การจำลองการทำงานของไมโครคอมพิวเตอร์ 8051

EMULATOR 8051

โดย

นาย บุญเลิศ ยุทธสุภากร

นาย วิวัฒน์ เลิศทนงศักดิ์

นาย สรพงษ์ กฤษณะจูฑะ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

การจำลองการทำงานของไมโครคอมพิวเตอร์ 8051

Emulation 8051

โดย

นาย บุญเลิศ ยุทธสุภากร 37014220

นาย วิวัฒน์ เลิศทนงศักดิ์ 37014413

นาย สรพงษ์ กฤษณะจูฑะ 37014473

อาจารย์ที่ปรึกษา

ผศ. เกรียงไกร วงศ์โรจน์ภรณ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมโทรคมนาคม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ปริญญานิพนธ์ปีการศึกษา 2540

ภาควิชาวิศวกรรมไทรคมนาคม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง **การจำลองการทำงานของไมโครคอมพิวเตอร์ 8051**

(Emulation 8051)

ผู้จัดทำ

1. นาย บุญเลิศ ยุทธสุภากร 37014220
2. นาย วิวัฒน์ เลิศทนงศักดิ์ 37014413
3. นาย สรพงษ์ กฤษณะจูฑะ 37014473

อาจารย์ที่ปรึกษา

(ผศ. เกรียงไกร วงศ์โรจน์ภรณ์)

การจำลองการทำงานของไมโครคอมพิวเตอร์ 8051

EMULATOR 8051

โดย

นาย บุญเลิศ ยุทธสุภากร 37014220

นาย วิวัฒน์ เลิศทนงศักดิ์ 37014413

นาย สรพงษ์ กฤษณะจูฑะ 37014473

อาจารย์ที่ปรึกษา

ผศ. เกรียงไกร วงศ์โรจน์ภรณ์

บทคัดย่อ

ปริญญานิพนธ์นี้เป็นการเสนอแนวทางพัฒนาเพื่อใช้ในการจำลองการทำงานของไมโครคอมพิวเตอร์แบบชิฟเดียว ตระกูล 8051 ขึ้นมาใช้งาน โดยที่ตัวอีมูลเลเตอร์มักถูกใช้เป็นเครื่องมือ สำหรับช่วยตรวจสอบหรือแก้ไขระบบไมโครคอมพิวเตอร์ที่มีข้อผิดพลาดขึ้นทางซอฟแวร์ ทั้งนี้เนื่องจากอีมูลเลเตอร์สามารถนำเอาข้อมูลต่าง ๆ ภายในรีจิสเตอร์ที่บกพร่องขึ้นมาตรวจสอบได้ เช่น ข้อมูลภายในรีจิสเตอร์ ข้อมูลที่เก็บไว้ในหน่วยความจำ หรือข้อมูลที่รับเข้ามาจากภายนอกระบบ เป็นต้น และเมื่อผู้ใช้สามารถตรวจสอบข้อมูลได้อย่างถูกต้อง การวิเคราะห์ปัญหาที่เกิดขึ้นจะทำได้อย่างรวดเร็วและแม่นยำ ทำให้ระยะเวลาที่ใช้ในการแก้ปัญหาลดน้อยลง

ABSTRTACT

This paper presents the development of emlation for single chip mircomputer 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 external 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.

สารบัญ

สารบัญรูป

บทที่ 1 บทนำ

ปัจจุบันนี้มีการนำเอาระบบไมโครคอมพิวเตอร์มาเป็นส่วนสำคัญสำหรับควบคุมเครื่องใช้ไฟฟ้ากันอย่างแพร่หลาย เช่น อุปกรณ์ควบคุมการทำงานอัตโนมัติในโรงงานอุตสาหกรรม หรือแม้กระทั่งอุปกรณ์ที่ใช้ในสำนักงานก็พบได้หลายชนิด เครื่องใช้ไฟฟ้าดังกล่าวมีขนาดเล็กลงและมีการทำงานสลับซับซ้อนมากขึ้น สาเหตุที่เป็นเช่นนี้เนื่องมาจากผู้ผลิตมักจะนำไมโครคอมพิวเตอร์แบบชิฟเดียวมาใช้งานเป็นจำนวนมาก

ไมโครคอมพิวเตอร์แบบชิฟเดียว (single chip microcomputer) จะมีข้อดีกว่าระบบไมโครคอมพิวเตอร์ทั่ว ๆ ไป คือ อุปกรณ์สนับสนุนต่าง ๆ ที่ต้องใช้งานร่วมกับไมโครโปรเซสเซอร์จะอยู่ภายในชิฟเดียวกันกับไมโครโปรเซสเซอร์ อุปกรณ์เหล่านี้ได้แก่ หน่วยความจำทั้งชนิดแรมและรอมพอร์ทแบบขนานและพอร์ทแบบอนุกรม ด้วยจุดเด่นข้อนี้ไมโครคอมพิวเตอร์แบบชิฟเดียวจึงมักถูกนำมาใช้งานควบคุมที่มีโปรแกรมไม่สลับซับซ้อนมากนัก แต่ต้องการความเร็วในการทำงานสูงพอสมควร

เมื่อพิจารณาในอีกด้านหนึ่งจะเห็นว่าการใช้ไมโครคอมพิวเตอร์เป็นส่วนควบคุมหลักแล้ว ในกรณีที่อุปกรณ์นั้นขัดข้องหรือมีจุดผิดพลาดในโปรแกรมที่ใช้งาน การตรวจสอบแก้ไขจะมีขั้นตอนที่ยุ่งยากโดยเริ่มจากการตรวจหาจุดผิดพลาดของฮาร์ดแวร์ก่อน โดยอาศัยเครื่องมือวัด เช่นลอจิกโปรปหรือออสซิสโลสโคป ตรวจหาจุดเสียที่เกิดขึ้นตามแผนผังของวงจรที่มีอยู่ ถ้าพบว่าอุปกรณ์ใดทำงานบกพร่องก็ทำการเปลี่ยนใหม่ แต่ปัญหาจะยุ่งยากมากถ้าการผิดพลาดที่เกิดขึ้นนั้นมีสาเหตุมาจากซอฟแวร์เพราะจำทำให้สัญญาณลอจิกต่าง ๆ ที่ป้อนเข้าไปควบคุมหรือกระตุ้นให้ฮาร์ดแวร์ทำงานจะผิดพลาดหมด การแก้ไขโดยใช้เครื่องมือทางด้านฮาร์ดแวร์ที่กล่าวผ่านมาข้างต้นทำได้ยากมาก ทั้งนี้เนื่องจากเมื่อไมโครคอมพิวเตอร์ทำการรันโปรแกรมจะเสมือนกับว่าตัวของมันอยู่ในโลกปิดเลยทีเดียว ที่เป็นเช่นนี้เพราะว่าในขณะหนึ่งที่รันโปรแกรมเราจะไม่สามารถทราบได้เลยว่าไมโครโปรเซสเซอร์กำลังทำตามคำสั่งที่ถูกบันทึกไว้ในหน่วยความจำใด เกิดผิดพลาดอะไร ค่าผลลัพธ์ที่ได้ผิดพลาดไปมากน้อยเพียงใด ด้วยเหตุนี้จึงมีการนำเทคนิคของการอีมูลเลท (emulation technique) มาใช้งานโดยอาศัยหลักการง่าย ๆ คือ ทำการสร้างไมโครคอมพิวเตอร์ขึ้นมาอีกชุดหนึ่งที่ทำงานเลียนแบบบไมโครโปรเซสเซอร์ตัวนั้นได้ทุกประการ แต่ในขณะเดียวกันผู้ใช้งานสามารถตรวจสอบการทำงานต่าง ๆ ของโปรแกรมได้ตลอดเวลา

บทที่ 2 ทฤษฎีและหลักการ

2.1 วัตถุประสงค์และขอบเขตของโครงงาน

วัตถุประสงค์ของโครงงานนี้เพื่อต้องการพัฒนาอุปกรณ์ที่เรียกว่า การจำลองการทำงานของไมโครคอมพิวเตอร์แบบชิฟเดียวตระกูล 8051 (emulation 8051) ที่มีราคาถูกขึ้นมา เพื่อใช้เป็นเครื่องมือสำหรับตรวจสอบหรือแก้ไข รวมถึงพัฒนาซอฟแวร์ของระบบไมโครคอมพิวเตอร์แบบชิฟเดียวตระกูล 8051

อีมูลเลเตอร์ที่พัฒนาขึ้นมีขอบเขตในการพัฒนา 2 ส่วนด้วยกันคือ

1. วงจร
2. โปรแกรมควบคุมระบบอีมูลเลเตอร์

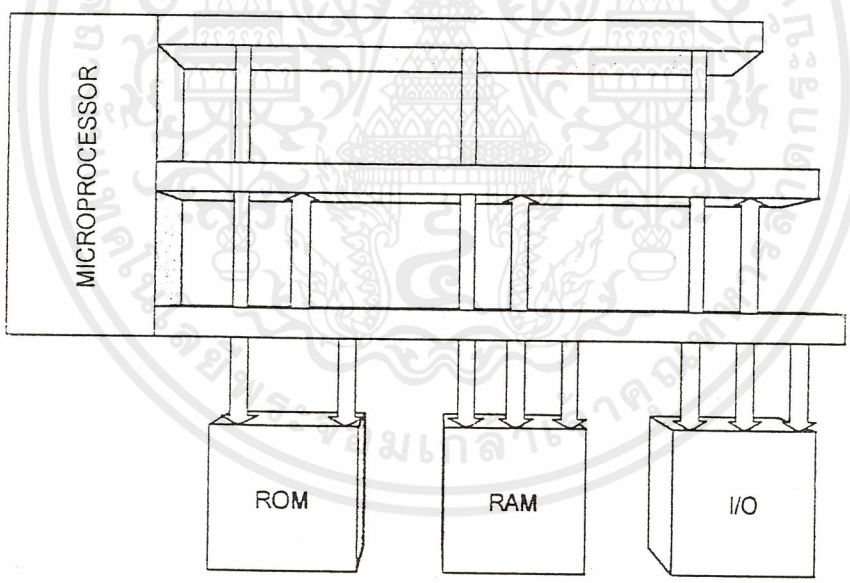
โดยที่อีมูลเลเตอร์นี้จะอาศัยคอมพิวเตอร์ส่วนบุคคล (IBM - PC) เป็นเทอร์มินัล

2.2 โครงสร้างและหลักการทำงานทั่วไปของอีมูลเลเตอร์

ระบบไมโครคอมพิวเตอร์โดยทั่วไปจะประกอบด้วยบัสสามชนิดด้วยกันคือ

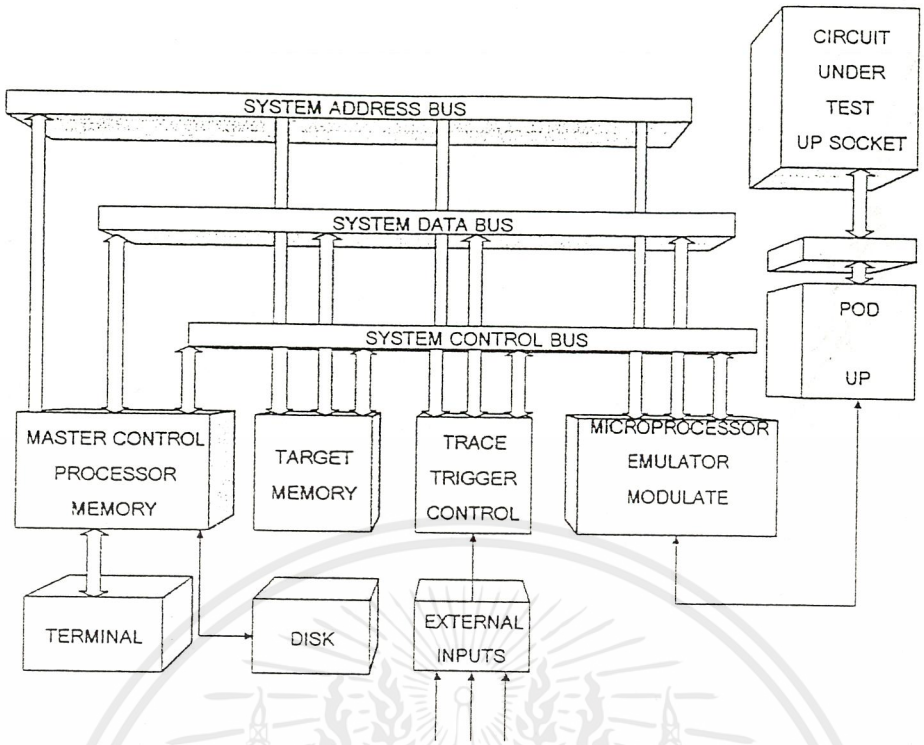
1. แอดเดรสบัส
2. ดาต้าบัส
3. คอนโทรลบัส

โดยบัสทั้ง 3 จะเชื่อมต่อกับอุปกรณ์ต่าง ๆ ดังรูปที่ 2.1



รูปที่ 2.1 แสดงระบบพื้นฐานของไมโครคอมพิวเตอร์ทั่วไป

จากรูปที่ 2.1 ถ้านำสัญญาณจากบัสต่าง ๆ ออกมาภายนอกระบบ และสามารถควบคุมสัญญาณเหล่านี้ได้โดยผ่านทางแป้นพิมพ์และจอภาพ จะสามารถทราบถึงการทำงานของโปรแกรมรวมถึงข้อมูลที่นำเข้าหรือส่งออกไปภายนอกระบบเพื่อควบคุมอุปกรณ์ทางด้านฮาร์ดแวร์ดังกล่าวมาแล้วข้างต้น



รูปที่ 2.2 แสดงแผนผังของการอีมูลเลเตอร์เข้าไปในแผ่นวงจรที่มี
ไมโครคอมพิวเตอร์เป็นส่วนควบคุมหลัก

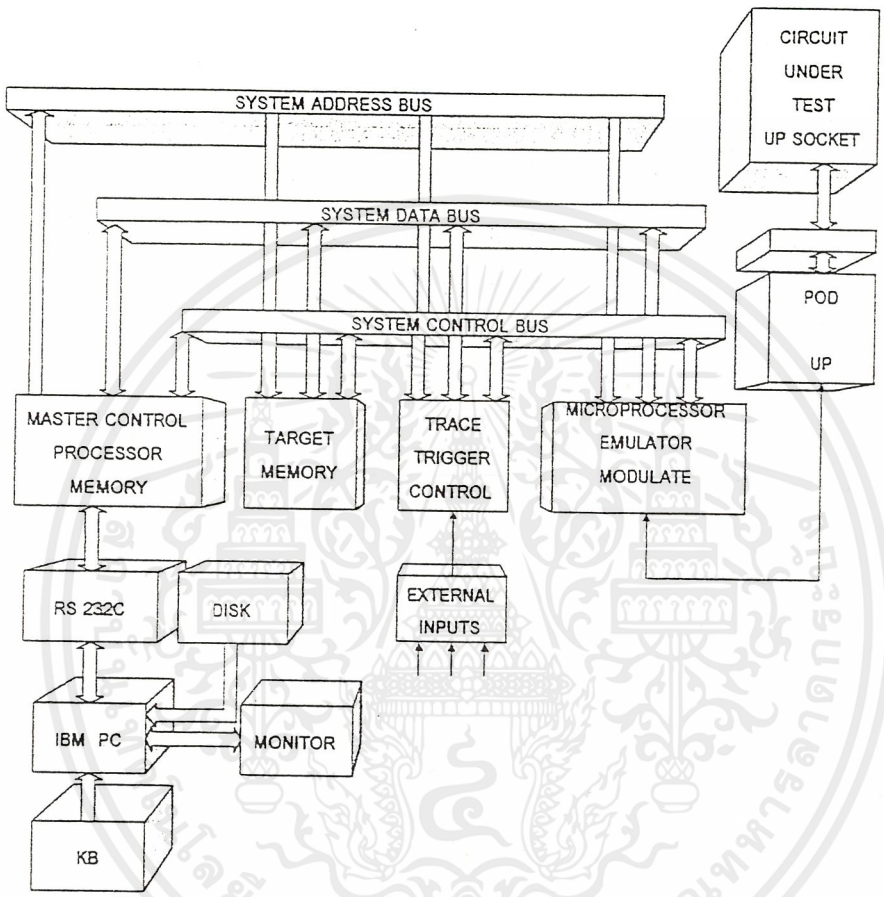
จากรูปที่ 2.2 สมมุติว่าแผ่นวงจรที่จะนำมาตรวจสอบมีโครงสร้างดังที่แสดงไว้ในรูปที่ 2.1 โดยถอดชิฟไมโครโปรเซสเซอร์ออกแล้วใส่สายเชื่อมต่อจากพ็อต (Pod) เข้าไปแทน ดังนั้นสายบัสต่าง ๆ ของแผ่นวงจรจะถูกเชื่อมต่อเข้าไปในระบบของอีมูลเลเตอร์ (emulator) โดยผ่านทางพ็อตและไมโครโปรเซสเซอร์อีมูลเลเตอร์โมดูล (microprocessor emulation module) สำหรับอีมูลเลเตอร์ทั่ว ๆ ไปแล้ว ไมโครโปรเซสเซอร์อีมูลเลชั่นโมดูลนี้จะเปลี่ยนแปลงตามไมโครโปรเซสเซอร์ของแผ่นวงจรที่นำมาตรวจสอบ โมดูลเหล่านี้จะทำหน้าที่แทนไมโครโปรเซสเซอร์ของแผ่นวงจร ดังนั้นข้อมูลต่าง ๆ ที่อยู่บนแผ่นวงจรสามารถย้ายมายังหน่วยความจำของอีมูลเลเตอร์ได้ทันที ทำให้สะดวกในการตรวจสอบแก้ไขผ่านทางเทอร์มินัล ผลที่ได้สามารถเก็บไว้ในดิสก์ หรือ สามารถอ่านโปรแกรมที่บันทึกไว้ในแผ่นดิสก์ลงไปในหน่วยความจำได้เช่นกัน โดยผ่านทางมาสเตอร์คอนโทรลโปรเซสเซอร์ (master control processor) นอกจากนั้นขณะรันโปรแกรมยังสามารถนำสัญญาณภายนอกเข้ามาเก็บไว้เพื่อทำการทดสอบเปรียบเทียบกับโปรแกรมว่าถูกต้องและสอดคล้องกันหรือไม่ตามความสามารถที่ผ่านมาข้างต้น

ในรายงานฉบับนี้เป็นการเสนอการสร้างอีมูลเลเตอร์อย่างง่ายขึ้นมาใช้งาน โดยเลือกเฉพาะใช้กับไมโครคอมพิวเตอร์แบบชิฟเดียวเบอร์ 8051 ของบริษัทอินเทล

2.3 การใช้เครื่อง IBM - PC เป็นเทอร์มินัล

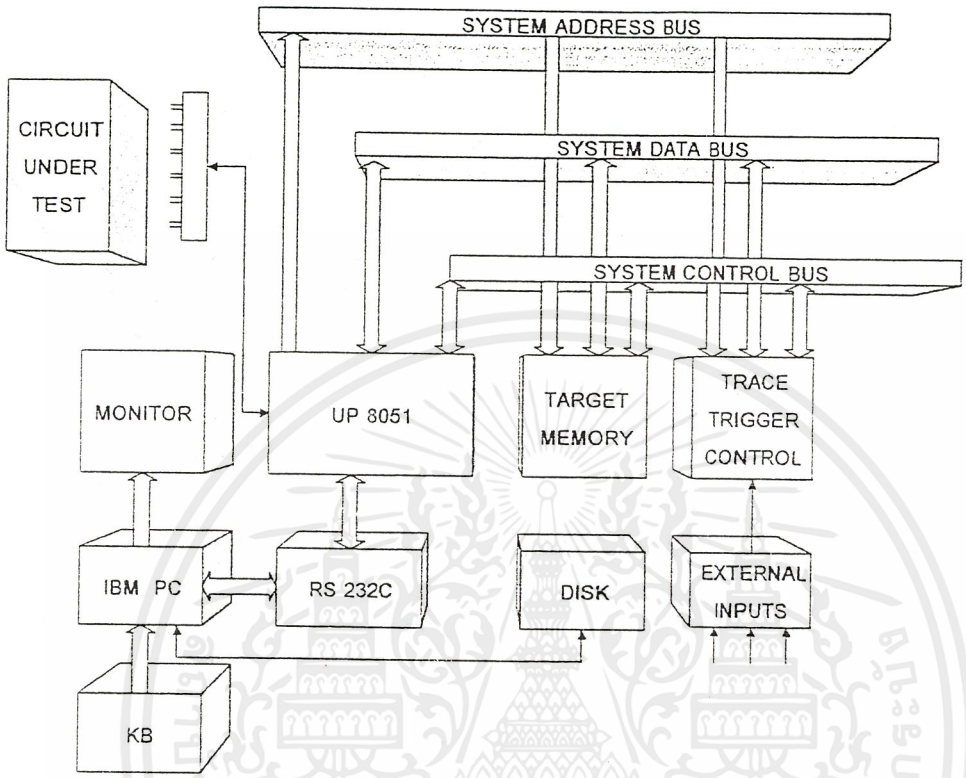
ในหัวข้อนี้จะเป็นการกล่าวถึงการออกแบบสร้างอีมูลเลเตอร์ขึ้นมาใช้งาน โดยเริ่มจากรูปที่ 2.2 ส่วนของเทอร์มินัลจะใช้เครื่องคอมพิวเตอร์ส่วนบุคคล IBM - PC ซึ่งมีตัวขับจานแม่เหล็กอยู่แล้วจึงใช้งานได้เลย

การติดต่อสื่อสารกับอีมูลเลเตอร์กระทำผ่านทางพอร์ตอนุกรม RS - 232 ดังนั้นสามารถเขียนรูปใหม่ ได้ดังรูป 2.3



รูปที่ 2.3 แสดงแผนผังของการอีมูลเลทเข้าไปในแผ่นวงจรโดยใช้ IBM - PC เป็นเทอร์มินัล

เมื่อพิจารณาต่อมาหากตัวอีมูลเลเตอร์ถูกออกแบบให้ใช้งานกับไมโครคอมพิวเตอร์ที่มีไมโครโปรเซสเซอร์เบอร์เดียวกันหมด จะสามารถทำการตัดส่วนของพ็อต และอีมูลเลชั่นโมดูลออกได้ โดยใช้มาสเตอร์คอนโทรลโปรเซสเซอร์เบอร์เดียวกับไมโครโปรเซสเซอร์บนแผ่นวงจร เมื่อกระทำดังนี้บัสของอีมูลเลเตอร์จะเหมือนกับบัสของวงจรที่นำมาตรวจสอบทุกประการ ดังนั้นแผนผังของวงจรจะเขียนได้ดังรูปที่ 2.4



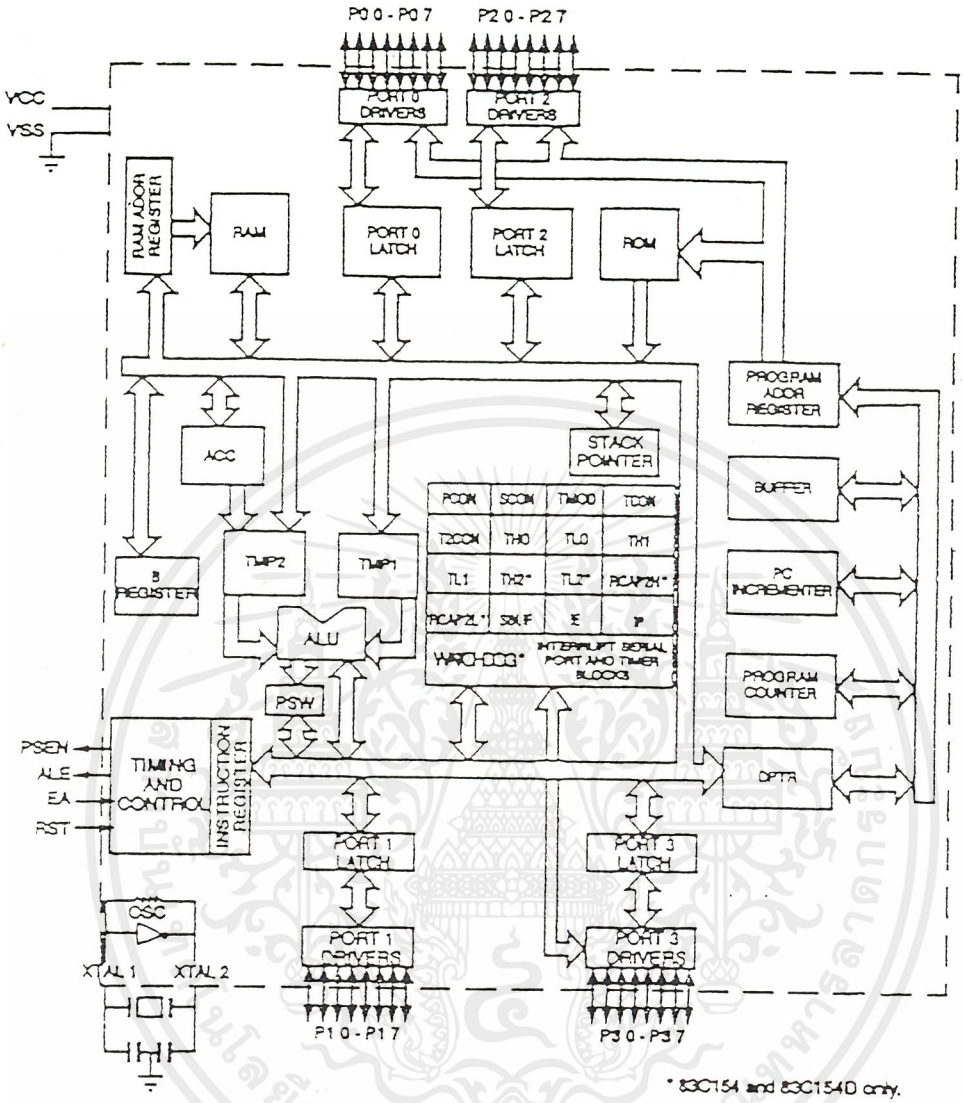
รูปที่ 2.4 แสดงแผนผังของวงจรอีมูลเลเตอร์ที่ทำการออกแบบ

สำหรับการเลือกไมโครโปรเซสเซอร์ที่ใช้ในอีมูลเลเตอร์นั้น จะเลือกใช้ไมโครคอมพิวเตอร์แบบชิฟเดียวเพราะมีการนำไปใช้งานด้านการควบคุมกันเป็นที่แพร่หลาย ไมโครคอมพิวเตอร์แบบชิฟเดียวก็มีหลายบริษัทที่ผลิตออกมา แต่เมื่อพิจารณาถึงความนิยมในการใช้งานจะพบว่า มีการนำเอาไมโครโปรเซสเซอร์ของบริษัทอินเทลมาใช้งานอย่างแพร่หลาย โดยจะเลือกใช้ในตระกูล 8051 (MCS - 51)

ไมโครคอมพิวเตอร์แบบชิฟเดียวตระกูล 8051 เป็นไมโครคอมพิวเตอร์ที่เหมาะสมกับการนำมาใช้งานทางด้านการควบคุมเป็นอย่างมาก เพราะมีความเร็วในการทำงานสูง มีคำสั่งที่จัดการกับข้อมูลอย่างมีประสิทธิภาพ การแลกเปลี่ยนหรืออ้างอิงข้อมูลภายในสามารถกระทำได้อย่างคล่องตัว

2.4 โครงสร้างและการทำงานของไมโครคอมพิวเตอร์แบบชิฟเดียวตระกูล MCS - 51

บล็อกไดอะแกรมของโครงสร้างภายใน MCS - 51 จะเป็นดังรูปที่ 2.5



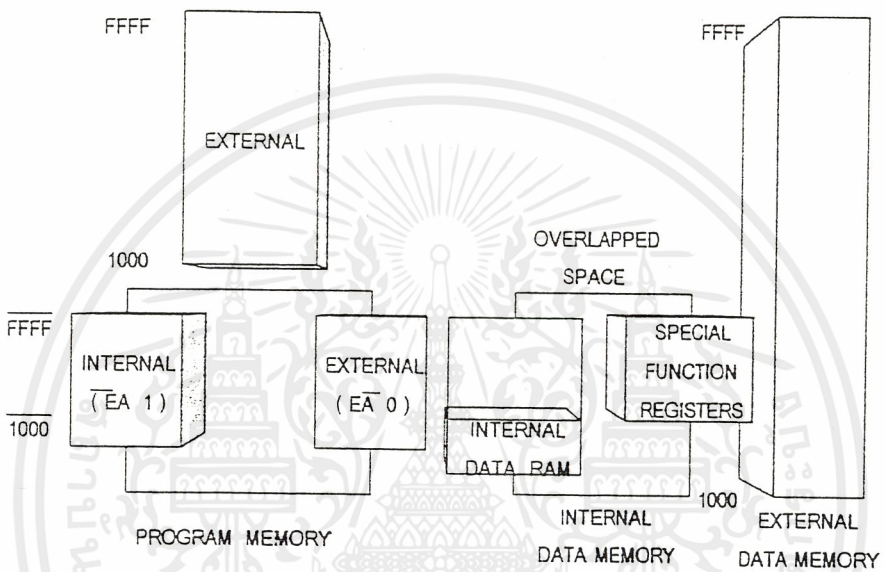
รูปที่ 2.5 แสดงโครงสร้างภายในของไมโครคอมพิวเตอร์แบบชิฟเดียวตระกูล MCS - 51 พร้อมทั้งขาสัญญาณต่าง ๆ

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 จำนวน 2 ชุด
- มีโครงสร้างของการอินเทอร์รัปท์ได้ 5 แห่ง ประกอบด้วยระดับของความสำคัญ 2 ระดับ
- มีพอร์ทอนุกรมที่ทำงานแบบฟลูดูเพล็กซ์ 1 พอร์ท
- สามารถอ้างอิงแอดเดรสแบบบิทได้ในกรณีที่มีการประมวลผลแบบบูลีน (Boolean Processing)

ลักษณะของการแม็พหน่วยความจำจะเป็นดังรูปที่ 2.6



รูปที่ 2.6 แสดงการแม็พของหน่วยความจำ (memory map)

จากรูปที่ 2.6 8051 จะแยกแอดเดรสออกเป็น 2 ส่วน คือ

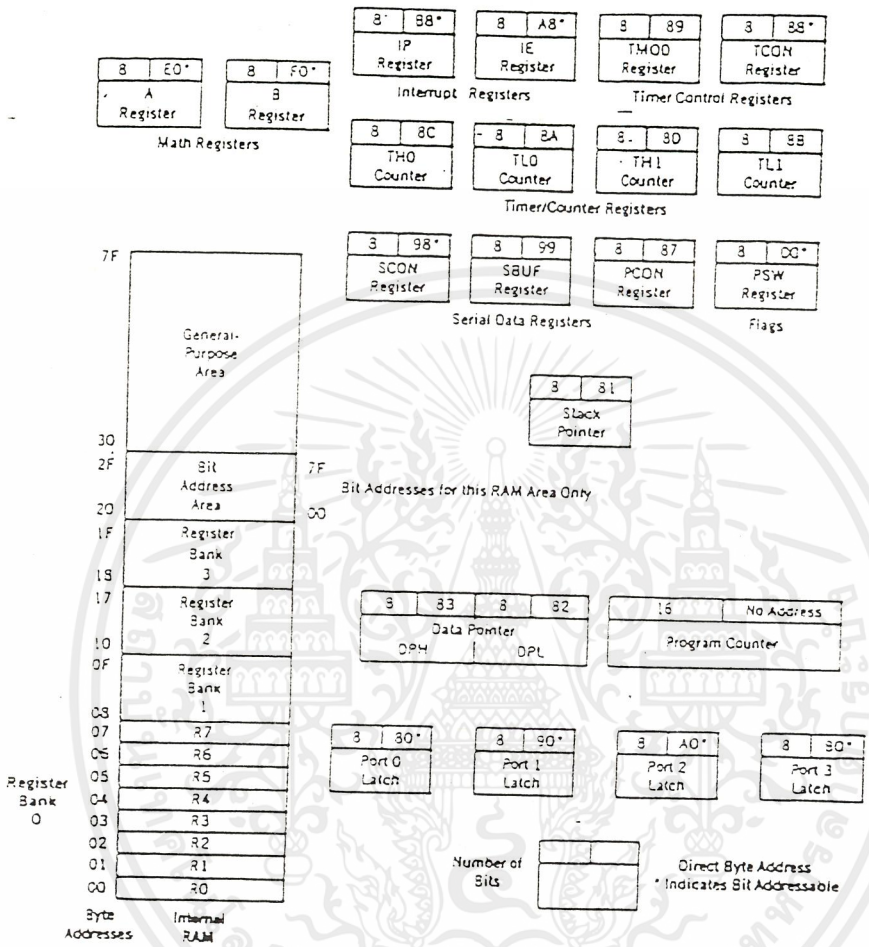
1. หน่วยความจำที่ใช้เก็บโปรแกรม
2. หน่วยความจำที่ใช้เก็บข้อมูล

สำหรับหน่วยความจำที่ใช้เก็บโปรแกรมสามารถมีได้ถึง 64 กิโลไบท์โดยที่ในจำนวน 64 กิโลไบท์นี้อยู่ภายในตัว 8051 จำนวน 4 กิโลไบท์ การอ้างอิงหน่วยความจำ 4 กิโลไบท์นี้จะถูกควบคุมด้วยขา $\overline{EA}$ ถ้าขา $\overline{EA}$ มีลอจิกเป็น 0 จะใช้หน่วยความจำภายนอกทั้งหมด 64 กิโลไบท์ แต่ถ้าเป็นลอจิก 1 จะใช้หน่วยความจำภายใน 4 กิโลไบท์แรก จากนั้นจะใช้หน่วยความจำภายนอกจนกระทั่งครบ 64 กิโลไบท์ ส่วนหน่วยความจำที่ใช้เก็บข้อมูลประกอบด้วยแรมภายในตัวจำนวน 128 ไบท์ พร้อมกับรีจิสเตอร์สำหรับทำหน้าที่พิเศษต่าง ๆ (Special Function Register) อีกจำนวน 21 รีจิสเตอร์ ดังนี้

- ACC Accumulator
- B B Register
- 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 Serital Data Buffer
- PCON Power Control

รายละเอียดของการแบ่งหน่วยความจำที่ใช้เก็บข้อมูลจะเป็นดังรูปที่ 2.7



รูปที่ 2.7 แสดงถึงการแบ่งหน่วยความจำที่ใช้ในการเก็บข้อมูล

จากรูปที่ 2.7 หน่วยความจำภายในที่ใช้เก็บข้อมูลจะถูกแบ่งออกเป็น 2 ส่วน คือ แอดเดรส 0 ถึง 127 ใช้เก็บข้อมูลภายใน และแอดเดรสที่ 128 ถึง 255 จะทำหน้าที่เป็นรีจิสเตอร์ที่ทำหน้าที่พิเศษตามชื่อที่กำหนด และมีตำแหน่งที่แน่นอน เฉพาะส่วนแรกตั้งแต่แอดเดรส 0 ถึง 31 จะถูกแบ่งออกเป็น 4 กลุ่ม ๆ ละ 8 ไบท์ เรียกแต่ละกลุ่มว่ารีจิสเตอร์แบงค์ 0 ถึง 3 เรียงกันไปตามลำดับเช่นกัน หรือจะเรียกใช้งานตามค่าแอดเดรสโดยตรงเลยก็ได้

สำหรับรายละเอียดของรีจิสเตอร์ที่ทำหน้าที่พิเศษต่าง ๆ จะเป็นดังนี้

แอคคิวมูเลเตอร์ เป็นรีจิสเตอร์ที่ใช้เก็บข้อมูลหรือผลลัพธ์ของคำสั่ง

รีจิสเตอร์ บี เป็นรีจิสเตอร์ที่ใช้เฉพาะคำสั่งการคูณและหารเท่านั้น

สแตกพอยเตอร์ เป็นรีจิสเตอร์ขนาด 8 บิทค่าของสแตกจะอยู่ที่ใด ๆ ของหน่วยความจำแรมจำนวน 128 ไบท์ภายในชิฟ เมื่อ 8051 ถูกรีเซตค่าสแตกจะถูกกำหนดเริ่มต้นไว้ที่ 07H

ดาต้าพอยเตอร์ เป็นรีจิสเตอร์ขนาด 16 บิท แยกออกเป็นไบท์สูงและไบท์ต่ำ หน้าที่หลักคือ เอาไว้เก็บค่าแอดเดรสจำนวน 16 บิท

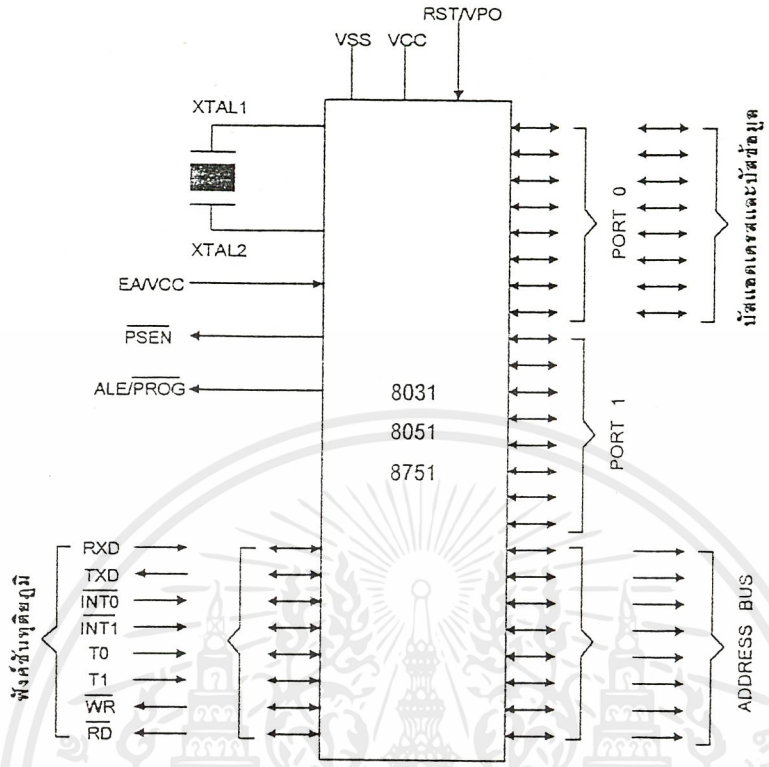
พอร์ท 0 ถึง พอร์ท 3 จะเป็นพอร์ทขนาด 8 บิท ที่สั่งให้เป็นอินพุทหรือเอาท์พุทก็ได้ และแบบที่ค้างสภาวะทางลอจิกไว้ได้โดยสั่งผ่านทางรีจิสเตอร์พิเศษ คือ รีจิสเตอร์ P0 ถึง P3 พอร์ท 0 และพอร์ท 2 ยังเชื่อมต่อกับหน่วยความจำภายนอกด้วย นอกจากนั้น พอร์ท 3 ยังถูกใช้เป็นพอร์ทอนุกรมรับสัญญาณอินเทอร์รัปท์จากภายนอกเป็นไทเมอร์และสัญญาณอ่านหรือเขียนข้อมูลไปยังหน่วยความจำภายนอกด้วยดังนี้

ขาของพอร์ท 3	หน้าที่อื่นนอกจากเป็น I / O
P 3.0	RXD (รับสัญญาณอนุกรม)
P 3.1	TXD (ส่งสัญญาณอนุกรม)
P 3.2	$\overline{\text{INT0}}$ (อินเทอร์รัปท์จากภายนอก)
P 3.3	$\overline{\text{INT1}}$ (อินเทอร์รัปท์จากภายนอก)
P 3.4	T0 (ไทเมอร์ 0 รับสัญญาณจากภายนอก)
P 3.5	T1 (ไทเมอร์ 1 รับสัญญาณจากภายนอก)
P 3.6	$\overline{\text{WR}}$ (สัญญาณเขียนหน่วยความจำภายนอก)
P 3.7	$\overline{\text{RD}}$ (สัญญาณอ่านหน่วยความจำภายนอก)

บัฟเฟอร์ข้อมูลอนุกรม ถูกแบ่งเป็น 2 รีจิสเตอร์ ตัวที่หนึ่งสำหรับส่งข้อมูลออกและอีกตัวหนึ่งสำหรับรับข้อมูลเข้า

รีจิสเตอร์ควบคุมและแสดงสถานะ ได้แก่ รีจิสเตอร์ IP IE TMOD TCON PCON ซึ่งแต่ละตัวจะประกอบด้วยบิทที่ใช้ควบคุมและแสดงสถานะสำหรับการอินเทอร์รัปท์ของระบบไทม์เมอร์และพอร์ทอนุกรม

เพื่อให้เห็นภาพพจน์ของ 8051 ที่เกี่ยวข้องพอร์ทและส่วนต่าง ๆ ได้ชัดเจนขึ้นจะเขียนหน้าที่ของพอร์ทเมื่อทำงานกับหน่วยความจำภายนอกได้ดังรูปที่ 2.8



รูปที่ 2.8 แสดงหน้าที่ของพอร์ทเมื่อเชื่อมต่อกับหน่วยความจำภายนอก

หลังจากทำความเข้าใจกับโครงสร้างภายในแล้ว ลองพิจารณาถึงการใช้งานของสายสัญญาณที่ปรากฏภายนอกบ้างเพื่อให้เข้าใจได้ดีขึ้น

จากโครงสร้างภายในของ MCS - 51 ในรูปที่ 2.5 จะมีบัสสองทิศทาง 4 เส้น และพอร์ทขนาด 8 บิท เมื่อใช้หน่วยความจำภายในตัว ในกรณีที่ไม่ใช้หน่วยความจำภายใน พอร์ท 0 และพอร์ท 2 จะถูกใช้เป็นบัสของข้อมูลและแอดเดรส ดังนั้นพอร์ททั้งสองยังคงใช้งานเป็นอินพุทและเอาท์พุทโดยพอร์ท 0 ทำหน้าที่เป็นสายสัญญาณแอดเดรส A0.....A7 และเป็นบิทข้อมูล D0.....D7 ส่วนพอร์ท2 ใช้งานเป็นสายสัญญาณแอดเดรส A8.....A15

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

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

ขา $\overline{EA}$ เป็นอินพุทที่ใช้ร่วมกับแอดเดรสภายนอกโดยมีค่าลอจิก ‘0’ เมื่อโปรเซสเซอร์อ่านคำสั่งจากหน่วยความจำภายนอก

จะเห็นได้ว่า 8051 มีวงจรนับและตั้งเวลาชนิด 16 บิทอยู่ 2 ตัว เมื่อทำหน้าที่เป็นวงจรตั้งเวลา รีจิสเตอร์ที่ทำหน้าที่ตั้งเวลาจะเพิ่มขึ้นหนึ่งทุก ๆ รอบคำสั่งของเครื่อง และจะนับด้วยอัตราสูงสุดที่ 1 / 12 ของความเร็วสัญญาณนาฬิกาของโปรเซสเซอร์ วงจรนับและตั้งเวลา 0 และ 1 มีวิธีโปรแกรมให้ทำงานได้ต่างกัน 4 แบบ รวมทั้งการทำงานเป็น 8 บิทหรือ 16 บิท และการบรรจุค่าพรีเซ็ตหนึ่งค่าโดยอัตโนมัติ

8051 เป็นชิฟที่มีการอินเตอร์เฟสอนุกรมชนิด 2 ทิศทาง ทำให้รับและส่งข้อมูลพร้อมกันตัวรับข้อมูลชนิดอะซิงโครนัส (Asynchronous Receiver) มีบัฟเฟอร์สำหรับข้อมูลเป็นพิเศษเพิ่มความเร็วในการสื่อสารและสามารถเลือกโปรแกรมใช้งานแบบหนึ่งแบบใดใน 4 แบบ ด้วยการใช้โปรแกรมควบคุมอัตราการส่งข้อมูลและรูปแบบข้อมูล

8051 สามารถรับการอินเตอร์รัปท์ได้ 5 แหล่ง คือ $\overline{\text{INT0}}$ $\overline{\text{INT1}}$ (กำหนดให้ใช้ระดับพัลส์หรือขอบขาพัลส์ก็ได้) และจากวงจรนับและตั้งเวลาที่ 0 ถึง 1 รวมทั้งจากพอร์ทอนุกรมโดยสามารถกำหนดลำดับความสำคัญของการอินเตอร์รัปท์ได้ 2 ระดับ แต่ละแหล่งของอินเตอร์รัปท์จะกำหนดเป็นเวกเตอร์เฉพาะชี้แอดเดรส ดังนั้นเมื่อมีการอินเตอร์รัปท์เข้ามา ตัวโปรเซสเซอร์จะกระโดดไปที่ส่วนของโปรแกรมที่ทำงานตามวัตถุประสงค์ของอินเตอร์รัปท์นั้น หลังจากนั้นข้อมูลต่าง ๆ ของโปรแกรมเคาน์เตอร์ลงในสแตก

2.5 การจัดการหน่วยความจำของ 8051

หน่วยความจำของ 8051 แบ่งออกเป็น 2 แบบ ตามลักษณะการใช้งานคือ

1. หน่วยความจำที่ใช้ในการเก็บโปรแกรม
2. หน่วยความจำที่ใช้ในการเก็บข้อมูล

2.6 หน่วยความจำที่ใช้ในการเก็บโปรแกรมของ 8051

เป็นหน่วยความจำที่ใช้เก็บคำสั่งในรูปรหัสภาษาเครื่องซึ่งต้องการให้ 8051 ทำงานเมื่อ8051ทำงานก็จะอ่านข้อมูลที่เก็บไว้ในหน่วยความจำประเภทนี้เข้าไปถอดรหัสแล้วสร้างสัญญาณควบคุมส่วนอื่น ๆ ตามการทำงานของแต่ละคำสั่งนั้น หน่วยความจำแบบนี้จะต้องเป็นแบบรอม และผู้ที่ใช้ต้องเขียนข้อมูลในแต่ละคำสั่งของหน่วยความจำเป็นรหัสภาษาเครื่อง 8051 ตามลำดับการทำงานที่ต้องการ (หน่วยความจำแบบรอมเมื่อปิดไฟแล้วข้อมูลก็ไม่มีการสูญหาย) การเขียนข้อมูลลงไปในรอมจะต้องใช้เครื่องมือพิเศษในระหว่างการทำงานของ 8051 ผู้ใช้จะไม่สามารถใช้คำสั่งทำการเขียนข้อมูลลงในหน่วยความจำแบบนี้ได้จำนวนตำแหน่งสูงสุดของหน่วยความจำแบบนี้ที่ 8051 จะใช้งานได้คือ 65356 ตำแหน่ง ค่าของตำแหน่งจะเขียนเป็นเลขฐาน 16 ได้ตั้งแต่ 0000H ถึง FFFFH หน่วยความจำตำแหน่ง 0000H ถึง FFFFH จำนวน 4 กิโลไบท์ นั้นผู้ใช้จะเลือกได้ว่าเป็นตำแหน่งของรอมที่อยู่ภายในหรือภายนอก 8051 ถ้าต้องการให้ 8051 ทำงานตามคำสั่งที่เก็บไว้ในรอมภายใน 8051 ก็ให้ป้อนสัญญาณสภาวะลอจิก 1 เข้าที่ขา $\overline{\text{EA}}$ ของ 8051 ส่วนหน่วยความจำที่ตำแหน่ง 1FFFH ถึง FFFFH จะต้องต่ออยู่กับภายนอก 8051 เสมอ

2.7 ลำดับสัญญาณการติดต่อเพื่ออ่านข้อมูลจากอีพรอม

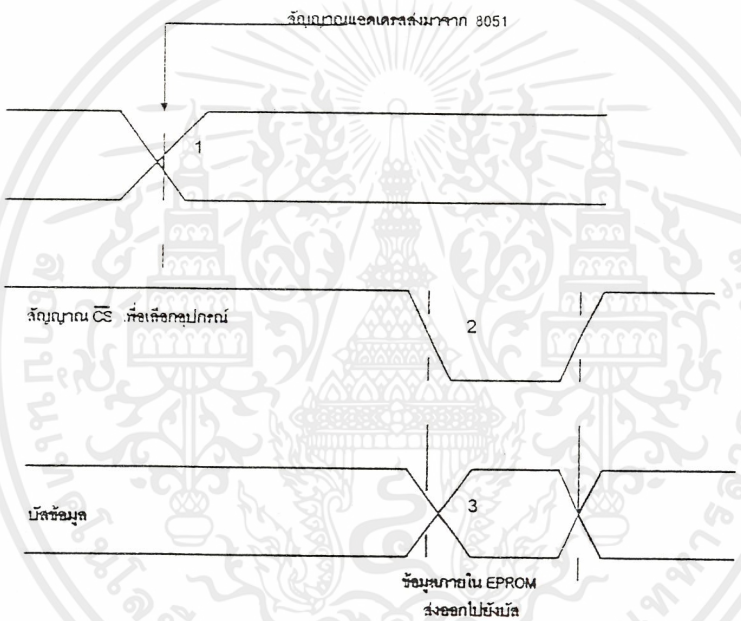
จากแผนภาพเวลาของสัญญาณในรูปที่2.9 แสดงให้เห็นถึงลำดับเหตุการณ์เมื่อมีการอ่านข้อมูลจากอีพรอมดังนี้

1. เริ่มต้นไมโครคอนโทรลเลอร์จะต้องทำการส่งค่าแอดเดรสของหน่วยความจำที่ต้องการอ่านออกมาก่อนเป็นลำดับแรก

2. ในช่วงเวลาถัดมาไมโครคอนโทรลเลอร์จะขับสัญญาณ $\overline{CS}$ ให้เป็นลอจิกต่ำ เพื่อเลือกให้อีพรอมมีการทำงานขึ้น หลังจากนั้นจะหยุดรอระยะเวลาช่วงหนึ่งประมาณ 100 ถึง 300 นาโนวินาทีตามประเภทของอีพรอมที่ใช้งาน ช่วงเวลานี้เรียกว่า แอสเซสไทม์ (access time) ทั้งนี้เพื่อเป็นการให้อีพรอมทำการถอดรหัสแอดเดรสที่รับเข้าและเตรียมพร้อมในการส่งข้อมูลออกมา

3. ไมโครคอนโทรลเลอร์จะขับสัญญาณ $\overline{OE}$ เป็นระดับลอจิกต่ำ เพื่อสั่งงานให้มีการนำข้อมูลภายในอีพรอมส่งออกมายังบัสข้อมูล จากนั้นไมโครคอนโทรลเลอร์จึงจะทำการรับข้อมูลภายในบัสเก็บเข้ายังรีจิสเตอร์ภายในต่อไป

4. ในช่วงเวลาต่อมาไมโครคอนโทรลเลอร์จะต้องทำการขับสัญญาณ $\overline{OE}$ ให้กลับไปเป็นระดับลอจิกสูงเช่นเดิม เพื่อเป็นการทำให้สภาวะขาสัญญาณบัสข้อมูลของอีพรอมมีสภาวะเป็นอิมพิแดนซ์สูง และไม่มีผลต่อการทำงานอื่น ๆ



รูปที่ 2.9 แสดงแผนภาพเวลาของการอ่านข้อมูลจากตำแหน่งภายในหน่วยความจำอีพรอม

1. ได้รับสัญญาณแอดเดรสซึ่งถูกส่งออกมาจาก CPU
2. สัญญาณเลือกอุปกรณ์ส่งมาให้กับอีพรอมเพื่อให้ทำงาน
3. ข้อมูลภายในอีพรอมถูกนำออกมายังบัสข้อมูล

2.8 การเชื่อมต่อหน่วยความจำโปรแกรมภายนอก 8051

เนื่องจากระบบบัสแอดเดรสและบัสข้อมูลของไมโครคอนโทรลเลอร์ 8051 เป็นลักษณะแบบใช้การมัลติเพล็กซ์จากพอร์ทเดียวกัน กล่าวคือ ในระยะเวลาเริ่มต้น เส้นสัญญาณเหล่านี้ของพอร์ทจะใช้ในการส่งค่าแอดเดรสของตำแหน่งที่ต้องการติดต่อด้วย ในช่วงระยะเวลาต่อมาจึงเปลี่ยนไปเป็นสภาวะอิมพิแดนซ์สูงเพื่อใช้งานในฐานะของบัสข้อมูล แต่เนื่องจากว่าอีพรอมที่ใช้งานกันทั่วไปนั้นไม่ใช้การมัลติเพล็กซ์ และมี

ขาสัญญาณบัสแอดเดรสและบัสข้อมูลแยกออกจากกันโดยชัดเจนดังนั้นการเชื่อมต่ออีพรอมเพื่อทำหน้าที่เป็นหน่วยความจำโปรแกรม จึงจำเป็นต้องมีวงจรประเภทแลตช์ (latch) ประกอบเพิ่มเติมขึ้น เพื่อทำการค้างค่าแอดเดรสที่ส่งออกมาจาก 8051 ในช่วงเวลาแรกให้กับขาสัญญาณแอดเดรสของอีพรอมต่อไป

จากตารางในรูปที่ 2.10 แสดงให้เห็นถึงสัญญาณต่าง ๆ ของ 8051 ซึ่งนำมาใช้การติดต่อกับหน่วยความจำภายนอก

สัญญาณ	คำจำกัดความ	ขาสัญญาณ	หน้าที่
$\overline{EA}$	External Access	31	เลือกประเภทหน่วยความจำภายในหรือภายนอก
ALE	Address Enable	30	สัญญาณเอาท์พุทสำหรับการแลตช์ข้อมูลแอดเดรสจากบัส
P 2.0 - P 2.3	Port 2	21 - 28	เป็นข้อมูลแอดเดรสไบท์สูงของหน่วยความจำ
P 0.0 - P 0.7	Port 0	39 - 32	มัลติเพล็กซ์สัญญาณของบัสแอดเดรสและบัสข้อมูล
$\overline{PSEN}$	Program Store Enable	29	สัญญาณระบุการอ่านให้กับหน่วยความจำอีพรอม

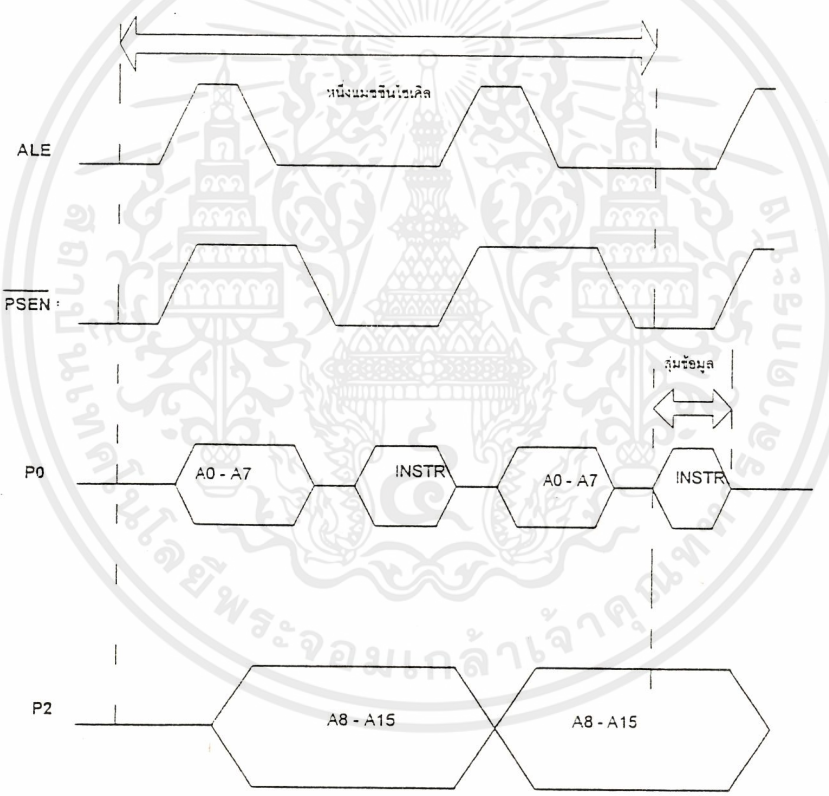
รูปที่ 2.10 แสดงสัญญาณของ 8051 ที่ใช้ระหว่างการติดต่อเพื่ออ่านข้อมูลจากหน่วยความจำภายนอก

สัญญาณ $\overline{EA}$ (External Access) ใช้ในการกำหนดเลือกว่า จะอ่านข้อมูลมาจากหน่วยความจำโปรแกรมภายนอกหรือภายในตัวไอซี ซึ่งหากเป็นระดับลอจิกต่ำจะอ่านข้อมูลมาจากหน่วยความจำโปรแกรมภายนอก สิ่งที่ควรให้ความสังเกตคือ เมื่อมีการอ่านข้อมูลจากหน่วยความจำโปรแกรมภายใน และมีการใช้งานแอดเดรสที่อยู่ในช่วงที่สูงเกินกว่าค่าสูงสุดของหน่วยความจำภายใน เช่น หากเป็นเบอร์ 8051 ซึ่งมีหน่วยความจำโปรแกรมภายในสูงสุดเพียง 4 กิโลไบท์ แต่มีการอ้างอิงตำแหน่งที่สูงเกินกว่า 4 กิโลไบท์ เป็นต้น กรณีเช่นนี้ 8051 จะทำการอ่านค่าแอดเดรสที่สูงเกินกว่านี้มาจากหน่วยความจำโปรแกรมภายนอกโดยอัตโนมัติ

จากแผนภาพเวลาการติดต่อกับหน่วยความจำโปรแกรมภายนอกของ 8051 ในรูปที่ 2.11 จะเห็นได้ว่าภายในช่วงเวลาของแมชชีนไซเคิลหนึ่ง ๆ นั้น 8051 มีการไปนำข้อมูลมาถึง 2 ครั้งด้วยกัน ดังนั้นการทำงานของ 8051 จึงจะไปอ่านข้อมูลโปรแกรมมาเป็นจำนวนทวีคูณเสมอ ในช่วงเวลาเริ่มต้นของการติดต่อกับหน่วยความจำภายนอก พอร์ท 0 จะเป็นค่าแอดเดรสไบต์ต่ำ (A0 - A7) และเวลาต่อมาจึงจะเป็นบัสข้อมูล

การส่งค่าของแอดเดรสไบท์ต่ำนี้จะอยู่ราวช่วงเวลาขอบขาลงของสัญญาณ ALE และจะยังคงอยู่จนเมื่อสัญญาณ $\overline{\text{PSEN}}$ เปลี่ยนไปเป็นระดับสัญญาณลอจิกต่ำ ดังนั้นการออกแบบจึงมักใช้สัญญาณ ALE นี้ในการทำให้ไอซีแลตซ์ภายนอกค้างระดับสัญญาณแอดเดรสเหล่านี้ไว้ ส่วนสัญญาณ $\overline{\text{PSEN}}$ จะใช้ในการเลือกให้ไอซีอีพรอมทำงานและอ่านค่าข้อมูลกลับมา

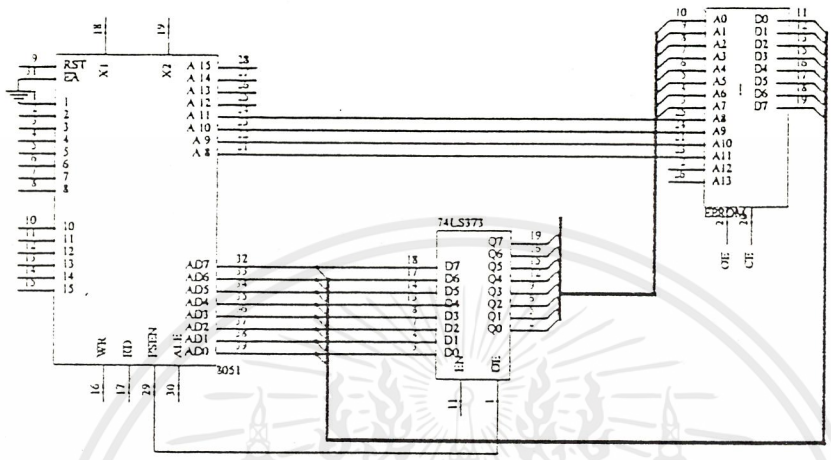
ขณะเมื่อสัญญาณ $\overline{\text{PSEN}}$ เป็นระดับลอจิกต่ำ อีพรอมก็จะทำการถอดรหัสค่าแอดเดรสและส่งข้อมูลที่อยู่ในตำแหน่งนั้นออกมา โดยสัญญาณ $\overline{\text{PSEN}}$ นี้จะค้างสภาวะการเป็นลอจิกต่ำไว้ช่วงเวลาหนึ่งเพื่อให้ข้อมูลถูกส่งออกมาจากอีพรอมเรียบร้อยแล้ว จึงค่อยกลับไปเป็นระดับลอจิกสูงเช่นเดิม และในช่วงเวลาขาขึ้นของสัญญาณ $\overline{\text{PSEN}}$ นี้เองที่ 8051 ทำการสุ่มอ่านข้อมูลเข้ามาสำหรับข้อมูลทางพอร์ท 2 ซึ่งเป็นค่าแอดเดรสไบท์สูง (A8 - A15) นั้นจะถูกส่งออกมาในราวช่วงกึ่งกลางระหว่างที่สัญญาณ ALE เป็นลอจิกสูง ซึ่งก็เป็นเวลาใกล้เคียงกับการส่งค่าแอดเดรสไบท์ต่ำออกมาทางพอร์ท 0 สำหรับค่าแอดเดรสพอร์ท 2 นั้นจะค้างอยู่ตลอดช่วงรอบเวลาของการติดต่อกับหน่วยความจำโปรแกรม



รูปที่ 2.11 แสดงแผนภาพสัญญาณเวลาแสดงการติดต่อกับหน่วยความจำภายนอก

การใช้งานไมโครคอนโทรลเลอร์ 8051 แบบที่ไม่มีหน่วยความจำโปรแกรมภายใน จำเป็นจะต้องเชื่อมต่อเข้ากับหน่วยความจำโปรแกรมซึ่งเป็นไอซีอีพรอม และจะต้องกำหนดให้เริ่มต้นที่แอดเดรส 0000H เสมอ ทั้งนี้เพราะเมื่อมีการรีเซตหรือเริ่มต้นการจ่ายไฟฟ้าให้กับระบบ 8051 จะได้เริ่มต้นทำงานตามคำสั่งที่แอดเดรสนี้ทันที

จากวงจรดังในรูปที่ 2.12 แสดงให้เห็นถึงวิธีการเชื่อมต่อหน่วยความจำโปรแกรมภายนอกให้กับระบบ 8051 ด้วยไอซีหน่วยความจำอีพรอมขนาด 8K * 8 บิตเบอร์ 2764 ในรูปนี้ใช้สัญญาณ CS เชื่อมต่อโดยตรงเข้ากับขาสัญญาณ $\overline{PSEN}$ ของ 8051 ซึ่งจะมีผลทำให้ไม่ว่าจะทำคำสั่งใด ๆ ที่เกี่ยวข้องกับหน่วยความจำโปรแกรมแล้ว ก็จะเป็นการเลือกให้อีพรอมนี้ทำงานได้ตลอดเวลา



รูปที่ 2.12 แสดงการเชื่อมต่อหน่วยความจำภายนอกเข้ากับ 8051

2.9 หน่วยความจำที่ใช้ในการเก็บข้อมูล

หน่วยความจำที่ 8051 จะใช้สำหรับพักเก็บข้อมูล แล้วเรียกมาใช้ใหม่ในระหว่างการทำงานของ 8051 การอ่านหรือเขียนข้อมูลจากหน่วยความจำจะกระทำโดยคำสั่งที่เก็บไว้ในหน่วยความจำที่ใช้ในการเก็บโปรแกรม หน่วยความจำแบบนี้จะเป็นแบบรอม ถ้ามีไฟเลี้ยงอยู่ข้อมูลที่เก็บไว้จะไม่สูญหาย แต่ถ้าปิดเครื่องหรือไม่จ่ายไฟให้แก่แรมแล้ว ข้อมูลในแรมก็จะสูญหายไป การสูญหายของข้อมูลไม่ได้หมายความว่าไม่มีอะไรอยู่เลยแต่เป็นการที่มีข้อมูลใหม่ซึ่งไม่ใช่ข้อมูลที่เก็บไว้เดิมเข้ามาแทนที่ หน่วยความจำแบบอ้างอิงของ 8051 จะมีอยู่ 2 ชุด ชุดหนึ่งอยู่ภายใน 8051 จำนวน 128 ไบท์ที่ตำแหน่ง 00H ถึง 7FH และอีกชุดหนึ่งจะต้องต่ออยู่ภายนอกของวงจรรวม 8051 มีได้สูงสุด 65536 ไบท์ (64 กิโลไบท์)

2.10 หน่วยความจำข้อมูลภายนอก

2.10.1 ลักษณะสมบัติของหน่วยความจำแรมแบบสแตติก

กระบวนการนำข้อมูลเข้าไปจัดเก็บไว้ภายในหน่วยความจำแรมจะเรียกว่า การเขียน (Write) ไปยังหน่วยความจำ และเมื่อมีการนำข้อมูลที่จัดเก็บไว้นั้นออกมา ก็จะเรียกว่า การอ่าน (Read) จากหน่วยความจำ ซึ่งกระบวนการทั้งสองนี้จะต้องกระทำภายในช่วงรอบเวลาที่กำหนดไว้เช่นกัน สำหรับลักษณะสมบัติของหน่วยความจำแรม มีความคล้ายคลึงกับหน่วยความจำอีพรอมมาก รวมทั้งประเภทของสัญญาณติดต่อ คือ บัสแอดเดรส บัสข้อมูล และกลุ่มสัญญาณควบคุมทั้งหมดเพียงแต่มีการเพิ่มสัญญาณ $\overline{WR}$ เพื่อระบุถึงการนำข้อมูลจากบัสข้อมูลลงไปเก็บยังหน่วยความจำตำแหน่งที่ได้ระบุแอดเดรสมาเท่านั้น

A col Address - Column (LSB)

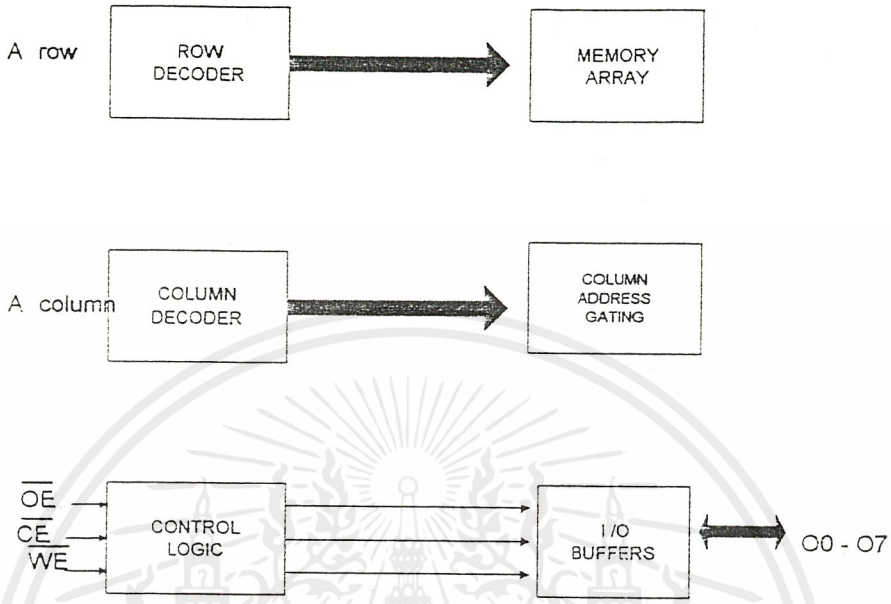
A row Address - Row

$\overline{CE}$ Chip Enable

$\overline{WE}$ Write Enable

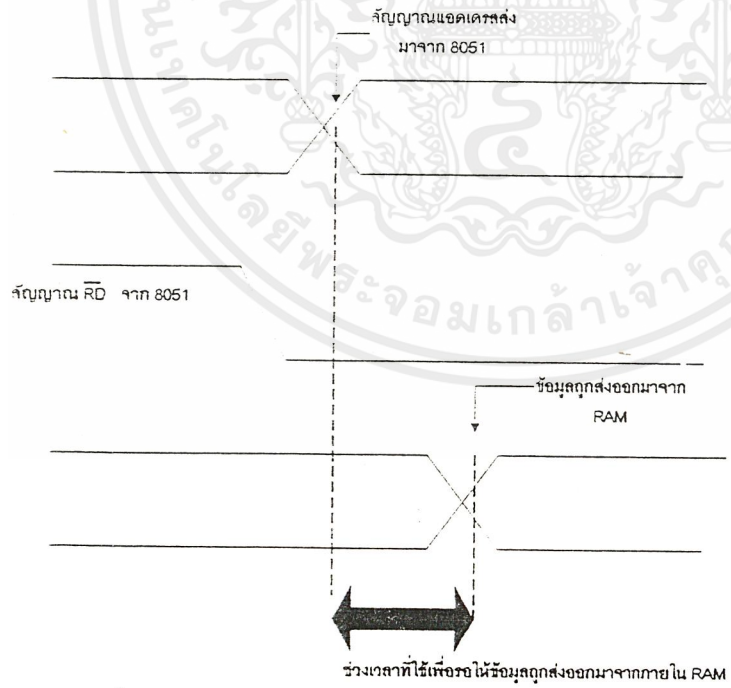
$\overline{OE}$ Output Enable

O0 - O7 Output



รูปที่ 2.13 แสดงแผนภาพสัญญาณของหน่วยความจำแรม

2.10.2 ลำดับสัญญาณการติดต่อเพื่ออ่านข้อมูลจากหน่วยความจำแรม



รูปที่ 2.14 แสดงแผนภาพเวลาลำดับเหตุการณ์การอ่านข้อมูลจากหน่วยความจำแรม

จากแผนภาพเวลาของสัญญาณในรูปที่ 2.14 แสดงถึงลำดับเหตุการณ์เมื่อมีการอ่านข้อมูลจากหน่วยความจำแรมดังนี้

1. ไมโครคอนโทรลเลอร์จะต้องเริ่มต้นด้วยการส่งค่าแอดเดรสของหน่วยความจำที่ต้องการออกมาทางบัสแอดเดรส

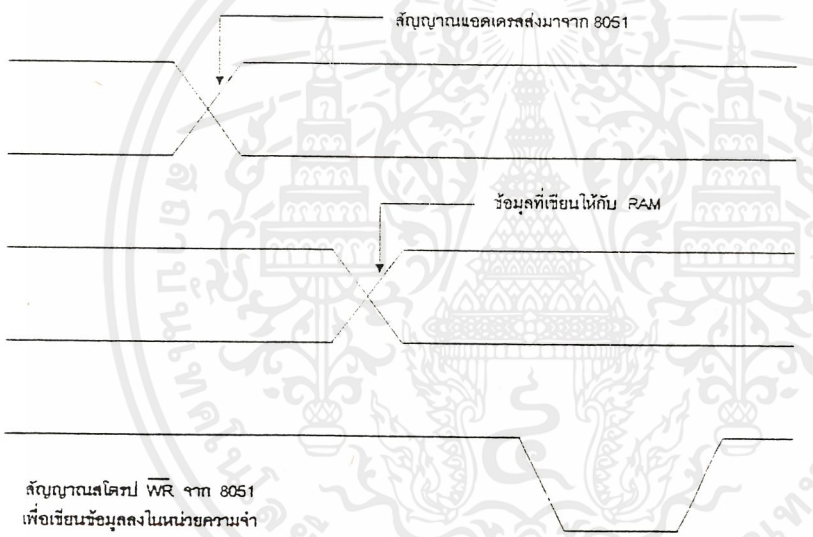
2. สัญญาณ $\overline{OE}$ จะต้องถูกเปลี่ยนให้เป็นระดับลอจิกต่ำ เพื่อระบุว่าต้องการอ่านข้อมูลจากหน่วยความจำ

3. หลังจากนั้นไมโครคอนโทรลเลอร์จะต้องหยุดรอในราวช่วงระยะเวลาหนึ่ง (Read Access time) เพื่อให้วงจรภายในแรมทำการถอดรหัสแอดเดรสและอ่านข้อมูล

4. หลังจากสิ้นสุดเวลาในข้อ 3 แล้วข้อมูลจะถูกส่งออกมาทางขาสัญญาณของบัสข้อมูลและไมโครคอนโทรลเลอร์สามารถอ่านข้อมูลนี้ไปประมวลผลต่อไป

2.10.3 ลำดับสัญญาณการติดต่อเพื่อเขียนข้อมูลยังหน่วยความจำแรม

สำหรับแผนภาพเวลาของสัญญาณในรูปที่ 2.15 แสดงให้เห็นถึงลำดับเหตุการณ์เมื่อมีการเขียนข้อมูลลงในหน่วยความจำแรมดังนี้



รูปที่ 2.15 แสดงแผนภาพเวลาลำดับเหตุการณ์การเขียนข้อมูลลงในหน่วยความจำแรม

1. ช่วงเริ่มต้นไมโครคอนโทรลเลอร์จะต้องส่งค่าแอดเดรสของหน่วยความจำที่ต้องการออกมาทางบัสแอดเดรส

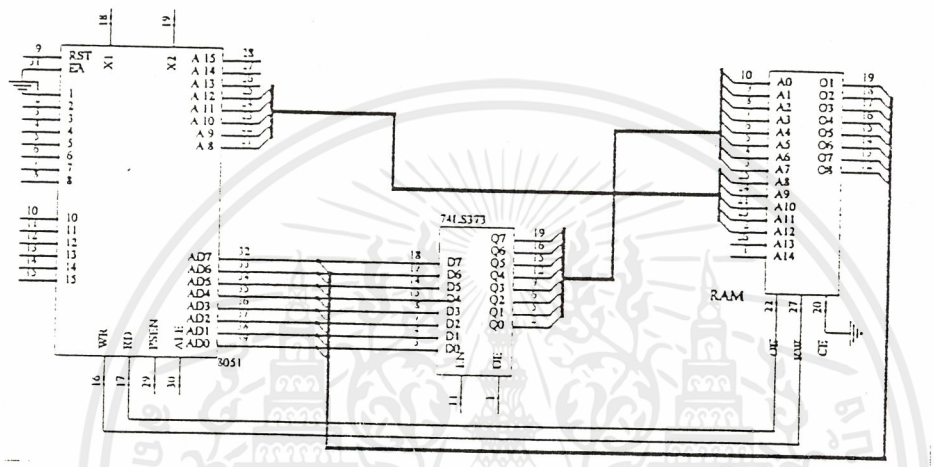
2. เวลาถัดไปในบัสข้อมูลจะเป็นข้อมูลที่ต้องการเขียนลงในหน่วยความจำ

3. ไมโครคอนโทรลเลอร์จะต้องทำการหยุดรอในช่วงระยะเวลาหนึ่งเช่นกัน (write access time) เพื่อให้วงจรภายในแรมทำการถอดรหัสแอดเดรสและเตรียมการเขียนข้อมูล

4. เมื่อสิ้นสุดเวลาที่หยุดรอไมโครคอนโทรลเลอร์จะต้องขับสัญญาณ $\overline{WR}$ ให้เป็นระดับลอจิกต่ำหรือพัลส์ เพื่อให้ข้อมูลภายในบัสข้อมูลถูกนำไปยังตำแหน่งที่ต้องการภายในหน่วยความจำแรมต่อไป

2.10.4 การเชื่อมต่อกับหน่วยความจำข้อมูลภายนอก

การเชื่อมต่อหน่วยความจำแรมเข้ากับระบบของไมโครคอนโทรลเลอร์ 8051 จะใช้วิธีการเหมือนกับการเชื่อมต่อกับหน่วยความจำอีพรอมโดยในรูปที่ 2.16 แสดงให้เห็นถึงวงจรของการเชื่อมต่อเข้ากับหน่วยความจำแรมมีหลักการทำงานดังที่ได้กล่าวมา ไอซี 74LS373 ทำหน้าที่ในการค้างหรือแลตช์ค่าแอดเดรสให้กับอินพุทของหน่วยความจำแรม สำหรับขาสัญญาณ $\overline{OE}$ (หรือ $\overline{RD}$) จะต่อเข้ากับขาสัญญาณ $\overline{RD}$ (P3.7) และขาสัญญาณ $\overline{WR}$ (P3.6) ของ 8051 ในที่นี้เนื่องจากว่าภายในวงจรนี้มีไอซีหน่วยความจำเพียงตัวเดียวจึงได้ทำการต่อสัญญาณ CS กับสัญญาณกราวด์โดยตรงดังรูป



รูปที่ 2.16 แสดงการเชื่อมต่อหน่วยความจำข้อมูลภายนอกกับ 8051

ในรูปที่2.17 แสดงให้เห็นถึงแผนภาพเวลาของสัญญาณที่เกี่ยวข้องดังนี้ เมื่อเริ่มการติดต่อกับหน่วยความจำข้อมูลภายนอก จะมีการส่งค่าของแอดเดรสไบท์ต่ำของตำแหน่งหน่วยความจำที่ต้องการออกมายังพอร์ท 0 ในราวก่อนช่วงเวลาขอบขาลงของสัญญาณ ALE ซึ่งจะนำไปใช้ในการควบคุมไอซีแลตช์ เพื่อทำการค้างค่าแอดเดรสไบท์ต่ำไว้ ดังนั้นในเวลาต่อมาพอร์ท 0 ก็จะว่างและพร้อมที่จะนำไปใช้ในฐานะของบัสข้อมูล ส่วนค่าของแอดเดรสไบท์สูง จะถูกส่งออกมาทางพอร์ท 2 ราวเวลาประมาณกึ่งกลางระหว่างที่สัญญาณ ALE เป็นระดับลอจิกสูง เมื่อ 8051 ทำการขับสัญญาณ $\overline{RD}$ หรือ $\overline{WR}$ ให้เป็นระดับลอจิกต่ำก็จะมีผลทำให้เกิดการส่งและรับข้อมูลระหว่างหน่วยความจำแรมกับบัสข้อมูลขึ้นได้

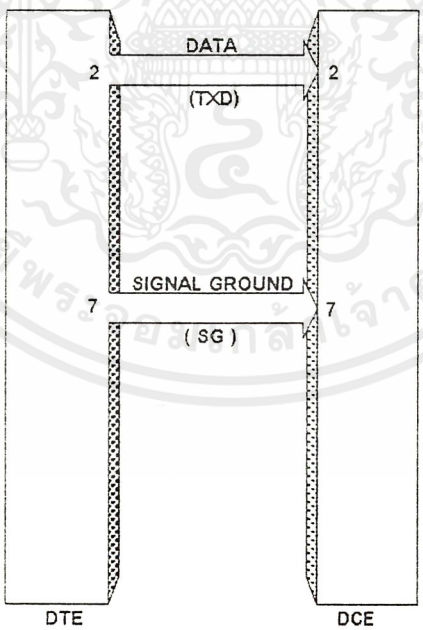
เพื่อป้องกันไม่ให้อุปกรณ์ส่งข้อมูลบนสายเส้นเดียวกัน อุปกรณ์จึงถูกแบ่งออกเป็น 2 ชนิด เช่น อุปกรณ์อย่างเช่นเทอร์มินัลซึ่งใช้สายเส้นที่ 2 สำหรับเอาท์พุทเรียกว่า ดีทีอี (DTE Data Terminal Equipment) อุปกรณ์อย่างเช่นโมเด็มซึ่งใช้สายเส้นที่ 2 สำหรับอินพุทเรียกว่า ดีซีอี (DCE Data Communication Equipment)

2.11.1 อุปกรณ์ DTE และ DCE

ตามมาตรฐาน RS 232 อุปกรณ์ DTE ควรใช้หัวต่อตัวผู้และอุปกรณ์ DCE ควรใช้หัวต่อตัวเมียเมื่อทราบอุปกรณ์หนึ่งเป็น DTE และอีกตัวหนึ่งเป็น DCE ในทางทฤษฎีแล้วสามารถเชื่อมต่อได้อย่างง่ายดาย โดยการเชื่อมต่อสายที่มีหมายเลขตรงกัน เรียกว่าการเชื่อมต่อแบบตรงไปตรงมา แต่มีผู้ผลิตบางรายที่ไม่ได้ทำตามมาตรฐานและทำให้เกิดปัญหา โดยจะถือว่าอุปกรณ์หนึ่งเป็น DTE และอีกตัวหนึ่งเป็น DCE และแต่ละฝ่ายส่งสัญญาณที่อีกฝ่ายต้องการบนสายที่ตรงกัน

2.11.2 การสื่อสารทางเดียว

วงจรหลักที่ถูกใช้สำหรับการสื่อสารมีอยู่ 3 วงจร คือสายเส้นที่ 2 สำหรับข้อมูลจาก DTE ไปยัง DCE สายเส้นที่ 3 สำหรับข้อมูลจาก DCE ไปยัง DTE และสายเส้นที่ 7 สำหรับซิกเนลกราวด์ (signal ground) ซึ่งเป็นจุดอ้างอิงร่วมสำหรับขั้วและแรงดันไฟฟ้าของสายอื่น ๆ ในกรณีที่ง่ายที่สุดซึ่งมีเพียงอุปกรณ์หนึ่งส่งและอีกตัวรับใช้สายเพียง 2 เส้นก็เพียงพอคือสายเส้นที่ 2 หรือ 3 และสายเส้นที่ 7 ดังในรูปที่ 2.18



รูปที่ 2.18 การเชื่อมต่อทางเดียวอย่างง่าย

2.11.3 ฮาร์ดแวร์แฮนด์เช็คกิ้ง

อุปกรณ์ฝ่ายส่งจำเป็นต้องรู้ว่าอุปกรณ์ฝ่ายรับพร้อมที่จะรับข้อมูลหรือไม่ ดังนั้นทางฝ่ายรับต้องมีข่าวสารถูกส่งกลับจากอุปกรณ์ฝ่ายรับไปยังฝ่ายส่ง เพื่อแจ้งว่ามันพร้อมหรือไม่ ข่าวสารนี้เรียกว่าโฟลว์คอนโทรล

จาก DTE ไปสู่ DCE

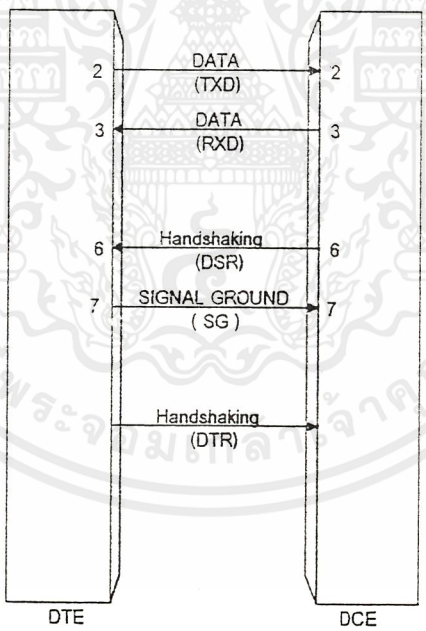
เมื่ออุปกรณ์ DTE ส่งข้อมูลไปที่อุปกรณ์ DCE ข้อมูลถูกส่งไปตามสายเส้นที่ 2 และใช้สายเส้นที่ 7 ซิกแนลกราวนด์ ตามปกติอุปกรณ์ DCE ควบคุมการส่งแฮนด์เช็คกิ้งจากอุปกรณ์ DTE บนสายเส้นที่ 6 ถ้าเครื่องพิมพ์เป็น DCE และคอมพิวเตอร์เป็น DTE สายเส้นที่ 6 บนคอมพิวเตอร์ควรถูกเชื่อมต่อเข้ากับสายเส้นที่ 6 บนเครื่องพิมพ์และเครื่องพิมพ์จะรักษาระดับแรงดันไฟฟ้าบวกบนสายเส้นที่ 6 ตราบเท่าที่มันสามารถรับข้อมูล เมื่อมันต้องการที่จะบอกคอมพิวเตอร์ให้หยุดการส่งข้อมูล มันจะลดแรงดันไฟฟ้าบนสายเส้นที่ 6 ให้เป็นสถานะลบ

จาก DCE ไปสู่ DTE

เมื่ออุปกรณ์ DCE จะสื่อสารกับอุปกรณ์ DTE สายเส้นที่ 3 ต้องถูกใช้สำหรับการส่งผ่านข้อมูลและถ้าต้องการแฮนด์เช็คกิ้ง จะใช้สายเส้นที่ 20 เพื่อส่งแฮนด์เช็คกิ้งจากอุปกรณ์ DTE ไปยังอุปกรณ์ DCE สายเส้นที่ 20 มีชื่อว่า DTR (data terminal ready)

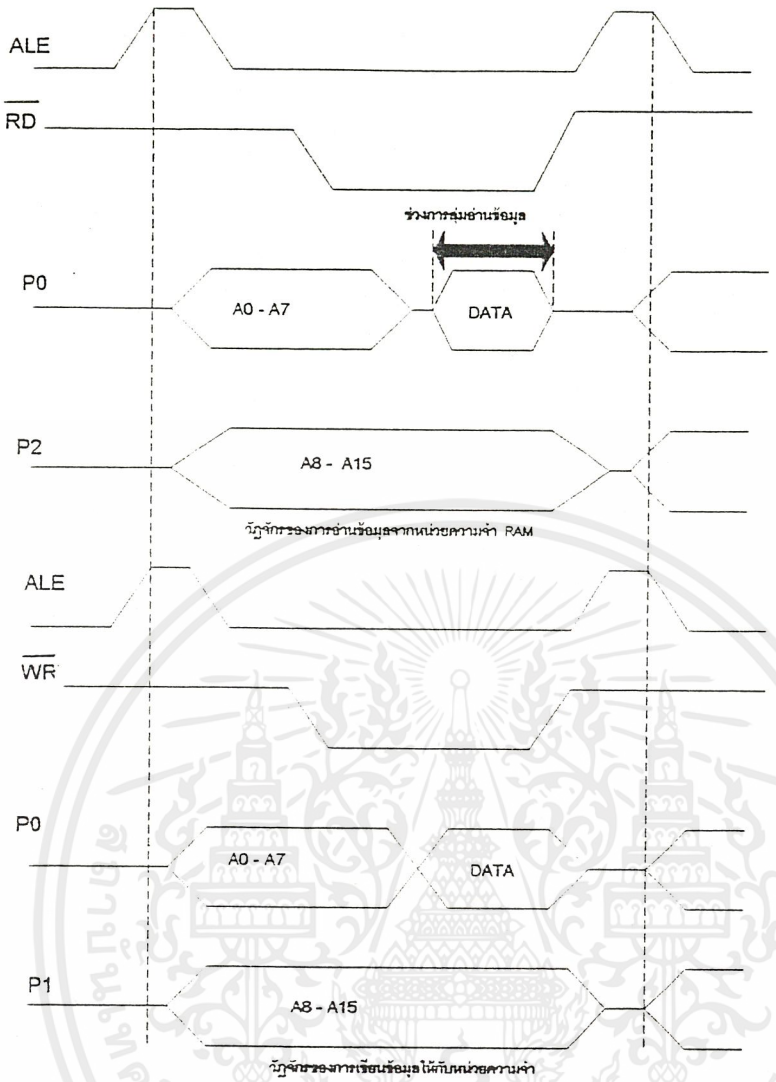
2.11.4 การสื่อสารสองทาง

ในหลายกรณีที่ข้อมูลถูกส่งผ่านในสองทิศทาง โดยเฉพาะเมื่อเครื่องคอมพิวเตอร์สองตัวสื่อสารกัน รวมทั้งในกรณีที่ใช้ซอฟแวร์แฮนด์เช็คกิ้งด้วยเช่นกันจำนวนสายที่น้อยที่สุดที่จำเป็นในการสื่อสารสองทางคือสามเส้น ได้แก่ สายข้อมูลในแต่ละทิศทาง ทำให้จำนวนสายรวมเป็น 5 เส้น ดังรูปที่ 2.19



รูปที่ 2.19 การสื่อสารพร้อมด้วยวงจรแฮนด์เช็คกิ้ง

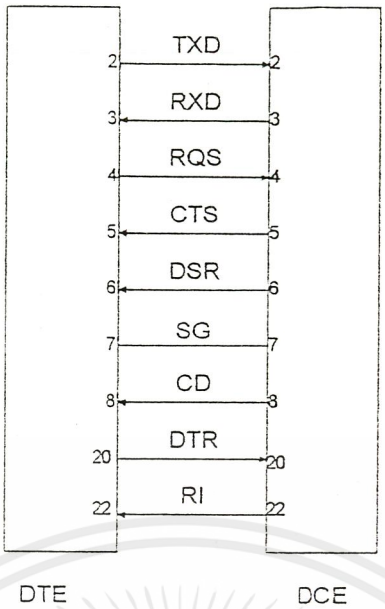
เมื่อสายแฮนด์เช็คกิ้งชุดที่สองถูกนำมาใช้เพิ่มเติมลงไปในแต่ละทิศทาง สายทั้งหมดที่ใช้คือเจ็ดเส้น บางครั้งอาจมีการเพิ่มสายอีกสองเส้น เพื่อทำให้โมเด็มสามารถให้ข้อมูลมากขึ้นแก่คอมพิวเตอร์หรือเทอร์มินัล ได้แก่ ซีดี (CD carrier detect) ถูกเชื่อมต่อเข้ากับขา 8 เพื่อแจ้งการคงอยู่ของสัญญาณพาหะและ อาไอ (RI ring indicator) ถูกเชื่อมต่อกับขา 22 เพื่อแสดงว่าโมเด็มกำลังถูกเรียกโดยอุปกรณ์ระยะไกล ดังนั้นจำนวนวงจรทั้งหมดจะกลายเป็นเก้าดังรูปที่ 2.20



รูปที่ 2.17 แสดงแผนภาพเวลาการติดต่อกับหน่วยความจำแรม ของ 8051

2.11 มาตรฐาน RS - 232 ที่ใช้ในการต่อฮาร์ดแวร์

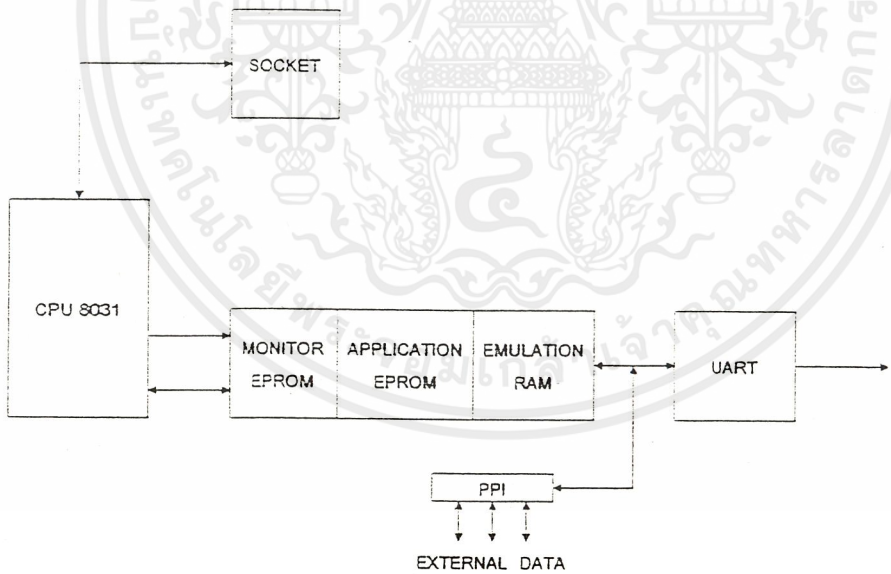
เพื่อที่จะทำให้อุปกรณ์จากผู้ผลิตต่างกันทำงานร่วมกันได้ มาตรฐานหลายชนิดจึงได้รับการออกแบบขึ้น มาตรฐานที่ใช้กันกว้างขวางที่สุดคือ RS - 232 ถูกประกาศโดย Electronic Industries Association มาตรฐาน RS-232 ที่ร่างขึ้นในตอนแรกสำหรับกำหนดการเชื่อมต่อระหว่างเทอร์มินัล และโมเด็มระบุคุณสมบัติทางไฟฟ้าของวงจรระหว่างการเชื่อมต่อวงจร ชื่อวงจรตามมาตรฐานจำได้ยากในทางปฏิบัติจึงใช้ตัวย่อแทน ตัวอย่างเช่น สายเส้นที่ 2 มีชื่อว่า BA แต่ใช้กันทั่วไปว่า TXD ตามมาตรฐาน RS 232 สายเส้นที่2 นำข้อมูลจากเทอร์มินัลไปสู่โมเด็ม เพื่อให้การทำงานถูกต้องเทอร์มินัลต้องส่งเอาท์พุทออกที่สายเส้นที่ 2 และโมเด็มต้องรับข้อมูลบนสายเส้นที่ 2 เพราะฉะนั้นสายเส้นที่ 2 เป็นสายส่งข้อมูลสำหรับอุปกรณ์บางอย่างและเป็นสายรับข้อมูลสำหรับอุปกรณ์อย่างอื่น การเชื่อมต่อโดยตรงจากสายเส้นที่ 2 บนอุปกรณ์หนึ่งเข้ากับสายเส้นที่ 2 บนอุปกรณ์อีกตัวหนึ่ง สามารถทำได้ต่อเมื่ออุปกรณ์หนึ่งส่งข้อมูลบนสายเส้นที่ 2 และอีกตัวหนึ่งรับข้อมูลบนสายเส้นที่ 2



รูปที่ 2.20 การเชื่อมต่อ RS 232 แบบมาตรฐานเก้าเส้น

2.12 การออกแบบ 8051 อีมูลเลเตอร์

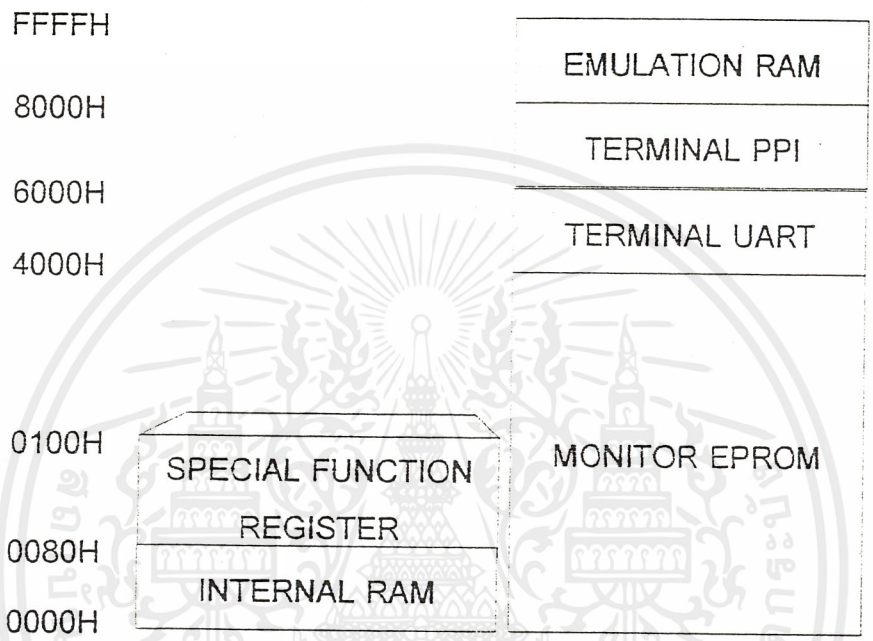
ในส่วนของการออกแบบ 8051 อีมูลเลเตอร์ที่ออกแบบสามารถเขียนเป็นบล็อกไดอะแกรมได้ดังรูปที่ 2.21



รูปที่ 2.21 แสดงบล็อกไดอะแกรมของ 8051 อีมูลเลเตอร์

จากรูปไมโครโปรเซสเซอร์จะใช้เบอร์ 8031 ซึ่งจะเหมือนกับ 8051 เพียงแต่ไม่มีหน่วยความจำภายในแบบรอมภายใน โดยขาทุกขาของไมโครโปรเซสเซอร์จะถูกต่อออกมารอไว้ที่ซอกเก็ตเพื่อต่อไปยังบอร์ดอื่นด้วยสายแถบขนาด 40 เส้น หน่วยความจำของระบบประกอบด้วย โปรแกรมควบคุมระบบของ

8051 อีมูลเลเตอร์ และอีมูลเลชั่นแรมที่ใช้สำหรับเก็บโปรแกรมของบอร์ดอื่น ๆ หรือโปรแกรมอื่น ๆ ที่สั่งงานให้ 8051 อีมูลเลเตอร์ทำงานนอกเหนือไปจากโปรแกรมที่กำหนดไว้ในโปรแกรมควบคุมระบบ การติดต่อกับ IBM - PC จะกระทำผ่านทางชิฟ UART ซึ่งทำหน้าที่เป็นพอร์ทอนุกรมและสักษณะของหน่วยความจำจะเป็นดังรูปที่2.22



รูปที่ 2.22 แสดงการแม็พหน่วยความจำของ 8051 อีมูลเลเตอร์

2.13 8255 (พอร์ทขนานที่โปรแกรมได้ THE 8255 PROGRAMMABLE PERIPHERAL INTERFACE)

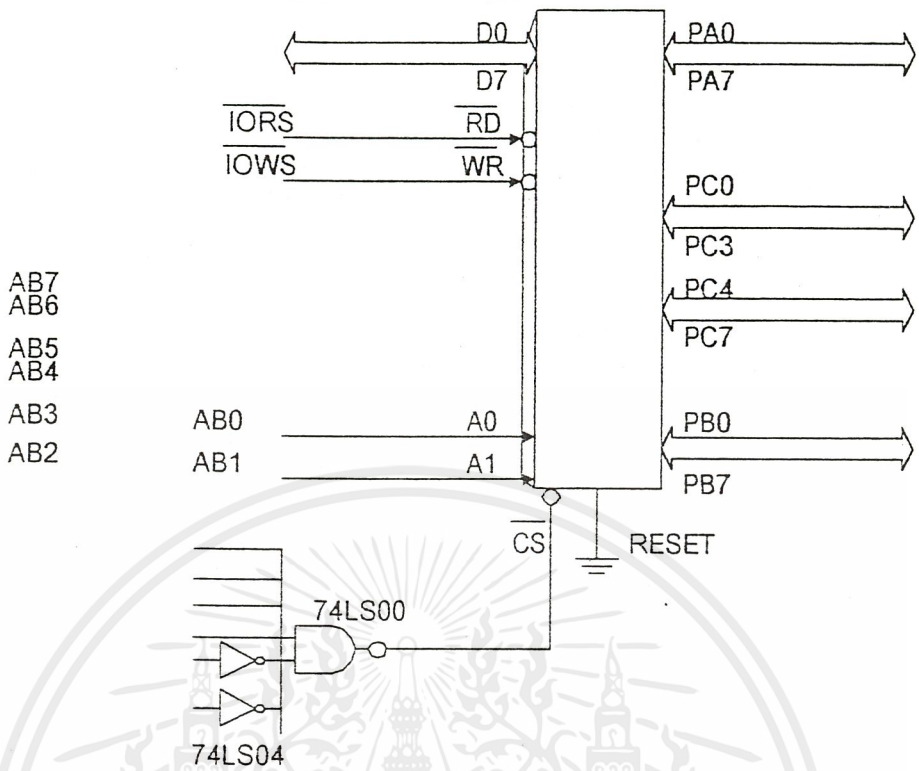
มีชื่อเรียกย่อ ๆ ว่า 8255 PPI เป็นพอร์ท 8 บิต 3 พอร์ท ที่โปรแกรมได้ โดยแบ่งเป็นพอร์ท A , B , C โดยที่พอร์ทซียังแบ่งได้อีก คือ แบ่งเป็นพอร์ท ซี 4 บิทล่าง และพอร์ทซี 4 บิทบน

การทำงานมี 3 โหมด คือ

1. Basic I/O ไม่มีการทำแฮนด์เช็คกิ้ง (Handshaking)
2. Stobe I/O มีการทำแฮนด์เช็คกิ้ง
3. Stobe Bidirectional I/O ส่งข้อมูล 2 ทิศทางมีแฮนด์เช็คกิ้ง

2.13.1 การเชื่อมต่อ 8255 (interface the 8255)

การต่อดังแสดงในรูป 2.23 ขา $\overline{CS}$ ก็คือชิฟซีเล็ก เป็นขาเลือกชิฟนี้ให้ทำงานหรือไม่ทำงาน โดยถ้าเห็น $\overline{CS}$ (มีขีดข้างบน $\overline{CS}$ แสดงว่า จะทำงานได้ต้องให้ลอจิก ' 0') ส่วน A0 , A1 จะเป็นการเลือกพอร์ทได้ 4 พอร์ทคือ A,B,C และพอร์ตควบคุม



รูปที่ 2.23 การเชื่อมต่อ 8255 กับบัสทั้ง 3 ของซีพียู

บทที่ 3 การคำนวณและการสร้าง

3.1 วงจรที่ใช้ในการทดลองและรายละเอียดของวงจรที่ใช้งาน

สำหรับวงจรที่จะเป็นดังรูปที่ 3.3 โดยรายละเอียดการทำงานคร่าว ๆ ของวงจรเป็นดังนี้ โดยเริ่มจาก IC2 เป็นไมโครคอมพิวเตอร์แบบชิฟเดียวเบอร์ 8031 เนื่องจากต้องการใช้หน่วยความจำภายนอกดังนั้นขา 31 จึงถูกต่อลงกราวด์ ขา 21 ถึง 28 เป็นแอดเดรสบัสโดยใช้ IC 5 (74LS224) เป็นตัวบัฟเฟอร์ IC 6 (74LS245) ทำหน้าที่แอดเดรสและดาต้าบัฟเฟอร์ให้ขา 32 ถึง 39 ของไมโครโปรเซสเซอร์ IC 6 จะถูกควบคุมทิศทางการส่งออกหรือรับเข้าด้วยเอาท์พุทของแอนด์เกตโดยมีอินพุทจากขา 17 ($\overline{RD}$) และขา 29 ($\overline{PSEN}$) ขา 30 ($\overline{ALE}$) ขา 17 ($\overline{RD}$) และขา 29 ($\overline{PSEN}$) จะถูกบัฟเฟอร์ด้วย IC4 (74LS244) สัญญาณที่ผ่านการบัฟเฟอร์แล้วจะถูกต่อไปยัง IC1 โดยตรงยกเว้นขา 9 และขา 18 , 19 ซึ่งต่อกับก้อนผลึกจะมีสวิตซ์ให้เลือกว่าต้องการต่อมาจากบอร์ดที่ถูกอีมูลเลเตอร์เอง

สัญญาณมัลติเพล็กซ์ระหว่างแอดเดรสและข้อมูลที่ได้รับจาก IC 6 ถูกแยกสัญญาณแอดเดรสออกโดย IC10 (74LS373) โดยมีสัญญาณ ALE เป็นตัวควบคุม เอาท์พุทที่ได้จาก IC 10 จะเป็นสัญญาณของแอดเดรสอย่างเดียว ขาที่สำคัญ ๆ ของ 8031 มีดังนี้

ALE

Address Latch Enable ขานี้จะส่งสัญญาณที่มีความถี่ 1 / 6 เท่าของสัญญาณจากออสซิลเลเตอร์ สัญญาณนี้จะส่งออกมาตลอดเวลายกเว้นบางครั้งของการติดต่อกับหน่วยความจำสำหรับข้อมูลภายนอก 8051 สัญญาณนี้จะใช้บอกกับอุปกรณ์ภายนอก 8051 ว่าขณะนี้สัญญาณแอดทีฟ (เป็นลอจิก 1) จะมีการส่งข้อมูลที่เป็น 8 บิทล่างของตำแหน่งหน่วยความจำภายนอก 8051 ที่ต้องการติดต่อออกไปทางพอร์ท 0 อุปกรณ์ภายนอกจะใช้สัญญาณนี้ในการหน่วงข้อมูลไว้ที่พอร์ท 0 จะส่งค่าตำแหน่งหน่วยความจำออกมาเพียงชั่วขณะเท่านั้นซึ่งในเวลาต่อมาพอร์ท 0 จะใช้รับส่งข้อมูลกับหน่วยความจำภายนอก สัญญาณ ALE จะสามารถต่อเข้ากับอุปกรณ์ TTL ชนิด LS ได้ถึง 8 อินพุท

$\overline{PSEN}$

Program Store Enable เป็นขาที่ 29 ขานี้ปกติจะให้ลอจิก 1 แต่จะส่งลอจิก 0 เมื่อต้องการอ่านคำสั่ง (Fetch Instruction) ที่จะนำไปทำงานมาจากหน่วยความจำสำหรับโปรแกรมภายนอก 8051 ในกรณีที่อ่านคำสั่งซึ่งเก็บอยู่ในหน่วยความจำสำหรับโปรแกรมภายใน 8051 แล้วสัญญาณนี้จะไม่เปลี่ยนลอจิกเป็น 0 ขา $\overline{PSEN}$ นี้สามารถต่อไปยังขาอินพุทของ TTL ชนิด LS ได้ถึง 8 อินพุท

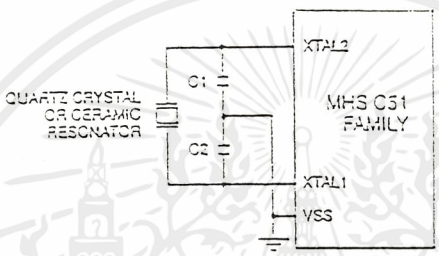
$\overline{EA}$

External Access ขา 31 ขานี้เป็นขาอินพุทที่ต่อเข้าไปยังวงจรไทม์มิ่งและคอนโทรลเพื่อควบคุมการสร้างสัญญาณ $\overline{PSEN}$ ถ้าป้อนสัญญาณลอจิก 0 เข้าที่ขา EA นี้แสดงว่าโปรแกรมในตำแหน่ง 0000H ถึง FFFFH ที่ต้องการให้ทำงานถูกเก็บไว้ภายนอก 8051 จะต้องสร้างสัญญาณ $\overline{PSEN}$ ออกไปยังภายนอกเพื่อทำการอ่านคำสั่งเข้ามาทำงาน แต่ถ้าสัญญาณที่ป้อนให้ขา $\overline{EA}$ เป็น 1 หมายความว่าโปรแกรมในตำแหน่ง 0000H

ถึง 0FFFH ถูกเก็บไว้ภายใน 8051 การทำงานในตำแหน่งหน่วยความจำช่วงนี้จะอ่านคำสั่งต่าง ๆ จากรอมภายใน 8051

XTAL 1

ขาที่ 19 ของบอร์ด 8031 ขานี้จะต่อเข้ากับขาที่เป็นวงจรขยายแบบป้อนกลับ (inverting amplifier) ที่ประกอบเป็นวงจรออสซิลเลเตอร์ ในรูปที่ 3.1 จะเห็นว่าวงจรภายในของออสซิลเลเตอร์แนนเกต จะทำหน้าที่เป็นวงจรขยายแบบกลับเฟสของสัญญาณที่จะควบคุมให้มีการออสซิลเลตหรือไม่ก็ขึ้นกับสัญญาณ RD ถ้าต้องการใช้สัญญาณนาฬิกาจากภายนอกเป็นสัญญาณนาฬิกา ควบคุมการทำงานของ 8051 ก็ให้ป้อนสัญญาณเข้ามาที่จุดนี้แต่ถ้าต้องการใช้วงจรออสซิลเลเตอร์ภายในก็ให้ต่อคริสตอล (crystal) ดังรูปที่ 3.1 โดยค่าคาปาซิ เตอร์ในวงจรควรมีค่าประมาณ 20 PF



รูปที่ 3.1 วงจรออสซิลเลเตอร์ภายใน 8031

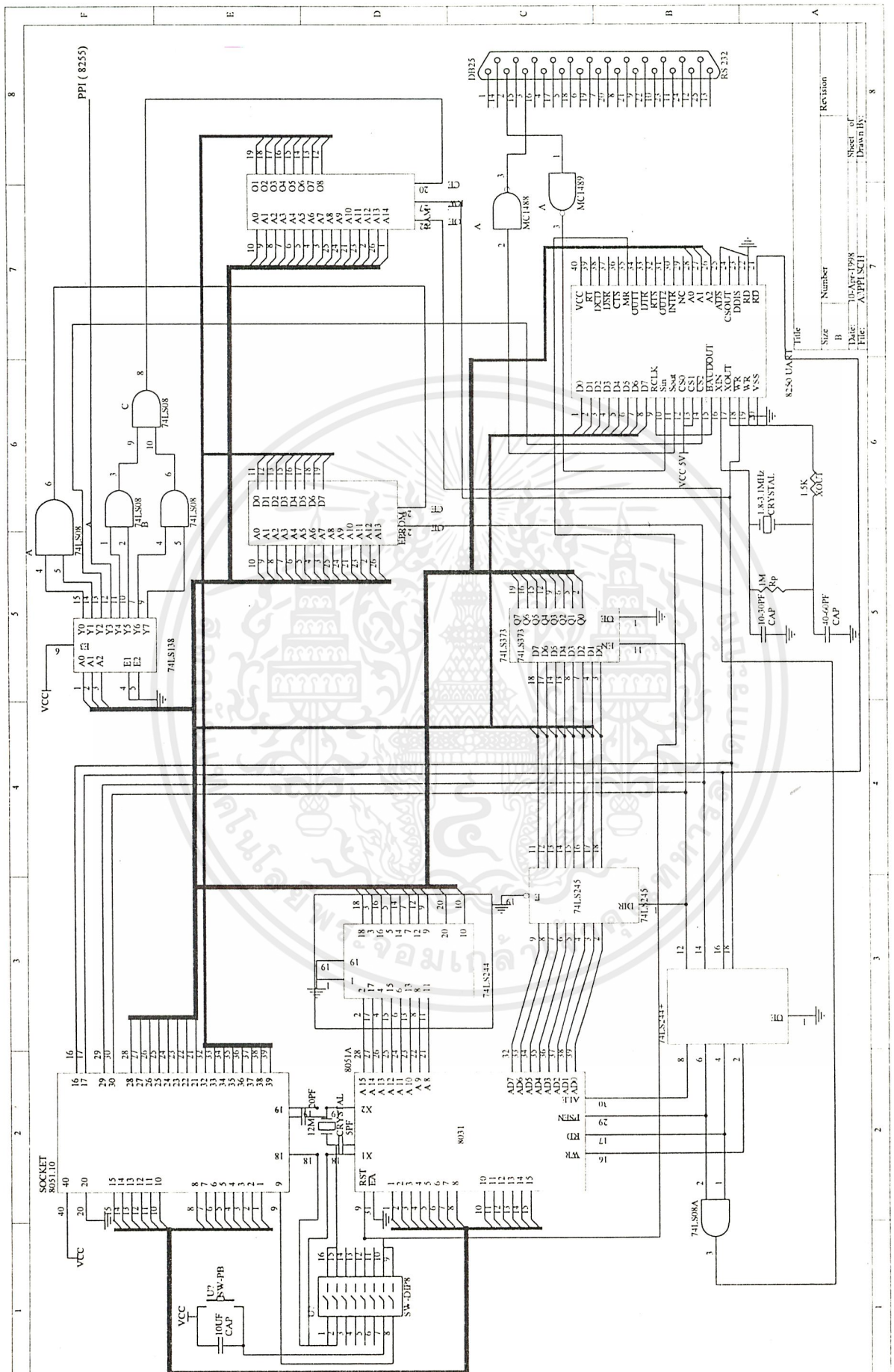
XTAL 2

ขาที่ 18 ของบอร์ด 8031 ขานี้เป็นจุดเอาท์พุทของวงจรขยายแบบกลับเฟสสัญญาณที่ประกอบเป็นวงจรออสซิลเลเตอร์ (อินพุทคือขา XTAL 1) ถ้าจะใช้สัญญาณนาฬิกาที่สร้างมาจากภายนอกมาเป็นสัญญาณนาฬิกาของ 8051 ให้ปล่อยขานี้ลอยไว้แล้วป้อนสัญญาณนาฬิกาจากภายนอกเข้ามาที่ขา XTAL 1 ดังรูปที่ 3.2

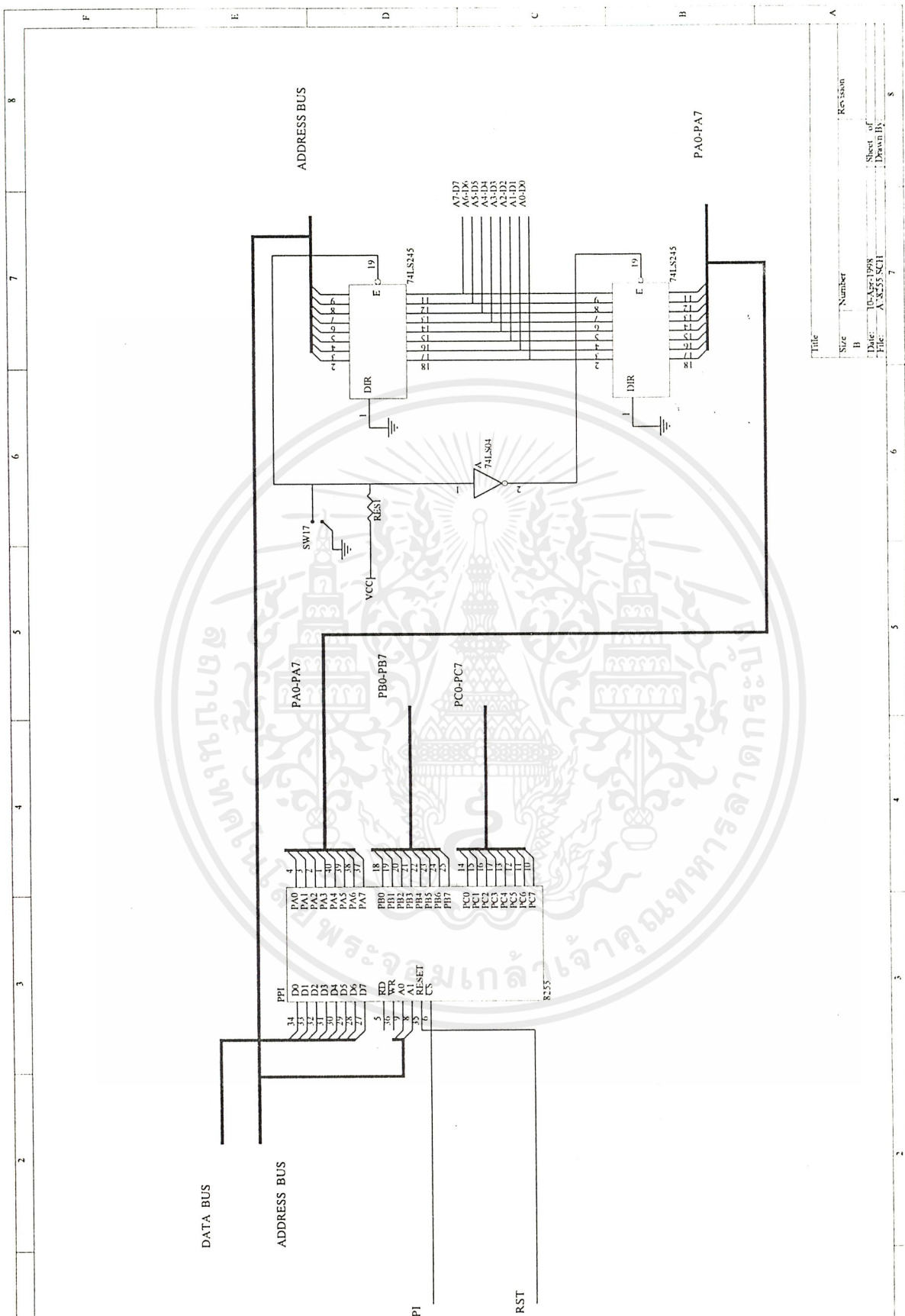
รูปที่ 3.2 8031 ที่ทำงานโดยสัญญาณที่มาจากภายนอก

สายแอดเดรสเส้นที่ 13 14 15 ถูกถอดรหัสด้วย IC9 (74LS138) ซึ่งจะทำให้เอาท์พุทบล็อกละ 8 กิโลไบท์ โดยบล็อกที่ 1 และ 2 จะเป็นแอดเดรสของอีพรอม บล็อกที่ 3 เป็นของส่วนที่ทำหน้าที่เปลี่ยนข้อมูลขนานเป็นอนุกรมและข้อมูลอนุกรมเป็นขนาน (UART Universal Asynchronous Receiver and Transmitter) ซึ่งทำหน้าที่เปลี่ยนข้อมูลแบบขนานเป็นอนุกรมออกไปที่ขา 19 และรับข้อมูลแบบอนุกรมจากขาที่ 3 แล้วเปลี่ยนให้เป็นขนานส่งเข้าสู่บัสข้อมูล

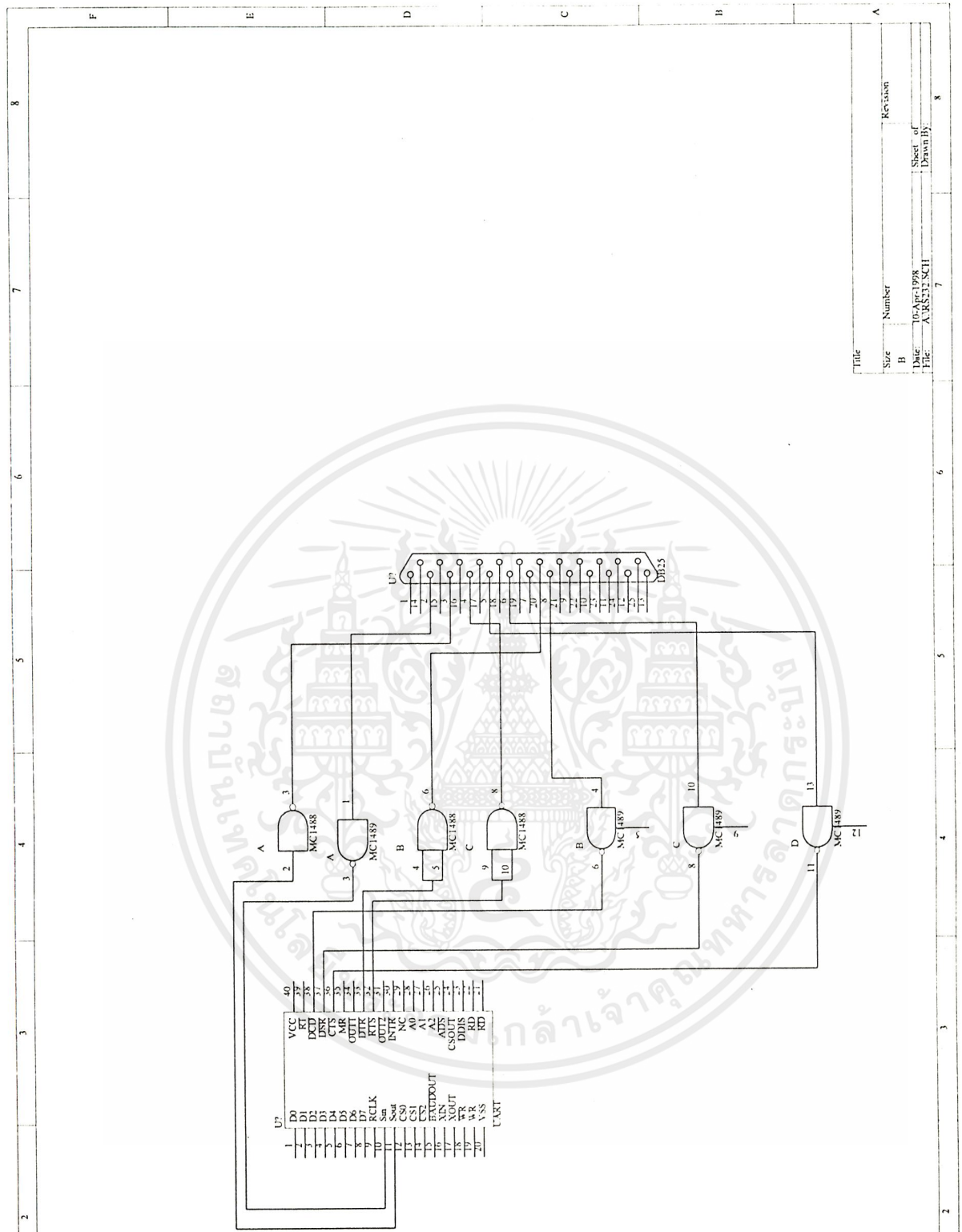
IC 16 (MC 1488 and MC1489) เป็นตัวเปลี่ยนระดับของสัญญาณให้เข้าสู่มาตรฐานของ RS - 232 ขา 12 ของยูอาสต่ออยู่กับแอดเดรสเอศูนย์ เพื่อระบุว่าข้อมูลที่ส่งไปเป็นข้อมูลของคำสั่งหรือข้อมูลที่ต้องการส่งออก หรือเป็นข้อมูลรับเข้ามาจากพอร์ทอนุกรม ขาข้อมูลทั้งหมดต่อมาจากเอาท์พุทของ IC 6 ขา 13 (RD) ขา 10 ($\overline{WR}$) ต่อมาจากเอาท์พุทของ IC 4 ส่วนขา 21 (RST) ต่อมาจาก 8031 สำหรับขา 9 ขา 20 และขา 25 ต่ออยู่กับส่วนกำเนิดสัญญาณ (Baud - rate generator) เพื่อกำหนดอัตราบอดในการรับและส่งข้อมูลหน่วยความจำชนิดแรมของระบบขนาด 32 กิโลไบท์ใช้สัญญาณควบคุมจาก IC 9 ซึ่งนำมาแอนด์กันเพื่อที่จะครอบคลุมได้ถึง 32 กิโลไบท์



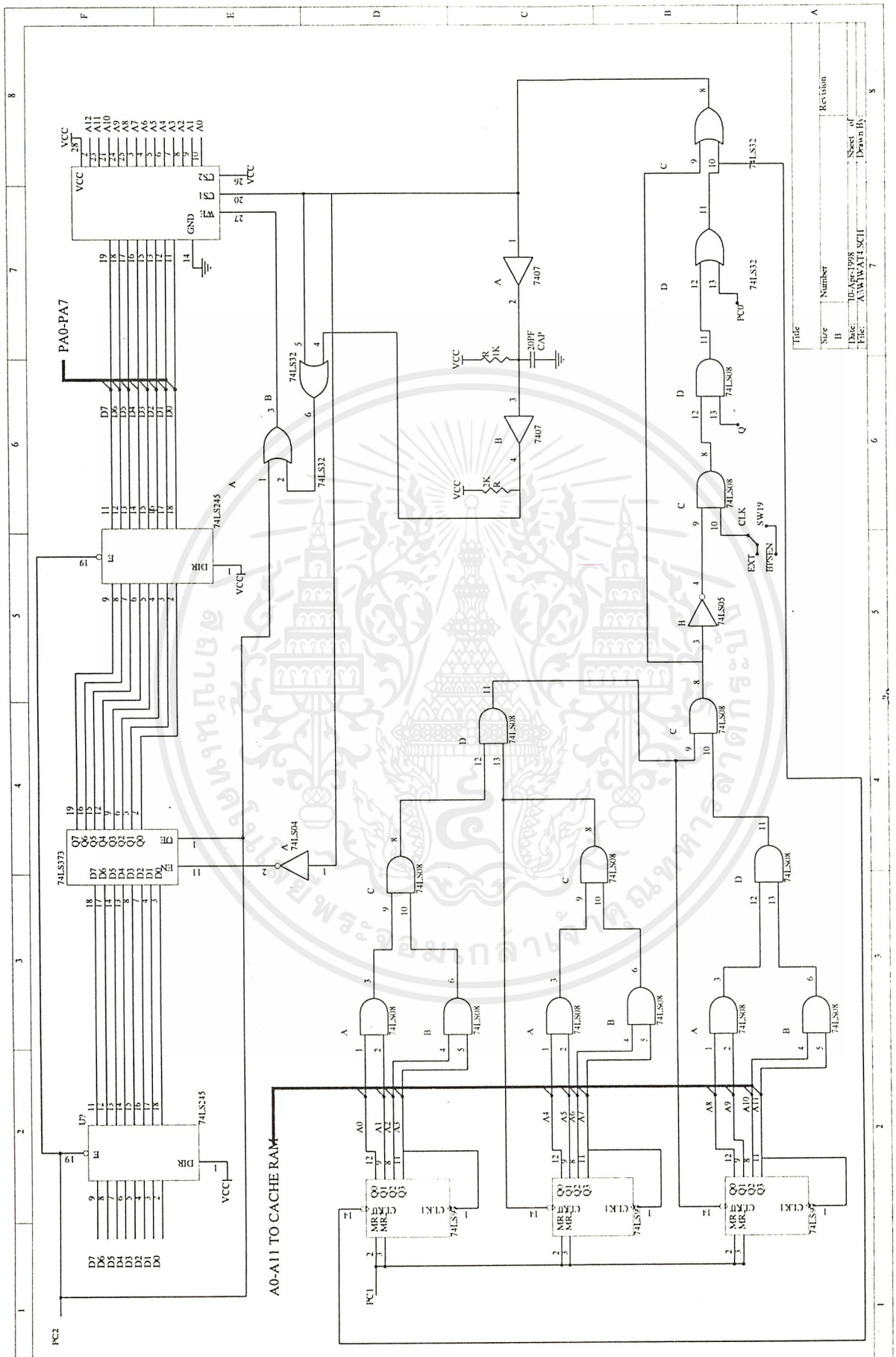
รูปที่ 3.3 แสดงวงจรของ 8051 อีมูลเลเตอร์



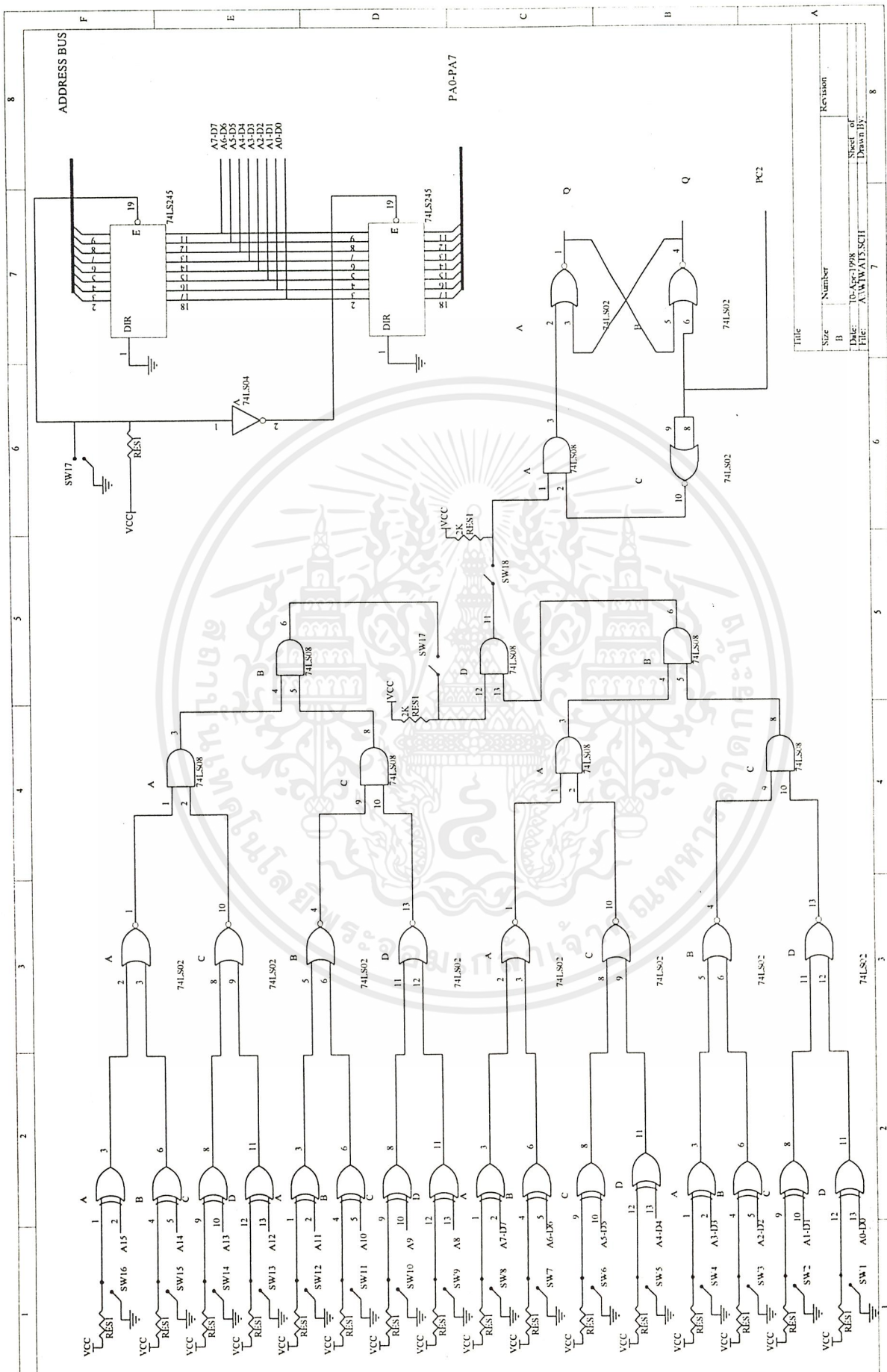
รูปที่ 3.3.1 แสดงวงจรของ 8051 อีมูลเลเตอร์ (ต่อ)



รูปที่ 3.3.2 แสดงวงจรของ 8051 อีมูลเลเตอร์ (ต่อ)



รูปที่ 3.4 แสดงวงจรส่วนตัวจับข้อมูล



รูปที่ 3.4.1 แสดงวงจรส่วนดักจับข้อมูล

3.2 Timer/Counter ของ MCS - 51

MCS-51 มีTime/Counter 2 Channels โดยเป็น register แบบนับขึ้น16 bits โดยกรณีTimer จะนับ clock ภายในและกรณี Counter จะนับ clock จากภายนอก (T0 และ T1) register แบบ 8 bits 2 ตัวที่ใช้ในการควบคุมคือ TCON และ TMOD

3.2.1 Timer / Counter Control register (TCON)

ใช้เป็น flag บอกสถานะว่ามี overflow หรือ interrupt ควบคุม Timer/Counter ให้ on/off

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1 ถ้า overflow Timer1 overflow จะ set เป็น 1 และ reset เป็น 0 โดยอัตโนมัติ เมื่อJump ไปยัง Timer Interrupt service routine ที่ vector interrupt 001 BH

TR1 Timer 1 run control การเริ่ม หรือหยุดทำงานของ timer โดยการ set (on) หรือ reset(off)

TF0 Timer 0 overflow flag เป็น 1 เมื่อมีการ overflow ของ timer และถูก reset เป็น 0 โดยอัตโนมัติ เมื่อกระโดดไปตำแหน่ง 000BH เพื่อปฏิบัติ timer interrupt service routine

TR0 Timer 0 on/off (1 ให้เริ่มนับ ; 0 ให้หยุดนับ)

IE1 External interrupt 1 edge flag (จะถูก set โดย hardware เมื่อมีการ interrupt จากINTI และreset เมื่อ ปฏิบัติ Interrupt routine)

IT1 Interrupt 1 Type control คือ 0 = level active(หรือ level trig)

1 = transition (หรือ edge trig)

IE0 External Interrupt 0 edge flag (จะถูก set โดย hardware เมื่อมีการ interrupt จาก INTO และ reset เมื่อ ปฏิบัติ Interrupt routine)

IT0 Interrupt 0 Type control

Note กรณี edge trig นั้นตอนเป็น counter จะนับทุกครั้งที่ขา Tx เปลี่ยนจาก 1 เป็น 0

Timer/Counter Mode control register (TMOD)

ใช้ set ชนิดการทำงานของ timer/counter 0 และ 1

3.2.2 TMOD (ไม่สามารถใช้เป็น bit addressable)

GATE	C/T	M1	M0	GATE	C/T	M1	M0

GATE ถ้า GATE = 1 TR ตัวที่ x (ใน TCON) = 1 และ Timer / Counter ตัวที่ x จะทำงานต่อเมื่อ ขา INTx เป็น สูงและ GATE = 0 Timer / Counter จะทำงานเมื่อ TRx = 1 ใน TCON

C/T เท่ากับ 0 timer (อินพุทจากคล็อกภายใน)

เท่ากับ 1 counter (อินพุทจากขา Tx โดย x = 0 หรือ 1)

M1&M0 ใช้ในการกำหนดโหมดการทำงาน

Mode	M1	M0	การทำงาน
0	0	0	13 บิตไทม์เมอร์
1	0	1	16 บิตไทม์เมอร์ - เคาน์เตอร์
2	1	0	8 บิตออโต้รีโหลด ไทม์เมอร์ - เคาน์เตอร์
3	1	1	ไทม์เมอร์ - เค่าน์เตอร์ ทั้ง 0 และ 1 การทำงานต่างกัน

กรณีโหมด 2 ไทม์เมอร์ - เคาน์เตอร์

1. THx ค่าที่ใช้เพื่อเริ่มนับ
2. TLx ค่าเริ่มต้นที่ counter เริ่มนับขึ้นโดยรับค่านี้มาจาก THx

กรณีโหมด 3

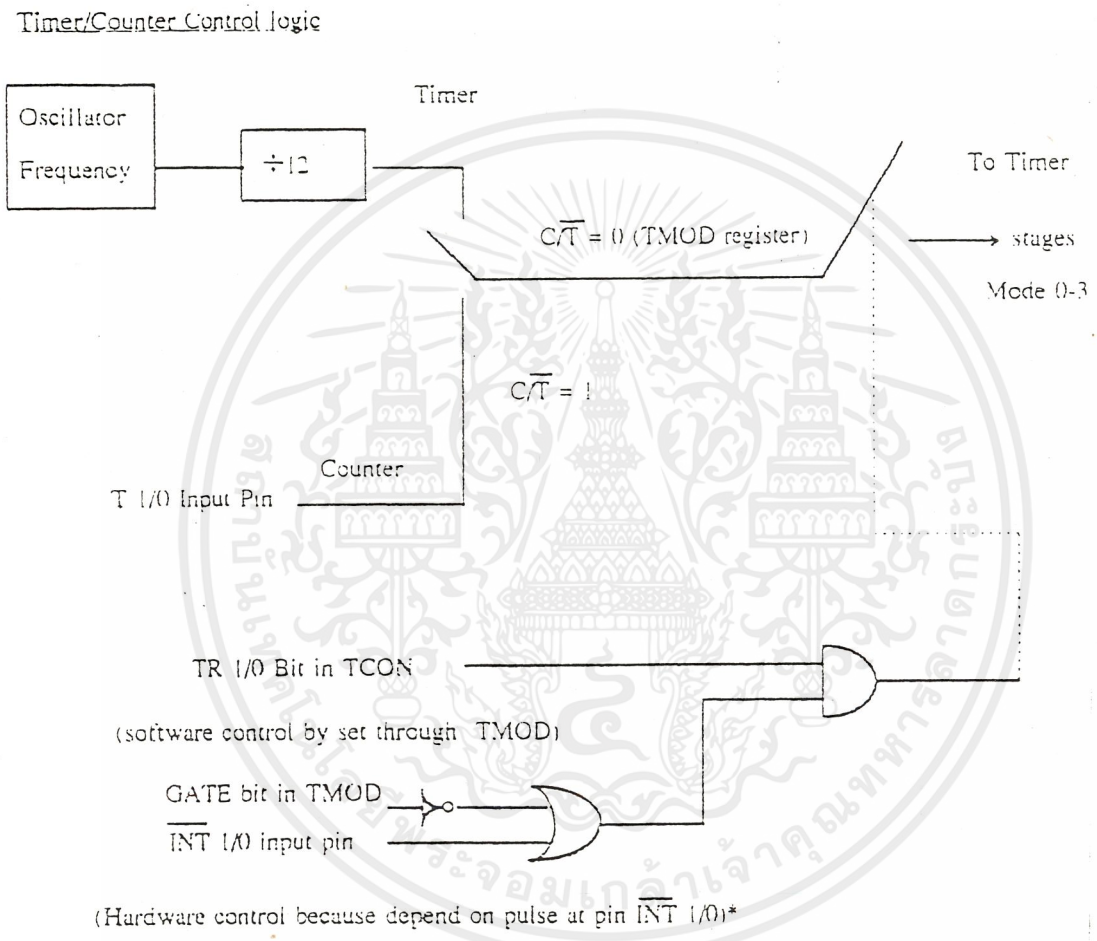
สำหรับไทม์เมอร์ - เคาน์เตอร์

1. 8 บิตไทม์เมอร์ 2 ชุด (กรณีไทม์เมอร์)
2. 8 บิตเคาน์เตอร์ 1 ชุด (กรณีเคาน์เตอร์)

สำหรับ Time

1. ไทม์เมอร์ ใช้หยุดไทม์เมอร์ - เคาน์เตอร์ 0
2. เคาน์เตอร์ ไม่ให้ใช้

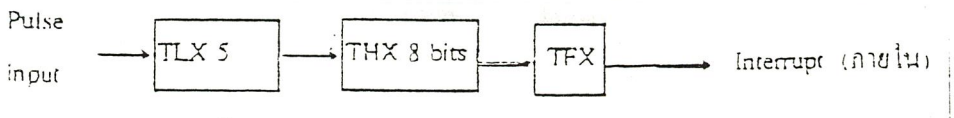
ข้อสังเกต ใช้ ไทม์เมอร์ - เคาน์เตอร์ 0 ได้สมบูรณ์แบบทั้ง 4 โหมด (แต่ไทม์เมอร์ - เคาน์เตอร์ 1 ได้เพียง 3 โหมด)



รูปที่ 3.5 แสดงการควบคุมการทำงานแบบ ไทม์เมอร์ - เคาน์เตอร์

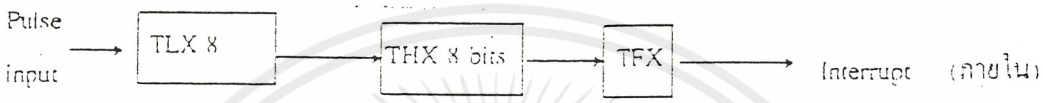
แต่ละโหมดของไทม์เมอร์ - เคาน์เตอร์มีลักษณะดังนี้

โหมด 0 13 บิตไทม์เมอร์ - เคาน์เตอร์ โดยสามารถนับจาก 0000H - 1FFFH เมื่อเกิดการโอเวอร์โฟลจะเกิดอินเตอร์รัฟท์ภายในและเริ่มนับใหม่จาก 0000H



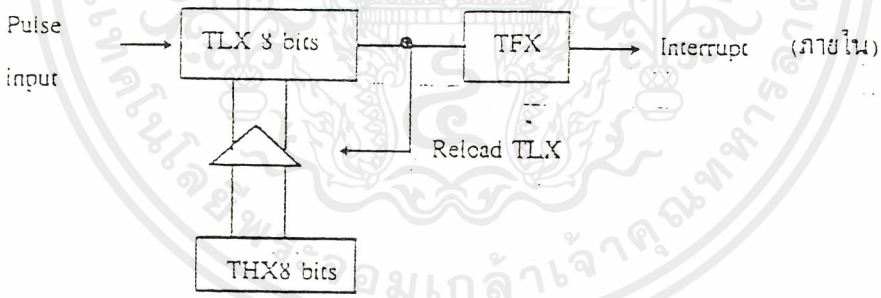
รูปที่ 3.6 แสดงไทม์เมอร์ - เคาน์เตอร์ โหมด 0

โหมด 1 16 บิทไทม์เมอร์ - เคาน์เตอร์ โดยสามารถนับจาก 0000H - FFFFH เมื่อโอเวอร์โฟลจะเกิดอินเตอร์รัฟท์ภายในและเริ่มนับใหม่จาก 0000H



รูปที่ 3.7 แสดงไทม์เมอร์ - เคาน์เตอร์โหมด 1

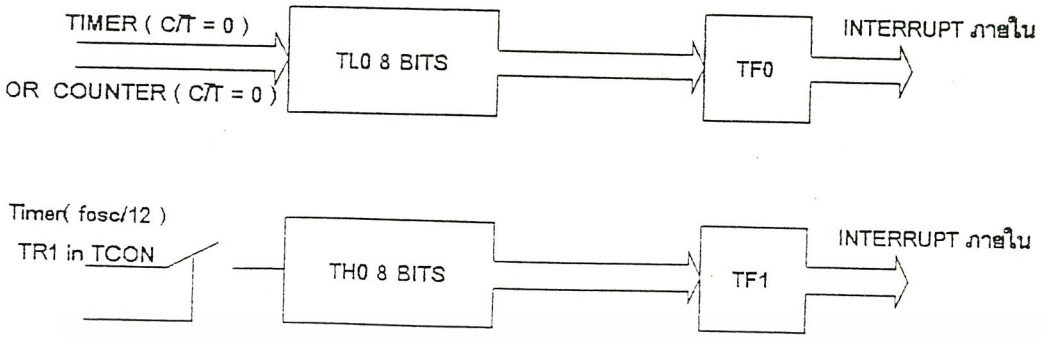
โหมด 2 ออโต้รีโหลด (Auto - reload) ของTL จาก TH โดยสามารถนับจากค่าเริ่มต้นจากค่าใน THx จนถึง FFH เมื่อโอเวอร์โฟลว์จะเกิด อินเทอร์รัฟภายในและเริ่มนับใหม่จากค่าที่อยู่ใน THX



รูปที่ 3.8 แสดงไทม์เมอร์ - เคาน์เตอร์โหมด 2

NOTE THX8 เก็บค่าเริ่มต้นโดย THX →TLX แล้วเริ่มนับและเมื่อนับถึง FFH แล้วTLX จะนำเอาค่าใน THX8 กลับมาเริ่มนับใหม่ ไม่ใช่กลับไปเริ่มนับ 00H

โหมด 3 8 bits timer 2 ชุดหรือ counter เพียง 1 ชุด โดยสามารถนับจาก 00H

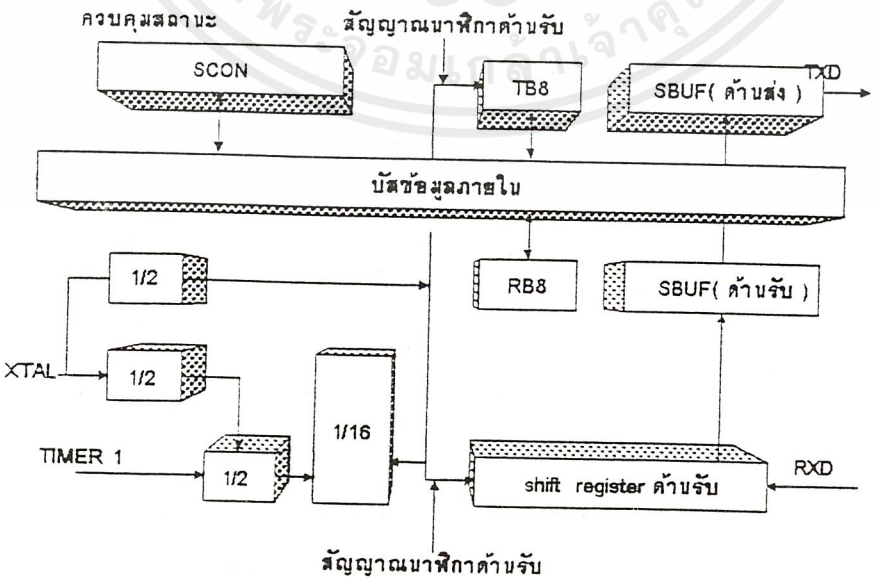


รูปที่ 3.9 แสดงไทม์เมอร์ - เคาน์เตอร์ โหมด 3

NOTE กรณีอินเทอร์รัปท์เนื่องจากเกิดโอเวอร์โฟลว์ของ ไทม์เมอร์ - เคาน์เตอร์ 0 หรือจะกระโดดที่โปรแกรมเมมโมรี่ตำแหน่ง 001BH ตามลำดับ

3.3 การจัดการข้อมูลอนุกรมของ 8051

พอร์ทอนุกรมของ 8051 มีโครงสร้างการทำงานในแบบที่เรียกว่า ฟูลดูเพล็กซ์ (Full Duplex) ซึ่งหมายถึงความสามารถในการรับและการส่งข้อมูลอนุกรมได้ในเวลาเดียวกัน จากรูปที่ 3.10 แสดงให้เห็นถึงแผนภาพการทำงานอย่างง่ายของวงจรส่วนจัดการข้อมูลอนุกรมของ 8051 โดยทางด้านวงจรของตัวส่งประกอบด้วยรีจิสเตอร์ SBUF ทำหน้าที่เก็บข้อมูลที่จะส่งออกการใช้คำสั่งเขียนหรือโอนย้ายข้อมูลมายังรีจิสเตอร์นี้จะเป็นการส่งข้อมูลนั้นออกไปยังพอร์ตอนุกรมทางขาสัญญาณ TxD โดยอัตโนมัติ ส่วนวงจรด้านตัวรับ (Receiver) ประกอบด้วยรีจิสเตอร์ SBUF เช่นเดียวกัน แต่ทำหน้าที่เก็บข้อมูลที่นำมาจากส่วนของวงจรเลื่อนบิทหรือชิฟรีจิสเตอร์ของวงจรจัดการข้อมูลอนุกรมภายใน สัญญาณข้อมูลอนุกรมที่รับเข้าจะผ่านมาทางขาสัญญาณ RxD



รูปที่ 3.10 แสดงแผนภาพการทำงานของวงจรส่วนการรับและส่งข้อมูลอนุกรมของ 8051

พอร์ทอนุกรมของ 8051 สามารถโปรแกรมให้ทำหน้าที่ในรูปแบบต่าง ๆ กัน 4 แบบ (หรือเรียกว่า โหมดการทำงาน) โดยการกำหนดค่าบิท SM0 และ SM1 ซึ่งอยู่ภายในรีจิสเตอร์ควบคุมและบอกสถานะ SCON โหมดการทำงานทั้ง 4 แบบของอนุกรม มีดังนี้

โหมด 0 เป็นการขยายพอร์ทอินพุตเอาท์พุต โดยทำงานร่วมกับไอซีชิฟรีจิสเตอร์ภายนอกประเภททีทีแอลหรือซีมอส

โหมด 1 ใช้สำหรับการเชื่อมต่ออนุกรมแบบ UART (Universai asynchronous receiver / transmitter) โดยการใช้กลุ่มข้อมูลแบบ 10 บิท และสามารถเปลี่ยนแปลงอัตราความเร็วในการส่งข้อมูลได้

โหมด 2 ใช้สำหรับการเชื่อมต่ออนุกรมแบบ UART โดยการใช้กลุ่มข้อมูลแบบ 11 บิทและกำหนดอัตราความเร็วในการส่งข้อมูลคงที่

โหมด 3 ใช้สำหรับการเชื่อมต่ออนุกรมแบบ UART โดยการใช้กลุ่มข้อมูลแบบ 11 บิทและสามารถเปลี่ยนแปลงอัตราความเร็วในการส่งข้อมูลได้

นอกจากนี้โหมด 2 และ 3 ยังมีการดำเนินการแบบพิเศษออกไป โดยสามารถนำมาใช้ประโยชน์ในการสื่อสารข้อมูลแบบที่มีไมโครโปรเซสเซอร์หลายตัวทำงานร่วมกันได้

จากรูปที่ 3.10 ชิฟรีจิสเตอร์ภายในตัวส่งจะทำหน้าที่ในการเลื่อนบิทข้อมูลออกไปภายนอกโดยไม่มีการบัฟเฟอร์และเมื่อใดที่มีการเขียนข้อมูลให้กับรีจิสเตอร์ SBUF แสดงว่ามีความต้องการที่จะส่งข้อมูลนี้ออกไปแบบอนุกรม สำหรับชิฟรีจิสเตอร์ทางด้านรับจะทำการเลื่อนบิทข้อมูลที่รับเข้ามาเก็บไว้ เมื่อบิทของข้อมูลที่รับมาครบถ้วนจะถูกย้ายไปเก็บยังรีจิสเตอร์ SBUF ต่อไป อย่างไรก็ตามการย้ายข้อมูลนี้จะเกิดขึ้น ก็ต่อเมื่อรีจิสเตอร์ SBUF นั้นไม่มีข้อมูลที่จะทำการส่งหรือได้ส่งข้อมูลออกไปเสร็จสิ้นแล้ว

ชื่อบิท SCON ตำแหน่ง 98 h ค่าบิทเริ่มต้น 0000 0000

SM0	SM1	SM2	REN	TB8	RB8	TI	RI

ชื่อบิท	ตำแหน่ง	หน้าที่
SM0	SCON.7	แฟล็กกำหนดการทำงานแบบมัลติโปรเซสเซอร์
SM1	SCON.6	บิทเลือกโหมดการทำงาน
SM2	SCON.5	บิทเลือกโหมดการทำงาน
REN	SCON.4	แฟล็กยอมให้มีการรับข้อมูล
TB8	SCON.3	ค่าของบิทที่ 9 สำหรับการส่งข้อมูลออก

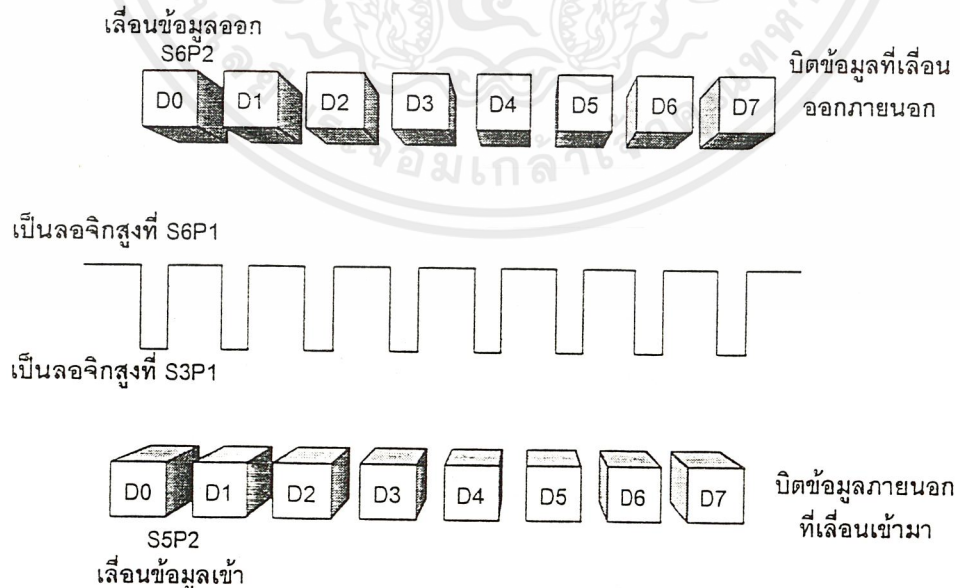
RB8	SCON.2	ค่าของบิทที่ 9 ของข้อมูลที่รับเข้า
TI	SCON.1	แฟล็กแสดงการอินเตอร์รัปท์ภายหลังการส่งข้อมูล
RI	SCON.0	แฟล็กแสดงการอินเตอร์รัปท์เมื่อมีข้อมูลรับเข้า

รูปที่ 3.11 แสดงรีจิสเตอร์ควบคุมการทำงานและบอกสถานะการสื่อสารข้อมูลอนุกรม SCON

3.3.1 พอร์ตอนุกรมโหมด 0

การทำงานของพอร์ทอนุกรมในโหมด 0 เป็นการขยายพอร์ทอินพุตหรือพอร์ทเอาท์พุต ให้มีจำนวนมากขึ้น โดยจะทำการสร้างสัญญาณนาฬิกาขึ้นเพื่อให้จังหวะของการทำงานที่พร้อมกัน (Synchronizing) สำหรับการเลื่อนบิทเข้าหรือออกจากไอซีรีจิสเตอร์ภายนอก เมื่อมีการโอนย้ายข้อมูลมายังรีจิสเตอร์ในแต่ละครั้งก็จะส่งผลให้เกิดการส่งบิทข้อมูลทั้ง 8 บิทออกมา แม้ว่าแฟล็กสถานะ T1 จะยังคงมีค่าเป็น 1 อยู่ก็ตาม นอกจากนี้แล้วเมื่อใดก็ตามที่ค่าของแฟล็กสถานะ RI เป็นค่า 1 ก็ควรที่จะย้ายข้อมูลที่รับเข้ามานั้นออกไปจากรีจิสเตอร์ SBUF เสียก่อนที่จะได้มีการกำหนดค่าแฟล็ก RI ให้เป็น 0 เพื่อรับข้อมูลใหม่ต่อไป

การทำงานของพอร์ตอนุกรมในโหมด 0 เป็นการรับและส่งข้อมูลอนุกรมจำนวน 8 บิท โดยใช้เพียงขาสัญญาณ RxD เท่านั้น ส่วนขาสัญญาณ TxD จะนำไปใช้เพื่อเป็นขาสัญญาณนาฬิกาในการให้จังหวะการเลื่อนข้อมูลกับวงจรเลื่อนบิทภายนอก สำหรับอัตราการเลื่อนบิท (หรืออัตราบอด) จะถูกกำหนดไว้คงที่ที่ค่า 1 / 12 ของความถี่ออสซิลเลเตอร์ จากรูปที่ 3.12 แสดงให้เห็นถึงแผนภาพเวลาสัญญาณต่าง ๆ ในโหมด 0 เมื่อมีการรับและการส่งข้อมูล 1 ไบท์โดยสัญญาณนาฬิกาในการเลื่อนบิทนี้จะเกิดขึ้นภายในตัวของ 8051 เอง และมีจุดประสงค์เพื่อนำไปใช้สำหรับวงจรชิฟรีจิสเตอร์ภายนอกเท่านั้น



รูปที่ 3.12 แสดงแผนภาพเวลาของสัญญาณอนุกรมโหมด 0

สัญญาณนาฬิกาที่สร้างขึ้นทางขาสัญญาณ TxD นี้จะสลับค่าไปมาจากระดับลอจิกสูงไปต่ำในราวช่วงใกล้เคียงกับเวลาขอบขาลงของสัญญาณ ALE ซึ่งอยู่ในคาบเวลาออสซิลเลเตอร์ที่ 15 ภายหลังจากที่ได้ทำคำสั่งการโอนย้ายข้อมูลมายังรีจิสเตอร์ SBUF หรือคำสั่งที่ทำให้ค่าแฟล็กสถานะ RI เป็นค่า 0 หลังจากนั้นสัญญาณนาฬิกานี้ก็จะเปลี่ยนแปลงอีกครั้งราวช่วงใกล้เคียงกับเวลาขอบขาลงของสัญญาณ ALE ในคาบเวลาออสซิลเตอร์หลังจากนั้นอีก 6 คาบ และจะดำเนินไปในลักษณะเช่นนี้จนกระทั่งข้อมูลทั้ง 8 บิทได้ถูกส่งออกไปเรียบร้อยแล้ว เมื่อสัญญาณขอบขาขึ้นของสัญญาณนาฬิกานี้เกิดขึ้นครบจำนวน 8 ครั้งแล้วจึงจะมีผลทำให้แฟล็กสถานะ TI หรือ RI มีค่าเป็น 1 ขึ้นและสถานะของขอบสัญญาณ TxD ก็จะเป็นระดับลอจิกสูงไปโดยตลอด

ข้อมูลที่จะส่งออกไปภายนอกจะถูกเลื่อนบิทนัยสำคัญต่ำออกไปก่อน โดยจะเริ่มขึ้นในเวลาเริ่มต้นของคาบออสซิลเลเตอร์ ภายหลังจากที่ได้ทำคำสั่งการโอนย้ายข้อมูลมายังรีจิสเตอร์ SBUF สำหรับบิทแรกของข้อมูลที่รับเข้ามานั้นจะถูกแลตช์ไว้ด้วยขอบขาขึ้นของสัญญาณนาฬิกาในคาบเวลาออสซิลเลเตอร์ที่ 24 ภายหลังจากที่มีการกำหนดให้แฟล็กสถานะ RI เป็นค่า 0 หลังจากนั้นในเวลาคาบเวลาออสซิลเลเตอร์อีก 12 คาบถัดมาก็จะได้รับบิทต่อไป ซึ่งจะดำเนินเช่นนี้จนครบ 8 บิท

3.3.2 พอร์ทอนุกรมโหมด 1

การทำงานในโหมด1 เป็นการสื่อสารข้อมูลอนุกรมจำนวนจำนวน 10 บิท ประกอบด้วยบิทเริ่มต้นจำนวน 1 บิท บิทข้อมูลจำนวน 8 บิทและบิทสุดท้ายอีก 1 บิท ดังแสดงในรูปที่ 3.13 โดยข้อมูลจะถูกส่งออกทางขาสัญญาณ TxD และรับเข้ามาทางขาสัญญาณ ในส่วนของข้อมูล 8 บิทที่ได้รับหรือทำการส่งออกจะเป็นบิทนัยสำคัญต่ำเป็นลำดับแรก และบิทสุดท้ายของข้อมูลที่รับเข้ามาจัดเก็บไว้ในบิท RB8 ภายในรีจิสเตอร์ SCON สำหรับอัตราความเร็วในการส่งข้อมูลของโหมด 1 นั้นสามารถกำหนดเลือกได้

รูปที่ 3.13 รูปแบบของสัญญาณข้อมูลอนุกรมในโหมด 1 ซึ่งมีลักษณะเดียวกับรูปแบบของการสื่อสารแบบอะซิงโครนัสที่ใช้ข้อมูล 8 บิท บิทเริ่มต้นและบิทสุดท้ายอย่างละ 1 บิท

3.3.2.1 อัตราการส่งข้อมูลอนุกรมโหมด 1

กรณีใช้ **Timer 1** ทำงานในโหมด 2 (8 - bit automatic reload)

ความถี่อัตราบอด = (2^{SMOD} / 32) * (ความถี่ออสซิลเลเตอร์ / (12 * (256 - TH 1)))

โดย SMOD เป็นค่าของบิทภายในรีจิสเตอร์ PCON (มีค่าเป็น 0 หรือ 1)

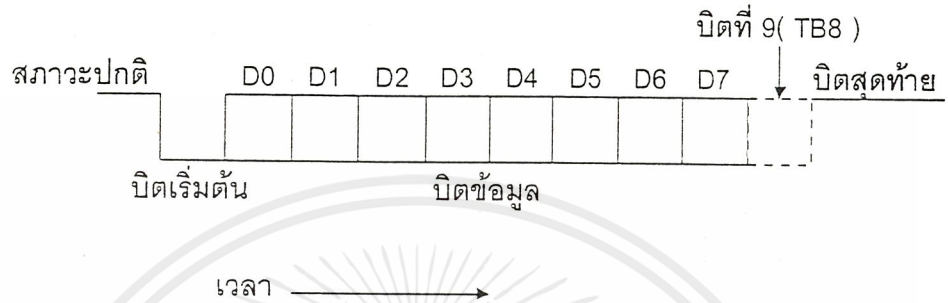
TH 1 เป็นค่าภายในรีจิสเตอร์ TH 1 ซึ่งใช้เป็นค่าสำหรับ Reload

กรณีใช้ **Timer 2** ทำงานในโหมดอื่น ๆ ที่ไม่ใช่โหมด 2

ความถี่อัตราบอด = (2^{SMOD} / 32) * อัตราการโอเวอร์โฟลว์ของไทม์เมอร์ 1

<u>3.3.3 พอร์ทอนุกรมโหมด 2 และ 3</u>

การทำงานโหมด 2 หรือโหมด 3 ของพอร์ทอนุกรมจะทำการรับ/ ส่งข้อมูลจำนวน 11 บิทเช่นเดียวกัน ซึ่งประกอบด้วย บิทเริ่มต้น บิทข้อมูลจำนวน 8 บิท บิทข้อมูลบิทที่ 9 และบิทสุดท้ายดังแสดงในรูปที่ 3.14 แต่สำหรับโหมด 3 จะสามารถเปลี่ยนแปลงอัตราการส่งข้อมูลได้ไม่ได้ถูกกำหนดไว้คงที่เช่นในโหมด 2



รูปที่ 3.14 แสดงรูปแบบของสัญญาณอนุกรมในโหมด 2 ซึ่งมีลักษณะรูปแบบการสื่อสารที่ใช้บิทข้อมูลจำนวน 9 บิท บิทเริ่มต้น และบิทสุดท้ายอย่างละ 1 บิท

อัตราบอดของโหมด 2

ความถี่อัตราบอด = (2^{SMOD} / 32) * ความถี่ออสซิลเลเตอร์

อัตราบอดของโหมด 3

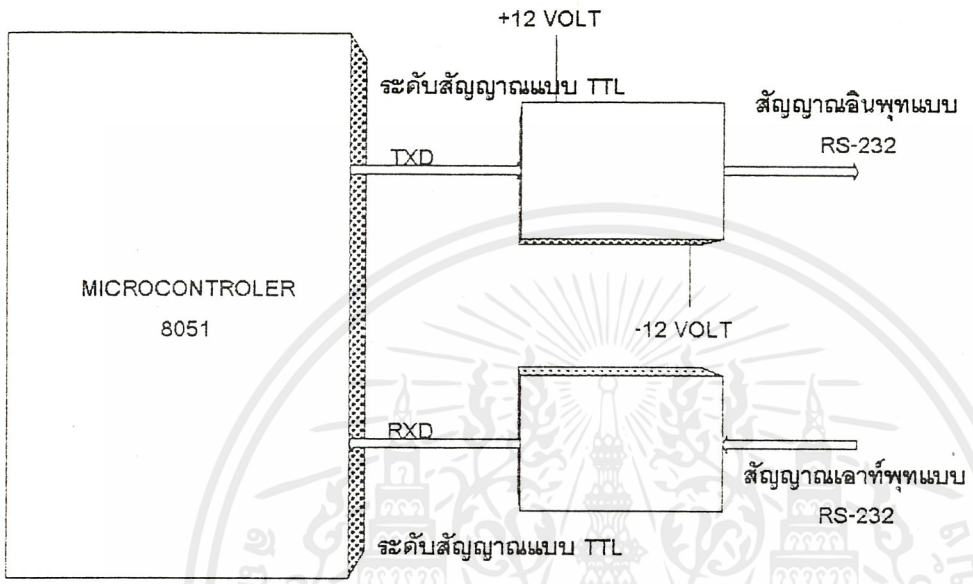
ใช้อัตราการโอเวอร์โฟลว์ของไทม์เมอร์ 1 หรือ ๆ ไทม์เมอร์ 2 โดยการคำนวณอัตราบอดเช่นเดียวกับของพอร์ทอนุกรมที่ทำงานในโหมด 1

การส่งข้อมูลอนุกรมในโหมด 2 และ 3 จะต้องนำค่าข้อมูลนั้นไปเก็บยังรีจิสเตอร์ SBUF สำหรับค่าของบิทที่ 9 ที่เพิ่มขึ้นนั้นนำมาจากค่าของบิท TB8 ภายในรีจิสเตอร์ SCON ซึ่งจะต้องได้รับการกำหนดค่าจากผู้ใช้งาน เมื่อข้อมูลถูกเลื่อนบิทส่งออกไปภายนอกเรียบร้อยแล้ว แฟล็กสถานะ T1 จึงจะมีค่าเป็น 1 เช่นเดียวกับโหมดอื่น ๆ ที่ผ่านมา และผู้ใช้จะต้องทำการเปลี่ยนกลับให้เป็นค่า 0 ตามเดิม สำหรับการรับข้อมูลจะถูกนำมาเก็บไว้ภายในรีจิสเตอร์ SBUF เช่นเดียวกันโดยค่าของบิทที่ 9 จะนำไปเก็บไว้ยังบิท RB8 ภายในรีจิสเตอร์ SCON

<u>3.4 การเชื่อมต่อแบบมาตรฐาน RS- 232 C กับวงจรอีมูลเลเตอร์</u>

ในการเชื่อมต่อแบบอนุกรมเข้ากับอุปกรณ์คอมพิวเตอร์ต่าง ๆ มักจะกำหนดใช้การเชื่อมต่อตามมาตรฐาน RS-232C ทั้งนี้เพื่อให้มีการใช้งานเส้นสัญญาณหรือรูปแบบของตัวเชื่อมต่อที่สอดคล้องกัน จะได้ลดปัญหาการเข้ากันไม่ได้ระหว่างสัญญาณของอุปกรณ์ที่มาเชื่อมต่อกัน เนื่องจากระดับโวลเตจที่ใช้และการแทนความหมายของระดับลอจิกตามมาตรฐานนี้แตกต่างกันไปจากที่ใช้งานกันในระบบดิจิตอลทั่วไปโดยระดับสัญญาณของ RS-232 เป็นแบบไบโพลาร์ (bipolar) ระดับโวลต์เตจทางด้านลบช่วง - 3V ถึง -20V แทนค่าลอจิก 1 และโวลเตจทางด้านบวกช่วง 3V ถึง 20V แทนลอจิก 0ดังนั้นจะเห็นได้ว่ามีความจำเป็นต้องเพิ่มเติมอุปกรณ์หรือวงจรพิเศษเข้าไป เพื่อเปลี่ยนระดับโวลเตจจากระบบ 0V ถึง 5 V จากขาสัญญาณของ 8051

เป็นระดับโวลเตจที่สูงกว่าค่า 3V หรือต่ำกว่า -3V ดังในรูปที่ 3.15 ซึ่งแสดงให้เห็นว่าระดับสัญญาณแบบ TTL จากขาสัญญาณ TxD และ RxD ของ 8051 จะต้องถูกปรับเปลี่ยนไปเป็นระดับสัญญาณ RS-232 ก่อน ที่จะทำการส่งออกไปในสายนำสัญญาณต่อไป

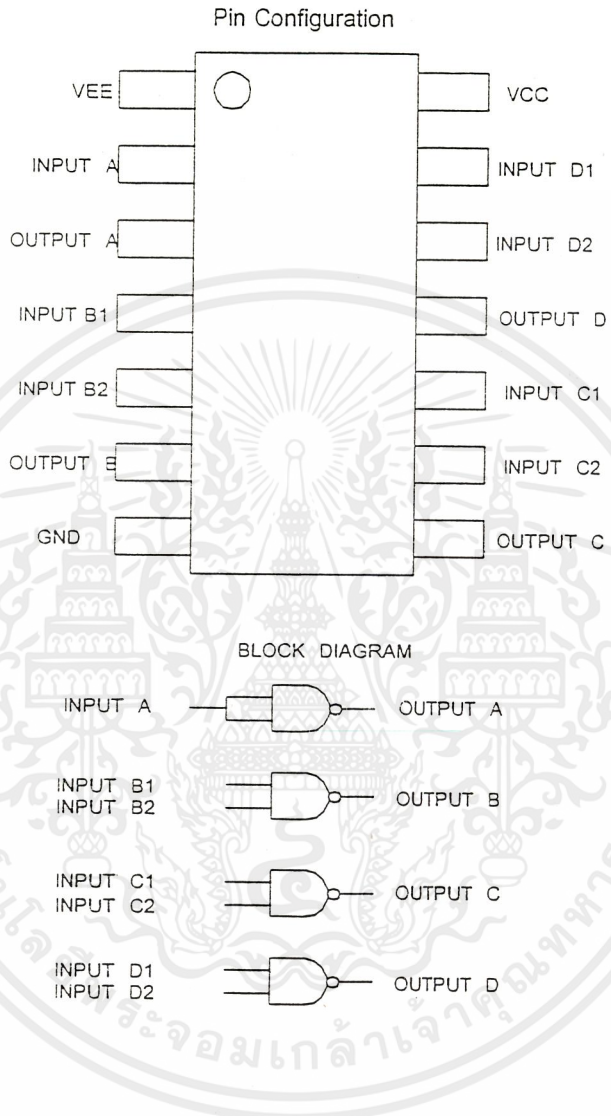


รูปที่ 3.15 แสดงให้เห็นถึงแนวการเปลี่ยนแปลงระดับสัญญาณ

โดยทั่วไปรูปแบบของวงจรก็สามารถทำได้ในหลายลักษณะ ทั้งการออกแบบสร้างวงจรเปลี่ยนระดับสัญญาณโดยการใช้ทรานซิสเตอร์หรือใช้ไอซีวงจรรวม ดังเช่นในรูปที่ 3.16 และ 3.17แสดงให้เห็นการนำไอซีวงจรรวมเบอร์ MC1488 เปลี่ยนแปลงระดับสัญญาณจาก TTL กับ RS-232 สำหรับวงจรทางด้านส่ง (driver) และเบอร์ MC1489 สำหรับวงจรทางด้านรับ (Receiver) เพื่อเปลี่ยนจากระดับ RS-232 เป็น TTL

MC1488

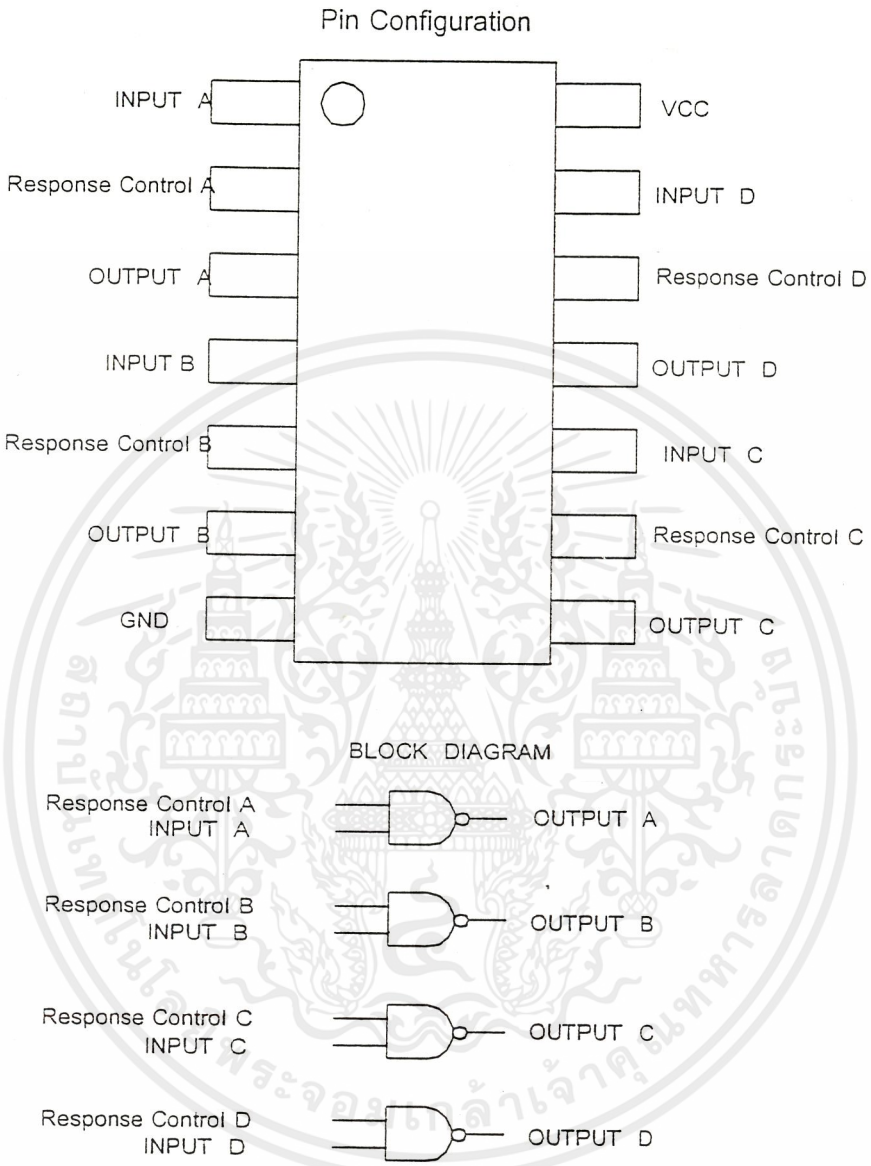
QUAD LINE DRIVER RS-232C



รูปที่ 3.16 แสดงแผนภาพวงจรขับสัญญาณทางด้านส่ง (TTL to RS-232 converter) โดยการใช้ไอซีเบอร์ MC1488

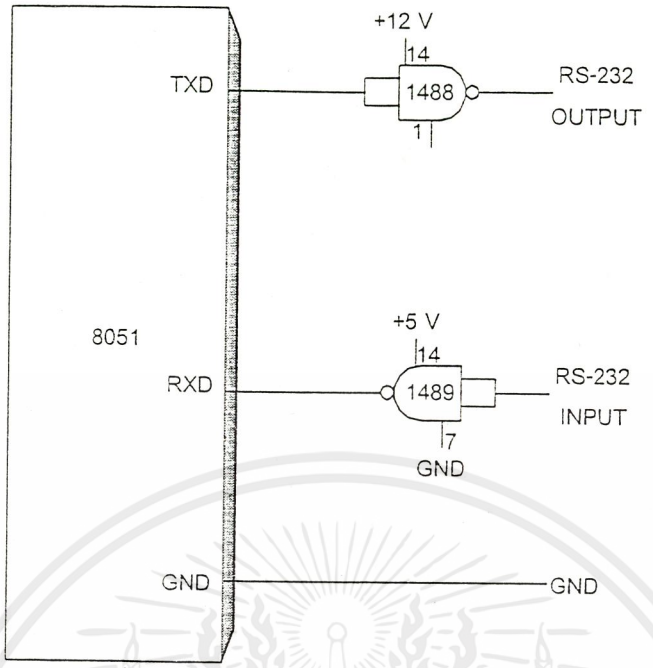
MC1489

QUAD LINE DRIVER RS-232C



รูปที่ 3.17 แสดงแผนภาพวงจรขับสัญญาณทางด้านรับ (RS232 to TTL converter) โดยการใช้ไอซีเบอร์ MC 1489

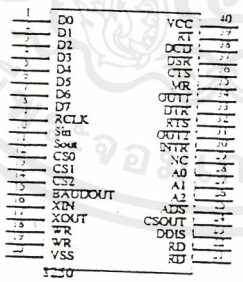
การนำเอาไอซี 1488 และ 1489 ไปใช้งานนั้นสามารถนำไปใช้งานตามรูปที่ 3.18 ซึ่งจะสังเกตเห็นว่ามีการใช้เส้นสัญญาณติดต่อพื้นฐานระหว่างกันเพียงสามเส้นเท่านั้นคือ เส้นสัญญาณสำหรับการส่งข้อมูล (Tx) ให้กับอุปกรณ์อื่น เส้นสัญญาณสำหรับการรับข้อมูล (Rx) ที่ส่งมาจากอุปกรณ์อื่นและสัญญาณกราวด์ (GND) เท่านั้น



รูปที่ 3.18 วงจรแสดงการเชื่อมต่อไอซี MC 1488 และ MC1489 เพื่อทำหน้าที่ปรับระดับสัญญาณอนุกรมให้เหมาะสม

3.5 8250 (UART)

8250 เป็นไอซีขนาด 40 ขามีการทำงานเพื่อควบคุมสิ่งต่าง ๆ ที่เกี่ยวกับการสื่อสารแบบอนุกรมได้หมด โครงสร้างทางฮาร์ดแวร์ของอะแดปเตอร์การ์ดนี้จึงไม่ยุ่งยากมากนัก การจัดวางขาของไอซี 8250 มีรายละเอียดดังรูปที่ 3.19



รูปที่ 3.19 แสดงการจัดวางขาของไอซี 8250

3.5.1 การใช้งานรีจิสเตอร์ต่าง ๆ บน 8250

เพียงจากการใช้งานบอร์ดอะแดปเตอร์สื่อสารที่จะต้องโปรแกรมค่าไมโครโค้ดเข้าไปใน 8250 ก่อน ดังนั้นผู้ใช้งาน 8250 จำเป็นต้องเข้าใจว่ารีจิสเตอร์ของ 8250 มีความหมายอย่างไรดังนี้

<u>รีจิสเตอร์ควบคุมสายสื่อสาร</u> (line control register) ในการควบคุมรูปแบบของข้อมูลแบบอะซิงโครนัสนั้น ผู้โปรแกรมจะต้องกำหนดค่าลงในรีจิสเตอร์ควบคุมสายสื่อสารรีจิสเตอร์ตัวนี้มี 8 บิทโดยแต่ละบิทมีความหมายดังนี้

บิท 0 กำหนดความยาวของข้อมูล WLSo

บิท 1 กำหนดความยาวของข้อมูล WLS

บิท 2 จำนวนสต๊อปบิต (STB)

บิท 3 พาริตี้อีนาเบิล (REN)

บิท 4 เลือกพาริตี้คู่ (EPS)

บิท 5 การแทรกพาริตี้ (stick parity)

บิท 6 เซตเบรก

บิท 7 DLAB (divisor latel access bit)

<u>การกำหนดอัตราบอด</u> (baud rate generator) อัตราบอดได้รับการกำหนดเทียบกับสัญญาณนาฬิกา 1.8432 MHz และสามารถโปรแกรมตัวหาร ได้ตั้งแต่หนึ่งถึงสองกำลังสิบหกลบหนึ่ง ค่าความถี่เอาท์พุทของตัวกำหนดอัตราบอดมีค่าเท่ากับ 16* อัตราบอด ดังนั้นตัวหาร = ความถี่สัญญาณนาฬิกา / (อัตราบอด * 16) การกำหนดอัตราบอดด้วยการกำหนดตัวหารนี้ตัวหารจึงเป็นค่าที่กำหนดในรีจิสเตอร์ 2 ตัว ตัวหารนี้จะต้องถูกกำหนดค่าก่อนแล้วโปรแกรมลงมาในรีจิสเตอร์นี้ การกำหนดต้องให้ DLAB = 1 แล้วให้ลดลงมาในรีจิสเตอร์ 3F8 ซึ่งเรียกกันว่า LSB ของตัวหารส่วน 3F9 เมื่อ DLAB = 1 จะเป็นค่าของตัวหาร MSB ค่าของตัวหารเมื่อเทียบกับสัญญาณ 1.8432 MHz เป็นดังตารางในรูปที่ 3.20

อัตราบอด	ตัวหาร		ค่าผิดพลาด
	ฐานสิบ	ฐานสิบหก	
110	1047	417	0.026
300	384	180	-
600	192	0C0	-
1200	96	060	-
1800	64	040	-
2400	48	030	-
3600	32	020	-
4800	24	018	-
9600	12	00C	-

รูปที่ 3.20 ตารางแสดงค่าตัวหารสำหรับการกำหนดอัตราบอด

<u>รีจิสเตอร์แสดงสถานะสายสื่อสาร</u> (line status register) รีจิสเตอร์ตัวนี้เป็นรีจิสเตอร์ที่จะให้ข้อมูลแก่ซีพียูที่เกี่ยวกับการสื่อสารข้อมูลในสายสื่อสาร ค่าของบิทต่าง ๆ ในรีจิสเตอร์เป็นดังนี้

บิท 0 บิทนี้เป็นบิทที่บอกสถานะการรับข้อมูล ถ้าเป็น ' 1' แสดงว่าการรับข้อมูลเข้ามาในบัฟเฟอร์ได้ครบทุกบิทแล้ว บิทนี้จะรีเซตให้เป็น 0 เมื่อซีพียูได้อ่านข้อมูลในบัฟเฟอร์ไปแล้ว หรือจะให้ซีพียูเขียนข้อมูลกลับมายังรีจิสเตอร์ได้

บิท 1 บิทนี้ถ้ามีค่าเป็น ' 1' แสดงว่าเกิดโอเวอร์โฟล์ (OR) กล่าวคือขณะที่มีข้อมูลที่บัฟเฟอร์แต่ซีพียูยังไม่ได้อ่านไป ปรากฏมีข้อมูลใหม่มาเขียนทับบนบัฟเฟอร์นี้ บิทนี้จะรีเซตโดยซีพียู เมื่อซีพียูอ่านค่าจากรีจิสเตอร์นี้ไปแล้ว

บิท 2 บิทนี้ถ้ามีค่าเป็น ' 1' แสดงว่าเกิดความผิดพลาดทางพาริตี้ (PE) กล่าวคือ ถ้ามีการตรวจสอบบิทพาริตี้แล้วไม่เป็นไปตามที่กำหนดไว้ บิทนี้จะได้รับการรีเซตโดยซีพียู เมื่อซีพียูอ่านค่าจากรีจิสเตอร์นี้ไปแล้ว

บิท 3 บิทที่ใช้ในการตรวจสอบความผิดพลาดทางเฟรม (FE)

บิท 4 บิทนี้เรียกว่า break interrupt (BI) บิทนี้จะได้รับการเซตให้มีค่าเป็น ' 1' ถ้าหากว่ารับข้อมูลอินพุตเป็น 0 เป็นระยะเวลายาวนานกว่าเวิร์ดของการสื่อสาร

บิท 5 บิทนี้เป็นบิทที่บอกว่า 8250 พร้อมที่จะรับข้อมูลจากสายสื่อสาร บิทนี้จะได้รับการเซตให้มีค่าเป็น '1' บิทนี้ยังคงสร้างสัญญาณอินเตอร์รัปท์เพื่อส่งไปบอกซีพียูด้วย บิทนี้จะมีสถานะเซตเมื่อมีการส่งถ่ายข้อมูลจากโฮลด์รีจิสเตอร์ไปยังชิปรีจิสเตอร์เพื่อพร้อมที่จะส่ง

บิท 6 เป็นบิทที่จะบอกว่าชิฟรีจิสเตอร์ว่างเปล่า บิทนี้จะได้รับการเซตให้มีค่าเป็น'1' เพื่อบอกว่าพร้อมส่งแล้ว

บิท 7 จะเป็น 0 ตลอด

<u>รีจิสเตอร์กำหนดอินเตอร์รัปท์ (IRR)</u> ไอซี 8250 มีขีดความสามารถในการส่งอินเตอร์รัปท์ภายในชิฟ เพื่อว่าให้มีการทำงานระหว่าง 8250 กับซีพียูเป็นไปได้อย่างมีประสิทธิภาพสูง 8250 กำหนดความสำคัญของอินเตอร์รัปท์ไว้ 4 ระดับคือ ระดับแรกสถานะการรับข้อมูลจากสายสื่อสารระดับสองการพร้อมรับข้อมูลระดับที่สาม ขณะรีจิสเตอร์โฮลดิงสำหรับส่งข่าวระดับที่สี่สัญญาณสถานะโมเด็ม

ในขณะที่มีความต้องการอินเตอร์รัปท์หลายระดับพร้อมกัน 8250 จะให้ระดับที่มีความสำคัญน้อยกว่ารอไว้ก่อน โดยเก็บสถานะการอินเตอร์รัปท์นี้ไว้ในรีจิสเตอร์กำหนดอินเตอร์รัปท์ความหมายของรีจิสเตอร์นี้มีดังนี้

บิท 0 เป็นบิทที่ใช้แสดงว่ามีอินเตอร์รัปท์เกิดขึ้นหรือไม่

บิท 1 - 2 เป็นบิทที่แสดงความหมายบอกว่าการอินเตอรรัปท์ที่เกิดขึ้นนั้นมาจากการอินเตอร์รัปท์ตามฟังก์ชั่นใด

บิท 3 -7 มีค่าเป็น '0'

<u>รีจิสเตอร์อีนาเบิลอินเตอร์รัปท์ (interrupt enable register)</u> ในcom 1 เมื่อให้ DLAB = 0 พอร์ท 3F9 จะเป็นรีจิสเตอร์อีนาเบิลอินเตอร์รัปท์ ผู้ใช้สามารถกำหนดให้เกิดอินเตอร์รัปท์หรือไม่ก็ได้ โดยการกำหนดค่าลงในรีจิสเตอร์นี้ ข้อมูลที่อยู่ในรีจิสเตอร์นี้มีความหมายดังนี้คือ

บิท 0 อีนาเบิลอินเตอร์รัปท์การพร้อมรับข้อมูล

บิท 1 อีนาเบิลอินเตอร์รัปท์โฮลดิงรีจิสเตอร์ว่าง

บิท 2 อีนาเบิลอินเตอร์รัปท์สถานะการรับข้อมูลจากสายสื่อสาร

บิท 3 อีนาเบิลอินเตอร์รัปท์สถานะโมเด็ม

บิท 4 - 7 มีค่าเป็น ' 0'

<u>รีจิสเตอร์ควบคุมโมเด็ม (modem control register)</u> รีจิสเตอร์ตัวนี้มีไว้ให้ซีพียูส่งผ่านข้อมูลมาเก็บเพื่อเป็นรหัสสำหรับควบคุมการทำงานของโมเด็ม การกำหนดพอร์ทของรีจิสเตอร์ตัวนี้คือ 3FC ข้อมูลต่าง ๆ ที่มีในรีจิสเตอร์ตัวนี้มีความหมายดังนี้

บิท 0 บิทนี้มีความหมายถึงการควบคุมสัญญาณ DTR เมื่อบิทนี้มีค่าเป็น 1 เอาท์พุตที่ DTR จะได้รับการกำหนดให้เป็น 0 และถ้าบิทนี้มีค่าเป็น 0 เอาท์พุต DTR จะได้รับการกำหนดให้เป็น 1

บิท 1 บิทนี้มี ความหมายถึงสัญญาณ RTS ซึ่งจะมีผลเหมือนกับบิท 0 ในกรณีของ DTR

บิท 2 บิทนี้ใช้ควบคุ มขา เอาท์พุต 1 ซึ่งมีผลเหมือนบิท 0

บิท 3 บิทนี้ใช้ควบคุมขาเอาท์พุต 2

บิท 4 บิทนี้จะใช้สำหรับการกำหนดวงรอบสำหรับการตรวจสอบ 8250 เมื่อบิท 4 นี้ได้รับการเซตให้เป็น 1 สิ่งที่เกิดขึ้นเป็นดังนี้ ข้อมูลที่ SOUT จะได้รับการเซตให้เป็นลอจิก 1 ขาข้อมูลอินพุต SIN จะได้รับการแยกตัวออก ข้อมูลของเอาท์พุตชิฟรีจิสเตอร์จะได้รับการป้อนกลับมายังรีจิสเตอร์ข้อมูลอินพุต ส่วนสัญญาณ CTS , DSR , RLSD , RI จะได้รับการแยกออกจากระบบแต่สัญญาณควบคุมโมเด็มคือ DTR , RTS , OUT1 , OUT2 จะต่อเข้ากับสัญญาณทั้งสี่ที่เป็นอินพุตดังนั้นจึงตรวจสอบระบบการทำงานได้

<u>3.6 การส่งข้อมูลแบบอะซิงโครนัส</u>

การสื่อสารแบบอะซิงโครนัสนั้น ข้อมูลจะถูกส่งผ่าน พอร์ทอนุกรม (Serial port) บิทต่อบิทซึ่งต่างจากการสื่อสารที่ผ่านพอร์ทขนาน (Parallel port) ที่จะมีการส่งข้อมูลทีละไบท์ และคำว่าอะซิงโครนัสนั้นเนื่องจากไม่สนใจจังหวะเวลาในการส่งข้อมูล อย่างไรก็ตามความยาวของข้อมูลจะถูกกำหนด โดยแต่ละไบท์ของข้อมูลประกอบด้วย

1. บิทเริ่มต้น 1 บิท
2. บิทข้อมูล 7 หรือ 8 บิท
3. พาริตี้บิท
4. บิทสุดท้าย 1 หรือ 2 บิท

สภาวะสายว่างนั้นแรงดันจะสูง เมื่อส่งบิท 0 แรงดันจะต่ำและ 1 แรงดันจะสูง นั่นคือเมื่อส่งสัญญาณบิทเริ่มต้นนั้น จะขับให้แรงดันสายต่ำลง จากนั้นข้อมูลก็จะถูกส่ง และจะตามด้วยพาริตี้บิทและการส่งบิทสุดท้ายส่งจะขับให้แรงดันสายต่ำลง พาริตี้บิทจะใช้ในการตรวจสอบความผิดพลาดในการส่ง การใช้พาริตี้คู่หรือคี่ ค่าพาริตี้จะถูกเซตให้ค่าไบท์ที่ส่งรวมกับค่าพาริตี้นั่นออกมาเป็นเลขคี่

สำหรับการสื่อสารนี้จะผ่าน RS 232 ซึ่งจะประกอบด้วยตัวคอนเนคเตอร์ 25 ขา ในแต่ละปลายซึ่งขาที่สำคัญในการสื่อสารมีดังนี้

สัญญาณ	ชื่อย่อ	ขาของคอนเนคเตอร์
Request to send	RTS	4
Clear to send	CTS	5
Data set ready	DSR	6
Data terminal ready	DTR	20
Transmit data	TxD	2
Receive data	RxD	3
Ground	GND	7

3.6.1 การแอดเซส PC Serial Port โดยอาศัย BIOS

การแอสเซสพอร์ทอนุกรมบนเครื่องคอมพิวเตอร์สามารถทำได้โดยใช้ดอส หรือ รอม- ไบออสหรือผ่านทางดอสและไบออสโดยการควบคุมฮาร์ดแวร์ การแอสเซสโดยใช้ดอสนั้นมีข้อเสียเนื่องจากดอสไม่สามารถเตรียมการป้อนกลับสถานะของพอร์ทนั่นคือมันจะเตรียมการเขียนและอ่านโดยไม่มีการตอบรับ วิธีที่นิยมคืออินเตอร์รัปท์รอมไบออส (interrupt ROM - BIOS)

ROM - BIOS ได้เตรียมบริการในการแอสเซสไว้ 4 อย่างโดยการ อินเตอร์รัปท์ 14H ดังนี้

1. สถานะเริ่มต้นพอร์ท (port initialization)

ก่อนที่จะมีการใช้งาน พอร์ทอนุกรมเราจำเป็นจะต้องมีการปรับค่าเริ่มแรกโดยการ อินเตอร์รัปต์ 14H และบริการ ที่ 0 ดังเช่นการอินเตอร์รัปท์อื่น ๆ รีจิสเตอร์ AL จะใช้สำหรับเก็บค่าบริการ AL จะใช้เก็บค่าเริ่มแรก ซึ่งจะแปลงให้อยู่ในรูปข้อมูล 1 ไบท์ดังนี้

บิท 0-1 บิทข้อมูล

บิท 2 บิทสต๊อป

บิท 3-4 บิทพาริตี้

บิท 5-7 บิทที่ใช้กำหนดอัตราบอด

BAUD	BIT PATTERN	PARITY	BIT PATTERN
9600	1 1 1	NO PARITY	00 หรือ 10
4800	1 1 0	ODD	01
2400	1 0 1	EVEN	11
1200	1 0 0		
600	0 1 1		
300	0 1 0		

สำหรับจำนวนของบิทสตอปจะตัดสินโดยบิทที่ 2 ถ้าบิทที่2 = 1 จะใช้ 2 บิทในการหยุด และบิทที่ 2=1 จะใช้ 1 บิทในการหยุดและสุดท้ายเป็นบิทข้อมูลซึ่งแสดงโดยบิทที่ 0 และ 1 อย่างไรก็ตามจะใช้ 10 แทนข้อมูล 7 บิทและ 11 แทนข้อมูล 8 บิทเท่านั้น

<u>2.การส่งข้อมูล</u> อินเทอร์รัปท์ 14H และบริการที่ 1 จะใช้ในการส่งข้อมูล 1 ไบท์ผ่านพอร์ทอนุกรมที่ระบุไว้ใน DX และข้อมูลที่จะส่งจะต้องเก็บไว้ใน AL สำหรับสภาวะการส่งจะถูกเก็บไว้ใน AH และฟังก์ชั่น sport( ) จะใช้ในการส่งข้อมูล 1 ไบท์ผ่านพอร์ทที่ระบุไว้

```
/* send a character out the serial  port*/
void sport (port,c )
int port; char c;
{
union   REGS  r;
r.x.dx   =  port   ;  /*serial  port */
r.h.ah   = 1         ; /*send character function */
r.h.al    = c         ; /*char to send */
int 86  ( 0x14,&r,&r);
if (r.h.ah & 128 )
printf ( "send error detected in serial port.");
exit (1 ); }
```

3.การเช็คสถานะของพอร์ท อินเทอร์รัปต์ 14H และบริการที่ 3 จะใช้สำหรับตรวจสอบสภาวะของพอร์ทที่ต้องการตรวจสอบจะถูกระบุใน DX โดยที่ AH and AL จะระบุสภาวะของพอร์ทนั้นดังนี้คือ

LINE STATUS (AH)	BIT	MODEM STATUS	BIT
MEANING WHEN SET		MEANING WHEN SET	
DATA READY	0	Change in clear to send	0
OVERRUN ERROR	1	change in data set ready	1
PARITY ERROR	2	trailing edge ring detector	2
FRAMING ERROR	3	change in line signal	3
BREAK-DETECT ERROR	4	clear to send	4
TRANSFER HOLDING		data set ready	5
REGISTER EMPTY	5	ring indicator	6
TRANSFER SHIFT	6	line signal detected	7
TIME - OUT ERROR	7		

BIOS อินเตอร์รัปท์ 14 H , บริการที่ 2 ใช้ในการอ่านข้อมูลจากพอร์ทอนุกรม โดยที่พอร์ทอนุกรมที่ใช้จะระบุไว้ใน DX และอักษรที่อ่านได้จะถูกเก็บไว้ใน AL และจากการส่งอักษรนั้นสภาวะของบิท 7 ใน AH แสดงถึงว่าการส่งนั้นสำเร็จหรือผิดพลาด

ฟังก์ชั่น rport( ) จะใช้ในการอ่านไบท์ ข้อมูลจากพอร์ทที่ระบุไว้

```
/*  read a character from a byte */
rport (port )
int  port;
{
union  REGS  r;   /* wait for a character */
while (!(check_stalat (port) & 256))
if (kbit( ) {  /* abort on keypress */
getch( );
exit (1);   }
r.x.dx   = port;
r.h.ah   = 2;   /* read character function*/
int86 (0x14,&r,&r);
if (r.h.ah& 128 )
pimtf ( "read error detected in serial port");
return r.h.al;
}
```

บทที่ 4 การทดลองและผลการทดลอง

ทำการเชื่อมต่อบอร์ดกับเทอร์มินัลเมื่อทำการ คอมไพล์ในภาษา C จะแสดงหน้าจอดังนี้

```
**********************************************************************************
*                          8051  EMULATOR PROJECT                                *
**********************************************************************************
        USE COMMUNICATION PORT :
        PRESS THE RESET BUTTON ON EMULATOR BOARD
        AND PRESS ANY KEY TO CONTINUE.....
```

เมื่อสามารถติดต่อกันได้จะขึ้น PROMPT> เตรียมรับตัวอักษร
เราสามารถทดสอบคำสั่งต่าง ๆ ที่ใช้ในการอีมูลเลทได้ดังนี้

C	ตรวจสอบการเชื่อมต่อ
b	เปลี่ยนค่า baudrate
D	การนำเอาข้อมูลในหน่วยความจำขึ้นมาแสดงบนจอภาพ
G	รันโปรแกรม
R	ดูค่าภายในรีจิสเตอร์
S	ทำงานทีละคำสั่ง
J	กระโดดไปทำงานที่แอดเดรสถูกเรียก
H	แสดงคำสั่งที่ใช้ในการอีมูลเลท

4.1 ทดสอบคำสั่ง C

เป็นการตรวจสอบการเชื่อมต่อกันระหว่างบอร์ดกับเทอร์มินัล

```
PROMPT> C
WAIT WHILE CHECKING CONNECTION.....CONNECTION OK!
```

4.2 ทดสอบคำสั่ง b

เป็นการเปลี่ยน baudrate ค่าใหม่

```
PROMPT>b  4800
BUADRATE NOW CHANGED TO 4800
```

4.3 <u>ทดลองคำสั่ง L</u>

ทำการโหลดไฟล์ไปยังแรม

PROMPT>L

FILE NAME : LOAD.HEX

4.4 <u>ทดลองคำสั่ง D</u>

การนำเอาข้อมูลภายในหน่วยความจำขึ้นแสดงที่จอภาพ

PROMPT>D 8000

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8000	7A	1	7B	2	7C	3	7D	4	7E	5	7F	6	74	0	4	4
8010	4	4	4	0	F6	7F	CF	E4	1	EF	FF	FF	FF	FF	FF	FF
8020	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
8030	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

DUMP MEMORY FINISH

4.5 <u>ทดสอบคำสั่ง E</u>

ทำการเปลี่ยนแปลงข้อมูลที่ตำแหน่งแอดเดรสที่รับเข้ามา

โดยเราต้องการเปลี่ยนข้อมูลภายในหน่วยความจำ 8005 จาก 03 เป็น07

PROMPT> E 8005

EDIT ADDRESS 8005 : 03) 07

PROMPT> D 8000

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8000	7A	1	7B	2	7C	7	7D	4	7E	5	7F	6	74	0	4	4
8010	4	4	4	0	F6	7F	CF	E4	1	EF	FF	FF	FF	FF	FF	FF
8020	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
8030	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF

DUMP MEMORY FINISH

4.6 <u>ทดสอบคำสั่ง G</u>

ทำการรันโปรแกรมถึงที่ตำแหน่งแอดเดรสใด

PROMPT>G 8000

NOW GO!

4.7 ทดสอบคำสั่ง R

ทำการดูค่าภายในรีจิสเตอร์ที่ตำแหน่งนั้นแอดเดรสนั้น

PROMPT>R

R0= 5C R1=A3 R2=0 R3=2 R4=3 R5=4 R6=7F R7=7B

ACC=0 PSW=0 DPH=FF DPL=F4 PC=8000

4.8 ทดสอบคำสั่ง S

ทำการรันโปรแกรมทีละคำสั่ง

PROMPT>S

SINGLE STEP

PROMPT>R

R0= 5C R1=A3 R2=1 R3=2 R4=3 R5=4 R6=7F R7=7B

ACC=0 PSW=0 DPH=FF DPL=F4 PC=8002

PROMPT>S

SINGLE STEP

PROMPT>R

R0= 5C R1=A3 R2=1 R3=2 R4=3 R5=4 R6=7F R7=7B

ACC=0 PSW=0 DPH=FF DPL=F4 PC=8004

ผลการทดลองใช้งาน

ผลที่ได้จากการทดลองใช้งานถือว่าเป็นไปตามทฤษฎีที่ออกแบบไว้แล้ว โดยเริ่มจากการเขียนชุดคำสั่งตัวอย่างขึ้นมาแล้วทำการคอมไพล์ให้เป็นไฟล์ที่มีการวางรหัสของคำสั่งเป็นแบบเลขฐานสิบหก (hex intel format *.hex) จากนั้นทำการดาวน์โหลดและทดลองคำสั่งต่าง ๆ ดังผ่านมาแล้ว การทดลองในลักษณะนี้เป็นการนำเอา 8051 อีมูลเลเตอร์มาใช้งานเพื่อช่วยในการพัฒนาโปรแกรมต่าง ๆ ตามแต่ผู้ใช้จะเป็นผู้เขียนขึ้นมา ผู้ใช้สามารถทำการทดสอบความถูกต้องของโปรแกรมได้ก่อนที่จะนำไปใช้งานจริง ขั้นตอนการสร้างหรือเตรียมโปรแกรมเพื่อใช้กับ 8051 อีมูลเลเตอร์จะเป็นดังนี้

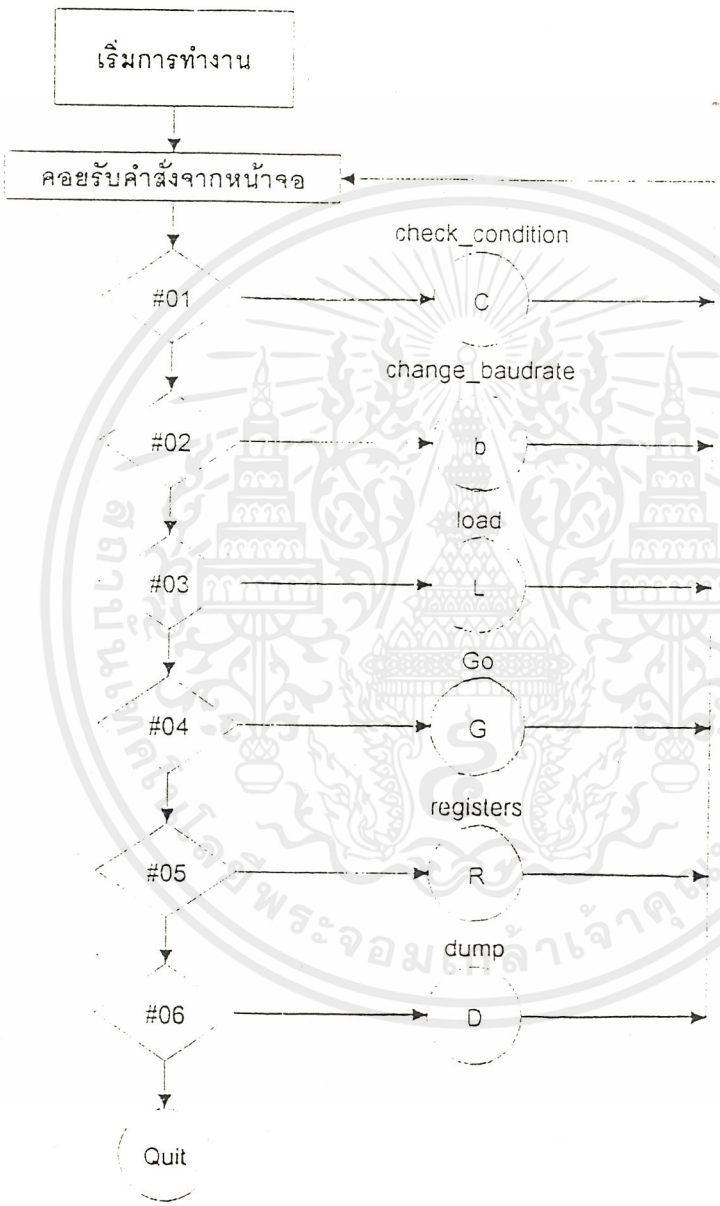
1. ใช้ EDITOR DOS พิมพ์โปรแกรมที่ต้องการ
2. ทำการคอมไพล์โปรแกรมโดยใช้คอมไพเลอร์ของ MCS - 51 ของบริษัท SAX51 กำหนดให้วางไฟล์ในรูปของ HEX INTEL FORMAT ดังตัวอย่างดังรูป
3. ทำการดาวน์โหลดลงไปใน 8051 อีมูลเลเตอร์ตามขั้นตอนต่าง ๆ

เมื่อนำเอา 8051 อีมูลเลเตอร์ไปใช้อีมูลเลตเข้าไปในแผ่นวงจรที่ใช้งานอื่น ๆ เพื่อทำการตรวจสอบการทำงานพบว่ามีข้อกำหนดเกิดขึ้นคือมีความล่าช้าในการส่งมากแต่ผลที่ได้ยังถูกต้องแต่อาจมีปัญหาเรื่องการหยุดโปรแกรมที่อาจเสียหายเนื่องจากสภาพต่าง ๆ ที่เกิดจากการต่อตัวอุปกรณ์อีมูลเลเตอร์ เพราะทางผู้ทำไม่ได้ทำแผ่นปรินท์ขึ้นมาเพียงแต่ทำลงในบอร์ดอเนกประสงค์จึงมีปัญหาทางด้านฮาร์ดแวร์เป็นอย่างมาก

บทที่ 5 บทวิจารณ์และสรุป

ในปัจจุบันนี้อุปกรณ์ที่ใช้ในการตรวจสอบและพัฒนาระบบไมโครคอมพิวเตอร์ ล้วนใช้เทคนิคของการอีมูลเลทั้งสิ้น ทั้งนี้เนื่องจากข้อดีของการอีมูลเลทก็คือ สามารถตรวจหาจุดผิดพลาดได้โดยง่ายและไม่ยุ่งยากซับซ้อน โดยการอีมูลเลทนั้นใช้ไมโครคอมพิวเตอร์เป็นส่วนควบคุมหลัก ตัวอย่างเช่น ในกรณีที่อุปกรณ์ที่ใช้เกิดการขัดข้อง การตรวจสอบแก้ไขจะมีขั้นตอนที่ยุ่งยากโดยเริ่มจากการตรวจหาจุดผิดพลาดทางฮาร์ดแวร์ก่อน โดยอาจใช้ออสซิลโลสโคปในการหาจุดเสียที่เกิดขึ้นตามแผนผังของวงจรที่มีอยู่ โดยถ้าอุปกรณ์ใดทำงานบกพร่องก็จะทำการเปลี่ยนใหม่ แต่ปัญหาจะยุ่งยากถ้าการผิดพลาดเกิดขึ้นเนื่องจากสาเหตุทางซอฟแวร์เพราะจะทำให้สัญญาณลอจิกต่าง ๆ ที่ป้อนเข้าไปควบคุมให้ฮาร์ดแวร์ทำงานจะผิดพลาดหมด ดังนั้นจึงนำมาบันทึกไว้ในหน่วยความจำคำสั่งใด เกิดข้อผิดพลาดอย่างไร ฯลฯ

ภาคผนวก



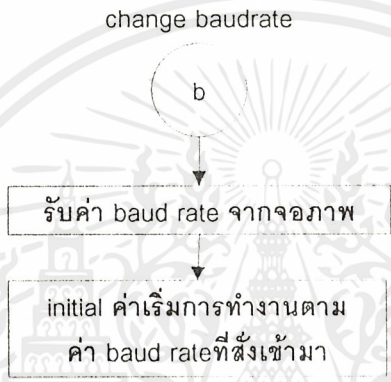
บล็อกไดอะแกรมแสดงการรับคำสั่งจากเทอร์มินัลแล้วทำการเชื่อมต่อ

C : Check condition

ตรวจสอบความถูกต้องของการรับส่ง

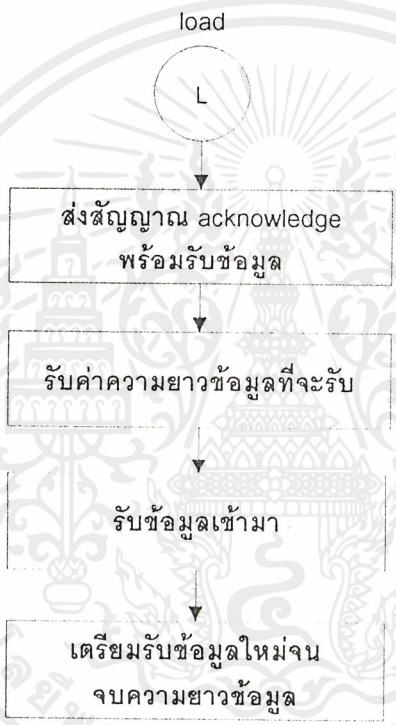
C (CHECK STATUS) ตรวจสอบการรับส่งระหว่างบอร์ดอีมูลเลเตอร์กับเทอร์มินัล

บล็อกไดอะแกรมแสดงการทำงานของคำสั่ง C



b (CHANGE BAUDRATE) เปลี่ยนอัตราบอดในการทำงาน

บล็อกไดอะแกรมแสดงการทำงานของคำสั่ง b



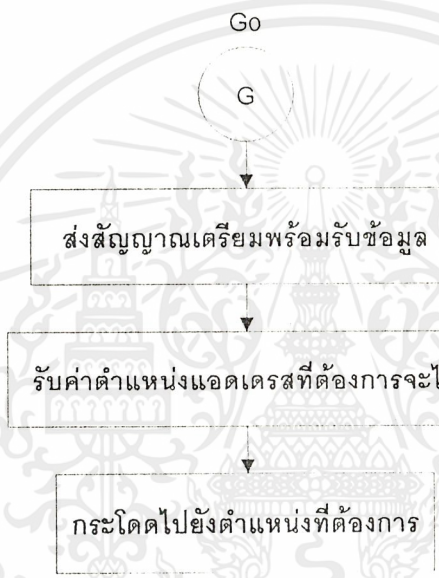
L (LOAD) ทำการโหลดโปรแกรมที่จะใช้ในการอีมูลเลท

บล็อกไดอะแกรมแสดงการทำงานของคำสั่ง L

Dump แสดงหน่วยความจำจากตำแหน่งที่ต้องการ

D (DUMP) แสดงหน่วยความจำจากตำแหน่งที่ต้องการ

บล็อกไดอะแกรมแสดงการทำงานของคำสั่ง D



G(GO) ทำการรันโปรแกรมที่ตำแหน่งที่กำหนด

ปล็อกไดอะแกรมแสดงการทำงานของคำสั่ง G

R (REGISTER) ทำการดูค่าต่าง ๆ ที่อยู่ภายในรีจิสเตอร์

บล็อกไดอะแกรมแสดงการทำงานของคำสั่ง R

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>

int portaddr;
unsigned char code;

/* change ascii for hex number to integer */

charhextoint(c)
char c;
{
   if((c>47)&&(c<58)) { c=c-48; }
   else { if((c>64)&&(c<71)) { c=c-55; }
        else if((c>96)&&(c<103)) { c=c-87; }
           else { c=100; }
      }
   return(c);
}

/* initial uart */

port_init(lsb,msb)
unsigned char lsb,msb;
{
   outportb(portaddr+3,0x80);
   outportb(portaddr,lsb);
   outportb(portaddr+1,msb);
   outportb(portaddr+3,code);
}

unsigned char check_status()
{
  return(inportb(portaddr+5));
}

sport(c)
char c;
{
  outportb(portaddr+3,code);
  outportb(portaddr,c);
}

unsigned char rport()
{
  outportb(portaddr+3,code);
  return(inportb(0x2f8));
}


int waitread()
{
 int data;
 while(((int)(check_status()&1)!=1)&&(!kbhit()))
 {}
 data=rport();
 return(data);
}



int send_recieve(c)
unsigned char c;
{
  int i=0;
  int d;

  if((int)(check_status()&1)==1) { rport(); };
  sport(c);
  while(((int)(check_status()&1)!=1) && (i<=30000))
  {
    delay(50);
```

```
/*    sport(c); */
    i++;
  }
  if(i<=30000)
  {
    d=rport();
    if(d==c) {return(0);}
    else {return(1);}
  }
  else return(2);
}


int send_recieve2(c)
unsigned char c;
{
  int i=0;
  int d;

  if((int)(check_status()&1)==1) { rport(); };
  sport(c);
  while(((int)(check_status()&1)!=1) && (i<=30000))
  {
    delay(50);
/*    sport(c); */
    i++;
  }
  if(i<=30000)
  {
    d=rport();
    return(d);
  }
  else return(-1);
}


int check_connection()
{
  int i;

  printf("Wait while Checking connection...");
  if(send_recieve(0x01)==0)
  { printf("Connection Ok!\n\n");
    return(0);
  }
  else { printf("Connection Fail!\n\n");
        return(1);
       }
}


run_change_baudrate(s)
char s[80];
{
  int baud=0;
  unsigned char lsb,msb,i;

  for(i=2;i<=5;i++)
  { baud=(baud)*10+s[i]-48; }

  switch(baud)
  {
    case 110  :lsb=0x17; msb=0x04; break;
    case 300  :lsb=0x80; msb=0x01; break;
    case 600  :lsb=0xc0; msb=0x00; break;
    case 1200 :lsb=0x60; msb=0x00; break;
    case 2400 :lsb=0x30; msb=0x00; break;
    case 4800 :lsb=0x18; msb=0x00; break;
    case 9600 :lsb=0x0c; msb=0x00; break;
```

```
                result=send_recieve(value[1]);
                i++;
               };
            } else { result=1; }
          } else { result=1; };

          value[1]=fgetc(fp);
        value[1]=fgetc(fp);
          value[1]=fgetc(fp);
        value[1]=fgetc(fp);

/*      fclose(fp);

        if(result==0)
        { printf("Load success.\n\n"); }
        else { printf("Load fail!\n\n"); }  */

       } while((value[0]>0)&&(result==0)); /* end-dowhile */
       fclose(fp);
    }/* end-if */
}/* end-load */


go(s)
char s[80];
{
  unsigned char hi_byte,lo_byte;

  hi_byte=(charhextoint(s[2])*16)+charhextoint(s[3]);
  lo_byte=(charhextoint(s[4])*16)+charhextoint(s[5]);
  break_point(hi_byte,lo_byte);
  if(send_recieve(0x04)==0)
  { printf("Now Go!\n\n"); }
  else { printf("Go error!\n\n"); }
}


registers()
{ int i;
  unsigned char d;

  i=0;
  do
  {
    d=read_1byte(0xef,i);
    printf("R%d  = %2X ",i,d);
    i++;
  }while(i<=7);
  printf("\n");
  d=read_1byte(0xef,0xe0);
  printf("ACC = %2X ",d);
  d=read_1byte(0xef,0xd0);
  printf("PSW = %2X ",d);
  d=read_1byte(0xef,0x83);
  printf("DPH = %2X ",d);
  d=read_1byte(0xef,0x82);
  printf("DPL = %2X ",d);
  d=read_1byte(0xff,0xf0);
  printf("PC = %X%X",d/16,d%16);
  d=read_1byte(0xff,0xf1);
  printf("%X%X\n\n",d/16,d%16);

}


dump(s)
char s[80];
{
  unsigned i;
  unsigned char dump_mem[64],hi_byte,lo_byte;

  if(send_recieve(0x06)==0)
```

```
    default  : lsb=0x01; msb=0x00; baud=0000; break;
   }
   change_baudrate(lsb,msb,baud);
}

change_baudrate(lsb,msb,baud)
unsigned char lsb,msb;
int baud;
{
   char value[3];
   int  result,i=0;

   value[0]=0x02;
   value[1]=lsb;
   value[2]=msb;
   do
   {
     result=send_recieve(value[i]);
     i++;
   }while((result==0)&&(i<3));

   if(i>=3)
   { port_init(lsb,msb);
     printf("Baudrate now changed to %d.\n",baud);
   }
   else printf("Change baudrate error!\n");
   printf("\n");
}


load()
{
   unsigned char value[5];
   int result,i=0;
   FILE *fp;
   char fname[80];

   printf("FILE NAME : ");
   gets(fname);
   printf("\n");

   if((fp=fopen(fname,"rt"))==NULL)
   { printf("No file or error opening %s !\n",fname); }
   else
   {
      value[1]=fgetc(fp);
      value[1]=fgetc(fp);
      do
      {
       if(send_recieve(0x03)==0)
       {
          value[1]=fgetc(fp);
         value[2]=fgetc(fp);
         value[0]=charhextoint(value[1])*16+charhextoint(value[2]);
         value[1]=fgetc(fp);
         value[2]=fgetc(fp);
         value[3]=fgetc(fp);
         value[4]=fgetc(fp);
         value[1]=charhextoint(value[1])*16+charhextoint(value[2]);
         value[2]=charhextoint(value[3])*16+charhextoint(value[4]);
         if((send_recieve(value[1])==0)&&(send_recieve(value[2])==0))
         {
             value[1]=fgetc(fp);
             value[1]=fgetc(fp);
             result=send_recieve(value[0]);
             i=0;
             while((result==0)&&(i<value[0]))
             {
              value[1]=fgetc(fp);
              value[2]=fgetc(fp);
              value[1]=charhextoint(value[1])*16+charhextoint(value[2]);
```

```
   {
     hi_byte=(charhextoint(s[2])*16)+charhextoint(s[3]);
     lo_byte=(charhextoint(s[4])*16)+charhextoint(s[5]);

     if((send_recieve(hi_byte)==0) && (send_recieve(lo_byte)==0))
          {
       printf("       ");
       for(i=0;i<=15;i++)
       {printf(" %2X ",i);}
       printf("\n");

       i=0;
       do
       { dump_mem[i]=send_recieve2(0x06);
         if(i%16==0) {printf("%X%X%X%X : "
                   ,hi_byte/16,hi_byte%16,lo_byte/16,lo_byte%16);};
        printf(" %2X ",dump_mem[i]);
        if(i%16==15)
        { printf("\n");
          if(lo_byte<0xf0) {lo_byte+=0x10;}
          else { lo_byte+=0x10;
                 hi_byte++;
                }
        }
        i++;
       }while((dump_mem[i-1]>=0)&&(i<64));

       if((i>=64)&&(send_recieve(0x00)==0))
       {

        printf("Dump memory finish.\n\n");
       }
       else printf("\nDump memory error!\n\n");
     } else printf("\nDump memory error!\n\n");
   }
   else printf("\nDump memory error!\n\n");
}


write_enable()
{
  if(send_recieve(0x07)==0)
  { printf("Write enable success.\n\n"); }
  else { printf("Write enable fail!\n\n"); }
}



move()
{
  if(send_recieve(0x08)==0)
  { printf("Move success.\n\n"); }
  else { printf("Move fail!\n\n"); }
}



break_point(hi_byte,lo_byte)
unsigned char hi_byte,lo_byte;
{
  if((send_recieve(0x0b)==0)&&
     (send_recieve(lo_byte)==0)&&(send_recieve(hi_byte)==0))
  { /* printf("Set break point at %X%X%X%X\n",hi_byte/16,hi_byte%16,
                                        lo_byte/16,lo_byte%16);
    */
  } else { printf("Set break point error!\n"); }
}


get_command_size(opcode)
```

```
unsigned char opcode;
{
  switch(opcode)
  {
    case 0x00   : return(1);
    case 0x01   : return(2);
    case 0x02   : return(3);
    case 0x03   : return(1);
    case 0x04   : return(1);
    case 0x05   : return(2);
    case 0x06   : return(1);
    case 0x07   : return(1);
    case 0x08   : return(1);
    case 0x09   : return(1);
    case 0x0A   : return(1);
    case 0x0B   : return(1);
    case 0x0C   : return(1);
    case 0x0D   : return(1);
    case 0x0E   : return(1);
    case 0x0F   : return(1);
    case 0x10   : return(3);
    case 0x11   : return(2);
    case 0x12   : return(3);
    case 0x13   : return(1);
    case 0x14   : return(1);
    case 0x15   : return(2);
    case 0x16   : return(1);
    case 0x17   : return(1);
    case 0x18   : return(1);
    case 0x19   : return(1);
    case 0x1A   : return(1);
    case 0x1B   : return(1);
    case 0x1C   : return(1);
    case 0x1D   : return(1);
    case 0x1E   : return(1);
    case 0x1F   : return(1);
    case 0x20   : return(3);
    case 0x21   : return(2);
    case 0x22   : return(1);
    case 0x23   : return(1);
    case 0x24   : return(2);
    case 0x25   : return(2);
    case 0x26   : return(1);
    case 0x27   : return(1);
    case 0x28   : return(1);
    case 0x29   : return(1);
    case 0x2A   : return(1);
    case 0x2B   : return(1);
    case 0x2C   : return(1);
    case 0x2D   : return(1);
    case 0x2E   : return(1);
    case 0x2F   : return(1);
    case 0x30   : return(3);
    case 0x31   : return(2);
    case 0x32   : return(1);
    case 0x33   : return(1);
    case 0x34   : return(2);
    case 0x35   : return(2);
    case 0x36   : return(1);
    case 0x37   : return(1);
    case 0x38   : return(1);
    case 0x39   : return(1);
    case 0x3A   : return(1);
    case 0x3B   : return(1);
    case 0x3C   : return(1);
    case 0x3D   : return(1);
    case 0x3E   : return(1);
    case 0x3F   : return(1);
    case 0x40   : return(2);
    case 0x41   : return(2);
    case 0x42   : return(2);
    case 0x43   : return(3);
```

```
case 0x44   : return(2);
case 0x45   : return(2);
case 0x46   : return(1);
case 0x47   : return(1);
case 0x48   : return(1);
case 0x49   : return(1);
case 0x4A   : return(1);
case 0x4B   : return(1);
case 0x4C   : return(1);
case 0x4D   : return(1);
case 0x4E   : return(1);
case 0x4F   : return(1);
case 0x50   : return(2);
case 0x51   : return(2);
case 0x52   : return(2);
case 0x53   : return(3);
case 0x54   : return(2);
case 0x55   : return(2);
case 0x56   : return(1);
case 0x57   : return(1);
case 0x58   : return(1);
case 0x59   : return(1);
case 0x5A   : return(1);
case 0x5B   : return(1);
case 0x5C   : return(1);
case 0x5D   : return(1);
case 0x5E   : return(1);
case 0x5F   : return(1);
case 0x60   : return(2);
case 0x61   : return(2);
case 0x62   : return(2);
case 0x63   : return(3);
case 0x64   : return(2);
case 0x65   : return(2);
case 0x66   : return(1);
case 0x67   : return(1);
case 0x68   : return(1);
case 0x69   : return(1);
case 0x6A   : return(1);
case 0x6B   : return(1);
case 0x6C   : return(1);
case 0x6D   : return(1);
case 0x6E   : return(1);
case 0x6F   : return(1);
case 0x70   : return(2);
case 0x71   : return(2);
case 0x72   : return(2);
case 0x73   : return(1);
case 0x74   : return(2);
case 0x75   : return(3);
case 0x76   : return(2);
case 0x77   : return(2);
case 0x78   : return(2);
case 0x79   : return(2);
case 0x7A   : return(2);
case 0x7B   : return(2);
case 0x7C   : return(2);
case 0x7D   : return(2);
case 0x7E   : return(2);
case 0x7F   : return(2);
case 0x80   : return(2);
case 0x81   : return(2);
case 0x82   : return(2);
case 0x83   : return(1);
case 0x84   : return(1);
case 0x85   : return(3);
case 0x86   : return(2);
case 0x87   : return(2);
case 0x88   : return(2);
case 0x89   : return(2);
case 0x8A   : return(2);
case 0x8B   : return(2);
```

```
case 0x8C   : return(2);
case 0x8D   : return(2);
case 0x8E   : return(2);
case 0x8F   : return(2);
case 0x90   : return(3);
case 0x91   : return(2);
case 0x92   : return(2);
case 0x93   : return(1);
case 0x94   : return(2);
case 0x95   : return(2);
case 0x96   : return(1);
case 0x97   : return(1);
case 0x98   : return(1);
case 0x99   : return(1);
case 0x9A   : return(1);
case 0x9B   : return(1);
case 0x9C   : return(1);
case 0x9D   : return(1);
case 0x9E   : return(1);
case 0x9F   : return(1);
case 0xA0   : return(2);
case 0xA1   : return(2);
case 0xA2   : return(2);
case 0xA3   : return(1);
case 0xA4   : return(1);
case 0xA5   : return(0);
case 0xA6   : return(2);
case 0xA7   : return(2);
case 0xA8   : return(2);
case 0xA9   : return(2);
case 0xAA   : return(2);
case 0xAB   : return(2);
case 0xAC   : return(2);
case 0xAD   : return(2);
case 0xAE   : return(2);
case 0xAF   : return(2);
case 0xB0   : return(2);
case 0xB1   : return(2);
case 0xB2   : return(2);
case 0xB3   : return(1);
case 0xB4   : return(3);
case 0xB5   : return(3);
case 0xB6   : return(3);
case 0xB7   : return(3);
case 0xB8   : return(3);
case 0xB9   : return(3);
case 0xBA   : return(3);
case 0xBB   : return(3);
case 0xBC   : return(3);
case 0xBD   : return(3);
case 0xBE   : return(3);
case 0xBF   : return(3);
case 0xC0   : return(2);
case 0xC1   : return(2);
case 0xC2   : return(2);
case 0xC3   : return(1);
case 0xC4   : return(1);
case 0xC5   : return(2);
case 0xC6   : return(1);
case 0xC7   : return(1);
case 0xC8   : return(1);
case 0xC9   : return(1);
case 0xCA   : return(1);
case 0xCB   : return(1);
case 0xCC   : return(1);
case 0xCD   : return(1);
case 0xCE   : return(1);
case 0xCF   : return(1);
case 0xD0   : return(2);
case 0xD1   : return(2);
case 0xD2   : return(2);
case 0xD3   : return(1);
```

```
    case 0xD4   : return(1);
    case 0xD5   : return(3);
    case 0xD6   : return(1);
    case 0xD7   : return(1);
    case 0xD8   : return(2);
    case 0xD9   : return(2);
    case 0xDA   : return(2);
    case 0xDB   : return(2);
    case 0xDC   : return(2);
    case 0xDE   : return(2);
    case 0xDF   : return(2);
    case 0xE0   : return(1);
    case 0xE1   : return(2);
    case 0xE2   ; return(1);
    case 0xE3   : return(1);
    case 0xE4   : return(1);
    case 0xE5   : return(2);
    case 0xE6   : return(1);
    case 0xE7   : return(1);
    case 0xE8   : return(1);
    case 0xE9   : return(1);
    case 0xEA   : return(1);
    case 0xEB   : return(1);
    case 0xEC   : return(1);
    case 0xED   : return(1);
    case 0xEE   : return(1);
    case 0xEF   : return(1);
    case 0xF0   : return(1);
    case 0xF1   : return(2);
    case 0xF2   : return(1);
    case 0xF3   : return(1);
    case 0xF4   : return(1);
    case 0xF5   : return(2);
    case 0xF6   : return(1);
    case 0xF7   : return(1);
    case 0xF8   : return(1);
    case 0xF9   : return(1);
    case 0xFA   : return(1);
    case 0xFB   : return(1);
    case 0xFC   : return(1);
    case 0xFD   : return(1);
    case 0xFE   : return(1);
    case 0xFF   : return(1);
  }
}

single_step()
{
  char command_size;
  unsigned char addr1,addr2,d;

  addr1=read_1byte(0xff,0xf0);
  addr2=read_1byte(0xff,0xf1);
  d=read_1byte(addr1,addr2);
  command_size=get_command_size(d);
  break_point(addr1,addr2+command_size);
  if(send_recieve(0x04)==0)
  { printf("Single step...\n"); }
  else { printf("Single step error!\n\n"); }
}



change_port()
{
   int portnum;
   char c;

   printf("Use communication port : ");
   c=getch();printf("%c",c);
   if(c==49) { portaddr=0x3f8; printf("\n\n"); }
   else {
```

```
        if(c==50) { portaddr=0x2f8; printf("\n\n"); }
       }
}


int read_1byte(hi_byte,lo_byte)
unsigned char hi_byte,lo_byte;
{
   if((send_recieve(0x09)==0)&&
      (send_recieve(hi_byte)==0)&&(send_recieve(lo_byte)==0))
   {
     return(send_recieve2(0x09));
   };
}


int write_1byte(hi_byte,lo_byte,data)
unsigned char hi_byte,lo_byte,data;
{
  if((send_recieve(0x0a)==0)&&
     (send_recieve(hi_byte)==0)&&(send_recieve(lo_byte)==0)&&
     (send_recieve(data)==0))
  { return(0); } else { return(1); }
}


edit_data(s)
char s[80];
{
  unsigned char hi_byte,lo_byte,d2;
  char s2[2];
  int d;

  hi_byte=charhextoint(s[2])*16+charhextoint(s[3]);
  lo_byte=charhextoint(s[4])*16+charhextoint(s[5]);
  d=read_1byte(hi_byte,lo_byte);
  if(d>=0)
  {
     printf("Edit address %X%X%X%X : %X%X ==> ",hi_byte/16,hi_byte%16,
                                        lo_byte/16,lo_byte%16,
                                        d/16,d%16);

     gets(s2);
     d2=charhextoint(s2[0])*16+charhextoint(s2[1]);
     if(write_1byte(hi_byte,lo_byte,d2)!=0)
       { printf("Error writing data!\n"); }
     else { printf("\n"); };
  } else { printf("Error reading data!\n\n"); }
}


jump_toaddress(s)
char s[80];
{
  unsigned char hi_byte,lo_byte,d2;

  hi_byte=charhextoint(s[2])*16+charhextoint(s[3]);
  lo_byte=charhextoint(s[4])*16+charhextoint(s[5]);
  if((send_recieve(0x0c)==0)&&
     (send_recieve(hi_byte)==0)&&(send_recieve(lo_byte)==0))
  { printf("Set current address jump to %X%X%X%X\n\n",hi_byte/16,hi_byte%16,
                                     lo_byte/16,lo_byte%16);
  } else { printf("Jump address error!\n\n"); }
}

help()
{
  printf("    Command List\n");
  printf(" ====================\n");
  printf(" C - Check connection\n");
  printf(" b - Change baud rate\n");
  printf(" L - Load file to ram\n");
  printf(" D - Dump memory\n");
```

```
  printf(" E - Edit memory\n");
  printf(" R - Show registers\n");
  printf(" S - Single step\n");
  printf(" G - Run program\n");
  printf(" J - Jump to address\n");
  printf(" Q - Quit\n");
  printf(" H - Help\n");
  printf(" ======================\n\n");
}


main()
{
  int port;
  int d;
  char command[80];

  clrscr();

  printf("*********************************************\n");
  printf("*          8051 EMULATOR PROJECT            *\n");
  printf("*********************************************\n\n");

  change_port();
/*  portaddr=0x2f8;  */
  code=0x3;
  port_init(0x0c,0x00);

  printf("Press the reset button on emulator board\n");
  printf("And press any key to continue....");
  getch();
  printf("\n\n");
  if(check_connection()==1) { exit(); };
  printf("Press H for help...\n\n");

  do
  {
    printf("PROMPT>");
    gets(command);

    switch(command[0])
    {
      case 'C':check_connection();break;
      case 'b':run_change_baudrate(command);break;
      case 'L':load();break;
      case 'G':go(command);break;
      case 'R':registers();break;
      case 'D':dump(command);break;
      case 'W':write_enable();break;
      case 'M':move();break;
      case 'B':break_point(command);break;
      case 'S':single_step();break;
      case 'P':change_port();break;
      case 'E':edit_data(command);break;
      case 'J':jump_toaddress(command);break;
      case 'H':help();break;
    }
  }while(command[0]!='Q');
  exit(0);
}
```

```
        UART   EQU    4000H   ;RECEIVE BUFF REGIST FOR READ,TRANMITTER
        INTE   EQU    4001H  ;INTERRUPT ENABLE REGISTER
        LCR    EQU    4003H  ;LINE CONTROL REGISTER(DATA FORMAT)
        MODEC  EQU    4004H  ;MODEM CONTROL REGISTER
        LSR    EQU    4005H  ;LINE Status register
        MODES  EQU    4006H  ;Modem Status register

;*************************************************************************

        ORG     0000H

          CLR     RS0             ;set register bank 0
          CLR     RS1
          MOV     A,#90H
          MOV     DPTR,#6003H
          MOVX    @DPTR,A
          MOV     A,#0FFH
          MOV     DPTR,#6002H
          MOVX    @DPTR,A

          MOV     DPTR,#0FFF0H    ;set add for hi_byte
          MOV     A,#80H          ;set for start hi
          MOVX    @DPTR,A
          INC     DPTR
          MOV     A,#00H          ;set add for lo_byte
          MOVX    @DPTR,A         ;set for start low
          INC     DPTR            ;set for break-point hi-byte
          MOVX    @DPTR,A
          INC     DPTR            ;set for break-point lo-byte
          MOVX    @DPTR,A
          INC     DPTR            ;save old address
        MOVX    @DPTR,A

        ACALL   FIRST_INTERNAL_MEM

          MOV     R2,#0CH         ;lsb for set baud rate
          MOV     R3,#00H         ;msb
        ACALL   INIT

PROMPT: ACALL   RECIEVE         ;wait for recieve
        MOV     R0,B            ;save b to r0,,code for command

        CJNE    R0,#01H,NEXT_COMMAND1
        LJMP    CHECK_CONNECTION

NEXT_COMMAND1:
        CJNE    R0,#02H,NEXT_COMMAND2
        LJMP    CHANGE_BAUDRATE

NEXT_COMMAND2:
        CJNE    R0,#03H,NEXT_COMMAND3
        LJMP    LOAD

NEXT_COMMAND3:
        CJNE    R0,#04H,NEXT_COMMAND4
        LJMP    GO

NEXT_COMMAND4:
        CJNE    R0,#05H,NEXT_COMMAND5
        LJMP    REGISTERS

NEXT_COMMAND5:
        CJNE    R0,#06H,NEXT_COMMAND6
        LJMP    DUMP

NEXT_COMMAND6:
        CJNE    R0,#07H,NEXT_COMMAND7
        LJMP    WRITE_ENABLE

NEXT_COMMAND7:
        CJNE    R0,#08H,NEXT_COMMAND8
```

```
        LJMP    MOVE

NEXT_COMMAND8:
        CJNE    R0,#09H,NEXT_COMMAND9
        LJMP    READ_1BYTE

NEXT_COMMAND9:
        CJNE    R0,#0AH,NEXT_COMMAND10
        LJMP    WRITE_1BYTE

NEXT_COMMAND10:
        CJNE    R0,#0BH,NEXT_COMMAND11
        LJMP    BREAK_POINT

NEXT_COMMAND11:
        CJNE    R0,#0CH,NEXT_COMMAND12
        LJMP    JUMP_TOADDRESS

NEXT_COMMAND12:

        LJMP    PROMPT


;*************************************************************************
CHECK_CONNECTION:
        ACALL   SEND
        LJMP    PROMPT

CHANGE_BAUDRATE:
        ACALL   SEND
        ACALL   RECIEVE
        MOV     R2,B
        ACALL   SEND
        ACALL   RECIEVE
        MOV     R3,B
        ACALL   SEND

        ACALL   INIT
        LJMP    PROMPT

LOAD:
        ACALL   SEND
        ACALL   RECIEVE
        MOV     DPH,B           ;add for byte hi
        ACALL   SEND
        ACALL   RECIEVE
        MOV     DPL,B
        ACALL   SEND
        ACALL   RECIEVE
        MOV     R2,B            ;numbers of bytes to load
        ACALL   SEND

LOAD_AGAIN:
        MOV     A,R2
        JZ      END_LOAD
        ACALL   RECIEVE
        MOV     A,B
        ACALL   SEND
        MOVX    @DPTR,A
        INC     DPL
        MOV     A,DPL
        JZ      LOAD_INC_DPH
LOAD_AGAIN2:
        DEC     R2
        JMP     LOAD_AGAIN
LOAD_INC_DPH:
        INC     DPH             ;inc for dpl full
        JMP     LOAD_AGAIN2
END_LOAD:
        LJMP PROMPT
```

```
GO:
        ACALL   SEND

        MOV     DPTR,#BREAK_ROUTINE     ;
        MOV     A,DPL
        PUSH    ACC                     ;push add for break routine
        MOV     A,DPH
        PUSH    ACC
        MOV     DPTR,#0FFF1H
        MOVX    A,@DPTR
        PUSH    ACC                     ;push add for PC
        MOV     DPTR,#0FFF0H
        MOVX    A,@DPTR
        PUSH    ACC

        MOV     DPTR,#0EFD0H
        MOVX    A,@DPTR
        PUSH    ACC
        MOV     DPTR,#0EF83H
        MOVX    A,@DPTR
        PUSH    ACC
        MOV     DPTR,#0EF82H
        MOVX    A,@DPTR
        PUSH    ACC
        MOV     DPTR,#0EFE0H
        MOVX    A,@DPTR
        PUSH    ACC
        MOV     DPTR,#0EF00H
        MOVX    A,@DPTR
        PUSH    ACC
        MOV     DPTR,#0EF01H
        MOVX    A,@DPTR
        PUSH    ACC
        MOV     DPTR,#0EF02H
        MOVX    A,@DPTR
        PUSH    ACC
        MOV     DPTR,#0EF03H
        MOVX    A,@DPTR
        PUSH    ACC
        MOV     DPTR,#0EF04H
        MOVX    A,@DPTR
        PUSH    ACC
        MOV     DPTR,#0EF05H
        MOVX    A,@DPTR
        PUSH    ACC
        MOV     DPTR,#0EF06H
        MOVX    A,@DPTR
        PUSH    ACC
        MOV     DPTR,#0EF07H
        MOVX    A,@DPTR
        PUSH    ACC

        POP     07
        POP     06
        POP     05
        POP     04
        POP     03
        POP     02
        POP     01
        POP     00
        POP     ACC
        POP     DPL
        POP     DPH
        POP     PSW

        RET
        LJMP    PROMPT

REGISTERS:
        ACALL   SEND
```

```
        ACALL   RECIEVE
        MOV     A,B
        MOV     DPH,A
        ACALL   SEND

        ACALL   RECIEVE
        MOV     A,B
        MOV     DPL,A
        ACALL   SEND

        ACALL   RECIEVE
        MOVX    A,@DPTR
        MOV     B,A
        ACALL   SEND
        LJMP    PROMPT


DUMP:
        ACALL   SEND
        ACALL   RECIEVE
        MOV     DPH,B
        ACALL   SEND
        ACALL   RECIEVE
        MOV     DPL,B
        ACALL   SEND

DUMP_AGAIN:
        ACALL   RECIEVE
        MOV     A,B
        JZ      EXIT_DUMP
        MOVX    A,@DPTR
        MOV     B,A
        ACALL   SEND
        INC     DPL
        MOV     A,DPL
        JZ      DUMP_INC_DPH
        JMP     DUMP_AGAIN
DUMP_INC_DPH:
        INC     DPH
        JMP     DUMP_AGAIN
EXIT_DUMP:
        ACALL   SEND
        LJMP    PROMPT


WRITE_ENABLE:
        ACALL   SEND
        MOV     A,#0FFH         ; 11111111
        MOV     DPTR,#6002H
        MOVX    @DPTR,A
        MOV     R2,#20H
        DJNZ    R2,$
        MOV     A,#0F8H         ; 11111000
        MOVX    @DPTR,A
        LJMP    PROMPT

MOVE:
        ACALL   SEND
        MOV     A,#0FFH         ; 11111111
        MOV     DPTR,#6002H
        MOVX    @DPTR,A

        MOV     A,#0FDH         ; 11111101
        MOVX    @DPTR,A

        MOV     R2,#10H
MOVE_AGAIN:
        MOV     DPH,#0F0H
        MOV     DPL,#00H
        ACALL   PULSE
        MOV     A,R3
```

```
        MOVX    @DPTR,A
        INC     DPL
        DJNZ    R2,MOVE_AGAIN

        LJMP    PROMPT


READ_1BYTE:
        ACALL   SEND
        ACALL   RECIEVE
        MOV     DPH,B
        ACALL   SEND
        ACALL   RECIEVE
        MOV     DPL,B
        ACALL   SEND
        MOVX    A,@DPTR
        ACALL   RECIEVE
        MOV     B,A
        ACALL   SEND
        LJMP    PROMPT

WRITE_1BYTE:
        ACALL   SEND
        ACALL   RECIEVE
        MOV     DPH,B
        ACALL   SEND
        ACALL   RECIEVE
        MOV     DPL,B
        ACALL   SEND
        ACALL   RECIEVE
        MOV     A,B
        MOVX    @DPTR,A
        ACALL   SEND

        LJMP    PROMPT

BREAK_POINT:
        ACALL   SEND
        ACALL   RECIEVE
        MOV     A,B
        PUSH    ACC
        MOV     DPL,A
        ACALL   SEND

        ACALL   RECIEVE
        MOV     A,B
        PUSH    ACC
        MOV     DPH,A
        ACALL   SEND

        MOVX    A,@DPTR
        MOV     B,A
        MOV     A,#22H                  ;set for return
        MOVX    @DPTR,A
        MOV     DPTR,#0FFF2H
        POP     ACC
        MOVX    @DPTR,A
        POP     ACC
        INC     DPTR
        MOVX    @DPTR,A
        INC     DPTR                    ;keep old code
        MOV     A,B
        MOVX    @DPTR,A
        LJMP    PROMPT

JUMP_TOADDRESS:
        ACALL   SEND
        MOV     DPTR,#0FFF0H
        ACALL   RECIEVE
        MOV     A,B
        MOVX    @DPTR,A
        ACALL   SEND
```

```
        ACALL   RECIEVE
        INC     DPTR
        MOV     A,B
        MOVX    @DPTR,A
        ACALL   SEND
        LJMP    PROMPT


BREAK_ROUTINE:

        PUSH    PSW
        PUSH    DPH
        PUSH    DPL
        PUSH    ACC
        PUSH    00
        PUSH    01

        MOV     R1,#0FFH
        MOV     DPTR,#0EF00H
        MOV     R0,#00H
BREAK_ROUTINE_AGAIN:
        MOV     A,@R0
        MOVX    @DPTR,A
        INC     DPTR
        INC     R0
        DJNZ    R1,BREAK_ROUTINE_AGAIN

        MOV     DPTR,#0EF01H
        POP     ACC
        MOVX    @DPTR,A
        MOV     DPTR,#0EF00H
        POP     ACC
        MOVX    @DPTR,A
        MOV     DPTR,#0EFE0H
        POP     ACC
        MOVX    @DPTR,A
        MOV     DPTR,#0EF82H
        POP     ACC
        MOVX    @DPTR,A
        MOV     DPTR,#0EF83H
        POP     ACC
        MOVX    @DPTR,A
        MOV     DPTR,#0EFD0H
        POP     ACC
        MOVX    @DPTR,A

        MOV     DPTR,#0FFF2H    ;delete break point
        MOVX    A,@DPTR
        MOV     B,A
        MOV     A,#00H
        MOVX    @DPTR,A
        MOV     A,B
        MOV     DPTR,#0FFF0H
        MOVX    @DPTR,A

        MOV     DPTR,#0FFF3H
        MOVX    A,@DPTR
        MOV     B,A
        MOV     A,#00H
        MOVX    @DPTR,A
        MOV     A,B
        MOV     DPTR,#0FFF1H
        MOVX    @DPTR,A

        MOV     DPTR,#0FFF4H            ;return old code
        MOVX    A,@DPTR
        MOV     B,A
        MOV     A,#00H
        MOVX    @DPTR,A
        MOV     DPTR,#0FFF0H
        MOVX    A,@DPTR
        MOV     R2,A
```

```
        MOV     DPTR,#0FFF1H
        MOVX    A,@DPTR
        MOV     R3,A
        MOV     DPH,R2
        MOV     DPL,R3
        MOV     A,B
        MOVX    @DPTR,A

      LJMP    PROMPT


FIRST_INTERNAL_MEM:

      PUSH    PSW
        PUSH    DPH
        PUSH    DPL
        PUSH    ACC
        PUSH    00
        PUSH    01

        MOV     R1,#0FFH        ;set current register to add ef00
        MOV     DPTR,#0EF00H
        MOV     R0,#00H
FIRST_AGAIN:
        MOV     A,@R0
        MOVX    @DPTR,A
        INC     DPTR
        INC     R0
      DJNZ    R1,FIRST_AGAIN

        MOV     DPTR,#0EF01H
        POP     ACC
        MOVX    @DPTR,A
        MOV     DPTR,#0EF00H
        POP     ACC
        MOVX    @DPTR,A
        MOV     DPTR,#0EFE0H
        POP     ACC
        MOVX    @DPTR,A
        MOV     DPTR,#0EF82H
        POP     ACC
        MOVX    @DPTR,A
        MOV     DPTR,#0EF83H
        POP     ACC
        MOVX    @DPTR,A
        MOV     DPTR,#0EFD0H
        POP     ACC
      MOVX    @DPTR,A
      RET

EXIT_MAIN:   JMP     $


;*****************************************************************************

INIT:
      PUSH    PSW
      PUSH    ACC
      PUSH    DPH
      PUSH    DPL
        PUSH    02
        PUSH    03

        MOV     DPTR,#4003H     ;LCR register
      MOV     A,#10000011B    ;set DLAB bit
      MOVX    @DPTR,A

      MOV     DPTR,#4000H     ;set baud bit 9600
        MOV     A,R2
        MOVX    @DPTR,A

        MOV     DPTR,#4001H
```

```
        MOV     A,R3
       MOVX   @DPTR,A

        POP     03
        POP     02
       POP     DPL
       POP     DPH
       POP     ACC
       POP     PSW
       RET

;*****************************************************************************
;RECIEVED CHAR B

RECIEVE: PUSH PSW
        PUSH ACC
         PUSH DPH
         PUSH DPL

WAIT_R:  MOV DPTR,#4005H
        MOVX A,@DPTR
         ANL A,#1
        JZ WAIT_R

         MOV     DPTR,#4003H     ;LCR reg
        MOV     A,#00000011B    ;set 8-bit data, no parity check
        MOVX    @DPTR,A

        MOV DPTR,#4000H
        MOVX A,@DPTR
         MOV B,A

EXIT_RECIEVE:
         POP DPL
         POP DPH
        POP ACC
        POP PSW
        RET


;*****************************************************************************
;B => CHAR TO SEND

SEND:
        PUSH    PSW
        PUSH    ACC
        PUSH    B
        PUSH    02
        PUSH    03
        PUSH    04
       PUSH    DPH
       PUSH    DPL

WAIT_S:
        MOV     DPTR,#4003H
        MOV     A,#00000011B
       MOVX    @DPTR,A

       MOV     DPTR,#4005H
       MOVX    A,@DPTR
       ANL     A,#00100000B
       JZ      WAIT_S

        MOV     R4,#02H

DELAY_S:  MOV     R3,#0FFH
DELAY1_S: MOV     R2,#0ffH
        DJNZ    R2,$
        DJNZ    R3,DELAY1_S
        DJNZ    R4,DELAY_S

        MOV     DPTR,#4000H     ;THR register
```

```
        MOV     A,B
        MOVX    @DPTR,A

      POP     DPL
      POP     DPH
        POP     04
        POP     03
      POP     02
        POP     B
      POP     ACC
      POP     PSW
      RET


PULSE:  PUSH    PSW
        PUSH    ACC
        PUSH    DPH
        PUSH    DPL
        PUSH    00
        MOV     R0,#20H
        MOV     DPTR,#6002H
        MOV     A,#0FCH             ; 11111100
        MOVX    A,@DPTR
PULSE1: DJNZ    R0,PULSE1
        MOV     DPTR,#6000H
        MOVX    A,@DPTR
        MOV     R3,A
        MOV     A,#0FDH             ; 11111101
        INC     DPTR
        INC     DPTR
        MOVX    @DPTR,A
        MOV     R0,#20H
PULSE2: DJNZ    R0,PULSE2
        POP     00
        POP     DPL
        POP     DPH
        POP     ACC
        POP     PSW
        RET

;*******************************************************************************
      END
```

กิตติกรรมประกาศ

ขอขอบคุณ ผศ. เกรียงไกร วงศ์โรจน์ภรณ์ ที่ได้สละเวลาให้คำปรึกษาในการทำให้ปริญญานิพนธ์ชิ้นนี้เสร็จสมบูรณ์ได้ด้วยดี นอกจากนี้ ขอแสดงความขอบคุณไปถึง รศ.ดร. สุวิพล สิทธิชีวภาค ที่ให้ความอุปการะสถานที่ในการทำงานและอุปกรณ์ที่ใช้ในการทดลองเป็นอย่างดี รวมทั้งเพื่อน ๆ ทุกคนที่คอยให้กำลังใจ ขอบคุณโคที่มีส่วนให้คำปรึกษาอย่างมาก, ขอบคุณเก๋สำหรับเสบียงของพวกเรา และเพื่อนๆ ร่วมห้องโปรเจค ทุกคนที่ได้ทำงานร่วมกันมาและคอยเป็นกำลังใจในการทำงานอย่างมาก

จึงขอขอบคุณมา ณ โอกาสนี้ด้วย

หนังสืออ้างอิง

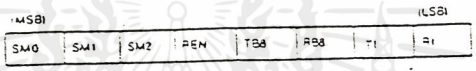
1. คัมภีร์การใช้งานการสื่อสารอนุกรมบน PC , Mastering Serial Communications , Peter W. Gofton , บริษัท ซีเอ็ดยูเคชั่น จำกัด
2. การเขียนโปรแกรมภาษา ซี สำหรับวิศวกรรม , ธันวา ศรีประมง, มหาวิทยาลัยเทคโนโลยีมหานคร
3. การเขียนโปรแกรมแบบโอโอพีด้วยเทอร์โบและบอร์แลนด์ C++ , Robert Lafore , บริษัท ซีเอ็ดยูเคชั่น จำกัด
4. การเขียนโปรแกรมคอมพิวเตอร์ด้วยเทอร์โบซี , มนตรี พจนารภลาวัลย์ , บริษัท ซีเอ็ดยูเคชั่น จำกัด
5. การประยุกต์ใช้งานไมโครคอนโทรลเลอร์ ตระกูล MCS - 51 , ผศ. สมยศ จุณณะปิยะ , สถาบันเทคโนโลยีพระจอมเกล้าลาดกระบัง
6. Microprocessors principle and applications , Charles M. Gilmore , McGraw-Hill international editions.
7. Intel Microcommunication Handbook 1989.

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

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)



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

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	

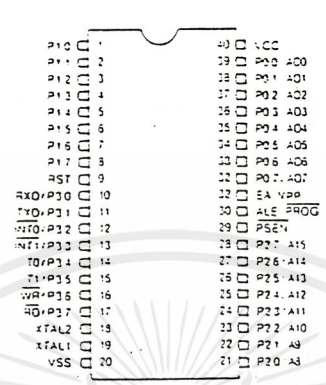
MNEMONIC	ASSEMBLY LANGUAGE FORM	HEX OP-CODE	BYTES	CYCLES	PSW FLAGS AFFECTED
MOV					
Move Source Operand to Destination Operand	MOV A,#data	74	2	1	
	MOV A,Daddr	E5	2	1	
	MOV A,@Ri	E6,E7	1	1	
	MOV A,Rn	E8-EF	1	1	
	MOV Daddr,A	F5	2	1	
	MOV Daddr,#data	75	3	2	
	MOV Daddr,Daddr	85	3	2	
	MOV Daddr,@Ri	86,87	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 Memory to Acc	MOVC A,@A+DPTR	93	1	2	
	MOVC A,@A+PC	83	1	2	
MOVX					
Move byte from Ext Data Memory to Acc	MOVX A,@Ri	E2,E3	1	2	
	MOVX A,@DPTR	E0	1	2	
Move byte in Acc to Ext Data Memory	MOVX @Ri,A	F2,F3	1	2	
	MOVX @DPTR,A	F0	1	2	
MUL					
Multiply Acc by B Reg	MUL AB	A4	1	4	Cy, OV see note 7
NOP					
No Operation	NOP	00	1	1	
ORL					
Logical Inclusive OR of Source Operand with Destination Operand	ORL Daddr,A	42	2	1	
	ORL Daddr,#data	43	3	2	
	ORL A,#data	44	2	1	
	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
	SUBB A,Daddr
	SUBB A,@Ri
	SUBB A,Rn
		94
		95
		96,97
		98-9F
			2
			2
			1
			1
				1
				1
				1
				1
					Ac.
					Cy
					OV

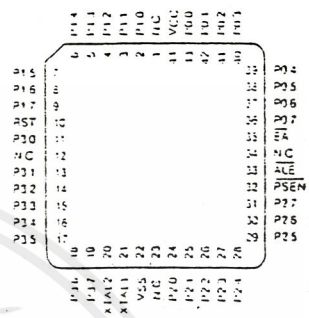
SWAP Swap Nibbles within Acc	SWAP A	C4	1	1	
XCH Exchange bytes of Acc and Source Operand	XCH A,Daddr XCH A,@Ri XCH A,Rn	C5 C6,C7 C8-CF	2 1 1	1 1 1	
XCHD Exchange the Least Sig Nibble of Acc and Source Operand	XCHD A,@Ri	D6-D7	1	1	
XRL Logical Exclusive OR of Source Operand with Destination Operand	XRL Daddr,A XRL Daddr,#data XRL A,#data XRL A,Daddr XRL A,@Ri XRL A,Rn	62 63 64 65 66,67 68-6F	2 3 2 2 1 1	1 2 1 1 1 1	

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

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	ADDC	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