ปีการศึกษา 2532

การสื่อสารข้อมูลโดยผ่านคลื่นความถี่วิทยุ

โดย

นางสาวกรชุลี ใช้สถิตย์ 291004

นายคเณศ นิพัทธ์สุขกิจ 291015

นายพงศ์ธวัช ริมดุสิต 291145

อาจารย์ที่ปรึกษา

อ.พลผดุง ผดุงกุล

ปริญญานิพนธ์ปีการศึกษา 2532

เรื่อง การสื่อสารข้อมูลโดยผ่านคลื่นความถี่วิทยุ
DATA TRANSMISSION BY RADIO FREQUENCY

ผู้จัดทำ

1. น.ส.	กรชุลี	ใช้สถิตย์	29.1004
2. นาย	คเณศ	นิพัทธ์สุขกิจ	29.1015
3. นาย	พงศ์ธวัช	ริมดุสิต	29.1145

..... อาจารย์ที่ปรึกษา

(อาจารย์ พลผดุง ผดุงกุล)

การสื่อสารข้อมูลโดยผ่านคลื่นความถี่วิทยุ

DATA TRANSMISSION BY RADIO FREQUENCY

กรชุลี ใช้สถิตย์ 29.1004
คเณศ นิพัทธ์สุชกิจ 29.1015
พงศ์ธวัช ริมดุสิต 29.1145
อาจารย์ที่ปรึกษา :
อาจารย์พลผดุง ผดุงกุล
ปีการศึกษา 2532

บทคัดย่อ

โครงงานนี้ทำขึ้นเพื่อใช้ในการติดต่อระหว่างระบบที่ควบคุมด้วยไมโครโปสเซสเซอร์ (Microprocessor) 2 ระบบหรือมากกว่า กล่าวคือในระบบรวมทั้งหมดจะประกอบด้วยตัวแม่ (MASTER) และตัวลูก (SLAVE) โดยรหัสของเครื่องตัวลูก (ADDRESS) นั้นใช้ตัวเข้ารหัส (Encoder) และตัวถอดรหัส (Decoder) แบบ 2 สถานะ 8 บิต ฉนั้นจึงสามารถมีเครื่องลูกได้ทั้งหมด 2^8 เครื่อง การติดต่อกันของแต่ละเครื่องนั้นทำได้โดยให้ตัวแม่เป็นตัวเรียกตัวลูกโดยส่งรหัสของตัวลูกมอดูเลท (Modulate) ไปกับความถี่วิทยุย่านความถี่ประมาณ 300 เมกะเฮิร์ท (Megahertz) เมื่อตัวลูกรับสัญญาณได้ก็จะทำการดีมอดูเลท (Demodulate) แล้วนำรหัสที่ได้มาเปรียบเทียบกับรหัสของตัวเองที่เราตั้งไว้ที่ตัวถอดรหัส ถ้าตรงกันตัวถอดรหัสจะมีเอาพุท (output) ออกไปทำการอินเตอร์รัพท์ (Interrupt) หน่วยประมวลผลกลาง (CPU) ของระบบตัวลูกเครื่องนั้น แล้วตัวลูกก็จะส่งสัญญาณตอบรับการอินเตอร์รัพท์ (INTACK) กลับมาที่ตัวแม่จากนั้นตัวแม่ก็จะเริ่มติดต่อกับตัวลูกเครื่องนั้นโดยไม่เกี่ยวข้องกับเครื่องอื่น โดยจะมีการติดต่อกันเป็นอะซิงโครนัส (Asynchronus) โดยใช้ 8251 เป็นพอร์ทอนุกรม (Serial Port) แบบฮาล์ฟดูเพล็กซ์(Half duplex) และมีอัตราโบด์ (Buad Rate) เท่ากับ 9600 ใช้กับหน่วยประมวลผลกลางเบอร์ 8088 ในการติดต่อและใช้ความถี่วิทยุย่าน 300 เมกะเฮิร์ท เป็นคลื่นพาห์โดยเป็นการผสมคลื่นแบบพัลส์โคด (Pulse Code Modulation)

DATA TRANSMITTION BY RADIO FREQUENCY

KONCHULEE CHAISATID 291004
KANESE NIPATSUKKIT 291015
PONGTHAWAT RIMDUSIT 291145
ADVISER
POLPADUNG PADUNGKUL
YEAR 1989

ABSTRACK

THIS PROJECT IS MADE FOR CONTACT BETWEEN SYSTEM WHICH IS CONTROLLED BY TWO SYSTEM OR MORE MICROPROCESSOR. IN SYSTEM IS CONSIST OF MASTER AND SLAVE. ADDRESS OF SLAVE USES TWO STATES EIGHT BITS TYPE ENCODER SO WE CAN USE 2^8 SLAVES. THE COMMUNICATION OF EACH SYSTEM WORKING CALLS SLAVE BY SEND CODE OF SLAVE 'S ADDRESS AND MODULATE WITH , 300 MHz , WHEN SLAVES RECIEVE THIS CODE , THEY WILL DECODE THIS CODE AND COMPARE WITH THEIR ADRESSES IF IT 'S IN THE SAME THE DECODER (ONLY ONE SLAVE) ,A DECODER WILL SEND OUTPUT. THIS OUTPUT WILL INTERRUPT A CPU OF SLAVE AND SLAVE WILL SEND INTERRUPT ACKNOWLEDGE BACK TO THE MASTER. THEN THE MASTER CAN CONTACT WITH ONLY THIS SLAVE.

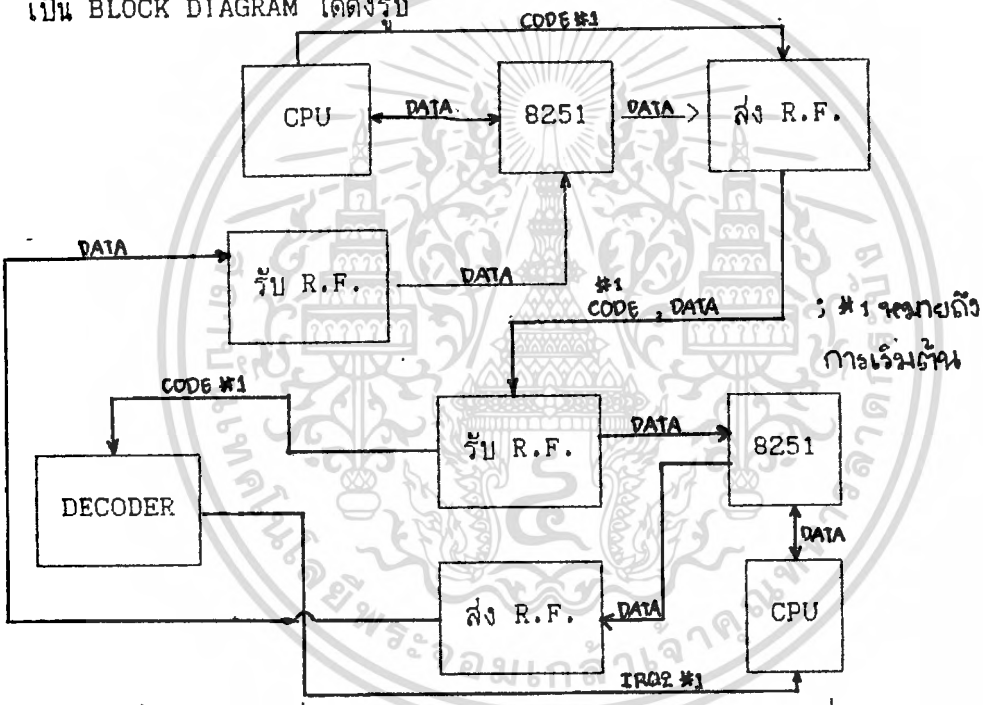
THE COMMUNICATION IS ASYNCHRONUS TYPE BY 8251 , SERIAL PORT , AND COMMUNICATION IS HALF DUPLEX WHICH HAS 9600 BAUD RATE.CPU 'S 8088THIS PROJECT IS MADE FOR COMMUNICATION BETWEEN SYSTEM.

สารบัญ

บทที่ 1

บทนำ

ในปัจจุบันนี้ระบบการสื่อสารข้อมูลเป็นสิ่งที่มีความจำเป็นในวงการคอมพิวเตอร์มากขึ้นเรื่อยๆ ฉนั้นการพัฒนาระบบการสื่อสารข้อมูลจึงเป็นสิ่งที่ความแก่การสนับสนุน โครงงานนี้ทำขึ้นเพื่อศึกษาระบบการสื่อสารระหว่างระบบที่มีไมโครโปรเซสเซอร์ควบคุมหลายระบบโดยจะต้องมีเครื่องหนึ่งเป็นตัว MASTER ส่วนเครื่องที่เหลือก็จะเป็นตัว SLAVE ซึ่งสื่อกลางของการติดต่อทั้งหมดเป็นคลื่นความถี่วิทยุที่ความถี่ 300 MHz เราสามารถเขียนระบบการติดต่อนี้เป็น BLOCK DIAGRAM ได้ดังรูป



เริ่มต้นจากการที่ CPU ของตัว MASTER ต้องการติดต่อกับเครื่องตัว SLAVE ตัวใดตัวหนึ่ง CPU ก็จะทำการส่ง ADDRESS ของตัว SLAVE ที่ต้องการจะติดต่อด้วย (เลือกโดย software) ไปเข้ารหัสโดยใช้ IC เบอร์ MM53200 จากนั้นก็จะนำสัญญาณจากตัวเข้ารหัสนี้ไป Modulate ก็คลื่นพาห์ที่ความถี่ 300 MHz ส่งออกอากาศไป

ที่ตัว SLAVE ทุกตัวจะสามารถรับสัญญาณที่ส่งออกไปนี้ได้ จากนั้นตัว SLAVE จะนำสัญญาณนี้ไปทำการ Demodulate แล้วนำสัญญาณนี้ไปถอดรหัสโดยใช้ IC เบอร์ MM53200 ซึ่งเราจะตั้งรหัสไว้ก่อนแล้ว ตัวถอดรหัสก็จะทำการเปรียบเทียบรหัสของสัญญาณที่เข้ามากับรหัสที่ตั้งไว้ว่าตรงกันหรือไม่ ถ้าตรงกันตัวถอดรหัสก็จะส่ง output ออกมาค่าหนึ่งซึ่งเราจะนำเอา output นี้ไปใช้ในการ INTERRUPT CPU ของตัว SLAVE ตัวนั้นเมื่อ CPU ถูก INTERRUPT แล้วก็จะส่งสัญญาณ CS (CHIP SELECTED) ออกไปให้ 8251 เพื่อที่จะส่งสัญญาณ ACK ตอบรับการติดต่อไปยังเครื่อง MASTER ในขณะนั้นเครื่อง SLAVE ตัวอื่นๆก็จะ

an immitation of dignity

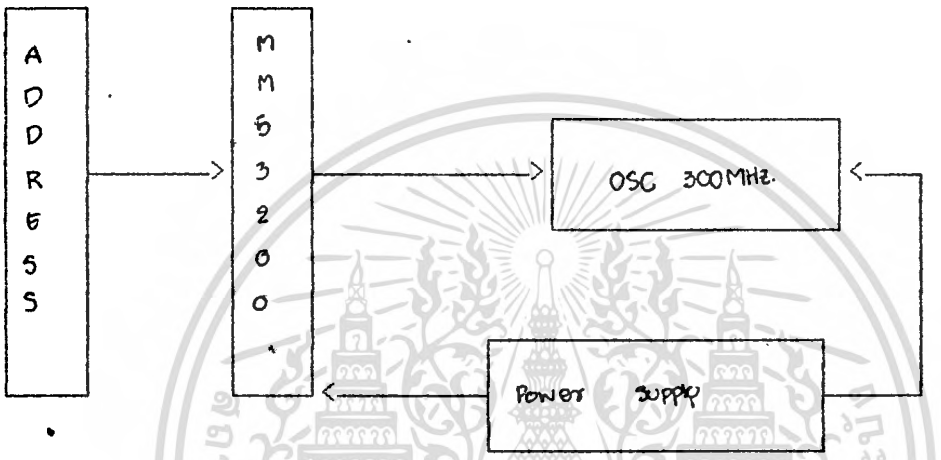
สามารถทำงานต่อไปได้โดยไม่เกี่ยวข้องกับการติดต่อนี้

การติดต่อระหว่างเครื่องทั้งสองต่อไปนี้ก็จะใช้ PROTOCAL แบบ XMODEM ซึ่งจะกล่าวรายละเอียดในบทต่อไป

บทที่ 2

เครื่องส่งและเครื่องรับ

จะขออธิบายถึงเฉพาะส่วนของเครื่องส่ง R.F. เท่านั้น ส่วนระบบการติดต่อระหว่าง CPU กับ PORT (8251) จะกล่าวต่อไปภายหลัง ลักษณะของภาคส่งเป็นดังรูป



ตัว MM 53200 เป็นตัว Encoder ซึ่งจะอธิบายตัวของมันต่อไป ในที่นี้ขอให้ทราบว่า MM 53200 เป็นตัวบังคับให้เกิดการออสซิลเลท คือเมื่อเอาพุทของ MM 53200 มีสถานะสูง (ใกล้ระดับไฟเลี้ยง) จะมีผลทำให้วงจร OSC (ออสซิลเลเตอร์) เกิดการออสซิลเลทที่ความถี่ในย่าน 300 MHz แต่ถ้าเอาพุทของ MM 53200 อยู่ในสถานะต่ำ (ใกล้ระดับกราวด์) จะมีผลให้วงจร OSC หยุดการทำงานเมื่อวงจร OSC เกิดการออสซิลเลทที่ความถี่นี้ คลื่นนี้จะเป็นคลื่นพาห์ของสัญญาณเอาพุทจาก MM 53200 เพื่อส่งออกอากาศไป ลักษณะเช่นนี้อาจกล่าวได้ว่าเครื่องส่งแบบนี้ทำงานในแบบของ Pulse Modulation

หัวใจของวงจรในภาคนี้ คือวงจรกำเนิดความถี่ (Oscillator) หลักการเกิดการ Oscillate มี 2 ประการคือ

1. เฟสชิฟท์ (Phase shift) รอบลูป (Loop) ของระบบมีค่ารวมกันเป็น 0° หรือ $2n\pi$ radian พอดี

2. อัตราการขยายรอบลูป (Loop gian) มีค่าเป็น 1 พอดี ถ้าต้องการให้ขนาดของสัญญาณที่ได้จากการออสซิลเลทมีขนาดคงที่

วงจรออสซิลเลทแบ่งได้เป็น 2 แบบ

1. วงจรออสซิลเลทคลื่นรูป SINE

2.วงจรออสซิลเลทแบบไม่ใช่คลื่นรูป SINE เช่น รูปสี่เหลี่ยม เป็นต้น ในโครงงานนี้เราใช้วงจร OSC แบบคลื่นรูป SINE

วงจร OSC คลื่นรูป SINE จะให้กำเนิดสัญญาณคลื่นรูป SINE ออกมาตลอดเวลา โดยมีขนาดและความถี่คงที่ ชนิดของวงจร OSC แบบนี้แบ่งได้จากอุปกรณ์ที่ใช้ ซึ่งแบ่งได้เป็น

- วงจร OSC ที่ใช้ LC
- วงจร OSC ที่ใช้ RC
- วงจร OSC ที่ใช้ Crystal

ลักษณะเฉพาะของแต่ละแบบคือ

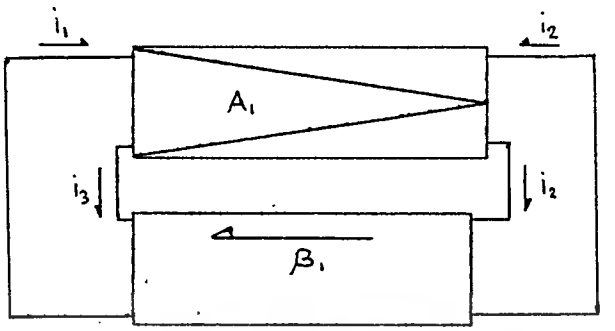
1.วงจร OSC ที่ใช้ LC
- ความถี่เปลี่ยนแปลงได้ง่าย
- โดยทั่วไปเสถียรภาพของความถี่ไม่ดี
- มักใช้กันโดยทั่วไป
- ใช้ในย่านความถี่สูง

2.วงจร OSC ที่ใช้ RC
- ใช้ในช่วงความถี่ต่ำ
- ความถี่สามารถเปลี่ยนแปลงได้
- เสถียรภาพของความถี่ไม่ดี

3.วงจร OSC ที่ใช้ Crystal
- เสถียรภาพของวงจรดีเป็นพิเศษ
- ใช้ในย่านความถี่สูง
- การทำให้ความถี่เปลี่ยนแปลงทำได้ยาก

จากคุณสมบัติของวงจรทั้ง 3 แบบ จะเห็นว่าเราต้องการวงจร OSC ที่ความถี่สูง เราจึงเลือกใช้วงจร OSC แบบที่ใช้ LC

ต่อไปจะกล่าวถึงการออสซิลเลท โดยอุปกรณ์ที่ใช้เป็นลักษณะของกระแส คือเป็นทรานซิสเตอร์ เนื่องจากทรานซิสเตอร์ทำงานในลักษณะกระแสดังนั้นการป้อนกลับมาทางด้านสัญญาณเข้าจะต้องอยู่ในรูปของกระแสเช่นกัน เราสามารถเขียนโครงสร้างของวงจรได้ดังรูป



จากรูปค่า A_1 และ β_1 จะเท่ากับ

$$A_1 = i_2 / i_1$$

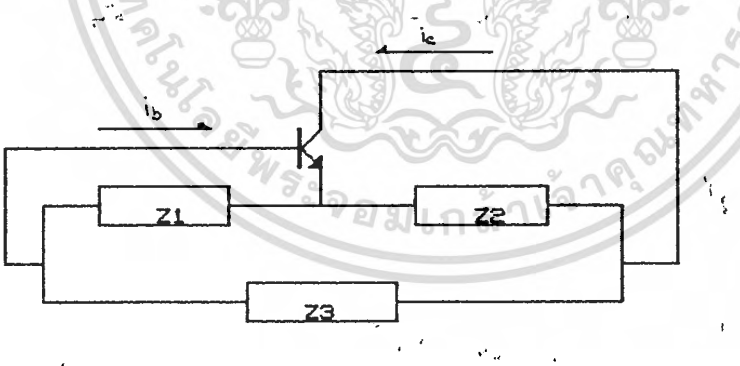
$$\beta_1 = i_3 / i_2$$

การออสซิลเลทจะเริ่มเมื่อ $i_1 < i_3$ และที่ $i_1 = i_3$ การออสซิลเลทจะเริ่มมีค่าคงที่

$A_1 * \beta_1 = i3 / i_1 > 1$ (เริ่มการออสซิลเลท)

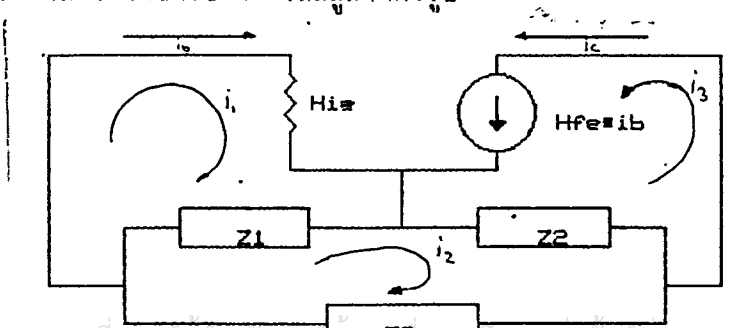
$A_1 * \beta_1 = i3 / i_1 = 1$ (ออสซิลเลทแบบต่อเนื่อง)

ในวงจรที่ใช้ในโครงงานนี้เป็นวงจร OSC แบบ Hartley จะขออธิบายวงจรแบบนี้จากรูป



รูปที่ 1 แสดงวงจร OSC แบบเชื่อมต่อสามจุด

เราสามารถเขียนเป็นวงจรสมมูลได้ดังรูป



รูปที 2 แสดงวงจรสมมูลของวงจรรูปที่ 1

จากรูปจะได้ว่า

$$(h_{ie} + Z_1)i_1 - (Z_1 * i_2) = 0$$

$$(-Z_1 * i_1) + (Z_1 + Z_2 + Z_3)i_3 + (Z_2 * i_3) = 0$$

เพราะว่า $i_1 = i_b$, $i_3 = i_c = h_{fe} * i_b$ ดังนั้นสมการ(1) จะได้เป็น

$$(h_{ie} + Z_1)i_b - (Z_1 * i_2) = 0$$

$$(h_{fe} * Z_2 - Z_1)i_b + (Z_1 + Z_2 + Z_3)i_2 = 0$$

ตามสมการบนจะได้คำตอบก็ต่อเมื่อ $i_b = 0$ และ $i_2 = 0$ ดังนั้นสัมประสิทธิ์ดีเทอมิแนนท์ จะต้องเป็น $= 0$

$$= (h_{ie} + Z_1)(Z_1 + Z_2 + Z_3) + Z_1(h_{fe} * Z_2 - Z_1)$$

$$= h_{ie}(Z_1 + Z_2 + Z_3) + Z_1(Z_2(1 + h_{fe}) + Z_3) = 0 \quad --(3)$$

ถ้าค่า Z_1 , Z_2 และ Z_3 เป็นค่า reactance จริงๆ แล้ว jX_1 , jX_2 และ jX_3 จากสมการที่(3) จะมีค่าเป็น

$$jh_{ie}(X_1 + X_2 + X_3) - X_1 * X_2(1 + h_{fe}) - X_1 * X_3 = 0$$

และจากจำนวนจินตภาพและจำนวนจริงจากสมการบนทำได้ว่า

$$X_1 + X_2 + X_3 = 0 \qquad -----(4)$$

$$X_2(1 + h_{fe}) + X_3 = 0 \qquad -----(5)$$

สมการที่(4) จะแสดงข้อกำหนดของความถี่สัญญาณ ส่วนสมการที่(5) จะแสดงข้อกำหนดของศักดาสัญญาณ จากสมการที่(5) จะได้ว่า

$$X_2 = -(X_3)/(1 + h_{fe}) \qquad -----(6)$$

จากสมการนี้จะเห็นว่า X_2 และ X_3 เป็นค่า reactance ชนิดต่างกันแทนสมการที่(6) ลงในสมการที่(4) ทำให้

$$X_1 = -(h_{fe} * X_3)/(1 + h_{fe}) \qquad -----(7)$$

จากสมการนี้จะเห็นว่าค่า X_1 และ X_3 เป็นค่า reactance ต่างชนิดกันและจากสมการที่(6) และ (7) จะได้ว่า

$$h_{fe} = X_1 / X_2$$

ผลจากสมการเหล่านี้สรุปได้ว่า

1.ค่า reactance ของ X_1 และ X_2 จะต้องเป็นคนละชนิดกับ X_3

2.ค่า reactance ต่างๆ ควรจะเป็นตามข้อกำหนดดังนี้

$$|X_2| < |X_3| \ , \ |X_1| < |X_3|$$

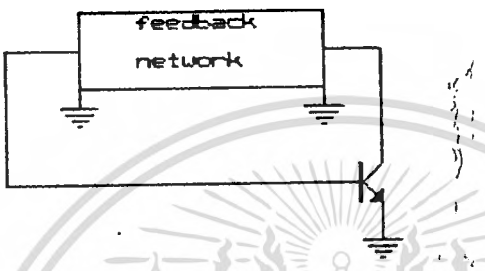
3.h_{fe} ควรจะมีค่าเป็น $h_{fe} = X_1 / X_2$

ค่า h_{fe} ในข้อ 3 เป็นค่าที่การออสซิลเลท เป็นแบบอิ่มตัวแล้ว ถ้าในกรณีของการ

เริ่มการออสซิลเลท ค่า h_{fe} ควรจะเป็นมากกว่า X_1 / X_2

ส่วนข้อ 1 กับข้อ 2 จะทำให้ได้วงจรพื้นฐานเป็น 2 แบบคือแบบ Hartley กับ Colpits คือเป็น Hartley เมื่อ X_1 และ X_2 เป็น L และ X_3 เป็น C ส่วนเป็น Colpits เมื่อ X_1 และ X_2 เป็น C และ X_3 เป็น L

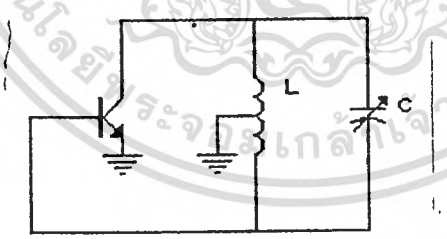
จากรูป



รูปที่ 3 แสดงวงจร OSC แบบ Common Emitter

เป็นวงจรแบบอิมิตเตอร์ร่วม (common emitter) จะเห็นว่า phase shift ของวงจรป้อนกลับต้องมีค่าเป็น 180° เพราะตัวขยายแบบคอมมอนอิมิตเตอร์เองก็มีเฟสที่ต่างกันระหว่างอินพุทกับเอาพุทอยู่ 180° อยู่แล้ว ซึ่งจะทำให้ phase shift รอบลูปมีค่าเป็น 360° พอดี

ในโครงงานนี้เราใช้วงจร OSC แบบ common emitter ชนิด Hartley (Hartley Oscillator)



รูปที่ 4 แสดงวงจร OSC แบบ HARTLEY

จากรูปสามารถอธิบายได้ว่า เมื่อตัวเหนี่ยวนำและตัวเก็บประจุเกิด resonance จะเสมือนเป็นความต้านทานค่าหนึ่งต่ออยู่ทางด้าน Collector ของทรานซิสเตอร์ ดังนั้นจะเกิดการป้อนกลับทาง Base จาก Collector โดยมี phase shift เป็น 180° เนื่องจากเป็นวงจรแบบ common emitter นอกจากนี้ยังมี phase shift อีก 180° เนื่องจากจุดแท็ปของตัวเหนี่ยวนำ จึงทำให้เกิดการออสซิลเลทได้โดยที่ความถี่ของการออสซิลเลทจะเท่ากับ

$$F_o = 1/2\pi\sqrt{LC}$$

ในโครงงานนี้เราเลือกใช้ C เป็นแบบปรับค่าได้ขนาด 5-30 PF เพื่อให้ง่ายต่อ

การปรับเลือกความถี่อื่นๆมาใช้ ฉนั้นเราจึงสามารถคำนวณหาค่า L ได้เมื่อกำหนดให้ค่า C มีค่าเท่ากับ 15 PF จากสมการ

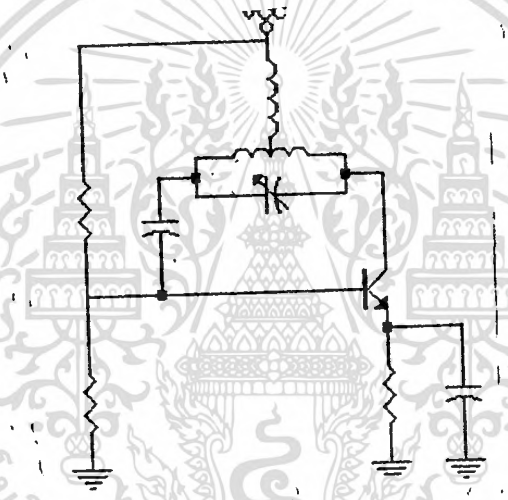
$$F_o = 1/2\pi\sqrt{LC}$$

$$300*10^6 = 1/2\pi\sqrt{L*15*10^{-12}}$$

$$L = 18.8 \text{ nH}$$

ซึ่งเป็นค่าที่น้อยจึงเลือกทำ L บนลายทองแดง

วงจรจริงที่ใช้ในโครงงานนี้เป็นดังรูปโดยแสดงวงจร Bias และตัวเก็บประจุ Bypass ไว้ด้วยที่ขา emitter เพื่อใช้ในการเพิ่ม gain ที่ความถี่สูง



รูปที่ 5 แสดงวงจรจริงที่ใช้พร้อมการ BIAS

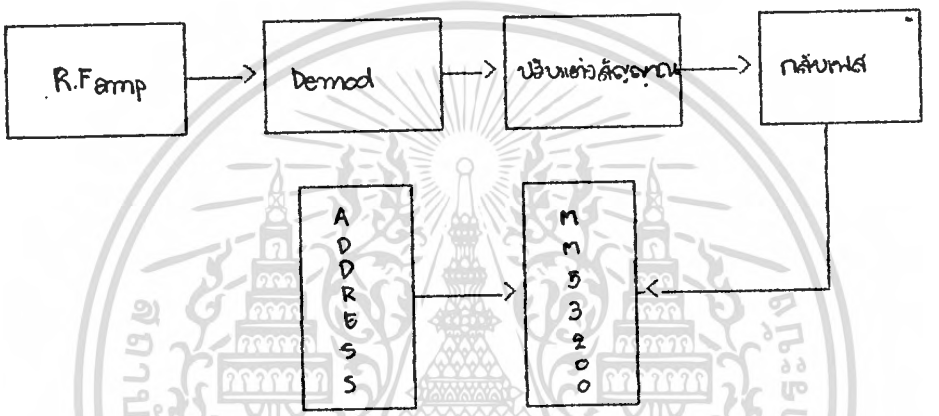
ในวงจรจริงที่ใช้งานนั้นเนื่องจากตัว MASTER กับ ตัว SLAVE นั้นมีส่วนที่ทำงานไม่เหมือนกันกล่าวคือตัว MASTER นั้นจะต้องมีการเลือก ADDRESS ของเครื่อง SLAVE ด้วย จึงต้องมีการใส่ตัว ENCODE ที่วงจรของตัว MASTER ดังจะขออธิบายวงจรดังนี้

จากตัว ENCODER และจาก 8251 ก็จะป้อนสัญญาณให้กับ MULTIPLEXER เพื่อเลือกสัญญาณออกไปใช้เพียงสัญญาณเดียวจากนั้นเราก็ใช้สัญญาณมาควบคุมตัว MULTIPLXER ซึ่งจะต้องคุม 2 ขาคือขา 11 ซึ่งจะเป็นตัวเลือกว่าจะให้สัญญาณตัวไหนออกมาใช้งาน ส่วนอีกขาคือขา 6 ซึ่งใช้ควบคุมการ enable การทำงาน สัญญาณทั้งสองนี้ได้มาจากการ out port address 1 bit ในส่วนนี้ทั้ง IC ENCODER และ MULTIPLEXER เราใช้ไฟเลี้ยงขนาด 5 โวลท์แต่เพื่อให้การส่งมีคุณภาพดีขึ้นคือสามารถส่งได้ไกลที่สุด จึงให้ภาคส่งมีขนาดไฟเลี้ยง 9 โวลท์ และขยายสัญญาณที่จะส่งให้มีขนาด 9 โวลท์ด้วยซึ่งทำได้โดยการขยายโวลท์ด้วยทรานซิสเตอร์ต่อเป็นแบบสวิทช์ซึ่ง เมื่อมีสัญญาณเข้ามาที่ขาเบสให้ทรานซิสเตอร์ทำงานสัญญาณ output เป็น 9 โวลท์และเมื่อไม่มีสัญญาณที่ขาเบสทรานซิสเตอร์ก็จะไม่ทำงานสัญญาณ output เป็น 0 โวลท์แล้วจึงนำสัญญาณนี้ผ่านภาคออสซิสเลเตอร์ส่งออก

อากาศไป ส่วนในตัว SLAVE นั้นจะมีการส่งสัญญาณเพียงของ 8251 เท่านั้นเราจึงใช้ตัว MULTIPLEXER เพื่อควบคุมการทำงานของภาคนี้เท่านั้นแล้วนำสัญญาณที่จะส่งมาขยายเช่นเดียวกับของตัว MASTER

เครื่องรับ

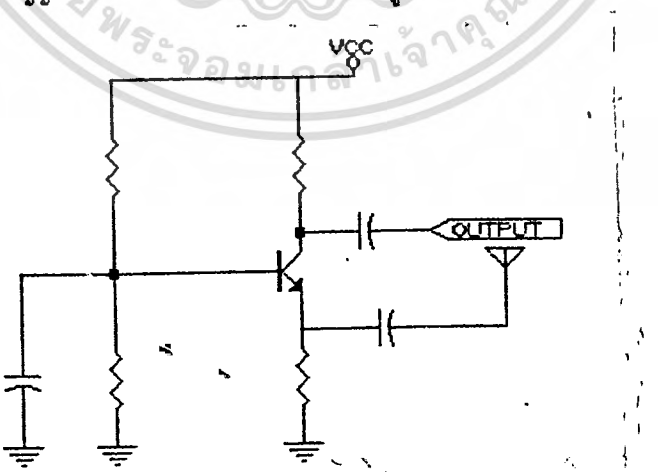
หลักการของเครื่องรับสามารถเขียนเป็น Block Diagram ได้ดังนี้



จะขออธิบายวงจรโดยแบ่งเป็นส่วนๆ เพื่อความเข้าใจได้ง่ายดังต่อไปนี้

R.F. Ampliflier

เนื่องจากสัญญาณที่เข้ามาทางสายอากาศนั้นมีค่าต่ำมาก ฉนั้นก่อนที่จะนำสัญญาณนี้ไปใช้จึงควรที่จะขยายสัญญาณก่อน ซึ่งวงจรขยายเป็นดังรูป



รูปที่ 6 แสดงวงจร R.F. Ampliflier

เป็นวงจรขยายสัญญาณแบบ Common Base เนื่องจากวงจรขยายแบบนี้จะให้ Freq. Response สูงกว่าวงจรขยายแบบอื่นๆ คือกล่าวคือมีช่วงของ Band Width กว้างจึงนำมาใช้ขยายสัญญาณในโครงงานนี้ได้เป็นอย่างดี อัตราขยายของวงจรแบบนี้มีค่า

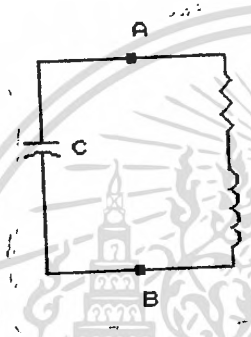
เท่ากับ

$$A_v = Z_c / Z_E$$

<u>วงจรเลือกความถี่ (Demodulation)</u>

เมื่อได้ขนาดของสัญญาณใหญ่พอสมควรแล้ว เราจะนำสัญญาณนี้มาแยกเอาคลื่นพาห์ออกโดยใช้วงจรเลือกความถี่

วงจรนี้จะประกอบด้วย Tank Circuit ซึ่งประกอบด้วยตัวเหนี่ยวนำและตัวเก็บประจุดังรูป



รูปที่ 7 แสดง PARALLEL RESONANCE

ที่ความถี่ Resonance จะทำให้ค่า $X_c = X_L$ ซึ่งหาความถี่นี้ได้จากสมการ

$$W_o = 1/\sqrt{LC}$$

ที่ความถี่นี้ทำให้ค่า Impedance ระหว่างจุด A กับ B มีค่าสูงสุด ทำให้กระแสไหลผ่านได้น้อยที่สุด ในที่นี้เราตั้ง f_o ไว้ที่ 300 MHz ทำให้เราสามารถตัดสัญญาณคลื่นพาห์ออกไปได้ เราสามารถคำนวณค่าของ L ที่ใช้ได้เมื่อตั้งค่า C ไว้ที่ 4 PF

$$L = 1/(4*10^{-12})(2*300*10^{6})^2$$

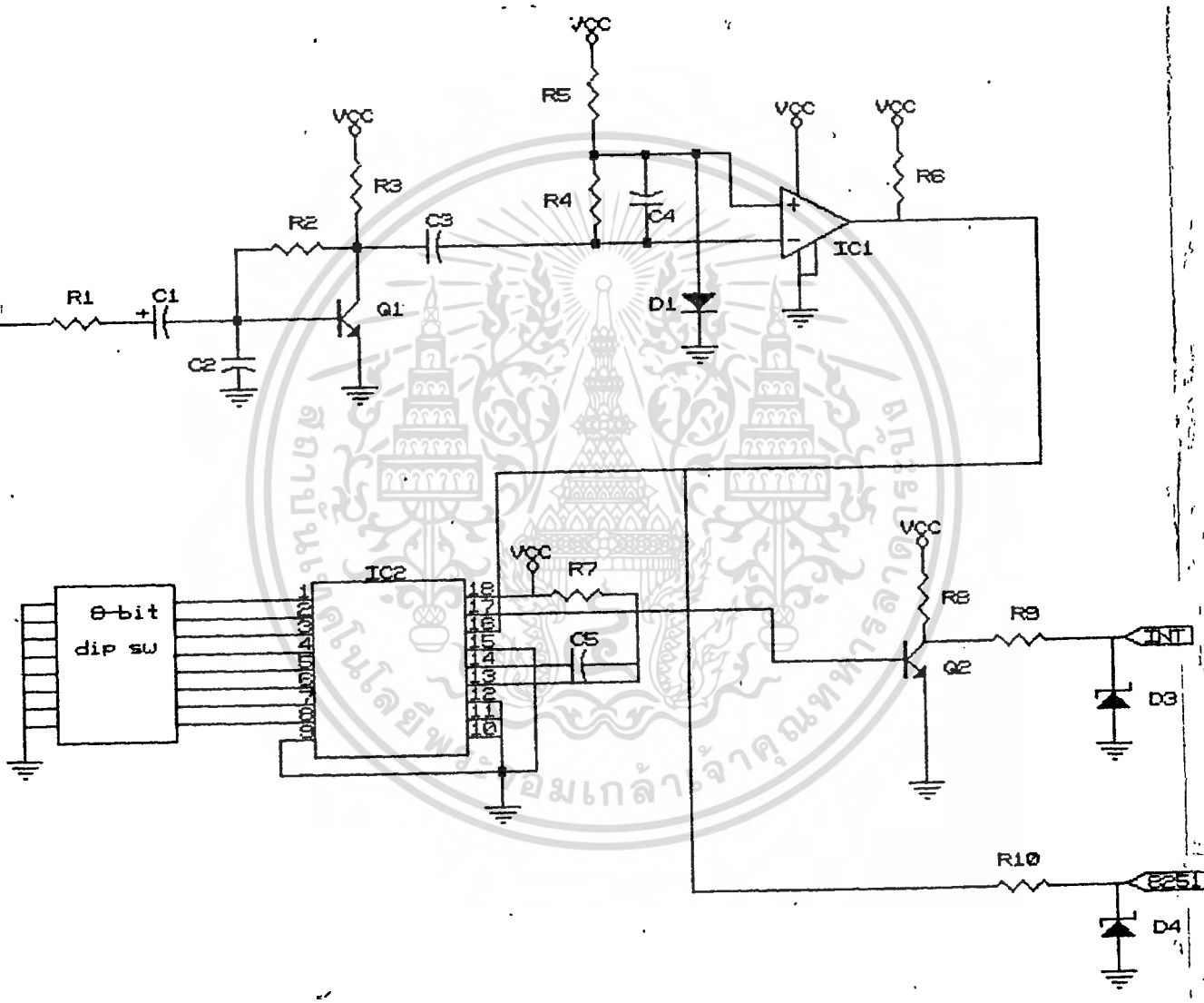
$$= 70 \text{ nH}$$

ในวงจรจริงเราใช้สลักจูนที่แกนเป็นเฟอร์ไรท์ (ferrite) พันลวดไว้ประมาณ 1 รอบครึ่ง นอกจากนี้เราใช้ R_{FC} เป็นตัวช่วยตัดสัญญาณคลื่นพาห์ที่ยังหลงเหลืออยู่อีกทีหนึ่ง

<u>วงจรปรับแต่งสัญญาณและกลับเฟส</u>

เมื่อได้สัญญาณ output จากภาค Demodulate แล้วเรายังไม่สามารถนำสัญญาณนี้ไปใช้งานได้ เนื่องจากขนาดของสัญญาณ และรูปร่างของสัญญาณยังเพี้ยนไปจากที่ส่งออกมาอยู่ ดังนั้นจึงต้องนำสัญญาณนี้ไปเข้าวงจรแต่งสัญญาณก่อน โดยนำสัญญาณผ่าน R กับ C แล้วเข้าวงจรขยายโดยใช้ทรานซิสเตอร์ต่อให้เป็นวงจรขยายแบบ Common Emitter โดยมี R 2.7 M เป็นตัวกำหนดอัตราขยาย จากนั้นก็นำไปเข้าวงจร Comparater เทียบแรงดันอ้างอิงที่ 0.6 โวลท์ และเนื่องจากวงจรขยายแบบ Common Emitter จะกลับเฟสไป 180° ดังนั้นเราจึงต้องป้อน input เข้าที่ขาลบ เพื่อกลับเฟสสัญญาณอีกครั้ง สัญญาณที่ได้

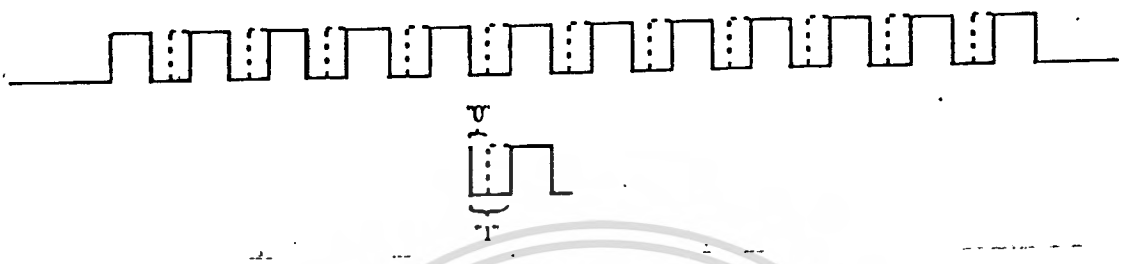
ออกมาจะสามารถนำไปใช้ได้ ซึ่งเราจะนำไปป้อนให้กับ IC DECODER หรือให้กับ 8251 แต่เนื่องจาก 8251 มีไฟเลี้ยงขนาด 5 โวลท์ ส่วนสัญญาณที่ออกมามีขนาด 9 โวลท์ จึงต้องลดระดับสัญญาณลงให้เป็นขนาด 5 โวลท์ ซึ่งเราใช้ ZENER DIODE ดังแสดงในรูป



รูปแสดงวงจรปรับแต่งสัญญาณของเครื่อง SLAVE

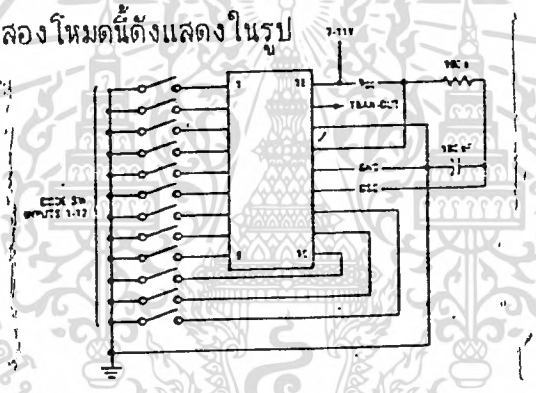
การทำงานของ MM 53200

ไอซีเบอร์นี้เป็นไอซีสำหรับเข้ารหัสและถอดรหัสดิจิตอลขนาด 12 บิต โดยมีการเว้นช่วงว่าง (blanking) และมีสตาร์ทบิตให้โดยอัตโนมัติ ดังรูป

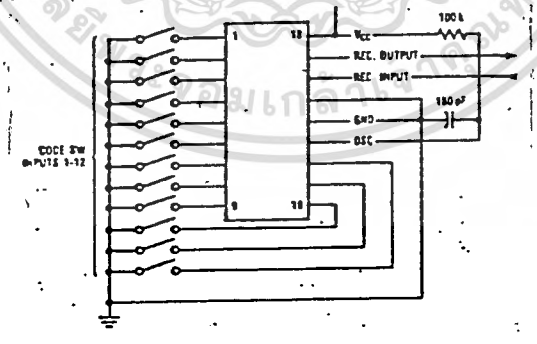


รูปที่ 8 แสดงสัญญาณที่ส่งออกจาก MM 53200 เมื่อใช้งานในโหมดการส่ง

การทำงานของไอซีเบอร์นี้แบ่งออกเป็นสองลักษณะคือ ต่อแบบโหมดการส่งและในโหมดการรับ การต่อในทั้งสองโหมดนี้ดังแสดงในรูป



รูปที่ 9 การต่อใช้งานไอซี MM 53200 ในโหมดการส่งข้อมูล

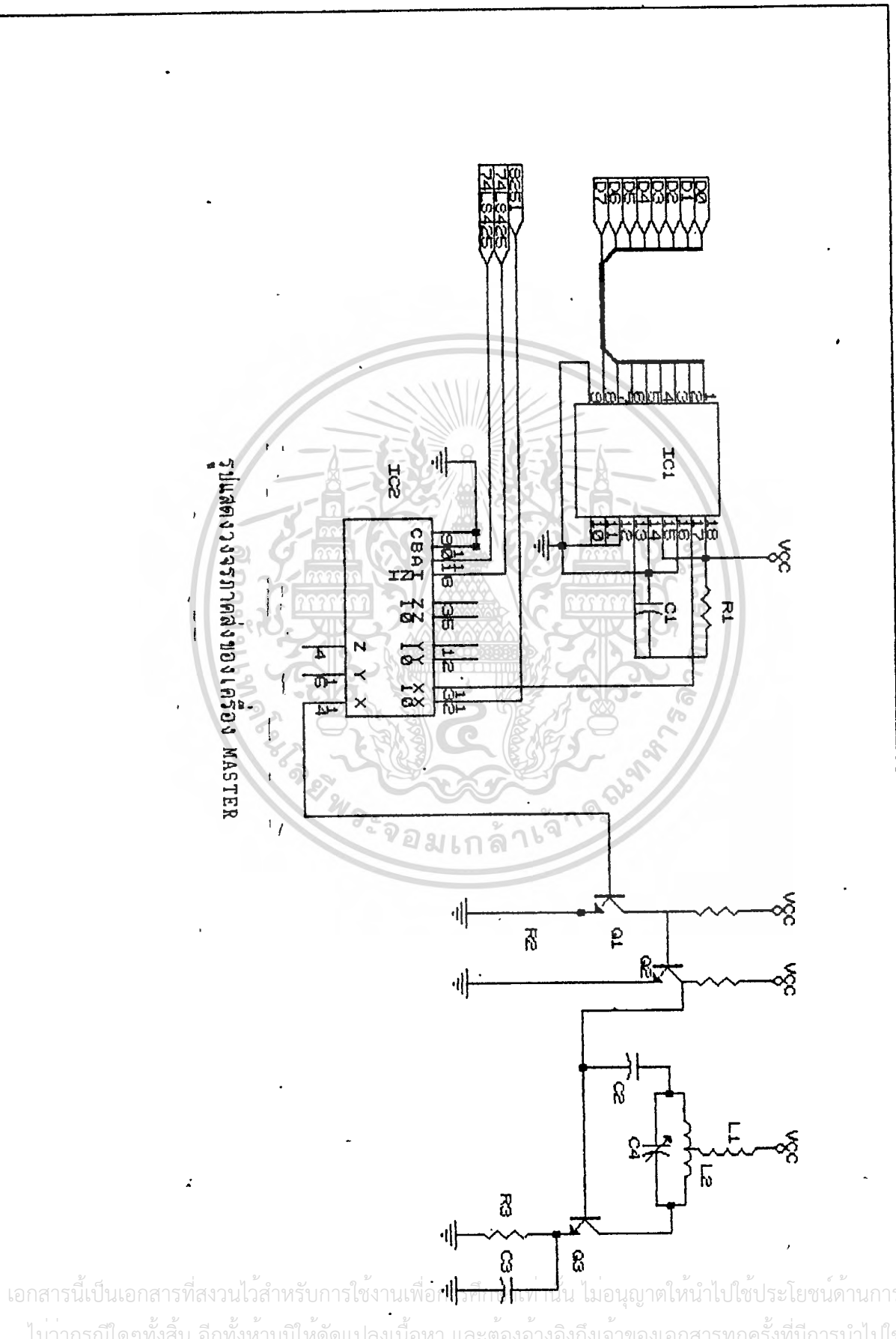


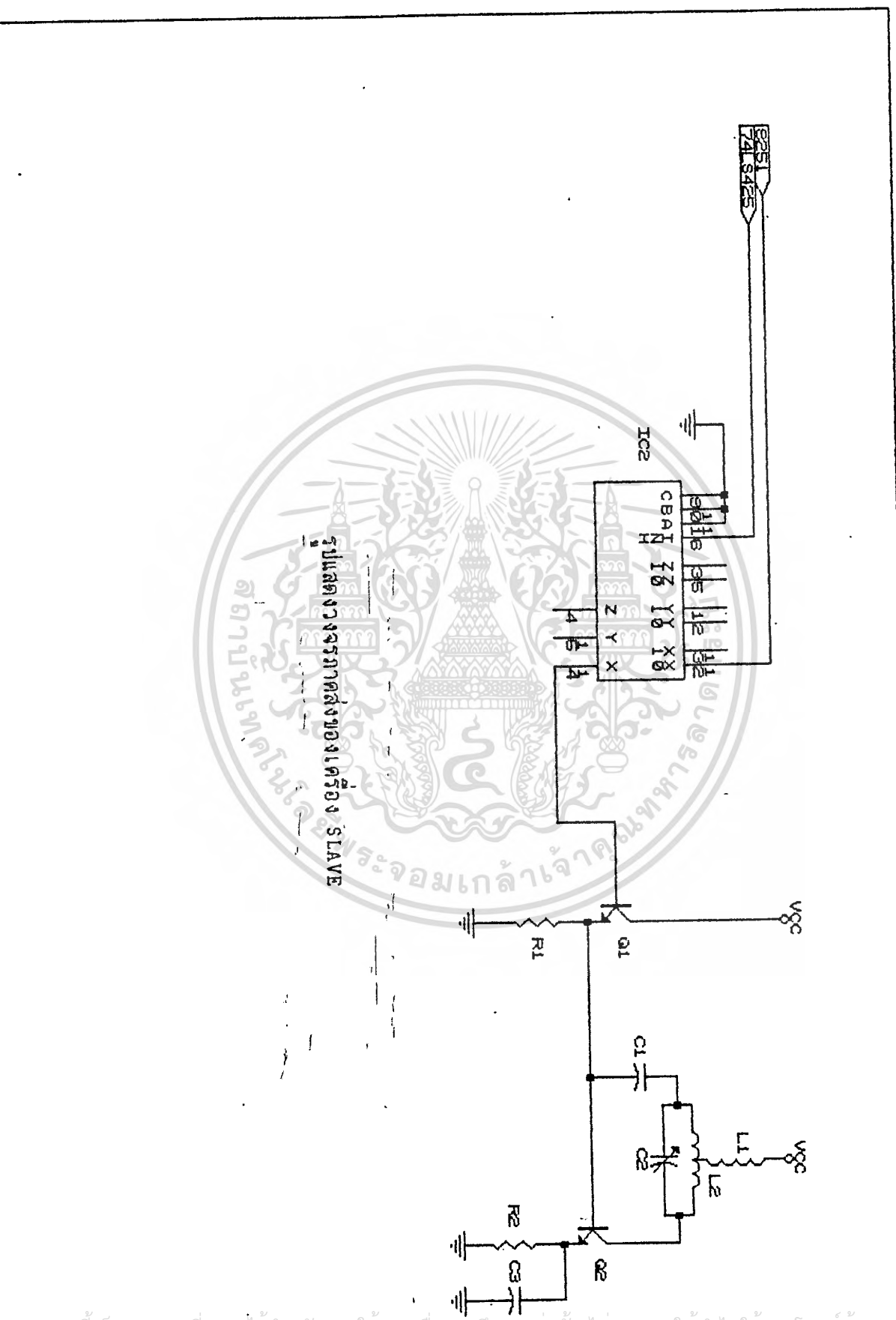
รูปที่ 10 การต่อใช้งานไอซี MM 53200 ในโหมดการรับข้อมูล

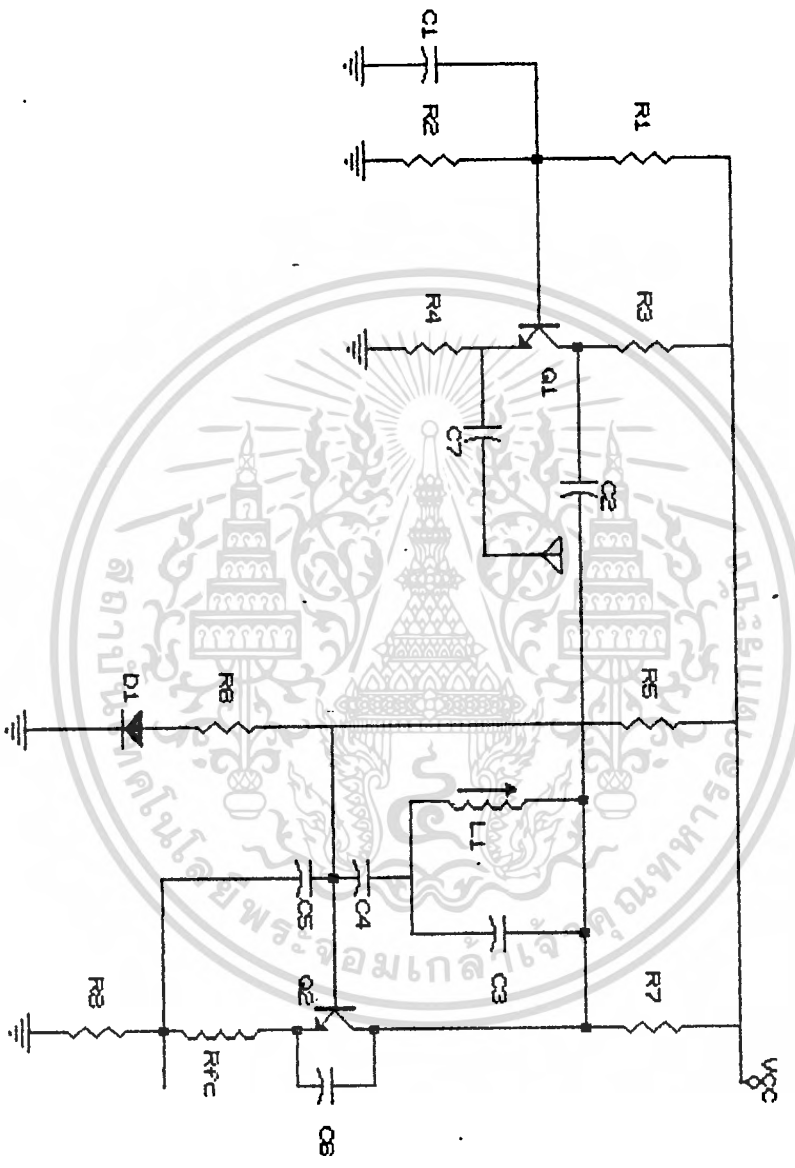
ในโหมดของการส่งข้อมูลนั้น เมื่อเราต่อวงจรถูกต้องและจ่ายไฟเลี้ยงให้แก่ไอซี มันจะกำเนิดสัญญาณดิจิตอลขนาด 12 บิตส่งออกมาทางขา 17 โดยเป็นสัญญาณแบบอนุกรม วนซ้ำอยู่ตลอดเวลาที่ไอซีทำงาน โดยมีลักษณะของสัญญาณเช่นเดียวกับในรูป

ในโหมดการรับนั้น เราต้องต่อใช้งานไอซีดังในรูป เมื่อไอซีทำงานโดยรับข้อมูลเข้าทางขา 16 มันจะทำการเปรียบเทียบรหัสที่มันได้รับขนาด 12 บิต ว่ามีรหัสตรงกับที่ตั้งไว้ด้วยดิพสวิทช์ ที่ต่ออยู่กับตัวมันหรือไม่ โดยทำการเปรียบเทียบในแบบอนุกรม ถ้ารหัสที่รับได้

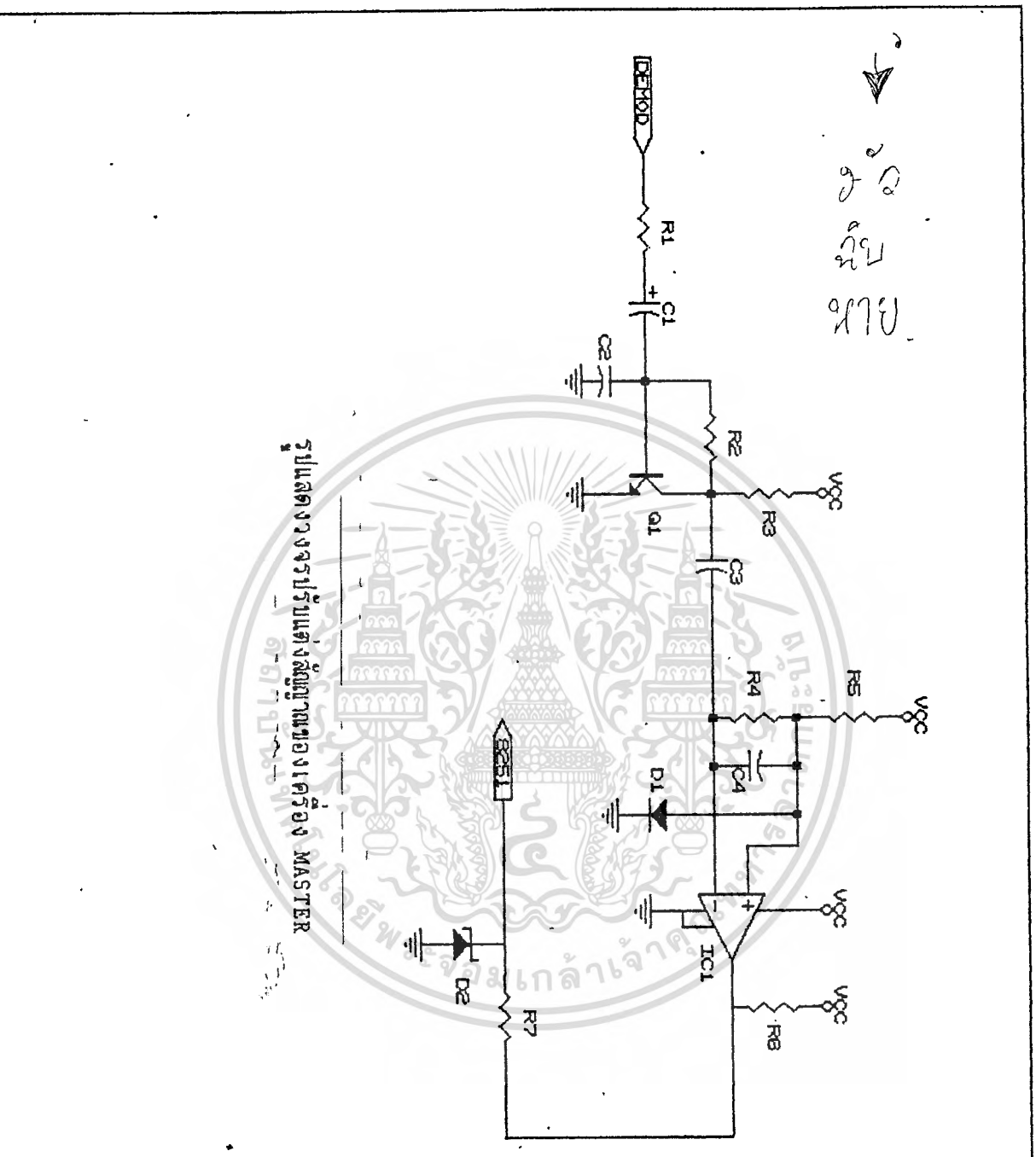
ไม่ตรง ไอซีจะรีเซ็ท ตัวเองแล้วรอเปรียบเทียบรหัสในชุดต่อไป ถ้ารหัสที่ได้รับตรงกันมัน จะรอตรวจสอบว่าตรงกันต่อไปอีก 3 ครั้ง จึงจะทำให้เอาพุทที่ขา 17 เปลี่ยนสถานะจาก สูง เป็น ต่ำ หลังจากนั้นไอซีจะต้องได้รับรหัสที่ถูกต้องอีกภายในเวลา 128 มิลลิวินาที (ที่ความถี่สัญญาณนาฬิกาเท่ากับ 100 KHz) เอาพุตของไอซีจึงจะยังคงสถานะเป็นต่ำ อยู่ต่อไป ถ้าไม่เช่นนั้นแล้ว เอาพุทของไอซีก็จะเปลี่ยนสถานะเป็นสูง แล้วรอทำการตรวจ สอบรหัสอีก 4 ครั้งต่อไป







รูปแสดงวงจร R.F. AMP. และ DEMODULATE



บทที่ 3

ลักษณะการทำงานของวงจร INTERFACE ในการทดลอง

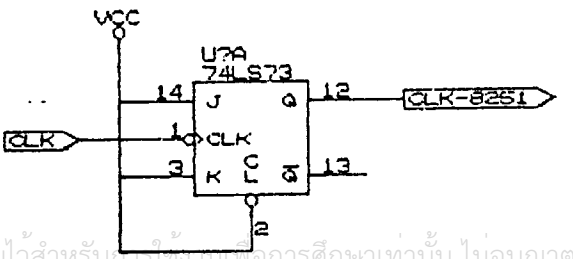
ในการทดลองนี้จะใช้ 8251 2 ตัว ตัวหนึ่งเป็น MASTER อีกตัวหนึ่งเป็น SLAVE เพื่อใช้ในการสื่อสารข้อมูลอนุกรมแบบ HALF DUPLEX คือเป็นการผลัดกันรับ-ส่งข้อมูลระหว่าง CPU 8088 2 ตัว (MASTER และ SLAVE) และเป็นการสื่อสารแบบอะซิงโครนัส รูปแบบโปรโตคอลคือ X-MODEM สำหรับการทดลองในภาคเรียนนี้ จะให้ SLAVE RUN PROGRAM รอการ INTERRUPT จาก MASTER ก่อน แล้วจึงจะให้ MASTER ส่งสัญญาณ INTERRUPT (IRQ2) ซึ่งจะได้สัญญาณนี้มาจากการถอดรหัสเพื่อเลือก ADDRESS ของเครื่อง SLAVE นี้ เมื่อ SLAVE ได้รับสัญญาณ IRQ2 แล้ว จึงจะดำเนินการติดต่อรับ-ส่งข้อมูลกับ MASTER ในแบบ X-MODEM โดยผ่านทางความถี่คลื่นวิทยุต่อไป

HARDWARE ที่เกี่ยวกับ 8251

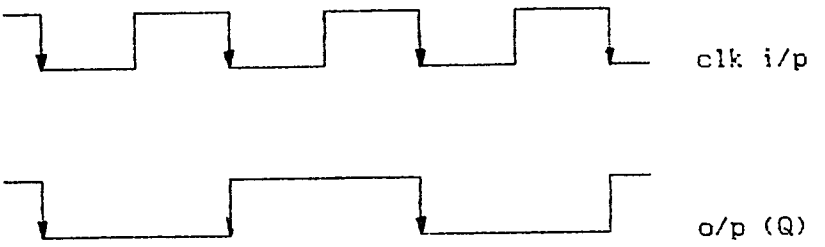
1. CLOCK

สำหรับ clock ที่ 8251 ต้องการต้องไม่เกิน 3 MHz ซึ่ง clock ที่มาจาก CPU 8088 มีค่าเท่ากับ 4.77 MHz ดังนั้นเราจึงต้องใช้ JK Flip-Flop มาเพื่อหาร clock ของ CPU ลงให้เหลือครึ่งหนึ่งคือเหลือเท่ากับ 2.39 MHz ก่อนจึงจะนำมาจ่ายให้ 8251 ได้ ตามวงจรในรูปที่ 1 และผลลัพธ์ในรูปที่ 2

INPUT				OUTPUT
CLR	CLK	J	K	Q
H	↓	H	H	TOGGLE



รูปที่ 1



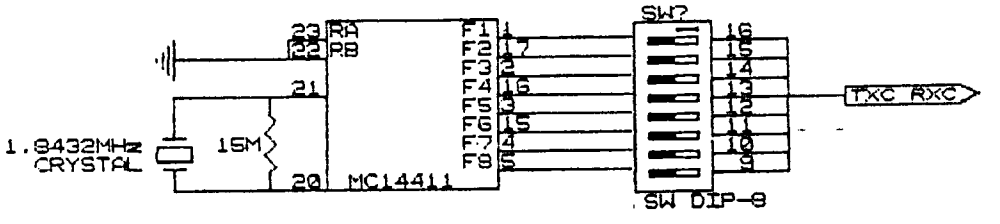
รูปที่ 2

2.TxC และ RxC (transmitter clock และ recieve clock)

ขานี้ใช้กำหนดอัตราโบด์ (Baud Rate) ของการส่งและรับข้อมูลของ 8251 ทั้ง MASTER และ SLAVE ต้องใช้อัตราโบด์เดียวกัน สำหรับการทดลองนี้ใช้อัตราโบด์เท่ากับ 9600 และวงจรที่ใช้กำเนิดอัตราโบด์ทั้ง MASTER และ SLAVE ใช้วงจรเดียวกันคือ วงจรรูปที่3 และมีตารางแสดงการทำงานของ MC 14411 ดังตารางข้างล่าง

RATE SELEECT		RATE
A	B	
0	0	*1
0	1	*8
1	0	*16
1	1	*64

O/P NUMBER	O/P RATE *1
F1	9600
F2	7200
F3	3600
F4	2400
F5	1800
F6	1200
F7	600
F8	300



รูปที่ 3

3.$\overline{CS}$ (CHIP SELECT)

เราใช้ 74LS138 ในการ DECODE เบอร์ PORT โดยเราเลือกใช้ PORT เบอร์ Ø3Ø1H เป็น PORT CONTROL และ PORTเบอร์ Ø3ØØH เป็น PORT DATA

ในการเลือกเบอร์ PORT ที่จะเป็น CHIP SELECTนั้นเราจะต้องให้ A_9 จาก 8088 เป็น 1เสมอ ถ้าเราใช้อุปกรณ์ที่อยู่นอกเหนือจากอุปกรณ์ที่อยู่บนMAIN BODRD และให้ AEN (ADDRESS ENABLE) จะต้องเป็น Ø ตลอดถ้าไม่เป็นการทำ DMA ส่วน A_0 จะเป็นตัวเลือกว่าจะเป็น PORT CONTROL หรือ PORT DATA โดยต่อไว้กับขา C/$\overline{D}$ ของ 8251 ซึ่งจะกล่าวในตอนต่อไป

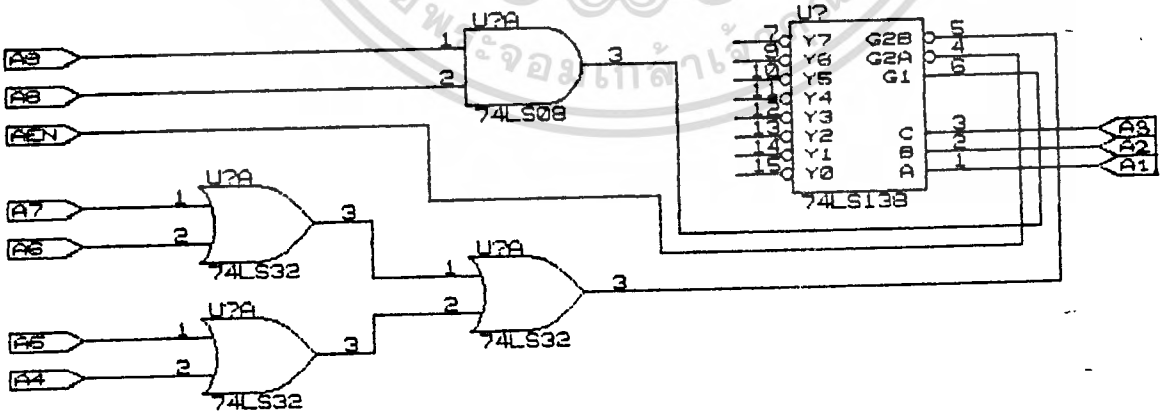
74LS138 สามารถ DECODE PORT เบอร์ดังต่อไปนี้ได้โดยใช้วงจรดังรูปที่ 4

$\overline{Y}_0$ = Ø3ØØH,Ø3Ø1H $\overline{Y}_1$ = Ø3Ø2H,Ø3Ø3H

$\overline{Y}_2$ = Ø3Ø4H,Ø3Ø5H $\overline{Y}_3$ = Ø3Ø6H,Ø3Ø7H

$\overline{Y}_4$ = Ø3Ø8H,Ø3Ø9H $\overline{Y}_5$ = Ø3ØAH,Ø3ØBH

$\overline{Y}_6$ = Ø3ØCH,Ø3ØDH $\overline{Y}_7$ = Ø3ØEH,Ø3ØFH



รูปที่ 4

4. C/$\overline{D}$ (CONTROL/DATA)

ใช้ A_0 จาก CPU ต่อโดยตรงกับขานี้เพื่อเป็นการเลือกว่าขณะนี้เป็นการให้ CPU ส่งรหัสควบคุมกับเลือก mode และ CPU รับรหัสแสดงสถานะ ถ้า A_0 เป็น 1 หรือในกรณีที่ทั้งรับทั้งส่งข้อมูลจะต้องให้ A_0 เป็น "Ø" โดยขานี้ของ 8251 ต้องใช้คู่กับขา $\overline{RD}$ และ $\overline{WR}$

5. $\overline{RD}/\overline{WR}$ (READ/WRITE)

สำหรับ 8088 จะมีขา IOR เป็น OUTPUT แอคทีฟที่ลอจิก ๐ เพื่อเป็นการอ่านข้อมูลจาก PORT อยู่แล้วและยังมีขา IOW เป็น OUTPUT แอคทีฟที่ลอจิก Ø เพื่อเป็นการเขียนข้อมูลลงบน PORT ดังนั้น ขา RD ของ 8251 จึงสามารถต่อกับ IOR ของ 8088ได้เลย และสำหรับ WR ของ 8251 ก็เช่นกันสามารถต่อกับขา IOW ของ 8088 ได้โดยตรง

ตารางข้างล่างต่อไปนี้จะแสดงการทำงานร่วมกันระหว่างขา C/D , RD , WR , และ CS ของ 8251

C/$\overline{D}$	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	ความหมาย
Ø	Ø	1	Ø	ข้อมูลจาก 8251 ถูกส่งไปยังบัสข้อมูล
Ø	1	Ø	Ø	ข้อมูลจากบัสข้อมูลไปยัง 8251
1	Ø	1	Ø	8251 อ่านรหัสแสดงสถานะจากบัสข้อมูล
x	x	x	1	บัสข้อมูล 8251 ลอยตัว

6.RESET

เนื่องจากการ RESET ของ 8251 จะต้อง RESET ตามการ RESET PROGRAM ที่เกี่ยวข้องกับ 8251 ของ CPU ดังนั้นการต่อขา RESET (ACTIVE ที่ 1) กับขา RESET DRIVER ของ CPU ซึ่งจะเป็นขาที่ ACTIVE ที่ 1 ในระยะแรกที่เริ่มเปิดเครื่อง และจะ ACTIVE ต่อไปจนกระทั่งระบบต่างๆของเครื่องพร้อมที่จะทำงานได้จึงจะตกเป็น Ø นั้นไม่พอเพียงกับการ RESET 8251 เพราะการ RESET 8251 คือการเคลียร์ BUFFER ข้อมูลใน 8251ให้ว่างก่อนเพื่อพร้อมที่จะดำเนินการส่งหรือรับข้อมูลต่อไปทุกครั้งที่มีการRUN PROGRAM เกี่ยวกับการส่งหรือรับข้อมูล ดังนั้นจึงต้องมีการทำ INTERNAL RESET 8251 ก่อนทุกครั้งด้วย SOFTWARE ซึ่งจะสั่ง INTRENAL RESET ได้โดยการกำหนดที่ COMMAND WORDในส่วนโปรแกรม INITIAL ของการดำเนินงานกับ 8251

7. $\overline{CTS}$ (CLEAR TO SEND)

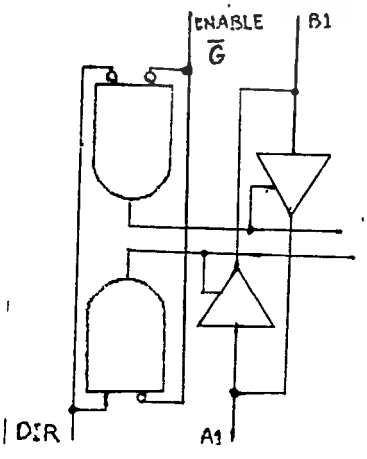
เป็นสัญญาณ INPUT จากภายนอก เมื่อ $\overline{CTS}$ เป็น "Ø" ตัว 8251 จึงจะสามารถ

ส่งข้อมูลออกไปได้ (ถ้าสัญญาณ TxEN (TRANSMITTER ENABLE)ในรหัสควบคุมเป็น "1" ด้วย) ดังนั้นทั้ง MASTER และ SLAVE เราจะเอาขานี้ของ 8251 ต่อไว้กับ GND

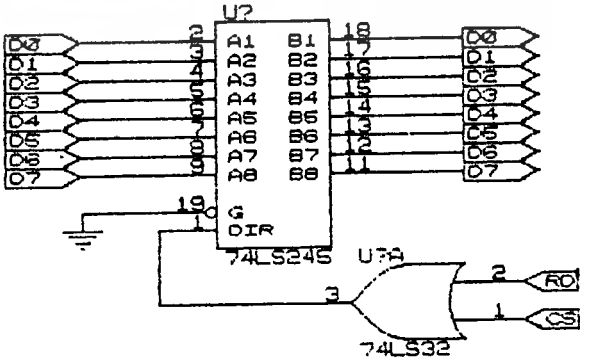
8. $\underline{D_0-D_7}$

เราควรต่อ BUFFER 2 ทาง (74LS245) ระหว่างขา D_0-D_7 ของ CPU (ทั้ง MASTER และ SLAVE) กับขา D_0-D_7 ของ 8251 เพื่อที่จะได้กำหนดทิศทางการไหลของข้อมูลระหว่าง CPU กับ 8251ได้ถูกต้องยิ่งขึ้น โดยที่เราสามารถกำหนดทิศทางการไหลของข้อมูลได้จากขา Enable Q (ขา 19) กับขา 1 ของ 74LS245 คือเราให้ขา Enable Q ต่อกับ GND และขา 1 ต่อไว้กับ OUTPUT ของ OR Gate ซึ่งเป็นการ OR กันระหว่าง RD และ CS ของ 8251 ถ้าขา 1 เป็น "Ø" (RD และ CS เป็น "Ø" เป็นการอ่าน Port) ข้อมูลจาก 8251 ก็จะไหลไปทาง CPU ได้ และถ้าขา 1 ของ 74LS138 เป็น "1" (เมื่อ CS เป็น "Ø" และ RD เป็น "1" เป็นการเขียน Port)ข้อมูลจาก CPU ก็จะไหลไปให้ 8251 ได้ ซึ่งวงจรเกทภายในของ 74LS138 จะเป็นไปตามรูปที่ 5 และทิศทางการไหลของข้อมูลจะเป็นตามตารางข้างล่างนี้ ส่วนลักษณะการต่อวงจรจะเป็นไปตามรูปที่ 6

DIR	DATA FLOW
1	CPU TO 8251
Ø	8251 TO CPU

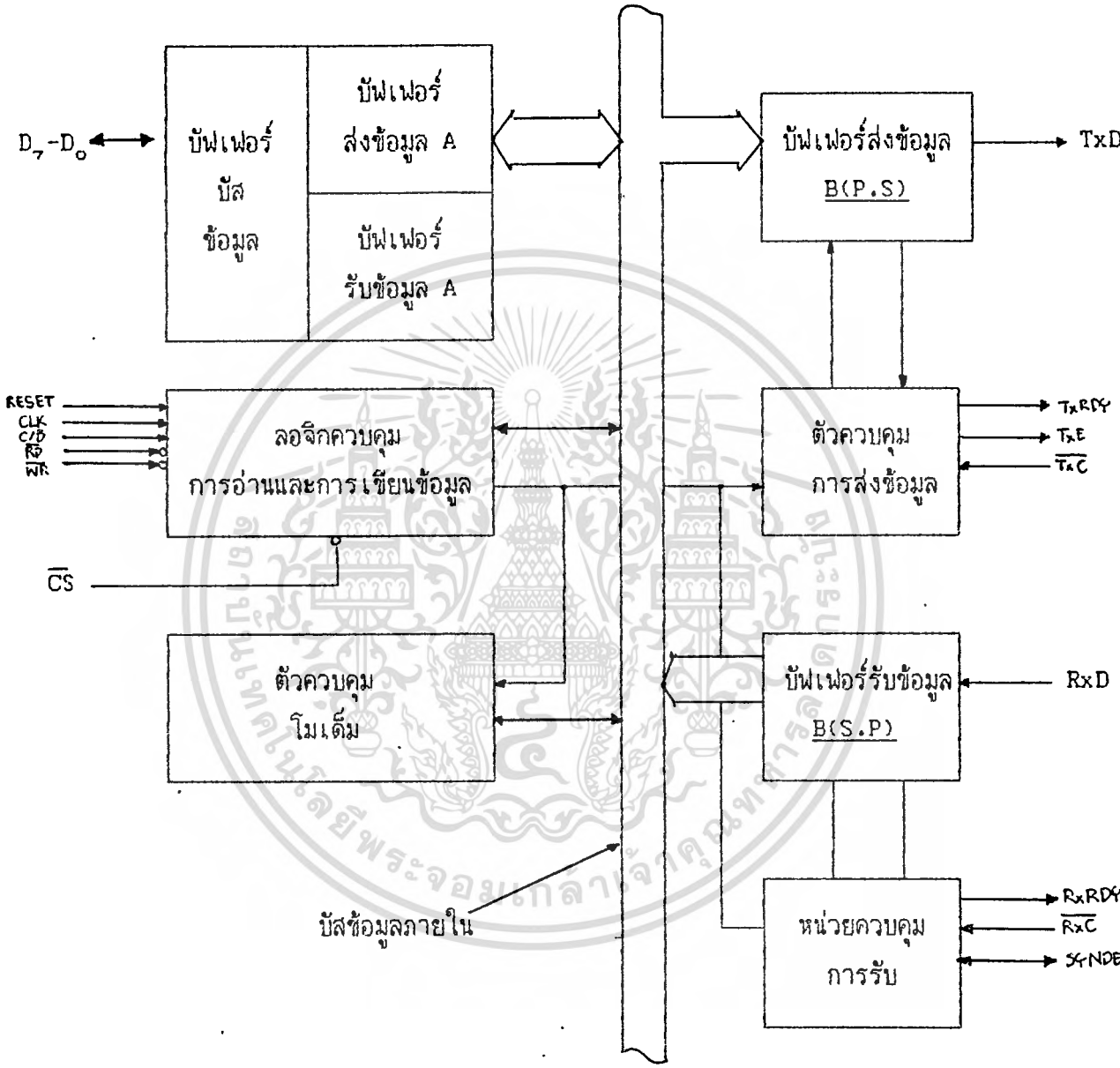


รูปที่ 5



รูปที่ 6

โครงสร้างภายในของ 8251



SOFTWARE ของ 8251

การเลือก mode และ command word

MODE WORD ของ 8251 แบบ ASYNCHRONOUS

D7	D6	D5	D4	D3	D2	D1	DØ
1	1	Ø	1	1	1	Ø	1

ทั้ง master และ slave = ØDDH

คำอธิบาย

-"D_0" และ "D_1" การเลือก 2 BIT นี้ จะเป็นไปเพื่อเลือกตัวคูณอัตราโบด์สำหรับการใช้งานในขณะนี้จะหมายถึงการเลือกอัตราโบด์คูณ 1 กับความถี่ที่ขา T_xC และที่ขา R_xC

-"D_2" และ "D_3" การเลือก 2 BIT นี้ จะเป็นการเลือกความยาวของข้อมูลสำหรับการใช้งานนี้ เลือกความยาวข้อมูลเท่ากับ 8 BIT

-"D_4" BIT นี้ถ้าเป็น 1 หมายถึงให้มีการตรวจสอบพาริตี้

-"D_5" ถ้า BITนี้เป็น Ø หมายถึงรูปแบบการเติม BIT ให้เป็นพาริตี้คี่

-"D_6" และ "D_7" หมายถึงการเลือกจำนวน STOP BIT ถ้า D_6 เท่ากับ 1 และ D_7 เท่ากับ 1 จำนวน STOP BIT เท่ากับ 2 BIT

COMMAND WORD

D7	D6	D5	D4	D3	D2	D1	DØ
Ø	Ø	Ø	1	Ø	1	Ø	1

ทั้ง master และ slave = 15H

คำอธิบาย

-"D_0" = T_xEN ถ้า BIT นี้เป็น 1 หมายถึงให้ ENABLE T_x

-"D_1" = DATA TERMINAL READY ถ้า BIT นี้เป็น 1 หมายถึงทำให้สัญญาณ DTR เป็น Ø

-"D_2" = R_xEN ถ้า BIT นี้เป็น 1 หมายถึงให้ ENBLE R_x

-"D_3" = จะเป็นการสั่งให้ส่งด้วยอักขระเบรก แต่ในที่นี้จะให้ BIT นี้เป็น Ø คือหมายถึงให้ทำงานตามปรกติ

-"D_4" = จะเป็นการ RESET ความผิดพลาด ถ้าเป็น 1 จะทำให้เกิดการ RESET PARITY ERROR , OVERRUN ERROR , FRAMING ERROR เองโดยอัตโนมัติ

-"D_5" =ถ้าเป็น 1 หมายถึงทำให้สัญญาณ RST เป็น Ø

-"D_6" = ถ้าเป็น 1 จะเป็นการสั่ง 8251 ทำการ INTERNAL RESET ซึ่งจะทำให้ตัว 8251 พร้อมที่จะรับคำสั่ง

-"D_7"= ถ้าเป็นการส่งแบบ ASYNCRONAUS จะต้องให้ BIT นี้เป็น Ø

ระหัสควบคุมเพิ่มเติมของ master เนื่องจากการ reset 8251 ที่ต่อกับ master จะเป็นการ RESET เพียงครั้งแรกตอนเปิดเครื่องเท่านั้น ดังนั้นการ RUN PROGRAM ที่เกี่ยวกับ 8251 ทุกครั้งเราควรจะทำการ internal reset ตัว 8251 ก่อน โดยการเลือก mode word และ command word ทุกครั้ง โดยให้บิตที่ 6 ในรหัสควบคุม เป็น 1 เพราะฉนั้นรหัสควบคุมสำหรับการ reset จึงเป็น 55H และควรสั่ง internal reset หรับการ reset 8251

STATUS WORD

เราจะพิจารณาเฉพาะ DØ D1เท่านั้นสำหรับในการทดลองนี้เพื่อตรวจสอบว่า 8251 พร้อมที่จะรับหรือส่งข้อมูลหรือไม่(สามารถตรวจสอบได้ทาง HARDWARE)

D7	D6	D5	D4	D3	D2	D1	DØ
DSR	DE1	FE	OE	PE	TxEMP	RxRDY	TxRDY

เมื่อต้องการที่จะให้ 8251 ในการติดต่อรับ-ส่งข้อมูลเราจะมีขั้นตอนดังต่อไปนี้

1. ส่งรหัสเลือก MODE (=7DH) โดยส่งไปทาง PORT ที่ถอดรหัสทาง HARDWARE ไว้ (=3Ø1H)

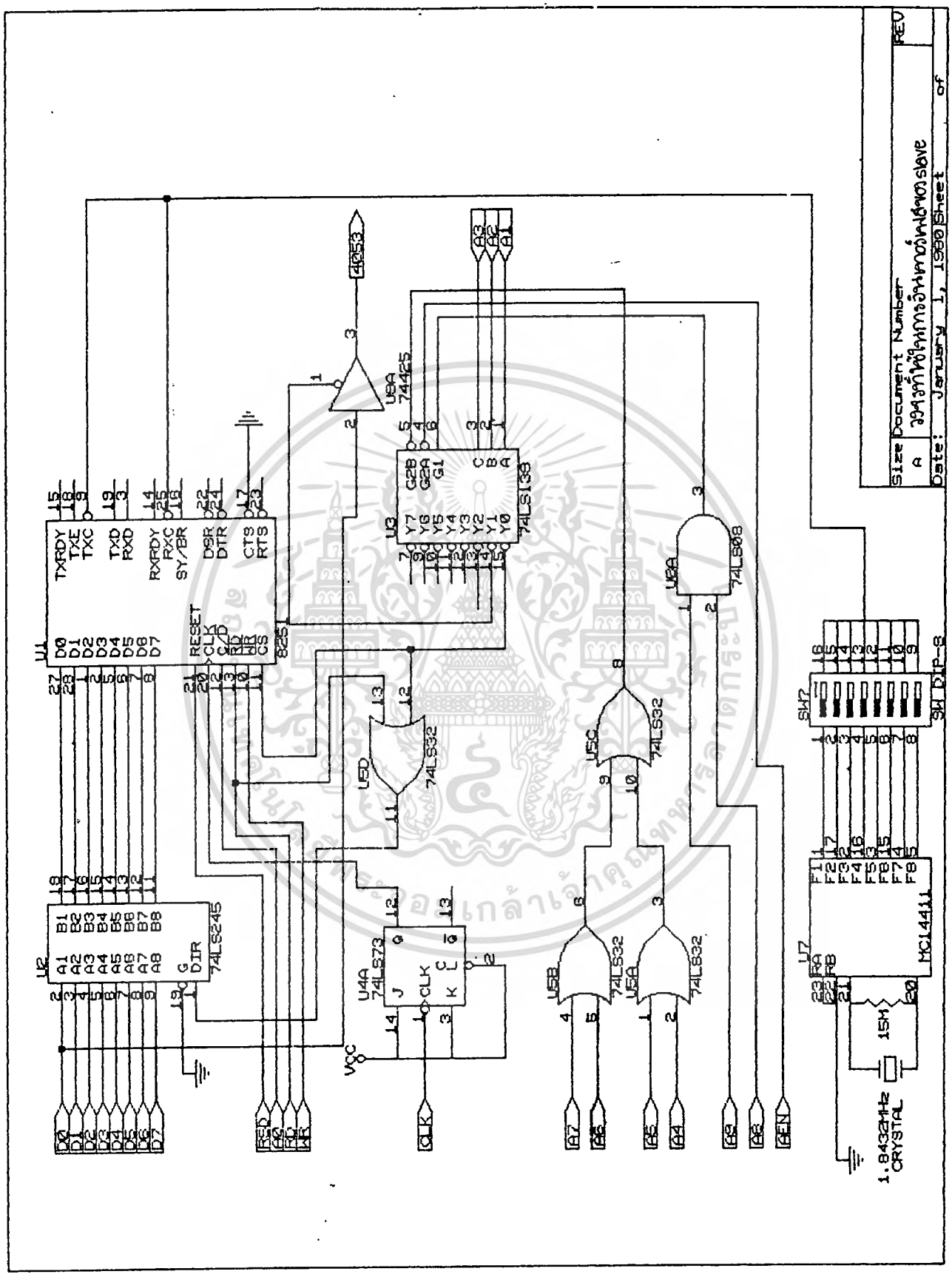
2. ส่ง COMMAND WORD โดยทำการ INTERNAL RESET ก่อน (=55H) ส่งไปทาง PORT 3Ø1H

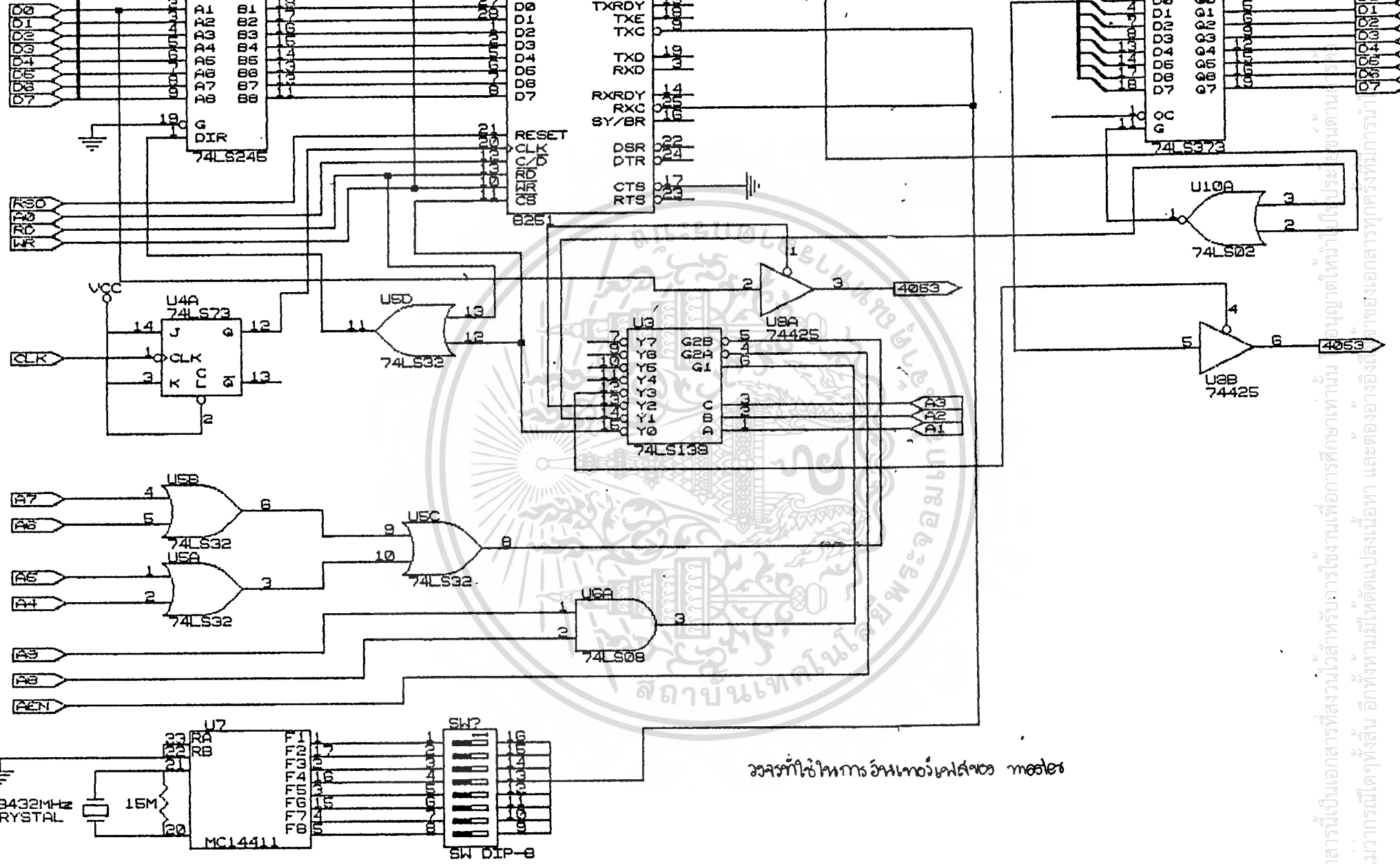
3. ส่งรหัสเลือก MODE อีกทีหนึ่ง

4. ส่ง COMMAND WORD ที่ใช้จริง (=15H)

5. อ่านรหัสแสดงสถานะ STATUS WORD ทาง PORT เบอร์เดิม

6. ตรวจดูว่า 8251 พร้อมจะรับข้อมูลหรือยัง (BIT D_1 ใน STATUS WORD = 1 หรือไม่) หรือพร้อมที่จะส่งข้อมูลหรือยัง (BIT D_0 = 1 หรือไม่) ถ้าพร้อมก็จะรับ-ส่งข้อมูลกันทาง PORT เบอร์ 3ØØH ถ้ายังไม่พร้อมก็ให้ทำในขั้นตอนนี้ต่อไปเรื่อยๆ





วงจรที่ใช้ในการอินเตอร์เฟสของ master

บทที่ 4
การอินเทอร์รัพต์

สำหรับโครงงานในภาคเรียนนี้ เราต้องการให้ MASTER เป็นตัวเลือกเครื่อง SLAVE โดยการส่งรหัสมายังเครื่อง SLAVE จะมีเครื่อง SLAVE เพียงเครื่องเดียวเท่านั้นที่จะสามารถถอดรหัสได้ และสัญญาณที่ได้จากการถอดรหัสนี้ เราจะนำไปทริกขา IRQ2 8259 เพื่อจัด INTERRUPT CPU ของเครื่อง SLVAE ตัวนั้นต่อไป

ขั้นตอนของการ INTERRUPT 8259

1. มีสัญญาณขอ INTERRUPT เข้ามาทางขา IRQ2
2. CPU จะเก็บ FLAG และ RETURN ADDRESS ลงบน STACK และทำให้ DATA BUS อยู่ใน INPUT MODE
3. CPU ส่งสัญญาณ ACKNOWLEDGE โดยส่งมาเป็นพัลซ์ 2 ลูก
   -ลูกแรกจะทำให้ 8259 ทำตาม MODE ที่สั่งไว้
   -ลูกที่สอง 8259 ส่ง INTERRUPT VECTOR ไปให้ CPU
4. เมื่อ CPU ได้รับ INTERRUPT VECTOR ก็จะทำการหาตำแหน่ง ADDRESS เริ่มต้นของโปรแกรมทำงาน

จาก INTERRUPT VECTOR ที่กล่าวมาแล้วจะขึ้นอยู่กับ CHANNEL ของการ INTERRUPT และการ INITIAL 8259 เอง

สำหรับการจัดลำดับความสำคัญของการ INTERRUPT ขึ้นอยู่กับการเลือก MODE ตอนสั่ง INITIAL 8259 ซึ่งเราไม่จำเป็นจะต้องสั่งเองถ้าใช้เครื่องคอมพิวเตอร์ CPU 8088

การ โปรแกรม 8259

ภายใน 8259 จะมีรีจิสเตอร์ ที่ใช้ในการเก็บคำสั่งต่างๆ ที่เรียกว่า COMMAND WORD อยู่ 2 ชนิด คือ

1. INITIAL COMMAND WORD (ICW) ซึ่งจะมีอยู่ 4 ชนิดคือ

1.1 ICW1

A_0	D_7	D6	D5	D_4	D3	D2	D_1	D_0
0	A_7	A_6	A_5	1	LTIM	ADI	SNGL	IC4

-D_0 เป็นการเลือกว่าจะต้องการสั่ง ICW4 หรือไม่ ถ้าต้องการ BIT นี้จะเป็น 1

-D_1 เป็นการเลือก MODE ว่าจะใช้ 8259 เพียงตัวเดียว หรืออยู่ใน CASCADE MODE ถ้า BIT นี้เป็น 1 จะใช้ 8259 เพียงตัวเดียว

-D_2 BIT นี้จะถูกใช้งานเฉพาะในกรณีที่ทำงานร่วมกับ CPU 8080 เพราะฉนั้น BIT นี้จึงถูก RESET เป็น 0

-D_3 เป็นการเลือกลักษณะสัญญาณที่จะมา TRIG IRQ2 ว่าจะเป็นการ LEVEL TRIGGER หรือ EDGE TRIGGER ถ้า BIT นี้เป็น 1 จะเป็นการเลือกแบบ LEVEL TRIGGER แต่สำหรับการ TRIG ที่ใช้ในโครงงานนี้จะเป็นแบบ EDGE TRIGGER

-D_5 D_6 D_7 BIT ทั้ง 3 นี้จะถูกใช้เฉพาะกับ CPU 8080▣8085 เท่านั้น ดังนั้นจึง RESET ข้อมูลใน BIT ทั้ง 3 นี้เป็น 0 ทั้งหมด

1.2 ICW2

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	1	0	0	0

-D_0-D_2 ข้อมูลในทั้ง 3 BIT นี้จะถูกใช้งานในแต่เฉพาะ CPU 8080▣8085 เท่านั้นดังนั้นจึง ให้ข้อมูลทั้ง 3 ตัวนี้เป็น 0 ทั้งหมด

-D_3-D_7 สำหรับ CPU 8088 จะถูกสั่งให้เป็นไปตามตารางข้างบนอยู่แล้ว ข้อมูลพวกนี้จะถูกใช้เพื่อประกอบการเปิดตาราง VECTOR ของ CPU

1.3 ICW3 สำหรับ COMMAND WORD ตัวนี้จะใช้กับระบบที่มี 8259 มากกว่า 1 ตัว ดังนั้นในระบบนี้จึงไม่ต้องนำมาพิจารณา

1.4 ICW4

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	SFNM	BUF	M/S	AEOI	MPM

-D_0 สำหรับ CPU 8088 BIT นี้จะถูก SET ให้เป็น 1

-D_1 ถ้าเป็น 1 จะอยู่ใน MODE AUTO END OF INTERRUPT แต่ถ้าเป็น 0 จะอยู่ใน MODE NORMAL END OF INTERRUPT

-D_2-D_3 จะใช้ 2 BIT นี้ทำงานร่วมกันโดย D_2 จะใช้ใน MODE

CASCADE เพื่อเลือกว่าจะ MASTER หรือ SLAVE (ไม่เกี่ยวข้องกับโครงงานนี้) ถ้า BIT D_2 นี้เป็น 1 แสดงว่า 8259 ตัวนั้นเป็น MASTER สำหรับ D_3 ถ้าเป็น 1 8259 จะถูกสั่งให้ทำงานใน MODE BUFFER

-D_4 ถ้าข้อมูลใน BIT นี้เป็น 1 จะเป็นการให้ 8259 เข้าสู่ MODE FULLY NESTED ซึ่งเป็นระบบขนาดใหญ่ที่มี 8259 ทำงานร่วมกันหลายตัวในลักษณะ CASCADE

2.OPERATION COMMAND WORD (OCW) มีอยู่ 3 ชนิดคือ

2.1 OCW1

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	M7	M6	M5	M4	M3	M2	M1	MØ

– D_0-D_7 ถ้า BIT ไหนเป็น 1 จะเป็นการ MASK (การ DISABLE) สำหรับการขอ INTERRUPT ใน CHANNEL นั้นและสำหรับในโครงงานนี้จะเป็น COMMAND WORD เพียงตัวเดียวที่ต้อง SET ค่าให้มันใหม่โดย OCW1=FBH ซึ่งหมายความว่าเราให้ BIT D_2 เป็นศูนย์แสดงว่าการขอ INTERRUPT ทาง CHANNEL นี้สามารถกระทำได้

2.2 OCW2

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Ø	R	SL	EOI	Ø	Ø	L_2	L_1	L_0

-D_0-D_2 ข้อมูลใน BIT ทั้งสามนี้จะใช้สำหรับกำหนดว่าคำสั่งที่ส่งให้กับ OCW2 นี้จะมีผลต่อ CHANNEL ซึ่งจะเป็นรหัสเลขฐานสองของ CHANNEL ที่ต้องการ จะอย่างไรก็ตาม BIT ทั้งสามนี้จะถูกใช้งานหรือไม่จะขึ้นอยู่กับข้อมูลใน BIT D_6

-D_5-D_7 BIT ทั้งสามนี้จะใช้งานร่วมกันเพื่อกำหนดโหมดการทำงานต่างๆ ของ 8259 โดย BIT D_5 จะใช้กำหนดว่าคำสั่งที่ส่งให้กับ 8259 นี้อยู่ในโหมด END OF INTERRUPT (EOI) หรือไม่ BIT D_6 จะใช้กำหนดว่าคำสั่งนี้ CHANNEL ใดของการ INTERRUPT (สั่งจาก D_0-D_2) จะได้รับผลจากคำสั่งนี้หรือไม่ BIT D_7 จะถูกใช้ในการกำหนดว่าคำสั่งนี้เป็นคำสั่งในโหมด ROTATING PRIORITY หรือไม่

2.3 OCW3

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Ø	Ø	ESMM	SMM	Ø	1	P	RR	RIS

-D_0-D_1 ข้อมูลใน BIT ทั้ง 2 นี้จะถูกใช้ในการอ่านสถานะของ 8259 โดยข้อมูลใน BIT D_0 (RIS)จะใช้สำหรับเลือก REGIRTER สถานะของ 8259 คือ IR (INTERRUPT REQUEST REGISTER เป็น REGISTER ที่แสดงว่ามีการขอ INTETRRUP มาทาง CHANNEL ใดบ้าง) เมื่อ BIT นี้เป็น 1 และ IS (IN-SERVICE REGISTER เป็น REGISTER ขนาด 8 BIT ที่แสดงว่า CHANNEL ใดบ้างที่กำลังได้รับการตอบสนองอยู่) เมื่อ BIT นี้เป็น Ø ซึ่ง 8259 จะส่งสถานะนี้ไปยัง DATA BUS เมื่อพัลส์ของสัญญาณ READ ถูกส่งให้ 8259 แต่ทั้งนี้ต้องขึ้นอยู่กับขา RR (D_1) ด้วยคือ RR ต้องเป็น 1 จึงจะมีการส่งสถานะออกไปบน DATA BUS ได้

-D_2 ถ้าข้อมูลใน BIT นี้ถูก SET เป็น 1 8259 จะทำงานใน MODE POLL

-D_5 ถ้า BIT นี้เป็น 1 8259 จะทำงานใน MODE SPECIAL MASK แต่อย่างไรก็ตามผลของข้อมูลใน BIT นี้ ยังต้องขึ้นกับข้อมูลใน BIT D_6 ด้วย

-D_6 ถ้าต้องการให้ 8259 ทำงานใน MODE MASK ต้องให้ข้อมูลใน BIT นี้เป็น 1

สำหรับ ADDRESS ที่จะถูกอ้างเพื่อใช้สำหรับการส่ง COMMAND WORD ไปให้ 8259 จะเป็นไปตามตารางข้างล่างนี้

ADDRESS	COMMAND
0020H	ICW1
0021H	ICW2
0021H	ICW3
0021H	ICW4
0021H	OCW1
0020H	OCW2
0020H	OCW3

สำหรับการโปรแกรมตัว 8259 ใน IBM/PC นั้นนอกจาก OCW1 แล้วตัว PC ทำการ SET 8259 เองอยู่แล้วเพราะฉนั้นไม่สมควรที่จะ SET ใหม่เพราะอาจทำให้ระบบทำงานผิดพลาดได้

บทที่ 5

วิธีการและรูปแบบในการติดต่อรับส่งข้อมูล

ในการส่งข้อมูลระหว่าง PERSONAL COMPUTER สิ่งที่หลีกเลี่ยงไม่ได้คือความผิดพลาดต่างๆ ที่อาจจะเกิดขึ้นได้ในระหว่างการติดต่อ ดังนั้นจึงได้มีการจัดรูปแบบการรับส่งข้อมูลเพื่อให้ได้ข้อมูลที่ถูกต้องและใช้เวลารวดเร็วในการรับส่ง ซึ่งจะมีการตรวจสอบความถูกต้องของข้อมูลโดยมีการส่งข้อมูลเป็นบล๊อคๆ ให้มีการเช็คผลบวกของข้อมูลในแต่ละบล๊อค รวมทั้งมีการใช้โปรโตคอลแบบ XMODEM ในการติดต่อดังจะมีรายละเอียดดังนี้

5.1 **การส่งผ่านข้อมูลทีละบล็อค** ในขั้นตอนการทำงานของการรับส่งข้อมูลเมื่อมีการโหลดไฟล์ที่ต้องการส่งออกไปเราจะมีการจัดการเกี่ยวกับข้อมูลในไฟล์โดยจัดแบ่งเป็นบล๊อคขนาด 256 ไบต์ ทั้งนี้เพื่อให้ง่ายต่อการส่งเนื่องจากในแต่ละบล๊อคจะมีข้อมูลขนาดไม่มากนักซึ่งเมื่อมีข้อมูลที่ผิดพลาดในการส่งของบล๊อคที่ผ่านมาก็จะง่ายและไม่เสียเวลามากในการส่งข้อมูลบล็อคเดิมกลับไป และข้อผิดพลาดที่อาจจะเกิดขึ้นสามารถเช็คได้จากผลบวกของข้อมูลในบล๊อคนั้นๆ และการควบคุมการส่งข้อมูลในแต่ละบล๊อคจะมีรูปแบบที่เรียกว่าโปรโตคอลดังจะได้กล่าวต่อไป

5.2 **การเช็คผลบวกของข้อมูล** ในแต่ละบล๊อคของข้อมูลเราจะทำการบวกข้อมูลในแต่ละไบต์เข้าด้วยกันทั้งนี้จนครบ 256 ไบต์ และจะเอาผลรวมที่ได้เฉพาะไบต์ต่ำไบต์เดียวเท่านั้น เพื่อส่งออกไปให้ตัวรับ และในทำนองเดียวกันเมื่อตัวรับได้รับข้อมูลจนครบ 256 ไบต์ แล้วก็จะทำการบวกข้อมูลทั้งหมดนั้นเข้าด้วยกันและจะเอาผลรวมที่ได้เฉพาะไบต์ต่ำนั้นมาทำการเปรียบเทียบกับผลรวมที่ตัวส่งได้ส่งไปให้ว่าเท่ากันหรือไม่ ถ้าเท่ากันก็จะรับข้อมูลบล๊อคต่อไปแต่ถ้าไม่เท่ากันก็จะให้ตัวส่งส่งข้อมูลบล๊อคเดิมกลับมาใหม่ โดยวิธีการนี้จะทำให้ข้อมูลมีโอกาสผิดพลาดน้อยมาก และเป็นวิธีการที่นิยมใช้กันมาก โดยจะใช้โปรโตคอลแบบ XMODEM ควบคุมการติดต่อ

5.3 **โปรโตคอลแบบ XMODEM** เป็นเทคนิคในการควบคุมการรับส่งข้อมูล และจะต้องมีการกำหนดรูปแบบให้เหมือนกันทั้งฝ่ายรับและฝ่ายส่ง โดยการใช้อักขระควบคุมในตารางของแอสกี้ เพื่อควบคุมและการกำหนดการรับส่งบล๊อคของข้อมูล อักขระที่ใช้ในการควบคุมนี้ได้แก่

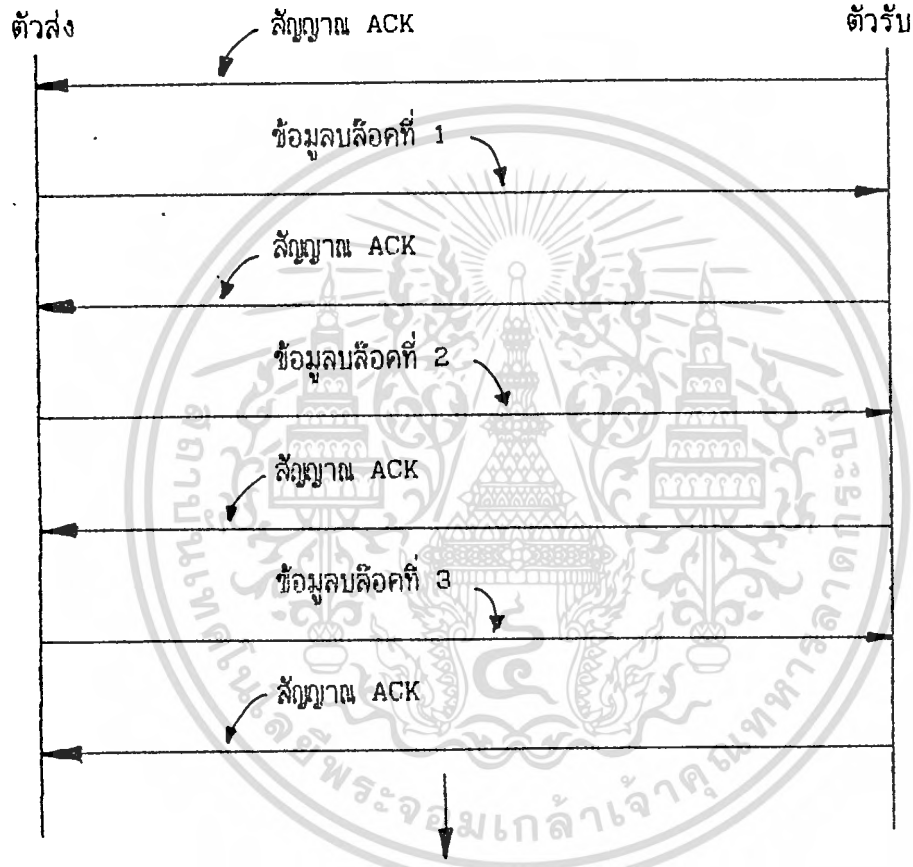
ACK (ACKNOWLEDGE) เป็นรหัสที่ใช้บอกตัวส่งว่าพร้อมแล้วต้องการจะติดต่อด้วยและเป็นสัญญาณที่ใช้บอกตัวส่งว่าข้อมูลถูกต้องให้ส่งข้อมูลบล๊อคต่อไปมาได้ ซึ่งมีค่าเท่ากับ 06 ในรหัสแอสกี้

NAK (NEGATIVE ACKNOWLEDGE) เป็นรหัสที่ใช้บอกฝ่ายส่งว่าข้อมูลในบล๊อคที่ผ่านมาผิดพลาดให้ส่งข้อมูลบล๊อคเดิมมาอีกครั้ง ซึ่งมีค่าเท่ากับ 21 ในรหัสแอสกี้

ETB (END OF TRANSMISSION BLOCK) ใช้ระบุจุดจบของข้อมูลในบล๊อคนั้น ซึ่งมีค่าเท่ากับ 23 ในรหัสแอสกี้

EOT (END OF TRANSMISSION) เป็นรหัสที่ใช้ระบุว่าสิ้นสุดการส่งผ่านข้อมูล หรือข้อมูลได้รับเรียบร้อยแล้วให้ยกเลิกการติดต่อ ซึ่งมีค่าเท่ากับ 04 ในรหัสแอสกี้

ลักษณะการติดต่อแบบใช้โปรโตคอล แสดงได้ดังรูปที่ 5.1

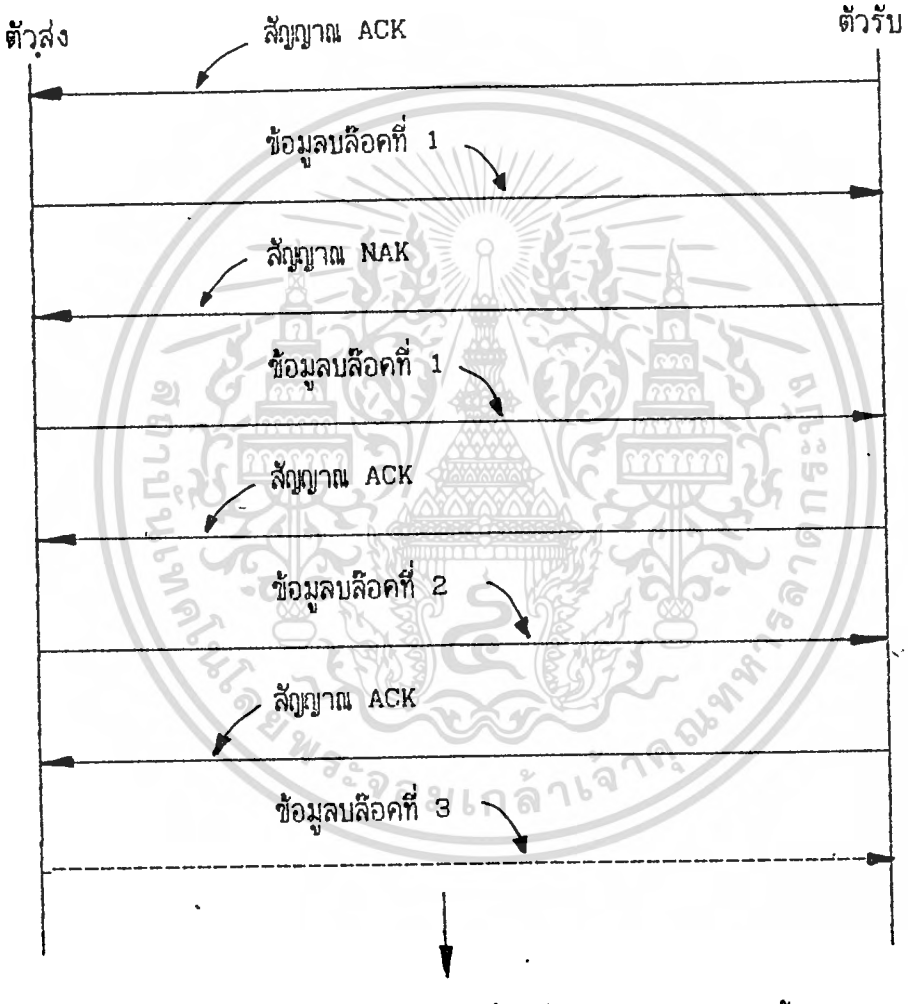


รูปที่ 5.1 แสดงการส่งผ่านข้อมูลทีละบล๊อคของการส่งสัญญาณกึ่งทางคู่โดยสมมติว่าการส่งผ่านข้อมูลนี้ไม่มีข้อผิดพลาดเกิดขึ้น

โดยที่ทางตัวส่งจะไม่ส่งข้อมูลไปให้จนกว่าฝ่ายรับจะทำการส่งสัญญาณ ACK มาให้ฝ่ายส่ง เมื่อฝ่ายส่งได้รับสัญญาณนี้แล้วจะทำการส่งข้อมูลบล๊อคแรกไปให้ ซึ่งจะเป็นข้อมูลขนาด 256 ไบต์ และจะส่งสัญญาณ ETB ปิดท้ายไปเพื่อทำการบอกฝ่ายรับว่าสิ้นสุดข้อมูลบล๊อคที่หนึ่งแล้ว จากนั้นฝ่ายส่งจะทำการส่งผลบวกของข้อมูลทั้ง 256 ไบต์นั้น ไปให้ฝ่ายรับ และฝ่ายรับเมื่อได้รับผลรวมของข้อมูลแล้วก็จะทำการเช็คกับผลรวมของข้อมูลที่ทางฝ่ายรับได้รวมเอาไว้ว่าตรงกันหรือไม่ ถ้าตรงกันแสดงว่าข้อมูลที่ได้รับมานั้นถูกต้อง ทางฝ่ายรับก็จะส่งสัญญาณ ACK ตอบกลับไปให้ฝ่ายส่งเพื่อให้ส่งข้อมูลบล๊อคที่ 2 มายังฝ่ายรับต่อไปและ

จะทำอย่างนี้ไปเรื่อยๆ จนกว่าทางฝ่ายรับจะเจอ EOF (END OF FILE) ซึ่งเป็นการบอกให้ฝ่ายรับรู้ว่ามันเป็นไบต์สุดท้ายของข้อมูล เมื่อฝ่ายรับทำการเช็คผลบวกแล้วว่าถูกต้องก็จะทำการส่งสัญญาณ EOT ตอบกลับไปเพื่อยกเลิกการติดต่อ

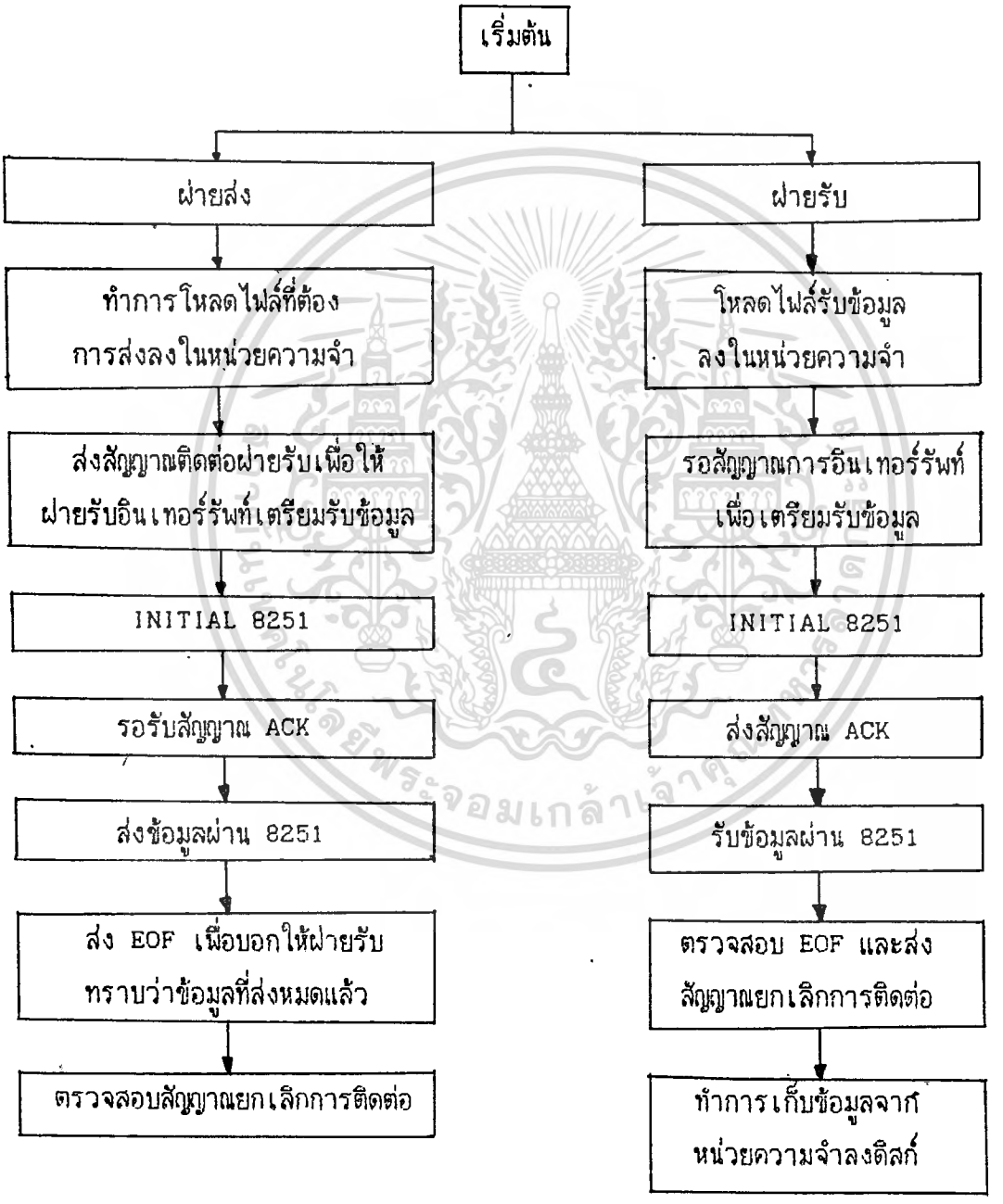
สำหรับในกรณีที่ข้อมูลที่ส่งมาเมื่อผ่านการเช็คผลรวมของข้อมูลแล้วไม่ตรงกัน ทางฝ่ายรับก็จะส่งสัญญาณ NAK ตอบกลับไปดังแสดงในรูปที่ 5.2



รูปที่ 5.2 แสดงการส่งข้อมูลซ้ำเมื่อมีการผิดพลาดเกิดขึ้น

ลักษณะการทำงานของโปรแกรมรับและส่งข้อมูล

สำหรับการโปรแกรมการทำงานของระบบการรับส่งข้อมูลจะแสดงได้เป็นโฟล์วชาร์ต ดังรูปที่ 5.3



รูปที่ 5.3 แสดงโฟล์วชาร์ตของระบบการรับส่งข้อมูล

สำหรับโปรแกรมการทำงานของตัวส่งหรือ MASTER แสดงได้ดังโฟล์วชาร์ตในรูปที่ 5.4 ซึ่งมีขั้นตอนการทำงานดังนี้

1. จะทำการโหลดไฟล์ที่ต้องการส่งไปให้ SLAVE เก็บไว้ในหน่วยความจำของเครื่องในที่นี้จะทำการจองเนื้อที่ในหน่วยความจำเอาไว้ 20 กิโลไบต์ ดังนั้นไฟล์ที่จะทำการส่งควรจะมีขนาดไม่เกิน 20 กิโลไบต์ แต่ทั้งนี้ถ้าต้องการส่งข้อมูลที่มีขนาดมากกว่า 20 กิโลไบต์ เราสามารถจองเนื้อที่ในหน่วยความจำให้มากกว่านี้ได้ แต่อาจจะมีปัญหาเนื่องจากจะทำให้โปรแกรมของเราใหญ่เกินไปเสียเวลาในการโหลดนาน หรืออาจจะใช้วิธีที่ 2 คือพัฒนาโปรแกรมใหม่ให้สามารถโหลดไฟล์ข้อมูลไม่ต้องมากนักคือประมาณ 1 กิโลไบต์แต่โหลดบ่อยครั้งกว่าเดิม ทั้งนี้จะทำให้ทางฝ่ายรับเมื่อรับข้อมูลแล้วก็จะทำการเก็บข้อมูลจากหน่วยความจำลงดิสก์ ซึ่งเป็นขั้นตอนที่ช้าทำให้ทางฝ่ายส่งต้องรอส่งข้อมูลบล๊อคต่อไปนาน สาเหตุที่เราให้ฝ่ายรับทำการโหลดไฟล์ข้อมูลก่อนเนื่องจากในขั้นตอนนี้ต้องใช้เวลามาก ดังนั้นถ้าฝ่ายรับทำงานเร็วกว่าคืออาจจะส่งสัญญาณ ACK มาในขณะที่ฝ่ายส่งยังทำการโหลดข้อมูลอยู่อาจจะทำให้สัญญาณ ACK สูญหายได้

2. ส่งสัญญาณการอินเทอร์รัพท์ไปให้ฝ่ายรับ เพื่อให้มีการรันโปรแกรมเพื่อทำการรับข้อมูลที่จะส่งไปให้

3. ทำการ INITIAL 8251 ให้พร้อมที่จะทำงาน

4. รอรับสัญญาณ ACK

5. ขั้นตอนการส่งข้อมูล เมื่อรับสัญญาณ ACK ได้แล้ว จะทำการส่งข้อมูลออกไปทีละไบต์ ในขณะเดียวกันก็จะทำการบวกค่ารหัสแอสกี้ของตัวอักษรนั้น รวมทั้งทำการเช็คหาไบต์สุดท้าย (1AH) ในกรณีที่บล๊อคของข้อมูลนี้น้อยกว่า 256 ไบต์ และทำการส่ง EOF ออกไปเพื่อให้ฝ่ายรับรู้ว่าเป็นข้อมูลตัวสุดท้าย หรือเช็คจำนวนข้อมูลจำนวน 256 ไบต์ ในกรณีที่ไฟล์ข้อมูลมีขนาดใหญ่ และยังไม่พบ EOF (1AH) จากนั้นส่ง ETB ออกไปเพื่อให้ฝ่ายรับรู้ว่าเป็นการสิ้นสุดบล๊อคข้อมูลนั้นๆ

สำหรับในกรณีที่ไม่สามารถรับสัญญาณ ACK ได้ จะทำการยกเลิกการติดต่อ ทั้งนี้อาจจะพัฒนาโปรแกรมให้ตรวจเช็คสัญญาณนี้ หลายๆ รอบก่อนก็ได้โดยทำการส่งสัญญาณขอให้ตอบไปยังฝ่ายรับ ให้ทำการส่งสัญญาณเดิมกลับมาใหม่ แต่ถ้ายังไม่สามารถรับสัญญาณ ACK ได้แสดงว่าตัวกลางของการสื่อสารนี้เลวร้ายมากก็ควรยกเลิกการติดต่อ

6. ทำการส่งผลรวมของข้อมูล (SUM OF DATA) เพื่อให้ฝ่ายรับทำการเช็คดูว่าข้อมูลที่ส่งไปกับข้อมูลที่รับได้ตรงกันหรือไม่

7. รอรับสัญญาณการตอบกลับมาของฝ่ายรับ ซึ่งสัญญาณที่ตอบกลับมาอาจจะได้แก่สัญญาณ ACK สัญญาณ EOT และสัญญาณ NAK

โดย ถ้าเป็นสัญญาณ ACK แสดงว่าข้อมูลที่ส่งไปถูกต้องให้ส่งข้อมูลบล๊อคต่อไป ไปได้

ถ้าเป็นสัญญาณ NAK แสดงว่าข้อมูลที่ส่งไปผิดพลาดให้ส่งข้อมูลบล๊อคเดิมกลับไป

ถ้าเป็นสัญญาณ EOT แสดงว่าข้อมูลที่ส่งไปซึ่งเป็นข้อมูลบล๊อคสุดท้ายถูกต้องให้ยกเลิกการติดต่อ

8. ในกรณีที่ข้อมูลที่ส่งไปยังไม่หมดก็จะกลับไปทำตามขั้นตอนการส่งข้อมูลดังที่กล่าวมาแล้วตามลำดับจนกว่าจะเจอ EOF

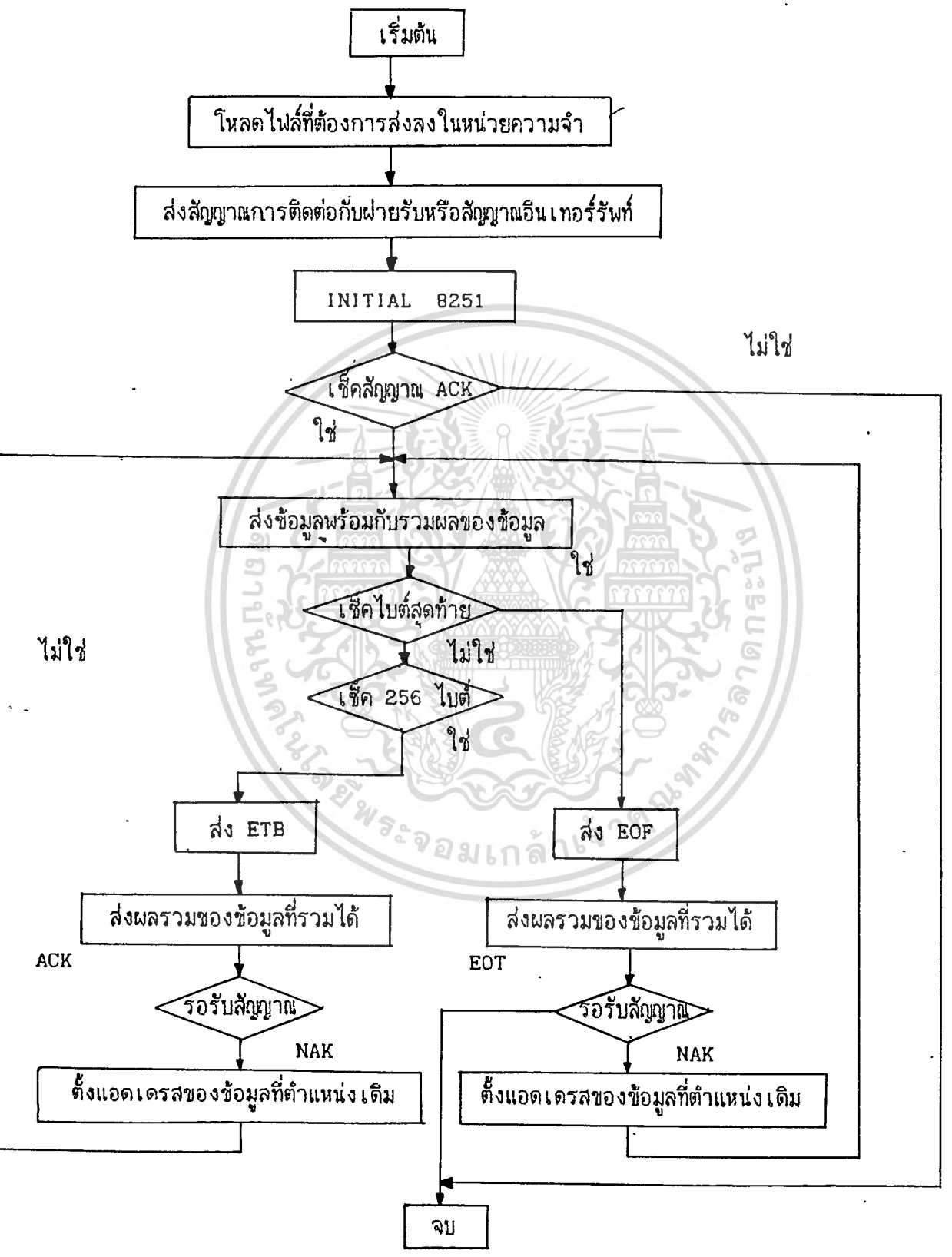
สำหรับโปรแกรมการทำงานของตัวรับหรือ SLAVE แสดงได้ดังโฟล์วชาร์ตในรูปที่ 5.5 ซึ่งมีขั้นตอนการทำงานดังนี้

1. จะรอรับสัญญาณการเรียกจากตัวส่ง หรือสัญญาณการอินเทอร์รัพท์
2. ทำการ INITIAL 8251 เพื่อให้พร้อมที่จะทำงาน
3. ส่งสัญญาณ ACK
4. ขั้นตอนการรอรับข้อมูล เมื่อมีข้อมูลเข้ามาทางฝ่ายรับจะทำการเช็คว่าเป็น EOF หรือ ETB หรือไม่ ถ้าไม่ใช่ทั้ง 2 อย่างก็จะทำการบวกค่าของรหัสแอสกี้ของข้อมูลนั้นๆ ไปเรื่อยๆ จนกว่าจะเจอ EOF ซึ่งแสดงว่าเป็นข้อมูลไบต์สุดท้าย หรือเจอ ETB ซึ่งแสดงว่าเป็นการสิ้นสุดของข้อมูลบล๊อคนั้นแล้ว
5. รับผลรวมของข้อมูลที่ส่งมาจากตัวส่ง
6. ทำการเช็คผลรวมที่ได้ในขั้นตอนที่ 4 และที่ 5 ว่าตรงกันหรือไม่

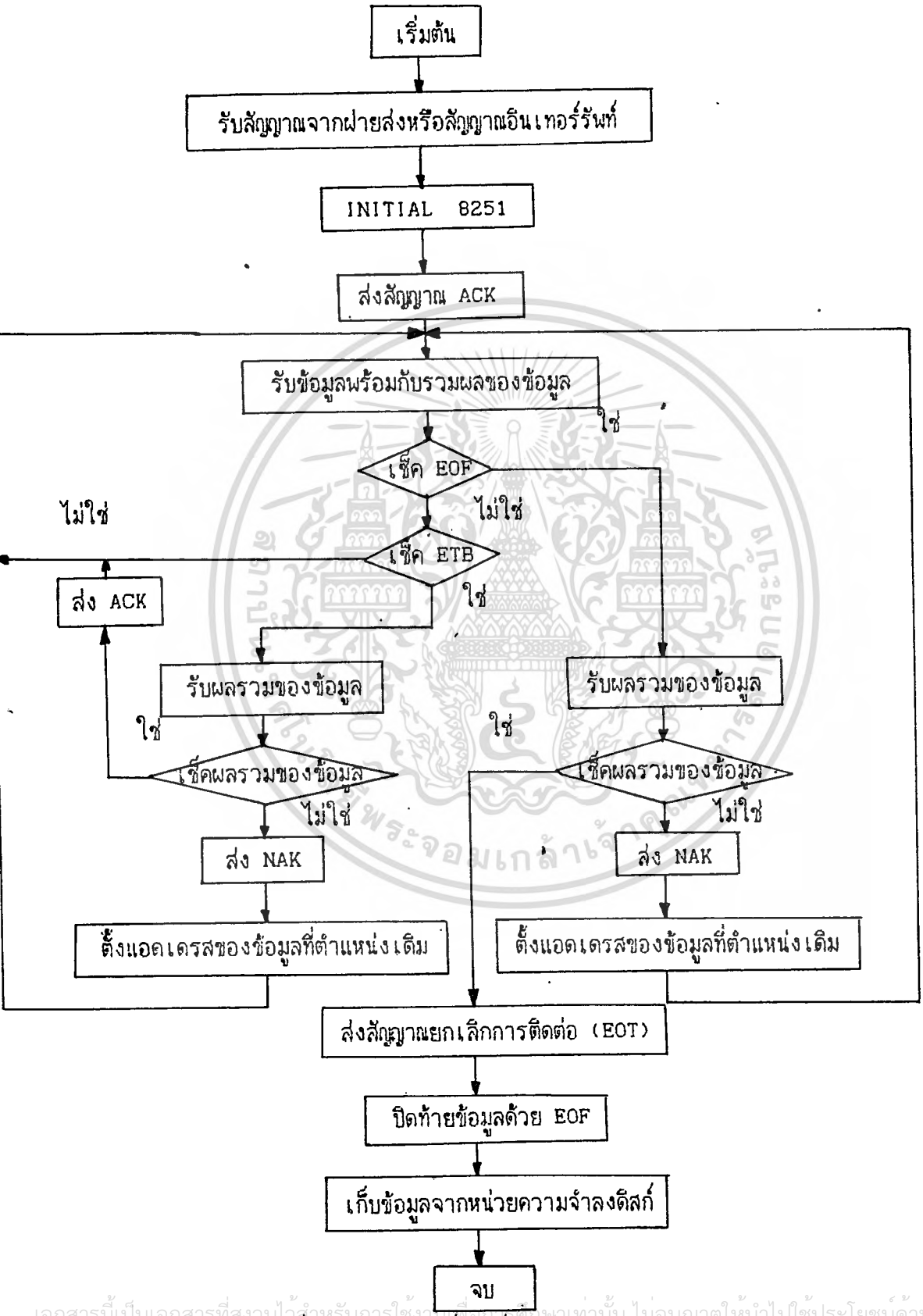
- ถ้าตรงกันจะทำการส่งสัญญาณ ACK กลับไปและรอรับข้อมูลบล๊อคใหม่
- ถ้าไม่ตรงกันจะทำการส่งสัญญาณ NAK กลับไปและรอรับข้อมูลบล๊อคเดิม
- ในกรณีที่เจอ EOF และตรวจสอบผลรวมของข้อมูลแล้วถูกต้องก็จะทำการส่งสัญญาณ EOT ออกไปให้ตัวส่งเพื่อเป็นการบอกว่าข้อมูลได้รับเรียบร้อยแล้วให้ทำการยกเลิกการติดต่อ

7. ในกรณีที่ยังไม่เจอ EOF จะทำขั้นตอนการรับข้อมูลตามลำดับจนกว่าจะเจอ EOF
8. ปิดท้ายข้อมูลที่รับมาได้เรียบร้อยแล้วด้วย 1AH
9. ทำการเก็บข้อมูลที่รับได้ จากหน่วยความจำลงดิสก์

สำหรับซอร์สโปรแกรมของทั้งตัวรับและตัวส่งได้แสดงเอาไว้ในภาคผนวก



รูปที่ 5.4 แสดงโฟล์วชาร์ตของการส่งข้อมูลของ MASTER



รูปที่ 5.5 แสดงโฟล์วชาร์ตของการรับข้อมูลของ SLAVE

บทที่ 6

ผลการทดลอง

จากการสร้างภาค R.F.ทั้งภาคส่งและภาครับนั้นได้ผลการทดลองในส่วนต่างๆดังนี้

ภาครับ

สายอากาศ การสร้างจะต้องทำ Ground Plane ล้อมรอบสายอากาศเพื่อให้เกิดการ Matching Impedance กับภาค input ของ R.F.Amp. เพื่อทำให้การรับสัญญาณได้ผลดีที่สุด

ภาค R.F.Amp และ Demodulate ลายทองแดงเป็นสิ่งที่ต่อระวังเป็นอย่างมากในภาคนี้เนื่องจากที่ความถี่สูงนั้นลายทองแดงจะมีสภาพเหมือน L ซึ่งจะทำให้ค่าสัญญาณต่างๆเพี้ยนไปได้ ฉนั้นการสร้างเราจึงต้องพยายามให้ลายทองแดงจากตัวทรานซิสเตอร์ที่ต่อไปยัง element ต่างๆสั้นที่สุดเช่นจากขาเบสของตัวทรานซิสเตอร์ไปยังตัว R ต้องให้สั้นที่สุด

การปรับให้ได้สัญญาณที่ดีที่สุดนั้นเราไม่สามารถที่จะปรับสัญญาณของแต่ละภาคได้เนื่องจาก Impedance ของภาคต่างๆนั้นต่อเนื่องกันและการที่เราจะนำสาย Probe ไปทดลองวัดสัญญาณที่จุดใดจุดหนึ่งนั้นก็จะทำให้ Impedance ที่จุดนั้นเปลี่ยนแปลงไปเมื่อเราปรับและวัดสัญญาณที่จุดนั้นให้ได้สัญญาณที่แรงที่สุดแล้ว เมื่อนำสาย Probe ออกก็จะเกิดการเปลี่ยนแปลงของ Impedance ทำให้เกิดความผิดพลาดไปอีก ฉนั้นเราจะต้องทำการต่อภาคต่างๆรวมกันจนถึงภาคของการปรับแต่งสัญญาณ หรืออย่างน้อยก็ก่อนเข้าวงจร Comparater แล้วจึงทำการจับสัญญาณที่จุด output ของวงจรแล้วปรับให้ได้สัญญาณแรงที่สุดโดยการจูนจากตัว L ปรับค่า หรือจูนจากเครื่องส่งก็ได้ แต่ควรจะจูนทั้งสองอย่าง

ภาคส่ง ภาคส่ง

ส่วนสำคัญของภาคนี้คือ L และ C ที่ใช้ การสร้าง L บนลายทองแดงนั้นเป็นไปได้ยากที่จะได้ค่าที่ถูกต้องแน่นอนเนื่องมากจากการควบคุมค่าต่างๆทำได้ยาก จากสูตรการสร้าง L บนลายทองแดงแบบ circular spiral

$$L = 5.05*10^{-3}\ l(\ln(t/(W-t)-1.76 \quad nH/mil$$

เมื่อ W คือความกว้างของลายทองแดง และ t เป็นความหนาของลายทองแดง

ซึ่งจะเห็นได้ว่าการที่จะควบคุมทั้งความกว้างและความหนาของลายทองแดงนั้นทำได้ยาก จึงเปลี่ยนมาใช้แบบ square spiral ซึ่งมีสูตรการสร้างดังนี้

$$L = 8.5*A^{1/3}*n^{5/3} \quad nH$$

เมื่อ A เป็นพื้นที่ของลายทองแดงหน่วยเป็นตารางเซนติเมตร และ n เป็นจำนวนรอบ

จากการสร้างจริงนั้นเราใช้เครื่องต้นแบบมาวัดขนาดของลายทองแดงส่วนที่เป็น L นี้

แล้วจึงนำไปสร้างซึ่งก็ทำให้ง่ายขึ้น ซึ่งที่ทำเช่นนี้ก็เพื่อให้ค่า L เป็นไปอย่างแน่นอนนั่นเอง

ส่วน C ใช้เป็น Trimmer ซึ่งเสถียรภาพและความแน่นอนนั้นก็ขึ้นกับราคาของตัวมัน ซึ่งมีความแตกต่างกันมาก ในโครงงานที่ใช้เป็นแบบราคาถูกจึงทำให้เกิดความผิดพลาดได้ง่ายดังนั้นถ้าต้องการให้การทำงานเป็นไปอย่างแม่นยำที่สุดก็ควรที่จะใช้ Trimmer ที่มีราคาสูง

นอกจากนี้อีกส่วนที่สำคัญของวงจรภาคนี้ก็คือกำลังส่งซึ่งกำลังส่งนั้นขึ้นอยู่กับจุดแท็ปของ L ที่ต่อขึ้นไปยัง Vcc ถ้าจุดแท็ปนี้แบ่ง L ที่สร้างจากลายทองแดงได้ขนาดไม่เป็นสัดส่วนที่เหมาะสมก็อาจจะทำให้ได้กำลังส่งที่อ่อนเพราะว่าค่า B ของวงจรออสซิสเลเตอร์แบบ Hartley นี้ขึ้นอยู่กับ

$$B(w) = L_1 / L_2$$

<u>ภาคการอินเทอร์เฟสคอมพิวเตอร์</u>

จากผลการทดลองเมื่อได้มีการเชื่อมต่อระหว่าง PC ทั้ง 2 เครื่องโดยไม่ผ่านการอินเทอร์เฟสกับภาครับส่ง RF ข้อมูลที่รับได้ค่อนข้างถูกต้องโดยเฉพาะกับไฟล์ขนาดไม่เกิน 10 กิโลไบต์ แต่ถ้ามากกว่า 10 กิโลไบต์อาจจะทำให้ข้อมูลช่วงท้ายๆ ของไฟล์หายไปเนื่องจากทางตัวส่งไม่สามารถรับสัญญาณ ACK หรือ NAK ได้มีสาเหตุมาจากสัญญาณที่ส่งเกิดผิดพลาดขึ้นเพราะจากการทดลองส่งข้อมูลอย่างต่อเนื่องจะมีบางช่วงของข้อมูลที่ไม่สามารถรับข้อมูลได้ (BLANK หรือ 00 ในรหัสแอสกี้) สันนิษฐานว่าบัฟเฟอร์ที่เชื่อมต่อระหว่างบัสข้อมูลกับ INPUT PARALLEL ไม่ได้ PULL UP ไว้เพราะฉนั้นการรับส่งข้อมูลนานทำให้ขนาดของสัญญาณตกลงมาได้ และจากโปรแกรมการรับส่งเราสามารถแก้ไขโปรแกรมได้โดยให้มีการตรวจสอบสัญญาณว่ามาหรือไม่ภายในช่วงเวลาที่กำหนด ถ้าสัญญาณยังไม่มาก็ให้ตัวรอรับสัญญาณ (ในที่นี้คือตัว MASTER) ส่งสัญญาณขอให้ตอบ (ENQ) ไปให้แก่ SLAVE จากนั้น SLAVE จะทำการตรวจสอบว่าก่อนหน้านี้ได้มีการส่งสัญญาณอะไรออกไปให้ตัว MASTER ก็จะทำการส่งสัญญาณนั้นไปอีกครั้งหนึ่ง จะทำให้มีการรับส่งข้อมูลต่อไปได้

สำหรับการรับส่งข้อมูลเมื่อได้มีการ INTERFACE กับ RF โดยวิธีนี้ไม่สามารถทำการรับส่งข้อมูลออกไปให้ถูกต้องได้ เนื่องจากภาครับของภาค RF นั้นถึงแม้ว่าจะไม่มีการส่งสัญญาณออกมาจากภาคส่งแต่ที่ภาคสุดท้ายของภาครับก็จะมีสัญญาณออกมา ซึ่งเกิดเนื่องมาจากที่ส่วน DEMODULATE นั้นเกิดการออสซิลเลทที่ วงจร TANK เมื่อไม่มีสัญญาณเข้า ดังนั้นสัญญาณนี้ก็จะถูกส่งไปป้อนเข้าวงจรปรับแต่งสัญญาณ แล้วเข้าวงจร Comparator ทำให้เกิดการออสซิลเลทที่ภาคนี้ ฉนั้นเมื่อเราต่อภาค RF เข้ากับการติดต่อของ 8251 จึงทำให้เมื่อเปิดเครื่องแล้วและให้ 8251 ทำงานในโหมดรับ สัญญาณที่ออกมาจากภาคสุดท้ายของภาครับ RF จึงเข้าไปเป็น DATA ของ 8251 ด้วย ซึ่งส่วนนี้เราไม่สามารถแก้ไขได้ เพราะว่าถ้าเราทำให้ภาคสุดท้ายไม่เกิดการออสซิลเลทแล้วสัญญาณที่ออกมาก็จะเป็น 0 โวลท์ซึ่งถ้าไป

INTERFACE กับ PC เมื่อ 8251 ทำงานในโหมดรับ ก็ยังคิดว่าเป็น DATA เหมือนกัน เนื่องจาก 8251 จะไม่คิดว่าสัญญาณที่เข้าเป็น DATA เมื่อสัญญาณเป็น HIGH อยู่ก่อนซึ่งเราไม่สามารถทำได้

ถ้าเราจะทำการ INTERFACE RF กับ 8251 เราต้องเพิ่ม SOFTWARE ในส่วนต้นโปรแกรม ดังต่อไปนี้

MASTER

```
;***************
;เพื่อเปิด RF ภาคส่ง
;***************
MOV  DX,0304H
MOV  AL,00H       ;IF D0=0 TO ON TRANSMISSION RF
OUT  DX,AL
;************************
;เพื่อเลือก ENABLE การส่งรหัส
;************************
MOV  DX,0306H
MOV  AL,00H       ;IF D1=0 FOR SEND CODE
OUT  DX,AL
MOV  DX,0302H
MOV  AL,0AAH      ;0AAH IS CODE OF SLAVE
OUT  DX,AL
MOV  AL,00H       ;STOP TO SEND CODE
OUT  DX,AL
;*********************************
;เพื่อเลือกให้ภาคส่งสามารถติดต่อกับ 8251 ได้
;*********************************
MOV  DX,0306H
MOV  AL,01H       ;IF D1=1 FOR CONTACT WITH 8251
OUT  DX,AL
```

SLAVE

```
;***************
;เพื่อเปิด RF ภาคส่ง
;***************
MOV   DX,0302H
MOV   AL,00H        ;IF D0=0 TO ON TRASMISSION RF
OUT   DX,AL
```

สำหรับเบอร์พอร์ท และ DATA ที่กล่าวถึงในโปรแกรมข้างต้นได้เตรียมภาค HARDWARE ไว้รองรับเรียบร้อยแล้ว

ในกรณีที่ต้องการให้โปรแกรมของ SLAVE รันเพื่อรอรับการอินเทอร์รัพท์เราจะต้องเปลี่ยนส่วนแรกของโปรแกรมดังแสดงไว้ในโปรแกรม SLAVE1.ASM ในภาคผนวก สำหรับ ROUTINE ให้คงไว้เหมือนเดิม (SLAVE.ASM)

:

บทที่ 7
สรุปผลการทดลองและวิจารณ์

สรุปผลการทดลอง ในโครงงานนี้ในส่วนของการ INTERFACE กับ PC นั้นเราสามารถรับส่งข้อมูลได้อย่างถูกต้อง แต่เมื่อให้มีการรับส่งข้อมูลผ่านทางความถี่วิทยุตามโครงงานนี้ปรากฏว่ามีข้อผิดพลาดมากจนไม่สามารถนำมาใช้ในงานจริงได้ซึ่งไม่ได้เกิดจากความผิดพลาดของส่วนวงจรใด ดังที่ได้กล่าวมาแล้วข้างต้น ฉนั้นถ้าต้องการที่จะพัฒนาระบบนี้ให้สามารถใช้งานจริงได้ ก็จะต้องทำการพัฒนาภาครับส่ง RF ให้เป็นไปตามที่ได้แนะนำไว้

ภาคผนวก

```
;*********************************************************
; Program  for  send data  ( Master.asm )
;*********************************************************
 /              dosseg
                .model small
;*************
;DATA SEGMENT
;*************
                .data
namebuff        db    49
                db    ?
                db    50 dup(?)
buff            equ   20000
databuff        db    buff dup(?)
intro           db    0dh,0ah,'Enter Pathname for send:','$'
crlf            db    0dh,0ah,'$'
handle          dw    ?
psum            db    100 dup(0)
summing         db    10,13,10,13,'Summing data block #:  $'
summings        db    ' is $'
lask            db    ?
sizs            db    10,13,'File size = $'
sizes           dw    0
bytes           db    '  bytes$'
count           db    1
rec_NAK         dw    0
save_time       dw    ?
tell_you        db    0dh,0ah,'Initial 8251 and wait for ACK'
                db    ' form Receiver  5 sec.','$'
er1             db    '*** File not found ***','$'
er2             db    '*** Path not found ***','$'
er3             db    '*** Too many open file ***','$'
er4             db    '*** Wrong attribute or Access denied ***','$'
er5             db    '*** Invalid handle ***','$'
er6             db    '*** Invalid access code ***','$'
er7             db    '*** Error or Unknow ***','$'
continue        db    '* Send the next block $'
r_NAK_string    db    '* Send it again $'
ready           db    '* Send data successful $'
ACK_ready       db    'Receive ACK ready $'
```

```
ACK_fail        db    10,13,'Can not receive ACK $'
abort           db    10,13,'*** Can not contact $'
ssum            db    10,13,'Send summing OK $'
ssEOF           db    10,13,'Send end of file $'
sssETB          db    10,13,'Send end of block $'
p_8251c         equ    0301h
p_8251d         equ    0300h
EOT             equ    04
EOF             equ    1ah
ACK             equ    06
NAK             equ    21
ETB             equ    23

;************
;CODE SEGMENT
;************
                .code
                mov dx,@data
                mov ds,dx
                call clear_screen
                call set_cursor
                call open_file
                call read_file          ;load file for send in memmory
                call initial            ;initial 8251
                call tell
                call check_ACK          ;wait for ACK
                call send_f_size        ;send file size
                call check_ACK

                mov bx,offset databuff
                mov si,offset psum
send1:          call send_data
                mov dl,last             ;check last byte
                cmp dl,EOF
                jz s_EOF
                call send_ETB
                call send_summing
                call check_ACK_NAK_EOT
                jmp send1
```

```
s_EOF:          call send_EOF
                call send_summing
                call check_ACK_NAK_EOT
                cmp ax,NAK
                jz send1

exit:           mov ah,4ch
                int 21h

;**********************
;Send Summing of block
;**********************
send_summing    proc    near
s_sum:          mov dx,p_8251c
                in al,dx
                test al,01
                jz s_sum
                mov dx,p_8251d
                mov al,[si]
                out dx,al
                mov dx,offset ssum
                call print_string
                ret
send_summing    endp

;************************************
; Send Data and Summing Data until
;equal 256 bytes or found end of file
;************************************
send_data       proc    near
set:            mov cx,256              ;number of bytes in 1 block
                call delay
                xor ax,ax
                mov bp,ax
                mov [si],ax             ;push zero to si before sum data
pop_data:       mov al,[bx]             ;read data from buffer
                cmp al,1ah              ;if end of file
                jz sum1
                push ax
```

```
status:         mov dx,p_8251c
                in al,dx
                test al,01
                jz status
                pop ax
                mov dx,p_8251d
                out dx,al
                add [si],al             ;sum data
                inc bx                  ;next byte
                inc bp
                loop pop_data

sum1:           cmp cx,256              ;if no data in this block
                jz no_data              ;jump return
                mov last,al
                mov dx,offset summing   ;display summing
                call print_string
                xor dx,dx
                mov dl,[count]
                call binidec
                mov dx,offset summings
                call print_string
                xor dx,dx
                mov dl,[si]
                call binihex
                ret
no_data:        mov last,al
                ret
send_data       endp

;*********************************
;Check Control Signal from Receiver
;*********************************
check_ACK_NAK_EOT  proc    near
                call set_time
test_again:     call read_time
                cmp ax,[save_time]
                jae contact_fail
                mov dx,p_8251c
                in al,dx
```

```
                test al,02
                jz test_again
                mov dx,p_8251d
                in al,dx
                cmp al,ACK
                jz correct
                cmp al,NAK
                jz wrong
                cmp al,EOT
                jz complete

correct:        inc si
                inc [count]
                mov dx,offset continue
                call print_string
                mov [rec_NAK],0
                ret

wrong:          inc [rec_NAK]
                mov ax,[rec_NAK]
                cmp ax,10
                jz not_care
                mov ax,bp
                sub bx,ax
                mov dx,offset r_NAK_string
                call print_string
                mov ax,NAK
                ret

not_care:       cmp last,EOF
                jnz correct
                jz test_again0

complete:       mov dx,offset ready
                call print_string
                ret
```

```
contact_fail:   mov dx,offset abort
                call print_string
                mov ah,4ch
                int 21h
check_ACK_NAK_EOT  endp

;************************************
;Receive ACK for contact from Receiver
;************************************
check_ACK       proc    near
                call set_time

c_ACK:          call read_time
                cmp ax,[save_time]
                jae cancel
                mov dx,p_8251c
                in al,dx
                test al,02              ;test RxRDY
                jz c_ACK
                mov dx,p_8251d
                in al,dx
                cmp al,06
                jnz c_ACK
                mov dx,offset ACK_ready
                call print_string
                ret
cancel:         mov dx,offset ACK_fail
                call print_string
                mov ah,4ch
                int 21h
check_ACK       endp

;*****************
;Send size of file
;*****************
send_f_size     proc    near
                mov cx,[sizes]
s_f_s:          mov dx,p_8251c
                in al,dx
```

```
                test al,01
                jz s_f_s
                mov dx,p_8251d
                mov al,cl
                out dx,al
                call delay
s_f_s1:         mov dx,p_8251c
                in al,dx
                test al,01
                jz s_f_s1
                mov dx,p_8251d
                mov al,ch
                out dx,al
                ret
send_f_size     endp

;*******
;  Delay
;*******
delay           proc    near
                push bx
                mov bx,0
delay1:         dec bx
                jnz delay1
                pop bx
                ret
delay           endp

;*****************
;Open sending File
;*****************
open_file       proc    near
ask:            mov dx,offset intro     ;ask for name file
                call print_string

ans:            mov dx,offset namebuff  ;input name file
                mov ah,0ah
                int 21h
                mov dx,offset crlf
                call print_string
```

```
zero:           mov bl,namebuff+1       ;put zero in name file handle
                mov bh,0
                mov [namebuff+bx+2],0

open:           mov dx,offset namebuff+2
                mov al,0
                mov ah,3dh              ;open file
                int 21h
                mov handle,ax           ;save handle
                jc  error1
                ret
error1:         call error
open_file       endp

;*****************
;Read Sending File
;*****************
read_file       proc    near
read:           mov cx,buff             ;number of bytes for read
                mov bx,handle
                mov dx,offset databuff
                mov ah,3fh              ;read file
                int 21h
                jc error2               ;if error
                cmp ax,0                ;if found end of file
                jz close1
                mov cx,ax
                cmp cx,buff             ;if file size more than buffer
                pushf
                add [sizes],cx          ;save it before read again
                popf
                jnz disp                ;if file size less than buffer
d1:
                mov bx,handle
                jmp read                 ;read again

disp:           mov dx,offset sizs      ;display file size
                call print_string
                mov dx,[sizes]
                call binidec
```

```
                mov dx,offset bytes
                call print_string
close1:         call close_file
                ret
error2:         call error
read_file       endp

;************************
;Close File for Openning
;************************
close_file      proc    near
                mov bx,handle           ;close file
                mov ah,3eh
                int 21h
                jc error3
                ret

error3:         call error
close_file      endp

;*****************************
;Change Binary to Hexadecimal
;*****************************
binihex         proc    near
                push bx
                xor bx,bx
                mov bx,dx
                mov ch,4

rot:            mov cl,4
                rol bx,cl
                mov al,bl
                and al,0fh
                add al,30h
                cmp al,3ah
                jl printit
                add al,7h
```

```
printit:        mov dl,al
                mov ah,2
                int 21h
                dec ch
                jnz rot
                mov dx,offset crlf
                call print_string
                pop bx
                ret
binihex         endp

binidec         proc    near
                push si
                mov ax,dx
                mov si,10
                xor cx,cx
nonzero:        xor dx,dx
                div si
                push dx
                inc cx
                or ax,ax
                jne nonzero
w:              pop dx
                call writehexd
                loop w
                pop si
                ret
binidec         endp

writehexd       proc    near
                push dx
                cmp dl,10
                jae hexletter
                add dl,"0"
                jmp short wdigit
```

```
hexletter:      add dl,"A"-10
wdigit:         mov ah,2
                int 21h
                pop dx
                ret
writehexd       endp

;**************
;INITIAL  8251
;**************
initial         proc    near
                mov dx,p_8251c          ;control port
                mov al,55h              ;int. reset
                out dx,al
                mov al,0ddh             ;mode word
                out dx,al
                mov al,15h              ;command word
                out dx,al
                ret
initial         endp

;***********
;TELL  USER
;***********
tell            proc    near
                mov dx,offset tell_you
                call print_string
                mov dx,offset crlf
                call print_string
                ret
tell            endp

;*************
;Print String
;*************
print_string    proc    near
                mov ah,09h
                int 21h
                ret
print_string    endp
```

```
;*****************
;SEND  End of File
;*****************
send_EOF        proc    near
eof1:           mov dx,p_8251c
                in al,dx
                test al,01
                jz eof1
                mov dx,p_8251d
                mov al,EOF
                out dx,al
                mov dx,offset ssEOF
                call print_string
                ret

send_EOF        endp

;*****************
;Send End of Block
;*****************
send_ETB        proc    near
etb1:           mov dx,p_8251c
                in al,dx
                test al,01
                jz etb1
                mov dx,p_8251d
                mov al,ETB
                out dx,al
                mov dx,offset sssETB
                call print_string
                ret
send_ETB        endp

;***********
;  SET TIME
;***********
set_time        proc    near
                push es
                mov ax,0
                mov es,ax
```

```
                mov ax,word ptr es:[046ch] ;read data in address 0:46CH to
                add ax,91
                mov [save_time],ax
                pop es
                ret
set_time        endp

;************
;  READ TIME
;************
read_time       proc    near
                push es
                mov ax,0
                mov es,ax
                mov ax,word ptr es:[046ch] ;read data in address 0:46CH to
                pop es
                ret
read_time       endp

;*************
;  SET CURSOR
;*************

set_cursor      proc    near
                push ax
                push bx
                push dx
                xor dx,dx
                xor bx,bx               ;set cursor rountine
                mov ah,2                ;goto column 0, row 0
                int 10h
                pop dx
                pop bx
                pop ax
                ret
set_cursor      endp
```

```
;***************
;  CLEAR_SCREEN
;***************
clear_screen    proc    near
                push ax
                push bx
                push cx
                push dx
                xor ax,ax
                xor cx,cx
                mov dh,24
                mov dl,79
                mov bh,7
                mov ah,6
                int 10h
                pop dx
                pop cx
                pop bx
                pop ax
                ret
clear_screen    endp

;***************
;ERROR  ROUTINE
;***************
error           proc    near
                cmp ax,2
                jz e1
                cmp ax,3
                jz e2
                cmp ax,4
                jz e3
                cmp ax,5
                jz e4
                cmp ax,6
                jz e5
                cmp ax,12
                jz e6
                jnz e7
```

```
e1:             mov dx,offset er1
                jmp e0
e2:             mov dx,offset er2
                jmp e0
e3:             mov dx,offset er3
                jmp e0
e4:             mov dx,offset er4
                jmp e0
e5:             mov dx,offset er5
                jmp e0
e6:             mov dx,offset er6
                jmp e0
e7:             mov dx,offset er7
                jmp e0
e0:             call print_string

                mov ah,4ch
                int 21h
error           endp

                END
```

```
;*****************************************************************
;Program  for receive data  (Slave.asm)
;*****************************************************************
                dosseg
                .model small
;***************
; DATA  SEGMENT
;***************
                .data
namebuff1       db    49
                db    ?
                db    50 dup(?)

intro1          db    0dh,0ah,'Save data in file name:','$'
handle1         dw    ?
file_sizes      dw    0
lf_s            db    ?
mf_s            db    ?
databuff        db    20000 dup (?)
psum            db    100 dup(0)
?sum            db    ?
last1           db    0
count           db    1
num_NAK         dw    0
save_time       dw    ?
tell_f_s        db    10,13,'File size = $'
bytes           db    ' bytes$',10,13
er1             db    '*** File not found ***','$'
er2             db    '*** Path not found ***','$'
er3             db    '*** Too many open file ***','$'
er4             db    '*** Wrong attribute or Access denied ***','$'
er5             db    '*** Invalid handle ***','$'
er6             db    '*** Invalid access code ***','$'
er7             db    '*** Error or Unknow ***','$'
crlf            db    10,13,'$'
ACK_ready       db    10,13,'Send ACK ready $',10,13
cmt             db    ' Can not contact $'
ask1_user       db    'Do you want to contact ?(Y/N)$'
found_ETB       db    10,13,'Found end of block ',10,13,'$'
found_EOF       db    10,13,'Found end of file',10,13,'$'
```

```
sEOT            db    10,13,'Data in ready and Contact complete',10,13,'$'
fool            db    10,13,'Press only key Y or N stupid $',10,13
uns             db    10,13,'Send ACK unsuccessful $'
ss_NAK          db    10,13,'Send NAK and receive data again $'
d_sum           db    ' Can not receive sum of data $'
p_8251c         equ   0301h
p_8251d         equ   0300h
EOT             equ   04
EOF             equ   1ah
ACK             equ   06
NAK             equ   21
ETB             equ   23
fail            equ   24

;***************
; CODE  SEGMENT
;***************
                .code
                mov dx,@data
                mov ds,dx
                call clear_screen
                call set_cursor
                call initial
                mov dx,offset ask1_user
                call print_string
                call dicision
                call send_ACK
                call rec_file_size          ;receive size of file
                call send_ACK

                mov di,offset databuff
                mov si,offset psum
r_data:         call receive_data           ;receive data 256 bytes and
                cmp ax,fail                 ;summing data
                jz out1
                call check_sum              ;receive sum of data and check
                cmp ax,NAK
                jz s_NAK
```

```
                cmp last1,EOF
                jz s_EOT
                cmp ax,fail
                jz out1
                call send_ACK
                inc si
                jmp r_data

s_NAK:          inc [num_NAK]           ;check send NAK
                call send_NAK
                mov ax,[num_NAK]
                cmp ax,10
                jz not_care             ;send NAK equal 10 th
                jmp r_data

not_care:       cmp last1,EOF           ;check last block
                jz s_EOT
                xor ax,ax
                mov [num_NAK],ax        ;yet set it for receive again
                mov ax,di
                add ax,bp
                mov di,ax
                jmp r_data

s_EOT:          call send_EOT           ;complete
out1:           call create_file
                call write_file
                call close_file
                jmp exit

exit:           mov ah,4ch
                int 21h
```

```
;*********************************
;Receive size of file from Sender
;*********************************
rec_file_size  proc    near
               call set_time
r_f_s1:        call read_time
               cmp ax,[save_time]
               jae eexit
               mov dx,p_8251c
               in al,dx
               test al,02                ;test RxRDY
               jz r_f_s1
               mov dx,p_8251d
               in al,dx
               mov [lf_s],al             ;receive first byte
r_f_s2:        mov dx,p_8251c
               in al,dx
               test al,02                ;test RxRDY
               jz r_f_s2
               mov dx,p_8251d
               in al,dx
               mov [mf_s],al             ;receive secound byte
               mov ah,[mf_s]
               mov al,[lf_s]
               mov [file_sizes],ax       ;push it
               mov dx,offset tell_f_s
               call print_string
               mov dx,[file_sizes]       ;display it
               call binidec
               mov dx,offset bytes
               call print_string
               ret
eexit:         mov dx,offset cmt         ;if can not contact
               call print_string
               mov ah,4ch
               int 21h
rec_file_size  endp
```

```
;******************
; Send Acknowledge
;******************
send_ACK        proc    near
                call set_time
sACK0:          call read_time
                cmp ax,[save_time]
                jae unsu0
                mov dx,p_8251c
                in al,dx
                test al,01                      ;test TxRDY
                jz sACK0
                mov al,ACK
                mov dx,p_8251d
                out dx,al
                mov dx,offset ACK_ready
                call print_string
                ret

unsu0:          mov dx,offset uns               ;if can not send
                call print_string
                mov ah,4ch
                int 21h
send_ACK        endp

;*****************************
; Receive  data  and Sum data
;*****************************
receive_data    proc    near
receive0:       xor ax,ax
                mov bp,ax
                call set_time
receive_data1:  call read_time
                cmp ax,[save_time]
                jae abort
                mov dx,p_8251c
                in al,dx
                test al,02                      ;test RxRDY
                jz receive_data1
```

```
                mov dx,p_8251d
                in al,dx
                cmp al,ETB
                jz r_ETB                    ;found end of block
                cmp al,EOF
                jz r_EOF                    ;found end of file
                mov [di],al
                add [si],al                 ;summing data
                inc di
                inc bp
                jmp receive_data1

r_ETB:          mov dx,offset found_ETB
                call print_string
                ret

r_EOF:          mov last1,al
                mov dx,offset found_EOF
                call print_string
                ret

abort:          mov dx,offset cmt
                call print_string
                mov ax,fail
                ret
receive_data    endp

;********************
; Check both of sum
;********************
check_sum       proc    near
                call set_time
c_sum1:         call read_time
                cmp ax,[save_time]
                jae c_sum3
                mov dx,p_8251c
                in al,dx
                test al,02
                jz c_sum1
```

```
                mov dx,p_8251d
                in al,dx
                mov ?sum,al
                mov al,[si]
                cmp ?sum,al
                jnz c_sum2
                ret

c_sum2:         mov ax,NAK
                ret

c_sum3:         mov dx,offset d_sum
                call print_string
                mov ax,fail
                ret
check_sum       endp

;*****************************
; Send Negative Acknowledge
;*****************************
send_NAK        proc    near
                push cx
sNAK1:          mov dx,p_8251c
                in al,dx
                test al,01
                jz sNAK1
                mov dx,p_8251d
                mov al,NAK
                out dx,al
                mov dx,offset ss_NAK
                call print_string
                mov ax,di
                mov cx,bp
                sub ax,cx
                mov di,ax
                pop cx
                ret
send_NAK        endp
```

```
;**************************
; Send End of Transmission
;**************************
send_EOT        proc    near
sEOT1:          mov dx,p_8251c
                in al,dx
                test al,01
                jz sEOT1
                mov dx,p_8251d
                mov al,EOT
                out dx,al
                mov dx,offset sEOT
                call print_string
                ret
send_EOT        endp

;************************
; Create receiving file
;************************
create_file     proc    near
ask1:           mov dx,offset intro1
                mov ah,9
                int 21h

ans1:           mov dx,offset namebuff1
                mov ah,0ah
                int 21h

zero1:          mov bl,namebuff1+1
                mov bh,0
                mov [namebuff1+bx+2],0

cre1:           mov dx,offset namebuff1+2
                mov cx,0
                mov ah,3ch
                int 21h
                mov handle1,ax
                jc error1
                ret
```

```
error1:         call error
create_file     endp

;*******************************
; Write receiving file to disk
;*******************************
write_file      proc    near
write1:         mov cx,[file_sizes]
                mov bx,handle1
                mov dx,offset databuff
                mov ah,40h
                int 21h
                jc error2
                ret
error2:         call error
write_file      endp

;***********************
; Close receiving file
;***********************
close_file      proc    near
                mov bx,handle1
                mov ah,3eh
                int 21h
                jc error3
                ret
error3:         call error
close_file      endp

;*************************
; Change Binary to Decimal
;*************************
binidec         proc    near
                push si
                mov ax,dx
                mov si,10
                xor cx,cx
```

```
nonzero:        xor dx,dx
                div si
                push dx
                inc cx
                or ax,ax
                jne nonzero
w:              pop dx
                call writehexd
                loop w
                pop si
                ret
binidec         endp

writehexd       proc    near
                push dx
                cmp dl,10
                jae hexletter
                add dl,"0"
                jmp short wdigit

hexletter:      add dl,"A"-10
wdigit:         mov ah,2
                int 21h
                pop dx
                ret
writehexd       endp

;**************
;INITIAL  8251
;**************
initial         proc    near
                mov dx,p_8251c          ;control port
                mov al,55h              ;int. reset
                out dx,al
                mov al,0ddh             ;mode word
                out dx,al
                mov al,15h              ;command word
                out dx,al
                ret
initial         endp
```

```
;****************
; Print  String
;****************
print_string    proc    near
                mov ah,09h
                int 21h
                ret
print_string    endp

;*************
;  SET TIME
;*************
set_time        proc    near
                push es
                mov ax,0
                mov es,ax
                mov ax,word ptr es:[046ch]  ;read data in address 0:46CH
                add ax,91
                mov [save_time],ax
                pop es
                ret
set_time        endp

;*************
;  READ_TIME
;*************
read_time       proc    near
                push es
                mov ax,0
                mov es,ax
                mov ax,word ptr es:[046ch]  ;read data in address 0:46CH
                pop es
                ret
read_time       endp
```

```
;*************
;  SET_CURSOR
;*************

set_cursor      proc    near
                push ax
                push bx
                push dx
                xor dx,dx
                xor bx,bx                       ;set cursor rountine
                mov ah,2                        ;goto column 0, row 0
                int 10h
                pop dx
                pop bx
                pop ax
                ret
set_cursor      endp

;***************
;  CLEAR_SCREEN
;***************
clear_screen    proc    near
                push ax
                push bx
                push cx
                push dx
                xor ax,ax
                xor cx,cx
                mov dh,24
                mov dl,79
                mov bh,7
                mov ah,6
                int 10h
                pop dx
                pop cx
                pop bx
                pop ax
                ret
clear_screen    endp
```

```
;***************
;ERROR  ROUTINE
;***************
error           proc    near
                cmp ax,2
                jz e1
                cmp ax,3
                jz e2
                cmp ax,4
                jz e3
                cmp ax,5
                jz e4
                cmp ax,6
                jz e5
                cmp ax,12
                jz e6
                jnz e7

e1:             mov dx,offset er1
                jmp e0
e2:             mov dx,offset er2
                jmp e0
e3:             mov dx,offset er3
                jmp e0
e4:             mov dx,offset er4
                jmp e0
e5:             mov dx,offset er5
                jmp e0
e6:             mov dx,offset er6
                jmp e0
e7:             mov dx,offset er7
                jmp e0
e0:             call print_string

                mov ah,4ch
                int 21h
error           endp
```

```
;**********************
; Wait for key Y or N
;**********************
dicision        proc    near
dd1:            mov ah,01
                int 21h
                cmp al,6eh
                jz exit1
                cmp al,4eh
                jz exit1
                cmp al,79h
                jz dd2
                cmp al,59h
                jz dd2
                mov dx,offset fool
                call print_string
                jmp dd1
dd2:            ret
exit1:          mov ah,4ch
                int 21h
dicision        endp

                END
```

```
;*****************************************************************
;Program  for receive data  (Slave1.asm)
;*****************************************************************
                dosseg
                .model small
;***************
; DATA  SEGMENT
;***************
                .data
namebuff1       db    49
                db    ?
                db    50 dup(?)

intro1          db    0dh,0ah,'Save data in file name:','$'
handle1         dw    ?
file_sizes      dw    0
lf_s            db    ?
mf_s            db    ?
databuff        db    20000 dup (?)
psum            db    100 dup(0)
?sum            db    ?
last1           db    0
count           db    1
num_NAK         dw    0
save_time       dw    ?
tell_f_s        db    10,13,'File size = $'
bytes           db    ' bytes$',10,13
wait_int        db    'Waiting for Interrupt (IRQ2) ....$'
er1             db    '*** File not found ***','$'
er2             db    '*** Path not found ***','$'
er3             db    '*** Too many open file ***','$'
er4             db    '*** Wrong attribute or Access denied ***','$'
er5             db    '*** Invalid handle ***','$'
er6             db    '*** Invalid access code ***','$'
er7             db    '*** Error or Unknow ***','$'
crlf            db    10,13,'$'
ACK_ready       db    10,13,'Send ACK ready $',10,13
cmt             db    ' Can not contact $'
ask1_user       db    'Do you want to contact ?(Y/N)$'
found_ETB       db    10,13,'Found end of block ',10,13,'$'
```

```
found_EOF       db    10,13,'Found end of file',10,13,'$'
sEOT            db    10,13,'Data in ready and Contact complete',10,13,'$'
fool            db    10,13,'Press only key Y or N stupid $',10,13
uns             db    10,13,'Send ACK unsuccessful $'
ss_NAK          db    10,13,'Send NAK and receive data again $'
d_sum           db    ' Can not receive sum of data $'
ans_int         db    10,13,'* Interrupt Complete * $'
p_8251c         equ    0301h
p_8251d         equ    0300h
EOT             equ    04
EOF             equ    1ah
ACK             equ    06
NAK             equ    21
ETB             equ    23
fail            equ    24

;****************
; CODE  SEGMENT
;****************
                .code

main            proc    near
                mov ax,@data
                mov ds,ax

                in al,21h                   ;Program 8259 for int
                and al,0fbh
                out 21h,al

                push ds
                mov ax,cs
                mov ds,ax
                mov ah,25h                  ;Set  new int vector
                mov al,0ah
                mov dx,offset isr
                int 21h
                pop ds
```

```
                call clear_screen
                call set_cursor
                mov dx,offset wait_int      ;Display message
                mov ah,9h
                int 21h

main1:          mov ah,1
                int 16h                     ;check that press any
                jz main1                    ;key

                mov ah,4ch                  ;Return to Dos
                int 21h

main            Endp

;-------------------------------------------------
Isr             proc
                Push ax
                Push bx
                Push cx
                Push dx
                Push si
                Push di
                Push es
                Push ds

                mov dx,offset ans_int
                call print_string
                call initial
                mov dx,offset ask1_user
                call print_string
                call dicision
                call send_ACK
                call rec_file_size          ;receive size of file
                call send_ACK

                mov di,offset databuff
                mov si,offset psum
```

```
r_data:     call receive_data          ;receive data 256 bytes and
            cmp ax,fail                ;summing data
            jz out1
            call check_sum             ;receive sum of data and check
            cmp ax,NAK
            jz s_NAK
            cmp last1,EOF
            jz s_EOT
            cmp ax,fail
            jz out1
            call send_ACK
            inc si
            jmp r_data

s_NAK:      inc [num_NAK]              ;check send NAK
            call send_NAK
            mov ax,[num_NAK]
            cmp ax,10
            jz not_care                ;send NAK equal 10 th
            jmp r_data

not_care:   cmp last1,EOF              ;check last block
            jz s_EOT
            xor ax,ax
            mov [num_NAK],ax           ;yet set it for receive again
            mov ax,di
            add ax,bp
            mov di,ax
            jmp r_data

s_EOT:      call send_EOT              ;complete
out1:       call create_file
            call write_file
            call close_file
            jmp exit

exit:       mov al,20h                 ;Enable  Interrupt
            out 20h,al
            Pop ds
            Pop es
```

```
                Pop di
                Pop si
                Pop dx
                Pop cx
                Pop bx
                Pop ax
                Iret
Isr             endp
```

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้มีอาจที่จะสำเร็จลงได้ตามจุดประสงค์ ถ้าขาดการให้คำแนะนำจากอาจารย์ที่ปรึกษา อ.พลผดุง ผดุงกุล และอาจารย์ท่านอื่นอีกหลายท่าน ตลอดจนความร่วมมือในการคิดแก้ไขปัญหาจากเพื่อนๆ ภาควิชาคอมพิวเตอร์และภาควิชาไฟฟ้ากำลัง จึงขอขอบคุณมา ณ โอกาสนี้

Z-80

เอกสารอ้างอิง

ยืน ภู่วรวรรณ , "ทฤษฎีและการประยุกต์ไมโครโปเซสเซอร์ Z-80",บริษัท ซีเอ็ดยูเคชัน จำกัด , พ.ศ.2532

ธานินทร์ ถาวรศาสนวงศ์ และ ทินกร ดุ๊ก , "การอินเทอร์เฟส IBM PC" , ฟิสิกส์เซ็นเตอร์ การพิมพ์

สุริยัน ศรีสวัสดิ์กุล , "ระบบสื่อสารข้อมูลคอมพิวเตอร์" , ฟิสิกส์เซ็นเตอร์ การพิมพ์

ROBERT LAFORE , "ASSEMBLY LANGUAGE PRIMER FOR THE IBM PC AND XT" , A PLUME / WAITE BOOK

ชูชัย ธนสารตั้งเจริญ, ดนัย แสงสุริยศิลป์, ทินกร ดุ๊ก, ธงชัย อุดมกิจโกศล และ ธานินทร์ ถาวรศาสนวงศ์ , "การใช้งาน Z-80" , ฟิสิกส์เซ็นเตอร์ การพิมพ์

ดร.สิทธิชัย โภไคยอุดม, ดร.นิรศักดิ์ วรสุนทโรสถ และ โตะมิโอะ อิวะสะกิ , "ทฤษฎีและการคำนวณวงจรอิเลคทรอนิคส์", ซีเอ็ดยูเค่ชัน

วคิน เพิ่มทรัพย์, "คู่มือ MS-DOS PC-DOS", ดวงกลมสมัย