ACCESS KEY

เครื่องอ่านบัตรแถบแม่เหล็ก

โดย

1.นายธานินทร์ ปั้นทอง 36013196
2.นายประวิทย์ เตชะวิริยะวงศ์ 36013200
3.นายไพศาล เกาะโพธิ์ 36013205
4.นายวีระพัฒน์ พงษ์โสภา 36013213
5.นายเสฎฐวุฒิ สุขุมาลวงศ์ 36013226
6.นายอุดม ปานแย้ม 36013229

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตร
ปริญญาอุตสาหกรรมศาสตร์บัณฑิต ภาควิชา เทคนิคอุตสาหกรรม
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2538

ใบรับรองปริญญานิพนธ์

ภาควิชาเทคนิคอุตสาหกรรม คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง

เครื่องอ่านบัตรแถบแม่เหล็ก

ACCESS KEY CONTROL

โดย

นายธานินทร์ ปั้นทอง	36013196
นายประทีป เตชะวิริยะวงศ์	36013200
นายไพศาล เกาะไพธิ์	36013205
นายวีระพัฒน์ พงษ์โสภา	36013213
นายเสฏฐวุฒิ สุขุมาลางศ์	36013226
นายอุดม ปานเย็น	36013229

อาจารย์ที่ปรึกษา อ. อุทัย ศรีธีระวิโรจน์

ภาควิชา เทคนิคอุตสาหกรรม

ปีการศึกษา 2538

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตร์บัณฑิต

คณะกรรมการสอบปริญญานิพนธ์

..... ประธานกรรมการ

( )

..... กรรมการ

( )

..... กรรมการ

( )

..... กรรมการ

( )

คำนำ

ดังที่ได้ทราบกันแล้วว่า *ไมโครโปรเซสเซอร์* มีบทบาทมากในยุคปัจจุบันและได้นำมาประยุกต์ใช้งานกันอย่างแพร่หลายซึ่งบุคคลทั่วไปที่มีความรู้ทางด้านนี้ส่วนใหญ่ ได้หันมาให้ความสนใจและความสำคัญโดยเล็งเห็นประโยชน์ต่อการนำไปใช้งานต่างๆเป็นอย่างมาก จึงได้มีการพัฒนาและนำเอาอุปกรณ์พวกนี้มาใช้ประโยชน์ต่างๆ เช่น *ไมโครคอมพิวเตอร์ ไมโครคอนโทรเลอร์* และอื่นๆอีกมากมาย

จากประโยชน์ของอุปกรณ์ดังกล่าว ทางคณะผู้จัดทำปริญญานิพนธ์ชุดนี้ได้เล็งเห็นประโยชน์ จึงได้ออกแบบระบบ ACCESS KEY ซึ่งเป็นระบบควบคุมการเปิดปิด ด้วยเครื่องอ่านแถบแม่เหล็ก ตามโปรแกรมฟังก์ชั่นที่เรากำหนด สามารถนำไปใช้ เพื่อควบคุมการเปิดปิด ของระบบภายนอกต่างๆ เช่น ปิด-เปิด ประตูด้วยระบบไฟฟ้า บันทึกเวลาการปฏิบัติงาน และอื่นๆ เพื่อเป็นประโยชน์ต่อการศึกษาและค้นคว้าต่อการทำงานในอนาคตต่อไป

อนึ่ง ในปริญญานิพนธ์เล่มนี้ ได้รวบรวมเนื้อหาเพียงบางส่วนที่ได้ค้นคว้ามาตลอด 4 เดือน โดยที่ตัวโครงงานยังไม่แล้วเสร็จสมบูรณ์ ซึ่งประกอบไปด้วย วงจร การทำงาน และ คำอธิบายของตัวอุปกรณ์ทางด้านอิเล็กทรอนิกส์ที่เกี่ยวข้องต่างๆ หากพบข้อผิดพลาดประการใด ทางคณะผู้จัดทำขอรับไว้เพื่อปรับปรุงให้ดีขึ้นต่อไป

คณะผู้จัดทำ

สารบัญ

ACCESS KEY

เครื่องอ่านบัตรแถบแม่เหล็ก

รายละเอียดทั่วไป

เครื่องอ่านบัตรแถบแม่เหล็ก สามารถอ่านแถบแม่เหล็กที่บันทึกตามมาตรฐาน ISO7811 โดยจะอ่านเฉพาะข้อมูลในแถบที่ 2 (ABA TRACK) ข้อมูลที่อ่านจะเก็บไว้ในหน่วยความจำของเครื่องพร้อมทั้ง วันและเวลาในขณะนั้น ข้อมูลจะไม่สูญหายแม้ว่าไฟฟ้าจะดับเพราะมีแบตเตอรี่จ่ายไฟให้หน่วยความจำอยู่ภายใน

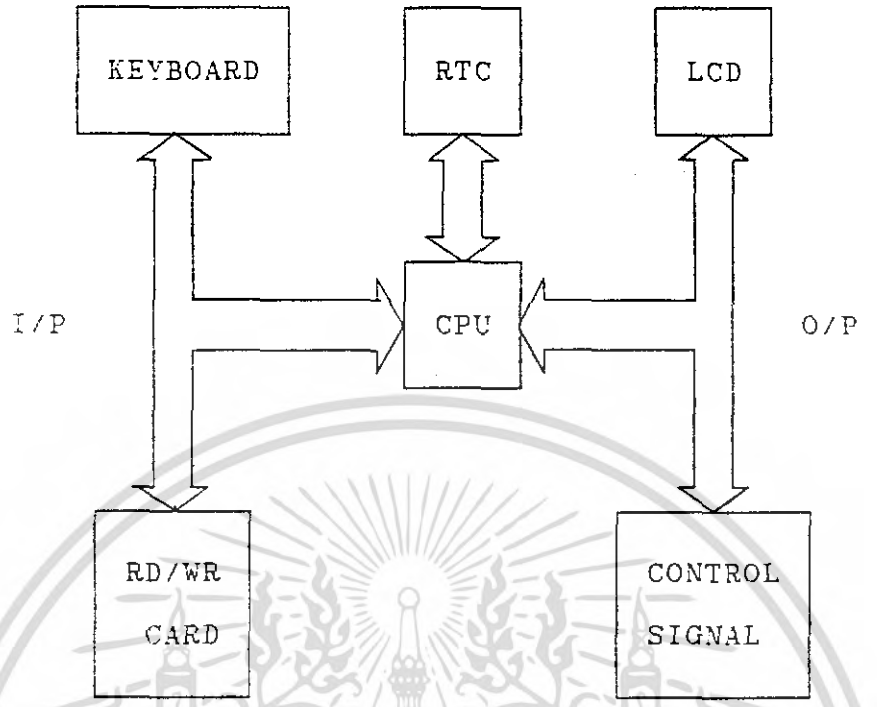
เครื่องอ่านบัตรยังใช้งานได้อีกหลายลักษณะ เช่น ใช้ร่วมกับคีย์บอร์ดรับข้อมูลเพิ่มเติม หรือใช้บัตรปิด-เปิดประตู ซึ่งบัตรเป็นเสมือนกุญแจ การเปลี่ยนลักษณะบางอย่างสามารถทำได้ทันทีโดยใช้คำสั่งที่มีอยู่แล้วในตัวเครื่อง

ลักษณะเด่นของเครื่องอ่านบัตรแถบแม่เหล็ก

- บันทึกวัน เดือน ปี และเวลา ที่อ่านบัตรได้ด้วยตัวเอง ภายในเครื่องมีนาฬิกาและปฏิทินทำให้บันทึกวันและเวลาได้ด้วยตัวเองอย่างเที่ยงตรง การตั้งเวลาสามารถทำได้ที่ตัวเครื่องโดยตรงได้เลย
- มีหน่วยความจำขนาดใหญ่ สามารถเก็บข้อมูลได้มาก บันทึกข้อมูลในบัตรทั้งหมดพร้อมทั้งวันและเวลาที่อ่านบัตร รวมทั้งข้อมูลอื่นๆ เช่น เข้าหรือออก โดยจะเก็บไว้ตลอดเวลาแม้ไฟฟ้าจะดับ ถ้าต้องการบันทึกข้อมูลอื่นอีก เช่น หมายเลขรหัส หรือ จำนวน ก็สามารถทำได้ ความจุของเครื่องขึ้นอยู่กับขนาดของข้อมูล ซึ่งสามารถปรับได้ตามความต้องการ

วัตถุประสงค์ของการสร้างเครื่องอ่านบัตรแถบแม่เหล็ก

- นำไปควบคุมความปลอดภัยของห้องที่มีความลับมาก ๆ
- บันทึกวัน เดือน ปี ที่มีการเข้าออกหรือทุกครั้งที่มีการใช้บัตรแถบแม่เหล็กเพื่อเป็นประโยชน์ในการเก็บข้อมูล



BLOCK DIAGRAM

การทำงาน จาก BLOCK DIAGRAM

ภาค INPUT เครื่องอ่านบัตรแถบแม่เหล็กจะแบ่ง INPUT ออกเป็นส่วนสำคัญ ๆ คือ ส่วนของการอ่าน-เขียน บัตรแถบแม่เหล็กและส่วนของการป้อนข้อมูลหรือคีย์บอร์ด

ส่วนของการอ่าน-เขียนบัตรแถบแม่เหล็ก ในส่วนของการอ่านเขียนบัตรแถบแม่เหล็กจะทำงานคล้าย ๆ กับการบันทึกเสียงทั่วไป แต่จะบันทึกข้อมูลไว้ที่ TRAC 2 เท่านั้น ซึ่งจะเหมือนกับบัตรแถบแม่เหล็กของตู้ ATM ของธนาคารทั่ว ๆ ไป ซึ่งปัจจุบันอุปกรณ์ที่ใช้ในการบันทึกเสียงเป็นแม่เหล็กแบบสารแม่เหล็กหลายชนิดเช่น เทปเสียง แผ่นดิสก์ อุปกรณ์เหล่านี้ประกอบด้วยวัสดุหลายชิ้นมาซ้อนกันตามชนิดหรือรูปแบบการใช้งาน บัตรแถบแม่เหล็กที่ใช้จะต้องถูกบันทึก ข้อมูลหรือ ENCODE ข้อมูลต่าง ๆ ที่ต้องการลงไปก่อน และเมื่อนำมาอ่านกับเครื่องอ่านแถบแม่เหล็กข้อมูลที่ได้จะเป็นแบบ SEPRIER DATA และจะมี CLOCK รวมอยู่ด้วยสาเหตุที่ต้องมี CLOCK ก็เพราะว่าในการรูดบัตรนั้น ความเร็วของการรูดบัตรไม่เป็น LINERA แล้วส่ง OUT PUT ที่ได้ไปยัง CPU เพื่อทำการประมวลผลต่อไป

ส่วนของการป้อนข้อมูลเพิ่มเติมหรือคีย์บอร์ดในส่วนนี้จะมีหน้าที่หลัก ๆ คือ จะเป็นตัวป้อนรหัสเพิ่มเติมหลังจากการอ่านข้อมูลจากแถบแม่เหล็กแล้ว และยังทำหน้าที่ในการตั้งวัน เดือน ปีเวลา และข้อมูลอื่น ๆ อีก ลงไปในหน่วยความจำ ในส่วนของคีย์บอร์ดนี้จะใช้ IC เบอร์ MM74C923 ซึ่งการเข้ารหัสของการกดคีย์บอร์ดจะเป็นตัวเลขฐานสอง 5 บิต อย่างอัตโนมัติและเมื่อคีย์ใดถูกกดจะมีสัญญาณจากขา DATA AVAILABLE บอกสภาวะของการ

กดคีย์ให้กับไมโครคอนโทรเลอร์ได้รับรู้ ในการกดคีย์ทั่ว ๆ ไป จะเกิดการดีเบาซ์ขึ้น การแก้ดีเบ้าซ์สามารถทำได้ทั้งทางฮาร์ดแวร์ และซอฟแวร์ การแก้ดีเบ้าซ์ทางฮาร์ดแวร์สามารถทำได้โดยต่อตัวเก็บประจุค่า 1 ไมโครฟารัดที่ขา KBM (KEYBOARD BOUNCE MASKK) ทำให้ตัว IC 74C923 ไม่สนใจ การกดคีย์ในช่วง 10 มิลลิวินาทีแรก แต่หลังจากนั้นจะแสตซ์ข้อมูลช่วงเวลาของการดีย์เบาซ์ สามารถปรับได้ตามค่าของตัวเก็บประจุ

หน่วยประมวลผลกลางหรือ CPU ในหน่วยนี้จะใช้ IC เบอร์ 8031 เป็นไอซีคอนโทรเลอร์และใช้ MEMORY จากภายนอกมาต่อใช้งานในภาคนี้เป็นส่วนที่สำคัญมากคือจะรับข้อมูลจากหัวอ่านแถบแม่เหล็ก มาทำการประมวลผลร่วมกับโปรแกรมที่อยู่ภายในหน่วยความจำที่เขียนไว้ โปรแกรมที่เขียนขึ้นนั้นจะเป็นหัวใจสำคัญของหน่วย CPU นี้เพราะถ้าไม่มีโปรแกรมก็ไม่สามารถที่จะแสดงผลอะไรออกมาได้เลย เนื่องจาก CPU ที่ใช้เป็น คอนโทรเลอร์และยังต้องมีการติดต่อกับหน่วยความจำภาค INPUT ภาค OUTPUT ซึ่งเป็น LCD ในแต่ละภาคการทำงานมักจะไม่พร้อมกันจึงต้องนำฐานเวลามาใช้ร่วมกับ CPU เพื่อเป็นฐานเวลามาตรฐาน เพื่อป้องกันการผิดพลาดของการรับ-ส่งข้อมูล ซึ่ง CPU นี้มีความเร็วของการรับส่งข้อมูลสูงถึง 12 MHZ

ภาค OUTPUT ในภาคนี้จะมีส่วนประกอบหลัก ๆ 2 ส่วน คือ ภาคแสดงผล และภาคของสัญญาณควบคุม

ภาคของการแสดงผล จะใช้อุปกรณ์แบบ LCD แถวเดียวซึ่งเป็นชุดสำเร็จรูป จะทำหน้าที่หลัก ๆ คือ แสดงสถานะต่างๆ ของการทำงาน เช่น ในสภาวะการอ่านข้อมูลจากบัตรแถบแม่เหล็ก ภาคแสดงผลก็จะบอกสถานะของบัตรนั้น ๆ ตามโปรแกรมที่ได้เขียนไว้ในหน่วยความจำ และหลังจากนั้น ก็จะแสดงขั้นตอนของการทำงานหรือสภาวะต่าง ๆ ของการทำงานต่อไป

ภาคของสัญญาณควบคุม ในส่วนนี้จะทำหน้าที่ส่งสัญญาณต่าง ๆ ตามที่ได้โปรแกรมเอาไว้ไปควบคุมอุปกรณ์ภายนอก ซึ่งอาจจะเป็นประตูไฟฟ้าหรือสัญญาณเตือนภัยต่าง ๆ แล้วแต่ผู้ใช้จะกำหนดซึ่งง่ายต่อการออกแบบ

Central ProcessingUnit

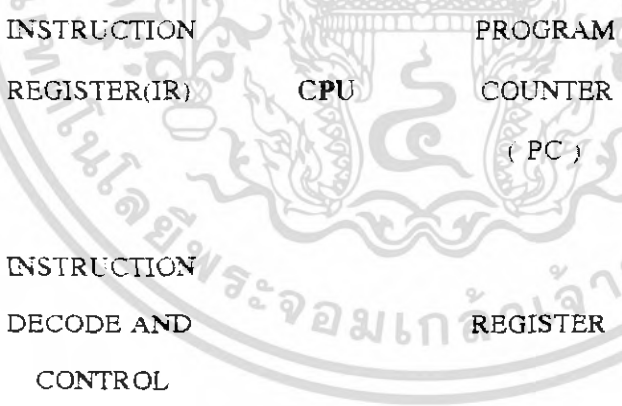
หน่วยประมวลผลกลาง

หน่วยประมวลผลกลาง เปรียบเสมือนส่วนสมองของระบบคอมพิวเตอร์ การกระทำต่างๆในระบบและการประมวลข้อมูลจะถูกควบคุมโดย CPU ทั้งสิ้น CPU จะทำหน้าที่สั่งให้เกิดการกระทำต่างๆ ในระบบได้สอดคล้องซึ่งกันและกัน การกระทำภายในตัว CPU จะมีขั้นตอนที่สลับซับซ้อน ทั้งนี้เนื่องจากภายในตัว CPU ประกอบด้วยวงจรตรรกมากมาย โดยแต่ละส่วนจะทำงานต่อเนื่องกันไปตลอดเวลา พอจะสรุปการทำงานได้เป็นสองขั้นตอนดังนี้

ขั้นตอนแรก คือ การแฟทช์คำสั่ง (fetch instruction)

ขั้นตอนที่สอง คือ การกระทำตามคำสั่ง(execute instruction)

องค์ประกอบภายในตัว CPU ได้ถูกออกแบบไว้ให้สามารถ เข้าใจและปฏิบัติตามคำสั่งที่อยู่ในรูปแบบของรหัสเลขฐานสอง (BINARY CODE) กลุ่มของรหัสเลขฐานสองเหล่านี้เรียกว่า ชุดคำสั่ง (Instruction Set)



BLOCK DIAGRAM ของ CPU

จากรูป เป็นการเขียน BLOCK DIAGRAM อย่างง่ายของหน่วยประมวลผลกลาง เริ่มจากกลุ่มของรีจิสเตอร์ (REGISTER) ใช้สำหรับเก็บหรือพักข้อมูลชั่วคราว หน่วยคำนวณและหน่วยตรรก (ARITHMETIC AND LOGIC UNIT) ทำหน้าที่เกี่ยวกับการกระทำทางด้านการคำนวณและตรรก ส่วนถอดรหัสคำสั่งและควบคุม (INSTRUCTION DECODE AND CONTROL UNIT) ทำหน้าที่ แปลความหมายของคำสั่งว่าจะทำอะไร แล้วจึงสร้างสัญญานขึ้นชุดหนึ่งที่สอดคล้องกับคำสั่งนั้นๆ ส่งไปยังส่วนต่างๆ เพื่อให้ได้ผลตามที่ต้องการ รีจิสเตอร์คำสั่ง(I NSTRUCTION REGISTER,IR) ทำหน้าที่เก็บรหัสคำสั่งซึ่งอยู่ในรูปของเลขฐานสองไว้ชั่วคราว รีจิสเตอร์โปรแกรมเคาท์เตอร์ (PROGRAM COUNTER,PC) ใช้สำหรับเก็บตำแหน่งแอดเดรสของหน่วยความจำ ที่มีคำสั่ง จะต้องกระทำถัดไปเก็บบันทึกอยู่

ในเครื่องอ่านบัตรแถบแม่เหล็กจะใช้ CPU เบอร์ 8031 ซึ่งเป็นไมโครคอนโทรเลอร์ที่มีหน่วยความจำภายในมีความเร็วในการทำงานสูงกว่า CPU เบอร์ Z-80 สาเหตุที่ไม่ใช้ CPU เบอร์ Z-80 เพราะว่า CPU เบอร์ Z-80 เหมาะสำหรับงานประมวลผลทางระบบคอมพิวเตอร์มากกว่า และใช้อุปกรณ์ในการควบคุมมากมาย แต่สำหรับไมโครคอนโทรเลอร์นั้นจะใช้อุปกรณ์น้อยที่สุด และการประมวลผลจะทำงานไม่ซับซ้อนมากนัก และการทำงานซ้ำๆกัน และภายในไอซีก็อนโทรเลอร์จะมีชุดคำสั่งเหมาะสมสำหรับควบคุมอินพุทและเอาท์พุท การเชื่อมต่อกับอินพุทและเอาท์พุทโดยใช้เพียงบิทเดียว ตัวอย่างเช่น มอเตอร์ถูกควบคุมให้หมุนหรือหยุดหมุนโดยใช้โซลินอยน์ สามารถใช้เอาท์พุทพอร์ทเพียงบิทเดียวมาควบคุมไมโครคอนโทรเลอร์ มีคำสั่งเซ็ทและเคลียร์ในระดับบิทที่แยกออกมาเป็นส่วนๆต่างหาก และสามารถจัดเก็บข้อมูลในระดับบิทได้ เช่นการแอนด์ออร์ เอกซ์คลูซีฟออร์ บิทต่อบิท มีคำสั่งกระโดดเมื่อบิทเซ็ทหรือเคลียร์ และอื่นๆอีกมาก ส่วนเหล่านี้จะไม่มีในไมโครโปรเซสเซอร์ ซึ่งมักกระทำข้อมูลในระดับไบท์หรือมากกว่า

ข้อแตกต่าง ของไมโครโปรเซสเซอร์ Z-80 และไมโครคอนโทรเลอร์ 8031

การกำหนดขา	Z-80	8031
จำนวนขาทั้งหมด	70	40
ขาของบัสแอสเดรส	16(คงที่)	16
ขาของบัสข้อมูล	8(คงที่)	8
ขาอินเทอรัพท์	2(คงที่)	2
ขาอินพุท/เอาท์พุท	0	32

โครงสร้าง

รีจิสเตอร์ขนาด 8 บิท	20	34
รีจิสเตอร์ขนาด 16 บิท	4	2
ขนาดของสแต็ก	64k	128
รอมภายใน	0	0
แรมภายใน	0	0
หน่วยความจำภายนอก	64k	128k
แฟลก	0	4
ไทเมอร์	0	4
พอร์ทขนาน	0	4
พอร์ทอนุกรม	0	1

ชุดคำสั่ง (ชนิด / ตัวแปร)

เคลื่อนย้ายข้อมูลภายนอก	4/14	2/6
เคลื่อนย้ายข้อมูลเป็นบล็อก	2/4	0
จัดการในระดับบิท	4/4	12/12
กระโดดอันเนื่องมาจากข้อมูลระดับบิท	0	3/3
สแตกก	3/15	2/2
หนึ่งไบท์	203	49
หลายไบท์	490	62

-ขา Vss (ขา20) เป็นขาสำหรับต่อลงกราว์ด

-ขา Vcc (ขา40) เป็นขาที่ต่อแรงดันไฟกระแสตรงขนาด 5v และใช้สำหรับการโปรแกรม

-ขา PORT0 (P0.0-P0.7/AD0-AD7) (ขา 32-39) เป็นไอโอ 8 บิท แบบ OPEN DRAIN BIDIRECTIONAL สามารถที่จะรับโหลดทีทีแอลได้ 8 ตัว

-ขา PORT1 (P1.0-P1.7) (ขา 1-8)เป็นพอร์ทไอโอ 8 บิท แบบ OPEN DRAIN BIDIRECTIONAL ด้วยการพูลอัพภายใน ถ้าเป็นพอร์ทเอาท์พุท บัฟเฟอร์สามารถรับ โหลด TTLS ได้ 4 ตัว

-ขา PORT2 (P2.0-2.7) (ขา 21-28) เป็นพอร์ทไอโอ 8 บิท แบบ OPEN DRAIN BIDIRECTIONAL ด้วยการพูลอัฟภายใน พอร์ท2 ทำหน้าที่เป็นบัพเฟอร์ เอาท์พุทสามารถจ่ายโหลดทีทีแอลเอสได้ 4 ตัว พอร์ทจะถูกใช้งานเป็นตัวส่งแอดเดรสไบท์สูงด้วย

-ขา PORT3 (P3.0-P3.7) (ขา 10-17) เป็นพอร์ทไอโอ 8 บิท แบบพูลอัฟภายใน นอกจากทำเป็นพอร์ทไอโอที่สามารถรับโหลด TTL ได้ 4 ตัว และยัง ใช้งานพิเศษตามรายการข้างล่างนี้ด้วย

ขาพอร์ท	ขา	การทำงานตามฟังก์ชั่นพิเศษ
P3.0	10	RXD พอร์ทอนุกรมอินพุท
P3.1	11	TXD พอร์ทอนุกรมเอาท์พุท
P3.2	12	INT0 อินเตอร์รัพภายนอกตัวที่ 1
P3.3	13	INT1 อินเตอร์รัพภายนอกตัวที่ 2
P3.4	14	T0 สัญญาณกระตุ้นเข้าที่ตัวตั้งเวลาและตัวนับ
P3.5	15	T1 สัญญาณกระตุ้นเข้าที่ตัวตั้งเวลาและตัวนับ
P3.6	16	WR สัญญารควบคุมการเขียน
P3.7	17	RD สัญญาณควบคุมการอ่าน

การทำงานตามฟังก์ชั่นต้องเริ่มโปรแกรมด้วยการส่งค่า "1" ไปแลทช์ก่อนที่จะทำงานต ฟังก์ชั่น

-ขา RST (ขา 9) ต้องคงสถานะค่าสูงเป็นเวลาประมาณอย่างน้อยประมาณ 2 cycleระหว่างที่ออสซิเลเตอร์ทำงาน ขณะที่ต้องการรีเซ็ทท้งระบบงาน โดยจะต่อ รีจีสเตอร์พูลดาวน์ (8.2 กิโลโอห์ม) จากขา RST ไปลงกราวด์ และเพื่อให้ตัวชิพรีเซ็ทได้โดยอัตโนมัติขณะเปิดไฟจะใช้ค่าคาปาซิเตอร์ 10 ไมโครฟารัด ต่อคร่อมระหว่างขา RST กับ ขา VCC

-ขา ALE/PROG (ขา 30) เป็นขาแอดเดรสแลทช์ อีเนเบิลด้วยการส่งพัลส์ออกไปใช้สำหรับแลทช์ค่าแอดเดรสไบท์ต่ำจากพอร์ท 0 ในระหว่างการเข้าถึงข้อมูลจากหน่วยความจำ ALE จะถูกส่งสัญญาณนาฬิกาออกมาในอัตราความเร็วคงที่ที่ 1/8 ของความถี่ออสซิเลเตอร์ตลอดเวลาความถี่ของสัญญาณนี้ จะลดลงเท่าหนึ่ง จากการทำงานแบบเข้าถึงของหน่วยความจำภายนอก

-ขา PSEN (ขา 29) PROGRAM STORAGE ENABLE เป็น สโตรบ อ่านข้อมูลจากโปรแกรมในหน่วยความจำภายนอก โดยจะสร้างสโตรบต่ำ 2 ครั้ง ภายในแต่ละ cycle machine สัญญาณจะมีสถานะสูง หรือ พัลส์ต่ำทั้งสองลูกจะหายไปเมื่อทำงานในช่วงการอ่านหรือเขียนข้อมูลจากหน่วยความจำภายนอก

-ขา EA/VPP (ขา 31) มีสถานะต่ำเป็นการควบคุมให้ CPU ทำงานตามโปรแกรมหน่วยความจำภายนอก ซึ่งสามารถขยายโปรแกรมได้ยาวถึง 64 กิโลไบท์

-ขา XTAL1 (ขา 19) ใช้เป็นตัวอินพุท เข้าสู่ตัวออสซิเลเตอร์ขยายแบบ INVERT

-ขา XTAL2 (ขา 18) ใช้เป็นตัวเอาท์พุทจากตัวออสซิเลเตอร์ขยายแบบINVERT

ชุดคำสั่ง

ชุดคำสั่งจะมีทั้งหมด 111 คำสั่ง ประกอบด้วยคำสั่งที่มี 1 ไบต์อยู่ 49 คำสั่ง 2ไบต์ 45 คำสั่ง และอีก 17 คำสั่งที่เหลือมีขนาดยาว 3 ไบต์ รูปแบบคำสั่งออปโค้ดจะประกอบด้วยคำสั่งของนิวมอนิค ที่ตามด้วยตัวโอเปอร์แรนด์ที่เป็นตัวรับการถ่ายทอด, แหล่งกำเนิดในฟิลด์โอเปอร์แรนด์นี้จะเป็นแบบข้อมูลคงที่หรือตามแบบการใช้โหมดการกำหนดตำแหน่งเลขที่อยู่ ตามการออกแบบของอินเทล การกำหนดแอดเดรสหลายไบต์และตัวโอเปอร์แรนด์ข้อมูลจะเกบไบต์ที่มีความสำคัญต่ำไว้ที่แอดเดรสต่ำแหน่งสูง และไบต์ที่มีความสำคัญสูงไว้ที่แอดเดรสตำแหน่งต่ำ

ลักษณะการทำงานตามฟังก์ชั่น

ชุดคำสั่งจะถูกแบ่งเป็นลักษณะการทำงานตามฟังก์ชั่นได้ 4 กลุ่มคือ

กลุ่มการถ่ายเทข้อมูล

กลุ่มคณิตศาสตร์

กลุ่มตรรกศาสตร์

กลุ่มการควบคุมการถ่ายเท

 # กลุ่มการถ่ายเทข้อมูล

การถ่ายเทข้อมูลนับเป็นส่วนสำคัญของการทำงาน IC ตระกูลนี้จะมีการแบ่งการใช้งานย่อยออกไปเป็น 3 ขั้นด้วยกันคือ

- เพื่อจุดประสงค์ทั่วไป
- ด้วยการทำงานเฉพาะที่แอกคิวมิวเลเตอร์
- เป้าหมายการกำหนดเลขที่อยู่แอดเดรส

การทำงานลักษณะงานทั้งหมดนี้ไม่มีผลต่อแฟลก PSW ยกเว้นการใช้คำสั่ง POP หรือ MOV เข้ารจิสเตอร์ PSW

การถ่ายเทข้อมูลเพื่อจุดประสงค์ทั่วไป ได้แก่การใช้คำสั่ง

MOV ที่จะทำงานในลักษณะการถ่ายเทข้อมูลเป็นขนาดไบต์หรือบิตก็ได้

จากตัวแหล่งกำเนิดเข้าสู่ตัวรับข้อมูลในฟิลด์โอเปอร์แรนด์

PUSH จะทำงานโดยเพิ่มค่าในเรจิสเตอร์ SP ก่อนแล้วจึงถ่ายเทข้อมูลขนาด 1 ไบต์ จากตัวแหล่งกำเนิดที่ฟิลด์โอเปอร์แรนด์กำหนดไว้ไปยังบริเวณสแต็กตามตำแหน่งที่เรจิสเตอร์ SP กำหนด

POP การถ่ายเทข้อมูลขนาด 1 ไบต์ จากบริเวณสแต็กตามตำแหน่งที่เรจิสเตอร์ SP กำหนดไปยังตัวเรจิสเตอร์ที่โอเปอร์แรนด์กำหนด และหลังจากนั้นเรจิสเตอร์ SP จะลดค่าลงหนึ่งค่า

การกำหนดการถ่ายเทโดยใช้แอ็กคิวมิวเลเตอร์ จะมีคำสั่ง

XCH คำสั่งแลกเปลี่ยนขนาดไบต์ ระหว่างแหล่งกำเนิดโอเปอร์แรนด์กับแอกคิวมิวเลเตอร์

XCHD คำสั่งแลกเปลี่ยนขนาดนิบเบิ้ลทางอันดับต่ำของแหล่งกำเนิดโอเปอร์แรนด์กับนิบเบิ้ลอันดับต่ำของแอ็กคิวมิวเลเตอร์

MOVX การเคลื่อนย้ายขนาด 1 ไบต์ ระหว่างหน่วยความจำข้อมูลภายนอกกับแอ็กคิวมิวเลเตอร์ แอดเดรสภายนอกสามารถที่จะถูกกำหนดได้ด้วยเรจิสเตอร์ DPTR เต็มขนาด 64 กิโลไบต์หรือเรจิสเตอร์ R1 หรือ R0 ขนาด 8 บิต ภายใน 256 ไบต์

MOVC การเคลื่อนย้ายขนาด 1 ไบต์จากหน่วยความจำโปรแกรมเข้าสู่แอ็กคิวมิวเลเตอร์ โดยใช้ตัวโอเปอร์แรนด์ใน A เป็นค่าชนีตัวชี้ตารางข้อมูลได้ถึง 256 ไบต์ ด้วยการใช้ร่วมกับเรจิสเตอร์ DPTR หรือ PC เป็นฐานเรจิสเตอร์ และไบต์ที่ถูกกำหนดแอดเดรส จะถ่ายเทเข้าสู่แอ็กคิวมิวเลเตอร์

การถ่ายเทข้อมูลกำหนดตำแหน่งข้อมูลรหัส

MOV DPTR,#DATA เป็นการโหลดขนาดข้อมูลโดยทันที 16 บิต เข้าสู่คู่เรจิสเตอร์ DPH กับ DPL รวมเป็นเรจิสเตอร์ DATA POINTER ขนาด 16 บิต ซึ่งสามารถที่จะกำหนดตำแหน่งได้ถึง 64 กิโลไบต์

 # กลุ่มทางคณิตศาสตร์

ตัว IC มีคำสั่งเกี่ยวกับการทำงานทางคณิตศาสตร์ทางพื้นฐานสี่งานด้วยกัน และจะใช้ขนาดข้อมูล 8 บิต ที่ไม่คิดเครื่องหมายเป็นตัวคำนวนโดยตรง อย่างไรก็ตามการใช้แฟลก OVERFLOW ยังคงใช้งานการบวกและลบ เพื่อบริการข้อมูลที่เป็นตัวเลขลงตัวทางบวกและลบได้อยู่ทางคณิตศาสตร์ ยังสามารถที่จะทำงานโดยตรง ด้วยการใช้แทนค่าข้อมูลด้วย

PACK DECIMAL (BCD) การใช้งาน PACK DECIMAL คือการแทนตัวเลขฐานสิบในแต่ละหลักด้วยค่าไบนารี่ 4 บิต ดังนั้นในหนึ่งไบต์ก็จะแทนได้ 2 หลักตัวเลขฐานสิบ

คำสั่งการบวกกัน

INC (increment) เป็นการบวกหนึ่งเข้ากับแหล่งกำเนิดโอเปอร์แรนด์ และใส่ค่าใหม่กลับเข้าตัวโอเปอร์แรนด์เดิม

ADD เป็นการบวกค่าในแอ็กคิวมิวเลเตอร์เข้ากับค่าในแหล่งกำเนิดโอเปอร์แรนด์ และใส่ผลลัพธ์กลับคืนมาที่แอ็กคิวมิวเลเตอร์

ADDC (add with carry) เป็นการบวกค่าในแอ็กคิวมิวเลเตอร์กับค่าในแหล่งกำเนิดโอเปอร์แรนด์ แล้วบวกค่าที่อยู่ในบิตต้าทด (CY) และใส่ผลลัพธ์กลับคืนมาที่แอ็กคิวมิวเลเตอร์

DA (decimal-add-adjust) สำหรับการบวกกันทางตัวเลข BCD เป็นการปรับค่าผลรวม ซึ่งเป็นผลลัพธ์จากการบวกกันทางไบนารี่ของระบบตัวเลข BCD ขนาด 2 หลัก สองจำนวน การ Pack Decimal ผลรวม ด้วยการใช้คำสั่ง DA จะได้ผลลัพธ์เก็บกลับมาที่แอ็กคิวมิวเลเตอร์ ถ้าผลลัพธ์ BCD ทำให้บิตต้าทด CY เซต จะแสดงว่าค่าที่ Packed แล้ว จะมีค่ามากกว่า 99 ส่วนผลลัพธ์ตัวที่น้อยกว่า ต้าทด CY จะเคลียร์

คำสั่งการลบกัน

SUBB (Subtrac with borow) เป็นการนำตัวเลขที่อยู่ในแหล่งกำเนิดโอเปอร์แรนด์ ลบออกจากตัวเลขที่อยู่ในแอ็กคิวมิวเลเตอร์ และหลังจากนั้นก็นำค่าที่อยู่ในบิตทด CY ไปลบอีกครั้งหนึ่ง แล้วนำผลลัพธ์กลับมาเก็บที่แอ็กคิวมิวเลเตอร์

DEC (Decrement) เป็นการลบหนึ่งออกจากตัวเลขที่อยู่ในแหล่งกำเนิดโอเปอร์แรนด์ และนำผลลัพธ์กลับมาเก็บที่ตัวโอเปอร์แรนด์นั้นๆ

การคูณกัน

MUL จะเป็นคำสั่งการคูณกันแบบไม่คิดเครื่องหมายของตัวเลขที่อยู่ในแอ็กคิวมิวเลเตอร์ A กับตัวเลขในเรจิสเตอร์ B แล้วนำผลลัพธ์ที่ได้ ซึ่งขนาดสูงสุดได้ 2 ไบต์ นำกลับเก็บที่ AB โดยที่ A จะรับอันดับไบต์ ส่วน B จะรับอันดับไบต์สูง ค่าบิต OV ใน PSW จะเคลียร์ถ้าเรจิสเตอร์ใน B เป็น 0 และจะเซตถ้า B ไม่เป็น 0 ส่วนบิต CY จะเคลียร์และจะไม่มีผลต่อบิต AC

คำสั่งการหารกัน

DIV จะเป็นคำสั่งการหารกันด้วยตัวเลขที่ไม่คิดเครื่องหมายที่อยู่ในแอ็กคิว

มิวเลเตอร์ A (ถือเป็นตัวตั้ง) ที่ถูกหารด้วยตัวเลขที่อยู่ในรีจิสเตอร์ B (ถือเป็นตัวหาร) และนำผลลัพธ์ที่ได้กลับมาไว้ที่แอกคิวมิวเลเตอร์ และเศษส่วนไว้ที่รีจิสเตอร์ B การหารด้วยค่า 0 จะไม่มีผลต่อข้อมูลในรีจิสเตอร์ A และ B และจะเซตบิต OV ส่วนการหารด้วยค่าอื่น บิต OV จะเคลียร์และจะไม่มีผลต่อบิต AC ถ้าไม่มีการหารกันด้วยตัวเลขที่ไม่เป็นแบบที่กล่าวมาแล้ว ค่าบิตแฟลกต่างๆใน PSW จะมีผลดังต่อไปนี้ เนื่องจากการหารด้วยค่าตัวเลขต่างๆ

บิต CY เซต ถ้ามีการทำงานเนื่องจากผลของบิตอันดับสูงมีการทดเข้าสู่หรือยืมออกจากตัวทด

บิต AC เซต ถ้าผลจากการทำงานเกิดมีการทดลองจากนิบเบิ้ลต่ำหรือสี่บิตอันดับต่ำ ระหว่างการบวกกัน หรือมีการยืมจากนิบเบิ้ลสูงเข้าสู่นิบเบิ้ลต่ำ ระหว่างการลบกัน นอกเหนือจากนี้ บิต AC จะเคลียร์

บิต OV เซต ถ้าผลจากการทำงานเกิดตัวทดทดเข้าสู่บิตอันดับสูงสุดของผลลัพธ์ แต่ไม่มีการทดบิตสูงสุดเข้าสู่บิตตัวทด หรือในทางกลับกัน คือ OV จะเซต ถ้าผลลัพธ์ทำให้มีการทดจากบิตสูงสุดเข้าสู่บิตตัวทด แต่ไม่มีการทดเข้าสู่บิตอันดับสูงสุดของผลลัพธ์ ส่วนผลทางด้านอื่น เช่นทดทั้งสองครั้ง หรือไม่ทดทั้งสองครั้งในผลลัพธ์ บิต OV จะเคลียร์ บิต OV ยังใช้เป็น Two's Complement ทางคณิตศาสตร์เพราะมันจะเซต ถ้าผลลัพธ์ที่แสดงเครื่องหมายไม่สามารถที่จะแสดงลงภายในขนาด 8 บิต

บิต P เซต ถ้าค่าฐานตัวเลข Modulo 2 รวมกับของ 8 บิตในแอกคิวมิวเลเตอร์เป็นจำนวนคี่ และบิต P จะเคลียร์ถ้ารวมกันเป็นคู่ (Even Parity)

เมื่อค่าในบิตต่างๆ ถูกเขียนเข้าไปยังเรจิสเตอร์ PSW ค่าบิต P จะไม่เปลี่ยนแปลงค่าและจะมีผลตามค่าพาริตี้ของ A เสมอ

#กลุ่มตรรกศาสตร์

การทำงานทางพื้นฐานทางตรรกศาสตร์ของ IC ตระกูลนี้จะทำได้ทั้งขนาดไบต์และบิตโอเปอร์แรนด์ คำสั่งการทำงานโอเปอร์แรนด์ภายในตัวเองจะมี

CLR ปรับค่าในแอกคิวมิวเลเตอร์ หรือ การให้ตำแหน่งแอดเดรสตามบิตนั้นๆ เป็น 0

SETB ปรับค่าในตำแหน่งแอดเดรสตามบิตนั้นๆเป็น 1

CPL ด้วยการสั่งกลับค่า หรือ Complement ข้อมูลในแอกคิวมิวเลเตอร์ โดยไม่มีผลใดต่อค่าแฟลกใน PSW หรือการให้ตำแหน่งแอดเดรสตามบิตนั้นๆ

RL, RLC, RR, RRC, SWAP ทั้ง 5 คำสั่งนี้เป็นการสั่งทำงานการรวนบิต ที่สามารถที่สั่งให้ทำบนตัวแอกคิวมิวเลเตอร์ PL เป็นการรวนซ้าย RR เป็นการรวนขวา RLC

เป็นการวนซ้ายผ่านบิตทด C RRC เป็นการรวนขวาผ่านบิตทด C และ SWAP เป็นการวนซ้ายสี่ครั้ง สำหรับ RLC และ RRC ค่าทดแฟลก CY จะมีค่าเท่ากับค่าบิตสุดท้ายที่วนออกมา การ SWAP จะวนซ้ายค่าข้อมูลในแอกคิวมิวเลเตอร์ เป็นการเปลี่ยนค่าบิต 3 ถึง 0 กับ บิต 7 ถึง 4

คำสั่งการทำงานร่วมระหว่าง 2 โอเปอร์แรนด์

ANL เป็นการ AND กันทางตรรกศาสตร์ระหว่างแหล่งกำเนิด 2 โอเปอร์แรนด์ซึ่งจะสั่งให้ทำงานตรรกข้อมูลขนาดเป็นไบต์หรือบิตก็ได้และจะนำผลกลับมาเก็บไว้ที่ตำแหน่งตัวโอเปอร์แรนด์ที่สั่งในตัวโอเปอร์แรนด์เป็นตัวแรก

ORL เป็นการ OR ทางตรรกกันระหว่างแหล่งกำเนิด 2 โอเปอร์แรนด์ ซึ่งจะสั่งให้ทำงานตรรกข้อมูล ขนาดเป็นไบต์หรือบิตก็ได้ และจะนำผลกลับมาเก็บไว้ที่ตำแหน่งตัวโอเปอร์แรนด์ ที่สั่งในโอเปอร์แรนด์เป็นตัวแรก

XRL เป็นการ XOR กันทางตรรกระหว่างแหล่งกำเนิดกัน 2 โอเปอร์แรนด์ ซึ่งจะสั่งให้ทำงานตรรกข้อมูลขนาดเป็นไบต์หรือบิตก็ได้ และจะนำผลกลับมาเก็บไว้ที่ตำแหน่งตัวโอเปอร์แรนด์ที่สั่งในโอเปอร์แรนด์เป็นตัวแรก

#กลุ่มคำสั่งควบคุมการถ่ายเทข้อมูล

คำสั่งควบคุมการถ่ายเทข้อมูล มี 3 รูปแบบด้วยกันคือ การเรียกโปรแกรมย่อย โดยไม่ต้องตั้งข้อแม้ แล้วกลับคืนมาที่โปรแกรมหลักและการกระโดดไป และการกระโดดไปด้วยการตั้งข้อแม้ และการใช้อินเตอร์รัพท์ การใช้คำสั่งควบคุมการทำงาน ด้วยเหตุจากการกำหนดข้อแม้ของตัวโปรแกรมหลักที่ทำงานอยู่ จะไม่เป็นไปตามลำดับในหน่วยความจำโปรแกรม

การเรียกโปรแกรมย่อยโดยไม่ต้องตั้งข้อแม้ แล้วกลับคืนมาที่โปรแกรมหลัก และการกระโดดไปตามคำสั่งการเรียกโปรแกรมย่อย โดยไม่ต้องตั้งข้อแม้ แล้วกลับคือมา และการกระโดดไป เป็นการควบคุมค่าข้อมูลของตัวนับโปรแกรมในขณะนั้น ให้กระโดดไปยังตำแหน่งเอดเดรสใหม่ที่ต้องการจะกระโดดไป การถ่ายเทมีให้ใช้ทั้งแบบโดยตรงและทางอ้อม

ACALL และ LCALL จะทำงานตามลำดับดังนี้ โดยคำสั่ง จะ PUSH ค่าตำแหน่งของคำสั่งตัวต่อมาของโปรแกรมหลักไว้ที่บริเวณสแต็ก และถ่ายเทควบคุมเปลี่ยนตำแหน่งใหม่เป็นแอดเดรสที่จะกระโดดไป ACALL จะมีขนาดคำสั่ง 2 ไบต์ โดยใช้เป้าหมายแอดเดรสที่จะกระโดดไปภายใน 2 กิโลเพจ หรือใช้รหัสแอดเดรสทั้งหมด 11 บิต LCALL จะมีขนาดคำสั่ง 3 ไบต์ เป้าหมายแอดเดรส จะกระโดดไปได้เต็ม 64 กิโลไบต์หรือใช้รหัสแอดเดรสเต็ม 16 บิต

ในการใช้แอดเดรสขนาด 11 บิตของ ACALL จะเป็นการเปลี่ยนค่าบิตของ PC เฉพาะช่วง 11 บิต กลุ่มอันดับต่ำเท่านั้น ส่วนอีก 5 บิตในตำแหน่งกลุ่มอันดับสูงของ PC ในขระที่เหลือ จะไม่เปลี่ยนแปลง ถ้าคำสั่ง ACALL อยู่ที่ตำแหน่ง 2 ไบต์สุดท้าย ของเพจขนาด 2 กิโลไบต์หลังจากการเรียกโปรแกรมย่อยแล้วจะเป็นการเรียกตำแหน่งเพจใหม่ เพราะ PC จะเพิ่มค่าไปถึงตำแหน่งคำสั่งตัวต่อมา ซึ่งจะเป็นการเรียกเพจใหม่

RET คำสั่งนี้เป็นการควบคุมการถ่ายเทกลับคืนสู่โปรแกรมหลักซึ่งตำแหน่งได้เก็บอยู่ที่สแต็ก ด้วยคำสั่งการเรียกทำงานโปรแกรมย่อย ก่อนหน้าที่จะทำโปรแกรมย่อย คำสั่ง RET นี้ จะดึงเอาข้อมูลจากสแต็กที่ชี้ด้วย SP หรือที่เรียกว่า POP มาไว้ที่ PC และค่า SP จะลดลง 2 ค่า

AJMP, LJMP, SJMP, เป็นการควบคุมถ่ายเทไปยังเป้าหมายที่ถูกกำหนดในโอเปอร์แรนด์ การทำงานของคำสั่ง AJMP และ LJMP จะมีลักษณะการเปลี่ยนแปลงและการทำงานเช่นเดียวกับ ACALL และ LCALL ยกเว้นที่ไม่มีการกลับมาที่ทำงานที่เดิม ส่วน SJMP เป็นกระโดดถอยหลังหรือเดินหน้า ภายใน 256 ไบต์เท่านั้น จากตำแหน่งของคำสั่งนี้ ต่อจาก SJMP ซึ่งจะกระโดดได้ -128 ถึง 127

JMP @A+DPTR คำสั่งนี้ใช้ความสัมพันธ์ร่วมกับรีจิสเตอร์ DPTR ค่าโอเปอร์แรนด์ใน A จะใช้เป็น OFSET (0-255) ต่อค่าแอดเดรสใน DPIR ดังนั้น ค่าแอดเดรสที่ถูกชี้ด้วย EFFECTIVE จะกระโดไปในส่วนใดๆ ของหน่วยความจำโปรแกรม

การกระโดดแบบมีข้อแม้

คำสั่งการกระโดดแบบมีข้อแม้จะกระโดดไปสู่เป้าหมายที่กำหนดได้ ขึ้นอยู่กับข้อแม้ที่ตั้งไว้ และจะกระโดดไปได้ไกลจากตำแหน่งของคำสั่งตัวต่อจากคำสั่งนี้ได้ภายใน -128 ถึง 127 ซึ่งจะมีคำสั่งต่างๆพอสรุปได้ดังนี้

JZ จะกระโดดได้ถ้าค่าในแอคคิามิวเลเตอร์เป็น 0

JNZ จะกระโดดได้ถึงค่าในแอกคิวมิวเลเตอร์ไม่เป็น 0

JC จะกระโดดถ้าค่าในแฟลกตัวทดเซต

JNC จะกระโดดถ้าค่าในแฟลกตัวทดไม่เซตหรือเคลียร์

JB จะกระโดดได้ถ้าค่าในบิตที่ถูกกำหนดด้วยการกำหนดเลขที่อยู่โดยตรง (DIRECT ADDRESS) เซต

JNB จะกระโดดได้ถ้าค่าในบิตที่ถูกกำหนดด้วยการกำหนดเลขที่อยู่โดยตรง ไม่เซตหรือเคลียร์

JBC จะกระโดดได้ถ้าค่าในบิตที่ถูกกำหนดด้วยการกำหนดเลขที่อยู่โดยตรงเซตและจะเคลียร์ค่าในบิตใหม่ตามตำแหน่งของการกำหนดเลขที่อยู่โดยตรง

CJNZ เป็นการเปรียบเทียบกันระหว่างโอเปอร์แรนด์ตัวแรก กับโอเปอร์แรนด์ ตัวที่สอง และจะกระโดดไปถ้าหากทั้งสองค่านี้ไม่เท่ากัน และบิตทด

CY จะเซตถ้าหากค่าโอเปอร์แรนด์ มีค่าน้อยกว่าค่าโอเปอร์แรนด์ตัวที่สองแต่ถ้าค่าโอเปอร์แรนด์กลับกัน บิต CY จะเคลียร์ การเปรียบเทียบกันสามารถเปรียบเทียบได้ระหว่าง A กับค่าไบต์ของหน่วยความจำของมูลภายในหรือระหว่างค่าที่ให้โดยทันทีกับ A หรือกับตัวรีจิสเตอร์อื่นๆในแบลงค์ที่ถูกเรียกให้ทำงานหรือกับข้อมูลในหน่วยความจำ ข้อมูลภายในที่กำหนดเลขที่อยู่โดยอ้อม จากตัว @Ri (I=1,2,3...)

DJNZ เป็นการลดค่าข้อมูลภายในที่กำหนดจากตัวแหล่งกำเนิดโอเปอร์แรนด์และนำผลกลับไปตามการกำหนดของโอเปอร์แรนด์ตัวนั้น การกระโดดจะเกิดขึ้นถ้าการลดค่านั้นแล้วมีผลลัพธ์ไม่เป็นตัวแหล่งกำเนิดโอเปอร์แรนด์ของคำสั่งนี้จะเป็นค่าไบต์ใดๆ ในหน่วยความจำข้อมูลภายในและการกำหนดเลขที่อยู่ ทั้งแบบโดยตรง หรือโดยตัวรีจิสเตอร์สามารถที่จะถูกใช้เป็นตัวกำหนดตำแหน่งได้จากตำแหน่งกำเนิดโอเปอรแรนด์

การกลับคืนจากอินเตอร์รัพท์

RETI ควบคุมการถ่ายเทเช่นเดียวกับ RET แต่จะเพิ่มความสามารถในการ ENABLE อินเตอร์รัพท์ของการจัดระดับทางไพเออริตี้ที่กำลังทำอยู่

คำอธิบายความหมายของคำย่อต่างๆ ที่ใช้ในชุดคำสั่งจะอธิบายในชุดต่อไปนี้

A,ACC แอกคิวมิวเลเตอร์

AC ตัวทดสำรอง (Auxiliary carry)

Addr ค่าแอดเดรสในหน่วยความจำขนาด 12 บิต

Bb ตำแหน่งที่ใช้งาน โดย b มีค่าเท่ากับ 0-7

BUS พอร์ตของบัส

C บิตทด

CLK สัญญาณนาฬิกา

CNT ตัวนับจำนวนเหตุการณ์

CRR การแปลงเลขฐานจากผลลัพธ์ในรีจิสเตอร์

D นิวมอนิคส์ ที่ใช้ตัวเลขขนาด 4 บิต

data ข้อมูลตัวเลขขนาด 8 บิต หรือนิพจน์

I การอินเตอร์รัพท์

PC	ตัวนับโปรแกรม
PSW	คำแสดงสถานะโปรแกรม
Ri	ตัวชี้ข้อมูลในหน่วยความจำ (i= 0 หรือ 1)
Rr	ตัวรีจิสเตอร์ที่ถูกใช้ (r = 0,1 หรือ 0-7)
SP	ตัวชี้สแต็ก
T	ตัวจับเวลา
X	นิวมอนิคส์ที่ใช้แรมภายนอก

#ตัวบอกการใช้โหมดการกำหนดเฉพที่อยู่ข้อมูลโดยทันที

@	ตัวบอกการใช้โหมดการกำหนดเลขที่อยู่ข้อมูลโดยทางอ้อม
$	ค่าของตัวนับโปรแกรมในปัจจุบัน
(X)	ค่าข้อมูลที่อยู่ X
((X))	ค่าข้อมูลตำแหน่งแอดเดรสที่ถูกชี้โดย X
<----	ถูกแทนด้วย

รายละเอียดชุดคำสั่ง

ACALL addr 11

ACALL จะเรียกโปรแกรมย่อยในแอดเดรสที่กำหนดแบบไม่มีเงื่อนไข คำสั่งนี้จะเพิ่มค่าใน PC 2 ครั้ง เพื่อรับแอดเดรสจากคำสั่งต่อไป แล้ว PUSH ผลลัพธ์ขนาด 16 บิตไปเก็บไว้ที่สแต็ก และเพิ่มค่าตัวชี้สแต็ก 2 ครั้ง โดยแอดเดรสที่ถูกเรียกเข้าใน PC จะได้รับจากรหัสบิต 7-5 และไบต์ที่ 2 ของคำสั่ง CALL ดังนั้นโปรแกรมย่อยจะอยู่ภายในขอบเขต 2 กิโลไบต์ของหน่วยความจำโปรแกรม

add เป็นการบวกตัวแปรจากไบต์ที่ถูกกำหนดกับค่าในแอกคิวมิวเลเตอร์ แล้วจะเก็บผลลัพธ์ไว้ในแอกคิวมิวเลเตอร์ ถ้ามีตัวทดจากบิต7 หรือบิต 3 บิตทดหรือauxiliary จะเซตตามลำดับ ถ้าบวกกันโดยไม่ใช้เครื่องหมาย บิตทดจะชี้การเกิด overflow (OV) OV จะเซต เมื่อมีตัวทดจากบิต 6 ไม่ออกนอก บิต 7 เมื่อมีการบวกตัวเลขลงตัวที่มีเครื่องหมาย OV จะชี้จำนวนลบจากผลรวมของโอเปอร์แรนด์ตัวที่เป็นบวก หรือจะชี้ผลบวกของโอเปอร์แรนด์ที่เป็นลบ

Add with carry ADDC เป็นการบวกตัวแปรที่ถูกกำหนดกับแอกคิวมิวเลเตอร์พร้อมทั้งแฟลกทดแล้วเก็บข้อมูลในแอกคิวมิวเลเตอร์ แฟลกตัวทด และ auxiliary

จะเซตตามลำดับ ถ้ามีการทดจากบิต 7 หรือบิต 3 เมื่อมีการบวก ตัวเลข ลงตัวที่ไม่มีเครื่องหมายการเซตค่าในแฟลกตัวทดจะเป็นการแสดงถึงการเกิดตัวเลขเกินค่าเมื่อเกิด overflow ใช้กับ <srcbuyte> ในโหมดรีจิสเตอร์ โหมดโดยตรง โหมดรีจิสเตอร์โดยอ้อมและโหมดโดยทันที

Absolute Jump AJMP จะทำการกระโดดการทำงานของโปรแกรมไปยังอดเดรสที่ถูกกำหนดอยู่ในรูปแบบของ ออปโค้ดไบต์แรกที่บิต 7-5 และไบตที่ 2 ของชุดคำสั่ง โดยบิตที่เหลือที่อยู่นัยสำคัญสูงอีก 5 บิตของ PC หลังจากที่ PC นี้เพิ่มค่าขึ้นครั้งละ 1 สองครั้ง แล้วจะยังคงที่ไม่ถูกทำให้เปลี่ยนแปลงจากคำสั่งนี้ ฉะนั้นเป้าหมาย ของโปรแกรมที่จะกระโดดไปจะต้องอยู่ในของเขต 2 กิโลไบต์ ต่อจากแอดเดรสของคำสั่งที่ต่อจาก AJMP

ANL จะทำงานตรรก AND ในแต่ละไบต์ระหว่างตัวแปรที่ถูกกำหนดและเก็บผลลัพธ์ไว้ที่ <dest-byte> โดยที่ไม่ผลต่อแฟลกตัวโอเปอร์แรนด์ทั้งสอง สามารถที่จะกำหนดด้วยโหมดกำหนดเลขที่ 6 โหมด เมื่อตัว <destbyte> จะเป็นแอกคิวมิวเลเตอร์ ตัว <src-byte> ข้อมูลสามารถจะให้อยู่ในโหมดรีจิสเตอร์โดยตรง รีจิสเตอร์โดยอ้อมหรือแอดเดรสโดยทันที และเมื่อตัว<dest-byte> ใช้เป็นโหมดกำหนดโดยตรง <src-byte> สามารถจะเป็นแอกคิวมิวเลเตอร์ หรือโหมดโดยทันที

ตรรก AND สำหรับตัวขนาดบิต

ถ้าค่าบูลลีนของ <src-byte> เป็น 0 ดังนั้นจะเป็นการเคลียร์แฟลกตัวทด ถ้าตรรกเป็นหนึ่งแฟลกตัวทดจะคงสถานะของตัวมันเอง slash "/" หน้าตัวโอเปอร์แรนด์ในภาษาแอสแซมบลี้จะเป็นการกลับค่าบิตที่ใช้ โดยเลขที่อยู่แอดเดรสของบิตนั้นซึ่งค่าในบิต <src-byte> และแฟลกอื่นไม่ถูกกระทบกระเทือนหลังทำคำสั่งนี้แล้ว เปรียบเทียบและกระโดดหากไม่เท่ากันCJNE จะเปรียบเทียบข้อมูลของโอเปอร์แรนด์ 2 ตัวแรก ซึ่งถ้าไม่เท่ากันก็จะระโปยังแอดเดรสที่ได้จากการรวมของค่าสัมพันธ์ที่คิดเครื่องหมายของไบต์ที่ 3 ของสั่งนี้กับค่าใน PC หลังจาก PC เพิ่มค่าไปเพื่อจะทำชุดคำสั่งต่อไป แฟลกตัวทดจะถ้าตัวเลขจำนวนเต็มที่ไม่คิดเครื่องหมายของ <dest-byte> น้อยกว่าค่าตัวเลขจำนวนที่ไม่คิดเครื่องหมาย <src-byte> ถ้าเป็นอย่างอื่นคือมากกว่าหรือเท่ากับ<src-byte> ทุกตัวทดจะเคลียร์ ค่าข้อมูลโอเปอร์แรนด์จะไม่มี การเปลี่ยนค่าทั้ง 2 โอเปอร์แรนด์

สามารถใช้โหมดการกำหนดเลขที่อยู่ได้ 4 โหมด โดยที่แอกคิวมิวเลเตอร์อาจจะใช้เป็นตัวเปรียบเทียบตัวเลขที่ถูกกำหนดอลขที่อยู่ในโหมดโดยตรงหรือโหมดโดยทันที และตัวเลขที่ถูกกำหนดโดยอ้อมในตำแหน่ง RAM หรือของข้อมูลในรีจิสเตอร์ ทำงานกับข้อมูลในโหมดโดยตรง

เคลียร์แอกคิวมิวเลเตอร์

ค่าในแอกคิวมิวเลเตอร์จะถูกเคลียร์ทั้งหมด <คือ 0 ทุกบิต> โดยไม่มีผลต่อค่าแฟลกใดๆ

เคลียร์บิต

ค่าบิตที่ถูกกำหนดจะถูกเคลียร์ <reset= 0> โดยไม่มีผลกระทบต่อค่าแฟลกใดๆ

Complement Accumulator

แต่ละบิตของแอกคิวมิวเลเตอร์จะถูกทำให้เป็น 1's complement หรือกลับค่า

Complement Bit

บิตที่ถูกกำหนดจะถูกกลับค่า CPL สามารถทำงานที่บิตหดหรือบิทที่ถูกกำหนดเลขที่โดยตรงเมื่อคำสั่งนี้ถูกใช้ในการปรับปรุงขาเอาต์พุตค่าที่ถูกอ่านจะเป็นค่าเอาต์พุตแลทช์ข้อมูลของขานั้นไม่ใช้อ่านจากขาโดยตรง

การปรับค่าภายในแอกคิวมิวเลเตอร์เป็นเลขฐานสิบหลังทำคำสั่ง

ค่าบิตในแอกคิวมิวเลเตอร์จะถูกปรับให้อยู่ในรูปแบบของตัวเลข BINARY CODED DECIMAL (BCD) เป็นตัวเลข 2 หลัก หลักละ 4 บิต ในรูปแบบ PACKED BCD หลังทำคำสั่ง ADD หรือ ADDC ถ้าค่าในบิต 3-0 ของแอกคิวมิวเลเตอร์มากกว่า 9_{10} (XXXX1010-XXXX1111) หรือถ้าบิต AC เป็น 1 ตัวแอกคิวมิวเลเตอร์จะถูกเพิ่มค่ามากขึ้นด้วยการเพิ่มค่า 6_{10} เข้าเพื่อให้ได้ค่า BCD ที่เหมาะสม การบวกภายในแฟลกตัวทดจะเซต ถ้าผลลัพท์เกิน 4 บิต low-order โดยที่ตัวทดจะเซตและบิตนิบเบิ้ลสูง ก็จะถูกตรวจสอบเช่นเดียวกันถ้าบิต 4-7 มีค่าเกิน 9 (1010XXXX-1111XXXX) หรือถ้าบิตทดเป็น 1 บิต เหล่านี้ก็จะถูกเพิ่มขึ้นอีก 6 ก็จะเกิด OVER FLOW บิตทดก็จะปรับเป็นหนึ่งและจะ

ไม่มีผลต่อบิต OV โดยที่ความหมายว่าผลรวมของค่า BCD ทั้งสองค่า มากกว่า 100 คำสั่งนี้จะเป็นในการเปลี่ยนเป็น decimal ซึ่งจะใช้จำนวน 00H, 06H. 60H, 66H เพิ่มไปยังแอกคิวมิวเลเตอร์และยังขึ้นอยู่กับ PSW ด้วย

Decrement

ตัวแปรที่ถูกกำหนดจะลดลงไปหนึ่งค่าโดยไม่ผลกระทบต่อค่าแฟลกใดๆ ใช้กับแอกคิวมิวเลเตอร์ รีจิสเตอร์ direct address หรือรีจิสเตอร์-indirect

DIVIDE

DIV เป็นการหารตัวเลขลงตัว 8 บิต ในแอกคิวมิวเลเตอร์ด้วยตัวเลขลงตัวในรีจิสเตอร์ B โดยไม่คิดเครื่องหมาย แฟลกต้วทดกับ overflow จะเคลียร์ ถ้า B เป็น 00H ค่าจะกลับสู่แอกคิวมิวเลเตอร์และรีจิสเตอร์ B จะ undifine overflag จะเคลียร์ โดย over flow จะเซตถ้ามีเศษจะนำไปเก็บที่ B

การลดค่าหนึ่งค่าและกระโดดถ้าตัวแปรที่กำหนดไม่เป็นศูนย์

DJNZ เป็นการลดค่าตัวแปรที่ถูกกำหนดลง 1 ค่า ถ้าผลลัพธ์ไม่เป็น 0 ก็จะกระโดดไปยังแอดเดรสที่ถูกกำหนดจากโอเปอร์แรนด์ตัวที่ 2 หรือรหัสตัวสุดท้ายของคำสั่งที่คิดเครื่องหมายกับค่าใน PC ที่ชี้แอดเดรสของคำสั่งชุดต่อมา แต่ถ้าค่าที่ถูกกำหนดมีค่าเป็น 00H ก็จะถูกลดเป็น UNDERFLOW มีค่า 0FFH โดยที่ไม่มีไม่มีผลต่อแฟลกใดๆ ตำแหน่งที่ถูกกำหนดให้มีค่าลดลงอาจถูกกำหนดด้วยค่ารีจิสเตอร์ หรือโหมดการกำหนดเลขที่อยู่โดยตรงก็ได้

INCREMENT

INC เป็นการเพิ่มตัวแปรที่ถูกกำหนดด้วย 1 ถ้าค่าต้นเป็น 0FFH จะเปลี่ยนเป็น 00H โดยไม่มีการเปลี่ยนแปลง แฟลกใช้กับรีจิสเตอร์. direct address หรือรีจิสเตอร์ indirect

INCREMENT DATA POINTER

เป็นการเพิ่มค่าของริสเตอรตัวชี้ข้อมูล (DPTR) ขนาด 16 บิต ขึ้น 1 ค่า โดย OVERFLOW ที่เกิดจากไบต์นัยต่ำที่เพิ่มจาก 0FFH เป็น 00H จะ

ไปเพิ่มค่าในไบต์นัยสูงคำสั่งนี้เป็นการเพิ่มค่าในรีจิสเตอร์ขนาด 16 บิต เท่านั้น โดยไม่มีผลต่อแฟลกใดๆ

กระโดดค่าบิตที่ถูกกำหนดเป็น 1

ถ้าบิตที่ถูกกำหนดตำแหน่งมีค่าเป็น 1 คำสั่งจะกระโดดไปยัง address ที่กำหนดโดยมีจุดปลาย ที่ได้จากผลของการบวกกันด้วยค่า Signed relative displacement ของไบต์ที่ 3 ของคำสั่งนี้ กับค่าของ PC ที่ชี้คำสั่งไบต์แรกของคำสั่งตัวต่อมาแล้วเก็บไว้ที่ PC หลังจาก PC เพิ่มค่าแล้วจะเป็นการกระโดดตามค่า PC โดยที่ ไม่มีผลต่อแฟลกใดๆ

กระโดดถ้าบิตที่ถูกกำหนดเป็น 1 และจะเคลียร์บิตนั้น

ถ้าบิตที่ถูกกำหนดมีค่าเป็น 1 คำสั่งเคลียร์บิตามตำแหน่งนั้นและกระโดไปยัง address ที่กำหนดไปยังจุดปลายที่ได้จากผลของการบวกกันด้วยค่า signed relative displacement ของไบต์ที่ 3 ของคำสั่งนี้กับค่าของ PC ที่ชี้คำสั่งไบต์แรกของคำสั่งตัวต่อมา แล้วเก็บไว้ที่ PC หลังจาก PC เพิ่มค่าแล้ว จะเป็นการกระโดดตามค่า PC โดยที่ไม่มีผลต่อแฟลกใดๆ

กระโดดถ้าบิตทดเป็น 1

ถ้าบิตทดเป็น 1 จะกระโดดปยัง address ที่กำหนดในคำสั่งด้วยค่าที่ได้จากการบวกของค่า signed relative diplacement ไปที่ 2 ของคำสั่งนี้กับของ PC ที่ชี้ไบต์แรกของคำสั่งตัวต่อมา แล้วเก็บไว้ที่ PC ลังจาก PC เพิ่มค่าแล้วจะเป็นการกระโดดตามค่า PC โดยไม่มีผลต่อแฟลกใดๆ

กระโดดโดยอ้อม

เป็นการโดดเป็นำม่มีเงื่อนไขโดยการกระโดดไปตามค่าผลลัพธ์ที่ได้จากการบวกเอาค่าข้อมูลในแอกคิวมิวเลเตอร์กับ DPTR ขนาด 16 บิต ซึ่งผลจากการบวกกันของไบต์ต่ำ ถ้าเกิดตัวทดจะทดไปสู่บิตในไบต์สูง ซึ่งผลที่ได้จะเก็บใน PC โดยที่ค่าของแอกคิวมิวเลเตอร์และ DPTR ไม่มีการเปลี่ยนแปลงและไม่ผลต่อแฟลกใดๆ

กระโดดถ้าบิตที่ถูกกำหนดไม่เป็น 1

ถ้าบิตที่ถูกกำหนดเป็น 0 จะกระโดดไปยัง address ที่กำหนดในคำสั่งด้วยค่าได้จากการบวกกันของค่า signed relative displacement ไบต์ที่ 3 ของคำสั่งนี้กับค่าของ PC ที่ชี้ไบต์แรกของคำสั่งตัวต่อมา แล้วเก็บไว้ที่

PC หลังจาก PC เพิ่มค่าแล้วจะเป็นการกระโดดตามค่า PC โดยที่ไม่ผลต่อแฟลกใดๆ

กระโดดถ้าบิตทดไม่ถูกเซต

ถ้าบิตทดเป็น 0 จะกระโดดไปยัง address ที่ถูกกำหนดในคำสั่งด้วยค่าที่ได้จากการบวกกันของค่า signed relative displacement ไบต์ที่ 3 ของคำสั่งนี้กับค่าของ PC ที่ชี้ไบต์แรกของคำสั่งตัวต่อมา แล้วเก็บไว้ที่ PC หลังจาก PC เพิ่มค่าแล้วจะเป็นการ กระโดดตามค่า PC โดยที่ไม่มีผลต่อแฟลกใดๆ

กระโดดถ้าค่าในแอกคิวมิวเลเตอร์ไม่เป็น 0

ถ้าค่าในแอกคิวมิวเลเตอร์ไม่เป็น 0 หมด จะกระโดดไปยัง addressที่กำหนดในคำสั่งด้วยค่าที่ได้จากการบวกกันของค่า signed relative displacement ไบต์ที่ 3 ของคำสั่งนี้กับค่าของ PC ที่ชี้ไบต์แรกของคำสั่งต้าต่อมาแล้วเก็บไว้ที่ PC หลังจาก PC เพิ่มค่าแล้ว จะเป็นการกระโดดตามค่า PC โดยที่ไม่มีผลต่อแฟลกใดๆ

กระโดดถ้าค่าในแอกคิวมิวเลเตอร์เป็น 0 หมด

ถ้าค่าในแอกคิวมิวเลเตอร์เป็น 0 หมด จะกระโดดไปยัง address ที่กำหนดในคำสั่งด้วยค่าที่ได้จากการบวกกันของค่า signed relative displacement ไบต์ที่ 3 ของคำสั่งนี้กับค่าของ PC ที่ชี้ไบต์แรกของคำสั่งตัวต่อมาแล้วเก็บไว้ที่ PC หลังจาก PC เพิ่มค่าแล้วจะเป็นการกระโดดตามค่า PC โดยไม่มีผลต่อแฟลกใดๆ

LONG CALL

LCALL จะเรียกโปรแกรมในย่อยตาม address ที่ถูกกำหนดมาในคำสั่ง คำสั่งจะเพิ่มค่าที่ PC ขึ้นอีก 3 เพื่อการให้ PC ชี้ที่คำสั่งชุดต่อมา แล้วจึงจะ PUSH ผลลัพธ์ของที่ได้ไปเก็บไว้ที่หน่วยความจำบริเวณสแต็ก ซึ่งถูก SP โดยการเก็บไบต์ของ PC เป็นตัวแรกของไบต์สูง ตัวต่อมา SP ก็จะเพิ่มค่าขึ้น 2 หลังจากนั้น PC จะถูกโหลดด้วยค่าไบต์สูงและไบต์ต่ำของ address เริ่มต้น โปรแกรมย่อยที่ถูกเรียกด้วยคำสั่งนี้คือ ไบต์ตัวที่ 2 และไบต์ตัวที่ 3 ของคำสั่งรหัสคำสั่งนี้ตามลำดับ โปรแกรมก็จะทำงานอย่างต่อเนื่องด้วยคำสั่งที่ address นี้ ดังนั้นตัวโปรแกรมย่อยสามารถจะอยู่ ณ

ที่ไดของหน่วยความจำโปรแกรมขนาด 64 ไบต์ โดย ไม่มีผลต่อแฟลกใดๆ

LONG JUMP

LJMP เป็นการกระโดดไปยังค่า address ที่ถูกกำหนด โดยไม่มีเงื่อนไข โดยการโหลดค่าไบต์สูงและไบต์ต่ำด้วยรหัสคำสั่งของไบต์ตัวที่ 2 และ 3 ตามลำดับเข้า PC ดังนั้น ตำแหน่ง address ที่จะกระโดดไปสามารถที่จะอยู่ในตำแหน่งใดๆภายในหน่วยความจำ โปรแกรม 64 กิโลไบต์ โดยไม่ผลต่อแฟลกใดๆ

MOVE BYTE VARIABLE

ไบต์ตัวแปรที่ถูกกำหนดโดยโอเปอร์แรนด์ตัวที่ 2 จะถูกย้ายมายังตำแหน่ง ที่ถูกกำหนดโดยโอเปอร์แรนด์ตัวแรก โดยค่าข้อมูลที่ถูกย้ายจะยังคงไม่เปลี่ยนแปลงซึ่งจะไม่มีผลต่อรีจิสเตอร์ตัวอื่นและค่าแฟลกใดๆ คำสั่งนี้มีความยืดหยุ่นสูงเนื่องจากสามารถที่จะกำหนดโหมดเลขที่อยู่ต่างๆ ของข้อมูลทั้งแบบ source และ destination ไปถึง 15 แบบ

MOVE BIT DATA

รหัสโอเปอร์แรนด์ตัวที่ 2 จะหนดตัวแปรบูลลีนและลอกไปยังตำแหน่งที่กำหนดโดยรหัสโอเปอร์แรนด์ตัวแรก การใช้คำสั่งนี้จะต้องกำหนดให้แฟลกตัวทดเป็น โอเปอร์แรนด์ตัวใดตัวหนึ่ง ที่ถูกกำหนดให้เป็นเลขที่อยู่โดยตรง คำสั่งนี้จะไม่มีผลต่อรีจิสเตอร์ตัวอื่นและค่าแฟลกใดๆ

โหลดค่าคงที่ขนาด 16 บิตเข้าที่ตัวชี้ข้อมูล (DATA POINTER)

ตัวชี้ข้อมูล DPTR จะถูกโหลดด้วยค่าคงที่ ที่ถูกกำหนดขนาด 16 บิท ด้ายค่าที่อยู่ในไบต์ตัวที่ 2 และตัวที่ 3 ของคำสั่ง โดยที่ค่าไบต์ตัวที่ 2 จะเป็นไบต์นัยต่ำและไบต์ที่ 3 จะเป็นไบต์นัยสูงของ DPTR โดยที่ไม่มีผลต่อค่าแฟลกใดๆ

ย้ายค่ารหัสไบต์ของหน่วยความจำโปรแกรม

การใช้คำสั่งนี้เป็นการโหลดค่ารหัส หรือค่าคงที่จากหน่วยความจำโปรแกรมด้วยแอดเดรสที่ชี้ ที่อยู่ของรหัสนี้ ซึ่งได้จากการรวมกันของค่าจำนวนเต็มที่อยู่ในแอกคิวมิวเลเตอร์กับค่าที่อยู่ใน <base-reg> ขนาด 16 บิต ซึ่งอาจจะเป็นรีจิสเตอร์ PC หรือ DPTR ถ้าเป็นกรณีของ PC

PC จะต้องถูกเพิ่มค่าเป็นค่าแอดเดรสที่ชี้ที่คำสั่งถัดไปของคำสั่งนี้ก่อนจึงจะรวมกับค่าในแอกคิวมิวเลเตอร์ส่วนของ DPTR จะคงใช้ค่าเดิมของ DPTR ที่ไม่เปลี่ยนแปลง ตัวทดที่เกิดจากการบวกกันของไบต์นัยต่ำจะทดไปรวมกับไบต์ นัยสูง โดยไม่มีผลกระทบต่อค่าแฟลกใดๆ

MOV ข้อมูลจากหน่วยความจำภายนอก

คำสั่ง MOV X จะทำการย้ายค่าข้อมูลระหว่าง ACC กับขนาด 1 ไบต์ ของหน่วยความจำภายนอก คำสั่งมี 2 แบบ เพื่อการแอดเดรสข้อมูลมีขนาด 8 บิต และ 16 บิต ของการกำหนดเลขที่อยู่ข้อมูลภายนอก คำสั่งแบบแรกจะใช้ค่าใน R_0 หรือ R_1 ของการใช้รีจิสเตอร์แบงค์ ในขณะนั้นโดยที่ค่าแอดเดรส 8 บิตนี้จะส่งออกที่ port 0 ซึ่งจะมัลติแพล็กส์กับข้อมูลที่ส่งออกที่ port 0 นี้เช่นกัน ด้วยขนาดแอดเดรส 8 บิตนี้จะพอเพียงสำหรับการ decode เพื่อการขยาย port ภายนอกหรือการแอดเดรสข้อมูล ram ภายนอกได้ถึง 256 ไบต์ ซึ่งหากต้องการใช้ข้อมูลที่มีขนาดมากกว่านี้ ก็สามารถที่จะแอดเดรส ด้วยการกำหนดที่ค่าแอดเดรสไบต์นัยสูงก่อนที่ port 2 ในแบบที่ 2 ของคำสั่งนี้ สามารถใช้ตัว DPTR เป็นตัวชี้แอเดรสขนาด 16 บิตได้พร้อมกัน โดยที่ไบต์นัยสูลของ DPTR (หรือ DPH) จะส่งไปที่ port 2 ไบต์นัยต่ำของ DPTR หรือ DPL จะส่งไปที่ port 0 ซึ่ง เป็น port ที่มีมัลติแพล็กส์ กับข้อมูลการทำงานของ port 2 ชี้ตัวรีจิสเตอร์ port 2 ของ SFR จะยังคงเก็บค่าเดิมอยู่ขณะที่ buffer port 2 จะส่งค่าของ DPH ออกที่ขา port 2 ใช้รูป แบบคำสั่งนี้ จะเพิ่มความเร็วและประสิทธิภาพขึ้น ในการเข้าถึงข้อมูลที่มีขนาดใหญ่ (สูงสุด 64 กิโลไบต์) เพราะไม่จำเป็นต้องเซตค่า port address นัยสูงก่อน

MULTIPLY

MUL AB เป็นการข้อมูลจำนวนเต็มไม่คิดเครื่องหมายขนาด 8 บิต ใน ACC กับรีจิสเตอร์ B โดยผลคูณที่ได้มีขนาด 16 บิต ACC จะเก็บไบต์นัยต่ำและไบต์นัยสูงจะเก็บไว้ที่ B ถ้าผลคูณมากกว่า 255 (0FFH) บิตแฟลก 0V จะถูกเซตเป็น 1 แฟลกทดจะเคลียร์ตลอด No การทำงาน การ execute จะต่อเนื่องไปตามคำสั่งไม่มีแฟลกใด หรือ รีจิสเตอร์ใดกระทบกระเทือนนอกจาก PC

LOGICAL-OR for byte variables

ORL จะแสดงการตรรก OR กันไปตามบิตระหว่างตัวแปรที่ถูกกำหนดในคำสั่งและเก็บผลลัพธ์ขนาดไบต์ไว้ที่ ที่กำหนดด้วย <dest-byte> โดยไม่ผลต่อบิตแฟลกใดๆ โอเปอร์แรนด์ทั้งสองตัวจะใช้การกำหนดที่อยู่ของตัวแปรได้ 6 โหมดโดยถ้า <dest-byte> เป็น ACC <src-byte> จะใช้โหมดรีจิสเตอร์โหมดโดยตรง โหมดโดยทันที หรือโหมดโดยอ้อมหรือถ้า <src-byte> เป็น ACC หรือโหมดโดยทันที <dest-byte> จะถูกกำหนดเป็นโหมดที่อยู่โดยตรง ถ้าใช้คำสั่งนี้แก้ไขที่พอร์ตเอาต์พุตค่าที่ใช้จะเป็นค่าที่แท้จริงที่อ่านจากวงจรข้อมูลแลทช์เอาต์พุตไม่ใช่จากขาอินพุต

การตรรก OR ด้ายขนาดตัวแปรบิต

เซตค่าแฟลกทดถ้าค่าทางบูลลีนต้องการตรรก 1 คงค่าเดิมให้ค่าเดิมถ้าเป็นอย่างอื่นจะทิ้งตัวทดไว้ การใช้ slash "/" ในรูปแบบของภาษาแอสแซมบลี้ จะหมายถึงการกลับค่าเดิมของการกำหนดตัวแปรบิตนั้นๆ (ก่อนที่ OR ก้น) แต่ค่าของบิตใน <src-bit> จะคงค่าเดิม

POP FROM STACK

ค่าข้อมูลของ ram ภายในที่ถูกกำหนดตำแหน่งด้วยตัวชี้ stack จะถูกอ่านเข้าตามตำแหน่งต่างๆ ที่ถูกกำหนดเลขที่อยู่โดยตรงจากแอดเดรสในโอเปอร์แรนด์ของคำสั่งและตัวชี้ stack จะถูกลดค่าลง 1 ค่า หลังการอ่านข้อมูลแล้ว (หรือ POP ข้อมูลแล้ว) โดยที่ไม่มีผลกระทบต่อค่าแฟลกใดๆ

PUSH ONTO STACK

ตัวชี้ stack จะเพิ่มค่าขึ้น 1 ค่า เราจึงลอกข้อมูลที่ถูกกำหนดเลขที่อยู่โดยตรงในคำสั่งเข้าไปยัง ram ภายในที่ถูกชี้ด้วยค่าตัวชี้ stack ที่เพิ่มค่าแล้วโดยไม่มีผลต่อแฟลกใดๆ

RETURN จากโปรแกรมย่อย

RET จะ POP เอาไบต์สูงและไบต์ต่ำของ PC ที่เก็บอยู่ในบริเวณ stack ด้วยการลดค่าตัวชี้ stack 2 ครั้ง โปรแกรมก็จะคืนกลับค่าของ PC เพื่อทำงานคำสั่งที่ต่อจากตัวคำสั่ง ACALL หรือ L CALL ได้ โดยทันที โดยไม่มีผลกระทบต่อแฟลกใดๆ

RETURN จากอินเตอร์รัพต์

RETI จะ POP เอาไบต์สูงและไบต์ต่ำของ PC ที่เก็บอยู่ในบริเวณ stack ด้วยการ ลดค่าตัวชี้ stack 2 ครั้ง และค่าทางตรรกอินเตอร์รัพต์อินาเบิล ทางระดับความสำคัญจะคืนกลับทำให้สามารถทำการอินเตอร์รัพต์ตามความสำคัญที่ตั้งไว้ต่อไป ได้ จะไม่มีผลกระทบใดๆ ต่อตัวรีจิสเตอร์และตัวรีจิสเตอร์ PSW จะไม่มีการรับ ค่าต่างๆก่อนเข้าสู่การอินเตอร์รัพต์กลับคืนโดยอัตโนมัติ โปรแกรมจะทำงานต่อไปที่แอดเดรสของคำสั่งตัวต่อมาของคำสั่งที่ได้รับสัญญาณอินเตอร์รัพต์ขณะนั้นโดยที่เมื่อทำคำสั่งนั้นแล้ว เข้าทำโปรแกรการอินเตอร์รัพต์หลังจากทำคำสั่ง RETI ก็จะกลับ มาทำคำสั่งหลักต้วต่อมาดังกล่าว โดยทันทีถ้ามีการอินเตอร์รัพต์ตัวอื่นที่มีระดับเดียวกันหรือระดับต่ำกว่ารออยู่ เมื่อคำสั่ง RETI ถูกทำงานไปแล้วความสำคัญก็จะต้องคืนกลับไปทำงาน โปรแกรมหลักที่ทำงานอยู่ก่อนที่จะเกิดการอินเตอร์รัพต์ โดยให้ทำงาน 1 คำสั่งแล้วจึงจะเข้าอินเตอร์รัพต์ต้วต่อมาที่รออยู่ก่อนแล้วได้

เคลื่อนแอกคิวมิวเลเตอร์ ทางซ้าย

จำนวน 8 บิต ในแอกคิวมิวเลเตอร์จะถูกวนกลับไปทางซ้าย โดยที่ตำแหน่งบิต 7 จะวนกลับมาที่ตำแหน่งบิต 0 โดยไม่มีผลกระทบต่อค่าแฟลกใดๆ

เคลื่อนแอกคิวมิวเลเตอร์ ทางซ้าย ผ่านแฟลกต้วทด

จำนวนบิต 8 บิตในแอกคิวมิวเลเตอร์รวมทั้งบิตทดใน PSW จะถูกวนกลับไป ทางซ้ายโดยที่ ที่ตำแหน่งบิต 7 จะวนเข้าบิตทด และบิตทดจะวนเข้าตำแหน่งบิต 0 โดยไม่มีผลกระทบต่อแฟลกใดๆ

เลื่อนแอกคิวมิวเลเตอร์ไปทางขวา

จำนวนบิต 8 บิต ในแอกคิวมิวเลเตอร์ จะถูกวนกลับไปทางขวา โดยที่ตำแหน่งบิต 0 จะวนเข้าตำแหน่งบิต 7 และไม่มีผลกระทบต่อแฟลกใดๆ เลื่อนแอกคิวมิวเลเตอร์ทางขวาผ่านบิตทดจำนวนนบิต 8 บิต ในแอกคิวมิวเลเตอร์ รวมทั้งบิตทดใน PSU จะถูกวนกลับไปทางขวา โดยที่ตำแหน่งบิต 0 จะวนเข้าที่บิตทด และบิตทดจะวนเข้าที่ตำแหน่งบิต 7 โดยไม่มีผลกระทบต่อค่าแฟลกใดๆ

เซต BIT

SETB เป็นการเซตบิตที่ถูกกำหนดเป็น 1 สามารถใช้คำสั่งนี้กระทำต่อบิตทดหรือบิตที่ถูกกำหนดเลขที่อยู่โดยตรงใดๆ ได้โดยไม่มีผลกระทบต่อค่าแฟลกใดๆ

SHOT JUMP

เป็นคำสั่งการกระโดดแบบไม่มีเงื่อนไขไปยังแอดเดรสที่ถูกกำหนด โดยจะกระโดดไปตามค่าแอสเดรสที่ได้จากการบวกเอาข้อมูลแบบ sign displacement ของรหัสคำสั่งไบต์ที่ 2 กับตัวชี้โปรแกรม PC ที่มีค่าแอสเดรสที่ชี้ไบต์แรกของ-คำสั่งต้าต่อมาแล้วเก็บไว้ที่ PC เพราะฉนั้นตำแหน่งของคำสั่งที่จะกระโดดไปจะอยู่ข้างหน้าไม่ห่างเกิน 127 ไบต์ หรือ ตามมาข้างหลังไม่เกิน 128 ไบต์

SUBB

SUBB เป็นการลบตัวแปรที่ถูกกำหนดและบิตทดออกจากค่าในแอกคิวมิวเลเตอร์และเก็บผลกลับคืนเข้าที่แอกคิวมิวเลเตอร์ คำสั่ง SUBB จะเซตค่าบิตทด (ตัวยืม) ถ้าเกิดมีการยืมค่าจากบิต 7 และจะเคลียร์บิตทด ถ้าไม่มีการยืมค่า ถ้าบิตทด C ถูกเซตก่อนที่จะทำคำสั่ง SUBB นี้ จะเป็นการชี้ว่าต้องการตัวยืมก่อนที่ทำการขั้นตอนการลบต่างๆ ดังนั้น ตัวทดจะทำการลบ กว่าออกจากตัวแอกคิวมิวเลเตอร์ก่อน แล้วจึงลบด้วยค่าตัวแปรที่ถูกกำหนดด้วยค่าตัวแปรที่ถูกกำหนดในโอเปอร์แรนด์ <scr-byte> บิต AC จะเซตถ้ามีตัวยืมจากบิต 3 และเคลียร์ถ้าไม่มีการยืมดังกล่าว OV จะเซตถ้ามีตัวยืมจากบิต 6 แต่ไม่เกิดใน 7 หรือเกิดยืมในบิต 7 แต่ไม่เกิดในบิต 6 เมื่อมีการลบกันตัวเลขจำนวนเติมที่คิดเครื่องหมายบิต OV จะเป็นตัวชี้ตัวเลขลบที่เกิดขึ้นเมื่อค่าตัวเลขลบถูกลบด้วยค่าตัวเลขบวก หรือเกิดเป็นผลลัพธ์ค่าตัวเลขบวกเมื่อค่าตัวเลขบวกถูกออกจากค่าตัวเลขลบ

SWAP นิบเบิ้ล ในแอกคิวมิวเลเตอร์

SWAP A เป็นการแลกเปลี่ยนต่ำแหน่งข้อมูลขนาด 4 บิต (หรือ NIBBLE) ภายใน ACC คือบิต 3-0 กับ 7-4 โดยไม่มีผลกระทบต่อค่าแฟลกใดๆแลกเปลี่ยนข้อมูลแอกคิวมิวเลเตอร์กับข้อมูลตัวแปร

XCH จะโหลดค่าข้อมูลที่ถูกกำหนดด้วยค่าตัวแปร<byte> เก็บใน ACC และขณะเดียวกันก็จะโหลดค่าของ ACC เก็บในตำแหน่งที่ตัวแปร

กำหนด โดยทั้งใน source และdestination ของโอเปอร์แรนด์สามารสให้ทำงานในโหมดรีจิสเตอร์โดยตรงหรือโดยอ้อมด้วยรีจิสเตอร์

EXCHANGE DIGIT

XCHD เป็นการสลับค่าขนาดนิบเบิ้ล (nibble) กันระหว่างนิบเบิ้ลนัยต่ำของค่าข้อมูลของ ACC ซึ่งโดยทั่วไปจะเป็นการแทนค่าตัวเลข BCD (หรือตัวเลขฐานสิบหก) กับค่าข้อมูลนิบเบิ้ลนัยต่ำที่เก็บอยู่ใน ram ภายในโดยการกำหนดตำแหน่งที่อยู่ด้วยโหมดการกำหนดเลขที่อยู่โดยรีจิสเตอร์ ส่วนนิบเบิ้ลนัยสูง (บิต 4-7) ของแต่รีจิสเตอร์จะไม่มีผลต่อการเปลี่ยนแปลงนี้ และไม่มีผลกระทบต่อค่าแฟลกใดๆ

LOGICAL EXCLUSIVE-OR FOR BYTE VARIABLES

XRL เป็นการแสดงค่าการทำ EX-OR ระหว่างตัวแปรในโหมดโดยอ้อมกับค่าที่เก็บผลลัพธ์ขนาดไบต์ไว้ที่ ที่กำหนดด้วย <dest-byte>โดยไม่มีผลต่อบิตแฟลกใดๆ โอเปอร์แรนด์ทั้ง 2 ตัวจะใช้การกำหนดเลขที่อยู่ของตัวแปรได้ 6 โหมดโดยถ้า <dest-byte> เป็น ACC.<src-byte>จะใช้โหมดรีจิสเตอร์โหมดโดยตรงโหมดโดยทันทีหรือโหมดโดยอ้อมรีจิสเตอร์ หรือถ้า <src-byte>เป็น ACC หรือโหมดโดยทันที <dest-byte> จะถูกำหนดเลขที่อยู่โดยตรง

การใช้งานตัวไมโครคอนโทรเลอร์

ในเรื่องนี้จะแบ่งการใช้งานของตัวไมโครคอนโทรเลอร์ออกเป็น 3 หัวข้อด้วยกันคือ

1. เทคนิคการโปรแกรม
2. เทคนิคการโปรแกรมให้ทำงานกับอุปกรณ์ต่อพ่วง
3. ตัวอย่างวงจรการต่อพ่วงกับอุปกรณ์ภายนอก

ในหัวข้อแรกจะเป็นตัวอย่างการเขียนโปรแกรมแบบโปรแกรมย่อยของการใช้งานด้านการควบคุมต่างๆ และบางโปรแกรมจะเป็นโปรแกรมทางคณิตสาศตร์และเทคนิคการโปรแกรม LOOK-UP TABLE ว่าจะมีวิทีการโปรแกรมได้อย่างไร

ส่วนเทคนิคการโปรแกรมให้ทำงานกับอุปกรณ์ต่อพ่วง จะรวมเอาลักษณะการโปรแกรมที่จะต้องควบคุมกับพอร์ตอนุกรม พอร์ตขนานและการโปรแกรมตัวจับเวลา/ตัวนับของไอซีตระกูลนี้อธิบายถึงการติดตั้งซอฟต์แวร์ในการเริ่มใช้ติดตั้งตัวจับเวลา การรับและส่งข้อมูลทางอนุกรมที่ต้องการเรียกใช้จากโปรแกรมอื่น

หัวข้อสุดท้ายจะเป็นวงจรตัวอย่างของการต่อพ่วงกับอุปกรณ์ภายนอก ไม่ว่าจะเป็นหน่วยความจำคือ ram หรือ eprom หรือการขยายพอร์ตต่างๆ

1. เทคนิคการโปรแกรม

1.1 โปรแกรมย่อยการแปลงระบบฐานตัวเลข

การใช้คำสั่งหารสามารถที่จะใช้เป็นส่วนหนึ่งของโปรแกรมในการแปลงตัวเลขจากระบบหนึ่งไปเป็นอีกระบบหนึ่งได้ เช่น BINBCD จะเป็นโปรแกรมย่อยที่จะแปลงระบบตัวเลขไบนารี่ที่ไม่คิดเครื่องหมายที่มีค่าอยู่ในแอกคิวมิวเลเตอร์ ซึ่งจะมีค่าได้ตั้งแต่ 0-255 ไปเป็นตัวเลข BCD ซึ่งจะแทนตัวเลขนี้ได้ด้วยขนาด 2 ไบต์ โดยที่หลักร้อยจะอยู่ในตัวแปร HUND ส่วนหลักสิบและหลักหน่วยจะรวมกันเป็น PACK BCD ของตัวแปร TENONE ขนาด 1 ไบต์ที่แทนตัวเลขหลักสิบหลักละ 4 บิต

```
;
: MULBCD UNPACK TWO BCD DIGITS RECEIVED IN ACCUMULATOR,
; FIND THEIR PRODUCT, AND RETURN PRODUCT
; IN PACKED BCD FORMAT IN ACCUMULATOR
;
MULBCD:     MOV B,#10H          ; DIVIDE INPUT BY 16
            DIV  AB             ; A&B HOLD SEPARATED DIGITS
                                ; (EACH RIGHT JUSTIFIED IN REGISTER)
```

```
        MUL  AB          ; A HOLD PRODUCT IN BINARY FORMAT(0-99)
                         ; (DECIMAL =0-63H)
        MOV  B,#10       ; DIVIDE PRODUCT BY 10
        DIV  AB          ; A HOLD NUMBER OF TJENS, B HOLDS
        SWAP A
        ORL  A,B         ; PACKDIGITS
        RET
```

1.2 โปรแกรมย่อยทางคณิตสาสตร์ที่แทนตัวเลขหลายหลัก

ด้วยคำสั่ง ADDC และ SUBB ช่วยให้เราสามารถโปรแกรมตัวเลขของการบวกหรือลบให้มีการใช้ตัวทดหรือตัวยืมได้ เมื่อมีการทำงานทางคณิตสาสตร์ที่แทนตัวเลขหลยหลักด้วยการใช้หลักการวนลูปหรือทำซ้ำคำสั่งจำนวนครั้งเท่ากับจำนวนหลักตัวเลขที่นำมาคิด และถ้าตัวเลขหลายหลักที่ไม่นับเครื่องหมายนำมาคำนวนเกิดมีตัวเลขทดที่เกินหลักที่ตั้งไว้ในโปรแกรมก็สามารถที่จะตรวจสอบได้เพื่อขยายหลักตัวเลข ในทำนองเดียวกันถ้าเกิดตัวที่ถูกลบไม่พอกับจำนวนหลักที่มีก็สามารถที่จะเขียนโปรแกรมการตรวจสอบได้เช่นกัน ดังตัวอย่างโปรแกรมย่อยที่ให้มานี้

```
;
: SUBSTR SUBSTRACT STRING INDICATED BY R1
: FROM STRING INDICATED BY R0 T0
;PRECISION INDICATED BY R2
: CHECK FOR SIGNED UNDERFLOW WHEN DONE.
;
SUBSTR:  CLR  C          : BORROW=0
SUBS1:   MOV  A,@R0      ; LOAD MINUEND BYTE
         SUBB A,@R1      ; SUBTRACT SUBTRAHEND BYTE
         MOV  @R0,A      ; STORE DIFERENCE BYTE
         INC  R0
         INC  R1
         DJNZ R2,SUBS1   ; LOOP UNTIL DONE
;
; WHEN DONE, TEST IF OVERFLOW OCCURED
; ON LAST ITERATION OF LOOP
```

```
                JNB  OV, OV_OK
;
; (OVERFLOW RECOVERY ROUTINE)
OV_OK:          RET          ; RETURN
```

1.3 สำคับการโปรแกรทำงานแบบ LOOK-UP TABLE

จะมีสองลักษณะงานของการใช้คำสั่ง MOVC ในการสำคับงานสามขั้นตอนเพื่อที่จะทำงานแบบ LOOK-UP TABLE หรือการอ่านค่าตารางข้อมูลคงที่ที่มีอยู่ใน EPROM หรือ ROM มีดังนี้คือ

1.3.1ในการใช้ลักษณะงานของ DPTR จะต้องโหลดค่าตำแหน่งแอดเดรสเริ่มต้นของตารางข้อมูลเข้าตัวชี้ข้อมูล DPTR ต่อไปก็โหลดค่าดัชนีที่ต้องการข้อมูลในตารางลงในแอกคิวมิวเลเตอร์และสำดับที่สามก็ใช้คำสั่ง MOVC A,@A+DPTR ซึ่งการใช้ตัวชี้ข้อมูลเช่นนี้ช่วยให้สามารถชี้ตารางข้อมูลได้ทั้งแบบสั้นๆ และข้อมูลที่ซับซ้อนมากขึ้นได้ และถ้าใช้ตารางข้อมูลที่มีมากกว่า 256 ค่า ก็สามารถที่จะคำนวนค่าตัวรีจิสเตอร์ DPH และ DPL หรือปรับปรุงค่าได้ด้วยคำสั่งคณิตสาสตร์ที่มีอยู่ในชุดคำสั่งนี้ เพื่อที่จะได้ค่าตำแหน่งข้อมูลนั้นๆ

1.3.2 ลักษณะที่สองเป็นการใช้ฐานข้อมูลของตัวชี้โปรแกรม PC ซึ่งจะใช้ได้เฉพาะตารางข้อมูลแบบสั้น ภายในเพจแอดเดรสเดียวกันกับโปรแกรมที่เรียกดูข้อมูล การใช้ลักษณะงานเช่นนี้จะมีข้อดี ที่ไม่มีผลต่อตัวชี้ข้อมูล DPTR แต่อย่างไร การใช้ลักษณะงานนี้ จะมีประโยชน์สำหรับโปรแกรมบริการอินเตอร์รัพต์ หรือดปรแกรมต่างๆ ที่มีสถานะที่ต้องใช้ต้า DPTR อยู่แล้ว สำดับการโปรแกรมจะมี 3 ขั้นตอนเช่นเดียวกับลักษณะงานแรก คือโหลดค่าดัชนีเข้าที่แอกคิวมิวเลเตอร์ และปรับปรุงค่า ออฟเซต ตำแหน่งค่าที่อยู่แอดเดรสของตัวชี้โปรแกรมที่จะใช้คำสั่ง LOOK-UP TABLE ในขณะนั้นด้วยการบวกค่าในออฟเซตเข้ากับค่าในแอกคิวมิวเลเตอร์ แล้วจึงเริ่มใช้คำสั่ง MOVC A,@A+PC

ตามตัวอย่างข้างล่างนี้จะเป็นการใช้งานในการเก็บค่าแบบแมตริก ที่มีหลายมิติ (DIMENSION) โดยใช้ตารางแทนค่าต่างๆ ในแมตริดนี้ในการรับค่าจากตารางนี้ จะใช้ตัวแปรที่แทนค่าดัชนีของแมตริก สำหรับมิติของแมตริกจะใช้ MDIMEN x NDIMEN) โดยกำหนดตัวแปรแอดเดรส base เป็นค่าเริ่มแรกของตารางและให้ INDEXI, INDEXJ เป็นดัชนีของ 2 มิตินี้ ด้วย ตำแหน่งของข้อมูลในแมตริก (INDEXI,INDEXJ) จะถูกคำนวนได้ด้วยสูตร

INTRY ADDRESS = [BASE + (NDIMEN x INDEXI) + INDEXJ]

ในโปรแกรย่อย MATRIX1 สามารถที่จะเข้าถึงข้อมูลได้ไม่มากกว่า 255 ELEMENTS เช่นตัวอย่าง มิติ 11 x 21 จะมีค่า 231 ข้อมูลตารางจะเป็นค่าคงที่ ที่ใช้คำสั่งเทียบด้วย DB (DATA BYTE) แทนหนึ่งไบต์

```
;MATRX1 LOAD CONSTANT READ FROM TWO DIMANSIONAL LOOK-UP
;       TABLE IN PROGRAM MEMORY INTO ACCUMULATOR
;       USING LOCAL TABLE LOOK-UP INSTRUCTION, 'MOVC A,@A+PC'.
;       THE TOTAL NUMBER OF TABLE ENTRIES IS ASSUMED TO
:       BE SMALL, I.E. LESS THAN ABOUT 255 ENTRIES.
:       TABLE USED IN THISS EXAMPLE IS 11 x 21.
;       DESIRED ENTRY ADDRESSIS GIVEN BY THE FORMULA,
;
:       [(BASE ADDRESS) = (21 x INDEXI)= (INDEXJ)]
INDEXI      EQU R6              :FIRST COORDINATE OF ENTRY (0-10)
INDEXJ      DATA 23H    ;SECOND COORDINATE OF ENTRY (0-20)
;
MATRX1:     MOV   A,INDEXI
            MOV   B,#21
            MUL   AB            ;(21 x  INDEXI)
            ADD   A,INDEXJ      :ADD IN OFFSET WITHIN ROW
:
:           ALLOW FOR INSTRUCTION BYTE BETWEEN 'MOVC' AND
:           ENRY (0,0)
            INC   A
            MOVC A,@A+PC
            RET
BASE1:      DB    1             ;(ENTRY 0,0)
            DB    2             ;(ENTRY 0,1)
;           ..    ....
            DB    21            ;(ENTRY 0,20)
            DB    22            ;(ENTRY 1,0)
```

```
;           ..      ....
            DB      42          ;(ENTRY 1,20)
;           ..      ....
;           ..      ....
            DB      231         ;(ENTRY 10,20)
```

ส่วนโปรแกรมย่อยMATRI2 จะเป็นโปรแกรมที่ขยายขนาดของแมตริกได้กว้างโดยไม่มีจำกะด ด้วยการใช้คำสั่งการคูณและบวกแบบตัวเลขขนาด / ไบต์ และใช้ลักษณะงานของ DPTR เป็นตัวชี้รับข้อมูลจากตาราง มีข้อจำกัดเพียงว่าตัวดัชนีจะต้องมีค่าระหว่าง 0 ถึง 255.

```
MATRX2:     MOV   A,INDEXI      ;LOAD  FIRST  COORDINATE
            MOV   B,#NDIMEN
            MOV   AB            ;INDEXI x NDIMEN
            ADD   A,#LOW(BASE2)       ;ADD  IN 16-BIT BASE ADDRESS
            MOV   DPL,A
            MOV   A,B
            ADDC  A,#HIGHT(BASE2)
            MOV   DPH,A         ;DPTR = (BASE ADDR) + (INDEXI + NDIMEN)
            MOV   A,INDEXJ
            MOVC  A,@A+DPTR     ;ADD INDEXJ AND FETCH BYTE
            RET
            ...   .....
BASE2:      DB    0             ;(ENTRY 0,0)
            DB    0             ;(ENTRY 0,1)
            ..    ....
            DB    0             ;(ENTRY 0,NDIMEN-1)
            DB    0             ;(ENTRY 1,0)
            ..    ....
            DB    0             ;(ENTRY 1,NDIMEN-1)
            ..    ....
            DB    0             ;(ENTRY MDIMEN-1,NDIMEN-1)
```

1.4 การเก็บสถานะต่างๆของ CPU ระหว่างการเข้าบริการอินเตอร์รัพต์

เมื่อฮาร์ดแวร์ของ IC รับรู้การร้องขออินเตอร์รัพต์โปรแกรมจะควบคุมการกระโดดไปยังตำแหน่งของการบริการนี้อย่างอัตโนมัติ คล้ายกับการใช้คำสั่ง LCALL และค่าแอดเดรสของโปรแกรมหลักจะถูกเก็บไว้ที่ TOP OF STACK ก่อนจะถูกเรียกเข้าโปรแกรมการอินเตอร์รัพต์ หลังจากที่ทำงานบริการนี้เรียบร้อยแล้วด้วยคำสั่ง RETI สุดท้าย ก็จะคืนค่าแอดเดรสของโปรแกรมหลักที่เก็บไว้ที่ STACK เพื่อให้สามารถคืนกลับไปยังตำแหน่งโปรแกรมหลักต่อไปได้ และขณะเดียวกันโปรแกรมอินเตอร์รัพต์จะต้องไม่ไปเปลี่ยนค่าต่างๆของรีจิสเตอร์ที่โปรแกรมหลักใช้อยู่ ดังนั้นก่อนที่จะใช้บริการอินเตอร์รัพต์ ภายในโปรแกรมของอินเตอร์รัพต์จะต้องมีคำสั่งการเก็บค่าต่างๆที่จำเป็นของโปรแกรม และจะต้องคืนค่าเหล่านี้หลังการทำงานบริการนี้แล้วซึ่งโปรแกรมตัวอย่างลักษณะการทำงานนี้เขียนได้ดังนี้

```
;
LOC-TMP     EQU   $              ;REMEMBER LOCATION COUNTER
:
            ORG   0003H          ;STARTING ADDRESS FOR INTERUPT
ROUTINE
            LJMP  SERVER         ;JUMP TO ACTUAL SERVICE ROUTINE
LOCATE
            ....  .....
            ORG   LOC-TMP        ;RESTORE LOCATION COUNTER
SERVER:     PUSH  PSW            ;
            PUSH  ACC            ;SAVE ACCUMULATOR
            PUSH  B              ;SAVE B REGISTER
            PUSH  DPL            ;SAVE DATA POINTER
            PUSH  DPH
            PUSH  PSW.#00001000B      ;SELECT BANK REGISTER BANK1
;           ...   .....
;           ...   .....
            POP   DPH            ;RESTORE REGISTER IN REVERSE ORDER
            POP   DPL
            POP   B
```

```
POP    ACC
POP    PSW          ;RESTORE PSSW AND RE-SELECT RIGINAL
                    ;REGISTEEER BANK
RETI                ;RETURN TO MAIN PROGRAM AND RESTORE
                    ;INTERUPT LOGIC
```

ดังนั้นตามตัวอย่างถ้า SPมีค่า 1FH เมื่อมีการบริการอินเตอร์รัพต์ค่าใน STACK ก็จะ PUSH เก็บค่าของรีจิสเตอร์ต่างๆ ตามตำแหน่ง สุดท้าย SP จะชี้ที่ 26H ก่อนจะสิ้นสุดการอินเตอร์รัพต์ ก็จะคืนค่ารีจิสเตอร์ต่างๆให้ด้วยคำสั่ง POP ตามลำดับของโปรแกรมแบบเข้าก่อนออกหลัง (FILO) โปรแกรมตามตัวอย่างนี้มักจะใช้เป็นหลักในการโปรแกรมอินเตอร์รัพต์แบบต่างๆ

1.5 การผ่านค่าพารามิเตอร์ต่างๆด้วย STACK

เราสามารถใช้ stack เป็นตัวผ่านค่าตัวแปรพารามิเตอร์ต่างๆ จากหรือเข้าสู่โปรแกรมย่อย โปรแกรมย่อยสามารถกำหนดเลขที่อยู่โดยอัตโนมัติเองค่าพารามิเตอร์ที่ได้จากข้อมูลที่ใช้แสตกชี้หรือ POP เอาตัวชี้ค่าแสตกเก็บชั่วคราวเข้าในรีจิสเตอร์ชั่วคราวก่อนเข้ากระบวนการทำงาน ข้อดีอย่างหนึ่งที่ให้มาตามตัวอย่างนี้คือ แบบง่ายๆ ที่ตัวแปรไม่จำเป็นต้องถูกย้ายที่อยู่ใหม่สำหรับตัวพารามิเตอร์การผ่านค่าพารามิเตอร์อาจมีเป็นจำนวนมาก และถูกเรียกจากโปรแกรมย่อยหลายครั้ง จะมีการใช้เทคนิคการเรียกที่แตกต่างในการเรียกค่าเหล่านี้มาใช้ ตัวอย่างโปรแกรมย่อย HEXASC จะแปลงค่าฐาน 16 เป็นรหัส ASCII สำหรับตัวเลขหลักต่ำครั้งละหลัก โดยที่โปรแกรมย่อยเริ่มแรกจะอ่านค่าพารามิเตอร์ที่เก็บอยู่บนแสตกที่ถูกเรียกจากโปรแกรมหลัก RO และค่าแสตกเริ่มในโปรแกรมจะเป็นที่เก็บค่าตัวชี้ ตำแหน่งโปรแกรมหลักก่อนเรียกตัวนับโปรแกรมย่อย เริ่มต้นจะลดค่าแสตก 2 ตำแหน่ง ด้วยการถอนค่าในรีจิสเตอร์ที่ถูกพักช่วยคราวด้วยค่าแสตกในขณะนั้นชั่วคราว 2 ค่า ก็จะเป็นค่าที่ชี้ตัวแปรพารามิเตอร์ เพื่อนำค่าเลขฐาน 16 มาได้ ก็จะทำการแปลงเป็นรหัส ASCII ด้วยการเปรียบเทียบตามตำแหน่งของตารางที่เก็บรหัส ASCII คงที่ เมื่อได้ค่ารหัส ASCII แล้วก็จะเก็บคืนผ่านเข้าในบริเวณแสตก ก่อนจะคืนกลับโปรแกรมหลัก เนื่องจากโปรแกรมย่อยไม่ได้ไปเปลี่ยนค่าของตัวชี้แสตกแต่อย่างไรก็จะคืนค่าตัวชี้โปรแกรมให้กลับคืนสู่โปรแกรมหลักได้ และช่วยให้โปรแกรมนำค่าพารามิเตอร์ที่เปลี่ยนเป็น ASCII เรียบร้อยแล้ว จากบริเวณแสตกคืนกลับมาได้เช่นกัน

ตัวโปรแกรมหลักตามตัวอย่างข้างล่างนี้ เป็นการแปลงตัวเลข BCD 3 หลัก ก็จะเขียนผ่านค่าตัวแปรเข้าแสตกก่อนที่ เรียกโปรแกรมย่อย HEXASCII ได้ด้วยคำสั่ง PUSH เอาค่าตัวเลขฐาน 16 เข้า และ POPเอาค่าตัวเลขรหัส ASCII ออกสู่แอกคิวมิวเลเตอร์ หลังการเรียกโปรแกรมย่อย

แปลงเป็นตัวเลขแล้ว แล้วจึงส่วค่ารหัส ASCII จากแอกคิวมิวเลเตอร์ออกสู่ PORT ด้วยการเรียกโปรแกรมย่อย SP-OUT

```
HEXASC:   MOV   RO,SP       ;
          DEC   RO          :ACCESS LOCATION PARAMETER PUSHED
          DEC   RO
          XCH   A,@RO       ;READ INPUT PARAMETER AND SAVE
                            ;ACCUMULATOR
          ANL   A,#0FH      ;MASK ALL BUT LOW-ORDER 4 BITS
          ADD   A,#02       :ALLOW FOR OFFSET FROM MOVC TO TABLE
          MOVC  A,@A+PC     ;READ LOOK-UP TABLE ENTRY
          XCH   A,@RO       :PASS BACK TRANSLATED VALUE AND
                            ;RESTORE ACCUMULATOR
          RET               :RETURN TO  MAIN PROGRAM
ASCTBL:   DB    '0'         ;ASCII CODE FOR NUMBER 00-0FH
          DB    '1'
          DB    '2'
          DB    '3'
          DB    '4'
          DB    '5'
          DB    '6'
          DB    '7'
          DB    '8'
          DB    '9'
          DB    'A'
          DB    'B'
          DB    'C'
          DB    'D'
          DB    'E'
          DB    'F'
;
```

```
;MAIN PROGRAM
;
        PUSH  HUND        ;
        CALL  HEXASC      ;CONVERT HUNDREDS DIGIT
        POP   ACC
        CALL  SP-OUT      ;TRANSMIT HUNDREDS CHARACTER
        PUSH  TENONE      ;
        CALL  HEXASC      ;CONVERT ONE'S PLACE DIGIT
        MOV   A,TENONE    ;BUT LEAVE ON STACK!
        SWAP  A           ;RIGHT JUSTIFY TEN'S PLACE
        PUSH  ACC         ;
        CALL  HEXASC      ;CONVERT TEN'S PLACE DIGIT
        POP   ACC
        CALL  SP-OUT      ;TRANSMIT TEN'S PLACE CHARACTER
        POP   ACC
        CALL  SP-OUT      ;TRANSMIT ONE'S PLACE CHARACTER
```

1.6 เทคนิคการกระโดดไปหลายทางตามลักษณะงาน

มีข้อแตกต่างมากมาย ในการกระโดดไปยังส่วนใดใดของโปรแกรมโดยจะมีการกำหนดหรือหาค่าตำแหน่งของคำสั่งนั้นขึ้นมาในช่วงการทำงานในขณะนั้น ซึ่งข้อดีนี้ทำให้ตัวโปรแกรมมีความอ่อนตัวในการใช้งานชนิดต่างๆ

ในสถานะของการกระโดดหลายทางนี้ ตำแหน่งที่กระโดดจะสามารถรู้ได้ต่อเมื่อมีการแอสแซมบลี้โปรแกรมนั้นแล้ว หนึ่งในจำนวนหลายๆ งานจะถูกเลือกให้ทำงานขึ้นอยู่กับค่าดัชนีที่ถูกหามาได้ในขณะนั้น ที่กำลังให้โปรแกรมทำงานอยู่ วิธีที่ดีที่สุดที่จะใช้ให้โปรแกรมทำงานในลักษณะนั้นก็คือการใช้คำสั่ง MOVC และการกระโดดโดยทางอ้อม ด้วยการใช้ค่า OFFSET ของตารางค่าต่างอย่างสั้นๆ มีในตัวโปรแกรมเป็นตัวชี้ ค่าสัมพันธ์การเริ่มตำแหน่งโปรแกรมทั่วไปที่ต้องการให้มี

คำสั่ง JMP @A+DPTR จะทำงานในลักษณะการกระโดดไปยังตำแหน่งที่ถูกหามาได้กับตัวชี้ตารางของการกระโดดค้ายโปรแกรมตัวอย่าง ซึ่งคำสั่งจะบวกค่าที่อยู่ในแอกคิวมิวเลเตอร์แบบไม่คิดเครื่องหมายที่หามาได้กับค่าตัวชี้ข้อมูล DPTR ขนาด 16 บิต ผลรวมที่ได้จะเป็นค่าตัวชี้แอดเดรสของโปรแกรมที่คำสั่งนี้จะกระโดดไปเพื่อทำงานตามลักษณะงานต่างๆ ที่ถูกคำนวนให้ทำในระหว่างที่ให้ตัวโปรแกรมทำงานอยู่

ตามตัวอย่างข้างล่างนี้จะอ่านค่าที่กำหนดให้กระโดดเข้ามา 1 ใน 4 งานเก็บเข้าในแอกคิวมิวเลเตอร์ จากตัวแปร MEMSEL ซึ่งเป็นตัวชี้ที่เก็บอยู่ในแรมภายใน ซึ่งลักษณะงานทั้ง 4 จะเป็นการอ่านค่าข้อมูลจากหน่วยความจำที่แตกต่างกันทั้งขนาดและที่อยู่ โดยใช้ค่าใน R0 เป็นตัวชี้หรืออาจใช้ R1 ร่วมด้วย เมื่อเป็นข้อมูลขนาดใหญ่ ประโยชน์ที่ใช้ลักษณะงานพิมพ์ที่อาจมีลักษณะขนาดและ บัฟเฟอร์แตกต่างกัน การเขียนโปรแกรมตามตัวอย่างข้างล่างนี้จะมีข้อชี้ขนาดของตารางกระโดดรวมทั้งงานโปรแกรมที่ให้ทำงานชนิดต่างๆ รวมกันแล้วจะต้องไม่เกิน 256 ไบต์ รหัสคำสั่งโดยที่ไม่ขึ้นกับค่าแพงแอดเดรส

```
;
MEMSEL      EQU   R3
;
JUMP_4:     MOV   A,MEMSEL
            MOV   DPTR,#JMPTBL
            MOVC  A,@A+DPTR
            JMP   @A+DPTR
JUPTBL:           DB    MEMSP0-JMPTBL
            DB    MEMSP1-JMPTBL
            DB    MEMSP2-JMPTBL
            DB    MEMSP3-JMPTBL
MEMSP0:     MOV   A,@R0          ;READ FROM INTERNAL RAM
            RET
MEMSP1      MOVX  A,@R0          ;READ 256 BYTE EXTERNAL RAM
            RET
MEMSP2      MOV   DPL,R0         ;READ 64 BYTE EXTERNAL RAM
            MOV   DPH,R1
```

```
                MOVX A,@DPTR
                RET
MEMSP3:         MOV  A,R1           ;READ 4K BYTE EXTERNAL RAM
                ANL  A,#07H
                ANL  P1,#11111000B
                ORL  P1,A
                MOVX A,@R0
                RET
```

แต่ถ้าหากต้องการงานที่มีขนาดใหญ่กว่านี้จะมีวิธีเดียว แต่จะต้องไม่เกินการกระโดดไป 128 งาน จากการกำหนดตารางกระโดดที่มีความยาวได้ 1 เพจพอดี และโปรแกรมย่อยทั้งหมดจะต้องอยู่ภายใน 2k เพจของหน่วยความจำโปรแกรมโดยจะมีวิธีตามตัวอย่าง

```
OPTION          EQU  R3
;               ...  ....
;               ...  ....
JMP128          MOV  A,OPTION
                RL   A              ;MULTIPLY BY 2 FOR 2-BYTE JUMP TABLE
                MOV  DPTR,#INSTBL;FIRST ENTRY IN JMP TABLE
                JMP  @A+DPTR
;               ...  ....
INSTBL:         AJMP PROC00 ;128 CONSEUTIVE
                AJMP PROC01         ;JMP INSTRUCTION
                AJMP PROC02
;               ...  ....
;               ...  ....
                AJMP PROC7E
                AJMP PROC7F
;
RTEMP           EQU  R7
;               ...  ....
```

1.7 การกระโดดไปที่ต่างๆ โดยการคำนวนในขณะโปรแกรมทำงาน

บางครั้งการเลือกกระโดดไปทำงานต่างๆ 128 อย่าง อาจไม่พอและเมื่อไม่มีโปรแกรมถูกจำกัดโดยขอบเขต 2 k หรือการกระโดดไปยังงานต่างๆ ที่ยังไม่อาจรู้ตำแหน่งในช่วงการแอสแซมเบลอร์ และไม่สามารถที่จะแก้ไขได้ด้วยการคำนวนตำแหน่ง แอดเดรสที่จะกระโดดไปในช่วงการทำงานตามโปรแกรมโดยการคำนวนทางคณิตศาสตร์หรือคำสั่งที่เกี่ยวเนื่อง กับการใช้ TABLE LOOK-UP แล้วจึงให้กระโดดโดยอ้อม ไปยังที่ตำแหน่งที่ต้องการได้ เมื่อคำนวนดัชนีการกระโดดได้แล้ว ก็จะใช้เป็นตัวหาค่าตำแหน่งจะกระโดดตามตารางแอดเดรสที่ได้จาก แอสแซมเบอร์และ PUSH ใส่ค่าในบริเวณแสตก ด้วยการให้แอดเดรสไบต์ต่ำเข้าแสตกก่อนกระโดดแล้วจึงตามด้วยแอดเดรสไบต์สูง หลังการพบคำสั่ง RET และก็จะ POP คืนค่าในแสตกที่จะกระโดดไปใส่ในตัวชี้โปรแกรม PC ซึ่งจะทำงานได้ตามตำแหน่งที่ถูกคำนวนจากโปรแกรมที่ได้ต่อไป

2.เทคนิคการต่อพ่วงกับอุปกรณ์นอกด้วยโปรแกรม

การรับส่งข้อมูลผ่าน PORT เข้า-ออกนั้น บางครั้งการส่งอาจไม่ได้ส่งครั้งละ 8 บิตเสมอไป ดังนั้นถ้ามีการส่งค่าข้อมูลที่ไม่ใช่ขนาด 8 บิต เราสามารถที่จะใช้โปรแกรมช่วยในการตรวจสอบการรับส่งได้เป็นขนาดต่างๆ แล้วแต่ข้อกำหนด สมมติให้มีการส่งข้อมูลมาแลทช์ที่ PORT OUTPUT ขนาด 5 บิต 3 PORT โดยกำหนดเป็น PORT X, Y, Z PORT เหล่านี้เป็น PORT ชื่อใหม่จะเป็น PORT สมมติในขณะที่ PORT ในทาง HARD WARE ของ IC ตระกูลนี้ จะมี PORT 1 และ 2 รามได้ถึง 16 บิต

เราสามารถที่จะกำหนดใหม่ให้มีรูปแบบดังรูป โดยจะเอาขา P2.7 ซึ่งอาจจะเป็นบิตทดสอบหรือขาควบคุมส่งออกด้วยการบังคับจากซอฟต์แวร์ มีข้อหน้าสังเกตุว่า เรากำหนดให้บิต PORT Z กลับกันกับการลำดับตามขาของ PORT 8031 ที่เป็นเช่นนี้อาจจะเกิดจาก การวางตัว 8031 ที่มีขา PORT ที่ต่อออกไปเข้าที่ห้าต่อต้องสลับกัน ซึ่งถ้าเกิดกรณีที่ไม่เรียงลำดับเช่นนี้ ก็สามารถที่จะ PORGRAM ได้เช่นกัน

PORT 'Z'					PORT 'Y'					PORT 'X'				
PZ0	PZ1	PZ2	PZ3	PZ4	PY4	PY3	PY2	PY1	PY0	PX4	PX3	PX2	PX1	PX0
P27	P26	P25	P24	P23	P22	P21	P20	P17	P16	P15	P14	P13	P12	P11	P10

รูปการแบ่ง PORT X, Y, Z จาก PORT ของ 8031

ลักษณะการโปรแกรมจะเห็นว่าการเขียนไปที่ PORT สมมุติประการหนึ่งนั้นจะไม่มีผลต่อการเปลี่ยนแปลงกับ PORT สมมุติตัวอื่นๆเลย โดยที่ตัวโปรแกรมย่อยนี้จะแบ่งโปรแกรมย่อยออกเป็น 3 โปรแกรมด้วยกันสำหรับการถูกเรียกจากโปรแกรมหลัก ในการที่จะเขียนข้อมูลจากแอกคิวมิวเลเตอร์ไปยัง PORT สมมุติ เฉพาะแต่ละตัวของ OUT_PX,OUT_PY, OUT_PZ ซึ่งแต่ละโปรแกรมก็จะถูกกำหนดใหม่ เพื่อใช้งานได้ตามกำหนดทาง HARDWARE ที่สมมุติขึ้นมา

2.2 การปรับปรุงการทำงาน PORT ให้ดีกว่าเดิมจากตัวอย่างโปรแกรม 2.1 จะเห็นว่ามีการเรียกใช้ PORT สมมุติ 2 PORT ซึ่งจะเรียกใช้ PORT จริงซ้ำกันอยู่พอร์ตหนึ่ง คือ PORT X และ PORT Y ในลำดับการทำงานของ PROGRAM ถ้าเกิดว่ามีการเรียกใช้ PORT X ใน PROGRAM หลัก ขณะที่มีการเรียก PORT Y ในการบริการ INTERUP จะเห็นว่า PROGRAM เก่าจะใช้คำสั่ง MOV P1,A ซึ่งเป็นพอร์ตหนึ่งที่ถูกเรียกซ้ำกันจากการเรียกพอร์ตสมมุติ ทั้ง 2 พอร์ต ในช่วงเวลาใกล้เคียงกันก็จะทำให้ข้อมูลที่ถูกเขียนเข้าไปที่รีจิสเตอร์ ที่พอร์ตในตอนแรก จะถูกเขียนทับเข้าไปในการเรียกใช้พอร์ตที่ 2 ของลำดับการทำงานช่วงต่อมา ทำให้พอร์ตสมมุติพอร์ตแรกถูกเปลี่ยนค่าได้ ดังนั้นถ้ามีการเรียกใช้โปรแกรม ในลักษณะเช่นนี้จะมีวิธีป้องกันไม่ให้ข้อมูล OUTPUT ของพอร์ตสมมุติแต่ละบิตหายไปได้ การใช้เทคนิคของการ Read-modify-Write feature มาใช้คือคำสั่ง ANL. AND ที่เอาท์พุต ดังตัวอย่าง

```
                PX-MAP      DATA  22H
                PY-MAP      DATA  21H
                PZ-MAP      DATA  22H

OUT_PX:         ANL   P1,#11100000B :CLEAR BITS P1.4-P1.0
                ORL   P1,A          ;SET P1 PIN FOR EACH ACC BIT SET
                RET
;               ...   .....
;               ...   .....
OUT_PY          MOV   B#20H
                MUL   AB            ;SHIFT B A LEFT 5 BITS.
                ANL   P1,#00011111B ;CLEAR PY FIELD OF PORT1
                ORL   P1,A          ;SET PY PITS ON PORT1
                MOV   A,B           ;LOAD 2 BITS SHIFTED INTO B
```

```
                ANL   P2.#11111100B ;AND UPDATE P2
                ORL   P2,A
                RET
;               ...   .....
;               ...   .....
OUT_PZ          RRC   A             ;MOVE ORIGINAL ACC.0 INTO CY
                MOV   P2.6,C        ;AND STORE TO PIN P2.6
                RRC   A             ;MOVE ORIGINAL ACC.1 INTO CY
                MOV   P2.5,C        ;AND STORE TO PIN P2.5
                RRC   A             ;MOVE ORIGINAL ACC.2 INTO CY
                MOV   P2.4,C        ;AND STORE TO PIN P2.4
                RRC   A             ;MOVE ORIGINAL ACC.3 INTO CY
                MOV   P2.3,C        ;AND STORE TO PIN P2.3
                RRC   A             ;MOVE ORIGINAL ACC.4 INTO CY
                MOV   P2.2,C        ;AND STORE TO PIN P2.2
                RET
```

2.3 โปรแกรมหน่วงเวลา

การใช้งาน 8031 ส่วนใหญ่ต้องการควบคุม OUTPUT ช่วงเวลาหนึ่ง ซึ่งการโปรแกรมหน่วยงเวลา สามารถที่จะสร้างสัญญาณ STROBE OUTPUT ออกมาได้ เช่นต้องการช่วงกว้าง 50 μsec. การใช้ $ ตามตัวอย่างเป็นตัวอักษรพิเศษที่หมายถึงแอดเดรสของคำสั่งนี้ ซึ่งเป็นการขจัดการใช้LABEL แทนตำแหน่งของคำสั่งเช่นคำสั่งของ DJNZ จะวนทำงานอยู่ในคำสั่งจนกว่า R2 จะถูกลดลงเป็น 0 เป็นการทำงานขา WR มีสถานะต่ำอยู่ช่วงเวลา 50 μsec.

```
                ...   ..
                CLR   WR
                MOV   R2,#24
                DJNZ  R2,$
                SETB  WR
                ...   ..
```

2.4 การเริ่มต้นโปรแกรมพอร์ตอนุกรมและตัวจับเวลา

เมื่อมีการกำหนดให้ใช้งานในลักษระรับส่งข้อมูลแบบอนุกรมจำเป็นจะต้องมีการเริ่มต้นโปรแกรม การใช้งานลักษณะนี้ รวมทั้งโปรแกรมรับข้อมูล เข้ามาเก็บและส่งข้อมูลผ่านพอร์ตข้อมูลอนุกรม

ตามตัวอย่างที่ให้มานี้เป็นการกำหนด 8031 ใช้พอร์ตอนุกรมส่งข้อมูลด้วยอัตราความเร็ว 2400 บ๊อด โดยโปรโตคอลแต่ละข้อมูลที่รับส่ง จะเป็นข้อมูลขนาด 7 บิต odd parity และ stop บิต 1 บิต ดังนั้นส่งข้อมูลได้ 2400 บ๊อด ประมาณเท่ากับ 265 ตัวอักษรต่อวินาที

โดยการกำหนดเริ่มแรก การรับและส่งข้อมูลอนุกรมนั้นจะต้องเริ่มเซตให้อยู่ใน 8-bit UART mode คือ SM0, SM1=01 enable ให้สามารถรับข้อมูลทั้งหมดด้วย SM2=0 และ REN=1 ส่วนแฟลก์ของการส่งจะเป็นอิสระ เพื่อให้ตัวซอฟแวร์รู้ถึงตัวรีจิสเตอร์ output สามารถเรียกใช้ได้ ทั้งหมดที่กล่าวมานี้จะโปรแกรมอยู่บนบรรทัด SPINIT

ตัวจับเวลา 1 จะถูกโปรแกรมเป็นแบบ AUTO-RELOAD เพื่อให้ได้ 2400 band ตามกำหนดเช่น สัญญาณนาฬิกาภายใน 1 MHz. ตามสูตร

$$1*10^{6} / 32*2400 = 13$$

ดังนั้น ตัวจับเวลาหนึ่งจะถูกโหลดค่าย่า -13 หรือ 0F3H เข้า TH1 ตามตัวอย่างการเริ่มโปรแกรม

```
:
:               INITIALIZE SERIAL PORT
;               FOR 8-BIT UART MODE
;               & SET TRANSMIT READY FLAG
SPINT:          MOV             SCON,#01010010B
;               INITIALIZE TIMER 1 FOR
;               AUTO-RELOAD AT 32x2400Hz.
;               (TO USED AS GATED 16-BIT COUNTER.)
;TINIT:         MOV   TCON,#1101001B
                MOV   TH1,#13
                SETB  TR1
;               ...     ......
```

2.5การโปรแกรมการรับส่งข้อมูลผ่านพอร์ตอนุกรม

SP_OUT จะเป็นโปรแกรมย่อยแบบธรรมดา ที่ส่งข้อมูลจากแอกคิวมิวเลเตอร์ โดยเริ่มแรกจะต้องคำนวนหาบิต PARITY แล้วจึงแทรกข้อมูลเข้าไปในไบต์ที่ 7 รอจนกว่าตัวส่งจะยอมให้ส่งด้วยการตรวจสอบบิต TI เมื่อบิต TI "1" ก็จะให้ เริ่มส่งข้อมูลนั้นได้

```
;
;SP_OUT      ADD ODD PARITY TO ACC AND
;           TRANSMIT WHEN SERIAL PORRT READY
;
SP_OUT      :    MOV   C.P
             CPL   C
             MOV   ACC.7.C
             JNB   TI.S
             CLR   TI
             MOV   SBUF.A
             RET
```

SP_IN จะเป็นโปรแกรมที่กลับกันคือรอตอบว่า BUFFER ได้รับข้อมูลแล้วหรือยังด้วยการตรวจสอบบิต RI ถ้าบิต RI เซตก็จะรับข้อมูลเข้ามาจะมีการเซตแฟลกหด ถ้าเกิดข้อผิดพลาดทาง odd-parity และคืนค่าที่ถูก maask 7 บิตกลับสู่แอกคิวมิวเลเตอร์

```
;
;SP_IN       INPUT NEXT CHARCTER FROM SERIAL PORT.
;           SET CARRY IF ODD-PARAITY ERROR
SP_IN       JNB   RI.$
             CLR   RI
             MOV   A,SBUF
             MOV   C,P
             CPL   C
             AL    A,#7FH
             RET
```

2.6 การส่งข้อความตัวอักษรผ่านพอร์ตอนุกรม

การใช่งานส่งตัวอักษรผ่านพอร์ตอนุกรมได้ในรูปแบบ ASCII มักจะส่งข้อมูลเป็นข้อความ ซึ่งอาจจะเป็นข้อความบอกความผิดพลาด การทำงานกำลังอยู่ในสถานะใด และข้อความการวิเคราะห์ การทำงานของระบบ ซึ่งข้อความเหล่านี้ สามารถที่จะเขียนเป็นส่วนหนึ่งของโปรแกรมได้และสามารถใช้คำสั่งเทียมของคำสั่ง DEFINE BYTE (DB) เป็นตัวกำหนดข้อความ ดังตัวอย่างการเขียนโปรแกรมส่งผ่านพอร์ตอนุกรมนี้

```
CR          EQI   0DH          ;ASCII CARRIAGE RET
LF          EQU   0AH          ;ASCII LINE-FEED
ESC         EQU   1BH          ;ASCII ESCAPE CODE
;           ...   ....
            CALL  XSTRNG
            DB    CR,LF        ;NEW LINE
            DB    'INTEL DELIVERS'   ;MASSAGE
            DB    ESC          ;ESCAPE CHARCTER
;
;       (CONTINUATION OF PROGRAM)
;
;           ...   .....
XSTRING:    POP   DPH          ;LOAD DPTR WITH FIRST CHARACTER
            POP   DPL
XSTR_1:     CLR   A            ;(ZERO OFFSET)
            MOVC  A,@A+DPTR    ;FETCH FIRST CHARACTER OF STRING
XSTR_2:     JNB   TI,$         ;WAIT  UNTIL TRANSMITITER READY
            CLR   TI           ;MARK AS NOT READY
            MOV   SBUF,A       ;OUTPUT NEXT CHARACTER
            INC   DPTR         ;BUMP POINTER
            CLR   A
            MOVC  A,@A+DPTR    ;GET NEXT OUTPUT CHARACTER
            CJNE  A,#ESC,XSTR_2;LOOP UNTIL ESCAPE READ
```

```
        MOV   A,#1
        JMP   @A+DPTR      :RETURN TO CODE AFTER ESCAPE
```

2.7การวิเคราะห์และทำตามข้อแม้ต่างๆ ที่เกิดขึ้น

ก่อนที่จะทำงานตามข้อมูลที่ได้รับ ตัวโปรแกรมย่อยก็จะหาข้อกำหนดที่แน่นอนที่ต้องการให้ทำ โดยการเปรียบเทียบกับค่าที่เข้ามา เช่นในการส่งตัวอักษรต่างๆ จากการ EDIT ข้อความต่างๆ และส่งไปยังเครื่องพิมพ์นั้น โปรแกรมมาตราฐาน ก็จะต้องมีการส่งตัวอักษรตามรหัสที่แน่นอน นอกจากนั้นยังต้องมีการส่งคำควบคุมเครื่องพิมพ์ไปด้วย เช่น [DEL], [CR], [LF], [BEL], [ESC] หรือ [SP] ซึ่งเมื่อรับรหัสของค่าเหล่านี้ ก้จะหมายถึงการบังคับให้เครื่องตามความหมายของคำนั้นด้วยโปรแกรมย่อย แต่ละรูปแบบ งานของคำนั้น ดังนั้นค่ารหัสอักษรที่มีค่าต่ำกว่า 20H ก็จะถูกแปลงเป็นค่า [NUL] และถูกเปรียบเทียบให้ทำงานตามคุณสมบัติของเครื่องพิมพ์ ซึ่งก็จะเขียนโปรแกรมเพื่อให้กระโดดไปตามรหัสที่ต้องการเครื่องพิมพ์ด้วยคำสั่ง CJNE

```
;
CHAR        EQU   R7              ;CHARACTER CODE VARIABLE
:
INTERP:     CJNE  CHAR,#7FH,INTP_1   :SKIP UNLESS RUBOUT
;           ..    .....              (SPECIAL ROUTINE FOR RUOUT
CODE)
            RET
INTP_1      CJNE  CHAR,#07H,INTP_2   ;SKIP UNLESS BELL
;           ..    .....              (SPECIAL ROUTINE FOR BELL CODE)
            RET
INTP_2      CJNE  CHAR,#0AH,INTP_3   ;SKIP ULESS LFEED
;           ...   .....               (SPECIAL ROUTINE FOR RETURN
CODE
            RET
INTP_3      CJNE  CHAR,#0DH,INTP_4   ;SKIP UNLESS ESCAPE
;           ...   ......              (SPECIAL ROUTINE FOR ESCAPE
CODE)
```

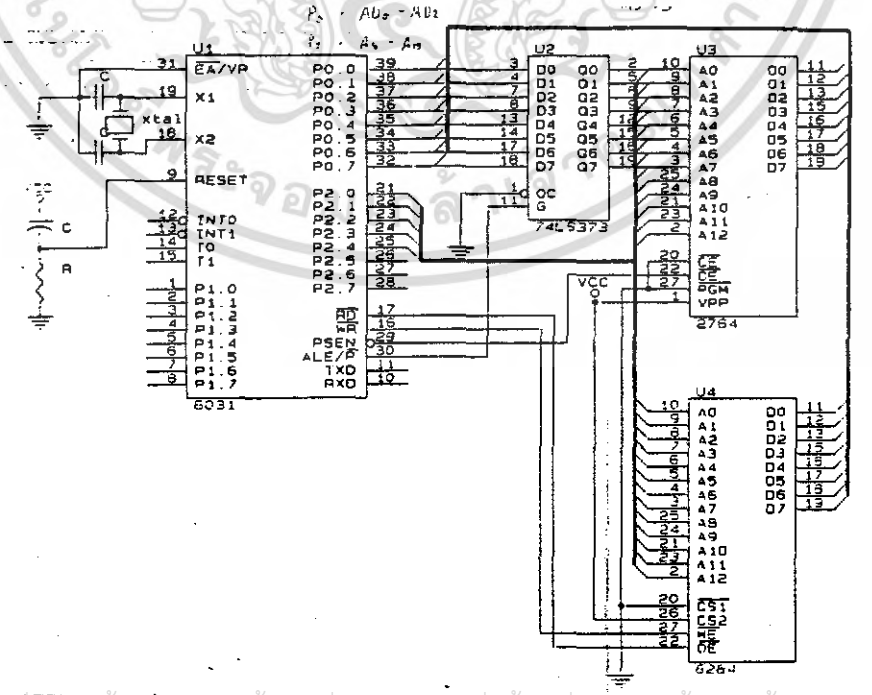
```
                RET
INTP_5          CJNE    CHAR,#20H,INTP_6    ;SKIP UNLESS SPACE
;               ...     . . . . . .         (SPECIAL ROUTINE FOR SPACE CODE)
                RET
INTP_6          JC      PRINTC              ;JUMP IF CODE 20H
                MOV     CHAR.#0             ;REPLACE CONTROL CHARACTER
WITH
                                            ;NULL CODE
PRINTC:                                     ;PROCESS STNADARD PRINTING

;               ...     . . . . .           ;CHARACTER
                RET
:
;               ...     . . . . .
                CLR     EA                  ;DISABLE ALL INTERRUPTS
                CLR     TR1                 ;STOP TIMER 1
                MOV     A,#LOW (-1000-7)    ;LOAD LOW-ORDER DESIRED COUNT
                ADD     A,TL1               ;CORRECT FOR TIMER OVERRUN
                MOV     TL1,A               ;RELOAD LOW-ORDER BYTE .
                MOV     A,#HIGH (-1000+7)   ;REPEAT FOR HIGH -ORDER BYTE.
                ADDC    A,TH1
                MOV     TH1,A
                SETB    TR1                 ;RESTART TIMER
                ...     . . . . .
RDTIME:         MOV     A,TH0               ;SAMPLE TIMER0 (HIGH)
                MOV     R0,TL0              ;SAMPLE TIMER0 (LOW)
                CJNE    A,TH0,RDTIME        ;REPEAT IF NECESSARY
                MOV     R1,A                ;STORE VALID READ
                RET
```

3.ตัวอย่างวงจรการต่อพ่วงกับอุปกรณ์ภายนอก

3.1 วงจรการใช้งานร่วมกับ EPROM และ STATIC RAM ภายนอก

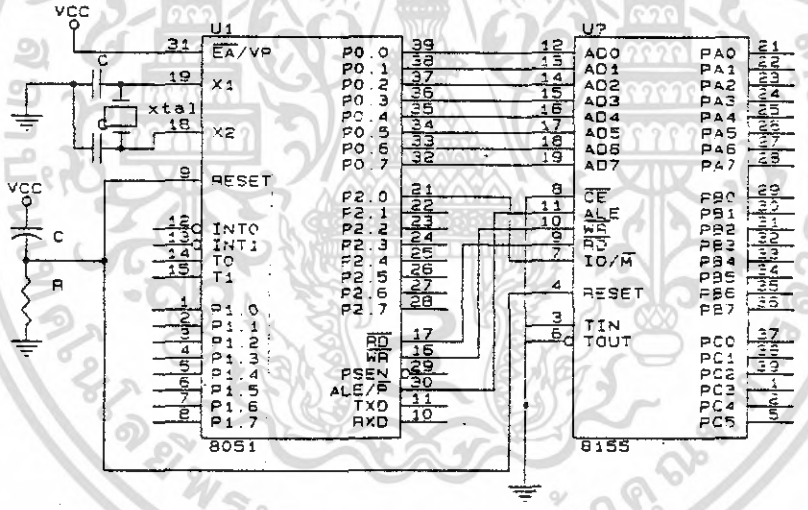
การพัฒนาโปรแกรมเพื่อให้เครื่องต้นแบบที่ใช้ไมโครคอนโทรลเลอร์ ทำงานได้ตามข้อกำหนดต่างๆ อย่างไม่ผิดพลาด ส่วนใหญ่จะพัฒนาโปรแกรมด้วยการให้ตัวไมโคร 8031 ตัวนี้ทำงานตามโปรแกรมที่เก็บอยู่ข้างนอก เช่น จะใช้ตัว EPOM แต่ระหว่างการพัฒนาจะใช้ RAM BACK UP ที่มีแบตเตอร์รี่ ที่จ่ายไฟสำรอง หรือ EPROM EMULATOR ที่รับส่งข้อมูลอนุกรมกับ IBM PC ก็ได้ เพื่อความสะดวกในการอ่านและเขียนแก้ไขโปรแกรม ลักษณะการต่อวงจรที่ให้ทำงาน ตามโปรแกรมที่อยู่ภายนอก แสดงดังรูปพร้อมการต่อวงจร STATIC RAM (6264) ขนาด 8Kx8 BYTE ภายนอก เพื่อใช้ในการเข้าถึงข้อมูล โดยที่วงจรจะบังคับให้ขา EA มีสถานะต่ำ เพื่อเป็นการบอกให้ไมโครทำงานโปรแกรมภายนอก ซึ่งอยู่ EPROM ขนาด 8Kx8 BYTE ตัวแลทช์ 74LS373 จะทำการแลทช์ข้อมูลแอดเดรสไว้เนื่องจากขา DB0-DB7 เป็นแบบมัลติเพลซ์ข้อมูลและแอดเรส ดังนั้นจึงต้องแลทช์แยกเอาค่าแอดเดรสมาแสดงที่ขา output ของตัว 74LS373 ไปถอดรหัสโปรแกรมในตัว EPROM (2764) ขนาด 8Kx8 BYTE เพื่อให้ตัวไมโครคอนโทรลเลอร์สามารถทำงานตามโปรแกรมภายนอกได้อย่างถูกต้อง ข้อสังเกตุเมื่อเราใช้โปรแกรมจากภายนอกไมโครจะทำให้ตัวไมโครมีขาพอร์ตที่ลดน้อยไปประมาณ 16 ขา คือขา DB0-DB7 และขา P20-P27 ส่วนการเข้าถึงข้อมูล RAM จะใช้ขา RD WR เป็นขาควบคุม



รูปวงจรไมรโครคอนโทรลเลอร์ที่ใช้ EPROM ภายนอก

3.2 วงจรการขยายพอร์ตและวิธีการต่อ RS232

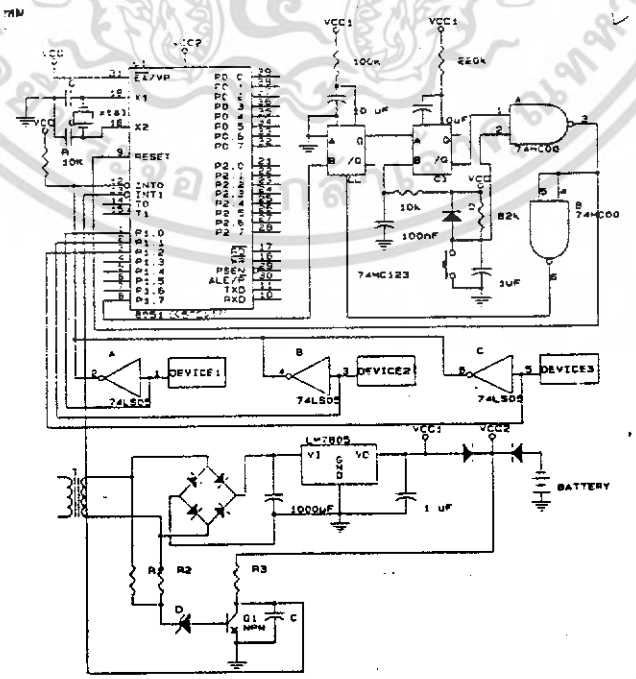
SHIP #8255 ทำงานเป็นพอร์ตได้ 3 พอร์ต โดยไม่มี ROM หรือ RAM ลักษณะการต่อ จากรูป เนื่องจาก ขาแอดเดรสของ 8255 คือ A0 และ A1 ไม่สามารถจัดการกับการมัลติเพลกซ์แอดเดรสกับข้อมูลเองได้ จึงต้องต่อกับขาเอาว์พุตของตัวแลทช์ 74LS373 ที่ใช้ในการแอดเดรสข้อมูล โปรแกรม EPROM ภายนอกให้ได้ค่าการถอดรหัสแอดเดรส A0, A1 ได้ 4 ตำหน่ง เพื่อใช้ในการส่งข้อมูลออกและเข้าพอร์ตทั้ง3ของ 8255 และอีกหนึ่งแอดเดรสที่เหลือจะใช้ในการจัดการส่งคำควบคุมลักษณะการใช้งานในการเริ่มต้นโปรกรมเข้าที่ SHIP 8255 นี้



รูปการต่อวงจรที่ใช้ SHIP# 8255 และการต่อ RS232

3.3 วงจรการใช้งานการอินเตอร์รัพต์ชนิดต่างๆและการรีเซต

เมื่อมีการร้องขอการอินเตอร์รัพต์รัพต์จากการต่อใช้วงจรอินเตอร์รัพต์หลายแหล่งตัว 8031 สามารถที่จะจัดการการตอบรับการอินเตอร์รัพต์ของแหล่งจขต่างๆ ได้จากการถอดรหัสของพอร์ตหนึ่งเพื่อให้โปรแกรมทำงาน ตามแหล่งของการร้องขออินเตอร์รัพต์ได้ถูกต้องนอกจากการใช้ขา INT0 ในการตรวจจับการอินเตอร์รัพต์แล้ว ตัว 8031 ยังสามารถที่จะตรวจจับ แหล่งจ่ายไฟที่จ่ายให้กับ 8031 เมื่อเกิดกรณีที่ไฟฟ้าดับ และตัว 8031 ต้องการที่จะเก็บข้อมูลไว้เพื่อที่จะทำงานต่อไปได้ เมื่อไฟฟ้าจ่ายให้ใหม่อีก ลักษณะเช่นนี้จะใช้ INT1 จากวงจรทรานซิสเตอร์ตรวจจับแหล่งจ่ายไฟทุกครึ่ง cycle โดยที่โปรแกรมบริการอินเตอร์รัพต์ ตัวที่จะคอย RELOAD ค่าค่าหนึ่ง เข้าสู่ตัวจับเวลาทุกครั้งที่โปรแกรมนี้ทำงาน แต่ถ้าไม่มีไฟจ่าย ก็จะไม่สร้างสัญญาณ INT1 ดังนั้นตัวจับเวลาจะเกิด OVERFLOW โปรแกรมการของบริการอินเตอร์ ตัวจับเวลาจะทำงาน ซึ่งจะเป็นการเก็บค่าทุกค่าของรีจิสเตอร์ต่างๆ ใน CPU ให้หมดก่อนที่ไฟจะดับถึงขั้นที่ตัว 8031 จะทำงานไม่ได้อีกต่อไป เมื่อมีการจ่ายไฟมาอีก วงจรการรีเซตอัตโนมัติ ที่ใช้ตัว IC 74HC123 จะสร้างสัญญาณรีเซตจุดให้เริ่มทำงานจากค่าเดิมต่างๆต่อไปได้อีก ในขณะการทำงานปกติ ถ้าเกิดมีสัญญาณรบกวนเข้ามาทำให้ 8031 หยุดทำงาน หรือเกิดมีอาการ HANG ขึ้นมาเฉยๆ วงจรที่ใช้ IC 74HC123 นี้จะทำหน้าที่เป็น WATCH DOG สร้างสัญญาณรีเซตให้เครื่องทำงานต่อไปได้ ด้วยการตรวจสอบขา P1.7 ที่ส่งสัญญาณพัลส์มาตลอดเวลาที่ 8031 ทำงานปกติ แต่ถ้า 8031 หยุดทำงานสัญญาณที่ P1.7 ก็จะขาดหายไป ทำงานวงจร WATCH DOG ชุดนี้สร้างสัญญาณรีเซตขึ้นมาใหม่



รูปการต่อวงจรใช้งานด้านอินเตอร์รัพต์ต่างๆและ WATCH DOG

MSM 6242BRS/ GS-VK/ JS

DIRECT BUS CONNECTED CMOS REAL TIME CLOCK/CALENDAR

หลักการทั่วไป

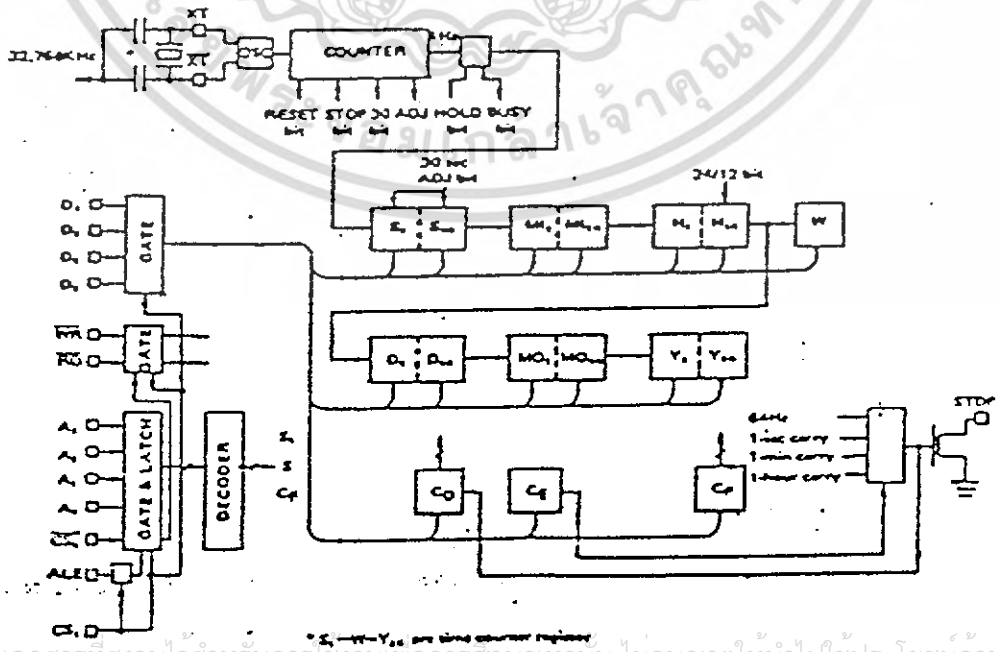
MSM 6242B เป็น CMOS แบบ Silicon gate ของ Real Time Clock/ Calendar สำหรับใช้ในการติดต่อเชื่อมโยงโดยตรงกับไมโครโปรเซสเซอร์เพื่อประโยชน์ในการใช้งานของไมโครคอมพิวเตอร์ โดยมีตัว chip 32.768 KHz คริสตัล ออสซิสเลเตอร์ ซึ่งเป็นฐานเวลาและเป็นตัวจัดแอดเดรสข้อมูลขนาด 4 bit ทางด้านอินพุทและเอาพ์พุท จัดเวลาสำหรับข้อมูลเป็นวินาที นาที ชั่วโมง วันของสัปดาห์ วันที่ เดือน และปี ข้อมูลขาเข้าจะถูกควบคุมโดยแอดเดรสขนาด 4 bit ซึ่งมี Chip Select ของ (CS0, CS1), WRITE, READ และ ALE เป็นตัวควบคุม นอกจากนี้ยังมีคอนโทรล Registor D.E. และ F เป็นตัวปรับเวลาให้ผิดพลาดได้ +- 30 วินาทีโดยมี INTERRUPT REQUEST (IRQ FLAG) และ BUSY status bit เป็นตัวกำหนด clock ของ STOP bit. HOLD bit และ RESET bit ทั้ง 4 ตัวเลือกของ INTERRUPT นี้จะเป็นอัตราส่วนที่หาได้จาก STD.P (STANDARD PULSE) เป็น OUTPUT ที่ใช้คอนโทรล Registor ทางด้านอินพุท เป็น T_0, T_1 และ ITRPT/STND (INTERRUPT/ STANDARD) การ INTERRUPT OUTPUT (STD.P) สามารถกระทำโดยทาง MASK bit MSM6242B นี้สามารถที่จะทำให้เกิดเป็นเวลา 12 หรือ 24 ชั่วโมง หรือ เป็นเวลาแบบ Leap Year อัตโนมัติก็ได้

ปกติเบอร์ MSM 6242B จะoperateได้ที่แรงไฟ 5V. +-10% ที่อุณหภูมิ -30°cถึง 85°c ส่วน Battery back up จะใช้เมื่อแรงไฟต่ำกว่า 2 volt ทั้งนี้เพื่อรักษาเวลาให้คงไว้เมื่อแรงไฟจาก MAIN ดับลง MSM 6242B ที่กล่าวถึงมีขนาด 18-pin Plastic DIP,24 pin FLAT, 18 pin PLCC

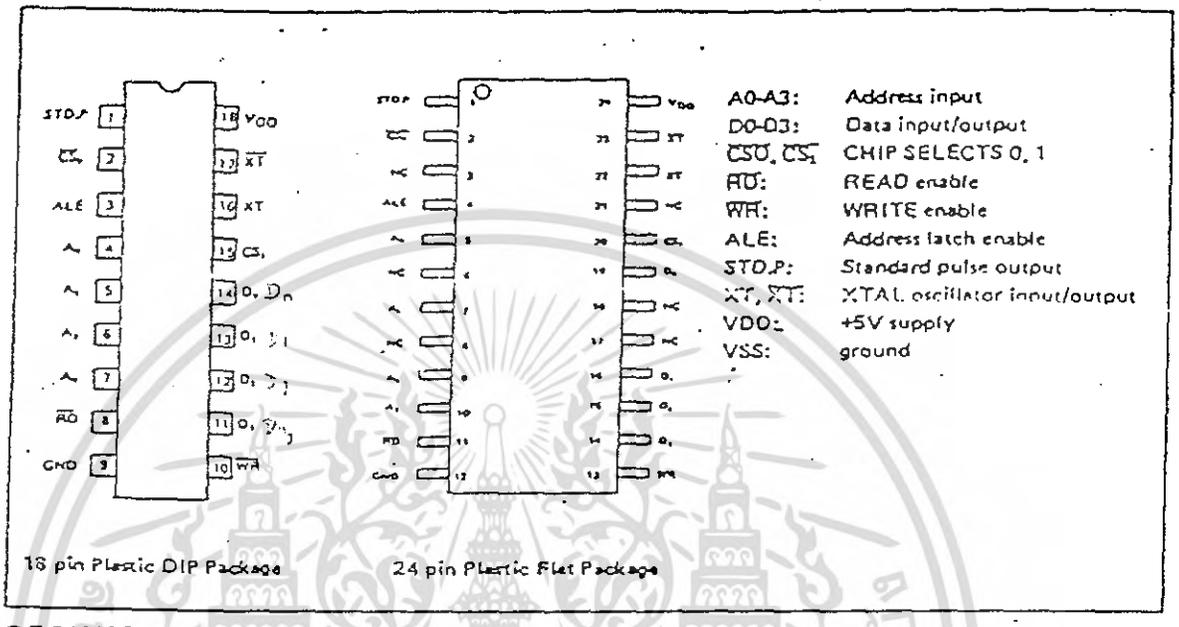
ลักษณะสำคัญแสดงดังต่อไปนี้

เวลา	เดือน	วันที่	ปี	วันของสัปดาห์
23:59:59	12	31	80	7

- 4 bit data bus
- 4 bit address bus
- READ,WRITE,ALE and CHIP SELECT INPUTS
- Status registers-IRQ and BUSY
- Selecable interrupt outputs- 1/64 second, 1 second, 1 minute, 1 hour
- Interrupt masking
- 32.768 KHz crystal controlled operation
- 12/24 hour format
- Auto leap year
- -- 30 second error correction
- Single 5V supply
- Battery backup down to V_{DD}-2.0V
- Low power dissipation : 20/w max at V_{DD} - 2V 150/w max at V_{DD} - 5V
- 18 pin plastic DIP, 24 pin FLAT and 18-pin PLCC



รูปร่างลักษณะขาของ MSM6242B



REGISTER TABLE

Address Input	Address Input				Register Name	Data				Count value	Description
	A_3	A_2	A_1	A_0		D_3	D_2	D_1	D_0		
0	0	0	0	0	S_1	S_4	S_4	S_2	S_1	0 ~ 9	1-second digit register
1	0	0	0	1	S_{10}	*	S_{40}	S_{20}	S_{10}	0 ~ 5	10-second digit register
2	0	0	1	0	MI_1	mi_8	mi_4	mi_2	mi_1	0 ~ 9	1-minute digit register
3	0	0	1	1	MI_{10}	*	mi_{40}	mi_{20}	mi_{10}	0 ~ 5	10-minute digit register
4	0	1	0	0	H_1	h_8	h_4	h_2	h_1	0 ~ 9	1-hour digit register
5	0	1	0	1	H_{10}	*	PM/AM	h_{20}	h_{10}	0 ~ 2 or 0 ~ 1	PM/AM, 10-hour digit register
6	0	1	1	0	D_1	d_8	d_4	d_2	d_1	0 ~ 9	1-day digit register
7	0	1	1	1	D_{10}	*	*	d_{20}	d_{10}	0 ~ 3	10-day digit register
8	1	0	0	0	MO_1	mo_8	mo_4	mo_2	mo_1	0 ~ 9	1-month digit register
9	1	0	0	1	MO_{10}	*	*	*	MO_{10}	0 ~ 1	10-month digit register
A	1	0	1	0	Y_1	Y_8	Y_4	Y_2	Y_1	0 ~ 9	1-year digit register
B	1	0	1	1	Y_{10}	Y_{80}	Y_{40}	Y_{20}	Y_{10}	0 ~ 9	10-year digit register
C	1	1	0	0	W	*	w_4	w_2	w_1	0 ~ 6	Week register
D	1	1	0	1	C_D	30 sec. ADJ	IRQ FLAG	BUSY	HOLD	—	Control Register D
E	1	1	1	0	C_E	t_1	t_0	ITRPT /STND	MASK	—	Control Register E
F	1	1	1	1	C_F	TEST	24/12	STOP	REST	—	Control Register F

REST = RESET

ITRPT/STND = INTERRUPT/STANDARD

Note 1) — Bit * does not exist (unrecognized during a write and held at "0" during a read).

Note 2) — Be sure to mask the AM/PM bit when processing 10's of hour's data.

Note 3) — BUSY bit is read only. The IRQ FLAG bit can only be set to a "0". Setting the IRQ FLAG to a "1" is done by hardware.

หมายเหตุ

1. Bit * ไม่ได้ใช้ (ไม่รู้จักระยะเวลาในการเขียนและยึดหลักในการอ่านเป็น "0")
2. Mask AM/PM bit เมื่อประมวลผลข้อมูลได้ 10's หรือของ HOUR
3. BUSY bit คือ การอ่านอย่างเดียว IRQ FLAG bit สามารถ set เป็น "0" ได้อย่างเดียว ซึ่งในการ set IRQ FLAG เป็น "1" ไม่มีใน hardware

ลักษณะฟังก์ชั่นของ Register ต่าง ๆ คือ

S_1, S_{10}, MI_1, MI_{10}, H_1, H_{10}, D_1, D_{10}, MO_1, MO_{10}, Y_1,Y_{10}, W

a) ฟังก์ชั่นทั้งหมดนี้เป็นคำย่อของ SECOND 1, SECOND 10, MINUTE 1, MINUTE 10, HOUR1 HOUR 10, DAY 1, DAY 10, MONTH 1, MONTH 10, YEAR 1, YEAR 10, และ WEEK ซึ่งค่าทั้งหมดนี้ คือ ค่าของ BCD

b) รีจิสเตอร์ทุกตัวเป็น Logic บวกทั้งหมด ยกตัวอย่างเช่น (S_8, S_4, S_2, S_1) = 1001 ซึ่งมีค่าเท่ากับ 9 Sec

c) ถ้าข้อมูลที่เขียนซึ่งเป็นข้อจำกัดของข้อมูลที่ออกจาก Colck Register มีข้อผิดพลาดเกิดขึ้นก็จะอ่าน Read ข้อมูลในภายหลังได้

d) PM/AM bit (HOUR)

ในการ set mod เลือก 24 หรือ 12 Hour Mode PM/AM bit นับถึง 24 หรือ 12 ตัว มันเองจะไม่รู้ถึงเวลาขณะนั้น คือเมื่ออ่านครบ mode ที่เลือกที่จะอ่านเป็น 24 หรือ 12 "HOUR" กลับเริ่ม start bit ใหม่หรือ bit เริ่มต้น คือ "0" และอ่านตามลำดับ bitไปเรื่อย ๆ จาก 0 ก็เป็น 1.2.....

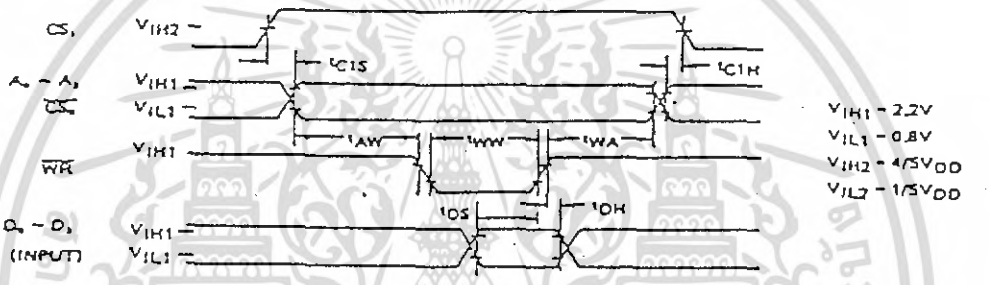
e) MSM 6242B นี้จากทศสมัยคริสเตียน ได้ออกแบบ ให้ทำงานเป็น Register Y_1, Y_{10} และ Leap Year โดยอัตโนมัติ ได้ผลที่ได้จากการ set ทุกครั้งดังกล่าวไม่เกี่ยวกับ การ set วัน (day) และเดือน (month) จึงแสดงตัวอย่างเช่น

ถ้าข้อมูลที่เขียนเป็นวันที่ 29 เดือนกุมภาพันธ์ ปี 1985 หรือวันที่ 31 พฤศจิกายน ปี 1985 เมื่อเวลาล่วงไปตัว MSM 6242B จะเปลี่ยนวัน เดือน ปี โดยอัตโนมัติ คือ เป็นวันที่ 1 มีนาคม ปี 1985 หรือวันที่ 1 เดือนธันวาคม ปี 1985 ตามลำดับ ข้อมูลที่ได้จะละเอียดและเที่ยงตรง ซึ่งจะนำเป็น PULSE เกิดขึ้นสำหรับตัวเลขต้าใด ๆ ของวันด้วย

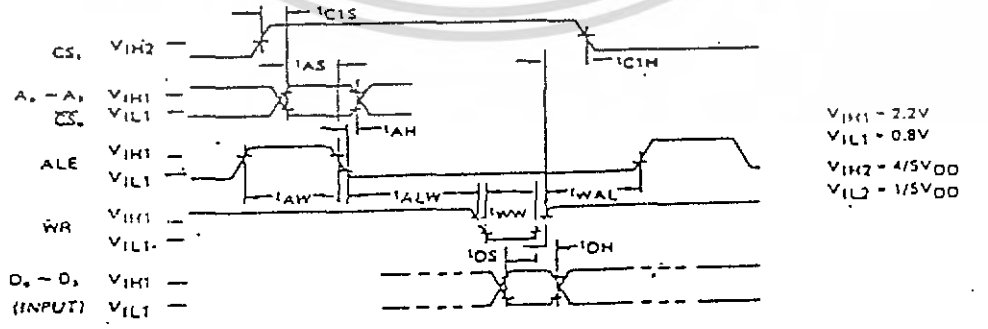
f) Register W จะเป็นข้อมูลที่บอกถึงเลข 0-6 ดังแสดงในตารางและบอกความหมายของวันได้อย่างชัดเจน

SWITCHING CHARACTERISTICS

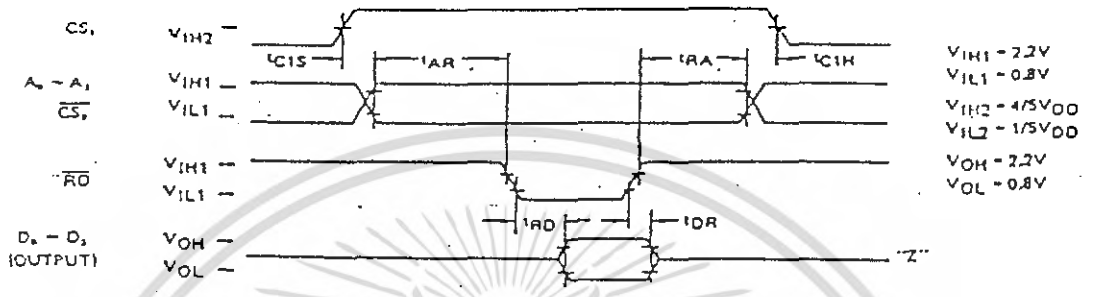
1. WRITE mode (ALE = V_{DD}) ; (V_{DD} = 5V+/- 10% = Ta = -30~ +85°C)



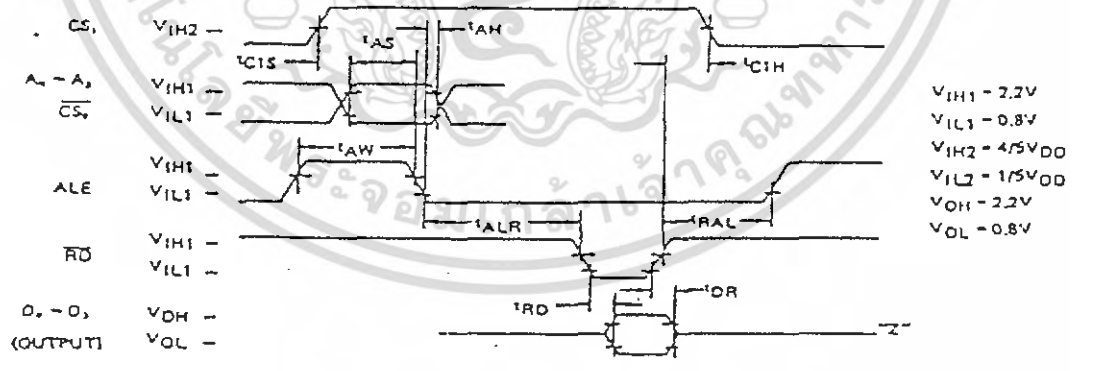
2.WRITE mode (WITH use of ALE) ; (V_{DD} = 5V ± 10% Ta = -3°C)



3. READ mode (V_{DD} = ALE) ; (V_{DD} =5V $\pm$ 10% . Ta = -30 ~ +85°C)



4. READ mode (WITH use of ALE) ; (V_{DD} = 5V $\pm$ 10% . Ta = -30 ~ +85°C)



Note : อันนี้เป็นการทำตามเวลาที่กำหนดโดยการรวมวันระหว่าง t_1 และ to :

t_1 = 0 และ to = 1..... น้อยกว่า 1 Sec (วินาที)

t_1 = 1 และ to = 0..... น้อยกว่า 1 minute (นาที)

t_1 = 1 และ to = 1..... น้อยกว่า 1 hour (ชั่วโมง)

: 12 HOUR MODE

24 HOUR MODE

CD REGISTER (CONTROL D REGISTER)

a) HOLD (DO)

ตั้งค่านี้เป็น "1" เพื่อยับยั้งหรือหยุดสัญญาณ clock 1 Hz โดย S_1 Control ซึ่งเวลา BUSY= 0 . Register S_1 ~ W สามารถที่จะอ่านหรือเขียน ในระหว่างนี้ carry ที่เกิดที่ S_1 counter จะเพิ่มเป็น 1 Sec หลังจาก HOLD = 0 (สภาวะนี้ถึง HOLD = 1 ก็จะไม่มีการเพิ่มมากเกินกว่า 1 Sec) ถ้า CS_1 = 0 เมื่อนั้น HOLD = 0 โดยไม่คำนึงสภาพเงื่อนไขอื่น ๆ

b) BUSY (D_1)

status bit ซึ่งแสดงถึงสภาพการ interface ด้วย microcontroiler/microprocessor ในส่วนของวิธีการเขียนภายในและอ่านจาก S_1 ~ W (address 0~C) อ้างถึงตามรูป Flow Chart ดังแสดงในรูปที่ 10

c) IRQ FLAG (D_2)

status bit อันนี้จะมีหน้าที่เดียวกันกับ output level ของ STD.P output เมื่อ STD.P = 0 และ IRQ = 1; เมื่อ STD.P = 1 ดังนั้น IRQ = 0

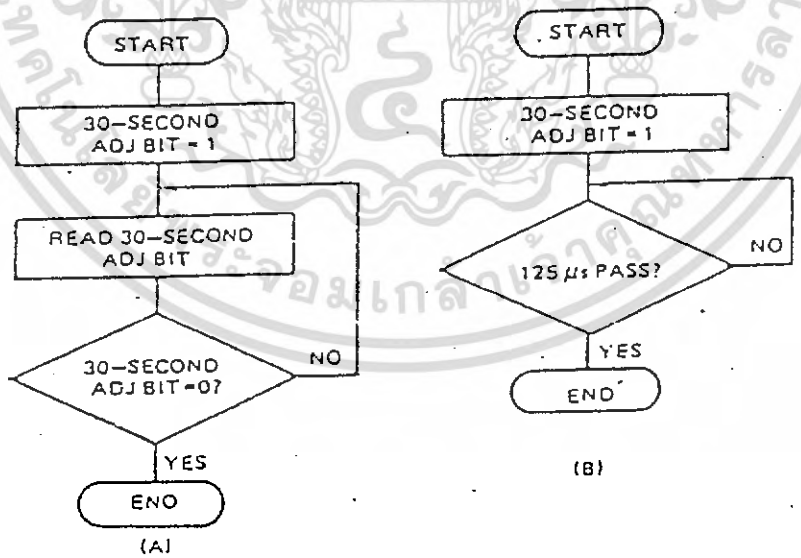
IRQ FLAG จะแสดงการ interrupt โดยจะเกิดขึ้นกับ microcomputer ถ้า IRQ = 1 เมื่อมีการทำ Register C_E (MASK) = 0, ดังนั้น STD.P output จะเปลี่ยนแปลงไปตามให้คล้องกับเวลาที่ set โดย D_3 (t_1) และ D_2 (to) ของ Register E เมื่อ D_1 ของ Register E (ITRPT/STND) = 1 (interrupt mode) STD.P output ก็ยังคงอยู่ในระดับ LOW จนกระทั่ง IRQ FLAG เขียนได้เป็น "0"

เมื่อIRQ = 1 และเวลาสำหรับการinterruptใหม่ก็จะเกิดขึ้นเมื่อITRPT/STND = 0 (Standard output Mode)outputของ STD.P ก็ยังอยู่ในสภาพ low จนกระทั่งเป็น"0"เป็น การเขียน IRQ FLAG อีกประการหนึ่ง IRQ FLAG จะเป็น "0" โดยอัตโนมัติของตัวมันเองภาย หลังตาม 7.8125 ms

เมื่อเขียน HOLD เป็น 30 Sec ปรับ bit Register D..IRQ FLAG bit ที่เขียนจะเป็น "1"

d) +- 30 ADJ (D_3)

เป็นการเขียน bit D_3 ให้เป็น "1" เพื่อรักษาระดับเวลาที่ +- 30 Sec ตลอดเวลาซึ่งเวลารีจิสเตอร์(clock Register) ไม่ควรจะอ่านหรือได้จากการเขียนเป็น 125 Sec หลัง bit D_3 = 1 ตัว bit นี้จะกลับสู่สภาวะ "0" โดยอัตโนมัติ



CE REGISTER (CONTROL E REGISTER)

a) MASK (DO)

บิทนี้จะควบคุม output ของ STD.P เมื่อ MASK = 1 ดังนั้น STD.P = 1 (open) เมื่อ MASK = 0 ดังนั้น STD.P = output mode

ความเกี่ยวข้องระหว่าง MASK BIT และ STD.P output แสดงในรูป

b) INTRPT/STND (D_1)

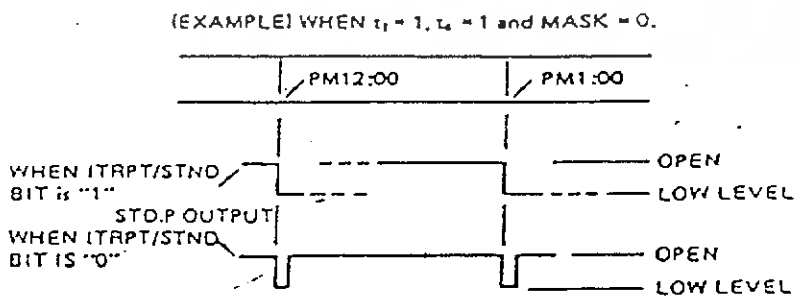
INTRPT/STND input จะใช้เป็น switch ของ STD.P output ระหว่าง MODE ทั้งสองทำให้เกิด interrupt และ waveform เวลามาตรฐานเมื่อ INTRPT/STND=0 WAVE-FORM ที่ได้ cycle จะคงที่และมีระดับสัญญาณเป็น LOW โดย Pulse มีความกว้าง 7.8125 ms เป็นการแสดงคล้ายของ STD.P output ในเวลานี้ MASK bit จะมีค่า = 0 ในขณะที่คาบเวลาทั้ง 2 mode กำหนดโดย TO (D_2) และ T_1 (D_3) ของ Register F

c) TO (D2), T1 (D3)

bit ทั้งสองเป็นตัวกำหนดคาบเวลาของ STD.P output ใน interrupt ทั้งสองและเวลาคงที่ตาม waveform mode ดังตารางแสดงค่าเวลาที่มีส่วนร่วมกับ TO input ของ T1 จะมีลักษณะเช่นเดียวกันหรือเกี่ยวข้องกันกับ INTRPT/STND และ STD.P

Timing ของ STD.P output จะระบุโดย T1 และ To เป็นตัวกำหนดเวลานั่นคือกำหนด carry เป็น clock digit (ตัวเลขตัวใดตัวหนึ่งใน 0-9) ตัวอย่างเช่น

เมื่อ ti = 1 และ MASK = 0



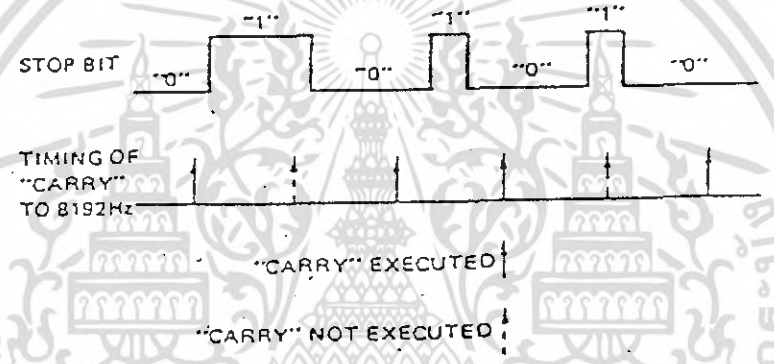
d) LOW LEVEL PULSE ของ (ITRPT/STAN) waveform จะมีความกว้างในช่วง cycle ที่คงที่คือ 7.8125 ms ซึ่งไม่เกี่ยวกับ TO/T1 input

e) Fixed Cycle ของ waveform mode สามารถใช้ปรับความถี่ของความถี่ oscillatorได้

f) ระยะเวลา +- 30 Sec จะเกิดการปรับค่า carry ซึ่งจะเป็นเหตุให้ STD.P output เป็น LOW เมื่อ TO/T1 = 1.0 หรือ 1.1 อย่างไรก็ตามเมื่อ T1/TO = 0.0 และ ITRPT/STND = 0 carry จะไม่เกิดขึ้นและ STD.P output จะเกิดกลับคืนสู่สภาพเดิม

g) STD.P output คือเป็นจุด STOP = 1 ขณะเวลา ITRPT/STND = 0

h) STD.P outputไม่มีการเปลี่ยนแปลงเป็นผลมาจากการเขียนข้อมูลของ Register S1~H1



(c) 24/12 (D2)

บิทนี้เป็น bit ที่เลือก mode tune ของ 24 HOUR หรือ 12 HOUR

ถ้า D2 = 1-24 hour ดังนั้น PM/AM bit ก็จะไม่ใช้

ถ้า D2 = 0-12 hour ดังนั้น PM/AM bit ก็จะใช้

"24 HOUR / 12 HOUR" การ set bit กระทำได้ดังต่อไปนี้

1. REST bit = 1

2. 24/12 hour bit = 0 หรือ 1

3. REST bit = 0

REST bit จะต้อง = 1 ถึงจะเขียนเป็น 24/12 hour bit ได้

(d) TEST (D3)

เมื่อ TEST FLAG เป็น "1" input ที่นับเป็น SECOND จะมาจาก counter/divider stage นาแทนที่ของ divider stage ที่ 15 แล้วทำให้ SECOND counter นับที่ความถี่ 5.4163 KHz มา แทน 1Hz เมื่อ TEST = 1 (Test Mode) ที่ STOP & REST (Reset) flag จะไม่หยุด หรือยับยั้งการนับเวลาภายใน flag เมื่อ HOLD = 1 ระยะเวลาของ (TEST = 1) TEST ภายในตัวนับก็จะหยุดนับ อย่างไรก็ ตามเมื่อ HOLD FLAG ถ้าอยู่เฉยๆ หรือ HOLD = 0 จะไม่รับในการ update

CH_1 (Chip Select)

V_{IH} และ V_{IL} ของ CH_1 มี 3 function

a) มีการเชื่อมโยงติดต่อกับ microcontroller/microprocessor

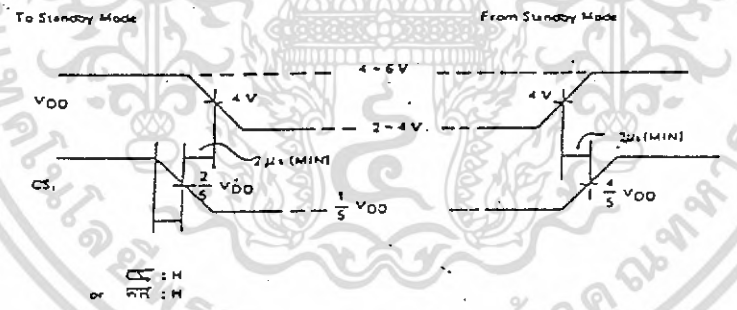
b) หยุดยับยั้ง control bus, data bus, address bus และลด input gate ผ่าน current ใน stand by mode

c) ป้องกันข้อมูลภายในเมื่อ mode เคลื่อนย้ายมาจาก stand by mode นอกเหนือจาก function ที่กล่าวอีกคือ

a) VDD ที่ค่ามากกว่า 4/5 VOLT ควรจะดัดแปลงตัว MSM 6242B สำหรับนำไปเชื่อมโยง ใช้งานกับ microcontroller/microprocessor ที่ใช้ไฟ 5 V. ได้

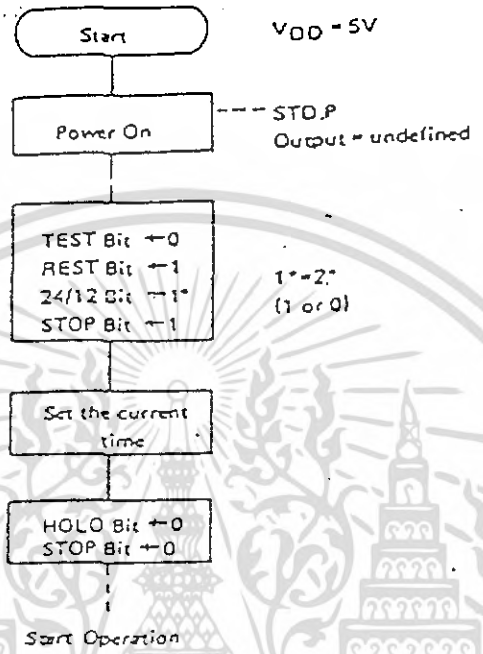
b) ในการเคลื่อนย้าย stand by mode 1/5 VDD ควรจะดัดแปลงการใช้เหมือนกันเพื่อใช้กับ Data bus ทั้งหมด ซึ่งใน stanby mode ควรจะดัดแปลงให้ได้ประมาณ 0 V.

c) ที่ stand by mode ควรจะปฏิบัติตามคำสั่งของ Timing chart



การนำไปใช้งาน

Power Supply

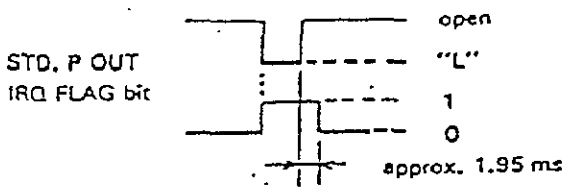


ลักษณะเพิ่มเติม

เมื่อทำการเขียน IRQ FLAG bit เป็น "0" ตัวมันเองจะ clear bit อย่างไรก็ตามถ้า "0" เป็นตัวกำหนดถึง IRQ FLAG bit เมื่อทำการเขียนถึง bit อื่น ๆ เช่น 30 Sec ADJ bit และ HOLD bit ควรจะทำ IRQ FLAG = 1 ซึ่งก่อนการเขียน IRQ FLAG = 1 ต้อง clear หรือลบล้างโดยกำหนดให้ IRQ FLAG ให้เป็น "0" เสียก่อนแล้วค่อย SET ตัวมันใหม่เป็น "1"

เมื่อ IRQ FLAG bit กลับกลายมาเป็น "1" ในบางกรณีเมื่อเขียนใหม่ซ้ำของ $t_1.t_0$ หรือ ITRPT/STND bit ของ Register C_E แน่นอนจะต้องเขียนเป็น "0" หลังจากเขียนแล้ว IRQ FLAG ก็จะใช้เท่ากับ 1

ความเกี่ยวดองระหว่าง STD.P OUT และ IRQ FLAG bit ดังแสดงในรูป ความถี่ที่เกี่ยาข้องระหว่าง SDT.P OUT และ IRQ FLAG BIT ดังแสดงในรูป



DOT MATRIX LCD MODULE

ปัจจุบัน LCD เป็นที่นิยมกันอย่างมาก สำหรับการแสดงผลในเครื่องมือ เครื่องใช้ต่างๆ ขณะนี้ผู้ผลิต LCD MODULE ออกมา คือเป็น MODULE ที่มีตัว LCD และวงจรควบคุมมาให้พร้อม (เรียกว่า LCM) ซึ่งทำให้ผู้ใช้สามารถต่อเข้ากับบระบบไมโครได้ง่ายและสะดวกสำหรับการเขียนโปรแกรม

LCD MODULE มีอยู่มากมายหลายรุ่น และมีคุณสมบัติแตกต่างกันไป ในที่นี้จะกล่าวถึงการใช้งานแบบ DOT MATRIX ซึ่งมีราคาถูก และเพียงพอต่องานส่วนใหญ่

<u>คุณสมบัติของ DOT MATRIX LCD MODULE มีดังนี้</u>

1. มีให้เลือกหลายรุ่นตามการใช้งาน โดยมีจำนวนตัวอักษรและบรรทัดแตกต่างกันไป
2. ตัวอักษรแสดงด้วย DOT MATRIX ขนาด 5*8 DOT
3. สามารถต่อเข้ากับระบบไมโครได้ 2 ลักษณะ คือแบบ MEMORY MAP และแบบผ่าน 8255 PORT
4. การใช้งานง่ายและสะดวก
5. มีคำสั่งพิเศษสำหรับอำนวยความสะดวกมากมาย
6. สามารถแสดงผลเป็นตัวอักษรภาษาอังกฤษและตัวเลขได้ 160 ตัว และสัญญลักษณ์พิเศษ อีก 32 ตัวรวมทั้งสามารถกำหนดอักษรที่ออกแบบเองได้อีก 8 ตัว
7. กินกระแสไฟน้อย และมีน้ำหนักเบา ทำงานด้วยไฟเลี้ยงระดับ 5 V. เท่านั้น

ขาสัญญาณของ LCD MODULE

PIN	SYMBLO	LEVEL	FUNCTION
1	Vss	---	0 V END
2	Vcc	---	+ 5V POWER SUPPLY
3	Vee	---	- V FOR LIQUID CRYSTAL DRIVE
4	Rs	H/L	REGISTER SELECT H:DATA INPUT L:INSTRUCTION INPUT
5	R/W	H/L	H:DATA READ L:DATA WRITE
6	E	H	ENABLE SIGNAL (LH)
7	DB 0	H/L	DATA BUS BIT 0
8	DB 1	H/L	DATA BUS BIT 1
9	DB 2	H/L	DATA BUS BIT 2
10	DB 3	H/L	DATA BUS BIT 3
11	DB 4	H/L	DATA BUS BIT 4
12	DB 5	H/L	DATA BUS BIT 5
13	DB 6	H/L	DATA BUS BIT 6
14	DB 7	H/L	DATA BUS BIT 7

การต่อเข้ากับระบบไมโคร

LCD MODULE จะต่อเข้ากับระบบไมโครได้ 2 ลักษณะ คือแบบ MEMORY MAP โดยผ่าน LCD BUS ขนาด 20 PIN และแบบ I/O PORT โดยผ่าน 8255 BUS ขนาด 26 PIN โดยแต่ละแบบจะมีหลักการดังนี้

การต่อแบบ MEMORY MMAP

1. สามารถต่อเข้ากับ CHIP เบอร์ทั่ว ๆ ไปได้ เช่น 8051 หรือ Z80 โดยจะทำให้ระบบไมโครมองเห็น LCD MODULE ในลักษณะของ MEMORY ได้ทันที

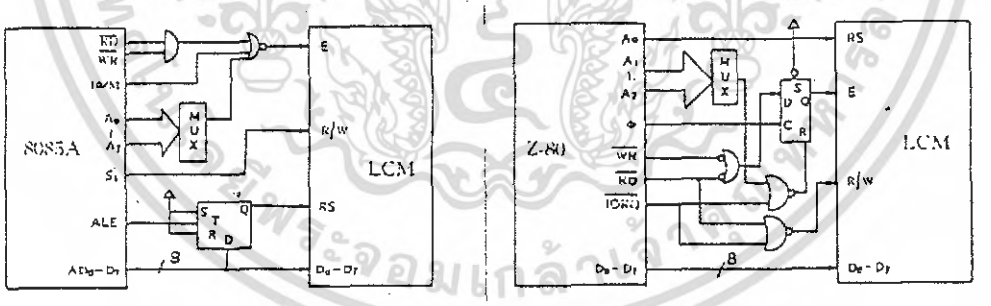
2. ผู้ใช้สามารถเห็นและอ่านข้อมูลจาก LCD MODULE ได้ทำให้มองเสมือนว่าเป็น MEMORY BUFFER ไปในตัว

3. เนื่องจากสามารถอ่านข้อมูลกลับได้ จึงทำให้สามารถตรวจสอบ FLAG ความพร้อมในขณะที่ LCD MOFULR กำลังทำงานได้

4. ใช้ได้กับบอร์ดที่มี LCD BUS มาให้พร้อมเท่านั้น

5. ทำให้กินพื้นที่ของหน่วยความจำไปส่วนหนึ่ง และต้องมีการ DECODE ละเอียดพอสมควร

6. การจัดขาสัญญาณจะต้องเป็นไปตามแบบของ CHIP แต่ละเบอร์ค่าย



การต่อแบบ I/O PORT

1. สามารถต่อเข้ากับ I/O PORT ใด ๆ ก็ได้ โดยใช้สายสัญญาณจำนวน 11 เส้น และใช้โปรแกรมเป็นตัวสร้างสัญญาณขึ้นมาให้ตรงกับข้อกำหนดของ LCD MODULE

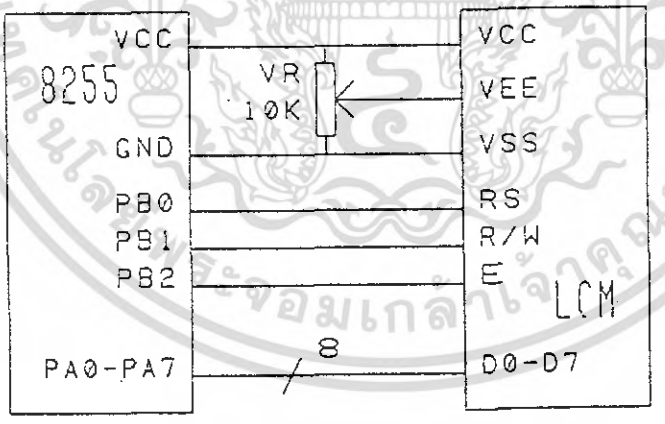
2. ผู้ใช้จะเขียนข้อมูลให้ LCD MODULE ได้อย่างเดียว ซึ่งผู้ใช้ความกำหนด MEMORYส่วนหนึ่งให้เป็นเสมือน BUFFER ให้กับ LCD MODULE อีกที

3. เนื่องจากไม่สามารถอ่านข้อมูลกลับได้ จึงต้องใช้การหน่วงเวลาของระบบไมโครเองเพื่อรอให้ LCD MODULE การทำงานการต่าง ๆ

4. ใช้ได้กับบอร์ดทั่ว ๆ ไปที่มี PORT

5. ไม่เปลืองส่วนของ MEMORY ในการใช้งาน

6. การจัดขาสัญญาณกระทำได้อย่างอิสระ



ชุดคำสั่งควบคุมและการแสดงข้อความ

การเขียนหรืออ่านข้อมูลกับ LCD MODULE ก็คือการกำหนดคุณสมบัติต่าง ๆ ในการใช้งานของ LCD ตามชุดคำสั่งควบคุมและรวมถึงการเขียนข้อมูลที่เป็นข้อความ เพื่อให้ปรากฏบนแผงแสดงด้วย โดยมีรายละเอียดตามตาราง

INSTRUCTION	RS	R/W	DATA BIT 7	6	5	4	3	2	1	0	EXE. TIME ( )
CLEAR DISPLAY	0	0	0	0	0	0	0	0	0	1	1640
CURSOR AT HOME	0	0	0	0	0	0	0	0	1	*	1640
ENTRY MODE SET	0	0	0	0	0	0	0	1	I/D	S	40
DISPLAY ON/OFF	0	0	0	0	0	0	1	D	C	B	40
DISPLAY SHIFT	0	0	0	0	0	1	S/C	R/L	*	*	40
FUNCTION SET	0	0	0	0	1	DL	N	F	*	*	40
SET CGRAM ADD.	0	0	0	1	CGRAM ADDRESS						40
SET DDRAM ADD.	0	0	1	DDRAM ADDRESS							40
BUSY,ADD. READ	0	1	BF	ADDRESS							0
CGRAM,DDRAM WR	1	0	WRITE DATA								40
CGRAM,DDRAM RD	1	1	READ DATA								40

ความเข้าใจพื้นฐาน

1. การเขียนข้อมูลให้กับ LCD MODULE จะแบ่งเป็น 2 ลักษณะคือ INSTRUCTION และ DATA โดยจะกำหนดด้วยขาสัญญาณ RS คือถ้า RS = 0 จะหมายถึงส่งสัญญาณควบคุม และถ้า RS = 1 จะหมายถึงการเขียนหรือการอ่าน DATA

2. หลักการในการเขียนข้อมูลให้ LCD MODULE นี้ คือเมื่อมีการเขียนข้อมูลไปแล้ว ตัว LCD MODULE จะต้องใช้เวลาในการทำงานชั่วขณะหนึ่ง (ตามค่า EXECUTE TIME ในตาราง) จึงจะสามารถเขียนข้อมูลต่อไปได้

3. การเขียนข้อมูลให้ LCD MODULE นี้ทำได้ทั้งแบบ 8 BIT และ 4 BIT โดยกรณี 4 BIT การเขียนข้อมูลจะทำเหมือนกับ 8 BIT เพียงแต่ให้เขียน 2 ครั้ง คือ DB4-DB7 ก่อน แล้วตามด้วย DB0-DB3 และจะต้องกำหนดคุณสมบัติตามค่า D6 ในคำสั่ง FUNCTION-SET ด้วย

4.DDRAM (DISPLAY DATA RAM) คือหน่วยความจำภายในตัว LCD MODULE ที่เป็น BUFFER ข้อมูลโดยถ้าเขียนรหัส ASCII ใด ๆ ลงไปในหน่วยความจำนี้ ก็จะปรากฏเป็นตัวอักษรที่แท้ แสดงทันที

5. GRAM (CHARACTER GENERATOR RAM) คือหน่วยความจำภายในตัว LCD MODULE สำหรับเก็บภาพตัวอักษรที่ผู้ใช้สามารถสร้างโดยได้เอง (8 ตัว) โดยจะอ้าง ADDRESS ได้ทั้งหมด 64 BYTE คือ 8 ตัวอักษรคูณกับ 8 ROW

แนวทางการเขียนโปรแกรมควบคุม

1. เมื่อจ่ายไฟเลี้ยงให้กับ LCD MODULE ครั้งแรก ภายในจะมีการ RESET ระบบโดยอัตโนมัติ ซึ่งจะใช้เวลา 10 ms หลังจากที่ระดับแรงไฟขึ้นถึง 4.5 V. แล้ว ทั้งนี้ระบบ RESET ดัง กล่าวจะกระทำสิ่งต่าง ๆ ดังต่อไปนี้

- ทำการ CLEAR จอภาพทั้งหมด (CLEAR DISPLAY)

- กำหนดคุณสมบัติด้วยคำสั่ง FUNCTION SET คือ DL = 1 (ติดต่อกับระบบไมโครในแบบ 8 bit),N = 0 (แสดงข้อมูล 1 บรรทัด) F = 0 (กำหนดอักษรแบบ 5*7 DOTS)

- กำหนดคุณสมบัติด้วยคำสั่ง DISPLAY ON/OFF คือ D=0 (ไม่แสดงข้อมูล), C=0 (CURSOR OFF), B=0 (BLINK OFF)

- กำหนดคุณสมบัติด้วยคำสั่ง ENTRY MODE SET คือ I/D = 1 (INCREMENT), S=0 (NO SHIFT)

การใช้งาน LCD MODULE ต้องรอให้ขบวนการ RESET ภายในทำงานเรียบร้อยก่อน ซึ่งจะตรวจสอบได้ด้วย BF (BUSY FLAG) หรืออาจจะใช้การหน่วงเวลาได้

2. การใช้งาน LCD MODULE จะเกี่ยวข้องกับการทางด้านโปรแกรมเป็นส่วนใหญ่ ชุดคำสั่งต่างๆ รวมทั้งการอ่านหรือการเขียนข้อมูลนั้นจะถูกกำหนดด้วยขาสัญญาณทั้งหมดที่มี อายุปกติโปรแกรมจะต้องกำหนดคุณสมบัติต่าง ๆ ที่ต้องการไว้ที่ส่วนต้น และจากนั้นก็จะเป็นการอ่านและเขียนข้อมูลลงใน DDRAM ซึ่งก็คือข้อความที่จะให้แสดงนั่นเอง

<u>รายละเอียดแต่ละคำสั่ง</u>

1. CLEAR DISPLAY

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	0	0

สำหรับการ CLEAR DISPLAY โดยจะทำการเขียนตัวอักษร SPACE ลงใน DDRAM ทั้งหมดและกำหนดค่า DDRAM ADDRESS ให้เป็น 0 พร้อมทั้ง CURSOR จะกลับไปที่ตำแหน่งซ้ายบนสุดของจอภาพ

2. CURSOR AT HOME

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	0	1	*

สำหรับกำหนดค่า DDRAM ADDRESS ให้เป็น 0 พร้อมทั้ง CURSOR จะไปสู่ตำแหน่งซ้ายสุดของจอภาพ โดยที่ข้อมูลใน DDRAM ไม่มีการเปลี่ยนแปลง

3. ENTRY MODE SET

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	0	0	1	I/D	S

I/D = 0 กำหนดทิศทางของ CURSOR และDDRAM ให้เป็นแบบ DECREMENT

I/D = 1กำหนดทิศทางของ CURSOR และ DDRAM ให้เป็นแบบ INCREMENT

S = 0 เมื่อเขียนข้อมูลแล้ว ตัว CURSOR จะถูกเปลี่ยนไปทิศทางตามค่า I/D

S = 1 เมื่อเขียนข้อมูลแล้ว ตัว CURSOR จะอยู่กับที่ และตัวอักษรจะถูกดันไปทิศทางตาม ค่า I/D

การกำหนด I/D และ S นี้ให้กำหนดก่อนการเขียนข้อมูลใน DDRAM และเมื่อกำหนดแล้วจะต้องไม่ใช้คำสั่ง CLEAR DISPLAY อีก

4. DISPLAY ON/OFF

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	1	1	D	C	B

D = 0 กำหนดให้ OFF DISPLAY

D = 1 กำหนดให้ ON DISPLAY

C = 0 กำหนดให้ OFF CURSOR

C = 1 กำหนดให้ ON CURSOR โดย CURSOR จะเป็นเส้นขีดใต้ตัวอักษร

B = 0 กำหนดให้ ไม่มีการกระพริบที่ตำแหน่ง CURSOR

B = 1 กำหนดให้ มีการกระพริบ (เป็นรูปสี่เหลี่ยม)

5. DISPLAY SHIFT

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	0	1	S/C	R/L	*	*

S/C = 0 กำหนดให้เลื่อน CURSOR ตามทิศทาง R/L ไป 1 ตำแหน่ง

S/C = 1 กำหนดให้เลื่อนข้อความตามทิศทาง R/L ไป 1 COLUMN (เลื่อนทุกบรรทัด)

R/L = 0 กำหนดให้มีทิศทางไปทางซ้าย

R/L = 1 กำหนดให้มีทิศทางไปทางขวา

6.FUNCTION SET

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	0	1	DL	N	F	*	*

DL = 0 กำหนดให้การติดต่อกับ LCD MODULE เป็นแบบ 4 BIT

DL = 0 กำหนดให้การติดต่อกับ LCD MODULE เป็นแบบ 8 BIT

N = 0 กำหนดจำนวนบรรทัดแบบ 1/8 DUTY และ 1/11 DUTY

N = 1 กำหนดจำนวนบรรทัดแบบ 1/6 DUTY F = 0 กำหนดให้ตัวอักษรเป็นแบบ 5*7 DOTS

F = 1 กำหนดให้ตัวอีกษรเป็นแบบ 5*10 DOTS

7. SET CGRAM ADDRESS

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	0	1	CGRAM ADDRESS					

สำหรับการกำหนด ADDRESS ของ CGRAM เมื่อได้ทำการกำหนดไว้แล้ว การอ่านและเขียน DATA ที่ต่อจากนี้จะเป็นไปตาม ADDRESS ที่กำหนดทันที

8. SET DDRAM ADDRESS

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	DDRAM ADDERESS						

การอ่านและเขียน DATA ที่ต่อจากนี้จะเป็น ADDRESS ที่กำหนดทันที

9. BUSY FLAG AND ADDRESS READ

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	1	BF	ADDRESS						

สำหรับการอ่านค่า BF (BUSY FLAG) ซึ่งบอกความพร้อมของ LCD MODULE ในการรับข้อมูล ถ้า BF = 0 หมายถึงว่าพร้อมที่จะรับข้อมูลไปได้ แต่ถ้า BF = 1 หมายถึงว่ายังไม่พร้อม

<u>การอ่านและเขียนข้อมูลกับ DDRAM/CGRAM</u>

1. WRITE DATA TO DDRAM OR CGRAM

RS	R/W	DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
1	0	DATA

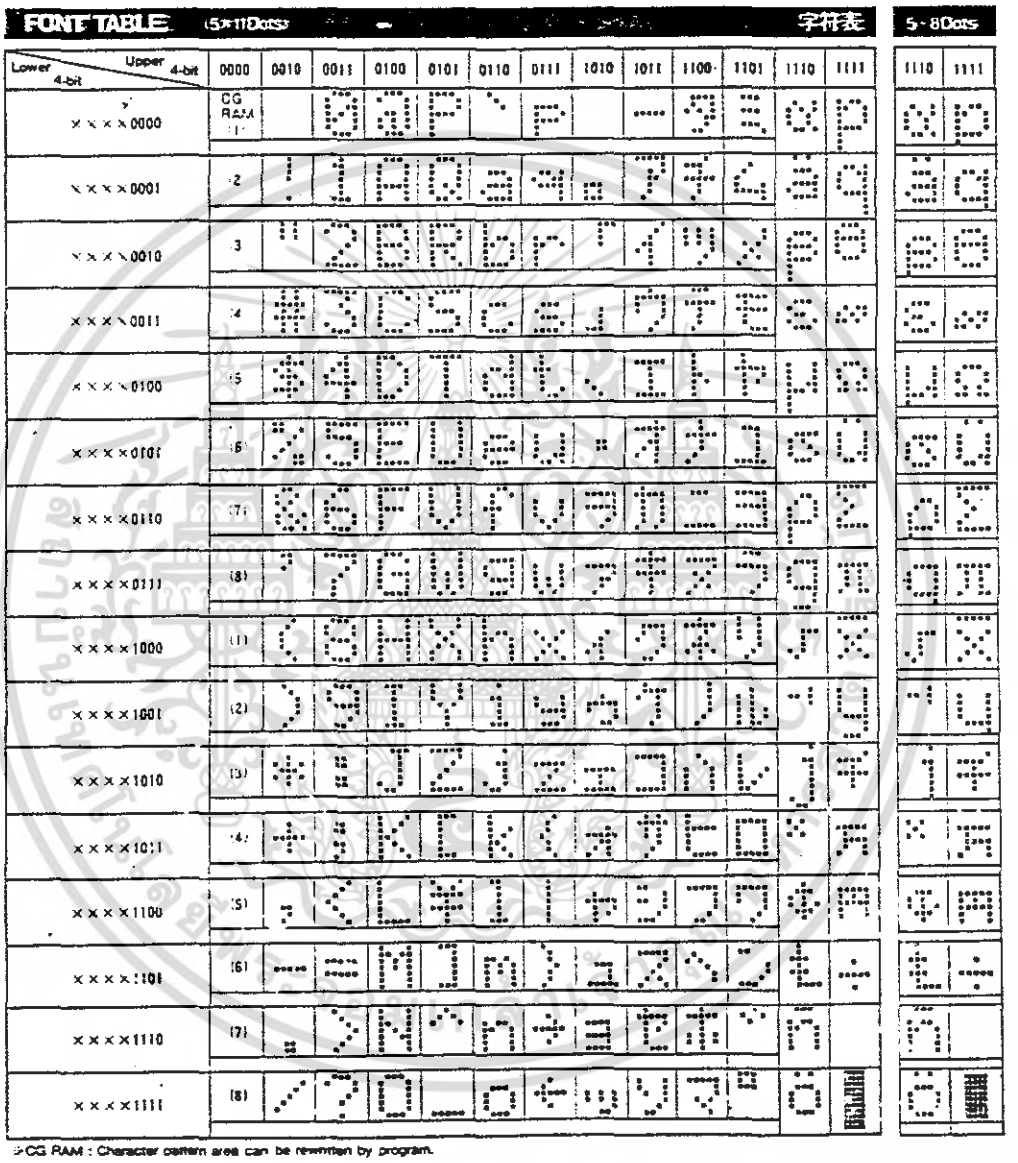
การเขียนข้อมูลลงหน่วยความจำ DDRAM หรือ CGRAM เมื่อทำการเขียนหลัง ADRESS จะถูกเพิ่มหรือลดลง โดยอัตโนมัติ ตามที่กำหนด จากค่าใน คำสั่ง ENTRY MODE SET และการเขียนจะ เป็น DDRAM หรือ CGRAM ก็ขึ้นกับว่าก่อนหน้าคำสั่งนี้ มีการกำหนด ADDRESS ที่ใด

2.READ DATA FROM DDRAM OR CGRAM

RS	R/W	DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
1	1	DATA

การอ่านข้อมูลจากหน่วยความจำ DDRAM หรือ CGRAM โดยเมื่อทำการเขียนแล้ว ADDRESS จะถูกเพิ่มขึ้นหรือลดลงโดยอัตโนมัติตามที่กำหนดจาก I/D ในคำสั่ง ENTRY MODE SET และการอ่านจะเป็น DDRAM หรือCGRAM ก็ขึ้นกับว่าก่อนหน้าคำสั่งนี้ มีการกำหนด ADDRESS ที่ใช้

ตารางอักษร



*CG RAM : Character pattern area can be rewritten by program.

8255 PROGRAMMABLE

PERIPHERAL INTERFACE

ในการนำเอาไมโครโปรเซสเซอร์ไปใช้งานนั้นจำเป็นต้องให้ ไมโครโปรเซสเซอร์สามารถติดต่อกับโลกภายนอกได้ ซึ่ง ก็คือ ให้มันสามารถส่งสัญญาณมาควบคุมอุปกรณ์ต่างๆ ได้เช่น สเต็ปปิ้งมอเตอร์ ควบคุมอุปกรณ์ไฟฟ้าต่างๆ ส่วนที่ทำได้ ไมโครโปรเซสเซอร์สามารถติดต่อกับโลกภายนอกได้ที่รู้จักกันดีคือ PORT ซึ่งมีอยู่หลายลักษณะด้วยกัน เช่น เป็นไอซี แบบไตรสเตท # 74LS244 หรือพวก แลทช์ เช่น #74LS34 สามารถต่อใช้งากับ CPU ได้โดยง่ายที่สุด โดยตัว CPU จะเป็นตัวควบคุมการอ่านเขียน PORT หากเป็นการอ่านข้อมูลจาก PORT มักจะใช้ ไอซี แบบไตรสเตท เป็น PORT INPUT โดยตัว CPU จะส่งสัญญาณไปเปิดเกทของไตรสเตทนี้ให้ข้อมูลเข้ามาสู่สายข้อมูล (DATABUS) และเข้าสู่ไมโครโปรเซสเซอร์หรือ CPU ต่อไป แต่สำหรับพอร์ทเอ้าพุทก็จะใช้แลทช์ฟลิบ-ฟลอบ ทำหน้าที่รับสัญญาณข้อมูลจากไมโครโปรเซสเซอร์มาแลทช์ไว้ที่ตัวมัน ไมโครโปรเซสเซอร์มาแลทช์ไว้ที่ตัวมัน (CPU ส่งสัญญาณมาทริก) เพื่อให้อุปกรณ์ภายนอกนั้นรับสัญญาณจากตัวแลทช์นี้ไปอีกทีหนึ่ง ที่ทำเช่นนี้เพราะตัวไมโครโปรเซสเซอร์หรือ CPU นี้ทำงานเร็วมากซึ่งในช่วงของการส่งข้อมูลออกพอร์ทจะใช้เวลาไม่กี่ไมโครเซท (us) ซึ่ง อาจทำให้อุปกรณ์ ภายนอกรับไม่ทัน

มีบริษัทต่าง ๆ เล็งเห็นความสำคัญของการติดต่อระหว่าง CPU กับอุปกรณ์ภายนอกนี้ จึงได้ทำไอซีสำเร็จรูปขึ้นหลายเบอร์ และที่จะขอนำมากล่าวในที่นี้ก็คือ ไอซีเบอร์ 8255 ซึ่งเป็นของบริษัทอินเทล ซึ่งได้ออกแบบใช้กับ CPU เบอร์ 8080 แต่เราสามารถนำมาประยุกต์ใช้กับเบอร์อื่น ๆ ได้โดยไม่ยาก

สาเหตุที่ 8255 เป็นที่นิยมมากก็เพราะว่ามันสามารถถูกโปรแกรมให้ทำงานในลักษณะต่างๆ ไม่ว่าจะเป็นอินพุท เอ้าพุท หรือแม้แต่แต่แบบ แฮนด์เชคกิ้ง (Handshaking) ได้ทั้งยังราคาถูก

ลักษณะทั่วไปของ 8255

เป็นไอซีขนาด 40 ขา ตัวแบน โดยแยกเป็นลักษณะของบล็อกง่าย ๆ ดังรูปที่ 1 คือจะมีพอร์ทให้ใช้งานได้ถึง 3 พอร์ท (เป็นขนาด 8 บิท) พอร์ท A, พอร์ท B, พอร์ท C โดยพอร์ท C นี้สามารถแยกได้เป็น 2 ส่วนคือ พอร์ท C บนตั้งแต่ PC_4 -PC_7 จำนวน 4 บิทและพอร์ท C ล่างตั้งแต่ PC_0-PC_3 โดยพอร์ททุกพอร์ท (A,B,C) สามารถโปรแกรมได้ให้เป็นอินพุทหรือเอ้าพุท ซึ่งจะได้กล่าวถึงการโปรแกรมในรายละเอียดต่อไป

- กลุ่มควบคุมชุด A จะควบคุมพอร์ท A และพอร์ท C บน
- กลุ่มควบคุมชุด B จะควบคุมพอร์ท B และพอร์ท C ล่าง
- กลุ่มควบคุมโลจิกการเขียนอ่าน

การทำงานของ 8255 จะใช้สัญญาณควบคุมจากตัวไมโครโปรเซสเซอร์มาควบคุมโดยจะมีการส่งคำสั่ง (Control word) มาที่กลุ่มควบคุมชุด A,B แล้วกลุ่มควบคุมชุดนี้ก็จะสั่งต่อไปที่พอร์ทเพื่อให้เป็นไปตามข้อกำหนดของคำสั่งนั้น ๆ เช่น ให้พอร์ท A เป็น อินพุทพอร์ท B เป็นเอ้าพุทเหล่านี้เป็นต้น ส่วนกรณีเมื่อมีการอ่านเขียนพอร์ทจาก CPU นั้น กลุ่มควบคุมโลจิกการเขียนอ่านจะเป็นตัวที่ส่งสัญญาณไปบอกแก่กลุ่มควบคุมชุด ในแต่ละชุดอีกที ทั้งนี้แล้วแต่ว่า CPU จะมีการอ่านเขียนพอร์ทของกลุ่มควบคุมชุดใด

ต่อไป เรามาดูถึงความหมาย ของขาต่าง ๆ ของไอซี 8255 เพื่อจะได้ต่อใช้งานได้อย่าง ต้องต่อไป

D0-D7 เป็นขาข้อมูลของ 8255 ที่ใช้ติดต่อกับตัวไมโครโปรเซสเซอร์ซึ่งข้อมูลที่จะเข้าออกสู่พอร์ทต่าง ๆ ของ 8255 จะต้องผ่านขาข้อมูลนี้ CS เป็นขาอินพุทที่รับสัญญาณโลจิก"0"จากภายนอกเพื่อแสดงว่าต้องการเลือกใช้ไอซีเบอร์นี้หากได้รับโลจิก "1" ก็จะทำให้ไอซีตัวนี้ไม่ทำงานคือ ไม่รับรู้สัญญาณ

RD เป็นขาอินพุทที่รับสัญญาณจากตัวไมโครโปรเซสเซอร์ โดยหากมีโลจิกเป็น "0" จะเป็นการแสดงว่า CPU ต้องการที่จะอ่านข้อมูลจากตัว 8255

WR เป็นขาอินพุทที่รับสัญญาณจากตัวไมโครโปรเซสเซอร์ โดยหากมีโลจิกเป็น "0" ก็จะเป็นการแสดงว่า CPU ต้องการที่จะเขียนข้อมูลเข้าสู่ตัว 8255

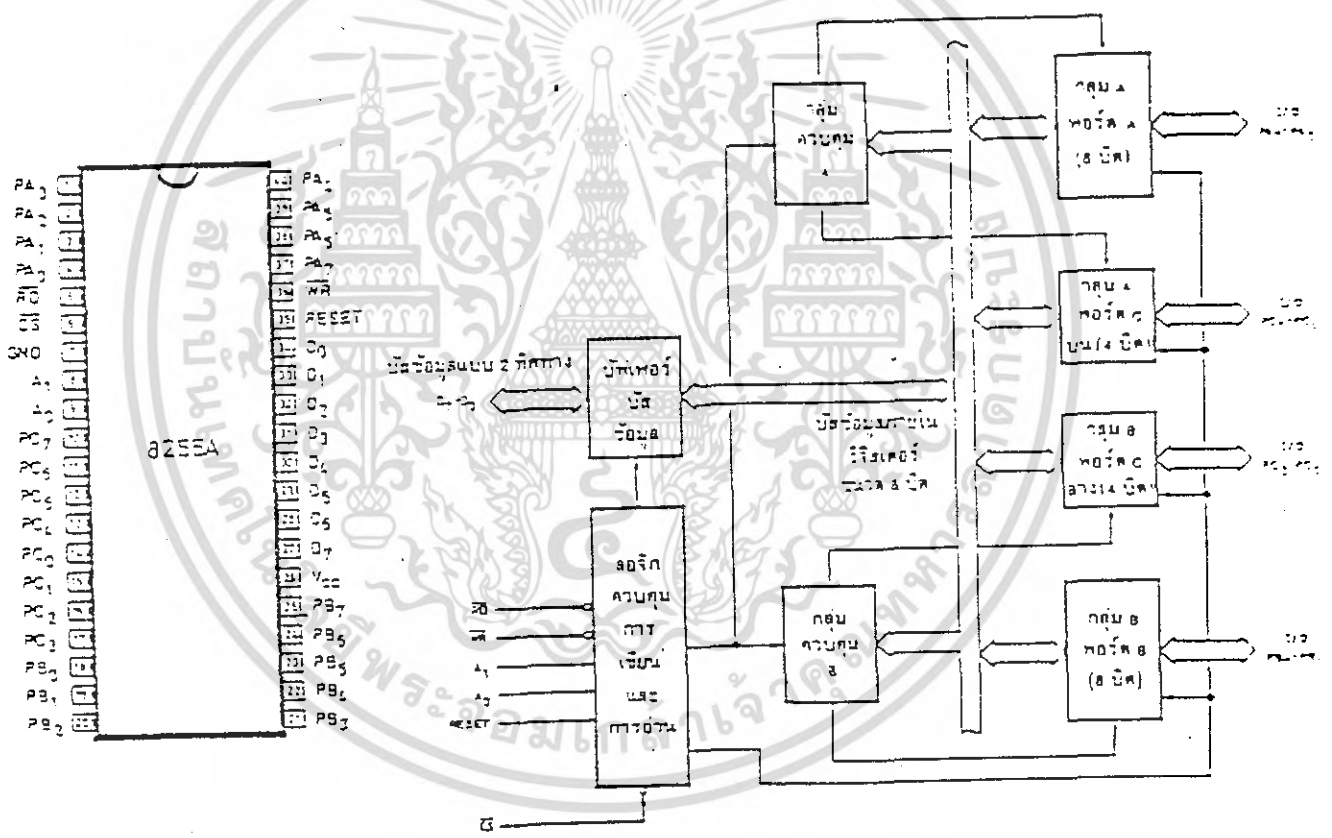
A0-A1 เป็นอินพุทที่รับแอดเดรสจากตัวไมโครโปรเซสเซอร์ ที่ถอดระหัสตำแหน่งของ 8255 เรียบร้อยแล้วโดยจะมีตำแหน่งใช้งาน 4 ตำแหน่งเพื่ออ่านเขียน รีจิสเตอร์ (พอร์ท) ของ 8255 ที่มีอยู่ด้วยกัน 4 ตัว

RESET เป็นขาอินพุทที่รับสัญญาณจากภายนอกเข้ามาทำการรีเซทตัว 8255 โดยหาก ได้ รับโลจิก "1"จะทำให้พอร์ททุกพอร์ทเป็น อินพุทพอร์ทั้งหมดทั้งนี้เพื่อไม่ต้องการให้มี สัญญาณออกไปรบกวนต่อระบบภายนอกเพื่อ 8255 ได้รับสัญญาณรีเซท

PA_0-PA_7 เป็นขาสัญญาณพอร์ท A ที่ใช้ติดต่อกับโลกภายนอก

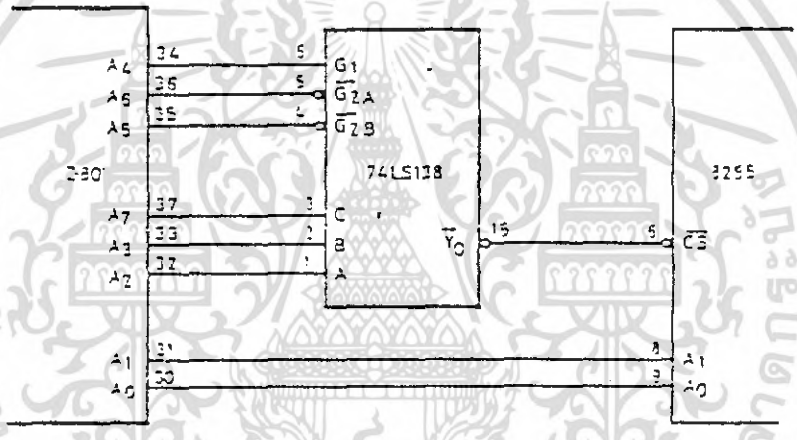
PB_0-PB_7 เป็นขาสัญญาณพอร์ท B ที่ใช้ติดต่อกับโลกภายนอก

PC_0-PC_7 เป็นขาสัญญาณพอร์ท C ซึ่งสามารถโปรแกรมแยกกันได้อีกต่างหาก



การต่อใช้งาน 8255

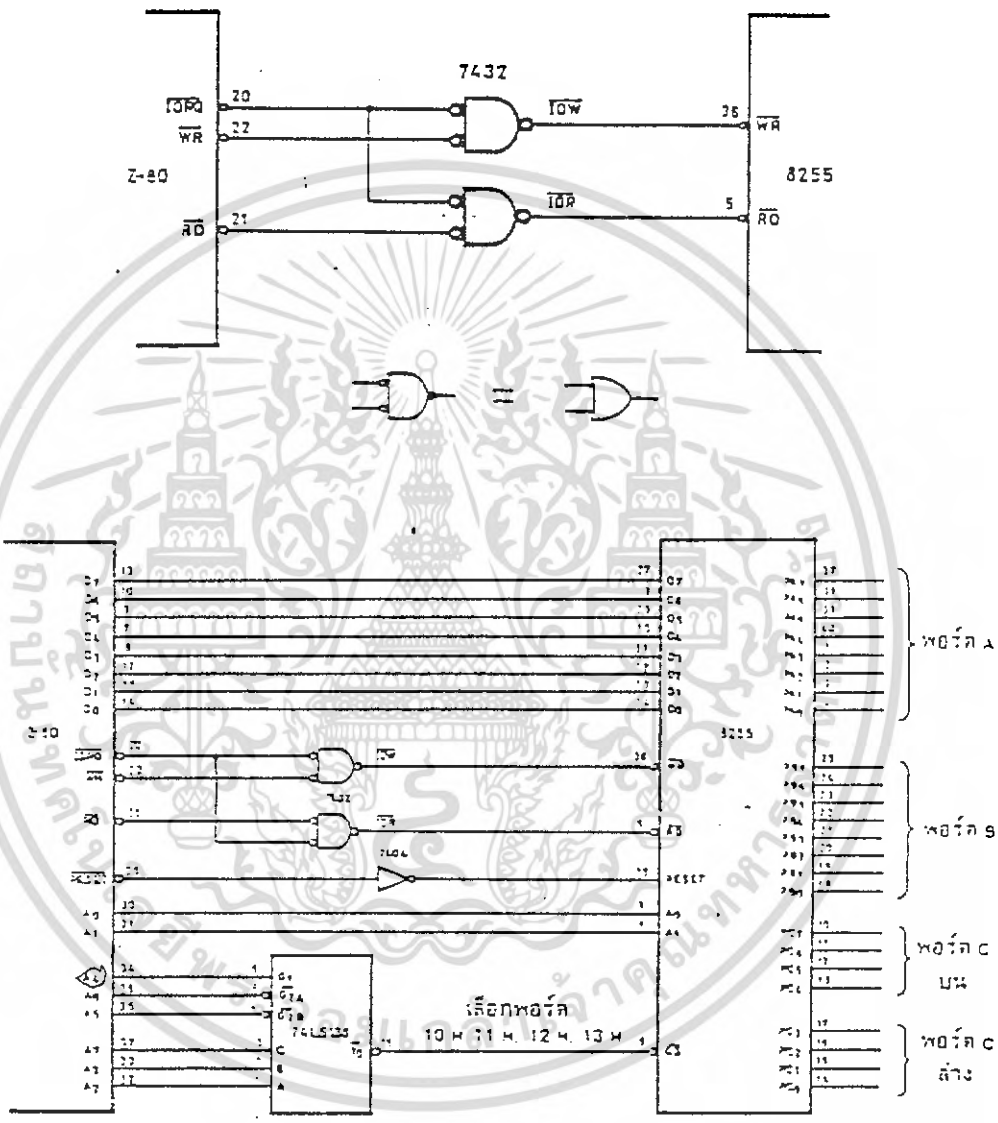
หากดูที่ขาของ 8255 ที่รูป แล้วจะสังเกตุเห็นได้ว่าเราสามารถต่อขา 8255 บางส่วนได้โดยตรงกับขาไมโครโปรเซสเซอร์เลย (Z-80) เช่นขา D0-D7,A0-A1 เป็นต้น หากแต่บางขาเราจำเป็นต้องมีการดัดแปลงสัญญาณที่ได้จาก CPU (ซึ่งกรณีนี้เราใช้เบอร์ Z-80) เสียก่อน โดยหากเราถอดระหัสแอดเดรสของ 8255 ให้เป็นพอร์ทที่แอดเดรส 10H, 11H, 12H, 13H, เราสามารถทำ ดีโค็ดเดอร์ได้โดยง่ายดังรูป



รูปแสดงการดีโค็ดแอดเดรสพอร์ทให้ 8255

สังเกตได้ว่า CS จะแอดดีฟทุกครั้งหาก Z-80 อ้างพอร์ทที่แอดเดรส 000100XX โดยค่าของ XX คือ A0, A1 ที่เราจะต่อตรงเข้ากับ 8255 เพื่อทำการเลือกรีจิสเตอร์ควบคุมและพอร์ททั้งสามของ 8255

สัญญาณการควบคุมอีกส่วนที่สำคัญก็คือสัญญาณควบคุมการอ่านเขียนพอร์ทของ 8255 โดยหาก WR ได้รับแอดดีฟ "0" ก็จะเป็นการเขียนข้อมูลจาก CPU เข้าสู่ตัว 8255 หรือออกสู่พอร์ทที่ต้องการ และเช่นเดียวกันหาก RD ได้รับโลจิก "0" ก็จะเป็นการอ่านข้อมูลพอร์ทจากตัว 8255 เข้าสู่ CPU ซึ่งหากต่อ 8255 เข้ากับตัว Z-80 แล้วเราจะเป็นต้องทำสัญญาณการอ่านเขียน พอร์ทของ Z-80



การโปรแกรม 8255

เราได้ทราบมาแล้วในรูป ว่าโครงสร้างภายในของ 8255 มีกลุ่มควบคุมชุดอยู่ 3 กลุ่ม ซึ่งทั้งสามกลุ่มนี้จะทำงานร่วมกันดังที่กล่าวมา และเราสามารถจะควบคุมการทำงานของพอร์ทจาก CPU ได้ โดยสั่งงานมาที่กลุ่มควบคุมดังกล่าว แต่ตัว CPU จะมองเห็น 8255 เป็น 4 พอร์ทด้วยกันโดยแต่ละพอร์ทเสมือนเป็น รีจิสเตอร์ที่ CPU สามารถจะทำการอ่าน เขียนได้ แต่ละพอร์ทจะอยู่คนละแอดเดรสกันดังที่เราได้ทำการดีโค้ดให้ 8255 ที่แอดเดรส 10H, 11H, 12H, 13H (ตามสัญญาณ A0-A1) และเราจะได้ตำแหน่งของพอร์ท 8255 แต่ละตัวดังนี้

10J ========= พอร์ท A

11H ========= พอร์ท B

12H ========= พอร์ท C

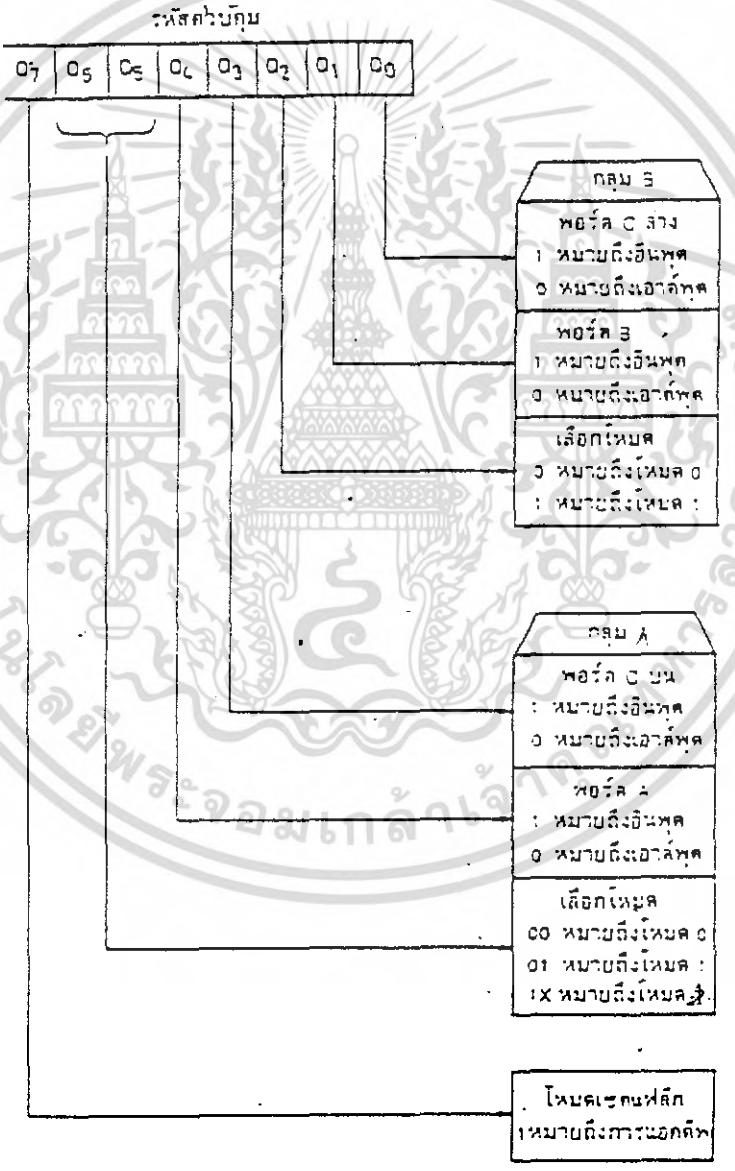
13H ========= พอร์ท Control

ซึ่งหาก มีการอ่านเขียนไปยังพอร์ทดังกล่าวก็จะใช้ร่วมกับสัญญาณ RD, WR โดย WR หมายถึง เอ้าพุท ข้อมูลและ RD แอคตีฟหมายถึง อินพุทข้อมูล ดังนั้นเราจะได้โลจิกที่ทำของ 8255 ในลักษณะต่าง ๆ ดังนี้

RD	WR	A1	A2	
1	0	0	0	เขียนพอร์ต A ซึ่งเป็นข้อมูล
0	1	0	0	อ่านพอร์ต A ซึ่งเป็นข้อมูล
1	0	0	1	เขียนพอร์ต B ซึ่งเป็นข้อมูล
0	1	0	1	อ่านพอร์ต B ซึ่งเป็นข้อมูล
1	0	1	0	เขียนพอร์ต C ซึ่งเป็นข้อมูล
0	1	1	0	อ่านพอร์ต C ซึ่งเป็นข้อมูล
1	0	1	1	เขียนข้อมูลซึ่งเป็นรหัสควบคุม
0	1	1	1	อ่านเข้ามาซึ่งไม่มีความหมายใด

แสดงตารางของโลจิกเมื่อทำการติดต่อกับ 8255

การใช้งานเราจะต้องส่งระหัสควบคุม (Control code) เข้าไปยังพอร์ทควบคุม หรือเรียกอีกอย่างหนึ่งว่า รีจีสเตอร์คาบคุม ซึ่งจะเป็นข้อมูลขนาด 1 ไบท์ส่งไปที่ แอดเดรส 13H กรณีนี้เราถอดระหัสไว้ ที่ 13H โดยความหมายของแต่ละบิทที่เราส่งไปโปรแกรมการทำงานเป็น ดังนี้



ความหมายของแต่ละบิทในรหัสควบคุม

บิต D7 เป็นบิตที่แสดงว่าในไบท์นี้เป็นระหัสควบคุม ถ้าเป็น "1" โดยแต่ละบิตจะมีผลต่อการเปลี่ยนแปลงโหมดต่าง ๆ ของ 8255 หากเป็น "0" จะเป็นการเซตบิทของพอร์ท C

บิต D6,D5 เป็นการเลือกโหมดของพอร์ท A ซึ่งจะมีอยู่ด้วยกัน 3 โหมด คือ 0,1,2

บิต D4 เป็นการกำหนดให้พอร์ท A ให้เป็น อินพุทหรือเอ้าพุท โดยหากเป็น "1" ก็จะเป็นอินพุท หากเป็น "0" ก็แสดงว่าให้เป็นเอ้าพุท

บิต D3 เป็นการกำหนดให้พอร์ท C บน ให้เป็นอินพุทหรือเอ้าพุท โดยหากเป็น "1" ก็จะเป็นอินพุท หากเป็น "0" ก็แสดงว่าให้เป็นเอ้าพุท

บิต D2 เป็นการกำหนดโหมดการทำงานของพอร์ท B โดยหากเป็น "0" หมายถึงเลือกให้พอร์ท B ทำงานในโหมด 1

บิต D1 เป็นการกำหนดให้พอร์ท B ให้เป็นอินพุทหรือเอ้าพุท โดยหากเป็น "1" ก็จะเป็นอินพุท หากเป็น "0" ก็แสดงว่าให้เป็น เอ้าพุท

บิต D0 เป็นการกำหนดให้พอร์ท C ล่าง ให้เป็น อินพุทหรือเอ้าพุทโดยหากเป็น "1" ก็จะเป็นอินพุท หากเป็น "0" ก็แสดงว่าให้เป็น เอ้าพุท

การโปรแกรมจะเริ่มจากการส่งค่าระหัสควบคุม 1 ไบท์ดังที่กล่าวนี้ไปสู่พอร์ทควบคุม หลังจากนั้นหากต้องการเรียกไปถึงพอร์ทใดก็สามารถอ้างได้ตามแอดเดรสทันที เช่น ต้องการโปรแกรมให้พอร์ท A,B,C ทั้งหมดเป็นเอ้าพุทพอร์ท เราก็จะได้ระหัสควบคุมเป็น 1000000 หรือ 80H เราก็จะสั่งเป็น

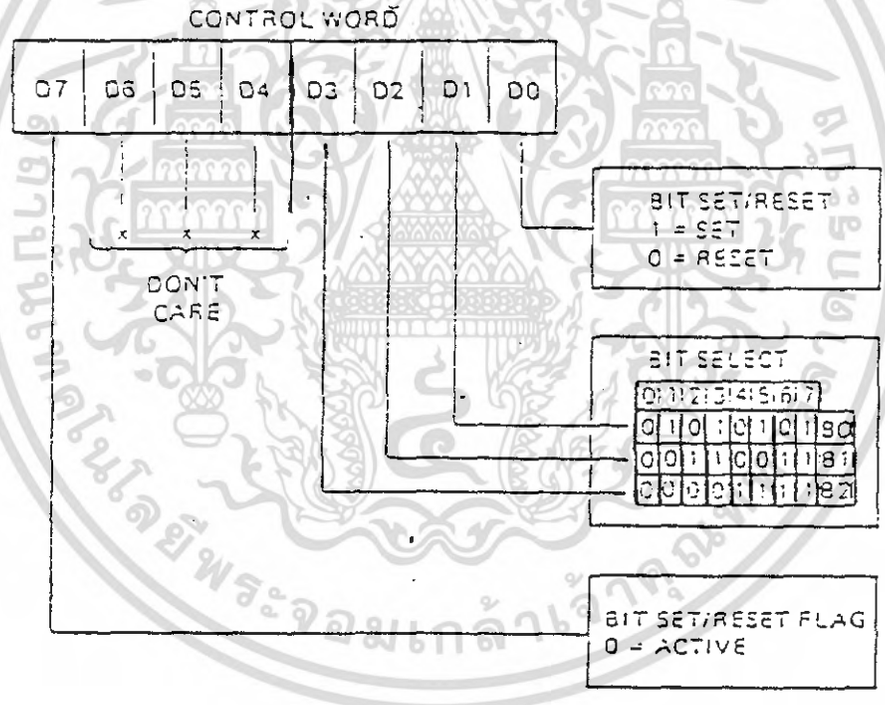
```
LD A. 080H    ; กำหนดระหัสควบคุม
OUT (013H),A  ; เป็นการส่งระหัสควบคุมสู่รีจิสเตอร์ควบคุม 8255
```

จากนี้ทั้งสามพอร์ทก็จะเป็น เอ้าพุทพอร์ทให้เราตามต้องการเมื่อเราจะส่งค่าออกไปก็สามารถกระทำได้โดยง่าย เช่นต้องการส่งค่า 088H ออกไปที่พอร์ท B,C เราจะทำดังนี้

```
LD A, 088H    ; ค่าข้อมูลที่ต้องการส่ง
OUT (011H),A  ; ส่งออกไปพอร์ท B
OUT (012H),A  ; ส่งออกไปพอร์ท C
```

<u>กรณีพิเศษของระหัสควบคุม</u>

ปกติระหัสที่เราต้องส่งไปถึงพอร์ทควบคุม หรือรีจีสเตอร์ควบคุมนี้จะต้องเป็นการเซทโหมด. พอร์ทอินพุทเอ้าพุทและในระหัสนั้น บิต 7 จะต้องเป็น "1ไ เสมอ ที่นี้หาก บิต 7 นี้เป็น "0" บ้างจะเกิดอะไรขึ้น ถ้าหากบิต 7 เป็น "10" และถูกส่งไปที่แอดเดรสของพอร์ทควบคุมแล้ว 8255 จะถือว่าเป็นคำสั่งของการ เซท/รีเซท บิตของพอร์ท C ทันที โดยจะมีฟอร์มแมทดังรูป



แสดงถึงระหัสควบคุมที่ใช้ในการเซท/รีเซท บิต

ตัวอย่างเช่น	LD A, 80H	**; ระหัสควบคุมให้ทั้งสามพอร์ตเป็น** OUTPUT
	OUT (13H),A	**; ส่งสู่รีจีสเตอร์ควบคุม**
	LD A, 00001001B	: ระหัสควบคุมใช้เซทบิตที่ 4 ของพอร์ท C เป็น "1"
	OUT (013H),A	: บิต 4 พอร์ท C เป็น "1"
	DEC A	**; เปลี่ยนเป็นรีเซท**
	OUT (013H),A	; กำหนดให้ บิต 4 พอร์ท C เป็น "0"

จะเห็นว่าสามารถนำไปประยุกต์ใช้งานได้ เช่นเป็นตัวสร้าง ฟัลซ์และกำหนดใช้งานเปิด-ปิด อุปกรณ์ด้วย พอร์ท C ที่มีคำสั่งไม่ยุ่งยากและเป็น อิสระเป็นต้น

การทำงานในโหมด 0

จัดว่าเป็นโหมดพื้นฐานที่ นิยมใช้กันมากที่สุดเนื่องด้วยความง่ายตรงไปตรงมา คือทั้งสามพอร์ทเราสามารถจะให้พอร์ทใดเป็น อินพุท,เอ้าพุท ได้โดยเฉพาะพอร์ท C ยังแยกให้เป็น 2 ชุด ๆ ละ 4 บิต ซึ่งในแต่ละชุดนี้ก็สามารถจะโปรแกรมให้ชุดหนึ่งเป็น อินพุทหรือเอ้าพุทได้อีก ฉะนั้นโดยสรุปแล้วก็จะมีพอร์ทที่จะโปรแกรมให้เป็นอินพุทหรือเอ้าพุทได้เสมือน 4 พอร์ทคือ พอร์ท A,พอร์ท B, พอร์ท C บนและพอร์ท C ล่าง (แต่โปรแกรมแยกเฉพาะบิตในแต่ละพอร์ทไม่ได้)

1	0	0	X	X	0	X	X

BIT 7 6 5 4 3 2 1 0

แสดงระหัสคำสั่งโหมด 0

ซึ่งหากเราดูที่ ระหัสคำสั่งแล้วจะเห็นว่ามีอยู่ 4 บิต ที่ถูกกำหนดตายตัวคือ บิต 7,5,บิต 6 และบิต 2 ส่วนที่เหลืออีก 4 บิต ก็คือข้อกำหนดว่าจะให้พอร์ทใดเป็น อินพุท เอ้าพุท นั้นเอง ซึ่งหากเราให้พอร์ทใดเป็นอินพุทเราก็ใส่โลจิก "1" ที่บิตนั้น หรือหากต้องการให้พอร์ทใดเป็น เอ้าพุทก็ให้ใส่ "0" ที่บิตนั้น จะเห็นได้ว่าจะมีความเป็นไปได้ในการกำหนดลักษณะของพอร์ทในโหมดนี้อยู่ 16 อย่างด้วยกัน

<u>ตัวอย่างเช่น</u> 1) ต้องการให้ทุกพอร์ทเป็น เอ้าพุทหมด เราจะได้ระหัสคำสั่งเป็น :-

1	0	0	0	0	0	0	0

2) ต้องการให้ พอร์ท A ---- เอ้าพุท
พอร์ท B ---- อินพุท
พอร์ท C บน-- อินพุท
พอร์ท C ล่าง-- เอ้าพุท

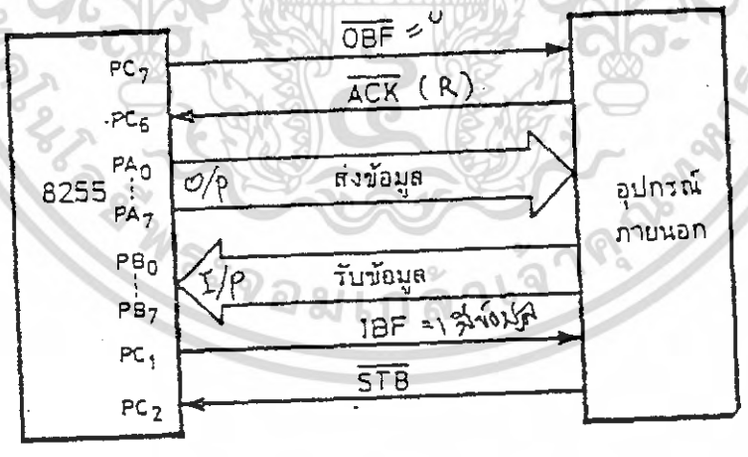
เราจะได้ระหัสคำสั่งเป็น

1	0	0	0	1	0	1	0

		A	PA7-PA0
D0-D7	**8255**	C	PC7-PC4
		C	PC3-PC0
		B	PB7-PB0

แสดงตัวอย่าง PORT ที่ถูกโปรแกรมในโหมด 0

การทำงานในโหมด 1



รูปแสดงสัญญาณต่างเมื่อ 8255 อยู่ในโหมด 1

วิธีการทำ Handshake นี้จะมีประโยชน์มากเพราะปกติอุปกรณ์ภายนอกมักจะทำงานได้ช้ากว่าตัวไมโครโปรเซสเซอร์อยู่แล้ว ด้วยวิธีการนี้จะทำให้ตัวไมโครโปรเซสเซอร์สามารถติดต่อกับอุปกรณ์ภายนอกด้วย 8255 ได้อย่างมีประสิทธิภาพ

ในรูปจะแสดงให้เราเห็นถึงบล็อกของการติดต่อระหว่างอุปกรณ์ภายนอกกับ 8255 ซึ่งได้กำหนดให้พอร์ท A เป็นเอ้าพุทพอร์ท และพอร์ท B เป็นอินพุทพอร์ท (เราอาจกำหนดให้อยู่ในลักษณะอื่นก็ได้) และเช่นเดียวกับโหมด 0 ที่เราได้กล่าวมาแล้วคือ ก่อนที่เราจะใช้งาน 8255 อันดับแรกเลยเราต้องทำการโปรแกรมมันก่อนโดยการส่งคำสั่งควบคุม ขนาด 1 ไบท์ไปที่รีจีสเตอร์ควบคุมของ 8255 ก่อนซึ่งหากเรากำหนดให้พอร์ทเป็นดังรูปที่ 25 เราจะได้คำสั่งควบคุมเป็น

1	0	1	0	X	1	1	1

BIT D7 D6 D5 D4 D3 D2 D1 D0

ในบิทอื่น ๆ นอกจากบิทที่ 3.0 นั้นนักศึกษาสามารถเข้าใจได้เพราะเคยกล่าวมาแล้ว ส่วนบิท ที่ใส่เครื่องหมาย X นั้นเราต้องมาพิจารณาเพื่อใส่ค่าต่อไป คือ ในลักษณะการทำงานโหมด 1 นี้เราจะมีตารางที่แสดงถึงการใช้ พอร์ท C เพื่อเป็นสัญญาณควบคุมทั้งพอร์ท A และพอร์ท B ในกรณีอินพุทดังนี้

ขา	กรณีอินพุท	กรณีเอ้าท์พุท
PC0	$INTR_B$	$INTR_B$
PC1	IBF_B	OBF_B
PC2	STB	ACK_B
PC3	$INTR_A$	$INTR_A$
PC4	STB_A	I / O
PC5	IBF_A	I / O
PC6	I / O	ACK_A
PC7	I / O	OBF_A

ตารางแสดงถึงขาของพอร์ท C ที่ถูกใช้เป็นสัญญาณควบคุมของพอร์ท A,B ในการทำงาน

โหมด 1 ทั้งกรณีอินพุทและเอ้าพุท

ฉะนั้นจากตารางเราจะเห็นว่าในบิทที่ 3 นั้นขึ้นอยู่ว่าผู้ใช้ต้องการกำหนดให้ PC6,PC7 เป็น อินพุท หรือเอ้าพุท (โดยใส่ 1 เมื่อต้องการเป็นอินพุทและใส่ 0 เมื่อต้องการให้เป็น เอ้าพุท) ส่วนใน บิทที่ 0 นั้นไม่สนใจเพราะเราไม่สามารถกำหนดอะไรได้ (PC0-PC2 ถูกนำไปเป็นสัญญาณควบคุมแก่พอร์ท B แล้วและ PC3 ถูกนำไปเป็นสัญญาณควบคุมแก่พอร์ท A แล้ว เช่นเดียวกัน)

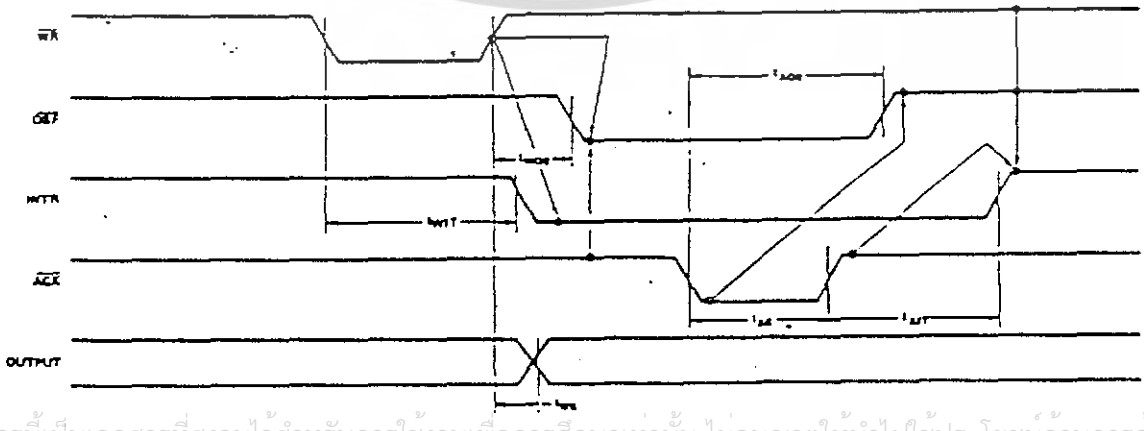
ลำดับสัญญาณในกรณีเอาท์พุทโหมด 1

การเอ้าพุทข้อมูลหมายถึงการส่งข้อมูลจากตัว CPU ออกมาสู่ตัว 8255 ที่พอร์ทนั้น ๆ เพื่อรอสัญญาณการรับข้อมูลจากอุปกรณ์ภายนอกรับเอาข้อมูลนี้ไป สัญญาณที่ใช้ในกรณีเอ้าพุทข้อมูลที่ตัว 8255 มีดังนี้ :-

- OBF (OUTPUT BUFFER FULL) เป็นเอ้าพุท จะแอคตีฟที่โลจิก 0 เป็นตัวบอกว่าขณะนี้ที่ตัว 8255มีข้อมูลจาก CPU อยู่ยังไม่ถูกอ่านจากอุปกรณ์ภายนอก

- ACK (ACKNOWLEDGE) เป็นอินพุทแอคตีฟที่โลจิก 0 เป็นสัญญาณจากอุปกรณ์ภายนอกที่ส่งมาเพื่อรับเอาข้อมูลจาก 8255 ไป

- INTR (INTERRUPT REQUEST) เป็นสัญญาณเอ้าพุทจากตัว 8255 แอคตีฟที่โลจิค 0 ปกติเราจะใช้ไปทริกให้กับ CPU ที่ขา INT เพื่อบอกให้ CPU ทราบว่าข้อมูลได้ถูกอ่านจาก 8255 ไปแล้ว ซึ่งเราสามารถ เซท/รีเซท ว่าต้องการให้เกิดสัญญาณที่ขา INTR นี้หรือไม่ก็ได้โดยซอฟร์แวร์



แสดงถึงลำดับสัญญาณที่ 8255 ส่งข้อมูลออกมาเพื่อให้อุปกรณ์ภายนอกอ่านข้อมูลไป

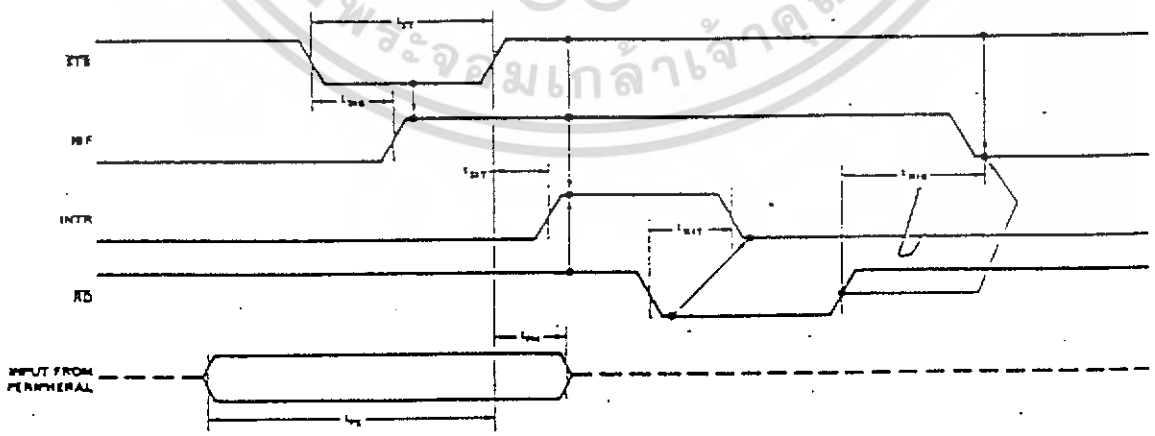
<u>อธิบายได้ดังนี้</u> เมื่อ CPU ส่งสัญญาณการเขียนข้อมูลเข้ามาเก็บไปที่ตัว 8255 (WR) จะเป็นผลทำให้ OBF แสดงภาวะว่ามีข้อมูลเข้ามาแลทชีไว้ที่พอร์ท A (อาจเป็นพอร์ทอื่นตามที่เราโปรแกรมไว้) โดยจะให้โลจิกออกเป็น 0 และสัญญาณ INTR ก็จะเป็น 0 ด้วยหากเราทำการเซทไว้ก่อน ที่นี้สัญญาณก็จะค้างอยู่อย่างนี้ต่อไปเรื่อยหากไม่มีสัญญาณการอ่านข้อมูลจากอุปกรณ์ภายนอกมาอ่านสัญญาณนั้น คือ ACK

เมื่ออุปกรณ์ภายนอกต้องการอ่านเอาข้อมูลจากตัว 8255 ก็จะทำการตราจดูสัญญาณที่ขา OBF ของ 8255 ก่อนว่าเป็น 0 หรือไม่เพราะหากได้ 0 เป็นการแสดงว่าข้อมูลมีอยู่ในพอร์ท พร้อมที่จะทำการอ่านได้ จึงส่งสัญญาณแอคตีฟโลจิกศูนย์มาที่ขา ACK ดังแสดงในรูป เป็นการอ่านเอาข้อมูล จาก 8255 ออกไปนั่นเอง จากนั้นสัญญาณ OBF,INTR ก็จะกลับคืนเป็นโลจิก 1 ดังเดิมซึ่งจะทำให้ CPU ทราบได้ว่ามีการอ่านเอาข้อมูลที่พอร์ทไปเรียบร้อยแล้ว CPU สามารถที่จะทำการส่งข้อมูลเข้ามาใหม่ได้

<u>ลำดับสัญญาณในกรณีอินพุท โหมด 1</u>

การอินพุทพอร์ท หมายถึง การที่อุปกรณ์ภายนอกส่งข้อมูลเข้ามาเก็บไว้ที่ตัว 8255 (แต่ต้องตรวจดูก่อนเช่นกันว่า 8255 ว่างหรือไม่) เพื่อให้ตัว CPU มาอ่านเอาข้อมูลนั้นไปสู่ตัว CPU ต่อไป

รูปลำดับสัญญาณและความหมายที่ต้องใช้ในกรณีอินพุทมีดังนี้ :-



รูปแสดงลำดับสัญญาณที่เกิดขึ้นในกรณีการทำงานโหมด 1 อินพุท

- IBF (INPUT BUFFER FULL) เป็นสัญญาณเอ้าพุท ที่แอคตีฟโลจิก 1 จะเป็นตัวแสดงว่าขณะนี้นั้นข้อมูลในตัว 8255 เต็มอยู่หรือไม่ หมายความว่าข้อมูลถูกอ่านด้วย CPU ไปหรือยัง หากยัง ที่ขาสัญญาณนี้ก็จะแสดงโลจิกเป็น 1 อยู่ แต่หากมีการอ่านเอาข้อมูลไปแล้วก็จะแสดงโลจิกเป็น 0 เพื่อให้อุปกรณ์ภายนอกสามารถส่งข้อมูลตัวต่อไปเข้ามาได้ทันที

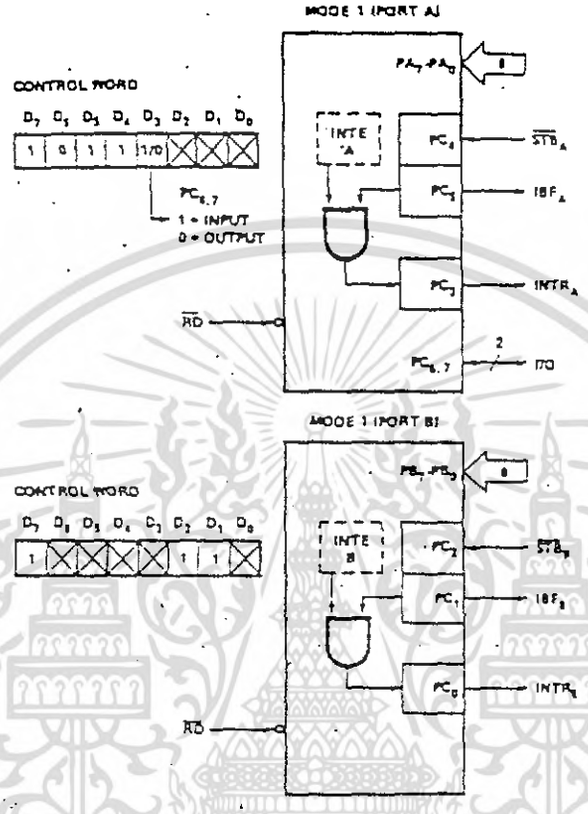
- STB (STROBE INPUT) เป็นสัญญาณอินพุทที่แอคตีฟที่โลจิก 0 จะเป็นขาที่รับสัญญาณจากอุปกรณ์ภายนอก เพื่อแลทช์ข้อมูลที่ส่งมาจากภายนอกเข้าสู่ตัว 8255

- INTR (INTERRUPT REQUEST) เป็นสัญญาณเอ้าพุท แอคตีฟที่โลจิก 1 มักใช้เป็นสัญญาณหริกอินเตอร์รัฟท์ให้กับ CPU เพื่อบอกให้ทราบว่าอุปกรณ์ภายนอกส่งข้อมูลมาแล้ว อธิบายได้ดังนี้ เมื่ออุปกรณ์ภายนอกต้องการส่งข้อมูลเข้าสู่ตัว 8255 ก็ต้องตรวจดูก่อนว่าที่พอร์ทนั้นมีข้อมูลตกค้างอยู่หรือว่างหรือไม่โดยการตรวจที่ขาสัญญาณ IBF ว่าเป็นโลจิก 0 หรือไม่เพราะหากเป็นโลจิก 0 จะหมายถึงว่าง แต่หากเป็นโลจิก 1 จะหมายถึงว่าข้อมูลยังคงอยู่ เมื่อตรวจดูและทราบว่าเป็น 0 ก็สามารถส่งข้อมูลไปได้โดยส่งสัญญาณ พัลซ์โลจิก 0 ไปที่ขา STB เพื่อบอกให้ 8255 ได้ทำการแลทช์ข้อมูลที่ส่งให้นั้น และเมื่อแลทช์ข้อมูลไว้แล้วก็เป็นผลทำให้ IBF มีโลจิกไปเป็น 1 เพื่อแสดงถึงว่าได้แลทช์ข้อมูลส่งให้นั้น และเมื่อแลทช์ข้อมูลเข้าสู่พอร์ทแล้ว พร้อมกันนั้นก็จะให้สัญญาณ INTR เป็นโลจิก 1 เพื่อแสดงให้ตัว CPU ทราบว่าได้แลทช์ข้อมูลจากอุปกรณ์ภายนอกไว้แล้ว (หากเราต่อสัญญาณอินเตอร์รัฟท์สู่ตัว CPU) และสัญญาณก็จะค้างลักษณะนี้ต่อไปจนกว่าจะมีสัญญาณการอ่านข้อมูลจาก CPU มาอ่านเอาข้อมูลจากพอร์ท 8255 ไปคือสัญญาณ RD และเมื่อ CPU ส่งสัญญาณ RD มาอ่านแล้วก็จะเป็นผลให้สัญญาณ IBF กลับเป็นโลจิก 1 และ INTR กลับเป็นโลจิก 0 ดังเดิม ซึ่งหากอุปกรณ์ภายนอกตรวจดูที่สัญญาณ IBF ก็จะทราบว่ามันสามารถส่งข้อมูลตัวต่อไปมาที่ 8255 ได้แล้ว ขอให้สังเกตสัญญาณในรูป ประกอบด้วย

ตัวอย่าง เมื่อเราโปรแกรมให้พอร์ท A เป็น อินพุทและพอร์ท B เป็นอินพุท ก็สามารถแสดงถึงรูปของขาสัญญาณที่ใช้งานและคำสั่งควบคุมที่ส่งไปให้ 8255 ดังรูป นี้

1	0	1	1	I/O	1	1	X

PC6,PC7



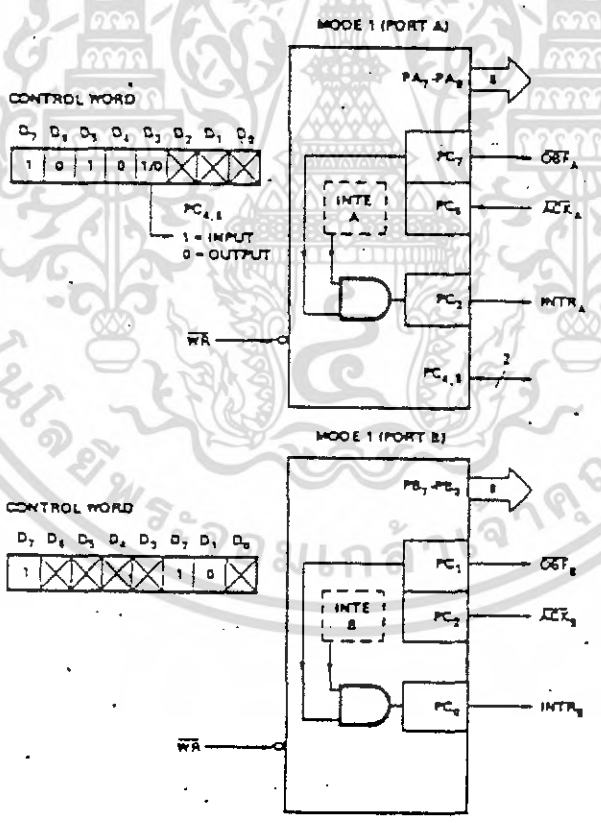
รูปแสดงถึงขาสัญญาณเมื่อ 8255 ถูกโปรแกรมให้พอร์ท A เป็นอินพุทและพอร์ท B เป็นอินพุท

ปกติสัญญาณ INTR นักจะไม่ใช้เพราะเราสามารถอ่านข้อมูลจากพอร์ท C ของ 8255 มาตรวจดูสถานะการทำงาน (IBF,OBF) ได้โดยการตรวจเช็คบิทเหล่านั้น แต่หากเราต้องการให้สัญญาณ INTR นี้เราก็สามารถทำได้ โดยเราสามารถจะเซทหรือรีเซทบิทเพื่อจะกำหนดให้เกิดสัญญาณ INTR หรือไม่ก็ได้ไม่ว่ากรณีของอินพุทหรือเอ้าพุท ซึ่งวิธีการก็คือการส่งคำสั่งควบคุมที่บิท 7=0 ไปที่รีจิสเตอร์ควบคุมซึ่งเหมือนกับการใช้คำสั่งเซทบิทพอร์ท C ดังที่ได้กล่าวมาแล้วในตอนต้น

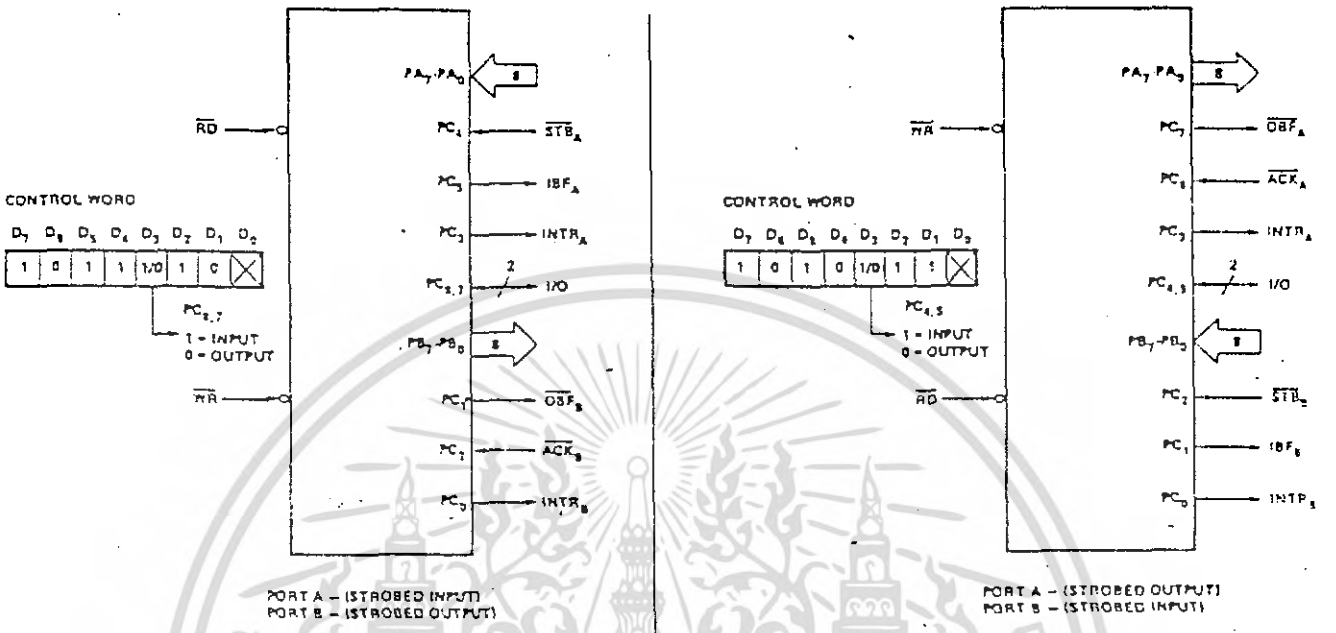
โดยจะขอสรุปว่าหากต้องการให้มีการส่งสัญญาณ INTR เกิดขึ้นก็ส่งค่า 1 ไปที่บิทนั้น หากไม่ต้องการให้เกิดสัญญาณ INTR ก็ส่ง 0 ไปที่บิทนั้น ทั้งนี้เพราะจากรูป นั้นเอ้าพุทที่ได้สัญญาณ INTR คือผลจากการ AND กันของสัญญาณ INTE (เป็นเสมือนตัวกำหนดสัญญาณ

อินเตอร์รัฟท์) กับ IBF (กรณีอินพุท) หรือ OBF (กรณีเอ้าพุท) และเราสามารถจะเปิดปิดเกทเพื่อให้สัญญาณอินเตอร์รัฟท์ INTE ผ่านได้หรือไม่โดย กำหนดเซทหรือรีเซทที่ขาพอร์ท C ของสัญญาณ ACK, STB แล้วแต่กรณีว่าเป็นอินพุทหรือเอ้าพุท ดังจะสรุปดังนี้

PORT	กำหนดให้พอร์ทเป็น	สัญญาณ INT	ให้ทำการ SET /RESET ที่
A	INPUT	INTE A	PC4 (STBA)
A	OUTPUT	INTE A	PC6 (ACKA)
B	INPUT	INTE B	PC2 (STBB)
B	OUTPUT	INTE B	PC2 (ACKA)

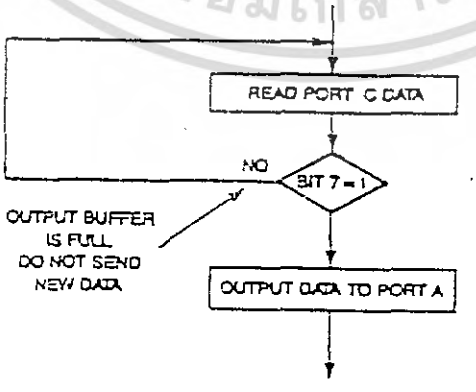


แสดงถึง 8255 ในโหมด 1 ที่โปรแกรมพอร์ท A และพอร์ท B เป็นเอ้าพุท

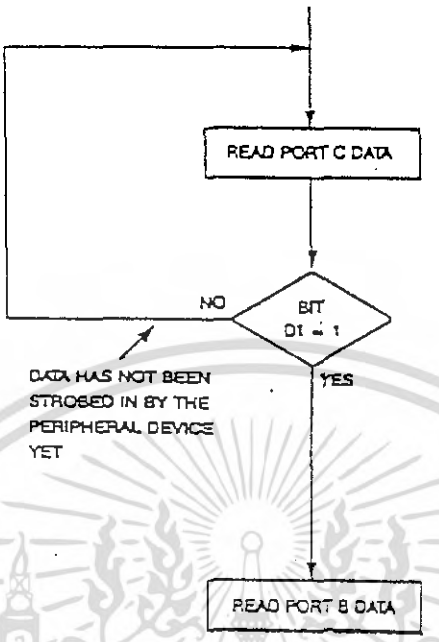


แสดงถึงลักษณะของพอร์ท A,B ที่ถูกโปรแกรมผสมกัน ในโหมด 1

ในการโปรแกรมเพื่อส่งข้อมูลออกไปสู่ 8255 เข้าพุทและรับข้อมูลจาก 8255เข้าสู่ CPU สามารถทำได้โดยง่ายดังแสดงโฟลชาร์ทและตัวอย่างโปรแกรมข้างล่างนี้

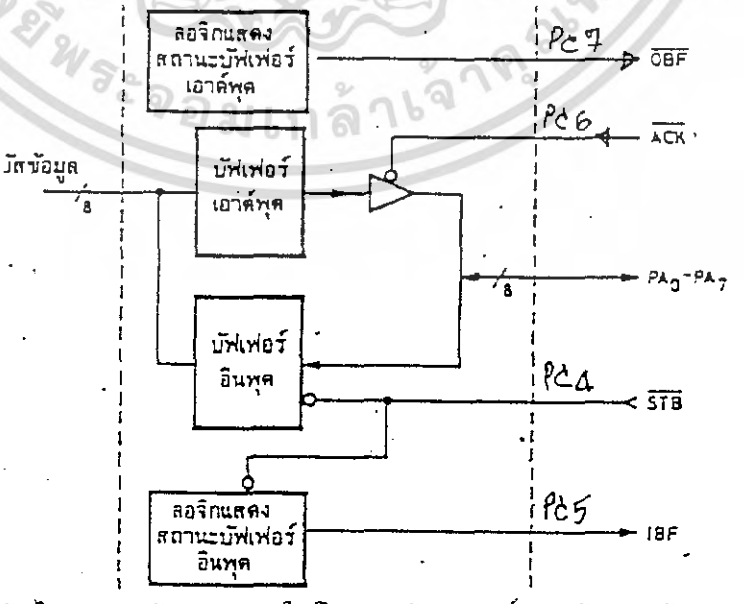


รูปแสดงโฟลชาร์ทและโปรแกรมของการส่งข้อมูลออกไปที่พอร์ท A



การทำงานในโหมด 2

การทำงานในโหมดที่ 2 นี้จะสามารถทำได้เฉพาะพอร์ท A เท่านั้นเนื่องจากว่าการทำงานในโหมดนี้คือการที่กำหนดให้พอร์ท A เป็นได้ทั้งอินพุทและเอ้าพุทได้ในพอร์ทเดียวซึ่งทำให้ต้องสายสัญญาณควบคุมมากขึ้นเป็น 5 เส้น ซึ่งก็ใช้พอร์ท C เป็นขาสัญญาณควบคุมนี้ทำให้ไม่เพียงพอแก่พอร์ท B ที่จะให้เป็นโหมด 2 ฉะนั้นพอร์ท B จึงสามารถทำงานได้เฉพาะโหมด 0,1 เท่านั้น



แสดงถึงบล็อกไดอะแกรมการทำงานในโหมด 2 ของพอร์ท A

การทำงานในโหมดนี้คือการใช้พอร์ท A เป็นทั้งอินพุทแลทช์ข้อมูลและเอ้าพุทแลท์ข้อมูลโดยเอ้าพุทแลท์ก็จะหมายถึงการที่พอร์ท A รับข้อมูลจาก CPU มาทำการแลทช์ไว้เพื่อรอการอ่านข้อมูลนี้ไปด้วยอุปกรณ์ภายนอก ส่วนกรณีของอินพุทแลทช์ก็หมายถึงการเก็บข้อมูลที่อุปกรณ์ภายนอกส่งมาแลทช์ไว้เพื่อรอให้ CPU ทำการอ่านข้อมูลนี้ไปนั่นเอง

การทำงาน โดยทั่วไปก็เหมือนกันกับการทำงานในโหมด 1 ที่ได้กล่าวมาเพียงแต่เป็นการรวมเอาพอร์ทการรับส่งไว้เป็นช่องเดียวนั่นคือ เมื่อกระทำการเอ้าพุทก็จะมีสัญญาณ OBF,ACK,INTR ใช้ติดต่อควบคุมและเมื่อกระทำการอินพุทก็จะมีสัญญาณ IBF, STB, INTR ใช้ในการติดต่อควบคุมการทำงานโดยจะขอกล่าวเป็นลำดับดังนี้

การส่งข้อมูลจาก CPU ไปสู่ 8255 เพื่อออกสู่ภายนอกนั้นขั้นแรกก็ต้องตรวจดูก่อนว่าพอร์ท A ว่างหรือไม่โดยการอ่านค่า บิท PC7 (OBF) มาดูว่าเป็น 1 หรือไม่หากเป็น 1(ว่าง) ก็สามารถส่งข้อมูล ออกไปแลทช์ไว้ที่ 8255 ได้ และเมื่ออุปกรณ์ภายนอกต้องการรับข้อมูลไปก็จะตรวจที่ PC7 (OBF) นี้ ว่าได้โลจิกเป็น 0 หรือไม่ หากเป็น 0 (หมายถึงมีข้อมูลอยู่) ก็จะทำการอ่านเอาไปโดยการส่งสัญญาณ ACK มาที่บิท PC6 (ACK) เพื่ออ่านเอาข้อมูลไปเป็นผลทำให้สถานะของ PC7 (OBF) กลับคืนเป็น 1 อีกครั้งเพื่อแสดงตัวว่า 8255 ว่างที่จะรับข้อมูลตัวต่อไปจาก CPU แล้ว

การรับข้อมูลจาก อุปกรณ์ภายนอกนั้น ก่อนที่จะทำการส่งข้อมูลจากอุปกรณ์ภายนอกมาที่พอร์ท A นั้น อุปกรณ์ภายนอกจะทำการตรวจดูก่อนว่าพอร์ท A ว่างหรือไม่โดยการตรวจที่ บิท PC5 (IBF) ว่าเป็นโลจิก 0 หรือไม่ หากเป็น 0 แสดงว่าว่างก็จะส่งสัญญาณ STB มาที่ขา PC4 เป็นการบอกให้ 8255 ได้ทำการแลทช์ข้อมูลของอุปกรณ์ภายนอกที่ส่งมาไว้ที่พอร์ท A เมื่อแลทช์แล้ว PC4 (IBF) จะเปลี่ยนโลจิกไปเป็น 1 ทันทีเพื่อบอกให้อุปกรณ์ภายจอกทราบว่าข้อมูลถูกแลทช์เข้าสู่พอร์ท A เรียบร้อยแล้วอย่าเพิ่งส่งข้อมูลมาอีก ถึงตอนนี้เมื่อ CPU ตรวจดูที่ PC5 (IBF) ดูโดยการอ่านพอร์ท C ก็จะทราบว่ามีข้อมูลส่งมาแล้วจึงทำการอ่านข้อมูลไปได้ (ส่ง RD มาก่อน) เป็นผลทำให้ PC5 (IBF) กลับโลจิกเป็น 0 อีกครั้งเพื่อให้อุปกรณ์ภายนอกส่งข้อมูลตัวต่อ ไปมาที่ 8255 อีก

ขาสัญญาณเคาบคุมของพอร์ท C ที่ใช้ในการทำสัญญาณควบคุมต่าง ๆ นั้นแสดงดังรูป สังเกตเห็นได้ว่า PC0-PC2 นั้นเราสามารถกำหนดให้เป็น อินพุทหรือเอ้าพุทได้อีกต่างหาก ในกรณีที่พอร์ท B ถูกโปรแกรมในโหมด 0 (เพราะไม่ต้องมีสัญญาณควบคุม) แต่หากพอร์ท B ถูกโปรแกรมในโหมด 1 จะทำให้ต้องใช้ PC0-PC2 นี้เป็นสัญญาณควบคุมของพอร์ท B

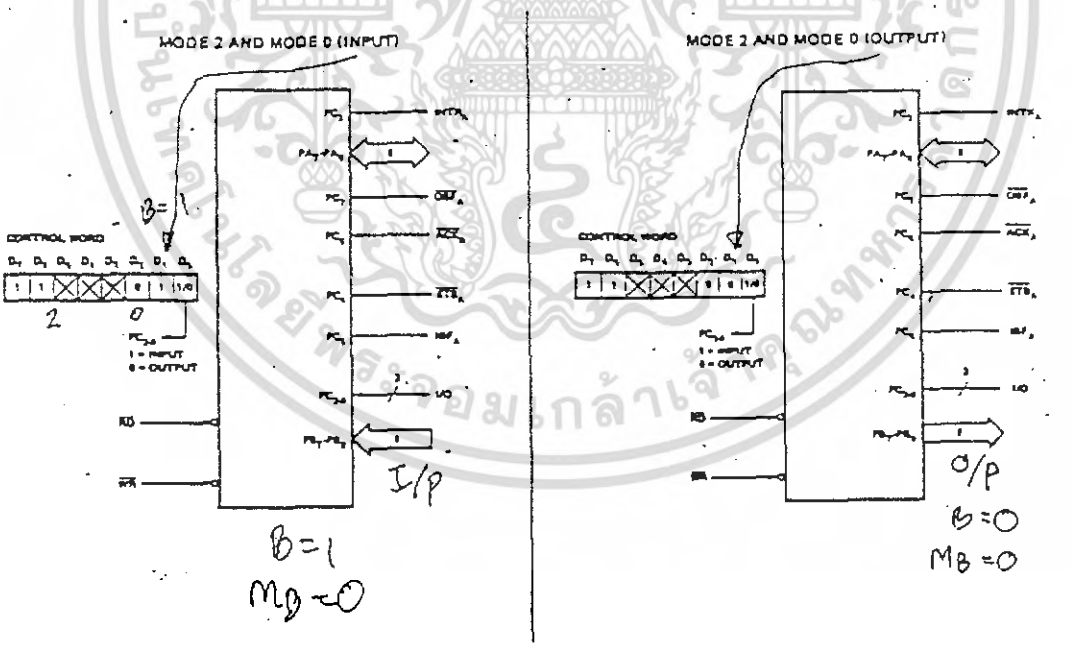
พอร์ตC	PC0	PC1	PC2	PC3	PC4	PC5	PC6	PC7
ความหมาย	I/O	I/O	I/O	$INTR_A$	STB_A	IBF_A	ACK_A	OBF_A

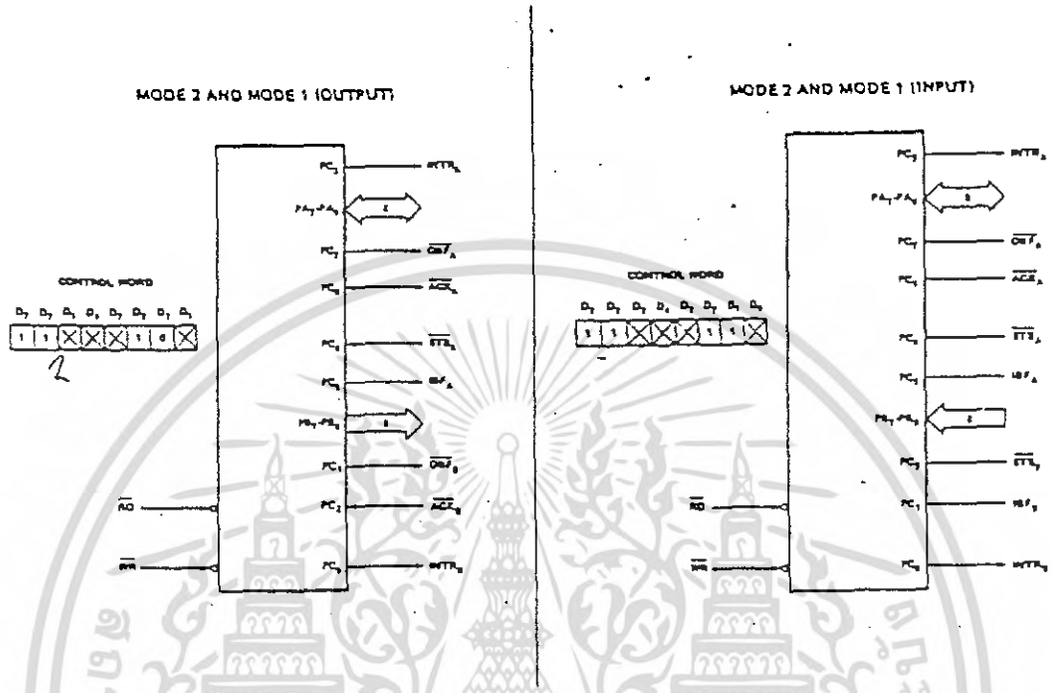
รูปแสดงถึงพอร์ท C ที่ถูกใช้เป็นสัญญาณควบคุมในโหมด 2

ส่วนสัญญาณ INTR นั้นเราสามารถกำหนดให้มีการเกิดอินเตอร์รัฟท์หรือไม่ก็ได้ โดยการเซท/รีเซทบิท ดังนี้ กรณี อินพุท สามารถทำการ เซท/รีเซท ได้ที่บิท PC4 (STB)

กรณี เอ้าพุท สามารถทำการ เซท/รีเซท ได้ที่บิท PC6 (ACK)

เราสามารถโปรแกรมให้พอร์ท A ในโหมด 2 ทำงานร่วมกับพอร์ท B ซึ่งเป็นโหมด 0,1ก็ได้ ดังนี้

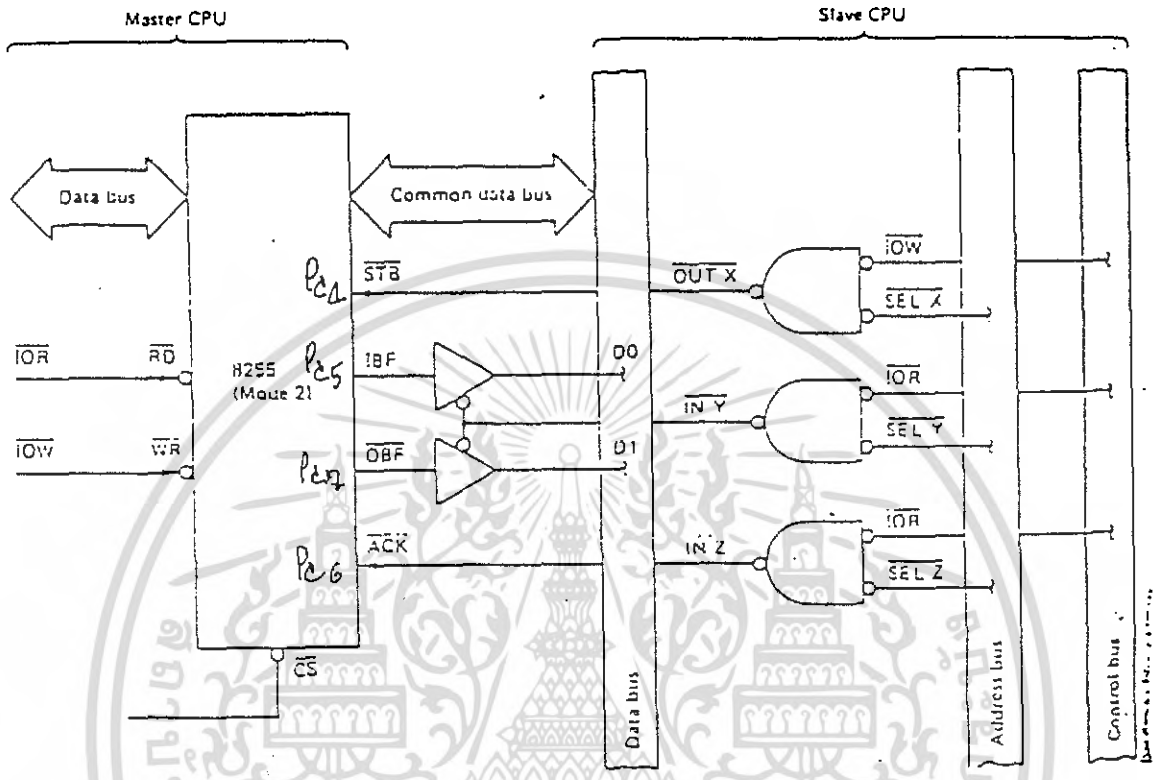




รูปแสดงการโปรแกรมโหมด 2 ร่วมกับโหมด 0,1

<u>ตัวอย่างการใช้งานในโหมด 2</u>

สมมุติเรามี ไมโครโปรเซสเซอร์บอร์ดอยู่ 2 ชุด โดยในบอร์ดที่หนึ่งมี 8255 อยู่ขอเรียกบอร์ดนี้ว่า มาสเตอร์บอร์ด (Master board) ส่วนในบอร์ดที่ 2 เป็นบอร์ดที่อาจไม่มี 8255 อยู่ก็ได้ซึ่งจะขอเรียกว่า สเลฟบอร์ด (Slave board) หากเราต้องการส่งข้อมูลไปมาระหว่าง 2 บอร์ดนี้โดยอาจส่งข้อมูลจาก มาสเตอร์บอร์ดไปสู่สเลฟบอร์ดหรือในทางกลับกันเราจะส่งข้อมูลจากสเลฟบอร์ดไปสู่มาสเตอร์บอร์ดไปสู่สเลฟบอร์ดหรือในทางกลับกันเราจะส่งข้อมูลจากสเลฟบอร์ดไปสู่มาสเตอร์บอร์ด ก็สามารถเขียนโปรแกรมการควบคุมได้โดยไม่ยาก โดยบอร์ดทั้งสองมีการติดต่อถึงกันดังแสดงในรูปตัวอย่างรูปต่อไปนี้



รูปแสดงตัวอย่างการส่งสายสัญญาณเพื่อส่งข้อมูลไปมาระหว่างไมโครโปรเซสเซอร์

จากรูป เมื่อเราโปรแกรมให้ 8255 ทำงานในโหมด 2 แล้ว พอร์ท A ก็จะเป็นพอร์ทส่งรับข้อมูล 2 ทิศทาง ซึ่งถูกต่อโดยตรงกับสาย DATA BUS ของ สเลฟบอร์ดและ IBF,OBF ถูกต่อเข้าสู่

บิท DO,DO1 ของสเลฟบอร์ด โดยสามารถอ่านค่าได้เมื่อสเลฟ อินพุทพอร์ทมาที่ INY

<u>มาสเตอร์ส่งข้อมูลให้สเลฟ</u> มาสเตอร์บอร์ดจะตรวจดูที่พอร์ท C ขาสัญญาณ OBF ก่อนว่าเป็น 1 หรือไม่หากเป็น 1 แสดงว่าพอร์ท A ว่างก็จะส่งข้อมูลมาสู่พอร์ท A เป็นผลทำให้ OBF เป็นโลจิก 0 ซึ่งทางสเลฟบอร์ดก็สามารถทราบได้โดยการอินพุทพอร์ท INY เข้ามาดูที่บิท D1 (OBF) เมื่อทราบว่าเป็น 0 ก็จะส่งสัญญาณเข้าพอร์ท INZ เกิดสัญญาณ ACK ไปรับเอาข้อมูลมาเก็บที่ สเลฟได้ และ OBF ก็จะกลับเป็น 1 ใหม่เพื่อรอให้มาสเตอร์ส่งข้อมูลมาที่ 8255 อีก

<u>สเลฟส่งข้อมูลให้มาสเตอร์</u> สเลฟบอร์ดก็จะอ่านข้อมูลอินพุทพอร์ท INY เพื่อตรวจดูว่า IBF เป็น 0 หรือไม่(ว่างหรือไม่) หากเป็น 0 ก็สามารถส่งข้อมูลออกไปและจะสั่งเอ้าพอร์ท INX ออกไปด้วยเพื่อให้เป็นสัญญาณ STB ให้ 8255 ทำการแลทช์ข้อมูลนั้นไว้ในพอร์ท A เป็นผลทำให้ IBF เป็น 1 ซึ่งหาก มาสเตอร์ตรวจที่สัญญาณ IBF นี้ดูก็จะทราบว่ามีข้อมูลจากสเลฟมา ก็จะทำการส่งสัญญาณการอ่าน RD มาอ่านเอาข้อมูลไปเก็บไว้ ทำให้ IBF เป็น 0 อีกครั้งเพื่อรอการรับข้อมูลจากสเลฟเข้ามาใหม่ซึ่งสเลฟบอร์ดสามารถส่งข้อมูลตัวต่อๆไปเข้ามาได้

จากตัวอย่างนี้เราสามารถนำไปประยุกต์ทำเป็น PRINT BUFFER ได้หรือใช้กับการส่งข้อมูลมาตรฐานบางประการ เช่น IEEE 488

<u>คุณสมบัติทางไฟฟ้า</u>

8255 มีข้อกำหนดของกระแสเอ้าพุท เมื่อเป็น LOW (IO_L) ที่กระแส 1.7 MA และรับกระแสเอ้าพุทเมื่อเป็น HIGH (IO_H) WFH 200 MA ฉะนั้นหากนำไปใช้กับ อุปกรณ์เช่น LED ซึ่งจะต้องใช้กระแสประมาณ 20–40 MA แล้วก็ต้องนำ IC TTL มาใช้ร่วมด้วยดังรูป

รูปแสดงการเพิ่มกระแสเมื่อมีการขับอุปกรณ์ที่ต้องการกระแสนั้นมากขึ้น

คุณสมบัติของ บอร์ด ANT-32

- เป็นบอร์ดคอนโทรลใช้กับไมโครคอนโทรลเลอร์ในตระกูล MCS-51 (8031/8032) ใช้ CPU เบอร์ 80C32 ทำงานที่ความถี่สัญญาณนาฬิกา 11.0592 MHz
- ใช้งานหน่วยความจำบนบอร์ดได้ 3 ตำแหน่งด้วยกัน คือ

  U2 เป็นหน่วยความจำโปรแกรม (PROGRAM MEMORY) ใช้กับ EPROM ขนาด 8-32KByte เบอร์ 2764, 27128 หรือ 27256

  U3 เป็นหน่วยความจำข้อมูล (DATA MEMORY) ใช้กับ RAM ขนาด 8KByte เบอร์ 6264 หรือ 32KByte เบอร์ 62256 สามารถแบคอัพข้อมูลได้โดยใช้แบตเตอรี่ลิเธี่ยม

  U4 เป็นหน่วยความจำโปรแกรมและข้อมูล (PROGRAM AND DATA MEMORY) ใช้กับ EPROM, RAM หรือ EEPROM ขนาด 8-32 KByte โดยใช้ EPROM เบอร์ 2764, 27256 ใช้ RAM เบอร์ 6264, 62256 หรือ EEPROM เบอร์ 2864(A), 28256(A)
- มีพอร์ต I/O เบอร์ 8255 จำนวน 2 ตัว (48 บิท) สำหรับต่อไปใช้งานภายนอก
- มีพอร์ต LCD สำหรับการต่อใช้งานกับ LCD แบบ DOT MATRIX
- มีวงจร SERIAL INTERFACE DRIVER RS232 ด้วยชิพเบอร์ MAX232 สำหรับการต่อเข้ากับเครื่องไมโครคอมพิวเตอร์
- มีวงจร Watchdog Timer, Powerup/down Reset ด้วยชิพเบอร์ MAX691
- มีวงจร RTC (Real Time Clock) ใช้ชิพเบอร์ DS1202
- มีคอนเน็คเตอร์สำหรับ PORT 1 ของไมโครคอนโทรลเลอร์โดยเฉพาะ
- มีคอนเน็คเตอร์สำหรับ SYSTEM BUS ทำให้ขยายระบบได้ง่าย และสามารถใช้กับบอร์ดขยายต่าง ๆ ที่จะมีขึ้นในอนาคต
- สามารถเลือกเบอร์และชนิดหน่วยความจำ หรือกำหนดคุณสมบัติต่าง ๆ ของบอร์ดได้ด้วยจัมพ์เปอร์
- สามารถพัฒนาโปรแกรมได้ทั้งภาษาเบสิค และแอสเซมบลี้ โดยใช้ซอฟท์แวร์ BASIC32 และ REM31

แนวทางในการพัฒนาโปรแกรม

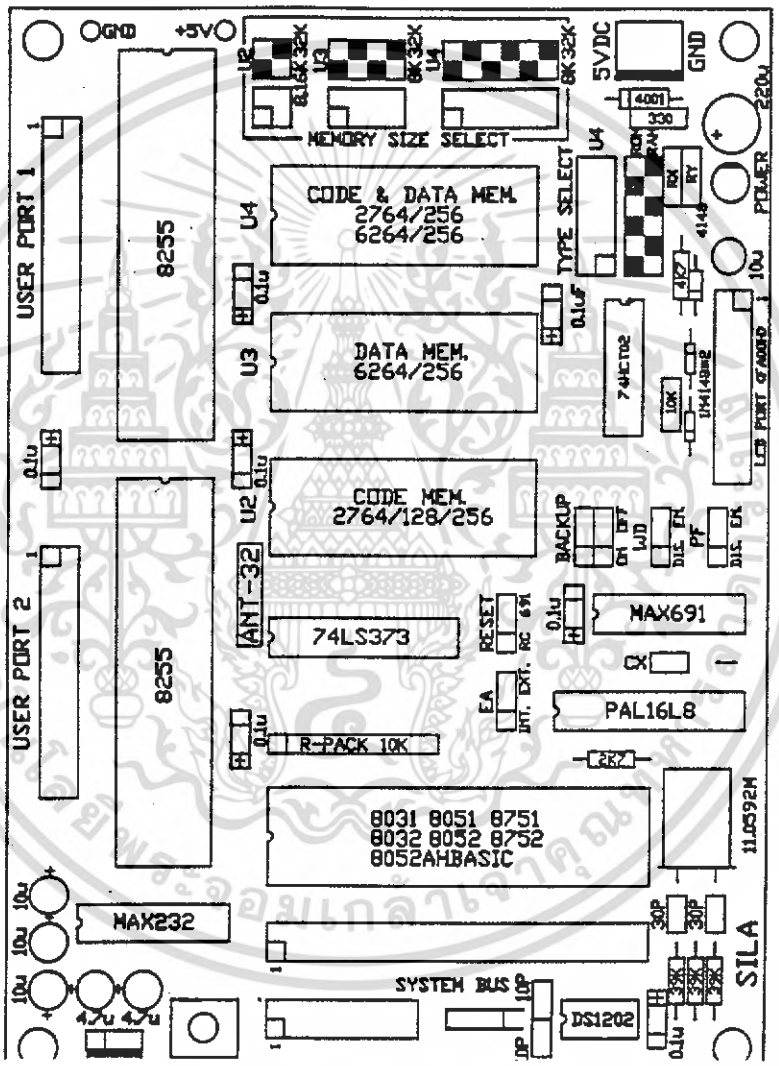
การใช้งานบอร์ด ANT-32 ผู้ใช้จำเป็นต้องเขียนโปรแกรมควบคุมที่เรียกกันว่า มอนิเตอร์โปรแกรมขึ้นมาโดยเฉพาะเพื่อทำให้งานที่ต้องการพัฒนาสำเร็จได้ ในขั้นตอนการพัฒนานี้เองที่เป็นจุดเด่นของ ANT-32 โดยมีโปรแกรมให้เลือก 2 ลักษณะด้วยกัน คือ REM31 และ BASIC32 หลักการของ ทั้งสองโปรแกรมก็คือให้ผู้ใช้นำ EPROM ที่บรรจุโปรแกรมนี้ไปเสียบลงบนบอร์ด ANT-32 ที่ตำแหน่ง หน่วยความจำ U2 (EPROM) แล้วทำการต่อสาย SERIAL PORT ระหว่างบอร์ด ANT-32 กับ เครื่องไมโครคอมพิวเตอร์ (PC, XT, AT, PS/2) จากนั้นที่เครื่อง PC ให้ใช้โปรแกรมสำหรับ การสื่อสารข้อมูลอนุกรม (ให้มาพร้อม REM31 และ BASIC32) ผู้ใช้จะสามารถติดต่อกับบอร์ด ANT-32 ได้ ตามลักษณะของโปรแกรมที่ใช้ดังนี้

REM31_ (8031 REMOTE MONITOR) ใช้พัฒนาโปรแกรมด้วยภาษาแอสเซมบลี้ด้วย REM31 ผู้ใช้จะมีชุดคำสั่งในการพัฒนาโปรแกรมถึง 19 คำสั่ง ลักษณะคำสั่งจะคล้ายคลึงกับคำสั่ง DEBUG ของ DOS ทำให้ผู้ที่คุ้นเคยอยู่ก่อนแล้ว จะใช้งานได้ง่ายขึ้น REM31 ใช้กับ CPU ได้ทั้งเบอร์ 8031 และ 8032

BASIC32 (8032 BASIC INTERPRETER) ใช้พัฒนาโปรแกรมด้วยภาษาเบสิคกับ CPU 8032 ภาษาเบสิคตัวนี้ก็คือตัวเดียวกับ BASIC-52 ของ INTEL นั่นเอง โดย BASIC32 นี้ยังได้เปลี่ยนแปลงและเพิ่มเติมคำสั่งใหม่เข้าไปเพื่อให้เหมาะกับบอร์ด ANT-32 ยิ่งขึ้น

และในกรณีที่ผู้ใช้มี EPROM EMULATOR (EE-232) ก็สามารถใช้พัฒนาโปรแกรมได้ทั้งภาษาแอสแซมบลี้โดยใช้โปรแกรม 8031 ASSEMBLER หรือภาษาซีโดยใช้โปรแกรม 8031 C COMPILER ซึ่งทั้งสองโปรแกรมจำเป็นต้องใช้เครื่อง PC ช่วยในการพัฒนาโปรแกรมด้วยเช่นกัน

ภาพแสดงบอร์ดและตำแหน่งจัมพ์เปอร์



ก่อนการใช้งานบอร์ด ANT-32 ผู้ใช้จำเป็นต้องเลือกขนาด, เบอร์ของหน่วยความจำและกำหนดคุณสมบัติต่าง ๆ ของบอร์ดให้ถูกต้องด้วยจัมพ์เปอร์ ซึ่งมีทั้งหมด 9 ชุดด้วยกัน

จัมพ์เปอร์ EA	สำหรับเลือกใช้หน่วยความจำโปรแกรม (PROGRAM MEMORY) ตำแหน่งแอดเดรสเริ่มต้น OOOOH ใน INT. (INTERNAL) หรือ EXT. (EXTERNAL)
จัมพ์เปอร์ RESET	สำหรับเลือกสัญญาณรีเซท CPU จากวงจร RC หรือ MAX691
จัมพ์เปอร์ U2 SIZE	สำหรับเลือกขนาดหน่วยความจำโปรแกรม U2 (EPROM) เป็น 8,16KYyte(2764,27128) หรือ 32KByte(27256)
จัมพ์เปอร์ U3 SIZE	สำหรับเลือกขนาดหน่วยความจำข้อมูล U3 (RAM) เป็น 8KYyte (6264) หรือ 32KYyte(62256)
จัมพ์เปอร์ U4 TIPE	สำหรับเลือกชนิดหน่วยความจำ U4 เป็น EPROM (หน่วยความจำโปรแกรม) หรือ RAM (หน่วยความจำข้อมูล)
จัมพ์เปอร์ U4 SIZE	สำหรับเลือกขนาดหน่วยความจำโปรแกรมและข้อมูล U4 เป็น 8KYyte(XX64) หรือ 32KByte (XX256)
จัมพ์เปอร์ BACKUP	สำหรับเลือก ON/OFF การสำรองข้อมูล (MAX691) ของ U3
จัมพ์เปอร์ WD	สำหรับเลือก EN.(ENABLE)/DIS.(DISABLE) วงจร WATCHDOG (MAX691)
จัมพ์เปอร์ PE	สำหรับเลือก EN.(ENABLE)/DIS.(DISABLE) วงจร POWER FAIL DETECTOR (MAX691)

* จัมพ์เปอร์ BACKUP, WD และ PF จะใช้งานได้ก็ต่อเมื่อเสียบใช้งานชิพ MAX691 ด้วย

การปรับจัมพ์เปอร์เลือกหน่วยความจำ

ไมโครคอนโทรลเลอร์ตระกูล 8031(32) สามารถต่อกับหน่วยความจำภายนอกได้ถึง 128 KYyte โดยแบ่งออกเป็นสองส่วนคือ 64 KByte เป็นหน่วยความจำโปรแกรม (PROGRAM MEMORY) และอีก 64 KByte เป็นหน่วยความจำข้อมูล (DATA MEMORY) ซึ่งหน่วยความจำทั้งสองส่วนนี้มีตำแหน่งแอดเดรสที่ OOOOH-FFFFH เหมือนกัน แต่จะถูกแยกออกจากกันด้วยสัญญาณควบคุมที่ต่างกัน โดยสัญญาณ $PSEN^R$ ใช้ควบคุมในการอ่านหน่วยความจำโปรแกรม (EPROM) สัญญาณ RD^R และ WR^R ใช้ควบคุมการอ่านและเขียนหน่วยความจำข้อมูลและพอร์ทอินพุท/เอาท์พุท และสำหรับการอ่านหน่วยความจำโปรแกรมและข้อมูล (PROGRAM AND DATA MEMORY) ใช้สัญญาณ $GET\_^R\_$ ซึ่งสัญญาณนี้ได้จากการ AND สัญญาณ $PSEN\_^R\_$ และ RD^R หน่วยความจำส่วนนี้สามารถใช้ได้กับ EPROM หรือ RAM

สำหรับบอร์ด ANT-32 ได้จัดหน่วยความจำออกเป็น 3 ส่วนด้วยกันคือ

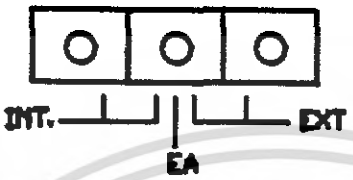
U2 เป็นหน่วยความจำโปรแกรม (PROGRAM MEMORY) แอดเดรส OOOOH-7FFFH

U3 เป็นหน่วยความจำข้อมูล (DATA MEMORY) แอดเดรส OOOOH-7FFFH

U4 เป็นหน่วยความจำโปรแกรมและข้อมูล (PROGRAM AND DATA MEMORY) แอดเดรส 8OOOH-F7FFH

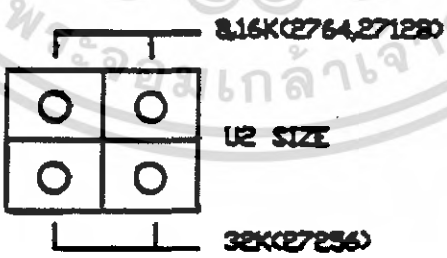
ส่วนแอดเดรส F8OOH-FFFFH ใช้เป็นตำแหน่งของพอร์ทอินพุท/เอาท์พุท หน่วยความจำ U2, U3 และ U4 สามารถเลือกขนาด (SIZE) และชนิด (TYPE) ได้ด้วยจัมพ์เปอร์ดังต่อไปนี้

การเลือกใช้หน่วยความจำโปรแกรมภายใน/ภายนอก (จัมพ์เปอร์ EA)



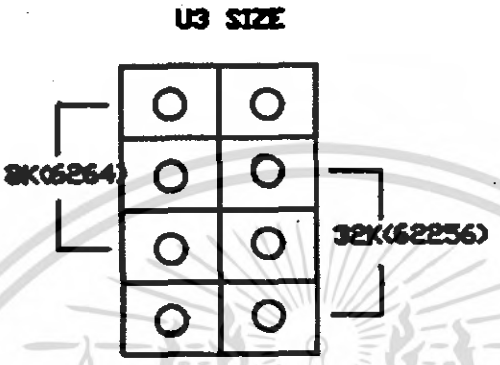
ในการใช้งานหน่วยความจำโปรแกรมที่แอดเดรสเริ่มต้น OOOOH สามารถเลือกใช้หน่วยความจำส่วนนี้ได้ด้วยจัมพ์เปอร์ EA ทั้งแบบใช้หน่วยความจำโปรแกรมภายใน (INTERNAL PROGRAM MEMORY) สำหรับ CPU เบอร์ 8051, 8052, 8751, 8752, 8052AHBASI โดยปรับจัมพ์เปอร์นี้ที่ตำแหน่ง INT.(INTERNAL) หรือใช้หน่วยความจำโปรแกรมภายนอก (EXTERNAL PROGRAM MEMORY) สำหรับ CPU เบอร์ 8031, 8032 ปรับจัมพ์เปอร์นี้ไปตำแหน่ง EXT. (EXTERNAL)

การเลือกขนาดหน่วยความจำ U2 (จัมพ์เปอร์ U2 SIZE)



จัมพ์เปอร์ U2 SIZE สำหรับเลือกขนาดหน่วยความจำโปรแกรม (PROGRAM MEMORY ตำแหน่ง U2 แอดเดรสเริ่มต้นที่ OOOOH เป็น EPROM ใช้ได้ 3 ขนาดคือ 8,16 KByte (เบอร์ 2764, 27128) หรือ 32 KByte (เบอร์ 27256)

การเลือกขนาดหน่วยความจำ U3 (จัมพ์เปอร์ U3 SIZE)

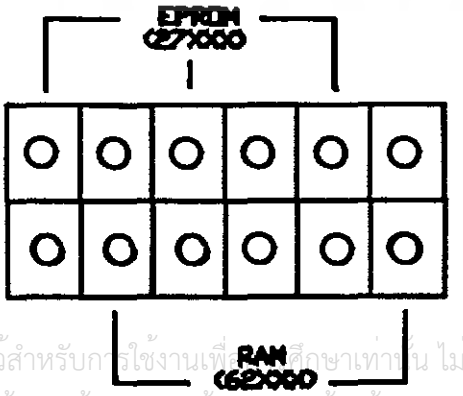


จัมพ์เปอร์ U3 SIZE สำหรับเลือกขนาดหน่วยความจำข้อมูล (DATA MEMORY) ตำแหน่ง U3 แอดเดรสเริ่มต้นที่ OOOOH เป็น RAM ใช้ได้ 2 ขนาดคือ 8KByte (เบอร์ 62640 หรือ 32 KByte (เบอร์ 62256)

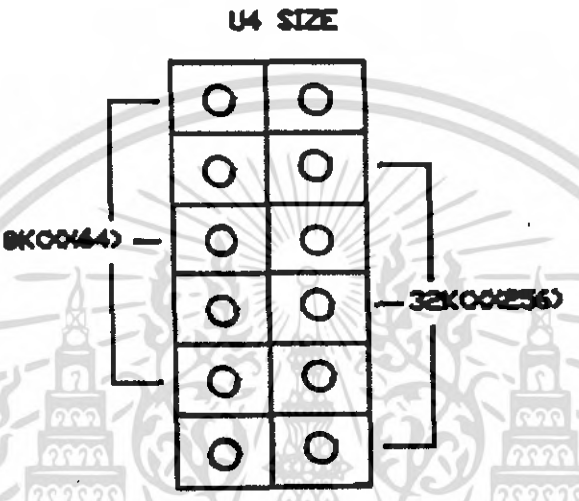
* หน่วยความจำ U3 (RAM) นี้สามารถสำรองข้อมูลในช่วงไฟดับ (BACKUP) ได้โดยบนบอร์ด ANT-32 ต้องมีชิพ MAX691 แบตเตอรี่ลิเธี่ยม และจัมพ์เปอร์ BACKUP ต้องอยู่ที่ตำแหน่ง ON*

การเลือกชนิดหน่วยความจำ U4 (จัมพ์เปอร์ U4 TYPE)

จัมพ์เปอร์ U4 TYPE สำหรับเลือกชนิดหน่วยความจำโปรแกรมและข้อมูล (PROGRAM AND DATA MEMORY) ตำแหน่ง U4 แอดเดรสเริ่มต้นที่ 8OOOH เป็น EPPROM (27XXX) หรือ RAM (62XXX) และขนาดของ U4 เลือกได้ด้วยจัมพ์เปอร์ U4 SIZE



การเลือกขนาดหน่วยความจำ U4 (จัมพ์เปอร์ U4 SIZE)



จัมพ์เปอร์ U4 SIZE สำหรับเลือกขนาดหน่วยความจำโปรแกรมและข้อมูล (PROGRAM AND DATA MEMORY) ตำแหน่ง U4 (เลือกเป็น EPROM หรือ RAM ด้วยจัมพ์เปอร์ U4) ได้ 2 ขนาดคือ 8 KByte (EPROM เบอร์ 2764, RAM เบอร์ 6264) หรือ 32 KByte (EPROM เบอร์ 24256, RAM เบอร์ 62256)

การใช้งานหน่วยความจำขนาด 32 KByte ที่ตำแหน่ง U4 นี้จะสามารถใช้ได้ 30 KByte (แอดเดรส 8000H-F7FFH) เท่านั้น

TTL I/O (8255)

8255 Programmable Peripheral Interface (PPI) เป็นชิพพอร์ทแบบขนานที่เป็นที่นิยมใช้งานกันมากมาย สำหรับบอร์ด ANT-32 ใช้พอร์ท 8255 จำนวน 2 ตัวทำหน้าที่เป็นพอร์ททำให้มีพอร์ทอินพุท/เอาท์พุทถึง 24 X 2 = 48 บิท โดยแบ่งเป็น USER PORT 1 และ 2 มีตำแหน่งแอดเดรสดังนี้

USER PORT 1 (U10) แอดเดรส F800H = 8255 offset addr = actual addr

Port A ตำแหน่งแอดเดรส F800H + 00H = F800H

Port B ตำแหน่งแอดเดรส F800H + 01H = F801H

Port C ตำแหน่งแอดเดรส F800H + 02H = F802H

Mode Port ตำแหน่งแอดเดรส F800H + 03H = F803H

USER PORT 2 (U11) แอดเดรส FC00H = 8255 offset addr = actual addr

Port A ตำแหน่งแอดเดรส FC00H + 00h = FC00H

Port B ตำแหน่งแอดเดรส FC00H + 01H = FC01H

Port C ตำแหน่งแอดเดรส FC00H + 02H = FC02H

Mode Port ตำแหน่งแอดเดรส FC00H + 03H = FC03H

ก่อนที่จะใช้งานพอร์ท 8255 ผู้ใช้ต้องทำการกำหนดโหมดการทำงาน (configulation) ของพอร์ท A, B และ C ให้เป็นพอร์ทอินพุทหรือเอาท์พุท โดยทำการเขียนค่า control code ไปที่ Mode Port ซึ่ง Mode Port นี้สามารถเขียนได้เท่านั้นไม่สามารถอ่านได้ ในที่นี้จะกล่าวเฉพาะการทำงานในโหมด 0 ซึ่งเป็นโหมดที่ใช้งานได้สะดวกและง่ายต่อการทำความเข้าใจ ดังแสดงค่า control code ดังตารางที่ 1

ตารางที่ 1 8255 MODE O CONFIGULATION

PORT A (PA0-PA7)	PORT C บน (PC4-PC7)	PORT B (PB0-PB7)	PORT C ล่าง (PC0-PC3)	CONTROL CODE (HEX)
OUTPUT	OUTPUT	OUTPUT	OUTPUT	80H
OUTPUT	OUTPUT	OUTPUT	INPUT	81H
OUTPUT	OUTPUT	INPUT	OUTPUT	82H
OUTPUT	OUTPUT	INPUT	INPUT	83H
OUTPUT	INPUT	OUTPUT	OUTPUT	88H
OUTPUT	INPUT	OUTPUT	INPUT	89H
OUTPUT	INPUT	INPUT	OUTPUT	8AH
OUTPUT	INPUT	INPUT	INPUT	8BH
INPUT	OUTPUT	OUTPUT	OUTPUT	90H
INPUT	OUTPUT	OUTPUT	INPUT	91H
INPUT	OUTPUT	INPUT	OUTPUT	92H
INPUT	OUTPUT	INPUT	INPUT	93H
INPUT	INPUT	OUTPUT	OUTPUT	98H
INPUT	INPUT	OUTPUT	INPUT	99H
INPUT	INPUT	INPUT	OUTPUT	9AH
INPUT	INPUT	INPUT	INPUT	9BH

DEMO TTL I/O PROGRAM

```
100 REM TTLIO.BAS
110 REM TEST 8255 PORT ON ANT-32 V2.0
120 REM SILA RESEARCH CO.,LTD.
130 PA=0F800H : PB=0F801H : PC=0F802H : PPP-0F803H
140 QA=0FC00H : QB=0FC01H : QC=0FC02H : QP=0FC03H
150 XBY (PP)=80H : XBY (QP)=80H : REM SET CONTROL COD
160 FOR M=101 TO 1 STEP -25
170 FOR I=0 TO 7
180 J=2**I : K=2** (7-I)
190 XBY (PA)=J : XBY (PB)=K : XBY (PC)=J
200 XBY (QA)=K : XBY (QB)=J : XBY (QC)=K
210 FOR L=0 TO M : NEXT L
220 NEXT I
230 FOR I=7 TO 0 STEP -1
240 J=2**I : K=2** (7-I)
250 XBY (PA)=J : XBY (PB)=K : XBY (PC)=J
260 XBY (QA)=K : XBY (QB)=J : XBY (QC)=K
270 FOR L=0 TO M : NEXT L
280 NEXT I
290 NEXT M
300 GOTO 160
```

```
FILENAME      TTLIO.ASM
DESCRIPTION   TEST 8255 USER PORT 1 & 2
HARDWARE      ANT-32 VERSION 2.0 BOARD
ASSEMBLER     SXA51 ON PC-DOS
COMPANY       SILA RESEARCH CO.,LTD.

        ORG  8100H  ; STARTING ADDR FOR REM31
********** VARIABLE SET **********

                        ;** USER PORT 1 **
P1A     EQU     0F800H          ;PORT A
P1B     EQU     0F801H          ;PORT B
P1C     EQU     0F802H          ;PORT C
P1P     EQU     0F803H          ;MODE PORT
                        ;** USER PORT 2 **
P2A     EQU     0FC00H          ;PORT A
P2B     EQU     0FC01H          ;PORT B
P2C     EQU     0FC02H          ;PORT C
P2P     EQU     0FC03H          ;MODE PORT
********** TTLIO MAIN **********
TLIO :  MOV     A,#80H          ;SET CONTROL CODE
        MOV     DPTR, #UPIP     ;USER PORT1
        MOVX    @DPTR,A         ;PA, PB, PC=OUTPUT
        MOV     DPTR, #UP2P     ;USER PORT2
        MOVX    @DPTR, A        ;PA, PB, PC=OUTPUT

TLIO1 : MOV     R2, #HIGH 10001 ;R2R3 DELAY
        MOV     R3, #LOW 10001
        MOV     R6, #01H        ;DEFAULT BIT PATTERN 1
        MOV     R7, #80H        ;
```

```
        MOV     B, #8           ;LOOP=8
TLIO2 : MOV     A, R6
        MOV     DPTR, #UPP1A
        MOVX    @DPTR, A
        MOV     DPTR, #UP1C
        MOVX    @DPTR,A
        MOV     DPTR, #UP2B
        MOVX    @DPTR, A
        RL      A               :ROTATE LEFT
        MOV     R6,A
        MOV     A, R7
        MOV     DPTR, #UP1B
        MOVX    @DPTR,A
        MOV     DPTR, #UP2A
        MOVX    @DPTR, A
        MOV     DPTR, #UP2C
        MOVX    @DPTR, A
        RR      A               :ROTATE RIGHT
        MOV     R7, A
        LCALL   DELAY           ;DELAY
        DJNZ    B, TTLIO2

        MOV     R6,#80H         ;DEFAULT BIT PATTERN
        MOV     R7, #01H
        MOV     B, #8           :LOOP=8
```

```
TTLIO3: MOV     A, R6
        MOV     DPTR, #UP1A
        MOVX    @DPTR,A
        MOV     DPTR, #UP1C
        MOVX    @DPTR, A
        MOV     DPTR, #UP2B
        MOVX    @DPTR,A
        RR      A               ;ROTATE RIGHT
        MOV     R6,A
        MOV     A, R7
        MOV     DPTR, #UP1B
        MOVX    @DPTR, A
        MOV     DPTR, #UP2A
        MOVX    @DPTR,A
        MOV     DPTR, #UP2C
        MOVX    @DPTR, A
        RL      A               ;ROTATE LEFT
        MOV     R7,A
        LCALL   DELAY           ;DELAY
        DJNZ    B, TTLIO3
        SJMP    TTLIO1
```

```
: ********** DELAY SUB. **********
DELAY  : MOV      A, R2
         MOV      R0, A
         MOV      A, R3
         MOV      R1, A
DELAY1 : XCH      A, R1          ;ROR1=ROR1-1
         JNZ      DELAY2
         DEC      R0
DELAY2 : DEC      A
         XCH      A, R1
         MOV      A, R0
         ORL      A, R1
         JNZ      DELAY1
         RET
         END
```

LCD PORT

บอร์ด จะมี ให้พร้อมสำหรับการต่อใช้งานโดยสามารถต่อเข้ากับ
แบบ ได้ทันที ซึ่งจะใช้ขาสัญญาณทั้งหมด 14 ขา และสำหรับการใช้
งาน นั้น จะมีการจัดวางจรในแบบ ซึ่งจะช่วยให้การเขียนโปร
แกรมทำได้ง่าย โดยจะมองเห็นตำแหน่งต่าง ๆ ที่สรุปได้ดังนี้

ADDRESS	ลักษณะของ PORT ที่ติดต่อ
FA00H	สำหรับเขียนคำสั่ง (RS=0 R/W=0)
FA01H	สำหรับอ่านค่า BUSY (RS=0 R/W=1)
FA02H	สำหรับเขียนข้อมูล (RS=1 R/W=0)
FA03H	สำหรับอ่านข้อมูล (RS=1 R/W=1)

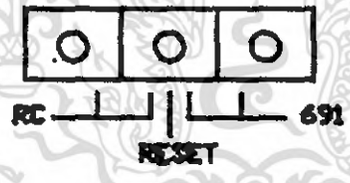
การอ่านค่า LCD แบบ DOT MATRIX นี้ จะสามารถเลือกรุ่นใด ๆ ก็ได้ โดยจะมีจำนวนตัว
อักษรต่อบรรทัด และจำนวนบรรทัดตามที่ต้องการ เพราะสายสัญญาณที่ใช้จะใช้แบบเดียวกัน
หมด จะแตกต่างกันก็ที่โปรแกรมเท่านั้น การนับหมายเลขขั้วต่อของ LCD PORT จะไม่
เหมือน การนับทั่ว ๆ ไป จึงควรดูให้แน่ใจก่อนการต่อใช้งาน (ลักษณะการจัดขาดูได้จากภาพ
และ CONNECTER ท้ายเล่ม) อีกประการหนึ่ง หมายเลขตัวต่อที่ด้าน LCD ก็มักจะมีหลาย
แบบ อาจจะเป็นแถวคู่หรือแถวเดี่ยวก็ได้ แต่ทั้งนี้หมายเลข 1-14 ของขาสัญญาณก็จะตรงกัน
หมด กล่าวคือต่อหมายเลขให้ตรงเป็นใช้ได้ รายละเอียดของการใช้งาน ตัว LCD นี้ ให้อ่าน
เพิ่มเติมได้จากคู่มือของ LCD อีกที

OPTIONAL MICROPROCESSOR SUPERVISORY CIRCUITS

MAX691 Microprocessor Supervisory Circuits เป็นชิพของ MAXIM ใช้สำหรับจัดการเกี่ยวกับ การมอนิเตอร์เพาเวอร์ซัพพลาย (power supply monitoring และการควบคุมแบตเตอรี่ (battery control) ซึ่งเป็นวงจรในส่วนที่มีความสำคัญอย่างยิ่งกับระบบไมโครโปรเซสเซอร์ อันได้แก่ Microprocessor Reset, Power Fail Detector, CMOS RAM Write Protection และ Watchdog Timer ในที่นี้จะขอกล่าวถึง รายละเอียดในส่วนต่าง ๆ ของ MAX691 (U8) เท่าที่จำเป็นต่อการใช้งานกับบอร์ด ANT-32 เท่านั้น

Microprocessor Reset (จัมพ์เปอร์ RESET)

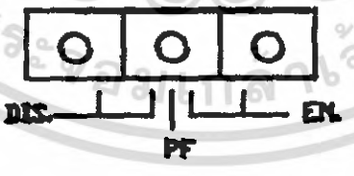
สัญญาณรีเซท CPU บนบอร์ด ANT-32 สามารถเลือกใช้ได้จาก 2 แหล่งคือ RC (จากวงจร R/C รีเซท) หรือ 691 (จากวงจรรีเซทของ MAX691) โดยจัมพ์เปอร์ RESET



ถ้าไม่ได้ใช้งาน MAX691 ให้ปรับจัมพ์เปอร์ไปที่ตำแหน่ง RC วงจร R/C รีเซทจะทำการรีเซท CPU ในช่วง power up เท่านั้น ส่วนสัญญาณรีเซท CPU ของ MAX691 จะทำการรีเซท CPU ในช่วง power up และ power down โดยที่ขา RESET จะเป็นลอจิก "1" เมื่อแรงดัน Vcc ตกลงต่ำกว่า 4.5 โวลท์และจะเป็นลอจิก "0" หลังจากแรงดัน Vcc สูงกว่า 4.75 โวลท์ประมาณ 50 มิลลิเซคคั่น (ms) ซึ่งก็หมายความว่า CPU จะถูกรีเซทเมื่อเริ่มจ่ายไฟด้วยพัลส์ที่มีความกว้าง 50 ms และจะถูกรีเซทอีกครั้งเมื่อไฟตก นอกจากนี้แล้วขา RESET จะถูกใช้เมื่อเลือกใช้งาน Watchdog Timer ซึ่งจะได้กล่าวถึงในหัวข้อถัดไป

Power Fail Detector จัมพ์เปอร์ PF)

วงจรตรวจจับแรงดันไฟตก (Power Fail Detector สำหรับระบบไมโครฯ นับว่ามีความสำคัญมากกับระบบที่ต้องการเก็บค่าพารามิเตอร์หรือข้อมูลบางอย่างลง RAM ก่อนที่ระบบจะหยุดทำงาน เพื่อที่ว่าเมื่อระบบเริ่มทำงานใหม่อีกครั้งจะได้นำเอาข้อมูลที่เก็บไว้ก่อนไฟดับมาประมวลผลเพื่อใช้งานต่อไป โครงสร้างภายในของวงจรนี้เป็นวงจร voltage comparator โดยรับแรงดันอินพุทที่ต้องการตรวจสอบจากภายนอกเข้าที่ขา PFI (Power Fail Input) นำมาเปรียบเทียบกับแรงดันอ้างอิง (reference voltage) 1.3 โวลท์ ซึ่งขาเอาท์พุทคือ ขา PFO (Power Fail Output) จะเป็นลอจิก "0" เมื่อแรงดันที่ขา PFI ต่ำกว่า 1.3 โวลท์ ขา PFI รับแรงดันมาจากวงจร voltage divider ภายนอก ซึ่งก็คือ Rx และ Ry โดยทำการตรวจจับแรงดัน Vcc 5 โวลท์ ค่าอัตราส่วนของวงจร voltage divider สามารถกำหนดได้จากหลักการที่ว่าแรงดันที่ขา PFI จะตกลงถึงค่า 1.3 โวลท์ก่อนที่แรงดัน +5 โวลท์จากแหล่งจ่ายไฟจะตกลงถึง 4.75 โวลท์ โดยปกติแล้วขา PFO จะต่อเข้ากับขาอินเทอร์รัพท์ของ CPU เพื่อที่ว่าเมื่อเกิดสภาวะไฟตก CPU จะถูกอินเทอร์รัพท์เพื่อให้ CPU ไปทำขบวนการนำเอาข้อมูลที่จำเป็นเก็บลง RAM ก่อนที่แรงดัน Vcc จะตกลงต่ำกว่า 4.75 โวลท์และที่แรงดันนี้เองที่ CPU จะถูกรีเซทอีกครั้ง (power down reset)



ขา PFO ของ MAX691 จะต่อเข้าขา INT1 ของ CPU สามารถเลือกใช้งานวงจร Power Fail Detector นี้ได้ด้วยจัมพ์เปอร์ PF โดยปรับไปที่ EN. (ebabke) หรือไม่ใช้งานโดยปรับไปที่ DIS. (disable)

การคำนวณหาค่า Rx, Ry

กำหนดค่าแรงดัน Vcc ในขณะไฟตกในช่วงที่ CPU ยังสามารถทำงานได้ที่ 4.8 โวลท์ และเพื่อให้ง่ายต่อการคำนวณจึงกำหนดค่า Ry = 10K สามารถคำนวณหาค่า ได้ดังนี้

Rx = (4.8-1.3 / 1.3/Ry)

= 3.5 / (1.3/10K)

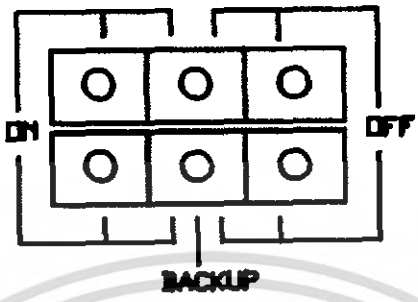
= 26.923K

เลือกใช้ค่า Rx = 27K, Ry = 10K

* ค่าแรงดัน 4.8 โวล์นี้สามารถเปลี่ยนแปลงได้ตามลักษณะของงานที่ใช้*

CMOS RAM Write Protection (จัมพ์เปอร์ BACKUP)

การสำรองข้อมูล (data backup ของ CMOS RAM (U3) และ RTC (DS1202) ในช่วงไฟดับสำหรับบอร์ด ANT-32 ใช้ MAX691 จัดการในส่วนของแบตเตอรี่ และสัญญาณ Chip Enalbe ของ RAM โดย MAX691 รับแรงดันจากแบตเตอรี่ลิเธี่ยม 3 โวลท์เข้าที่ขา Vbatt และแรงดัน Vout จาก MAX691 ต่อเข้าที่ขา Vcc ของ CMOS RAM และ RTC ในการใช้งานปกติขณะใช้ไฟ 5 โวลท์ ที่ขา Vout จะเสมือนถูกต่อเข้ากับขา Vcc ภายใน MAX691 และจะต่อเข้ากับขา Vbatt เมื่อแรงดัน Vcc ต่ำกว่าแรงดันของแบตเตอรี่ ดังนั้นแบตเตอรี่จะถูกใช้งานเฉพาะในช่วงที่ไฟดับเท่านั้น สำหรับ CMOS RAM เพื่อป้องกันข้อมูลสูญหายในขณะไฟดับ MAX691 ได้ออกแบบวงจรควบคุมสัญญาณ Chip Enable ของ RAM โดยที่ขา CE IN ของ MAX691 รับสัญญาณ Chip Enable มาจากวงจร Address Decoder และสัญญาณที่ขา CE OUT จาก MAX691 จะต่อเข้าขา Chip Enable ของ RAM สัญญาณที่ขา CE OUT จะเป็นไปตามสัญญาณที่ขา CE IN ก็ต่อเมื่อแรงดัน Vcc สูงเกิน 4.65 โวลท์ และจะเป็นลอจิก "1" เมื่อแรงดัน Vcc ต่ำกว่า 4.65 โวลท์เพื่อป้องกัน CPU เขียนค่าข้อมูลที่ไม่ถูกต้องลง RAM ในระหว่าง power up และ power down

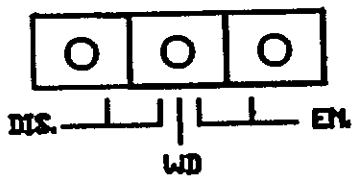


การสำรองข้อมูลของ RAM (U3) สามารถเลือก ON / OFF ได้ด้วยจัมพ์เปอร์ BACKUP ซึ่งมีอยู่ 2 ตำ ส่วน RTC นั้นจะทำการสำรองข้อมูลตลอดเวลาไม่สามารถ OFF ได้

Watchdog Timer (จัมพ์เปอร์ WD)

Watchdog Timer เป็นวงจรที่ทำหน้าที่ตรวจสอบการทำงานของระบบไมโครฯ ว่าทำงานในสภาวะปกติหรือไม่ ถ้าระบบทำงานผิดปกติหรือเกิดอาการแฮงค์ วงจรส่วนนี้จะทำการรีเซท CPU ให้เริ่มทำงานใหม่อีกครั้ง Watchdog Timer จึงมีความสำคัญและจำเป็นมากสำหรับระบบไมโครฯ ซึ่งจะเป็นการเพิ่มเสถียรภาพการทำงานของระบบให้ดียิ่งขึ้น

หลักการทำงานของ **Watchdog Timer** ก็คือ CPU ต้องส่งสัญญาณไปกระตุ้นที่ขา WDI (WATCHDOG INPUT) ของ MAX691 โดยใช้พอร์ต P1.7 ที่ขา OSC IN และ OSC SEL ของ MAX691 ไม่ได้ต่อใช้งาน (ปล่อยลอยไว้) CPU ต้องทำการเปลี่ยนสภาวะ (toggle) ที่ขา WDI ทุก ๆ 1.6 วินาที (Watchdog Timeout Period = 1.6 วินาที) โดยใช้คำสั่ง CPL P1.7 เพื่อให้แน่ใจว่าซอฟท์แวร์ได้ทำงานอย่างถูกต้อง ถ้าฮาร์ดแวร์หรือซอฟท์แวร์เกิดทำงานผิดพลาด ซึ่งจะมีผลทำให้สภาวะที่ขา WDI ไม่เปลี่ยนแปลงตามเวลาที่กำหนดไว้ MAX691 จะส่งสัญญาณรีเซทเป็นพัลส์บวกที่ขา RESET กว้าง 50 ms เพื่อรีเซทให้ CPU กลับไปทำงานใหม่อีกครั้ง และที่ขา RESET นี้จะส่งพัลส์รีเซทออกมาทุก ๆ 1.6 วินาทีจนกว่าจะมีการเปลี่ยนสถานะที่ขา WDI อีกครั้ง



การใช้งาน Watchdog Timer สามารถเลือกใช้งานได้ด้วยจัมพ์เปอร์ WD โดยปรับไปที่ EN. (enable) เมื่อต้องการใช้งาน หรือปรับไปที่ DIS.(disable) เมื่อไม่ต้องการใช้ และในกรณีต้องการเปลี่ยนค่า Watchdog Timeout Period เป็นค่าอื่น สามารถกระทำได้โดยต่อสายจัมพ์ที่ขา OSC SEL (ขา 8) ต่อลง ground และเพิ่มคาปาซิเตอร์ Cx ที่ขา 7 ตามตำแหน่งที่พิมพ์ไว้บนบอร์ด ANT-32 ซึ่งจะมีผลทำให้ค่าความกว้างของพัลส์รีเซท (Reset Timeout Period) เปลี่ยนไปตามค่า Cx ค้ายดังแสดงไว้ในตารางที่ 2

ตารางที่ 2 การเลือกค่า Reset Pulse Width และ Watchdog Timeout

OSC SEL	OSC IN	Watchdog Timeout Period		Reset Timeout
ขา 8	ขา 7	Normal	After Reset	Preiod
Floating	Floating	1.6 sec	1.6 sec	50 ms
Low	Cx	(400ms/47pF)	Cx(1.6sec/47pF)	Cx(200ms/47pF)Cx

<u>ตัวอย่าง</u> ต้องการใช้งาน Watchdog Timer โดยกำหนดค่า Timeout Period = 1 วินาที ค่า Cx สามารถคำนวณโดยใช้สูตร

Cx = (Timeout Period 47 pF) / 400 ms

= (1 sec x 47 pF) / 0.4 sec

= 117.5 pF

เลือกใช้ค่า Cx = 100 pF

ค่า Watchdog Timeout Period สำหรับการใช้งานปกติ (Normal) และหลังจาก CPU ถูกรีเซท (Immediately After Reset) ค่า Reset Timeout Period เมื่อใช้ค่า Cx 100pF หาได้จากสูตรที่แสดงไว้ในตารางที่ 2 ดังนี้

Watchdog Timeout Period (Normal) = (400ms/47pF) X 100pF

= 0.85 sec

Watchdog Timeout Period (After Reset) = (1.6sec/47pF) X 100pF

= 3.4 sec

Reset Timeout Period = (200ms/47pF) X 100pF

= 0.43 sec

<u>คำแนะนำสำหรับการใช้งาน Watchdog Timeer</u>

ในการเขียนโปรแกรมเพื่อใช้งาน Watchdog Timer ควรจะพัฒนาโปรแกรมด้วยภาษาแอสเซมบลี้ ซึ่งเมื่อเริ่มเขียนโปรแกรมต้องไม่ใช้ Watchdog Timer โดยปรับจัมพ์เปอร์ WD ไปที่ DIS.(disable) และเมื่อเขียนโปรแกรมจนได้โปรแกรมที่ทำงานได้ตามที่ต้องการแล้วจึงทำการแทรกคำสั่ง CPL P1.7 โดยประมาณหรือใช้วิธีคำนวณสูงว่า CPU จะวนกลับมาทำคำสั่งนี้ภายในเวลาไม่เกินค่าของ Watchdog Timeout Period ปรับจัมพ์เปอร์ WD ไปที่ EN.(enable) เพื่อใช้งาน Watchdog Timer ผู้ใช้สามารถตรวจสอบว่า CUP ได้กลับมาทำคำสั่งนี้ในช่วงเวลาที่กำหนดหรือไม่ โดยสังเกตจากการทำงานของโปรแกรม ถ้า CPU กลับมาทำคำสั่งนี้ภายในเวลาที่กำหนดโปรแกรมจะทำงานได้ตามปกติเหมือนกับการทำงานของโปรแกรมก่อนที่จะใช้งาน Watchdog Timer แต่ถ้า CUP หรือ Watchdog Timer ไม่ทันตามเวลาที่กำหนด จะสังเกตเห็นว่า CPU จะถูกรีเซทโดยจะกลับไปทำงานในส่วนต้นของโปรแกรมตลอดเวลา หรืออาจใช้ลอจิกโพรบ, ออสซิลโลสโคป ตรวจจับสัญญาณที่ขา P1.7 ของ CPU โดยต้องให้ระยะห่างระหว่างพัลส์แต่ละลูกไม่เกินค่า Watchdog Timeout Period

OPTIONAL REAL TIME CLOCK & CALENDAR

สำหรับการใช้งานระบบไมโครฯ ที่มีเวลามาเกี่ยวข้องด้วย จำเป็นต้องมีวงจรในส่วนที่หน้าที่เป็น RTC (Real Time Clock) คือนาฬิกาเวลาจริง ซึ่งบอร์ด ANT-32 ใช้ REC เบอร์ DS1202 Serial Timekeeper Chip ของ DALLAS SEMICONDUCT โดยต่อร่วมกับอุปกรณ์ภายนอกเพียงเล็กน้อยและที่สำคัญคือ ไม่ต้องทำการปรับแต่ง ซึ่งเมื่อ ใช้ RTC นี้ต้องมีชิพ DS1202 และ MAX691 รวมทั้งคริสตอล 32.768 KHz และแบตเตอรี่ ลิเธี่ยมบนบอร์ด ANT-32 ด้วย

DS1202 (U7) ประกอบไปด้วย Real Time Clock / Calendar และ Static R ขนาด 24 ไบท์ ทำการอินเตอร์เฟสกับ CPU ในแบบอนุกรม โดยใช้สายเพียง 3 เส้น (1) ขา RST (Reset), (2) ขา I/O (Data line) และ (3) ขา SCLK (Seri clock) ขาสัญญาณทั้งสามนี้จะต่อเข้ากับขา P1.6, P1.4 และ P1.5 ของ CPU ตามลำดับ เมื่อต้องการทราบค่าเวลา CPU ต้องทำการอ่านเวลาจาก RTC เพราะว่า DS1202 ขาสำหรับไปอินเตอร์รัพท์ CPU CPU สามารถเขียนหรืออ่านข้อมูลของ CLOCK หรือ P ได้ 2 วิธีคือ Single-byte และ Multiple-byte โดยทั้งสองวิธี CPU ต้อง command byte (8 บิท) ให้ DS1202 เพื่อบอกให้ DS1202 ทราบว่าจะทำการเขียนหรืออ่าน CLOCK หรือ RAM พร้อมตำแหน่ง (address) และตามด้วยข้อมูล ในขณะที่ กำลังติดต่อกับ DS1202 สัญญาณที่ขา RST ต้องเป็นลอจิก "1" ขา SCLK จะเป็นสัญญาณ Serial Clock เพื่อทำการเขียนหรืออ่านข้อมูล โดยจะใช้สัญญาณ Clock 1 ลูกสำหรับ ข้อมูล 1 บิท ส่วนขา I/O เป็นข้อมูลอนุกรม โดยจะเป็นอินพุทเมื่อทำการเขียนและ เอาท์พุทที่ทำการอ่าน โดยข้อมูลที่จะเขียนหรืออ่านนี้จะเริ่มจากบิท O และจบด้วยบิท ค่าของ command byte ในการเขียนและอ่าน CLOCK และ RAM

DEMO CLOCK / CALENDAR PROGRAM

```
100 REM ********** RTC.BAS **********
110 REM TEST DS1202 RTC ON ANT-32 V2.0
120 REM BEFORE RUNNING TS PROGRAM
130 REM USER MUST BE XDOWN RTCSUB.HEX
140 REM SILA RESEARCH CO.,LTD.
150 PORT1=PORT1.AND.0BFH : REM RST  = "0"
170 COMBYTE=18H : DATBYTE=19H : REM INTERNAL RAM
180 GOSUB 1010 : REM SET TIME
190 XBY (0FC03H)=80h : REM USER PORT2 PA, PB, PC=OUTPUT
200 REM
210 REM SECOND CHANG CHECK
220 DBY (COMBYTE)=81h : REM READ SECOND
230 CALL 903AH
240 B=DBY (DATBYTE)
250 IF B=A THEN GOTO 230
260 A=B
270 GOSUB 2000 : REM TIME DISPLAY
280 GOTO 230
290 REM
1000 REM ---------- SET TIME SUBROUTINE ------------
1010 DBY (COMBYTE)=8EH  :  REM WRITE PROTECTION "OFF"
1030 DBY (DATBYTE)=00H  :  CALL 0OOOOH
1040 DBY (COMBYTE)=80H  :  REM WRITE SEC AND CLR CHFLAG
1050 DBY (DATBYTE)=00H  :  CALL 9000H
1060 DBY (COMBYTE)=82H  :  REM WRITE MINUTE
1070 DBY (DATBYTE)=00H  :  CALL 9000H
1080 DBY (COMBYTE)=84H  :  REM WRITE HOUR
1090 DBY (DATBYTE)=00H  :  CALL 9000H
```

```
1100 DBY (COMBYTE)=8EH  :  REM WRITE PROTECTION "ON"
1110 DBY (DATBYTE)=80H  :  CALL 9000H
1120 RETURN
1130 REM
2000 REM ------- TIME DISPLAY SUBROUTINE ----------
2010 REM OUTPUT : USER PORT2 PA=HOUR, PB=MIN, PC=SEC
2020 REM OUTPUT :
2030 XBY (OFCOOH)=A : REM OUTSEC TO PA
2040 DBY (COMBYTE) = 83H : REM READ MINUTE
2050 CALL 903AH
2060 XBY 9OFCO1H)=DBY (DATBYTE) : REM OUT MIN TO PB
2070 DBY (COMBYTE)=85H : REM READ HOUR
2080 CALL 903AH
2090 XBY (OFCO2H)=DBY (DATBYTE) : REM OUTHOUR TO PC
2100 PHO. XBY (OFCO2H0, : REM PRINT HOUR
2110 PHO. XBY (OFCO1H), : REM PRINT MINUTE
2120 PHO. XBY (OFCOOH)  : REM PRINT SECOND
2130 RETURN

; FILENAME      RTCSUB.ASM
; DESCRIPTION   RTC SUBROUTINE FOR BASIC CALL
; HARDWARE      ANT-32 VERSION 2.0 BOARD
; ASSEMBLER     SXA51 ON PC-DOS
; COMPANY       SILA RESEARCH CO.,LTD.
        ORG     9000H           ;STARTING ADDR FOR ASSEMBLY
; P1.4 = I/O DATA
; P1.6 = RST
; P1.5 = SCLK
```

```
; ********** BYTEWR SUB *************
; WRITE SINGLE BYTE TO STC
; IN  = 18H COMMAND
;     = 19H DATA
; REG = A, B, R6, R7
BYTEWR : SETB   RSO          ; SELECT REG BANK-3
         SETB   RS1

         MOV    R6, 18H      ; COMMAND BYTE
         MOV    R7, 19H      ; DATA BYTE
         CLR    P1.4         ; COMMAND BYTE "WRITE"
         LCALL  DELAY
         SETB   P1.6         ; RST = "1"
         LCALL  DELAY
         MOV    B, #8        ; SEND COMMAND
         CLR    C
BYTEWR1  MOV    A, R6
         RRC    A
         MOV    R6, A
         MOV    P1.4, C
         LCALL  SCLKRW
         DJNZ   B, BYTEWR1
         MOV    B, #8        ; SEND DATA BYTE
         CLR    C
BYTEWR2 : MOV   A, R7
         RRC    A
         MOV    R7, A
         MOV    P1.4, C
         LCALL  SCLKRW
         DJNZ   B, BYTEWR2
```

```
        CLR   P1.6        ; RST  = "0"
        LCALL  DELAY
        CLR   RSO         ; SELECT REG BANK-O
        CLR   RS1
        RET
; ************** BYTERD SUB. ***************
' READ SINGLE BYTE EROM STC
; IN  =  18H COMMAND
: OUT =  19H DATA
; REG =  A, B, R6, R7
BYTERD :  SETB   RSO       ; SELECT REG BANK-3
        SETB   RS1
        MOV    R6, 18H     : COMMAND BYTE
        SETB   P1.4        ; COMMAND BYTE "READ"
        LCALL  DELAY
        SETB   P1.6        ; RST   ="1"
        LCALL  DEPAY
        MOV    B, #8       ; SEND COMMAND
        CLR    C
BYTERD1 : MOV    A, R6
        RRC    A, R6
        MOV    R6, A
        MOV    P1.4, C
        LCALL  SCLKCOM
        DJNZ   B.BYTERD1
        MOV    B, #8       ; RECEIVE DATA BYTE
        MOV    R7, #0      ; CLEAR DATA BUFFER
```

```
BYTERD2 : LCALL  SCLKRW
        MOV    A, R7
        MOV    C, P1.4      ; READ SERIAL DATA
        RRC    A
        MOV    R7, A
        DJNZ   B, BYTERD2
        CLR    P1.6         ; RST  = "0"
        LCALL  DELAY
        MOV    19H, R7      ; DATA BYTE
        CLR    RS0          ; SELECT REG BANK-0
        CLR    RS1
        RET
; ********** SCLKCOM SUB. **********
; SERIAL CLOCK FOR WRITE COMMAND
; A FALLING EDGE
; FOLLOWED BY A RISING EDGE
SCLKCOM : CLR   P1.5
        LCALL  DELAY
        SETB   P1.5
        LCALL  DELAY
        RET
; ********** SCLKRW SUB. **********
; SERIAL CLOCK FOR READ/WRITE DATA
; A RISING EDGE
; FOLLOWED BY A FALLING EDGE
SCLKRW :  SETB   P1.5
        LCALL  DEPAY
        CLR    P1.5
        LCALL  DELAY
        RET
```

```
; ************* DELAY SUB. *************
; PULSE DELAY
; REG = R1
DELAY :   MOV    R1, #5
          DJNZ   R1, $
          RET
          END
; FILENAME       REC.ASM
; DESCRIPTION    TEST DS1202 SERIAL TIMEKEEPER CHIP
; HARDWARE       ANT-32 VERSION 2.0 BOARD
; ASSEMBLER      SXA51 ON PC-DOS
; COMPANY        SILA RESEARCH CO.,LTD.
          ORG    8100H        ; STARTING ADDR FOR REM31
; ***************** VARIBALE SET ****************
                       ;** USER PORT 2 **
UP2A      EQU    OFC00H       ; PORT A
UP2B      EQU    OFC01H       ; PORT B
UP2C      EQU    OFC02H       ; PORT C
UP2P      EQU    OFC03H       ; MODE PORT
; ***************** MAIN ****************
; P1.4 = I/O DATA
; P1.6 = RST
; P1.5 = SCLK
MAIN      CLR    P1.6         ; RST  = "0"
          SETB   P1.5         ; SCLK = "1"
          LCALL  DELAY
          LCALL  SETTIME
          MOV    DPTR, #UP2P  ; SET CONTROL CODE
          MOV    A, #80H      ; PA, PB, PC=OUTPUT
          MOVX   @DPTR, A
```

```
MAIN1 :   LCALL  TDISP        ; TIME DISPLAY
          LJMP   MAIN1        ; LOOP AGAIN
; ***************** SETTIME SUB. ************
SETTIME : MOV    R6, #8EH     : WRITE PROTECTION COMMAND
          MOV    R7, #OOH
          LCALL  BYTEWR

          MOV    R6, #80H     : WRITE SECOND AND CLR CHFLA
          MOV    R7, #OOH     : SEC=O
          LCALL  BYTEWE

          MOV    R6, #82H     : WRITE MINUTE
          MOV    R7, #OOH     : SEC=O

          MOV    R6, #82H     ; WRITE MINUTE
          MOV    R7, #OOH     : MIN=O
          LCALL  BYTEWR

          MOV    R6, #84H     : WRITE HOUR
          MOV    R7, #OOH     : HOUR=O
          LCALL  BYTEWR
          RET

*************** BYTEWR SUB. *****************
RITE SINGLE BYTE TO STC
N  =  R6 COMMAND
   =  R7 DATA
EG = A, B, R6, R7
EWR :     CLR    P1.4         ; COMMAND BYTE "WRITE"
```

```
        LCALL  DELAY
        SETB   P1.6        ; RST  = "1"
        LCALL  DELAY
        MOV    B, #8       ; SEND COMMAND
        CLR    C
EWR1 :  MOV    A, R6
        PRC    A
        MOV    R6, A
        MOV    P1.4, C
        LCALL  SCLKRW
        DJNZ   B, BYTEWR1

        MOV    B, #8       ; SEND DATA BYTE
        CLR    C
EWR2 :  MOV    A, R7
        PRC    7
        MOV    R7, A
        MOV    P1.4, C
        LCALL  SCLKRW
        DJNZ   B, BYTEWR2
        CLR    P1.6        ; RST  = "O"
        LCALL  DELAY
        RET
; ********** BYTERD SUB. **********
: READ SINGLE BYTE EROM STC
; IN   = R6 COMMAND
; OUT  = R7 DATA
; REG  = A, B, R6, R7
BYTERD :  SETB   P1.4      ; COMMAND BYTE "READ"
```

```
        LCALL  DELAY
        SETB   P1.6        ; RST  = "1"
        LCALL  DELAY
        MOV    B, #8       ; SEND COMMAND
        CLR    C
BYTERD1 : MOV    A, R6
        PRC    A
        MOV    R6, A
        MOV    P1.4, C
        LCALL  SCLKCOM
        DJNZ   B, BYTERD1

        MOV    B, #8       ; RECEIVE DATA BYTE
        MOV    R7, #0      ; CLEAR DATA BUFFER
BYTERD2 : LCALL  SCLKRW
        MOV    A, R7
        MOV    C, P1.4     ; READ SERIAL DATA
        PRC    A
        MOV    R7, A
        DJNZ   B, BYTERD2
        CLR    P1.6        ; RST  = "O"
        LCALL  DELAY
        RET
; ********** SCLKCOM SUB. **********
; SERIAL CLOCK FOR WRITE COMMAND
; A FALLING EDGE
; FOLLOWED BY A RISING EDGE
CLKCOM :  CLR    P1.5
        LCALL  DELAY
        SETB   P1.5
```

```
        LCALL  DELAY
        RET
********** SCLKRW SUB. **********
SERIAL CLOCK FOR READ/WRITE DATE
A RISING EDGE
FOLLOWED BY A FALLING EDGE

CLKRW :  SETB  P1.5
        LCALL  DELAY
        CLR    P1.5
        LCALL  DELAY
        RET
********** DELAY SUB. **********
PULSE DELAY
REG = R1

ELAY :   MOV   R1, #5
        DJNZ   R1, $
        RET

********** TDISP SUB. **********
TIME DISPLAY
AT 8255 USER PORT 1
PA (HOURS), PB (MINUTES), PC (SECONDS)

DISP:    MOV    R6, #81H     ;READ SECOND
        LCALL  BYTERD
        MOV    DPTR, #UP2A  ;OUT SEC TO PORT A
        MOV    A, R7
        MOVX   @DPTR,A
```

```
MOV    R6,#83H      ;READ MINUTE
LCALL  BYTERD
MOV    DPTR. #UP2B  ;OUT MIN TO PORT B
MOV    A, R7
MOVX   #DPTR, A
MOV    R6, #85H     READ HOUR
LCALL  BYTERD
MOV    DPTR. #UP2C  ;OUT HOUR TO PORT C
MOV    A, R7
MOVX   @DPTR, A
RET
END
```

ANT-32 SPECIFICATION

CPU..... 80C32 CHMOS 8-bits Microcontroller

CLOCK SPEED..... 11.0592 MHz

INTERNAL RAM..... 256 Byte

PROGRAM MEMORY..... U2 8-32KByte:2764 27128 27256 (EPRO

default = 28 pins socket

DATA MEMORY..... U3 8-32KByte:6264 62256 (ram-BACKUP

default = 28 pins socket

PROGRAM & DATA MEMORY..... U4 8-32KByte:2764 27256 (EPROM)

6264 62256 (RAM)

2864 28256 (EEPROM)

DEFAULT = 6264 8KByte RAM

INTERNAL PORT..... 12 bits I/O (PORT1 TO T1 INTO INT1)

PORT..... USER PORT 1,2 48 bits 8255 I/O

LCD PORT (DOT MATRIX ONLY)

SERIAL INTERFACE..... RS232C use MAX232 chip

BACKUP, WATCHDOG, PF DETECTOR. use MAX691 chip (options)

REAL TIME CLOCK..... use DS1202 chip (options)

DATA RETENTION TIME..... over 4 Years for RAM and RTC

CONNECTOR..... 40 pins - System Expansion

26 pins x 2 - User Port 1, 2

16 pins - Internal Port

14 pins - LCD Port

3 pins - RS232C Port

2 pins - 5VDC Power Supply

POWER CONSUMTION..... +5VDC 168 mA (approximate)

SIZE..... 10.2 cm. x 14.2 cm.

ANT-32 MEMORT MAP

OOOOH

U2 (OOOOH-7FFFH) U3 (OOOOH-7FFFH)

CODE PROGRAM DATA MEMORY

EPROM RAM (backup)

2764 6264

27128

27256 62256

8000H

U4 (8000H-F7FFH) CODE AND DATA MEMORY

EPROM EEPROM RAM

2764 2864 6264

27256 28256 62256

F8OOH

U10 (F800H-F9FFH) 8255 USER PORT 1

FAOOH

(FAOOH-FBFFH) LCD PORT

FCOOH

U11 (FCOOH-FDFFH) 8255 USER PORT 2

FEOOH

RESERVE

FFFFH

การติดตั้ง REM31

REM31 สามารถใช้ได้ทั้งบอร์ด JAZZ-31 และบอร์ด ANT-32 รวมทั้งสามารถใช้กับบอร์ดอื่นๆ หรือบอร์ดที่ผู้ใช้พัฒนาขึ้นเองด้วย โดยจะต้องมีคุณสมบัติพื้นฐานทาง HARDWARE ที่เข้ากับ REM-31 ได้ ซึ่งคุณสมบัติพื้นฐานทั้งหมดจะเป็นดังนี้

1. ใช้ไมโครคอนโทรลเลอร์ในตระกูล MCS-51 (เช่น 8031/8032) โดยมีฐานเวลาเป็น 11.0592 MHz

2. มี SOCKET สำหรับหน่วยความจำ EPROM เบอร์ 27256 โดยเป็นหน่วยความจำที่มีตำแหน่งเริ่มที่ 0000 ในส่วนของ PROGRAM MEMORY

3. มีหน่วยความจำ RAM อย่างน้อย 256 BYTE ที่ตำแหน่ง 8000-80FFH ซึ่งเป็นทั้งส่วน PROGRAM และ DATA ได้ในขณะเดียวกัน สำหรับเป็น WORKING AREA

4. มีวงจรเพื่อเป็นการสื่อสารทาง RS232 (SERIAL PORT) ที่ต่อระหว่างตัวไมโครคอนโทรลเลอร์กับเครื่องคอมพิวเตอร์ PC

ในการติดตั้งเข้ากับบอร์ด JAZZ-31 หรือ ANT-32 สามารถกระทำได้ดังนี้

1. นำ REM31 EPROM เสียบลงบอร์ดที่ตำแหน่ง U2 และเลือกเบอร์เป็น 27256

2. ต้องแน่ใจว่ามี RAM อยู่ที่ตำแหน่ง U4 โดยจะเป็นเบอร์ 6264 หรือ 62256 ก็ได้

3. ต่อสาย RS232 ด้านขั้ว 3 PIN เข้ากับบอร์ด (กรณี JAZZ-31 ให้ใช้ HW-RS232 หรือ SERIAL-1) และด้านขั้ว DB25 หรือ DB9 ให้ ต่อเข้ากับเครื่อง PC

4. จ่ายไฟเข้าบอร์ด และเปิดเครื่อง PC ได้

5. ที่เครื่อง PC ให้ผู้ใช้ใช้แผ่นโปรแกรม REM-31 UTILITY โดยจะใช้แผ่นเลยก็ได้หรือจะ COPY ลงบน HARDDISK ก่อน และเรียกใช้จาก HARDDISK ก็ได้ ทั้งนี้ให้เลือกโปรแกรม REM-31 ที่เป็น BATCH FILE ได้เลย

เริ่มต้นใช้งาน

เมื่อทำการติดตั้ง REM31 เข้ากับบอร์ดเป็นที่เรียบร้อยแล้ว ผู้ใช้จะต้องเรียกโปรแกรมสื่อสาร XTALK ซึ่งอยู่บนแผ่น REM31 UTILITY ที่ให้มาด้วยแล้ว การเรียกโปรแกรมสื่อสารสามารถกระทำด้วย BATCH FILE ได้ดังนี้

A>REM31

โปรแกรมจะทำการกำหนดคุณสมบัติของการสื่อสารไว้ ตามลักษณะของการใช้งานกับ REM31 โดยเฉพาะ ทั้งนี้จะมีค่าเบื้องต้นต่างๆ คือ BAUD-RATE = 9600 DATA-BIT = 6 STOP-BIT = 1 PARITY = NONE ค่าต่างๆ เหล่านี้ผู้ใช้จะสามารถเปลี่ยนแปลงได้ตามความต้องการ โปรแกรมจะใช้เวลาสักครู่ จากนั้นจะเข้าสู่โหมดของการสื่อสารทันที โดยจะเป็นจอภาพว่างๆ พร้อมกับข้อความในบรรทัดสุดท้ายดังนี้

^ A for ATtention. ^ F to Switch : Capture Off : Local

ในขณะนี้เครื่องจะรอการกดคีย์ SPACE โดยโปรแกรม REM31 ที่อยู่ในตัว EPROM บนบอร์ดจะทำการคำนวณค่า BAUD RATE ให้โดยอัตโนมัติ พร้อมทั้งกำหนดบอร์ดของ CHIP ที่ใช้งานบนบอร์ดว่าเป็น 8031 หรือ 8032 ด้วย ถ้าผู้ใช้กดคีย์อื่นๆ ที่ไม่ใช่ SPACE เครื่องก็จะไม่กระทำอะไร เมื่อผู้ใช้กดคีย์ SPACE แล้ว ที่จอภาพจะปรากฏข้อความดังนี้

REM31 Remote Monitor V1.0

1991 Sila Research Co.,Ltd.

Baud Rate : 9600 CPU : 8031:

เครื่องหมาย PROMPT "-" แสดงถึงความพร้อมในการใช้งานตามคำสั่งต่างๆ โดยก่อนอื่นใดให้ลองใช้คำสั่ง H (Help) ดู โดยกดคีย์ H แล้วตามด้วย ENTER

-H « กดคีย์ ENTER>

ที่จอภาพก็จะปรากฏข้อความที่เป็นการสรุปคำสั่งที่มีอยู่ของ REM31 ทั้งหมด คำสั่งต่างๆ ของ REM31 นี้จะช่วยให้ผู้ใช้เข้าถึงบอร์ดเป้าหมายได้ ไม่ว่าจะเป็นการป้อนหรือขอดูข้อมูลในหน่วยความจำ การแก้ไข และการสั่งให้ทำงานตามโปรแกรมที่เขียนขึ้นด้วยให้ผู้ใช้ลองป้อนข้อมูลลงหน่วยความจำที่ตำแหน่ง 8100H ดังนี้ (XX คือค่าใดๆ ที่มีอยู่ก่อน)

```
-E 8100       < กดคีย์ ENTER >
8100:XX 12  < กดคีย์ ENTER >
8101:XX 34  < กดคีย์ ENTER >
8103:XX       < กดคีย์ ENTER >
```

ค่าที่ป้อนเข้าไปจะสามารถขอดูได้ด้วยคำสั่ง ดังนี้

```
- D 8100 810F    < กดคีย์ ENER >
8100:  12 34 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00
-
```

จะเห็นว่าแนวทางการใช้ REM31 จะคล้ายคลึงกับโปรแกรม DEBUG ที่มีอยู่ใน DOS ซึ่งถ้าผู้ใช้คุ้นเคยกับโปรแกรม DEBUG อยู่แล้ว ก็จะเรียนรู้ได้รวดเร็วขึ้น อย่างไรก็ตาม จุดที่สำคัญขอเน้นในที่นี้คือ โปรแกรมที่เรียกบนเครื่อง PC เป็นโปรแกรมสำหรับการสื่อสารทาง SERIAL เท่านั้น การทำงานทุกอย่างที่เห็นบนจอภาพจะทำงานด้วยโปรแกรมที่อยู่ในตัว EPROM บนบอร์ดที่กำลังพัฒนา และสิ่งต่างๆ ที่เกิดขึ้นบนจอภาพจะเป็นสิ่งที่เกิดขึ้นจริงบนบอร์ดด้วย คุณสมบัตินี้เรียกว่าเป็น REAL TIME นั่นหมายความว่าถ้าผู้ใช้เขียนโปรแกรมและสั่งให้ทำงาน การทำงานจะมีสภาวะเหมือนความเป็นจริงทุกประการ (ยกเว้นกรณีใช้คำสั่ง เท่านั้น) ถึงแม้ว่าโปรแกรมที่เขียนจะทำงานอยู่ใน และมี เริ่มต้นที่ไม่ใช้ ก็ตาม เพราะฉะนั้นผู้ใช้สามารถทดสอบโปรแกรมได้จนแน่ใจ เมื่อเรียบร้อยแล้วก็สามารถเปลี่ยน ORIGIN ของโปรแกรมเป็น 0000H และนำไปโปรแกรมตัว จากนั้นนำมาเสียบแทนตัว EPROM บนบอร์ดได้เลย

โปรแกรมสื่อสาร XTALK ที่ใช้กับ REM31 นี้ เป็นโปรแกรมที่เรียกว่า TERMINAL EMULATOR ใช้สำหรับการสื่อสารทาง MODEM หรือการสื่อสารเข้ากับระบบคอมพิวเตอร์ที่เป็น MULTI-USER โปรแกรมสื่อสารนี้ผู้ใช้อาจใช้ตัวอื่นก็ได้ตามความถนัด เช่น PROCOM ขอเพียงแต่ให้มีคุณสมบัติในการสื่อสารที่เหมือนกัน อย่างไรก็ตาม สำหรับ REM31 จะมีการอ้างอิงการใช้คำสั่งต่างๆ ของโปรแกรมสื่อสาร โดยใช้ XTLAKเป็นหลัก

ข้อกำหนดเบื้องต้น

การใช้งาน REM31 จะมีหลักการพื้นฐานเกี่ยวกับรูปแบบของคำสั่ง การกำหนดตัวเลข และแนวทางต่างๆ ที่เป็นไปทำนองเดียวกัน โดยจะสรุปเป็นข้อๆ ดังนี้

1. ในการป้อนคำสั่งแต่ละคำสั่ง จะต้องใช้รูปแบบให้ตรงกับที่ได้กำหนดไว้ ซึ่งเริ่มด้วยคำสั่งที่เป็นอักษรตัวเดียว และเว้นอย่างน้อย 1 ช่อง ตามด้วยตัวแปรต่างๆ ที่มีหรือไม่มี และจบท้ายด้วยการกด ENTER เสมอ

2. REM31 จะรับรู้ทั้งอักษรตัวเล็กและตัวใหญ่ โดยจะให้ผลแหมือนกันทุกประการ

3. จำนวนตัวเลขต่างๆ ที่ป้อนเข้าไป จะถือว่าเป็นเลขฐานสิบหกเสมอ (HEX) และไม่ต้องใช้อักษร H ลงท้าย หรือใส่เลขศูนย์นำหน้าตัวอักษร ยกเว้นกรณีที่ใช้คำสั่ง A (Assembler) จะใส่อักษร H ลงท้ายหรือไม่ก็ได้ และต้องใส่ศูนย์นำหน้าตัวอักษร

4. ในกรณีที่ป้อนคำสั่งไม่ถูกต้อง REM31 จะแสดงข้อความว่า ^Syntax Error ให้ทราบโดยเครื่องหมาย ^ จะชี้ไปยังตำแหน่งของคำถามที่ ผิดพลาดด้วย

5. เครื่องหมาย [ ] ของรูปคำสั่ง หมายถึงว่าค่าที่อยู่ภายในจะกำหนดหรือไม่ก็ได้

6. ความหมายของข้อมูลต่างๆ ที่อยู่ในรูปแบบของคำสั่งจะเป็นดังนี้

addr	หมายถึงค่า ADDRESS ขนาด 8 หรือ 16 บิท	เช่น 817F
range	หมายถึงช่วงของ ADDRESS ขนาด 16 บิท	เช่น 8100 81FF
byte	หมายถึงข้อมูลทั่วไปขนาด 8 บิท	เช่น A7
value	หมายถึงข้อมูลทั่วไปขนาด 8 หรือ 16 บิท	เช่น 12E5
reg,var	หมายถึงค่าตัวแปรที่กำหนดเฉพาะในคำสั่งนั้นๆ	

7. การกระทำคำสั่งต่างๆ เกี่ยวกับหน่วยความจำ ปกติจะใช้ส่วนที่เป็น DATA MEMORY เสมอ ยกเว้นคำสั่ง GO,TRACE,UASM เท่านั้น จะใช้ส่วน PROGRAM MEMORY

คำสั่ง A (Assembler)

หน้าที่ สำหรับการทำ MINI-ASSEMBLER

รูปแบบ A addr

คำอธิบาย คำสั่งที่ทำให้ผู้ใช้สามารถป้อนโปรแกรมเป็นภาษา ASSEMBLY ได้ทันที ลงในหน่ายความจำของบอร์ด ซึ่งสะดวกกว่าการป้อนเป็น OP-CODE มาก เมื่อป้อนคำสั่งพร้อมทั้งกำหนด ADDRESS เริ่มต้นแล้ว REM31 จะแสดงค่า ADDRESS และตาม ด้วยเครื่องหมาย ":" พร้อมทั้งรอรับการป้อนข้อมูลจากผู้ใช้ภาษา ASSEMBLY ที่ป้อนต้องตรงตามข้อกำหนดของ INTEL และสามารถใช้ชื่อแทนค่า ADDRESS หรือ BIT-ADDRESS ได้ เช่น ACC P1 RS0 CY เป็นต้น เมื่อป้อนจบแต่ละบรรทัดให้กด ENTER

ถ้าโปรแกรมที่ป้อนเรียบร้อยดี REM31 ก็จะแสดงค่า ADDRESS ตามความยาวอัตโนมัติเพื่อการป้อนบรรทัดต่อไป แต่ถ้าโปรแกรมไม่ถูกต้องREM31จะแสดงข้อความ Assembler Command Error หรือ Assembler Syntax Error ให้ผู้ใช้ทราบ และจะแสดงค่า ADDRESS เดิมเพื่อให้ผู้ใช้ป้อนใหม่อีกครั้งเมื่อผู้ใช้ป้อนโปรแกรมเสร็จแล้ว ให้กด ENTER เพื่อออกจากการใช้คำสั่ง การแปลจะกระทำในลักษณะบรรทัดต่อบรรทัดเพราะฉะนั้นจะไม่สามารถกำหนด LABEL หรือตัวแปรต่างๆ ได้ แต่จะต้องกำหนดค่าตายตัวลงไปเช่นค่า ADDRESS หรือค่าข้อมูลใดๆ กรณีที่เป็นคำสั่ง เกี่ยวกับการกระโดดที่มีค่า RELATIVE หรือ ABSOLUTE ADDRESS (11 BIT-ADDRESS) ให้ป้อนเป็นค่า ADDRESS จริงได้เลย REM31 จะทำการคำนวณให้โดยอัตโนมัติ

กรณีที่อ้างถึงชื่อ BIT-ADDRESS ถ้าเป็นชื่อที่มีความหมายของบิทอยู่แล้ว จะไม่สามารถใช้ชื่อของในลักษณะที่มีจุดชั้นกลางได้ แต่ถ้าเป็นชื่อที่ไม่มีความหมาย ก็จะอ้างในลักษณะมีจุดชั้นกลางได้ ดูตัวอย่างดังต่อไปนี้

EXO ไม่สามารถใช้เป็น IE.0 ได้

CY ไม่สามารถใช้เป็น PSW.7 ได้

ACC.1 ใช้เป็น ACC.1 ได้ เพราะไม่มีชื่อตามความหมาย

P1.4 ใช้เป็น P1.4 ได้ เพราะไม่มีชื่อตามความหมาย

คำสั่งที่มีการกำหนดตัวเลข จะถือว่าเป็นเลขฐานสิบหกเสมอ และ ถ้าเป็นตัวเลขที่มีอักษร A-F นำหน้า ให้ใส่ศูนย์นำหน้าก่อนด้วย

ตัวอย่าง

- A 8100	ทำ MINI - ASSEMBLER ที่ ADDRESS 8100H
8100:CLR A	ป้อนโปรแกรมตามต้องการ
8101:MOV P1,	
8103:MOV R2,#0	

```
8105:MOV R3,#0
8107:DJNZ R3,8107
8109:DJNZ R2,8105
810B:INC A
810C:SJMP 8101
810E:                    ออกจาก ASSEMBLER ด้วยคีย์ ENTER
-
```

คำสั่ง B (Break)

หน้าที่ สำหรับการขอดูหรือกำหนด ADDRESS ที่ต้องการ BREAK โปรแกรม

รูปแบบ B [addr]

คำอธิบาย การกำหนด ADDRESS เพื่อการ BREAK จะช่วยให้ผู้ใช้สามารถทดสอบโปรแกรมได้อย่างสะดวกเพราะจะทำให้ดูค่า REGISTER หรือค่าในหน่วยความจำได้ ณ.จุดที่ทำการและสามรถสั่งให้ทำงานต่อได้ โดยจะมีผลเกิดขึ้นในขณะที่ใช้งานด้วยคำสั่ง G (RUN) คำสั่ง B นี้จะใช้เพื่อดูหรือกำหนดค่า ADDRESS เพื่อการ BREAK โดย ถ้าใช้คำสั่งนี้และไม่ได้ใส่ ADDRESS ต่อท้าย ก็จะเป็นการขอดูค่าที่กำหนดเอาไว้ แต่ถ้า กำหนดค่า ADDRESS ต่อท้ายคำสั่ง ก็จะเป็นการกำหนด ADDRESS สำหรับ BREAK กรณีที่ไม่ต้องการให้มีการ BREAK ก็ให้กำหนด BREAK ADDRESS เท่ากับ 0

ตัวอย่าง

```
-B 812E                  กำหนดค่า BREAK ADDRESS เท่ากับ 0
-B                       ขอดูค่า BREAK ADDRESS ที่กำหนดเอาไว้
812
-
```

คำสั่ง C (Compare)

หน้าที่ สำหรับการเปรียบข้อมูลในหน่วยความจำ

รูปแบบ C range addr

C range H byte

คำอธิบาย สำหรับการเปรียบเทียบข้อมูล โดยจะกระทำได้ 2 ลักษณะคือ การเปรียบเทียบเป็น BLOCK และการเปรียบค่า BYTE การเปรียบเทียบนี้จะแสดง ADDRESS และข้อมูลที่ ไม่ตรงให้ทราบ

ตัวอย่าง

```
-C 8100 81FF 8800        เปรียบเทียบข้อมูลเป็น BLOCK
8117:58 8817:F8          แสดง ADDRESS และข้อมูลที่ไม่ตรงกัน
814E:14 884E:16
-
```

คำสั่ง D (Display)

หน้าที่ สำหรับการดูข้อมูลในหน่วยความจำ

รูปแบบ D [addr] [addr]

คำอธิบาย สำหรับการแสดงข้อมูล DATA MEMORY โดยจะแสดงเป็นค่า HEX และ ASCII ในขณะเดียวกัน ทั้งนี้ถ้าผู้ใช้ไม่ได้กำหนด ADDRESS REM31 จะนำค่า ADDRESS ล่าสุดที่ใช้งานมาเป็น ADDRESS เริ่มต้น REM31 ซึ่งจะทำให้ผู้ใช้สามารถดูไปได้อย่างต่อเนื่อง แต่ถ้ากำหนด ADDRESS เริ่มต้น ก็จะแสดงตาม ADDRESS ที่กำหนด โดยจะแสดงเป็นจำนวน 128 Byte แต่ถ้าผู้ใช้กำหนด ADDRESS ทั้งเริ่มต้นและสุดท้าย REM31 ก็จะแสดงตามที่ต้องการ

ตัวอย่าง

```
-D 8100

8100: 90 F5 81 74 80 FE E4 90 - FE 00 F8 7A 00 7B 00 25   ...t.......z{.%
8110: FE DA F4 24 58 F4 82 00 - 00 00 00 00 00 00 00 00   ...$X...........
8120: 00 00 00 00 00 00 00 00 - 32 00 00 00 00 00 00 00   ........2.......
8130: 00 00 00 22 00 00 00 00 - 00 00 01 01 00 00 00 00   ...".............
8140: 00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00   ................
8150: 00 00 02 00 00 00 03 00 - 00 00 00 00 00 00 00 00   ................
8160: 00 00 00 00 00 00 00 00 - 07 00 00 00 00 00 00 00   ................
8170: 00 00 00 00 00 00 00 00 - 00 00 02 00 00 00 00 00   ................
-
```

คำสั่ง E (Enter)

หน้าที่ สำหรับการป้อนข้อมูลลงในหน่วยความจำ

รูปแบบ E addr

คำอธิบาย สำหรับการป้อนข้อมูลลงใน DATA MEMORY โดยจะป้อนเป็นเลข HEX ได้ทันที ADDRESS คือตำแหน่งเริ่มต้นที่จะป้อนเมื่อใช้คำสั่งนี้ REM31 จะแสดงค่า ADDRESS พร้อมทั้งข้อมูลเดิมที่มีอยู่ และจากให้ผู้ใช้ใส่ค่าใหม่ที่ต้องการลงไป เมื่อต้องการออกจากคำสั่งนี้ให้กด ENTER อีกครั้ง

ตัวอย่าง

```
-E 8100              ป้อนข้อมูลที่ ADDRESS 8100H
8100:74 78           ค่าเดิมคือ 74H และเปลี่ยนเป็น 78H
8101:A5 30
8102:02
-
```

คำสั่ง F (Fill)

หน้าที่ สำหรับการใส่ค่าข้อมูลลงในหน่วยความจำ

รูปแบบ F range byte

คำอธิบาย สำหรับการนำค่าข้อมูลใดๆ หนึ่งค่า (BYTE) ใส่ลงใน DATA MEMORY ตามช่วง ADDRESS ที่กำหนด

ตัวอย่าง

```
-F 8100 87FF FF      นำค่า FFH ใส่ใน DATA MEMORY ตั้งแต่ 8100 ถึง 87FF
-
```

คำสั่ง G (Go)

หน้าที่ สำหรับการสั่งให้ทำงานตามโปรแกรม

รูปแบบ G [addr]

คำอธิบาย สำหรับการสั่งให้กระโดดที่ไปทำงานตามโปรแกรม โดยจะไปยัง ADDRESS ที่กำหนด ต่อจากคำสั่ง ในกรณีที่ไม่ได้กำหนด ADDRESS REM31 จะนำค่า ADDRESS ที่อยู่ใน PC มาใช้ (ค่า PC ดูได้จากคำสั่ง R) เมื่อบอร์ดทำงานตามโปรแกรมที่ผู้ใช้เรียกขึ้นแล้ว ก็จะสามารถดูผลต่างๆ ที่เกิดขึ้นได้ตามความเป็นจริง การหยุดโปรแกรมจะทำได้ในกรณีต่อไปนี้

1. มีการกำหนด BREAK ADDRESS เอาไว้ โดยเมื่อโปรแกรมทำงานไปจนถึงจุด BREAK ก็จะหยุดโปรแกรมและแสดงค่า ADDRESS ที่ทำการ BREAK ไว้ พร้อมกับแสดงเครื่องหมาย PROMPT "-" โดยผู้ใช้จะดูสถานะต่างๆ ได้ตามต้องการ ทั้งนี้ PC จะมีค่า ADDRESS ที่ต่อจากจุด BREAK ด้วย ซึ่งผู้ใช้จะสั่งให้ทำงานต่อได้ทันที

2.มีการกำหนด END VECTOR ที่โปรแกรมของผู้ใช้เอาไว้ END VECTOR คือส่วนของโปรแกรมใน REM31 ที่ผู้ใช้สามารถกระโดดไปได้ เมื่อสิ้นสุดโปรแกรมที่เขียนขึ้น โดยใช้คำสั่งดังนี้

JUMP 0033H

โดย REM31 จะแสดงข้อความ END PROGRAM ให้ทราบ และจะเข้าสู่เครื่องหมาย PROMPT "-" เพื่อการใช้งานต่อไป

3. ด้วยการกดคีย์ RESET บนบอร์ด ทั้งนี้ REM31 จะแสดงข้อความว่า SYSTEM RESET ... ให้ทราบ และเข้าสู่เครื่องหมาย PROMPT "-" เพื่อการใช้งานต่อไป

การหยุดโปรแกรมในข้อ 1 และ 2 จะเป็นการหยุดที่ผู้ใช้สามารถดูผลที่เกิดขึ้นจริงใน REGISTER และ SFR ได้ ทั้งนี้เพื่อการตรวจสอบการทำงานของโปรแกรมนั้นเอง แต่การหยุดด้วยคีย์ RESET จะไม่สามารถเห็นค่า REGISTER และ SFR ที่เปลี่ยนไปรายละเอียดในเรื่องนี้ให้ทำความเข้าใจได้จากหัวข้อเรื่อง " รายละเอียดทาง SOFTWARE "

ตัวอย่าง 1

	-G 8100	สั่งให้โปรแกรมทำงานที่ ADDRESS 8100H
	8117 Break	มีการกำหนดจุด Breakที่ ADDRESS 8117H ไว้
	-	
ตัวอย่าง 2	-G	สั่งให้โปรแกรมทำงานที่ ADDRESS จากค่า PC
	End Program.....	โปรแกรมทำงานจนถึงจุดที่มี END VECTOR
	-	

คำสั่ง H (Help)

หน้าที่ สำหรับการแสดงชุดคำสั่งของ REM31

รูปแบบ H

คำอธิบาย สำหรับการแสดงชุดคำสั่งทั้งหมดของ REM31 โดยจะเป็นการสรุปรูปแบบและหน้าที่ของแต่ละคำสั่ง ให้เห็นได้ภายในจอภาพเดียว ทั้งนี้เพื่อความสะดวกของผู้ใช้ในการทบทวนหรือค้นหา ในขณะที่ใช้งานอยู่

ตัวอย่าง -G

```
REM31 Command Summary ...
A addr          Assembler
B [addr]        Break Set/Display
C range addr    Compare Block
  range A byte  Compare Byte
D [addr] [addr] Display Memory
E addr          Enter Data
F range byte    Fill Memory
G [addr]        Go (Run)
H               Help
I [addr]        Internal RAM Set/Display
L [addr]        Download INTEL-HEX File from PC
M range addr    Move Memory
N               New (Clear User INT-RAM, Register & SFR)
O range         Upload INTEL-HEX File to PC
Q               Quit (Power off or Reset)
R [reg value]   Register & SFR Set/Display
S range byte ...  Search Memory (8 Byte Max)
T [addr]        Trace (Single Step)
U [addr] [addr] Unassembler (Program Memory)
V [var value]   Variable Set/Display
-
```

คำสั่ง I (Internal RAM)

หน้าที่ สำหรับการขอดูข้อมูลและกำหนดค่าใน INTERNAL RAM

รูปแบบ I [addr]

คำอธิบาย สำหรับการขอดูข้อมูลและกำหนดค่าใน INTERNAL RAM ที่อยู่ภายในตัว CHIP โดยถ้าเป็น 8031 จะมี 128 BYTE (00-7FH) และ 8032 จะมี 256 BYTE (00-FFH) ถ้าผู้ใช้ไม่กำหนด ADDRESS ก็จะเป็นการแสดงค่า โดย REM31 จะแสดงเป็นจำนวนเท่าที่มีในแต่ละเบอร์ การแสดงข้อมูลจะมีรูปแบบในทำนองเดียวกับคำสั่ง D คือแสดงในแบบ HEX และแบบ ASCII ถ้าผู้ใช้กำหนด ADDRESS (ADDRESS ที่กำหนดต้องไม่เกิน 8 บิท) ก็จะเป็นการป้อนข้อมูลลงใน INTERNAL RAM โดยการป้อนข้อมูลจะมีหลักการในทำนองเดียวกับคำสั่ง E การป้อนข้อมูลในตำแหน่ง 00-1F จะเป็นการป้อนใน REGISTER R0-R7 ซึ่งอาจจะกระทำได้อีกทางด้วยคำสั่ง R ทั้งนี้เพราะ CHIP ในตระกูล MCS-51 ออกแบบให้ INTERNAL ในส่วนต้น คือ REGISTER จำนวน 4 BANKS

กรณีที่แสดงข้อมูล REM31 จะแสดงเครื่องหมาย "*" ที่ ADDRESS ตั้งแต่ 40-7FH เอาไว้ด้วย ทั้งนี้เพื่อให้ผู้ใช้ทราบว่าเป็น WORKING AREA ที่ใช้เพื่อการทำงานของ REM31 เอง ซึ่งปกติผู้ใช้ควรใช้งานในส่วนนี้

ตัวอย่าง

```
-I 20          ป้อนข้อมูลใน INTERNAL RAM ที่ ADDESS 20H
20:00 78       ค่าเดิมคือ 00H เปลี่ยนเป็น 78H
21:00 45
22:00
-
```

คำสั่ง L (Download)

หน้าที่ สำหรับการ DOWNLOAD ข้อมูลหรือโปรแกรมจากเครื่อง PC ลงบอร์ด

รูปแบบ L [addr]

คำอธิบาย สำหรับการ DOWNLOAD ข้อมูลหรือโปรแกรมจากเครื่องคอมพิวเตอร์ PC ลงมายังบอร์ดที่กำลังพัฒนา โดยจะต้องเป็น FILE ในรูปแบบของ INTEL-HEX ค่า ADDRESS ที่กำหนดจะดปีนตำแหน่งที่ต้องการให้โหลดลง ถ้าไม่กำหนด REM31 ก็จะใช้ค่าที่กำหนดมาจาก FILE การ DOWNLOAD นี้ต้องอาศัยการใช้คำสั่งของโปรแกรมสื่อสาร XTALK ร่วมด้วย รายละเอียดให้ทำความเข้าใจกับตัวอย่างอีกที

ตัวอย่าง -L กำหนดให้ทำการ DOWLOAD จากเครื่อง PC

Download INTEL-HEX FILE.....

Use ^A and SE Command to send File

เครื่องจะแสดงข้อความให้ทราบว่า กำลังรอรับ FILE จากเครื่อง PC ในระหว่างนี้ผู้ใช้จะคีย์อะไรไม่ได้ นอกจากขบวนการส่ง FILE เท่านั้น เพราะไม่เช่นนั้น REM31 จะเข้าใจว่าเป็น ERROR จากการสื่อสาร และจะแสดงข้อความว่า Download Error ให้ทราบ

ขบวนการส่ง FILE จาก XTALK จะกระทำได้ดังนี้

1. ให้กด ^A เพื่อเข้าสู่โหมดการใช้คำสั่งของ XTALK โดยเครื่องจะแสดงคำว่า Cmmand ? ที่บรรทัดสุดท้ายของจอภาพ และ พร้อมที่จะรับคำสั่งต่างๆ ของ XTALK

2. ใช้คำสั่ง และตามด้วยชื่อ ที่ต้องการส่ง ดังนี้

SE FILENAME.HEX

โดย FILE ที่จะส่งต้องเป็น INTEL-HEX FILE และต้องอยู่ใน DIRECTORY นั้นๆ แต่ถ้าอยู่ที่อื่นผู้ใช้สามารถใช้คำสั่ง CD เพื่อการเปลี่ยน CURRENT DIRECTORY ได้

3. เครื่องจะแสดงข้อความ Sending file ... บนจอภาพ และทำการส่ง FILE ออกไปทาง SERIAL PORT ทันที

เมื่อเรียบร้อยแล้ว REM31 จะกลับมาแสดงเครื่องหมาย PROMPT "-" ตามเดิม

คำสั่ง M (Move)

หน้าที่ สำหรับการ COPY ข้อมูลเป็น BLOCK

รูปแบบ M range addr

คำอธิบาย สำหรับการ COPY ข้อมูลหรือโปรแกรมใน DATA MEMORY จากส่วนหนึ่งไปยังอีกส่วนหนึ่ง โดยจะทำการ COPY เป็นช่วงความยาวตามที่กำหนด ค่า rang คือช่วงของข้อมูลที่เป็นแม่แบบ และ addr คือ ADDRESS เริ่มต้นของส่วนที่ต้องการใส่ข้อมูล การใช้คำสั่ง MOVE นี้จะมีระบบคำนวณการ COPY แบบขึ้นหรือลงโดยอัตโนมัติ เพราะฉะนั้นจึงสามารถ COPY แม้กระทั่งช่วงที่มี ADDRESS ซ้อนกับแม่แบบ

ตัวอย่าง -M 8100 81FF 9000 ทำการ COPY ข้อมูลจาก 8100-81FFH ไปที่ 9000-90FFH

-

คำสั่ง N (New)

หน้าที่ สำหรับการ CLEAR ข้อมูลใน REGISTER

รูปแบบ N

คำอธิบาย สำหรับการ CLEAR ข้อมูล REGISTER.INTERNAL RAM และ SFR โดยจะใช้ในขณะ เริ่มต้นก่อนการทดสอบโปรแกรม เพื่อช่วยให้สะดวกยิ่งขึ้น การ CLEAR ข้อมูลจะมีรายละเอียดดังนี้

1. CLEAR ข้อมูลใน INTERNAL RAM ตั้งแต่ 00-3FH (64 BYTE) ซึ่งจะรามทั้ง R0-R7 ทั้ง 4 BANKS ด้วย

2. CLEAR ข้อมูลใน SFR โดยจะกระทำคล้ายกับการ RESET ซึ่งมีรายละเอียดคือ ACC=0 B=0 PSW=0 SP=7 DPTR=0

3. กำหนดให้ PC (PROGRAN COUNTER) มีค่าเท่ากับ 8100H

คำสั่ง P (uPload)

หน้าที่ สำหรับการ UPLOAD ข้อมูลหรือโปรแกรมจากบอร์ดขึ้นเครื่อง PC

รูปแบบ P range

คำอธิบาย สำหรับการ UPLOAD ข้อมูลหรือโปรแกรมจากบอร์ดที่พัฒนา ขึ้นไปยังเครื่อง PC ซึ่งจะเป็นขบวนการที่ตรงข้ามกับคำสั่ง L (DOWNLOAD) คือ ช่วง ADDRESS ที่ต้องการ

FILE ที่ UPLOAD จะเป็น INTEL-HEX FILE เช่นกัน ทั้งนี้ขบวนการ UPLOAD ก็คือการนำข้อมูลมาปรากฏบนจอภาพ ในแบบ INTEL-HEX FILE นั้นเองโดยขั้นตอนในการเก็บข้อมูลลง DISK จะอาศัยคำสั่งของ XTALK ช่วย (คำสั่ง CAPTURE) รายละเอียดให้ทำความเข้าใจกับตัวอย่างอีกที

ตัวอย่าง -P 8100 81FF ให้ทำการ UPLOAD ข้อมูลตั้งแต่ 8100-81FFH

Upload INTEL-HEX FILE ....

Use ^A and CA Command yo Upload File

Type ENTER to Start and Finish

REM31 จะแสดงข้อความให้ผู้ใช้ทราบถึงความพร้อมในการรับ FILE ซึ่งในขั้นตอนนี้ จะต้องใช้คำสั่งของ XTALK ช่วยในการรับ FILE โดยทำได้ดังนี้

1.ให้กด ^A เพื่อการเข้าสู่โหมดการใช้คำสั่งของ XTALK โดยเครื่องจะแสดงคำว่า Command ? ที่บรรทัดสุดท้ายของจอภาพและพร้อมที่จะรับคำสั่งต่างๆ ของ XTALK

2.ใช้คำสั่ง CA และตามด้วยชื่อ FILE ที่ต้องการ UPLOAD ข้อมูล ดังนี้

CA FILENAME.HEX

กรณีที่เป็น FILE ที่มีอยู่ใน DISK แล้ว เครื่องจะถามความแน่ใจอีกครั้งว่าจะให้ลบก่อนหรือไม่ หรือให้ ต่อจาก ที่มีอยู่แล้ว

3. เมื่อเครื่อง PC พร้อมที่จะรับ FILE แล้ว ให้กด ENTER อีกครั้ง REM31ก็จะส่ง FILE ให้ปรากฏบนจอ และขณะเดียวกับที่เครื่อง PC จะรับข้อความเหล่านี้เก็บลง FILE ด้วย

4. เมื่อ REM31 หยุดส่ง แสดงว่าได้ทำการรับข้อมูลเรียบร้อยแล้วให้กด ^A อีกครั้ง และใช้คำสั่ง CA OFF เพื่อหยุดขบวนการรับ FILE จากนั้นให้กด ENTER อีกครั้ง REM31จะแสดงเครื่องหมาย PROMPT ตามปกติ

คำสั่ง Q (Quit)

หน้าที่ สำหรับหยุดการใช้งาน REM31

รูปแบบ Q

คำอธิบาย สำหรับหยุดการใช้งาน REM31 และพร้อมที่จะทำการปิดเครื่องหรือกด ENTER อีกครั้งทั้งนี้ในการทำคำสั่ง Q REM31 จะยกเลิกตัวแปรต่างๆ ที่ได้จำเอาไว้ เช่น BAUD-RATE และเมื่อผู้ใช้ทำการ RESET ก็จะมีสภาวะเหมือนกับขณะที่เปิดเครื่องครั้งแรก คือจะรอการกดคีย์ SPACE นั้นเอง

คำสั่ง R (Register)

หน้าที่ สำหรับการดูและและกำหนดค่าใน REGISTER

รูปแบบ R [reg value]

คำอธิบาย สำหรับการดูและและกำหนดค่าใน REGISTER ถ้าผู้ใช้ไม่ได้กำหนดข้อมูลต่อท้ายคำสั่งก็จะเป็นการแสดงค่าใน REGISTER ซึ่งจะแสดงทั้งส่วน SFR และ REGISTER R0-R7 พร้อมกับค่า PROGRAM COUNTER (PC) ในการแสดงค่า R0-R7 จะเป็นไปตาม BANK ที่กำหนดจาก PSW และถ้ามีเครื่องหมาย "*" อยู่ที่ใด ก็แสดงว่าเป็นเครื่องหมายที่เก็บในลักษณะ VIRTURL

ถ้าผู้ใช้ใช้คำสั่งโดยมีข้อมูลต่อท้าย คือกำหนดชื่อ REGISTER ที่ต้องการ และตามด้วยข้อมูลที่กำหนด ก็จะเป็นการกำหนดค่าให้กับ REGISTER นั้นทันที กรณีที่ใส่ชื่อ REGISTER ไม่ถูกต้อง REM31 จะแสดง ^Syntax Errorให้ผู้ใช้ทราบ และค่าที่กำหนดจะต้องไม่เกิน 8 บิตด้วย

คำสั่ง S (Search)

หน้าที่ สำหรับการค้นหาตำแหน่งของข้อมูลในหน่วยความจำ

รูปแบบ S range byte (8 MAX)

คำอธิบาย สำหรับการค้นหาตำแหน่งของข้อมูลใน DATA MEMORY โดยจะเปรียบเทียบกับข้อมูลตามที่กำหนด ซึ่งจะใส่ได้สูงสุดถึง 8 BYTE ค่า range คือช่วง ADDRESS ที่จะทำการค้นหา และค่า BYTE คือค่าข้อมูลสำหรับการเปรียบเทียบ ค่า BYTE นี้จะกำหนดกี่ BYTE ก็ได้ แต่ต้องไม่เกิน 8 BYTE และแต่ละ BYTE จะต้องเว้นด้วย BLANK เสมอ

คำสั่ง T (Trace)

หน้าที่ สำหรับการสั่งให้ทำงานตสโปรแกรมเพียง 1 ครั้ง

รูปแบบ T [addr]

คำอธิบาย สำหรับการสั่งให้กระโดดไปทำงานตามโปรแกรม โดยจะทำเพียง 1 คำสั่ง เท่านั้น (SINGLE STEP) โดยถ้าผู้ใช้กำหนด ADDRESS ต่อท้าย REM31 ก็จะไปทำงานตาม ADDRESS นั้น แต่ถ้าไม่กำหนด REM31 จะนำค่า ADDRESS ที่อยู่ใน PC มาใช้ (PC คือ PROGRAM COUNTER โดยดูได้จากคำสั่ง R) คำสั่ง T จะใช้เพื่อการทดสอบโปรแกรม ซึ่งผู้ใช้สามารถเห็นผลการทำงานที่เปลี่ยนแปลงในแต่ละคำสั่ง ทั้งนี้เมื่อใช้คำสั่ง T แล้ว REM31 จะแสดงค่า REGISTER ต่างๆ ที่สำคัญบนจอภาพ เพื่อความสะดวกในการทดสอบด้วย (ผู้ใช้อาจจะใช้คำสั่ง R เพื่อดูค่าทั้งหมดได้) การใช้คำสั่ง T จะมีเงื่อนไขต่างๆ ดังนี้

1. คำสั่งนี้จะไม่สนค่า BREAK ที่ผู้ใช้กำหนดเอาไว้ด้วยคำสั่ง B

2. การทำงานที่คำสั่งนี้ จะหยุดเฉพาะคำสั่งที่อยู่ต่อไปเท่านั้น ไม่สามารถกระโดดไปตามลักษณะของคำสั่งได้ เช่นถ้าไปถึงคำสั่ง SJMP REM31 ไม่หยุดตามค่า ADDRESS ของคำสั่ง SJMP แต่จะหยุดเมื่อไปถึงคำสั่งถัดไปที่ต่อจาก SJMP (เรียกว่า SEQUENCIAL SINGLE STEP)

3. เมื่อถึงคำสั่ง LCALL REM31 จะไปกระทำใน SUBROUTINE จนจบแล้วจึงมาหยุดที่คำสั่งถัดไปของ LCALL

4. คำสั่งนี้จะใช้ได้กับหน่วยความจำที่เป็นทั้ง PROGRAM และ DATA MEMORY (8000-FFFH) และขณะเดียวกันต้องใช้ RAM ในการทำงานด้วย

คำสั่ง U (Unassembler)

หน้าที่ สำหรับการทำ UN-ASSEMBLER หรือ DIS-ASSEMBLER

รูปแบบ U [addr] [addr]

คำอธิบาย สำหรับการทำ DIS-ASSEMBLER ข้อมูลใน PROGRAM MEMORY ซึ่งสามารถกำหนดความยาวด้วย ADDRESS ที่ต่อท้ายคำสั่ง ทั้งนี้ถ้าผู้ใช้ไม่กำหนด ADDRESS REM31 จะนำค่า ADDRESS ล่าสุดที่ใช้งานมาเป็น ADDRESS เริ่มต้น ซึ่งจะทำให้ผู้ใช้สามารถดูไปได้อย่างต่อเนื่อง แต่ถ้ากำหนด ADDRESS เริ่มต้น ก็จะแสดงตาม ADDRESS นั้น โดยจะแสดงเป็นจำนวน 16 บรรทัด แต่ถ้าผู้ใช้กำหนด ADDRESS เริ่มต้นและสุดท้าย REM31 ก็จะแสดงตามที่ต้องการ ขบวนการ DIS-ASSEMBLER นี้ ผู้ใช้อาจจะเก็บข้อความที่ปรากฏบนจอภาพลง DISK ได้ ทั้งนี้ขอให้ใช้วิธีการในทำนองเดียวกับการ UPLOAD ข้อมูลด้วยคำสั่ง P

คำสั่ง V (Vector)

หน้าที่ สำหรับการดูหรือกำหนด VECTOR ADDRESS

รูปแบบ V [var value]

คำอธิบาย สำหรับการดูหรือกำหนดค่า INTERRUPT VECTOR ADDRESS ปกติ MCS-51 จะกำหนด ADDRESS แน่นอนสำหรับการกระโดดไปยัง SUBROUTINE ในกรณีที่เกิดการ INTERRUPT โดย ADDRESS เหล่านี้จะอยู่ในช่วงต้นๆ ของ PROGRAM MEMORY แต่เนื่องจาก PROGRAM MEMORY ถูกใช้เป็น MONITOR แล้ว REM31 จึงมีระบบโยกย้าย VECTOR ADDRESS เพื่อให้ผู้ใช้สามารถใช้งานเกี่ยวกับ INTERRUPT ได้อย่างปกติ คำสั่ง V จึงใช้เพื่อการดูหรือกำหนดค่าเหล่านี้ กรณีที่ใช้คำสั่งโดยไม่มีข้อมูลต่อท้าย ก็จะเป็นการขอดูค่าที่กำหนดเอาไว้ แต่ถ้ามีการใส่ข้อมูลต่อท้าย ก็จะเป็นการกำหนดค่าที่ต้องการ var จะใช้ตัวเลข 1-5 แทน (กรณี 8032 จะเท่ากับ 1-6) และ value คือ VECTOR ADDRESS ที่กำหนด

รายละเอียดทาง SOFTWARE

POWER และ RESET

การจ่ายเข้าบอร์ดครั้งแรกกับการกดคีย์ RESET จะส่งให้ REM31 ที่กำหนดค่าตัวแปรและคุณสมบัติต่างๆ เปลี่ยนแปลงไป ซึ่งนอกเหนือจากที่เกิดขึ้นภายในตัว SHIP โดยรายละเอียดจะสรุปได้ดังนี้

การทำงาน	POWER-UP	RESET
1. ทำการ CLEAR ข้อมูลทั้งหมดใน INTERNAL RAM	YES	NO
2. ทำงานการ AUTO BAUD-RATE และกำหนดคุณสมบัติ ในการสื่อสารของ SERIAL PORT	YES	NO
3. ทำงานการ AUTO CPU กำหนดเบอร์ที่ใช้โดยอัตโนมัติ	YES	NO
4. กำหนด ADDRESS สำหรับคำสั่ง D และ U ให้เริ่มต้นที่ 8100H พร้อมทั้งค่า RS232 SPEED ที่ใช้งาน	YES	NO
5. กำหนดคุณสมบัติในการสื่อสารตามที่จำไว้ขณะ POWER-UP	NO	YES
6. กำหนดค่า SFR ในส่วนที่เป็น VIRTUAL ดังนี้ ACC=0 B=0 PSW=0 SP=7 DPTR=0	YES	YES
7. กำหนดค่า PC (PROGRAM COUNTER) = 8100H	YES	YES

ความเข้าใจเกี่ยวกับระบบ VIRTUAL

ในการใช้งาน REM31 ผู้ใช้จะดูเหมือนว่าสามารถอ่านและเขียนข้อมูลใน REGISTER หรือ SFR ได้อย่างปกติ แต่ในความเป็นจริง REGISTER และ SFR จะถูกใช้เพื่อการทำงานของ REM31 อยู่ตลอดเวลา และไม่สามารถมีค่าที่แน่นอนที่กำหนดได้ เพราะฉะนั้นจึงต้องมีการจัด MEMORY ส่วนหนึ่ง ที่เรียกว่า VIRTUAL MEMORY เพื่อเป็นตัวเสมือนให้กับ REGISTER และ SFR เหล่านั้น โดยผู้ใช้จะอ่านและเขียนกับส่วนนี้แทนและเมื่อขณะที่ต้องการ RUN (หรือ

TRACE) ก็จะทำขบวนการโหลดค่าใน VIRTUAL เข้าไปยังตัว REGISTER และ SFR จริง แล้วจึงออกไปสู่โปรแกรมของผู้ใช้ ในทางกลับกันเมื่อจบโปรแกรมของผู้ใช้ (หรือมีการ BREAK หรือ TRACE) REM31 ก็จะโหลดค่าจากตัว REGISTER และ SFR จริง เข้าสู่ส่วนที่เป็น VIRTUAL MEMORY

ปกติผู้ใช้จะไม่รู้สึกถึงขบวนการอันนี้เลย แต่อย่างไรก็ตาม ระบบ VIRTUAL จะมีผลต่อความเข้าใจในการใช้งานด้วย เพราะใน REM31 จะมี REGISTER และ SFR บางส่วนที่เป็น VIRTUAL และอีกส่วนจะเป็นค่าจริง (ส่วนที่เป็น VIRTUAL จะมีเครื่องหมาย "*" กำกับไว้ เมื่อใช้คำสั่ง R) ลองทำความเข้าใจกับตัวอย่างดังต่อไปนี้

USER END VECTOR

ใน REM31 จะมีระบบ END VECTOR สำหรับให้ผู้เขียนไว้ที่ท้ายสุดของโปรแกรม ทั้งนี้เมื่อจบโปรแกรมของผู้ใช้แล้ว ก็จะเข้าสู่เครื่องหมาย PROMPT "." เพื่อใช้งานต่อไป END VECTOR สามารถเขียนได้ด้วยคำสั่งดังนี้

LJMP 0033H (OPCODE = 02 00 33)

ทั้งนี้เมื่อโปรแกรมกระโดดไปยัง ADDRESS 0033H แล้ว REM31 จะทำขบวนการคืนค่า REGISTER และ SFR ต่างๆ ด้วย ซึ่งผู้ใช้จะสามารถตรวจสอบการทำงานต่างๆ ของโปรแกรมได้อย่างสะดวก ลักษณะเช่นนี้จะคล้ายคลึงกับขบวนการ BREAK นั้นเอง

INTERRUPT VECTOR ADDRESS

REM31 จะอนุญาตให้ผู้ใช้กำหนด VECTOR ADDRESS เพื่อการ INTERRUPT ได้โดยจะกำหนดด้วยคำสั่ง V หรือจะกำหนดจากโปรแกรมของผู้ใช้ก็ได้ทั้งนี้ค่าที่เก็บ VECTOR ADDRESS เหล่านี้จะอยู่ที่ DATA MEMORY ตั้งแต่ 8000-800BH ดังนี้

VECTOR NAME	MCS-51 ADDRESS (CODE)	REM31 ADDRESS (DATA)
1. INT0	0003H	8000-8001H
2. TF0	000BH	8002-8003H
3. INT1	0013H	8004-8005H
4. TF1	001BH	8006-8007H
5. R1 + T1	0023H	8008-8009H
6. TF2 + EXF2 (8032)	002BH	800A-800BH

ตัวอย่างเช่น ถ้าผู้ใช้ต้องการให้ VECTOR ADDRESS ของ INT1 มีค่าเท่ากับ 8245H ซึ่งเป็นตำแหน่งที่มีโปรแกรมย่อยอยู่ จะสามารถเขียนกำหนดไว้ที่โปรแกรมดังนี้

```
MOV     DPTR,#8004H
MOV     A,#82H
MOVX    @DPTR,A
INC     DPTR
MOV     A,#45H
MOVX    @DPTR,A
.....
```

รายละเอียดทั่วไป

1. REM31 จะกำหนดความยาวของ BUFFER เพื่อการรับคำสั่งทางจอภาพให้ 40 ตัวอักษร ซึ่งเพียงพอต่อการใช้งานเป็นอย่างดี แต่ถ้าผู้ ใช้คีย์เกินกำหนด REM31 จะส่งเสียงเตือนให้ทราบ และจะหยุดอยู่ที่จำนวน 40 เท่านั้น จนกว่าจะกด

2. โปรแกรมสื่อสาร XTALK เป็นเพียงตัวช่วยในการติดต่อกับ REM31 เท่านั้น เพราะฉะนั้นผู้ใช้อาจจะออกจากโปรแกรม XTALK (ด้วยคำสั่ง QU) เข้าสู่ DOS และเรียกโปรแกรมใหม่อีกครั้ง โดยที่ไม่มีผลต่อการทำงานของ REM31 แต่อย่างใด

3. ระบบ AUTO BAUD-RATE สามารถใช้ได้กับความเร็ว 1200-9600

4. ระบบ AUTO CPU จะกำหนดเบอร์ได้ 2 เบอร์คือ 8031 และ 8032

การใช้งานทั่วไปของ XTALK

โปรแกรมสื่อสาร XTALK (หรือ CROSSTALK) เป็นโปรแกรมเพื่อการสื่อสารซึ่งสามารถใช้งานได้ในหลายลักษณะ เช่น ใช้เป็น TERMINAL EMUTOR (กรณีกับ REM31 จะใช้แบบนี้) ใช้เพื่อการส่งและรับข้อมูลจากเครื่อง PC หรือใช้เพื่อการสื่อสารทาง MODEM ปกติ XTALK จะเรียกใช้งานดังนี้

A>XTALK FILENAME.XTK

FILE ที่ต่อท้ายคำสั่ง คือ FILE สำหรับเก็บค่าตัวแปรต่างๆ ของการสื่อสารไว้ FILE นี้จะถูกสร้างได้จากตัว XTALK เอง หรือจะสร้างจาก EDITOR ก็ได้ เพราะจะต้องมีลักษณะเป็น TEXT FILE กรณีที่ใช้กับ REM31 บรรทัดสุดท้ายของ FILE นี้จะต้องลงท้ายด้วยคำว่า "GO LOCAL" เสมอ เมื่อเข้าโปรแกรม XTALK แล้ว เครื่อง PC ก็พร้อมที่จะทำการสื่อสารทันที โดยผู้ใช้สามารถเรียกใช้คำสั่งของ XTALK ได้ดังนี้

1. กด ^A เพื่อการป้อนคำสั่งที่บรรทัดล่างสุดของจอภาพ

2. กด ^F เพื่อการป้อนคำสั่งเช่นกัน และขณะเดียวกัน ก็จะแสดงตัวแปรต่างๆ ของการสื่อสารให้ปรากฏบนจอภาพด้วย

ชุดคำสั่งของ XTALK จะมีลักษณะเป็นตัวอักษร 2 ตัว โดยผู้ใช้สามารถจะดูชุดคำสั่งทั้งหมดได้ด้วยการใช้คำสั่ง HE (HELP) ชุดคำสั่งต่างๆ เหล่านี้ จะใช้เพื่อการเปลี่ยนแปลงตัวแปรต่างๆ ในการสื่อสาร หรือเพื่ออำนวยความสะดวกทั่วไปในขณะทำงาน โดยถ้าต้องการคำอธิบายในแต่ละคำสั่ง ก็สามารถใช้คำสั่ง HE และตามด้วยคำสั่งนั้นๆ อีกที (HELP แบบละเอียด) เมื่อผู้ใช้ต้องการกลับเข้าสู่การสื่อสาร ให้กด ENTER อีกครั้ง ชุดคำสั่งของ XTALK ที่ใช้งานบ่อยๆ จะสรุปได้ดังนี้

HE [XX]	HELP	สำหรับการขอดูชุดคำสั่งทั้งหมดหรือการขอดูรายละเอียดของคำสั่ง XX ที่ต้องการ
PO 1.2	PORT	กำหนดเลข SERIAL PORT ของเครื่อง PCที่จะใช้งาน
PA NO.OD.EV	PARITY	กำหนดลักษณะของ PARITY BIT
SE FILENAME	SEND	ส่ง FILENAME ออกทาง SERIAL PORT
SP 12...96	SPEED	กำหนดความเร็วของการสื่อสาร
QU	QUIT	ออกจากโปรแกรม XTALK เข้าสู่ DOS
CA FILENAME	CAPTURE	สำหรับเก็บสิ่งที่ปรากฏบนจอภาพลง FILE
ST 1.2	STOP	กำหนดจำนวน STOP BIT
CD DIRECTORY	CHANG	เปลี่ยนแปลง DIRECTORY ที่ใช้งาน

หลักการทำงานของวงจร

จากโครงงานนี้วงจรจะแบ่งออกเป็น 2 ส่วนคือ

1. Control bonkl
2. I/O Card

Control boord ในส่วนของ Control Boord นี้จะใช้ Boord สำเร็จรูป ANT - 32 ในการควบคุมการทำงานทั้งหมดของเครื่อง Accerss Key โดย boord มี หน้าที่ ในการควบคุมการทำงานต่าง ๆ ดังนี้ คือ

ใช้ในการบอกเวลา แต่ระบบโดยบน boord จะใช้ IC RTC (Real time clock) เบอร์ DS 1202 เป็นตัวบอกเวลาการรับส่งข้อมูลจะทำแบบ Series เพื่อการประหยัดจำนวนขา สัญญาณในการต่อโดยในการต่อจะใช้ขาสัญญาณเพียง 3 เส้นเท่านั้น คือ ขา P1.4 , P1.5 , P1.6 ในการตัดต่อ

มีการต่อขยาย Port ออกมาเป็น User Post 1 และ User Post 2 จาก Post Po ของไมโครโปรเซสเซอร์ 8032 โดยใช้ IC 8255 โดยใน Project นี้ได้ใช้ Uaer Port 1 ในการติดต่อรับข้อมูลจาก Keyboard : ใช้ในการตรวจสอบลำดับของสัญญาณ Interupt ; และใช้สำหรับส่งสัญญาณไปควบคุมที่ I/O Card เพื่อควบคุม การปิดเปิดประตู หรือ ปิด เปิด Ararm

มีการตรวจสอบ Wodr Dog เพื่อป้องกันไม่ให้ไมโครโปรเซสเซอร์ทำงานผิดผลาดโดยใช้ IC เบอร์ Max 691 A

และมี LCD Port สำหรับใช้ในการติดต่อกับจอ LCD เพื่อใช้ในการแสดงผล

I/O CARD

Card นี้จะถูกควบคุมโดย Control boord โดยต่อเข้ากับ User Part 1 และ Zotermal post หน้าที่ของ Card นี้คือจะทำหน้าที่ในการสแกน Key Boord และใช้ในการตรวจรับสัญญาณ Semsor จาก ไมโคร สวิสซ์ อีกด้าย การ Scom Key Boord จะทำโดยใช้ IC เบอร์ 74 C 922 เป็น IC สแกน Key boord เข้า 16 ปุ่มใช้ Copool tormo ภายนอก 1 ตัว ในการกำหนด ค่าวงจรที่ในการสแกนและเมื่อมีการกด Key Boord จะทำให้สัญญาณ DA (Data arilable ของ 74 C 922 มีค่าเป็น High C1) ซึ่งขา DA นี้จะไปต่อเข้าที่ I/P ของ Nor Gate ซึ่งโดยปกติ I/P ทุก I/P ของ Nor Gote จะเป็น Low ("O")ซึ่งจะทำให้ O/P ของ Nor gote มีค่าเป็น High ("1") ซึ่งจะต่อกับขา Interrupt ของ Interrupt Part ทำให้ไม่มีการ Active ของสัญญาณ INT แต่เมื่อมีการ Keyboord จะทำให้ PA เป็น Hight ("1") ดังนั้นจะทำให้ O/P ของ Norgate เป็น LOW ("O") เป็นผลให้

IMT Active ก็จะทำให้ UP สามารถรู้ว่ามีอุปกรณ์ภายนอกมาขอ Interrupt โดยสามารถตรวจจาก Part B ของ User Post 1 ว่าอุปกรณ์ใดเป็นตัวเรียก Imterrupt ได้ โดยแต่ละบิตของ Part B จะปสดงสถานะของอุปกรณ์ภายนอกแต่ละตัวคือ โดยปกติแต่ละบิตจะมีสถานะเป็น High หมดเมื่ออุปกรณ์มีการขอ Imterrupt จะทำให้บิตที่ต่ออุปกรณ์ที่ของ Imterrupt ตัวนั้น เป็น Low ทำให้เราสามารถที่จะทราบได้ว่า อุปกรณ์ตัวใดเป็น Imterrupt แต่ Data ที่ได้จากการอ่านตำแหน่งของปุ่มที่กดจะถูกอ่านเข้ามาที่ Post A ของ User Post 1ในส่วนของ Semsor ทั้ง 2 ตัวนั้นจะเป็นไมโครสวิสช์ ซึ่งโดยปกติจะ Closs อยู่ตลอดเวลาซึ่งจะทำ I/P ของ Not Gate ต่อลง GND ทำให้ G/P ของ Not gate เป็น High ซึ่งต่อเป็น I/P ของ IC Memo Srble 74 Ls ซึ่งจะสร้างพิวส์ออกมาเมื่อสถานที่ I/P เปลี่ยนจาก High เป็น Lowเมื่อ Semsor ทำงาน ซึ่งก็คือ หมายถึงว่า ประตูถูกงัดหรือ เครื่อง Access Key ถูกต้อง จะทำให้ไมโครสวิทช์ Open ทำให้ O/P ของ Not Gate เป็น Low ทำให้ Memostable จะสร้างพิวส์ออกมามีความแห้งของพิวส์ประมาณเท่ากับ 0.7 RC เท่าที่ S/P ของ Nor Gate เป็นผลให้ O/P ของ Not gate เป็น Low ทำให้สัญญาณ Imterrupt Active ทำให้ Up สามารถรู้ว่ามีอุปกรณ์ภายนอก Imterrupt โดยสามารถตรวจสอบจากที่ Post B ของ User Post 1 ว่าอุปกรณ์ใดเป็นต้าขอ Imterrupt ถ้า บิตใดเป็น "0" แสดงว่าอุปกรณ์ที่ต่อกับบิตนั้นเป็นต้าขอ Imterrupt ซึ่งโดยปกติที่ไม่มีการ Imterrupt ทุกบิตที่ Post B จะเป็น Hight หมด

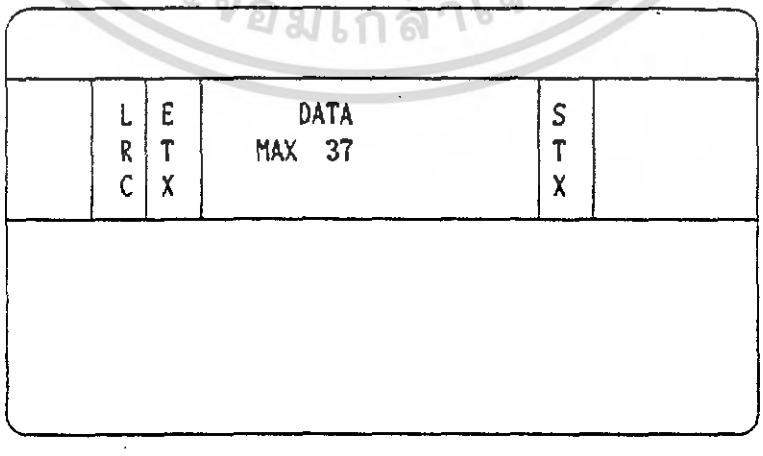
ส่วนที่ Post C จะทำหน้าที่เป็น O/P ในการควบคุมการเปิด ปิดประตูและ บีด , เปิด Sinor และ Buzzer ซึ่งที่ บิต 0 ของ post C ซึ่งจะต่อเข้ากับ Nor gate จะทำหน้าที่ควบคุม การ ปิด,เปิด ของ ระบบ Alarm ทั้งหมด โดย บิต 1,2 ของ Post C จะทำหน้าที่ ควบคุม Sinor และ Buzzer โดยใช้ Transistor เป็น Switch ในการ ON/OFF Relay ให้ทำงาน ส่วน IC นั้นจะทำหน้าที่ในการสร้างความถี่เพื่อป้อนให้กับ Buzzer เพื่อทำให้เกิดเสียงขึ้น

บัตรแม่เหล็ก (Magnetic Card)

เป็นบัตรที่สามารถบันทึกข้อมูล และสามารถโปรแกรมให้ทำงานได้ในหน้าที่ที่แตกต่างกันออกไป เช่นใช้เป็นบัตรรูดเข้าประตู , ใช้เป็นบัตรโปรแกรมเครื่อง , ใช้เป็นบัตรตรวจรับทราบเหตุฉุกเฉิน (Alarm Reset Card)

ชนิดของบัตร	หน้าที่
Programming Card (บัตร "P")	ใช้โปรแกรมให้เครื่องทำงานตามที่ต้องการ หรือเปลี่ยนโปรแกรมที่มีอยู่ในเครื่องให้เป็นโปรแกรมตามวัตถุประสงค์ของผู้ใช้
Access Card (บัตร "N")	ใช้รูดเพื่อต้องการที่จะเข้าประตู
Reset Alarm Card (บัตร "A")	ใช้รูดเพื่อหยุดเสียงการแจ้งเตือนของระบบ และ Reset ระบบให้กลับคืนสู่สภาวะปกติ

รูปแบบของบัตรแถบแม่เหล็กนี้ ด้านหลังจะมีแถบแม่เหล็กสีดำ ซึ่งสามารถบักทึกข้อมูลของบัตรไว้ภายใน



ปุ่ม Function การทำงานหน้าชุดควบคุม

ปุ่มดังกล่าวที่นำมาใช้งานนี้มีทั้งหมด 16 ปุ่ม ซึ่งแต่ละปุ่มจะมีหน้าที่การทำงานดังนี้

ชนิด	หน้าที่
1. ปุ่ม Prog	ใช้สำหรับเปิดเครื่องเข้าการโปรแกรมหรือเข้าสู่ Function การทำงานของระบบทั้งหมด
2. ปุ่ม Disa	ใช้สำหรับปิดสัญญานแจ้งเตือนเวลามีการโจรกรรม หรือการบุกรุก และเพื่อรับทราบเหตุฉุกเฉิน
3. ปุ่ม →	ทำหน้าที่เลือกโปรแกรมการทำงานไปข้างหน้าตามลำดับ
4. ปุ่ม ←	ทำหน้าที่เลือกโปรแกรมการทำงานถอยหลังตามลำดับ
5. ปุ่ม Enter	ใช้สำหรับให้เครื่องรับทราบ หรือตกลงคำสั่งในโปรแกรมนั้น ๆ
6. ปุ่ม Clear	ใช้สำหรับยกเลิกรายการในโปรแกรม หรือออกจากโปรแกรมย่อย
7. ปุ่มตัวเลข 0-9	เป็นปุ่มตัวเลข 10 ตัว ที่ใช้ควบคู่กับปุ่มต่าง ๆ เวลาใช้งาน หรือโปรแกรม

ขั้นตอนการทำงาน

เมื่อจ่ายแหล่งจ่ายไฟเข้าเครื่องควบคุม ระยะเริ่มแรกที่ LCD จะปรากฏคำว่า "KMITL ACCESS KEY " และจะ Delet ประมาณ 5 วินาที ตัวอักษรคำนี้ก็จะหายไปจากนั้นจะมีคำว่า "SWIP YOUR CARD" ขึ้นมาแทนสลับกับ "วันที่ / เดือน / ปี " และ เวลาเป็น "ชั่วโมง : นาที" สลับกันไปสลับกันมาเป็นอย่างนี้ตลอดจนกว่าจะมีบัตรมารูดที่หัวอ่าน

บัตรใบแรกที่ถูกนำมารูดที่หัวอ่านบัตร บัตรใบนั้น ๆ จะเป็นบัตรโปรแกรมทันทีโดยเครื่องควบคุมจะบันทึกข้อมูลที่แถบแม่เหล็กของบัตรนั้น ลงในเครื่อง และ Reset ให้บัตรนั้นเป็นบัตรโปรแกรม (Program Card) ไปโดยอัตโนมัติ ซึ่งถ้าจะเข้าสู่ Function Program ต้องใส่ Pass Word ที่เครื่องควบคุม Set ไว้จำนวน 4 หลักลงไป

เมื่อเข้าสู่ฟังก์ชั่นโปรแกรมได้แล้ว ก็จะสามารถเลือ Menu ของเครื่อง เพื่อจะ Set การทำงานของระบบที่เราต้องการได้ เช่น

1. การตั้งเวลา วัน เดือน ปี
2. การป้อนบัตรใหม่ เข้าสู่ระบบ
3. การลบบัตรทิ้งออกไปจากระบบ
4. เลือกบัตรใช้งาน เช่น เป็นบัตรโปรแกรม , เป็นประตูและบัตร Reset การแจ้งเตือนของระบบเมื่อมีผู้บุรุกเข้าประตู
5. อื่น ๆ ดังแสดงตามขั้นตอนการโปรแกรม

ในสภาวะปกติของระบบที่จอ LCD จะปรากฏคำว่า "SWIP YOUR CARD" สลับกันโชว์กับคำว่า "วันที่ / เดือน / ปี..... ชั่วโมง : วินาที" เท่านั้น

ขั้นตอนการโปรแกรมค่าตัวแปรของระบบ (System Parameter)

EDIT TIME / DATE

เป็นโปรแกรมฟังก์ชั่นของ เวลา ซึ่งจะป้อนในรูปของชั่วโมง : นาที และเป็นโปรแกรมฟังก์ของ วัน / เดือน / ปี โดยมีขั้นตอนการทำโปรแกรมดังนี้

สมมุติต้องการเป็น วันที่ 23 May 1991 เวลา 13:30 น. ขั้นตอนจะเป็นดังนี้

1. รูดบัตรโปรแกรม แล้วตามด้วยรหัส PIN	Edit Time/date
2. กดปุ่ม ENTER	Time :
3. กดปุ่ม หมายเลข 1330 (13:30 น.)	Time : 1330
4. กดปุ่ม ENTER ยืนยันการเปลี่ยน	Save
	Date : -

5. กดปุ่มหมายเลข 230591 (23 May 1991)	Date : 230591
	Save
6. กดปุ่ม ENTER ยืนยันการเปลี่ยน จากนั้นอีก	Edit Time / Date
ประมาณ 5 วินาที จะกลับสู่ MENU เดิม	
7. กดปุ่ม CLEAR เพื่อออกสู่สภาวะปกติของระบบ	Swipe your cardm
	23 / 05/ 91 13 : 30

Enter new card

เป็นโปรแกรมฟังก์ชั่น ตั้งเครื่อง Access Key ซึ่งจะไม่ยอมรับบัตรใบใดเลยถ้าบัตรใบนั้นไม่ได้โปรแกรมให้บันทึกลงในฐานข้อมูลของเครื่องไว้ก่อนโดยใช้ฟังก์ชั่นโปรแกรมนี้

ขั้นตอนการบันทึกข้อหมายเลขบัตรลงฐานข้อมูล ก็คือ โดยการนำบัตรแต่ละใบมารูดที่หัวอ่านเพื่อไปบันทึกข้อมูลนี้ลงในเครื่องควบคุม แล้วก็กำหนดค่าตัวแปรต่าง ๆ สำหรับบัตรแต่ละใบ ซึ่งมีขั้นตอนคำเนินการดังนี้

1. รูดบัตรโปรแกรมแล้วกดรหัส	Edit Time / Date
2. กดปุ่ม → เลือกจนถึง	Enter New Card
3. กดปุ่ม ENTER	Swipe Your Card
4. นำบัตรแถบแม่เหล็กมารูด (บัตรที่ต้อง	Ind 0002 : Valid
การป้อนเข้าเครื่อง)	Cd Type : NOR
5. กดปุ่มเลือกตัวอักษร PRO , A และ NOR	Cd Type : PRO
เพื่อเลือก ชนิดของบัตรมาใช้งาน	Save
(สมมุติ เลือก PRO) แล้วกด Enter บัตรใบ	
นี้จะเป็นบัตรโปรแกรม	
6. กดปุ่ม Enter จะเลื่อนไปรายการถัดไปเป็น	New Pin :
การโอนรหัสตัวเลข 4 หลัก (สมมุติต้องการ	
รหัส 1234)	
7. กดปุ่มตัวเลข 1,2,3 และ 4 ตามลำดับ	New Pin : * * * * *
8. กดปุ่ม Enter	Save
จากนั้นอีกประมาณ 5 วินาที จะกลับสู่สภาวะ	Swipe Your Card
เพื่อรอการป้อนบัตรใหม่ ๆ เข้าไปอีก	
9. กดปุ่ม CLEAR จะกลับสู่สภาวะ Menu เดิม	Enter New Card

10.กดปุ่ม CLEAR จะกลับสู่สภาวะปกติของระบบ Swipe Your Card

วัน / เดือน / ปี ชั่วโมง : นาที

kDELETE EXIST CARD

เป็นโปรแกรมฟังก์ชั่น บัตรที่สูญหายไป จำเป็นต้องยกเลิกออกจากฐานข้อมูลในเครื่อง เมื่อบัตรถูกยกเลิก มันจะลบออกจากฐานข้อมูลในหน่วยความจำอย่างถาวร โดยมีขั้นตอนดังต่อไปนี้

สมมติว่าเราต้องการยกเลิกบัตรหมายเลขซึ่งแทนด้วย Index No : 0003

1. รูดบัตรโปรแกรมแล้วกดรหัส Pin 4 หลัก	Edit Time / Date
2. กดปุ่ม เลือกฟังก์ชันจนกระทั่งจอ LCD แสดง	Delet Exist Card
3. กดปุ่ม ENTER	Index No :
4. กดปุ่ม ตัวเลข 0003	Index No : 0003
5. กดปุ่ม ENTER	SAVE
จากนั้นอีกประมาณ 5 วินาที จะกลับสู่สภาวะ	Index NO :
เพื่อเริ่มลบบัตรหรือยกเลิกบัตรต่อไป	
6. กดปุ่ม CLEAR เพื่อกลับสู่สภาวะ MENU เดิม	Delete Exist Card
7. กดปุ่ม CLEAR จะกลับสู่สภาวะปกติของระบบ	Swipe Your Card
	วันที่ / เดือน / ปี ชั่วโมง : นาที

DOOR STRIKE TIME

เป็นโปรแกรมฟังก์ชั่น เมื่อกลอนประตูคลายออก เราต้องให้เวลาพอเพียงสำหรับให้คนผลักประตูเข้าไปก่อนที่มันจะล็อคอีกครั้ง ดังนั้น Door Strike Time จึงต้องตั้งให้มีเวลาอย่างน้อย 3 วินาที (ตั้งได้ตั้งแต่ 3-250 วินาที)

สมมติว่าต้องการโปรแกรม Door Strike Time เป็น 5 วินาที จะมีขั้นตอนการตั้งโปรแกรมดังนี้

1. รูดบัตรโปรแกรมแล้วกดรหัส Pin 4 หลัก	Edit Time / Date
2. กดปุ่ม เลือกฟังก์ชั่นจนจอ LCD แสดง	Door Strike Tm.
3. กดปุ่ม ENTER	Str Tm 003 : -

(003 เป็นเวลาที่ตั้งมาแล้วกับเครื่อง)

4. กดปุ่ม เลข 5 เปลี่ยนเวลา Door Strike Time — Str Tm 003: 5-
เป็น 5 วินาที
5. กดปุ่ม ENTER เพื่อยืนยันการเปลี่ยน — SAVE
จากนั้นอีกประมาณ 5 วินาที จะกลับสู่สภาวะ — Str TM 005 : -
6. กดปุ่ม CLEAR เพื่อกลับสู่สภาวะ MENU เดิม — Door Strike Tm.
7. กดปุ่ม CLEAR เพื่อกลับสู่สภาวะปกติของระบบ — Swipe Your Card
วันที่ / เดือน / ปี ชั่วโมง : นาที

ALARM OUTPUT DURATION

เป็นโปรแกรมฟังก์ชั่น เพื่อโปรแกรมเวลาที่เครื่องจะส่งสัญญาณเตือนภัยให้กระดิ่งหรือไซเรนดังนานเท่าใด ซึ่งสามารถโปรแกรมได้ 1 นาที เป็นต้นไป

เมื่อมีเหตุเตือนภัยเกิดขึ้น เนื่องจากเครื่องจับได้ว่ามีผู้พังประตูเข้าไป หรือฝาเครื่องถูกเปิดออกโดยผู้ที่ไม่ได้รับอนุญาต เครื่องจะส่งสัญญาณเตือนภัยออกมาซึ่งติดตั้งภายนอกเครื่อง เพื่อแจ้งเตือนภัยให้ผู้ที่รับผิดชอบหรือเกี่ยวข้องทราบ

1. รูดบัตรโปรแกรม แล้วกดรหัส Pin — Edit Time / Date
2. กดปุ่ม → เพื่อเลือก MENU จนจอ LCD แสดง — Alm Oupput Dur
3. กดปุ่ม ENTER — Alarm 001 : -
("001" เป็นเวลาที่ตั้งจากเครื่องไว้แล้ว)
4. กดปุ่มหมายเลข 015 เพื่อเปลี่ยนเวลาเป็น 15 นาที — Alarm 001 : 015
5. กดปุ่ม ENTER เพื่อยืนยันการเปลี่ยนเวลา — SAVE
จากนั้นอีกเวลาประมาณ 5 วินาที จะกลับสู่สภาวะ — Alarm 015 : -
6. กดปุ่ม CLEAR เพื่อกลับสู่สภาวะ MENU เดิม — Alm Output Dur
7. กดปุ่ม CLEAR เพื่อกลับสู่สภาวะปกติของระบบ — Swipe Your Card
วัน / เดือน / ปี ชั่วโมง : นาที

DOOR ACCESS BEEP DURATION

เป็นโปรแกรมฟังก์ชั่น เพื่อระบุว่าประตูจะเปิดทิ้งไว้ได้นานเท่าใดก่อนที่ระบบจะส่งสัญญาณเป็นเตือนด้วย FUZZFE ภายในเครื่อง (ซึ่งการเข้าประตูจะต้องได้รับอนุญาติจากระบบเท่านั้น) ซึ่งจะมีขั้นตอนการโปรแกรมดังนี้

สมมติว่าต้องการโปรแกรม Acs Beep Dur เท่ากับ 30 วินาที จะได้

1.รูดบัตรโปรแกรมแล้วกดรหัส Pin	Edit Time / Date
2. กดปุ่ม → เพื่อเลือก MENU จนจอ LCD แสดง	Acs Beep Dur
3. กดปุ่ม ENTER	B Dur 010 : -
("010") เป็นเวลาที่ตั้งไว้กับเครื่องไว้แล้ว	
4. กดปุ่มหมายเลข 030 เพื่อเปลี่ยนเวลาเป็น 30 วินาที	B Dur 001 : 030
5. กดปุ่ม ENTER เพื่อยืนยันการเปลี่ยนเวลา	SAVE
จากนั้นอีกเวลาประมาณ 5 วินาที จะกลับสู่สภาวะ	B Dur 030 : -
6. กดปุ่ม CLEAR เพื่อกลับสู่สภาวะ MENU เดิม	Acs Beep Dur
7. กดปุ่ม CLEAR เพื่อกลับสู่สภาวะปกติของระบบ	Swipe Your Card
	วัน / เดือน / ปี ชั่วโมง : นาที

<u>CHECK CARD INFORMATION</u>

คำสั่งนี้ใช้ตรวจสอบบัตรว่า ดี หรือ เสีย ถ้าบัตรที่ดี (ใช้ได้) เมื่อรูดบัตรผ่านเข้าเครื่องควบคุมที่จอ LCD จะแสดง INDEX No. ของบัตรได้ถูกต้อง โดยมีขั้นตอนการเช็คได้ดังนี้

1. รูดบัตรโปรแกรมแล้วกดรหัส Pin	Edit Time / Date
2. กดปุ่ม → เพื่อเลือก MENU จนจอ LCD แสดง	Check Card Info
3. กดปุ่ม ENTER	Swipe Your Card
4. นำบัตรที่เราต้องการทราบ Index No. xxxx มารูด	Index No : xxxx
จากนั้นตัวเลขประมาณ 5 วินาที จะกลับสู่สภาวะ	
เพื่อรูดบัตรต่อไป	
5. กดปุ่ม CLEAR เพื่อกลับสู่สภาวะ MENU เดิม	Check Card Info
6. กดปุ่ม CLEAR เพื่อกลับสู่สภาวะปกติของระบบ	Swipe Your Card
	วันที่ / เดือน / ปี ชั่วโมง : นาที

ALARM MODE

เป็นคำสั่งที่สามารถเลือกว่า ถ้ามีการบุกรุกผ่านประตูเพื่อเข้าห้องควบคุมระบบจะสั่งหรือไม่สั่งการแจ้วเตือนภัยออกมา แล้วแต่เราจะเลือกคำสั่งตัวนี้เป็น ON หรือ OFF โดยมีขั้นตอนการโปรแกรมดังนี้

สมมติให้คำสั่ง MODE นี้ เป็น OFF ซึ่งจะไม่มีการแจ้งเตือน

1. รูดบัตรโปรแกรม แล้วกดรหัส PIN	Edit Time / Date
2. กดปุ่ม → เลือก MENU จนจอ LCD แสดง	Alarm Mode
3. กดปุ่ม ENTER	Alarm Mode : ON
(เครื่องควบคุมจะตั้งไว้ที่ ON ไว้อยู่แล้ว)	
4. กดปุ่ม → เพื่อเลือกเป็น OFF ระบบ ALARM	Alarm Mode : OFF
5. กดปุ่ม ENTER เพื่อยืนยันการเปลี่ยน	SAVE
จากนั้นอีกเวลาประมาณ 5 วินาทีจะกลับสู่สภาวะ	Alarm Mode : OFF
6. กดปุ่ม CLEAR เพื่อกลับสู่สภาวะ MENU เดิม	Alarm Mode
7. กดปุ่ม CLEAR เพื่อกลับสู่สภาวะปกติของระบบ	Swipe Your Card
	วันที่ / เดือน / ปี ชั่วโมง : นาที

SYSTEM DISARMING

เป็นคำสั่งปิดสัญญาณแจ้งเตือนจากที่มีผู้บุกรุกโดยไม่ได้รับอนุญาติ เพื่อผ่านประตูเข้าไปในห้อง และ RESET ระบบสัญญาณแจ้งเตือนด้วย เพื่อให้ระบบการแจ้งเตือนที่เกิดขึ้นกลับสู่สภาวะการทำงานที่ปกติ โดยมีขั้นตอนการใช้ดังนี้

1. รูดบัตรที่ใช้สำหรับ RESET การแจ้งเตือน	Pin No.-
2. กดปุ่มรหัส Pin No. ของบัตร 4 หลัก	Pin No.****
3. กดปุ่ม DISARM LAD จะแสดงการเกิด Alarm	System Alarmed
4. กดปุ่ม DISARM อีกครั้งเพื่อ RESET ALARM	System Disarmed
5. กดปุ่ม CLEAR เพื่อกลับสู่สภาวะปกติของระบบ	Swipe Your Card
	วันที่/เดือน/ปี ชั่วโมง:นาที

DOOR ACCESS MODE

เป็นโปรแกรมฟังก์ชั่น เพื่อควบคุมประตูให้ล็อค หรือคลายล็อคตลอดเวลา โดยที่ไม่ต้องใช้บัตรมารูด เพื่อผ่านเข้าออก ซึ่งมีไว้เพื่อบางทีต้องการขนย้ายเครื่อง ซึ่งต้องใช้เวลานาน ๆ ในการเข้า-ออก บริเวณห้องควบคุม โดยมีขั้นตอนการทำงานดังนี้

1. รูดบัตรที่ใช้สำหรับโปรแกรม แล้วกดรหัส PIN	Edit Time/Date
2. กดปุ่ม→ลือก Menu จนจอ LCD แสดง	Door Acs Mode
3. กดปุ่ม Enter (เครื่องควบคุมจะตั้งไว้ที่ ON ไว้อยู่แล้ว)	Door Acs : ON
4. กดปุ่ม→เพื่อเลือกเป็น OFF ให้ประตูคลายล็อค	Door Acs : OFF
5. กดปุ่ม Enter เพื่อยืนยันการเปลี่ยน	<<SAVE>>
จากนั้นอีกประมาณ 5 วินาทีจะกลับสู่สภาวะ	Door Acs : Off
6. กดปุ่ม CLEAR เพื่อกลับสู่สภาวะ Menu เดิม	Door Acs Mode
7. กดปุ่ม CLEAR เพื่อกลับสู่สภาวะปกติของระบบ	Swipe Your Card
	วันที่/เดือน/ปี ชั่วโมง:นาที

<u>การนำไปใช้งานของระบบตามความต้องการต่าง ๆ ซึ่งจะมีขั้นตอนดังนี้</u>

<u>เข้าประตู</u>

1. รูดบัตรผ่านเข้า (ACCESS CARD) Pin No:-
2. กดปุ่มใส่รหัส Pin. 4 หลัก Pin No:****
3. กดปุ่มรหัส Pin ถูกต้อง (ประตูเปิด) Access Granted
4. กดปุ่มรหัส Pin ผิด (ประตูไม่เปิด) Access Denied
5. เปิดประตูค้างนานเกินไปตามเวลาที่ตั้ง Door Not Closed
   (มีเสียงแจ้งเตือนดังด้วย BUZZER)
6. เมื่อเปิดประตูได้แล้วเวลาปิดประตูที่จอ LCD จะแสดงสถานะกลับสู่สภาวะปกติของเครื่อง

Swipe YourCard

วันที่/เดือน/ปี ชั่วโมง:นาที

<u>ออกประตู</u>

1. กดสวิทช์ (ภายในห้องข้าง ๆ ประตูควบคุม) (ประตูเปิดเอง)
   Access Granted
2. เปิดประตูค้างนานเกินเวลาที่ตั้งโปรแกรม
   (มีเสียง BUZZER แจ้งเตือนภายในตู้ควบคุม) Door Not Closed
3. เมื่อเปิดประตูได้แล้ว เวลาปิด ระบบจะ
   กลับคืนสู่สภาวะปกติของเครื่อง Swipe Your Card

วันที่/เดือน/ปี ชั่วโมง:นาที

<u>ต้องการใช้บัตรโปรแกรมเพื่อเลือกฟังก์ชั่น ซึ่งถ้าบัตรนี้สามารถใช้เป็นบัตรผ่านเข้าได้ด้วย หรือมีตัว PRO และ NOR อยู่ในบัตรเดียวกัน</u>

1. รูดบัตร Program Card + Access Card (อยู่ในใบเดียวกัน)	Pin No:-
2. กดปุ่ม Program ใส่รหัส Pin. (เพื่อเข้าสู่การโปรแกรม)	Edit Time
3. ถ้าไม่กดปุ่ม Program แล้วรหัส Pin (เพื่อต้องการผ่านเข้าห้อง)	Access Granted
4. จะเป็นไปตามขั้นตอนการเข้าประตู	

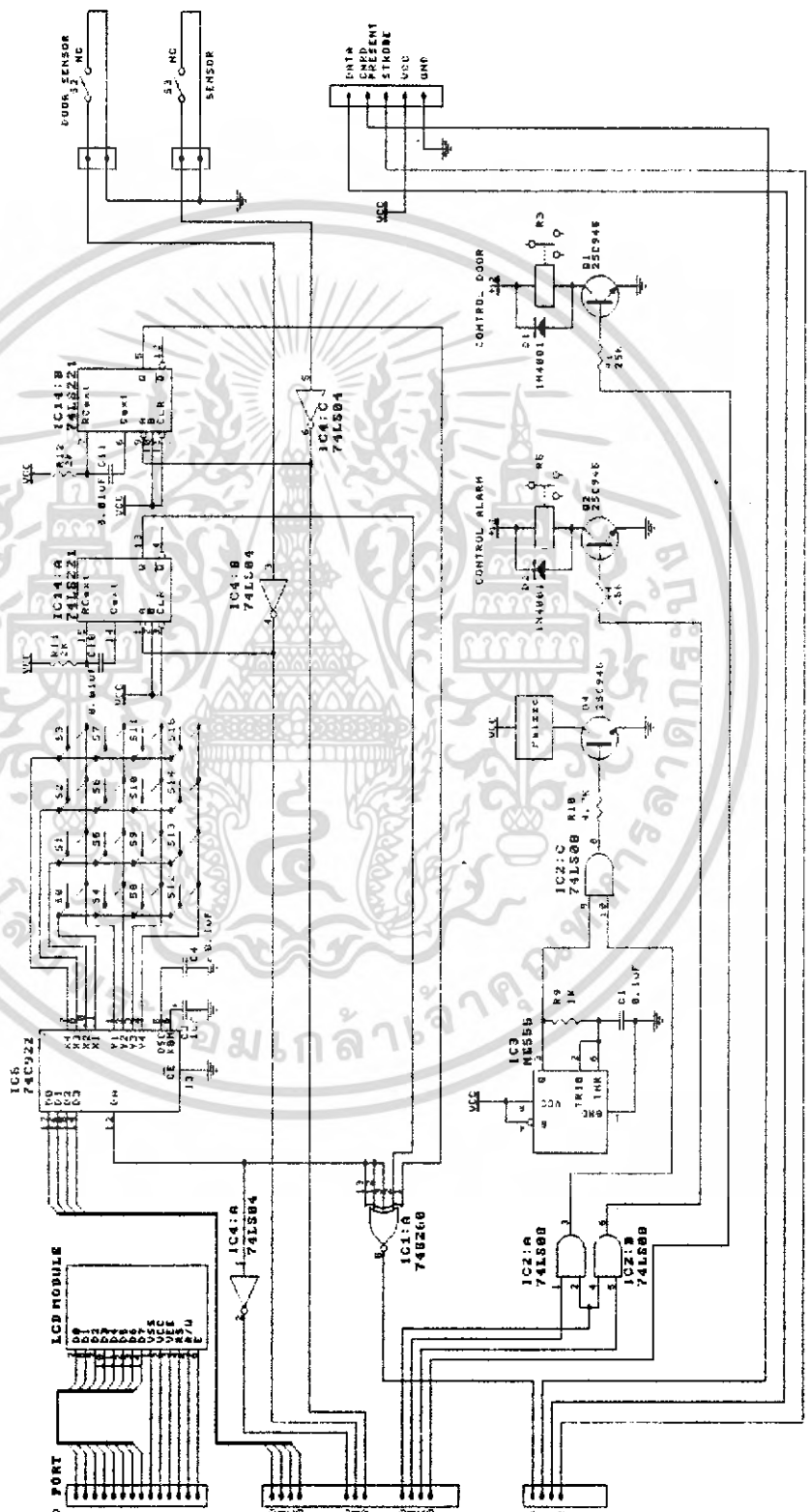
<u>กรณีใช้บัตรที่ยังไม่ได้มีการ ENTERED หรือโปรแกรมเข้าเครื่องควบคุม มารูด จะไม่สามารถเข้าห้องได้</u>

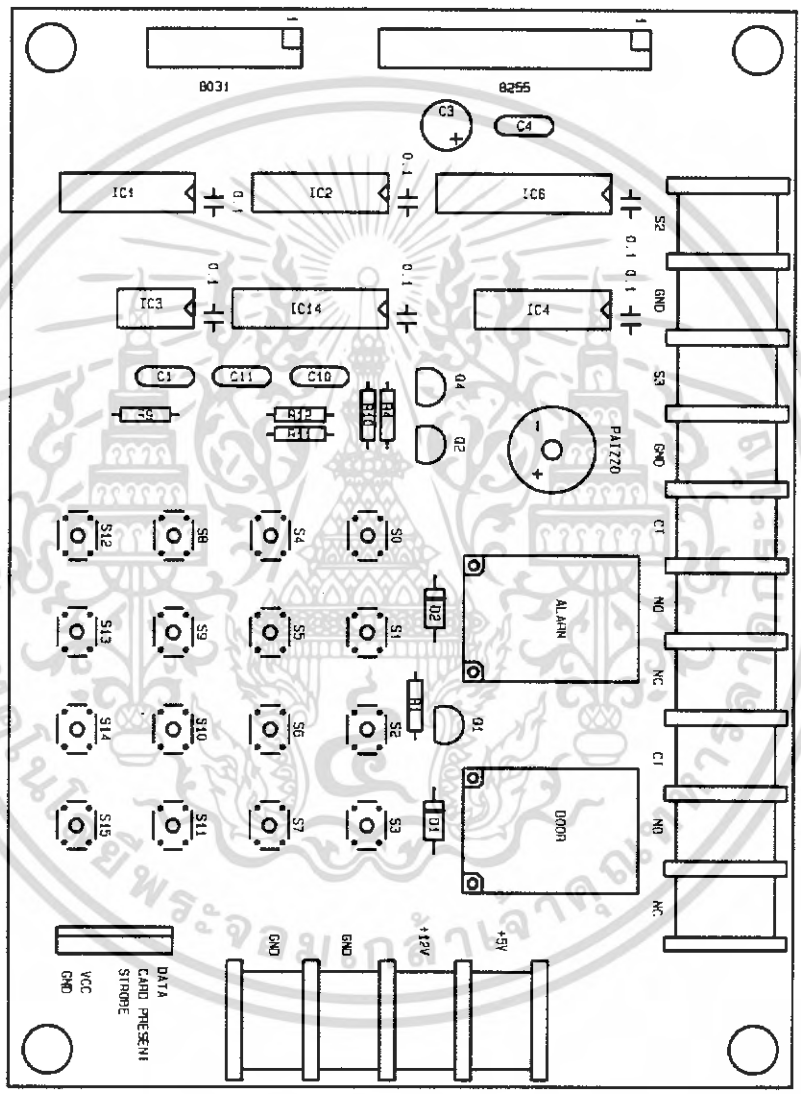
การทำงานของระบบจะเป็น

1. รูดบัตรอื่นที่ยังไม่ได้โปรแกรมเข้าเครื่อง	Invalid Card
2. ระบบจะกลับคืนสู่สภาวะปกติ	Swipe Your Card วันที่ / เดือน / ปี ชั่วโมง : นาที

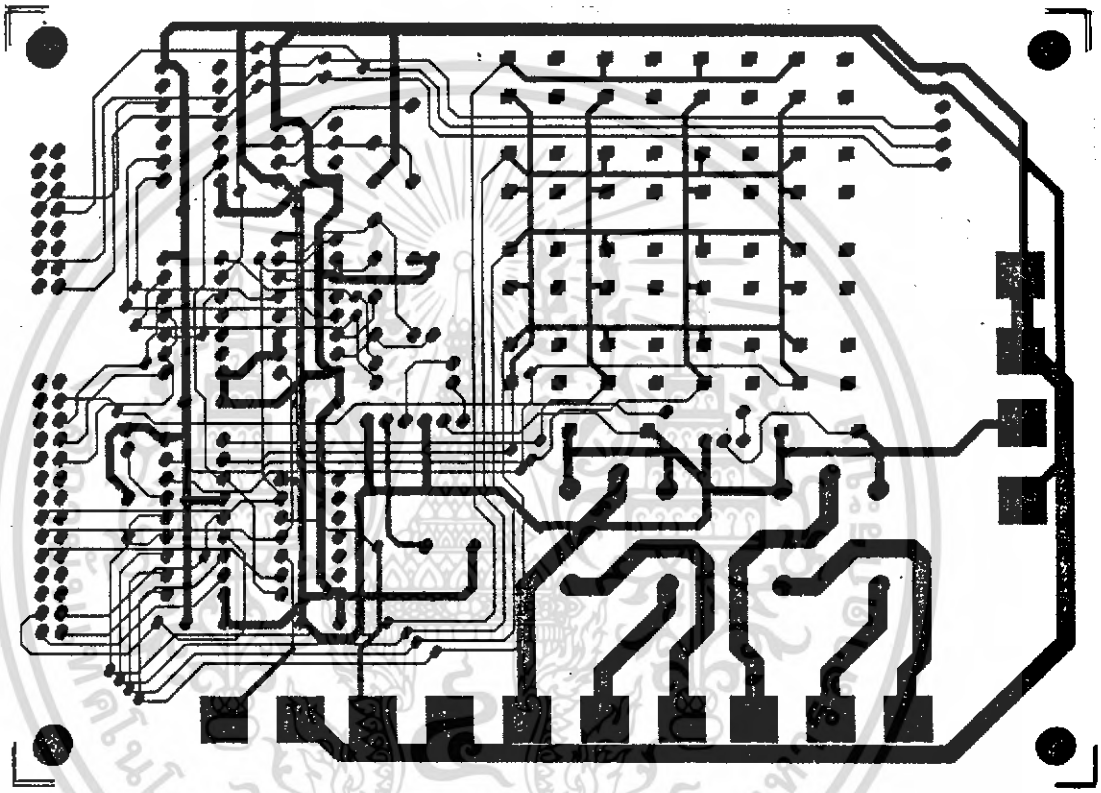
<u>กรณีมีผู้บุกรุกเพื่อจะผ่านเข้าห้องโดยไม่ได้รับอนุญาติ โดยการงัดแงะ หรือ ผังประตูเข้าไป ระบบการแจ้งจะเป็นดังนี้</u>

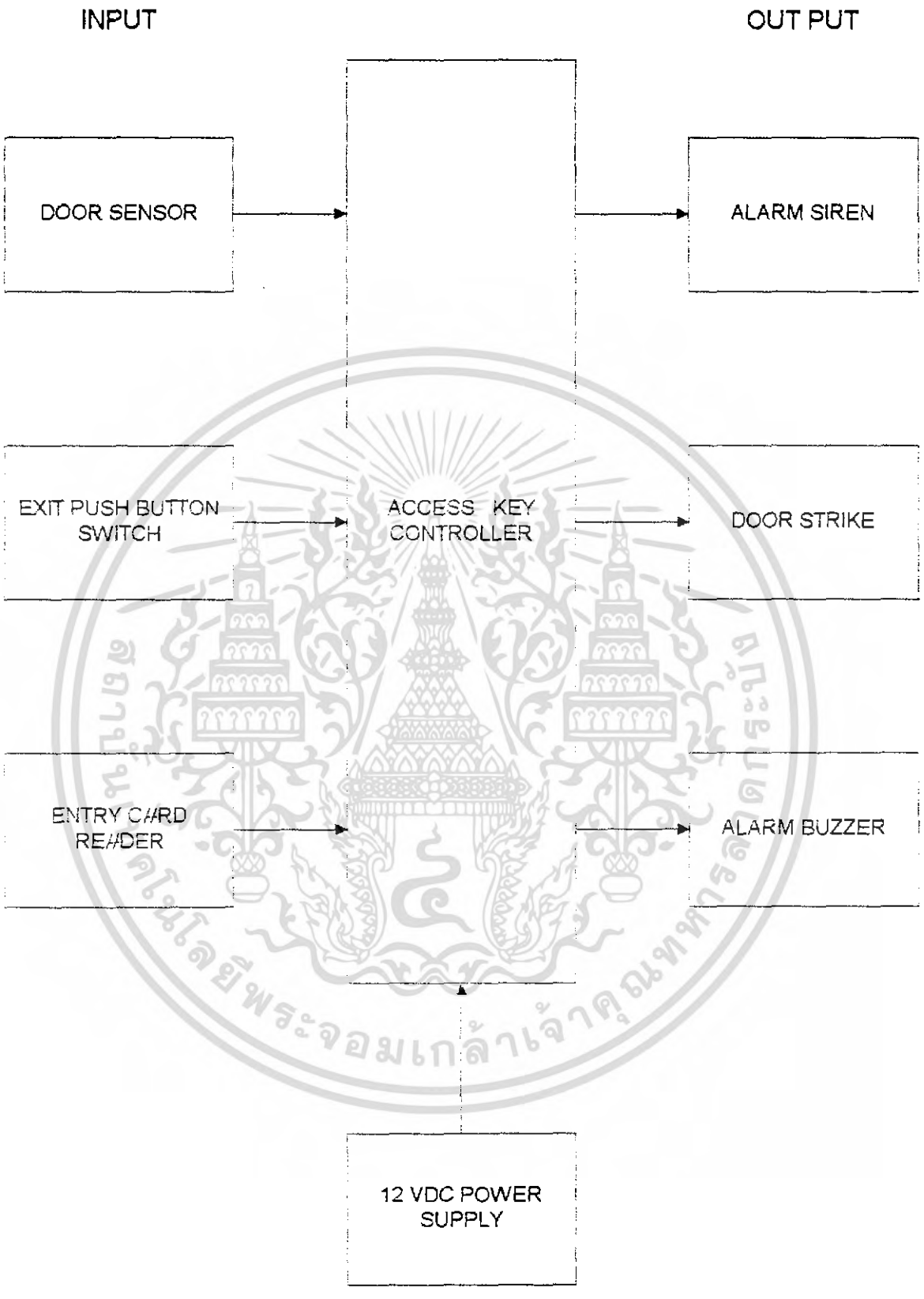
1. ถ้าผังหรืองัดแงะประตูเข้าไปได้ (จะมีสัญญาณแจ้งเตือนภายนอกตู้ควบคุม)	Door Force Open
2. เมื่อผู้ที่รับผิดชอบหรือเกี่ยวข้องรับทราบเหตุแล้ว และต้องการหยุดเสียงใช้บัตร Reset Alarm Card หรือ บัตร ตัว " A " มารูด	Pin No : -
3. ใส่รหัส Pin 4 หลัก	System Alarmed
4. กดปุ่ม DISA เพื่อหยุดเสียง	System Disarmed
5. กดปุ่ม CLEAR เพื่อกลับสู่สภาวะของเครื่องทำงานปกติ	Swipe Your Card วันที่ / เดือน / ปี ชั่วโมง : นาที



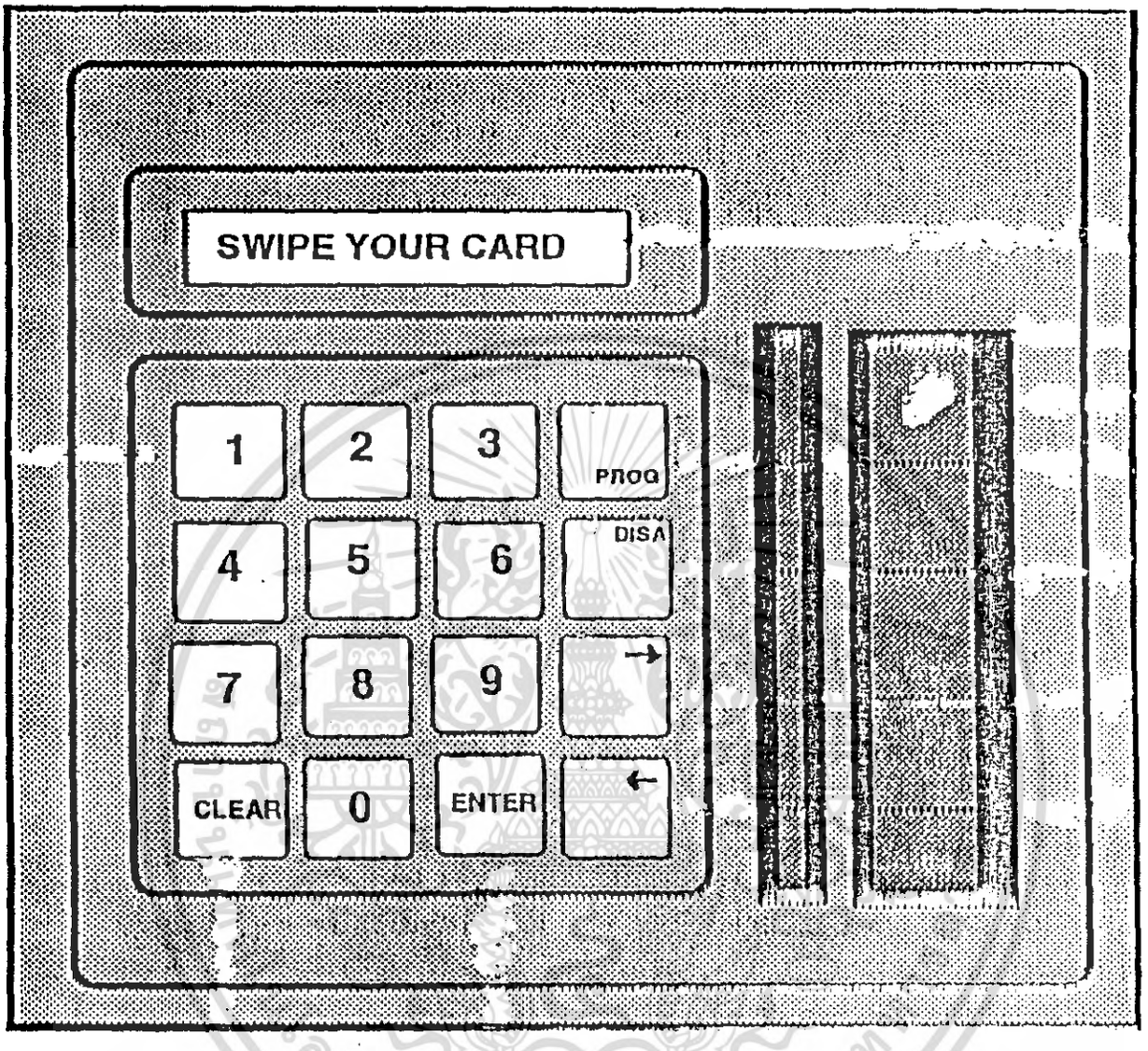


รายปริ้นต์





BLOCK DIAGRAM



ภาคผนวก

```
/*-------------------------------------------------
 *                    Access Control
 *
 * Original Author Paisarn k.
 *
 * NOTE:
 *
 *
 *
 *
 *
 -------------------------------------------------*/

#ifndef _COMPILER_51
#include "c51\8051reg.h"
#include "c51\8051io.h"
#include "c51\8051int.h"
#include "sysio.h"
#include "timer.c"
#include "lcd.c"
#include "key.c"
#include "io.c"
#include "rtc.c"
#include "card.c"
#include "func.c"
#else
#include <stdio.h>
#include <conio.h>
#define  INTERRUPT(_SER_) void interrupt
#endif
```

```
#define SYSCODE 1234
int firsttime = 1;          /* system power on the first time */
/*int cardentry = 0;*/        /* card buffer valid indicater */
int locktime  = 3;          /* Door strike time */
int beeptime  = 10;         /* Door access beep time */
char palarm   = 0;          /* port alarm buffer */
char dooropen = 0;          /* door opened */
char doormode = 0;          /* door access mode 0=on 1=off*/
char keypress = 0;          /* key pressed */
char sysprot  = 0;          /* system protect */
/*
 * The mainline is just a driver.
 */
main()
{
    int ch;

    init_timer();   /* init. timer0 */
    card_init();    /* init. Card Reader */
    lcd_init();     /* init. LCD display */
    user_init();    /* init. USER port */
    enable();       /* enable interrupt */
/*    En_timer(); */    /* enable timer0 interrupt */
    clear_alarm();  /* clear alarm */
    while(Vcardst!=CARDVAL)
        if (Vcardst==CARDERR) card_clear();
    printf("\n");
    for(ch=0; ch<40; ch++)
```

```
    {
        printf("%x ", (int)bcard[ch]);
    }

    writestr("\nKMITL ACCESS KEY");
    delay(5);       /* delay to 5 sec. */
    while(1)
    {
        writestr("\nSWIPE YOUR CARD.");
        delaycard(10);
        time_disp();
        delaycard(5);
        if (Vcardst==CARDVAL)
        {
            if (firsttime)
                first_set();
            else
                normal_card();
            card_clear();
        }
        else:
            card_clear();
    }
}

/*-------------------- Function Program --------------------*/

/*--- Function Delay ---*/
#define TIMESEC         0x0EFF  /* assigned time loop at 1 sec. */
delay(value)
```

```
int value;
/* value = delay time in sec.*/
{
    unsigned int count;

    while(value--)
        for(count=0; count<TIMESEC; count++);
}

/*--- Function Delay card ---*/
int delaycard(value)
int value;
/* value = delay time in sec.*/
/* return -1 on card entry and 0 for not card */
{
    unsigned int count;

    while(value--)
        for(count=0; count<TIMESEC; count++)
            if (Vcardst==CARDVAL) return(-1);
    return(0);
}

/*--- Function Delay key ---*/
int delaykey(value)
int value;
/* value = delay time in sec.*/
/* return scan key on key pressed and 0 for not pressed or time out */
{
    unsigned int count;
```

```
enter_pass(passindex)
int passindex;
/* passindex = pass word index */
/* return 0 on womg pass word and -1 pass word ok */
{
    int pass;

    writestr("\nPin No.: ");
    pass = getpass();
    if (pass!=load_pass())
    {
        writestr("\n Access  Denied ");
        delay(3);
        return(0);
    }
    return(-1);
}

/*--- Function Setup ---*/
first_set()
{
    int pass;

    writestr("\nSYS CODE : ");
    pass = getpass();
    if (pass!=SYSCODE)
    {
        writestr("\nWrong SYS CODE");
        delay(3);
        return;
```

```
    }
    save_card();
    function();
    firsttime = 0;
}
/*--- Function Clear Alarm ---*/
clear_alarm()
{
    palarm = 0x0D;              /* on lock. off alarm */
    user_ptwr(PORT_ALARM, palarm);
}
/*--- Function On Door Locker <open> ---*/
lock_on()

{
    palarm &= 0x07;
    user_ptwr(PORT_ALARM, palarm);
}
/*--- Function On Door Locker <close> ---*/
lock_off()
{
    if (!doormode)
    {
        palarm |= 0x08;
        user_ptwr(PORT_ALARM, palarm);
    }
}
/*--- Function Beep ON/OFF ---*/
beep(onoff)
int onoff;
```

```
/* onoff = 0 for on and 1 for off */
{
    if (!onoff) /* when on beep */
    {
        if (!doormode)
            palarm |= 0x02;
    }
    else /* when off beep */
        palarm &= 0x0D;
    user_ptwr(PORT_ALARM, palarm);
}
/*--- Function Normal Card ---*/
normal_card()
{
    if (enter_pass())

    {
        writestr("\n Access Granted ");
        lock_on();
        if (locktime<beeptime)
        {
            delay(locktime);
            lock_off();
            if (!delaydoor(beeptime-locktime))
            {
                writestr("\nDoor Not Closed");
                beep(0);    /* on beep */
            }
            while(dooropen);  /* wait for door closed */
            beep(1);    /* off beep */
```

```
        }
        else
        {
            delay(beeptime);
            if (dooropen)
            {
                writestr("\nDoor Not Closed");
                beep(0);    /* on beep */
            }
            if (delaydoor(locktime-beeptime))  /* not time out */
            {
                beep(0);            /* on beep */
                while(dooropen);    /* wait for door closed */
            }
            lock_off();
            while(dooropen);    /* wait for door closed */
            beep(1);            /* off beep */
        }
    }
}
```

```
/*-----------------------------------------------------------------
 *                        CARD DRIVER
 * Original Author Paisarn K.
 * NOTE:
 -----------------------------------------------------------------*/

#ifndef _COMPILER_51
#include "\c51\8051reg.h"
#include "\c51\8051io.h"
#include "\c51\8051int.h"
#include "\c51\8051bit.h"
#include "card.h"       /* 8051 register definitions */
#else
#include <stdio.h>
#include <conio.h>
#define  INTERRUPT(_SER_) void interrupt
#endif

#ifndef _CARD_DRIVER_
#define _CARD_DRIVER_

char Vcardbuf[50];              /* card's data ABA format buffer  */
register char Vcard1;
register char Vcardst;          /* card status                    */
register char Vbitcnt;
register char Vdatacnt;
register char Vsum;

/*--- Function Save Card info. to data base ---*/
save_card()
```

```
{
}
/*--- Function Load Card info. from data base ---*/
load_card()
{
}
/*--- Function Load pass word from data base ---*/
int load_pass()
/* return pass word of swipe card */
{
    return(2345);
}

card_init()
{
    card_clear();
    setbit(IT1);          /*  IT1  set falling edgetype for INT1      */
    setbit(EX1);          /*  EX1  enable INT1 interrupt              */
    clrbit(IE1);
}
char Btemp;          /* new mon */
char Bpar;
card_clear()
{
    Vcardst = CARDIDLE;
    Vsum = 0;
    Vbitcnt = ABAMXB;
    Vcard1  = 0;
    Vdatacnt = 0;
    Btemp = 0;          /* new mon */
```

```
    Bpar = 0x10;
}
register char bcard[50]
INTERRUPT(_IE1_) card1_hand()
{
    if (Vcardst!=CARDERR && !(P1&0x02) && Vcardst!=CARDVAL)
    {
        if (!(P1&0x01))
        {
            Btemp |= 0x20;
            Bpar = ~Bpar;
        }
        Btemp >>= 1;
        Vbitcnt--;
        if (!Vbitcnt)
        {
            if (Btemp==0x0B)
                Vcardst = CARDSTX;
            if (Btemp==0x1F)
                Vcardst = CARDETX;
            if (((Btemp&0x10)!=Bpar) && Vcardst>=CARDSTX)
            {
                Vcardst = CARDERR;
                Vdatacnt = 0;
            }
            bcard[Vdatacnt++] = Btemp;
            Vbitcnt = ABAMXB;
            Btemp = 0;
        }
        if (Vcardst==CARDETX)
```

```
            Vcardst = CARDVAL;
        }
        clrbit(IE1);
}

/*INTERRUPT(_IE1_) card_hand() asm*/
card_hand() asm
{
                mov     R0,#Vcard1
                mov     R2,Vcardst
                mov     C,CD_DATA       /* read data bit    */
                cpl     C
                mov     A,[R0]
                rrc     A
                mov     [R0],A
                anl     A,#%00011111
                cjne    R2,#CARDIDLE,card_hand4
                mov     Vcardst,#CARDIN
                sjmp    card_hand5
card_hand4
                cjne    R2,#CARDIN,card_hand8
card_hand5
                cjne    A,#0,card_hand6
                sjmp    card_hand100
card_hand6
                mov     Vcardst,#CARDSTX    /* meet some 1's bit */
                mov     Vbitcnt,#ABAMXB
                sjmp    card_hand10
card_hand8
                cjne    R2,#CARDSTX,card_hand50
```

```
card_hand10
        dec     Vbitcnt
        mov     R3,Vbitcnt
        cjne    R3,#0,card_hand100
        mov     DPTR,#Vcardbuf      /* save a data    */
        mov     A,DPL
        add     A,Vdatacnt
        mov     DPL,A
        mov     A,[R0]
        anl     A,#ABABIT
        cjne    A,#STX,card_hand12
        mov     Vcardst,#CARDDAT
        sjmp    card_hand45
card_hand12
        cjne    A,#ETX,card_hand15      /* card valid */
        mov     Vcardst,#CARDVAL
card_hand15
        mov     R3,#ABAMXB          /* check odd parity !!! */
        mov     R4,#0
card_hand20
        rrc     A
        jnc     card_hand25
        inc     R4
card_hand25
        djnz    R3,card_hand20
        mov     A,R4
        jb      A.0,card_hand40
        mov     Vcardst,#CARDERR
        sjmp    card_hand100
card_hand40
```

```
            mov     A,[R0]
            anl     A,#%00001111
            movx    [DPTR],A
            inc     Vdatacnt
card_hand45
            xrl     Vsum,A
            mov     Vbitcnt,#ABAMXB
            sjmp    card_hand100
card_hand50
            cjne    R2,#CARDDAT,card_hand55
            sjmp    card_hand10
card_hand55
            cjne    R2,#CARDVAL,card_hand60
            sjmp    card_hand10
card_hand60
            cjne    R2,#CARDERR,card_hand100
            sjmp    card_hand100
card_hand100
            CLR     IE1
            ret
}

#endif
~
```

```
/*---------------------------------------------------------------
 *                        FUNCTION MENU
 * Original Author Paisarn K.
 * NOTE:
 ---------------------------------------------------------------*/

#ifndef _COMPILER_51
#include "\c51\8051reg.h"
#include "\c51\8051io.h"
#include "\c51\8051int.h"
#else
#include <stdio.h>
#include <conio.h>
#define INTERRUPT(_SER_) void interrupt
#endif

#ifndef _FUNCTION_
#define _FUNCTION_

/*--- Function Set RTC ---*/
int set_time()
/* return 0 on complete and KEY_CLR on not complete */
{
    int ch;
    char date[10];

    writestr("\n Edit Time/Date ");
    while((ch=readch())!=KEY_ENT)
    {
        switch(ch)
```

```
        {
            case KEY_CLR: return(KEY_CLR);
            case KEY_F2:  return(ch);
            case KEY_F3:  return(ch);
            case KEY_F4:  return(ch);
        }

    }
    writestr("\nTime: ");
    rtc_stime(getnum());
    writestr("\nDate: ");
    rtc_sdate(readnum(date));
    lcd_mode(LCD_CUROFF);
    writestr("\n    << Save >>    ");
    if (delaykey(1)==KEY_CLR) return(KEY_CLR);
    return(0);
}

/*--- Function Check Card Info. ---*/
int card_info()
/* return 0 on complete and KEY_CLR on not complete */
{
    int ch;

    writestr("\nCheck Card Info.");
    while((ch=readch())!=KEY_ENT)
    {
        switch(ch)
        {
            case KEY_CLR: return(KEY_CLR);
```

```
            case KEY_F2:  return(ch);
            case KEY_F3:  return(ch);
            case KEY_F4:  return(ch);
        }

    }
/* reserved for card reader */
    while(readch()!=KEY_CLR);
    writestr("\n    << Save >>    ");
    if (delaykey(1)==KEY_CLR) return(KEY_CLR);
    return(0);
}

/*--- Function Enter New Card ---*/
int enter_card()
/* return 0 on complete and KEY_CLR on not complete */
{
    int ch;

    writestr("\n Enter New Card ");
    while((ch=readch())!=KEY_ENT)
    {
        switch(ch)
        {
            case KEY_CLR: return(KEY_CLR);
            case KEY_F2:  return(ch);
            case KEY_F3:  return(ch);
            case KEY_F4:  return(ch);
        }
```

```
    }
/* reserved for card reader */
    while(readch()!=KEY_CLR);
    writestr("\n   << Save >>   ");
    if (delaykey(1)==KEY_CLR) return(KEY_CLR);
    return(0);
}

/*--- Function Delete Exist Card ---*/
int del_card()
/* return 0 on complete and KEY_CLR on not complete */
{
    int ch;

    writestr("\nDel. Exist Card ");
    while((ch=readch())!=KEY_ENT)
    {
        switch(ch)
        {
            case KEY_CLR: return(KEY_CLR);
            case KEY_F2:  return(ch);
            case KEY_F3:  return(ch);
            case KEY_F4:  return(ch);
        }

    }
/* reserved */
    while(readch()!=KEY_CLR);
    writestr("\n   << Save >>   ");
    if (delaykey(1)==KEY_CLR) return(KEY_CLR);
```

```
    return(0);
}

/*--- Function Door Strike Time ---*/
extern int locktime;
int door_time()
/* return 0 on complete and KEY_CLR on not complete */
{
    int ch;
    char string[18];

    writestr("\nDoor Stike Time ");
    while((ch=readch())!=KEY_ENT)
    {
        switch(ch)
        {
            case KEY_CLR: return(KEY_CLR);
            case KEY_F2:  return(ch);
            case KEY_F3:  return(ch);
            case KEY_F4:  return(ch);
        }

    }
    sprintf(string, "\nStr Tm %03d : ", locktime);
    writestr(string);
    locktime = getdelay();
    if (locktime<3)
        locktime = 3;
    if (locktime>250)
        locktime = 250;
```

```
    sprintf(string, "\nStr Tm %03d :     ", locktime);
    writestr(string);
    while(readch()!=KEY_CLR);
    writestr("\n    << Save >>    ");
    if (delaykey(1)==KEY_CLR) return(KEY_CLR);
    return(0);
}

/*--- Function  Set Alarm Time---*/
int alarm_time()
/* return 0 on complete and KEY_CLR on not complete */
{
    int ch;

    writestr("\nAlm. Output Dur.");
    while((ch=readch())!=KEY_ENT)
    {
        switch(ch)
        {
            case KEY_CLR: return(KEY_CLR);
            case KEY_F2:  return(ch);
            case KEY_F3:  return(ch);
            case KEY_F4:  return(ch);
        }

    }
/* reserved */
    while(readch()!=KEY_CLR);
    writestr("\n    << Save >>    ");
    if (delaykey(1)==KEY_CLR) return(KEY_CLR);
```

```
    return(0);
}

/*--- Function  Set Busser Time---*/
extern int beeptime;
int buzzer_time()
/* return 0 on complete and KEY_CLR on not complete */
{
    int ch;
    char string[18];

    writestr("\n Acs. Beep Dur. ");
    while((ch=readch())!=KEY_ENT)
    {
        switch(ch)
        {
            case KEY_CLR: return(KEY_CLR);
            case KEY_F2:  return(ch);
            case KEY_F3:  return(ch);
            case KEY_F4:  return(ch);
        }

    }
    sprintf(string, "\nBp Dur %03d : ", beeptime);
    writestr(string);
    beeptime = getdelay();
    if (beeptime<10)
        beeptime = 10;
    if (beeptime>250)
        beeptime = 250;
```

```
    sprintf(string, "\nBp Dur %03d :     ", beeptime);
    writestr(string);
    while(readch()!=KEY_CLR);
    writestr("\n   << Save >>   ");
    if (delaykey(1)==KEY_CLR) return(KEY_CLR);
    return(0);
}

/*--- Function  Set Alarm Mode---*/
int alarm_mode()
/* return 0 on complete and KEY_CLR on not complete */
{
    int ch;

    writestr("\nAlm. Mode ON/OFF");
    while((ch=readch())!=KEY_ENT)
    {
        switch(ch)
        {
            case KEY_CLR: return(KEY_CLR);
            case KEY_F2:  return(ch);
            case KEY_F3:  return(ch);
            case KEY_F4:  return(ch);
        }

    }
/* reserved */
    while(readch()!=KEY_CLR);
    writestr("\n   << Save >>   ");
    if (delaykey(1)==KEY_CLR) return(KEY_CLR);
```

```
    return(0);
}

/*--- Function Reset Alarm ---*/
int alarm_reset()
/* return 0 on complete and KEY_CLR on not complete */
{
    int ch;

    writestr("\n System Alarmed ");
    while((ch=readch())!=KEY_ENT)
    {
        switch(ch)
        {
            case KEY_CLR: return(KEY_CLR);
            case KEY_F2:  return(ch);
            case KEY_F3:  return(ch);
            case KEY_F4:  return(ch);
        }

    }
/* reserved */
    while(readch()!=KEY_CLR);
    writestr("\n   << Save >>   ");
    if (delaykey(1)==KEY_CLR) return(KEY_CLR);
    return(0);
}

/*--- Function Door Access Mode ---*/
int access_mode()
```

```
/* return 0 on complete and KEY_CLR on not complete */
{
    int ch;

    writestr("\nDoor Access Mode");
    while((ch=readch())!=KEY_ENT)
    {
        switch(ch)
        {
            case KEY_CLR: return(KEY_CLR);
            case KEY_F2:  return(ch);
            case KEY_F3:  return(ch);
            case KEY_F4:  return(ch);
        }

    }
    while(1)
    {
        if (!doormode)
            writestr("\nDoor Acs : ON   ");
        else
            writestr("\nDoor Acs : OFF  ");
        ch = readch();
        if (ch==KEY_F3)
            doormode = doormode ? 0 : 1;
        if (ch==KEY_ENT)
            break;
    }
    if (doormode)
        lock_on();
```

```
    writestr("\n   << Save >>   ");
    if (delaykey(1)==KEY_CLR) return(KEY_CLR);
    return(0);
}

/*--- Function View Transaction ---*/
int view_tran()
/* return 0 on complete and KEY_CLR on not complete */
{
    int ch;

    writestr("\nView Transaction");
    while((ch=readch())!=KEY_ENT)
    {
        switch(ch)
        {
            case KEY_CLR: return(KEY_CLR);
            case KEY_F2:  return(ch);
            case KEY_F3:  return(ch);
            case KEY_F4:  return(ch);
        }

    }
/* reserved */
    while(readch()!=KEY_CLR);
    writestr("\n   << Save >>   ");
    if (delaykey(1)==KEY_CLR) return(KEY_CLR);
    return(0);
}
```

```
function()
{
    int func, key;
    func = 0;
    while(1)
    {
        switch(func)
        {
            case 0: key = set_time(); break;
            case 1: key = card_info(); break;
            case 2: key = enter_card(); break;
            case 3: key = del_card(); break;
            case 4: key = door_time(); break;
            case 5: key = alarm_time(); break;
            case 6: key = buzzer_time(); break;
            case 7: key = alarm_mode(); break;
            case 8: key = alarm_reset(); break;
            case 9: key = access_mode(); break;
            case 10: key = view_tran(); break;
        }
        if (key==KEY_CLR) break;
        if (key==KEY_F3) func++;
        if (key==KEY_F4) func--;
        if (key==KEY_F2) func = 8;
        if (func<0) func = 10;
        if (func>10) func = 0;
    }
}
#endif
```

```
/*-----------------------------------------------------------------
 *                      INTERRUPT DRIVER
 * Original Author Paisarn K.
 * NOTE:
 -----------------------------------------------------------------*/

#ifndef _COMPILER_51
#include "\c51\8051reg.h"
#include "\c51\8051io.h"
#include "\c51\8051int.h"
#else
#include <stdio.h>
#include <conio.h>
#define  INTERRUPT(_SER_) void interrupt
#endif

#ifndef _INTERRUPT_DRIVER_
#define _INTERRUPT_DRIVER_

#define INT_TIMER0
#define INT_TIMER1
#define INT_IE0
#define INT_IE1
#define INT_SER

/*--- Enable interrupt ---*/
int_enable(char intmask)
/* intmask = interrupt type */
{
}
```

```
/*--- Disable interrupt ---*/
int_disable(char intmask)
/* intmask = interrupt type */
{
}
/*--- End of interrupt ---*/
int_end(char intmask)
/* intmask = interrupt type */
{
}
#endif
~
/*-------------------------------------------------------------
 *                IO DRIVER
 * Original Author Pausarn K.
 * NOTE:
 -------------------------------------------------------------*/

#ifndef _COMPILER_51
#include "\c51\8051reg.h"
#include "\c51\8051io.h"
#include "\c51\8051int.h"
#include "sysio.h"
#else
#include <stdio.h>
#include <conio.h>
#define  INTERRUPT(_SER_) void interrupt
#endif

/* Assigned Function Init. SYSTEM IO PORT */
```

```
user_init()
{
        /* set mode of 8255 */
        *(char *)PORT_USER1_M = 0x9A; /* A=in, B=in, Ch=in, Cl=out */
        *(char *)PORT_USER2_M = 0x9A; /* A=in, B=in, Ch=in, Cl=out */
        /* clear output port */
        *(char *)PORT_USER1_C = 0x00;
        *(char *)PORT_USER2_C = 0x00;
}

/*--- Write user port ---*/
user_ptwr(port, ch)
char *port;
char ch;
/* port = port address to write */
/* ch   = data to write */
{
        *port = ch;
}

/*--- Read user port ---*/
int user_ptrd(port)
char *port;
/* port = port address to read */
/* return = value to read */
{
        return(*port);
}
~
```

```
/*-----------------------------------------------------------------
 *                        KEY DRIVER
 * Original Author Paisarn K.
 * NOTE:
 -----------------------------------------------------------------*/

#ifndef _COMPILER_51
#include "\c51\8051reg.h"
#include "\c51\8051io.h"
#include "\c51\8051int.h"
#else
#include <stdio.h>
#include <conio.h>
#define  INTERRUPT(_SER_) void interrupt
#endif

#ifndef _KEY_
#define _KEY_
/*--- Assigned keyboard driver function prototype ---*/
#define KEY_1       0x00      /* define key code */
#define KEY_2          0x01
#define KEY_3          0x02
#define KEY_4          0x04
#define KEY_5          0x05
#define KEY_6          0x06
#define KEY_7          0x08
#define KEY_8          0x09
#define KEY_9          0x0A
#define KEY_0          0x0D
#define KEY_ENT        0x0E
```

```
#define KEY_CLR      0x0C
#define KEY_F1               0x03
#define KEY_F2               0x07
#define KEY_F3               0x0B
#define KEY_F4               0x0F
#define KEY_BIT      0x0F          /* define key bit field */
#define KEY_DELAY    0xFF
#define KEY1         '1'       /* define scan key */
#define KEY2         '2'
#define KEY3         '3'
#define KEY4         '4'
#define KEY5         '5'
#define KEY6         '6'
#define KEY7         '7'
#define KEY8         '8'
#define KEY9         '9'
#define KEY0         '0'
#define NUM_OFF      0          /* assigned display number off */
#define NUM_ON       0          /* assigned display number on */


/*--- Read character from key ---*/
/* Note. wait until keyboard pressed */
char readch()
/* return character code (ASCII) */
{
        char key;
        unsigned int count;


    key = 0;
    lcd_mode(LCD_CURON);
```

```
    for(count=0; count<KEY_DELAY; count++);
    while(user_ptrd(KEY_PORTC)&0x01);
    for(count=0; count<KEY_DELAY; count++)
        key = user_ptrd(KEY_PORT) & KEY_BIT;
    while(!(user_ptrd(KEY_PORTC)&0x01));
    switch(key)
    {
        case KEY_1: return(KEY1);
        case KEY_2: return(KEY2);
        case KEY_3: return(KEY3);
        case KEY_4: return(KEY4);
        case KEY_5: return(KEY5);
        case KEY_6: return(KEY6);
        case KEY_7: return(KEY7);
        case KEY_8: return(KEY8);
        case KEY_9: return(KEY9);
        case KEY_0: return(KEY0);
        default: return(key);


/*--- Read Number from key <8digit> ---*/
/* Note. wait until <Enter> pressed */
char *readnum(str, disp)
char *str;
char disp;
/* str = pointer to user buffer */
/* disp= display number on/off */
/* return pointer of buffer */
{
        int count;
```

```
  char *spoint;


  spoint = str;
  count = 0;
  while(1)
  {
while((str[count] = readch())==KEY_F1 || str[count]==KEY_F2 ||
    str[count]==KEY_F3); /*check func-key*/
if(str[count]==KEY_ENT || str[count]==KEY_CLR)
{
    if (str[count]==KEY_CLR) count = 0;
    str[count] = 0;
    break;
}
if !(count==0 && str[count]==KEY_F4))
            {
    if (str[count]==KEY_F4)
        lcd_mode(LCD_CURL);
    else
    {
        if (disp==NUM_OFF)
            writech('*');
        else
            if (count<8)
                writech(str[count]);
    }
    if(str[count]==KEY_F4)
    {
        writech(' ');
        lcd_mode(LCD_CURL);
```

```
                count -= 2;
            }
            if (count<8) count++;
                }
        }
        return(spoint);
}

/*--- Get pass word <4digit> ---*/
int getpass()
{
        int count;
    char str[6];

        count = 0;
        while(1)
        {
       while((str[count] = readch())==KEY_F1 || str[count]==KEY_F2 ||
           str[count]==KEY_F3); /*check func-key*/
       if(str[count]==KEY_ENT || str[count]==KEY_CLR)
       {
           if (str[count]==KEY_CLR) count = 0;
           str[count] = 0;
           break;
       }
       if(!(count==0 && str[count]==KEY_F4))
                {
           if (str[count]==KEY_F4)
             lcd_mode(LCD_CURL);
           else
```

```
            {
                if(count<4)
                    writech('*');
            }
            if(str[count]==KEY_F4)
            {
                writech(' ');
                lcd_mode(LCD_CURL);
                count -= 2;
            }
            if (count<4)
                count++;
                }
            }
    return(atoi(str));
}

/*--- Get number <4digit> ---*/
int getnum()
{
        int count;
    char str[6];

        count = 0;
        while(1)
        {
       while((str[count] = readch())==KEY_F1 || str[count]==KEY_F2 ||
          str[count]==KEY_F3); /*check func-key*/
      if(str[count]==KEY_ENT || str[count]==KEY_CLR)
        {
```

```
            if (str[count]==KEY_CLR) count = 0;
            str[count] = 0;
            break;
        }
        if(!(count==0 && str[count]==KEY_F4))
                {
            if (str[count]==KEY_F4)
                lcd_mode(LCD_CURL);
            else
            {
                if(count<4)
                    writech(str[count]);
            }
            if(str[count]==KEY_F4)
            {
                writech(' ');
                lcd_mode(LCD_CURL);
                count -= 2;
            }
            if (count<4)
                count++;
                }
        }
    return(atoi(str));
}

/*--- Get Delay value <3digit> ---*/
int getdelay()
{
    int count;
```

```
char str[6];

    count = 0;
    while(1)
    {
  while((str[count] = readch())==KEY_F1 || str[count]==KEY_F2 ||
     str[count]==KEY_F3); /*check func-key*/
  if(str[count]==KEY_ENT || str[count]==KEY_CLR)
  {
     if (str[count]==KEY_CLR) count = 0;
     str[count] = 0;
     break;
  }
  if(!(count==0 && str[count]==KEY_F4))
          {
     if (str[count]==KEY_F4)
       lcd_mode(LCD_CURL);
     else
     {
        if(count<3)
           writech(str[count]);
     }
     if(str[count]==KEY_F4)
     {
        writech(' ');
        lcd_mode(LCD_CURL);
        count -= 2;
     }
     if (count<3)
        count++;
```

```
                }
            }
    return(atoi(str));
}

/*--- Check key pressed status ---*/
int key_press()
/* return -1 on key pressed and 0 for other */
{
    if (user_ptrd(KEY_PORTC)&0x01)
        return(0);
    return(-1);
}

#endif
```

```
/*-----------------------------------------------------------------
 *                        LCD DRIVER
 * Original Author Paisarn K.
 * NOTE: Assigned LCD driver function prototype (16char/1line)
 -----------------------------------------------------------------*/

#ifndef _COMPILER_51
#include "\c51\8051reg.h"
#include "\c51\8051io.h"
#include "\c51\8051int.h"
#else
#include <stdio.h>
#include <conio.h>
#define  INTERRUPT(_SER_) void interrupt
#endif

/*--- Assigned LCD driver function prototype (16char/1line) ---*/
#define MAXCHAR         16  /* display 16 charater per 1 line */
#define LCD_DELAY          0x0FFF /* assigned LCD delay loop */
#define LCD_NORMAL      0   /* assigned value for normal display mode */
#define LCD_LEFT        1   /* assigned value for display from left */
#define LCD_RIGHT       2   /* assigned value for display from right */
#define LCD_CURON       3   /* assigned value for cursor on */
#define LCD_CUROFF      4   /* assigned value for cursor off */
#define LCD_CURR        5   /* assigned value for move cursor right */
#define LCD_CURL        6   /* assigned value for move cursor left */

/*--- Initialize LCD module ---*/
lcd_init()
{
```

```
        /* init. LCD */
        lcd_write(LCD_PCOMM, 0x38);   /* port 8 bit, 16char/1line, 5*7dots */
    lcd_write(LCD_PCOMM, 0x0F);   /* on display, on cursor, cursor blink */
        lcd_write(LCD_PCOMM, 0x06);   /* ddram address increment ,cursor shift right*/
    lcd_clr();
    lcd_mode(LCD_NORMAL);
}
/*--- Write data to LCD ---*/
int numchar = MAXCHAR;
lcd_write(port, ch)
char *port, ch;
/* port = port to write <see define> */
/* ch   = data to write */
{
        int loop;
    char addr;

    addr = 0x80;
    for(loop=0; addr&0x80 && loop<LCD_DELAY; loop++)
        addr = *(char *)LCD_PSTAT;
        *port = ch;
}

/*--- Clear LCD display and move cursor to home position ---*/
lcd_clr()
{
        lcd_write(LCD_PCOMM, 0x01); /* clear display and move cursor to home position */
    numchar = MAXCHAR;
}
/*--- Set display mode ---*/
```

```
int lcd_mode(mode)
int mode;
/* mode = value of display mode */
/* return 0 on success and -1 on failed */
{
    switch(mode)
    {
                case LCD_NORMAL: /* ddram address increment, cursor shift right */
                        lcd_write(LCD_PCOMM, 0x06);
                        break;
                case LCD_LEFT:      /* Display shift from left */
                        lcd_write(LCD_PCOMM, 0x05);
                        break;
                case LCD_RIGHT: /* Display shift from right */
                        lcd_write(LCD_PCOMM, 0x07);
                        break;
        case LCD_CUROFF:
                lcd_write(LCD_PCOMM, 0x0C);
                break;
        case LCD_CURON:
                lcd_write(LCD_PCOMM, 0x0F);
                break;
        case LCD_CURR:  /* move cursor right 1 column */
                lcd_write(LCD_PCOMM, 0x14);
                if (numchar==8)
                    lcd_write(LCD_PCOMM, 0xC0);
                numchar--;
                break;
        case LCD_CURL:  /* move cursor legt 1 column */
                lcd_write(LCD_PCOMM, 0x10);
```

```
            if (numchar==8)
                lcd_write(LCD_PCOMM, 0x87);
            numchar++;
            break;
        default: return(-1);
    }
    return(0);
}

/*--- Write character to LCD ---*/
writech(ch)
char ch;
/* ch = character write to LCD */
{
    if (ch=='\n' || !numchar)
    {
            lcd_clr();
    }
        else
    {
        if (numchar==8)
            lcd_write(LCD_PCOMM, 0xC0);
        numchar--;
            lcd_write(LCD_PWR, ch);
    }
}

/*--- Write string to LCD ---*/
writestr(str)
char *str;
```

```
/* str = pointer of string for write to LCD */
{
    lcd_mode(LCD_CUROFF);
    while(*str)
        writech(*str++);
}
```

```
/*-----------------------------------------------------------------
 *                          RTC DRIVER
 * Original Author Paisarn K.
 * NOTE:
 -----------------------------------------------------------------*/

#ifndef _COMPILER_51
#include "\c51\8051reg.h"
#include "\c51\8051io.h"
#include "\c51\8051int.h"
#else
#include <stdio.h>
#include <conio.h>
#define  INTERRUPT(_SER_) void interrupt
#endif

/*--- Assigned Real time clock driver function prototype ---*/
/*--- Initialize RTC ---*/
rtc_init()
{
}
/*--- Set date ---*/
rtc_sdate()
{
}
/*--- Get date ---*/
rtc_date()
{
}
/*--- Set time ---*/
```

```
rtc_stime(hour, min, sec)
int hour, min, sec;
/* hour = hour to set */
/* min  = minute to set */
/* sec  = second to set */
{
}
/*--- Get time() ---*/
rtc_time()
{
}
```

```
#ifndef _COMPILER_51
#include "\c51\8051reg.h"
#include "\c51\8051io.h"
#include "\c51\8051int.h"
#include "sysio.h"
#include "sysvar.h"
#else
#include <stdio.h>
#include <conio.h>
#define  INTERRUPT(_SER_) void interrupt
#endif
/*--- START OF TIMER PROGRAM ---*/
/*--- This routine for initialize timer for use to broadcast ---*/
#define DELAY       1       /* define timer 0 interrupt value */
Init_timer(void)
{
```

```
    TMOD |= 0x02;              /* T0=8 bit auto-reload */
    TH0   = DELAY;             /* Timer 0 reload value */
        TL0   = DELAY;             /* Timer 0 initial value */
        IE    |= 0x02;             /* enable timer 0 interrupt */
    TCON &= 0xDF;              /* set end of timer 0 interrupt */
}

/*--- This routine for enable timer interrupt ---*/
En_timer(void)
{
    TCON |= 0x10;              /* On timer 0 */
}

/*--- This routine for disable timer interrupt ---*/
Dis_timer(void)
{
    TCON &= 0xEF;              /* Off timer 0*/
}

/*--- This routine for tell CPU interrupt processed ---*/
End_timer(void)
{
    TCON &= 0xDF;              /* set end of timer 0 interrupt */
}

/*--- This routine for report temp (broadcast). to Monitor program ---*/
/* timer interrupt service routine for broadcast */
#define TIMETICK 0x0FFF        /* define tick time for access */
unsigned int ttick = 0;         /* timer scale */
INTERRUPT(_TF0_) timerh()      /* interrupt service routine */
```

```
{
    char flag;

    Dis_timer();
    disable();                  /* disable all interrupt */
    if (!ttick)
        {
      flag = user_ptrd(PORT_DOOR);
      if (!(flag&0x01))  /* key pressed */
         keypress++;
      else
         keypress = 0;
      if (!(flag&0x02))  /* door open */
         dooropen++;
      else
         dooropen = 0;  /* door close */
      if (!(flag&0x04))  /* system protect */
         sysprot++;
      else
         sysprot = 0;
      if (!(flag&0x08));  /* card entry */
      ttick = 0xFFFF-TIMETICK;
    }
    ttick++;
    enable();                   /* enable all interrupt */
    En_timer();
        End_timer();                /* end of timer 0 interrupt */
}
/*--- END OF TIMER PROGRAM ---*/
```

DMC161C

• Display Format(16character × 1line) • Display Fonts(5×8dots) • Driving Method(1/8D)

ABSOLUTE MAXIMUM RATINGS

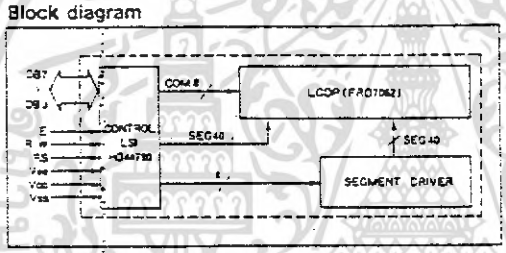
Item	Symbol	Test Condition	Standard Value min.	typ.	max.	Unit
Power Supply Voltage for Logic	Vcc~Vss	——	0	—	7	V
Power Supply Voltage for LCD Drive	Vcc~Vee	——	0	—	13.5	V
Inout Voltage	Vi	——	Vss	—	Vcc	V
Operating Temperature	Ta	——	0	—	+50	°C
Storage Temperature	Tstg	——	−20	—	+70	°C

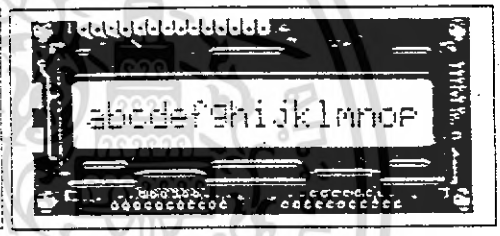
ELECTRICAL CHARACTERISTICS

Item	Symbol	Test Condition	Standard Value min.	typ.	max.	Unit
Input "High" Voltage	V_{IH}	——	2.2	--	Vcc	V
Input "Low" Voltage	V_{IL}	——	-0.3	—	0.6	V
Output "High" Voltage	V_{OH}	$-I_{OH}$=0.205mA	2.4	--	--	V
Output "Low" Voltage	V_{OL}	I_{OH}=1.2mA	—	-	0.4	V
Power Supply Current	Icc	Vcc=5.0V	--	0.5	2.0	mA

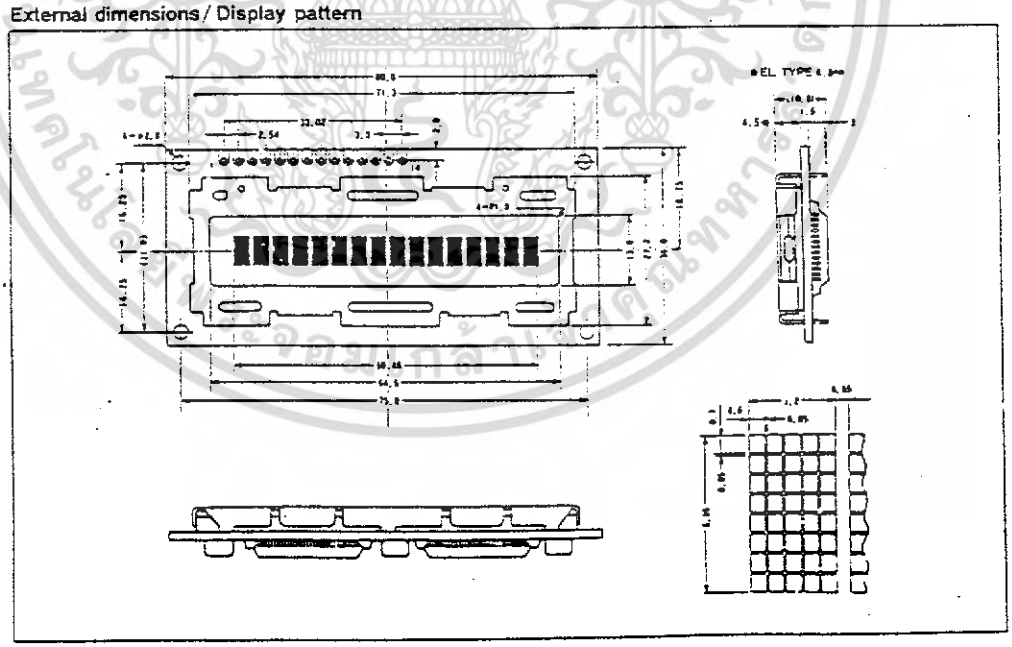
* Vcc=5.0V±5%, Ta=25°C

Block diagram





External dimensions / Display pattern



DMC162

●Display Format(16character ×2line) ● Display Fonts(5×8dots) ● Driving method(1/16D)

ABSOLUTE MAXIMUM RATINGS

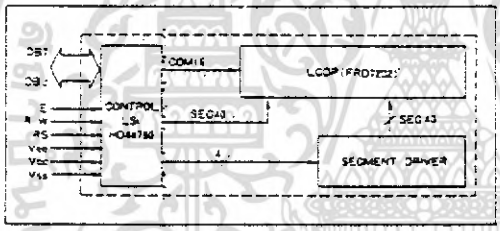
Item	Symbol	Test Condition	Standard Value min.	typ.	max.	Unit
Power Supply Voltage for Logic	Vcc-Vss	—	0	—	7	V
Power Supply Voltage for LCD Drive	Vcc-Vee	—	0	—	13.5	V
Input "High" Voltage	VI	—	Vss	—	Vcc	V
Operating Temperature	Ta		0		+50	°C
Storage Temperature	Tstg	—	−20		+70	°C

ELECTRICAL CHARACTERISTICS

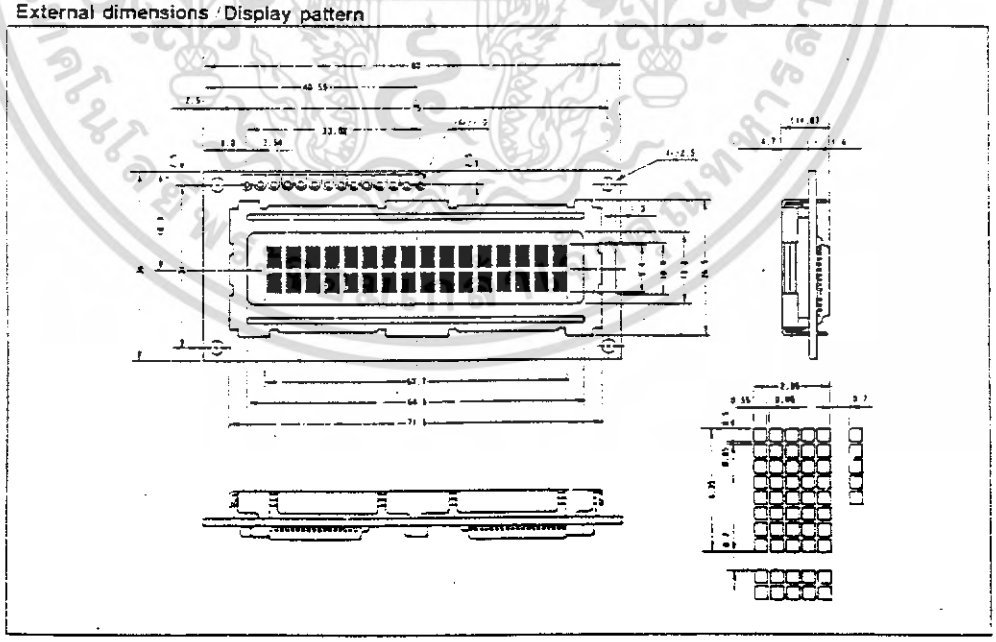
Item	Symbol	Test Condition	Standard Value min.	typ.	max.	Unit
Input Voltage	VIH	—	2.2	—	Vcc	V
Input "Low" Voltage	VIL	—	−0.3	—	0.6	V
Output "High" Voltage	VOH	−IOH=0.205mA	2.4	·	—	V
Output "Low" Voltage	VOL	IOH=1.2mA	—	·	0.4	V
Power Supply Current	ICC	Vcc=5.0V	—	0.5	2.0	mA

* Vcc=5.0V ± 5%, Ta=25°C

Block diagram



External dimensions / Display pattern



DMC164

• Display Format(16chacter ×4line) • Display Fonts(5×8dots) • Driving Method(1/16D)

ABSOLUTE MAXIMUM RATINGS

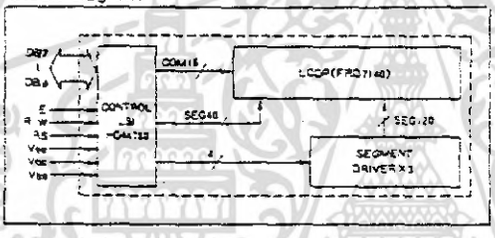
Item	Symbol	Test Condition	Standard Value min.	typ.	max.	Unit
Power Supply Voltage for Logic	Vcc−Vss	——	0	—	5.5	V
Power Supply Voltage for LCD Drive	Vcc−Vee	——	0	—	6.0	V
Input Voltage	Vi	——	Vss	—	Vcc	V
Operating Temperature	Ta	——	0	—	+50	°C
Storage Temperature	Tstg	——	−20	—	+70	°C

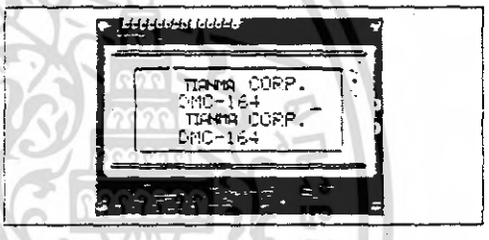
ELECTRICAL CHARACTERISTICS

Item	Symbol	Test Condition	Standard Value min.	typ.	max.	Unit
Input "High" Voltage	VIH	——	2.2	—	Vcc	V
Input "Low" Voltage	VIL	——	−0.3	—	0.6	V
Output "High" Voltage	VOH	−IOH=0.205mA	2.4	—	—	V
Output "Low" Voltage	VOL	IOH=1.2mA	—	—	0.4	V
Power Supply Current	Icc	Vcc=5.0V	—	2.4	4.0	mA

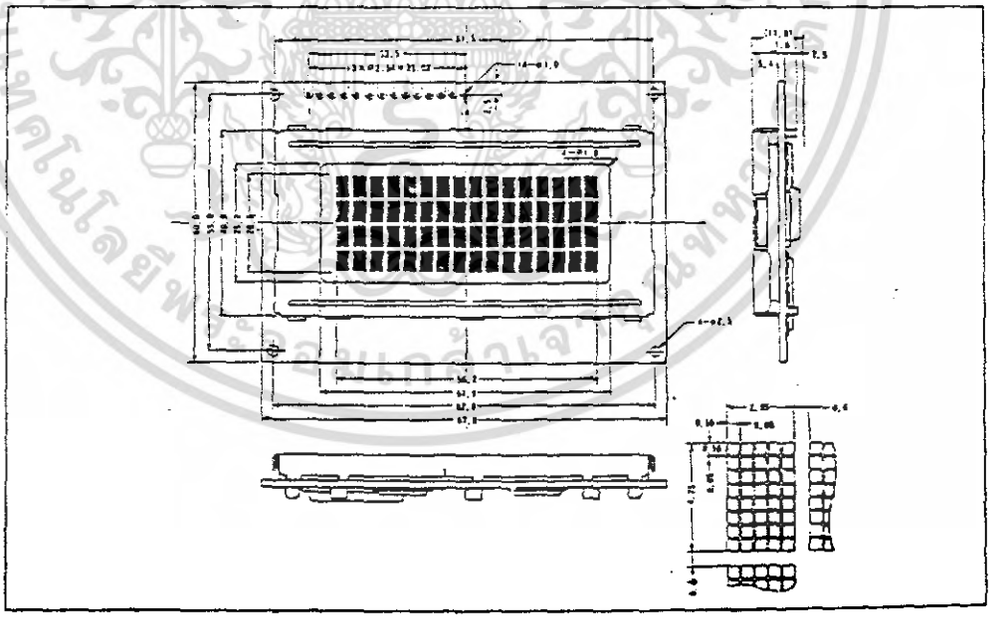
*Vcc=5.0V±5%, Ta=25°C

Block diagram





External dimensions / Display pattern



DMC202

• Display Format(20character ×2line) • Display Fonts(5×8dots) • Driving Method(1/16D)

ABSOLUTE MAXIMUM RATINGS

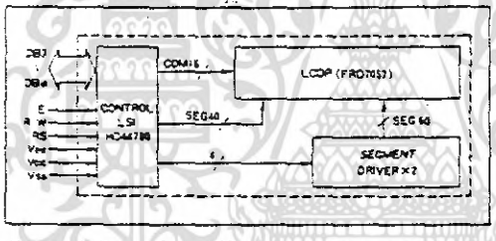
Item	Symbol	Test Condition	Standard Value min.	typ.	max.	Unit
Power Supply Voltage for Logic	Vcc−Vss	—	0	—	7	V
Power Supply Voltage for LCD Drive	Vcc−Vee	—	0	—	13.5	V
Input Voltage	Vi	—	Vss	—	Vcc	V
Operating Temperature	Ta	—	0	—	+50	°C
Storage Temperature	Tstg	—	−20	—	+70	°C

ELECTRICAL CHARACTERISTICS

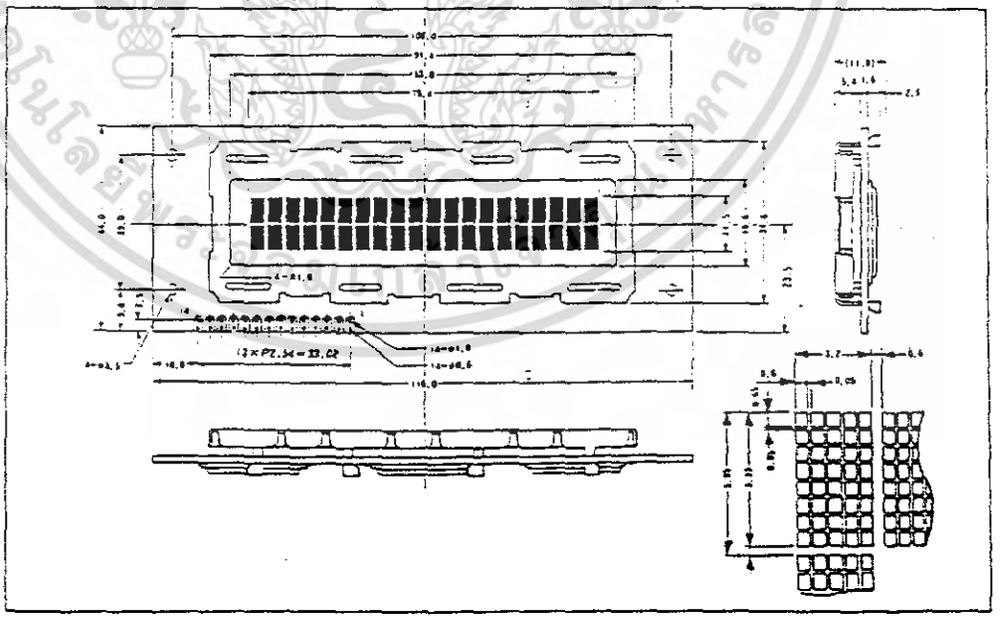
Item	Symbol	Test Condition	Standard Value min.	typ.	max.	Unit
Input "High" Voltage	VIH	—	2.2	—	Vcc	V
Input "Low" Voltage	VIL	—	−0.3	—	0.6	V
Output "High" Voltage	VOH	−IOH=0.205mA	2.4	—	—	V
Output "Low" Voltage	VOL	IOH=1.2mA	—	—	0.4	V
Power Supply Current	Icc	Vcc=5.0V	—	1.5	3.0	mA

*Vcc=5.0V±5%, Ta=25°C

Block diagram



External dimensions / Display pattern



HE [XX]	HELP	สำหรับการขอดูชุดคำสั่งทั้งหมด หรือการขอดูรายละเอียดของคำสั่ง XX ที่ต้องการ
PO 1,2	PORT	กำหนดเลข SERIAL PORT ของเครื่อง PC ที่จะใช้งาน
PA NO,OD,EV	PARITY	กำหนดลักษณะของ PARITY BIT
SE FILENAME	SEND	ส่ง FILENAME ออกทาง SERIAL PORT
SP 12,...,96	SPEED	กำหนดความเร็วของการสื่อสาร
QU	QUIT	ออกจากโปรแกรม XTALK เข้าสู่ DOS
CA FILENAME	CAPTURE	สำหรับการเก็บสิ่งที่ปรากฏบนจอภาพลง FILE
ST 1,2	STOP	กำหนดจำนวน STOP BIT
CD DIRECTORY	CHANGE	เปลี่ยนแปลง DIRECTORY ที่ใช้งาน

SXA51 8051/52 Cross Assembler

Introduction

Congratulations. You are now the owner of a fast, reliable cross-assembler for the Intel 8051/52 series of microprocessors.

Many cross-assembler manufacturers base their assemblers on a "table-driven", cross-assembler shell. They simply plug the mnemonics for a new target processor into the shell and introduce another cross-assembler. Binary Technology has been specializing in the 8051/52 since 1982 and the SXA51 was developed exclusively for this family. As a result, SXA51 is faster, more flexible, and has more specific error codes than most competitors' products.

SXA51 is an absolute assembler meaning that it does not produce relocatable modules. Relocating assemblers require the use of a linker in order to generate absolute code. SXA51 generates absolute code in only one step, thus considerably shortening the development process. We provide for modular program development by including a utility which creates one large object file from several smaller object files. The HEX utility sorts and merges Intel hex format files and reports on start address, finish address, gaps, overlaps, checksum, and entry address. It can also compare and report on the differences between two object files.

Another advantage of SXA51 is its flexibility. Labels and symbols can be any length. Underscores are valid alpha characters. Dollar signs can be embedded in symbols for readability and labels do not have to be against the left margin. All this provides the freedom to use any programming style you wish; even an indented modular style.

This manual assumes familiarity with the 8051/52 family and with their assembly language mnemonics. If you feel unfamiliar in this area, we recommend the *'EMBEDDED CONTROLLER HANDBOOK'* (Intel order number 210918-005). Chapter 7, *'MCS-51 Programmers Guide and Instruction Set'* contains most of what you will need.

Syntax Notation

The syntax notation used in this manual is as follows: items in square brackets are optional, ellipsis (...) are used to indicate that 'more of the same' is permissible, and italics mean substitute your own names. A typical source file line might be:

[*LABEL:*] OPCODE [*operand1*[, *operand2*]] [;*comment...*]

This means an optional label, followed by "OPCODE" followed (optionally) by one or two operands, followed by an optional comment.

Invoking SXA51

The cross-assembler is invoked as: SXA51 *filename*. This will assemble the source file *filename*.asm while outputting the object (machine) code in Intel hex format into *filename*.hex. Error messages and source lines containing errors will be written to the screen.

The above behaviour can be modified with the following flags: -n will suppress generation of the object file. -l will generate a file *filename*.lst which will contain a complete listing file, and a symbol table. -c will replace the symbol table with a symbol cross-reference. -d will make the assembler more verbose while it is running. These flags can be combined in any way desired, for example:

```
SXA51 -lcd foobar
```

will assemble 'foobar.asm'. A listing file with a cross reference (foobar.lst) will be generated and SXA51 will report on its progress as it runs.

In summary:

-l = Make a listing file (*filename*.lst)
-d = Be verbose
-n = Do not make an object (*filename*.hex) file
-c = Include a symbol cross-reference listing

Symbols and Labels

Symbols and labels can be any length and may contain underscores and dollar signs. All characters A-Z and 0-9 may be used but the first character must be alphabetic. SXA51 does not differentiate between upper case and lower case (i.e., LaBeL: and LABEL: are the same). Only the first 16 characters are significant. This means that SXA51 will treat SYMBOL_SEVENTY_FOUR and SYMBOL_SEVENTY_FIVE as the same. Dollar signs inside symbols or labels are ignored by the assembler. Labels must be terminated with a colon and do not have to be against the left margin.

SXA51 stores all symbols and labels as 16 bit values.

Comments

Comments begin with a semicolon. SXA51 will ignore any text to the right of a semicolon.

SXA51 Controls

The operation of the assembler can also be controlled from inside the source through the use of controls. A leading dollar sign '$' is required. Some controls cannot be changed once the assembler has begun generating code. These are the *primary* controls and are identified as such in the descriptions below. Primary controls should only be used at the beginning of the source.

$[no]date (Defaults: MS-DOS $nodate; VMS $date)

This causes the system date and time to be included on the top right of all listing pages. The format is Mon 04-Apr-1987 12:08. It is in 24 hour format.

$eject

This forces the listing to continue at the beginning of the next page. It will do so in the manner dictated by the setting of '$formfeed' below.

$[no]formfeed (Default: $noformfeed)

$formfeed causes multiple linefeeds (based on the value of '$length') to be used instead of a formfeed. $noformfeed is for older printers that don't respond to a formfeed.

$[no]list (Default: $list)

$nolist turns off the listing. This is useful if you have a large file and only want to examine the listing of a small part of it. List and nolist do not affect the symbol table or cross-reference. Therefore, a $nolist at the beginning of the source will cause only the symbol table or cross-reference to be generated.

Sif *expression*
Selse
Sendif

Conditional assembly. Causes the assembler to jump over blocks of code based on the evaluation of the expression following 'Sif'. For example:

```
TEST_ONE  EQU    0
TEST_TWO  EQU    1

START:
          do this
          do this
          do this
Sif TEST_ONE
          don't do this
          don't do this
          don't do this
Selse
          do this
          do this
          do this
Sendif
Sif NOT(TEST_TWO)
          don't do this
          don't do this
          don't do this
Sendif
          do this
          do this
          etc.
```

Slength *n* (Default: MS-DOS Slength 66; VMS Slength 0)

(Primary) Sets the length (in lines) of the listing page.

$include "*filename*"

Include the text from another file at this point as if it were written into this file. The lines from an include file are marked in the listing file with a '+'. Includes cannot be nested which means that a file 'included' in another file cannot have an include in it. The name of the file to be included must be enclosed in double quotes.

For example:

```
$include "C:\subdir\header.inc" (MS-DOS)
                    - or -
$include "DUA0:[subdir]header.inc" (VAX/VMS)
```

$[no]nullfill (Default: $nonullfill)

'DS n' (define space) causes some assemblers to generate an entry in the object file which will cause n nulls (zeros) to be loaded into program memory. Other assemblers simply jump over n locations and do not generate any program code for that space. With SXA51 you have a choice. $nullfill generates zeros while $nonullfill does not. The latter is mandatory when 'DS'ing over a space that is to be overlaid later with another object file. (See the description of the HEX utility.)

$title "*string*"

Causes *string* to be added to the top of each page of the program listing. The title must be enclosed in double quotes. It will default to the program source name if the quotes are omitted. The title can be no more than 48 characters in length.

$width n (Default: $width 132)

(Primary) Sets the width (in characters) of the program listing. This actually sets the width of the listing page header and the symbol table or cross reference. It will not truncate source lines.

$xref (Default: $noxref)

(Primary) Generates a symbol cross-reference. This is the same as using the '-c' invocation flag.

Assembler Directives

DB *expression*[, *expression*...]

DB (define byte) will place a byte or sequence of bytes into program memory. It can also insert an ascii string into sequential memory locations if the string is enclosed in single quotes. The following will place a 'null terminated' string into sequential memory locations beginning at the label 'POINTER:'

```
POINTER:    DB 'This is a string',0
```

DS *expression*

DS (define space) will set aside space in program memory based on the evaluation of *expression* (also see the description of $nullfill).

DW *expression*[, *expression*...]

DW (define word) will write a 16 bit value or sequence of values into program memory high byte first, then low byte. *Note that this is reversed from earlier Intel standards.*

ORG *expression*

Resets the program memory pointer to a new value. Subsequent code will be placed in memory beginning at *expression*.

END [*expression*]

End terminates the program and (optionally) indicates the code entry address.

SYMBOL_NAME EQU *expression*
SYMBOL_NAME BIT *expression*

Assigns the value of *expression* to 'symbol_name'. The 'BIT' operator is retained for compatability with Intel assemblers. SXA51 makes no distinction between the two types.

SYMBOL_NAME SET *expression*

SET is the same as EQU but does not complain if the same symbol is assigned a new value later on.

Expressions

Expressions can be made up of numbers and/or symbols. Numbers are assumed to be decimal unless indicated otherwise. Hex numbers must begin with a digit and end with an "H". 30H and 0F4H are valid hex numbers; F3H is not. Octal numbers end with an "O" or "Q" and binary numbers end with a "B". The special symbol, "$", refers to the program location "at the beginning of the present line". For example, the line, "JMP $", will create an endless loop.

The order of precedence for operators is:

Highest: unary + unary - HIGH LOW
** (exponentiation)
* / % (%=MOD)
+ - MOD SHR SHL AND OR XOR
Lowest: EQ NE GE LE GT LT

The expression parser will evaluate left-to-right if the operators are of equal precedence. The best rule of thumb is to use parentheses when in doubt.

Bit Operations

SXA51 adheres to the Intel convention of bit notation. That is, *n.m* where *n* is the bit-addressable byte address and *m* is the bit inside that byte. For example, 22H.3 refers to the fourth bit in bit-addressable byte 3. (Bit-addressable memory begins at location 20H). This would translate to bit address 13H. Notations such as ACC.7 are also permitted.

There is a small price to pay for the added speed of working with an absolute assembler. Specifically, you will have to keep your code module origins and routine addresses organized. This is not as bad as it might initially sound, and the difference between a 30 second assemble and a 15 minute assemble and link is worth it.

Suppose that you have finished one module and are now working on a second that contains a routine which will be called by the first. The location of this routine will probably move relative to the second module's origin as the module evolves. The best way to deal with this is to put an LJMP instruction at the beginning of the second module causing a jump into the actual routine. The first module will call the routine at the location of the jump. In this way, the first module will not require changes or reassembly as work proceeds on the second module. In fact, the first module could be burned into a PROM to speed up the downloading process during development and debugging. Several routines in one module can be done in this manner by using a sequence of LJMPs at the beginning of the module. Once the code is stable, the extra LJMPs can be removed without danger of changing the operation of the program.

Another modular development technique used with SXA51 is to create an $include file which contains the EQUates for origins and EQUates for the locations of "public" routines. Public routines are any which must be called from a different module. This file can then be included in each of the modules so there will be no differences of opinion regarding the location of any given routine.

One example of this method is demonstrated by the use of the Binary Technology M/DP monitor-debugger. The M/DP contains several useful routines such as CHROUT (send a character out the serial port). The location given in the M/DP documentation for CHROUT is 0008H. (This location actually contains an LJMP to the real CHROUT routine inside the M/DP.) The next step is to create a file ('mdp.inc', for example) which contains the line 'CHROUT EQU 0008H'. Then, in your program, insert the line '$include "mdp.inc"'. Thereafter, a 'CALL CHROUT' will cause the program to find its way to the routine in the MD P ROM.

SXA51 Error Messages

Byte:

B	Value out-of-bounds. Must be between 0 and 255

Control:

C1	A primary control must be used before any code generation
C2	Unrecognized control

Expression:

E1	Unbalanced parentheses
E2	Cannot translate variable name
E3	Divide by zero
E4	Incorrect bit address
E5	General expression error
E6	Radix error
E7	Bit.n: n must be between 0 and 7

Include

I	Include nesting too deep

Multiply defined.

M1	Label multiply defined
M2	EQU or SET multiply defined

Offset error:

O1	ACALL or AJMP must be inside 4096 byte 'page'
O2	SJMP must be within +127 or -128

Syntax:

S	General.
S1	First char. on line must be alphabetic
S2	Label error: probably a reserved word
S3	First char. (after label) must be alphabetic
S4	Reserved word in EQU or SET
S5	Incorrect use of opcode
S6	Cannot evaluate ORG expression
S7	Incorrect register or data type
S8	Cannot DEC DPTR
S9	Expected 'A'
S10	Expected comma

S11 Expected 'AB'

S12 Expected '@A+DPTR'

S13 Expected '@A+PC' or '@A+DPTR'

S14 Expected 'A, @DPTR', 'A, @R0' or 'A, @R1'

S15 Expected '@R0' or '@R1'

S16 SETB operand must be Bit or C

S17 String must be enclosed in quotes or is too long

Unrecognized:

U General

U0 Opcode

Pass 1 errors:

"Incorrect forward reference"

This error is caused if a symbol is referenced that has not yet been defined yet and the symbol is one which can affect the location of the code being generated. An error is flagged during the first pass of the assembler and it is aborted. Such uses would be: "DS *undefined_symbol*" or "JMP *undefined_label*" (This is the generic version of the jump instruction which will be converted by the assembler into either a one or a two byte jump instruction.) A "CALL *undefined_label*" will fail for the same reason as the JMP.

"No END statement."

This is a non-fatal error. It is good practice to include an END statement.

Built-in 8051/52 Register Names

Bit addresses

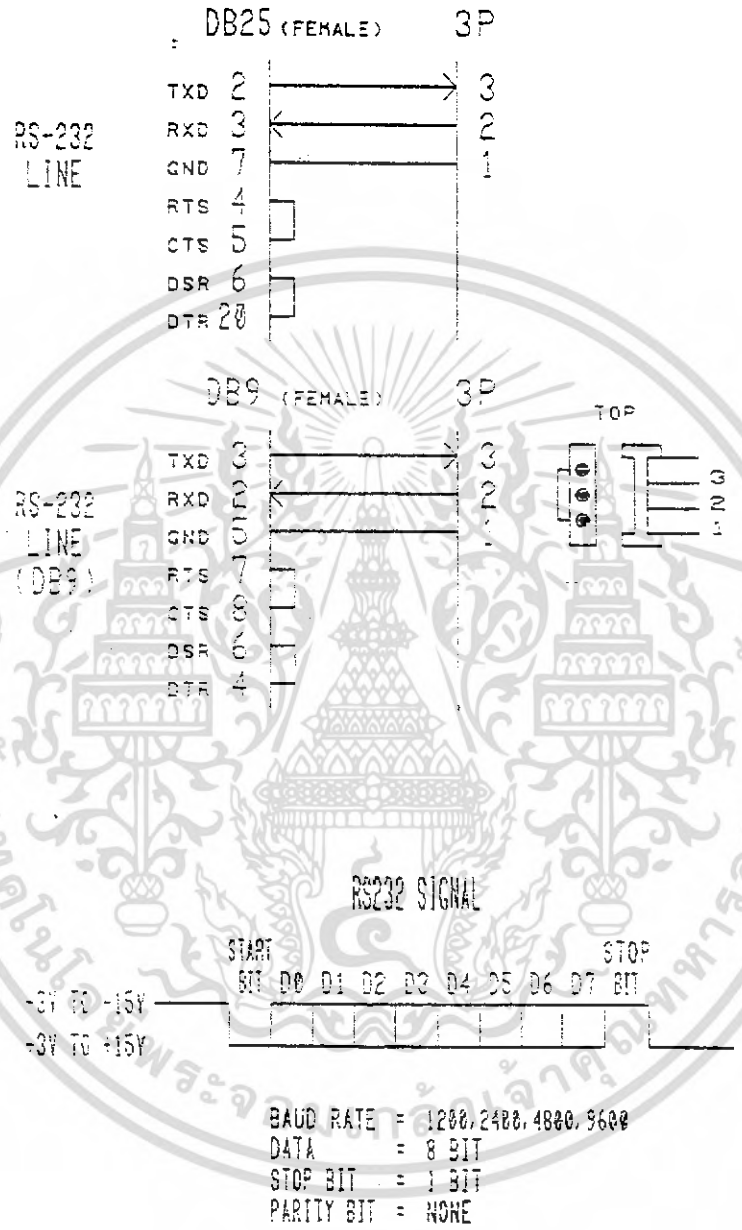
TF1	8F	SM0	9F	EA	AF
TR1	8E	SM1	9E		
TF0	8D	SM2	9D	ET2	AD
TR0	8C	REN	9C	ES	AC
IE1	8B	TB8	9B	ET1	AB
IT1	8A	RB8	9A	EX1	AA
IE0	89	TI	99	ET0	A9
IT0	88	RI	98	EX0	A8
RD	B7			CY	D7
WR	B6			AC	D6
T1	B5	PT2	BD	F0	D5
T0	B4	PS	BC	RS1	D4
INT1	B3	PT1	BB	RS0	D3
INT0	B2	PX1	BA	OV	D2
TXD	B1	PT0	B9		
RXD	B0	PX0	B8	P	D0

Direct addresses

ACC	E0	P3	B0	T2CON	C8
B	F0	PCON	87	TH0	8C
DPH	83	PSW	D0	TL0	8A
DPL	82	RCAP2H	CB	TH1	8D
IE	A8	RCAP2L	CA	TL1	8B
IP	B8	SBUF	99	TH2	CD
P0	80	SCON	98	TL2	CC
P1	90	SP	81	TMOD	89
P2	A0	TCON	88		

Predefined Origins

RESET	0000
EXTI0	0003
TIMER0	000B
EXTI1	0013
TIMER1	001B
SINT	0023



MCS®-51 INSTRUCTION SET

Table 10. 8051 Instruction Set Summary

Interrupt Response Time: Refer to Hardware Description Chapter.

Instructions that Affect Flag Settings(1)

Instruction	Flag C	OV	AC	Instruction	Flag C	OV	AC
ADD	X	X	X	CLR C	0		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C,bit	X		
MUL	0	X		ANL C,/bit	X		
DIV	0	X		ORL C,bit	X		
DA	X			ORL C,/bit	X		
RRC	X			MOV C,bit	X		
RLC	X			CJNE	X		
SETB C	1						

(1) Note that operations on SFR byte address 208 or bit addresses 209-215 (i.e., the PSW or bits in the PSW) will also affect flag settings.

Note on instruction set and addressing modes:

Rn — Register R7–R0 of the currently selected Register Bank.

direct — 8-bit internal data location's address. This could be an Internal Data RAM location (0–127) or a SFR (i.e., I/O port, control register, status register, etc. (128–255)).

@Ri — 8-bit internal data RAM location (0–255) addressed indirectly through register R1 or R0.

#data — 8-bit constant included in instruction.

#data 16 — 16-bit constant included in instruction.

addr 16 — 16-bit destination address. Used by LCALL & LJMP. A branch can be anywhere within the 64K-byte Program Memory address space.

addr 11 — 11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.

rel — Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is −128 to +127 bytes relative to first byte of the following instruction.

bit — Direct Addressed bit in Internal Data RAM or Special Function Register.

Mnemonic		Description	Byte	Oscillator Period
ARITHMETIC OPERATIONS				
ADD	A,Rn	Add register to Accumulator	1	12
ADD	A,direct	Add direct byte to Accumulator	2	12
ADD	A,@Ri	Add indirect RAM to Accumulator	1	12
ADD	A,#data	Add immediate data to Accumulator	2	12
ADDC	A,Rn	Add register to Accumulator with Carry	1	12
ADDC	A,direct	Add direct byte to Accumulator with Carry	2	12
ADDC	A,@Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDC	A,#data	Add immediate data to Acc with Carry	2	12
SUBB	A,Rn	Subtract Register from Acc with borrow	1	12
SUBB	A,direct	Subtract direct byte from Acc with borrow	2	12
SUBB	A,@Ri	Subtract indirect RAM from ACC with borrow	1	12
SUBB	A,#data	Subtract immediate data from Acc with borrow	2	12
INC	A	Increment Accumulator	1	12
INC	Rn	Increment register	1	12
INC	direct	Increment direct byte	2	12
INC	@Ri	Increment direct RAM	1	12
DEC	A	Decrement Accumulator	1	12
DEC	Rn	Decrement Register	1	12
DEC	direct	Decrement direct byte	2	12
DEC	@Ri	Decrement indirect RAM	1	12

Table 10. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
ARITHMETIC OPERATIONS (Continued)				
INC	DPTR	Increment Data Pointer	1	24
MUL	AB	Multiply A & B	1	48
DIV	AB	Divide A by B	1	48
DA	A	Decimal Adjust Accumulator	1	12
LOGICAL OPERATIONS				
ANL	A,Rn	AND Register to Accumulator	1	12
ANL	A,direct	AND direct byte to Accumulator	2	12
ANL	A,@Ri	AND indirect RAM to Accumulator	1	12
ANL	A,#data	AND immediate data to Accumulator	2	12
ANL	direct,A	AND Accumulator to direct byte	2	12
ANL	direct,#data	AND immediate data to direct byte	3	24
ORL	A,Rn	OR register to Accumulator	1	12
ORL	A,direct	OR direct byte to Accumulator	2	12
ORL	A,@Ri	OR indirect RAM to Accumulator	1	12
ORL	A,#data	OR immediate data to Accumulator	2	12
ORL	direct,A	OR Accumulator to direct byte	2	12
ORL	direct,#data	OR immediate data to direct byte	3	24
XRL	A,Rn	Exclusive-OR register to Accumulator	1	12
XRL	A,direct	Exclusive-OR direct byte to Accumulator	2	12
XRL	A,@Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL	A,#data	Exclusive-OR immediate data to Accumulator	2	12
XRL	direct,A	Exclusive-OR Accumulator to direct byte	2	12
XRL	direct,#data	Exclusive-OR immediate data to direct byte	3	24
CLR	A	Clear Accumulator	1	12
CPL	A	Complement Accumulator	1	12

Mnemonic		Description	Byte	Oscillator Period
LOGICAL OPERATIONS (Continued)				
RL	A	Rotate Accumulator Left	1	12
RLC	A	Rotate Accumulator Left through the Carry	1	12
RR	A	Rotate Accumulator Right	1	12
RRC	A	Rotate Accumulator Right through the Carry	1	12
SWAP	A	Swap nibbles within the Accumulator	1	12
DATA TRANSFER				
MOV	A,Rn	Move register to Accumulator	1	12
MOV	A,direct	Move direct byte to Accumulator	2	12
MOV	A,@Ri	Move indirect RAM to Accumulator	1	12
MOV	A,#data	Move immediate data to Accumulator	2	12
MOV	Rn,A	Move Accumulator to register	1	12
MOV	Rn,direct	Move direct byte to register	2	24
MOV	Rn,#data	Move immediate data to register	2	12
MOV	direct,A	Move Accumulator to direct byte	2	12
MOV	direct,Rn	Move register to direct byte	2	24
MOV	direct,direct	Move direct byte to direct	3	24
MOV	direct,@Ri	Move indirect RAM to direct byte	2	24
MOV	direct,#data	Move immediate data to direct byte	3	24
MOV	@Ri,A	Move Accumulator to indirect RAM	1	12

Table 10. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
DATA TRANSFER (Continued)				
MOV	@Ri,direct	Move direct byte to indirect RAM	2	24
MOV	@Ri,#data	Move immediate data to indirect RAM	2	12
MOV	DPTR,#data16	Load Data Pointer with a 16-bit constant	3	24
MOVC	A,@A+DPTR	Move Code byte relative to DPTR to Acc	1	24
MOVC	A,@A+PC	Move Code byte relative to PC to Acc	1	24
MOVX	A,@Ri	Move External RAM (8-bit addr) to Acc	1	24
MOVX	A,@DPTR	Move External RAM (16-bit addr) to Acc	1	24
MOVX	@Ri,A	Move Acc to External RAM (8-bit addr)	1	24
MOVX	@DPTR,A	Move Acc to External RAM (16-bit addr)	1	24
PUSH	direct	Push direct byte onto stack	2	24
POP	direct	Pop direct byte from stack	2	24
XCH	A,Rn	Exchange register with Accumulator	1	12
XCH	A,direct	Exchange direct byte with Accumulator	2	12
XCH	A,@Ri	Exchange indirect RAM with Accumulator	1	12
XCHD	A,@Ri	Exchange low-order Digit indirect RAM with Acc	1	12

Mnemonic		Description	Byte	Oscillator Period
BOOLEAN VARIABLE MANIPULATION				
CLR	C	Clear Carry	1	12
CLR	bit	Clear direct bit	2	12
SETB	C	Set Carry	1	12
SETB	bit	Set direct bit	2	12
CPL	C	Complement Carry	1	12
CPL	bit	Complement direct bit	2	12
ANL	C,bit	AND direct bit to CARRY	2	24
ANL	C,/bit	AND complement of direct bit to Carry	2	24
ORL	C,bit	OR direct bit to Carry	2	24
ORL	C,/bit	OR complement of direct bit to Carry	2	24
MOV	C,bit	Move direct bit to Carry	2	12
MOV	bit,C	Move Carry to direct bit	2	24
JC	rel	Jump if Carry is set	2	24
JNC	rel	Jump if Carry not set	2	24
JB	bit,rel	Jump if direct Bit is set	3	24
JNB	bit,rel	Jump if direct Bit is Not set	3	24
JBC	bit,rel	Jump if direct Bit is set & clear bit	3	24
PROGRAM BRANCHING				
ACALL	addr11	Absolute Subroutine Call	2	24
LCALL	addr16	Long Subroutine Call	3	24
RET		Return from Subroutine	1	24
RETI		Return from interrupt	1	24
AJMP	addr11	Absolute Jump	2	24
LJMP	addr16	Long Jump	3	24
SJMP	rel	Short Jump (relative addr)	2	24

Table 10. 8051 Instruction Set Summary (Continued)

Mnemonic		Description	Byte	Oscillator Period
PROGRAM BRANCHING (Continued)				
JMP	@A + DPTR	Jump indirect relative to the DPTR	1	24
JZ	rel	Jump if Accumulator is Zero	2	24
JNZ	rel	Jump if Accumulator is Not Zero	2	24
CJNE	A,direct,rel	Compare direct byte to Acc and Jump if Not Equal	3	24
CJNE	A,#data,rel	Compare immediate to Acc and Jump if Not Equal	3	24
CJNE	Rn,#data,rel	Compare immediate to register and Jump if Not Equal	3	24
CJNE	@Ri,#data,rel	Compare immediate to indirect and Jump if Not Equal	3	24
DJNZ	Rn,rel	Decrement register and Jump if Not Zero	2	24
DJNZ	direct,rel	Decrement direct byte and Jump if Not Zero	3	24
NOP		No Operation	1	12

All mnemonics copyrighted © Intel Corporation 1980

Table 11. Instruction Opcodes in Hexadecimal Order

Hex Code	Number of Bytes	Mnemonic	Operands
00	1	NOP	
01	2	AJMP	code addr
02	3	LJMP	code addr
03	1	RR	A
04	1	INC	A
05	2	INC	data addr
06	1	INC	@R0
07	1	INC	@R1
08	1	INC	R0
09	1	INC	R1
0A	1	INC	R2
0B	1	INC	R3
0C	1	INC	R4
0D	1	INC	R5
0E	1	INC	R6
0F	1	INC	R7
10	3	JBC	bit addr, code addr
11	2	ACALL	code addr
12	3	LCALL	code addr
13	1	RRC	A
14	1	DEC	A
15	2	DEC	data addr
16	1	DEC	@R0
17	1	DEC	@R1
18	1	DEC	R0
19	1	DEC	R1
1A	1	DEC	R2
1B	1	DEC	R3
1C	1	DEC	R4
1D	1	DEC	R5
1E	1	DEC	R6
1F	1	DEC	R7
20	3	JB	bit addr, code addr
21	2	AJMP	code addr
22	1	RET	
23	1	RL	A
24	2	ADD	A,#data
25	2	ADD	A,data addr
26	1	ADD	A,@R0
27	1	ADD	A,@R1
28	1	ADD	A,R0
29	1	ADD	A,R1
2A	1	ADD	A,R2
2B	1	ADD	A,R3
2C	1	ADD	A,R4
2D	1	ADD	A,R5
2E	1	ADD	A,R6
2F	1	ADD	A,R7
30	3	JNB	bit addr, code addr
31	2	ACALL	code addr
32	1	RETI	

Hex Code	Number of Bytes	Mnemonic	Operands
33	1	RLC	A
34	2	ADDC	A,#data
35	2	ADDC	A,data addr
36	1	ADDC	A,@R0
37	1	ADDC	A,@R1
38	1	ADDC	A,R0
39	1	ADDC	A,R1
3A	1	ADDC	A,R2
3B	1	ADDC	A,R3
3C	1	ADDC	A,R4
3D	1	ADDC	A,R5
3E	1	ADDC	A,R6
3F	1	ADDC	A,R7
40	2	JC	code addr
41	2	AJMP	code addr
42	2	ORL	data addr,A
43	3	ORL	data addr, #data
44	2	ORL	A, #data
45	2	ORL	A,data addr
46	1	ORL	A,@R0
47	1	ORL	A,@R1
48	1	ORL	A,R0
49	1	ORL	A,R1
4A	1	ORL	A,R2
4B	1	ORL	A,R3
4C	1	ORL	A,R4
4D	1	ORL	A,R5
4E	1	ORL	A,R6
4F	1	ORL	A,R7
50	2	JNC	code addr
51	2	ACALL	code addr
52	2	ANL	data addr,A
53	3	ANL	data addr, #data
54	2	ANL	A, #data
55	2	ANL	A,data addr
56	1	ANL	A,@R0
57	1	ANL	A,@R1
58	1	ANL	A,R0
59	1	ANL	A,R1
5A	1	ANL	A,R2
5B	1	ANL	A,R3
5C	1	ANL	A,R4
5D	1	ANL	A,R5
5E	1	ANL	A,R6
5F	1	ANL	A,R7
60	2	JZ	code addr
61	2	AJMP	code addr
62	2	XRL	data addr,A
63	3	XRL	data addr, #data
64	2	XRL	A, #data
65	2	XRL	A,data addr

Table 11. Instruction Opcodes in Hexadecimal Order (Continued)

Hex Code	Number of Bytes	Mnemonic	Operands
66	1	XRL	A,@R0
67	1	XRL	A,@R1
68	1	XRL	A,R0
69	1	XRL	A,R1
6A	1	XRL	A,R2
6B	1	XRL	A,R3
6C	1	XRL	A,R4
6D	1	XRL	A,R5
6E	1	XRL	A,R6
6F	1	XRL	A,R7
70	2	JNZ	code addr
71	2	ACALL	code addr
72	2	ORL	C,bit addr
73	1	JMP	@A + DPTR
74	2	MOV	A,# data
75	3	MOV	data addr,# data
76	2	MOV	@R0,# data
77	2	MOV	@R1,# data
78	2	MOV	R0,# data
79	2	MOV	R1,# data
7A	2	MOV	R2,# data
7B	2	MOV	R3,# data
7C	2	MOV	R4,# data
7D	2	MOV	R5,# data
7E	2	MOV	R6,# data
7F	2	MOV	R7,# data
80	2	SJMP	code addr
81	2	AJMP	code addr
82	2	ANL	C,bit addr
83	1	MOVC	A,@A + PC
84	1	DIV	AB
85	3	MOV	data addr, data addr
86	2	MOV	data addr,@R0
87	2	MOV	data addr,@R1
88	2	MOV	data addr,R0
89	2	MOV	data addr,R1
8A	2	MOV	data addr,R2
8B	2	MOV	data addr,R3
8C	2	MOV	data addr,R4
8D	2	MOV	data addr,R5
8E	2	MOV	data addr,R6
8F	2	MOV	data addr,R7
90	3	MOV	DPTR, # data
91	2	ACALL	code addr
92	2	MOV	bit addr,C
93	1	MOVC	A,@A + DPTR
94	2	SUBB	A, # data
95	2	SUBB	A,data addr
96	1	SUBB	A,@R0
97	1	SUBB	A,@R1
98	1	SUBB	A,R0

Hex Code	Number of Bytes	Mnemonic	Operands
99	1	SUBB	A,R1
9A	1	SUBB	A,R2
9B	1	SUBB	A,R3
9C	1	SUBB	A,R4
9D	1	SUBB	A,R5
9E	1	SUBB	A,R6
9F	1	SUBB	A,R7
A0	2	ORL	C,/bit addr
A1	2	AJMP	code addr
A2	2	MOV	C,bit addr
A3	1	INC	DPTR
A4	1	MUL	AB
A5		reserved	
A6	2	MOV	@R0,data addr
A7	2	MOV	@R1,data addr
A8	2	MOV	R0,data addr
A9	2	MOV	R1,data addr
AA	2	MOV	R2,data addr
AB	2	MOV	R3,data addr
AC	2	MOV	R4,data addr
AD	2	MOV	R5,data addr
AE	2	MOV	R6,data addr
AF	2	MOV	R7,data addr
B0	2	ANL	C,/bit addr
B1	2	ACALL	code addr
B2	2	CPL	bit addr
B3	1	CPL	C
B4	3	CJNE	A, # data,code addr
B5	3	CJNE	A,data addr,code addr
B6	3	CJNE	@R0, # data,code addr
B7	3	CJNE	@R1, # data,code addr
B8	3	CJNE	R0, # data,code addr
B9	3	CJNE	R1, # data,code addr
BA	3	CJNE	R2, # data,code addr
BB	3	CJNE	R3, # data,code addr
BC	3	CJNE	R4, # data,code addr
BD	3	CJNE	R5, # data,code addr
BE	3	CJNE	R6, # data,code addr
BF	3	CJNE	R7, # data,code addr
C0	2	PUSH	data addr
C1	2	AJMP	code addr
C2	2	CLR	bit addr
C3	1	CLR	C
C4	1	SWAP	A
C5	2	XCH	A,data addr
C6	1	XCH	A,@R0
C7	1	XCH	A,@R1
C8	1	XCH	A,R0
C9	1	XCH	A,R1
CA	1	XCH	A,R2
CB	1	XCH	A,R3

Table 11. Instruction Opcodes in Hexadecimal Order (Continued)

Hex Code	Number of Bytes	Mnemonic	Operands
CC	1	XCH	A,R4
CD	1	XCH	A,R5
CE	1	XCH	A,R6
CF	1	XCH	A,R7
D0	2	POP	data addr
D1	2	ACALL	code addr
D2	2	SETB	bit addr
D3	1	SETB	C
D4	1	DA	A
D5	3	DJNZ	data addr,code addr
D6	1	XCHD	A,@R0
D7	1	XCHD	A,@R1
D8	2	DJNZ	R0,code addr
D9	2	DJNZ	R1,code addr
DA	2	DJNZ	R2,code addr
DB	2	DJNZ	R3,code addr
DC	2	DJNZ	R4,code addr
DD	2	DJNZ	R5,code addr
DE	2	DJNZ	R6,code addr
DF	2	DJNZ	R7,code addr
E0	1	MOVX	A,@DPTR
E1	2	AJMP	code addr
E2	1	MOVX	A,@R0
E3	1	MOVX	A,@R1
E4	1	CLR	A
E5	2	MOV	A,data addr

Hex Code	Number of Bytes	Mnemonic	Operands
E6	1	MOV	A,@R0
E7	1	MOV	A,@R1
E8	1	MOV	A,R0
E9	1	MOV	A,R1
EA	1	MOV	A,R2
EB	1	MOV	A,R3
EC	1	MOV	A,R4
ED	1	MOV	A,R5
EE	1	MOV	A,R6
EF	1	MOV	A,R7
F0	1	MOVX	@DPTR,A
F1	2	ACALL	code addr
F2	1	MOVX	@R0,A
F3	1	MOVX	@R1,A
F4	1	CPL	A
F5	2	MOV	data addr,A
F6	1	MOV	@R0,A
F7	1	MOV	@R1,A
F8	1	MOV	R0,A
F9	1	MOV	R1,A
FA	1	MOV	R2,A
FB	1	MOV	R3,A
FC	1	MOV	R4,A
FD	1	MOV	R5,A
FE	1	MOV	R6,A
FF	1	MOV	R7,A

สรุปชื่อ SFR และ BIT ADDRESS ที่ใช้กับ REM31

SFR

80 P0	81 SP	82 DPL	83 DPH
87 PCON	88 TCON	89 TMOD	8A TL0
8B TL1	8C TH0	8D TH1	90 P1
98 SCON	99 SBUF	A0 P2	A8 IE
B0 P3	B8 IP	C8 T2CON	CA RCAP2L
CB RCAP2H	CC TL2	CD TH2	D0 PSW
E0 ACC	F0 B		

BIT ADDRESS

P0.0	IE1	P1.6	P2.1	ES	PX0	TR2	ACC.1	B.4
P0.1	TR0	P1.7	P2.2	ET2	PT0	CT2	ACC.2	B.5
P0.2	TF0	RI	P2.3	EA	PX1	CPRL2	ACC.3	B.6
P0.3	TR1	TI	P2.4	RXD	PT1	P	ACC.4	B.7
P0.4	TF1	RB8	P2.5	TXD	PS	OV	ACC.5	
P0.5	P1.0	TB8	P2.6	INT0	PT2	RS0	ACC.6	
P0.6	P1.1	REN	P2.7	INT1	TF2	RS1	ACC.7	
P0.7	P1.2	SM2	EX0	T0	EXF2	F0	B.0	
IT0	P1.3	SM1	ET0	T1	RCLK	AC	B.1	
IE0	P1.4	SM0	EX1	WR	TLCK	CY	B.2	
IT1	P1.5	P2.0	ET1	RD	EXEN2	ACC.0	B.3	