8051 เรียลไทม์ อิน - เซอร์กิต อีมูเลเตอร์

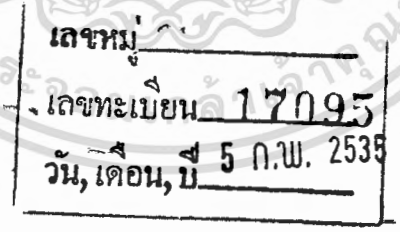
8051 Real Time In - circuit Emulator

ชรินทร์ บัณฑุกุล

CHARIN BANTUKUL

อาจารย์ที่ปรึกษา

รองศาสตราจารย์ ดร. จเร สุรวัฒน์ปัญญา



วิทยานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรมหาบัณฑิต

ภาควิชาวิศวกรรมโทรคมนาคม

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหาร ลาดกระบัง

ปีการศึกษา 2534

บทคัดย่อ

วิทยานิพนธ์นี้เป็นการเสนอแนวทางการพัฒนาอุปกรณ์ที่เรียกว่า เรียลไทม์ อิน-เซอร์กิต อีมูเลเตอร์ (Real Time In-circuit Emulator) ของไมโครคอมพิวเตอร์แบบชิพเดียว ตระกูล 8051 ขึ้นมาใช้งาน ตัวอีมูเลเตอร์มักถูกใช้เป็นเครื่องมือ สำหรับช่วยตรวจสอบหรือแก้ไขระบบไมโครคอมพิวเตอร์ที่มีข้อผิดพลาดขึ้นทางซอฟต์แวร์ ทั้งนี้เนื่องจากอีมูเลเตอร์สามารถนำเอาข้อมูลต่างๆ ภายในระบบคอมพิวเตอร์ที่บกพร่องขึ้นมาตรวจสอบได้ เช่น ข้อมูลภายในรีจิสเตอร์ ข้อมูลที่เก็บไว้ในหน่วยความจำ หรือข้อมูลที่รับเข้ามาจากนอกระบบ เป็นต้น และเมื่อผู้ใช้สามารถตรวจสอบข้อมูลได้อย่างถูกต้อง การวิเคราะห์ปัญหาที่เกิดขึ้นจะทำได้อย่างรวดเร็วและแม่นยำ ทำให้ระยะเวลาที่ใช้ในการแก้ปัญหาน้อยลงจากเดิมมาก

ผลการทดสอบการใช้งานปรากฏว่า 8051 อีมูเลเตอร์ที่สร้างขึ้น มีประสิทธิภาพทัดเทียมกับอีมูเลเตอร์ราคาแพงที่มีขายอยู่ในปัจจุบัน แต่มีราคาถูกกว่ามาก

ABSTRACT

This paper presents the development of Real Time In-circuit Emulator for Single chip microcomputer in 8051 family. An emulator is frequently used for testing and solving trouble shoot in microcomputer system having software error. An emulator can take some data from failure computer system such as data in registers, data in memory or input data from ex--ternal system to check and correct. After the user get the correct data, analysis of problem can be easily done and the problem solving time can be greatly reduced.

Experimental results show that the developed 8051 emulator has perfor--mances comparable to the commercially available emulator at present. But the developed one has much less cost.

กิตติกรรมประกาศ

ผู้จัดทำวิทยานิพนธ์ใคร่ขอขอบคุณเป็นอย่างสูงต่อ รองศาสตราจารย์ ดร.จเร สุรวัฒน์ปัญญา อาจารย์ที่ปรึกษา ผู้ให้แนวความคิด และคำปรึกษาตั้งแต่ผู้ทำวิทยานิพนธ์มีความคิดคร่าวๆ จนกระทั่งวิทยานิพนธ์ฉบับนี้เสร็จสมบูรณ์

สุดท้ายนี้ขอขอบคุณ คุณพนม เพชรจตุพร ที่ได้ให้ความช่วยเหลือในการตรวจสอบและแก้ไขทั้งในด้านโปรแกรมและวงจรต่างๆ เพื่อให้วิทยานิพนธ์ฉบับนี้สมบูรณ์ยิ่งขึ้น

สารบัญ

บทที่ 1 บทนำ

1.1 กล่าวนำ

ปัจจุบันนี้มีการนำเอาระบบไมโครคอมพิวเตอร์มาเป็นส่วนสำคัญสำหรับควบคุมเครื่องใช้ไฟฟ้าต่างๆ อย่างแพร่หลาย เช่น อุปกรณ์ควบคุมการทำงานอัตโนมัติในโรงงานอุตสาหกรรม หรือแม้กระทั่งอุปกรณ์ที่ใช้ในสำนักงานก็พบได้หลายชนิด เครื่องใช้ไฟฟ้าดังกล่าวมีขนาดเล็กลงและมีการทำงานสลับซับซ้อนมากขึ้น สาเหตุที่เป็นเช่นนี้เนื่องมาจากผู้ผลิตมักจะนำไมโครคอมพิวเตอร์แบบชิพเดียวมาใช้งานเป็นจำนวนมาก

ไมโครคอมพิวเตอร์แบบชิพเดียว (Single chip Microcomputer) จะมีข้อดีกว่าระบบไมโครคอมพิวเตอร์ทั่วๆ ไป คือ อุปกรณ์สนับสนุนต่างๆ ที่ต้องใช้งานร่วมกับไมโครโปรเซสเซอร์จะอยู่ภายในชิพเดียวกันกับไมโครโปรเซสเซอร์ อุปกรณ์เหล่านี้ได้แก่ หน่วยความจำทั้งชนิดแรมและรอม พอร์ทแบบขนานและพอร์ทแบบอนุกรม ด้วยจุดเด่นข้อนี้ไมโครคอมพิวเตอร์แบบชิพเดียวจึงมักถูกนำมาใช้งานควบคุมที่มีโปรแกรมไม่สลับซับซ้อนมากนัก แต่ต้องการความเร็วในการทำงานสูงพอสมควร

เมื่อพิจารณาในอีกด้านหนึ่งจะเห็นว่าการใช้ไมโครคอมพิวเตอร์เป็นส่วนควบคุมหลักแล้ว ในกรณีที่อุปกรณ์นั้นขัดข้องหรือมีจุดผิดพลาดในโปรแกรมที่ใช้งาน การตรวจสอบแก้ไขจะมีขั้นตอนที่ยุ่งยากโดยเริ่มจากการตรวจหาจุดผิดพลาดของฮาร์ดแวร์ก่อน โดยอาศัยเครื่องมือวัด เช่น ลอจิกโปรบหรือออสซิลโลสโคปตรวจหาจุดเสียที่เกิดขึ้นตามแผนผังของวงจรที่มีอยู่ ถ้าพบว่าอุปกรณ์ใดทำงานบกพร่องก็ทำการเปลี่ยนใหม่ แต่ปัญหาจะยุ่งยากมากถ้าการผิดพลาดที่เกิดขึ้นนั้นมีสาเหตุมาจากซอฟท์แวร์เพราะจะทำให้สัญญาณลอจิกต่างๆ ที่ป้อนเข้าไปควบคุมหรือกระตุ้นให้ฮาร์ดแวร์ทำงานจะผิดพลาดหมด การแก้ไขโดยใช้เครื่องมือทางด้านฮาร์ดแวร์ที่กล่าวผ่านมาช้างต้นทำได้ยากมาก ทั้งนี้เนื่องจากเมื่อไมโครคอมพิวเตอร์ทำการรันโปรแกรมจะเสมือนกับว่าตัวของมันอยู่ในโลกปิดเลยทีเดียว ที่เป็นเช่นนี้เพราะว่าในขณะหนึ่งๆ จะไม่สามารถทราบได้เลยว่าไมโครโปรเซสเซอร์กำลังทำตามคำสั่งที่ถูกบันทึกไว้ในหน่วยความจำคำสั่งใด เกิดข้อผิดพลาดอะไร ค่าผลลัพธ์ที่ได้ผิดพลาดไปมากน้อยเพียงใด ด้วยเหตุนี้จึงมีการนำเทคนิคของการอีมูเลท (Emulation Techniques) [1] มาใช้งานโดยอาศัยหลักการง่ายๆ คือ ทำการสร้างไมโครคอมพิวเตอร์ขึ้นมาอีกชุดหนึ่งที่ทำงานเรียนแบบไมโครโปรเซสเซอร์ตัวนั้นได้ทุกประการ แต่ในขณะเดียวกันผู้ใช้งานสามารถตรวจสอบการทำงานต่างๆ ของโปรแกรมได้ตลอดเวลา

1.2 <u>วัตถุประสงค์และขอบเขตของวิทยานิพนธ์</u>

วัตถุประสงค์ของวิทยานิพนธ์นี้เพื่อต้องการพัฒนาอุปกรณ์ที่เรียกว่า เรียลไทม์ อิน-เซอร์กิต อีมูเลเตอร์ ของไมโครคอมพิวเตอร์แบบชิพเดียวตระกูล 8051 ที่มีราคาถูกขึ้นมา เพื่อใช้เป็นเครื่องมือสำหรับตรวจสอบหรือแก้ไข รวมถึงพัฒนาซอฟต์แวร์ของระบบไมโครคอมพิวเตอร์แบบชิพเดียวตระกูล 8051

อีมูเลเตอร์ที่พัฒนานี้มีคุณสมบัติพิเศษกว่าอีมูเลเตอร์ราคาแพงที่มีขายอยู่ในปัจจุบันคือ มีส่วนตรวจจับข้อมูล (Data Capture) ที่สามารถตรวจจับข้อมูลจากบัสข้อมูลของระบบเอง และยังสามารถประยุกต์ใช้ในการตรวจจับข้อมูลจากภายนอกได้อีกด้วย คุณสมบัติพิเศษนี้สามารถนำไปใช้วิเคราะห์ข้อผิดพลาดของระบบที่เกิดจากฮาร์ดแวร์ได้อีกด้วย

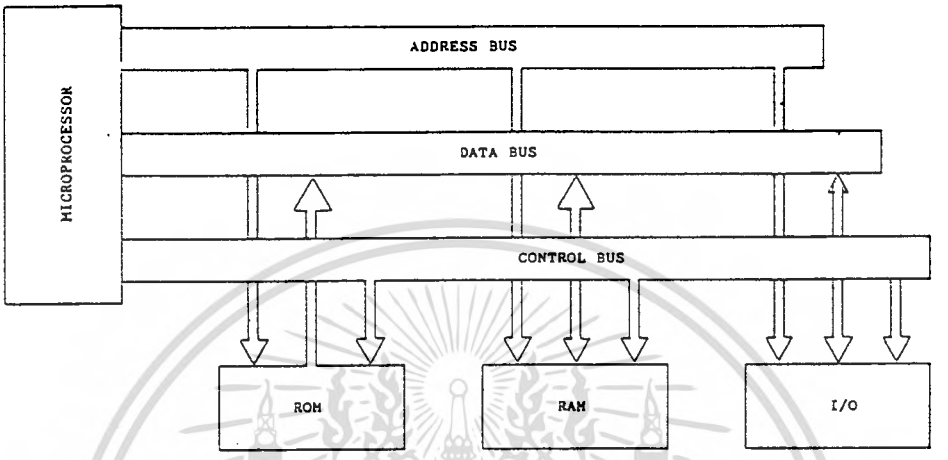
อีมูเลเตอร์ที่พัฒนาขึ้นนี้มีขอบเขตในการพัฒนา 2 ส่วนด้วยกันคือ

1. วงจร ประกอบด้วยวงจรของอีมูเลเตอร์และส่วนตรวจจับข้อมูลซึ่งทำงานร่วมกับอีมูเลเตอร์
2 โปรแกรมควบคุมระบบของอีมูเลเตอร์

โดยที่อีมูเลเตอร์นี้จะอาศัยคอมพิวเตอร์ส่วนบุคคล (IBM-PC) เป็นเทอร์มินอล

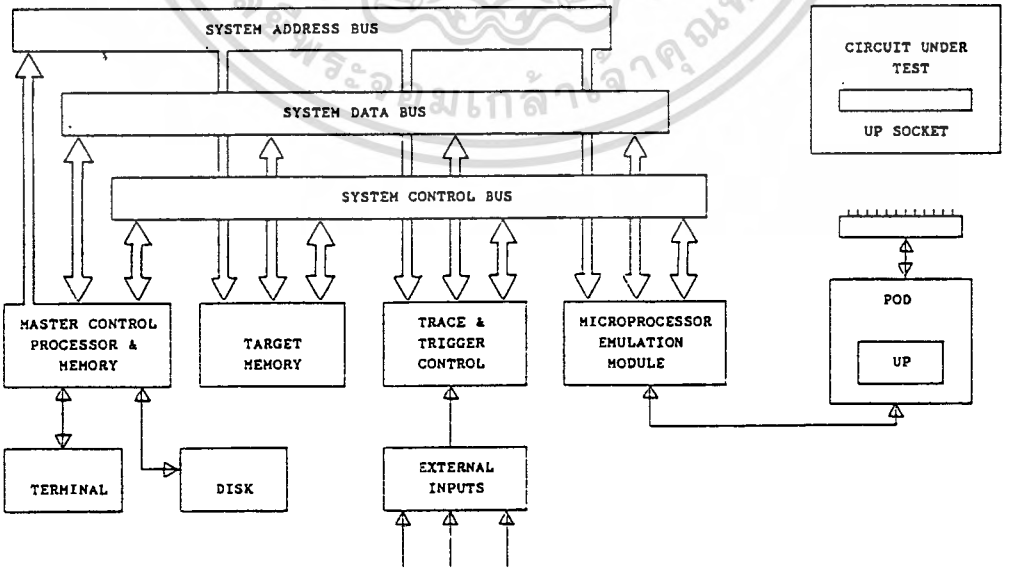
1.3 โครงสร้างและหลักการทำงานทั่วไปของอีมูเลเตอร์

ระบบไมโครคอมพิวเตอร์โดยทั่วไปจะประกอบด้วยบัสสามชนิดด้วยกันคือ แอดเดรสบัส ดาต้าบัส และคอนโทรลบัส [2] โดยบัสทั้งสามจะเชื่อมต่อกับอุปกรณ์ต่างๆ ดังรูปที่ 1.1



รูปที่ 1.1 แสดงระบบพื้นฐานของไมโครคอมพิวเตอร์ทั่วไป

จากรูปที่ 1.1 ถ้านำสัญญาณจากบัสต่างๆ ออกมาภายนอกระบบ และสามารถควบคุมสัญญาณเหล่านี้ได้โดยผ่านทางแป้นพิมพ์และจอภาพ จะสามารถทราบถึงการทำงานของโปรแกรมรวมถึงข้อมูลที่นำเข้าหรือส่งออกไปภายนอกระบบเพื่อควบคุมอุปกรณ์ทางด้านฮาร์ดแวร์ดังกล่าวมาแล้วข้างต้น



รูปที่ 1.2 แสดงแผนผังของการอีมูเลทเข้าไปในแผ่นวงจรที่มีไมโครคอมพิวเตอร์เป็นส่วนควบคุมหลัก (Circuit Under Test)

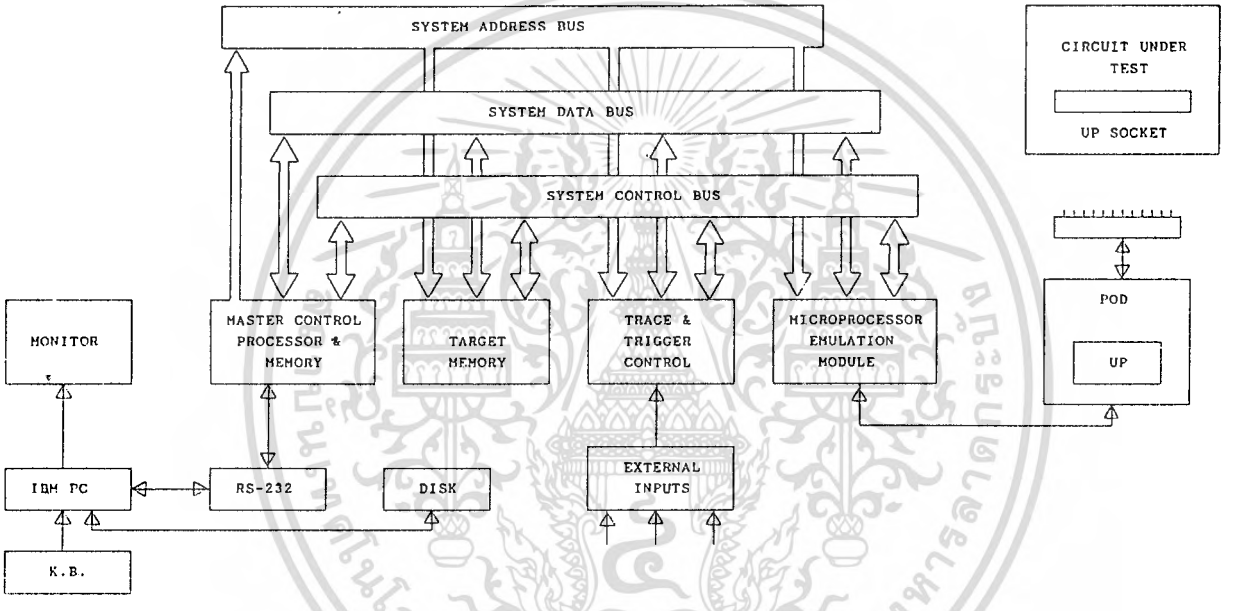
จากรูปที่ 1.2 สมมุติว่าแผ่นวงจรที่จะนำมาตรวจสอบมีโครงสร้างดังที่แสดงไว้ใน รูปที่ 1.1 โดยถอดชิพไมโครโปรเซสเซอร์ออกแล้วใส่สายเชื่อมต่อจากพ็อต (Pod)เข้าไปแทน ดังนั้นสายบัสต่างๆ ของแผ่นวงจรจะถูกเชื่อมต่อเข้าไปในระบบของอีมูเลเตอร์ (Emulator) โดยผ่านทางพ็อตและไมโครโปรเซสเซอร์อีมูเลชั่นโมดูล (Microprocessor Emulation Module) สำหรับอีมูเลเตอร์ทั่วๆ ไปแล้ว ไมโครโปรเซสเซอร์อีมูเลชั่นโมดูล นี้จะเปลี่ยนแปลงไปตามไมโครโปรเซสเซอร์ของแผ่นวงจรที่นำมาตรวจสอบ โมดูลเหล่านี้จะทำหน้าที่แทนไมโครโปรเซสเซอร์ของแผ่นวงจร ดังนั้นข้อมูลต่างๆ ที่อยู่บนแผ่นวงจรสามารถย้ายมายังหน่วยความจำของอีมูเลเตอร์ได้ทันที (Taget Memory) ทำให้สะดวกในการตรวจสอบแก้ไขผ่านทางเทอร์มินอล (Terminal) ผลที่ได้สามารถเก็บไว้ในแผ่นดิสก์ หรือ สามารถอ่านโปรแกรมที่บันทึกไว้ในแผ่นดิสก์ลงไปในหน่วยความจำได้เช่นกัน โดยผ่านทาง มาสเตอร์คอนโทรลโปรเซสเซอร์ (Master Control Processor) นอกจากนั้นขณะรันโปรแกรมยังสามารถนำสัญญาณภายนอก (External Inputs) เข้ามาเก็บไว้เพื่อทำการตรวจสอบเปรียบเทียบกับโปรแกรมว่าถูกต้องและสอดคล้องกันหรือไม่ ตามความสามารถที่ผ่านมาข้างต้นจะเห็นว่าเมื่อใช้เทคนิคการอีมูเลทเข้าไปในวงจรของไมโครคอมพิวเตอร์แล้ว เสมือนหนึ่งว่าได้เปิดเข้าไปในโลกที่ปิดอยู่ของมัน ซึ่งจะทำให้การวิเคราะห์ระบบทำได้รวดเร็วขึ้นเป็นอย่างมาก

ในวิทยานิพนธ์ฉบับนี้เป็นการเสนอการสร้างอีมูเลเตอร์อย่างง่ายขึ้นมาใช้งาน โดยเลือกเฉพาะใช้กับไมโครคอมพิวเตอร์แบบชิพเดียวเบอร์ 8051 ของบริษัทอินเทล

บทที่ 2 ทฤษฎีและการออกแบบ

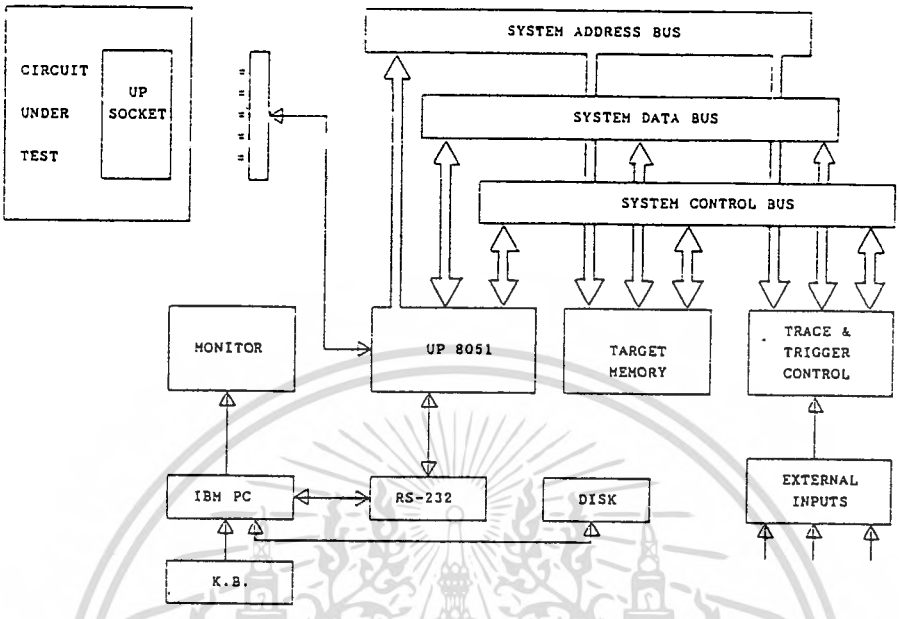
2.1 การใช้เครื่อง IBM-PC เป็นเทอร์มินอล

ในหัวข้อนี้จะเป็นการกล่าวถึงการออกแบบสร้างอีมูเลเตอร์ขึ้นใช้งาน โดยเริ่มจากรูปที่ 1.2 ในหัวข้อที่ 1 ส่วนของเทอร์มินอลจะใช้เครื่องคอมพิวเตอร์ส่วนบุคคล IBM-PC ซึ่งมีตัวขับจานแม่เหล็ก (Floppy Disk Drive) อยู่แล้วจึงใช้งานได้เลย การติดต่อสื่อสารกับอีมูเลเตอร์กระทำผ่านทางพอร์ทอนุกรม RS-232 [3] ดังนั้นสามารถเขียนรูปใหม่ได้ดังรูปที่ 2.1



รูปที่ 2.1 แสดงแผนผังของการอีมูเลทเข้าไปในวงจรโดยใช้ IBM-PC เป็นเทอร์มินอล

เมื่อพิจารณาต่อมาหากตัวอีมูเลเตอร์ถูกออกแบบให้ใช้งานกับไมโครคอมพิวเตอร์ที่มีไมโครโปรเซสเซอร์เบอร์เดียวกันตลอด จะสามารถทำการตัดส่วนของพ็อด และอีมูเลชั่นโมดูลออกได้ โดยใช้มาสเตอร์คอนโทรลโปรเซสเซอร์เบอร์เดียวกับไมโครโปรเซสเซอร์บนแผ่นวงจร เมื่อกระทำดังนี้บัสของอีมูเลเตอร์จะเหมือนกับบัสของวงจรที่ทำมาตรวจสอบทุกประการ ดังนั้นแผนผังของวงจรจะเขียนได้ดังรูปที่ 2.2



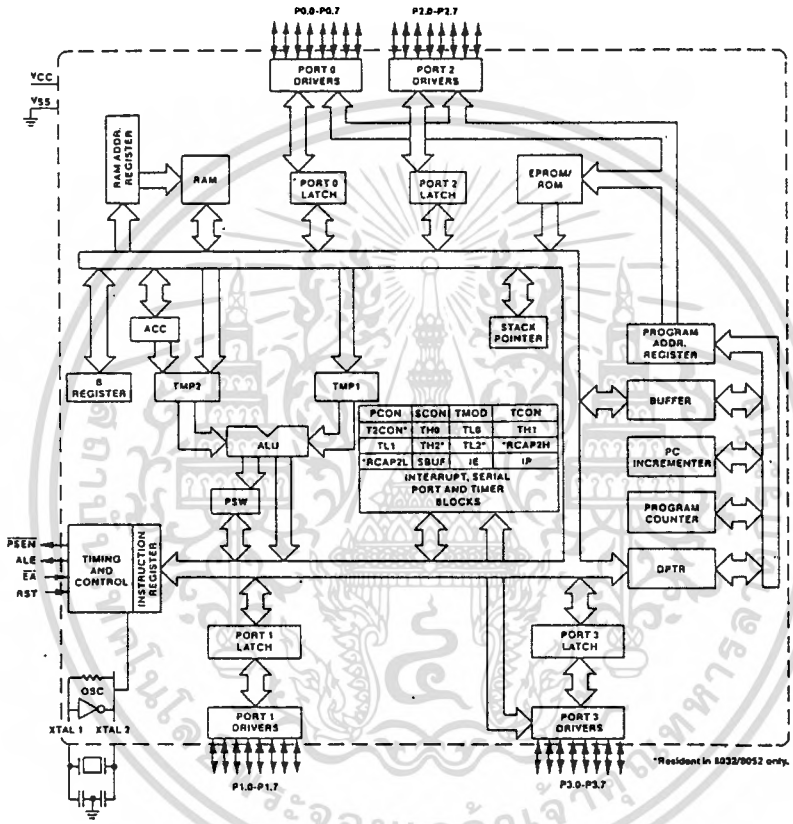
รูปที่ 2.2 แสดงแผนผังของวงจรอีมูเลเตอร์ที่ทำการออกแบบ

สำหรับการเลือกไมโครโปรเซสเซอร์ที่ใช้ในอีมูเลเตอร์นั้น จะเลือกใช้ไมโครคอมพิวเตอร์แบบชิพ-เดียว เพราะมีการนำไปใช้งานด้านการควบคุมกันเป็นที่แพร่หลาย ไมโครคอมพิวเตอร์แบบชิพเดียวก็มีหลายบริษัทที่ผลิตออกมา เช่น บริษัทอินเทล บริษัทโมโตโรลา เป็นต้น แต่เมื่อพิจารณาถึงความนิยมในการใช้งานจะพบว่า มีการนำเอาไมโครโปรเซสเซอร์ของบริษัทอินเทลมาใช้งานอย่างแพร่หลาย ไมโครคอมพิวเตอร์แบบชิพเดียวตระกูล 8048 ของบริษัทอินเทลได้ถูกนำมาใช้งานทางด้านควบคุมหลายปีแล้ว คาดว่ามีผู้ใช้งานเป็นจำนวนมากที่คุ้นเคยกับลักษณะโครงสร้างของไมโครโปรเซสเซอร์จากบริษัทอินเทล ดังนั้นจึงเลือกใช้ไมโครโปรเซสเซอร์ของบริษัทอินเทลเช่นกัน แต่จะเลือกใช้ในตระกูล 8051 (MCS-51) [4] ซึ่งคาดว่าจะได้รับความนิยมแทน 8048 เนื่องจากมีประสิทธิภาพดีกว่ากันมาก ไม่ว่าจะเป็นขนาดของหน่วยความจำ ชุดคำสั่ง ความเร็วในการทำงาน ดังนั้นเพื่อความเข้าใจในวงจรของบทถัดไป (บทที่ 3) ได้ดียิ่งขึ้นจึงจะขอกล่าวถึง MCS-51 พอเป็นสังเขปดังนี้

ไมโครคอมพิวเตอร์แบบชิพเดียวตระกูล MCS-51 เป็นไมโครคอมพิวเตอร์ที่เหมาะสมกับการนำมาใช้งานทางด้านการควบคุมเป็นอย่างมาก เพราะมีความเร็วในการทำงานสูง มีคำสั่งที่จัดการกับข้อมูลอย่างมีประสิทธิภาพ การแลกเปลี่ยนหรืออ้างอิงข้อมูลภายในสามารถกระทำได้อย่างคล่องตัว

2.2 โครงสร้างและการทำงานของไมโครคอมพิวเตอร์แบบชิพเดียวตระกูล MCS-51

บล็อกไดอะแกรมของโครงสร้างภายใน MCS-51 จะเป็นดังรูปที่ 2.3



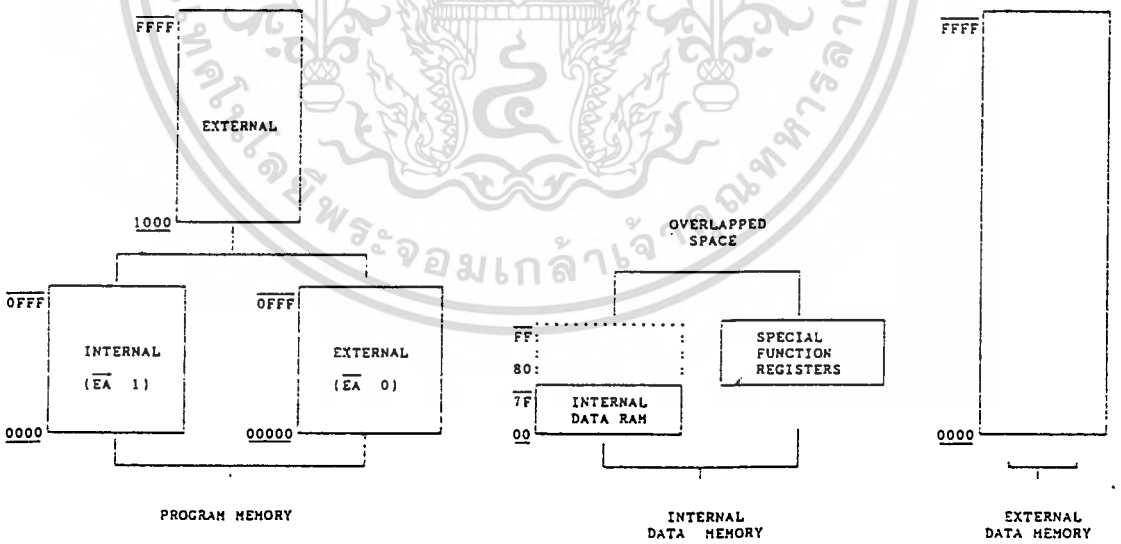
รูปที่ 2.3 แสดงโครงสร้างภายในของไมโครคอมพิวเตอร์แบบชิพเดียวตระกูล MCS-51 พร้อมทั้งขาสัญญาณต่างๆ

โครงสร้างหลักของ MCS-51 จะประกอบด้วย

- CPU ขนาด 8 บิท
- มีวงจรออสซิลเลเตอร์ภายในตัว

- มีหน่วยความจำแบบรอม ขนาด 4 กิโลไบท์
- มีหน่วยความจำแบบแรม ขนาด 128 ไบท์
- มีรีจิสเตอร์สำหรับทำหน้าที่พิเศษ 21 รีจิสเตอร์
- มีสายนำสัญญาณสำหรับเชื่อมต่อกับอุปกรณ์ภายนอก 32 เส้น
- สามารถต่อหน่วยความจำภายนอกได้ 64 กิโลไบท์
- มีไทเมอร์/เคาน์เตอร์ ขนาด 16 บิท จำนวนสองชุด
- มีโครงสร้างของการอินเทอร์รัพท์ได้ห้าแห่งประกอบด้วยระดับของความสำคัญสองระดับ
- มีพอร์ทอนุกรมที่ทำงานแบบ ฟูลดูเพล็กซ์หนึ่งพอร์ท
- สามารถอ้างอิงแอดเดรสแบบบิทได้ในกรณีที่มีการประมวลผลแบบบูลีน (Boolean Processing)

ลักษณะการแม็พของหน่วยความจำจะเป็นดังรูปที่ 2.4



รูปที่ 2.4 แสดงการแม็พของหน่วยความจำ (Memory Map)

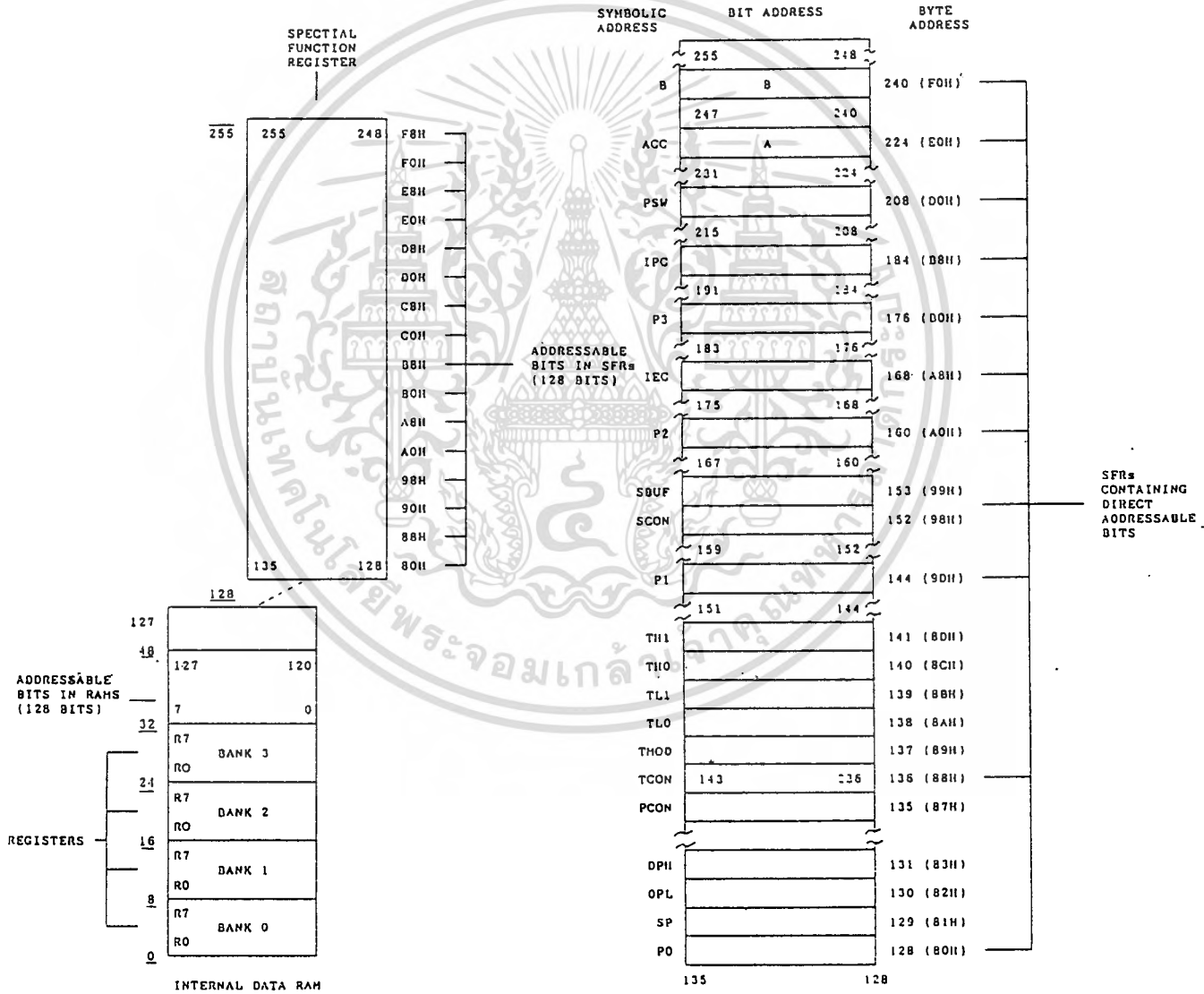
จากรูปที่ 2.4 MCS- 8051 จะแยกแอดเดรสออกเป็นสองส่วน คือ หน่วยความจำที่ใช้เก็บโปรแกรม และหน่วยความจำที่ใช้เก็บข้อมูล สำหรับหน่วยความจำที่ใช้เก็บโปรแกรมสามารถมีได้ถึง

64 กิโลไบท์โดยที่ในจำนวน 64 กิโลไบท์นี้อยู่ภายในตัว MCS-51 จำนวน 4 กิโลไบท์ การอ้างอิงหน่วยความจำ 4 กิโลไบท์นี้จะถูกควบคุมด้วยขา $\overline{EA}$ ถ้าขา $\overline{EA}$ มีลอจิกเป็น 0 จะใช้หน่วยความจำภายนอกทั้งหมด 64 กิโลไบท์ แต่ถ้าเป็นลอจิก 1 จะใช้หน่วยความจำภายใน 4 กิโลไบท์แรกจากนั้นจะใช้หน่วยความจำภายนอกจนกระทั่งครบ 64 กิโลไบท์ ส่วนหน่วยความจำที่ใช้เก็บข้อมูลประกอบด้วยแรมภายในตัวจำนวน 128 ไบท์ พร้อมกับรีจิสเตอร์สำหรับทำหน้าที่พิเศษต่างๆ (Special Function Registor) อีกจำนวน 21 รีจิสเตอร์ ซึ่งมีชื่อต่างๆ ดังนี้

- ACC Accumulator
- B B Registor
- PSW Program Status Word
- SP Stack Pointer
- DPTR Data Pointer (แยกเป็น DPH และ DPL)
- P0 Port 0
- P1 Port 1
- P2 Port 2
- P3 Port 3
- IP Interrupt Priority
- IE Interrupt Enable
- TMOD Timer/Counter Mode
- TCON Timer/Counter Control
- TH0 Timer/Counter 0 (High Byte)
- TL0 Timer/Counter 0 (Low Byte)
- TH1 Timer/Counter 1 (High Byte)
- TL1 Timer/Counter 1 (low Byte)

- SCON Serial control
- SBUF Serial Data Buffer
- PCON Power Control

รายละเอียดของการแบ่งหน่วยความจำที่ใช้เก็บข้อมูลจะเป็นดังรูปที่ 2.5



รูปที่ 2.5 แสดงถึงการแบ่งหน่วยความจำที่ใช้เก็บข้อมูล

จากรูปที่ 2.5 หน่วยความจำภายในที่ใช้เก็บข้อมูลจะถูกแบ่งออกเป็นสองส่วนคือ แอดเดรส 0 ถึง 127 ใช้เก็บข้อมูลภายใน และแอดเดรสที่ 128 ถึง 255 จะทำหน้าที่เป็นรีจิสเตอร์ที่ทำหน้าที่พิเศษตามชื่อที่กำหนด และมีตำแหน่งที่แน่นอน เฉพาะส่วนแรกตั้งแต่แอดเดรส 0 ถึง 31 จะถูกแบ่งออกเป็นสี่กลุ่มๆ ละแปดไบท์ เรียกแต่ละกลุ่มว่ารีจิสเตอร์แบงค์ 0 ถึง 3 เรียงกันไปตามลำดับ แต่ละแบงค์จะประกอบด้วยรีจิสเตอร์สำหรับใช้งานทั่วไปอีกแปดตัวเริ่มจาก 0 ถึง 7 เรียงไปตามลำดับเช่นกัน หรือจะเรียกใช้งานตามค่าแอดเดรสโดยตรงเลยก็ได้

สำหรับรายละเอียดของรีจิสเตอร์ที่ทำหน้าที่พิเศษต่างๆ จะเป็นดังนี้

- แอกคิวมูเลเตอร์ เป็นรีจิสเตอร์ที่ใช้เก็บข้อมูลหรือผลลัพท์ของคำสั่ง
- รีจิสเตอร์บี เป็นรีจิสเตอร์ที่ใช้เฉพาะคำสั่งการคูณและหารเท่านั้น
- สแต็กพ็อยเตอร์ เป็นรีจิสเตอร์ขนาดแปดบิท ค่าของสแต็กจะอยู่ที่ใดๆ ของหน่วยความจำแรมจำนวน 128 ไบท์ภายในชิพ เมื่อ 8051 ถูกรีเซ็ทค่าสแต็กจะถูกกำหนดเริ่มต้นไว้ที่ 07H
- ดาต้าพ็อยเตอร์ เป็นรีจิสเตอร์ขนาด 16 บิท แยกออกเป็นไบท์สูง (DPH) และไบท์ต่ำ (DPL) หน้าที่หลักคือเอาไว้เก็บค่าแอดเดรสจำนวน 16 บิท
- พอร์ท 0 ถึง 3 จะเป็นพอร์ทขนาด แปดบิทที่สั่งให้เป็นอินพุทหรือเอาท์พุทก็ได้ และแบบที่ค้างสภาวะทางลอจิกไว้ได้โดยสั่งผ่านทางรีจิสเตอร์ที่ทำหน้าที่พิเศษ คือรีจิสเตอร์ P0 ถึง P3 พอร์ท 0 และพอร์ท 2 ยังใช้เชื่อมต่อกับหน่วยความจำภายนอกด้วย นอกจากนั้น พอร์ท 3 ยังถูกใช้เป็นพอร์ทอนุกรมรับสัญญาณอินเตอร์รัพท์จากภายนอก เป็นไทเมอร์ และสัญญาณอ่านหรือเขียนข้อมูลไปยังหน่วยความจำภายนอกอีกด้วยดังนี้

ขาของพอร์ท 3	หน้าที่อื่นนอกจากเป็น I/O
P3.0	RXD (รับสัญญาณอนุกรม)
P3.1	TXD (ส่งสัญญาณอนุกรม)
P3.2	$\overline{\text{INT0}}$ (อินเทอร์รัพท์จากภายนอก)
P3.3	$\overline{\text{INT1}}$ (อินเทอร์รัพท์จากภายนอก)
P3.4	T0 (ไทเมอร์ 0 รับสัญญาณจากภายนอก)
P3.5	T1 (ไทเมอร์ 1 รับสัญญาณจากภายนอก)

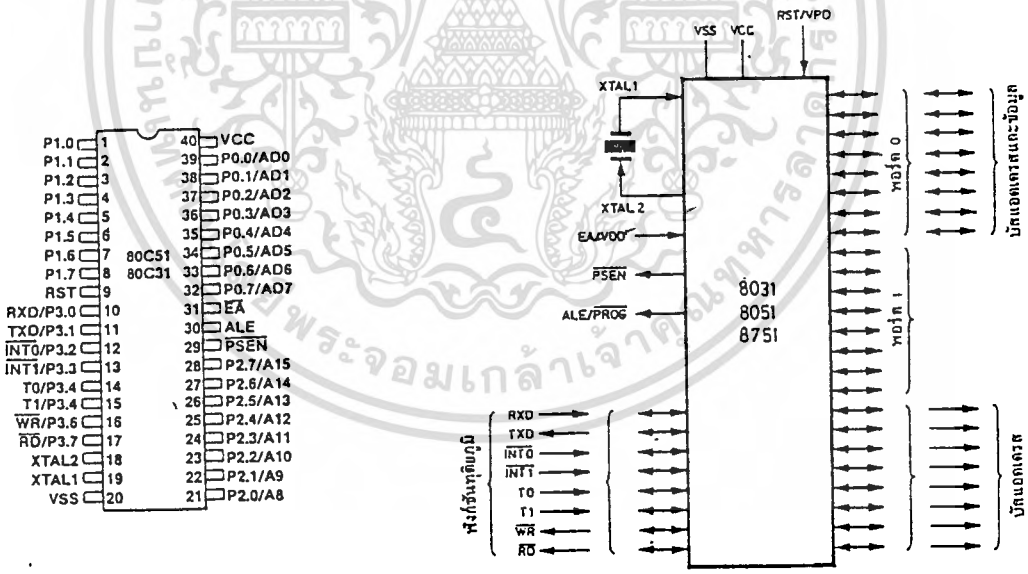
P3.6 $\overline{WR}$ (สัญญาณเขียนหน่วยความจำภายนอก)

P3.7 $\overline{RD}$ (สัญญาณอ่านหน่วยความจำภายนอก)

- บัฟเฟอร์ข้อมูลอนุกรม ถูกแยกเป็นสองรีจิสเตอร์ ตัวที่หนึ่งสำหรับส่งข้อมูลออกและอีกตัวหนึ่งสำหรับรับข้อมูลเข้า

- รีจิสเตอร์ควบคุมและแสดงสภานะ ได้แก่ รีจิสเตอร์ IP IE TMOD TCON SCON และ PCON ซึ่งแต่ละตัวจะประกอบด้วยบิทที่ใช้ควบคุมและแสดงสถานะสำหรับการอินเทอร์รัพท์ของระบบไทเมอร์ และพอร์ทอนุกรม

เพื่อให้เห็นภาพพจน์ของ 8051 ที่เกี่ยวข้องพอร์ทและส่วนต่างๆ ได้ชัดเจนขึ้นจะเขียนหน้าที่ของพอร์ทเมื่อทำงานกับหน่วยความจำภายนอกได้ดังรูปที่ 2.6



รูปที่ 2.6 แสดงหน้าที่ของพอร์ทเมื่อเชื่อมต่อกับหน่วยความจำภายนอก

หลังจากทำความเข้าใจกับโครงสร้างภายในแล้ว ลองพิจารณาถึงการใช้งานของสายสัญญาณที่ปรากฏภายนอกบ้างเพื่อให้เข้าใจได้ดีขึ้น

จากโครงสร้างภายในของ MCS-51 ในรูปที่ 2.3 จะมีบัสสองทิศทาง 4 เส้น และพอร์ทขนาด 8 บิท เมื่อใช้หน่วยความจำภายในตัว ในกรณีที่ไม่ใช้หน่วยความจำภายใน พอร์ท 0 และพอร์ท 2 จะถูกใช้เป็นบัสของข้อมูลและแอดเดรส ดังนั้นพอร์ททั้งสองยังคงใช้งานเป็นอินพุทและเอาท์พุทโดย พอร์ท 0 ทำหน้าที่เป็นสายสัญญาณแอดเดรส A0.....A7 และเป็นบิทข้อมูล D0.....D7 ส่วน พอร์ท 2 ใช้งานเป็นสายสัญญาณแอดเดรส A8.....A15

เอาท์พุทของขา $\overline{RD}$ และ $\overline{WR}$ มาจากสายเอาท์พุทของพอร์ท 3 เพื่อใช้ในการอ่านและเขียนข้อมูลของหน่วยความจำภายนอก

ขา $\overline{PSEN}$ เป็นขารับสัญญาณสำหรับเปิดให้มีการอ่านหน่วยความจำภายนอกโดยสัญญาณ $\overline{PSEN}$ จะทำงานสองครั้งในหนึ่งรอบคำสั่งเหมือนสัญญาณ ALE เนื่องจากมีการอ่านข้อมูลจำนวนสองไบท์ในแต่ละรอบคำสั่ง และขา $\overline{PSEN}$ นี้จะทำงานต่อเมื่อมีภาษาเครื่องเก็บอยู่ในหน่วยความจำภายนอกเท่านั้น

ขา $\overline{EA}$ เป็นอินพุทที่ใช้ร่วมกับแอดเดรสภายนอกโดยมีค่าลอจิก "0" เมื่อโปรเซสเซอร์อ่านคำสั่งจากหน่วยความจำภายนอก

จะเห็นว่า 8051 มีวงจรนับและตั้งเวลาชนิด 16 บิทอยู่สองตัว เมื่อทำหน้าที่เป็นวงจรตั้งเวลา รีจิสเตอร์ที่ทำหน้าที่ตั้งเวลาจะเพิ่มค่าขึ้นหนึ่งทุกๆ รอบคำสั่งของเครื่อง และจะนับด้วยอัตราสูงสุดที่ 1/12 ของความเร็วสัญญาณนาฬิกาของโปรเซสเซอร์ ส่วนการทำงานเป็นวงจรนับรีจิสเตอร์จะนับเพิ่มขึ้นหนึ่งเมื่อสัญญาณป้อนให้อินพุท T0 หรือ T1 เป็นขอบของสัญญาณขาลง อัตราการนับสูงสุดคือ 1/24 ของความเร็วสัญญาณนาฬิกาของโปรเซสเซอร์ วงจรนับและตั้งเวลา 0 และ 1 มีวิธีโปรแกรมให้ทำงานได้ต่างกันถึง 4 แบบ รวมทั้งการทำงานเป็น 8 บิท หรือ 16 บิท และการบรรจุค่าพรีเซ็ตหนึ่งค่าได้อย่างอัตโนมัติ

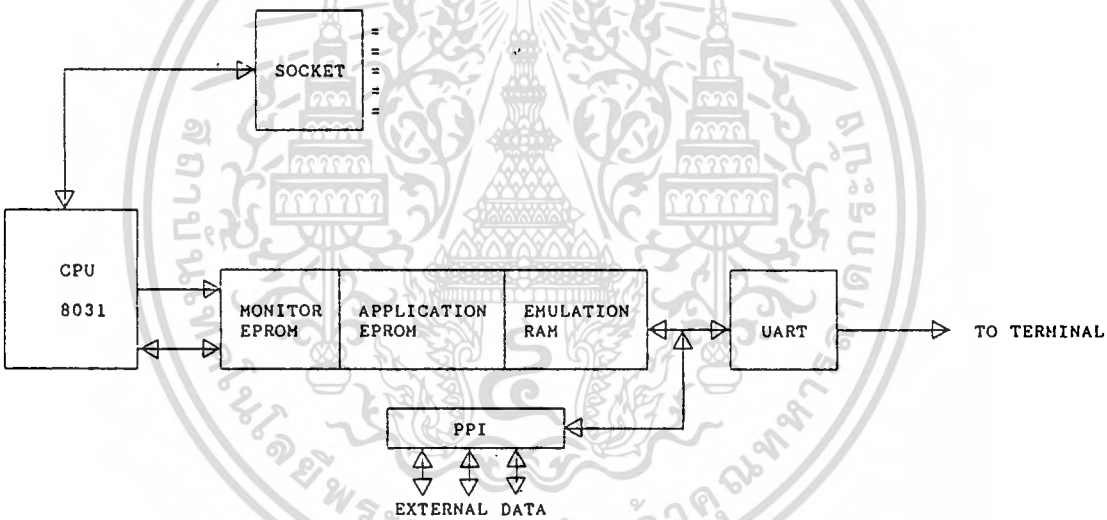
8051 เป็นชิพที่มีอินเตอร์เฟสอนุกรมชนิดสองทิศทาง ทำให้รับและส่งข้อมูลพร้อมกัน ตัวรับข้อมูลชนิดอะซิงโครนัส (Asynchronous Receiver) มีบัฟเฟอร์สำหรับข้อมูลเป็นพิเศษเพิ่มความเร็วในการสื่อสาร และสามารถเลือกโปรแกรมใช้งานแบบหนึ่งแบบใดใน 4 แบบ ด้วยการใช้โปรแกรมควบคุมอัตราการส่งข้อมูลและรูปแบบของข้อมูล

8051 สามารถรับการอินเทอร์พัพท์ได้ 5 แหล่งคือ $\overline{INT0}$ $\overline{INT1}$ (กำหนดให้ใช้ระดับพัลส์

หรือขอบชาพัลส์ก็ได้) และจากวงจรนับและตั้งเวลาที่ 0 ถึง 1 รวมทั้งจากพอร์ทอนุกรม โดยสามารถกำหนดลำดับความสำคัญของการอินเทอร์รัพท์ได้ 2 ระดับ แต่ละแหล่งของอินเทอร์รัพท์จะกำหนดเป็นเวกเตอร์เฉพาะ (ชี้แอดเดรส) ดังนั้นเมื่อมีอินเทอร์รัพท์เข้ามา ตัวโปรเซสเซอร์จะกระโดดไปที่ส่วนของโปรแกรมที่ทำงานตามวัตถุประสงค์ของอินเทอร์รัพท์นั้น หลังจากเก็บข้อมูลต่างๆ ของโปรแกรม-เคาน์เตอร์ลงในสแต็ก

2.3 <u>การออกแบบ 8051 อีมูเลเตอร์</u>

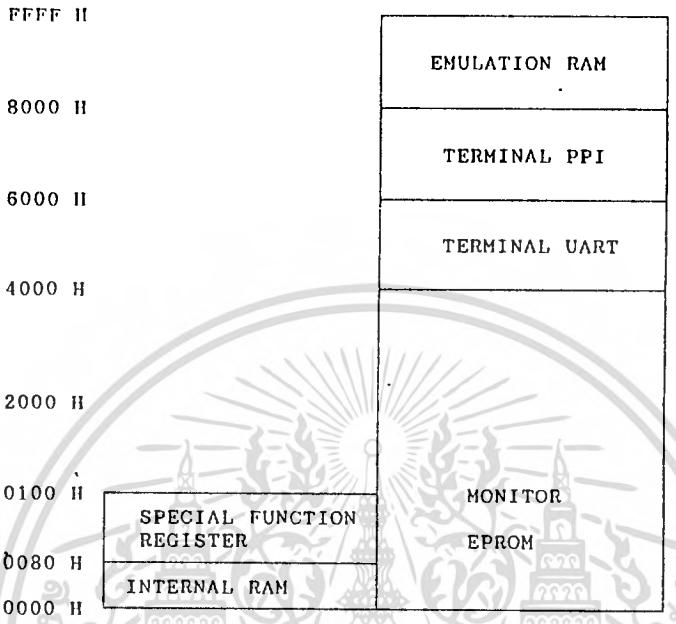
ในส่วนของการออกแบบ 8051 อีมูเลเตอร์ที่ออกแบบสร้างสามารถเขียนเป็นบล็อกไดอะแกรมได้ดังรูปที่ 2.7



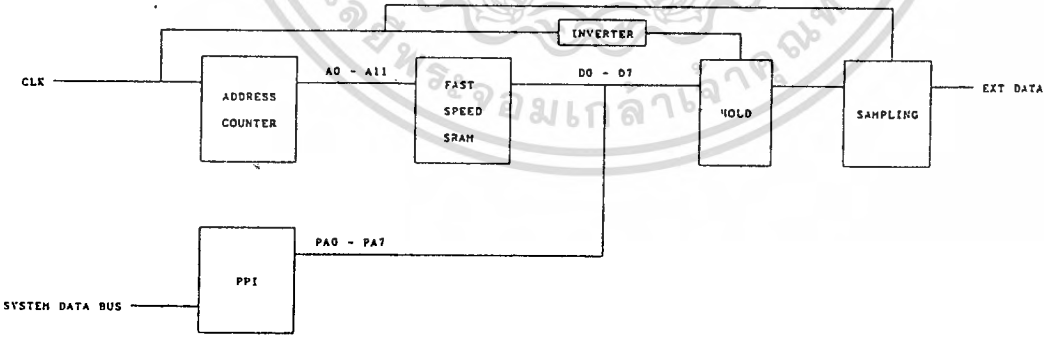
รูปที่ 2.7 แสดงบล็อกไดอะแกรมของ 8051 อีมูเลเตอร์

จากรูปที่ 2.7 ไมโครโปรเซสเซอร์จะใช้เบอร์ 8031 ซึ่งจะเหมือนกับ 8051 เพียงแต่ไม่มีหน่วยความจำแบบรอมภายใน โดยขาทุกขาของไมโครโปรเซสเซอร์จะถูกต่อออกมารอไว้ที่ซ็อคเก็ตเพื่อต่อไปยังบอร์ดอื่นด้วยสายแถบขนาด 40 เส้น หน่วยความจำของระบบจะประกอบด้วย โปรแกรมควบคุมระบบของ 8051 อีมูเลเตอร์ และอีมูเลชั่นแรมที่ใช้สำหรับเก็บโปรแกรมของบอร์ดอื่นหรือโปรแกรมอื่นๆ ที่จะสั่งให้ 8051 อีมูเลเตอร์ทำงานนอกเหนือไปจากโปรแกรมที่กำหนดไว้ในโปรแกรมควบคุมระบบ การติดต่อกับ IBM-PC จะกระทำผ่านทางชิพ USART [5] ซึ่งทำหน้าที่เป็นพอร์ทอนุกรม

ส่วนการรับสัญญาณของ 8051 จะผ่านทางชิพ PPI (Programmable peripheral interface) เบอร์ 8255 และลักษณะของหน่วยความจำจะเป็นดังรูปที่ 2.8



รูปที่ 2.8 แสดงการแม็พหน่วยความจำของ 8051 อีมูเลเตอร์



รูปที่ 2.9 แสดงบล็อกไดอะแกรมของส่วนตักจับข้อมูล (Data capture)

จากรูปที่ 2.9 แสดงถึงส่วนตักจับข้อมูล (Data capture) เพื่อนำเอาสัญญาณจากบัสข้อมูลเข้ามาวิเคราะห์ในระบบ เนื่องจากการเปลี่ยนแปลงของข้อมูลบนบัสข้อมูลมีความเร็วสูง จึงเลือกใช้หน่วยความจำชนิดความเร็วสูง (Fast Ram) เก็บข้อมูล จากรูป สัญญาณข้อมูลจะถูกสุ่ม(Sampling) เข้ามาและคงไว้ (Hold) บนบัสข้อมูลของฟาสแรม ฟาสแรมจะเก็บข้อมูลเข้าไปยังตำแหน่งแอดเดรสที่ส่งออกมาจากส่วนสร้างสัญญาณแอดเดรส (Address counter) ส่วนสร้างสัญญาณแอดเดรสจะเพิ่มค่าขึ้นที่ละหนึ่งตามสัญญาณนาฬิกาที่เข้ามากระตุ้น

เมื่อต้องการอ่านข้อมูลในฟาสแรมเข้ามายังระบบ จะกระทำโดยสร้างสัญญาณนาฬิกาเข้ามาเป็นตัวกระตุ้นโดยผ่านทาง PPI เพื่อสร้างสัญญาณแอดเดรสให้ฟาสแรม ข้อมูลที่ส่งออกมาจากฟาสแรมจะถูกรับเข้าสู่ระบบโดยผ่านทาง PPI เช่นเดียวกัน

2.4 <u>ชุดคำสั่งที่ใช้กับ 8051 อีมูเลเตอร์</u>

ชุดคำสั่งที่ออกแบบไว้ใน 8051 อีมูเลเตอร์มีคำสั่งดังต่อไปนี้

A(uto)	หมายถึง ให้ 8051 อีมูเลเตอร์ รันโปรแกรมที่ตำแหน่งแอดเดรสที่กำหนดให้ พร้อมทั้งตรวจจับและบันทึกข้อมูลที่เกิดขึ้นบนบัสข้อมูล
B(reak)	หมายถึง การกำหนดให้โปรแกรมหยุดเพื่อดูค่าต่างๆ ในรีจิสเตอร์
C(opy)	หมายถึง การลอกข้อมูลจากที่หนึ่งไปยังอีกที่หนึ่งบนหน่วยความจำ
D(ump)	หมายถึง การนำเอาข้อมูลในหน่วยความจำขึ้นมาแสดงบนจอภาพ
E(dit)	หมายถึง การแก้ไขข้อมูลในหน่วยความจำ
F(ind)	หมายถึง ให้ค้นหาข้อมูลที่กำหนด และแสดงตำแหน่งข้อมูลที่ค้นพบ
G(o)	หมายถึง ให้ 8051 อีมูเลเตอร์รันโปรแกรมที่ตำแหน่งแอดเดรสที่กำหนดให้
H(elp)	หมายถึง ให้แสดงคำสั่งที่ใช้งานใน 8051 อีมูเลเตอร์
I(nsert)	หมายถึง ให้ใส่ค่าที่ตามมาหลังคำสั่งลงไปยังตำแหน่งแอดเดรสที่ระบุ
J(ump)	หมายถึง ให้แสดงการกระโดดเพื่อรันโปรแกรมบางส่วนภายในโปรแกรมควบคุมระบบ
L(oad)	หมายถึง ให้รอรับโปรแกรมที่ทำการคอมไพล์เรียบร้อยแล้วและอยู่ในรูปของ Hex Intel's Format แล้วบรรจุลงในหน่วยความจำ

M(ove) หมายถึง การย้ายข้อมูลจากแคชแรมมายังหน่วยความจำของระบบ

P(reprocessor) หมายถึง การกรองข้อมูลที่บันทึกได้จากบัสข้อมูลของ 8051 อีมูเลเตอร์

S(ingle) หมายถึง ให้รันโปรแกรมที่ละคำสั่งพร้อมทั้งแสดงค่าต่างๆ ในเรจิสเตอร์

U(nassembler) หมายถึง ให้เปลี่ยนออปโค้ต (Operation code) จากเลขฐานสิบหก ให้เป็นภาษาแอสเซมเบลอร์

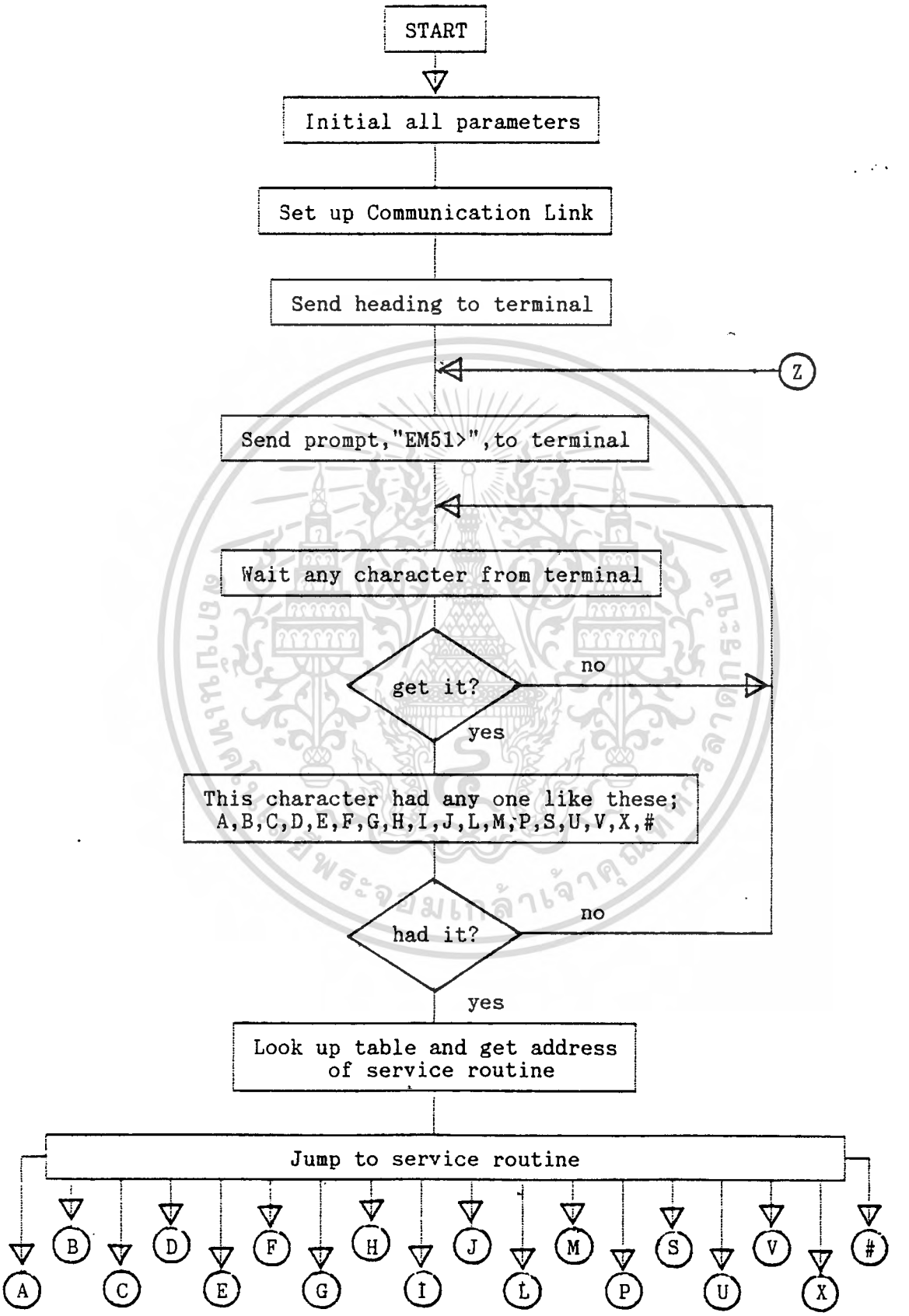
V(erify) หมายถึง ให้ทำการเปรียบเทียบข้อมูลภายในหน่วยความจำ

eX(ternal) หมายถึง ให้เก็บข้อมูลจากภายนอกลงบนแคชแรม โดยใช้สัญญาณนาฬิกาจากภายนอก

หมายถึง ให้นำเลขฐานสิบหกสองจำนวนมาบวกและลบกันแล้วแสดงผล

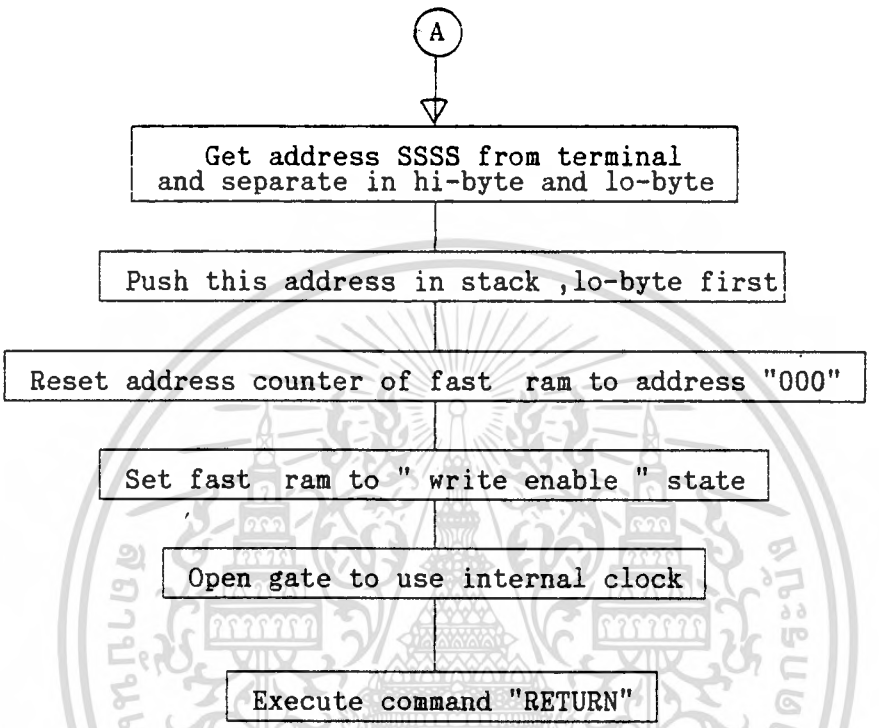
2.5 <u>โฟลว์ชาร์ทแสดงการทำงานของ 8051 อีมูเลเตอร์</u>

โฟลว์ชาร์ทการทำงานของโปรแกรมแต่ละคำสั่งจะมีรายละเอียดดังนี้



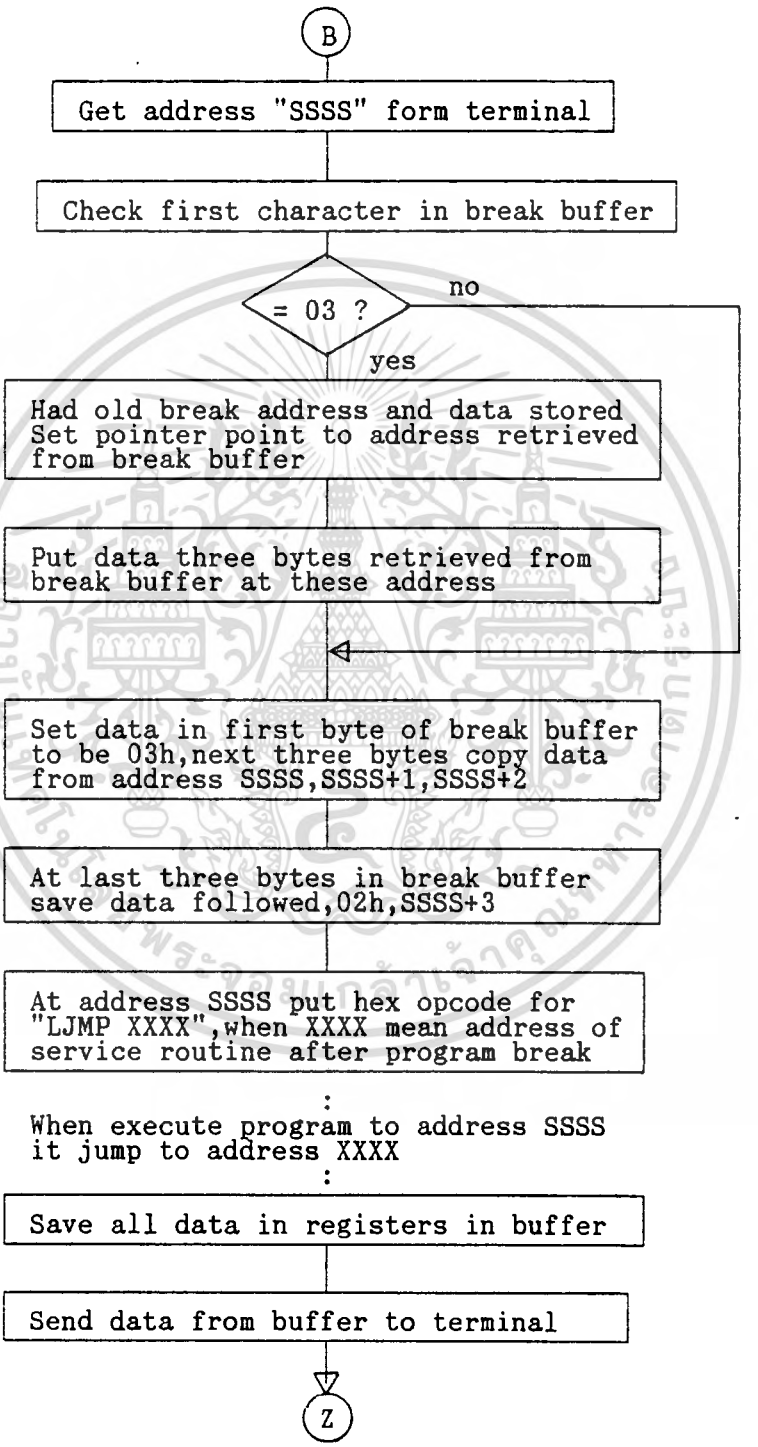
รูปที่ 2.10 แสดงโฟล์ชาร์ทของการเริ่มต้น

Routine ; A(uto)
Syntax ; A SSSS
Detail : Execute program at address SSSS and capture data



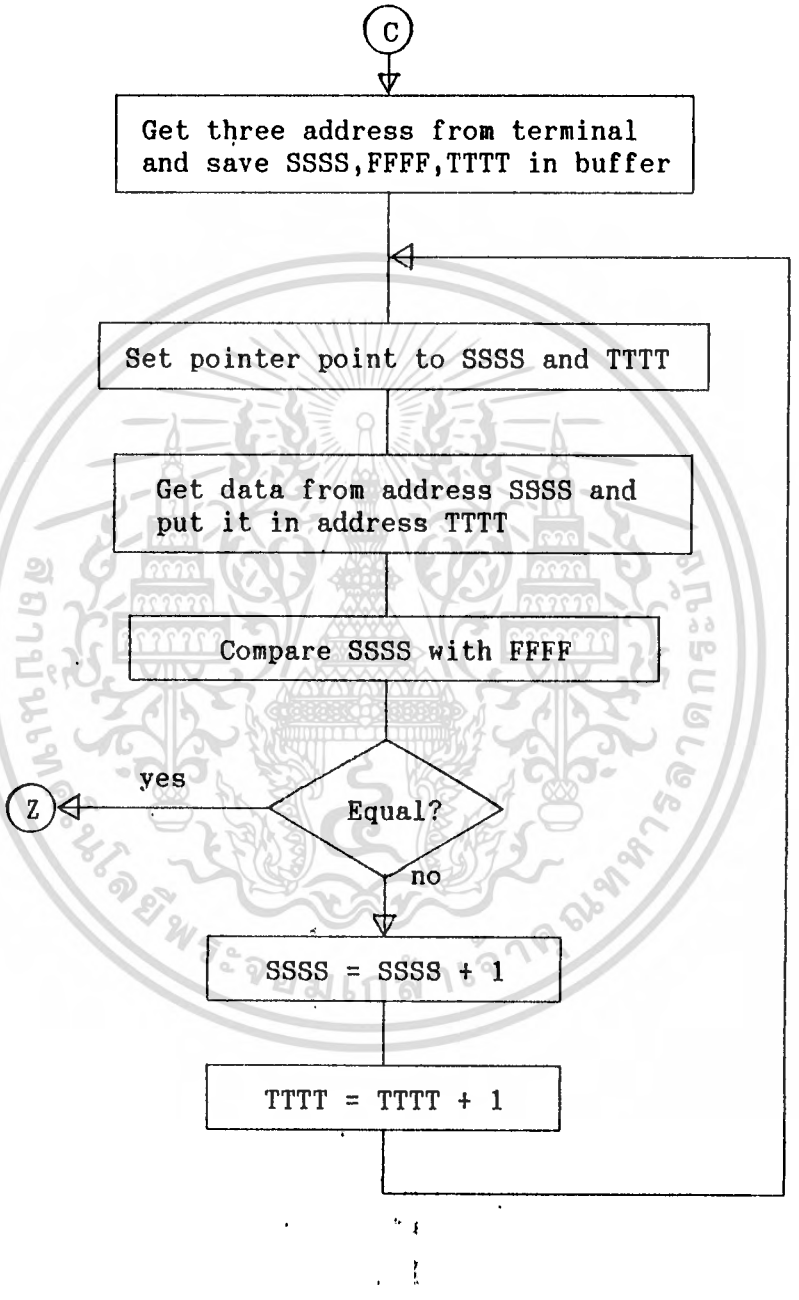
รูปที่ 2.11 แสดงโฟล์ชาร์ทของคำสั่ง A(uto)

Routine ; B(reak)
Syntax ; B SSSS
Detail ; Program break at address "SSSS" and show data in some registers.



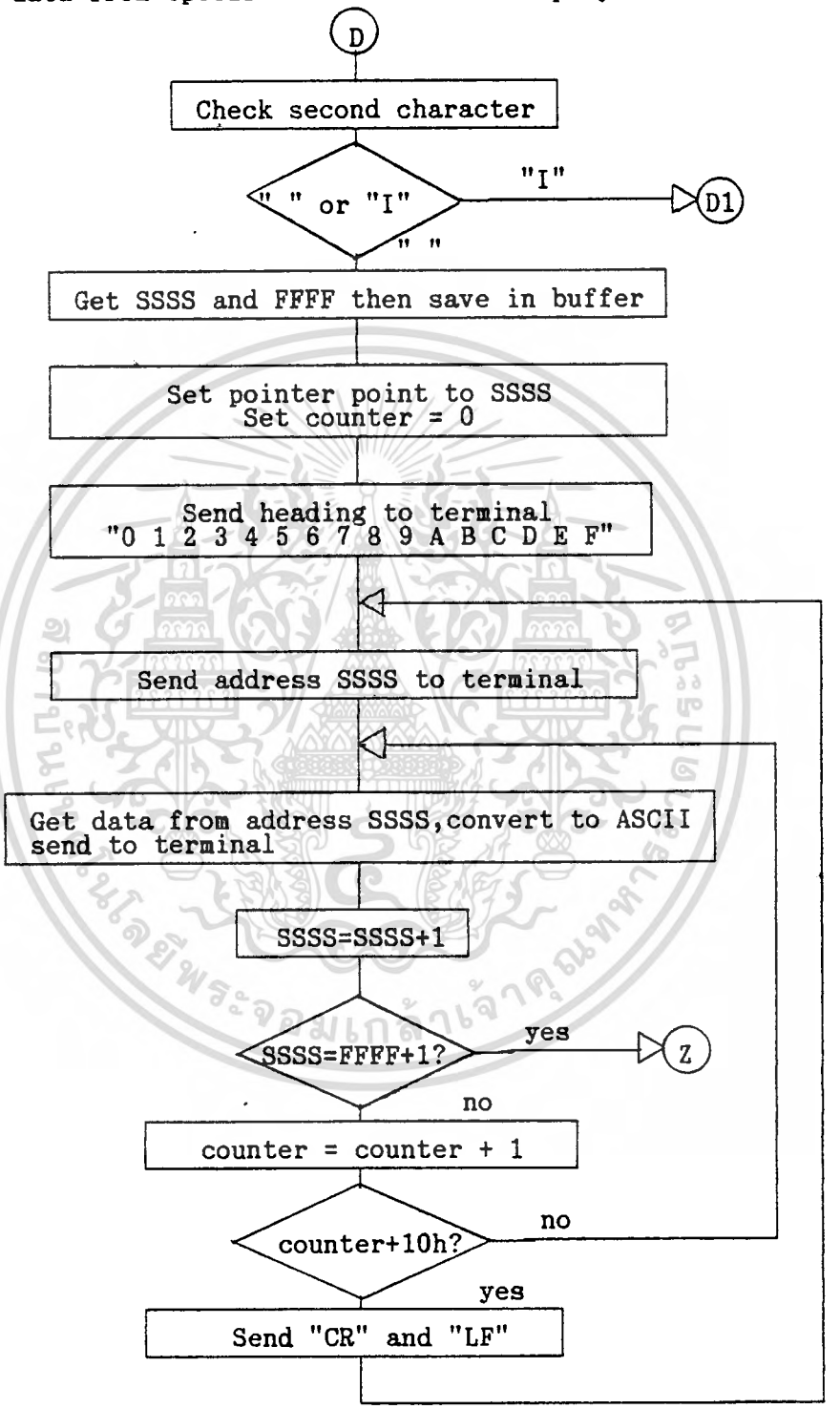
รูปที่ 2.12 แสดงโฟล์ชาร์ทของคำสั่ง B(reak)

```
Routine ; C(opy)
Syntax  ; C SSSS FFFF TTTT
          when SSSS = start address of source block
               FFFF = end address of source block
               TTTT = start address of target block
Detail  ; Copy data from one block of memory to another
```

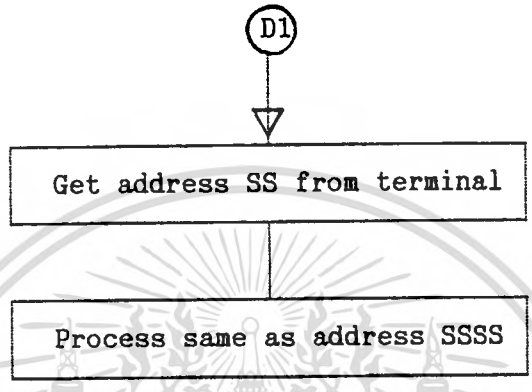


รูปที่ 2.13 แสดงโฟล์ชาร์ทของคำสั่ง C(opy)

```
Routine ; D(ump)
Syntax  ; D SSSS FFFF   (for dump external memory)
          DI SS FF      (for dump internal memory)
Detail  ; Dump data from specified address and display on terminal
```

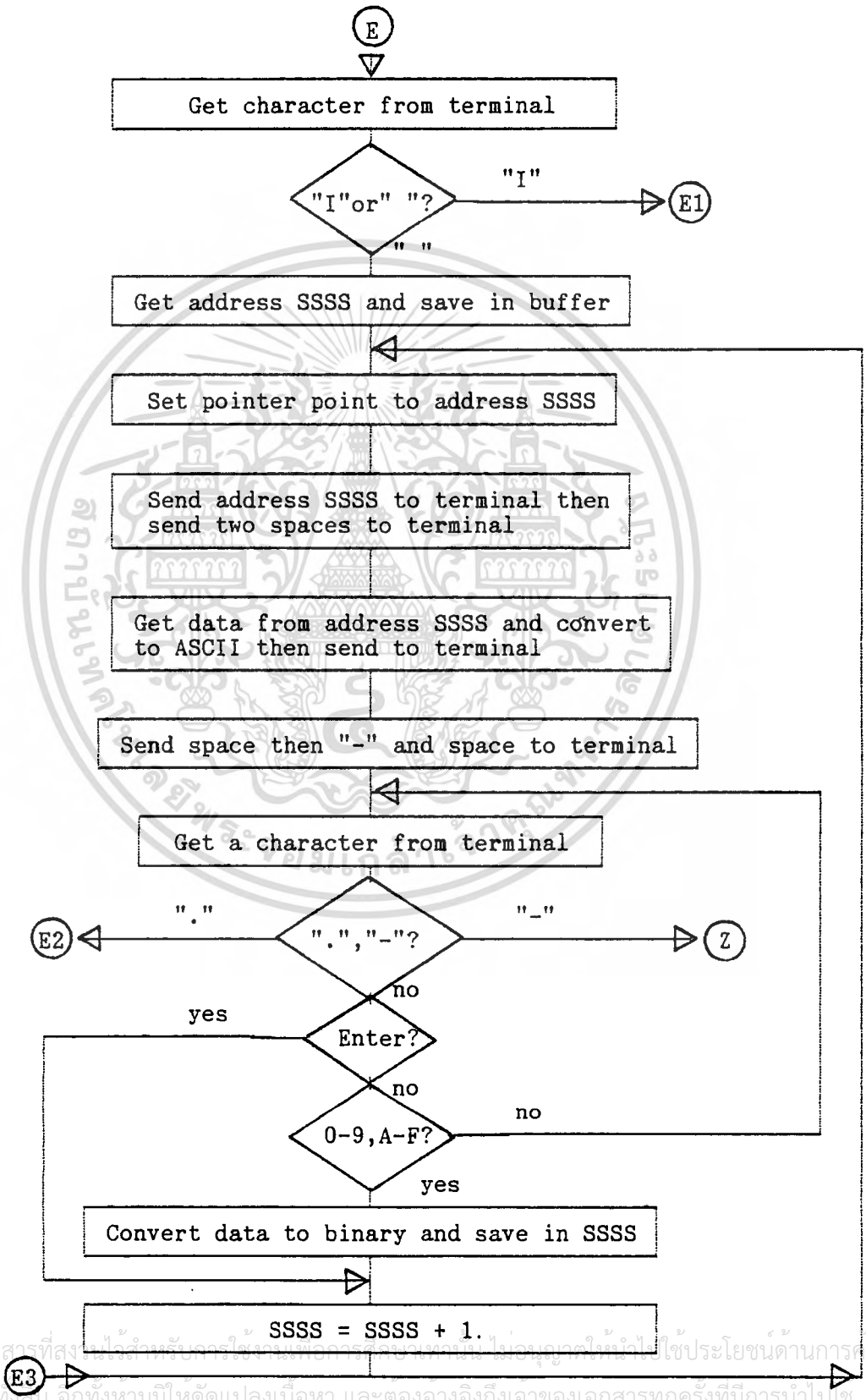


รูปที่ 2.14ก แสดงโฟล์ชาร์ทของคำสั่ง D(ump)

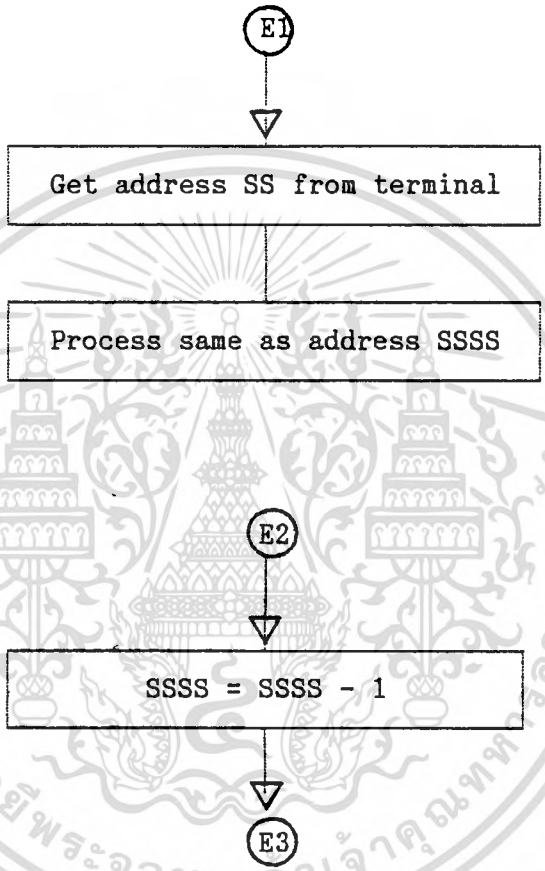


รูปที่ 2.14ข แสดงโฟล์ชาร์ทของคำสั่ง D(ump)

```
Routine ; E(dit)
Syntax  ; E SSSS    (for external memory)
          EI SS     (for internal memory)
Detail  ; Edit data in memory at specified address,end by "-".
```

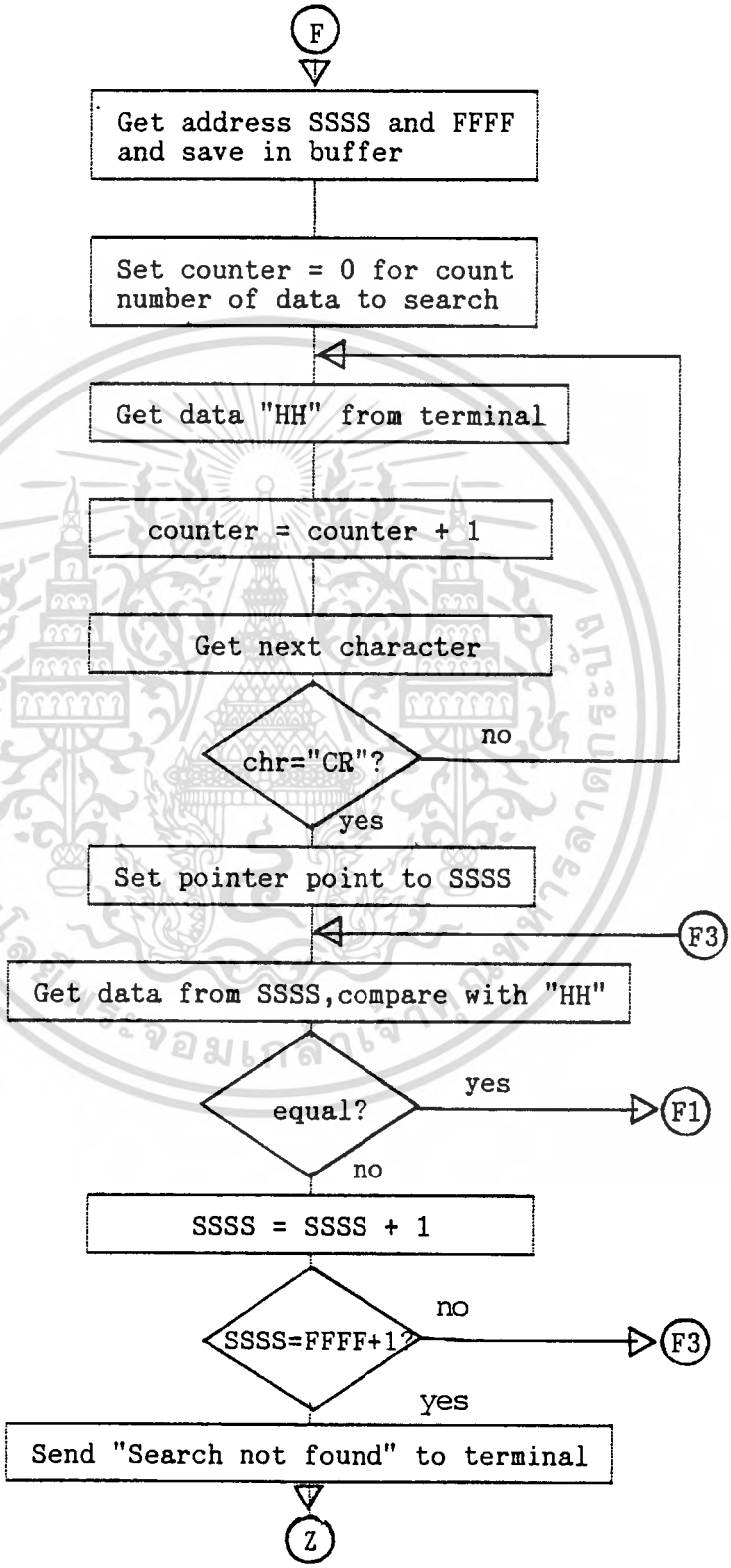


รูปที่ 2.15ก แสดงโฟล์ชาร์ทของคำสั่ง E(dit)

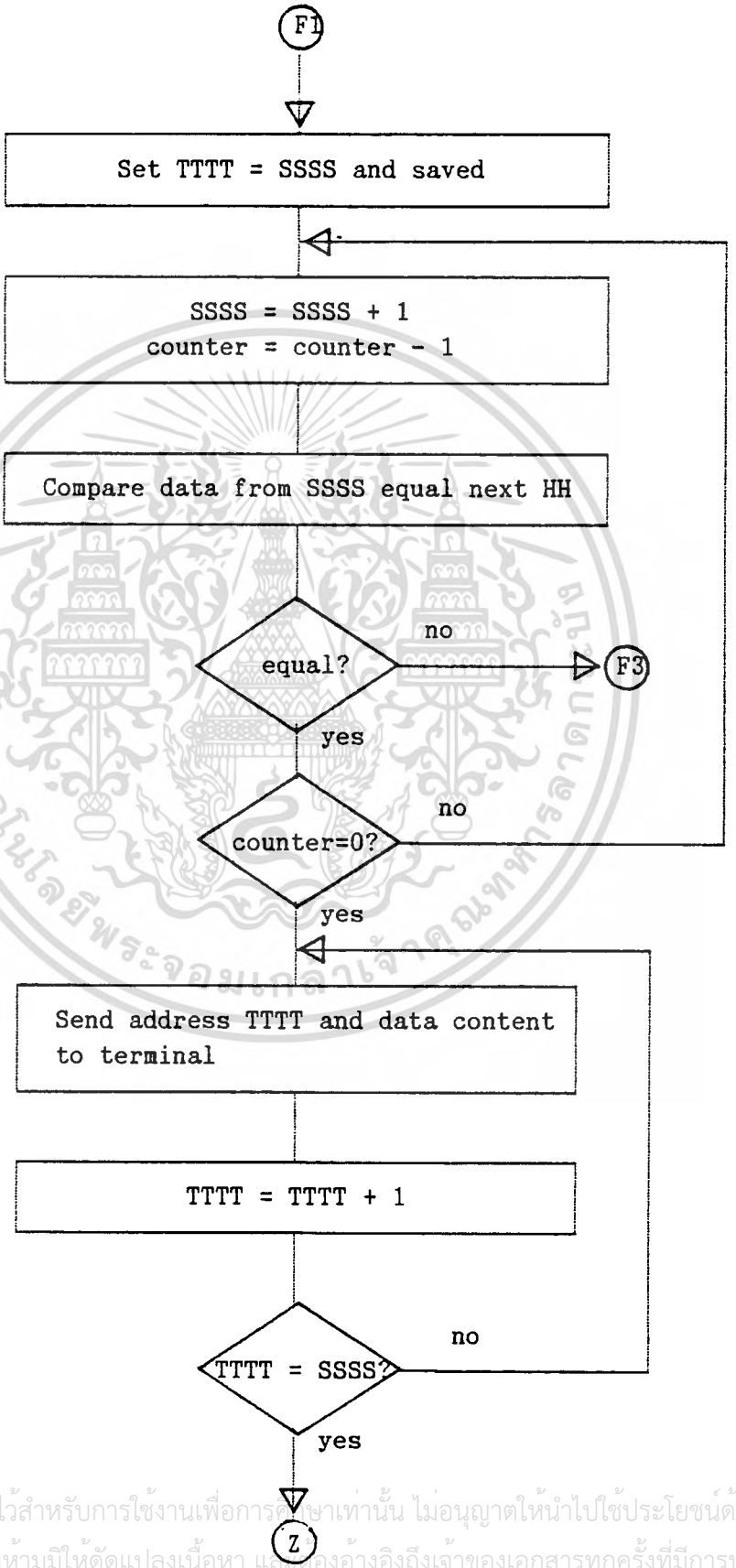


รูปที่ 2.15ข แสดงโฟล์ชาร์ทของคำสั่ง E(dit)

Routine : F(ind)
Syntax : F SSSS FFFF HH HH HH
Detail : Find data in specified block of memory

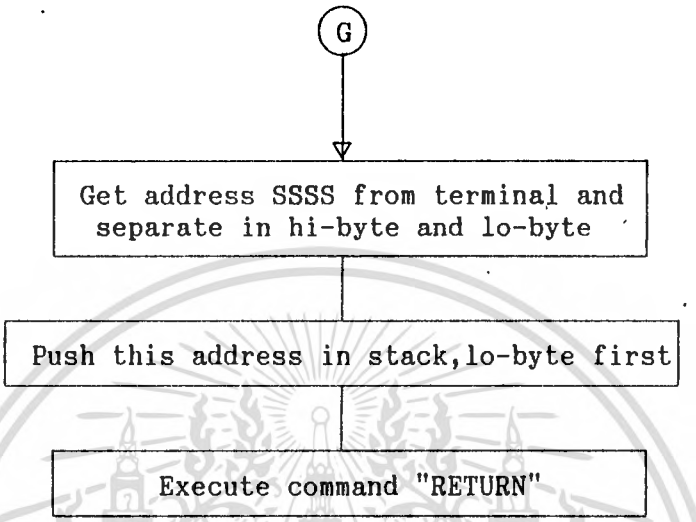


รูปที่ 2.16ก แสดงฟลัชชาร์ทของคำสั่ง F(ind)



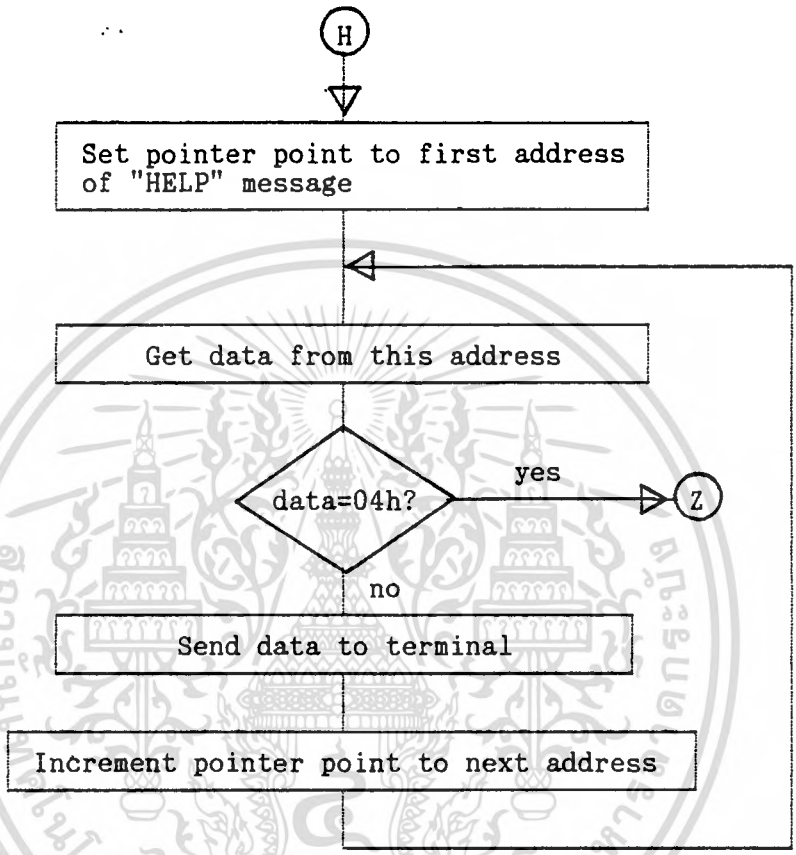
รูปที่ 2.16ข แสดงโฟล์ชาร์ทของคำสั่ง F(ind)

```
Routine ; G(o)
Syntax  ; G SSSS    (for start execute program)
          G         (for continue after program break)
Detail  ; Execute program at address SSSS or continue after break.
```



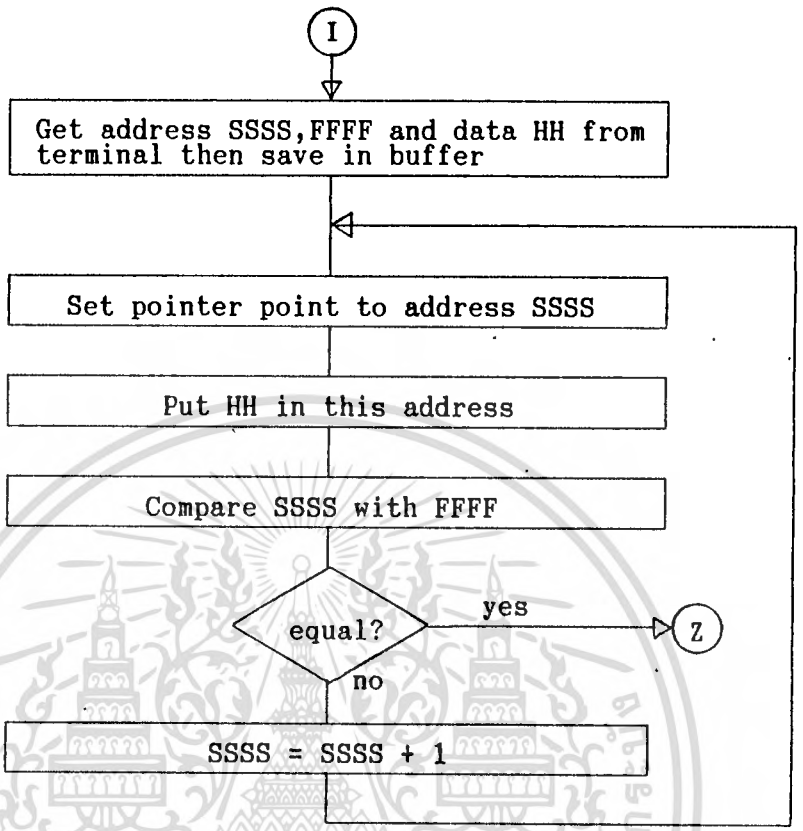
รูปที่ 2.17 แสดงไฟล์ชาร์ทของคำสั่ง G(o)

```
Routine ; H(elp)
Syntax  ; H
Detail  ; Display all command used in EMULATOR to terminal.
```



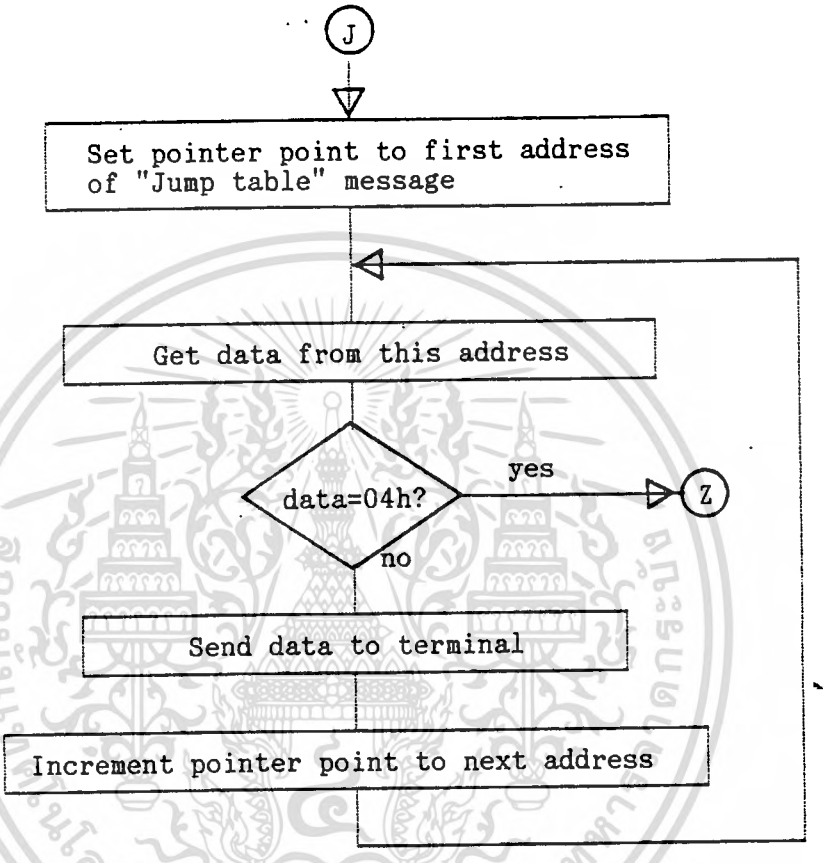
รูปที่ 2.18 แสดงโฟล์ชาร์ทของคำสั่ง H(elp)

```
Routine ; I(nsert)
Syntax  ; I SSSS FFFF HH
Detail  ; Insert "HH" entire memory from address SSSS to FFFF.
```



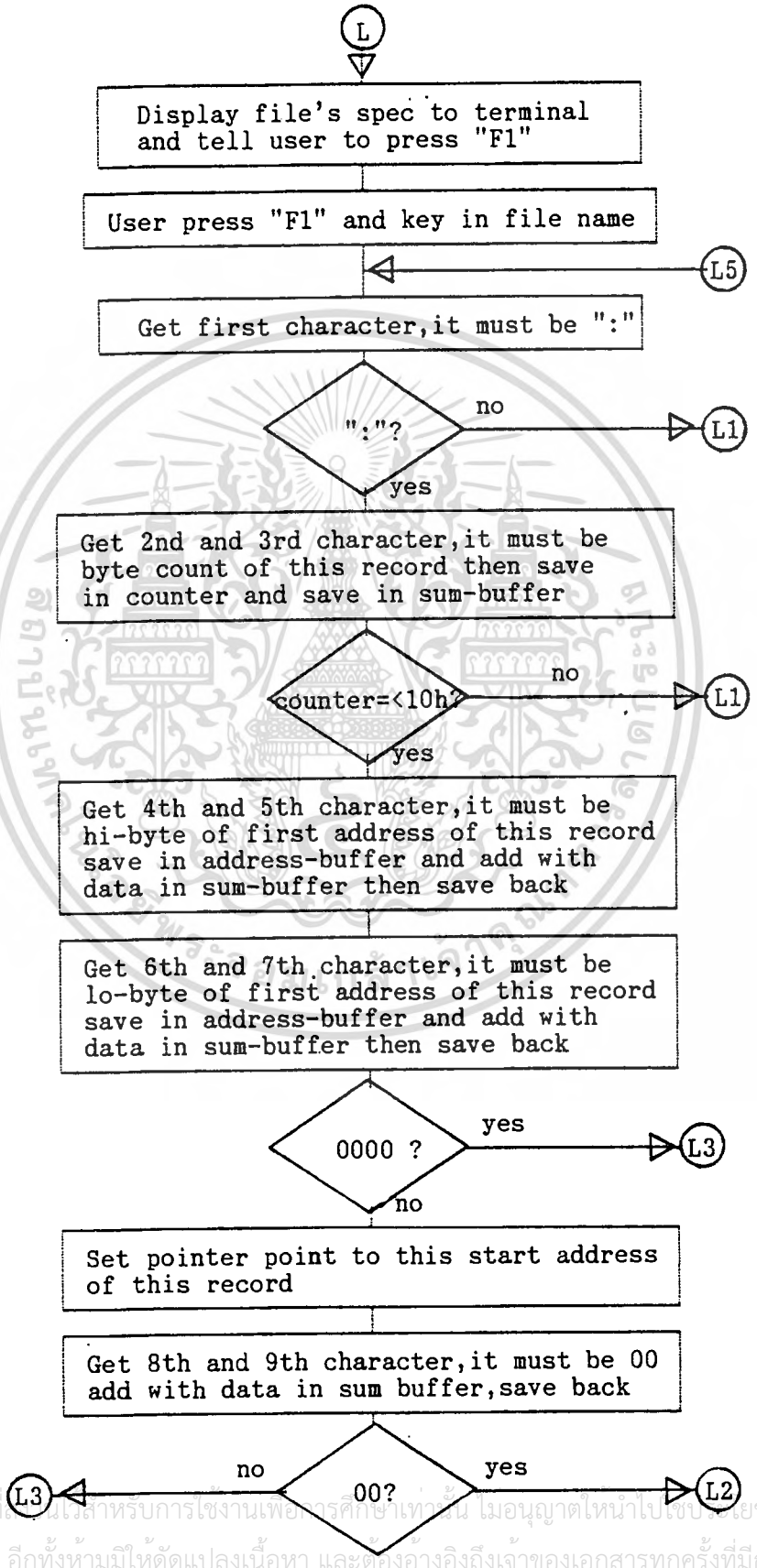
รูปที่ 2.19 แสดงโฟล์ชาร์ทของคำสั่ง I(nsert)

```
Routine ; J(ump table)
Syntax  ; J
Detail  ; Display jump address in monitor ROM on terminal.
```

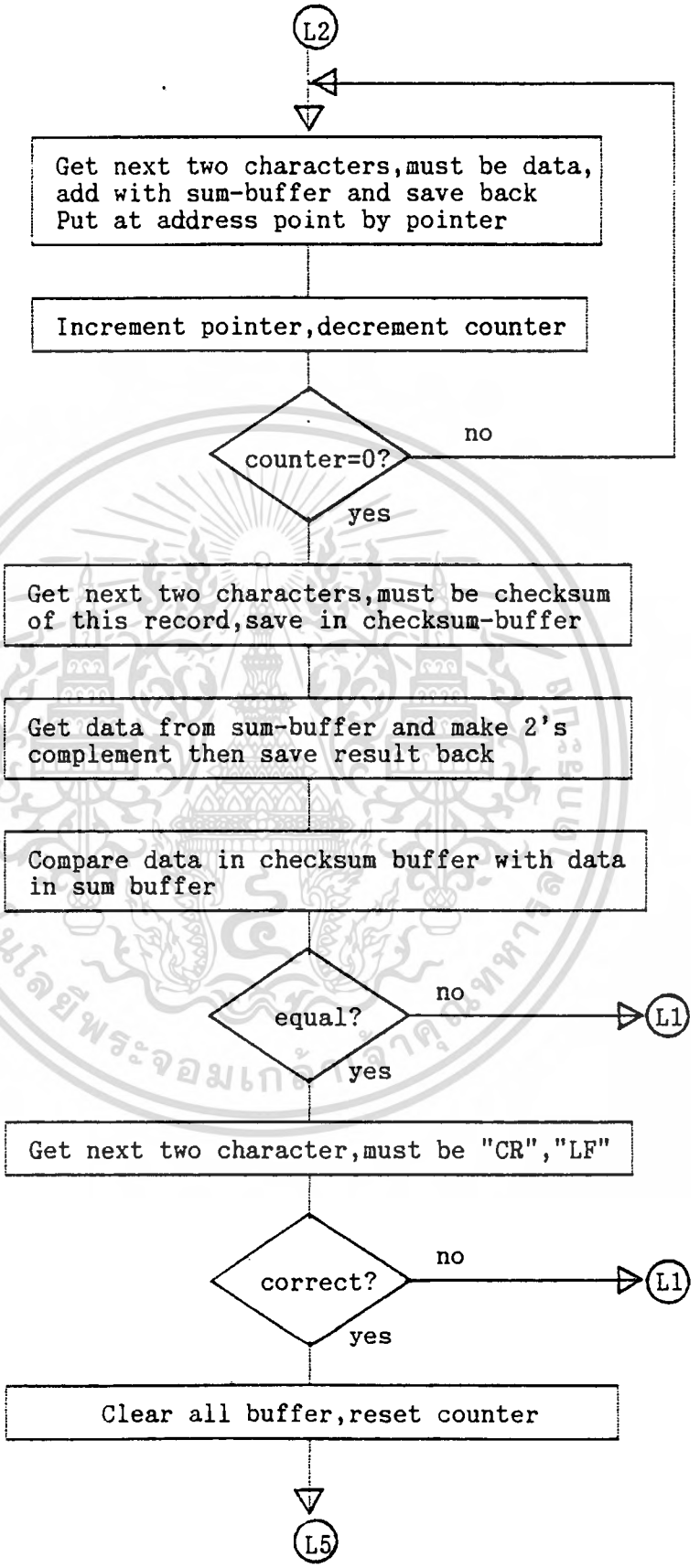


รูปที่ 2.20 แสดงโฟล์ชาร์ทของคำสั่ง J(ump)

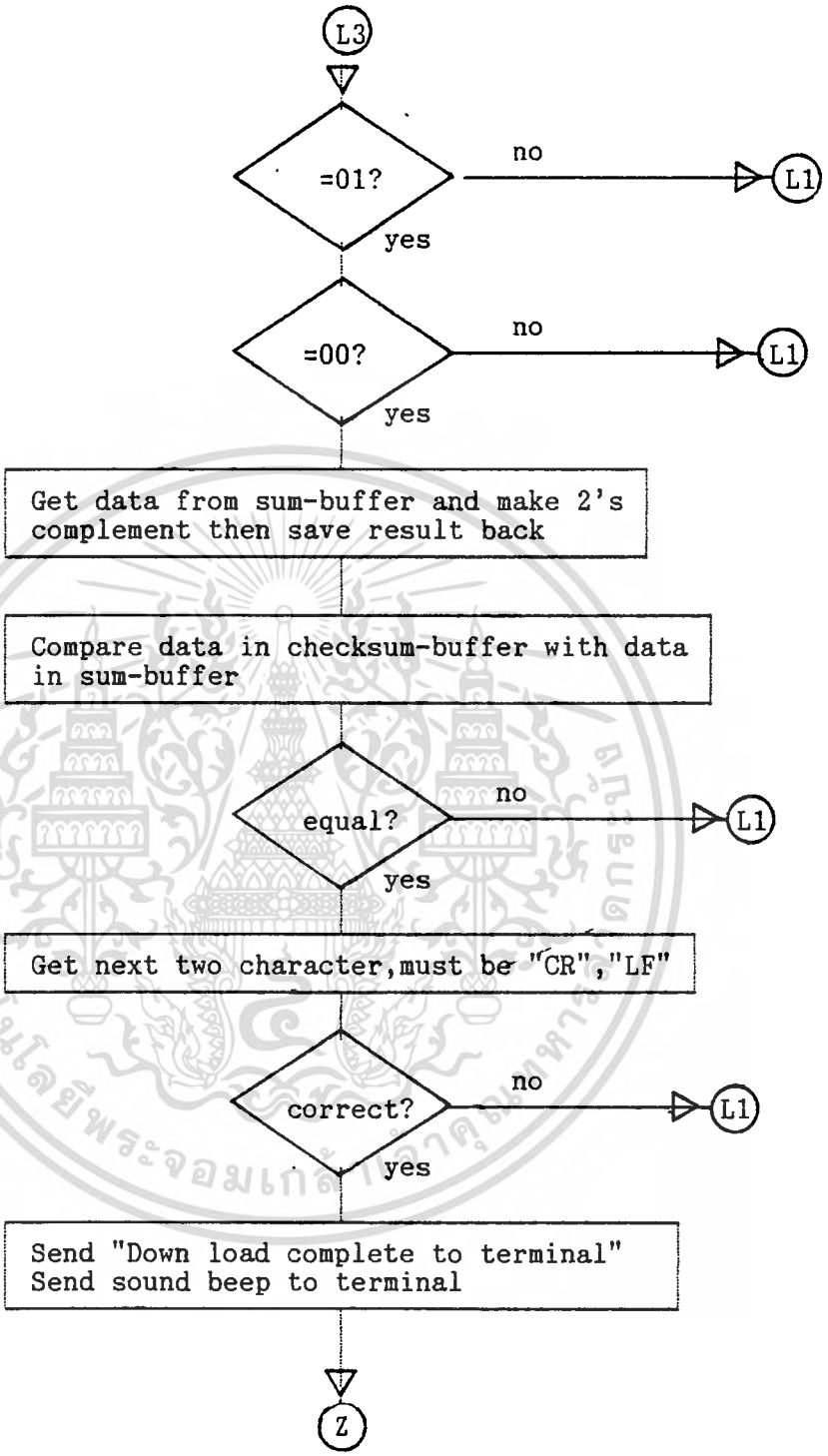
Routine ; L(oad)
Syntax ; L
detail ; Load user program from terminal (hex intel format file).



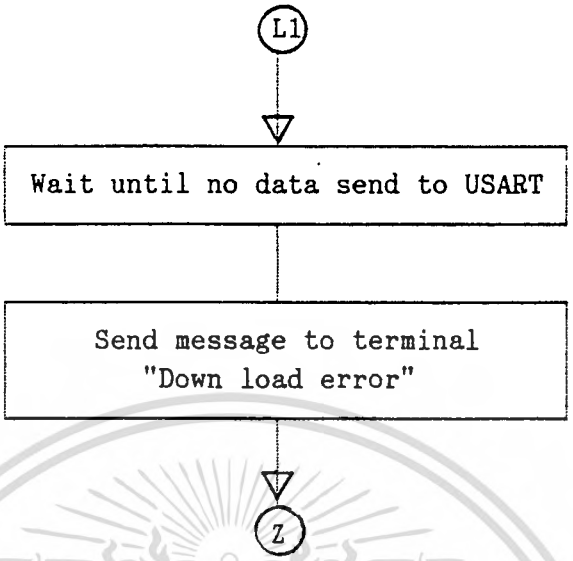
รูปที่ 2.21ก แสดงโฟล์ชาร์ทของคำสั่ง L(oad)



รูปที่ 2.21ข แสดงโฟล์ชาร์ทของคำสั่ง L(oad)



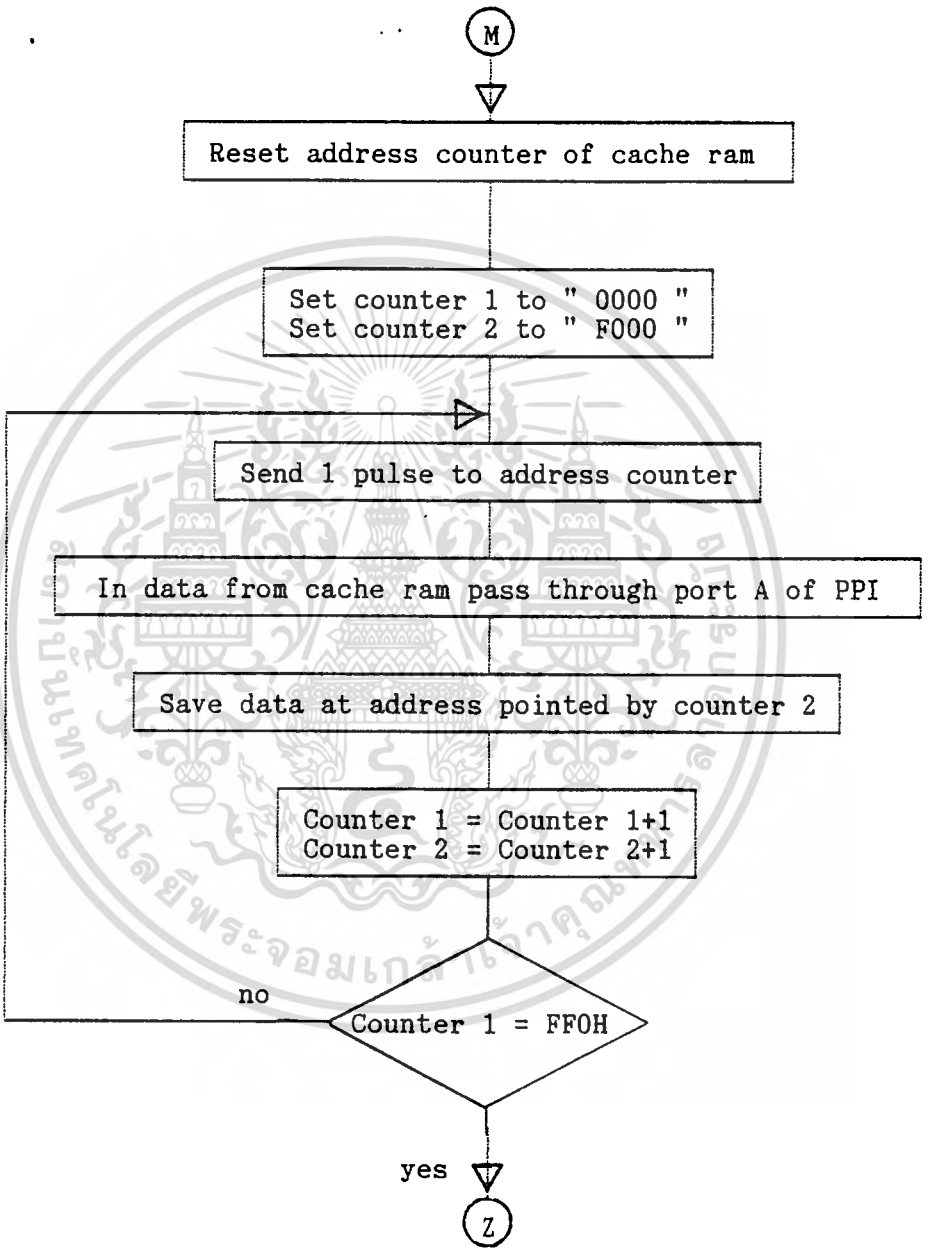
รูปที่ 2.21ค แสดงโฟล์ชาร์ทของคำสั่ง L(oad)




```
INTEL HEX FORMAT

DATA RECORD:

BYTE 1        COLON(:).
     2..3     NUMBER OF DATA BYTE IN THIS RECORD .
     4..5     LOAD ADDRESS FOR THIS RECORD, HIGH BYTE.
     6..7     LOAD ADDRESS FOR THIS RECORD, LOW BYTE.
     8..9     RECORD TYPE,MUST BE "00".
     10..X    DATA BYTE, TWO ASCII-HEX CHARRACTERS.
     X+1..X+2 CHECKSUM, TWO ASCII-HEX CHARACTERS.
     X+3..X+4 CR,LF.

END RECORD:

BYTE 1        COLON(:).
     2..3     RECORD LENGTH, MUST BE "00"
     4..7     START ADDRESS, "0000"
     8..9     RECORD TYPE, "01" OR "00" IS ACCEPTABLE.
     10..11   CHECK SUM.
     12..13   CR,LF.

THE CHECKSUM IS THE TWO'S COMPLEMENT OF 8-BIT SUM,WITHOUT CARRY,
OF ALL THE DATA BYTE, THE TWO BYTE IN THE LOAD ADDRESS, AND
THE BYTE COUNT.
```

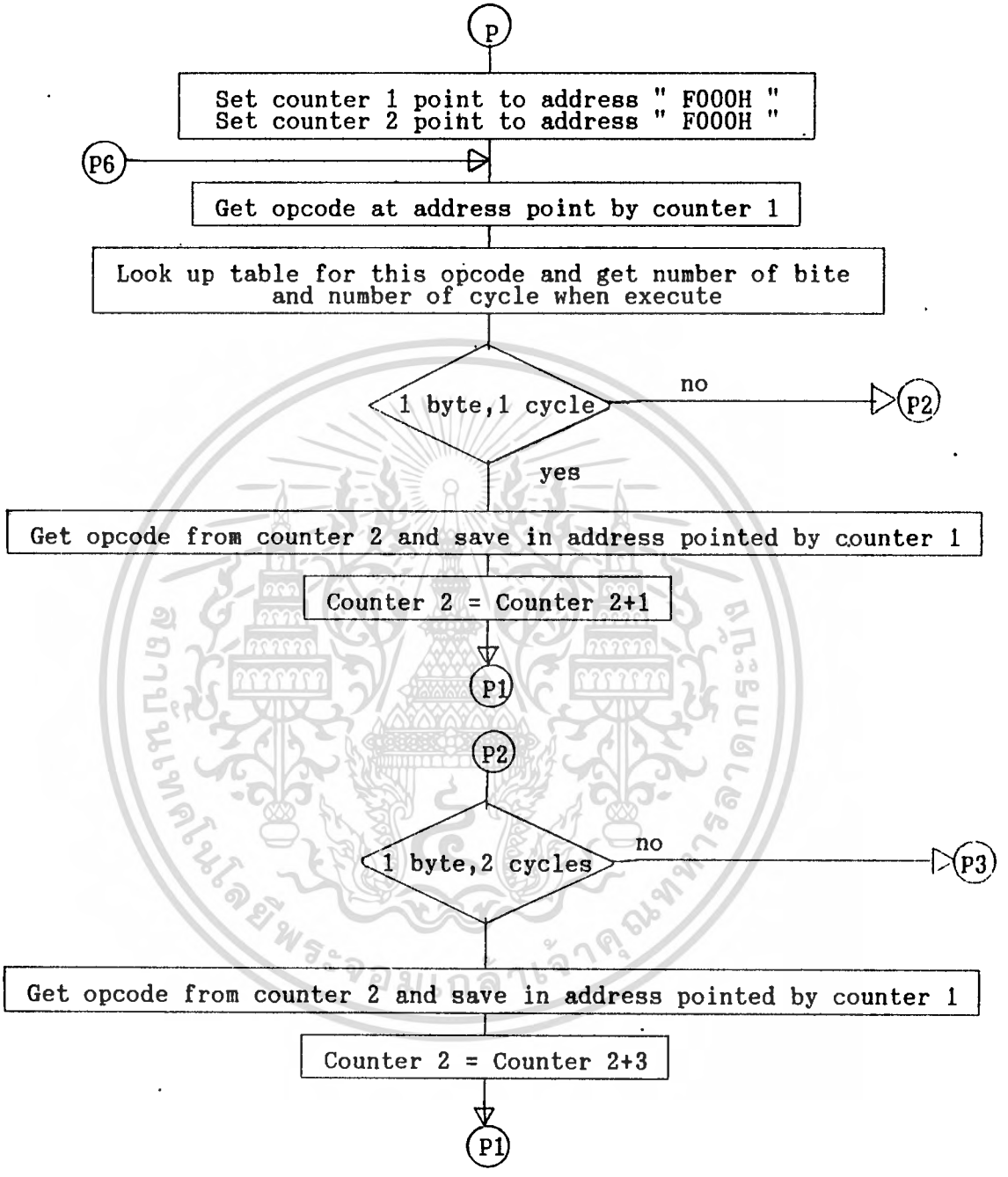
รูปที่ 2.21ง แสดงโฟล์ชาร์ทของคำสั่ง L(oad)

```
Routine ; M(ove)
Syntax  ; M
Detail  ; Move data from cache ram to data ram address F000H
```

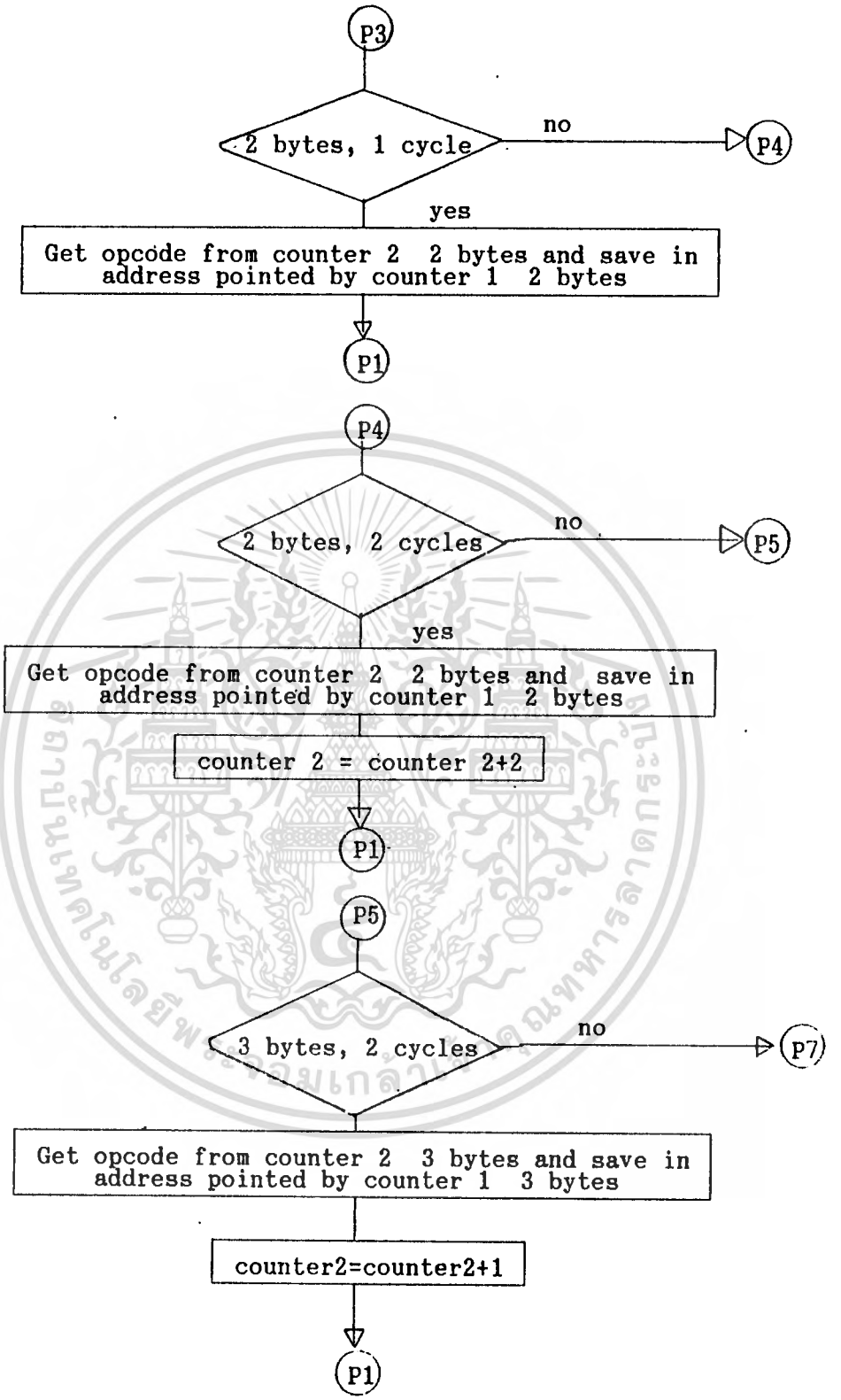


รูปที่ 2.22 แสดงโฟล์ชาร์ทของคำสั่ง M(ove)

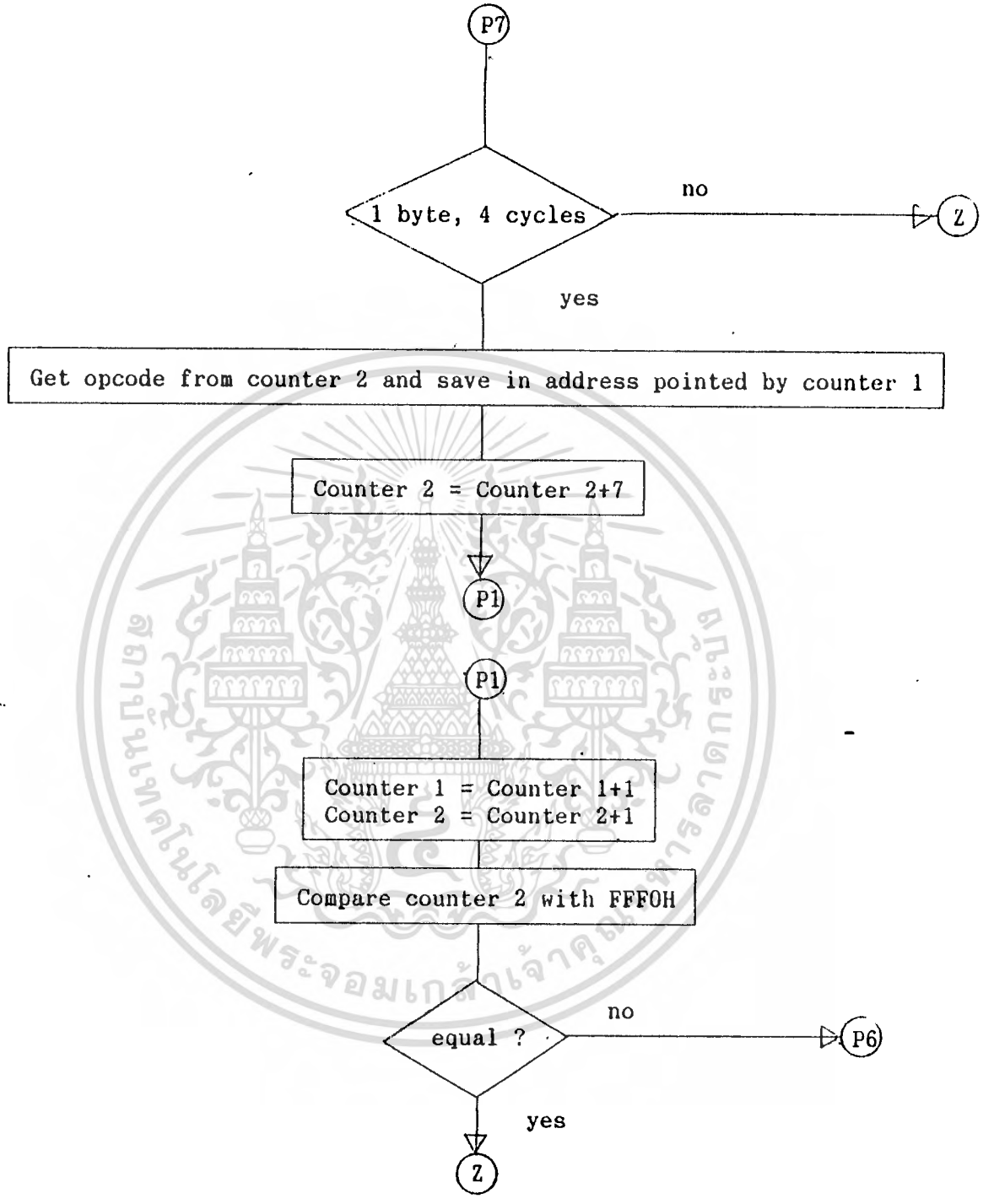
```
Routine ; P(reprocessor)
Syntax  ; P
Detail  ; Remove and adjust data which capture by cache ram
          when executed by 8051 microprocessor.
```



รูปที่ 2.23ก แสดงโฟล์ชาร์ทของคำสั่ง P(reprocessor)

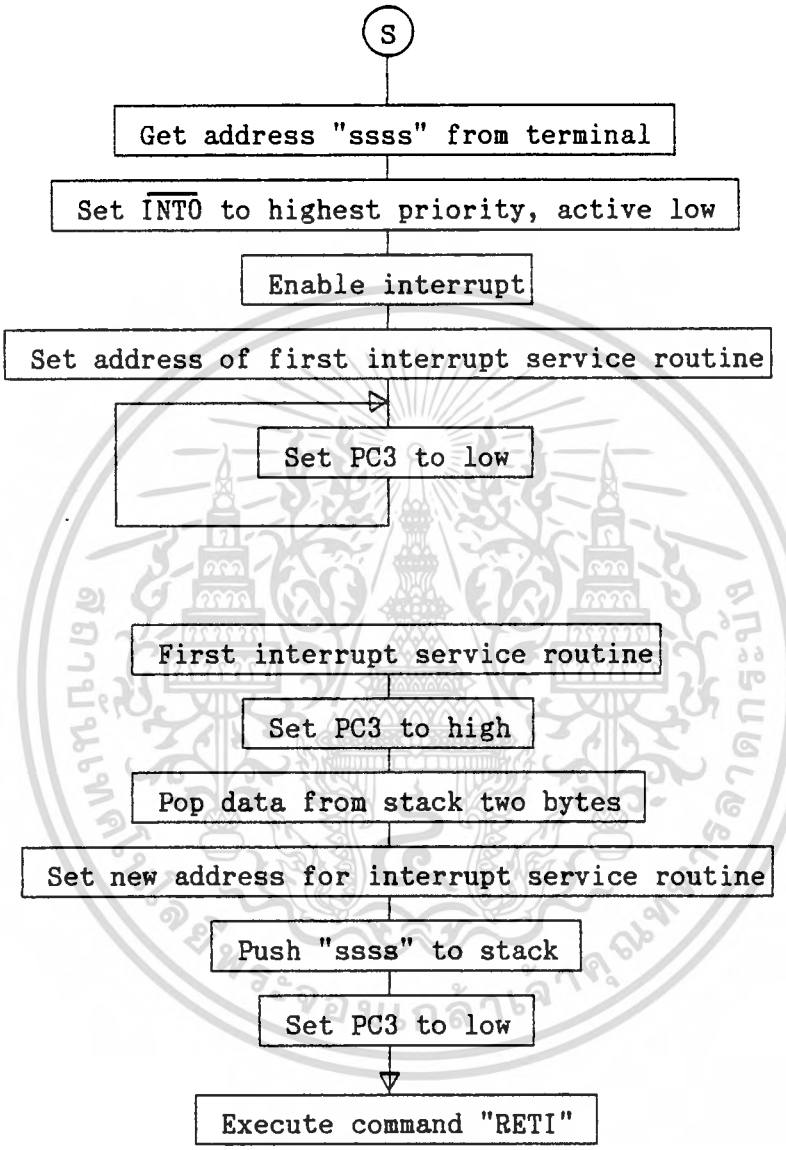


รูปที่ 2.23ข แสดงฟลัชชาร์ทของคำสั่ง P(reprocessor)

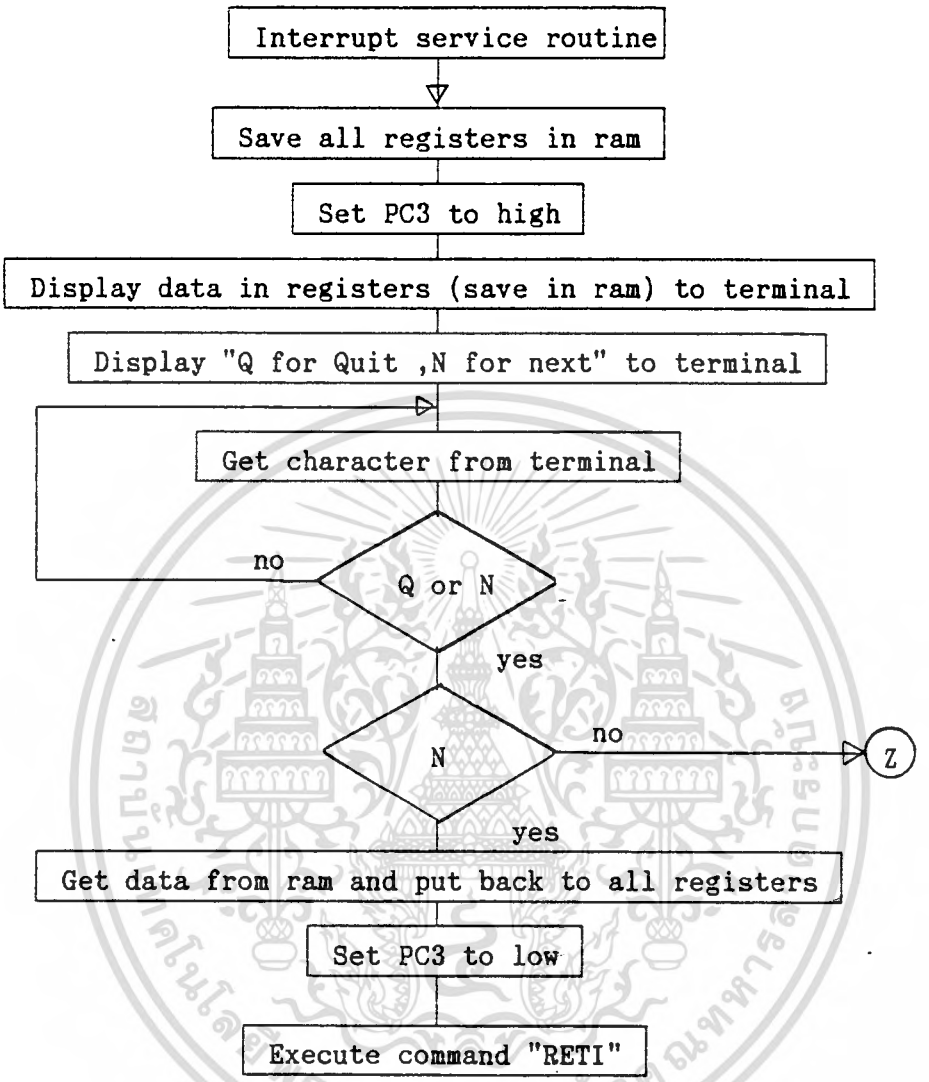


รูปที่ 2.23ค แสดงโฟล์ชาร์ทของคำสั่ง P(reprocessor)

```
Routine ; S(ingle step)
Syntax  ; S ssss
Detail  ; Run single step, start at address "ssss"
```

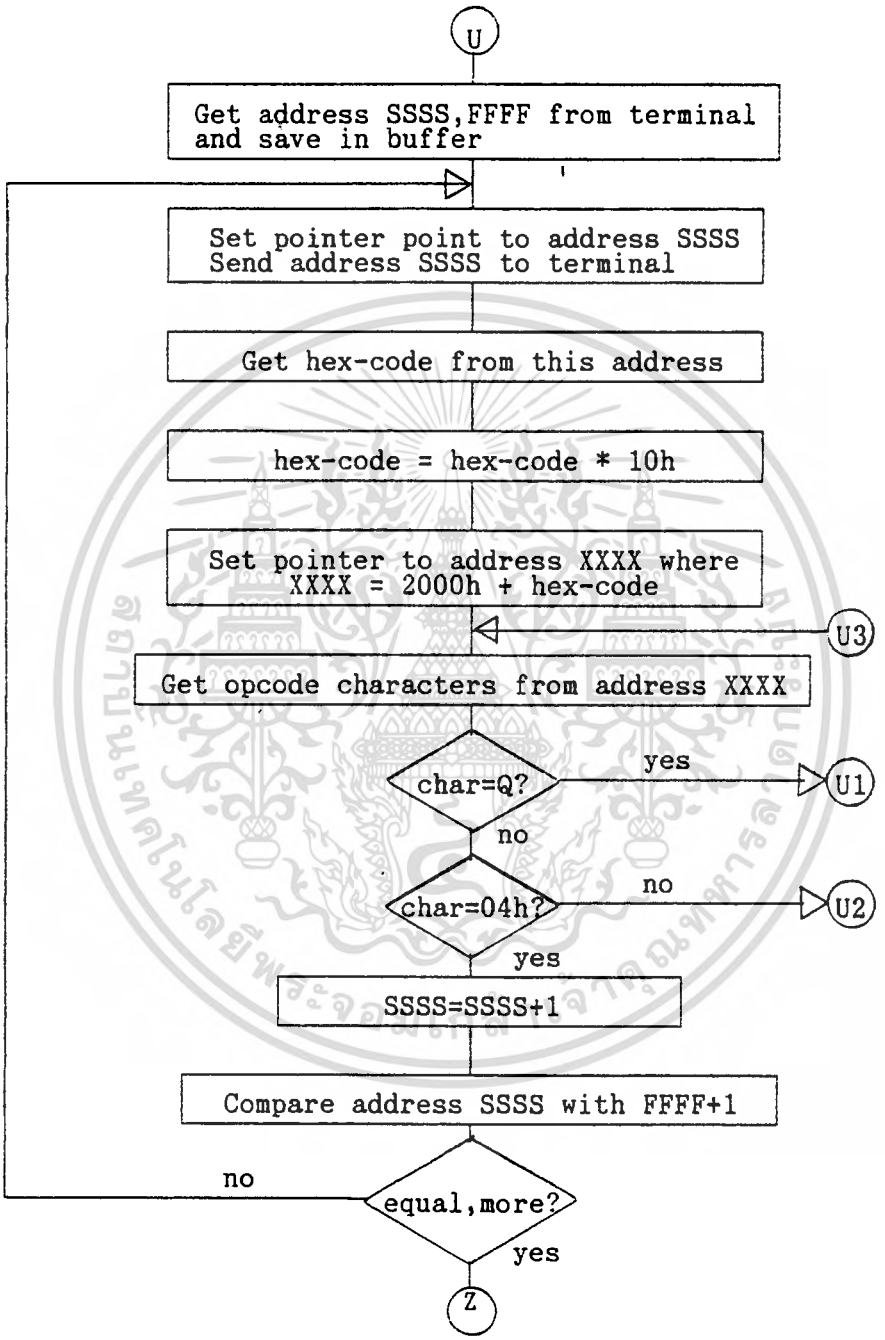


รูปที่ 2.24ก แสดงโฟล์ชาร์ทของคำสั่ง S(ingle)



รูปที่ 2.24ข แสดงโฟล์ชาร์ทของคำสั่ง S(ingle)

```
Routine ; U(assembler)
Syntax  ; U SSSS FFFF
Detail  ; Disassembler hex-code from specified memory
```



รูปที่ 2.25ก แสดงโฟล์ชาร์ทของคำสั่ง U(nassembler)

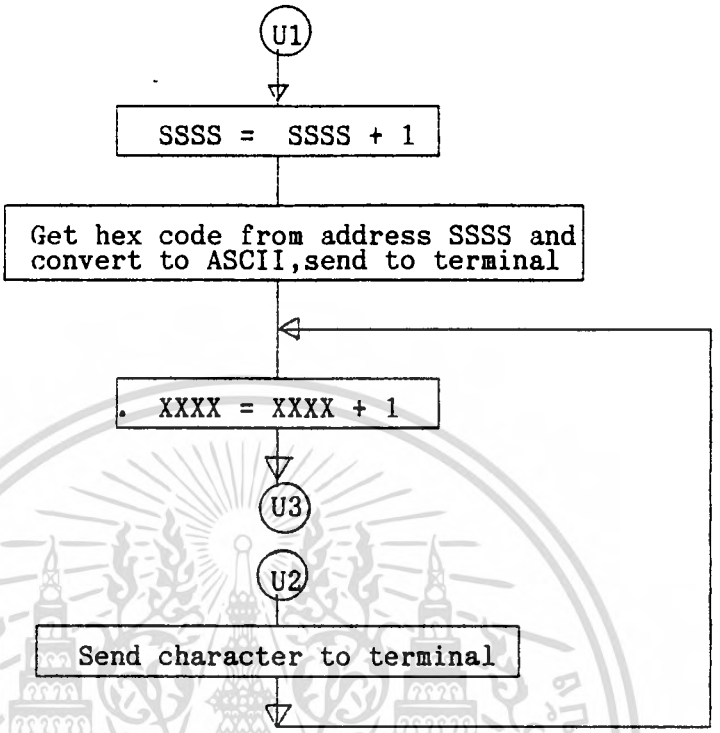


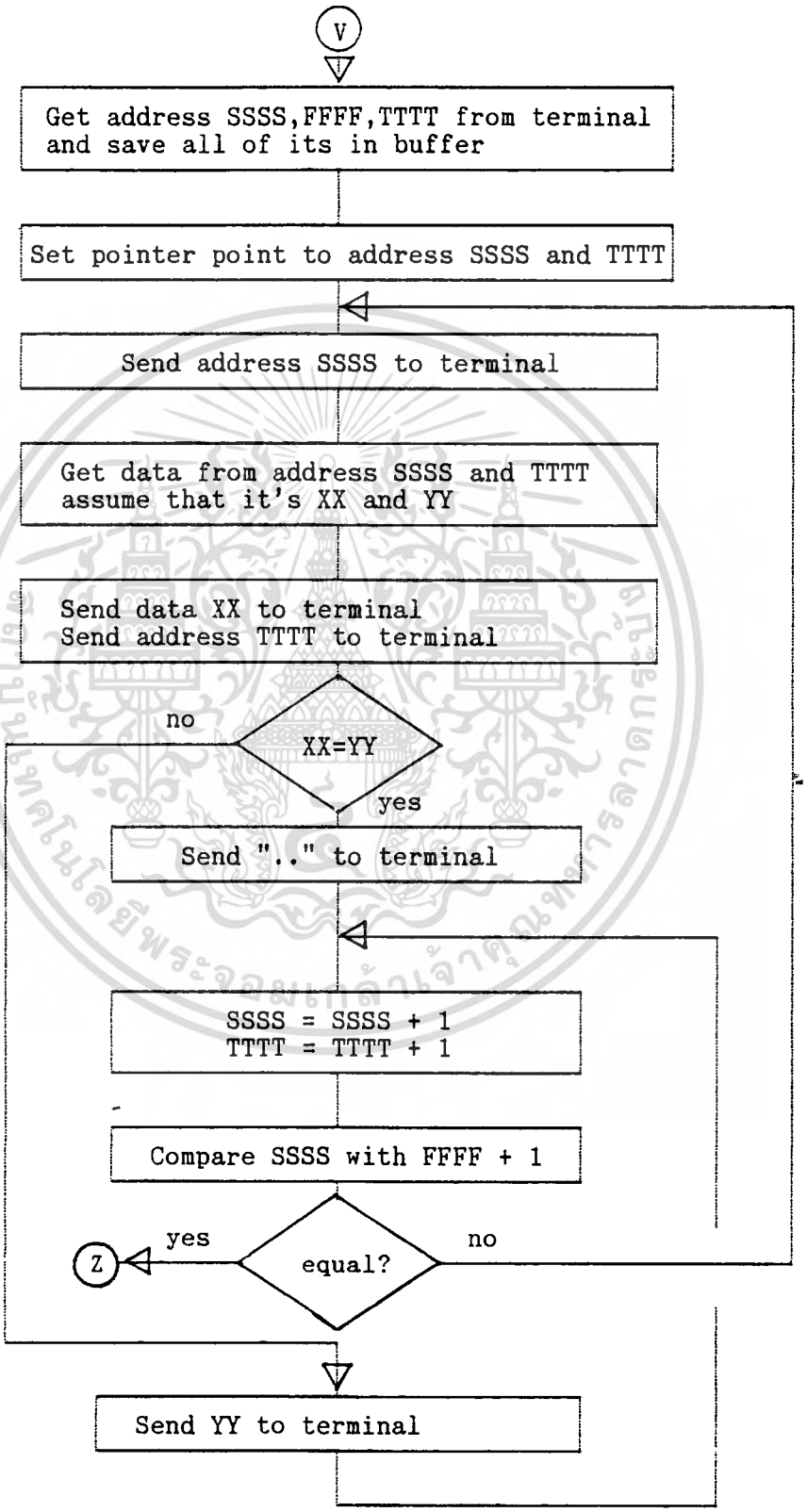
TABLE OF ASSEMBLY LANGUAGE

- Table start at address 2000h
- At these locations divided in 10h bytes per block
- One block saved pattern of one instruction like followed

address	content	detail
2000	"N"	Begin one instruction
2001	"O"	
2002	"P"	
2003	04	End one instruction
⋮		
2010	"A"	Begin next instruction
2011	"J"	
2012	"M"	
2013	"P"	
2014	" "	
2015	"Q"	Need operand follow
2016	"Q"	Need operand again
2017	04	End one instruction
⋮		

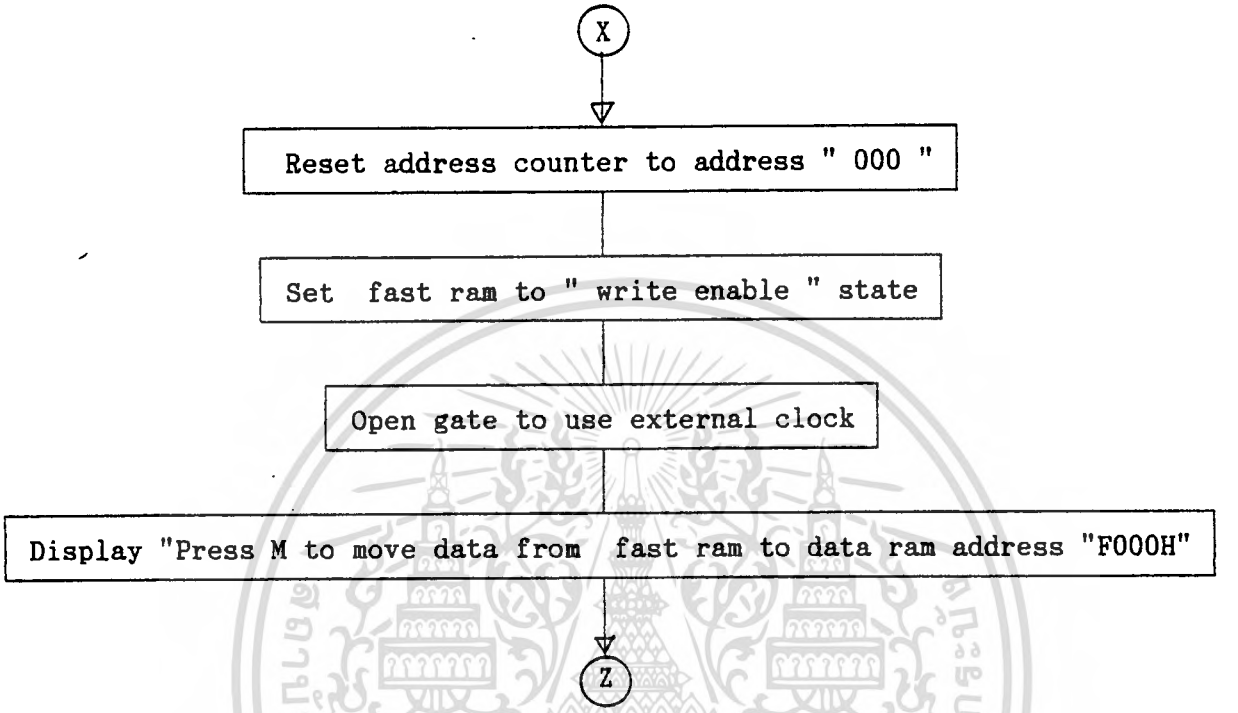
รูปที่ 2.25ข แสดงโฟล์ชาร์ทของคำสั่ง U(nassembler)

Routine ; V(erify)
Syntax ; V SSSS FFFF TTTT
Detail ; Compare data in two block of memory.



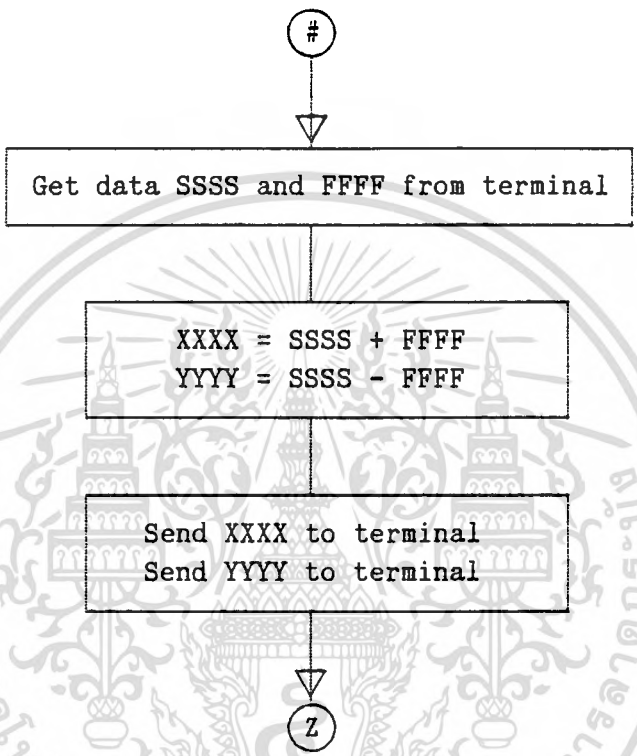
รูปที่ 2.26 แสดงโฟล์ชาร์ทของคำสั่ง V(erify)

Routine ; External capture
Syntax ; X
Detail ; Use data capture unit as logic analyzer



รูปที่ 2.27 แสดงโฟล์ชาร์ทของคำสั่ง eX(ternal)

```
Routine ; #
Syntax  ; # SSSS FFFF
Detail  ; Display result of add and substract between SSSS and FFFF
```



รูปที่ 2.28 แสดงโฟล์ชาร์ทของคำสั่ง #

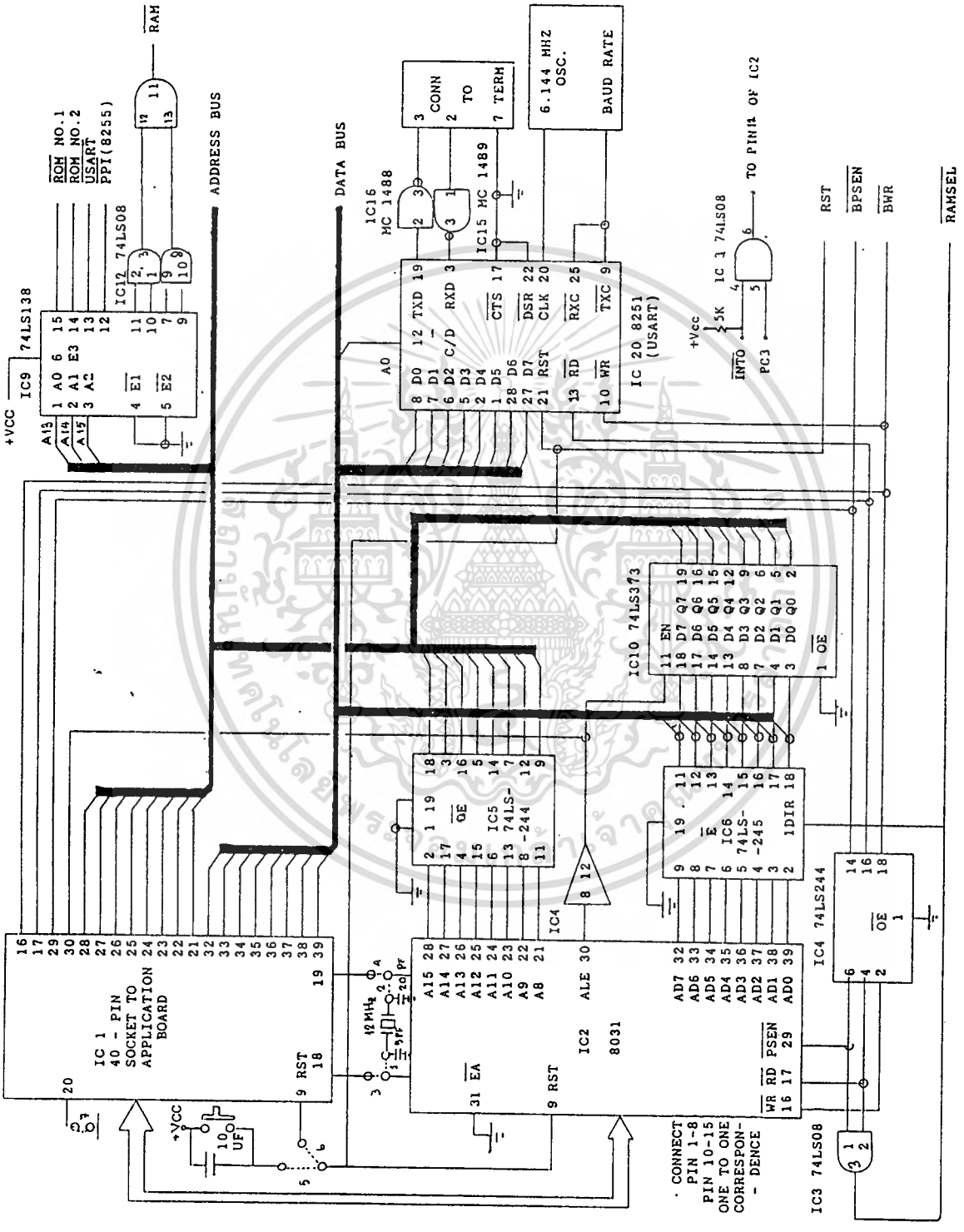
บทที่ 3 วงจรที่ใช้ในการทดลองและการทดลองใช้งาน

3.1 รายละเอียดของวงจรที่ใช้งาน

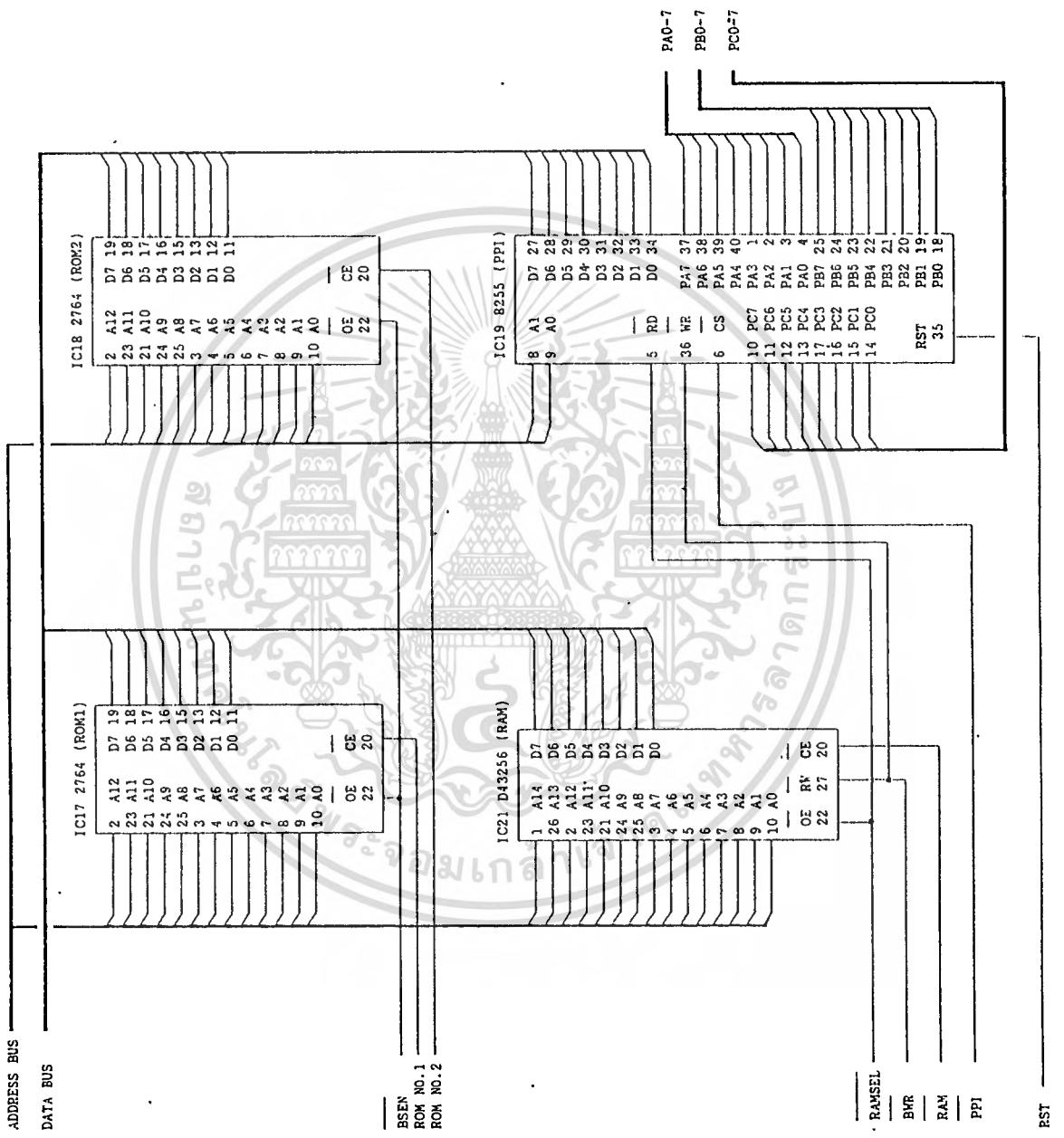
สำหรับวงจรที่สร้างขึ้นจะเป็นดังรูปที่ 3.1 3.2 3.3 และรูปที่ 3.4 โดยเริ่มพิจารณาจากรูปที่ 3.1 และ 3.2 ก่อน IC2 เป็นไมโครคอมพิวเตอร์แบบชิพเดียวเบอร์ 8031 เนื่องจากต้องการใช้หน่วยความจำภายนอกดังนั้นขา 31 จึงถูกต่อลงกราวด์ ขา 21 ถึง 28 เป็นแอดเดรสบัสใช้ IC5(74LS244) เป็นตัวบัฟเฟอร์ IC6(74LS245) ทำหน้าที่แอดเดรสและดาต้าบัฟเฟอร์ให้ขา 32 ถึง 39 ของไมโครโปรเซสเซอร์ IC6 จะถูกควบคุมทิศทางการส่งออกหรือรับเข้าด้วยเอาต์พุทของแอนด์เกตโดยมีอินพุทจากขา 17($\overline{RD}$) และขา 29($\overline{PSEN}$) ขา 30(ALE) ขา 17($\overline{RD}$) และขา 29($\overline{PSEN}$) จะถูกบัฟเฟอร์ด้วย IC4(74LS244) สัญญาณที่ผ่านการบัฟเฟอร์แล้วจะถูกต่อไปยัง IC1 ซึ่งเป็นเต้ารับขนาด 40 ขา โดยเรียงตามขาของ 8031 ส่วนขาที่เหลืออื่นๆ ถูกต่อไปยัง IC1 โดยตรงยกเว้นขา 9 และขา 18 และ 19 ซึ่งต่อกับก้อนผลึกจะมีสวิตช์ให้เลือกว่าต้องการต่อมาจากบอร์ดที่ถูกอีมูเลทหรือจากบอร์ดของอีมูเลเตอร์เอง

สัญญาณมัลติเพล็กซ์ระหว่างแอดเดรสและข้อมูลที่ได้จาก IC6 ถูกแยกสัญญาณแอดเดรสออกโดย IC10 (74LS373) โดยมีสัญญาณ ALE เป็นตัวควบคุม เอาต์พุทที่ได้จาก IC10 จะเป็นสัญญาณของแอดเดรสอย่างเดียว

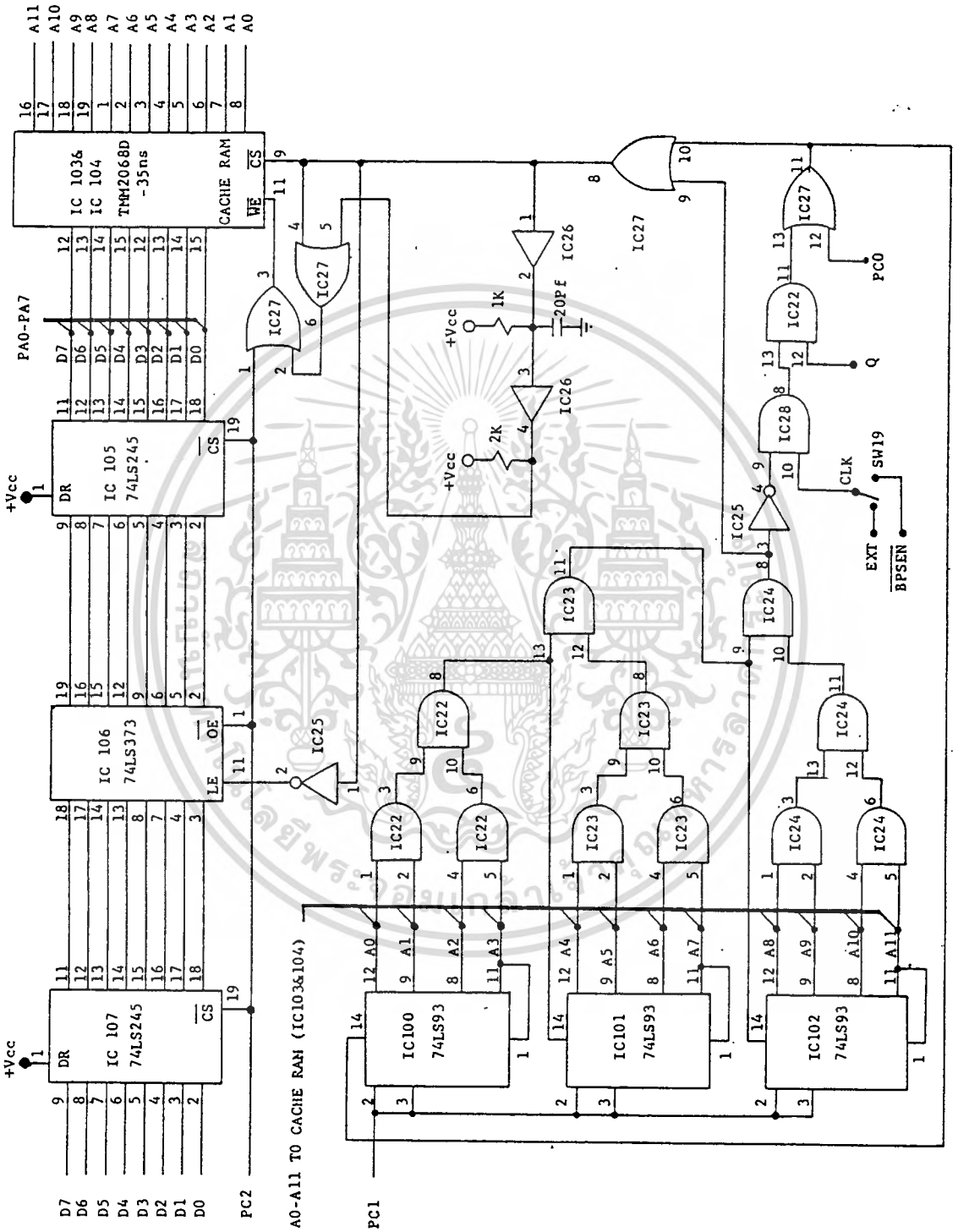
สายแอดเดรสเส้นที่ 13 14 และ 15 ถูกถอดระหัสด้วย IC9 (74LS138) ซึ่งจะให้เอาต์พุทบล็อกละ 8 กิโลไบต์ โดยบล็อกที่ 1 และ 2 จะเป็นแอดเดรสของ EPROM บล็อกที่ 3 เป็นของ USART (Universal Synchronous Asynchronous Receiver and Transmitter) ซึ่งทำหน้าที่เปลี่ยนข้อมูลแบบขนานเป็นแบบอนุกรมออกไปที่ขา 19 และรับข้อมูลแบบอนุกรมจากขาที่ 3 แล้วเปลี่ยนให้เป็นแบบขนานส่งเข้าสู่บัสข้อมูล ส่วน IC15 และ IC16 (MC1488 และ MC1489) เป็นตัวเปลี่ยนระดับของสัญญาณให้เข้าสู่มาตรฐานของ RS-232 ขา 12 ของ USART ต่ออยู่กับแอดเดรส A0 เพื่อระบุว่า ข้อมูลที่ส่งเข้าไป เป็นข้อมูลของคำสั่งหรือข้อมูลที่ต้องการส่งออก หรือเป็นข้อมูลรับเข้าจากพอร์ทอนุกรม ขาข้อมูลทั้งหมดต่อมาจากเอาต์พุทของ IC6 ขา13 ($\overline{RD}$) ขา10 ($\overline{WR}$) ต่อมาจากเอาต์พุทของ IC4 ส่วนขา 21(RST) ต่อมาจาก 8031 สำหรับขา 9 ขา 20 และขา 25



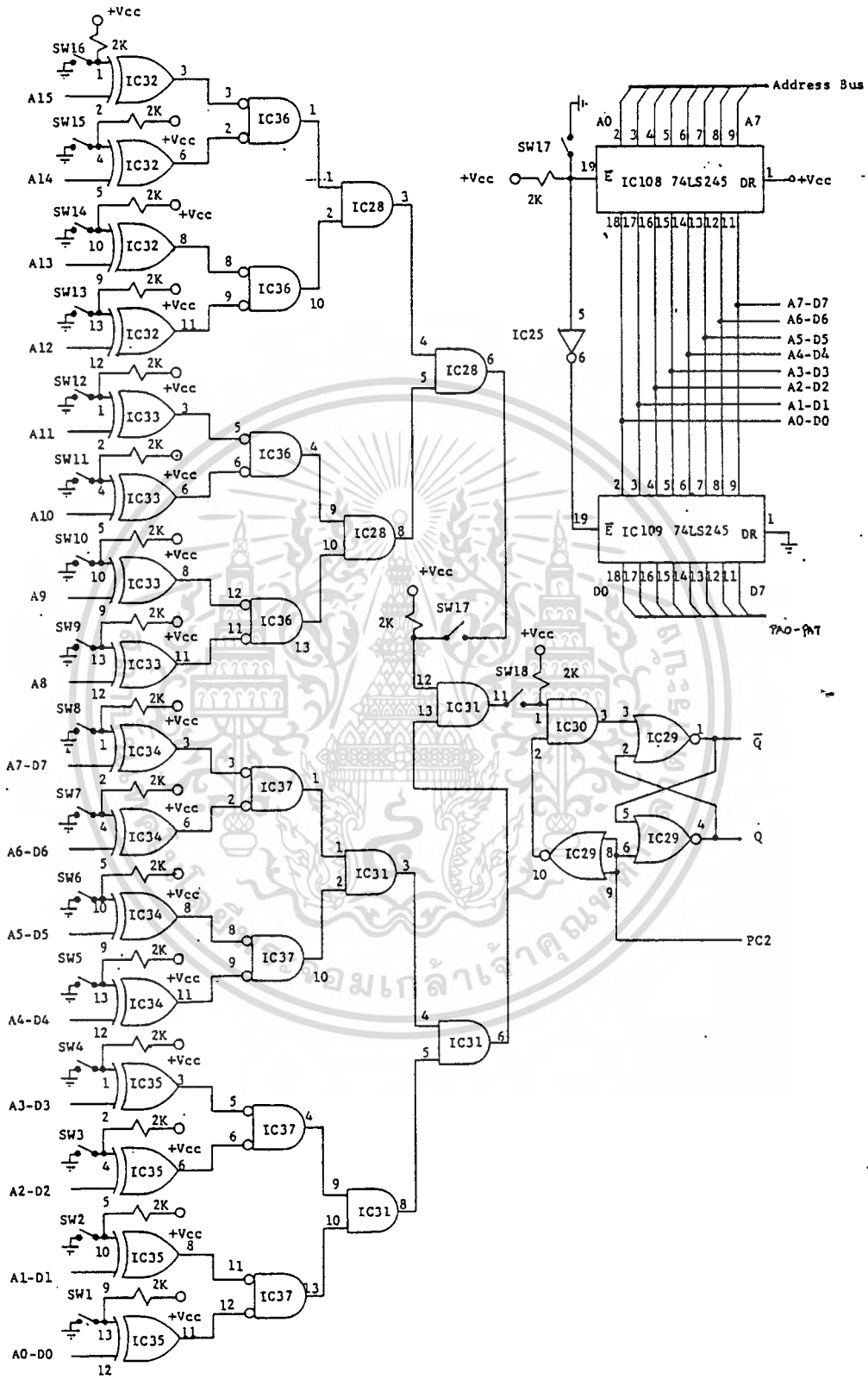
รูปที่ 3.1 แสดงวงจรของ 8051 อีมูเลเตอร์



รูปที่ 3.2 แสดงวงจรของ 8051 อีมูเลเตอร์



รูปที่ 3.3 แสดงวงจรของส่วนดักจับข้อมูลที่ประกอบร่วมกับ 8051 อีมูเลเตอร์



รูปที่ 3.4 แสดงวงจรของส่วนดักจับข้อมูลที่ประกอบร่วมกับ 8051 อีมูเลเตอร์

ต่ออยู่กับส่วนกำเนิดสัญญาณไบด์ (Baud-rate Generator) เพื่อกำหนดอัตราไบด์ในการรับและส่งข้อมูล หน่วยความจำชนิดแรมของระบบขนาด 32 กิโลไบต์ใช้สัญญาณควบคุมจาก IC9 ซึ่งนำมาแอนด์กันเพื่อที่จะครอบคลุมได้ถึง 32 กิโลไบท์

รูปที่ 3.3 และ 3.4 เป็นส่วนที่ใช้จับสัญญาณลอจิกเพื่อนำไปวิเคราะห์หาข้อผิดพลาดซึ่งประกอบด้วย IC100 IC101 และIC102 รวมกับแอนด์เกททำหน้าที่รวมกันเป็นวงจรนับแอดเดรส (Address Counter) ให้กับฟาสแรม โดยที่แอดเดรสจะมีค่าเพิ่มขึ้นทีละหนึ่งเมื่อได้รับสัญญาณนาฬิกาจากภายนอก (กรณีที่ต้องการเขียนข้อมูล)หรือจากพอร์ท PC0 ของ IC19 (ในกรณีที่ต้องการอ่านข้อมูลจากฟาสแรมเข้ามายังแรมของระบบผ่านทางพอร์ท PA0 ถึง PA7 ของ IC19) สายสัญญาณของวงจรนับแอดเดรสจะถูกต่อไปยัง IC103 และ IC104 ซึ่งเป็นฟาสแรมขนาด 4 กิโลไบต์ 4 บิตซึ่งต่อกัน 2 ตัวเพื่อเพิ่มขนาดเป็น 4 กิโลไบท์ 8 บิท ขาข้อมูลของฟาสแรมจะถูกต่ออยู่กับ PA0 ถึง PA7 ของ IC19 และยังถูกต่อกับ IC105 ซึ่งเป็นบัฟเฟอร์ของข้อมูลจาก IC106 ในกรณีที่อ่านข้อมูลเข้าอีมูเลเตอร์ IC105 จะถูกควบคุมด้วยพอร์ท PC2 ให้อยู่ในสภาวะอิมพีแดนซ์สูง (High Impedance) เพื่อป้องกันการรบกวนระหว่างข้อมูลเก่าและข้อมูลที่อ่านจากฟาสแรม IC107 ทำหน้าที่บัฟเฟอร์ข้อมูลที่จับได้จากภายนอกและส่งต่อให้กับ IC106 ซึ่งจะแลทช์ (Latch) ข้อมูลไว้เพื่อเขียนลงในฟาสแรม

สัญญาณที่ได้จาก IC105 อีกส่วนหนึ่งจะถูกส่งไปยังส่วนจุดชนวน (Trigger Unit) ซึ่งแบ่งออกเป็นการจุดชนวนแบบไบท์และการจุดชนวนแบบคำ เพื่อทำหน้าที่ตรวจหาข้อมูลหรือแอดเดรสที่กำหนดโดยผู้ใช้ผ่านทางสวิตช์ ถ้าข้อมูลหรือแอดเดรสที่ตรวจจับได้มีค่าเหมือนกับที่ผู้ใช้กำหนด จะทำให้เอาต์พุทของแอนด์เกททุกตัวมีลอจิกเป็น 1 ทั้งหมด ทำให้ อาร์-เอส ฟลิป-ฟลอป ถูกเซ็ต จะทำให้สัญญาณนาฬิกาสามารถผ่านไปยังวงจรนับแอดเดรส และแคชแรมทำการเริ่มเก็บข้อมูล อาร์-เอส ฟลิป-ฟลอป จะถูกรีเซ็ตโดย PC2 ของ IC19 เมื่อเริ่มทำการเก็บข้อมูลในครั้งถัดไป

สำหรับส่วนของโปรแกรมได้แสดงไว้ในภาคผนวกที่ 2 และ 3 โดยภาคผนวกที่ 2 จะเป็นโปรแกรมที่ใช้เครื่อง IBM-PC ส่วนภาคผนวกที่ 3 จะเป็นโปรแกรมที่ใช้ควบคุม 8031 อีมูเลเตอร์

3.2 ขั้นตอนการใช้งานของ 8051 อีมูเลเตอร์

3.2.1 ด้าน IBM-PC ทำการรันโปรแกรมที่มีชื่อว่า Terminal .EXE เป็นอันดับแรกโดย

โปรแกรมจะให้เลือกพอร์ทของ RS-232 และเลือกอัตราโบด์ที่ต้องการใช้ดังรูปที่ 3.5 ทำการเลือกพอร์ทให้สอดคล้องกับเครื่อง IBM PC และเลือกอัตราโบด์เช่นเดียวกับที่ตั้งไว้ใน 8051 อีมูเลเตอร์ ขั้นตอนนี้เครื่อง IBM PC จะส่งสัญญาณ Space (20H) ไปยังบอร์ด 8051 อีมูเลเตอร์เพื่อทดสอบการเชื่อมต่อสัญญาณ

```
*********************************************
*                                           *
*              Charin  Bantukul.            *
*    Communication Link between 8051 & IBM  *
*          @ Copyright 1985, 1986           *
*        Version 1.5, June 1, 1986          *
*                                           *
*********************************************

        ** PRESS F3 to display function keys **

Which port do you wish to use? [1]

Which baud rate do you wish to use? [ 9600]
     0)   300
     1)  1200
     2)  1800
     3)  2400
     4)  4800
     5)  9600
     6) 19200
```

รูปที่ 3.5 แสดงการกำหนดพอร์ทและบอดเรท

ด้าน 8051 อีมูเลเตอร์ เมื่อผ่านขั้นตอนของ IBM PC แล้วให้ทำการป้อนไฟให้แก่วงจร 8051 อีมูเลเตอร์จะทำงานและส่งตัวอักษรเริ่มต้น พร้อมทั้งสัญลักษณ์พรอมพ์ (EM51>) ออกไปยังเครื่อง IBM PC ถ้าเครื่องหมาย "EM51>" ปรากฏบนจอภาพแล้วแสดงว่าพอร์ทสื่อสารทั้งสองด้านได้เชื่อมต่อเข้าหากันเรียบร้อยแล้ว และ 8051 อีมูเลเตอร์พร้อมที่จะรับคำสั่งต่อไป ข้อความบนจอภาพจะเป็นดังรูปที่ 3.6

```
MCS-51 Real Time In Circuit Emulator version 1.00
Copyright 1986 by CHARIN  B. of KMITL

EM51>
```

รูปที่ 3.6 แสดงความพร้อมที่จะปฏิบัติงานของ 8051 อีมูเลเตอร์

3.2.2 ทดลองเรียกดูคำสั่ง H(elp)

8051 อีมูเลเตอร์จะแสดงรายละเอียดของคำสั่งที่ใช้งานแสดงขึ้นบนจอภาพดังรูปที่ 3.7

```
LAST 20 BYTES OF EXT. RAM USED BY BREAK ROUTINE
RAM @ 48H-50H IS USED BY MONITOR
RAM ABOVE 50H IS USED FOR STACK
COMMANDS ARE
A   AAAA                    GO @ AAAA AND CAPTURE DATA
B   {AAAA}                  BREAK AT ADRESS AAAA
C   SSSS FFFF TTTT          COPY BLOCK OF MEMORY
D   SSSS {FFFF}             DUMP PROGRAM MEMORY
DI  SS {FF}                 DUMP INTERNAL MEMORY
E   SSSS                    EDIT EXTERNAL MEMORY
EI  SS                      EDIT INTERNAL MEMORY
F   SSSS FFFF HH HH HH      FIND DATA 'HH HH HH'
G   {AAAA}                  GO @ AAAA OR BREAKPOINT
H                           HELP
I   SSSS FFFF HH            INSERT 'HH' INTO MEMORY
J                           LIST JUMP TABLE
L                           DOWNLOAD FROM TERMINAL TO EXT.RAM
M                           MOVE DATA FROM CACHE RAM TO MEMORY
P                           COMMAND FILTER FOR 8051
S   AAAA                    SINGLE STEP @ AAAA
U   SSSS FFFF               UNASSEMBLER EXTERNAL MEMORY
V   SSSS FFFF TTTT          VERIFY BLOCK OF MEMORY
X                           CAPTURED EXT. DATA BY EXT. CLOCK
#   MMMM NNNN
EM51>
```

รูปที่ 3.7 แสดงรายละเอียดของชุดคำสั่งที่ใช้กับ 8051 อีมูเลเตอร์หลังจากกดคีย์ตัวอักษร"H"

3.2.3 ทดลองคำสั่ง L(oad)

คำสั่งนี้จะเป็นการดาวน์โหลดโปรแกรมลงในหน่วยความจำตามที่กำหนดไว้ โดยสมมุติว่าต้องการเขียนโปรแกรมเพื่อให้ 8051 อีมูเลเตอร์ แสดงข้อความ " Hello! Nice to meet you again." ขึ้นมาบนจอภาพ ดังนั้นจะเขียนโปรแกรมได้ว่า

```
A>type sample.asm
          CPU   "8051.TBL"
          HOF   "INT8"
;
         ORG 8000H        ;start address of user's RAM
;
CR:      EQU       0DH
LF:      EQU       0AH
;
START:   MOV DPTR,#HELLO ;pointer to address of pattern
         LCALL 0AD0H     ;put all characters to terminal
         LJMP 0026H      ;jump to show prompt

HELLO:   DFB  CR,LF,"Hello,nice to meet you again.",CR,LF,04H
;
         END
```

หลังจากทำการคอมไพล์แล้วจะได้ไฟล์พร้อมที่จะดาวน์โหลดในรูปของ Intel Hex Format ดังรูปที่ 3.8 สมมุติว่าให้ชื่อไฟล์นี้ว่า "Sample.Hex"

```
A>type sample.hex
:10800000908009120AD00200260D0A48656C6C6F38
:108010002C6E69636520746F206D65657420796FBF
:0B8020007520616761696E2E0D0A0477
:00000001FF
```

รูปที่ 3.8 แสดงตัวอย่างไฟล์ที่พร้อมจะดาวน์โหลด

จากนั้นให้กดคีย์ "L" ที่จอภาพจะบอกให้กดฟังก์ชั่นคีย์ F1 เมื่อพร้อมที่จะทำการดาวน์โหลด นำแผ่นดิสก์ที่มีไฟล์ Sample.Hex บันทึกอยู่ใส่เข้าไปในไดร์ แล้วป้อนชื่อไฟล์ Sample.Hex เมื่อเครื่องทำการดาวน์โหลดเรียบร้อยแล้วจะแสดงผลออกมาบนจอภาพดังรูปที่ 3.9

```
EM51>L
Download from Terminal to 8051 Emulator
File must be intel 8 bits short or long format
Press function key F1 for begin process

driver: Name of download file? SAMPLE.HEX
driver: downloading...
driver: download complete

EM51>
```

รูปที่ 3.9 แสดงการดาวน์โหลดโปรแกรมลงในหน่วยความจำแบบแรม

3.2.4 ทดลองคำสั่ง D(ump)

คำสั่งนี้จะเป็นการแสดงค่าในหน่วยความจำขึ้นมาในจอภาพจากหัวข้อที่ 3.2.3 ถ้าโปรแกรมดาวน์โหลดได้ถูกต้อง เมื่อทำการดัมหน่วยความจำจากตำแหน่งที่ 8000H ถึง 8020H จะเห็นโปรแกรมที่เป็นภาษาเครื่องปรากฏบนจอภาพ ถ้าคีย์คำสั่ง " D 8000 8020" ที่เทอร์มินอล ผลที่แสดงออกมาจะเป็นดังรูปที่ 3.10

```
EM51>D 8000 802F

      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
8000  90 80 09 12 0A D0 02 00  26 0D 0A 48 65 6C 6C 6F     ........ &..Hello
8010  2C 6E 69 63 65 20 74 6F  20 6D 65 65 74 20 79 6F     ,nice to  meet yo
8020  75 20 61 67 61 69 6E 2E  0D 0A 04 FF FF FF FF FF     u again. ........
EM51>
```

รูปที่ 3.10 แสดงการดัมหน่วยความจำในตำแหน่งที่ต้องการขึ้นมาบนจอภาพ

ในทำนองเดียวกันถ้าต้องการดูค่าในหน่วยความจำแบบแรมภายในชิพของ 8031 ก็ใช้คำสั่ง " DI " แทน ซึ่งหมายถึง " Dump Internal " เช่น " DI 80 90 " เป็นต้น

3.2.5 <u>ทดลองคำสั่ง G(o)</u>

คำสั่งนี้เป็นคำสั่งที่บังคับให้ 8051 อีมูเลเตอร์ ไปทำการรันโปรแกรมที่ดาว์นโหลดลงไปยังตำแหน่ง 8000H โดยคีย์คำสั่ง " G 8000 " ผลที่ได้จะแสดงข้อความตามที่ต้องการดังรูปที่ 3.11

```
EM51>G 8000

Hello,nice to meet you again.

EM51>
```

รูปที่ 3.11 แสดงการใช้คำสั่ง G(o) สั่งให้อีมูเลเตอร์รันโปรแกรมที่ตำแหน่งแอดเดรส 8000H

3.2.6 <u>ทดลองคำสั่ง A(uto) M(ove) และ P(reprocessor)</u>

คำสั่งทั้งสามนี้ต้องใช้ร่วมกัน โดยเริ่มจาก A(uto) จะเป็นการสั่งให้โปรมแกรมรันที่แอดเดรสที่กำหนดพร้อมทั้งเก็บข้อมูลที่เกิดขึ้นบนบัสของข้อมูลไว้ที่แคชแรม ดังนั้นจึงต้องใช้คำสั่ง M(ove) เพื่อทำการย้ายข้อมูลเข้ามายังหน่วยความจำภายในหลังจากนั้นต้องใช้คำสั่ง P(reprocessor)เพื่อกรองข้อมูลที่ได้ให้ถูกต้องทั้งนี้เนื่องจาก สัญญาณนาฬิกาที่ใช้กระตุ้นในส่วนของวงจรดักจับข้อมูลใช้สัญญาณจาก PSEN ซึ่งมีจำนวนสองลูกในหนึ่งรอบการทำงานของเครื่อง (Machine cycle) ดังนั้นจึงจับข้อมูลได้สองไบท์ ซึ่งจะได้ข้อมูลถูกต้องในกรณีที่คำสั่งนั้นมีสองไบท์ในหนึ่งรอบการทำงาน นอกเหนือจากนี้แล้วจะจับข้อมูลได้เกิน จึงต้องใช้คำสั่ง P(reprocessor)เพื่อแก้ข้อมูลให้ถูกต้อง ดังแสดงในรูปที่ 3.12

```
EM51>A 8000

Hello,nice to meet you again.

EM51>M
                Data stored at F000H - FFDFH

EM51>P
                Data was filtered and stored at F000H

EM51>D F000 F03F

      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
F000  22 90 80 09 12 0A D0 E4  93 B4 0D B4 F7 12 0A F1    ".....  ........
F010  12 0A FD C0 83 C0 82 C0  E0 90 40 01 E0 FC E0 FC    ........  ..@.....
F020  E0 FC E0 FC E0 FC E0 FC  E0 FC E0 FC E0 FC E0 FC    ........  ........
F030  E0 FC E0 FC E0 FC E0 FC  E0 FC E0 FC E0 FC E0 FC    ........  ........
EM51>
```

รูปที่ 3.12 แสดงการใช้คำสั่ง A(uto) M(ove) และ P(reprocessor)

มีข้อสังเกตอีกประการหนึ่งคือเมื่อใช้คำสั่ง A(uto) จะเก็บข้อมูล 22 ซึ่งคือคำสั่ง RET ได้ก่อนเสมอ คำสั่งนี้จะบังคับให้อีมูเลเตอร์รันที่ตำแหน่งแอดเดรสที่บรรจุอยู่ใน 2 ไบท์บนสุดของสแตก (Stack) และคำสั่ง RET นี้ จะตามหลังคำสั่งให้แคชแรมเริ่มเก็บข้อมูลแต่ในกรณีที่ใช้คำสั่ง A(uto) ร่วมกับวงจรส่วนจุดชนวน (Trigger Unit) ข้อมูลที่เก็บได้เริ่มต้นจะเป็นข้อมูลที่ถูกกำหนดโดยส่วนจุดชนวน

3.2.7 <u>ทดลองคำสั่ง eX(ternal)</u>

คำสั่งนี้มีไว้เพื่อประยุกต์การใช้งานของส่วนดักจับข้อมูล เมื่อต้องการดักจับข้อมูลจากภายนอกและใช้สัญญาณกระตุ้นจากภายนอก ในกรณีนี้จะเห็นได้ว่าสามารถใช้ส่วนดักจับข้อมูลนี้เป็นเครื่องมืออีกชิ้นหนึ่งนอกเหนือไปจากการทำงานร่วมกับตัวอีมูเลเตอร์เอง คำสั่งนี้จะใช้ร่วมกับคำสั่ง M(ove) เพื่อย้ายข้อมูลที่เก็บได้จากแคชแรมเข้ามายังหน่วยความจำของระบบแล้วนำมาแสดงผลบนจอภาพ ตัวอย่างการใช้งานแสดงดังรูปที่ 3.13

```
EM51>X
                    Press M for move data to memory
EM51>M
                    Data stored at F000H - FFDFH
EM51>
```

รูปที่ 3.13 แสดงการใช้คำสั่ง eX(ternal) ร่วมกับคำสั่ง M(ove)

3.2.8 <u>ทดลองคำสั่ง B(reak)</u>

คำสั่งนี้จะใช้ในการหยุดโปรแกรมบางช่วงเพื่อทำการตรวจสอบค่าในรีจีสเตอร์ต่างๆ จากโปรแกรมตัวอย่างถ้าต้องการให้โปรแกรมหยุดรันชั่วคราวที่ตำแหน่งแอดเดรส 8003H แล้ว จะต้องคีย์ " B 8003 " ก่อน เพื่อเป็นการบอกตำแหน่งหยุดแก่โปรแกรม เมื่อคีย์เรียบร้อยแล้วใช้คำสั่ง " G " สั่งให้รันโปรแกรมที่ตำแหน่งแอดเดรส 8000H โปรแกรมจะถูกหยุดที่ตำแหน่ง 8003H ดังแสดงในรูปที่ 3.14 พร้อมทั้งแสดงค่าในรีจีสเตอร์ออกมาบนจอภาพ ถ้าต้องการรันโปรแกรมต่อจากตำแหน่งที่หยุดไปจนจบให้กด " G " เพียงอย่างเดียวเท่านั้น ส่วนการยกเลิกตำแหน่งการหยุดให้กด " BC "

```
EM51>B 8003

EM51>G 8000

BREAK AT LOCATION 8003
ACC = 0D    PSW = 09    B = 0D    DPTR = 8009    SP = 55
EM51>G

Hello,nice to meet you again.

EM51>
```

รูปที่ 3.14 แสดงการหยุดโปรแกรมบางช่วงเพื่อตรวจสอบค่าในรีจีสเตอร์หรือข้อมูลในหน่วยความจำแบบแรมตามที่ต้องการ

3.2.9 ทดลองคำสั่ง S(ingle)

คำสั่งนี้จะใช้เมื่อต้องการให้อีมูเลเตอร์เริ่มรันโปรแกรมตามตำแหน่งแอดเดรสที่กำหนด โดยรันที่ละคำสั่งพร้อมทั้งแสดงค่าต่างๆ ในรีจิสเตอร์ขึ้นบนจอภาพ ดังแสดงในรูปที่ 3.15

```
EM51>S 8000

BREAK AT LOCATION 8003
ACC = F7    PSW = 09    B = 0D    DPTR = 8009    SP = F4

        Press N for next step Q for quit from step mode

STEP>N
BREAK AT LOCATION 0AD0
ACC = F7    PSW = 09    B = 0D    DPTR = 8009    SP = F4

        Press N for next step Q for quit from step mode

STEP>N
BREAK AT LOCATION 0AD1
ACC = 00    PSW = 08    B = 0D    DPTR = 8009    SP = F4

        Press N for next step Q for quit from step mode

STEP>Q
EM51>
```

รูปที่ 3.15 แสดงการรันโปรแกรมที่ละคำสั่งพร้อมทั้งแสดงค่าในรีจิสเตอร์ต่างๆ

3.2.10 <u>ทดลองคำสั่ง U(nassembler)</u>

คำสั่งนี้จะใช้ในการดิสแอสเซมเบลอร์ (Disassembler)ระหัสภาษาเครื่องในหน่วยความจำให้เป็นภาษาแอสเซมเบลอร์ของ 8051 เพื่ออำนวยความสะดวกในการอ่านโปรแกรมและทำการแก้ไขในหน่วยความจำอีมูเลเตอร์ สมมุติว่าจะทำการดิสแอสเซมเบลอร์โปรแกรมตัวอย่างที่ตำแหน่งแอดเดรส 8000H จำนวน 20 ไบท์ โดยใช้คำสั่ง " U 8000 8020 " ผลที่ได้บนจอภาพจะเป็นดังรูปที่ 3.16

```
EM51>U 8000 8010

8000    MOV DPTR,#8009H
8003    LCALL 0AD0H
8006    LJMP 0026H
8009    INC R5
800A    INC R2
800B    ORL A,R0
800C    XRL A,6CH
800E    XRL A,R4
800F    XRL A,R7
8010    ADD A,R4
EM51>
```

รูปที่ 3.16 แสดงการดีสแอสเซมเบลอร์โปรแกรมในหน่วยความจำ

จากรูปที่ 3.16 จะเห็นว่าคำสั่ง MOV, LCALL, LJMP ถูกต้องตามโปรแกรมตัวอย่าง แต่หลังจากนั้นจะมีคำสั่งเกิดขึ้นมาอีกทั้งนี้เนื่องจากส่วนโปรแกรมดีสแอสเซมเบลอร์ไม่สามารถตัดสินใจได้ว่าไบท์ใดเป็นระหัสภาษาเครื่อง และไบท์ใดเป็นระหัส ASCII ของตัวอักษร ดังนั้น จึงทำการดิสแอสเซมเบลอร์ทั้งหมด ซึ่งโปรแกรมดิสแอสเซมเบลอร์ทั่วๆ ไป จะเป็นเช่นนี้เสมอ

3.2.11 <u>ทดลองคำสั่ง C(opy)</u>

คำสั่งนี้ทำหน้าที่ลอกข้อมูลกลุ่มหนึ่งไปยังตำแหน่งแอดเดรสที่กำหนด จากตัวอย่างโปรแกรมจะทำการลอกข้อมูลจากแอดเดรส 8000H ถึง 802FH ไปยังตำแหน่งแอดเดรส 8030H ผู้ใช้ต้องคีย์คำสั่ง

ดังนี้ "C 8000 802F 8030" หลังจากนั้นทำการดัมดูตำแหน่ง 8030H จะได้ผลดังรูปที่ 3.17 จะเห็นได้ว่าข้อมูลจะเหมือนกับตำแหน่งที่ 8000H ถึง 802FH

```
EM51>C 8000 802F 8030

EM51>D 8000 805F

      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
8000  90 80 09 12 0A D0 02 00  26 0D 0A 48 65 6C 6C 6F     ........ &..Hello
8010  2C 6E 69 63 65 20 74 6F  20 6D 65 65 74 20 79 6F     ,nice to  meet yo
8020  75 20 61 67 61 69 6E 2E  0D 0A 04 FF FF FF FF FF     u again. ........
8030  90 80 09 12 0A D0 02 00  26 0D 0A 48 65 6C 6C 6F     ........ &..Hello
8040  2C 6E 69 63 65 20 74 6F  20 6D 65 65 74 20 79 6F     ,nice to  meet yo
8050  75 20 61 67 61 69 6E 2E  0D 0A 04 FF FF FF FF FF     u again. ........
EM51>
```

รูปที่ 3.17 แสดงการคัดลอกข้อมูลจากที่หนึ่งไปยังอีกที่หนึ่ง

3.2.12 ทดลองคำสั่ง E(dit)

คำสั่งนี้เป็นคำสั่งที่ช่วยให้ผู้ใช้งานทำการแก้ไขข้อมูลในหน่วยความจำได้โดยตรงเพื่อเป็นการไม่เสียเวลาต้องคอมไพล์แล้วดาวน์โหลดโปรแกรมลงไปใหม่ ในกรณีที่โปรแกรมผิดพลาดเล็กน้อย ยกตัวอย่างเช่นโปรแกรมในหัวข้อ 3.2.8 ถ้าต้องการแก้ไขให้โปรแกรมรันและแสดงข้อความออกมาสองบรรทัดในจอภาพ จะทำได้โดยแก้ไขข้อมูลที่ตำแหน่งแอดเดรสที่ 8006H โดยสั่งให้โปรแกรมกระโดดไปที่ตำแหน่ง 8030H จะต้องแก้ไขข้อมูลเป็น 02, 80, 30 วิธีการคือผู้ใช้คีย์ "E 8006" อีมูเลเตอร์จะแสดงตำแหน่งแอดเดรสที่ 8006H และข้อมูลเดิมคือ 02 ออกมาบนจอภาพ ที่ตำแหน่งนี้เราไม่ทำการแก้ไขให้กด Enter อีมูเลเตอร์จะแสดงตำแหน่งแอดเดรสถัดไปคือ 8007H และข้อมูลเดิมเป็น 00 ให้คีย์ข้อมูลใหม่คือ 80 เข้าไปแล้วกด Enter ตำแหน่งที่ 8008H ก็ทำเช่นเดียวกันโดยเปลี่ยนข้อมูลเป็น 30 จากนั้นอีมูเลเตอร์จะแสดงตำแหน่งที่ 8009 ซึ่งเราไม่ต้องการเปลี่ยนข้อมูลและต้องการออกจากคำสั่งแก้ไขนี้ ผู้ใช้ต้องคีย์ "-" (เครื่องหมายลบ) ลงไป อีมูเลเตอร์จะกลับมายัง " EM51> " อีกครั้ง เมื่อทดลองดัมจะเห็นข้อมูลที่ทำการเปลี่ยนแปลงดังรูปที่ 3.18

```
EM51>E 8006

8006 02-

8007 00-80

8008 26-30

8009 0D--
EM51>D 8000 805F

      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
8000  90 80 09 12 0A D0 02 80  30 0D 0A 48 65 6C 6C 6F     ........ 0..Hello
8010  2C 6E 69 63 65 20 74 6F  20 6D 65 65 74 20 79 6F     ,nice to  meet yo
8020  75 20 61 67 61 69 6E 2E  0D 0A 04 FF FF FF FF FF     u again. ........
8030  90 80 09 12 0A D0 02 00  26 0D 0A 48 65 6C 6C 6F     ........ &..Hello
8040  2C 6E 69 63 65 20 74 6F  20 6D 65 65 74 20 79 6F     ,nice to  meet yo
8050  75 20 61 67 61 69 6E 2E  0D 0A 04 FF FF FF FF FF     u again. ........
EM51>G 8000

Hello,nice to meet you again.

Hello,nice to meet you again.

EM51>
```

รูปที่ 3.18 แสดงการแก้ไขข้อมูลในหน่วยความจำผ่านทางแป้นพิมพ์

ในกรณีที่ต้องการถอยหลังไปหนึ่งแอดเดรสเพื่อทำการตรวจสอบหรือแก้ไขซ้ำ จะทำได้โดยใช้คีย์ " Back space " หรือ " . " ก็ได้

3.2.13 <u>ทดลองคำสั่ง I(nsert)</u>

คำสั่งนี้จะทำหน้าที่ใส่ข้อมูลที่เหมือนๆ กัน ลงในหน่วยความจำตามที่กำหนด เช่น ถ้าต้องการเปลี่ยนข้อมูลในหน่วยความจำตั้งแต่แอดเดรส 8000H จนถึง 802FH ให้เป็น 00 ทั้งหมด ผู้ใช้ต้องคีย์คำสั่ง " I 8000 802F 00 " แล้วทดลองดัมดูจะได้ผลออกมาดังรูปที่ 3.19

```
EM51>I 8000 802F 00

EM51>D 8000 805F

      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
8000  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00     ........ ........
8010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00     ........ ........
8020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00     ........ ........
8030  90 80 09 12 0A D0 02 00  26 0D 0A 48 65 6C 6C 6F     ........ &..Hello
8040  2C 6E 69 63 65 20 74 6F  20 6D 65 65 74 20 79 6F     ,nice to  meet yo
8050  75 20 61 67 61 69 6E 2E  0D 0A 04 FF FF FF FF FF     u again. ........
EM51>
```

รูปที่ 3.19 แสดงการใส่ข้อมูลเข้าไปในหน่วยความจำที่มีลักษณะทุกแอดเดรสที่กำหนดมีข้อมูลเหมือนกันหมด

3.2.14 <u>ทดลองคำสั่ง V(erify)</u>

คำสั่งนี้จะทำหน้าที่นำเอาข้อมูลในช่วงแอดเดรสที่กำหนดไว้สองช่วง ขึ้นมาเปรียบเทียบกัน เช่น ถ้าต้องการเปรียบเทียบข้อมูลในแอดเดรสช่วง 8010H ถึง 801FH กับข้อมูลที่เริ่มจากตำแหน่งแอดเดรสที่ 8025H (ซึ่งเราใส่ข้อมูลเป็น " 00 " ไว้) ผู้ใช้ต้องคีย์ "V 8010 801F 8025" ผลที่ได้จะเป็นดังรูปที่ 3.20 ซึ่งข้อมูลของกลุ่มแรก (8000H ถึง 8010H) จะถูกแสดงบนจอภาพทุกแอดเดรส ส่วนกลุ่มที่สองจะถูกนำมาเปรียบกับกลุ่มแรกถ้าข้อมูลเหมือนกันจะแสดงเครื่องหมาย ".." ถ้าข้อมูลต่างกัน จะนำข้อมูลที่ต่างกันขึ้นมาแสดงบนจอภาพ

```
EM51>V 8010 801F 8025

8010  00     8025  ..
8011  00     8026  ..
8012  00     8027  ..
8013  00     8028  ..
8014  00     8029  ..
8015  00     802A  ..
8016  00     802B  ..
8017  00     802C  ..
8018  00     802D  ..
8019  00     802E  ..
801A  00     802F  ..
801B  00     8030  90
801C  00     8031  80
801D  00     8032  09
801E  00     8033  12
801F  00     8034  0A
EM51>


EM51>D 8010 803F

      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F
8010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ........ ........
8020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00    ........ ........
8030  90 80 09 12 0A D0 02 00  26 0D 0A 48 65 6C 6C 6F    ........ &..Hello
EM51>
```

รูปที่ 3.20 แสดงการใช้คำสั่งเปรียบเทียบข้อมูล

3.2.15 <u>ทดลองคำสั่ง #</u>

คำสั่งนี้จะช่วยให้ผู้ใช้ทำการบวกและลบเลขฐานสิบหกสองจำนวนแล้วแสดงผลออกมา ประโยชน์ที่ได้รับคือ ผู้ใช้ไม่ต้องนับตำแหน่งแอดเดรสเมื่อทำการแก้ไขโปรแกรมในหน่วยความจำ เช่น คำสั่งกระโดดแบบอ้างอิง (Relative Jump) ซึ่งคำสั่งเหล่านี้ต้องบอกจำนวนไบท์ที่กระโดดไป มิได้บอกเป็นตำแหน่งแอดเดรสเหมือนคำสั่งกระโดดแบบตรง (Direct Jump) วิธีการใช้คำสั่งนี้ผู้ใช้ต้องกดคีย์ " # XXXX YYYY คือเลขฐานสิบหกสี่หลัก) บนจอภาพจะปรากฏเลขสองจำนวนคือ ค่าผลบวกและผลต่างออกมาดังรูปที่ 3.21

```
EM51># 5050 3030

5050 + 3030 = 8080
5050 - 3030 = 2020
EM51>
```

รูปที่ 3.21 แสดงการใช้คำสั่งบวกและลบเลขฐานสิบหก

3.2.16 ทดลองคำสั่ง F(ind)

คำสั่งนี้จะใช้ในการค้นหาข้อมูลที่ต้องการ เช่น จากรูปที่ 3.19 ต้องการค้นหาข้อมูล '80 09 และ 12 เรียงกันไปตามลำดับภายในหน่วยความจำตั้งแต่แอดเดรสที่ 8000H ถึง 803FH ผลที่ได้จะเป็นดังรูปที่ 3.22

```
EM51>F 8000 803F 80 09 12

Found at address followed;

8001  80
8002  09
8003  12
EM51>F 8000 803F 00 00 00

Search not found

EM51>
```

รูปที่ 3.22 แสดงการค้นหาข้อมูลที่ต้องการในหน่วยความจำที่กำหนด

3.2.17 ทดลองคำสั่ง J(ump)

คำสั่งนี้ใช้แสดงตำแหน่งแอดเดรสของโปรแกรมย่อยที่อยู่ภายในโปรแกรมควบคุมระบบ โดยใช้คำสั่ง " J " ผลที่ได้เป็นดังรูปที่ 3.23

```
EM51>J
0026 COOL       COOL START (RESET STACK POINTER)
0029 WARM       WARM START
002C IN         GET A BYTE FROM THE UART
002F INCH       GET A CHARACTER (ZERO PARITY)
0032 INHEX      GET A HEX CHAR (CARRY=NONHEX)
0035 BYTES      2 BYTE HEX TO HI/LOBYTE
0038 BADDR      BUILD ADDRESS
003B THRADR     THREE ADDRESSES
003E OUT        OUTPUT BYTE IN ACC.
0041 OUTCH      OUTPUT CHARACTER IN ACC.
0044 OUTS       OUTPUT SPACE
0047 OUT2S      OUTPUT 2 SPACES
004A CRLF       OUTPUT [CR] AND [LF]
004D OUT2H      OUTPUT LOC. POINTED BY R0
0050 OUTR0      OUTPUT CONTENTS OF R0
0053 OUTC2HS    OUTPUT PROG. MEM. POINTED BY DPTR AND SPACE
0056 PDATA      OUTPUT MESSAGE POINTED BY DPTR
0059 INC16      INCREMENT 2 BYTE NO. (R0=LOW BYTE)
005C DEC16      DECREMENT 2 BYTE NO. (R0+1=HIGH BYTE)
005F CPY        BLOCK COPY
0062 BRKPT      BREAKPOINT ROUTINE
EM51>
```

รูปที่ 3.23 แสดงตำแหน่งแอดเดรัสของโปรแกรมย่อย

3.3 ผลการทดลองใช้งาน

ผลที่ได้จากการทดลองใช้งานที่ได้ถือว่าเป็นไปตามทฤษฎีที่ออกแบบไว้ทุกประการดังกล่าวผ่านมาแล้วในบทที่ 3 โดยเริ่มจากการเขียนชุดคำสั่งตัวอย่างขึ้นมาแล้วทำการคอมไพล์ให้เป็นไฟล์ที่มีการวางระหัสของคำสั่งเป็นแบบ Hex Intel Format จากนั้นทำการดาวน์โหลดและทดลองคำสั่งต่างๆ ดังผ่านมาแล้วข้างต้น การทดลองในลักษณะนี้เป็นการนำเอา 8051 อีมูเลเตอร์มาใช้งานเพื่อช่วยในการพัฒนาโปรแกรมต่างๆ ตามแต่ผู้ใช้จะเป็นผู้เขียนขึ้นมา ผู้ใช้สามารถทำการทดสอบความถูกต้องของโปรแกรมได้ก่อนที่จะนำไปใช้งานจริง ขั้นตอนการสร้างหรือเตรียมโปรแกรมเพื่อใช้กับ 8051 อีมูเลเตอร์จะเป็นดังนี้

1. ใช้เวิร์ดโปรเซสเซอร์พิมพ์โปรแกรมที่ต้องการ ในการทดลองนี้ใช้โปรแกรมไมโครสตาร์ของบริษัทบอร์แลนด์ (รวมถึงการเขียนโปรแกรมควบคุมของ 8051 อีมูเลเตอร์ด้วย)

2. ทำการคอมไพล์โปรแกรมโดยใช้คอมไพเลอร์ของ MCS-8051 ของบริษัทใดก็ได้ กำหนดให้วางไฟล์ในรูปของ Hex Intel Format ดังตัวอย่างดังรูปที่ 4.1 (ค)

3. ทำการดาวน์โหลดลงไปใน 8051 อีมูเลเตอร์ตามขั้นตอนต่างๆ ในบทที่ 3

เมื่อนำเอา 8051 อีมูเลเตอร์ไปใช้อีมูเลตเข้าไปในแผ่นวงจรที่ใช้งานอื่นๆ เพื่อทำการตรวจสอบการทำงาน พบว่ามีข้อกำหนดเกิดขึ้นคือ

1. แผ่นวงจรนั้น จะต้องไม่มีการเสียหายใดๆ เกี่ยวกับบัสต่างๆ ของระบบ เพราะจะทำให้เกิดการโหลดเข้ามายัง 8051 อีมูเลเตอร์ ทำให้ 8051 อีมูเลเตอร์ไม่ทำงาน (หรือกล่าวง่ายๆ ว่าฮาร์ดแวร์ของแผ่นวงจรจะต้องทำงานตามปกติของมันก่อนที่จะนำมาตรวจสอบ)

2. หน่วยความจำของแผ่นวงจรทำนำมาตรวจสอบจะต้องถูกถอดออก และโปรแกรมในหน่วยความจำเหล่านั้นจะต้องสามารถถ่ายลงไปไว้ในระบบของ 8051 อีมูเลเตอร์ได้ ทั้งนี้เพื่อทำการตรวจสอบและแก้ไขความผิดพลาดที่เกิดจากโปรแกรมอันเป็นประเด็นหลักของการใช้อีมูเลเตอร์

3. ถ้าในแผ่นวงจรใช้ MCS-51 เบอร์ 8051 หรือ 8751 ซึ่งเป็นรอม (อีพรอม) ภายในตัวโปรแกรมที่ถูกบันทึกอยู่ภายในจะต้องถูกอ่านออกมาทำการแก้ไขจนกระทั่งสามารถใช้งานได้บน 8051 อีมูเลเตอร์

```
A>TYPE SAMPLE.ASM
          CPU   "8051.TBL"
          HOF   "INT8"
;
          ORG 8000H       ;start address of user's RAM
;
CR:       EQU       0DH
LF:       EQU       0AH
;
START:    MOV DPTR,#HELLO ;pointer to address of pattern
          LCALL 0AD0H     ;put all characters to terminal
          LJMP 0026H      ;jump to show prompt

HELLO:    DFB  CR,LF,"Hello,nice to meet you again.",CR,LF,04H
;
          END
```

(ก)

```
A>TYPE SAMPLE.LST
 0000                      CPU   "8051.TBL"
 0000                      HOF   "INT8"
                    ;
 8000               ORG 8000H       ;start address of user's RAM
                    ;
 000D =             CR:       EQU       0DH
 000A =             LF:       EQU       0AH
                    ;
 8000 908009        START:    MOV DPTR,#HELLO ;pointer to address of pattern
 8003 120AD0                  LCALL 0AD0H     ;put all characters to terminal
 8006 020026                  LJMP 0026H      ;jump to show prompt

 8009 0D0A48656CHELLO:    DFB  CR,LF,"Hello,nice to meet you again.",CR,LF,04H
                    ;
 0000                         END
```

(ข)

```
A>TYPE SAMPLE.HEX
:10800000908009120AD00200260D0A48656C6C6F38
:108010002C6E69636520746F206D65657420796FBF
:0B8020007520616761696E2E0D0A0477
:00000001FF
```

(ค)

รูปที่ 4.1 แสดงตัวอย่างโปรแกรมที่ใช้ในการทดลอง

(ก) โปรแกรมที่ยังไม่ได้คอมไพล์

(ข) โปรแกรมที่คอมไพล์เรียบร้อยแล้ว

(ค) Hex Intel Format file พร้อมที่จะดาวน์โหลด

ปัญหาที่กล่าวมาข้างต้น เป็นปัญหาพื้นฐานของอีมูเลเตอร์ทั่วๆ ไปทุกบริษัทที่สร้างขึ้น มิได้เกิดแต่เฉพาะ 8051 ที่สร้างขึ้นเท่านั้น ถ้าแผ่นวงจรที่นำมาอีมูเลตไม่เกิดปัญหาดังกล่าวผ่านมาข้างต้นแล้ว การใช้งานของ 8051 อีมูเลเตอร์ที่สร้างขึ้นจะมีประสิทธิภาพทัดเทียมกับอีมูเลเตอร์ราคาแพงของหลายๆ บริษัทที่เดียว

บทที่ 4 สรุปผลและวิจารณ์

4.1 สรุป

จากการทดลองใช้งาน 8051 อีมูเลเตอร์พบว่าผลที่ได้น่าพอใจเมื่อเทียบกับอีมูเลเตอร์ของบริษัทอื่นๆ ที่สร้างขึ้นและมีราคาแพงกว่าราคา 8051 อีมูเลเตอร์หลายๆ เท่าตัว

อีกส่วนหนึ่งที่พัฒนาเพิ่มเข้ากับ 8051 อีมูเลเตอร์ก็คือ ส่วนตรวจจับข้อมูล (Data Capture) ซึ่งสามารถประยุกต์ใช้งานเป็นส่วนวิเคราะห์ข้อมูล (Data Analyzer) ได้ ทำให้สามารถนำเอาสภาวะทางลอจิกของแผ่นวงจรที่ถูกอีมูเลต ขึ้นมาเปรียบเทียบกับข้อมูลที่ได้จากตัวอีมูเลเตอร์เองทำให้สามารถหาจุดเสียบนแผ่นวงจรได้

ในปัจจุบัน บริษัทอินเทลได้ออกไมโครคอมพิวเตอร์แบบชิพเดียวมาอีกตระกูลหนึ่งคือ MCS-52 แต่ชุดคำสั่งเป็นลักษณะของภาษาเบสิก แต่ว่าความเร็วในการทำงานยังสู้ของ MCS-51 ไม่ได้เพราะไมโครโปรเซสเซอร์ต้องเสียเวลาในการแปลภาษาเบสิกให้เป็นภาษาเครื่องก่อนทุกขั้นตอนโดยเฉพาะคำสั่งวนรอบ (เช่น FOR.....NEXT) จะใช้เวลามาก จะทำให้การประมวลผลข้อมูลที่นำเข้ามาจากภายนอกระบบทำได้ช้าลงกว่า MCS-51 มาก

4.2 ข้อแสนอแนะ

ในปัจจุบันนี้อุปกรณ์ที่ใช้ในการตรวจสอบและพัฒนาระบบไมโครคอมพิวเตอร์ ล้วนแต่ใช้เทคนิคของการอีมูเลทั้งสิ้น ดังนั้นผู้ที่สนใจในการพัฒนาอุปกรณ์ประเภทนี้ขึ้นมาใช้งานเองหรือต้องการพัฒนาขึ้นมาเพื่อการค้า ควรจะพัฒนาวงจรให้มีขนาดเล็กลงเพื่อลดต้นทุนลง ในส่วนของวงจรตรวจจับข้อมูลสามารถพัฒนาต่อให้เป็นเครื่องวิเคราะห์สภาวะทางลอจิกได้ โดยเพิ่มจำนวนบิทให้มากขึ้นและใช้สัญญาณนาฬิกาที่มีความถี่สูงเป็นตัวกำหนดอัตราการสุ่ม (Sampling rate) ข้อมูล

ข้อจำกัดของอีมูเลเตอร์ที่พัฒนาขึ้นมาก็คือ สามารถที่จะนำไปใช้กับไมโครคอมพิวเตอร์ในตระกูล MCS-51 เท่านั้น ดังนั้นผู้ที่สนใจจึงควรนำหลักการไปใช้ในการพัฒนาอีมูเลเตอร์ของไมโครคอมพิวเตอร์เบอร์อื่นๆ ต่อไป

เอกสารอ้างอิง

[1] Lancea . Leventhal, *Introduction to Microprocessors. Software, Hardware, Programming.* Newjersey : Prentice-Hall, 1978, P. 57, 255,265,275,278.

[2] Ronald J. Tocci, Lester P. Laskowski, *Microprocessors and Microcomputer Hardware and Software.* Newjersey : Prentice-Hall 1987, PP. 170-172.

[3] Lewis C. Eggebrecht, *Interfacing to The IBM Personal Computer.* Howard W. Sams & Co.,Inc, 1983, PP. 200-202.

[4] Intel, *Embedded Control Application Handbook.* 1989, PP. 2.1-2.30

Intel, *8-Bit Embedded Controller Handbook.* 1989, PP. 5.1-8.59

Siemens, *Microcomputer Components (Microcontroller).* 1988, PP. 39-104.

[5] Intel, *Microcommunications Handbook.* 1989, PP. 2.1-2.25

ภาคผนวก

ภาคผนวกที่ 1

แสดงชุดคำสั่งและรายละเอียดของไมโครคอมพิวเตอร์ชิพเดียวตระกูล MCS-51

ภาคผนวกที่ 2

แสดงโปรแกรมที่รันบน IBM-PC สำหรับใช้ติดต่อสื่อสารกับ 8051 อีมูเลเตอร์ พร้อมส่วนอัพโหลดและดาวน์โหลดโปรแกรมใช้คอมไพเลอร์ของบริษัทไมโครซอฟท์ มาเป็นตัวคอมไพล์ทั้งส่วนของภาษาซีและภาษา แอสเซมบลี้ ขั้นตอนได้บันทึกไว้ในโปรแกรมเรียบร้อยแล้ว

ภาคผนวกที่ 3

แสดงโปรแกรมควบคุมระบบทั้งหมดของ 8051 อีมูเลเตอร์ การคอมไพล์โปรแกรมใช้คอมไพเลอร์ของบริษัทใดก็ได้ ในกรณีที่เป็นคอมไพเลอร์ที่สร้างขึ้นมาเฉพาะเบอร์ 8051 เช่น ASM 51 ของบริษัทอินเทล ให้ตัดสองบรรทัดแรกของโปรแกรมออกคือ

```
CPU    "8051.TBL"
HOF    "INT8"
```

ส่วนนี้เป็นส่วนที่กำหนดขึ้นมาเป็นพิเศษเพื่อบอกคอมไพเลอร์ที่ผู้ทำวิทยานิพนธ์ใช้งานอยู่ทราบว่าผู้ใช้ต้องการคอมไพล์ให้กับ CPU เบอร์อะไร (ใช้โปรแกรม C16)

ภาคผนวกที่ 4

แสดงไฟล์ซาร์ทของโปรแกรมที่รันบน IBM-PC มีชื่อว่า Terminal.exe ซึ่งใช้ติดต่อระหว่าง IBM-PC กับ 8051- Emulator

ภาคผนวกที่ 1

ชุดคำสั่งของไมโครคอมพิวเตอร์แบบชิพเดียว MCS-51

8051 Family Instruction Set Summary

MNEMONIC	ASSEMBLY LANGUAGE FORM	HEX OP-CODE	BYTES	CYCLES	PSW FLAGS AFFECTED
ACALL					
2K in Page (11 bits) Call	ACALL Paddr	see note 1	2	2	
ADD					
Add Operand to Acc	ADD A,#data	24	2	1	AC, Cy, OV
	ADD A,Daddr	25	2	1	
	ADD A,@Ri	26,27	1	1	
	ADD A,Rn	28-2F	1	1	
ADDC					
Add Operand with Cy to Acc	ADDC A,#data	34	2	1	AC, Cy, OV
	ADDC A,Daddr	35	2	1	
	ADDC A,@Ri	36,37	1	1	
	ADDC A,Rn	38-3F	1	1	
AJMP					
2K in Page (11 bits) JMP	AJMP Paddr	see note 2	2	2	
ANL					
Logical AND of Source Operand with Destination Operand	ANL Daddr,A	52	2	1	
	ANL Daddr,#data	53	3	2	
	ANL A,#data	54	2	1	
	ANL A,Daddr	55	2	1	
	ANL A,@Ri	56,57	1	1	
	ANL A,Rn	58-5F	1	1	
Logical AND of Source Operand with Cy	ANL C,Baddr	82	2	2	Cy
Logical AND of Source Operand Complemented with Cy	ANL C,/Baddr	B0	2	2	Cy
CJNE					
Compare Operands and Jump Relative if not Equal	CJNE A,#data,Roff	B4	3	2	Cy see note 3
	CJNE A,Daddr,Roff	B5	3	2	
	CJNE @Ri,#data,Roff	B6,B7	3	2	
	CJNE Rn,#data,Roff	B8-BF	3	2	
CLR					
Clear Acc	CLR A	E4	1	1	
Clear Carry Flag	CLR C	C3	1	1	Cy
Clear Bit Operand	CLR Baddr	C2	2	1	
CPL					
Complement Acc	CPL A	F4	1	1	
Complement Carry Flag	CPL C	B3	1	1	Cy
Complement Bit Operand	CPL Baddr	B2	2	1	
DA					
Decimal Adjust Acc for Addition	DA A	D4	1	1	Cy see note 4

MNEMONIC	ASSEMBLY LANGUAGE FORM	HEX OP-CODE	BYTES	CYCLES	PSW FLAGS AFFECTED
DEC Decrement Operand	DEC A	14	1	1	
	DEC Daddr	15	2	1	
	DEC @Ri	16,17	1	1	
	DEC Rn	18-1F	1	1	
DIV Divide Acc by B Reg	DIV AB	84	1	4	Cy, OV see note 5
DJNZ Decrement Operand and Jump Relative if Not 0	DJNZ Daddr,Roff	D5	3	2	
	DJNZ Rn,Roff	D8-DF	2	2	
INC Increment Operand	INC A	04	1	1	
	INC Daddr	05	2	1	
	INC @Ri	06,07	1	1	
	INC Rn	08-0F	1	1	
	INC DPTR	A3	1	2	
JB Jump Relative if Bit is Set	JB Baddr,Roff	20	3	2	
JBC Jump Relative if Bit is Set and Clear Bit	JBC Baddr,Roff	10	3	2	Cy, OV AC see note 6
JC Jump Relative if Cy is CLR	JC Roff	40	2	2	
JMP Jump Indirect	JMP @A+DPTR	73	1	2	
JNB Jump Relative if Bit is CLR	JNB Baddr,Roff	30	3	2	
JNC Jump Relative if Cy is CLR	JNC Roff	50	2	2	
JNZ Jump Relative if Acc = 0	JNZ Roff	70	2	2	
JZ Jump Relative if Acc = 0	JZ Roff	60	2	2	
LCALL Long (16 bits) Call	LCALL Paddr	12	3	2	
LJMP Long (16 bits) Jump	LJMP Paddr	02	3	2	

MNEMONIC	ASSEMBLY LANGUAGE FORM	HEX OP-CODE	BYTES	CYCLES	PSW FLAGS AFFECTED
MOV					
Move Source Operand	MOV A,#data	74	2	1	
to Destination	MOV A,Daddr	E5	2	1	
Operand	MOV A,@Ri	E6,E7	1	1	
	MOV A,Rn	E8-EF	1	1	
	MOV Daddr,A	F5	2	1	
	MOV Daddr,#data	75	3	2	
	MOV Daddr,Daddr	85	3	2	
	MOV Daddr,@Ri	86,87	2	2	
	MOV Daddr,Rn	88-8F	2	2	
	MOV @Ri,A	F6,F7	1	1	
	MOV @Ri,#data	76,77	2	1	
	MOV @Ri,Daddr	A6,A7	2	2	
	MOV Rn,A	F8-FF	1	1	
	MOV Rn,#data	78-7F	2	1	
	MOV Rn,Daddr	A8-AF	2	2	
	MOV DPTR,#data16	90	3	2	
Move Cy to Bit	MOV Baddr,C	92	2	2	
Move Bit to Cy	MOV C,Baddr	A2	2	1	Cy
MOVC					
Move byte from Program	MOVC A,@A+DPTR	93	1	2	
Memory to Acc	MOVC A,@A+PC	83	1	2	
MOVX					
Move byte from Ext	MOVX A,@Ri	E2,E3	1	2	
Data Memory	MOVX A,@DPTR	E0	1	2	
to Acc					
Move byte in Acc	MOVX @Ri,A	F2,F3	1	2	
to Ext Data Memory	MOVX @DPTR,A	F0	1	2	
MUL					
Multiply Acc	MUL AB	A4	1	4	Cy, OV
by B Reg					see note 7
NOP					
No Operation	NOP	00	1	1	
ORL					
Logical Inclusive OR	ORL Daddr,A	42	2	1	
of Source Operand	ORL Daddr,#data	43	3	2	
with Destination	ORL A,#data	44	2	1	
Operand	ORL A,Daddr	45	2	1	
	ORL A,@Ri	46,47	1	1	
	ORL A,Rn	48-4F	1	1	
Logical Inclusive OR of Source Operand with Cy	ORL C,Baddr	72	2	2	Cy
Logical Inclusive OR of Source Operand Complemented with Cy	ORL C,/Baddr	A0	2	2	Cy
POP					
Pop Stack and Place in Destination Operand	POP Daddr	D0	2	2	

MNEMONIC	ASSEMBLY LANGUAGE FORM	HEX OP-CODE	BYTES	CYCLES	PSW FLAGS AFFECTED
PUSH Push Operand onto Stack	PUSH Daddr	C0	2	2	
RET Ret from Subroutine	RET	22	1	2	
RETI Ret from Interrupt Routine	RETI	32	1	2	
RL Rotate Acc Left 1 Bit	RL A	23	1	1	
RLC Rotate Acc Left 1 Bit Thru Cy	RLC A	33	1	1	Cy
RR Rotate Acc Right 1 Bit	RR A	03	1	1	
RRC Rotate Acc Right 1 Bit Thru Cy	RRC A	13	1	1	Cy
SETB Set Carry Flag	SETB C	D3	1	1	Cy
Set Bit Operand	SETB Baddr	D2	2	1	
SJMP Jump Relative	SJMP Roff	80	2	2	
SUBB Subtract Operand with Borrow	SUBB A,#data	94	2	1	Ac, Cy, OV
	SUBB A,Daddr	95	2	1	
	SUBB A,@Ri	96,97	1	1	
	SUBB A,Rn	98-9F	1	1	
SWAP Swap Nibbles within Acc	SWAP A	C4	1	1	
XCH Exchange bytes of Acc and Source Operand	XCH A,Daddr	C5	2	1	
	XCH A,@Ri	C6,C7	1	1	
	XCH A,Rn	C8-CF	1	1	
XCHD Exchange the Least Sig Nibble of Acc and Source Operand	XCHD A,@Ri	D6-D7	1	1	
XRL Logical Exclusive OR of Source Operand with Destination Operand	XRL Daddr,A	62	2	1	
	XRL Daddr,#data	63	3	2	
	XRL A,#data	64	2	1	
	XRL A,Daddr	65	2	1	
	XRL A,@Ri	66,67	1	1	
	XRL A,Rn	68-6F	1	1	

NOTES:

1. Starting with 11H as the opcode base, the final opcode is formed by placing bits 8, 9 and 10 of the target address in bits 5, 6 and 7 of the opcode. The 8 possible opcodes in hexadecimal are then: 11, 31, 51, 71, 91, B1, D1, F1.

2. Starting with 01H as the opcode base, the final opcode is formed by placing bits 8, 9 and 10 of the target address in bits 5, 6 and 7 of the opcode. The 8 possible opcodes in hexadecimal are then: 01, 21, 41, 61, 81, A1, C1, E1.

3. The Carry Flag is set if the Destination Operand is less than the Source Operand. Otherwise the Carry Flag is cleared.

4. The Carry Flag is set if the BCD result in the Accumulator is greater than decimal 99.

5. The Overflow Flag is set if the B Register contains zero (flags a divide by zero operation). Otherwise the Overflow Flag is cleared.

6. If any of the condition code flags are specified as the operand of this instruction, they will be reset by the instruction if they were originally set.

7. The high byte of the 16-bit product is placed in the B Register, the low byte in Accumulator.

ABBREVIATIONS:

A, Acc — Accumulator
AB — Register Pair
Baddr — Bit Address
/Baddr — Complemented Contents of Bit Address
C, Cy — Carry Bit
Daddr — Data Address
#data — Data Coded in Instruction
DPTR — Data Pointer
Paddr — Program Memory Address
PC — Program Counter
Rel — Relative
Rn — General Purpose Registers 0-7
@Ri — Indirect Address Register 0-1
Roff — Relative Offset Address

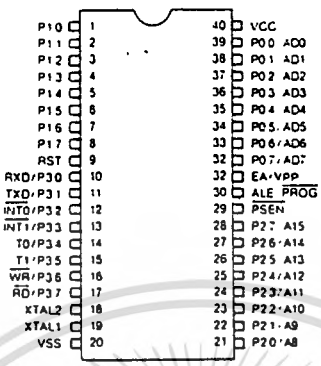
8051 Family Hex Opcode to Mnemonics Cross Reference

Hex Code	*Number of Bytes*	*Number of Cycles*	*Mnemonic*	*Operands*
00	1	1	NOP	
01	2	2	AJMP	*Paddr*
02	3	2	LJMP	*Paddr*
03	1	1	RR	A
04	1	1	INC	A
05	2	1	INC	*Daddr*
06	1	1	INC	@R0
07	1	1	INC	@R1
08	1	1	INC	R0
09	1	1	INC	R1
0A	1	1	INC	R2
0B	1	1	INC	R3
0C	1	1	INC	R4
0D	1	1	INC	R5
0E	1	1	INC	R6
0F	1	1	INC	R7
10	3	2	JBC	*Baddr,Roff*
11	2	2	ACALL	*Paddr*
12	3	2	LCALL	*Paddr*
13	1	1	RRC	A
14	1	1	DEC	A
15	2	1	DEC	*Daddr*
16	1	1	DEC	@R0
17	1	1	DEC	@R1
18	1	1	DEC	R0
19	1	2	DEC	R1
1A	1	1	DEC	R2
1B	1	1	DEC	R3
1C	1	1	DEC	R4
1D	1	1	DEC	R5
1E	1	1	DEC	R6
1F	1	1	DEC	R7
20	3	2	JB	*Baddr,Paddr*
21	2	2	AJMP	*Paddr*
22	1	2	RET	
23	1	1	RL	A
24	2	1	ADD	A,*#data*
25	2	1	ADD	A,*Daddr*
26	1	1	ADD	A,@R0
27	1	1	ADD	A,@R1
28	1	1	ADD	A,R0
29	1	1	ADD	A,R1
2A	1	1	ADD	A,R2
2B	1	1	ADD	A,R3
2C	1	1	ADD	A,R4
2D	1	1	ADD	A,R5
2E	1	1	ADD	A,R6
2F	1	1	ADD	A,R7

Hex Code	Number of Bytes	Number of Cycles	Mnemonic	Operands
30	3	2	JNB	*Baddr,Paddr*
31	2	2	ACALL	*Paddr*
32	1	2	RETI	
33	1	1	RLC	A
34	2	1	ADDC	A,*#data*
35	2	1	ADDC	A,*Daddr*
36	1	1	ADDC	A,@R0
37	1	1	ADDC	A,@R1
38	1	1	ADDC	A,R0
39	1	1	ADDC	A,R1
3A	1	1	ADDC	A,R2
3B	1	1	ADDC	A,R3
3C	1	1	ACCC	A,R4
3D	1	1	ADDC	A,R5
3E	1	1	ADDC	A,R6
3F	1	1	ADDC	A,R7
40	2	2	JC	*Paddr*
41	2	2	AJMP	*Paddr*
42	2	1	ORL	*Daddr*,A
43	3	2	ORL	*Daddr , data*
44	2	1	ORL	A,*#data*
45	2	1	ORL	A,*Daddr*
46	1	1	ORL	A,@R0
47	1	1	ORL	A,@R1
48	1	1	ORL	A,R0
49	1	1	ORL	A,R1
4A	1	1	ORL	A,R2
4B	1	1	ORL	A,R3
4C	1	1	ORL	A,R4
4D	1	1	ORL	A,R5
4E	1	1	ORL	A,R6
4F	1	1	ORL	A,R7
50	2	2	JNC	*Paddr*
51	2	2	ACALL	*Paddr*
52	2	1	ANL	*Daddr*,A
53	3	2	ANL	*Daddr , data*
54	2	1	ANL	A,*#data*
55	2	1	ANL	A,*Daddr*
56	1	1	ANL	A,@R0
57	1	1	ANL	A,@R1
58	1	1	ANL	A,R0
59	1	1	ANL	A,R1
5A	1	1	ANL	A,R2
5B	1	1	ANL	A,R3
5C	1	1	ANL	A,R4
5D	1	1	ANL	A,R5
5E	1	1	ANL	A,R6
5F	1	1	ANL	A,R7
60	2	2	JZ	*Roff*
61	2	2	AJMP	*Paddr*
62	2	1	XRL	*Daddr*,A
63	3	2	XRL	*Daddr , data*
64	2	1	XRL	A,*#data*
65	2	1	XRL	A,*Daddr*
66	1	1	XRL	A,@R0
67	1	1	XRL	A,@R1
68	1	1	XRL	A,R0
69	1	1	XRL	A,R1
6A	1	1	XRL	A,R2
6B	1	1	XRL	A,R3
6C	1	1	XRL	A,R4
6D	1	1	XRL	A,R5
6E	1	1	XRL	A,R6
6F	1	1	XRL	A,R7
70	2	2	JNZ	*Roff*
71	2	2	ACALL	*Paddr*
72	2	2	ORL	C,*Baddr*
73	1	2	JMP	@A+DPTR
74	2	1	MOV	A,*#data*
75	3	2	MOV	*Daddr,#data*
76	2	1	MOV	@R0,*#data*
77	2	1	MOV	@R1,*#data*
78	2	1	MOV	R0,*#data*
79	2	1	MOV	R1,*#data*
7A	2	1	MOV	R2,*#data*
7B	2	1	MOV	R3,*#data*
7C	2	1	MOV	R4,*#data*
7D	2	1	MOV	R5,*#data*
7E	2	1	MOV	R6,*#data*
7F	2	1	MOV	R7,*#data*
80	2	2	SJMP	*Roff*
81	2	2	AJMP	*Paddr*
82	2	2	ANL	C,*Baddr*
83	1	2	MOVC	A,@A+PC
84	1	4	DIV	AB
85	3	2	MOV	*Daddr,Daddr*
86	2	2	MOV	*Daddr*,@R0
87	2	2	MOV	*Daddr*,@R1
88	2	2	MOV	*Daddr*,R0
89	2	2	MOV	*Daddr*,R1
8A	2	2	MOV	*Daddr*,R2
8B	2	2	MOV	*Daddr*,R3
8C	2	2	MOV	*Daddr*,R4
8D	2	2	MOV	*Daddr*,R5
8E	2	2	MOV	*Daddr*,R6
8F	2	2	MOV	*Daddr*,R7
90	3	2	MOV	DPTR,*#data*
91	2	2	ACALL	*Paddr*
92	2	2	MOV	*Baddr*,C
93	1	2	MOVC	A,@A+DPTR
94	2	1	SUBB	A,*#data*
95	2	1	SUBB	A,*Daddr*
96	1	1	SUBB	A,@R0
97	1	1	SUBB	A,@R1
98	1	1	SUBB	A,R0
99	1	1	SUBB	A,R1
9A	1	1	SUBB	A,R2
9B	1	1	SUBB	A,R3
9C	1	1	SUBB	A,R4
9D	1	1	SUBB	A,R5
9E	1	1	SUBB	A,R6
9F	1	1	SUBB	A,R7
A0	2	2	ORL	C,/*Baddr*
A1	2	2	AJMP	*Paddr*
A2	2	1	MOV	C,*Baddr*
A3	1	2	INC	*DPTR*

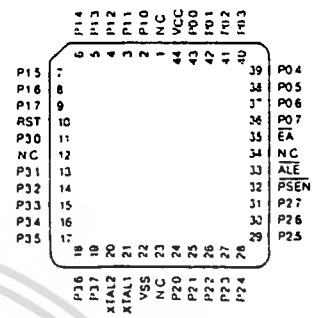
Hex Code	Number of Bytes	Number of Cycles	Mnemonic	Operands
A4	1	4	MUL	AB
A5			reserved	
A6	2	2	MOV	@R0,*Daddr*
A7	2	2	MOV	@R1,*Daddr*
A8	2	2	MOV	R0,*Daddr*
A9	2	2	MOV	R1,*Daddr*
AA	2	2	MOV	R2,*Daddr*
AB	2	2	MOV	R3,*Daddr*
AC	2	2	MOV	R4,*Daddr*
AD	2	2	MOV	R5,*Daddr*
AE	2	2	MOV	R6,*Daddr*
AF	2	2	MOV	R7,*Daddr*
B0	2	2	ANL	C,/*Baddr*
B1	2	2	ACALL	*Paddr*
B2	2	1	CPL	*Baddr*
B3	1	1	CPL	C
B4	3	2	CJNE	A,*#data,Paddr*
B5	3	2	CJNE	A,*Daddr,Paddr*
B6	3	2	CJNE	@R0,*#data,Paddr*
B7	3	2	CJNE	@R1,*#data,Paddr*
B8	3	2	CJNE	R0,*#data,Paddr*
B9	3	2	CJNE	R1,*#data,Paddr*
BA	3	2	CJNE	R2,*#data,Paddr*
BB	3	2	CJNE	R3,*#data,Paddr*
BC	3	2	CJNE	R4,*#data,Paddr*
BD	3	2	CJNE	R5,*#data,Paddr*
BE	3	2	CJNE	R6,*#data,Paddr*
BF	3	2	CJNE	R7,*#data,Paddr*
C0	2	2	PUSH	*Daddr*
C1	2	2	AJMP	*Paddr*
C2	2	1	CLR	*Baddr*
C3	1	1	CLR	C
C4	1	1	SWAP	A
C5	2	1	XCH	A,*Daddr*
C6	1	1	XCH	A,@R0
C7	1	1	XCH	A,@R1
C8	1	1	XCH	A,R0
C9	1	1	XCH	A,R1
CA	1	1	XCH	A,R2
CB	1	1	XCH	A,R3
CC	1	1	XCH	A,R4
CD	1	1	XCH	A,R5
CE	1	1	XCH	A,R6
CF	1	1	XCH	A,R7
D0	2	2	POP	*Daddr*
D1	2	2	ACALL	*Paddr*
D2	2	1	SETB	*Baddr*
D3	1	1	SETB	C
D4	1	1	DA	A
D5	3	2	DJNZ	*Daddr,Paddr*
D6	1	1	XCHD	A,@R0
D7	1	1	XCHD	A,@R1
D8	2	2	DJNZ	R0,*Roff*
D9	2	2	DJNZ	R1,*Roff*
DA	2	2	DJNZ	R2,*Roff*
DB	2	2	DJNZ	R3,*Roff*
DC	2	2	DJNZ	R4,*Roff*
DD	2	2	DJNZ	R5,*Roff*
DE	2	2	DJNZ	R6,*Roff*
DF	2	2	DJNZ	R7,*Roff*
E0	1	2	MOVX	A,@DPTR
E1	2	2	AJMP	*Paddr*
E2	1	2	MOVX	A,@R0
E3	1	2	MOVX	A,@R1
E4	1	1	CLR	A
E5	2	1	MOV	A,*Daddr*
E6	1	1	MOV	A,@R0
E7	1	1	MOV	A,@R1
E8	1	1	MOV	A,R0
E9	1	1	MOV	A,R1
EA	1	1	MOV	A,R2
EB	1	1	MOV	A,R3
EC	1	1	MOV	A,R4
ED	1	1	MOV	A,R5
EE	1	1	MOV	A,R6
EF	1	1	MOV	A,R7
F0	1	1	MOVX	@DPTR,A
F1	2	2	ACALL	*Paddr*
F2	1	2	MOVX	@R0,A
F3	1	2	MOVX	@R1,A
F4	1	1	CPL	A
F5	2	1	MOV	*Daddr*,A
F6	1	1	MOV	@R0,A
F7	1	1	MOV	@R1,A
F8	1	1	MOV	R0,A
F9	1	1	MOV	R1,A
FA	1	1	MOV	R2,A
FB	1	1	MOV	R3,A
FC	1	1	MOV	R4,A
FD	1	1	MOV	R5,A
FE	1	1	MOV	R6,A
FF	1	1	MOV	R7,A

8051 Pin Configuration



40L DIP

N.C. = Not Connected
Multiplexed functions omitted for clarity



44L PLCC

8051 Special Function Registers

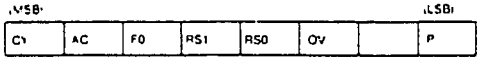
Symbol	*Name*	*Address*
*P0	Port 0	80H
SP	Stack Pointer	81H
DPTR	Data Pointer	
DPL	Low Byte	82H
DPH	High Byte	83H
PCON	Power Control	87H
*TMOD	Timer/Counter Mode Control	88H
*TCON	Timer/Counter Control	89H
TL0	Timer/Counter 0 Low Byte	8AH
TL1	Timer/Counter 1 Low Byte	8BH
TH0	Timer/Counter 0 High Byte	8CH
TH1	Timer/Counter 1 High Byte	8DH
*P1	Port 1	90H
*SCON	Serial Port Control	98H
SBUF	Serial Data Buffer	99H
*IE	Interrupt Enable	A8H
*P2	Port 2	A0H
*IP	Interrupt Priority	B8H
*P3	Port 3	B0H
*PSW	Program Status Word	D0H
*ACC	Accumulator	E0H
*B	B Register	F0H

**Bit Addressable*

ROM Device: 8051 (4K ROM)
ROMless Device: 8031
EPROM Device: 8751 (4K EPROM)
Internal RAM: 128 Bytes
Primary Vendor: Intel

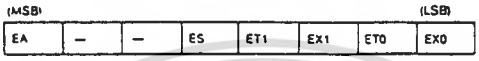
Timer/Counter: 2
I/O: Four 8-bit Ports
Interrupts: 5 Sources
Max. Speed: 16 MHZ

PSW: Program Status Word (D0H) [Bit Addressable]



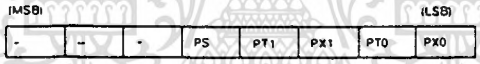
Sym	*Position*	*Function*	*Sym*	*Position*	*Function*
CY	PSW.7	Carry flag	RS0	PSW.3	Register bank select 0
AC	PSW.6	Auxiliary Carry flag	OV	PSW.2	Overflow flag
F0	PSW.5	Flag 0	—	PSW.1	Reserved
RS1	PSW.4	Register bank select 1	P	PSW.0	Acc Parity flag

IE: Interrupt Enable Register (A8H) [Bit Addressable]



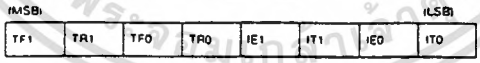
Sym	*Position*	*Function*	*Vector Address*
EA	IE.7	Disables all interrupts	
—	IE.6	Reserved	
—	IE.5	Reserved	
ES	IE.4	Serial Port	23H
ET1	IE.3	Timer 1 overflow interrupt	1BH
EX1	IE.2	External interrupt 1	13H
ET0	IE.1	Timer 0 overflow interrupt	0BH
EX0	IE.0	External interrupt 0	03H

IP: Interrupt Priority Register (B8H) [Bit Addressable]



Sym	*Position*	*Function*	*Sym*	*Position*	*Function*
—	IP.7	Reserved	PT1	IP.3	Timer 1
—	IP.6	Reserved	PX1	IP.2	External interrupt 1
—	IP.5	Reserved	PT0	IP.1	Timer 0
PS	IP.4	Serial Port	PX0	IP.0	External interrupt 0

TCON: Timer/Counter Control Register (88H) [Bit Addressable]



Sym	*Position*	*Function*	*Sym*	*Position*	*Function*
TF1	TCON.7	Timer 1 overflow flag	IE1	TCON.3	Interrupt 1 edge flag
TR1	TCON.6	Timer 1 run control bit	IT1	TCON.2	Interrupt 1 type control
TF0	TCON.5	Timer 0 overflow flag	IE0	TCON.1	Interrupt 0 edge flag
TR0	TCON.4	Timer 0 run control bit	IT0	TCON.0	Interrupt 0 type control

TMOD: Timer/Counter Mode Control Register (89H)

(MSB)							(LSB)
GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0

Sym	*Function*
GATE	Gating control when set
C/$\overline{T}$	Timer/Counter Select 0=Timer 1=Counter

M1	*M0*	*Operating Mode*
0	0	MCS-48 Timer
0	1	16-bit Timer/Counter
1	0	8-bit auto-reload Timer/Counter
1	1	(Timer 0) TL0=8-bit Timer Counter
		(Timer 0) TH0=8-bit Timer Counter
1	1	(Timer 1) T/C1 stopped

SCON: Serial Control Register (98H) [Bit Addressable]

(MSB)							(LSB)
SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Sym		*Function*		
SM1	*SM0*	*Mode*	*Operating*	*Baud Rate*
0	0	0	Shift Register	$f_{osc.}/12$
0	1	1	8-bit UART	variable
1	0	2	9-bit UART	$f_{osc.}/64$ or $f_{osc.}/32$
1	1	3	9-bit UART	variable

Sym	*Position*	*Function*
SM2	TMOD.5	Enables multiprocessor communication in Modes 2 and 3
REN	TMOD.4	Enables serial reception
TB8	TMOD.3	9th data bit transmitted in Modes 2 and 3
RB8	TMOD.2	9th data bit received in Modes 2 and 3
TI	TMOD.1	Transmit Interrupt Flag
RI	TMOD.0	Receive Interrupt Flag

Commonly Used Baud Rates:

			Timer 1		
Baud Rate	f_{osc}	*SMOD*	*C/$\overline{T}$*	*Mode*	*Reload Value*
Mode 0 Max: 1 MHZ	12 MHZ	X	X	X	X
Mode 2 Max: 375K	12 MHZ	1	X	X	X
Modes 1, 3: 62.5K	12 MHZ	1	0	2	FFH
19.2K	11.059 MHZ	1	0	2	FDH
9.6K	11.059 MHZ	0	0	2	FDH
4.8K	11.059 MHZ	0	0	2	FAH
2.4K	11.059 MHZ	0	0	2	F4H
1.2K	11.059 MHZ	0	0	2	E8H
137.5	11.986 MHZ	0	0	2	1DH
110	6 MHZ	0	0	2	72H
110	12 MHZ	0	0	1	FEEBH

ภาคผนวกที่ 2

โปรแกรมที่ใช้งานบนเครื่องคอมพิวเตอร์ส่วนบุคคล IBM PC/XT

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

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

        This program written by Charin Buntukul.
        Source codes for Communication Link between EM51 and IBM PC.

                *  Version 1.5 -- June 1, 1986

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

                              PRODUCTS USED
                              -------------
                            IBM PC     DOS  V3.00
                            Microsoft C     V3.00
                            Microsoft MASM V4.00
                            Microsoft LINK V3.05

     1) MS TERMONAL;                                  --TERMINAL.C
                                                      --DOS.H
                                                      --STDIO.H
                                                      --TIME.H
                                                      ++TERMINAL.OBJ
     2) MS SCRNIO;                                    --SCRNIO.C
                                                      --DOS.H
                                                      ++SCRNIO.OBJ
     3) MASM KBDIO;                                   --KBDIO.ASM
                                                      ++KBDIO.OBJ
     4) MASM ASYNCIO;                                 --ASYNCIO.ASM
                                                      ++ASYNCIO.OBJ
     5) LINK TERMINAL+SCRNIO+KBDIO+ASYNCIO;           --TERMINAL.OBJ
                                                      --SCRNIO.OBJ
                                                      --KBDIO.OBJ
                                                      --ASYNCIO.OBJ
                                                      --SLIBC.LIB
                                                      --SLIBFP.LIB
                                                      --EM.LIB
                                                      ++TERMINAL.EXE

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

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

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

main(argc,argv)
  int argc;
  char *argv[];
  {
    int x;
    extern int cport;

    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 EM51 logo and returns COM port --
*/

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

   if (argc < 3)
          {
            cls();
            if (cport == 0)
                  {
                    locate(1,13);
                    bdos(9,"*********************************************$");
                    locate(2,13);
                    bdos(9,"*                                           *$");
                    locate(3,13);
                    bdos(9,"*             Charin  Bantukul              *$");
                    locate(4,13);
                    bdos(9,"*   Communication Link between 8051 & IBM   *$");
                    locate(5,13);
                    bdos(9,"*          @ Copyright 1985, 1986           *$");
                    locate(6,13);
                    bdos(9,"*        Version 1.5, June 1, 1986          *$");
                    locate(7,13);
                    bdos(9,"*                                           *$");
                    locate(8,13);
                    bdos(9,"*********************************************$");
                    locate(11,17);
                    printf("** PRESS F3 to display function keys **");
```

```
                    locate(13,1);
                    bdos(9, "Which port do you wish to use? [1]$");
                    buf[0] = 48;
                    while (*buf != 49 && *buf != 50)
                       {
                         locate(13,36);
                         x = getline(buf,3);
                         if (*buf == '\0')
                            { *buf = '1'; break; }
                       }
                    port = *buf - 48;
                  }
               else port = cport;
            locate(16,1);
            bdos(9,"Which baud rate do you wish to use? [ 9600]$");
            locate(17,8);
            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)
               {
                 locate(16,45);
                 printf("  \b\b");
                 getline(buf,2);
                 if (*buf != 0) c = *buf;
                        else { c = '5'; break; }
               }
         }
      else { port = *argv[1] - 48; c = *argv[2]; }
   x = c;
   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 --
*/

term(port,argc)
int port,argc;
```

```
{
  char c, flag, tflag;
  int x,y,z,base,pchar;
  extern char scrn;

  if (argc < 3) cls();
  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; z = 10;
  outp(base+1,0);                       /* Reset interrupts on 8250      */
  outp(base+1,1);                       /* Set interrupts on 8250        */
  while (inpcnt(port)) c = commin(port);
  while (inpcnt(port) == 0)
     {
       if (y == 25)
          {
            delay(1);
            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 programmer.\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 '$':  bdos(2,c); commout(port,c); tflag = 1;
                                     if (inpcnt(port)) commclean(port,c);
                                     break;
                          case 0:    flag = 1; break;
                          case XOFF: while ((pchar = charin()) == -1) ;
                                     c = pchar;
                                     if (c == '$')
                                        {
                                          bdos(2,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);
                        }
             }
        }
      if (inpcnt(port))
        {
          c = commin(port);
          if (c == '$') { if (!tflag) bdos(2,c); }
                  else { bdos(2,c); tflag = 0; }
          if (c == BS) { bdos(2,' '); bdos(2,c); }
        }
    }
 }

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

commclean(port,c)
 int port; char c;
 { delay(1); commflush(port); commout(port,c); }

/*
endterm(port,base)             -- Ends terminal emulation --
*/
```

```
endterm(port,base)
 /* Disable DTR, RTS, OUT2 */
 { commflush(port); outp(base+4,0); uninit(port); }

/*
delay(seconds)                  -- Time delay procedure --
*/

delay(seconds)
int seconds;
 {
   long start, time();
   int temp;

   temp = seconds + 1; start = time(NULL);
   for (;;) if ((time(NULL)-start) >= temp) break;
 }

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

setbaud(port,baudrate)
int baudrate, 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);
      }
 }

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

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

char *s;
```

```
int lim;
 {
   int i;
   char c;

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

/*
dumpbin(port)          -- Dumps a file to EM51 --
*/

dumpbin(port)
int port;
 {
   extern FILE *fopen();
   char c, filename[64], buf;
   FILE *fp;
   int ch, pchar, x, z;
   extern char scrn;

   printf("\ndriver: Name of download file? ");
   getline(filename,64);
   if (*filename == 0)
     { printf("driver: no action taken"); commout(port,'$'); return(-1); }
   fp = fopen(filename,"rb");
   if (fp == 0)
      { printf("driver: file not found"); commout(port,'$'); return(-1); }
   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) bdos(2,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"); bdos(2,c); }
              else
                {
                  commout(port,'$');
                  printf("\ndriver: download terminated by operator");
                  printf("\ndriver: reset the EM51 on a binary download.");
                }
       else { printf("\ndriver: download complete"); bdos(2,c); }
    fclose(fp);
  }

/*
readasci(port)          -- copies data from EM51 to file --
*/

readasci(port)
int port;
 {
   extern FILE *fopen();
   char c, filename[64], buf[2];
   FILE *fp;
   int count, ch, pchar, flag, copy, bytcnt, x, z, res;
   extern char scrn;

   putchar(13); printf("driver: Name of file to copy to? ");
   getline(filename,64);
   if (*filename == 0)
      { printf("driver: no action taken"); commout(port,CR); return(-1); }
   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); return(-1); }
      }
   fp = fopen(filename,"w+b");
   if (fp == 0)
      { printf("driver: Can't open ", filename); commout(port,CR); return(-1); }
   commout(port,'c');
   printf("driver: Type command, Copy will activate upon next <CR>...");
   delay(1); 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)
          { bdos(2,c); if (c == BS) { bdos(2,' '); bdos(2,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(1);
                      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 == '$')
                    {
                      commout(port,c);
                      printf("\ndriver: Copy aborted by operator...");
                      delay(1);
                      commflush(port);
                      commout(port,CR);
                      break;
                    }
                else if (flag) flag = 0;
                else if (c == 0) flag = 1;
                else commout(port,c);
              }
    if (z != 0 || c == '$' || res == EOF)
          {
            fclose(fp);
            unlink(filename);
            if ((z != 0 || res == EOF) && !scrn) commout(port,CR);
            return(-1);
          }
       else fclose(fp);
    if (!scrn) commout(port,CR);
 }

/*
help()          -- prints help to terminal --
*/

help()
 {
   int x, oldloc;

   oldloc = savloc();
   locate(0,0);
   eel();
   printf("Function Keys:\n");
   eel();
   putchar('\n');
   eel();
   printf("     F1  --  Download a file to EM51");
   printf("\tF2  --  Copy from EM51 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);
}
```

```
/******************** SCRNIO.C ************************/
#include <dos.h>

/*=========================================================================

                    initscr.c  -  initialize CRT screen                  */
cls()
{
   union REGS reg;
   reg.x.ax = 0x0002;                  /* . . .set mode to BW 80X25 */
   int86(0x10,&reg,&reg);              /* . . . . .execute interrupt */

   reg.x.ax = 0x0500;                  /* . . . . .select page zero */
   int86(0x10,&reg,&reg);

   reg.x.ax = 0x0600;                  /* . . . clear entire screen */
   reg.x.bx = 0x0200;                  /* . . . . .  normal display */
   reg.x.cx = 0x0000;                  /* . . . . upper left corner */
   reg.x.dx = 0x184F;                  /* . . .  lower right corner */
   int86(0x10,&reg,&reg);
   return(0);
}

/*=========================================================================

                        Address Cursor                                   */
locate(row,col)
int row,col;
{
   union REGS reg;

   reg.x.ax = 0x0200;
   reg.x.bx = 0x0000;
   reg.x.dx = (row <<8 ) + col;
   int86(0x10,&reg,&reg);
   return(0);
}

/*=========================================================================

                        Erase to end of line                             */
eeol()
{
   union REGS sreg,rreg;

   sreg.x.ax = 0x0300;              /* read cursor position */
   sreg.x.bx = 0x0000;
   int86(0x10,&sreg,&rreg);

   sreg.x.cx = rreg.x.dx;           /* erase */
   sreg.x.ax = 0x0600;
   sreg.x.bx = 0x0200;
   sreg.x.dx = (rreg.x.dx & 0xFF00) + 0x4F;
   int86(0x10,&sreg,&rreg);
```

```
    return(0);
}

/*=========================================================================

                        Erase entire line                              */
eel()
{
    union REGS sreg,rreg;

    sreg.x.ax = 0x0300;     /* read cursor postion */
    sreg.x.bx = 0x0000;
    int86(0x10,&sreg,&rreg);

    sreg.x.cx = rreg.x.dx & 0xFF00;
    sreg.x.ax = 0x0600;
    sreg.x.bx = 0x0200;
    sreg.x.dx = (rreg.x.dx & 0xFF00) + 0x4F;
    int86(0x10,&sreg,&rreg);
    return(0);
}
/*+++
*/
/*=========================================================================

                    Save cursor location                               */
savloc()
{
    union REGS sreg,rreg;

    sreg.x.ax = 0x0300;     /* Read cursor pos */
    sreg.x.bx = 0x0000;
    int86(0x10,&sreg,&rreg);

    return(rreg.x.dx);
}

/*=========================================================================

                Restore previously saved cursor location               */
reloc(oldloc)
int oldloc;
{
    union REGS reg;

    reg.x.ax = 0x0200;
    reg.x.bx = 0x0000;
    reg.x.dx = oldloc;
    int86(0x10,&reg,&reg);
}
```

```
;.......... KBDIO.ASM ..............
;
_TEXT   SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  CS:_TEXT
        PUBLIC  _charin
_charin PROC    NEAR
        MOV     AH,06H
        MOV     DL,0FFH
        INT     21H
        MOV     AH,00H
        JNZ     leave
        MOV     AX,-1
leave:  RET
_charin ENDP
_TEXT   ENDS
        END
```

```
;.............. ASYNCIO.ASM ..............
;
;             IBM PC Communications I/O Routines
bufsiz  EQU     1000

DGROUP  GROUP   _DATA
_DATA   SEGMENT WORD PUBLIC 'DATA'
snt_xoff  DB    ?               ;flag to check if an XOFF has been sent
snt_xoff2 DB    ?               ;Flags for COM2:
got_xoff  DB    ?               ;flag to check if an XOFF has been received
got_xoff2 DB    ?
see_xoff  DB    ?               ;flag to see if we are interested in XON/XOFF
see_xoff2 DB    ?
err_flg   DB    ?               ;reception error flag --for COM1:
err_flg2  DB    ?               ;--for COM2:
circ_ct   DW    ?               ;count of characters used in the buffer
circ_ct2  DW    ?
circ_buf  DB    bufsiz DUP(?)   ;allow bufsiz maximum buffered characters
circ_buf2 DB    bufsiz DUP(?)   ;Buffer 2
circ_top  DW    ?
circ_top2 DW    ?
circ_in   DW    ?               ;pointer to last char. placed in buffer
circ_in2  DW    ?
circ_cur  DW    ?               ;pointer to next char. to be retrieved from
circ_cur2 DW    ?               ;the buffer
_DATA   ENDS
        PAGE
_TEXT   SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  CS:_TEXT,DS:DGROUP
;
;A set of C callable functions to support
;interrupt driven character I/O on the  IBM PC.
;Input is buffered, output is polled.
;
        PUBLIC  _comminit       ;initialize the comm port,
        PUBLIC  _uninit         ;remove initialization,
        PUBLIC  _set_xoff       ;enable/disable XON/XOFF,
        PUBLIC  _get_xoff       ;read XON/XOFF state,
        PUBLIC  _rcvd_xoff      ;returns true if XOFF rcvd,
        PUBLIC  _sent_xoff      ;returns true if XOFF sent,
        PUBLIC  _inpcnt         ;returns count of rcv chars,
        PUBLIC  _commin         ;get one char from buffer,
        PUBLIC  _commflush      ;flush input buffer,
        PUBLIC  _commout        ;output a character,
        PUBLIC  _rcv_err        ;reception error
;
;A better description can be found in the comment
;block  in each function.
;

;
;
false   EQU     0
true    EQU     1
base    EQU     03F8H           ;base for serial board
```

```
base2   EQU     02F8H           ;base for serial port #2
;
lcr     EQU     base+3          ;Line control register
ier     EQU     base+1          ;Interrup Enable Register
mcr     EQU     base+4          ;modem control register
lcr2    EQU     base2+3
ier2    EQU     base2+1
mcr2    EQU     base2+4
enbldrdy  EQU   01H             ;enable 'data-ready' interrupt bit
intctlr   EQU   21H             ;OCW 1 FOR 8259 CONTROLLER
enblirq4  EQU   0EFH            ;Enable COMMUNICATIONS (IRQ4)
dataport  EQU   base            ;transmit/receive data port
maskirq4  EQU   10H             ;BIT TO DISABLE COMM INTERRUPT (IRQ4)
enblirq3  EQU   0F7H            ;Enable COMMUNICATIONS (IRQ3) (COM2:)
dataport2 EQU   base2           ;transmit/receive data port
maskirq3  EQU   08H             ;Bit to disable IRQ3

mdmsta  EQU     base+5          ;line status register
mdmmsr  EQU     base+6          ;modem status register
mdmbad  EQU     base            ;lsb baud resgister
mdmbd1  EQU     base+1          ;msb baud rate register
mdmsta2 EQU     base2+5
mdmmsr2 EQU     base2+6
mdmbad2 EQU     base2
mdmbd12 EQU     base2+1
mdmcd   EQU     80H             ;mask for carrier dectect
setbau  EQU     80H             ;code for Divisor Latch Access Bit
mdmtbe  EQU     20H             ;8250 tbe flag
mdmbrk  EQU     40H             ;command code for 8250 break
linmod  EQU     03H             ;line mode=8 bit, no parity
mdmmod  EQU     0BH             ;modem mode = DTR and RTS HIGH
stop2   EQU     04H             ;bit for two stop bits if baud < 300
rs8259  EQU     20H             ;OCW 3 FOR 8259
rstint  EQU     20H             ;specific EOI for COMM interrupt
xoff    EQU     13H             ;XOFF character
xon     EQU     11H             ;XON character
;
;       MISCELLANEOUS EQUATES
;
cr      EQU     13
lf      EQU     10
doscall EQU     33              ;interrupt number for DOS call
cnstat  EQU     11              ;function number for console status
cnin    EQU     1               ;function number for console input
setintvect  EQU 25H             ;set interrupt vector function number
        PAGE
_inthdl PROC    NEAR
inthdlr:STI                     ;interrupts back on
        PUSH    CX
        PUSH    DX
        PUSH    BX
        PUSH    AX
        PUSH    DS
        MOV     AX,DGROUP       ;get DGROUP data segment
        MOV     DS,AX           ;and make that our new data segment
```

```
        MOV     BX,circ_in      ;get circ_buf in ptr
        MOV     DX,dataport     ;get data port number
        IN      AL,DX           ;get received character
        PUSH    AX
        MOV     DX,mdmsta       ;check for receiver error
        IN      AL,DX
        AND     AL,0EH
        JZ      cont
        MOV     err_flg,1
cont:   POP     AX
        XOR     CX,CX
        MOV     CL,see_xoff     ;check if interested in XON/XOFF
        CMP     CL,true
        JNZ     ck_full         ;not interested goto ck if buf full
        MOV     CL,AL           ;put char in cl for testing
        AND     CL,7FH          ;turn off any parity bits
        CMP     CL,xoff         ;see if we got an xoff
        JNZ     ck_xon
        MOV     got_xoff,true   ;code for handling XON/XOFF from remote
        JMP     clnup
ck_xon: CMP     CL,xon
        JNZ     reg_ch
        MOV     got_xoff,false
        JMP     clnup
;
;Normal character; not  XON/XOFF, or XON/XOFF disabled.
;
reg_ch: TEST    snt_xoff,true   ;see if sentxoff is set
        JNZ     ck_full         ;if so, don't send another XOFF
        CMP     circ_ct,(bufsiz * 3)/4  ;allow buf to become 3/4 full before
                                        ;sending XOFF
        JB      savch           ;if it's OK, continue
        PUSH    AX              ;save character
        MOV     CL,xoff         ;get XOFF character
        MOV     snt_xoff,true   ;reset sentxoff
        CALL    comout          ;and send it
        POP     AX              ;retrieve character
        JMP     SHORT savch     ;if we're here, the circ_buf has bufsiz-80H
                                ;characters
ck_full:CMP     circ_ct,bufsiz  ;see if circ_buf already full
        JZ      clnup           ;jump if so, do not place character in bfr

savch:  MOV     [BX],AL         ;save new character in circ_buf
        INC     circ_ct         ;bump circ_buf count
        CMP     BX,circ_top     ;are we at the top of the circ_buf?
        JZ      reset_in        ;jump if so
        INC     BX              ;else, bump ptr
        JMP     SHORT into_buf
reset_in:
        LEA     BX,circ_buf     ;reset circ_in to bottom of buf.
into_buf:
        MOV     circ_in,BX      ;save new ptr
clnup:
        MOV     AL,rstint
        OUT     rs8259,AL       ;issue specific EOI for 8259
```

```
        POP     DS              ;get back entering DS
        POP     AX
        POP     BX
        POP     DX
        POP     CX
        IRET
_inthdl ENDP
        PAGE
;Interrupt Handler for COM2:

_inthdl2 PROC   NEAR
inthdlr2:
        STI                     ;interrupts back on
        PUSH    CX
        PUSH    DX
        PUSH    BX
        PUSH    AX
        PUSH    DS
        MOV     AX,DGROUP       ;get DGROUP data segment
        MOV     DS,AX           ;and make that our new data segment
        MOV     BX,circ_in2     ;get circ_buf in ptr
        MOV     DX,dataport2    ;get data port number
        IN      AL,DX           ;get received character
        PUSH    AX
        MOV     DX,mdmsta2      ;check for receiver errors
        IN      AL,DX
        AND     AL,0EH
        JZ      cont2
        MOV     err_flg2,1
cont2:  POP     AX
        XOR     CX,CX
        MOV     CL,see_xoff2    ;check if interested in XON/XOFF
        CMP     CL,true
        JNZ     ck_full2        ;not interested goto ck if buf full
        MOV     CL,AL           ;put char in cl for testing
        AND     CL,7FH          ;turn off any parity bits
        CMP     CL,xoff         ;see if we got an xoff
        JNZ     ck_xon2
        MOV     got_xoff2,true  ;code for handling XON/XOFF from remote
        JMP     clnup3
ck_xon2:CMP     CL,xon
        JNZ     reg_ch2
        MOV     got_xoff2,false
        JMP     clnup3
;
;Normal character; not  XON/XOFF, or XON/XOFF disabled.
;
reg_ch2:TEST    snt_xoff2,true  ;see if sentxoff is set
        JNZ     ck_full2        ;if so, don't send another XOFF
        CMP     circ_ct2,(bufsiz * 3)/4  ;allow buf to become 3/4 full before
                                         ;sending XOFF
        JB      savch2          ;if it's OK, continue
        PUSH    AX              ;save character
        MOV     CL,xoff         ;get XOFF character
        MOV     snt_xoff2,true  ;reset sentxoff
```

```
        CALL    comout2         ;and send it
        POP     AX              ;retrieve character
        JMP     SHORT savch2    ;if we're here, the circ_buf has bufsiz-80H
                                ;characters
ck_full2:
        CMP     circ_ct2,bufsiz ;see if circ_buf already full
        JZ      clnup3          ;jump if so, do not place character in bfr
savch2: MOV     [BX],AL         ;save new character in circ_buf
        INC     circ_ct2        ;bump circ_buf count
        CMP     BX,circ_top2    ;are we at the top of the circ_buf?
        JZ      reset_in2       ;jump if so
        INC     BX              ;else, bump ptr
        JMP     SHORT into_buf2
reset_in2:
        LEA     BX,circ_buf2    ;reset circ_in to bottom of buf.
into_buf2:
        MOV     circ_in2,BX     ;save new ptr
clnup3:
        MOV     AL,rstint
        OUT     rs8259,AL       ;issue specific EOI for 8259
        POP     DS              ;get back entering DS
        POP     AX
        POP     BX
        POP     DX
        POP     CX
        IRET
_inthdl2 ENDP
        PAGE
;
;
;
;set_xoff(port,flag)        Enable (flag != 0) or disable
;int flag;                 (flag == 0) XON/ XOFF protocol
;int port;                 for the character input stream.
;If enabled, an XOFF will be sent when  the buffer
;reaches 3/4 full. NOTE: an XON WILL be sent auto-
;matically if this is enabled. 3/28/85 CCH
;
;
_set_xoff PROC   NEAR
        PUSH    BP
        MOV     BP,SP
        MOV     BX,[BP+4]
        CMP     BX,1
        JNZ     port2
        MOV     BX,[BP+6]
        CMP     BX,0
        JNZ     to_on
        MOV     see_xoff,false
        JMP     done1
to_on:  MOV     see_xoff,true
        JMP     done1
port2:  MOV     BX,[BP+6]
        CMP     BX,0
        JNZ     to_on2
```

```
        MOV     see_xoff2,false
        JMP     done1
to_on2: MOV     see_xoff2,true
done1:  POP     BP
        RET
_set_xoff ENDP
        PAGE
;
;flag = get_xoff(port)  Returns the current setting
;int port;              of the XON/ XOFF flag set
;                       by set_xoff(), above.
;
_get_xoff PROC   NEAR
        PUSH    BP
        MOV     BP,SP
        XOR     AX,AX
        MOV     BX,[BP+4]
        CMP     BX,1
        JNZ     get2
        MOV     AL,see_xoff
        JMP     gone
get2:   MOV     AL,see_xoff2
gone:   POP     BP
        RET
_get_xoff ENDP
        PAGE
;
;flag = sent_xoff(port);    Returns true if an XOFF
;int port;                  character was sent, indicating
;                           the receive buffer is  3/4 full.
;
_sent_xoff PROC   NEAR
        PUSH    BP
        MOV     BP,SP
        XOR     AX,AX
        MOV     BX,[BP+4]
        CMP     BX,1
        JNZ     sent2
        MOV     AL,snt_xoff
        JMP     later
sent2:  MOV     AL,snt_xoff2
later:  POP     BP
        RET
_sent_xoff ENDP
        PAGE
;
;flag = rcv_err(port)   Returns true if a receiver error
;                       has occured on the specified port
;
_rcv_err PROC    NEAR
        PUSH    BP
        MOV     BP,SP
        XOR     AX,AX
        MOV     BX,[BP+4]
        CMP     BX,1
```

```
        JNZ     rcv_e2
        MOV     AL,err_flg
        JMP     leav
rcv_e2: MOV     AL,err_flg2
leav:   POP     BP
        RET
_rcv_err ENDP
        PAGE
;
;flag = rcvd_xoff(port) Returns true if an XOFF was
;                       received; will return false as
;soon as an XON is received. Does not effect data output,
;only indicates the above. (Obviously useless for binary
;data.)
;
_rcvd_xoff PROC   NEAR
        PUSH    BP
        MOV     BP,SP
        XOR     AX,AX
        MOV     BX,[BP+4]
        CMP     BX,1
        JNZ     rcvd2
        MOV     AL,got_xoff
        JMP     lateron
rcvd2:  MOV     AL,got_xoff2
lateron:POP     BP
        RET
_rcvd_xoff ENDP
        PAGE
;
;count = inpcnt(port)   Returns the number of characters
;int port;              available in the input buffer.
;

_inpcnt PROC    NEAR
        PUSH    BP
        MOV     BP,SP
        MOV     BX,[BP+4]
        CMP     BX,1
        JNZ     cnt2
        MOV     AX,circ_ct
        JMP     seeya
cnt2:   MOV     AX,circ_ct2
seeya:  POP     BP
        RET
_inpcnt ENDP
        PAGE
;
;commflush(port)        Flush the input buffer.
;int port;
;
_commflush PROC   NEAR
        PUSH    BP
        MOV     BP,SP
        MOV     BX,[BP+4]
```

```
        CMP     BX,1
        JNZ     flush2
        MOV     err_flg,false
        MOV     snt_xoff,false
        MOV     got_xoff,false
        LEA     BX,circ_buf
        MOV     circ_in,BX
        MOV     circ_cur,BX
        XOR     AX,AX
        MOV     circ_ct,AX
        JMP     adios
flush2: MOV     err_flg2,false
        MOV     snt_xoff2,false
        MOV     got_xoff2,false
        LEA     BX,circ_buf2
        MOV     circ_in2,BX
        MOV     circ_cur2,BX
        XOR     AX,AX
        MOV     circ_ct2,AX
adios:  POP     BP
        RET
_commflush ENDP
        PAGE
;
; --------- Init -----------------------------------
; Program initialization:
;   --  Set up vector for RS232 interrupt (0CH) or (0BH)
;   --  Enbl IRQ4 or IRQ3
;   --  Enbl RS232 interrupt on data ready
;
; --------------------------------------------------

_cominit PROC   NEAR
        PUSH    BP
        MOV     BP,SP

;  ---- Which port to init?

        MOV     BX,[BP+4]
        CMP     BX,1
        JNZ     init2

;  ---- Set up  INT x'0C' for IRQ4 (COM1:)

        MOV     err_flg,false
        MOV     snt_xoff,false
        MOV     got_xoff,false
        MOV     see_xoff,false
        MOV     circ_ct,0
        LEA     BX,circ_buf     ;initialize address pointers to buffer
        MOV     circ_in,BX
        MOV     circ_cur,BX
        ADD     BX,bufsiz-1
        MOV     circ_top,BX
        PUSH    DS              ;save DS for the moment...
```

```
        PUSH    CS
        POP     DS
        LEA     DX,inthdlr      ;relative adddres of interrupt handler
        MOV     AL,0CH          ;interrupt number for comm.
        MOV     AH,setintvect   ;function number for setting int vector
        INT     doscall         ;set interrupt in 8086 table
        POP     DS              ;get back DS.

;  ---- Enbl IRQ4 on 8259 interrupt controller

        CLI
        IN      AL,intctlr      ;get current masks
        AND     AL,enblirq4     ;Reset IRQ4 mask
        OUT     intctlr,AL      ;And restore to IMR

;  ---- Enbl 8250 data ready interrupt

        MOV     DX,lcr          ;DX ==> LCR
        IN      AL,DX           ;Reset DLAB for IER access
        AND     AL,7FH
        OUT     DX,AL
        MOV     DX,ier          ;Interrupt Enbl Register
        MOV     AL,enbldrdy     ;Enable 'data-ready' interrupt
        OUT     DX,AL

;  ---- Enbl OUT2 on 8250

        MOV     DX,mcr          ;modem control register
        MOV     AL,08H          ;Enable OUT2
        OUT     DX,AL
        STI
        JMP     goodday

;  ---- Set up  INT x'0B' for IRQ3 (COM2:)

init2:  MOV     err_flg2,false
        MOV     snt_xoff2,false
        MOV     got_xoff2,false
        MOV     see_xoff2,false
        MOV     circ_ct2,0
        LEA     BX,circ_buf2    ;initialize address pointers to buffer
        MOV     circ_in2,BX
        MOV     circ_cur2,BX
        ADD     BX,bufsiz-1
        MOV     circ_top2,BX
        PUSH    DS
        PUSH    CS
        POP     DS
        LEA     DX,inthdlr2     ;relative adddres of interrupt handler
        MOV     AL,0BH          ;interrupt number for comm.
        MOV     AH,setintvect   ;function number for setting int vector
        INT     doscall         ;set interrupt in 8086 table
        POP     DS

;  ---- Enbl IRQ3 on 8259 interrupt controller
```

```
        CLI
        IN      AL,intctlr      ;get current masks
        AND     AL,enblirq3     ;Reset IRQ4 mask
        OUT     intctlr,AL      ;And restore to IMR

;  ---   Enbl 8250 data ready interrupt

        MOV     DX,lcr2         ;DX ==> LCR
        IN      AL,DX           ;Reset DLAB for IER access
        AND     AL,7FH
        OUT     DX,AL
        MOV     DX,ier2         ;Interrupt Enbl Register
        MOV     AL,enbldrdy     ;Enable 'data-ready' interrupt
        OUT     DX,AL

;  ---   Enbl OUT2 on 8250

        MOV     DX,mcr2         ;modem control register
        MOV     AL,08H          ;Enable OUT2
        OUT     DX,AL
        STI

goodday:POP     BP
        RET
_comminit ENDP
        PAGE
;
;uninit(port)            Removes the interrupt structure
;int port;               installed by comminit(). Must be
;done before passing control to the DOS, else chars received
;will be stored into the next program loaded!
;
_uninit PROC    NEAR
        PUSH    BP
        MOV     BP,SP

; ---   Which port to uninit?

        MOV     BX,[BP+4]
        CMP     BX,1
        JNZ     uninit2

; ---   Disable IRQ4 on 8259

        CLI
        IN      AL,intctlr      ;get OCW1 from 8259
        OR      AL,maskirq4     ;disable communications interrupt
        OUT     intctlr,AL

; ---   Disable 8250 data ready interrupt

        MOV     DX,lcr          ;DX ==> LCR
        IN      AL,DX           ;Reset DLAB for IER access
        AND     AL,7FH
```

```
        OUT     DX,AL
        MOV     DX,ier          ;Interrupt Enbl Register
        MOV     AL,0            ;Disable all 8250 interrupts
        OUT     DX,AL

;  ---   Disable OUT2 on 8250

        MOV     DX,mcr          ;modem control register
        MOV     AL,0            ;Disable OUT2
        OUT     DX,AL
        STI
        JMP     laterdays

; ---   Disable IRQ3 on 8259

uninit2:CLI
        IN      AL,intctlr      ;get OCW1 from 8259
        OR      AL,maskirq3     ;disable communications interrupt
        OUT     intctlr,AL

; ---   Disable 8250 data ready interrupt

        MOV     DX,lcr2         ;DX ==> LCR
        IN      AL,DX           ;Reset DLAB for IER access
        AND     AL,7FH
        OUT     DX,AL
        MOV     DX,ier2         ;Interrupt Enbl Register
        MOV     AL,0            ;Disable all 8250 interrupts
        OUT     DX,AL

;  ---   Disable OUT2 on 8250

        MOV     DX,mcr2         ;modem control register
        MOV     AL,0            ;Disable OUT2
        OUT     DX,AL
        STI

laterdays:
        POP     BP
        RET
_uninit ENDP
        PAGE
;
;char commin(port)       Return a character from the input
;int port;               buffer. Assumes you have called
;                        inpcnt() to see if theres any characters to get.
;
_commin PROC    NEAR
        PUSH    BP
        MOV     BP,SP
        MOV     BX,[BP+4]
        CMP     BX,1
        JNZ     in2
        MOV     BX,circ_cur
        XOR     AX,AX
```

```
        MOV     AL,[BX]         ;get next char from circ_buf
        DEC     circ_ct         ;decrement circ_buf count
        CMP     BX,circ_top     ;are we at the top of the circ_buf?
        JZ      reset_cur       ;jump if so
        INC     BX              ;else, bump ptr
        JMP     SHORT upd_cur
reset_cur:
        LEA     BX,circ_buf     ;reset circ_in to bottom of buf.
upd_cur:MOV     circ_cur,BX     ;save new ptr
        XOR     CX,CX
        MOV     CL,see_xoff     ;check if interested in XON/XOFF
        CMP     CL,true
        JNZ     clnup2          ;not interested, so goto return
        CMP     snt_xoff,true   ;have we sent an xoff?
        JNZ     clnup2          ;no, so return
        CMP     circ_ct,80H     ;yes, so see in buf is now emptying
        JG      clnup2          ;not empty enuf to send xon, jump to ret
        MOV     snt_xoff,false
        MOV     CL,xon
        PUSH    AX              ;save char
        CALL    comout
        POP     AX
        JMP     clnup2

in2:    MOV     BX,circ_cur2
        XOR     AX,AX
        MOV     AL,[BX]         ;get next char from circ_buf
        DEC     circ_ct2        ;decrement circ_buf count
        CMP     BX,circ_top2    ;are we at the top of the circ_buf?
        JZ      reset_cur2      ;jump if so
        INC     BX              ;else, bump ptr
        JMP     SHORT upd_cur2
reset_cur2:
        LEA     BX,circ_buf2    ;reset circ_in to bottom of buf.
upd_cur2:
        MOV     circ_cur2,BX    ;save new ptr
        XOR     CX,CX
        MOV     CL,see_xoff2    ;check if interested in XON/XOFF
        CMP     CL,true
        JNZ     clnup2          ;not interested, so goto return
        CMP     snt_xoff2,true  ;have we sent an xoff?
        JNZ     clnup2          ;no, so return
        CMP     circ_ct2,80H    ;yes, so see in buf is now emptying
        JG      clnup2          ;not empty enuf to send xon, jump to ret
        MOV     snt_xoff2,false
        MOV     CL,xon
        PUSH    AX              ;save char
        CALL    comout2
        POP     AX
        JMP     clnup2

clnup2: POP     BP
        RET
_commin ENDP
        PAGE
```

```
;
;commout(port,c)         Output the character to the
;char c;                 serial port. This is not buffered
;int port;               or interrupt driven.
;
_commout PROC    NEAR
        PUSH    BP
        MOV     BP,SP
        MOV     BX,[BP+4]
        CMP     BX,1
        JNZ     out2
        MOV     CL,[BP+6]
        STI
        CALL    comout
        JMP     fine
out2:   MOV     CL,[BP+6]
        STI
        CALL    comout2
fine:   POP     BP
        RET
_commout ENDP
        PAGE
;
;Local  subroutine: output CL to the port.
;
comout: MOV     DX,mdmsta
        IN      AL,DX           ;get 8250 status
        AND     AL,mdmtbe       ;check for transmitter ready
        JZ      comout          ;jump if not to wait
        MOV     AL,CL           ;get char to al
        MOV     DX,dataport
        OUT     DX,AL           ;output char to 8251
        RET

comout2:MOV     DX,mdmsta2
        IN      AL,DX           ;get 8250 status
        AND     AL,mdmtbe       ;check for transmitter ready
        JZ      comout2         ;jump if not to wait
        MOV     AL,CL           ;get char to al
        MOV     DX,dataport2
        OUT     DX,AL           ;output char to 8251
        RET
_TEXT   ENDS
        END
```

ภาคผนวกที่ 3

โปรแกรมควบคุมระบบของ 8051 อีมูเลเตอร์

```
;USE 8051 TABLE AND HEX OUTPUT FILE IS SHORT INTEL FORMATT
          CPU       "8051.TBL"
          HOF       "INT8"
;
TITL "8051 IN CIRCUIT EMULATOR Ver 1.0 by CHARIN B."
;
;SYSTEM DEFINE:-
;         THIS PROGRAM RUN ON CPU 8031 WHICH-
;         MONITOR PROGRAM     ADD 0000H - 3FFFH (16K)
;         TERMINAL LINK (UART) ADD 4000H - 4001H
;         EXTERNAL PAR PORT   ADD 6000H - 6003H
;         APPLICATION RAM     ADD 8000H - FFFFH (32K)
;*****************************************************************
;
;MCS-51 INTERNAL REGISTERS ASSIGNMENT.
;
P0:         EQU       80H       ;PORT 0
SP:         EQU       81H       ;STACK POINTER
DPL:        EQU       82H       ;DATA POINTER (LOW)
DPH:        EQU       83H       ;DATA POINTER (HIGH)
PCON:       EQU       87H       ;POWER CONTROL REGISTER
TCON:       EQU       88H       ;TIMER REG
TMOD:       EQU       89H       ;TIM MODE REG
TL0:        EQU       8AH       ;TIM 0 LOW BYTE
TL1:        EQU       8BH       ;TIM 1 LOW BYTE
TH0:        EQU       8CH       ;TIM 0 HIGH BYTE
TH1:        EQU       8DH       ;TIM 1 HIGH BYTE
P1:         EQU       90H       ;PORT 1
SCON:       EQU       98H       ;SERIAL PORT CONTROL REG
SBUF:       EQU       99H       ;SERIAL PORT DATA BUFF
P2:         EQU       0A0H      ;PORT 2
IE:         EQU       0A8H      ;INTERRUPT ENABLE REG
P3:         EQU       0B0H      ;PORT 3
IP:         EQU       0B8H      ;INTERRUPT PRIORITY REG
PSW:        EQU       0D0H      ;PROG STAT WORD
ACC:        EQU       0E0H      ;ACCUMULATOR (DIRECT ADD)
B:          EQU       0F0H      ;B REG
;
;MCS-51 INTERNAL BIT ADDRESS ASSIGNMENT.
;
CY:         EQU       0D7H      ;CARRY FLAG
AC:         EQU       0D6H      ;AUXILIARY-CARY FLAG
F0:         EQU       0D5H      ;USER FLAG 0
RS1:        EQU       0D4H      ;REGISTER SELECT MSB
RS0:        EQU       0D3H      ;REGISTER SELECT LSB
OV:         EQU       0D2H      ;OVERFLOW FLAG
P:          EQU       0D0H      ;PARITY FLAG
PS:         EQU       0BCH      ;PRIORITY SERIAL PORT
PT1:        EQU       0BBH      ;PRIORITY TIMER 1
PX1:        EQU       0BAH      ;PRIORITY EXTERNAL 1
PT0:        EQU       0B9H      ;PRIORITY TIMER 0
PX0:        EQU       0B8H      ;PRIORITY EXTERNAL 0
EA:         EQU       0AFH      ;ENABLE ALL INTERRUPT
ES:         EQU       0ACH      ;ENABLE SERIAL INTERRUPT
ET1:        EQU       0ABH      ;ENABLE TIMER 1 INTERUPT
```

```
EX1:        EQU         0AAH        ;ENABLE EXTERNAL 1 INTERR
ET0:        EQU         0A9H        ;ENABLE TIMER 0 INTERRUPT
EX0:        EQU         0A8H        ;ENABLE EXTERNAL 0 INTERR
SM0:        EQU         09FH        ;SERIAL MODE 0
SM1:        EQU         09EH        ;SERIAL MODE 1
SM2:        EQU         09DH        ;SERIAL MODE 2
REN:        EQU         09CH        ;SERIAL RECEPTION ENABLE
TB8:        EQU         09BH        ;TRANSMITT BIT 8
RB8:        EQU         09AH        ;RECEIVE BIT 8
TI:         EQU         099H        ;TRANSMIT INTERRUPT FLAG
RI:         EQU         098H        ;RECEIVE INTERRUPT FLAG
TF1:        EQU         08FH        ;TIMER 1 OVERFLOW FLAG
TR1:        EQU         08EH        ;TIMER 1 RUN CONTROL BIT
TF0:        EQU         08DH        ;TIMER 0 OVERFLOW FLAG
TR0:        EQU         08CH        ;TIMER 0 RUN CONTROL BIT
IE1:        EQU         08BH        ;EXT INTERR. 1 EDGE FLAG
IT1:        EQU         08AH        ;EXT INTERR. 1 TYPE FLAG
IE0:        EQU         089H        ;EXT INTERR. 0 EDGE FLAG
IT0:        EQU         088H        ;EXT INTERR. 0 TYPE FLAG
;
;MONITOR PROGRAM EQUATE.
;
MONBASE:    EQU         0000H       ;8K MONITOR
APPBASE:    EQU         2000H       ;8K APPLICATION EPROM
RAMBASE:    EQU         8000H       ;32K RAM
UARTBASE:   EQU         4000H       ;TERMINAL (uart) LOCATION
UARTDATA:   EQU         UARTBASE    ; C/D* = A0
UARTCONT:   EQU         UARTBASE+1  ;CONTROL AND STATUS WORD
PPIBASE:    EQU         6000H       ;PPI LOCATION
PPIPA:      EQU         PPIBASE     ;DATA TO PORT A
PPIPB:      EQU         PPIBASE+1   ;DATA TO PORT B
PPIPC:      EQU         PPIBASE+2   ;DATA TO PORT C
PPICONT:    EQU         PPIBASE+3   ;CONTROL WORD
RAMEND:     EQU         0FFFFH      ;END OF EXTERNAL RAMSPACE
BYTENUM:    EQU         RAMEND-8    ;RENAME HERE ON USED BY BREAK ROUTINE
HIADD:      EQU         0FFF0H      ;SAVE HI START ADD WHEN DOWNLOAD TO RAM
LOADD:      EQU         0FFF1H      ;SAVE LO START ADD WHEN DOWN LOAD TO RAM
DATBUF:     EQU         0FFF0H      ;SAVE DATA FOR SEARCH
DATCNT:     EQU         0FFF7H      ;SAVE BYTE COUNT FOR DATA TO SEARCH
STACK:      EQU         50H         ;STACK BEGINS HERE
;
;RESERVED INTERNAL RAM LOCATIONS
;
RESERVED:   EQU         48H          ;RESERVED INTERNAL RAM
HIBYTE:     EQU         RESERVED     ;TOP OF 16-BIT ADDRESS
LOBYTE:     EQU         RESERVED+1   ;BOTTOM OF 16 BIT ADDRESS
FIRST:      EQU         RESERVED+2   ;START ADDRESS HIGH (LOW IN RESERVED+3)
LAST:       EQU         RESERVED+4   ;END ADDRESS HIGH (LOW IN RESERVED+5)
TO:         EQU         RESERVED+6   ;TO ADDRESS HIGH (LOW IN RESERVED+7)
;
;RESERVED REGISTER BANK (AND ALIASES FOR THOSE REGISTERS)
;
RESBANK:    EQU         1           ;REGISTER BANK 1
REG0:       EQU         8*RESBANK
REG1:       EQU         REG0+1      ;SOMETIME CAN'T USE R1
```

```
REG2:     EQU       REG0+2    ;THEN ASSIGN OTHER ONES
REG3:     EQU       REG0+3
REG4:     EQU       REG0+4
REG5:     EQU       REG0+5
REG6:     EQU       REG0+6
REG7:     EQU       REG0+7
;
;CHARACTER EQUATES
;
CR:       EQU       0DH       ;CARRIAGE RETURN
LF:       EQU       0AH       ;LINE FEED
SPACE:    EQU       20H       ;SPACE CHARACTER
TAB:      EQU       09H       ;TAB CHARACTER
DOT:      EQU       2EH       ;PERIOD
BACKSP:   EQU       08H       ;BACKSPACE
EOT:      EQU       04H       ;end of text
;
;*************************************************************
;
          ORG       MONBASE
START:    LJMP      BGNHERE
;
;RESERVED INTERUPT AREA FOR USER  PROGRAM
;
          ORG       START+03H
          LJMP      0FFE0H              ;INTERRUPT VECTORS JUMP TO RAM
;
          ORG       START+0BH
          LJMP      0FFE3H
;
          ORG       START+13H
          LJMP      0FFE6H
;
          ORG       START+1BH
          LJMP      0FFE9H
;
          ORG       START+23H
          LJMP      0FFECH
;
;*************************************************************
;
LJMPTBL:  LJMP      COOL      ;JUMP TABLE FOR USER PROGRAM
          LJMP      WARM
          LJMP      IN
          LJMP      INCH
          LJMP      INHEX
          LJMP      BYTES
          LJMP      BADDR
          LJMP      THRADR
          LJMP      OUT
          LJMP      OUTCH
          LJMP      OUTS
          LJMP      OUT2S
          LJMP      CRLF
          LJMP      OUT2H
```

```
          LJMP      OUTR0
          LJMP      OUTC2HS
          LJMP      PDATA
          LJMP      INC16
          LJMP      DEC16
          LJMP      CPY
          LJMP      BRKPT
;
;THE MSSG FOLLOWING BELOW MUST DISPLAY ON TERMINAL IF ICE-51 READY.
;
MSIGNON:
          DFB       CR, LF, "MCS-51 Real Time In Circuit Emulator version 1.00"
          DFB       CR, LF, "Copyright 1986 by CHARIN  B. of KMITL", LF, 04H
;
;************************************************************
;
BGNHERE:  MOV SP,#STACK         ;BGN OF ADD OF STACK
          MOV STACK,SP          ;STORE STACK POINTER FOR COOL START
          SETB RS0              ;SELECT REGISTER BANK 1
          CLR RS1               ;CLEAR DATA
          CLR A
          MOV DPTR,#BYTENUM     ;BREAK POINT
          MOV R0,#{RAMEND-BYTENUM+1}
CLEAR:    MOVX @DPTR,A          ;CLEAR SPACE USED BY BREAK ROUTINE
          INC DPTR              ;NEXT
          DJNZ R0,CLEAR
DLAY:     DJNZ ACC,DLAY         ;WAIT HERE FOR UART TO WAKE-UP
          DJNZ R0,DLAY
          MOV DPTR,#UARTCONT    ;INIT UART
          CLR A
          MOVX @DPTR,A          ;SEND THREE ZEROS BECUASE DON'T KNOW
          NOP                   ;WHAT MODE OF UART WAKES UP
          DJNZ ACC,$-1          ;DELAY TIME
          MOVX @DPTR,A
          NOP
          DJNZ ACC,$-1
          MOVX @DPTR,A
          NOP
          DJNZ ACC,$-1
          MOV A,#40H            ;SOFTWARE RESET UART
          MOVX @DPTR,A
          CLR A                 ;DELAY TIME TO MAKE SURE
          NOP
          DJNZ ACC,$-1
          MOV A,#11001110B      ;SET MODE OF UART
          MOVX @DPTR,A
          MOVX A,@DPTR          ;GET STATUS OF UART
;         JNB {ACC OR 0},$-1    ;WAIT FOR TXRDY
          NOP
          NOP
          NOP
          MOV A,#00110111B      ;COMMAND ASSIGNMENT
          MOVX @DPTR,A
          MOV DPTR,#MSIGNON     ;SEND HEADING TO TERMINAL
          LCALL PDATA
```

```
COOL:     MOV SP,STACK          ;SET STACK
          MOV A,#90H            ;SET PA (IN),PB&PC (OUT)
          MOV DPTR,#6003H
          MOVX @DPTR,A
          MOV A,#0FFH
          MOV DPTR,#6002H
          MOVX @DPTR,A
;
;*************************************************************
;
WARM:     MOV DPTR,#MPROMPT     ;WARM START
          LCALL PDATA           ;SEND "PROMPT" TO TERMINAL
          LCALL INCH            ;GET CHARACTER FROM UART
          JNB {ACC OR 6},$+5    ;JUMP IF NOT A LETTER
          CLR {ACC OR 5}        ;CONVERT LOWER TO UPPER CASE
          MOV R7,A              ;SAVE CHARACTER INPUT
          MOV DPTR,#FUNTAB      ;POINT TO FUNCTION TABLE
SCAN:     CLR A
          MOVC A,@A+DPTR
          JZ WARM               ;END OF TABLE
          CJNE A,0FH,NEXT       ;COMPARE WITH SAVED CHARACTER
          MOV A,#01H
          MOVC A,@A+DPTR        ;GET HIGH BYTE OF JUMP
          MOV R2,A              ;SAVE IN R2
          MOV A,#02H
          MOVC A,@A+DPTR        ;GET LOW BYTE OF JUMP
          MOV R3,A              ;SAVE IN R3
          PUSH REG3             ;PUSH R3 (LOW BYTE)
          PUSH REG2             ;PUSH R2 (HIGH BYTE)
          RET                   ;POP & JUMP TO SERVICE ROUTINE
NEXT:     INC DPTR
          INC DPTR
          INC DPTR
          LJMP SCAN             ;WAIT FOR NEXT INSTRUCTION
;
;*************************************************************
;
HELP:     MOV DPTR,#MHELP       ;GET ADD OF HELP MSSG
          LCALL PDATA           ;SEND DATA TO TERMINAL
          LJMP WARM             ;RETURN
;
;*************************************************************
;
DUMP:     LCALL BADDR           ;GET DATA
          JNC DMP3
          CLR {ACC OR 5}        ;CONVERT LOWER TO UPPER CASE
          CJNE A,#"I",DMP1
          LJMP DUMPI            ;DUMP INTERNAL RAM
;
DMP1:     LJMP WARM
;
DMP3:     MOV FIRST,HIBYTE     ;SAVE START ADDRESS TO DUMP
          MOV {FIRST+1},LOBYTE
          MOV LAST,#0FFH
          MOV {LAST+1},#0FFH
```

```
          CJNE A,#CR,DMP4      ;DEFAULT IF NO END ADDRESS ENTERED
          LJMP {NEWLIN-6}
;
DMP4:     LCALL BADDR2         ;GET END ADDRESS TO DUMP
          JB F0,DMP5           ;CARRIAGE RETURN READ
          MOV LAST,HIBYTE      ;SAVE END ADDRESS TO DUMP
          MOV{LAST+1},LOBYTE
DMP5:     MOV DPTR,#MINDEX     ;SEND HEADING
          LCALL PDATA
NEWLIN:
          MOV DPH,FIRST        ;DPTR POINT TO ADDRESS TO DUMP
          MOV DPL,{FIRST+1}
          ANL DPL,#0F0H
          LCALL CRLF           ;SEND NEW LINE
          MOV R0,DPH
          LCALL OUTR0          ;DISPLAY ADDRESS
          MOV R0,DPL
          LCALL OUTR0
          LCALL OUT2S          ;SEND 2 SPACES
SPACES:   MOV A,{FIRST+1}      ;BLANK LOCATIONS BEFORE FIRST
          XRL A,DPL
          JZ MORE              ;JUMP IF EQUAL
          LCALL OUT3S          ;SEND 3 SPACES
          INC DPTR
          LCALL MIDCHK         ;SEPERATE ADDRESS 0-7 AND 8-F
          LJMP SPACES
;
;
MORE:     LCALL OUTC2HS        ;COMPARE IF END ADDRESS TO DUMP
          MOV A,LAST
          CJNE A,DPH,MOREC     ;NO,DUMP NEXT ADDRESS
          MOV A,{LAST+1}
          CJNE A,DPL,MOREC
          LJMP ENDHEX
;
MOREC:    INC DPTR
          LCALL MIDCHK
          JZ DASCII            ;NOW DUMP ASCII
          LJMP MORE
;
ENDHEX:   INC DPTR
          LCALL MIDCHK
          JZ DASCII            ;NOW DUMP ASCII (LAST LINE)
          LCALL OUT3S
          LJMP ENDHEX
;
DASCII:   LCALL OUT4S          ;SHOW ASCII EQUIVALENT
          MOV DPH,FIRST        ;SET POINTER POINT AGAIN
          MOV DPL,{FIRST+1}
          ANL DPL,#0F0H
DASCI1:   MOV A,{FIRST+1}
          XRL A,DPL
          JZ DASCI2
          LCALL OUTS
          INC DPTR
```

```
          LCALL MIDCHK
          LJMP DASCI1
;
DASCI2:   CLR A              ;CONVERT TO ASCII CHARACTER
          MOVC A,@A+DPTR
          MOV B,A
          CLR C
          SUBB A,#SPACE      ;IN ASCII RANGES
          JC BAD3
          SUBB A,#{7FH-SPACE}
          JC OK3
BAD3:     MOV B,#DOT         ;IF NOT ASCII
OK3:      MOV A,B            ;IF ASCII
          LCALL OUTCH        ;DISPLAY TO TERMINAL
;
          MOV A,LAST         ;COMPARE SSSS=FFFF+1?
          CJNE A,DPH,DASCI3
          MOV A,{LAST+1}
          CJNE A,DPL,DASCI3
          LJMP WARM
;
DASCI3:   INC DPTR
          LCALL MIDCHK
          JNZ DASCI2
          MOV A,#16
          ADD A,{FIRST+1}
          ANL A,#0F0H
          JNC $+4
          INC FIRST
          MOV {FIRST+1},A
          LJMP NEWLIN
;
;*************************************************************
;
TABENT:   EQU      11        ;LENGTH OF SFR TABLE ENTRY
OFF1:     EQU      6         ;OFFSET TO HEX ADDRESS
OFF2:     EQU      5         ;OFFSET TO FETCH CONTENTS
OFF3:     EQU      8         ;OFFSET TO CHANGE CONTENTS
;
DUMPI:                       ;DUMP INTERNAL RAM
          LCALL BADDR        ;GET ADDRESS
          JNC $+5
          LJMP WARM
;
          MOV FIRST,LOBYTE   ;SAVE BEGIN ADDRESS
          MOV LAST,#0FFH     ;SET DEFAULT OF END ADDRESS
          XRL A,#CR          ;NEXT CHARACTER IS "CR"?
          JZ {LINE-6}
          LCALL BADDR2       ;YES GET END ADDRESS
          JB F0,$+6          ;CARRIAGE RETURN READ
          MOV LAST,LOBYTE
          MOV A,#7FH         ;TOP OF RAM
          CLR C
          SUBB A,FIRST
          JC DSFR            ;DONE RAM--DO SFR'S
```

```
        MOV DPTR,#MINDEX    ;SEND HEADING TO TERMINAL
        LCALL PDATA
LINE:   LCALL CRLF
        MOV A,#7FH          ;TOP OF RAM
        CLR C
        SUBB A,FIRST
        JC DSFR             ;DONE RAM--DO SFR'S
        MOV R0,FIRST
        ANL REG0,#0F0H      ;8=R0
        LCALL OUTR03S       ;DISPLAY ADDRESS
SPICES: MOV A,FIRST         ;SPACES BEFORE BEGINNING
        XRL A,R0
        JZ NXTBYT
        LCALL OUT3S         ;SEND 3 SPACES
        INC R0
        MOV A,#07H
        ANL A,R0
        JNZ $+5
        LCALL OUTS          ;MIDDLE OF LINE
        LJMP SPICES
;
NXTBYT: LCALL OUT2H         ;SEND DATA TO TERMINAL
        LCALL OUTS
        MOV A,R0
        XRL A,LAST
        JNZ $+5
        LJMP WARM
;
        INC R0
        MOV A,#07H
        ANL A,R0
        JNZ $+5
        LCALL OUTS          ;MIDDLE OF LINE
        MOV A,#0FH
        ANL A,R0
        JNZ NXTBYT          ;NOT END OF LINE
        MOV A,FIRST
        ANL A,#0F0H
        ADD A,#10H
        MOV FIRST,A
        LJMP LINE
;
DSFR:   MOV DPTR,#{SFRTAB-TABENT}    ;IN CASE OF SPACIAL FUNCTION
        MOV R1,#6
        LCALL CRLF
DSFR1:  MOV A,#TABENT
        ADD A,DPL
        MOV DPL,A
        JNC $+4
        INC DPH
        MOV A,#0FF1         ;OFFSET TO SFR HEX LOCATION
        MOVC A,@A+DPTR
        CLR C
        SUBB A,FIRST
        JC DSFR1            ;NOT TO FIRST YET
```

```
;
DSFR5:    MOV A,#OFF1
          MOVC A,@A+DPTR       ;GET HEX LOCATION OF SFR
          MOV R0,FIRST
          XRL A,R0
          JZ DSFR10
          INC FIRST            ;WASN'T A VALID SFR
          MOV A,FIRST
          JZ {LINE-6}          ;BACK TO RAM
          CJNE A,LAST,DSFR5
          LJMP WARM            ;DONE
;
DSFR10:   LCALL OUTR0
          MOV A,#"="
          LCALL OUTCH
          LCALL PDATA          ;OUTPUT LOCATION LABEL (ALTERS DPTR)
          MOV A,#":"
          LCALL OUTCH
          MOV A,#LOW{DSFR20}
          PUSH ACC
          MOV A,#HIGH{DSFR20}
          PUSH ACC
          MOV A,#{OFF2-4}      ;OFFSET TO "MOV R0,SFR"
          JMP @A+DPTR
;
DSFR20:   LCALL OUTR0
          LCALL OUT2S
          DJNZ R1,DSFR30
          MOV R1,#6
          LCALL CRLF
;
DSFR30:   INC FIRST
          MOV A,#{TABENT-4}
          ADD A,DPL
          MOV DPL,A
          JNC $+4
          INC DPH
          MOV A,FIRST
          JNZ $+5
          LJMP LINE
          DEC A
          XRL A,LAST
          JNZ DSFR5
          LJMP WARM
;
;
;************************************************************
;
COPY:     LCALL THRADR         ;GET 3  ADDRESS(SSSS,FFFF,TTTT)
          LCALL CPY            ;COPY DATA
          LJMP WARM
;
;************************************************************
;
VERIFY:   LCALL THRADR         ;GET 3 ADDRESS
```

```
VER10:  MOV DPH,FIRST       ;GET START ADDRESS SSSS
        MOV DPL,{FIRST+1}
        CLR A
        MOVC A,@A+DPTR
        MOV R1,A            ;SAVE DATA IN R1
        MOV DPH,TO          ;GET TARGET ADDRESS
        MOV DPL,{TO+1}
        CLR A
        MOVC A,@A+DPTR
        MOV R2,A            ;SAVE DATA IN R2
        XRL A,R1
        JZ VER20            ;JUMP IF DATA EQUAL
        LCALL CRLF          ;IF NOT DISPLAY BOTH DATA
        MOV R0,#FIRST       ;SEND ADDRESS SSSS
        LCALL OUT2H
        MOV R0,#{FIRST+1}
        LCALL OUT2H
        LCALL OUT2S
        MOV R0,#REG1        ;R1
        LCALL OUT2H         ;SEND DATA
        LCALL OUT4S
        MOV R0,#TO
        LCALL OUT2H         ;SEND ADDRESS TTTT
        MOV R0,#{TO+1}
        LCALL OUT2H
        LCALL OUT2S
        MOV R0,#REG2        ;R2
        LCALL OUT2H         ;SEND DATA
        LJMP VER40
VER20:  LCALL CRLF          ;IF NOT,DISPLAY BOTH DATA
        MOV R0,#FIRST       ;SEND ADDRESS SSSS
        LCALL OUT2H
        MOV R0,#{FIRST+1}
        LCALL OUT2H         ;SEND DATA
        LCALL OUT2S
        MOV R0,#REG1        ;R1
        LCALL OUT2H         ;SEND DATA
        LCALL OUT4S
        MOV R0,#TO
        LCALL OUT2H         ;SEND ADDRESS TTTT
        MOV R0,#{TO+1}
        LCALL OUT2H
        LCALL OUT2S
        MOV DPTR,#DDOT      ;DISPLAY ".." WHEN DATA EQUAL
        LCALL PDATA         ;SEND DATA
VER40:  MOV A,FIRST         ;COMPARE SSSS=TTTT
        CJNE A,LAST,VER30
        MOV A,{FIRST+1}
        CJNE A,{LAST+1},VER30
        LJMP WARM
VER30:  MOV R0,#{FIRST+1}   ;SSSS = SSSS + 1
        LCALL INC16
        MOV R0,#{TO+1}      ;TTTT = TTTT + 1
        LCALL INC16
        LJMP VER10
```

```
;
;*************************************************************
;
ALTER:    LCALL BADDR          ;GET ADDRESS TO EDIT
          JNC ALT05
          JNB F0,ALTEND        ;ERROR
          CLR {ACC OR 5}       ;CONVERT LOWER TO UPPER CASE
          CJNE A,#"I",ALTEND   ;ERROR
          LJMP SUBSTUT
;
ALT05:    MOV FIRST,HIBYTE     ;SAVE ADDRESS TO BE EDIT
          MOV {FIRST+1},LOBYTE
ALT10:    LCALL CRLF
          MOV R0,$FIRST        ;SHOW ADDRESS
          LCALL OUT2H
          INC R0
          LCALL OUT2H          ;SEND 2 SPACES
ALT15:    LCALL OUTS
          MOV DPH,FIRST        ;SET POINTER TO POINT
          MOV DPL,{FIRST+1}
          CLR A
          MOVC A,@A+DPTR       ;GET DATA
          MOV R1,A
          MOV R0,$REG1         ;SHOW DATA
          LCALL OUT2H
          MOV A,#"-"           ;SHOW "-"
          LCALL OUTCH
          LCALL BYTES          ;GET NEW DATA IF EDIT
          JNC ALT20
          XRL A,#BACKSP        ;IF BACKSPACE
          JZ BACKUP
          XRL A,$BACKSP        ;RESTORE A
          XRL A,$DOT
          JZ BACKUP
ALTEND:   LJMP WARM            ;ERROR
;
ALT20:    JB F0,ALT30          ;<CR> OR SPACE
          MOV A,LOBYTE
          MOVX @DPTR,A         ;CHANGE DATA
ALT30:    MOV R0,#{FIRST+1}
          LCALL INC16          ;NEXT BYTE
          MOV A,B
          XRL A,#CR            ;IF "CR"
          JZ ALT10
          MOV A,{FIRST+1}
          ANL A,#07H
          JZ ALT10
          LJMP ALT15
;
BACKUP:   MOV R0,#{FIRST+1}    ;IF BACK SPACE
          LCALL DEC16          ;DECREMENT TO LAST ADDRESS
          LJMP ALT10
;
;*************************************************************
;
```

```
MODIFY:   LCALL BADDR
          MOV DPH,HIBYTE
          MOV DPL,LOBYTE
MOD10:    LCALL INCH
          CJNE A,#BACKSP,NOTBS
          DEC DPL
          MOV A,#0FFH
          CJNE A,DPL,MOD10
          DEC DPH
          LJMP MOD10
;
NOTBS:    CJNE A,#EOT,$+6
          LJMP WARM
          MOVX @DPTR,A
          INC DPTR
          LJMP MOD10
;
;*************************************************************
;
SUBSTUT:
          LCALL BADDR          ;GET DATA
          MOV R0,LOBYTE
IF010:    MOV A,#7FH           ;IF ADDRESS >= 80H OR <= F0H
          CLR C
          SUBB A,R0
          JNC ELS010
          MOV A,#0F0H
          CLR C
          SUBB A,R0
          JC ELS010
THN010:   MOV DPTR,#(SFRTAB-TABENT)
SUB60:    MOV A,#TABENT        ;INCREMENT TABLE POINTER
          ADD A,DPL
          MOV DPL,A
          JNC $+4
          INC DPH
          MOV A,#0FF1          ;FIND 1ST ENTRY >= R0
          MOVC A,@A+DPTR
          CLR C
          SUBB A,R0
          JC SUB60             ;NOT FOUND YET
SUB65:    MOV A,#0FF1          ;INCREMENT R0 TO LINE IN TABLE
          MOVC A,@A+DPTR
          XRL A,R0
          JZ SUB70             ;FOUND IT
          INC R0
          LJMP SUB65
;
ELS010:   LJMP LS010           ;RELAY STATION
;
SUB70:    LCALL CRLF           ;DISPLAY ADDRESS
          LCALL OUTR03S
          LCALL PDATA          ;ALTERS DPTR (ADDS 4)
          MOV A,#":"
          LCALL OUTCH
```

```
                MOV REG1,R0             ;MOV R1,R0
                MOV A,#LOW(SUB80)
                PUSH ACC
                MOV A,#HIGH(SUB80)
                PUSH ACC
                MOV A,#(OFF2-4)
                JMP @A+DPTR             ;CRASHES R0
;
SUB80:          LCALL OUTR0             ;DISPLAY DATA
                MOV A,#"-"
                LCALL OUTCH
                MOV R0,REG1             ;RESTORE R0
                LCALL BYTES
IF020:          JNC ELS020              ;IF CHARACTER = DOT OR BACKSPACE
                MOV A,B
                XRL A,#BACKSP
                JZ THN020
                MOV A,B
                XRL A,#DOT
                JZ THN020
                LJMP WARM               ;ERROR
;
THN020:                                 ;THEN BACKUP
IF030:          CJNE R0,#80H,ELS030
THN030:         MOV R0,#7FH
                LJMP IF010
;
ELS030:         MOV A,DPL
                CLR C
                SUBB A,#(TABENT+4)
                MOV DPL,A
                JNC $+4
                DEC DPH
                MOV A,#(OFF1)
                MOVC A,@A+DPTR
                MOV R0,A
                LJMP IF010
ELS020:
                JB F0,SUB90
                SETB F0
                MOV R0,LOBYTE
                MOV A,#LOW(SUB90)
                PUSH ACC
                MOV A,#HIGH(SUB90)
                PUSH ACC
                MOV A,#(OFF3-4)
                JMP @A+DPTR             ;CRASHES R0
;
SUB90:          MOV R0,REG1             ;RESTORE R0
IF035:          JB F0,NDI035
THN035:         MOV DPTR,#MNONO
                LCALL PDATA
NDI035:
IF040:          CJNE R0,#0F0H,NDI040            ;IF R0 = 0F
THN040:         MOV R0,#0FFH            ;THEN R0 = FFH
```

```
NDI040:
          INC R0
          LJMP IF010
LS010:    LCALL CRLF
          LCALL OUTR03S
SUB20:    LCALL OUT2H
          MOV A,#"-"
          LCALL OUTCH
          MOV REG1,R0
          LCALL BYTES          ;CLOBBERS R0
          MOV R0,REG1
IF050:    JNC ELS050           ;IF CHAR. = BACKSP. OR DOT
          MOV A,B
          XRL A,#BACKSP
          JZ THN050
          MOV A,B
          XRL A,#DOT
          JZ THN050
          LJMP WARM
;
THN050:                        ;THEN BACKUP
IF060:    MOV A,R0
          JNZ NDI060           ;IF R0 = 0
THN060:   MOV R0,#0F1H         ;THEN R0 = F1H
NDI060:
NDI050:
          DEC R0
F010:     LJMP IF010           ;ALSO USED FOR RELAY
;
ELS050:
IF070:    JB F0,NDI070         ;IF NOT SPACE OR CR
THN070:   MOV A,LOBYTE         ;THEN CHANGE BYTE
          MOV @R0,A
NDI070:   INC R0
          MOV A,#07H
          ANL A,R0
          JZ F010
          LCALL OUT2S
          LJMP SUB20
;*************************************************************
;
INSERT:   LCALL THRADR         ;GET ADDRESS
          MOV DPH,FIRST        ;SAVE
          MOV DPL,{FIRST+1}
INS10:    MOV A,{TO+1}         ;POINT TO TARGET ADDRESS
          MOVX @DPTR,A         ;PUT DATA
          MOV A,DPH
          CJNE A,LAST,INS20
          MOV A,DPL
          CJNE A,{LAST+1},INS20
          LJMP WARM
;
INS20:    INC DPTR             ;NEXT
          LJMP INS10
;
```

```
;*************************************************************
;
BREAK:                          ;  REMOVE OLD BREAKPOINT
          MOV DPTR,#BYTENUM     ;  AND PUT OPCODE BACK
          CLR A                 ;  IF NEW BREAK POINT
          MOVC A,@A+DPTR
          MOV R1,A              ;SAVE OLD BYTENUM IN R1
IF100:    JZ NDI100             ;IF THERE IS AN OLD BREAKPOINT
THN100:   MOV FIRST,#HIGH{BYTENUM+1}    ;THEN REMOVE IT
          MOV FIRST+1,#LOW{BYTENUM+1}
          MOV LAST,#HIGH{BYTENUM}
          MOV A,#LOW{BYTENUM}
          ADD A,R1              ;ADD OLD BYTENUM
          MOV LAST+1,A
          JNC $+4
          INC LAST
          MOV DPTR,#{RAMEND-1}
          CLR A
          MOVC A,@A+DPTR
          MOV TO,A
          MOV DPTR,#RAMEND
          CLR A
          MOVC A,@A+DPTR
          CLR C
          SUBB A,R1             ;SUBTRACT OLD BYTENUM
          MOV TO+1,A
          JNC $+4
          DEC TO
          LCALL CPY
          CLR A                 ;CLEAR END OF RAM
          MOV DPTR,#BYTENUM
          MOV R0,#{RAMEND-BYTENUM+1-3}     ;LEAVE JUMP INSTRUCTION
CLEAR1:   MOVX @DPTR,A
          INC DPTR
          DJNZ R0,CLEAR1
NDI100:
          LCALL BYTES           ;INSTALL NEW BREAKPOINT
IF150:    JNC NDI150            ;IF NOT VALID HEX
          LJMP WARM             ;THEN END
;
NDI150:
IF155:    JNB F0,NDI155         ;IF CR
          XRL A,#SPACE
          JZ NDI100             ;AND NOT SPACE
          LJMP WARM             ;THEN END
;
NDI155:   MOV FIRST,HIBYTE      ;GET NEW ADDRESS OF BREAK POINT
          MOV FIRST+1,LOBYTE
          MOV DPTR,#BYTENUM
          MOV A,#3              ;IN CASE OF DEFAULT TO REMOVE
          MOVX @DPTR,A          ;THREE BYTES HEX-CODE
          MOV R1,A              ;SAVE DEFAULT NEW BYTENUM
IF160:    MOV A,B
          XRL A,#CR
          JZ NDI160
```

```
THN160:   LCALL INHEX          ;GET BYTENUM
IF170:    JNC ELS170           ;IF NOT VALID HEX
THN170:   MOV A,B              ;THEN CHECK FOR SPACE OR CR
          XRL A,#SPACE
          JZ THN160
          MOV A,B
          XRL A,#CR
          JZ NDI170
          MOV DPTR,#BYTENUM
          CLR A
          MOVX @DPTR,A
          LJMP WARM
;
ELS170:   MOV DPTR,#BYTENUM    ;ELSE ACCEPT ONLY 3, 4 or 5
          MOVX @DPTR,A
          MOV R1,A             ;SAVE NEW BYTENUM
          SUBB A,#3
          JC BAD4
          SUBB A,#{6-3}
          JNC BAD4
          LJMP NDI170
;
BAD4:     MOV DPTR,#BYTENUM    ;CLEAR STATUS OF BREAK POINT
          CLR A
          MOVX @DPTR,A
          LJMP WARM            ;THEN JUMP TO PROMPT
;
NDI170:                        ;FIRST,FIRST+1 AND HIBYTE,LOBYTE
NDI160:                        ;  BOTH CONTAIN BREAK ADDRESS
                               ;R1 CONTAINS NEW BYTENUM
          MOV DPTR,#{RAMEND-2}
          MOV A,#02H           ;"LJMP"
          MOVX @DPTR,A
          MOV A,FIRST+1
          ADD A,R1             ;ADD BYTENUM
          MOV LAST+1,A
          MOV DPTR,#RAMEND
          MOVX @DPTR,A         ;PUT NEW BREAK POINT ADDRESS
          MOV A,FIRST
          JNC $+3
          INC A
          MOV LAST,A
          MOV DPTR,#{RAMEND-1}
          MOVX @DPTR,A
          MOV R0,#{LAST+1}
          LCALL DEC16          ;ADJUST (LAST,LAST+1)
          MOV TO,#HIGH{BYTENUM+1}
          MOV TO+1,#LOW{BYTENUM+1}
          LCALL CPY            ;SAVE INSTRUCTIONS IN END OF RAM
          MOV DPL,LOBYTE
          MOV DPH,HIBYTE       ;INSERT JUMP TO BRKPT
          MOV A,#02H           ;"LJMP"
          MOVX @DPTR,A
          INC DPTR
          MOV A,#HIGH{BRKPT}   ;ADDRESS OFF BREAK POINT ROUTINE
```

```
          MOVX @DPTR,A
          INC DPTR
          MOV A,#LOW(BRKPT)
          MOVX @DPTR,A
          LJMP WARM
;
;*************************************************************
;
BRKPT:    PUSH ACC              ;SAVE CONTENT OF REGISTERS
          PUSH PSW
          PUSH B
          PUSH DPH
          PUSH DPL
          MOV STACK,SP          ;SAVE CURRENT STACK LEVEL FOR COOL START
          SETB RS0              ;SELECT REGISTER BANK 1
          CLR RS1
          MOV DPTR,#MBRK1       ;SEND ADD.OF BRK-PNT TO TERMINAL
          LCALL PDATA
          MOV DPTR,#(BYTENUM)
          CLR A
          MOVC A,@A+DPTR
          MOV R0,A
          MOV DPTR,#(RAMEND) ;ADDRESS OF BRK-PNT.
          CLR A
          MOVC A,@A+DPTR
          CLR C
          SUBB A,R0
          MOV B,A
          MOV DPTR,#(RAMEND-1)
          CLR A
          MOVC A,@A+DPTR
          MOV R0,A
          JNC $+3
          DEC R0
          LCALL OUTR0
          MOV R0,B
          LCALL OUTR0
          MOV DPTR,#MBRK2       ;SEND 'ACC =' TO TERMINAL
          LCALL PDATA
          MOV A,STACK
          CLR C
          SUBB A,#4
          MOV R0,A
          LCALL OUT2H
          MOV DPTR,#MBRK3       ;SEND 'PSW =' TO TERMINAL
          LCALL PDATA
          INC R0
          LCALL OUT2H
          MOV DPTR,#MBRK4       ;SEND 'B =' TO TERMINAL
          LCALL PDATA
          INC R0
          LCALL OUT2H
          MOV DPTR,#MBRK5       ;SEND 'DPTR =' TO TERMINAL
          LCALL PDATA
          INC R0
```

```
            LCALL OUT2H
            INC R0
            LCALL OUT2H
            MOV DPTR,#MBRK6     ;SEND 'SP =' TO TERMINAL
            LCALL PDATA
            MOV R0,#STACK
            LCALL OUT2H
            LJMP WARM
;
;************************************************************
;
GO:         LCALL BYTES         ;GET ADDRESS TO EXECUTE
            JNC $+5
            LJMP WARM
;
            JNB F0,GXXXX
            CJNE A,#CR,GO       ;LOOK IF "CR" FOLLOW COMMAND "G"
            MOV HIBYTE,#HIGH(BYTENUM+1)   ;EXECUTE "G" AFTER BREAK POINT
            MOV LOBYTE,#LOW(BYTENUM+1)    ;RECALL ALL STATUS BACK
            MOV SP,STACK        ;RESTORE STACK IF NECESSARY
            POP DPL
            POP DPH
            POP B
            POP PSW
            POP ACC
GXXXX:      PUSH LOBYTE         ;EXECUTE
            PUSH HIBYTE
            RET
;
;************************************************************
;
HEXMATH:    LCALL BADDR         ;GET FIRST DATA
            JNC $+5
            LJMP WARM
            MOV FIRST,HIBYTE    ;SAVE DATA
            MOV (FIRST+1),LOBYTE
            LCALL BADDR         ;GET SECOND DATA
            JNC $+5
            LJMP WARM
;
            MOV A,(FIRST+1)
            ADD A,LOBYTE
            MOV (LAST+1),A      ;STORE LOW BYTE OF SUM
            MOV A,FIRST
            ADDC A,HIBYTE
            MOV LAST,A          ;STORE HIGH BYTE OF SUM
;
            MOV A,(FIRST+1)
            CLR C
            SUBB A,LOBYTE
            MOV (TO+1),A        ;STORE LOW BYTE OF DIFFERENCE
            MOV A,FIRST
            SUBB A,HIBYTE
            MOV TO,A            ;STORE HIGH BYTE OF DIFFERENCE
            LCALL CRLF
```

```
        MOV R0,#FIRST       ;POINT TO ADDEND
        LCALL OUT4HS
        MOV A,#"+"
        LCALL OUTCH
        LCALL OUTS
        MOV R0,#HIBYTE      ;POINT TO AUGEND
        LCALL OUT4HS
        MOV A,#"="
        LCALL OUTCH
        LCALL OUTS
        MOV R0,#LAST        ;POINT TO SUM
        LCALL OUT4HS
        LCALL CRLF
        MOV R0,#FIRST       ;POINT TO SUBTRAHEND
        LCALL OUT4HS
        MOV A,#"-"
        LCALL OUTCH
        LCALL OUTS
        MOV R0,#HIBYTE      ;POINT TO MINUEND
        LCALL OUT4HS
        MOV A,#"="
        LCALL OUTCH
        LCALL OUTS
        MOV R0,#TO          ;POINT TO DIFFERENCE
        LCALL OUT4HS
        LJMP WARM
;
;*************************************************************
;
JUMPTABL: MOV DPTR,#MJUMP   ;POINTER POINT TO BEGIN OF TABLE
        LCALL PDATA         ;SEND DATA
        LJMP WARM
;
;*************************************************************
;
UNASS:  LCALL BADDR         ;GET DATA
        MOV FIRST,HIBYTE    ;START ADDRESS FOR DISASSEMBLER
        MOV {FIRST+1},LOBYTE
        MOV LAST,HIBYTE     ;SET DEFAULT TO END AT START+10H
        MOV {LAST+1},{LOBYTE+10H}
        CJNE A,#CR,UNASS1   ;DEFAULT IF NO END ADDRESS ENTERED
        LJMP DISASS
UNASS1: LCALL BADDR2        ;GET END ADDRESS FOR DISASSEMBLER
        MOV LAST,HIBYTE
        MOV{LAST+1},LOBYTE
DISASS: LCALL CRLF          ;
        MOV R0,#FIRST       ;SEND ADDRESS OF THIS INSTRUCTION
        LCALL OUT2H         ;SEND HIBYTE ADD
        MOV R0,#{FIRST+1}
        LCALL OUT2H         ;SEND LOBYTE ADD
        LCALL OUT4S         ;
        MOV DPH,FIRST       ;GET ADDRESS FOR TAKE HEX OPCODE
        MOV DPL,{FIRST+1}
        CLR A               ;CLEAR A
        MOVC A,@A+DPTR      ;GET HEX OPCODE
```

```
            MOV R0,#00          ;LOOK TABLE
            MOV R1,#20H
            MOV R2,A
            ANL A,#0FH          ;ADD MAP = A*10H
            SWAP A
            ADD A,R0
            MOV DPL,A           ;GET CHR OPCODE
            MOV A,R2
            ANL A,#0F0H
            SWAP A
            ADD A,R1
            MOV DPH,A
DISASS2:    CLR A
            MOVC A,@A+DPTR      ;GET DATA
            CJNE A,#04H,DISASS4 ;END OF INSTRUCTION?
            CLR A               ;COMPARE SSSS WITH FFFF
            MOV A,FIRST
            SUBB A,LAST
            JC DISASS6          ;JUMP IF SS<FF
            CLR A
            MOV A,{FIRST+1}
            SUBB A,{LAST+1}
            JC DISASS6          ;JUMP IF SS<FF
            LJMP WARM
DISASS6:    CLR A
            MOV R0,#{FIRST+1}   ;INC ADDRESS
            LCALL INC16
            LJMP DISASS
DISASS4:    CJNE A,#"Q",DISASS5 ;HAD OPERAND TO GET?
            PUSH DPL            ;YES,SAVE POINTER
            PUSH DPH
            MOV DPH,FIRST       ;GET HEX OPR ADD IN RAM
            MOV DPL,{FIRST+1}   ;THIS BYTE IS OPCODE
            INC DPTR
            MOV FIRST,DPH
            MOV {FIRST+1},DPL
            CLR A
            MOVC A,@A+DPTR
            MOV B,A             ;TEMP STORE
            LCALL OUTHL
            MOV A,B
            LCALL OUTHR
            POP DPH             ;GET POINTER BACK
            POP DPL
            INC DPTR            ;NEXT ASCII IN TABLE
            LJMP DISASS2        ;GET NEXT CHR OF OPCODE
DISASS5:    LCALL OUTCH         ;NO,IT ONLY ONE CHARACTER
            INC DPTR
            LJMP DISASS2
;
;************************************************************
;
SEARCH:     MOV DPTR,#DATCNT    ;POINT TO BYTE COUNT
            CLR A
            MOVX @DPTR,A
```

```
          LCALL BADDR          ;GET START ADDRESS
          MOV FIRST,HIBYTE     ;SAVE START ADDRESS
          MOV {FIRST+1},LOBYTE
          LCALL BADDR          ;GET FINISH ADDRESS
          MOV LAST,HIBYTE      ;SAVE FINISH ADDRESS
          MOV {LAST+1},LOBYTE
SEARCH2:  LCALL BYTES          ;GET FIRST DATA
          JC SERR1             ;IF NOT HEX,ERROR
          CLR A
          MOV DPTR,#DATCNT     ;GET OLD BYTE COUNT
          MOVC A,@A+DPTR
          INC A                ;INCREMENT BYTE COUNT
          MOVX @DPTR,A
          DEC A
          MOV DPTR,#DATBUF     ;ADD OFFSET
          ADD A,DPL
          MOV DPL,A
          MOV A,LOBYTE         ;SAVE DATA FOR SEARCH
          MOVX @DPTR,A
          MOV A,B
          CJNE  A,#SPACE,SEARCH1
          SJMP SEARCH2              ;GET NEXT DATA
SERR1:    LJMP COOL
SEARCH1:  MOV DPH,FIRST        ;BEGIN TO SEARCH ADDRESS
          MOV DPL,{FIRST+1}
          CLR A
          MOVC A,@A+DPTR
          MOV R1,A             ;SAVE DATA IN R1
          MOV DPTR,#DATBUF     ;GET DATA
          CLR A
          MOVC A,@A+DPTR
          XRL A,R1
          JZ SEARCH3           ;JUMP IF DATA EQUAL
          CLR A                ;COMPARE SSSS WITH FFFF
          MOV A,FIRST
          SUBB A,LAST
          JC SEARCH4           ;JUMP IF SS<FF
          CLR A
          MOV A,{FIRST+1}
          SUBB A,{LAST+1}
          JC SEARCH4           ;JUMP IF SS<FF
          LJMP SEARCH5
SEARCH4:  MOV R0,#{FIRST+1}    ;SSSS = SSSS + 1
          LCALL INC16
          LJMP SEARCH1
SEARCH5:  MOV DPTR,#NOTFND     ;DISPLAY SEARCH NOT FOUND
          LCALL PDATA
          LJMP COOL
;
;IF FIND DATA EQUAL ONE BYTE
;
SEARCH3:  LCALL NXTSCH         ;CHK ALL DATA
          JC SEARCH1           ;ALL DATA NOT EQUAL
          MOV R0,#{FIRST+1}
          LCALL DEC16          ;DEC POINTER 1 STEP
```

```
          MOV DPTR,#FND        ;FOUND,DISPLAY TO TERMINAL
          LCALL PDATA
SEARCH8:  LCALL CRLF           ;IF NOT,DISPLAY BOTH DATA
          MOV R0,#TO           ;SEND ADDRESS SSSS
          LCALL OUT2H
          MOV R0,#{TO+1}
          LCALL OUT2H          ;SEND DATA
          LCALL OUT2S
          MOV DPH,TO
          MOV DPL,{TO+1}
          CLR A
          MOVC A,@A+DPTR
          MOV R0,A
          LCALL OUTC2HS        ;SEND DATA
          LCALL OUT4S
          MOV A,FIRST          ;COMPARE SSSS=TTTT
          CJNE A,TO,SEARCH7
          MOV A,{FIRST+1}
          CJNE A,{TO+1},SEARCH7
          LJMP COOL
SEARCH7:  MOV R0,#{TO+1}       ;SSSS = SSSS + 1
          LCALL INC16
          LJMP SEARCH8
;
;*************************************************************
;
NXTSCH:   MOV TO,FIRST         ;SAVE FIRST ADD IF FOUND
          MOV {TO+1},{FIRST+1}
          CLR A
          MOV R1,A             ;SET COUNTER
NXTSCH2:  MOV DPTR,#0FFF0H     ;FIRST ADD OF DATA TO SEARCH
          MOV A,R1             ;GET OFFSET
          ADD A,DPL
          MOV DPL,A
          CLR A
          MOVC A,@A+DPTR       ;GET DATA
          MOV R2,A
          MOV DPH,FIRST        ;GET DATA FROM MEM
          MOV DPL,{FIRST+1}
          CLR A
          MOVC A,@A+DPTR
          XRL A,R2
          JNZ NXTSCH1          ;NOT EQUAL
          MOV R0,#{FIRST+1}    ;EQUAL THEN CHK NEXT DATA
          LCALL INC16
          INC R1
          MOV DPTR,#0FFF7H     ;GET BYTE COUNT
          CLR A
          MOVC A,@A+DPTR
          XRL A,R1
          JNZ NXTSCH2          ;COMPARE NUMBER OF BYTES
          CLR C                ;ALL EQUAL
          RET
NXTSCH1:  SETB C               ;ALL NOT EQUAL
          RET
```

```
;
;************************************************************
;
MOVE:     MOV A,#0FFH          ;RESET COUNTER
          MOV DPTR,#6002H      ;PORT C
          MOVX @DPTR,A
          MOV A,#0FDH
          MOVX @DPTR,A
          MOV DPTR,#0000H      ;GET TOTAL FCF BYTES
          MOV FIRST,#0F0H      ;ADD OF DATA STORED F000H
          MOV {FIRST+1},#00H
INDAT1:   PUSH DPH             ;SAVE BYTE COUNT
          PUSH DPL
          LCALL PULSE          ;GET DATA FROM CACHE RAM
          MOV DPH,FIRST
          MOV DPL,{FIRST+1}
          MOVX @DPTR,A         ;SAVE DATA
          MOV R0,#{FIRST+1}
          LCALL INC16          ;INC TO NEXT ADD
          POP DPL
          POP DPH
          INC DPTR
          CLR A
          MOV A,DPH            ;COMPARE BYTE COUNT
          XRL A,#0FH
          JNZ INDAT1
          MOV A,DPL
          XRL A,#0DFH
          JNZ INDAT1
          MOV DPTR,#MOVEPAT    ;DISPLAY TO TERMINAL
          LCALL PDATA
          LJMP WARM
;
;************************************************************
;
;OUT ONE PULSE AT PC0,HLH
;USE R0 R1 R3 AND DPTR
;
PULSE:    MOV R0,#020H         ;DELAY INNER LOOP
          MOV DPTR,#6002H      ;POINT TO PORT C
          MOV A,#0FCH          ;SET BIT 0 TO LOW
          MOVX @DPTR,A         ;OUT
PULSE1:   DJNZ R0,PULSE1
          MOV DPTR,#6000H      ;POINT TO PORT A
          MOVX A,@DPTR
          MOV R3,A
          MOV A,#0FDH          ;SET BIT 0 TO HI
          INC DPTR
          INC DPTR
          MOVX @DPTR,A
          MOV R0,#20H
PULSE2:   DJNZ R0,PULSE2       ;DELAY AGAIN
          MOV A,R3             ;GET DATA BACK
          RET
;
```

```
;*************************************************************
;
PREPRO:   MOV FIRST,#0F0H      ;SAVE DATA AFTER PROCESS
          MOV {FIRST+1},#00H
          MOV LAST,#0FFH       ;END OF RAW DATA
          MOV {LAST+1},#0DFH
          MOV TO,#0F0H         ;RAW DATA POINTER
          MOV {TO+1},#00H
PREPRO7:  MOV DPH,TO
          MOV DPL,{TO+1}
          CLR A
          MOVC A,@A+DPTR
          MOV DPTR,#3000H      ;POINT TO TABLE
          ADD A,DPL
          MOV DPL,A
          CLR A
          MOVC A,@A+DPTR       ;GET DATA
          CJNE A,#11H,PREPRO1  ; 1 CYCLE 1 BYTE
          MOV DPH,TO
          MOV DPL,{TO+1}
          CLR A
          MOVC A,@A+DPTR
          MOV DPH,FIRST
          MOV DPL,{FIRST+1}
          MOVX @DPTR,A
          MOV R0,#{FIRST+1}    ;NEXT BYTE
          LCALL INC16
          MOV R0,#{TO+1}       ;STEP ONE OPCODE
          LCALL INC16
          MOV R0,#{TO+1}       ;NEXT BYTE
          LCALL INC16
          LJMP BYCNT
PREPRO1:  CJNE A,#12H,PREPRO2  ;1 CYCLE 2 BYTE
          MOV DPH,TO
          MOV DPL,{TO+1}
          CLR A
          MOVC A,@A+DPTR
          MOV DPH,FIRST
          MOV DPL,{FIRST+1}
          MOVX @DPTR,A
          MOV R0,#{FIRST+1}    ;NEXT BYTE
          LCALL INC16
          MOV R0,#{TO+1}       ;NEXT BYTE
          LCALL INC16
          MOV DPH,TO
          MOV DPL,{TO+1}
          CLR A
          MOVC A,@A+DPTR
          MOV DPH,FIRST
          MOV DPL,{FIRST+1}
          MOVX @DPTR,A
          MOV R0,#{FIRST+1}    ;NEXT BYTE
          LCALL INC16
          MOV R0,#{TO+1}       ;NEXT BYTE
          LCALL INC16
```

```
          LJMP BYCNT
PREPRO2:  CJNE A,#21H,PREPRO3  ; 2 CYCLE 1 BYTE
          MOV DPH,TO
          MOV DPL,{TO+1}
          CLR A
          MOVC A,@A+DPTR
          MOV DPH,FIRST
          MOV DPL,{FIRST+1}
          MOVX @DPTR,A
          MOV R0,#{FIRST+1}    ;NEXT BYTE
          LCALL INC16
          MOV R0,#{TO+1}       ;NEXT BYTE
          LCALL INC16
          MOV R0,#{TO+1}       ;NEXT BYTE
          LCALL INC16
          MOV R0,#{TO+1}       ;NEXT BYTE
          LCALL INC16
          MOV R0,#{TO+1}       ;NEXT BYTE
          LCALL INC16
          LJMP BYCNT
PREPRO3:  CJNE A,#22H,PREPRO4  ;2 CYCLE 2 BYTE
          MOV DPH,TO
          MOV DPL,{TO+1}
          CLR A
          MOVC A,@A+DPTR
          MOV DPH,FIRST
          MOV DPL,{FIRST+1}
          MOVX @DPTR,A
          MOV R0,#{FIRST+1}    ;NEXT BYTE
          LCALL INC16
          MOV R0,#{TO+1}       ;NEXT BYTE
          LCALL INC16
          MOV DPH,TO
          MOV DPL,{TO+1}
          CLR A
          MOVC A,@A+DPTR
          MOV DPH,FIRST
          MOV DPL,{FIRST+1}
          MOVX @DPTR,A
          MOV R0,#{FIRST+1}    ;NEXT BYTE
          LCALL INC16
          MOV R0,#{TO+1}       ;STEP NEXT BYTE
          LCALL INC16
          MOV R0,#{TO+1}       ;STEP NEXT BYTE
          LCALL INC16
          MOV R0,#{TO+1}       ;STEP NEXT BYTE
          LCALL INC16
          LJMP BYCNT
PREPRO4:  CJNE A,#23H,PREPRO5  ;2 CYCLE 3 BYTE
          MOV DPH,TO
          MOV DPL,{TO+1}
          CLR A
          MOVC A,@A+DPTR
          MOV DPH,FIRST
          MOV DPL,{FIRST+1}
```

```
        MOVX @DPTR,A
        MOV R0,#{FIRST+1}    ;NEXT BYTE
        LCALL INC16
        MOV R0,#{TO+1}       ;NEXT BYTE
        LCALL INC16
        MOV DPH,TO
        MOV DPL,{TO+1}
        CLR A
        MOVC A,@A+DPTR
        MOV DPH,FIRST
        MOV DPL,{FIRST+1}
        MOVX @DPTR,A
        MOV R0,#{FIRST+1}    ;NEXT BYTE
        LCALL INC16
        MOV R0,#{TO+1}       ;NEXT BYTE
        LCALL INC16
        MOV DPH,TO
        MOV DPL,{TO+1}
        CLR A
        MOVC A,@A+DPTR
        MOV DPH,FIRST
        MOV DPL,{FIRST+1}
        MOVX @DPTR,A
        MOV R0,#{FIRST+1}    ;NEXT BYTE
        LCALL INC16
        MOV R0,#{TO+1}       ;NEXT BYTE
        LCALL INC16
        MOV R0,#{TO+1}       ;STEP NEXT BYTE
        LCALL INC16
        LJMP BYCNT
PREPRO5: CJNE A,#41H,PREPRO6
        MOV DPH,TO
        MOV DPL,{TO+1}
        CLR A
        MOVC A,@A+DPTR
        MOV DPH,FIRST
        MOV DPL,{FIRST+1}
        MOVX @DPTR,A
        MOV R0,#{FIRST+1}    ;NEXT BYTE
        LCALL INC16
        MOV R0,#{TO+1}       ;NEXT BYTE
        LCALL INC16
        MOV R0,#{TO+1}       ;NEXT BYTE
        LCALL INC16
        MOV R0,#{TO+1}       ;NEXT BYTE
        LCALL INC16
        MOV R0,#{TO+1}       ;NEXT BYTE
        LCALL INC16
        MOV R0,#{TO+1}       ;NEXT BYTE
        LCALL INC16
        MOV R0,#{TO+1}       ;NEXT BYTE
        LCALL INC16
        MOV R0,#{TO+1}       ;NEXT BYTE
```

```
            LCALL INC16
            LJMP BYCNT
PREPRO6:    MOV R0,#{TO+1}
            LCALL INC16
BYCNT:      CLR A
            MOV A,TO
            XRL A,#0FFH
            JNZ BYCNT1
            CLR C
            MOV A,{TO+1}
            SUBB A,#0DFH
            JC BYCNT1
            MOV DPH,FIRST
            MOV DPL,{FIRST+1}
BYCNT2:     CLR A
            MOVX @DPTR,A
            INC DPTR
            MOV A,DPH
            XRL A,#0FFH
            JNZ BYCNT2
            MOV A,DPL
            XRL A,#0DFH
            JNZ BYCNT2
            MOV DPTR,#MVPPAT
            LCALL PDATA
            LJMP WARM
BYCNT1:     LJMP PREPRO7
;
;*************************************************************
;
EXTCLK:     MOV A,#0FFH
            MOV DPTR,#6002H
            MOVX @DPTR,A
            MOV A,#0F8H
            MOVX @DPTR,A
            MOV DPTR,#EXTPAT
            LCALL PDATA
            LJMP WARM
;
;*************************************************************
ANALYS:     LCALL BADDR          ;GET ADDRESS TO EXECUTE
            JC ANALYS2
            CJNE A,#CR,ANALYS2
ANALYS1:    PUSH LOBYTE          ;EXECUTE
            PUSH HIBYTE
            MOV A,#0F8H
            MOV DPTR,#6002H
            MOVX @DPTR,A
            RET
ANALYS2:    LJMP WARM
;
;*************************************************************
;
;ROUTINE PDATA
;WRITES A MESSAGE TO THE TERMINAL
```

```
;ENTER WITH DPTR POINTING TO BEGINNING OF MESSAGE
;AND 04H AT END OF MESSAGE.
;
PDATA1:   LCALL OUTCH          ;WRITE CHARACTER TO TERMINAL
          INC DPTR             ;NEXT CHARACTER
PDATA:    CLR A
          MOVC A,@A+DPTR
          CJNE A,#EOT,PDATA1   ;FIND END OF TEXT.
          RET
;
;*************************************************************
;
;ROUTINE IN
;READ AN 8-BIT CHAR FROM 8251A UART
;
IN:       PUSH DPH             ;SAVE DPTR
          PUSH DPL
          MOV DPTR,#UARTCONT   ;GET UART CONTROL WORD
          MOVX A,@DPTR
          JNB (ACC OR 1),$-1   ;WAIT FOR RX RDY
          DEC DPL              ;POINT TO DATA REGISTER
          MOVX A,@DPTR         ;GET CHARACTER
          POP DPL              ;GET OLD DPTR
          POP DPH
          RET
;
;*************************************************************
;
;ROUTINE INCH
;READ A CHARACTER, ZERO HIGH BIT, & ECHO BACK TO TERMINAL
;
INCH:     LCALL IN             ;READ 8-BIT CHAR.
          ANL A,#7FH           ;ZERO HIGH BIT
          LJMP OUT             ;ECHO BACK TO TERMINAL
;
;*************************************************************
;
;ROUTINE OUTCH
;OUTPUT A CHARACTER
;
OUTCH:    LCALL OUT            ;OUT ONE CHARACTER TO UART
          JNC NOINPU           ;TEST FOR INPUT DURING OUTPUT
          LCALL IN             ;GET RID OF THE CHARACTER
          LJMP COOL            ;RESET STACK POINTER
;
NOINPU:   RET
;
;*************************************************************
;
;ROUTINE OUT
;SEND "A" REGISTER TO UART
;SET CARRY IF RX BUFFER IS FULL
;
OUT:      PUSH DPH             ;SAVE DPTR
          PUSH DPL
```

```
            PUSH ACC              ;SAVE OUTPUT BYTE
            MOV DPTR,#UARTCONT    ;GET UART CONTROL WORD
            MOVX A,@DPTR
            JNB {ACC OR 0},$-1    ;WAIT FOR TX EMPTY
            MOV C,{ACC OR 1}      ;MOV RX RDY TO CARRY
            DEC DPL               ;POINT TO DATA REGISTER
            POP ACC               ;RESTORE OUTPUT BYTE
            MOVX @DPTR,A          ;AND OUTPUT IT
            POP DPL               ;GET OLD DPTR
            POP DPH
            RET
;
;
;*************************************************************
;
;ROUTINE INHEX
;READS 1 ASCII HEX CHAR, CONVERTS TO BINARY IN ACC. &
;STORES ORIGINAL CHAR IN B REG.
;CARRY SET IF NOT VALID HEX, CLEARED OTHERWISE.
;
INHEX:      LCALL INCH            ;GET CHARACTER
            MOV B,A               ;STORE IN B
            CJNE A,#CR,NOTCR      ;IF <CR> SEND <LF> ALSO
            MOV A,#LF
            LCALL OUTCH           ;SEND <LF>
NOTCR:      MOV A,B               ;RESTORE CHARACTER
            JNB {ACC OR 6},$+5    ;IF LOWER CASE
            CLR {ACC OR 5}        ;CONVERT TO UPPER CASE
            CLR C
            SUBB A,#"0"           ;TESTING
            JC BAD                ; ACC < '0'
            SUBB A,#10
            JNC $+7               ; ACC > '9'
            ADD A,#10             ;RESTORE
            LJMP GOOD             ; '0' <= ACC <= '9'
;
            ADD A,#"0"+10-"A"+10      ;CORRECT FOR (-'0'-10) & MAP 'A' INTO 10
            JB {ACC OR 7},BAD         ; '9' < ACC < 'A'
            CLR C
            SUBB A,#16
            JNC BAD               ; ACC > 'F'
            ADD A,#16
GOOD:       CLR C
            RET
BAD:        SETB C                ;SET BIT FOR INDICATER
            RET
;
;*************************************************************
;
;ROUTINE BYTES
;READ IN TWO BYTE HEX NUMBER TO HIBYTE,LOBYTE
;LAST CHARACTER READ RETURNED IN B REGISTER
;
;ERROR CODES:        CARRY: F0:
;LEADING <CR> OR SPACE  0 1 (CHAR IN ACC)
```

```
;LEADING NONHEX CHARACTER 1 1 (CHAR IN ACC)
;OTHER NONHEX CHARACTER  1 0 (CHAR IN ACC)
;
BYTES:    SETB F0              ;NO HEX READ YET
          CLR A
          MOV HIBYTE,A         ;CLEAR OLD DATA
          MOV LOBYTE,A
MORE1:    LCALL INHEX          ;READ NEW DATA
          JNC OK1              ;JUMP IF HEX DIGIT
          MOV A,B              ;LOOK AT ASCII
          CJNE A,#CR,$+6
          LJMP CR1
;
          CJNE A,#SPACE,BAD1
CR1:      CLR C
          RET
;
OK1:      CLR F0
          MOV R0,#LOBYTE       ;POINTER
          XCHD A,@R0           ;PUT 0 DIGIT IN LOW END OF LOBYTE
          SWAP A               ;ACC NOW HAS DIGIT 1 IN HIGH END
          XCHD A,@R0           ;ACC NOW HAS DIGITS 1,0
          XCH A,@R0            ;ACC HAS DIGITS 2,X;LOBYTE DONE.
          DEC R0               ;POINT TO HIBYTE
          XCHD A,@R0           ;ACC NOW HAS DIGITS 2,3
          SWAP A               ;ACC HAS DIGITS 3,2
          XCH A,@R0            ;HIBYTE DONE
          LJMP MORE1           ;LOOK FOR ANOTHER DIGIT
;
BAD1:     SETB C
          RET
;
;
;*************************************************************
;
;ROUTINE BUILD ADDRESS
;READ IN A 16-BIT HEX NUMBER TO HIBYTE,LOBYTE.
;RETURNS WITH CARRY & F0 SET IF A SECOND COMMAND CHARACTER
;IS FOUND.  CHARACTER WILL BE IN ACCUMULATOR.
;RETURNS WITH CARRY SET AND F0 CLEAR IF NON-HEX.
;
BADDR:    LCALL BYTES
          JNC ADDROK
          JB F0,ADDRNOK        ;SECOND COMMAND CHARACTER
          LJMP WARM            ;ERROR
ADDROK:   JB F0,BADDR          ;LEADING SPACE OR CR
ADDRNOK:
          RET
;
;*************************************************************
;
;ROUTINE BUILD ADDRESS #2
;READ IN A 16-BIT HEX NUMBER TO HIBYTE,LOBYTE.
;IGNORES LEADING SPACES.  RETURNS WITH F0 SET IF
;A LEADING CARRIAGE RETURN IS FOUND.
```

```
;
BADDR2:   LCALL BYTES
          JNC $+5
          LJMP WARM           ;NON-HEX
;
          JNB F0,ADDROK2      ;NUMBER READ
          CJNE A,#CR,BADDR2   ;IGNORE LEADING SPACE
ADDROK2:
          RET
;
;*************************************************************
;
;ROUTINE OUTPUT SPACES
;OUTPUTS 1, 2, 3, OR 4 SPACES
;
OUT4S:    LCALL OUTS          ;SEND 1 SPACE
OUT3S:    LCALL OUTS
OUT2S:    LCALL OUTS
OUTS:     MOV A,#SPACE        ;POINT TO PATTERN
          LJMP OUTCH
;
;*************************************************************
;
;ROUTINE OUTPUT CODE, TWO HEX, SPACE
;OUTPUTS PROGRAM MEMORY BYTE POINTED OUT BY DPTR
;AS TWO ASCII CHARACTERS FOLLOWED BY A SPACE.
;USES B REGISTER AS TEMP. STORE
;
OUTC2HS:
          CLR A
          MOVC A,@A+DPTR      ;GET HEX DATA
          MOV B,A             ;TEMP STORE
          LCALL OUTHL         ;SEND FIRST DIGIT
          MOV A,B
          LCALL OUTHR         ;NEXT SEND SECOND DIGIT
          LJMP OUTS           ;THEN SEND 1 SPACE
;
;*************************************************************
;
;ROUTINE MIDCHK
;INSERTS A SPACE IF LOW NYBBLE OF DPTR = 8.
;RETURNS WITH LOW NYBBLE IN ACC.
;DESTROYS B REGISTER.
;
MIDCHK:   MOV A,DPL           ;GET NIBBLE OF LO BYTE ADDRESS
          ANL A,#0FH          ;MASK 4 BIT LOW
          XRL A,#08H          ; 08?
          JNZ NOTMID
          XCH A,B             ; YES, OUT ONE SPACE
          LCALL OUTS
          XCH A,B
NOTMID:   XRL A,#08H          ;RESTORE A
          RET
;
;*************************************************************
```

```
;
;ROUTINES OUTPUT HEX LEFT
;OUTPUT HEX RIGHT
;CONVERTS A NYBBLE IN ACC. TO ASCII AND SENDS IT
;
OUTHL:    SWAP A
OUTHR:    ANL A,#0FH
          JNB {ACC OR 3},H2    ;<8
          JB {ACC OR 2},H1     ;>=C
          JNB {ACC OR 1},H2    ;<A
H1:       ADD A,#07H
H2:       ADD A,#"0"           ;CONVERT TO ASCII
          LJMP OUTCH
;
;
;*************************************************************
;
;ROUTINE OUTPUT TWO HEX
;OUTPUTS HEX CONTENTS OF LOCATION POINTED OUT BY R0
;
OUT2H:    MOV A,@R0            ;SET R0 POINT TO DATA
          LCALL OUTHL
          MOV A,@R0
          LJMP OUTHR
;
;*************************************************************
;
;ROUTINE CRLF
;SENDS A CARRIAGE RETURN AND A LINE FEED
;
CRLF:     MOV A,#CR            ;SEND CR
          LCALL OUTCH
          MOV A,#LF            ;SEND LF
          LJMP OUTCH
;
;*************************************************************
;
;ROUTINE DECREMENT 16
;DECREMENTS A 16-BIT NUMBER POINTED OUT BY R0.
;ENTER WITH LOW BYTE @R0 & HIGH BYTE @(R0-1).
;CARRY SET ON OVERFLOW, CLEARED OTHERWISE.
;
DEC16:    CLR C                ;CLEAR CARRY FLAG
          DEC @R0              ;DECREMENT LOW BYTE FIRST
          MOV A,@R0
          CPL A
          JNZ DECEND
          DEC R0               ;NEXT DECREMENT HIGH BYTE
          DEC @R0
          MOV A,@R0
          CPL A
          JNZ DECEND
          SETB C               ;DECREMENTED ALL,SET CARRY FLAG
DECEND:   RET
;
```

```
;
;************************************************************
;
;ROUTINE INCREMENT 16
;INCREMENTS A 16-BIT NUMBER POINTED OUT BY R0.
;ENTER WITH LOW BYTE @R0 & HIGH BYTE @(R0-1).
;CARRY SET ON OVERFLOW, CLEARED OTHERWISE.
;
INC16:    CLR C                ;SET FLAG
          INC @R0              ;INC LBYTE FIRST
          MOV A,@R0
          JNZ INCEND
          DEC R0               ;NEXT INC HBYTE
          INC @R0
          MOV A,@R0
          JNZ INCEND
          SETB C               ;ALL,SET CARRY FLAG
INCEND:   RET
;
;
;************************************************************
;
;ROUTINE THREE ADRESSES
;GETS THREE 16-BIT HEX NUMBERS AND STORES THEM IN
;"FIRST", "LAST", AND "TO" RESPECTIVELY.
;
THRADR:   LCALL BADDR          ;GET 16 BIT ADDRESS
          JC THRERR
          MOV FIRST,HIBYTE     ;SAVE IN "FIRST"
          MOV {FIRST+1},LOBYTE  ;SAVE IN "FIRST + 1"
          LCALL BADDR          ;GET NEXT 16 ADDRESS
          JC THRERR
          MOV LAST,HIBYTE      ;SAVE IN "LAST"
          MOV {LAST+1},LOBYTE   ;SAVE IN "LAST + 1"
          LCALL BADDR          ;GET NEXT 16 ADDRESS
          JC THRERR
          MOV TO,HIBYTE        ;SAVE IN "TO"
          MOV {TO+1},LOBYTE    ;SAVE IN "TO+1"
          RET
THRERR:   LJMP COOL            ;IF ERROR
;
;************************************************************
;
;ROUTINE COPY
;COPIES PROGRAM MEMORY LOCATED "FIRST" TO "LAST"
;TO RAM LOCATION STARTING AT "TO"
;
CPY:      MOV DPH,FIRST        ;GET DPTR
          MOV DPL,{FIRST+1}
          CLR A                ;CLEAR OLD VALUE
          MOVC A,@A+DPTR       ;GET NEW VALUE
          MOV DPH,TO
          MOV DPL,{TO+1}
          MOVX @DPTR,A         ;PUT DATA
          MOV A,FIRST
```

```
            CJNE A,LAST,CPY2
            MOV A,{FIRST+1}
            CJNE A,{LAST+1},CPY2
            RET                     ;DONE
;
CPY2:       MOV R0,#{FIRST+1}       ;INCREMENT TO NEXT ADDRESS
            LCALL INC16
            MOV R0,#{TO+1}
            LCALL INC16
            LJMP CPY                ;COPY AGAIN
;
;*************************************************************
;
;ROUTINE OUT, 2 HEX, 3 SPACES
;OUTPUTS LOCATION POINTED OUT BY R0 FOLLOWED BY 3 SPACES
;
OUT2H3S:
            LCALL OUT2H             ;SEND TWO HEX ASCII
            LJMP OUT3S              ;SEND THREE SPACES
;
;*************************************************************
;
;ROUTINE OUTPUT R0
;OUTPUTS THE CONTENTS OF R0
;(8051 CAN'T ACCESS SFR'S INDIRECTLY)
;
OUTR0:      MOV A,R0
            LCALL OUTHL
            MOV A,R0
            LJMP OUTHR
;
;
;*************************************************************
;
;ROUTINE OUTPUT R0, 3 SPACES
;OUTPUTS THE CONTENTS OF R0 AND THREE SPACES
;
OUTR03S:    LCALL OUTR0
            LJMP OUT3S
;
;*************************************************************
;
;ROUTINE OUTPUT FOUR HEX, SPACE
;
;OUTPUTS TWO CONSECUTIVE INTERNAL MEMORY LOCATIONS,
;THE LOWER OF WHICH (HIGH BYTE OF NUMBER) IS POINTED
;OUT BY R0.
;
OUT4HS:     LCALL OUT2H             ;SEND TWO HEX FIRST
            INC R0
            LCALL OUT2H             ;THEN SEND NEXT TWO FIRST
            LJMP OUTS
;
;
;*************************************************************
```

```
;
;ROUTINE DOWN-LOAD FROM TERMINAL
;
;GET DATA IN "INTEL FORMAT 8 BITS" FROM TERMINAL AND PUT DATA
;TO EXTERNAL RAM.
;
DOWNLOAD:  MOV DPTR,#MDOWNL1  ;SEND MSSG TO TERMINAL.
           LCALL PDATA
           LCALL CRLF
           LCALL CHKCOLON     ;CHK BEGIN OF THIS RECORD
           LCALL BCOUNT       ;GET TOTAL BYTES OF DATA IN THIS RECORD
           LCALL ADDRESS      ;GET BEGIN OF ADDRESS TO PUT DATA
           MOV DPTR,#HIADD    ;SAVE ADDRESS FOR COMPARE AT EOF
           MOV A,R4
           MOVX @DPTR,A        ;SAVE HI ADD
           MOV DPTR,#LOADD
           MOV A,R5
           MOVX @DPTR,A       ;SAVE LO ADD
           PUSH ACC           ;PUSH LO ADD
           DEC DPL
           MOVX A,@DPTR
           PUSH ACC           ;PUSH HI ADD
           POP DPH
           POP DPL
           LCALL SAVEDATA
CONTINUE:  LCALL CHKCOLON     ;CHK COLON
           LCALL BCOUNT       ;CHK BYTECOUNT
           LCALL ADDRESS      ;CHK ADDRESS
           MOV A,R4
           MOV DPH,A
           MOV A,R5
           MOV DPL,A
           LCALL SAVEDATA     ;SAVE DATA
           LJMP CONTINUE
;
;*************************************************************
;
;NOT INTELL FORMAT SHOW ERROR
;FORMAT : FIRST CHARACTER MUST BE ":"
;         NEXT TWO CHARACTERS MUST BE BYTE COUNT
;         NEXT FOUR CHARACTERS MUST BE ADDRESS OF DATA
;         NEXT TWO CHARACTERS MUST BE "00"
;         NEXT OF TWO CHARACTERS MUST BE DATA
;         LAST TWO CHARACTERS MUST BE CHECKSUM ERROR
;
NOTINTEL:  LJMP WARM
;
;
CHKCOLON: LCALL IN            ;GET ":" FROM TERMINAL
          CJNE A,#":",NOTINTEL
          RET
;
BCOUNT:   MOV R6,#00          ;CLEAR BYTE COUNT
          LCALL INASCII2      ;GET ADD FROM TERMINAL
          MOV R6,A            ;SAVE DATA FOR CHKSUM ERROR
```

```
          MOV R7,A            ;SAVE BYTECOUNT IN R7
          SUBB A,#00
          JZ ENDREC           ;LAST RECORD OF FILE
          CLR C               ;LONG FORMAT NOT MORE THAN 20H
          SUBB A,#21H
          JNC NOTINTEL
          NOP
          RET
ENDREC:   LCALL ADDRESS       ;CHECK BEGIN ADDRESS
          MOV A,R4            ;TEST FORMAT OF LAST RECORD
          SUBB A,#00          ;TEST USING START ADDRESS?
          JNZ NOSTADD         ;IF NOT,COMPARE START ADDRESS
          MOV A,R5
          SUBB A,#00
          JZ HEADING
NOSTADD:  MOV DPTR,#HIADD
          MOVX A,@DPTR        ;GET HI BYTE OF START ADDRESS
          SUBB A,R4
          JNZ NOTINTEL
          INC DPTR
          MOVX A,@DPTR        ;GET LO BYTE OF START ADDRESS
          SUBB A,R5
          JNZ NOTINTEL
HEADING:  LCALL INASCII2
          SUBB A,#00H         ;CHECK "00"
          JZ OK2
          SUBB A,#01H         ;CHECK "01"
          JZ OK2
          LJMP NOTINTEL
OK2:      MOV DPTR,#MDOWNL3   ;DOWNLOAD COMPLETE
          LCALL PDATA
          LJMP WARM
;
ADDRESS:  LCALL INASCII2      ;GET HI ADD BYTE
          MOV R4,A
          ADD A,R6            ;FOR CHECK SUM ERROR
          MOV R6,A
          LCALL INASCII2      ;GET LO ADD BYTE
          MOV R5,A
          ADD A,R6            ;FOR CHECK SUM
          MOV R6,A
          RET
;
;*************************************************************
;
;ROUTINE GET ALL DATA IN ONE RECORD WHEN DOWN LOAD
;INPUT     R7 = BYTE COUNT
;        DPTR = POINT TO RAM WHICH DATA IS SAVED
;
SAVEDATA: LCALL INASCII2      ;CHK BGN OF DATA GROUP
          SUBB A,#00H         ;TEST ZERO FLAG
          JNZ NOTINTEL        ;IF NOT ZERO,ERROR
STILLGET: LCALL INASCII2      ;GET DATA BYTE
          MOVX @DPTR,A
          ADD A,R6
```

```
          MOV R6,A              ;CHKSUM PROCESS
          INC DPTR              ;NEXT ADDRESS
          DJNZ R7,STILLGET      ;NEXT BYTE
          LCALL INASCII2        ;CHKSUM DATA OF THIS RECORD
          DEC A                 ;CONVERT 2'COMPLEMENT FORM
          CPL A
          SUBB A,R6             ;COMPARE CHKSUM
          JNZ NOTINTEL          ;IF NOT ZERO,ERROR
          LCALL IN              ;GET "CR"
          LCALL IN              ;GET "LF"
          RET
;
;*************************************************************
;
;ROUTINE INASCII
;READS 1 ASCII HEX CHAR, CONVERTS TO BINARY IN ACC. &
;STORES ORIGINAL CHAR IN B REG.
;CARRY SET IF NOT VALID HEX, CLEARED OTHERWISE.
;
INASCII1: LCALL IN              ;GET CHARACTER
          JNB {ACC OR 6},UPCASE   ;IF LOWER CASE
          CLR {ACC OR 5}        ;CONVERT TO UPPER CASE
UPCASE:   CLR C
          SUBB A,#"0"           ;TESTING
          JC BAD2               ; ACC < '0'
          SUBB A,#10
          JNC ATOF              ; ACC > '9'
          ADD A,#10             ;RESTORE
          LJMP GOOD1            ; '0' <= ACC <= '9'
ATOF:     ADD A,#{"0"+10-"A"+10}   ;CORRECT FOR (-'0'-10) & MAP 'A' INTO 10
          JB {ACC OR 7},BAD2       ; '9' < ACC < 'A'
          CLR C
          SUBB A,#16
          JNC BAD2              ; ACC > 'F'
          ADD A,#16
GOOD1:    CLR C
          RET
BAD2:     SETB C                ;SET BIT FOR INDICATER
          RET
;
;
;*************************************************************
;
;CONVERT 2 ASCII'S CHARACTER IN HEX FORM TO 1 BYTE OF HEX
;
INASCII2: LCALL INASCII1        ;GET FIRST ASCII
          JC NOTINT
          ANL A,#0FH            ;MASK 4 BITS HI
          SWAP A
          MOV B,A               ;SAVE 4 BITS HI OF HEX 1 BYTE
          LCALL INASCII1        ;GET SECOND ASCII
          JC NOTINT
          ANL A,#0FH            ;NOW HAVE 4 BIT LO OF HEX 1 BYTE
          ADD A,B               ;GET HEX 1 BYTE
          RET
```

```
NOTINT:   LJMP NOTINTEL
;
;*************************************************************
;
SINGSTP:  LCALL BYTES          ;GET USER ADDRESS FOR STEP
          JNC $+5
          LJMP WARM            ;IF NOT JMP WARM
          JNB F0,START1
          CJNE A,#CR,SINGSTP
;
          LJMP WARM
;
START1:   MOV DPTR,#0FFEDH     ;SAVE STATUS SSTEP
          MOV A,#0AAH
          MOVX @DPTR,A
          INC DPTR             ;NEXT SAVE ADD FOR STEP
          MOV A,HIBYTE
          MOVX @DPTR,A
          INC DPTR
          MOV A,LOBYTE
          MOVX @DPTR,A
          MOV TCON,#00H        ;SET LOW LEVEL LOGIC
          MOV IP,#01H          ;SET HIGHEST PRIORITY
          MOV IE,#81H          ;ENABLE INTERUPT
          MOV DPTR,#USERAD     ;SET FIRST INT SERVICE ROUTINE
          MOV R1,DPH
          MOV R0,DPL
          MOV DPTR,#0FFE0H     ;SET NEW SERVICE ROUTINE
          MOV A,#02
          MOVX @DPTR,A
          INC DPTR
          MOV A,R1             ;AT ADDRESS USEADD
          MOVX @DPTR,A
          INC DPTR
          MOV A,R0
          MOVX @DPTR,A
          MOV DPTR,#6002H      ;INTERUPT TRIG LOW
          MOV A,#0F7H
          MOVX @DPTR,A
START2:   NOP                  ;DUMMY LOOP
          NOP
          LJMP START2
;
USERAD:   MOV A,#0FFH          ;SET INT TRIG HI
          MOV DPTR,#6002H
          MOVX @DPTR,A
          MOV DPTR,#SERADD     ;SET ADD OF SERVICE ROUTINE
          MOV R1,DPH
          MOV R0,DPL
          MOV DPTR,#0FFE0H     ;SAVE ADD NEW SERVICE ROUTINE
          MOV A,#02            ;LJMP OPCODE
          MOVX @DPTR,A
          INC DPTR
          MOV A,R1
          MOVX @DPTR,A
```

```
          INC DPTR
          MOV A,R0
          MOVX @DPTR,A
          POP DPH              ;POP OLD ADD SERVICE ROUTINE
          POP DPL
          PUSH LOBYTE          ;PUT 'NEW IN THERE
          PUSH HIBYTE
          MOV A,#0F7H          ;OPEN INTERUPT
          MOV DPTR,#6002H
          MOVX @DPTR,A
          RETI
SINGST1:  NOP
          NOP
          LJMP SINGST1
;
SERADD:   NOP
          POP REG1             ;GET PC + 1
          POP REG0
          PUSH REG0            ;THEN PUSH BACK
          PUSH REG1
          PUSH ACC
          PUSH PSW
          PUSH B
          PUSH DPH
          PUSH DPL
          MOV A,#0FFH          ;DISABLE HARD WARE INTERUPTED
          MOV DPTR,#6002H
          MOVX @DPTR,A
          MOV DPTR,#0FFEEH     ;SAVE ADDRESS OF NEXT INSTRUCTION
          MOV A,R1
          MOVX @DPTR,A
          INC DPTR
          MOV A,R0
          MOVX @DPTR,A
          MOV DPTR,#MBRK1      ;SEND ADD.OF BRK-PNT TO TERMINAL
          LCALL PDATA
          MOV DPTR,#0FFEFH
          CLR A
          MOVC A,@A+DPTR
          MOV B,A
          MOV DPTR,#0FFEEH
          CLR A
          MOVC A,@A+DPTR
          MOV R0,A
          LCALL OUTR0
          MOV R0,B
          LCALL OUTR0
          MOV DPTR,#MBRK2      ;SEND 'ACC =' TO TERMINAL
          LCALL PDATA
          MOV A,SP
          CLR C
          SUBB A,#4
          MOV R0,A
          LCALL OUT2H
          MOV DPTR,#MBRK3      ;SEND 'PSW =' TO TERMINAL
```

```
        LCALL PDATA
        INC R0
        LCALL OUT2H
        MOV DPTR,#MBRK4     ;SEND 'B =' TO TERMINAL
        LCALL PDATA
        INC R0
        LCALL OUT2H
        MOV DPTR,#MBRK5     ;SEND 'DPTR =' TO TERMINAL
        LCALL PDATA
        INC R0
        LCALL OUT2H
        INC R0
        LCALL OUT2H
        MOV DPTR,#MBRK6     ;SEND 'SP =' TO TERMINAL
        LCALL PDATA
        MOV R0,STACK
        LCALL OUT2H
        LCALL CRLF
        MOV DPTR,#HSTEP
        LCALL PDATA
WCHR2:  MOV DPTR,#STEPPAT
        LCALL PDATA
WCHR:   LCALL INCH
        JNB {ACC OR 6},WCHR3 ;IF NOT LETTER
        CLR {ACC OR 5}
        CJNE A,#4EH,WCHR1
        LJMP NXTSTEP
WCHR1:  CJNE A,#51H,WCHR3
        POP DPL             ;TAKE ALL VALUE BACK
        POP DPH
        POP B
        POP PSW
        POP ACC
        POP DPH             ;REMOVE ADD OF SERV ROUTINE
        POP DPL
        MOV IE,#00H         ;DISABLE INTERUPT
        MOV IP,#00H
        MOV TCON,#00H
        MOV DPTR,#COOL      ;GO BACK TO COOL START
        PUSH DPL
        PUSH DPH
        RETI
WCHR3:  LCALL CRLF
        LJMP WCHR2
;
;
NXTSTEP: MOV A,#0F7H        ;SET HARDWARE TO INTERUPT
        MOV DPTR,#6002H
        MOVX @DPTR,A
        POP DPL             ;POP ALL VALUE BACK
        POP DPH
        POP B
        POP PSW
        POP ACC
        RETI
```

```
;
;
;************************************************************
;
FUNTAB:                         ;FUNCTION TABLE
          DFB       "A"
          DWM       ANALYS
          DFB       "E"
          DWM       ALTER       ;EDIT DATA
          DFB       "B"
          DWM       BREAK       ;BREAK POINT
          DFB       "C"
          DWM       COPY        ;COPY DATA BLOCK TO BLOCK
          DFB       "D"
          DWM       DUMP        ;DUMP DATA
          DFB       "F"
          DWM       SEARCH      ;FIND DATA IN MEM
          DFB       "G"
          DWM       GO          ;EXECUTE PROGRAM
          DFB       "H"
          DWM       HELP        ;DISPLAY HELP
          DFB       "I"
          DWM       INSERT      ;INSERT DATA ENTIRE BLOCK OF MEMORY
          DFB       "J"
          DWM       JUMPTABL    ;DISPLAY JUMP TABLE
          DFB       "L"
          DWM       DOWNLOAD    ;DOWN LOAD FORM TERMINAL
          DFB       "M"
          DWM       MOVE        ;MOVE DATA FROM CACHE RAM TO MEMORY
          DFB       "P"
          DWM       PREPRO      ;FILTER COMMAND
          DFB       "S"
          DWM       SINGSTP     ;SINGLE STEP
          DFB       "U"
          DWM       UNASS       ;UNASSEMBLER
          DFB       "V"
          DWM       VERIFY      ;COMPARE TWO BLOCK OF DATA
          DFB       "X"
          DWM       EXTCLK      ;CAP DATA BY EXT. CLOCK
          DFB       "#"
          DWM       HEXMATH     ;ADD AND SUBSTRACT
          DFB       00H         ;END OF TABLE
;
;************************************************************
;
SFRTAB:             ;SPECIAL FUNCTION REGISTER TABLE
          DFB       "P0  ",04H
          MOV R0,P0
          RET
          CLR F0
          RET
          DFB       "SP  ",04H
          MOV R0,SP
          RET
          CLR F0
```

```
RET
DFB        "DPL ",04H
MOV R0,DPL
RET
MOV DPL,R0
RET
DFB        "DPH ",04H
MOV R0,DPH
RET
MOV DPL,R0
RET
DFB        "PCON",04H
MOV R0,PCON
RET
MOV PCON,R0
RET
DFB        "TCON",04H
MOV R0,TCON
RET
MOV TCON,R0
RET
DFB        "TMOD",04H
MOV R0,TMOD
RET
MOV TMOD,R0
RET
DFB        "TL0 ",04H
MOV R0,TL0
RET
MOV TL0,R0
RET
DFB        "TL1 ",04H
MOV R0,TL1
RET
MOV TL1,R0
RET
DFB        "TH0 ",04H
MOV R0,TH0
RET
MOV TH0,R0
RET
DFB        "TH1 ",04H
MOV R0,TH1
RET
MOV TH1,R0
RET
DFB        "P1  ",04H
MOV R0,P1
RET
MOV P1,R0
RET
DFB        "SCON",04H
MOV R0,SCON
RET
MOV SCON,R0
```

```
                RET
                DFB         "SBUF",04H
                MOV R0,SBUF
                RET
                MOV SBUF,R0
                RET
                DFB         "P2  ",04H
                MOV R0,P2
                RET
                CLR F0
                RET
                DFB         "IE  ",04H
                MOV R0,IE
                RET
                MOV IE,R0
                RET
                DFB         "P3  ",04H
                MOV R0,P3
                RET
                MOV P3,R0
                RET
                DFB         "IP  ",04H
                MOV R0,IP
                RET
                MOV IP,R0
                RET
                DFB         "PSW ",04H
                MOV R0,PSW
                RET
                MOV PSW,R0
                RET
                DFB         "ACC ",04H
                MOV R0,ACC
                RET
                MOV ACC,R0
                RET
                DFB         "B   ",04H
                MOV R0,B
                RET
                MOV B,R0
                RET
;
;************************************************************
;
MPROMPT:    DFB         CR,LF,"EM51>",04H
;
MDOWNL1:    DFB         CR,LF,"Download from Terminal to 8051 Emulator"
            DFB         CR,LF,"File must be intel 8 bits short or long format"
            DFB         CR,LF,"Press function key F1 for begin process",04H
;
MDOWNL3:    DFB         07H,CR,LF,04H
;
FSPACE:     DFB         "     ",04H
;
DDOT:       DFB         "..",04H
```

```
;
FND:        DFB         CR,LF,"Found at address followed;",CR,LF,04H
;
NOTFND:     DFB         CR,LF,"Search not found",CR,LF,04H
;
MOVEPAT:    DFB         CR,LF,TAB,TAB,"Data stored at F000H - FFDFH",CR,LF,04H
;
MVPPAT:     DFB         CR,LF,TAB,TAB,"Data was filtered and stored at F000H",CR,LF,04H
;
EXTPAT:     DFB         CR,LF,TAB,TAB,"Press M for move data to memory"CR,LF,04H
;
HSTEP:      DFB         CR,LF,TAB,"Press N for next step Q for quit from step mode",CR,LF,04H
;
STEPPAT:    DFB         CR,LF,"STEP>",04H
;
;
MHELP:      DFB         CR,LF,"REGISTER BANK 1 IS RESERVED"
            DFB         CR,LF,"LAST 20 BYTES OF EXT. RAM USED BY BREAK ROUTINE"
            DFB         CR,LF,"RAM @ 48H-50H IS USED BY MONITOR"
            DFB         CR,LF,"RAM ABOVE 50H IS USED FOR STACK"
            DFB         CR,LF,"COMMANDS ARE"
            DFB         CR,LF,"A  AAAA",TAB,TAB,TAB,"GO @ AAAA AND CAPTURE DATA"
            DFB         CR,LF,"B  {AAAA}    ",TAB,TAB,"BREAK AT ADRESS AAAA      "
            DFB         CR,LF,"C  SSSS FFFF TTTT",TAB,"COPY BLOCK OF MEMORY"
            DFB         CR,LF,"D  SSSS {FFFF}",TAB,TAB,"DUMP PROGRAM MEMORY"
            DFB         CR,LF,"DI SS {FF}",TAB,TAB,"DUMP INTERNAL MEMORY"
            DFB         CR,LF,"E  SSSS",TAB,TAB,TAB,"EDIT EXTERNAL MEMORY"
            DFB         CR,LF,"EI SS",TAB,TAB,TAB,"EDIT INTERNAL MEMORY"
            DFB         CR,LF,"F  SSSS FFFF HH HH HH",TAB,"FIND DATA 'HH HH HH'"
            DFB         CR,LF,"G  {AAAA}",TAB,TAB,"GO @ AAAA OR BREAKPOINT"
            DFB         CR,LF,"H",TAB,TAB,TAB,"HELP"
            DFB         CR,LF,"I  SSSS FFFF HH",TAB,TAB,"INSERT 'HH' INTO MEMORY"
            DFB         CR,LF,"J", TAB, TAB,TAB, "LIST JUMP TABLE"
            DFB         CR,LF,"L", TAB, TAB,TAB, "DOWNLOAD FROM TERMINAL TO EXT.RAM"
            DFB         CR,LF,"M", TAB, TAB,TAB, "MOVE DATA FROM CACHE RAM TO MEMORY"
            DFB         CR,LF,"P", TAB, TAB,TAB, "COMMAND FILTER FOR 8051"
            DFB         CR,LF,"S  AAAA",TAB,TAB,TAB,"SINGLE STEP @ AAAA"
            DFB         CR,LF,"U  SSSS FFFF",TAB,TAB,"UNASSEMBLER EXTERNAL MEMORY"
            DFB         CR,LF,"V  SSSS FFFF TTTT",TAB,"VERIFY BLOCK OF MEMORY"
            DFB         CR,LF,"X",TAB,TAB,TAB,"CAPTURED EXT. DATA BY EXT. CLOCK"
DFB         CR,LF,"#    MMMM NNNN ",TAB,TAB,"HEX ADDITION & SUBTRACTION"
DFB         CR,LF,"           WHEN SSSS,FFFF,TTTT MEAN SOURCE,FINISH,TARGET ADDRESS"
            DFB         CR,LF,04H
;
MBRK1:      DFB         CR,LF,"BREAK AT LOCATION ",04H
MBRK2:      DFB         CR,LF,"ACC = ",04H
MBRK3:      DFB         "    PSW = ",04H
MBRK4:      DFB         "    B = ",04H
MBRK5:      DFB         "    DPTR = ",04H
MBRK6:      DFB         "    SP = ",04H
;
MINDEX:     DFB         CR,LF,"      0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F",04H
;
MNONO:      DFB         CR,LF,"OOPS, THAT",27H,"'S A NO-NO !",04H
;
```

```
MJUMP:    DFB       CR,LF, "0026 COOL",TAB, "COOL START (RESET STACK POINTER)"
          DFB       CR,LF, "0029 WARM",TAB, "WARM START"
          DFB       CR,LF, "002C IN",TAB,TAB, "GET A BYTE FROM THE UART"
          DFB       CR,LF, "002F INCH",TAB, "GET A CHARACTER (ZERO PARITY)"
          DFB       CR,LF, "0032 INHEX",TAB, "GET A HEX CHAR (CARRY=NONHEX)"
          DFB       CR,LF, "0035 BYTES",TAB, "2 BYTE HEX TO HI/LOBYTE"
          DFB       CR,LF, "0038 BADDR",TAB, "BUILD ADDRESS"
          DFB       CR,LF, "003B THRADR",TAB, "THREE ADDRESSES"
          DFB       CR,LF, "003E OUT",TAB, "OUTPUT BYTE IN ACC."
          DFB       CR,LF, "0041 OUTCH",TAB, "OUTPUT CHARACTER IN ACC."
          DFB       CR,LF, "0044 OUTS",TAB, "OUTPUT SPACE"
          DFB       CR,LF, "0047 OUT2S",TAB, "OUTPUT 2 SPACES"
          DFB       CR,LF, "004A CRLF",TAB, "OUTPUT [CR] AND [LF]"
          DFB       CR,LF, "004D OUT2H",TAB, "OUTPUT LOC. POINTED BY R0"
          DFB       CR,LF, "0050 OUTR0",TAB, "OUTPUT CONTENTS OF R0"
          DFB       CR,LF, "0053 OUTC2HS",TAB, "OUTPUT PROG. MEM. POINTED"
          DFB       " BY DPTR AND SPACE"
          DFB       CR,LF, "0056 PDATA",TAB, "OUTPUT MESSAGE POINTED BY DPTR"
          DFB       CR,LF, "0059 INC16",TAB, "INCREMENT 2 BYTE NO."
          DFB       " (R0=LOW BYTE)"
          DFB       CR,LF, "005C DEC16",TAB, "DECREMENT 2 BYTE NO."
          DFB       " (R0+1=HIGH BYTE)"
          DFB       CR,LF, "005F CPY",TAB, "BLOCK COPY"
          DFB       CR,LF, "0062 BRKPT",TAB, "BREAKPOINT ROUTINE"
          DFB       04H
;
;
;...END OF 8051 IN CIRCUIT EMULATOR,WRITTEN BY CHARIN B.
;
;
          ORG       2000H
A00:      EQU       2000H
          DFB       "NOP",04H
A01:      EQU       A00+10H
          ORG       A01
          DFB       "AJMP QH",04H
A02:      EQU       A01+10H
          ORG       A02
          DFB       "LJMP QQH",04H
A03:      EQU       A02+10H
          ORG       A03
          DFB       "RR A",04H
A04:      EQU       A03+10H
          ORG       A04
          DFB       "INC A",04H
A05:      EQU       A04+10H
          ORG       A05
          DFB       "INC QH",04H
A06:      EQU       A05+10H
          ORG       A06
          DFB       "INC @R0",04H
A07:      EQU       A06+10H
          ORG       A07
          DFB       "INC @R1",04H
A08:      EQU       A07+10H
```

```
        ORG     A08
        DFB     "INC R0",04H
A09:    EQU     A08+10H
        ORG     A09
        DFB     "INC R1",04H
A0A:    EQU     A09+10H
        ORG     A0A
        DFB     "INC R2",04H
A0B:    EQU     A0A+10H
        ORG     A0B
        DFB     "INC R3",04H
A0C:    EQU     A0B+10H
        ORG     A0C
        DFB     "INC R4",04H
A0D:    EQU     A0C+10H
        ORG     A0D
        DFB     "INC R5",04H
A0E:    EQU     A0D+10H
        ORG     A0E
        DFB     "INC R6",04H
A0F:    EQU     A0E+10H
        ORG     A0F
        DFB     "INC R7",04H
A10:    EQU     A0F+10H
        ORG     A10
        DFB     "JBC QH,QH",04H
A11:    EQU     A10+10H
        ORG     A11
        DFB     "ACALL QH",04H
A12:    EQU     A11+10H
        ORG     A12
        DFB     "LCALL QQH",04H
A13:    EQU     A12+10H
        ORG     A13
        DFB     "RRC A",04H
A14:    EQU     A13+10H
        ORG     A14
        DFB     "DEC A",04H
A15:    EQU     A14+10H
        ORG     A15
        DFB     "DEC QH",04H
A16:    EQU     A15+10H
        ORG     A16
        DFB     "DEC @R0",04H
A17:    EQU     A16+10H
        ORG     A17
        DFB     "DEC @R1",04H
A18:    EQU     A17+10H
        ORG     A18
        DFB     "DEC R0",04H
A19:    EQU     A18+10H
        ORG     A19
        DFB     "DEC R1",04H
A1A:    EQU     A19+10H
        ORG     A1A
```

```
        DFB     "DEC R2",04H
A1B:    EQU     A1A+10H
        ORG     A1B
        DFB     "DEC R3",04H
A1C:    EQU     A1B+10H
        ORG     A1C
        DFB     "DEC R4",04H
A1D:    EQU     A1C+10H
        ORG     A1D
        DFB     "DEC R5",04H
A1E:    EQU     A1D+10H
        ORG     A1E
        DFB     "DEC R6",04H
A1F:    EQU     A1E+10H
        ORG     A1F
        DFB     "DEC R7",04H
A20:    EQU     A1F+10H
        ORG     A20
        DFB     "JB QH,QH",04H
A21:    EQU     A20+10H
        ORG     A21
        DFB     "AJMP QH",04H
A22:    EQU     A21+10H
        ORG     A22
        DFB     "RET",04H
A23:    EQU     A22+10H
        ORG     A23
        DFB     "RL A",04H
A24:    EQU     A23+10H
        ORG     A24
        DFB     "ADD A,#QH",04H
A25:    EQU     A24+10H
        ORG     A25
        DFB     "ADD A,QH",04H
A26:    EQU     A25+10H
        ORG     A26
        DFB     "ADD A,@R0",04H
A27:    EQU     A26+10H
        ORG     A27
        DFB     "ADD A,@R1",04H
A28:    EQU     A27+10H
        ORG     A28
        DFB     "ADD A,R0",04H
A29:    EQU     A28+10H
        ORG     A29
        DFB     "ADD A,R1",04H
A2A:    EQU     A29+10H
        ORG     A2A
        DFB     "ADD A,R2",04H
A2B:    EQU     A2A+10H
        ORG     A2B
        DFB     "ADD A,R3",04H
A2C:    EQU     A2B+10H
        ORG     A2C
        DFB     "ADD A,R4",04H
```

```
A2D:    EQU     A2C+10H
        ORG     A2D
        DFB     "ADD A,R5",04H
A2E:    EQU     A2D+10H
        ORG     A2E
        DFB     "ADD A,R6",04H
A2F:    EQU     A2E+10H
        ORG     A2F
        DFB     "ADD A,R7",04H
A30:    EQU     A2F+10H
        ORG     A30
        DFB     "JNB QH,QH",04H
A31:    EQU     A30+10H
        ORG     A31
        DFB     "ACALL QH",04H
A32:    EQU     A31+10H
        ORG     A32
        DFB     "RETI",04H
A33:    EQU     A32+10H
        ORG     A33
        DFB     "RLC A",04H
A34:    EQU     A33+10H
        ORG     A34
        DFB     "ADDC A,#QH",04H
A35:    EQU     A34+10H
        ORG     A35
        DFB     "ADDC A,QH",04H
A36:    EQU     A35+10H
        ORG     A36
        DFB     "ADDC A,@R0",04H
A37:    EQU     A36+10H
        ORG     A37
        DFB     "ADDC A,@R1",04H
A38:    EQU     A37+10H
        ORG     A38
        DFB     "ADDC A,R0",04H
A39:    EQU     A38+10H
        ORG     A39
        DFB     "ADDC A,R1",04H
A3A:    EQU     A39+10H
        ORG     A3A
        DFB     "ADDC A,R2",04H
A3B:    EQU     A3A+10H
        ORG     A3B
        DFB     "ADDC A,R3",04H
A3C:    EQU     A3B+10H
        ORG     A3C
        DFB     "ADDC A,R4",04H
A3D:    EQU     A3C+10H
        ORG     A3D
        DFB     "ADDC A,R5",04H
A3E:    EQU     A3D+10H
        ORG     A3E
        DFB     "ADDC A,R6",04H
A3F:    EQU     A3E+10H
```

```
        ORG     A3F
        DFB     "ADDC A,R7",04H
A40:    EQU     A3F+10H
        ORG     A40
        DFB     "JC QH",04H
A41:    EQU     A40+10H
        ORG     A41
        DFB     "AJMP QH",04H
A42:    EQU     A41+10H
        ORG     A42
        DFB     "ORL QH,A",04H
A43:    EQU     A42+10H
        ORG     A43
        DFB     "ORL QH,QH",04H
A44:    EQU     A43+10H
        ORG     A44
        DFB     "ORL A,#QH",04H
A45:    EQU     A44+10H
        ORG     A45
        DFB     "ORL A,QH",04H
A46:    EQU     A45+10H
        ORG     A46
        DFB     "ORL A,@R0",04H
A47:    EQU     A46+10H
        ORG     A47
        DFB     "ORL A,@R1",04H
A48:    EQU     A47+10H
        ORG     A48
        DFB     "ORL A,R0",04H
A49:    EQU     A48+10H
        ORG     A49
        DFB     "ORL A,R1",04H
A4A:    EQU     A49+10H
        ORG     A4A
        DFB     "ORL A,R2",04H
A4B:    EQU     A4A+10H
        ORG     A4B
        DFB     "ORL A,R3",04H
A4C:    EQU     A4B+10H
        ORG     A4C
        DFB     "ORL A,R4",04H
A4D:    EQU     A4C+10H
        ORG     A4D
        DFB     "ORL A,R5",04H
A4E:    EQU     A4D+10H
        ORG     A4E
        DFB     "ORL A,R6",04H
A4F:    EQU     A4E+10H
        ORG     A4F
        DFB     "ORL A,R7",04H
A50:    EQU     A4F+10H
        ORG     A50
        DFB     "JNC QH",04H
A51:    EQU     A50+10H
        ORG     A51
```

```
        DFB     "ACALL QH",04H
A52:    EQU     A51+10H
        ORG     A52
        DFB     "ANL QH,A",04H
A53:    EQU     A52+10H
        ORG     A53
        DFB     "ANL QH,QH",04H
A54:    EQU     A53+10H
        ORG     A54
        DFB     "ANL A,#QH",04H
A55:    EQU     A54+10H
        ORG     A55
        DFB     "ANL A,QH",04H
A56:    EQU     A55+10H
        ORG     A56
        DFB     "ANL A,@R0",04H
A57:    EQU     A56+10H
        ORG     A57
        DFB     "ANL A,@R1",04H
A58:    EQU     A57+10H
        ORG     A58
        DFB     "ANL A,R0",04H
A59:    EQU     A58+10H
        ORG     A59
        DFB     "ANL A,R1",04H
A5A:    EQU     A59+10H
        ORG     A5A
        DFB     "ANL A,R2",04H
A5B:    EQU     A5A+10H
        ORG     A5B
        DFB     "ANL A,R3",04H
A5C:    EQU     A5B+10H
        ORG     A5C
        DFB     "ANL A,R4",04H
A5D:    EQU     A5C+10H
        ORG     A5D
        DFB     "ANL A,R5",04H
A5E:    EQU     A5D+10H
        ORG     A5E
        DFB     "ANL A,R6",04H
A5F:    EQU     A5E+10H
        ORG     A5F
        DFB     "ANL A,R7",04H
A60:    EQU     A5F+10H
        ORG     A60
        DFB     "JZ QH",04H
A61:    EQU     A60+10H
        ORG     A61
        DFB     "AJMP QH",04H
A62:    EQU     A61+10H
        ORG     A62
        DFB     "XRL QH,A",04H
A63:    EQU     A62+10H
        ORG     A63
        DFB     "XRL QH,QH",04H
```

```
A64:    EQU     A63+10H
        ORG     A64
        DFB     "XRL A,#QH",04H
A65:    EQU     A64+10H
        ORG     A65
        DFB     "XRL A,QH",04H
A66:    EQU     A65+10H
        ORG     A66
        DFB     "XRL A,@R0",04H
A67:    EQU     A66+10H
        ORG     A67
        DFB     "XRL A,@R1",04H
A68:    EQU     A67+10H
        ORG     A68
        DFB     "XRL A,R0",04H
A69:    EQU     A68+10H
        ORG     A69
        DFB     "XRL A,R1",04H
A6A:    EQU     A69+10H
        ORG     A6A
        DFB     "XRL A,R2",04H
A6B:    EQU     A6A+10H
        ORG     A6B
        DFB     "XRL A,R3",04H
A6C:    EQU     A6B+10H
        ORG     A6C
        DFB     "XRL A,R4",04H
A6D:    EQU     A6C+10H
        ORG     A6D
        DFB     "XRL A,R5",04H
A6E:    EQU     A6D+10H
        ORG     A6E
        DFB     "XRL A,R6",04H
A6F:    EQU     A6E+10H
        ORG     A6F
        DFB     "XRL A,R7",04H
A70:    EQU     A6F+10H
        ORG     A70
        DFB     "JNZ QH",04H
A71:    EQU     A70+10H
        ORG     A71
        DFB     "ACALL QH",04H
A72:    EQU     A71+10H
        ORG     A72
        DFB     "ORL C,QH",04H
A73:    EQU     A72+10H
        ORG     A73
        DFB     "JMP @A+DPTR",04H
A74:    EQU     A73+10H
        ORG     A74
        DFB     "MOV A,#QH",04H
A75:    EQU     A74+10H
        ORG     A75
        DFB     "MOV QH,#QH",04H
A76:    EQU     A75+10H
```

```
        ORG     A76
        DFB     "MOV @R0,#QH",04H
A77:    EQU     A76+10H
        ORG     A77
        DFB     "MOV @R1,#QH",04H
A78:    EQU     A77+10H
        ORG     A78
        DFB     "MOV R0,#QH",04H
A79:    EQU     A78+10H
        ORG     A79
        DFB     "MOV R1,#QH",04H
A7A:    EQU     A79+10H
        ORG     A7A
        DFB     "MOV R2,#QH",04H
A7B:    EQU     A7A+10H
        ORG     A7B
        DFB     "MOV R3,#QH",04H
A7C:    EQU     A7B+10H
        ORG     A7C
        DFB     "MOV R4,#QH",04H
A7D:    EQU     A7C+10H
        ORG     A7D
        DFB     "MOV R5,#QH",04H
A7E:    EQU     A7D+10H
        ORG     A7E
        DFB     "MOV R6,#QH",04H
A7F:    EQU     A7E+10H
        ORG     A7F
        DFB     "MOV R7,#QH",04H
A80:    EQU     A7F+10H
        ORG     A80
        DFB     "SJMP QH",04H
A81:    EQU     A80+10H
        ORG     A81
        DFB     "AJMP QH",04H
A82:    EQU     A81+10H
        ORG     A82
        DFB     "ANL C,QH",04H
A83:    EQU     A82+10H
        ORG     A83
        DFB     "MOVC A,@A+PC",04H
A84:    EQU     A83+10H
        ORG     A84
        DFB     "DIV AB",04H
A85:    EQU     A84+10H
        ORG     A85
        DFB     "MOV QH,QH",04H
A86:    EQU     A85+10H
        ORG     A86
        DFB     "MOV QH,@R0",04H
A87:    EQU     A86+10H
        ORG     A87
        DFB     "MOV QH,@R1",04H
A88:    EQU     A87+10H
        ORG     A88
```

```
        DFB     "MOV QH,R0",04H
A89:    EQU     A88+10H
        ORG     A89
        DFB     "MOV QH,R1",04H
A8A:    EQU     A89+10H
        ORG     A8A
        DFB     "MOV QH,R2",04H
A8B:    EQU     A8A+10H
        ORG     A8B
        DFB     "MOV QH,R3",04H
A8C:    EQU     A8B+10H
        ORG     A8C
        DFB     "MOV QH,R4",04H
A8D:    EQU     A8C+10H
        ORG     A8D
        DFB     "MOV QH,R5",04H
A8E:    EQU     A8D+10H
        ORG     A8E
        DFB     "MOV QH,R6",04H
A8F:    EQU     A8E+10H
        ORG     A8F
        DFB     "MOV QH,R7",04H
A90:    EQU     A8F+10H
        ORG     A90
        DFB     "MOV DPTR,#QQH",04H
A91:    EQU     A90+10H
        ORG     A91
        DFB     "ACALL QH"
A92:    EQU     A91+10H
        ORG     A92
        DFB     "MOV QH,C",04H
A93:    EQU     A92+10H
        ORG     A93
        DFB     "MOVC A,@A+DPTR",04H
A94:    EQU     A93+10H
        ORG     A94
        DFB     "SUBB A,#QH",04H
A95:    EQU     A94+10H
        ORG     A95
        DFB     "SUBB A,QH",04H
A96:    EQU     A95+10H
        ORG     A96
        DFB     "SUBB A,@R0",04H
A97:    EQU     A96+10H
        ORG     A97
        DFB     "SUBB A,@R1",04H
A98:    EQU     A97+10H
        ORG     A98
        DFB     "SUBB A,R0",04H
A99:    EQU     A98+10H
        ORG     A99
        DFB     "SUBB A,R1",04H
A9A:    EQU     A99+10H
        ORG     A9A
        DFB     "SUBB A,R2",04H
```

```
A9B:    EQU     A9A+10H
        ORG     A9B
        DFB     "SUBB A,R3",04H
A9C:    EQU     A9B+10H
        ORG     A9C
        DFB     "SUBB A,R4",04H
A9D:    EQU     A9C+10H
        ORG     A9D
        DFB     "SUBB A,R5",04H
A9E:    EQU     A9D+10H
        ORG     A9E
        DFB     "SUBB A,R6",04H
A9F:    EQU     A9E+10H
        ORG     A9F
        DFB     "SUBB A,R7",04H
AA0:    EQU     A9F+10H
        ORG     AA0
        DFB     "ORL C,/QH",04H
AA1:    EQU     AA0+10H
        ORG     AA1
        DFB     "AJMP QH",04H
AA2:    EQU     AA1+10H
        ORG     AA2
        DFB     "MOV C,QH",04H
AA3:    EQU     AA2+10H
        ORG     AA3
        DFB     "INC DPTR",04H
AA4:    EQU     AA3+10H
        ORG     AA4
        DFB     "MUL AB",04H
AA5:    EQU     AA4+10H
        ORG     AA5
        DFB     "RESERVED",04H
AA6:    EQU     AA5+10H
        ORG     AA6
        DFB     "MOV @R0,QH",04H
AA7:    EQU     AA6+10H
        ORG     AA7
        DFB     "MOV @R1,QH",04H
AA8:    EQU     AA7+10H
        ORG     AA8
        DFB     "MOV R0,QH",04H
AA9:    EQU     AA8+10H
        ORG     AA9
        DFB     "MOV R1,QH",04H
AAA:    EQU     AA9+10H
        ORG     AAA
        DFB     "MOV R2,QH",04H
AAB:    EQU     AAA+10H
        ORG     AAB
        DFB     "MOV R3,QH",04H
AAC:    EQU     AAB+10H
        ORG     AAC
        DFB     "MOV R4,QH",04H
AAD:    EQU     AAC+10H
```

```
        ORG     AAD
        DFB     "MOV R5,QH",04H
AAE:    EQU     AAD+10H
        ORG     AAE
        DFB     "MOV R6,QH",04H
AAF:    EQU     AAE+10H
        ORG     AAF
        DFB     "MOV R7,QH",04H
AB0:    EQU     AAF+10H
        ORG     AB0
        DFB     "ANL C,/QH",04H
AB1:    EQU     AB0+10H
        ORG     AB1
        DFB     "ACALL QH",04H
AB2:    EQU     AB1+10H
        ORG     AB2
        DFB     "CPL Z",04H
AB3:    EQU     AB2+10H
        ORG     AB3
        DFB     "CPL C",04H
AB4:    EQU     AB3+10H
        ORG     AB4
        DFB     "CJNE A,#QH,QH",04H
AB5:    EQU     AB4+10H
        ORG     AB5
        DFB     "CJNE A,QH,QH",04H
AB6:    EQU     AB5+10H
        ORG     AB6
        DFB     "CJNE @R0,#QH,QH",04H
AB7:    EQU     AB6+10H
        ORG     AB7
        DFB     "CJNE @R1,#QH,QH",04H
AB8:    EQU     AB7+10H
        ORG     AB8
        DFB     "CJNE R0,#QH,QH",04H
AB9:    EQU     AB8+10H
        ORG     AB9
        DFB     "CJNE R1,#QH,QH",04H
ABA:    EQU     AB9+10H
        ORG     ABA
        DFB     "CJNE R2,#QH,QH",04H
ABB:    EQU     ABA+10H
        ORG     ABB
        DFB     "CJNE R3,#QH,QH",04H
ABC:    EQU     ABB+10H
        ORG     ABC
        DFB     "CJNE R4,#QH,QH",04H
ABD:    EQU     ABC+10H
        ORG     ABD
        DFB     "CJNE R5,#QH,QH",04H
ABE:    EQU     ABD+10H
        ORG     ABE
        DFB     "CJNE R6,#QH,QH",04H
ABF:    EQU     ABE+10H
        ORG     ABF
```

```
        DFB     "CJNE R7,#QH,QH",04H
AC0:    EQU     ABF+10H
        ORG     AC0
        DFB     "PUSH QH",04H
AC1:    EQU     AC0+10H
        ORG     AC1
        DFB     "AJMP QH",04H
AC2:    EQU     AC1+10H
        ORG     AC2
        DFB     "CLR Z",04H
AC3:    EQU     AC2+10H
        ORG     AC3
        DFB     "CLR C",04H
AC4:    EQU     AC3+10H
        ORG     AC4
        DFB     "SWAP A",04H
AC5:    EQU     AC4+10H
        ORG     AC5
        DFB     "XCH A,QH",04H
AC6:    EQU     AC5+10H
        ORG     AC6
        DFB     "XCH A,@R0",04H
AC7:    EQU     AC6+10H
        ORG     AC7
        DFB     "XCH A,@R1",04H
AC8:    EQU     AC7+10H
        ORG     AC8
        DFB     "XCH A,R0",04H
AC9:    EQU     AC8+10H
        ORG     AC9
        DFB     "XCH A,R1",04H
ACA:    EQU     AC9+10H
        ORG     ACA
        DFB     "XCH A,R2",04H
ACB:    EQU     ACA+10H
        ORG     ACB
        DFB     "XCH A,R3",04H
ACC1:   EQU     ACB+10H
        ORG     ACC
        DFB     "XCH A,R4",04H
ACD:    EQU     ACC1+10H
        ORG     ACD
        DFB     "XCH A,R5",04H
ACE:    EQU     ACD+10H
        ORG     ACE
        DFB     "XCH A,R6",04H
ACF:    EQU     ACE+10H
        ORG     ACF
        DFB     "XCH A,R7",04H
AD0:    EQU     ACF+10H
        ORG     AD0
        DFB     "POP QH",04H
AD1:    EQU     AD0+10H
        ORG     AD1
        DFB     "ACALL QH",04H
```

```
AD2:    EQU     AD1+10H
        ORG     AD2
        DFB     "SETB QH",04H
AD3:    EQU     AD2+10H
        ORG     AD3
        DFB     "SETB C",04H
AD4:    EQU     AD3+10H
        ORG     AD4
        DFB     "DA A",04H
AD5:    EQU     AD4+10H
        ORG     AD5
        DFB     "DJNZ QH,QH",04H
AD6:    EQU     AD5+10H
        ORG     AD6
        DFB     "XCHD A,@R0",04H
AD7:    EQU     AD6+10H
        ORG     AD7
        DFB     "XCHD A,@R1",04H
AD8:    EQU     AD7+10H
        ORG     AD8
        DFB     "DJNZ R0,QH",04H
AD9:    EQU     AD8+10H
        ORG     AD9
        DFB     "DJNZ R1,QH",04H
ADA:    EQU     AD9+10H
        ORG     ADA
        DFB     "DJNZ R2,QH",04H
ADB:    EQU     ADA+10H
        ORG     ADB
        DFB     "DJNZ R3,QH",04H
ADC:    EQU     ADB+10H
        ORG     ADC
        DFB     "DJNZ R4,QH",04H
ADD:    EQU     ADC+10H
        ORG     ADD
        DFB     "DJNZ R5,QH",04H
ADE:    EQU     ADD+10H
        ORG     ADE
        DFB     "DJNZ R6,QH",04H
ADF:    EQU     ADE+10H
        ORG     ADF
        DFB     "DJNZ R7,QH",04H
AE0:    EQU     ADF+10H
        ORG     AE0
        DFB     "MOVX A,@DPTR",04H
AE1:    EQU     AE0+10H
        ORG     AE1
        DFB     "AJMP QH",04H
AE2:    EQU     AE1+10H
        ORG     AE2
        DFB     "MOVX A,@R0",04H
AE3:    EQU     AE2+10H
        ORG     AE3
        DFB     "MOVX A,@R1",04H
AE4:    EQU     AE3+10H
```

```
        ORG     AE4
        DFB     "CLR A",04H
AE5:    EQU     AE4+10H
        ORG     AE5
        DFB     "MOV A,QH",04H
AE6:    EQU     AE5+10H
        ORG     AE6
        DFB     "MOV A,@R0",04H
AE7:    EQU     AE6+10H
        ORG     AE7
        DFB     "MOV A,@R1",04H
AE8:    EQU     AE7+10H
        ORG     AE8
        DFB     "MOV A,R0",04H
AE9:    EQU     AE8+10H
        ORG     AE9
        DFB     "MOV A,R1",04H
AEA:    EQU     AE9+10H
        ORG     AEA
        DFB     "MOV A,R2",04H
AEB:    EQU     AEA+10H
        ORG     AEB
        DFB     "MOV A,R3",04H
AEC:    EQU     AEB+10H
        ORG     AEC
        DFB     "MOV A,R4",04H
AED:    EQU     AEC+10H
        ORG     AED
        DFB     "MOV A,R5",04H
AEE:    EQU     AED+10H
        ORG     AEE
        DFB     "MOV A,R6",04H
AEF:    EQU     AEE+10H
        ORG     AEF
        DFB     "MOV A,R7",04H
AF0:    EQU     AEF+10H
        ORG     AF0
        DFB     "MOVX @DPTR,A",04H
AF1:    EQU     AF0+10H
        ORG     AF1
        DFB     "ACALL QH",04H
AF2:    EQU     AF1+10H
        ORG     AF2
        DFB     "MOVX @R0,A",04H
AF3:    EQU     AF2+10H
        ORG     AF3
        DFB     "MOVX @R1,A",04H
AF4:    EQU     AF3+10H
        ORG     AF4
        DFB     "CPL A",04H
AF5:    EQU     AF4+10H
        ORG     AF5
        DFB     "MOV QH,A",04H
AF6:    EQU     AF5+10H
        ORG     AF6
```

```
        DFB     "MOV @R0,A",04H
AF7:    EQU     AF6+10H
        ORG     AF7
        DFB     "MOV @R1,A",04H
AF8:    EQU     AF7+10H
        ORG     AF8
        DFB     "MOV R0,A",04H
AF9:    EQU     AF8+10H
        ORG     AF9
        DFB     "MOV R1,A",04H
AFA:    EQU     AF9+10H
        ORG     AFA
        DFB     "MOV R2,A",04H
AFB:    EQU     AFA+10H
        ORG     AFB
        DFB     "MOV R3,A",04H
AFC:    EQU     AFB+10H
        ORG     AFC
        DFB     "MOV R4,A",04H
AFD:    EQU     AFC+10H
        ORG     AFD
        DFB     "MOV R5,A",04H
AFE:    EQU     AFD+10H
        ORG     AFE
        DFB     "MOV R6,A",04H
AFF:    EQU     AFE+10H
        ORG     AFF
        DFB     "MOV R7,A",04H
;
;
        ORG 3000H
;
;NUMBER OF CYCLES AND NUMBER OF BYTES
;
B00:    DFB     11H
B01:    DFB     22H
B02:    DFB     23H
B03:    DFB     11H
B04:    DFB     11H
B05:    DFB     12H
B06:    DFB     11H
B07:    DFB     11H
B08:    DFB     11H
B09:    DFB     11H
B0A:    DFB     11H
B0B:    DFB     11H
B0C:    DFB     11H
B0D:    DFB     11H
B0E:    DFB     11H
B0F:    DFB     11H
B10:    DFB     23H
B11:    DFB     22H
B12:    DFB     23H
B13:    DFB     11H
B14:    DFB     11H
```

```
B15:    DFB     12H
B16:    DFB     11H
B17:    DFB     11H
B18:    DFB     11H
B19:    DFB     21H
B1A:    DFB     11H
B1B:    DFB     11H
B1C:    DFB     11H
B1D:    DFB     11H
B1E:    DFB     11H
B1F:    DFB     11H
B20:    DFB     23H
B21:    DFB     22H
B22:    DFB     21H
B23:    DFB     11H
B24:    DFB     12H
B25:    DFB     12H
B26:    DFB     11H
B27:    DFB     11H
B28:    DFB     11H
B29:    DFB     11H
B2A:    DFB     11H
B2B:    DFB     11H
B2C:    DFB     11H
B2D:    DFB     11H
B2E:    DFB     11H
B2F:    DFB     11H
B30:    DFB     23H
B31:    DFB     22H
B32:    DFB     21H
B33:    DFB     11H
B34:    DFB     12H
B35:    DFB     12H
B36:    DFB     11H
B37:    DFB     11H
B38:    DFB     11H
B39:    DFB     11H
B3A:    DFB     11H
B3B:    DFB     11H
B3C:    DFB     11H
B3D:    DFB     11H
B3E:    DFB     11H
B3F:    DFB     11H
B40:    DFB     22H
B41:    DFB     22H
B42:    DFB     12H
B43:    DFB     23H
B44:    DFB     12H
B45:    DFB     12H
B46:    DFB     11H
B47:    DFB     11H
B48:    DFB     11H
B49:    DFB     11H
B4A:    DFB     11H
B4B:    DFB     11H
```

```
B4C:    DFB     11H
B4D:    DFB     11H
B4E:    DFB     11H
B4F:    DFB     11H
B50:    DFB     22H
B51:    DFB     22H
B52:    DFB     12H
B53:    DFB     23H
B54:    DFB     12H
B55:    DFB     12H
B56:    DFB     11H
B57:    DFB     11H
B58:    DFB     11H
B59:    DFB     11H
B5A:    DFB     11H
B5B:    DFB     11H
B5C:    DFB     11H
B5D:    DFB     11H
B5E:    DFB     11H
B5F:    DFB     11H
B60:    DFB     22H
B61:    DFB     22H
B62:    DFB     12H
B63:    DFB     23H
B64:    DFB     12H
B65:    DFB     12H
B66:    DFB     11H
B67:    DFB     11H
B68:    DFB     11H
B69:    DFB     11H
B6A:    DFB     11H
B6B:    DFB     11H
B6C:    DFB     11H
B6D:    DFB     11H
B6E:    DFB     11H
B6F:    DFB     11H
B70:    DFB     22H
B71:    DFB     22H
B72:    DFB     21H
B73:    DFB     22H
B74:    DFB     12H
B75:    DFB     23H
B76:    DFB     12H
B77:    DFB     12H
B78:    DFB     12H
B79:    DFB     12H
B7A:    DFB     12H
B7B:    DFB     12H
B7C:    DFB     12H
B7D:    DFB     12H
B7E:    DFB     12H
B7F:    DFB     12H
B80:    DFB     22H
B81:    DFB     22H
B82:    DFB     22H
```

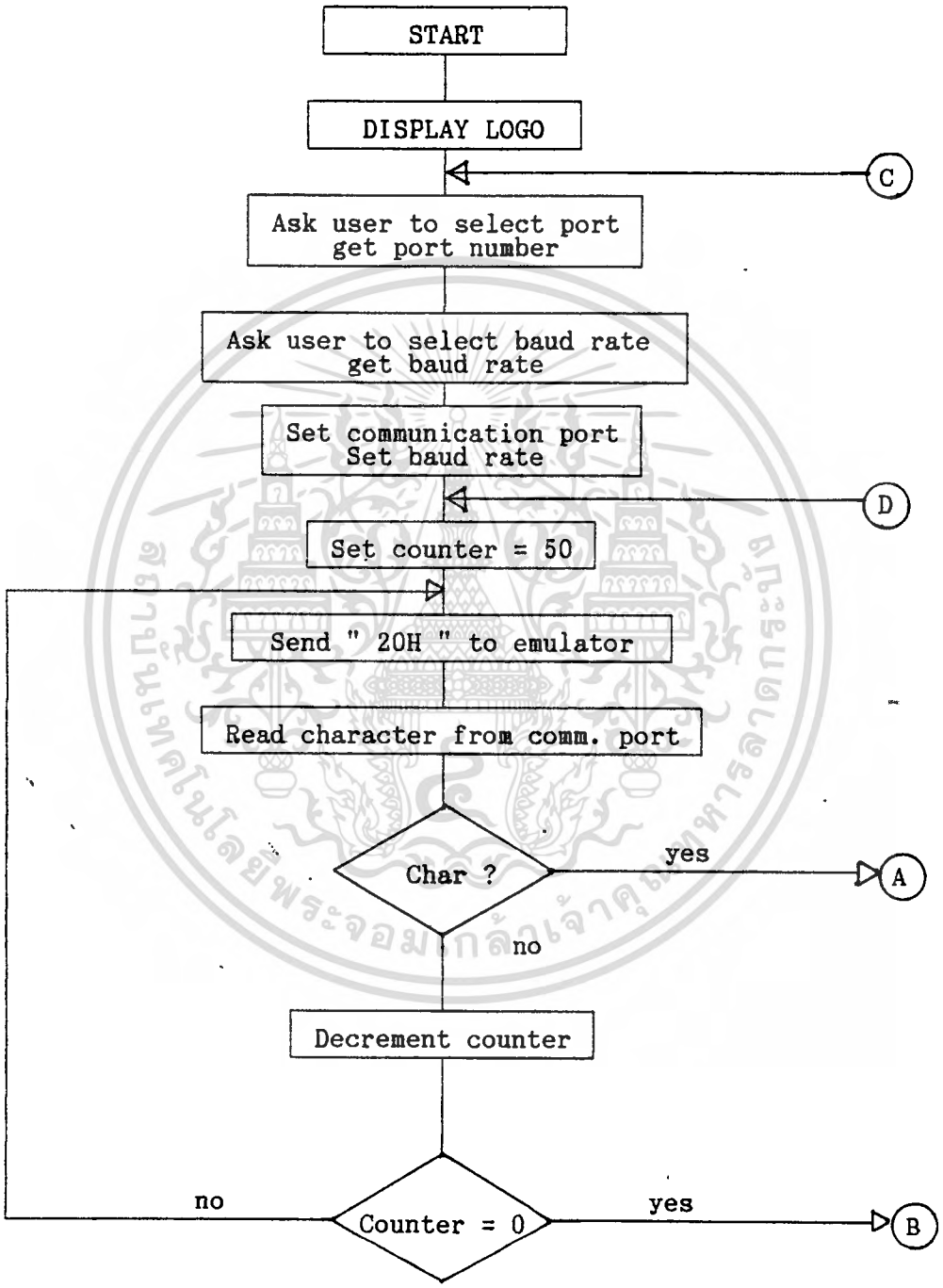
```
B83:    DFB     21H
B84:    DFB     41H
B85:    DFB     23H
B86:    DFB     22H
B87:    DFB     22H
B88:    DFB     22H
B89:    DFB     22H
B8A:    DFB     22H
B8B:    DFB     22H
B8C:    DFB     22H
B8D:    DFB     22H
B8E:    DFB     22H
B8F:    DFB     22H
B90:    DFB     23H
B91:    DFB     22H
B92:    DFB     22H
B93:    DFB     11H
B94:    DFB     12H
B95:    DFB     12H
B96:    DFB     11H
B97:    DFB     11H
B98:    DFB     11H
B99:    DFB     11H
B9A:    DFB     11H
B9B:    DFB     11H
B9C:    DFB     11H
B9D:    DFB     11H
B9E:    DFB     11H
B9F:    DFB     11H
BA0:    DFB     22H
BA1:    DFB     22H
BA2:    DFB     12H
BA3:    DFB     21H
BA4:    DFB     41H
BA5:    DFB     00H
BA6:    DFB     22H
BA7:    DFB     22H
BA8:    DFB     22H
BA9:    DFB     22H
BAA:    DFB     22H
BAB:    DFB     22H
BAC:    DFB     22H
BAD9:   DFB     22H
BAE:    DFB     22H
BAF:    DFB     22H
BB0:    DFB     22H
BB1:    DFB     22H
BB2:    DFB     12H
BB3:    DFB     11H
BB4:    DFB     23H
BB5:    DFB     23H
BB6:    DFB     23H
BB7:    DFB     23H
BB8:    DFB     23H
BB9:    DFB     23H
```

```
BBA:    DFB     23H
BBB:    DFB     23H
BBC:    DFB     23H
BBD:    DFB     23H
BBE:    DFB     23H
BBF:    DFB     23H
BC0:    DFB     22H
BC1:    DFB     22H
BC2:    DFB     12H
BC3:    DFB     11H
BC4:    DFB     11H
BC5:    DFB     12H
BC6:    DFB     11H
BC7:    DFB     11H
BC8:    DFB     11H
BC9:    DFB     11H
BCA:    DFB     11H
BCB:    DFB     11H
BCC:    DFB     11H
BCD:    DFB     11H
BCE:    DFB     11H
BCF:    DFB     11H
BD0:    DFB     22H
BD1:    DFB     22H
BD2:    DFB     12H
BD3:    DFB     11H
BD4:    DFB     11H
BD5:    DFB     23H
BD6:    DFB     11H
BD7:    DFB     11H
BD8:    DFB     22H
BD9:    DFB     22H
BDA:    DFB     22H
BDB:    DFB     22H
BDC:    DFB     22H
BDD:    DFB     22H
BDE:    DFB     22H
BDF:    DFB     22H
BE0:    DFB     21H
BE1:    DFB     22H
BE2:    DFB     21H
BE3:    DFB     21H
BE4:    DFB     11H
BE5:    DFB     12H
BE6:    DFB     11H
BE7:    DFB     11H
BE8:    DFB     11H
BE9:    DFB     11H
BEA:    DFB     11H
BEB:    DFB     11H
BEC:    DFB     11H
BED:    DFB     11H
BEE:    DFB     11H
BEF:    DFB     11H
BF0:    DFB     11H
```

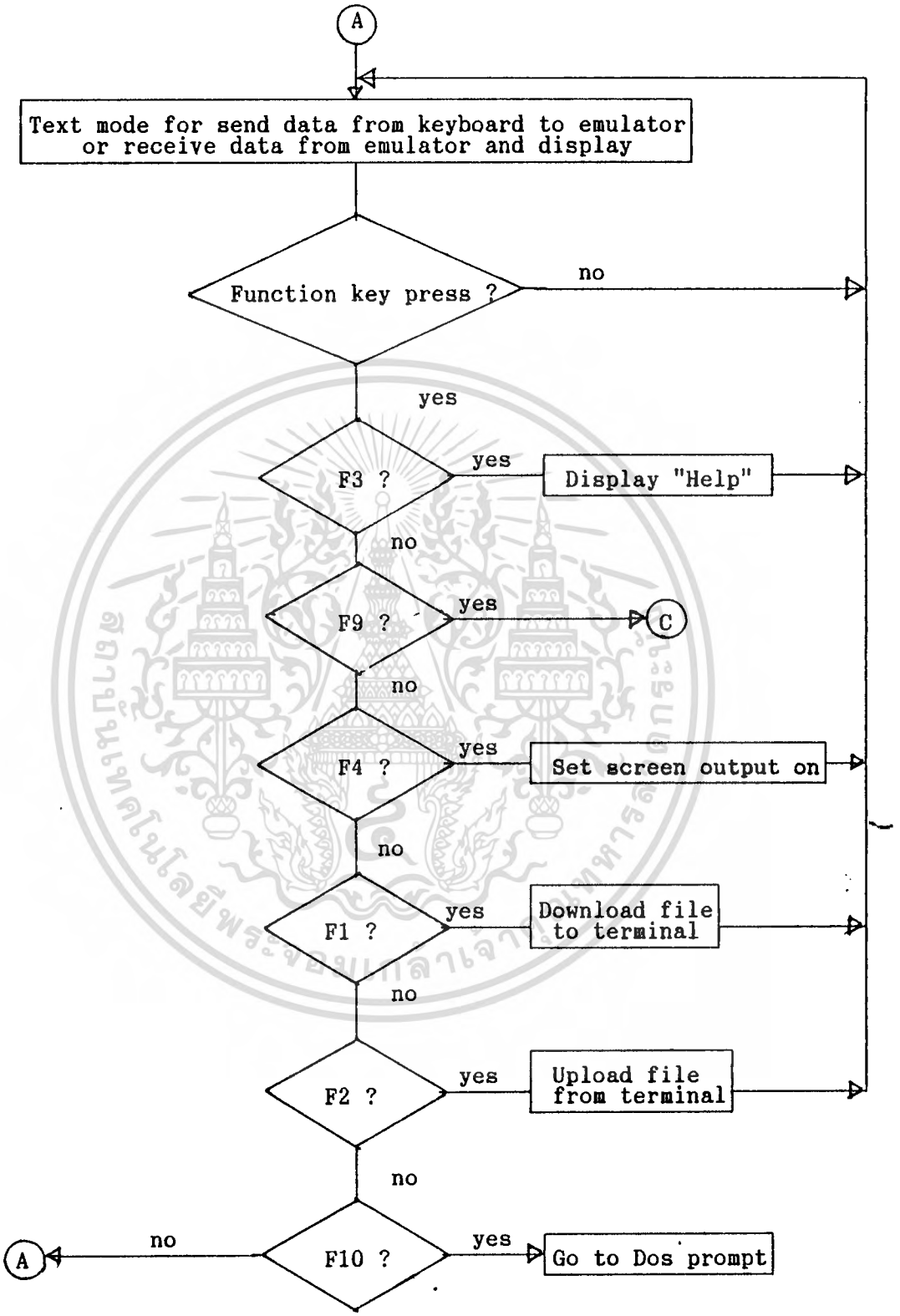
```
BF1:        DFB         22H
BF2:        DFB         21H
BF3:        DFB         21H
BF4:        DFB         11H
BF5:        DFB         12H
BF6:        DFB         11H
BF7:        DFB         11H
BF8:        DFB         11H
BF9:        DFB         11H
BFA:        DFB         11H
BFB:        DFB         11H
BFC:        DFB         11H
BFD:        DFB         11H
BFE:        DFB         11H
BFF:        DFB         11H
;
;
            END
```

ภาคผนวกที่ 4

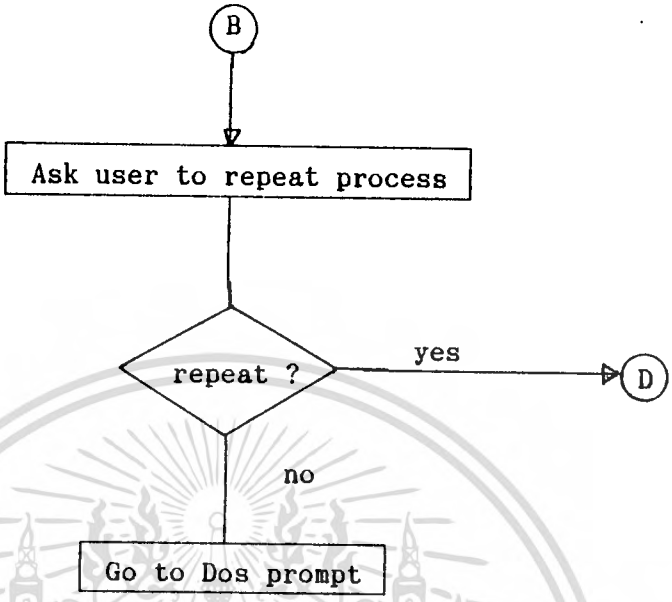
โฟลว์ชาร์ทของโปรแกรมที่ใช้งานบนเครื่องคอมพิวเตอร์ส่วนบุคคล IBM-PC



รูปที่ 5.1 แสดงโฟลว์ชาร์ทของโปรแกรม terminal.exe



รูปที่ 5.2 แสดงโฟลว์ชาร์ทของโปรแกรม terminal.exe



รูปที่ 5.3 แสดงโฟลว์ชาร์ทของโปรแกรม terminal.exe