อาจารย์ที่ปรึกษา อาจารย์ไพศาล สิทธิโยภาสกุล

อาจารย์อรรถสิทธิ์ เหล่าสกุล

ภาควิชา เทคนิคอุตสาหกรรม

ปีการศึกษา 2536

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับเป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตร์บัณฑิต

คณะกรรมการสอบปริญญานิพนธ์

..... ประธานกรรมการ

( )

..... กรรมการ

( )

..... กรรมการ

( )

..... กรรมการ

( )

ลิขสิทธิ์ของคณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เครื่องควบคุมอุปกรณ์ไฟฟ้าภายในอาคาร

DISTRIBUTE CONTROL SYSTEM

โดย นายบัณฑิต ตัญบุญยกิจ 35.102010
นายสมชาย ศิริเศรษฐวงศ์ 35.102029
นายสมศักดิ์ สมบูญจันทร์ 35.102030
นายสัญชัย พรหมเทพ 35.102032

อาจารย์ที่ปรึกษา อาจารย์ไพศาล สิทธิโยภาสกุล
อาจารย์อรรถสิทธิ์ เหล่าสกุล

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้จะกล่าวถึงการออกแบบเครื่องควบคุมอุปกรณ์ไฟฟ้าภายในอาคาร ซึ่งสามารถควบคุมอุปกรณ์ไฟฟ้าได้ 512 ห้อง โดยแต่ละห้องจะมีอุปกรณ์ไฟฟ้าได้ 8 ชิ้น และมีตัวตรวจจับพร้อมอุปกรณ์แยกเฉิน 4 ชุดต่อ 1 ห้อง การควบคุมทั้งหมดสามารถสั่งงานได้ทาง PC และแสดงสถานะของอุปกรณ์ทุกตัวได้ทางจอภาพของ PC ได้เช่นกัน โดยใช้ Z80180 CONTROL BOARD เป็นตัวควบคุมอุปกรณ์ต้นทางและปลายทาง และยังทำหน้าที่ INTERFACE ระหว่าง PC กับอุปกรณ์ต้นทางและปลายทาง Z80180 CONTROL BOARD สามารถทำงานได้อย่างอิสระทำให้เราสามารถปิดเครื่อง PC หรือนำ PC ไปใช้งานอย่างอื่นได้ โดยไม่กระทบกระเทือนต่อการทำงานของวงจร

ABSTRACT

THIS THESIS IS THE MENTION OF THE CONTROL FOR THE ELECTRICAL APPLIANCE IN THE BUILDING , WHICH CAN BE ABLE TO CONTROL THE EQUPMENT THE EQUPMENT UP TO 512 ROOMS , BY THE EACH ROOM CAN BE CONTROL FOR 8 UNITS FOR ONE ROOM , FOR THE ALL CONTROL UNITS CAN BE PROCESS AND SHOW THE ALL FUNCTION ON THE PC. IT USED THE Z80180 CONTROL BOARD TO BE CONTROL BOARD TO BE CONTROL THE ORDER OF THE START AND END EQUIPMENT

กิตติกรรมประกาศ

ในการสร้างโครงงานปริญญานิพนธ์นี้ ทางผู้จัดทำได้รับความอนุเคราะห์ช่วยเหลือ และให้คำปรึกษาแนะนำแนวทางในการดำเนินงาน ตลอดจนความช่วยเหลือทางด้านเครื่องมือที่ใช้ในการวิจัยต่างๆ จากท่านอาจารย์ไพศาล สิทธิโยภาสกุล และท่านอาจารย์อรรถสิทธิ์ เหล่าสกุล จนกระทั่งโครงงานนี้สำเร็จลุล่วงไปได้

ทีมงานผู้จัดทำโครงงานขอกราบขอบพระคุณท่านอาจารย์ ที่ให้ความกรุณามา ณ ที่นี้

คณะผู้จัดทำโครงงาน

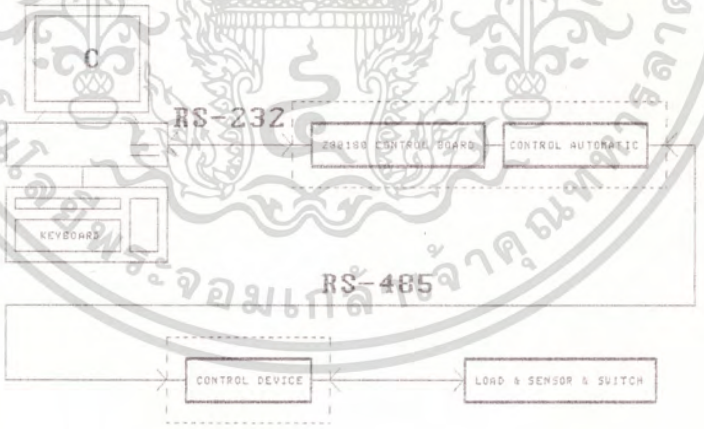
สารบัญ

ระบบควบคุมอุปกรณ์ไฟฟ้าภายในอาคาร

ระบบควบคุมอุปกรณ์ไฟฟ้าภายในอาคาร ทำหน้าที่ควบคุมอุปกรณ์ไฟฟ้าภายในห้องพักในอาคาร และรับสัญญาณจากอุปกรณ์ตรวจจับ(SENSOR) พร้อมกับส่งสัญญาณไปควบคุมอุปกรณ์ฉุกเฉินในห้องพักนั้นๆ ส่วนผลลัพธ์ของการทำงานจะแสดงที่จอ Computer เพื่อให้ผู้ใช้ได้รับรู้สถานะของอุปกรณ์ไฟฟ้าในห้องพักนั้นๆ โดยระบบนี้สามารถควบคุมอุปกรณ์ไฟฟ้าภายในห้องพักในแต่ละห้องได้หลายชิ้น รับสัญญาณตรวจจับได้ 4 I/P พร้อมกับ Switch ฉุกเฉินได้ 4 ชิ้น และสามารถควบคุมห้องพักได้สูงสุด 512 ห้อง ซึ่งแบ่งเป็น 16 ชั้น ๆ ละ 32 ห้องพัก

วัตถุประสงค์

1. เพื่อศึกษาโครงสร้างของ CPU Z80180 และประยุกต์ใช้งาน
2. เพื่อศึกษาวิธีการออกแบบโปรแกรมควบคุมการทำงานของ BOARD CONTROL Z80180 โดยใช้ภาษา ASSEMBLY พร้อมกับโปรแกรมอำนวยความสะดวกในการสั่งงาน ระบบควบคุมไฟฟ้าภายในอาคารด้วย Computer โดยใช้ภาษา C
3. เพื่อศึกษาการติดต่อสื่อสารข้อมูลอนุกรม โดยใช้ RS-232 กับ RS-485
4. เพื่อเป็นแนวทางในการนำเทคโนโลยีทางด้านอิเล็กทรอนิกส์-คอมพิวเตอร์มาประยุกต์มาประยุกต์ช่วยงานทางด้านธุรกิจ เพื่อประหยัดพลังงานและบุคลากร



ระบบควบคุมอุปกรณ์ไฟฟ้าภายในอาคารนี้จะประกอบด้วยส่วนต่างๆ 5 ส่วนทำงานร่วมกันคือ

1. ส่วน COMPUTER
2. ส่วน BOARD CONTROL Z80180 (Z180 CONTROL PACK)
3. ส่วนควบคุมแบบ AUTOMATIC
4. ส่วนควบคุมอุปกรณ์ปลายทาง
5. ส่วน LOAD และ SWITCH ฉุกเฉิน และ SENSOR

1. ส่วน COMPUTER ทำหน้าที่ แสดงผลลัพธ์การทำงานออกทางจอภาพและสามารถสั่งงานหรือทำการควบคุมอุปกรณ์ปลายทางได้โดย Keyboard หรือ Mouse ซึ่งการทำงานนี้จะต้องมีตัวโปรแกรมของระบบที่จะสั่งงานอุปกรณ์ปลายทาง และตัวโปรแกรมที่พูดถึงนี้เขียนด้วยภาษา C ส่วนนี้จะติดต่อกับส่วนที่ 2 (Board Control Z80180)โดยผ่านทาง Port COM1 หรือ COM2 ของ Computer กับ Serial Port 0 ของส่วนที่ 2 โดยส่งข้อมูลแบบอนุกรมซึ่งใช้ RS-232 เพื่อนำข้อมูลผ่าน RS-232 มาแสดงผลที่จอภาพ และเพื่อส่งข้อมูลออกไปควบคุมอุปกรณ์ปลายทาง

2. ส่วน BOARD CONTROL Z80180 (Z180 CONTROL PACK) ทำหน้าที่ ประมวลผลข้อมูลตามโปรแกรม Control1.asm (ภาษา Assembly Z180 ซึ่งใช้ควบคุมอุปกรณ์ปลายทางและรับ-ส่งข้อมูลระหว่าง Computer กับ Board Control Z80180) และทำหน้าที่ติดต่อกับส่วนที่ 1 และส่วนที่ 3 คือ ส่วน Computer เพื่อส่งข้อมูลมาประมวลผลที่ Board นี้และส่งไปควบคุมอุปกรณ์ปลายทาง หรือนำข้อมูลจาก Board นี้ ส่งให้ส่วน Computer เพื่อแสดงผลออกที่จอภาพ อีกส่วนหนึ่งก็คือ ส่วนควบคุมแบบ Automatic เพื่อส่งข้อมูลไปควบคุมอุปกรณ์ปลายทาง หรือรับข้อมูลสถานะจากอุปกรณ์ปลายทางมาประมวลผล ซึ่งส่วนนี้จะทำงานโดยขึ้นกับโปรแกรมที่สั่งให้ CPU ทำงานสำหรับการติดต่อกับส่วนต่าง ๆ

3. ส่วนควบคุมแบบ Automatic ทำหน้าที่ รับข้อมูลจากส่วนที่ 2 มา ENCODER โดย IC MC145026 แล้วส่งข้อมูลผ่าน DS75176 (Multipoint RS-485/RS-422 Transceivers) สู่ส่วนควบคุมปลายทาง ซึ่งจะเห็นว่าที่ใช้ DS75176 ก็เพื่อทำให้สามารถส่งข้อมูลได้ไกล ๆ และยังทำหน้าที่ รับข้อมูลจากส่วนที่ 4 คือส่วนควบคุมอุปกรณ์ปลายทางมา DECODER โดย MC145027 แล้วส่งข้อมูลไปให้ส่วนที่ 2 คือส่วน Board Control Z80180 เพื่อประมวลผล

4. ส่วนควบคุมอุปกรณ์ปลายทาง ทำหน้าที่ รับ-ส่ง ข้อมูลกับส่วนควบคุมแบบ Automatic เพื่อนำข้อมูลมาควบคุมอุปกรณ์ไฟฟ้า หรือนำข้อมูลสถานะต่างๆ ของอุปกรณ์ไฟฟ้า และตัวตรวจจับ (SENSOR) ส่งไปให้ส่วนควบคุมแบบ Automatic เพื่อส่งผ่านไปยังส่วน Board Control Z80180 เพื่อประมวลผล

5. ส่วน LOAD และ SENSOR และ LOAD SENSOR คือ ส่วนของอุปกรณ์ไฟฟ้าหลายชิ้น, อุปกรณ์ SENSOR 4 ชิ้น, อุปกรณ์ LOAD SENSOR 4 ชิ้น

รายละเอียดทางด้าน HARDWARE

1. ส่วน COMPUTER จะใช้ Computerที่มี PORT COM1 หรือ COM2 ซึ่งไม่มี Hardware อย่างอื่นเพิ่มเติมจึงขอกล่าวถึงส่วนโปรแกรม C ที่ใช้งานเท่านั้น

2. ส่วน BOARD CONTROL Z80180 จะประกอบด้วย CPU Z80180, I/O PORT 8255 รวมอยู่ในส่วนเดียวกัน

3. ส่วนควบคุมแบบ automatic นี้ประกอบด้วย Decoder 3 to 8 Line (74LS138) 4 to 16 Line (74154), ENCODER (MC145026), DECODER (MC145027), 74LS244, Multipoint RS-485/RS-422 Transceivers (DS75176)

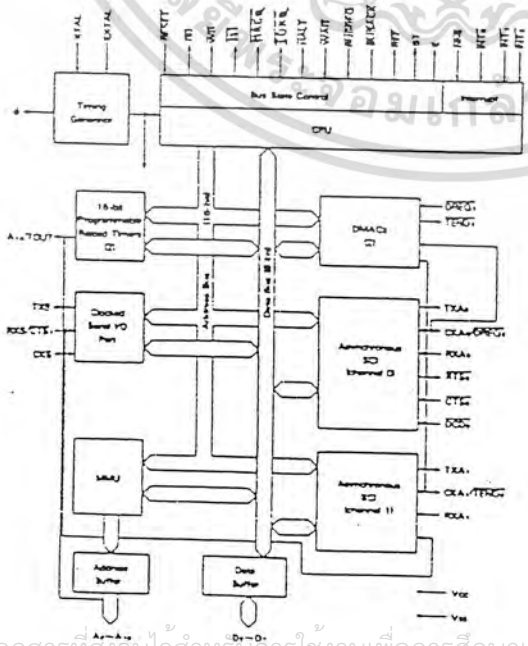
4. ส่วนควบคุมอุปกรณ์ปลายทาง ซึ่งประกอบด้วยวงจร DECODER (MC145027), ENCODER, Multipoint RS-485/RS-422 Transceivers (DS75176), T FLIP-FLOP (74LS107), Decoder 2 to 4 Line (74LS139), 74LS154, 74LS244

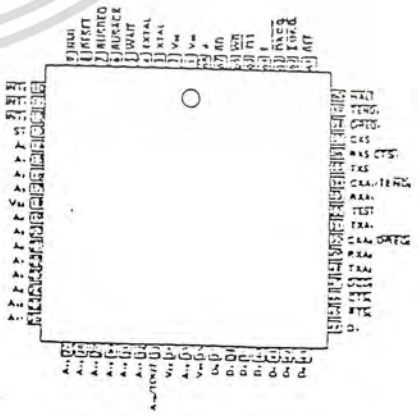
5. ส่วน LOAD และ SENSOR และ LOAD SENSOR ประกอบด้วย อุปกรณ์ไฟฟ้า,อุปกรณ์ SENSOR , อุปกรณ์ LOAD SENSOR ซึ่งอุปกรณ์ทุกตัวล้วจะใช้ผ่าน Driver เพื่อส่วยขับ LOAD ได้เต็มที่ และอาจมีวงจรเพิ่มเติมทางส่วน SENSOR แต่ละชนิด

Z80180

Z80180 เป็น CPU ที่มีความสามารถสูงที่ได้รวม CHIP สำคัญอื่นๆไว้ใน CPU CHIPเดียว จึงทำให้มีลักษณะคล้ายกับ CPU ที่ใช้ในงาน CONTROL ในจำพวก "SINGLE CHIP" แต่เนื่องจาก SINGLE CHIP มีข้อดีคือ เป็นระบบเล็กราคาถูก แต่ข้อเสียคือ การโปรแกรม CONTROL ค่อนข้างยากในตอนเริ่มต้นและกับระบบงานที่ใหญ่ขึ้น แต่ Z80180 ทางด้านโปรแกรมจะสะดวกอย่างมาก เพราะคำสั่งที่ใช้มีมาก และตรงไปตรงมาทั้งคู่มือภาษาไทย และตัวอย่างการใช้งานอย่างมากมาย เพราะCPU Z80180 นี้เป็น SUPPER COMPAT Z80 คือคำสั่งทั้งหมดยังเป็นZ80 และได้เพิ่มชุดคำสั่งขึ้นมาเพื่อเพิ่มความสะดวกในการใช้งานขึ้นอีก

เมื่อมองดูระบบ MICRO CONTROLLER "SINGLE CHIP" แล้ว Z80180 จะด้อยกว่าตรงที่ไม่มี ROM, RAM และ PORT แต่ถ้าเป็นในระดับงานอุตสาหกรรมแล้วระบบของ Z80180 กับ CHIP MICRO CONTROLLER แล้วจะไม่ต่างกันเลย เพราะความต้องการเนื้อที่ในการเก็บข้อมูลมาก และ PORT มากตาม จึงทำให้ต้องต่อเพิ่มภายนอกขึ้น จึงทำให้ Z80180 ในระดับงาน CONTROL อุตสาหกรรมคล่องตัวกว่ามาก เพราะภายใน Z80180 ประกอบด้วย เป็น CMOS, OSCILATOR ในตัว RUN ที่ 10 MHz , MMU CHIP อ้าง MEMORY 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:-





ฮากการใช้งาน

AO-A19	ADDRESS BUS ระหว่าง RESET จะเป็น HIGH IMPEDANCE
BUSAK	BUS ACKNOWLEDGE เป็นขา OUTPUT ACTIVE LOW ทำงานก็ต่อเมื่อ Z80 180 ตอบสนองต่อการขอ BUS ของ BUSRQ และจะทำให้ BUS ข้อมูล BUS ADDRESS และสัญญาณ CONTROL บางเส้นเป็น HIGH INPEDANCE
BUSRQ	BUS REQUEST เป็นขา INPUT ACTIVE LOW ซึ่งจะมีความสำคัญสูงกว่า NMI โดยจะมีการตรวจสอบสัญญาณนี้ทุกๆ การสิ้นสุดของ MACHINE CYCLE
CXA0,CXA1	ASYNCHRONOUS CLOCK 0 และ 1 เป็นขาสัญญาณ CLOCK แบบ 2 ทิศทาง คือ จะใช้เป็นขา INPUT หรือ OUTPUT ก็ได้
CXS	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
CTS0-CTS1	CLEAR TO SEND 0 และ 1 เป็นขา INPUT ACTIVE LOW ใช้ในการควบคุม MODEM
D0-D7	DATA BUS เป็นแบบ 2 ทิศทาง
DCD0	DATA CARRIER DETECT 0 เป็นขา INPUT ACTIVE LOW ใช้ควบคุมในการติดต่อกับ MODEM ของ ASCI CHANEL 0
DREQ0-DREQ1	DMA REQUEST 0 และ 1 เป็นขา INPUT ACTIVE LOW ใช้ในการขอDMA และขานี้จะโปรแกรมได้ว่าให้ตรวจสัญญาที่ขอบหรือระดับได้
E	ENABLE CLOCK เป็นขา OUTPUT ACTIVE HIGH ซึ่งใช้ซิงค์การทำงานกับอุปกรณ์ภายนอกระหว่างการทำงานเกี่ยวกับ BUS และใช้เชื่อมต่อกับอุปกรณ์ในตระกูล 68XX และ 80XX
HALT	เป็นขา OUTPUT ACTIVE LOW จะทำงานเมื่อทำคำสั่ง HALT หรือ SLP
INT0	MASKABLE INTERRUPT 0 เป็นขา INPUT ACTIVE LOW สัญญาณที่ขานี้จะถูกตรวจทุก ๆ การสิ้นสุดของคำสั่ง
INT1,INT2	เช่นเดียวกับ INT0 แต่มีระดับความสำคัญรองลงมาตามลำดับ
IORQ	เป็นขา OUTPUT เพื่อบอกว่ากำลังติดต่อกับ I/O หรือ IOE ใน 64180
M1	MACHINE CYCLE 1 เป็นขา OUTPUT ACTIVE LOW จะทำงานเมื่อFETCH OP-CODE หรือเป็นขา LIR ของ 64180

NMI NON MASKABLE INTERRUPT เป็นขา INPUT ACTIVE LOWขานี้จะตอบรับการ INTERRUPT เสมอ โดยไม่สามารถหยุดด้วย SOFTWARE

RD เป็นขาที่ใช้ทำการอ่านข้อมูลจาก MEMORY หรือ I/O

RFSH เป็นขาที่ให้ ADDRESS LOW (A0-A7) ไป REFRESH DYNAMIC RAM หรือขา REF ของ 64180

RTS0 REQUEST TO SEND เป็นขา OUTPUT ACTIVE LOW ขานี้ ใช้โปรแกรมสัญญาณควบคุมโมเด็มของ ASCI CANEL 0

RXA0, RXA1 RECEIVE DATA 0 และ 1 เป็นขารับสัญญาณจากSERIAL PORT ของASCI

RXS CLOCK SERIAL RECEIVE DATA เป็นขารับสัญญาณ SERIAL ของ CSIO

ST STATUS เป็นขา OUTPUT ACTIVE HIGH ใช้แสดงสถานะการทำงานของ CPU โดยร่วมกับ M1 และ HALT ดังตาราง:-

ST	HALT	M1	Operation
0	1	0	CPU operation (1st op-code fetch)
1	1	0	CPU operation (2nd 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)

TEND0-TEND1 TRANSFER END 0 และ 1 เป็นขา OUTPUT ACTIVE LOW ใช้แสดงถึงว่าการ 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

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/DREQ0	ระหว่าง RESET ขานี้จะเป็น CKA0 แต่ถ้า DM1 หรือ SM1 ใน DMA MODE REGISTOR (DMODE) ถูก SET เป็น 1 จะเป็นขา DREQ0
CKA1/TEND0	ระหว่าง RESET จะเป็นขา CKA1 แต่ถ้า BIT CKA1D ใน ASCI ถูก SET จะเป็นขา TEND0
RXS/CTS1	ระหว่าง RESET ขานี้จะเป็นขา RXS ถ้า BIT CTS1E ใน ASCI ถูก SET จะเป็นขา CTS1

INTERNAL I/O REGISTOR

ซึ่งมีด้วยกัน 64 I/O ADDRESS ดังแสดงในรูปะ-

	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	11H
			↓	↓
			XX010011	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	19H
			↓	↓
			XX011111	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 Reg.	RCR	XX110110	36H
	Reserved		XX110111	37H
MMU	MMU Common Base	CBR	XX111000	38H
	MMU Bank Base Regist	BBR	XX111001	39H
	MMU Common/Bank Area	CBAR	XX111010	3AH
I/O	Reserved		XX111011	3BH
			↓	↓
			XX111100	3CH
	Operation Mode Control Register	OMCR	XX111110	3DH
	I/O Control Register	ICR	XX111111	3FH

จาก MAP I/O ภายในจะเห็นว่าโปรแกรม Z80 เก่า ที่เรามีอยู่อาจจะมีการส่ง PORT ซ้ำกับ I/O ภายใน ทำให้โปรแกรมเดิมทำงานไม่ได้ สามารถแก้ไขได้โดยการโปรแกรมย้ายMAP I/O ภายใน โดยการ CONTROL BITใน 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 ภายในไป I/O ADDRESS 80H เป็นต้นไป จะโปรแกรมได้เป็น

```
LD   A, 80H
OUT0 (3FH), A
```

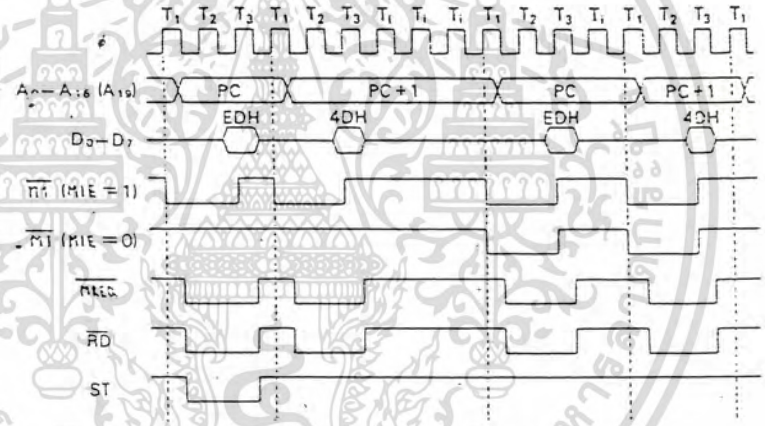
ส่วน IOSTP : IOSTOP MODE BIT 5 เป็น 1 จะทำให้ I/O ภายในหยุดทำงานเมื่อ RESET BIT นี้จะเป็น 0

OPERATION MODE

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

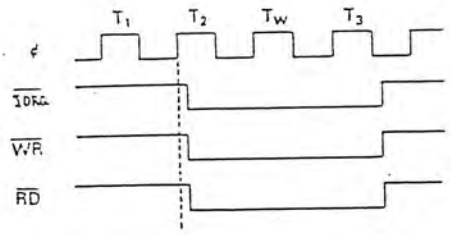
BIT	7	6	5	4	3	2	1	0
	M1E	M1TE	IOC					
	(R/W)	(W)	(R/W)					

M1E (M1 ENABLE) ระหว่าง RESET BIT นี้จะเป็น 1 M1 OUTPUT จะเป็น LOW เมื่อ FETCH OPCODE และเนื่องจากการทำคำสั่ง RETI ของ Z80180 จะถูกกระทำ 2 ครั้ง ใน 1 คำสั่ง จึงทำให้เกิด M1 ขึ้น 2 ครั้งด้วย อันอาจจะทำให้เกิด INTERRUPT เข้ามาได้เมื่อยังทำไม่หมดคำสั่ง ด้วยเหตุนี้ BIT M1E จะถูก SET เป็น 0 สำหรับ Z80180 เพื่อให้ M1 ถูกทำงานปกติคือ เมื่อทำคำสั่ง RETI จะมี M1 เพียงครั้งเดียวดังรูป:-

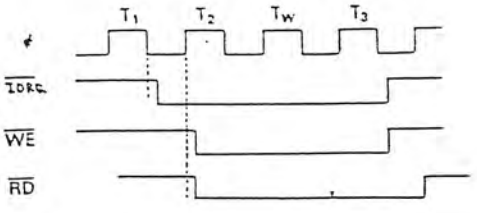


M1TE (M1 TEMPORARY ENABLE) ใช้กับการต่อ INTERFACE กับ Z80 PIO

IOC เป็น BIT ใช้ควบคุม TIMING ของ IORQ และ RD ให้เหมือน Z80 หรือ 64180 โดยถ้า BIT นี้ถูก SET เป็น 1 TIMING จะเป็นของ 64180 คือ IORQ และ RD จะ ACTIVE ที่ขอบขาลงของ T1 แต่ถ้า BIT นี้เป็น 0 TIMING จะเป็นของ Z80 คือ จะ ACTIVE ที่ขอบขาขึ้นของ T2 เพื่อให้ใช้อุปกรณ์สนับสนุนของ Z80ได้ ระหว่าง RESET BIT นี้จะเป็น 1 ดังรูป:



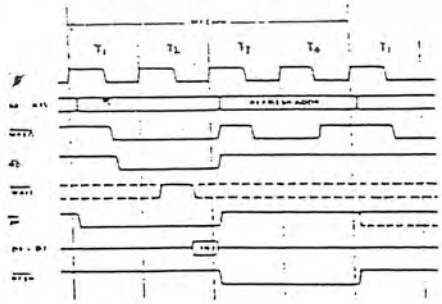
I/O Read Write Cycle When $\overline{IOC}=0$

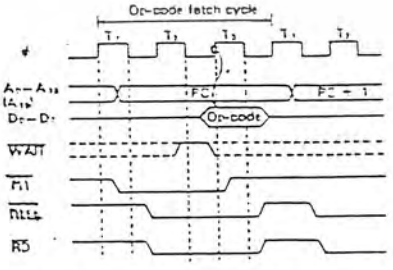


I/O Read Write Cycle When $\overline{IOC}=1$

เกี่ยวกับ TIMING

ให้ดูรายละเอียดในคู่มือฉบับภาษาอังกฤษ แต่กล่าวสรุปได้ว่า Z80180ใช้เวลาในการทำคำสั่งใน 1 MACHINE CYCLE น้อยกว่า Z80 อย่ 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 ทำงานด้วยความถี่ที่สูงขึ้น จึงอาจทำให้ MEMORY หรือ I/O ทำงานไม่ทันจึงต้องมีสัญญาณมาเป็นตัวช่วยกำหนดความพร้อมระหว่าง CPU กับอุปกรณ์ภายนอกนั่นก็คือ สัญญาณ WAIT ซึ่ง Z80 นั้นจะต้องให้อุปกรณ์ภายนอกส่งสัญญาณนี้มาให้ แต่ 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				

BIT7, 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

BIT5, 4 IWI1, IWI0 (I/O WAIT INSERTION)

จะทำการเพิ่ม WAIT STATE ให้กับ I/O ภายนอกจาก 1-6 ดังตาราง

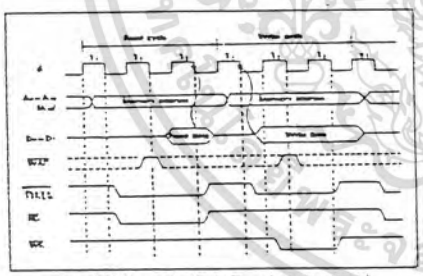
IWI1	IWI0	I/O ภายนอก	INT0
0	0	1	2
0	1	2	4
1	0	3	5
1	1	4	6

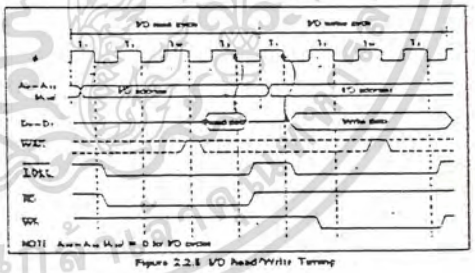
จะเห็นว่า WAIT STATE ของ I/O มากกว่า MEMORY อยู่ 1 T STATE เพราะขณะเข้าถึง I/O ปกติ WAIT STATE จะถูกเพิ่มขึ้น 1 อยู่แล้ว ดังนั้นเมื่อเพิ่ม WAIT STATE เข้าไปก็จะรวมกับที่มีอยู่ปกติ และส่วน INTO ก็เช่นเดียวกัน ขณะเกิด INTO ปกติ จะมี WAIT STATE อยู่ 2 WAIT STATE อยู่แล้ว และขณะที่ RESET BIT CONTROL WAIT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

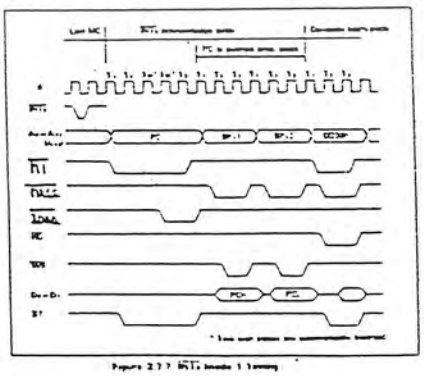
ตัวอย่างเช่น เราต้องการเพิ่ม WAIT STATE ในการเข้าถึง MEMORY 2 WAIT STATE จะโปรแกรม ดังนี้:-

```
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 มีม
                ;  การกำหนดเกี่ยวกับ DMA  ดังนั้นเราจึง FILL
                ;  เฉพาะ BIT ที่จะต้องการโปรแกรม
```

ดูรูปขณะ WAIT STATE ปกติ







HALT และ LOW POWER MODE

มีด้วยกัน 4 MODE คือ

HALT MODE	โดยทำคำสั่ง 76H จะทำให้ CPU หยุดทำคำสั่ง แต่การทำงานต่างๆ ของ 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 หยุดทำงานการออกจากSLEEP MODE โดยการ RESET หรือ INTERRUPT
IOSTOP MODE	ใช้หยุดการทำงาน 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 MODEโดยการ SET BIT ใน ICR แล้วตามด้วยคำสั่ง SLP จะทำให้ IO ภายในหยุดทำงานและ CPU หยุดทางานเพื่อเป็นการประหยัดพลังงาน 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 (PHYSTCAL) โดยการแบ่ง 64 KBYTE LOGICAL (คือ ADDRESS ปกติที่ใช้เช่นเดียวกับ Z80) เป็น 3 ส่วน ในการใช้งานด้วยกัน 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 2^2 คือ 4 รูปแบบ ดังรูป:-

1

2

เช่น A1 > BANK > A0

CBAR = D4H

A1 > BANK = A0 (OH)

CBAR = FOH (ตอน RESET เป็นเช่นนี้)

3

A1 = BANK > A0

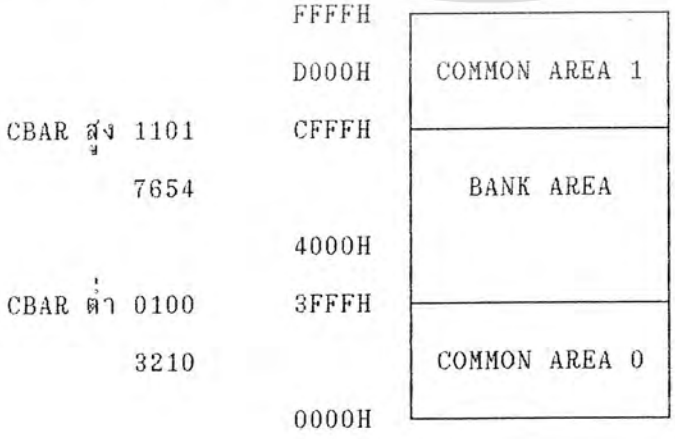
CBAR = FFH

4

A1 = BANK = A0 (OH)

CBAR = 00H

MAP LOGICAL ปกติ



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

จากที่กล่าวมาเรายังไม่พูดถึงการขยายMEMORY 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 ADDRESSของส่วนนั้น ๆ (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

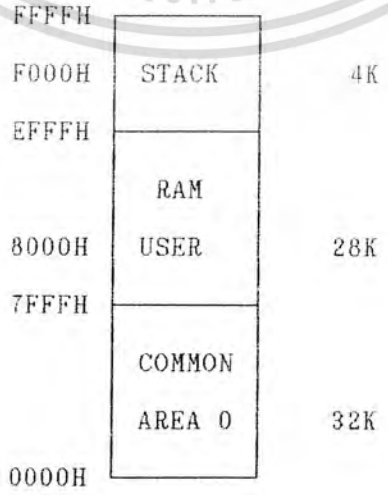
CA 3 - CA 0 เป็นตัวกำหนด ADDRESS เริ่มต้นของ COMMON AREA 1

BA 3 - BA 0 เป็นตัวกำหนดจุด ADDRESS สุดท้าย ของพื้นที่ 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-7FFFHและ RAM ใช้งานที่ 10000H โดยมีพื้นที่ STACK ที่ 18000H จากนั้นต้องกำหนด 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 RAM USER (BANK) ก็เช่นเดียวกัน

```
  10000
- 8000
-------
  08000
```

ที่ BBR = 08 H

ดังนั้นการโปรแกรมจากโจทย์ตัวอย่างก็จะเป็น

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

เมื่อคำนวณกลับจะได้ CBAR นิบเบิ้ลต่ำ ADDRESS สุดท้ายของ BANK เป็น

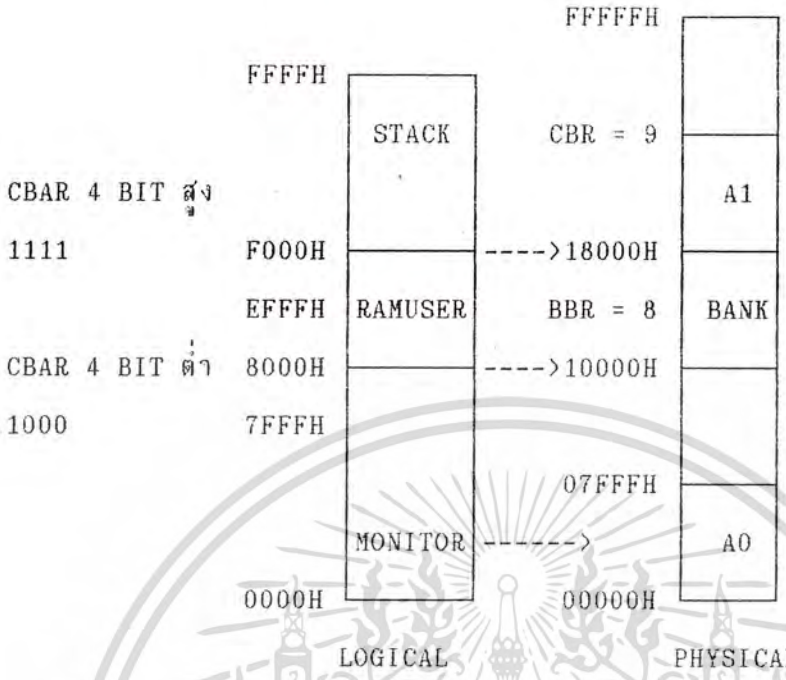
8000 + (BBR = (8) X 1000 H) = 10000 H

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
```



สรุป

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 ของพื้นที่นั้น ๆ

INTERRUPT

มีด้วยกัน 12 INTERRUPT แบ่งเป็น 4 INTERRUPT ภายนอก และ 8 INTERRUPT ภายใน โดยมีลำดับความสำคัญจากมากไปหาน้อย ดังนี้ TRAP(ภายใน),(ภายนอก) NMI, INT0, INT1 , 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 INT1, INT2 และ INTERRUPT ภายในทั้งหมดยกเว้น "TRAP" โดย 3 BIT สูงของ IL สามารถโปรแกรมได้ แต่ 5 BIT หลังจะถูก FIX ดังรูป:-

Interrupt Source	Priority	IL	Fixed Code
		b7 b6 b5	b4 b3 b2 b1 b0
INT 1	Highest	. . .	0 0 0 0 0
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 ENABLE2, 1, 0 ใช้ ENABLE และ DISABLE INTERRUPT ภายนอก ถ้าเป็น 0 จะ DISABLE แต่ BIT นี้จะไม่ทำให้เกิด INTERRUPT ขึ้นทันทีจนกว่าจะทำคำสั่ง EI ดังนั้น INTO จะต่างกับ Z80 ตรงที่มีส่วนนี้ แต่เมื่อเกิด RESET ITE0 จะถูก SET เป็น 1 โดยอัตโนมัติเพื่อให้ขึ้นกับ คำสั่ง EI หรือ DI อย่างเดียว เช่น Z80 แต่ ITE1และ ITE2 จะเป็น 0

TRAP จะเป็น 1 เมื่อทำคำสั่งที่ไม่มีใน Z80180 TRAP สามารถRESET 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 ให้คือ ถ้าเป็นคำสั่ง OPCODE 2 BYTE UFO จะเป็น 0 และจะทำให้PC 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 2000H ก็จะเกิด INTERRUPT TRAP ขึ้น และรู้ด้วยว่าเป็นคำสั่ง 2 BYTE และ PC ก็ชี้คำสั่งถัดไปคือ ADDRESS 2002 แต่ FLAG UFO จะถูกทำให้เป็น 0 เพื่อปรับค่า PC นี้ด้วยการลดลง 1 เช่น ADDRESS 2001 ซึ่งก็คือตำแหน่งข้อมูลที่ผิดนั่นเอง

TRAP INTERRUPT เป็นเหมือน NMI คือ ไม่สามารถหยุดได้เมื่อเกิดกระทำคำสั่งผิดขึ้นซึ่งเป็นตัวช่วยให้เกิดความน่าเชื่อถือทางด้าน SOFTWARE และอาจใช้เพิ่มคำสั่งได้อีกด้วย BIT TRAP ใน ITC จะถูก SETเป็น 1 และUFO 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 ทุก ๆ 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 ทุก ๆ 2 ms (หรือ 256 ครั้ง ทุก ๆ 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	1.25μs	1.66μs	2.5μs	4.0μs
0	1	20 states	2.5 μs	3.3 μs	5.0μs	8.0μs
1	0	40 states	5.0 μs	6.6 μs	10.0μs	16.0μs
1	1	80 states	10.0μs	13.3μs	20.0μs	32.0μs

DMA CONTROLLER (DMAC)

มีด้วยกัน 2 CHANEL เพื่อเป็นการเพิ่มความเร็วในการ TRANFER ข้อมูล โดยการกระทำไม่ต้องผ่าน CPU โดยมีความสามารถดังนี้

MEMORY ADDRESS SPACE	โดยสามารถกำหนดตำแหน่ง SOURCE และ DESTINATION ที่ใดก็ได้ใน 1024 KBYTE
I/O ADDRESS SPACE	กำหนดที่ใดก็ได้ใน 64 KBYTE ทั้ง SOURCE และDESTINATION
TRANSFER LENGTH	ใช้เป็น COUNTER ในการ TRANSFER ได้เป็น BLOCK ๆ ละ 64 K BYTE
DREQ	เป็นขา INPUT จะตรวจจับที่ระดับหรือขอบของสัญญาณ
TEND	เป็นขา OUTPUT เพื่อบอกกับอุปกรณ์ภายนอกว่าทำ DMAหมด BLOCK แล้ว
TRANSFER RATE	การ TRANSFER แต่ละครั้ง จะเกิดทุก ๆ 6 CLOCK และ WAIT STATE สามารถเพิ่มเข้าไปใน DMA ได้ สำหรับ MEMORY หรือ I/O ที่ทำงานช้า ที่ระบบ SYSTEM CLOCK (ϕ) = 6 MHz อัตราการ TRANSFER จะสูงถึง 1 MBYTE 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) คือเมื่อ TRANSFER ครบ 1 หรือ 2 BYTE ก็จะคืน BUS ให้ UP จนอุปกรณ์พร้อมก็จะ TRANSFER ข้อมูลต่อ ทำอย่างนี้สลับกันไปจนหมด BLOCKใช้สำหรับอุปกรณ์ที่ทำงานช้าและBURST (TRANSFER แบบต่อเนื่อง) คือ ทำจนจบ BLOCK จึงจะคืน BUS ให้ UP
CHANEL 1	จะใช้กับ MEMORY <-> I/O โดย MEMORY ADDRESS เพิ่มหรือลดได้

DMAC REGISTOR

CHANEL 0 มี SAR0 (I/O ADDRESS 20H-22H) เป็นตัวกำหนด SOURCE ADDRESS ได้ถึง 1 MBYTE หรือ 64 KBYTE สำหรับ I/O

DAR0 (I/O ADDRESS 23H-25H) ใช้กำหนด DESTINATION เช่นเดียวกับ SAR0

BCR0 (I/O ADDRESS 26H-27H) ใช้กำหนด BYTE ในการ TRANSFER โดยสูงสุดได้ 64 KBYTE และเมื่อทำการ TRANSFER 1 BYTE REGISTOR ตัวนี้จะลดลง 1

CHANEL 1 มี MAR 1 (I/O ADDRESS 28H-2AH) ใช้กำหนด PHYSICAL ADDRESS ได้ถึง 1 MBYTE โดยอาจจะให้เป็น SOURCE หรือ DESTINATION ก็ได้

IAR1 (I/O ADDRESS 2BH-2CH) ใช้กำหนด ADDRESS ของI/O โดยอาจจะให้เป็น SOURCE หรือ DESTINATION ก็ได้

BCR1 (I/O ADDRESS 2EH-2FH) เช่นเดียวกับ BCR0

DMA/WAIT CONTROL REGISTOR (DCNTL I/O ADDRESS 32H) ประกอบด้วย

BIT	7	6	5	4	3	2	1	0
	MWI1	MWI0	IWI1	IWI0	DMS1	DMS0	DIM1	DIM0

MWI1, MWI0 : MEMORY WAIT STATE INSERTION ใช้กำหนดจำนวน WAIT STATE ของ CPU หรือ DMAC ดูหัวข้อ WAIT STATE GENERATOR

IWI1, IWI0 : I/O WAIT INSERTION กำหนด WAIT STATE I/O หรือ CPU ดูหัวข้อ WAIT STATE GENERATOR

DMS1, DMS0 : DMA REQUEST SENSE ใช้กำหนดการรับรู้ของขา INPUT DREQ0 , DREQ1 เมื่อเป็น 0 จะตรวจที่ LEVEL และถ้าเป็น 1 จะตรวจที่EDGE ระหว่าง RESET 2 BIT นี้ จะเป็น 0

DIM1, DIM0 : DMA CHANEL 1 I/O และ MEMORY MODE ใช้กำหนด SOURCE และ DESTINATION ของ CHANEL 1 เมื่อ RESET 2 BIT นี้จะเป็น 0 สามารถ SET ได้ดังตาราง

DIM1	DIM0	TRANSFER MODE	ADDRESS INCREMENT/DECREMENT
0	0	MEMORY=>I/O	MAR1+1 ,IAR1 FIXED
0	1	MEMORY=>I/O	MAR1-1 ,IAR1 FIXED
1	0	I/O=>MEMORY	IAR1 FIX , MAR1+1
1	1	I/O=>MEMORY	IAR1 FIX , MAR1-1

ASYNCRONOUS SERIAL COMMUNICATION INTERFACE (ASCI)
มีด้วยกัน 2 CHANEL โดยมี BLOCK DIAGRAM ดังรูป:-

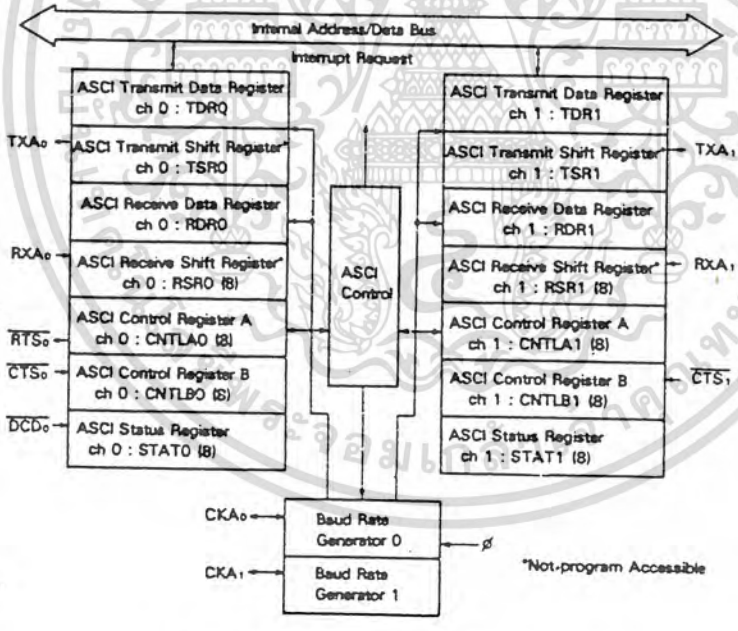


Figure 60. ASCI Block Diagram

TSR 0,1 เป็น SHIFT REGISTOR ที่รับข้อมูลจาก TRANSMIT DATA REGISTOR (TDR) แล้วนำข้อมูลนั้น SHIFT ออกที่ขา TXA

TDR 0,1 (I/O ADDRESS 06H, 07H) เป็น REGISTOR ที่ใช้ส่ง DATA ออกไปที่ขา TXA โดยการนำข้อมูลใน TDR ส่งไปที่ TSR

เมื่อ TSRว่างลงและสามารถที่จะเขียนข้อมูลเข้าไปที่ TDR ได้อีก ในขณะที่ TSR กำลัง SHIFT ข้อมูลออกไปที่ขา TXA

RSR 0,1 เป็น REGISTOR ที่รับข้อมูลจาก RXA PIN เมื่อรับเต็ม BUFFER แล้วก็จะ SHIFT ไปที่ RDR ถ้า RSR ไม่ว่างเมื่อมีการรับข้อมูล BYTE ต่อไปเข้าอีกจะเกิดข้อมูลทับซ้อนกันขึ้น จะทำให้เกิดการผิดพลาดและผลของการผิดพลาดก็จะแสดงที่ REGISTOR สถานะ REGISTOR นี้ไม่สามารถโปรแกรมได้

RDR 0,1 (I/O ADDRESS 08H, 09H) คือ REGISTORที่ใช้เก็บข้อมูลที่รับเข้ามาจาก RXA PIN และในขณะที่ RDR กำลังบรรจุข้อมูลที่รับเข้ามาจาก RSR ข้อมูล BYTE ถัดไปสามารถรับเข้ามาต่อได้

STAT 0,1 (I/O ADDRESS 04H, 05H) แต่ละ CHANEL จะมี REGISTOR ใช้สำหรับตรวจสอบการสื่อสาร เกี่ยวกับการผิดพลาดและ สถานะสัญญาณ CONTROL MODEM การ ENABLE และ DISABLE ASCI ดังรูป:-

BIT	7	6	5	4	3	2	1	0
STAT0	RDRF	OVRN	PE	FE	RIE	DCDO	TRDE	TIE

BIT	7	6	5	4	3	2	1	0
STAT1	RDRF	OVRN	PE	FE	RIE	CTSIE	TRDE	TIE

RDRF : RECEIVE DATA REGISTOR FULL จะถูก SET เป็น 1 เมื่อข้อมูลที่รับเข้ามาถูกส่งเข้ามาที่ RDR เรียบร้อยแล้ว (ครบ BYTE) แต่ถ้าการรับเกิด ERRORขึ้น RDRF ก็จะถูก SET ค้างและข้อมูลที่ผิดนั้นก็จะถูกส่งมาที่ RDR และคงอยู่ ดังนั้นจะต้องทำการ CLEAR FLAG ERROR RDRF จะถูก CLEAR เป็น 0 เมื่ออ่าน RDR, DCDO เป็น HIGH สำหรับ CHANEL 0, IOSTOP และการ RESET

OVRN : OVERUN ERROR จะเป็น 1 เมื่อ RDR เต็ม และ RSR เต็มแล้วยังมีการรับข้อมูลเข้ามาอีกจะถูก CLEAR ได้เมื่อ EFR BIT ใน CNTLA เป็น 0,DCD0 เป็น HIGH IOSTOP และ RESET

PE : PARITY ERROR เป็น 1 เมื่อข้อมูลที่รับเข้ามา PARITY ผิด และ CLEAR ได้เช่นเดียวกับ OVERUN

FE : FRAMING ERROR เมื่อข้อมูลที่รับเข้ามารูปแบบผิดไปจากที่กำหนด BIT FE จะถูก SET เป็น 1 และการ CLEAR เช่นเดียวกับ OVERUN

RIE : RECEIVE INTERRUPT ENABLE เมื่อเป็น 1 จะอนุญาตให้ ASCI ทำการขอ INTERRUPTได้ เมื่อ RDRF, OVRN, PE หรือ FE ถูก SET เป็น 1 ด้วยเมื่อนั้น ASCI ก็จะให้สัญญาณINTERRUPT สำหรับ ASCI CHANEL 0 INTERRUPT สามารถเกิดขึ้นโดยการเปลี่ยนแปลงที่ขารับสัญญาณ INPUT ภายนอกที่ขา DCD0 จาก LOW เป็น HIGH และ RIE จะถูก CLEAR เป็น 0 ระหว่าง RESET

DCD0 : DATA CARRIER DETECT BIT นี้จะถูก SET เป็น 1 เมื่อขา INPUT DCD0 เป็น HIGH และจะถูก CLEAR เป็น 0 จากการอ่าน STAT 0 ครั้งแรกจากนั้นขา INPUT DCD0 จะถูกเปลี่ยนจาก HIGH เป็น LOW และระหว่าง RESET เมื่อ DCD0 เป็น 1 ส่วนของภาครับจะไม่ทำงาน

CTSIE: CHANEL 1 CTS ENABLE ก็ CHANEL 1 มีขา INPUT CTS1 ภายนอก ซึ่งMULTIPLEX กับ RXS เมื่อ SET BIT นี้เป็น 1 จะถูกเลือกเป็นขา CTS1

TDRE : TRANSMIT DATA REGISTOR EMPTY เป็นตัวบอกว่าข้อมูลพร้อมที่จะส่งได้หรือไม่ ถ้าเป็น 1 คือ พร้อมที่จะส่งข้อมูลแล้วให้เขียนข้อมูลเข้าไปที่ TDR ได้ และเมื่อมีการเขียนข้อมูลเข้าไปที่ TDR ก็จะทำให้ TDRE เป็น 0 และข้อมูลในTDR ก็จะถูกส่งให้ TSR จน TDR ว่างลง TDRE ก็จะกลับเป็น 1 อีกครั้ง

TIE : TRANSMIT INTERRUPT เมื่อเป็น 1 จะอนุญาตให้ ASCI ใช้การส่งแบบINTERRUPTได้ โดยที่ TDRE ต้องเป็น 1 ด้วย TIE จะถูก CLEAR เป็น 0 ระหว่าง RESET

CNTLA 0,1 (I/O ADDRESS 00H-01H)เป็น REGISTORกำหนดการทำงานประกอบด้วย

BIT 7 6 5 4 3 2 1 0

CNTLA0	MPE	RE	TE	RTS0	MPBR	MOD2	MOD1	MOD0

BIT	7	6	5	4	3	2	1	0
CNTLA1	MPE	RE	TE	CKAID	MPBR	MOD2	MOD1	MOD0

MPE : MULTIPROCESSOR MODE ENABLE ใช้ ENABLE ในการสื่อสารแบบไมโครโปรเซสเซอร์ร่วมจากเมื่อมีการเลือก MODE การสื่อสารแล้ว (MP = 1 ใน CNTLB) ในการสื่อสารแบบนี้ FORMAT ของการสื่อสารจะมี BIT พิเศษเพิ่มเข้ามาเรียกว่า MPB BIT ซึ่ง BIT นี้จะถูกใช้ในการตรวจสอบหรือใช้งานเมื่อ ENABLE MPE ให้เป็น 1 และถ้า MPB = 1เมื่อนั้นภาครับของ MULTI PROCESSOR จะทำงาน คือ RDRF และ ERROR FLAG จะทำงาน แต่ถ้า MPB = 0 ASCI จะไม่สนใจข้อมูล BYTE นั้นๆ ถ้า MPE = 0จะไม่สามารถทำการสื่อสารแบบไมโครโปรเซสเซอร์ร่วมได้ แม้จะ SET MP เป็น 1 แล้วก็ตาม

RE : RECEIVER ENABLE ถ้าเป็น 1 จะENABLE การรับของ ASCI แต่ถ้าเป็น0 จะ DISABLE การรับ แต่ RDRFและ ERROR FLAG จะไม่ถูก RESET ตาม

TE : TRANSMIT ENABLE เป็น 1 จะENABLE การส่ง ถ้าเป็น 0 จะ DISABLE แต่ TDRE FLAG จะไม่ถูก RESET ตาม

RTS0 : REQUEST TO SEND CHANEL 0 เป็น BIT ที่ให้ผลเช่นเดียวกับขา0UTPUT RTS0 คือถ้า BIT นี้เป็น 1 ขา0UTPUT RTS0 ก็จะเป็น 1 ถ้า BIT นี้เป็น 0 ขา0UTPUT ก็เป็น 0 RTS0 BIT นี้จะถูก SET เป็น 1 ระหว่าง RESET

CKAID: CKA1 CLOCK DISABLE ซึ่งขา CKA1 จะ MULTIPLEX กับ TENDO เมื่อ BITนี้เป็น 1 จะเลือกเป็นขา TENDO แต่ถ้าเป็น 0 ก็จะเป็นขา CLOCKของ ASCI CHANEL 1 BIT นี้จะเป็น 0 ระหว่าง RESET

MPBR/EFR MULTIPROCESSOR BIT RECEIVE / ERROR FLAG เมื่อ BITนี้ถูกอ่านจะใช้ดู MPB BITในกรณีที่พบไมโครโปรเซสเซอร์ที่จะทำการติดต่อกันแล้ว และจะ DISABLE ไมโครโปรเซสเซอร์ตัวอื่นๆ ก็โดยการส่ง MPB BIT ให้เป็น 0 เมื่ออ่านจะได้รู้ว่า MPB เป็น 0 จริง แต่ถ้าเขียน 0 ให้ BIT นี้จะเป็นการ RESET ERROR FLAG ในการรับ

MOD 2, 1, 0 : ASCI DATA FORMAT MODE 2, 1, 0 โดย

MOD 2 = 0 7 BIT MOD1 = 0 NO PARITY

1 8 BIT 1 PARITY ENABLE

MOD 0 = 0 1 STOP BIT

1 2 STOP BIT

ASCI CONTROL REGISTOR B 0, 1 (CNTLB 0, 1 I/O ADDRESS 02H, 03H)
ประกอบด้วย

BIT	7	6	5	4	3	2	1	0
	MPBT	MP	CTS/PS	PEO	DR	SS2	SS1	SS0

MPBT : MULTIPROCESSOR BIT TRANSMIT ใช้ส่ง MPB BIT โดยถ้า MPBT = 1 เมื่อนั้น MPB BIT = 1 และ MPBT = 0 MPB ก็ = 0 ด้วย ระหว่าง RESET ไม่สามารถกำหนดได้

MP : MULTIPROCESSOR MODE ถ้าเป็น 1 จะเป็นการ SET การติดต่อแบบไมโครโปรเซสเซอร์ร่วม โดยใช้ FORMAT ของ MOD 2 กับ MOD 0 โดยยกเว้น MOD 1 ดังนี้ START BIT + 7 หรือ 8 DATA BIT + MPB BIT + 1 หรือ 2 STOP BIT ระหว่าง RESET MP จะเป็น 0

CTS/PS: CLEAR TO SEND /PRESCALE เมื่ออ่าน BIT นี้จะใช้แสดงสถานะของขา INPUT CTS ภายนอก ถ้าขา CTSเป็น HIGH ภาคส่งของ ASCI จะไม่ทำงาน แต่ถ้าเขียนเข้าไปที่ BIT นี้จะเป็นการกำหนด BAUD RATE BIT นี้เป็น 0 ระหว่าง RESET

PEO : PARITY EVEN ODD BIT นี้จะไม่มีผลต่อการ ENABLE หรือ DISABLE ของ PARITY (MOD 1 ใน CNTLA) แต่จะใช้เลือกว่าเมื่อมีการ ENABLE PARITY ใน MOD 1 จะให้ PARITY คู่หรือคี่ ถ้า PEO = 0 คือ คู่ แต่ถ้า = 1 คือ คี่

DR : DIVIDE RATIO ใช้กำหนด BAND RATE BIT นี้จะเป็น 0 ระหว่าง RESET

SS2, 1, 0 : SOURCE / SPEED SELECT 2, 1, 0 ใช้กำหนด CLOCK ว่าจะให้เป็นภายในหรือภายนอก(โดยภายนอก คือขา CLOCK CKA) และเป็นตัวกำหนด BAUD RATE ด้วย ระหว่าง RESET ทั้ง 3 BIT นี้จะเป็น 1 คือ เป็นการใช้CLOCK จากภายนอกนั่นเองซึ่งจากที่กล่าวมาในการกำหนด BAUD RATE

CLOCK SERIAL I/O PORT (CSI/O)

มี 1 CHANEL ซึ่งเป็น SYNCHRONOUS SERIAL I/O PORT โดยใช้ได้เฉพาะเป็น HALF DUPLEX เท่านั้นและ DATA ถูกกำหนดเป็น 8 BIT โดย CLOCK ที่ใช้ในการซิงค์เลือกได้ว่าจะใช้จาก SYSTEM CLOCK หรือ CLOCK ภายนอกที่ขา (CKS) ก็ได้ ซึ่ง CSI/O ประกอบไปด้วย 2 REGISTOR คือ:-

CSI/O TRANSMIT/RECEIVE DATA REGISTOR (TRDR I/O ADDRESS 0BH) ใช้ในการส่งและรับข้อมูล โดยระบบต้องเป็น HALF-DUPLEX(คือการส่งและรับจะเกิดพร้อมกันไม่ได้)

CSI/O CONTROL/STATUS REGISTOR (CNTR I/O ADDRESS 0AH) เป็นตัวบอกสถานะ

PROGRAMMABLE RELOAD TIMER (PRT)

มีด้วยกัน 2 CHANEL เป็น 16 BIT PROGRAMMABLE RELOAD TIMER และสำหรับ CHANEL 1 มีขา OUTPUT สามารถให้สัญญาณได้ทั้ง 2 CHANEL ประกอบด้วย:-

TIMER DATA REGISTOR (TMDR : I/O ADDRESS - CH0 ; 0DH, 0CH, CH1 ; 15H, 14H)

เป็น REGISTOR 16 BIT ใช้กำหนด TIMER โดยADDRESS I/O สูงเก็บค่า TIMER ค่าสูงระหว่าง RESET TMDR0 และ TMDR1 จะเป็น 0FFFFH โดย TMDR จะนับลง 1 ครั้งทุก ๆ 20 CLOCK SYSTEM เมื่อ TMDR นับลงเป็น 0 ค่าใน RELOAD จะถูก LOAD มาให้ TMDR โดยอัตโนมัติ การอ่านค่าใน TMDR อ่านได้เลยโดยไม่ต้องหยุด PRT แต่ถ้าเป็นการเขียนต้องหยุด PRT ก่อน

TIMER RELOAD REGISTOR (RLDR : I/O ADDRESS - CH0 ; 0FH, 0EH, CH1 ; 17H , 16H)

ใช้ LOAD ค่าที่อยู่ใน RLDR ไปให้ TMDR เมื่อ TMDR ลดลงเป็น 0

SECONDARY BUS INTERFACE

E CLOCK OUTPUT TIMING เป็นสัญญาณ BUS ที่ 2 เพื่อใช้เชื่อมต่อ INTERFACE เป็นไปได้ง่ายกับอุปกรณ์ PERIPHERAL ในตระกูลอื่นๆ เช่น 68XX และ 80XX และเป็นสัญญาณที่ทำให้ระบบเกิดความน่าเชื่อถือในการใช้งาน เพราะจะติดต่อกับอุปกรณ์ก็ต่อเมื่อมีสัญญาณที่ขานี้ จากรูปจะที่ MREQ หรือ IORQ จะเกิดขึ้นช่วงที่ T STATE แรก ซึ่งยังไม่ใช่ช่วงของ DATA ที่อ่านหรือเขียน จึงทำให้อาจเกิดข้อมูลผิดพลาดกับอุปกรณ์ภายนอก แต่สัญญาณจะให้สัญญาณ ACTIVE HIGH เมื่อมีการจ่ายหรือรับ DATA เท่านั้น

FREE RUNNING COUNTER (18H)

เป็นREGISTOR I/O ที่ใช้อ่านได้อย่างเดียวใช้สำหรับการ REFRESH DYNAMIC RAM ซึ่งเป็น COUNTER นับลง 8 BIT(A0-A7) แบบอิสระโดยจะนับลง 1 ครั้งทุก ๆ 10 CLOCK และถ้าเกิดมีการเขียนข้อมูลไปที่ REGISTOR นี้ จะทำให้ช่วงเวลาของการ REFRESH DYNAMIC RAM, BAUD RATE ของ ASCI และ CSI/O ไม่แน่นอน

ถึงแม้อยู่ใน IO STOP MODE ก็ตาม FRERUNNING COUNTER นี้ก็ยังนับอยู่อย่างต่อเนื่อง ซึ่งในขณะ RESET จะมีค่าเป็น OFFH

คำสั่งเพิ่มเติม 12 คำสั่ง

SLP เมื่อใช้คำสั่งนี้ CPU จะหยุดทำงานบางอย่างทำให้ใช้กำลังงานต่ำ

MLT MULTPLY ใช้สำหรับคูณเลข 8 BIT 2 จำนวน โดยผลลัพธ์จะเป็น 16 BIT โดย REGISTOR ที่ใช้ในการคูณอาจจะเป็น BC, DE, HL หรือ SP โดยผลลัพธ์จะได้ที่ REGISTOR คู่นั้น

OTIM, OTIMR, OTDM, OTDMR - BLOCK I/O

เป็นคาสั่ง OUT PORT เป็น BLOCK ของ PORT ADDRESS ต่ำ A0-A7 เท่านั้น คือจะทำการ OUT ข้อมูลเป็น BLOCK โดยที่ PORT เพิ่มหรือลดตามจำนวนข้อมูล โดยใช้ HL เป็นตัวชี้ข้อมูลที่จะ OUT ออกไป และ C เป็น NUMBER PORT ในคำสั่ง OTIM และ OTDM ก็คือจะเพิ่มค่า HL ที่ชี้ขึ้นเป็นหนึ่งหรือลดลง 1 ตามด้วย PORT เพิ่มขึ้นหรือลดลงด้วยและค่า B จะลดลง 1 ซึ่ง B จะเป็น COUNTER ในการส่ง DATA ส่วน OTIMR และ OTDMR จะมีลักษณะเช่นเดียวกับ OTIM และ OTDM เพียงแต่จะทำการส่งข้อมูลเพิ่มขึ้นหรือลง และ PORT NUMBER เพิ่มขึ้นหรือลดลงตามค่า B จนกระทั่ง B = 0

TSTIO ใช้สำหรับ TEST I/O PORT คือ จะทำการอ่านค่า PORT ที่กำหนดโดย REGISTOR C เข้ามาแล้วทำการ AND กับ DATA 8 BIT ที่ต้องการ โดยที่ค่าข้อมูลที่ IN เข้ามานั้นไม่เปลี่ยนแปลง แต่จะให้ผลที่ FLAG และ PORT ที่ IN เข้ามาจะเป็นเฉพาะ ADDRESS ต่ำ A0-A7 เท่านั้นสามารถเปรียบเทียบเป็นโปรแกรมได้ดังนี้:-

```
XOR   A                 LD C, NUMBER PORT
IN    A, (PORT)         TSTIO  70H
LD    B, A              JP Z, OK
LD    A, 70H
```

```
AND  B
JP   Z, OK
```

TST g-TEST REGISTOR โดยค่าที่กำหนดใน REGISTOR จะ AND กับ ACCUMULATOR ซึ่งจะทำให้มีผลต่อ FLAG ตามคำสั่ง AND แต่ค่าใน ACCUMULATOR และREGISTOR ไม่เปลี่ยนแปลง เช่น ตัวอย่าง:-

```
LD  A, 7          LD A, 7
LD  C, A          TST B
AND B             JR Z, OK
LD  A, C
JR  Z, OK
```

TST m-TEST IMMEDIATE เช่นเดียวกับ REGISTOR เพียงแต่ข้อมูลเป็น DATA โดยตรงที่ AND กับ ACCUMULATOR

TST (HL) - TEST MEMORY คือ จะนำค่าใน MEMORY ที่ถูกชี้โดย HL AND กับ ACCUMULATOR โดยค่าทั้ง 2 ไม่เปลี่ยนแปลงแต่ให้ผลการกระทำที่ FLAG

INO g (m) - INPUT, IMMEDIATE I/O IN ค่าจาก PORT 8 BIT (A0-A7) มายัง REGISTOR ใด ๆ ก็ได้ A, BC, DE, HL

OUTO (m), g-OUTPUT, IMMEDIATE I/O OUT ค่าจาก REGISTOR ใด ๆ ไปยัง PORT 8 BIT (A0-A7) REGISTOR ก็มี A, BC, DE, HL

CODE คำสั่งใหม่

ข้อกำหนดต่าง ๆ ในการใช้งาน

1. การป้อนคำสั่งจะผ่านทาง KEYBOARD ของเครื่อง PC โดยแต่ละคำสั่งจะต้อง KEY ให้ถูกต้องตามรูปแบบที่กำหนด และคำสั่งเหล่านั้นจะถูกกระทำเมื่อกด ENTER
2. ผู้ใช้สามารถใช้อักษรตัวเล็กหรือตัวใหญ่ได้ตามต้องการ
3. จำนวนตัวเลขต่าง ๆ ที่ป้อนเข้าไปจะถือเป็นเลขฐาน 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. การออกจากสภาวะต่าง ๆ ใช้ KEY ESC จะทำให้กลับสู่เครื่องหมาย PROMPT ตามเดิม
7. การกระทำกับข้อมูลที่ต้องเป็น RAM หากมีการกระทำในพื้นที่เป็น ROM หรือ RAM ที่ถูก PROTECT ไว้ FUNCTION ที่ใช้นั้นจะไม่ถูกกระทำและจะมีข้อความบอก
8. เมื่อมีการทำคำสั่งที่ไม่มีในชุดคำสั่ง Z80180 จะเกิด TRAP INTERRUPT ขึ้นและจะแสดงค่าตำแหน่งที่ผิดนั้นบน DISPLAY
9. RAM บน BOARD ที่ใส่ไปกับเครื่องจะเป็น 8 KBYTE แต่การอ้าง RAM บน DEBUG 180 จะเป็นของขนาด 32 KBYTE คือ ของ USER ตั้งแต่ 0800H-0EFFFH และ 0F000H-0FFFFH จะเป็นที่เก็บพารามิเตอร์ร่วมของทุก ๆ PAGE รวมทั้งเป็นส่วนของ STACK ทั้ง USER และ SYSTEM ด้วย ดังนั้นจะเกิดทับซ้อนกันขึ้น 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 ออกเป็น 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 เพราะเมื่อ

เราเขียนข้อมูลเกินที่เก็บภายในตัว RAM ก็จะทำให้กลับมาเริ่มที่ตำแหน่งเริ่มต้นของตัวเอง อีก (เมื่อวงจรเลือก 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 SOFTWARE PROCOMใส่ใน DRIVE แล้วเรียก Z180 ก็จะเข้าสู่การทำงานของ PROCOM เพื่อใช้เป็นตัวติดต่อกับ BOARD Z80180 ที่ DISPLAY จะมีข้อความปรากฏ

```
A>Z180
```

รายละเอียดคร่าว ๆ ด้านล่างของจอ จะบอกสถานะต่าง ๆ ไว้โดยเมื่อ KEY ALT Z จะเป็น HELP MENU

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

การ LOAD ข้อมูลจาก PC ไปยัง BOARD Z80180 ใช้ PAGE UP

การ LOAD ข้อมูลจาก BOARD Z80180 ใช้ PAGE DOWN

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 ทั่ว ๆ ไป

5. เมื่อเริ่มจ่ายไฟให้ BOARD SOFTWARE 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 อื่น เครื่องจะไม่สามารถเข้าสู่ DEBUG ได้ต้อง RESET BOARD Z80180 ใหม่ แล้วทำอย่างการเดิมถูกต้องก็จะมีข้อความ

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

ตอนนี้ก็จะใช้คำสั่งต่างๆ ที่มีใน DEBUG ได้แล้ว แต่ถ้าการตั้ง BAUD RATE บน PC ไม่ได้อยู่ที่ CENTER เมื่อเริ่มจ่ายไฟหรือกด RESET จะไม่มีคำ Z180 ขึ้นมาแสดง ซึ่งอาจจะเป็นอักษรที่ไม่สามารถสื่อความหมายได้หรืออาจไม่ขึ้นอะไรเลย ก็ให้กด ENTER 1 หรือ 2 ครั้ง ก็จะมีข้อความและเครื่องหมาย PROMPT 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 ต่าง ๆ ก็โดยการกด KEY "?" แล้วกด ENTER ก็จะขึ้น HELP MENU ซึ่งแบ่งออกเป็น 3 PAGE จะรอการกด KEY ใด ๆ ก็ได้ก็จะแสดง PAGE ต่อไป
2. เนื่องจาก Z80180 อ้าง MEMORY ได้ 1 MBYTE แต่คำสั่งต่าง ๆ ที่มีให้ใช้สามารถอ้างเกิน 64 KBYTE จึงใช้เทคนิคในการแบ่งเป็น PAGE ๆ ละ 64K โดย SOFTWARE ของ DEBUG จะ PROGRAM แบ่งเป็น ROM 0000H-7FFFH และ RAM 8000H-EFFFH ซึ่งเป็นของ USER ในการใช้งาน และส่วน F000-FFFFH จะใช้เป็นที่เก็บ PARAMETER ต่าง ๆ และเป็นส่วน STACK ที่ใช้ร่วมกันทุก ๆ PAGE เช่นเดียวกับตำแหน่ง ROM ดังนั้นข้อมูลใน PAGE อื่นจะเชื่อมต่อกันโดยส่วน STACK นี้

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

3.1 เมื่อ DUMP MEMORY หรือดูการ DISASSEMBLER เป็นจำนวนมาก ๆ สามารถหยุดการแสดงผลโดย "CONTROL S" และจะให้แสดงผลต่อก็กด KEY ใด ๆ ก็ได้

3.2 เมื่อกด KEY ESC จะทำให้กลับสู่ MAIN PROGRAM คือกลับสู่เครื่องหมาย PROMPT ยกเว้น ในขณะใช้ "?" HELP MENU

3.3 เมื่อกระทำการ RUN โปรแกรมของ USER สามารถทำงานใน FUNCTIONอื่นได้อีก 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 โปรแกรมของผู้ใช้ก่อนอันดับแรกเมื่อมีการเปิดเครื่อง (AUTO START) และสามารถออกจาก AUTO START ได้ภายใน 5 วินาที ที่เปิดเครื่อง โดยการกด KEY ใด ๆ ก็ได้บน PC จากนั้นก็ให้เข้าไป CLEAR ADDRESS EFFFH ถ้าไม่ต้องการให้เกิด AUTO START อีกเมื่อเปิดเครื่อง

5. เมื่อต้องการ RUN PROGRAM ที่เขียนขึ้นโดยโปรแกรมที่เขียนขึ้นไม่ให้วน LOOP ต้องมีการหยุดโปรแกรมอาจจะเขียนด้วย HALT หรือ RST 18H ซึ่ง RST 18H จะหยุดโปรแกรมโดยการแสดงค่า REGISTOR ส่วน HALT จะทำให้ CPU หยุดทำคำสั่งและจะให้ทำงานต่อก็โดยการกด ESC เพื่อกลับสู่ PROMPT

6. ระบบเรียกใช้โปรแกรมย่อย "SYSTEM CALL" ซึ่งมีลักษณะเป็นโปรแกรมสำเร็จรูปมี 32 โปรแกรมด้วยกัน ซึ่งทำให้อำนวยความสะดวกแก่การศึกษาและทดลองให้เป็นไปได้อย่างดี

7. โปรแกรมทดลอง เพื่อการศึกษาและเป็นแนวคิดให้เกิด IDEA ใหม่ ๆ ขึ้น

คำสั่ง A

ทำหน้าที่ 2 อย่างด้วยกันคือ ตาราง ASCII และ MINI ASSEMBLE

รูปแบบ A , A addr

คำสั่ง B (BREAK)

ทำหน้าที่ หยุดโปรแกรมเพื่อ CHECK ค่า REGISTOR ต่าง ๆ ซึ่งมีรูปแบบหลายอย่างดังนี้

รูปแบบ

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

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 ข้อมูลไม่ตรงกันเครื่องก็จะแสดง ADDRESS นั้นให้ทราบ

คำสั่ง D (DISPLAY)

ทำหน้าที่

```
D
D addr
D , addr
D start addr , final addr
```

คำสั่ง E

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

EDIT จะช่วยให้ไม่ต้องเขียนโปรแกรมใหม่

รูปแบบ E ; ENTER ตามค่า ADDRESS ล่าสุดที่จำเอาไว้

E addr ;

E start addr , final addr , + byte

คำสั่ง F

ทำหน้าที่ แบ่งเป็น FILL และFIND 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

คำสั่ง G (GO)

ทำหน้าที่ RUN โปรแกรมที่ผู้ใช้เขียนขึ้น ซึ่งมีรูปแบบ ดังนี้

G <ENT> ; RUN ตาม PC ที่อยู่ขณะนั้น

G addr <ENT> ; RUN จาก ADDRESS ที่กำหนด

G , addr ; RUN จาก PC ถึง ADDRESS ที่กำหนด

G addr1 , addr2; RUN จาก ADDRESS1 ถึง ADDRESS2

คำสั่ง H (HEX MATCH)

ทำหน้าที่ ใช้สำหรับคำนวณต่าง ๆ แบ่งเป็นฐาน 10 และ 16 (HEX)

รูปแบบ H NUMBER (HEX) , NUMBER (HEX) ; ใช้คำนวณ + - x และ /

H s NUMBER (HEX) ; เปลี่ยนเลข HEX เป็น DECIMAL

H NUMBER (DECIMAL) ; เปลี่ยนเลข DECIMAL เป็น HEX

คำสั่ง I (INPUT)

ทำหน้าที่ สำหรับอ่านข้อมูลจาก PORT

รูปแบบ I PORT

I PORT , L

คำสั่ง K (KEEP TEXT FILE)

ทำหน้าที่ รับข้อมูลที่เป็น TEXT FILE

รูปแบบ K addr

คำสั่ง L (LOAD)

ทำหน้าที่ สำหรับการ DOWN LOAD ข้อมูลจาก PC ลงบน BOARD

รูปแบบ L

L addr (offset)

คำสั่ง M (MOVE)

ทำหน้าที่ สำหรับการ COPY ข้อมูลเป็น BLOCK ๆ

รูปแบบ M start addr , final addr , destination

หมายเหตุ การใช้คำสั่ง MOVE นี้จะไม่คำนึงถึง PAGE เพราะฉะนั้นเวลาทำการ COPY ข้อมูล ต้องคำนึงถึง 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

คำสั่ง O (OUTPUT)

ทำหน้าที่ สำหรับการส่งข้อมูลไปยัง PORT

รูปแบบ O PORT , HEX

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

คำสั่ง Q (QUIT)
ทำหน้าที่ สำหรับการออกจากโปรแกรม DEBUG
รูปแบบ Q <ENT>

คำสั่ง R (REGISTOR)
ทำหน้าที่ สำหรับการกำหนดและดูค่า REGISTOR
รูปแบบ R REG , VALUE

คำสั่ง S (SHOW EXAMPLE PROGRAM)
ทำหน้าที่ โหลดโปรแกรมตัวอย่างใน MONITOR มาไว้ยัง RAM USER

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

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

คำสั่ง V (VARIEBLE)

ทำหน้าที่ สำหรับกำหนด RST , INTERRUPT ภายในและภายนอก

รูปแบบ V VAR , VALUE

คำอธิบาย ใช้กำหนดค่า ADDRESS สำหรับการ RESTART , INTERRUPT ทั้งภายในและภายนอกกรณีไม่ใส่ค่า 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
```

ในกรณี ผู้ใช้จะใส่ค่าแหน่ง ADDRESS ของการ RESTART หรือ INTERRUPT ไว้ในการเขียนโปรแกรมเลย โดยไม่ต้องใช้ FUNCTION V ก็ให้ใส่ค่า ADDRESSในตำแหน่งดังต่อไปนี้:-

	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 จากส่วนต่าง ๆ อาทิ เช่น INTERRUPT 38 TIMER นอกจากใช้ FUNCTION V แล้วเมื่อเราเขียนโปรแกรม อาจจะไม่สะดวกสำหรับการที่ต้องจำตำแหน่งที่จะต้อง INTERRUPT เราสามารถใส่ตำแหน่งที่จะให้ไปทำโปรแกรม INT ไว้ยังตำแหน่ง ADDRESS ที่กำหนดไว้ให้ดังค่า 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 ใด ๆ ก็ได้ แต่ถ้ามีการ DI ดังที่กล่าวมาแล้ว ก็ทำได้โดยต่อสาย SERIAL ให้เรียบร้อย เมื่อเริ่มจ่ายไฟเข้าเครื่องภายใน 5 วินาที ให้กด KEY "ESC" บนเครื่อง PC หรือ KEY ใด ๆ ก็ได้ ก็จะเข้าสู่การทำงานของ MONITOR และเมื่อไม่ต้องการให้เกิด AUTO RUN อีกให้ CLEAR ตำแหน่ง 0EFFFH โดยใส่ค่า 00H ไปที่ตำแหน่งนี้

4. การทำ AUTO RUN จะมีการเก็บ PARAMETER ต่าง ๆ นั้นก็คือ PAGE ที่ใส่ AUTO RUNไว้ ค่า BAUD RATE ที่ติดต่ออยู่ขณะนั้น ถ้ามีการตั้ง AUTO RUN ไว้หลาย PAGE โปรแกรมจะทำการ RUN AUTO RUN NUMBER น้อยก่อน และถ้าผู้ใช้เกิดไปเปลี่ยน BAUD RATE บน PC ทำให้เมื่อต้องการออกจาก AUTO RUN ทำไม่ได้ วิธีแก้ก็คือ 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 นี้จะมีประโยชน์มากในกรณีที่เรา RUN โปรแกรมแล้ว BREAK โปรแกรม ให้แสดงค่าสถานะต่างๆ ออกที่ PRINTER เพื่อ CHECK ดูความถูกต้องของโปรแกรม จะทำให้สะดวกและรวดเร็วขึ้น เช่น ในกรณี STEP, TRACE, BREAK, DISASSEM เป็นต้น

6. การใช้ SYSTEM CALL ต่าง ๆ ต้องคำนึงถึงการใช้ REGISTOR ด้วย

การโปรแกรม EPROM

การโปรแกรม EPROM เราจะใช้ UNIVERSAL PROGRAMMER & TESTER ซึ่งเป็นเครื่องที่ใช้ในการโปรแกรม EPROM,EEPROM,FLASH EPROM,PAL,GAL,PEEL,EPLD และสามารถ Test IC TTL ตระกูล 74/54, CMOS 40/45, DRAM, SRAM, PALได้อีกด้วย ซึ่งลักษณะตัวเครื่องจะเป็นกล่องที่มีช่อง SOCKET ขนาด 40 ขาอยู่ 1 อัน ซึ่งมีไว้สำหรับใส่ IC ที่จะ Copy หรือ Test ซึ่งตัวเครื่องจะต่อกับ CARD ที่ให้มากับเครื่องทาง PARALLEL PRINTER PORT ของ PC ทำให้การทำงานเร็วกว่า PORT RS-232 ทั่วไปถึง 10 เท่า และยังมีภาคจ่ายไฟอยู่ในตัวอีกด้วย

การใช้งานก็เพียงแต่ต่อCARD ลงบน SLOT PC จากนั้นก็ต่อสาย Connectระหว่าง CARD กับ เครื่อง UNIVERSAL PROGRAMMER & TESTER แล้ว RUN Software ที่ให้มากับเครื่องเท่านี้ก็สามารถจะโปรแกรมหรือ Test IC ได้แล้ว ซึ่งภายใน Disk ที่ให้มาจะมี File อยู่หลาย File แต่ที่เราจะกล่าวถึงเพียง 5 File คือ EPP512.EXE

ขั้นตอนการโปรแกรม EPROM

การโปรแกรม EPROM จะมีอยู่ 2 วิธีคือ 1. Copy ระหว่าง EPROM อีกตัวไปยัง EPROM อีกตัว 2. Copy File ลงสู่ EPROM ซึ่งจะเห็นว่า การโปรแกรมวิธีแรกจะง่ายกว่า เพียงแต่ท่านหา EPROM ที่มีโปรแกรมอยู่แล้วมาเสียบลงที่ SOCKET ของเครื่องโปรแกรม แล้วท่านก็ LOAD ข้อมูลใน EPROM ออกมาดู จากนั้นให้นำ EPROM ตัวใหม่ที่ยังไม่ได้โปรแกรมเสียบลงที่ SOCKET แทนตัวแรก แล้วเลือกอักษร A ก็จะทำการ Copy โปรแกรมลงสู่ EPROM ตัวใหม่ จากนั้นก็ Check ให้แน่ว่าข้อมูลที่ Copy ลงไปถูกต้องหรือไม่ โดยเลือกอักษร C ก็จะทำการเปรียบเทียบ ถ้าไม่ Error ก็แสดงว่า Copy สมบูรณ์ ซึ่งการโปรแกรมลักษณะแรกนี้ง่ายและคล้ายๆกับวิธีที่ 2 เราจึงกล่าวเพียงวิธีที่ 2 เท่านั้น คือ การ Copy File ลงสู่ EPROM ซึ่งจะมีขั้นตอนดังนี้

1. ให้คีย์โปรแกรมใส่ SK.COM (Notepad) และต้องตั้งชื่อ File *.asm (หรืออาจจะใช้โปรแกรม wordstar ก็ได้) แล้วนำไป ASSEMBLER ให้กลายเป็น File *.obj หรือถ้าเป็น Z80 ก็จะใช้โปรแกรม T80 ซึ่งสะดวกสำหรับ Z80 เพียงแต่ท่านคีย์โปรแกรมที่จะ Copy ใส่ในโปรแกรม T80 แล้ว Assembler ด้วยโปรแกรม T80 และจะมีให้เลือกอีกว่าต้องการให้

เป็น File *.bin, *.hex, *.lst ซึ่งจะสะดวกมาก แต่ข้อเสียคือ ใช้กับ Z80 เท่านั้น ต่อมาเมื่อได้ File *.bin หรือ *.obj แล้ว ก็เข้าสู่โปรแกรมโดย File EPP512.EXE

2. เมื่อเข้าสู่ File EPP512.EXE แล้วจะเห็น Menu แรก ซึ่งจะต้อง Enter เพื่อเข้าสู่ Main Menu และจะเห็นว่า Cursor กระพริบอยู่บรรทัดล่างสุดเพื่อรอรับการตอบว่าจะเลือก function ใด ซึ่งแต่ละ function จะมีหน้าที่ดังนี้ คือ

1. DOS SHELL - ออกจากโปรแกรม EPP512.EXE เพื่อเข้าสู่ DOS ชั่วคราว และถ้าท่านจะกลับเข้าสู่โปรแกรมอีกก็เพียงแต่พิมพ์ EXIT แล้วกด Enter ก็จะเข้าสู่โปรแกรมทันที

2. Load BIN file to buffer - จะใช้ Load File *.bin, *.obj เข้าสู่โปรแกรม

3. Save buffer to disk - Save File ลงบนแผ่น Disk

4. Edit buffer - ใช้แก้ไขตำแหน่ง start , end , destination , data , ASCII , binary

5. Change I/O base addr - เปลี่ยนฐานตำแหน่ง I/O ซึ่งเป็นตำแหน่งของ CARD ที่ต่อกับ SLOT ของ Computer ซึ่งถ้าไม่ตรงตำแหน่งก็จะ Copy ไม่ได้

6. Display loaded file history - แสดงชื่อไฟล์ที่ Load โดยฟังก์ชัน 2

7. Display buffer - แสดง File ที่ Load มา

T. Type select - เลือกขนาดของ EPROM

M. Mfr.select - เลือกบริษัทที่ผลิต EPROM

S. Program Speed,algo - เลือกความเร็ว และ Voltage (Vpp) ในการ Copy EPROM

Z. Target zone - เลือกตำแหน่งของ EPROM

B. Blank check - เช็ค EPROM ว่ามีข้อมูลอยู่หรือไม่

P. Program - ใช้ Program ลง EPROM ออกมาดู

R. Read - อ่านข้อมูลที่ Program ลง EPROM ออกมาดู

C. Compare & display error - ใช้เปรียบเทียบข้อมูลที่โปรแกรมกับข้อมูลใน EPROM

Q. Quit - ออกจากโปรแกรม EPP.EXE

D. Display - แสดงผลข้อมูล

A. Auto (B & P & V) - ทำฟังก์ชั่น B, P, V เรียงตามลำดับ

V. Verify - ใช้เปรียบเทียบระหว่าง buffer กับ EPROM

3. เลือกฟังก์ชั่น 5 เพื่อเลือกตำแหน่ง DIP SW ที่ปรับบน CARD
4. เลือกฟังก์ชั่น 2 เพื่อ Load File *.bin, *.obj (ใช้ปุ่ม tab เพื่อเลือก File)
5. เลือกฟังก์ชั่น 6 เพื่อแสดง File ที่ Load มา
6. เลือกฟังก์ชั่น 7 เพื่อแสดงข้อมูลใน buffer
7. เลือกฟังก์ชั่น T เพื่อเลือกขนาด EPROM
8. เลือกฟังก์ชั่น M เพื่อเลือกบริษัทที่ผลิต EPROM หรือเบอร์ EPROM
9. เลือกฟังก์ชั่น S เพื่อเลือกความเร็วและ voltage Vpp ที่ใช้ในการ copy EPROM
10. เลือกฟังก์ชั่น Z เพื่อเลือกตำแหน่งของ buffer และตำแหน่งของ EPROM
11. เลือกฟังก์ชั่น A เพื่อทำการ check และโปรแกรมข้อมูลลง EPROM
12. เลือกฟังก์ชั่น C เพื่อทำการเปรียบเทียบข้อมูลที่โปรแกรมกับข้อมูลใน EPROM

ขั้นตอนการ TEST IC TTL,CMOS,SRAM,DRAM

ให้เข้าสู่ File ICTEST.EXE ก็จะปรากฏ Main Menu คล้าย ๆ กับ ขั้นตอนการโปรแกรม EPROM หรือ PAL จากนั้นก็เลือกตำแหน่ง I/O ที่ขึ้นอยู่กับการปรับ DIP SW บน CARD แล้วจึงนำ IC ที่จะ Test เสียบลง Connector แล้วเลือกฟังก์ชั่น S และกด Y หรือปุ่มบนเครื่อง ก็จะปรากฏเบอร์ของ IC ที่ทำการทดสอบ (แต่ถ้าไม่ปรากฏก็แสดงว่าไม่สามารถ Test IC เบอร์นั้นได้) จากนั้นให้เลือกฟังก์ชั่น T เพื่อเลือกเบอร์และชนิดของ IC ที่จะ Test เมื่อเลือกเสร็จแล้วให้เลือก L หรือ F เพื่อ Test IC และถ้าปรากฏ OK ก็แสดงว่า IC นั้นไม่เสีย ซึ่งถ้าเป็น SRAM และ DRAM และมี Menu ย่อยต่างหากอีก ซึ่งจะคล้าย ๆ กันและใช้เหมือนกัน

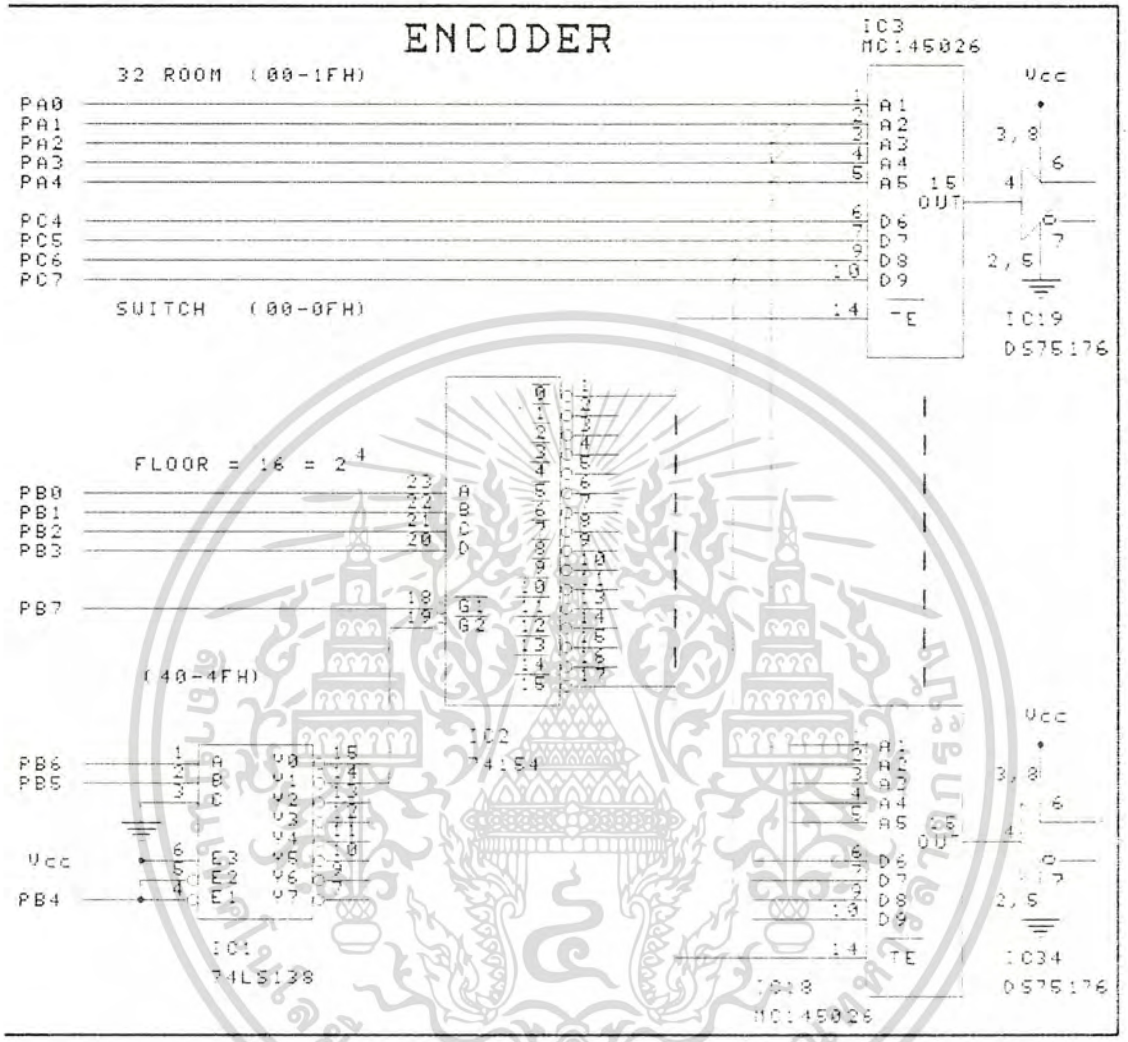
ขั้นตอนการออกแบบและการสร้าง

เนื่องจากส่วน Hardware ที่ต้องออกแบบสร้างมีอยู่ 2 ส่วนใหญ่ ๆ เราจึงแยกกล่าวทีละส่วนดังนี้

1. ส่วนควบคุมแบบ Automatic ซึ่งจะแบ่งตามหน้าที่ได้เป็น 2 ส่วน คือ
   1.1 ENCODER
   1.2 DECODER
2. ส่วนควบคุมอุปกรณ์ปลายทาง ซึ่งจะแบ่งตามหน้าที่ได้เป็น 2 ส่วน คือ
   2.1 DECODER
   2.2 ENCODER

1. ส่วนควบคุมแบบ Automatic จะต่อกับส่วน Board Control Z80180 ทาง Port ของ 8255 ทั้ง 3 Port คือ PA,PB,PC และ จะกำหนดให้ 8255 ทำงานใน Mode 0 คือ PA,PB,PC เป็น I/O ทั้งหมด ซึ่งจะกำหนดให้ PA_0-PA_7, PB_0-PB_7, PC_4-PC_7 เป็น O/P และ PC_0-PC_3 เป็น I/P ส่วนการออกแบบสร้างส่วนนี้จะแยกเป็น 2 ส่วน คือ

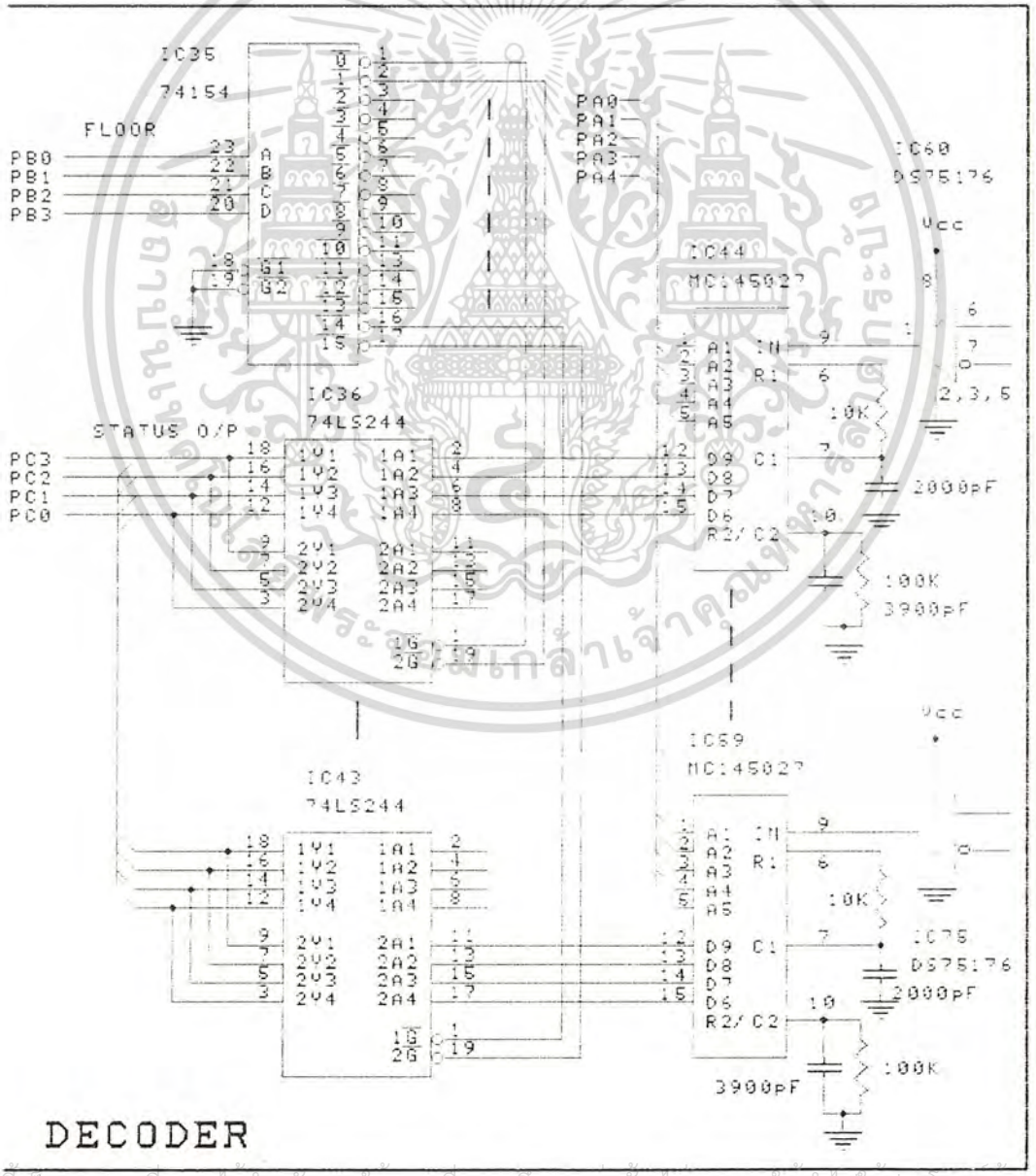
1.1 ENCODER จะใช้ IC 74LS138, 74154, MC145026x16, DS75176x16 ส่วนนี้จะนำข้อมูลทาง I/O Port ของ 8255 มาทำเป็นข้อมูลในการควบคุมอุปกรณ์ไฟฟ้าและ SWITCH ลูกเฉิน รวมกันทั้งหมด 12 สัญญาณ ด้วยเหตุนี้ เราจึงใช้ ENCODER MC145026 ซึ่งแปลงข้อมูล 9 Bit(A_1-A_5,D_8-D_9) ขนานเป็นอนุกรม ส่งให้ Multipoint RS-485/RS-422 Transceivers ยังส่งข้อมูลไปให้ส่วนควบคุมอุปกรณ์ปลายทาง (ซึ่งสังเกตว่าข้อมูล 4 Bit(PC_4-PC_7) จะให้ความแตกต่างถึง 16 ค่า คือ 0-F ซึ่งเราจะใช้ข้อมูล 0-B สั่งงานอุปกรณ์ไฟฟ้าและ SWITCH ลูกเฉิน แต่ละตัวรวม 12ชิ้น และข้อมูล C-D ใช้ Decoder IC 74LS139 เพื่อเลือกรับสถานะของอุปกรณ์ไฟฟ้า และ SWITCH ลูกเฉิน กับ SENSOR รวมทั้งหมด 12 ชิ้น และข้อมูล E ใช้ CLEAR T FLIP-FLOP และข้อมูล F ใช้กริก MC145026ของ ENCODER ของส่วนควบคุมอุปกรณ์ปลายทาง ให้ส่งข้อมูลสถานะกลับมายัง DECODER ของส่วนควบคุมแบบ Automatic) ส่วนต่อไปคือ ส่วนถอดรหัสเลือกชิ้น ซึ่งจะใช้ PB_4-PB_6 ต่อกับ 74LS138 และ PB_7 ต่อกับ 74154 และ O/P Y1 ที่ต่อกับ 74154 และ PB_0-PB_3 ต่อกับ 74154 ดังรูป จะเห็นว่า IC 74154 กับ 74LS138 จะทำงานได้เมื่อข้อมูล PB_0-PB_7 มีค่า 0 1 0 0 X X X X = 40H - 4FH ซึ่งค่า PB_0-PB_3 ใช้เลือกชิ้นที่ต้องการ(1-16) ส่วนค่า PB_4-PB_7 ใช้ควบคุม 74LS138,74151ให้



ทางาน ซึ่งจะเห็นว่าเมื่อ 74LS138 ทางาน ก็จะทาให้ 74154 ทางานตาม เป็นผลให้ O/Pของ 74154 ที่ตรงกับค่า PB_0-PB_3 นั้นทางาน เช่น ค่า E ก็จะทาให้ Y_{14} ทางาน ก็จะทาให้ MC-145026 ของชั้นนั้นทางาน จึงทาให้ข้อมูลขนาด 9 Bit (A_1-A_5,D_6-D_9) ถูกแปลงเป็นข้อมูลอนุกรม และส่งผ่านข้อมูลให้กับ DS75176 เพื่อแปลงข้อมูลอนุกรมแบบ UNBALANCE ไปเป็นข้อมูลอนุกรมแบบ BALANCE ซึ่งจะทาให้สัญญาณนั้นส่งไปได้ไกลๆ ซึ่งถูกส่งต่อให้ DECODER ของส่วนควบคุมอุปกรณ์ปลายทาง

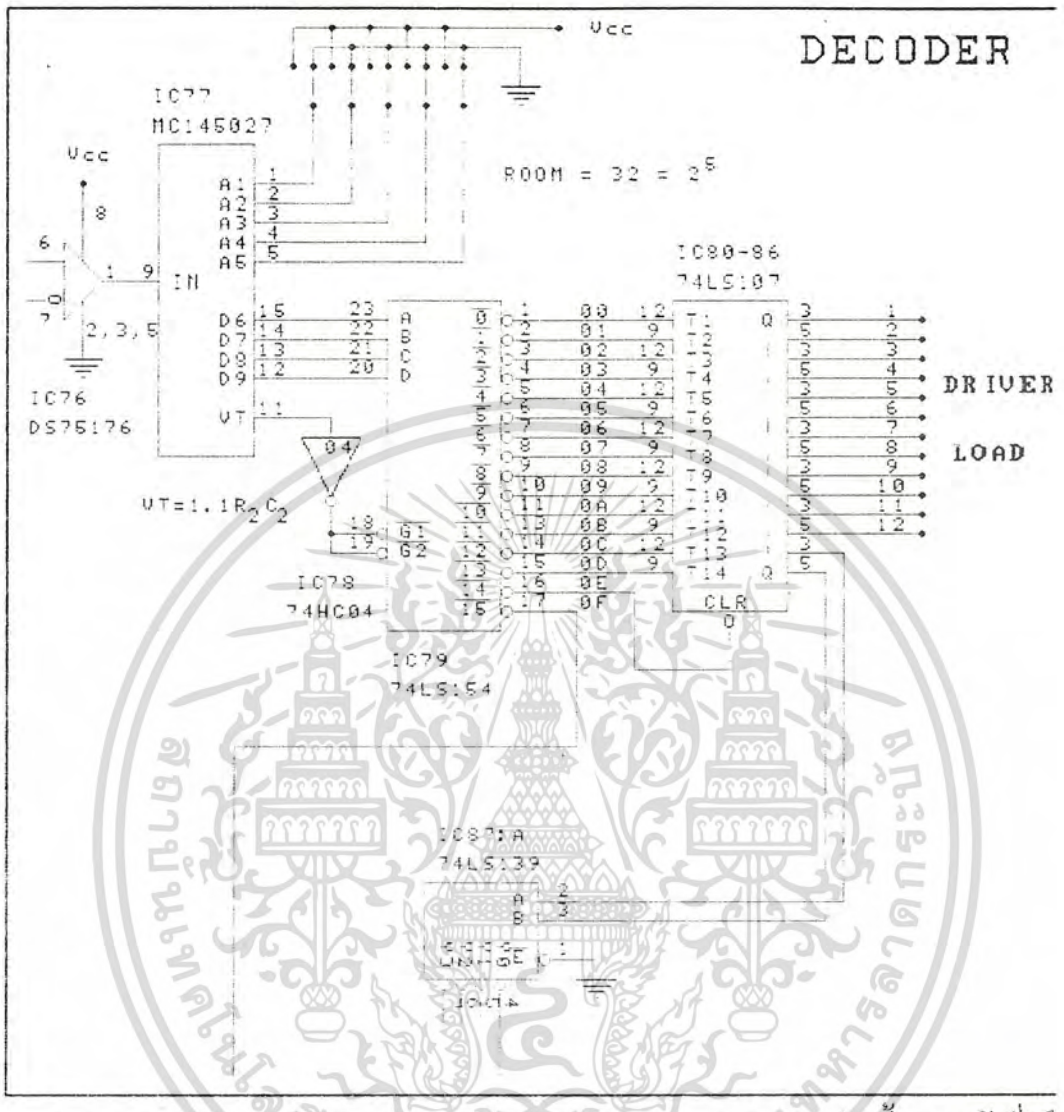
1.2 DECODER จะใช้ IC 74154, 74LS244x8, MC145027x16, DS75176X16 ซึ่งส่วนนี้จะนาข้อมูลจากทางด้าน I/O PORT ของ 8255 มาทาการถอดรหัสโดย 74154 ให้ได้เป็น

สัญญาณเลือกชั้นที่จะรับข้อมูลสถานะของอุปกรณ์ไฟฟ้าหรือ SWITCH ฉุกเฉินหรือ SENSOR ทั้งหมด 16 สัญญาณ ที่ต้องแยกวงจรกันนั้นเนื่องจากการส่งข้อมูลและการรับข้อมูลสถานะ จะกินเวลาช่วงเวลากันและใช้เวลาไม่เท่ากัน ดังนั้นจึงต้องแยกวงจรกัน จากการที่ระบบสามารถควบคุมได้ถึง 16 ชั้น ดังนั้นจึงต้องถอดรหัสสัญญาณออกมาทั้งหมด 16 สัญญาณ ซึ่งสามารถใช้แค่ข้อมูลจาก I/O Port เพียง 4 Bit ก็พอ (PB_0-PB_3) เพื่อเลือกชั้นที่ต้องการ IC 74154 ก็จะให้ O/P ของชั้นนั้นทำงาน ก็จะส่ง"0"มายังขา G1,G2 ของ 74LS244 เพื่อรับข้อมูล 4 Bitจาก MC145027 ผ่าน 74LS244 ตัวที่ทำงาน สู่ PC_0-PC_3 ของ 8255 ซึ่งแต่ละชั้น จะใช้รับข้อมูลสถานะห้องละ 4 ครั้งๆละ 4 Bit รวมเป็น 16 Bit หรือ 16 สัญญาณ (อุปกรณ์ไฟฟ้า 8 ตัว,อุปกรณ์ SWITCH ฉุกเฉิน 4 ตัว, อุปกรณ์ SENSOR 4 ตัว) ดังรูป

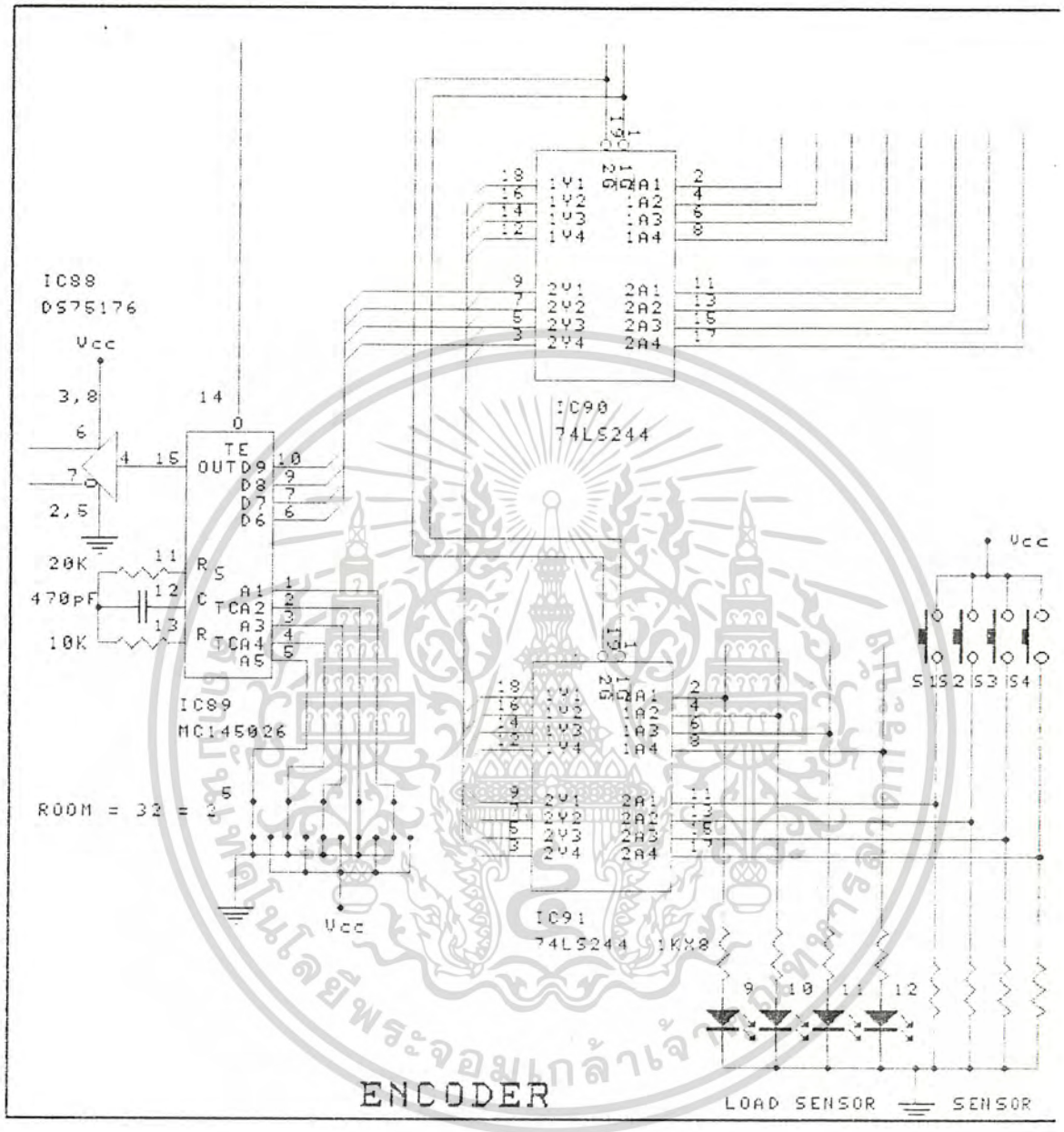


2. ส่วนควบคุมอุปกรณ์ปลายทาง ส่วนนี้จะรับข้อมูลอนุกรมแบบ Balance จากส่วนควบคุมแบบ Automatic มาแปลงเป็นข้อมูลอนุกรมแบบ Unbalance โดย DS75176 จากนั้นค่อยแปลงเป็นข้อมูลขนาน 4 Bit(D_8-D_9) เพื่อส่งไปควบคุมอุปกรณ์ไฟฟ้าและ SWITCH ฉุกเฉิน รวมทั้งหมด 12 ตัว และจะส่งข้อมูลสถานะของอุปกรณ์ไฟฟ้าและ SWITCH ฉุกเฉินและ SENSOR ทั้งหมด 16 ตัว โดยจะส่งไปครั้งละ 4 Bit 4 ครั้งไปยัง DECODER ของส่วนควบคุมแบบ Automatic เพื่อส่งไปยัง PC_0-PC_3 ของ 8255

2.1 DECODER จะใช้ DS75176, MC145027, 74LS154, 74LS107x7, 74LS139 ส่วนนี้จะรับข้อมูลแบบ Balance จาก ENCODER ของส่วนควบคุมแบบ Automatic มาแปลงกลับเป็นข้อมูลอนุกรมแบบ Unbalance ด้วย DS75176 จากนั้นค่อยแปลงเป็นข้อมูลขนาน 4 Bit (D_8-D_9) ด้วย MC145027 ซึ่งจะเห็นว่าขา A_1-A_5 ของ MC145027 จะต่อกับ DIP SW 5 อัน เพื่อใช้เลือกห้องนั่นเอง คือ ในแต่ละชั้นจะมีทั้งหมด 32 ห้อง ซึ่งทั้ง 32 ห้องจะต่อขนานกันอยู่ แต่จะปรับ DIP SW ทั้ง 5 ต่างกัน ซึ่งจะได้ 2^5 = 32 ห้อง ดังนั้น ถ้าทาง ENCODER ของส่วนควบคุมแบบ Automatic ส่งข้อมูล PA_0-PA_4 มาตรงกับ DIP SW ทั้ง 5 ของห้องใดห้องนั้นก็จะมีข้อมูลขนาน 4 Bit(D_8-D_9) ส่งออกมาสู่ IC 74LS154 เพื่อเลือก O/P Y_0-Y_{15} แต่ 74LS 154 จะทำงานได้ ก็จะต้องมีสัญญาณ VT เป็น "1" ผ่าน NOT GATE ก็จะให้ O/P เป็น "0" ทำให้ 74LS154 ทำงาน และถอดรหัสข้อมูล D_8-D_9 ว่าจะให้ Y ใดทำงาน ดังรูป ซึ่งจะเห็นว่า O/P Y ของ 74LS154 จะต่อกับขา CLK ของ 74LS107 ซึ่งเป็น DUAL JK FLIP-FLOP แต่จะให้ขา J กับ K ลอยไว้ จึงกลายเป็น T FLIP-FLOP นั่นเอง ดังนั้นเมื่อสภาวะปกติคือ 74LS 154 ยังไม่ทำงานขา O/P Y_0-Y_{15} จะเป็น "1" แต่ถ้า 74LS154 ทำงานขา O/P Y เพียงเส้นเดียวจะเป็น "0" ทำให้เป็นสัญญาณ CLK ให้กับ T FLIP-FLOP หรือเป็นสัญญาณ CLEAR ให้กับ T FLIP-FLOP หรือเป็นสัญญาณนาฬิกาให้กับ MC145026 ซึ่งการทำงานครั้งแรกจะเป็นการสั่งให้ CLEAR T FLIP-FLOP ก่อน โดยส่งข้อมูล E ให้กับ 74LS154 ซึ่งข้อมูล E จะทำให้ขา Y_{14} ของ 74LS154 เป็น"0" ทำให้ T FLIP-FLOP ทุกตัวถูก CLEAR เป็น "0" ต่อมาจึงค่อยส่งข้อมูล 0-B เพื่อทำให้ T FLIP-FLOP ตัวที่เลือกทำงานในสภาวะ Toggle ดังนั้น O/P Q ที่เลือกจะเป็น "1" คือ มีไฟมารอที่ SW ดังนั้นเพียงแต่ผู้ใช้สับ SW ก็จะทำให้อุปกรณ์ไฟฟ้าที่เลือกนั้นทำงานได้ ดังรูปหน้าต่อไป



2.2 ENCODER จะใช้ IC 74LS244x2, MC145026, DS75176 ส่วนนี้จะทำหน้าที่เลือกข้อมูลสถานะ 4 Bit ใดใน 16 Bit ส่งกลับมายัง DECODER ของส่วนควบคุมแบบ Automatic คือถ้าต้องการรับข้อมูลสถานะ 4 Bitใดเข้ามา ก็จะต้องส่งค่า C หรือ D มาเพื่อทำให้ Y_{12}หรือ Y_{13} ของ 74LS154 ทำงาน จึงทำให้ขา Q ของ T FLIP-FLOP เป็น "1" ส่งให้ 74LS139 Decoder ออกมาเพื่อเลือก O/P Y ทำงาน เพื่อส่งค่า "0" ให้กับ 74LS244 ที่ขา G1หรือ G2 ทำงาน ก็จะทำให้ ข้อมูลสถานะ 4 Bitที่เลือกนั้น ถูกส่งผ่าน 74LS244 ให้กับ D_6-D_9 ของ MC 145026 จากนั้นจึงส่งข้อมูล F มาเพื่อทำให้ Y_{15} ทำงาน จึงเป็นสัญญาณกริกให้กับ MC 145026 ซึ่งจะทำให้ MC145026 ทำงาน จึงรับข้อมูลสถานะ กับ A_1-A_5 (DIP SW 5 อัน)รวมทั้งหมด 9 Bit มา ENCODER เพื่อแปลงเป็นข้อมูลอนุกรมแบบ Unbalance จากนั้นจึงส่งข้อมูลอนุกรมนี้ให้กับ DS75176 ดังนั้นข้อมูลอนุกรมแบบ Unbalance ถูกแปลงเป็นข้อมูลอนุกรมแบบ Balance ส่งให้กับ DECODER ของส่วนควบคุมแบบ Automatic ดังรูป



การทำงานของแต่ละวงจร

การทำงานของแต่ละวงจรจะขอกล่าวถึงเป็นส่วน ๆ แบบเดียวกับส่วนฮาร์ดแวร์ดังนี้

1. **ส่วน Computer** จะต้องเข้าสู่ PROGRAM ของระบบก่อน โดยเข้าสู่ File PCOM1.EXE หรือ PCOM2.EXE ซึ่งขึ้นอยู่กับการต่อ RS-232 เข้ากับ Port COM1 หรือ COM2 ซึ่งเมื่อเข้าสู่โปรแกรมแล้ว จะต้องรอเวลาระยะหนึ่ง เพื่อทำการ LOAD DATA จากส่วน Board Control Z80180 สู่ส่วน Computer จากนั้นก็จะแสดง VERSION ของโปรแกรม แล้วหยุดรอรับ KEY ใดๆ เมื่อกด KEY ใดๆ แล้วก็จะแสดง MENU หลักของโปรแกรม โดยมีดังนี้คือ FILE, EDIT, DISPLAY, SETUP, VERSION, QUIT ดังรูปหน้าต่อไป โดยจะมี BAR อยู่ที่ FILE ถ้าต้องการเลือก MENU ใดๆ ก็เลื่อน BAR ไปในตำแหน่งต้องการเลือก

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

ส่วนของ UPDATE DATA เป็นส่วนที่ต้องนำ DATA จากส่วน Board Control Z80180 สู่ส่วน Computer ซึ่งต้องมีการรอเวลาชั่วระยะเวลาหนึ่ง เพื่อนำข้อมูลมาประมวลผลในส่วนของการแสดงผล คือส่วนของ 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>เพื่อเป็นการบอกว่าต้องการจะออกจากส่วนของSAVE FILE ไปในส่วนของ MENU FILE

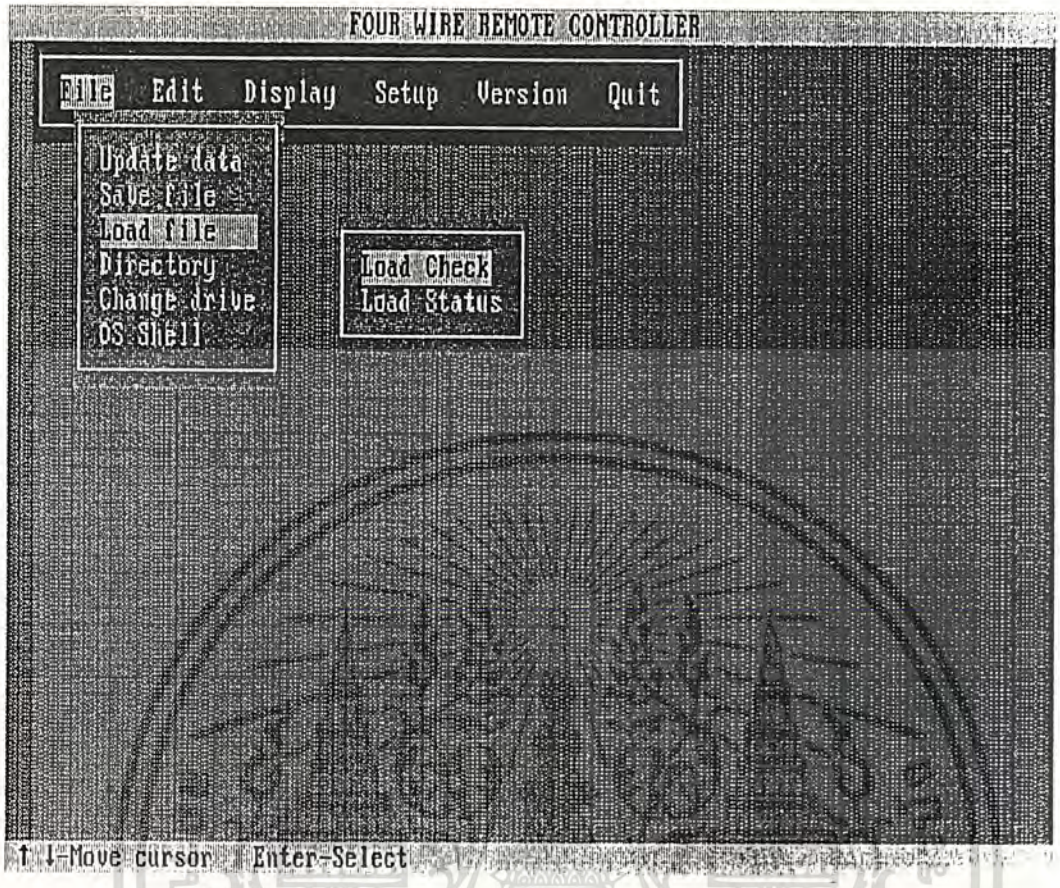
ส่วนของ LOAD FILE นั้นจะมี MENU อีก 2 อย่าง คือ LOAD CHECK, LOAD STATUS ในส่วนของ LOAD CHECK นั้นจะเป็นส่วนนำสถานะของการ CHECK IN หรือ CHECK OUT จาก FILE ชื่อ CHECK.CFG ออกมาให้ตัวแปร ถ้าไม่สามารถนำข้อมูลออกจาก FILE ชื่อ CHECK.CFG มาให้ตัวแปรได้ จะมีข้อความแสดงไม่สามารถนำข้อมูลออกมาจาก FILE ชื่อ CHECK.CFG มาให้ตัวแปรและในส่วนของ 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

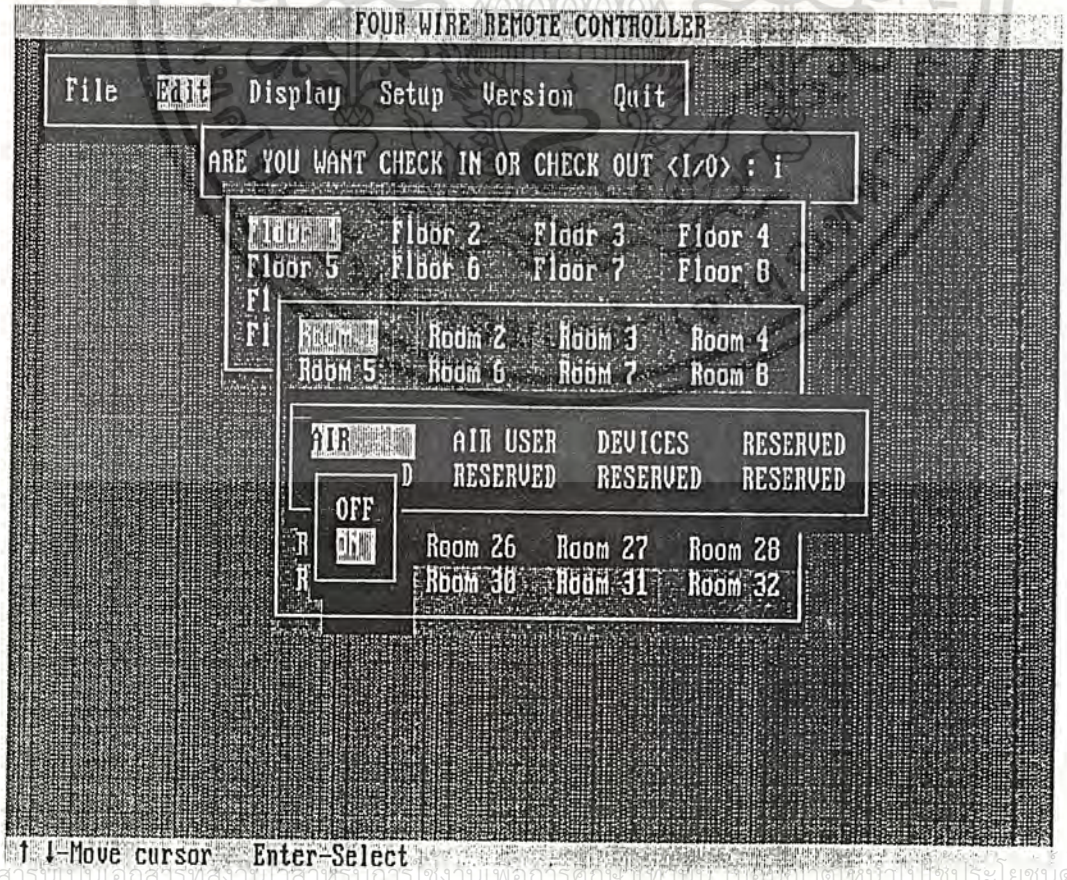
ส่วนของ DIRECTORY นั้นเป็นส่วนแสดงรายชื่อ FILE ทั้งหมดที่มีอยู่ใน DISKโดยสามารถดูรายชื่อ FILE ใน DISK ใหนก็ได้ ถ้าไม่สามารถดูรายชื่อได้จะมีข้อความแสดงออกมาว่าไม่สามารถแสดงรายชื่อได้

ส่วน CHANGE DRIVE นั้น เป็นส่วนของการการเปลี่ยน DRIVE โดยสามารถเลือกได้สูงสุด 5 DRIVE แต่จะมีตัว CHECK ว่าในคอมพิวเตอร์มี DRIVE สูงสุดกี่ DRIVE ก็แสดงแต่ละ DRIVE ของคอมพิวเตอร์ โดยสามารถเปลี่ยน DRIVE โดยการเลื่อน KEY ลูกศรไปทางซ้ายหรือทางขวาได้ โดยให้ BAR นั้นไปอยู่ตำแหน่งของ DRIVE ที่ต้องการแล้วกด KEY <ENTER> แล้ว ก็จะเป็นการเปลี่ยน DRIVE ที่ต้องการแล้ว โดยสามารถดูชื่อ FILE ตาม DRIVE ที่เราต้องการ

ส่วน OS SHELL จะเป็นส่วนที่ออกจากโปรแกรมชั่วคราวเพื่อไประบบของDOS 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 SWITCH ได้ในแต่ละ SWITCH โดยต้องมีการเลือกชั้นก่อนแล้วเลือกห้องและหลังจากนั้นก็จะเป็นการเลือก SWITCH ที่จะเปิด,ปิด ถ้าเป็นการ CHECK OUT เป็นการปิด SWITCH ในแต่ละห้องโดยต้องเลือกชั้นแล้วเลือกห้อง ก็จะเป็นการปิด SWITCH ทุกตัวของชั้นนั้นในห้องนั้น

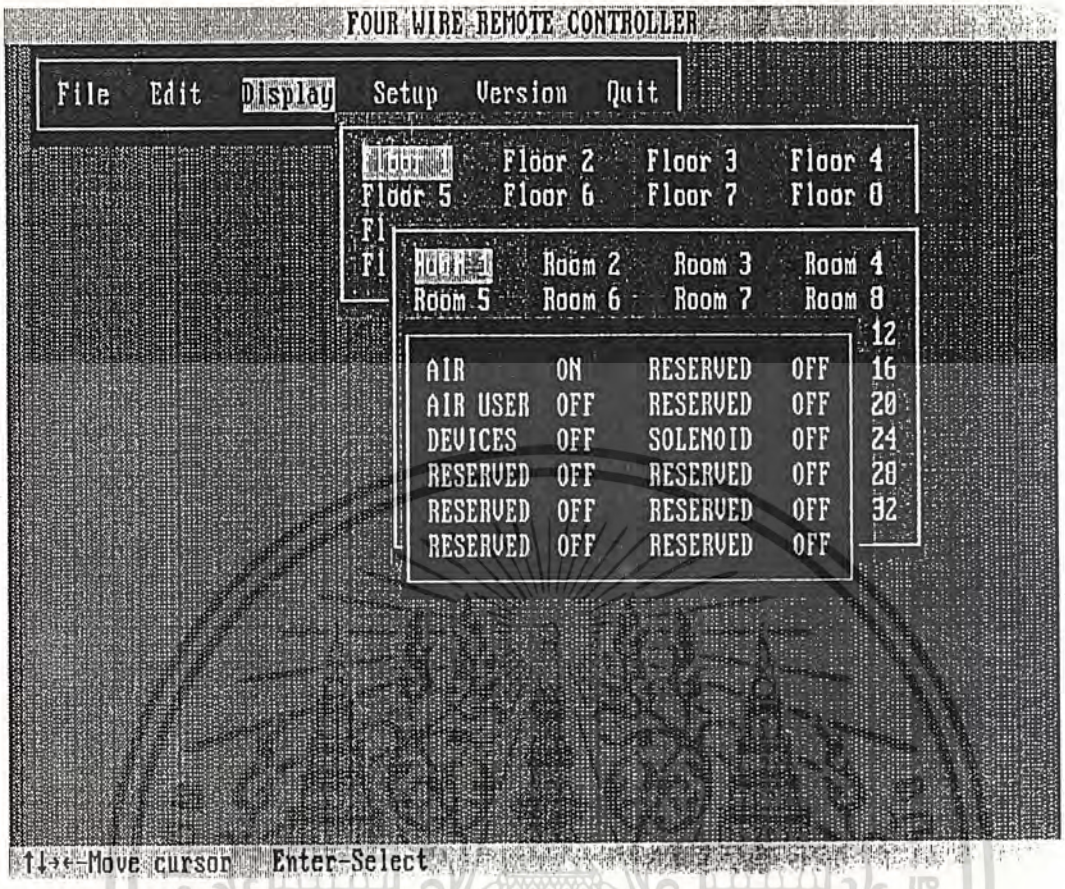


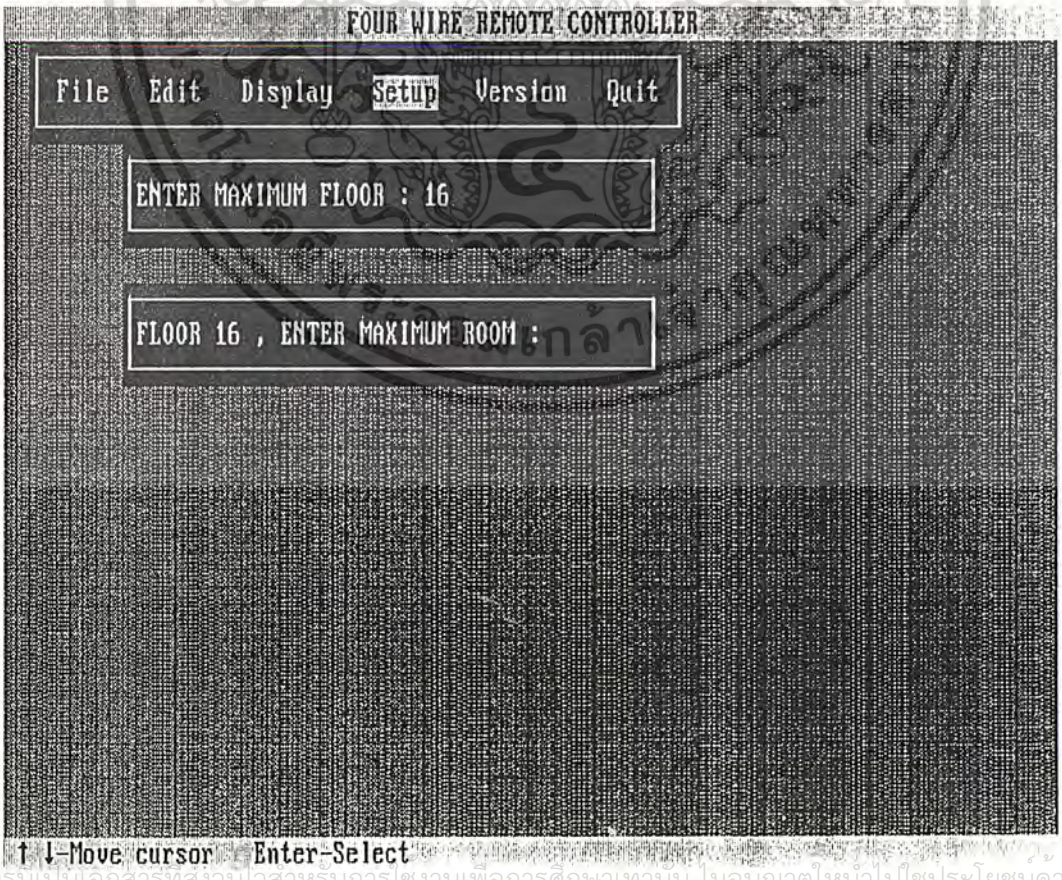


ส่วน DISPLAY นั้นจะเป็นส่วนแสดงว่ารายละเอียดของ SWITCHในแต่ละห้องของชั้นนั้นว่า มี SWITCH ตัวไหนมีการเปิด, ปิด อยู่หรือเปล่าและยังแสดงว่ามีตัว SENSOR ว่าตอนนี้ทำงานอยู่หรือเปล่าด้วย ดังรูปหน้าต่อไป

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

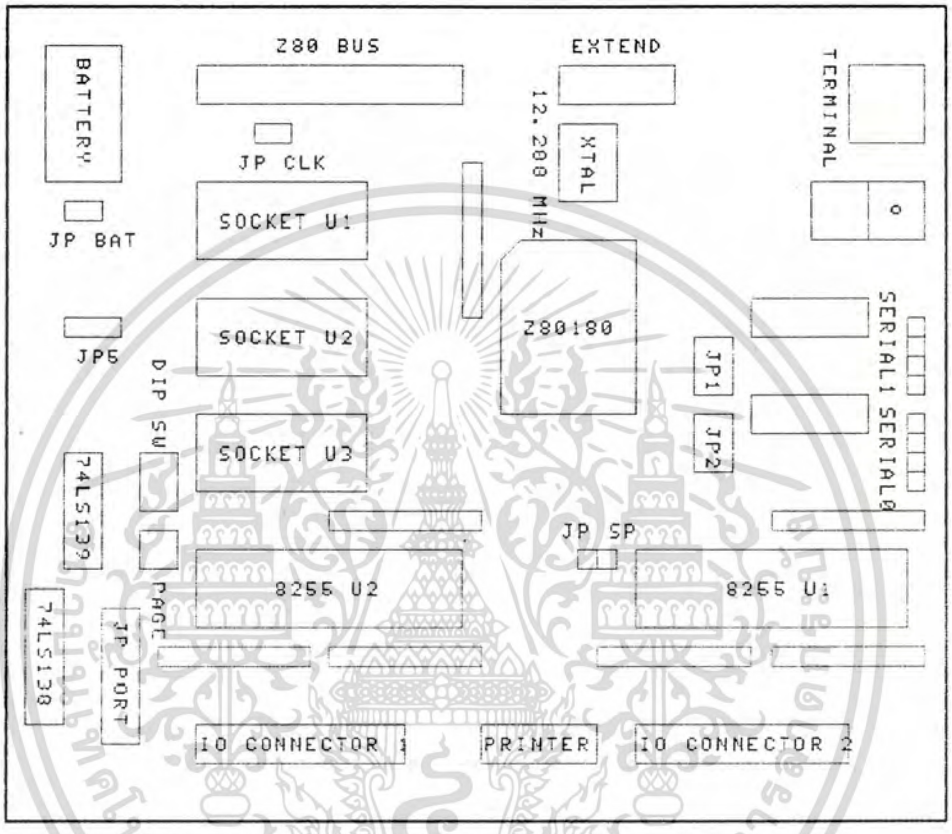
ส่วนของ QUIT นั้นจะเป็นการออกจากโปรแกรม โดยถ้าจะเข้ามายังโปรแกรมใหม่นั้นจะต้องมีการ LOAD FILE ชื่อ PCOM1.EXE หรือ PCOM2.EXE ใหม่





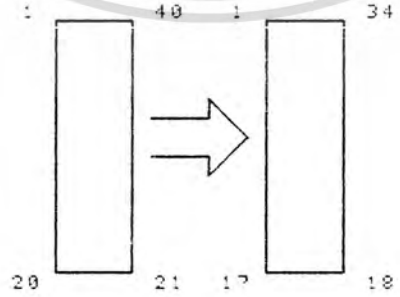
2. ส่วน Board Control Z80180 จะต้อง SET ค่าตามนี้

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



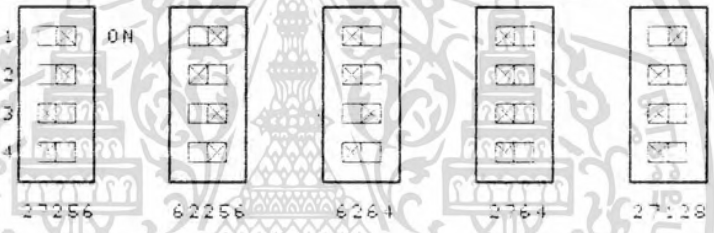
2. LED ของ 8255 จะสลับกับความเป็นจริงคือ ที่ SCREEN PC71 เป็น PC72 และ PC72 เป็น PC71

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:-

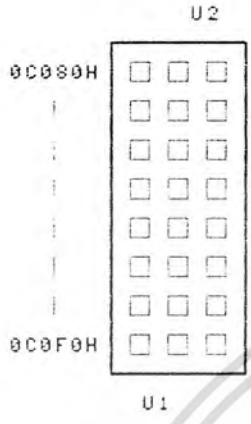
8255 ทั้ง 2 ตัวบน BOARD จะอ้างที่ 0C0XXH

8255 ทั้ง 2 ตัวบน BOARD จะอ้างที่ XCH - XFH

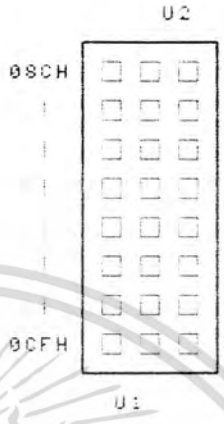
ซึ่งค่า X นั้นยังขึ้นอยู่กับการเลือก JUMPER PORT อีกครั้งหนึ่ง

JP PORT (JP 4) ใช้เลือก NUMBER PORT โดยสามารถเลือกได้ดังรูป

เมื่อเลือก PAGE เป็น OCOXXH

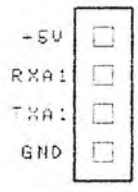


เมื่อเลือก PAGE เป็น XCH - XFH



ซึ่ง BOARD สำเร็จที่ค่าขึ้น จะถูกกำหนดเป็น OC080H ของ 8255 (U1) และ OC090H ของ 8255 (2) ซึ่งเป็นของ PRINTER PORT ด้วย ซึ่งผู้ใช้สามารถเปลี่ยนแปลงได้ แต่เมื่อใช้กับ SOFTWARE DEBUG ต้องกำหนดตามนี้

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



SERIAL 0 เป็น CONNECTOR SERIAL PORT ของ ASCI CHANEL 0 และยังเลือกได้ว่าให้เป็นขั้วต่อของ CLOCK SERIAL I/O ได้โดยการ 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 ใช้เองในกรณีสัญญาณครบปกติบน BOARD จะถูก JUMP ไว้ให้

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

JP CLK เป็น JUMPER เลือกCLOCK 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 สำหรับINTERFACE กับอุปกรณ์ภายนอก

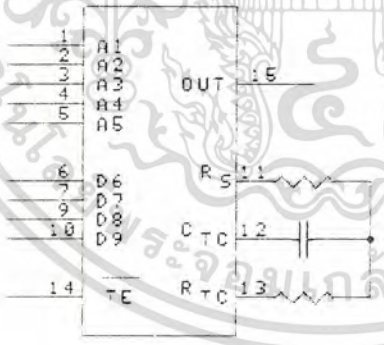
Specification

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 Kbyte)
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 Select 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

3. ส่วนควบคุมแบบ Automatic และ 4. ส่วนควบคุมอุปกรณ์ปลายทาง จะขออธิบายตามการทำงานของระบบ ซึ่งอาจจะไม่เรียงกันดังนี้

3.1 ENCODER เมื่อส่วน Board Control Z80180 ส่งค่า Port PA,PB,PC ตามโปรแกรม Control1.asm ที่เก็บอยู่ใน EPROM ก็จะเกิดการทำงานของแต่ละส่วนดังนี้คือ PB_0-PB_7 จะมีค่า 40H-4FH จะถูกถอดรหัสด้วย 74LS138 ทำให้ O/P Y_1 ทำงาน คือเป็น"0" ซึ่งจะทำให้ 74151 ทำงานและเลือกชั้นตามค่า 0-F ทำให้ O/P Y_0-Y_{15} เพียง 1 เส้นเท่านั้นที่ทำงานคือเป็น "0" ทำให้มีการทริกให้ MC145026 ทำงาน จึงทำให้ข้อมูล 9 Bit(A_1-A_5,D_6-D_9)ถูกแปลงเป็นข้อมูลอนุกรมส่งออกที่ขา DATA OUT ไป 2 ครั้งติดกันสู่ DS75176 ซึ่งจะแปลงข้อมูลอนุกรมที่ส่งมา(Unbalance) ไปเป็นข้อมูลอนุกรมแบบ Balance ส่งให้สู่ส่วนต่อไป คือส่วน DECODER ของส่วนควบคุมอุปกรณ์ปลายทาง

ดังรูปวงจร ENCODER ใช้ MC145026 ในการทำงานเข้ารหัสข้อมูลส่งออกไป จะเห็นว่ามีอุปกรณ์ R,C ต่อร่วมเพียง 3 ตัว ซึ่งทั้ง 3 ตัวนี้จะเป็นตัวกำหนดความถี่ออสซิลเลเตอร์ ในวงจรนี้จะใช้ความถี่ 88.7 KHz ซึ่งคำนวณโดยสูตร



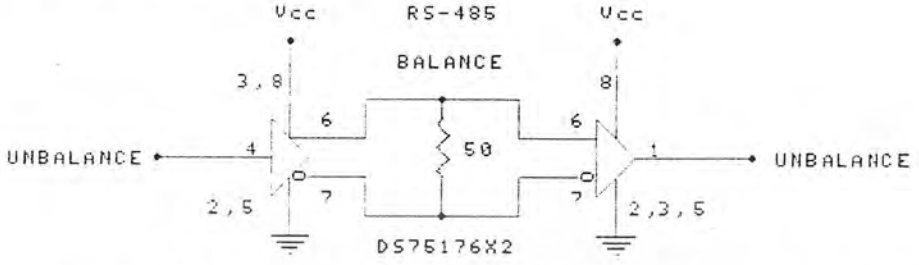
$$f_{osc} = 1 / 2.3\ R_{TC}C_{TC}'$$

$$C_{TC}' = C_{TC} + 20\ PF$$

$$100\ pF \leq C_{TC} \leq 15\ \mu F$$

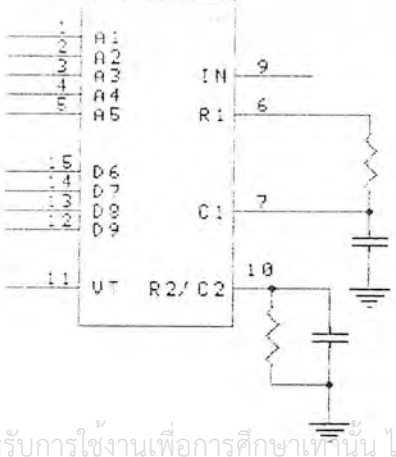
$$R_{TC} \geq 10K\ ;\ R_S = 2R_{TC}$$

ส่วน DS75176 RS-485/RS-422 Transceiver จะแสดงดังรูป ซึ่งจะต่อกับ DS75176 RS-485/RS-422 Transceiver



4.1 DECODER ก็จะรับข้อมูลอนุกรมแบบ Balance มาแปลงกลับด้วย DS75176ไปเป็นข้อมูลอนุกรมแบบ Unbalance แล้วส่งเข้า DATA IN ของ MC145027 ทั้ง 32 ห้องใน 1 ชั้น ซึ่งจะต่อกันแบบขนาน แต่จะมีเพียงห้องเดียวที่ปรับ DIP SW ตรงกับทางด้าน ENCODER ก็จะทำให้ข้อมูลอนุกรมที่ส่งมาติดๆกัน match address กัน และข้อมูล 4 Bit ทั้ง 2 ครั้งจะต้อง match กันด้วย ดังนั้นเมื่อเป็นตามเงื่อนไขแล้วก็จะทำให้ขา VT เป็น "1" เพียง 0.429 msec ซึ่งคำนวณได้จาก สูตร 1.1 R_2C_2 เมื่อ VT เป็น "1" ก็จะทำให้มีข้อมูล D_8-D_9 ส่งมาให้ส่วน 74LS154 ก็จะทำให้ 74LS154 ทำงานและถอดรหัสข้อมูลว่าเป็นข้อมูลใด ก็จะทำให้ O/P Y นั้นทำงาน ซึ่งถ้าเป็น 0 ก็จะทำให้ Y_0 ทำงาน เป็นต้น เมื่อ O/P Y อยู่ระหว่าง Y_0-Y_{11} ก็จะทำให้ T FLIP-FLOP นั้นทำงาน ทำให้ O/P Q เปลี่ยนสถานะจาก"0"เป็น"1" ก็จะทำให้อุปกรณ์ไฟฟ้านั้นทำงานเมื่อสับ SW ปิด ส่วนถ้าเป็น Y_{12} หรือ Y_{13} ก็จะทำให้ Q เป็น"1" แต่ 2 O/P นี้จะถูกส่งไปถอดรหัสอีกครั้งด้วย 74LS139 เพื่อใช้ควบคุมการรับข้อมูลสถานะ 4 Bit แต่ถ้าเป็น Y_{14} ก็จะทำให้ T FLIP-FLOP ถูก CLEAR ทุกตัว เพราะขา CLEAR นี้จะต่อร่วมกัน แต่ถ้าเป็น Y_{15} ก็จะเป็นการทริกให้กับ MC145026

ดังรูปวงจร DECODER ใช้ MC145027 ในการถอดรหัสข้อมูลที่ส่งมา จะเห็นว่ามี R_1,C_1, R_2,C_2 ซึ่งคำนวณได้จากสูตร



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

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

$R_1 >= 10K$; $C_1 >= 400pF$; $R_2 >= 100K$; $C_2 >= 700pF$

ซึ่งวงจรส่วนนี้จะต้องมีทั้งหมด 32 ห้อง หรือ 32 ชุดต่อ 1 ชั้นเท่านั้น

4.2 ENCODER จะทำหน้าที่รับข้อมูลสถานะ 4 Bit ที่ถูกถอดรหัสโดย 74LS139 ผ่าน 74 LS244 สู่ D_6-D_9 ของ MC145026 ซึ่ง MC145026 ก็จะแปลงข้อมูลขนานเป็นข้อมูลอนุกรมดังที่กล่าวมา แต่ต้องมีสัญญาณ TE ก่อนจึงจะเข้ารหัสที่จะส่งได้ เมื่อมีสัญญาณ TE แล้วก็จะส่งข้อมูลอนุกรมไปให้ส่วนต่อไปได้ ซึ่งส่วนนี้จะมีทั้งหมด 32 ชุดเหมือนกัน

3.2 DECODER จะทำหน้าที่รับข้อมูลสถานะจากส่วน ENCODER เพื่อแปลงเป็นข้อมูลอนุกรมแบบ Unbalance ด้วย DS75176 และส่งเข้า DATA IN ของ MC145027 และถ้า match กันก็จะทำให้ได้ข้อมูล D_6-D_9 ออกมาเข้าสู่ 74LS244 ต่อที่ 74154 เลือก ก็จะทำให้มีข้อมูลส่งให้ PC_0-PC_3 เพื่อส่งให้ส่วน Board Control Z80180 นำไปประมวลผลต่อ

5. ส่วน LOAD และ SENSOR และ LOAD SENSOR ก็จะประกอบด้วยอุปกรณ์ไฟฟ้า 9 ตัว เช่น พัดลม , แอร์ , ตู้เย็น , TV , หลอดไฟ เป็นต้น ส่วน LOAD SENSOR 4 ตัว เช่น ไฟฉุกเฉิน , โซลีนอย , เครื่องดับเพลิง เป็นต้น ส่วน SENSOR เช่น SENSOR ควัน SENSOR แอร์ , CARD MAGNETIC เป็นต้น ซึ่งอุปกรณ์ที่กล่าวมา ถ้าทำงานก็ให้ "1" และถ้าไม่ทำงานก็ให้ "0" เป็นต้น

การออกแบบขั้นสุดท้าย

การออกแบบขั้นสุดท้ายหลังจากที่เราได้ทำการทดลองวงจรมาเรียบร้อยแล้ว การออกแบบขั้นสุดท้ายก็คือการออกแบบแผ่นวงจรพิมพ์ ซึ่งในที่นี้เราใช้ PROTEL ช่วยในการออกแบบ ซึ่ง Project นี้จะแยกเป็น 3 ส่วนคือ

1. ส่วนควบคุมแบบ Automatic
2. ส่วนควบคุมอุปกรณ์ปลายทาง
3. ส่วนขับ LOAD และ SENSOR

ส่วนควบคุมแบบ Automatic จะแบ่งเป็น 2 แผ่น คือ 1.ส่วน ENCODE 2.ส่วน DECODE ซึ่งทั้ง 2 แผ่น ออกแบบเป็นแผ่นวงจรแบบสองหน้าทรูโฮล

ส่วนควบคุมอุปกรณ์ปลายทาง ออกแบบเป็นแผ่นวงจรแบบสองหน้าทรูโฮล

ส่วนขับ LOAD และ SENSOR ออกแบบเป็นแผ่นวงจรแบบหน้าเดียว

เมื่อออกแบบวงจรทั้ง 3 ส่วนเสร็จแล้ว จากนั้นเราก็นำเอาวงจรที่ออกแบบโดย PROTEL มา PLOT ลงกระดาษ แล้วนำไปถ่ายฟิล์ม negative เมื่อได้ฟิล์มมาแล้วก็นำไปทำตามขั้นตอนของดรายฟิล์ม และอีกส่วนหนึ่งนำไปทำทรูโฮล

สรุปและวิจารณ์

ปัญหาที่เกิดขึ้น

-Supply ที่ใช้จ่ายกระแสไม่เพียงพอ

-ส่วน DECODER ไม่ทำงาน

-โปรแกรมที่ทำงานกับส่วนประมวลผลทำงานเร็วเกินกว่าที่ส่วนส่งและรับข้อมูลจะทำงานทัน

-การสแกนอุปกรณ์และ SENSOR ทั้ง 512 ห้อง ใช้เวลา 35 วินาที

-การใช้ RS-422 ทำให้ต่อขนานกับตัวรับได้ไม่เพียงพอ

การแก้ไข

-ใช้ Supply ที่จ่ายกระแสสูงขึ้น

-เนื่องจาก MC145027 ไม่ match กับ MC145026 ดังนั้นจึงต่อ Resistor ลงกราวด์

-ทำการเขียนโปรแกรมให้มีการหน่วงเวลาไว้ด้วย

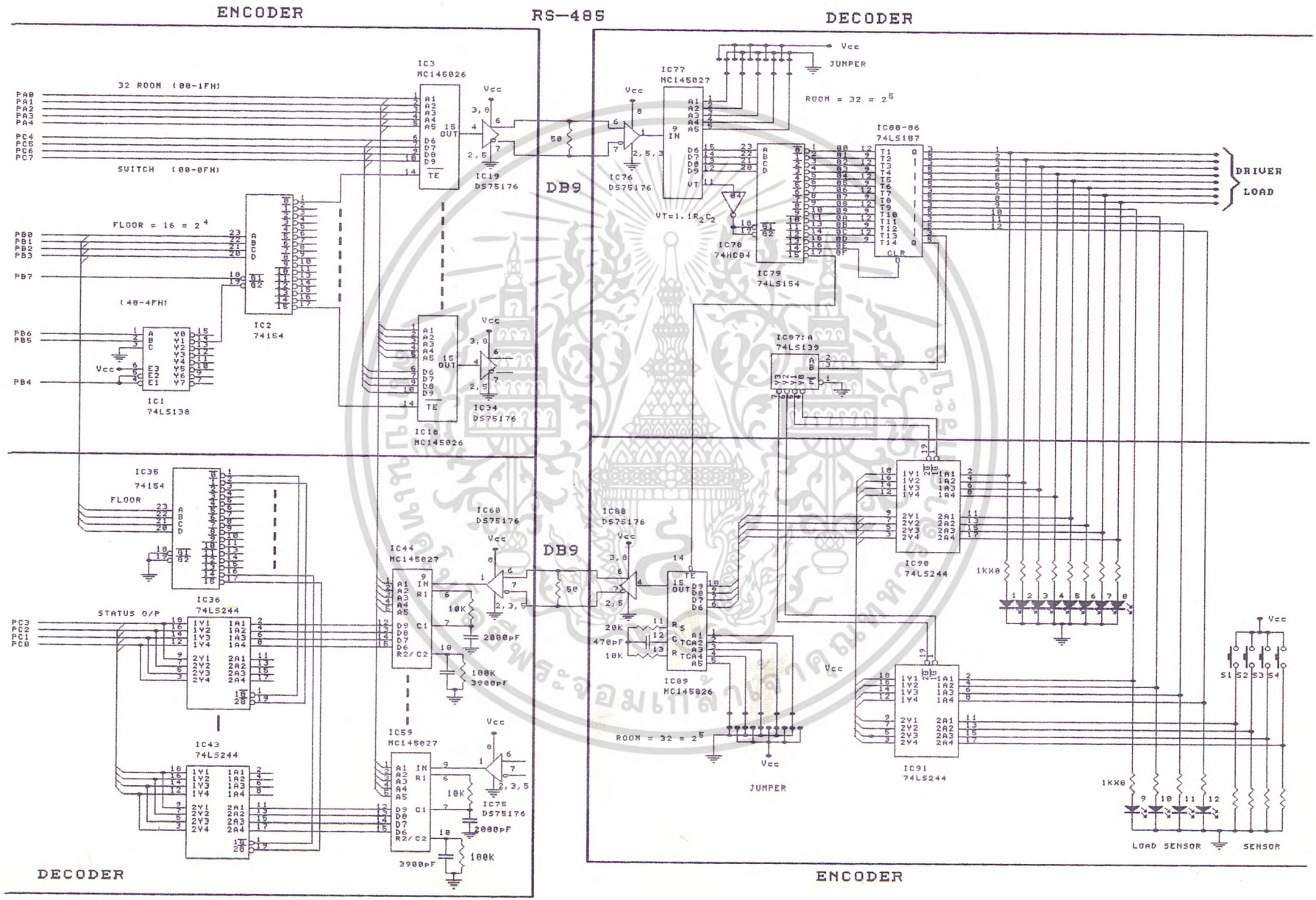
-ทำการแก้โปรแกรมให้เช็คแต่ห้องที่มีคนอยู่

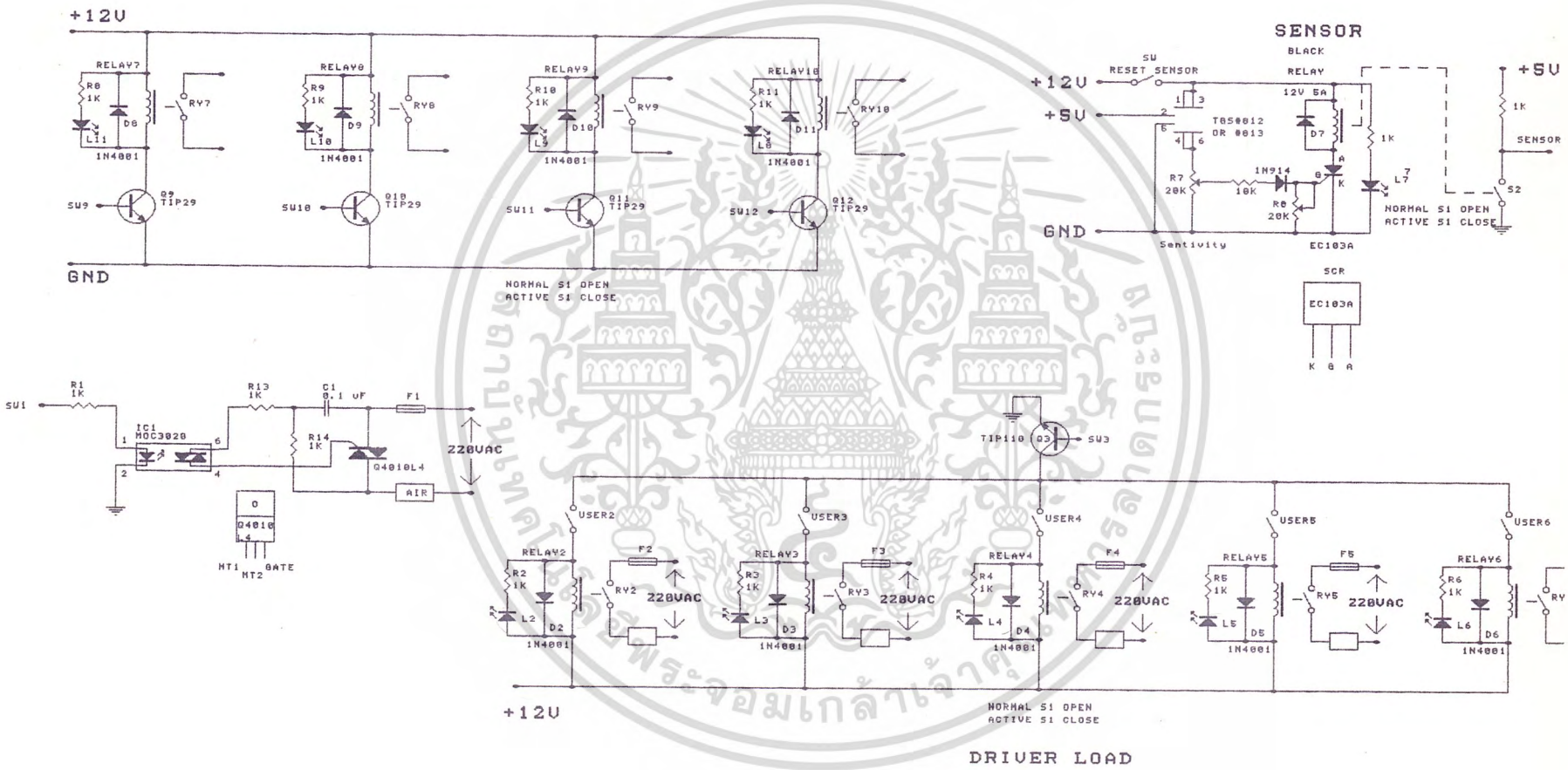
-ทำการเปลี่ยน RS-422 เป็น RS-485

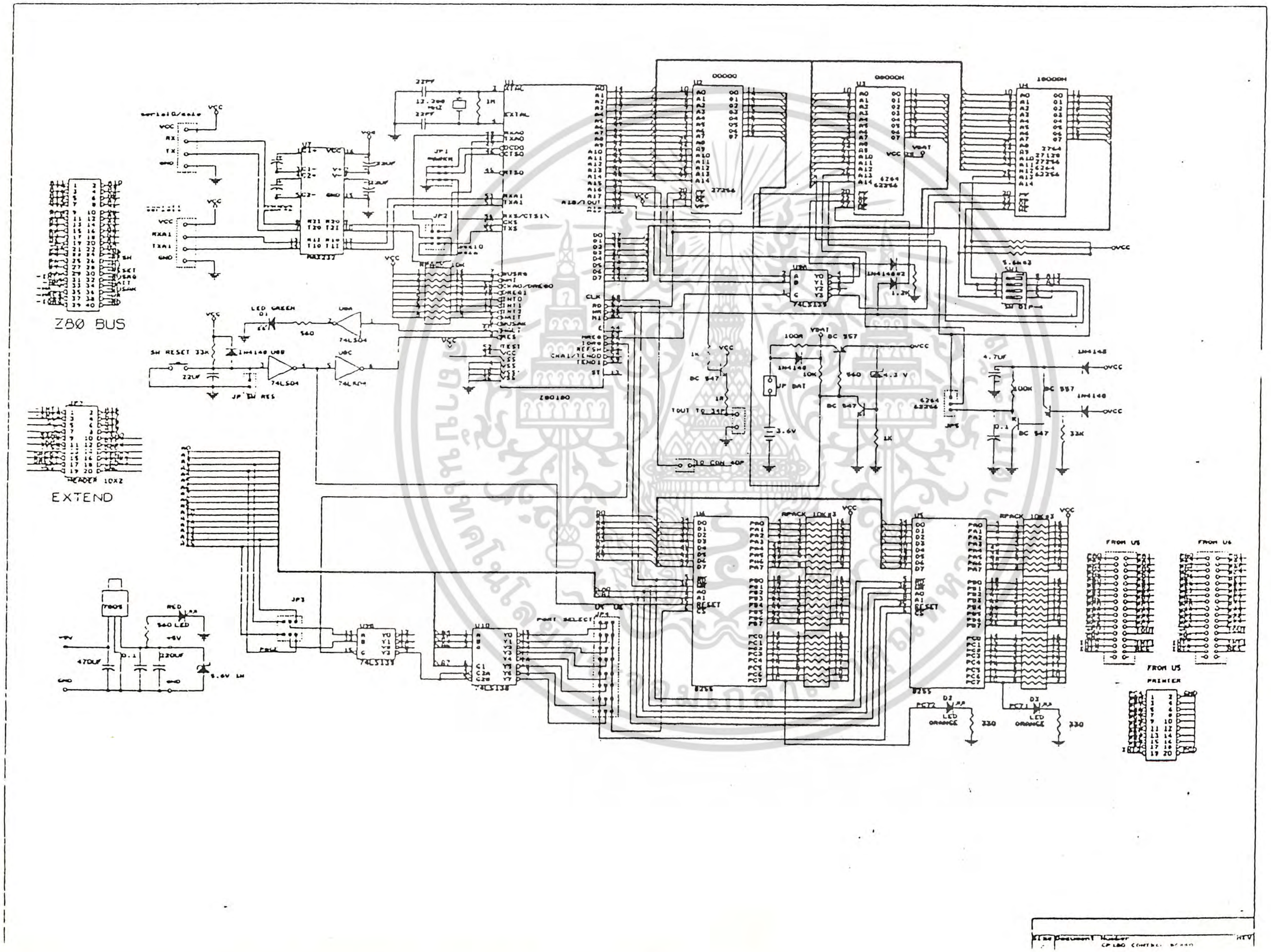
สรุปผลการปฏิบัติงาน

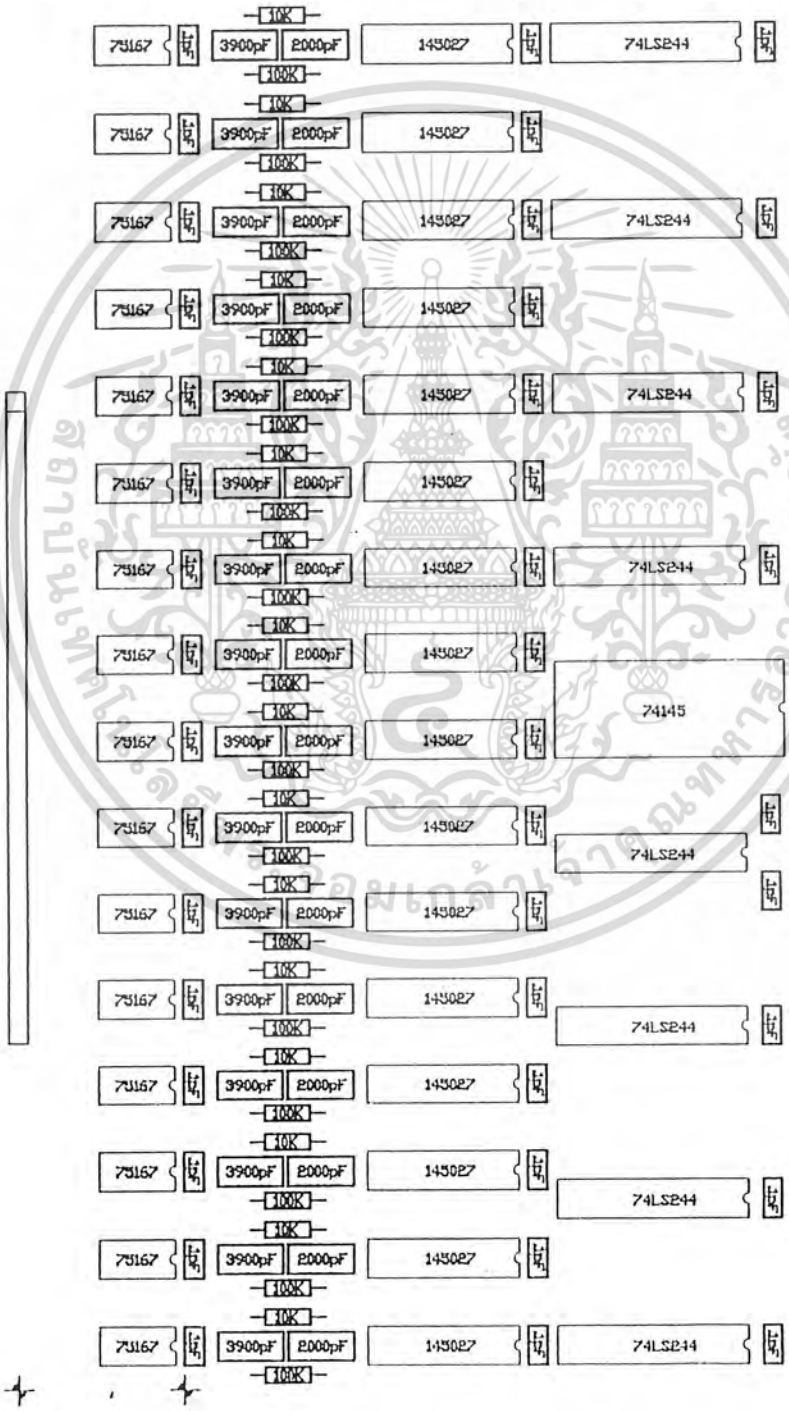
จากการที่ได้ทดลองให้ระบบได้ทำงานเต็มที่พบว่าการทำงานได้ตามจุดประสงค์ที่ต้องการ อาจจะมีปัญหาบ้างทางด้านฮาร์ดแวร์ หรือซอฟต์แวร์แต่ก็ได้ทำการแก้ไขปัญหาไปเรียบร้อยแล้ว ทำให้ระบบสามารถใช้ควบคุมอุปกรณ์ไฟฟ้าตามห้องพักได้ตามวัตถุประสงค์ นอกจากนี้ทางด้านฮาร์ดแวร์ยังได้ออกแบบให้สามารถเขียนโปรแกรมควบคุมเพิ่มเติม หรือเปลี่ยนแปลงให้เข้ากับงานที่จะประยุกต์ใช้ในอนาคตต่อไป

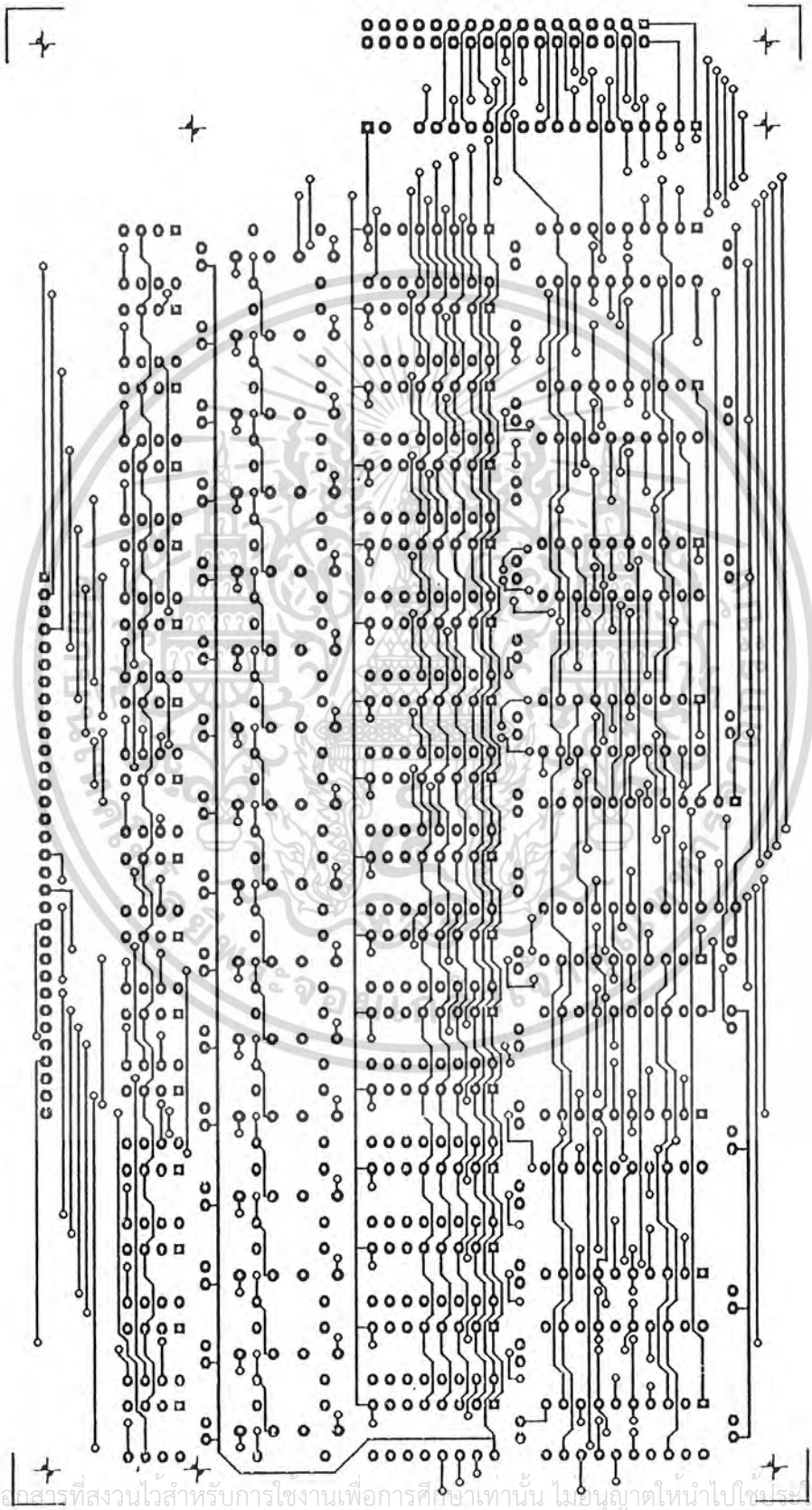
ภาคผนวก

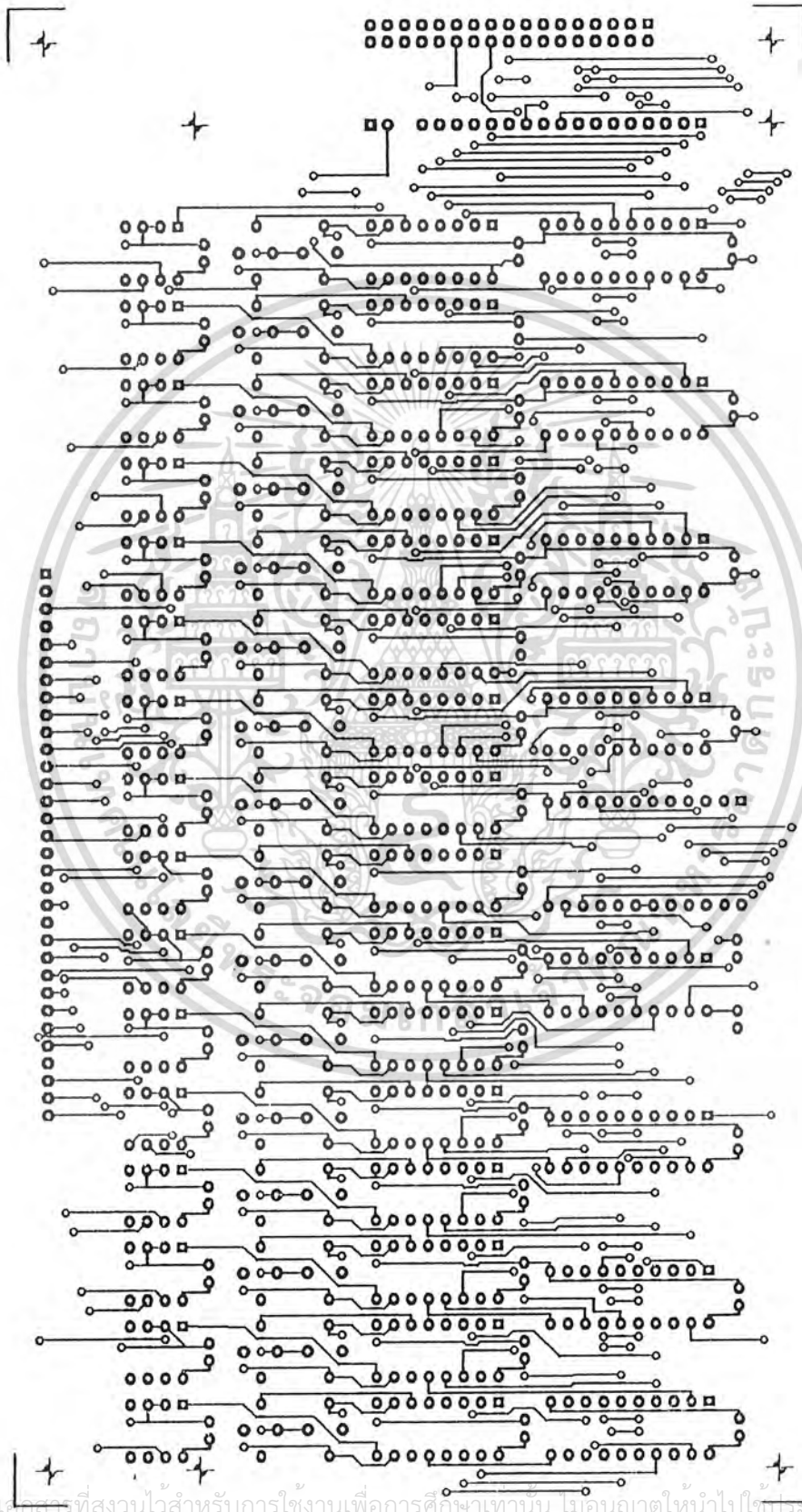

ENCODER
RS-485
DECODER
32 ROOM (00-1FH)
SWITCH (00-0FH)
FLOOR = 16 = 2^4
(40-4FH)
IC3
MC145026
IC19
DS75176
IC2
74154
IC1
74LS138
IC34
DS75176
IC18
MC145026
DB9
IC77
MC145027
IC76
DS75176
ROOM = 32 = 2^5
JUMPER
IC80-86
74LS107
IC78
74HC04
IC79
74LS154
IC87:A
74LS139
DRIVER
LOAD
IC90
74LS244
1KX8
IC91
74LS244
IC88
DS75176
IC89
MC145026
IC36
74154
FLOOR
IC36
74LS244
STATUS O/P
IC44
MC145027
IC60
DS75176
IC43
74LS244
IC59
MC145027
IC75
DS75176
10K
2000pF
100K
3900pF
20K
470pF
S1 S2 S3 S4
LOAD SENSOR
SENSOR
DECODER
ENCODER

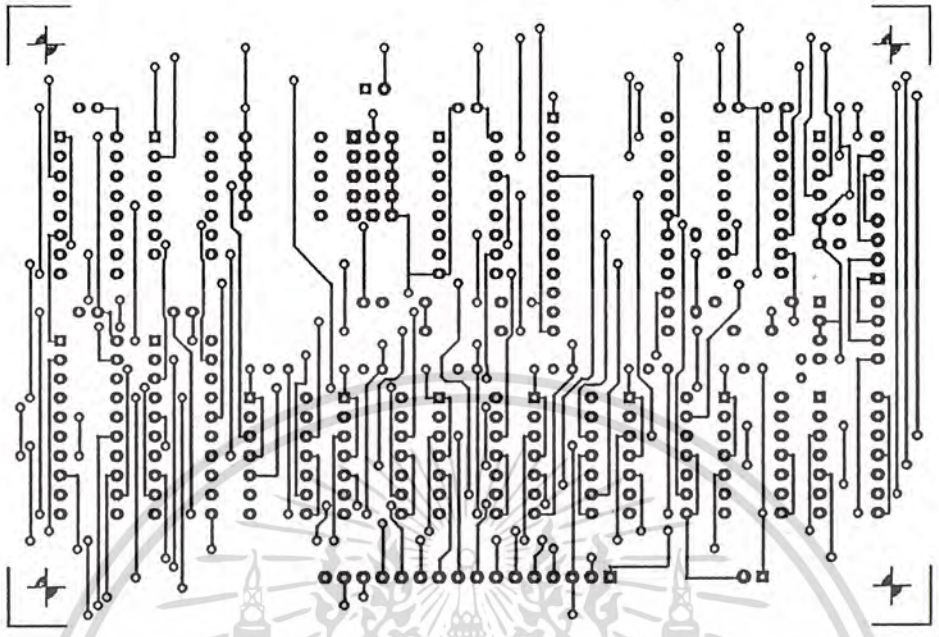

+12V
RELAY7
RELAY8
RELAY9
RELAY10
R8 1K
R9 1K
R10 1K
R11 1K
D8
D9
D10
D11
1N4001
RY7
RY8
RY9
RY10
SW9
SW10
SW11
SW12
Q9 TIP29
Q10 TIP29
Q11 TIP29
Q12 TIP29
GND
NORMAL S1 OPEN
ACTIVE S1 CLOSE
SENSOR
BLACK
SW
RESET SENSOR
RELAY
12V 5A
+12V
+5V
T85#012
OR #013
D7
1K
+5V
SENSOR
R7 20K
10K
1N914
R8 20K
L7
S2
NORMAL S1 OPEN
ACTIVE S1 CLOSE
GND
Sentivity
EC103A
SCR
EC103A
K G A
SW1
R1 1K
IC1 MOC3020
R13 1K
C1 0.1 uF
F1
R14 1K
Q4010L4
220VAC
AIR
Q4010
MT1 MT2 GATE
TIP110 Q3
SW3
USER2
USER3
USER4
USER5
USER6
RELAY2
RELAY3
RELAY4
RELAY5
RELAY6
F2
F3
F4
F5
R2 1K
R3 1K
R4 1K
R5 1K
R6 1K
L2
L3
L4
L5
L6
D2
D3
D4
D5
D6
RY2
RY3
RY4
RY5
RY
220VAC
220VAC
220VAC
220VAC
+12V
NORMAL S1 OPEN
ACTIVE S1 CLOSE
DRIVER LOAD


Z80 BUS
EXTEND
HEADER 10X2
Z80180
MAX232
27256
6264
62256
74LS139
74LS138
74LS04
8255
RPACK 10Kx8
PORT SELECT
FROM U5
FROM U6
PRINTER
LED GREEN
SW RESET
7805
VCC
VBAT
3.6V
SW DIP-4
Size
Document Number
REV
CP180 CONTROL BOARD

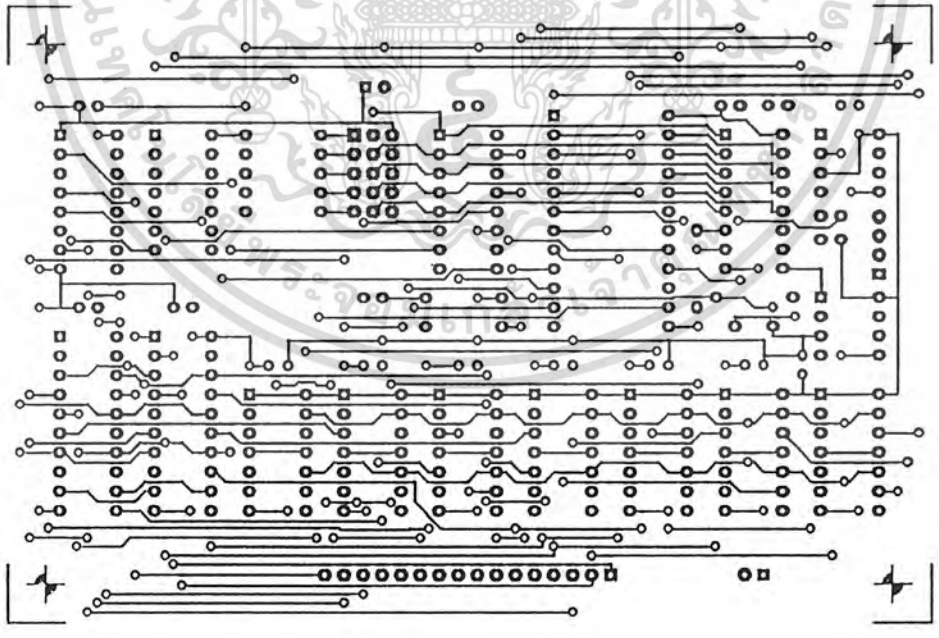




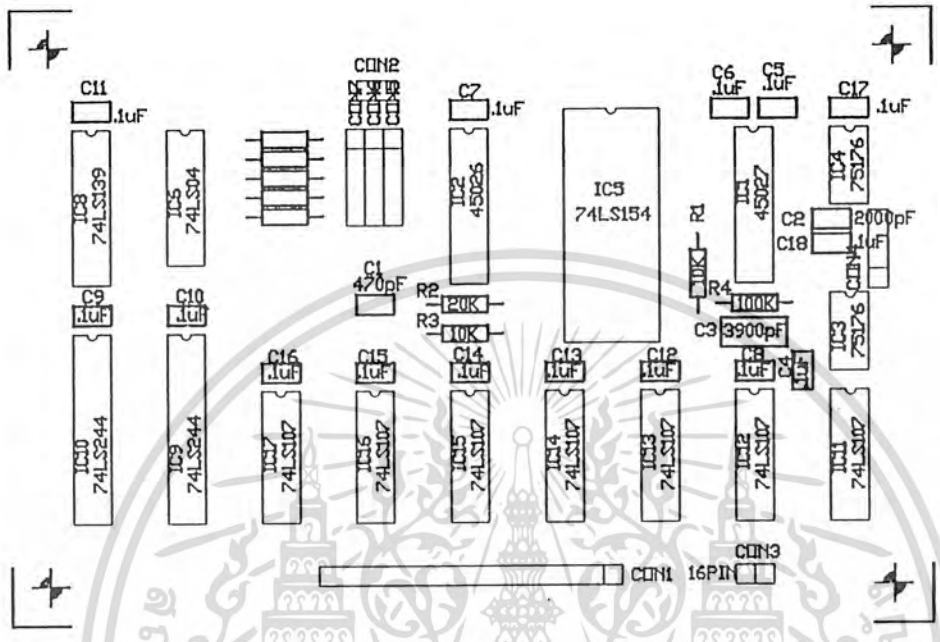




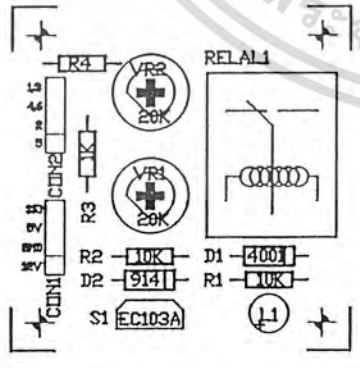
RX Top Layer

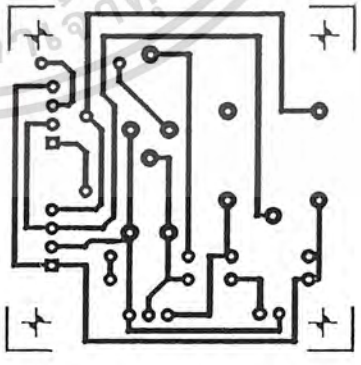


RX Bottom Layer



RX Top Overlay

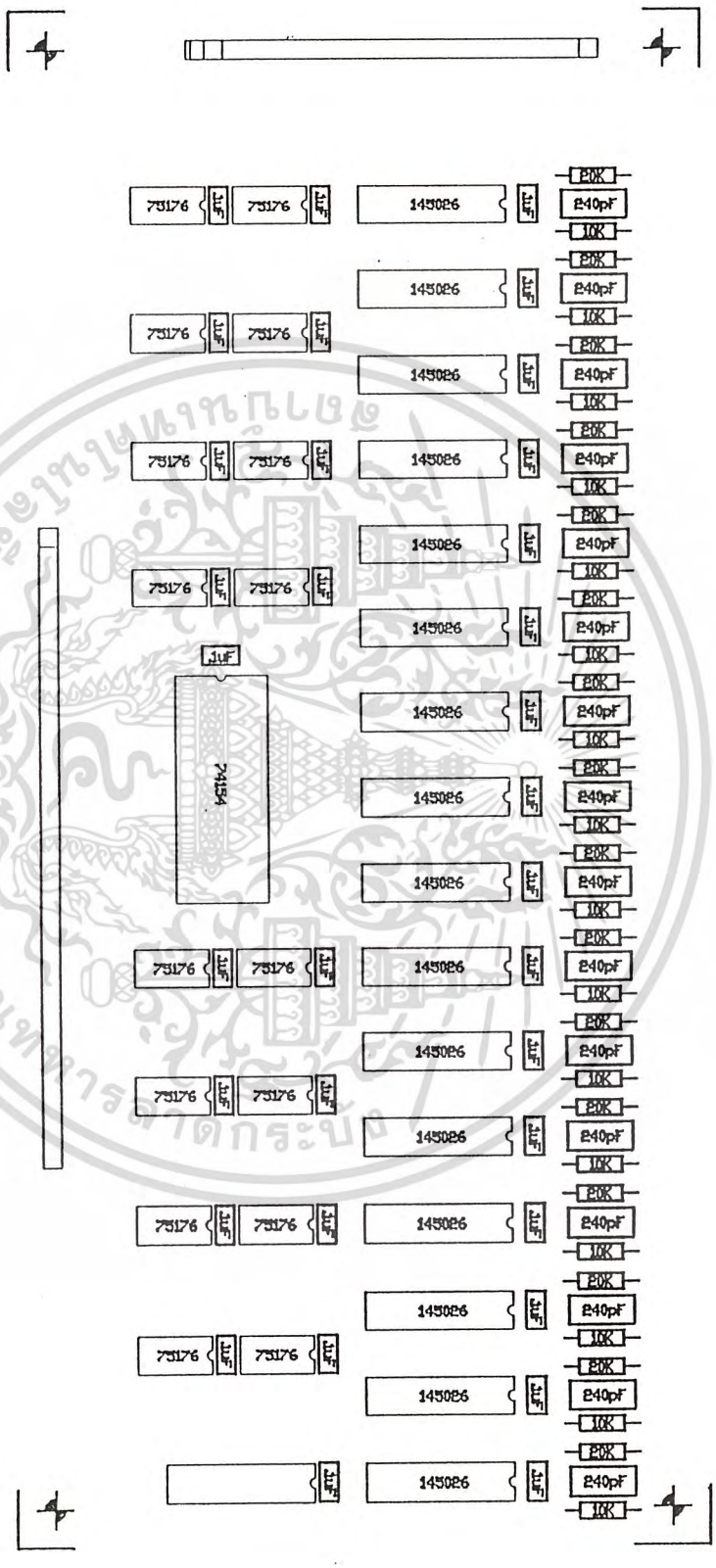


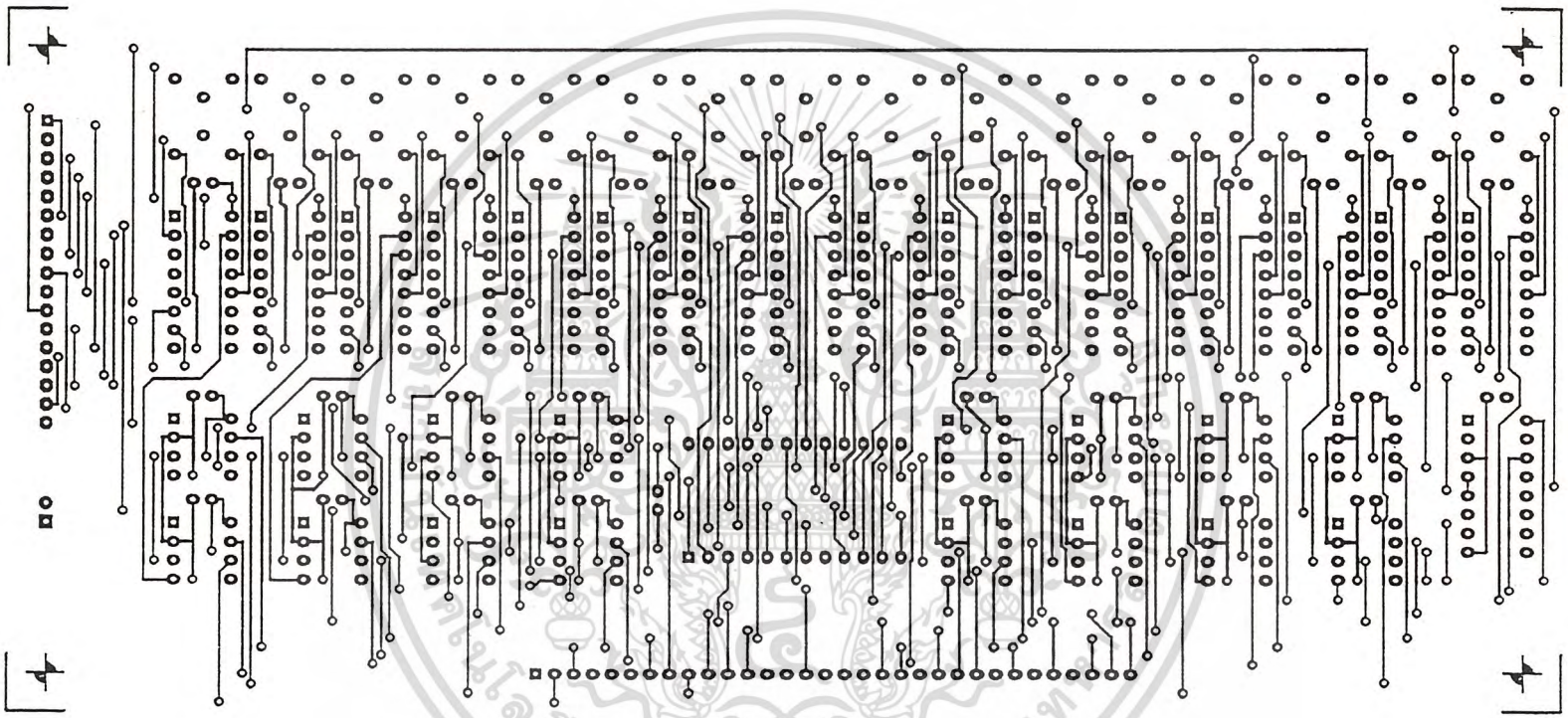


RELAY Top Overlay

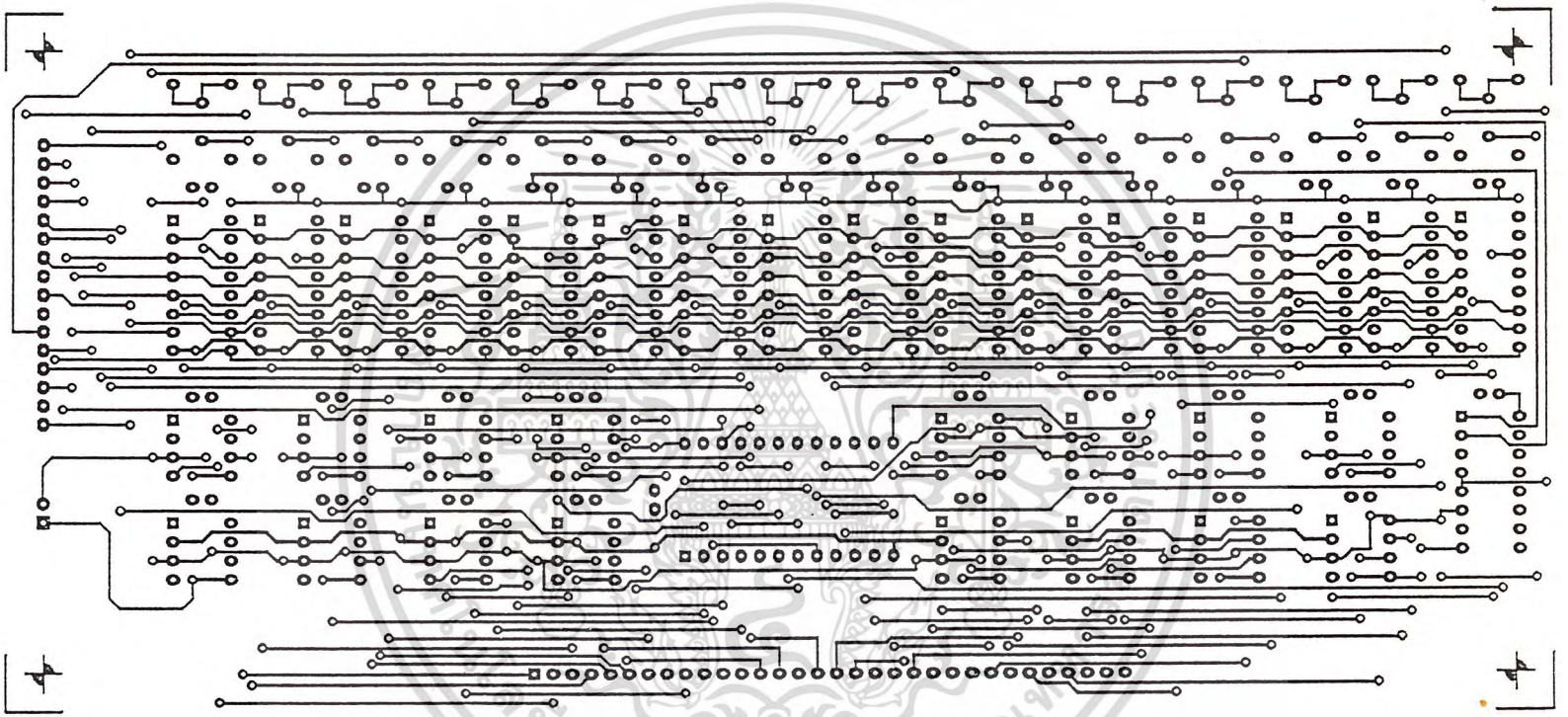
RELAY Bottom Layer

26 Top Overlay





26 Top Layer



26 Bottom Layer

รายการอุปกรณ์

ชุดควบคุมแบบ AUTOMATIC

ENCODER		DECODER	
R 10K 1/4W 5%	16 ตัว	R 10K 1/4W 5%	16 ตัว
R 20K 1/4W 5%	16 ตัว	R 100K 1/4W 5%	16 ตัว
C 470pF เซรามิก	16 ตัว	C 2000pF เซรามิก	16 ตัว
C 0.1μF	34 ตัว	C 3900pF เซรามิก	16 ตัว
IC 74LS138	1 ตัว	C 0.1	41 ตัว
IC 74154	1 ตัว	IC 74154	1 ตัว
IC MC145026	16 ตัว	IC 74LS244	8 ตัว
IC DS75176	16 ตัว	IC MC145027	16 ตัว
สายแพร์ 40เส้น	1 ม	IC DS75176	16 ตัว
DB9	16 ชุด	Connector 34 ขา	1 ตัว

ชุดควบคุมอุปกรณ์ปลายทาง

DECODER		ENCODER	
R 10K 1/4W 5%	1 ตัว	R 10K 1/4W 5%	1 ตัว
R 100K 1/4W 5%	1 ตัว	R 20K 1/4W 5%	1 ตัว
C 2000pF เซรามิก	1 ตัว	C 470pF เซรามิก	1 ตัว
C 3900pF เซรามิก	1 ตัว	C 0.1	4 ตัว
C 0.1μF	12 ตัว	IC 74LLS244	2 ตัว
IC 74HC04	1 ตัว	MC145026	1 ตัว
IC 74LS139	1 ตัว	DS75176	1 ตัว
IC 74LS154	1 ตัว	DB9	1 ชุด
IC 74LS107	7 ตัว	Connector	
IC MC145027	1 ตัว	Jumper 5	1 อัน
IC DS75176	1 ตัว	R คร่อมสาย 50	2

DRIVER LOAD

R 1K	1/4W 5%	13 ตัว
D 1N4001		13 ตัว
LED		13 ดวง
TR TIP110		1 ตัว
TR TIP29		6 ตัว
RELAY 12V 5A BLACK		13 ตัว
ตลับ Fuse		9 อัน
SW ผู้ใช้		13 อัน
R 1K	1/2W 5%	2 ตัว
R 330	1/4W 5%	1 ตัว
C 0.1μF 630V		1 ตัว
Q 4010L4		1 ตัว
IC MOC3020		1 ตัว
สายไฟ		
Connector		

SENSOR

R 1K	1/4W 5%	2 ตัว
R 10K	1/4W 5%	1 ตัว
R 20K	เกือกม้าแบบนอน	2 ตัว
D 1N914		1 ตัว
D 1N4001		1 ตัว
LED		1 ดวง
RELAY 12V 5A BLACK		1 ตัว
TGS # 812 หรือ # 813		1 ตัว
SW ผู้ใช้		1 ตัว

```
                .ORG  0000H
                ;
                .EQU   NUMBER     .  41BH
                .EQU   MAX_FLOOR  .  10H
                .EQU   MAX_ROOM   .  20H
                .EQU   NO_FLOOR   .  00H
                .EQU   STATU      .  04H
                .EQU   CNTLA0     .  00H
                .EQU   CNTLB0     .  02H
                .EQU   RDR0       .  08H
                .EQU   TDR0       .  06H
                .EQU   DCNTL      .  32H
                .EQU   IL         .  33H
                .EQU   ITC        .  34H
                .EQU   RCR        .  36H
                .EQU   CBR        .  38H
                .EQU   BBR        .  39H
                .EQU   CBAR       .  3AH
                .EQU   OMCR       .  3EH
                .EQU   P1CNTL     .  8FH
                .EQU   P1ROOM     .  9CH
                .EQU   P1FLOOR    .  8DH
                .EQU   P1SWITCH   .  8EH
;
;
START:          XOR  A
DELT1:          DEC  A
                NOP
                JP   NZ,DELT1
                JP   MAIN
                ;
                .ORG 0042H
                ;
ASCII:          .DW  0F6D6H   ;ADDRESS AT 1MS
                ;
                .ORG 100H
                ;
MAIN:           LD   HL,0AC00H
DEL:            DEC  HL
                LD   A,H
                OR   L
                JP   NZ,DEL
                LD   A,81H
                OUT  (P1CNTL),A
                LD   SP,0FECCH
                IN0  A,(ITC)
                LD   E,A
                BIT  7,A
                JP   Z,DIRE
                POP  HL
                RES  7,A
                OUT0 (ITC),A
                ;
DIRE:           LD   A,0F3H
                OUT0 (CBAR),A
                XOR  A
                OUT0 (BBR),A
                OUT0 (CBR),A
                LD   I,A
                LD   A,40H
                OUT0 (RCR),A
                IM   2
                EI
                LD   A,81H
                OUT  (OMCR),A
```

```
                LD   A,83H
                OUTO (RCR),A
                IN0  A,(DCNTL)
                AND  9FH
                OUTO (DCNTL),A
                ;
                LD   A,02H
                OUTO (CNTLB0),A
                LD   A,64H
                OUTO (CNTLA0),A
                LD   A,08H
                OUTO (STAT0),A
                XOR  A
                LD   (CFRS),A

WORK:           LD   A,(POWER_ON)
                CP   3FH
                JP   Z,WORKE

                LD   HL,DATAL
                LD   BC,NUMBER
                ;
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,0FH
                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
                IN   A,(P1SWITCH)
                RES  7,A
                OUT  (P1SWITCH),A
                LD   C,1FH
DELY:           CALL DELAY
                DEC  C
                JP   NZ,DELY
                SET  7,A
                OUT  (P1SWITCH),A
```

```
                LD   A,03FH
                LD   (POWER_ON),A

WORK0:          LD   A,(CHECK)
                CP   0F3H
                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

                CALL CHECKERR

                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   A,(IX+00)
                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


AFB1:           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,PAS
                CALL PUTSW1

SEEE:           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
                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   SEE1
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

SEE1:           LD   A,0E0H
                OUT  (P1SWITCH),A
                CALL DELAY
                LD   A,(ROOM)
                OUT  (P1ROOM),A
```

```
                CALL DELAY
                LD   A,(FLOOR)
                OR   40H
                OUT  (PIFLOOR),A
                CALL DELAY
                LD   A,(FLOOR)
                OUT  (PIFLOOR),A
                POP  BC
                POP  DE
                CALL DELAY
                IN   A,(PISWITCH)
                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,(PISWITCH)
                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,LNA

                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 AFR.
                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
ASAI:           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,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
ST1:            RET

;** FIND  POWER OR SWITCH

SCAN:           PUSH AF
                AND  07H
                INC  A
                LD   (POSITION),A
                LD   B,A
                POP  AF
                AND  38H
                JP   Z,DER
                PUSH HL
                INC  HL
                LD   C,(HL)
                POP  HL
                JP   L2
DER:            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:             SR   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 PUTSWI

SEETG:          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)
RU1:            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,00UH
                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
                    IN0  A,(RDR0)
                    CP   'U'
                    JP   Z,RT_INT1
                    CP   'E'
                    JP   Z,RT_INT2
                    CP   'S'
                    JP   Z,RT_INT3
                    POP  AF
                    EI
                    RETI
;
RT_INT1:            PUSH HL
                    PUSH DE
                    PUSH BC
                    LD   DE,1024
                    LD   HL,DATA1
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
                    OUT0 (STAT0),A
                    LD   HL,DATA1
                    LD   B,4

RT_INT21:
                    IN0  A,(STAT0)
                    BIT  7,A
                    JP   Z,RT_INT21
                    IN0  A,(RDR0)
                    LD   (HL),A
```

```
                INC  HL
                DEC  B
                JP   NZ,RT_INT21
;
                LD   A,0F5H
                LD   (CHECK),A
                LD   A,8
                OUT0 (STAT0),A

                CALL DELAY
                POP  BC
                POP  HL
                POP  AF
                EI
                RETI

RT_INT3:        PUSH HL
                PUSH BC
                LD   A,0
                OUT0 (STAT0),A
                LD   HL,CFRS

RT_INT31:       IN0  A,(STAT0)
                BIT  7,A
                JP   Z,RT_INT31
                IN0  A,(RDR0)
                LD   (HL),A
                INC  HL
                LD   B,A

RT_INT32:       IN0  A,(STAT0)
                BIT  7,A
                JP   Z,RT_INT32
                IN0  A,(RDR0)
                LD   (HL),A
                INC  HL
                DEC  B
                JP   NZ,RT_INT32

                XOR  A
                LD   (POWER_ON),A
                LD   A,8
                OUT0 (STAT0),A

                CALL DELAY
                POP  BC
                POP  HL
                POP  AF
                EI
                RET

;
;************************
;      SEND ASCII      *
;************************
; I/P = C
; REG = NONE

TX_ASCII:       PUSH AF
TX_ASC1:        IN0  A,(STAT0)
                BIT  1,A
                JP   Z,TX_ASC1
                OUT0 (TDR0),C
                POP  AF
                RET
```

```
DELAY:          PUSH AF
                PUSH HL
                PUSH BC
                LD   HL,020H
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
                ;
                .ORG  08000H
                ;
DATA1:          .RS   1025
DATA1:          .RS   5
TMP_BC:         .DW   0
TMP_DE:         .DW   0
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
CFRS:           .RS   20H
                .END
```

```
/* ------------------------------------------------------ *
 *           Four wire controller program                 *
 *            make for special project                    *
 *        of Industrial Electrical Technology             *
 *                March.29,1993                           *
 *       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>
/* ----- number of control room ----- */
#define FLOOR          16
#define ROOM           32
#define SWITCH          8
/* ------ 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
/* ------- Shadow 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    7
/* ---------------- Address Cursor --------------------*/
BIOSDATA far *bios = MK_FP(0x0040,0);
/* --------- window definition structures ------------ */
struct text_info windows [MAX_WINDOWS];
struct text_info wxw;   /* 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};
static int  r[16][32];
static int fl[16],sws[16][32][8][2];
static int chk[16][32];
static int te[1024] ;
int port ;
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);
        Init_port(0x0000);
        clrscr();
        make_scrn();              /* make background screen */
        bottom("Plase Wait. ..",FALSE,FALSE);

        delay(10);
        writeport('U');           /* loading data in borard z13 */
        for(i=0;i<1024;i++)
           te[i]=readport();
        load(TRUE);               /* 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 number 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 number 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(TRUE);                      /* 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. . . .",FALSE,FALSE);
                       writeport('U');      /* loading data in borard zilc */
                       for(i=0;i<1024;i++)
                           te[i]=readport();
                       bottom("-Move cursor   Enter-Select",FALSE,VER);
                       break;
              case 1 :
                       save(FALSE);         /* save configulation file */
                       break;
              case 2 :
                       load(FALSE);         /* 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(int arrow)
{
        FILE *out;
        MENU sav = {2,YELLOW,MAGENTA,"Save Check","Save Status"};
        char ch;
        int c,i,j,sa,x=0,y=0;
```

```
if (arrow1==FALSE)
{
   ch = move_cursor(26,6,1,2,sav,&sa,&x,&y);
   if (ch==CR)
   {
       switch(sa)
       {
         case 0 :
                  bottom("Saving Check Config File . . ."
                                          ,FALSE,FALSE);
                  out = fopen("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++)
                      putc(chk[i][j],out);
                  fclose(out);
                  break;
         case 1 :
                  bottom("Saving Status Config File . . ."
                                          ,FALSE,FALSE);
                  out = fopen("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++)
                    putc(ta[i],out);
                 fclose(out);
                 break;
       }
   close_window();
}
else
{
  bottom("Saving Check Config File . . .",FALSE,FALSE);
  out = fopen("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++)
            putc(chk[i][j],out);

      bottom("-Move cursor   Enter-Select",FALSE,VER);
      fclose(out);
      bottom("Saving Status Config File . . .",FALSE,FALSE);
      out = fopen("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++)
         putc(te[i],out);
      fclose(out);
   }
   bottom("-Move cursor   Enter-Select",FALSE,VER);
}

save_setup()
{
   FILE *out;
   char ch;
   int c,l,j,sa;
     bottom("Saving Setup Config File . . .",FALSE,FALSE);
         out = fopen "SETUP.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;
         }
            putc(flr,out);
            for(i=0;i<flr;i++)
               putc(room[i],out);
   bottom("-Move cursor   Enter-Select",FALSE,VER);
   fclose(out);
```

```
        bottom("-Move cursor   Enter-Select",FALSE,WHR);
}
/* ----- loading configulation file ----- */
load(int arrow1)
{
        FILE *in;
        MENU lod = {2,YELLOW,MAGENTA,"Load Check","Load Status"};
        char ch;
        int c,i,j,la,x=0,y=0;
      . if(arrow1==FALSE)
        {
           ch = move_cursor(26,7,1,2,lod,&la,&x,&y);
           if (ch==CR)
           {
               switch(la)
               {
                  case 0:
                          bottom("Loading Check Config File . . ."
                                                  ,FALSE,FALSE);
                          in = fopen("CHECK.CFG","r");
                          if (in == NULL)
                          {
                             open_window(26,12,53,14,YELLOW,BROWN,1,TRUE);
                             cprintf(" Can not load check 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;
                             }
                          fclose(in);
                          break;
                  case 1:
                          bottom("Loading Status Config File . . ."
                                                  ,FALSE,FALSE);
                          in = fopen("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;
                          }
```

```
                fclose(in);
                break;
            }
        }
        close_window();
    }
    else
    {
        bottom("Loading Check Config File . . .",FALSE,FALSE);
        in = fopen("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;
            }

        bottom("-Move cursor   Enter-Select",FALSE,VER);
        fclose(in);

        bottom("Loading Setup Config File . . .",FALSE,FALSE);
        in = fopen("SETUP.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;
        }
        c = getc(in);
        flr = c;
        for (i = 0; i < flr; i++)
        {
            c = getc(in);
            rom[i] = c;
        }
        bottom("-Move cursor   Enter-Select",FALSE,VER);
        fclose(in);
    }
    bottom("-Move cursor   Enter-Select",FALSE,VER);
}
/* ---- 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 = {8,LIGHTRED,BLUE,
        "Switch 1","Switch 2","Switch 3","Switch 4",
        "Switch 5","Switch 6","Switch 7","Switch 8"};
        /* 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
          {
           clrs();
           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 == '0' || check == 'o')
 {
    chk[flor][rm] = WHITE;
    r[flor][rm]   = WHITE;
       i=0;
       aress= flor*0x40+rm*2;
       te[aress]&=(~power(2,7-i&7))&0xff;
       frs[0]=((((poer<<1)<<1)|1)<<5)|rm;
       frs[1]=(flor<<4)|i;
       writeport('E');
       writeport(frs[0]&0xDF);
       writeport(frs[1]);
       writeport(frs[0]&0xDF);
       writeport(frs[1]);
 }
 else
 {
    chk[flor][rm] = YELLOW;
    r[flor][rm]   = YELLOW;
    x3 = 0;
    y3 = 0;
    swit = 0;
    do
    {
      ch3 = move_cursor(22,12,4,2,sw,&swit,&x3,&y3);
      if(ch3 == CR)
      {
        x4=0;
        y4=0;
        sta = 0;
        do
        {
          bottom("-Move cursor   Enter-Select",FALSE,BIDIR);
          x = (x3*11)+16+strlen(sw.string[0]);
          y =  y3+14;
          ch4 = move_cursors(2,sws[flor][rm][swit],x,y,1,2,status,&sta,&x4,&y4);
          if (ch4 == CR)
          {
            sws[flor][rm][swit][sta]   = YELLOW;
            sws[flor][rm][swit][abs(sta-1)] = WHITE;
            aress= flor*0x40+rm*2;
            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');
            writeport(frs[0]);
            writeport(frs[1]);
            writeport(frs[0]&0xDF);
            writeport(frs[1]);
          }
          close_window();
          bottom("-Move cursor   Enter-Select",FALSE,BIDIR);
        } while (ch4 != ESC && ch4 != CR);
      }
      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,BOR);
}
/* ----- 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;
        cirs();
        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;
```

```
            }
            }
            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 one is on or off */
        int py,count;
        int flor,rm,sress;
        int x1,y1,x2,y2;
        clrs();
        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;
```

```
    writeport('S');
    writeport(flr);
    for(i=0;i<flr;i++)
       writeport(rom[i]);
}
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 1993",
                       "DIRECTED BY",
                       "MR.SOMCHAI SIRISEETAWONG",
                       "KING MONGKUT'S INSTITUTE OF TECHNOLOGY",
                       "LADKRABANG",
                       "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&1)
                for (count=1; count<=strlen(text[j]) ; count++)
                {
                   temp = slide(text[j],count);
                   sound(50);
```

```
    writeport('S');
    writeport(flr);
    for(i=0;i<flr;i++)
        writeport(rom[i]);
}
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[l];
                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 1993",
                        "DIRECTED BY",
                        "MR.SOMCHAI SIRISATTAWONG",
                        "KING MONGKUT'S INSTITUTE OF TECHNOLOGY",
                        "LADKRABANG",
                        "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);
```

```
        }
        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 < 0)
                              *x = dimy-1;
                        }
                        break;
            case 0x48 :
                        if (dimx > 1)    /* move up */
                        {
                           *y--;
                           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,
    int temp = 0;
    int width = 0;
    char ch = 0;
```

```
        }
        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;
```

```
                    (*x)++;
                    if (((*x) > dimx-1) || ((*x) == max % dimx) && ((*y) == max / dimx))
                      (*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,x1;
    char s[140],m[140],d;
    va_start( argptr , format ) ;
    vsprintf( s , fmt , argptr ) ;
    cprintf("%s" , s ) ;
    memset(m,'\0',140);
    x1 = 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 :  *num = atoi(m);
                       if(*num > mx || *num < 0)
                       {
                        clrscr();
                        cprintf("INVALID NUMBER, PRESS ANY KEY TO AGAIN");
                        getch();
                        clrscr();
                        cprintf("%s",s);
                        i = 0;
                        memset(m,'\0',140);
                        again = TRUE;
                       }
                       else if ( *num == 0 )
                       {
                        *num = mx ;
                        again = FALSE;
                       }
                       else
                        again = FALSE;
                       break;
           case  0 :  d = getch();
                       if(d == 0x4B)
                       {
                       if(x1 < wherex() - 1)
                       {
                           gotoxy(wherex() - 1,wherey());
                           cprintf("%c",' ');
                           gotoxy(wherex() - 1,wherey());
                       }
                       else
                       {
                           gotoxy(x1,wherey());
                           cprintf("%c",' ');
                           gotoxy(x1,wherey());
                       }
                       i--;
                       m[i] = '\0';
                       }
                       break;
            }
    }while(again && d != ESC);
    return(d);
}
/* -------------- Make screen ---------------- */
make_scrn()
{
        char *bar = " FOUR 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 + 2;
        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[140];
        int i;
        for (i=0; i<len; i++)
```

```
        {
            str[i] = string[i];
        }
        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);
}
/* ------- Initize port series -------- */
Init_port(unsigned active_port)
{
       union REGS r;
       port = active_port;
       r.h.al = 0xf3;
       r.x.dx = port;
       int86(0x14,&r,&r);
       return(r.h.ah);
}
/* ------- sent character to port series -------- */
writeport(char fmt)
{
     union REGS r;
     r.h.ah = 0x01;
     r.h.al = fmt;
     r.x.dx = port;
     int86(0x14,&r,&r);
}
/* ------ recive character from port series ------- */
readport()
{
      union REGS r;
      r.x.dx = port;
      r.h.ah = 2;
      int86(0x14,&r,&r);
      return(r.h.al);
}
```

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

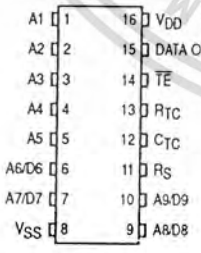
D SUFFIX
SOG
CASE 751B

DW SUFFIX
SOG
CASE 751G

ORDERING INFORMATION

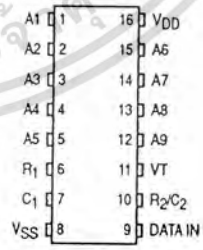
MC145026P, SC41342P	Plastic DIP
MC145026D, SC41342D	SOG Package
MC145027P, SC41343P	Plastic DIP
MC145027DW, SC41343DW	SOG Package
MC145028P, SC41344P	Plastic DIP
MC145028DW, SC41344DW	SOG Package

PIN ASSIGNMENTS



MC145026
SC41342
ENCODERS

MC145027
SC41343
DECODERS



MC145028
SC41344
DECODERS

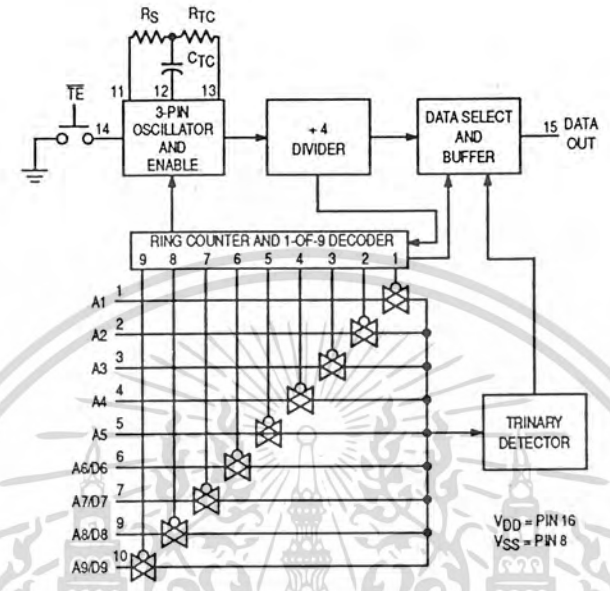


Figure 1. MC145026 Encoder Block Diagram

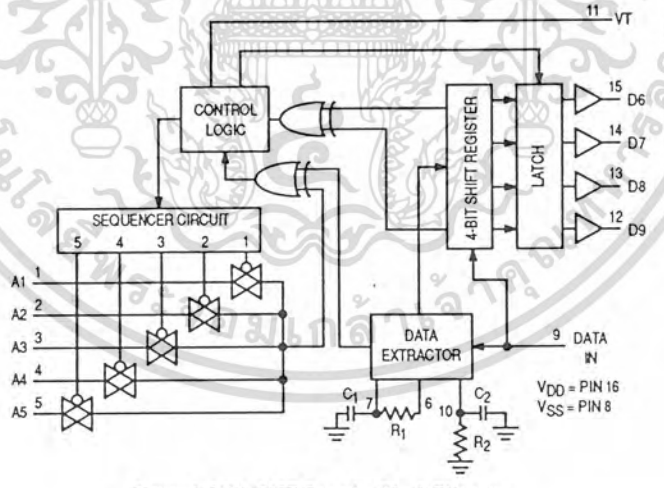


Figure 2. MC145027 Decoder Block Diagram

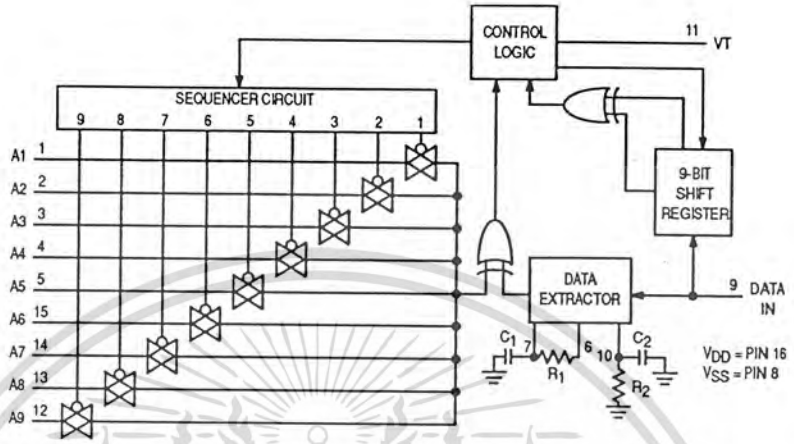


Figure 3. MC145028 Decoder Block Diagram

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_{out} = 4.5 or 0.5 V) (V_{out} = 9.0 or 1.0 V) (V_{out} = 13.5 or 1.5 V)	5.0	—	1.5	—	1.5	—	1.5	V
		10	—	3.0	—	3.0	—	3.0	
		15	—	4.0	—	4.0	—	4.0	
V_{IH}	High-Level Input Voltage (V_{out} = 0.5 or 4.5 V) (V_{out} = 1.0 or 9.0 V) (V_{out} = 1.5 or 13.5 V)	5.0	3.5	—	3.5	—	3.5	—	V
		10	7.0	—	7.0	—	7.0	—	
		15	11	—	11	—	11	—	
I_{OH}	High-Level Output Current (V_{out} = 2.5 V) (V_{out} = 4.6 V) (V_{out} = 9.5 V) (V_{out} = 13.5 V)	5.0	−2.5	—	−2.1	—	−1.7	—	mA
		5.0	−0.52	—	−0.44	—	−0.36	—	
		10	−1.3	—	−1.1	—	−0.9	—	
		15	−3.6	—	−3.0	—	−2.4	—	
I_{OL}	Low-Level Output Current (V_{out} = 0.4 V) (V_{out} = 0.5 V) (V_{out} = 1.5 V)	5.0	0.52	—	0.44	—	0.36	—	mA
		10	1.3	—	1.1	—	0.9	—	
		15	3.6	—	3.0	—	2.4	—	
I_{in}	Input Current — $\overline{TE}$ (MC145026 and SC41342, Pullup Device)	5.0	—	—	3.0	11	—	—	µA
		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 A1-A5, A6/D6–A9/D9 (MC145026 and SC41342), A1-A5 (MC145027), A1-A9 (MC145028)	5.0	—	—	—	±110	—	—	µA
		10	—	—	—	±500	—	—	
		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 (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

ELECTRICAL CHARACTERISTICS — SC41343 and SC41344 (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})	2.8	—	0.05	—	0.05	—	0.05	V
		5.0	—	0.05	—	0.05	—	0.05	
		10	—	0.05	—	0.05	—	0.05	
V_{OH}	High-Level Output Voltage (V_{in} = 0 V or V_{DD})	2.8	2.75	—	2.75	—	2.75	—	V
		5.0	4.95	—	4.95	—	4.95	—	
		10	9.95	—	9.95	—	9.95	—	
V_{IL}	Low-Level Input Voltage								V
	(V_{out} = 2.3 V or 0.5 V)	2.8	—	0.84	—	0.84	—	0.84	
	(V_{out} = 4.5 V or 0.5 V)	5.0	—	1.5	—	1.5	—	1.5	
	(V_{out} = 9.0 V or 1.0 V)	10	—	3.0	—	3.0	—	3.0	
V_{IH}	High-Level Input Voltage								V
	(V_{out} = 0.5 V or 2.3 V)	2.8	1.96	—	1.96	—	1.96	—	
	(V_{out} = 0.5 V or 4.5 V)	5.0	3.5	—	3.5	—	3.5	—	
	(V_{out} = 1.0 V or 9.0 V)	10	7.0	—	7.0	—	7.0	—	
I_{OH}	High-Level Output Current								mA
	(V_{out} = 1.4 V)	2.8	−0.73	—	−0.7	—	−0.55	—	
	(V_{out} = 4.5 V)	5.0	−0.59	—	−0.5	—	−0.41	—	
	(V_{out} = 9.0 V)	10	−1.3	—	−1.1	—	−0.9	—	
I_{OL}	Low-Level Output Current								mA
	(V_{out} = 0.4 V)	2.8	0.35	—	0.3	—	0.24	—	
	(V_{out} = 0.5 V)	5.0	0.8	—	0.6	—	0.4	—	
	(V_{out} = 1.0 V)	10	3.5	—	2.9	—	2.3	—	
I_{in}	Input Current — Data In	10	—	±0.3	—	±0.3	—	±1.0	µA
I_{in}	Input Current	2.8	—	—	—	±30	—	—	µA
	A1-A5 (SC41343),	5.0	—	—	—	±140	—	—	
	A1-A9 (SC41344)	10	—	—	—	±600	—	—	
C_{in}	Input Capacitance (V_{in} = 0)	—	—	—	—	7.5	—	—	pF
I_{DD}	Quiescent Current	2.8	—	—	—	60	—	—	µA
		5.0	—	—	—	75	—	—	
		10	—	—	—	150	—	—	
I_{dd}	Dynamic Supply Current	2.8	—	—	—	300	—	—	µA
	(f_c = 20 kHz)	5.0	—	—	—	500	—	—	
		10	—	—	—	1000	—	—	

SWITCHING CHARACTERISTICS — MC145026, MC145027, MC145028, and SC41342* (C_L = 50 pF, T_A = 25°C)

Symbol	Characteristic	V_{DD}	Guaranteed Limit		Unit
			Min	Max	
t_{TLH}, t_{THL}	Output Transition Time (Figures 4 and 8)	5.0 10 15	— — —	200 100 80	ns
t_r	Data In Rise Time (Decoders) (Figure 5)	5.0 10 15	— — —	15 15 15	µs
t_f	Data In Fall Time (Decoders) (Figure 5)	5.0 10 15	— — —	15 5.0 4.0	µs
f_{osc}	Encoder Clock Frequency (Figure 6)	5.0 10 15	0.001 0.001 0.001	2.0 5.0 10	MHz
f	Decoder Frequency (Referenced to Encoder Clock) (Figure 14)	5.0 10 15	1.0 1.0 1.0	240 410 450	kHz
t_w	$\overline{TE}$ Pulse Width (Encoders) (Figure 7)	5.0 10 15	65 30 20	— — —	ns

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

SWITCHING CHARACTERISTICS — SC41342 (C_L = 50 pF, T_A = 25°C)

Symbol	Characteristic	V_{DD}	Guaranteed Limit		Unit
			Min	Max	
t_{TLH}, t_{THL}	Output Transition Time (Figures 4 and 8)	2.5	—	450	ns
f_{osc}	Encoder Clock Frequency (Figure 6)	2.5	1.0	250	kHz
t_w	$\overline{TE}$ Pulse Width (Figure 7)	2.5	—	—	ns

SWITCHING CHARACTERISTICS — SC41343 and SC41344 (C_L = 50 pF, T_A = 25°C)

Symbol	Characteristic	V_{DD}	Guaranteed Limit		Unit
			Min	Max	
t_{TLH}, t_{THL}	Output Transition Time (Figures 4 and 8)	2.8 5.0 10	— — —	320 200 100	ns
t_r	Data In Rise Time (Figure 5)	2.8 5.0 10	— — —	15 15 15	µs
t_f	Data In Fall Time (Figure 5)	2.8 5.0 10	— — —	15 15 5.0	µs
f	Decoder Frequency (Referenced to Encoder Clock) (Figure 14)	2.8 5.0 10	1.0 1.0 1.0	100 240 410	kHz

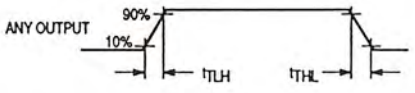


Figure 4.

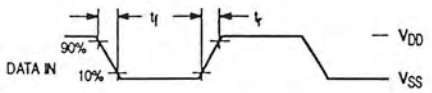


Figure 5.

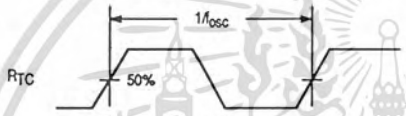
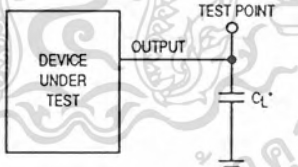


Figure 6.

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.

Upon power-up the MC145026 can continuously transmit data as long as $\overline{TE}$ remains low. The device can transmit two-word sequences by pulsing $\overline{TE}$ low. However, no application should be designed to rely upon the first data word transmitted after power-up, because this first word is invalid data.

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.

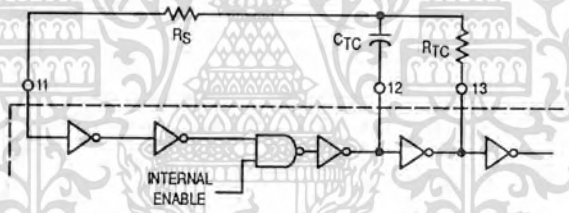
VT remains high until either a mismatch is received or no input signal is received for four data periods.

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.,

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

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

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

$R_S \approx 2 R_{TC}$

$R_S \ge 20 \text{ k}$

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

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

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

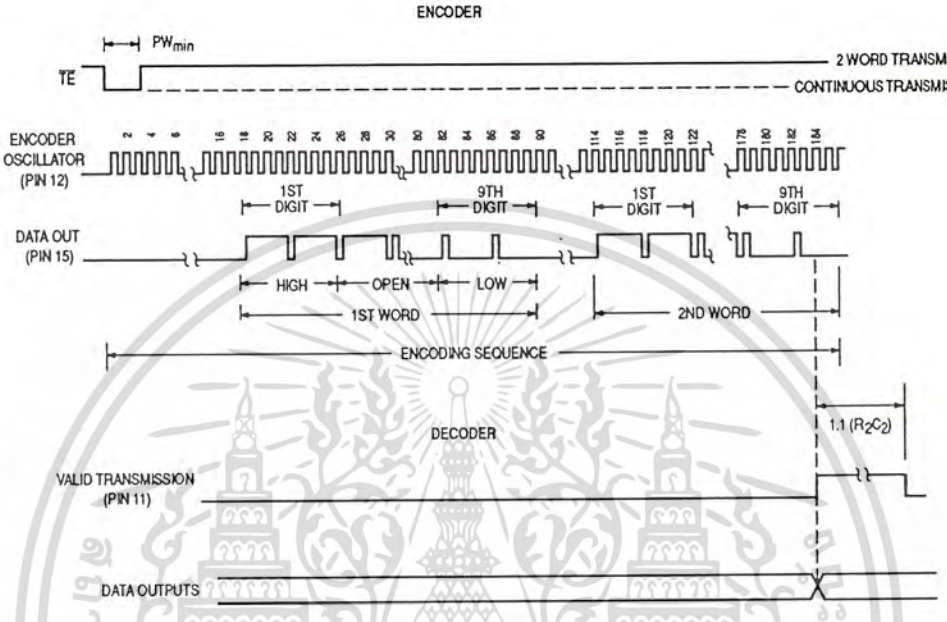


Figure 10. Timing Diagram

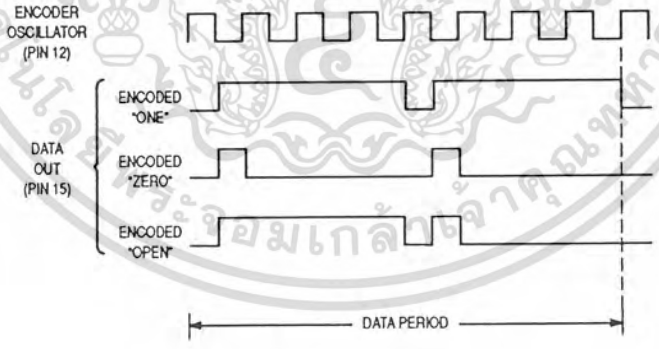


Figure 11. Encoder Data Waveforms

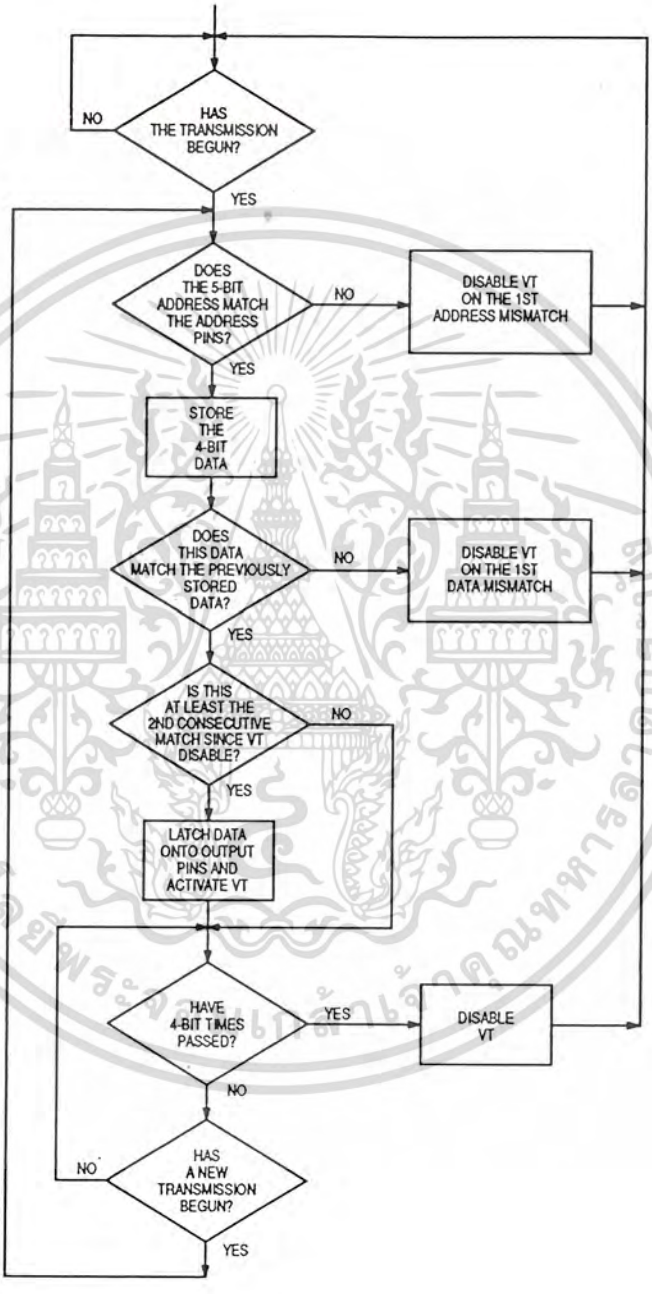


Figure 12. MC145027 Flowchart

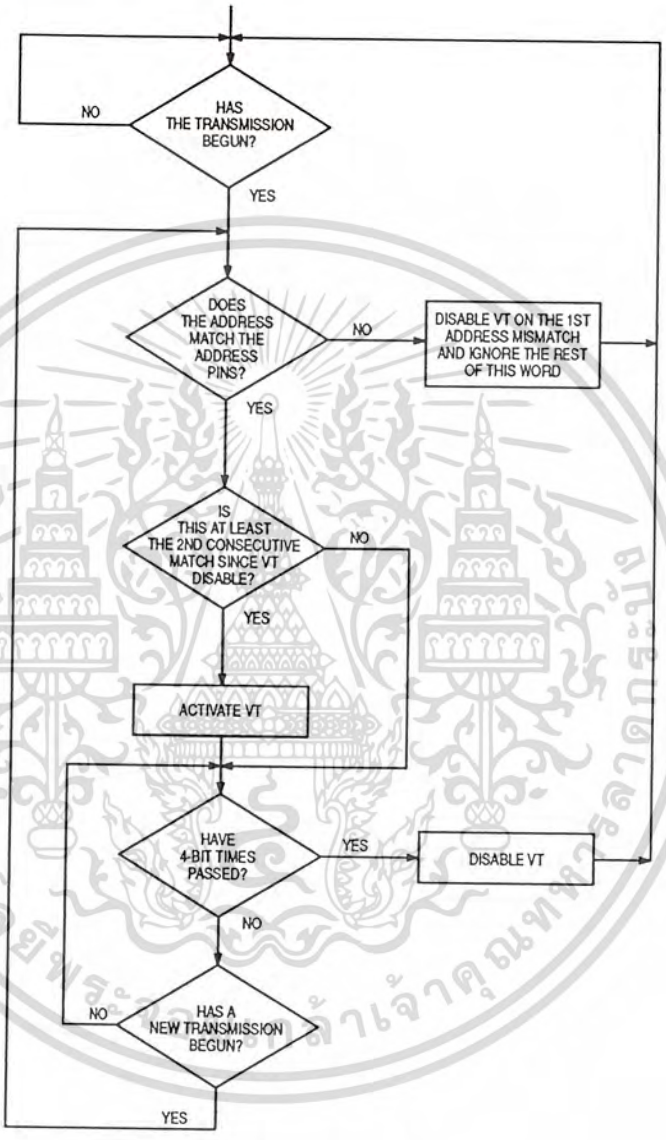


Figure 13. MC145028 Flowchart

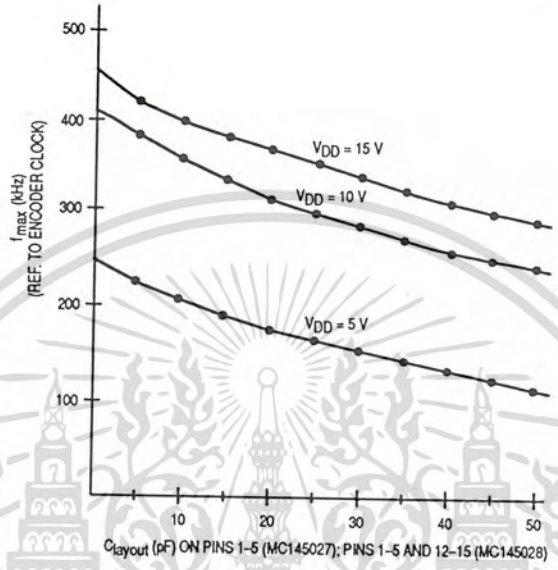
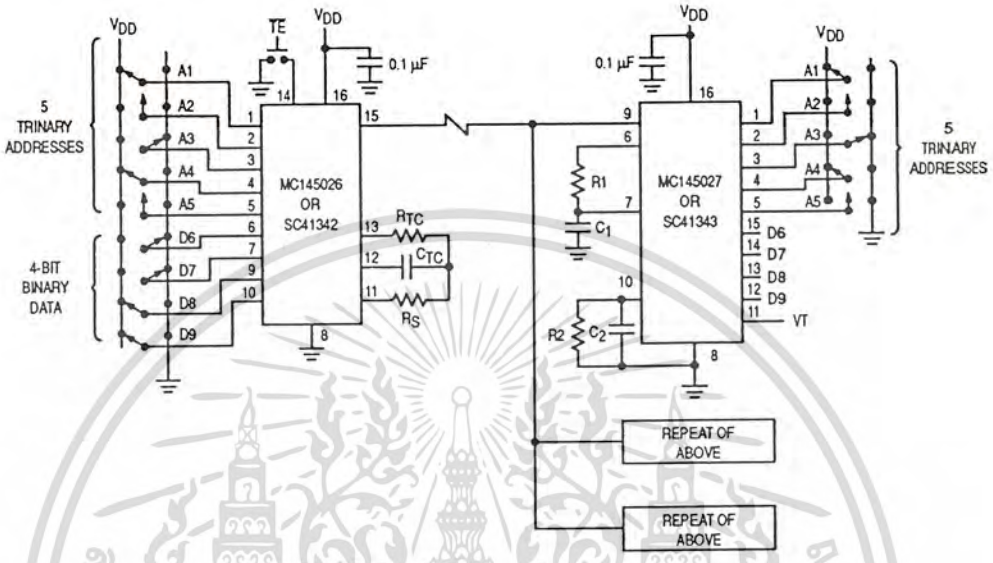


Figure 14. f_{max} vs C_{layout} — Decoders Only



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

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

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

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

Example R/C Values
(All Resistors and Capacitors are ±5%)

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

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.

Trinary Switch Manufacturers

Midland Ross–Electronic Connector Div.	617/491-5400
Greyhill	312/354-1040
Augat/Alcoswitch	617/685-4371
Aries Electronics	201/996-6841

The above companies may not have the switches in a DIP. For more info, call them or consult EEM or Gold Book. *Ask for SPDT with center OFF.*

Alternative: A SPST can be placed in series between a SPDT and the Encoder or Decoder to achieve trinary action.

Motorola cannot recommend one supplier over another and in no way suggests that this is a complete listing of trinary switch manufacturers.

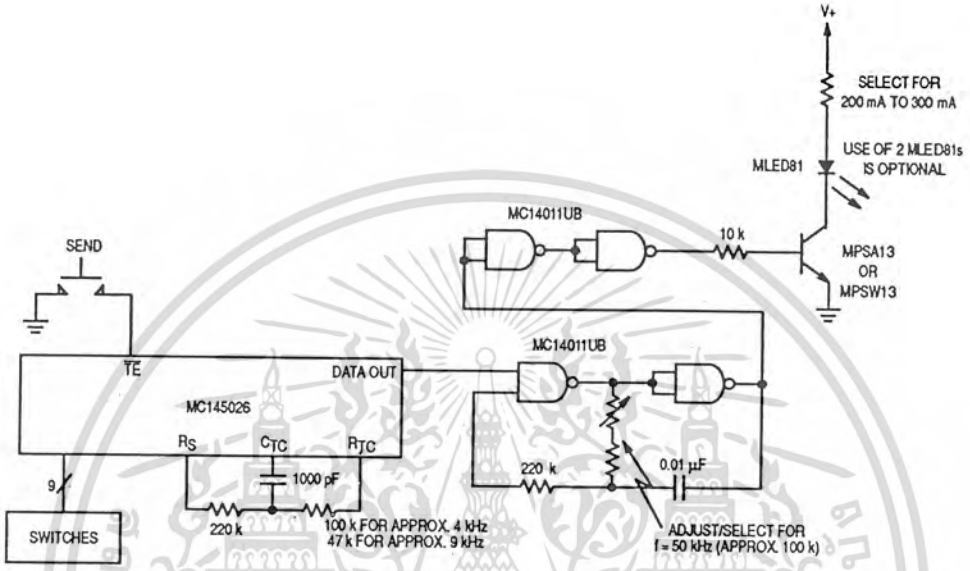


Figure 16. IRED Transmitter Using RC Oscillator to Generate Carrier Frequency

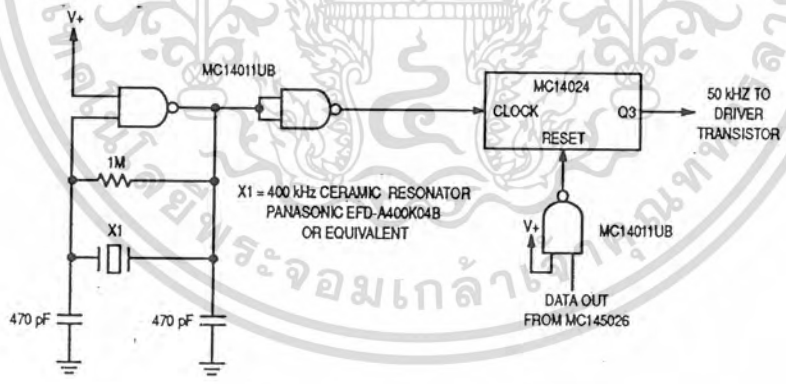


Figure 17. Using a Ceramic Resonator to Generate Carrier Frequency

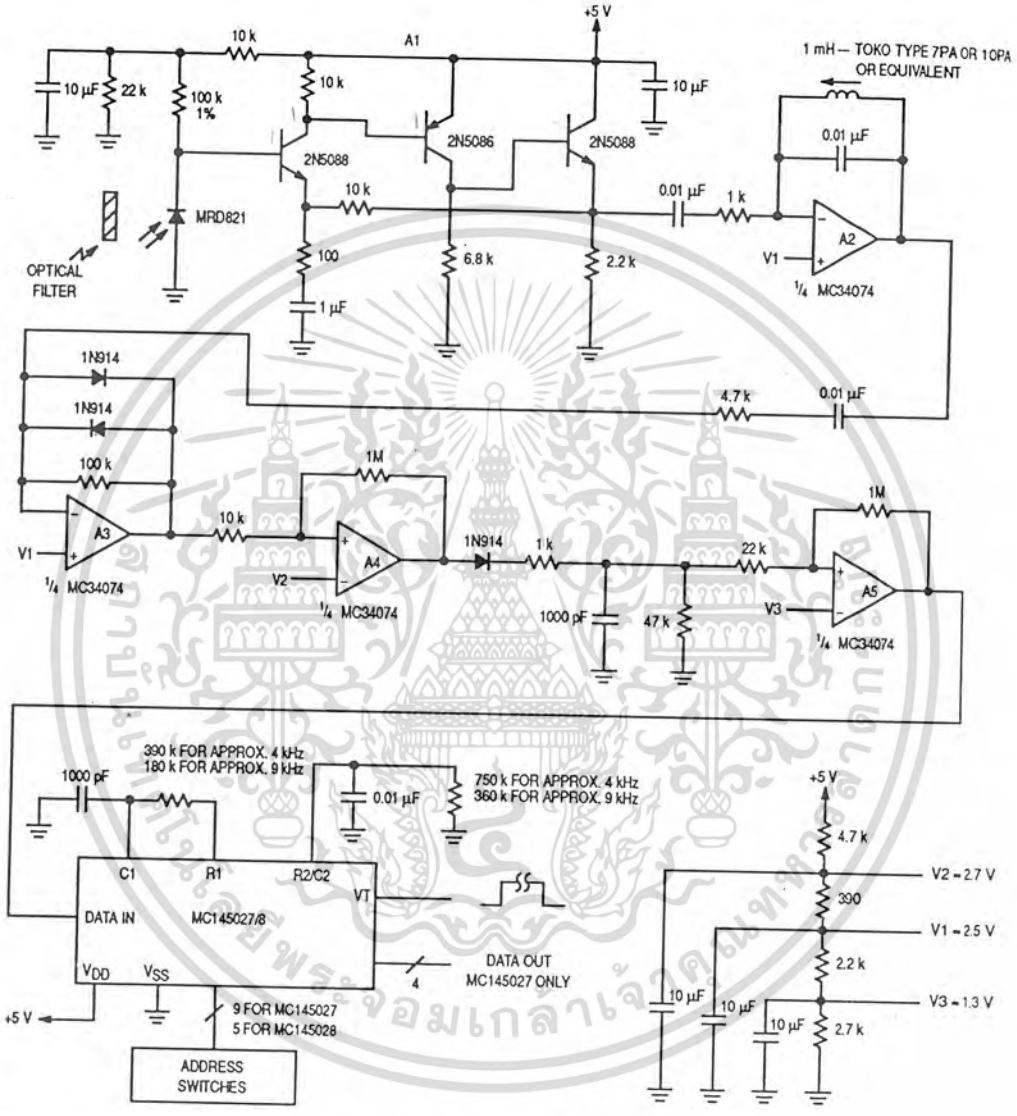


Figure 18. Infrared Receiver

หนังสืออ้างอิง

- ดร.วิทยา เรืองพรวิสุทธิ์ "คู่มือโปรแกรมภาษา C สำหรับผู้เริ่มต้น" บริษัท ซีเอ็ดยูเคชั่น จำกัด พ.ศ.2536
- ทีมงาน อีทีที "คู่มือ Board CP-180 (Control Pack Z80180)" บริษัท อีทีที จำกัด พ.ศ.2535
- ทีมงาน อีทีที "คู่มือไอซี ไมโครโปรเซสเซอร์" บริษัท อีทีที จำกัด พ.ศ.2535
- ทีมงาน ซีเอ็ด "รวมโครงงานอิเล็กทรอนิกส์ เล่ม 2" บริษัท ซีเอ็ดยูเคชั่น จำกัด พ.ศ.2535
- ทีมงาน ซีเอ็ด "คู่มือดูชาไอซี" บริษัท ซีเอ็ดยูเคชั่น จำกัด พ.ศ.2529
- ทีมงาน ซีเอ็ด "คู่มือ/เทียบเบอร์ ไอซี TTL" บริษัท ซีเอ็ดยูเคชั่น จำกัด พ.ศ.2532
- ทีมงาน ซีเอ็ด "คู่มือทรานซิสเตอร์" บริษัท ซีเอ็ดยูเคชั่น จำกัด พ.ศ.2533
- ทีมงาน National Semiconductor "Semiconductor Master Selection Guide" National Semiconductor ค.ศ.1990
- ทีมงาน National Semiconductor "Interface Databook" National Semiconductor ค.ศ.1990

สถานที่ติดต่อ

นายบัณฑิต ตัญบุญยกิจ

49/188 ถ.สุขุมวิท ต.บางเมือง อ.เมือง จ.สมุทรปราการ

นายสมชาย ศิริเศรษฐวงศ์

275 ถ.มหาไชย แขวงสำราญราษฎร์ เขตพระนคร กรุงเทพฯ 10200

นายสมศักดิ์ สมบุญจันทร์

กรมสื่อสารทหารอากาศ ดอนเมือง กรุงเทพฯ โทร. 534-3358

นายสัญชัย พรหมเทพ

2/25 ช.3 ม.ธนินทร ถ.วิภาวดีรังสิต ดอนเมือง กรุงเทพฯ โทร.536-1768
