การพัฒนาไมโครโปรเซสเซอร์ให้เป็นลอจิกแอนะไลเซอร์

MODIFICATION OF AN IBM PC AS LOGIC ANALYZER

วิชิต ตั้งสุขนิรันดร

WICHIT TANGSUKNIRUNDORN

อาจารย์ที่ปรึกษา

รองศาสตราจารย์ ดร. วัลลภ สุระกำพลธร

ADVISOR

ASSOC. PROF. DR. WANLOP SURAKAMPONTHORN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิศวกรรมไฟฟ้า

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหาร ลาดกระบัง

พ.ศ. 2534

ISBN 974-8157-71-7

บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ การพัฒนาไมโครโปรเซสเซอร์ให้เป็นเลอจิกแอนนาไลเซอร์
MODIFICATION OF AN IBM PC AS LOGIC ANALYZER

ชื่อนักศึกษา นายวิชิต ตั้งสุขนิรันดร รหัสประจำตัว 28620012
หลักสูตร วิศวกรรมศาสตรมหาบัณฑิต สาขาวิชา วิศวกรรมไฟฟ้า
ภาควิชา อิเล็กทรอนิคส์
อาจารย์ผู้ควบคุมวิทยานิพนธ์ รศ.ดร.วัลลภ สุระกำพลธร

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
รศ.ดร.วัลลภ	สุระกำพลธร	
รศ.ดร.จเร	สุรวัฒน์ปัญญา	
รศ.ดร.มนัส	สังวรศิลป์	
ผศ.พิพัฒน์	เลาหสงคราม	
รศ.ดร.ชม	ตั้งทวีถาน	

ค่าระดับคะแนนที่เป็นเอกฉันท์จากคณะกรรมการสอบ GOOD
วัน/เดือน/ปี ที่สอบ 25 มิถุนายน 2535 เวลา 12.00 น.
สถานที่สอบ คณะวิศวกรรมศาสตร์

บัณฑิตวิทยาลัยรับรองแล้ว

(รศ.ดร.สุทัศน์ อัตถากร)
รองคณบดีฝ่ายบัณฑิตศึกษา, คณะวิศวกรรมศาสตร์
รักษาราชการแทนคณบดีบัณฑิตวิทยาลัย
วันที่ 25 เดือน มิถุนายน พ.ศ. 2535

หมายเหตุ การวัดผลวิทยานิพนธ์ให้ใช้ค่าระดับคะแนนดังนี้

ค่าระดับคะแนน	ผลการศึกษา
O	Outstanding (ดีเยี่ยม)
G	Good (ดี)
P	Pass (ผ่าน)
F	Fail (ไม่ผ่าน)

หัวข้อวิทยานิพนธ์ การพัฒนาไมโครโปรเซสเซอร์ให้เป็นลอจิกแอนนะไลเซอร์
นักศึกษา นาย วิชิต ตั้งสุขนิรันดร
อาจารย์ที่ปรึกษา รศ. ดร. วัลลภ สุระกำพลธร
ระดับการศึกษา วิศวกรรมศาสตรมหาบัณฑิต สาขา วิศวกรรมไฟฟ้า
พ.ศ. 2534

บทคัดย่อ

วิทยานิพนธ์ฉบับนี้ เป็นการแสดงผลงานวิจัยเกี่ยวกับการพัฒนาไมโครโปรเซสเซอร์ให้เป็นลอจิกแอนนะไลเซอร์ โดยเสนอแนวทางการพัฒนาระบบไมโครโปรเซสเซอร์เบอร์ HD64180 ซึ่งมีขนาด 8 บิท ให้เป็นเครื่องลอจิกแอนนะไลเซอร์ ที่สามารถใช้ในการตรวจสอบค่าของลอจิกในคอมพิวเตอร์ หรือ อุปกรณ์อิเลคตรอนิกส์ต่างๆได้ถึง 32 ช่องสัญญาณ โดยมีอัตราการสุ่มสัญญาณที่ต้องการสูงสุด 50 เมกกะเฮิทซ์ และสามารถลดทอนอัตราการสุ่มสัญญาณได้ด้วยอัตรา 5-2-1 (50 , 20 , 10 เมกกะเฮิทซ์ จนถึง 100 เฮิทซ์) การควบคุมการทำงาน และการแสดงผลการจัดกระทำโดยอาศัยเครื่องคอมพิวเตอร์ส่วนบุคคล IBM PC/XT/AT เป็นเทอร์มินัล ซึ่งมีการเชื่อมต่อกับเครื่องลอจิกแอนนะไลเซอร์ผ่านทางพอร์ทอนุกรม RS-232 เมื่อผู้ใช้เครื่องลอจิกแอนนะ-ไลเซอร์ สามารถตรวจสอบข้อมูลได้อย่างถูกต้อง การวิเคราะห์ปัญหาที่เกิดขึ้นจะทำได้อย่างรวดเร็วและแม่นยำ ทำให้ระยะเวลาที่ใช้ในการแก้ปัญหาน้อยลงจากเดิมมากมาย

ผลการทดสอบการใช้งานปรากฏว่า แผงวงจร (CIRCUIT BOARD) และซอฟแวร์ (SOFTWARE) ที่สร้างขึ้นเป็นลอจิกแอนนะไลเซอร์ มีประสิทธิภาพสูงเป็นที่น่าพอใจ

THESIS TITLE MODIFICATION OF AN IBM PC AS LOGIC ANALYZER

STUDENT MR. WICHIT TANGSUKNIRUNDORN

THESIS ADVISOR ASSOC. PROF. DR. WANLOP SURAKAMPONTHORN

LEVEL OF STUDY MASTER'S IN ELECTRICAL ENGINEERING

YEAR 1991

ABSTRACT

This paper presents the development of Logic Analyzer for 8 bit microprocessor HD 64180. The Logic Analyzer is frequently used for detecting, trouble shoot solving in microcomputer system and electronics equipment.

The Logic Analyzer can detect 32 bit datas, select sampling rate from 50 MHz to 100 Hz in ratio 5-2-1 , control and display results by using IBM PC/XT/AT as terminal. The data communication link between Logic Analyzer and PC is RS-232 serial port. After the user get the logic data , analysis of problem can easily be done and the problem solving time can be greatly reduced.

Experimental results show the Logic Analyzer has high efficiency performance comparable to the commercially available logic analyzer at present.

กิตติกรรมประกาศ

ผู้จัดทำวิทยานิพนธ์ใคร่ขอขอบพระคุณเป็นอย่างสูงต่อ รองศาสตราจารย์ ดร.วัลลภ สุระกำพลธร อาจารย์ที่ปรึกษา และรองศาสตราจารย์ ดร. จเร สุรวัฒน์ปัญญา ซึ่งอาจารย์ทั้งสองได้ให้คำแนะนำ , แนวความคิด และคำปรึกษาจนกระทั่งวิทยานิพนธ์ฉบับนี้เสร็จสมบูรณ์

สุดท้ายนี้ขอขอบคุณ คุณ จิระ จิริยะสิน แห่งบริษัทถาวรคอมพิวเตอร์ และคุณ พนม เพชรจตุพร ที่ได้ให้ความช่วยเหลือด้านวัสดุอุปกรณ์ การตรวจสอบ และแก้ไขทั้งในด้านโปรแกรม รวมไปถึงวงจรต่างๆ ซึ่งทำให้วิทยานิพนธ์ฉบับนี้มีความสมบูรณ์ยิ่งขึ้น

สารบัญ

สารบัญตาราง

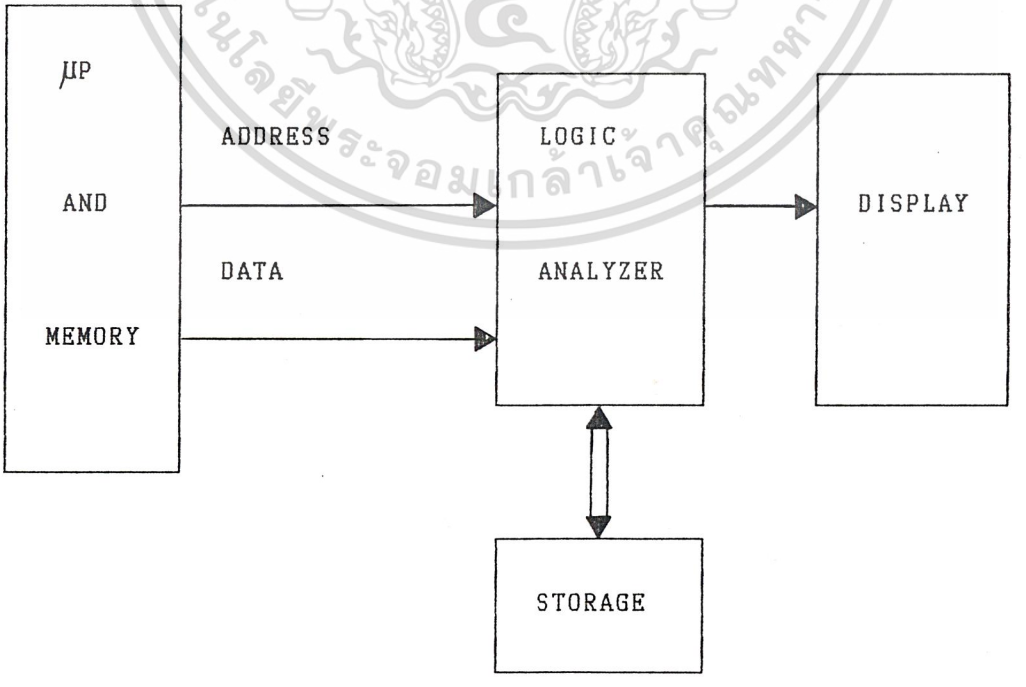
สารบัญภาพ

บทที่ 1
บทนำ

1.1 กล่าวนำ

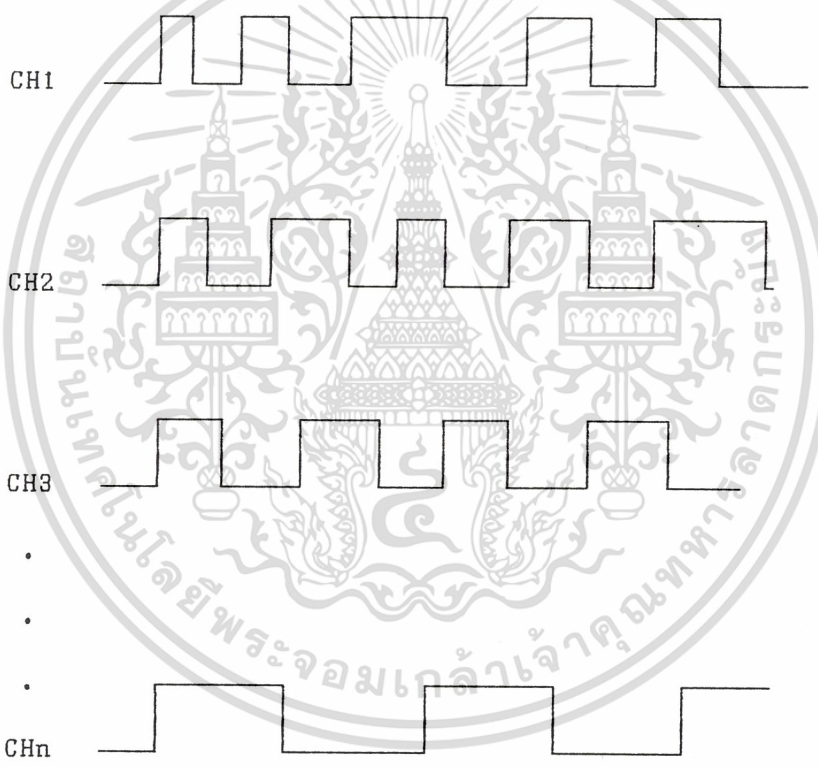
ในปัจจุบันนี้เทคโนโลยีทางด้านคอมพิวเตอร์ได้เจริญรุดหน้าไปอย่างรวดเร็วโดยทั่วไปมักจะเป็นชิพ (CHIP) สำเร็จที่สร้างขึ้นมาเฉพาะงาน การวิเคราะห์การทำงานของชิพสนับสนุนเหล่านี้ ถ้าจะใช้เครื่องมือวัดธรรมดาเช่น ออสซิสโลสโคป หรือ มิเตอร์ มาทำการวัดสัญญาณต่างๆ จะยุ่งยากมากขึ้น และขณะเดียวกัน จะสามารถตรวจสอบได้เฉพาะความผิดพลาดทางฮาร์ดแวร์เท่านั้น ส่วนคำสั่งทางซอฟแวร์จะวิเคราะห์ไม่ได้เลยด้วยเหตุนี้จึงมีการพัฒนาเครื่องวิเคราะห์สัญญาณทางตรรกขึ้นมาใช้งาน

เครื่องวิเคราะห์สัญญาณทางตรรก จะทำหน้าที่นำเอาสัญญาณที่วัดได้ขึ้นมาแสดงให้ผู้ใช้ได้ทราบถึง สภาวะทางตรรกในช่วงเวลาหนึ่งๆได้พร้อมกันหลายช่องสัญญาณดังรูปที่ 1.1



รูปที่ 1.1 แสดงการใช้เครื่องวิเคราะห์สัญญาณทางตรรก ทำการวิเคราะห์สัญญาณจากบอร์ดไมโครโปรเซสเซอร์

จากรูปที่ 1.1 สมมุติว่าต้องการทราบขั้นตอนการทำงานของบอร์ดไมโครโปรเซส-เซอร์บอร์ดหนึ่ง จะกระทำโดยใช้เครื่องวิเคราะห์สัญญาณทางตรรกเข้าช่วยโดยใช้สายนำสัญญาณจับสัญญาณที่บัสแอดเดรสและบัสข้อมูลดังรูป เครื่องวิเคราะห์สัญญาณทางตรรกจะแสดงผลที่จับได้ออกมาทางจอภาพในลักษณะสัญญาณทางตรรกดังนี้



จากรูปที่ได้จะทำให้สามารถตรวจสอบได้ทันทีว่าในขณะนั้นไมโครโปรเซสเซอร์กำลังทำคำสั่งอะไรอยู่ ถูกต้องหรือไม่ ด้วยเหตุนี้จึงทำให้การวิเคราะห์การทำงานของระบบที่ต้องการตรวจสอบทำได้เร็วขึ้น

1.2 วัตถุประสงค์และขอบเขตของวิทยานิพนธ์

วัตถุประสงค์ของวิทยานิพนธ์นี้เพื่อต้องการพัฒนาอุปกรณ์ที่เรียกว่า เครื่องวิเคราะห์สัญญาณทางตรรก ที่มีประสิทธิภาพสูง และมีราคาที่ไม่แพงขึ้นมา เพื่อใช้เป็นเครื่องมือสำหรับตรวจสอบข้อมูลทางตรรก

เครื่องวิเคราะห์สัญญาณทางตรรกที่พัฒนานี้เป็นการพัฒนาไมโครโปรเซสเซอร์ให้เป็นเครื่องวิเคราะห์สัญญาณทางตรรก ที่สามารถตรวจรับสัญญาณได้ ถึง 32 ช่องสัญญาณ ซึ่งสามารถรองรับได้กับเครื่องคอมพิวเตอร์ส่วนบุคคลในปัจจุบัน 32 บิทได้ และอัตราการสุ่มสัญญาณได้ถึง 50 เมกกะเฮิทซ์

เครื่องวิเคราะห์ที่พัฒนาขึ้นนี้มีขอบเขตในการพัฒนา 2 ส่วนด้วยกัน คือ

1. วงจร ประกอบด้วยวงจรของเครื่องวิเคราะห์สัญญาณทางตรรก ซึ่งพัฒนามาจากไมโครโปรเซสเซอร์เบอร์ HD64180 ซึ่งสามารถตรวจจับสัญญาณได้ถึง 32 ช่องสัญญาณ และสามารถใช้อัตราการสุ่มสัญญาณได้ที่ความถี่สูงพอสมควร

2. โปรแกรมควบคุมระบบของเครื่องวิเคราะห์สัญญาณทางตรรก

โดยที่เครื่องวิเคราะห์สัญญาณทางตรรกนี้จะอาศัยคอมพิวเตอร์ส่วนบุคคล (IBM PC) เป็นเทอร์มินอล

1.3 โครงสร้างและหลักการทำงานทั่วไปของเครื่องวิเคราะห์สัญญาณทางตรรก

โครงสร้างและหลักการทำงานทั่วไปของลอจิกแอนนะไลเซอร์นั้นสามารถแสดงดังบล็อกไดอะแกรมในรูปที่ 1.3 ซึ่งประกอบไปด้วยส่วนย่อยๆดังนี้

1. ส่วนสัญญาณด้านขาเข้าแบบขนาน (PARALLEL DATA INPUT)

วงจรจะประกอบไปด้วย วงจรศักดาไฟฟ้าขีดเริ่มเปลี่ยน (THRESHOLD VOLTAGE) วงจรแปลงจาก ECL-TTL , วงจรหน่วงสัญญาณ , อินเวอร์เตอร์ , วงจรสุ่มตัวอย่าง (SAMPLE/LATCH) ซึ่งจุดประสงค์ของส่วนนี้จะเป็นตัวปรับระดับของสัญญาณขาเข้าให้มีระดับที่เหมาะสมเพื่อนำไปประมวลผลต่อไป

2. ส่วนรู้จำของคำ (WORD RECOGNIZER)

วงจรของส่วนรู้จำของคำจะให้สัญญาณขาออกเป็นสูง (HIGH) เมื่อสัญญาณทางตรรกของสัญญาณที่รับเข้ามาจากแท่งทดสอบ (PROBE) เข้ากันได้กับสัญญาณขาเข้าที่เลือกไว้ (QUALIFIER INPUT) ซึ่งสัญญาณจากส่วนรู้จำของคำนี้เป็นสัญญาณกระตุ้นแบบอะซิงโครนัส (ASYNCHRONOUS TRIGGER PLUSE)

3. ส่วนหน่วยความจำรับสัญญาณแบบขนาน (PARALLEL ACQISITION MEMORY)

เป็นวงจรที่ประกอบไปด้วยหน่วยความจำแบบแรม (RAM) ตัวนับแอดเดรส (ADDRESS COUNTER), ซึ่งหน่วยความจำนี้เป็นตัวเก็บสัญญาณขาเข้าซึ่งผ่านการปรับระดับของสัญญาณแล้ว และตัวนับแอดเดรสเป็นตัวเลือกบริเวณของแอดเดรสที่ต้องการพิจารณา

4. ส่วนหน่วงสัญญาณกระตุ้น (TRIGGER DELAY)

วงจรหน่วงสัญญาณกระตุ้นจะเป็นวงจรประกอบด้วย วงจรกระตุ้น (TRIGGER) และดีเลย์ต่างๆ เช่น TRIGGER DELAY COUNTER , TRIGGER GATE STAGE ,

DELAYED GATE STAGE และอื่นๆ ซึ่งจะทำให้เกิดสัญญาณขาออกที่หน่วงการกระตุ้น (DELAYED TRIGGER OUTPUT) เป็นสัญญาณที่จะไปกระตุ้น วงจร PARALLLEL ACQUISITION MEMORY

5. วงจรคาบของเวลา (TIME BASE)

เป็นวงจรประกอบด้วยความถี่มาตรฐาน , วงจรแบ่งความถี่ ซึ่งหน้าที่ของวงจรจะเป็นส่วนให้สัญญาณนาฬิกากับตัวเอ็มพียู (MPU) และวงจรควบคุมการแสดงผล

6. วงจรสัญญาณขาเข้าแบบอนุกรม (SERIAL/SIGNATURE INPUT)

เป็นวงจรทำหน้าที่เปรียบเทียบสัญญาณขาเข้า ปรับแต่งระดับของสัญญาณให้เหมาะสม เพื่อส่งเข้าสู่วงจรในส่วนของอนุกรม (SERIAL) อื่นๆต่อไป

7. ตัวกำเนิดสัญญาณซีกนะเชอะ (SIGNATURE GENERATER)

จะเป็นตัวรับสัญญาณแบบอนุกรมเข้ามาและถอดรหัส (DECODE) ของสัญญาณขาเข้า และทำให้เป็นสัญญาณขาออกสิบหกบิท (16-BIT OUTPUT) และส่งเข้าไปยังแรมแสดงผลในหน่วยเอ็มพียู (MPU) ต่อไป

8. วงจรรับสัญญาณขาเข้าแบบอนุกรม (SERIAL DATA ACQUISITION)

วงจรรับสัญญาณขาเข้าแบบอนุกรม ประกอบด้วย ตัวกำเนิดสัญญาณโบด์เรด (BAUD RATE GENERATOR),USART(PROGRAMMABLE COMMUNICATION INTERFACE) และ ดาต้าบัสบัฟเฟอร์ ซึ่งวงจรนี้จะเป็นตัวจ่ายสัญญาณนาฬิกาแบบภายนอกและภายใน (INTERNAL/EXTERNAL) และรับสัญญาณขาเข้าแบบอนุกรมแล้วแปลงเป็นสัญญาณขาออกแบบขนาน โดยส่งผ่านดาต้าบัสบัฟเฟอร์เข้าสู่ดาต้าบัสเอ็มพียูต่อไป

9. คีบอร์ดและเอ็มพียู (KEYBOARD AND MPU)

- คีบอร์ดจะเป็นตัวส่งสัญญาณ 2 ข้อมูลไปยังเอ็มพียู โดยผ่านแลตช์และเกต เพื่อจัดเวลาให้เหมาะสม

- เอ็มพียูจะเป็นหน่วยของไมโครโปรเซสเซอร์ , รอม , แรม , ตัวถอดระหัสให้กับแอดเดรส ซึ่งเอ็มพียูจะเป็นหน่วยประมวลผลต่างๆนั่นเอง

10. หน่วยควบคุมการแสดงผล (DISPLAY CONTROL)

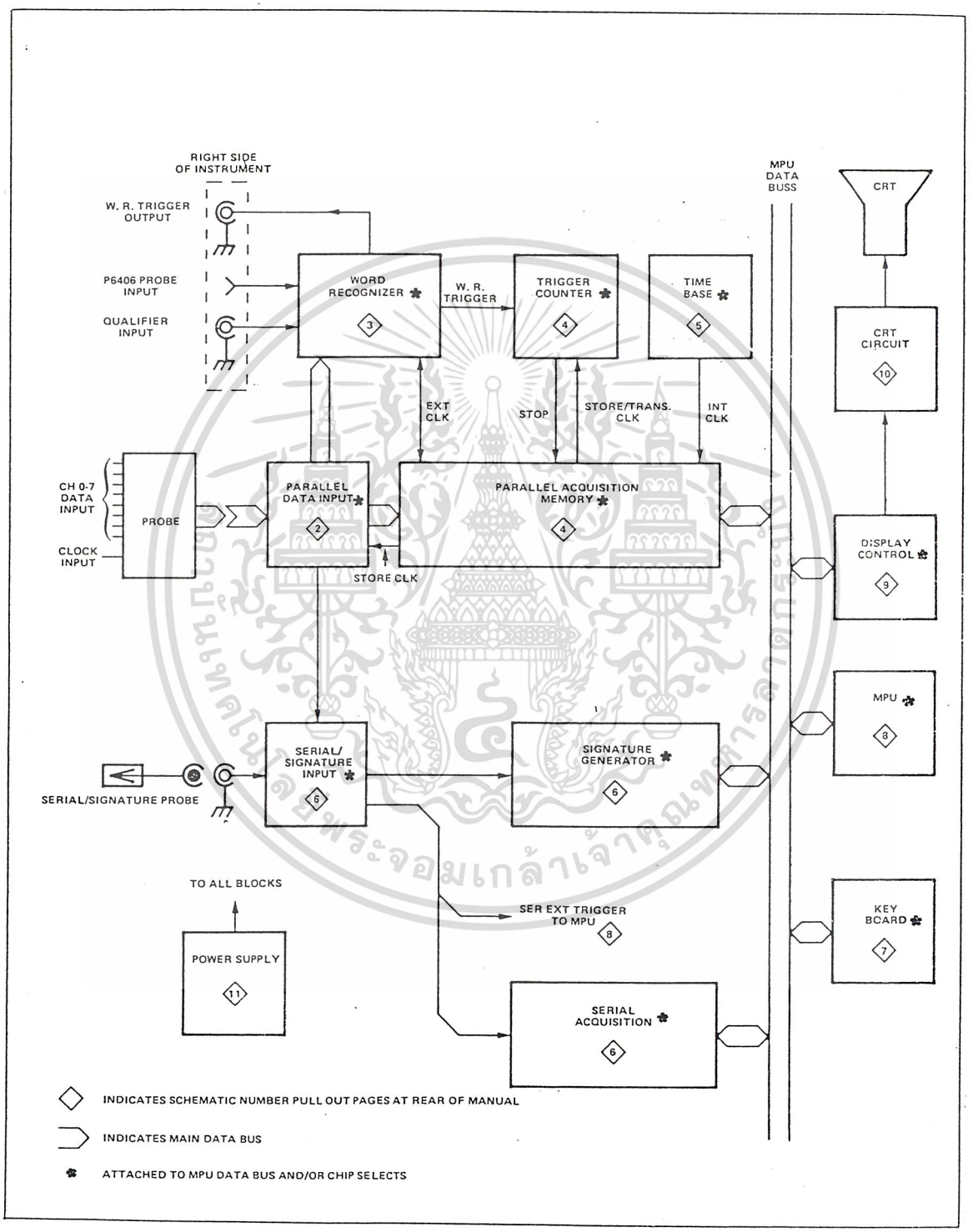
จะเป็นวงจรประกอบขึ้นด้วย RAM ซึ่งจะเก็บข้อมูลที่แสดงผล ซึ่งได้รับการประมวลผลจากเอ็มพียูไว้ เพื่อจะส่งออกไปยังจอภาพต่อไป

11. จอภาพ (CRT)

เป็นหน่วยที่ใช้แสดงผลออกมาให้เห็น

12. วงจรจ่ายกำลัง (POWER SUPPLY)

เป็นวงจรจ่ายกำลัง หรือ จ่ายศักดาไฟฟ้าแบบดีซีในวงจรทางอิเลคตรอนิกส์ทุกๆภาคนั่นเอง



รูปที่ 1.3 โครงสร้างพื้นฐานของเครื่องวิเคราะห์สัญญาณทางตรรก

บทที่ 2

ทฤษฎีและการออกแบบ

2.1 แนวการออกแบบ

โดยอาศัยแนวความคิดจากบทนำและกำหนดรูปแบบที่ต้องการใหม่ดังนี้

1. เครื่องวิเคราะห์สัญญาณทางตรรกที่สร้างขึ้น ต้องติดต่อใช้งานร่วมกับ เครื่องคอมพิวเตอร์ส่วนบุคคล เพื่ออาศัยจอภาพและจานแม่เหล็กเป็นตัวแสดงผลที่วัดได้ และเก็บบันทึกข้อมูลที่วัดได้เพื่อนำมาใช้ภายหลัง

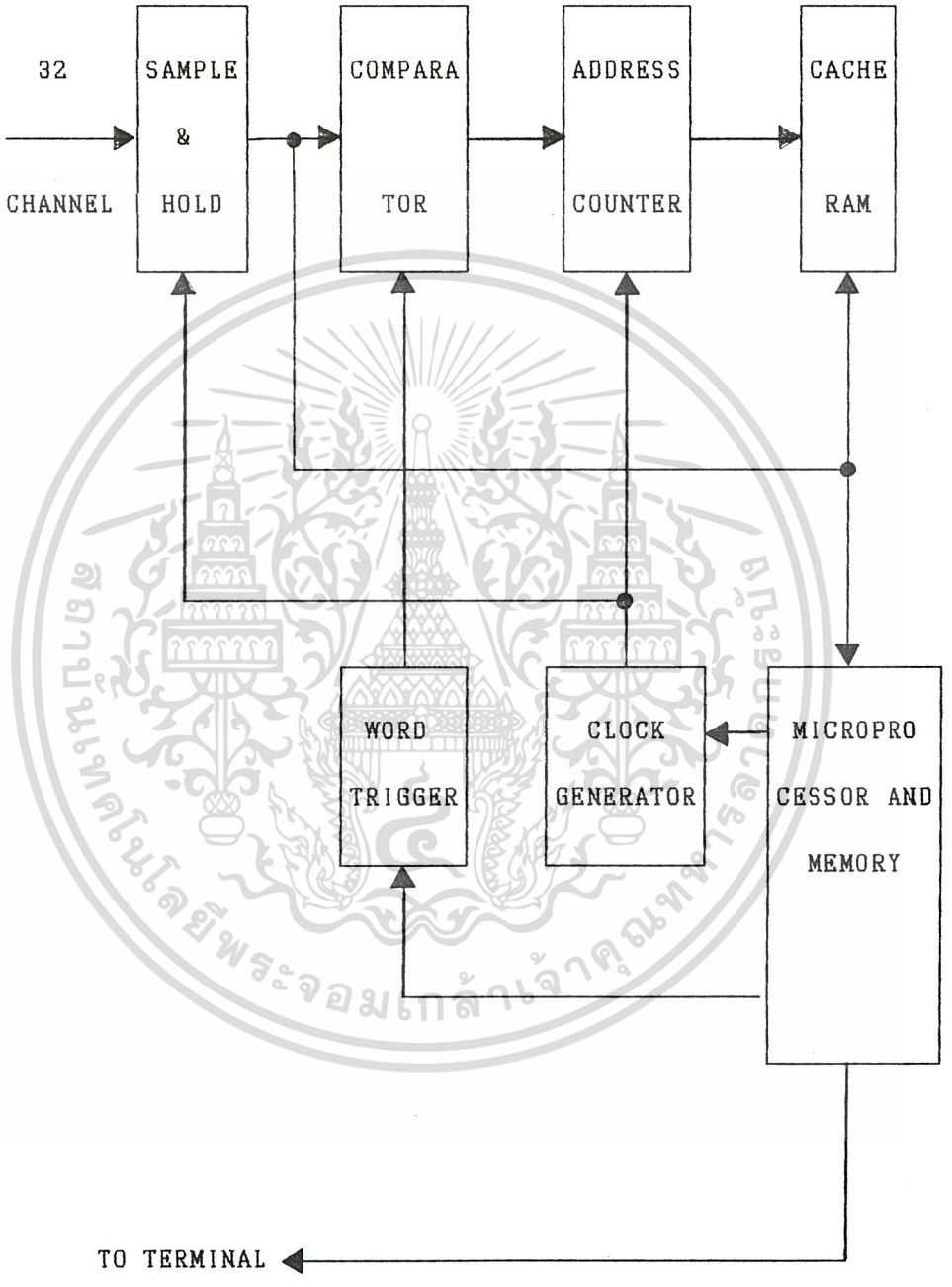
2. เครื่องวิเคราะห์สัญญาณทางตรรกที่สร้างขึ้นจะรับหรือส่งข้อมูลระหว่างตัวมันกับเครื่องคอมพิวเตอร์ส่วนบุคคลผ่านทางพอร์ทอนุกรมเท่านั้น ทั้งนี้เนื่องจากพอร์ทอนุกรมมีอยู่แล้วในเครื่องคอมพิวเตอร์ส่วนบุคคลทั่วๆไปหลายยี่ห้อ ถ้าสร้างเครื่องวิเคราะห์สัญญาณทางตรรกแล้วใส่ลงไปในเครื่องคอมพิวเตอร์เลย จะทำให้ไปผูกติดกับเครื่องคอมพิวเตอร์แต่ละยี่ห้อมากเกินไป ทำให้ไม่สะดวกในการใช้งาน

3. ช่วงสัญญาณที่ต้องการมีขนาด 32 ช่องสัญญาณ ซึ่งสามารถรองรับกับเครื่องคอมพิวเตอร์ส่วนบุคคลในปัจจุบันขนาด 32 บิทได้

4. อัตราการสุ่มสัญญาณเมื่อใช้สัญญาณนาฬิกาภายในเครื่อง (INTERNAL CLOCK) จะต้องสูงกว่าข้อมูลในบัสของเครื่องคอมพิวเตอร์ส่วนบุคคลที่มีอยู่ในปัจจุบัน และสามารถลดทอนได้พอสมควรต่อการใช้งานในที่นี้จึงเลือกใช้การสุ่มสัญญาณในอัตรา 50 เมกกะเฮิทซ์ และสามารถลดทอนลงด้วยอัตราการ 5-2-1 เช่น 50 , 20 , 10 , 5 เมกกะเฮิทซ์ เป็นต้น

5. สามารถกำหนดสภาวะลอจิก เพื่อให้เริ่มเก็บข้อมูลได้ทั้ง 32 ช่องสัญญาณ (DOUBLE WORD TRIGGERING) ทำให้ผู้ใช้แยกการใช้งานของช่องสัญญาณด้านเข้าไปจับสัญญาณจากบัสแอดเดรสหรือบัสข้อมูลได้ตามต้องการ

โดยอาศัยข้อกำหนดดังกล่าวข้างต้นจะสามารถเขียนแผนผังของบล๊อกได้ดังรูปที่ 2.1 สัญญาณทางตรรกที่รับเข้ามาจะเข้าผ่านส่วนสุ่มและคงค่า (SAMPLE AND HOLD) เพื่อทำให้สัญญาณที่รับเข้ามาคงที่ (STABLE) เสียก่อน จากนั้นสัญญาณส่วนหนึ่งจะถูกส่งไปยังส่วนเปรียบเทียบสัญญาณ (COMPARATOR) ส่วนนี้จะทำหน้าที่เปรียบเทียบสัญญาณที่รับเข้ามากับข้อมูลที่ผู้ใช้กำหนดผ่านทางส่วนของข้อมูลกระตุ้น (WORD TRIGER) เมื่อข้อมูลตรงกันก็จะส่งสัญญาณไปให้ส่วนนับตำแหน่งแอดเดรส (ADDRESS COUNTER) ทำงาน ส่วนนับตำแหน่งแอดเดรสจะรับสัญญาณนาฬิกามาจากส่วนสร้างสัญญาณนาฬิกา (CLOCK GENERATOR) และจะทำการเพิ่มตำแหน่งแอดเดรสขึ้นทีละหนึ่งไปเรื่อยๆ สัญญาณด้านออกของส่วนนับตำแหน่งแอดเดรสจะถูกต่อเข้ากับหน่วยความจำแรมแบบแคช (CACHE RAM) ที่ขาแอดเดรสที่สอดคล้องกัน ข้อมูลที่ได้จากส่วนสุ่มและคงค่าอีกส่วนหนึ่ง จะถูกต่อเข้าที่ขาข้อมูลของหน่วยความจำแรมแบบแคช เมื่อถึงขั้นตอนนี้จะเห็นว่า ข้อมูลที่รับเข้ามาจะถูกเก็บเข้าไว้ในหน่วยความจำแรมแบบแคชทั้งหมด เมื่อหน่วยความจำแรมแบบแคชทำการเก็บข้อมูลจนเต็มแล้วจะส่งข้อมูลเข้ามาเก็บไว้ในหน่วยความจำแรมภายในส่วนของไมโครโปรเซสเซอร์ ต่อจากนี้ไมโครโปรเซสเซอร์จะทำการส่งข้อมูลมายังคอมพิวเตอร์ส่วนบุคคล เพื่อทำการแสดงผลต่อไป



รูปที่ 2.1 แสดงแผนภาพบล๊อกของเครื่องวิเคราะห์สัญญาณทางตรรก

2.2 โครงสร้างและการทำงานของไมโครคอมพิวเตอร์แบบชิพเดียวตระกูล HD64180

ดังที่เราทราบกันอยู่แล้วว่า ไมโครโปรเซสเซอร์เป็นเสมือนหัวใจสำคัญของระบบไมโครคอมพิวเตอร์ ซึ่งวงจรไมโครโปรเซสเซอร์ที่นำมาใช้เป็นเสมือนหนึ่งหัวใจในวงจรที่ใช้เป็นลอจิกแอนนะไลเซอร์ กับ ไอบีเอ็ม พีซี (IBM PC)ก็คือไมโครโปรเซสเซอร์เบอร์ HD64180

จากแผนผังของบล๊อกจะเห็นว่าส่วนที่ควบคุมการทำงานของระบบทั้งหมดได้แก่ ส่วนของไมโครโปรเซสเซอร์ ในที่นี้ใช้ไมโครโปรเซสเซอร์ HD64180 ของบริษัทฮิตาชิ ซึ่งอาจจะใหม่สำหรับหลายๆท่านจึงใคร่ของกล่าวถึงเล็กน้อยพอเป็นสังเขป

ด้วยความก้าวหน้าทางเทคโนโลยี CMOS และ หน่วยการทำงานซึ่งทำงานด้วยไมโครโค้ด (Micro-code) ไมโครโปรเซส 8 บิท HD64180 มีข้อได้เปรียบในประสิทธิภาพการทำงานที่สูง ค่าใช้จ่ายที่ต่ำและกินไฟในขณะทำงานน้อย ขณะที่ซอฟแวร์ก็ยังสามารถใช้ร่วมกับซอฟแวร์ของไมโครโปรเซสเซอร์ของผู้ผลิตอื่นๆได้

การทำงานของไมโครโปรเซสเซอร์ตระกูลนี้ ได้พัฒนาให้สามารถทำงานที่ความถี่สูงได้ และใช้ระบบไปป์ไลน์นิ่ง (Pipelining) ขณะที่ชุดคำสั่งก็มีเพิ่มมากขึ้น และยังมีส่วนจัดการหน่วยความจำ (Memory Management Unit หรือ MMU) ซึ่งมีที่ว่างทั้งแบบ 1 เมกกะไบท์ และแบบ 512 กิโลไบท์

ค่าใช้จ่ายของระบบจะลดลงเนื่องจากว่า ฟังก์ชันการที่งานที่สำคัญพร้อมทั้งส่วน MMU ได้ถูกรวบรวมไว้อยู่ในชิพแล้ว เช่น มีตัวควบคุมการทำ DMA 2 แชนแนล (Direet Memory Access Controller หรือ DMAC) , ตัวสร้างสภาวะรอ (Wait state genererator) , การทำรีเฟรชไดนามิกแรม (dynamic Ram refresh) , มีส่วนอินเทอร์เฟสในการสื่อสารข้อมูลอนุกรมแบบอะซิงค์โครนัส 2 แชนแนล (Asynchronous Serial Communication Interface หรือ ASCI) , มีส่วนสัญญาณนาฬิกา สำหรับพอร์ท I/O แบบอนุกรม (Clocked Serial I/O port หรือ CS I/O) ,มีตัวนับเวลาแบบรีโหลด ซึ่งสามารถโปรแกรมได้ 2 แชนแนล (Programmable Reload Timer

หรือ PRT) , ตัวควบคุมการอินเทอร์รัพ 12 ชนิด และการอินเทอร์เฟสกับบัส 2 แบบ คือแบบ 80xx และ 68xx ในโหมดการทำงานแบบกินไฟน้อย (Low Power Consumption) เราสามารถใช้ซอฟแวร์ควบคุมให้ CPU ทำงานได้โดยกินไฟน้อยมาก

ไมโครโปรเซสเซอร์เบอร์นี้ จะมีประโยชน์ในการในลักษณะที่ต้องการประสิทธิภาพการทำงานสูง กินไฟน้อยและซอฟแวร์ที่สามารถใช้ร่วมกับซอฟแวร์มาตรฐานอื่น ๆ

* HD 64180Z จะใช้ร่วมกันกับ (Compatible) Z 80180 (Z180) ซึ่งผลิตและจำหน่ายโดยบริษัท Zilog ได้ *

ส่วนสำคัญของไมโครโปรเซสเซอร์ HD 64180

- ความถี่ในการทำงานสูงถึง 8 MHz
- ส่วน MMU ซึ่งมีหน่วยความจำ 2 ขนาด คือ 1 เมกกะไบท์ และ 512 กิโลไบท์
- ส่วน DMAC 2 แชนแนล ซึ่งควบคุมการถ่ายโอน (Tranfer) ข้อมูลระหว่างหน่วยความจำกับหน่วยความจำ , หน่วยความจำกับ I/O และหน่วยความจำกับหน่วยความจำแมปป์ (Memory Mapped) I/O
- มีขา WAIT และตัวสร้างสภาวะรอ สำหรับอุปกรณ์อินเตอร์เฟสที่ทำงานช้า
- สามารถโปรแกรมการทำรีเฟรชไดนามิกแรม (Refresh DRAM)
- มีส่วนอินเตอร์เฟสสำหรับการสื่อสารข้อมูลอนุกรมแบบอะซิงค์โครนัส 2 ช่องสัญญาณ ซึ่งสามารถโปรแกรมอัตราโบด(Buad Rate) และสัญญาณแฮนด์เชคกิ้ง(Handshaking)
- มีส่วนสัญญาณนาฬิกาสำหรับพอร์ท I/O แบบอนุกรม ซึ่งทำงานด้วยความเร็วสูง (400 กิโลบิทต่อวินาที ที่สัญญาณนาฬิกาของระบบ 8 เมกกะเฮิทซ์)
- มีส่วนตั้งเวลา(Timer) 16 บิท แบบรีโหลดซึ่งสามารถโปรแกรมได้
- ส่วนควบคุมการอินเทอร์รัพจาก 12 แหล่ง โดยที่จะเกิดจากการอินเทอร์รัพภายใน 8 แหล่ง และอีก 4 แหล่งจะเกิดจากการอินเตอร์รัพภายนอก

- มีบัส 2 แบบใช้ในการอินเตอร์เฟสกับหน่วยความจำและอุปกรณ์มาตรฐานอื่น ๆ
- ส่วนการสร้างสัญญาณนาฬิกา (Clock Generator) บนชิฟ

ส่วนสถาปัตยกรรมทางด้านซอฟแวร์

- สามารถใช้งานร่วมกับ CP/M-80 , CP/M PLUS
- เพิ่มชุดคำสั่งอีก 7 คำสั่ง รวมทั้งคำสั่งเกี่ยวกับการคูณ
- สามารถทำ I/O Address Relocation รีจิสเตอร์บนชิฟ
- มีโหมด SLEEP และ SYSTEM STOP สำหรับใช้ในการทำงานที่กินไฟต่ำ

ทางด้านเทคโนโลยี CMOS

- กินไฟในการปฏิบัติงานต่ำ

  100 mW ขณะที่ทำงาน 8 เมกกะเฮิทซ์

  25 mW ขณะที่อยู่ในโหมด SYSTEM STOP (ทำงานที่ 8 เมกกาเฮริทซ์)

- V_{cc} = 5 V $\pm$ 10 %

2.2.1. ลักษณะโดยรวมของ HD 64180

2.2.1.1 บล๊อกสัญญาณ

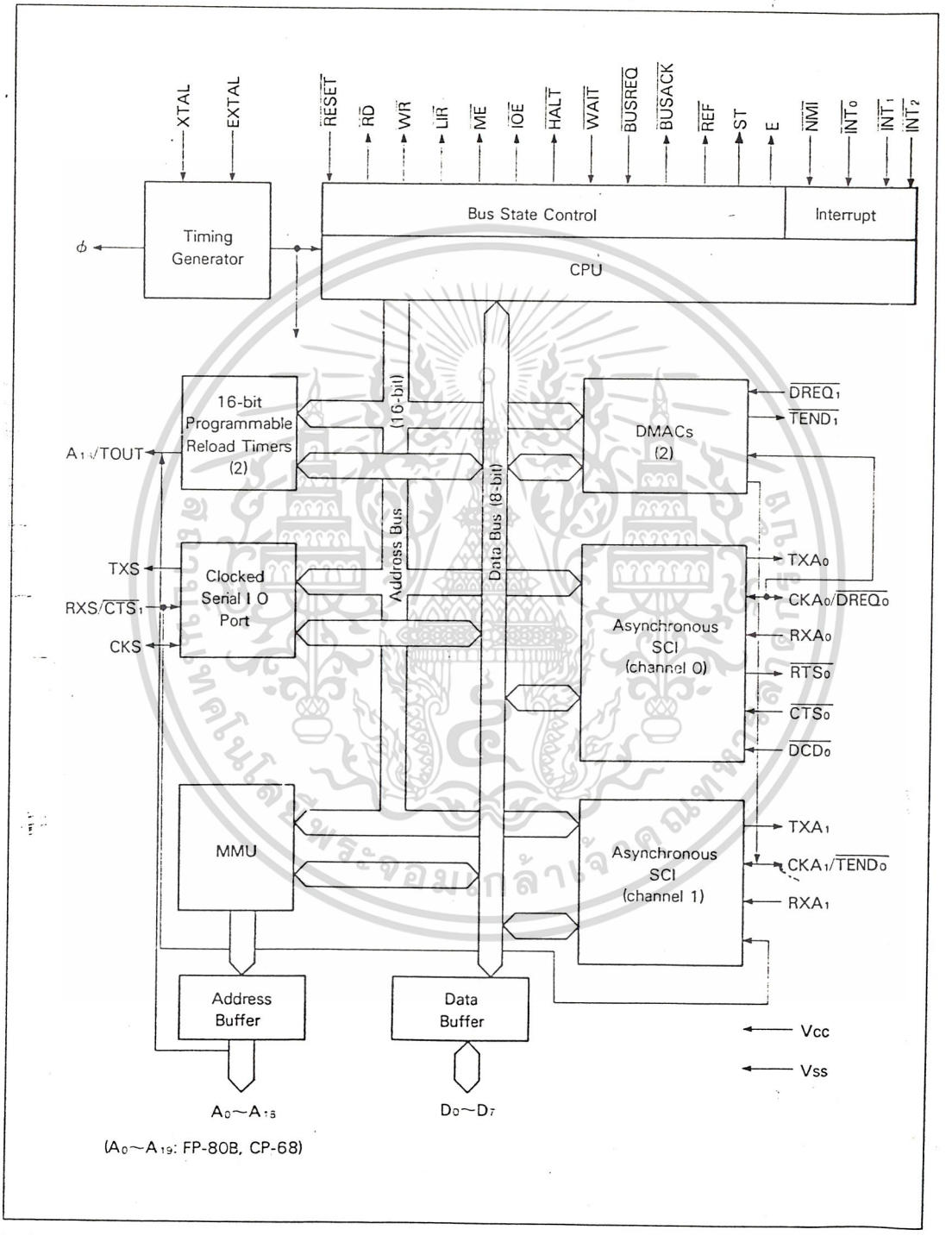
HD64180 จะเป็นการเชื่อมระหว่าง CPU ที่มีประสิทธิภาพการทำงานสูงกับอุปกรณ์ I/O ซึ่งใช้ในงานต่าง ๆ ไว้บนชิพเดียว

ส่วน CPU จะประกอบด้วย 5 บล๊อก ที่สำคัญ

- ตัวสร้างสัญญาณนาฬิกา (Clock Generator)
- ตัวควบคุมสภาวะบัส (Bus State Controller)
- ตัวควบคุมการอินเตอร์รัพ (Interrupt Controller)
- ส่วนจัดการหน่วยความจำ (MMU)
- ส่วนประมวลผลกลาง (CPU)

ส่วนอุปกรณ์ I/O จะประกอบด้วย 4 ฟังก์ชันการทำงาน

- ตัวควบคุมการทำงาน DMA (DMAC 2 แชนแนล)
- ส่วนอินเตอร์เฟสสำหรับการสื่อสารข้อมูล แบบอะซิงค์โครนัส (ASCI 2 แชนแนล)
- สัญญาณนาฬิกาสำหรับพอร์ทอนุกรม I/O (CSI/O 1 แชนแนล)
- ตัวตั้งเวลาแบบรีโหลดซึ่งสามารถโปรแกรมได้ (PRT 2 แชนแนล)



รูปที่ 2.2.1.1 บล็อกสัญญาณของ HD 64180Z

2.2.1.2 การจัดวางขาสัญญาณ

HD 6418OZ

2.2.1.3 สถาปัตยกรรมของ CPU

ตัวสร้างสัญญาณนาฬิกา (Clock generator)

เป็นตัวสร้างสัญญาณนาฬิกาให้ระบบ(ϕ)จากผลึกแร่ความถี่ (Crystal) ซึ่งสามารถใช้ทางสัญญาณนาฬิกาให้กับอุปกรณ์ I/O และอุปกรณ์อื่น ๆ ได้

ตัวควบคุมสภาวะของบัส (Bus State Controller)

เป็นตัวกระทำการที่เกี่ยวกับสภาวะ และการควบคุมบัสทั้งหมด เช่นสัญญาณบัสในสภาวะรอ , สภาวะรีเซท , สภาวะการทำรีเฟรชไดนามิกแรม และสภาวะการทำ DMA และยังสร้างสัญญาณควบคุมบัสเพื่อให้ใช้งานร่วมกับอุปกรณ์รอบข้างอื่น ๆ ได้

ตัวควบคุมการอินเตอร์รัพ (Interrupt Controller)

เป็นตัวจัดลำดับการอินเตอร์รัพจาก 12 แหล่ง ซึ่งเป็นจากภายใน 8 แหล่งและจากภายนอก 4 แหล่ง และยังสามารถโปรแกรมการตอบสนองการอินเตอร์รันได้

ส่วนจัดการหน่วยความจำ (MMU)

เป็นส่วนที่ทำการแปลง พื้นที่ว่างทางโลจิคัล (Logical Address Space) ขนาด 64 กิโลไบท์ ไปเป็นพื้นที่ว่างทางฟิสิกคัล (Physical Address Space) ขนาด 1 เมกกะไบท์ หรือ 512 กิโลไบท์ และเป็นส่วนจัดการติดต่อกับ I/O

ส่วนประมวลผลกลาง (CPU)

ทำงานด้วยไมโครโค๊ด ซึ่งชุดคำสั่งเป็นมาตรฐาน คำสั่งหลายคำสั่งจะใช้จำนวนสัญญาณนาฬิกาที่น้อยกว่า และยังเพิ่มเติมอีก 7 คำสั่ง

2.2.1.4 อุปกรณ์ I/O

ตัวควบคุมการทำ DMA

มีตัวควบคุมการทำ DMA 2 แชนแนล ซึ่งใช้ในการถ่ายโอนข้อมูลได้ด้วยความเร็วสูง ระหว่างหน่วยความจำกับหน่วยความจำ , หน่วยความจำกับ I/O , หน่วยความจำ กับหน่วยความจำแม็บป์ I/O (Memory mapped I/O) ตัวควบคุมการทำ DMA จะมีการตรวจสอบสัญญาน 2 แบบให้เลือก คือ ตรวจสอบที่ขอบขา (Edge Sense)หรือ ตรวจสอบที่ระดับสัญญาณ (Level Sense) DMAC สามารถติดต่อหน่วยความจำได้เต็มทั้งแบบ 1 เมกกะไบท์ และแบบ 512 กิโลไบท์ และสามารถถ่ายโอนข้อมูลได้ครั้งละ 64 กิโลไบท์

ส่วนอินเตอร์เฟสสำหรับการสื่อสารข้อมูลแบบอะซิงโครนัส (ASCI)

ตัว ASCI จะแบ่งออกเป็น ตัว UART (พอร์ทอนุกรม) แบบฟูลดูเพล็ก 2 ตัว และตัวโปรแกรมอัตราโบด (Buad Rate) , สัญญาณการควบคุมโมเด็ม , และการใช้งานในลักษณะมัลติโปรเซสเซอร์ (Multiprocessor) ASCI สามารถใช้ DMAC ควบคุมการถ่ายโอนข้อมูลแบบอนุกรมความเร็วสูง

สัญญาณนาฬิกาสำหรับพอร์ทอนุกรม I/O (CSI/O)

CSI/O จะสร้างสัญญาณนาฬิกาสำหรับการรับส่งแบบผลักกันรับส่ง (Half Duplex) สำหรับการเชื่อมต่อกับไมโครโปรเซสเซอร์ หรือ ไมโครคอมพิวเตอร์

ตัวตั้งเวลาแบบรีโหลด (PRT)

PRT 2 แชนแนล โดยที่แต่ละแชนแนลจะประกอบด้วย รีจิสเตอร์ 16 บิท สำหรับการตั้งเวลาและรีจิสเตอร์ 16 บิท สำหรับข้อมูลรีโหลด เวลาจะถูกหารด้วย 20 จากสัญญาณนาฬิกาของระบบและ PRT แชนแนลที่ 1 จะสามารถใช้ในการสร้างสัญญาณเอาท์พุทได้

2.2.2. สถาปัตยกรรมด้านฮาร์ดแวร์ของ HD 64180

2.2.2.1 คำอธิบายเกี่ยวกับขาสัญญาณต่าง ๆ

XTAL (IN)

- สำหรับต่อกับผลึกแร่ความถี่ (Crystal) ขานี้จะปล่อยลอยไว้ ถ้าเราใช้สัญญาณนาฬิกา TTL จากภายนอก

EXTAL (IN)

- สำหรับต่อกับผลึกแร่ความถี่ (Crystal) สัญญาณนาฬิกา TTL จากภายนอก จะถูกป้อนเข้าที่ขานี้ ซึ่งอินพุทนี้จะถูกปรับปรุงสัญญาณโดยผ่าน Schmit trigger ภายใน

ϕ (OUT)

- สัญญาณนาฬิกาของระบบ , ความถี่ที่ได้ก็คือ ความถี่ครึ่งหนึ่งของตัวกำเนิดสัญญาณ

RESET : CPU Reset (IN)

- ขณะที่สัญญาณขานี้เป็น LOW จะเป็นการให้ค่าเริ่มต้นใหม่แก่ HD64180 สัญญาณเอาท์พุทของทุกขาจะอยู่ในสภาวะไม่ทำงานใช้ในขณะ RESET

A_0-A_{19} : Address Bus (OUT,3-STATE)

- บัสแอดเดรส 3 สภาวะระหว่างRESET บัสแอดเดรสจะอยู่ในสภาวะHigh Impedance

A_{18}/TOUT

- เป็นขาที่ใช้งานร่วมกันระหว่าง แอดเดรสและสัญญาณเอาท์พุทของ PRT แชนแนลที่ 1

ขณะที่เกิดการ RESET ขานี้ จะถูกใช้งานเป็นแอดเดรสถ้าต้องการจะใช้เป็นสัญญาณนาฬิกาจาก PRT แชนแนลที่ต้องควบคุมโดยซอฟแวร์

D_0-D_7 : Data Bus (IN/OUT,3-STATE)

- เป็นบัสข้อมูล 2 ทิศทางเมื่อมีการ RESET บัสข้อมูลจะอยู่ในสภาวะ High Impedance ถ้าต้องการขอใช้บัสก็ส่งสัญญาณ BUSREQ และ BUSACK ให้เป็น LOW

RD : Read (OUT,3-STATE)

- ใช้ระหว่างไซเคิลการอ่าน เพื่อที่จะสามารถรับข้อมูลจากหน่วยความจำภายนอก หรืออุปกรณ์ I/O มาที่บัสข้อมูล

WR : Write (OUT,3-STATE)

- ใช้ระหว่างไซเคิลการเขียน เพื่อที่จะสามารถส่งข้อมูลจากบัสข้อมูลไปที่หน่วยความจำภายนอก หรือ อุปกรณ์ I/O

ME : Memory Enagble (OUT,3-STATE)

- เพื่อแสดงว่า CPU กำลังปฏิบัติงาน การอ่าน หรือเขียนระหว่างหน่วยความจำอยู่ขานี้จะเป็น LOW ก็ต่อเมื่อ
  - a) กำลังทำการเฟทช์คำสั่ง และ Operand
  - b) กำลังอ่านหรือเขียนข้อมูลระหว่างหน่วยความจำ
  - c) ระหว่างไซเคิลการติดต่อหน่วยความจำของการทำ DMA
  - d) ระหว่างไซเคิลการทำรีเฟรช DRAM

IOE : I/O Enable (OUT,3-STATE)

- เพื่อแสดงว่า CPU กำลังปฏิบัติงาน การอ่านหรือเขียนระหว่าง I/O ขานี้ จะเป็น LOW ก็ต่อเมื่อ

  a) กำลังทำการอ่านหรือเขียนข้อมูลกับ I/O

  b) ระหว่างไซเคิลการทำ DMA กับ I/O

  c) ระหว่างไซเคิลการตอบรับการอินเทอร์รัพ INT

WAIT : Bus Cycle Wait (IN)

- เป็นสัญญาณขาอินพุทที่รับเข้ามาเพื่อขยายไซเคิลของการทำงาน คือว่าขณะที่ขอบขาลงของสัญญาณนาฬิกาลูกที่ 2 CPU จะทำการตรวจสอบว่าสภาวะของขาสัญญาณนี้เป็น LOW หรือไม่ ถ้าเป็น LOW ก็จะทำการเพิ่มสภาวะรอ (TW) และจะตรวจสอบที่ขอบขาลงของ Tw จนกระทั่งสัญญาณที่ขา WAIT เป็น HIGH จึงจะทำงานต่อไป

E : Enable (OUT)

- สร้างสัญญาณนาฬิกา เพื่อซิงค์โครนัสสำหรับการเชื่อมต่อกับตระกูล HD63xx และตระกูล 6800/6500 และอุปกรณ์รอบข้าง (Peripheral) อื่น ๆ

BUSREQ : Bus Request (IN)

- เป็นขาสัญญาณเพื่อร้องขอการใช้บัส ขณะที่ขานี้มีสภาวะเป็น LOW CPUจะหยุดทำงานและบัสแอดเดรส , บัสข้อมูล , ขา RD , ขา WR , ME และ IOE จะอยู่ในสภาวะ High Impedance

BUSACK : Bus Acknowledge (OUT)

- เมื่อ CPU ได้ปลดปล่อยบัสออกจากระบบเรียบร้อยแล้วสัญญาณที่ขานี้จะเป็น LOW เพื่อให้อุปกรณ์ที่ขอใช้บัสมานั้นเข้าควบคุมบัสแทน

HALT : Halt/Sleep Status (OUT)

- สัญญาณที่ขานี้จะเป็น LOW หลังจาก CPU ทำคำสั่ง HALT หรือ SLP จะใช้ขานี้ร่วมกับขา LIR และ ST เป็นตัวแสดงสภาวะของ CPU

LIR : Load Instruction Register (OUT)

- สัญญาณที่ขานี้จะเป็น LOW ขณะที่มีการเฟทช์ไซเคิล เราจะใช้ขานี้ร่วมกับขา HALT และขา ST เพื่อแสดงสภาวะของ CPU

ST : Status (OUT)

- ใช้งานร่วมกับขา HALT และ LIR เพื่อแสดงสภาวะของ CPU

* อย่าต่อขานี้ในลักษณะ pull down (ต่อเชื่อมกับความต้านทานลงกราวด์)*

ตารางที่ 2.2.2.1 สรุปสภาวะการทำงาน

ST	HALT	LIR	OPERATION (การปฏิบัติการ)
0	1	0	CPU ปฏิบัติงาน เฟทออปโคดที่ 1
1	1	0	CPU ปฏิบัติงาน เฟทออปโคดที่ 2 และ 3
1	1	1	CPU กำลังปฏิบัติงาน (Machine cycle)
0	X	1	กำลังทำ DMA
0	0	0	ทำงานในโหมด HALT
1	0	1	ทำงานในโหมด SLEEP รวมไปถึง SYSTEM STOP

NOTE X : Don't care

MC : Machine cycle

REF : Refresh (OUT)

- เป็น LOW เพื่อแสดงว่า CPU อยู่ในไซเคิลการรีเฟรชไดนามิค RAM และขาแอดเดรส 8 บิทล่าง (A_0-A_7) จะเป็นแอดเดรสของการรีเฟรช

NMI : Non Maskable Interupt (IN)

- เมื่อมีการเปลี่ยนแปลงระดับสัญญาณจาก HIGH ไปเป็น LOW ที่ขาสัญญาณนี้จะทำให้ CPU ทำการเก็บข้อมูลต่างๆ และจะไปทำงานที่เซอร์วิสรูทีน ที่แอดเดรส 0066H ข้อมูลต่างๆ สามารถเรียกกลับคืนมาได้ โดยคำสั่ง RETN (Return from Non-maskable Interrupt)

INT_0 : Maskable Interrupt ระดับ 0 (IN)

- ถ้าเรากำหนดให้มีการตอบสนองการอินเตอร์รัพท์โดยซอฟแวร์ เมื่อขาสัญญาณนี้เป็น LOW จะทำให้เกิดการอินเตอร์รัพขึ้น CPU จะทำการเก็บข้อมูลต่างๆ การตอบสนองการอินเตอร์รัพ INT_0 จะมีทางเลือกอยู่ 3 ทางซึ่งสามารถโปรแกรมได้โดยซอฟแวร์

โหมด	การทำงาน
0	คำสั่งจะถูกเฟทช์และทำงานจากบัสข้อมูล
1	คำสั่งจะถูกเฟทช์และทำงานจากแอดเดรส 0038H
2	จะใช้เวกเตอร์ อินเทอร์รัพของระบบ โดยเวกเตอร์แอดเดรส 8 บิทล่าง เฟทช์จากบัสข้อมูล

ในทุกๆโหมด เราสามารถเรียกข้อมูลกลับมาได้ โดยทำคำสั่ง RETI (Return from Interrupt)

INT_1 , INT_2 : Maskable Interrupt ระดับ 1,2 (IN)

- ถ้ามีการกำหนดให้มีการตอบสนองการอินเทอร์รัพได้ (Maskable) กำหนดจากซอฟแวร์ เมื่อขาสัญญาณนี้เป็น LOW จะทำให้เกิดการอินเทอร์รัพขึ้น CPU จะเก็บข้อมูลต่างๆไว้ และจะไปทำงานที่เซอร์วิสรูทีน โดยใช้อินเทอร์รัพเวกเตอร์ของระบบคล้ายกับการ INT_0 ในโหมดที่ 2

$DREQ_0$: DMA Requst แชนแนล 0 (IN)

- เมื่อขาสัญญาณนี้ถูกตรวจพบว่าเป็น LOW ตัวDMAC จะจัดการให้มีการทำ DMAแชนแนล 0 โดยจะยอมให้มีการถ่ายโอนข้อมูลจากหน่วยความจำกับ I/O DREQ ϕ จะไม่สามารถ

ใช้งานในการถ่ายโอนข้อมูลจากหน่วยความจำกับหน่วยความจำ และ ขานี้ยังใช้งานเป็น CKA_0 ด้วย

$TEND_0$ - Transfer End แชนแนล 0 (OUT)

- เมื่อถึงไซเคิลสุดท้ายการเขียนในการทำ DMA แชนแนล 0 CPU จะทำให้สัญญาณที่ขา $TEND_0$ เป็น LOW ที่ขานี้ขึ้นมาเพื่อแสดงว่าการทำ DMA กับอุปกรณ์ภายนอกเสร็จสมบูรณ์ และขานี้ยังใช้งานเป็นขา CKA_1 ด้วย

$DREQ_1$: DMA Request แชนแนล 1 (IN)

- เมื่อขาสัญญาณนี้ถูกตรวจสอบว่าเป็น LOW ตัว DMAC จะจัดการให้มีการทำ DMA แชนแนล 1 โดยจะยอมให้มีการถ่ายโอนข้อมูลจากหน่วยความจำ กับ I/O เท่านั้น

$TEND_1$: Transfer End แชนแนล 1 (OUT)

- เมื่อถึงไซเคิลสุดท้ายการเขียนในการทำ DMA แชนแนล 1 CPU จะทำให้สัญญาณที่ขา $TEND_1$ เป็น LOW เพื่อแสดงว่าการทำ DMA กับอุปกรณ์ภายนอกเสร็จสมบูรณ์

TXA_0 : Asynchronous Transmit Data แชนแนล 0 (OUT)

- ขาสำหรับส่งข้อมูลอนุกรมแบบอะซิงโครนัส ASCI แชนแนล 0

RXA_0 : Asynchronous Recieve Data แชนแนล 0 (IN)

- ขาสำหรับรับข้อมูลอนุกรมแบบอะซิงโครนัส ASCI แชนแนล 0

CKA_0 : Asynchronous Clock แชนแนล 0 (IN/OUT)

- เป็นขาสัญญาณนาฬิกาของการรับส่งข้อมูลแชนแนล 0 ขานี้ใช้งานร่วมกับขา $DREQ_0$ (กำหนดจากซอฟแวร์)

RST_0 : Request to send แชนแนล 0 (OUT)

- เป็นสัญญาณควบคุมโมเด็มในการที่จะส่งข้อมูลของ ASCI แชนแนล 0

CTS_0 : Clear to send แชนแนล 0 (IN)

- เป็นขาสัญญาณอินพุท เพื่อรับสัญญาณตอบกลับให้มีการส่งข้อมูลได้ สำหรับแชนแนล 0

DCD_0 : Data carrier Detect แชนแนล 1 (OUT)

- ขาสัญญาณอินพุท เป็นสัญญาณควบคุมโมเด็ม

TXA_1 : Asynchronous Transmit data แชนแนล 1 (OUT)

- ASCI แชนแนล 1 ที่ใช้ในการส่งข้อมูลอนุกรม แบบอะซิงโครนัส

RX1 : Asynchronous Receive Data แชนแนล 1 (IN)

- ASCI แชนแนล 1 ที่ใช้ในการรับข้อมูลอนุกรมแบบอะซิงโครนัส

CKA_1 : Asynchronous Clock แชนแนล 1 (IN/OUT)

- สัญญาณนาฬิกาในการรับส่งข้อมูลอนุกรมแชนแนล 1 แบบอะซิงโครนัส ขานี้ใช้ร่วมเป็น (Multiplex) ขา $TEND_0$ ด้วย

CTS_1 : Clear to send แชนแนล 1 (IN)

- เป็นขาสัญญาณอินพุทเพื่อรับสัญญาณตอบกลับจากโมเด็ม ให้มีการส่งข้อมูลได้สำหรับ ASCI แชนแนล 1 ขานี้ใช้ร่วมเป็นขา RXS

TXS : Clocked Serial Transmit Data (OUT)

- ขาสัญญาณนาฬิกา สำหรับการส่งข้อมูลอนุกรม ซึ่งมาจาก พอร์ท CSI/O

RXS : Clock Serial Receive Data (IN)

- ขาสัญญาณนาฬิกา สำหรับการรับข้อมูลอนุกรม ซึ่งมาจาก CSI/O ซึ่งขานี้สามารถใช้งานเป็นขา CTS1 ด้วย

CKS : Serial Clock (IN/OUT)

- สัญญาณนาฬิกาอินพุท เอาท์พุท สำหรับ CSI/O

TOUT : Timer Output (OUT)

- เป็นตัวตั้งเวลาจะให้เอาท์พุทเป็น พัลส์ (Pulse) จากตัว PRT แชนแนล 1 ขานี้จะใช้เป็นขา A_{18} ด้วย

Vcc - แหล่งจ่ายไฟ

Vss - กราวด์

คำอธิบายเกี่ยวกับ ขาที่มีการใช้งานหลายหน้าที่

A_{18}/TOUT - ระหว่างที่ RESET การทำงานของขานี้จะเป็นขา A_{18} แต่ถ้าบิท TOC_1 หรือ TOC_0 ใน TCR (Timer Control Register) ถูกเซตเป็น 1 ขานี้ จะเป็นขา TOUT

CKA_0/$dREQ_0$ - ระหว่างที่ RESET การทำงานของขานี้จะเป็นขา CKA_0 แต่ถ้าบิท DM1 หรือ SM1 ใน DMA Mode Register (DMODE) ถูกเซตเป็น 1 ขานี้จะเป็นขา $DREQ_0$

CKA1/TEND - ระหว่างที่ RESET การทำงานของขานี้จะเป็น CK1 แต่ถ้าบิท CKA1D ใน ASCI Control Register ch1 (CNTLA1) ถูกเซตเป็น 1 ขานี้ จะทำงานเป็นขา TEND ถ้าบิทนั้นมีค่าเป็น 0 ก็จะทำงานเป็นขา CKA1

RXS/CTS - ระหว่างที่ RESET การทำงานของขานี้เป็น RXS แต่ถ้าบิท CTSIE ASCI ใน Status Register ch1 (STAT1) ถูกเซตเป็น 1 ขานี้ จะทำงานเป็นขา CTS1 ถ้าบิทนั้นมีค่าเป็น 0 จะทำงานเป็นขา RXS

2.2.2.2 รีจิสเตอร์ภายในของ I/O

ในตัวHD64180 จะมีการจองเนื้อที่สำหรับตำแหน่งของ I/O 64 ไบท์ ซึ่งรีจิสเตอร์เหล่านี้ จะใช้สำหรับติดต่อกับโมดูล I/O ภายในต่าง (ASCI ,CSI/O ,PRT) และหน่วยควบคุมต่างๆ (DMAC , ส่วนรีเฟรช DRAM , ส่วนอินเทอร์รัฟ , ส่วนสร้างสภาวะรอ , (MMU)) เพื่อที่จะไม่สับสนในการอ้างตำแหน่ง I/O ภายนอก กับตำแหน่ง I/O ภายใน HD64180 จึงทำการแบ่งเนื้อที่ 64 ไบท์ ล่างสุดของเนื้อที่ I/O ทั้งหมด 64 กิโลไบท์

I/O Control Register (ICR)

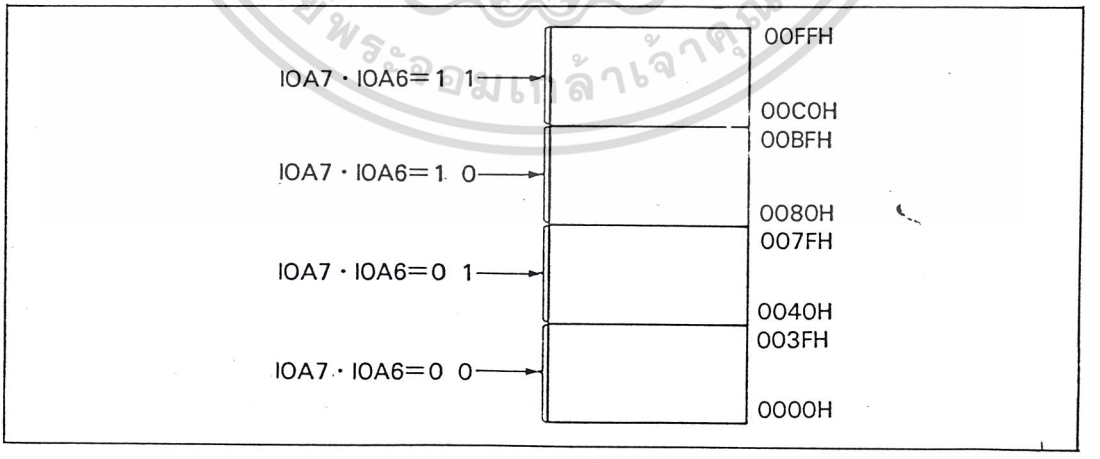
ICR จะวางอยู่ที่ตำแหน่งของชั้นที่ I/O ภายใน ซึ่ง ICR จะทำหน้าที่เอ็นเนเบิ้ล (Enable) หรือ ดิสเอเบิ้ล(Disable) การใช้งานในโหมด IOSTOP

I/O Control Register (ICR : ตำแหน่ง I/O = 3FH)

บิท	7	6	5	4	3	2	1	0
	IOA 7	IOA 6	IOSTP	-	-	-	-	-
	R/W	R/W	R/W					

- IOA 7,6 :I/O Address Relocation (บิท 7,6)

จะเป็นตัวบ่งบอกตำแหน่งของ I/O ภายในที่เลือกใช้งาน ดังรูปที่ 2.2.2.2.1 สังเกตว่า 8 บิทบนของตำแหน่ง I/O ภายในจะเป็น 0 เสมอ ระหว่างที่ทำการ RESET IOA 7 และ IOA 6 จะถูกเคลียร์ค่าเป็น 0



รูปที่ 2.2.2.2.1 การจัดวางตำแหน่งของ I/O ภายใน

- IOSTP : IOSTOP Mode (บิท 5)

เป็นบิทที่ใช้สำหรับการเลือกใช้ในการทำงานโหมด IOSTOP ถ้าบิทนี้ถูกเซตค่าเป็น 1 ในกรณีปกติแล้วบิทนี้จะมีค่าเป็น 0 ระหว่างที่ทำการ RESET บิทนี้จะถูกเคลียร์ค่าเป็น 0

การจัดวางตำแหน่งของรีจิสเตอร์ภายในของ I/O

การจัดตำแหน่งต่างๆของรีจิสเตอร์ภายในของ I/O จะเป็นดังตารางที่ 2.2.2.2.1 ซึ่งตำแหน่งต่างๆจะอยู่ในขอบเขต 64 ไบท์ ซึ่งการระบุในช่วงใดขึ้นกับบิท IOA 6 และ IOA 7 ใน ICR

ตารางที่ 2.2.2.2.1 การจัดวางตำแหน่งของรีจิสเตอร์ภายในของ I/O (1)

	Register	Mnemonic	Address	
			Binary	Hexadecimal
ASCI	ASCI Control Register A Ch 0	CNTLA0	XX000000	00H
	ASCI Control Register A Ch 1	CNTLA1	XX000001	01H
	ASCI Control Register B Ch 0	CNTLB0	XX000010	02H
	ASCI Control Register B Ch 1	CNTLB1	XX000011	03H
	ASCI Status Register Ch 0	STAT0	XX000100	04H
	ASCI Status Register Ch 1	STAT1	XX000101	05H
	ASCI Transmit Data Register Ch 0	TDR0	XX000110	06H
	ASCI Transmit Data Register Ch 1	TDR1	XX000111	07H
	ASCI Receive Data Register Ch 0	RDR0	XX001000	08H
	ASCI Receive Data Register Ch 1	RDR1	XX001001	09H
CSI/O	CSI/O Control Register	CNTR	XX001010	0AH
	CSI/O Transmit/Receive Data Register	TRDR	XX001011	0BH
Timer	Timer Data Register Ch 0L	TMDR0L	XX001100	0CH
	Timer Data Register Ch 0H	TMDR0H	XX001101	0DH
	Reload Register Ch 0L	RLDR0L	XX001110	0EH
	Reload Register Ch 0H	RLDR0H	XX001111	0FH
	Timer Control Register	TCR	XX010000	10H
	Reserved		XX010001	11H
			⟩	⟩
			XX010011	13H
	Timer Data Register Ch 1L	TMDR1L	XX010100	14H
	Timer Data Register Ch 1H	TMDR1H	XX010101	15H
	Reload Register Ch 1L	RLDR1L	XX010110	16H
	Reload Register Ch 1H	RLDR1H	XX010111	17H
Others	Free Running Counter	FRC	XX011000	18H
	Reserved		XX011001	19H
			⟩	⟩
			XX011111	1FH

ตารางที่ 2.2.2.2.1 การจัดวางตำแหน่งรีจิสเตอร์ภายในของ I/O (2)

	Register	Mnemonic	Address	
			Binary	Hexadecimal
DMA	DMA Source Address Register Ch 0L	SAR0L	XX100000	20H
	DMA Source Address Register Ch 0H	SAR0H	XX100001	21H
	DMA Source Address Register Ch 0B	SAR0B	XX100010	22H
	DMA Destination Address Register Ch 0L	DAR0L	XX100011	23H
	DMA Destination Address Register Ch 0H	DAR0H	XX100100	24H
	DMA Destination Address Register Ch 0B	DAR0B	XX100101	25H
	DMA Byte Count Register Ch 0L	BCR0L	XX100110	26H
	DMA Byte Count Register Ch 0H	BCR0H	XX100111	27H
	DMA Memory Address Register Ch 1L	MAR1L	XX101000	28H
	DMA Memory Address Register Ch 1H	MAR1H	XX101001	29H
	DMA Memory Address Register Ch 1B	MAR1B	XX101010	2AH
	DMA I/O Address Register Ch 1L	IAR1L	XX101011	2BH
	DMA I/O Address Register Ch 1H	IAR1H	XX101100	2CH
	Reserved		XX101101	2DH
	DMA Byte Count Register Ch 1L	BCR1L	XX101110	2EH
	DMA Byte Count Register Ch 1H	BCR1H	XX101111	2FH
	DMA Status Register	DSTAT	XX110000	30H
	DMA Mode Register	DMODE	XX110001	31H
	DMA/WAIT Control Register	DCNTL	XX110010	32H
INT	IL Register (Interrupt Vector Low Register)	IL	XX110011	33H
	INT/TRAP Control Register	ITC	XX110100	34H
	Reserved		XX110101	35H

ตารางที่ 2.2.2.2.1 การจัดวางตำแหน่งรีจิสเตอร์ภายในของ I/O (3)

	Register	Mnemonic	Address	
			Binary	Hexadecimal
Refresh	Refresh Control Register	RCR	XX110110	36H
	Reserved		XX110111	37H
MMU	MMU Common Base Register	CBR	XX111000	38H
	MMU Bank Base Register	BBR	XX111001	39H
	MMU Common/Bank Area Register	CBAR	XX111010	3AH
I/O	Reserved		XX111011	3BH
			⟩	⟩
			XX111101	3DH
	Operation Mode Control Register	OMCR	XX111110	3EH
	I/O Control Register	ICR	XX111111	3FH

การอ้างตำแหน่งเกี่ยวกับ I/O

เนื่องจากว่า รีจิสเตอร์ภายในของ I/O จะอยู่ในพื้นที่ของ I/O ทั้งหมด คือ อยู่ระหว่าง 0000A ถึง 00FFH ดังนั้นการอ้างตำแหน่งของรีจิสเตอร์ภายในของ I/O ส่วนหนึ่งเราจะอ้างโดยให้ 8 บิทบนเป็น 0 และจะใช้เพียง 8 บิทล่างแต่ในคำสั่งเกี่ยวกับ I/O ต่างๆ (OUT (m) ,A/IN A ,(m) /OUTI /INT) เป็นต้น) จะใช้การอ้างที่ 8 บิทบนของบัสแอดเดรสด้วย จึงทำให้ยากในการอ้างถึงรีจิสเตอร์ภายในของ I/O HD64180 จึงเพิ่มคำสั่งพิเศษขึ้นมาเพื่อใช้สำหรับติดต่อกับรีจิสเตอร์ภายในของ I/O โดยตรงนั้น ซึ่งจะบังคับให้ 8 บิทบนของ I/O เป็น 0 คำสั่งที่เพิ่มมาคือ INO , OUTO , OTIM , OTIMR , OTDM , OTDMR และ TSTIO

สิ่งที่น่าสังเกตอย่างหนึ่ง เมื่อมีการเขียนข้อมูลลงบนรีจิสเตอร์ภายในของ I/O ก็จะมีสัญญาณการเขียน I/O เหมือนกัน เกิดขึ้นที่บัสภายนอก ซึ่งเป็นการแสดงไซเคิลของการเขียนภายใน ยกตัวอย่างว่า ถ้าการติดต่อกับภายนอกไม่มีสภาวะรอแล้ว ไซเคิลการอ่านจาก I/O ภายในจะเหมือนกับไซเคิลการอ่าน I/O จากภายนอก แต่อย่างไรก็ตามก็มิใช่การอ่านข้อมูลจากภายนอกเข้ามา

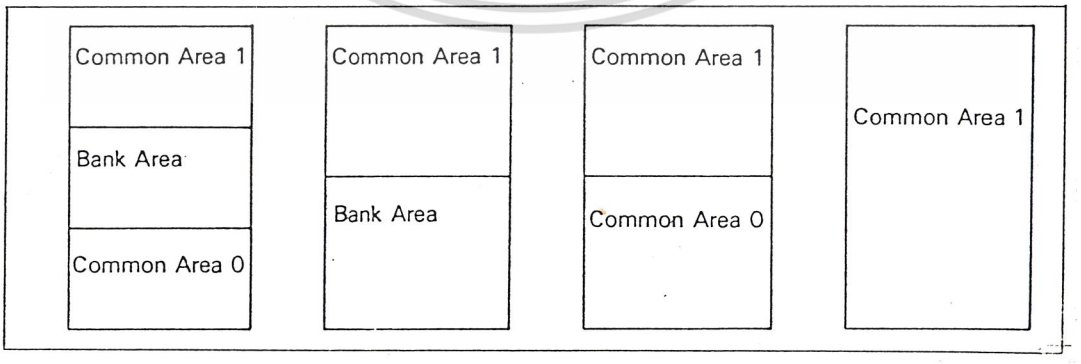
ตามปกติแล้ว ตำแหน่งของ I/O ภายนอกจะหลีกเลี่ยงการทับซ้อน (Overlap) กับตำแหน่งของ I/O ภายในเพื่อป้องกันความสับสน

2.2.2.3 ส่วนจัดการหน่วยความจำ (Memory Mangement Unit หรือ MMU)

ส่วนนี้จะเป็นหน่วยความจำขนาด 64 กิโลไบท์บน CPU (จาก 0000H ถึง FFFFH) ให้เพิ่มขึ้นได้ถึง 512 กิโลไบท์ (00000H ถึง 7FFFFH) หรือเป็น 1 เมกกะไบท์ (จาก 00000H ถึง FFFFFH) ได้

2.2.2.3.1 พื้นที่โลจิคัลแอดเดรส(Logical Address Space)

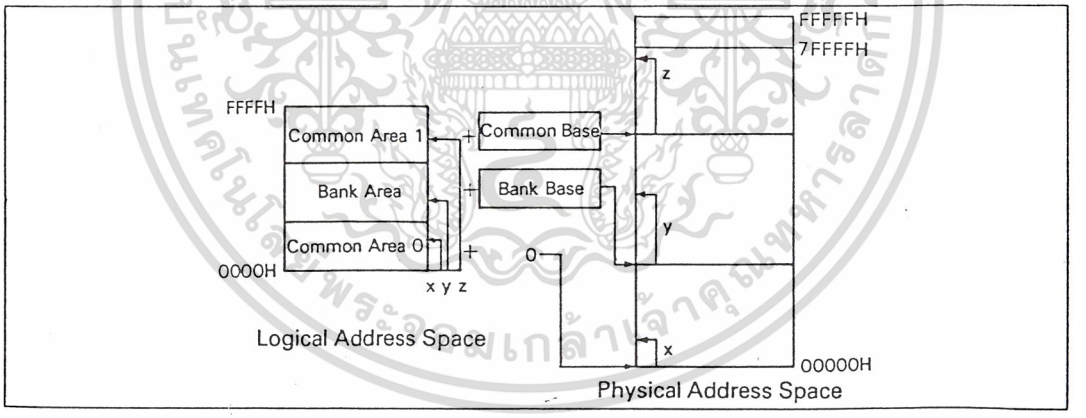
พื้นที่โลจิคัลแอดเดรส 64 กิโลไบท์บน CPU จะแบ่งเป็น 3 ส่วน เป็น Common Area 0 , Bank Area และ Common Area 1 ดังรูปที่ 2.2.2.3.1เป็นการแสดงการจัดวางหน่วยความจำแบบต่าง ๆ



รูปที่ 2.2.2.3.1 ตัวอย่างการจัดวางพื้นที่โลจิคัลบน CPU

2.2.2.3.2 การแปลงโลจิคัลแอดเดรส (Logical Address) เป็นฟิสิกคัลแอดเดรส (Physical Adress)

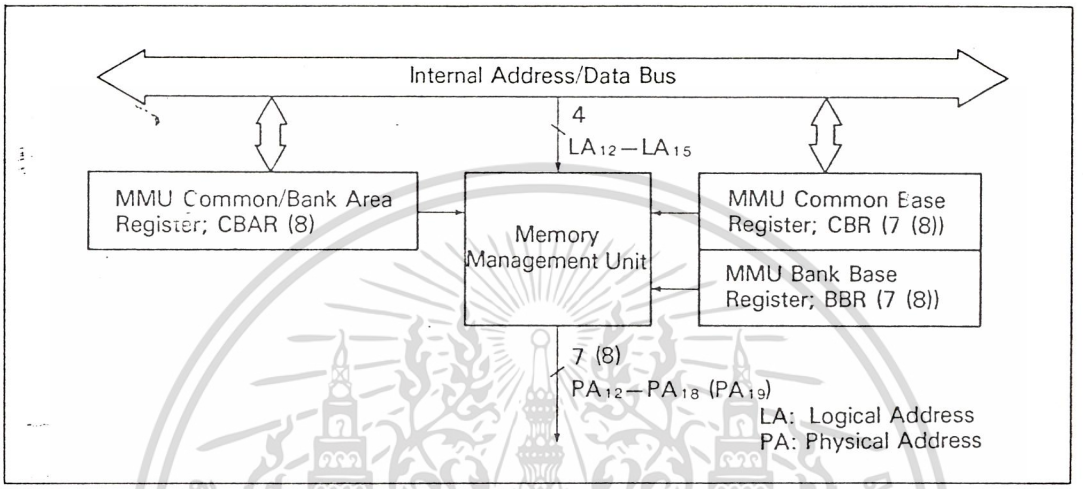
รูปที่ 2.2.2.3.2 แสดงตัวอย่างของการขยายโลจิคัลแอดเดรส 3 ส่วน เพื่อทำเป็นฟิสิกคัลแอดเดรส 512 กิโลไบท์ (1 เมกกะไบท์) จุดที่สำคัญของการขยายนี้ก็คือ ในส่วนของ Common Area และ Bank Area สามารถใช้ทับซ้อน (Overlap) กัน และส่วน Common Area 1 และ Bank Area สามารถที่จะอ้างหน่วยความจำใหม่ได้(ภายในขอบเขต 4 กิโลไบท์ของฟิสิกคัลแอดเดรส) ถ้ามี Common Area 0 จะใช้ส่วนนี้เป็นจุดล่างสุดหน่วยความจำ คือ 00000H การขยายก็ทำโดยนำตำแหน่งของหน่วยความจำรวมกับฐาน (Base) ของส่วนต่างๆ



รูปที่ 2.2.2.3.2 ตัวอย่างการขยายหน่วยความจำบน CPU เพื่อใช้งาน 512 กิโลไบท์

2.2.2.3.3 บล๊อกไดอะแกรมของ MMU

รูปที่ 2.2.2.3.3 แสดงบล๊อกสัญญาณของ MMU ซึ่งแปลงจาก 16 บิทแอดเดรส (64 กิโลไบท์) ทางโลจิกคัล ไปเป็น 19 บิทแอดเดรส (512 กิโลไบท์) หรือ แบบ 20 บิทแอดเดรส (1 เมกกะไบท์) ทางฟิสิกคัล



รูปที่ 2.2.2.3.3 บล๊อกไดอะแกรมของ MMU

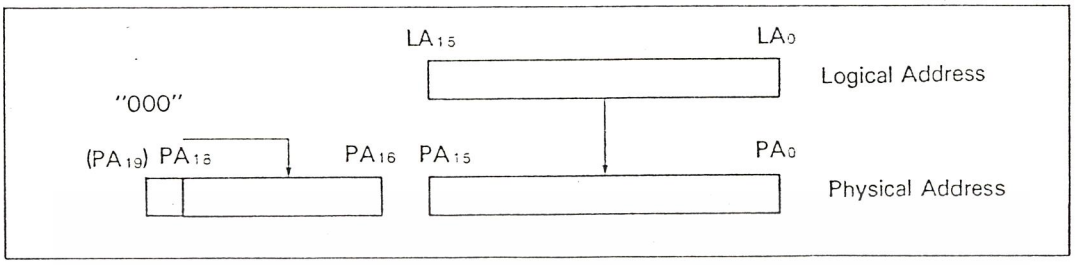
การแปลงหน่วยความจำจะขึ้นอยู่กับชนิดไซเคิลของ CPU ต่างๆ ดังนี้

1. ไซเคิลเกี่ยวกับหน่วยความจำ

การแปลงหน่วยความจำจะเกิดขึ้นในทุกๆ การติดต่อข้อมูลกับหน่วยความจำรวมไปถึงการเฟทช์คำสั่งและตัวดำเนินการ (Operand) การอ่านหรือการเขียนหน่วยความจำ การเฟทช์อินเทอร์รัปเวกเตอร์ และการเริ่มต้นทำงานรูทีนการอินเทอร์รัพ

2. ไซเคิลเกี่ยวกับ I/O

ในการกระทำเกี่ยวกับไซเคิลของ I/O จะไม่มีการแปลงหน่วยความจำ การอ้างหน่วยความจำของ I/O จะใช้บัสแอดเดรส 16 เส้นโดยตรงจาก CPU ส่วน 3 บิทบน (A16 - A18 (A19)) จะเป็น 0 ระหว่างไซเคิลของ I/O



รูปที่ 2.2.2.3.4 การแปลงหน่วยความจำของ I/O

3. ไซเคิลเกี่ยวกับ DMA

เมื่อมีการทำ DMA ตัว DMAC จะใช้บัสภายนอกอยู่แล้ว เราจึงไม่จำเป็นต้องมีการแปลงเพื่อขยายหน่วยความจำ รีจิสเตอร์ขนาด 19 บิท หรือ 20 บิท ในตัว DMAC จะติดต่อโดยตรงกับบัสที่ใช้งาน (A0 - A18 (A19))

2.2.2.3.4 รีจิสเตอร์ต่าง ๆ ของ MMU

ส่วน MMU จะมีรีจิสเตอร์ 3 ตัวซึ่งใช้งานเกี่ยวกับหน่วยความจำ

1. MMU Common / Bank Area Register (CBAR)

2. MMU Common Base Register (CBR)

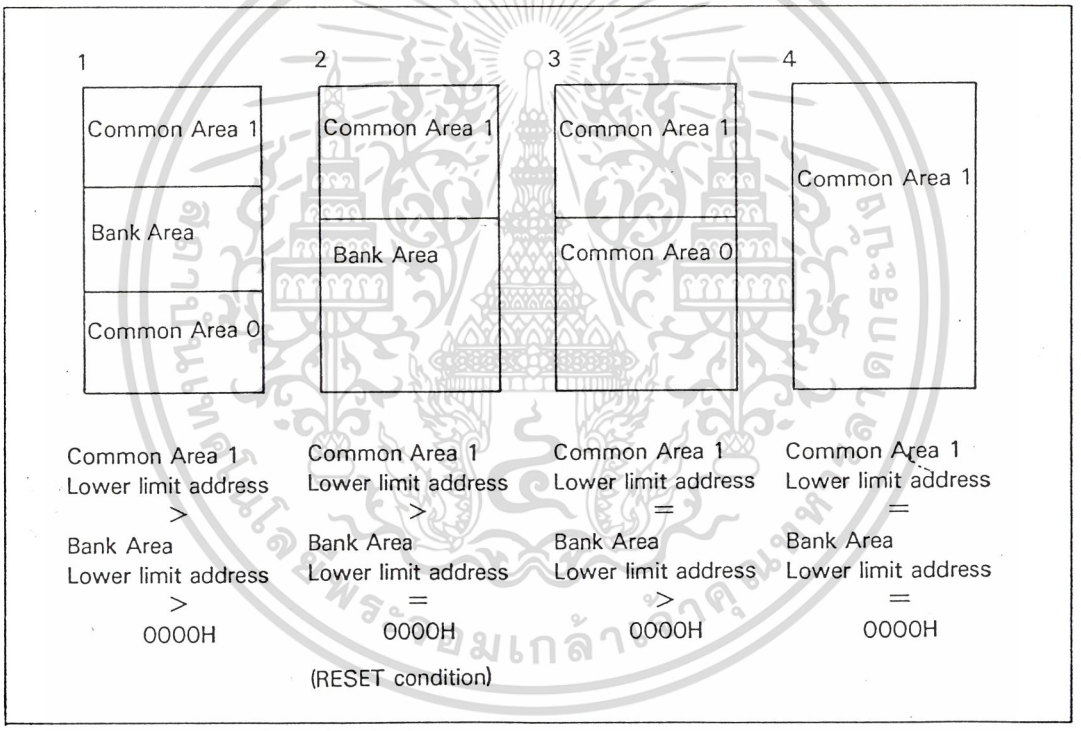
3. MMU Bank Base Area (BBR)

CBAR ใช้ระบุการจัดเรียงหน่วยความจำทางโลจิคัล ขณะที่ CBR และ BBR ใช้ในการอ้างตำแหน่งซึ่งเป็นฐานของพื้นที่ในส่วนนั้นๆ

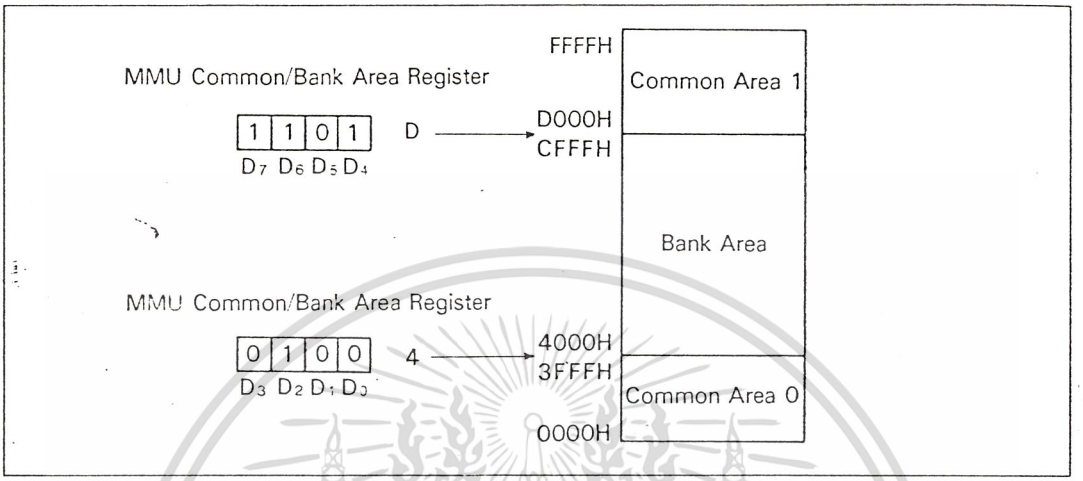
CBAR ยังแยกเป็นส่วนย่อยอีกคือ ส่วน CA (Common Area) ซึ่งใช้เป็นตำแหน่งเริ่มต้นของ Common Area 1 (ส่วนด้านบน) หรือเป็นจุดสิ้นสุดของ Bank Area

ส่วน BA (Base Area) จะเป็นจุดเริ่มต้นของ Bank Area หรือเป็นจุดสิ้นสุดของ Common Area 0 (พื้นที่ด้านล่าง)

ตัว CA และ BA สามารถที่จะโปรแกรมให้มีค่าเท่าไรก็ได้ภายใน 64 กิโลไบท์ โดยมีข้อจำกัดอยู่แต่เพียงว่า (CA จะต้องมากกว่าค่า BA) รูปที่ 2.2.2.3.5 แสดงตัวอย่างการจัดเรียงหน่วยความจำทางโลจิคัล โดยการกำหนดจาก CA และ BA และรูปที่ 2.2.2.3.6 แสดงการกำหนดค่าในรีจิสเตอร์



รูปที่ 2.2.2.3.5 การจัดเรียงหน่วยความจำทางโลจิคัล



รูปที่ 2.2.2.3.6 แสดงการกำหนดค่าในรีจิสเตอร์สำหรับการจัดหน่วยความจำทางโลจิคัล

คำอธิบายเกี่ยวกับรีจิสเตอร์ของ MMU

MMU Common / Bank Area Register (CBAR)

CBAR เป็นรีจิสเตอร์ขนาด 8 บิท ใช้สำหรับกำหนดขอบเขตของพื้นที่ต่างๆ ทางโลจิคัล ซึ่งแบ่งเป็น 3 พื้นที่ Common Area 0 ,Bank Area และ Common Area 1

MMU Common / Bank Area Register (CBAR : ตำแหน่ง I/O = 3AH)

บิท	7	6	5	4	3	2	1	0
	CA 3	CA 2	CA 1	CA 0	BA 3	BA 2	BA 1	BA 0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

- CA0-CA3 (บิท 4 - 7)

เป็นส่วนสำหรับระบุขอบเขตล่างของ Common Area 1 เมื่อมีการ RESET ทุกบิทของ CA เป็น 1

- BA0-BA3 (บิท 0 - 3)

เป็นส่วนสำหรับระบุของเขตล่างของ Bank Area เมื่อมีการ RESET ทุกบิทของ BA จะเป็น 0

MMU Common Base Area Register (CBR)

CBR ใช้ระบุเบสแอดเดรส (Base Address)ในการสร้างฟิสิกคัลแอดเดรส ขนาด 19บิท (20บิท) สำหรับติดต่อ Common Area 1 เมื่อมีการ RESET ทุกบิทของ CBR จะเป็น 0

MMU Common Base Register (CBR : ตำแหน่ง I/O = 38H)

บิท	7	6	5	4	3	2	1	0
	CB7	CB6	CB5	CB4	CB3	CB2	CB1	CB0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

MMU Bank Base Register (BBR)

BBR ใช้ระบุเบสแอดเดรส(Base Address) ในการสร้างฟิสิกคัลแอดเดรสขนาด 19บิท (20บิท)สำหรับติดต่อ Base Area เมื่อมีการ RESET ทุกบิทของ BBR จะเป็น 0

MMU Bank Base Area (BBR : ตำแหน่ง I/O = 39H)

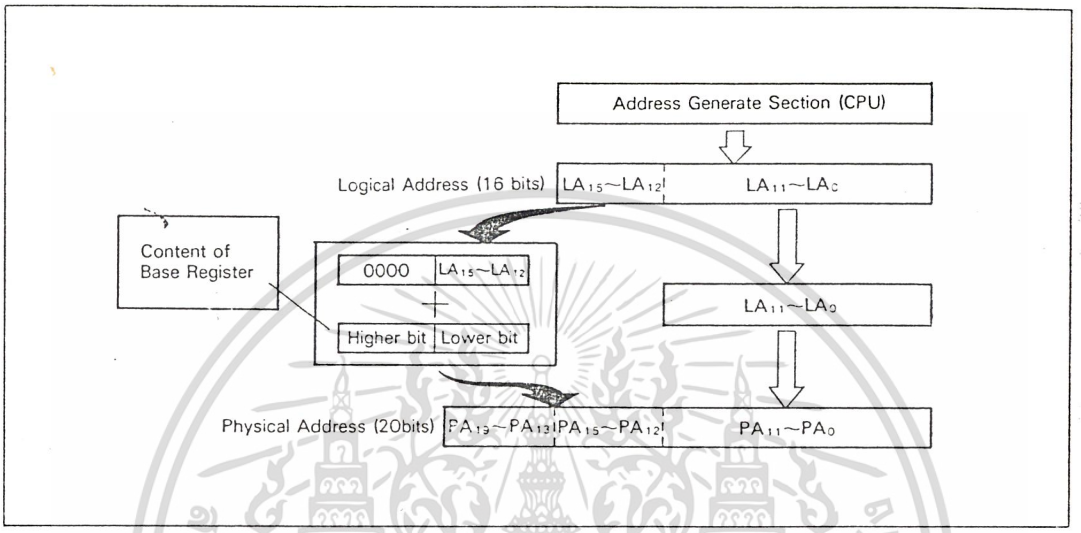
บิท	7	6	5	4	3	2	1	0
	BB7	BB6	BB5	BB4	BB3	BB2	BB1	BB0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

2.2.2.3.5 การแปลงเป็น ฟิสิกคัลแอดเดรส(Physical address Translation)

รูปที่ 2.2.2.3.7 แสดงการแปลงเป็นฟิสิกคัลแอดเดรส ซึ่งใช้รีจิสเตอร์ที่เกี่ยวข้อง 3 ตัว คือ (BAR ,CBR ,BBR) MMU จะตรวจสอบว่าจะติดต่อกับโลจิกคัลแอดเดรสส่วนใดจาก CBAR โดยที่จะนำ เบสแอดเดรส (Base address) 7 บิท รวม 4 บิทบนของโลจิคัลแอดเดรส (Logical Address) ก็จะได้ ฟิสิกคัลแอดเดรส 19 บิท (20บิท) ค่าใน CBR จะใช้เป็นเบสแอดเดรสของส่วน Common Area 1 ส่วน Common Area 0 จะใช้เบสแอดเดรสที่ 00000 H รูปที่ 2.2.2.3.7 จะแสดงการแปลงจากโลจิคัลแอดเดรส (Logical Address) เป็น ฟิสิกคัลแอดเดรส (Physical Address)

2.2.2.3.6 MMU และการ RESET

ระหว่างที่ทำการ RESET ทุกบิทของ CA จะเป็น 1 หมด ส่วน BA จะเป็น 0 ใน CBAR ส่วน CBR และ BBR จะเป็น 0 หมดทุกบิท เพราะฉะนั้นหลังการ RESET เราสามารถอ้างตำแหน่งบน CPU ได้โดยตรงกับตำแหน่งที่ได้ใช้งานจริงๆ สำหรับ 64 กิโลไบท์แรก



รูปที่ 2.2.2.3.7 การสร้างฟิสิกคัลแอดเดรส (Physical Address Generation)

2.2.2.3.7 เวลาของการติดต่อรีจิสเตอร์ของ MMU

เมื่อข้อมูลถูกเขียนไปที่ CBAR , CBR หรือ BBR ค่าที่เขียนใหม่จะมีผลทันทีตามไซเคิลการเขียน I/O

การเขียนโปรแกรมควรแน่ใจว่า การทำงานของ CPU ไม่ถูกรบกวน เทคนิคง่ายๆ อันหนึ่ง ก็คือ ควรวางรูทีนการโปรแกรมเกี่ยวกับ MMU ทั้งหมด ไว้ในส่วน Common Area ที่สามารถใช้งานได้ตลอด

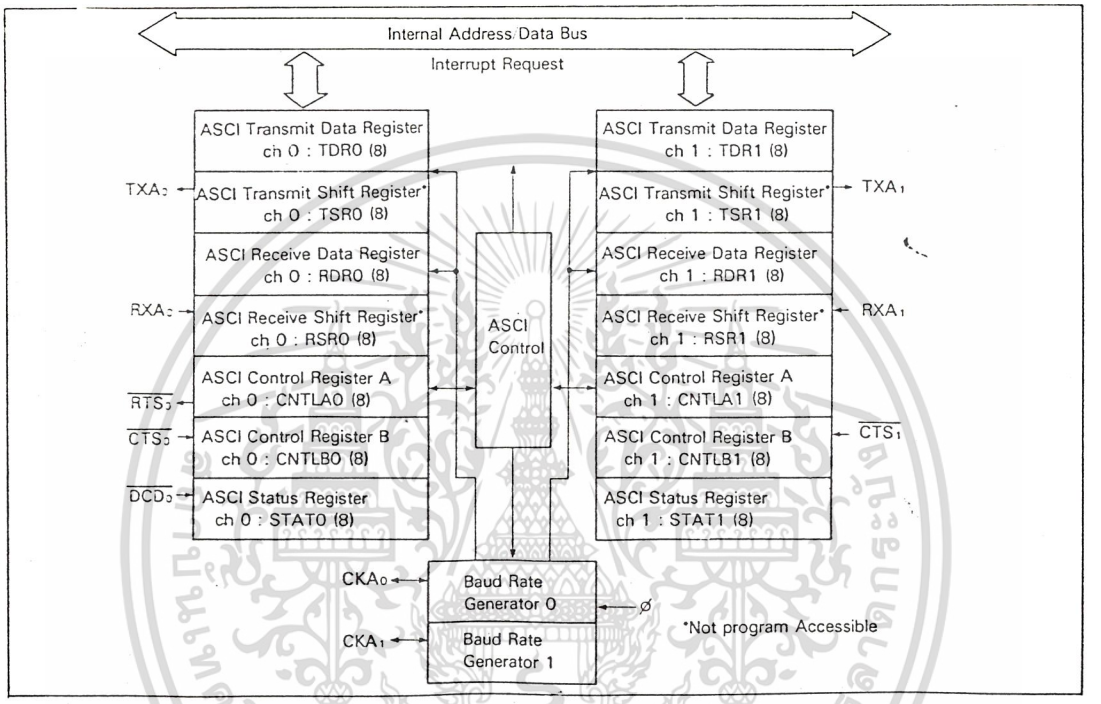
2.2.2.4 ส่วนอินเทอร์เฟสการสื่อสารข้อมูลอนุกรมแบบอะซิงโครนัส (ASYNCHRONOUS SERIAL COMMUNICATION INTERFACE หรือ ASCI)

ใน HD64180 มีส่วน ASCI ซึ่งทำหน้าที่ติดต่อสื่อสารข้อมูลอนุกรม 2 แชนแนล ณ. ซึ่งทั้ง 2 แชนแนลสามารถที่จะรับส่งข้อมูลได้ในแต่ละแชนแนล และสามารถใช้งานได้หลายแบบตามมาตรฐานทาง UART(Universal Asychronous Receiver Transmetter) รวมทั้งมาตรฐาน SCI (Serial communication Interface)

ฟังก์ชั่นต่างๆของ ASCI ในแต่ละแชนแนลสามารถโปรแกรมแยกกันอย่างอิสระ ซึ่งมีดังต่อไปนี้

- การติดต่อสื่อสารแบบทั้งส่งและรับ (Full Duplex)
- ความยาวข้อมูลเป็นแบบ 7 บิท หรือ 8 บิท
- บิทที่ 9 เป็นบิทควบคุมในการใช้งานสื่อสารระหว่างหน่วยประมวลผลหลายหน่วย (Multiprocessor)
- 1 หรือ 2 บิทหยุด (Stop bits)
- การตรวจสอบพาริตี้ (Parity) ทั้งแบบคี่ , คู่ หรือ แบบไม่มีการตรวจสอบ
- การป้องกันความผิดพลาดโดยการตรวจสอบพาริตี้ , โอเวอร์รัน (Overrun) และเฟรมข้อมูล (Frame)
- สามารถโปรแกรมอัตราโบด (Buad Rate) ทั้งโหมด หาร 16 หรือ หาร 64 จากสัญญาณนาฬิกาของ CPU เช่น ถ้า f_c = 6.144 MHz ความเร็วในการส่งจะเท่ากับ 38.4 kbit ต่อ วินาที
- สัญญาณควบคุมโมเด็ม - แชนแนล 0 มี DCD_0 CTS_0 และ RTS_0
  - แชนแนล 1 มี CTS_1
- สามารถโปรแกรมให้ยอม (Enable) หรือ ห้าม (Disable) การใช้งานได้
- สามารถทำงานร่วมกับ DMAC

2.2.2.4.1 บล๊อกไดอะแกรมของ ASCI



รูปที่ 2.2.2.4.1 บล๊อกไดอะแกรมของ ASCI

2.2.2.4.2 รายละเอียดรีจิสเตอร์ต่างๆของ ASCI

ASCI Transmit Shift Register 0 , 1 (TSR 0 , 1)

รีจิสเตอร์นี้จะรับข้อมูลจาก TDR (Transmit Data Register) และข้อมูลจะถูกเลื่อนไปเพื่อส่งออกที่ขา TXA เมื่อการส่งข้อมูลเสร็จเรียบร้อย ไบท์ต่อไปก็จะถูกใส่เข้าไปใน TRS แล้วทำการส่งข้อมูลออกไป ถ้าไม่มีข้อมูลต่อไปจะส่ง TSR จะส่งระดับสัญญาณเป็น HIGH ตลอด รีจิสเตอร์นี้ไม่สามารถโปรแกรมได้

ASCI Transmit Data Register 0 , 1 (TDR 0 , 1 : ตำแหน่ง I/O = 06H , 07H)

ข้อมูลใน TDR จะถูกส่งไปที่ TSR ทันทีที่ TSR ว่าง ข้อมูลจะถูกเขียนไปที่ TSR ขณะกำลังส่งข้อมูลออกไป ดังนั้นตัวส่งของ ASCI จะมีบัฟเฟอร์ถึง 2 ตัว ข้อมูลสามารถที่ถูกอ่านหรือเขียนกับ TDR ได้ ถ้าข้อมูลถูกอ่านจาก TDR การทำงานในการส่งข้อมูลจะไม่มีผลกระทบจากการทำงานของการอ่านนี้

ASCI Receive Shift Register 0 , 1 (RSR 0 , 1)

รีจิสเตอร์นี้จะทำการเลื่อนข้อมูลเข้ามาจากขา RXA เมื่อรีจิสเตอร์เต็ม ข้อมูลจะถูกส่งต่อไปยัง RDR (ASCI Receiver Data Register) ถ้า RDR ว่าง แต่ถ้า RDR ไม่ว่างเมื่อมีการรับข้อมูลใหม่เข้ามา จะเกิดความผิดพลาดโอเวอร์รัน (Overrun) รีจิสเตอร์นี้ไม่สามารถโปรแกรมได้

ASCI Receiver Data Register 0 , 1 (RDR 0 ,1 : ตำแหน่ง I/O = 08H , 09H)

ข้อมูลที่รับเข้ามาจากขา RXA จะนำมาเก็บไว้ที่ RSR หลังจากนั้นจะมีการส่งข้อมูลไปที่ RDR โดยอัตโนมัติถ้า RDR ว่าง ดังนั้นจะเห็นว่าการรับของ ASCI จะมีบัฟเฟอร์ถึง 2 ตัว

ASCI RDR เป็นรีจิสเตอร์สำหรับอ่านอย่างเดียว อย่างไรก็ตามถ้าบิท RDRF = 0 ข้อมูลสามารถเขียนไปที่ ASCI RDR ได้

ASCI Status Register 0 , 1 (STAT 0 , 1)

รีจิสเตอร์นี้จะแสดงผลการสื่อสารข้อมูลของ ASCI , ความผิดพลาดและสภาวะของสัญญาณควบคุมต่างๆ และยังสามารถกำหนดการใช้การอินเทอร์รัพสำหรับ ASCI ได้ด้วย

ASCI Status Register 0 (STAT 0 : ตำแหน่ง I/O = 04H)

บิท	7	6	5	4	3	2	1	0
	RDRF	OVRN	PE	FE	RIE	$DED_{ฺ}$	TDRE	TIE
	R	R	R	R	R/W	R	R	R/W

ASCI Status Register 1 (STAT 1 : I/O Address = 05H)

บิท	7	6	5	4	3	2	1	0
	RDRF	OVRN	PE	FE	RIE	CTSIE	TDRE	TIE
	R	R	R	R	R/W	R/W	R	R/W

- RDRF : Receiver Data Register Full (บิท 7)

บิท RDRF จะถูกเซตเป็น 1 เมื่อข้อมูลถูกโหลดใส่ใน RDR ขณะที่เกิดความผิดพลาดเกี่ยวกับเฟรม หรือ พาริตี้ ค่า RDRF ก็จะยังคงค่าเดิม และข้อมูลก็จะถูกโหลดใส่ RDR เหมือนเดิม บิท RDRF จะถูกเคลียร์ค่าเป็น 0 เมื่อมีการอ่านค่าจาก RDR หรือเมื่อ DCD_{0} เป็น HIGH หรือ อยู่ในโหมด IOSTOP และ ระหว่างการ RESET

- OVRN : Overrun error (บิท 6)

บิท OVRN จะถูกเซตเป็น 1 เมื่อ RDR เต็ม และ RDR เต็ม บิท OVRN จะถูกเคลียร์เป็น 0 เมื่อบิท EFR (Error Flag Reset) ของรีจิสเตอร์ CNTLA มีค่าเป็น 0 หรือเมื่อบิท DCD_0 เป็น HIGH หรืออยู่ในโหมด IOSTOP และระหว่างการ RESET

- PE : Parity Error (บิท 5)

บิท PE จะถูกเซตเป็น 1 เมื่อมีการตรวจสอบพบความผิดพลาดของพาริตี้ของข้อมูลที่รับเข้ามา บิท PE จะถูกเคลียร์ค่าเป็น 0 เมื่อบิท EFR (Error Flag Reset) ของรีจิสเตอร์ CNTLA มีค่าเป็น 0 หรือเมื่อบิท DCD_0 เป็น HIGH หรืออยู่ในโหมด IOSTOP และระหว่างการ RESET

- FE : Framing Error (บิท 4)

ถ้าข้อมูลที่รับเข้ามาไม่มีบิทหยุด (Stop bit) (เช่น อาจเป็น 0 หรือ 1) บิท FE จะถูกเซตค่าเป็น 1 บิท FE จะถูกเคลียร์ค่าเป็น 0 เมื่อบิท EFR (Error Flag Reset) ของรีจิสเตอร์ CNTLA มีค่าเป็น 0 หรือเมื่อ DCD_0 เป็น HIGH หรืออยู่ในโหมด IOSTOP และระหว่างการ RESET

- RIE : Receiver Interrupt Enable (บิท 3)

บิท RIE ถ้าโปรแกรมให้มีค่าเป็น 1 จะหมายถึงการยอมให้มีการอินเทอร์รัปเมื่อขอรับข้อมูลเมื่อบิท RIE ถูกเซตค่าเป็น 1 แล้วแฟลกต่างๆคือ RDRF , OVRN , PE และ FE มีค่าเป็น 1 แล้วจะมีการสร้างสัญญาณ เพื่อขออินเทอร์รัปในการรับข้อมูลสำหรับแชนแนล 0 การอินเทอร์รัปจะเกิดขึ้นเมื่อมีการเปลี่ยนแปลงสัญญาณ DCD_0 จาก LOW ไปเป็น HIGH

ขณะทำการ RESET บิท RIE จะถูกเคลียร์เป็น 0

- DCD_0 : Data Carrier Detect (บิท 2 STAT0)

บิท DCD_0 จะถูกเซตเป็น 1 เมื่อสัญญาณขา DCD_0 เป็น HIGH และจะถูกเคลียร์เป็น 0 เมื่อมีการเปลี่ยนแปลงสัญญาณจาก HIGH เป็น LOW หรือ ระหว่างการ RESET เมื่อบิท DCD_0 = 1 ส่วนรับข้อมูลจะถูกรีเซทสภาวะและจะห้ามไม่ให้มีการรับข้อมูล

- CTS1E : Channel 1 CTS Enable (บิท 2 STAT 1)

สัญญาณ CTS1 ใช้ขาสัญญาณร่วมกัน (multiplex) กับสัญญาณ RXS สำหรับ CSI/O บิท CTS1E ถ้ามีการโปรแกรมเป็น 1 หมายถึงการเลือกใช้ขาสัญญาณในฟังก์ชัน CTS1 และถ้ามีการโปรแกรมเป็น 0 หมายถึงการเลือกใช้ในฟังก์ชัน RXS

- TDRE : Transmit Data Register Empty (บิท 1)

ถ้าบิท TDRE = 1 จะแสดงว่ารีจิสเตอร์ TDR ว่าง และข้อมูลไบท์ต่อไปสามารถส่งไปที่ TDR ได้ หลังจากส่งข้อมูลไปที่ TDR แล้ว บิท TDRE จะถูกเคลียร์ค่าเป็น 0 จนกระทั่ง ASCI ส่งข้อมูลจาก TDR ไปที่ TSR แล้ว บิท TDRE จึงจะกลับมาเป็น 1 อีกครั้งหนึ่ง บิท TDRE ถูกเซตเป็น 1 ในกรณีที่ทำงานในโหมด IOSTOP หรือระหว่าง RESET

บิท TDRE จะเป็น 0 เมื่อสัญญาณ CTS เป็น HIGH

- TIE : Transmit Interrupt Enable (บิท 0)

การเซตค่าบิท TIE เป็น 1 เพื่อยอมให้มีการร้องขอการอินเทอร์รัพสำหรับการส่งข้อมูล ถ้าบิท TIE = 1 และ บิท TDRE = 1 การขออินเทอร์รัพสำหรับการส่งข้อมูลจะเกิดขึ้น บิท TIE จะถูกเคลียร์เป็น 0 ระหว่างการ RESET

ASCI Control Register A0 , 1 (CNTL A0 , 1)

ASCI แต่ละแชนแนลจะมีโหมดในการทำงานหลายโหมด โดยใช้รีจิสเตอร์นี้เป็นส่วนควบคุม เช่น การยอมให้มีการรับส่ง , รูปแบบของข้อมูล (Data Format) และ

การติดต่อสื่อสารระหว่างหลายๆโปรเซสเซอร์ (Multiprocessor Communication)

ASCI Control Register A0 (CNTL A0 : ตำแหน่ง I/O = 00H)

บิท	7	6	5	4	3	2	1	0
	MPE	RE	TE	RTS$_0$	MPBR/ EFR	MOD 2	MOD 1	MOD 0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

ASCI Control Register A1 (CNTL A1 : ตำแหน่ง I/O = 01H)

บิท	7	6	5	4	3	2	1	0
	MPE	RE	TE	CKA1D	MPBR/ EFR	MOD 2	MOD 1	MOD 0
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

- MPE : Multiprocessor Mode Enable (บิท 7)

ASCI มีโหมดสำหรับสื่อสารแบบ มัลติโปรเซสเซอร์ (Multiprocessor Communication) ซึ่งมีบิทพิเศษทำหน้าที่เลือกรูปแบบการติดต่อ ถ้าบิท MP ใน CNTLB ถูกเซตเป็น 1 จะสามารถเลือกรูปแบบข้อมูลได้ ถ้าบิท MP ใน CNTBL เป็น 0 จะไม่สามารถเลือกโหมดได้ ส่วนบิท MPE ถ้าถูกเซตเป็น 1 ข้อมูลไบท์ที่รับมา จะต้องมีบิท MPB (Multiprocessor Bit) เป็น 1 เท่านั้น ซึ่งมีผลกับการตรวจสอบข้อผิดพลาด (แฟลก RDRF และ Error) ถ้าบิท MPB เป็น 0 จะไม่สนใจข้อมูลไบท์อื่นๆ ถ้าบิท MPE มีค่าเป็น 0 ข้อมูลทุกไบท์ (ยกเว้นบิท MPB) จะมีผลกับ บิท RDRF และ แฟลก Error ระหว่างการ RESET บิทนี้ จะถูกเคลียร์เป็น 0

- RE : Receiver Enable (บิท 6)

เมื่อบิท RE ถูกเซตค่าเป็น 1 หมายถึงการยอมให้ตัวรับของ ASCI รับข้อมูลได้เมื่อ RE มีค่าเป็น 0 จะเป็นการห้ามการรับข้อมูลและการรับข้อมูลที่กำลังทำอยู่จะถูกอินเทอร์รัฟ อย่างไรก็ตาม บิท RDRF และแฟลกที่แสดงความผิดพลาด และข้อมูลเดิมในรีจิสเตอร์จะคงค่าเดิมไว้

บิท RE จะถูกเคลียร์เป็น 0 เมื่ออยู่ในโหมด IOSTOP และระหว่าง RESET

- TE : Transmit Enabli (บิท 5)

เมื่อบิท TE ถูกเซตค่าเป็น 1 หมายถึงการยอมให้ตัวส่งของ ASCI ส่งข้อมูลได้ เมื่อ TE มีค่าเป็น 0 จะเป็นการห้ามการส่งข้อมูลและการส่งข้อมูลที่กำลังทำงานอยู่จะถูกอินเทอร์รัฟอย่างไรก็ตาม บิท TDRE และข้อมูลเดิมในรีจิสเตอร์จะคงค่าเดิมไว้

บิท TE จะถูกเคลียร์เป็น 0 เมื่ออยู่ในโหมด IOSTOP และระหว่าง RESET

- RTS_0 : Request To Send Channel 0 (บิท 4 ใน CNTL A0)

เมื่อมีการเซตบิท RTS_0 เป็น 0 สัญญาณ RTS_0 จะเป็น LOW และเมื่อบิท RTS_0 เป็น 0 สัญญาณเอาท์พุทขา RTS_0 จะเป็น HIGH และขณะทำการ RESET เอาท์พุท บิท RTS_0 จะถูกเซตค่าเป็น 1

- CKA1D : CKA1 Clock Disable (บิทใน CNTL A1)

เมื่อมีการเซตบิท CKA1D เป็น 1 จะแสดงถึงขาสัญญาณ $CKA1/TEND_0$ ซึ่งทำหน้าที่เป็นสัญญาณ $TEND_0$ เมื่อมีการเคลียร์ บิท CKA1D เป็น 0 ขาสัญญาณนี้จะถูกใช้เป็นสัญญาณ CKA1 ซึ่งเป็นขาสัญญาณนาฬิกาติดต่อ กับภายนอกแชนแนลสัญญาณที่ 1 ขณะทำการ RESET บิท CKA1D จะถูกเคลียร์ค่าเป็น 0

- MPBR/EFR : Multiprocessor Bit Receiver Error Flag Reset (บิท 3)

เมื่อบิท MP ใน CNTLB = 1 เมื่อมีการอ่านบิท MPBR จะเก็บค่าบิทของ MPB สำหรับการทำงานครั้งล่าสุด ถ้าบิทนี้ถูกเขียนด้วย 0 จะทำให้ทำงานในฟังก์ชันEFR เพื่อจะเคลียร์ค่าของแฟลกแสดงความผิดพลาดทั้งหมด(OVRN , FE และ PE)ให้เป็น 0 ระหว่างการ RESET บิท MPBR/EFR จะไม่สามารถกำหนดค่าได้ (อาจเป็น 1 หรือ 0 ก็ได้)

- MOD 2 ,1 ,0 : ASCI Data Format Mode 2, 1 ,0 (บิท 2 - 0)

บิทเหล่านี้สามารถโปรแกรม รูปแบบของข้อมูลได้ดังนี้

MOD 2	=	0	ข้อมูล 7 บิท
	=	1	ข้อมูล 8 บิท
MOD 1	=	0	ไม่มีพาริตี้ (No Parity)
	=	1	ตรวจสอบพาริตี้
MOD 0	=	0	1 บิทหยุด (Stop Bit)
	=	1	2 บิทหยุด (Stop Bits)

รูปแบบของข้อมูลจะมาจากการกำหนดของบิททั้ง 3 ตามตาราง 2.2.2.4.1

ตารางที่ 2.2.2.4.1 แสดงรูปแบบของข้อมูลจากการกำหนดค่าบิท MOD 0 , 1 , 2

MOD2	MOD1	MOD0	Data Format
0	0	0	Start + 7 bit data + 1 stop
0	0	1	Start + 7 bit data + 2 stop
0	1	0	Start + 7 bit data + parity + 1 stop
0	1	1	Start + 7 bit data + parity + 2 stop
1	0	0	Start + 8 bit data + 1 stop
1	0	1	Start + 8 bit data + 2 stop
1	1	0	Start + 8 bit data + parity + 1 stop
1	1	1	Start + 8 bit data + parity + 2 stop

ASCI Control Register B0 , 1 (CNTL B0, 1)

รีจิสเตอร์CNTL B0 , 1 จะใช้สำหรับการเซตรูปแบบการใช้งานมัลติโปรเซสเซอร์ , เซตการตรวจสอบพาริตี้ และ การกำหนดอัตราส่งข้อมูล (Buad Rate)

ASCI Control Register B0 (CNTL B0 : ตำแหน่ง I/O = 02H)

ASCI Control Register B1 (CNTL B1 : ตำแหน่ง I/O = 03H)

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

- MPBT : Multiprocessor Bit Transmit (บิท 7)

เมื่อมีการเลือกการติดต่อ แบบมัลติโปรเซสเซอร์ (Multiprocessor) คือ เซตบิท MP = 1 บิท MPBT ใช้ในการระบุค่าบิทข้อมูลของ MPB ที่จะทำการส่ง ถ้าบิท MPBT = 1 ฉะนั้น MPB = 1 จะถูกส่งออกไป ถ้า MPBT = 0 จะถูกส่งออกไป ขณะทำการ RESET สภาวะของบิท MPBT จะไม่สามารถกำหนดได้ว่าเป็นเช่นไร

- MP : Multiprocessor Mode (บิท 6)

เมื่อบิท MP ถูกเซตเป็น 1 หมายถึง การใช้งานมัลติโปรเซสเซอร์ ซึ่งมีรูปแบบของข้อมูลโดยกำหนดจาก บิท MOD 2 (ระบุจำนวนบิทข้อมูล) และบิท MOD 0 (จำนวนของบิทหยุด) ในรีจิสเตอร์ CNTL A รูปแบบของข้อมูลจะเป็นดังนี้

บิทเริ่ม (Start Bit) + 7 หรือ 8 บิทข้อมูล + บิท MPB + 1 หรือ 2 บิท หยุด

ในการใช้งานสื่อสารมัลติโปรเซสเซอร์ (MP = 1) จะไม่มีการกำหนดการตรวจสอบพาริตี้ ขณะ RESET บิท MP จะถูกเคลียร์ค่าเป็น 0

- CTS/PS : Clear To Send / Prescale (บิท 5)

เมื่อส่งข้อมูลบิทนี้จะแสดงสภาวะของสัญญาณ CTS ที่รับเข้ามาเพื่อควบคุมถ้าขา CTS เป็น HIGH บิท CTS/PS จะเป็น 1 ถ้าขาสัญญาณ CTS เป็น 1 บิท TDRE จะเป็น 0 (TRD เต็มอยู่) หมายถึงห้ามส่งข้อมูลออก สำหรับแชนแนลสัญญาณที่ 1 สัญญาณ CTS ที่รับเข้ามา จะใช้ขาร่วมกันกับขา RXS (Clocked Serial Receive Data) ดังนั้น บิท CTS/PS จะมีผลเฉพาะการอ่านถ้าบิท CTS1E = 1 เท่านั้นและขาสัญญาณ CTS1/RXS จะถูกเลือกใช้เพื่อทำหน้าที่เป็นขาสัญญาณ CTS การ RESET จะไม่มีผลต่อบิท CTS/PS

เมื่อรับข้อมูลบิท CTS/RXS จะระบุอัตราส่วนเริ่มต้นการสร้างอัตราการรับข้อมูล (Buad Rate) ถ้าบิท CTS/PS ถูกเซตเป็น 1 สัญญาณนาฬิกาของระบบ (0) จะถูกนำมาสเกลลง 30 เท่า ถ้าบิท CTS/PS มีค่าเป็น 0 สัญญาณนาฬิกาของระบบจะถูกนำมาสเกลลง 10 เท่า ขณะทำการ RESET บิท CTS/PS จะเป็น 0

- PEO : Parity Even Odd (บิท 4)

บิท PEO จะเป็นการเลือกการตรวจสอบพาริตี้คู่หรือคี่ ถ้าบิท PEO เป็น 0 จะเป็นการเลือกพาริตี้คู่ ถ้าบิท PEO เป็น 1 จะเป็นการเลือกพาริตี้ (Odd Parity) ขณะทำการ RESET บิท PEO จะเป็น 0

- DR : Divide Ratio (บิท 3)

บิท DR ระบุตัวหารเพื่อสร้างอัตราการรับส่งข้อมูล (Buad Rate) ถ้าบิท DR เป็น 1 จะใช้ 64 เป็นตัวหาร ซึ่งจะหารจากสัญญาณนาฬิกาของการสุ่ม (Data Sampling Clock) ทำให้เกิดอัตราการสุ่มขึ้น (Sanpling Rate) ขณะ RESET บิท DR จะเป็น 0

- SS 2, 1, 0 : Source / Speed Select 2, 1, 0 (บิท 2, 1, 0)

บิทเหล่านี้จะระบุตัวหารตัวสุดท้ายเพื่อสร้างอัตราการรับส่งข้อมูล เพื่อทำการรับส่งข้อมูลจริงๆ รายละเอียดตามตารางที่ 2.2.2.4.2

ตารางที่ 2.2.2.4.2 แสดงถึงอัตราส่วนการหาร

SS 2	SS 1	SS 0	อัตราส่วนการหาร
0	0	0	/1
0	0	1	/2
0	1	0	/4
0	1	1	/8
1	0	0	/16
1	0	1	/32
1	1	0	/64
1	1	1	สัญญาณนาฬิกาภายนอก (External Clock)

ขาสัญญาณ CKA_0 และ CKA_1 ของ ASCI เป็นขาที่ใช้ร่วมกัน (Multiplex) กับขาสัญญาณ $DREQ_0$ และ $TEND_0$ ของ DMAC ระหว่างการ RESET ขาสัญญาณเหล่านี้ จะทำงานในฟังก์ชันเป็นสัญญาณนาฬิกาของข้อมูล (รับเข้ามาใน HD 64180) แต่ถ้าบิท S0 , S1 , S2 ได้รับการโปรแกรมให้มีค่าเป็น 1 ขาสัญญาณเหล่านี้ จะทำงานฟังก์ชันเป็นสัญญาณนาฬิกาของข้อมูล (ส่งออกจาก HD64180) อย่างไรก็ตาม ถ้า DMAC แชนแนลที่ 0 ได้รับการระบุให้ทำงานเพื่อการถ่ายโอนข้อมูลแล้ว ก็จะไม่ต้องคำนึงถึงการเซตค่าของบิท SS2 , SS1 , SS0 และถ้าบิท CKA1D ในรีจิสเตอร์ CMTLA มีค่าเป็น 1 แล้ว ขาสัญญาณ CKA_1/$TEND_0$ จะทำหน้าที่เป็นขาสัญญาณ $TEND_0$ เพื่อทำงานในฟังก์ชันของ DMA โดยไม่ต้องคำนึงถึงการเซตค่าของบิท SS2 , SS1 , SS0 เลย

อัตราการรับส่งข้อมูล (Bual Rate) และการกำหนดค่าของบิทต่างๆได้แสดงดังตารางที่ 2.2.2.4.3

ตารางที่ 2.2.2.4.3 รายละเอียดอัตราการรับส่งข้อมูล

Prescaler		Sampling Rate		Baud Rate				General Divide Ratio	Baud Rate (Example) (BPS)			CKA	
PS	Divide Ratio	DR	Rate	SS2	SS1	SS0	Divide Ratio		ϕ=6.144 MHz	ϕ=4.608 MHz	ϕ=3.072 MHz	I/O	Clock Frequency
0	$\phi \div 10$	0	16	0	0	0	$\div 1$	$\phi \div 160$	38400		19200	O	$\phi \div 10$
				0	0	1	2	320	19200		9600		20
				0	1	0	4	640	9600		4800		40
				0	1	1	8	1280	4800		2400		80
				1	0	0	16	2560	2400		1200		160
				1	0	1	32	5120	1200		600		320
				1	1	0	64	10240	600		300		640
				1	1	1	—	fc÷16	—	—	—	I	fc
		1	64	0	0	0	$\div 1$	$\phi \div 640$	9600		4800	O	$\phi \div 10$
				0	0	1	2	1280	4800		2400		20
				0	1	0	4	2560	2400		1200		40
				0	1	1	8	5120	1200		600		80
				1	0	0	16	10240	600		300		160
				1	0	1	32	20480	300		150		320
				1	1	0	64	40960	150		75		640
				1	1	1	—	fc÷64	—	—	—	I	fc
1	$\phi \div 30$	0	16	0	0	0	$\div 1$	$\phi \div 480$		9600		O	$\phi \div 30$
				0	0	1	2	960		4800			60
				0	1	0	4	1920		2400			120
				0	1	1	8	3840		1200			240
				1	0	0	16	7680		600			480
				1	0	1	32	15360		300			960
				1	1	0	64	30720		150			1920
				1	1	1	—	fc÷16	—	—	—	I	fc
		1	64	0	0	0	$\div 1$	$\phi \div 1920$		2400		O	$\phi \div 30$
				0	0	1	2	3840		1200			60
				0	1	0	4	7680		600			120
				0	1	1	8	15360		300			240
				1	0	0	16	30720		150			480
				1	0	1	32	61440		75			960
				1	1	0	64	122880		37.5			1920
				1	1	1	—	fc÷64	—	—	—	I	fc

2.2.2.4.3 สัญญาณควบคุม MODEM ต่างๆ

ASCI แชนแนล 0 จะมีขาสัญญาณภายนอกคือ CTS_0 , DCD_0 และ RTS_0 เพื่อควบคุมโมเด็ม (MODEM) ส่วน ASCI แชนแนล 1 จะมีขา CTS_1 แบ่งกันใช้งาน (Multiplex) กับ ขา RXS (Clocked Serial Receive Data)

- CTS_0 : Clear To Send 0 (ขาอินพุท)

ขา CTS_0 จะรับสัญญาณภายนอกในการควบคุมการสั่งข้อมูลของ ASCI แชนแนลที่ 0 เมื่อสัญญาณ CTS_0 เป็น HIGH บิท TDRE ของแชนแนลที่ 0 จะเป็น 0 ด้วย โดยไม่สนใจว่า TDRO (Transmit Data Register) จะเต็มหรือว่าว่างอยู่ เมื่อ CTS_0 เป็น LOW บิท TDRE จะตอบสนองสภาวะของ TDR_0 สังเกตได้ว่าในการที่จะห้ามการส่งข้อมูลออกไม่สามารถทำโดยการให้สัญญาณ CTS_0 เป็น HIGH แต่จะทำได้โดยการห้ามที่บิท TDRE เท่านั้น

- DCD_0 : Data Carrier Detect 0 (ขาอินพุท)

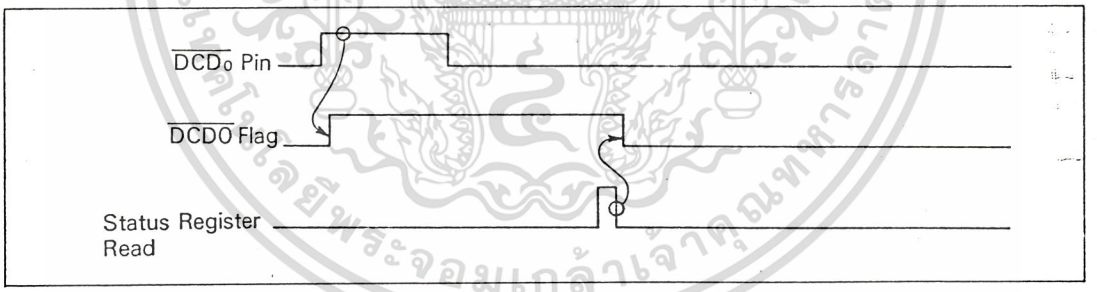
ขา DCD_0 จะรับสัญญาณภายนอกเพื่อควบคุมการรับข้อมูลของ ASCI แชนแนล 0 เมื่อสัญญาณ DCD_0 เป็น HIGH บิท RDRF ของแชนแนลที่ 0 จะเป็น 0 โดยไม่สนใจว่า RDR_0 (Receive Data Register)จะเต็มหรือว่างอยู่และแฟลกความผิดพลาด(Error Flag) จะเป็น 0 (เช่น PE , OVRN และ FE) หลังจากสัญญาณ DCD_0 เป็น LOW แฟลกต่างๆ จะยังไม่กลับไปทำงานตามปกติ จนกระทั่งรีจิสเตอร์ STATO ถูกอ่าน สังเกตว่าเมื่อรีจิสเตอร์ STATO ถูกอ่านครั้งแรกจะเป็นการทดให้แฟลกต่างๆกลับสู่การทำงานปกติ ซึ่งจะแสดงบิท DCD_0 = 1 แม้ว่าขาอินพุท DCD_0 จะเป็น HIGH ดังนั้นรีจิสเตอร์ STATO ควรจะถูกอ่าน 2 ครั้ง เพื่อแน่ใจว่าบิท DCD_0 เป็น 0 แล้ว

- RTS_0 : Request To Send 0 (ขาเอาท์พุท)

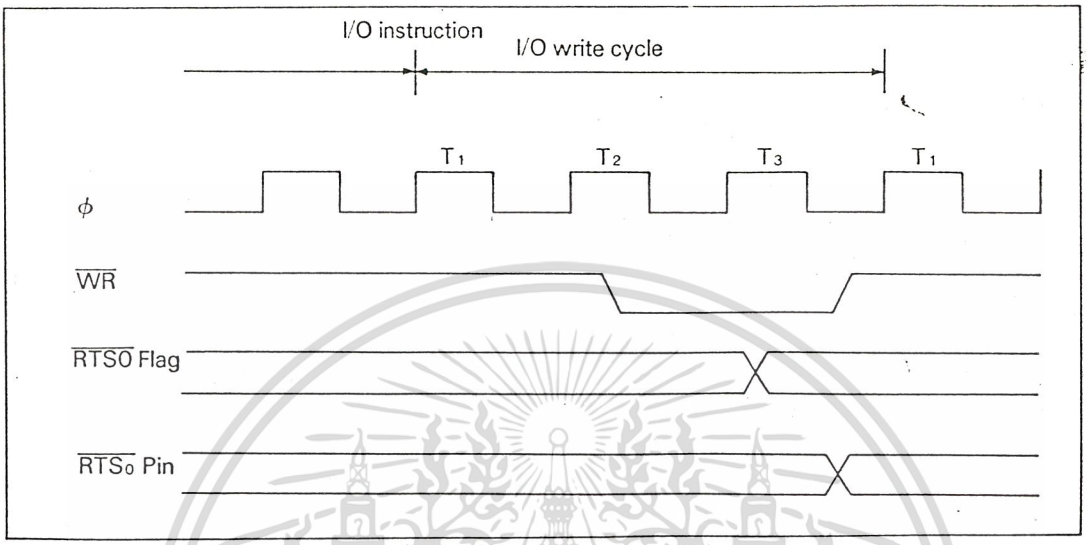
สัญญาณ RTS_0 จะเป็นสัญญาณควบคุมการติดต่อสื่อสาร(เริ่มต้นหรือหยุด)กับอุปกรณ์อื่นๆ สำหรับการส่งข้อมูล โดยจะต่อกับขาสัญญาณ CTS ของอุปกรณ์ที่จะติดต่อสัญญาณ RTS_0 จะเป็นพอร์ทเอาท์พุท 1 บิท ซึ่งไม่มีผลกับแฟลกหรือรีจิสเตอร์ของอุปกรณ์ที่จะติดต่อ

- CTS_1 : Clear To Send (ขาอินพุท)

ขาสัญญาณ CTS_1 เป็นขาสัญญาณที่ใช้ร่วมกัน (Multiplexed) กับขา RXS โดยสามารถเลือกการใช้งานจากบิท CTS1E ในรีจิสเตอร์ STAT1 โดยเซตให้เป็น 1 เพื่อให้ขานี้ทำงานในฟังก์ชัน CTS_1 ซึ่งการทำงานจะเหมือนกับสัญญาณ CTS_0 รูปที่ 2.2.2.4.2 (a) และ (b) แสดงสัญญาณสำหรับควบคุม MODEM

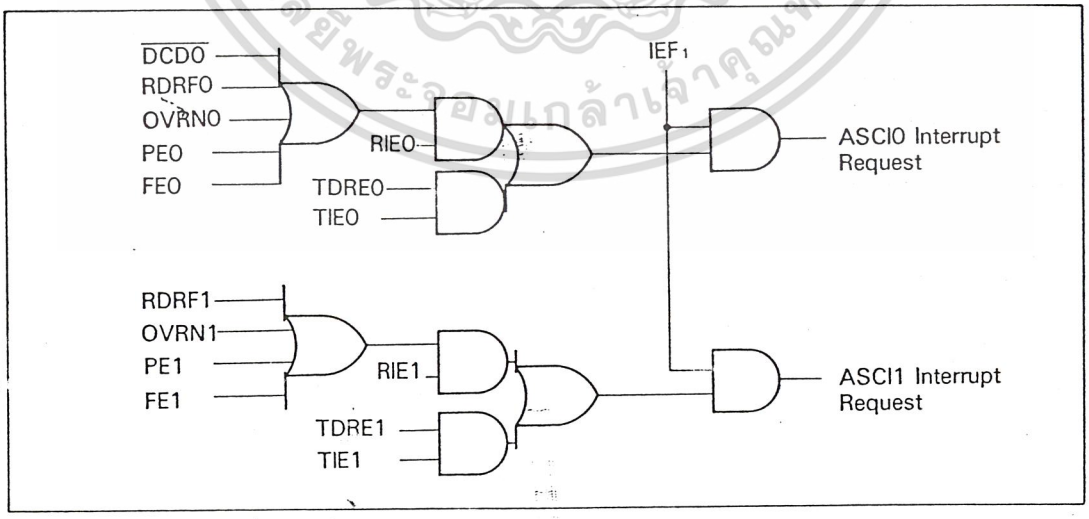


รูปที่ 2.2.2.4.2 (a) สัญญาณเวลาสัญญาณ DCD_0



รูปที่ 2.2.2.4.2 (b) สัญญาณเวลาสัญญาณ RTS$_0$

2.2.2.4.4 การอินเทอร์รัฟ ASCI



รูปที่ 2.2.2.4.3 วงจรสร้างสัญญาณร้องขอ (Reqrest) การอินเทอร์รัฟ ASCI

2.2.2.4.5 การทำงานติดต่อระหว่าง ASCI กับ DMAC

การทำงานของ ASCI กับ DMAC แชนแนลที่ 0 จะใช้รูปแบบการติดต่อของ DMAC กับแฟลกของ ASCI เพื่อเป็นสัญญาณร้องขอการทำ DMA

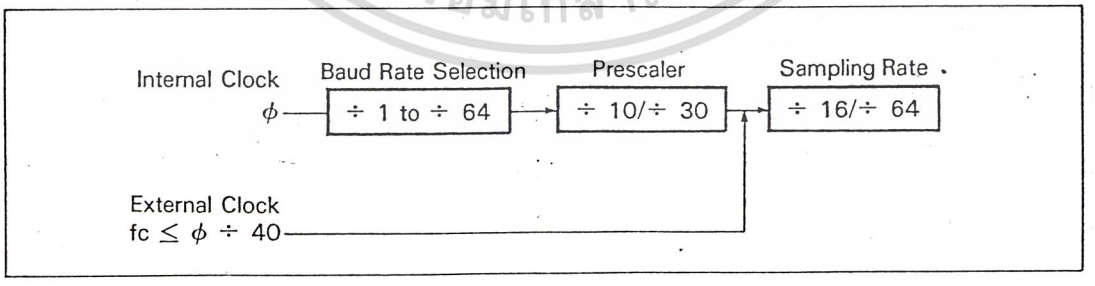
2.2.2.4.6 ASCI กับ การ RESET

ขณะทำการ RESET รีจิสเตอร์แสดงสภาวะและควบคุมของ ASCI จะถูกให้ค่าเริ่มต้นใหม่ตามที่ได้อธิบายไปในหัวข้อต่าง ๆ

การทำงานรับและส่งจะหยุดการทำงานระหว่าง RESET แต่อย่างไรก็ตามข้อมูลใน TDR (Transmit Data Register) และ RDR (Receiver Data Register) จะไม่เปลี่ยนแปลง

2.2.2.4.7 สัญญาณนาฬิกาของ ASCI (ASCI CLOCK)

ถ้าเป็นสัญญาณนาฬิกาจากภายนอก สัญญาณนาฬิกานี้จะถูกหารด้วยอัตราสุ่ม (Sampling Rate) คือ หาร 16/หาร 64 ดังรูปที่ 2.2.2.4.4



รูปที่ 2.2.2.4.4 บล๊อกไดอะแกรมของสัญญาณนาฬิกาของ ASCI

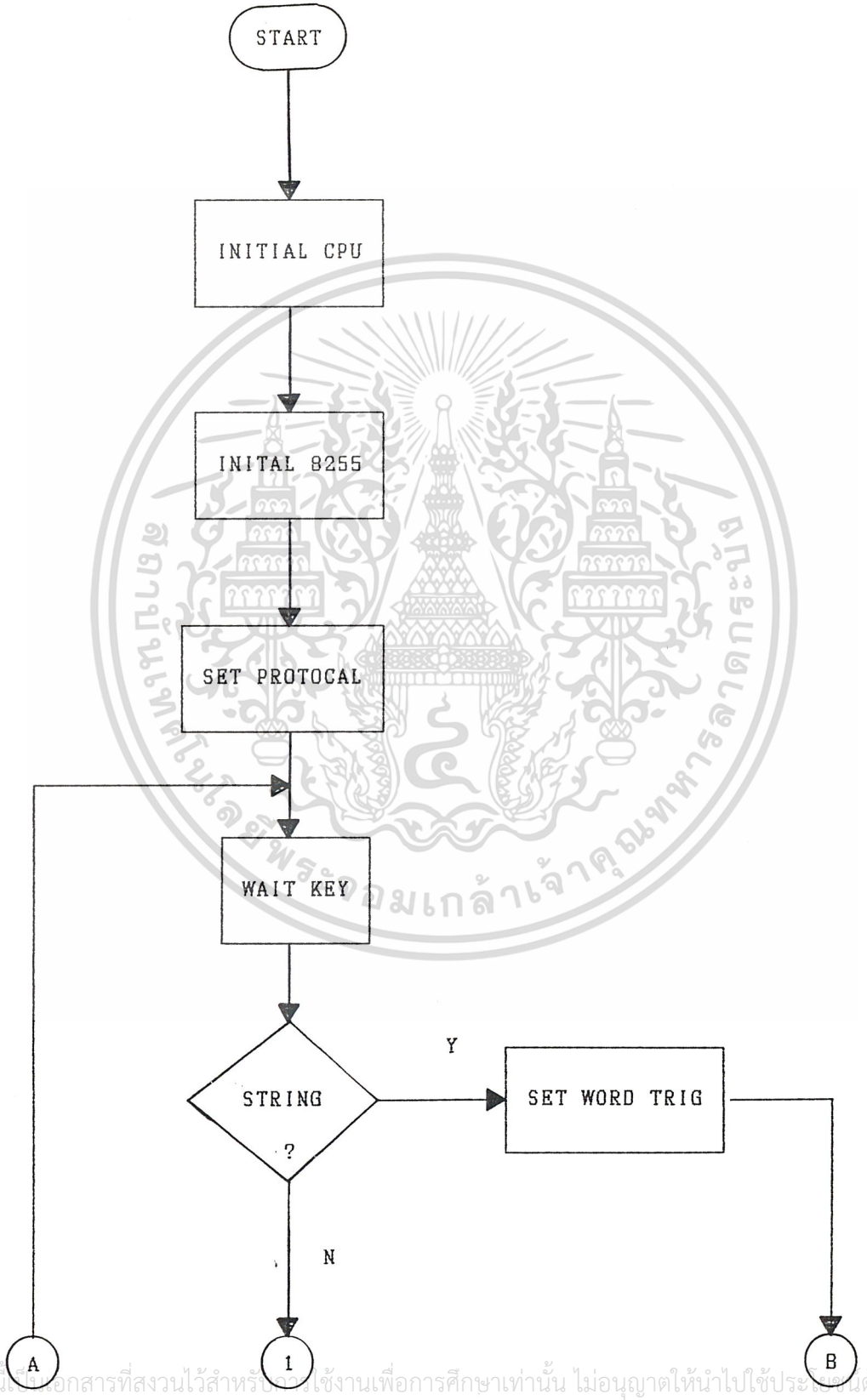
2.3 โฟร์ชาร์ทแสดงการทำงานของโปรแกรมควบคุมเครื่องวิเคราะห์สัญญาณทางตรรก

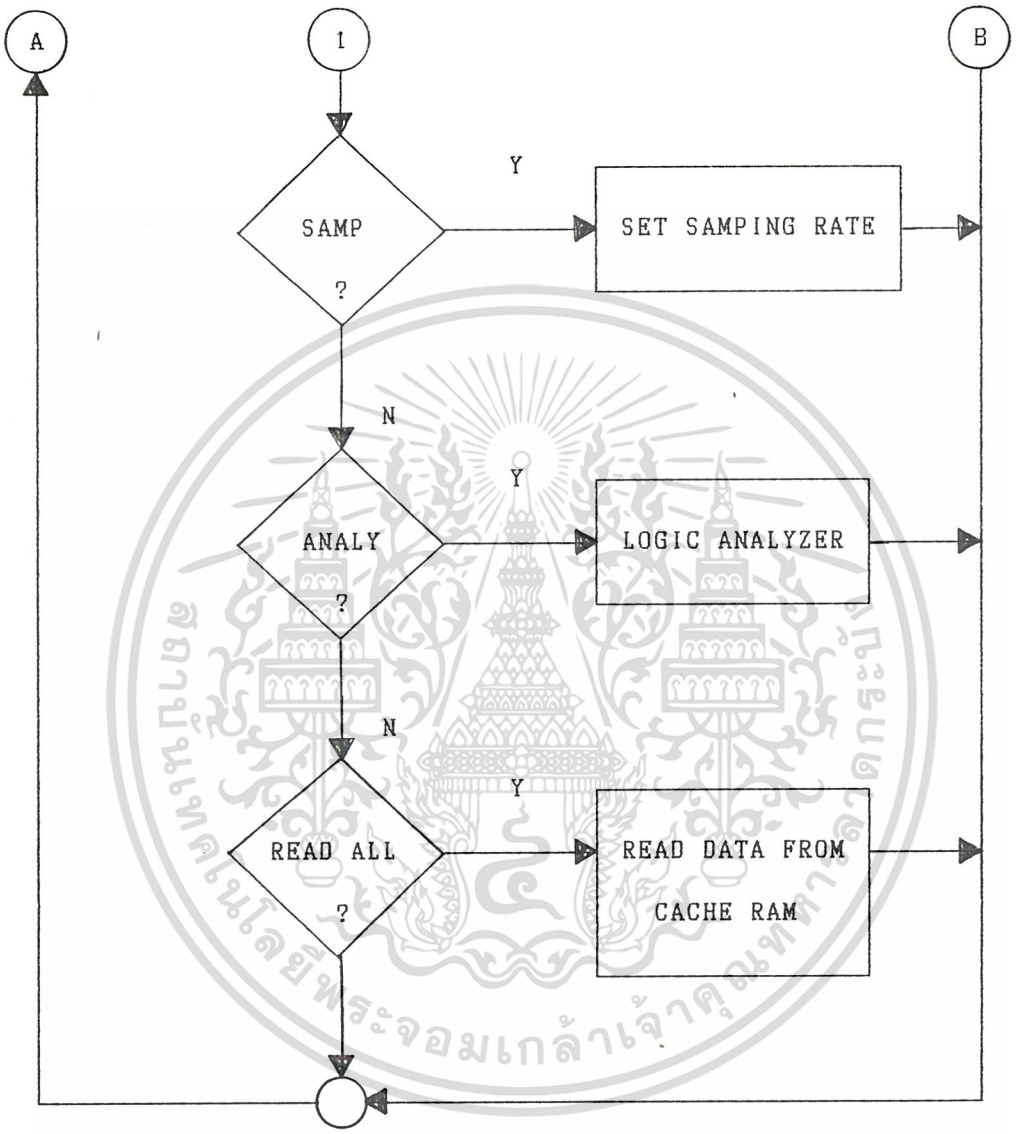
จะขอกล่าวถึง โฟร์ชาร์ทของโปรแกรมควบคุมของเครื่องวิเคราะห์สัญญาณทางตรรก ซึ่งมีหลักการของโปรแกรมตามโฟร์ชาร์ท ซึ่งแสดงในรูปภาพที่ 2.3.1 และมีรายละเอียดดังต่อไปนี้

1. เริ่มต้นด้วยการเริ่มต้น (START UP)
2. เป็นการกำหนดค่าเริ่มต้น (INITIALIZE) ของตัว CPU ที่เราใช้งาน (HD64180)
3. เป็นการกำหนดค่าเริ่มต้นให้กับ IC 8255 (PROGROMMABLE PERIPHERAL INTERFACE)
4. เป็นการกำหนดค่าเริ่มต้น (SET PROTOCAL) โดยเป็นการกำหนดคุณสมบัติของการติดต่อสื่อสารแบบอนุกรม เช่น กำหนดจำนวนบิทที่ใช้ (8 บิท), บิทเริ่มต้น (START BIT), บิทหยุด (STOP BIT)และคุณสมบัติอื่นๆที่จำเป็นสำหรับการสื่อสารแบบอนุกรม
5. เป็นขั้นตอนรอรับคำสั่งจาก KEY BOARD ว่าเราจะให้ทำงาน หรือ เลือกการทำงานของอะไรก่อน
6. รับคำสั่ง "T" (TRIGGER WORD) เมื่อได้รับสัญญาณจาก KEY BOARD เป็นอักษร "T" ก็จะเข้าไปสู่ซับรูทีน (SUBROUTINE) ในการเซตค่าของ WORD TRIG ซึ่งเป็นข้อมูลเริ่มต้นที่จะกระตุ้นและกำหนดให้เครื่องวิเคราะห์สัญญาณทางตรรกเริ่มตรวจรับและเก็บข้อมูล เพื่อทำงานต่อไป
7. รับคำสั่ง "S" (SAMPLING RATE) จะเป็นซับรูทีนในการเซตค่าของความถี่ที่ใช้ในการตรวจรับข้อมูล
8. รับคำสั่ง "Z" (LOGIC ANALYZER) จะเป็นซับรูทีนใหญ่ในการทำให้เครื่องวิเคราะห์สัญญาณทางตรรกรับข้อมูลจากการตรวจจับ แสดงผลของสภาวะลอจิก (PLUSE TRAIN)บนจอ
9. รับคำสั่ง "R" (READ DATE FORM CACHE RAM) ซึ่งเป็นคำสั่งให้อ่านข้อมูลจาก CACHE RAM ไปยัง SRAM เพื่อแสดงผล

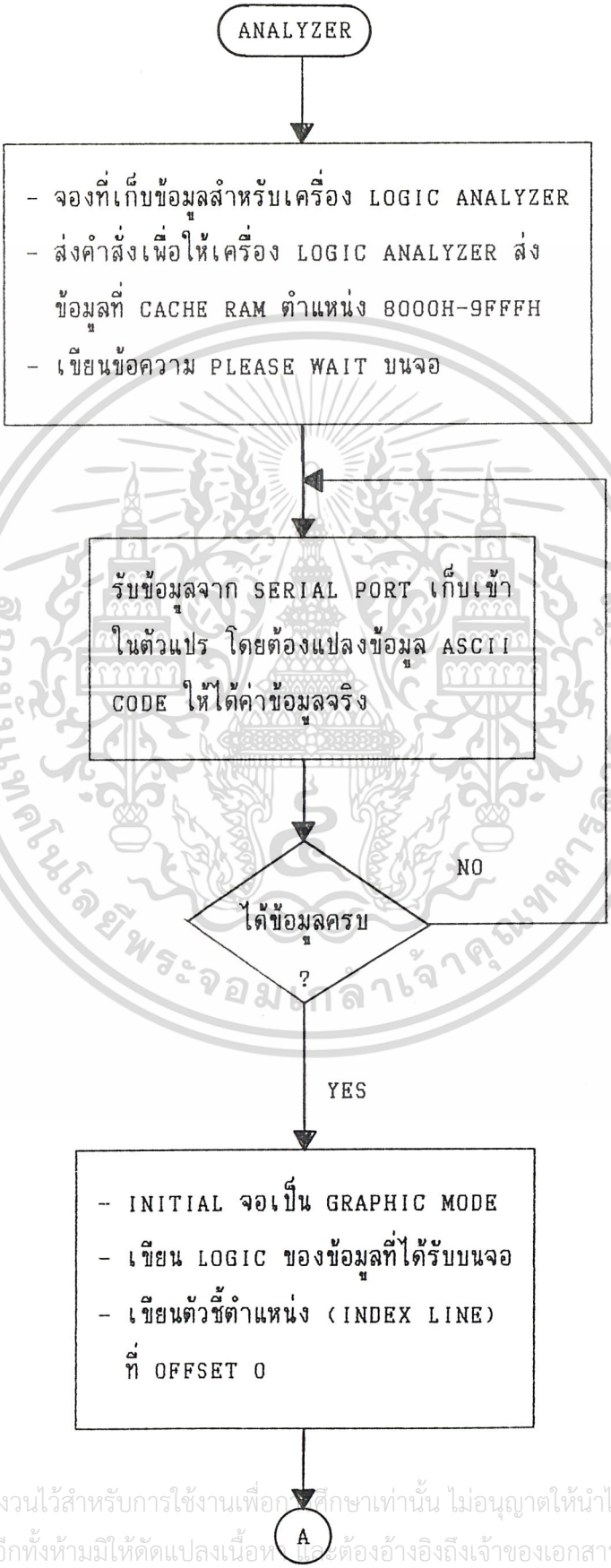
ภายในโปรแกรมการทำงานของเครื่องวิเคราะห์สัญญาณทางตรรก จะมีซับรูทีนที่สำคัญมากอยู่หนึ่งตัว คือ ซับรูทีนของ Z (LOGIC ANALYZER) ซึ่งเป็นโปรแกรมการเอาข้อมูลที่ตรวจรับได้ออกมาแสดงสภาวะทางตรรกทางจอภาพนั่นเอง ซึ่งการทำงานของซับรูทีนนี้ แสดงในรูปภาพที่ 2.3.2 ดังมีการทำงานดังต่อไปนี้

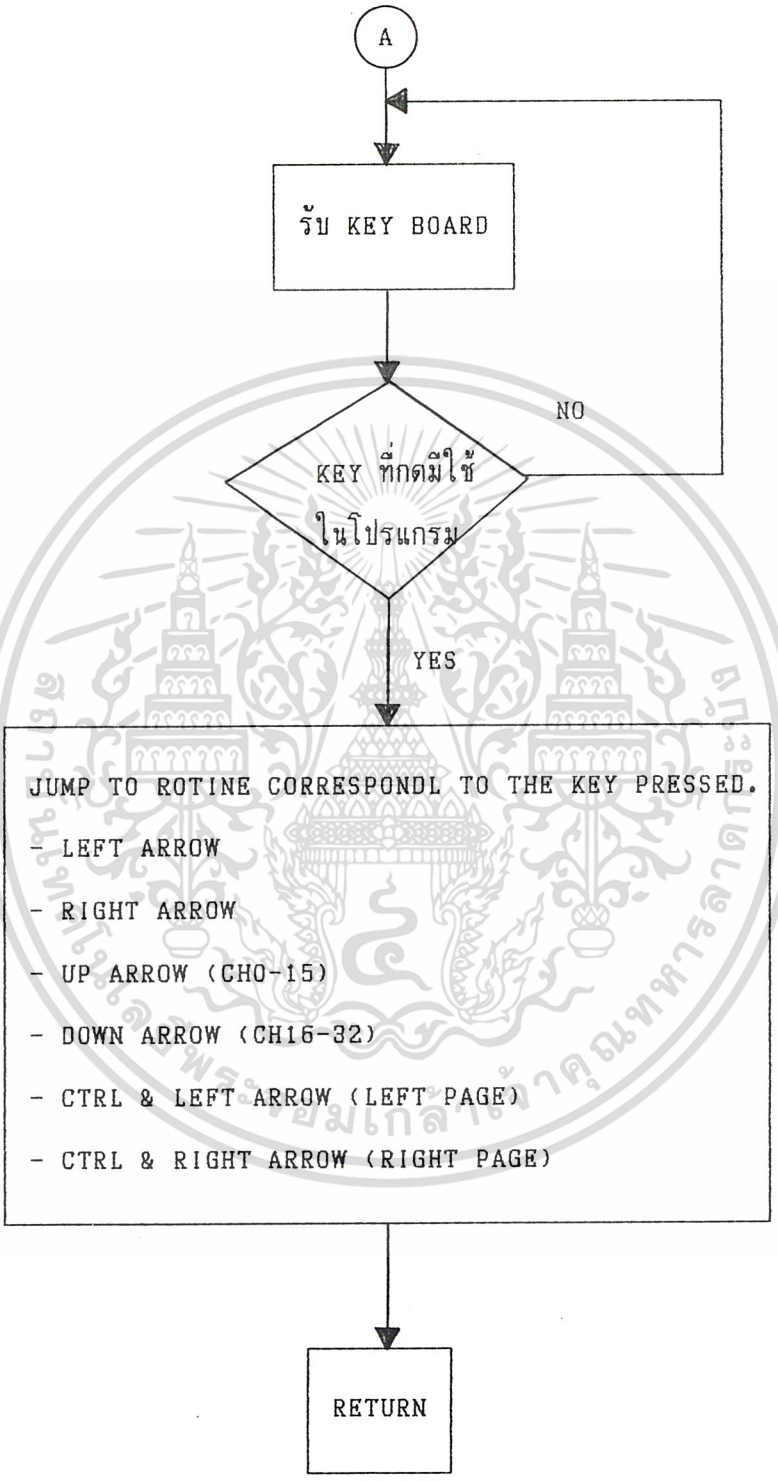
1. เริ่มต้นด้วยการสำรองพื้นที่ภายในหน่วยความจำ ที่จะเก็บข้อมูลสำหรับเครื่องวิเคราะห์สัญญาณทางตรรก และส่งคำสั่งเพื่อให้เครื่องวิเคราะห์สัญญาณทางตรรกส่งข้อมูลที่CACHE RAM ตำแหน่ง 8000H-9FFFH มาให้เพื่อนำไปแสดงผลระหว่างการทำงานช่วงนี้ จะแสดงข้อความว่า "PLEASE WAIT"เพื่อบอกกับผู้ใช้ว่ากำลังทำการถ่ายเทข้อมูลอยู่
2. รับข้อมูลจาก SERIAL PORT (COM. 1)เก็บเข้าในตัวแปรโดยจะแปลงข้อมูลที่รับได้ในรูปของ ASCII CODE ให้ได้ข้อมูลจริงที่จะแสดงให้ผู้ใช้ทราบ
3. ตรวจสอบว่าได้รับข้อมูลมาครบถ้วนหรือยัง ถ้ายังให้กลับไปรับให้ครบ ถ้ารับข้อมูลครบแล้ว ให้กำหนดค่าเริ่มต้นให้กับจอภาพเป็น GRAPHIC MODE ต่อจากนั้นเขียนข้อมูลที่ได้รับให้อยู่ในรูปของสภาวะทางตรรก (PLUSE TRAIN)ขึ้นบนจอภาพ และเขียนตัวชี้ตำแหน่ง (INDEX LINE) ที่จุดเริ่มต้น (OFFSET 0)
4. รอรับคำสั่งจาก KEY BOARD จากผู้ใช้เครื่องว่า ต้องการตรวจสอบหรือใช้งานอะไรอีก เมื่อได้รับคำสั่งจาก KEY BOARD ในรูปของ KEY ที่มีใช้ในโปรแกรม ก็จะไปยังซับรูทีนย่อยตาม KEY นั้น ซึ่งมี KEY ที่ใช้งานดังนี้
   - ลูกศรทางซ้ายมือ(LEFT ARROW)เป็นการเลื่อนตัวชี้ตำแหน่งไปทางซ้ายมือ
   - ลูกศรทางขวามือ(RIGHT ARROW)เป็นการเลื่อนตัวชี้ตำแหน่งไปทางขวามือ
   - ลูกศรขึ้น(UP ARROW)เป็นการเปลี่ยนหน้าแสดงผลไปยังCH0~CH15
   - ลูกศรลง(DOWN ARROW)เป็นการเปลี่ยนหน้าการแสดงผลไปยังCH16~CH31
   - CTRL และ LEFT ARROW เป็นการเลื่อนหน้าของการแสดงผลไปทางหน้าการแสดงทางซ้ายหนึ่งหน้า
   - CTRL และ RIGHT ARROW เป็นการเลื่อนหน้าของการแสดงผลไปทางหน้าการแสดงทางขวาหนึ่งหน้า





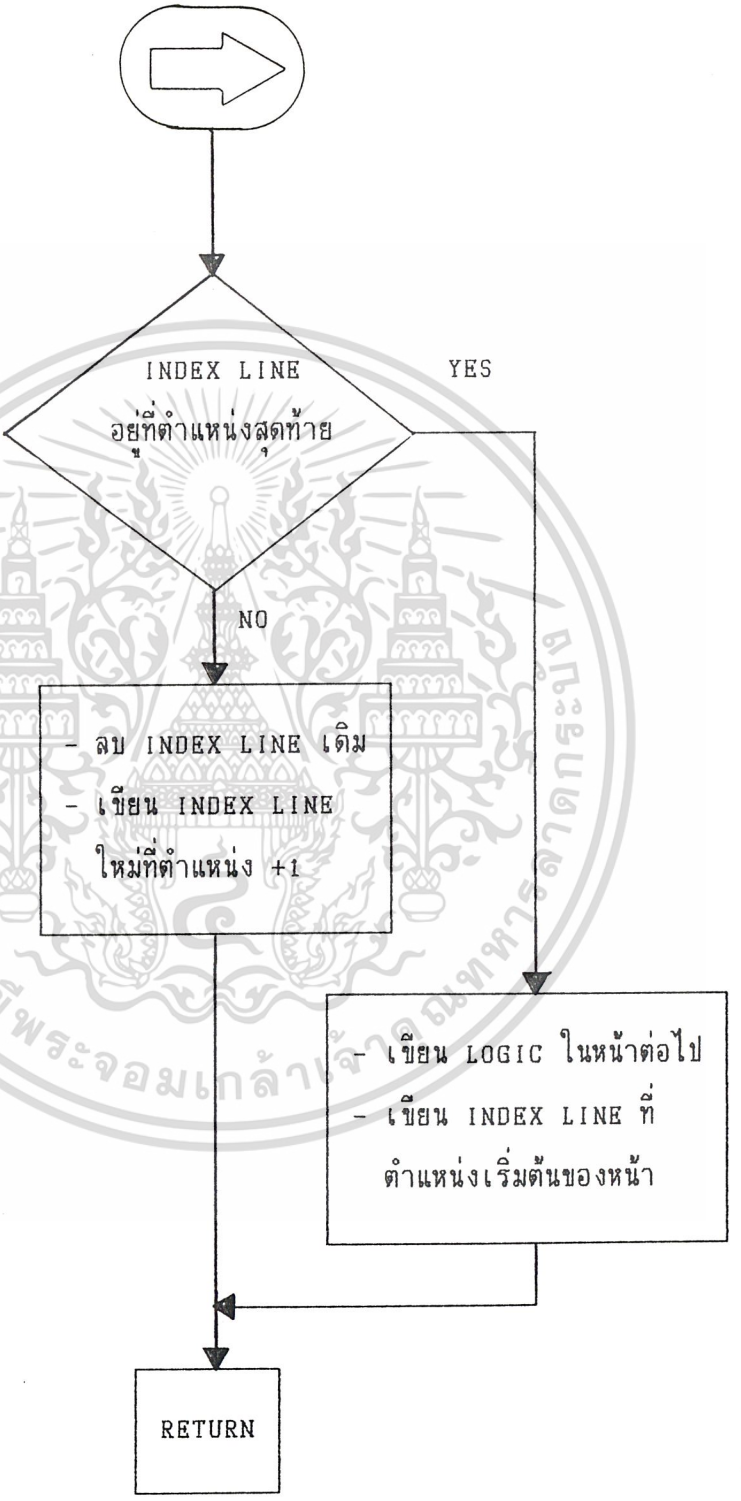
รูปที่ 2.3.1 แสดงถึงโฟร์ชาร์ทของโปรแกรมการทำงาน
ของเครื่องวิเคราะห์สัญญาณทางตรรก



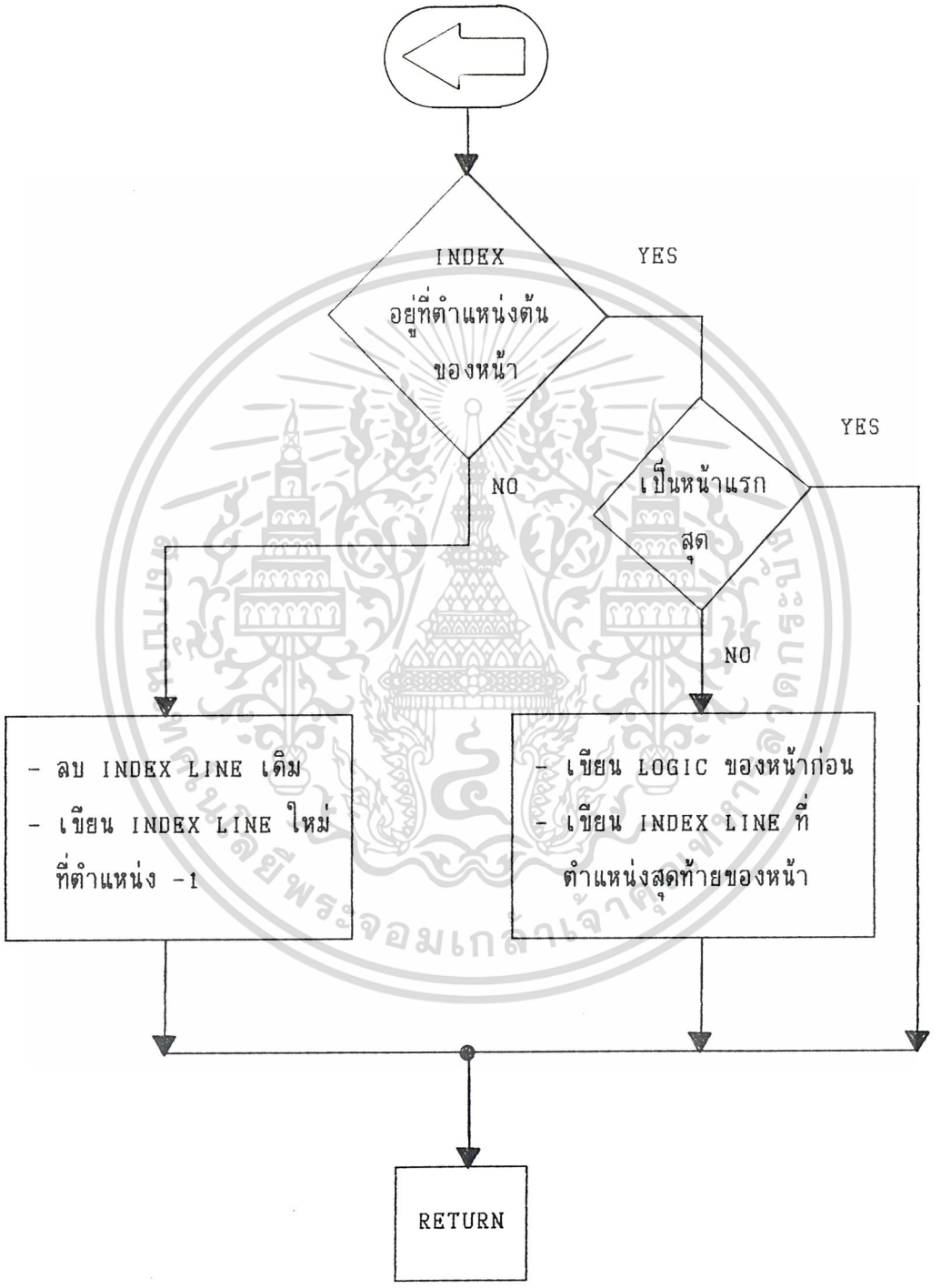


รูปที่ 2.3.2 แสดงถึงโฟร์ชาร์ทของโปรแกรม

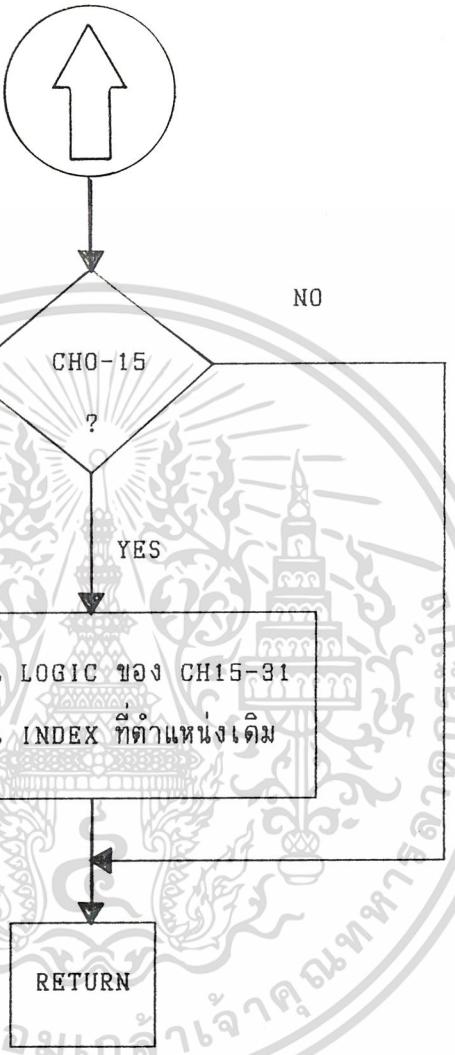
ตัวซับรูทีน (SUBROUTINE) LOGIC ANALYZER



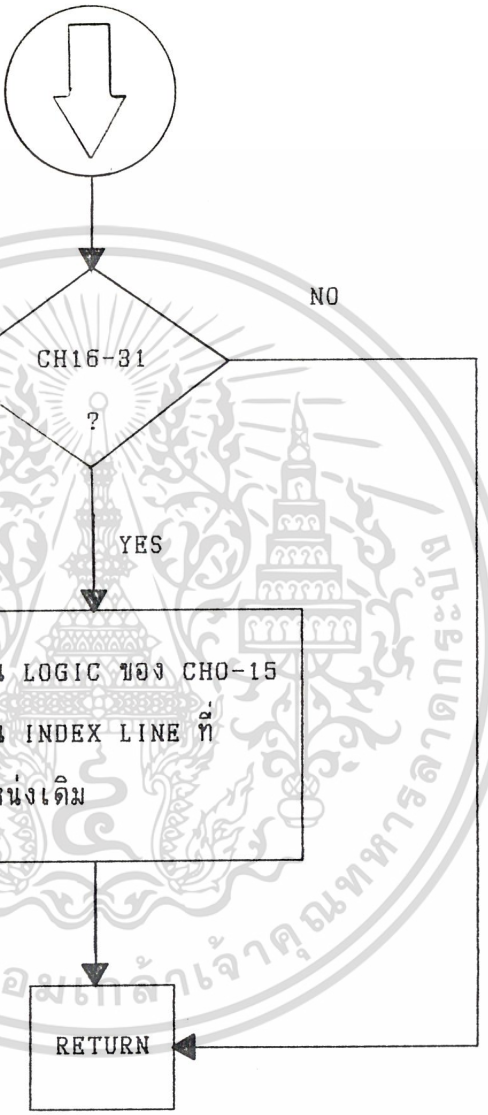
รูปที่ 2.3.3 แสดงถึงโฟร์ชาร์ทของโปรแกรมซับรูทีน LEFT ARROW



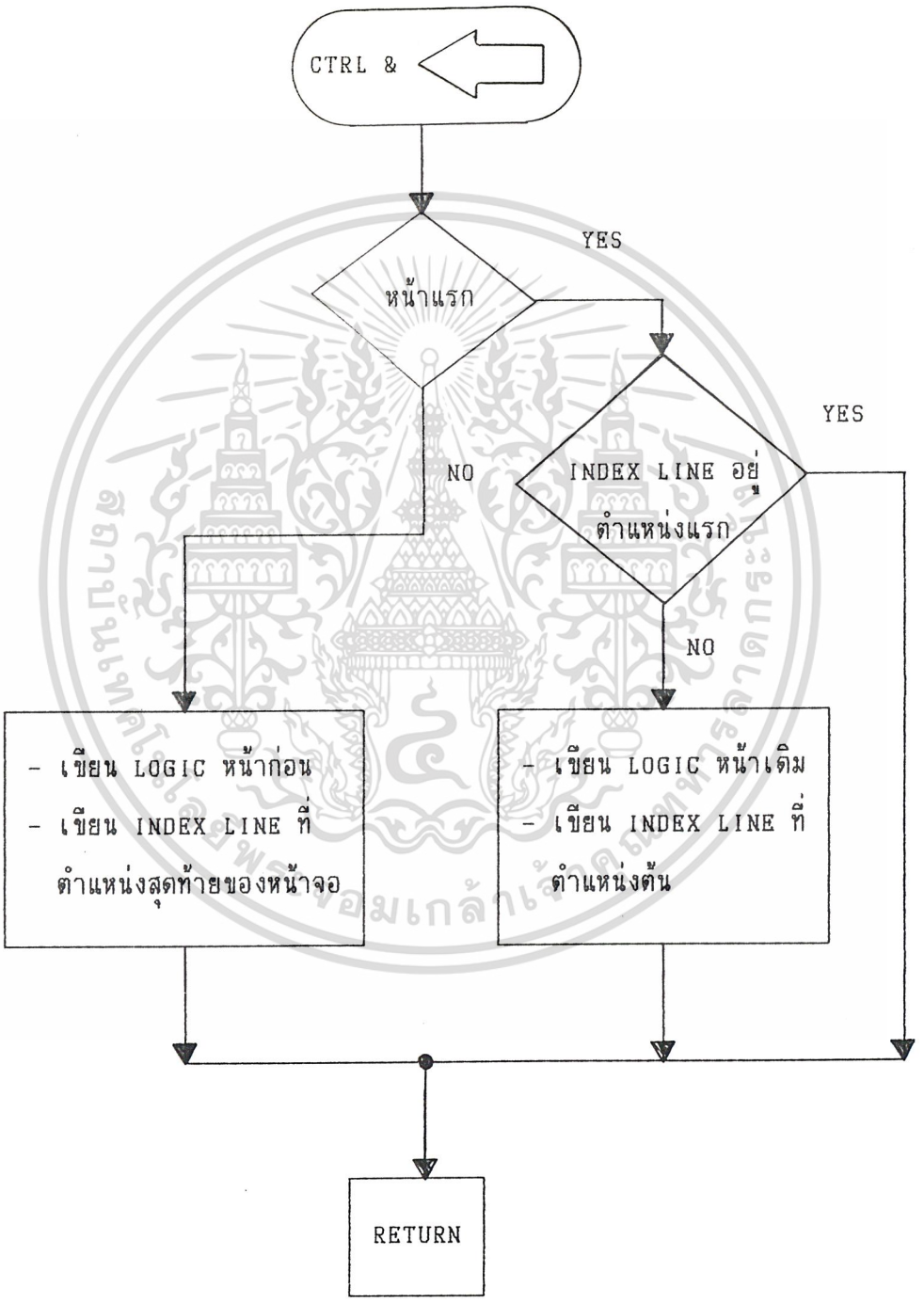
รูปที่ 2.3.4 แสดงถึงโฟร์ชาร์ทของโปรแกรมซับรูทีน RIGHT ARROW



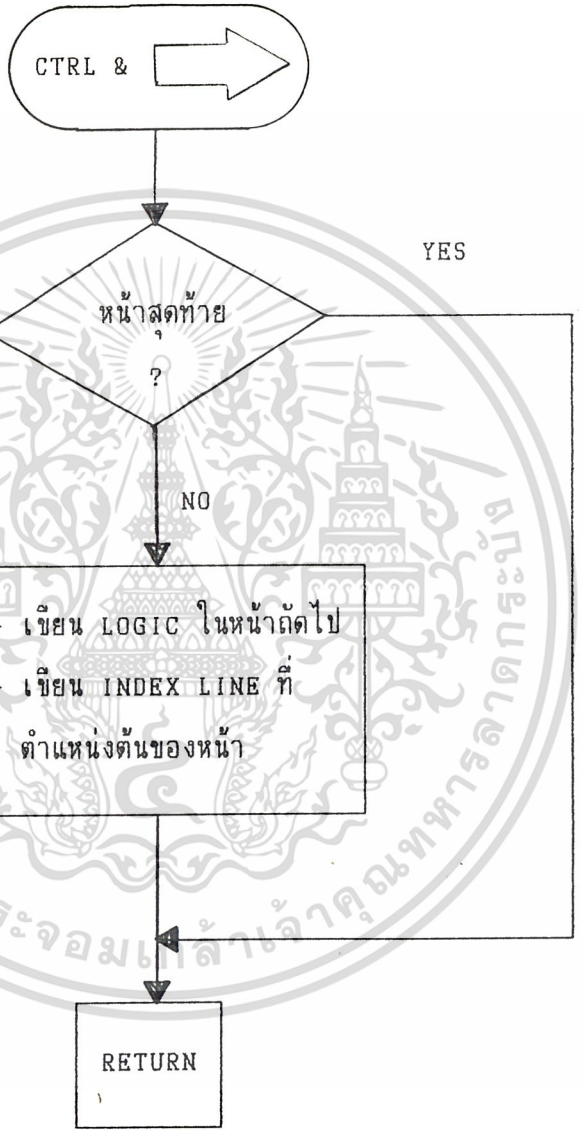
รูปที่ 2.3.5 แสดงถึงโฟร์ชาร์ทของโปรแกรมซับรูทีน UP ARROW



รูปที่ 2.3.6 แสดงถึงโฟร์ชาร์ทของโปรแกรมซับรูทีน DOWN ARROW



รูปที่ 2.3.7 แสดงถึงโฟร์ชาร์ทของโปรแกรมซับรูทีน CTRL & LEFT ARROW



รูปที่ 2.3.8 แสดงถึงโฟร์ชาร์ทของโปรแกรมซับรูทีน CTRL & RIGHT ARROW

บทที่ 3

วงจรที่ใช้ในการทดลองและการทดลองใช้งาน

3.1 รายละเอียดของวงจรที่ใช้งาน

เริ่มจากวงจรในแผ่นที่สอง ขั้วต่อ JP2 จะเป็นขั้วต่อของสายนำสัญญาณทางตรรก 32 เส้น แต่ละเส้น คือ สัญญาณด้านเข้า 1 ช่องสัญญาณ รวมทั้งสิ้น 32 ช่องสัญญาณ สัญญาณ 32 ช่องสัญญาณจะถูกแบ่งออกเป็น 4 กลุ่ม กลุ่มละ 8 ช่องสัญญาณ (CH1~CH8 , CH9~CH16 , CH17~CH24 , CH25~CH32) เพื่อสะดวกต่อการทำความเข้าใจจะขอยกตัวอย่างการอธิบายเพียงกลุ่มเดียวเท่านั้น คือ CH1~CH8 สัญญาณส่วนหนึ่งของกลุ่มนี้จะถูกป้อนมาเข้ายังไอ.ซี.U55 ซึ่งทำหน้าที่ค้างข้อมูล (74ACT374) ตามจังหวะกระตุ้นของสัญญาณนาฬิกา AGCLK ที่ขา 11 สัญญาณที่ได้จากไอ.ซี.U55 จะปรากฏออกมาที่ขา Q หรือไม่ขึ้นอยู่กับการควบคุมสภาวะทางตรรกของสัญญาณ DDIREN ซึ่งต่ออยู่กับขา 2 ของไอ.ซี.U21 (74ACT86) ถ้าสัญญาณ DDIREN เป็นตรรกะ 1 สัญญาณด้านออกของไอ.ซี.U21 จะให้ตรรกะ 0 เป็นการบังคับให้ ไอ.ซี.U55 ส่งสัญญาณที่ค้างไว้ออกมาภายนอกสัญญาณด้านออกของไอ.ซี.U55 ส่วนหนึ่งจะถูกป้อนไปยังขาข้อมูลของแคชแรม (CACHE RAM)อีกส่วนหนึ่งจะป้อนเป็นสัญญาณด้านเข้าหนึ่งของแนนเกท (74ACT00 , U39 , U26) สัญญาณด้านเข้าอีกขาหนึ่งของแนนเกท ได้มาจากไอ.ซี.U54 (74ACT374) แนนเกททั้งแปดตัวจะทำหน้าที่ร่วมกับไอ.ซี.U15 ทำหน้าที่เป็นเวิร์ดทริกเกอร์ (WORD TRIGGER) ยกตัวอย่างเช่น ต้องการตั้งให้เครื่องเริ่มทำการเก็บข้อมูลเมื่อมีข้อมูล OAAH เข้ามากระตุ้น วิธีการคือ ทำการส่งข้อมูล OAAH ไปค้างไว้ที่ไอ.ซี.U54 สัญญาณจากไอ.ซี.U54 จะถูกป้อนไปเป็นสัญญาณด้านเข้าของแนนเกท ฉะนั้นเมื่อสัญญาณด้านเข้า CH1~CH8 ที่ผ่านไอ.ซี.U55 มาเป็น OAAH เช่นเดียวกัน สัญญาณออกของแนนเกทจะเป็นดังนี้

	CH8	CH7	.....	CH2	CH1	
จาก U54	1	0	1	0	1	0	1	0	= AAH
จาก U55	1	0	1	0	1	0	1	0	= AAH
O/Pของแนนเกท	0	1	0	1	0	1	0	1	= 55H

ข้อมูล 55H นี้จะถูกป้อนไปยังสัญญาณด้านเข้า (P0~P7) ของ U15

ข้อมูลของเวิร์ดทริก 0AAH จะถูกนำมาทำคอมพลีเมนต์ของหนึ่ง(1'S COMPLEMENT) ผลที่ได้จะเป็น 55H ข้อมูลนี้จะถูกส่งออกไปค้างค่าไว้ที่สัญญาณด้านออกของไอ.ซี.U68 (74ACT374) ซึ่งจะถูกป้อนไปยังสัญญาณด้านเข้า (Q0~Q7)ของไอ.ซี.U15 (74ACT521)

ไอ.ซี.U15 จะทำหน้าที่เปรียบเทียบข้อมูลที่ป้อนเข้ามายังกลุ่มสัญญาณขา P กับขา Q เมื่อใดก็ตามที่สัญญาณที่ขา P และ Q เท่ากัน(ในที่นี้คือ 55H) สัญญาณด้านออกที่ขา 19 จะให้ตรรกะเป็น 0 ถ้าไม่เท่ากันจะให้ตรรกะเป็น 1

โดยอาศัยหลักการทำนองเดียวกัน ถ้าต้องการตั้งเวิร์ดทริกทั้ง 32 ช่องสัญญาณก็ต้องใช้วงจรเช่นเดียวกันกับที่ได้กล่าวมาแล้วข้างต้นดังวงจรในแผ่นที่ 3 , 4 และ 5 และเมื่อสัญญาณเข้าทั้ง 32 ช่องสัญญาณมีค่าเท่ากับเวิร์ดทริกซ์ทั้งหมด ตรรกะที่ขา 19 ของไอ.ซี. U15 , U14 , U13 และ U12 จะเป็น 0 ทั้งหมด สัญญาณเหล่านี้จะถูกส่งต่อไปเป็นสัญญาณด้านเข้าให้กับนอร์เกท U29 ตรรกะด้านออกของนอร์เกทจะให้ตรรกะ 1 ออกมา ในกรณีนี้

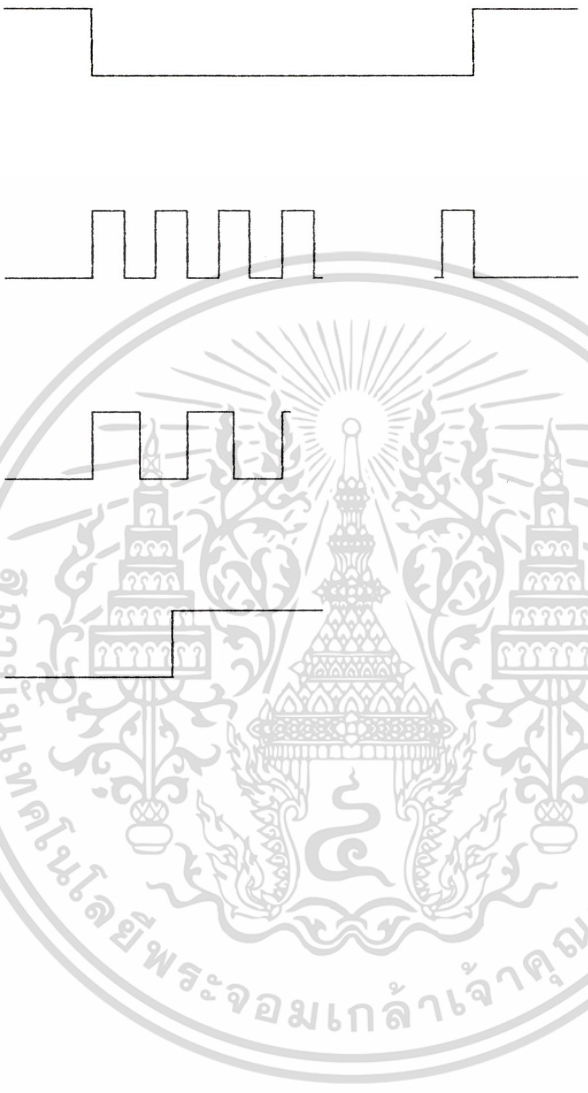
ในช่วงเวลาต่อมาเมื่อสัญญาณที่เข้าทางสายนำสัญญาณ CH1~CH32 เปลี่ยนแปลงไปมีค่าไม่เท่ากับเวิร์ดทริกตรรกะที่ขา 19 ของไอ.ซี.U15 , U14 , U13 และ U12 จะเปลี่ยนสภาวะไปเป็น 1 (บางตัวหรือทุกตัว) ทำให้ตรรกะของสัญญาณออกของ U29 เปลี่ยนกลับเป็นตรรกะ 0 เมื่อพิจารณาดูที่จุดนี้จึงเสมือนว่า เมื่อใดก็ตามที่ข้อมูลที่รับเข้ามามีค่าเท่ากับเวิร์ดทริกที่ตั้งเอาไว้ ที่จุดสัญญาณออกของ U29 จะให้สัญญาณนาฬิกาออกมา 1 ลูกคลื่น สัญญาณนี้จะถูกนำไปเป็นสัญญาณกระตุ้นต่อไป

ในบางกรณี การตั้งเวิร์ดทริกไม่จำเป็นจะต้องตั้งทุกบิท โดยละบางบิทไว้ไม่ให้ความสนใจ (DON'T CARE) บิทใดที่ต้องการละไว้โดยไม่ต้องสนใจ จะทำการส่งตรรกะ 0 ป้อนเข้าที่สัญญาณเข้าของแนนเกท ทำให้ตรรกะด้านออกของแนนเกทเป็น 1 เสมอ ฉะนั้นด้านขา Q ของไอ.ซี.U15 จะต้องได้รับตรรกะ 1 จากไอ.ซี.U68 ให้สอดคล้องกัน เช่น ถ้าให้ X แทนบิทที่ละไว้

	CH8	CH7					CH2	CH1
สัญญาณข้อมูล	0	1	0	1	X	X	1	0
การตั้งเวิร์ดทริก	0	1	0	1	0	0	1	0
O/Pแนนเกท(P)	1	0	1	0	1	1	0	1
ตรรกะที่ขา Q	1	0	1	0	1	1	0	1

จากที่แสดงด้านบน บิทใดที่ละไว้ไม่ต้องสนใจ จะต้องส่งตรรกะ 0 ไปยังสัญญาณด้านเข้าของแนนเกท และส่งตรรกะ 1 ไปยังสัญญาณด้านเข้าขา Q ที่สอดคล้องกับบิทที่ละไว้ ผลที่ได้ก็จะทำให้เกิดสภาวะ P=Q เช่นเดียวกับในกรณีข้างต้น

สัญญาณออกของไอ.ซี.U29 ในสภาวะปกติที่สัญญาณข้อมูลขา P ไม่เท่ากับข้อมูลของขา Q จะมีตรรกะเป็น 0 ส่งผลให้เกิดสภาวะการเคลียร์ขึ้นที่ไอ.ซี.U16 (74ACT163) เมื่อใดก็ตามที่ข้อมูลเท่ากับเวิร์ดทริก ตรรกะที่ขา 1 ซึ่งเป็นขาเคลียร์จะเปลี่ยนเป็นตรรกะ 1 ซึ่งยกเลิกการเคลียร์ ทำให้ไอ.ซี.U16 เริ่มทำการนับสัญญาณนาฬิกาที่มีความถี่ 50 เมกกะเฮิทซ์ ไอ.ซี.U16 จะทำงานร่วมกับไอ.ซี.U27 เพื่อทำหน้าที่ตรวจสอบว่าสัญญาณนาฬิกาที่ได้จากการเปรียบเทียบข้อมูลของเวิร์ดทริกนั้นเท่ากันจริงหรือไม่ (ในบางโอกาสตรรกะ 1 อาจจะเป็นสัญญาณที่เรียกว่า กลิทช์พัลซ์ (GLIFCH PULSE)) ขอให้พิจารณาจากรูปดังนี้



เมื่อสัญญาณตรรกะที่ขา CLR ของไอ.ซี.U16 เป็น 0 ไอ.ซี.U16 ซึ่งถูกตั้งไว้ให้ทำหน้าที่เป็นตัวนับเลขฐานสิบหก(โดยทำการต่อขา A , B , C , D ลงกราวด์) จะเริ่มทำการนับและให้สัญญาณออกมาที่ขา Q_A , Q_B , Q_C และ Q_D สัญญาณเหล่านี้จะถูกส่งต่อไปเข้าเป็นสัญญาณเข้ากลุ่ม A ของไอ.ซี.U27 (74ACT85) สัญญาณด้านเข้าอีกกลุ่ม คือ

กลุ่ม B จะรับมาจากไอ.ซี.U41 (74ACT374) ซึ่งจะเป็นตัวค้างการตั้งโปรแกรมการนับเพื่อตรวจสอบความยาวของสัญญาณ CLR เช่นถ้าต้องการให้เกิดการนับขึ้นเมื่อความกว้างของสัญญาณ CLR กว้างเท่ากับ 10 ลูกคลื่นของสัญญาณนาฬิกา 50 เมกกะเฮิทซ์ จะกระทำได้โดยทำการส่งข้อมูล 0AH ออกมา ทำการค้างค่าไว้ที่ไอ.ซี.U41 และเมื่อข้อมูล Q_A ถึง Q_D ผ่านการนับมาได้ครบ 10 ลูกคลื่น ข้อมูลกลุ่ม A เท่ากับ ข้อมูลกลุ่ม B สัญญาณ TRIGGER จะเปลี่ยนสภาวะจาก 0 เป็น 1 และเมื่อผ่านไปอีกหนึ่งลูกคลื่นของ 50 เมกกะ-เฮิทซ์ ข้อมูลกลุ่ม A จะไม่เท่ากับ ข้อมูลกลุ่ม B จึงทำให้ตรรกะของสัญญาณ TRIGGER เปลี่ยนมาเป็น 0

ในกรณีที่ข้อมูลที่เกิดจากกลิทช์พัลส์แม้ว่าตรรกะของขา CLR จะเป็น 0 แต่ช่วงเวลาจะน้อยจนกระทั่งการนับจะไม่เท่ากับข้อมูลที่ตั้งไว้ที่ขา B0~B3 ดังนั้นสัญญาณ TRIGGER จะไม่มีโอกาสที่เกิดสัญญาณนาฬิกาขึ้น การออกแบบในลักษณะนี้จะทำให้ผู้ใช้สามารถปรับตั้งการเปรียบเทียบความยาวของสัญญาณ CLR ได้ตามต้องการ

ดังที่ได้กล่าวมาแล้วว่าอัตราการสุ่มสัญญาณเมื่อให้สัญญาณนาฬิกาภายใน จะเลือกใช้การสุ่มสัญญาณในอัตรา 50 เมกกะเฮิทซ์ และสามารถลดทอนลงด้วยอัตราการลดทอนเป็น 5-2-1 เช่น 50 , 20 , 10 , 5 , 2 , 1 เมกกะเฮิทซ์ ซึ่งใช้วงจรในแผ่นที่ 6 จะเป็นส่วนที่ใช้สร้างสัญญาณนาฬิกาที่ใช้สุ่มข้อมูลเข้ามาในระบบ ฐานของสัญญาณนาฬิกาได้มาจากก้อนผลิกสองก้อน ซึ่งมีความถี่ 20 และ 50 เมกกะเฮิทซ์ และฐานนาฬิกาอีกตัวหนึ่งได้มาจากสัญญาณนาฬิกาจากวงจรภายนอก (POD CLK) กล่าวคือ การจะเลือกใช้ฐานของสัญญาณนาฬิกาจากก้อนผลิกก้อนใดก้อนหนึ่ง หรือสัญญาณนาฬิกาจากภายนอกขึ้นอยู่กับการเลือกสัญญาณ S1 และ S2 ที่ให้กับไอ.ซี.U1 (74ACT153) ดังตารางต่อไปนี้

S1	S0	Y(7)		Y(9)	
0	0	50	เมกกะเฮิทซ์	5	เมกกะเฮิทซ์
0	1	20	"	10	"
1	0	0	"	POD CLK	
1	1	0	"	20	เมกกะเฮิทซ์

โดยที่เอาพุทของ U1 ขาที่ 7 ซึ่งเป็นอินพุทให้กับ U9 (74F160A) ซึ่ง U9 จะเป็นวงจรหารความถี่ โดยที่ขาเอาพุท QA (14) จะเป็นค่าจากสัญญาณอินพุทหารด้วยสอง และขาเอาพุท QD (11) จะเป็นค่าจากสัญญาณอินพุทหารด้วยสิบ โดยส่งสัญญาณนาฬิกาต่อไปยังวงจร U8 (74LS390) , U19 (74LS390) และ U18 (74LS390) ซึ่งจะเป็นวงจรสำหรับการหารด้วย 10 ตามลำดับ และสัญญาณนาฬิกาที่ได้จากการเลือกและหารแล้วจะป้อนไปยัง U17 (74ACT151) จะเป็นตัวเปิดสัญญาณนาฬิกาตามต้องการออกไป ซึ่งได้จากการเลือกสัญญาณ S2 , S3 , และ S4 ป้อนให้กับ U17 นั่นเอง

สัญญาณนาฬิกาที่ใช้การสุ่มข้อมูลจากสายนำสัญญาณ CH1~CH32 (STCLK , $\overline{\text{STCLK}}$) จะถูกป้อนมายังขาสัญญาณเข้าของแอนด์เกทและออร์เกท ดังวงจรในแผ่นที่ 8 ในสภาวะเริ่มต้นก่อนที่จะมีเกิดสัญญาณ TRIGGER ฟลิปฟลอป U20 จะอยู่ในสภาวะปกติ ตรรกะที่ขา Q จะเป็น 0 ทั้งคู่ ทำให้เกิดการปิดกั้นสัญญาณ STCLK และ $\overline{\text{STCLK}}$ ไม่ให้ผ่านแอนด์เกทและออร์เกทออกไป เมื่อเกิดสัญญาณ TRIGGER มากระตุ้นฟลิปฟลอป ทำให้ตรรกะของขา Q เปลี่ยนเป็น 1 และ $\bar{Q}$ เปลี่ยนเป็นตรรกะ 0 ส่งผลให้เกิดสัญญาณนาฬิกาขึ้นที่จุดสัญญาณออกของออร์เกท U32 คือสัญญาณ AQCLK และสัญญาณ $\overline{\text{WR}}$

สัญญาณนาฬิกา AGCLK จะถูกส่งต่อไปเป็นสัญญาณนาฬิกาขา 11 ซึ่งเป็นขา CP ของไอ.ซี.U55 นั้นหมายความว่า การสุ่มข้อมูลของ U55 จะเป็นไปตามสัญญาณนาฬิกา AQCLK ซึ่งสามารถเลือกได้ตามต้องการโดยมีอัตราการทดขนาด 5-2-1 ดังได้อธิบายผ่านมาแล้ว

ข้อมูลที่สุ่มได้จากไอ.ซี.U55 ส่วนหนึ่งจะถูกส่งผ่านมาเป็นสัญญาณด้านเข้าที่ขาข้อมูลของแรมแบบแคช (VT62A168) ขนาด 16 บิท x 8 กิโลไบท์สองตัว (U34 , U35) บัสข้อมูลแต่ละเส้นของแรมแบบแคช จะรองรับสัญญาณที่สุ่มเข้ามาจากสายนำสัญญาณแต่ละช่องสัญญาณ ดังวงจรในแผ่นที่ 7 เรียงกันดังนี้ คือ

บิท	ไอ.ซี.	ช่องนำสัญญาณ
D0	U34	CH1
D1	U34	CH2
D2	U34	CH3
D3	U34	CH4
.	.	.
.	.	.
.	.	.
D15	U34	CH16
D0	U35	CH17
D1	U35	CH18
.	.	.
.	.	.
.	.	.
D15	U35	CH32

วงจรอีกส่วนหนึ่งในวงจรแผ่นนี้ คือ วงจรนับตำแหน่งแอดเดรสของแรมแบบแคช ได้แก่ ไอ.ซี.U46 , 45 , 44 และ 43 (74ACT163) สัญญาณนาฬิกาของ U46 ได้จากสัญญาณ AQCLK เช่นกัน ดังนั้นวงจรนับตำแหน่งแอดเดรสนี้ จะทำงานสอดคล้องกับการสุ่มสัญญาณของ U55 , 69 , 56 , 70 และสัญญาณ $\overline{WR}$ ของแรมแบบแคชเสมอ

เมื่อวงจรนับตำแหน่งแอดเดรสนับจนครบ 8 กิโลไบท์ จะทำให้ตรรกะในสายสัญญาณ CA0~CA12 เป็น 1 ทั้งหมดทุกเส้น สัญญาณส่วนนี้ส่วนหนึ่ง จะถูกส่งต่อไปยังบัสแอดเดรสของแรมแบบแคช อีกส่วนหนึ่งจะต่อมายังแอนด์เกทในวงจรแผ่นที่ 8 เมื่อ CA0~CA12 เป็นตรรกะ 1 ทั้งหมด จะทำตรรกะด้านออกของแอนด์เกท U31 ขา 8 เป็น 1 ส่งผลให้เกิดการกระตุ้น ฟลิปฟลอป U20 ให้เปลี่ยนสถานะขา Q จากเดิมเป็นตรรกะ 0 เป็นตรรกะ 1 ซึ่งจะเป็นการปิดเกทไม่ยอมให้สัญญาณนาฬิกา STCLK ผ่านออกไปนั้น หมายความว่า สัญญาณ AQCLK จะไม่มีอีกต่อไปทำให้ส่วนสุ่มสัญญาณหยุดทำงาน และส่วนนับตำแหน่งแอดเดรสหยุดทำงานไปโดยอัตโนมัติ

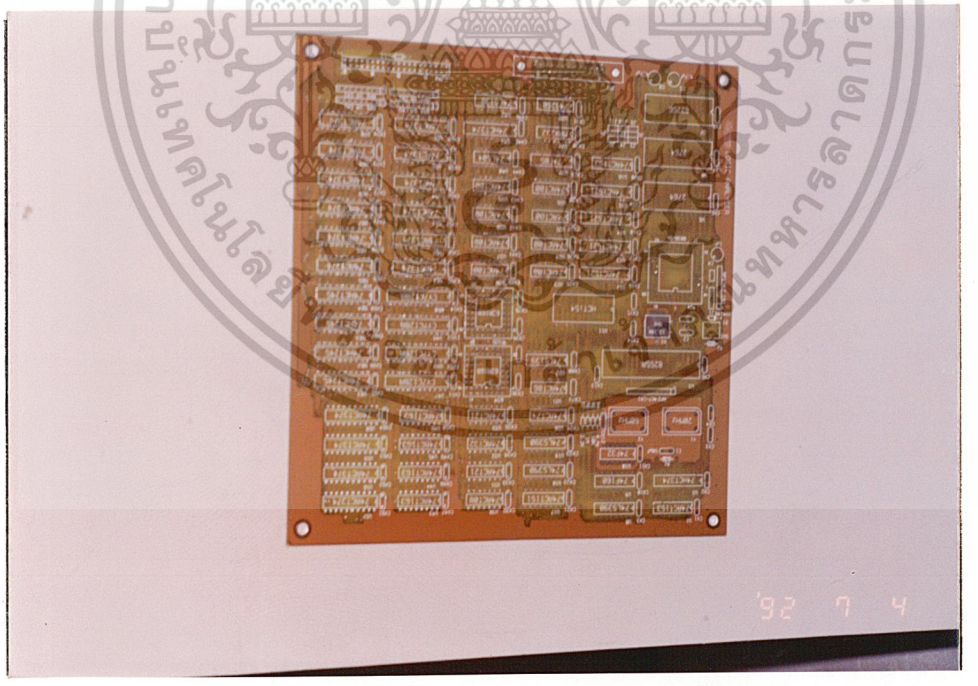
ในขณะนี้ ข้อมูลที่สุ่มเข้ามาจะถูกเก็บไว้ในแรมแบบแคชเรียบร้อยแล้ว พร้อมที่จะถูกอ่านออกมาเพื่อแสดงผลไป

ส่วนควบคุมระบบทั้งหมดได้แก่ วงจรในแผ่นที่ 1 ประกอบด้วยไมโครโปรเซสเซอร์เบอร์ HD64180 พอร์ทอนุกรมของ HD64180 จะต่อเข้ากับไอ.ซี.U28 (MAX232) ซึ่งทำหน้าที่เปลี่ยนระดับสัญญาณจาก TTL เป็นมาตรฐานของ RS232 เพื่อเชื่อมต่อข้อมูลเข้ากับเครื่องคอมพิวเตอร์ส่วนบุคคล IBM PC หน่วยความจำแรมของระบบจะมีขนาด 32 กิโลไบท์ (U7 , HM62256) ส่วน U5 และ U6 เป็นหน่วยความจำรอม สำหรับเก็บโปรแกรมจัดการระบบ ไอ.ซี.U3 (PPI , 8255) ทำหน้าที่เป็นบัสเฟอร์ระหว่างบัสของระบบที่กล่าวผ่านมาแล้วข้างต้น กับ บัสของระบบไมโครโปรเซสเซอร์ HD64180 โดยพอร์ท A และ พอร์ท C จะถูกทำตัวเสมือนกับบัสข้อมูลของส่วนตรรกะภายนอก โดยพอร์ท A ถือเป็นบัสข้อมูลที่ส่งออกไปยังส่วนตรรกะ และพอร์ท B ถือเป็นบัสข้อมูลที่รับเข้ามาจากส่วนตรรกะ สำหรับพอร์ท B0 ถึง พอร์ท B3 จะถูกนำมาถอดรหัสโดยไอ.ซี.U11 (74ACT154) สัญญาณออกของ U11 จะถูกนำไปควบคุมไอ.ซี.ต่างๆที่ใช้ในการค้างข้อมูล พอร์ท B4 จะใช้ควบคุมฟลิปฟลอปที่ทำหน้าที่จ่ายสัญญาณนาฬิกา AQCLK และสัญญาณ $\overline{WR}$ พอร์ท B5 จะทำหน้าที่สร้างสัญญาณนาฬิกา STEP ในกรณี ที่ต้องการอ่านข้อมูลจากแรมแบบแคชของส่วนตรรกะเข้ามายังแรมของไมโครโปรเซสเซอร์ เมื่อส่งสัญญาณ STEP มา 1 ลูกคลื่น ส่วนนับตำแหน่งแอดเดรสจะเพิ่มค่าขึ้นหนึ่งค่า ทำให้แรมแบบแคชส่งข้อมูล

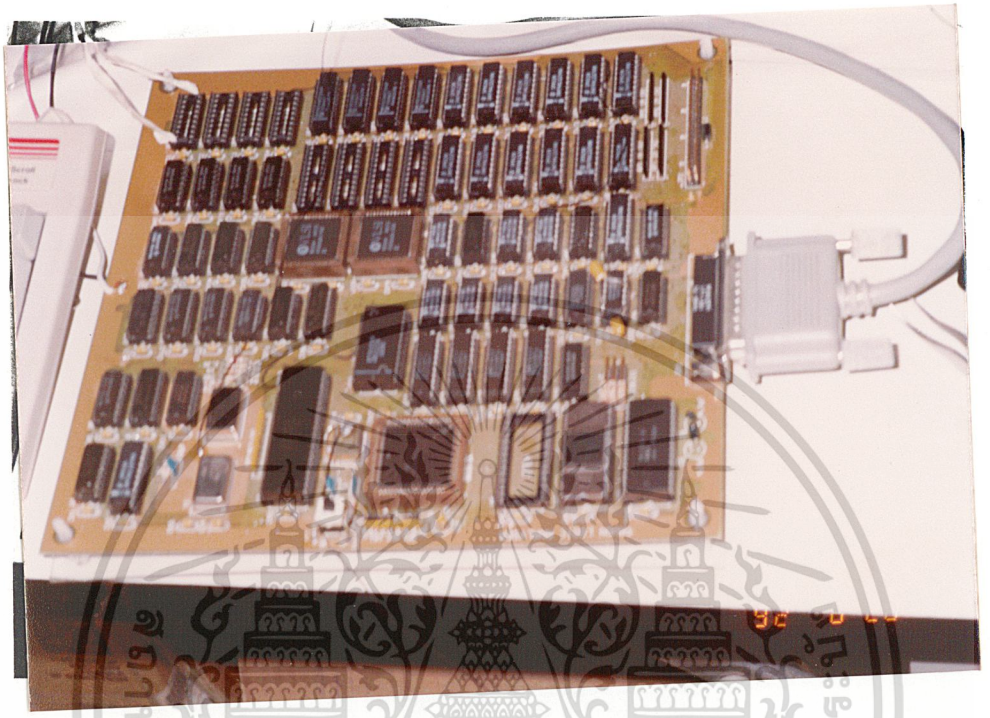
มายังพอร์ท C ไมโครโปรเซสเซอร์จะรับข้อมูลเข้ามาเก็บไว้ในแรม กระทำเช่นนี้จนกระทั่งครบ 8 กิโลไบท์ ผลที่ได้ คือ ข้อมูลจะถูกอ่านเข้ามาไว้ในแรมของระบบไมโครโปรเซสเซอร์ทั้งหมด

จากนี้ ไมโครโปรเซสเซอร์จะส่งข้อมูลเหล่านี้ ผ่านทางพอร์ทอนุกรม RS232 ให้เครื่องคอมพิวเตอร์ส่วนบุคคลทำการแสดงผลต่อไป

ซึ่งจากรูปที่ 3.1.1 แสดงถึงแผงวงจร (PRINTED CIRCUIT BOARD) ต้นแบบที่ใช้ในการพัฒนาไมโครโปรเซสเซอร์เบอร์ HD64180 ให้เป็นเครื่องวิเคราะห์สัญญาณทางตรรก ส่วนในรูปที่ 3.1.2 เป็นภาพแสดงถึงแผงวงจรพร้อมอุปกรณ์ที่ใช้งาน และในรูปภาพที่ 3.1.3 เป็นภาพแสดงถึงการต่อวงจรใช้สำหรับการทดสอบ



รูปภาพที่ 3.1.1 แสดงถึงแผงวงจรต้นแบบ



รูปภาพที่ 3.1.2 แสดงแผงวงจรพร้อมอุปกรณ์ที่ใช้งาน

รูปภาพที่ 3.1.3 แสดงถึงอุปกรณ์ที่ใช้ทำการทดสอบ

3.2 ขั้นตอนการใช้งานของเครื่องวิเคราะห์สัญญาณทางตรรก

3.2.1 ด้าน IBM-PC ทำการรับโปรแกรมที่มีชื่อว่า TERMINAL.EXE เป็นอันดับแรกโดยโปรแกรมจะให้เลือกพอร์ทของ RS-232 และเลือกอัตราของโบร์ดที่ต้องการใช้ โดยทำการเลือกพอร์ทที่ 1 ให้สอดคล้องกับเครื่อง IBM PC (COM. PORT#1) และเลือกอัตราโบร์ดเช่นเดียวกับที่ตั้งไว้ ในเครื่องวิเคราะห์สัญญาณทางตรรก ขั้นตอนนี้เครื่อง IBM PC จะส่งสัญญาณ SPACE (20H) ไปยังเครื่องวิเคราะห์สัญญาณทางตรรก เพื่อทดสอบการเชื่อมต่อสัญญาณ

ด้านเครื่องวิเคราะห์สัญญาณทางตรรก เมื่อผ่านขั้นตอนของ IBM PC แล้ว ให้ทำการป้อนไฟ (PDWER SUPPLY) ให้แก่วงจรเครื่องวิเคราะห์สัญญาณทางตรรก ซึ่งก็จะทำงาน พร้อมทั้งสัญลักษณ์พรอมป์ (READY>) ออกไปยังเครื่อง IBM PC ถ้าเครื่องหมาย READY> ปรากฏบนจอภาพแล้ว แสดงว่าพอร์ทสื่อสารทั้งสองด้านได้เชื่อมต่อเข้าหากันเรียบร้อยแล้ว และเครื่องวิเคราะห์สัญญาณทางตรรกพร้อมที่จะรับคำสั่งต่อไป ข้อความบนจอภาพจะเป็นดังรูปที่ 3.2.1

```
PC/XT Training Kit
Copyright 1991
by  Wichit T.
**********
```

```
Communication link set up
type H for display Help
```

```
Ready>
```

รูปที่ 3.2.1 แสดงข้อความบนจอภาพเมื่อเครื่องพร้อมจะรับคำสั่ง

3.2.2 ทดลองเรียกดูคำสั่ง H (ELP)

เครื่องวิเคราะห์สัญญาณทางตรรกจะแสดงรายละเอียดของคำสั่งที่ใช้งาน แสดงขึ้นบนจอภาพดังรูปที่ 3.2

```
Ready>H
  User RAM area start at address D000h

  Command used follow:

  B SSSS                    for break point
  C SSSS FFFF TTTT          for copy data from one area to other
  D SSSS FFFF               for dump data from memory
  E SSSS                    for edit data in memory
  F SSSS FFFF HH HH HH      for find data 'hh hh hh' in memory
  G SSSS                    for execute program
  H                         for display this message
  I SSSS FFFF HH            for insert 'hh' into memory
  L                         for download from terminal
  R                         for read data from cache ram
  S                         for set sampling rate
  T                         for set triggering word
  W                         for write cache ram
  PS                        for display pulse train

Ready>
```

รูปที่ 3.2.2
แสดงรายละเอียดของชุดคำสั่งที่ใช้กับเครื่องวิเคราะห์สัญญาณทางตรรก
หลังจากกดคีย์ตัวอักษร "H"

3.2.3 ทดลองคำสั่ง T (TRIGGERING WORD)

คำสั่งนี้จะเป็นที่ใช้การตั้งค่าของเวิร์ดทริกที่เราต้องการ กล่าวคือ เมื่อใดก็ตามที่ข้อมูลรับเข้ามามีค่าเท่ากับเวิร์ดทริกที่ตั้งเอาไว้แล้ว เครื่องวิเคราะห์สัญญาณทางตรรกจะเริ่มตรวจรับข้อมูล ดังที่กล่าวมาแล้วว่า เราจะแบ่งสัญญาณ 32 ช่องสัญญาณ เป็น 4 กลุ่ม ฉะนั้นเวิร์ดทริกก็จะมีอยู่ 4 กลุ่ม เช่นกัน (CHANNEL 0 , 1 , 2 , 3)เครื่องวิเคราะห์สัญญาณทางตรรกจะแสดงรายละเอียดของคำสั่งที่ใช้งาน ซึ่งแสดงบนจอภาพดังรูปที่ 3.2.3 ซึ่งแสดงถึงการเลือกเวิร์ดทริกตามที่ผู้ใช้ต้องการ

```
Ready>T
Set trigger bit, 1 for logic 1, 0 for logic 0 and D for don't care

Channel 0 for bit 7 .......bit  0

Channel 1 for bit 15 ......bit  8

Channel 2 for bit 23 ......bit 16

Channel 3 for bit 24 ......bit 31

Select channel for setting : 0
Set trigger bits of CH0 : DDDDDDDD

Ready>T
Set trigger bit, 1 for logic 1, 0 for logic 0 and D for don't care

Channel 0 for bit 7 .......bit  0

Channel 1 for bit 15 ......bit  8

Channel 2 for bit 23 ......bit 16

Channel 3 for bit 24 ......bit 31

Select channel for setting : 1
Set trigger bits of CH1 : 00010011
Ready>T
Set trigger bit, 1 for logic 1, 0 for logic 0 and D for don't care

Channel 0 for bit 7 .......bit  0

Channel 1 for bit 15 ......bit  8

Channel 2 for bit 23 ......bit 16

Channel 3 for bit 24 ......bit 31

Select channel for setting : 2
Set trigger bits of CH2 : 10000000

Ready>T
Set trigger bit, 1 for logic 1, 0 for logic 0 and D for don't care

Channel 0 for bit 7 .......bit  0

Channel 1 for bit 15 ......bit  8

Channel 2 for bit 23 ......bit 16

Channel 3 for bit 24 ......bit 31

Select channel for setting : 3
Set trigger bits of CH3 : 11110000
```

รูปที่ 3.2.3
แสดงรายละเอียดของชุดคำสั่งหลังจากกดคีย์ตัวอักษร "T"

3.2.4 ทดลองคำสั่ง S (AMPLING RATE)

คำสั่งนี้จะเป็นการแสดงค่าอัตราการสุ่มสัญญาณที่ต้องการ ดังในรูปที่ 3.4 ซึ่งเราสามารถเลือกอัตราการสุ่มสัญญาณได้ 2 กรณี

1. เราเลือกอัตราการสุ่มสัญญาณจากภายนอก โดยใช้คำสั่ง S และ E+ หรือ S และ E- (เครื่องหมายบวก (+) แสดงว่าเราไม่กลับเฟสของสัญญาณ และเครื่องหมายลบ (-) แสดงว่าเราผ่านวงจรกลับเฟสของสัญญาณ)

2. เราเลือกค่าอัตราการสุ่มสัญญาณจากภายในที่ความถี่ 5 เมกกะเฮิทซ์

```
Ready>S E+

Ready>S E-


Ready>S 5000000
```

รูปที่ 3.2.4
แสดงรายละเอียดของชุดคำสั่งที่ใช้กับเครื่องวิเคราะห์สัญญาณทางตรรก
หลังจากกดคีย์ตัวอักษร "S"

3.2.5 ทดลองคำสั่ง W (RITE CACHE RAM)

คำสั่งนี้จะเป็นการสั่งให้เครื่องวิเคราะห์สัญญาณทางตรรก นำสัญญาณที่ตรวจจับได้เก็บไว้ในแคชแรม

```
Ready>W
 write cache ram
```

รูปที่ 3.2.5
แสดงรายละเอียดของชุดคำสั่งที่ใช้กับเครื่องวิเคราะห์สัญญาณทางตรรก
หลังจากกดคีย์ตัวอักษร "W"

3.2.6 ทดลองคำสั่ง R (EAD DATA FROM CACHE RAM)

คำสั่งนี้จะเป็นการสั่งให้เครื่องวิเคราะห์สัญญาณทางตรรก นำข้อมูลที่เก็บไว้ในแคชแรม ไปเก็บไว้ในหน่วยความจำของแรมในเครื่องคอมพิวเตอร์ส่วนบุคคล เพื่อแสดงผลสภาวะทางตรรกบนจอภาพ เมื่อได้รับคำสั่ง PF5 (DISPLAY PULSE TRAIN)

```
Ready>R
Ready>
```

รูปที่ 3.2.6
แสดงรายละเอียดของคำสั่งที่ใช้กับเครื่องวิเคราะห์สัญญาณทางตรรก
หลังจากกดคีย์ตัวอักษร "R"

3.2.7 ทดลองคำสั่ง D (UMP)

คำสั่งนี้จะเป็นการแสดงค่าในหน่วยความจำขึ้นมาในจอภาพ เมื่อทำการดัมหน่วยความจำจากตำแหน่งที่ 8000 H จะเห็นโปรแกรมที่เป็นภาษาเครื่องปรากฏบนจอภาพ ถ้าคีย์คำสั่ง "D 8000" ที่เทอร์มินอล ผลที่แสดงออกมาจะเป็นดังรูปที่ 3.2.7

```
Ready>D 8000

      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F

8000  B7 13 80 F0 ED 14 80 F0  44 15 80 F0 1F 16 80 F0    ........ D.......
8010  CB 17 80 F0 4F 18 80 F0  CB 19 80 F0 CF 1A 80 F0    ....O... ........
8020  CB 1B 80 F0 8F 1C 80 F0  CD 1D 80 F0 23 1E 80 F0    ........ ....#...
8030  80 1F 80 F0 0E 23 80 F0  02 21 80 F0 0D 25 80 F0    .....#.. .!...%..
8040  C2 26 80 F0 25 27 80 F0  80 28 80 F0 0D 25 80 F0    .&..%'.. .(...%..
8050  C2 26 80 F0 25 27 80 F0  80 28 80 F0 C9 29 80 F0    .&..%'.. .(...)..
8060  C3 20 80 F0 00 21 80 F0  80 22 80 F0 11 00 80 F0    . ...!.. ."......
8070  FF 01 80 F0 FF 02 80 F0  01 03 80 F0 01 04 80 F0    ........ ........
8080  00 05 80 F0 21 06 80 F0  00 07 80 F0 90 08 80 F0    ....!... ........
8090  3E 09 80 F0 FE 0A 80 F0  77 0B 80 F0 E5 0C 80 F0    >....... w.......
80A0  E1 0D 80 F0 23 0E 80 F0  2B 0F 80 F0 88 10 80 F0    ....#... +.......
80B0  86 11 80 F0 AF 12 80 F0  B7 13 80 F0 ED 14 80 F0    ........ ........
80C0  44 15 80 F0 1F 16 80 F0  CB 17 80 F0 4F 18 80 F0    D....... ....O...
```

รูปที่ 3.2.7
แสดงการดัมหน่วยความจำในตำแหน่งที่ต้องการขึ้นมาบนจอภาพ

3.3 ผลการทดลองใช้งาน

ผลที่ได้จากการทดลองใช้งานที่ได้นั้น เครื่องวิเคราะห์สัญญาณทางตรรกชุดนี้สามารถตรวจรับข้อมูลได้ถูกต้องตามทฤษฎีที่ออกแบบไว้ ดังที่กล่าวมาแล้วในบทที่ 3 โดยเครื่องวิเคราะห์สัญญาณทางตรรก สามารถแสดงผลออกมาในรูปสภาวะทางตรรกบนจอภาพได้ถูกต้อง และสามารถเลื่อนตัวชี้ตำแหน่งไปอ่านค่าตามตำแหน่งต่างๆที่ต้องการได้ถูกต้อง กล่าวคือได้ป้อนสัญญาณทดสอบให้เครื่องวิเคราะห์สัญญาณทางตรรกตรวจรับ ซึ่งเมื่อใช้คำสั่ง D (UMP) ซึ่งเป็นคำสั่งจะแสดงค่าในหน่วยความจำขึ้นมาบนจอภาพ โดยการดัมหน่วยความจำจากตำแหน่ง 8000H จะได้ผลแสดงออกมาบนจอภาพดังรูปภาพที่ 3.3.1 ซึ่งข้อมูลได้ถูกต้องตามที่เราป้อนสัญญาณทดสอบเข้าไป

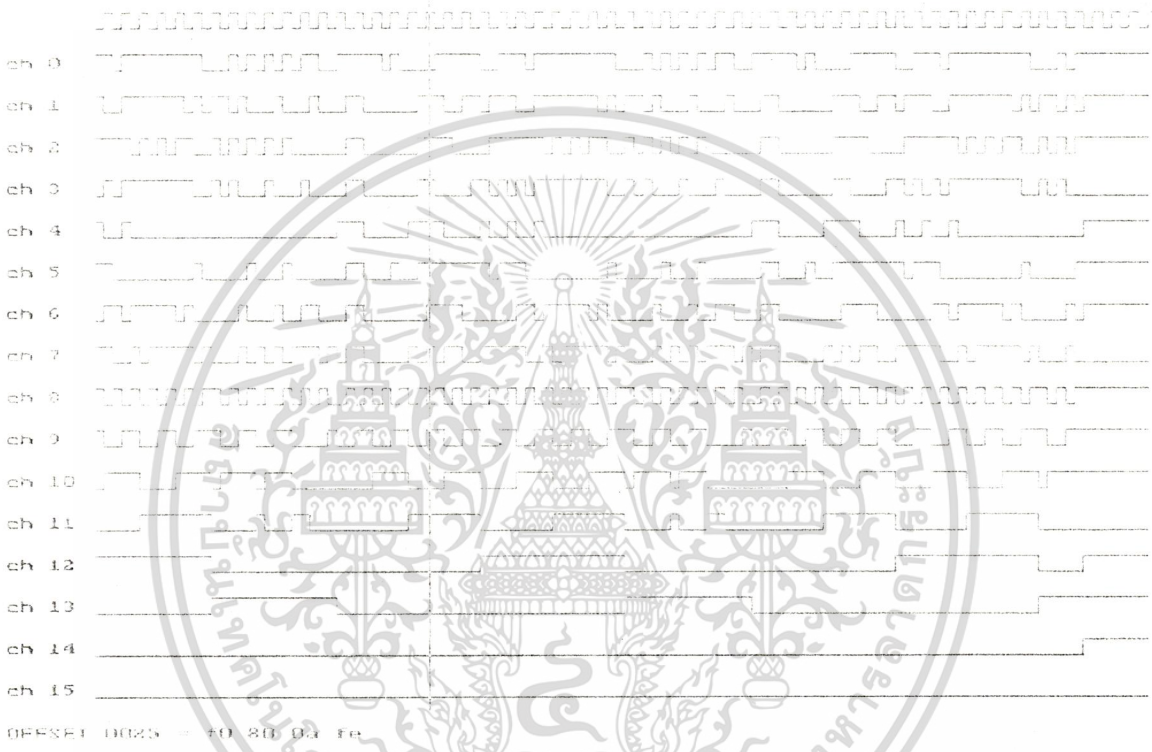
```
Ready>D 8000

      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F

8000  B7 13 80 F0 ED 14 80 F0  44 15 80 F0 1F 16 80 F0   ........ D.......
8010  CB 17 80 F0 4F 18 80 F0  CB 19 80 F0 CF 1A 80 F0   ....O... ........
8020  CB 1B 80 F0 8F 1C 80 F0  CD 1D 80 F0 23 1E 80 F0   ........ ....#...
8030  80 1F 80 F0 0E 23 80 F0  02 24 80 F0 0D 25 80 F0   .....#.. .$...%..
8040  C2 26 80 F0 25 27 80 F0  80 28 80 F0 0D 25 80 F0   .&..%'.. .(...%..
8050  C2 26 80 F0 25 27 80 F0  80 28 80 F0 C9 29 80 F0   .&..%'.. .(...)..
8060  C3 20 80 F0 00 21 80 F0  80 22 80 F0 11 00 80 F0   . ...!.. ."......
8070  FF 01 80 F0 FF 02 80 F0  01 03 80 F0 01 04 80 F0   ........ ........
8080  00 05 80 F0 21 06 80 F0  00 07 80 F0 00 08 80 F0   ....!... ........
8090  3E 09 80 F0 FE 0A 80 F0  77 0B 80 F0 E5 0C 80 F0   >....... w.......
80A0  E1 0D 80 F0 23 0E 80 F0  2B 0F 80 F0 B8 10 80 F0   ....#... +.......
80B0  86 11 80 F0 AF 12 80 F0  B7 13 80 F0 ED 14 80 F0   ........ ........
80C0  44 15 80 F0 1F 16 80 F0  CB 17 80 F0 4F 18 80 F0   D....... ....O...
80D0  CB 19 80 F0 CF 1A 80 F0  CB 1B 80 F0 8F 1C 80 F0   ........ ........
80E0  CD 1D 80 F0 23 1E 80 F0  80 1F 80 F0 0E 23 80 F0   ....#... .....#..
80F0  02 24 80 F0 0D 25 80 F0  C2 26 80 F0 25 27 80 F0   .$...%.. .&..%'..
8100  80 28 80 F0 0D 25 80 F0  C2 26 80 F0 25 27 80 F0   .(...%.. .&..%'..
8110  80 28 80 F0 C9 29 80 F0  C3 20 80 F0 00 21 80 F0   .(...).. . ...!..
8120  80 22 80 F0 11 00 80 F0  FF 01 80 F0 FF 02 80 F0   ."...... ........
8130  01
Ready>
```

รูปภาพที่ 3.3.1

แสดงผลการดัมหน่วยความจำในตำแหน่งที่ตรวจรับสัญญาณได้ขึ้นมาบนจอภาพ

จากนั้นเราใช้คำสั่ง W (RITE) , R (EAD) และ PF5 เพื่อให้เครื่องวิเคราะห์สัญญาณทางตรรก แสดงผลออกมาในรูปของสภาวะทางตรรกบนจอภาพ ซึ่งรูปภาพที่ 3.3.2 แสดงผลสภาวะทางตรรกของข้อมูลที่ช่องสัญญาณ CH0~CH15 ที่ตัวชี้ตำแหน่งที่ 0025 และใช้คำสั่งลูกศรลง (DOWN ARROW) ก็จะได้ผลดังภาพที่ 3.3.3 ซึ่งแสดงผลสภาวะทางตรรกของข้อมูลที่ช่องสัญญาณ CH16~CH31 ในตำแหน่งตัวชี้ตำแหน่งที่ 0025 เช่นเดียวกัน จากนั้นเราใช้คำสั่ง CTRL และ ลูกศรทางขวา (RIGHT ARROW) ก็จะได้ผลของสภาวะทางตรรกในหน้าถัดไป ซึ่งได้เป็นรูปภาพที่ 3.3.4 ซึ่งแสดงผลสภาวะทางตรรก ข้อมูลที่ช่องสัญญาณ CH0~CH15 ที่ตัวชี้ตำแหน่งที่ 008A (ผลหน้าถัดมา) โดยใช้คำสั่งลูกศรลงจะได้ผลดังรูปภาพที่ 3.3.5 ซึ่งแสดงผลสภาวะทางตรรกข้อมูลที่ช่องสัญญาณ CH16~CH31 ที่ตัวชี้ตำแหน่งที่ 008A เช่นกัน ส่วนรูปภาพที่ 3.3.6 และรูปภาพที่ 3.3.7 เป็นภาพถ่ายที่ได้จากจอภาพ ซึ่งแสดงผลสภาวะทางตรรก

จากผลการทดลองจะเห็นได้ว่าได้ผลที่น่าพอใจ แต่ก็ยังมีหัวข้อที่ควรปรับปรุงเพื่อพัฒนาให้เครื่องวิเคราะห์สัญญาณทางตรรกเครื่องนี้ สามารถใช้งานได้อย่างมีประสิทธิภาพและสมบูรณ์ยิ่งขึ้น ซึ่งจะกล่าวในหัวข้อเสนอแนะ



รูปภาพที่ 3.3.2

แสดงผลสภาวะทางตรรกของข้อมูล

ช่องสัญญาณ CH0~CH15 ณ. ตัวชี้ตำแหน่งที่ 0025

รูปภาพที่ 3.3.3

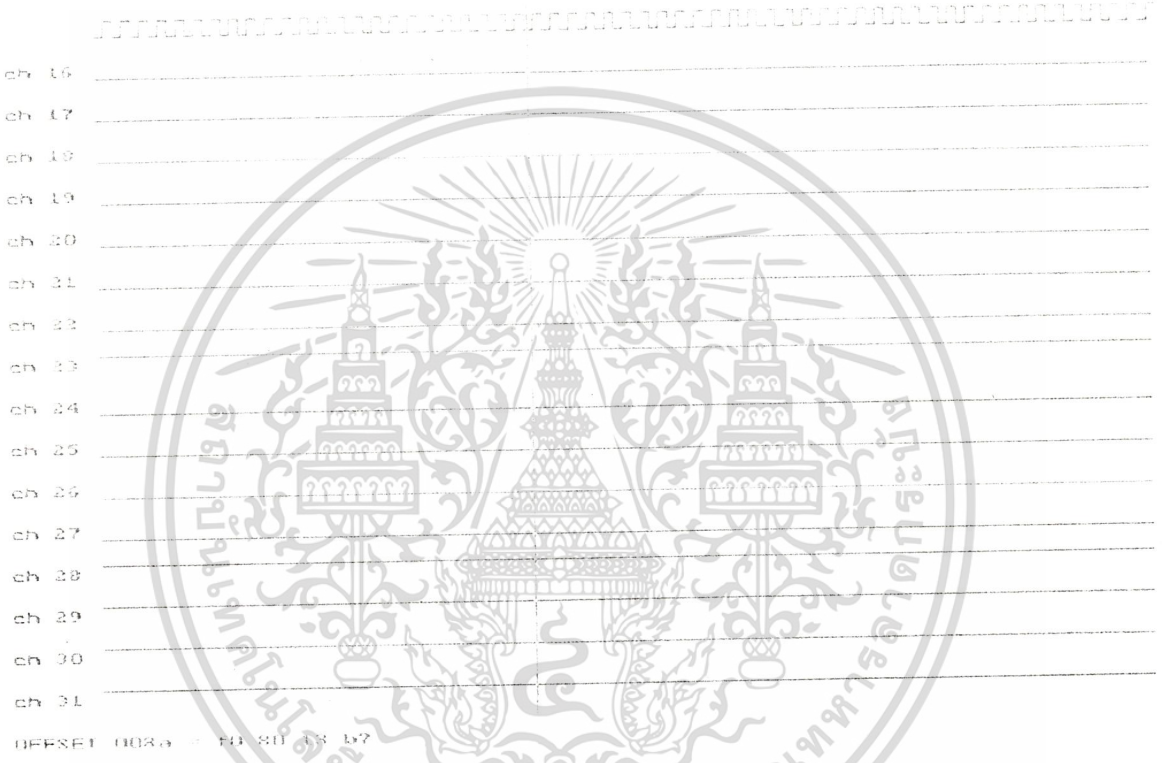
แสดงผลลภาวะทางตรรกของข้อมูล

ช่องสัญญาณ CH16~CH31 ณ. ตัวชี้ตำแหน่งที่ 0025

รูปภาพที่ 3.3.4

แสดงผลสภาวะทางตรรกของข้อมูล

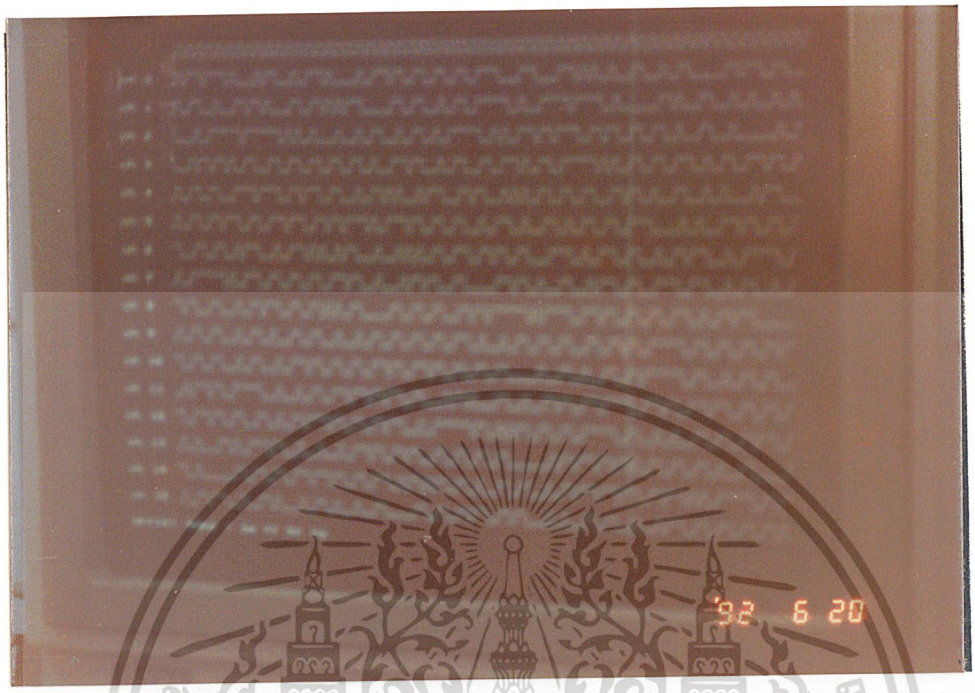
ช่องสัญญาณ CH0~CH15 ณ. ตัวชี้ตำแหน่งที่ 008A



รูปภาพที่ 3.3.5

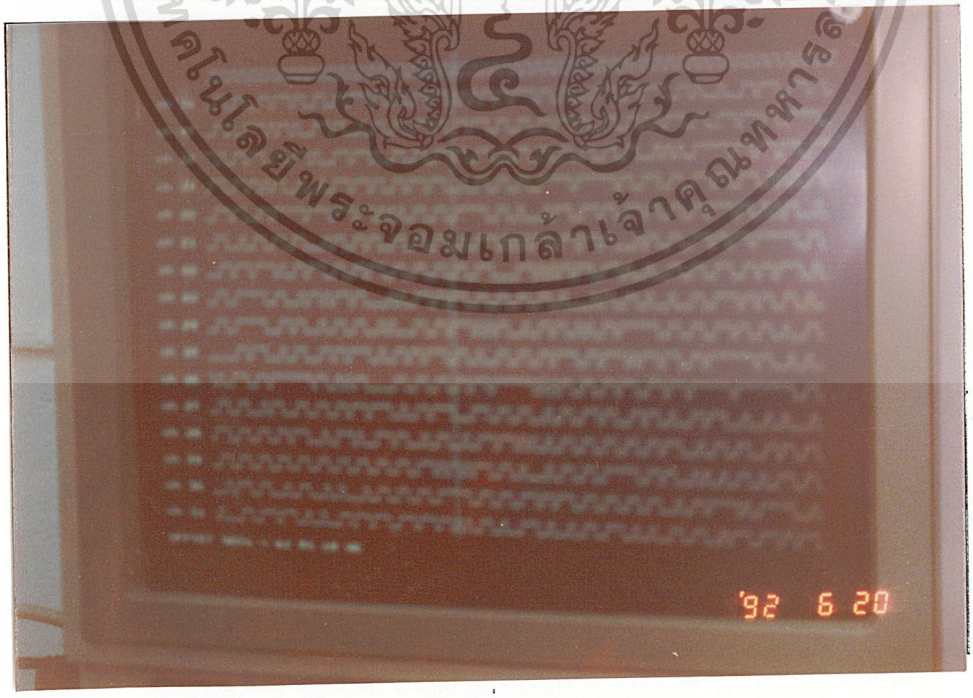
แสดงผลสภาวะทางตรรกของข้อมูล

ช่องสัญญาณ CH17~CH31 ณ. ตัวชี้ตำแหน่ง 008A



รูปภาพที่ 3.3.6

ภาพถ่ายจากจอภาพ แสดงผลสภาวะทางตรรก

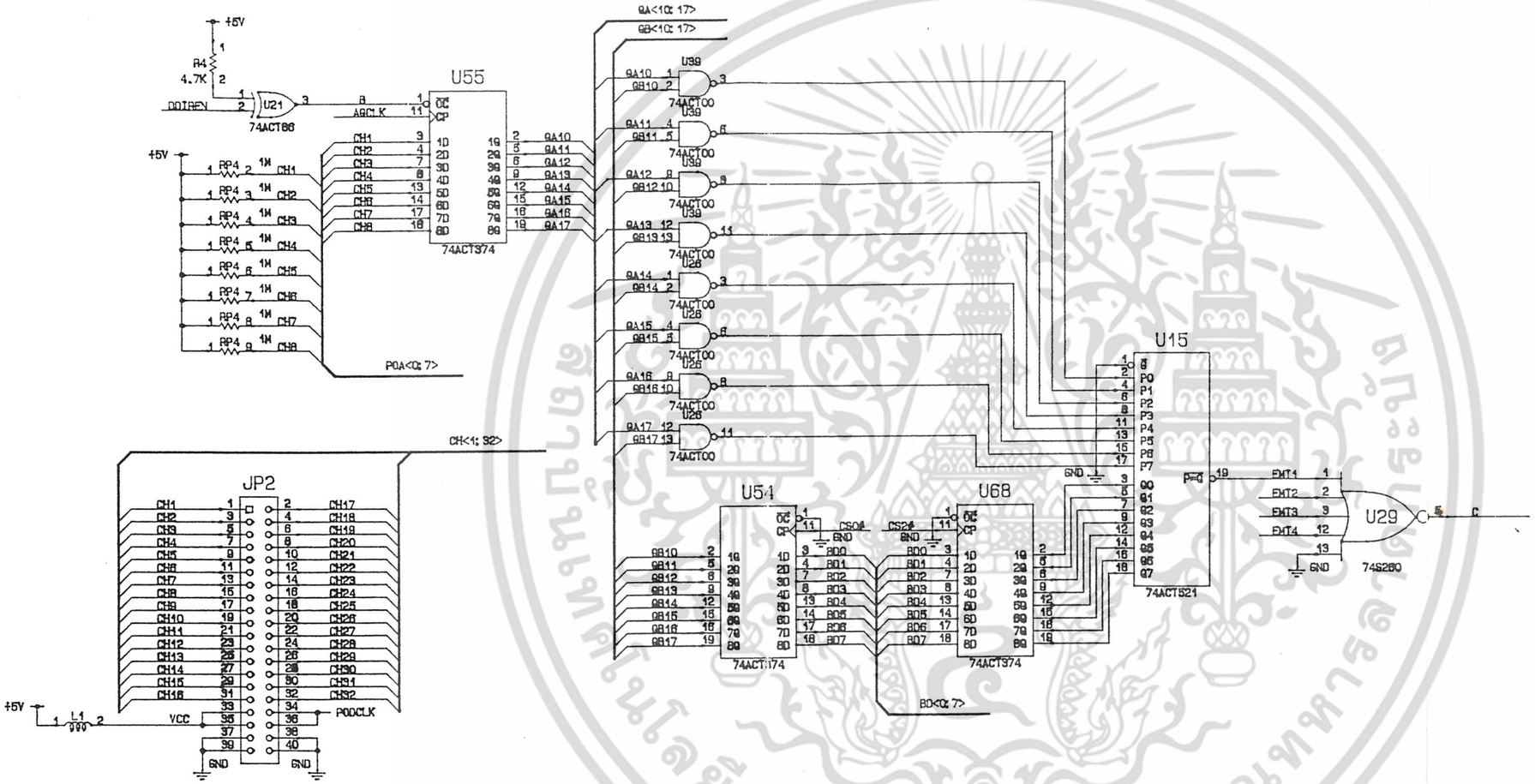


รูปภาพที่ 3.3.7

ภาพถ่ายจากจอภาพ แสดงผลสภาวะทางตรรก

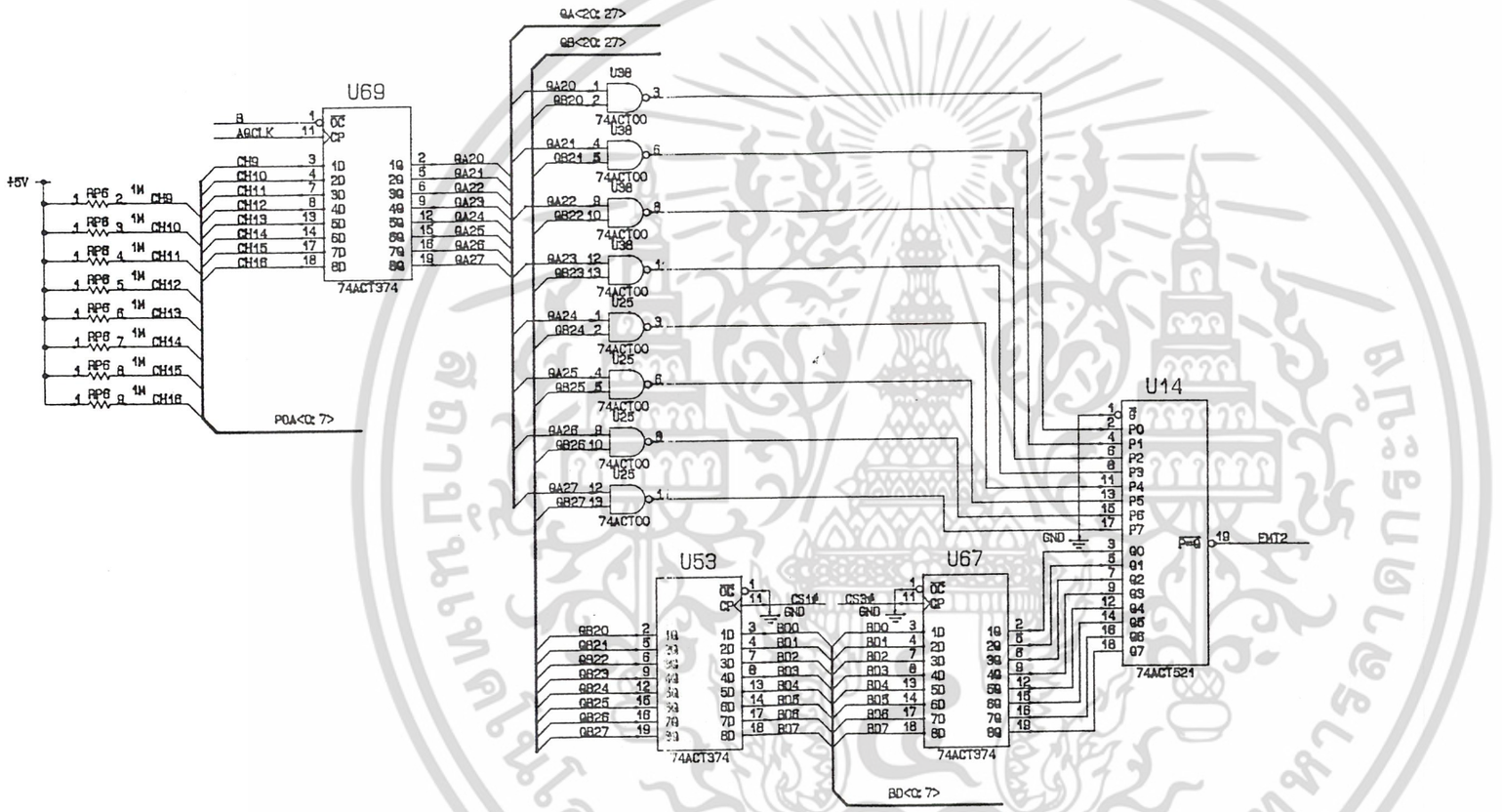
วงจรของ เครื่องวิเคราะห์สัญญาณทางตรรก แผ่นที่ 1

แสดงถึงวงจรของหน่วยประเมินผลกลาง , วงจรส่วนของ PPI และ หน่วยความจำเอสแรม



วงจรของเครื่องวิเคราะห์สัญญาณทางตรรก แผ่นที่ 2

แสดงถึงวงจรการเปรียบเทียบสำหรับ เวิร์ดทริกสำหรับข้อมูล CH1~CH8 (ข้อมูลกลุ่มที่ 1)

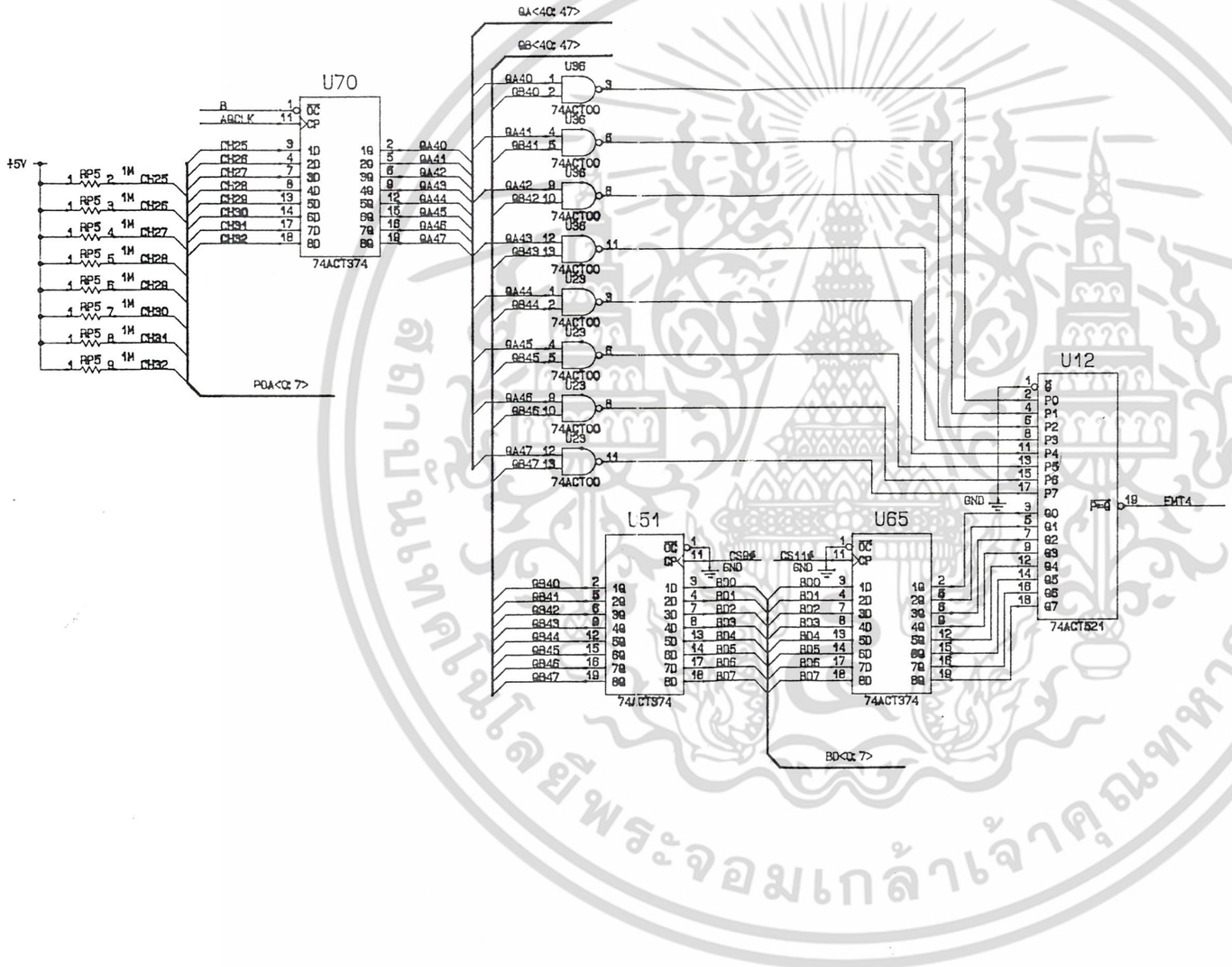


วงจรของเครื่องวิเคราะห์สัญญาณทางตรรก แผ่นที่ 3

แสดงถึงวงจรการเปรียบเทียบสำหรับ เวิร์ดทริกสำหรับข้อมูล CH9~CH16 (ข้อมูลกลุ่มที่ 2)

วงจรของเครื่องวิเคราะห์สัญญาณทางตรรก แผ่นที่ 4

แสดงถึงวงจรการเปรียบเทียบสำหรับ เวิร์ดทริกสำหรับข้อมูล CH17~CH24 (ข้อมูลกลุ่มที่ 3)

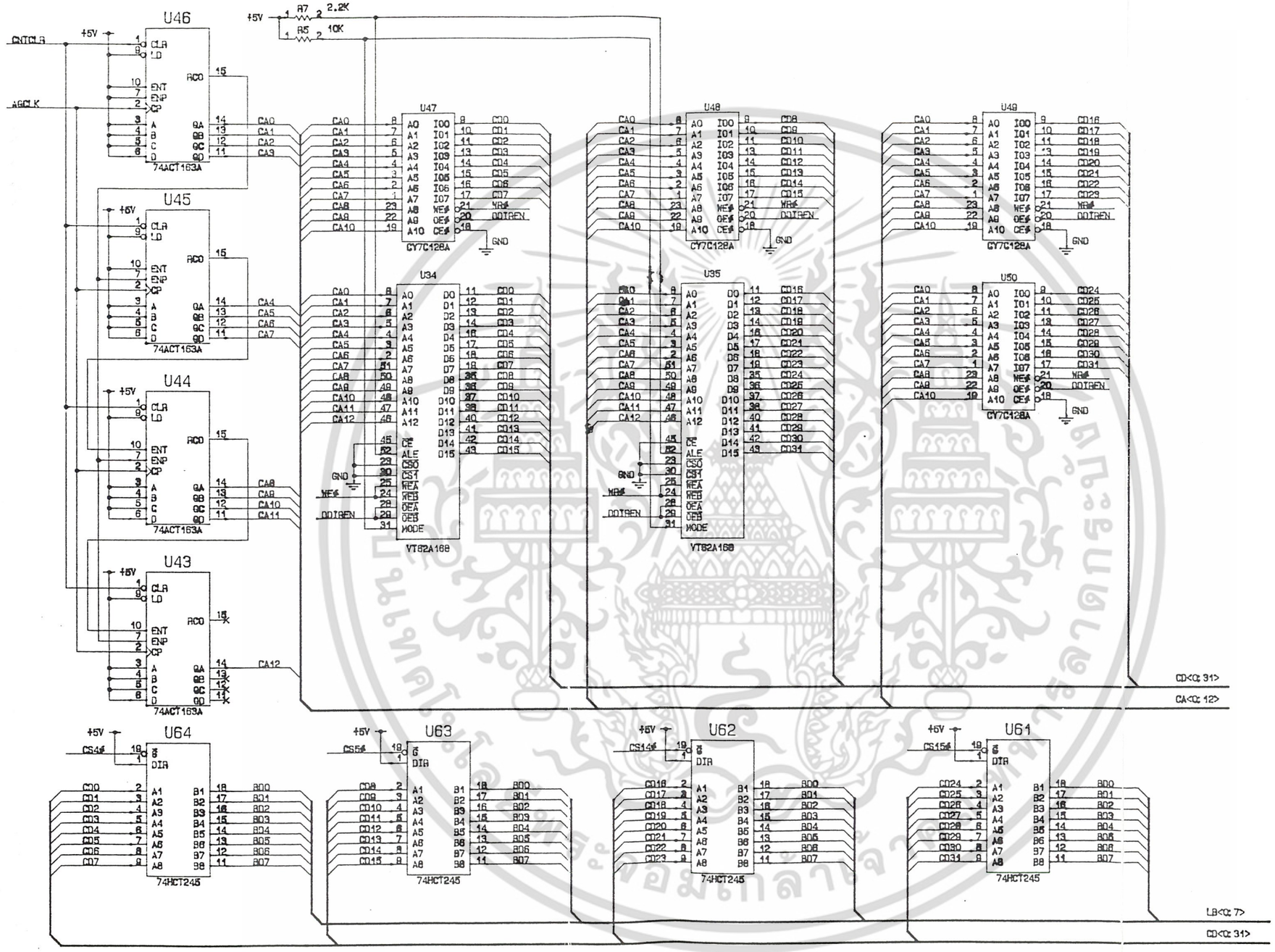


วงจรของเครื่องวิเคราะห์สัญญาณทางตรรก แผ่นที่ 5

แสดงถึงวงจรการเปรียบเทียบสำหรับ เวิร์ดทริกสำหรับข้อมูล CH25~CH32 (ข้อมูลกลุ่มที่ 4)

วงจรของเครื่องวิเคราะห์สัญญาณทางตรรก แผ่นที่ 6

แสดงถึงวงจรตรวจสอบสัญญาณเวิร์ดทริก และ วงจรคาบของเวลา



วงจรของเครื่องวิเคราะห์สัญญาณทางตรรก แผ่นที่ 7

แสดงถึงวงจรของส่วนความจำแบบแคชแรม

วงจรของเครื่องวิเคราะห์สัญญาณทางตรรก แผ่นที่ 8

แสดงถึงวงจรการนับแอดเดรส และ เซตหน่วยความจำแบบแคชแรม

บทที่ 4

สรุปผลและวิจารณ์

4.1 สรุป

ผลที่ได้จากการทดลองใช้งานที่ได้นั้น เครื่องวิเคราะห์สัญญาณทางตรรกชุดนี้สามารถตรวจรับข้อมูลได้ถูกต้องตามทฤษฎีที่ออกแบบไว้ดังที่กล่าวผ่านมาแล้วในบทที่ 3 ซึ่งผลจากการการทดลองเป็นไปดังนี้คือ

1. สามารถตรวจจับสัญญาณได้ถึง 32 ช่องสัญญาณ แต่ละช่องสัญญาณแยกจากกันโดยเด็ดขาด

2. อัตราการสุ่มสัญญาณสามารถจัดตั้งได้ เมื่อใช้สัญญาณนาฬิกาภายใน ซึ่งอัตราสูงสุดที่กำหนดไว้คือ 50 เมกกะเฮิทซ์ และสามารถลดทอนลงได้ด้วยอัตรา 5-2-1 เช่น 50 , 20 , 10 , 5 , 2 , 1 เมกกะเฮิทซ์เป็นต้น

3. การกำหนดข้อมูลเริ่มต้นการทำงาน (WORD TRIGGERING) สามารถกำหนดได้ทั้ง 32 ช่องสัญญาณ โดยสามารถกำหนดสถานะทางตรรกเป็นได้ทั้งตรรก 1 , 0 หรือไม่กำหนด (DON'T CARE) ก็ได้ ซึ่งทั้งนี้ขึ้นอยู่กับความต้องการของผู้ใช้งาน

4. การควบคุมการทำงานและการแสดงผล กระทำผ่านทางเครื่องคอมพิวเตอร์ส่วนบุคคล โดยผ่านทางพอร์ทอนุกรม RS-232

ซึ่งจากผลการทดลองพบว่าผลที่ได้น่าพอใจพอสมควรเมื่อเทียบกับเครื่องชนิดเดียวกันของบริษัทอื่นๆที่สร้างขึ้นมา

4.2 ข้อเสนอแนะ

ดังที่ทราบกันดีอยู่แล้วว่า ในสภาวะปัจจุบันเทคโนโลยีทางด้านไมโครคอมพิวเตอร์ได้ก้าวหน้าไปอย่างรวดเร็ว เพราะฉะนั้นอุปกรณ์ที่ใช้ในการตรวจสอบ และพัฒนาระบบไมโครคอมพิวเตอร์ชุดนี้ ผู้ที่สนใจควรจะพัฒนาอุปกรณ์ชิ้นนี้เพิ่มเติม เพื่อให้เครื่องมีประสิทธิภาพสูงขึ้นดังนี้

1. โดยอาศัยคุณสมบัติของไมโครโปรเซสเซอร์เบอร์ HD64180 ซึ่งมีข้อได้เปรียบไมโครโปรเซสเซอร์เบอร์อื่นได้แก่

1.1 สามารถทำงานด้วยสัญญาณนาฬิกาสูงถึง 12.288เมกกะเฮิทซ์ ทำให้การจัดการข้อมูลเป็นไปได้อย่างรวดเร็ว เพื่อให้สอดคล้องกับการพัฒนาด้านไมโครคอมพิวเตอร์ในสภาวะปัจจุบัน

1.2 สามารถอ้างอิงหน่วยความจำได้สูงถึง 1 เมกกะไบท์ และมีส่วนควบคุมหน่วยความจำโดยตรงทำให้ การใช้งานเป็นไปอย่างมีประสิทธิภาพ

1.3 มีพอร์ทอนุกรม 2 พอร์ทภายในตัว ทำให้ใช้งานได้พอดี กล่าวคือพอร์ทหนึ่งใช้เชื่อมต่อกับเครื่องคอมพิวเตอร์ส่วนบุคคล อีกพอร์ทหนึ่งใช้สำหรับทำการวิเคราะห์สัญญาณแบบอนุกรม

1.4 มีส่วนควบคุมการเข้าถึงหน่วยความจำโดยตรง (DMA) จากคุณสมบัติดังกล่าวข้างต้น และจากการออกแบบไว้เมื่อต้องการเก็บข้อมูลให้ตัวคอมพิวเตอร์ส่วนบุคคลทำการเก็บข้อมูลไว้ ถ้าออกแบบเพิ่มเติมให้เครื่องวิเคราะห์สัญญาณทางตรรก สามารถเชื่อมต่อกับตัวขับจานแม่เหล็กได้โดยตรงแล้ว จะทำให้การเก็บข้อมูลที่ได้จากการตรวจจับมีความรวดเร็วขึ้นและมีประสิทธิภาพสูงขึ้น การเชื่อมต่อตัวขับจานแม่เหล็กก็กระทำได้ไม่ยากนัก ทั้งนี้เนื่องจากมีส่วนกระทำ DMA อยู่ภายในชิพนี้อยู่แล้ว

2. เมื่อใช้เครื่องวิเคราะห์สัญญาณทางตรรก ทำการจับสัญญาณที่ได้จากบัสข้อมูลของไมโครดปรเซสเซอร์เบอร์อื่นๆ ควรจะมีโปรแกรมสนับสนุนช่วยเปลี่ยน การแสดงผลจากรูปสภาวะทางตรรกให้เป็นภาษาแอสแซมบลี้ ของไมโครโปรเซสเซอร์เบอร์นั้นๆด้วย

ในกรณีนี้ จะเห็นว่าหน่วยความจำของไมโครโปรเซสเซอร์ HD64180 มีเหลือเฟือที่จะบรรจุโปรแกรมเหล่านี้ลงไปได้ โดยผ่านทางจานแม่เหล็กที่เชื่อมต่ออยู่ หรือบันทึกลงไปในอีพรอมก็ได้

3. อัตราการสุ่มสัญญาณควรจะเพิ่มขึ้นให้ได้ถึง 100 เมกกะเฮิทซ์ เพื่อรองรับการนำไปใช้วิเคราะห์สัญญาณของบอร์ดใหม่ๆของคอมพิวเตอร์ส่วนบุคคล ซึ่งปัจจุบันใช้ไมโครโปรเซสเซอร์เบอร์ 80486 DXZ ซึ่งทำงานด้วยความเร็วสูงถึง 50 เมกกะเฮิทซ์แล้ว

4. ในช่วงที่เครื่องวิเคราะห์สัญญาณทางตรรก ทำการที่จะดึงข้อมูลที่ตรวจจับได้มาแสดงบนจอภาพนั้น จากซอฟแวร์ที่เขียนขึ้นมานั้น ต้องรอให้อ่านข้อมูลทั้ง 8 กิโลไบท์ทั้งหมดก่อน แล้วจึงจะแสดงผลทางจอภาพ ซึ่งใช้เวลาในการทำงานนานพอสมควรจากการใช้งานของเครื่องตามความเป็นจริงแล้ว ผู้ใช้งานไม่จำเป็นต้องการตรวจสอบข้อมูลทั้งหมด 8 กิโลไบท์ก็ได้ น่าจะมีการปรับปรุงซอฟแวร์ให้สามารถเลือกขนาดของข้อมูลที่ต้องการตรวจสอบได้ ทำให้ประหยัดเวลาในการทำงานของเครื่องได้อีกมาก

จากที่กล่าวมาแล้วข้างต้น จะเห็นว่าการเลือกใช้ไมโครโปรเซสเซอร์เบอร์ HD64180 นั้น เป็นการเปิดให้มีการพัฒนาต่อไปในอนาคตได้ด้วย รวมทั้งชุดคำสั่งต่างๆจะเหมือนกับ Z-80 ผู้พัฒนาสามารถหาอุปกรณ์สนับสนุนได้ง่ายกว่าใช้ไมโครโปรเซสเซอร์เบอร์อื่นๆ เช่น โปรแกรมแอสแซมเบอร์ คู่มือสนับสนุนต่างๆเป็นต้น

เอกสารอ้างอิง

(1) LEWIS C. EGGEBERT , INTERFACING TO THE IBM PERSONAL COMPUTER. HOWARD W. SAME & CO., INC , 1983.

(2) HITACHI , HITACHI 8-BIT MICROPROCESSOR HD64180. HARDWARE MANNAL.

(3) LANCEA LEVENTHAL , INTRODUCTION TO MICROPROCESSORS. SOFTWARE , HARDWARE , PROGRAMMING NEWJERSEY : PRENTICE-HALL , 1978.

(4) HITACHI , HITACHI 8-BIT MICROPROCESSOR HD64180RI , HD64180Z APPLICATION NOTE.

ภาคผนวก 1

ชุดคำสั่งของไมโครคอมพิวเตอร์

แบบชิพเดียว HD64180

A. Instruction Set

The following explains the symbols in instruction set.

1. Register

g, g', ww, xx, yy, and zz specify a register to be used. g and g' specify an 8-bit register. ww, xx, yy, and zz specify a pair of 16-bit registers. The following tables show the correspondence between symbols and registers.

g,g'	Reg.
000	B
001	C
010	D
011	E
100	H
101	L
111	A

ww	Reg.
00	BC
01	DE
10	HL
11	SP

xx	Reg.
00	BC
01	DE
10	IX
11	SP

yy	Reg.
00	BC
01	DE
10	IY
11	SP

zz	Reg.
00	BC
01	DE
10	HL
11	AF

NOTE: Suffixed H and L to ww,xx,yy,zz (ex.wwH,IXL) indicate upper and lower 8-bit of the 16-bit register respectively.

2. Bit

b specifies a bit to be manipulated in the bit manipulation instruction. The following table shows the correspondence between b and bits.

b	Bit
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

3. Condition

f specifies the condition in program control instructions. The following shows the correspondence between f and conditions.

f	Condition
000	NZ non zero
001	Z zero
010	NC non carry
011	C carry
100	PO parity odd
101	PE parity even
110	P sign plus
111	M sign minus

4. Restart Address

v specifies a restart address. The following table shows the correspondence between v and restart addresses.

v	Address
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

5. Flag

The following symbols show the flag conditions.

- · : not affected
- ↑ : affected
- × : undefined
- S : set to 1
- R : reset to 0
- P : parity
- V : overflow

6. Miscellaneous

$(\quad)_M$	:	data in the memory address
$(\quad)_I$	:	data in the I/O address
m or n	:	8-bit data
mn	:	16-bit data
r	:	8-bit register
R	:	16-bit register
$b\cdot(\quad)_M$	:	a content of bit b in the memory address
b·gr	:	a content of bit b in the register gr
d or j	:	8-bit signed displacement
S	:	source addressing mode
D	:	destination addressing mode
·	:	AND operation
+	:	OR operation
$\oplus$	:	EXCLUSIVE OR operation
**	:	added new instructions to Z80

1 . Data Manipulation Instructions

(1) Arithmetic and Logical Instructions (8-bit)

Operation name	MNEMONICS	OP code	Addressing							Bytes	States	Operation	Flag					
			IMMED	EXT	IND	REG	REGI	IMP	REL				7 S	6 Z	4 H	2 P/V	1 N	0 C
ADD	ADD A,g	10 000 g				S		D		1	4	Ar+gr→Ar	↕	↕	↕	V	R	↕
	ADD A, (HL)	10 000 110					S	D		1	6	Ar+(HL)$_M$→Ar	↕	↕	↕	V	R	↕
	ADD A,m	11 000 110	S					D		2	6	Ar+m→Ar	↕	↕	↕	V	R	↕
		‹ m ›																
	ADD A, (IX+d)	11 011 101			S			D		3	14	Ar+(IX+d)$_M$→Ar	↕	↕	↕	V	R	↕
		10 000 110																
		‹ d ›																
	ADD A, (IY+d)	11 111 101			S			D		3	14	Ar+(IY+d)$_M$→Ar	↕	↕	↕	V	R	↕
		10 000 110																
		‹ d ›																
ADC	ADC A,g	10 001 g				S		D		1	4	Ar+gr+c→Ar	↕	↕	↕	V	R	↕
	ADC A, (HL)	10 001 110					S	D		1	6	Ar+(HL)$_M$+c→Ar	↕	↕	↕	V	R	↕
	ADC A,m	11 001 110	S					D		2	6	Ar+m+c→Ar	↕	↕	↕	V	R	↕
		‹ m ›																
	ADC A, (IX+d)	11 011 101			S			D		3	14	Ar+(IX+d)$_M$+c→Ar	↕	↕	↕	V	R	↕
		10 001 110																
		‹ d ›																
	ADC A, (IY+d)	11 111 101			S			D		3	14	Ar+(IY+d)$_M$+c→Ar	↕	↕	↕	V	R	↕
		10 001 110																
		‹ d ›																
AND	AND g	10 100 g				S		D		1	4	Ar·gr→Ar	↕	↕	S	P	R	R
	AND (HL)	10 100 110					S	D		1	6	Ar·(HL)$_M$→Ar	↕	↕	S	P	R	R
	AND m	11 100 110	S					D		2	6	Ar·m→Ar	↕	↕	S	P	R	R
		‹ m ›																
	AND (IX+d)	11 011 101			S			D		3	14	Ar·(IX+d)$_M$→Ar	↕	↕	S	P	R	R
		10 100 110																
		‹ d ›																
	AND (IY+d)	11 111 101			S			D		3	14	Ar·(IY+d)$_M$→Ar	↕	↕	S	P	R	R
		10 100 110																
		‹ d ›																
Compare	CP g	10 111 g				S		D		1	4	Ar−gr	↕	↕	↕	V	S	↕
	CP (HL)	10 111 110					S	D		1	6	Ar−(HL)$_M$	↕	↕	↕	V	S	↕
	CP m	11 111 110	S					D		2	6	Ar−m	↕	↕	↕	V	S	↕
		‹ m ›																
	CP (IX+d)	11 011 101			S			D		3	14	Ar−(IX+d)$_M$	↕	↕	↕	V	S	↕
		10 111 110																
		‹ d ›																
	CP (IY+d)	11 111 101			S			D		3	14	Ar−(IY+d)$_M$	↕	↕	↕	V	S	↕
		10 111 110																
		‹ d ›																
COMPLEMENT	CPL	00 101 111						S/D		1	3	$\overline{Ar}$→Ar	·	·	S	·	S	·
DEC	DEC g	00 g 101				S/D				1	4	gr−1→gr	↕	↕	↕	V	S	·
	DEC (HL)	00 110 101					S/D			1	10	(HL)$_M$−1→(HL)$_M$	↕	↕	↕	V	S	·
	DEC (IX+d)	11 011 101			S/D					3	18	(IX+d)$_M$−1→	↕	↕	↕	V	S	·
		00 110 101										(IX+d)$_M$						
		‹ d ›																
	DEC (IY+d)	11 111 101			S/D					3	18	(IY+d)$_M$−1→	↕	↕	↕	V	S	·
		00 110 101										(IY+d)$_M$						
		‹ d ›																
INC	INC g	00 g 100				S/D				1	4	gr+1→gr	↕	↕	↕	V	R	·
	INC (HL)	00 110 100					S/D			1	10	(HL)$_M$+1→(HL)$_M$	↕	↕	↕	V	R	·
	INC (IX+d)	11 011 101			S/D					3	18	(IX+d)$_M$+1→	↕	↕	↕	V	R	·
		00 110 100										(IX+d)$_M$						
		‹ d ›																
	INC (IY+d)	11 111 101			S/D					3	18	(IY+d)$_M$+1→	↕	↕	↕	V	R	·
		00 110 100										(IY+d)$_M$						
		‹ d ›																

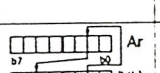
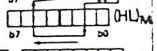
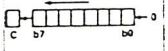
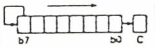
(to be continued)

Operation name	MNEMONICS	OP code	Addressing							Bytes	States	Operation	Flag					
			IMMED	EXT	IND	REG	REGI	IMP	REL				7	6	4	2	1	0
													S	Z	H	P/V	N	C
MULT	MLT ww **	11 101 101 01 ww1 100				S/D				2	17	wwHr × wwLr→ww,	·	·	·	·	·	·
NEGATE	NEG	11 101 101 01 000 100						S/D		2	6	0−Ar→Ar	I	I	I	V	S	I
OR	OR g	10 110 g				S		D		1	4	Ar+gr→Ar	I	I	R	P	R	R
	OR (HL)	10 110 110					S	D		1	6	Ar+(HL)$_M$→Ar	I	I	R	P	R	R
	OR m	11 110 110 〈 m 〉	S					D		2	6	Ar+m→Ar	I	I	R	P	R	R
	OR (IX+d)	11 011 101 10 110 110 〈 d 〉			S			D		3	14	Ar+(IX+d)$_M$→Ar	I	I	R	P	R	R
	OR (IY+d)	11 111 101 10 110 110 〈 d 〉			S			D		3	14	Ar+(IY+d)$_M$→Ar	I	I	R	P	R	R
SUB	SUB g	10 010 g				S		D		1	4	Ar−gr→Ar	I	I	I	V	S	I
	SUB (HL)	10 010 110					S	D		1	6	Ar−(HL)$_M$→Ar	I	I	I	V	S	I
	SUB m	11 010 110 〈 m 〉	S					D		2	6	Ar−m→Ar	I	I	I	V	S	I
	SUB (IX+d)	11 011 101 10 010 110 〈 d 〉			S			D		3	14	Ar−(IX+d)$_M$→Ar	I	I	I	V	S	I
	SUB (IY+d)	11 111 101 10 010 110 〈 d 〉			S			D		3	14	Ar−(IY+d)$_M$→Ar	I	I	I	V	S	I
SUBC	SBC A,g	10 011 g				S		D		1	4	Ar−gr−c→Ar	I	I	I	V	S	I
	SBC A,(HL)	10 011 110					S	D		1	6	Ar−(HL)$_M$−c→Ar	I	I	I	V	S	I
	SBC A,m	11 011 110 〈 m 〉	S					D		2	6	Ar−m−c→Ar	I	I	I	V	S	I
	SBC A,(IX+d)	11 011 101 10 011 110 〈 d 〉			S			D		3	14	Ar−(IX+d)$_M$−c→Ar	I	I	I	V	S	I
	SBC A,(IY+d)	11 111 101 10 011 110 〈 d 〉			S			D		3	14	Ar−(IY+d)$_M$−c→Ar	I	I	I	V	S	I
TEST	TST g **	11 101 101 00 g 100				S				2	7	Ar·gr	I	I	S	P	R	R
	TST (HL) **	11 101 101 00 110 100					S			2	10	Ar·(HL)$_M$	I	I	S	P	R	R
	TST m **	11 101 101 01 100 100 〈 m 〉	S							3	9	Ar·m	I	I	S	P	R	R
XOR	XOR g	10 101 g				S		D		1	4	Ar⊖gr→Ar	I	I	R	P	R	R
	XOR (HL)	10 101 110					S	D		1	6	Ar⊖(HL)$_M$→Ar	I	I	R	P	R	R
	XOR m	11 101 110 〈 m 〉	S					D		2	6	Ar⊖m→Ar	I	I	R	P	R	R
	XOR (IX+d)	11 011 101 10 101 110 〈 d 〉			S			D		3	14	Ar⊖(IX+d)$_M$→Ar	I	I	R	P	R	R
	XOR (IY+d)	11 111 101 10 101 110 〈 d 〉			S			D		3	14	Ar⊖(IY+d)$_M$→Ar	I	I	R	P	R	R

(2) Rotate and Shift Instructions

Operation name	MNEMONICS	OP code	Addressing							Bytes	States	Operation	Flag					
			IMMED	EXT	IND	REG	REGI	IMP	REL				7 S	6 Z	4 H	2 P/V	1 N	0 C
Rotate and Shift Data	RLA	00 010 111						S/D		1	3		·	·	R	·	R	I
	RL g	11 001 011 00 010 g				S/D				2	7		I	I	R	P	R	I
	RL (HL)	11 001 011 00 010 110					S/D			2	13		I	I	R	P	R	I
	RL (IX+d)	11 011 101 11 001 011 ‹ d › 00 010 110			S/D					4	19		I	I	R	P	R	I
	RL (IY+d)	11 111 101 11 001 011 ‹ d › 00 010 110			S/D					4	19		I	I	R	P	R	I
	RLCA	00 000 111						S/D		1	3		·	·	R	·	R	I
	RLC g	11 001 011 00 000 g				S/D				2	7		I	I	R	P	R	I
	RLC (HL)	11 001 011 00 000 110					S/D			2	13		I	I	R	P	R	I
	RLC (IX+d)	11 011 101 11 001 011 ‹ d › 00 000 110			S/D					4	19		I	I	R	P	R	I
	RLC (IY+d)	11 111 101 11 001 011 ‹ d › 00 000 110			S/D					4	19		I	I	R	P	R	I
	RLD	11 101 101 01 101 111						S/D		2	16		I	I	R	P	R	·
	RRA	00 011 111						S/D		1	3		·	·	R	·	R	I
	RR g	11 001 011 00 011 g				S/D				2	7		I	I	R	P	R	I
	RR (HL)	11 001 011 00 011 110					S/D			2	13		I	I	R	P	R	I
	RR (IX+d)	11 011 101 11 001 011 ‹ d › 00 011 110			S/D					4	19		I	I	R	P	R	I
	RR (IY+d)	11 111 101 11 001 011 ‹ d › 00 011 110			S/D					4	19		I	I	R	P	R	I
	RRCA	00 001 111						S/D		1	3		·	·	R	·	R	I
	RRC g	11 001 011 00 001 g				S/D				2	7		I	I	R	P	R	I
	RRC (HL)	11 001 011 00 001 110					S/D			2	13		I	I	R	P	R	I
	RRC (IX+d)	11 011 101 11 001 011 ‹ d › 00 001 110			S/D					4	19		I	I	R	P	R	I
	RRC (IY+d)	11 111 101 11 001 011 ‹ d › 00 001 110			S/D					4	19		I	I	R	P	R	I

(to be continued)

Operation name	MNEMONICS	OP code	Addressing							Bytes	States	Operation	Flag						
			IMMED	EXT	IND	REG	REGI	IMP	REL				7	6	4	2	1	0	
													S	Z	H	P/V	N	C	
Rotate and Shift Data	RRD	11 101 101 01 100 111						S/D		2	16		↕	↕	R	P	R	-
	SLA g	11 001 011 00 100 g				S/D				2	7		↕	↕	R	P	R	↕
	SLA (HL)	11 001 011 00 100 110					S/D			2	13		↕	↕	R	P	R	↕
	SLA (IX+d)	11 011 101 11 001 011 < d > 00 100 110			S/D					4	19		↕	↕	R	P	R	↕
	SLA (IY+d)	11 111 101 11 001 011 < d > 00 100 110			S/D					4	19		↕	↕	R	P	R	↕
	SRA g	11 001 011 00 101 g				S/D				2	7		↕	↕	R	P	R	↕
	SRA (HL)	11 001 011 00 101 110					S/D			2	13		↕	↕	R	P	R	↕
	SRA (IX+d)	11 011 101 11 001 011 < d > 00 101 110			S/D					4	19		↕	↕	R	P	R	↕
	SRA (IY+d)	11 111 101 11 001 011 < d > 00 101 110			S/D					4	19		↕	↕	R	P	R	↕
	SRL g	11 001 011 00 111 g				S/D				2	7		↕	↕	R	P	R	↕
	SRL (HL)	11 001 011 00 111 110					S/D			2	13		↕	↕	R	P	R	↕
	SRL (IX+d)	11 011 101 11 001 011 < d > 00 111 110			S/D					4	19		↕	↕	R	P	R	↕
	SRL (IY+d)	11 111 101 11 001 011 < d > 00 111 110			S/D					4	19		↕	↕	R	P	R	↕

(3) Bit Manipulation Instructions

Operation name	MNEMONICS	OP code	Addressing							Bytes	States	Operation	Flag					
			IMMED	EXT	IND	REG	REGI	IMP	REL				7	6	4	2	1	0
													S	Z	H	P/V	N	C
Bit Set	SET b,g	11 001 011 11 b g				S/D				2	7	1→b·gr	·	·	·	·	·	·
	SET b,(HL)	11 001 011 11 b 110					S/D			2	13	1→b·(HL)$_M$	·	·	·	·	·	·
	SET b,(IX+d)	11 011 101 11 001 011 ‹ d › 11 b 110			S/D					4	19	1→b·(IX+d)$_M$	·	·	·	·	·	·
	SET b,(IY+d)	11 111 101 11 001 011 ‹ d › 11 b 110			S/D					4	19	1→b·(IY+d)$_M$	·	·	·	·	·	·
Bit Reset	RES b,g	11 001 011 10 b g				S/D				2	7	0→b·gr	·	·	·	·	·	·
	RES b,(HL)	11 001 011 10 b 110					S/D			2	13	0→b·(HL)$_M$	·	·	·	·	·	·
	RES b,(IX+d)	11 011 101 11 001 011 ‹ d › 10 b 110			S/D					4	19	0→b·(IX+d)$_M$	·	·	·	·	·	·
	RES b,(IY+d)	11 111 101 11 001 011 ‹ d › 10 b 110			S/D					4	19	0→b·(IY+d)$_M$	·	·	·	·	·	·
Bit Test	BIT b,g	11 001 011 01 b g				S				2	6	$\overline{b \cdot gr} \rightarrow z$	X	↕	S	X	R	·
	BIT b,(HL)	11 001 011 01 b 110					S			2	9	$\overline{b \cdot (HL)_M} \rightarrow z$	X	↕	S	X	R	·
	BIT b,(IX+d)	11 011 101 11 001 011 ‹ d › 01 b 110			S					4	15	$\overline{b \cdot (IX+d)_M} \rightarrow z$	X	↕	S	X	R	·
	BIT b,(IY+d)	11 111 101 11 001 011 ‹ d › 01 b 110			S					4	15	$\overline{b \cdot (IY+d)_M} \rightarrow z$	X	↕	S	X	R	·

(4) Arithmetic Instructions (16-bit)

Operation name	MNEMONICS	OP code	Addressing							Bytes	States	Operation	Flag					
			IMMED	EXT	IND	REG	REGI	IMP	REL				7	6	4	2	1	0
													S	Z	H	P/V	N	C
ADD	ADD HL,ww	00 ww1 001				S		D		1	7	$HL_R+ww_R \rightarrow HL_R$	·	·	X	·	R	↕
	ADD IX,xx	11 011 101 00 xx1 001				S		D		2	10	$IX_R+xx_R \rightarrow IX_R$	·	·	X	·	R	↕
	ADD IY,yy	11 111 101 00 yy1 001				S		D		2	10	$IY_R+yy_R \rightarrow IY_R$	·	·	X	·	R	↕
ADC	ADC HL,ww	11 101 101 01 ww1 010				S		D		2	10	$HL_R+ww_R+c \rightarrow HL_R$	↕	↕	X	V	R	↕
DEC	DEC ww	00 ww1 011				S/D				1	4	$ww_R-1 \rightarrow ww_R$	·	·	·	·	·	·
	DEC IX	11 011 101 00 101 011						S/D		2	7	$IX_R-1 \rightarrow IX_R$	·	·	·	·	·	·
	DEC IY	11 111 101 00 101 011						S/D		2	7	$IY_R-1 \rightarrow IY_R$	·	·	·	·	·	·
INC	INC ww	00 ww0 011				S/D				1	4	$ww_R+1 \rightarrow ww_R$	·	·	·	·	·	·
	INC IX	11 011 101 00 100 011						S/D		2	7	$IX_R+1 \rightarrow IX_R$	·	·	·	·	·	·
	INC IY	11 111 101 00 100 011						S/D		2	7	$IY_R+1 \rightarrow IY_R$	·	·	·	·	·	·
SBC	SBC HL,ww	11 101 101 01 ww0 010				S		D		2	10	$HL_R-ww_R-c \rightarrow HL_R$	↕	↕	X	V	S	↕

2 . Data Transfer Instructions

(1) 8-Bit Load

Operation name	MNEMONICS	OP code	Addressing							Bytes	States	Operation	Flag					
			IMMED	EXT	IND	REG	REGI	IMP	REL				7	6	4	2	1	0
													S	Z	H	P/V	N	C
Load 8-bit Data	LD A,I	11 101 101 01 010 111						S/D		2	6	Ir→Ar	↕	↕	R	IEF	R	·
	LD A,R	11 101 101 01 011 111						S/D		2	6	Rr→Ar	↕	↕	R	IEF	R	·
	LD A,(BC)	00 001 010					S	D		1	6	BC w→Ar ①	·	·	·	·	·	·
	LD A,(DE)	00 011 010					S	D		1	6	DE w→Ar	·	·	·	·	·	·
	LD A,(mn)	00 111 010 ‹ n › ‹ m ›		S				D		3	12	mn w→Ar	·	·	·	·	·	·
	LD I,A	11 101 101 01 000 111						S/D		2	6	Ar→Ir	·	·	·	·	·	·
	LD R,A	11 101 101 01 001 111						S/D		2	6	Ar→Rr	·	·	·	·	·	·
	LD (BC),A	00 000 010					D	S		1	7	Ar→ BC w	·	·	·	·	·	·
	LD (DE),A	00 010 010					D	S		1	7	Ar→ DE w	·	·	·	·	·	·
	LD (mn),A	00 110 010 ‹ n › ‹ m ›		D				S		3	13	Ar→ mn w	·	·	·	·	·	·
	LD g,g'	01 g g'				S	D			1	4	gr'→gr	·	·	·	·	·	·
	LD g,(HL)	01 g 110				D	S			1	6	HL w→gr	·	·	·	·	·	·
	LD g,m	00 g 110 ‹ m ›	S			D				2	6	m→gr	·	·	·	·	·	·
	LD g,(IX+d)	11 011 101 01 g 110 ‹ d ›			S	D				3	14	IX+d w→gr	·	·	·	·	·	·
	LD g,(IY+d)	11 111 101 01 g 110 ‹ d ›			S	D				3	14	IY+d w→gr	·	·	·	·	·	·
	LD (HL),m	00 110 110 ‹ m ›	S				D			2	9	m→ HL w	·	·	·	·	·	·
	LD (IX+d),m	11 011 101 00 110 110 ‹ d › ‹ m ›	S		D					4	15	m→ IX+d w	·	·	·	·	·	·
	LD (IY+d),m	11 111 101 00 110 110 ‹ d › ‹ m ›	S		D					4	15	m→ IY+d w	·	·	·	·	·	·
	LD (HL),g	01 110 g				S	D			1	7	gr→ HL w	·	·	·	·	·	·
	LD (IX+d),g	11 011 101 01 110 g ‹ d ›			D	S				3	15	gr→ IX+d w	·	·	·	·	·	·
	LD (IY+d),g	11 111 101 01 110 g ‹ d ›			D	S				3	15	gr→ IY+d w	·	·	·	·	·	·

① Interrupts are not sampled at the end of LD A, I or LD A, R.

(2) 16-Bit Load

Operation name	MNEMONICS	OP code	Addressing							Bytes	States	Operation	Flag						
			IMMED	EXT	IND	REG	REGI	IMP	REL				7 S	6 Z	4 H	2 P/V	1 N	0 C	
Load 16-bit Data	LD ww,mn	00 ww0 001 〈 n 〉 〈 m 〉	S			D				3	9	mn→ww$_R$	·	·	·	·	·	·	
	LD IX,mn	11 011 101 00 100 001 〈 n 〉 〈 m 〉	S					D		4	12	mn→IX$_R$	·	·	·	·	·	·	
	LD IY,mn	11 111 101 00 100 001 〈 n 〉 〈 m 〉	S					D		4	12	mn→IY$_R$	·	·	·	·	·	·	
	LD SP,HL	11 111 001						S D		1	4	HL$_R$→SP$_R$	·	·	·	·	·	·	
	LD SP,IX	11 011 101 11 111 001						S D		2	7	IX$_R$→SP$_R$	·	·	·	·	·	·	
	LD SP,IY	11 111 101 11 111 001						S D		2	7	IY$_R$→SP$_R$	·	·	·	·	·	·	
	LD ww, (mn)	11 101 101 01 ww1 011 〈 n 〉 〈 m 〉		S		D				4	18	(mn+1)$_W$→wwHr (mn)$_W$→wwLr	·	·	·	·	·	·	
	LD HL, (mn)	00 101 010 〈 n 〉 〈 m 〉		S				D		3	15	(mn+1)$_W$→Hr (mn)$_W$→Lr	·	·	·	·	·	·	
	LD IX, (mn)	11 011 101 00 101 010 〈 n 〉 〈 m 〉		S				D		4	18	(mn+1)$_W$→IXHr (mn)$_W$→IXLr	·	·	·	·	·	·	
	LD IY, (mn)	11 111 101 00 101 010 〈 n 〉 〈 m 〉		S				D		4	18	(mn+1)$_W$→IYHr (mn)$_W$→IYLr	·	·	·	·	·	·	
	LD (mn),ww	11 101 101 01 ww0 011 〈 n 〉 〈 m 〉		D		S				4	19	wwHr→(mn+1)$_W$ wwLr→(mn)$_W$	·	·	·	·	·	·	
	LD (mn),HL	00 100 010 〈 n 〉 〈 m 〉		D				S		3	16	Hr→(mn+1)$_W$ Lr→(mn)$_W$	·	·	·	·	·	·	
	LD (mn),IX	11 011 101 00 100 010 〈 n 〉 〈 m 〉		D				S		4	19	IXHr→(mn+1)$_W$ IXLr→(mn)$_W$	·	·	·	·	·	·	
	LD (mn),IY	11 111 101 00 100 010 〈 n 〉 〈 m 〉		D				S		4	19	IYHr→(mn+1)$_W$ IYLr→(mn)$_W$	·	·	·	·	·	·	

(3) Block Transfer

Operation name	MNEMONICS	OP code	Addressing							Bytes	States	Operation	Flag					
			IMMED	EXT	IND	REG	REGI	IMP	REL				7	6	4	2	1	0
													S	Z	H	P/V	N	C
Block Transfer Search Data	CPD	11 101 101 10 101 001					S	S		2	12	$Ar-(HL)_M$ $BC_R-1\rightarrow BC_R$ $HL_R-1\rightarrow HL_R$	I	③ I	I	② I	S	·
	CPDR	11 101 101 10 111 001					S	S		2	14 12	$BC_R\neq 0$ $Ar\neq(HL)_M$ $BC_R=0$ or $Ar=(HL)_M$ Q { $Ar-(HL)_M$; $BC_R-1\rightarrow BC_R$; $HL_R-1\rightarrow HL_R$ } Repeat Q until $Ar=(HL)_M$ or $BC_R=0$	I	③ I	I	② I	S	·
	CPI	11 101 101 10 100 001					S	S		2	12	$Ar-(HL)_M$ $BC_R-1\rightarrow BC_R$ $HL_R+1\rightarrow HL_R$	I	③ I	I	② I	S	·
	CPIR	11 101 101 10 110 001					S	S		2	14 12	$BC_R\neq 0$ $Ar\neq(HL)_M$ $BC_R=0$ or $Ar=(HL)_M$ Q { $Ar-(HL)_M$; $BC_R-1\rightarrow BC_R$; $HL_R+1\rightarrow HL_R$ } Repeat Q until $Ar=(HL)_M$ or $BC_R=0$	I	③ I	I	② I	S	·
	LDD	11 101 101 10 101 000					S	D		2	12	$(HL)_M\rightarrow(DE)_M$ $BC_R-1\rightarrow BC_R$ $DE_R-1\rightarrow DE_R$ $HL_R-1\rightarrow HL_R$	·	· R	R	② I	R	·
	LDDR	11 101 101 10 111 000					S	D		2	14 ($BC_R\neq 0$) 12 ($BC_R=0$)	Q { $(HL)_M\rightarrow(DE)_M$; $BC_R-1\rightarrow BC_R$; $DE_R-1\rightarrow DE_R$; $HL_R-1\rightarrow HL_R$ } Repeat Q until $BC_R=0$	·	· R	R	② I	R	·
	LDI	11 101 101 10 100 000					S	D		2	12	$(HL)_M\rightarrow(DE)_M$ $BC_R-1\rightarrow BC_R$ $DE_R+1\rightarrow DE_R$ $HL_R+1\rightarrow HL_R$	·	· R	R	② I	R	·
	LDIR	11 101 101 10 110 000					S	D		2	14 ($BC_R\neq 0$) 12 ($BC_R=0$)	Q { $(HL)_M\rightarrow(DE)_M$; $BC_R-1\rightarrow BC_R$; $DE_R+1\rightarrow DE_R$; $HL_R+1\rightarrow HL_R$ } Repeat Q until $BC_R=0$	·	· R	R	② I	R	·

② P/V=0 : $BC_R-1=0$
P/V=1 : $BC_R-1\neq 0$
③ Z=1 : $Ar=(HL)_M$
Z=0 : $Ar\neq(HL)_M$

(4) Stack and Exchange

Operation name	MNEMONICS	OP code	Addressing							Bytes	States	Operation	Flag					
			IMMED	EXT	IND	REG	REGI	IMP	REL				7	6	4	2	1	0
													S	Z	H	P/V	N	C
PUSH	PUSH zz	11 zz0 101				S		D		1	11	$zzLr \rightarrow (SP-2)_M$ $zzHr \rightarrow (SP-1)_M$ $SP_R-2 \rightarrow SP_R$	·	·	·	·	·	·
	PUSH IX	11 011 101 11 100 101						S/D		2	14	$IXLr \rightarrow (SP-2)_M$ $IXHr \rightarrow (SP-1)_M$ $SP_R-2 \rightarrow SP_R$	·	·	·	·	·	·
	PUSH IY	11 111 101 11 100 101						S/D		2	14	$IYLr \rightarrow (SP-2)_M$ $IYHr \rightarrow (SP-1)_M$ $SP_R-2 \rightarrow SP_R$	·	·	·	·	·	·
POP	POP zz	11 zz0 001				D		S		1	9	$(SP+1)_M \rightarrow zzHr$ ④ $(SP)_M \rightarrow zzLr$ $SP_R+2 \rightarrow SP_R$	·	·	·	·	·	·
	POP IX	11 011 101 11 100 001						S/D		2	12	$(SP+1)_M \rightarrow IXHr$ $(SP)_M \rightarrow IXLr$ $SP_R+2 \rightarrow SP_R$	·	·	·	·	·	·
	POP IY	11 111 101 11 100 001						S/D		2	12	$(SP+1)_M \rightarrow IYHr$ $(SP)_M \rightarrow IYLr$ $SP_R+2 \rightarrow SP_R$	·	·	·	·	·	·
Exchange	EX AF,AF'	00 001 000						S/D		1	4	$AF_R \leftrightarrow AF_R'$	·	·	·	·	·	·
	EX DE,HL	11 101 011						S/D		1	3	$DE_R \leftrightarrow HL_R$	·	·	·	·	·	·
	EXX	11 011 001						S/D		1	3	$BC_R \leftrightarrow BC_R'$ $DE_R \leftrightarrow DE_R'$ $HL_R \leftrightarrow HL_R'$	·	·	·	·	·	·
	EX (SP),HL	11 100 011						S/D		1	16	$Hr \leftrightarrow (SP+1)_M$ $Lr \leftrightarrow (SP)_M$	·	·	·	·	·	·
	EX (SP),IX	11 011 101 11 100 011						S/D		2	19	$IXHr \leftrightarrow (SP+1)_M$ $IXLr \leftrightarrow (SP)_M$	·	·	·	·	·	·
	EX (SP),IY	11 111 101 11 100 011						S/D		2	19	$IYHr \leftrightarrow (SP+1)_M$ $IYLr \leftrightarrow (SP)_M$	·	·	·	·	·	·

④ In the case of POP AF, Flag is written a current contents of the stack.

3 . Program Control Instructions

Operation name	MNEMONICS	OP code	Addressing							Bytes	States	Operation	Flag						
			IMMED	EXT	IND	REG	REGI	IMP	REL				7	6	4	2	1	0
													S	Z	H	P/V	N	C
Call	CALL mn	11 001 101 〈 n 〉 〈 m 〉		D						3	16	(PCHr)←(SP−1)$_M$ (PCLr)←(SP−2)$_M$ mn→PC$_R$ SP$_R$−2→SP$_R$	·	·	·	·	·	·
	CALL f,mn	11 f 100 〈 n 〉 〈 m 〉		D						3	6 f : false 16 f : true	continue : f is false CALL mn : f is true	·	·	·	·	·	·
Jump	DJNZ j	00 010 000 〈j-2							D	2 2	8 Br=0 7 Br≠0	Br−1→Br continue : Br=0 PC$_R$−j→PC$_R$: Br≠0	·	·	·	·	·	·
	JP f,mn	11 f 010 〈 n 〉 〈 m 〉		D						3 3	6 f : false 9 f : true	mn→PC$_R$: f is true continue : f is false	·	·	·	·	·	·
	JP mn	11 000 011 〈 n 〉 〈 m 〉		D						3	9	mn→PC$_R$	·	·	·	·	·	·
	JP (HL)	11 101 001					D			1	3	HL$_R$→PC$_R$	·	·	·	·	·	·
	JP (IX)	11 011 101 11 101 001					D			2	6	IX$_R$→PC$_R$	·	·	·	·	·	·
	JP (IY)	11 111 101 11 101 001					D			2	6	IY$_R$→PC$_R$	·	·	·	·	·	·
	JR j	00 011 000 〈j-2							D	2	8	PC$_R$−j→PC$_R$	·	·	·	·	·	·
	JR C,j	00 111 000 〈j-2							D	2 2	6 8	continue : C=0 PC$_R$−j→PC$_R$: C=1	·	·	·	·	·	·
	JR NC,j	00 110 000 〈j-2							D	2 2	6 8	continue : C=1 PC$_R$−j→PC$_R$: C=0	·	·	·	·	·	·
	JR Z,j	00 101 000 〈j-2							D	2 2	6 8	continue : Z=0 PC$_R$−j→PC$_R$: Z=1	·	·	·	·	·	·
	JR NZ,j	00 100 000 〈j-2							D	2 2	6 8	continue : Z=1 PC$_R$−j→PC$_R$: Z=0	·	·	·	·	·	·
Return	RET	11 001 001						D		1	9	(SP)$_M$→PCLr (SP−1)$_M$→PCHr SP$_R$−2→SP$_R$	·	·	·	·	·	·
	RET f	11 f 000							D	1 1	5 f : false 10 f : true	continue : f is false RET : f is true	·	·	·	·	·	·
	RETI	11 101 101 01 001 101						D		2	12 (Z) 12 (8Z)	(SP)$_M$→PCLr (SP−1)$_M$→PCHr SP$_R$−2→SP$_R$	·	·	·	·	·	·
	RETN	11 101 101 01 000 101						D		2	12	(SP)$_M$→PCLr (SP−1)$_M$→PCHr SP$_R$−2→SP$_R$ IEF$_2$→IEF$_1$	·	·	·	·	·	·
Restart	RST v	11 v 111						D		1	11	(PCHr)←(SP−1)$_M$ (PCLr)←(SP−2)$_M$ 0→PCHr v→PCLr SP$_R$−2→SP$_R$	·	·	·	·	·	·

4 . I/O Instructions

Operation name	MNEMONICS	OP code	Addressing							Bytes	States	Operation	Flag					
			IMMED	EXT	IND	REG	REGI	IMP	I/O				7	6	4	2	1	0
													S	Z	H	P/V	N	C
INPUT	IN A,(m)	11 011 011 〈 m 〉						D	S	2	9	Am→Ar m→A₀~A₇ Ar→A₈~A₁₅	·	·	·	·	·	·
	IN g,(C)	11 101 101 01 g 000				D			S	2	9	BC→gr g=110 : Only the flags will change. Cr→A₀~A₇ Br→A₈~A₁₅	↕	↕	R	P	R	·
	IN0 g,(m) **	11 101 101 00 g 000 〈 m 〉				D			S	3	12	00m→gr g=110 : Only the flags will change. m→A₀~A₇ 00→A₈~A₁₅	↕	↕	R	P	R	·
	IND	11 101 101 10 101 010					D		S	2	12	BC→HL HL₀−1→HL₀ Br−1→Br Cr→A₀~A₇ Br→A₈~A₁₅	X	⑤ ↕	X	X	⑥ I	X
	INDR	11 101 101 10 111 010					D		S	2	14 Br≠0) 12 Br=0)	BC→HL Q HL₀−1→HL₀ Br−1→Br Repeat Q until Br=0 Cr→A₀~A₇ Br→A₈~A₁₅	X	S	X	X	⑥ I	X
	INI	11 101 101 10 100 010					D		S	2	12	BC→HL HL₀−1→HL₀ Br−1→Br Cr→A₀~A₇ Br→A₈~A₁₅	X	⑤ ↕	X	X	⑥ I	X
	INIR	11 101 101 10 110 010					D		S	2	14 Br≠0) 12 Br=0)	BC→HL Q HL₀−1→HL₀ Br−1→Br Repeat Q until Br=0 Cr→A₀~A₇ Br→A₈~A₁₅	X	S	X	X	⑥ I	X

(to be continued)

⑤ Z=1 : Br−1=0
Z=0 : Br−1≠0

⑥ N=1 : MSB of Data=1
N=0 : MSB of Data=0

Operation name	MNEMONICS	OP code	Addressing							Bytes	States	Operation	Flag					
			IMMED	EXT	IND	REG	REGI	IMP	I/O				7 S	6 Z	4 H	2 P/V	1 N	0 C
OUTPUT	OUT (m),A	11 010 011 〈 m 〉						S	D	2	10	$Ar \rightarrow (Am)_1$ $m \rightarrow A_0 \sim A_7$ $Ar \rightarrow A_8 \sim A_{15}$	·	·	·	·	·	·
	OUT (C),g	11 101 101 01 g 001				S			D	2	10	$gr \rightarrow (BC)_1$ $Cr \rightarrow A_0 \sim A_7$ $Br \rightarrow A_8 \sim A_{15}$	·	·	·	·	·	·
	OUT0 (m),g **	11 101 101 00 g 001 〈 m 〉				S			D	3	13	$gr \rightarrow (00m)_1$ $m \rightarrow A_0 \sim A_7$ $00 \rightarrow A_8 \sim A_{15}$		⑤			⑥	
	OTDM **	11 101 101 10 001 011					S		D	2	14	$(HL)_M \rightarrow (00C)_1$ $HL_0 - 1 \rightarrow HL_0$ $Cr - 1 \rightarrow Cr$ $Br - 1 \rightarrow Br$ $Cr \rightarrow A_0 \sim A_7$ $00 \rightarrow A_8 \sim A_{15}$	I	I	I	P	I ⑥	I
	OTDMR **	11 101 101 10 011 011					S		D	2	16 (Br=0) 14 (Br=0)	Q { $(HL)_M \rightarrow (00C)_1$ $HL_0 - 1 \rightarrow HL_0$ $Cr - 1 \rightarrow Cr$ $Br - 1 \rightarrow Br$ Repeat Q until $Br = 0$ $Cr \rightarrow A_0 \sim A_7$ $00 \rightarrow A_8 \sim A_{15}$	R	S	R	S	I ⑥	R
	OTDR	11 101 101 10 111 011					S		D	2	14 (Br=0) 12 (Br=0)	Q { $(HL)_M \rightarrow (BC)_1$ $HL_0 - 1 \rightarrow HL_0$ $Br - 1 \rightarrow Br$ Repeat Q until $Br = 0$ $Cr \rightarrow A_0 \sim A_7$ $Br \rightarrow A_8 \sim A_{15}$	X	S ⑤	X	X	I ⑥	X
	OUTI	11 101 101 10 100 011					S		D	2	12	$(HL)_M \rightarrow (BC)_1$ $HL_0 + 1 \rightarrow HL_0$ $Br - 1 \rightarrow Br$ $Cr \rightarrow A_0 \sim A_7$ $Br \rightarrow A_8 \sim A_{15}$	X	I	X	X	I ⑥	X
	OTIR	11 101 101 10 110 011					S		D	2	14 (Br=0) 12 (Br=0)	Q { $(HL)_M \rightarrow (BC)_1$ $HL_0 + 1 \rightarrow HL_0$ $Br - 1 \rightarrow Br$ Repeat Q until $Br = 0$ $Cr \rightarrow A_0 \sim A_7$ $Br \rightarrow A_8 \sim A_{15}$	X	S	X	X	I ⑥	X
	TSTIO m **	11 101 101 01 110 100 〈 m 〉	S						S	3	12	$(0C)_1 \cdot m$ $Cr \rightarrow A_0 \sim A_7$ $00 \rightarrow A_8 \sim A_{15}$	I	I ⑤	S	P	R ⑥	R
	OTIM **	11 101 101 10 000 011					S		D	2	14	$(HL)_M \rightarrow (00C)_1$ $HL_0 - 1 \rightarrow HL_0$ $Cr - 1 \rightarrow Cr$ $Br - 1 \rightarrow Br$ $Cr \rightarrow A_0 \sim A_7$ $00 \rightarrow A_8 \sim A_{15}$	I	I	I	P	I ⑥	I
	OTIMR **	11 101 101 10 010 011					S		D	2	16 (Br=0) 14 (Br=0)	Q { $(HL)_M \rightarrow (00C)_1$ $HL_0 + 1 \rightarrow HL_0$ $Cr + 1 \rightarrow Cr$ $Br - 1 \rightarrow Br$ Repeat Q until $Br = 0$ $Cr \rightarrow A_0 \sim A_7$ $00 \rightarrow A_8 \sim A_{15}$	R	S ⑤	R	S	I ⑥	R
	OUTD	11 101 101 10 101 011					S		D	2	12	$(HL)_M \rightarrow (BC)_1$ $HL_0 - 1 \rightarrow HL_0$ $Br - 1 \rightarrow Br$ $Cr \rightarrow A_0 \sim A_7$ $Br \rightarrow A_8 \sim A_{15}$	X	I	X	X	I	X

⑤ Z=1 : Br−1=0
Z=0 : Br−1≠0

⑥ N=1 : MSB of Data=1
N=0 : MSB of Data=0

5 . Special Control Instructions

Operation name	MNEMONICS	OP code	Addressing							Bytes	States	Operation	Flag					
			IMMED	EXT	IND	REG	REGI	IMP	REL				7	6	4	2	1	0
													S	Z	H	P/V	N	C
Special Function	DAA	00 100 111						S/D		1	4	Decimal Adjust Accumulator	I	I	I	P	·	I
Carry Control	CCF	00 111 111								1	3	$\overline{C}\rightarrow C$	·	·	R	·	R	I
	SCF	00 110 111								1	3	$1\rightarrow C$	·	·	R	·	R	S
CPU Control	DI	11 110 011								1	3	$0\rightarrow IEF_1$, $0\rightarrow IEF_2$ ⑦	·	·	·	·	·	·
	EI	11 111 011								1	3	$1\rightarrow IEF_1$, $1\rightarrow IEF_2$ ⑦	·	·	·	·	·	·
	HALT	01 110 110								1	3	CPU halted	·	·	·	·	·	·
	IM 0	11 101 101 01 000 110								2	6	Interrupt mode 0	·	·	·	·	·	·
	IM 1	11 101 101 01 010 110								2	6	Interrupt mode 1	·	·	·	·	·	·
	IM 2	11 101 101 01 011 110								2	6	Interrupt mode 2	·	·	·	·	·	·
	NOP	00 000 000								1	3	No operation	·	·	·	·	·	·
	SLP **	11 101 101 01 110 110								2	8	Sleep	·	·	·	·	·	·

⑦ Interrupts are not sampled at the end of DI or EI.

B. Instruction Summary in Alphabetical Order

** : Added new instructions to Z80

MNEMONICS	Bytes	Machine Cycles	States
ADC A,m	2	2	6
ADC A,g	1	2	4
ADC A, (HL)	1	2	6
ADC A, (IX+d)	3	6	14
ADC A, (IY+d)	3	6	14
ADD A,m	2	2	6
ADD A,g	1	2	4
ADD A, (HL)	1	2	6
ADD A, (IX+d)	3	6	14
ADD A, (IY+d)	3	6	14
ADC HL,ww	2	6	10
ADD HL,ww	1	5	7
ADD IX,xx	2	6	10
ADD IY,yy	2	6	10
AND m	2	2	6
AND g	1	2	4
AND (HL)	1	2	6
AND (IX+d)	3	6	14
AND (IY+d)	3	6	14
BIT b, (HL)	2	3	9
BIT b, (IX+d)	4	5	15
BIT b, (IY+d)	4	5	15
BIT b,g	2	2	6
CALL f,mn	3	2	6 (If condition is false)
	3	6	16 (If condition is true)

(to be continued)

MNEMONICS	Bytes	Machine Cycles	States
CALL mn	3	6	16
CCF	1	1	3
CPD	2	6	12
CPDR	2	8	14 (If $BC_R \neq 0$ and $Ar \neq (HL)_M$)
	2	6	12 (If $BC_R = 0$ or $Ar = (HL)_M$)
CP (HL)	1	2	6
CPI	2	6	12
CPIR	2	8	14 (If $BC_R \neq 0$ and $Ar \neq (HL)_M$)
	2	6	12 (If $BC_R = 0$ or $Ar = (HL)_M$)
CP (IX+d)	3	6	14
CP (IY+d)	3	6	14
CPL	1	1	3
CP m	2	2	6
CP g	1	2	4
DAA	1	2	4
DEC (HL)	1	4	10
DEC IX	2	3	7
DEC IY	2	3	7
DEC (IX+d)	3	8	18
DEC (IY+d)	3	8	18
DEC g	1	2	4
DEC ww	1	2	4
DI	1	1	3

(to be continued)

MNEMONICS	Bytes	Machine Cycles	States
DJNZ j	2	5	9 (If Br≠0)
	2	3	7 (If Br=0)
EI	1	1	3
EX AF,AF'	1	2	4
EX DE,HL	1	1	3
EX (SP),HL	1	6	16
EX (SP),IX	2	7	19
EX (SP),IY	2	7	19
EXX	1	1	3
HALT	1	1	3
IM 0	2	2	6
IM 1	2	2	6
IM 2	2	2	6
INC g	1	2	4
INC (HL)	1	4	10
INC (IX+d)	3	8	18
INC (IY+d)	3	8	18
INC ww	1	2	4
INC IX	2	3	7
INC IY	2	3	7
IN A,(m)	2	3	9
IN g,(C)	2	3	9
INI	2	4	12
INIR	2	6	14 (If Br≠0)
	2	4	12 (If Br=0)
IND	2	4	12
INDR	2	6	14 (If Br≠0)

(to be continued)

MNEMONICS	Bytes	Machine Cycles	States
INDR	2	4	12 (If Br=0)
IN0 g,(m)**	3	4	12
JP f,mn	3	2	6 (If f is false)
	3	3	9 (If f is true)
JP (HL)	1	1	3
JP (IX)	2	2	6
JP (IY)	2	2	6
JP mn	3	3	9
JR j	2	4	8
JR C,j	2	2	6 (If condition is false)
	2	4	8 (If condition is true)
JR NC,j	2	2	6 (If condition is false)
	2	4	8 (If condition is true)
JR Z,j	2	2	6 (If condition is false)
	2	4	8 (If condition is true)
JR NZ,j	2	2	6 (If condition is false)
	2	4	8 (If condition is true)

(to be continued)

MNEMONICS	Bytes	Machine Cycles	States
LD A, (BC)	1	2	6
LD A, (DE)	1	2	6
LD A,I	2	2	6
LD A, (mn)	3	4	12
LD A,R	2	2	6
LD (BC),A	1	3	7
LDD	2	4	12
LD (DE),A	1	3	7
LD ww,mn	3	3	9
LD ww,(mn)	4	6	18
LDDR	2	6	14 (If $BC_R \neq 0$)
	2	4	12 (If $BC_R = 0$)
LD (HL),m	2	3	9
LD HL,(mn)	3	5	15
LD (HL),g	1	3	7
LDI	2	4	12
LD I,A	2	2	6
LDIR	2	6	14 (If $BC_R \neq 0$)
	2	4	12 (If $BC_R = 0$)
LD IX,mn	4	4	12
LD IX,(mn)	4	6	18
LD (IX+d),m	4	5	15
LD (IX+d),g	3	7	15
LD IY,mn	4	4	12
LD IY,(mn)	4	6	18
LD (IY+d),m	4	5	15
LD (IY+d),g	3	7	15

(to be continued)

MNEMONICS	Bytes	Machine Cycles	States
LD (mn),A	3	5	13
LD (mn),ww	4	7	19
LD (mn),HL	3	6	16
LD (mn),IX	4	7	19
LD (mn),IY	4	7	19
LD R,A	2	2	6
LD g,(HL)	1	2	6
LD g,(IX+d)	3	6	14
LD g,(IY+d)	3	6	14
LD g,m	2	2	6
LD g,g'	1	2	4
LD SP,HL	1	2	4
LD SP,IX	2	3	7
LD SP,IY	2	3	7
MLT ww**	2	13	17
NEG	2	2	6
NOP	1	1	3
OR (HL)	1	2	6
OR (IX+d)	3	6	14
OR (IY+d)	3	6	14
OR m	2	2	6
OR g	1	2	4
OTDM**	2	6	14
OTDMR**	2	8	16 (If Br≠0)
	2	6	14 (If Br=0)
OTDR	2	6	14 (If Br≠0)
	2	4	12 (If Br=0)

(to be continued)

MNEMONICS	Bytes	Machine Cycles	States
OTIM**	2	6	14
OTIMR**	2	8	16 (If Br≠0)
	2	6	14 (If Br=0)
OTIR	2	6	14 (If Br≠0)
	2	4	12 (If Br=0)
OUTD	2	4	12
OUTI	2	4	12
OUT (m),A	2	4	10
OUT (C),g	2	4	10
OUT0 (m),g**	3	5	13
POP IX	2	4	12
POP IY	2	4	12
POP zz	1	3	9
PUSH IX	2	6	14
PUSH IY	2	6	14
PUSH zz	1	5	11
RES b,(HL)	2	5	13
RES b,(IX+d)	4	7	19
RES b,(IY+d)	4	7	19
RES b,g	2	3	7
RET	1	3	9
RET f	1	3	5
			(If condition is false)
	1	4	10
			(If condition is true)
RETI	2	10 (Z)	22 (Z)
		4 (R1)	12 (R1)
RETN	2	4	12

(to be continued)

MNEMONICS	Bytes	Machine Cycles	States
RLA	1	1	3
RLCA	1	1	3
RLC (HL)	2	5	13
RLC (IX+d)	4	7	19
RLC (IY+d)	4	7	19
RLC g	2	3	7
RLD	2	8	16
RL (HL)	2	5	13
RL (IX+d)	4	7	19
RL (IY+d)	4	7	19
RL g	2	3	7
RRA	1	1	3
RRCA	1	1	3
RRC (HL)	2	5	13
RRC (IX+d)	4	7	19
RRC (IY+d)	4	7	19
RRC g	2	3	7
RRD	2	8	16
RR (HL)	2	5	13
RR (IX+d)	4	7	19
RR (IY+d)	4	7	19
RR g	2	3	7
RST v	1	5	11
SBC A,(HL)	1	2	6
SBC A,(IX+d)	3	6	14
SBC A,(IY+d)	3	6	14
SBC A,m	2	2	6

(to be continued)

MNEMONICS	Bytes	Machine Cycles	States
SBC A,g	1	2	4
SBC HL,ww	2	6	10
SCF	1	1	3
SET b,(HL)	2	5	13
SET b,(IX+d)	4	7	19
SET b,(IY+d)	4	7	19
SET b,g	2	3	7
SLA (HL)	2	5	13
SLA (IX+d)	4	7	19
SLA (IY+d)	4	7	19
SLA g	2	3	7
SLP**	2	2	8
SRA (HL)	2	5	13
SRA (IX+d)	4	7	19
SRA (IY+d)	4	7	19
SRA g	2	3	7
SRL (HL)	2	5	13
SRL (IX+d)	4	7	19
SRL (IY+d)	4	7	19
SRL g	2	3	7
SUB (HL)	1	2	6
SUB (IX+d)	3	6	14
SUB (IY+d)	3	6	14
SUB m	2	2	6
SUB g	1	2	4
**TSTIO m	3	4	12
**TST g	2	3	7

(to be continued)

MNEMONICS	Bytes	Machine Cycles	States
TST m**	3	3	9
TST (HL)**	2	4	10
XOR (HL)	1	2	6
XOR (IX+d)	3	6	14
XOR (IY+d)	3	6	14
XOR m	2	2	6
XOR g	1	2	4

ภาคผนวก 2

โปรแกรมที่ใช้งาน

เครื่องคอมพิวเตอร์ส่วนบุคคล IBM PC

```
;
;
;**********
;ppi equate
;**********
;
;
;***************
;address equate
;***************
;
;
;******************
;character equates
;******************
;
;
;********************
;start main program
;********************
;
        .org    0000
start:  ld b,0ffh       ;delay for wake up
wake:   djnz wake
        jp start1       ;jump pass interupt table
;
        .org  0008h
bpoint: jp brksrv       ;break point service routine
```

```
        .org  0010h
         jp warm          ;user monitor
;
        .org    100h
start1:  di               ;disablle interupt
         ld b,50h         ;clear all buffer
         ld hl,0fe00h
start2:  ld (hl),00
         inc hl
         djnz start2
;
;initial cpu
;
;set I/O wait state = 4 state
         ld a,30h
         ld b,00h
         ld c,32h
         out (c),a
;set refresh = 10 cycles
         ld c,36h
         out (c),a
         ld a,81h
         out (c),a;
         ld a,09h
         out (c),a
;interrupt vecter
         ld hl,0ff00h
         ld a,h
         ld i,a           ;ld int vect hi
```

```
        ld a,1
        ld b,00h
        ld c,33h        ;ld int vect lo
        out (c),a
;enable interrupt
        inc c
        ld a,07h
        out (c),a
;set cts1* enable
        ld a,04h
        ld c,05h
        out (c),a
;initial 8255
        ld a,89h        ;control word of 8255
        out (ctrl),a    ;all pa,pb are O/P,pc I/P
;
        ld sp,0fdffh    ;set stack
        im 1            ;set interupt mode 1
        ld hl,brate     ;  data send out for sampling rate
uplink: ld b,00h
        ld a,75h        ;protocal 1stt,8 bits,2 stp
        ld c,01h        ;set format of communication
        out (c),a
        ld c,03h
        outi            ;out and inc
        call getchr     ;wait chr from terminal
        call getchr
        cp 20h          ;"space" ?
        jr nz,uplink    ;no cont wait
```

```
        pop af
        ld hl,signon    ;display sign on mssg
        call sndout
warm:   ld sp,0fdffh    ;set stack
        ld a,0ffh
        out (pa),a
        ld a,8dh        ;10001101B
        out (pb),a
rdy:    ld hl,ready     ;display prompt
        call sndout
;
aa:     call getchr     ;wait command from terminal
        call lc2uc      ;convert lowercase to upper case
        ld (chrbuf),a   ;save in chrbuf
        call sndchr     ;echo back
        cp 'D'
        jp z,dump       ;dump memory
        cp 'H'
        jp z,help       ;display help
        cp 'R'
        jp z,rdall      ;read data from cache ram
        cp 'S'
        jp z,samp       ;set sampling rate
        cp 'T'
        jp z,strig      ;set word trig
        cp 'Z'
        jp z,analy
        jp rdy
;
```

```
;
signon:  .db    cr,lf,"                          PC/XT Training Kit",cr,lf
         .db    cr,lf,"                            Copyright 1992  ",cr,lf
         .db    cr,lf,"                          by    Wichit  T.  ",cr,lf
         .db    cr,lf,"                          ******************",cr,lf
         .db    cr,lf,cr,lf,cr,lf
         .db    cr,lf,"                      Communication link set up",cr,lf
         .db    cr,lf,"                       type H for display Help",cr,lf
         .db    cr,lf,cr,lf,cr,lf,cr,lf,cr,lf,cr,lf,04h
ready:   .db    cr,lf,"Ready>",04h

;
;**********************
;start logic analyzer
;**********************
;
analy:  ld hl,analypat
        call sndout
;
        in a,(pb)       ;disable decoder
        and 0f0h
        or  2dh
        out (pb),a
        push af
        ld a,0fh        ;check bout
        out (pa),a
        ld b,07h
        call enable
        call delay1
```

```
        pop af
        or  80h         ;set pb
        and 0fh         ;clr pb4,pb5,pb6
        out (pb),a      ;clear clock,step=0
        call delay1
        or 11010000b
        out (pb),a
        call delay3
        call delay3
        call delay3
        call delay3
        call delay3
        call delay3
        call delay3
        call delay3
        call delay3
        call delay3
        call delay3
        call delay3
        call delay3
        jp  warm

analypat:    .db  cr,lf," Logic Analyzer:",04h
;
;*****************************
;convert binary to hex ascii
;input: (a) = fb
;output: (h) = 46h (F)
;       (l) = 42h (B)
```

```
        ld a,lf
        call sndchr
        ret
;
;**********
;time delay
;**********
;
delay0:   nop
          ret
;
delay1:   ld b,0
del1:     djnz   del1
          ret
;
delay2:   ld b,0ah
del2:     djnz  del2
          ret
;
delay3: ld hl,0ffffh
loop:   dec hl
        ld a,h
        or l
        jp nz,loop
        ret
;
;*****************************************
;dump data from memory and send to terminal OK
;*****************************************
```

```
;
dump:   call getcmd      ;get command line
        ld de,chrbuf     ;first chr "D"
        inc de
        ld a,(de)        ;next space
        cp 20h
        jp nz,err
        inc de           ;next start address
        ld a,(de)        ;get ascii
        ld h,a           ;first ascii in h
        inc de
        ld a,(de)
        ld l,a           ;second in l
        call hex2bn      ;ascii to hex (in hl result in acc)
        ld hl,first      ;save hi-add in first
        ld (hl),a        ;save
        ld de,chrbuf+4   ;next convert lo address
        ld a,(de)        ;get ascii
        ld h,a
        inc de
        ld a,(de)
        ld l,a
        call hex2bn
        ld hl,first+1    ;save lo-add in first+1
        ld (hl),a
        ld de,chrbuf+6
        ld a,(de)
        cp 20h
        jp nz,err
```

```
        inc de
        ld a,(de)       ;get ascii
        ld h,a          ;first ascii in h
        inc de
        ld a,(de)
        ld l,a          ;second in l
        call hex2bn     ;result in acc
        ld hl,last      ;save hi-add in last
        ld (hl),a       ;save
        ld de,chrbuf+9  ;next convert lo address
        ld a,(de)       ;get ascii
        ld h,a
        inc de
        ld a,(de)
        ld l,a
        call hex2bn
        ld hl,last+1    ;save lo-add in last+1
        ld (hl),a
        ld hl,first+1
        ld a,(hl)
        and 0f0h        ;begin at xxx0h
        ld (hl),a
        call crlf
        ld hl,amap      ;send address header
        call sndout
        call sndadd     ;send add in first,first+1 to terminal
        call space2     ;send 4 spaces
        call space1
dump3:  ld a,(first)    ;hl poit to start address
```

```
        ld h,a
        ld a,(first+1)
        ld l,a
        ld b,10h        ;display hex 10h byte
        push hl
dump1:  push hl
        ld a,(hl)       ;get data
        push bc
        call bn2hex     ;convert to ascii
        pop bc
        ld a,h          ;send to terminal
        call sndchr
        ld a,l
        call sndchr
        ld a,b
        cp 9h
        jr nz,spc1
        call space1     ;send 1 spaces
spc1:   call space1
        pop hl
        inc hl
;
        push bc
        ld b,00h
        ld c,05h
        in a,(c)
        and 80h
        pop bc
        jp z,cnt
```

```
;
        push bc
        ld b,00h
        ld c,09h
        in a,(c)        ;getdata
        pop bc
        jp  warm
;
cnt:    djnz dump1      ;continue
        call space3
        pop  hl
        ld b,10h        ;display ascii 10h chr
dascii: ld a,(hl)       ;load ascii to reg a
        cp 20h
        jr c,j1         ;ascii of control
        cp 80h
        jp c,j2
j1:     ld a,2eh        ;ascii of dot
j2:     call sndchr     ;send ascii to display
        inc hl
;
        ld a,b
        cp 3h
        jr nz,j3
        call space1     ;send 1 spaces
j3:     djnz dascii
;
        ld a,h          ;find total bytes to dump
        ld (first),a
```

```
        ld a,l
        ld (first+1),a
        push hl
        call crlf
        pop hl
        ld a,(last)
        ld d,a
        ld a,(last+1)
        ld e,a
        xor a
        sbc hl,de
        jr nc,dump2
        call sndadd
        call space2     ;send 4 spaces
        call space1
        jp dump3
dump2:  jp  warm
;
errmsg:  .db    cr,lf,"Syntax error,type H for help",cr,lf,04h
;
;****
;test
;****
;
test:   ld hl,testmsg
        call sndout
        ret
;
testmsg: .db cr,lf,"TEST",cr,lf,04H
```

```
;
;**************************
;get one chr from terminal
;**************************
;
getchr:  call gchr       ;get a character
         jr z,getchr
         and 7fh
         or a
         scf
         ret
;
gchr:    push bc
         ld b,00h
         ld c,05h
         in a,(c)
         and 80h
         pop bc
         ret z
         push bc
         ld b,00h
         ld c,09h
         in a,(c)        ;getdata
         pop bc
         or a
         ret
;
;******************************
;get address ret when press cr
```

```
;add save in chrbuf
;*******************************
;
getadd: ld hl,chrbuf     ;clear chr buffer
        ld b,10h         ;10h bytes
gadd1:  ld (hl),00h
        inc hl
        djnz gadd1
        ld hl,chrbuf     ;hl point to chrbuf
gadd3:  push hl
        call getchr      ;get one chr
        call lc2uc       ;convert to upper case
        call sndchr      ;echo back
        pop hl
        cp 0dh           ;cr?
        jr nz,gadd2
        ld (hl),0ffh     ;yes,close data with ffh
        ret
gadd2:  cp 08h           ;back space?
        jr z,gadd3       ;yes get again
        ld (hl),a        ;no save chr
        inc hl
        jr gadd3
;
;*************************************
;get data from terminal save in chrbuf
;cr -> ret,space -> jp to rdy
;*************************************
```

```
getdat: ld hl,chrbuf     ;clear chr buffer
        ld b,10h         ;10h bytes
gdat1:  ld (hl),00h
        inc hl
        djnz gdat1
        ld hl,chrbuf     ;hl point to chrbuf
        ld b,3h          ;test
gdat3:  push hl
        call getchr      ;get one chr
        call lc2uc       ;convert to upper case
        pop hl
        cp 0dh           ;cr?
        jr nz,gdat2
        ld (hl),0ffh     ;yes,close data with 0ffh
        ret
gdat2:  cp 08h           ;back space?
        jr z,gdat3       ;yes get again
        cp 1bh
;
        push af
        push bc
        ld b,00h
        ld c,09h
        in a,(c)         ;getdata
        pop bc
        pop af
;
        jr z,gdat4
                         ;no save chr
```

```
        djnz j4         ;test
        push af
        ld a,08h        ;test
        call sndchr     ;test
        call sndchr     ;test
        dec hl
        dec hl
        ld b,2h
        pop af
j4:     ld (hl),a
        call sndchr     ;echo back
        inc hl
        jr gdat3
gdat4:  pop hl
        jp warm
;
;*****************
;get command line
;*****************
;
getcmd: ld hl,chrbuf    ;clear chr buffer
        ld b,10h        ;10h bytes
gcmd1:  ld (hl),00h
        inc hl
        djnz gcmd1
        ld hl,chrbuf    ;hl point to chrbuf
        ld (hl),a       ;save commnd data
        inc hl
gcmd3:  call getchr     ;get one chr
```

```
        call lc2uc      ;convert to upper case
        call sndchr     ;echo back
        cp 0dh          ;cr?
        jr nz,gcmd2
        ld (hl),0ffh    ;yes,close data with ffh
        ret
gcmd2:  cp 08h          ;back space?
        dec hl          ;if back space hl=hl-1
        jr z,gcmd3      ;yes get again
        inc hl          ;if no back space hl = hl
        ld (hl),a       ;no save chr
        inc hl
        jr gcmd3
;
;***************************
;send help mssg to terminal
;***************************
;
help:   ld hl,hmssg     ;hl point to help mssg
        call sndout     ;send out
        call crlf
        jp warm
;
hmssg:  .db     cr,lf,"  User RAM area start at address 8000h",cr,lf
        .db     cr,lf,"  Command used follow:",cr,lf
        .db     cr,lf,"  D SSSS FFFF              for dump data from memory"
        .db     cr,lf,"  H                        for display this message"
        .db     cr,lf,"  R                        for read data from cache ram"
        .db     cr,lf,"  S                        for set sampling rate"
```

```
        .db     cr,lf,"  T                      for set triggering word"
        .db     cr,lf,"  Z                      for write cache ram "
        .db     cr,lf,"  F5                     for  display pulse train "
;
;*****************************
;convert hex ascii to binary
;input: (h) = 44 (D)
;       (l) = 37 (7)
;output: (a) = d7
;*****************************
;
hex2bn: ld a,l
        call a2hex
        ld b,a
        ld a,h
        call a2hex
        rrca
        rrca
        rrca
        rrca
        or b
        ret
;
a2hex:  sub '0'
        cp 10
        jr c,a2hex1
        sub 7
a2hex1: ret
;
```

```
;
;******************************
;convert lower-case to upper-case
;input: chr lower-case in acc
;output: chr upper-case iin acc
;******************************
;
lc2uc:  cp 'a'
        jr c,exit
        cp 'z'+1
        jr nc,exit
        sub 'a'-'A'
exit:   ret
;
;***********************
;read data from cache ram    OK
;***********************
;
rdall:  ld hl,8000h        ;start add for data save
        ld a,0ffh
        out (0c0H),a       ;set all port A to hi
        ld a,0ch           ;counter clear,cram enable,clear clock,step no pulse
        or 00100000b
        out (0c1h),a
        ld bc,1000h        ;start counter 1000 bytes
rdall1: call pulse         ;inc counter
        ld a,44h           ;get data from pod 1  pb 7 =>DDIREN(if 0 is read)
        out (0c1h),a       ;                     pb 6 =>CNTCLR(if 0 is clear clock)
        in a,(0c2h)        ;                     pb 5 =>STEP(if 0 is enable decode)
```

```
        ld (hl),a       ;                       pb 4 =>CNTCLK(if 0 is clear f/f)
        inc hl
        ld a,45h        ;get data from pod 2
        out (0c1h),a
        in a,(0c2h)
        ld (hl),a
        inc hl
        ld a,4eh        ;get data from pod 3
        out (0c1h),a
        in a,(0c2h)
        ld (hl),a
        inc hl
        ld a,4fh        ;get data from pod 4
        out (0c1h),a
        in a,(0c2h)
        ld (hl),a
        inc hl
        dec c           ;dec counter
        xor a
        or c
        jr nz,rdal1     ;chk counter
        or b
        jr nz,rdal2
        jp warm
rdal2:  dec b
        jp rdal1
;
pulse:  in a,(pb)
        res 5,a
```

```
        out (pb),a
        call delay1
        set 5,a
        out (pb),a
        ret
;
;******************
;set sampling rate
;******************
;
samp:   call getcmd     ;get command line
        ld de,chrbuf
        inc de
        ld a,(de)
        cp 20h          ;chk space
        jp nz,err
        inc de
        ld a,(de)
        cp '1'          ;bgn with 1?
        jp z,one
        cp '2'          ;bgn with 2?
        jp z,secnd
        cp '5'          ;bgn with 5?
        jp z,five
        cp 'E'
        jp z,extl
        jp nz,err
one:    inc de          ;chk tailling zero
        xor a
```

```
        ld b,a
        ld c,10h
        call chkzero
        ld a,b
        or c
        ld c,a
        jp smap
secnd:  inc de          ;chk tailling zero
        xor a
        ld b,a
        ld c,20h
        call chkzero
        ld a,b
        or c
        ld c,a
        jp smap
five:   inc de          ;chk tailling zero
        xor a
        ld b,a
        ld c,50h
        call chkzero
        ld a,b
        or c
        ld c,a
smap:   cp 11h          ;10 hz
        jr nz,s21
        ld a,75h
        ld (drate),a
        jp ssss
```

```
s21:    cp 21h          ;20 hz
        jr nz,s51
        ld a,7dh
        ld (drate),a
        jp ssss
s51:    cp 51h          ;50 hz
        jr nz,s12
        ld a,61h
        ld (drate),a
        jp ssss
s12:    cp 12h          ;100hz
        jr nz,s22
        ld a,65h
        ld (drate),a
        jp ssss
s22:    cp 22h          ;200hz
        jr nz,s52
        ld a,6dh
        ld (drate),a
        jp ssss
s52:    cp 52h          ;500hz
        jr nz,s13
        ld a,51h
        ld (drate),a
        jp ssss
s13:    cp 13h          ;1khz
        jr nz,s23
        ld a,55h
        ld (drate),a
```

```
        jp ssss
s23:    cp 23h          ;2khz
        jr nz,s53
        ld a,5dh
        ld (drate),a
        jp ssss
s53:    cp 53h          ;5khz
        jr nz,s14
        ld a,41h
        ld (drate),a
        jp ssss
s14:    cp 14h          ;10khz
        jr nz,s24
        ld a,45h
        ld (drate),a
        jp ssss
s24:    cp 24h          ;20khz
        jr nz,s54
        ld a,4dh
        ld (drate),a
        jp ssss
s54:    cp 54h          ;50khz
        jr nz,s15
        ld a,31h
        ld (drate),a
        jp ssss
s15:    cp 15h          ;100khz
        jr nz,s25
        ld a,35h
```

```
        ld (drate),a
        jp ssss
s25:    cp 25h          ;200khz
        jr nz,s55
        ld a,3dh
        ld (drate),a
        jp ssss
s55:    cp 55h          ;500khz
        jr nz,s16
        ld a,21h
        ld (drate),a
        jp ssss
s16:    cp 16h          ;1mhz
        jr nz,s26
        ld a,25h
        ld (drate),a
        jp ssss
s26:    cp 26h          ;2mhz
        jr nz,s56
        ld a,2dh
        ld (drate),a
        jp ssss
s56:    cp 56h          ;5mhz
        jr nz,s17
        ld a,11h
        ld (drate),a
        jp ssss
s17:    cp 17h          ;10mhz
        jr nz,s27
```

```
        ld a,15h
        ld (drate),a
        jp ssss
s27:    cp 27h          ;20mhz
        jr nz,s57
        ld a,1dh
        ld (drate),a
        jp ssss
s57:    cp 57h          ;50mhz
        jr nz,s18
        ld a,00h
        ld (drate),a
        jp ssss
s18:    jp err
extl:
;        call test
;        inc de
;        cp '+'
;        jr nz,extl1
;        ld a,18h
        ld a,98h
        ld (drate),a
        jp ssss
extl1:  cp '-'
        jp nz,err
        ld a,98h
        ld (drate),a
ssss:   in a,(pb)       ;disable decoder
        and 0f0h
```

```
        or  2dh
        out (pb),a
        ld a,(drate)    ;send word trig
        out (pa),a
        ld b,06h        ;send to u54
        call enable
        call crlf
        jp warm
;
chkzero: ld a,(de)
        cp '0'
        ret nz
        inc b
        inc de
        jr chkzero
;
ratetab: .db    75h,7dh,61h,65h,6dh,51h,55h,5dh,41h,45h,00,00,00,00,00,00
         .db    4dh,31h,35h,3dh,21h,25h,2dh,11h,15h,1dh,00,00,00,00,00,00
         .db    00h,18h,98h,00h,00h,00h,00h,00h,00h,00h,00,00,00,00,00,00
;
;*****************
;send data in acc
;*****************
;
snddat:  call bn2hex     ;convert to ascii
         push hl
         ld a,h          ;send to terminal
         call sndchr     ;send hi add
         pop hl
```

```
        ld a,l          ;then send lo add
        call sndchr
        ret
;
brkpat: .db     cr,lf,"Break at address : ",04h
regaf:  .db     "Register  AF = ",04h
regbc:  .db     "Register  BC = ",04h
regde:  .db     "Register  DE = ",04h
reghl:  .db     "Register  HL = ",04h
regix:  .db     "Register  IX = ",04h
regiy:  .db     "Register  IY = ",04h
brkdsp: .db     "Hit space bar to continue else go to prompt",04h
;
;**************
;set word trig   *?*
;**************
;
strig:  ld hl,trgpat
        call sndout
        call getchr
        push af
        call sndchr
        pop af
        ld (ftrig),a
        cp '0'
        jp nz,ch01
sch00:  ld hl,trgch0
        call sndout
        jr gtr          ;get trig
```

```
;
ch01:   cp '1'
        jp nz,ch02
        ld hl,trgch1
        call sndout
        jr gtr
;
ch02:   cp '2'
        jp nz,ch03
        ld hl,trgch2
        call sndout
        jr gtr
;
ch03:   cp '3'
        jp nz,ch04
        ld hl,trgch3
        call sndout
        jr gtr
;
ch04:   call crlf
        jp strig
;
gtr:    ld hl,chrbuf    ;clear chr buffer
        ld b,10         ;10h bytes
gtr1:   ld (hl),00h
        inc hl
        djnz gtr1
        ld hl,chrbuf    ;hl point to chrbuf
gtr3:   push hl         ;
```

```
        call getchr     ;get one chr
        call lc2uc      ;convert to upper case
        call sndchr     ;echo back
        pop hl
        cp 0dh          ;cr?
        jr nz,gtr2
        ld (hl),0ffh    ;yes,close data with ffh
        jr gtr4
gtr2:   cp 08h          ;back space?
        jr z,gtr3       ;yes get again
        ld (hl),a       ;no save chr
        inc hl
        jr gtr3
gtr4:   ld hl,chset     ;hex
        ld de,chrbuf
        call maskb
        in a,(pb)       ;disable decoder
        and 0f0h
        or  2dh
        out (pb),a
;
        ld a,(ftrig)
        cp '0'
        jp z,sndch0
        cp '1'
        jp z,sndch1
        cp '2'
        jp z,sndch2
        cp '3'
```

```
        jp z,sndch3
        jp warm          ;test
;
;Send Channal 0
;
sndch0: ld a,(chset)     ;send word trig
        out (pa),a
        ld b,0h          ;send to u54
        call enable
        call delay1
        ld a,(chset)
        cpl
        out (pa),a
        ld b,02h         ;send to u68
        call enable
        call crlf
        jp  warm
;
;Send Channal 1
;
sndch1: ld a,(chset)     ;send word trig
        out (pa),a
        ld b,01h          ;send to u54
        call enable
        call delay1
        ld a,(chset)
        cpl
        out (pa),a
        ld b,03h         ;send to u68
```

```
        call enable
        call crlf
        jp  warm
;
;Send Channal 2
;
sndch2: ld a,(chset)    ;send word trig
        out (pa),a
        ld b,08h         ;send to u54
        call enable
        call delay1
        ld a,(chset)
        cpl
        out (pa),a
        ld b,0ah         ;send to u68
        call enable
        call crlf
        jp  warm
;
;Send Channal 3
;
sndch3: ld a,(chset)    ;send word trig
        out (pa),a
        ld b,09h         ;send to u54
        call enable
        call delay1
        ld a,(chset)
        cpl
        out (pa),a
```

```
        ld b,0bh        ;send to u68
        call enable
        call crlf
        jp  warm
;
enable:  in a,(pb)      ;read value from port b
         and 0f0h       ;
         or  b          ;selec chip,disable gate
         out (pb),a
         call delay2
         and 0dfh       ;enable gate
         out (pb),a
         call delay2    ;delay pulse width
         or 20h         ;disable
         out (pb),a     ;
         call delay2
         and  0f0h
         or  2dh
         out (pb),a
         ld a,0ffh
         out (pa),a
         ret
;
maskb:   ld a,(de)      ;bit 7
        cp '0'
        jr nz,a70
        res 7,(hl)
        jr a72
a70:    cp '1'
```

```
        jr nz,a71
        set 7,(hl)
        jr a72
a71:    res 7,(hl)
        inc hl
        set 7,(hl)
        dec hl
a72:    inc de
        ld a,(de)       ;bit 6
        cp '0'
        jr nz,a60
        res 6,(hl)
        jr a62
a60:    cp '1'
        jr nz,a61
        set 6,(hl)
        jr a62
a61:    res 6,(hl)
        inc hl
        set 6,(hl)
        dec hl
a62:    inc de
        ld a,(de)       ;bit 5
        cp '0'
        jr nz,a50
        res 5,(hl)
        jr a52
a50:    cp '1'
        jr nz,a51
```

```
        set 5,(hl)
        jr a52
a51:    res 5,(hl)
        inc hl
        set 5,(hl)
        dec hl
a52:    inc de
        ld a,(de)       ;bit 4
        cp '0'
        jr nz,a40
        res 4,(hl)
        jr a42
a40:    cp '1'
        jr nz,a41
        set 4,(hl)
        jr a42
a41:    res 4,(hl)
        inc hl
        set 4,(hl)
        dec hl
a42:    inc de
        ld a,(de)       ;bit 3
        cp '0'
        jr nz,a30
        res 3,(hl)
        jr a32
a30:    cp '1'
        jr nz,a31
        set 3,(hl)
```

```
        jr a32
a31:    res 3,(hl)
        inc hl
        set 3,(hl)
        dec hl
a32:    inc de
        ld a,(de)       ;bit 2
        cp '0'
        jr nz,a20
        res 2,(hl)
        jr a22
a20:    cp '1'
        jr nz,a21
        set 2,(hl)
        jr a22
a21:    res 2,(hl)
        inc hl
        set 2,(hl)
        dec hl
a22:    inc de
        ld a,(de)       ;bit 1
        cp '0'
        jr nz,a10
        res 1,(hl)
        jr a12
a10:    cp '1'
        jr nz,a11
        set 1,(hl)
        jr a12
```

```
a11:    res 1,(hl)
        inc hl
        set 1,(hl)
        dec hl
a12:    inc de
        ld a,(de)       ;bit 0
        cp '0'
        jr nz,a00
        res 0,(hl)
        jr a02
a00:    cp '1'
        jr nz,a01
        set 0,(hl)
        jr a02
a01:    res 0,(hl)
        inc hl
        set 0,(hl)
        dec hl
a02:    inc de
        ret
;
trgpat:  .db    cr,lf,"Set trigger bit, 1 for logic 1, 0 for logic 0 and D

         .db    cr,lf,"Channel 0 for bit 7 .......bit  0",cr,lf
         .db    cr,lf,"Channel 1 for bit 15 ......bit  8",cr,lf
         .db    cr,lf,"Channel 2 for bit 23 ......bit 16",cr,lf
         .db    cr,lf,"Channel 3 for bit 24 ......bit 31",cr,lf
         .db    cr,lf,"Select channel for setting : ",04h
trgch0:  .db    cr,lf,"Set trigger bits of CH0 : ",04h
trgch1:  .db    cr,lf,"Set trigger bits of CH1 : ",04h
```

```
trgch2:  .db    cr,lf,"Set trigger bits of CH2 : ",04h
trgch3:  .db    cr,lf,"Set trigger bits of CH3 : ",04h
;
;*****************************************
;send add hex in first,first+1 to terminal
;*****************************************
;
sndadd: ld hl,first      ;send start address
        ld a,(hl)
        call bn2hex      ;convert hi add
        ld a,h
        push hl
        call sndchr      ;send hi add
        pop hl
        ld a,l
        call sndchr
        ld hl,first+1    ;next convert lo address
        ld a,(hl)
        call bn2hex
        ld a,h
        push hl
        call sndchr      ;send lo add out
        pop hl
        ld a,l
        call sndchr
        ret
;
amap:   .db     cr,lf,"      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F"
;
```

```
;************************
;send space to terminnal
;************************
;next call
space4: call space3      ;send 4 space to terminal
space3: call space2      ;send 3 space to terminal
space2: call space1      ;send 2 space to terminal
space1: ld a,20h         ;send 1 space to terminal
        call sndchr
        ret
;
;*************************************
;send msssg to terminal,mssg point by hl
;*************************************
;
sndout:  ld a,(hl)       ;get data
         cp 04h          ;eot
         ret z
         push bc
         push af
         ld b,00h
sout1:   ld c,05h
         in a,(c)
         and 02h
         jr z,sout1
         pop af
         ld c,07h
         out (c),a       ;send out
         pop bc
```

```
          inc hl
          jr sndout
;
;*******************************
;send one character to terminal
;data in acc
;*******************************
;
sndchr: push bc
        push af
        ld b,00h
sout2:  ld c,05h
        in a,(c)
        and 02h
        jr z,sout2
        pop af
        ld c,07h
        out (c),a       ;send out
        pop bc
        ret
;
;
        .end
```

ภาคผนวก 3

โปรแกรมควบคุมระบบของลอจิกแอนะไลเซอร์

```
#include <dos.h>
#include <stdio.h>
#include <time.h>
#include <conio.h>
/*************************************************************************

        Notes on latest VERSION:
        Directions for creating an executable run file:

                        PRODUCTS USED
                        -------------
                        IBM PC    DOS  V3.10

  IBM PC     C    V1.00
  TURBO C++ V1.00 UP
  TURBO ASM V1.00 UP

1) TERMINAL;  terminal.cppuse turbo C++
2) SCRNIO;          scrnio.cppuse turbo C++
3) LOGIC1;logic1.cppuse turbo C++
3) TASM KBDIO;              kbdio.asmuse turbo assembly
4) TASM KTASYNC;            ktasync.asmuse turbo assembly
5) tlink \tc\lib\c0m terminal scrnio kbdio ktasync logic1,
terminal,mterminal,\tc\lib\graphics \tc\lib\cm
*************************************************************************/
```

```
#define F1   59
#define F2   60
#define F3   61
#define F4   62
#define F563
#define F9   67
#define F10  68
#define CR      0x0D
#define BS      0x08
#define XOFF    0x13
#define EOFM    0x1A
#define FILLER  0xFF
#define BEL     0x07
#define ETX     0x03

voidanalyzer(void);
int term(int port,int argc);
int getline(char *s,int lim);          /* get line into s, return length */
void setbaud(int port,int baudrate);
void commclean(int port,char c);
void endterm(int port,int base);
void   dumpbin(int port);
void help(void);
void readasci(int port);
```

```
int logo(int argc,char *argv[_);
voideel(void);
intsavloc(void);
voidreloc(int oldloc);


extern"C" {charinpcnt(int port);
charcommin(int port);
voidset_xoff(int port,char c);
void comminit(int port);
voiduninit(int port);
voidcommflush(int port);
voidcommout(int port,char c);
int charin(void);
charrcvd_xoff(int port);
charrcv_err(int port);
}


char scrn;       /* Screen Output of Data During Transmissions toggle */
int cport;       /* Global communication port integer value           */


main(int argc,char *argv[_)
  {
int x;
//int cport;no need extern
```

```
cport = 0;
cport = logo(argc,argv);
x = term(cport,argc);
    argc = 1;
    while (x == 1) { cport = logo(argc,argv); x = term(cport,argc); }
  }

/*+++
logo(argc,argv) -- Prints Training logo and returns COM port --
*/

int logo(int argc,char *argv[])
 {
   char c, buf[3];
   int x, baudrate, port;

   if (argc < 3)
           {
 clrscr();
             if (cport == 0)
                    {
  gotoxy(13,1);
  printf("*************************************************");
  gotoxy(13,2);
  printf("*                                               *");
```

```
gotoxy(13,3);
printf("*                    TAVON COMPUTER                    *");
gotoxy(13,4);
printf("*        IBM PC/XT COMPATIBLE TRAINING KIT             *");
gotoxy(13,5);
printf("*              @ Copyright 1989, 1991                  *");
gotoxy(13,6);
printf("*            Version 5.5, June 1, 1991                 *");
gotoxy(13,7);
printf("*                                                      *");
gotoxy(13,8);
printf("********************************************************");
gotoxy(17,11);
printf("** PRESS F3 to display function keys **");
gotoxy(1,13);
printf( "Which port do you wish to use? [1_");
                    buf[0_ = 48;
                   while (*buf != 49 && *buf != 50)
                        {
 gotoxy(36,13);
 x = getline(buf,3);
 if (*buf == '\0')
{ *buf = '1'; break; }
}
```

```
   port = *buf - 48;
}
else port = cport;
 gotoxy(1,16);
 printf("Which baud rate do you wish to use? [ 9600_");
 gotoxy(8,17);
 printf("0)   300\n");
 printf("\t1)  1200\n");
 printf("\t2)  1800\n");
 printf("\t3)  2400\n");
 printf("\t4)  4800\n");
 printf("\t5)  9600\n");
 printf("\t6) 19200\n");
 c = 47;
 while ((c < 48) || (c > 54))
{
   gotoxy(45,16);
   printf("  \b\b");
   getline(buf,2);
   if (*buf != 0) c = *buf;
else { c = '5'; break; }
}
    }
   else { port = *argv[1_ - 48; c = *argv[2_; }//assign port and baud
    x = c;                                          //from command line
```

```
    switch (x)
       {
         case '0': baudrate =   300; break;
         case '1': baudrate =  1200; break;
         case '2': baudrate =  1800; break;
         case '3': baudrate =  2400; break;
         case '4': baudrate =  4800; break;
         case '5': baudrate =  9600; break;
         case '6': baudrate = 19200;
  }
    setbaud(port,baudrate);
    return(port);
 }

/*+++
term(port,argc)          -- Terminal .. returns on F10 --
*/

int term(int port,int argc)
 {
   char c, flag, tflag;
   int x,y,base,pchar;
   //char scrn;no need extern

   if (argc < 3) clrscr();
```

```
   if (port == 1) base = 0x3F8;          /* COM1                              */
else base = 0x2F8;          /* COM2                                          */
   outp(base+3,7);                       /* No Parity -> N,8,2 == 7           */
                                         /* Parity -> E,8,1 == 27             */
   comminit(port);                       /* Init port                         */
   set_xoff(port,1);                     /* We want auto XON/XOFF protocol    */
   outp(base+4,11);                      /* DTR, RTS and OUT2 on              */
   scrn = 0;                             /* Screen output off                 */
   y = c = tflag = flag = 0;
   outp(base+1,0);                       /* Reset interrupts on 8250          */
   outp(base+1,1);                       /* Set interrupts on 8250            */

   while (inpcnt(port)) c = commin(port);//clear buffer
   while (inpcnt(port) == 0)
      {
        if (y == 25)
           {
delay(1000);
             outp(base+1,0);
             outp(base+1,1);
             while (inpcnt(port)) c = commin(port);
           }
        commout(port,' ');
        y++;
        for (x = 1; x < 50; x++) printf(" \b");
```

```
        if (y >= 50)
           {
             printf("No response from Training kit.\n");
             printf("Retry? [<CR>=Yes_ ");
             while (((x = charin()) == -1) && (inpcnt(port) == 0)) ;
             if (!inpcnt(port))
                if (x != CR)
                        { printf("No."); return(-1); }
                   else {
                         y = 0;
                         printf("Yes.\n");
                         outp(base+1,0);                    /* Reset INTs */
                         outp(base+1,1);                    /* Set INTs   */
                         while (inpcnt(port)) c = commin(port);
                        }
           }
}

 for (;;)
    {
       if ((pchar = charin()) != -1)
          {
            c = pchar;
            switch (flag)
               {
```

```
                case 0: switch (c)
{
  case '$':  putchar(c); commout(port,c); tflag = 1;
                                          if (inpcnt(port)) commclean(port,c);
                                          break;
                                case 0:   flag = 1; break;
                                case XOFF: while ((pchar = charin()) == -1) ;
                                          c = pchar;
                                          if (c == '$')
                                            {
  putchar(c); commout(port,c);
                                              tflag = 1;
                                              commclean(port,c);
                                            }
                                          break;
                                default:  commout(port,c);
                              }
                            break;
                case 1: flag = 0;
                        switch (c)
                          {
  case F10: endterm(port,base); return(0);
                                case F9:  commout(port,'z'); commout(port,CR);
endterm(port,base); return(1);
  case F1:  dumpbin(port); break;
```

```
  case F2:  readasci(port); break;
  case F3:  help(); break;
  case F4:  scrn = !scrn;
printf("\ndriver: Output ");
if (scrn) printf("On.");
 else printf("Off.");
commout(port,CR);
break;
  case F5: //endterm(port,base);
analyzer();
comminit(port);  //init port
set_xoff(port,1); //We want auto XON/XOFF protocol
outp(base+4,11); // DTR, RTS and OUT2 on
commflush(port);
commout(1,'\r');
break;
                        }
                }
            }
        if (inpcnt(port))
            {
              c = commin(port);
if (c == '$') { if (!tflag) putchar(c); }
else { putchar(c); tflag = 0; }
```

```
 if (c == BS) { putchar(' '); putchar(c); }

           }

  }

 }


/*

commclean(port,c)                    -- Communicatios port cleanup procedure --

*/


void commclean(int port,char c)

 { delay(1000); commflush(port); commout(port,c); }


/*

endterm(port,base)                   -- Ends terminal emulation --

*/


void endterm(int port,int base)

 /* Disable DTR, RTS, OUT2 */

 { commflush(port); outp(base+4,0); uninit(port); }



/*+++

setbaud(port,baudrate)               -- Sets baud on port to baudrate --

*/
```

```
void setbaud(int port,int baudrate)
 {
   int base;
   switch (port)
      {
        case 1:  base = 0x3F8; break;
        case 2:  base = 0x2F8; break;
        default: base = 0x3F8;
      }
   switch (baudrate)
      {
        /* Access baud rate divisor: LSB of divisor and then MSB of divisor */
        case   300: outp(base+3,0x80); outp(base,0x80); outp(base+1,1); break;
        case  1200: outp(base+3,0x80); outp(base,0x60); outp(base+1,0); break;
        case  1800: outp(base+3,0x80); outp(base,0x40); outp(base+1,0); break;
        case  2400: outp(base+3,0x80); outp(base,0x30); outp(base+1,0); break;
        case  4800: outp(base+3,0x80); outp(base,0x18); outp(base+1,0); break;
        case  9600: outp(base+3,0x80); outp(base,0x0C); outp(base+1,0); break;
        case 19200: outp(base+3,0x80); outp(base,0x06); outp(base+1,0); break;
        default:    outp(base+3,0x80); outp(base,0x0C); outp(base+1,0);
      }
 }

/*
```

```
getline(s,lim)          -- gets a line into s with limit lim
*/

int getline(char *s,int lim)          /* get line into s, return length */

 {
   int i;
   char c;

   i = 0;
   while ((c = getchar()) != '\n' && i < lim)
 if (c == '\b' && i != 0) i--;
 else {*(s+i) = c; i++; }
   *(s+i) = '\0';
   return(i);
 }

/*+++
dumpbin(port)           -- Dumps a file to programmer --
*/

void dumpbin(int port)
 {
   char c, filename[64_, buf;
   FILE *fp;
```

```
  int ch, pchar, x, z;
  //char scrn;no need extern

  printf("\ndriver: Name of download file? ");
  getline(filename,64);
  if (*filename == 0)
{ printf("driver: no action taken"); commout(port,'$');  }
  fp = fopen(filename,"rb");
  if (fp == 0)
{ printf("driver: file not found"); commout(port,'$');  }
  printf("driver: downloading..."); commflush(port); x = 0; z = 0;
  for (;;)
     {
       if (!rcvd_xoff(port) && (inpcnt(port) == 0))
          {
            if (z == 0)
               {
                 ch = fgetc(fp);
                 if (ch == EOF) { buf = FILLER; z = 1; }
                          else {
                                 buf = ch;
                                 if (scrn)
if (buf != EOFM) putchar(buf);
                                }
               }
```

```
            commout(port,buf);
          }
        if (inpcnt(port))
          { if ((c = commin(port)) != BEL) x = 1; break; }
        if ((pchar = charin()) != -1)
          { c = pchar; if (c == '$' || c == ETX) { x = 2; break; } }
      }
   if (x != 0)
          if (x == 1)
{ printf("\ndriver: error during download"); putchar(c); }
            else
              {
               commout(port,'$');
               printf("\ndriver: download terminated by operator");
               printf("\ndriver: reset Training kit on a binary download.");
              }
  else { printf("\ndriver: download complete"); putchar(c); }
   fclose(fp);
}

/*+++
void readasci(int port)          -- copies data from prom to file --
*/
```

```
void readasci(int port)
 {
   char c, filename[64_, buf[2_;
   FILE *fp;
   int count, ch, pchar, flag, copy, bytcnt, x, z, res;
   //char scrn;no need extern

   putchar(13); printf("driver: Name of file to copy to? ");
   getline(filename,64);
   if (*filename == 0)
  { printf("driver: no action taken"); commout(port,CR);  }
   fp = fopen(filename,"rb");
   if (fp != 0)
      {
        fclose(fp);
        printf("driver: File exists!  Overwrite it? (Y/N)? ");
        getline(buf,2);
        if ((*buf != 'Y') && (*buf != 'y'))
   { printf("driver: no action taken"); commout(port,CR);  }
  }
   fp = fopen(filename,"w+b");
   if (fp == 0)
  { printf("driver: Can't open ", filename); commout(port,CR); }
   commout(port,'$');
   printf("driver: Copy will activate upon next <CR>...");
```

```
delay(1000); commflush(port); commout(port,CR);
flag = copy = bytcnt = z = res = 0;
for (;;)
   if (inpcnt(port))
           {
              if (copy && rcv_err(port))
                 {
                   commout(port,'$');
                   printf("\ndriver: Reception error abort...");
                   printf("\ndriver: Buffer flushed.");
                   commflush(port);
                   break;
                 }
              c = commin(port);
              if (scrn || !copy)
{ putchar(c); if (c == BS) { putchar(' '); putchar(c); } }
              if (copy && c == '>')
                 {
                   if ((z = fseek(fp,-1L,1)) != 0) break;
                   while (fgetc(fp) != 10)
                      {
                        if ((z = fseek(fp,-1L,1)) != 0) break;
                        if ((res = fputc(' ',fp)) == EOF)
                           { printf("\ndriver: Write Error"); break; }
```

```
            if ((z = fseek(fp,-2L,1)) != 0) break;
          }
        if (z == 0)
          if ((z = fseek(fp,0L,1)) == 0)
            if ((res = fputc(EOFM,fp)) == EOF)
              printf("\ndriver: Write Error");
        break;
      }
    if (copy) if ((res = fputc(c,fp)) == EOF)
              { printf("\ndriver: Write Error"); break; }
    bytcnt++;
    if (bytcnt % 100 == 0)
      if ((pchar = charin()) != -1)
        {
          c = pchar;
          if (c == CR && !flag)
            {
              commflush(port);
              copy = 1;
              if (!scrn) printf("...");
              commout(port,CR);
            }
          else if (c == '$')
                 {
                   commout(port,c);
```

```
                                printf("\ndriver: Copy aborted by operator...");
delay(1000);
                                commflush(port);
                                commout(port,CR);
                                break;
                            }
                        else if (flag) flag = 0;
                        else if (c == 0) flag = 1;
                        else commout(port,c);
                    }
            }
        else if ((pchar = charin()) != -1)
            {
            c = pchar;
            if (c == CR && !flag)
                {
                commflush(port);
                copy = 1;
                if (!scrn) printf("...");
                commout(port,CR);
                }
            else if (c == '$')
                    {
                      commout(port,c);
```

```
                                printf("\ndriver: Copy aborted by operator...");
 delay(1000);
                                commflush(port);
                                commout(port,CR);
                                break;
                              }
                        else if (flag) flag = 0;
                        else if (c == 0) flag = 1;
                        else commout(port,c);
                      }
    if (z != 0 || c == '$' || res == EOF)
             {
               fclose(fp);
               unlink(filename);
               if ((z != 0 || res == EOF) && !scrn) commout(port,CR);
    }
       else fclose(fp);
    if (!scrn) commout(port,CR);
 }

/*+++
help()            -- prints help --
*/
```

```
void help(void)
 {
   int x, oldloc;

   oldloc = savloc();
   gotoxy(0,0);
   eel();
   printf("Function Keys:\n");
   eel();
   putchar('\n');
   eel();
   printf("     F1  --  Download a file to Training kit");
   printf("\tF2  --  Copy data from Training kit to a file\n");
   eel();
   printf("     F3  --  List Function Keys      ");
   printf("\tF4  --  Toggle Screen Output of Data\n");
   eel();
   printf("     F9  --  Restart program         ");
   printf("\tF10 --  Exit to DOS\n");
   eel();
   putchar('\n');
   eel();
   for (x = 0; x < 7; x++) printf("-----------");
   reloc(oldloc);
 }
```

```
/*this program was written for logic analyzer machine*/
#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>

#define F1  59
#define F2  60
#define F3  61
#define F4  62
```

```
/*this program was written for logic analyzer machine*/
#include <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <dos.h>

#define F1  59
#define F2  60
#define F3  61
#define F4  62
#defineESC27
#define R_ARROW77
#define LR_ARROW 116
#define L_ARROW75
#define LL_ARROW 115
#define U_ARROW0x48
#define D_ARROW 0x50
#define INDEX_RANGE 100
#defineINDEX_SPACE10

//***************      prototype      ************************
void    analyzer(void);
void gr_init(void);
void new_color(void);
```

```
voiddisp_logic(unsigned int start_add,char page
,unsigned char data[2048_[4_);
void write_ch(unsigned int space,char page);
voidfind_CR();
void draw_ref(unsigned int a,unsigned int b);
charpointer_manager(unsigned int x);
voidshow_add(unsigned int x,unsigned char data[2048_[4_);
void setbau(int port,int baudrate);
voidread_data(unsigned char data[2048_[4_);
voidsend_comm(void);
charhi_lo(unsigned char chan
,unsigned char new_data,unsigned char old_data);
unsigned charreceive_comm(void);
unsigned charwhich_byte(unsigned char chan);

extern"C" {charinpcnt(int port);
charcommin(int port);
voidcommflush(int port);
voidcommout(int port,char c);
voidset_xoff(int port,char c);
}
//*****************   main program **********************************

/*void main(void)
{
```

```
    analyzer();
} use here for stand alone logic analyzer only*/

//*****************************************************************

void    analyzer(void)//logic analyzer function
//dump data from trainer kit and display as
//logic diagram
{
unsigned chardata[2048_[4_;
unsigned int size,address=0,index_pos=0;
void *index;
charch=0,page=0,flag=1,second_key=0;

setbau(0,0);  //set com1 as 9600,n,8,1
printf("Please wait ,,,\n");
read_data(data);    //read data from triner kit into data[_[_ array
gr_init();//initial graphic
//---------------- grab index line ----------------------
line(0,0,0,getmaxy()-20);              //line for grab
size = imagesize(0,0,0,getmaxy()-20);   //get memory size for image
index = malloc(size);                   //allocate memory
getimage(0,0,0,getmaxy()-20,index);  //grab line
putimage(0,0,index,XOR_PUT);//erase index line
```

```
new_color();     //set new palette
for(;ch!=0xf;)       //exit when press 'ESC'
{
if (flag)//draw new page of logic data
{
cleardevice();//clear graphics screen
disp_logic(address,page,data);//display logic of (address,page)
putimage(52+index_pos*5,0,index,XOR_PUT);
}
flag = 0;//clear redraw status
show_add(address+index_pos,data);//display index address' value
second_key=0;                    //status for double byte key
if (!(ch = getch()))//check double byte key
{ch = getch();second_key=1;}
if (second_key) switch(ch)//is double byte key
{
case LR_ARROW:index_pos = INDEX_RANGE;//long right arrow
case R_ARROW:if (pointer_manager(index_pos)!=1)
{
putimage(52+index_pos*5,0,index,XOR_PUT);//erase
index_pos++;  //move to next address
putimage(52+index_pos*5,0,index,XOR_PUT);//draw line
}
else
{
```

```
if (address<0x7b0) address += INDEX_RANGE-INDEX_SPACE;
index_pos = INDEX_SPACE;//offset of page
flag=1;//draw new page
}
break;
case LL_ARROW:index_pos = 0;flag=1;//long left arrow
case L_ARROW:if (pointer_manager(index_pos)!=-1)//left arrow
{
putimage(52+index_pos*5,0,index,XOR_PUT);//erase
index_pos--;          //move to previous address
putimage(52+index_pos*5,0,index,XOR_PUT);
}
else if (address>=INDEX_RANGE-INDEX_SPACE)//address is out
{                                         //of page
address -= INDEX_RANGE-INDEX_SPACE;
flag=1;                    //draw new page
index_pos = INDEX_RANGE-INDEX_SPACE;
}
break;
case D_ARROW:page=1;//down arrow show ch16 - ch31
cleardevice();
flag=1;
break;
case U_ARROW:page=0;//up arrow show ch0 - ch15
cleardevice();
```

```
flag=1;
break;
}
else switch(ch)//is not double byte key
{
case ESC:ch=0xf;break;//quit program
}
}
outp(0x3f9,1); //restore 8250 interrupt
closegraph();        //close graphic screen
}

voiddisp_logic(unsigned int start_add,char page//display logic on screen
,unsigned char data[2048_[4_)
{
unsigned charoffset,chan;
unsigned charnew_data[4_,old_data[4_;

unsigned intspace_y,gen_pur;
charup=-10,down=10,transition,byte_num;

space_y=getmaxy()/18;                    //space of Y-axis
write_ch(space_y,page);//write ch number on screen
draw_ref(50,space_y+textheight("c"));//draw reference logic
for (offset=0;offset<117;offset++)
```

```
{
for (gen_pur=0;gen_pur<=3;gen_pur++)
{
  new_data[gen_pur_ = data[offset+start_add_[gen_pur_;//give new data
  if (offset==0) old_data[gen_pur_=new_data[gen_pur_;
}
for (chan=0;chan<=15;chan++)
{
byte_num = which_byte(chan)+(page*2);
moveto(50+(offset*5),(chan+2)*space_y+textheight("c"));
switch(transition = hi_lo(chan,new_data[byte_num_
  ,old_data[byte_num_))
{
case -2: break;// do nothing low to low
case -1: linerel(0,up);moverel(0,down);break;//high to low
case  1: linerel(0,up);break;                  //low to high
case  2: moverel(0,up);break; //high to high
}
linerel(5,0);
}
for (gen_pur=0;gen_pur<=3;gen_pur++)
old_data[gen_pur_ = new_data[gen_pur_;//give new old_data
}
}
```

```
voidshow_add(unsigned int x,unsigned char data[2048_[4_)
//display address and data at the index line
{
charscreen_out[80_;

setviewport(0,getmaxy()-10,getmaxx(),getmaxy(),1);//set new viewport
clearviewport();//clear old data
sprintf(screen_out,"address %04x = %02x %02x %02x %02x",
x,data[x_[3_,data[x_[2_,data[x_[1_,data[x_[0_);
outtextxy(0,0,screen_out);//display text
setviewport(0,0,getmaxx(),getmaxy(),1);//restore view port
}

voidread_data(unsigned char data[2048_[4_)//get data from trainer kit
{
intx,y;
unsigned chartemp,temp1;
div_ttest;
#define LINE  512//this num cannnot exceed 512

delay(1000);//make sure that 8250 has already init itself
send_comm();//send command to emulator

for (x=0;x<=(4*LINE)-1;x++)//one line equal to 4*x-1
{
```

```
test = div(x,4);
if (!test.rem) find_CR();//find carriage return
for (y=0;y<=3;y++)
{
do temp = receive_comm();
while(temp==0x20);
if (temp<0x40) temp -= 0x30; else temp -= 0x37;//convert
temp1 = receive_comm();
if (temp1<0x40) temp1 -= 0x30; else temp1 -= 0x37;//convert
data[x_[y_ = (temp<<4) + temp1;//combine to get data value
}
}
}

voidsend_comm(void)
{
charoutput[50_ = {"\rd8000 9fff\r\n"};// \r is <carriage return>
chartemp;                              // 1 line eq to 0f byte

    temp=0;
do{
if ((inportb(0x3fd) & 0x20)!=0x20) //check status
delay(100);
outp(0x3f8,output[temp_);//send command to emulator
temp++;
```

```
}while(output[temp_!='\n');
}

voidfind_CR()
{
unsigned charx,in,temp[3_={0x0d,0x0a,0x30};

for (x=0;x<=2;x++)//compare with temp which is carriage return
{
in=receive_comm();
if (x<2) {if (in != temp[x_) x=0;} else {if (in < temp[x_) x=0;}
}
for (x=1;x<=3;x++) receive_comm();//to skip unused data
}

// combine logic2 here

//-------------------------------------------------------------------
void setbau(int port,int baudrate)//set baud rate and port
{
int base;

switch (port)
      {
        case 1:  base = 0x3F8; break;
```

```
        case 2:  base = 0x2F8; break;
        default: base = 0x3F8;
      }
   switch (baudrate)
      {
        /* Access baud rate divisor: LSB of divisor and then MSB of divisor */
        case   300: outp(base+3,0x80); outp(base,0x80); outp(base+1,1); break;
        case  1200: outp(base+3,0x80); outp(base,0x60); outp(base+1,0); break;
        case  1800: outp(base+3,0x80); outp(base,0x40); outp(base+1,0); break;
        case  2400: outp(base+3,0x80); outp(base,0x30); outp(base+1,0); break;
        case  4800: outp(base+3,0x80); outp(base,0x18); outp(base+1,0); break;
        case  9600: outp(base+3,0x80); outp(base,0x0C); outp(base+1,0); break;
        case 19200: outp(base+3,0x80); outp(base,0x06); outp(base+1,0); break;
        default:    outp(base+3,0x80); outp(base,0x0C); outp(base+1,0);
  }
  outp(base+3,7); //no parity 8 bit 2 stop
  outp(base+4,11);
  //outp(base+1,0);
}

unsigned charreceive_comm(void)//com receive macro
{
chartemp;

do{
```

```
temp = inport(0x3fd);
}while((temp & 0x1) != 0x1 );//check status
return(inportb(0x3f8));
}

void draw_ref(unsigned int a,unsigned int b)//draw reference line
{
charx;

moveto(a,b);
for (x=0;x<59;x++)
{
linerel(5,0);
linerel(0,-10);//up
linerel(5,0);
linerel(0,10);//down
}
}

charpointer_manager(unsigned int x)//to indicate is at the end
{//or at the end of disp range
switch(x)
{
case INDEX_RANGE:return(1);
case 0:return(-1);
```

```
default:return(0);
}
}


unsigned char which_byte(unsigned char chan)//for indicate byte to disp
{
div_tx;
x = div(chan,8);
return(x.quot);
}


charhi_lo(unsigned char chan//for indicate transition
,unsigned char new_data,unsigned char old_data)
{
div_tx;  //x.quot mean which byte x.rem mean which bit
//x.rem mean which bit


x = div(chan,8);
if (((old_data ^ new_data) >> x.rem) & 0x01) //change
{
if ((old_data >> x.rem) & 0x01) return(-1);
else return(1);  //-1 = high to low , 1= low to high
}
else //unchange
```

```
{
if ((old_data >> x.rem) & 0x01) return(2);
else return(-2);//-2 = low to low , 2 = high to high
}
}

void gr_init(void)//init graphic mode
{
   int gdriver = VGA, gmode = VGAHI, errorcode;

   /* initialize graphics mode */
   initgraph(&gdriver, &gmode, "");

   /* read result of initialization */
   errorcode = graphresult();

   if (errorcode != grOk)  /* an error occurred */
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);             /* return with error code */
   }
}
```

```
void new_color(void)
{
   unsigned char i,j,k,l;

   for (i=0;i<=3;i++) for (j=0;j<=3;j++) setpalette(k++,i+j*4);
}

void write_ch(unsigned int space,char page)//disp ch number
{
char x;
    charscreen_out[80_;

for (x=0;x<=15;x++)//assign ch number to screen
{
if (page) sprintf(screen_out,"ch %d",x+16);
else sprintf(screen_out,"ch %d",x);
outtextxy(0,(x+2)*space,screen_out);
}
}
```

ภาคผนวก 4

ผลงานวิจัยในระหว่างการศึกษาปริญญาโท ที่ได้รับการเผยแพร่ในการสัมมนาทางวิชาการวิศวกรรมไฟฟ้า 9 สถาบัน ครั้งที่ 11

การประยุกต์ ไอบีเอ็ม พีซี สำหรับเป็นดาต้าแอนนะไลเซอร์ชนิดข้อมูลแบบอนุกรม

วิชิต ตั้งสุขนิรันดร

ดร. วัลลภ สุระกำพลธร

ภาควิชาอิเล็คทรอนิคส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้า

เจ้าคุณทหารลาดกระบัง

บทคัดย่อ

บทความนี้เสนอผลงานวิจัยเกี่ยวกับการประยุกต์ ไอบีเอ็ม พีซี (IBM. PC) มาใช้ทำหน้าที่เป็น ดาต้าแอนนะไลเซอร์(DATA ANALYZER) ซึ่งดาต้าแอนนะไลเซอร์ตัวนี้จะสามารถใช้ตรวจสอบข้อมูลได้ทั้งแบบอนุกรม และ แบบขนาน แต่บทความนี้จะขอกล่าวถึงแบบอนุกรมเพียงอย่างเดียว ซึ่งมีความสะดวกในการใช้งาน โดยมีฟังก์ชั่นการใช้งานที่ครบถ้วน ทำให้ประหยัดรายจ่ายที่ต้องไปซื้อดาต้าแอนนะไลเซอร์ พร้อมกันนั้นยังเป็นการขยายขีดความสามารถของ ไอบีเอ็ม พีซีออกไปอีก

Abstract

In this paper, a modification of the IBM PC to work as DATA - ANALYZER is presented. This DATA ANALYZER can detect in serial mode. It has complete functions in serial mode, easy to use and low cost.

1. บทนำ

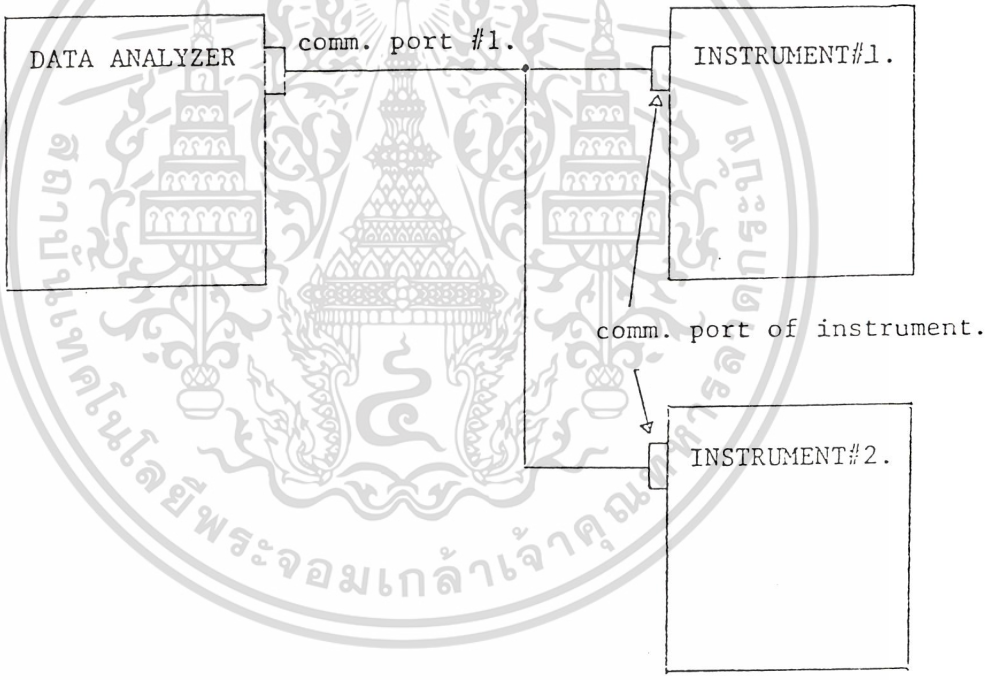
ในยุคปัจจุบันนี้ ดังที่ท่านทั้งหลายได้ทราบอยู่แล้วว่า การติดต่อสื่อสารข้อมูลระหว่างอุปกรณ์ต่าง ๆ กับคอมพิวเตอร์ หรือระหว่างอุปกรณ์กับอุปกรณ์ ได้มีบทบาทและมีความสำคัญเพิ่มมากขึ้นตลอดเวลา ในการติดต่อสื่อสารข้อมูลของอุปกรณ์เหล่านี้ อาจเกิดความผิดพลาดขึ้นได้ จึงจำเป็นอยู่เองที่ต้องมีอุปกรณ์ในการตรวจสอบและวิเคราะห์ข้อมูล ที่จะใช้ในการติดต่อสื่อสารกันนี้ เพื่อแก้ปัญหาความผิดพลาดนี้ อุปกรณ์สำหรับตรวจสอบและวิเคราะห์ข้อมูลนี้ก็คือ ดาต้า แอนนะไลเซอร์ (DATA ANALYZER) นั้นเอง แต่การที่จะจัดซื้อหรือสร้าง ดาต้า แอนนะไลเซอร์ เฉพาะอย่างเดียวนั้นก็เป็นการสิ้นเปลืองเงินมาก ดังนั้นถ้าสามารถนำ ไอบีเอ็ม พีซี (IBM PC) ซึ่งเป็นอุปกรณ์ที่ใช้กันแพร่หลายอยู่แล้วมาประยุกต์ใช้งานเป็น ดาต้า แอนนะไลเซอร์ อีกหน้าที่หนึ่ง ก็จะเป็นการประหยัดกว่า และรวมไปถึงทำให้สามารถขยายขีดความสามารถในการใช้งาน ไอบีเอ็ม พีซี ให้กว้างขวางหลายหน้าที่เพิ่มขึ้นไปอีก

ดาต้า แอนนะไลเซอร์ ตัวนี้สามารถใช้ตรวจสอบหรือวิเคราะห์ข้อมูลได้ทั้งแบบอนุกรม และแบบขนาน แต่ในบทความชุดนี้จะขอกล่าวถึงแบบอนุกรมเพียงอย่างเดียว ซึ่งการประยุกต์ก็ไม่ยากโดยเพียงมีโปรแกรมเพิ่มขึ้นมาอีกชุดเดียว ก็สามารถใช้งาน ไอบีเอ็ม พีซี เป็น ดาต้า แอนนะไลเซอร์ ที่ใช้ตรวจสอบและวิเคราะห์ข้อมูลแบบอนุกรมได้แล้ว และในบทความชุดนี้จะมีผลการทดสอบของการใช้งานในแบบอนุกรมให้ดูด้วย

2. ดาต้า แอนนะไลเซอร์ ที่ใช้งานในแบบอนุกรม

โดยปกติการติดต่อสื่อสารข้อมูลระหว่างอุปกรณ์ต่าง ๆ กับคอมพิวเตอร์ หรืออุปกรณ์กับอุปกรณ์ใด ๆ ในแบบอนุกรมจะเป็นการติดต่อสื่อสารแบบ อซิงโครนัส (ASYNCHRONOUS COMMUNICATION) โดยมีมาตรฐานการต่อเป็นแบบ RS-232-C หรือ EIA STANDARD รูปแบบของข้อมูลขึ้นอยู่กับว่าจะติดต่อกันเป็นแบบ ASCII CODE, BINARY CODE หรือ EBCDIC CODE ก็ขึ้นอยู่กับการกำหนดขึ้นมาใช้งานของแต่ละอุปกรณ์

รูปภาพที่ 1. เป็นภาพที่แสดงรูปของบล็อคไดอะแกรมในการเอาดาต้า แอนนะไลเซอร์เข้าไปตรวจสอบหรือวิเคราะห์อุปกรณ์ที่เราสนใจ กล่าวคือเมื่ออุปกรณ์ตัวที่ 1 และตัวที่ 2 มีการติดต่อสื่อสารข้อมูลกันโดยผ่านทางคอมมูนิเคชั่นพอรต์(COMMUNICATION PORT OF INSTRUMENT) แล้วเกิดปัญหาขึ้น เราจำเป็นต้องใช้ดาต้าแอนนะไลเซอร์เข้าไปตรวจสอบ หรือวิเคราะห์ข้อมูลดูว่าความผิดพลาดเกิดขึ้นที่อุปกรณ์ตัวที่ 1 หรือตัวที่ 2 เพื่อจะได้แก้ปัญหาได้อย่างถูกต้องและลุล่วงไป



รูปภาพที่ 1 เป็นบล็อคไดอะแกรมของการใช้ดาต้าแอนนะไลเซอร์

ดังที่ได้กล่าวมาแล้วว่ามาตรฐานการต่อกันเป็นแบบ RS-232-C และกำหนดข้อต่อแบบ DB-25 (DB-25 CONANECTOR) ซึ่งแต่ละขาของข้อต่อกำหนดไว้ดังรูปภาพที่ 2 แต่ในการใช้งานในการวิจัยนี้จะใช้ขาของข้อต่อเพียงขาที่ 2,3,4,5,6,7, และ 20, เท่านั้น ซึ่งแต่ละขาของข้อต่อมีหน้าที่ดังนี้ คือ.-

ขาที่ 2 (Transmit Data) เป็นขาที่สัญญาณข้อมูลถูกส่งออกจากตัวคอมพิวเตอร์ ไปยังอุปกรณ์ตัวที่ต่อด้วย

ขาที่ 3 (Receive Data) เป็นขาที่รับสัญญาณข้อมูลเข้ามายังตัวคอมพิวเตอร์

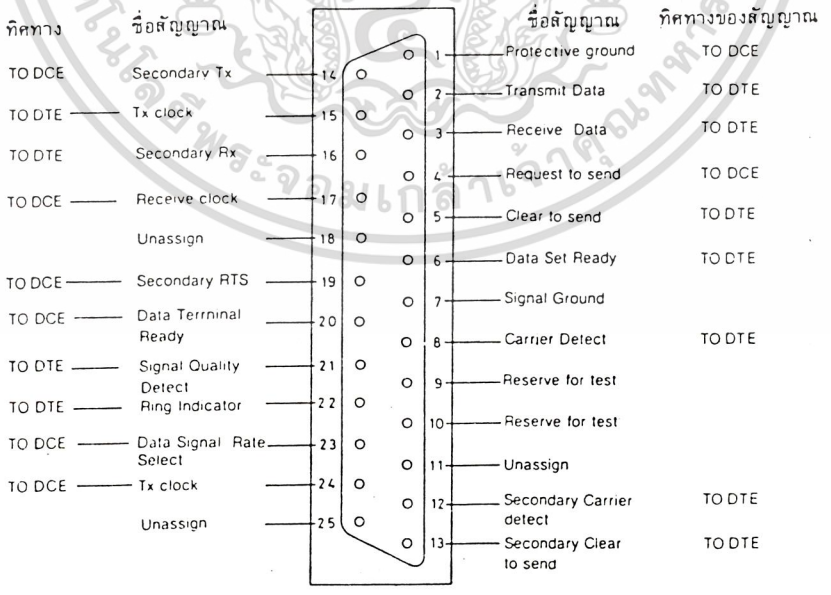
ขาที่ 4 (Request To Send) เป็นขาที่ใช้สำหรับส่งสัญญาณควบคุมจากคอมพิวเตอร์ ไปยังอุปกรณ์ใด ๆ เพื่อตรวจสอบว่าอุปกรณ์ พร้อมจะรับสัญญาณข้อมูลได้หรือยัง

ขาที่ 5 (Clear To Send) เป็นขาที่ใช้สำหรับรับสัญญาณควบคุมจากอุปกรณ์ใด ๆ เมื่ออุปกรณ์รับกำลังบอกว่าพร้อมที่จะรับสัญญาณข้อมูลแล้ว

ขาที่ 6 (Data Set Ready) เป็นขาที่สัญญาณควบคุมซึ่งเป็นการบอกตัวคอมพิวเตอร์ ว่าตัวอุปกรณ์เรียบร้อยแล้วและพร้อมที่จะส่งสัญญาณข้อมูลได้แล้ว

ขาที่ 7 (Signal Ground) เป็นขาที่ทำหน้าที่เป็นระดับแรงดันสัญญาณอ้างอิง สำหรับทุก ๆ สายของสัญญาณจะมีแรงดันเป็น "0" เมื่อเทียบกับสัญญาณตัวอื่น

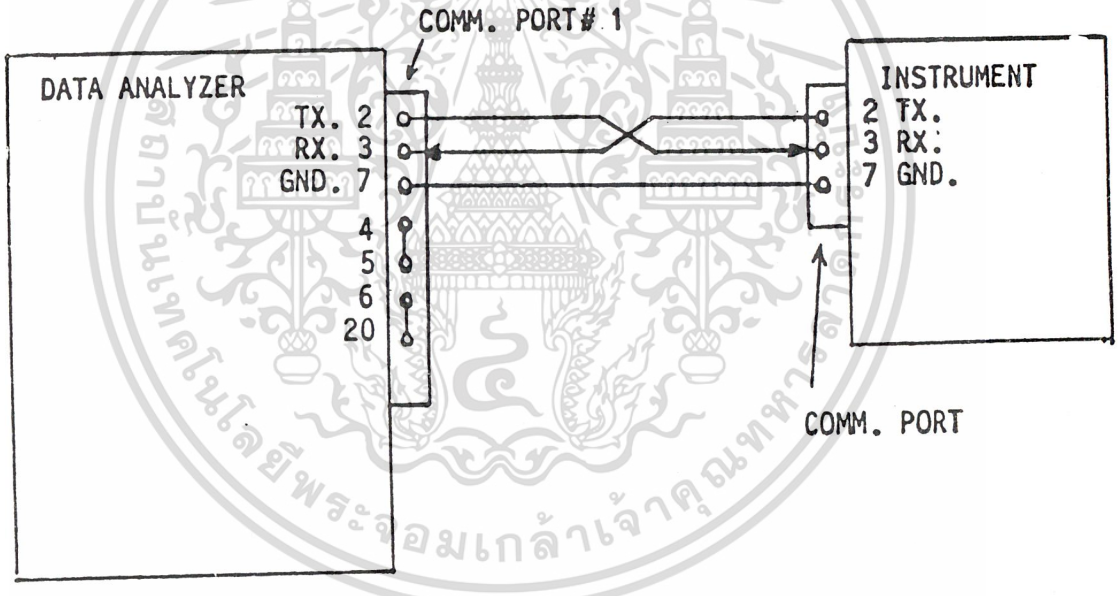
ขาที่ 20(Data Terminal Ready) เป็นขาที่สัญญาณควบคุมซึ่งตัวคอมพิวเตอร์เปิดให้กับอุปกรณ์เมื่อ - พร้อมที่จะติดต่อกับอุปกรณ์ใด ๆ



รูปภาพที่ 2 การกำหนดของข้อต่อแบบ RS-232-C

3. การใช้งาน ไอบีเอ็ม พีซี เป็นดาต้าแอนนะไลเซอร์ และ ผลการทดสอบ

โดยที่เราใช้งาน ไอบีเอ็ม พีซี เป็นดาต้าแอนนะไลเซอร์ เป็นเพียงตัวรับข้อมูลแบบอนุกรมจากอุปกรณ์ที่เราต้องการตรวจสอบและวิเคราะห์เพียงอย่างเดียว ดังนั้นเราจึงกำหนด RS-232-C PIN ASSIGNMENT ที่คอมมูนิเคชั่นพอร์ตที่ 1 (COMMUNICATON PORT # 1) ของ ไอบีเอ็ม พีซี ดังรูปภาพที่ 3 ซึ่งบล็อคไดอะแกรมในรูปภาพที่ 3 นี้เป็นการใช้งานตัวดาต้าแอนนะไลเซอร์ เข้าไปตรวจสอบวิเคราะห์ข้อมูลแบบอนุกรมของอุปกรณ์ใด ๆ ซึ่งในการทดสอบของเราจะใช้ ไอบีเอ็ม พีซี อีกตัวหนึ่งทำตัวเหมือนอุปกรณ์ที่ต้องการให้ตรวจสอบและวิเคราะห์ข้อมูลแบบอนุกรม



รูปภาพที่ 3 แสดงบล็อคไดอะแกรมของการต่อเพื่อใช้งานตรวจสอบของดาต้าแอนนะไลเซอร์

ดังกล่าวมาแล้วว่าการประยุกต์ใช้งานในแบบอนุกรมนี้ เพียงแต่ใช้โปรแกรมเพียงชุดเดียวก็สามารถใช้งานได้ดี ซึ่งโปรแกรมชุดนี้เขียนขึ้นด้วยภาษาเบสิค (BASIC) ซึ่งเป็นภาษาที่เข้าใจง่ายไม่สลับซับซ้อน ซึ่งโปรแกรมชุดนี้มีโฟลว์ชาร์ท ดังแสดงในรูปภาพที่ 4 ดังจะขออธิบายหลักการของโปรแกรมตามโฟลว์ชาร์ทดังนี้

1. เริ่มต้นด้วยการกำหนดค่าเริ่มต้น (INITIALIZE) ของตัวแปร (VARIABLE) และค่าคงที่ (CONSTANT) ต่าง ๆ โดยที่เราจะเก็บข้อมูลที่รับเข้ามาในรูปของ STRING VARIABLE

2. กำหนดค่าเริ่มต้นของคอมมูนิเคชั่นพอร์ต (COMM. PORT) ตั้ง (SET) ตัวซับรูทีน (SUBROUTINE) อินเทอรัพรูทีน (INTERUPT ROUTINE) เมื่อมีข้อมูลเข้ามาที่คอมมูนิเคชั่นบัฟเฟอร์ (COMM. BUFFER)

3. ตั้ง (SET) ตารางของ ASCII & EBCDIC CODE ซึ่งเป็นการตั้งค่าของASCII & EBCDIC CODE ว่าค่า ๆ ตัวนี้มีเป็นตัวอะไรนั่นเอง

4. เป็นการวาดภาพของตาราง MAIN MENU บนจอภาพ ซึ่งเป็นภาพของตารางคำสั่งและข้อมูลที่รับเข้ามา

5. พิมพ์ วันที่, เวลา และจำนวนของข้อมูลที่รับเข้ามา และแปลง CODE เสร็จเรียบร้อยแล้ว

6. รับข้อมูลจากคีย์บอร์ด (KEYBOARD) เก็บไว้ใน STRING VARIABLE ซึ่งเป็นข้อมูลประเภท ฟังก์ชั่นคีย์ (FUNCTION KEY) หรือ คอนโทรลคีย์ (CONTROL KEY)

7. เป็นการตรวจสอบไปตามรูทีน (ROUTINE) ที่เป็น ฟังก์ชั่นคีย์, คอนโทรลคีย์ต่าง ๆ ว่าเป็นประเภทใด

8. ถ้าตรวจสอบว่าเป็นฟังก์ชั่นคีย์ก็จะไปทำรูทีนต่าง ๆ ของฟังก์ชั่นคีย์ ซึ่งมีอยู่ 8 รูทีน คือ QUIT, DATA, SEARCH, START, STOP, TEST, PRINT, และ EXTENDED MENU เมื่อโปรแกรมทำตามรูทีนของฟังก์ชั่นคีย์เสร็จแล้วก็จะกลับไปที่ (ก) เพื่อรับข้อมูลใหม่

9. ถ้าตรวจสอบว่าไม่ใช่ฟังก์ชั่นคีย์ ก็จะตรวจสอบต่อมาว่าเป็นคอนโทรลคีย์ใช่หรือไม่ถ้าใช่ ก็จะไปทำรูทีนต่าง ๆ ของคอนโทรลคีย์ ซึ่งมีอยู่ 4 รูทีน คือ HOME, END, PgUp และ PgDn

และไปแสดงผลบนจอภาพ จากนั้นก็จะไปที่ (ก) เพื่อรอรับข้อมูลใหม่

10. ในกรณีที่ตรวจสอบแล้วว่าไม่มีรูทีนเป็นฟังก์ชั่นคีย์ หรือ คอนโทรลคีย์ก็จะทำในลำดับต่อไป คือ ตรวจสอบว่ามีข้อมูลเพิ่ม และยังไม่ได้แปลงข้อมูลเข้ามาหรือไม่ ซึ่งถ้าไม่มีข้อมูลเพิ่มก็จะกลับไป (ก) เพื่อรอรับข้อมูลใหม่

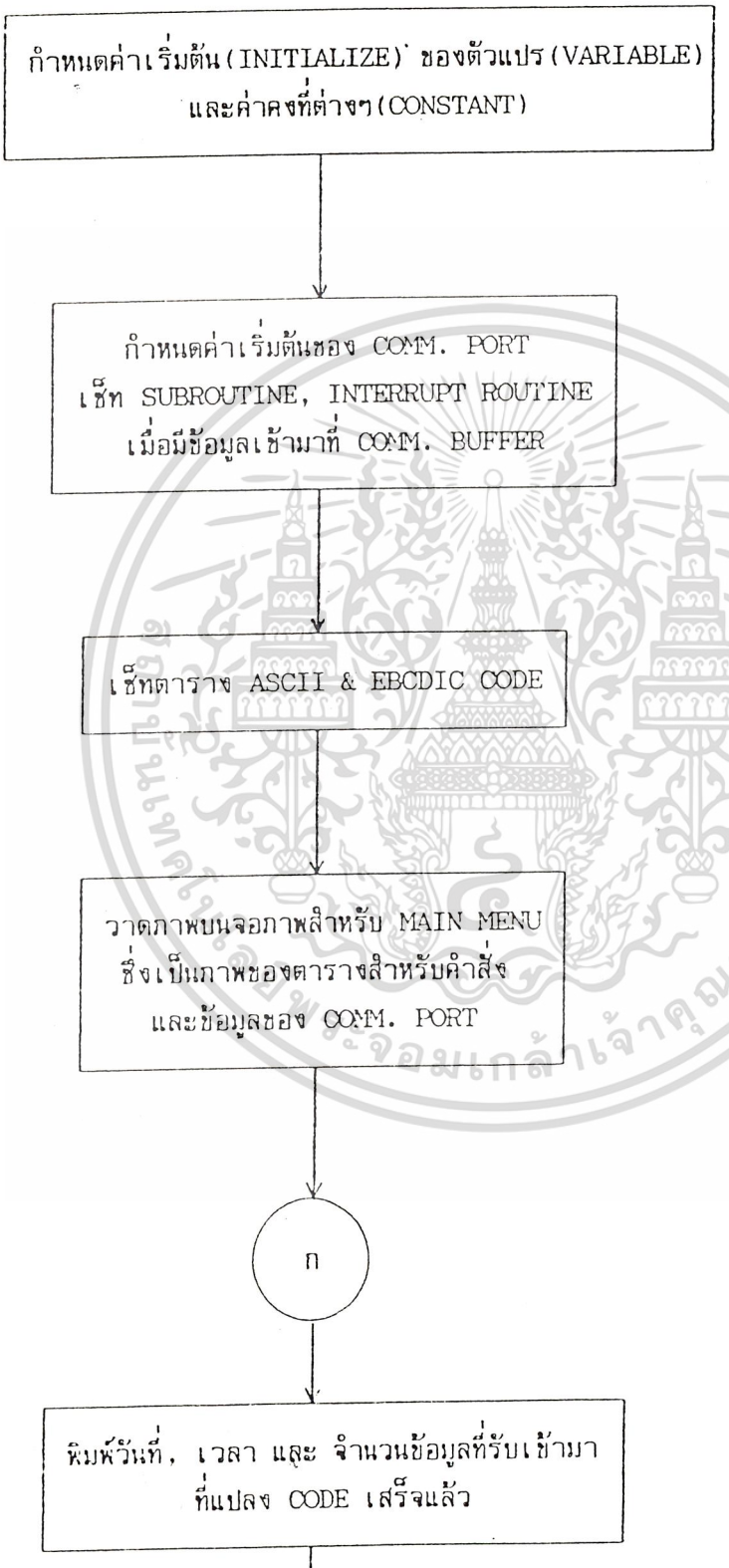
11. แต่ถ้ามีข้อมูลใหม่เพิ่มและยังไม่ได้แปลงข้อมูลเข้ามาก็จะผ่านโปรแกรมที่ทำการแปลงข้อมูลให้อยู่ในรูปของ DEC, HEX, OCTAL, ASCII & EBCDIC CODE และขั้นสุดท้ายก็จะแสดงข้อมูลทั้งหมดให้เป็น BINARY CODE และเก็บข้อมูลที่แปลงได้ไว้ใน STRING VARIABLE ย่อย ๆ

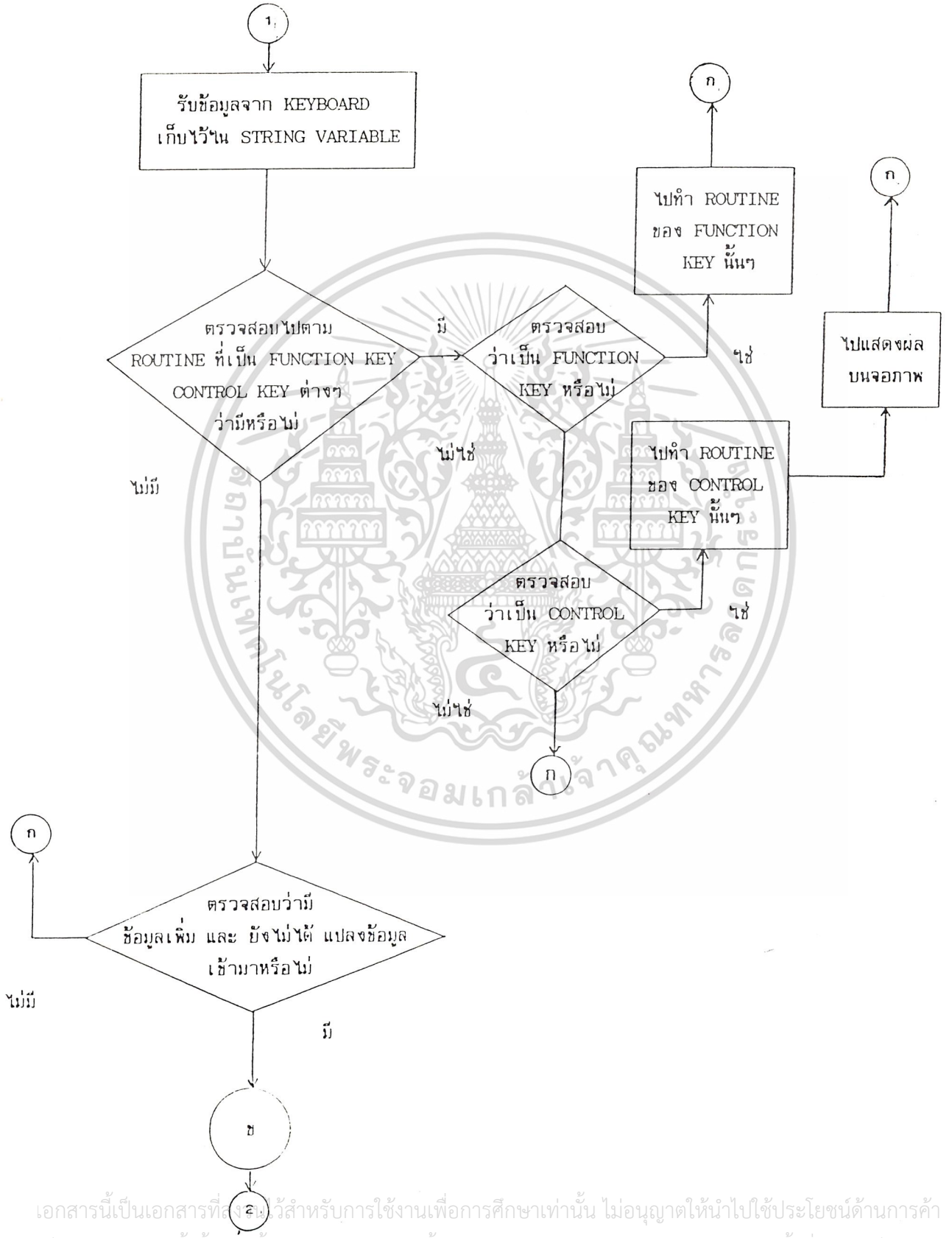
12. ขั้นตอนถัดมาก็จะนำข้อมูลที่แปลงแล้วที่อยู่ใน STRING VARIABLE ย่อย ๆ เหล่านี้ไปเก็บไว้ใน STRING VARIABLE ใหญ่เพื่อจะนำออกไปแสดงผลบนจอภาพต่อไป

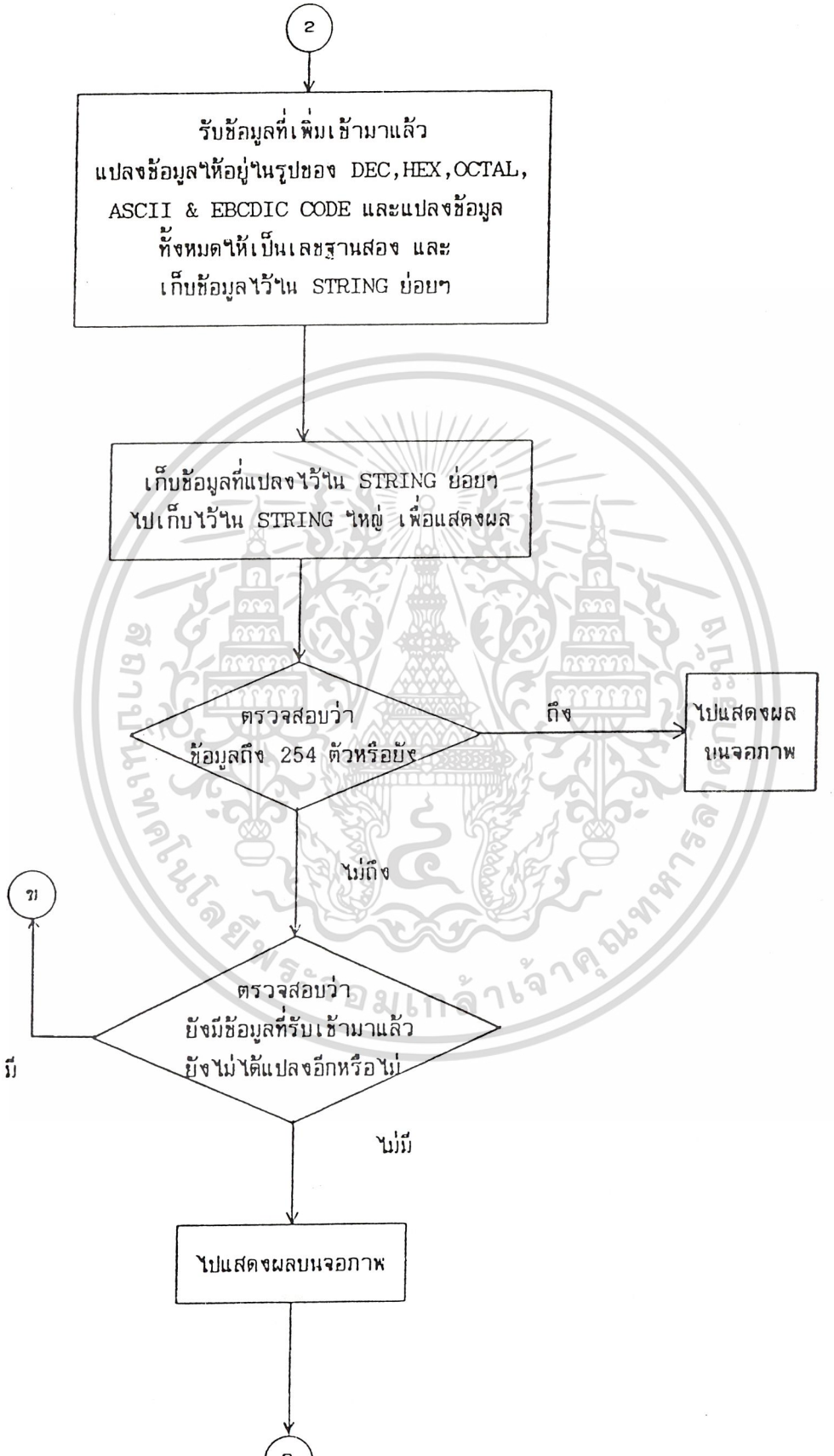
13. ตรวจสอบว่าข้อมูลที่รับเข้าและแปลงข้อมูลเรียบร้อยแล้วมีข้อมูลถึง 254 ตัวหรือยัง ถ้าข้อมูลมีถึง 254 ตัว แล้ว (แสดงว่า STRING VARIABLE ตัวที่ใช้เก็บข้อมูลเต็มแล้ว) ก็จะนำข้อมูลชุดนี้ออกแสดงผลบนจอภาพ

14. ถ้าข้อมูลยังไม่ถึง 254 ตัว ก็จะตรวจสอบว่ายังมีข้อมูลที่รับเข้ามาแล้วยังไม่ได้แปลงอีกหรือไม่ ถ้ามีก็จะกลับไปยัง (ข) เพื่อแปลงข้อมูลเหล่านี้ หรือ ถ้าไม่มีข้อมูลเหล่านี้ก็จะไปแสดงผลบนจอภาพและต่อจากนั้นก็จะกลับไปที่ (ก) เพื่อรอรับข้อมูลใหม่

ซึ่งลำดับขั้นตอนทั้ง 14 ขั้นตอนที่กล่าวมาแล้วนี้ก็คือหลักการของโปรแกรมที่เขียนขึ้นมานั่นเอง







รูปภาพที่ 4 แสดงถึงโฟลว์ชาร์ทของโปรแกรม

โปรแกรมชุดนี้เป็นโปรแกรมภาษาเบสิค (BASIC) ใช้ชื่อว่า TABL.BAS และได้ใช้ BASIC COMPILER ทำให้ได้เป็น TABL.EXE FILE ออกมา เมื่อเรารัน (RUN) โปรแกรม TABL.EXEแล้วจะได้ผลออกมาดังในตารางที่ 1 ซึ่งเป็น SERIAL STATE MENU ออกมา

```
Serial State Menu        <Data Entry in Dec>
Data1 = 000              <Search in Hex> #=> 0
Data2 = 000               Search = XX
NO.  HEX  76543210  DEC  OCT  ASCII  EBCDIC
```

```
07-20-1988  15:10:18
Received =  0    chrs.
Received Error

1  =>  Quit
2  =>  Data=
3  =>  Search
4  =>  Start
5  =>  Stop

7  =>  Test
9  =>  Print
0  =>  Extended Menu

Home => Go To Top
End  => Go To Bottom
PgUp => Next Page
PgDn => Prev. Page
```

ตารางที่ 1 แสดงผลของ SERIAL STATE MENU

จากตารางที่ 1 จะเห็นได้ว่าตารางด้านซ้ายมือ เป็นตารางผลของข้อมูลที่รับเข้ามา ส่วนตารางขวามือเป็นตารางที่ใช้ในการเลือกใช้งาน ในฟังก์ชั่นต่าง ๆ โดยขึ้นอยู่กับหมายเลขที่เลือกใช้งาน ดังนี้ คือ.-

หมายเลขที่ 1. QUIT ในกรณีที่เราจะไม่ใช้งานแล้ว

หมายเลขที่ 2. DATA ในกรณีที่เราต้องการตรวจสอบข้อมูลแบบที่มีเงื่อนไขว่าให้เครื่องคาค้า - แอนนะไลเซอร์จะตรวจสอบข้อมูลและแสดงผลของข้อมูลได้ก็ต่อเมื่อพบข้อมูล DATA 1 และ DATA 2ที่เราตั้งไว้เสียก่อน

หมายเลข 3. SEARCH เป็นการให้หาข้อมูลที่เราสนใจว่าอยู่ที่ใดบ้าง และมีกี่ตัวในข้อมูลที่กำลังตรวจสอบอยู่ โดยเครื่องคาค้าแอนนะไลเซอร์จะใส่เครื่องหมายปีกกากำกับบอกกับเราว่าข้อมูลนั้น ๆ อยู่ที่ใดและมีจำนวนกี่ตัว

หมายเลข 4. START ใช้ในกรณีให้เริ่มตรวจสอบข้อมูลที่รับเข้ามา

หมายเลข 5. STOP หยุดตรวจสอบข้อมูล

หมายเลข 7. TEST เป็นฟังก์ชั่นที่ใช้ทดสอบว่าข้อมูลที่มีค่า 0-254 นั่นเป็นอย่างไรในรูปแบบของ HEX CODE, BINARY CODE, DECIMAL, OCTAL, ASCII CODE, และ EBCDIC CODE ตามลำดับทั้งประโยชน์ก็เป็น TEST FUNCTION และยังเป็นตารางมาตรฐานเพื่อเทียบค่าต่าง ๆ ที่กล่าวมาแล้ว โดยไม่ต้องไปค้นหาที่อื่น

หมายเลข 9. PRINT ให้พิมพ์ข้อมูลที่แสดงผลออกมาที่เครื่องพิมพ์

หมายเลข 0. EXTENED MENU เป็นเมนูซึ่งให้เราเลือกรูปแบบของข้อมูลที่เราสนใจ เช่น BAUD RATE, WORD SIZE, STOP BITS และอื่น ๆ ดังแสดงในตารางที่ 2

```
Extended Serial Interface Menu                                  Comm Port#1

 Baud rate:      75   110   150   300   600   1200   2400   4800   9600
 Word size:      5    6     7     8
 Stop bits:      1    1½    2
 Parity:         No parity   Odd    Even
 Input logic:    Negative    Positive
 Search format:  Hex   Dec   Oct    Bin

Cursor movement keys are as shown on the keypad.  You may only use the 8, 2, 4,
and 6 keys for cursor movement.  (Note:  Keypad should be in numeric mode.)
          ↑
          8                  When you have finished,
      ←6  4→                 press the RETURN key
          2
          ↓
```

ตารางที่ 2 แสดงผลของ EXTENED MENU

ตารางที่ 3. แสดงผลของการรับข้อมูลที่ทดสอบส่งมาจาก ไอบีเอ็ม พีซี อีกตัวหนึ่ง(กำหนดให้เป็นเสมือนตัวอุปกรณ์ที่ต้องการทดสอบ) มายังเครื่องดาต้าแอนนะไลเซอร์ โดยการส่งข้อมูลในรูปของ QUICK BROWN FOX JUMP OVER THE LAZY DOS'S BACK 1234567890 ซึ่งเป็น ASCII CODE เป็นข้อมูลทดสอบว่าผลการตรวจสอบของเครื่องดาต้าแอนนะไลเซอร์เป็นอย่างไร ซึ่งผลการทดสอบให้เห็นว่าสามารถตรวจรับได้ทุกตัวอักษร (โดยการกดฟังก์ชั่นหมายเลข 4 START เมื่อเริ่มรับข้อมูล และกดฟังก์ชั่นหมายเลข 5 STOP เมื่อต้องบอกให้เครื่องหยุดรับข้อมูล)

ตารางที่ 4. เมื่อเครื่องดาต้าแอนนะไลเซอร์แสดงผลของการตรวจสอบข้อมูลได้ดังในตารางที่ 3 แล้ว ลองให้เครื่องวิเคราะห์หาข้อมูลที่มีค่า 61 (HEX) ดูว่าเป็นอย่างไร ซึ่งก็สามารถวิเคราะห์หาได้ดังในตารางที่ 4 นี้ ซึ่งมีข้อมูล 61 (HEX) อยู่ 2 ตัว และเมื่อพบข้อมูลที่ต้องการหาแล้วจะใส่เครื่องหมายปีกกาให้ เพื่อเป็นตัวบ่งชี้ให้เห็นว่าอยู่ที่ใดนั่นเอง(โดยการกดฟังก์ชั่นหมายเลข 3 - SEARCH และกดค่า 61 (HEX) เพื่อบอกเครื่องดาต้าถอนนะไลเซอร์ให้หาข้อมูล 61 (HEX))

ตารางที่ 5 แสดงถึงการทดสอบในกรณีรับข้อมูลแบบที่มีเงื่อนไข ว่าจะต้องตรวจสอบข้อมูลตัวที่ 1 (DATA 1 = 102) ซึ่งเป็น 102 (DEC) และข้อมูลตัวที่ 2 (DATA 2 = 111)ซึ่งเป็น 111 (DEC) เสียก่อนจึงจะตรวจรับข้อมูลตัวถัดไปเข้ามา ซึ่งประโยชน์ใช้ในการส่งข้อมูลที่เรารู้ว่ามี SYNC. WORD หรือ START WORD ว่ามีค่าเป็นอะไร (กดฟังก์ชั่นหมายเลข 2 DATA และตั้งข้อมูลที่เป็นเงื่อนไขทั้งสองตัว)

ตารางที่ 6 แสดงถึงการทดสอบที่ต่อเนื่องจากตารางที่ 5 โดยกรณีรับข้อมูลอย่างมีเงื่อนไขมาได้แล้ว และให้เครื่องดาต้าแอนนะไลเซอร์ตรวจสอบข้อมูลค่า 6F (HEX) ดู ซึ่งเครื่องสามารถตรวจตรวจพบแล้วก็ใส่เครื่องหมายปีกกา เพื่อเป็นตัวบ่งชี้ให้เห็น ดังในตารางนี้ (กดฟังก์ชั่นหมายเลข 3 SEARCH และกดข้อมูลค่า 6F (HEX) ต่อมา)

4. สรุปและวิจารณ์

การประยุกต์ให้ ไอบีเอ็ม พีซี มาใช้งานในหน้าที่ของดาต้า แอนนะไลเซอร์ ในแบบอนุกรมนี้ได้ประโยชน์อย่างยิ่ง และยังสามารถเพิ่มฟังก์ชั่นของการใช้งานออกไปได้อีก โดยอาศัยการเขียนโปร-

แกรมเพิ่มเติมเข้าไป ส่วนในกรณีที่จะใช้ตรวจสอบข้อมูลแบบขนานนั้น จำเป็นที่จะต้องมีการสร้างฮาร์ด-แวร์ (HARDWARE) และเขียนโปรแกรมเพิ่มเติมขึ้นมา ซึ่งอยู่ในระหว่างการค้นคว้าเพิ่มเติมต่อไป

เอกสารอ้างอิง

1. Technical Reference, IBM personal computer XT hardware reference library, revised edition, IBM corp,1983
2. Lewis C. Eggebrecht, Interfacing to the IBM personal computer, Howard W. Sams & Co.,In., Indiana, 1983.
3. Lyle J. Graham, Your IBM PC, Osborne / Mc Graw-Hill 2600 Tenth Street Berkeley, California 94710.

ตารางที่ 3

Serial State Menu <Data Entry in Dec>

Data1 = 000 <Search in Hex> #=> 0
Data2 = 000 Search = XX

NO.	HEX	76543210	DEC	OCT	ASCII	EBCDIC
1	71	01110001	113	161	q	
2	75	01110101	117	165	u	
3	69	01101001	105	151	i	
4	63	01100011	099	143	c	
5	6B	01101011	107	153	k	
6	20	00100000	032	040	SP	DS
7	62	01100010	098	142	b	
8	72	01110010	114	162	r	
9	6F	01101111	111	157	o	?
10	77	01110111	119	167	w	
11	6E	01101110	110	156	n	>
12	20	00100000	032	040	SP	DS
13	66	01100110	102	146	f	
14	6F	01101111	111	157	o	?

07-28-1988 14:36:49

Received = 57 chrs.
Received Error

1 => Quit
2 => Data=
3 => Search
4 => Start
5 => Stop

7 => Test
9 => Print
0 => Extended Menu

Home => Go To Top
End => Go To Bottom
PgUp => Next Page
PgDn => Prev. Page

Serial State Menu <Data Entry in Dec>

Data1 = 000 <Search in Hex> #=> 0
Data2 = 000 Search = XX

NO.	HEX	76543210	DEC	OCT	ASCII	EBCDIC
14	6F	01101111	111	157	o	?
15	78	01111000	120	170	x	
16	20	00100000	032	040	SP	DS
17	6A	01101010	106	152	j	
18	75	01110101	117	165	u	
19	6D	01101101	109	155	m	-
20	70	01110000	112	160	p	
21	20	00100000	032	040	SP	DS
22	6F	01101111	111	157	o	?
23	76	01110110	118	166	v	
24	65	01100101	101	145	e	
25	72	01110010	114	162	r	
26	20	00100000	032	040	SP	DS
27	74	01110100	116	164	t	

07-28-1988 14:38:28

Received = 57 chrs.
Received Error

1 => Quit
2 => Data=
3 => Search
4 => Start
5 => Stop

7 => Test
9 => Print
0 => Extended Menu

Home => Go To Top
End => Go To Bottom
PgUp => Next Page
PgDn => Prev. Page

Serial State Menu <Data Entry in Dec>

Data1 = 000 <Search in Hex> #=> 0
Data2 = 000 Search = XX

NO.	HEX	76543210	DEC	OCT	ASCII	EBCDIC
27	74	01110100	116	164	t	
28	68	01101000	104	150	h	
29	65	01100101	101	145	e	
30	20	00100000	032	040	SP	DS
31	6C	01101100	108	154	l	%
32	61	01100001	097	141	a	/
33	7A	01111010	122	172	z	:
34	79	01111001	121	171	y	
35	20	00100000	032	040	SP	DS
36	64	01100100	100	144	d	
37	6F	01101111	111	157	o	?
38	67	01100111	103	147	g	
39	27	00100111	039	047	'	PRE
40	73	01110011	115	163	s	

07-28-1988 14:40:21

Received = 57 chrs.
Received Error

1 => Quit
2 => Data=
3 => Search
4 => Start
5 => Stop

7 => Test
9 => Print
0 => Extended Menu

Home => Go To Top
End => Go To Bottom
PgUp => Next Page
PgDn => Prev. Page

Serial State Menu <Data Entry in Dec>

Data1 = 000 <Search in Hex> #=> 0
Data2 = 000 Search = XX

NO.	HEX	76543210	DEC	OCT	ASCII	EBCDIC
40	73	01110011	115	163	s	
41	20	00100000	032	040	SP	DS
42	62	01100010	098	142	b	
43	61	01100001	097	141	a	
44	63	01100011	099	143	c	
45	6B	01101011	107	153	k	
46	20	00100000	032	040	SP	DS
47	31	00110001	049	061	1	
48	32	00110010	050	062	2	SYN
49	33	00110011	051	063	3	
50	34	00110100	052	064	4	PN
51	35	00110101	053	065	5	RS
52	36	00110110	054	066	6	UC
53	37	00110111	055	067	7	EOT

07-28-1988 14:42:05

Received = 57 chrs.
Received Error

1 => Quit
2 => Data=
3 => Search
4 => Start
5 => Stop

7 => Test
9 => Print
0 => Extended Menu

Home => Go To Top
End => Go To Bottom
PgUp => Next Page
PgDn => Prev. Page

```
Serial State Menu        <Data Entry in Dec>

Data1 = 000                <Search in Hex> #=> 0
Data2 = 000                 Search = XX

NO.   HEX   76543210   DEC   OCT   ASCII   EBCDIC

53    37    00110111   055   067   7       EOT
54    38    00111000   056   070   8
55    39    00111001   057   071   9
56    30    00110000   048   060   0
57    0D    00001101   013   015     CR    CR
```

```
07-28-1988  14:43:58

Received =  57   chrs.
Received Error


1  =>  Quit
2  =>  Data=
3  =>  Search
4  =>  Start
5  =>  Stop

7  =>  Test
9  =>  Print
0  =>  Extended Menu

Home => Go To Top
End  => Go To Bottom
PgUp => Next Page
PgDn => Prev. Page
```

```
Serial State Menu      <Data Entry in Dec>
Data1 = 000                <Search in Hex> #=> 2
Data2 = 000                 Search = 61

NO.  HEX  76543210  DEC  OCT  ASCII  EBCDIC

27   74   01110100  116  164  t
28   68   01101000  104  150  h
29   65   01100101  101  145  e
30   20   00100000  032  040     SP    DS
31   6C   01101100  108  154  l        %
32  (61)  01100001  097  141  a        /
33   7A   01111010  122  172  z        :
34   79   01111001  121  171  y
35   20   00100000  032  040     SP    DS
36   64   01100100  100  144  d
37   6F   01101111  111  157  o        ?
38   67   01100111  103  147  g
39   27   00100111  039  047           PRE
40   73   01110011  115  163  s
```

```
07-28-1988  14:46:01
Received =  57   chrs.
Received Error


1  =>  Quit
2  =>  Data=
3  =>  Search
4  =>  Start
5  =>  Stop

7  =>  Test
9  =>  Print
0  =>  Extended Menu

Home =>  Go To Top
End  =>  Go To Bottom
PgUp =>  Next Page
PgDn =>  Prev. Page
```

```
Serial State Menu      <Data Entry in Dec>
Data1 = 000                <Search in Hex> #=> 2
Data2 = 000                 Search = 61

NO.  HEX  76543210  DEC  OCT  ASCII  EBCDIC

40   73   01110011  115  163  s
41   20   00100000  032  040     SP    DS
42   62   01100010  098  142  b
43  (61)  01100001  097  141  a        /
44   63   01100011  099  143  c
45   6B   01101011  107  153  k
46   20   00100000  032  040     SP    DS
47   31   00110001  049  061  1
48   32   00110010  050  062  2        SYN
49   33   00110011  051  063  3
50   34   00110100  052  064  4        PN
51   35   00110101  053  065  5        RS
52   36   00110110  054  066  6        UC
53   37   00110111  055  067  7        EOT
```

```
07-28-1988  14:48:00
Received =  57   chrs.
Received Error


1  =>  Quit
2  =>  Data=
3  =>  Search
4  =>  Start
5  =>  Stop

7  =>  Test
9  =>  Print
0  =>  Extended Menu

Home =>  Go To Top
End  =>  Go To Bottom
PgUp =>  Next Page
PgDn =>  Prev. Page
```

ตารางที่ 4

ตารางที่ 5

```
Serial State Menu          <Data Entry in Dec>
Data1 = 102                <Search in Hex> #=> 2
Data2 = 111                 Search = 61

NO.   HEX   76543210   DEC   OCT   ASCII   EBCDIC

1     66    01100110   102   146   f
2     6F    01101111   111   157   o         ?
3     78    01111000   120   170   x
4     20    00100000   032   040     SP      DS
5     6A    01101010   106   152   j
6     75    01110101   117   165   u
7     6D    01101101   109   155   m         -
8     70    01110000   112   160   p
9     20    00100000   032   040     SP      DS
10    6F    01101111   111   157   o         ?
11    76    01110110   118   166   v
12    65    01100101   101   145   e
13    72    01110010   114   162   r
14    20    00100000   032   040     SP      DS
```

```
07-28-1983  14:51:56

Received =  45    chrs.
Received Error


1  =>  Quit
2  =>  Data=
3  =>  Search
4  =>  Start
5  =>  Stop

7  =>  Test
9  =>  Print
0  =>  Extended Menu

Home =>  Go To Top
End  =>  Go To Bottom
PgUp =>  Next Page
PgDn =>  Prev. Page
```

```
Serial State Menu          <Data Entry in Dec>
Data1 = 102                <Search in Hex> #=>0
Data2 = 111                 Search = XX

NO.   HEX   76543210   DEC   OCT   ASCII   EBCDIC

1     66    01100110   102   146   f
2     6F    01101111   111   157   o         ?
3     78    01111000   120   170   x
4     20    00100000   032   040     SP      DS
5     6A    01101010   106   152   j
6     75    01110101   117   165   u
7     6D    01101101   109   155   m         -
8     70    01110000   112   160   p
9     20    00100000   032   040     SP      DS
10    6F    01101111   111   157   o         ?
11    76    01110110   118   166   v
12    65    01100101   101   145   e
13    72    01110010   114   162   r
14    20    00100000   032   040     SP      DS
```

```
07-28-1983  14:53:51

Received =  45    chrs.
Received Error


1  =>  Quit
2  =>  Data=
3  =>  Search
4  =>  Start
5  =>  Stop

7  =>  Test
9  =>  Print
0  =>  Extended Menu

Home =>  Go To Top
End  =>  Go To Bottom
PgUp =>  Next Page
PgDn =>  Prev. Page
```

Serial State Menu 07-28-1988 14:56:23

Data1 = 102 Received 45 chrs.
Data2 = 111 Search = 6F

NO.	HEX	76543210	DEC	OCT	ASCII	EBCDIC
1	66	01100110	102	146	f	
2	(6F)	01101111	111	157	o	?
3	78	01111000	120	170	x	
4	20	00100000	032	040	SP	DS
5	6A	01101010	106	152	j	
6	75	01110101	117	165	u	
7	6D	01101101	109	155	m	-
8	70	01110000	112	160	p	
9	20	00100000	032	040	SP	DS
10	(6F)	01101111	111	157	o	?
11	76	01110110	118	166	v	
12	65	01100101	101	145	e	
13	72	01110010	114	162	r	
14	20	00100000	032	040	SP	DS
15	74	01110100	116	164	t	
16	68	01101000	104	150	h	
17	65	01100101	101	145	e	
18	20	00100000	032	040	SP	DS
19	6C	01101100	108	154	l	%
20	61	01100001	097	141	a	/
21	7A	01111010	122	172	z	
22	79	01111001	121	171	y	
23	20	00100000	032	040	SP	DS
24	64	01100100	100	144	d	
25	(6F)	01101111	111	157	o	?
26	67	01100111	103	147	g	
27	27	00100111	039	047		PRE
28	73	01110011	115	163	s	
29	20	00100000	032	040	SP	DS
30	62	01100010	098	142	b	
31	61	01100001	097	141	a	/
32	63	01100011	099	143	c	
33	6B	01101011	107	153	k	
34	20	00100000	032	040	SP	DS
35	31	00110001	049	061	1	
36	32	00110010	050	062	2	SYN
37	33	00110011	051	063	3	
38	34	00110100	052	064	4	PN
39	35	00110101	053	065	5	RS
40	36	00110110	054	066	6	UC
41	37	00110111	055	067	7	EOT
42	38	00111000	056	070	8	
43	39	00111001	057	071	9	
44	30	00110000	048	060	0	
45	0D	00001101	013	015	CR	CR

ตารางที่ 6