ปีการศึกษา 2535

การควบคุมและตรวจสอบอุปกรณ์ภายในบ้าน

โดย

นาย เทอดชัย ธเนศวโรดม

นางสาว วัฒนา มณีฉาย

.....

อาจารย์ที่ปรึกษา

รศ.ดร. ชม กิ้มปาน

อ. สมศักดิ์ มิตะถา

ปริญญานิพนธ์ปีการศึกษา 2535

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การควบคุมและตรวจสอบอุปกรณ์ภายในบ้าน

ผู้จัดทำ

นาย เทอดชัย ธเนศวโรดม

นางสาว วัฒนา มณีฉาย

.....อาจารย์ที่ปรึกษา

(อ.สมศักดิ์ มิตะถา)

.....อาจารย์ที่ปรึกษา

(รศ.ดร. ชม กิ้มปาน)

การควบคุมและตรวจสอบอุปกรณ์ภายในบ้าน

HOME EQUIPMENT CONTROL

จัดทำโดย

1. นาย เทอดชัย ธเนศวโรดม ปี 4D 321107
2. นางสาว วัลนา มณีฉาย ปี 4D 321290

อาจารย์ที่ปรึกษา

1. รศ.ดร.ชม กิ้มปาน
2. อ.สมศักดิ์ มิตะถา

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2535

สารบัญ

การควบคุมและตรวจสอบอุปกรณ์ภายในบ้าน

ผู้จัดทำ

1. นายเทอดชัย ธเนศวโรดม

2. นางสาววัฒนา มณีฉาย

รศ.ดร. ชม กิ้มปาน

อ. สมศักดิ์ มิตะถา อาจารย์ที่ปรึกษา

ปีการศึกษา 2535

บทคัดย่อ

เนื่องจากในปัจจุบัน ความสะดวกสบายและความรวดเร็วในการทำงานเป็นที่ต้องการของมนุษย์มากขึ้นจึงได้มีการพัฒนาเทคโนโลยีในระบบต่างๆขึ้นมากมายเพื่อสนองความต้องการนี้

โครงงานนี้เป็นเทคโนโลยีอีกอย่างหนึ่ง ที่ให้ความสะดวกในการควบคุมอุปกรณ์เครื่องใช้ไฟฟ้าต่างๆ ภายในบ้าน ซึ่งในระบบนี้ประกอบด้วยโมดูลควบคุมการเปิดปิด และ โมดูลควบคุมย่อย การควบคุมอุปกรณ์สามารถทำได้โดยผ่านทางโมดูลควบคุมการเปิดปิดอุปกรณ์โดยตรง หรือผ่านทางโมดูลควบคุมย่อย ซึ่งมีอยู่หลายจุด โมดูลควบคุมย่อยแต่ละจุดสามารถตรวจสอบและควบคุมอุปกรณ์ได้ทั้งระบบ

HOME EQUIPMENT CONTROL

BY

MR. TERDCHAI TANETWARODOM

MISS WATTANA MANEECHY

DR. CHOM KIMPAN

SOMSAK MITTATA ADVISOR

ACADEMIC YEAR 1992

ABSTRACT

Nowadays,convenience and speed in working are important things for human. So there are a lot of projects and researches for responding these demands.

This project is also one that make convenience in controlling of electrical equipments in homes. This system consists of these equipments ; controlled modules (terminals), sensor module, and control modules. There are two ways to control equipments, by control the controlled module directly or control pass any controlled module. Each control module can control all controlled module on the system.

วัตถุประสงค์

เพื่อศึกษาและทดลองสร้าง ระบบเครือข่ายควบคุมการทำงาน โดยใช้ไมโครคอนโทลเลอร์ MCS-51 ในการเชื่อมต่อเป็นระบบเครือข่ายสำหรับควบคุมภายในบ้าน และ ศึกษาโปรโตคอลการติดต่อระหว่างอุปกรณ์ควบคุมจำนวนหลายตัว กับ DTE (Data Terminal Equipment) หลายตัวบนเครือข่ายเดียวกัน และสามารถแยกย่อยได้ ดังนี้

1. กำหนดโปรโตคอลการรับส่ง
2. กำหนดคุณสมบัติทางฮาร์ดแวร์ของการรับส่ง
3. สร้างโปรแกรม และ การ์ดบนคอมพิวเตอร์ เพื่อจำลองเป็นตัวควบคุมหลัก
4. สร้างโมดูลควบคุมการเปิดปิดเครื่องใช้ไฟฟ้า
5. สร้างโมดูลควบคุมย่อย เพื่อจำลองตัวเป็นตัวควบคุมย่อย

บทที่ 1

S-ARTnet และ RS-485

1.1 S-ARTnet -- A Powerful Controller Network

NETWORK CONTROLLER

ระบบ ARTnet ใช้ IC เบอร์ 8031 กับ address latch และ EPROM ใช้สายส่ง แบบ serial RS-232 ซึ่งใช้ขา GND , TXD , RXD ในระบบ network นี้ มีอุปกรณ์ sends point (DTE) ได้ถึง 60 ตัว และมี control point (DCE) ได้ถึง 60 ตัวเช่นเดียวกัน

ระบบ power ใช้ 15 v ซึ่งแปลงจาก 5 v โดยใช้ LM2577-15.0 switch mode regulator ดังแสดงในรูป 2a ในส่วนของ cable interface แสดงในรูป 2b มีการใช้ TR 4 ตัว เพื่อแปลงสัญญาณ จาก 0 , 5 v ไปเป็น 0 , 7.5 , 15 v เพื่อใช้ในการส่ง โดย Q_3 เป็น สวิทซ์ 15 v Q_4 เป็นสวิทซ์ 0 v และถ้า Q_3 และ Q_4 ไม่นำกระแสทั้งคู่ก็จะได้ 7.5 v จาก R_4 และ R_5 (Voltage Device) ส่วนในการรับสัญญาณจะใช้ LM 311 comparator เพื่อปรับสัญญาณให้เป็น 0 และ 5 v

S-ART SATELLITES

สัญญาณ 15 v แสดงถึง clock ส่วน 0 และ 7.5 v เป็น data โดย 0 v หมายถึง '0' และ 7.5 v หมายถึง '1'

ส่วน word ที่ใช้ส่งจากตัว controller มี 3 รูปแบบ คือ SYNC , READ , WRITE

สัญญาณ SYNC ประกอบด้วย '1' จำนวน 8 บิท หรือมากกว่า ตามด้วย '0' 1 บิท

สัญญาณ READ ใช้สำหรับตรวจสอบสถานะของ terminal ส่วนสัญญาณ WRITE ใช้สำหรับการเปลี่ยนสถานะของ terminal

ลักษณะสัญญาณ READ และ WRITE แสดงดังรูป 5b และ 5c ซึ่งมีส่วนที่ต่างกัน คือ ในบิทที่ 6 '0' หมายถึง 'READ' และ '1' หมายถึง 'WRITE' ส่วน output ของ Terminal จะเอาไปขับ LED หรือ solid-state relay

1.2 RS-485

ผู้ที่ออกแบบระบบนี้ได้นำ RS-485 มาใช้กับการต่อ PC กับ embedded controller หลายตัว และได้ออกแบบระบบ network ซึ่งมีคุณสมบัติ ดังนี้

- ใช้งานได้ในสภาพแวดล้อมแบบโรงงานอุตสาหกรรม
- ใช้ทรัพยากรจาก embedded controller น้อย (ทั้ง CPU time และ memory)
- แต่ละ node มีราคาต่ำ
- ปรับไปใช้ได้กับ microcontroller ที่นิยมใช้กัน
- สามารถปรับปรุงหรือเปลี่ยนแปลง hardware และ software ที่ใช้ควบคุมได้ และได้เรียกชื่อ network protocal นี้ว่า tiny-NSP (Nine-bit Serial Protocal)

NETWORK HARDWARE - THE PHYSICAL LAYER

ที่ใช้ shielded twisted-pair RS-485 เพราะว่าราคาถูกและใช้ง่าย และมี data requirement ต่ำ และยังป้องกัน noise ได้ดีด้วย

RS-485 สามารถใช้ต่อกับอุปกรณ์ได้เต็มที่ คือ 32 transmitters และ 32 recievers โดยใช้สาย 1 คู่ และในเวลาหนึ่ง transmitter จะ active ได้แค่ตัวเดียวเท่า นั้น ตัว embedded controller จะเชื่อมต่อกับ twisted pair ด้วย RS-485 transciever (transmitter and reciever) เช่น 75176

PROTOCAL FRAME AND MASSAGE

การส่งใช้แบบ asynchronous serial communication การออกแบบรูปแบบของข้อความ (frame and massage format) ที่ดี จะช่วยลดเวลาในการติดต่อระหว่าง embedded controller กับ network ลงได้อย่างมาก

frame ที 2 ชนิด คือ address frame และ data frame ซึ่งจะแยกให้เห็นความ แตกต่างโดยใช้บิทที่ 9 ของข้อมูล โดยเป็น '1' สำหรับ address frame และ '0' สำหรับ data frame

address frame คือ address ของ node ที่จะต้องรับข้อมูล

data frame ประกอบด้วย command , status imformation , error control และ data อื่นที่ต้องการใช้

ในการรับข้อมูลของแต่ละ node นั้น จะรับข้อมูลที่เป็น address เข้ามาตรวจสอบว่าตรงกับ address ของตนเองหรือไม่ ถ้าใช่ก็จะทำการรับข้อมูลที่เหลือเข้ามา แต่ถ้าไม่ใช่ก็จะไม่สนใจข้อมูลที่ตามมา

NETWORK ACCESS CONTROL

การใช้ RS-485 ทำให้มีเพียง node เดียวเท่านั้น ที่ทำการส่งข้อมูลได้ในขณะใดขณะหนึ่ง แต่เราต้องการให้ทุก node มีโอกาสส่งข้อมูลได้เหมือนๆกัน ดังนั้นจึงใช้วิธีการ simple polled master-slave โดยที่ master จะทำการ poll slave แต่ละตัวแบบวนรอบ (periodically) เพื่อรับข้อมูลสถานะ , เก็บข้อมูล และ ส่งคำสั่งควบคุม ซึ่ง slave ก็จะมีหน้าที่ตอบสนองคำสั่งของ master ดังนั้นจึงเป็น protocal แบบ order/reply massage protocal

polled protocal มีข้อเสีย คือ การที่ slave จะส่งข้อมูลไปยัง master นั้น ต้องรอการ poll จาก master ก่อนทำให้ข้อมูลสำคัญที่ต้องการส่งไปถึง master อย่างเร่งด่วนนั้น เกิดการล่าช้าเกินไป ใน prototype network (ต้นแบบ) ใช้ PC เป็น master และ ใช้ embedded controller เป็น slave

MESSAGE STRUCTURE

การออกแบบโครงสร่างข้อมูลที่ดีจะต้องมีขนาดของข้อมูลน้อยแต่มีประสิทธิภาพ และ ป้องกัน ความผิดพลาดในการส่งได้ตามปกติจะให้master เป็น address 0 , slave address 1-244 และ 255 สำหรับ slave ทั้งหมด

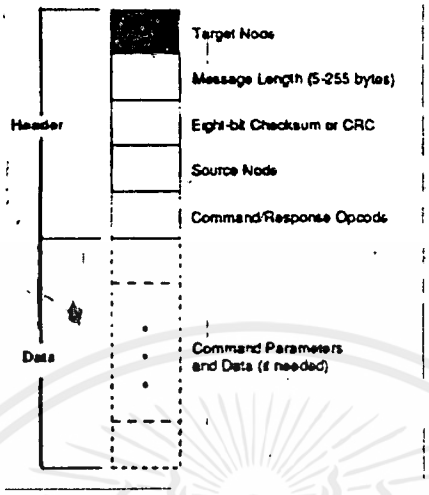
การส่งข้อมูลสามารถส่งได้ตั้งแต่ 1 ถึง 255 ไบท์ ดังนั้น แต่ละ node จะต้องมี RAM สำหรับเก็บข้อมูลเหล่านี้ด้วย

ERROR CHECKING

ใช้ eight-bit check sum หรือ CRC ซึ่งตรวจสอบได้ทั้ง single error และ multiple error

MESSAGE

ชุดของข้อมูลแสดงดัง รูปที่ 1.1



เมื่อ master ส่งข้อมูลไปแล้วจะต้องได้รับข้อมูลกลับมาเสมอ ยกเว้นข้อมูลที่ส่งไปยัง slave ทุกตัวพร้อมกัน อย่างน้อยที่สุด slave จะต้องตอบสนองต่อคำสั่ง 'status query' และ 'reset network'ได้ ถ้าหาก slave ได้รับคำสั่งที่ไม่ทราบความหมาย ก็จะต้องตอบกลับไปว่า 'command not supported' slave จะตอบสนองต่อคำสั่งเฉพาะบางชนิดเท่านั้น ถ้าหาก master พยายามจะละเมิดกฎ slave ก็ต้องตอบกลับไปว่า 'access violation'

ERROR RECOVERY

ทุก ๆ network protocal จะต้องมี procedure ที่จัดการกับ error ซึ่งได้แก่

- slave เป้าหมาย ไม่สามารถทำงานตามคำสั่งได้
- slave เป้าหมาย ทำงานยุงจนไม่สามารถตอบสนองต่อคำสั่งได้
- slave เป้าหมาย ไม่ได้ต่ออยู่กับ network
- hardware ทำงานผิดพลาด
- เกิด noise ขึ้น ทำให้ข้อมูลผิดพลาด (terminal ทุกตัวรวมทั้ง master)

ใน protocal นี้ การแก้ไข error เป็นหน้าที่ของ master เพื่อลดการใช้ทรัพยากรของ slave

เงื่อนไขความผิดพลาด (error condition) 4 ประเภทที่ master ต้องทำการตรวจสอบ มีดังนี้

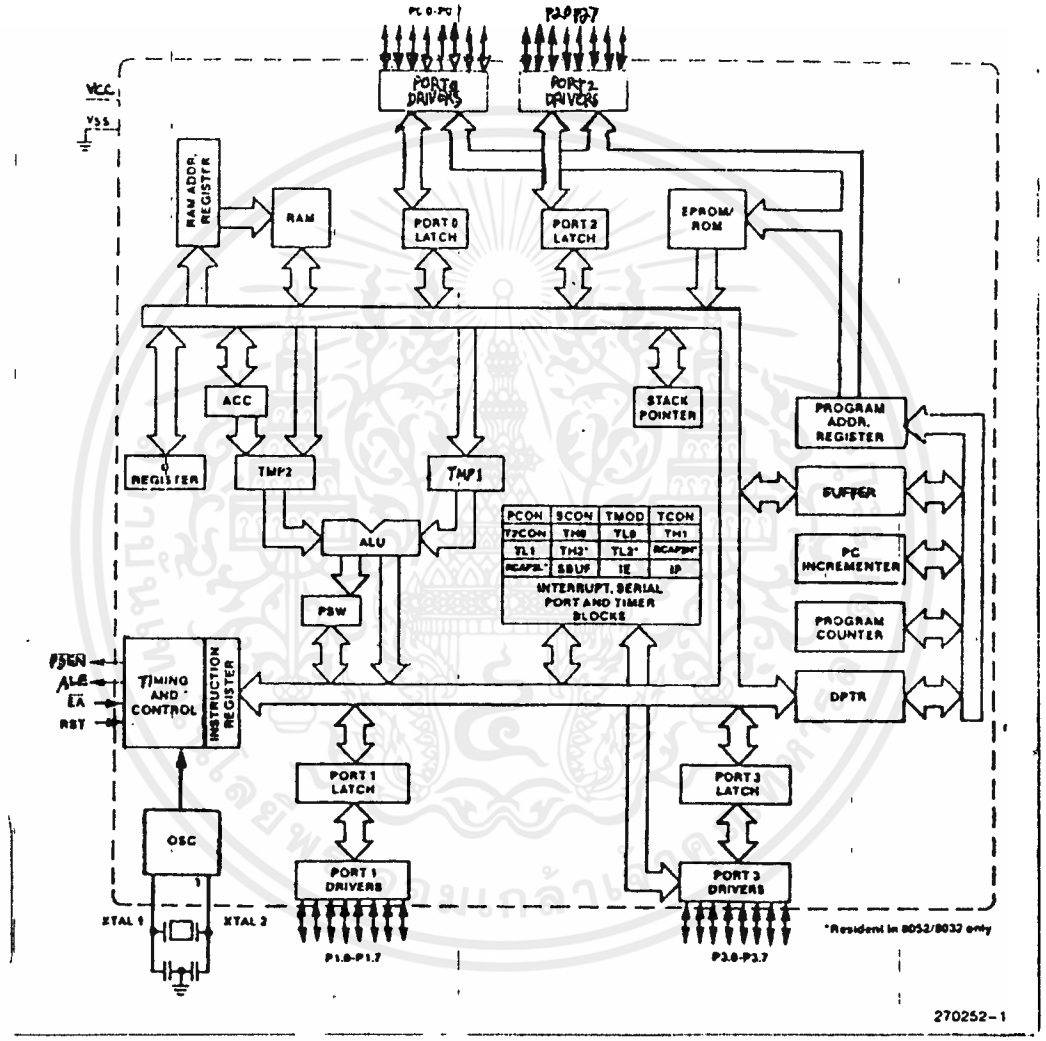
- ผลตอบสนองที่ถูกต้องของ slave ชี้ให้เห็นว่า slave ไม่สามารถปฏิบัติตามคำสั่งได้
- ผลตอบสนองของ slave ไม่ถูกต้อง
- slave ไม่ตอบสนองข้อมูลที่ส่งไปในช่วงเวลาที่กำหนด
- slave ทำการส่งข้อมูลโดยที่ยังไม่มีการ poll

เมื่อเงื่อนไขแรกเกิดขึ้น master จะต้องแจ้งให้ผู้ใช้ทราบ ถ้าหากเป็นเงื่อนไขอื่นให้ master ทำการส่งข้อมูลนั้นใหม่เป็นจำนวนครั้งที่ระบุไว้ ถ้าหากผลตอบสนองยังผิดพลาดอีก ก็ให้ลบ node นั้นออกจาก list of active slave ซึ่งจะต้องแจ้งให้ผู้ใช้ทราบด้วย ถ้าหากเกิดความผิดพลาดนอกเหนือจากนี้ ส่วนของ reset procedure จะต้องทำงานทันที

สถาปัตยกรรมของ 8051

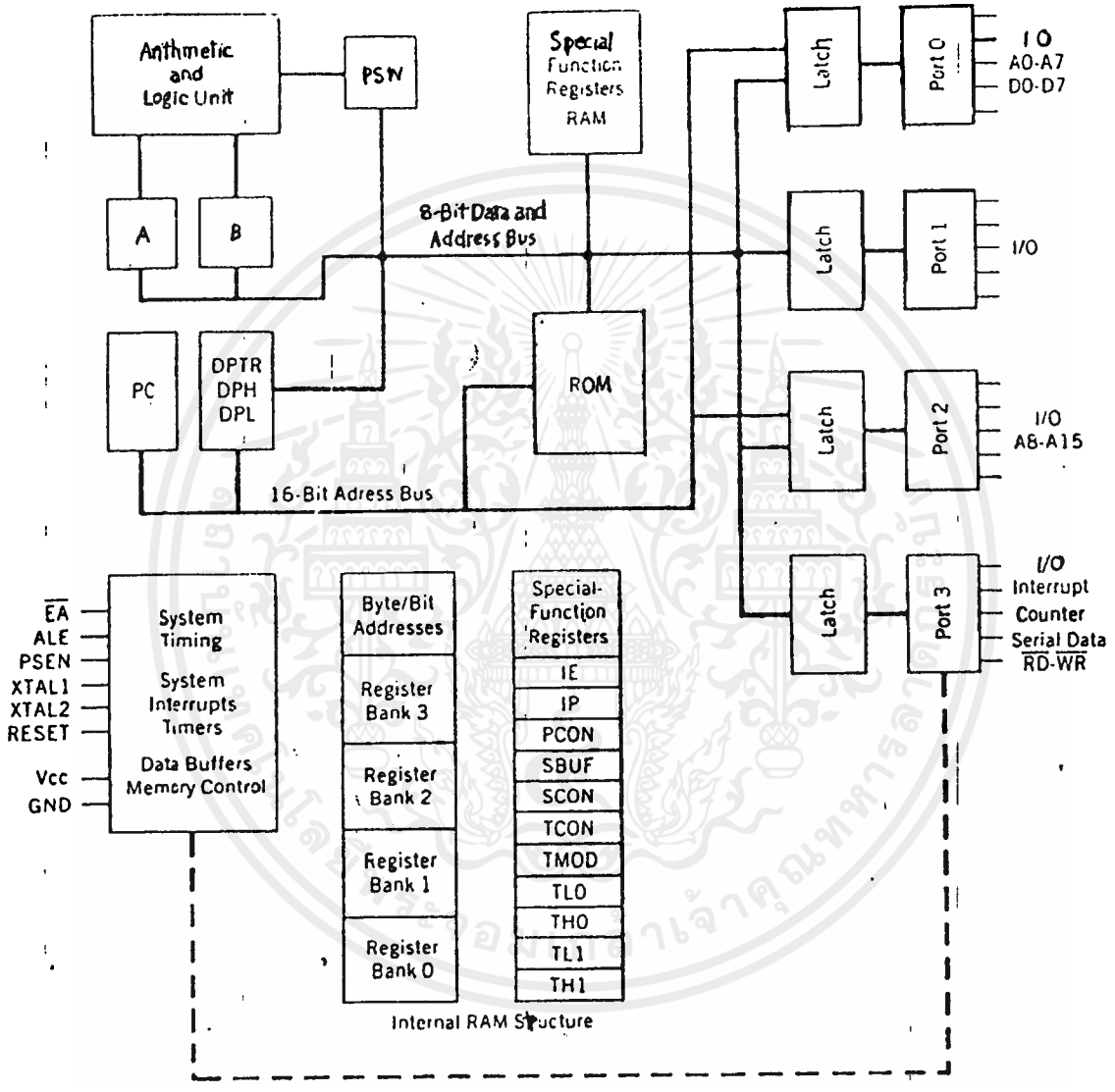
สถาปัตยกรรมภายในของ 8051 จะเป็นดังรูปที่ 2.1 ซึ่งประกอบไปด้วยส่วนสำคัญหลักๆ ดังนี้

- เป็น CPU ขนาด 8 บิท ประกอบด้วยรีจิสเตอร์ A (แอคคิวมูเลเตอร์) และรีจิสเตอร์ B
- มีโปรแกรมเคาเตอร์ (Program counter,PC) และ ดาต้าพอยท์เตอร์ (Datapointer ,DPTR)
- มี Programm status word ขนาด 8 บิท
- มีหน่วยความจำรอม(ROM) หรือ อีพรอม(EPROM เฉพาะ 8751) ขนาด 0 กิโลไบท์(8031) ถึง 4 กิโลไบท์ (8051)
- มีหน่วยความจำแรมภายใน ขนาด 128 ไบท์ ประกอบด้วย
  1) รีจิสเตอร์แบงค์ 4 แบงค์ แต่ละแบงค์ประกอบด้วยรีจิสเตอร์ 8 รีจิสเตอร์ (R0 - R7)
  2) มีหน่วยความจำจำนวน 16 ไบท์ ที่สามารถอ้างแอดเดรสเพื่อควบคุมการทำงานในระดับบิทได้
  3) มีหน่วยความจำสำหรับใช้งานทั่วไป 80 ไบท์
- มีขารับสัญญาณ อินพุท/เอาท์พุท 32 ขา แบ่งออกเป็นกลุ่มๆละ 8 บิท ได้สี่กลุ่ม คือ P0 , P1 ,P2 , P3
- มีไทเมอร์/เคาท์เตอร์ ขนาด 16 บิท สองชุด คือ T0 และ T1
- มีพอร์ทอนุกรมที่ใช้รับส่งสัญญาณแบบฟูลดูเพลก (Full duplex) เรียกว่า SBUF
- มีรีจิสเตอร์ควบคุม ได้แก่ TCON , TMOD , SCON , PCON , IP และ IE
- สามารถทำการอินเทอร์รัพต์ได้ทั้งภายในและภายนอก การอินเทอร์รัพต์ภายในได้มา จากแหล่งกำเนิดอินเทอร์รัพต์สามแหล่ง การอินเทอร์รัพต์ภายนอกได้มาจากแหล่งกำเนิดการอินเทอร์รัพต์ภายนอกสองแหล่ง
- มีส่วนของออสซิเลเตอร์ และ วงจรสร้างสัญญาณนาฬิกาอยู่ภายใน



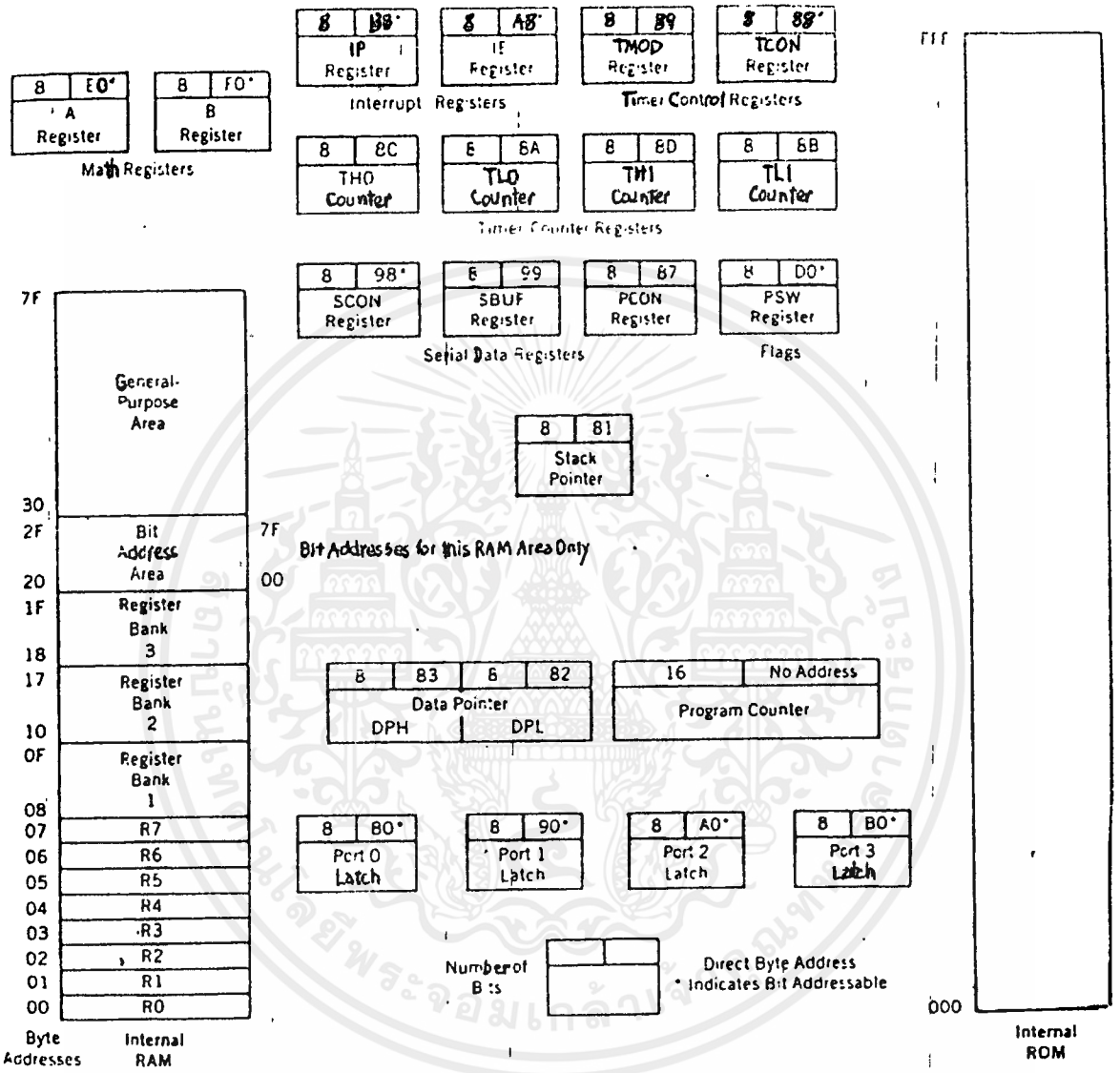
รูปที่ 2.1 แสดงสถาปัตยกรรมภายในของ 8051

จากรูปที่ 2.1 จะเขียนเฉพาะส่วนสำคัญๆ เป็นบล็อกไดอะแกรมได้ดังรูป 2.2



2.2 แสดงบล็อกไดอะแกรมภายในของ 8051

การแม็บหน่วยความจำภายในและรีจิสเตอร์พิเศษ (Special functoin register, SFR) แสดงได้ ดังรูปที่ 2.3

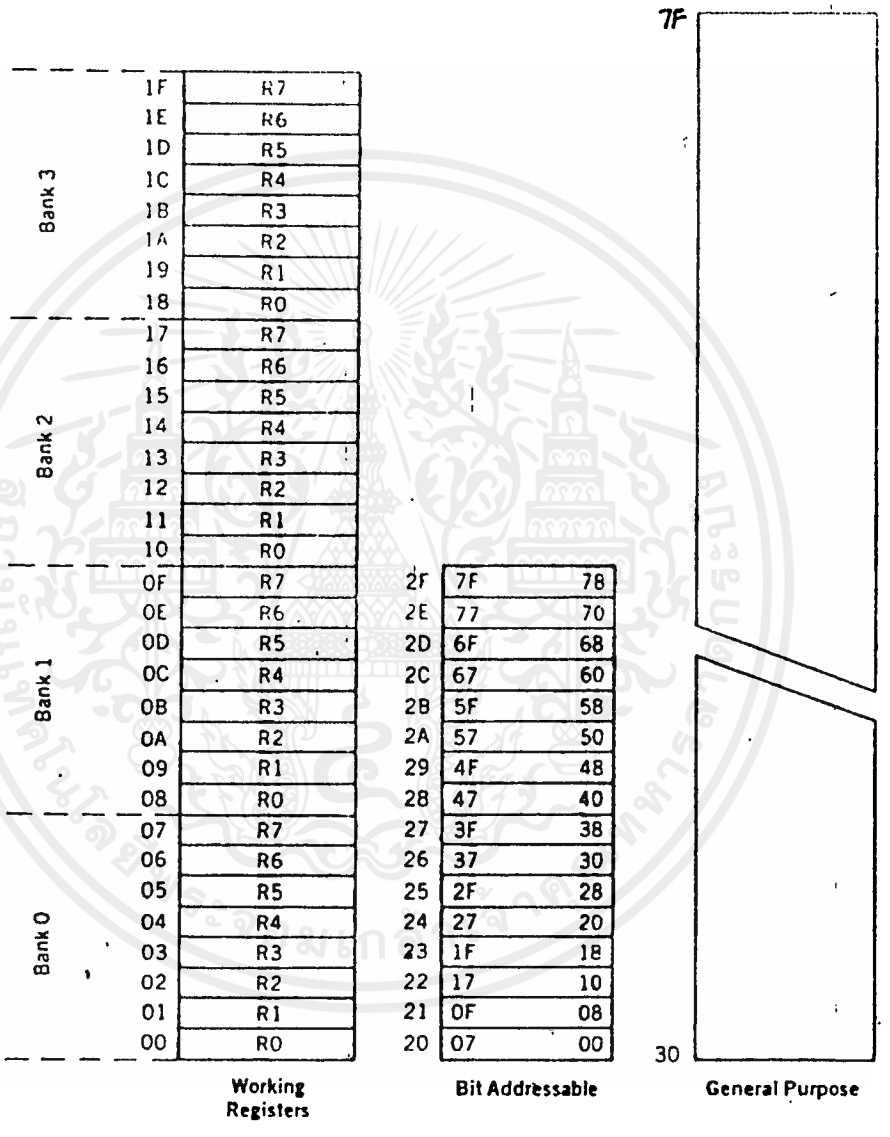


รูปที่ 2.3 แสดงการแม็บหน่วยความจำภายในของ 8051

จากรูปที่ 2.3 จะเห็นว่า 8051 มีทั้งหน่วยความจำแรมและรอมอยู่ภายในตัว ในกรณีที่หน่วยความจำภายในไม่เพียงพอ ก็ยังสามารถต่อหน่วยความจำภายนอกขึ้นมาใช้งานได้อีก

2.1 หน่วยความจำภายใน

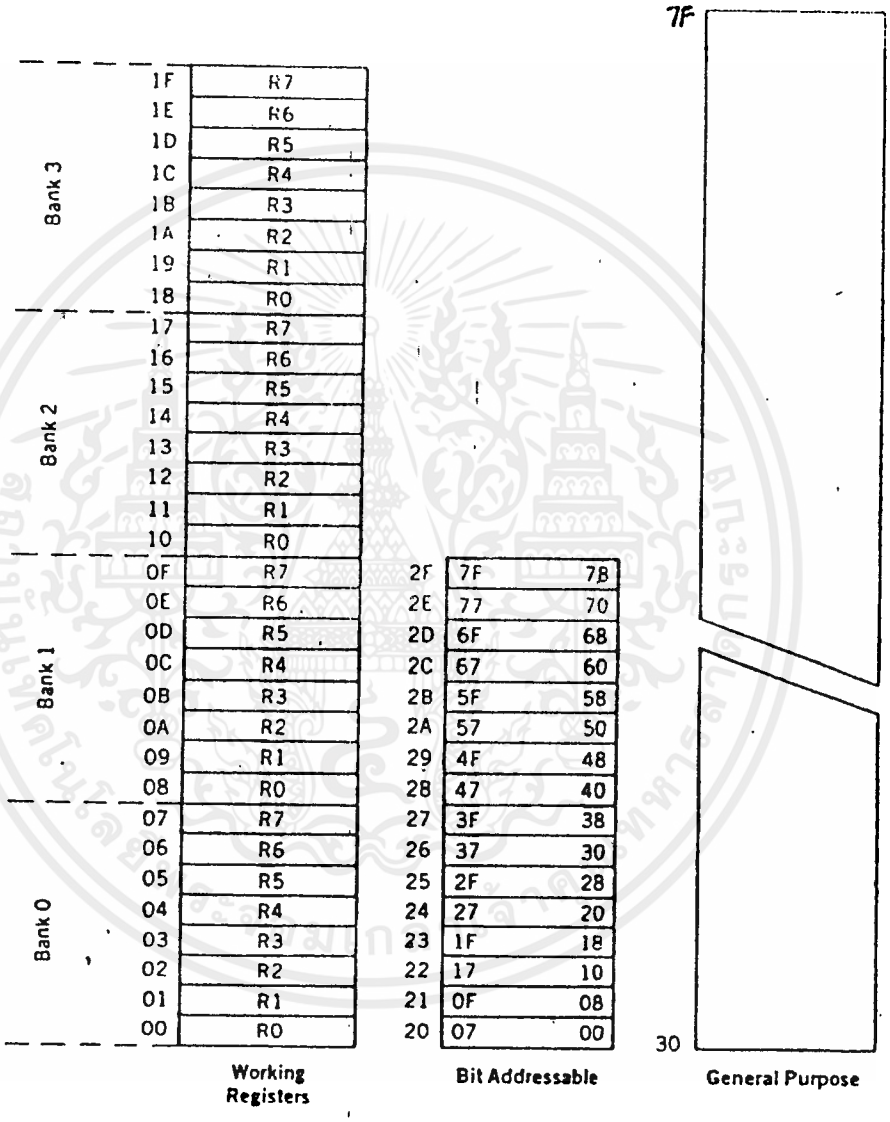
หน่วยความจำแรมภายในตัว 8051 จะมีขนาด 128 ไบท์ แยกการใช้งานออกเป็นสามส่วน ดังรูปที่ 2.4



รูปที่ 2.4 แสดงหน่วยความจำแรมภายในตัว 8051

2.1 หน่วยความจำภายใน

หน่วยความจำแรมภายในตัว 8051 จะมีขนาด 128 ไบท์ แยกการใช้งานออกเป็นสามส่วน ดังรูปที่ 2.4



รูปที่ 2.4 แสดงหน่วยความจำแรมภายในตัว 8051

- หน่วยความจำแรมตั้งแต่ แอดเดรส 00 ถึง 1F จำนวน 32 ไบท์แรกถูกใช้งานในลักษณะของรีจิสเตอร์ทั่วไป โดยแยกออกเป็น 4 แบงค์ (แบงค์ 0 ถึง แบงค์ 3) แต่ละ แบงค์ประกอบด้วยรีจิสเตอร์ 8 รีจิสเตอร์ คือ รีจิเตอร์ R0 ถึง รีจิสเตอร์ R7

ลักษณะการใช้งานจะทำได้สองแบบ คือ

1) มีการใช้งานแบบเป็นรีจิสเตอร์ เช่น

ถ้าเลือกแบงค์ 0 คำสั่ง MOV R0,#0BH หมายถึงให้เอาค่า 0BH ไปเก็บไว้ในรีจิสเตอร์ R0 (ขณะนี้ คือ ตำแหน่งแอดเดรส 00)

ถ้าเลือกแบงค์ 1 คำสั่ง MOV R0,#0BH หมายถึงให้เอาค่า 0BH ไปเก็บไว้ในรีจิสเตอร์ R0 (ขณะนี้ คือ ตำแหน่งแอดเดรส 08)

ถ้าเลือกแบงค์ 2 คำสั่ง MOV R0,#0BH หมายถึงให้เอาค่า 0BH ไปเก็บไว้ในรีจิสเตอร์ R0 (ขณะนี้ คือ ตำแหน่งแอดเดรส 10H)

การเลือกใช้รีจิสเตอร์แบงค์ได้ จะทำได้โดยกำหนดบิทเซ็ต (เป็น 1) หรือเคลียร์ (เป็น 0) ในบิท RS0 และ RS1 ในรีจิสเตอร์ PSW ข้อกำหนดจะเป็น ดังนี้

RS1	RS0	
0	0	เลือกรีจิสเตอร์แบงค์ 0
0	1	เลือกรีจิสเตอร์แบงค์ 1
1	0	เลือกรีจิสเตอร์แบงค์ 2
1	1	เลือกรีจิสเตอร์แบงค์ 3

2) การเรียกใช้ในลักษณะของหน่วยความจำแรมโดยตรง เช่น

```
MOV  00,#0BH   ; นำค่า 0BH ไปเก็บไว้ยังหน่วยความจำแรม
MOV  08,#0BH   ; ที่ตำแหน่งแอดเดรส 00 , 08 และ 10H
MOV  10H,#0BH  ;
```

จากที่กล่าวมาแล้วข้างต้น จะเห็นว่า รีจีสเตอร์แบงค์ใดที่ไม่ถูกเลือกใช้งานในลักษณะของรีจิสเตอร์แบงค์แล้ว ยังสามารถเลือกใช้งานได้เป็นหน่วยความจำแรมทั่วไป

- หน่วยความจำแรมตั้งแต่แอดเดรส 20H ถึง 2FH จะเป็นหน่วยความจำที่สามารถอ้างอิงแต่ละบิท ออกมาเป็นตำแหน่งแอดเดรสได้ (Bit addressable) จากรูปจะเห็นว่าถ้า อ้างเป็นไบท์ จะอ้างตำแหน่งแอดเดรสได้ 16 แอดเดรส คือ 20H ถึง 2FH แต่ถ้าจะอ้างตำ แหน่งในลักษณะของแต่ละบิทในแต่ละแอดเดรส เช่น บิท 0 ของแอดเดรสที่ 20H เพื่อนำมาใช้ งาน เฉพาะบิทที่อ้างถึงเท่านั้น การอ้างตำแหน่งแบบนี้ เรียกว่า บิทแอดเดรส ซึ่งมีค่าเรียงกัน ไปจาก บิทแอดเรส 00 (บิท 0 ของแอดเดรส 20H) เรียงไปจนถึง 7FH (บิท7 ของแอดเดรส 2FH) เช่น

บิทแอดเดรส 4FH หมายถึง บิท 7 ของไบท์แอดเดรส 29H

บิทแอดเดรส 70H หมายถึง บิท 0 ของไบท์แอดเดรส 2EH

	7	6	5	4	3	2	1	0 bit
ไบท์แอดเดรส	29H	4FH	4EH	4DH	4CH	4BH	4AH	49H	48H
ไบท์แอดเดรส	2EH	77H	76H	75H	74H	73H	72H	71H	70H

<----------บิทแอดเดรส----------->

- หน่วยความจำแรมตั้งแต่ แอดเดรส 30H ถึง 7FH จะเป็นหน่วยความจำที่ใช้งานทั่วไป และ มีการอ้างแอดเดรสแบบไบท์ได้อย่างเดียว

2.2 รีจิสเตอร์ใช้งานหน้าที่พิเศษ (Special Function Register , SFR)

หน่วยความจำแรมภายในตั้งแต่แอดเดรส 80H ถึง FF จะเป็นรีจิสเตอร์ที่ใช้งานเฉพาะหน้าที่ เช่น ทำหน้าที่เป็นรีจิสเตอร์ B เป็นต้น รีจิสเตอร์เหล่านี้ เรียกว่า Special function register หรือ เรียกสั้นๆว่า SFR การกำหนดชื่อและตำแหน่งแอดเดรสจะเป็น ดังรูปที่ 2.5

	8 Bytes								
F8									FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8	(T2CON)		(RCAP2L)	(RCAF2H)	(TL2)	(TH2)			CF
C0									C7
B8	IP								BF
B0	P3								B7
A8	IE								AF
A0	P2								A7
98	SCON	SBUF							9F
90	P1								97
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
80	P0	SP	DPL	DPH				PCON	87

รูปที่ 2.5 แสดงการแม็บของ SFR

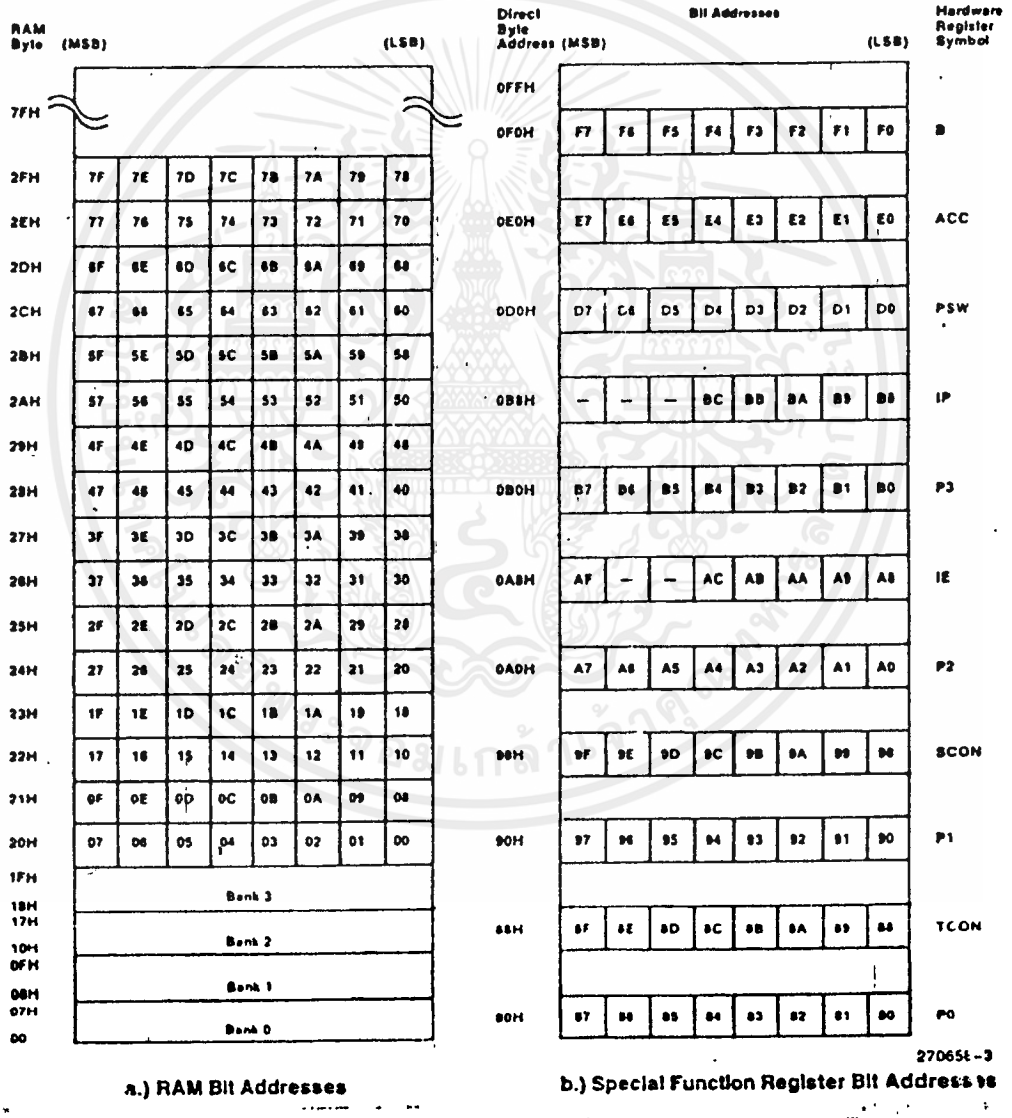
จากรูปที่ 2.5 จะเห็นว่าบางแอดเดรสก็จะว่างเอาไว้ ทางโรงงานผลิตจะสงวนไว้ใช้งานในภายภาคหน้า ที่ว่างเหล่านี้เมื่ออ่านข้อมูลออกมาจะได้ค่าที่ไม่แน่นอน (Random data) ถ้าลองเขียนข้อมูลเข้าไปผลที่ได้ก็จะไม่ถูกต้อง

รีจิสเตอร์บางตัวของ SFR ก็สามารถอ้างแอดเดรสแบบบิทแอดเดรสได้เช่นเดียวกัน แต่บางตัวก็อ้างไม่ได้ ดังรูปที่ 2.6

รายละเอียดของ SFR แต่ละรีจิสเตอร์เป็น ดังนี้

- แอคคิวมูเลเตอร์ (Accumulator , ACC)
- รีจิสเตอร์ B มักจะถูกใช้ในกรณีที่ทำการคูณ หาร เป็นส่วนมาก
- โปรแกรมสเตัสเวิร์ด(Programm status word , PSW) ซึ่งจะทำหน้าที่สองประการ คือ ใช้เป็นแฟลกเพื่อเก็บผลลัพธ์ของคำสั่งที่ได้กระทำเสร็จสิ้นไปแล้ว หรือ ใช้เป็น ข้อกำหนดในการทดสอบเงื่อนไขต่างๆ 8051 จะมีแฟลกเกี่ยวข้อง

ทางคณิตศาสตร์อยู่สี่แฟลก คือ Carry flag (C) , Auxilary carry flag (AC) , Overflow flag (OV) , Parity flag (P) นอกจากนั้นจะเป็นที่ผู้ใช้กำหนดใช้งานได้ตลอดเวลา(General purpose flag) อีกสามแฟลก คือ F0 flag , GF) flag , GF1 flag GF0 และ GF1 จะอยู่ใน PCON รีจิสเตอร์ ข้อแตกต่างระหว่างแฟลกที่ผู้ใช้กำหนดเอง และแฟลกที่เกี่ยวข้องกับทางคณิตศาสตร์คือ แฟลกที่ผู้ใช้กำหนดเองนั้นผู้ใช้สามารถเซ็ตหรือเคลียร์ได้บางแฟลก แต่เมื่อมีการใช้ คำสั่งเกี่ยวกับคณิตศาสตร์แล้ว แฟลกจะเป็นไปตามเงื่อนไขของคำสั่งนั้นๆ



รูปที่ 2.6 แสดงบิทแอดเดรสแม็บทั้งของ RAM และ SFR

รายละเอียดของโปรแกรมสเตตัสเวิร์ด จะเป็นดังนี้

7	6	5	4	3	2	1	0	bit
CY	AC	FO	RS1	RSO	OV	-	P	

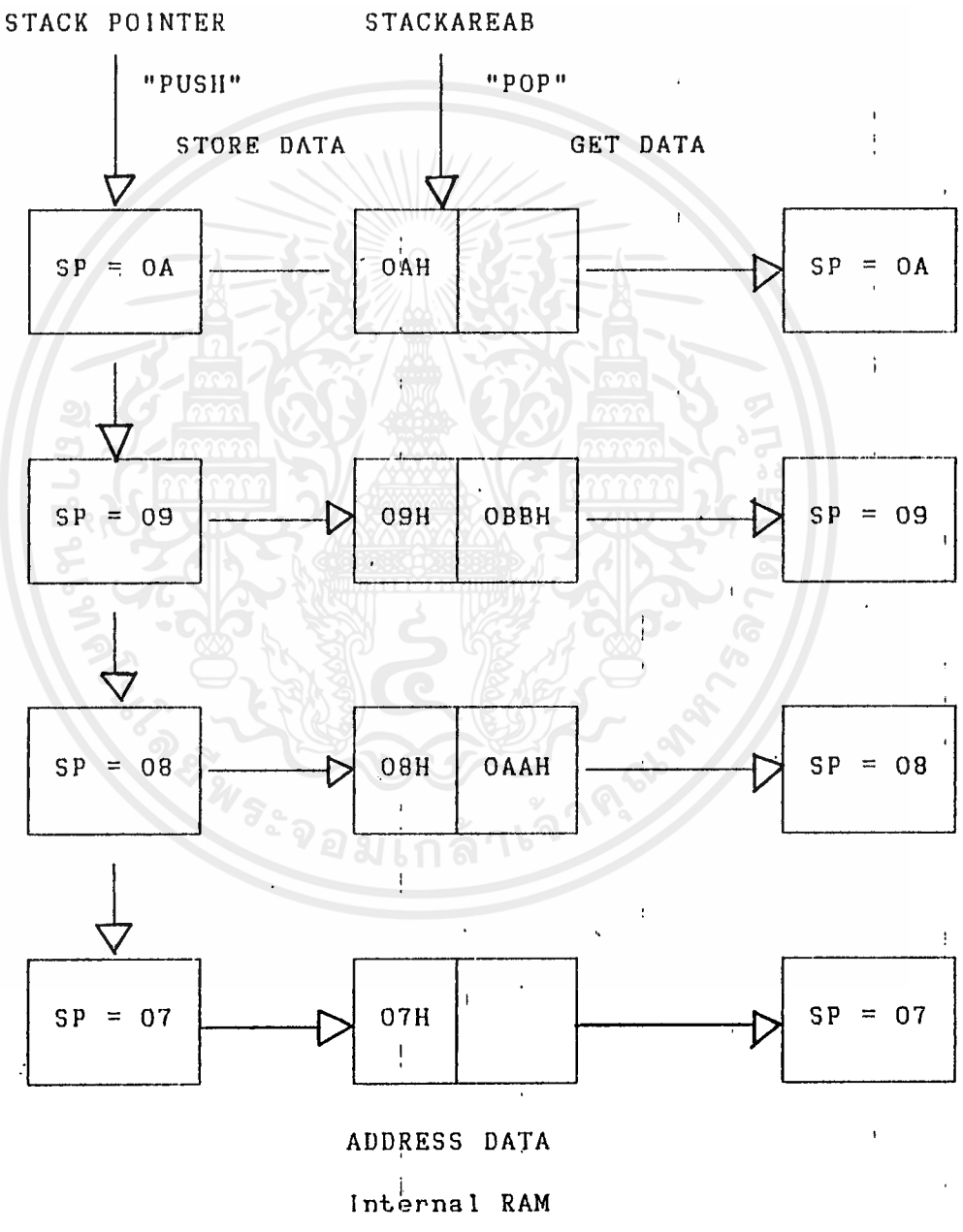
บิท	สัญญลักษณ์	รายละเอียด
7	CY	แคร์รี่แฟลก มักใช้ในคำสั่งทางคณิตศาสตร์ , JUMP , ROTATE และ BOOLEAN
6	AC	ออกซิเลียรี่แฟลกถูกใช้ในคำสั่งทางคณิตศาสตร์เกี่ยวกับ BCD
5	FO	แฟลกที่ผู้ใช้กำหนดได้ตามต้องการ
4	RS1	ใช้เลือกรีจิสเตอร์แบงค์ บิท 1
3	RSO	ใช้เลือกรีจิสเตอร์แบงค์ บิท 0
		RS1 RSO 0 0 เลือกรีจิสเตอร์แบงค์ 0 0 1 เลือกรีจิสเตอร์แบงค์ 1 1 0 เลือกรีจิสเตอร์แบงค์ 2 1 1 เลือกรีจิสเตอร์แบงค์ 3
2	OV	โอเวอร์โฟล์วแฟลก ใช้ในคำสั่งทางคณิตศาสตร์
1	-	สงวนไว้
0	P	พาริตี้แฟลก จะเป็น 1 หมายถึง พาริตี้คี่ เมื่อรีจิสเตอร์ A มีบิทที่เป็น 1 เป็นเลขคี่

- แสต็กพอยเตอร์ (Stack pointer)

แสต็ก หมายถึง พื้นที่ในหน่วยความจำแรมภายใน ที่ใช้เก็บข้อมูลที่แน่นอนได้ไวและขณะเดียวกันก็เรียกข้อมูลกลับคืนได้ไวด้วยแสต็กพอยเตอร์ของ 8051 จะมีขนาด 8 บิท ใช้เก็บตำแหน่งแอดเดรสของหน่วยความจำแรมเอาไว้ ที่ตำแหน่งนี้

เรียกตำแหน่งบนสุดของแสต็ก (Top of stack)

เมื่อข้อมูลถูกใส่ลงไปในแสต็ก ตัวแสต็กพอยเตอร์จะเพิ่มค่าของตัวมันเองขึ้น 1 ไบท์ แล้วจึงเก็บข้อมูลลงไป การใส่ข้อมูลลงในแสต็ก เรียกว่า Push ข้อมูลลงไปในแสต็ก ใน ทางกลับกัน เมื่อต้องการดึงข้อมูลออกจากแสต็ก ซึ่งเรียกว่า การ POP ข้อมูลจากแสต็ก หลังจาก ที่ดึงข้อมูลออกไปแล้ว แสต็กจะลดค่าตัวมันเองลง 1 ไบท์ ดังรูปที่ 2.7



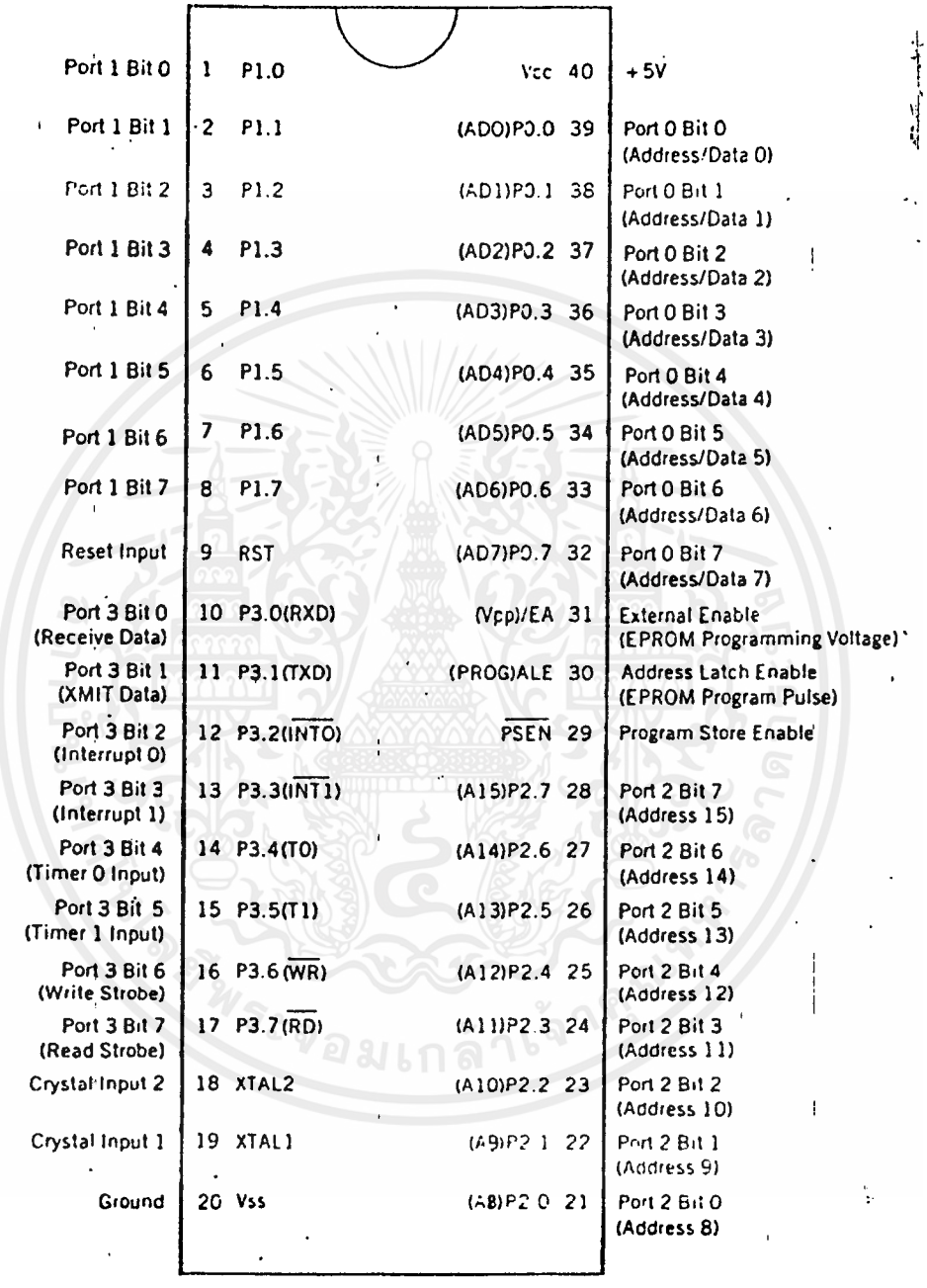
รูปที่ 2.7 การทำงานของแสต็ก และ แสต็กพอยเตอร์

- Data pointer (DPTR) จะเป็นรีจิสเตอร์สองตัวประกอบกัน ได้แก่ ดาต้าพอยเตอร์ไบท์สูง (DPH) และ ดาต้าพอยเตอร์ไบท์ต่ำ (DPL) วัตถุประสงค์หลักของการใช้งาน คือ เอาไว้ใช้เก็บค่าแอดเดรสขนาด 16 บิท เนื่องจากดาต้าพอยเตอร์ประกอบด้วยรีจิสเตอร์ขนาด 8 บิทสองรีจิสเตอร์ ฉะนั้นการใช้งานจะเป็นรีจิสเตอร์ ขนาด 16 บิทเลย หรือ แยกกันใช้เป็นรีจิสเตอร์ขนาด 8 บิท สองตัวที่ไม่ขึ้นแก่กันก็ได้

ในสภาวะเริ่มให้แสต็กพอยเตอร์ มีค่าเป็น 07 ซึ่งหมายถึงตำแหน่งแอดเดรสของแรมภายในที่แอดเดรส 07 เมื่อใช้คำสั่ง เช่น PUSH OAAH ซึ่งหมายถึง ให้เก็บข้อมูล OAAH ลงไปในแสต็ก แสต็กพอยเตอร์จะเพิ่มค่าไปเป็น 08 ก่อนที่จะเอาข้อมูลไปเก็บ ฉะนั้น ในขณะนี้ ตัวแสต็กจะมีตำแหน่งแอดเดรสเป็น 08 ข้อมูล OAAH จะถูกเก็บไว้ที่แรมตำแหน่งนี้ ต่อไปถ้าพบคำสั่ง PUSH อีก เช่น PUSH OBBH แสต็กพอยเตอร์จจะเพิ่มค่าเป็น 09 ก่อนแล้วจึงเอาข้อมูล OBBH ไปเก็บไว้ยังบริเวณของแสต็กของแรมที่ตำแหน่งแอดเดรส 09 ต่อมาเมื่อ POP แสต็กจะเอาข้อมูลที่ใส่เข้าไปครั้งหลังสุดออกมาก่อน แล้วแสต็กพอยเตอร์จะลดค่าลงมาเป็น 08

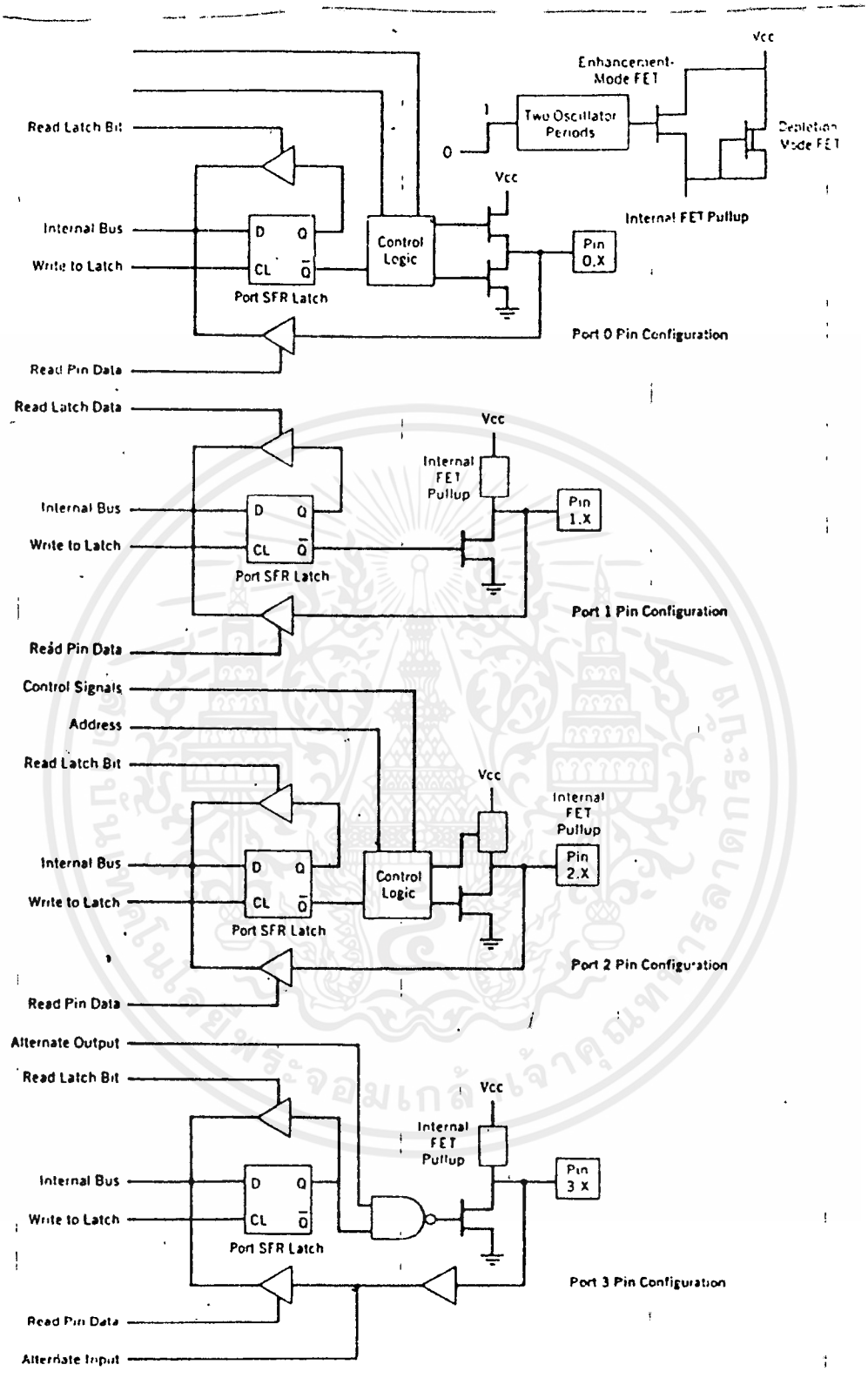
2.3 อินพุท/เอาท์พุทพอร์ท

อินพุท/เอาท์พุทพอร์ทของ 8051 จะมีการจัดวางเรียงกัน ดังรูปที่ 2.8



รูปที่ 2.8 แสดงพอร์ทของ 8051 เรียงตามขาต่างๆ

เพื่อให้ใช้งานได้สะดวกขึ้น 8051 จึงออกแบบให้แต่ละขาที่ใช้งานเป็นอินพุท/เอาท์พุท ยังสามารถใช้ขยายหน่วยความจำเพื่อเพิ่มประสิทธิภาพให้ระบบ หรือ ใช้เป็นพอร์ทอนุกรมก็ได้ แต่ละขาของ 8051 จะมีวงจรภายใน ดังรูปที่ 2.9



รูปที่ 2.9 แสดงวงจรภายในแต่ละขาของ 8051

แต่ละขาพอร์ทจะมีดาต้า ฟลิบฟลอบทำหน้าที่แลทช์ ข้อมูลก่อนจะส่งออกมาภายนอกขาของไอซีแต่ละขา การควบคุมกต่ละพอร์ท จะควบคุมผ่านรีจิสเตอร์ P0, P1 , P2 และ P3 ซึ่ง เป็นรีจิสเตอร์ใน SFR แต่ละพอร์ทประกอบด้วยอินพุท/เอาท์พุท 8 เส้น (หรือมีดาต้าฟลิบฟลอบ 8 ตัว ต่อหนึ่งพอร์ท)

พอร์ท 0 สามารถถูกกำหนดให้เป็นอินพุท เอาท์พุท แยกอิสระกันทั้งแปดเส้นก็ได้ แต่ถ้านำมาใช้ร่วมกันทั้งแปดเส้น ก็สามารถใช้เป็นบัสแอดเดรส และบัสข้อมูลด้านต่ำแบบสองทิศทาง (bi-directional low-order address and data bus) เพื่อต่อกับหน่วยความจำ ภายนอก เช่น เมื่อแต่ละขาของ 8051 ถูกใช้เป็นอินพุท จะต้องส่งลอจิก 1 ไปยังอินพุทของ ดาต้าฟลิบฟลอบ โดยเขียนไปยังรีจิสเตอร์ P0 ใน SFR ผลที่ได้ขา Q จะให้ลอจิก 0 ออกมาทำ ให้ทรานซิสเตอร์หยุดทำงาน ผลที่ได้คือที่ขาของ 8051 แต่ละขาของพอร์ท 0 จะเข้าสู่สภาวะ high impedance สัญญาณที่เข้ามาที่ขา แต่ละขาของพอร์ท 0 จะถูกส่งไปยังอินพุทบัฟเฟอร์ โดยตรง โดยไม่มีการรบกวนกับส่วนอื่น เมื่อใช้งานในลักษณะของเอาท์พุทให้ส่งลอจิก 0 ไปยัง ฟลิบฟลอบโดยผ่านทางรีจิสเตอร์ P0 ใน SFR ซึ่งจะทำให้เฟทตัวล่างทำงาน จะเป็นว่าสัญญาณที่พอร์ทจะเป็น 0 ทันที ในทางกลับกันถ้าส่งลอจิก 1 ไปบ้าง เฟทจะหยุดทำงานทำให้เกิด สภาวะการ float หรือ high impedance ในกรณีจะต้องใช้ความต้านทานต่อจากพอร์ทไปยังไฟเลี้ยงของระบบเพื่อดึงสัญญาณให้เป็นลอจิก 1 ซึ่งเราเรียกว่า Pullup resister มิฉะนั้น ลอจิก 1 ที่ได้จะมีระดับศักดาที่ต่ำเกินไปจนกลานเป็นลอจิก 0 ได้

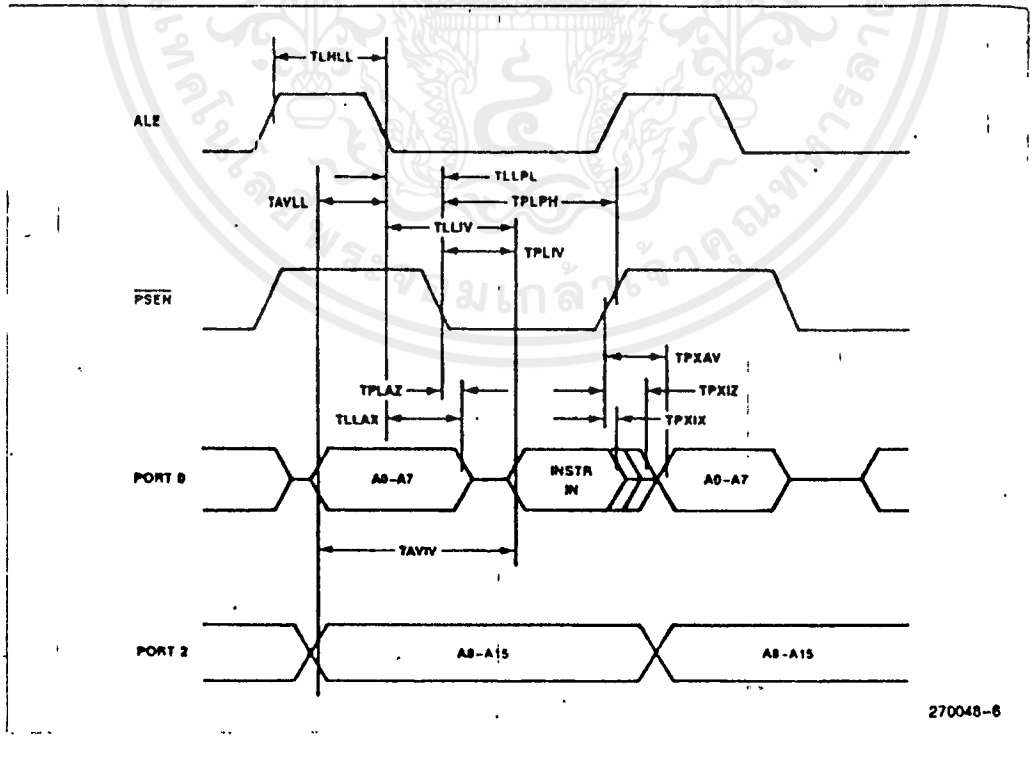
เมื่อพอร์ท 0 ถูกใช้เป็นบัสแอดเดรสเพื่อต่อกับหน่วยความจำภายนอก สัญญาณจะถูกส่งมายังส่วนควบคุมลอจิก(Control logic) เมื่อสัญญาณแอดเดรสเป็นลอจิก 1 ส่วนควบคุมลอจิก จะทำให้เฟทตัวบนทำงาน และเฟทตัวล่างหยุดทำงาน ส่งผลให้สัญญาณที่ขาพอร์ท 0 เป็นลอจิก 1 ในทางกลับกัน ถ้าสัญญาณแอดเดรสเป็น 0 ส่วนควบคุมลอจิกจะทำให้เฟทตัวบน หยุดทำงาน ส่วนเฟทตัวล่างจะทำงาน ทำให้ลอจิกที่ขาของพอร์ท 0 เป็นลอจิก 0 เมื่อสัญญาณ แอดเดรสถูกส่งออกมาที่พอร์ท 0 แล้วจะคงค้างสภาวะนี้ไว้ชั่วครู่ จากนั้นพอร์ท 0 จะเปลี่ยนตัว เองเป็นบัสข้อมูลทันที ถ้าเป็นการอ่านข้อมูลเข้าจากหน่วยความจำภายนอกมันจะถูกบังคับให้เป็น อินพุท โดยใส่ลอจิก 1 เข้าไปโดยอัตโนมัติ

พอร์ท 1 จะทำงานได้เพียงเป็นอินพุท/เอาท์พุทพอร์ทเท่านั้น จะเห็นว่าในรูป

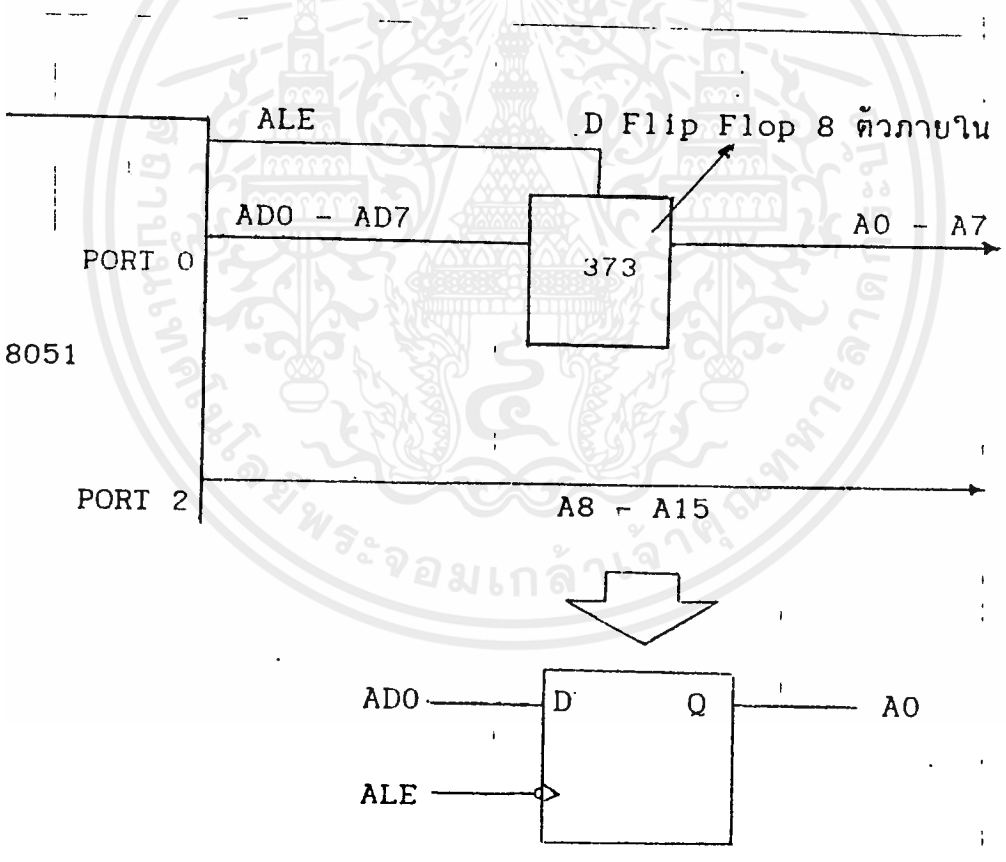
ขา Q ของตาต้าแลทช์จะต่ออยู่กับขาเกทของเฟทโดยตรง โดยมีเฟทอีกตัวหนึ่งทำหน้าที่เป็น Active pullup load เมื่อพอร์ท 1 ถูกใช้เป็นอินพุท ลอจิก 1 จะถูกส่งไปยังแลททำให้เฟทหยุดทำ งาน ทำให้ขาของพอร์ท 1 เป็นลอจิก 1 โดยผลจากการพูลอัพของแอคทีฟโหลดอยู่ตลอดเวลา เมื่อต่ออยู่กับวงจรภายนอก ถ้าลอจิกของวงจรภายนอกเป็น 1 ก็จะทำให้ลอจิกที่ขาของพอร์ท 1 ยังคงเป็นลอจิก1 อยู่เหมือนเดิม แต่ถ้าวงจรภายนอกเป็นลอจิก 0 มันจะดึงให้ระดับสัญญาพอร์ท เป็นลอจิก 0 ด้วย ด้วยเหตุนี้เองเมื่ออ่านข้อมูลผ่านทางบัฟเฟอร์เข้าไป ก็จะได้ลอจิกที่เป็นไปตามลอจิกภายนอกที่ต่ออยู่กับพอร์ท 1 เมื่อใช้พอร์ท 1 เป็นเอาท์พุท และส่งลอจิก 1 ตามต้อง การ แต่ถ้าส่งลอจิก 0 ออกไปยังพอร์ท 1 ก็จะทำให้ FET ทำงาน ทำให้ได้ลอจิก 0 ตามต้องการเช่นกัน

พอร์ท 2 ทำงานได้สองหน้าที่คือ หน้าที่แรกเป็นอินพุท/เอาท์พุท เช่นเดียวกับพอร์ท1 อีกหน้าที่หนึ่งจะทำหน้าที่ เป็นบัสแอดเดรสด้านสูง ที่ใช้งานร่วมกับพอร์ท 0 สำหรับต่อกับหน่วยความจำภายนอก ให้พิจารณาแผนผังของเวลาในการอ่านโปรแกรมจากหน่วยความจำภายนอก

EXTERNAL PROGRAM MEMORY READ CYCLE



จากรูปยังไม่ต้องสนใจกับสัญญาณ PSEN ก่อน ให้พิจารณาเฉพาะALE,PORT 0 และ PORT 2 เมื่อสัญญาณแอดเดรส A0 ถึง A7 ถูกส่งออกมาทางพอร์ท 0 และ A8 ถึง A15 ถูกส่งออกมาทางพอร์ท 2 เมื่อสัญญาณแอดเดรส A0 ถึง A15 มีระดับสัญญาณคงที่ หรือ สเตเบิล (Stable) แล้ว สัญญาณ ALE จะเปลี่ยนจากลอจิก 1 เป็นลอจิก 0 หลังจากนั้น ชั่วครู่ สัญญาณของพอร์ท 0 จะเปลี่ยนจากสัญญาณแอดเดรส เป็นสัญญาณข้อมูล ฉะนั้น ถ้าต้องการแยกเอาสัญญาณแอดเดรสออกมาจาก สัญญาณที่มัลติเพลกซ์ออกมาจากพอร์ท 0 ซึ่งเป็นทั้งแอดเดรส และ ข้อมูล จะต้องใช้ขอบขาลงของ ALE มาใช้งานร่วมกับ 74373 ซึ่งเป็น Octal D Latch ดังรูป



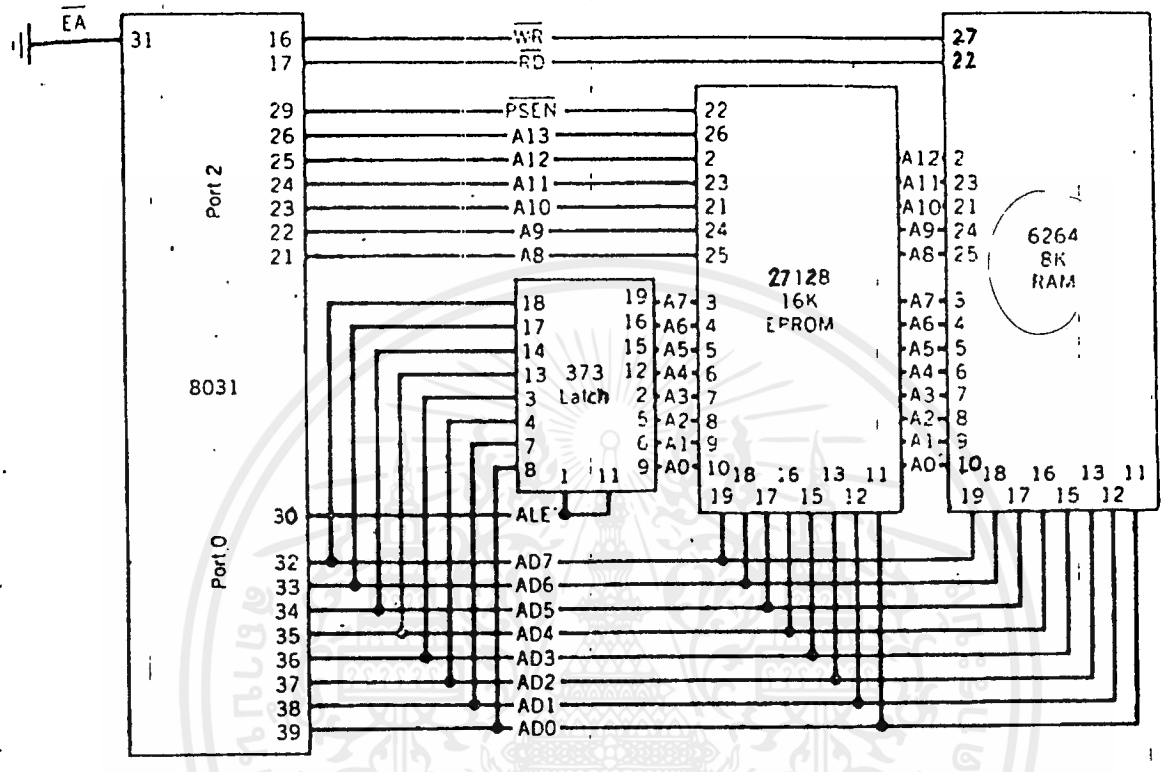
ในสถาวะที่ ALE เป็นลอจิก 1 จะเป็นช่วงที่ สัญญาณแอดเดรสออกมาจากพอร์ท 0 สัญญาณนี้จะถูกส่งผ่านไปยังเอาท์พุทของฟลิปฟลอป แต่ในขณะที่สัญญาณข้อมูลออกมาจากพอร์ท 0 ALE จะมีลอจิกเป็น 0 อยู่ก่อนแล้ว ทำให้ข้อมูลที่ขา D ของฟลิปฟลอปไม่สามารถผ่านไปยังขา Q ได้ นั่นหมายความว่าลอจิกที่ค้างที่ขา Q ของฟลิปฟลอปจะเป็นลอจิกของแอดเดรสเท่านั้น

พอร์ท 3 จะทำงานได้สองหน้าที่เช่นกัน หน้าที่แรกจะใช้เป็น อินพุท/เอาท์พุท เช่นเดียวกับพอร์ท 1 อีกหน้าที่หนึ่งทำงานดังนี้

ขา	ใช้งานเป็น	SFR
P3.0-RXD	รับข้อมูลแบบอนุกรม(Serial data input)	SBUF
P3.0-TXD	ส่งข้อมูลแบบอนุกรม(Serial data output)	SBUF
P3.2-INT0	รับอินเทอร์รัพภายนอกหมายเลข 0	TCON1
P3.3-INT1	รับอินเทอร์รัพภายนอกหมายเลข 1	TCON3
P3.4-T0	รับสัญญาณจากภายนอกของไทเมอร์หมายเลข 0	TMOD
P3.5-T1	รับสัญญาณจากภายนอกของไทเมอร์หมายเลข 1	TMOD
P3.6-WR	สัญญาณเขียนหน่วยความจำภายนอก(Writer)	-
P3.7-RD	สัญญาณอ่านหน่วยความจำภายนอก(Read)	-

2.4 การเชื่อมต่อกับหน่วยความจำภายนอก

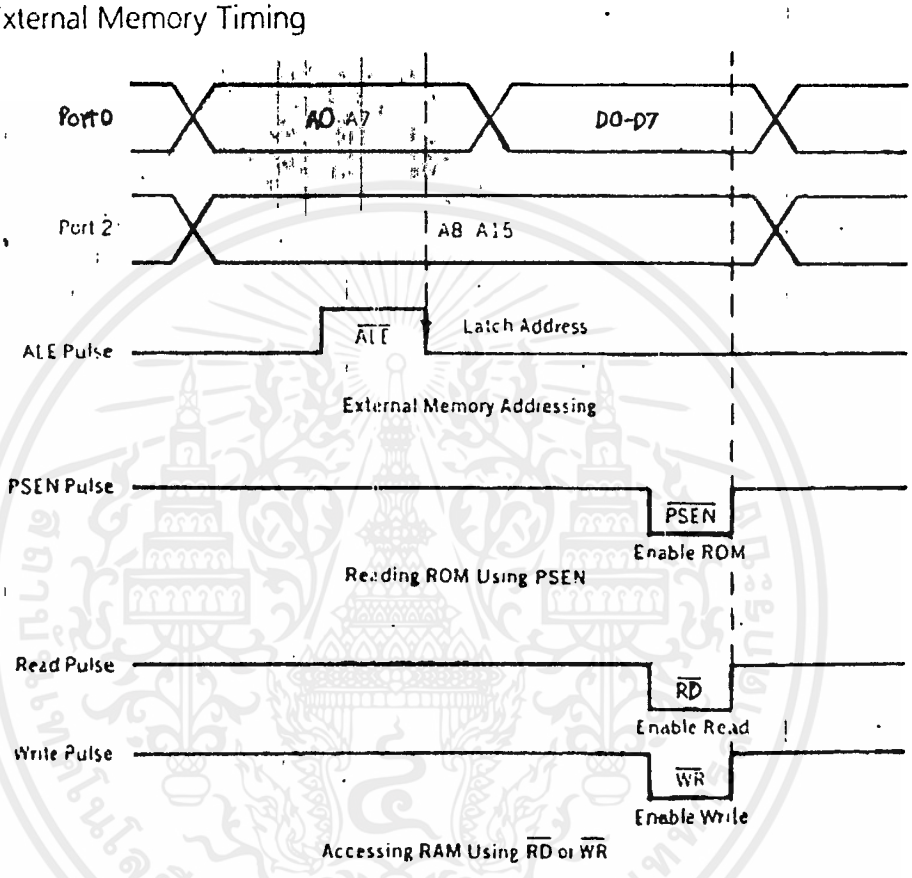
การใช้งาน 8051 บางครั้งความจำภายในตัวอาจน้อยเกินไป หรือไม่ต้องการใช้หน่วยความจำรอมภายใน (หรือ EPROM ภายใน) ที่มีขนาดแค่ 4 กิโลไบท์ สามารถต่อหน่วยความจำภายนอกขึ้นมาใช้งานได้ทั้ง 64 กิโลไบท์ โดยต่อขา 31 ของมันลงกราวน์ ดังรูปที่ 2.10



รูปที่ 2.10 (a) แสดงการต่อหน่วยความจำภายนอก

จากรูปเป็นการต่อ 8031(8031 ก็คือ 8051 ที่ไม่มีหน่วยความจำรอมภายใน ดังนั้นจึงต้องต่อหน่วยความจำภายนอกขึ้นมาใช้งาน) ใช้กับหน่วยความจำภายนอก โดยแบ่งเป็น EPROM ขนาด 16 กิโลไบ์ และ RAM ขนาด 8 กิโลไบท์ ขา 31 ของ 8031 ซึ่งเป็นขา EA (External access) จะถูกต่อลงกราวน์ เมื่อเป็นเช่นนี้พอร์ท 0 จะทำหน้าที่เป็นพอร์ทที่ให้สัญญาณแอดเดรสกับข้อมูล(AD0 - AD7) ด้านต่ำมัลติเพลกซ์กันออกมา, จากที่ได้กล่าวผ่านมาแล้ว การแยกเอาเฉพาะสัญญาณแอดเดรสออกมาให้หน่วยความจำ จะต้องใช้ 74LS373 คือสัญญาณ ALE(Address latch enable) ผลที่ได้จาก 74LS373 จะเป็นแอดเดรสอย่างเดียว และต่อตรงเข้าไปยังหน่วยความจำได้ทันที ส่วนข้อมูลที่ได้จาก

หน่วยความจำจะถูกต่อโดยตรงเข้ากับพอร์ท 0 สำหรับพอร์ท 2 จะเป็นสัญญาณแอดเดรสด้านสูง(A8 - A15) และเป็นสัญญาณที่ไม่ได้มีการมัลติเพลกซ์กับสัญญาณอื่นจึงต่อตรงได้เลย



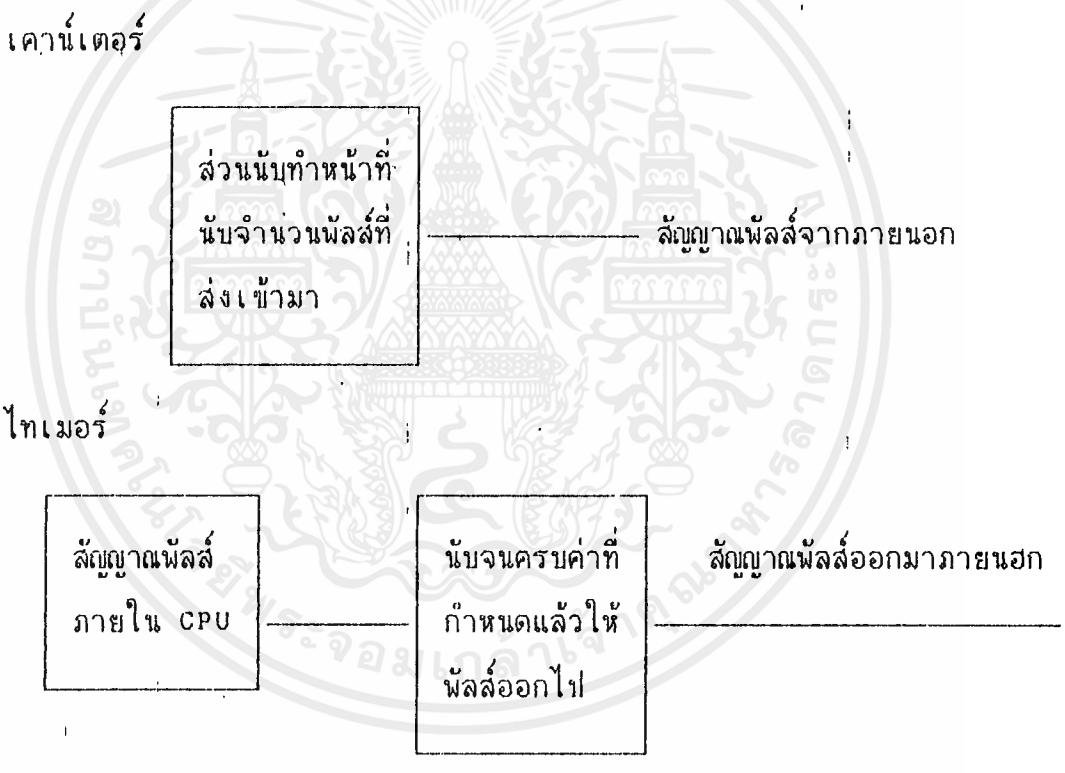
รูปที่ 2.10 (b) แผนผังของเวลา

จากแผนผังของเวลา จะเห็นว่าในช่วงแรกสัญญาณที่ออกมาจากพอร์ท 0 และพอร์ท 2 จะเป็นสัญญาณแอดเดรส สัญญาณแอดเดรสของพอร์ท 0 จะถูกแลทช์ด้วย 74LS373 ในช่วงขอบลงของพัลส์ของ ALE จากนั้นพอร์ท 0 จะเปลี่ยนมาเป็นบัสข้อมูล(D0 - D7) ในขณะนี้ถ้าเป็นคำสั่งอ่านข้อมูลจากหน่วยความจำ EPROM สัญญาณ PSEN (Program store enable)จะเป็นตัวกำหนดว่าที่ขอบบวกของมัน 8031 จะเอาข้อมูลที่ได้มาจาก EPROM ไปในตัว ในกรณีที่เป็นการอ่านหรือเขียนลงใน RAM จะใช้สัญญาณ RD หรือ WR อันใดอันหนึ่งแล้วแต่กรณี ถ้าเป็นการอ่านสัญญาณ RD ก็จะเป็นพัลส์ต่ำ ถ้าเป็นการเขียนสัญญาณ WR จะ

เป็นพัลส์สูง สัญญาณ RD และ WR จะเป็นพัลส์ต่ำพร้อมกันไม่ได้(สัญญาณในรูปเป็นการเปรียบเทียบให้ดูในแง่ของเวลาเท่านั้น ไม่ใช่พัลส์ออกมาต่ำทั้ง WR และ RD ให้นักศึกษาทำความเข้าใจให้ดี)

2.5 เคาท์เตอร์และไทม์เมอร์

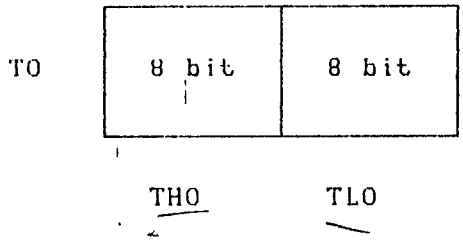
การใช้งาน 8051 บางกรณีนักศึกษาต้องการนับสิ่งที่เกิดขึ้นภายนอก เช่น ความถี่ของพัลส์ที่ป้อนเข้ามายัง 8051หรือให้สัญญาณพัลส์ที่มีขนาดแน่นอนออกไปภายนอก ในลักษณะนี้นักศึกษาต้องใช้ส่วนของเคาท์เตอร์ และ ไทเมอร์ ข้อแตกต่างของไทเมอร์ และเคาท์เตอร์จะเป็นดังนี้



เคาท์เตอร์จะทำหน้าที่นับจำนวนพัลส์ที่ป้อนเข้ามาจากภายนอก ส่วนไทเมอร์นั้น จะหน้าที่นับพัลส์ภายใน จนครบค่าที่กำหนดเป็น 200 พัลส์ เป็นต้น เมื่อครบแล้วก็จะเปลี่ยนสภาวะทางลอจิกที่ต่อออกไปภายนอกเป็นตรงกันข้าม เช่น จากลอจิก 0 เป็น 1 เป็นต้น

8051 จะมีไทเมอร์/เคาร์เตอร์ ขนาด 16 บิทสองตัวคือ T0 และ T1 เคาร์เตอร์จะถูกแบ่งเป็น รีจิสเตอร์ 8 บิท 2 ตัว คือ รีจิสเตอร์ T0 ซึ่งมีขนาด 16 บิท จะประกอบด้วยรีจิสเตอร์ขนาด 8 บิท สองตัวคือ TH0 และ TL0 และรีจิสเตอร์ T1 ซึ่งมีขนาด 16

บิทเช่นกันก็จะประกอบด้วยรีจิสเตอร์ขนาด 8 บิท สองตัวคือ TH1 และ TL1 ดังรูป

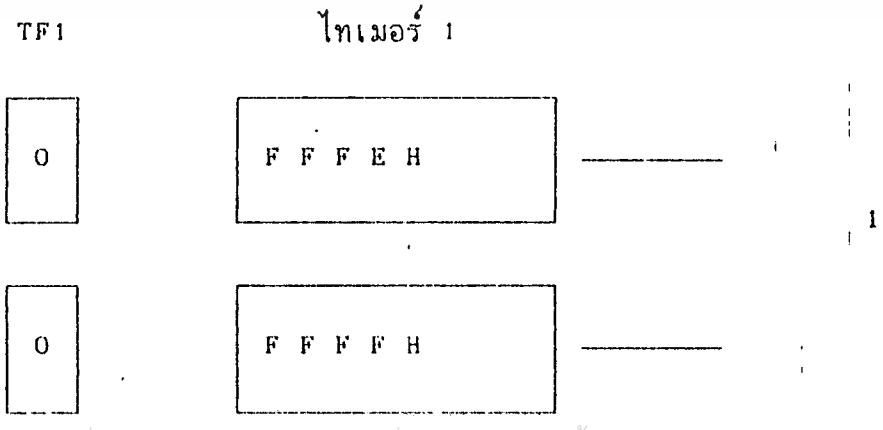


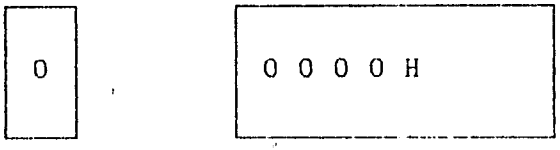
การควบคุมการทำงานจะสั่งผ่านรีจิสเตอร์ TCON (Timer control special function register) และ TMOD (Timer mode control special function register) โดยมีรายละเอียดดังนี้

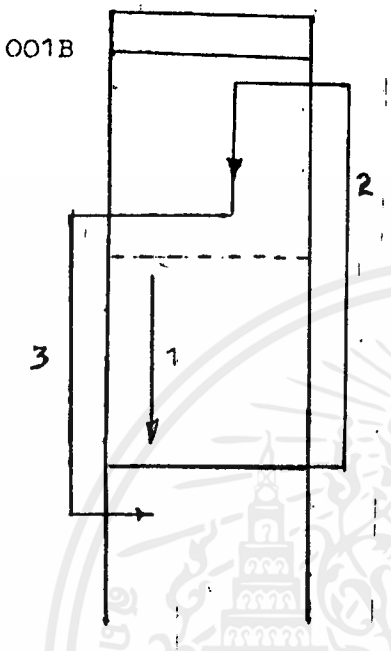
TCON

7	6	5	4	3	2	1	0
TF1	TR1	TF0	TR0	IT1	IL1	IT0	IL0

TF1 หมายถึง โอเวอร์โฟล์วแฟลกของไทเมอร์ 1 จะมีค่าเป็น 1 เมื่อเกิดโอเวอร์โฟล์วขึ้น(หมายถึงนับจนครบค่าสูงสุดคือทุกบิท ในไทเมอร์ 1 เป็น 1 ทั้งหมด แล้ววนรอบกลับมาเป็น 0-ทั้งหมดใหม่อีกครั้งหนึ่ง) TF1 จะถูกเคลียร์เป็น 0 เมื่อโปรเซสเซอร์กลับจากการทำงานในอินเทอร์รัพเซอร์วิสรูทีน(Interupt service routine) ที่เริ่มต้นด้วยตำแหน่งแอดเดรส 001BH เพื่อให้เข้าใจให้ดูจากรูป







1. ในกรณีที่ไทเมอร์ 1 ยังไม่เกิดโอเวอร์โฟล์ว สมมุติให้โปรแกรม RUN ที่ตำแหน่ง 1
2. เมื่อเกิดโอเวอร์โฟล์วขึ้นในไทเมอร์ 1 2 โปรแกรมจะกระโดดไปทำงานยังเซอร์วิสรูทีนที่เริ่มต้นที่ตำแหน่งแอดเดรส 001BH จนเสร็จสิ้น
3. แล้วทำการเคลียร์ TF1 เป็น 0 และกลับมาทำงานต่อในตำแหน่งถัดมาของโปรแกรมก่อนที่จะกระโดดไปยังเซอร์วิสรูทีน

TR1 หมายถึง บิทที่ใช้บังคับให้ไทเมอร์ 1 ทำงานหรือหยุดทำงาน ถ้า TR1 เป็นลอจิก 1 ไทเมอร์จะทำงาน ถ้าเป็นลอจิก 0 ไทเมอร์ 1 จะหยุดทำงาน (หยุดทำงานหมายถึง หยุดนับพัลส์แต่ไม่ใช่รีเซ็ทค่าในไทเมอร์ 1ให้เป็น 0000 ใหม่)

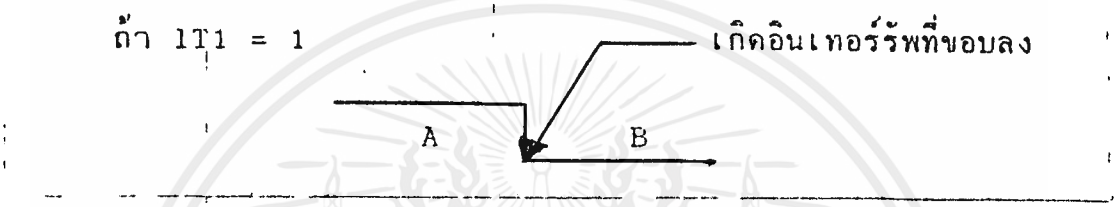
TF0 หมายถึง โอเวอร์โฟล์วแฟลกของไทเมอร์ 0 (การทำงานต่าง ๆ เป็นเช่นเดียวกับ TF1) โดยมีแอดเดรสเริ่มต้นของเซอร์วิสรูทีนเป็น 000BH

TR0 หมายถึง บิทที่ใช้บังคับให้ไทเมอร์ 0 ทำงาน หรือหยุดทำงาน เช่นเดียวกับ TR1

IE1 หมายถึง การยอมให้อินเทอร์รัพจากภายนอก โดยผ่านทางพอร์ท 3 ขา 3.3 (INT1) บิทนี้จะถูกเซ็ทเป็น 1 เมื่อสัญญาณจากภายนอกเข้ามากระตุ้นที่ขา 3.3 โดยจะทำการอินเทอร์รัพ เมื่อสัญญาณเปลี่ยนสภาวะจากลอจิก 1 มาเป็นลอจิก 0 (ขอบลบของพัลส์)ตำแหน่งของเซอร์วิสรูทีนอยู่ที่แอดเดรส 0013H เมื่อทำคำสั่ง RETI(Return from interupt) IE1 จะถูกเคลียร์โดยอัตโนมัติ จะเห็นได้ว่า บิทนี้ไม่มีส่วนเกี่ยวข้องใด ๆกับไทเมอร์/เคาท์เตอร์

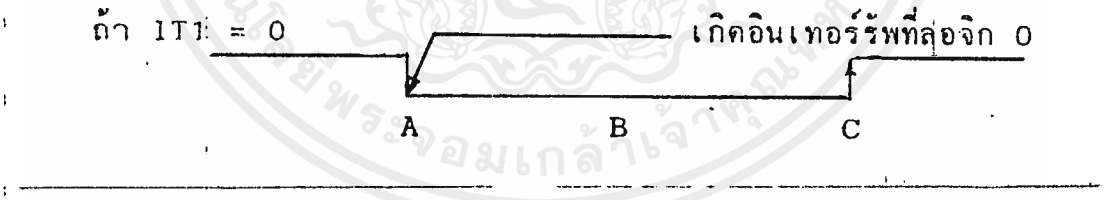
IT1 หมายถึง การควบคุมสัญญาณที่เข้ามากระตุ้นที่ขา 3.3 ของพอร์ท 3 ว่าให้เป็นการกระตุ้นแบบใด จึงจะเกิดการอินเทอร์รัพขึ้น ถ้า IT1 = 0 จะเป็นการกระตุ้นโดยลอจิก 0 (สัญญาณเป็น 0) ข้อแตกต่างที่น่าจดจำก็คือ

ถ้า IT1 = 1 เกิดอินเทอร์รัพท์ที่ขอบลง



ถ้าเกิดอินเทอร์รัพขึ้นแล้ว ไปเซอร์วิสรูทีนจนเสร็จสิ้น สมมุติว่ากลับมาที่เวลา B ซึ่งสัญญาณยังเป็นลอจิก 0 อยู่ในลักษณะนี้ จะไม่เกิดอะไรขึ้น แต่ถ้า

ถ้า IT1 = 1 เกิดอินเทอร์รัพท์ที่ลอจิก 0



เมื่อกลับมาจากเซอร์วิสรูทีนที่ตำแหน่ง B ไมโครโปรเซสเซอร์จะพบเป็นลอจิก 0 อีก ผลที่ได้ก็คือ จะเกิดอินเทอร์รัพขึ้นอีกครั้งหนึ่ง ในประเด็นนี้จะทำให้เกิดการอินเทอร์รัพซ้ำซ้อนขึ้น โดยที่เราไม่ต้องการ วิธีการแก้ไขก็คือก่อนที่จะ RUN คำสั่ง RETI ให้ตรวจสอบก่อนว่า สัญญาณกลับมาเป็นลอจิก 1 หรือยัง ถ้าเป็นลอจิก 1 แล้ว C จึงค่อย RUN คำสั่ง RETI

IE0 หมายถึง การยอมให้อินเทอร์รัพจากภายนอก โดยผ่านทางพอร์ท 3 ขา 3.2

(INTO)การอินเทอร์รัพที่ขานี้ สัญญาณจะต้องเป็นการกระตุ้นแบบขอบลงหรือโปรแกรมได้เช่นเดียวกับ IE1 ตำแหน่งเริ่มต้นของเซอร์วิสรูทีน อยู่ที่แอดเดรส 0003H

IT0 หมายถึง การควบคุมสัญญาณที่เข้ามาอินเทอร์รัพ IE0 เช่นเดียวกับ IT1

TMOD

7	6	5	4	3	2	1	0	bit
gate	C/T	M1	M0	gate	C/T	M1	M0	
[Timer 1				Timer 2				]

บิท

7/3 gate หมายถึง การอินาเบิลออร์เกท เพื่อจะควบคุมให้ไทเมอร์ 1 หรือ 0 ทำงานหรือหยุดทำงาน ถ้าเป็นลอจิก 1 ไทเมอร์จะทำงาน ถ้า TR1/0 ใน TCON รีจิสเตอร์ถูกเซ็ทเป็น 1 และ สัญญาณที่ต่ออยู่กับขา INT 1/0 เป็นลอจิก 1

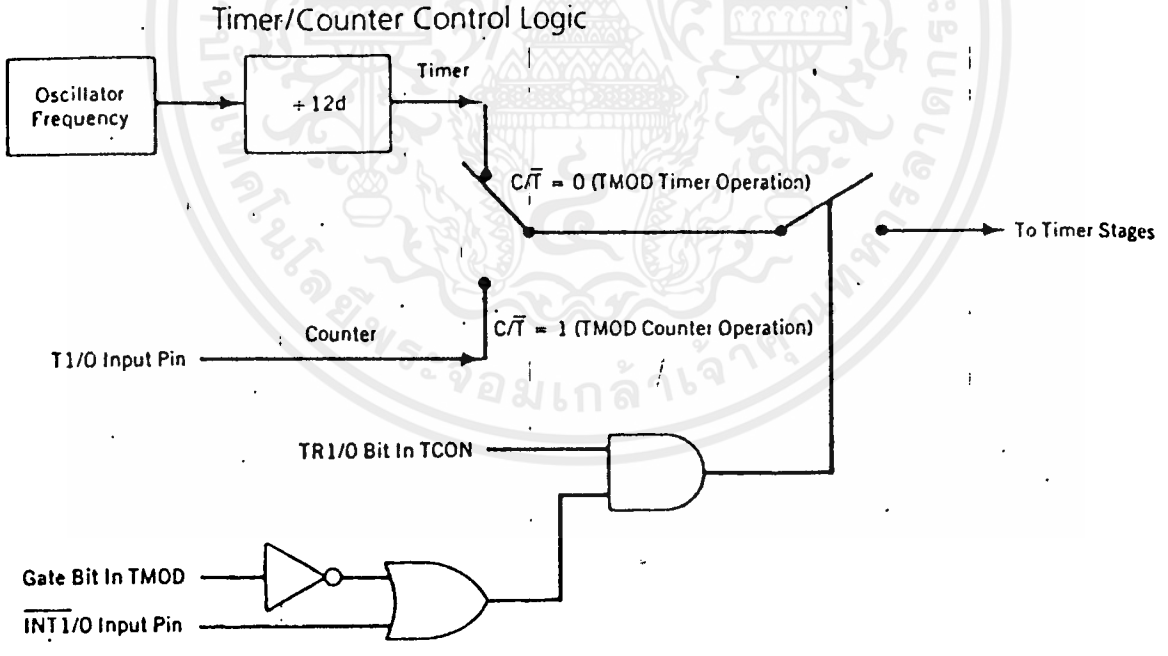
6/2 C/T หมายถึง ถ้าเป็นลอจิก 1 เป็นการเลือกให้ไทเมอร์ 1/0 ทำงานในลักษณะของเคาร์เตอร์ทำการนับพัลล์จากภายนอกที่ป้อนเข้ามาโดยผ่านทาง ขา 3.5 (T1) หรือ 3.4 (T0) ถ้าเป็นลอจิก 0 เป็นการเลือกให้ไทเมอร์ 1/0ทำงานในลักษณะของไทเมอร์ โดยการนับพัลล์ที่ได้จากภายในตัว 8051 เอง

5/1 M1 หมายถึง บิทที่ใช้ควบคุมให้ไทเมอร์/เคาท์เตอร์ทำงานในโหมดใดบิทที่ 1

4/O MO หมายถึง บิทที่ใช้ควบคุมให้ไทเมอร์/เคาร์เตอร์ ทำงานในโหมดใด บิทที่ 0 โดยมีรายละเอียดดังนี้

M1	MO	โหมด
0	0	0
0	1	1
1	0	2
1	1	3

การควบคุมให้ไทเมอร์/เคาร์เตอร์ ทำงานในลักษณะของไทเมอร์ หรือ เคาร์เตอร์ จะโครงสร้างภายในเป็นดังรูป 2.11

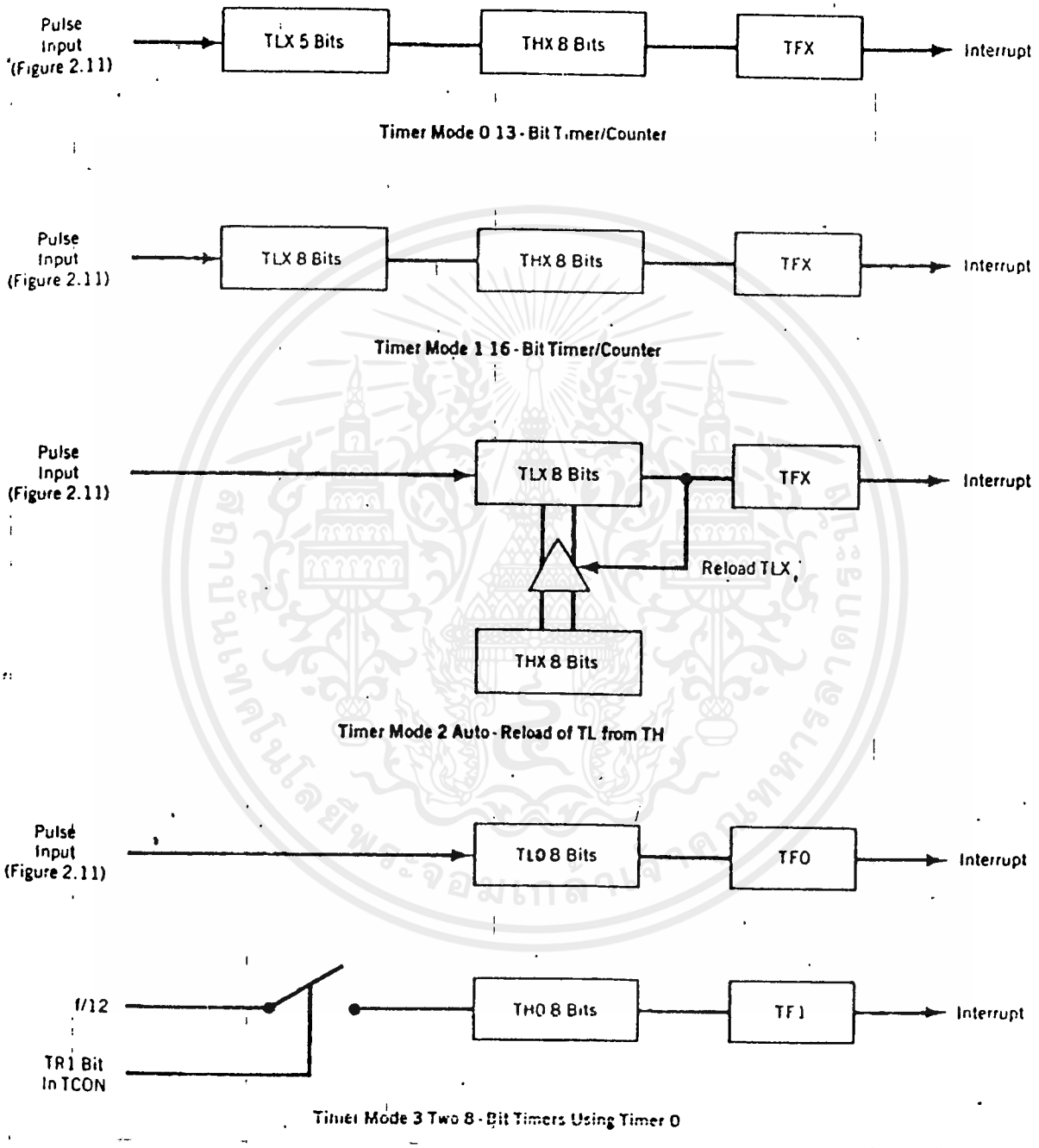


รูปที่ 2.11 แผนผังการควบคุมไทเมอร์/เคาท์เตอร์

จากรูปเมื่อเลือกให้ C/T เป็น 0 ไทเมอร์/เคาร์เตอร์ จะทำงานในโหมดของไทเมอร์ โดยการรับสัญญาณจากภายใน ที่ได้จากคริสตอลด้วย 12 เมื่อเลือกให้ C/T เป็น 1 ไทเมอร์/เคาร์เตอร์ จะทำงานในโหมดของเคาร์เตอร์ สวิทช์จะต่อเข้ากับสัญญาณพัลส์จากภายนอก ที่ป้อนเข้ามาทาง T1/0 อินพุท จะมีสวิทช์อีกตัวหนึ่ง ทำหน้าที่ต่อสัญญาณที่เลือกแล้วไปยังส่วนของ Timer stage จะเห็นว่า สัญญาณที่ควบคุมแอนเกท ซึ่งต้องเซ็ทเป็น 1 ก็คือ TR1/0 ใน TCON รีจิสเตอร์ถ้าเป็น 0 แอนเกทจะไม่ยอมให้สัญญาณจาก ออร์เกทผ่านไปได้เลย การที่แอนเกทจะเป็นลอจิก 1 เพื่อบังคับให้สวิทช์ปิดเอาท์พุทของออร์เกทจะต้องเป็น 1 ด้วย

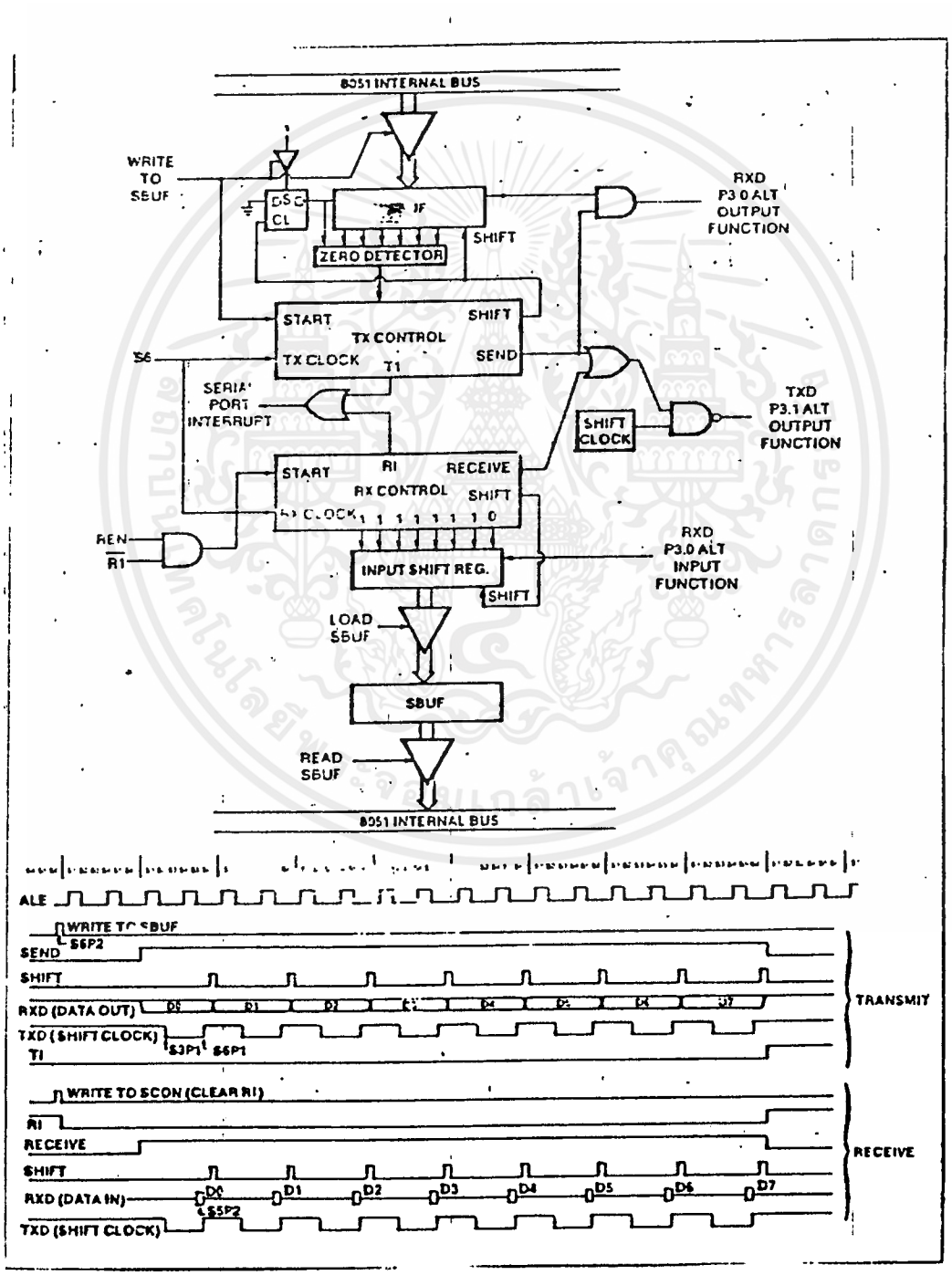
โหมดการทำงานของไทเมอร์ จะเป็นดังรูปที่ 2.12

FIGURE 2.12 Timer 1 and Timer 0 Operation Modes



รูปที่ 2.12 การทำงานในโหมดต่างๆ ของไทเมอร์

จากรูปในโหมด 0 จะใช้ TLX (X หมายถึงไทเมอร์ 0 หรือ 1) แค่ 5 บิท ส่น THX จะใช้ครบทั้ง 8 บิท ในโหมด 1 จะใช้ครบทั้ง 8 บิททั้งสองรีจิสเตอร์ ในโหมด 2 จะใช้ TLX เป็นตัวนับส่วน THX เป็นตัวตั้งค่า เมื่อ TLX นับจนครบเป็น 00 มันจะโหลดค่าเริ่มต้นที่เก็บไว้ใน THX เข้าตัวมันแล้วเริ่มนับถอยหลังจากค่านี้ไป การกระทำในลักษณะนี้เรียก Auto-Reload ส่วนโหมด 3 จะเป็นการใช้ไทเมอร์ 0 และ 1 ในโหมดที่แตกต่างกัน



บทที่ 3

Network Configuration

ปัญหาแรกของ network system design ก็คือจะเชื่อมต่อคอมพิวเตอร์เข้าด้วยกันด้วยวิธีใด มี configuration พื้นฐานอยู่สองแบบ คือ star และ loop แสดงในรูปที่ 3.1

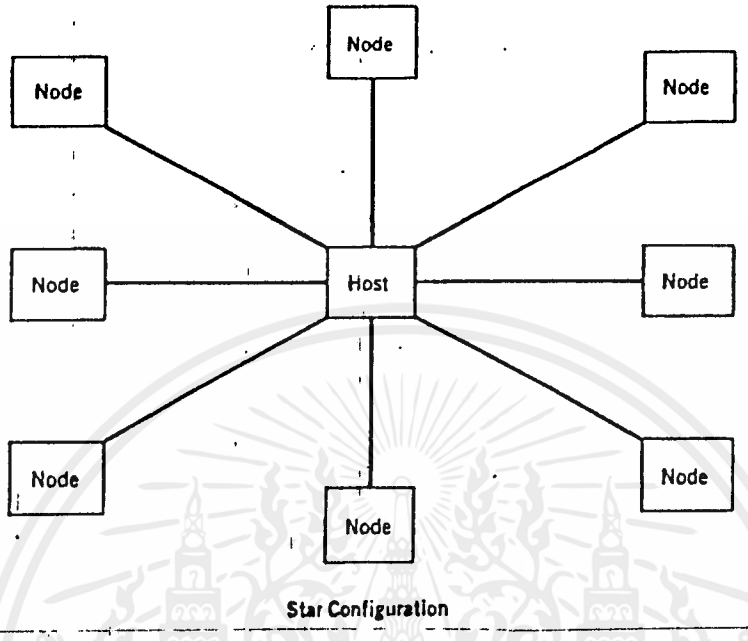
แบบ star สาย 1 สายจากคอมพิวเตอร์ศูนย์กลางไปยังคอมพิวเตอร์เครื่องอื่นแต่ละเครื่อง (จาก 'host' ไปยัง 'node') รูปแบบนี้ปกติจะมีการทำงานแบบ time-sharing โดย mainfram (ศูนย์กลาง) จะเชื่อมต่อกับ terminals หรือ PCS โดยใช้สายแยกกันสำหรับแต่ละ node ซึ่งแต่ละ node จะสนใจเฉพาะข้อมูลบนสายของมันเท่านั้น

แบบ loop จะใช้สายสื่อสารข้อมูลเพียงเส้นเดียวสำหรับเชื่อมต่อคอมพิวเตอร์ทั้งหมดเข้าด้วยกัน บนโครงข่ายแบบ loop อาจจะมี host สำหรับควบคุมการทำงานเพียงตัวเดียวก็ได้ หรือ คอมพิวเตอร์แต่ละตัวอาจเป็น host ในบางเวลาก็ได้ ตามปกติโครงข่ายแบบ loop จะใช้สำหรับการเก็บรวบรวมข้อมูลโดย host จะสอบถามไปยังแต่ละ node เป็นระยะ ๆ เพื่อรวบรวมข้อมูลล่าสุดเกี่ยวกับกระบวนการทำงาน ซึ่งแต่ละ node ก็สามารถรับรู้ถึงข้อมูลนั้น ๆ ได้ เพราะใช้สายส่งข้อมูลร่วมกัน

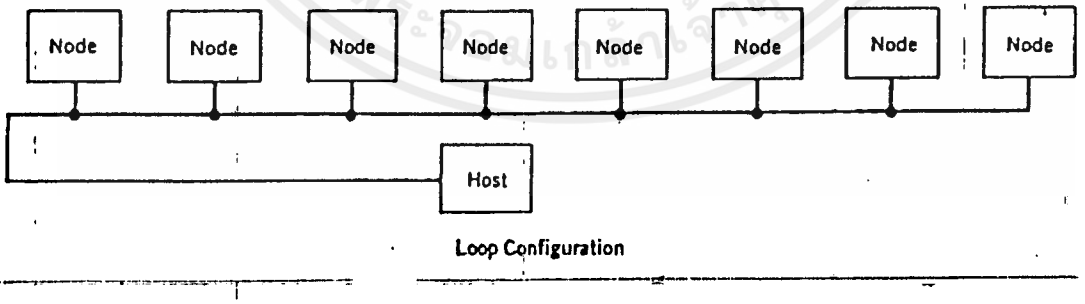
การเลือกรูปแบบโครงข่ายนั้นขึ้นอยู่กับ factors ภายนอก ซึ่งอยู่นอกเหนือการควบคุมของ system designer ซึ่งมีข้อมูลบางอย่างที่ควรจะพิจารณา ดังนี้

objective	network	comment
Reliability	Star	Single node loss per line loss
Fault isolation	Star	Fault traceable to node and line
Speed	Star	Each node has complete line use
Cost	Loop	Single line all node

แบบ star เหมาะสำหรับระบบที่มีจำนวน node น้อยๆ หรือ ระยะทางจาก host ไปยัง node สั้นๆ เพราะยิ่งจำนวน node มีมากขึ้น ค่าใช้จ่ายเกี่ยวกับสายส่งข้อมูลก็จะยิ่งมากขึ้นด้วย ดังนั้น ถ้าหากมีข้อจำกัดเรื่องค่าใช้จ่ายก็ควรพิจารณาแบบ loop ดีกว่า



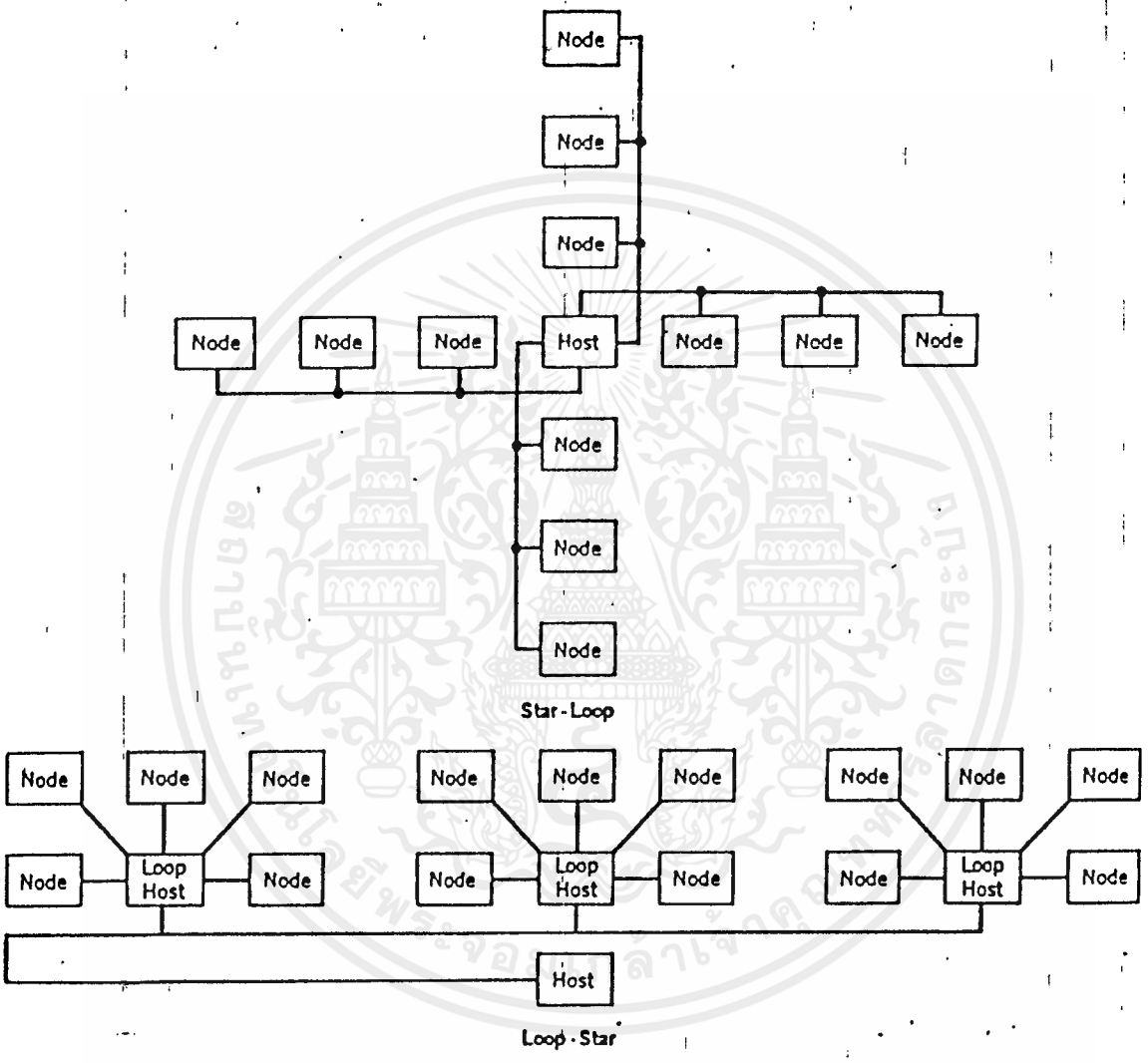
Star Conficguration



Loop Configuration

รูปที่ 3.1

FIGURE 9.2 Hybrid Communication Configurations



Loop Star

รูปที่ 3.2

ในระบบอุตสาหกรรมมักมี node จำนวนมาก และ มีระยะทางไกลกัน ทำให้ loop network ดีกว่าในกรณีนี้ซึ่งปกติจะต้องมี host สำหรับควบคุมการส่งข้อมูลบน loop host จะต้องมี software ที่คอยตรวจสอบหา node ที่ทำงานผิดพลาด เพื่อเพิ่มความเชื่อถือได้ของข้อมูลของระบบ การส่งข้อมูลด้วยความเร็วสูงเพื่อความเร็วในการตอบสนองสูง จะถูกใช้เมื่อจำเป็น

คำว่า 'Speed cost money : How fast do you want to go ?' จะเป็นจริงสำหรับการออกแบบระบบแบบ loop บนระบบส่วนใหญ่ที่ใช้ RS-232 และ RS-485 ซึ่งให้อัตราการสูง 100 kilobaud ในระยะทาง 4000 foot มักจะใช้สายส่งแบบคู่สาย (twisted - pair) ซึ่งมีราคาไม่แพง ถ้าต้องการให้อัตราการส่งสูงต้องส่งในระยะทางสั้นๆ หรือ ใช้สายส่งแบบสายเดียวซึ่งมีราคาแพงกว่า เช่น สายเคเบิล จะเห็นได้ว่า ค่าใช้จ่ายในการวางสาย จะเป็นปัจจัยหลักในการออกแบบระบบใหญ่ ๆ ซึ่งมี node จำนวนมาก

มี hybrid network จำนวนมาก ที่มีการรวมกันระหว่าง แบบ loop และแบบ star ดัง รูปที่ 3.2 แสดงสองระบบโครงข่ายที่นิยมใช้กันอย่างแพร่หลาย

3.1 8051 Data Communication Mode

8051 มักมีพอร์ทอนุกรมอยู่หนึ่งพอร์ท ซึ่งใช้ขา 3.0 (TXD) และ 3.1 (RXD) ในการรับและส่งข้อมูลตามลำดับ โดยใช้รีจิสเตอร์สองตัว คือ SBUF หนึ่งตัวสำหรับการส่ง และ อีกหนึ่งตัวสำหรับการรับ การเขียนข้อมูลลงบน SBUF เป็นการส่งข้อมูล ส่วนการอ่านข้อมูลจาก SBUF จะเป็นการรับข้อมูล ทั้งการส่งและการรับสามารถกระทำพร้อมกันได้ในเวลาเดียวกัน และตัวรับยังคงอยู่บนกระบวนการรับได้ในขณะที่ข้อมูลไบท์ก่อนยังคงอยู่ใน SBUF ไบท์แรกจะต้องถูกอ่านก่อนที่การรับจะเสร็จสิ้นลง กรือ ก่อนที่ไบท์ที่สองจะหายไป

ในทางฟิสิคัล data คือ ชุดลำดับของโวลต์เตจที่ถูก sample ตรงจุดกึ่งกลางของบิท ณ ความถี่ที่ถูดกำหนดโดย serial data mode และ โปรแกรมที่ควบคุม mode เท่านั้น อุปกรณ์ทุกตัวที่ต้องการสื่อสารข้อมูลต้องมีระดับ idle state , logic mode , charecter code และความถี่(baud rate) เดียวกัน สายที่ต่อกับ port จะต้องมีขั้วเดียวกัน ดังนั้น idle mode , logic mode จะถูกมองเห็นโดยทุก port

8051 มีพอร์ทอนุกรมที่สามารถเลือกการทำงานในโหมดต่าง ๆ ได้ 4 โหมด

Mode 0 ข้อมูลจะเข้าและออกผ่านทาง RXD และ TXD ด้วยการเลื่อนสัญญาณนาฬิกาเอาท์พุท ข้อมูลจะเป็นลักษณะแปดบิท ในการรับและส่งแต่ละครั้ง โดยที่ส่งค่า LSB ก่อน อัตราบ๊อดคงที่ที่ 1/12 ของความถี่ออสซิลเลเตอร์ รูปที่ 3.3 แสดงถึงแผนภูมิเวลาสำหรับการส่งและรับ

การส่งถูกเริ่มด้วยคำสั่งต่างๆในการใช้ SBUF เป็นรีจิสเตอร์รับข้อมูล สัญญาณคำสั่ง 'write to SBUF' เกิดที่ S6,P2 และบรรจุค่า '1' เข้าที่ตำแหน่งบิทที่ 9 ของตัวรีจิสเตอร์การเลื่อนส่ง และบอกให้ส่วนควบคุมการส่ง (TX Control Block) ให้ทำงานการส่ง ภายในช่วงเวลาหนึ่งวัฏจักรแมชชีนจะครอบคลุมให้เกิดพัลส์ 'Write to SBUF' ที่ S6,P2 และสัญญาณ SEND ต้องแอกทีฟสูงในช่วงการส่ง

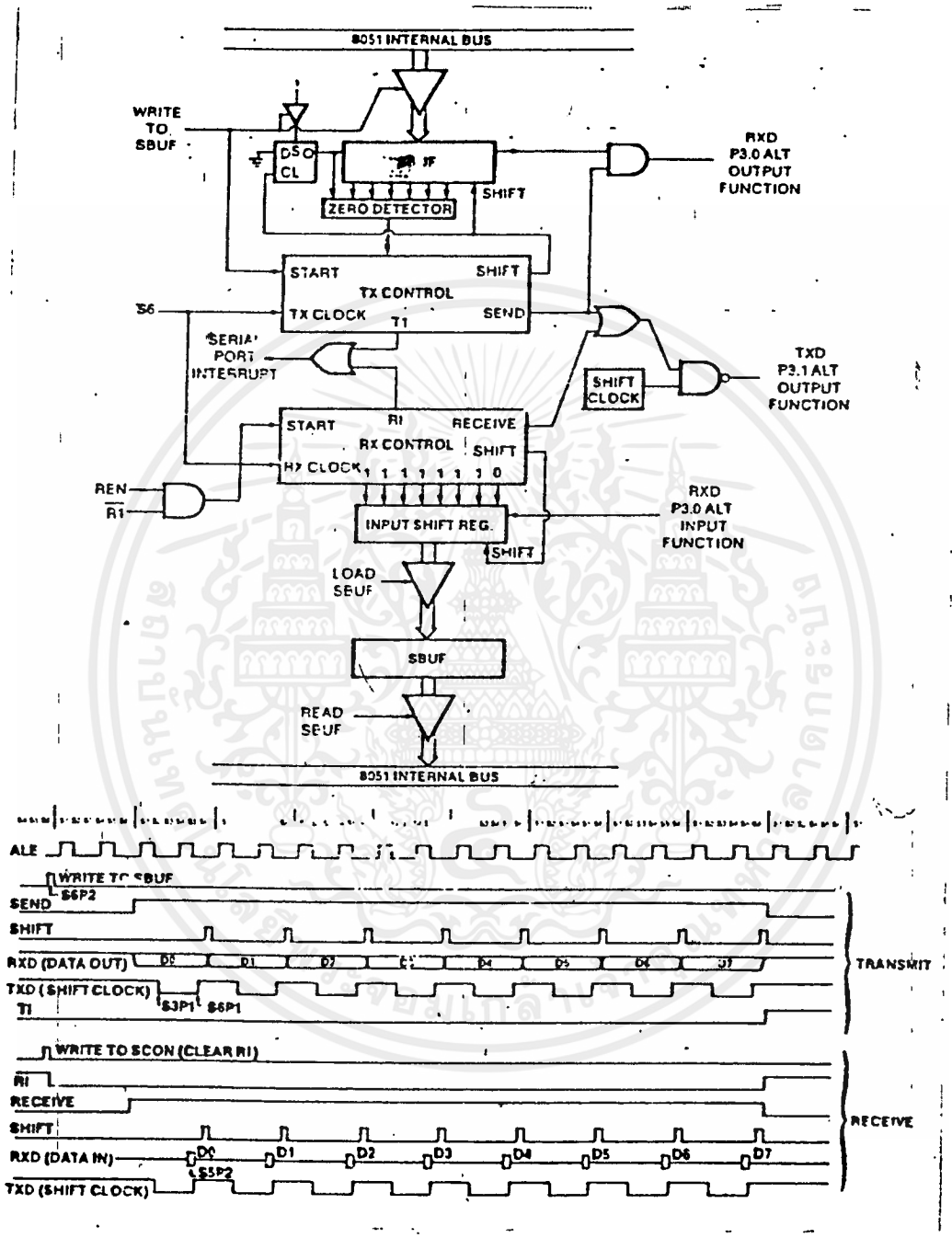
การอินาเบิ้ล SEND ที่มีแอกทีฟสูง จะเป็นการควบคุมให้ข้อมูลส่งออกจากตัวรีจิสเตอร์ การเลื่อนบิทออกที่ขา P3.0 และตัวเลื่อนสัญญาณนาฬิกา (shift clock) ก็จะถูกส่งออกที่ขา P3.1 โดยที่ลักษณะของสัญญาณ shift clock จะมีระดับต่ำที่ S3 , S4 และ S5 ของทุกวัฏจักรแมชชีน และมีระดับสูงช่วง S6 S1 และ S2 ในช่วง S6P2 ของทุกวัฏจักรแมชชีนขณะที่ SEND แอกทีฟ ค่าข้อมูลในรีจิสเตอร์การเลื่อนส่งจะถูกเลื่อนไปทางขวาหนึ่งตำแหน่งหรือบิท คือเป็นการส่งที่ขา P3.0 ไปหนึ่งบิทขณะนั้น

ในขณะที่เลื่อนบิทไปทางขวา ค่า '0' จะเข้าแทนที่ทางซ้าย เมื่อเลื่อนจน MSB เลื่อนออกจากตัวรีจิสเตอร์การเลื่อน ดังนั้น ค่า '1' จะเริ่มถูกบรรจุเข้าไปที่ตำแหน่งที่เก้าต่อจาก MSB ที่ถูกเลื่อนออกไป และทุกตำแหน่งทางซ้ายมือจะเป็น '0' หมดทุกตัว สถานะของแฟลกในการควบคุมการส่งจะทำการเลื่อนครั้งสุดท้าย และให้สัญญาณ SEND กลับคืนจากสภาพแอกทีฟเป็นระดับต่ำ และปรับ TI เป็นระดับสูงด้วย ส่งออกไปที่พอร์ทอนุกรมการอินเทอร์รัพต์ ทั้งสองสัญญาณที่ปรับระดับนี้จะเกิดช่วง S1P1 ของวัฏจักรที่สิบหลังจากที่สัญญาณ 'Write to SBUF' เริ่มสโตรบ

การรับข้อมูลจะถูกควบคุมด้วยการโปรแกรมเริ่มแรก การตั้งข้อแม้ให้ REN =1 และ RI =' 0 ที่ S6P2 ของวัฏจักรแมชชีนตัวใหม่ หน่วยควบคุม RX จะเขียนบิทค่า $1111\ 1110_2$ ไปยังรีจิสเตอร์การเลื่อนรับ และที่เฟส Clock ตัวใหม่ จะแอกทีฟสัญญาณ Recieve ให้สูง

เมื่อ Recieve ถูกอินาเบิ้ลให้สูงก็จะทำให้สัญญาณ Shift Clock ส่งฟังก์ชัน

ต่างๆ ออกที่ขา P3.1 Shift clock จะเปลี่ยนสถานะที่ S3P1 และ S6P1 ของทุกวัฏจักรแมชชีนที่ S6P2 ของวัฏจักรแม่ชชีนตัวแรก สัญญาณ Recieve เริ่มแอกทีฟสูง



รูปที่ 3.3 เป็นการแสดงถึงฟังก์ชันแผนภูมิแบบธรรมดาของพอร์ทอนุกรม

ในโหมด 0 และช่วงเวลาที่เกิดขึ้นและค่าในรีจิสเตอร์การเลื่อนรับจะรับข้อมูลเข้ามาและเลื่อนมาทางซ้ายหนึ่งตำแหน่ง และ ทุกค่าที่เข้ามาจากทางขวาจะเป็นค่าที่ถูก Sample เข้าที่ขา P3.0 ที่ช่วง S5P2 ของทุกวัฏจักรแมชชีน

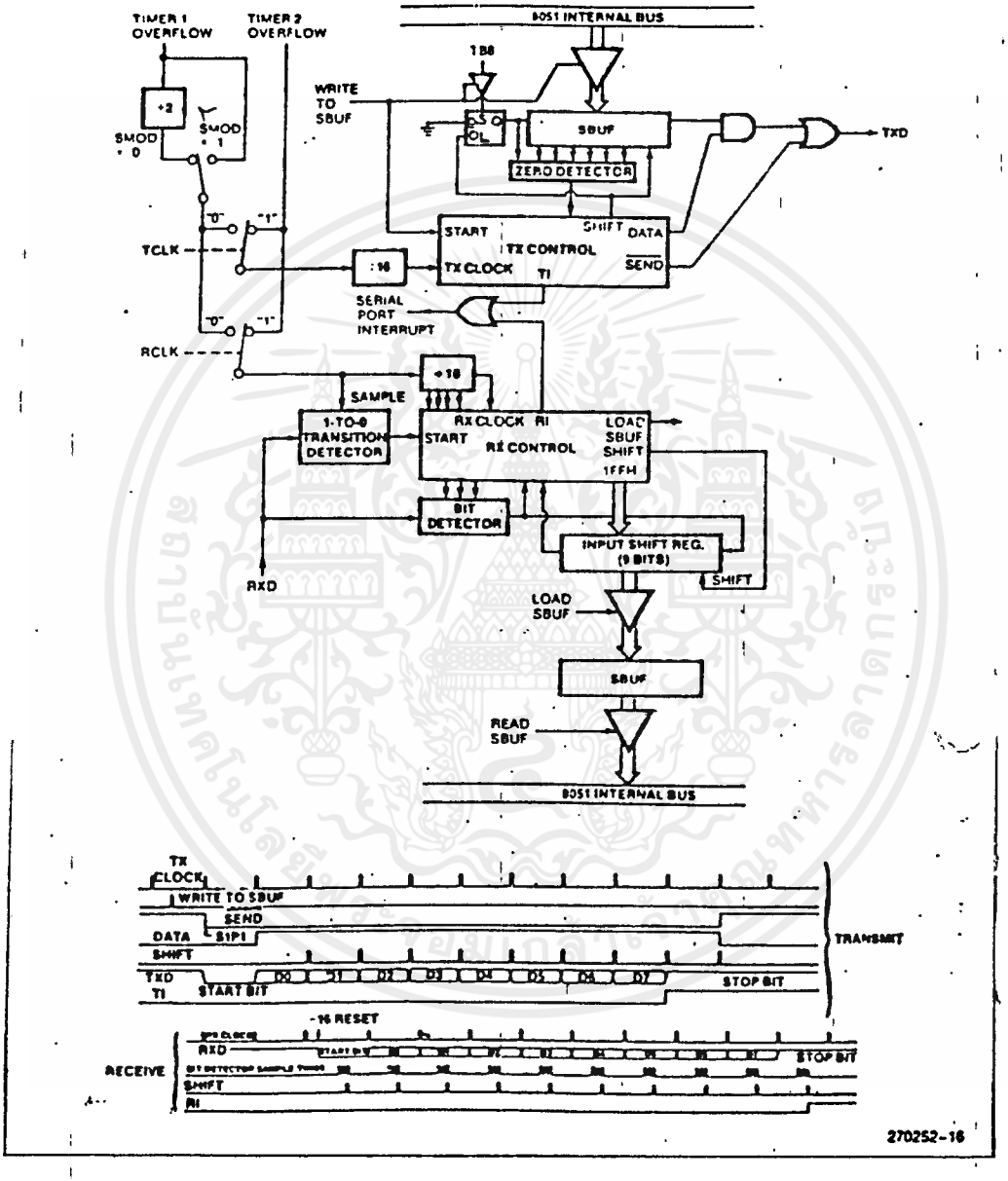
ขณะที่ข้อมูลบิทต่ำเข้ามาทางขวา ค่า '1' จะเลื่อนออกไปทางซ้าย เมื่อค่า '0' ถูกบรรจุเริ่มแรกที่เข้าทางตำแหน่งขวาสุดถูกเลื่อนมาอยู่ตำแหน่งซ้ายสุดในรีจิสเตอร์การเลื่อนจะมีผลให้ค่าแฟลกในหน่วยควบคุมการรับ (RX control Block) ให้ทำการเลื่อนเป็นครั้งสุดท้ายและเริ่มบรรจุข้อมูลทั้งหมดเข้าใน SBUF ที่ช่วง S1P1 ของวัฏจักรแมชชีนที่สิบหลังจากเริ่มส่งสัญญาณสโตรบ 'Write to SCON' ของ RI เคลียร์ ต่อจากนั้นสัญญาณ Recieve จะเคลียร์ และ RI จะปรับเป็น 1

Mode 1 จำนวนสิบบิทจะถูกส่งผ่าน TXD หรือรับผ่าน RXD ที่ประกอบด้วยบิท Start บิทข้อมูล 8 บิท และบิท Stop การรับบิท Stop จะส่งเข้า RB8 ในรีจิสเตอร์ SCON การตั้งอัตราความเร็วของบ็อดจะแปรผันได้ ทั้งตัวจับเวลา 1 หรือ 2 อาจใช้เป็นสัญญาณนาฬิกาสำหรับพอร์ทอนุกรมโดยสร้างอัตราความเร็วแปรผันด้วยการตั้งหรือเคลียร์ค่าในบิท T2CON เป็น TCLK และ RCLK รูปที่ 3.4 แสดงถึงแผนภูมิการใช้งานในโหมด 1 พร้อมกับแผนภูมิเวลาสำหรับการส่งและรับ

การส่งเริ่มต้นงานด้วยคำสั่งที่ใช้ SBUF เป็นรีจิสเตอร์รับข้อมูล สัญญาณ 'Write to SBUF' ก็จะบรรจุค่า '1' เข้าไปเป็นตำแหน่งที่เก้าในรีจิสเตอร์การเลื่อนส่งก็จะแสดงการถูกร้องขอให้ส่งข้อมูล การส่งข้อมูลที่ช่วง S1P1 ของวัฏจักรแมชชีน และจะตามด้วยบิทตัวต่อมาในช่วงเวลาของสัญญาณนาฬิกาที่หารด้วย 16 ที่ถูกตั้งที่ตัวนับ ดังนั้นแต่ละบิทจะถูกซิงค์ด้วยการหาร 16 ของตัวนับ ไม่ใช่ด้วยสัญญาณ 'Write to SBUF'

การส่งจะเริ่มด้วยการส่งแอกทีฟสัญญาณ SEND และใส่บิท Start เข้าที่ TXD ช่วงเวลาหลังจากนั้นหนึ่งบิท สัญญาณข้อมูลก็จะแอกทีฟ ซึ่งก็จะอีนาเบิ้ลการส่งบิทออกจากรีจิสเตอร์การเลื่อนส่งออกไปยังขา TXD พัลส์เลื่อนตัวบิทแรกจะเกิดขึ้นหลังเวลาทำงานแล้วหนึ่งบิท

ขณะที่ข้อมูลเลื่อนออกทางขวา ค่า '0'จะถูกใส่เข้าทางซ้าย เมื่อ MSB ของข้อมูลหนึ่งไบท์อยู่ที่ตำแหน่งเอาท์พุทของรีจิสเตอร์ตัวเลื่อน ขณะนั้น ค่า '1' จะถูก



รูปที่ 3.4 แสดงถึงแผนภูมิการใช้งานในโหมด 1 พร้อมกับแผนภูมิเวลาสำหรับสัญญาณการส่งและรับ

เริ่มบรรจุเข้าเป็นตำแหน่งที่เก้าที่เอาท์พุท หลังจากที่ MSB ถูกส่งออกไป และทุกตำแหน่งเมื่อถูกส่งไปแล้วที่เหลือในรีจิสเตอร์การเลื่อนจะเป็น '0' หมด สถานะแฟลกในหน่วยควบคุมการส่งก็จะเลื่อนเป็นตัวสุดท้าย และส่งสัญญาณ SEND ดิสเอเบิ้ล และการเซ็ต TI จะเกิดขึ้นในช่วงลูกที่ 10 ของการหาร 16 หลังจากการส่งสัญญาณ 'Write to SBUF'

การรับจะเริ่มงานด้วยการกระตุ้นจากการเปลี่ยนแปลง '1' เป็น '0' ที่ RXD สำหรับจุดนี้ RXD ก็จะถูก Sample ด้วยอัตรา 16 เท่าของอัตราบ็อดที่ถูกกำหนดเริ่มแรก เมื่อการส่งข้อมูลถูกรับได้ ตัวนับหาร 16 ก็จะถูกรีเซ็ต และค่า 01FFH ก็จะถูกเขียนเข้าไปในรีจิสเตอร์ตัวเลื่อน การรีเซ็ตตัวนับหาร 16 ก็จะเป็นการตั้งวนรอบด้วยการให้ขอบเขตของช่วงเวลาแต่ละบิทที่เข้ามา

16 คาบเวลาของตัวนับในแต่ละบิท จะเป็นเวลาที่เข้ามาในคาบที่ 16 ที่ตัวนับนับค่าที่ 7, 8 และ 9 จะเป็นช่วงเวลาของบิท เป็นการรับข้อมูลแต่บิทที่ Sample ค่าที่เข้ามาทาง RXD และค่าที่รับเข้ามาถูก Sample อย่างน้อย 2-3 ครั้ง การทำเช่นนี้จะเป็นการขจัด Noise ออกไป ถ้าค่าข้อมูลถูกรับในระหว่างช่วงเวลาบิทแรกมีไม่ใช่ค่า '0' วงจรการรับจะถูกรีเซ็ตและหน่วยรับก็จะกลับไปตรวจการเปลี่ยนแปลงจาก 0 ->1 ใหม่ ลักษณะงานเช่นนี้จะเป็นการป้องกันรับบิท Start ที่ผิดพลาดเข้ามาได้ ถ้าบิท Start ถูกรับเข้ามาถูกต้อง มันก็จะถูกเลื่อนเข้ารีจิสเตอร์ตัวเลื่อน และการรับข้อมูลก็จะเริ่มขึ้น

ขณะที่ข้อมูลเข้ามาทางขวา ค่า '1' จะถูกเลื่อนออกไปทางซ้าย เมื่อค่าบิท Start ถูกเลื่อนมาถึงทางซ้ายสุดในรีจิสเตอร์ตัวเลื่อน มันก็จะสั่งแฟลกในหน่วยควบคุมการรับให้เลื่อนอีกหนึ่งครั้งเป็นครั้งสุดท้าย และก็จะบรรจุข้อมูลเข้า SBUF และ RB8 (เพราะมีเก้าบิท) และเซ็ต RI สัญญาณการบรรจุเข้า SBUF , RB8 และการเซ็ต RI เป็น '1' จะปรากฏ ถ้าเพียงแต่กรณีใดต่อไปนี้ปรากฏในช่วงเวลาพัลส์การเลื่อนสุดท้ายเกิดขึ้น คือ

1. RI = 0

2. SM2 = 0 หรือ การรับ Stop bit = 1

ถ้าไม่เกิดทั้งสองกรณี การรับข้อมูลไบท์ก็จะล้มเหลว ถ้าเกิดทั้งสองกรณี ตัว Stop bit ก็จะไปเก็บที่ RB8 และข้อมูลไบท์ก็จะเข้า SBUF และ RI จะแอกทีฟสูงช่วงเวลานี้ไม่ว่าจะเกิดขึ้นทั้งสองกรณีหรือไม่ หน่วยควบคุมการรับก็จะกลับไปตรวจการเปลี่ยน

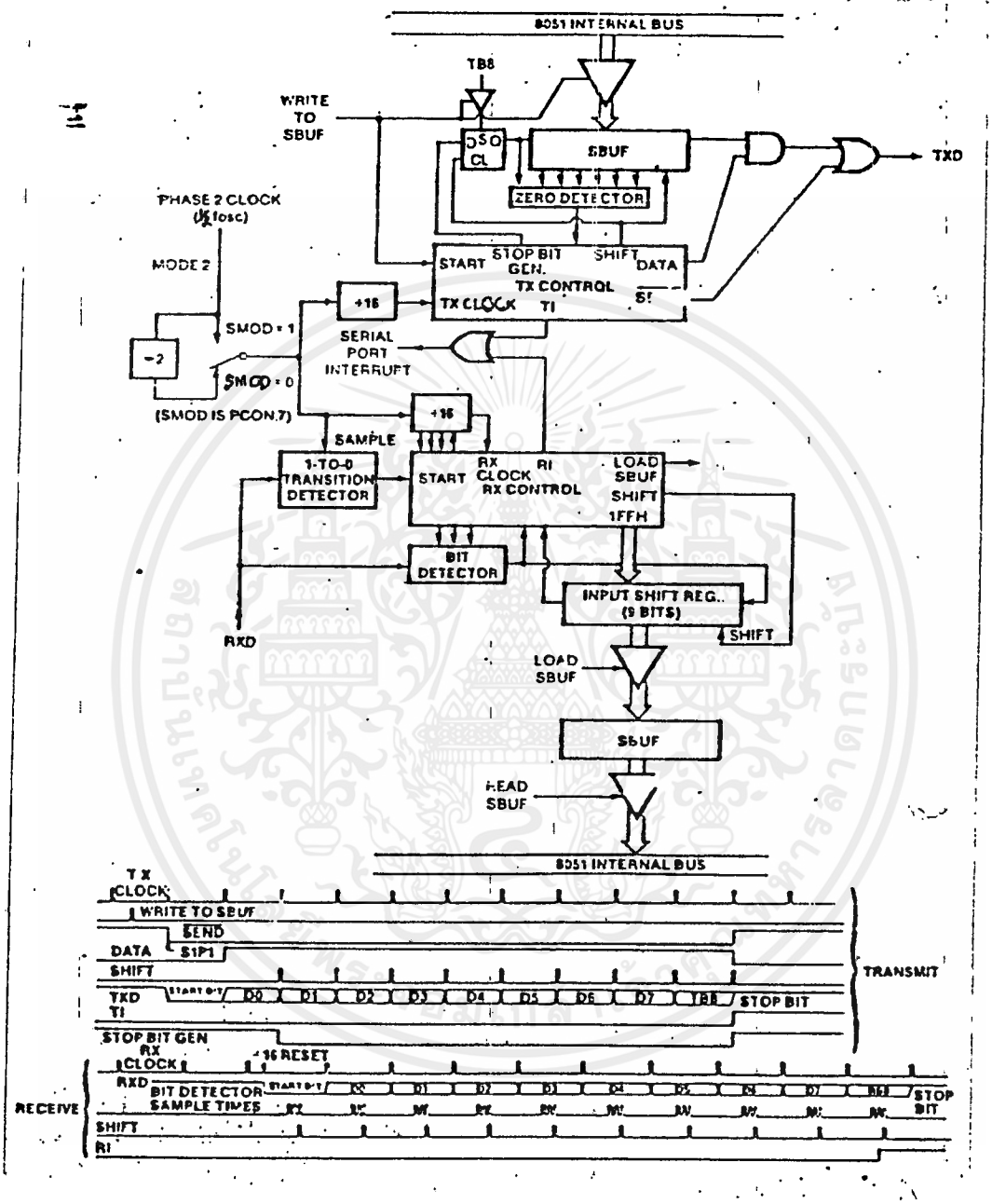
แปลง 1 - 0 ของการส่งข้อมูลผ่าน RXD ใหม่

Mode 2 and 3 จำนวน 11 บิทจะส่งออกที่ TXD และรับเข้าที่ RXD โดยมีบิท Start มีค่า '0' ข้อมูล 8 บิทมี LSB เป็นบิทแรก และโปรแกรมบิทที่เก้าได้ และบิท Stop มีค่า '1' การส่งข้อมูลบิทที่เก้า ใช้ TB8 เป็นตัวกำหนดค่า '0' หรือ '1' การรับข้อมูลบิทที่เก้าใช้ RB8 ใน SCON เป็นตัวรับ อัตราบ็อดสามารถโปรแกรมเลือกได้ทั้งแบบ 1/32 หรือ 1/64 ของความถี่ออสซิเลเตอร์ในโหมด 2 แต่โหมด 3 จะใช้ตัวแปรหลายค่าของอัตราบ็อด เกิดจากการใช้ตัวนับ 1 หรือ 2 ขึ้นอยู่กับสถานะ TCLK และ RCLK

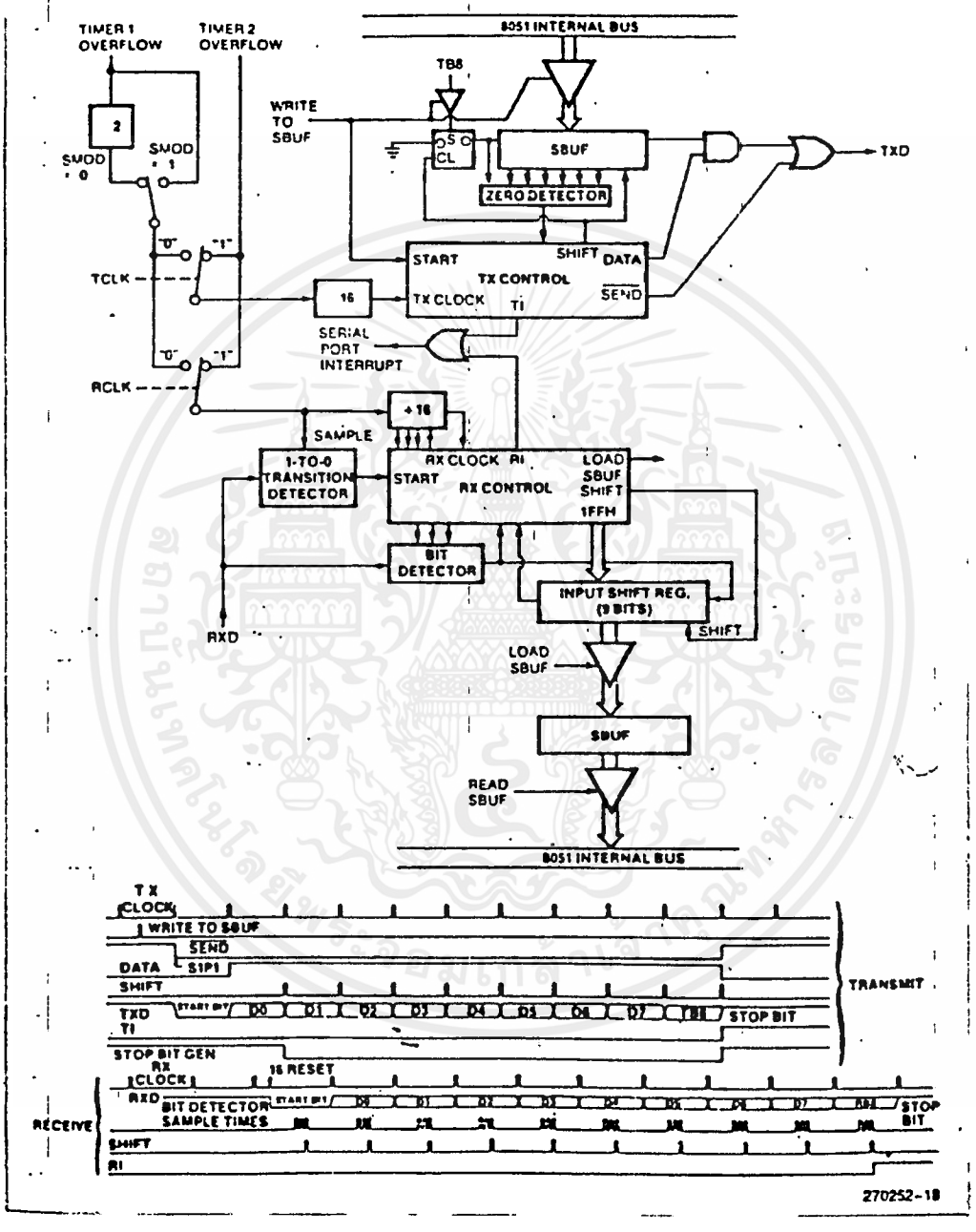
รูปที่ 3.5 และรูปที่ 3.6 แสดงแผนภูมิฟังก์ชันของพอร์ทอนุกรมในโหมด 2 และ 3 ส่วนของตัวรับจะทำงานเหมือนกับโหมด 1 ส่วนของตัวส่งต่างจากโหมดหนึ่ง 1 เพียงบิทที่เก้าของรีจิสเตอร์ตัวเลื่อนการส่ง

การส่งจะเริ่มด้วยคำสั่งใดๆ ที่ใช้ SBUF เป็นรีจิสเตอร์ตัวรับข้อมูล สัญญาณ 'Write to SBUF' ก็จะบรรจุ TB8 เข้าไปเป็นบิทที่เก้าของตำแหน่งในรีจิสเตอร์ตัวเลื่อนการส่งและแฟลกในหน่วยควบคุมการส่งก็จะให้การส่งถูกร้องขอ ช่วงการส่งจะเริ่มขึ้นที่ S1P1 ของวัฏจักรแมชชีนตัวต่อๆ มา ในการใช้ตัวนับหาร 16 ดังนั้น ช่วงแต่ละบิทจะซิงค์โคไนส์กับตัวนับหาร 16 โดยไม่ใช้สัญญาณของ 'Write to SBUF'

การส่งเริ่มด้วยสัญญาณ SEND แอกทีฟต่ำ ใส่บิท Start ออกที่ TXD หลังจากนั้นแต่ละบิทของข้อมูลในสัญญาณ DATA ก็จะแอกทีฟ ซึ่งจะอีนาเบิ้ลให้บิทของรีจิสเตอร์ตัวเลื่อนส่งออกที่ TXD ตามมา โดยตัวเลื่อนพัลส์ตัวแรกจะเกิดขึ้นหลังจากนั้นเล็กน้อย ที่สัญญาณนาฬิกาเลื่อนตัวแรก ถ้า '1' ถือเป็นบิท Stop จะใส่เข้าไปที่รีจิสเตอร์ตัวเลื่อนทางซ้ายสุดเป็นตำแหน่งบิทที่เก้า ดังนั้นหลังจากนั้นเล็กน้อยที่สัญญาณนาฬิกาเลื่อนตัวต่อมาจะใส่ค่า '0' เข้าไปเท่านั้น และทุกสัญญาณนาฬิกาจะเลื่อนเอาบิทออกทางขวา และใส่ค่า '0' เข้าทางซ้ายเมื่อ TB8 อยู่ที่ตำแหน่งเอาท์พุทของรีจิสเตอร์ตัวเลื่อน ดังนั้นบิท Stop จะส่งออกต่อ TB8 และทุกตำแหน่งในรีจิสเตอร์ตัวเลื่อนที่เหลือจะเป็นศูนย์หมดด้วยสถานะเช่นนี้จะทำให้แฟลกในหน่วยควบคุมการส่งเลื่อนเป็นครั้งสุดท้าย และให้สัญญาณ SEND ดิสเอเบิ้ลสูง และเซ็ต TI ด้วย ซึ่งจะเกิดขึ้นที่พัลส์ลูกที่ 11 ของตัวการ 16 สัญญาณนาฬิกาหลังจากส่งสโตรบสัญญาณ 'Write to SBUF'



รูปที่ 3.5 แสดงแผนภูมิฟังก์ชันของพอร์ทอนุกรมในโหมด 2



รูปที่ 3.6 แสดงแผนภูมิฟังก์ชันของพอร์ทอนุกรมในโหมด 3

การรับจะเริ่มทำงานด้วยการจับสัญญาณที่ขา RXD ช่วงเวลาการเปลี่ยนจาก '1' เป็น '0' สำหรับโหมดนี้ ตัว RXD จะถูก Sample ที่อัตรา 16 เท่าของอัตราบ็อดที่ถูกกกกำหนดมา เมื่อสภาวะการเปลี่ยนแปลงถูกรับได้ ตัวนับหาร 16 ก็จะเริ่มรีเซ็ตทันที และค่า 1FFH ก็จะเขียนเข้าไปที่รีจิสเตอร์ตัวเลื่อน

ที่ช่วงการนับลูกที่7,8และ 9ของแต่ละช่วงบิท ตัวดีเทคเตอร์จะ Sample ค่าบิทของสัญญาณที่เข้า RXD ค่าที่รับเข้ามาจะเป็นค่าที่คล้ายกับว่ามีการรับเข้า Sampling อย่างน้อย 2 ถึง 3 Sample ถ้าค่าถูกรับเข้าช่วงระหว่างบิทแรกไม่เป็น '0' วงจรตัวรับก็จะรีเซ็ต และหน่วยควบคุมก็จะกลับตรวจการเปลี่ยนแปลงจาก 1 เป็น 0 ใหม่ ถ้าบิท Start ถูกพิสูจน์ว่าถูกต้องมันก็จะเลื่อนเข้าสู่รีจิสเตอร์ตัวเลื่อนและการรับก็จะรับครบ Fram ของข้อมูลในโหมดนี้

ขณะที่ข้อมูลเข้ามาจากทางขวา ค่า '1'ก็จะถูกเลื่อนไปทางซ้ายออกไป เมื่อบิท Star ถูกเลื่อนมาถึงตำแหน่งทางซ้ายสุดของรีจิสเตอร์ตัวเลื่อน โดยในโหมด 2 และ 3 จะมีรีจิสเตอร์เก้า มันจะแฟลกให้หน่วยควบคุมการรับทำการเลื่อนครั้งสุดท้ายแล้วบรรจุค่าใน SBUF และ RB8 และเซ็ต RI สัญญาณการบรรจุ SBUF และ RB8 และการเซ็ต RI เป็น '1' จะถูกสร้างขึ้น ถ้าเพียงแต่เกิดกรณีใดกรณีหนึ่งต่อไปนี้ปรากฏในช่วงเวลาพัลส์การเลื่อนลูกสุดท้าย คือ

1. RI = 0

2. SM2 = 0 หรือ/และ การรับบิทที่เก้ามีค่า = 1

ถ้าทั้งสองกรณีไม่เกิดขึ้น การรับ Frame ของข้อมูลก็จะสูญหาย และ RI จะไม่เซ็ตถ้าทั้งสองกรณีเกิดขึ้น การรับบิทที่เก้าจะรับเข้า RB8 และแปดบิทแรกจะบรรจุเข้าใน SBUF ช่วงเวลาหนึ่งบิท หลังจากนั้นไม่ว่าจะได้นับข้อมูลหรือข้อมูลสูญหาย หน่วยควบคุมก็จะกลับตรวจสอบการเปลี่ยนแปลงค่า 1 เป็น 0 ที่อินพุทของ RXD ใหม่

บทที่ 4

สัญญาณต่างๆบนสล็อตของ PC

ภายใน IBM/PC ได้มีการออกแบบให้สามารถที่จะเพิ่มต่อวงจรอินเทอร์เฟสเข้าไปภายหลังได้ โดยผ่านทางสล็อตที่อยู่บนเมนบอร์ด (main board) ซึ่งมีจำนวน 5 สล็อต โดยแต่ละสล็อตจะมีจำนวนขาทั้งสิ้น 62 ขา แบ่งออกเป็นสองข้างๆละ 31 ขา แต่ละขาของสล็อตเหล่านี้จะเชื่อมต่อกับเส้นสัญญาณต่างๆบนเมนบอร์ด ซึ่งสัญญาณต่างๆของสล็อตประกอบด้วยสัญญาณต่อไปนี้

OSC (ออสซิเลเตอร์) ขานี้เป็นเอาท์พุทที่เชื่อมต่อกับสัญญาแคล็อกที่มีค่าความถี่สูงสุดบนเมนบอร์ด คือ 14.31818 เมกกะเฮิร์ท ซึ่งมีคาบเวลาประมาณ 70 นาโนวินาที (nanosec)

CLK (สัญญาณคล็อก) ขาสัญญาณนี้เป็นเอาท์พุท ซึ่งต่อกับสัญญาณคล็อกที่ถูกสร้างขึ้นโดยการหารสัญญาณออสซิเลเตอร์ด้วย 3 ทำให้ได้ความถี่ประมาณ 4.77 เมกกะเฮิร์ท หรือ มีช่วงเวลาในหนึ่งคาบ(ช่วงเวลาของคล็อกหนึ่งลูก)เท่ากับ 210 นาโนวินาที สัญญาณนี้เป็นสัญญาณที่ถูกใช้เป็นคล็อกของระบบ

RESET DRV (สัญญาณรีเซ็ต) ขาสัญญาณนี้เป็นเอาท์พุทใช้สำหรับรีเซ็ตระบบในขณะเปิดเครื่องหรือขณะที่แหล่งจ่ายไฟเลี้ยงขาด หรือไฟตก สัญญาณนี้จะแอคทีฟเมื่อลอจิก "1" จนกว่าระบบต่างๆ ภายใน IBM/PC พร้อมที่จะทำงานได้ จากนั้นสัญญาณนี้ก็จะเปลี่ยนกลับเป็นลอจิก "0"

A0-A19 (บัสแอดเดรส) ขาสัญญาณนี้เป็นเอาท์พุท ซึ่งใช้สำหรับกำหนดแอดเดรสของหน่วยความจำหรืออุปกรณ์ I/O ที่ 8088 ต้องการจะติดต่อด้วย จำนวนแอดเดรส 20 เส้น สามารถอ้างหน่วยความจำได้ถึง 1 เมกกะไบท์ (Mbyte) แต่จะมีแอดเดรสบางส่วนที่ถูกใช้งานโดย IBM/PC อยู่ก่อนแล้ว

การอ้างแอดเดรสของพอร์ท I/O นั้น จะใช้เพียง 16 เส้น คือ A0 - A15 ซึ่งทำให้อ้างแอดเดรสของพอร์ทได้ 64K พอร์ท แต่ภายใน IBM/PC ใช้เส้นแอดเดรสในการอ้างแอดเดรสของพอร์ทเพียง 10 เส้น คือจาก A0 - A9 และค่าแอดเดรสที่สามารถใช้งานต้องอยู่ในช่วง 0200H - 03FFH

D0-D7 (บัสข้อมูล) ขาสัญญาณนี้จะเป็นแบบสองทิศทาง (Bi-Directional)

ซึ่งติดต่อกับบัสข้อมูลของระบบ เพื่อทำหน้าที่ในการส่งผ่านข้อมูลระหว่างพอร์ท I/O กับ IBM/PC

ALE (Address Latch Enable) ขาสัญญาณนี้เป็นเอาท์พุทที่ถูกสร้างขึ้นเพื่อใช้สำหรับแสดงการเริ่มต้นของบัสไซเคิล และ แสดงให้อุปกรณ์ภายนอกทราบว่าแอดเดรสที่ 8088 ต้องการติดต่อด้วยนั้นถูกส่งออกมาบนบัสแอดเดรสแล้ว โดยที่สัญญาณนี้จะเปลี่ยนจากลอจิก "1" เป็น "0" เมื่อค่าแอดเดรสที่ถูกต้องถูกส่งออกมาบนบัสข้อมูลเรียบร้อยแล้ว

I/O CHRDY (I/O Channel Ready) ขาสัญญาณนี้เป็นอินพุทจะถูกนำให้เป็น "0" ด้วยหน่วยความจำหรืออุปกรณ์อินพุท/เอาท์พุท การใช้สัญญาณนี้ก็เพื่อให้อุปกรณ์ I/O ที่ช้าจะได้ติดต่อกับระบบด้วยการส่งสัญญาณมายังซีพียู เพื่อชิงโคนัสระบบได้

$\overline{\text{I/O CHCK}}$ (I/O Channel Check) ขาสัญญาณนี้เป็นอินพุทเป็นสัญญาณตรวจสอบของ I/O เพื่อบอกข้อมูลกับระบบเช่นเดียวกับการตรวจสอบพาริตี้ ดังนั้น ถ้าบน I/O มีข้อผิดพลาด สัญญาณนี้จะแอกทีฟเพื่อให้ส่งสัญญาณเตือนในลักษณะ parity error

IRQ2-IRQ7 (สัญญาณการขออินเทอร์รัพต์) เป็นขาอินพุทที่ใช้สำหรับการขออินเทอร์รัพต์จาก 8088 เพื่อให้สัญญาณ INT เข้าสู่ไมโครโปรเซสเซอร์ โดย IRQ2 มีลำดับความสำคัญสูงสุด และ IRQ7 มีลำดับความสำคัญน้อยที่สุด

$\overline{\text{IOR}}$ (I/O Read) ขาสัญญาณนี้เป็นเอาท์พุท แอกทีฟที่ลอจิก "0" เพื่อใช้ในการแสดงว่าบัสไซเคิลที่เกิดขึ้นนี้ เป็นบัสไซเคิลของการอ่านข้อมูลจากพอร์ท เพื่อให้พอร์ท I/O ที่มีแอดเดรสตรงกับแอดเดรสบนบัสแอดเดรสนั้นส่งข้อมูลออกมาบนบัสข้อมูล โดยข้อมูลจะต้องถูกส่งออกมาบนบัสข้อมูลก่อนขอบขาขึ้นของสัญญาณ IOR ประมาณ 30 นาโนวินาที

$\overline{\text{IOW}}$ (I/O Write) ขาสัญญาณนี้เป็นเอาท์พุท แอกทีฟที่ลอจิก "0" เพื่อใช้ในการแสดงว่า บัสไซเคิลที่เกิดขึ้นนี้ เป็นบัสไซเคิลของการเขียนลงบนพอร์ท I/O เพื่อให้พอร์ท I/O ที่มีแอดเดรสตรงกับแอดเดรสบนบัสแอดเดรสนั้น รับข้อมูลที่อยู่บนบัสข้อมูลไปเก็บไว้

$\overline{\text{MEMR}}$ (Memmory Read) ขานี้เป็นสัญญาณเอาพุท จะแอกทีฟที่ลอจิก "0" ในระหว่างบัสไซเคิลของการอ่านข้อมูลจากหน่วยความจำของซีพียู เพื่อให้หน่วยความจำที่มีแอดเดรสตรงกับค่าแอดเดรสบนบัสแอดเดรสนั้น ทำการส่งข้อมูลออกมาบนบัสข้อมูล

$\overline{\text{MEMW}}$ (Memmory Write) ขานี้เป็นสัญญาณเอาพุท จะแอกทีฟที่ลอจิก "0" ในระหว่างบัสไซเคิลของการเขียนข้อมูลลงในหน่วยความจำของซีพียู เพื่อให้หน่วยความจำที่มีแอดเดรสตรงกับค่าแอดเดรสบนบัสแอดเดรสนั้น ทำการรับข้อมูลที่อยู่บนบัสข้อมูลไป

เก็บไว้

DRQ1-DRQ3 (DMA Request) เป็นสัญญาณอินพุท แอกทีฟที่ลอจิก "1" ซึ่งอุปกรณ์ภายนอกสามารถใช้ในการขอ DMA จากระบบ โดยการป้อนระดับสัญญาณลอจิก "1" ให้ กับขา DRQ ขาใดขาหนึ่ง

$\overline{\text{DACK0}}$-$\overline{\text{DACK3}}$ (DMA Acknowledge) ขานี้เป็นสัญญาณเอาพุทแอกทีฟที่ลอจิก "0" ถูกสร้างขึ้นเพื่อแสดงให้วงจรภายนอกที่ขอ DMA ทราบว่าการขอ DMA นั้นได้รับการตอบสนองแล้ว

AEN (Address Enable) สัญญาณนี้เป็นเอาพุทที่ใช้ในการแสดงว่าบัสไซเคิลที่เกิดขึ้นในช่วงเวลาที่สัญญาณ AEN แอกทีฟ (ลอจิก "1") นั้นเป็นบัสไซเคิลของขบวนการ DMA สำหรับบนเมนบอร์ดของ IBM/PC นั้น จะใช้สัญญาณนี้ในการดิสเอเบิล (disable) พอร์ท I/O ต่างๆ ที่ไม่เกี่ยวข้องกับขบวนการ DMA ที่เกิดขึ้นนี้ เพราะในระหว่างขบวนการ DMA นั้น จะมีการส่งแอดเดรสของหน่วยความจำออกมาบนบัสแอดเดรส และ จะทำให้สัญญาณ IOR หรือ IOW แอกทีฟด้วย ดังนั้น ถ้าไม่ทำการดิสเอเบิลพอร์ท I/O ที่ไม่เกี่ยวข้องไว้ ก็จะทำให้พอร์ท I/O ที่มีแอดเดรสตรงกับค่าแอดเดรสบนบัสแอดเดรสนั้น ทำการอ่านหรือส่งข้อมูลออกมาบนลัสข้อมูลทำให้เกิดความผิดพลาดขึ้นได้

T/C (Terminal Count) สัญญาณนี้จะแอกทีฟเมื่อจำนวนไบท์ ในการส่งผ่านข้อมูลของขบวนการ DMA ในแชนแนลใดแชนแนลหนึ่งครบตามจำนวนที่กำหนดไว้

บัสของแหล่งจ่ายไฟของระบบ

+5Vdc ขานี้ต่อกับแหล่งจ่ายไฟ DC +5V ของระบบ โดยจะมีค่าความเที่ยงตรง 5%

+12Vdc ขานี้ต่อกับแหล่งจ่ายไฟ DC +12V ของระบบ โดยจะมีค่าความเที่ยงตรง 5%

-5Vdc ขานี้ต่อกับแหล่งจ่ายไฟ DC -5V ของระบบ โดยจะมีค่าความเที่ยงตรง 5%

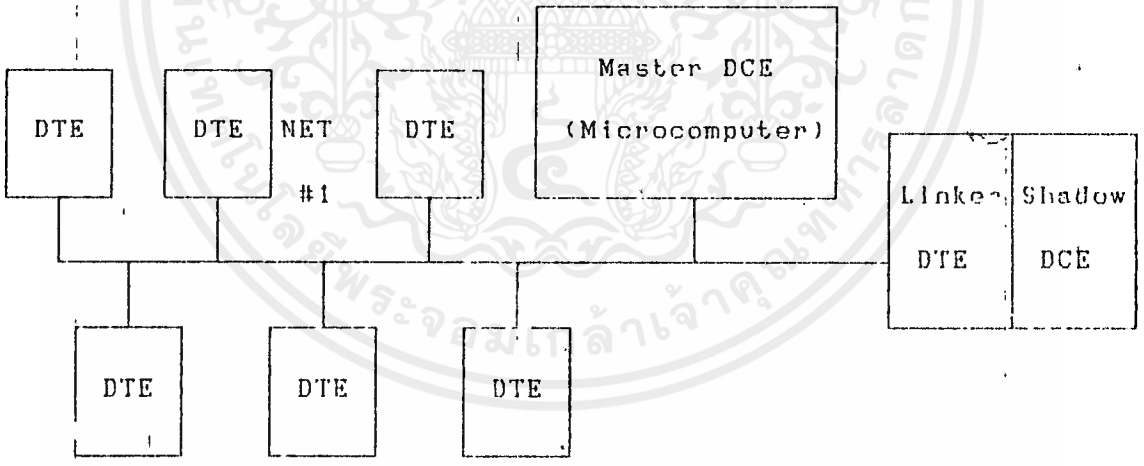
-12Vdc ขานี้ต่อกับแหล่งจ่ายไฟ DC -12V ของระบบ โดยจะมีค่าความเที่ยงตรง 5%

GND ขานี้จะต่อเข้ากับกราวน์ของระบบ

บทที่ 5

ผลการทดลอง

ลักษณะของระบบการควบคุมจะมี DTE และ ตัวควบคุม หลายตัวอยู่บนสายสัญญาณเดียวกันซึ่งจะใช้เป็นทั้งสายรับและสายส่ง โดยที่แต่ละตัวจะมีลักษณะเป็น half duplex เพื่อไม่ให้มีการรับสัญญาณของตัวเองขณะที่มีการส่ง ในการส่งข้อมูลตัวควบคุมทำหน้าที่ส่งข้อมูลการสั่งงานหรือตรวจสอบสถานะหรือการเริ่มทำงานของระบบให้กับ DTE แล้ว DTE จะส่งข้อมูลตอบรับกลับมาโดยใช้การส่งแบบ Asynchronous ความเร็วไม่น้อยกว่า 19.2 Kbit และ สายส่งสัญญาณเป็นแบบสองสายคือสายสัญญาณและสายกราวน์ หรือสามสาย คือ สายสัญญาณ สายสัญญาณไม่ว่าง และสายกราวน์ หรือ สี่สาย คือ สายสัญญาณ สายสัญญาณไม่ว่าง สายกราวน์ และ สายไฟบวก 5 volt ส่วนระยะเวลาช่วงห่างของการส่งข้อมูลแต่ละชุดไม่เกิน 250 millisecond หากเกินนี้ถือว่าเป็น noise



เราสามารถออกแบบและกำหนด protocal ของระบบได้ ดังนี้

5.1 HARDWARE

ระบบ hardware จะมี master 1 ตัว เป็น card ที่ใช้ต่อกับ PC จาก master จะต่อ กับ network ซึ่งมี terminal ได้ถึง 126 ตัว (ใช้ address แบบ 7 บิท)

terminal แบ่งออกได้เป็น 2 ประเภทใหญ่ๆ คือ ตัวควบคุม (Controller) และ DTE (Data Terminal Equipment)

DTE เป็นอุปกรณ์ปลายที่สามารถให้ผู้ใช้ควบคุม เช่น การเปิด-ปิด อุปกรณ์ได้โดยตรง (ใน DTE ตัวหนึ่งมี 8 แชเนล) ซึ่งในที่นี้ จำลองอุปกรณ์ภายในบ้านด้วย LED และ สามารถนำ DTE นี้ไปติดตั้งตามจุดต่างๆได้ นอกจากนี้ยังสามารถถูกควบคุมจากจุดอื่นๆโดยผ่านทางตัวควบคุมได้ ซึ่งมีลำดับการทำงานดังโฟลว์ชาร์ตรูปที่ 5.1 และ มีวงจรการเชื่อมต่อดังรูปที่ 5.2

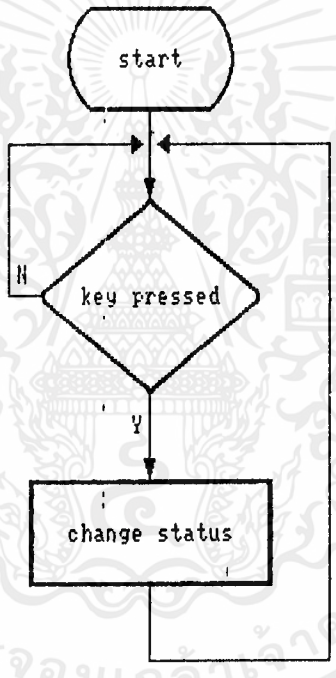
ตัวควบคุม ทำหน้าที่ส่งข้อมูลการสั่งงานและตรวจสอบสถานะของ DTE ทุกตัวบนโครงข่าย และ บางตัวทำตัวเป็น master ด้วย

ลักษณะของระบบควบคุมจะมี ตัวควบคุม และ อุปกรณ์ปลาย หลายตัวจำนวนไม่เกิน 32 ตัวเชื่อมต่ออยู่บนสายสัญญาณเดียวกัน ซึ่งใช้เป็นทั้งสายรับและสายส่งโดยที่แต่ละตัวมีลักษณะเป็น half duplex เพื่อไม่ให้มีการรับสัญญาณของตัวเองขณะที่มีการส่งข้อมูลบนโครงข่ายจะมีตัวควบคุมสำหรับควบคุมการทำงานของอุปกรณ์ปลาย และสำหรับเก็บรวบรวมข้อมูล โดยตัวควบคุมจะสอบถามไปยังแต่ละ node เป็นระยะๆเพื่อรวบรวมข้อมูลล่าสุดเกี่ยวกับกระบวนการทำงานของอุปกรณ์ปลายแต่ละตัว สลับกับการคอยตรวจสอบการกดคีย์เพื่อการควบคุมการทำงานจากผู้ใช้ โดยแสดงลำดับการทำงานดังโฟลว์ชาร์ตรูปที่ 5.3

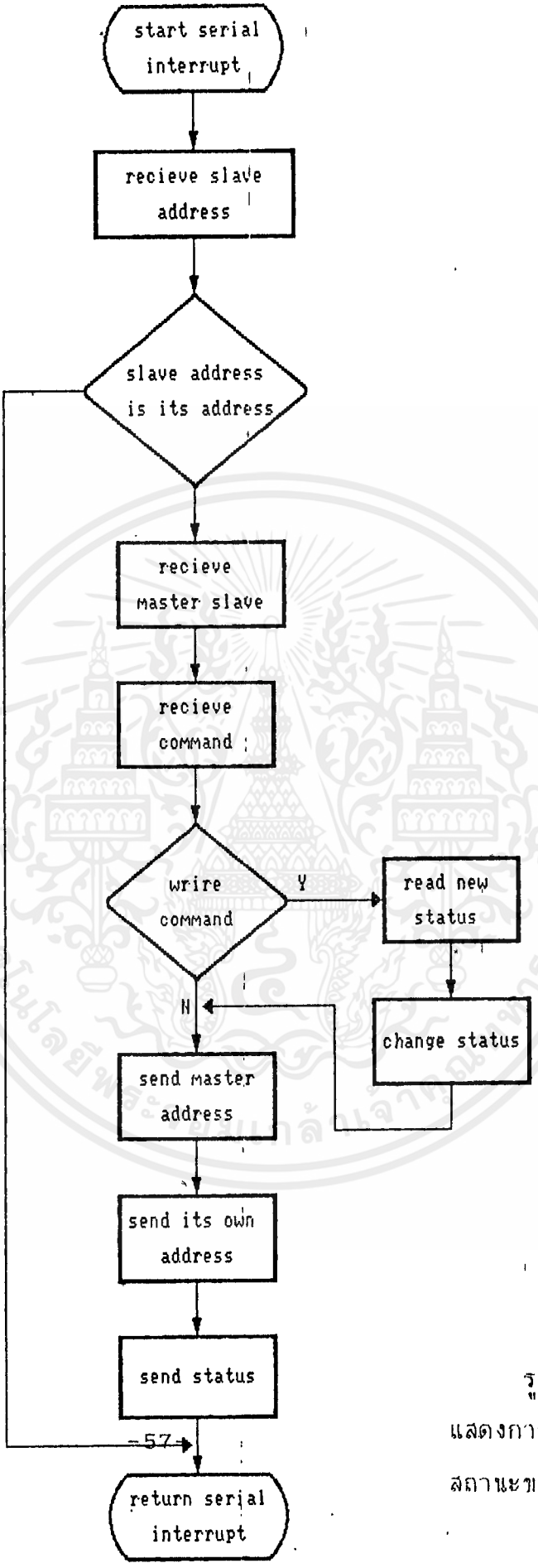
ในช่วงที่การ์ดทำการ polling อุปกรณ์บนสายสื่อสาร ตัว 8031 จะทำการรับส่งข้อมูลแบบอนุกรมโดยผ่านทางขา TXD และ RXD ในการส่งและรับตามลำดับ ซึ่งมีลำดับการทำงานดังโฟลว์ชาร์ตรูปที่ 5.4 ส่วนการตรวจสอบการกดคีย์จะมีการเชื่อมต่อคีย์เข้ากับตัว 8031 เป็นแบบเมตริกซ์ โดยจะส่งลอจิก "0" ออกทางพอร์ท 1 ด้านล่างทีละขาสลับกัน และ ทำการรับสถานะของการกดคีย์เข้าทางพอร์ท 3 ด้านล่าง ถ้ามีการกดคีย์ใดสถานะของคีย์นั้นจะมีค่าลอจิกเป็น "0" และ ลักษณะการเชื่อมต่อของตัวควบคุม แสดงดังรูปที่ 5.5

การ์ดเชื่อมต่อกับ PC

ส่วนเชื่อมต่อ PC เป็นการ์ดซึ่งติดตั้งอยู่บนสล็อตของเครื่อง PC ซึ่งช่วยให้ User สามารถใช้ PC ควบคุมอุปกรณ์ต่างๆบนโครงข่าย หรือทำตัวเป็นตัวควบคุมหลัก (Master) ของระบบได้ โดยการ์ดจะทำหน้าที่หลัก คือ คอย polling อุปกรณ์ทุกตัวบนสายสื่อสารทุกระยะว่ามีข่าวสารที่ต้องการรับส่งหรือไม่ ส่วนหน้าที่รอง คือ การตอบสนองสัญญาณอินเทอร์รัพต์จาก PC ซึ่ง PC จะส่งมาเมื่อต้องการอ่านค่าสถานะหรือเปลี่ยนสถานะ

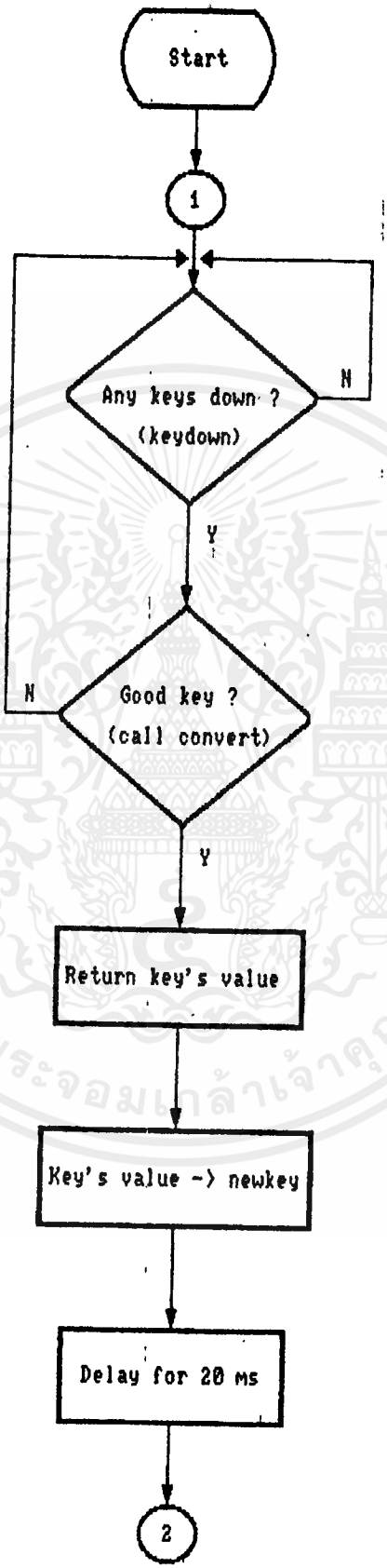


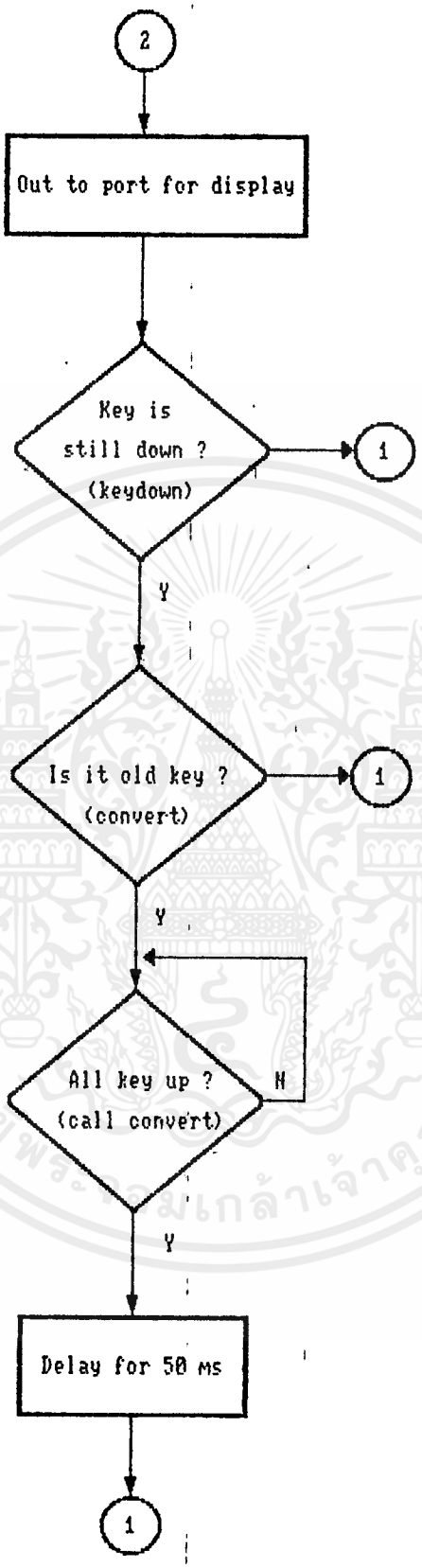
รูปที่ 5.1(a) แสดงการทำงานหลักของ DTE



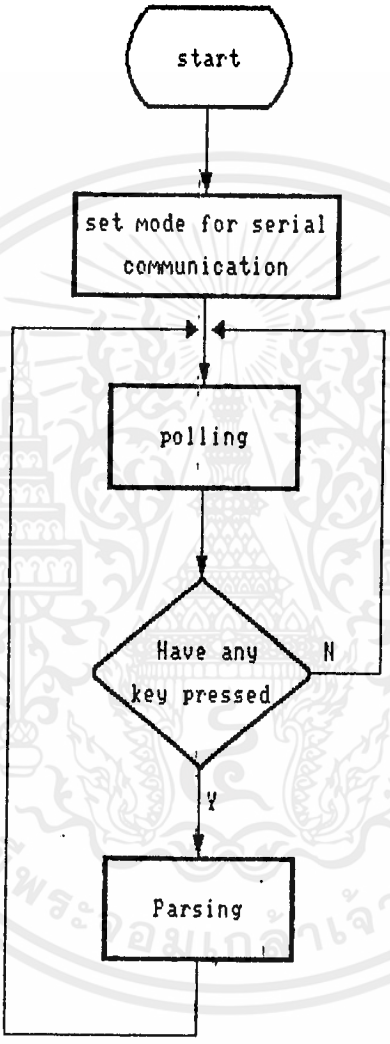
รูปที่ 5.1 (b)
แสดงการทำงานส่วนเปลี่ยน
สถานะของ LED

FLOWCHART PROGRAM DIEKEY

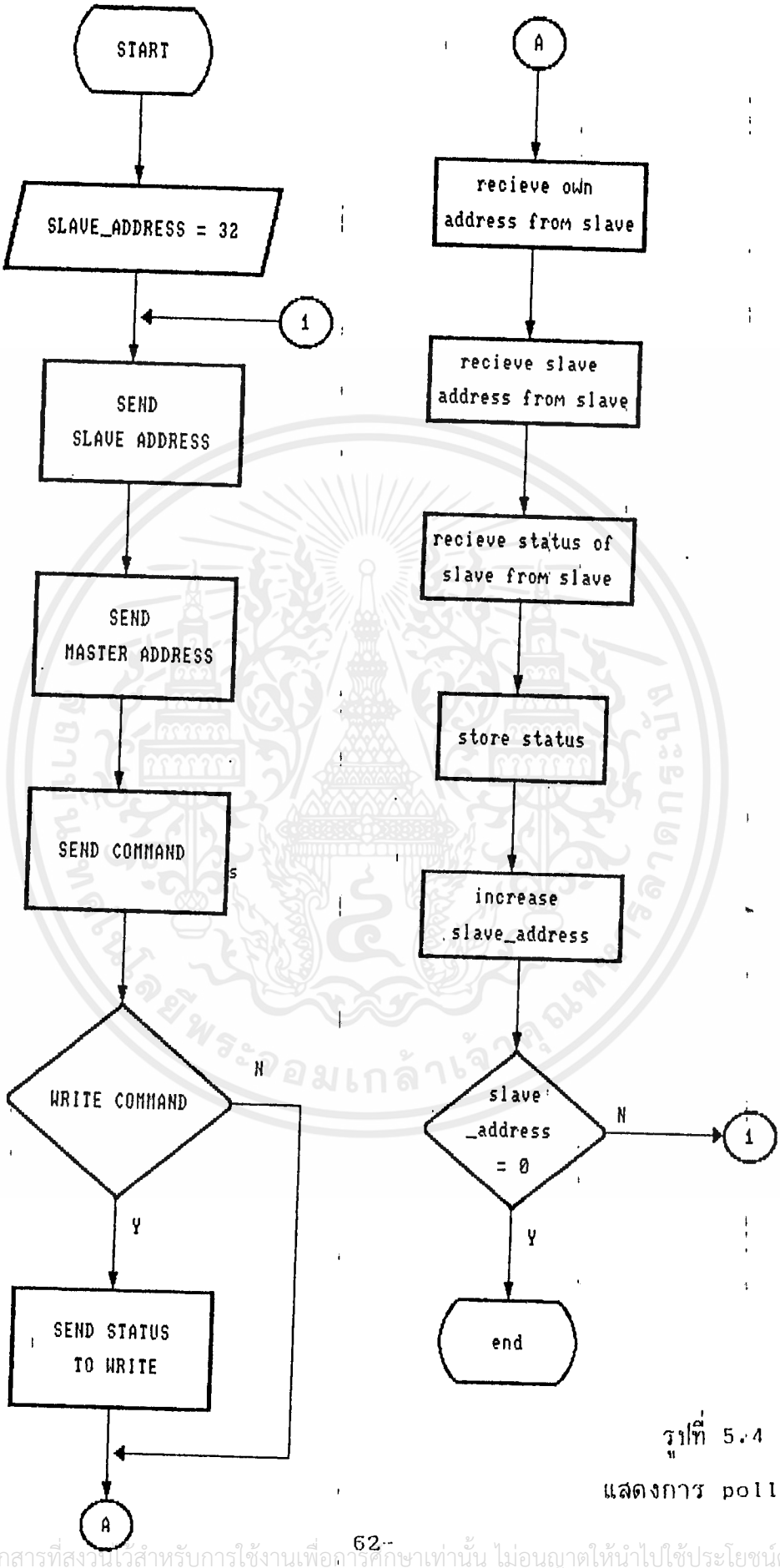




รูปที่ 5.1 (c) แสดงการตรวจสอบการกดคีย์ของ DTE



รูปที่ 5.3 แสดงการทำงานหลักของตัวควบคุม



รูปที่ 5.4

แสดงการ polling

ของอุปกรณ์ (DTE) บางตัว หลังจากเสร็จสิ้นการอินเทอร์รัพต์การ์ดก็จะทำการ polling อุปกรณ์บนสายสื่อสารเช่นเดิม

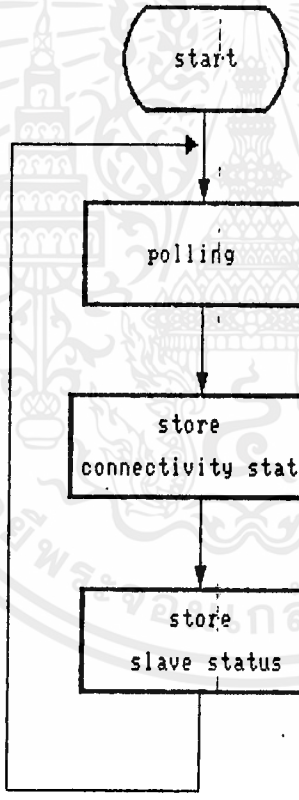
ในช่วงที่การ์ดทำงานตามหน้าที่หลัก คือ ช่วงที่การ์ดทำการ polling อุปกรณ์บนสายสื่อสาร ตัว 8031 จะทำการรับส่งข้อมูลแบบอนุกรมโดยผ่านทางขา TXD และ RXD ในการส่งและรับ ตามลำดับ ซึ่งมีลำดับการทำงานดังโฟลว์ชาร์ต รูปที่ 5.4

จากรูปที่ 5.6 ในช่วงที่การ์ดมีการติดต่อกับหน่วยประมวลผลกลาง(CPU) ของ PC

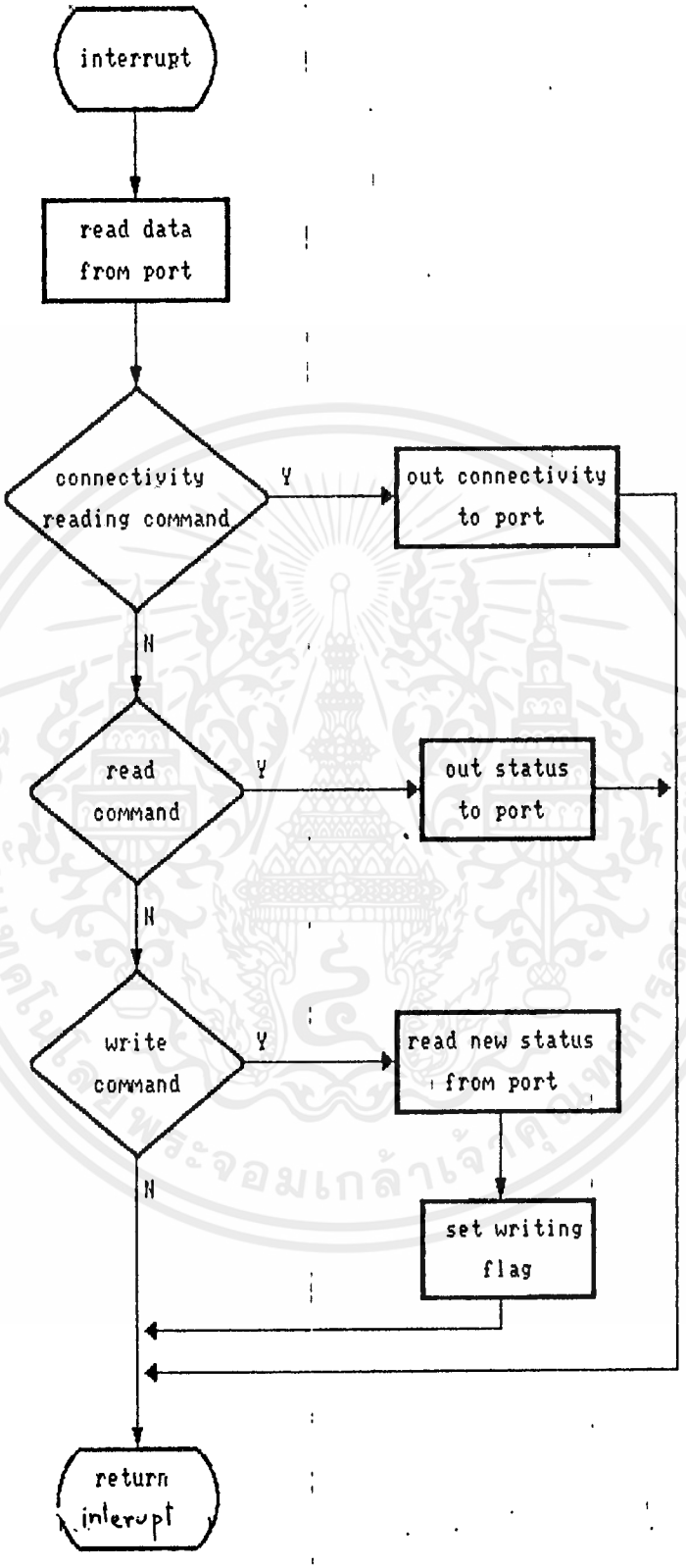
เมื่อ CPU ต้องการส่งข้อมูลให้แก่การ์ด (ข้อมูลจะถูกนำไปเก็บไว้ในหน่วยความจำของ 8031) แอดเดรสของพอร์ทที่ต้องการติดต่อด้วยจะถูกส่งมาบนบัสของสล็อต และจะถูกนำมาผ่านส่วน Decoder ถ้าแอดเดรสตรงกับค่า 217H ก็จะให้เอาท์พุทเป็นลอจิก "0" พร้อมกับสัญญาณ IOW กลายเป็นลอจิก"0" สัญญาณทั้งสองนี้จะถูกผ่าน OR gate แล้วนำสัญญาณที่ได้ไปควบคุมการทำงานตัว 74LS374 ให้ทำการแลทช์ข้อมูลที่ถูกส่งมารออยู่แล้วที่ขาดาต้าบัสไว้ และ สัญญาณเดียวกันนี้จะถูกนำไปผ่านดีฟลิปฟลอป(D flip-flop) เพื่อสร้างสัญญาณอินเทอร์รัพต์ให้แก่ขา INT0 ของตัว 8031 และ เข้าทำงานในอินเทอร์รัพต์รูทีน ดังโฟลว์ชาร์ต รูปที่ 5.7 เริ่มด้วยการควบคุมสัญญาณด้วยซอฟแวร์ให้ 74LS244 ทำงานเป็นบัฟเฟอร์ผ่านข้อมูลที่ถูกแลทช์ไว้ที่เอาท์พุทของ 74LS374 ให้แก่ขาทั้งแปดของพอร์ท 1 และ 8031 รับข้อมูลเข้ามา

หลังจากนั้น 8031 จะตอบรับการส่งข้อมูลของ CPU นั่นคือช่วงที่ CPU รับข้อมูลจากการ์ด โดย 8031 จะส่งข้อมูลออกมารอที่พอร์ท 1 ซึ่งเป็นจังหวะเดียวกัน ที่ PC ส่งสัญญาณ IOR เป็น "0" มาควบคุมตัว 74LS244 ให้ทำงานเป็น buffer ในทิศทางกลับกันกับ 74LS244 ตัวแรก ทำให้มีข้อมูลที่ถูกต้องรออยู่ที่ขาดาต้าบัสของสล็อต และ PC ทำการรับข้อมูลที่ขาดาต้าก่อนที่สัญญาณ IOR จะกลายเป็น "1" และก่อนสิ้นสุดอินเทอร์รัพต์รูทีนจะทำการเคลียร์ดีฟลิปฟลอบด้วยซอฟแวร์เพื่อให้สัญญาณ INT0 มีสถานะเป็น "1" เพื่อรอรับการถูกอินเทอร์รัพต์ครั้งต่อไป

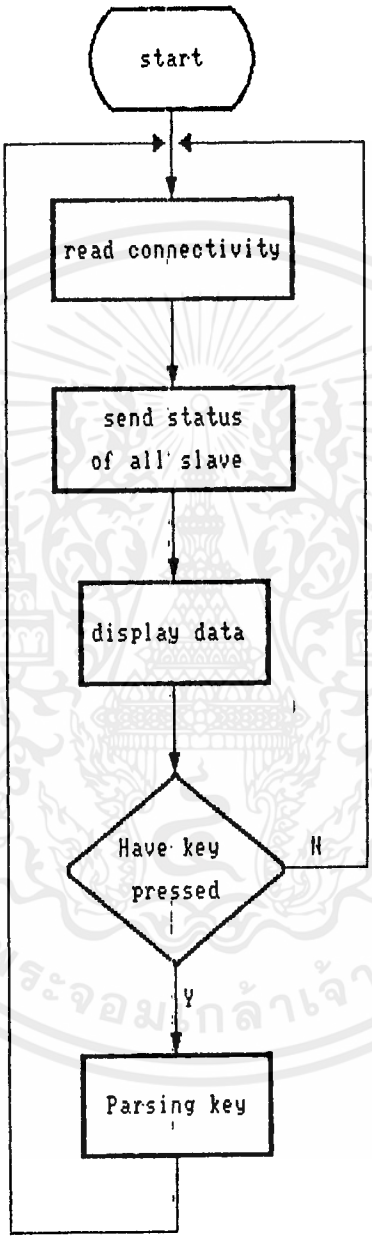
เนื่องจาก 8031 เป็นซีพียูตระกูล MCS-51 ที่ไม่มีหน่วยความจำรอมภายใน ดังนั้น จึงต้องมีการเชื่อมต่อ 8031 เข้ากับหน่วยความจำรอมภายนอกดังรูปที่ 5.2 , 5.5 และรูปที่ 5.6 ในที่นี้จะใช้หน่วยความจำรอมขนาด 2 กิโลไบท์โดยขา 31 ของ 8031 ซึ่งเป็นขา EA (External access) ถูกต่อลงกราวด์ เพื่อให้สามารถเข้าถึงหน่วยความจำ



รูปที่ 5.7 (a) แสดงการทำงานหลักของการ์ดเชื่อมต่อ PC

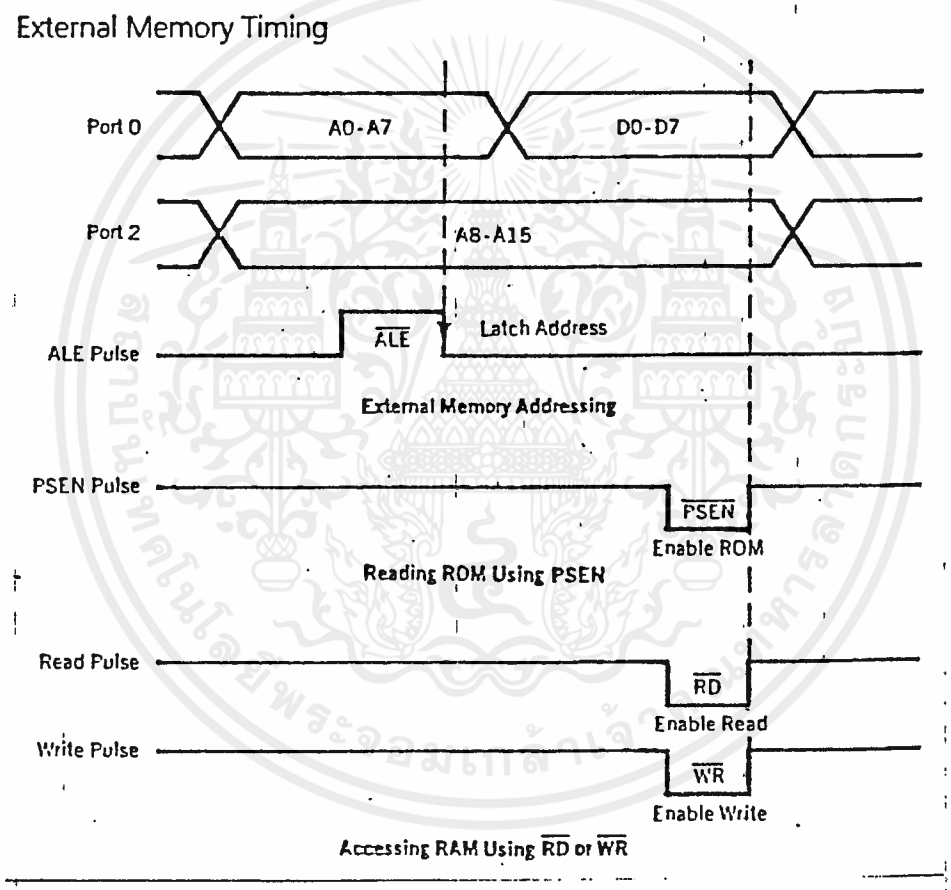


รูปที่ 5.7 (b) แสดงการทำงานในช่วงที่การ์ดมีการติดต่อกับ PC



รูปที่ 5.7 (c) แสดงการทำงานของ PC

รอมภายนอกได้ ระหว่างที่มีการเข้าถึงหน่วยความจำภายนอก พอร์ท 0 เป็นพอร์ทที่ให้สัญญาณแอดเดรสกับข้อมูล(AD0 - AD7) ด้านต่ำมัลติเพลกซ์กันออกมา ซึ่งการแยกเอาเฉพาะสัญญาณแอดเดรสออกมาให้หน่วยความจำ จะต้องใช้ 74LS373 เป็นตัวแลทช์สัญญาณออกมา โดยใช้สัญญาณ ALE จาก 8031 เป็นตัวควบคุมการทำงานของ 74LS373 ผลที่ได้จาก 74LS373 จะเป็นสัญญาณแอดเดรสอย่างเดียวและต่อตรงเข้าไปยังหน่วยความจำรอม ส่วนข้อมูลที่ได้จากหน่วยความจำรอมจะถูกต่อโดยตรงเข้ากับพอร์ท 0 สำหรับพอร์ท 2 จะเป็นสัญญาณแอดเดรสไบท์สูง (A8 - A15) และเป็นสัญญาณที่ไม่ได้มีการมัลติเพล็กซ์กับสัญญาณอื่นจึงต่อได้โดยตรง ซึ่งช่วงจังหวะการทำงานแสดง ดังรูปที่ 5.8

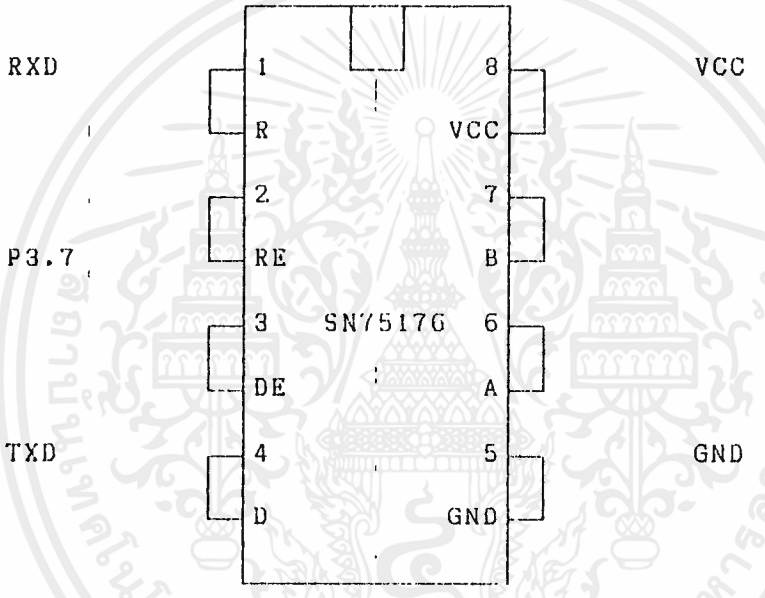


รูป 5.8 ผังเวลาของหน่วยความจำภายนอก

จากแบบผังของเวลา รูปที่ 5.8 ในช่วงแรกสัญญาณที่ออกมาจากพอร์ท 0 และพอร์ท 2 จะเป็นสัญญาณแอดเดรส สัญญาณแอดเดรสของพอร์ท 0 จะถูกแลทช์ด้วย 74LS373 ในช่วงเวลาของลงของพัลส์ของสัญญาณ ALE จากนั้นพอร์ท 0 จะเปลี่ยนมาเป็นบัสข้อมูล (D0 - D7) ในขณะนี้ ถ้าเป็นคำสั่งอ่านข้อมูลจากหน่วยความจำ EPROM สัญญาณ

PSEN (Program Store Enable) จาก 8031 จะเป็นตัวกำหนดว่าที่ขอบขาขึ้นของสัญญาณ PSEN 8031 จะทำการรับข้อมูลที่ได้มาจาก EPROM เข้าไป

นอกจากนี้ เนื่องจากในระบบโครงข่ายใช้สายคู่เดียวในการทั้งรับและส่งข้อมูลของเทอร์มินอลทุกตัวในระบบ ดังนั้น อุปกรณ์ทุกตัวที่อยู่บนสายสื่อสารจึงต้องมีส่วนเชื่อมต่อ (interface) ตัวมันเองกับสายสื่อสาร ซึ่งในที่นี้ใช้ไอซีเบอร์ SN75176 โดยทำหน้าที่ควบคุมทิศทางการไหลของข้อมูล ป้องกันมิให้สัญญาณที่ส่งออกไปเข้ามารบกวนการรับสัญญาณ ซึ่งการต่อเข้ากับวงจรของการ์ด ดังรูป



จากรูป ขา 1 และ ขา 4 ของตัว SN75176 ถูกต่อเข้ากับขารับและขาส่งสัญญาณ (RXD และ TXD) ตามลำดับ สำหรับการควบคุมทิศทางการไหลของข้อมูลบนสายสื่อสารจะถูกควบคุมจากขา P3.7 ของ 8031 โดยขา 2 และ ขา 3 ของ SN75176 ถูกต่อเข้ากับขา P3.7 เมื่อ ขา RE เป็นลอจิก "0" จะควบคุมให้การไหลของข้อมูลมีทิศทางในการรับ และเมื่อขา DE เป็นลอจิก "1" จะควบคุมให้การไหลของข้อมูลมีทิศทางในการส่ง ส่วนขา A และ ขา B จะถูกต่อเข้ากับสายสื่อสารของโครงข่าย

NETWORK HARDWARE

ในระบบ network นี้ ใช้สาย 2 เส้น คือ สายกราวน์ และสายสัญญาณ ในการส่งจะมีการ แปลงสัญญาณจาก 5 โวลต์ ไปเป็น 12 โวลต์ และ ในการรับสัญญาณจะ

ต้องแปลงกลับมาเป็น 5 โวลต์ ตามเดิม

5.2 NETWORK ACCESS CONTROL

จากการส่งข้อมูลโดยใช้สายสองเส้น ทำให้มีเพียงมีเพียง node เดียวเท่านั้นที่ทำการส่งข้อมูลได้ถูกต้องในขณะใดขณะหนึ่ง แต่เราต้องการให้ทุก node มีโอกาสส่งข้อมูลได้เช่นเดียวกัน ดังนั้น จึงใช้วิธีการ polling โดยที่ master จะทำการ poll ไปยัง terminal (DCE และ DTE) แต่ละตัวแบบวนรอบ เพื่อรับข้อมูลสถานะ เก็บข้อมูล และ ส่งคำสั่งควบคุม ซึ่ง terminal ก็จะมีหน้าที่ ต้องตอบสนองคำสั่งของ master ดังนั้น จึงเป็น protocal แบบ order/reply message protocal

ลักษณะของข้อมูล

ข้อมูลที่ใช้แบ่งออกเป็น 4 ชนิด คือ แอดเดรส คำสั่ง สถานะ และ ข้อมูลแสดง error

เมื่อ master ส่งข้อมูลไปแล้ว จะต้องได้รับข้อมูลกลับมาเสมอ ยกเว้นข้อมูลที่ส่งไปยัง terminal ทุกตัวพร้อมกัน terminal จะตอบสนองต่อคำสั่ง 'reser network' และคำสั่ง 'status query' และ ถ้าหาก terminal ได้รับคำสั่งนี้แล้วไม่ทราบความหมาย จะต้องตอบกลับไปว่า'command not support'

การตรวจสอบและแก้ไขความผิดพลาด

ในระบบ network จะต้องสามารถตรวจสอบและแก้ไขความผิดพลาดเหล่านี้ได้

-เกิด noise ขึ้น ทำให้ข้อมูลผิดพลาด ซึ่ง terminal ทุกตัวรวมทั้ง master จะต้องตรวจสอบได้ แล้วขอให้ส่งข้อมูลที่ผิดพลาดนั้นใหม่อีกครั้ง

-terminal เป้าหมายไม่สามารถทำงานตามคำสั่งได้ master จะต้องทำการตรวจสอบ หากเกิดกรณีนี้ขึ้นก็ต้องส่งข้อมูลคำสั่งซ้ำไปอีกเป็นจำนวนครั้งตามที่กำหนด ถ้าหากผลยังผิดพลาดอีกก็ต้องตัด terminal นั้นออกไปจาก list of active พร้อมกับแจ้งให้ผู้ใช้ทราบ

- ถ้าหากมีการส่งข้อมูลที่ตัวรับไม่ทราบความหมาย ก็ต้องมีการขอให้ส่งซ้ำ ถ้ายังเหมือนเดิมก็ต้องแจ้งให้ผู้ใช้ทราบ

-เกิดความผิดพลาดขึ้นที่ master ทำให้ไม่สามารถทำงานต่อได้

ต้องกำหนดให้มี master ตัวสำรองที่สามารถมาทำงานแทนเพื่อให้ระบบทำงานต่อไปได้ โดยการตรวจสอบสัญญาณจาก master ซึ่งถ้าหากขาดหายไปจนถึงช่วงเวลาที่กำหนด ตัวสำรองจะต้องทำงานแทนทันที แต่จะต้องตรวจสอบให้ดีว่าตัวสำรองเสียหรือไม่ (ทำให้ไม่ได้รับสัญญาณจาก master)

ลักษณะโครงสร้างของข้อมูล จะแตกต่างกันตามชนิดของอุปกรณ์ แต่มีส่วนที่เหมือนกัน คือ เป้าหมาย และ แหล่งที่มาของข้อมูล ดังนี้

MASTER

มีรูปแบบของข้อมูล 3 แบบ คือ

1) สัญญาณ READ มีความยาว 3 ไบท์ (ถ้า terminal มีหลาย channel ก็จะอ่านข้อมูลทั้งหมด โดยไม่เจาะจง channel)

เป้าหมาย	ที่มา(master)	คำสั่ง READ

2) สัญญาณ WRITE มีความยาว 4 ไบท์ (ถ้า terminal มีหลาย channel ก็ต้องระบุ channel ด้วย)

เป้าหมาย	ที่มา(master)	คำสั่ง WRITE	channel/staus

3) สัญญาณที่ส่งไปยัง terminal ทุกตัวพร้อมกัน มีความยาว 3 ไบท์ (ซึ่งจะไม่มีสัญญาณตอบกลับจาก terminal)

เป้าหมาย	ที่มา(master)	คำสั่ง

DTE

ข้อมูลที่ส่งจากด้านนี้เป็นสัญญาณตอบสนองต่อคำสั่งจาก master มี 2 รูปแบบ ดังนี้

1) สัญญาณตอบสนองการREADมีความยาว 3 ไบท์ ถ้า terminal มี 8 channel ก็จะส่งสถานะไปทั้งหมดซึ่งมีความยาว 1 ไบท์พอดี

เป้าหมาย(master)	ที่มา	status

2) สัญญาณตอบสนองการ WRITE

เป้าหมาย(master)	ที่มา	status

DCE

เป็นการตอบสนองสัญญาณpollจาก master ว่าDCE ต้องการส่งข้อมูลหรือควบคุมอุปกรณ์ตัวใดหรือไม่

1) สัญญาณแสดงความต้องการ READ มีความยาว 4 ไบท์ (ถ้าต้องการอ่านค่าจาก terminal ที่มี หลาย channel ก็จะอ่านข้อมูลทั้งหมดโดยไม่เจาะจง channel)

เป้าหมาย(master)	ที่มา(DCE)	คำสั่ง	เป้าหมาย(DTE)

2) สัญญาณแสดงความต้องการ WRITE มีความยาว 5 ไบท์ (ถ้า

terminal มีหลาย channel ก็ต้องระบุ channel ด้วย)

เป้าหมาย(master)	ที่มา(DCE)	คำสั่ง	เป้าหมาย(DTE)	channel/status

3) สัญญาณแสดงการตอบรับ การ poll จาก master เพื่อแสดงให้ master ทราบถึงการยังมีการติดต่อกันอยู่ระหว่างตัว DCE กับ ตัว master

เป้าหมาย(master)	ที่มา(DCE)	คำสั่ง NOP

การอ้าง ADDRESS ของ TERMINAL

ในระบบนี้ใช้ข้อมูลจำนวน 7 บิท ในการกำหนด address ของ terminal ซึ่งจะอ้างถึงอุปกรณ์ได้สูงสุดจำนวน 2^7 = 128 ตัว ในที่นี้จะให้

address 0 แทน all slave

address 1-30 แทน slave

address 31 แทน master1(PC)

address 32 แทน master2

5.3 สรุปผลการทดลอง

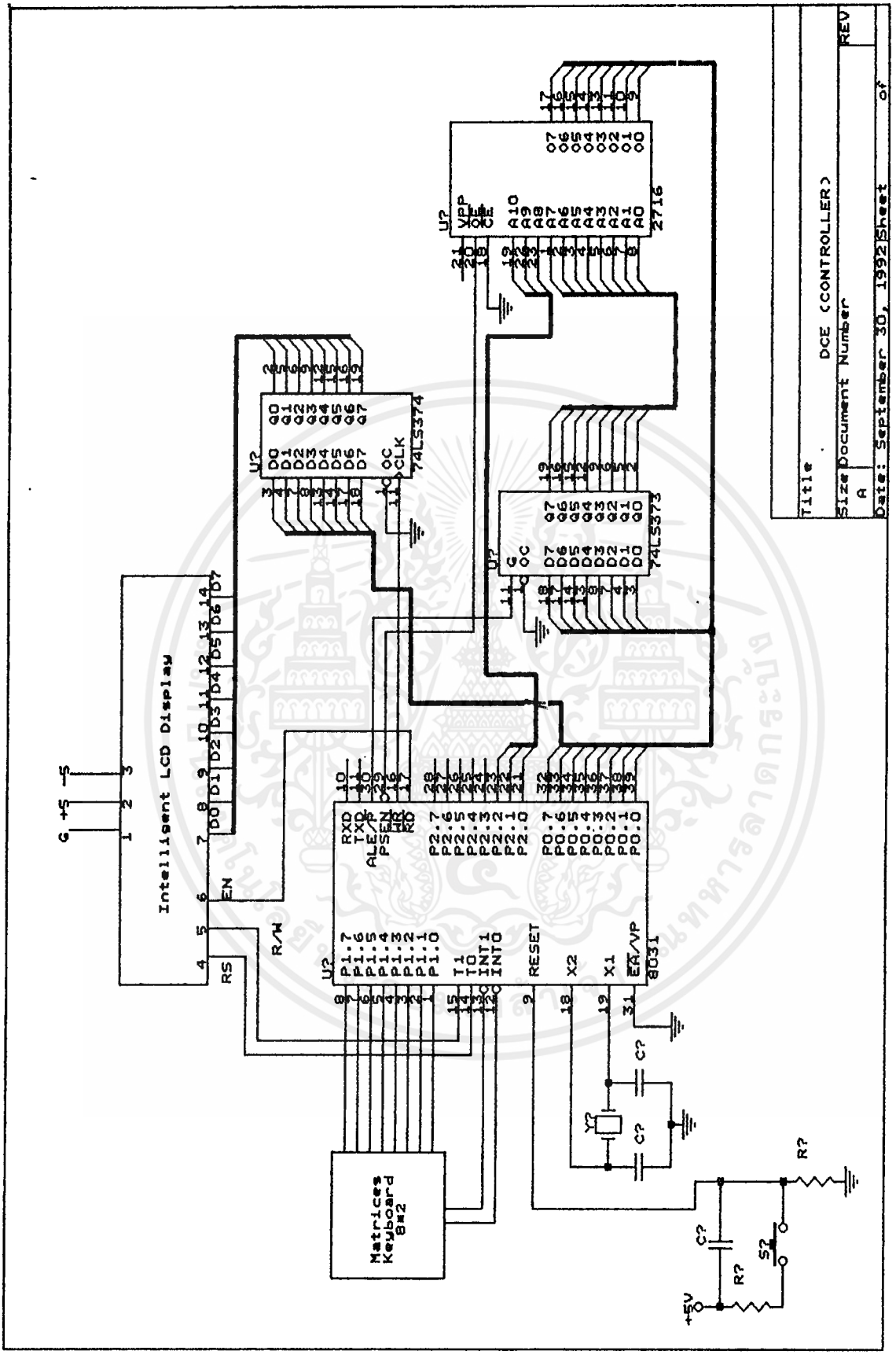
จากการทดลองได้สร้างตัวควบคุม (Controller) ขึ้น 1 ตัว และตัวอุปกรณ์ที่ใช้ควบคุมการเปิด-ปิด (DTE) 2 ตัว และการ์ดที่ใช้เป็นตัวควบคุมบนเครื่อง PC 1 ตัว

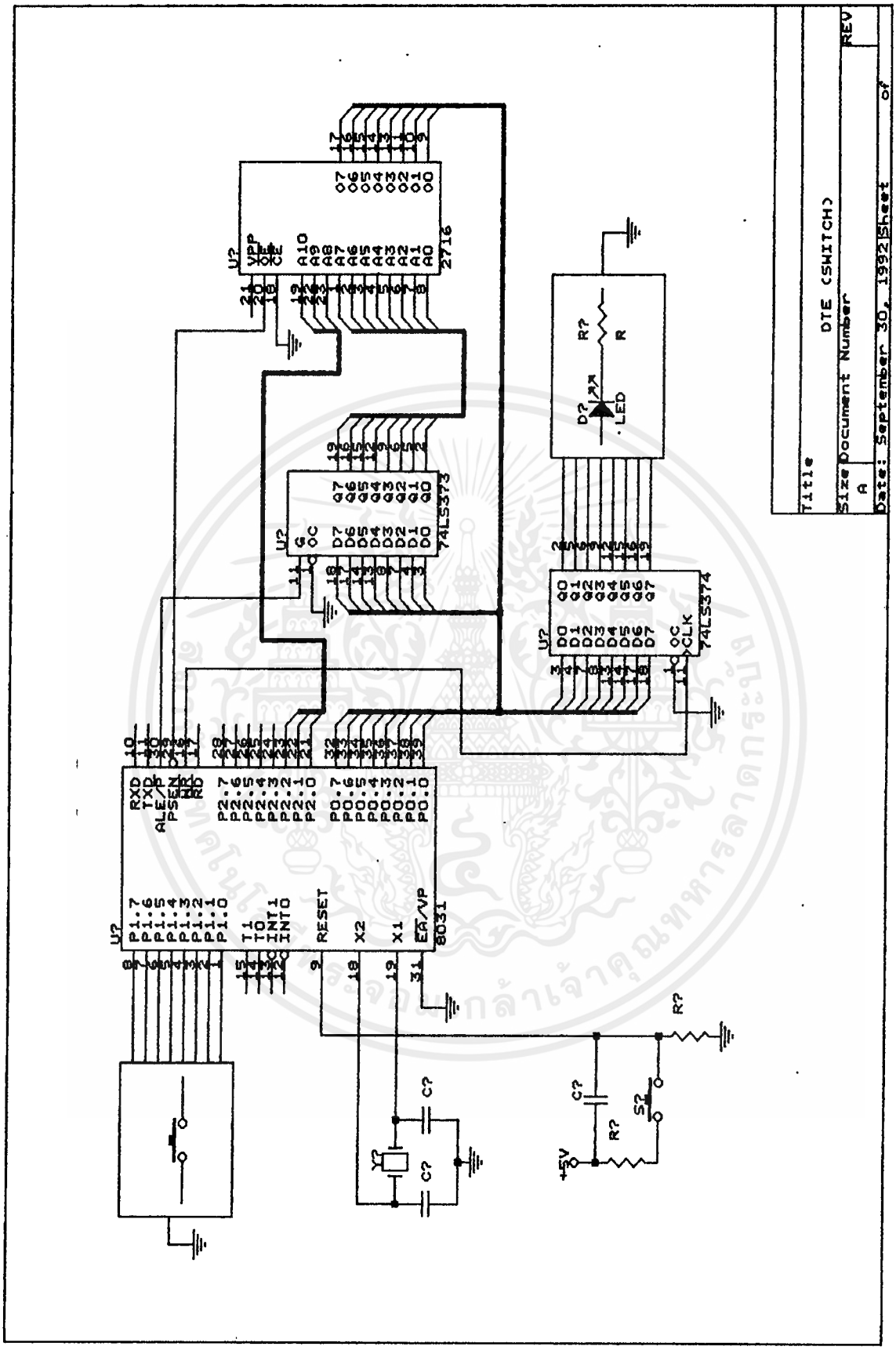
จากการนำเอาตัวควบคุม และตัวอุปกรณ์ที่ใช้ควบคุมการเปิด-ปิด (DTE) มาต่อเข้าด้วยกัน โดยการโปรแกรมให้ อุปกรณ์ที่ใช้ควบคุมการเปิด-ปิด(DTE) ตัวแรกแทนอุปกรณ์ 1 หมายเลข และอีกตัวหนึ่งแทนอุปกรณ์อีก 29 หมายเลข รวมเป็น 30 หมายเลข พบว่าสามารถตรวจสอบสถานะ และการทำการเปิด-ปิด อุปกรณ์ได้ด้วยการตอบสนองที่เร็วพอประมาณ และได้ทดลองต่ออุปกรณ์ปลายทาง 2 ตัวเข้าด้วยกัน คือ ตัวควบคุม และ อุปกรณ์ที่ใช้ควบคุมการเปิด-ปิด(DTE) โดยใช้สายโทรศัพท์ความยาว 100 เมตรพบว่าสามารถส่งข้อมูลถึงกันได้ถูกต้อง

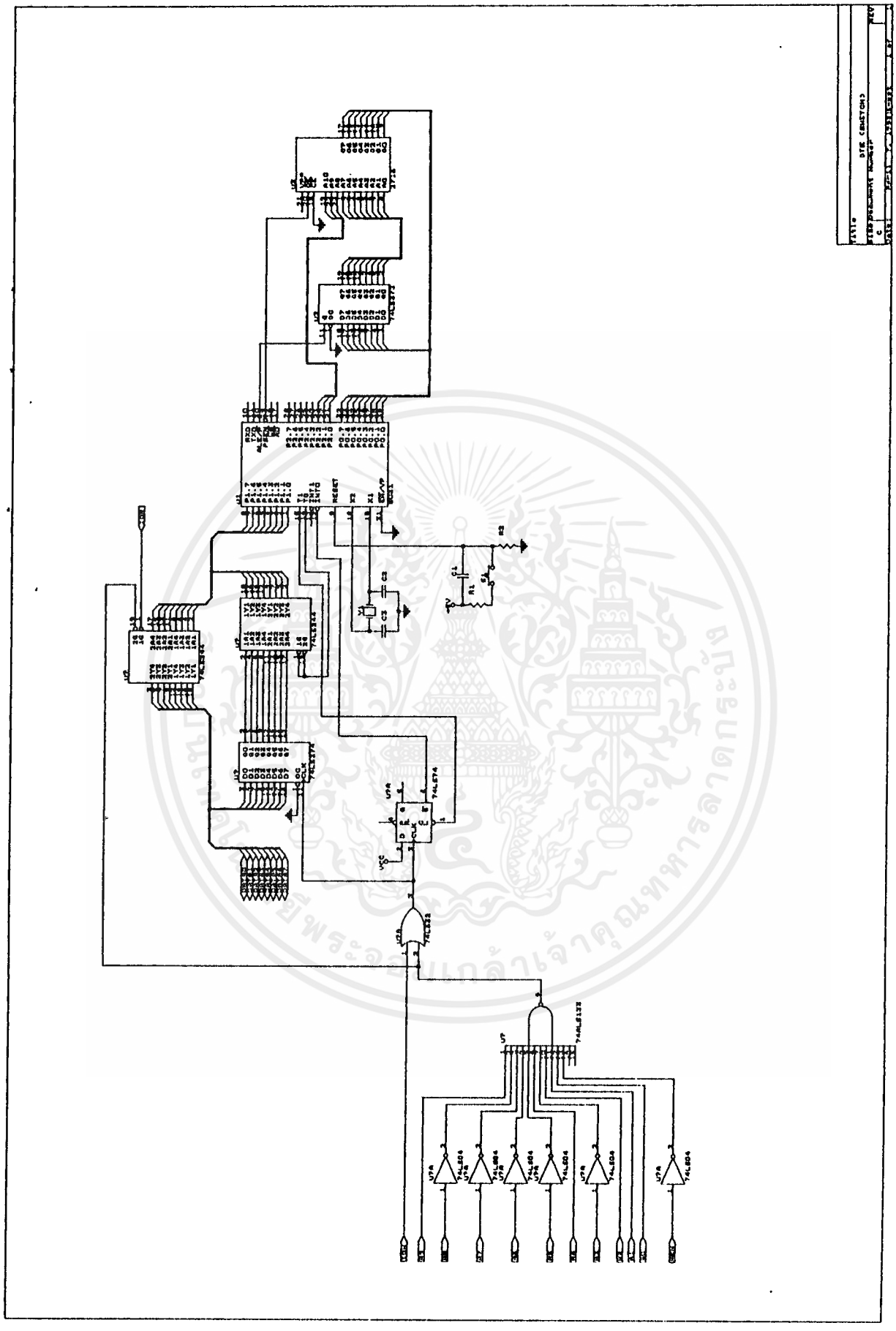
ในการต่อการ์ดบน PC อุปกรณ์ที่ใช้ควบคุมการเปิด-ปิด (DTE) ก็สามารถใช้ควบคุมอุปกรณ์และตรวจสอบสถานะได้เช่นกัน

ภาคผนวก ก

วงจรอาร์ดแมร์







ภาคผนวก ข

Source Code

โปรแกรมสำหรับตัวควบคุม

```
;*****************************************
;***** PROGRAM FOR CONTROLLER  (C1.ASM) ****
;*****************************************

        tar       equ   37h
        flag      equ   00h
        flags     equ   01h
        add_con   equ   02h
        state     equ   30h
        addr      equ   31h
        b_addr    equ   32h
        key       equ   33h
        cursor    equ   34h
        status    equ   35h

        zero      equ   00h
        one       equ   01h
        left      equ   0ah
        right     equ   0bh
        down      equ   0ch
        up        equ   0dh
        address   equ   0eh
        clear     equ   0fh

        org 0000h
      clr flags
      mov scon,#0c0h          ;set scon mode 3
      mov tmod,#20h
      mov th1,#0fdh           ;set baud rate = 9600
```

```
        setb tr1
        mov state,#01h
        mov cursor,#01h
        mov addr,#01h
        mov b_addr,#01h
        acall init_lcd
        acall show_dat
        mov a,#40h
        acall goto

main:   clr add_con
        acall poll
        mov a,state
        cjne a,#01h,change
        jb add_con,change
        acall show_dat
change: acall keydown
        acall delaya
        sjmp main

;-----------------------------------------------------------------
;       routine polling
;-----------------------------------------------------------------
poll:   mov r2,#20h             ;the amount of polled DTE = 32
poll2:  setb tb8                ;set 9th bit for target address

        setb p3.7
        mov a,r2                ;target address
s1:     mov tar,a               ;
```

```
        acall sdata             ;send target address  <1>
        acall delay

        clr tb8                 ;clear 9th bit
s2:     mov a,#0ffh             ;master address
        acall sdata             ;send source address  <2>
        acall delay

        jb flags,write1         ;Have data to send ?
s3:     mov a,#0fh              ;read command = #0fh or status to
        mov r0,a
        acall sdata             ;send read command    <3>

r00:    setb ren                ;set recieve enable
        clr p3.7
r01:    acall rcdata            ;master address     <1>

r02:    lcall rcdata            ;slave address    <2>

r03:    acall rcdata            ;command      <3>

r04:    mov a,#40h
        add a,tar
        mov r0,a
        lcall rcdata            ;read status      <4>
        mov @r0,a               ;store status
        acall delay

        djnz r2,poll2
```

```
          ret

write1:   mov a,tar
          cjne a,addr,s3          ;it's queue ?
          mov r0,#0f0h
          mov a,r0
          acall sdata
          acall delay

          mov a,status            ;send status
          acall sdata
          clr flags
          sjmp r00                ;go back to recieve

sdata:    mov sbuf,a
          jnb ti,$
          clr ti
          ret

rcdata:   mov r1,#04h
rcdata1:  mov r3,#0c4h
rcdata2:  jb ri,normal
          djnz r3,rcdata2
          djnz r1,rcdata1
          mov a,r2
          cjne a,addr,oh
          acall discon
          setb add_con
          sjmp oh
```

```
normal:  clr ri
         mov a,sbuf
oh:      ret

;-------------------------------------------------------------
;                  subroutine display 'disconnect'
;-------------------------------------------------------------
discon:  push acc
         mov a,#40h
         acall goto
         mov a,#58h              ;x
         acall putch
         acall putch
         acall putch
         acall putch
         acall putch
         acall putch
         acall putch
         acall putch
         pop acc
         ret

;-------------------------------------------------------------
;                     subroutine KEYDOWN
;                  for check any key(s) down
;-------------------------------------------------------------
keydown: mov a,#00h
         mov r0,#00h          ;clr r0
         mov a,#0c3h
```

```
anl a,p3
orl a,#3ch
mov p3,a

setb p1.0
clr p1.3            ;out 0 to p1 for scan row0
mov r0,p3           ;input all key to r0
mov a,r0            ;store all key to A
orl a,#0c3h         ;to 11XXXX11
cpl a               ;A = FFh if all keys up :now 00h
jnz have            ;if any key(s) down to have for ret

setb p1.3
clr p1.2            ;out 0 to p1 for scan row1
mov r0,p3           ;input all key to r0
mov a,r0            ;store all key to A
orl a,#0c3h         ;to 11XXXX11
cpl a               ;A = FFh if all keys up :now 00h
jnz have            ;if any key(s) down to have for ret

setb p1.2
clr p1.1            ;out 0 to p1 for scan row2
mov r0,p3           ;input all key to r0
mov a,r0            ;store all key to A
orl a,#0c3h         ;to 11XXXX11
cpl a               ;A = FFh if all keys up :now 00h
jnz have            ;if any key(s) down to have for ret

setb p1.1
```

```
          clr p1.0              ;out 0 to p1 for scan row3
          mov r0,p3             ;input all key to r0
          mov a,r0              ;store all key to A
          orl a,#0c3h           ;to 11XXXX11
          cpl a                 ;A = FFh if all keys up :now 00h
have:     jnz have3
          ret
have3:    acall convert
          acall vendit
          ret

vendit:   mov r3,key
          cjne r3,#zero,ones
          acall d_0_1             ;key 0
          ret

ones:     cjne r3,#one,lefts
          acall d_0_1              ;key 1
          ret

lefts:    cjne r3,#left,rights
          mov a,state             ;10th key is left key
          cjne a,#01h,left1       ;if state = 1
          mov a,cursor
          cjne a,#01h,left2       ;if cursor = 1
          mov a,#47h
          acall goto
          mov cursor,#08h
left1:    ret
```

```
left2:   mov a,cursor
         dec a                    ; update cursor
         mov cursor,a             ;
         add a,#3fh               ;
         acall goto               ; move cursor
         ret

rights:   cjne r3,#right,downs
         mov a,state              ;1ist key is right key
         cjne a,#01h,right1       ;if state = 1
         mov a,cursor
         cjne a,#08h,right2       ;if cursor = 8
         mov a,#40h
         acall goto
         mov cursor,#01
right1:  ret
right2:  inc cursor               ;update cursor
         mov a,cursor
         add a,#3fh
         acall goto
         ret

downs:   cjne r3,#down,ups         ;12nd key is dec key
         mov a,state
         cjne a,#01h,ok           ;if state = 1
         mov r7,addr
         cjne r7,#01,tt1
         mov addr,#32
         sjmp tt2
```

```
tt1:    dec addr                ;r7 store current address that
tt2:    acall show_dat          ;display to LCD
ok:     ret

ups:    cjne r3,#up,addrs       ;13nd key is inc key
        mov a,state
        cjne a,#01h,ok1
        mov r7,addr
        cjne r7,#32,tt3
        mov addr,#01
        sjmp tt4
tt3:    inc addr                ;addr store current address that
tt4:    acall show_dat          ;display to LCD
ok1:    ret

addrs:  cjne r8,#address,clears     ;if key = #address (14th)
        mov state,#02h
        mov a,#00h
        acall goto
        ret

clears: cjne r3,#clear,digit    ;if key = #clear (15th)
        mov a,state
        cjne a,#01h,clean
        mov addr,b_addr         ;load back-up addr.
        mov a,addr
        acall show_dat
        mov state,#01h
clean:  ret
```

```
digit:  mov a,state
        cjne a,#03h,dig1        ;if state = 3
dig0:   mov a,key
        add a,#30h
        acall putch
        mov a,addr
        add a,key
        clr c
        mov addr,a              ;new addr. value
        add a,#0d0h             ;if old a >= 20h, then carry = 1
        jnc d01
        mov state,#02h
        mov a,#00h
        acall goto
        ret
d01:    mov state,#01h
        acall show_dat
        ret
dig1:   cjne a,#02h,dig2
dig11:  mov a,key               ;first digit of new address
        add a,#30h
        acall putch
        mov b_addr,addr
        mov a,key               ;
        mov b,#0ah              ;change decimal to hex
        mul ab                  ;a*b -> a (low) ,b (high)
        mov addr,a              ;
        mov state,#03h
dig2:   ret
```

```
d_0_1:   mov a,state
         cjne a,#01h,d_stat1     ;if state = 1
         mov a,key
         add a,#30h              ;a = '0' or '1'
         acall putch             ;put new status bit
         mov a,cursor
         cjne a,#8,d01_2         ;if cursor = 8
         mov cursor,#0           ;cursor = 0
d01_2:   inc cursor              ;update cursor
         mov a,cursor
         add a,#3fh
         acall goto              ;mov cursor to new location
         mov a,key
         cjne a,#00h,d1
         mov r1,#11111101b
         acall find_a            ;return status to R2
         anl a,r2                ;R2 contains status of default addr.
         sjmp write

d1:      mov r1,#00000010b
         acall find_a            ;return status to R2
         orl a,r2
write:   mov @r0,a               ;update status in memory
         mov status,a            ;store status in temp memory(name
         setb flags
         ret

d_stat1: cjne a,#02h,dig0        ;if state = 2
         sjmp dig11
```

```
find_a:  mov a,addr
         add a,#40h              ;add with base in memory
         mov r0,a
         mov a,@r0
         mov r2,a                ;status value of default addr.
         mov a,r1                ;R1 = 00000001 or 11111110
         mov r4,cursor           ;cursor is a count value to rotate
rotate:  rr a                    ;rotate R1  for update display
         djnz r4,rotate
         ret

;-------------------------------------------------------------
;                    subroutine CONVERT
;     for convert key to numberkey 0-F and check for multikey down
;-------------------------------------------------------------
convert: clr a
         clr flag                ;assume that key hit is valid
         mov b,#0f7h             ;out 0 to p1 on row0 : origin row
         mov r0,#00h             ;clear r0 for in-output
         mov r2,#00h             ;clear r2 : maximum = 4
         mov r3,#00h             ;clear r3 : for store number of mult

row:     mov a,b                 ;check each row
         anl a,#0fh
         mov r1,a
         mov r0,p1               ;in port1 for change low byte
         push 0e0h               ; -
         mov a,r0                ;  |
```

```
        anl a,#0f0h             ;    > and r0,0f0h
        orl a,r1                ;    > or r0,r1
        mov r0,a                ;   |
        pop 0e0h                ; -

        mov p1,r0               ;out 0 to port1 for check each row
        mov a,#0c3h
        anl a,p3
        orl a,#03ch
        mov p3,a                ;out XX1111XX to port3 as input
        mov r1,p3               ;input port 3 to r1
        mov a,r1
        orl a,#0c3h             ;to 11XXXX11
        mov r1,a                ;keydown in r1

nohave: cjne r1,#0ffh,have1 ;if have key down to zero for check
        mov a,b
        rr a                    ;shift 1 bit for next row
        mov b,a
        inc r2                  ;count number of row
        cjne r2,#04h,row        ;check untill number of row = 4
        sjmp check              ;go to check multikeys down ?

have1:  mov r4,b                ;r4 store row that have keydown
zeros:  cjne r1,#0fbh,one0
        mov r5,#00h             ;A = 00
        inc r3                  ;count multikey down
        sjmp check
one0:    cjne r1,#0f7h,two0
```

```
        mov r5,#01h          ;A = 01
        inc r3               ;count multikey down
        sjmp check
two0:    cjne r1,#0efh,three0
        mov r5,#02h          ;A = 02
        inc r3               ;count multikey down
        sjmp check
three0:  mov r5,#03h          ;A = 03
        inc r3               ;count multikey down
loop:   inc r2               ;count number of row
        mov a,b
        rr a                 ;shift 1 bit for next row
        mov b,a
        cjne r2,#04h,row     ;check untill number of row = 4
check:  cjne r3,#01h,bad1    ;if have multikey -> bad
        sjmp good
bad1:   ljmp bad
good:   cjne r4,#0f7h,row1   ;if not row0 -> go row1
        cjne r5,#00h,xy10    ;if not column0
        mov key,#07h         ;number of key = 7
        sjmp back2
xy10:   cjne r5,#01h,xy20    ;if not column1
        mov key,#08h         ;number of key = 8
        sjmp back2
xy20:   cjne r5,#02h,xy30    ;if not column2
        mov key,#09h         ;number of key = 9
        sjmp back2
xy30:   mov key,#0fh          ;number of key = f
        sjmp back2
```

```
row1:   cjne r4,#0fbh,row2  ;if not row1 -> go row2
        cjne r5,#00h,xy11   ;if not column0
        mov key,#04h        ;number of key = 4
        sjmp back2
xy11:   cjne r5,#01h,xy21   ;if not column1
        mov key,#05h        ;number of key = 5
        sjmp back2
xy21:   cjne r5,#02h,xy31   ;if not column2
        mov key,#06h        ;number of key = 6
        sjmp back2
xy31:   mov key,#0eh        ;number of key = e
        sjmp back2

row2:   cjne r4,#0fdh,row3  ;if not row2 -> go row3
        cjne r5,#00h,xy12   ;if not column0
        mov key,#01h        ;number of key = 1
        sjmp back2
xy12:   cjne r5,#01h,xy22   ;if not column1
        mov key,#02h        ;number of key = 2
        sjmp back2
xy22:   cjne r5,#02h,xy32   ;if not column2
        mov key,#03h        ;number of key = 3
        sjmp back2
xy32:   mov key,#0dh        ;number of key = d
        sjmp back2

row3:   cjne r5,#00h,xy13   ;if not column0
        mov key,#00h          ;number of key = 0
        sjmp back2
```

```
xy13:    cjne r5,#01h,xy23    ;if not column1
         mov key,#0ah         ;number of key = a
         sjmp back2
xy23:    cjne r5,#02h,xy33    ;if not column2
         mov key,#0bh         ;number of key = b
         sjmp back2
xy33:    mov key,#0ch         ;number of key = c
         sjmp back2
bad:     setb flag
back2:   ret

;-------------------------------------------------------------
;              subroutine  delay
;-------------------------------------------------------------
delaya:   mov r4,#0ffh
delaya1:  mov r5,#60h
delaya2:  nop
          nop
          nop
          nop
          djnz r5,delaya2
          djnz r4,delaya1
          ret

;-------------------------------------------------------------
;                        subroutine  inc_dec
;-------------------------------------------------------------
show_dat:mov a,#00h
```

```
        acall goto
        acall show_add
        mov a,addr
        add a,#40h          ;real address on RAM
        mov r0,a
        mov a,@r0           ;store data in address 60+r7 to a
        push 0e0h
        mov a,#40h          ;shift cursor to position address 40
        acall goto
        pop 0e0h
        mov r6,#00h         ;count bits for shifting
data:   rlc a
        jc hight
low:    push 0e0h
        mov a,#30h          ;ascii 0 equal 30h
        acall putch         ;display 0 to LCD
        pop 0e0h
        inc r6
        cjne r6,#08h,data   ;shift bit until r6 = 8
        sjmp ready
hight:  push 0e0h
        mov a,#31h          ;ascii 1 equal 31h
        acall putch         ;display 1 to LCD
        pop 0e0h
        inc r6
        cjne r6,#08h,data   ;shift bit until r6 = 8
        ;mov a,#47h
        ;acall goto
ready:  mov a,cursor
```

```
          add a,#3fh
          acall goto          ;move cursor to default position
          ret

;-------------------------------------------------------------
;                    subroutine  show_add
;-------------------------------------------------------------
show_add:mov a,addr
          acall h_to_d
          swap a              ;high byte
          anl a,#0fh
          acall ascii
          acall putch         ;display hight byte
          mov a,addr          ;low byte
          acall h_to_d
          anl a,#0fh
          acall ascii
          acall putch         ;display low byte
          ret

;--------- routine for change hex address to dec for show_add --
h_to_d:   mov b,#0ah
          div ab                ;a/b -> a(result) ,b(remainder)
          swap a
          add a,b
          ret

;-------------------------------------------------------------
epulse:          nop
```

```
                nop
                setb p1.6               ;EN_pulse
                nop
                nop
                nop
                nop
                nop
                clr p1.6
                nop
                nop
                ret

;-----------------------------------------------------------------
clrscr:         push acc
                clr p1.4
                clr p1.5
                mov a,#01h              ;function clearscreen
                movx @dptr,a
                acall epulse
delay0:         mov r2,#0ffh
delay2:         mov r3,#01h
delay3:         nop
                djnz r3,delay3
                djnz r2,delay2
                pop acc
                ret

home:           push acc
                clr p1.4
```

```
                clr p1.5
                mov a,#2                ;function return home
                movx @dptr,a
                acall epulse
                acall delay
                pop acc
                ret

show_cur:       push acc
                clr p1.4
                clr p1.5
                mov a,#0eh
                movx @dptr,a
                acall epulse
                pop acc
                ret

hide_cur:       push acc
                mov a,#11h
                acall goto
                pop acc
                ret

goto:           clr p1.4
                clr p1.5
                push acc
                orl a,#80h              ;function set DD RAM addr
                movx @dptr,a
                acall delay             ;
```

```
                acall epulse
                acall delay             ;
                acall delay
                pop acc
                ret

putch:          setb p1.4
                clr p1.5
                movx @dptr,a
                acall epulse
                acall delay
                ret

delay:          mov r0,#030h
delay1:         djnz r0,delay1
                ret

ascii:  inc a
        movc a,@a+pc
        ret
        db 30h              ;0
        db 31h              ;1
        db 32h              ;2
        db 33h              ;3
        db 34h              ;4
        db 35h              ;5
        db 36h              ;6
        db 37h              ;7
        db 38h              ;8
```

```
        db 39h          ;9
        db 41h          ;a
        db 42h          ;b
        db 43h          ;c
        db 44h          ;d
        db 45h          ;e
        db 46h          ;f

;-----------------------------------------------------------------
init_lcd:       mov r0,#70              ;wait more than 15 ms
point0:         mov r1,#0ffh
point1:         djnz r1,point1
                djnz r0,point0

                clr p1.4                ;RS=0
                clr p1.5                ;R/W=0
                mov a,#3ch
                movx @dptr,a
                acall epulse

                mov r0,#20              ;wait more than 4 ms
point3:         mov r1,#0ffh
point4:         djnz r1,point4
                djnz r0,point3
                mov a,#3ch
                movx @dptr,a
                acall epulse

                mov r0,#120             ;wait more than 0.1 ms
```

```
point2:         djnz r0,point2
                mov a,#3ch
                movx @dptr,a
                acall epulse
                acall delayb

                mov a,#3ch              ;function set
                movx @dptr,a
                acall epulse
                acall delayb

                mov a,#8                ;display OFF
                movx @dptr,a
                acall epulse
                acall delayb

                mov a,#0eh              ;display ON
                movx @dptr,a
                acall epulse
                acall delayb

                mov a,#6                ;entry mode set
                movx @dptr,a
                acall epulse
                acall delayb

                mov a,#01
                movx @dptr,a
                acall epulse
```

```
                acall delayb
                ret

;-----------------------------------------------------------------
disp_ON:        mov a,#0eh
                movx @dptr,a
                acall epulse
                ret

;-----------------------------------------------------------------
;                       delayb
;-----------------------------------------------------------------
delayb:         mov r0,#30              ;wait more than 15 ms
point00:        mov r1,#0ffh
point01:        djnz r1,point01
                djnz r0,point00
                ret

                end
```

โปรแกรมสำหรับตัว DTE

```
;*************************************************
;********    PROGRAM FOR DTE  (DTE.ASM)    *********
;*************************************************
              status   equ    40h
              address  equ    41h
              command  equ    42h
                 org 0000
                 clr p3.5                ;enable ship interface
                 clr tb8
                 mov status,#00h         ;store output status
                 mov a,status
                 movx @dptr,a            ;output to LED
                 mov scon,#0f0h          ;set scon mode 3, ren, sm2
                 mov tmod,#20h
                 mov th1,#0fdh           ;set baud rate = 9600
                 setb tr1
                 mov ie,#90h             ;enable serial port inter-
                 clr p3.4                ;enable to read dipswitch
                 mov a,p1                ;read address dte
                 cpl a
                 mov address,a
                 setb p3.4
main:            sjmp getkey

;-----------------------------------------------------
;         routine polling-detection
;-----------------------------------------------------
         org 23h
r01:     acall rcdata            ;slave address   <1>
```

```
            mov r3,a                ;store target address in R3
            clr sm2

r02:        acall rcdata            ;recieve master address  <2>
            mov r4,a                ;store master address in R4

r03:        acall rcdata            ;read command     <3>
            mov command,a           ;store command in R5
 cjne a,#0fh,write1

s00:        setb p3.5
s1:         mov a,r4                ;master address
            acall sdata             ;send target address <1>
            acall delay

s2:         mov a,#0ffh             ;slave address
            acall sdata             ;send source address <2>
            acall delay

s3:         mov a,command           ;send command
            acall sdata
            acall delay
            mov a,status            ;read status
s32:        acall sdata            ;send status         <3>
            clr p3.5
ret_in:     setb sm2
            reti

write1:     mov a,command
```

```
        mov r3,a                ;store target address in R3
        clr sm2

r02:    acall rcdata            ;recieve master address  <2>
        mov r4,a                ;store master address in R4

r03:    acall rcdata            ;read command    <3>
        mov command,a           ;store command in R5
 cjne a,#0fh,write1

s00:    setb p3.5
s1:     mov a,r4                ;master address
        acall sdata             ;send target address <1>
        acall delay

s2:     mov a,#0ffh             ;slave address
        acall sdata             ;send source address <2>
        acall delay

s3:     mov a,command           ;send command
        acall sdata
        acall delay
        mov a,status            ;read status
s32:    acall sdata            ;send status        <3>
        clr p3.5
ret_in: setb sm2
        reti

write1: mov a,command
```

```
        cjne a,#0f0h,s00
        acall rcdata
        mov status,a
        movx @dptr,a
        sjmp s00

rcdata: jnb ri,$
        clr ri
        mov a,sbuf
        ret

sdata:  mov sbuf,a
        jnb ti,$
        clr ti
        ret

delay:  mov r6,#050h
delay1: djnz r6,delay1
        ret

;-------------------------------------------------- -- ------

    bounce  equ     14h
    next    equ     32h
    newkey  equ     70h
    flag    equ     00h
    getkey: mov p1,#0ffh        ;set port 1 as input
    scan:   acall keydown       ;keydown look for any key down
            jz  scan            ;if a=0 then no key down
            acall convert       ;convert return flag set if val
```

```
            jbc flag,scan          ;if flag = 1 (invalid) goto scan
            mov newkey,a           ;store key
            mov a,#bounce          ;
            acall softime          ;wait for 20 ms
            acall keydown          ;see if a key is still down
            jz  scan               ;if not down then must scan key
            acall convert          ;see if key is still valid
            jbc flag,scan          ;if flag = 1 (invalid) goto scan
            cjne a,newkey,scan     ;check for equal
            acall vendit           ;call for display
wait:       acall keydown          ;now wait for all keydown to go
            jnz wait               ;wait untill a = 0 , key all up
            mov a,#next            ;wait 50d ms and see if all still
            acall softime
            acall keydown          ;continute untill keys are up
            jnz wait               ;loop untill keys up for 50d ms
            sjmp scan              ;get next key

;---------------------------------------
; procedure keydown
;---------------------------------------
keydown:    mov a,p1               ;get state of p1 keys to r0
            cpl a                  ;a = ffh if all keys up
            ret                    ;if A not 00 then at least one

;---------------------------------------
; procedure convert
;---------------------------------------
convert:    clr flag
```

```
            clr a
            mov r3,#00h
            mov r1,p1
zero:       cjne r1,#0feh,one
            mov a,#0feh
            inc r3
one:        cjne r1,#0fdh,two
            mov a,#0fdh
            inc r3
two:        cjne r1,#0fbh,three
            mov a,#0fbh
            inc r3
three:      cjne r1,#0f7h,four
            mov a,#0f7h
            inc r3
four:       cjne r1,#0efh,five
            mov a,#0efh
            inc r3
five:       cjne r1,#0dfh,six
            mov a,#0dfh
            inc r3
six:        cjne r1,#0bfh,seven
            mov a,#0bfh
            inc r3
seven:      cjne r1,#07fh,check
            mov a,#07fh
            inc r3
check:      cjne r3,#01h,bad
good:       ret
```

```
bad:      setb flag
          ret

;-------------------------------------
; procedure softime
;-------------------------------------
softime:  push 07h
          push acc
          orl a,b
          cjne a,#00h,ok
          pop acc
          sjmp done
ok:       pop acc
timer:    mov r7,#0ech
onemil:   nop
          nop
          nop
          nop
          djnz r7,onemil
          nop
          djnz acc,timer
          cjne a,b,bdown
          sjmp done
bdown:    dec b
          sjmp timer
done:     pop 07h
          ret
```

```
;-------------------------------------
; procedure vendit
;-------------------------------------
  vendit:  cpl a                  ;only one bit of A is 1, others
           mov r6,a               ;copy A to R6
           anl a,status               ;A = 00h, if that bit (old state)
           cjne a,00h,off
           mov a,r6
           orl a,status               ;set LED bit OFF to ON
           sjmp new_st
  off:     mov a,r6
           cpl a                  ;only one bit of A is 0
           anl a,status               ;set LED bit ON to OFF
  new_st:  mov status,a          ;mov r7,a     ;save LED status in
           movx @dptr,a          ;out new status to LED
           ret
           end
```

โปรแกรมสำหรับการ์ดเชื่อมต่อ PC

```
;*************************************************
;******  PROGRAM FOR CARD ON PC  (CARD.ASM) ******
;*************************************************
            flag        equ     00h
            flags       equ     01h
            wr_stat     equ     30h
            addr        equ     31h
            tar         equ     32h
            status      equ     33h
            command     equ     34h
            addr2       equ     35h

                org 0000h
                ajmp start

                org 0003h
                clr p3.7                ;recieve command and addr
                mov a,p1                ;from pc
                setb p3.7

                jb flag,write_in1       ;2nd byte of write comm.
                mov 21h,a
                jb 0fh,discon           ;command check disconnect
                jb 0eh,write_in         ;write command
                anl a,#1fh
w_send:         add a,#60h              ;read status from memory
                mov r0,a
                mov a,@r0
send:           mov p1,a                ;send status of each dte
```

```
                clr p3.5
                setb p3.5
                reti

discon:         anl a,#03h
                add a,#26h
                mov r1,a
                mov a,@r1
                sjmp send
write_in:       anl a,#1fh
                mov addr,a
                setb flag
                sjmp w_send
write_in1:      mov wr_stat,a          ;keep status that be write
                clr flag
                setb flags
                inc a
                sjmp send

start:          clr flag
                clr flags
                mov scon,#0c0h         ;set scon mode 3
                mov tmod,#20h
                mov th1,#0fdh          ;set baud rate = 9600
                setb tr1

                clr ri
che_mast:       mov r0,#04h
che_mast1:      mov r1,#0c4h
```

```
che_mast2:      jb ri,slave
                djnz r1,che_mast2
                djnz r0,che_mast1

main:           setb ex0
                clr it0
                setb ea
set_in:         acall poll
                acall wait
                acall wait
                sjmp set_in

slave:          clr ri
                clr rb8
                jnb rb8,$
                mov a,sbuf
                cjne a,#31,slave
                clr ri
                jnb ri,$
                clr ri
                jnb ri,$
                clr ri
                mov a,#00h
                acall sdata
                acall delay
                mov a,#31h
                acall sbuf
                acall delay
                mov a,#81h
```

```
                acall sdata
                acall delay
                sjmp main

;-----------------------------------------------------------------
;       routine polling
;-----------------------------------------------------------------
poll:   mov r2,#1eh             ;the amount of polled DTE = 32
poll2:  setb tb8                ;set 9th bit for target address
        setb p3.6
        mov a,r2                ;target address

s1:     mov tar,a               ;
        acall sdata             ;send target address  <1>

        acall delay

        clr tb8                 ;clear 9th bit
s2:     mov a,#0ffh             ;master address
        acall sdata             ;send source address  <2>

        acall delay

        jb flags,write1         ;Have data to send ?
s3:     mov a,#0fh              ;read command = #0fh or status to
        mov r0,a
        acall sdata             ;send read command    <3>

r00:    setb ren                 ;set recieve enable
```

```
          clr p3.6
r01:      acall rcdata              ;master address     <1>

r02:      lcall rcdata              ;slave address    <2>

r03:      acall rcdata              ;command          <3>

r04:      mov a,#60h
          add a,tar
          mov r0,a
          lcall rcdata              ;read status      <4>
          mov @r0,a                 ;store status
          acall delay
          djnz r2,poll2

disc:     mov a,25h
          rrc a
          mov 29h,a
          mov a,24h
          rrc a
          mov 28h,a
          mov a,23h
          rrc a
          mov 27h,a
          mov a,22h
          rrc a
          mov 26h,a
          acall to_write
          ret
```

```
write1:  mov a,tar
         cjne a,addr,s3          ;it's queue ?
         mov r0,#0f0h
         mov a,r0
         acall sdata
         acall delay

         mov a,wr_stat           ;send status
         acall sdata
         clr flags
         sjmp r00                ;go back to recieve

sdata:   mov sbuf,a
         jnb ti,$
         clr ti
         ret

rcdata:  mov a,r2
         mov b,#08h
         div ab
         add a,#22h
         mov r1,a
         inc b

         mov r4,#04h
rcdata1: mov r3,#0c4h
rcdata2: jb ri,normal
         djnz r3,rcdata2
         djnz r4,rcdata1
```

```
          mov a,#7fh
shift1:   rl a
          djnz b,shift1
          anl a,@r1
          mov @r1,a
          sjmp oh

normal:   mov a,#80h
shift2:   rl a
          djnz b,shift2
          orl a,@r1
          mov @r1,a
          clr ri
          mov a,sbuf
oh:       ret

delay:          mov r0,#02ah
delay1:         djnz r0,delay1
                ret

wait:           mov r0,#0ffh
wait1:          mov r1,#0ffh
wait2:          djnz r1,wait2
                djnz r0,wait1
                ret

to_write: setb tb8
          setb p3.6
          mov a,#32h              ;dce addr.
```

```
        acall sdata
        acall delay
        clr tb8
        mov a,#31h
        acall sdata
        acall delay
        mov a,#35h
        acall sdata
        acall delay
        clr p3.6

        setb ren
        mov r4,#04h
loop1:  mov r3,#0c4h
loop2:  jb ri,out
        djnz r3,loop2
        djnz r4,loop1
        ret
out:    jnb ri,$
        clr ri
        jnb ri,$
        mov a,sbuf
        mov command,a
        clr ri
        jnb ri,$
        mov a,sbuf
        mov addr2,a             ;address for read/write
        clr ri
        mov a,command
```

```
        cjne a,#0f0h,read       ;write command
        jnb ri,$
        mov a,sbuf              ;status for writing
        clr ri
        mov status,a

pp2:    setb tb8                ;set 9th bit for target address
        setb p3.6
        mov a,addr2             ;target address
ss1:    acall sdata             ;send target address  <1>
        acall delay
        clr tb8                 ;clear 9th bit
ss2:    mov a,#31h              ;master address
        acall sdata             ;send source address  <2>
        acall delay
ss3:    mov a,#0f0h             ;write command
        acall sdata             ;send read command   <3>
        acall delay
ss4:    mov a,status            ;status for writing
        acall sdata
        acall delay

rr0:    setb ren                ;set recieve enable
        clr p3.6
rr1:    acall rcdata            ;master address    <1>
rr2:    lcall rcdata            ;slave address   <2>
rr3:    acall rcdata            ;command       <3>
rr4:    mov a,#60h
        add a,addr2
```

```
        mov r0,a
        lcall rcdata            ;read status      <4>
        mov @r0,a               ;store status
        acall delay

read:   setb p3.6
        mov a,#60h
        add a,addr2
        mov r0,a
        mov a,@r0
        mov sbuf,a              ;send status to dce (read)
        acall sdata
        acall delay
        clr p3.6
        ret
            end
```

โปรแกรมสำหรับเครื่อง PC

```
/*************************************************
*************** PROGRAM FOR PC ******************
*************************************************/

#include<dos.h>
#include<stdio.h>
#include <time.h>
#include<conio.h>

int  addr=1 , flag=0 , cursor=1;
char ch ;
unsigned char command , status[32] , temp_stat;
unsigned char b;
int xx , yy ;
unsigned char connect[2][4];

main()
{
  unsigned char discon0;
  int i,v,h;
  clrscr();
  slide();
  poll();

  for(i=1;i<=30;i++)
  {
      if((i % 8) == 1)
{
  discon0 = connect[0][(i-1)/8];
```

```
    b = 1;
}
        else
    b = b << 1;
        if((discon0 & b) != 0)
display(i);
        else
        {
    v = (i/17)*40 + 1 ;
    h = (i%17) + (i/17) ;
    gotoxy(v,h);
    printf("%2d    XXXXXXXX",i);
        }
    }
    for(;;)
     {
       poll();
       delay(200);
       check_key();
     }
}

poll()
{
    unsigned char data_out , discon0,discon1 , temp ,temp_w;
    int k , h , v , l ;
    register unsigned char b = 1;

    for(k=0;k<=3;k++)
```

```
    {
      data_out = 128 + k;
      outportb(535,data_out);
      delay(1);
      connect[1][k] = inportb(535);
    }
   for(k=1;k<=30;k++)
    {
      if((k % 8) == 1)
{
  discon0 = connect[0][(k-1)/8];
  discon1 = connect[1][(k-1)/8];
  b = 1;
}
      else
  b = b << 1;

if((discon1 & b) != 0)
  {
  if(flag==1 && k==addr)
     command = 64;
  else
     command = 32;
  data_out = command + k;
  outportb(535,data_out);
  delay(1);
  temp = inportb(535);
  if(temp != status[k])
  {
```

```
      outportb(535,data_out);
      delay(1);
      temp = inportb(535);
    }
      if(temp != status[k])
     {
        status[k] = temp;
        display(k);
     }
    if(command==64)
      {
        outportb(535,temp_stat);
        delay(1);
        temp_w = inportb(535);
        flag = 0;
      }
}
        else if((discon0 & b) != 0)
{
   v = (k/17)*40 + 1 ;
   h = (k%17) + (k/17) ;
   gotoxy(v,h);
   printf("%2d    XXXXXXXX",k);
}
    }
      for(l=0;l<4;l++)
        connect[0][l] = connect[1][l];
      gotoxy(xx,yy);
}
```

```
check_key()
{
   while(kbhit())
     {
       ch = getch();
       switch(ch)
{ case '0': zero();  break;
  case '1': one();   break;
  case 75:
     {
       if(cursor==1)
 cursor = 8;
       else
 cursor=cursor-1;
       slide();
       break;
     }
  case 77:
     {
       if(cursor==8)
 cursor = 1;
       else
       cursor=cursor+1;
       slide();
       break;
     }
  case 80: down();  break;
  case 72: up();
  }
```

```
    }
  }


  up()
   {
     int x , y ;
      if(addr==1)
        addr = 30;
      else
        addr = addr - 1;
      slide();
    }


  down()
   {
      if(addr==30)
        addr = 1;
      else
        addr = addr + 1;
      slide();
   }


slide()
{
      xx = (addr/17)*40 + 5 + cursor;
      yy = (addr%17) + (addr/17) ;
      gotoxy(xx,yy);
}
```

```
zero()
{
    int i;
    unsigned char and_zero = 128;

      putch('0');
      temp_stat = status[addr];
      for(i=1;i < cursor;i++)
        and_zero = and_zero >> 1;
      if(cursor==8)
        cursor = 1;
      else
        cursor = cursor + 1;
      slide();
      and_zero = ~(and_zero);
      temp_stat = (temp_stat & and_zero);
      flag = 1;
}

one()
{
   unsigned char or_one = 128;
   int j;

     putch('1');
     temp_stat = status[addr];
     for(j=1;j < cursor;j++)
       or_one = or_one >> 1;
     if(cursor==8)
```

```
        cursor = 1;
      else
        cursor = cursor + 1;
      slide();
      temp_stat = (temp_stat | or_one);
      flag = 1;
}

display(posit)
int posit;
{
   int x , y , i ;
   register int a , b = 128 ;
     x = (posit/17)*40+1;
     y = (posit%17)+(posit/17);
     gotoxy(x,y);
     printf("%2d",posit);
     x = x + 5 ;
     gotoxy(x,y);
     a = status[posit];
     if(a&b)
       putch('1');
     else
       putch('0');
     for(i=1;i<=7;i++)
       {
  b = b >> 1;
  if(a&b)
     putch('1');
```

```
    else
        putch('0');
          }
}
```

ภาคผนวก ค

คู่มือการใช้งาน

คู่มือการใช้งาน

DTE (Data Terminal Equipment) - คืออุปกรณ์ที่ใช้ควบคุมอุปกรณ์โดยตรง 1 ตัว ประกอบด้วย 8 channels ซึ่งแทนด้วย LED 8 ดวง และมีสวิทช์ควบคุม 8 ตัว

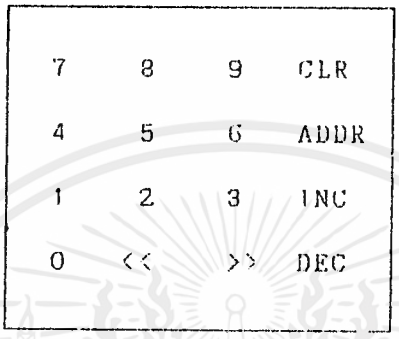
โปรแกรมผู้ใช้อัดใส่ ROM - DTE.ASM(DTE_CHE.ASM)

การใช้ 1.ก่อนการเริ่มต้นจ่าย supply ให้วงจร คือ ก่อนการ RESET วงจรต้องเซท dip-switch เพื่อกำหนดหมายเลขของอุปกรณ์ก่อน เมื่อจ่ายไฟเลี้ยงวงจร หรือ RESET วงจรก็จะได้อุปกรณ์มีหมายเลขตามนั้น

2.การ เปิด-ปิด อุปกรณ์ แทนด้วยหลอด LED 8 ดวงควบคุมได้โดยสวิทช์ 8 ตัว สามารถทำได้โดยการกดปุ่มสวิทช์ควบคุม ซึ่งทำงานแบบ TOGGLE คือ เมื่อกดปุ่ม หลอด LED จะเปลี่ยนสถานะเป็นตรงข้ามจากดับเป็นติดและจากติดเป็นดับ

CONTROLLER - คืออุปกรณ์ที่ใช้ตรวจดูสถานะของ DTE ประกอบด้วย คีย์บอร์ดควบคุมแบบ 16 คีย์ และใช้จอ LCD ในการแสดงผล

โปรแกรมที่ใช้ - C1.ASM



เมื่อเปิดเครื่อง ที่จอจะแสดงสถานะของอุปกรณ์ DTE หมายเลข 01 และ CURSOR

การเปลี่ยนหมายเลขอุปกรณ์ DTE ที่ต้องการดูสถานะ ทำได้โดยการกดคีย์ 'INC'และ'DEC' การเลื่อน CURCOR ทำได้โดยใช้คีย์ '<<' และ'>>' เพื่อเลื่อนไปทางซ้ายหรือทางขวา การเปลี่ยนสถานะของอุปกรณ์ที่ดูอยู่ทำได้โดยการเลื่อน CURSOR ไปที่ตำแหน่งนั้น แล้วกดหมายเลข '1' หรือ '0' เพื่อเปิดหรือปิดอุปกรณ์นั้น เมื่อกดคีย์ '0' หรือ '1' แล้ว CURSOR จะเลื่อนไปทางขวา 1 ตำแหน่ง

CARD – คืออุปกรณ์ควบคุมที่ใช้เชื่อมต่อกับเครื่อง PC การตรวจสอบสถานะและการควบคุมการเปิด-ปิด สามารถทำได้โดยผ่านทางเครื่อง PC

โปรแกรมที่ใช้ สำหรับ CARD คือ CARD.ASM

สำหรับเครื่อง PC คือ PC.EXE (PC.C)

การใช้ เมื่อเปิดเครื่อง PC ที่มี CARD เสียบอยู่และเชื่อมต่อกับระบบโครงข่ายควบคุม ขั้นแรกก็ต้อง RUN โปรแกรม PC.EXE จากนั้นที่จอภาพจะแสดงหมายเลขและสถานะของอุปกรณ์ DTE ทั้ง 30 ตัว ถ้าติดแสดงด้วย '1' ถ้าดับแสดงด้วย '0' และถ้าหากหมายเลขใดไม่ได้ต่ออยู่กับโครงข่าย ก็จะแสดงด้วยสถานะ 'XXXXXXXX' การควบคุมการ เปิด-ปิด ทำได้โดยใช้คีย์ลูกศรในการเลื่อน CURSOR และใช้คีย์ '0' หรือ '1' แทนการ ปิด-เปิด อุปกรณ์

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้สำเร็จลุล่วงได้ เนื่องด้วยการอนุเคราะห์คำปรึกษาที่ดีจาก รศ.ดร. ชม กิ้มปาน อาจารย์ สมศักดิ์ มิตะถา และ อาจารย์ ธันวา ศรีประโมง จึงใคร่ขอขอบคุณท่านผู้ได้กล่าวนามมา ณ ที่นี้

หนังสืออ้างอิง

บริษัท อีทีที จำกัด , "MCS-51 Microcontrollers".

Kerneth J. Ayala, "The 8051 Microcontroller Architecture Programming and Applications", West Publishing Company.

John Uffenbeck, "Microcomputers and Microprocessors", Pentice-Hall International Inc., 1985

ชานินทร์ ถาวรศาสนวงศ์ และ ทินกร ตุ๊ก, "การอินเทอร์เฟส IBM PC", สำนักพิมพ์ ฟิสิกส์เซ็นเตอร์การพิมพ์

Uyless D. Black, "Data Network", Prentice-Hall International Editions , 1989.