ปีการศึกษา 2530

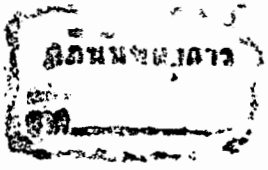
SMART MODEM

โดย

นาย อภิรักษ์ จิรโสภณ

อาจารย์ ที่ ปรึกษา

อาจารย์ สุชาติ แสงหิรัญ



ปริญญานิพนธ์ปีการศึกษา 2530

เรื่อง สมาร์ท โมเด็ม (SMART MODEM)

ผู้จัดทำ

นาย อภิรักษ์ จิรโสภณ

(.....) อาจารย์ที่ปรึกษา

นาย สุชาติ แสงหิรัญ

สมาร์ท โมเด็ม

นาย อภิรักษ์ จิรโสภณ

อาจารย์ สุชาติ แสงหิรัญ อาจารย์ ที่ ปรึกษา

ปีการศึกษา 2530

บทคัดย่อ

สมาร์ท โมเด็ม (Smart Modem) เป็นเซอร์กิต คาร์ด (Circuit card) ที่ใช้ส่งและรับข้อมูล (Transmit and receive data) จากเครื่องคอมพิวเตอร์ เครื่องหนึ่งไปยังอีกเครื่องหนึ่งที่อยู่ห่างไกล โดยใช้สายโทรศัพท์เป็นอุปกรณ์อินเตอร์เฟส ซึ่งคาร์ดนี้จะเปลี่ยนดาต้า ซิกแนล (Data signal) ที่อยู่ในรูปของ ดิจิตอล ซิกแนล (Digital-signal) ให้เป็น อนาล็อก ซิกแนล (Analog signal) แล้วส่งออกไป และจะรับดาต้า ซิกแนล ซึ่งอยู่ในรูปของอนาล็อก ซิกแนล แล้วนำมาเปลี่ยนให้อยู่รูปของ ดิจิตอล ซิกแนล จึงจะส่งไปยังเครื่องคอมพิวเตอร์ต่อไป

สมาร์ท โมเด็ม ที่ออกแบบขึ้นมานี้ ใช้เทคนิคการโมดูลเลชั่น (Modulation) และ ดีโมดูลเลชั่น (Demodulaltion) เป็นแบบ Frequency shift keying (FSK) ซึ่งใช้ชิพ (Chip) สำเร็จรูปเป็นตัวโมเด็ม (Modem) และลักษณะการติดต่อ เป็นแบบฮาร์ฟดูเพล็ก (Half duplex) ส่งด้วย 600 , 1200 บอด (Bauds)

Coupler Techniques ใช้แบบ Direct coupler techniques และการกดปุ่มโทรศัพท์ (การโทรเข้าไปยังอีกเครื่องหนึ่ง) ใช้สมาร์ทโมเด็มเซอร์กิตคาร์ดกดปุ่มให้โดยอัตโนมัติ

ซอฟแวร์ สนับสนุนการใช้เทคนิคการเขียนโปรแกรม แบบเรซิเดนท์ โปรแกรม (Resident program) ลงบนหน่วยความจำของคอมพิวเตอร์ เมื่อจะเรียกใช้ก็สะดวกในการใช้งาน โดยมีเมนู (Menu)ให้เลือกหลายอย่าง เครื่องคอมพิวเตอร์ที่สามารถใช้ได้กับสมาร์ทโมเด็มเซอร์กิตคาร์ด คือเครื่องไอบีเอ็ม PC/XT โดยสามารถเสียบคาร์ดลงในสล้อต (Slot) ได้เลย

SMART MODEM

APIRAK CHIRASOPONE

SUCHART SANGHIRUN ..ADVISOR..

ACADEMIOS YEAR 1987

ABSTRACT

SMART MODEM is a circuit card that use for transmit and receive data from the one computer to the other one that far, by Telephone Line which the interface tods. This card will be change Data Signel which is a Digitel Signal to be Analog Signal out puts , and can receive Data Signal that in Analog Signal Symbol. For - change in Digital. Signal and send to the computer go on.

This Smart Modem is design for technic Modulation and Demodulation. The Frequency Shift keying (FSK) use for a ready made Ship Modem. And the communication contect of Half Duplex can send about 600,1200 Bauds.

Coupler Techniques use by direct coupler techniques and the press-telephone.[The telephone call from one to the other]by Smart Modem circuit card that will do by Automatic.

Soft Ware is support by techic program writting Resident Program in the Computer Memory that very comfortable to use it by choose many manu. The computer that can use with Smart Modem circuit card is IBM PC/xt. But the speed of IBM PC/xt will not more 6 MHz.

สารบัญ

บทที่ 1

บทนำ

ในปัจจุบันนี้การสื่อสารข้อมูลได้เข้ามามีบทบาทอย่างมากในวงการธุรกิจ การที่จะย้ายแฟ้มข้อมูลต่างๆ ที่เก็บอยู่ในคอมพิวเตอร์ไปยังคอมพิวเตอร์อีกเครื่องหนึ่งซึ่งอยู่ห่างไกลซึ่งจะเป็นการติดต่อระหว่างบริษัทต่อบริษัทหรือระหว่างบุคคลนั้นจะสะดวกมากขึ้น เมื่อใช้การสื่อสารทางสายโทรศัพท์ ดังนั้น MODEM จึงเป็นส่วนประกอบที่สำคัญมากในการส่งข้อมูลระหว่างคอมพิวเตอร์ 2 เครื่องที่อยู่ห่างไกลซึ่ง MODEM จะเป็นตัวเปลี่ยนสัญญาณจากดิจิตอลไปเป็นสัญญาณอนาลอก แล้วจึงส่งข้อมูลไปทางสายโทรศัพท์ ส่วนการรับนั้นก็จะรับข้อมูลอนาลอกแล้วเปลี่ยนไปเป็นสัญญาณดิจิตอล ปัจจุบัน MODEM ได้รวมเอาวงจรเกี่ยวกับการกดปุ่มหรือหมุนโทรศัพท์และส่วนที่เป็นบัฟเฟอร์ ทำให้ MODEM กลายเป็นอุปกรณ์ที่ชาญฉลาดมากยิ่งขึ้น เช่น กรณีที่ไม่มีใครอยู่ที่เครื่องคอมพิวเตอร์ ในขณะที่มีสัญญาณข้อมูลจากอีกเครื่องส่งมาตัวโมเด็มก็จะจัดการตอบรับและเก็บข้อมูลเองได้ ซึ่งโมเด็มประเภทนี้เรียกว่า Intelligent Modem ส่วนโมเด็มอีกแบบคือ Acoustic coupled Modem ซึ่งใช้การส่งรับสัญญาณเสียงทางหูฟังโทรศัพท์และ อีกแบบหนึ่งคือ Internal Modems ซึ่งในปริญญานิพนธ์นี้ได้ทำแบบ Internal โดยสามารถต่อสัญญาณกับสายโทรศัพท์ได้โดยอัตโนมัติ

ปัจจุบันการนำเอาโมเด็มมาใช้ในประเทศเรานั้นจะมีปัญหาเกี่ยวกับสัญญาณรบกวนของสัญญาณโทรศัพท์ ทำให้ไม่สามารถจะส่งข้อมูลด้วย baud rate สูงๆได้ ซึ่งปัจจุบันก็ใช้กันอยู่ตั้งแต่ 300,600,1200,และ 2400 baud rate

บทที่ 2 ทฤษฎีโมเด็ม (MODEM THEORY)

คำนำ

เมื่อมีการเคลื่อนย้ายข้อมูล เช่น จากเทอร์มินอล (TERMINAL) ไปยังเครื่องคอมพิวเตอร์ ซึ่งตามปกติแล้วจะใช้ส่งแบบดิจิตอล พัลซ์ (DIGITAL PULSE) เราจะมาพิจารณาในการส่งสัญญาณพัลซ์ (TRANSMITTING PULSE) ที่เป็นโวลท์เตจ พัลซ์ (VOL--TAGE PULSE) ลงในสายส่ง (WIRE PAIR) ซึ่งจริงๆ แล้วเราไม่สามารถส่งสัญญาณผ่านชุมสายโทรศัพท์ผ่านอุปกรณ์สวิทชิ่ง (SWITCHING DEVICES) ได้แต่สามารถผ่านเข้าไปยังชุมสายท้องถิ่น (LOCAL EXCHANGE) ได้ โดยที่สามารถผ่านไปยังชุมสายเดียวกันได้แต่ก็ไม่สามารถผ่าน สวิทชิ่ง แมชีน (SWITCHING MACHINE) ได้ ทำให้เกิดความจำเป็นในการใช้สายแบบส่วนตัว (PRIVATE LINE) ถ้าหากว่าเทอร์มินอลอยู่ใกล้กับคอมพิวเตอร์ซึ่งระยะทางน้อยกว่า 100 เมตร ก็สามารถใช้ส่งสัญญาณกันได้

ระยะทางในการส่งสัญญาณแบบพัลซ์จะทำให้เกิดการเพี้ยนของสัญญาณนั้นได้ด้วยค่าของรีซิสแตนซ์ (RESISTANCE) , ชันท์ตัวประจุ (SHUNT CAPACITANCE) , และอินดัคแตนซ์ที่อนุกรม (SERIES INDUCTANCE) รวมด้วยกันในสายส่ง (TRANSMISSION LINE) ไดอีเล็คตริค (DIELECTRIC) เป็นตัวกำหนดการรั่ว (LEAKAGE) ผ่านฉนวนปกติแล้วไม่นำมาพิจารณา ส่วนไลน์ รีซิสแตนซ์ (LINE RESISTANCE) นั้น จะขึ้นอยู่กับความถี่มาก ซึ่งมันจะทำให้โวลท์เตจ พัลซ์ ลดลงที่ความถี่สูงๆ ตัวสุดท้ายคือ รีซิสแตนซ์ (RESISTANCE) จะเป็นตัวที่ทำให้มีผลต่อผิวเปลือกนอก (SKIN EFFECT) , ผลข้างเคียง (PROXIMITY EFFECT) และการกระจายการสูญเสีย (RADIATION LOSSES)

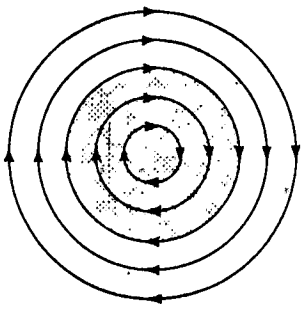
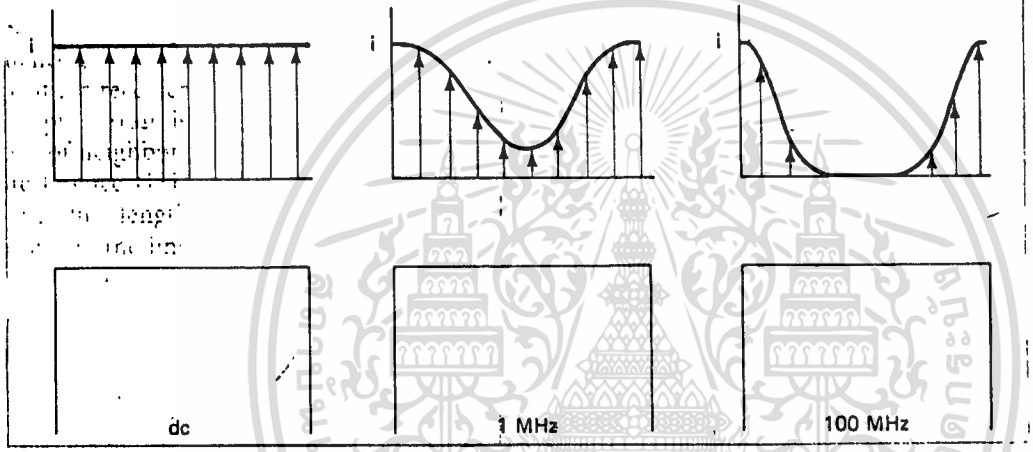


Figure 5-1 Flux Around and In Current-Carrying Conductor



รูปที่ 1

<u>โมเด็ม</u> (MODEM)

ในการส่งสัญญาณข้อมูลที่ระยะทางไกลๆ ก็จำเป็นที่จะต้องมีการเปลี่ยนแปลงสัญญาณ และส่งสัญญาณลงไปในสายโทรศัพท์ ซึ่งในสายโทรศัพท์นั้น มีช่องแบนด์วิดท์ (BAND WIDTH) ให้สัญญาณผ่านประมาณ 300-400 Hz ซึ่งก็เป็นช่วงความถี่เสียง ดังนั้น สัญญาณดิจิตอล (DIGITAL) ที่จะส่งก็จะต้องมีการผสมสัญญาณให้เป็นความถี่เสียง และถ้าจะรับสัญญาณก็ต้องมีการแปลงสัญญาณเสียงให้เป็น สัญญาณดิจิตอล ซึ่งเรียกว่า การดีโมดูเลท (DEMOULATE) ดังนั้นคำว่าโมเด็ม (MODEM) ก็ย่อมาจาก MODULATION AND DEMODULATION

ชนิดของการผสมสัญญาณ (TYPES OF MODULATION)

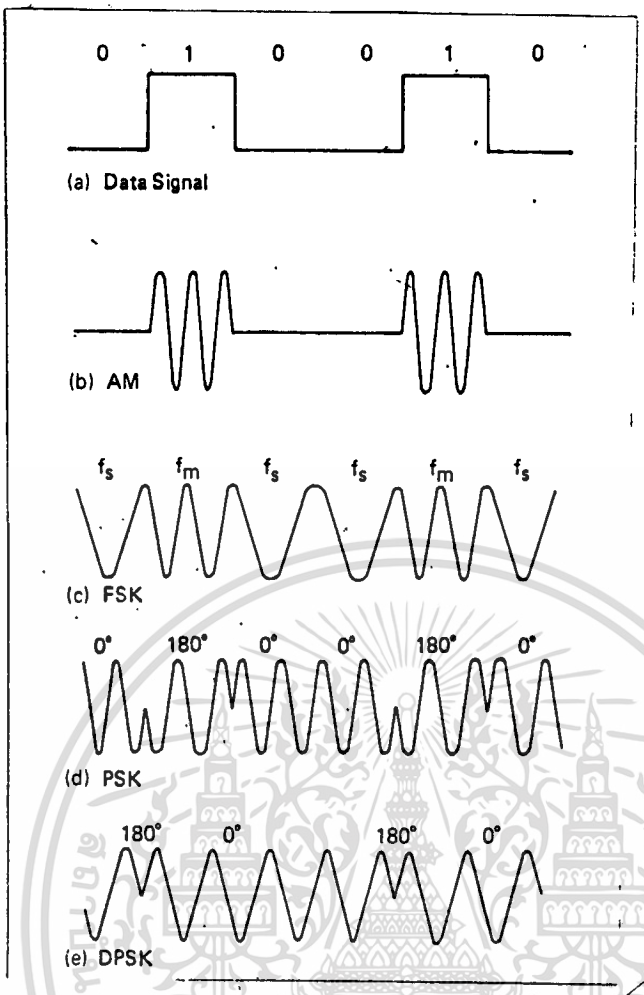
ความถี่หรือคลื่นพาหะถูกส่งระหว่างโมเด็มด้วยกันนั้น ก็จะถูกผสมสัญญาณในช่วงเวลาซึ่งมีการเปลี่ยนในทางด้านขนาด (AMPLITUDE), ความถี่ (FREQUENCY), มุม (PHASE) หรือการรวมของขนาดและมุมเข้าด้วยกัน ในการเปลี่ยนแปลงสัญญาณเหล่านี้ คือ สัญญาณตัวเลขฐานสอง ที่เป็นข้อมูลที่จะส่งออกไปนั่นเอง

การใช้วิธีการผสมสัญญาณแบบไหนนั้นขึ้นอยู่กับข้อมูล (DATA) ที่จะส่งไปว่ามากมายเพียงใด เช่น การผสมสัญญาณแบบ AM ถูกใช้เป็นแบบเงื่อนไขเปิด-ปิด (ON-OFF) ซึ่งความเร็วในการส่งสัญญาณอยู่ในช่วง Ø ถึง 5 b/s แบบ AM นี้จะไม่ใช้ส่งกัน เพราะว่าจะเกิดปัญหาการสูญเสียและสัญญาณรบกวนมีมาก

โมเด็มแบบอซิงค์โครนัส (ASYNCHRONOUS MODEMS) ที่ใช้ส่งแบบ FREQUENCY SHIFT KEYING (FSK) จะมีการส่งแบบ 2 ความถี่ คือ ความถี่หนึ่งให้เป็นโลจิก "1" (MARK) และอีกความถี่หนึ่งให้เป็นโลจิก "Ø" (SPACE)

โมเด็มแบบอซิงค์โครนัสที่ใช้ส่งแบบ PHASE SHIFT KEYING (PSK) ความถี่ที่ส่งจะไม่เปลี่ยนแปลง แต่จะเปลี่ยนแปลงทางด้านมุมที่ถูกเลื่อน (SHIFTED) ไป และสำหรับการส่งแบบ PSK นี้ ก็มีการเข้ารหัสให้เป็น 2 บิท ซึ่งเรียกว่า "ไดบิท" (DIBIT) ในการเปลี่ยนแปลงข้อมูลในเวลานั้นๆ ก็เข้ารหัสให้เป็น 2 บิท ดังนั้นสิ่งที่เกิดขึ้นคือ บิทเรท (BIT RATE) จะเป็น 2 เท่าของ 1 บอดเรท (BAUS RATE) เช่น จะเป็น 2 บิท ต่อบอด ซึ่งกรณีนี้เราใช้มุมไป 4 มุม ถ้าหากว่าเราใช้มุมไป 8 มุมแล้วแราก็สามารถนำแต่ละมุมแทนด้วยบิทได้ถึง 3 บิท ในการเปลี่ยนแปลงสัญญาณหนึ่งครั้ง

และแบบ PSK นี้จะแปลงเป็นแบบ DIFFERENTIAL PHASE SHIFT KEYING ก็ได้ โดยที่มุมที่เลื่อนนี้จะช้ากว่ามุมสมบูรณ์ (ABSOLUTE PHASE) ซึ่งจะเห็นได้จากรูปที่ 2



รูปที่ 2

<u>ชนิดของโมเด็ม (TYPES OF MODEMS</u>

โมเด็มที่ใช้กันทั่วๆ ไปนั้น จะมี มาตรฐาน อยู่ 2 แบบ คือ CITT และ BELL และเราจะพิจารณา มาตรฐานของ BELL เพราะว่าแบบ CITT ก็มีมาตรฐานใกล้เคียงกันด้วย มาตรฐานของ CITT จะบอกรหัสเป็น CITT.V.XX.

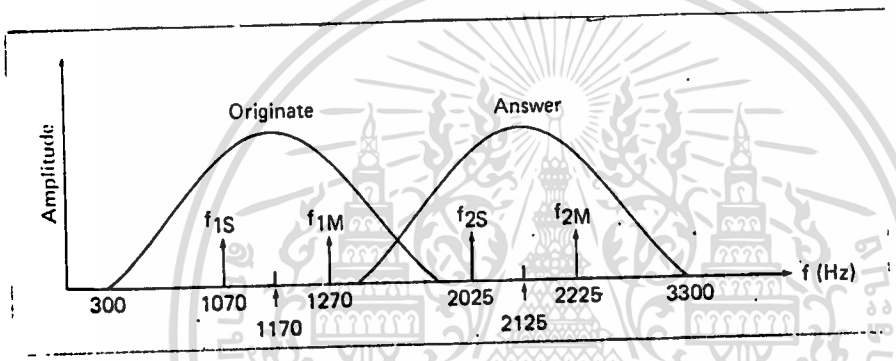
<u>แบบ BELL 103/113</u>

เป็นโมเด็มที่มีการทำงานเป็นแบบ ฟูล ดูเพล็กซ์ (FULL DUPLEX) ที่สามารถส่งในสายส่งแบบ 2เส้นได้ โดยที่แบนด์วิทของสัญญาณเสียงจะถูกแบ่งออกเป็น 2 ส่วน โดยจะเหมือนกันในการส่งแบบ 2 ทิศทาง คือจะส่งแบบ FSK ที่มีช่วงระหว่าง ความถี่มาร์ค (MARK) และสเปซ (SPACE) ห่างกัน 200 Hz. และการทำงานจะถูกจำกัดที่ 300 b/s ในการส่งแบบอซิงค์โครนัส

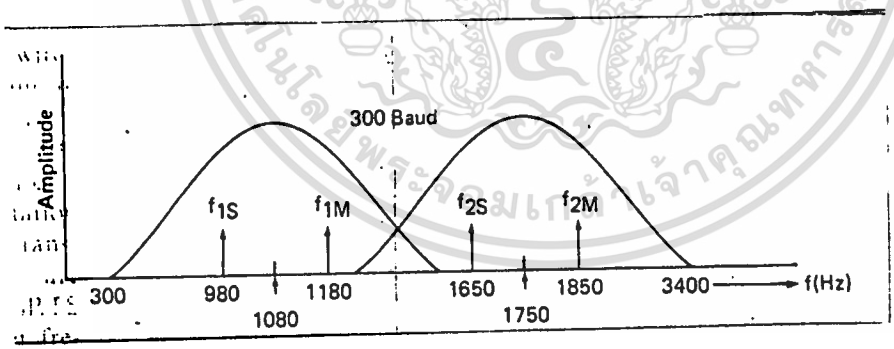
เมื่อโมเด็มเป็นฝ่ายเริ่มต้นส่ง ก็จะส่งด้วยความถี่ 1070/1270 Hz และ เมื่ออยู่ในโหมด (MODE) ของการรับ ก็จะส่งความถี่ที่ 2025/2225 Hz ความถี่ที่ต่ำจะถูกแทนด้วย SPACE (LOGIC LOW) และความถี่สูงกว่าจะถูกแทนด้วย MARK (LOGIC HIGH) ลักษณะรูปคลื่น ดังรูปที่ 3

และในมาตรฐานของ CCITT ที่คล้ายกับ BELL 103/113 คือ V.21 ลักษณะของช่องสัญญาณดังรูปที่ 4

แต่มีลักษณะแปลกกว่าของ BELL 103/113 คือ ในช่วงโลจิกสูง (MARK) จะเริ่มต้นให้เป็น 980 Hz หรือ 1650 Hz ก็ได้ และในโลจิกต่ำ (SPACE) จะให้เป็น 1180 หรือ 1850 Hz



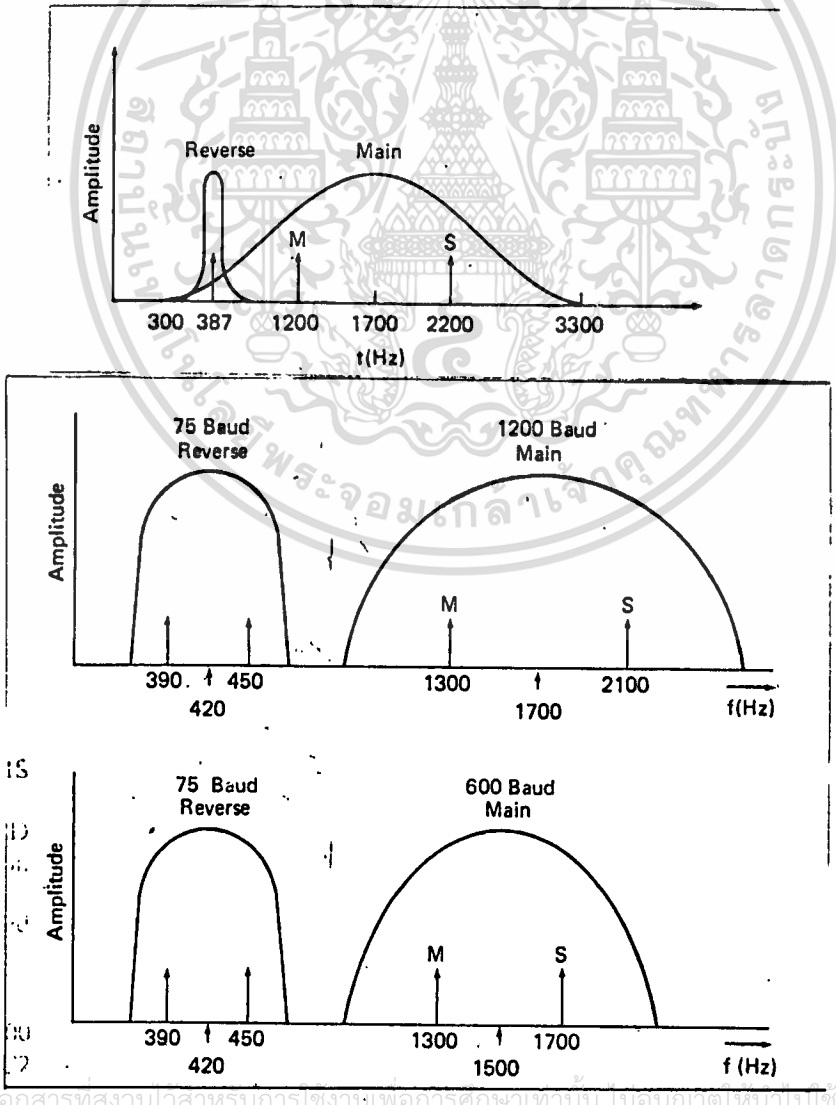
รูปที่ 3



รูปที่ 4

BELL 202

เป็นโมเด็มที่ใช้ส่งแบบ ฮาฟ ดูเพล็กซ์ [HALF DUPLEX] ออกแบบมาให้มีความเร็วที่ 1200b/s เป็นการผสมสัญญาณแบบ FSK ที่ซึ่งสัญญาณเสียง 1200Hz ให้เป็นโลจิกสูงและสัญญาณเสียง 2200Hz ให้เป็นโลจิกต่ำ และก็มีสัญญาณ RTS และ CTS ด้วย ซึ่งจะทำให้โมเด็ม อยู่ในรูปแบบเมื่อส่งและเมื่อรับ รูปที่ 5 แบบช่องสัญญาณของ HDX BELL 202 และในรูปที่ 6 เป็นช่องสัญญาณของ CITT V.23 ซึ่งมันจะมีช่องสัญญาณส่งข้อมูลกลับมา โดยที่ใช้บอดเรทต่ำ ๆ คือ 75 บอด ส่วนของ Bell 202 นั้นใช้ 150 บอดเรท มันมีประโยชน์ใช้ในการตรวจสอบข้อผิดพลาดของข้อมูล เช่น ถ้าสถานี A ส่งข้อมูล ไปยังสถานี B แล้วข้อมูลเกิดผิดพลาด สถานี B ก็ส่งสัญญาณ บอดเรทต่ำ ๆ นี้มายังสถานี A เพื่อบอกว่าการส่งมีความผิดพลาดเกิดขึ้น ซึ่งจะอธิบายกันอีกครั้งในการทดลอง -

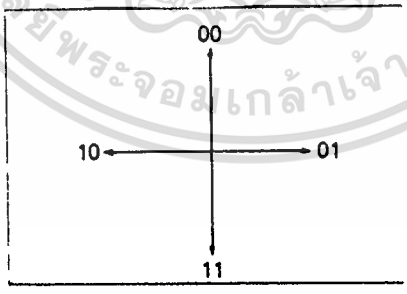


รูปที่5,6

BELL 212 A

เป็นโมเด็มที่ส่งความเร็วได้ สองอย่างคือ 0 ถึง 300b/s โดยใช้การผสมสัญญาณแบบ PSK ซึ่งมี สัญญาณคล้ายกับ Bell 103 และส่งได้ 1200b/s โดยใช้การผสมสัญญาณแบบ DPSK ในความเร็วต่ำความถี่ที่ใช้คือ 1070/1270Hz เป็นตัวส่งและความถี่ 2025/2225Hz เป็นตัวรับ ส่วนในความเร็วสูง ตัวส่งใช้ความถี่ 1200Hz ส่วนตัวรับใช้ความถี่ 2400Hz โมเด็ม Bell 212A ทำงานแบบ ฟูดูเพล็กซ์ บนสายมากกว่า 2 เส้นได้ ในโหมดของความเร็วสูง มันจะทำงานในแบบ บิท-ซิงค์โครนัส [Bit-Synchronous]หรือแบบตัวอักษร-อะซิงค์โครนัส [Character-asynchronous] ในแบบอะซิงค์โครนัส ความยาวของตัวอักษรเป็น 8,9,10 หรือ 11 บิท

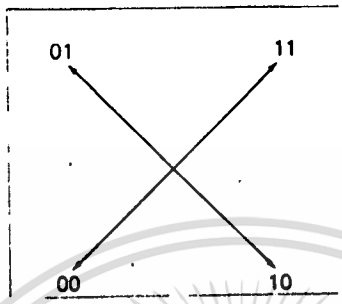
ในโหมดของความเร็วสูง สองบิทจะถูกเรียกว่า ไดบิท ซึ่งจะเกิดขึ้นในเวลาเดียวกันของการเปลี่ยนแปลงสัญญาณหนึ่งครั้ง Bell 212Aจะเลื่อนมุมไป 90 องศา สำหรับข้อมูล "00",มุม 0 องศา สำหรับข้อมูล "01" ,มุม 180 องศา สำหรับข้อมูล "10" และมุม 270 องศา สำหรับข้อมูล "11" ซึ่งแสดงได้ ดังรูปที่ 7



รูปที่ 7

BELL 201 B/C MODEM

เป็นโมเด็มแบบซิงค์โครนัส [Synchronus modem] ทำหน้าที่ 2400b/s ใช้การผสมสัญญาณแบบ DPSK ดังรูปที่ 8



รูปที่ 8

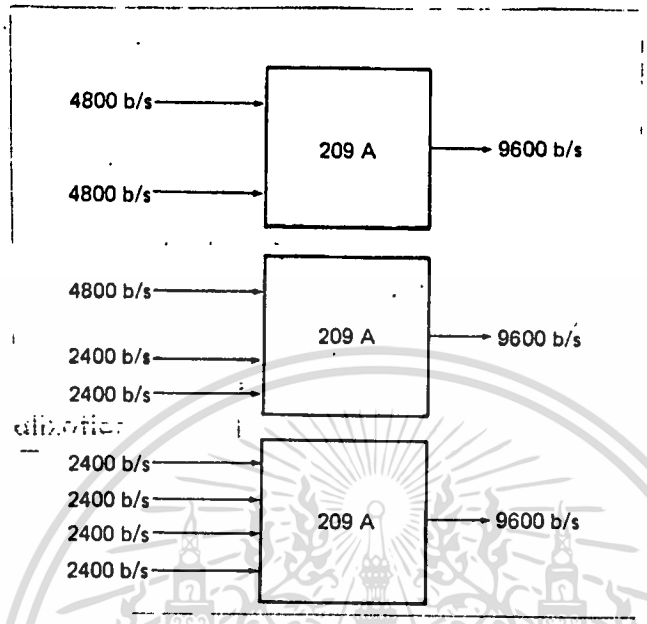
BELL 200 MODEM

เป็นโมเด็มแบบซิงค์โครนัส DPSK โมเด็ม ใช้มุมแปดมุมที่แตกต่างกัน โดยใช้สัญญาณพาหะ 1800Hz และ ทำหน้าที่ 2400 b/s และโดยที่บอดเรทคือ 1600 บอด

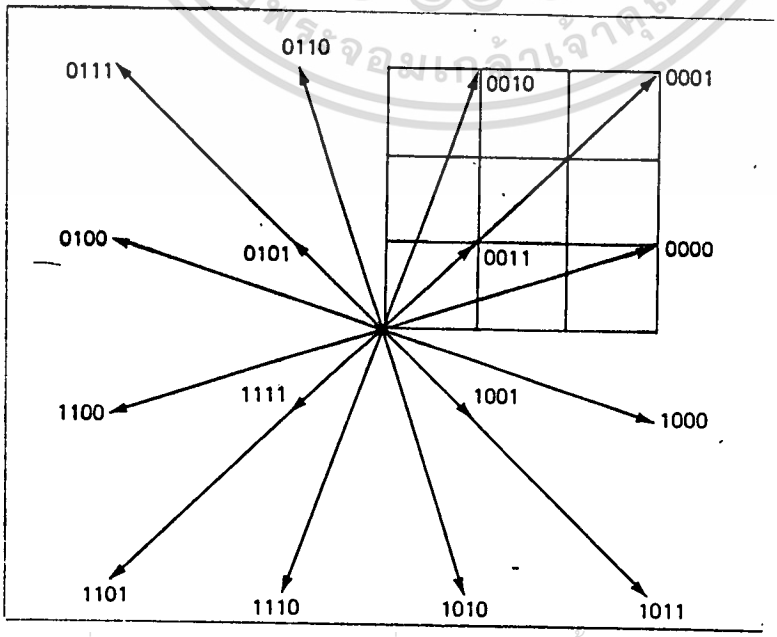
โมเด็มความเร็วสูง [HIGH SPEEF MODEM]

ประยุกต์ใช้แบบ PSK กับ AM เป็น[Quadrature Amplitude Modulation (QAM)] คือ นอก จากใช้การเลื่อนมุมแล้ว ยังใช้ขนาดมารวมกับการเลื่อนมุมด้วย ทำให้สามารถส่งสัญญาณ ดิจิตอล ได้ ครั้งละหลาย ๆ บิท ต่อการเปลี่ยนแปลงสัญญาณหนึ่งครั้ง

โมเด็ม Bell 209 เป็นโมเด็มแบบซิงค์โครนัส โมเด็มถูกออกแบบมาในการทำงานใช้สายมากกว่า 4 เส้น ลักษณะช่องสัญญาณ ดังรูปที่ 9- Bell 209 ใช้การ Mod สัญญาณแบบ QAM ดังรูปที่ 9 โดยที่มีมุมแตกต่างกัน 12 มุม และมีขนาดที่แตกต่างกันอีก 3 ขนาดสามารถส่งสัญญาณได้ถึง 9600b/s

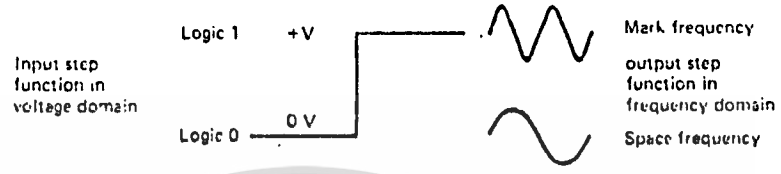


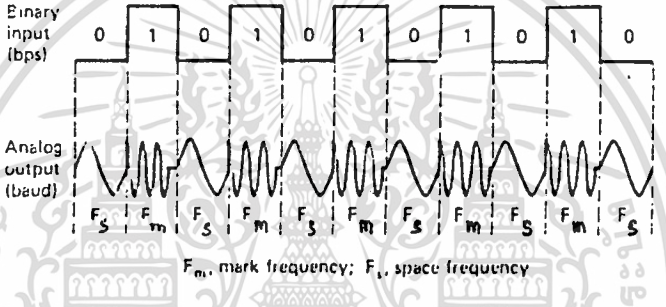
รูปที่ 9



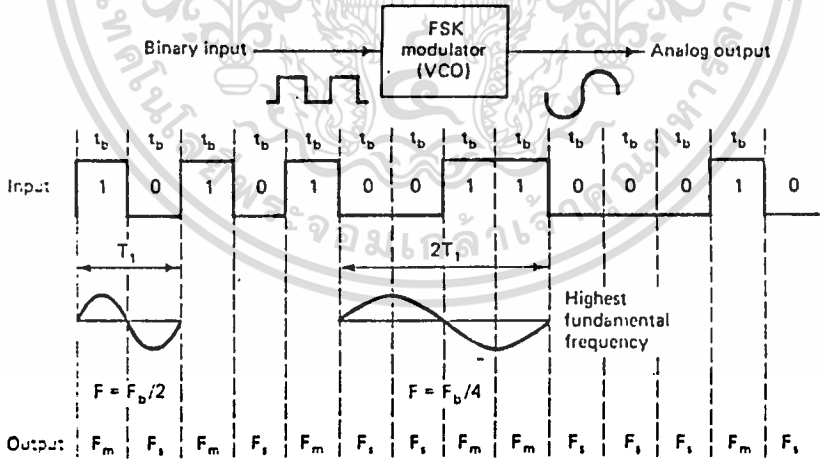
การส่งข้อมูลแบบ FSK

จากรูปที่ 10 จะเห็นว่าการส่งแบบ FSK นี้จะให้ลักษณะของข้อมูลที่ถูก Modulate แล้ว เป็น 2 ความถี่ด้วยกันคือ จะให้บิทข้อมูล 1 แทนด้วยความถี่สูง และ บิทข้อมูล 0 แทนด้วยความถี่ต่ำ หรือให้บิทข้อมูล 1 แทนด้วยความถี่ต่ำและบิทข้อมูล 0 แทนด้วยความถี่สูง ก็ได้





รูปที่ 10

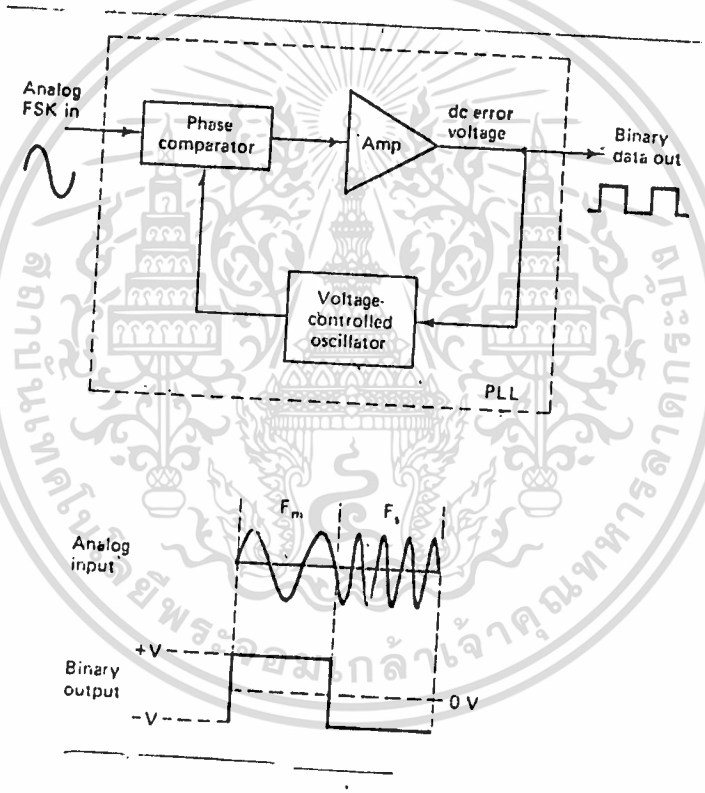


รูปที่ 11

จากรูปที่ 11 แสดงถึงบล็อคของการ Modulate แบบ FSK โดยที่การ Modulate แบบนี้จะเป็นลักษณะการส่งแบบ FM จึงถูกเรียกว่า โวลท์เตจคอลโทรล ออสซิลเลเตอร์ [Voltage control oscillator] หรือ Vco

การรับข้อมูลแบบ FSK

ในวงจรรับส่วนมากในการ Demoulate สัญญาณ FSK จะใช้วงจรเฟสล็อคลูป [Phase-locked loop] หรือ PLL ดังแสดงในรูป 12 การทำงาน PLL-FSK นี้จะมีลักษณะคล้ายกับการ Demodulate ของ PLL-FM

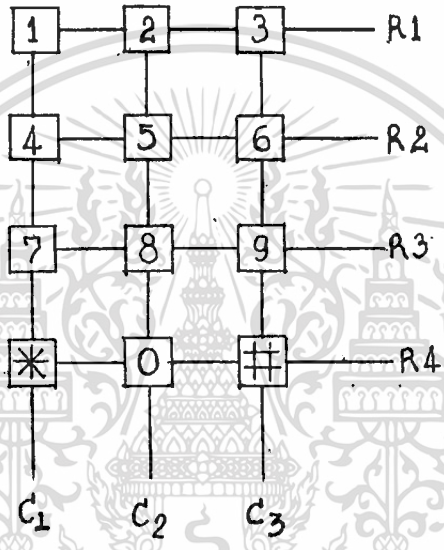


รูปที่ 12

บทที่ 3 วงจรช่วยกดปุ่มโทรศัพท์โดยอัตโนมัติ

3.1 การออกแบบคีย์ (KEY) ปุ่มกดของโทรศัพท์

ปกติแล้วคีย์ที่ใช้แบบกดปุ่มนั่นจัดเรียงกันเป็นแบบเมตตริกซ์ (Matrix) แต่ในที่นี้เราจะใช้คุณสมบัติของวงจรลอจิกเกท (Logic Gate Circuit) มาทำปุ่มกดของโทรศัพท์ ซึ่งการจำลองคีย์กดเบอร์โทรศัพท์จะมีลักษณะดังรูปที่ 3.1.1



รูปที่ 3.1.1 คีย์ที่ใช้กันทั่วไป

จากรูป ปกติแล้วขา Rx จะมีสถานะเป็นลอจิกหนึ่ง (Logic "1") และขา Cx เป็นลอจิก 0 (Logic "0") และขาเหล่านี้ถ้าเรานำไปต่อเข้ากับของไอซี (IC) เบอร์ TCM 5087 ซึ่งเป็นไอซีแบบ DUAL TONE MULTI-FREQUENCY (DTMF) ก็สามารถนำไปต่อใช้งานได้ทันที แต่ในที่นี้จะออกแบบจำลองคีย์แบบนี้โดยใช้วงจรลอจิก

ขาโลว์และขาคอลัมน์ (Low,Column) จะมีเอาท์พุท (Out Put) ออกมาเป็นความถี่ ดังนี้

f1 [R1] = 697 Hz f2 [R2] = 770 Hz f3 [R3] = 852 Hz

f4 [R4] = 941 Hz f5 [C1] =1209 Hz f6 [C2] =1336 Hz

f7 [C3] =1477 Hz *f8 [C4] =1633 Hz * ค่านี้ไม่ได้ใช้ในการออกแบบ

ค่าความถี่ต่าง ๆ เป็นเอาท์พุทแบบมาตรฐานของ DIMF แต่ถ้าของ ไอซีเบอร์ TCM 5087 นั้นก็มีความถี่แตกต่างกันไม่มากนัก ในการส่งแบบความถี่คู่ (Dual-Tone) จะต้องมีความถี่ 2 ความถี่ผสมกัน คือ ทางด้านความถี่สูงและความถี่ต่ำ ดังตารางข้างล่างนี้

สัญลักษณ์	ความถี่ต่ำ (Hz)	ความถี่สูง (Hz)
1	(R1) 701.3	(C1) 1215.9
2	(R1) 701.3	(C2) 1331.9
3	(R1) 701.3	(C3) 1477.9
4	(R2) 771.4	(C1) 1215.9
5	(R2) 771.4	(C2) 1331.7
6	(R2) 771.4	(C3) 1471.9
7	(R3) 857.2	(C1) 1215.9
8	(R3) 857.2	(C2) 1331.7
9	(R3) 857.2	(C3) 1471.9
0	(R4) 935.1	(C2) 1331.7
*	(R4) 935.1	(C1) 1251.9
#	(R4) 935.1	(C5) 1471.9

ตารางที่ 3.1.1

ความถี่นี้เป็นความถี่ที่ออกมาจากเอาท์พุทจริง ๆ ของไอซี TCM 5087 ซึ่งเราจะนำค่านี้ไปออกแบบวงจร

DCBA	No./Row.,Col.	R1	R2	R3	R4	C1	C2	C3
0001	1	0	1	1	1	1	0	0
0010	2	0	1	1	1	0	1	0
0011	3	0	1	1	1	0	0	1
0100	4	1	0	1	1	1	0	0
0101	5	1	0	1	1	0	1	0
0110	6	1	0	1	1	0	0	1
0111	7	1	1	0	1	1	0	0
1000	8	1	1	0	1	0	1	0
1001	9	1	1	0	1	0	0	1
0000	0	1	1	1	0	0	1	0
1010	*	1	1	1	0	1	0	0
1011	#	1	1	1	0	0	0	1

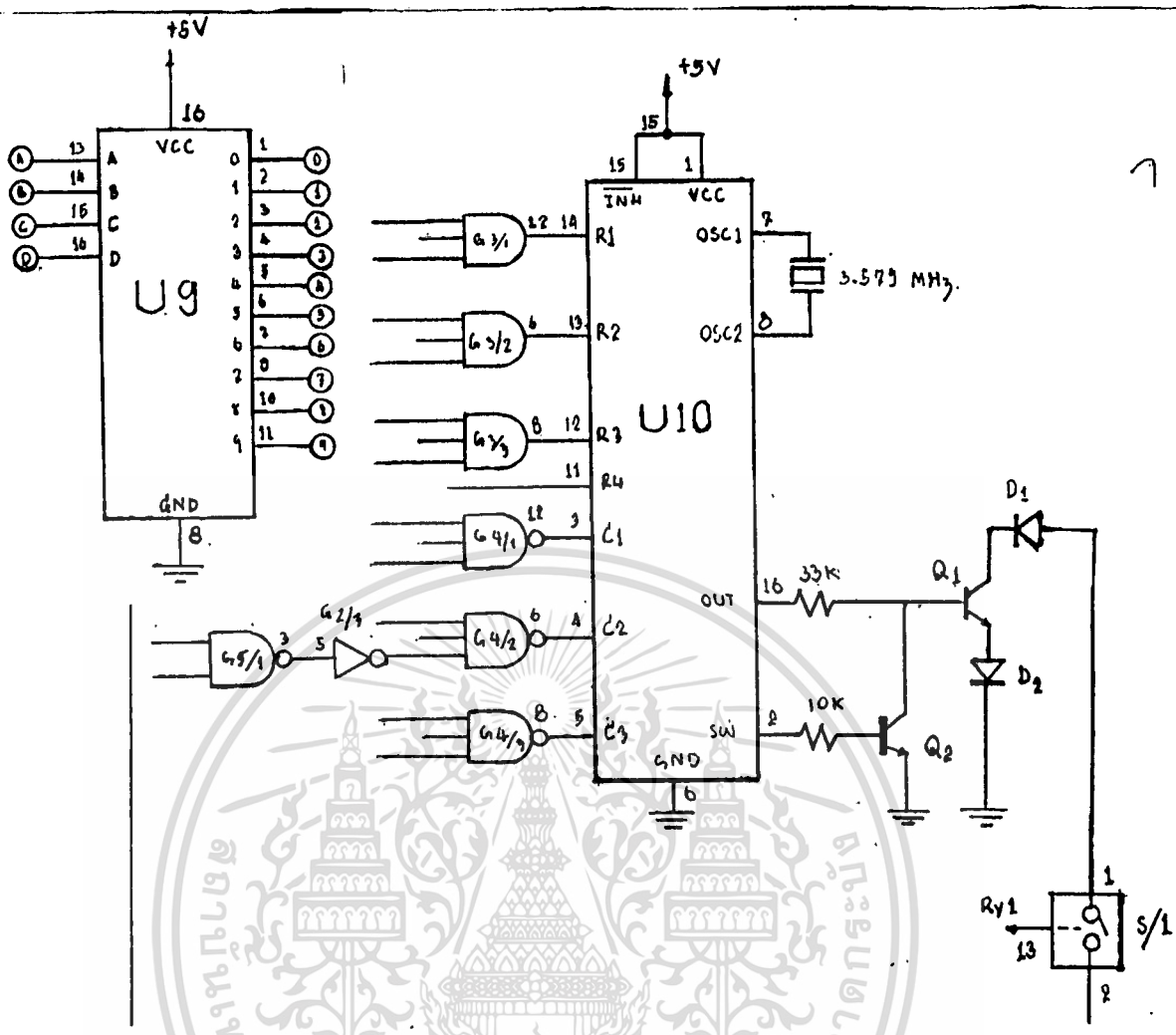
รูปที่ 3.1.2

ค่าที่ได้เราพิจารณาจากรูปที่ 3.1.2 คือ

$R1 = 1.2.3$ $R2 = \overline{\overline{4.5.6}}$ $R3 = 7.8.9$ $R4 = 0$

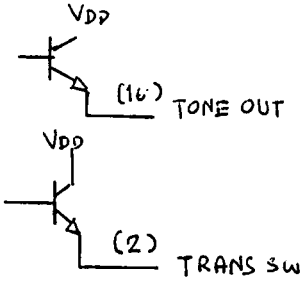
$C1 = \overline{1.4.7}$ $C2 = 2.5.\overline{\overline{(8.0)}}$ $C3 = \overline{3.6.9}$

ส่วนการส่งหมายเลข 0-9 เราใช้ไอซีเบอร์ 74LS42 ซึ่งเป็นไอซี ไบนารี-เดซิมอล-ดีโค้ด (Binary-Decimal Decode) ซึ่งดีโค้ดจาก 4 ไป 16 แต่ในที่นี้ใช้แค่ 0-9 และเอาท์พุทที่ทำงานก็คือ ลอจิก 0 ซึ่งก็ตรงกับ ตารางลอจิก (Truth Table) ที่ออกแบบพอดี จากหลักการออกแบบจะเห็นว่าเราใช้ ลอจิก0 เป็นตัวกำหนดหมายเลข ดังนั้น เวลาถอดรหัสบลูลีน (Boolean) จึงต้องใส่บาร์อีกตัวหนึ่ง เช่น $R1 = \overline{\overline{1+2+3}} = 1.2.3$ ถ้าหากว่าหมายเลขนั้นเราให้ทำงานที่ลอจิกหนึ่งแล้ว R1 ก็จะมีค่าเป็น $\bar{1}.\bar{2}.\bar{3}$ แทน



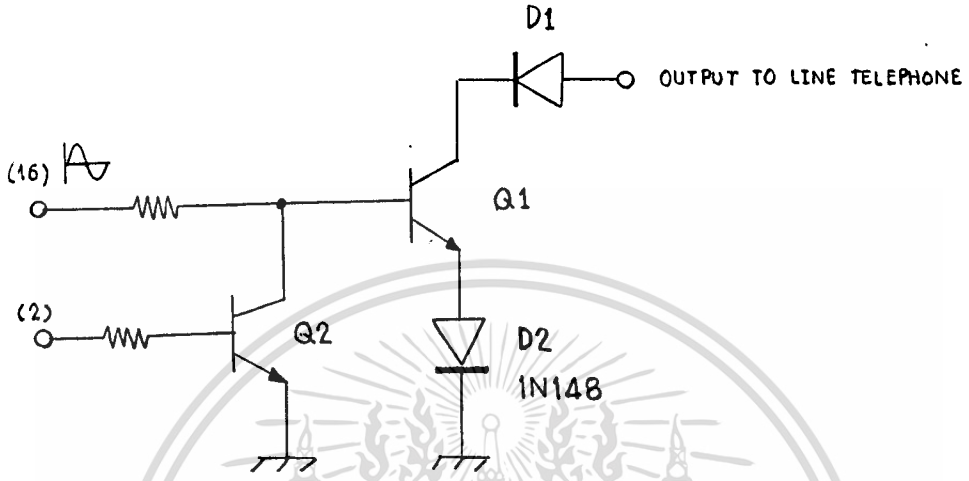
รูปที่ 3.1.3 วงจรแทนปุ่มกดโทรศัพท์

ต่อไป เมื่อเราสามารถออกแบบให้วงจรเราเลียนแบบปุ่มกดโทรศัพท์ได้แล้ว ก็ต้องมีสัญญาณเอาท์พุทที่จะออกไปทางสายโทรศัพท์ ซึ่งสัญญาณเอาท์พุทของ ไอซี เบอร์TCM5087 มีสัญญาณอยู่ 2 สัญญาณ คือ สัญญาณ OUT ขา 16 และสัญญาณ SW ขา 2 จากการสังเกตจะเห็นได้ว่าลักษณะภายในไอซีจะเป็นดังนี้



รูปที่ 3.1.4

ขา 2 จะถูกเตรียมการสำหรับในขณะที่ไม่มีการกดคีย์ เมื่อมีการกดคีย์หรือสัญญาณหมายเลขที่เราส่งมา เอาท์พุทจะเปิดวงจร (Open Circuit) ถ้าไม่มีการกดคีย์ก็จะมีกระแสไหลผ่านทรานซิสเตอร์ได้



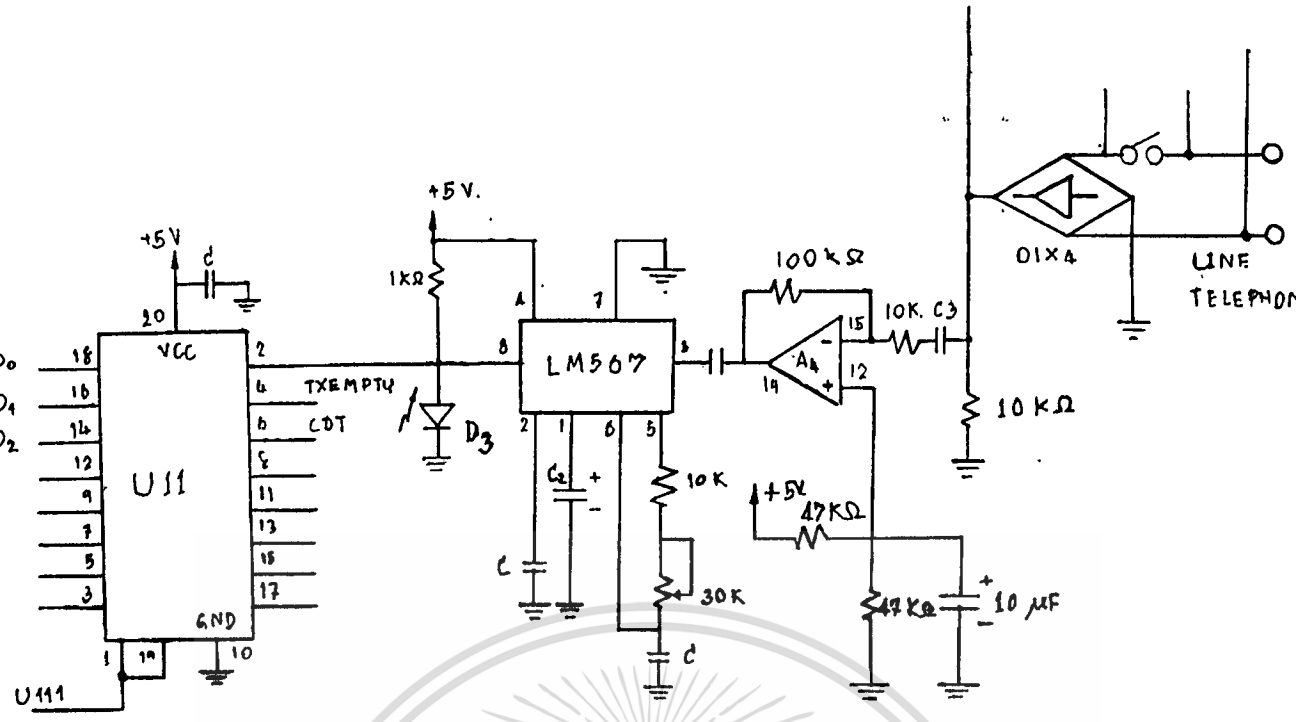
จากรูป D1 จะกั้นไฟลบจากสายโทรศัพท์ ส่วน D2 เป็นตัวฟิกซ์ไบแอส (Fixed Bias) ทำให้ Q1 ทำงานที่โวลท์เตจ ประมาณ 1.4 V.ขึ้นไป ส่วน Q2 เป็นตัวควบคุมการทำงานของเอาท์พุท คือ ในสภาวะที่ไม่มีการกดคีย์เข้ามา ขา 2 ของ Q2 ก็มีไบแอสเข้ามาทำให้มันนำกระแสซึ่งก็ไปออฟ (Off) ไม่ให้ Q1 ทำงาน แต่เมื่อมีการกดคีย์เข้ามา Q2 ก็ออฟ (Off)

3.2 วงจรตรวจ (CHECK) สัญญาณโทรศัพท์

วงจรส่วนนี้ทำหน้าที่หลายอย่างคือ

[1] เมื่อเป็นฝ่ายส่งจะตรวจสัญญาณ

- ไดอัล โทน (Dial Tone)
- บิวซี โทน (Busy Tone)
- ริง แบค โทน (Ring Back Tone)



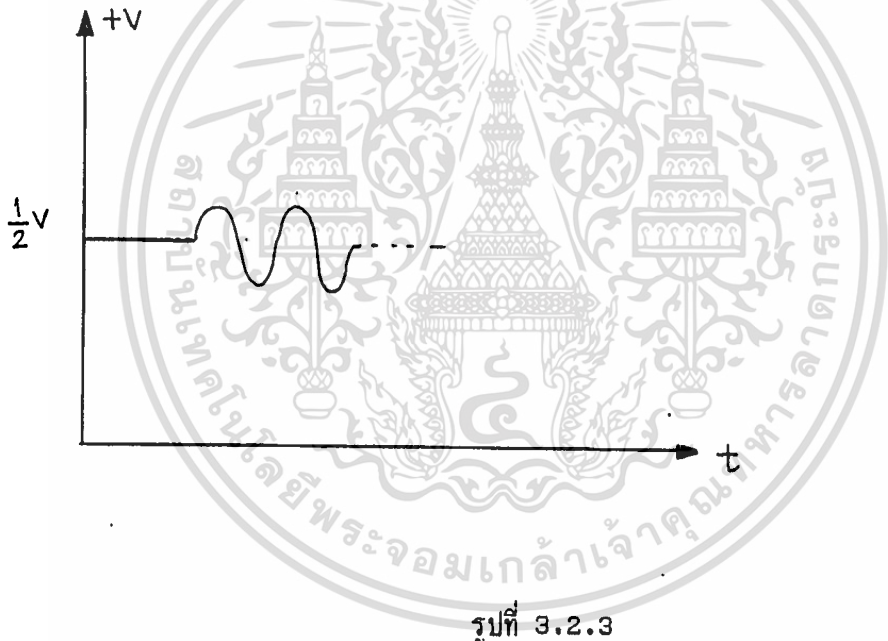
รูปที่ 3.2.1

[2] เมื่อเป็นฝ่ายรับจะตรวจสัญญาณผิดพลาด ซึ่งมีค่าประมาณ 487 Hz ซึ่งใช้วงจรเฟสล็อค ลูป (Phase Lock Loop) ใช้ไอซี เบอร์LM567 ซึ่งมีย่านความถี่กว้าง โดยปรับ R30k และ Voltage ที่เข้ามาทางขา 3 มีแรงดันสูง มันก็จะทำให้แบนด์วิธ (Band Width) กว้างขึ้น โดยที่ Fo = 1/R1C1 ซึ่งในที่นี้ใช้ R1 = 10k + VR30k ไว้ปรับค่า และใช้ C1 = 0.1uf โดยที่คำนวณใช้ fØ = 400Hz และกำหนด C1 ก่อน ดังนั้นจะได้ R1 = 1/(400x0.1uf) = 25k ซึ่งค่านี้จะใช้ VR 30k + (ความต้านทาน 10k)มาใช้งาน และในการใช้ตรวจสอบความผิดพลาดนั้น ค่าความถี่ประมาณ .487Hz ซึ่งค่า R1 ประมาณ 20k แต่ในที่นี้เราจะทำ BAND WIDTH ให้กว้าง โดยที่ให้ Vi สูง ซึ่งมีค่าประมาณไม่เกิน 4V. โดยใช้ ออปแอมป์ มาช่วยขยายสัญญาณจากสัญญาณโทรศัพท์และได้ใช้ R10k ต่อที่ขั้ว DIODE เพื่อ drop สัญญาณตรงช่วงนี้ ส่วน C2 นั้น เป็นโลว์ พาส ฟิวเตอร์ (Low Pass Filter) กำหนดให้เป็น 1uf และ C3 = 2C2 ดังนั้น

C3 = 2.2uf

ก่อนอื่นขอกล่าวถึง IC เบอร์TL094 ก่อน ซึ่งเบอร์นี้ในขณะที่จัดทำโครงการได้ไปขอข้อมูลจากผู้จำหน่าย แต่ไม่มีข้อมูลให้ จึงคาดว่าเป็นแบบ JFET LOW NOISE แต่ช่วงไฟเลี้ยงคงมีคุณสมบัติเหมือน LM324 ซึ่งเป็นแบบ G.P. AMPLIFIER คุณสมบัติอันนั้นคือ ไฟเลี้ยง เพราะเบอร์ LM324 ใช้งานในช่วง 1.5- 16 volts และคุณสมบัติทางด้านขาก็เหมือนกัน แต่ประสิทธิภาพการใช้งานของ ไอซี เบอร์TL094 ดีกว่า ไอซี เบอร์LM324 มาก เพราะ ไอซี เบอร์TL094 จะไม่มีสัญญานรบกวน (noise) เลยในการทดลอง แต่ไอซี เบอร์LM324 จะมีสัญญานรบกวนอยู่บ้าง

ลักษณะของวงจร Single supply เป็นดังนี้



รูปที่ 3.2.3

จะเห็นว่าลักษณะของรูปจะเหมือนกับการขยายของ คลาสเอ(Class A) โดยที่สัญญาณ AC จะทำให้แรงดันไฟ DC ของ 1/2V. เปลี่ยนแปลงตามสัญญาณ AC.

ในการใช้งานเราใช้งานแบบการขยายแบบกลับเฟส (Inverting Amplifier) ดังรูป โดยที่ ที่ขาอินพุทแบบการขยายแบบไม่กลับเฟส (Non Inverting) (+) จะต่อความต้านทาน (Resister) แบบแบ่งแรงดัน (Voltage Devider) ให้มีค่าเป็น +V/2 ส่วน C1 และ C2 นั้นเป็นคับเปิล คาปาซิเตอร์ (Couple Capcitor) สำหรับคับเปิลสัญญาณ โดยเป็นตัวกั้นแรงไฟ DC ด้วย ดังนั้น

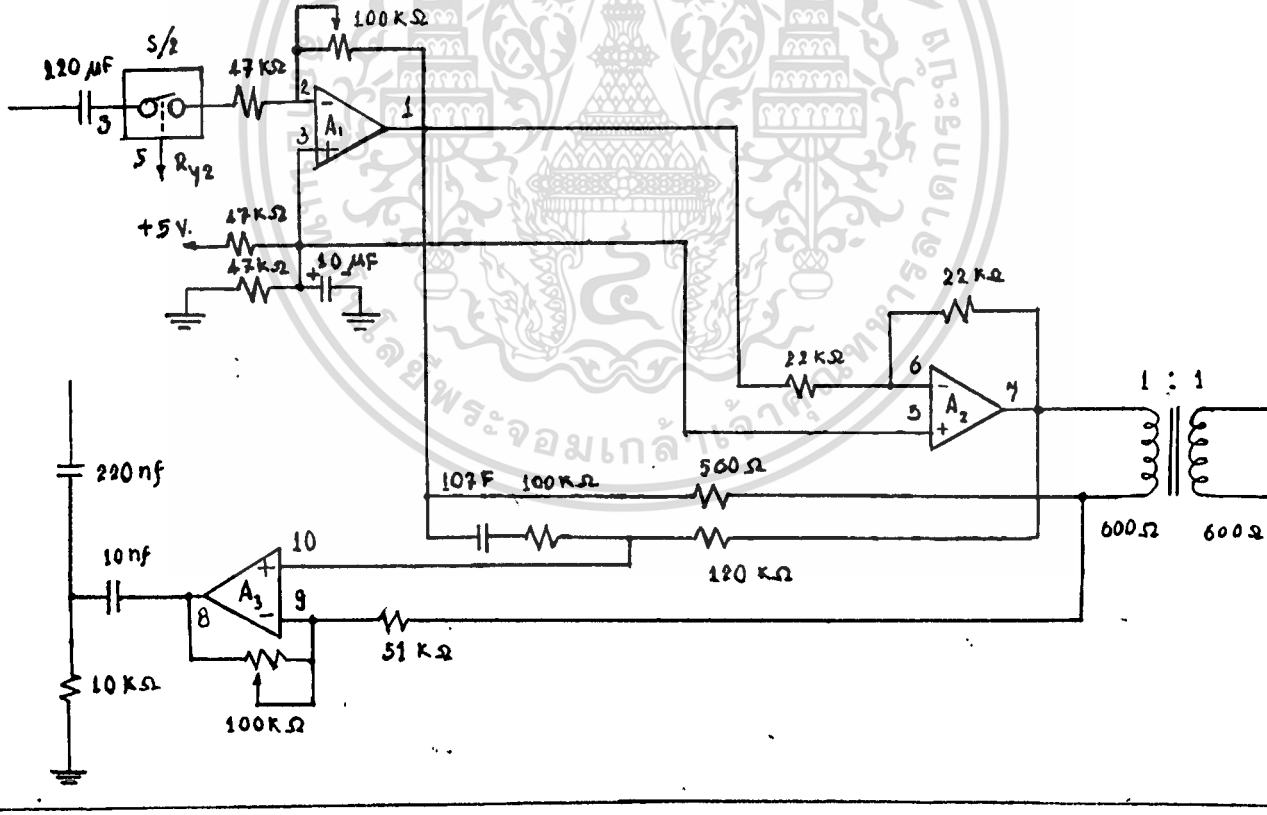
สรุป R1 = 10k-30k

C1 = 0.1uf

C2 = 1uf

C3 = 2.2uf

ส่วนออปแอมป์มที่ใช้นั้นได้ใช้ เบอร์TL094 ซึ่งนำมาใช้ทำเป็นแบบ SINGLE SUPPLY เพื่อว่าให้ใช้กับไฟเลี้ยง 5V. ได้อย่างไม่ยุ่งยากนัก โดยที่ขา 12 (+) ต้องมี VOLTAGE DEVICE ให้กับ OP-AMP ประมาณครึ่งหนึ่งของไฟเลี้ยง ซึ่งจะทำงานแบบ class A โดยที่ ถ้าปกติไม่มีสัญญาณใด ๆ OUT PUT จะมีค่าประมาณครึ่งหนึ่งของไฟเลี้ยง ถ้ามี INPUT เข้ามามันก็ SWING VOLTAGE ในช่วงไฟ +5V. เท่านั้น



รูปที่ 3.2.2

อัตราขยายทางด้านแรงดัน (Voltage Gain) = Vo/Vi = -R2/R1

และ (Vo)dc = (R4*(+V))/(R3+R4) = +V/2

ส่วนค่าของ C1 และ C2 เป็นตัวตอบสนองความถี่ต่ำ (Low-Frequency Response) จะได้ค่า C1 และC2 ดังนี้

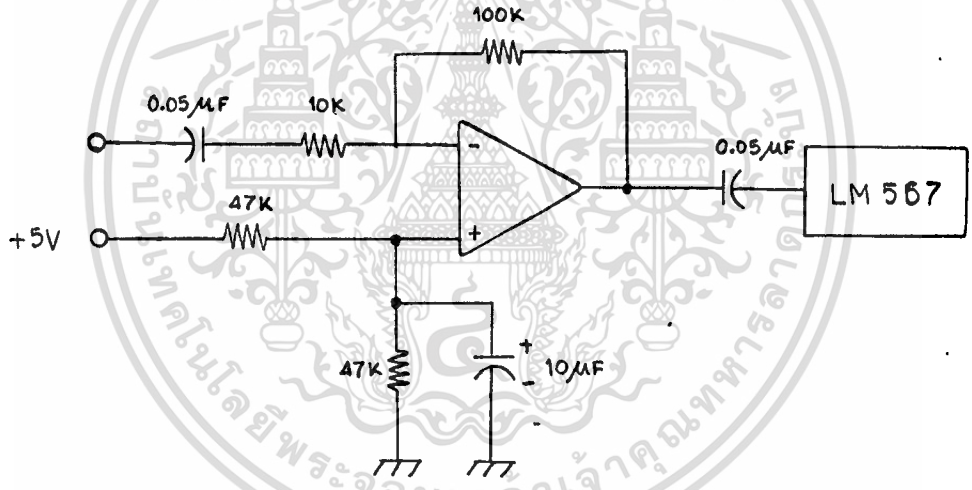
C1 = 1/[2*(3.1414)*fc*R1] , C2 = 1/[2*(3.1414)*fc*R2]

ถ้าให้ R1 = 10k และ fc = 400Hz

C1 = 1/[2*(3.1414)*(400Hz)*(10k)] = 0.03uf และในที่นี้ใช้ค่า 0.05uf

เพราะฉะนั้น C1 = 0.05uf

C2 = 0.05uf



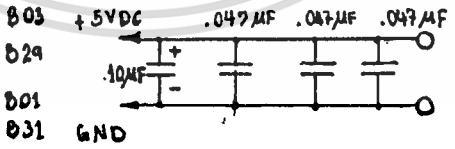
รูปที่ 3.2.3

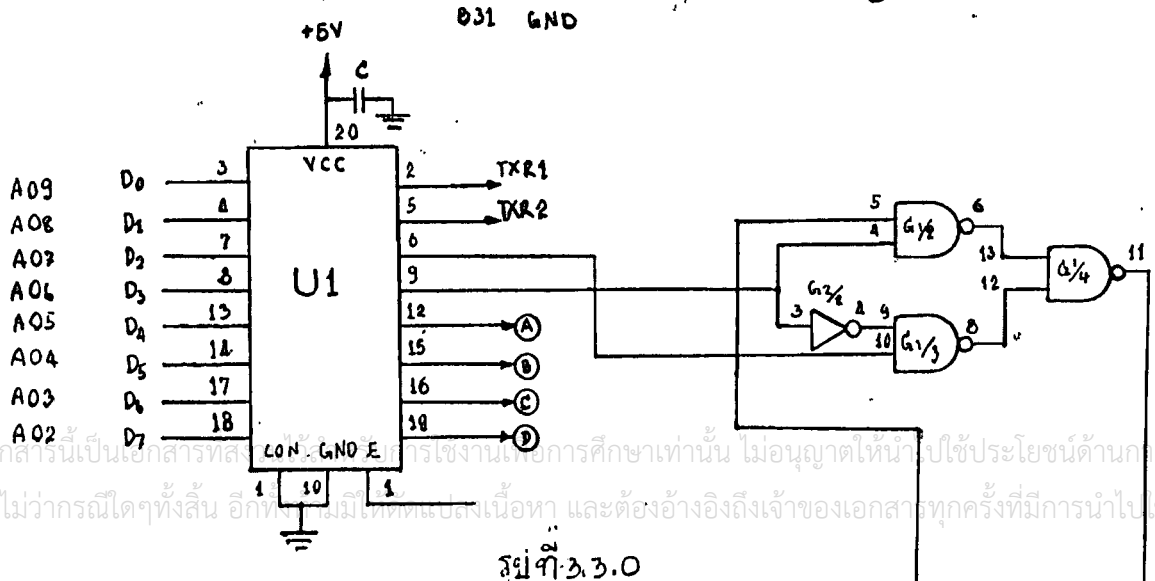
3.3 การออกแบบส่วนควบคุมอัตราการส่งข้อมูล (BAUD RATE)

การออกแบบส่วนนี้จะเห็นในรูปของตารางที่ 3.3.1 ได้ว่ามีเงื่อนไขเกี่ยวกับการกำหนด บอดเรท (BAUD RATE) อยู่ไม่มากนักคือ แบบมาตรฐาน CCITT V23 และ BELL 202 ดังนั้นจึงนำเงื่อนไขเหล่านี้มาเข้าตาราง เพื่อจะออกแบบให้มี OUT PUT ออกมาควบคุม คือขา TRS ซึ่งเป็นตัวแปรสำคัญด้วยและมีเงื่อนไข TXR1 และ TXR2 มาประกอบด้วย

Z	Y	X	A
CLK	D3	D2	TRS
Ø	Ø	Ø	Ø
Ø	Ø	1	1
Ø	1	Ø	Ø
Ø	1	1	Ø
1	Ø	Ø	Ø
1	Ø	1	1
1	1	1	1

ตารางที่ 3.3.1

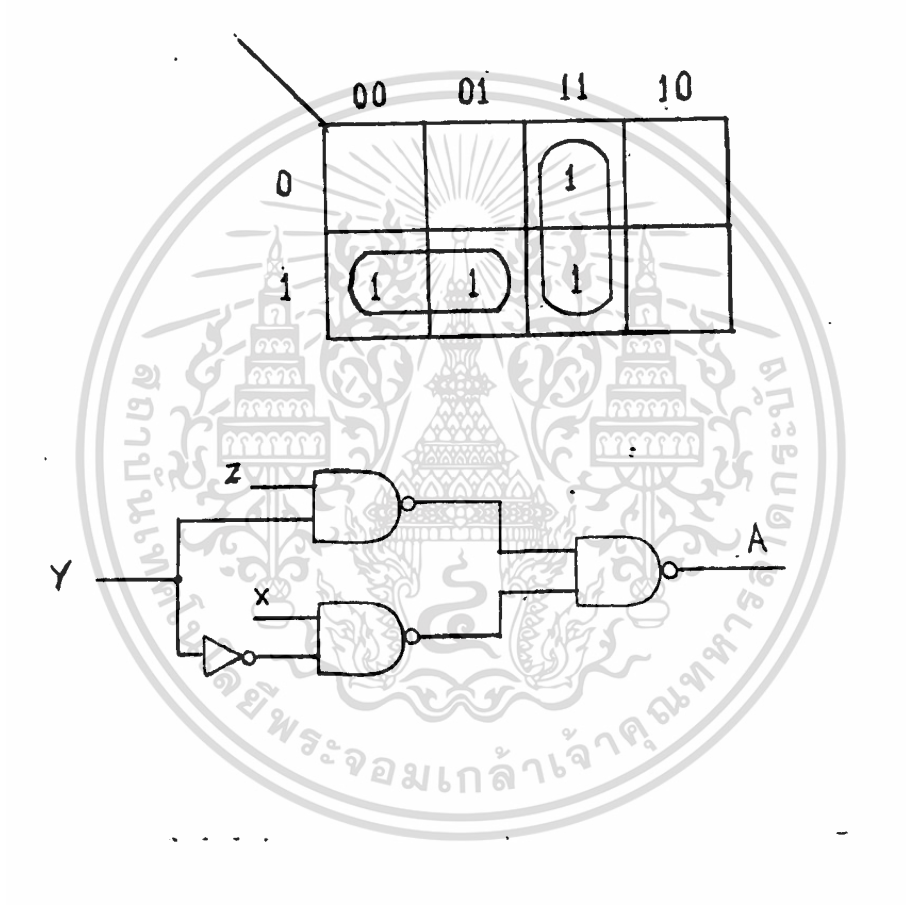




รูปที่ 3.3.0

หลังจากได้พิจารณาค่าต่าง ๆ และ หน้าที่ การทำงานในตารางแล้ว จะเห็นว่ามีตัวแปรอีกตัวหนึ่งคือ สัญญาณนาฬิกา (Clock) ของตัว TCM 3101 มาประกอบด้วยดังนั้นจึงจัดตารางใหม่เพื่อหา TRUTH TABLE ให้ได้วงจรออกมา

ดังตารางข้างล่าง และ TRUTH TABLE

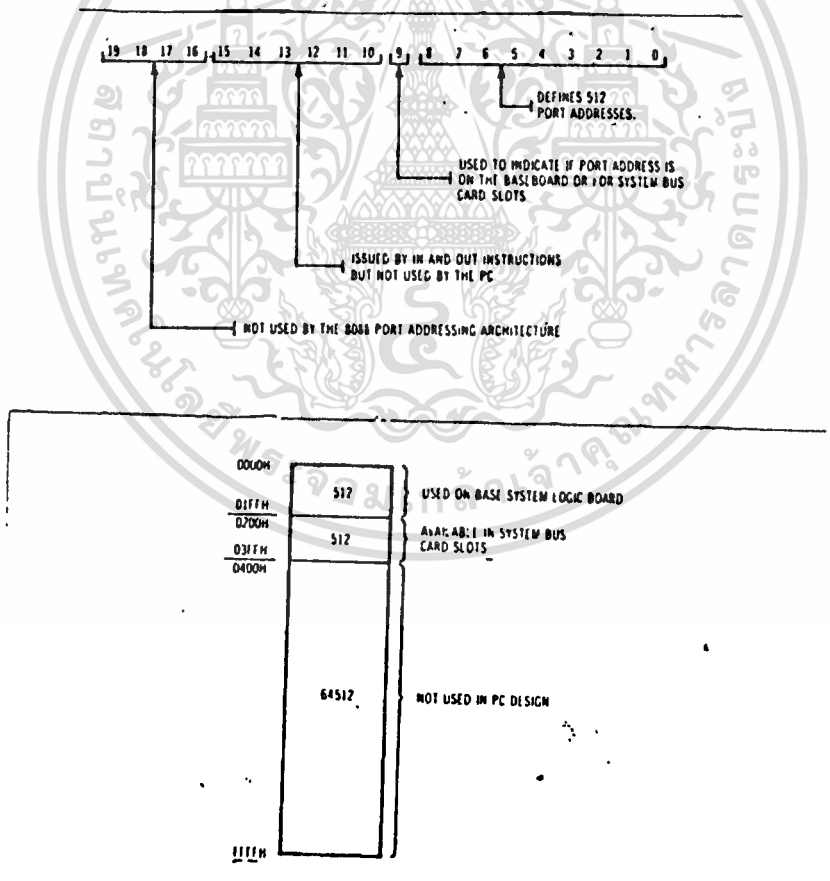


รูปที่ 3.3.1

ซึ่งหลังจากออกแบบแล้วจึงได้วงจรดังรูป 3.3.1 ก็ได้ค่าข้อมูลที่สามารถนำไปควบคุมพอร์ท (PORT) ได้ดังนี้คือ ค่า X8 และ X6 ซึ่งหน้าที่ของมันให้ดูในตารางที่ 3.3.1 ได้

3.4 การถอดรหัสตำแหน่งพอร์ทต่างๆ (DECODE ADDRESS PORT)

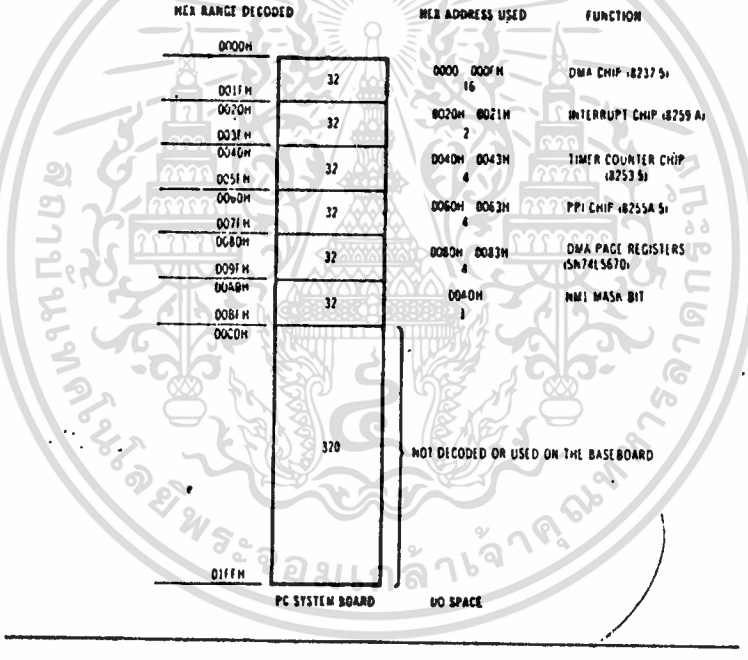
ก่อนอื่นมาทำความเข้าใจเกี่ยวกับการอ้างถึงพอร์ทต่างๆ (PORT) ต่างๆในเครื่องไอบีเอ็ม (IBM) ก่อน ปกติแล้ว 8088 โปรเซสเซอร์ (8088 Processor) จะมีการอ้างพอร์ทได้ถึง 65,536 แอดเดรส (65,536 Address) แต่ในเครื่องพีซี (PC) มีการออกแบบโดยใช้ไม่ถึง คือเพียง 10 บิท (Bit) ต่ำของแอดเดรสทั้งหมด ดังนั้น บิทที่ใช้คือบิท 0 และบิท 9 สามารถนำมาถอดรหัส (Decode) ได้ โดยที่ ถ้าบิท 9 มีค่าเป็นลอจิก "0" แล้ว จะเป็นพื้นที่ ที่พีซี ใช้งานเองบนบอร์ดของพีซี ซึ่ง สามารถแอดเดรสตำแหน่งถึง 512 พอร์ท และถ้าบิทมีลอจิกเป็น "1" แล้ว ก็หมายถึงพีซี เตรียมการใช้สำหรับสลอต 5 สลอตของพีซี ซึ่งมีการอ้างถึงแอดเดรสพอร์ทถึง 512 พอร์ทเหมือนกัน ดังรูป 3.4



รูปที่ 3.4

3.5 บนเบสบอร์ด (BASE BOARD)

เครื่องไอบีเอ็ม พีซี ได้ออกแบบสำหรับแอดเดรสพอร์ทจากตำแหน่ง ๐๐๐๐H ถึง ๐1FFH ซึ่งในช่วงแอดเดรสเหล่านี้ยังแบ่งแอดเดรสอุปกรณ์ต่างๆ อีกหลายประเภท ดังรูปที่ 3.5

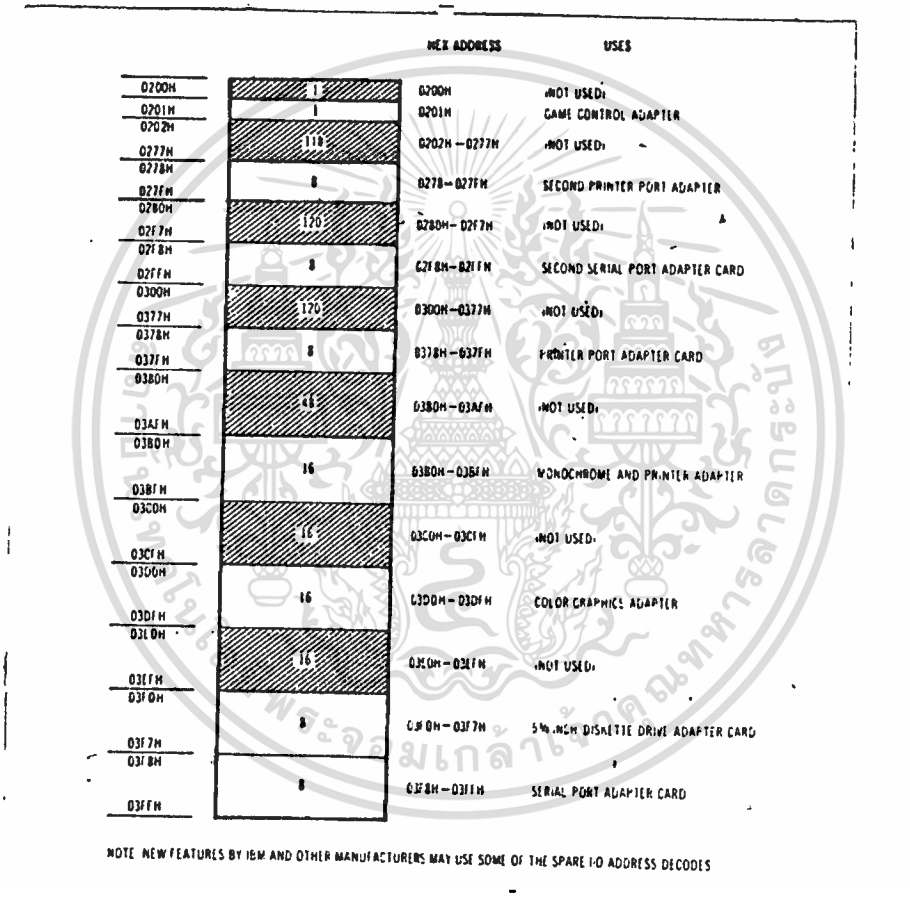


Baseboard I/O port address usage.

รูปที่ 3.5

3.6 บนคาร์ด สลอต (CARD SLOT)

ดังรูปที่ 3.6 จะมีช่องแอดเดรสจาก Ø200H ถึง Ø2FFH แอดเดรสนี้จะมีการจัดไว้สำหรับคาร์ดต่างๆ ที่นำมา เสียบบนสลอต แต่ก็ยังมี แอดเดรสว่างที่สามารถนำมาใช้งานในการทำโปรเจค (PROJECT) ได้คือเอาช่วงแอดเดรส Ø3EOH ถึง Ø3EFH นำไปใช้งาน



NOTE NEW FEATURES BY IBM AND OTHER MANUFACTURERS MAY USE SOME OF THE SPARE I-O ADDRESS DECODES

รูปที่ 3.6

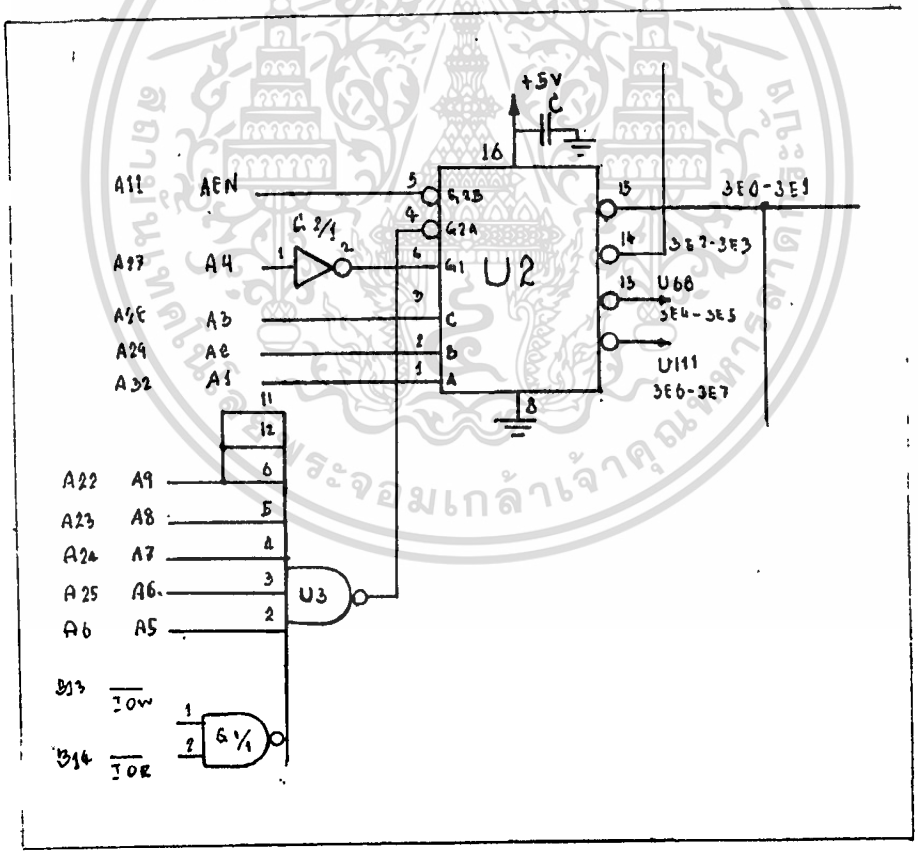
3.7 การถอดรหัสเพื่อนำไปแอดเดรสพอร์ท

นำเอาแอดเดรสที่ว่างอยู่คือช่วง Ø3EØH ถึง Ø3EFH ไปใช้งาน แต่ในโปรเจคนี้ใช้งานแค่ 4 พอร์ท แต่แอดเดรสไป 8 ตำแหน่ง ดังรูป 3.7

A9	A8	A7	A6	A5	A4	A3	A2	A1	AØ
1	1	1	1	1	Ø	X	X	X	X
3				E				X	

รูปที่ 3.7

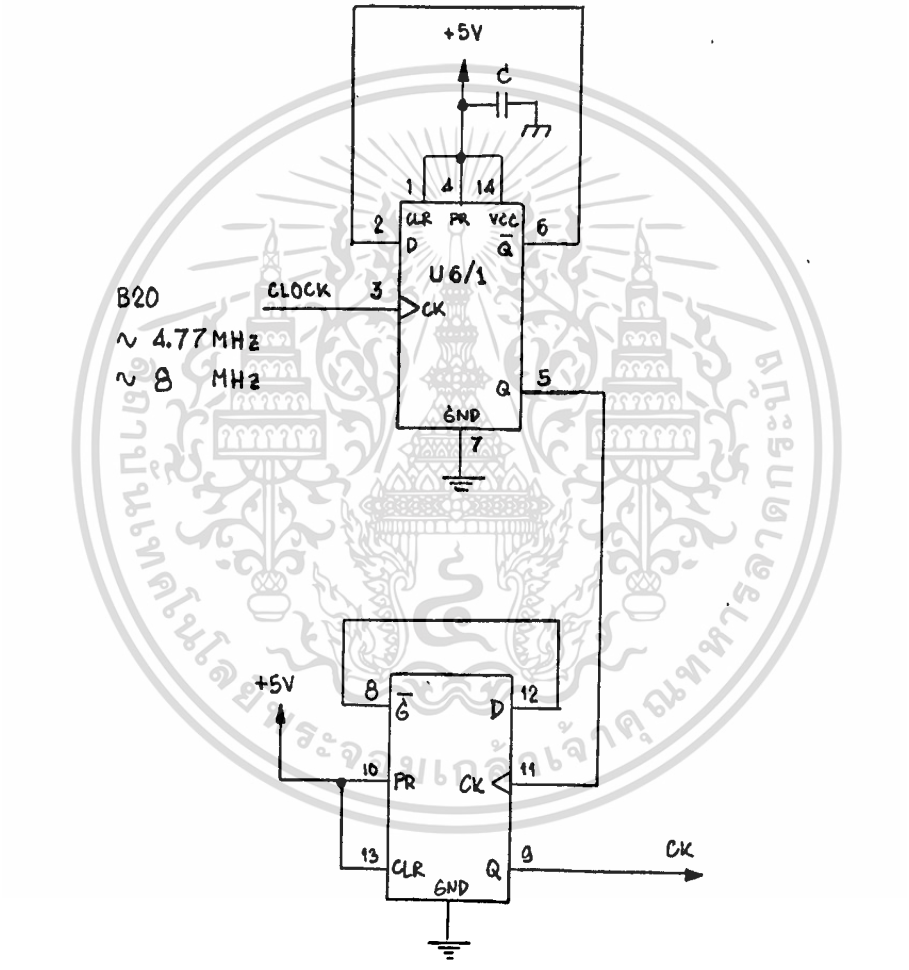
ดังนั้นจึงเห็นว่าบิทที่คงที่เสมอคือ บิทA4 ถึง บิทA9 ส่วนบิทAØ ถึง บิทA3 นั้นเปลี่ยนแปลงได้ ซึ่งได้ใช้ ไอซี (IC) เบอร์74LS138 ซึ่งเป็นไอซีถอดรหัสจาก 3 ไป 8 ดังรูปที่ได้ออกแบบไว้



รูปที่ 3.7.1

3.8 วงจรหารความถี่ 4 เท่า

โดยที่ไอซี เบอร์8251 ซึ่งเป็นไอซีแบบยูซาร์ท (USART) ที่ใช้ความถี่มาเป็นซิสเต็มคล็อค (System Clock)ได้ไม่เกิน 3MHz ในระบบคล็อคของพีซีนั้นมีค่า 4.77MHz-6MHz ดังนั้น เพื่อให้ 8251 ทำงานอย่างมีประสิทธิภาพถึง 75% ของระบบความถี่สูงสุดของมัน จึงได้หารความถี่ของพีซีลงอีกครึ่งหนึ่ง ดังรูป3.8



รูปที่ 3.8

3.9 หน้าที่ของพอร์ท

พอร์ทเบอร์ 3E0H-3E1H

เป็นพอร์ทที่เกี่ยวข้องกับไอซี 8251 ซึ่งมีหน้าที่ดังนี้

1. มีหน้าที่จัดการเกี่ยวกับตัวอักษรที่จะส่งและรับ พร้อมทั้งใส่บิทพิเศษ คือ บิทเริ่มต้น (Start Bit) , บิทหยุด (Stop Bit) , บิทพาริตี้ (Parrity Bit) ซึ่งแล้วแต่จะโปรแกรม (Programe)
2. ควบคุมสถานะต่างๆ เกี่ยวกับการเกิดข้อผิดพลาด (Error)
3. สามารถดูข้อมูลในรีจิสเตอร์ของ 8251 เพื่อเช็ค (Check) สภาวะ (Status) ของเงื่อนไขต่างๆ
4. รับและส่งข้อมูล

ต่อไปเป็นการเซ็ทบิทต่างๆ เพื่อที่จะควบคุม 8251 โดยที่สามารถดูลักษณะการทำงานของขา C/D, RD, WR, CS ว่ามีลักษณะอย่างไร เพื่อที่จะเซ็ทและอ้างพอร์ทได้ถูกต้อง

C/D	RD	WR	CS	
0	0	1	0	8251 DATA => DATA BUS
0	1	0	0	DATA BUS => 8251 DATA
1	0	1	0	STATUS => DATA BUS
1	1	0	0	DATA BUS => CONTROL
X	1	1	0	DATA BUS => 3.STATE
X	X	X	1	DATA BUS => 3.STATE

รูปที่ 3.9.1

ในการส่งข้อมูลในโปรเจ็คนี้ จะใช้ส่งแบบอซิงโครนัส (Asynchronous) ดังนั้น แบบซิงโครนัส (Synchronous) จะไม่กล่าวถึง การเขียนโปรแกรมควบคุม 8251 นั้น เรามีหลักการดังนี้ คือ เมื่อตอนเริ่มต้นโปรแกรมนั้น คำสั่งแรกที่เอาท์ (OUT PORT) ออกไปยัง พอร์ท เบอร์Ø3E1 นั้น ไอซี 8251 จะรู้โดยอัตโนมัติว่า จะต้องเป็นโหมดคำสั่งจัดเรียง (Mode Instrucion Fosrmat) ซึ่งถ้ากำหนดเป็นโหมดอซิงค์โครนัสแล้วคำสั่งต่อไปที่เอาท์ออกไปยังพอร์ท เบอร์Ø3E1H คือคอมมานด์ คำสั่งจัดเรียง(Command Instruction Format) ซึ่งเซ็ทระบบต่างๆ ที่ต้องการ ถ้าหากต้องการจะเซ็ทเป็นโหมดคำสั่งจัดเรียงอีกครั้ง ก็สามารถเซ็ทที่คอมมานด์ คำสั่งจัดเรียงได้ที่บิท D6 หรือเปิดเครื่องใหม่ รูปแบบและหมายเลขต่างๆ ของบิทที่จะเซ็ทนั้น ดูได้ในรูป 3.9.1

<u>พอร์ทเบอร์ 3E2H-3E3H</u>

ในพอร์ทนี้จะมีข้อมูลที่เอาท์ออกไปยังไอซีเบอร์ 74LS374 (U1) แบ่งเป็นสองส่วน คือ 4 บิทบน และ 4 บิทล่าง ซึ่งจะอธิบายดังต่อไปนี้

<u>4 บิทล่าง บิท DØ-D3</u>

4 บิทนี้ เป็นบิทที่ควบคุมโมเด็มเพื่อถอดรหัสหามาตรฐานการส่งและรับตามมาตรฐานสากล คือ BELL 202 และ CCITT V23 ตามข้อมูลของไอซีเบอร์นี้ ดังตาราง3.9.1

ซึ่งได้นำข้อมูลต่างๆ ที่เขียนมาควบคุมเพื่อให้ได้สัญญาณ TRS และข้อมูลที่ส่งไป คือ ØF8H ในกรณีที่ส่งแบบมาตรฐาน BELL 202 ส่ง 1200 บอด รับ 1200 บอด และไม่ส่งสัญญาณโทนให้กับ ไอซี โทน ดิโค้ด ก็เซ็ทให้ 4 บิทบน คือ BIT 4-7 เป็น "1" หมด หรือค่า ØF6H ในกรณีส่งแบบมาตรฐาน CCITT V23 โดยส่ง 600 บอดและรับ 600 บอด

			TXR2	TXR1	FUNCTION
HEX	D3	D2	D1	D0	
00	0	0	0	0	CCITT ส่ง-รับ 1200
01	0	0	0	1	CCITT ส่ง 75 รับ 1200
02	0	0	1	0	CCITT ส่ง 600 รับ 1200
03	0	0	1	1	DISABLED
04	0	1	0	0	CCITT ส่ง 1200 รับ 600
05	0	1	0	1	CCITT ส่ง 75 รับ 600
06	0	1	1	0	CCITT ส่ง 600 รับ 600
07	0	1	1	1	DISABLED
08	1	0	0	0	BELL ส่ง 1200 รับ 1200
09	1	0	0	1	BELL ส่ง 150 รับ 1200
0A	1	0	1	0	BELL ส่ง 1200 รับ 1200
0B	1	0	1	1	DISABLED
0C	1	1	0	0	BELL ส่ง 1200 รับ 1200
0D	1	1	0	1	BELL ส่ง 150 รับ 1200
0E	1	1	1	0	BELL ส่ง 1200 รับ 1200
0F	1	1	1	1	DISABLED

ตารางที่ 3.9.1

4 บิทบน บิท D4-D7

ไว้สำหรับส่งข้อมูลเพื่อถอดรหัส โดยไปต่อกับไอซี 74LS42 ซึ่งเป็น BINARY-TO-DECIMAL DECODE ซึ่งเราจะควบคุมบิทนี้ ดังนี้

พอร์ท 3E4H-3E5H

D7	D6	D5	D4	D3	D2	D1	D0	RESET	SET	COMMENT
RE	X	TR2	CTS	CT	RY2	RY1	TR1			
X	0	0	0	1	1	0	1	8D	0D	ต่อกับ LINE TELEPHONE, ส่งข้อมูล
X	0	1	0	1	1	0	0	AC	2C	LOOP BACK
X	0	0	0	1	0	1	1	8B	0B	TONE TO TELEPHONE
X	0	0	0	1	0	0	1	89	09	RECEIVE CARRIER
X	0	1	0	1	1	0	1	AD	2D	ตัดสายโทรศัพท์ 1 ข้าง TR2
1	0	1	0	1	0	0	0	A8	28	ตัดสายโทรศัพท์ และ OFF

ตารางที่ 3.9.2

จากตารางจะเห็นว่าต้องเซ็ทบิทต่างๆ ตามสภาวะที่โมเด็มกำลังทำงานตอนนั้นๆ จึงไม่อาจระบุค่าที่คงที่ได้ พอร์ทนี้เป็นพอร์ทที่ควบคุมสัญญาณอยู่ถึง 6 สัญญาณ ดังนี้

- บิท D0 เมื่ออยู่ในสภาวะ "0" เป็นการตัดโมเด็มออกจากคู่สายโทรศัพท์
  เมื่ออยู่ในสภาวะ "1" เป็นการต่อโมเด็มกับคู่สายโทรศัพท์
- บิท D1 เมื่ออยู่ในสภาวะ "0" จะตัดสัญญาณโทนคู่ (Dual Tone) ออกจากคู่สายโทรศัพท์

เมื่ออยู่ในสภาวะ "1" จะเป็นการส่งสัญญาณโทนคู่ เมื่อโทรออกไปยังชุมสายโทรศัพท์

- บิท D2 เมื่ออยู่ในสภาวะ "0" จะตัดสัญญาณที่ขา TXA ออก เพื่อกันการรบกวนในการรับข้อมูล

  เมื่ออยู่ในสภาวะ "1" จะเป็นการส่งสัญญาณออกไปยังสายโทรศัพท์

- บิท 3 เมื่ออยู่ในสภาวะ "0" เป็นการเปิดเกทของตัวกันชน (Buffer) เมื่อส่งสัญญาณขัดจังหวะ

  เมื่ออยู่ในสภาวะ "1" เป็นการดิสเอเบิล (Disable) สัญญาณอินเตอร์รับท์ (Interrupt)

- บิท 4 เมื่ออยู่ในสภาวะ "0" เป็นสัญญาณที่ให้ 8251 สามารถส่งข้อมูลไปได้

  เมื่ออยู่ในสภาวะ "1" เป็นสัญญาณที่ให้ 8251 ไม่สามารถส่งข้อมูลได้

- บิท 5 เมื่ออยู่ในสภาวะ "1" เป็นการตัดโมเด็มออกจากคู่สายโทรศัพท์

  เมื่ออยู่ในสภาวะ "0" เป็นการต่อโมเด็มกับคู่สายโทรศัพท์

- บิท 7 เมื่ออยู่ในสภาวะ "0" เป็นการทำงานรีเซ็ท (Reset) ตามปกติ

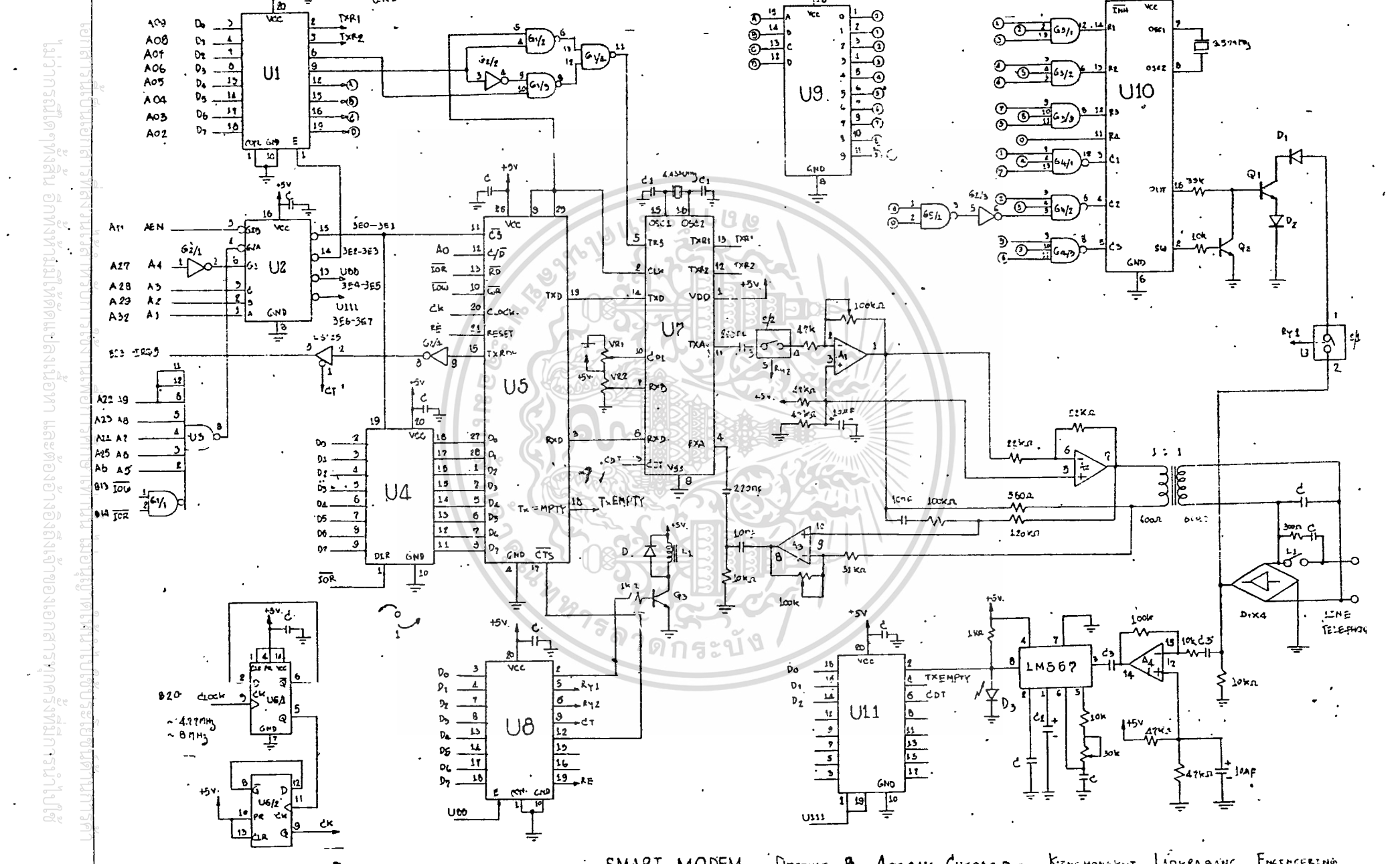
  เมื่ออยู่ในสภาวะ "1" เป็นการรีเซ็ท 8251

พอร์ทเบอร์ 3E6H-3E7H

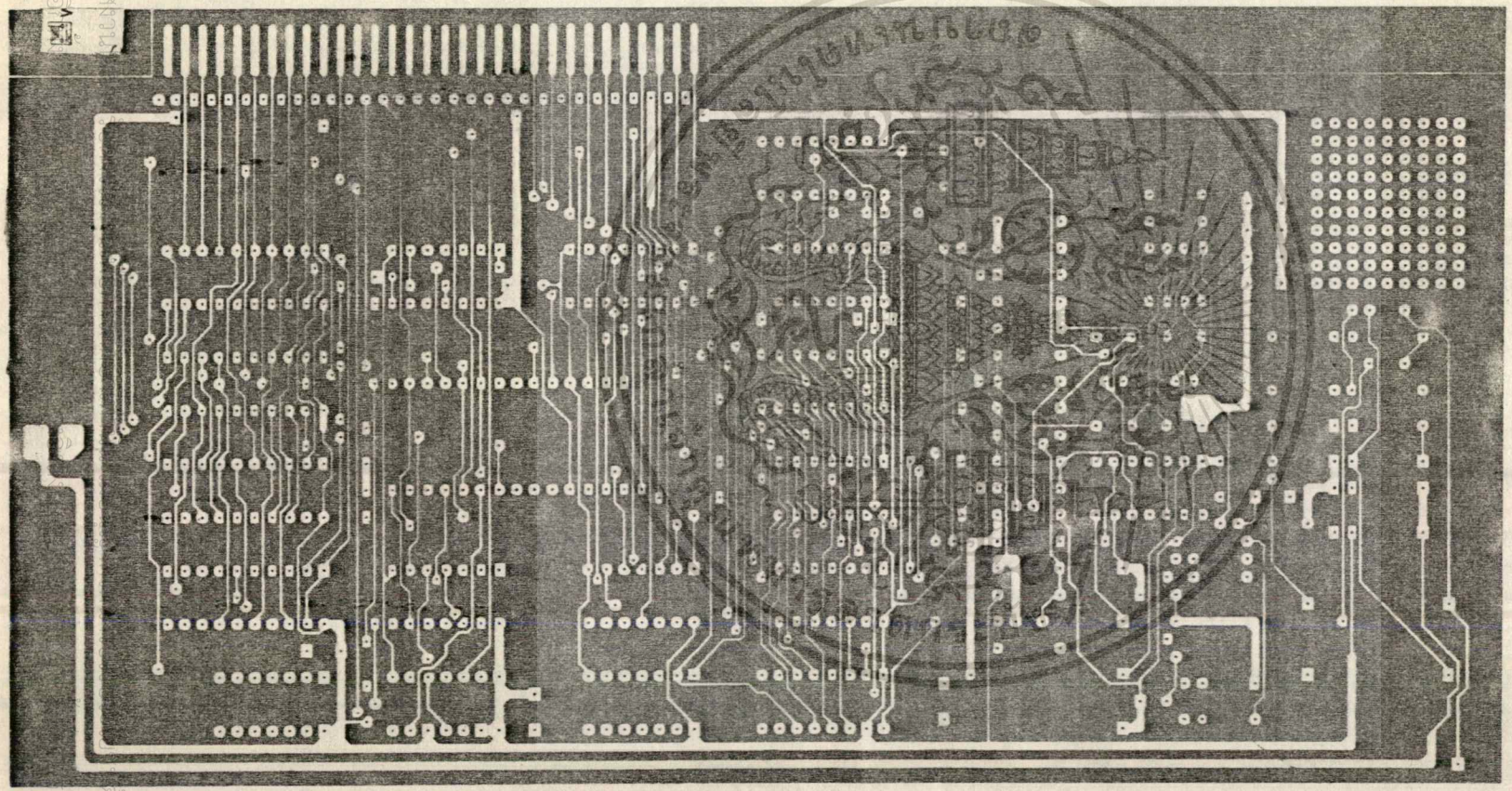
เป็นพอร์ทอินพุท (Input Port) ซึ่งจะเห็นตัวเซ็ทสัญญาณต่างๆ ดังนี้

D2	D1	D0	FUNCTION
X	X	0	LM 567 CHECK 400 Hz ON
X	X	1	LM 567 CHECK 400 Hz OFF
X	0	X	BUFFER ตัวส่งไม่ว่าง กำลังรับข้อมูลจาก CPU
X	1	X	BUFFER ตัวส่งไม่ว่าง เมื่อทำการส่งข้อมูลออกไปหมด
0	X	X	CARRIER FAIL
1	X	X	CARRIER YES

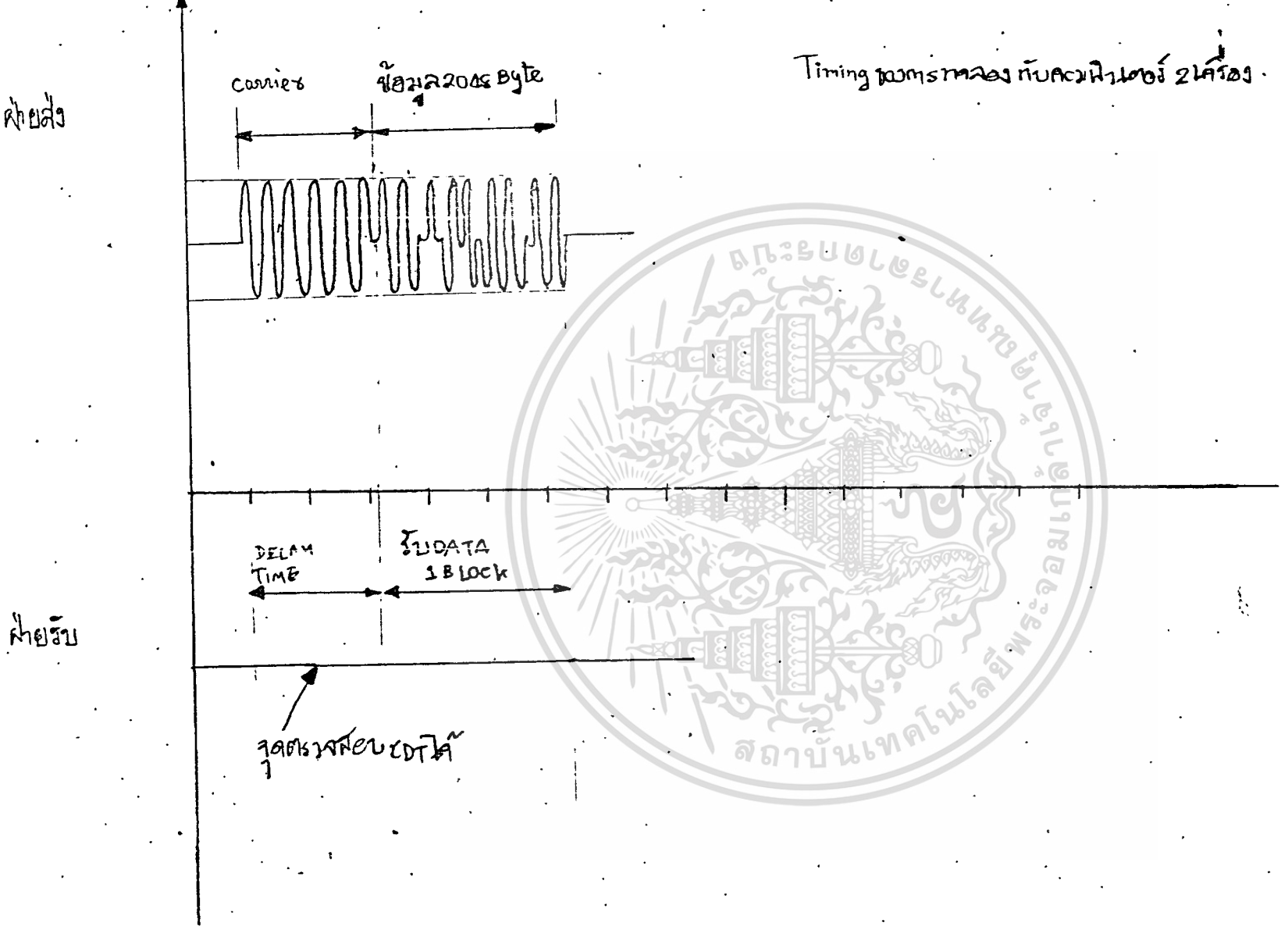
- บิท 0 เช็คสัญญาณไดอัลโทน (Dial Tone) , สัญญาณไม่ว่าง (Busy Tone) , สัญญาณตอบกลับ (Ring Back Tone) , สัญญาณผิดพลาด (Error)
- บิท 1 เมื่อมีสถานะ "Ø" แสดงว่ามีข้อมูลจาก ซีพียู (CPU), เมื่อมีสถานะ "1" แสดงว่าบัฟเฟอร์ (Buffer) ของตัวส่งไม่ว่าง
- บิท 2 เมื่อมีสถานะ "Ø" แสดงว่า No frequency carrier เมื่อมีสถานะ "1" แสดงว่า มี frequency carrier



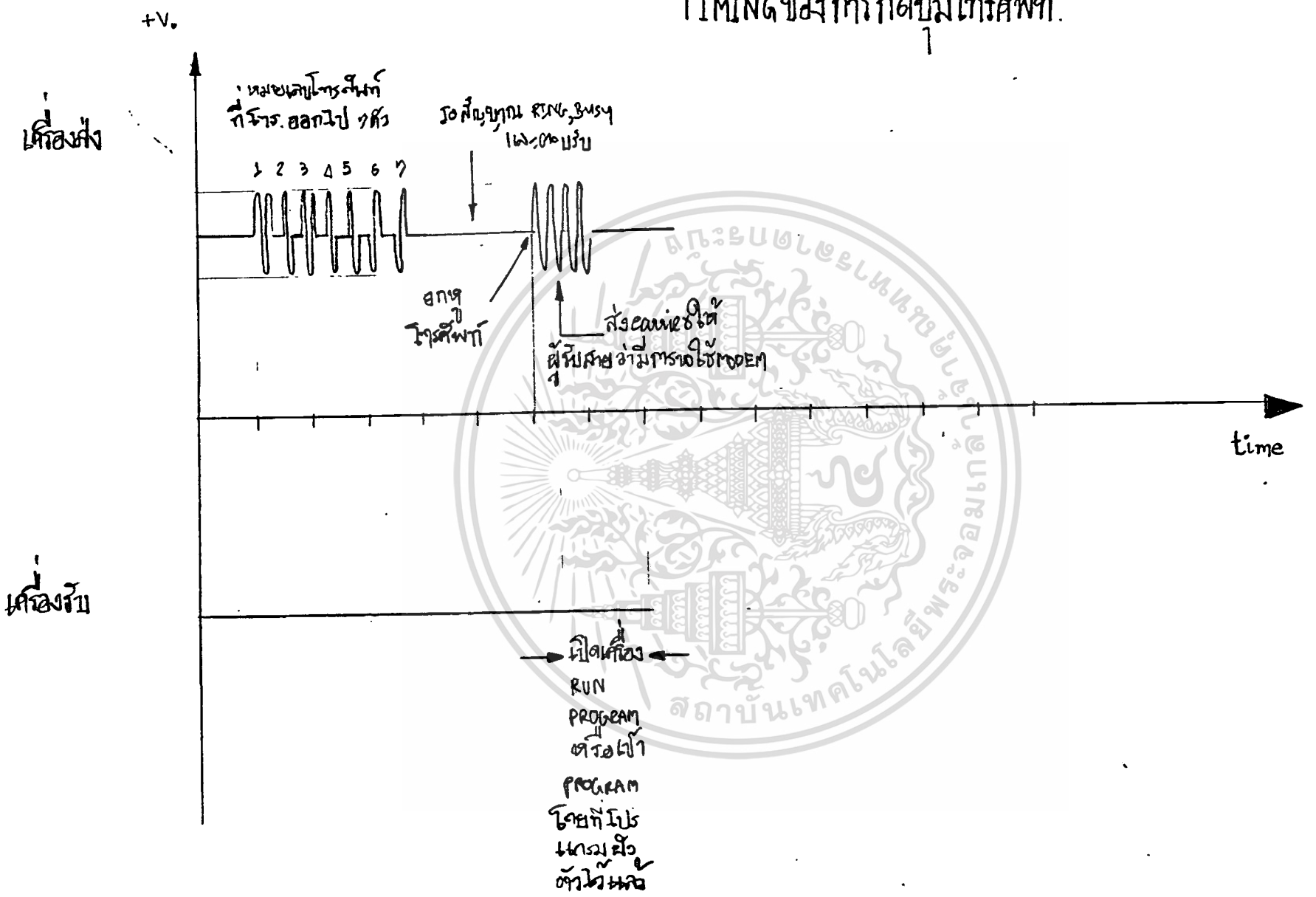
SMART MODEM Drawing & Assembly Chulalongkorn University Engineering

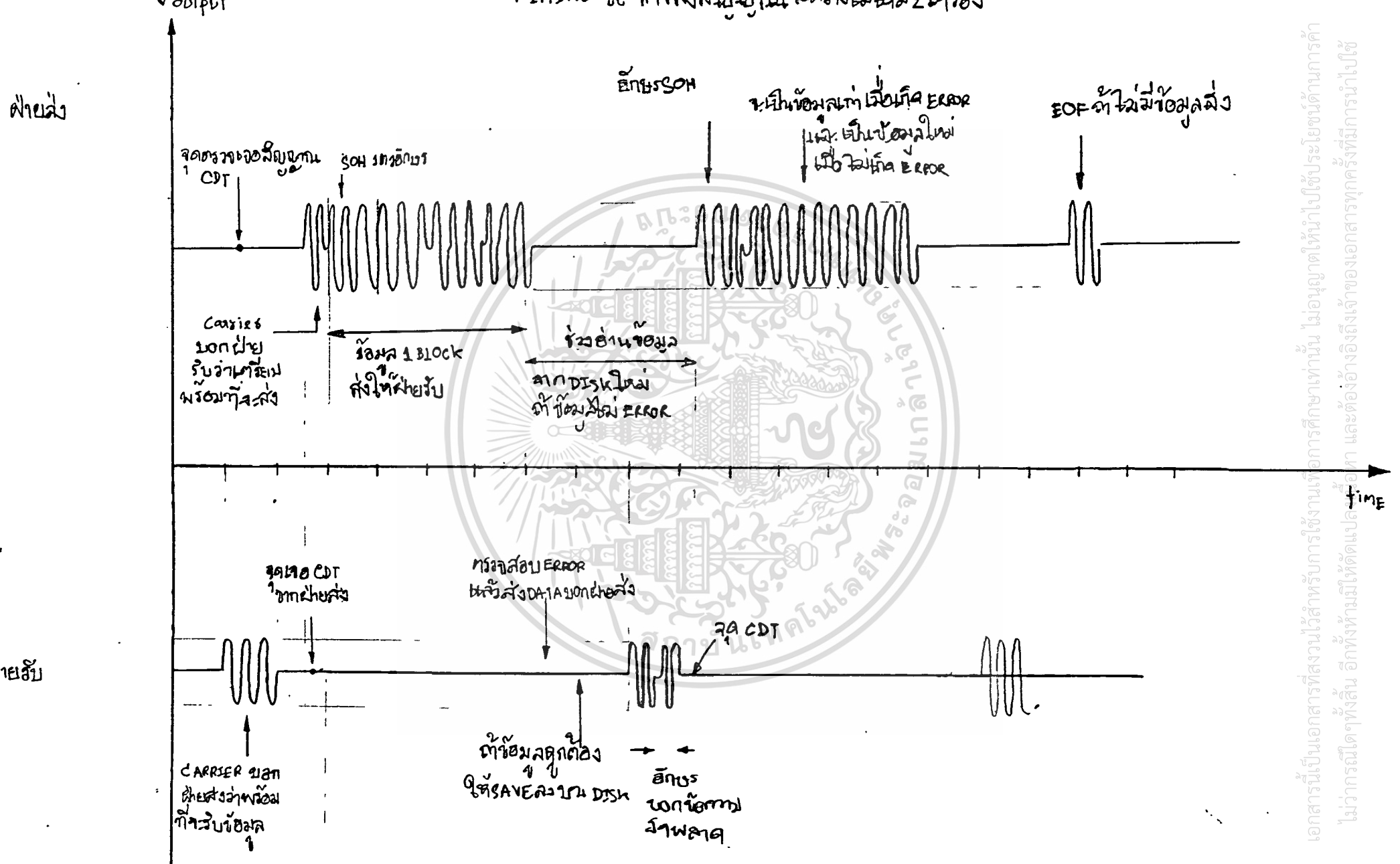


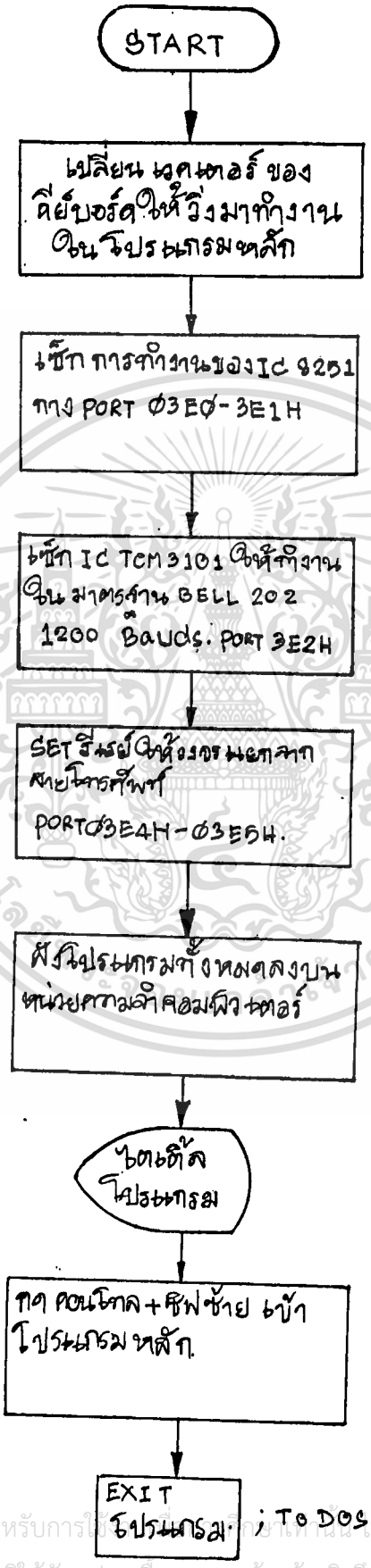
Timing ของการทดลอง กับคอมพิวเตอร์ 2 เครื่อง

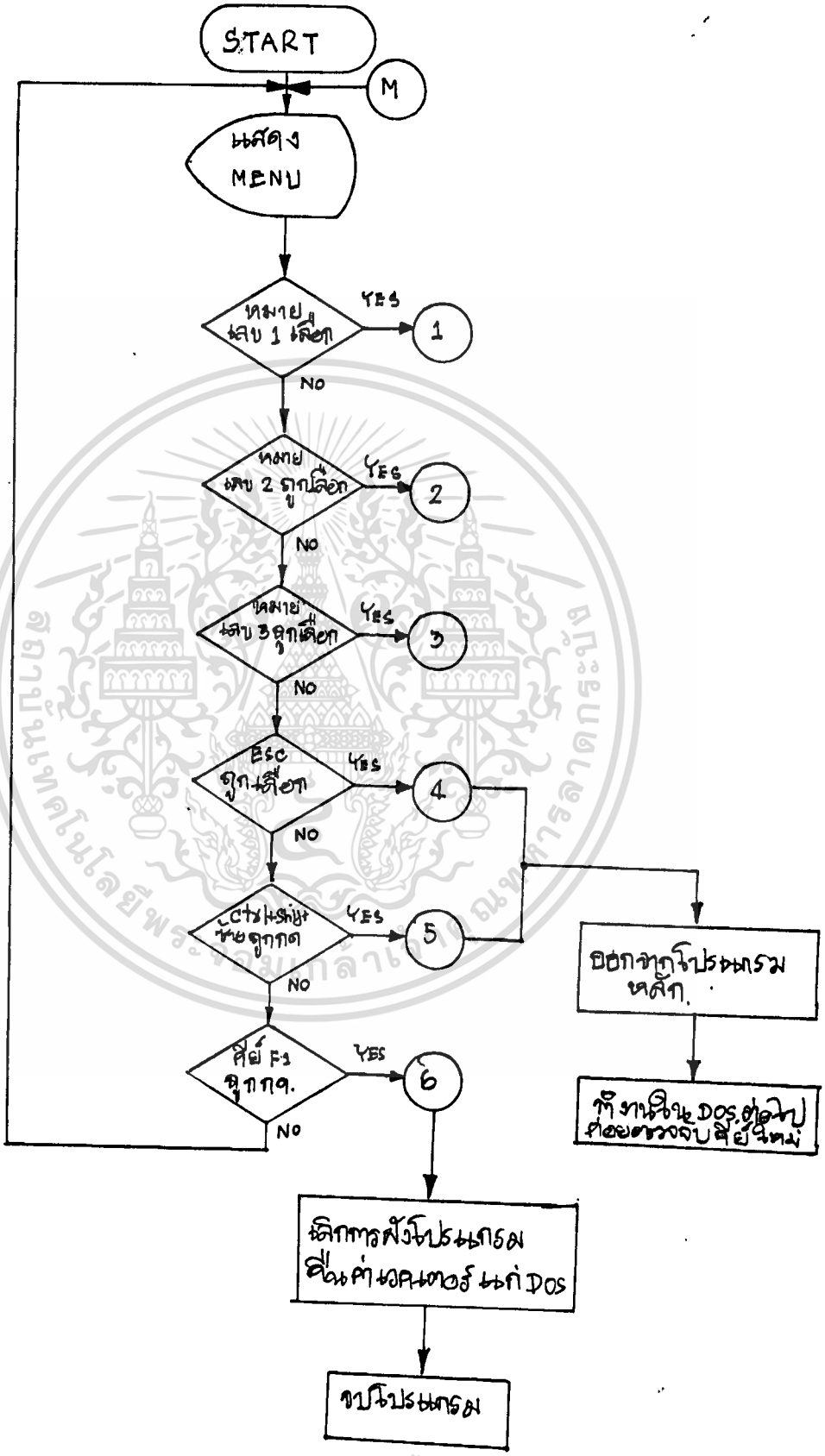


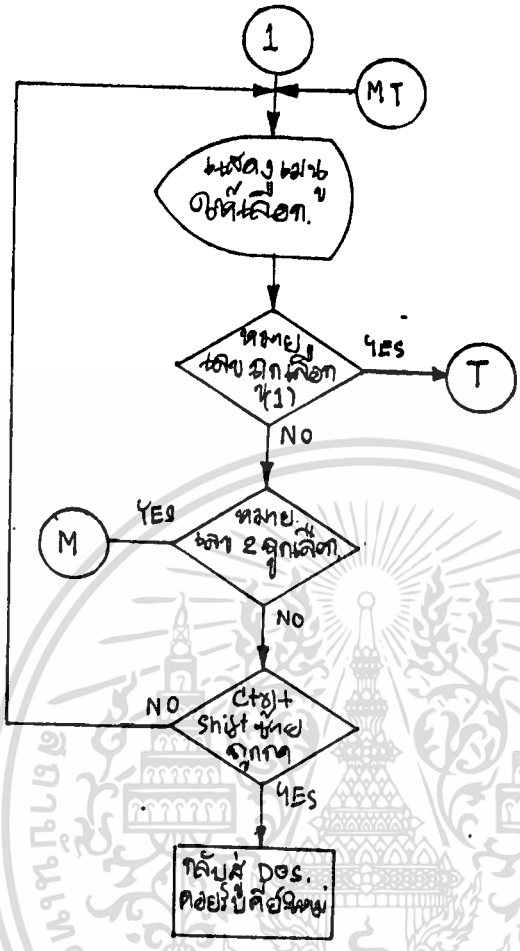
Timing ของการติดต่อโมเด็ม



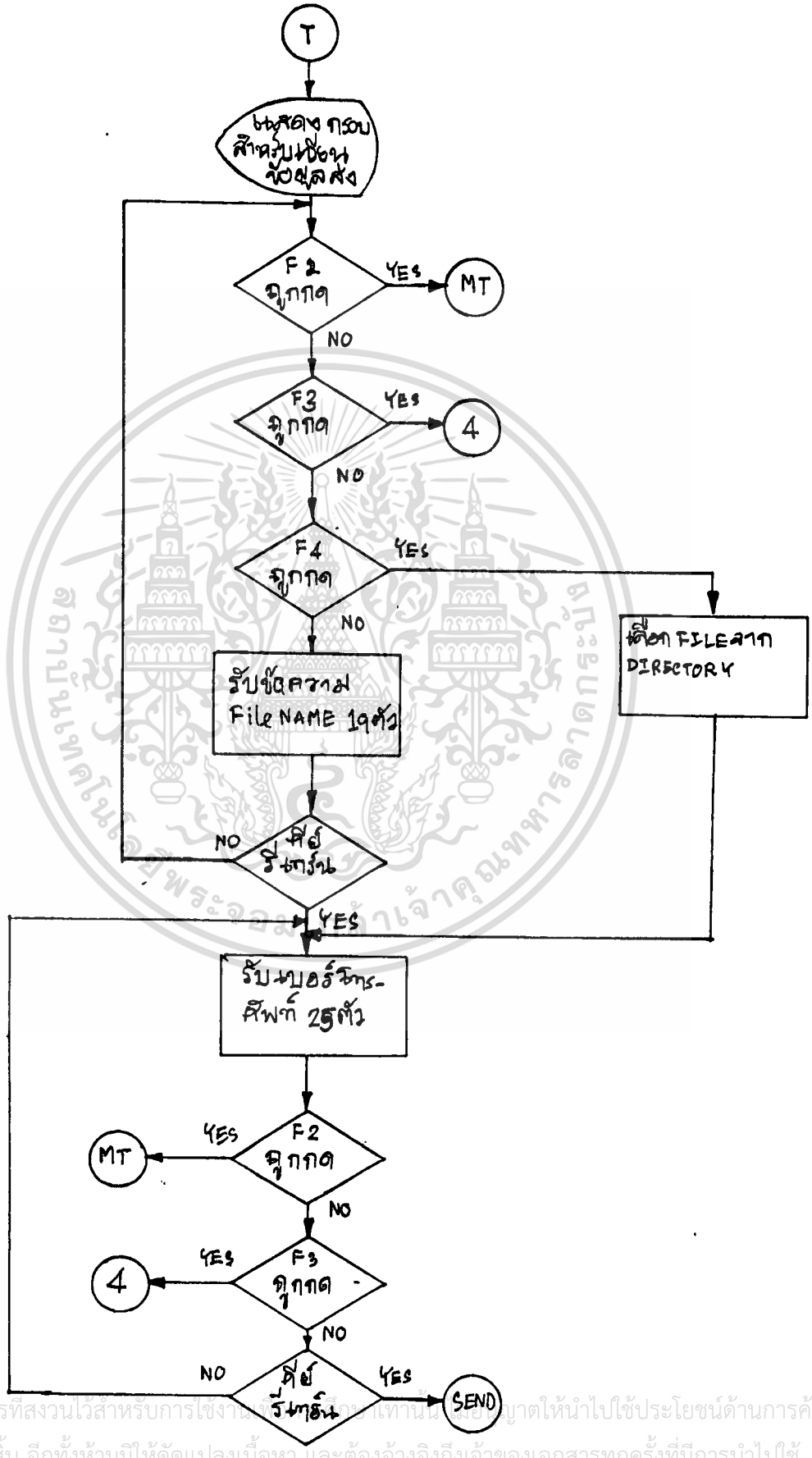

ฝ่ายส่ง
จุดตรวจจอสัญญาณ CDT
SOH รหัสอักขระ
อักขระ EOT
ระหว่างข้อมูลเก่าเมื่อเกิด ERROR
แล้วเป็นข้อมูลใหม่
เมื่อไม่เกิด ERROR
EOT ถ้าไม่มีข้อมูลส่ง
Carrier
บอกฝ่าย
รับว่าเครื่องพร้อมที่จะส่ง
ข้อมูล 1 BLOCK
ส่งให้ฝ่ายรับ
ระหว่างข้อมูล
จาก DISK ใหม่
ที่ข้อมูลไม่มี ERROR
time
จุดหาย CDT
ของฝ่ายส่ง
ตรวจสอบ ERROR
หลังส่ง DATA บอกฝ่ายส่ง
จุด CDT
ฝ่ายรับ
CARRIER บอก
ฝ่ายส่งว่าพร้อม
ที่จะรับข้อมูล
ถ้าข้อมูลถูกต้อง
ให้ SAVE ลงบน DISK
อักขระ
บอกข้อความ
ผิดพลาด

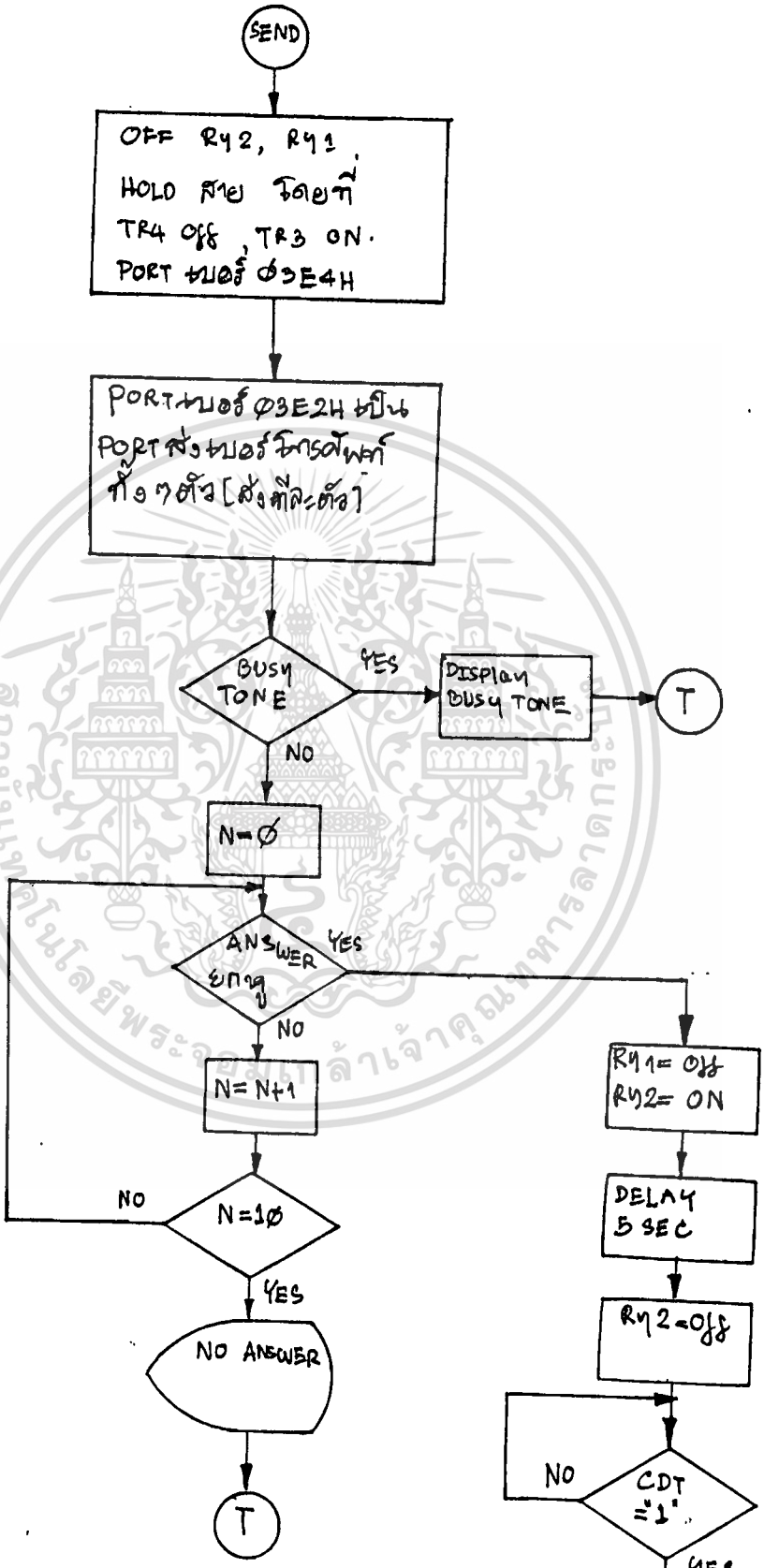


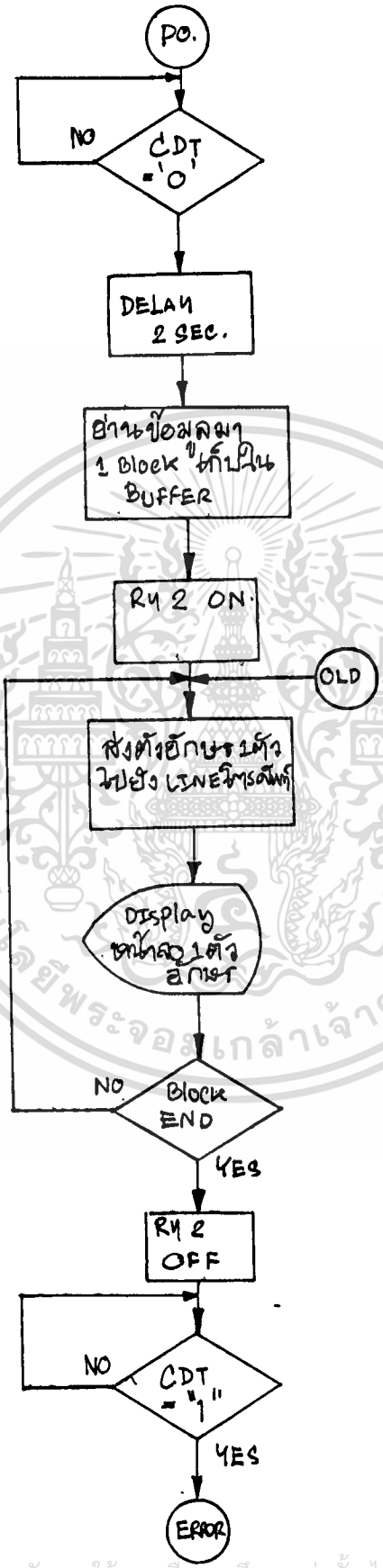


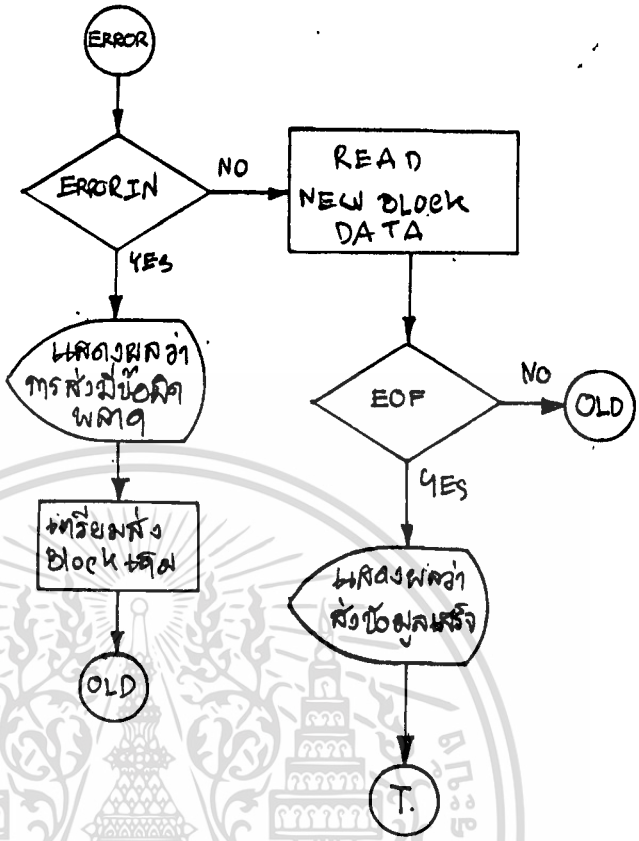


โฟว์ชาตส่วนนี้ เป็นส่วนย่อยของการส่งข้อมูล [แป้นเมนู] โปรแกรมย่อย T คือการส่งข้อมูลในโปรแกรมหลัก ซึ่งมีโปรแกรมย่อยหลายโปรแกรมอยู่ เช่น โปรแกรม ค้นหาไฟล์ เรียกใช้งานฟังก์ชัน ๑๗ โปรแกรมต่อโทรศัพท์ โดยอัตโนมัติ



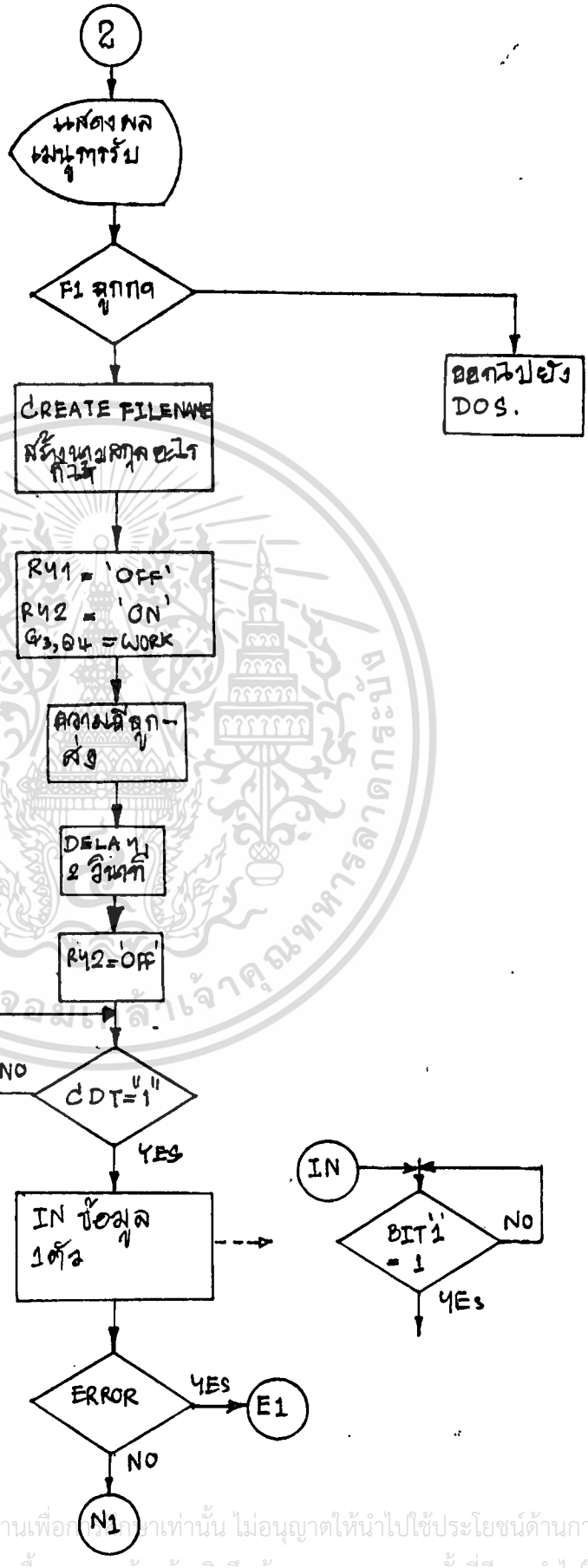


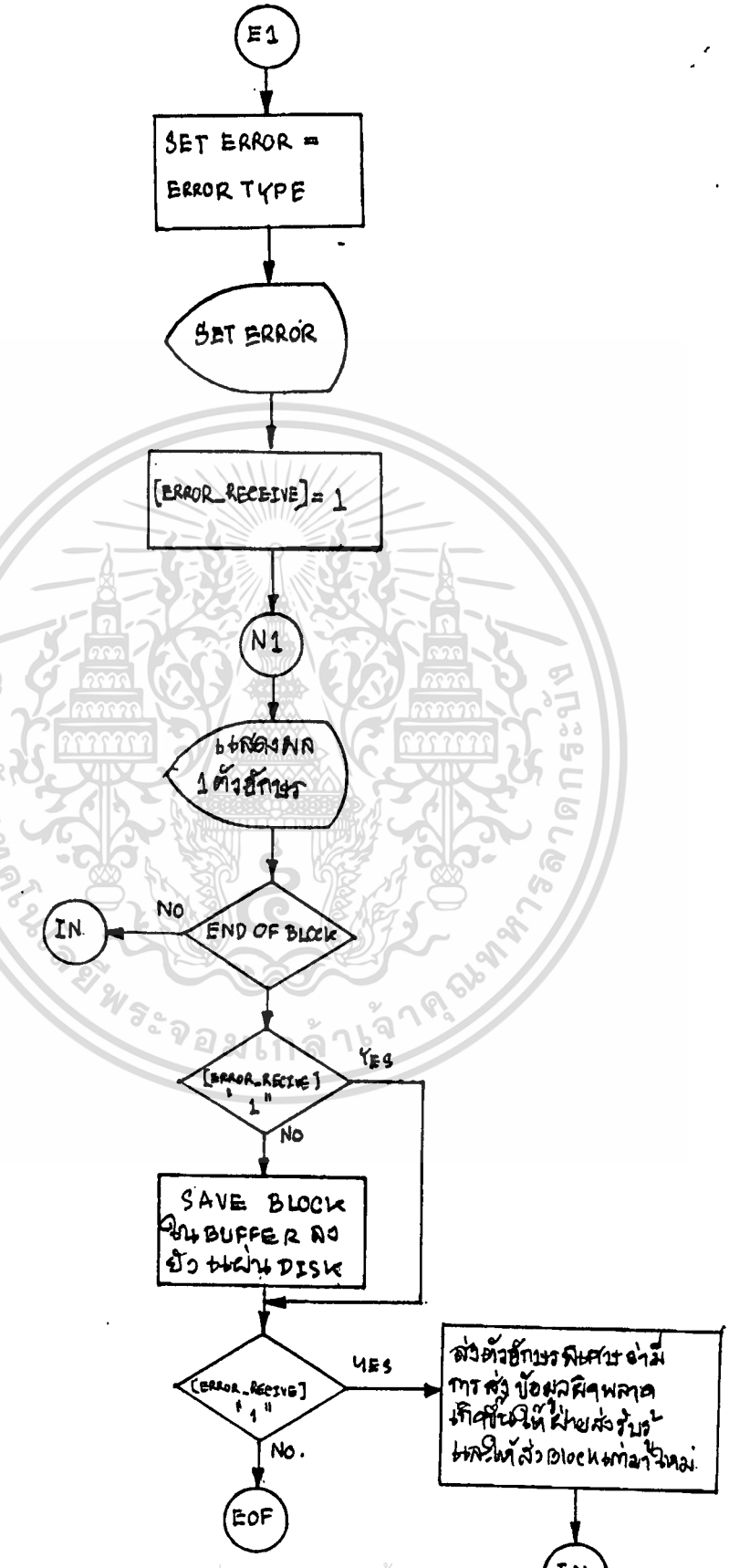


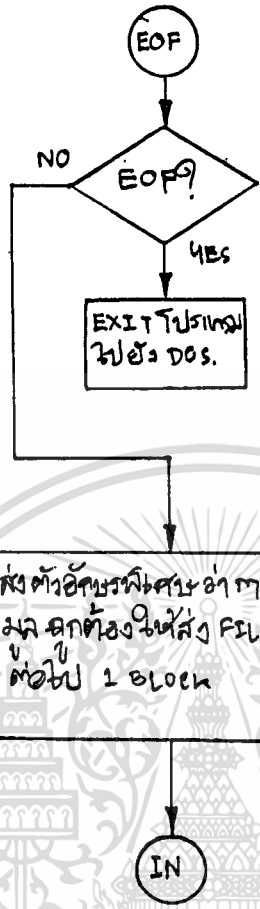


<u>การส่งข้อมูล</u> มีการทำงานดังนี้.

จะเรียก FILE NAME มาเก็บ และหมายเลขโทรศัพท์ 7 ตัว หลังจากนั้นจะส่งหมายเลขโทรศัพท์เพื่อเรียกโมเด็มอีกเครื่องหนึ่งในการตรวจสอบสัญญาณวิ่งกลับของอุปกรณ์โทรศัพท์ และใช้โปรแกรมตรวจสอบพร้อมกับส่วนวงจร LM567 หลังจากที่ฝ่าย ANSWER รับสายแล้ว ฝ่าย ORIGINAL จะส่งความถี่ 1200Hz ไปประมาณ 5 วินาที เพื่อให้ฝ่ายรับสายรู้ว่าเป็นสัญญาณจากโมเด็มมิใช่เสียงคนพูด. จากนั้นฝ่ายส่งจะรอรับสายโดยที่ไปตรวจสอบที่สัญญาณ CDT ถ้าถูกเช็คหมายความว่า ฝ่ายรับ พร้อมที่จะรับข้อมูล จากนั้นก็จะไปประมาณ 2 วินาทีสิ่งเริ่มส่งข้อมูลไปได้ โดยส่งไปครั้งละ 1 บล็อค จากนั้นก็ตรวจสอบการส่งว่ามีข้อผิดพลาดหรือไม่ หากฝ่ายรับที่ส่งข้อมูลมาให้ ถ้าไม่มีข้อผิดพลาดก็ส่งต่อลงถึง EOF หรือ DONE ถ้าหากว่ามีข้อผิดพลาดก็ทำการส่งข้อมูลเดิมอีกครั้ง. ในการส่งทุกครั้งจะมีตัวอักษรพิเศษส่งนำหน้าไปก่อนทุกครั้งใช้ 3 ตัว ตรวจสอบแต่ใน Flow chart ไม่ได้ใส่ไว้ รายละเอียดมีดังนี้.

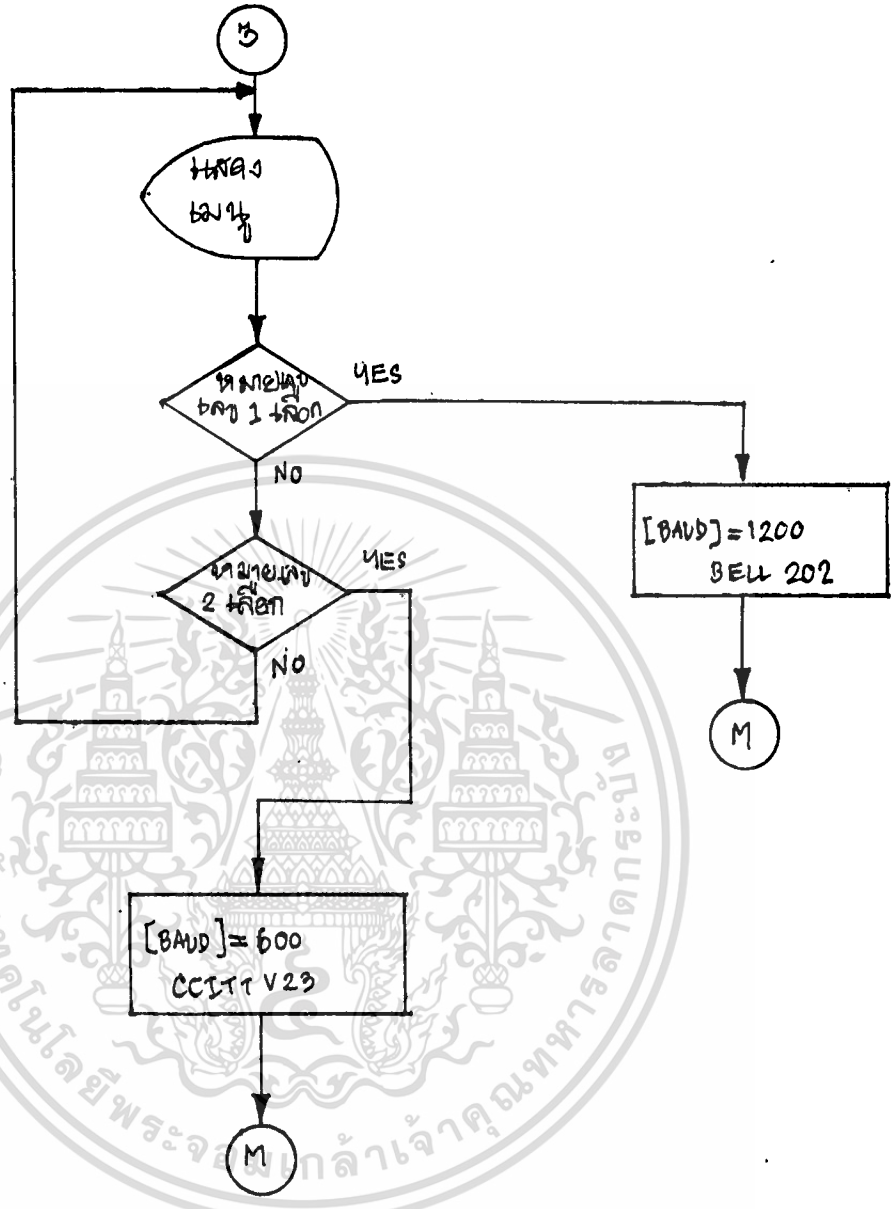






การทำงานของฝ่ายรับ

ฝ่ายรับละรอรับ สัญญาณจากโทรศัพท์ เมื่อ USER ยกหูโทรศัพท์ ที่นั้นก็ RUN โปรแกรม ถ้ายการเลือก FILE ที่จะสร้างขึ้นมาใหม่ แล้วเสร็จแล้ว ฝ่ายรับก็ต้องส่งสัญญาณความถี่ ไปประมาณ 2 วินาที แล้วหยุด และคอยรับข้อมูลจากฝ่ายส่ง โดยการตรวจสอบที่สัญญาณ CDT เมื่อฝ่ายส่ง ส่งข้อมูลมา ก็นับจำนวนอักษรนั้นครบ 1 BLOCK และตรวจสอบความผิดพลาด ถ้าหากว่ามีความผิดพลาดจะไม่ SAVE ข้อมูลลงบน DISK ถ้าไม่มีความผิดพลาดก็ SAVE ข้อมูลลงบน DISK และตรวจสอบตัวอักษรสุดท้ายว่า EOF หรือไม่. [จะส่งมาทีละ 3 ตัว] เป็นตัวตรวจสอบ. ถ้าหากข้อมูล EOF แล้ว ก็จะถือว่าจากโปรแกรมไปสู่ DOS. ต่อไป. ก่อนจบโปรแกรมก็ ปิดการ RELA ต่างๆ ที่ค้างอยู่ในสถานะไม่ทำงาน.



```
        .xlist                          ; off listing
        page    57,132                  ; line/page=57,width=132 col.
        title   DataCommunication       ; Title name
        subttl  Press key Ctrl-Lshift ; Subtitle
        .list                           ; listing
;=======================================================
;   PROJECT     ;   Smart Modem                         =
;   PROGRAMMER  ;   Apirak Chirasopone                  =
;   SECTION     :   Computer Technology                 =
;   DEPARTMENT  :   Industrial Instrumentation  Technology =
;   FACULTY     :   Engineering                         =
;   UNIVERSITY  :   Institute of technology Ladkrabang  =
;=======================================================

;*********************************************************
;------ IDENTIFICATION DIVISION                          *
;------ Program         : SMART MODEM                    *
;------ Procedure       : 1. Check key Ctrl_Lshift ,esc  *
;                        : 2. Display message             *
;                        : 3. Residet program             *
;                        : 4. Transmitte data             *
;                        : 5. Receive data                *
;                        : 6. Cancel program Press F1     *
;                        : 7. Send 'EOF' Then end send    *
;*********************************************************
;------ EQUATES DIVISION------
zero            equ     0
one             equ     1
four            equ     4
page0           equ     zero            ; page 0
base            equ     100h            ; Base address program
color           equ     0b800h          ; set color display
mon             equ     0b000h          ; set monochrom display
fun2            equ     2
dos21h          equ     21h
fun9            equ     9               ; function display
block           equ     500
;-------INTERRUPT CALL--------
no_key_int      equ     09h             ; keyboard interrupt
irq4            equ     0ch             ; level 4 of the 8259
video           equ     10h             ; routine provides the CRT
keyboard        equ     16h             ; keyboard service
resident        equ     27h             ; terminate and resident

;------- KEYBOARD SERVICE------
key_rd_flag     equ     2               ; read keyboard flag
key_ctrl        equ     04h             ; alternate key depressed
key_lshift      equ     02h             ; right shift key depressed

;------- PROCEDURE DIVISION
interrupt       Segment at zero         ; interrupt vector
                org     no_key_int*four ; 09*4
keyint          dw      2 dup(?)        ; keyboard vector
                org     irq4*four       ; 0C*4
incom0          dw      2 dup(?)        ; com0 vecter
interrupt       ends
;*************************************************************************
;       MAIN PROGRAM FOR TRANSMITTE AND RECEIVE ONE CHARACTER           *
;*************************************************************************
cseg            SEGMENT  PARA PUBLIC 'cseg'
                ASSUME   cs:cseg
                ORG      base           ; base = 100h

                INCLUDE  MA_CRO.PRO  ; include macro

start           Proc   near
                jmp    initial  ; jmp to initialize resident

                INCLUDE DB.PRO       ; include db

new_key:        sti                  ; enable interrupt
                push     ax          ; save registers
                push     bx
```

```
                push    cx
                push    dx
                push    si
                push    di
                push    ds
                push    es
                pushf

                mov  [rec_check],0
                call cs:old_keyboard    ; call old keyboard routine
                mov  ah,key_rd_flag     ; read keyborad flag
                int  keyboard           ; request function
                and  al,key_ctrl or key_lshift
                cmp  al,key_ctrl or key_lshift
                jne  exit2              ; short jum
                cmp  check1,1
                jne  work               ; if ready to pop and to old address
                mov  check2,1           ; key ctr,lshift and esc pressed

exit2:          jmp  exit_keyboard      ; to pop and to old address
work:
                Get_cur                 ; get current display mode
                cmp  al,7               ; monochrom adapter = 07 ?
                je   yes
                mov  ax,color           ; memory = 0b800h
                jmp  chg
yes:            mov  ax,mon             ; memory = 0b000h
chg:            mov  es,ax              ; es = address screen memory
                Read_po                 ; Read cursor position

                mov   [stdis],0
                mov   si,00             ; set index address screen memory
                push  si                ; pass parameter to subroutine save
                mov   bx,4000
                push  bx
                call  save              ; to save old screen transmitter
                pop   bx
                pop   si                ; return sp

                mov   check1,1          ; give check1 = 1 ,show that ready
                mov   check2,0          ;
                call  txmit             ; to transmitte and receive data
                mov   check1,0

                mov   [stdis],0         ; set parameter old
                mov   di,00             ; index screen
                push  di                ; to stack pointer
                mov   bx,4000
                push  bx
                call  Old               ; clrscr display transmitter
                pop   bx
                pop   di                ; retrun sp to normal

exit_keyboard:  pop   es
                pop   ds
                pop   di
                pop   si
                pop   dx
                pop   cx
                pop   bx
                pop   ax
                iret
                ;*********************
                ;..Save old screen..*
                ;*********************
save            proc  near
                mov   di,0              ; index buffer
                add   sp,2              ; sp = sp+4 index of screen
                pop   bx                ; index screen
                pop   si
                sub   sp,6              ; old position of ip
area1:          mov   cx,160            ; count character + attribute
area2:          mov   ah,es:[si]        ;
                cmp   [stdis],1         ; save space of transmitter ?
                jz    orther            ;
                mov   emty[di],ah       ; screen is saved into buffer
```

```
                jmp   increment
orther:         mov   emty1[di],ah     ; screen of receive is saved
increment:      inc   si
                inc   di
                loop  area2
                cmp   di,bx
                jnae  area1            ;jump if not above or equal
                ret
save            ENDP
                ;*********************
                ;..Display Message..*
                ;*********************
disp            Proc  near
                mov   si,0             ; index buffer
                add   sp,2             ; position di
                pop   di               ; index screen memory
                sub   sp,4             ; old position  of ip
                cmp   [check4],0
                je    move1
                cmp   [check4],1
                je    move1
                cmp   [check4],4
                jnz   move_1
move1:          mov   cx,28            ; count col.
                jmp   move2
move_1:         mov   cx,160
move2:          cmp   [check4],0
                jne   c1
                mov   al,menu[si]
                jmp   play
c1:             cmp   [check4],1       ; receive data
                jne   c2
                mov   al,tran[si]      ; transmitter
                jmp   play
c2:             cmp   [check4],2
                jne    c3
                mov   al,border[si]
                jmp   play
c3:             cmp   [check4],3
                jne   c4
                mov   al,border_1[si]
                jmp   play
c4:             cmp   [check4],4
                jne   play
                mov   al,baud_menu[si]
play:           mov   es:[di],al       ; move value border to screen memory
                inc   di
                mov   byte ptr es:[di],07; move attribute to screen memory
                inc   di
                inc   si
                loop  move2
                cmp   [check4],0
                je    lge
                cmp   [check4],1
                je    lge
                cmp   [check4],4
                jne   large
lge:            add   di,54*2
                cmp   si,len_menu
                jnae  move12            ; if no then goto move1
                jmp   end_disp
large:          cmp   [check4],2       ; 20 line border
                je    haft_len
                cmp   si,len_border_1
                jnae  move11
                jmp   end_disp
haft_len:       cmp   si,len_border
                jnae  move11
                jmp   end_disp
move11:         jmp   move_1
move12:         jmp   move1
end_disp:       ret
disp            endp
                ;*************************************
```

```
                ; WRITE MESSAGE  TO TRANSMITTE DATA *
                ;*************************************
message         Proc  near

                INCLUDE MESSAGE.PRO

message         ENDP
                ;**********************
                ;..RETURN OLD SCREEN..*
                ;**********************
Old             Proc  near
                mov  si,0               ; si = buffer
                add  sp,2               ; sp index di
                pop  bx                 ; retrun index screen
                pop  di
                sub  sp,6               ; sp index ip
put3:           mov  cx,160             ; attribute + ascii
put4:           cmp  [stdis],1
                jz   put5
                mov  al,emty[si]        ;
                jmp  put6
put5:           mov  al,emty1[si]
put6:           mov  es:[di],al         ; tranfer buffer to screen area
                inc  di
                inc  si
                loop put4               ; line full ?
                cmp  si,bx              ; string ready ?
                jnae put3               ; jump if not above or equal
                mov  dx,coordinate      ;
                mov  bh,00              ;-------------------------------
                mov  ah,fun2            ;                              !
                int  video              ; write old position to screen  !
                ret                     ;                              !
                                        ;-------------------------------
Old             ENDP
                ;*************
                ; MAIN MENU, *
                ;*************
txmit           Proc near
agian_play:     mov   di,1332             ; set index address screen memory
                push  di                  ; pass parameter to subroutine disp
                mov   [check4],0          ; border menu
                call  disp                ; to display border transmitter
                pop   di                  ; retrun sp
                mov   [xcordi],27         ; set cursor position x = 1
                mov   [ycordi],8          ; set cursor position y = 0
                mov   di,1334             ; row =2 , col. = 1
                push  di                  ; pass parameter to subroutine message
                mov   [check3],0          ; mov check  select display
                call  message             ; to write install
                pop   di                  ; retrun  sp
                mov   [xcordi],27         ; set cursor position x = 1
                mov   [ycordi],15         ; set cursor position y = 19
                mov   di,2454             ; row = 19 , col. = 1
                push  di                  ; push to suroutine
                mov   [check3],2          ; select display F1
                call  message             ; to write message
                pop   di                  ; retrun stack pointer
                set_cur 45,15
                ;=============
                ;     Menu   =
                ;=============
stand:          cmp  check2,1       ; key ctr,lshift pressed ?
                je   exit_out       ; if yes goto exit_out
tt:             mov   ah,01h        ; function 01 = keyboard status
                int   16h           ; transfer to rom driver
                jz    stand         ; loop if no key wait
                read                ; function 00h int 16h read char.
                cmp   ah,2          ; number 1
                jne   key2
                call  num1          ; to transmitter
                cmp   [m_check],1
                je    agian_play    ; to main menu again
                jmp   exit_out
key2:           cmp   ah,3
                jne   key3
```

```
                call  num2          ; to receiver
                jmp   exit_out
key3:           cmp   ah,4
                jne   esc
                call  baud_select
                jmp   agian_play
esc:            cmp   ah,1          ; key ESC ?
                je    exit_out      ; if key ESC then exit program
                cmp   ah,59         ; check key F1 to clear program
                jne   stand

end_resident:   assume ds:interrupt   ; clear interrupt vecter
                mov  ax,interrupt
                mov  ds,ax
                mov  ax,word ptr cs:[old_keyboard]
                mov  ds:[keyint],ax
                mov  ax,word ptr cs:[old_keyboard+2]
                mov  ds:[keyint+2],ax
exit_out:       ret
txmit           Endp

;End_Menu.....................................................................
                ;-------------------
                ; SELECT BAUD RATE
                ;-------------------
baud_select     Proc Near
baud_send:      mov   di,2100       ; set index address screen memory
                push  di            ; pass parameter to subroutine disp
                mov   [check4],4    ; border transmitter
                call  disp          ; to display border transmitter
                pop   di            ; retrun sp
                set_cur 29,19
baud_wait:      mov   ah,01h        ; function 01 = keyboard status
                int   16h           ; transfer to rom driver
                jz    baud_wait     ; loop if no key wait
                read                ; function 00h int 16h read char.
                cmp  ah,2           ; number 1
                jne  key_baud
                mov  [baud_rate],0F8H ; 1200 b/s
                jmp  exit_out_baud
key_baud:       cmp  ah,3
                jne  baud_wait
                mov  [baud_rate],0F6H ; 600 b/s
exit_out_baud:  mov dx,03E2H
                mov al,[baud_rate]
                out dx,al
exit_baud:      ret
baud_select     Endp
                ;-----------------
                ; TRANSMITTER DATA
                ;-----------------
num1            Proc near
menu_send:      mov   di,2150       ; set index address screen memory
                push  di            ; pass parameter to subroutine disp
                mov   [check4],1    ; border transmitter
                call  disp          ; to display border transmitter
                pop   di            ; retrun sp
                mov   [xcordi],35     ; set cursor position x = 1
                mov   [ycordi],13      ; set cursor position y = 0
                mov   di,2152        ; row =2 , col. = 1
                push  di              ; pass parameter to subroutine message
                mov   [check3],3      ; mov check  select display
                call  message         ; to write install
                pop   di              ; retrun  sp
                set_cur 54,19
stand_1:        cmp  check2,1       ; key ctr,lshift pressed ?
                je   exit_num1      ; if yes goto exit_out
                mov   ah,01h        ; function 01 = keyboard status
                int   16h           ; transfer to rom driver
                jz    stand_1       ; loop if no key wait
                read                ; function 00h int 16h read char.
                cmp  ah,2
                jne  key_2

                call s_file
                cmp  [m_tran],1     ; if = 0 then read menu again
```

```
                jne  menu_send
                mov  [m_check],0   ; exit program
                jmp  exit_num1
key_2:          cmp  ah,3
                jne  stand_1
                mov  [m_check],1
exit_num1:      ret
num1            Endp
                ;=============
                ; Send File  =
                ;=============
s_file          Proc near
                Assume ds:cseg
                push  cs
                pop   ds
                mov   di,00
                push  di            ; pass parameter to subroutine disp
                mov   [check4],3    ; border receive
                call  disp          ; to display border transmitter
                pop   di            ; retrun sp
                mov   [xcordi],1    ; set cursor position x = 1
                mov   [ycordi],0    ; set cursor position y = 0
                mov   di,02         ; row =2 , col. = 1
                push  di            ; pass parameter to subroutine message
                mov   [check3],6    ; mov check  select display
                call  message       ; to write install
                pop   di            ; retrun  sp
                mov   [xcordi],1    ; set cursor position x = 1
                mov   [ycordi],21   ; set cursor position y = 0
                mov   di,3362       ; row =22 , col. = 1
                push  di            ; pass parameter to subroutine message
                mov   [check3],7    ; mov check  select display
                call  message       ; to write install
                pop   di            ; retrun  sp
new_char:       mov  [xcordi],43
                mov  [ycordi],0
                set_cur [xcordi],[ycordi]
space:          mov  cx,19
                mov  al,' '
space_1:        dec  [xcordi]       ; clear only one character
                write               ; back to the old character
                set_cur [xcordi],[ycordi]  ; write character to screen
                loop space_1
                inc  [xcordi]
                set_cur [xcordi],0
                mov  si,offset fname     ; address file-name
                mov  cx,18
stand_11:       mov  ah,01h         ; function 01 = keyboard status
                int  16h            ; transfer to rom driver
                jz   stand_11       ; loop if no key wait
                read                ; function 00h int 16h read char.
                cmp  ah,60          ; F2
                jne  key_f2
out_a:          mov  [m_tran],0     ; again menu
                jmp  exit_s_file
key_f2:         cmp  ah,61          ; F3
                jne  key_f3
out_F3:         mov  [m_tran],1     ; exit
                jmp  exit_s_file
key_f3:         cmp  ah,62          ; F4
                jne  back
                Call directory
                cmp  [dir_check],2 ; F2 = 1  F3 = 2  open = 0
                je   out_F3         ; to exit program
                cmp  [dir_check],1
                je   out_a          ; to again menu
                Call telephone_num
                cmp  [dir_check],2
                je   out_F3
                cmp  [dir_check],1
                je   out_a
                jmp  open_dir       ; to open file
                ;=============
                ; BACK SPACE =
                ;=============
```

```
back:           cmp  ah,14
                jne  r_trun
                mov  al,' '             ; clear only one character
                dec  [xcordi]           ; back to the old character
                cmp  [xcordi],24        ; col. = 0 ?
                je   chang_l            ; if yes then goto old line
                jmp  wr_1               ; to del one character
chang_l:        mov  [xcordi],25
                mov  cx,18              ; set new count
                set_cur [xcordi],0
                mov  si,offset fname    ; begin buffer
                jmp  fix_posi
wr_1:           set_cur [xcordi],0      ; posirion of current cursor
                inc  cx                 ; increment counter
                dec  si                 ; clear one buffer
fix_posi:       write
                jmp  stand_11
                ;===================
                ; RETRUN CHECK KEY =
                ;===================
r_trun:         cmp  ah,28          ; key retrun
                je   process
                ;===============
                ; FILE PROCESS =
                ;===============
                mov  [si],al              ; get file_name
                inc  si
                inc  [xcordi]
                write
                set_cur [xcordi],[ycordi]
                dec  cx
                cmp  cx,00
                jne  stand_13
                jmp  new_char
stand_13:       jmp  stand_11
process:        inc  si
pro:            mov  al,0
                mov  [si],al

                call telephone_num
                cmp  [dir_check],2
                je   tel_F3
                cmp  [dir_check],1
                je   tel_a
                jmp  open_dir
tel_F3:         jmp  out_F3
tel_a:          jmp  out_a
open_dir:       open_file fname         ; open file name
                jc   failure            ; jmp if error open file
                mov  handle,ax          ; number of file_name

                call set_send_ic
                jmp  conti              ; to read file
failure:        call fail_open
                jmp  exit_s_file

conti:          read_file handle,len_buffer,buffer
                jc   failure_read
                cmp  ax,cx
                jz   no_end_file        ; jmp if = 1024 byte
                cmp  ax,0
                jnz  no_end_file
                jmp  end_file

no_end_file:    call on_ry2
                call fill               ; display charecter
                call send_data_file ; send data to modem2
                call check_block    ; protocal
                cmp [error_type],1  ; if error then mov file old pointer
                jz  no_end_file     ; again send
                jmp conti
end_file:       call end_of_file

close_send:     mov  ah,3EH             ; close file
```

```
                mov  bx,handle
                int  21H

                jmp  exit_s_file
failure_read:   call fail_read
                jmp  exit_s_file
failure_block:  call fail_block

exit_s_file:    mov al,28h        ; off hold line
                mov dx,03E4H
                out dx,al
                ret
s_file          Endp
                ;===============
                ; ON FREQUECY =
                ;===============
on_ry2          Proc Near
                mov dx,03E4H
                mov al,0DH
                out dx,al
                ret
on_ry2          Endp
                ;==============
                ; END_OF_FILE =
                ;==============
end_of_file     Proc Near

                mov al,04     ; EOF
                mov dx,03E0H
                out dx,al       ; out head transmitter

                ret
end_of_file     Endp

                ;============================
                ; SET 8251 WORK TRANSMITTER =
                ;============================
set_send_ic     Proc Near

                mov dx,03E2H        ; set baud rate
                mov al,[baud_rate]
                out dx,al

                mov dx,03E4H
                mov al,09H          ; ry1,ry2 off
                out dx,al

                mov dx,03E6H
wait_set:       in  al,dx       ; receive carrier CDT
                test al,04h
                jz wait_set


                mov [time],54    ; delay 3 sec.
                call time_delay

                ret
set_send_ic     Endp
                ;==============================
                ; SEND DATA TO LINE TELEPHONE =
                ;==============================
send_data_file  Proc Near
                mov si,offset buffer
                mov [time],10
                call time_delay
                mov al,01H     ; SOH   head block out
                mov dx,03E0H
                out dx,al      ; out head transmitter

                inc dx         ; port 03E1H
wait_status_in: in al,dx
                test al,01h
                jz wait_status_in

                mov cx,block    ; charecter 1024 byte
in_status:      mov dx,03E0H
```

```
                mov al,[si]
                out dx,al       ; data to line telephone
                inc si

                inc dx          ; port 03E1H
next_char:      in al, dx
                test al,01h
                jz next_char
                loop in_status  ; if cx = 00 then end block
                ret
send_data_file  Endp

                ;====================
                ; CHECK BLOCK ERROR =
                ;====================
check_block     Proc Near

                mov dx,03E4H
                mov al,09H    ; ry2 off
                out dx,al
                mov [time],5
                call time_delay

                mov dx,03E6H
wait_cdt:       in al,dx        ; receive carrier cdt
                test al,04h
                jz wait_cdt

                 mov dx,03E4H
                 mov al,0DH
                 out dx,al

                mov dx,03E1H  ; port status
no_rec:         in al,dx
                test al,02H
                jz no_rec

                mov dx,03E0H
                in al,dx        ; in error check
                cmp al,21       ; check error   NAK
                jz err_faile    ; yes then jmp to error
                mov [error_type],0 ; no error
                jmp wait_delay

err_faile:      mov [error_type],1
wait_delay:     mov [time],10
                call time_delay

carrier_send:   mov dx,03E4H
                mov al,0DH       ; send carrier
                out dx,al
exit_check:     ret
check_block     Endp
                ;============================
                ; TELEPHONE TO ANSWER USER =
                ;============================
telephone_num   Proc Near

                INCLUDE TEL.PRO

telephone_num   Endp

                ;++++++++++++++++++++
                ;  Directory File  +
                ;++++++++++++++++++++
directory       Proc near
                mov   [page_dir],0
                mov   [xcordi],1   ; set cursor position x = 1
                mov   [ycordi],0   ; set cursor position y = 0
                mov   di,02        ; row =2 , col. = 1
                push  di           ; pass parameter to subroutine message
                mov   [check3],11  ; mov check  select display
                call  message      ; write [send file] directory:
                pop   di           ; retrun  sp
n_char:         mov  [xcordi],43   ; maxinum of cursor position
```

```
                mov  [ycordi],0
                set_cur [xcordi],[ycordi]
s_pace:         mov  cx,19
                mov  al,' '             ; clear only one character
s_pace_1:       dec  [xcordi]           ; back to the old character
                write                   ; write character to screen
                set_cur [xcordi],[ycordi]
                loop s_pace_1
                inc  [xcordi]
                set_cur [xcordi],0
                mov  si,offset fname    ; address directory name
                mov  cx,18
                mov  [xcordi],25
                set_cur [xcordi],0
in_key:         mov  [dir_check],0 ; key F not press
                mov  ah,01h        ; function 01 = keyboard status
                int  16h           ; transfer to rom driver
                jz   in_key        ; loop if no key wait
                read               ; function 00h int 16h read char.
                cmp  ah,60         ; F2
                jne  k_F2
                mov  [m_tran],0    ; again menu
                mov  [dir_check],1 ; check key F press
                jmp  exit_dir
k_F2:           cmp  ah,61         ; F3
                jne  back_dir
                mov  [m_tran],1    ; exit
                mov  [dir_check],2
                jmp  exit_dir
                ;=============
                ; BACK SPACE =
                ;=============
back_dir:       cmp  ah,14         ; check back space
                jne  d_trun        ; if not then goto check retrun key
                mov  al,' '        ; clear only one character
                dec  [xcordi]      ; back to the old character
                cmp  [xcordi],24   ; col. = 0 ?
                je   d_chang       ; if yes then goto old line
                jmp  d_wr          ; to del one character
d_chang:        mov  [xcordi],25   ; fix position
                mov  cx,18         ; set new count
                set_cur [xcordi],0
                mov  si,offset fname
                jmp  dir_fix
d_wr:           set_cur [xcordi],0 ; posirion of current cursor
                inc  cx
                dec  si
dir_fix:        write              ; write character to screen
                jmp  in_key        ; to get new character
                ;===================
                ; RETRUN CHECK KEY =
                ;===================
d_trun:         cmp  ah,28         ; key retrun ?
                je   dir_pro       ; if yes to process string directory
                ;=============
                ; DIR BUFFER =
                ;=============
                mov  [si],al       ; get dir_name to buffer
                inc  si            ; inc buffer
                inc  [xcordi]      ; inc position column
                write              ; write character that current cursor
                set_cur [xcordi],[ycordi]
                dec  cx            ; dec count
                cmp  cx,00
                jne  dir_13        ; if cx <>0 then goto dir_13
                jmp  n_char        ; to clear string and get new string
dir_13:         jmp  in_key        ; to get new character
dir_pro:        inc  si            ; put 0 that end string
                mov  al,0
                mov  [si],al
                ;=================================
                ; SEARCH FOR FIRST MATCH DIRECTORY =
                ;=================================
                mov  ah,1ah              ; function = set dta
                mov  dx,offset dir_buffer ; address buffer
```

```
                int  dos21h
                mov [xcordi],2
                mov [ycordi],2
                mov  di,offset file_buffer
                mov  ah,4eh             ; function = search
                mov  cx,0               ; attribute 'normal'
                mov  dx,offset fname    ; address of ASCIIZ
                int  dos21h
                jc   exit_dir           ; to display file not found
                                        ; if no_match
start_dir:      mov  cx,20              ; set count for filename
                mov  bl,00              ; count number of filename
match:          mov  si,offset dir_buffer+30
                        ; offset address that byte 30-42
                        ; filename and extension in form of ascii string
                push cx                   ; save count file name
                mov cx,13                 ; count filename "."-extention
get_file_dir:   mov al,[si]               ; transfer string to display
                mov [di],al               ; filename to buffer
                set_cur [xcordi],[ycordi]
                write                     ; write to current position
                inc [xcordi]              ; increment position column
                inc si                    ; increment buffer
                inc di                    ; increment filename buffer
                loop get_file_dir
                call math_dir             ; to operating table
                mov cx,13
dir_clear:      mov al,' '
                mov [si],al
                dec si
                loop dir_clear
                mov ah,4fh          ; Search for next match
                int dos21h
                jnc d_match         ; Carry flag = clear to jmp
                pop cx
                mov [page_dir],0    ; dir space no file on disk
                jmp select_         ; if no match then to select file
d_match:        pop cx
                inc bl              ; count filname
                loop match          ; get next file name
                mov [page_dir],1    ; next dir wait to come display
select_:        call select_file
exit_dir:       ret
directory       Endp
                ;---------------------------
                ;   MOV POSITION TO EACH FILE
                ;---------------------------
 math_dir       Proc near
                push dx
                mov al,[xcordi]
                mov dl,3
                add al,dl                 ; increment position 3 position
                mov [xcordi],al
                cmp al,79
                jc less
                mov [xcordi],2
                inc [ycordi]
less:           pop dx
                ret
math_dir        Endp
                ;------------------
                ; Select File Name
                ;------------------
select_file     Proc near
set_position:   mov [arrow_ud],0
                mov [arrow_rl],0
                mov [xcordi],2
                mov [ycordi],2
                call find_xy        ; find _x_ position and _y_ position
wait_key:       set_cur [xcordi],[ycordi]
                mov  ah,01h         ; function 01 = keyboard status
                int  16h            ; transfer to rom driver
                jz   wait_key       ; loop if no key wait
                read                ; function 00h int 16h read char.
```

```
                cmp  ah,60          ; F2
                jne  d_F2
                mov  [m_tran],0     ; again menu
                mov  [dir_check],1 ; check key F press
                jmp  exit_select
d_F2:           cmp  ah,61          ; F3
                jne  arrow_up
                mov  [m_tran],1     ; exit
                mov  [dir_check],2
                jmp  exit_select
arrow_up:       cmp  ah,72           ; key arrow up
                jne  arrow_down
                cmp  [arrow_ud],0
                je   no_up
                dec  [arrow_ud]
                dec  [ycordi]
no_up:          jmp  wait_key
arrow_down:     cmp  ah,80           ; key arrow down
                jne  arrow_right
                cmp  [page_dir],1
                jz   check_down
no_next:        mov  al,[yposition]
                sub  al,[arrow_ud]
                cmp  al,0
                jz   no_down
                cmp  al,1
                jz   to_xposi
down:           inc  [arrow_ud]
                inc  [ycordi]
                set_cur  [xcordi],[ycordi]
                jmp  no_down
to_xposi:       mov  dl,[xposition]
                cmp  dl,[arrow_rl]  ; check x position that limitt by dir
                jc   no_down        ; if full than no move cursor
                jmp  down
check_down:     cmp  [ycordi],5
                jnz  no_next
                cmp  [xcordi],66
                jnz  no_down
                call clear_dir
                call read_file_next
                jmp  set_position
no_down:        jmp  wait_key
arrow_right:    cmp  ah,77          ; key arrow right
                jne  arrow_left
                mov  al,[arrow_ud]
                cmp  al,[yposition]
                jne  no_limitt      ; if <> y_position then no_check x_position
                mov  al,[arrow_rl]
                cmp  al,[xposition] ; check x_position limitt
                je   no_right       ; cursor not mov
                jmp  mov_pos
no_limitt:      mov  ah,[arrow_rl]
                cmp  ah,4           ; check column
                je   new_line_cur
mov_pos:        inc  [arrow_rl]
                mov  al,[xcordi]
                add  al,16
                mov  [xcordi],al
                jmp  wait_key
new_line_cur:   mov  ah,[arrow_ud]
                cmp  ah,3
                je   no_right
                mov  [arrow_rl],0
                inc  [arrow_ud]
                mov  [xcordi],2    ; new line
                inc  [ycordi]
no_right:       jmp  wait_key
arrow_left:     cmp  ah,75          ; key arrow left
                jne  get_name_file
                mov  al,[arrow_rl]
                cmp  al,0
                je   up_line
                dec  [arrow_rl]
```

```
                mov  al,[xcordi]
                sub  al,16
                mov  [xcordi],al
                jmp  wait_key
up_line:        cmp  [arrow_ud],0
                je   no_left
                dec  [arrow_ud]    ; to up line
                mov  [arrow_rl],4
                mov  [xcordi],66
                dec  [ycordi]
no_left:        jmp  wait_key
get_name_file:  cmp  ah,28          ; check key retrun
                jne  no_left
                mov  ax,0000
                mov  al,65
                mul  [arrow_ud]
                mov  cx,ax
                mov  al,13
                mul  [arrow_rl]     ; calculate position of file name
                add  ax,cx          ; di = 13X + 65Y
                mov  di,offset file_buffer
                add  ax,di
                mov  di,ax
                mov  si,offset fname
                inc  si
                inc  si
                mov  al,[si]
                cmp  al,'\'
                je   root_file
                mov  cx,12
full:           mov  al,[di]
                mov  [si],al    ; read filename
                inc  si
                inc  di
                mov  al,[di]
                cmp  al,' '         ; check space character
                je   count_full
                loop full
count_full:     mov  al,0           ; point end of filename
                mov  [si],al
                mov  [dir_check],0

                jmp  exit_select
root_file:      inc  si
                mov  al,[si]
                cmp  al,'\'
                je   _inc
                jmp  root_file
_inc:           inc  si
                jmp  full
exit_select:    ret
select_file     Endp
                ;---------------------------------------
                ;  CLEAR OLD DIRECTORY ON SCREEN MEMORY
                ;---------------------------------------
clear_dir       Proc Near
                mov  [xcordi],2
                mov  [ycordi],2
                mov  al,' '
write_space:    set_cur [xcordi],[ycordi]
                write
                inc  [xcordi]
                mov  ah,[xcordi]
                cmp  ah,78
                jnz  write_space
                mov  [xcordi],2
                inc  [ycordi]
                mov  ah,[ycordi]
                cmp  ah,6
                jnz  write_space
                ret
clear_dir       Endp
                ;---------------------------------------
                ;  READ NEXT FILE TO DISPLAY DIRECTORY
                ;---------------------------------------
read_file_next  Proc Near
```

```
                mov [xcordi],2
                mov [ycordi],2
                mov  di,offset file_buffer
                mov ah,4fh          ; Search for next match
                int dos21h
                mov  cx,20             ; set count for filename
                mov  bl,00             ; count number of filename
read_match:     mov  si,offset dir_buffer+30
                push cx
                mov cx,13
get_dir:        mov al,[si]
                mov [di],al               ; filename to buffer
                set_cur [xcordi],[ycordi]
                write
                inc [xcordi]
                inc si                    ; increment buffer
                inc di                    ; increment filename buffer
                loop get_dir
                call math_dir              ; to operating table
                mov cx,13
clrs:           mov al,
                mov [si],al
                dec si
                loop clrs
                mov ah,4fh          ; Search for next match
                int dos21h
                jnc match_d         ; Carry flag = clear to jmp
                pop cx
                mov [page_dir],0    ; dir space no file on disk
                jmp no_file_d       ; if no match then to select file
match_d:        pop cx
                inc bl              ; count filname
                loop read_match     ; get next file name
                mov [page_dir],1    ; next dir wait to come display
no_file_d:      ret
read_file_next  Endp
                ;===================
                ; FIND POSITION XY =
                ;===================
find_xy         Proc Near
                mov cl,00      ; quotian
                mov al,bl      ; division
sub_:           sub al,5       ; 5 = divisor
                jnc quo        ; to increamant quotian
                jmp xy_position ; to process y value
quo:            inc cl         ; quotian
                jmp sub_
xy_position     mov [yposition],cl ; position of y_position
                mov al,5
                mul cl         ; 5Y
                mov cl,al      ; cl = 5y
                mov al,bl
                sub al,cl      ; X = Z - 5Y   [ z = number of file name]
                mov [xposition],al ; position of x_position
                ret
find_xy         ENDP


                ;====================
                ; CHECK CODE RETRUN =
                ;====================
retrun_code     Proc  Near
                cmp [eof],0     ; first odh in ?
                jne oo_retrun   ;
                cmp al,13       ; 0dh ?
                jne no_re       ;
                inc [eof]       ;
                mov bx,01
                jmp code_re     ;
oo_retrun:      cmp [eof],1
                jne no_re
                cmp al,10       ; 0ah ?
                jne no_re       ;
                mov bx,2        ;
```

```
                mov [eof],0
                jmp code_re     ;
no_re:          mov [eof],0
                mov bx,0
code_re:        ret
retrun_code     ENDP
                ;================
                ; ERROR DISPLAY =
                ;================
;ERROR Open_File
fail_open       Proc Near
                mov  [xcordi],2
                set_cur [xcordi],1
                mov  cx,10
                mov  si,offset er_file
er_1:           mov  al,[si]
                write
                inc  si
                set_cur [xcordi],1
                inc  [xcordi]
                loop er_1
key_wait:       mov  ah,01h
                int  16H
                jz  key_wait
                read
                ret
fail_open       Endp
;ERROR Read_File
fail_read       Proc Near
                push si
                push cx
                push ax
                mov  [xcordi],2
                set_cur [xcordi],1
                mov  cx,10
                mov  si,offset error_read
er_read:        mov  al,[si]
                write
                inc  si
                set_cur [xcordi],1
                inc  [xcordi]
                loop er_read
key_read:       mov  ah,01h
                int  16H
                jz  key_read
                read
                pop si
                pop cx
                pop ax
                ret
fail_read       Endp
;ERROR Block
fail_block      Proc Near
                push cx
                push ax
                push si
                mov  [xcordi],2
                set_cur [xcordi],1
                mov  cx,10
                mov  si,offset er_block
er_2:           mov  al,[si]
                write
                inc  si
                inc  [xcordi]
                set_cur [xcordi],1
                loop er_2
key_block:      mov  ah,01h
                int  16H
                jz  key_block
                read
                pop si
                pop ax
                pop cx
                ret
```

```
fail_block      Endp
;=========================END [1] TRANSMITTER=================================
                ;---------------
                ; RECEIVER DATA
                ;---------------
num2            Proc near

                push  cs
                pop   ds
                mov   di,00
                push  di            ; pass parameter to subroutine disp
                mov   [check4],3    ; border receive
                call  disp          ; to display border receive
                pop   di            ; retrun sp
                mov   [xcordi],1    ; set cursor position x = 1
                mov   [ycordi],0    ; set cursor position y = 0
                mov   di,02         ; row =2 , col. = 1
                push  di            ; pass parameter to subroutine message
                mov   [check3],12   ; mov check  select display
                call  message       ; to write install
                pop   di            ; retrun  sp
                mov   [xcordi],1    ; set cursor position x = 1
                mov   [ycordi],21   ; set cursor position y = 0
                mov   di,3362       ; row =22 , col. = 1
                push  di            ; pass parameter to subroutine message
                mov   [check3],13   ; mov check  select display
                call  message       ; to write install
                pop   di            ; retrun  sp

drak_char:      mov  [xcordi],70
                mov  [ycordi],0
                set_cur [xcordi],[ycordi]
drak_space:     mov  cx,24
                mov  al,' '                 ; clear only one character
rec_space_1:    dec  [xcordi]               ; back to the old character
                write                       ; write character to screen
                set_cur [xcordi],[ycordi]
                loop rec_space_1
                inc  [xcordi]
                set_cur [xcordi],0
                mov  si,offset fname        ; address file-name
                mov  cx,23
rec_stand_11:   mov  ah,01h         ; function 01 = keyboard status
                int  16h            ; transfer to rom driver
                jz   rec_stand_11   ; loop if no key wait
                read                ; function 00h int 16h read char.
                cmp  ah,59          ; F1
                jne  rec_back
                mov  [m_fran],1     ; exit
                jmp  exit_num2
                ;=============
                ; BACK SPACE =
                ;=============
rec_back:       cmp  ah,14
                jne  rec_trun
                mov  al,' '                 ; clear only one character
                dec  [xcordi]               ; back to the old character
                cmp  [xcordi],46            ; col. = 0 ?
                je   rec_chang_1            ; if yes then goto old line
                jmp  rec_wr_1               ; to del one character
rec_chang_1:    mov  [xcordi],47
                mov  cx,23                  ;set new count
                set_cur [xcordi],0
                mov  si,offset fname        ; begin buffer
                jmp  rec_fix_posi
rec_wr_1:       set_cur [xcordi],0          ; position of current cursor
                inc  cx                     ; increment counter
                dec  si                     ; clear one buffer
rec_fix_posi:   write
                jmp  rec_stand_11
                ;====================
                ; RETRUN CHECK KEY =
                ;====================
```

```
rec_trun:       cmp  ah,28          ; key retrun
                je   rec_process
                ;===============
                ; FILE PROCESS =
                ;===============
                mov  [si],al              ; get file_name
                inc  si
                inc  [xcordi]
                write
                set_cur [xcordi],[ycordi]
                dec  cx
                cmp  cx,00
                jne  rec_stand_13
                jmp  drak_char
rec_stand_13:   jmp  rec_stand_11
rec_process:    inc  si
                mov  al,0
                mov  [si],al
                ;-------------------------------------
                ; FUNCTION 3CH CREATE OR TRUNCATE FILE
                ;-------------------------------------
                mov ah,3CH
                xor cx,cx           ; file normal
                mov dx,offset fname
                int dos21h
                jc  error_rec
                mov handle_rec,ax     ; create successful
                ;---------------
                ; RECEIVE DATA
                ;---------------
                call set_receive
rec_again:      call receive_data     ; receive 1k block and display

                jmp exit_num2
error_rec:      jmp drak_char
exit_num2:      ret
num2            Endp
                ;====================
                ; SET 8251 RECEIVER =
                ;====================
set_receive     Proc Near
                mov dx,03E2H
                mov al,[baud_rate]   ; set baud rate
                out dx,al

                mov dx,03E4H
                mov al,0DH          ; ry2 cn
                out dx,al

                mov [time],18     ; set 2 sec.
                call time_delay   ; send carrier delay 2 second

                mov dx,03E4H
                mov al,09H        ; off ry2    receive carrier
                out dx,al

                ; protocal user answer
                mov dx,03E6H
carrier_no:     in al,dx
                test al,04h  ; check carrier detect
                jz carrier_no
exit_re:        ret
set_receive     Endp
                ;================
                ; START RECEIVE =
                ;================
receive_data    Proc near

start_in:       mov [error_type],0

                mov cx,block   ; 1 block = 1k byte
                mov di,offset buffer

                mov dx,03E6H
cdt_in:         in al,dx          ; check cdt in again
```

```
                test al,04h
                jz cdt_in

                mov al,0DH
                mov dx,03E4h
                out dx,al

wait_in:        mov dx,03E1H   ; port status
                in al,dx
                test al,02h    ; check data in
                jz wait_in

get_data:       mov dx,03E0H
                in al,dx       ; get data
                cmp al,01      ; 01h = SOH start of text
                jnz eof_       ; jmp if not SOH
                jmp goto_get

eof_:           cmp al,04      ; eof EOT
                jnz goto_get   ; jmp if end of file
end_eof:        jmp exit_start_in
                ;--- end  check head----

goto_get:       mov dx,03E1H   ; port status
read_st:        in al,dx
                test al,02h    ; check data in
                jz read_st
                mov bl,al

                test al,08h
                jnz  parity_err  ; if = 1 then to display error
                mov al,bl
                test al,20h
                jnz framing_err
                jmp receivedata

parity_err:     nop
                jmp e1_rror
framing_err:    nop
e1_rror:        mov [error_type],1

receivedata:    mov dx,03E0H
                in al,dx       ; get data
                mov [di],al    ; load data to memory buffer
                inc di

                loop goto_get ; in new_data

                mov al,09H
                mov dx,03E4H
                out dx,al

                call fill
                cmp  [error_type],1
                jz   not_save   ; if error not save
                call note_file  ; save data to disk
not_save:       call protocal
                jmp start_in    ; new block again

exit_start_in:  mov al,28H     ; end receive cut hold line telephone
                mov dx,03E4H
                out dx,al
                ret
receive_data    Endp
                ;============
                ; SAVE FILE =
                ;============
note_file       Proc Near

save_file:      mov ah,40h           ; function save file
                mov bx,handle_rec
                mov cx,block         ; save 1k byte
                mov dx,offset buffer ; save from buffer
```

```
                int dos21h
                jc  error_rec_note

                jmp no_error_save
error_rec_note: call fail_block
                mov bh,1
                jmp exit_note
no_error_save:  mov bh,0              ; save ok
exit_note:      ret
note_file       Endp
                ;================
                ; OFF FREQUENCY =
                ;================
off_frg         proc Near
                mov al,09h
                mov dx,03E4H
                out dx,al
                ret
off_frg         Endp
                ;==================
                ; PROTOCAL ANSWER =
                ;==================
protocal        Proc Near

                mov [time],3
                call time_delay

                mov  dx,03E4H
                mov  al,0DH
                out  dx,al
                mov  [time],6
                call time_delay
                push bx         ; get check error save disk

                cmp [error_type],1
                jnz not_error_rec
                mov bh,021        ; NAK error
                jmp send_check
not_error_rec:  mov bh,06         ; ACK not error

send_check:     mov al,bh          ; send 1 data check error
                mov dx,03E0H       ; port 8251
                out dx,al

                mov dx,03E1H     ; chcek buffer send emty
txrd_rec:       in  al,dx
                test al,01h
                jz txrd_rec

                mov dx,03E4H
                mov al,09h         ; off ry2
                out dx,al
                pop bx
                ret
protocal        Endp
               ;=========================
                ;  FILL SCREEN BUFFER    =
                ;=========================
fill            Proc Near
                mov  si,offset buffer
                mov  [xcordi],1
                mov  [ycordi],1
get_file:       set_cur [xcordi],[ycordi] ; set current cursor
                mov  al,[si]
                call retrun_code     ; 0DH,0AH
                cmp  bx,0
                je   print_retrun
                jmp  in_si
print_retrun:   write                ; write character
in_si:          inc  si
                cmp  si,block
                je   exist_fill      ; display one border
                cmp  bx,1
```

```
                je   get_file
                cmp  bx,2
                je   re_code
                inc  [xcordi]          ; col. = col + 1
                cmp  [xcordi],78       ; col. = 79 ?
                jne  get_file          ; if <> 79 then inc col.
re_code:        mov  [xcordi],1        ; retrun
                inc  [ycordi]
                cmp  [ycordi],21
                jne  read_f
go_next:        scroll 1,1,20,78       ; scroll up one line
                mov  [xcordi],1
                mov  [ycordi],20       ; x = 1 ,y = 21
read_f:         jmp  get_file          ; to get new character
exist_fill:     ret
fill            Endp

                ;***************
                ; INITIAL PART *
                ;***************
initial:        ASSUME ds:interrupt
                mov ah,0fh              ; Get current display mode
                int video               ; return al = display mode
                xor ah,ah               ; funtion = 0
                int video               ; video = 10h set video mode
                mov dx,offset show      ;
                mov ah,fun9             ; functin = 9
                int dos2ih              ; display string begin program

                INCLUDE S_8251.PRO

                ;---------------------
                ; SET INTERRUPT VECTOR
                ;---------------------
                mov  ax,interrupt
                mov  ds,ax                  ; ds -> interrupt
                mov  ax,keyint
                mov  word ptr cs:[old_keyboard],ax
                mov  ax,keyint[2]
                mov  word ptr cs:[old_keyboard+2],ax
                cli                         ; set new_keyboard interrupt
                mov  keyint,offset new_key
                mov  keyint[2],cs
                sti
                mov  dx,offset initial+one  ; End address resident
                int  resident               ;Terminate resident int 27
                ;*************
                ; DATA PART *
                ;*************
show            DB  25 DUP(),201,25 DUP(205),187,0AH,0DH
                DB  25 DUP(),179,25 DUP(178),179,0AH,0DH
                DB  25 DUP(),179,178,'     __PROGRAMMER__     ',178,179,0AH,0DH
                DB  25 DUP(),179,178,'           By           ',178,179,0AH,0DH
                DB  25 DUP(),179,178,'  APIRAK  CHIRASOPONE  ',178,179,0AH,0DH
                DB  25 DUP(),179,178,'       NO. 296326       ',178,179,0AH,0DH
                DB  25 DUP(),179,25 DUP(178),179,0AH,0DH
                DB  25 DUP(),200,25 DUP(205),188,0AH,0DH
                DB  25 DUP(),'  TO COMMUNICATION PROGRAM    ',0AH,0DH
                DB  25 DUP(),'  ..PRESS Ctrl+LeftShift..    ',0AH,0DH,'$',
start           ENDP
cseg            ENDS
                END  start
```

```
Get_cur     Macro
            mov  ah,0fH             ; get current display mode
            int  video              ; int10H
            ENDM                    ; return value to al

Read_po     Macro
            push ax
            push bx
            push dx
            mov  ah,3               ; Read cursor position
            mov  bh,0               ; select page 0
            int  video              ; return ch = starting line for cursor
            mov  coordinate,dx      ;        cl = ending line for cursor
            pop  dx                 ;        dl = column (x coordunate)
            pop  bx
            pop  ax
            ENDM

set_cur     Macro  x,y      ; set position of current cursor
            push  ax
            push  bx
            push  dx
            mov   ah,02h    ; function 2 = position cursor
            mov   bh,00h    ; selsct page 0
            mov   dl,x      ; set x = col.
            mov   dh,y      ; set y = row
            int   video     ; int 10h
            pop   dx
            pop   bx
            pop   ax
            ENDM

read        Macro
            mov  ah,0     ; function 0 = read character
            int  keyboard ; transfer to rom driver (int 16h)
            ENDM

write       Macro
            ; write one character to screen memory
            push cx
            push ax
            push dx
            push bx
            mov  ah,0ah   ; function 0a = write character only
            mov  bh,0
            mov  cx,1
            int  video    ; transfer to rom driver (int 10h)
            pop  bx
            pop  dx
            pop  ax
            pop  cx
            ENDM

scroll      Macro uly,ulx,lry,lrx
            ; initialize window or scroll window contents up
            push ax
            push bx
            push dx
            push cx
            mov  ah,06h   ; function 6 = scroll up
            mov  al,1     ; scroll by one line
            mov  bh,07    ; normal video attribute
            mov  ch,uly   ; upper  left y
```

```
                mov  cl,ulx    ; upper  left x
                mov  dh,lry    ; lower  right y
                mov  dl,lrx    ; lower  right x
                int  video     ; int 10h
                pop  cx
                pop  dx
                pop  bx
                pop  ax
                ENDM

open_file       Macro f_name
                mov  ah,3DH
                mov  al,2
                mov  dx,offset f_name
                int  dos21h
                ENDM

read_file       Macro hd,le_b,b
                mov  ah,3FH             ; read file
                mov  bx,hd              ; handle
                mov  cx,le_b            ; len buffer
                mov  dx,offset b        ; buffer
                int  dos21h
                ENDM
```

```
 old_keyboard   dd  0          ; vecter old keyboard
coordinate      dw  0
sum             dw  0
baud_rate       db  0
head_file       db  0
error_type      db  0
checkI          db  00         ; status resident
check2          db  00         ; status resident
check3          db  00
check4          db  00
dir_check       db  0
m_check         db  00         ; check status menu
m_tran          db  00         ; check status menu transmitte data
sfdis           db  00         ; status save old display
time            db  0
eof             db  0
xcordi          db  0
ycordi          db  0
xoldt           db  0
yoldt           db  0
xoldr           db  0
yoldr           db  0
xposition       db  0
yposition       db  0
arrow_ud        db  0
arrow_rl        db  0
page_dir        db  0
check_error     db  0
rec_check       db  0          ; status transmitter and receiver
error_block     db  0
tel_buffer      db  13 dup( )
dir_buffer      db  43 dup(0)  ; scratch area for use by dos search function
buffer          db  1024 dup(?)
len_buffer      equ $-buffer
file_buffer     db  270 dup(0) ; buffer file_name
fname           db  25 dup(?)
handle          dw  0
handle_rec      dw  0
menu            db  201,24 dup(205),187
                db  186,24 dup( ),186
                db  186,' [1] Transmitte Data    ',186
                db  186,' [2] Receive Data       ',186
                db  186,' [3] Baud rate Set      ',186
                db  186,' [ESC] Exit communicate ',186
                db  186,24 dup( ),186
                db  200,24 dup(205),188
len_menu        equ $-menu

tran            db  201,24 dup(205),187
                db  186,24 dup( ),186
                db  186,' [1] Send File          ',186
                db  186,24 dup( ),186
                db  186,' [2] Main menu          ',186
                db  186,24 dup( ),186
                db  186,'     Select Number      ',186
                db  200,24 dup(205),188
len_tran        equ $-tran

baud_menu       db  201,24 dup(205),187
                db  186,24 dup( ),186
                db  186,' [1] 1200 baud rate bell',186
                db  186,24 dup( ),186
                db  186,' [2] 600 baud rate ccitt',186
```

```
                db  186,24 dup( ),186
                db  186,'     Select Number       ',186
                db  200,24 dup(205),188

telephone       db  201,24 dup(205),187
                db  186,24 dup( ),186
                db  186,24 dup( ),186
                db  186,24 dup( ),186
                db  200,24 dup(205),188
len_tel         equ $-telephone

border          db  201,78 dup(205),187
                db  186,78 dup( ),186   ; mov blank to screen
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  200,78 dup(205),188 ; mov broder to screen
len_border      EQU $-border

border_1        db  201,78 dup(205),187
                db  186,78 dup( ),186   ; mov blank to screen
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  186,78 dup( ),186
                db  200,78 dup(205),188 ; mov broder to screen
len_border_1    EQU $-border_1
emty            DB  4000 dup(0)
emty1           db  4000 dup(0)
low_char        db  '                              Receive Data With 1200'
                db  ' Bit Rate                                           '
character       db  '          Transmitte Data With 1200 Bit Rate'
                db  '..Pressed Character to sent                         '
len_char        equ $-character
charec          db  '      [S] Save File      [T] Transmitte Data'
                db  '     [N] Exit Communication                '
s_name          db  ' [Send File]  File Name:                    '
                db  'Number Of Telephone:                        '
d_name          db  ' [Send File]  Directory:                    '
                db  'Number Of Telephone:                        '
s_help          db  '  [F2] Menu Transmitte Data  [F3] Exit Com'
                db  'munication   [F4] Directory File'
receive_up      db  ' [Receive File] Create FileName Save To Dis'
```

```
                db  'k.:
receive_down    db  ' [F1] Exit To Main Menu'
                db  '
clrscr          db  ' [Crt+Lshift] Exit Program    [ESC] Menu[1]'
                db  '  Cancel Program ..Press _F1_ Key..'
ti              db  '       Smart Modem
len_ti          equ $-ti
til             db  '     Select Number      '
num_tx          db  '  [1] Transmitte Data   '
num_rx          db  '    [2] Receive Data    '
er_file         db  'Error.Open'
er_block        db  'Error..End'
error_read      db  'Error.Read'
ans_hold        db 'ANSWER HOLD LINE'
```

```
                ;************************************
                ; WRITE MESSAGE  TO TRANSMITTE DATA *
                ;************************************
                set_cur [xcordi],[ycordi]
                add sp,2                ; position of ax
                pop di                  ; retrun check to ax
                sub sp,4                ; position old ip
                mov si,00
long:           cmp [check3],1          ; check will display which ?
                je  connect             ; if no then display charec
                cmp [check3],2
                je  low                 ; if check = 2 then display F1
                cmp [check3],3          ; head of transmitter
                je  h_tran
                cmp [check3],4
                je  h_rec               ; head of receiver
                cmp [check3],5
                je  l_rec
                cmp [check3],6
                je  h_send
                cmp [check3],7
                je  l_send
                cmp [check3],8
                je  h_char
                cmp [check3],9
                je  l_char
                cmp [check3],10
                je  h_exit
                cmp [check3],11
                je  h_dir
                cmp [check3],12
                je  h_receive
                cmp [check3],13
                je  l_receive
                mov al,ti[si]           ; if yes then display character
                jmp chc                 ; ah = 0
connect:        jmp chr
low:            mov al,til[si]          ; mov display F1
                jmp chc
h_tran:         mov al,num_tx[si]
                jmp chc
h_rec:          mov al,num_rx[si]
                jmp chc
l_rec:          mov al,charec[si]
                jmp chc
h_send:         mov al,s_name[si]
                jmp chc
l_send:         mov al,s_help[si]
                jmp chc
h_char:         mov al,character[si]
                jmp chc
l_char:         mov al,low_char[si]
                jmp chc
h_exit:         mov al,clrscr[si]
                jmp chc
h_dir:          mov al,d_name[si]
                jmp chc
h_receive:      mov al,receive_up[si]
                jmp chc
l_receive:      mov al,receive_down[si]
                jmp chc
chr:            mov al,charec[si]       ; massage of receiver
```

```
chc:            mov es:[di],al          ; mov character to screen
                inc di
                mov byte ptr es:[di],70h ; mov attribute to screen
                inc di
                inc si
                cmp [check3],5
                jne lit1
                jmp len_
lit1:           cmp [check3],6
                jne lit2
                jmp len_
lit2:           cmp [check3],7
                jne lit3
                jmp len_
lit3:           cmp [check3],8
                jne lit4
                jmp len_
lit4:           cmp [check3],9
                jne lit5
                jmp len_
lit5:           cmp [check3],10
                jne lit6
                jmp len_
lit6:           cmp [check3],11
                jne lit7
                jmp len_
lit7:           cmp [check3],12
                jne lit8
                jmp len_
lit8:           cmp [check3],13
                jne little
len_:           cmp si,len_char
                jne lon
                jmp exit_message
little:         cmp si,len_ti           ; check buffer emty
                jne lon
                jmp exit_message
lon:            jmp long
exit_message:   ret
```

```
                ;==========================
                ; TELEPHONE TO ANSWER USER =
                ;==========================
tel_char:       mov  [xcordi],78    ; maxinum of cursor position
                mov  [ycordi],0
                set_cur [xcordi],[ycordi]
tel_space:      mov  cx,15
                mov  al,' '         ; clear only one character
tel_s_pace_1:   dec  [xcordi]       ; back to the old character
                write               ; write character to screen
                set_cur [xcordi],[ycordi]
                loop tel_s_pace_1
                inc  [xcordi]
                set_cur [xcordi],0
                mov  si,offset tel_buffer
                mov  cx,12
                mov  [xcordi],64
                set_cur [xcordi],0
tel_in_key:     mov  [dir_check],0  ; key F not press
                mov  ah,01h         ; function 01 = keyboard status
                int  16h            ; transfer to rom driver
                jz  .tel_in_key     ; loop if no key wait
                read                ; function 00h int 16h read char.
                cmp  ah,60          ; F2
                jne  tel_k_F2
                mov  [m_tran],0     ; again menu
                mov  [dir_check],1  ; check key F press
                jmp  exit_phone
tel_k_F2:       cmp  ah,61          ; F3
                jne  back_tel
                mov  [m_tran],1     ; exit
                mov  [dir_check],2
                jmp  exit_phone
                ;=============
                ; BACK SPACE =
                ;=============
back_tel:       cmp  ah,14          ; check back space
                jne  tel_trun       ; if not then goto check retrun key
                mov  al,' '         ; clear only one character
                dec  [xcordi]       ; back to the old character
                cmp  [xcordi],63    ; col. = 0 ?
                je   tel_chang      ; if yes then goto old line
                jmp  tel_wr         ; to del one character
tel_chang:      mov  [xcordi],64    ; fix position
                mov  cx,12          ; set new count
                set_cur [xcordi],0
                mov  si,offset tel_buffer
                jmp  tel_fix
tel_wr:         set_cur [xcordi],0  ; posirion of current cursor
                inc  cx
                dec  si
tel_fix:        write               ; write character to screen
                jmp  tel_in_key     ; to get new character
                ;====================
                ; RETRUN CHECK KEY =
                ;====================
tel_trun:       cmp  ah,28          ; key retrun ?
                je   tel_line
                mov  [si],al
                inc  si             ; inc buffer
                inc  [xcordi]       ; inc position column
                write               ; write character that current cursor
                set_cur [xcordi],[ycordi]
```

```
                dec  cx             ; dec count
                cmp  cx,00
                jne  tel_in
                jmp  tel_char       ; to clear string and get new string
tel_in:         jmp  tel_in_key     ; to get new character
tel_line:       call line_tel
                call tel_answer     ; to check line busy
exit_phone:     ret
                ;================
                ; SET HOLD LINE =
                ;================
line_tel        Proc Near
                mov ah,[baud_rate]
                and ah,0FH
                or  al,0F0H
                mov dx,03E2H   ; Ic 741s374 u1
                out dx,al
                mov dx,03E4H   ; IC 741s374 u8
                mov al,09H
                out dx,al
                mov [time],18
                call time_delay
                mov al,0BH
                out dx,al

                push bx
                mov [xcordi],64
                mov si,offset tel_buffer
                mov cx,0007H
tel_tel:        mov al,[si]
                push cx
                mov cx,0004H
                shl al,cl          ; shift left 4 bit
                pop cx
                mov dx,03E2H
                and al,0F0H
                or al,ah
                out dx,al          ; to 741s374->tcm5087->tel.line
                push ax
                mov ah,00
                mov al,[si]
                set_cur [xcordi],0
                inc [xcordi]
                call hex_asc       ; display ascii
                pop ax
                mov bh,0
                add bh,[si]
                mov bl,[si]
                mov [time],5
                call sound
                mov al,0F0H        ; off tone to send
                or  al,ah
                out dx,al
                mov [time],5
                call time_delay  ; delay 1 sec.
                inc si
                loop tel_tel

                mov dx,3E4H
                mov al,09H         ; ry1,ry2=off,tel._line =on
                out dx,al
                pop bx
                ret
```

```
line_tel        Endp
                ;=====================
                ; CONVERT HEX->ASCII =
                ;=====================
hex_asc         PROC    NEAR
                push cx
                push dx
                mov  dh,4
                mov  cl,4

hex_asc1:       rol  ax,cl
                mov  dl,al
                and  dl,0FH
                add  dl,'0'
                cmp  dl,'9'
                jbe  hex_asc2
                add  dl,'A'-'9'-1

hex_asc2:       dec  dh
                jnz  hex_asc1
                mov  al,dl
                write
                pop  dx
                pop  cx
                ret
hex_asc         ENDP
                ;========
                ; SOUND =
                ;========
sound           Proc Near
                push ax
                mov al,0B6H ; 10110110b
                out 43H,al  ; into timer2
tone:           mov ax,bx   ; mov 1/pitch into ax
                out 42h,al
                mov al,ah
                out 42H,al  ; to timer2
                in al,61h
                or al,3
                out 61h,al
                call time_delay
                and al,0FCH  ; off timer2
                out 61h,al
                pop ax
                ret
sound           EndP
                ;=============
                ; DELAY TIME =
                ;=============
time_delay      Proc Near
                push ax
                push cx
                push dx
                sti
                sub cx,cx
                sub dx,dx
                mov ah,1
                int 1AH
chk_time:       mov ah,0
                int 1ah
                cmp dl,[time]
                jb  chk_time
                pop dx
```

```
;*************************************************************************
;Pprt number                                                             *
;                                                                        *
;port(03E0H) Output data  port cpu to 8251                                *
;port(03E1H) Output control port cpu to 8251                              *
;           Input status port 8255 to cpu                                *
;port(03E2H) IC #741s374  output port                                     *
;           bit 4-7 count number telephone                               *
;           Control Tca3101 select standard                              *
;           Bit 0 control txr1                                           *
;           Bit 1 control txr2                                           *
;           Bit 2,3 control trs                                          *
;port(03E4H) IC #741s374  output port                                     *
;           bit 0 transister (Q3)                 '1'= on,'0'= off       *
;           bit 1 analog switch on off tone       '1'= on,'0'= off       *
;           bit 2 analog switch on off op-amp     '1'= on,'0'= off       *
;           bit 3 buffer interrupt on off (741s125) '1'= off,'0'=on      *
;           bit 4 cts of 8251 transmitte ready    '1'=no  ,'0'=yes       *
;           bit 5 transister (Q4)                 '1'= off,'0'=on        *
;           bit 6 don't care                                             *
;           bit 7 reset 8251                      '1'=on  ,'0'=off       *
;port(03E6)  IC #741s244  input port                                      *
;           bit 0 LM567 work                      '1'=off ,'0'=on        *
;           bit 1 TXEMPTY EMTY                    '1'=emty,'0'=no        *
;           bit 2 carrier fail (CDT)              '1'=no  ,'0'=yes       *
;*************************************************************************

            ; D7 D6 D5 D4 D3 D2 D1 D0
            ; 1  1  0  1  1  1  1  0        = DEh
            ; d1,d1 = set baud rate factor   = x16
            ; d2,d3 = set character lendth
            ; d4    = parity enable 1       = enable
            ; d5    = even parity genarate 0 = odd
            ; d6,d7 = set stop bit 11       = two stop bit

            mov dx,03E4H            ; port control reset
            mov al,0A8H    ;Q4=on,Q3=off,ry1,ry2=off,cts=0,reset=1,ct=1
            out dx,al               ; to U8
            mov [time],5
            call time_delay         ; time constant of restet 8251
            mov al,28H              ; reset=0
            out dx,al               ; to U8
            mov dx,03E2h            ; port number 03E2
            mov al,0F8H             ; B3 B2 B1 B0
                                    ; 1  0  0  0   baud rate 1200
            out dx,al               ; control TCM3101 Bit rate Bell 202
            mov al,0DEh             ; de = data control 8251
            mov dx,03E1h            ; output to control 8251
            out dx,al               ; to 8251
            mov al,15h              ; mov control port to al
            out dx,al               ; output to control 8251
```

บทที่ 5

บทสรุปและวิจารณ์

ปัญหาที่เกิดขึ้นจากการทดลองคือ มีความยุ่งยากในการขอใช้เครื่องคอมพิวเตอร์ 2 เครื่องพร้อมกัน และเครื่องโทรศัพท์ ทำให้การทดลองเป็นไปอย่างล้าช้า ซึ่งจากสาเหตุนี้ทำให้การแก้ปัญหาที่เกิดขึ้นในวงจรและส่วนโปรแกรม ไม่สามารถแก้ไขได้ทันตามจุดมุ่งหมายที่ตั้งไว้ จึงได้ทำแค่คาร์ดวงจรที่สามารถติดต่อระหว่างคอมพิวเตอร์ 2 เครื่อง เป็นคาร์ดซีเรียลพอร์ทที่ใช้การส่งข้อมูลแบบ Frequency shift keying (FSK) และสามารถนำไปใช้เป็นตัวต่อโทรศัพท์อัตโนมัติได้

ปัญหาที่เกิดขึ้นอีกประการหนึ่งคือ คุณภาพของตัวไอซีที่ใช้ซึ่งในการทดลองได้เปลี่ยนไอซีไปมากมายรวมทั้งประสิทธิภาพของไอซีบางตัวเช่น เบอร์ 8251 ซึ่งเป็นตัวสำคัญในการติดต่อระหว่างตัวโมเด็มกับคอมพิวเตอร์ ได้ทดลองเปลี่ยนถึง 3 ตัวจึงจะทำให้การรับส่งข้อมูลใช้ได้ดี และเนื่องจากสาเหตุหลักนี้เองทำให้การทดลองต้องล้าช้าไปอย่างมาก ตัวอย่างปัญหาที่ประสบคือไอซีเบอร์ 74LS244 ซึ่งเป็นอินพุทพอร์ททำงานผิดพลาดที่ขา CDT คือตัวไอซีเสีย ทำให้เอาท์พุทออกมาเป็น 1 ตลอด ทำให้โปรแกรมที่เขียนตรวจสอบสัญญาณตรงนี้ผิดพลาดไป

จากการทดลองโดยวิธี Loop back ภายในตัวคาร์ด ปรากฏว่ารับได้ 100% แต่เมื่อใช้ติดต่อระหว่างคอมพิวเตอร์ 2 เครื่อง ปรากฏว่าผิดพลาดไปประมาณ 2%

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปด้วยดีเพราะได้คำปรึกษาจากพี่ อนุศักดิ์ และคุณ สุชาติ แสงหิรัญ ซึ่งขอขอบคุณมา ณ.ที่นี้ด้วย รวมทั้งคุณ อดิเรก สุพรรณวรรษา คุณ เสน่ห์ พุ่มเอี่ยม น้องเก่ง น้องโต้ย ที่พิมพ์ต้นฉบับให้ และที่สำคัญที่สุดคือ คุณ วิชชุริยา เจียรนัย ที่ช่วยแปลบทคัดย่อและช่วยเป็นกำลังใจให้ผู้จัดทำตลอดมา

หนังสืออ้างอิง

1. Lewis C. Eggebrecht."Interfaceing to the IBM Personal Computer", Howard W. Sam. CO.,INC.,1983

2. Howard M. Berlin. "Design of Op-Amp Circuit", Howard W. Sam. CO.,INC. ,pp 165-168, 1980

3. "XT Technical Reference"

4. James W.Coffron. "Z80 Applications",Berkeley,pp.209-237,1983

5. Ray Duncan. "Advanced MS DOS",16011 N.E.36th Way,Box 97017 ,Redmond,Washington 98073-9717,1986

6. Russell Rector_George Alexy. "The 8086 Book Includes the 8088",Osbore/Mcgraw-Hill Brekeley

7. John Uffenbech,"The 8066/8088 Family design program and interfacing",Prentice Han Internation Edition,1987.

8. Leo J.Scanlon, "IBM PC Assembly language A Guide For Pro grammer",Robert J.Brady Co.

9. Jsk Group, "แอสแซมบลี้ 8086/8088",ฟิสิกส์เซ็นเตอร์การพิมพ์

10. ยืน ภู่วรวรรณ." เทคโนโลยี ไมโครคอมพิวเตอร์ 16 บิท",ซีเอ็ดยูเคชั่นจำกัด

11. น.ต. ดร. ไพศาล สงวนหมู่ และ รศ. ยืน ภู่วรวรรณ,"การสื่อสารข้อมูลและไมโครคอมพิวเตอร์เนตเวิร์ค,บริษัท ซีเอ็ดยูเคชั่น จำกัด

12. สุริยัน ศรีสวัสดิ์กุล,"ระบบสื่อสารข้อมูลคอมพิวเตอร์",ฟิสิกส์เซ็นเตอร์.