ปีการศึกษา 2533

ระบบอุปกรณ์ข้าวสารแบบโครงข่าย

DATA TEXT TERMINAL NETWORK

โดย

นาย ขจรเกียรติ	ธรรมบุตร	326403
นาย คเณศ	สถิตกิจพิเชษฐ์	326404
นาย พีระ	ลัพธิวณิชย์	326419
นาย วรพจน์	อัศวชัยพร	326420

อาจารย์ที่ปรึกษา

อาจารย์ ภากร หุตะสังกาศ

ระบบอุปกรณ์แสดงข่าวสารแบบโครงข่าย

นาย ชจรเกียรติ ธรรมบุตร
นาย คเณศ สถิตกิจพิเชษฐ์
นาย พีระ ลัพธิวณิชย์
นาย วรพจน์ อัศวชัยพร
อาจารย์ที่ปรึกษา
นาย ภากร หุตะสังกาศ
ปีการศึกษา 2533

บทคัดย่อ

DATATEXT TERMINAL NETWORK เป็นระบบช่วยวงจรสื่อสารเพื่อช่วยในการบอกข่าวสารโดยทั่วไปแก่ สาธารณชน โดยมี HOST COMPUTER เป็นตัวสร้าง, เก็บข่าวสารทั้งหมดได้ และถ่ายทอดข่าวสารในรูปของไฟล์ให้เก็บพักไว้ที่ FRONT END PROCESOR(FEP) ทำหน้าที่ ในการเก็บข่าวจาก HOST COMPUTER ที่เป็น FILE ที่ต้องการไว้ก่อนแล้วจึงจะกระจายไป ยัง TERMINAL เมื่อมีการ REQUEST ขอมา FEP เป็นตัวกลางระหว่าง HOST กับ TERMINAL ในการติดต่อส่งผ่านข่าวสาร การที่ใช้ FEP จะช่วยทุ่นงานของ HOST COMPUTER ในการที่ HOST สามารถทำงานอื่นๆ ได้ด้วยโดยไม่ต้องมายุ่งเกี่ยวกับการติดต่อกับ TERMINAL หลายๆ ตัว FEP จึงรับผิดชอบงานในการควบคุม LINEข่าวสาร MESSAGE ไปให้แก่ TERMINAL ตามที่ต้องการได้เลย ตัว TERMINAL ทำหน้าที่ในการแสดงข่าวสารออกจอภาพเพื่อให้ผู้ใช้สามารถรับบริการจากระบบ DIN นี้TERMINAL จะมี KEYPAD สำหรับใช้เลือก MENU CHOICE เมื่อ TERMINAL รับค่า KEYแล้วก็จะทำการ LOAD ข่าวที่ต้องการมาจาก FEP โดยผ่านทาง SERIAL PORT RS232โดยติดต่อกับแบบ FULL DUPLEX แบบมี PROTOCOLFEP จะส่ง FILE ซึ่งแบ่งออกเป็นBLOCK มาออกทาง SERIAL PORT เมื่อ TERMINAL ได้รับ MESSAGE แล้วก็นำมาเก็บไว้ในหน่วยความจำ แล้วจึง นำไป DISPLAY ออก CRT MONITOR

ข่าวสารที่บริการจาก TERMINAL จะเป็นพวก INFORMATION ต่างๆ เช่น ข่าวสารทั่วไป ราคาสินค้าในท้องตลาด, ข้อมูลทางสถิติต่างๆ เป็นต้น
ในการควบคุมระบบข่าย NETWORK จะควบคุมด้วย FEP ซึ่งเป็นศูนย์กลางใน

การติดต่อระหว่าง FEP กับ TERMINAL ทุกตัวในกรณีที่มี TERMINAL หลายๆ ตัว FEP จะต้องติดต่อกับทุกตัวเช่นเดียวกัน โดยสลับไปบริการทีละตัวอย่างต่อเนื่องโดยไม่ให้เกิดการขาดช่วงไปเลย ดังนั้น SOFTWARE ทาง FEP จึงต้องทำงานแบบ MULTITASKING

ABSTRACT

Data Terminal Network (DTN) is the assist communication circuit system that news was displayed by Host Computer to public stored all text and transfer to the terminals into files. On Data Communication Front End Processor (FEP) is a media between Host and Terminals when we haved requested data. FEP was used to save time of host computer. Host computer can operate independent other job and not concern any terminal. So FEP response to line control that connect to the terminal. When terminal request to FEP, FEP will be response and transfer message file to terminals. Terminal is used to display message on the screen. DTN is serviced to users. Terminals have keypad for select menu choice. The news from FEP was loaded when terminal have inkey by RS-232 serial port. Protocal in full duplex communication haved been used. FEP will be sent a block of file through serial port. When terminal take the message it stored in memory unit and display to CRT monitor.

The data text is several information, news, price of goods, statistical data etc.

Data text terminal network will be controled by FEP which is the center of communication between FEP and Terminal.

Many terminals will be controled alternately to service each data by FEP. So FEP used the software that can operate on multitasking.

กิตติกรรมประกาศ

ในการออกแบบการสร้างระบบ DATA TEXT TERMINAL NETWORK นี้ ผู้จัดทำได้มีความรู้ และ ความเข้าใจถึง การออกแบบและการนำเอาไมโครโปรเซสเซอร์มาประยุกต์ใช้ได้มากขึ้น ซึ่งต้องขอขอบคุณอาจารย์ ภากร หุตะสังกาศ อาจารย์ที่ปรึกษาโดยตรง ที่ได้กรุณาให้คำแนะนำปรึกษา ตลอดจนจัดหาเครื่องมือมาใช้ในการทดลอง และ ขอขอบคุณคณะอาจารย์ประจำภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง ทุกท่านที่ให้คำแนะนำและข้อเสนอแนะเกี่ยวกับโครงงานนี้

นาย ขจรเกียรติ ธรรมบุตร
นาย คเณศ สถิตกิจพิเชษฐ์
นาย พีระ ลัพธิวณิชย์
นาย วรพจน์ อัศวชัยพร

< ผู้จัดทำ >

สารบัญ

บทที่ 1
บทนำ

ข่าวสารและข้อมูล นับว่าเป็นสิ่งที่ทุกคนต้องมีความเกี่ยวข้องตลอดเวลา ในชีวิตประจำวันมนุษย์ย่อมต้องคลุกคลีกับเรื่องราวต่างๆ บางครั้งก็ต้องมีการบันทึกและจดจำ สิ่งต่างๆ เอาไว้เพื่อนำมาใช้งานอีก ในโลกปัจจุบัน ธุรกิจและอุตสาหกรรมมีความตื่นตัวกันมากมีการแข่งขันการตลอด การโฆษณา และระบบการบริหารที่ดแนวรัดกุม ก็ย่อมจำนำพาให้กิจการนั้น เจริญรุดหน้าได้มากกว่า เพราะฉะนั้น สิ่งหนึ่งที่สำคัญอย่างยิ่ง แนวคุมชัยชนะให้แก่การบริหารงานธุรกิจและอุตสาหกรรมเหล่านี้ ก็คือ ข่าวสารและข้อมูล เสมือนปรัชญาของสิ่งที่กล่าวไว้ว่า รู้เขารู้เรา รบร้อยครั้ง ชนะร้อยครั้ง

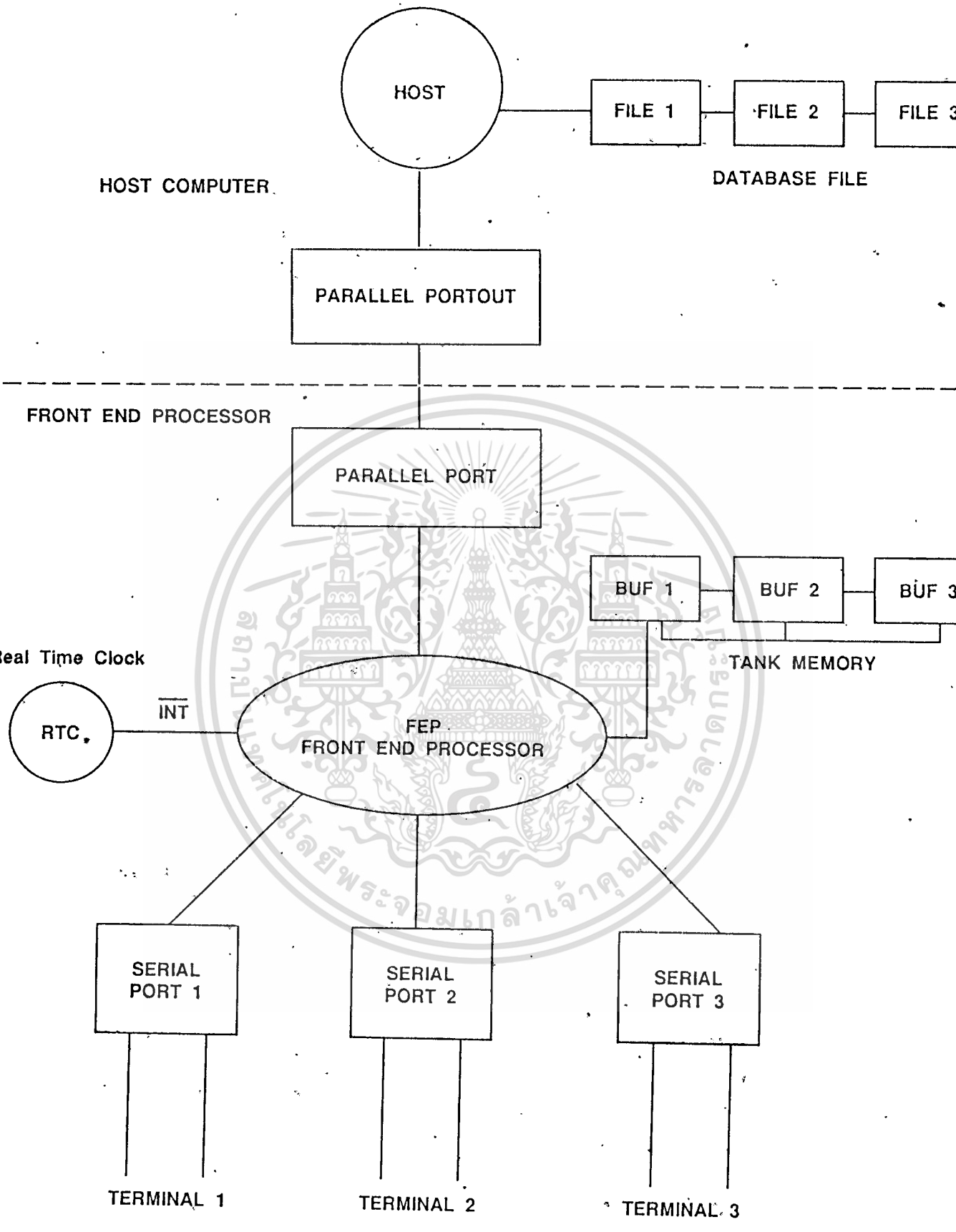
ดังนั้น ปริญญานิพนธ์ ชุดนี้จึงจัดเสนอโครงงานในชุดอุปกรณ์บริการข่าวสารแบบโครงข่าย DATATEXT TERMINAL NETWORK ซึ่งสามารถนำมาประยุกต์ในการนำเสนอบริการข่าวสารให้กับผู้ใช้ TERMINAL โดยติดต่อทาง KEYPAD ซึ่งมีฟังก์ชันให้เลือกในการเรียกใช้บริการ ในการทำงานของ TERMINAL แต่ละตัว จงมี CPU ของตัวมันเองทำงานควบคุม ในการแสดงผลออกทางจอภาพ รับส่งสิ่งทาง KEY ขณะติดต่อกับ NETWORK ทาง SERIAL PORT ใน TERMINAL แต่ละตัวจะรับผิดชอบในการทำงานของตัวมันเองในการโหลดข่าวสารจากคลังเก็บข้อมูล จาก FRONT END PROCESSOR โดยโหลดมา ทีละ FILE มาเก็บใน BUFFER ภายใน แล้วจึงนำข้อมูลมาแสดงผลทางจอภาพติดต่อทางสายโทรศัพท์ อีกที ทีละ PAGE โดยใช้คีย์เลือก

ตัวควบคุมการ LOAD ข้อมูลให้กับ TERMINAL นั่นคือ FRONT END PROCESSOR (FEP) จะทำงานเสมือนกัน CONCENTRATOR ที่รับข้อมูลมาจาก HOST แล้วสั่งข้อมูลDATA กระจายแจกจ่ายไปสู่ TERMINAL ทั้งหลายต่อไป FEP จะมี CPU ควบคุมการทำงานอีกเช่นเดียวกัน ทั้งนี้ เนื่องจาก FEP ต้องการประมวลผล เนื่องจากต้อง แปลงข้อมูลที่รับมาจาก HOST จาก PARALLEL PORT และแปลงเป็น SERIAL ส่งออก ไปทางSERIAL PORT ไปให้แก่ TERMINAL นอกจากนี้ FEP ยังต้องคอย SWITCH งานดู แลการร้องขอข้อมูลมาจาก TERMINAL ในการรับข่าวสาร DOWN – LOAD หรือ UPPRIE จาก HOST แล้วจึงจะรับการ REQUEST มาจาก TERMINAL แล้วจึงเลือก DATA ที่จะส่ง ไปให้กับ TERMINAL ต่อไป ดังนั้น FEP จึงต้องมี MEMORY สำรองขนาดใหญ่พอสมควรซึ่ง ต้องมากกว่าของ TERMINAL

ส่วนทางด้าน HOST ซึ่งให้ PC XT หรือ AT ก็ได้ ซึ่งมีวงจร INTERFACE PARALLEL CARD เสียบขั้ว SLOT ทำหน้าที่ ในการ TRANSFER ข้อมูลจาก PC ไปเก็บไว้ที่ FEP ส่งต่อไปให้ TERMINAL ต่อไปหน้าที่ของ HOST จะมีโปรแกรม EDIT MESSAGE

ซึ่งเขียนด้วย DBASE ทำหน้าที่ในการ UPDATE DELETE และ EXCHANGE DATA และจัดเก็บไฟล์เอาไว้ไม่ให้สูญหาย เมื่อ FEP มีการ REQUEST ก็สามารถเรียกหามาให้ได้ เมื่อ HOST TRANSFER ข่าวไปให้แก่ FEP เสร็จแล้ว HOST ยังสามารถทำงานอื่นต่อไปได้ FEP จัดการกับ TERMINAL เอง

หลักการทำงานของระบบ DATATEXT TERMINAL NETWORK



อธิบายการทำงานจาก BLOCK DIAGRAM

● HOST COMPUTER ใช้ PERSONAL COMPUTER IBM PC/XT COMPATIBLE จะมีโปรแกรม 2 FILE ที่ทำงานในส่วนนี้คือ

SEND.EXE

DATATEXT.PRG

SEND.EXE ทำหน้าที่ในการส่งข้อมูล FILE DATABASE .DBF ไปยัง FEP ทาง PARALLEL PORT การทำงานของโปรแกรมจะทำการผ่านข้อมูลจาก DISK และ TRANSFER เก็บไว้ใน MEMORY จากนั้นจึง DUMP ออกทาง PORT เหมือน PRINT ออก PRINTER โดยปกติในกรณีถ้า FEP ไม่ได้ถ่ายอยู่หรือไม่พร้อมก็จะ RESPONSE ตอบกลับสู่ PC ว่าไม่ว่าง หรือไม่ได้ถ่ายอยู่

DATATEXT PRO. จะเรียกโดยผ่านทาง DBASE III PLUS โดยเข้าสู่โปรแกรม DBASE แล้วเรียกคำสั่ง DO XXFILENAME XX โปรแกรมนี้จะทำงานทางด้าน OPERATOR คือเกี่ยวกับการสร้างข้อมูล, UPDATE, DELETE, INSERT, SORT, SEARCH DATA โดยจะสร้างรูปแบบการแสดงหน้าจอแบบ POP UP, PULL DOWN MENU ซึ่งทำให้เลือกใช้ได้อย่างง่ายสะดวกยิ่งขึ้น ในการจัดเก็บข้อมูลจะมีอยู่ 3 ไฟล์โดยแต่ละไฟล์จะมีโครงสร้างแตกต่างกัน โปรแกรมนี้จะเรียกไฟล์เหล่านี้มาใช้งานตลอดจนทั้งการ TRANSFER ส่งออกทาง PARALLEL PORT ด้วยการติดต่อทาง PARALLEL PORT จะมีสัญญาณ DATABUS DØ-D7, ACKNOWLEDGE STROBE และ BUSY วิธีการส่งข้อมูล PC จะรอรับสัญญาณ $\overline{ACK}$ ถ้าเป็น "LOW" ก็จะส่ง DATA ออกไปเสร็จแล้วจึงส่ง $\overline{STB}$ เป็น "LOW" ให้แก่ FEP ต่อจากนั้น PC จะรอรับ $\overline{ACK}$ ใหม่อีกครั้งหนึ่งถ้า FEP ส่ง BUSY เป็น "HIGH" ก็แสดงว่า FEP กำลังไม่ว่างให้ PC รอไว้ก่อนจนกว่า BUSY เป็น "LOW" และมีสัญญาณ $\overline{ACK}$ เป็น "LOW" ก็จะส่งข้อมูลไบต์ต่อไปได้อีก ซึ่งจะเป็นไปตามวัฏจักรเหมือนในตอนแรก

- **FEP FRONT END PROCESSOR**

CPU จะถูกจัดให้อยู่ในการทำงานแบบ 'MULTITASKING' โดยมี REAL TIME CLOCK INTERRUPT ให้แก่ CPU ตลอดเวลา ดังนั้น CPU จะทำงานโดยเปรียบเสมือนงานแต่ละงานเป็น SUBROUTINE หนึ่งของโปรแกรม CPU จะต้องถูก INTERRUPT ให้ไปตรวจสอบหรือสแกนงานแต่ละงานอยู่เสมอ ตามลำดับ PRIORITY ของงานนั้น โดย CPU จะจัดลำดับไว้ดังนี้

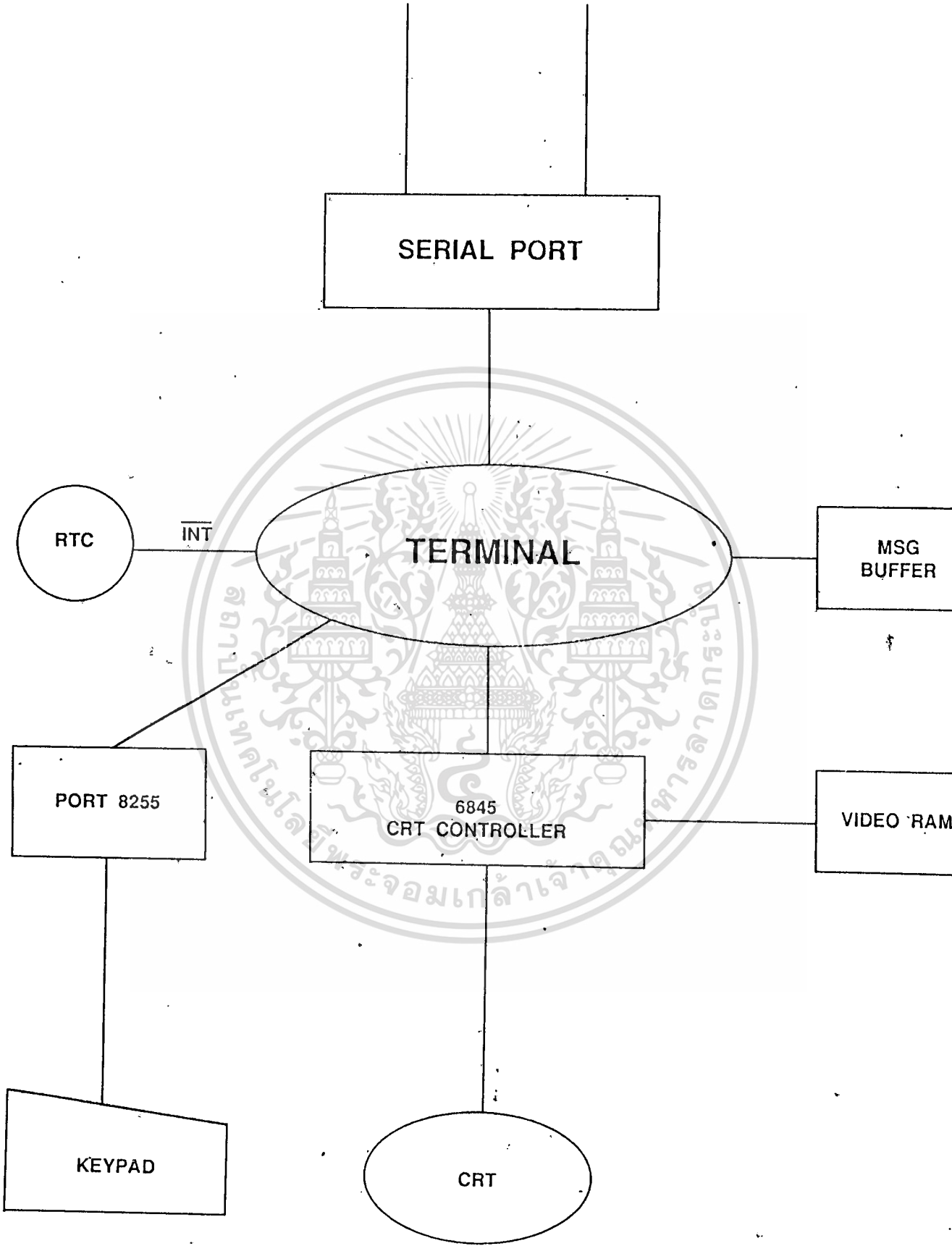
- PARALLEL PORT INPUT
- SERVICE SERIAL PORT
- DISPLAY 7 SEGMENT LED
- SCAN KEY

วิธีการทำงานของ CPU จะมีโปรแกรม SCHEDULER เป็นตัวจัดการแบ่งงานและเลือกงานมาให้ CPU โปรเซสตามช่วงเวลาของการอินเตอรัปท์ สมมติเช่น เมื่อ INT ที่ 1 SERIAL PORT มีการ REQUEST SCHEDULER จะไปเลือก SERIAL ROUTINE ในขณะที่ยังทำงานไม่เสร็จ แต่ถูก INT ที่ 2 เข้ามาอีก ปรากฏเช็คพบว่า PARALLEL PORT มีการ REQUEST SCHEDULER จะจัดให้ HIGH PRIORITY ได้ก่อน จึงทำการเลือก CALL PARALLEL ROUTINE ก่อน เมื่อทำงานจนเสร็จยังมีเวลาเหลือ ก่อนที่จะถูก INTERRUPT อีกก็จะกลับคืนสู่ SERIAL ROUTINE ตามเดิมเพื่อทำงานที่ค้างอยู่ให้เสร็จ ถ้าหากมี INT ที่ 3 เข้ามาเช็คว่ามี DISPLAY 7 SEGMENT REQUEST เนื่องจากอยู่ LOW PRIORITY กว่า SCHEDULER ก็จะทำงาน ทำงาน SERIAL ROUTINE จนเสร็จก่อน จึงจะไปทำงานของ DISPLAY ROUTINE การที่มี SCHEDULER เป็นตัวคอยจัดการป้อนงานให้ CPU โปรเซสได้อย่างต่อเนื่องเช่นนี้ จะทำให้ CPU ไม่มีเวลาว่างสูญเสียไปโดยเปล่าประโยชน์จากสภาวะ WAIT รออื่น ๆ โดยไม่จำเป็น และประการสำคัญ คือ ต้องการให้ CPU มีการตอบสนองกับการทำงานอย่างทันทีทันใด (ON LINE)

ในส่วนของ SCHEDULER MONITOR PROGRAM และ INTERRUPT ROUTINE จะเป็นส่วน OPERATING SYSTEM ของโปรแกรมใน FEP

อีกส่วนจะเป็นส่วนของ APPLICATION PROGRAM จะทำหน้าที่ทางด้านการตอบรับการติดต่อกับ PARALLEL PORT การนำข้อมูลไปเก็บไว้ใน "TANK MEMORY" การติดต่อทาง SERIAL PORT หลาย ๆ ตัว ทางด้านรับ (RX) ทางด้านส่ง (TX) การตอบถาม PROTOCOL ระหว่าง FEP กับ TERMINAL เมื่อด้าน TERMINAL ENQUIRE DATA มา FEP จะถามว่าต้องการ FILE ไหน TERMINAL จะส่ง FILE CODE มาให้ FEP จึงจะส่งข้อมูลไปให้ TERMINAL ต่อไปจนจบ MESSAGE ด้าน TERMINAL จะต้องตอบรับทราบ ACKNOWLEDGE เมื่อ FEP ส่งครบก็จะส่ง END OF TRANSMISSION มาให้ว่าหยุดการส่งของความแก่ TERMINAL

TERMINAL :



- **TERMINAL**

CPU จะอยู่ในการทำงานแบบ MULTITASKING เหมือนกับ FEP โดยที่ส่วนของ OS ทำหน้าที่ทาง SCHEDULER, MONOTOR PROGRAM และ INTERRUPT ROUTINE อีกส่วนเป็น APPLICATION PROGRAM ทำหน้าที่ทางด้านการรับคำสั่งทาง KEY แล้วไปทำการเปิด TABLE คำสั่ง เพื่อไปอ่านไฟล์จาก MESSAGE BUFFER MEMORY แล้วจึงมาทำการโปรเซสจัดแสดงผลออกหน้าจอ CRT โดยการส่งข้อมูลไปเก็บไว้ที่ VIDEO RAM

ลำดับ PRIORITY ของ TASK คือ

- SERIAL PORT
- KEYBOARD
- DISPLAY CRT

การทำงานจะให้ SERIAL PORT มีความสำคัญสูงเพราะเป็นการติดต่อกับ FEP จะต้องไม่มีความผิดพลาดเลยในการส่งผ่านข้อมูล เมื่อ TERMINAL ได้รับข้อมูลจาก FEP ก็จะนำไปเก็บไว้ใน MESSAGE BUFFER MEMORY เพื่อให้โปรแกรม DISPLAY CRT สามารถดึงข้อมูลไปแสดงผลได้ต่อไป การรับข้อมูลทาง KEY

0 9	เป็นคีย์ค่าตัวเลขที่ต้องการเลือก เช่นในวางใส่ค่าตัวเลขการเลือกหัวข้อ CHOICE
ENTER	ENTER DATA KEY
↑ ← → ↓	ARROW KEY ให้ CURSOR เคลื่อนไหวในทิศทางต่าง ๆ กัน
SCR UP	SCROLL UP เคลื่อนทีละ PAGE ในหน้าจอ
SCR DOWN	SCROLL DOWN

<u>รายละเอียดทาง HARDWARE (วงจร)</u>

HARDWARE ของระบบ DTN จะแบ่งออกเป็น 3 ส่วน คือ

1. HOST COMPUTER ใช้เครื่อง PERSONAL COMPUTER ที่มี PARALLEL PORT (PRINTER PORT) อย่างน้อย 1 PORT

CPU : 8088 PC XT/AT COMPATIBLE

INTERFACE CARD : PARARLLEL PORT 8255 PPI (HAND CHECKKING MODE) PORT A SEND RECEIVE

CONNECTION : สายต่อ 40 เส้นยาว 3 ฟุต

2. FEP FRONT END PROCEEESOR

CPU : Z80A

MEMORY : 32K (LOWER) FOR ROM&RAM

: 32K (UPPER) BLANK MEMORY EXPANSION RAM

BLANK : 32 K

PARALLE PORT : 8255 PPI 1 PORT

SERIAL PORT : 8251 UART 3 PORT

CLOCK : 4Mhz

REAL TIME CLOCK : 8253 GENERATER USER SIGNAL (PROGRAMMABLE TIMER)

PPI = PARALLEL PORT INTERFACE

UART = UNIVERSAL ASYCRONOUS RECEIVE/TRANSMIT

3. TERMINAL

CPU : Z80A

MEMORY : MAX. 64 KByte ROM & RAM

SERIAL PORT : 8251 USART 1 PORT

CRT CONTROLLER : 6845

KEYPAD : 4*4 KEYS

CLOCK : 3.579 MHz

REAL TIME CLOCK : CLOCK DIV 1 mS

บทที่ 2

ทฤษฎีและหลักการทำงาน

ทฤษฎีและหลักของระบบ FRONT END PROCESSOR

FRONT END PROCESSORS ฟร้อนท์ เอนท์ โปรเซสเซอร์

Host Computer เหมาะสมที่สุดในด้านการคำนวณทางตัวเลข การเก็บและ แก้ไขข้อมูล และงานด้านอื่นๆ ที่ใช้เวลา เพียงเศษส่วนของล้านวินาที ถ้าใช้คนทำงานบางที ต้องใช้เวลาเป็นวินาที มันเป็นการเสีย เวลาที่มีค่าไปถ้า Host Computer ต้องใช้เวลาทั้งหมดติดต่อกับเทอร์มินอล

FRONT END PROCESSORS หรือ FEP เป็นตัวช่วยการติดต่อสื่อสารของ Host Computer FEP ควบคุมการติดต่อสื่อสารของ Host Computer ได้ ทั่วโลก เมื่อ Host Computer ทำงานอื่นๆ เช่น คำนวณเงินเดือน FEP มี ขนาดเล็ก ใช้กับคอมพิวเตอร์เฉพาะงาน ซึ่งถูกให้ทำหน้าที่เดียวมันติดต่อกับ Host Computer ด้วยความเร็วสูง (หลายMBPS) ขณะที่เทอร์มินอลทำงานที่ ความเร็วต่ำ (หลาย KBPS) FEP บางทีหมายถึงFront Ends.

เหมือนตัวอย่างต่อไปนี้ ความคิดของศาสตราจารย์ผู้เชี่ยวชาญท่านหนึ่ง คิดว่าการพูด กับนักเรียนนั้นเสียเวลา ส่วนมากมักถามนักเรียนว่า เธอมีคำถามไหม และรอดูว่า ถ้านัก เรียนยกมือขึ้น ศาสตราจารย์ก็จะวาน ครูผู้ช่วย ใครมีปัญหาต้องเขียนและส่งให้ครูผู้ช่วย ครูผู้ ช่วยเขียนคำถามทั้งหมดลงบนกระดาษและส่งให้ศาสตราจารย์ ศาสตราจารย์รับมาและอ่าน อย่างรวดเร็ว และเขียนคำตอบลงบนกระดาษส่งให้ครูผู้ช่วย ครูผู้ช่วยส่งคำตอบให้นักเรียน ศาสตราจารย์พอใจมากเพราะไม่ต้องรอนักเรียนที่นึกคำถามไม่ออก และขณะเดียวกันนัก เรียนก็แยกคำตอบของศาสตราจารย์ได้ ศาสตราจารย์คิดคำตอบต่อไปได้ ครูผู้ช่วยนั้นทำหน้า ที่คล้ายกับ FEP ให้ศาสตราจารย์ ศาสตราจารย์เหมือนกับเป็น Host Computer ซึ่งทำงานได้ทันเวลา

ในตัวอย่างนี้ การติดต่อระหว่างนักเรียนและครูผู้ช่วยเกิดขึ้นที่ความเร็วช้าและการติดต่อระหว่างศาสตราจารย์และครูผู้ช่วยใช้เวลาเร็วกล่าว FEP ทำขึ้นด้วยส่วนประกอบทางโลจิก

CHANNEL INTERFACE ต่ออยู่กับ FEP ไปยัง อินพุท/เอาท์พุทความเร็วสูงของ Host Computer

PROCESSING UNIT บรรจุอยู่ใน FEP ทำหน้าที่เป็นโปรแกรมพิเศษสำหรับการติดต่อสื่อสาร โดยปกติโปรแกรมควบคุมเน็ทเวิร์คจะถูกโหลดอยู่ใน Processing Unit บนแผ่นดิสก์ หรือเทป

LINE INTERFACE (Front end ports) มีความสัมพันธ์ เป็นอินเตอร์เฟสความเร็วต่ำปกติต่อกับเทอร์มินอล ถ้า เทอร์มินอลอยู่ไกลจากการควบคุมจะใช้ Modem แทรกระหว่าง Line Interface กับ FEP

เนื่องจากจำนวนของพอร์ท คอมพิวเตอร์ มักจะถูกกำหนดโดยสถาปัตยกรรม Host Computer FEP สามารถเพิ่มจำนวนเทอร์มินอล ที่ติดต่อกับ Host Computer FEPยอมให้อุปกรณ์ที่ทำงานความเร็วต่ำ แบ่งไปยังพอร์ทที่ทำงานเร็ว เนื่องจากช่องอินเตอร์เฟสทำงานที่ความเร็วสูง FEP อาจจะต้องมีการรอข้อมูลเพื่อส่งไปยัง Terminalจนกระทั่งข้อมูลส่งมาด้วยความเร็วต่ำการรอรับข้อมูลเรียกว่า บัฟเฟอร์ (BUFFER)

FEP บางแบบใช้วิธีพิเศษ เรียกว่า โปรโตคอล ขบวนการซึ่งติดต่อระหว่าง Host และเทอร์มินอลถูกควบคุมโดย FEP ซึ่งเรียกว่า พูลลิ่ง (Polling) และ ซีเล็กติ้ง (Selecting)

การพูลลิ่ง เกิดขึ้นเมื่อ FEP มีการติดต่อกับเทอร์มินอล ว่ามีข้อมูลส่งไปยัง Host Computer หรือไม่ ที่จุดนี้ เทอร์มินอลอาจส่ง หรือไม่ส่งข้อมูลก็ได้

การซีเล็กติ้ง เกิดขึ้นเมื่อ FEP ติดต่อกับเทอร์มินอลว่าสามารถรับข้อมูลได้หรือยัง เทอร์มินอลที่ดี อาจตอบรับว่าได้ หรือว่ายังไม่ว่าง

เทอร์มินอลบางชนิดสามารถรับรู้ข้อมูลเหล่านั้นได้ เทอร์มินอลที่เลวไม่สามารถควบคุมโดยใช้การ พูลลิ่งและซีเล็กติ้ง Host Computer ก็สามารถทำการพูลลิ่งและซีเล็กติ้งได้ ง่ายและมีประสิทธิภาพกว่า

เปรียบเทียบกับตัวอย่างข้างต้นแล้ว การพูลลิ่ง ถูกระทำโดยครูผู้ช่วยเมื่อเขาเดินเก็บกระดาษคำถามจากนักเรียนรอบๆ ห้อง การซีเล็กติ้ง อาจเปรียบได้กับการแจกคำตอบของครูผู้ช่วย เทคนิคการซีเล็กติ้ง จะต้องใช้ครูผู้ช่วยถามนักเรียนว่า พร้อมที่จะรับคำตอบหรือยัง แล้วส่งคำตอบให้นักเรียน

FEP หลายๆ ตัวสามารถทำหน้าที่พิเศษแปรเปลี่ยนไปตามการผลิตและโมเดลหน้าที่ที่รู้จักกันโดยทั่วไปคือ การตรวจจับข้อผิดพลาดและแก้ไขข้อผิดพลาด (Error deteetion and correction) FEP ที่มีคุณลักษณะดีสามารถ ตรวจจับและแก้ไขการส่งบิท (bit) ที่เกิดขึ้นได้

หน้าที่อื่นๆ ที่ FEP บางตัวทำได้คือ การแปลงดาต้า (data conversino)

สามารถโค๊ดซึ่ง Host Computer และเทอร์มินอลที่ใช้โค๊ดต่างกันเข้าด้วยกัน ตัวอย่าง เทอร์มินอลที่ใช้ ASCII โค๊ดสามารถต่อกับ Host Computer ที่ใช้โค๊ด EBCDIC โดยใช้ FEP ที่มีคุณสมบัติการแปลงโค๊ด (Code Conversion) บางครั้งอาจมีการแปลงโปรโตคอล ด้วย การต่อเทอร์มินอลแบบบอชิงโครนัส ซึ่งไม่มีสัญญาณโปรโตคอลเข้ากับ Host Computer ที่ใช้ ชิงโครนัสโปรโตคอล ต้องใช้ การแปลงโปรโตคอล (Protocul Conversion) ส่วนสุดท้าย FEP บางตัวสามารถเปลี่ยนขนาน/ อนุกรมได้

การเปลี่ยนแปลงฟังก์ชั่นทั้งหมดนี้ทำได้โดยใช้อุปกรณ์แยกเช่น โปรโตคอลคอนเวอร์เตอร์ พาราแรล/ซีรีส์ คอนเวอร์เตอร์ ซึ่งมีราคาถูกกว่าที่จะสร้างฟังก์ชั่นเหล่านี้ไว้ในตัว FEP โดยตรง FEP สามารถสร้างให้แตกต่างกันแต่ละพอร์ท แม้ว่าเป็นHost Computer ชนิดเดียวกัน แต่ใช้ เทอร์มินอลต่างกัน

FEP สามารถทำ Histrical logging หรือ Statistical logging ได้

การ Logging บอกให้ทราบจำนวนของผู้ใช้ FEP รวมทั้งข้อมูลต่างๆ เกี่ยวกับการใช้งานของ USERS

FEP มีส่วนประกอบต่างๆ ที่แตกต่างกันหลายรูปแบบ จำนวนสายอินเตอร์เฟสของ FEP สามารถเปลี่ยนแปลงได้ตามโมเดลของ Host Computer

การใช้งานและการทำงานของชิพซัพพอร์ท 8255

8255 เป็นพอร์ทแบบขนานของอินเทลในชื่อว่า programmable peri-pheral interface (PP) ซึ่งได้รับการออกแบบมาเพื่อใช้กับซีพียูเบอร์ 8080 และ 8085 แต่กลับเป็นที่นิยมชมชื่นของผู้ออกแบบระบบไมโครคอมพิวเตอร์บ้านเราซึ่งส่วนมากใช้ซีพียูเบอร์ Z80 ทั้งนี้อาจจะเป็นเพราะ Z80-PIO ซึ่งเป็นพอร์ทแบบขนานเช่นกันราคาสูงเกินสมควร

ใน 8255 มีพอร์ทใช้งานอยู่ 3 พอร์ทให้ชื่อว่าพอร์ท A พอร์ท B และ พอร์ท C กับพอร์ทควบคุม (Control Port) อีก 1 พอร์ท พอร์ทควบคุม คือพอร์ทที่ทำหน้าที่ติดต่อกับซีพียู เพื่อควบคุมการทำงานของพอร์ทใช้งานทั้งสาม รายละเอียดจะกล่าวต่อไป หากต่อขาตามที่กล่าวมาโดยเฉพาะต่อ A_0 กับ A_0 และ A_1 กับ A_1 และถ้า PSn ที่ เลือกมาเป็น PS0 ตามบทความในตอนที่แล้ว คุณจะได้ชื่อพอร์ทดังนี้

00 H = พอร์ท A

01 H = พอร์ท B

02 H = พอร์ท C

03 H = พอร์ท ควบคุม

ก่อนที่จะใช้งาน

ขาของพอร์ทแต่ละพอร์ทจะมีชื่อตามชื่อพอร์ท คือ PA_0 ถึง PA_0 PB_0 ถึง PB_1 และ PC_0ถึง PC_1 แต่ก่อนที่จะใช้งานมันได้ เราจะต้องโปรแกรมมันก่อน โดยเขาท์พุทค่าที่เรียกว่าcontrol byte แก่ พอร์ทควบคุม ค่าดังกล่าวนี้จะถือว่าเป็นคำสั่งก็ได้และเป็นคำสั่งขนาด 1ไบท์ คือ 8 บิท

แต่ขอกล่าวเป็นส่วนรวมเสียก่อนว่า ค่าที่จะให้เมื่อต้องการเลือกเป็น I/P หรือ O/Pนั้นถ้าเป็น "1" เท่ากับกำหนดให้เป็น I/P ถ้าเป็น "0" เท่ากับกำหนดให้เป็น O/P และสำหรับพอร์ท C แบ่งเป็น 2 พอร์ทย่อย คือ พอร์ท CLO ขา PC_0 - PC_3 กับพอร์ท CHI ขา PC_4-PC_7 ซึ่งเราสามารถโปรแกรมให้ในแต่ละชีกเป็น I/P หรือ O/P ก็ได้

อีกประการหนึ่งการทำงานของ 8255 สามารถกำหนดโหมด (mode) การทำงานได้เป็น3 โหมด ซึ่งขอเว้นรายละเอียดไว้ก่อน โหมดทั้ง 3 ได้แก่

Mode 0 เป็น I/P หรือ O/P พอร์ท จัดได้ทั้ง 3 พอร์ท

Mode 1 เป็น I/P หรือ O/P พอร์ท มี handshaking จัดได้เฉพาะพอร์ท A และ B

Mode 2 เป็น bidlrectional มี handshaking จัดได้เฉพาะพอร์ท A

คอนโทรลไบท์หรือที่ว่าอาจถือเป็นคำสั่งก็ได้นั้น ทางอินเทลได้กำหนดความหมายของแต่ละบิทโดยให้ค่าแต่ละบิทเป็น 1 หรือ 0 ไว้ดังนี้

บิท 0 ให้พอร์ท CL_0 เป็น I/P หรือ O/P

บิท 1 ให้พอร์ท B เป็น I/P หรือ O/P

บิท 2 ให้พอร์ท B ทำงานในโหมด 0 หรือ 1 (0 = โหมด 0 และ 1 = โหมด 1)

บิท 3 ให้พอร์ท CH_1 เป็น I/P หรือ O/P

บิท 4 ให้พอร์ท A เป็น I/P หรือ O/P

บิท 5 และบิท 6 ให้พอร์ท A ทำงานในโหมดดังนี้

00 = โหมด 0

01 = โหมด 1

10 = โหมด 2

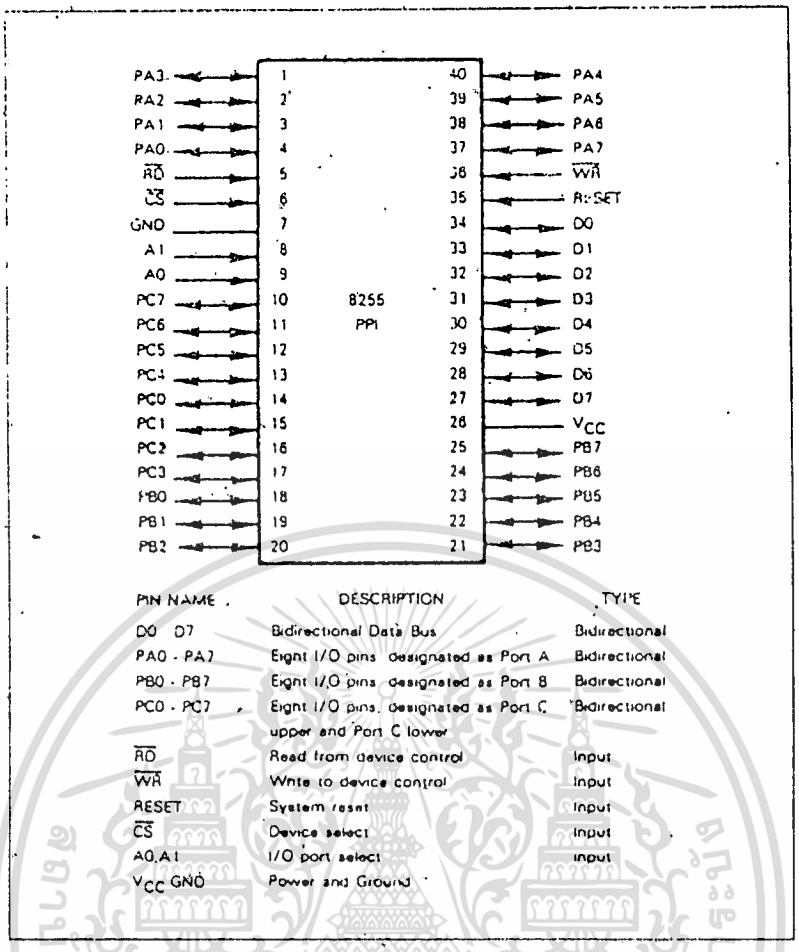
11 = โหมด 2

บิท 7 มีค่าเป็น 1 ประจำ เพื่อแสดงว่าไบท์นี้เป็นคอนโทรลไบท์

สมมติว่าต้องการให้ทุกพอร์ทเป็น O/P และให้พอร์ท A กับพอร์ท B ทำงานในโหมด 0 จะได้คอนโทรลไบท์เป็น 10000000 ซึ่งอ่านเป็นเลขฐานสิบหกได้เป็น 80H จะได้เป็นคำสั่งของ Z80 ว่า

LD A, 80H : ค่าคอนโทรลไบท์ OUT (03H)A: ให้แก่พอร์ควบคุมหรือถ้าต้องการให้พอร์ท A เป็น O/P พอร์ท B และ C เป็น I/P ให้พอร์ท A และ B ทำงานในโหมด 0 จะได้คอนโทรลไบท์เป็น 10001011 คือ 8BH

การทำงานในโหมด 0 หรือที่ เรียกว่า simple I/P port ผู้ที่ใช้ 8255 กับ Z80 มักจะใช้มากกว่าโหมดอื่น คือ ประมาณ 80% ทั้งนี้เพราะในโหมด 1 และ 2 จะต้องมีกรรมวิธีในการอินเทอร์รัพท์ซึ่งจะต้องจัดวงจรให้ ในการออกแบบระบบจึงต้องนำความกะทัดรัดและราคาไปเทียบกับ Z80 PIO ซึ่งสามารถดำเนินกรรมวิธีอินเทอร์รัพท์ได้โดยตรง ไม่ต้องพึ่งอุปกรณ์ด้านฮาร์แวร์



PIN NAME	DESCRIPTION	TYPE
D0 - D7	Bidirectional Data Bus	Bidirectional
PA0 - PA7	Eight I/O pins designated as Port A	Bidirectional
PB0 - PB7	Eight I/O pins designated as Port B	Bidirectional
PC0 - PC7	Eight I/O pins designated as Port C upper and Port C lower	Bidirectional
$\overline{RD}$	Read from device control	Input
$\overline{WR}$	Write to device control	Input
RESET	System reset	Input
$\overline{CS}$	Device select	Input
A0,A1	I/O port select	Input
V_{CC} GND	Power and Ground	

รูปที่ 1 แสดงการจัดขาของ 8255

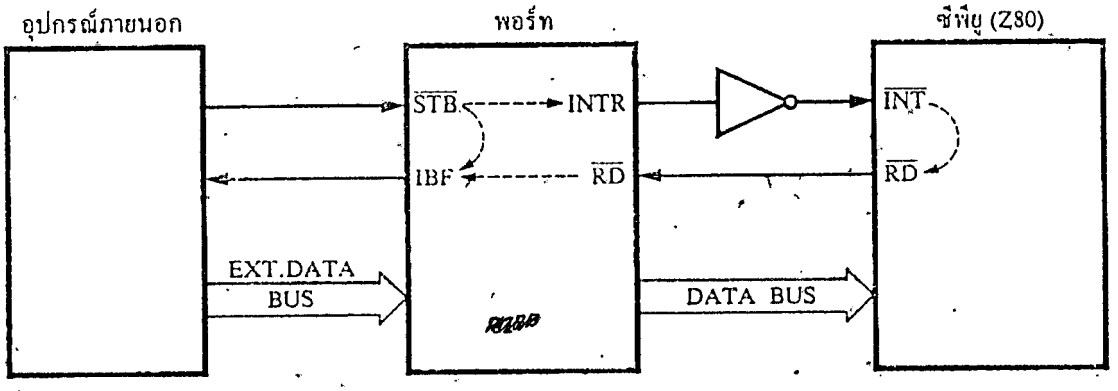
กรรมวิธี Handshaking

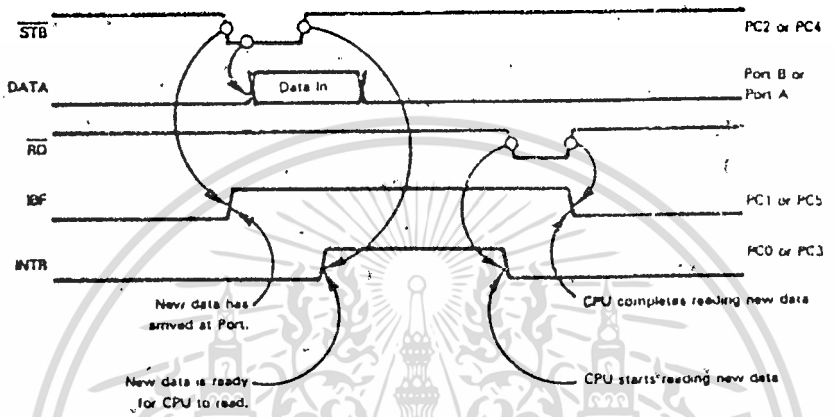
แต่ถ้าเราใช้ 8255 เพียงตัวเดียวในระบบของ Z80 ก็ไม่จำเป็นที่จะต้องจัดวงจรร้องขออินเทอร์รัพท์ให้ และในเรื่องของ handshaking ทั้งในแบบขนานและอันดับ ทั้งของอินเทลและของไซล็อก (และของบริษัทอื่นด้วย) มีลักษณะที่เหมือนกัน หากเข้าใจตัวใดตัวหนึ่งจะทำความเข้าใจตัวอื่นได้ไม่ยาก แต่ของ 8255 เป็นแบบที่ง่ายต่อการทำความเข้าใจ

กรรมวิธีของการ handshaking ซึ่งจะถือว่าเป็นโปรโตคอล (protocal) ก็ได้ เป็นกรรมวิธีระหว่างพอร์ทกับอุปกรณ์ภายนอกฝ่ายหนึ่ง และระหว่างพอร์ทกับซีพียูอีกฝ่ายหนึ่ง คือ พอร์จะเป็นตัวกลางของการติดต่อระหว่างซีพียูกับอุปกรณ์ภายนอก แบ่งเป็น 2 ลักษณะ คือเมื่อเป็น I/P กับ O/P พอร์ท

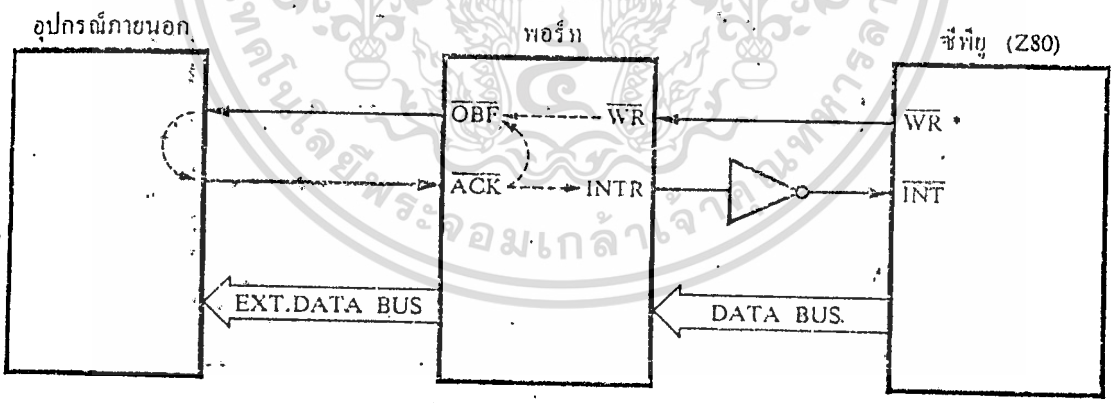
ตามรูปที่ 2 เป็นการแสดงกรรมวิธี handshaking เมื่อเป็นอินพุทพอร์ท (ชื่อขาของพอร์ทซึ่งไม่มีในรูปที่ 1) เริ่มที่อุปกรณ์ภายนอกเมื่อ มีข้อมูล จะส่งให้ ซีพียู มันจะทำให้ขา STB (strobe) ของพอร์ทได้รับลอจิก "0" พร้อมกับ ให้ข้อมูลทาง exlemal data bus จะทำให้พอร์ทรับข้อมูลนี้ไว้ แล้วดำเนินกรรมวิธี 2 ประการ คือ ประการแรก มันจะทำให้ IBF (input buffer full) เป็น "1" ค้างอยู่ เพื่อห้ามอุปกรณ์ภายนอกส่งข้อมูลซ้อนและมันจะร้องขออินเทอร์รัพท์โดยทำให้ขา INTR (interrupt) เป็น "1" ซีพียู (คือเราจะต้องเขียนโปรแกรมไว้ในอินเทอร์รัพเซอร์วิสรุทีน) จะอ่านค่าด้วยการอินพุทจากพอร์ททำให้ RD (read) เป็น "0" เป็นการแจ้งแก่พอร์ ทว่าได้รับข้อมูลแล้ว พอร์ทจะถอนอินเทอร์รัพและทำให้IBF เป็น "0" เป็นการแจ้งแก่อุปกรณ์ภายนอกว่าพร้อมที่จะรับข้อมูลใหม่ หากอุปกรณ์ภายนอกมีข้อมูลที่จะส่งก็จะทำให้ STB เป็น "0" ใหม่

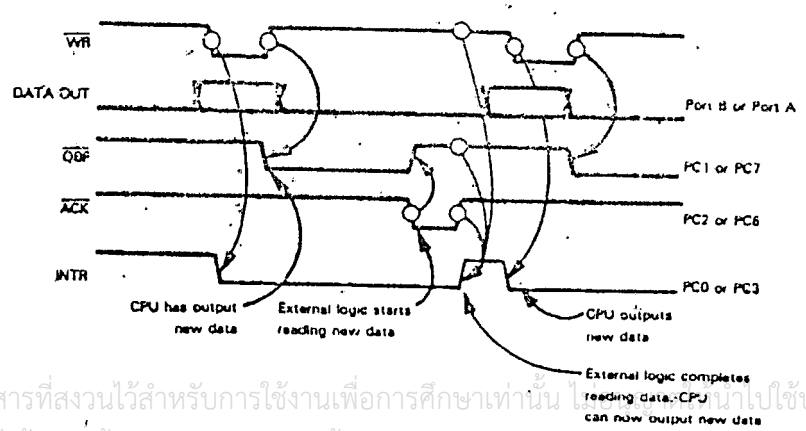
สำหรับกรรมวิธี handshaking เมื่อเป็นเอาท์พุทพอร์ท แสดงไว้ในรูปที่ 3 เริ่มด้วยเมื่อซีพียูต้องการส่งข้อมูลให้แก่อุปกรณ์ภายนอกก็จะเอาท์พุทให้แก่พอร์ท ซึ่งจะทำให้ ขา WR (write) เป็น "0" พอร์ทจะรับข้อมูลไว้และทำให้ OBF (output buffer full) เป็น 0 แจ้งแก่อุปกรณ์ภายนอกว่ามีข้อมูล เมื่ออุปกรณ์ภายนอกรับข้อมูลแล้วจะทำให้ ACK (acknowledge) เป็น"0" ที่พอร์ทจะทำให้ OBF เป็น "1" และทำให้ INTR เป็น "1" ไปเป็นการร้องขออินเทอร์รัพท์ ถ้าเป็ฯการส่งข้อมูลต่อเนื่อง ซีพียูก็จะส่งข้อมูลต่อไป พอร์ทจะถอนอินเทอร์รัพท์และดำเนินกรรมวิธีซ้ำเดิม แต่ถ้าซีพียูยังไม่มีข้อมูลก็สามารถสั่งให้พอร์ทถอนอินเทอร์รัพท์ได้ดังที่จะกล่าวต่อไป





รูปที่ 2 แสดงกรรมวิธี handshaking เมื่อเป็นอินพุทพอร์ทที่กำหนดให้เป็น Z80 เพื่อแสดงการต่อขาอินเทอร์รัพท์





รูปที่ 3 แสดงกรรมวิธี handshaking เมื่อเป็นเอาท์พุทพอร์ท

ชื่อขา	I/P	O/P	BIDIRECTION
PC_0	B INTR	B INTR	-
PC_1	B IBF	B $\overline{OBF}$	-
PC_2	B $\overline{STB}$	B $\overline{ACK}$	-
PC_3	A INTR	A INTR	A INTR
PC_4	A $\overline{STB}$	-	A $\overline{STB}$
PC_5	A IBF	-	A IBF
PC_6	-	A $\overline{ACK}$	A $\overline{ACK}$
PC_7	-	A $\overline{OBF}$	A $\overline{OBF}$

รูปที่ 4 แสดงการจัดขาของพอร์ท C เป็นขาสัญญาณในกรรมวิธี handshaking อักษร A และ B นำหน้าหมายถึงส่วนของพอร์ทนั้น เช่น BINIR หมายถึงเป็นขา INIR ของพอร์ท B

และจากรูปที่ 4 จะเห็นว่าทั้งพอร์ท A และ B เป็นอิสระต่อกัน ไม่เหมือน Z80-PIO ที่เมื่อให้พอร์ท A ทำงานในแบบ bidirectional แล้ว พอร์ท B จะเป็นได้เพียงอินพุทหรือเอาท์พุทพอร์ทธรรมดา แต่ข้อเสียของ 8255 คือเป็นการวางตัวในแบบตามตัว การเปลี่ยนแปลงต้องทำทางด้านฮาร์แวร์

เมื่อทำงานในโหมด 1 หรือ 2 จะเห็นว่าขาของพอร์ท C เมื่อเริ่มต้นจะมีทั้งลอจิก "1"หรือ "0" ซึ่งเราจะต้องเตรียมค่าให้ขาที่แอคทีฟ "1" เช่น INIR จะต้องให้เป็น "0" ส่วนขาที่แอคทีฟ "0" เช่น SIB จะต้องให้เป็น "1" และทำได้โดยให้พอร์ท C เป็นเอาท์พุทพอร์ท และเอาท์พุทค่าที่เตรียมไว้ให้แต่ในช่วงตอนระหว่างการทำงานหากต้องการเปลี่ยนแปลงค่าที่พอร์ท C เช่น ให้ยกเลิกการขออินเทอร์รัพท์ เพื่อไม่ให้กระทบกระเทือนการทำงานของขาอื่นๆ ควรสั่งทีละขาโดยใช้คำสั่ง 1 ไบท์ เช่นกัน ดังนี้

บิท 0 ให้ขาที่จะแอดเดรสถึงเป็น "0" หรือ "1"

บิท 1 ถึงบิท 3 เป็นการอ้างแอดเดรสถึงขาของพอร์ท C แต่ละขาดังนี้

000 = PC_0

001 = PC_1

010 = PC_2

011 = PC_3

100 = PC_4

101 = PC_5

110 = PC_6

111 = PC_7

บิท 4 ถึงบิท 6 ไม่ใช้ จะให้ค่าใดก็ได้ ส่วนมากให้เป็น 0

บิท 7 เป็น 0 ประจำแสดงว่าเป็นแอดเดรสไบท์

สมมติว่าต้องการให้พอร์ท b ถอนอินเทอร์รัพท์ คือทำให้ขา PC_0 ซึ่งเป็นขา B INTRเป็น "0" จะได้แอดเดรสไบท์เป็น 00000000 คือ 00H

8225 กับ Z80

การใช้งาน 8225 กับ Z80 ในโหมดที่มี handshakint หากใช้ตามที่ไซล็อกเตรียมไว้คือให้ Z80 ทำงานในอินเทอร์รัพท์โหมด 0 (IMO) ไม่เป็นที่นิยม เพราะจะต้องจัดวงจรจัดลำดับความสำคัญในการร้องขออินเทอร์รัพท์วงจรบอกค่ารีสตาร์ท (RST) และวงจรตอบรับอินเทอร์รัพท์ซึ่งเป็นเรื่องยุ่งยาก

หนทางแก้ ถ้าหากใช้ 8255 หลายตัว และไม่ได้ใช้พอร์ทของ Z80 ร่วมด้วย (พอร์ทของ Z80 ทำงานใน Z80 อินเทอร์รัพท์โหมด 2 เท่านั้น และอย่าสับสน ระหว่างโหมดการทำงานของพอร์ทกับโหมดของการอินเทอร์รัพท์ของซีพียู) จะทำได้โดย ให้ Z80 ทำงานใน อินเทอร์รัพท์โหมด 1 ซึ่งพอร์ทไม่ต้องให้ค่า RST หรืออินเทอร์รัพท์ เวคเตอร์ไบท์ต่ำ

และนำขา INTR ของแต่ละพอร์ทไปเข้าขา I/P ของ NOR เกท ซึ่งมีหลายอินพุท เมื่อมีอินเทอร์รัพท์เข้ามาให้เป็นหน้าที่ของซีพียูตรวจว่าพอร์ทใดร้องขออินเทอร์รัพท์ แต่ใน 8255 ไม่มีสเตตัสรีจิสเตอร์ (status register) จึงทำได้โดยอินพุทค่าที่พอร์ท C ของแต่ละชิพเข้ามาตรวจบิท 0 และบิท 3 ว่าพอร์ทใดร้องขออินเทอร์รัพท์

วิธีการนี้มีข้อเสียที่ช้าเพราะทำงานในแบบพูลลิ่ง (interrupt pooling method) ที่ต้องอ่านค่าพอร์ท C มาตรวจที่ละพอร์ท นอกจากนี้การจัดลำดับความสำคัญต้องทำในโปรแกรมทำให้ช้าขึ้นอีก แต่ก็มีข้อดีอยู่ที่จัดระบบได้ง่ายและคิดว่ามีงานเป็นจำนวนมากที่เดียวที่ข้อเสียดังกล่าวไม่เป็นผลเสีย ข้อสำคัญคือเมื่ออินพุทพอร์ท C แล้วต้องเอาท์พุทค่าเดิมนี้ออกทางพอร์ท C ทันที ไม่เช่นนั้นค่าที่มันแลทช์ไว้จะสูญเสียไป เรื่องนี้ความจริงเป็นกับ 8255 บางรุ่นเท่านั้น

การใช้ 8255 ร่วมกับพอร์ทหรืออุปกรณ์สนับสนุนของ Z80 เช่น CTC ซึ่งพวกนี้จะทำงานใน Z80 อินเทอร์รัพท์โหมด 2 (IM2) คือ อุปกรณ์พวกนี้จะให้อินเทอร์ รัพท์เวคเตอร์ไบท์ต่ำแก่ซีพียู เมื่อซีพียูตอบรับอินเทอร์รัพท์ แต่ 8255 ทำไม่ได้

การทำงานชิพซัพพอร์ท 6845

กานสร้างวงจรควบคุมจอภาพเราอาจสร้างได้จากไอซีแยกส่วน คือจากไอซีที่เป็นวงจรหารและวงจรชิฟท์รีจิสเตอร์กับพวกเกทต่างๆ หรือจากไอซีที่เป็น CRTC สำเร็จรูป มีอยู่หลายเบอร์ด้วยกัน เบอร์ที่นำมาเสนอนี้เป็นเบอร์ 6845D ซึ่งค่อนข้างแพร่หลายในบ้านเราหรือจะใช้เบอร์ 6545 ก็ได้ จากการทดลองของ ไอซี 2 เบอร์นี้สามารถใช้แทนกัน ได้โดยตรง

สร้างภาพบนจอภาพ

การสร้างภาพบนจอภาพทางดิจิตอลทำได้ 2 แบบคือ แบบอักขระ (character mode) กับแบบกราฟฟิค (graphics mode) ที่จะนำมากล่าวหรือที่ CRTC ทั้ง 2 เบอร์ทำได้อีกการแสดงภาพในแบบอักขระการแสดงภาพในแบบนี้ได้กล่าวไว้ค่อนข้างละเอียดใน วารสารไมโครคอมพิวเตอร์ ฉบับที่ 2 ถึง ฉบับที่ 5 จึงขอนำมากล่าวโดยย่อเพื่อเป็นพื้นฐานในการใช้งาน CRTC

การแสดงภาพในแบบอักขระจะแบ่งจอภาพเป็นคอลัมน์ในทางแนวนอนและเป็นบรรทัดอักขระในทางตั้ง เช่นเครื่องแอปเปิ้ลแบ่งจอภาพเป็น 40 คอลัมน์ 24 บรรทัดอักขระจะแสดงภาพอักขระได้ 40x24=960 ตัว อักขระแต่ละตัวเป็นแมทริกซ์แบบ 8x8 จุดภาพ ซึ่งจะต้องกำหนดให้เป็นภาพที่ต้องการเช่นเป็นภาพอักษร ตัวเลข เครื่องหมาย หรือภาพที่ใดไว้ในคาแรคเตอร์เยนฯ(character generator)

คาแรคเตอร์เยนฯ เป็นหน่วยความจำประเภท ROM ปัจจุบันนิยมใช้ EPROM หน่วยความจำนี้ 1 ไบท์ คือ 8 บิท สามารถทำให้เกิดจุดภาพได้ 8 จุดในทางแนวนอน ดังนั้นจึงต้องใช้หน่วยความจำนี้ 8 ไบท์ เพื่อทำให้เกิดภาพในแบบแมทริกซ์ 8x8 ได้ 1 ภาพ คือ 1 อักขระหากใช้ EPROM เบอร์ 2716 คือ 2 กิโลไบท์ จะได้ภาพอักขระเท่ากับ 2048/8= 256 ตัว

หน่วยความจำขนาด 2 กิโลไบท์มีขาแอดเดรส 11 ขา ในการสร้างแพทเทอร์นของภาพอักขระในคาแรคเตอร์เยนฯได้กำหนดไว้ว่าขาแอดเดรสตั้งแต่ A_3 ถึง A_{10} รวม 8 ขา เป็นการกำหนดว่าเราต้องการภาพอักขระใดใน 256 ภาพดังกล่าวมา ($2^8 = 256$) และเรียกว่ารหัสส่วนขา A_0 ถึง A_2 เป็นการกำหนดว่าเราจะเลือกไบท์ใดใน 8 ไบท์ของอักขระนั้นๆ ซึ่งจะสัมพันธ์อยู่กับสแกนไลน์ที่จะกล่าวต่อไป ขอเรียนย้ำตรงนี้ว่าเมื่อเรา กำหนดรหัส และกำหนดไบท์ที่ในรหัสนั้นเราจะได้ภาพ 1 ไบท์ คือ 8 จุดภาพบนจอภาพ ส่วน การที่จะกำหนดรหัสได้อย่างไร ไบท์ที่ได้อย่างไร เป็นเรื่องที่จะกล่าวถึงต่อไป

สร้างสถานีส่งโทรทัศน์

หากมีจอภาพแบบที่เรียกว่ามอนิเตอร์ สามารถสร้างสถานีส่งโทรทัศน์ไปยังจอภาพได้โดยง่าย เมื่อภาพที่จะนำออกแสดงเป็นภาพอักขระโดยสร้างสัญญาณ VSYNC (vertical synchronous) ทุกๆ 20 มิลลิวินาที นานประมาณ 1 มิลลิวินาที และ HSYNC (horizontal synchronous) ทุกๆ 64 ไมโครวินาทีนานประมาณ 5 ไมโคร วินาที กับให้สัญญาณจุดภาพ (ขาวหรือดำ) คุณจะได้ภาพออกจอภาพในแบบ 312 สแกนไลน์

ช่วงเวลา 64 ไมโครวินาทีคือช่วงเวลา 1 สแกนไลน์ คือช่วงเวลาที่จอภาพสร้างภาพได้ 1 เส้นในทางระดับช่วงเวลา 20 มิลลิวินาที คือช่วงเวลาของ 1 ฟิลด์ คือช่วงเวลาที่จอภาพสร้างภาพได้เต็มจอภาพ คือ 312 เส้นสแกนไลน์ค่านี้ได้มาจาก 20 มิลลิวินาที /64 ไมโครวินาที = 312.5 เส้น แต่ค่า 20 คลาดเคลื่อนได้เล็กน้อยจึงบัดเศษได้การส่งโทรทัศน์จริง ๆ จะส่งเป็นฟิลด์ต่อ 1 เฟรม จึงเรียกว่าระบบเส้น

แต่ภาพที่ได้จะตกจอภาพและเป็นเพราะจะเห็นการหันกวาดของอิเลคตรอนในช่วงกวาดกลับ จึงต้องให้สัญญาณภาพเป็นจุดดำก่อนหลังสัญญาณซิงค์ทั้งสองเรียกว่า แบล็งค์ใน 1 สแกนไลน์คือ 64 ไมโครวินาที จึงมีเวลาเหลือสำหรับแสดงภาพประมาณ 40 ไมโครวินาที และ 1 ฟิลด์คือ20 มิลลิวินาที จะมีเวลาแสดงภาพประมาณ 13 มิลลิวินาทีคือประมาณ 200 สแกนไลท์ ค่าเวลาต่างๆ เหล่านี้คลาดเคลื่อนได้เช่นกัน

สัญญาณที่ทำให้เกิดจุดภาพและซิงค์มีขนาดประมาณ

จุดขาว = 2 โวลท์

จุดดำ = 0.5 โวลท์

ซิงค์ = 0 โวลท์

ที่กล่าวว่าโดยประมาณเพราะในวงจรจอภาพมีวงจรภาคขยายสัญญาณ แต่ไม่ควรต่ำกว่าครึ่งหนึ่งและไม่ควรมากกว่านี้

ให้ทำความเข้าใจเสียในตอนนี้ก่อนว่าค่าแรงดันที่นำมากล่าวเป็นแรงดันที่เอาท์พุทของภาคผสมสัญญาณ (video composite) ซึ่งเป็นลีเนียร์ แต่ภาคอินพุทของวงจรนี้กลับเป็นดิจิตอลขอให้ดูรูปที่ 1 ประกอบ ดังนั้นเพื่อได้เกิดสัญญาณเอาท์พุทดังกล่าว อาจกำหนดให้จุดขาวเป็นลอจิก "1" จุดดำเป็นลอจิก "0" หรือกลับก็ได้ ส่วนสัญญาณซิงค์ก็ เช่นกัน จะกำหนดเป็น "1" หรือ "0" ก็ได้ ตามวงจรกำหนดให้จุดขาวเป็นสอจิก "1" จุดดำ เป็นลอจิก "0"สัญญาณซิงค์เป็นลอจิก "1" (เพื่อให้สอดคล้องกับ 6845)

ใน 1 สแกนไลน์เราต้องการให้เกิดจุดภาพกี่จุดก็ได้ในช่วง 200 ถึง 1200 จุดภาพขึ้นอยู่กับจำนวนคอลัมน์ที่เราต้องการ เช่นต้องการให้แสดงภาพแบบ 60 คอลัมน์ โดยให้อักขระหนึ่งๆ อยู่ในแมทริกซ์ 8 x n จุดภาพ ดังนั้นในช่วงเวลา 40 ไมโครวินาทีของ 1

สแกนไลท์ที่แสดงภาพได้ จะต้องเป็น 80 คอลัมน์และมีจุดภาพเป็น 80 x 8 = 640 จุดภาพ หากคิดทั้งสแกนไลท์ (64 ไมโครวินาที) จะเป็น 1024 จุดภาพ

การกำหนดจุดภาพกำหนดด้วยสัญญาณนาฬิกา DCLK (dot clock) ซึ่งจะชิฟท์ข้อมูลจากชิฟท์รีจิสเตอร์ (รูปที่ 1) ให้แก่วงจรภาคผสมสัญญาณ จากตัวอย่างที่ยกมาใน 1 ไมโครวินาทีเราต้องทำให้เกิดจุดภาถึง 16 จุดภาพจะได้ความถี่ของ DCLK ซึ่งก็คือความถี่ของคริสตอลเป็น16 MHz

ความถี่นี้ไม่จำเป็นต้องตรงตามนี้เช่นหากใช้คริสตอล 17.3 MHz จะได้ว่าใน 1 สแกนไลท์ (64 ไมโครวินาที) มี 1107.2 จุดภาพแต่จุดภาพจะเป็นเศษไม่ได้และต้องหารด้วยจำนวนจดต่อคอลัมน์คือ 8 ลงตัว ค่าที่ใกล้เคียงคือ 1104 จุดภาพคิดกลับจะได้ว่าใน 1 สแกนไลท์ยาว63.8 ไมโครวินาที เป็นค่าที่ยอมรับได้ จะได้ว่าใน 1 สแกนไลท์มี 138 คอมลัมน์แต่ใช้แสดงภาพเพียง 80 คอมลัมน์อีก 58 คอลัมน์ เป็นช่วงแบลงค์และ HSYNC

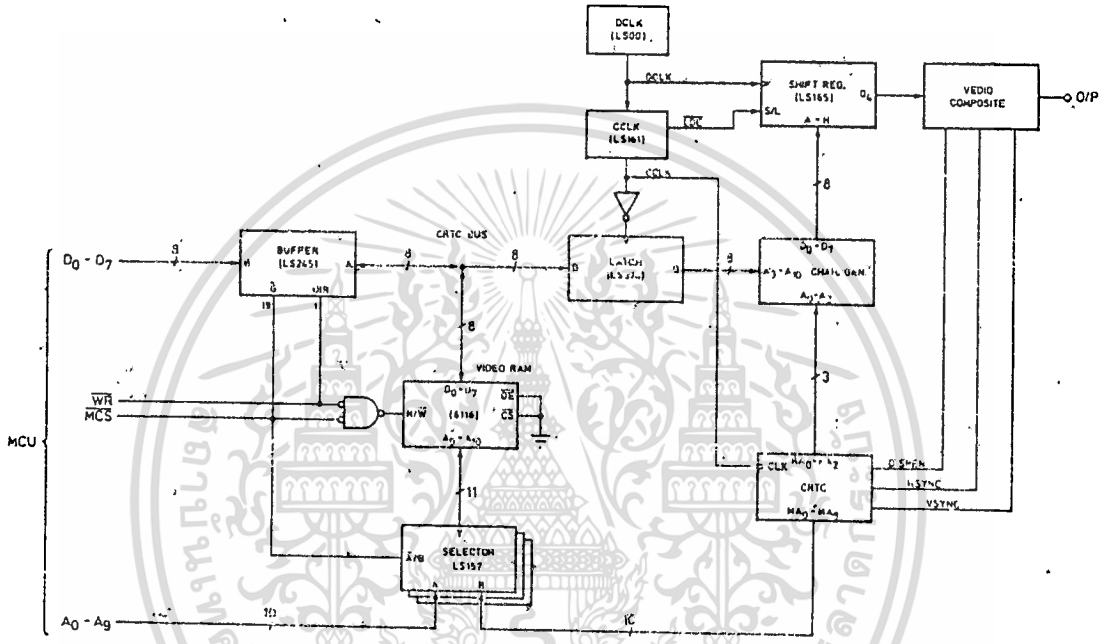
ความถี่ DCLK ซึ่งหารด้วยจำนวนจุดภาพต่อคอลัมน์คือ 8 เรียกว่าความถี่ CCLK (character clock) จะใช้เป็นหลักในการคำนวณต่อไป โดยเฉพาะเมื่อใช้ไอซีเป็น CRTCและเป็นสัญญาณในการโหลดข้อมูลแบบขนานจากคาแรคเตอร์เยนฯ ให้แก่ชิฟท์รีจิสเตอร์ กับส่งให้CRTC เพื่อสร้างสัญญาณอื่นๆ ต่อไป

เมื่อเรากำหนดช่วงเวลาของสแกนไลท์ได้แล้ว ขั้นต่อไปคือหาว่าใน 1 ฟิลด์จะมีกี่สแกนไลท์ วิธีง่ายๆ คือนำค่าเวลา 1 สแกนไลท์คูณด้วย 312+- ให้ได้ค่าเวลาใกล้เคียงกับ 20 มิลลิวินาทีที่สุด จะได้ว่าหากใช้คริสตอล 17.3 MHz ถ้าใช้ 313 สแกนไลท์จะได้เวลา 19.974 มิลลิวินาทีนับว่าใกล้เคียงที่สุด

ในการแสดงภาพหากต้องการให้แสดง 24 บรรทัดอักขระ ในแต่ละบรรทัดประกอบด้วยเส้นสแกนไลท์ 8 เส้น (จากแมทริกซ์ n x 8) เราต้องการเส้นสแกนไลท์เท่ากับ 24X8 =192 เส้นอยู่ในข้อจำกัดที่กล่าวมา และถึงแม้จะให้แสดงถึง 26 บรรทัดเท่ากับ 208 เส้นสแกนไลท์ก็ยังอยู่ในข้อจำกัด ในที่นี้จะให้แสดง 24 บรรทัด จึงเหลือคาบเวลาคิดเป็นเส้นสแกนไลท์ได้313 - 192 เท่ากับ 121 เส้น ซึ่งใช้เป็นช่วงแบลงค์และ VSYNC

ซีพียูกับจอภาพ

กำหนดให้โครงงานนี้แสดงภาพแบบ 80 คอลัมน์ 24 บรรทัดอักขระดังนั้นจำนวนอักขระทั้งหมดที่จะแสดงได้ต่อหนึ่งจอภาพคือ 80x24 = 1920 ตัว และจากที่ กล่าวแล้ว



รูปที่ 2 วงจรควบคุมจอภาพ

ว่าการที่จะให้อักขระใดไปปรากฏบนจอภาพเราเพียงแต่อ้างรหัสของมันและไบท์ที่ต่อ คาร์แรคเตอร์เยนฯ เรื่องไบท์ที่ซึ่งเราจะต้องอ้างจากไบท์ที่ 0 ถึง 7

ในการอ้างรหัสหากเราจัดหน่วยความจำหน่วยหนึ่งเป็น RAM ขนาด 1920 ไบท์ (ซึ่งคงต้องใช้ขนาด 2 กิโลไบท์) โดยให้ไบท์ 0 ของหน่วยความจำนี้เป็นอักขระแรกคือมุมบนซ้ายของจอภาพ ไบท์ที่ 1 จะเป็นอักขระที่ 2 คืออยู่ทางขวาของตัวแรก ไบท์ที่ 79 คืออักขระตัวสุดท้ายของบรรทัดแรก และไบท์ที่ 80 คือตัวแรกของบรรทัดที่ 2 และต่อๆ ไปจนถึงไบท์ที่ 1919 คืออักขระสุดท้ายอยู่ตรงมุมขวาล่างของจอภาพดังนั้นซีพียูจึงมองเห็นจอภาพเหมือนหน่วยความจำขนาด1920 ไบท์หน่วยหนึ่งเรียกหน่วยความจำนี้ว่า วิดีโอแรม (video RAM)

วงจรแสดงผลบนจอภาพ

เมื่อซีพียูต้องการแสดงภาพใดตรงที่ใดบนจอภาพก็จะส่งรหัสของอักขระนั้นมาเก็บในวิดีโอแรม ในไบท์ที่ต้องการอันสัมพันธ์กับจอภาพดังกล่าวมา การติดต่อของด้วยเท่านั้น และการทำใน แบบสุ่ม คือจะติดต่อกับไบท์ใดของวิดีโอแรมก็ได้

โดยปกติแล้ว CRTC เป็นผู้ติดต่อกับวิดีโอแรมอยู่ตลอดเวลา และทำในแบบซีเควนเชียลคือจะอ้างแอดเดรสถึงวิดีโอแรมตั้งแต่ไบท์ที่ 0 ถึงไบท์ที่ 1919 แล้วขึ้นไบท์ที่ 0 ใหม่ (รอบหนึ่งกินเวลาประมาณ 20 มิลลิวินาที)

เมื่อ CRTC อ้างแอดเดรสวิดีโอแรมจะได้ดาต้าคือรหัสตาที่กล่าวมา ซึ่งวงจรแล็ทช์จะแล็ทช์ค่าไว้ และนำไปเป็นแอดเดรส A_3 ถึง A_{10} ของคาแรคเตอร์เยนฯ ส่วนขา A_0 ถึง A_2นั้น CRTC เป็นผู้ให้ค่าแอดเดรสโดยตรง คาแรคเตอร์เยนฯ จะให้ดาต้า 1 ไบท์ แก่วงจรชิฟท์รีจิสเตอร์ ซึ่งจะถูก DCLK ชิฟท์ออกทีละบิทให้แก่วงจรภาคผสมสัญญาณส่งไปให้จอภาพอีกทอดหนึ่งเมื่อชิฟท์ครบ 8 บิทจะได้ภาค 8 จุดภาพบนจอภาพ

ตามที่กล่าวมาว่าใน 1 บรรทัดอักขระประกอบด้วยเส้นสแกนไลน์ 8 เส้น ซึ่งขอกำหนดเรียกเป็นไลน์ที่ 0 ถึง 7 การแสดงภาพในแต่ละบรรทัดอักขระสมมุติต้องการให้แสดงภาพอักขระ A ถึง Z จะเริ่มด้วยไลน์ที่ 0 คือส่วนหัวของอักขระคือไบท์ที่ 0 ของอักขระทุกตัวในคาแรคเตอร์เยนฯ ไปจนจบสแกนไลท์ แล้วขึ้นไลน์ที่ 1 คือไบท์ที่ 1 ของอักขระทุกตัว ไปจนถึงไลน์ที่ 7 คือส่วนฐานหรือคือไบท์ที่ 7 ของอักขระทุกตัว เป็นการแสดงภาพ 1 บรรทัดอักขระ

การทำงานจะเป็นในแบบ รหัสอักขระจะเปลี่ยนจาก A ถึง Z ทุกคอลัมน์คือทุก 1CCLK แต่ไลท์ที่เช่นไลน์ที่ 0 จะไม่เปลี่ยนตลอดสแกนไลน์นั้น เมื่อขึ้นสแกนไลน์ใหม่เช่นเป็นไลน์นั้นเมื่อขึ้นสแกนไลน์ใหม่เช่นเป็นไลท์ 1 รหัสอักขระจะเริ่มต้นเปลี่ยนจาก A ถึง Z ใหม่เป็นเช่นนี้จนครบ 8 สแกนไลท์ คือ 1 บรรทัดอักขระ ในบรรทัดอักขระต่อไป หากให้ แสดงภาพ ก. ถึง ฮ. ซ้ำอยู่ 8 เที่ยวดังกล่าวมา ทั้งหมดนี้เป็นหน้าที่ของ CRTC

ทำไมถึงเป็นแมทริกซ์ 8 x 8 หากใช้ 9 x 8 ปัญหาจะเกิดขึ้น ไม่สามารถจะหาหน่วยความจำขนาด 9 บิท มาทำเป็นคาแรคเตอร์เยนฯ ได้ ต้องข้ามไป 16 บิท (ใช้ 2716จำนวน 2 ตัวต่อขนานกัน) และใช้ชิฟท์รีจิสเตอร์ 2 ตัว จะได้ผลได้คุ้มค่า หากเปลี่ยนใหม่เป็น 8 x 12 ในรหัสอักขระหนึ่งๆ ต้องใช้หน่วยความจำถึง 16 ไบท์ แต่ใช้เพียง 12 ไบท์ ทั้งนี้เพราะหน่วยความจำต้องมีค่าเป็น 2^n เช่น 4,8,16 หรือ 32 และต่อๆไป การจัดในแบบ 8 x 12 ถึงจะเปลืองหน่วยความจำแต่ก็มีใช้กันอยู่บ้าง

CRTC 6845

ได้แสดงการจัดขาของ CRTC เบอร์ 6845 ไว้ในรูปที่ 3 ขออธิบายหน้าที่การทำงานเพิ่มดังนี้

ส่วนที่ติดต่อกับซีพียู

D_0 ถึง D_7 เป็นขาดาต้าต่อกับดาต้าบัสเบส

CS คือขา chip select เพื่อเลือกตัวมันเมื่อซีพียูต้องการจะติดต่อด้วย แอคตีฟเมื่อได้รับลอจิก "0"

RS คือขา register select ภายใน 6845 มีรีจิสเตอร์เพื่อใช้งานในหน้าที่ต่างๆ18 ตัว และแอดเดรสรีจิสเตอร์อีก 1 ตัว รวมเป็น 19 ตัว เมื่อให้ลอจิก "0" แต่ขานี้เป็นการเลือกติดต่อกับแอดเดรสรีจิสเตอร์ ค่าที่ให้คือหมายเลขรีจิสเตอร์ใช้งานที่เราต้องการติดต่อด้วยคือ0 ถึง 17 ($00 ถึง $11) เมื่อให้ลอจิก "1" แต่ขานี้เป็นการเลือกติดต่อกับรีจิสเตอร์อีก 18ตัว ตัวใดตัวหนึ่ง ซึ่งมีหมายเลขอยู่ในแอดเดรสรีจิสเตอร์ ปกติจะต่อขานี้กับ A_0 ของซีพียู

R/W คือขา read/write เมื่อให้ลอจิก "1" เป็นการอ่านจากรีจิสเตอร์ภายใน เมื่อให้ลอจิก "0" เป็นการเขียนกับรีจิสเตอร์ภายใน ซึ่งริจิสเตอร์แต่ละตัวให้อ่านและเขียนได้ไม่เหมือนกันดังจะกล่าวต่อไป

E คือขา enable synchronization signal. ชื่อประหลาด แต่ความจริงคือสัญญาณนาฬิกาของระบบไมโครคอมพิวเตอร์ เพื่อติดต่อกับ 6845 ในฐานะพอร์ท (ไม่เกี่ยวกับสัญญาณนาฬิกาไม่มีภาคแสดงภาพคือ DCLK หรือ CCLK แต่อย่างใด) ถ้าใช้ซีพียูเบอร์ 6502 หรือ6800 ให้ต่อกับสัญญาณนาฬิกา O_2 แต่ถ้าเป็น Z80 ได้จัดวงจรดังที่จะแสดงในรูปที่ 8 แต่นักศึกษาที่ทำโครงงานโดยใช้ Z80 กับ 6845 บอกว่าสามารถใช้ 0 ซึ่งเป็นสัญญาณนาฬิกาของระบบZ80 ติดได้โดยตรง

RESET คือขา reset เพื่อการีเซท 6845 เมื่อเริ่มต้นทำงาน แอคตีฟเมื่อได้รับลอจิก "0" จะเคลียร์ค่าในวงจรนับและทำให้ขาเอาท์พุททุกขาเป็น "0" แต่ไม่เคลียร์ค่าในรีจิสเตอร์ดังกล่าวมา ดังนั้นเมื่อได้รับลอจิก "1" ใหม่ มันจะเริ่มทำงานต่อไปทันที ปกติต่อกับสัญญาณรีเซทของระบบแต่อาจจัดวงจรรีเซทให้ต่างหากได้

V_{cc} และ V_{ss} คือขาไฟบวกและกราวนด์ตามลำดับ ปกติต่อกับแหล่งไฟเลี้ยงของระบบหากแยกแหล่งจ่ายต้องต่อ ขา V_{SS} คือขากราวน์ถึงกัน

ส่วนที่ติดต่อกับภาคแสดงภาพ

CLK คือขา character rate clock ต่อกับ CCLK ดังกล่าวมา เพื่อใช้เป็นจังหวะสัญญาณแก่วงจรนับต่างๆ ภายในตัว 6845 ให้เป็นสัญญาณออกเพื่อการสร้างภาพ ดังที่จะกล่าวต่อไปในหัวข้อนี้

MA_0 ถึง MA_{13} คือขา screen memory address คือขาที่จะอ้างแอดเดรสแก่วิดีโอแรม ซึ่งสามารถอ้างได้ถึง 16 กิโลไบท์ แต่ในโครงงานนี้เราใช้วิดีโอแรมขนาด 2 กิโลไบท์ จึงใช้ขาแอดเดรสเพียง 11 ขา คือ MA_0 ถึง MA_{10} ที่เหลือไม่ใช้ การต่อต้องต่อผ่านวงจรเลือก(selector : 74LS157) เพื่อให้ซีพียูเป็นผู้กำหนดว่าใคร ซีพียูหรือซีอาร์ทีซีจะเป็นผู้ติดต่อกับวิดีโอแรม

RA_0 ถึง RA_4 คือขา raster (scanline) address signal เป็นขากำหนดไลท์ที่ใน 1 บรรทัดอักขระ ของ 6845 มีถึง 5 ขาแสดงว่าเราสามารถกำหนดอักขระเป็นแมทริกซ์ได้ถึง n x 32 แต่ในโครงงานนี้เรากำหนดเพียง 8 x 8 จึงใช้พียง 3 ขา คือ RA_0 ถึง RA_2 โดยต่อตรงกับ ขา A_0 ถึง A_2 ของคาแรคเตอร์เยนฯ ตามลำดับ HSYNC คือขา horizontal synchronization ให้สัญญาณ HSYNC ตาที่กล่าวมาแต่ไม่ใช่ทุก ๆ 64 ไมโครวินาทีหากเป็นตามช่วงเวลาที่เรากำหนดด้วยการโปรแกรม และสัญญาณที่จะกล่าวต่อจากนี้เป็นสัญญาณที่เราจะต้องโปรแกรมทั้งสิ้น

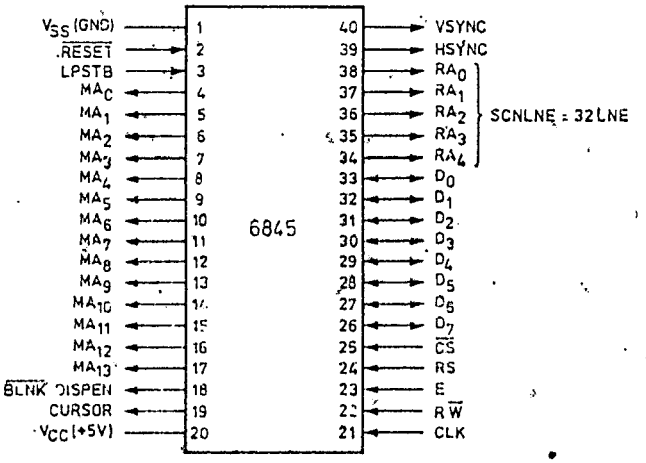
VSYNC คือขา vertical synchronization ให้สัญญาณ VSYNC

DISPEN คือขา display (video) enable ตามที่กล่าวมามีช่วงแบลงค์ที่ เราต้องให้จอภาพมืดอยู่ 2 ช่วงคือ ในทุกสแกนไลน์ก่อนและหลัง HSYNC และทุกฟิลด์ก่อนและหลัง

HSYNC และทุกฟิลด์ก่อนและหลัง VSYNC ผู้สร้าง 6845 ได้นำสัญญาณทั้ง 2 นี้มาร่วมกัน (NOR)แต่แทนที่จะให้เป็นสัญญาณในช่วงแบลงค์ กลับให้เป็นสัญญารในช่วงที่แสดงภาพได้ (OR) ทำให้ใช้สะดวกขึ้น ขอให้ดูในวงจรภาคผสมสัญญาณรูปที่ 1

CURSOR คือขา cursor enable แสดงตำบลที่ของเคอร์เซอร์ตามแต่เราจะโปรแกรม

LSTB คือขา light pen strobe ต่อกับวงจรปากกาแสงเมื่อ 6845 ได้รับ



	Pin Name	Description	Type
Microprocessor System Interface	D0-D7	Data Bus	Bidirectional Tristate
	$\overline{CS}$	Chip Select	Input
	RS	Register Select	Input
	R/$\overline{W}$	Read/Write Select	Input
	E	Enable Synchronization Signal	Input
	CLK	Character Rate Clock	Input
	$\overline{RESET}$	Reset	Input
	V_{CC}, V_{SS}	Power (+5 V) and Ground	
Screen Memory and Character Generator Signals	MA0-MA13	Screen Memory Address Bus	Output
	RA0-RA4	Raster-Scan Line Address Signals	Output
CRT Monitor Interface Signals	HSYNC	Horizontal Synchronization	Output
	VSYNC	Vertical Synchronization	Output
	DISPEN	Display (Video) Enable	Output
	CURSOR	Cursor enable	Output
	LPSTB	Light Pen Strobe	Input

รูปที่ 3 การจัดขาและหน้าที่ของ CRTC เบอร์ 6845

ล้อจิก "1" ที่ขานี้มันจะเก็บค่าแอดเดรสของวิดีโอแรมขณะนั้นคือตัวบลที่บนจอภาพขณะนั้น ซึ่งเราสามารถอ่านมาดำเนินการต่อไป เช่นทำให้บังเกิดภาพ ณ จุดที่เราจี้ปากกา แสงกับจอภาพ แต่เป็นที่น่าเสียดายว่าเป็นการแสดงภาพแบบอักขระ ไม่ใช่ในแบบกราฟฟิคจึงไม่ ค่อยใช้กัน

รีจิสเตอร์ภายใน

ใน 6845 มีรีจิสเตอร์ใช้งาน (parameter register) อยู่ 18 ตัว ดังแสดงในตารางที่ 1 มีชื่อเรียกว่า R_0 ถึง R_{17} ในการใช้งาน 6845 เราจะต้องโปรแกรมมันก่อนโดยให้ค่าแก่รีจิสเตอร์เหล่านี้ การให้ค่ารีจิสเตอร์เหล่านี้เราต้องอ้างถึงมันก่อนกับแอดเดรสรีจิสเตอร์แล้วจึงจะติดต่อมันได้ นั่นคือในการที่จะติดต่อกับรีจิสเตอร์ตัวใดตัวหนึ่งต้องใช้ข้อมูล 2 ไบท์ รีจิสเตอร์เหล่านี้มีทั้งอ่านได้ เขียนได้ หรือทั้งอ่านและเขียนได้ดังแสดงในรูป

เพื่อให้เห็นภาพการใช้งานรีจิสเตอร์เหล่านี้จะนำตัวอย่างการแสดงภาพที่กล่าวแล้วมาประกอบ และขอสรุปให้เห็นชัดเจนอีกครั้งดังนี้

– ใช้คริสตอล 17.3 MHz ผลิตความถี่เป็น DCLK และเนื่องจากให้ 1 คอมลัมน์มี 8จุดภาพ (แมทริกซ์ 8 x 8) จึงจัดวงจรหารแปดความถี่นี้เป็น CCLK ช่วงคลื่นของ CCLK จึงมีความยาวเป็น 0.46 ไมโครวินาที

– ใน 1 สแกนไลน์มี 138 คอลัมน์หรือคือ 138 CCLK แต่ใช้แสดงภาพเพียง 80CCLK เป็นช่วงแบลงค์และซิงค์ 58 CCLK

– ใน 1 ฟิลด์มี 313 สแกนไลน์แต่ใช้แสดงภาพเพียง 192 เส้นคือ 24 บรรทัดอักขระเหลือเป็นช่วงแบลงค์และซิงค์ 121 เส้น

	Register			Read (R) Write (W)	Bits	Range — Units
	R No.	Name/Function				
Horizontal Format and Timing	$0\ (00_{16})$	Horizontal Total		W	8	1 – 256 (0-FF_{16}) CLKs
	$1\ (01_{16})$	Characters Row		W	8	1 – 256 (0-FF_{16}) CLKs
	$2\ (02_{16})$	HSYNC Position		W	8	1 – 256 (0-FF_{16}) CLKs
	$3\ (03_{16})$	HSYNC Width		W	4	1 – 16 (0-F_{16}) CLKs
Vertical Format and Timing	$4\ (04_{16})$	Vertical Total		W	7	1 – 128 (0-$7F_{16}$) Character Rows
	$5\ (05_{16})$	VSYNC Adjust		W	5	1 – 32 (0-$1F_{16}$) Scan Lines
	$6\ (06_{16})$	Character Rows/Frame		W	7	1 – 128 (0-$7F_{16}$) Character Rows
	$7\ (07_{16})$	VSYNC Position		W	7	1 – 128 (0-$7F_{16}$) Character Rows
	$8\ (08_{16})$	Interlace Mode		W	2	0-3
	$9\ (09_{16})$	Scan Lines/Row		W	5	1 – 32 (0-$1F_{16}$) Scan Lines
Primary Operating Registers	$10\ (0A_{16})$	Cursor Start Scan Line		W	[1]	1 – 32 (0-$1F_{16}$) Scan Lines
	$11\ (0B_{16})$	Cursor Stop Scan Line		W	5	1 – 32 (0-$1F_{16}$) CLKs
	$12\ (0C_{16})$	(MSB)	Start Address (Top of Page)	W	6	1 – 16,384 (0000-$4FFF_{16}$)
	$13\ (0D_{16})$	LSB)		W	8	
	$14\ (0E_{16})$	(MSB)	Cursor Position	R/W	6	0 – 16,384 (0000-$4FFF_{16}$)
	$15\ (0F_{16})$	LSB)		R/W	8	
	$16\ (10_{16})$	(MSB)	Light Pen Position	R	6	0 – 16,384 (0000-$4FFF_{16}$)
	$17\ (11_{16})$	LSB)		R	8	

[1] Two bits used to specify cursor blink characteristics

ตารางที่ 1 แสดงรีจิสเตอร์ภายในของ 6845 ขอให้สังเกตในช่อง read/write ว่ารีจิสเตอร์ตัวใดอ่านได้ตัวใดเขียนได้ ช่อง BITS แสดงขนาดของรีจิสเตอร์แต่ละตัวว่าเป็นกี่บิทซึ่งจะเป็นการกำหนดว่าจะให้ค่าสูงสุดได้เท่าไรในช่องถัดไป

- หนึ่งบรรทัดอักขระมี 8 สแกนไลน์ (แมทริกซ์ 8 x 8)

จากที่ยกมาจะต้องให้รู้รีจิสเตอร์ต่างๆ ซึ่งขอกล่าวถึงเป็นส่วนรวมก่อนดังนี้ เนื่องจากการนับค่าต่างๆ เช่นใน 1 สแกนไลน์มี 138 คอลัมน์ ที่นับจาก 1 แต่ค่า ในรี จิสเตอร์ที่เราจะโปรแกรมเพื่อส่งตัวให้วงจรนับ นับจาก 0 ค่าที่เราจะให้จึงต้องลบด้วย หนึ่ง แต่ในบางเรื่องเป็นการกำหนดโดยตรงไม่ใช้การนับจึงไม่ต้องลบด้วย 1 ค่าที่ให้จึงมีทั้ง ลบหนึ่งและไม่ลบ จะให้ค่ารีจิสเตอร์ได้ดังนี้

- R_0 คือ horizontal total คือจำนวน CCLK ต่อ 1 สแกนไลน์ค่าที่ให้ คือ 137($89)

- R_1 คือ charactor/row คือจำนวนคอลัมน์ (CCLK) ต่อ 1 สแกนไลท์ที่แสดงภาพเท่ากับ 80 ($50)

- R_2 คือ HSYNC position เป็นตัวกำหนดว่าเราจะให้เริ่มต้นซิงค์ที่ CCLK ลูกที่เท่าไร ซึ่งต้องคิดถึงคาบเวลาที่เราจะให้สัญญาณซิงค์นานเท่าไร (5 ไมโครวินาที) คิดเป็นCCLK ได้ 12 ลูกจึงเหลือเป็นช่วงแบลงค์ 46 CCLK แบ่งออกด้านละเท่ากันคือ 23 CCLK ลูกที่103 ค่าที่ให้คือ 103 (S67)

- R_3 คือ HSYNC width คือความกว้างของสัญญาณซิงค์คิดเป็น CCLK คือ 12 ลูก (S0C)

- R_4 คือ vertical total คือจำนวนสแกนไลท์ทั้งหมดแต่คิดเป็นจำนวนบรรทัดอักขระ (row) ค่าที่ให้คือ 313/8 39($27) เศษ 1 ซึ่งจะนำไปให้ใน R_5

- R_5 คือ VSYNC adjust ซึ่งบ่งว่าเป็นการปรับแต่งคาบเวลาในการซิงค์และการกระทำก็เป็นดังนี้ แต่ค่าที่ให้คือเศษเป็นสแกนไลท์ที่เหลือจากการให้ค่า R_4 ในกรณีนี้คือ 1 จึงสุดแต่จะพิจารณาว่าเป็นค่าอะไร

- R_6 คือ character rows/frame หมายถึงจำนวนบรรทัดอักขระที่แสดงภาพคือ24($18)

- R_7 คือ VSYNC position กำหนดว่าเราจะให้เริ่มต้นซิงค์ที่บรรทัดอักขระที่เท่าใดจากที่กำหนดให้ จอภาพมี 313 เส้นคือ 39 บรรทัดอักขระ (เศษได้นำไปปรับแต่งแล้ว) แสดงภาพ 24 บรรทัดคงเหลือ 15 บรรทัด สำหรับ 6845 ได้กำหนดช่วง VSYNC ไว้ตายตัวคือ 16สแกนไลน์ (บวกกับที่ปรับแต่ง) คือ 2 บรรทัดเหลือเป็นช่วงแบลงค์ 13 บรรทัดจะแบ่งให้ชิดใดมากกว่าก็ได้ ผลไม่ต่างกันจนผิดสังเกต ให้แบลงค์ก่อน VSYNC เป็น 6 จะได้ R_7 เป็น 30 ($1E)

- R_8 คือ interlace mode สามารถตั้งให้ทำงานได้ 3 แบบคือ

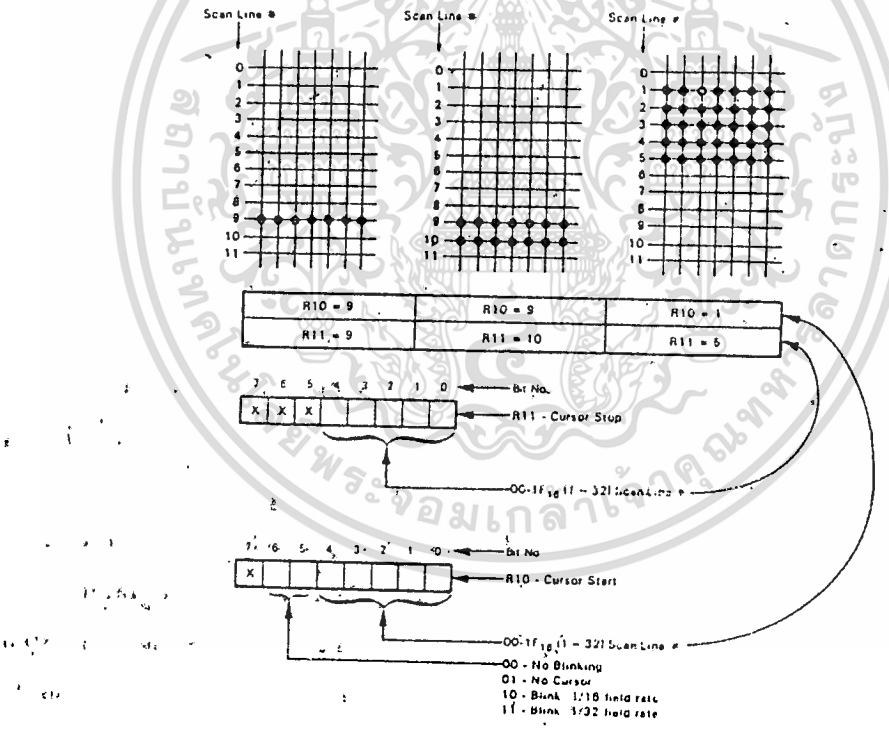
$00 คือ $02 เป็น normal mode (non-interlace)

$01 เป็น interlace sync.

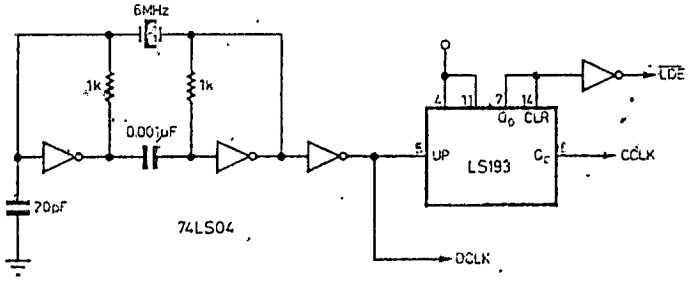
$11 เป็น interlace sync and video โดยทั่วไปให้ทำงานในแบบ normal mode ที่กล่าวมาทั้งหมดเป็นการใช้งานในโหมดนี้

- R_9 คือ scan lines/row ให้บ่งจำนวนสแกนไลท์ต่อ 1 บรรทัดอักขระคือ 8 แต่ต้องให้ค่าเป็น 7 ($07)

- R_{10} กับ R_{11} คือ cursor start กับ cursor stop เราสามารถกำหนดว่าให้เคอร์เซอร์เป็นขีดเพียงขีดเดียวหรือขีดหนาคือหลายขีดเป็นขีดล่าง ขีดกลาง หรือขีดบน หรือจะให้เป็นรูปสี่เหลี่ยมเต็มทั้งแมทริกซ์ จะให้กระพริบหรือไม่กระพริบช้าหรือเร็ว โดยกำหนดค่าในริจิสเตอร์ 2 ตัวนี้ หากต้องการให้เคอร์เซอร์เป็นรูปสี่เหลี่ยม กระพริบช้าจะต้องให้ค่า VR_{10} เท่ากับ $60 และ R_{11} เท่ากับ $07 ขอให้ดูรูปที่ 4 ประกอบ

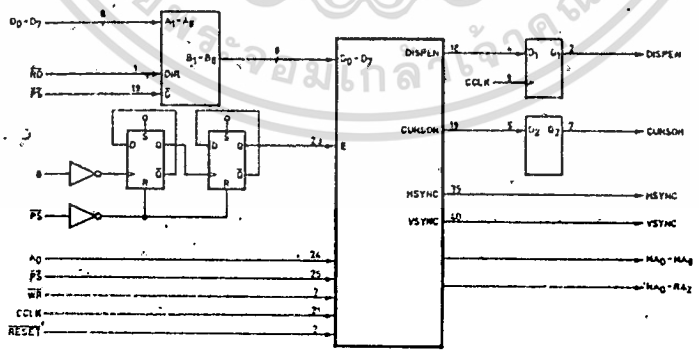


รูปที่ 4 แสดงการกำหนดค่าให้แก่ R_{10} และ R_{11} เพื่อกำหนดแบบของเคอร์เซอร์



รูปที่ 5 แสดงการจัดวงจรภาคสัญญาณนาฬิกา DCLK และ CCLK ไอซี 74LS193 ทำหน้าที่เป็นวงจรหารแปด

- R_{12} และ R_{13} คือ start address เป็นแอดเดรสเริ่มต้นของวิดีโอแรมที่จะให้ปรากฏภาพอักขระแรกที่มุมบนซ้ายตามที่ใช้หน่วยความจำ RAM ขนาด 2 กิโลไบท์ CRTC จะมองเห็นหน่วยความจำนี้ตั้งแต่แอดเดรส $000 ถึง $07FF จึงควรให้เริ่มต้น (initialize)ของแอดเดรสเริ่มต้นเป็น $0000 นั่นคือให้ค่า R_{12} และ R_{13} เป็น $00 โดยที่ R_{12} เป็นแอดเดรสไบท์สูง และ R_{13} เป็นแอดเดรสไบท์ต่ำ เช่นต้องการเลื่อนจอภาพขึ้น 1 บรรทัด แอดเดรสเริ่มต้นจะเป็น 80 คือ $0050 ต้องให้ R_{12} มีค่า $00 และ R_{13} มีค่า $50 และนี่คือวิธีเลื่อนจอภาพ (scroll)



รูปที่ 6 แสดงการจัดวงจรให้ 6845 74LS74 ต่อเพิ่มเพื่อให้ใช้ 6845 กับ Z80 ได้ แต่นักศึกษาที่ทดลองโครงงานนี้บอกว่าไม่จำเป็น หากต่อกับ 6502 ไม่ต้องใช้ไอซี 2 เบอร์นี้

ซึ่งจะต้องนำไปใช้และจะไม่กล่าวถึงอีก

- R_{14} และ R_{15} คือ cursor position เป็นตำแหน่งของเคอร์เซอร์ โดยมี R_{14} เป็นแอดเดรสไบท์สูง และ R_{15} เป็นแอดเดรสไบท์ต่ำอ้างถึงวิดีโอแรมเช่นกัน เมื่อเริ่มต้นเช่นเมื่อเคลียร์จอภาพควรให้เคอร์เซอร์อยู่มุมซ้าย ค่าที่ให้แก่ R_{14} และ R_{15} คือ $00 หมายถึงแอดเดรส $0000 ขอให้สังเกตว่าเราสามารถอ่านและเขียนกับรีจิสเตอร์ 2 ตัวนี้ได้

- R_{16} และ R_{17} คือ light pen position บอกตำแหน่งของปากกาแสงโดยอ้างถึงวิดีโอแรมเช่นกัน ทำได้ด้วยการอ่าน

ให้ค่า เริ่มต้น

รีจิสเตอร์ที่กล่าวมาทั้งหมดนี้มีรีจิสเตอร์ R_{12} ถึง R_{15} เท่านั้น โดยเฉพาะ R_{14} และ R_{15} ที่ต้องเปลี่ยนแปลงค่าตลอดเวลา แต่เมื่อโปรแกรมใช้งาน 6845 ควรให้ค่า แก่รีจิสเตอร์ทุกตัว การเขียนโปรแกรมควรกำหนดหน่วยความจำหน่วยหนึ่งเช่นให้ชื่อว่า REGTAB จะอยู่ที่แอดเดรสใดก็ได้ เก็บค่าที่เราจะโปรแกรมให้แก่ R_0 ถึง R_{15} เช่น

```
REGTAB  89  50  : R0 R1
        67  0C  : R2 R3
        27  01  : R4 R5
        18  1E  : R6 R7
        00  07  : R8 R9
        00  07  : R10R11
        00  00  : R12R13
        00  00  : R14R15
```

ที่เขียนเป็น 2 แถวเพื่อประหยัดเนื้อที่ และเขียนโปรแกรมดังโปรแกรมที่ 1 ซึ่งเขียนเป็นโปรแกรมภาษาเครื่องของ Z80 เมื่อให้ค่ารีจิสเตอร์แล้ว 6845 จะเริ่มทำงานทันที

CRTC ในวงจร

เพราะว่าไอซีที่เป็น CRTC ในรุ่นนี้ยังไม่เป็น CRTC คือวงจรควบคุมจอภาพอย่างสมบูรณ์ เราจึงต้องประกอบวงจรเพิ่มเป็นวงจรที่สมบูรณ์ตามรูปที่ 2 แต่ละรูปสี่เหลี่ยมส่วนมากจะมีความหมายสมบูรณ์ในตัวคือ ใช้ไอซีเบอร์ที่ระบุที่ต้องขยายความคือ

- วงจร DCLK และ CCLK ตามรูปที่ 5
- วงจร CRTC ดังรูปที่ 6
- สำหรับวงจร selector ใช้ 74LS157 จำนวน 3 ตัว ต่ออนุกรมกันเพื่อให้เลือกแอดเดรสได้ 11 สาย โปรแกรมที่ 1 โปรแกรมเพื่อให้ค่ารีจิสเตอร์ R_0 ถึง R_{15} ของ 6845ค่าที่จะให้เก็บอยู่ในหน่วยความจำตั้งแต่แอดเดรส REGTAB 00H ถึง + 0FH ใช้รีจิสเตอร์ Cของ OUT (ADRPRT) A เป็นการให้หมายเลขรีจิสเตอร์ที่จะติดต่อด้วยกับ addrest ส่วนคำสั่ง OUT (REGPRT),A เป็นการให้ค่าตามตารางแก่รีจิสเตอร์นั้นๆ

```
CRTPREP   LD    HL,REGPRT,A
          LD    C, 00H
          LD    B, 10H
CR1       LD    A,C
          OUT   (ADRPRT)
          LD    A, (HL)
          OUT   (REGPRT),A
          INC   C
          INC   HL
          DJNZ  CR1
```

การใช้งาน

ในการใช้งาน GRTC นอกจากโปรแกรมค่าเริ่มต้นให้แก่รีจิสเตอร์แล้วจะต้องเขียนโปรแกรมสนับสนุนการทำงานของมันที่จำเป็นได้แก่

- CURUP (cursor up) เลื่อนเคอร์เซอร์ขึ้นบน 1 บรรทัดอักขระ
- LF (linefeed) เลื่อนเคอร์เซอร์ลงล่าง 1 บรรทัดอักขระ
- CURLFT (cursor left) เลื่อนเคอร์เซอร์ซ้าย 1 อักขระ
- CURRGHT (cursor light) เลื่อนเคอร์เซอร์ไปต้นบรรทัดอักขระ
- CR (caniage return) เลื่อนเคอร์เซอร์ไปต้นบรรทัดอักขระ
- SCROLL (scrolling) เพื่อเลื่อนภาพทั้งหมดขึ้นบน 1 บรรทัดอักขระ
- HOME เคลียร์จอภาพคือให้ภาคคำทั้งจอภาพ และให้เคอร์เซอร์อยู่ที่ริมซ้ายบน

8251/9551-SIO

พอร์ทแบบอนุกรม (SIO: Input/Output Port) เป็นพอร์ทที่มีข้อมูลแบบขนานจากซีพียูแล้วตัวมันจะมาจับเรียงเป็นข้อมูลแบบอนุกรมส่งให้กับอุปกรณ์ภายนอก หรือในทางตรงข้าม มันจะรับข้อมูลแบบอนุกรมจากอุปกรณ์ภายนอกมาแปลงเป็นข้อมูลแบบขนานส่งให้กับซีพียู

เท่าที่กล่าวมานี้เป็นการมอง 8251 และ 9551 ในฐานะของพอร์ท แต่ในการรับส่งข้อมูลแบบอนุกรมจะไปเกี่ยวข้องกับการสื่อสารข้อมูลเป็นส่วนใหญ่ จึงได้ชื่อว่าเป็นตัวรับ - ส่ง(Receiver / Transmitter) นอกจากนี้ในการสื่อสารข้อมูลยังแบ่งประเภทของการรับส่งเป็นแบบ synchronous กับ asynchronous ซึ่งอุปกรณ์ที่ดำเนินการในเรื่องนี้อาจทำไม่ได้ทั้งหมดตามที่กล่าวมา แต่ไอซี 2 ตัวนี้ทำได้ จึงได้ชื่อว่าเป็น USART (U niversal Synchronous/Asynchronous Receiver/Transmitter)

ในการกล่าวถึงไอซีสองตัวนี้ควรจะกล่าวถึง 8251 เป็นหลัก และพูดถึง 9551 ภายหลัง เพราะ 8251 เป็นต้นแบบที่อินเทลคิดค้นขึ้นเพื่อใช้กับซีพียูตระกูล 8080 (รวมทั้ง Z-80)และมีจำหน่ายแพร่หลายในบ้านเรา ส่วน AMD 9551 ของ Advance Micro Devices ได้พัฒนามาจาก 8251 แต่เพราะความเป็นต้นแบบจะทำให้เข้าใจยาก จึงขอพูดถึง 9551 ก่อน แล้วพูดถึง 8251 ภายหลังส่วนในคำอธิบายขอรวมเรียกว่า USART

ซิงค์กับอะซิงค์

ดังกล่าวแล้วว่ารูปแบบของการ รับ-ส่งข้อมูลทำได้ 2 แบบคือ synchronous กับ asynchronous ซึ่งเป็นเรื่องยืดยาวพอสมควร จึงขอเรียนให้ทราบเฉพาะที่เกี่ยวข้องกับการศึกษาไอซี 2 ตัวนี้การส่งแบบ saynchronous มีรูปแบบดังรูปที่ 1 คืออักขระ (character)ทุกๆตัวจะถูกนำด้วยบิทนำ (start bit) โดยทั่วไปให้มีค่าเป็น "0" และปิดท้ายด้วยบิทตาม(stop bit) โดยทั่วไปให้มีค่าเป็น "1" ลักษณะการทำงานคือเมื่อยังไม่ส่งสัญญาณ สายส่งจะเป็น "H" คือ "1" เรียกว่า mark เมื่อเริ่มส่งจะเป็นบิทนำคือ "0" อยู่หนึ่งบิท ต่อไปจะเป็นข้อมูลหนึ่งอักขระจะมีกี่บิทแล้วแต่จะเลือกส่วนบิทใดจะเป็น "0" หรือ "1" อยู่หนึ่งถึงสองบิทหากต้องการส่งต่อไปก็จะเป็นบิทนำอีกถ้าหยุดส่งจะเป็น mark

สำหรับการส่งแบบ synchronous ไม่ใช้บิทนำและบิทตาม แต่จะส่งรหัสที่เรียกว่า SYNC characters จากที่กล่าวมานี้มองไม่เห็นว่ามันซิงค์หรือไม่ซิงค์กันตรงไหน และ ยิ่งดูรูปว่างสัญญาณอันเกี่ยวข้องกับสัญญาณนาฬิกาด้วยแล้วกลับเห็นว่า asynchronous กลับมีความเป็นซิงค์ในความรู้สึกของมากกว่าคิดว่าเข้าใจความแตกต่างของมันก็เพียงพอแล้ว

ที่กล่าวอักขระหนึ่งๆ จะมีกี่บิทตามแต่จะเลือกหรือตกลงกันระหว่างผู้กับผู้รับ ซึ่งมีให้เลือกตั้งแต่ 5 ถึง 8 บิท สำหรับ 5 บิท ไม่ทราบเพราะไม่คุ้นกับ EBCDIC ถ้าเป็น ASCII อักขระ 6 บิท จะส่งอักษรอังกฤษตัวใหญ่ รวมทั้งตัวเลขและสัญญาณในการควบคุมได้ ทั้งหมดรวมทั้งเครื่องหมายบางส่วน ถ้า 7 บิทจะได้อักษรเล็กกับเครื่องหมายที่เหลือ ถ้าเป็น 8 บิท

การใช้ USART

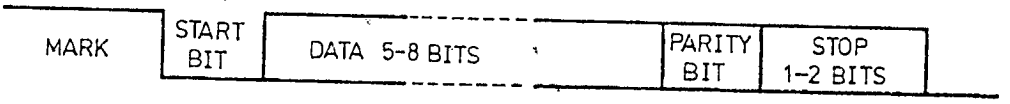
ปกติเมื่อกล่าวถึงชิพซัพพอร์ทเบอร์ใดจะกล่าวถึงเรื่องราวของมันเสียก่อน แล้วกล่าวถึงการใช้งาน แต่สำหรับเรื่องนี้เห็นว่าควรดำเนินการตรงข้ามเขาสร้าง USART หรือ SIOมาเพื่อให้สามารถส่งข้อมูลได้ไกลขึ้น แต่ขอเรียนย้ำทำความเข้าใจว่าตัวมันเอง ไม่ได้ทำให้ไกลขึ้น มันเพียงแต่เปลี่ยนข้อมูลจากซีพียูที่เดินมาแบบหน้ากระดานให้เป็นแบบเรียง ตามก้นเป็นแถวเรียงหนึ่งหากเรานำเอาท์พุทของมันไปต่อกับอุปกรณ์ใดก็ตามดื้อ เหมือนอย่าง ทำกับพอร์ทแบบขนานก็ลงไปได้ประมาณ 1 เมตรเช่นกัน

แต่ถ้าเราต้องการส่งข้อมูลในลักษณะไกล เช่น ให้เครื่องพิมพ์ที่อยู่ห่างกันไปสัก 10 เมตร เราต้องนำขาเอาท์พุทไปเข้าวงจรปรับระดับแรงดันหรือกระแสดังที่ใช้กันในข้อต่อมาตรฐานเช่น IEEE483 หรือกรรมวิธีอื่นอีกโดยวิธีนี้ระยะใช้งานไม่เกิน 200 เมตรเกินไปกว่านี้ก็หวังผลไม่ได้แน่นอนทางกลับกันคือ ซีพียูเป็นตัวรับก็เช่นกัน

หากจะให้ไกลต่อไปอีกก็ต้องนำไปผสมคลื่นพาห์ที่รู้จักกันแพร่หลาย MODEM (Modulator/Demodulator) แล้วเข้าสายโทรศัพท์ขององค์การ ถ้าจะให้ไกลต่อไปอีกก็ต้องไปผสมกับคลื่นพาห์ที่เป็นคลื่นวิทยุออกอากาศไปจะเป็นขอบสุริยาจักรวาลก็ยังไหว

ด้วยวิธีการเช่นนี้จึงไม่มีใครทำกับข้อมูลแบบขนาน เพราะถ้าข้อมูลขนาด 8 บิท จะต้องใช้โทรศัพท์ถึง 8 คู่สายหรือเครื่องวิทยุถึง 8 เครื่อง จึงเอาดีกันด้วยเร่งอัตราในการส่งที่เรียกว่า บวดเรท (baud rate) คือจำนวนบิทต่อวินาที การรับส่งข้อมูลในแบบนี้ไม่คิดเป็นอักขระต่อวินาที เพราะเราอาจเลือกส่งกี่บิทต่ออักขระก็ได้ และอาจเป็นมีบิทนำหรือตามหรือไม่ก็ได้บิทตามจะมีกี่บิทยังกำหนดได้อีก นอกจากนี้ยังกำหนดให้มีบิทตรวจพาริตี้ (parity) หรือไม่ก็ได้อีก

การตรวจพาริตี้ขึ้นอยู่กับการกำหนดว่าข้อมูล 1 อักขระของเราจะให้เป็นจำนวนคู่หรือคี่เช่น ให้เป็นจำนวนคี่ ถ้าไม่มีข้อมูลนั้นมีบิท "1" (จะคิดเป็นบิท "0" ก็ได้) อยู่ 4 ตัว เป็นจำนวนคู่ก็ให้บิทพาริตี้



รูปที่ 1 รูปแบบของการส่งสัญญาณอนุกรมแบบ asynchronous

บิท "1" จะได้เป็นคี่ ถ้าเป็นจำนวนคี่อยู่แล้วก็ให้เป็น "0" เพื่อการตรวจสอบ คิดค้นเท่านั้น ว่าข้อมูลที่ส่งมานั้นขาดหายไปหรือมีสัญญาณแปลกปลอมแทรกทำมาหรือไม่. แต่ถ้า หายไปหรือแทรกทำมาเป็นจำนวนคู่ก็ตรวจไม่พบ

บอดเรทขั้นต่ำประมาณ 100 ที่ชอบพูดกันเป็นมาตราฐานคือ 110 ขั้นสูงสุดไว้เป็นหมื่นแต่ไม่เคยเห็นที่ส่งทางสายได้อย่างเก่งประมาณ 1000 ทั้งนี้เพราะต้องให้น้อยกว่า คลื่นพาห์ ซึ่งทางสายอย่างเก่งเพียง 7-8 กิโลเฮริทซ์หากคุณสงสัยว่าทำไมไม่ส่งสัญญาณดิจิตอลไปทางสายโดยตรง เพราะตามตัวเลขที่ยกมาอาจส่งได้ถึง 14000-16000 บอดเรท ท่านผู้รับบอกว่าสัญญาณรูปซายน์ไปได้ไกลกว่ารูปสี่เหลี่ยมซึ่งไปได้สั้นนิดเดียว

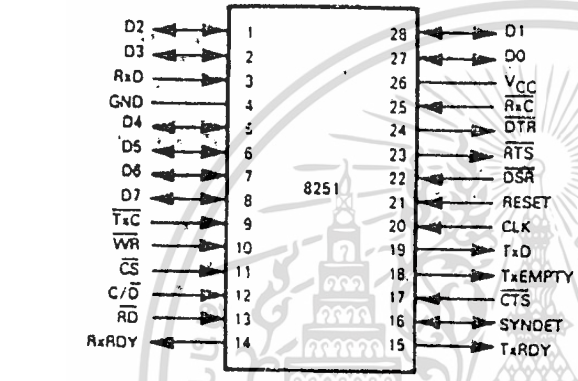
ขาและหน้าที่

เป็นไอซีขนาด 28 ขาดังแสดงการขัดขาและแผนผังภายในของ 8251 ไว้ในรูปที่ 2ซึ่งมีเรื่องที่จะเรียนให้ทราบก่อนว่า ชื่อขาบางขาจะซ้ำกับชื่อรีจิสเตอร์และทำหน้าที่เดียวกันด้วยแต่สัญญาณอาจกลับตรงข้ามกัน โปรดสังเกตด้วย อีกเรื่องหนึ่งคือขาในภาครับและภาคส่ง ผู้ที่คุ้นกับพอร์ทแบบขนานมาก่อนจะชวนให้คิดว่าเป็นขาที่ติดต่อกับอุปกรณ์ภายนอกทั้งหมด ความจริงไม่ใช่มีขาที่เป็นอินพุทและเอาท์พุทภาคละ 1 ขาเท่านั้นที่ติดต่อกับอุปกรณ์ภายนอก นอกนั้นติดต่อกับซีพียูขาที่ติดต่อกับอุปกรณ์ภายนอกและทำหน้าที่คล้ายกับขา handshaking อยู่ในส่วนที่ติดต่อกับโมเด็ม(MODEM) ซึ่งคุณจะใช้โมเด็มหรือไม่ ก็ไม่ต้องต่อมันได้เช่นกัน

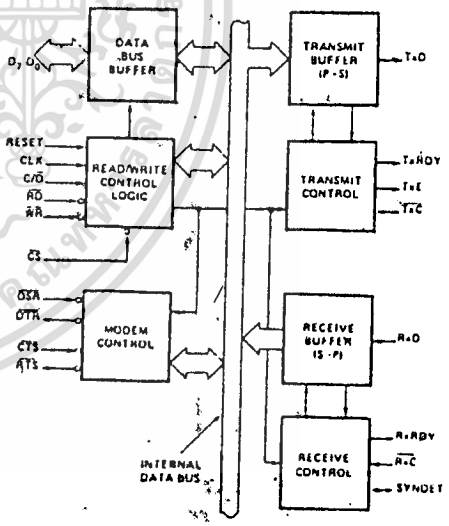
ในการอธิบายขาต่อไปนี้ กรุณาถือตามข้อกำหนดของอินเทลด้วยคือ ขาที่มีบาร์ (ขีดบน)จะแอคตีฟที่ลอจิก "1" ยกเว้นขาข้อมูล ขาสัญญาณนาฬิกาและไฟเลี้ยง

ส่วนที่ติดต่อกับซีพียู

* DATA BUS คือขา D_0 ถึง D_7 ต่อกับขาดาต้าของซีพียู ขานี้เมื่อซีพียูเขียนจะเป็นการให้คอนโทรลโค๊ด หรือข้อมูล แต่เมื่อซีพียูอ่านจะเป็นการอ่านสเตตัสของ USART หรือข้อมูลUSART ตัวหนึ่งๆ จึงเปรียบเสมือนมี 2 พอร์ท หรือ 2 หน่วยความจำ โดยพอร์ทหนึ่งเป็นพอร์ทข้อมูลซึ่งเป็นทั้งอินพุทและเอาทีพุทพอร์ท อีกพอร์ทหนึ่งเป็นพอร์ทควบคุมกับสเตตัส



PIN NAME	DESCRIPTION	TYPE
D0 - D7	Data Bus	Bidirectional
RESET	System reset	Input
CLK	Device Clock	Input
C/D	Control or Data select	Input
RD	Read data or status on Data Bus	Input
WR	Write Data or Control on Data Bus	Input
CS	Chip Select	Input
DSR	Data set ready indicator	Input
DTR	Data terminal ready indicator	Output
CTS	Clear to send data indicator	Input
RTS	Request to send data indicator	Output
TxD	Serial data output	Output
TxRDY	Transmitter ready to receive data on Data Bus	Output
TxEMPTY	Transmitter empty. Has no data to transmit	Output
TxC	Serial transmit data clock	Input
RxD	Serial data input	Input
RxRDY	A character is ready to transmit on Data Bus	Output
RxC	Serial data input clock	Input
SYNDET	Synchronous data detect or force	Bidirectional
VCC, GND	Power and Ground	



รูปที่ 2 แสดงการจัดขาและแผนผังภายในของ 8251 และ 9951

* ขา C/D เป็นการเลือกว่าจะติดต่อกับพอร์ทควบคุมกับสเตตัสหรือติดต่อกับพอร์ทข้อมูลโดยทั่วไปต่อกับขา A_0 ของซีพียู ดังนั้นพอร์ทหมายเลขคู่จึงเป็นพอร์ทข้อมูง และหมายเลขคี่คือพอร์ทควบคุม

* ขา RD และ WR เป็นการเลือกว่าซีพียูจะอ่านหรือเขียน ดังนั้นจึงเป็นการเลือกว่าจะอินพุทหรือเอาท์พุทข้อมูลกับเลือกว่าจะให้ค่าคอนโทรลโค๊ดหรือจะอ่านสเตตัส

* ขา CS เป็นการเลือกตัวมันได้มาจากการดีโค๊ดพอร์ทหรือแอดเดรสตามแบบการเลือกชิพ ชัพพอร์ทที่เคยกล่าวแล้ว

* ขา CLK คือขาสัญญาณนาฬิกาของระบบต่อมาจากที่จ่ายให้กับซีพียูไม่เกี่ยวกับสัญญาณนาฬิการที่กำหนดบวดเรทดังจะกล่าวต่อไป

* ขา RESET เป็นขารีเซ็ทซึ่ง USART จะถูกรีเซ็ทเมื่อขานี้ได้รับลอจิก "1" อาจต่อมาจากขารีเซ็ทของซีพียู (ถ้าเป็น Z-80 ต้องกลับลอจิก) หรือแยกต่างหาก

* ขาไฟเลี้ยง คือ GND กับ Vcc (0 กับ+ 5 โวลท์) ความจริงไม่ได้ต่อกับซีพียูโดยตรง แต่ส่วนมากใช้ไฟเลี้ยงของระบบจึงจัดไว้ในหัวข้อนี้ด้วย

ภาคส่งสัญญาณ (Serial output)

* ขา TxD คือขาข้อมูลออกแบบอนุกรมต่อกับอุปกรณ์ภายนอก

* ขา Tx RDY แสดงว่า USART พร้อมจะรับข้อมูลจากซีพียู แต่ในการใช้งานจะใช้ขานี้เป็นขาอินเตอร์รัพท์ซีพียูเพื่อแสดงว่าข้อมูลใน USART ได้ถูกส่งไปแล้ว ให้ซีพียูส่งข้อมูลใหม่ (แต่อาจไม่ต่อก็ได้โดยให้ซีพียูทำงานในแบบพูลลิง ดังจะกล่าวในรายละเอียดต่อไป)

* ขา TxEMPTY แสดงว่าข้อมูลในรีจิสเตอร์ TB ได้ถูกส่งไปแล้ว จะกล่าวในรายละเอียดต่อไป

* ขา TxC เป็นขาสัญญารนาฬิกาที่จ่ายให้แก่ภาคส่งสัญญาณซึ่งจะนำไปกำหนดบวดเรทต่อไป สัญญาณนี้ส่วนมากไม่ได้ต่อมาจากสัญญาณนาฬิกาของระบบซีพียู เพราะต้องการความถี่ตามแต่ จะกำหนดกับอุปกรณ์ภายนอก ซึ่งแม้จะนำสัญญาณนาฬิกาของระบบมาหารก็ไม่ตรง สัญญาณ นาฬิกานี้สำหรับ 9551 ต้องน้อยกว่าของซีพียู 4.5 เท่าส่วนของ 8251 ต้องน้อยกว่า 13 เท่าเมื่อน้อยกว่า 30 เท่าเมื่อทำงานใน synchronous mode

ภาครับสัญญาณ (Serial Input)

* ขา RxD คือขาข้อมูลเข้าแบบอนุกรม

* ขา RxRDY แสดงว่า USART พร้อมจะส่งข้อมูลให้แก่ซีพียู ใช้เป็นขาอินเตอร์รัพท์เช่นเดียวกับ TxRDY

* ขา SYNDET ใช้ใน synchronous mode เท่านั้น สามารถโปรแกรมให้เป็นได้ทั้งขาอินพุทและเอาท์พุท เมื่อเป็นเอาท์พุทจะให้สถานะ "1" เมื่อภาครับสัญญาณตรวจจับ SYNC character ได้และจะเป็น "0" อีกครั้งเมื่อซีพียูอ่านสเตตัส นอกจากนี้ยังทำหน้าที่เป็น break detect ด้วยถ้าดาต้าที่ได้รับมามีค่าเป็น "0" ทั้งอักขระคือตั้งแต่บิทนำจนถึงบิทตามหนึ่งบิท จะทำให้ขานี้เป็น "1" เช่นกัน และจะกลับเป็น "0" อีกครั้งเมื่อสัญญาณที่ขาข้อมูลคือ RxD เป็น "1"ใหม่เมื่อให้เป็นอินพุทเรียกว่า external SYNC ถ้ามีสัญญาณขอบขาขึ้นคือ เกิดขึ้นที่ขามัน ตัวรับจะถือว่าสัญญาณที่ขาดาต้าเป็นดาต้าทันทีเรียกว่าเป็นการซิงค์ และเมื่อเป็นซิงค์แล้วสามารถรับสัญญาณ "H" จากขานี้ได้

* ขา RxC เป็นขาสัญญาณนาฬิกา เช่นเดียวกับ TxC และส่วนมากใช้สัญญาณผลิตสัญญาณตัวเดียวกัน

ภาคติดต่อกับโมเด็ม

ดังกล่าวแล้วว่าหากเราไม่ใช้ก็อาจใช้สัญญาณนี้ไปทำงานใดก็ได้ถ้าคำอธิบายของไม่เกี่ยวกับโมเด็มโดยตรง แต่จะกล่าวถึงการทำงานของมันในฐานะไอซีตัวหนึ่งเท่านั้น หากต้องการทราบรายละเอียดให้ดูได้ในเรื่องที่เกี่ยวกับโมเด็มหรือ RS232

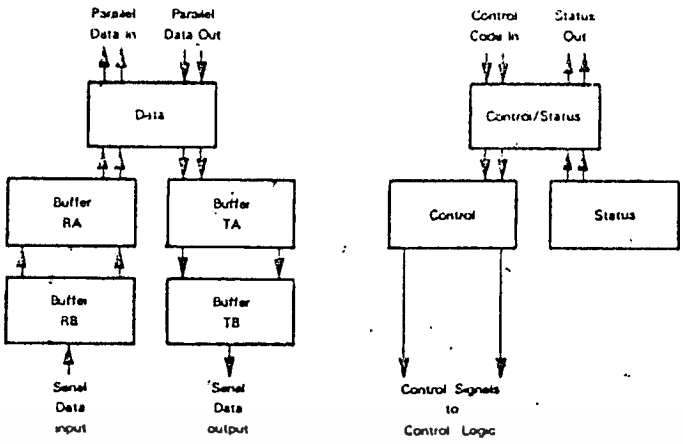
* ขา DSR (Data Set Ready) เป็นขาอินพุทจากอุปกรณ์ภายนอกซึ่งซีพียูจะอ่านได้จากสเตตัสรีจิสเตอร์ ใช้ในความหมายว่าอุปกรณ์ภายนอกพร้อมที่จะติดต่อด้วย

* ขา DTR (Data Terminal Ready) เป็นขาเอาท์พุทจาก USART ให้แกอุปกรณ์ภายนอกโดยซีพียูเป็นผู้ส่งใช้ในความหมายว่าซีพียูพร้อมจะติดต่อด้วยแต่อาจนำไปใช้ในการเปิดไฟเข้าอุปกรณ์ภายนอกก็ได้

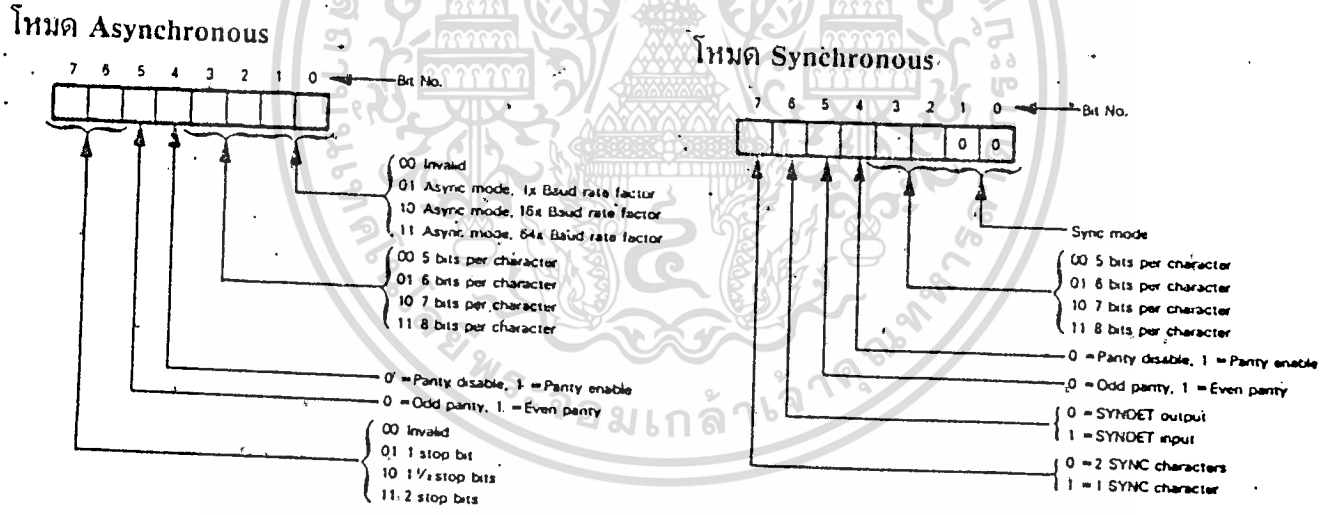
* CTS (Clear to Send) เป็นขาอินพุท ใช้ในความหมายว่าให้เริ่มการติดต่อได้แต่หากไม่ได้ต่อกับโมเด็มหรืออุปกรณ์อื่นใดจะต้องให้ลอจิก "0" แต่มัน ไม่เช่นนั้นภาคส่งสัญญาณจะไม่ทำงาน

* ขา RTS (Request to Send) เป็นขาเอาท์พุทจาก USART ให้แก่อุปกรณ์ภายนอกโดยซีพียูเป็นผู้สั่งเช่นเดียวกับ DTR

ของเรียนอีกครั้งว่าคำอธิบายของไม่เกี่ยวกับชื่อของเหล่านี้ เพราะเป็นเรื่อง ของโมเด็ม และจะเห็นว่านอกจากขา CTS แล้วไม่มีขาใดเกี่ยวข้องกับการทำงานของ USART เลยเปรียบได้



รูปที่ 3 การจัดบัฟเฟอร์และรีจิสเตอร์ของ 9951



รูปที่ 4 ความหมายของบิทต่างๆ ใน control byte

เสมือนกับพอร์ทของซีพียูเท่านั้น เพียงแต่ซีพียูดำเนินการโดยตรงไม่ได้ ต้องผ่าน control register หรือ status register

โครงสร้างภายใน

โครงสร้างภายในของ 9551 ในส่วนที่จะเกี่ยวกับคำอธิบายได้แสดงในรูปที่ 3 จะเห็นว่าเปรียบเสมือนเป็น 2 ส่วนใหญ่ หรือ 4 ส่วนย่อย ซึ่งแต่ละส่วนจะทำงานดังนี้

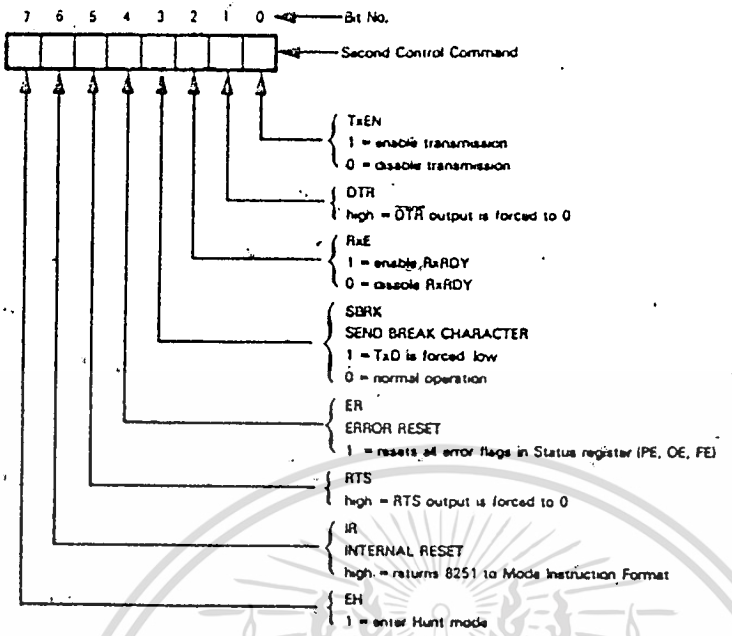
ส่วน control

รวมอยู่ในส่วนที่เรียกว่า control and status register เมื่อซีพียูอ้างถึงพอร์ทนี้คือพอร์ทหมายเลขคี่ และให้สัญญาณ WR จะเป็นการติดต่อกับ control register ซึ่งยังแบ่งย่อยออกเป็น control กับ command เมื่อ USART ถูกรีเซ็ทเช่นเมื่อเริ่มเปิดไฟเลี้ยงหรือเมื่อถูก internal reset ซึ่งจะกล่าวต่อไป มันจะถือว่าข้อมูลไบท์แรกที่ซีพียูส่งให้แก่ control register จะเป็นข้อมูลซึ่งเป็น control byte เพื่อเป็นการเลือกโหมดการทำงานของUSART ดังแสดงในรูที่ 4 คือ ถ้าบิท 0 และ บิท 1 เป็น "00" หมายความว่าให้ทำงานในซิงค์โหมดถ้าเป็นค่าอื่นทำงานในอะซิงค์โหมด และมีรายละเอียดต่อไปคือ

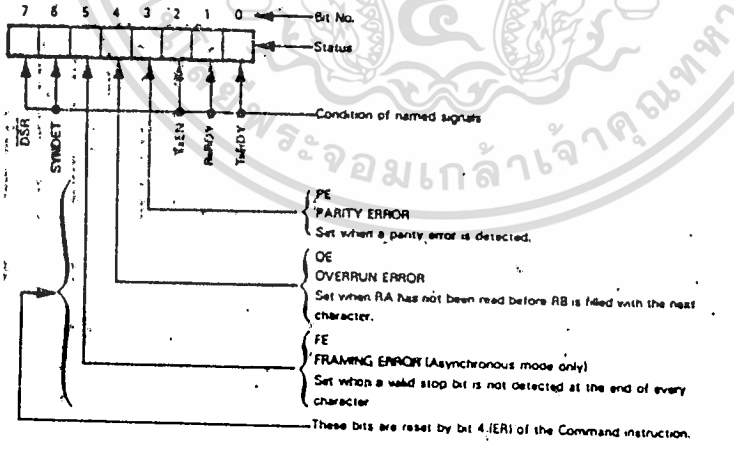
* อะซิงค์โหมด
บิท 0 - บิท 1 เลือกบอดเรท
บิท 2 - บิท 3 เลือกจำนวนบิทต่ออักขระ
บิท 4 - บิท 5 เลือกพาริตี้
บิท 6 - บิท 7 เลือกจำนวนบิทของบิทตาม
* ซิงค์โหมด
บิท 0 - บิท 1 เป็น 00
บิท 2 - บิท 5 (เช่นเดียวกับอะซิงค์โหมด)
บิท 6 เลือก SYNDET ให้เป็นอินพุทหรือเอาท์พุท
บิท 7 เลือกจำนวนอักขระของ SYNC character

ส่วน command

หลัง control byte แล้วหากเลือกซิงค์โหมด USART จะถือว่าข้อมูลที่ตามติดมาอีก2 ไบท์เป็น SYNC character หลังจากนี้หรือเมื่อเลือกอะซิงค์โหมดจะถือว่าข้อมูบที่ตามมาคือcommand byte และหลังจากนี้หากเลือกพอร์ทนี้แล้ว WR จะถือว่าเป็น command byte ทั้งสิ้นจนกว่าจะ internal reset บิทต่างๆ ในไบท์นี้มีชื่อและความหมายดังนี้ ซึ่งจะไม่กล่าวตามลำดับบิทแต่จะกล่าวเรียงเป็นกลุ่มไป คือ (ขอให้ดูรูปที่ 5 ประกอบด้วย)



รูปที่ 5 ความหมายของบิทต่างๆ ใน command byte



รูปที่ 6 ความหมายของบิทต่างๆ ในเสตตัสรีจิสเตอร์

บิท 0 TxEN teransmitter

บิท 3 SBRK (send break cjstsctrt)ให้ภาคส่งหยุดทำงานและให้ขา เป็น "0"

บิท 2 RxE ให้ขา RxD ทำงานจะเห็นว่าซีพียูไม่ควบคุมภาครับโดยตรงจะรับรู้โดยให้ RxD ทำงาน

บิท 7 EH (enable hunt mode)ให้ภาครับทำงานใน hunt mode จะกล่าวต่อไป

บิท 1 DTR ให้ขา DTR เป็นลอจิก "0" ดังกล่าวมาแล้ว

บิท 5 RTS ให้ขา RTS เป็นลอจิก "0" ดังกล่าวมาแล้ว

บิท 4 ER (crrror reset) เป็นการรีเซ็ท error flag ในสเตตัสรีจิสเตอร์ ได้แก่ PE,OE, และ FE

บิท 6 IR (crror reset)เป็นการรีเซ็ทค่าใน control และ command รีจิสเตอร์ใหม่ โดยถือว่าเริ่มต้นไบท์เป็น control byte ใหม่ดังกล่าวแล้ว

สเตตัสรีจิสเตอร์

เมื่อซีพียูอ้างถึงพอร์ทที่เป็น control and status register และเมื่ออ่าน RD จะเป็นการอ่านข้อมูลที่เก็บอยู่ในสเตตัสรีจิสเตอร์ ข้อมูลเหล่านี้เกิดขึ้นจากผลของการทำงานของUSART เองซึ่งแต่ละบิทเรียกว่าแฟลก (flag) ดังแสดงในรูปที่ 6 ซึ่งสามารถแบ่งได้เป็นจำพวกและมีความหมายดังนี้

พวกที่แสดงสถานะที่ขาของ USART แฟลกพวกนี้จะมีชื่อเดียวกันกับขา (ยกเว้น TxEN)จุดประสงค์เพื่อแสดงสถานะของขาเหล่านี้ เมื่อเราให้ซีพียูทำงานในแบบ pooling mode คือไม่ต่อขาเหล่านี้ให้อินเตอร์รัพท์ซีพียูโดยตรง แต่ให้ซีพียูแวะเวียนมาอ่านสถานะของขาเหล่านี้เอาเองหรือเมื่อเราต่อขาเหล่านี้รวมกันเช่นผ่าน OR เกทเข้าอินเตอร์รัพท์ซีพียู เมื่อซีพียูถูกอินเตอร์รัพท์จะมาตรวจที่แฟลกนี้ว่าขาอินเตอร์รัพท์ ค่าที่แฟลกเหล่านี้จะตรงตามขาคือเป็น "1" เมื่อที่ขาเป็น"L" และเป็น "0"1 เมื่อที่ขาเป็น "L" ได้แก่ TxRDY, RxRDY,SYNDET, DSR ส่วน TxENเกิดจากคำสั่งใน command

พวกที่เป็นผลมาจากการทำงาน คือ พวก error flag จะให้ค่าเป็น "1" เมื่อมีการผิดพลาดเกิดขึ้น ได้แก่

- PE (parity error) มีค่าเป็น "1" เมื่อตรวจสอบพาริตี้แล้วพบว่าผิด
- OE (overrun error) มีค่าเป็น "1" เมื่อซีพียูไม่ได้อ่านข้อมูลทางภาคจอและมีข้อมูลใหม่เข้ามาทับ

- FE (framing error) มีค่าเป็น "1" เมื่อตรวจไม่พบบิทตาม (stop bit) ตัวอักขระแต่ละตัว

ค่าของแฟลกเหล่านี้เมื่อเกิดขึ้นแล้วจะถูกยกเลิกได้ต่อเมื่อสั่ง ER (error) ใน command byte

ภาครับสัญญาณ

กรุณาย้อนกลับไปดูรูปที่ 3 อีกครั้ง ภาครับสัญญาณประกอบด้วยบัฟเฟอร์ RAและ RB กับใช้ดาต้ารีจิสเตอร์ร่วมกับภาคส่ง สัญญาณคือข้อมูลจะเข้าทางขา RxD (Serial Data Lnput) จึง RB จะจัดเรียงสัญญาณแบบอนุกรมให้เป็นแบบขนาน เมื่อได้รับข้อมูล ครบถ้วน จะส่งให้ RA ทันที เพื่อรอให้ซีพียูผ่านขาดต้า การทำงานในช่วงตอนนี้มีข้อแตกต่าง กันในระหว่าง 2 โหมดคือ

* อะซิงค์ จะตรวจจับสัญญาณตั้งแต่บิทนำจนถึงบิทตามเมื่อได้ครบสัญญาณตั้งแต่บิทนำจนถึงบิทตามเมื่อได้ครบถ้วนจะเรียงเฉพาะข้อมูลให้แก่ RA ถ้าไม่ครบถ้วน ซึ่งมีทางรู้ทางเดียวคือหาบิทตามไม่พบจะส่งข้อมูลเท่าที่จัดเรียงได้ ซึ่งอาจเป็นข้อมูลที่ผิดได้ RA แล้วเซ็ทแฟลก FE แสดงว่ามี framing error เกิดขึ้น

* ซิงค์ ไม่มีบิทนำและบิทตาม แต่จะพยายามซิงค์กับ SYNC character โดยเลื่อนข้อมูลที่เข้ามาที่ละบิท แล้วเทียบกับ SYNC character ที่ได้โปรแกรมไว้เมื่อตรงกันจะให้SYNCDET ซึ่งซีพียูจะต้องทราบ (จากการจัดวงจรอินเตอร์รัพท์หรือให้อ่านจากสเตตัสรีจิสเตอร์ดังกล่าวแล้ว) เพราะข้อมูลต่อๆ ไป USART จะถือว่าเป็นอักขระ แล้วส่งจาก RB ให้ RA ทันที การที่จะให้ USART กับ SYNC character เราจะต้องให้มันทำงานใน hunt mode ด้วยการโปรแกรมใน command byte ส่วนการออกจาก hunt mode ตัว USART จะดำเนินการเองเมื่อซิงค์ได้

ภาคส่งสัญญาณ

ประกอบด้วยบัฟเฟอร์ TA และ TB กับใช้ดาต้ารีจิสเตอร์ร่วกับภาครับเมื่อซีพียูจะส่งข้อมูลออกจะอ้างถึงพอร์ทดาต้าและให้สัญญาณ WR ข้อมูลจะถูกส่งผ่านดาต้าบัสให้แก่บัฟเฟอร์ TA ซึ่งจะส่งต่อให้ TB จัดเรียงเป็นข้อมูลแบบอนุกรมส่งออกทางขา TxD ในการจัดเรียงข้อมูลหากเราเลือกให้ส่งน้อยกว่า 8 บิทต่ออักขระ มันจะตัดบิทสูงออและกำหนดให้มี parity มันจะเพิ่มบัทนี้โดยให้จำนวนบิทที่เป็น "1" มีค่าเป็นคู่หรือเป็นคี่ตามที่โปรแกรมไว้ การดำเนินการในแต่ละโหมดมีข้อแตกต่างกันดังนี้

* อะซิงค์ จะเพิ่มบิทนำและบิทตามให้แก่ทุกอักขระ

* ซิงค์ ไม่มีบิทนำและบิทตามแต่เมื่อซีพียูสั่งให้ส่งได้มันจะส่ง SYNC character ไปจนกว่าซีพียูจะให้ข้อมูลแก่มัน

สำหรับอะซิงค์ เราจะต้องเตรียมข้อมูลไว้ 2 ไบท์ คือใน TB และ TA ก่อนที่จะให้USART ส่งข้อมูลออก

การทำงานของภาคส่งสัญญาณ

การทำงานเริ่มต้นและข้อแตกต่างของทั้ง 2 โหมด ได้กล่าวไว้ในหัวข้อโครงสร้างภายในแล้ว ในหัวข้อนี้จะกล่าวถึงเรื่องของการส่ง ซึ่งเหมือนกันทั้ง 2 โหมดโดยจะเริ่มจากภาคส่งสัญญาณได้รับคำสั่งให้ส่งคือ TxEN และขา CTS เป็น "L" ซึ่งมีขั้นตอนดังนี้ (เมื่อกล่าวถึงขาให้หมายถึงแฟลกที่ชื่อเดียวกันด้วย)

- ข้อมูลใน TB จะถูกส่งออกทันที (ในซิงค์หมายความว่าได้ออกจาก hunt mode แล้ว)

- เมื่อข้อมูลใน TB หมดมันจะรับจาก TA ทันที ช่วงจังหวะนี้มันจะทำให้ขา TxE เป็น"H" (เป็นสัญญาณที่ผู้ใช้ 8251 อาจต้องนำไปใช้ดังกล่าวต่อไป) และเป็น "L" เมื่อรับข้อมูลแล้ว

- เมื่อ TA ส่งข้อมูลให้ TB จะทำให้ขา TxRDY เป็น "H" เพื่อขอข้อมูลไบท์ต่อไปจากซีพียู(ด้วยการอินเตอร์รัพท์หรือพูลลิ่ง)

- เมื่อซีพียูส่งข้อมูลให้ TA จะทำให้ TxRDY เป็น "L"

การทำงานจะเป็นวงรอบไปเช่นนี้จนกว่าซีพียูจะสั่งหยุดส่ง แต่ถ้าในระหว่างการส่งนี้ซีพียูส่งข้อมูลให้ไม่ทันหรือไม่ส่ง ถ้าเป็นซิงค์โหมด USART จะส่ง SYNC character ไปจนกว่าจะได้รับข้อมูล ถ้าเป็นอะซิงค์โหมดมันจะส่ง mark ทำให้ขา TxD เป็น "H" จนกว่าจะได้รับข้อมูลเช่นกัน ทั้ง 2 โหมดนี้เราสามาร SBRK ใน command byte ได้ ซึ่งจะทำให้ขา TxD เป็น "L"

การทำงานของภาครับ

เช่นกันข้อแตกต่างในการจัดเรียงข้อมูลและการซิงค์ของทั้งสองโหมดนี้ได้กล่าวแล้วในเรื่องโครงสร้างภายใน ในหัวข้อนี้จะกล่าวถึงการทำงานโดยส่วนรวม ซึ่งเหมือน กันทั้ง 2โหมด (มีข้อแตกต่างบ้าง) ภาครับต่างกันภาคส่งตรงภาครับทำงานอยู่ตลอดเวลา เพียงแต่ซีพียูไม่รับทราบ เมื่อซีพียูต้องการอ่านคำจากพอร์ทนี้จะต้องอีนาเบิ้ลขา RxRDY ก่อนด้วยการให้ RxEเป็น "1" ใน command byte ซึ่งจะมีขั้นตอนการทำงานดังนี้ (เมื่อกล่าว

ถึงขาให้หมายถึงแฟลกชื่อเดียวกันด้วย)

– ข้อมูลขาเข้าจะเข้าทางขา RxD ส่งต่อให้ RB ซึ่ง RB จะจัดเรียงข้อมูลจนครบแล้วจะส่งให้ RA ช่วงที่ส่งนี้จะทำให้ RxRDY เป็น "H" เพื่อให้ซีพียูรับข้อมูล (ด้วยการอินเตอร์รัพท์หรือพูลลิ่ง)

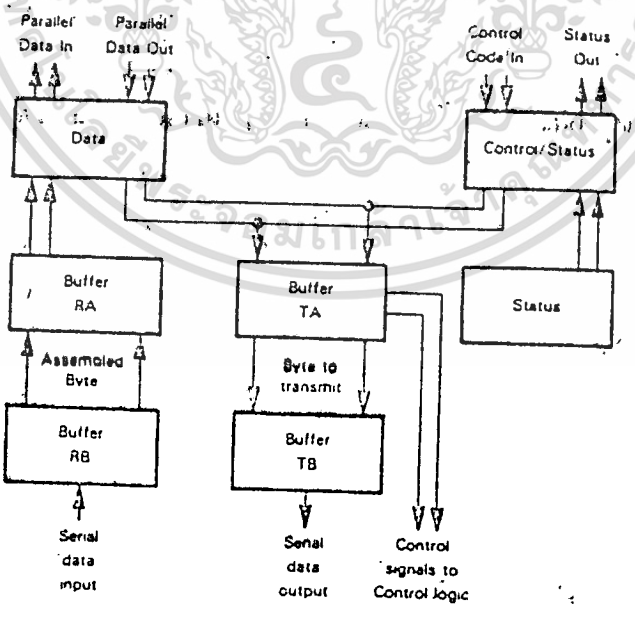
– เมื่อซีพียูรับข้อมูลจะทำให้ขา RxRDY เป็น "L" ตามเดิม

การทำงานจะเป็นวงรอบเช่นนี้จนกว่าซีพียูจะให้ RxE เป็น "0" ในการส่งค่าจาก RAให้ซีพียู หากซีพียูไม่รับ และถ้า RB มีข้อมูลใหม่จะถูกส่งไปทับข้อมูลเดิม แล้วให้ OE (overflow error)

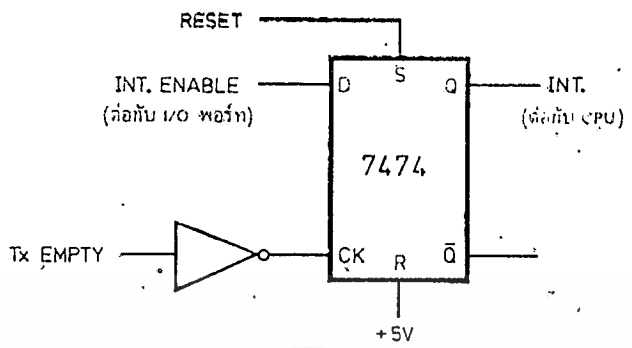
หากใช้ 8251

โดยที่ 8251 ไม่มี command control register เป็นของตนเอง ดังแสดงในรูปที่7 เพราะค่าเหล่านี้เมื่อได้รับจากซีพียูแล้วจะถูกส่งให้อุปกรณ์ต่างๆ ทันทีอินเทลออกแบบโดยประหยัดคือให้ใช้ register ร่วมกับ TA ผลที่ตามมาคือเมื่อเราจะให้ค่าในไบท์ดังกล่าว ต้องให้ในช่วงที่ TB ไม่ได้กำลังจะรับข้อมูลจาก TA เวลาที่เหมาะที่สุดคือเมื่อ TB พึ่งรับสัญญาณจาก TAซึ่งเราทราบโดยขา TxEMTY เปลี่ยนสถานะจาก "H" เป็น "L"

แต่ขานี้ไม่มีแฟลกในสเตตัสรีจิสเตอร์ จึงต้องดวงจรให้ทราบแน่ชัดว่าการอินเตอร์รัพท์มาจาก USART ตัวนี้แล้วจรวจในสเตตัสแฟลกว่าไม่มีขาอื่น



รูปที่ 7 วงจรจัดบัฟเฟอร์และรีจิสเตอร์ของ 8251



รูปที่ 8 วงจรจัดขา TxEMPTY ให้ขออินเตอร์รัพท์

ขออินเตอร์รัพท์ ก็จะเป็นการขออินเตอร์รัพท์จาก TxEMPTY แต่เราจะนำขานี้มาให้ขออินเตอร์รัพท์โดยตรงไม่ได้มันจะขอทุกครั้งที่ IB รับข้อมูลจาก IA จึงให้มันขอได้เมื่อเราต้องการจะส่งcommand/control byte เท่านั้น โดยจัดขาจากเอาท์พุทพอร์ทขาหนึ่งมาควบคุมดังแสดงในรูปที่ 8

เรื่องของ 8251 และ 9551 มีเพียงเท่านี้ และคิดว่า 8251 คงจะพัฒนาไปจากที่กล่าวถึง ในขั้นแรกนอกจาก 8251 แล้วจะต่อด้วย Z80-SIO ได้เตรียมไอซีไว้หมดแล้วแต่ไม่มีโอกาศทดลอง เมื่อทบทวนดูอีกครั้งเห็นว่าพอร์แบบที่ใช้น้อยมากหากไม่เกี่ยวกับข้อมูลระยะไกลเบอร์เดียวควรจะพอกับการทดลองใช้งาน

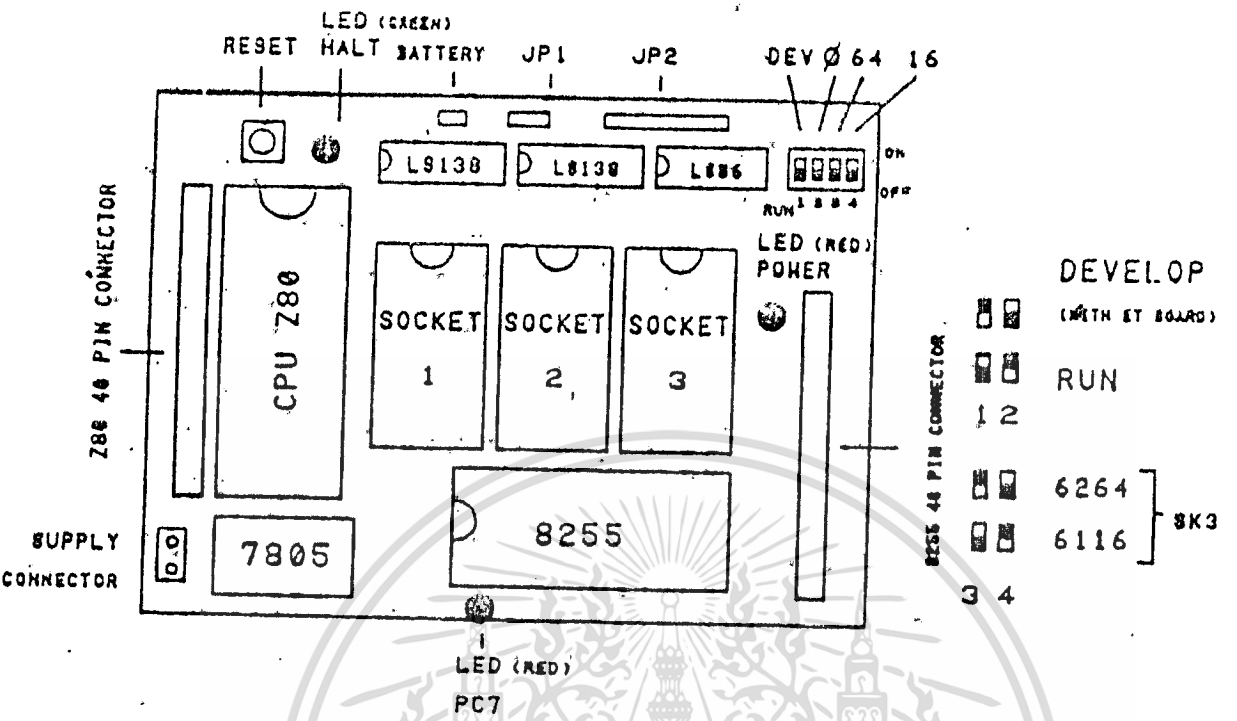
Z80

CP-A

CONTROL PACK ADVANCE

Reference Manual

Product By ETT



Socket 1 ใช้กับเบอร์ 2732,2764,6264

Socket 2 ใช้กับเบอร์ 6264 โดยจะสามารถ Backup ข้อมูลได้ ถ้าต่อ Battery

Socket 3 ใช้กับเบอร์ 6116,6264 โดยเลือกได้จาก Dip Switch ตัวที่ 3,4

JP 1 สำหรับเลือกสถานะของ Socket 1 โดยถ้าต่อกับ WR (62) จะใช้กับ 6264 และ 2732 แต่ถ้าต่อกับ Vcc (27) จะใช้กับ 2764

JP 2 สำหรับเลือกเบอร์ Port ของ 8255

Dip 1 สำหรับเลือกโหมดในการใช้งาน Dev. คือโหมดในขณะทำการพัฒนา Run คือโหมดสำหรับการใช้งาน

Dip 2 สำหรับตัดหรือต่อความถี่ Clock ปกติจะใช้ร่วมกับตัวที่ 1 คือถ้าขณะทำการพัฒนาจะ Off ไว้

Dip 3 เลือก Socket 3 ให้ใช้กับ 6264

Dip 4 เลือก Socket 4 ให้ใช้กับ 6116

(ตัวที่ 3,4 นี้จะ On ตัวใดตัวหนึ่งเท่านั้น)

ยุคของระบบควบคุมด้วยไมโครโปรเซสเซอร์กับ Z80 CP-A

ออะไร

อบอร์ดไมโครโปรเซสเซอร์ขนาดเล็ก ที่ประกอบด้วยระบบพื้นฐานในการทำงาน เช่น PU หน่วยความจำ รวมทั้งการจัด Decode ไว้อย่างเรียบร้อย และยังมี I/O ort สำหรับการต่อกับอุปกรณ์ Input Output ภายนอกให้อีก

าอะไรได้บ้าง

80 CP-A คือโครงสร้างทาง Hardware อย่างเดียว ซึ่งจะต้องมีการพัฒนาโปรแกรม พื่อให้มันทำงานตามที่ต้องการ โดยจะขึ้นอยู่กับลักษณะของงานนั้นๆที่นำไปใช้ โปร กรมนี่เองที่จะเป็นตัวกำหนดความสามารถของ Z80 CP-A ซึ่งทำให้มีความจำกัดน้อย ุก เพราะมันจะทำงานให้คุณตามแต่จินตนาการอันกว้างไกลของคุณ

หมาะกับใคร

าหรับผู้ที่จะทำโครงงานต่างๆทางด้านอิเลคทรอนิคส์ /คอมพิวเตอร์ และระบบควบคุม ลศาสตร์ ซึ่งจะช่วยลดขั้นตอนในการออกแบบ การทำเครื่องต้นแบบ รวมทั้งการ ลิต และเนื่องจากเป็นบอร์ดที่ออกแบบมาโดยเฉพาะ จึงทำให้มีเสถียรภาพในการทำ านสูง ในขณะที่เครื่องต้นแบบทั่วๆไปมักจะมีปัญหาเรื่องสัญญาณรบกวน เนื่องจากจุดต่อ างๆ

ากหรือไม่

ผู้ที่มีความเข้าใจทางด้านไมโครโปรเซสเซอร์อยู่แล้ว ก็สามารถทำได้ไม่ยากเลย ่กรณีผู้ที่เริ่มต้นใหม่ก็จะใช้เวลาในตอนต้นมากหน่อย อย่างไรก็ตาม ทีมงานอิทิทีพร้อม ะลนับสนุนการให้คำปรึกษาและวิชาการแก่กลุ่มลูกค้าเสมอ จึงทำให้มีความมั่นใจได้ใน รงงานที่จะพัฒนาขึ้น

ค่าแค่ไหน

งอยู่ งานบางอย่างก็สามารถใช้วงจรอิเลคทรอนิคส์ทำงานได้ โดยไม่จำเป็นต้องใช้ โครโปรเซสเซอร์ แต่อย่างไรก็ตามเมื่อเทียบกับความเร็วในการพัฒนาและราคาที่พอ มาะแล้ว วิธีนี้เป็นการลงทุนที่คุ้มค่าและให้ผลเป็นที่น่าพอใจ ที่สำคัญการทำโครง นด้านไมโครโปรเซสเซอร์จะมีการปรับปรุงแก้ไขเพิ่มเติมได้ง่าย โดยการกระทำทาง รแกรม ซึ่งจะเป็นเรื่องยากถ้าเป็นวงจรตายตัว จึงเหมาะเป็นอย่างยิ่งที่คุณจะฝาก รงงานชิ้นสำคัญของคุณไว้กับ Z80 CP-A

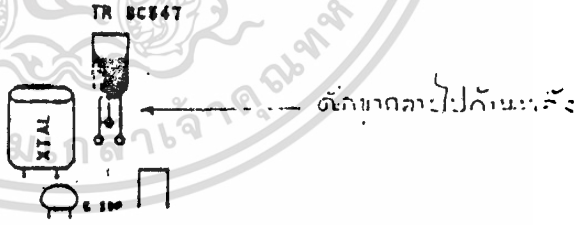
ในโหมด Dev นี้ ผู้ใช้จะเห็น Address ในตำแหน่ง 8000H A000H C000H ตามลำดับ และถ้านำ Ram 6264 มาเสียบที่ Socket 1 ก็จะสามารถทำโปรแกรมได้เลย โดยไม่การทำงานต่างๆทั้งหมดของอีทิบอร์ด การนำ Ram 6264 มาเสียบ จะต้องเปลี่ยน JP1 ให้อยู่ในตำแหน่ง 62 ด้วย ซึ่งจะต้องตัดลายปรั๊นที่ Default ไว้ที่ตำแหน่ง 27 ออก แล้วจึงต่อสายใหม่

การนำไปใช้งาน

- ทำการปรับ Dip switch ตัวที่ 1 ให้อยู่ในตำแหน่ง Run และตัวที่ 2 ให้อยู่ในตำแหน่ง On
- ใส่ตัว CPU Z80 ลงใน Socket ให้เรียบร้อย
- นำโปรแกรมที่พัฒนาเรียบร้อยแล้วอัดลงในตัว Eprom โดยใช้เครื่อง ET Burner หรือจะใช้เครื่องอื่นๆก็ได้ โดยโปรแกรมนี้จะต้องแปลงให้ทำงานที่ Address 0000H จากนั้นก็นำมาใส่ลงบนบอร์ดที่ Socket 1
- ถ้ามีการแก้ไขที่ JP1 ในขณะที่พัฒนา ก็ต้องเปลี่ยนกลับเป็นตำแหน่ง 27 เหมือนเดิม

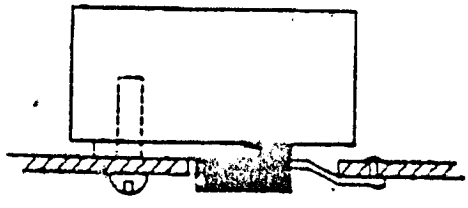
คำแนะนำในการประกอบ

ควรใส่อุปกรณ์ตามลำดับคือ ตัวต้านทาน, ไดโอด, ตัวเก็บประจุ, Socket ตัวเก็บประจุและไดโอดต้องใส่ให้ถูกขั้ว รวมทั้งทรานซิสเตอร์ต้องใส่ขาให้ถูก โดยให้ดูจากรูปดังนี้



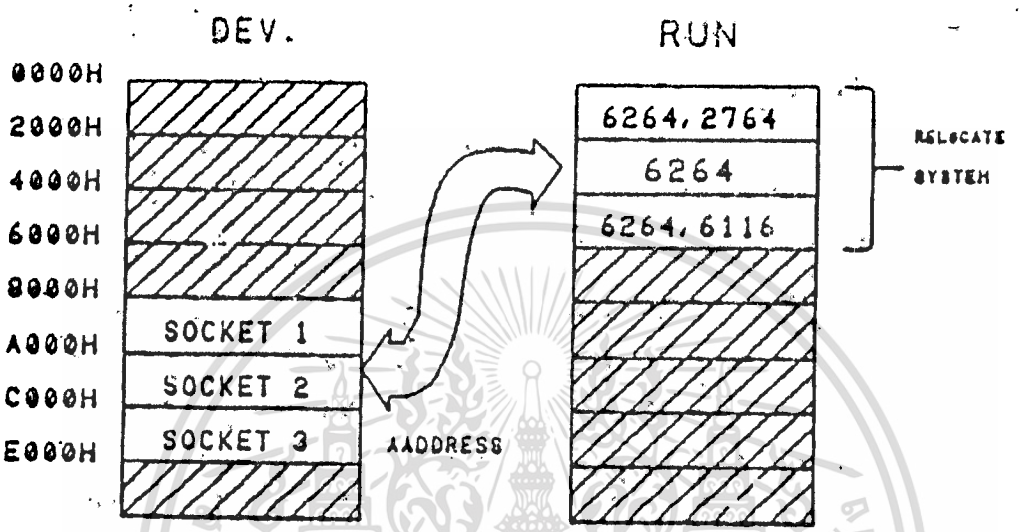
LED ขายาวคือ A ขาสั้นคือ K

การบัดกรีให้กระทำเฉพาะด้านล่างเพียงด้านเดียว ด้านบนไม่ต้องบัดกรีเพราะแผ่นปรั๊นเป็นแบบ Plate Through Holes รวมทั้งรูว่างๆก็ไม่ต้องบัดกรี การติดตั้งและบัดกรี Regulator IC ให้ดูดังรูป



เมื่อเรียบร้อยแล้วจึงใส่ IC ลงใน Socket โดยต้องระวังไม่ให้ผิดเบอร์ด้วย

รายละเอียดการจัดหน่วยความจำและ Port



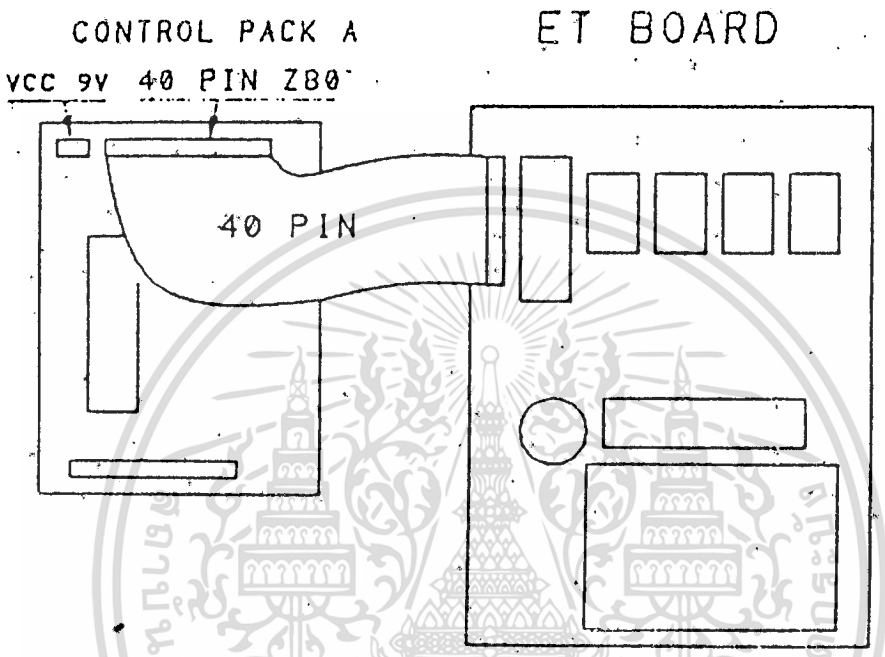
การจัดหน่วยความจำของ CP-A นี้จะอ้าง Address เริ่มต้นที่ 8000H เพื่อให้ทำโปรแกรมได้สะดวกจากเครื่องอีทีบอร์ด และเมื่อจะนำโปรแกรมไปใช้งาน CP-A มีระบบ Relocate เพื่อให้หน่วยความจำไปเริ่มต้นที่ 0000H ได้ (ในโหมด Run) เพื่อให้เป็น Address ที่ทำงานเมื่อเริ่มเปิดเครื่องหรือกด Reset

ระบบ Port จะทำการ Decode ไว้ทั้งหมด 8 ช่วง และจะสามารถเลือกเบอร์ได้ 7 ตำแหน่ง โดยการ Jump สายบนแผ่นปริ๊น ซึ่งมีเบอร์ Port ดังนี้

ตำแหน่ง	เบอร์ Port
1	00 - 1F (จริงๆใช้เพียง 00-03)
2	20 - 3F
3	60 - 7F
4	80 - 9F
5	A0 - BF
6	C0 - DF
7	E0 - FF

ส่วนเบอร์ Port 40 - 5F นั้น ไว้สำหรับการขยายได้อีกทางด้าน 8255 Connector

Z80 CP-A มีโหมดในการทำงาน 2 ลักษณะคือ การพัฒนาและการนำไปใช้งาน ในขั้นของการพัฒนานั้นจะทำได้โดยเสียบเข้ากับอีทีบอร์ดได้เลย หรือจะทำโดยใช้เครื่องอื่นๆก็ได้ แล้วจึงนำโปรแกรมมาทำงานโดยผ่าน Eprom หรือ Ram-pack ซึ่งผู้ใช้จะต้องเขียนโปรแกรมโดยอ้าง Address ตามลักษณะที่กำหนดไว้ ในการเสียบกับอีทีบอร์ดนั้นจะแสดงได้ดังภาพต่อไปนี้

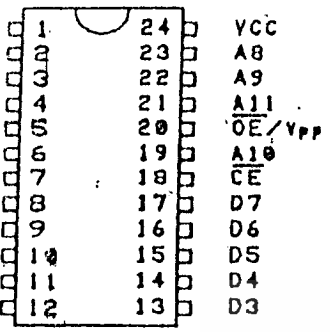


ซึ่งพอจะสรุปขั้นตอนได้ดังนี้

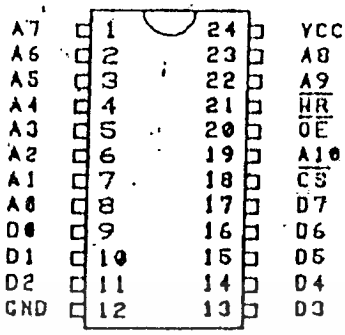
1. ทำการปรับ Dip switch ตัวที่ 1 ให้อยู่ในตำแหน่ง Dev. และตัวที่ 2 ให้อยู่ในตำแหน่ง Off ซึ่งเป็นการตัดความถี่ Clock ให้เหลือเพียง Clock จากอีทีบอร์ด
2. ผู้ใช้จะต้องเอา CPU Z80 ตัวใดตัวหนึ่งออกจากบอร์ด คือบนบอร์ด CP-A หรือบนอีทีบอร์ดก็ได้ เพื่อที่จะได้ไม่ซ้อนกัน
3. แหล่งจ่ายไฟ 5V ของทั้งสองบอร์ดจะต่อถึงกันหมด และเนื่องจากใช้กระแสมากขึ้น ผู้ใช้ควรจะใช้ Adapter ที่มีขนาดใหญ่พอสมควร (ของที่มากับเครื่องอีทีบอร์ดจะไม่พอ) และก็เสียบกับบอร์ดใดบอร์ดหนึ่งได้เลย ทั้งนี้ถ้ามีการต่อ Port ออกไปใช้งาน ก็ต้องดูด้วยว่ากระแสจ่ายได้เพียงพอหรือไม่ ถ้าไม่พอก็ควรแยกแหล่งจ่ายไฟออกเป็นอีกชุดหนึ่ง

รายละเอียดขา IC และ Connector

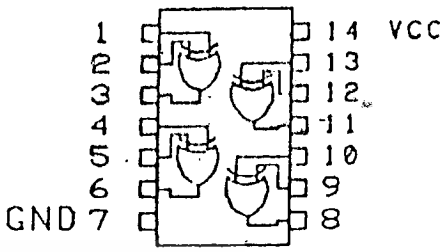
EPROM 2732



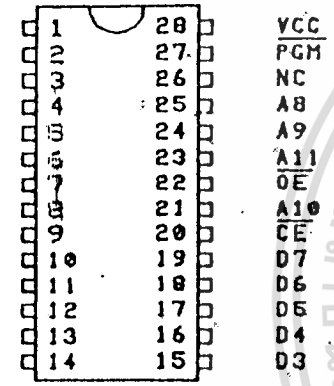
RAM 6116



TTL 74LS86

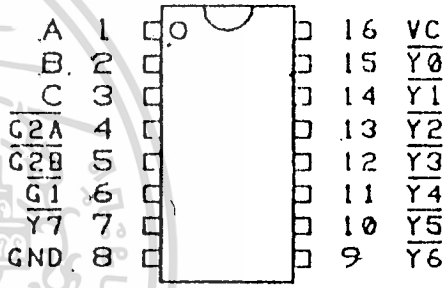


EPROM 2764

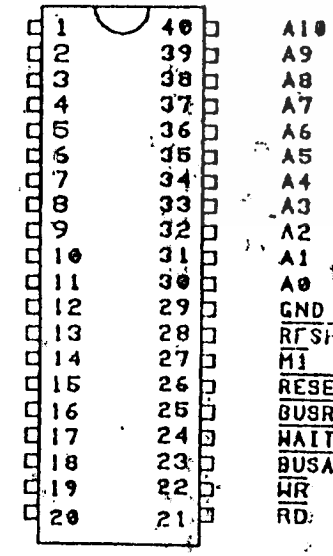


RAM 6264

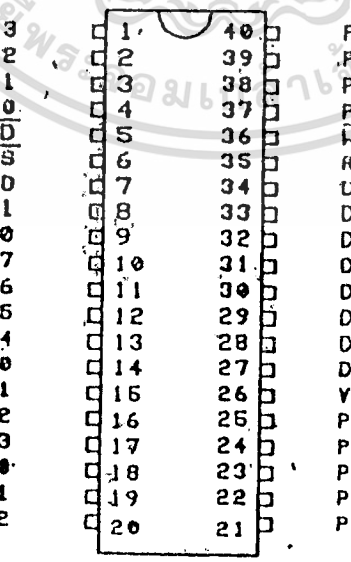
TTL 74LS138



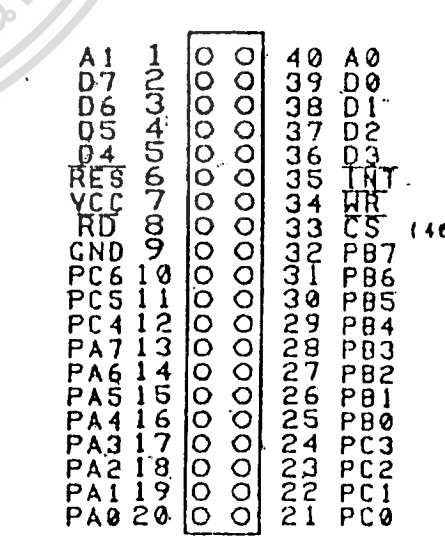
Z80 CPU

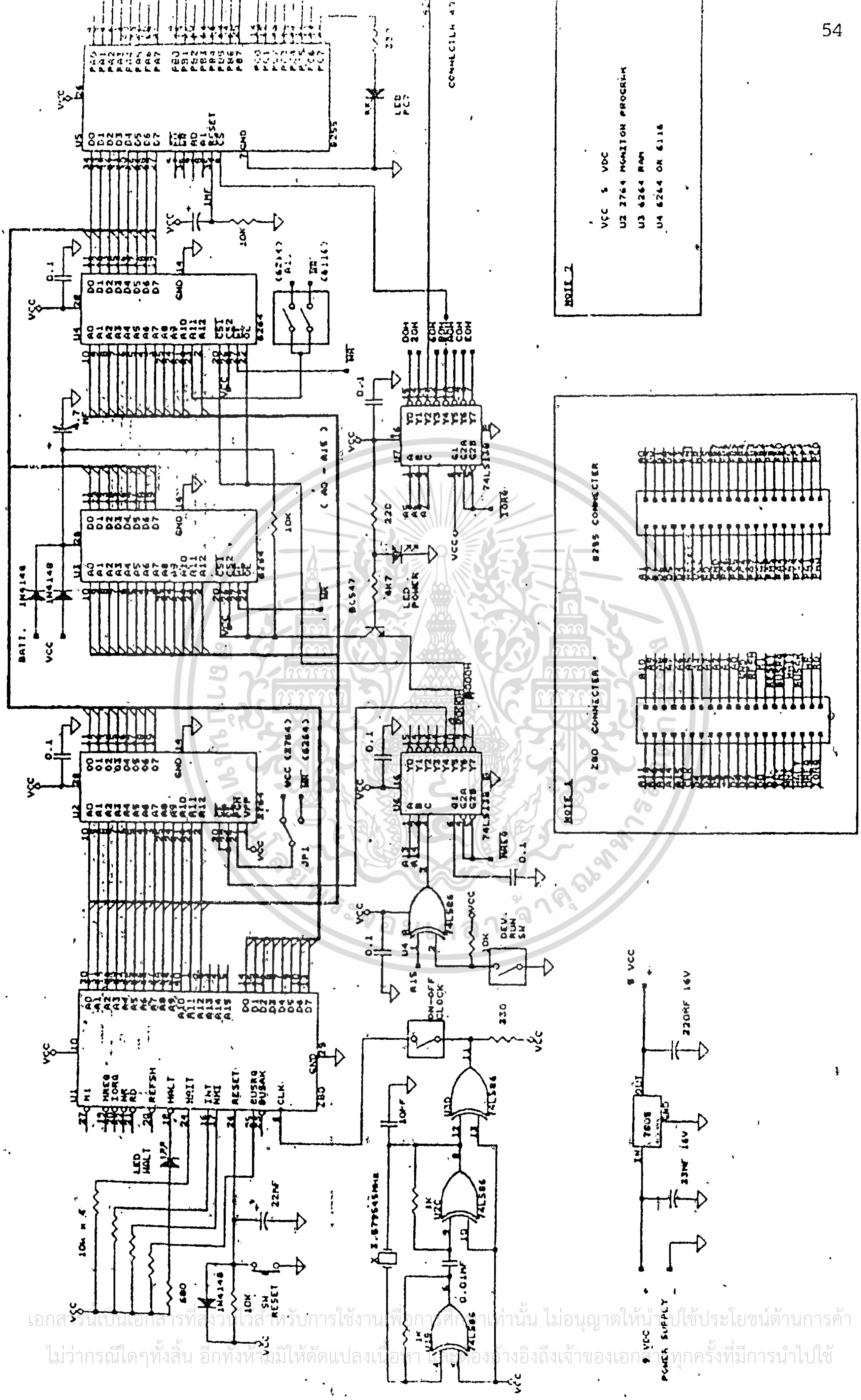


8255 PPI



8255 CONNECTOR





ification

CPU	Z-80A
Memory	6264/2764/2732 Socket (8 Kbyte at 8000H)
	6264 Ram (8 Kbyte at A000H) & Backup Circuit
	6264/6116 Socket (8 Kbyte at C000H)
Port	8255 I/O Port (8*3 bit)
LED	1 Power Red LED
	1 PC7 Port Red LED
	1 Halt Green LED
Diode	2 1N60
Transistor	1 BC547
TTL IC	3 74LS86 74LS138*2
Regulator IC	1 7805
Connector	1 40-Pin Expansion Header-Strip (Z80 Pin)
	1 40-Pin Peripheral Header-Strip (8255 Port)
	1 2-Pin For DC 9V Supply
Switch	4 Dip-Switch
	(Run/Develope, Clock-On, 6116, 6264 Select)
Clock Rate	3.579545 MHz
Power Supply	Comsumption 5V DC
	Main Input 9-12V DC
PCB Size	8.5 * 12 cm.

ภาคอุปกรณ์

IC

1 ตัว Z-80A CPU
1 ตัว 6264 Ram
2 ตัว 74LS138
1 ตัว 8255 I/O port
1 ตัว 74LS86

Socket

2 ตัว 40-Pins Socket
3 ตัว 16-Pins Socket
3 ตัว 28-Pins Socket

Resistor

8 ตัว 10 K
1 ตัว 4.7 K
2 ตัว 330
2 ตัว 1 K
1 ตัว 220
1 ตัว 680

ondensor

1 ตัว 220 uf ELE
1 ตัว 22 uf ELE
1 ตัว 1 uf TAN
1 ตัว 0.01 uf ไมล่าร์
1 ตัว 33 uf ELE
1 ตัว 4.7 uf TAN
6 ตัว 0.1 uf TAN
1 ตัว 10 pf CER

าเตล็ด

3 ตัว LED (แดง 2 ตัว , เขียว 1 ตัว)
2 ตัว 1N60
1 ตัว 7805
2 ตัว Connector 40 Pin
1 ตัว Dip Switch (4*)
1 ตัว X'TAL 3.579545 MHz
1 ตัว 1N4148
1 ตัว Head Sink พร้อมน็อต
1 ชุด ขั้วต่อ Supply
1 ตัว Reset Switch
1 ตัว BC 547

1 แผ่น CP-A PCB
1 เล่ม ชีทรายละเอียดการใช้งาน

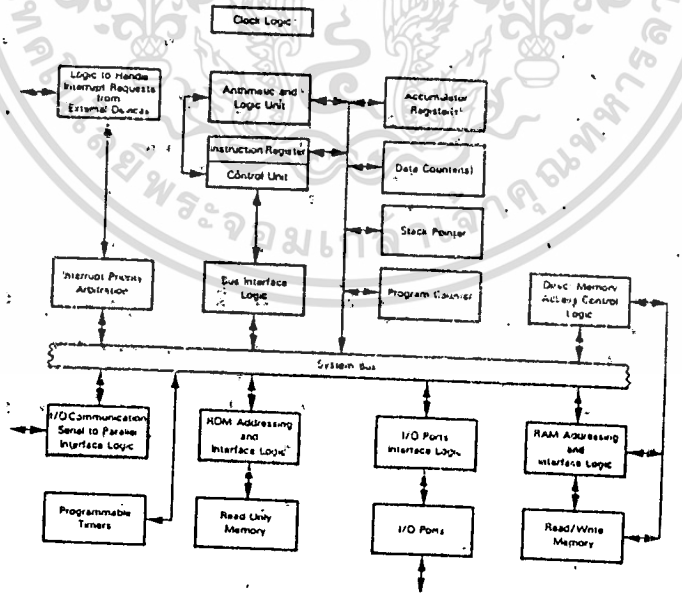
Z80

โดยทั่วไปโครงสร้างพื้นฐานของคอมพิวเตอร์ จะประกอบไปด้วย

- หน่วยควบคุม (CONTROL UNIT)
- หน่วยความจำ (MEMORY UNIT)
- หน่วยคำนวณ (ARITHMETIC UNIT)
- หน่วยรับและส่งสัญญาณ (I/O UNIT)

ด้วยการพัฒนาทางด้านเทคโนโลยีที่ทันสมัย ทำให้ชิ้นส่วนต่างๆที่ประกอบขึ้นเป็นหน่วยย่อยๆภายในเครื่องคอมพิวเตอร์มีขนาดเล็กลง แต่ประสิทธิภาพกลับสูงขึ้นและราคาก็ถูกลงอย่างมาก . ทำให้ความต้องการที่จะนำเอาเครื่องคอมพิวเตอร์มาใช้ในชีวิตประจำวันมีมากขึ้นเป็นลำดับ

ในปัจจุบันนี้ เราสามารถนำเอาวงจรอิเลคทรอนิคส์ที่ยุ่งยากและซับซ้อนมาบรรจุลงบนแผ่นวงจรเดียวที่มีขนาดเล็กมากซึ่งเรียกว่า LSI (LAST SCALE INTEGRATED CIRCUIT) และบรรจุอยู่ในตัวถัง ซึ่งต่อขาออกมาเพื่อใช้ในการติดต่อกับวงจรภายนอกสิ่งนี้เรียกว่า " ชิพ (CHIP)" องค์ประกอบย่อยๆ ในไมโครคอมพิวเตอร์ทั้งหมด จะประกอบขึ้นจากชิพเหล่านี้ เช่น หน่วยความจำ ประเภท ROM (READ ONLY MEMORY),RAM (RANDOM ACCESS MEMORY) อุปกรณ์สนับสนุน (CHIP SUPPORT) ต่างๆ และสิ่งที่เป็นหัวใจของระบบไมโครคอมพิวเตอร์ คือ หน่วยประมวลผลกลาง หรือ CPU (CENTRAL PROCESSING UNIT) ซึ่งภายในประกอบไปด้วยส่วนต่างๆ ดังบล็อกไดอะแกรมรูป



รูปที่ 1 แสดงบล็อคไดอะแกรมของ Z80

ซึ่งแต่ละบล็อกมีลักษณะการทำงานดังต่อไปนี้ คือ

1. ARITHMETICS LOGIC UNIT (ALU) เป็นหน่วยที่ทำหน้าที่ในการคำนวณฟังก์ชันพื้นฐานทางคณิตศาสตร์ และการกระทำฟังก์ชันทางลอจิก เช่น AND และ OR

2.CONTROL UNIT เป็นหน่วยที่ทำหน้าที่ในการส่งสัญญาณไปควบคุมอุปกรณ์ต่างๆที่ต่อเชื่อมกับ CPU ให้ทำงานร่วมกันได้อย่างถูกต้อง

3.DATA BUS เป็นบัสสองทิศทาง BI-DIRECTIONAL) ที่ใช้ในการส่งผ่านข้อมูลระหว่าง CPU กับอุปกรณ์อื่นๆภายในระบบ จำนวนเส้นของบัสข้อมูล (DATA BUS) จะขึ้นอยู่กับชนิดของ CPU เช่นในกรณีของ Z80 CPU จะส่งผ่านข้อมมูลทีละ 8 บิท ดังนั้นจะจำนวนเส้นของบัสข้อมูล 8 เส้น

4.CONTROL BUS หรือ บัสควบคุม เป็นบัสทางเดียว (UNI - DIRECTIONAL BUS) ที่ใช้ในการส่งผ่านสัญญาณควบคุมให้กับอุปกรณ์ต่างๆ ในระบบ

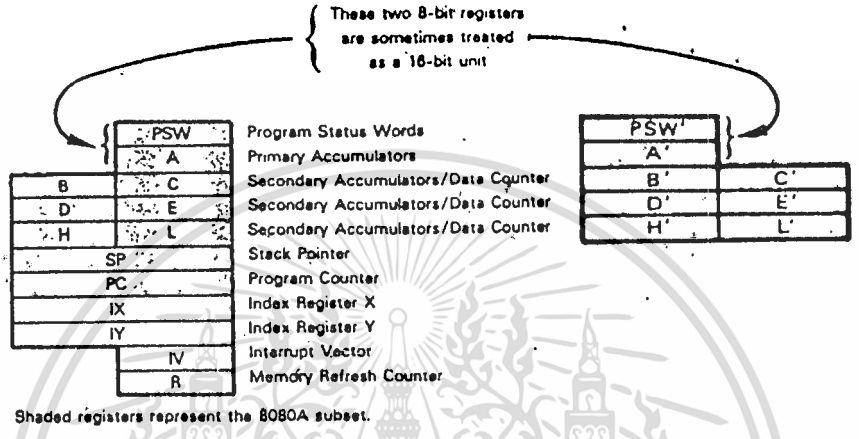
5.ADDRESS BUS เป็นบัสทางเดียว ใช้ส่งผ่านค่าแอดเดรสจาก CPU ออกไปยังหน่วยความจำเพื่อระบุตำแหน่งที่ต้องการรับหรือส่งข้อมูล หรือใช้ระบุตำแหน่งของมอร์ทI/O (INPUT/OUTPUT PORT) ที่ CPU ต้องการติดต่อด้วย.

ต่อไปนี้เราจะกล่าวถึงรายละเอียดของ Z80 CPU ซึ่งเป็นไมโครโปรเซสเซอร์ขนาด 8 บิทที่นิยมใช้อย่างแพร่หลายในปัจจุบัน

Z80 ไมโครโปรเซสเซอร์เป็น CPU ที่ผลิตจากบริษัท ZILOG INC. โดยกลุ่มวิศวกรชุดเดียวกับที่ผลิต 8080 CPU ของบริษัท INTIEL COORPERATION.,Z80 CPU ได้รับการพัฒนาให้มีข้อดีเหนือกว่า 8080 เช่น มีชุดคำสั่งมากถึง 158 คำสั่งโดยรวมชุดคำสั่งเดิมของ 8080 ไว้ 80 คำสั่ง นอกจากนี้ Z80 ยังมีรีจิสเตอร์มากกว่าใน 8080 ถึง 12ตัว และ 8080 เพียงตัวเดียวก็ยังไม่สามารถที่จะนำไปใช้งานได้ต้องต่อกับอุปกรณ์สนับสนุนอีก 2 ตัวคือ CLOCK GENERATOR CHIP,SYSTEM CONTROLER CHIP, รวมกันเรียกว่าTHREE CHIP PROCESSOR แต่ใน Z80 CPU ได้รวมเอาลักษณะพื้นฐานเหล่านี้ไว้ในชิพเดียวกัน และเพิ่มประสิทธิภาพทาง HARDWARE; SOFTWARE และการ INTERFACE ให้สูงขึ้น Z80 ไมโครโปรเซสเซอร์เพียงชิ้นเดียวไม่สามารถทำงานเป็นระบบคอมพิวเตอร์ได้ ต้องอาศัยอุปกรณ์อื่นๆ อีก 2 ส่วนคือ หน่วยความจำ (MEMORY) และหน่วยรับส่งข้อมูลเข้าออก (I/O DEVICE) ซึ่งในการทำงานตามคำสั่งจากโปรแกรมที่ป้อนเข้ามา Z80 CPU จะต้องทำการโอนย้ายคำสั่งหรือข้อมูลระหว่างหน่วยความจำ กับรีจิสเต่อร์ (REGISTER) ก่อนอื่นเราจะกล่าวถึงรีจิสเตอร์ภายในของ Z80 เสียก่อน

รีจิสเตอร์ต่างๆ ใน Z80 CPU.

Z80 CPU จะประกอบไปด้วยรีจิสเตอร์ถึง 22 ตัว รีจิสเตอร์เหล่านี้จะแบ่งได้2 กลุ่มคือ รีจิสเตอร์ที่ทำหน้าที่ทั่วๆ ไป และรีจิสเตอร์ที่ทำหน้าที่เฉพาะงาน



รูปที่ 2 แสดงรีจีสเตอร์ต่างๆภายใน Z80

1.**รีจิสเตอร์ที่ทำหน้าที่ทั่วๆไป** แบ่งเป็นรีจิสเตอร์หลัก ได้แก่ A,B,C,D,E,H, และ L มีความจุขนาด 8 บิท รีจิสเตอร์เหล่านี้ใช้เก็บข้อมูลชั่วคราว นอกจากนี้ยังสามารถรับข้อมูลจากหน่วยความจำหรืออาจจะทำการย้ายข้อมูลไปเก็บไว้ในหน่วยความจำก็ได้ และรีจิสเตอร์สำรอง ได้แก่ A:,B,:,C,:,D:,E:,H: และ L: ซึ่งเป็นรีจิสเตอร์ที่ทำหน้าที่เก็บข้อมูลที่มาจากรีจิสเตอร์หลักในกรณีที่ต้องใช้รีจิสเตอร์หลักในการทำงานอย่างอื่นก่อน ดังนั้นรีจิสเตอร์กลุ่มนี้จึงไม่สามารถกระทำขบวนการทางคณิตศาสตร์และลอจิกได้

รีจิสเตอร์ A เรียกว่า แอคคิวมูเลเตอร์ (ACCUMULATOR) ทำหน้าที่เก็บข้อมูลชั่วคราวที่ได้จากการทำขบวนการทางคณิตศาสตร์ เช่น บวกหรือลบข้อมูล 2 จำนวน ผลลัพท์ที่ได้จะเก็บไว้ในรีจิสเตอร์ A นี้ นอกจากนี้ ในการปฏิบัติตามคำสั่งที่ใช้กับข้อมูลขนาด16 บิท Z80 จะนำเอารีจิสเตอร์แฟลก "F" (FLAG REGISTER) มาใช้ร่วมกับรีจิสเตอร์A เรียกว่า คู่รีจิสเตอร์ AF ซึ่งมีขนาด 16 บิท นอกจากนี้ยังมีคู่รีจิสเตอร์ 16 บิท อื่นๆอีกคือ BC,DE, และ HL.

2.รีจิสเตอร์ที่ใช้งานเฉพาะอย่าง ได้แก่ รีจิสเตอร์ I,R,IX,IY,SP และ PCซึ่งทำหน้าที่ต่างๆดังนี้

รีจิสเตอร์ I (INTERRUPT PAGE ADDRESS REGISTER) เมื่อมีการอินเทอร์รัพเกิดขึ้นจำเป็นต้องบอกตำแหน่งของหน่วยความจำที่เก็บโปรแกรมตอบสนองการอินเทอร์รัพท์ รีจิสเตอร์ I จะทำหน้าที่เก็บค่า 8 บิทบนของตำแหน่งข้อมูลในหน่วยความจำ ส่วนค่า8 บิทล่างจะป้อนจากภายนอกให้แก่ CPU ค่าทั้งสองจะประกอบกันเป็นค่าแอดเดรสที่ระบุตำแหน่งของโปรแกรมการตอบสนองการอินเตอร์รัพท์.

รีจิสเตอร์ R (MEMORY REFRESH REGISTER) เป็นรีจิสเตอร์ขนาด 7 บิทที่ถูกใช้ในการรีเฟรช (REFRESH) DYNAMIC RAM และค่ารีจิสเตอร์ R จะเพิ่มขึ้นเองโดยอัตโนมัติ ในทุกๆครั้งที่การเฟทช์คำสั่งจากหน่วยความจำ RAM

รีจิสเตอร์ IX และ IY (INDEX REGISTER) เป็นรีจิสเตอร์ที่มีขนาด 16 บิทมี**ประโยชน์**ใช้บ่งบอกตำแหน่งในหน่วยความจำแบบ INDEX ADDRESSING MODE โดยจะกำหนดให้ค่าใน INDEX REGISTER เป็นค่าอ้างอิง แล้วใช้คำสั่งบ่งบอกว่าตำแหน่งของข้อมูลที่ต้องการอยู่ห่างจากค่าอ้างอิงนี้เท่าใด โดยจะบอกค่าระหว่างในรูปของ TWO COMPLEMENT.

รีจิสเตอร์ SP (STACK POINIER) มีขนาด 16 บิท ในหน่วจความจำชนิดRAM จะมีส่วนหนึ่งที่ถูกกำหนดให้เป็นที่เก็บข้อมูลชั่วคราว ส่วนนี้เรียกว่าสแตก (STACK)ซึ่งมีลักษณะการเก็บข้อมูลแบบ LIFO (LAST IN FIRST OUT) เราสามารถจะเก็บข้อมูลลงบนสแตกโดยใช้คำสั่ง PUSH และเมื่อต้องการดึงข้อมูลออกจากสแตกต้องใช้คำสั่ง POP.

รีจิสเตอร์ PC (PROGRAM COUNTER) เป็นรีจิสเตอร์ขนาด 16 บิทที่ใช้ในการเก็บตำแหน่งของหน่วยความจำที่ CPU จะเฟทช์ (FETCH) คำสั่งหลังจากที่เฟทช์คำสั่งเรียบร้อยแล้ว ค่าในรีจิสเตอร์ PC จะเพิ่มขึ้น และจะชี้ไปยังตำแหน่งของคำสั่งถัดไป เราจะสามารถเปลี่ยนแปลงค่าใน PC ได้โดยใช้คำสั่ง CALL หรือ JUMP.

รีจิสเตอร์ F (FLAG REGISTER) ประกอบด้วย

SIGN FLAG (S) : แฟลกเครื่องหมาย

ZERO FLAG (Z) : แฟลกศูนย์.

HALF CARRY FLAG (H): แฟลกทดครึ่ง

PARITY/OVERFLOW FLAG (P/V) : แฟลกพารีตีหรือโอเวร์โฟลว์

SUBTRACT FLAG (N) : แฟลกลบ

CARRY FLAG (C) : แฟลกตัวทัด

ผู้ผลิต Z80 ได้นำเอาแฟลกเหล่านี้มาประกอบร่วมกับ บิทว่าง (X; ไม่มีความหมาย

) อีก 2 บิทเพื่อทำเป็นรีจิสเตอร์ขนาด 8 บิท สำหรับรายละเอียดของแฟลกเหล่านี้จะ ไม่ขอกล่าวถึง.

รายละเอียดของขา Z80 (Z80 PIN OUTS).

A0-A15 (ADDRESS BUS): เป็นขาสัญญาณเอาท์พุทแบบ TRI-STATE ใช้บ่งบอกตำแหน่งหน่วยความจำได้ถึง 2^{16}=65536 ตำแหน่ง A0-A7 จะแสดงตำแหน่งของพอร์ทที่ Z80 ต้องการติดต่อด้วย นอกจากนี้ขา A0-A6 จะให้ค่ารีเฟรชแอดเดรสออกมาขณะที่Z80 ให้สัญญาณรีเฟรช.

D0-D7 (DATA BUS): เป็นขาสัญญาณอินพุท/เอาท์พุท TRI-STATE แบบสองทิศทางซึ่งเป็นทางผ่านของข้อมูลระหว่าง Z80 กับหน่วยความจำและอุปกรณ์ I/O.

M1 (MACHINE CYCLE ONE) : เป็นขาเอาท์พุทแอคทีฟที่ลอจิก "0" ขา M1 นี้จะแอคทีฟขณะที่ Z80 ทำการเฟทช์ออฟโคดของคำสั่ง ในกรณีที่คำสั่งที่จะเฟทช์เข้ามานั้นมีขนาด 2 ไบท์ M1 จะแอคทีฟในทุกๆ ไซเคิลการเฟทช์แต่ละไบท์.

MREQ (MEMORY REQUEST) : เป็นสายเอาท์พุทแบบ TRI-STATE แอคทีฟที่ลอจิก "0" เพื่อเป็นการบ่งบอกว่า Z80 กำลังกระทำการติดต่อกับหน่วยความจำ.

IORQ (INPUT/OUTPUT REQUEST) : เป็นสายเอาท์พุทแบบ TRI-STATE จะแอคทีฟที่ลอจิก "0" เพื่อเป็นการบ่งบอกว่า Z80 กำลังทำการติดต่อกับอุปกรณ์ I/Oและเมื่อ IORQ และ M1 แอคทีฟทั้งคู่จะเป็นบ่งบอกการตอบรับการอินเทอร์รัพท์(INTERRUPT ACKNOWLEDGE).

RD (MEMORY READ): เป็นขาเอาท์พุท TRI-STATE จะแอคทีฟที่ลอจิก "0" เมื่อ Z80 ต้องการอ่านข้อมูลจากหน่วยความจำหรืออุปกรณ์ I/O และ Z80 จะรับข้อมูลจากบัสข้อมูลเข้าไปเมื่อสัญญาณนี้เปลี่ยนระดับลอจิกจาก "0" เป็น "1"

WR (MEMORY WRITE) : เป็นขาเอาท์พุทแบบ TRI -STATE จะแอคทีฟที่ลอจิก "0" เมื่อ Z80 ต้องการส่งข้อมูลออกไปให้หน่วยความจำหรืออุปกรณ์ I/O.

RFSH (REFRESH): เป็นขาเอาท์พุท จะแอคทีฟเมื่อ 7 บิทล่าง (A0-A6) ของบัสแอดเดรสให้ค่ารีเฟรชออกมา.

HALT (HALT STATE): เป็นขาเอาท์พุท จะแอคทีฟที่ลอจิก "0" เมื่อ Z80 อยู่ในสภาวะของการ HALT คือ CPU จะทำคำสั่ง NOP (NO OPERATION) เพื่อให้เกิดการรีเฟรชได้และ Z80 จะหลุดพ้นจากสภาวะการ HALT เมื่อได้รับการรีเซ็ทหรือถูกอินเทอร์รัพท์

WAIT: เป็นขาอินพุทแอคทีฟที่ลอจิก "0" และจะมีการตรวจสอบสัญญาณนี้ที่ขอบ

ขาลงของคล็อกลูกที่ 2 ของทุกๆ MACHINE CYCLE เมื่อมีการตรวจพบว่าเขาอินพุทนี้แอคทีพนี้แอคทีพจะมีการแทรก WAIT STATE ให้กับแต่ละ MACHINE CYCLE เพื่อเป็นการรอให้อุปกรณ์ภายนอกทำงานให้ทันกับการทำงานของ Z80 และ Z80 จะแทรก WAIT STATE จนกว่าจะมีการตรวจสอบพบว่าขา WAIT จะมีลอจิกเป็น "1".

INT (INTERRUPT REQUEST): เป็นขาอินพุทแอคทีพที่ลอจิก "0" Z80 จะตรวจสอบระดับสัญญาณที่ขานี้ทุกๆ การสิ้นสุดของ INSTRUCTION CYCLE (LAST STATE).

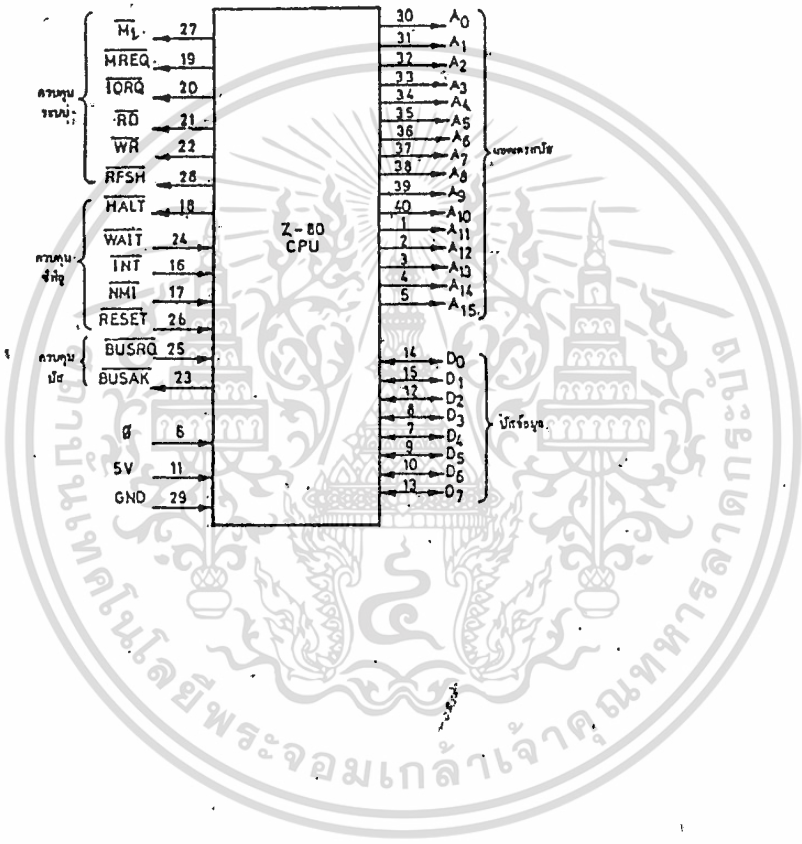
NMI (NON MASKABLE INTERRUPT): เป็นขาอินพุทแอคทีพที่ลอจิก "0" สัญญาณ NON MASKABLE INTERRUPT เป็นสัญญาณที่มีระดับความสำคัญในการขออินเทอร์รัพท์สูงกว่าสัญญาณ INTERRUPT REQUEST Z80 จะตอบรับการอินเทอร์รัพท์ชนิดนี้เสมอโดยที่ไม่สามารถ DISABLE ได้ด้วย SOFTWARE.

RESET : เป็นขาอินพุทแอคทีพที่ลอจิก "0" สัญญาณนี้จะทำการ INITIALIZE CPU โดยทำการรีเซ็ท INTERRUPT FLIP-FLOP และเซ็ทค่าในโปรแกรมเคาน์เตอร์ (PROGRAM COUNTER) ให้เป็น 0000H และในสภาวะการรีเซ็ทนี้ บัสแอดเดรสและบัสข้อมูลจะอยู่ในสภาวะ HIGH IMPEDANCE และสัญญาณควบคุมต่างๆ จะอยู่ในสภาวะ INACTIVE.

BUSRQ (BUS REQUEST): เป็นขาอินพุทแอคทีพที่ลอจิก "0" สัญญาณ BUS REQUEST เป็นสัญญาณที่มีลำดับความสำคัญสูงกว่าสัญญาณ NON MASKABLE INTERRUPTและมีการตรวจสอบสัญญาณนี้ทุกๆ การสิ้นสุดของ MACHINE CYCLE อุปกรณ์ภายนอกจะให้สัญญาณนี้แก่ Z80 เมื่อต้องการใช้บัสข้อมูลและบัสแอดเดรสโดยเปรียบเสมือนว่าเป็นการถอด Z80 ออกจากระบบบัส.

BUSAK (BUS ACKNOWLEDGE): เป็นขาเอาท์พุทแอคทีพที่ลอจิก "0" ขานี้จะแอคทีพเมื่อ Z80 ตอบสนองการต่อสัญญาณ BUS REQUEST และจะทำให้บัสข้อมูล บัสควบคุม และบัสแอดเดรสมีสภาวะเป็น HIGH IMPEDANCE ซึ่งทำให้อุปกรณ์ภายนอกใช้บัสเหล่านี้ได้โดยไม่มีผลต่อ CPU.

จากที่กล่าวมาทั้งหมดในบทนี้รายเป็นละเอียดที่ควรทราบในการศึกษาและนำZ80 ไปใช้งานร่วมกับอุปกรณ์สนับสนุนอื่นๆ (CHIP SUPPORT) ในบทต่อๆไปเราจะอธิบายรายละเอียดของอุปกรณ์ซึ่งจะมาประกอบเป็นระบบไมโครคอมพิวเตอร์.



รูปที่ 3 ลักษณะของขาไอซี Z80

8253

PROGRAMABLE TIMER

บล๊อกไดอะแกรมของ 8233 PROGRAMMABLE TIMER.

เมื่อพิจารณารีจิสเตอร์ภายในและโหมดของการทำงานของชิพนี้ จะเห็นได้ว่ามีบล๊อกของไทม์เมอร์ที่เป็นอิสระต่อกันอยู่ 3 ชุด ลักษณะการโปรแกรมเคาน์เตอร์ของไทม์เมอร์ทั้ง 3 ชุดนี้มีลักษณะเหมือนกัน

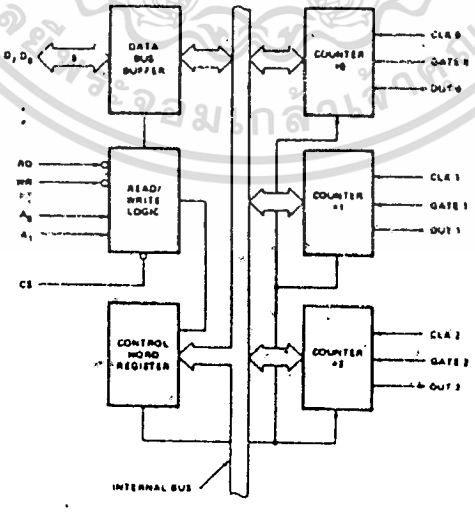
รูปที่ 1 แสดงบล๊อกของ DATA BUS BUFFER ซึ่งบล๊อกนี้จะเป็นบัฟเฟอร์ให้กับข้อมูลที่อยู่บนบัสข้อมูลที่เข้าและออกจากไมโครโปรเซสเซอร์กับรีจิสเตอร์ภายในของ8253 สำหรับบล๊อก READ/WRITE LOGIC บล๊อกนี้จะเป็นส่วนที่ใช้สำหรับควบคุมการอ่านและการเขียนของรีจิสเตอร์ของเคาน์เตอร์ และบล๊อกสุดท้ายเป็นบล๊อก CONTROL WORD REGISTER ซึ่งเป็นที่เก็บข้อความที่ถูกโปรแกรมเข้าไปโดยระบบไมโครโปรเซสเซอร์ ที่จริงแล้วรีจิสเตอร์ในบล๊อกนี้เป็นตัวกำหนดการทำงานของชิพนี้ว่าจะเป็นอย่างไร และแสดงการจัดขาของ 8253

PROGRAMMABLE INTERVAL TIMER

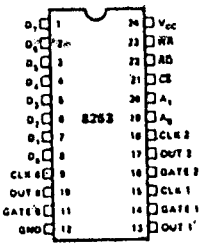
- MCS-85™ Compatible 8253-5
- 3 Independent 16-Bit Counters
- DC to 2 MHz
- Programmable Counter Modes
- Count Binary or BCD
- Single +5V Supply
- 24-Pin Dual In-Line Package

The Intel® 8253 is a programmable counter/timer chip designed for use as an Intel microcomputer peripheral. It uses nMOS technology with a single +5V supply and is packaged in a 24 pin plastic DIP.

It is organized as 3 independent 16-bit counters each with a count rate of up to 2 MHz. All modes of operation are software programmable.



รูปที่ 1 บล๊อกไดอะแกรมของ 8253



รูปที่ 2 แสดงการจัดเรียงขาของ 8253

สายสัญญาณ CLOCK ,GATE และ OUT ของเคาน์เตอร์.

เคาน์เตอร์แต่ละตัวจะมีสามสัญญาณต่อกับแต่ละบล็อกอยู่ 3 เส้น โดยสายสัญญาณที่มีชื่อว่า CLOCK และ GATE ใช้เป็นอินพุทส่วน OUT ใช้เป็นเอาท์พุท หน้าที่ในการทำงานของสายเหล่านี้เปลี่ยนแปลงได้ขึ้นกับว่า อุปกรณ์เหล่านี้ถูกกำหนดหน้าที่การทำงานเบื้องต้นไว้อย่างไรหรือถูกโปรแกรมมาอย่างไร และที่จะกล่าวต่อไปนี้ เป็นข้อกำหนดทั่วๆ ไปของสายสัญญาณ CLOCK,GATE และ OUT ของเคาน์เตอร์.

CLOCK : เป็นอินพุทที่ใช้ป้อนสัญญาณคล็อกให้แก่เคาน์เตอร์ ซึ่งเคาน์เตอร์ในที่นี้มีขนาด 16 บิท ความถี่ของสัญญาณคล็อกที่มากที่สุดที่ป้อนให้แก่เคาน์เตอร์เป็น 2.6 MHzและความถี่ของคล็อกที่น้อยที่สุดเป็น 0 Hz (DC) หรือ STATICOPERATION.

GATE : เป็นสายสัญญาณอินพุทที่ทำตัวเสมือน GATE ที่จะยอมหรือไม่ยอมให้สัญญาณคล็อกผ่านเข้าไม่ยังเคาน์เตอร์ และ GATE สามารถใช้เป็นสายสัญญาณที่ป้อนพัลส์(PULSE) เพื่อกระตุ้นให้เคาน์เตอร์เริ่มนับ ซึ่งขึ้นอยู่กับโหมดที่โปรแกรมให้กับเคาน์เตอร์.

OUT : เป็นสายสัญญาณเอาท์พุทของเคาน์เตอร์ ซึ่งการทำงานขึ้นอยู่กับการโปรแกรม.

รีจิสเตอร์ภายในของ 8253

ที่ปรากฏในรูป 3 นั้นเป็นรีจิสเตอร์ภายในของ 8353 ในขั้นต้นนี้จะพิจารณา MODE WORD REGISTER ก่อน รีจิสเตอร์นี้เป็นตัวกำหนดการทำงานทั้งหมดของ 8253จากที่ทราบมาแล้วว่าการทำงานของแต่ละเคาน์เตอร์ใน 8253 เป็นอิสระต่อกันอย่างสมบรูณ์ทำให้สามารถโปรแกรมการทำงานในเคาน์เตอร์แต่ละตัวได้ถึงในภายหลัง ต่อไปนี้เราจะมาพิจารณารีจิสเตอร์ภายในทั้ง 4 ในรูป 3

	RD	WR	A0	A1	
COUNTER 0	1	0	0	0	LOAD COUNTER 0
	0	1	0	0	READ COUNTER 0
COUNTER 1	1	0	0	1	LOAD COUNTER 1
	0	1	0	1	READ COUNTER 1
COUNTER 2	1	0	1	0	LOAD COUNTER 2
	0	1	1	0	READ COUNTER 2
MODE WORD OR	1	0	1	1	WRITE MODE WORD
CONTROL WORD	0	1	1	1	NO-OPERATION

รูปที่ 3 แสดงลอจิกของรีจีสเตอร์.

CONTROL WORD REGISTIER: เป็นรีจิสเตอร์ที่ใช้ควบคุมโหมดการทำงาน และใช้เลือกวิธีการนับของเคาน์เตอร์ว่าจะให้นับแบบไบนารี หรือ BCD (BINARY CODED DECIMAL) ก่อนที่จะใช้งานจะต้องโปรแกรมข้อมูลให้กับรีจิสเตอร์นี้เสียก่อน ซึ่งข้อมูลที่โปรแกรมต่อไปนี้จะเป็นตัวกำหนดลักษณะการทำงานของเคาน์เตอร์ รีจิสเตอร์นี้สามารถเขียนข้อมูลเข้าไปได้อย่างเดียวไม่สามารถอ่านออกมาได้และจะติดต่อกับรีจิสเตอร์นี้ได้เมื่อขาA0 และ A1 มีลอจิกเป็น "1"

COUNTIER #0,#1,#2: เคาน์เตอร์ทั้งสามนี้มีลักษณะที่เหมือนกันและทำงานอย่างเป็นอิสระต่อกันและกัน แต่ละเคาน์เตอร์นี้ มีขนาด 16 บิท. PRE-SETTABLE,DOWN COUNTIER และสามารถนับได้เป็น ไบนารี หรือ BCD ก็ได้ ข้อมูลที่อยู่ภายในเคาน์เตอร์เหล่านี้สามารถถูกอ่านโดยไมโครโปรเซสเซอร์ได้โดยไม่ทำให้ข้อมูลภายในเคาน์เตอร์นั้นเสียหาย ซึ่งระบบสามารถจะแสดงค่าในเคาน์เตอร์ได้ตลอดเวลา โดยไม่กระทบกระเทือนการทำงานทั้งหมดของเคาน์เตอร์.

การต่อ 8253 เข้ากับ Z80.

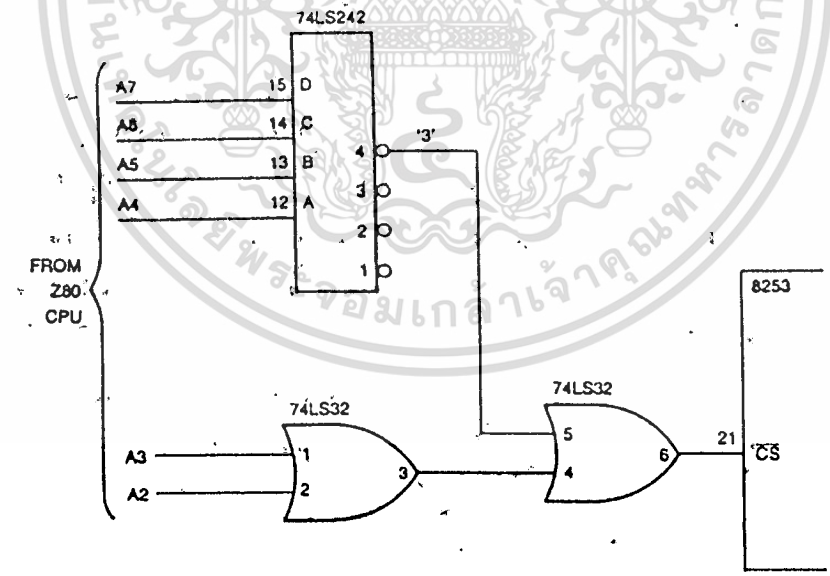
8253 เป็นเสมือนกับพอร์ท I/O จำนวน 4 พอร์ท ซึ่งแยกออกจากกันอย่างอิสระ เมื่อ Z80 ต้องการจะติดต่อกับพอร์ทเหล่านี้ เส้นสัญญาณอินพุท CS ซึ่งเป็นสายสัญญาณหลักที่ใช้ในการเลือกพอร์ท จะมีลอจิกเป็น "0" การนำข้อมูลเข้าออก ถ้าเทียบ I/O เหล่านี้กับ 8253 แล้ว พอร์ท I/O ก็คือรีจิสเตอร์ภายในทั้ง 4 ของ 8253

ในการถอดรหัสสายอินพุท CS ของ 8253 นั้นใช้สายแอดเดรส 6 บิทบน คือ A7-A2 ดังแสดงในตัวอย่าง (รูปที่ 8.4) โดยให้ 8253 มีลักษณะเป็นอุปกรณ์ I/O ของระบบ ซึ่งเราสมมุติว่า 8253 ถูกกำหนดให้อยู่ระหว่างตำแหน่ง I/O ที่ 30H กับ 33H ฉะนั้น 8253 จะประกอบไปด้วยพอร์ท I/O ที่ตำแหน่ง 30H กับ 33H ฉะนั้น 8253 จะประกอบไปด้วยพอร์ท I/O ที่ตำแหน่ง 30H,31H,32H และ 33H ดังรูป

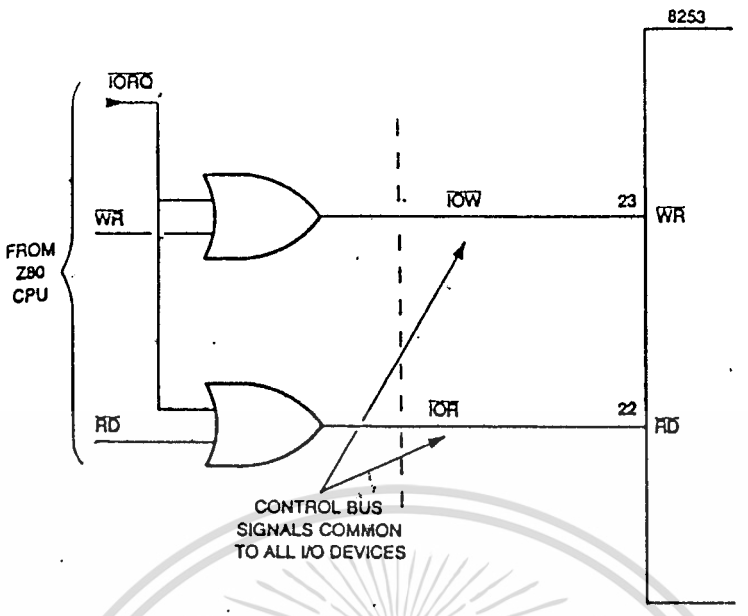
ส่วนสายสัญญาณอินพุท RD และ WR ของ 8253 นั้น จะถูกต่อโดยตรงกับ IOW และ IOR ซึ่งเป็นสายสัญญาณ ควบคุมของระบบ การต่อนี้เหมือนกับที่ใช้ใน 8255

บัสข้อมูลของ 8253 สามารถต่อโดยตรงเข้ากับบัสข้อมูลของ Z80 ได้ (สมมุติว่าในการใช้งานนี้ ไม่ต้องการใช้บัฟเฟอร์ข้อมูล) ในรูป ได้แสดงการต่อ 8253 เข้ากับZ80 อย่างสมบรูณ์ แต่เป็นแบบไม่ใช้กับบัฟเฟอร์ส่วนในรูป ถัดมาเป็น การต่อที่สมบรูณ์เมื่อใช้บัฟเฟอร์ บัฟเฟอร์ของบัสข้อมูลที่ใช้คือ 74LS245 ซึ่งเมื่อ 8253 ถูกเลือกและสัญญาณอินพุท RD มีลอจิกเป็น "0" (เป็นการอ่านจาก 8253) 74LS245 จะยอมให้ 8253 นำข้อมูลออกไปบนบัสข้อมูลของระบบ Z80 ได้ นอกจากนี้ 74LS245 จะยอมให้ข้อมูลจากบัสข้อมูลของระบบ Z80 เข้ามาที่ขา DATA INPUT ของ 8253 ได้

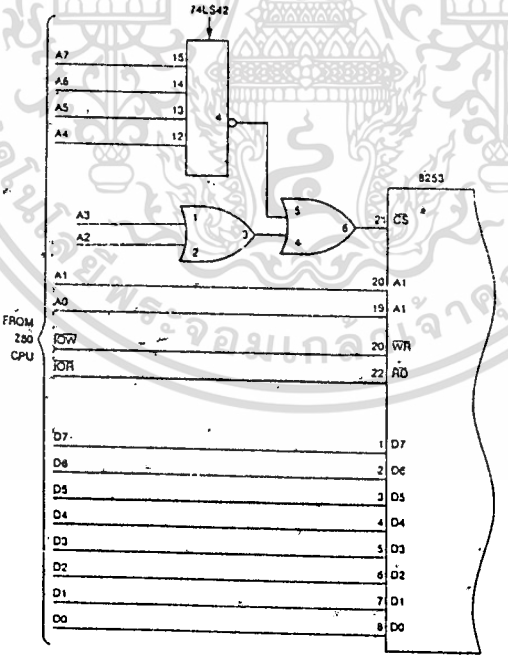
การโปรแกรม 8253 (CONTROL WORD FORMAT)



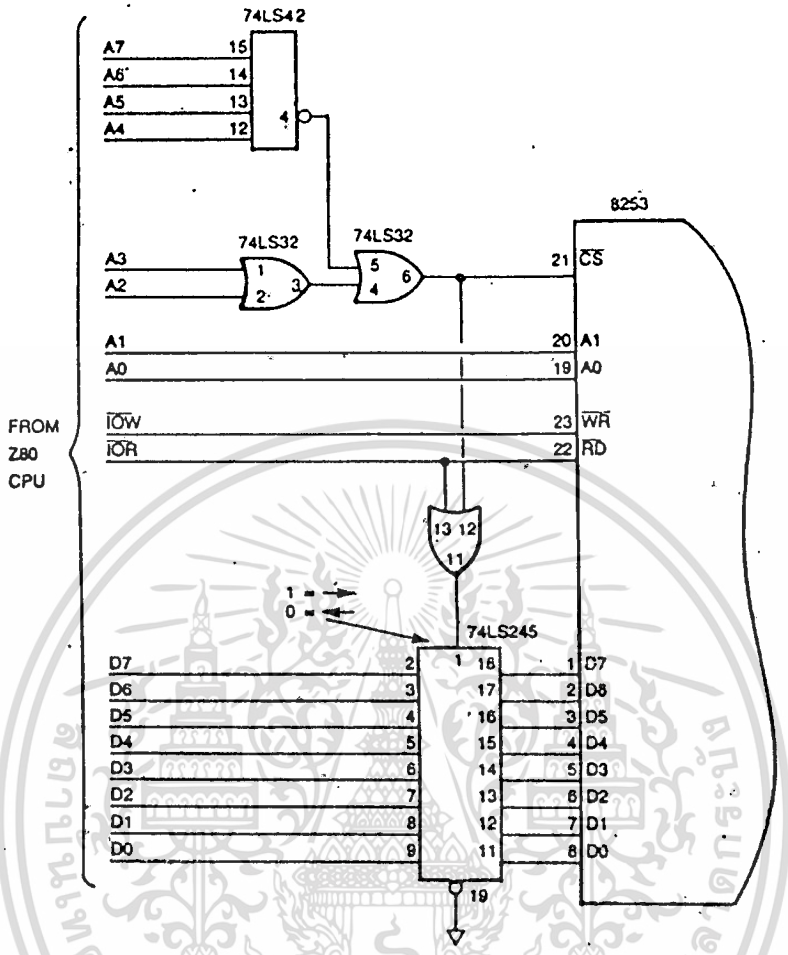
รูปที่ 4 วงจรที่แสดงถึงการถอดรหัส CS INPUT จากเส้นแอดเดรส A7-A2



รูปที่ 5 ลอจิกไดอะแกรมแสดงถึงการสร้างสัญญาณ IOR และ IOW



รูปที่ 6 วงจรสมบูรณ์ของการเชื่อมต่อระหว่าง 8253 กับระบบบัสของ Z80 โดยไม่ใช้บัฟเฟอร์



รูปที่ 7 การต่อ Z80 เข้ากับ 8253 โดยการใช้ปัฟเฟอร์

เนื่องจากเคาน์เตอร์ของ 8253 มีจำนวน 3 ตัวด้วยกัน ฉะนั้นการโปรแกรมเคาน์เตอร์นั้น จำเป็นต้องกำหนดเคาน์เตอร์ที่ต้องการจะโปรแกรมเสียก่อนการกำหนดทำได้โดยให้ลอจิกที่ถูกต้องการแล้ว เคาน์เตอร์นั้นจะถูกเซ็ทและจะอยู่ในสภาพนั้น จนกว่าจะมีคำสั่งควบคุมอื่นมาทำให้เปลี่ยนแปลง ส่วนการกำหนดค่าลอจิกของบิท D7 และ D6 สำหรับใช้เลือกเคาน์เตอร์เป็นดังนี้

D7	D6	COUNTER SELECT
0	0	0
0	1	1
1	1	2
1	1	ไม่มีความหมาย

เมื่อเลือกเคาน์เตอร์จากการใช้บิท D7 และ D6 ได้แล้ว ต่อไปบิท D5 และ D4 จะเป็นตัวกำหนดว่าเคาน์เตอร์นี้ (หรือรีจิสเตอร์) จะใช้ใน READ/LOAD MODE ซึ่งโหมดการอ่าน (READ MODE) เป็นโหมดที่ไมโครโปรเซสเซอร์อ่านข้อมูลจากเคาน์เตอร์ส่วนโหมดการโหลด (LOAD MODE) เป็นโหมดที่ไมโครโปรเซสเซอร์เขียนข้อมูลเข้าไปให้เคาน์เตอร์ม บิท D5 และ D4 ถูกกำหนดดังนี้

D5	D4	R/L DEFINITION
0	0	ค่าในเคาน์เตอร์ถูกแลทช์ หมายความว่า ค่าที่มีอยู่ในเคาน์เตอร์ที่ถูกกำหนดนี้ จะนำเข้าไปเก็บไว้ใน FLIP-FLOP ซึ่ง CPU สามารถอ่านออกไปได้
0	1	READ/LOAD เฉพาะไบท์ที่มีนัยสำคัญต่ำ (LEAST-SIGNIFICANT BYTE)
1	0	READ/LOAD เฉพาะไบท์ที่มีนัยสำคัญสูง (MOST-SIGNIFICANT BYTE)
1	1	READ/LOAD ไบท์ที่มีนัยสำคัญต่ำก่อน เสร็จแล้วตามด้วยไบท์ที่นัยสำคัญสูง

หมายเหตุ ใน 1 ไบท์ของเคาน์เตอร์ประกอบด้วย 16 บิทโดยบิท D0-D7 เป็น LEAST-SIGNIFICANT BYTE และ D8-D15 เป็น MOST-SIGNIFICANT BYTE.

CONTROL BYTE D7-D0

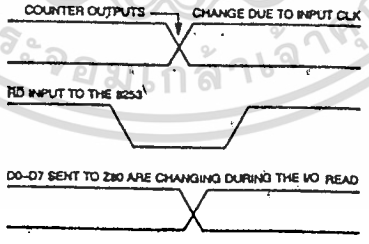
D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	RL1	RL0	M2	M1	M0	BCP

รูปที่ 8.8 ข้อกำหนดของแต่ละบิทของรีจิสเตอร์ควบคุม (control register)

เมื่อ D5 และ D4 มีค่าเป็น 00H เคาน์เตอร์จะถูกทำให้อยู่ในโหลดการแลทช์ (LATCH) ซึ่งเป็นโหมดที่ใช้สำหรับการอ่านค่าของเคาน์เตอร์ขณะที่เคาน์เตอร์ยังทำงานอยู่ การเขียนโหมดนี้ให้กับรีจิสเตอร์ควบคุม จะทำให้ค่าที่อยู่ในเคาน์เตอร์ถูกแลทช์ให้กับรีจิสเตอร์ภายใน และเมื่อทำการอ่านเคาน์เตอร์ค่านี้จะถูกอ่านออกไป.

ถ้าไม่อยู่ในโหมดการแลทช์ แล้วการอ่านข้อมูลจะเกิดข้อผิดพลาดขึ้นได้เพราะขณะที่ทำการอ่านข้อมูลนั้น ขบวนการที่เกิดขึ้นในเคาน์เตอร์จะทำให้ข้อมูลที่อยู่เดิมเปลี่ยนไปเป็นผลทำให้ข้อมูลที่ป้อนเข้า CPU เกิดผิดพลาดขึ้น ฉะนั้นเพื่อที่จะอ่านค่าของเคาน์เตอร์ให้ถูกต้องในขณะที่เคาน์เตอร์กำลังอยู่ในขบวนการนับอยู่นั้น สามารถทำได้โดยกำหนดคำสั่งควบคุมการแลทช์ (LATCH CONTROL WORD) ก่อนแล้วจึงให้คำสั่งควบคุมอื่นที่เป็นคำสั่งการอ่านในไบท์ต่อไป

ยังมีอีกวิธีหนึ่งสำหรับการทำให้ค่าข้อมูลของเคาน์เตอร์ไม่เปลี่ยนแปลงขณะทำการอ่าน โดยการใช้วงจรภายนอกที่ทำให้การนับของเคาน์เตอร์หยุดชั่วขณะในระหว่างที่ทำการอ่าน แต่ละวิธีที่กล่าวมานี้ก็มีข้อเสียในตัวมันเอง ดังในวิธีการแลทช์ อาจทำให้ไมโครโปรเซสเซอร์ทำการอ่านข้อมูลเก่าที่เกิดก่อนหน้านี้หลายๆ รอบ ซึ่งขึ้นกับความเร็วของการนับ และไบท์ของเคาน์เตอร์ที่กำลังถูกอ่าน ส่วนวิธีที่ทำให้การนับเคาน์เตอร์หยุดชั่วขณะโดยวงจรภายนอกนั้นมีข้อเสีย ก็คือจะต้องใช้อุปกรณ์ทางฮาร์ดแวร์มาเพิ่มเติมอีก ซึ่งอาจทำให้เกิดการเปลี่ยนแปลงการทำงานของระบบทั้งหมด ขึ้นอยู่กับความเหมาะสมว่าจะเลือกใช้วิธีใดจึงจะดีที่สุดสำหรับการใช้งานนั้นๆ



รูปที่ 9 ไดอะแกรมเวลาที่แสดงการผิดพลาดระหว่างการอ่านข้อมูลจากเคาน์เตอร์ของ 8253 ซึ่งสามารถแก้ไขโดยการที่ไมโครโปรเซสเซอร์ LATCH ข้อมูลเอาท์พุทของเคาน์เตอร์ก่อนที่จะทำการอ่าน

ยังมีอีก 4 บิท ที่เหลือของคำสั่งควบคุมในรูป 8.8 คือ D3,D2,D1 และ D0 แต่จะกล่าวถึง 3 บิทแรกก่อนคือ D3,D2 และ D1 บิทเหล่านี้เป็นบิทที่กำหนดโหมดการทำงานพื้นฐานของเคาน์เตอร์ ซึ่งต่อไปจะได้อธิบายและแสดงตัวอย่างการใช้เคาน์เตอร์ในแต่ละโหมดทั้ง 5 โหมดนี้ ในที่นี้มาดูลอจิกที่ให้กับ D3,D2 และ D1 ในแต่ละโหมดดังนี้

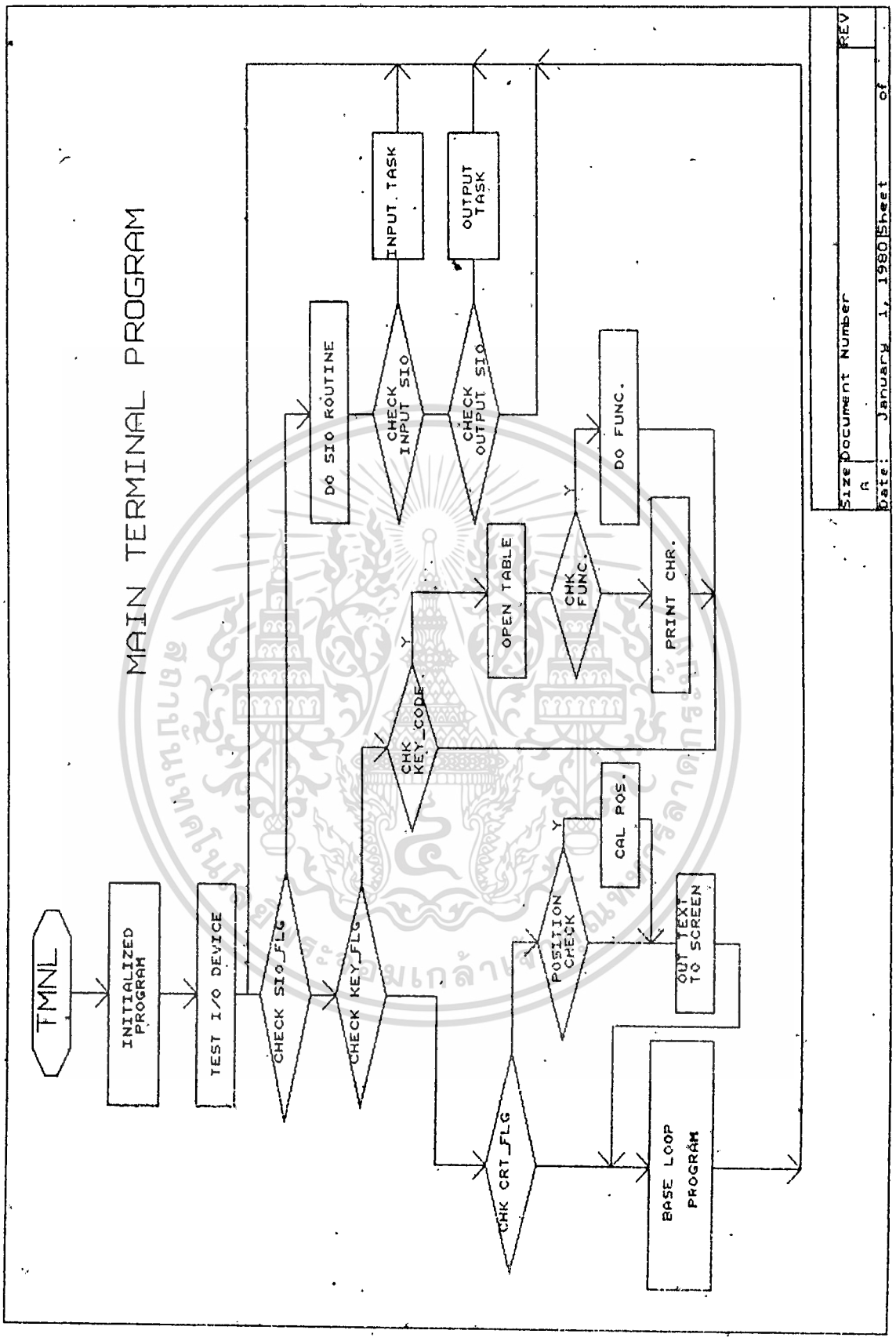
D3	D2	D1	MODE VALUE
0	0	0	MODE 0: INTERRUPT ON TERMINAL COUNT
0	0	1	MODE 1: PROGRAMABLE ONE-SHOT
X	1	0	MODE 2: RATE GENERATER
X	1	1	MODE 3: SQUARE WABE GENERATER
1	0	0	MODE 4: SOFTWARE TRIGGERED STROBE
1	0	1	MODE 5: SOFTWARE TRIGGER STROBE

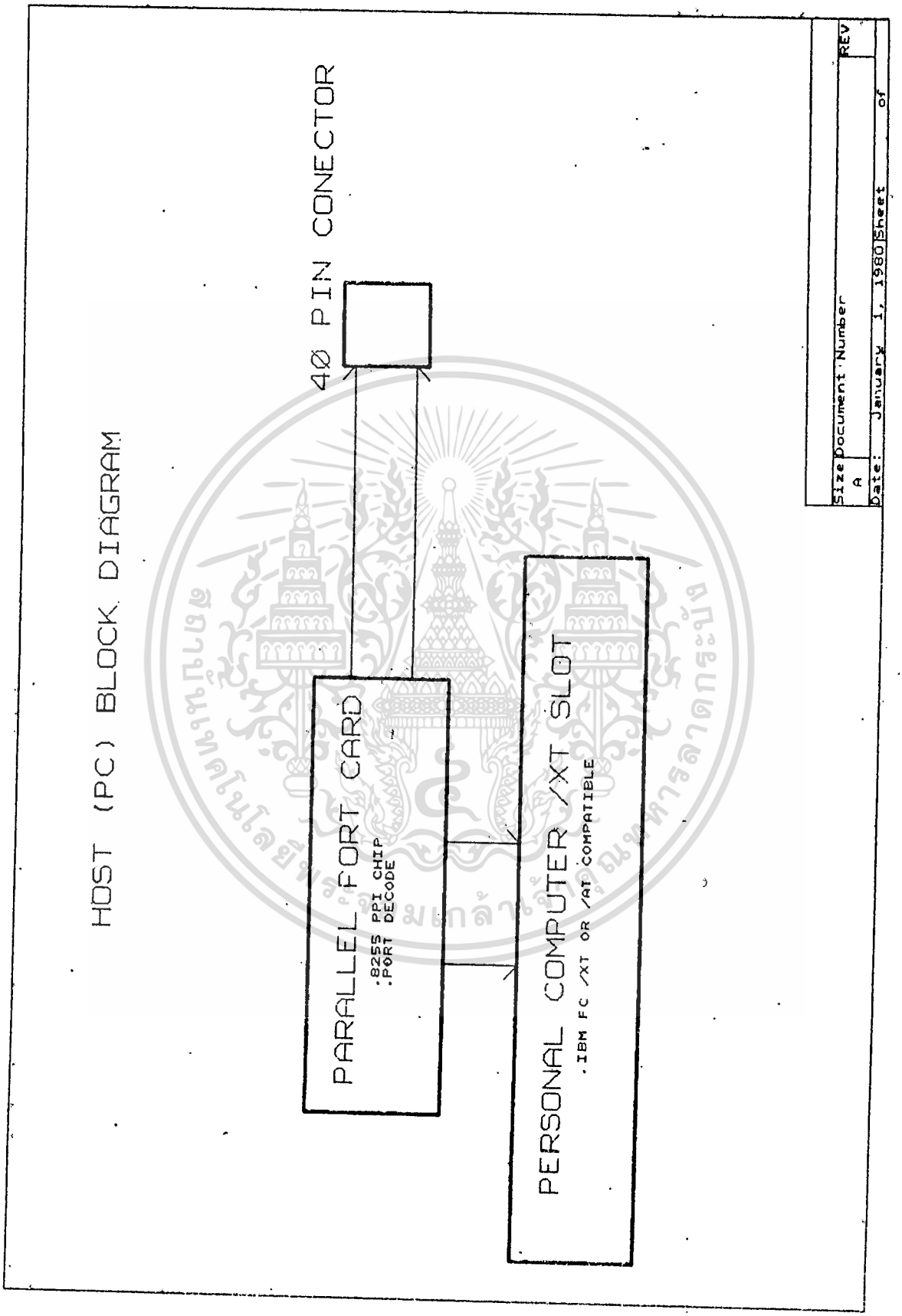
บิทสุดท้ายของคำสั่งควบคุมคือ D0 ใช้กำหนดลักษณะการนับของเคาน์เตอร์ว่ามีลักษณะการนับเป็นอย่างไร นั่นคือจะนับเป็น BCD หรือเป็นไบนารี ถ้า D0 มีลอจิกเป็น"1" เคาน์เตอร์ จะนับแบบ BCD ถ้า D0 มีลอจิก เป็น "0" จะนับแบบไบนารี ค่าที่มากที่สุดสำหรับการนับในโหมดการนับแบบไบนารีมีค่าเท่ากับ 2^{16} และในโหมดการนับแบบ BCDเป็น 10^4

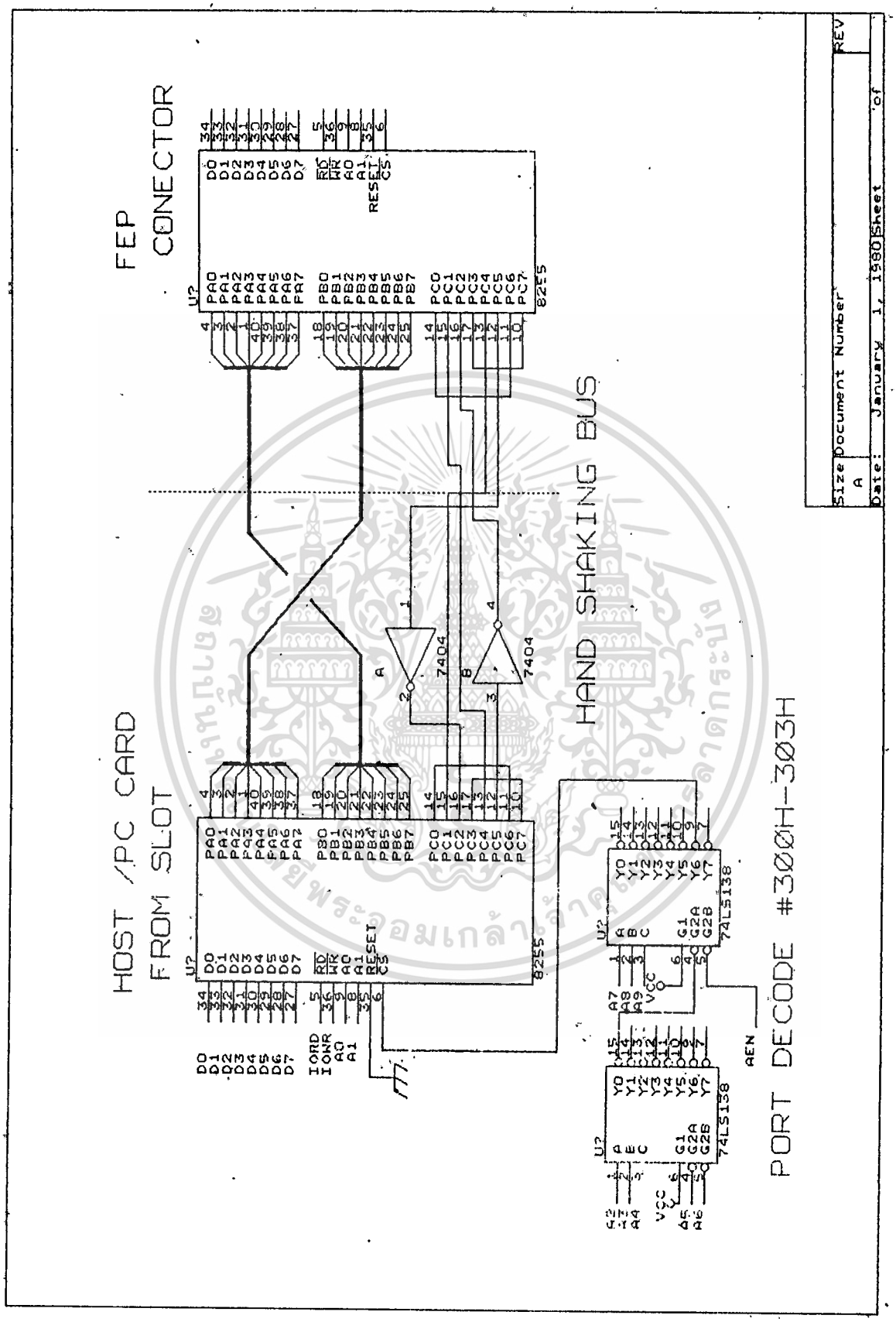
บทที่ 3

วงจรและหลักการทำงาน

การทำงานของ INTERFACE CARD ของ HOST PC นั้นประกอบด้วย PORT 8255PARALLEL PORT และ 74138 DECODE PORT ENABLE TO 8255 เพื่อการต่อเข้ากับบัสCPU 8088 ในการเชื่อมต่อกับ DATA BUS และ ADDRESS BUS รวมทั้ง CONTROL BUS การทำงานของ PARALLEL PORT เป็น MODE HAND SHAKING เช่นเดียวกับ FEPในการติดต่อกับเครื่องของ PC จะติดต่อผ่านพอร์ตของ FEP และมีสาย CONTROL ดังที่เห็นจากBLOCK DIAGRAM จะเห็นได้ว่านอกจากจะมี PORT DATA และสัญญาณควบคุม HAND SHAKING อีก 2 สัญญาณ







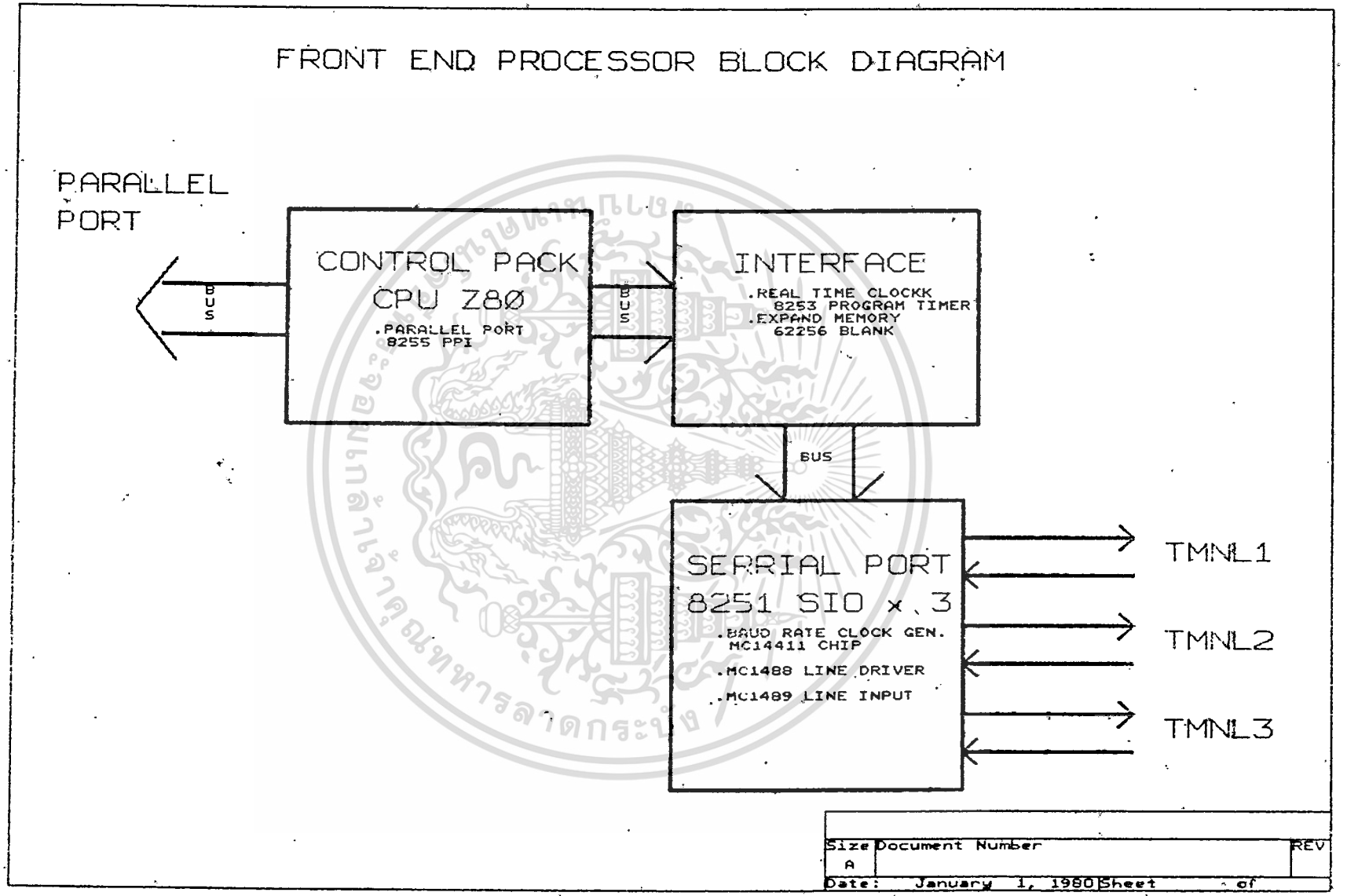
การทำงานของ FEP

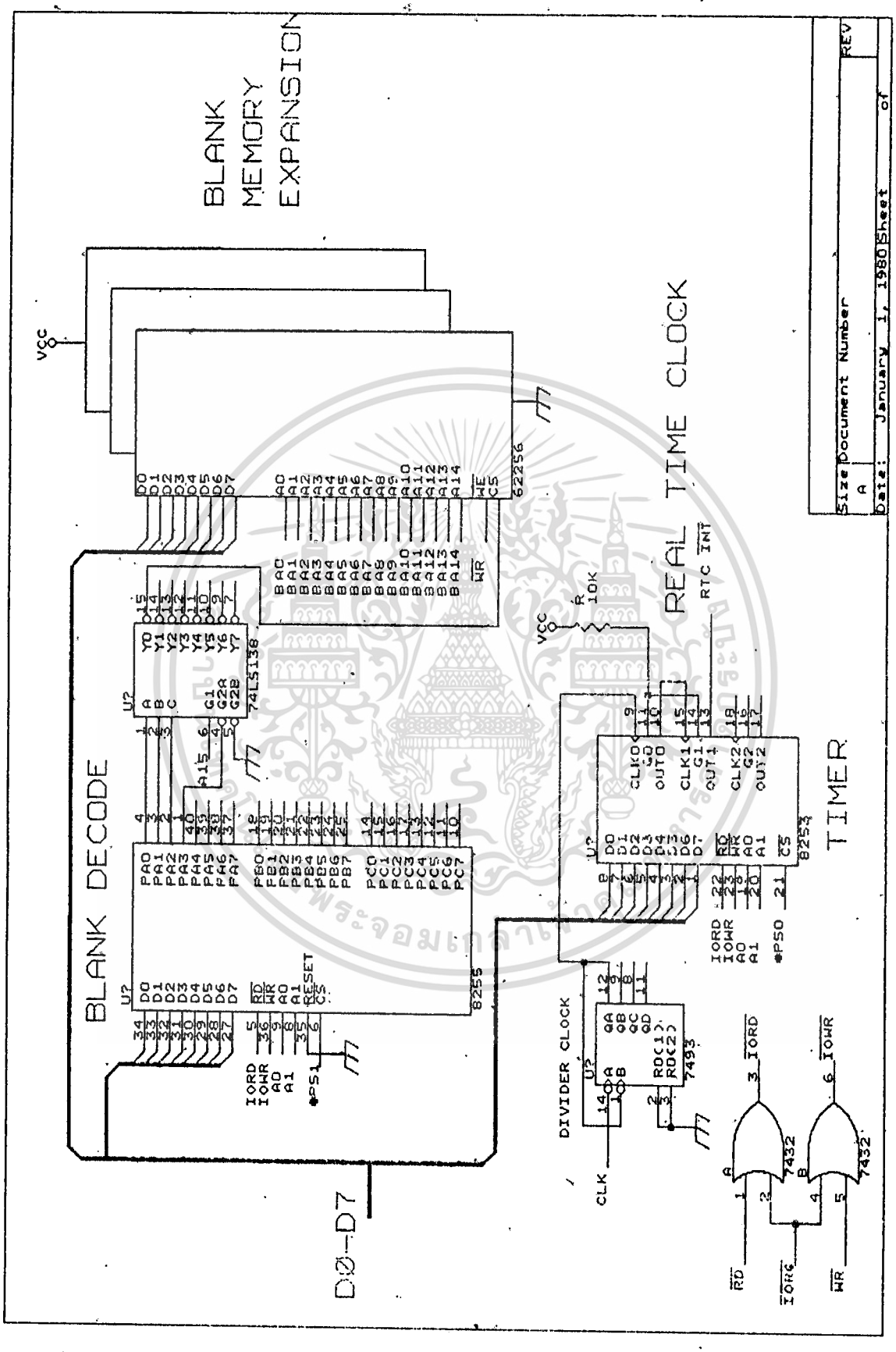
FEP ใช้ CONTROL PACK Z80 ทำหน้าที่เป็น CONTROLLER ซึ่งควบคุมการทำงานจากโปรแกรมที่เขียนขึ้นเก็บไว้ใน EPROM ในการทำงานของ FEP จะควบคุม PORT และจัดการ MEMORY ต่างๆ ดังนี้คือ การรับข้อมูลจาก HOST/PC ทาง PARALLEL PORT ซึ่งมี PORT 8255 เป็นตัวรับส่งข้อมูลกับ INTERFACE CARD ของ PC โดยผ่านสายแพ40 เส้น การทำงาน ของ PORT 8255 ถูกโปรแกรมให้เป็น MODE 1 HAND SHAKING

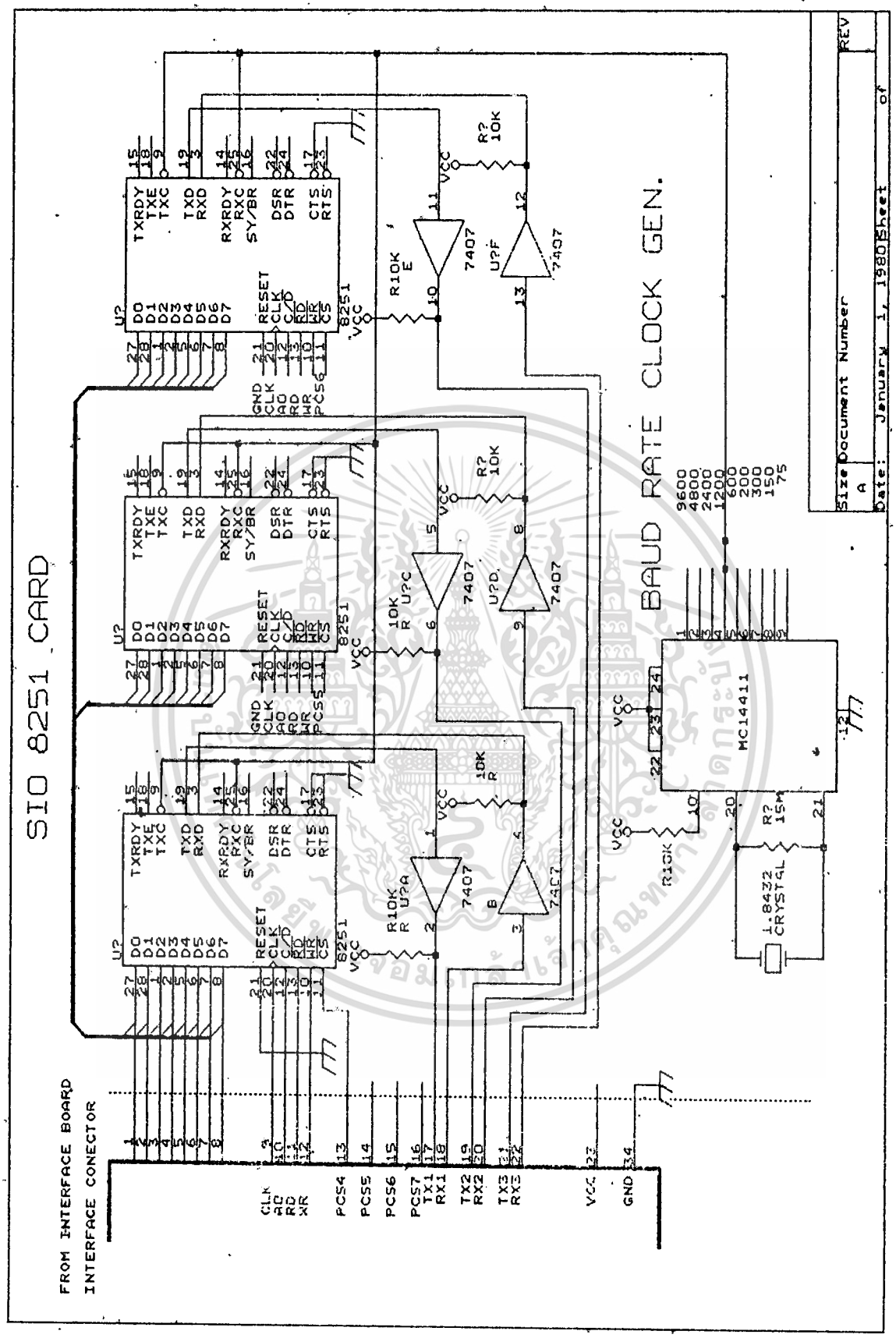
ทางด้าน PORT LATCH ทำหน้าที่ในการเลือก BLANK MEMORY โดยแบ่งMEMORY ของ Z80 ออกเป็น 32k ล่าง ใช้ DECODE แบบปกติ และ 32k บน ซึ่งต่อกับ 61256 (32k RAM) โดย 32k บนนี้ สามารถเลือกใช้คำสั่ง OUT PORT เป็นSELECT BLANK สามารถเลือกได้สูงสุด 8 BLAND

ในการติดต่อกับ TERMINAL 8 ตัวจะใช้ 8251 SIO PORT 3 ตัว โดยแต่ละตัว จะรับผิดชอบ TERMINAL แต่ละตัวไป บน SIO BOARD จะประกอบด้วย 8251 4 ตัวแต่ใช้งานเพียง 3 ตัว MC14411 จะสร้าง SIO CLOCK โดยการจาก X-TAL 1.8432MH, เมื่อได้สัญญาณ CLOCK มาแล้วแต่สามารถใช้กับ BAUD RATE 50-9600 BAUD. สามารถเลือกโดยจุด JUMPER บน BOARD ในที่นี้เซ็นให้คงตัวเท่ากับ SPEED 1200BD. ทั้งหมด

ส่วนทางด้าน REALTIME COLOCK จะสร้างสัญญาณโดย 8253PROGRAMMABLE TIME ทำหน้าที่สร้างสัญญาณ INTERRUPT CLOCK แก่ CPU โดยสามารถใช้โปรแกรมเปลี่ยนแปลงค่าเวลาการ INTERRUPT ได้ ตามต้องการ ในที่นี้ตั้งไว้ที่ 1 MS.


FRONT END PROCESSOR BLOCK DIAGRAM
PARALLEL PORT
BUS
CONTROL PACK
CPU Z80
.PARALLEL PORT 8255 PPI
BUS
INTERFACE
.REAL TIME CLOCKK 8253 PROGRAM TIMER
.EXPAND MEMORY 62256 BLANK
BUS
SERRIAL PORT
8251 SIO x 3
.BAUD RATE CLOCK GEN. MC14411 CHIP
.MC1488 LINE DRIVER
.MC1489 LINE INPUT
TMNL1
TMNL2
TMNL3
Size A
Document Number
REV
Date: January 1, 1980 Sheet of





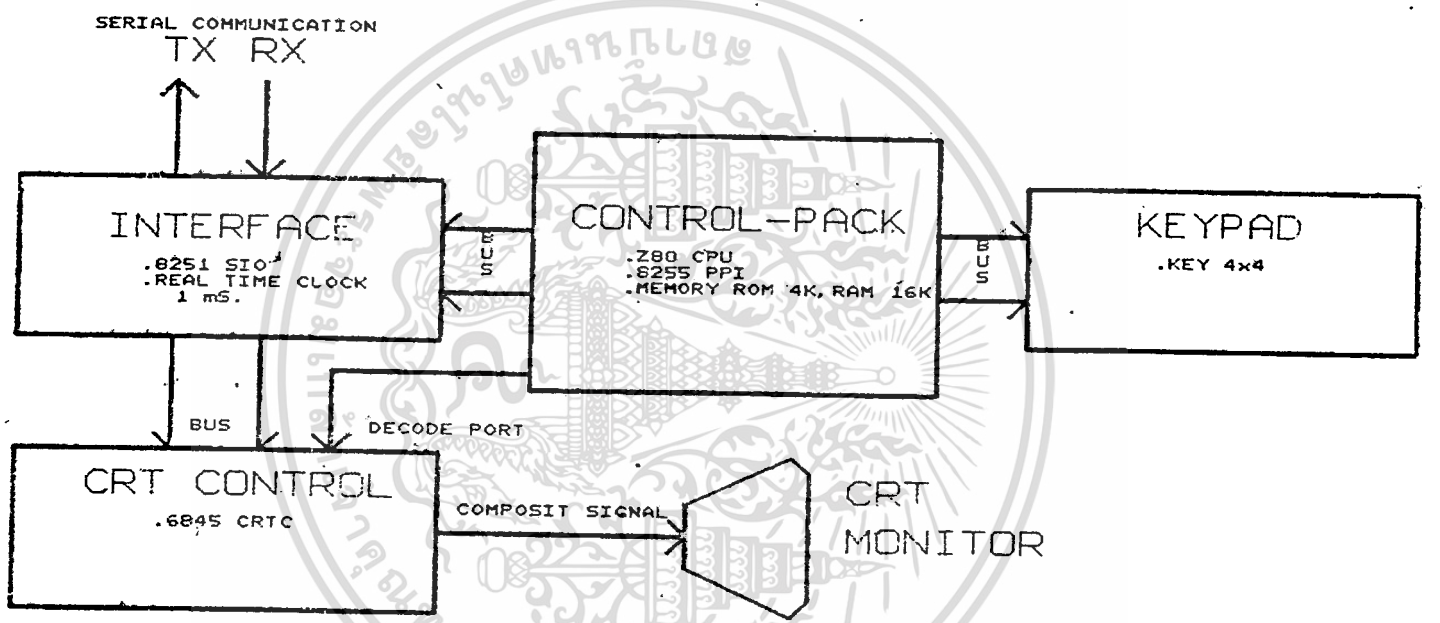
วงจรและหลักการทำงานของ TERMINAL

TERMINAL ใช้ CONTROL-PACK Z80 ทำหน้าที่เป็น CONTROLLER ซึ่งควบคุมโดยโปรแกรมในEPROM CONTROL-PACK Z80 จะควบคุมการทำงานของ 8255 ทำหน้าที่ SCAN KEYPAD4x4 และแสดง LED DISPLAY & DOT และรับ INPUT จาก DIPSW 8 ตัว โดยที่PORT C ทำหน้าที่ SCAN KEYPAD IN4 OUT4 จึงได้จำนวนคีย์ 4x4 = 16 คีย์ PORTAจะทำหน้าที่แสดงผลออก LED เพื่อแสดงสถานะการทำงาน และ PORT B รับข้อมูล INPUT จาก DIP SWIT 8 ตัว

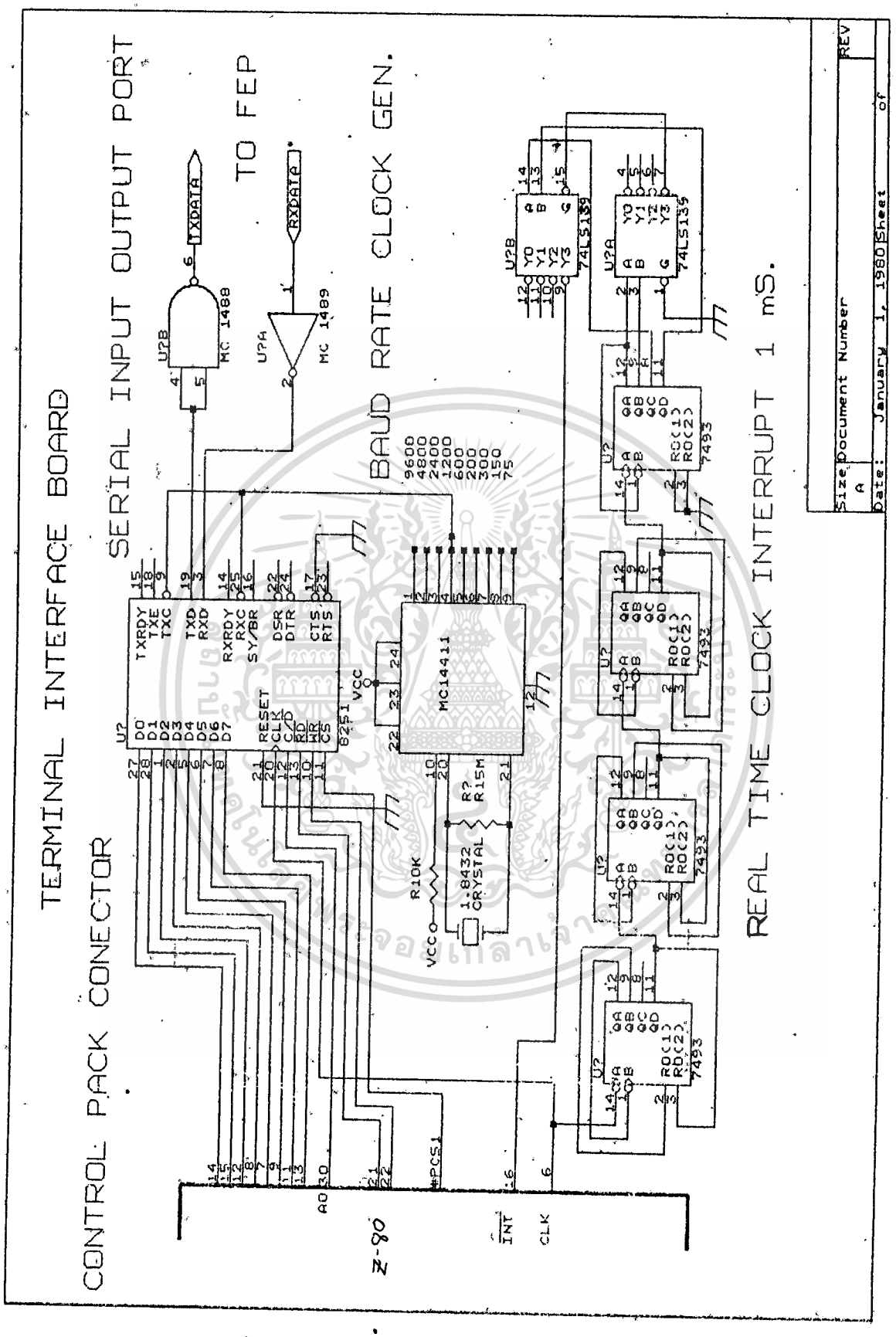
ส่วนทางวงจร INTERFACE จะมี 810 8251 ทำหน้าที่รับข้อมูล ทาง SERIAL PORT โดย MC 14411 ทำหน้าที่สร้าง BAUD RATE CLOCK ให้กับ 8265 เป็นตัวสำคัญในจัดการควบคุมการสร้างสัญญาณ VIDEO ที่ไป DISPLAY จอรับข้อมูล ในการทำงาน DISPLAY จอนี้ใช้แสดงแบบ TEXT MODE ซึ่งมี CHARACTER GEN. 2732 เป็นตัวเก็บ FONT การแสดงตัวอักษรเลาไว้ 6845 จะควบคุมโดยรับข้อมูลจาก CPU เก็บ CODE CHARACTER ไว้ใน VIDEO RAM 6116 จากนั้นก็จะนำข้อมูลใน VIDEO RAM มาเปิดไฟ CHARACTER.GEN. แล้วจะส่งข้อมูลนี้ออกมาเป็น SERIAL ไป แสดงผลออก OUTPUT CRTพร้อมอังจัดการสัญญาณ VEE.SYNC.และ HOR SYNC. TRANSISTOR จะเป็นตัว MIXERสัญญาณ VIDEO,V-SYN,H-SYN ให้ออกมาเป็น COMPOSITE VIDEO ต่อที่ OUTPUT ต่อไป

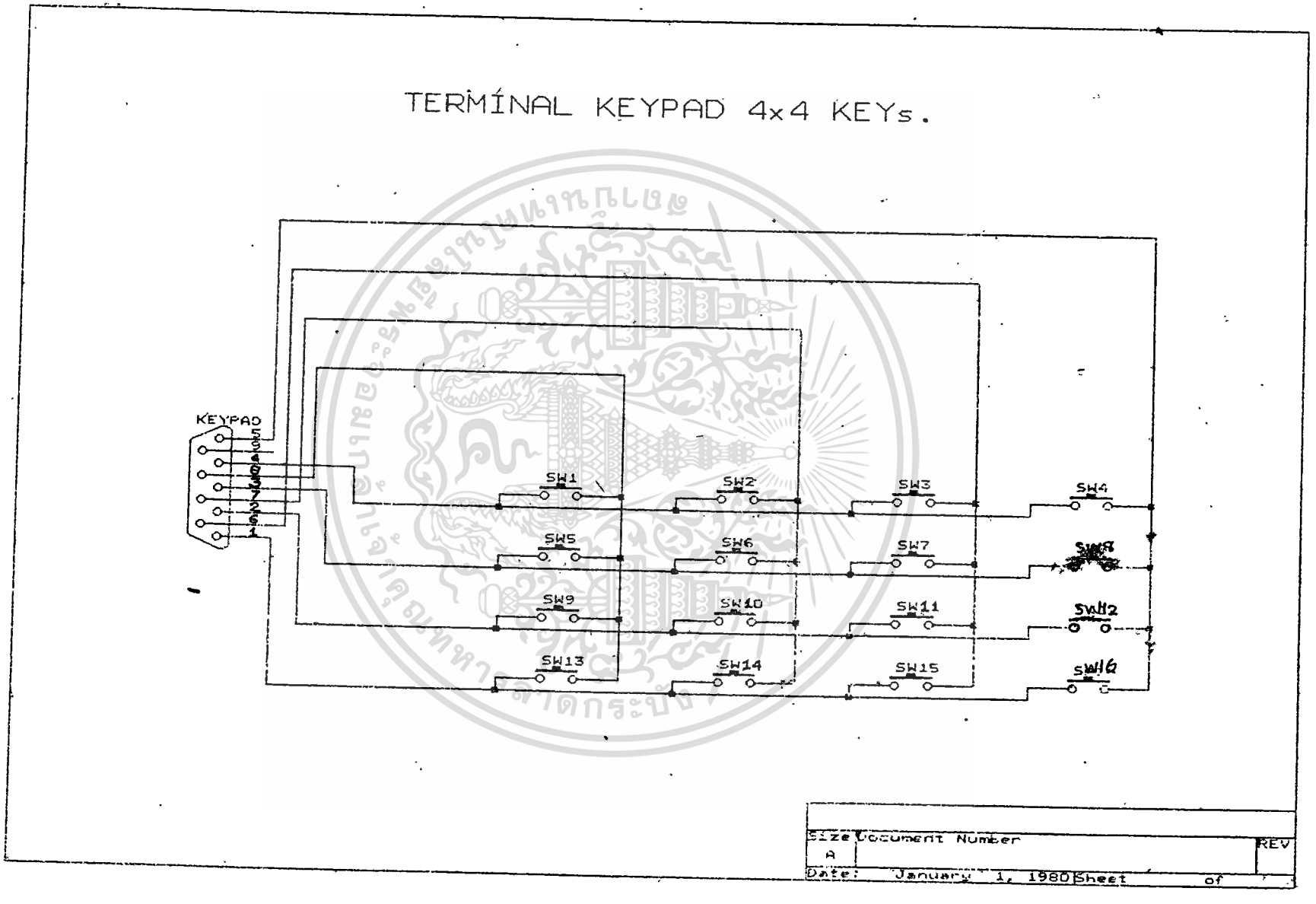
ส่วน RESETTIME CLOCK ใช้วงจรการทำงานที่ใช้ไฟ 74LS93 BIN COUNTER ทำหน้าที่หารความถี่จาก CPU CLOCK. แล้วใช้ 74139 DECODER ทำหน้าที่แปลงสัญญาณเป็น INTERRUPT CLOCK

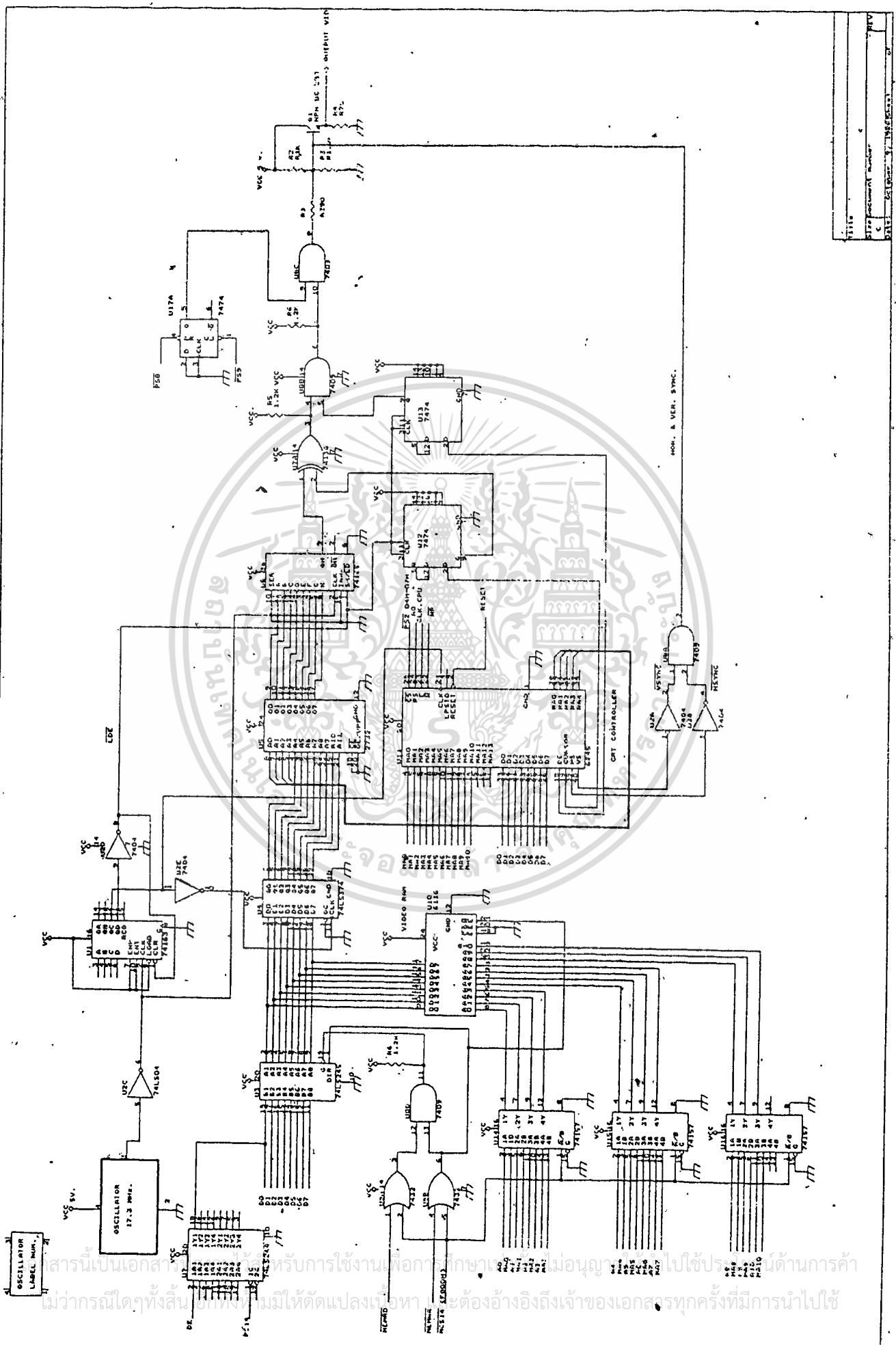
TERMINAL BLOCK DIAGRAM

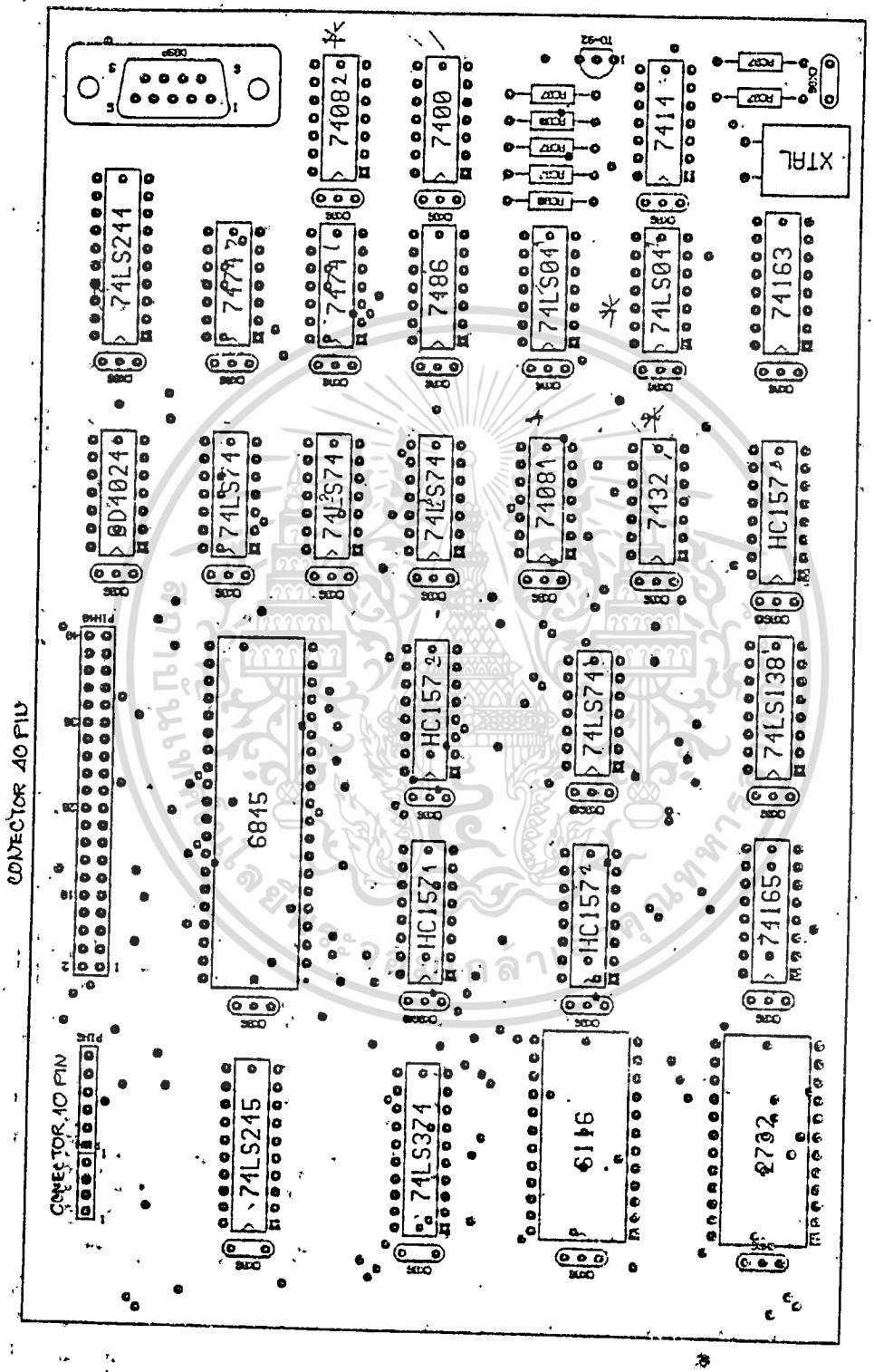


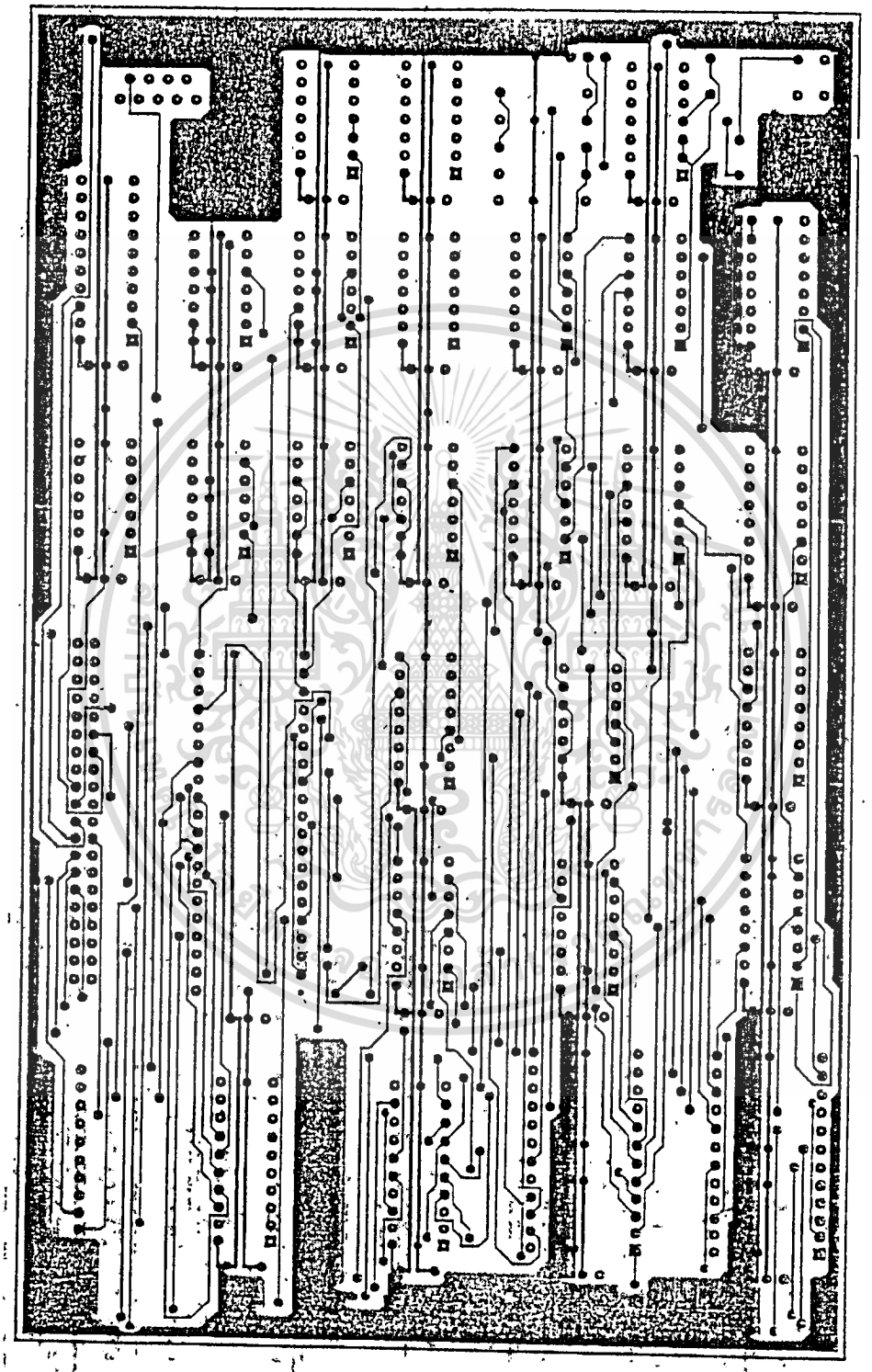
Size	Document Number	REV
A		
Date: January 1, 1980	Sheet of	

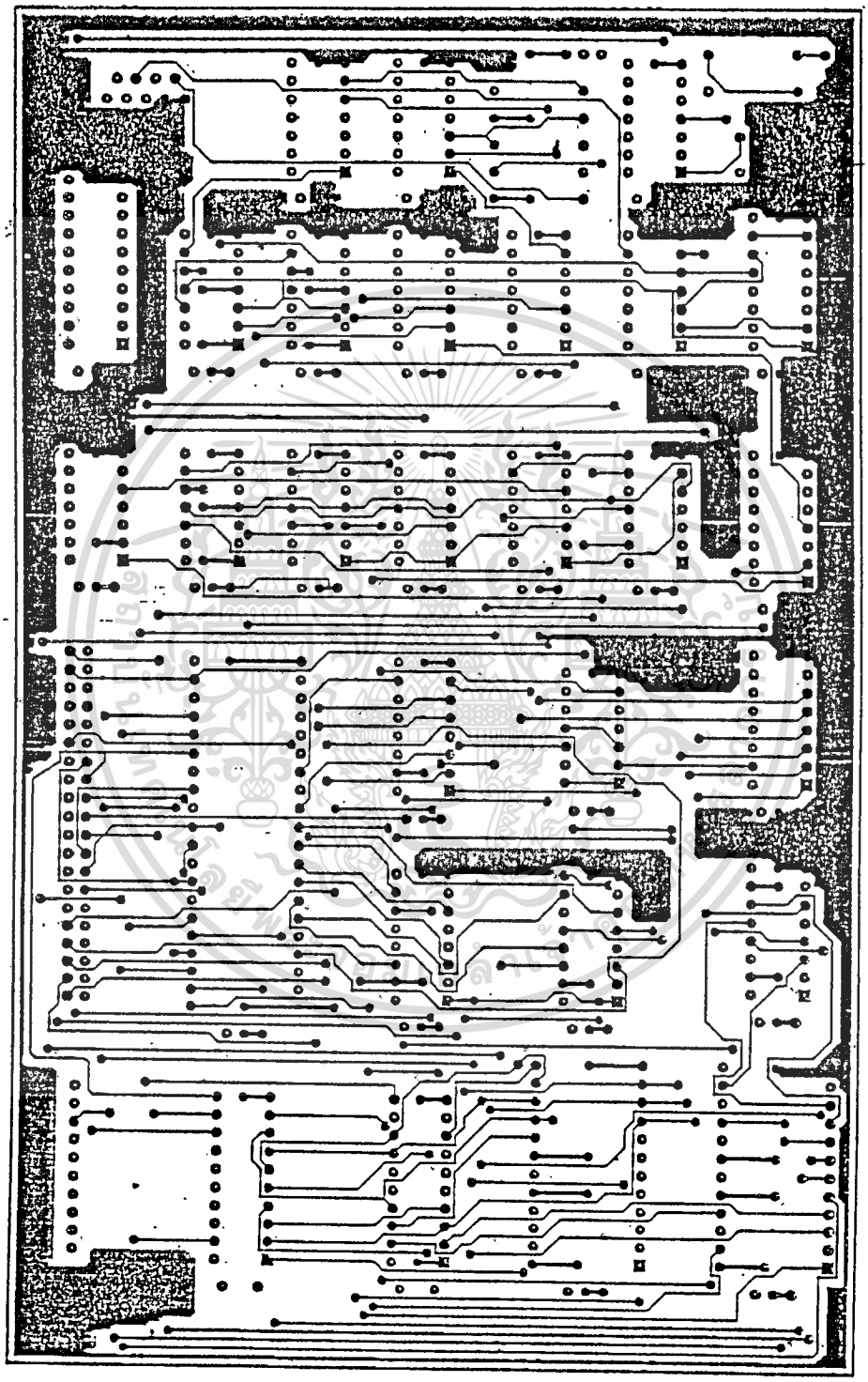


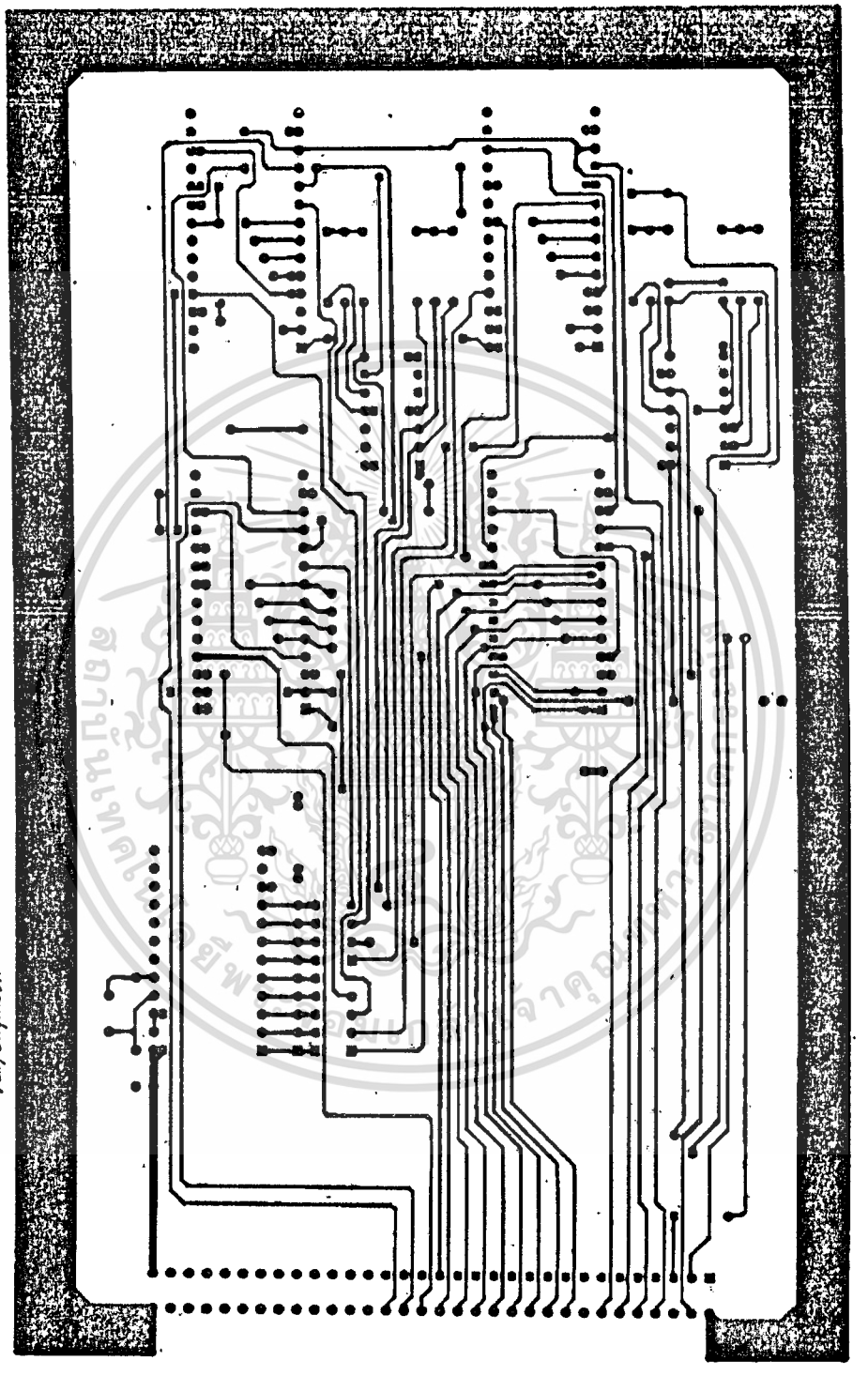


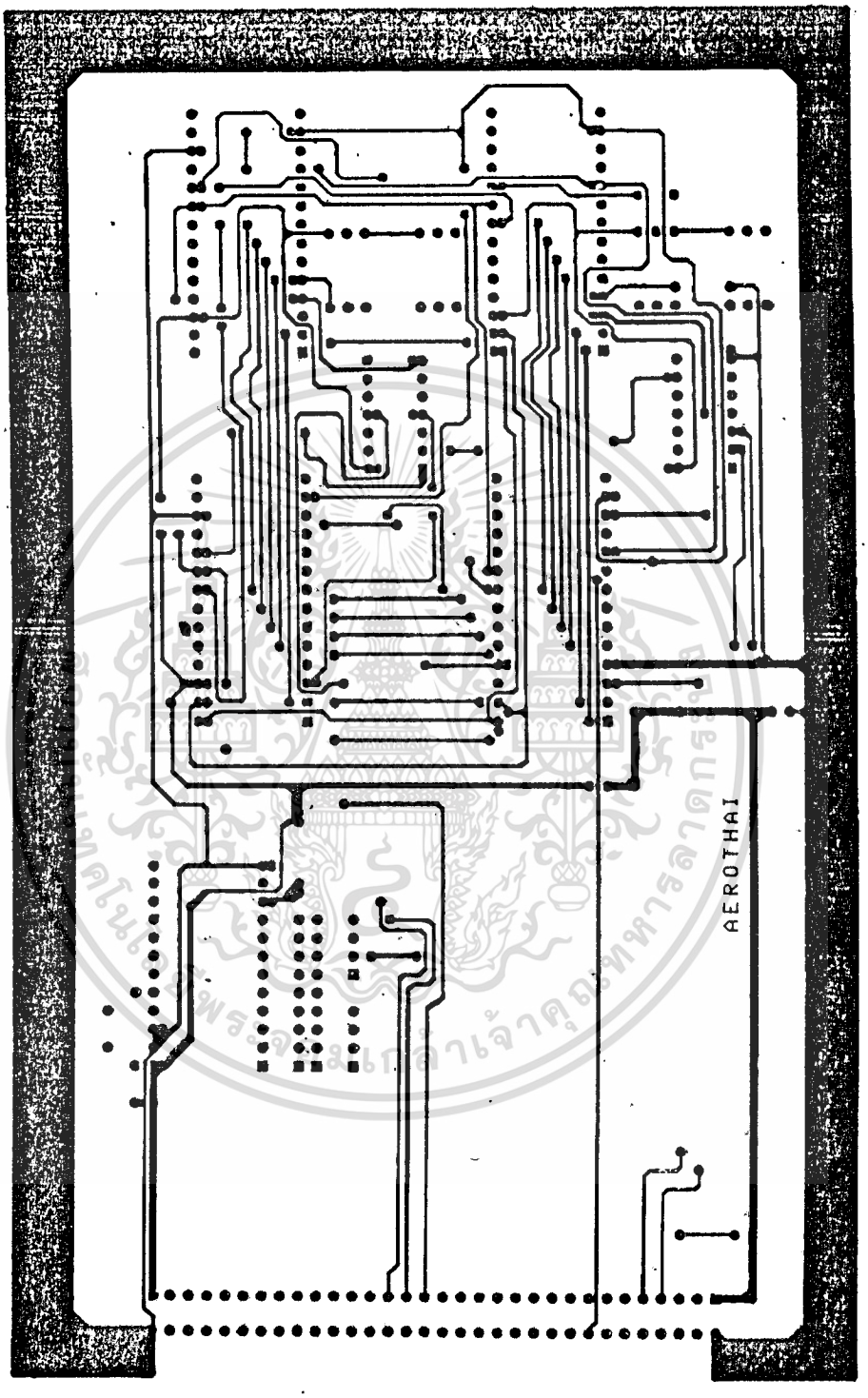












บทที่ 4

สรุปผลการทำงานและวิธีการใช้งาน

ในด้านการทำงานของวงจรระบบ DTN นี้ เนื่องจากได้ CONCEPT มาจากการได้ใช้งานระบบสื่อสารข้อมูลระบบไพ่ๆ ที่มีความสามารถสูง และ เล็งเห็นแนวทางการประยุกต์ใช้งานข้อมูลเป็นการให้ข่าวสาร บริการแก่ผู้ใช้ได้ทั่วไป ซึ่งไม่ต้องใช้ความรู้ทาง COMPUTER มากนัก เพียง SETUP ระบบได้แล้วก็สามารถ RUN ระบบให้ทำงานได้เองได้เลย

จากการทำงานของโปรเจคชุดนี้ อยู่ในขั้นน่าพอใจ แต่ยังมีขีดความสามารถที่ยังจำกัดอยู่ ซึ่งยังสามารถพัฒนาทาง SOFTWARE ให้เล่น UTILITY และ APPLICATION PROGRAM ให้มากกว่าเดิมได้ แต่อย่างน้อยโปรเจคชุดนี้ก็พอที่จะเป็นเครื่องต้นแบบให้เป็นแนวทางในการพัฒนาได้อีกในอนาคต

ภาคผนวก

LISTING ASSEMBLY PROGRAMMING

```
;PROGRAM LOAD FILE IN BUFFER
;*************** PROGRAM SEND DATA 8K PARALLEL PORT *************
CODE        SEGMENT    WORD
ASSUME      CS:CODE,DS:CODE
PPIMODE     EQU   0BCH
PORTA       EQU   0300H
PORTB       EQU   0301H
PORTC       EQU   0302H
CONTP       EQU   0303H
OBITSET     EQU   05H
OBITRESET   EQU   04H
IBITSET     EQU   09H
IBITRESET   EQU   08H
            ORG 100H
START:

_MAIN       PROC    NEAR
            MOV   AL,PPIMODE      ;PROGRAM 8255 PPI
            MOV   DX,CONTP
            OUT   DX,AL

            CALL  OPENFILE
            CALL  LOADFILE
            CALL  CLOSFILE

;SEND DATA TO PARALLEL PORT

            MOV   CX,1024                 ;SET 8K DATA
            MOV   BX,OFFSET DATABUF       ;SET LABEL DATA TO SEND
AGAIN:      MOV   AL,OBITSET
            MOV   DX,CONTP
            OUT   DX,AL
```

```
;************ OPEN FILE **********
OPENFILE    PROC    NEAR
            PUSH    AX
            MOV     AH,3DH
            MOV     AL,2
;            MOV     DX,SEG FNAME
;            MOV     DS,DX
            MOV     DX,OFFSET FNAME
            INT     21H
            JC      FAILURE
            MOV     HANDLE,AX
            MOV     AH,9
            MOV     DX,OFFSET OPENAVL
            INT     21H
            JMP     ENDO
FAILURE:    MOV     AH,9
            MOV     DX,OFFSET OPENFL
            INT     21H
ENDO:       POP     AX
            RET
OPENFILE    ENDP

;************ LOAD FILE **********
LOADFILE    PROC    NEAR
            PUSH    AX
            MOV     AH,3FH
            MOV     BX,HANDLE
            MOV     CX,1024
;            MOV     DX,SEG DATABUF
;            MOV     DS,DX
            MOV     DX,OFFSET DATABUF
            INT     21H
```

```
            JC      LOADFL
            MOV     AH,9
            MOV     DX,OFFSET _LOADAVL
            INT     21H
            JMP     ENDLD
LOADFL:     MOV     AH,9
            MOV     DX,OFFSET _LOADFL
            INT     21H
ENDLD:      POP     AX
            RET
LOADFILE    ENDP

;************ CLOSE FILE **********
CLOSFILE    PROC    NEAR
            PUSH    AX
            MOV     AH,3EH
            MOV     BX,HANDLE
            INT     21H
            JC      CLOFL
            MOV     AH,9
            MOV     DX,OFFSET CLOSEAVL
            INT     21H
            JMP     ENDCL
CLOFL:      MOV     AH,9
            MOV     DX,OFFSET CLOSFL
            INT     21H
ENDCL:      POP     AX
            RET
CLOSFILE    ENDP
;********************** DATA AREA **********************

HANDLE      DW      0000
```

```
;*************************************************************
;********* APPLICATION PROGRAM FRONT END PROCESSOR **********
;*************************************************************
            ;PROGRAM TEST 8251 BOARD 4 PORT
DATA1   EQU     E0H         ;DATA PORT 8251 PORT X 4
PORT1   EQU     E1H         ;CONTROL PORT 8251 X 4
DATA2   EQU     A0H
PORT2   EQU     A1H
DATA3   EQU     C0H
PORT3   EQU     C1H

SETC    EQU     20H
HIASC   EQU     01101111B ;SET 1STP   EVEN CHK 8BIT 64X
BITST   EQU     00010101B ;ENA TX,RX RST ERR

PPIMODE EQU     0BCH        ;MODE 8255 PARALLEL PORT INTERFACE CARD
PORTA   EQU     20H
PORTB   EQU     21H
PORTC   EQU     22H
CONTP   EQU     23H

BLANKMD EQU     81H         ;MODE 8255 MODE 0 MEMORY BLANK
BPORTA  EQU     30H         ;PORT SELECT PAGE
BPORTB  EQU     31H
BPORTC  EQU     32H
BCONTP  EQU     33H

EOF     EQU     '$'
NUL     EQU     00H
SOH     EQU     01H
STX     EQU     02H
ETX     EQU     03H
```

```
EOT       EQU     04H
ENQ       EQU     05H
ACK       EQU     06H
CR        EQU     0DH
LF        EQU     0AH
DLE       EQU     10H
NAK       EQU     15H
SYN       EQU     16H
ETB       EQU     17H
DN_LOAD   EQU     8000H

;************************ END CODE ****************************

TIMEP     EQU     3BH     ;CONTROL PORT TIMER 8253
TMC0      EQU     38H     ;COUNTER PORT 0
TMC1      EQU     39H     ;COUNTER PORT 1
TMC2      EQU     3AH     ;COUNTER PORT 2

;         ==========
          ORG   0000H
;         ==========

;PROGRAM 8253 TIMER PORT
          LD A,00110110B
          OUT (TIMEP),A   ;SETUP CONTROL PORT MODE 3 SQR GEN.
          LD A,100        ;SET TIMING VALUE
          OUT (TMC0),A
          LD A,00H
          OUT (TMC0),A
          LD A,01110101B   ;SETUP CONTROL PORT MODE 2 INT GEN.
          OUT (TIMEP),A
          LD A,67H
```

```
        OUT (TMC1),A
        LD A,15H
        OUT (TMC1),A
        LD A,10111011B  ;SETUP CONTROL PORT MODE 5 HD-TRIG
        OUT (TIMEP),A
        LD A,00H
        OUT (TMC2),A
        LD A,01H
        OUT (TMC2),A

;************* END PROGRAM TIMER PORT ******************

        IM   1              ;PROGRAMMING MODE 1 INTERRUPT
        DI
        ;************* PROGRAM SELECT BLANK
        LD   A,BLANKMD
        OUT  (BCONTP),A
        LD   A,00H          ;SELECT PAGE 0 BLANK
        OUT  (BPORTA),A

        ;************* PROGRAM SERIAL PORT 8251 x 4.....
        LD B,SETC      ;SET COUNTER OF SETUP LOOP
        XOR A               ;CLR DATA
        OUT (PORT1),A
        OUT (PORT2),A
        OUT (PORT3),A
REP:    LD A,40H            ;INTERNAL RESET
        OUT (PORT1),A
        OUT (PORT2),A
        OUT (PORT3),A
        LD A,HIASC          ;SET HIGH SPEED PORT MODE
        OUT (PORT1),A
```

```
            OUT (PORT2),A
            OUT (PORT3),A
            DJNZ REP
            LD   A,BITST
            OUT (PORT1),A
            OUT (PORT2),A
            OUT (PORT3),A

;*..............  END PROGRAM SERIAL PORT ..................

;*************** PROGRAM PPI8255 PARALLEL PORT *************
            LD     A,PPIMODE
            OUT    (CONTP),A      ;PROGRAM MODE HAND_SHAKE
            LD     A,09H          ;SET INTE
            OUT    (CONTP),A

;*************** END PROGRAM ********************************

START:      ;************* START PROGRAMMING **************
            DI
            LD   C,40H      ;SOUND TITLE
            CALL SOUND
            LD   C,20H
            CALL SOUND
            LD   A,0DH      ;READY KEY
            RST  10H
            ;*************** RECEIVE DATA FROM CARD.......
;$$$$$ LOAD FILE1 FROM  $$$$$
RECET:      ;RECEIVE DATA FROM PARALLEL PORT
            XOR  A
            LD   (RECCNT),A
            LD   A,01H              ;START LED
```

```
            LD   (RXFLG),A
            LD   A,06H
            OUT  (02H),A

            LD    HL,8000H
            CALL  LOFL1
            LD    HL,0C000H
            CALL  LOFL2

            ;************** END REC. AND REQ TO SEND.************

            LD   A,0DH              ;READY
            RST  10H

;************************** END SHUNT TEST ****************************
ENDTX:
            XOR  A
            LD   (SIOFLG),A

TEST:       LD B,33         ;SET COUNTER CHARACTOR
            LD IX,ASCTBL    ;ASCII TABLE
            LD A,BITST      ;SET INITIALIZED PORT
            OUT (PORT1),A
            OUT (PORT2),A
            OUT (PORT3),A
PRINT:      LD A,(IX+0)     ;LOAD ASCII CHAR.
            LD (TXDATA),A
            CALL SEND1      ;OUT PORT1
            CALL SEND2      ;OUT PORT2
            CALL SEND3      ;OUT PORT3
            INC IX          ;NEXT CHAR.
            LD   A,06H
```

```
        OUT   (02H),A
        LD    A,(SIOFLG)
        XOR   0FFH          ;SET TX FLAG
        LD    (SIOFLG),A
        OUT   (01H),A
        DJNZ  PRINT


        LD    HL,PROC1      ;LOAD SMPLE TEXT TO SAVE IN QQQ
        LD    (ADDTEXT),HL
        LD    HL,RRR0       ;START LINK_HEAD
        LD    (RLKHEAD),HL
        LD    (RLKCURR),HL
        LD    HL,RRR0+10H
        LD    (RLKTAIL),HL
        ;****************** INPUT PORT 1   >>>>>>>>>>>>
        CALL  LOADQ

        LD    A,0DH
        RST   10H

        ;*****************************************
        CALL  CHKREQ
        ;*****************************************
        CALL  CHKENQ
        ;*****************************************
        CALL  BASELOOP
        JP    START

;TABLE OF CHARACTOR TO PRINT OUT
ASCTBL:     DB 'INITIAL FEP SERIAL COMPLETE****',0AH,0DH
_ACKNOW:    DB 06H,06H,'REQ DATA SENDING BY FEP',04H
```

```
;$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
LOFL1:      PUSH AF
CHKRX:      IN   A,(PORTC)
            BIT  7,A
            LD   A,0A0H          ;SHOW RX LED WAIT STATE
            OUT  (01H),A
            JR   Z,CHKRX
            ;************** CHECK READY TO REXEIVE
            IN   A,(PORTA)       ;******** RECEIVE DATA *********
            LD   (DATAIN),A      ;SAVE IN DUMMY
            LD   (HL),A          ;SAVE IN BLANK MEMORY
            LD   A,(RXELG)       ;SHOW RX MOVING LED
            RLC  A
            LD   (RXFLG),A
            OUT  (01H),A

            LD   A,08H
            OUT  (CONTP),A       ;***** STOP INTE ****
            LD   A,09H
            OUT  (CONTP),A       ;***** START INTE ***
            INC  HL
            LD   (BLKPT),HL      ;SAVE BLANK POINTER
            LD   A,(DATAIN)
            CP   EOF
            JR   NZ,CHKRX        ;JUMP IF RREC. NOT COMPLETE
            LD   A,0FH
            OUT  (01H),A         ;LOAD FLIE 1 COMPLETE
            POP  AF
            RET
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
LOFL2:      PUSH AF
CHKRX2:     IN   A,(PORTC)
```

```
        BIT   7,A
        LD    A,0AH               ;SHOW RX LED WAIT STATE
        OUT   (01H),A
        JR    Z,CHKRX2
        ;************** CHECK READY TO REXEIVE
        IN    A,(PORTA)           ;********* RECEIVE DATA *********
        LD    (DATAIN),A          ;SAVE IN DUMMY
        LD    (HL),A              ;SAVE IN BLANK MEMORY
        LD    A,(RXFLG)           ;SHOW RX MOVING LED
        RLC   A
        LD    (RXFLG),A
        OUT   (01H),A

        LD    A,08H
        OUT   (CONTP),A           ;***** STOP INTE ****
        LD    A,09H
        OUT   (CONTP),A           ;***** START INTE ***
        INC   HL
        LD    (BLKPT),HL          ;SAVE BLANK POINTER
        LD    A,(DATAIN)
        CP    EOF
        JR    NZ,CHKRX2           ;JUMP IF RREC. NOT COMPLETE
        LD    A,0F0H
        OUT   (01H),A             ;LOAD FLIE 1 COMPLETE
        POP   AF
        RET

;^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
SENDET:         LD    B,10H
                LD    HL,_ACKNOW
ANOTHER:        LD    A,05H
                OUT   (CONTP),A
```

```
                LD    A,(HL)
                CP    EOF
                JP    Z,ENDTX
                OUT   (PORTB),A
                LD    A,02H
                LD    (TXFLG),A
                LD    A,06H             ;SELECT FLAG LED
                OUT   (02H),A
CHKTX:          IN    A,(PORTC)         ;CHECK HAND SHAKE
                BIT   6,A
                LD    A,(TXFLG)
                OUT   (01H),A           ;SHOW TX LED
                JP    Z,CHKTX
                LD    A,00H
                OUT   (01H),A           ;SHOW TX LED
                INC   HL
                LD    A,04H
                OUT   (CONTP),A
                JP    ANOTHER
                RET

;******************** USE UTILITY SOUND
SEND1:          PUSH  AF
CHKS1:          IN    A,(PORT1)         ;CHK TXRDY STATUS
                BIT   0,A
                JR    Z,CHKS1
                LD    A,(TXDATA)        ;SEND DATA
                OUT   (DATA1),A
                POP   AF
                RET

SEND2:          PUSH  AF
```

```
CHKS2:      IN A,(PORT2)     ;CHK TXRDY STATUS
            BIT 0,A
            JR Z;CHKS2
            LD A,(TXDATA)    ;SEND DATA
            OUT (DATA2),A
            POP   AF
            RET

SEND3:      PUSH  AF
CHKS3:      IN A,(PORT3)     ;CHK TXRDY STATUS
            BIT 0,A
            JR Z,CHKS3
            LD A,(TXDATA)    ;SEND DATA
            OUT (DATA3),A
            POP   AF
            RET

SOUND:
            PUSH AF
            PUSH BC
            PUSH HL
            LD   A,01H
            LD   B,0FFH
            LD   HL,8000H
            RST  10H
            POP  HL
            POP  BC
            POP  AF
            RET

;>>>>>>>>>>> RECEIVE DATA FROM SERIAL PORT <<<<<<<<<<<<
```

```
REC1:       PUSH AF
            IN   A,(PORT1)
            BIT  1,A
            JR   Z,CHKIN1
            IN   A,(DATA1)
            LD   (ABUFIN),A
            JR   END_REC1
CHKIN1:     LD   A,(RECFLG)
            SET  0,A
            LD   (RECFLG),A
END_REC1:   POP  AF
            RET
            ;...............
REC2:       PUSH AF
CHKIN2:     IN   A,(PORT2)
            BIT  1,A
            JP   Z,CHKIN2
            IN   A,(DATA2)
            LD   (ABUFIN),A
            POP  AF
            RET
            ;...............
REC3:       PUSH AF
CHKIN3:     IN   A,(PORT3)
            BIT  1,A
            JP   Z,CHKIN3
            IN   A,(DATA3)
            LD   (ABUFIN),A
            POP  AF
            RET
;&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
CHKENQ:     PUSH AF
```

```
                PUSH  HL
RECX:           CALL  REC1
                LD    A,(ABUFIN)
                CP    ENQ                    ;CHK ENQ
                JR    NZ,NO_ENQ
                LD    A,ACK
                LD    (TXDATA),A
                CALL  SEND1                  ;SEND ACK
                CALL REC1
                LD    A,(ABUFIN)
                CP    'A'                    ;CHK PREFIX
                JR    NZ,NO_ENQ
                LD    A,ACK
                LD    (TXDATA),A
                CALL  SEND1
                JR    ON_ENQ

NO_ENQ:         LD    A,NAK                  ;IF NOT THEN NAK
                LD    (TXDATA),A
                CALL  SEND1
                JR    RECX
NO_ACK:         LD    A,NAK

ON_ENQ:
                LD    HL,ADDQ1
                LD    (QQQ1),HL
RECC:           CALL  REC1
                LD    A,(ABUFIN)
                LD    HL,(QQQ1)
                LD    (HL),A
                INC   HL
                LD    (QQQ1),HL
```

```
        CP    ETX
        JR    NZ,RECC
        LD    A,ACK
        LD    (TXDATA),A
        CALL  SEND1
        POP   HL
        POP   AF
        RET

;#########################################################################
CHKREQ:     PUSH    AF
            PUSH    HL
            XOR     A
            LD      (RECFLG),A      ;CLEAR RECFLG=00
NO_REQ:     CALL    DELAYL          ;DELAY TIME FOR CHECK
            LD      A,ENQ           ;SEND 'ENQ'
            LD      (TXDATA),A
            CALL    SEND1
            CALL    REC1            ;REC RESPONSE
            LD      A,(RECFLG)
            BIT     0,A
            JP      NZ,NOREC1
            LD      A,(ABUFIN)
            CP      NAK             ;CHECK INPUT NAK
            JR      Z,NO_REQ
            CP      ACK             ;CHECK INPUT ACK
            JR      NZ,NO_REQ
            LD      A,'A'
            LD      (TXDATA),A
            CALL    SEND1           ;SEND 'A'
            CALL    REC1            ;REC RESPONSE
            LD      A,(RECFLG)
```

```
                BIT     0,A
                JP      NZ,NOREC1
                LD      A,(ABUFIN)
                CP      NAK             ;CHECK 'NAK'
                JR      Z,NO_REQ
                CP      ACK             ;CHECK 'ACK'
                JR      NZ,NO_REQ
                LD      HL,RRR1
SNDTXT:
                LD      A,(HL)
                LD      (TXDATA),A
                CALL    SEND1           ;SEND TEXT DATA
                CP      '$'
                JR      Z,ENDSND
                INC     HL
                JR      SNDTXT
ENDSND:         CALL    REC1            ;REC RESPONSE
                LD      A,(RECFLG)
                BIT     0,A
                JP      NZ,NOREC1
                LD      A,(ABUFIN)
                CP      ACK             ;CHECK 'ACK'
                JR      NZ,NO_REQ
                POP     HL
                POP     AF
                RET

NOREC1:         LD      A,(RECFLG)
                RES     0,A
                LD      (RECFLG),A
                JP      NO_REQ
```

```
DELAYL:     PUSH    AF
            PUSH    HL
            LD      HL,8000H
DELAIED:    DEC     DE
            NOP
            LD      A,D
            OR      E
            JR      NZ,DELAIED
            POP     HL
            POP     AF
            RET

LOADQ:      PUSH    AF              ;INPUT DATA IN (ADDTEXT)
            PUSH    BC
            PUSH    HL
            PUSH    DE
            LD      DE,(ADDTEXT)    ;START ADDRESS
            LD      HL,(RLKCURR)        ;DESTINATION ADDRESS
            LD      B,14
TLOAD:      LD      A,(DE)
            LD      (HL),A
            INC     DE
            INC     HL
            DJNZ    TLOAD

ENDLOADT:   LD      (DUMMYS),HL
            LD      HL,(RLKCURR)        ;INC TAIL 1 CELL
            LD      BC,10H
            ADD     HL,BC
            LD      (RLKTAIL),HL
            LD      (DUMMYSS),HL
```

```
                LD      HL,(DUMMYS)
                LD      A,(DUMMYSS)
                LD      (HL),A
                INC     HL
                LD      A,(DUMMYSS+1)
                LD      (HL),A
                POP     DE
                POP     HL
                POP     BC
                POP     AF
                RET
PROC1:  DB 'ARE YOU READY$'
SAMPLE:
        DB ' THIS IS MESSAGE FOR CHECKING INFORMATION FOR DATA TEXT'
        DB ' TERMINAL SERVICE .PLEASE CHECK AT YOUR TERMINAL.....$'
BASELOOP:       PUSH    AF
                XOR     A
                LD      A,(LEDFLG)
                RCL     A
                LD      (LEDFLG),A
                CALL    DELAY
                POP     AF
                RET

;DATA AREA
                ORG  3000H
RECCNT:  DEFS   1
RECFLG:  DEFS   1
BLKPT:   DEFS   2
TXDATA:  DEFS   1
RXDATA:  DEFS   1
DATAIN:  DEFS   1
```

```
TXFLG:     DEFS   1
RXFLG:     DEFS   1
DATAOUT:   DEFS   1
SIOFLG:    DEFS   1
ABUFIN:    DEFS   1
BBUFIN:    DEFS   1
CBUFIN:    DEFS   1
INFLG:     DEFS   1
ADDQ1:     DEFS   2
ADDR1:     DEFS   2
ADDTEXT:   DEFS   2
DUMMYS:    DEFS   2
DUMMYSS:   DEFS   2
               ORG  30F0H
RLKHEAD:   DEFS   2
RLKCURR:   DEFS   2
RLKTAIL:   DEFS   2
QLKHEAD:   DEFS   2
QLKCURR:   DEFS   2
QLKTAIL:   DEFS   2
               ORG  3100H
;R FOR REQUEST TO SEND PROCESS
RRR0:      DEFS   10H
RRR1:      DEFS   10H
RRR2:      DEFS   10H
RRR3:      DEFS   10H
RRR4:      DEFS   10H
RRR5:      DEFS   10H
RRR6:      DEFS   10H
RRR7:      DEFS   10H
RRR8:      DEFS   10H
RRR9:      DEFS   10H
```

```
RRRA:     DEFS   10H
RRRB:     DEFS   10H
RRRC:     DEFS   10H
RRRD:     DEFS   10H
RRRE:     DEFS   10H
RRRF:     DEFS   10H
;Q  SAVE PROCESS
QQQ0:     DEFS   10H
QQQ1:     DEFS   10H
QQQ2:     DEFS   10H
QQQ3:     DEFS   10H
QQQ4:     DEFS   10H
QQQ5:     DEFS   10H
QQQ6:     DEFS   10H
QQQ7:     DEFS   10H
QQQ8:     DEFS   10H
QQQ9:     DEFS   10H
QQQA:     DEFS   10H
QQQB:     DEFS   10H
QQQC:     DEFS   10H
QQQD:     DEFS   10H
QQQE:     DEFS   10H
QQQF:     DEFS   10H

 END
```

```
;*****************************************************
;**********  PROGRAM TEST TERMINAL BOARD  ***********
;**********  INTERFACE KEYPAD,CRT,SERIAL  ***********
;*****************************************************
;*************** Write by WORAPOD ASA.    ***********
;*****************************************************

;DEFINE  PORT DECODE
PADMODE   EQU     8AH           ;KEYBOARD PAD  PORT 8255
KPORTA    EQU     80H
KPORTB    EQU     81H
KPORTC    EQU     82H
KCONTP    EQU     83H

SDATA1    EQU     60H     ;DATA PORT 8251
SPORT1    EQU     61H     ;CONTROL PORT 8251
SETC      EQU     20H
HIASC     EQU     01101111B ;SET 1STP  EVEN CHK 8BIT 64X
BITST     EQU     00010101B ;ENA TX,RX RST ERR

;*********************************************
TXDATA        EQU     3000H
RXDATA        EQU     3001H
PPIDATA       EQU     3002H

;*********************************************
KEYDAT        EQU     3003H
DUMMY         EQU     3004H
DUMMYS        EQU     3005H
CRFLG         EQU     3007H
CHRPOS        EQU     3008H
SCRDAT        EQU     300AH
```

```
LOADCHR     EQU     300BH
CHRDAT      EQU     300DH
MODE_TM     EQU     300EH
SCRFLG      EQU     300FH
FUNCFLG     EQU     3010H
FLFLG       EQU     3011H
;***********************************************
CRTADS      EQU     6000H
SYSSTK      EQU     37FFH
DUM_SCR     EQU     3800H



;************************************************************
            ORG     0000H
;************************************************************
;ORIGIN OF PROGRAM >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
            LD      SP,SYSSTK
            IM      1
            DI
            LD      B,SETC      ;SET COUNTER OF SETUP LOOP
            XOR     A
            OUT     (SPORT1),A
REP:        LD      A,40H       ;INTERNAL RESET
            OUT     (SPORT1),A
            LD      A,HIASC     ;SET HIGH SPEED PORT
            OUT     (SPORT1),A
            DJNZ    REP
            LD      A,BITST     ;SET INITTIALIZED PORT
            OUT     (SPORT1),A
            LD      A,PADMODE   ;PROGRAM KEY PAD PORT
            OUT     (KCONTP),A
```

```
                LD      A,0AAH
                OUT     (KPORTA),A
                CALL     INITC          ;INIT CRT CONTROLLER
;****************** END INITIAL *************************
START:

                CALL CLR_CRT
                CALL TITLE2             ; DISPLAY NAME TITLE
                  CALL DELAY
                CALL TITLE1             ; DISPLAY SUBTITLE
                  CALL  DELAY
                CALL  CLR_CRT1          ;CLEAR SCREEN
                  CALL  DELAY
                XOR     A
                LD      (FUNCFLG),A
READKB:         CALL  CLR_COMM          ;CLEAR OLD COMMAND
NORMKEY:        LD    DE,CRTADS+1780    ;POSITION COMMAND:
READKBD:        IN    A,(KPORTB)        ;INPUT FROM DIP SWITCH
                LD    (MODE_TM),A
                OUT   (KPORTA),A
                CALL  SCANK             ;INPUT COMMAND ' KEY '
                CALL  KDELAY            ;DELAY BROUSE
                CALL  KEYMAP            ;MAP CODE KEY
                LD    A,(CHRDAT)        ;READ CODE TO USE
                LD    (DE),A            ;OUT SCREEN
                CP    7FH               ;CHECK <<<<<FUNCTION KEY>>>>>
                JP    NZ,NOFUNC
                LD    A,(FUNCFLG)
                BIT   0,A
                JR    NZ,READKBD
                LD    A,01H
                LD    (FUNCFLG),A
```

```
        LD    A,'F'              ;****** F *******
        LD    (DE),A
        INC   DE
        LD    A,'U'              ;****** U *******
        LD    (DE),A
        INC   DE
        LD    A,'N'              ;****** N *******
        LD    (DE),A
        INC   DE
        LD    A,'C'              ;****** C *******
        LD    (DE),A
        INC   DE
        JP    READKBD
NOFUNC: CP    '1'                ;FUNCTION 1....
        JR    NZ,NOF1
        LD    A,01H
        LD    (MODEFLG),A
        JP    END_OF_PROC
NOF1:   CP    '2'
        JR    NZ,NOF2
        LD    A,02H
        LD    (MODEFLG),A
        JP    END_OF_PROC
NOF2:   CP    '3'
        JR    NZ,NOF3
        LD    A,03H
        LD    (MODEFLG),A
        JP    END_OF_PROC
NOF3:   CP    '4'
        JR    NZ,NOF4
        LD    A,04H
        LD    (MODEFLG),A
```

```
                JP    END_OF_PROC
NOF4:           CP    '5'
                JR    NZ,NOF5
                LD    A,05H
                LD    (MODEFLG),A
                JP    END_OF_PROC
NOF5:           CP    '6'
                JR    NZ,NOF6
                LD    A,06H
                LD    (MODEFLG),A
                JP    END_OF_PROC
NOF6:           CP    '7'
                JR    NZ,NOF7
                LD    A,07H
                LD    (MODEFLG),A
                JP    END_OF_PROC
NOF7:           CP    '8'
                JR    NZ,NOF8
                LD    A,08H
                LD    (MODEFLG),A
                JP    END_OF_PROC
NOF8:           CP    '9'
                JR    NZ,NOF9
                LD    A,09H
                LD    (MODEFLG),A
                JP    END_OF_PROC
NOF9:           CP    '0'
                JR    NZ,NOF0
                LD    A,00H
                LD    (MODEFLG),A
                JP    END_OF_PROC
NOF0:
```

```
END_OF_PROC:  LD    A,(FUNCFLG)     ;CHECK STATUS FOR FUNCTION KEY
              BIT   0,A
              JP    Z,NORMKEY       ;IF NO FUNCTION GO TO NORMAL KEY
              LD    A,(MODEFLG)
              CP    01H
              JR    NZ,CHF2
              CALL  SAVE_SCR
              CALL  DO_F1
              CALL  SCANK
              CALL  KDELAY
              CALL  LOAD_SCR
              JP    DO_FUNC
CHF2:         BIT   2,A
              JR    Z,CHF3
              CALL  SAVE_SCR
              CALL  DO_F2
              CALL  SCANK
              CALL  LOAD_SCR
              JP    DO_FUNC
CHF3:         BIT   3,A
              JR    Z,CHF4
              CALL  CLR_CRT1
              JP    DO_FUNC
CHF4:         BIT   4,A
              JR    Z,CHF5
              CALL  CLR_CRT1
              CALL  MAIN_MENU
              CALL  SELECTF
              CALL  ENQF

CHF5:
```

```
DO_FUNC:    XOR   A
            LD    (FUNCFLG),A
NOOFFUNC:   JP    READKB              ;END FUNCTION KEY

;************** MAP CODE KEY ***************
KEYMAP:     PUSH  AF
            PUSH  BC
            PUSH  HL
            LD    HL,KTABLE
            LD    A,(KEYDAT)
            LD    C,A
MAPK:       LD    A,(HL)
            CP    C
            INC   HL
            JR    NZ,MAPK
            LD    BC,0010H
            ADD   HL,BC
            DEC   HL
            LD    A,(HL)
            LD    (CHRDAT),A
            POP   HL
            POP   BC
            POP   AF
            RET

KTABLE:     DB    71H, 72H, 74H, 78H
            DB    0B1H,0B2H,0B4H,0B8H
            DB    0D1H,0D2H,0D4H,0D8H
            DB    0E1H,0E2H,0E4H,0E8H
CHRTABLE:   DB    0AH, 1AH, 19H, 1BH
            DB    18H, '9', '8', '7'
```

```
            DB   7FH, '6', '5', '4'
            DB   '0', '3', '2', '1'

;PROGRAM TEST SERIAL PORT
TEST:       PUSH   AF
            PUSH   BC
            PUSH   IX
            LD     A,15H
            OUT    (SPORT1),A
            LD     B,33H          ;SET COUNTER CHARACTOR
            LD     IX,ASCTBL      ;ASCII TABLE
PRINT:      LD     A,(IX+0)       ;LOAD ASCII CHAR.
            LD     (IY+0),A
            INC    IY
            LD     (TXDATA),A
            CALL   SEND1          ;OUT PORT1
            INC    IX             ;NEXT CHAR.
            DJNZ   PRINT          ;LOOP NEXT CHAR.
            POP    IX
            POP    BC
            POP    AF
            RET

SEND1:      IN A,(SPORT1)     ;CHK TXRDY STATUS
            BIT 0,A
            JR Z,SEND1
            LD A,(TXDATA)     ;SEND DATA
            OUT (SDATA1),A
            RET
;TABLE OF CHARACTOR TO PRINT OUT
ASCTBL:DB ' THE QUICK BROWN FOX JUMPS OVER TO THE LAZY DOG',0AH,0DH
```

```
;  ************************************************************
;  ****          SUBROUTINE FOR CLEAR SCREEN              *******
;  ************************************************************
CLR_CRT:
            PUSH    AF
            PUSH    HL
            PUSH    DE
            LD      DE,800H          ; DEFINE 2000 BYTES
            LD      HL,CRTADS
CNO:        LD      A,20H
            LD      (HL),A
            INC     HL
            DEC     DE
            LD      A,D
            OR      E
            JR      NZ,CNO
            POP     DE
            POP     HL
            POP     AF
            RET
;***************************************
;***** CLEAR CRT NOT CLEAR BODY ********
;***************************************

CLR_CRT1:
            PUSH    AF
            PUSH    HL
            PUSH    DE
            LD      DE,1600
            LD      HL,CRTADS
```

```
CNOT:       LD    A,20H
            LD    (HL),A
            INC   HL
            DEC   DE
            LD    A,D
            OR    E
            JR    NZ,CNOT
            POP   DE
            POP   HL
            POP   AF
            RET

;   ************************************************************
;   ***********       DISPLAY TITLE NAME          *************
;   ***********  KMITL :    DATE    TIME         *************
;   ************************************************************
TITLE1: XOR   A
        LD    B,80        ;SET NUMBER OF SENTENSE
        LD    IY,TITLE11
        LD    IX,CRTADS+1600
        CALL  OUT_TEXT
        LD    B,80        ;SET NUMBER OF SENTENSE
        LD    IY,TITLE12
        LD    IX,CRTADS+1680
        CALL  OUT_TEXT
        LD    B,80        ;SET NUMBER OF SENTENSE
        LD    IY,TITLE13
        LD    IX,CRTADS+1760
        CALL  OUT_TEXT
        LD    B,80        ;SET NUMBER OF SENTENSE
        LD    IY,TITLE14
        LD    IX,CRTADS+1840
```

```
        CALL    OUT_TEXT
        RET
TITLE11: DB 08H,' ********** KMITL ** FACULTY OF ENGINEER **INDUSTRIAL COMPUTE
TITLE12: DB     '       DTN       :     DATE 31/08/90      :      TIME:    12:00
TITLE13: DB     '  **** COMMAND :
TITLE14: DB 08H,'<<< STATUS  >>>          IDLE       ACTIVE           DEAD


;*************************************************************
;************      DISPLAY TITLE NAME          **************
;************  PRESS  ENTER TOCONTINEUE        **************
;************           DATA TEXT              **************
;*************************************************************
TITLE2: XOR     A
        LD      B,40        ;SET NUMBER OF SENTENSE
        LD      IY,TITLE21
        LD      IX,CRTADS+580
        CALL    OUT_TEXT

        LD      B,40
        LD      IX,CRTADS+660
        LD      IY,TITLE22
        CALL    OUT_TEXT

        LD      IX,CRTADS+740
        LD      B,40
        LD      IY,TITLE23
        CALL    OUT_TEXT

        LD      B,40
        LD      IX,CRTADS+820
```

```
        LD      IY,TITLE21
        CALL    OUT_TEXT
         RET


TITLE21: DB 0BH,'**************************************',0CH
TITLE22: DB 0BH,'***.. DATA TEXT TERMINAL NETWORK ..***',0CH
TITLE23: DB 0BH,'***..,.... VERSION 1 ACADEMY .....***',0CH


;       ******************************************************
;       ************        DISPLAY MAIN MENU     ************
;       ******************************************************

MAIN_MENU:


        XOR     A
        LD      B,40
        LD      IX,CRTADS+80
        LD      IY,MTEXT
        CALL    OUT_TEXT

        LD      B,40
        LD      IX,CRTADS+240
        LD      IY,ATEXT
        CALL    OUT_TEXT

        LD       B,40
        LD      IX,CRTADS+320
        LD      IY,BTEXT
```

```
        CALL    OUT_TEXT

        LD       B,40
        LD      IX,CRTADS+400
        LD      IY,CTEXT
        CALL    OUT_TEXT

        LD       B,40
        LD      IX,CRTADS+480
        LD      IY,DTEXT
        CALL    OUT_TEXT


        RET
;.....
;********************** TEXT MESSAGE ***************************************
MTEXT:  DB       '                    {{{ DATA TEXT MENU }}}'
ATEXT:  DB      08H,'[1]',0CH,' ###### NEWS & INFORMATION...      '
BTEXT:  DB      08H,'[2]',0CH,' ***<< MARKETING >>***.....        '
CTEXT:  DB      08H,'[3]',0CH,' $$$ FROEIGN EXCHANGE $$$....      '
DTEXT:  DB      08H,'[4]',0CH,' INTERESTING (PRICEs).....         '


;*****************************************
;*********          DELAY TIME FOR DISPLAY
;*****************************************
DELAY:  PUSH    AF
        XOR     A
        LD      B,05H
DELAY1: LD      C,0FFH
DELAY2: LD      D,0FFH
DELAY3: DEC     D
```

```
        JR      NZ,DELAY3
        DEC     C
        JR      NZ,DELAY2
        DJNZ    DELAY1
        POP     AF
        RET

DELAYS: PUSH    AF
        XOR     A
        LD      B,10H
DELAYA: LD      C,FFH
DELAYB: LD      D,FFH
DELAYC: DEC     D
        JR      NZ,DELAYC
        DEC     C
        JR      NZ,DELAYB
        DJNZ    DELAYA
        POP     AF
        RET
;   ***********************************************
;   **********    ROUTINE PRINT SCREEN  ************
;   ***********************************************
OUT_TEXT:
        PUSH     AF
ETEXTL: LD      A,(IY+0)
        LD      (IX+0),A
        INC     IY
        INC     IX
        DJNZ    ETEXTL
        POP     AF
        RET
```

```
;*****************************************************************
;********  SAVE SCREEN INTO DUMMY MEMORY BUFFER        ***********
;*****************************************************************
SAVE_SCR:
        PUSH    AF
        PUSH    DE
        PUSH    HL
        PUSH    BC
        LD      DE,DUM_SCR
        LD      HL,6000H        ; CRT MEM
        LD      BC,1600         ; COUNTER
LOADIN: LD      A,(HL)
        LD      (DE),A
        INC     HL
        INC     DE
        DEC     BC
        LD      A,C
        OR      B
        JR      NZ,LOADIN
        POP     BC
        POP     HL
        POP     DE
        POP     AF
        RET

;*****************************************************************
;********  LOAD SCREEN FROM DUMMY MEMORY BUFFER        ***********
;*****************************************************************
LOAD_SCR:

        PUSH    AF
        PUSH    BC
```

```
        PUSH    DE
        PUSH    HL
        LD      DE,DUM_SCR
        LD      HL,6000H
        LD      BC,1600
LOADOUT: LD     A,(DE)
        LD      (HL),A
        INC     HL
        INC     DE
        DEC     BC
        LD      A,C
        OR      B
        JR      NZ,LOADOUT
        POP     HL
        POP     DE
        POP     BC
        POP     AF
        RET

;***************************************************************
;************        DO FUNCTION KEY              **************
;***************************************************************
DO_F1:  PUSH    AF
        LD      B,20
        LD      IX,CRTADS+345
        LD      IY,F1TEXT0
        CALL    OUT_TEXT
        LD      B,20
        LD      IX,CRTADS+425
        LD      IY,F1TEXT1
        CALL    OUT_TEXT
        LD      B,20
```

```
        LD      IX,CRTADS+505
        LD      IY,F1TEXT2
        CALL    OUT_TEXT
        LD      B,20
        LD      IX,CRTADS+585
        LD      IY,F1TEXT3
        CALL    OUT_TEXT
        LD      B,20
        LD      IX,CRTADS+665
        LD      IY,F1TEXT0
        CALL    OUT_TEXT
        POP     AF
        RET
F1TEXT0:  DB    '********************'
F1TEXT1:  DB    '*  SET TIME/DATE  *'
F1TEXT2:  DB    '*   DATE 26/01/90  *'
F1TEXT3:  DB    '*   TIME 12 : 00   *'

;********** FUNCTION 2 **********************
DO_F2:  PUSH    AF
        LD      B,20
        LD      IX,CRTADS+825
        LD      IY,F2TEXT0
        CALL    OUT_TEXT
        LD      B,20
        LD      IX,CRTADS+905
        LD      IY,F2TEXT1
        CALL    OUT_TEXT
        LD      B,20
        LD      IX,CRTADS+985
        LD      IY,F2TEXT2
        CALL    OUT_TEXT
```

```
        LD      B,20
        LD      IX,CRTADS+1065
        LD      IY,F2TEXT0
        CALL    OUT_TEXT
        POP     AF
        RET

F2TEXT0:  DB    '********************'
F2TEXT1:  DB    '*      SET SOUND     *'
F2TEXT2:  DB    '*      ON    OFF     *'


INITC:  PUSH    AF
        PUSH    HL
        LD      HL,REGTAB
        LD      C,00
        LD      B,10H
CR1:    LD      A,C
        OUT     (0A0H),A
        LD      A,(HL)
        OUT     (0A1H),A
        INC     C
        INC     HL
        DJNZ    CR1
        POP     HL
        POP     AF
        RET
REGTAB: DEFB    6AH
        DEFB    50H
        DEFB    57H
        DEFB    08H
        DEFB    1FH
```

```
        DEFB    03H
        DEFB    18H
        DEFB    1CH
        DEFB    00H
        DEFB    0AH
        DEFB    60H
        DEFB    07H
        DEFB    00
        DEFB    00
        DEFB    00
        DEFB    00
;*****************************************************
;***************     SCAN KEY PAD     ******************
;*****************************************************


SCANK:  PUSH    AF
        PUSH    BC
        PUSH    DE
        PUSH    HL
        XOR     A
        LD      C,KPORTC
COL:    LD      B,11111110B
COL1:   OUT     (C),B           ;OUT C LO
        IN      A,(C)           ;IN  C HI
        CPL
        AND     0F0H            ;DETECT HI BYTE
        CP      00H             ;NO KEYIN
        JR      NZ,KEYIN
        RLC     B
        LD      A,B
        CP      11101111B
```

```
        JR      NZ,COL1
        JR      COL
KEYIN:
        CPL
        AND     0F0H
        LD      C,A     ;SAVE HI BYTE
        LD      A,B
        CPL
        AND     0FH
        LD      B,A     ;SAVE LO BYTE
        OR      C
        LD      (KEYDAT),A      ;SAVE KEYDAT
        POP     HL
        POP     DE
        POP     BC
        POP     AF
        RET
KDELAY: PUSH    AF
        PUSH    DE
        LD      DE,7000H
LOOP:   DEC     DE
        NOP
        LD      A,D
        OR      E
        JR      NZ;LOOP
        POP     DE
        POP     AF
        RET
DATATEXT1:
```

```
        PUSH  AF
        PUSH  BC
        PUSH  DE
        PUSH  HL
        XOR   A
        LD    HL,DATATBL
        LD    (LOADCHR),HL
        LD    DE,CRTADS+160
        LD    (CHRPOS),DE
_LOAD:  LD    HL,(LOADCHR)
        LD    A,(HL)
        LD    (SCRDAT),A
        CP    '$'
        JR    Z,ENDTEXT
        CALL  CHKTXT        ;CHK TEXT  CR,LF
        LD    DE,(CHRPOS)
        LD    A,(SCRFLG)
        CP    01H
        JR    NZ,SCROK
        JR    DONTSCR
SCROK:  LD    A,(SCRDAT)
        LD    (DE),A        ;OUT SCREEN **************>
DONTSCR: INC  DE
        LD    (CHRPOS),DE   ;ANOTHER POS.
        LD    HL,(LOADCHR)
        INC   HL            ;ANOTHER CHAR.
        LD    (LOADCHR),HL
        JR    _LOAD
ENDTEXT: POP  HL
        POP   DE
        POP   BC
        POP   AF
```

```
        RET
DATATBL:
    DB    '.....  SUNDAY 24 FEBRUARY 1991 ',0AH,0DH,'    FROM BANGKOK PO
    DB    '       MILITARY SEIZES POWER ',0AH,0DH
    DB    '            .Chatchai detained ',0AH,0DH
    DB    '            .Martial law imposed',0AH,0DH
    DB    '            .Charter scrapped ',0AH,0DH
    DB    '       Supreme Commander Gen Sunthorn Kongsompong ,supported by the t
    DB    'armed forces chiefs wrested power from the Chatchai Choonhavan Gover
    DB    'in a swift and bloodless takeover.',0AH,0DH
    DB    '       Parliament and the Constitution were disolved and the country
    DB    'under martial law',0AH,0DH
    DB    '       The situation in Bangkok last night was calm with Don Muang in
    DB    'airport remaining open and no reports of disruption to telecommunica
    DB    '                          **************          $'


CHKTXT:   PUSH  AF
          PUSH  BC
          PUSH  HL
          XOR   A
          LD    (SCRFLG),A
          LD    (CRFLG),A      ;CLR FLAG
          LD    A,(SCRDAT)
          CP    0AH            ;CHK CR
          JR    NZ,NO_CR
          LD    A,(CRFLG)
          SET   0,A
          LD    (CRFLG),A
```

```
        JR    COMFLG
NO_CR:  LD    A,(SCRDAT)      ;CHK LF
        CP    0DH
        JR    NZ,ENDPROC      ;NO CR,LF
        LD    A,(CRFLG)
        SET   1,A
        LD    (CRFLG),A

COMFLG: XOR   A
        LD    HL,(CHRPOS)     ;LOAD CHAR. POSITION
        LD    BC,6000H        ;SEEK ABSOLUTE VALUE
        SBC   HL,BC
        XOR   A
        LD    BC,80
SUBS:   LD    (DUMMYS),HL
        SBC   HL,BC
        JR    NC,SUBS
        LD    A,(CRFLG)
        BIT   0,A             ;CHECK CR DO..
        JR    Z,NON_CR
        XOR   A               ;CLR CARRY
        LD    BC,(DUMMYS)
        LD    HL,(CHRPOS)
        SBC   HL,BC
        LD    (CHRPOS),HL     ;CR PROCESS  **********>>>>
        LD    A,01H
        LD    (SCRFLG),A
        JR    ENDPROC
NON_CR: BIT   1,A
        JR    Z,ENDPROC
        LD    HL,(CHRPOS)
        LD    BC,80
```

```
            ADD     HL,BC
            LD      (CHRPOS),HL         ;LF PROCESS *********>>>>>
            LD      A,01H
            LD      (SCRFLG),A
ENDPROC:
            POP     HL
            POP     BC
            POP     AF
            RET

CLR_COMM:   PUSH    AF
            PUSH    HL
            LD      B,50
            LD      HL,CRTADS+1780
CLRR:       LD      A,20H
            LD      (HL),A
            INC     HL
            DJNZ    CLRR
            POP     HL
            POP     AF
            RET

SELECTF:        PUSH    AF
                PUSH    BC
                PUSH    HL
STSL:           LD      B,30
                LD      DE,TSELECT
                LD      HL,CRTADS+1780
PTEXT:          LD      A,(DE)
                LD      (HL),A
                INC     HL
                INC     DE
```

```
        DJNZ    PTEXT
        CALL    SCANK
        CALL    KEYMAP
        LD      A,(CHRDAT)
        CP      '1'
        JR      NZ,NOS1
        LD      A,01H
        LD      (FLFLG),A
        JR      ESEL
NOS1:   CP      '2'
        JR      NZ,NOS2
        LD      A,02H
        LD      (FLFLG),A
        JR      ESEL
NOS2:   CP      '3'
        JR      NZ,NOS3
        LD      A,03H
        LD      (FLFLG),A
        JR      ESEL
NOS3:   CP      '4'
        JR      NZ,NOTFL
        LD      A,04H
        LD      (FLFLG),A
        JR      ESEL
NOTFL:  LD      B,30
        LD      DE,NOTFILE
        LD      HL,CRTADS+1780
NNNN:   LD      A,(DE)
        LD      (HL),A
        INC     DE
        INC     HL
        DJNZ    NNNN
```

```
                CALL    SCANK
                JP      STSL
ESEL:           POP     HL
                POP     DE
                POP     BC
                POP     AF
                RET
TSELECT:        DB      'PLEASE SELECT CHOICE  [1...4]:  '
NOTFILE:        DB      'OUT OF ORDER PSE SELECT AGAIN   '

ENQF:           PUSH    AF      ;LOAD DATATEXT FROM FEP 8000 CHR
                PUSH    BC
                PUSH    DE
                PUSH    HL
                LD      BC,8000
                LD      HL,4000H
                LD      A,(FLFLG)
                POP     HL
                POP     DE
                POP     BC
                POP     AF
                RET

            END
```

LISTING DATABASES PROGRAMMING

(CLIPPER)

```
* Program-ID......: MAIN.PRG
* Purpose.........: Main Menu
* Call from.......: Operating System
* Date Written....:  3-22-91 11:47:33 pm
* Language........: Clipper Summer 87
* Note............:

* memery variable
PRIVATE t_choice

* environment
SET EXCLUSIVE on
SET SCOREBOARD off
SET CONFIRM off

* display fixed text
SET COLOR TO
CLEAR

* displays and performs menu actions.
SET MESSAGE TO 24

* Menu loop
DO WHILE .t.

  * Display menu box
  SET COLOR TO +w/n
  @ 01,01, 23,78 BOX "ฬ๏ฝผฬ๏ฬฬฌ"
  @ 02,02 SAY REPLICATE(' ',76)
  @ 03,01 SAY 'ส'+REPLICATE('ว',76)+'ธ'
  @ 21,01 SAY 'ส'+REPLICATE('ว',76)+'ธ'
  @ 22,02 SAY REPLICATE(' ',76)
```

```
t_msg="Choose by "+chr(26)+" "+chr(27)+" then "+chr(17)+"ี; [Esc] to EXIT!."
@ 22,(80-LEN(t_msg))/2 SAY t_msg
SET COLOR TO w/n,n/w

SET WRAP on

@ 02,03 PROMPT "M.I.S     " MESSAGE "Informations' on business procedure"
@ 02,15 PROMPT "Currency  " MESSAGE "Currency's procedure                "
@ 02,27 PROMPT "Calculator" MESSAGE "Calculator's procedure              "
@ 02,39 PROMPT "Printer   " MESSAGE "Printing report procedure           "
@ 02,51 PROMPT "Utilities " MESSAGE "Utilities' procedure                "
@ 24,00

MENU TO t_choice

IF t_choice = 0          && Escaped out of menu
   EXIT
ELSE
   t_choose = REPLICATE('0',3-LEN(LTRIM(STR(t_choice,0))))+LTRIM(STR(t_choic
   DO menu&t_choose
ENDIF

ENDDO

CLEAR

RETURN
```

```
* Program-ID.....: MENU001.PRG
* Purpose.....: Sub Menu 001
* Call from.....: MAIN.PRG
* Date Written....:  3-22-91 11:48:18 pm
* Language.....: Clipper Summer 87
* Note.....:

* memory variable
PRIVATE t_choice

* environment

* display fixed text

* displays and performs menu actions.
SET MESSAGE TO 24

* Menu loop
DO WHILE .t.

  * Display prg box
  SET COLOR TO +w/n
  @ 04,03, 11,20 BOX "ลวมติวยต "

  t_msg="Choose by "+chr(24)+" "+chr(25)+" then "+chr(17)+"ว; [Esc]
  to EXIT!."
  @ 22,(80-LEN(t_msg))/2 SAY t_msg
  SET COLOR TO w/n,n/w

  SET WRAP on

  @ 05,05 PROMPT "B.O.T. News    " MESSAGE "To day News from Bordcasting
```

```
                                    of Thailand"
    @ 06,05 PROMPT "B.B.C. News    " MESSAGE "To day News from B.B.C. (UK)"
    @ 07,05 PROMPT "C.B.S. News    " MESSAGE "To day News from C.B.S. (USA)"
    @ 08,05 PROMPT "C.N.N. News    " MESSAGE "To day News from C.N.N. (USA)"
    @ 09,05 PROMPT "Rauter's News " MESSAGE "To day News from Rauter
                                    (International) "
    @ 10,05 PROMPT "V.O.A. News    " MESSAGE "To day News from Voice of
                                    America (USA) "
    @ 24,00

    MENU TO t_choice

    IF t_choice=0              && Escaped out of prg
       EXIT
    ELSE
       t_choose=REPLICATE('0',2-LEN(LTRIM(STR(t_choice,0))))+LTRIM
       (STR(t_choice,0))
*      DO prg&t_choose
       DO prg001
    ENDIF

ENDDO

RETURN
```

```
   @ 06,17 PROMPT "Update Datas " MESSAGE "Add/Delete/Edit currency's
                                  data"
   @ 07,17 PROMPT "Index Datas  " MESSAGE "Reindex currency's data          "
   @ 24,00

   MENU TO t_choice

   IF t_choice = 0            && Escaped out of prg
      EXIT
   ELSE
      t_choose = REPLICATE('0',2-LEN(LTRIM(STR(t_choice,0))))+LTRIM
      (STR(t_choice,0))
      DO prg2&t_choose
   ENDIF

ENDDO

RETURN
```

```
* Program-ID.....: MENU003.PRG
* Purpose.....: Sub Menu 003
* Call from.....: MAIN.PRG
* Date Written....:  3-22-91 11:49:05 pm
* Language.....: Clipper Summer 87
* Note.....:

* environment

* display fixed text

DO prg003

RETURN
```

```
* Program-ID.....: MENU004.PRG
* Purpose.....: Sub Menu 004
* Call from.....: MAIN.PRG
* Date Written....:  3-22-91 11:49:22 pm
* Language.....: Clipper Summer 87
* Note.....:

* memory variable
PRIVATE t_choice

* environment

* display fixed text

* displays and performs menu actions.
SET MESSAGE TO 24

* Menu loop
DO WHILE .t.

  * Display prg box
  SET COLOR TO +w/n
  @ 04,39, 07,52 BOX "éúêçúôç "

  t_msg="Choose by "+chr(24)+" "+chr(25)+" then "+chr(17)+"ù;
        [Esc] to EXIT!."
  @ 22,(80-LEN(t_msg))/2 SAY t_msg
  SET COLOR TO w/n,n/w

  SET WRAP on

  @ 05,41 PROMPT "M.I.S.     " MESSAGE "Hardcopy report about Excursion "
```

```
* Program-ID.....: MENU005.PRG
* Purpose.....: Sub Menu 005
* Call from.....: MAIN.PRG
* Date Written....:  3-22-91 11:50:09 pm
* Language.....: Clipper Summer 87
* Note.....:

* Memory variable
PRIVATE t_choice

* Environment

* Display fixed text

* Displays and performs menu actions.
SET MESSAGE TO 24

* Menu loop
DO WHILE .t.

  * Display prg box
  SET COLOR TO +w/n
  @ 04,51, 07,65 BOX "ลวมติวยต "

  t_msg="Choose by "+chr(24)+" "+chr(25)+" then "+chr(17)+"ว; [Esc]
  to EXIT!."
  @ 22,(80-LEN(t_msg))/2 SAY t_msg
  SET COLOR TO w/n,n/w

  SET WRAP on

  @ 05,53 PROMPT "Text Editor" MESSAGE "Built-in Wordprocessor"
```

```
@ 06,53 PROMPT "H E L P !  " MESSAGE "Help message on system"
@ 24,00

MENU TO t_choice

IF t_choice = 0              && Escaped out of prg
   EXIT
ELSE
   t_choose = REPLICATE('0',2-LEN(LTRIM(STR(t_choice,0))))+LTRIM
   (STR(t_choice,0))
   DO prg5&t_choose
ENDIF

ENDDO

RETURN
```

```
* Program-ID.....: MENU401.PRG
* Purpose.....: Sub Menu 001
* Call from.....: MENU004.PRG
*'Date Written....:  3-22-91 11:50:48 pm
* Language.....: Clipper Summer '87
* Note.....:

* memory variable
t_savscr01=SAVESCREEN(06,41, 13,57)
PRIVATE t_choice

* environment

* display fixed text

* displays and performs menu actions.
SET MESSAGE TO 24

* Menu loop
DO WHILE .t.

  * Display prg box
  SET COLOR TO +w/n
  @ 06,41, 13,57 BOX "ลวมติวยต "

  t_msg="Choose by "+chr(24)+" "+chr(25)+" then "+chr(17)+"วิ;
  [Esc] to EXIT!."
  @ 22,(80-LEN(t_msg))/2 SAY t_msg
  SET COLOR TO w/n,n/w

  SET WRAP on
```

```
   @ 07,42 PROMPT "Bangkok       " MESSAGE "Informations about Bangkok
                                   Metropolitant"
   @ 08,42 PROMPT "Cha Am/Hua Hin" MESSAGE "Informations about
                                   Cha Am/Hua Hin       "
   @ 09,42 PROMPT "Chiang-Mai    " MESSAGE "Informations about Chiang-Mai
   @ 10,42 PROMPT "Chiang-Rai    " MESSAGE "Informations about Chiang-Rai
   @ 11,42 PROMPT "Pattaya       " MESSAGE "Informations about Pattaya
   @ 12,42 PROMPT "Phuket        " MESSAGE "Informations about Phuket
   @ 24,00

   MENU TO t_choice

   IF t_choice=0              && Escaped out of prg
      EXIT
   ELSE
      t_choose=REPLICATE('0',2-LEN(LTRIM(STR(t_choice,0))))+LTRIM
      (STR(t_choice,0))
*     DO prg4&t_choose
      DO prg401
   ENDIF

ENDDO

RESTSCREEN(06,41, 13,57, t_savscr01)

RETURN
```

```
* Program-ID.....: MENU402.PRG
* Purpose........: PROGRAM 402
* Call from.....: MENU004.PRG
* Date Written....:  3-23-91  9:12:28 am
* Language.......: Clipper Summer 87
* Note...........:

* Open data
USE customer INDEX ndxnum, ndxnam ALIAS customer
GO BOTTOM

* memory variable
PUBLIC t_get[FCOUNT()] &&Assign array to number of fields
t_field=FCOUNT()
FOR t_count=1 TO t_field
    t_fldname=FIELD(t_count) &&Assign each array to field_name
    t_get[t_count]=SPACE(LEN(&t_fldname))
NEXT t_count
SAVE SCREEN TO t_savscr01
t_keypress=0

* environment

* display fixed text
@ 02,02 CLEAR TO 02,77
@ 04,02 CLEAR TO 20,77
@ 22,02 CLEAR TO 22,77
SET COLOR TO +w/n
@ 19,01 SAY "ú"+REPLICATE("ø",76)+"ù"
t_msg="[PgUp]-Prev   [PgDn]-Next   [Home]-BOF   [End]-EOF
      [F10]-Search"
```

```
@ 20,(80-LEN(t_msg))/2 SAY t_msg
t_msg="[F7]-Print All   [F81]-Print Summary   [ว]-Select print"
@ 22,(80-LEN(t_msg))/2 SAY t_msg
SET COLOR TO w/n,n/w

DO prg002


* Loop for display data
DO WHILE t_keypress=0 &&Do while not key press
   DO prg021
   CLEAR GETS
   t_keypress=INKEY(0)
   DO CASE
      CASE t_keypress=-09 &&[F10]
           DO prg022
      CASE t_keypress= 01 &&[Home]
           GO TOP
      CASE t_keypress= 06 &&[End]
           GO BOTTOM
      CASE t_keypress= 03 &&[PgDn]
           SKIP
           IF EOF()
              TONE(100,020)
              GO BOTTOM
           ENDIF
      CASE t_keypress= 18 &&[PgUp]
           SKIP -1
           IF BOF()
              TONE(100,020)
              GO TOP
           ENDIF
```

```
CASE t_keypress=-06 &&[F7] Print all
     DO WHILE t_keypress=-06 &&Do while press [F7] key
        IF ISPRINTER()
           t_savscr02=SAVESCREEN(11,05, 13,74)
           @ 11,05, 13,74 BOX "ทวนพพำไพ "
           t_msg="Wait! Printting"
           SET COLOR TO *w+/n
           @ 12,(80-LEN(t_msg))/2 SAY t_msg
           SET COLOR TO
           t_recno=RECNO()
           GO TOP
           SET CONSOLE OFF
           SET DEVICE TO PRINT
           DO WHILE .NOT. EOF()
              DO prg420
              SKIP
           ENDDO
           SET DEVICE TO SCREEN
           SET CONSOLE ON
           RESTSCREEN(11,05, 13,74, t_savscr02)
           GO t_recno
        ELSE
           t_savscr02=SAVESCREEN(11,05, 13,74)
           @ 11,05, 13,74 BOX "ทวนพพำไพ "
           t_msg="Error! Printer not ready. Retry? (Y/N)"
           SET COLOR TO *w+/n
           @ 12,(80-LEN(t_msg))/2 SAY t_msg
           SET COLOR TO
           t_keypress=0
           DO WHILE t_keypress=0
              t_keypress=INKEY()
              TONE(50,05)
```

```
            ENDDO
            RESTSCREEN(11,05, 13,74, t_savscr02)
            IF UPPER(CHR(t_keypress))="Y"
               LOOP
            ENDIF
         ENDIF
         EXIT
      ENDDO &&End do while press [F7] key
   CASE t_keypress=-07 &&[F8] Print summary
      DO WHILE t_keypress=-07 &&Do while press [F8] key
         IF ISPRINTER()
            t_recno=RECNO()
            GO TOP
            SET CONSOLE OFF
            SET DEVICE TO PRINT
            DO prg421
            SET DEVICE TO SCREEN
            SET CONSOLE ON
            GO t_recno
         ELSE
            t_savscr02=SAVESCREEN(11,05, 13,74)
            @ 11,05, 13,74 BOX "ทานพพาไพ "
            t_msg="Error! Printer not ready. Retry? (Y/N)"
            SET COLOR TO *w+/n
            @ 12,(80-LEN(t_msg))/2 SAY t_msg
            SET COLOR TO
            t_keypress=0
            DO WHILE t_keypress=0
               t_keypress=INKEY()
               TONE(50,05)
            ENDDO
            RESTSCREEN(11,05, 13,74, t_savscr02)
```



```
            IF UPPER(CHR(t_keypress))="Y"
```

```
            ENDIF
         ENDIF
         EXIT
      ENDDO &&End do while press [F8] key
   CASE t_keypress= 13 &&[Return] Print selected
      DO WHILE t_keypress= 13 &&Do while selected data
         IF ISPRINTER()
            SET CONSOLE OFF
            SET DEVICE TO PRINT
            DO prg420
            SET CONSOLE ON
            SET DEVICE TO SCREEN
         ELSE
            t_savscr02=SAVESCREEN(11,05, 13,74)
            @ 11,05, 13,74 BOX "กวนพพวไพ "
            t_msg="Error! Printer not ready. Retry? (Y/N)"
            SET COLOR TO *w+/n
            @ 12,(80-LEN(t_msg))/2 SAY t_msg
            SET COLOR TO
            t_keypress=0
            DO WHILE t_keypress=0
               t_keypress=INKEY()
               TONE(50,05)
            ENDDO
            RESTSCREEN(11,05, 13,74, t_savscr02)
            IF UPPER(CHR(t_keypress))="Y"
               LOOP
            ENDIF
         ENDIF
         EXIT
      ENDDO &&End do while selected data
   CASE t_keypress= 27 &&[Esc]
```

```
    ENDCASE
    t_keypress=0
ENDDO &&Enddo while not key press

*Close data
CLOSE DATABASE

RESTORE SCREEN FROM t_savscr01

RETURN
```

```
* Program-ID.....: PRG001.PRG
* Purpose........: PROGRAM 001
* Call from.....: MENU001.PRG
* Date Written....:  3-22-91 11:51:04 pm
* Language.......: Clipper Summer 87
* Note...........:

* memory variable
PRIVATE t_choice
t_savscr01=SAVESCREEN(06,25, 18,57)

* environment

* display fixed text

* displays and performs menu actions.
SET MESSAGE TO 24

* Menu loop
DO WHILE .t.

  * Display prg box
  @ 06,25, 18,57 BOX "ๆ๏ษตภ๏ไต "
  SET COLOR TO +w/n

  t_msg="Choose by "+chr(24)+" "+chr(25)+" then "+chr(17)+"ว;
  [Esc] to EXIT!."
  @ 22,(80-LEN(t_msg))/2 SAY t_msg
  SET COLOR TO w/n,n/w

  SET WRAP on
```

```
@ 07,27 PROMPT "Business' News          " MESSAGE "News about
                                          Business around the world "
@ 08,27 PROMPT "Economics News          " MESSAGE "News about
                                          Economics around the world"
@ 09,27 PROMPT "Entertainments News     " MESSAGE "News about
                                          Entertain, Movies, Songs  "
@ 10,27 PROMPT "Exclusive News            " MESSAGE "News about
                                            Exclusive                "
@ 11,27 PROMPT "Executive's News          " MESSAGE "News about
                                            Executive                "
@ 12,27 PROMPT "Forieng News              " MESSAGE "News from
                                            Abroad                   "
@ 13,27 PROMPT "Goverments' News          " MESSAGE "News from
                                            Goverments               "
@ 14,27 PROMPT "Local's News              " MESSAGE "News about
                                            Loval Area               "
@ 15,27 PROMPT "Politicals' News          " MESSAGE "News about
                                            Political around the World"
@ 16,27 PROMPT "Stock & Share Markes News " MESSAGE "News about
                                            Stock & Share Markets    "
@ 17,27 PROMPT "World Wide News           " MESSAGE "News around
                                            the World                ".
@ 24,00

MENU TO t_choice

IF t_choice = 0             && Escaped out of prg
   EXIT
ELSE
   SAVE SCREEN TO t_savscr02

   t_file="TEXT"+REPLICATE('0',3-LEN(LTRIM(STR(t_choice,0))))+LTRIM
```

```
      (STR(t_choice,0))
      SET COLOR TO +w/n
      @ 01,01, 23,78 BOX "วมติวยต "
      @ 21,01 SAY 'ล'+REPLICATE('ว',76)+'ถ'
      t_msg="Scroll by "+chr(24)+" "+chr(25)+" or [PgUp] [PgDn];
      [Esc] to EXIT!."
      @ 22,(80-LEN(t_msg))/2 SAY t_msg
      SET COLOR TO w/n,n/w

      IF .NOT. FILE("&t_file")
         @ 00,00 SAY "Cannot open file "+t_file
      ENDIF

      MEMOEDIT(MEMOREAD(t_file),02,02,20,77,.f.)

      RESTORE SCREEN FROM t_savscr02
   ENDIF

ENDDO

RESTSCREEN(06,25, 18,57,t_savscr01)

RETURN
```

```
* Program-ID.....: PRG002.PRG
* Purpose.....: Program 002
* Call from.....: PRG201.PRG
* Date Written....:  3-22-91 11:51:30 pm
* Language.....: Clipper Summer 87
* Note.....:

* memory variable

* environment

* display fixed text
t_msg="Currency"
@ 02,(80-LEN(t_msg))/2 SAY t_msg
@ 04,05 SAY "Interest Rate "
@ 05,10 SAY "USA Prime Rate      "
@ 06,10 SAY "THI Inter-Bank Loan   "
@ 07,10 SAY "Over Draft Loan (MLR) "
@ 08,10 SAY "Dued-dated Loan (MLR) "
@ 09,05 SAY "Currency Rate       Sold    Buy"
@ 10,10 SAY "US $          "
@ 11,10 SAY "UK            "
@ 12,10 SAY "Mark Germany  "
@ 13,10 SAY "  (1:100 )    "
@ 14,10 SAY "Malasia $     "
@ 15,10 SAY "Singapore $   "
@ 16,10 SAY "Hong Kong $   "
@ 17,10 SAY "Swiss         "
@ 18,10 SAY "China (Yuans) "
@ 05,40 SAY "Gold                  Sold    Buy"
@ 06,45 SAY "Cubic       "
@ 07,45 SAY "Hand maded "
```

```
@ 18,50 SAY "Reported on "

RETURN
```

```
ENDCASE

RETURN


FUNCTION calc

* memory variable
PRIVATE e_n_t_r_y,f_c_o_d_e,m_c_o_d_e,calc_d_e_c,d_e_c,a_c_c_u_m,
        e_n_t_e_r
PRIVATE d_e_c_cnt,r_o_w,c_o_l,calc_scr,pc_o_u_n_t,i_n_k_e_y,c_o_d_e
PRIVATE calc_c_l_r,row_s_a_v_e,col_s_a_v_e,proc_name

PUBLIC calc_m_e_m,calc_a_c_c,calc_d_e_c

PARAMETERS c_m_e_n_u,c_o_l

pc_o_u_n_t=PCOUNT()
row_s_a_v_e=ROW()
col_s_a_v_e=COL()

* environment
SET SCOREBOARD OFF
SET BELL OFF
SET DECIMALS TO 2

IF TYPE('calc_m_e_m')<>'N'
   calc_m_e_m=0
ENDIF
IF TYPE('calc_a_c_c')<>'N'
   calc_a_c_c=0
ENDIF
```

```
IF TYPE('calc_d_e_c')<>'N'
   calc_d_e_c=0
ENDIF
DO CASE
   CASE pc_o_u_n_t=0
        c_m_e_n_u=.t.
   CASE pc_o_u_n_t=1
        c_o_l=40
ENDCASE
IF c_m_e_n_u
   c_o_l=40
ENDIF
IF c_o_l>45
   c_o_l=40
ENDIF
SAVE SCREEN TO calc_scr
calc_c_l_r='w+'
IF TYPE('color_save')='C'
   calc_c_l_r=color_save
ENDIF
COLOR('BG+')
DO explode__ WITH 2,c_o_l,23,c_o_l+35,'BG+','BG+',.t.
@ 02,c_o_l TO 23,c_o_l+35 DOUBLE
@ 04,c_o_l+5 TO 06,c_o_l+30 DOUBLE
@ 19,c_o_l+12 TO 21,c_o_l+30
@ 20,c_o_l+5 SAY 'Memory'
r_o_w=7
DO WHILE r_o_w<17
   @ r_o_w,c_o_l+5 TO r_o_w+2,c_o_l+8
   @ r_o_w,c_o_l+10 TO r_o_w+2,c_o_l+13
   @ r_o_w,c_o_l+14 TO r_o_w+2,c_o_l+17
   @ r_o_w,c_o_l+18 TO r_o_w+2,c_o_l+21
```

```
    @ r_o_w,c_o_l+23 TO r_o_w+2,c_o_l+26
    @ r_o_w,c_o_l+27 TO r_o_w+2,c_o_l+30
    r_o_w=r_o_w+3
ENDDO
COLOR('GR+')
@ 08,c_o_l+6 SAY 'AC'
@ 08,c_o_l+11 SAY ' 7'
@ 08,c_o_l+15 SAY ' 8'
@ 08,c_o_l+19 SAY ' 9'
@ 08,c_o_l+24 SAY ' /'
@ 08,c_o_l+28 SAY ' R'
@ 11,c_o_l+6 SAY 'MR'
@ 11,c_o_l+11 SAY ' 4'
@ 11,c_o_l+15 SAY ' 5'
@ 11,c_o_l+19 SAY ' 6'
@ 11,c_o_l+24 SAY ' *'
@ 11,c_o_l+28 SAY ' %'
@ 14,c_o_l+6 SAY 'M-'
@ 14,c_o_l+11 SAY ' 1'
@ 14,c_o_l+15 SAY ' 2'
@ 14,c_o_l+19 SAY ' 3'
@ 14,c_o_l+24 SAY ' -'
@ 14,c_o_l+28 SAY 'CE'
@ 17,c_o_l+6 SAY 'M+'
@ 17,c_o_l+11 SAY ' 0'
@ 17,c_o_l+15 SAY ' .'
@ 17,c_o_l+19 SAY ' ='
@ 17,c_o_l+24 SAY ' +'
@ 17,c_o_l+28 SAY 'C '
IF c_m_e_n_u
   COLOR('BG+')
   @ 2,3 TO 23,38 DOUBLE
```



```
   COLOR('W+/BG')
```

```
      @ 03,04 CLEAR TO 22,37
      @ 05,05 SAY 'AC = Clear All'
      @ 06,05 say 'MC = Clear Memory'
      @ 07,05 say 'MR = Recall Memory'
      @ 08,05 say 'M+ = Add Total to Memory'
      @ 09,05 say 'M- = Sub Total from Memory'
      @ 10,05 say 'C<ENTER> = Clear Total & Entry'
      @ 11,05 say 'CE<ENTER> = Clear Entry'
      @ 13,05 say 'Use <Enter> for <=>'
      @ 15,05 say ' / = Divide'
      @ 16,05 say ' * = Multiply'
      @ 17,05 say ' - = Subtract'
      @ 18,05 say ' + = Add'
      @ 19,05 say ' R = Square Root'
      @ 20,05 say ' % = Percentage'
      @ 21,05 SAY ' X = EXIT'
   ENDIF
   @ 22,c_o_l+14 SAY ' X = EXIT '
   e_n_t_r_y=0
   calc_d_e_c=0
   d_e_c=.F.
   f_c_o_d_e='+'
   c_o_d_e=' '
   a_c_c_u_m=.T.
   e_n_t_e_r=.T.
   i_n_k_e_y=13
   m_c_o_d_e=' '
   DO WHILE .T.
      proc_name='CALCCLIP'
      help_code=' SOFT0200'
      COLOR('GR+/RB,gr+/rb')

      @ 20,c_o_l+15 SAY calc_m_e_m PICT '99999999999.99'
      COLOR('GR+/R,N/W+')
```

```
IF a_c_c_u_m
   @ 5,c_o_l+7 SAY 'T'
   @ 5,c_o_l+13 SAY calc_a_c_c PICT '99999999999.99'
ELSE
   @ 5,c_o_l+7 SAY f_c_o_d_e
   @ 5,c_o_l+13 SAY e_n_t_r_y+calc_d_e_c PICT '99999999999.99'
ENDIF
DISP_TIME(0,0,'i_n_k_e_y','c_o_d_e',.f.)
DO CASE
   CASE c_o_d_e>='0' .AND. c_o_d_e<='9'
        IF e_n_t_e_r .OR. f_c_o_d_e=' '
           calc_a_c_c=0
           e_n_t_r_y=0
           calc_d_e_c=0
           f_c_o_d_e='+'
        ENDIF
        a_c_c_u_m=.F.
        IF .NOT. d_e_c
           e_n_t_r_y=e_n_t_r_y*10+VAL(c_o_d_e)
        ELSE
           IF d_e_c_cnt<1000
              calc_d_e_c=calc_d_e_c+VAL(c_o_d_e)/d_e_c_cnt
              d_e_c_cnt=d_e_c_cnt*10
           ENDIF
        ENDIF
   CASE c_o_d_e='.'
        a_c_c_u_m=.F.
        d_e_c=.T.
        d_e_c_cnt=10
   CASE (c_o_d_e='+' .OR. c_o_d_e='-' .OR. c_o_d_e='/' .OR.
        c_o_d_e='*' .OR. c_o_d_e='R' .OR. c_o_d_e='%' .OR.
        i_n_k_e_y=13) .AND. m_c_o_d_e=' '
        a_c_c_u_m=.T.
```

```
IF i_n_k_e_y<>13 .AND. e_n_t_e_r
ELSE
   DO CASE
      CASE f_c_o_d_e='+'
           IF c_o_d_e='%'
              calc_a_c_c=calc_a_c_c+calc_a_c_c*.01*
              (e_n_t_r_y+calc_d_e_c)
           ELSE
              calc_a_c_c=calc_a_c_c+(e_n_t_r_y+calc_d_e_c)
           ENDIF
      CASE f_c_o_d_e='-'
           IF c_o_d_e='%'
              calc_a_c_c=calc_a_c_c-calc_a_c_c*.01*
              (e_n_t_r_y+calc_d_e_c)
           ELSE
              calc_a_c_c=calc_a_c_c-(e_n_t_r_y+calc_d_e_c)
           ENDIF
      CASE f_c_o_d_e='*'
           calc_a_c_c=calc_a_c_c*(e_n_t_r_y+calc_d_e_c)
      CASE f_c_o_d_e='/'
           calc_a_c_c=calc_a_c_c/(e_n_t_r_y+calc_d_e_c)
      CASE f_c_o_d_e='R'
           calc_a_c_c=SQRT(calc_a_c_c)
   ENDCASE
ENDIF
d_e_c=.F.
IF c_o_d_e='%'
   f_c_o_d_e=' '
ENDIF
```

```
IF i_n_k_e_y<>13
   f_c_o_d_e=c_o_d_e
   e_n_t_r_y=0
   calc_d_e_c=0
```

```
      ENDIF
CASE c_o_d_e='A'
      m_c_o_d_e='A'
CASE c_o_d_e='M'
      m_c_o_d_e='M'
CASE c_o_d_e='C' .AND. m_c_o_d_e=' '
      m_c_o_d_e='C'
CASE c_o_d_e='C' .AND. m_c_o_d_e='A'
      a_c_c_u_m=.T.
      calc_a_c_c=0
      calc_d_e_c=0
      e_n_t_r_y=0
      calc_m_e_m=0.00
      m_c_o_d_e=' '
      f_c_o_d_e=' '
CASE c_o_d_e='C' .AND. m_c_o_d_e='M'
      calc_m_e_m=0.00
      m_c_o_d_e=' '
CASE c_o_d_e='E' .AND. m_c_o_d_e='C'
      e_n_t_r_y=0
      calc_d_e_c=0
      d_e_c=.F.
      m_c_o_d_e=' '
      a_c_c_u_m=.F.
      f_c_o_d_e=' '
CASE m_c_o_d_e='C' .AND. i_n_k_e_y=13
      e_n_t_r_y=0
      calc_d_e_c=0
```

```
      calc_a_c_c=0
      d_e_c=.F.
      m_c_o_d_e=' '
      f_c_o_d_e=' '
      a_c_c_u_m=.T.
```

```
      CASE c_o_d_e='R' .AND. m_c_o_d_e='M'
           e_n_t_r_y=calc_m_e_m
           a_c_c_u_m=.F.
           d_e_c=.F.
           m_c_o_d_e=' '
      CASE c_o_d_e='-' .AND. m_c_o_d_e='M'
           calc_m_e_m=calc_m_e_m-calc_a_c_c
           m_c_o_d_e=' '
           calc_d_e_c=0
           e_n_t_r_y=0
           f_c_o_d_e=' '
           a_c_c_u_m=.T.
      CASE c_o_d_e='+' .AND. m_c_o_d_e='M'
           calc_m_e_m=calc_m_e_m+calc_a_c_c
           m_c_o_d_e=' '
           calc_d_e_c=0
           e_n_t_r_y=0
           f_c_o_d_e=' '
           a_c_c_u_m=.T.
      CASE c_o_d_e='X' .OR. i_n_k_e_y=27
           REST SCREEN FROM calc_scr
           COLOR(calc_c_l_r)
           SET CURSOR ON
           @ row_s_a_v_e,col_s_a_v_e SAY ''
           RETURN calc_a_c_c
      CASE .T.
           ? CHR(7)



      ENDCASE
      e_n_t_e_r=.f.
      IF i_n_k_e_y=13
        e_n_t_e_r=.T.
      ENDIF
    ENDDO
```

```
* Program-ID.....: PRG021.PRG
* Purpose.....: Program 021
* Call from.....: PRG201.PRG
* Date Written....:  3-22-91 11:53:00 pm
* Language.....: Clipper Summer 87
* Note.....:

* memory variable

* environment

* display fixed text
IF DELETE()
   t_msg=" DELETED! "
   SET COLOR TO *n/+w
ELSE
   t_msg="          "
ENDIF
@ 04,50 SAY t_msg
SET COLOR TO w/n,n/w

* Process get data
@ 05,33 GET currency->usa_prim PICTURE "@Z 999.99"
@ 06,33 GET currency->thi_bank PICTURE "@Z 999.99"
@ 07,33 GET currency->thi_mors PICTURE "@Z 999.99"
@ 08,33 GET currency->thi_mlrs PICTURE "@Z 999.99"
@ 10,25 GET currency->usa_doll_s PICTURE "@Z 999.99"
@ 10,33 GET currency->usa_doll_b PICTURE "@Z 999.99"
@ 11,25 GET currency->uk_pound_s PICTURE "@Z 999.99"
@ 11,33 GET currency->uk_pound_b PICTURE "@Z 999.99"
@ 12,25 GET currency->ger_mark_s PICTURE "@Z 999.99"
@ 12,33 GET currency->ger_mark_b PICTURE "@Z 999.99"
```

```
@ 13,25 GET currency->jap_yens_s PICTURE "@Z 999.99"
@ 13,33 GET currency->jap_yens_b PICTURE "@Z 999.99"
@ 14,25 GET currency->mal_doll_s PICTURE "@Z 999.99"
@ 14,33 GET currency->mal_doll_b PICTURE "@Z 999.99"
@ 15,25 GET currency->sin_doll_s PICTURE "@Z 999.99"
@ 15,33 GET currency->sin_doll_b PICTURE "@Z 999.99"
@ 16,25 GET currency->hks_doll_s PICTURE "@Z 999.99"
@ 16,33 GET currency->hks_doll_b PICTURE "@Z 999.99"
@ 17,25 GET currency->sws_fran_s PICTURE "@Z 999.99"
@ 17,33 GET currency->sws_fran_b PICTURE "@Z 999.99"
@ 18,25 GET currency->chn_yuan_s PICTURE "@Z 999.99"
@ 18,33 GET currency->chn_yuan_b PICTURE "@Z 999.99"
@ 06,65 GET currency->cub_gold_s PICTURE "@Z 999.99"
@ 06,72 GET currency->cub_gold_b PICTURE "@Z 999.99"
@ 07,65 GET currency->hnd_gold_s PICTURE "@Z 999.99"
@ 07,72 GET currency->hnd_gold_b PICTURE "@Z 999.99"
@ 18,62 GET currency->rpt_date

RETURN
```

```
* Program-ID.....: PRG022.PRG
* Purpose.....: Program 022
* Call from.....: PRG201.PRG
* Date Written.....:  3-22-91 11:53:13 pm
* Language.....: Clipper Summer 87
* Note.....:

* Memory variable
DECLARE t_bar[FCOUNT()] &&Assign array to number of fields
PRIVATE t_choice
t_field=FCOUNT()
FOR t_count=1 TO t_field
    t_bar[t_count]=STUFF(SPACE(20),1,LEN(FIELD(t_count)),FIELD
    (t_count)) &&Assign each array to field_name
NEXT t_count
t_savscr02=SAVESCREEN(09,27, 17,50)
t_keypress=0
t_search=SPACE(120)
t_recno=RECNO()

* Environment

* Display fixed text
@ 09,27, 17,50 BOX "ÉÍ»ºÈ¼ÍÌº "
t_msg="ÛÛ Search Items ÛÛ"
@ 09,31 SAY t_msg
@ 15,27 SAY "Ú"+REPLICATE("Ä",22)+"¿"
t_msg="  Use   then Ù   "
@ 16,28 SAY t_msg

* Get search item
DO WHILE .t.
```

```
t_choice=ACHOICE(10,29, 14,48, t_bar)
IF t_choice=0
   EXIT
ELSE
   t_savscr03=SAVESCREEN(11,02, 13,77)
   @ 11,02 TO 13,77
   @ 12,03 GET t_search PICTURE "@S74"
   READ
   RESTSCREEN(11,02, 13,77, t_savscr03)
   IF READKEY()=12 &&[Esc]
      LOOP
   ELSE
      t_lookup=t_bar[t_choice]
      LOCATE FOR &t_lookup=TRIM(t_search)
      IF FOUND()
         EXIT
      ELSE
         GO t_recno
         t_savscr03=SAVESCREEN(11,02, 13,77)
         @ 11,02 TO 13,77
         t_msg="Search specified item not found!"
         t_msg=STUFF(SPACE(120),1,LEN(t_msg),t_msg)
         @ 12,03 GET t_msg PICTURE "@S74"
         CLEAR GETS
         FOR t_count=1 TO 20
             TONE(5000,3)
             TONE(2000,5)
         NEXT
         RESTSCREEN(11,02, 13,77, t_savscr03)
      ENDIF
   ENDIF
ENDIF
```

```
ENDDO

RESTSCREEN(09,27, 17,50, t_savscr02)

RETURN
```

```
* Program-ID.....: PRG023.PRG
* Purpose.....: Program 023
* Call from.....: PRG202.PRG
* Date Written....:  3-22-91 11:53:27 pm
* Language.....: Clipper Summer 87
* Note.....:

* memory variable
t_msg="            "

* environment

* display fixed text
@ 04,05 SAY t_msg

* Process get data
@ 04,50 GET t_gets[01] PICTURE "@Z 999.99"
@ 05,50 GET t_gets[02] PICTURE "@Z 999.99"
@ 06,50 GET t_gets[03] PICTURE "@Z 999.99"
@ 07,50 GET t_gets[04] PICTURE "@Z 999.99"
@ 09,25 GET t_gets[05] PICTURE "@Z 999.99"
@ 09,33 GET t_gets[06] PICTURE "@Z 999.99"
@ 10,25 GET t_gets[07] PICTURE "@Z 999.99"
@ 10,33 GET t_gets[08] PICTURE "@Z 999.99"
@ 11,25 GET t_gets[09] PICTURE "@Z 999.99"
@ 11,33 GET t_gets[10] PICTURE "@Z 999.99"
@ 12,25 GET t_gets[11] PICTURE "@Z 999.99"
@ 12,33 GET t_gets[12] PICTURE "@Z 999.99"
@ 13,25 GET t_gets[13] PICTURE "@Z 999.99"
@ 13,33 GET t_gets[14] PICTURE "@Z 999.99"
@ 14,25 GET t_gets[15] PICTURE "@Z 999.99"
```

```
@ 14,33 GET t_gets[16] PICTURE "@Z 999.99"
@ 15,25 GET t_gets[17] PICTURE "@Z 999.99"
@ 15,33 GET t_gets[18] PICTURE "@Z 999.99"
@ 16,25 GET t_gets[19] PICTURE "@Z 999.99"
@ 16,33 GET t_gets[20] PICTURE "@Z 999.99"
@ 17,25 GET t_gets[21] PICTURE "@Z 999.99"
@ 17,33 GET t_gets[22] PICTURE "@Z 999.99"
@ 19,25 GET t_gets[23] PICTURE "@Z 999.99"
@ 19,33 GET t_gets[24] PICTURE "@Z 999.99"
@ 20,25 GET t_gets[25] PICTURE "@Z 999.99"
@ 20,33 GET t_gets[26] PICTURE "@Z 999.99"
@ 20,50 GET t_gets[27]

RETURN
```

```
* Program-ID.....: PRG201.PRG
* Purpose.....: Program 201
* Call from.....: MENU002.PRG
* Date Written....:  3-22-91 11:53:42 pm
* Language.....: Clipper Summer 87
* Note.....:

* memory variable
SAVE SCREEN TO t_savscr01
t_keypress=0

* environment

* display fixed text
@ 02,02 CLEAR TO 02,77
@ 04,02 CLEAR TO 20,77
@ 22,02 CLEAR TO 22,77
t_msg="[PgUp]-Prev   [PgDn]-Next   [Home]-First   [End]-Last
[F10]-Search"
@ 22,(80-LEN(t_msg))/2 SAY t_msg

DO prg002

* Open data
USE currency INDEX ndxdate ALIAS currency
GO TOP

* Loop for display data
DO WHILE t_keypress=0 &&Do while not key press
   DO prg021
   CLEAR GETS
   t_keypress=INKEY(0)
```

```
    DO CASE
       CASE t_keypress=-09 &&[F10]
            DO prg022
       CASE t_keypress= 01 &&[Home]
            GO TOP
       CASE t_keypress= 06 &&[End]
            GO BOTTOM
       CASE t_keypress= 03 &&[PgDn]
            SKIP
            IF EOF()
               TONE(100,020)
               GO BOTTOM
            ENDIF
       CASE t_keypress= 18 &&[PgUp]
            SKIP -1
            IF BOF()
               TONE(100,020)
               GO TOP
            ENDIF
       CASE t_keypress= 27 &&[Esc]
            EXIT
    ENDCASE
    t_keypress=0
ENDDO &&Enddo while not key press

*Close data
CLOSE DATABASE

RESTORE SCREEN FROM t_savscr01

RETURN
```

```
* Program-ID.....: PRG202.PRG
* Purpose.....: Program 202
* Call from.....: MENU002.PRG
* Date Written....:  3-22-91 11:53:56 pm
* Language.....: Clipper Summer 87
* Note.....:

* Open data
USE customer INDEX ndxnum, ndxnam ALIAS customer
GO BOTTOM

* memory variable
PUBLIC t_get[FCOUNT()] &&Assign array to number of fields
t_field=FCOUNT()
FOR t_count=1 TO t_field
    t_fldname=FIELD(t_count) &&Assign each array to field name
    t_get[t_count]=SPACE(LEN(&t_fldname))
NEXT t_count
SAVE SCREEN TO t_savscr01
t_keypress=0

* environment

* display fixed text
@ 02,02 CLEAR TO 02,77
@ 04,02 CLEAR TO 20,77
@ 22,02 CLEAR TO 22,77
SET COLOR TO +w/n
@ 19,01 SAY "ร"+REPLICATE("ฬ",76)+"น"
t_msg="[PgUp]-Prev   [PgDn]-Next   [Home]-BOF   [End]-EOF
       [F10]-Search"
@ 20,(80-LEN(t_msg))/2 SAY t_msg
```

```
t_msg="[Ins]-Append   [Del]-Delete/Recall   [↵]-Select Update"
@ 22,(80-LEN(t_msg))/2 SAY t_msg
SET COLOR TO w/n,n/w

DO prg002



* Loop for display data
DO WHILE t_keypress=0 &&Do while not key press
   DO prg021
   CLEAR GETS
   t_keypress=INKEY(0)
   DO CASE
      CASE t_keypress=-09 &&[F10]
         DO prg022
      CASE t_keypress= 01 &&[Home]
           GO TOP
      CASE t_keypress= 06 &&[End]
           GO BOTTOM
      CASE t_keypress= 03 &&[PgDn]
           SKIP
           IF EOF()
              TONE(100,020)
              GO BOTTOM
           ENDIF
      CASE t_keypress= 18 &&[PgUp]
           SKIP -1
           IF BOF()
              TONE(100,020)
              GO TOP
           ENDIF
      CASE t_keypress= 22 &&[Ins]
```

```
DO WHILE t_keypress= 22 &&Do while press insert key
   DO prg023
   READ
   t_savscr03=SAVESCREEN(11,02, 13,77)
   IF READKEY()=12
      t_msg="Add data CANCEL!"
      t_time=03
      t_tone01=5000
      t_dura01=03
      t_tone02=2000
      t_dura02=20
   ELSE
      APPEND BLANK
      FOR t_count=1 TO t_field
         t_fldname=FIELD(t_count)
         REPLACE &t_fldname WITH t_get[t_count]
      NEXT t_count
      COMMIT
      t_msg="Add data COMPLETE!"
      t_time=01
      t_tone01=40
      t_dura01=10
      t_tone02=50
      t_dura02=20
   ENDIF
   @ 11,02 TO 13,77
   t_msg=STUFF(SPACE(120),1,LEN(t_msg),t_msg)
   @ 12,03 GET t_msg PICTURE "@S74"
   CLEAR GETS
   FOR t_count=1 TO t_time
      TONE(t_tone01,t_dura01)
      TONE(t_tone02,t_dura02)
```

```
            NEXT
            RESTSCREEN(11,02, 13,77, t_savscr03)
            EXIT
         ENDDO &&End do while press insert key
      CASE t_keypress= 07 &&[Del]
         IF DELETE()
            TONE(100,03)
            RECALL
         ELSE
            TONE(500,03)
            DELETE
         ENDIF
      CASE t_keypress= 13 &&[Return]
         DO WHILE t_keypress= 13 &&Do while selected data
            DO prg021
            READ
            t_savscr03=SAVESCREEN(11,02, 13,77)
            IF READKEY()=12
               t_msg="Update data CANCEL!"
               t_time=10
               t_tone01=5000
               t_dura01=03
               t_tone02=2000
               t_dura02=05
            ELSE
               t_msg="Update data COMPLETE!"
               t_time=05
               t_tone01=40
               t_dura01=10
               t_tone02=50
               t_dura02=20
            ENDIF
```

```
                t_msg=STUFF(SPACE(120),1,LEN(t_msg),t_msg)
                @ 12,03 GET t_msg PICTURE "@S74"
                CLEAR GETS
                FOR t_count=1 TO t_time
                    TONE(t_tone01,t_dura01)
                    TONE(t_tone02,t_dura02)
                NEXT
                RESTSCREEN(11,02, 13,77, t_savscr03)
                EXIT
            ENDDO &&End do while selected data
        CASE t_keypress= 27 &&[Esc]
            EXIT
    ENDCASE
    t_keypress=0
ENDDO &&Enddo while not key press

*Close data
CLOSE DATABASE

RESTORE SCREEN FROM t_savscr01

RETURN
```

```
* Program-ID.....; PRG203.PRG
* Purpose.....: PROGRAM 203
* Call from.....: MENU002.PRG
* Date Written....:  3-22-91 11:54:12 pm
* Language.....: Clipper Summer 87
* Note.....:

* memory variable
t_savscr01=SAVESCREEN(11,02, 13,77)
t_ndx=0

* environment
USE currency INDEX ndxdate ALIAS currency
GO TOP

DO WHILE LEN(INDEXKEY(t_ndx))>0 &&Do while for checking order
   t_ndx=t_ndx+1
ENDDO
t_ndx=t_ndx-1

DECLARE t_ndxfil[t_ndx], t_ndxkey[t_ndx] &&Assign index_file,
        index_key to array
FOR t_count=1 TO t_ndx
    t_ndxfil[t_count]=FIELD(t_count)
    t_ndxkey[t_count]=INDEXKEY(t_count)
NEXT t_count

t_dbf=DBF()+".DBF"

* display fixed text
@ 11,02 TO 13,77 DOUBLE
```

```
@ 12,03 CLEAR TO 12,76

* Process get data
PACK
FOR t_count=1 TO t_ndx
    t_ntxfil=t_ndxfil[t_count]
    t_ntxfil="NDX"+LEFT(t_ntxfil,03)+".NTX"
    t_ntxkey=t_ndxkey[t_count]
    t_msg="DBF: "+t_dbf+" ₪ NTX: "+t_ntxfil+" ₪ KEY: "+t_ntxkey
    t_msg=STUFF(SPACE(80),1,LEN(t_msg),t_msg)
    @ 12,03 GET t_msg PICTURE "@S74"
    CLEAR GETS
    INDEX ON &t_ntxkey TO &t_ntxfil
    TONE(50,03)
NEXT t_count

RESTSCREEN(11,02, 13,77, t_savscr01)

RETURN
```

```
* Program-ID.....: PRG420.PRG
* Purpose.....: Program 420
* Call from.....: MENU402.PRG
* Date Written....:  3-22-91 11:51:30 pm
* Language.....: Clipper Summer 87
* Note.....:

* memory variable
t_row=PROW()

* environment

* display fixed text
t_msg="Data Member"
@ t_row+00,(80-LEN(t_msg))/2 SAY t_msg
@ t_row+01,00 SAY REPLICATE("_",80)
IF DELETE()
   t_msg=" DELETED! "
ELSE
   t_msg="          "
ENDIF
@ t_row+02,05 SAY t_msg

*Process get data

@ t_row+03,05 SAY "Member Number "+customer->number
@ t_row+04,05 SAY "Name "+customer->name+" Surname "
                  +customer->surname
@ t_row+05,05 SAY "National "+customer->nation
@ t_row+06,05 SAY "Address "
t_lcount=MLCOUNT(customer->address, 60)
FOR t_count=1 TO t_lcount
```

```
    t_prnline=MEMOLINE(customer->address, 60, t_count)
    @ t_row+06+t_count-1,13 SAY t_prnline.
NEXT t_count
@ t_row+06+t_count,05 SAY "Tel. "
@ t_row+06+t_count,10 SAY customer->tel PICTURE "@R (NNN)-NNN-NNNN"
@ t_row+07+t_count,05 SAY "ID Card "+customer->idcard
@ t_row+08+t_count,05 SAY "Passport Number "+customer->passport
@ t_row+09+t_count,00 SAY REPLICATE("=",80)+CHR(10)+CHR(13)

RETURN
```

```
* Program-ID.....:.PRG421.PRG
* Purpose.....: Program 421
* Call from.....: MENU402.PRG
* Date Written....:  3-22-91 11:51:30 pm
* Language.....: Clipper Summer 87
* Note.....:

* memory variable
PRIVATE t_lcount, t_pageno, t_lineno, t_linepage
t_pageno=00
t_linepage=23

* Environment

DO WHILE .NOT. EOF() &&Do while not EOF

   t_pageno=t_pageno+01 &&Increase page number

   * Print fixed text
   ** Header
   @ 00,00 SAY "Reported date: "+DTOC(DATE())
   @ 00,70 SAY "Page "+TRANSFORM(t_pageno,"@Z 9,999")

   ** Page title
   t_msg="Data Member"
   @ 01,(80-LEN(t_msg))/2 SAY t_msg
   t_msg="Summary Report"
   @ 02,(80-LEN(t_msg))/2 SAY t_msg
   @ 03,00 SAY "                                        Customer
   @ 04,00 SAY "Member No.          Name                Surname"
   @ 05,00 SAY "     ID Card        Address"
   @ 07,00 SAY " Passport           Tel.                Nation"
```

```
ENDDO &&Enddo while not EOF

RETURN
```

```
* Program-ID.....: PRG501.PRG
* Purpose.....: PROGRAM 501
* Call from.....: MENU005.PRG
* Date Written....:
* Language.....: Clipper Summer 87
* Note.....:

* Memory variable
DECLARE t_files[ADIR("*.*")]
PRIVATE t_choice, t_edit
ADIR("*.*",t_files)

* Menu
SAVE SCREEN TO t_savscr01

@ 00,32 CLEAR TO 24,47
@ 00,32 TO 24,47 DOUBLE
t_choice=ACHOICE(01,34, 23,45, t_files)

IF t_choice<>0
   t_edit=t_files[t_choice]
   DO prg510 WITH t_edit
ENDIF

RESTORE SCREEN FROM t_savscr01

RETURN
```

```
* Program-ID.....: PRG502.PRG
* Purpose.....: PROGRAM 502
* Call from.....: MENU005.PRG
* Date Written....:  3-22-91 11:51:04 pm
* Language.....: Clipper Summer 87
* Note.....:

* memory variable

* environment

* display fixed text

SAVE SCREEN TO t_savscr02

t_file="CURRENCY.HLP"
SET COLOR TO +w/n
@ 01,01, 23,78 BOX "ฺฺฺฺฺฺฺฺ "
@ 21,01 SAY 'ล'+REPLICATE('ฺ',76)+'ฤ'
t_msg="Scroll by "+chr(24)+" "+chr(25)+" or [PgUp] [PgDn];
       [Esc] to EXIT!."
@ 22,(80-LEN(t_msg))/2 SAY t_msg
SET COLOR TO w/n,n/w

IF .NOT. FILE("&t_file")
  @ 00,00 SAY "Cannot open file "+t_file
ENDIF

MEMOEDIT(MEMOREAD(t_file),02,02,20,77,.f.)

RESTORE SCREEN FROM t_savscr02
```

RETURN

```
t_tabsize = 4

* Environtment
SET BELL ON
SET SCOREBOARD OFF

DO WHILE .T.

   IF .NOT. t_eresume
      DO config

      IF LASTKEY() = 27
         @ t_bott + 1, 79 SAY ""
         EXIT
      ENDIF

      IF t_usrfun
         t_u_func = "mfunc"
         SET SCOREBOARD OFF
      ELSE

         IF t_browse
            t_u_func = ""
         ELSE
            t_u_func = .F.
         ENDIF

         IF t_tops = 0
            SET SCOREBOARD OFF
         ELSE
            SET SCOREBOARD ON
         ENDIF
```

```
      ENDIF

      IF .NOT. (t_pre_fil == t_edfile)
         t_usrmemo = MEMOREAD(t_edfile)
         t_pre_fil = t_edfile
         t_lineno = 1
         t_colno = 0
         t_rel_row = 0
         t_rel_col = 0
         t_altered = .F.
      ENDIF

      t_msg_len = t_righ - t_left - 25
      CLEAR
      @ t_tops, t_left, t_bott, t_righ BOX "ๆ๏บตกา๏ใต"
      @ t_bott + 1, t_left SAY splash(LOWER(t_edfile), t_msg_len)

   ENDIF

   t_intcount = 1
   t_deja_vu = .F.
   t_eresume = .F.
   t_usrmemo = MEMOEDIT(t_usrmemo, t_tops + 1, t_left + 1,
               t_bott - 1, t_righ - 1, t_update, t_u_func,
               t_lilong, t_tabsize, t_lineno, t_colno,
               t_rel_row, t_rel_col)

   IF .NOT. t_edfile == "none" .AND. .NOT. EMPTY(t_usrmemo) .AND.
            t_ret_val = 23
      t_altered = .F.
```

```
      IF .NOT. MEMOWRIT(t_edfile, t_usrmemo)
         SET COLOR TO w*+
         @ 24,79 SAY ""
         ? "DISK WRITE ERROR"
         SET COLOR TO
         CLOSE DATABASES
         EXIT
      ENDIF

      @ t_bott + 1, t_left SAY splash("Write successful.", t_msg_len)
   ENDIF

ENDDO

RETURN

****
*       mfunc()
*
*       memoedit user function
****
FUNCTION mfunc

PARAMETERS mode, t_line, t_col
PRIVATE t_keypress
t_ret_val = 0

DO CASE

   CASE mode = 3
        * initialization..global variables "t_intcount" and "t_deja_vu"
        *       control the initialization process..note that this is
        *       much simpler when the parameters passed to MEMOEDIT
```

```
*          are known in advance (which is usually true)

IF t_intcount = 1
   * set initial insert mode
   t_ins_mode = READINSERT()

   IF (t_inst_on .AND. .NOT. t_ins_mode) .OR. (.NOT. t_inst_on
      .AND. t_ins_mode)
      * toggle insert mode
      t_ret_val = 22
   ELSE
      * insert mode correct
      t_intcount = 2
      @ t_bott + 1, t_righ - 25 SAY IF(t_inst_on, "I", " ")
   ENDIF

ENDIF

IF t_intcount = 2
   * set initial scroll state (defaults ON if t_update OFF)

   IF ((.NOT. t_scrl_on .AND. .NOT. t_update) .OR. (t_scrl_on
      .AND. t_update)) .AND. .NOT. t_deja_vu
      * need to toggle
      t_deja_vu = .T.
      t_ret_val = 35
   ELSE
      * scroll state correct
      t_intcount = 3
      t_deja_vu = .F.
      @ t_bott + 1, t_righ - 24 SAY IF(t_scrl_on, "S", " ")

   ENDIF
ENDIF
```

```
        IF t_intcount = 3
           * set initial word wrap..always defaults ON

           IF .NOT. t_wrap_on .AND. .NOT. t_deja_vu
              * need to toggle
              t_deja_vu = .T.
              t_ret_val = 34
           ELSE
              * word wrap correct
              t_intcount = 4
              t_deja_vu = .F.
              @ t_bott + 1, t_righ - 23 SAY IF(t_wrap_on, "W", " ")
           ENDIF

        ENDIF

        IF t_intcount = 4
           * finished initialization..note that if all defaults are
           *      correct we reach this point on the first call
           t_ret_val = 0
        ENDIF

     CASE mode = 0
        * idle
        @ t_bott + 1, t_righ - 20 SAY "Line: " + splash(LTRIM
          (STR(t_line)), 4)
        @ t_bott + 1, t_righ - 8 SAY "Col: " + splash(LTRIM
          (STR(t_col)), 3)

     OTHERWISE
        * keystroke exception
        t_keypress = LASTKEY()
```

```
* save values to possibly resume edit
t_lineno = t_line
t_colno = t_col
t_rel_row = ROW() - t_tops - 1
t_rel_col = COL() - t_left - 1

IF mode = 2
   t_altered = .T.
ENDIF

DO CASE

   CASE t_keypress = 23
        * ^W..ignore (disable)
        t_ret_val = 32

   CASE t_keypress = 273
        * Alt-W..write file
        IF .NOT. t_altered
          * no changes to write
          @ t_bott + 1, t_left SAY splash("No changes to
            write.", t_msg_len)
        ELSE
          * write and resume
          @ t_bott + 1, t_left SAY SPACE(t_msg_len)
          @ t_bott + 1, t_left SAY "Writing " + LOWER
            (t_edfile) + "..."
          t_ret_val = 23
          t_eresume = .T.

        ENDIF

   CASE t_keypress = 301 .OR. t_keypress = 27
        * Esc/Alt-X..exit
```

```
IF .NOT. t_altered
   * no change
   t_ret_val = 27
ELSE
   * changes have been made to memo
   @ t_bott + 1, t_left SAY SPACE(t_msg_len)
   @ t_bott + 1, t_left SAY "Abandon [ynw]? "
   t_response = " "

   DO WHILE .NOT. t_response $ "YNW"
      t_response = UPPER(CHR(INKEY(0)))
   ENDDO

   @ t_bott + 1, t_left SAY SPACE(t_msg_len)

   DO CASE

      CASE t_response = "Y"
         * abort
         t_ret_val = 27

      CASE t_response = "N"
         * ignore
         t_ret_val = 32

      CASE t_response = "W"
         * save and exit
         @ t_bott + 1, t_left SAY SPACE(t_msg_len)


         @ t_bott + 1, t_left SAY "Writing " + LOWER
           (t_edfile)+"..."
         t_ret_val = 23
   ENDCASE
```

```
        ENDCASE
ENDCASE

RETURN t_ret_val

****
*       splash()
*
*       splash with spaces
****
FUNCTION splash

PARAMETERS t_string, t_length
RETURN SUBSTR(t_string + SPACE(t_length), 1, t_length)

****
*       config
*
*       set parameters for simulation
****
PROCEDURE config

t_edfile = splash(t_edfile, 64)
CLEAR


@ 1, 0 SAY "File To Edit_" GET t_edfile PICTURE "@K"
@ 3, 0 SAY "Window Coordinates:"
@ 5, 5 SAY "Top_____" GET t_tops
@ 6, 5 SAY "Left_____" GET t_left
@ 7, 5 SAY "Bottom_____" GET t_bott
@ 8, 5 SAY "Right_____" GET t_righ
```

```
@ 10, 5 SAY "Line Length (memowidth)_" GET t_lilong
@ 11, 5 SAY "Tab Size_________________" GET t_tabsize

@ 13, 5 SAY "Allow Updates [yn]_____" GET t_update
@ 14, 5 SAY "Allow Browse [yn]______" GET t_browse
@ 15, 5 SAY "Use User Function [yn]__" GET t_usrfun

@ 17, 5 SAY "Insert Mode On [yn]____" GET t_inst_on
@ 18, 5 SAY "Scroll State On [yn]___" GET t_scrl_on
@ 19, 5 SAY "Word Wrap On [yn]______" GET t_wrap_on
READ

t_edfile = LTRIM(TRIM(t_edfile))

RETURN
```