ภาควิชาครุศาสตร์วิศวกรรม

คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

.....

ปริญญานิพนธ์ โปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS - 51

MCS-51 SIMULATOR

ชื่อนักศึกษา

1. นายธานินทร์	โพธิ์อำไพ	รหัสประจำตัว	40031413
2. นายปัทมราช	ดอกไม้	รหัสประจำตัว	40031422
3. นายสุภกิจ	นุตยะสกุล	รหัสประจำตัว	40031435

หลักสูตร ครุศาสตร์อุตสาหกรรมบัณฑิต สาขาวิชา อิเล็กทรอนิกส์และคอมพิวเตอร์

อาจารย์ผู้ควบคุมปริญญานิพนธ์

อาจารย์ไพบูลย์ พวงวงศ์ตระกูล

คณะกรรมการสอบปริญญานิพนธ์		ลายมือชื่อ
1. อาจารย์ไพบูลย์	พวงวงศ์ตระกูล	
2. อาจารย์พีระวุฒิ	สุวรรณจันทร์	
3. อาจารย์โกศล	ตราชู	
4. อาจารย์ปิยะ	ศุภวราสุวัฒน์	
5. อาจารย์สุระชัย	พิมพ์สาลี	

วันเดือนปีที่สอบ วันที่ 11 พฤศจิกายน 2541 เวลา 12.00 น. ถึง 13.00 น.

สถานที่สอบ ห้อง ค. 310 คณะครุศาสตร์อุตสาหกรรม

ภาควิชารับรองแล้ว

.....

(ธีระพล เทพหัสดิน ณ อยุธยา)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

วันที่ 12 เดือน ก.พ. พ.ศ. 42

ปริญญานิพนธ์

โปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS - 51

MCS - 51 SIMULATOR

นายธานินทร์ โพธิ์อำไพ

นายปัทมราช ดอกไม้

นายสุภกิจ นุตยะสกุล

ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรครุศาสตร์อุตสาหกรรมบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์

ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

ปริญญานิพนธ์

เรื่อง โปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS - 51

MCS - 51 SIMULATOR

ผู้จัดทำ

1. นายธานินทร์ โพธิ์อำไพ
2. นายปัทมราช ดอกไม้
3. นายสุภกิจ นุตยะสกุล

อาจารย์ที่ปรึกษา

ลงนาม _____

(อาจารย์ไพบูลย์ พวงวงศ์ตระกูล)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

ลงนาม _____

(ผศ.ดร.ธีระพล เทพหัสดิน ณ อยุธยา)

ปริญญานิพนธ์

เรื่อง โปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS - 51

MCS - 51 SIMULATOR

วัตถุประสงค์

1. ศึกษาการเขียนโปรแกรมบนวินโดว์ในรูปแบบวิชวลเพื่อประยุกต์ใช้งาน
2. ศึกษารูปแบบของคำสั่ง และการใช้งานของไมโครคอนโทรลเลอร์ MCS-51
3. สร้างโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS-51

ประโยชน์ที่คาดว่าจะได้รับ

1. สามารถนำโปรแกรมไปใช้งานบนระบบปฏิบัติการวินโดว์ได้
2. สามารถช่วยให้การเรียนการสอนในวิชาไมโครคอนโทรลเลอร์ ได้มีการทดลองและฝึกหัดการเขียนโปรแกรมได้สะดวกขึ้น
3. ช่วยลดค่าใช้จ่ายในการจัดซื้อเครื่องซิงเกิลบอร์ดเพื่อใช้ในการทดลอง
4. สามารถที่จะพัฒนาโปรแกรมของ MCS-51 ได้สะดวก รวดเร็ว และมีประสิทธิภาพสูงขึ้น

โปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS - 51

นายธานินทร์ โพธิ์อำไพ
นายปัทมราช ดอกไม้
นายสุภกิจ นุตยะสกุล

อาจารย์ที่ปรึกษา

อาจารย์ไพบูลย์ พวงวงศ์ตระกูล

ปีการศึกษา 2541

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้นำเสนอโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS-51 โดยมีคุณสมบัติเฉพาะของโปรแกรม ดังนี้ สามารถจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS-51 ได้, สามารถแสดงค่าของรีจิสเตอร์และแฟล็กต่างๆ ได้, จำลองการทำงานของอุปกรณ์ต่อร่วมเพื่อการติดต่อได้, มีการทำงานภายใต้ระบบปฏิบัติการวินโดว์, แสดงค่าข้อมูลของหน่วยความจำที่ตำแหน่งต่างๆ ได้

MCS - 51 SIMULATOR

MR.THANIN PHOAMPHAI

MR.PATTAMARACH DOKMAI

MR.SUPAKIT NOOTYASKOOL

ADVISOR

MR.PAIBOON PONGWONGTRAGULL

1998

ABSTRACT

This thesis presents the MCS-51 simulator.The properties of a softwere thus able simulation MCS-51 micro controller, able to show register and flag, simulation of board interface, use by windows operating system, show data of memory at any position.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ สำเร็จลุล่วงได้ด้วยดีเนื่องจากความอนุเคราะห์ของอาจารย์ที่ปรึกษาปริญญานิพนธ์ และอาจารย์ประจำภาควิชาครุศาสตร์วิศวกรรมทุกท่าน ที่ได้กรุณาให้คำปรึกษา, ข้อเสนอแนะต่างๆ ที่เป็นประโยชน์ ขอขอบคุณ คุณพ่อ คุณแม่ ที่ให้การอุปการะ ขอขอบคุณเพื่อนๆ สาขาอิเล็กทรอนิกส์และคอมพิวเตอร์ ห้อง 2 ทุกคน ที่ช่วยเป็นกำลังใจ และสุดท้ายขอขอบคุณสมาชิกทุกท่านในกลุ่ม ที่ช่วยให้ปริญญานิพนธ์นี้เสร็จสมบูรณ์

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญตาราง

สารบัญภาพ

สารบัญภาพ (ต่อ)

รูปภาพ **หน้า**

สารบัญภาพ (ต่อ)

สารบัญภาพ (ต่อ)

สารบัญภาพ (ต่อ)

สารบัญภาพ (ต่อ)

สารบัญภาพ (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปริญญานิพนธ์

ในกระบวนการทำงานหลายๆ ด้าน เราจะสังเกตเห็นว่าได้มีการนำไมโครคอนโทรลเลอร์ MCS-51 มาใช้งานทางด้านการควบคุมหรืองานอื่นๆ ตามแต่ผู้ใช้ต้องการอยู่หลายๆ งานด้วยกัน เช่น การควบคุมการเปิดปิดสวิตช์ไฟ การควบคุมการแสดงผลของตัวอักษรวิ่ง เป็นต้น ดังนั้นจึงแสดงให้เราเห็นว่าไมโครคอนโทรลเลอร์มีประโยชน์และความสำคัญอย่างมาก แต่กระบวนการเรียนรู้นอกจากการศึกษาในหนังสือหรือตำราเรียนแล้ว สิ่งหนึ่งที่จะช่วยให้นักศึกษามีความเข้าใจมากยิ่งขึ้นก็คือการได้ฝึกหัด การเขียนโปรแกรมและทำการทดลอง แต่ปัญหาที่เกิดขึ้นคือนักศึกษาไม่มีเครื่องซิงเกิ้ลบอร์ดมาใช้ในการทดลองหรือศึกษาให้เพียงพอต่อความต้องการ หรือถ้าจะซื้อเองก็ทำได้ยากเนื่องจากเครื่องซิงเกิลบอร์ดมีราคาแพง ดังนั้นจึงมีความคิดที่จะเขียนโปรแกรมเพื่อจำลองการทำงานของ MCS-51 และมีการจำลองส่วนของอุปกรณ์ทางด้านฮาร์ดแวร์ ไว้ด้วยเพื่อให้เห็นถึงผลของการทำงานตามคำสั่งของโปรแกรม MCS-51 ซึ่งทำให้ผู้ใช้มีความสะดวกในการใช้งานได้ดีขึ้น

1.2 ขีดความสามารถของโครงงาน

โครงงานนี้มีขีดความสามารถดังต่อไปนี้

1. สามารถจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS-51 ได้
2. สามารถแสดงค่ารีจิสเตอร์ต่างๆ ได้ เช่น ACC, B, PSW เป็นต้น
3. สามารถแสดงผลการทำงานของโปรแกรมบนจอคอมพิวเตอร์ได้
4. สามารถบันทึก (Save) และเปิด (Open File) ที่เขียนขึ้นแล้วได้
5. สามารถอ่านคู่มือ (Help) เพื่อช่วยในการใช้โปรแกรมได้
6. โปรแกรมทำงานภายใต้ระบบปฏิบัติการวินโดว์ 95/98
7. มีชุดอินเตอร์เฟสจำลอง 8255

1.3 เนื้อหาโดยสังเขป

เนื้อหาภายในปริญญานิพนธ์ฉบับนี้แบ่งออกเป็นบทต่างๆ เพื่อความสะดวกต่อการศึกษาและทำความเข้าใจ ในแต่ละบทจะประกอบด้วยเนื้อหาที่สำคัญดังนี้

บทที่ 2 ทฤษฎีและหลักการ ประกอบด้วยเนื้อหาในทางทฤษฎีที่เกี่ยวข้อง ซึ่งทำให้ผู้อ่านได้มีความรู้ความเข้าใจเพื่อเป็นพื้นฐานเสียก่อนจึงเป็นประโยชน์ต่อการทำความเข้าใจกับการทำงานของโปรแกรมที่ใช้งานจริงต่อไป

บทที่ 3 การออกแบบการสร้างและการทำงาน ประกอบด้วยเนื้อหาในเรื่องของการออกแบบโปรแกรม การสร้างหน้าจอต่างๆ การสร้างชุดอินเตอร์เฟสจำลองการทำงาน การสร้างไฟล์ช่วยเหลือ และอื่นๆ เพื่อให้ผู้อ่านได้เข้าใจวิธีการในการออกแบบและการสร้างเพื่อทำการพัฒนาหรือปรับปรุงโปรแกรมให้มีประสิทธิภาพเพิ่มขึ้นในโอกาสต่อไป

บทที่ 4 การทดลองและผลการทดลอง กล่าวถึงขั้นตอนในการทำการทดลอง และผลการทดสอบโปรแกรม ในชุดคำสั่งของโปรแกรมในส่วนต่างๆ

บทที่ 5 บทสรุปปัญหาแนวทางแก้ไขและพัฒนา กล่าวถึงปัญหาที่เกิดขึ้นจากการทำงานของผู้ศึกษา และแนวทางในการแก้ไขที่ได้รับจากอาจารย์ที่ปรึกษาปริญญานิพนธ์ ในด้านต่างๆ

ในภาคผนวกแสดงรายละเอียดของโปรแกรม และส่วนต่างๆ ที่ใช้จัดทำโครงงานดังนี้

ภาคผนวก ก คู่มือการใช้งานโปรแกรม

ภาคผนวก ข ใบงานการทดลอง

บทที่ 2
ทฤษฎีและหลักการ

2.1 กล่าวนำ

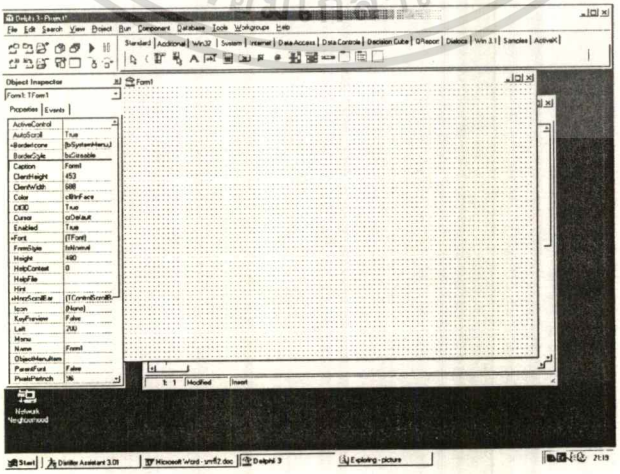
การเขียนโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS-51 นั้นเราใช้โปรแกรมเดลไฟในการเขียนโปรแกรมซึ่งเป็นโปรแกรมที่ทำงานภายใต้ระบบปฏิบัติการวินโดว์ ดังนั้นเรายังต้องศึกษาการทำงานของคำสั่งต่างๆ ของไมโครคอนโทรลเลอร์เบอร์ 8051, การเปิด Hexfile, โครงสร้างของหน่วยความจำจำลอง, การติดต่อกับชุดอินเตอร์เฟสจำลอง ซึ่งสามารถกล่าวโดยละเอียดได้ดังต่อไปนี้

2.2 การเขียนโปรแกรมด้วยภาษาเดลไฟ

2.2.1 หน้าจอของเดลไฟ

หน้าจอของเดลไฟแบ่งออกได้เป็น 4 ส่วนใหญ่ๆ ดังรูปที่ 2.1คือ

1. หน้าต่างหลัก
2. หน้าต่างฟอร์ม (Form)
3. หน้าต่างยูนิต (Unit)
4. หน้าต่างออบเจ็กต์อินสเปคเตอร์ (Object Inspector)



รูปที่ 2.1 หน้าจอของเดลไฟโดยรวม

1. หน้าต่างหลัก

จากรูปที่ 2.2 หน้าต่างหลักประกอบด้วย

1. ไตเติลบาร์ (Title bar) จะแสดงรายชื่อของชิ้นงาน (Application) แต่ละชิ้นงาน
2. เมนูบาร์ (Menu bar) แสดงรายการตั้งแต่ File จนถึง Help
3. กล่องคอมโปเนนต์ (Component) จะเป็นที่เก็บคอมโปเนนต์ต่างๆ ที่จะไปวางไว้บนฟอร์ม
4. สปีดบาร์ (Speed Bar) คือ ส่วนของปุ่มที่ใช้ในการเรียกการทำงานดังรูปที่ 2.3 ประกอบไปด้วย

    4.1 Open Project คือ ปุ่มที่ใช้เปิดโปรเจกต์ใหม่ที่สร้างไว้แล้ว

    4.2 Save File คือ ปุ่มที่ใช้เปิดไฟล์ใหม่ที่สร้างไว้แล้ว

    4.3 Save All คือ ปุ่มที่ใช้เซฟโปรเจกต์ของงานที่กำลังทำอยู่

    4.4 Save File คือ ปุ่มที่ใช้เซฟเฉพาะบางไฟล์

    4.5 Add File To Project คือ ปุ่มที่ใช้ในการนำยูนิต (ไฟล์นามสกุล PAS) ของโปรเจกต์อื่นเข้ามาทำงานร่วมกับโปรเจกต์ที่เรากำลังทำงานอยู่

    4.6 Remove File To Project คือ ปุ่มที่ใช้ในการนำยูนิตของโปรเจกต์ที่กำลังใช้งานอยู่ออกไป

    4.7 Select Unit From List คือ ปุ่มที่ใช้เลือกยูนิตที่มีอยู่แล้วใน โปรเจกต์ขึ้นมาใช้งาน

    4.8 Toggle From/Unit คือ ปุ่มที่ใช้ในการสลับกันทำงานระหว่างหน้าต่างฟอร์มกับหน้าต่างยูนิต

    4.9 Select Form From List คือ ปุ่มที่ใช้เลือกฟอร์มที่มีอยู่แล้วในโปรเจกต์ขึ้นมาใช้งาน

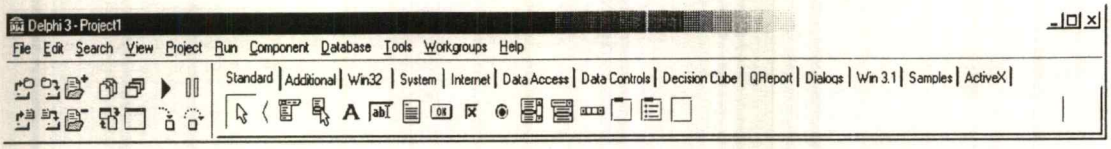
    4.10 New Form คือ ปุ่มที่ใช้เรียกฟอร์มใหม่ขึ้นมาใช้งานร่วมกับโปรเจกต์ที่กำลังใช้งานอยู่

    4.11 Run คือ ปุ่มที่ทำให้ตัว Project compile เริ่มทำงาน

    4.12 Trace into คือปุ่มที่ใช้แสดงผลการทำงานของโปรเจกต์ทีละขั้นตอน

    4.13 Pause คือ ปุ่มที่ทำให้การทำงานของโปรเจกต์หยุดชั่วคราว

    4.14 Step Over คือ ปุ่มที่ใช้ในการข้ามการแสดงผลบางช่วงไป

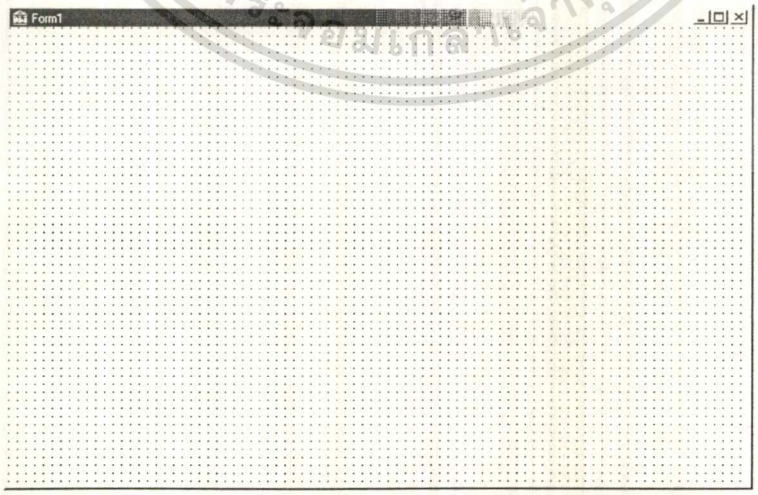


รูปที่ 2.2 หน้าต่างหลัก

รูปที่ 2.3 สปีดบาร์ (Speed bar)

2. หน้าต่างฟอร์ม

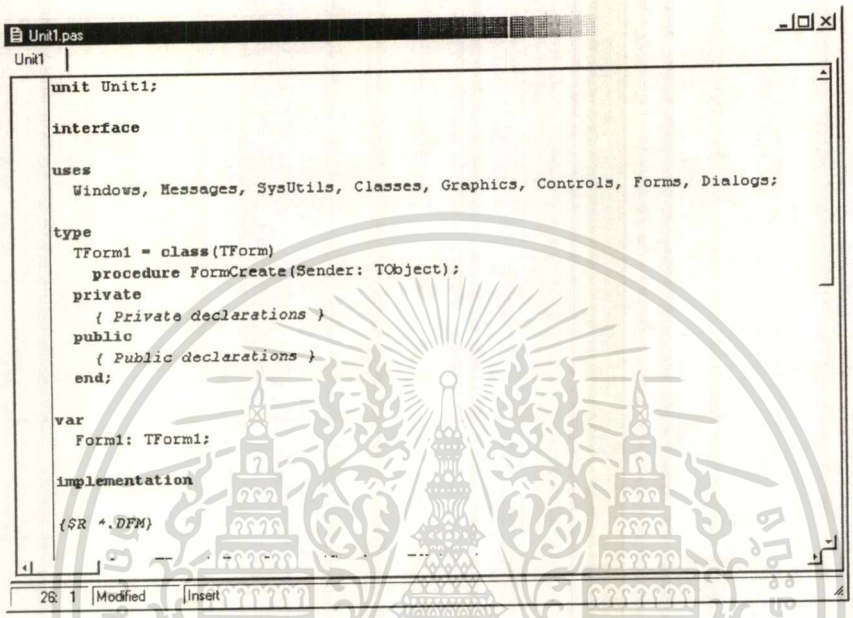
หน้าต่างฟอร์มแสดงดังรูปที่ 2.4 เป็นหน้าต่างที่มีไว้สำหรับแสดงผลในการทำงานของตัวโปรเจกต์ เวลาใช้ให้นำคอมโปเนนต์ที่ต้องการใช้ลงไปวาง แล้วกำหนด Properties กับ Events ในออบเจ็กต์อินสเปคเตอร์



รูปที่ 2.4 หน้าต่างฟอร์ม (Form)

3. หน้าต่างยูนิต

หน้าต่างยูนิตแสดงดังรูปที่ 2.5 เป็นหน้าต่างที่มีไว้สำหรับใส่เงื่อนไขในการทำงานของโปรแกรม โดยผ่านทาง Events ของออบเจ็กต์อินสเปคเตอร์ หรือจะใส่เงื่อนไขโดยตรงก็ได้

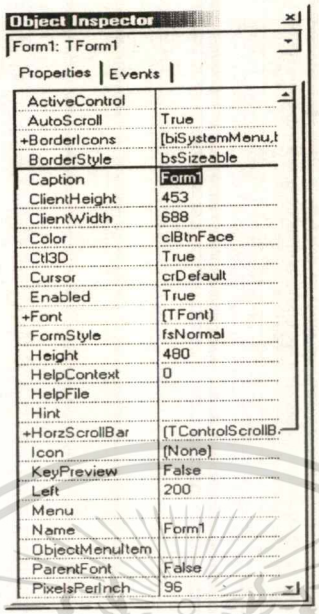


รูปที่ 2.5 หน้าต่างยูนิต (Unit)

4. หน้าต่างออบเจ็กต์อินสเปคเตอร์

หน้าต่างออบเจ็กต์อินสเปคเตอร์ดังแสดงในรูปที่ 2.6จะประกอบด้วย

1. Properties จะเป็นตัวกำหนดการทำงานของคอมโปเนนต์ต่างๆ
2. Events จะเป็นตัวผ่านจากคอมโปเนนต์ไปกำหนดเงื่อนไขต่างๆ ในยูนิต เช่น
   - OnClick จะทำงานก็ต่อเมื่อเราเลือกด้วยเมาส์หรือด้วยคีย์บอร์ด
   - OnMouseDown จะทำงานก็ต่อเมื่อเราคลิกเมาส์ลง
   - OnMouseUp จะทำงานก็ต่อเมื่อเราคลิกเมาส์ขึ้น
   - OnKeyDown จะทำงานโดยการกดคีย์ลงจะไปทำงานที่ key code ที่กำหนดไว้ในเงื่อนไขของ Events นั้นๆ ในโปรแกรม



รูปที่ 2.6 หน้าต่าง Object Inspector

2.2.2 เริ่มต้นใช้งาน

การเริ่มต้นใช้งานจริงนี้จะเริ่มอธิบายตั้งแต่การเปิดใช้งานโปรแกรมเดลไฟ,การเปิด โปรเจกต์ใหม่, การวางคอมโปเนนต์, การกำหนดพร็อพเพอร์ตี้, การกำหนดอีเวนต์, การกำหนดเงื่อนไข, การเซฟไฟล์, การคอมไพล์, การเปิดโปรเจกต์เก่าขึ้นมาแก้ไข และการนำโปรเจกต์ไปใช้งานจริงหลังจากคอมไพล์แล้ว

1. การเปิดโปรแกรมเดลไฟขึ้นมาใช้งาน

เลือกไปที่ Start แล้วเลือกไปที่ Programs / Borland Delphi 3 ก็จะแสดงหน้าจอของโปรแกรมเดลไฟ 3 ออกมา

2. การเปิดโปรเจกต์ใหม่และการใช้งาน

1. ไปที่เมนูบาร์ เลือก File / New Application
2. กำหนดชื่อ Project ใหม่ให้เหมาะสมกับการใช้งาน

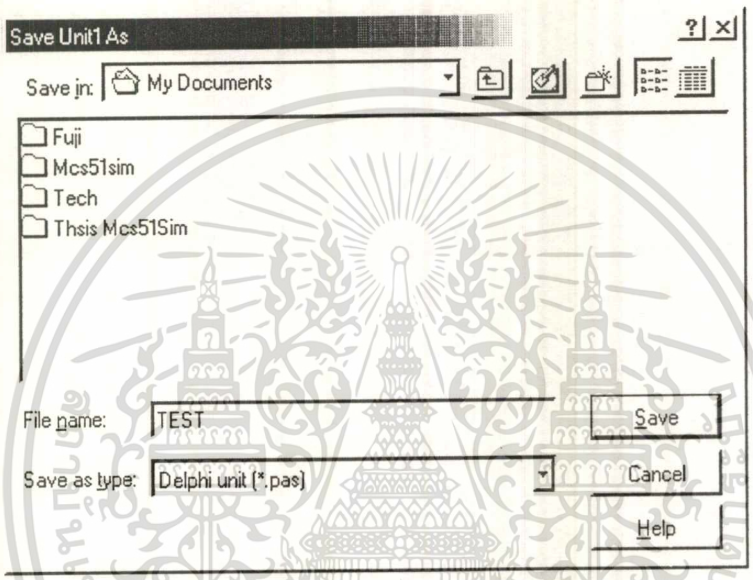
2.1 ไปที่เมนูบาร์ เลือก File / Save Project As หรือจะใช้เมาส์ไปคลิกที่ปุ่ม Save All บนสปีดบาร์

2.2 จะมีหน้าต่าง Save Unit As แสดงขึ้นมาดังรูปที่ 2.7 เป็นหน้าต่างที่มีไว้ให้เซฟไฟล์ยูนิต ให้พิมพ์ชื่อไฟล์ยูนิตที่เราต้องการลงในช่อง File name

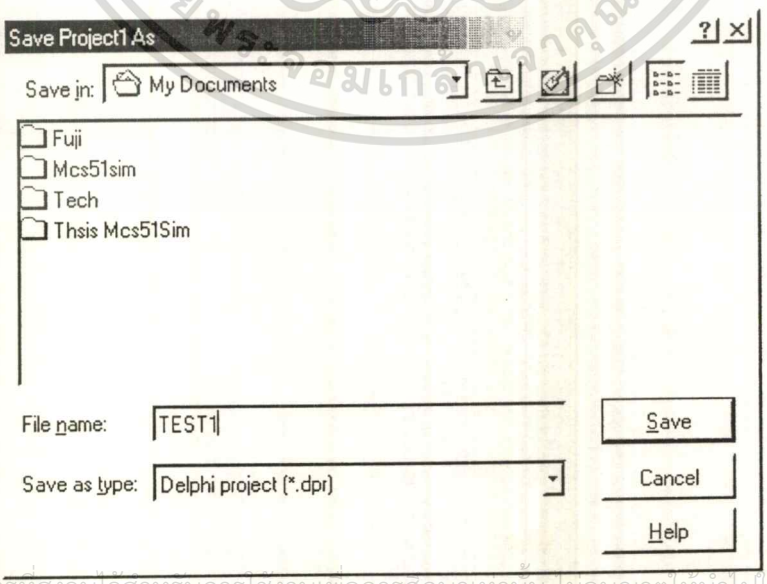
2.3 คลิกไปที่ Save

2.4 จะมีหน้าต่าง Save Project1 As แสดงขึ้นมา เป็นหน้าต่างที่มีไว้ให้เซฟไฟล์โปรเจกต์ ให้พิมพ์ชื่อไฟล์โปรเจกต์ที่เราต้องการลงในช่อง File name

2.5 คลิกไปที่ Save



รูปที่ 2.7 หน้าต่าง Save Unit As



รูปที่ 2.8 หน้าต่าง Save Project As

2.2.3 ตัวอย่างการเริ่มต้นใช้งาน

ตัวอย่างการกำหนดชื่อโปรเจกต์ใหม่

1. PROGRAM PNEW {TEST1.dpr}

2. Unit PNEW1 {TEST.pas}

อธิบายตัวอย่างการกำหนดชื่อโปรเจกต์ใหม่

บรรทัดที่ 1 คือการตั้งชื่อโปรเจกต์ใหม่ โดยให้ชื่อว่า TEST1 แล้วเซฟเป็นสกุล dpr

บรรทัดที่ 2 คือการตั้งชื่อยูนิตใหม่ โดยให้ชื่อว่า TEST แล้วเซฟเป็นสกุล pas ทั้งสองบรรทัดข้างบนนี้จะทำได้ โดยการเลือกไปที่สปีดบาร์แล้วเลือกที่ Save All หรือเลือกที่ เมนู File / Save All จะแสดงหน้าต่าง Save Unit1 As ดังในรูปที่ 2.7 ให้พิมพ์ TEST ลงในช่อง File name แล้วกด Save หลังจากนั้นจะมีหน้าต่าง Save Project1 As แสดงขึ้นมา ดังในรูปที่ 2.8 ให้พิมพ์ TEST1 ลงในช่อง File name แล้วกด Save

3. จัดการกำหนดคอมโปเนนต์ที่ต้องการ

ตัวอย่างการวางคอมโปเนนต์

Components

1. Form == Form1
   - Caption = Pnew
2. Standard :: Button == Button1
   - Caption = OK
3. Standard :: Label == Label1
   - Caption = Hi

อธิบายตัวอย่างการวางคอมโปเนนต์

ข้อ 1 เป็นการให้ Caption ของ Form1 เป็น Pnew โดยผ่านทาง Properties

ข้อ 2 เป็นการนำคอมโปเนนต์ Button จากกล่องคอมโปเนนต์ Standard มาวางบน Form1 จะได้ Button1 แล้วให้ค่า Properties ของ Button1 ให้ Caption เป็น OK

ข้อ 3 เป็นการนำคอมโปเนนต์ Label จากกล่องคอมโปเนนต์ Standard มาวางบน Form1 จะได้ Label1 แล้วให้ค่า Properties ของ Label1 ให้ Caption เป็น Hi

4. เมื่อกำหนดคอมโปเนนต์ตามตัวอย่างได้แล้วก็จะได้ฟอร์มออกมาจัดการใส่เงื่อนไขโดยผ่าน Events

ตัวอย่างการใส่เงื่อนไขโดยผ่าน Events

Events OnClick ของ Button1

```
1: procedure Tform1.Button1Click(Sender: TObject)
2: begin
3: Label1.Caption := Hello;
4: end;
```

อธิบายตัวอย่างการใส่เงื่อนไขโดยผ่าน Events

การที่เราจะเข้าถึง Events ตามในตัวอย่างนี้ได้ เราต้องเข้าไปกำหนด Events (Object) ที่ชื่อว่า Button1 โดยทำดังต่อไปนี้

1. ใช้เมาส์เลือกไปที่วัตถุ Button1
2. ใช้เมาส์เลือกไปที่ Events ในออบเจ็กต์อินสเปคเตอร์
3. ใช้เมาส์ดับเบิลคลิกไปที่ OnClick
4. และหน้าต่างยูนิตก็จะแสดงขึ้นมา
5. แล้วใส่เงื่อนไขตามบรรทัดที่ 3 ในตัวอย่างลงไป (Label.Caption := Hello;)

5. ทำการคอมไพล์ (COMPILE) ให้อยู่ในสกุล exe

5.1 ขั้นตอนในการคอมไพล์มี 2 วิธี คือ

5.1.1 เลือกไปที่เมนู Project / Compile หรือกด Ctrl+F9

5.1.2 เลือกไปที่เมนู Run / Run หรือกด F9 แต่วิธีที่ง่ายที่สุด คือใช้เมาส์เลือกไปที่ปุ่ม Run

5.2 เมื่อเข้าใจขั้นตอนการคอมไพล์แล้วก็ทดลองนำโปรแกรม PNEW มาทำดู จะได้ผล

5.3 เมื่อต้องการจะปิดผลการรัน ก็ให้ใช้เมาส์เลือกไปที่ปุ่ม X ตรงมุมขวาบนของหน้าต่างที่กำลังรันอยู่

6. ทดลองปิดโปรเจกต์ PNEW แล้วเปิดดูใหม่โดย

6.1 เลือกไปที่เมนู File / Open หรือเลือกไปที่ปุ่ม Open Project

6.2 ใช้เมาส์เลือกไปที่ชื่อโปรแกรมที่ต้องการจะเปิด

6.3 แล้วเลือกปุ่ม Open โปรแกรมก็จะเปิดออกมาพร้อมที่จะรับการแก้ไข

7. การนำโปรแกรมที่คอมไพล์แล้วมาทำเป็น SHORTCUT เพื่อที่จะใช้งานได้เลยมีขั้นตอนดังต่อไปนี้

7.1 ใช้เมาส์คลิกขวา แล้วเลือกไปที่ New / Shortcut ก็จะแสดงหน้าต่าง Create Shortcut ขึ้นมา

7.2 ให้เลือกไปที่ปุ่ม Browse ไฟล์ที่เราต้องการที่จะทำเป็น Shortcut

7.3 เมื่อเลือกได้แล้วให้ไปเลือกที่ปุ่ม Open จะกลับมาที่หน้าต่าง Create Shortcut

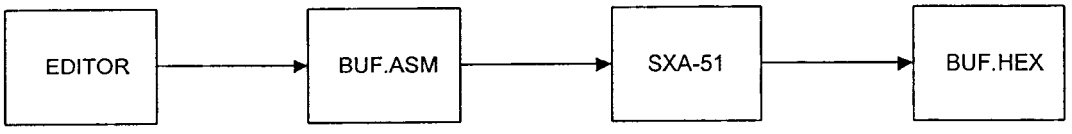
7.4 เลือกไปที่ Next และ Finish ก็จะได้ Shortcut

2.3 การอ่านชุดคำสั่ง MCS - 51 เพื่อแปลงเป็นไฟล์ .HEX

การอ่านชุดคำสั่ง MCS-51 เพื่อแปลงเป็น Hex ไฟล์ในโปรแกรมมีกระบวนการ ดังรูปที่ 2.9 จะทำการบันทึกข้อมูลที่อยู่ในอิดิเตอร์ไว้ที่ไฟล์ BUF.ASM จากนั้นจึงเรียกโปรแกรม SXA51 เพื่อทำการคอมไพล์โปรแกรมซึ่งจะได้ไฟล์ BUF.HEX ลักษณะการเรียกใช้งานโปรแกรม SXA51 ในโปรแกรมเดลไฟจะใช้คำสั่ง CREATE PROCESS ซึ่งมีรูปแบบในการใช้งานคำสั่ง ดังนี้

CREATEPROCESS (mli, o, c:\sxa51_LBUF.HEX, NE, NIC, flass, o, mil, nil, startImfo, proeImfo)

Create process เป็นการสลับการทำงานของโปรแกรมให้ไปเรียกใช้การทำงานภายนอก



รูปที่ 2.9 กระบวนการในการคอมไพล์โปรแกรม

2.4 โครงสร้าง Intel Hexfile

อินเทลเฮกซ์ฟอร์แมต (Intel Hexformat) มีรูปแบบคล้ายกับแบบแอสกี้เฮกซ์ฟอร์แมต แต่ในรูปแบบของอินเทลนี้จะมีการอ้างแอดเดรส และตรวจสอบความผิดพลาดของข้อมูลนี้ด้วย ทั้งนี้เพื่อให้การบันทึกข้อมูลมีความถูกต้องน่าเชื่อถือมากขึ้น รูปแบบของข้อมูลแบบนี้มีลักษณะดังตารางที่ 2.1

ตารางที่ 2.1 รูปแบบของข้อมูลในระบบ Intel Hexformat

ชื่อสัญลักษณ์	จำนวนตัวอักษร	รายละเอียด
Record Mark	1	ในแต่ละบรรทัดของข้อมูลที่ต้องการบันทึก ต้องเริ่มด้วยตัวโคลอน (:)
Record length	2	จำนวนข้อมูลที่ต้องการบันทึก
Address Field	2	แสดงแอดเดรสที่เก็บข้อมูลในไฟล์แรก ปกติมีค่า 00
Record Type	2	ชนิดของข้อมูลที่ทำการบันทึก โดยถ้าเป็นข้อมูล 00 หมายถึง ข้อมูลของโปรแกรม 01 หมายถึง จบแฟ้มข้อมูล 02 หมายถึง แอดเดรสเพิ่มเติม 03 หมายถึง แอดเดรสเริ่มต้น
Datafield	เปลี่ยนแปลง	จะขึ้นอยู่กับรหัสชนิดของข้อมูลที่ทำการบันทึก 00 หมายถึง จำนวนข้อมูลที่ต้องการโปรแกรม 01 หมายถึง ไม่ใช้งาน (ว่าง) 02 หมายถึง เซกเมนต์สำหรับแอดเดรสที่มากกว่า 64 กิโลไบต์ ข้อมูลจะถูกเริ่มต้นเก็บที่เซกเมนต์ x 10h บวกกับค่าของพื้นที่แอดเดรส ตัวอย่าง แอดเดรสเพิ่มเติมมีค่า F800h พื้นที่ของแอดเดรสเท่ากับ 1000h ดัง

ตารางที่ 2.1 (ต่อ) รูปแบบของข้อมูลในระบบ Intel Hexformat

ชื่อสัญลักษณ์	จำนวนตัวอักษร	รายละเอียด
Checksum	2	นั้นข้อมูลจะไปเริ่มต้นบันทึกที่ตำแหน่ง F900h การคำนวณค่า Checksum มีดังนี้ รวมค่าของข้อมูลทั้งหมดที่ทำการบันทึก จากนั้นทำทูคอมพลีเมนต์ ค่า Check Sum คือไบต์ค่าของข้อมูลสุดท้ายหลัง ทำทูคอมพลีเมนต์

ตัวอย่างที่ 2.1 เพิ่มข้อมูลแบบอินเทลเฮกซ์ฟอร์เมต

: 03801300028223 C 3

: 20820000D28AE589440A54/FF589DZAF757F00757E00

257D00D28ED2AAE57FB403F1302AA9D

: 16822000020000C28E858D7D858B7E758D00758B00D

8E057F32C1

: 0000000 \ FF

ตัวอย่างที่ 2.2 การคำนวณ Checksum สำหรับบรรทัดที่ 1

(1) รวมค่าของข้อมูลทั้ง 7 ค่าได้ 10 7h หรือ 100000111b

(2) ทำทูคอมพลีเมนต์ได้ 011111000+1=011111001b หรือ F9h

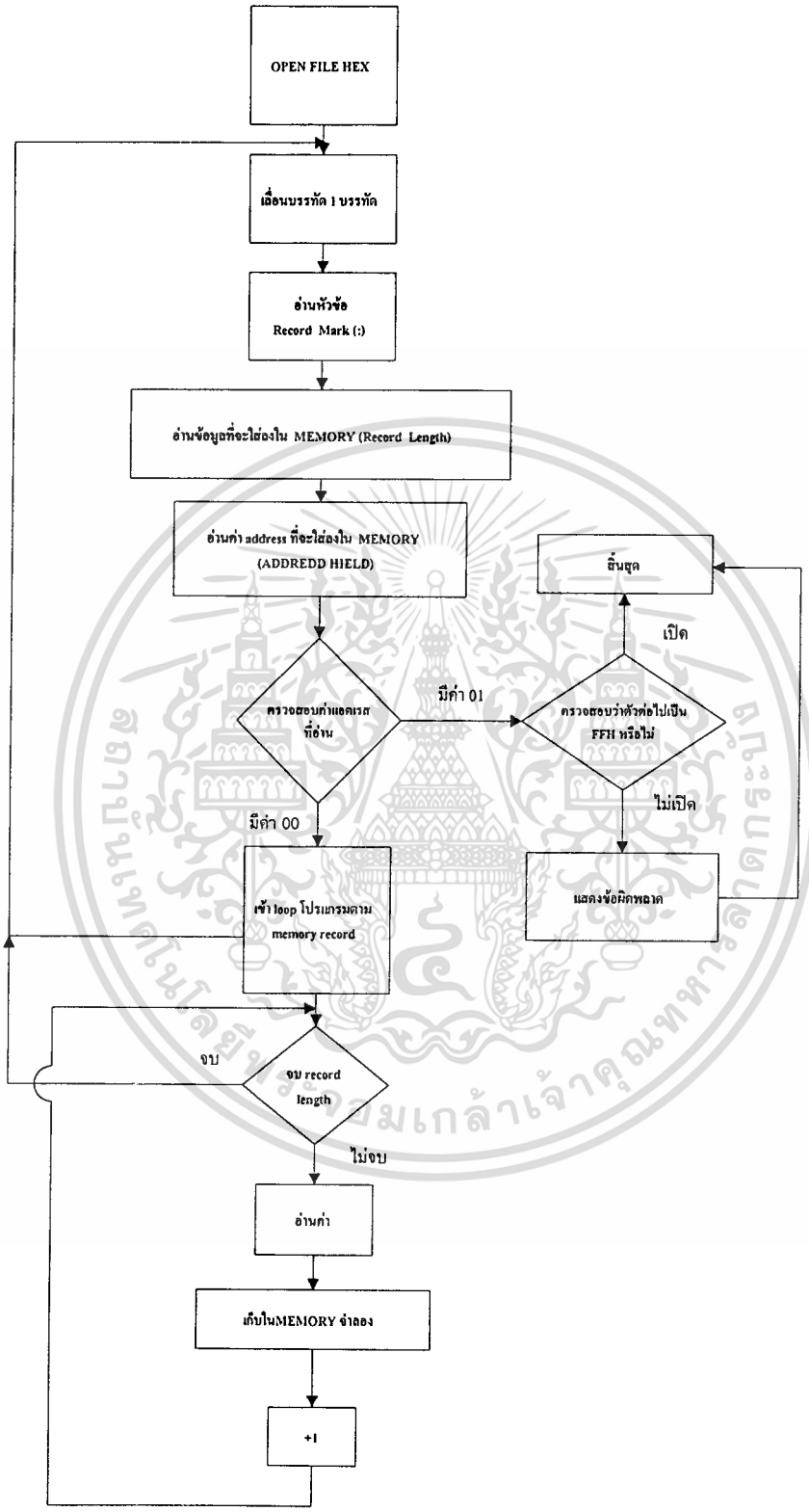
(3) ค่า Checksum เท่ากับ F9

ค่า Checksum นี้มีประโยชน์ในการตรวจสอบข้อมูลที่ทำการบันทึกว่าถูกต้องหรือไม่

2.5 การเปิด Hexfile แล้วเปิดหน่วยความจำ

จากรูปแบบการหา Format Hex File ของอินเทลในการเปิดข้อมูลใช้ไฟล์และเก็บในหน่วยความจำนั้นทำได้ดังรูปที่ 2.10

จะสังเกตได้ว่า data file จะไม่มีค่า 02H เพราะ ในความหมายว่า Program ของ MCS-51 จะใช้ MEMORY ไม่ถึง 64 Kbytes ดังนั้นจึงไม่ต้องใช้



รูปที่ 2.10 การทำงานการเปิด HEX File

2.6 โครงสร้างของหน่วยความจำจำลอง

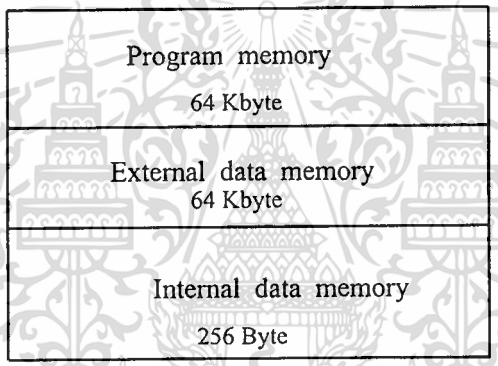
ในการจำลองการทำงานของ MCS-51 นั้นต้องค้นหาหน่วยความจำของ MCS-51 ด้วย โดยหน่วยความจำที่ใช้ใน MCS-51 ประกอบด้วย 3 แบบ คือ

1.) Program Memory ใช้เก็บโปรแกรมสูงสุด 64 KB

2.) External Data Memory ใช้เก็บข้อมูลสูงสุด 64 KB

3.) Internal Data Memory ใช้เก็บข้อมูลภายในโปรแกรม ซึ่งเป็นรีจิสเตอร์ต่าง ๆ 256 Byte

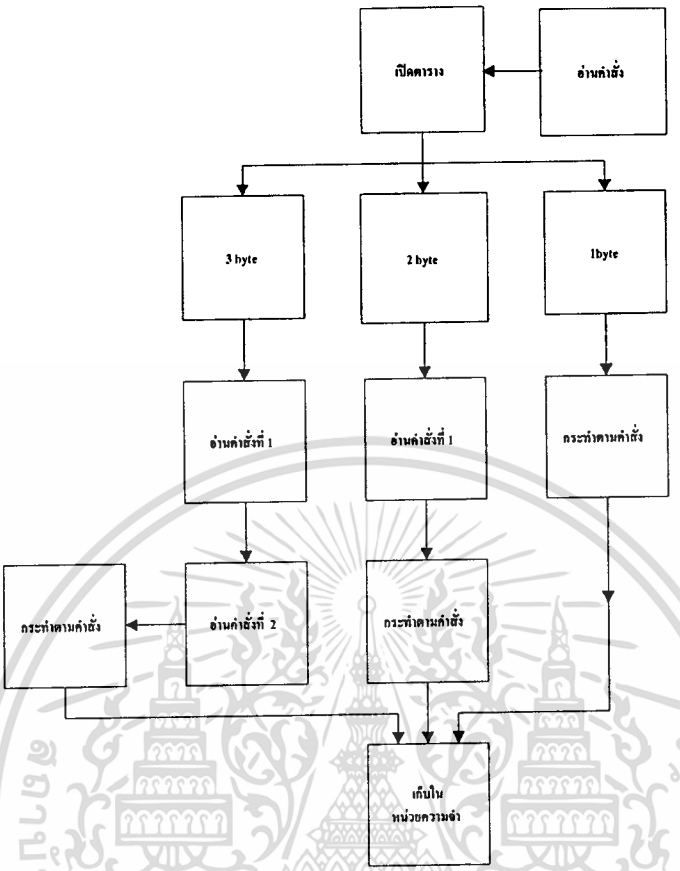
จาก Memory ทั้ง 3 แบบแสดงโครงสร้างดังรูปที่ 2.11 ซึ่งจะมีขนาด 8 Bit และในส่วน Internal Data Memory จะต้องสามารถกระทำระดับบิตได้



รูปที่ 2.11 โครงสร้างของหน่วยความจำจำลอง

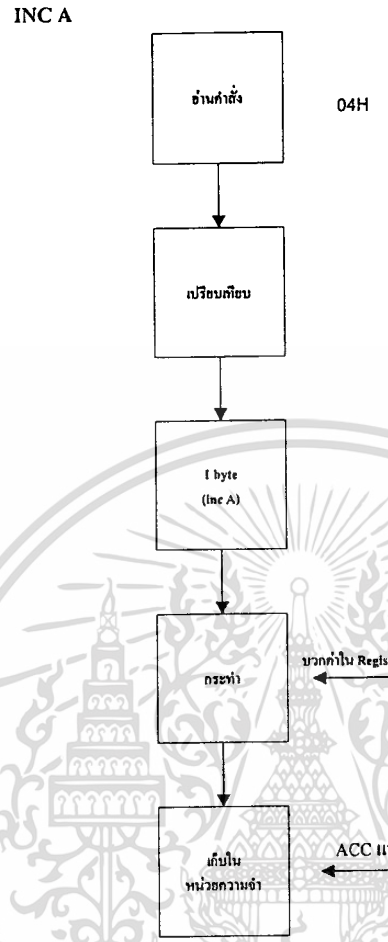
2.7 การ Run คำสั่งของ MCS-51

ทำได้โดยการอ่านค่าจาก Program memory จำลอง และนำไปเปรียบเทียบกับตารางหน่วยความจำ ซึ่งข้อมูลในตารางจะทำตามรูปแบบ และชนิดของค่าสิ่งนั้น ๆ ถ้าได้ผล หรือผลของการกระทำใดๆ จะเก็บลงในหน่วยความจำเปรียบเทียบกับการทำงานจริงของ CPU ไมโครคอนโทรลเลอร์เบอร์ 8051

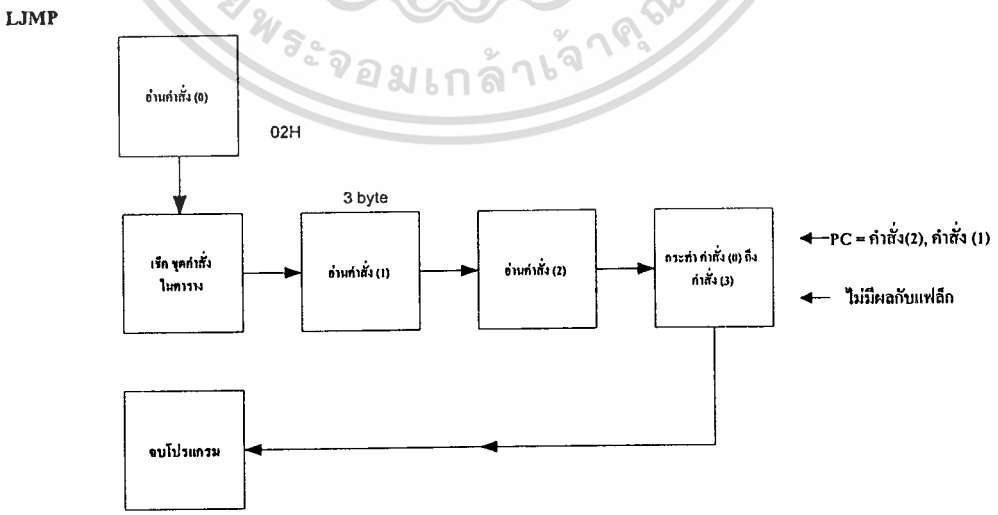


รูปที่ 2.12 โครงสร้างในการรันชุดคำสั่ง 3 ไบต์

จากรูปที่ 2.12 จะสังเกตได้ว่าเมื่อ อ่านคำสั่งจาก Program Memory จะนำไปเปรียบเทียบกับตาราง ถ้าตรงกัน คำสั่งที่มีขนาด 1 Byte จะกระทำตามคำสั่งนั้นได้เลย แต่ถ้าเป็นคำสั่ง 2 Byte อ่านคำสั่งอีกครั้งจากนั้นจึงไปกระทำตามคำสั่ง และถ้าเป็นคำสั่งที่มีขนาด 3 Byte จะต้องอ่านคำสั่งเพิ่มอีก 2 ครั้ง รวมเป็น 3 ครั้ง จากนั้นจึงจะทำตามคำสั่งนั้นได้ ซึ่งสามารถแสดงให้เห็นดังขั้นตอนการทำงานของคำสั่ง INC A และคำสั่ง LJMP ดังรูปที่ 2.13 และรูปที่ 2.14 ตามลำดับ



รูปที่ 2.13 ขั้นตอนการทำงานของคำสั่งขนาด 1 ไบต์



รูปที่ 2.14 ขั้นตอนการทำคำสั่ง 2 ไบต์

2.8 การเช็คแฟล็กในชุดคำสั่ง ของ MCS-51

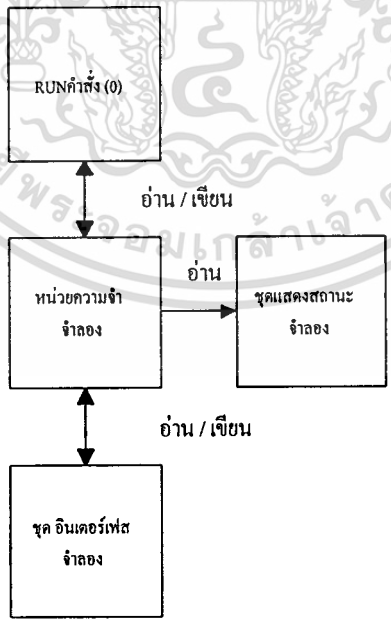
การเช็คแฟล็กในโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS-51 จะตรวจสอบการกระทำแฟล็กในที่ต้องการรันคำสั่ง MCS-51 ซึ่งคำสั่งที่กระทำเกี่ยวกับแฟล็กจะเป็นคำสั่งเกี่ยวข้องกับรีจิสเตอร์ A ซึ่งในการเขียนโปรแกรมจะยกเอาคำสั่งที่มีผลกับแฟล็กไว้อีกพวกหนึ่ง

2.9 การติดต่อกับชุดอินเตอร์เฟสจำลอง และ ชุดแสดงสถานะจำลอง

เมื่อทำการรันคำสั่ง MCS-51 ผลของการทำงานคำสั่งนั้นจะนำไปแสดงผลออกหน้าจอภาพ ซึ่งแบ่งเป็น 2 ส่วน คือ

1. ชุดแสดงสถานะจำลอง
2. ชุดอินเตอร์เฟสจำสอง

ชุดแสดงสถานะจำลอง คือ ชุดที่แสดงผลรีจิสเตอร์ภายใน , หน่วยความจำโปรแกรมภายนอก และหน่วยความจำข้อมูลภายนอก ในการแสดงผลจะนำไปแสดงผลในรูปแบบเท็ก แสดงค่าของการเปลี่ยนแปลงเป็นตัวเลข โดยมีกระบวนการทำงานแสดงดังรูปที่ 2.15



รูปที่ 2.15 ความสัมพันธ์ของตัวแสดงผลต่างๆ เมื่อโปรแกรมทำงาน

2.10 การสร้าง Help File ด้วย Microsoft Help Workshop 4.0

2.10.1 สิ่งที่ต้องใช้ในการสร้าง Help File

1. Microsoft Help Workshop ซึ่งจะมีอยู่ใน โปรแกรม Delphi หรือ C++ Builder
2. โปรแกรม Delphi
3. Microsoft Word ตั้งแต่เวอร์ชั่น 6 ขึ้นไป

2.10.2 ข้อจำกัดของระบบ Help

เทคโนโลยี 32 บิต ของ Windows 95 นั้นจะยอมให้การสร้าง Help ไฟล์ ได้โดยใช้โปรแกรมทางด้าน WinHelp โดยจะอยู่ภายใต้ข้อจำกัด ดังตารางที่ 2.2

ตารางที่ 2.2 ข้อจำกัดของระบบ Help

หัวข้อ	ข้อจำกัด
ขนาดของ Help ไฟล์	2 กิกะไบต์
หัวข้อต่อหัวข้อไฟล์ .rtf	ไม่จำกัด
หัวข้อต่อ Help ไฟล์	ไม่จำกัด
หัวข้อต่อ keyword	64,000
ความยาวหัวข้อ footnote	16,383 ตัวอักษร
ความยาว keyword	255 ตัวอักษร
สตริง Help title	127 ตัวอักษร
สตริง Topic title	127 ตัวอักษร
สตริง Custom window title	50 ตัวอักษร
ชื่อ Custom window	8 ตัวอักษร
สตริง Copyright	255 ตัวอักษร
สตริง Browse	50 ตัวอักษร
รูปภาพอ้างอิง	65,535 รูปภาพ ต่อ Help ไฟล์
ชื่อไฟล์	259 ตัวอักษร
ชื่อ ตัวอักษร	31 ตัวอักษร
ความยาวตัวอักษร	20

ตารางที่ 2.2 (ต่อ) ข้อจำกัดของระบบ Help

หัวข้อ	ข้อจำกัด
ไฟล์ ผิดพลาด	ไม่จำกัด
สตริง Citation	2,000 ตัวอักษร
การกำหนด Window	255 ต่อ โปรเจกต์ไฟล์
ชื่อ Window	50 ตัวอักษร
ไฟล์สารบัญ	ไม่จำกัด
หัวข้อสารบัญ	9 ระดับ
สตริงหัวข้อสารบัญ	255 ตัวอักษร
ข้อความหัวข้อสารบัญ	ไม่จำกัด

2.10.3 การสร้าง Topic File

1. เปิดโปรแกรม word processor หรือ โปรแกรม text editor ที่สามารถทำเป็นไฟล์ rich text format หรือ ไฟล์ (*.rtf) ได้
2. เขียนหัวข้อแต่ละหัวข้อโดยแยกจากกันด้วยเส้นแบ่งหน้า
3. เพิ่มเชิงอรรถ (footnote) ที่จุดเริ่มต้นของไฟล์
4. บันทึกไฟล์จากโปรแกรมให้มีนามสกุลเป็น rich text format (*.rtf)

<u>หมายเหตุ</u>

ก่อนที่จะทำการสร้าง Topic File ที่สมบูรณ์ให้คุณทดสอบไฟล์ word หรือ text editor ของคุณด้วยการคอมไพล์แบบทดสอบไฟล์เพียงหัวข้อสั้นๆ ก่อน

2.10.4 การเพิ่ม topic ID ที่หัวข้อ

1. ทำการแทรกที่ตำแหน่งเริ่มต้นของหัวข้อที่ต้องการเพิ่ม topic ID
2. แทรกเครื่องหมาย (#) ที่เชิงอรรถ (footnote)
3. พิมพ์ชื่อ topic ID ที่ส่วนของการอธิบายเชิงอรรถโดยมีข้อจำกัดดังนี้
    1. topic ID สามารถมีช่องว่างได้แต่ควรหลีกเลี่ยงด้านหน้าและด้านหลัง
    2. ไม่ใช้ตัวอักษรที่สงวนไว้แล้ว คือ (# = + @ * % !)
    3. ไม่ใช้ตัวอักษรเกินกว่า 255 ตัวอักษร
    4. ไม่ควรเริ่ม topic ID ด้วยตัวเลขถ้าจะใช้ ID ในส่วนของ MAP

2.10.5 การเพิ่ม title footnote ที่หัวข้อ

1. ทำการแทรกที่ตำแหน่งเริ่มต้นของหัวข้อที่ต้องการเพิ่ม title footnote
2. แทรกเครื่องหมาย ($) ที่เชิงอรรถ (footnote)
3. พิมพ์ title ที่ส่วนของการอธิบายเชิงอรรถ

หมายเหตุ

title สามารถกำหนดได้มากว่า 255 ตัวอักษร โดยจะนับรวม ช่องว่างด้วย

2.10.6 การสร้าง Index สำหรับหัวข้อ

1. ทำการแทรกที่ตำแหน่งเริ่มต้นของหัวข้อที่ต้องการ
2. แทรกตัวอักษร K ตัวใหญ่ ที่เชิงอรรถ (footnote)
3. พิมพ์รายการของ keyword ที่ส่วนของการอธิบายเชิงอรรถ โดยมีข้อสังเกตจำกัดดังนี้
    1. แยก keyword แต่ละตัวด้วยเครื่องหมาย (;)
    2. ไม่สามารถใช้ช่องว่างก่อนหรือหลัง keyword ได้
    3. ไม่ใช้เป็นพาหะส่งค่ากลับ
    4. ไม่ใช้ตัวอักษรเกิน 255 ตัวอักษร ต่อ หนึ่ง keyword ได้

หมายเหตุ

KLink macro จะทำการค้นหาหัวข้อที่มีตัวอักษร K ตัวใหญ่ที่อยู่หน้าหัวข้อนั้น

2.10.7 การสร้าง Project File

1. ในโปรแกรม Help Workshop กดที่ปุ่ม File menu และกดที่ปุ่ม New
2. ดับเบิลคลิกที่ Help Project
3. พิมพ์ชื่อของโปรเจกต์(.hpj) ไฟล์ และคลิกที่ปุ่ม OK

หมายเหตุ

เมื่อคุณสร้างโปรเจกต์ไฟล์ โปรแกรม Help Workshop ก็จะทำการสร้าง Help ไฟล์ที่คุณต้องการได้โดยก่อนที่คุณจะทำการคอมไพล์คุณควรเลือกไฟล์ .rtf ให้กับโปรเจกต์ไฟล์ก่อน

2.10.8 การอ้างถึงตำแหน่งของไฟล์รูปภาพ

1. ใน Help Workshop เปิด โปรเจกต์ไฟล์
2. คลิกที่ปุ่ม Bitmaps
3. คลิกที่ปุ่ม Add และอ้างไปยังที่อยู่ของรูปภาพที่เราต้องการ

2.10.9 การยอมให้โปรแกรมแสดงเพียงหัวข้อเดียว

1. ใน Help Workshop เปิด โปรเจกต์ไฟล์
2. คลิกที่ปุ่ม Map
3. ถ้าคุณต้องการเพิ่มหัวข้อเดี่ยวๆ ให้คลิกที่ปุ่ม Add
4. ถ้าคุณต้องการอินคูดไฟล์จากโปรแกรมภาษาซี ให้คลิกที่ปุ่ม Include

2.10.10 การชี้บอกหัวข้อของ Help File

1. ใน Help Workshop เปิด โปรเจกต์ไฟล์
2. คลิกที่ปุ่ม Options
3. ในบล็อกของ Help Title ให้พิมพ์หัวข้อที่ต้องการ

<u>หมายเหตุ</u>

หัวข้อของ Help ที่กำหนดสามารถมีได้ถึง 127 ตัวอักษร

2.10.11 การกำหนด Contents File ที่ Help File

1. ใน Help Workshop เปิด โปรเจกต์ไฟล์
2. คลิกที่ปุ่ม Options
3. คลิกที่แท็บ Files
4. ในบล็อกของ content ไฟล์ ให้พิมพ์ชื่อของ content ไฟล์ ที่ต้องการ

2.10.12 การกำหนดหน้าต่างที่สอง

1. ใน Help Workshop เปิด โปรเจกต์ไฟล์
2. คลิกที่ปุ่ม Windows
3. คลิกที่ Add
4. พิมพ์ชื่อของหน้าต่างที่สอง ไม่พิมพ์ชื่อเดียวกับที่กำหนดไว้ในหน้าต่างหลัก

หมายเหตุ

1. คุณสามารถที่จะกำหนดหน้าต่างที่สองได้ถึง 255 หน้าต่าง
2. คุณสามารถกำหนดการเข้าถึงของหน้าต่างเป็นแบบอัตโนมัติกับแบบแบ่งเป็นหัวข้อเดี่ยว

2.10.13 การสร้าง Help File

เมื่อได้ทราบถึงสิ่งที่ต้องใช้และทราบข้อกำหนดต่างๆ ในการสร้าง Help File แล้วจุดต่อไปคือ การกำหนดว่าจะเป็นโปรแกรมช่วยเหลือในลักษณะใด

1. ส่วนประกอบ

รูปแบบของการสร้างไฟล์ช่วยเหลือนี้ หรือ Help File จะสร้างมาจากไฟล์เอกสารของ Microsoft Word แล้วบันทึกไว้อยู่ในรูปแบบไฟล์ RTF (RICH TEXT FORMAT) หลังจากนั้นเราจะใช้ Help Compiler เป็นตัว แปลงไปเป็นไฟล์ให้มีนามสกุล .hlp (Help File) และจะมีไฟล์ที่มีนามสกุล .cnt เป็นตัวกำหนด Content ของ Help

2. ขั้นตอนการสร้าง

ก่อนอื่นให้สร้างไฟล์ Content ซึ่งเป็นไฟล์ Help Contents การสร้างนี้จะเรียกจากโปรแกรม HCW.EXE ซึ่งเป็นโปรแกรม Microsoft Help Workshop การที่จะสร้างไฟล์ Contents นี้จะต้องลำดับหัวข้อหลักและย่อย ฉะนั้นจะต้องมีรูปแบบของการแบ่งและจัดลำดับของข้อความที่ต้องการอธิบายไว้ก่อนที่จะจัดสร้างขึ้นมา

เมื่อลำดับเสร็จแล้ว เราจะมาสร้าง Help Contents กัน เริ่มจากเลือกไปที่ File New และเลือก Help Contents (ส่วนนี้จะเป็นการกำหนดชื่อไฟล์ Contents และ Title ที่แสดงขณะที่เปิด Help ขึ้นมา) หลังจากที่กำหนดชื่อไฟล์และ Title เสร็จแล้วให้กำหนดลำดับของ Help กดปุ่ม Edit อันที่สอง (อันล่าง)

ให้กำหนด Title และ Topic ID เพื่อที่จะระบุการเชื่อมไปยัง RTF(RICH TEXT FORMAT) ไฟล์ได้ ส่วนตรง Help File ซึ่งสามารถระบุ Help File ที่ต้องการเรียกได้ (ดูวิธีการกำหนด Topic ID จากหัวข้อข้างต้น)

เมื่อได้พิมพ์อะไรทุกอย่างตาม ที่ต้องการแล้วคราวนี้ก็มาถึงการบันทึกข้อมูล โดยเลือกไปที่ File/Save As... เลือกไฟล์ Type ไปที่ Save As Type ไปที่ Rich Text Format ใส่ชื่อไฟล์ตามที่ต้องการ

กลับมาที่ Microsoft Help Workshop ต่อ คราวนี้ให้สร้าง Help Project ขึ้นมา ตัวนี้จะเป็นตัวระบุรูปแบบของไฟล์ Help เช่นนำข้อมูลมาจาก rtf ไฟล์อะไร มีรูปแบบอะไรบ้าง และอื่นๆ อีกมากมาย

ให้เลือกไปที่ File/New/Help Project และใส่ชื่อไฟล์ลงไป จากนั้นก็ทำการระบุชื่อไฟล์ RTF ที่ใช้

การกำหนด RTF ไฟล์นี้ถ้ามีการแบ่งออกเป็นหลายๆ ไฟล์ก็สามารถนำมาเพิ่มลงไปให้ครบได้ เมื่อกำหนดแล้ว จะมีส่วน Section FILES ขึ้นมา เพื่อบอกว่ามีการนำไฟล์ RTF อะไรมาใช้บ้าง ขั้นตอนต่อไปคือการนำไปคอมไพล์เพื่อสร้างเป็น *.HLP ไฟล์ ซึ่งก็จะเป็นไฟล์สมบูรณ์ที่สามารถนำไปใช้แสดงข้อมูลได้ การคอมไพล์นี้ให้เลือกไปที่ File/Compile ตัวคอมไพล์จะแจ้งให้ทราบว่ามีการทำอะไรไปบ้างใน Help File

จากขั้นตอนต่างๆ ที่ได้กล่าวมาแล้วก็จะทำให้สามารถที่จะสร้าง Help File ได้ เพื่อใช้ประโยชน์ในการเรียนรู้การทำงานของโปรแกรมได้ด้วยความสะดวกสบาย

บทที่ 3
การออกแบบและการสร้าง

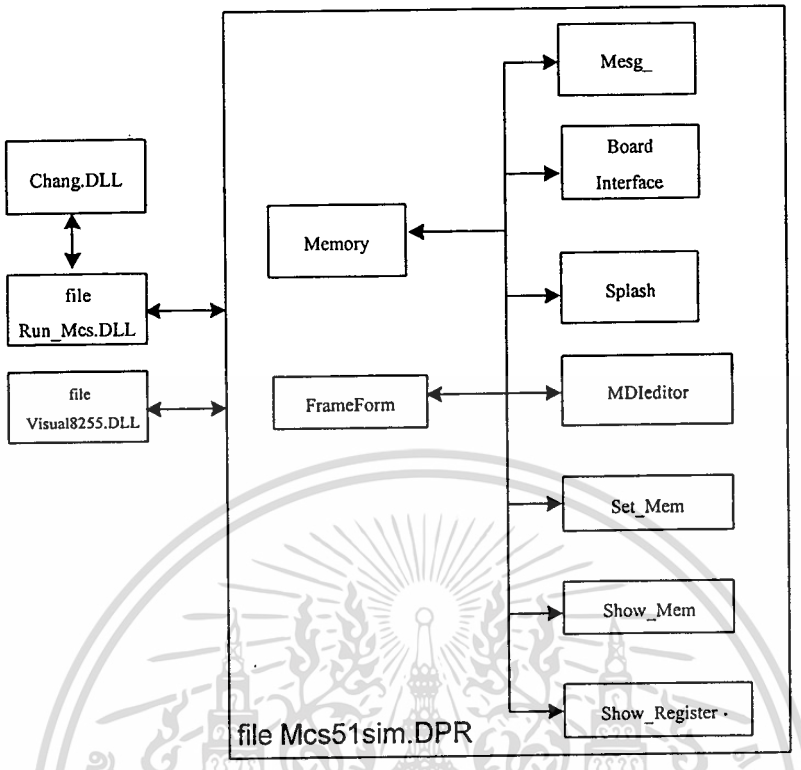
3.1 กล่าวนำ

การออกแบบและการสร้างโปรแกรม ได้แบ่งการทำงานของโปรแกรมออกเป็นส่วนๆ เพื่อให้ง่ายต่อการเขียนโปรแกรมและการพัฒนาโปรแกรมต่อไปในอนาคต ซึ่งเราได้แบ่งโปรแกรมออกเป็น 2 ส่วนใหญ่ๆ คือ ส่วนของการแสดงผล และส่วนของการคำนวณภายใน ซึ่งในส่วนของการแสดงผลนั้นเราจะใช้การทำเทรด (thread) เข้ามาช่วยในการเขียนโปรแกรม ซึ่งสามารถอธิบายรายละเอียดต่างๆ ของการออกแบบและการสร้างโปรแกรม ได้ดังนี้

3.2 โครงสร้างของโปรแกรม

โครงสร้างของโปรแกรมแบ่งออกเป็นส่วนๆ ดังนี้

1. ฟอร์มเมนูหลักของโปรแกรม ทำหน้าแสดงผลกับรับข้อมูลจากผู้ใช้
2. ฟอร์มแสดงการอินเตอร์เฟสจำลอง ทำหน้าแสดงผลรับข้อมูลจากผู้ใช้
3. ฟอร์มแสดงรูปการเข้าสู่โปรแกรมของโปรแกรม ทำหน้าแสดงผล
4. ฟอร์มอิดิเตอร์ ทำหน้ารับข้อมูลจากผู้ใช้
5. หน่วยความจำจำลอง
6. ฟอร์ม Set Memory ทำหน้าแสดงผลกับรับข้อมูลจากผู้ใช้
7. ฟอร์มแสดงผลข้อมูลในหน่วยความจำ ทำหน้าแสดงผล
8. ฟอร์มแสดงผลข้อมูลริจิสเตอร์จำลอง ทำหน้าแสดงผล
9. ชุดคำสั่งของ MCS - 51
10. ฟังก์ชันการกระทำระดับบิต
11. ฟังก์ชัน 8255 จำลอง



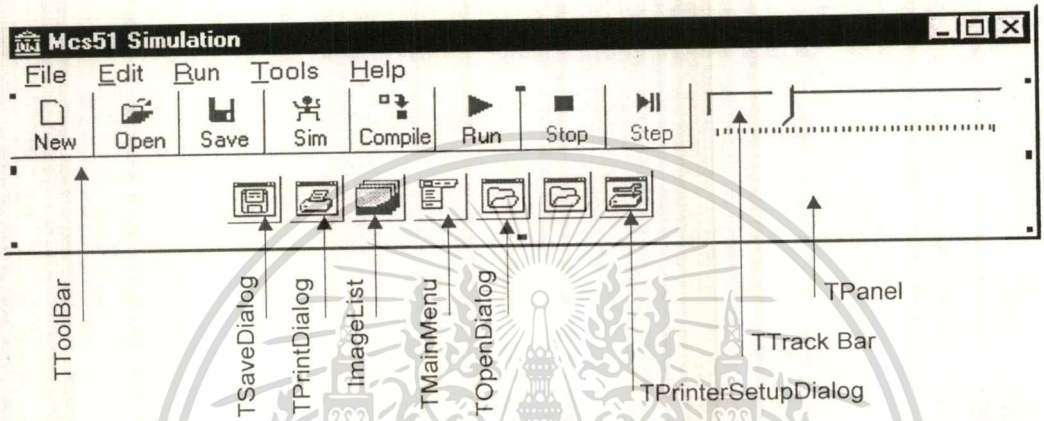
รูปที่ 3.1 โครงสร้างของโปรแกรม

โครงสร้างของโปรแกรมในรูปที่ 3.1 ประกอบด้วย

Formfram	คือ ฟอร์มเมนูหลัก
B_interface	คือ ฟอร์มชุดอินเตอร์เฟสจำลอง
MDIeditor	คือ ฟอร์มที่รับข้อมูลอินพุทจากผู้ใช้
Set_Mem	คือ ฟอร์มที่รับข้อมูลจากผู้ใช้ซึ่งกำหนดค่าของหน่วยความจำจำลอง
Show_Mem	คือ ฟอร์มที่แสดงผลข้อมูลในหน่วยความจำจำลอง
Show_Register	คือ ฟอร์มที่แสดงผลข้อมูลในรีจิสเตอร์จำลอง
Mesg_	คือ ฟอร์มที่แสดงผลข้อความการทำงานให้ผู้ใช้รับทราบ
Memory	คือ หน่วยความจำ MCS-51 จำลอง
Run_Mcs	คือ ชุดคำสั่งจำลองของ MCS-51
ChangeDLL	คือ ชุดของฟังก์ชันกระทำระดับบิต
Splash	คือ ฟอร์มชื่อโปรแกรม
Visual8255	คือ ฟังก์ชันจำลองการทำงาน 8255
Mcs51sim	คือ โปรเจคไฟล์โปรแกรม Mcs51Sim

3.3 การสร้างฟอร์มเมนูหลักของโปรแกรม

ฟอร์มเมนูหลักมีหน้าที่ในการรับคำสั่งการทำงานหลักๆ ของโปรแกรม ซึ่งคำสั่งที่ทำงานหลักๆ ของโปรแกรมประกอบ ไปด้วย New file, Open file, Save file, Compile และ Run โดยฟอร์มที่สร้างเป็นฟอร์มแบบ Win95 form ซึ่งทำงานแยกอิสระต่อกันดังรูปที่ 3.2

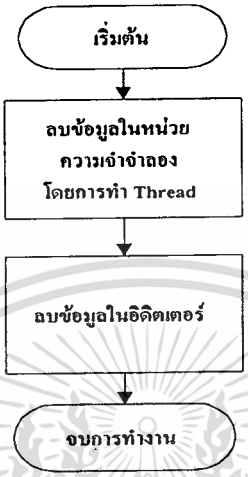


รูปที่ 3.2 ชื่อคอมโปเนนต์ของฟอร์มเมนูหลัก

3.3.1 รายละเอียดในการกำหนด TMainmenu ประกอบด้วย

- File ประกอบด้วย New , Open, Save ,Save As , Printsetup , Print ,Exit
- Edit ประกอบด้วย Copy ,Cut, Paste, Select All
- Run ประกอบด้วย Sxa51 Compile, Compile, Run, Stop, Step, Change Mode
- Tool ประกอบด้วย Set Memory, Refresh All, Address 8522, Address Show, Editor, Register, Memory, Message, BoardSimulation
- ภายใน Address Show ประกอบด้วย Internal Memory, External Memory, Program Memory

3.3.2 คำสั่ง New Text File

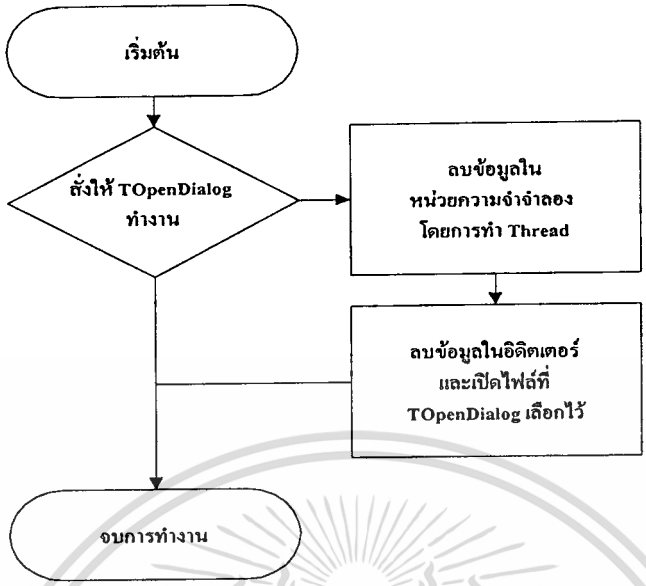


รูปที่ 3.3 กระบวนการในการทำงานของ New Text File

จากรูปที่ 3.3 เมื่อเริ่ม New Button โปรแกรมจะลบข้อมูลที่อยู่ภายใน หน่วยความจำจำลอง โดยการเทรด เพื่อลดระยะเวลาในการทำลูปเพื่อลบข้อมูล ซึ่งในการลบข้อมูลจะใช้ระยะเวลานาน ดังนั้นจึงต้องกำหนดให้เป็นเทรดเพื่อแบ่งเวลาในการทำงานให้โปรแกรมไปทำงานในส่วนอื่น

3.3.3 คำสั่ง Open Text File

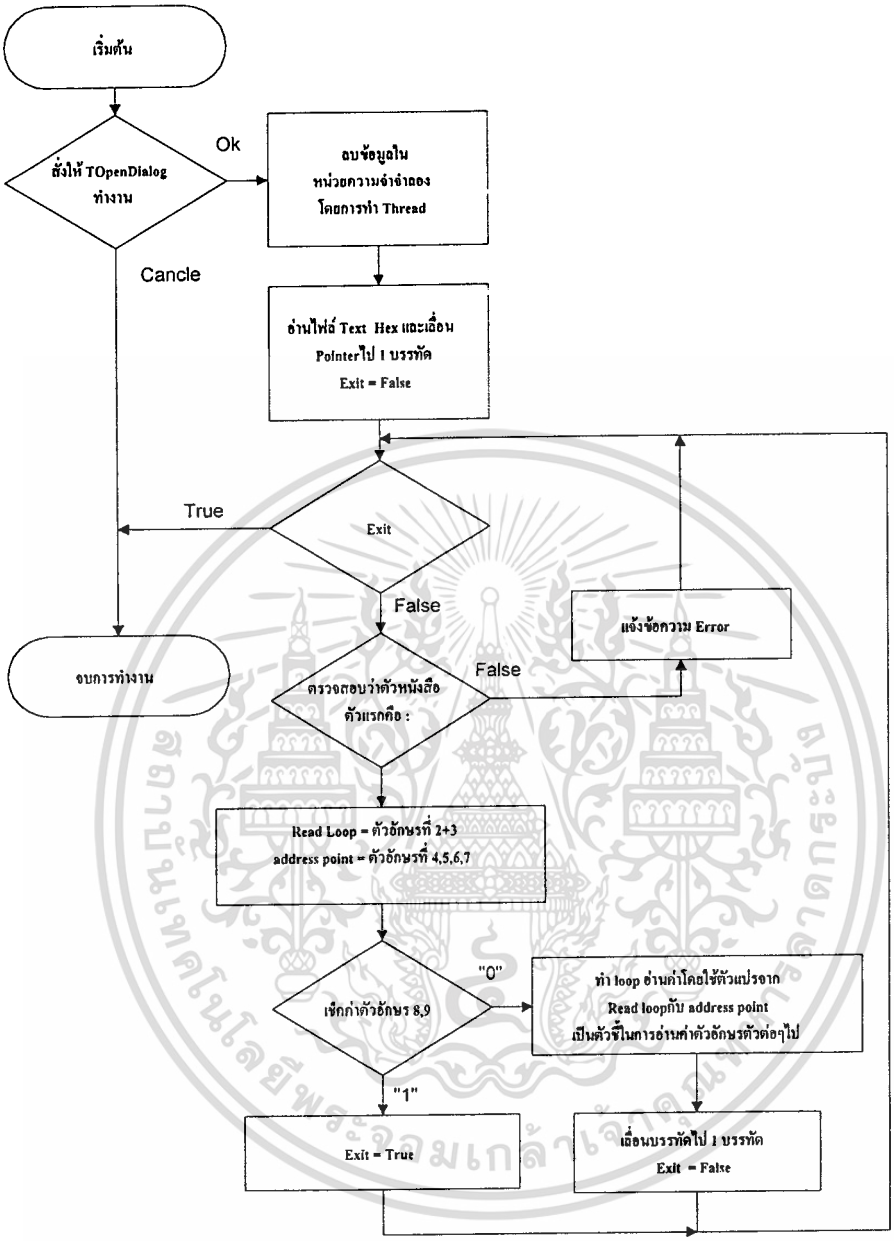
คำสั่ง Open เรียกใช้ไดอะล็อก (Dialog) มาตราฐานของโปรแกรมซึ่งได้กำหนดให้เปิดไฟล์ในรูปแบบ เท็กไฟล์ (Text File)และเก็บข้อมูลของเท็กไฟล์ที่เปิดในตัวแปรเท็กไฟล์ซึ่งเป็นตัวแปรแบบสตริง (String) จากนั้นทำการลบข้อมูลภายในหน่วยความจำจำลอง และสั่งให้อิดิเตอร์ (Editor) เปิดไฟล์ โดยใช้ชื่อตัวแปรของเท็กไฟล์ในการเปิด ดังแสดงการทำงานในรูปที่ 3.4



รูปที่ 3.4 กระบวนการในการทำงานของ Open Text File

3.3.4 คำสั่ง Open Hex File Format

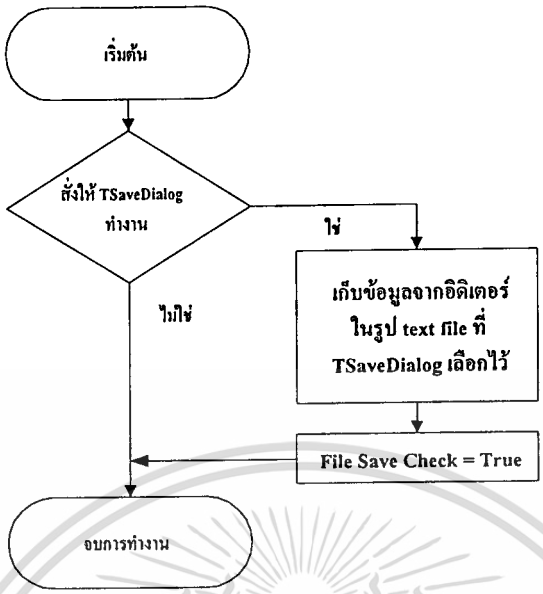
โปรแกรมจำลองการทำงานได้ออกแบบให้ผู้ใช้สามารถเปิดไฟล์ ในรูปแบบเฮกฟอร์แมต (Hex format) ได้ เมื่อได้ชื่อไฟล์จาก OpenDialog โปรแกรมจะลบข้อมูลในหน่วยความจำจำลองทั้งหมด ด้วยกระบวนการเทรด จากนั้นจะเข้าสู่การเปลี่ยนไฟล์จากรูปแบบของเท็กไฟล์ไปเป็น รูปแบบของข้อมูลเฮกไฟล์ (Hex File) โดยเลื่อนบรรทัดไป 1 บรรทัด และทำการตรวจสอบว่าตัวอักษรตัวแรกเป็นตัว ":" หรือไม่ ถ้าไม่ใช่ให้ออกจากการทำโปรแกรมย่อยตัวนี้ เมื่อตรวจสอบแล้วว่าใช่ให้อ่านตัวอักษรตัวต่อไป ตัวที่ 2,3 ซึ่งจะเก็บค่าของ จำนวนความยาวของข้อมูลที่บรรจุอยู่ในบรรทัดนั้น โดยอยู่ในรูปของเลขฐาน 16 และจะต้องทำการแปลงจากข้อมูลในรูปแบบของ สตริง ให้อยู่ในรูปแบบของ ตัวเลข เพื่อนำไปใช้ในการทำลูปทดสอบ ต่อไป ตัวที่ 4,5,6,7 ซึ่งจะเก็บค่าของ ข้อมูลของแอดเดรสเริ่มต้นของข้อมูล เช่น 0010H แสดงว่า ตัวอักษรตัวแรกของชุดข้อมูลจะอยู่ที่ แอดเดรส 0010H ต่อไปเช็คตัวอักษรตัวที่ 8,9 ซึ่งจะบอกว่าในชุดข้อมูลนั้นว่าเป็นข้อมูลหรือคำสั่ง ถ้ามีค่า 00 แสดงว่าเป็นข้อมูล ก็จะทำการทำลูปเพื่ออ่านตัวอักษรตัวต่อไปเป็นคู่ๆ พร้อมกับนำค่าที่ได้ใส่ไปในหน่วยความจำเป็นชุดๆด้วย ตามจำนวนของตัวแปรข้อมูลสูงสุดที่เก็บไว้ในครั้งแรก แต่ถ้ามีค่าเป็น 01 แสดงว่าเป็นคำสั่งซึ่งตัวต่อไปจะมีค่าเป็น FFH แสดงว่าสิ้นสุดข้อมูลที่เก็บอยู่ในไฟล์นั้น ก็จะส่งข้อมูลออกไปยัง Exit ให้ออกจากการกระทำคำสั่งอ่านข้อมูล ดังแสดงการทำงานในรูปที่ 3.5



รูปที่ 3.5 กระบวนการในการทำงานของ Open Hex file format

3.3.5 คำสั่ง Save File As

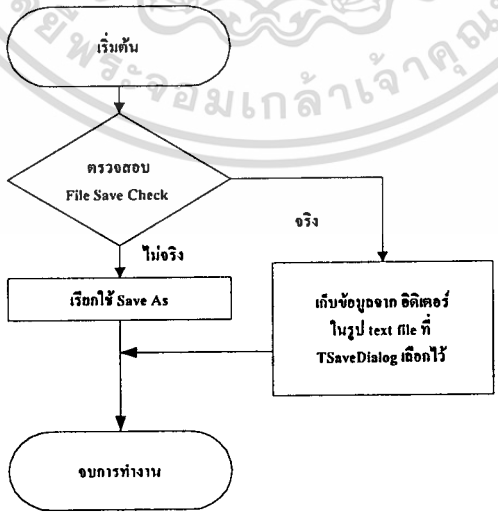
โปรแกรมจะเรียกใช้คอมโปเนนต์ (Component) TSaveDialog โดยข้อมูลที่เก็บจะกำหนดให้อยู่ในรูปเท็กไฟล์ฟอร์แมต (Text File Format) โดยการส่งให้อิดิเตอร์เป็นตัวเก็บข้อมูล และทำการเช็คค่าของ Save Check เพื่อตรวจสอบว่าได้มีการเก็บข้อมูลของไฟล์นั้นแล้ว ซึ่งจะรวมถึงการเก็บข้อมูลของชื่อไฟล์ที่ได้บันทึก (Save) ลงไปด้วย ดังแสดงการทำงานในรูปที่ 3.6



รูปที่ 3.6 กระบวนการในการทำงานของ Save File As

3.3.6 คำสั่ง Save File

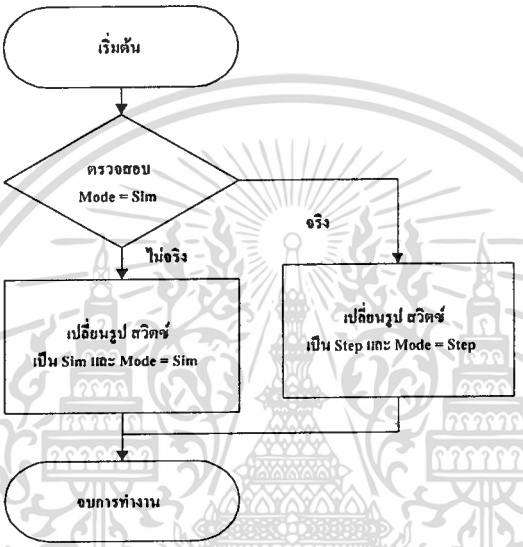
จะตรวจสอบว่าได้มีการใช้คำสั่ง Save File As ไปในครั้งแรกเพื่อกำหนดชื่อไฟล์ หรือยัง ซึ่งถ้ายังไม่ได้กำหนดจะเรียกใช้ Save File As เพื่อทำการเก็บข้อมูล แต่ถ้าเก็บค่าแล้วจะสั่งให้ อิดิเตอร์เก็บข้อมูลตามตัวแปรที่ได้กำหนดค่าเอาไว้ ดังแสดงการทำงานในรูปที่ 3.7



รูปที่ 3.7 กระบวนการในการทำงานของ Save File

3.3.7 คำสั่ง Change Mode

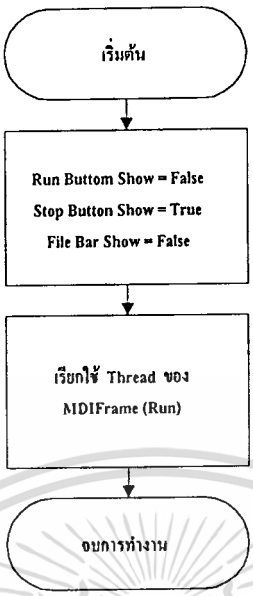
ในการรัน (Run)โปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์MCS-51 ได้กำหนดโหมดในการทำงานออกเป็น 2 โหมดคือ การจำลองการทำงาน (Simulation) กับ การทำงานทีละคำสั่ง (Run Step) ซึ่งในคำสั่ง Change Mode จะเป็นการปรับรูปแบบการรันโปรแกรม ดังแสดงการทำงานในรูปที่ 3.8



รูปที่ 3.8 กระบวนการในการทำงานของ Change Mode

3.3.8 คำสั่ง RUN

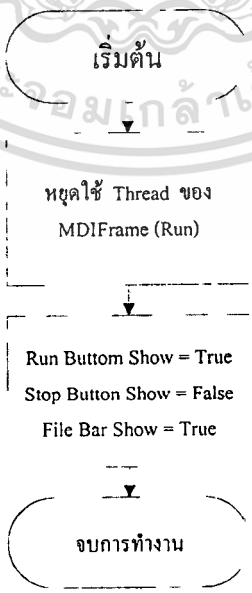
ในคำสั่ง Run จะกำหนดให้ ปิดปุ่ม (Button) ที่ไม่ได้ใช้และจะทำให้เกิดการทำงานที่ผิดพลาด เช่นเมื่อผู้ใช้ กดปุ่ม Run ซ้อนกันได้ 2 ครั้งทำให้เกิดเทรดซ้อนกัน 2 ตัว ทำให้โปรแกรมหยุดการทำงานได้ เพราะการแบ่งเวลาและการจัดหน่วยความจำคาบเกี่ยวกัน ดังนั้นจึงต้องปิดปุ่มที่ไม่ได้ใช้หรือใช้ไม่ได้ จากนั้นจึงกำหนดให้มีการสร้างเทรดของ MDIFRAME (Run) โปรแกรมขึ้นมา เมื่อเทรดของ MDIFRAME ทำงานจะกำหนดให้เทรดต่างๆ ของการแสดงผลของโปรแกรมทำงานด้วย ดังแสดงการทำงานในรูปที่ 3.9



รูปที่ 3.9 กระบวนการในการทำงานของ Run

3.3.9 คำสั่ง Stop

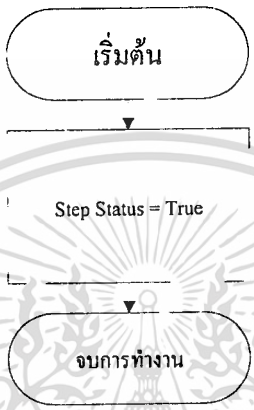
ปุ่มของ Stop จะแสดงเมื่อมีการกดปุ่ม Run ดังนั้นเมื่อปุ่ม Stop ถูกกด ปุ่ม Run จะต้องเอกทีฟ (Active) ขึ้นมาและจะต้องทำการปิดเทรดที่ทำการสร้างขึ้นมาเพื่อไม่ให้การทำงานของโปรแกรมค้าง โดยเทรดที่เปิดจะถูกปิดทั้งหมด ดังแสดงการทำงานใน รูปที่ 3.10



รูปที่ 3.10 กระบวนการในการทำงานของ Stop

3.3.10 คำสั่ง Step

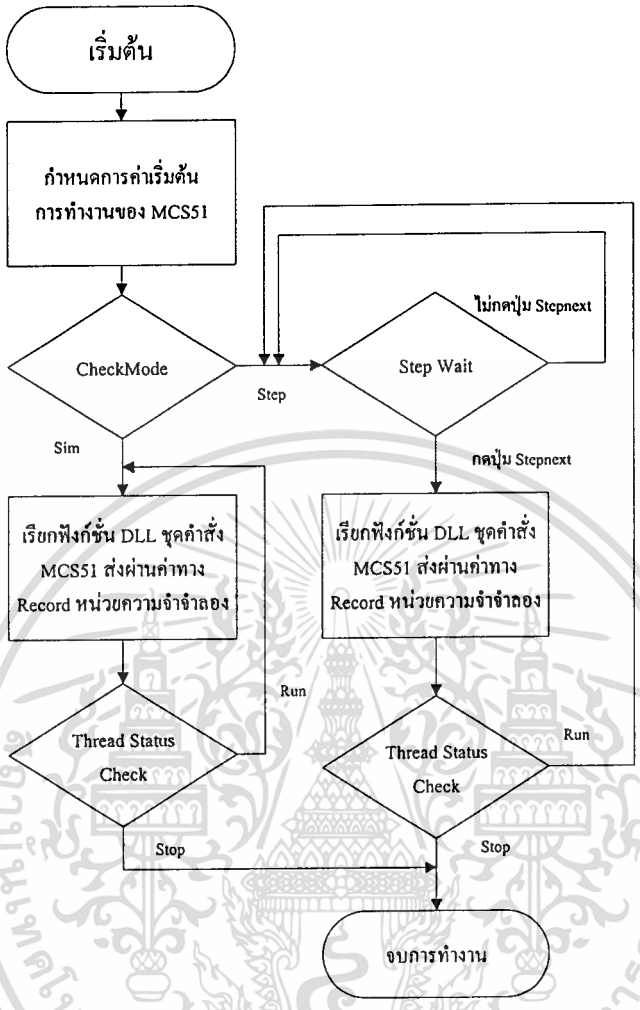
เมื่อสั่งให้โปรแกรมทำงานในโหมด Step การที่จะทำให้โปรแกรมทำงานต่อเนื่องไปได้จะต้องมีการกดปุ่ม Step เพื่ออนุญาตให้มีการอ่านชุดคำสั่งต่อไป และเมื่อทำการอ่านชุดคำสั่งต่อไปเสร็จก็จะรอจนกว่าจะมีการกดปุ่ม Step อีกครั้ง ดังแสดงการทำงานในรูปที่ 3.11



รูปที่ 3.11 กระบวนการในการทำงานของ Step

3.3.11 Thread MDIFrame (Run)

ในเทรดของ MDIFrame จะเป็นการทำงานแบบมัลติทาสกิง (Multitasking) ซึ่งจะแบ่งเวลาไปทำงานในส่วนอื่นๆของโปรแกรมได้ กระบวนการทำงานของเทรดตัวนี้เริ่มจากการกำหนดค่าเริ่มต้นของ MCS-51 เช่นกำหนดให้ Sp=7 เป็นต้น และยังอนุญาตให้เทรดที่เกี่ยวข้องในการแสดงผลของโปรแกรมทำงานด้วย โดย Thread MDIFRAME จะมีความเร็วเท่ากับ Primary Thread ส่วนเทรดตัวอื่นๆ จะมีความเร็วในการทำงานรองลงมา ต่อจากนั้นทำการเช็คว่าโปรแกรมทำงานอยู่ในโหมด Step หรือไม่ ถ้าทำงานในโหมด Step จะเข้าสู่การควบคุมแบบ Step โดยมีการรอเพื่อให้ผู้ใช้กดปุ่ม Step จึงจะทำงาน ส่วนการทำงานในโหมด Simulation จะไม่มีการรอการกดปุ่ม Step แต่จะใช้การเช็คค่าระยะเวลาหน่วงจาก TrackBar เพื่อปรับระดับความเร็วในการจำลองการทำงาน ดังรูปที่ 3.12



รูปที่ 3.12 กระบวนการในการทำงานของ Thread MDIFRAME (Run)

3.4 การสร้างหน่วยความจำจำลองและตัวแปรเชื่อมโยงกับส่วนต่างๆ ของโปรแกรม

ในหน่วยความจำจำลอง จะมีโครงสร้างประกอบด้วยตัวแปร แบบ array มีหลายตัวรวมกัน มีโครงสร้างแบบ Record ซึ่งประกอบด้วย

Record ชื่อ TMcs_Memory

- Internal Data Memory มีขนาด 0 ถึง 255 ชนิด byte
- External Data Memory มีขนาด 0 ถึง 65535 ชนิด byte
- Program Data Memory มีขนาด 0 ถึง 65535 ชนิด byte

ส่วนการส่งข้อมูลผ่าน 8255 จำลอง ซึ่งจะต้องส่งข้อมูลของ พอร์ต PA,PB,PC และ Control word ผ่านตัวแปรที่รวมกันเป็น Record ประกอบด้วย

Record ชื่อ TMcs_Memory

- PA ชนิด byte
- PB ชนิด byte
- PC ชนิด byte
- Control Word ชนิด byte
- External Port สำหรับ 8255 ซึ่งคือ แอดเดรสเชื่อมโยง กับ External Data Memory โดยมีขนาด 0 ถึง 3 ชนิด byte
- PA ขนาด 0 ถึง 7 ชนิด Boolean เพื่อใช้ในการกระทำระดับบิต
- PB ขนาด 0 ถึง 7 ชนิด Boolean เพื่อใช้ในการกระทำระดับบิต
- PC ขนาด 0 ถึง 7 ชนิด Boolean เพื่อใช้ในการกระทำระดับบิต
- Control Word ขนาด 0 ถึง 7 ชนิด Boolean เพื่อใช้ในการกระทำระดับบิต

ตัวแปรและ Record ทั้งหมดที่สร้างขึ้นจะอยู่ที่ Tmemory ซึ่งผู้ใช้งานสามารถเรียกผ่าน Class เพื่อส่งผ่านค่าได้ โดยประกาศตัวแปรเป็นดังนี้

```
TMcs_Memory = record
In_Data_Mem:array[0..255] of byte;  //ใช้ใน Internal Data Memory
Ex_Data_Mem:array[0..65536] of byte; //ใช้ใน External Data Memory
External_prog_Mem:array[0..65536] of byte; //ใช้ใน External Program Memory
Address_point_ExProg:word; //ใช้เป็น Point ในการชี้ของ External Program Memory
Address_point_ExMem:word; //ใช้เป็น Point ในการชี้ของ External Data Memory
Address_point_IntData:byte; //ใช้เป็น Point ในการชี้ของ Internal Data Memory
Show_External_Prog:word; //ใช้เลื่อนตัวชี้ข้อมูลในการแสดงค่าของ Show Memory
Show_External_Mem:word; //ใช้เลื่อนตัวชี้ข้อมูลในการแสดงค่าของ Show Memory
Show_Internal_Data:byte; //ใช้เลื่อนตัวชี้ข้อมูลในการแสดงค่าของ Show Memory
Reg_bank:byte;//เก็บค่าว่า รีจิสเตอร์ R0-R7 อยู่ใน Bank ที่เท่าไร
End;
```

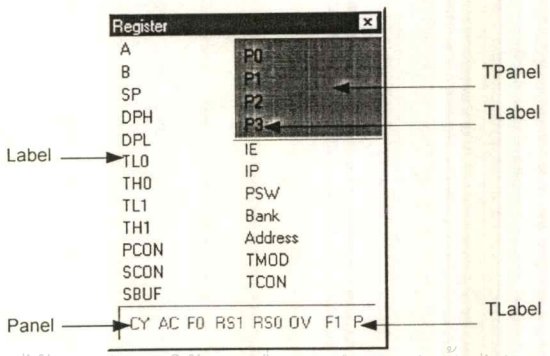
ตัวแปรที่กำหนดขึ้นไว้เพื่ออ้างถึงตำแหน่งของหน่วยความจำภายในของไมโครคอนโทรลเลอร์ MCS-51 ประกอบด้วยค่าคงที่ต่างๆ ดังตารางที่ 3.1

ตารางที่ 3.1 ค่าคงที่ (Constant) ที่ใช้ในโปรแกรม โดยเป็นค่าคงที่ของรีจิสเตอร์ภายใน

ชื่อตัวแปร	ค่าที่กำหนด	ชื่อตัวแปร	ค่าที่กำหนด	ชื่อตัวแปร	ค่าที่กำหนด
_ACC	$0e0	_PCON	$87	_TH1	$8d
_B	$0f0	_PSW	$0d0	_R0	$00
_DPH	$83	_SCON	$98	_R1	$01
_DPL	$82	_SBUF	$99	_R2	$02
_IE	$0a8	_SP	$81	_R3	$03
_IP	$0b8	_TMOD	$89	_R4	$04
_P0	$80	_TCON	$88	_R5	$05
_P1	$90	_TL0	$8a	_R6	$06
_P2	$0a0	_TH0	$8c	_R7	$07
_P3	$0b0	_TL1	$8b		

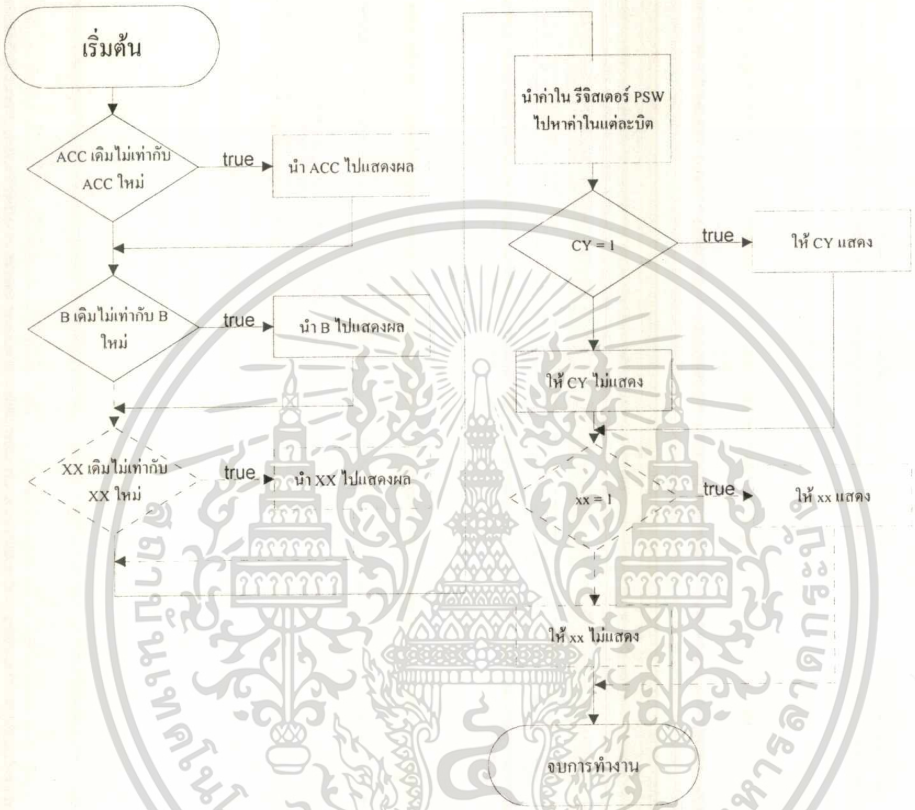
3.5 การสร้างฟอร์มแสดงสถานะของรีจิสเตอร์จำลอง

ฟอร์มแสดงสถานะของรีจิสเตอร์จำลองนี้จะต้องแสดงค่ารีจิสเตอร์ภายในโดยเรียกใช้ Class ของ Tmemory ซึ่งจะมีการประกาศตัวแปร และค่าคงที่เอาไว้ทำให้ในฟอร์มแสดงสถานะ รีจิสเตอร์จำลองไม่จำเป็นต้องประกาศตัวแปรอีก ลักษณะของฟอร์มแสดงสถานะของรีจิสเตอร์จำลองจะต้องมีการเปลี่ยนแปลงค่าของตัวเลข ดังนั้นจึงต้องมีการใช้เทรดเพื่อลดระยะเวลาในการแสดงผลและแบ่งเวลาระหว่างการทำงาน (Process) ในการแสดงผลให้ส่วนต่างๆของโปรแกรมได้ใช้ด้วย รูปของฟอร์มเป็นดังรูปที่ 3.13



รูปที่ 3.13 ฟอร์มแสดงสถานะของรีจิสเตอร์จำลอง

ฟอร์มแสดงสถานะใช้คอมโปเนนต์ Tlabel ในการแสดงข้อความโดยผ่านค่าให้ตรงๆ ผ่านทาง caption ของ Tlabel ส่วนในการแสดงของรีจิสเตอร์ PSW จะใช้การกำหนดคุณสมบัติการ Visible ของ Tlabel มาช่วยในการแสดงลักษณะโครงสร้างของโปรแกรมภายในที่ใช้แสดงเป็นดังนี้



รูปที่ 3.14 โครงสร้างการแสดงค่าของ Tlabel ต่างๆ ในฟอร์มแสดงสถานะรีจิสเตอร์จำลอง

จากรูปที่ 3.14 ในส่วนของบล็อกที่เป็นเส้นประนั้นจะแทนรีจิสเตอร์ต่างๆ ที่ไม่ได้แสดง ซึ่งละไว้ เพื่อให้ง่ายต่อการดู ชาร์ทในโปรแกรมส่วนของการแยกค่ารีจิสเตอร์ PSW ในแต่ละบิตนั้น ใช้การเขียนโปรแกรมเป็นภาษาแอสเซมบลี้เพื่อให้ง่ายต่อการกระทำระดับบิต ซึ่งลักษณะของคำสั่งในส่วนของ แอสเซมบลี้เป็นดังตารางที่ 3.2

ตารางที่ 3.2 คำสั่งภาษาแอสเซมบลี้ที่ใช้ในการเขียนโปรแกรมแสดงสถานะแฟล็ก

```
asm
        push ax
        push bx
        mov al,x
        and al,$01 //check p
        mov p,al
        mov al,x
        and al,$02 //check f
        shr al,$1
        mov f1,al
        mov al,x
        and al,$04 //check ov
        shr al,$2
        mov ov,al
        mov al,x
        and al,$08 //check rs0
        shr al,3
        mov rs0,al
        mov al,x
        and al,$10 //check rs1
        shr al,4
        mov rs1,al
        mov al,x
        and al,$20 //check f0
        shr al,5
        mov f0,al
        mov al,x
        and al,$40 //check ac
        shr al,6
        mov ac,al
        mov al,x
        and al,$80 //check cy
        shr al,7
        mov cy,al
        pop bx
        pop ax
    end;
```

เมื่อแยกค่าแต่ละตัวด้วย แอสเซมบลี้แล้วก็กำหนดการแสดงหรือไม่ให้ Tlabel นั้นทำการ Active

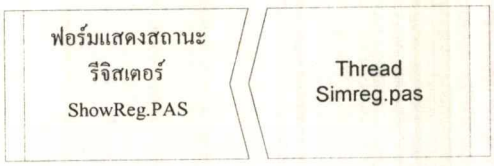
```
if p=0 then _P.Enabled:=false else _P.Enabled:=true;
if f0=0 then _f0.Enabled:=false else _f0.Enabled:=true;
if ov=0 then _OV.Enabled:=false else _OV.Enabled:=true;
if rs0=0 then _Rs0.Enabled:=false else _Rs0.Enabled:=true;
if rs1=0 then _Rs1.Enabled:=false else _Rs1.Enabled:=true;
if f0=0 then _F0.Enabled:=false else _F0.Enabled:=true;
if ac=0 then _AC.Enabled:=false else _AC.Enabled:=true;
if cy=0 then _CY.Enabled:=false else _CY.Enabled:=true;
```

Enable เป็นการกำหนดว่าจะเลือกให้ตัวอักษรนั้นแสดงผลหรือไม่ถ้าแสดงผล ตัวอักษรจะเอกทีฟขึ้นมา Enabel มีลักษณะเป็น Boolean

3.5.1 เทรดในฟอร์มแสดงสถานะของรีจิสเตอร์

ในการแสดงผลของฟอร์มแสดงสถานะของรีจิสเตอร์ฟอร์มเดิมถ้าไม่ใช้การทำเทรดการแสดงผลจะทำได้ช้ามาก เพราะโปรแกรมไม่สามารถแบ่งเวลาไปทำงานในส่วนอื่นๆได้ การทำเทรด เพิ่มเข้ามาในส่วนนี้จึงทำได้โดยการสร้างโปรแกรมขึ้นมาใหม่อีกชุดหนึ่งมีโครงสร้างเป็น

class แบบ Tthread ซึ่งตั้งชื่อ Class ที่สร้างขึ้นมาชื่อ SIMREG.PAS และ Class ของเทรดที่สร้างขึ้นมาจะเรียกใช้ตัวแปรที่ได้จากโปรแกรมแสดงสถานะรีจิสเตอร์ โดยใช้ชื่อว่า ShowReg ซึ่งมีส่วนประกอบดังรูปที่ 3.15



รูปที่ 3.15 ส่วนประกอบของฟอร์มแสดงสถานะของรีจิสเตอร์

คุณสมบัติของเทรดกำหนดให้มีความสำคัญเป็น TpNormal ซึ่งจะแบ่งเวลาการทำงานเป็นเวลาเดียวกับ Primary Thread และกำหนดเวลาในการ Wait ไว้ที่ 15 msec

```
constructor TVisualReg.Create(ARegister:TRegister);
begin
    Buf:=ARegister;
    inherited Create(true);
    priority := tpNormal;
end;
procedure TVisualReg.Doexecute;
begin
    repeat
     Synchronize(UpdataShape);
     sleep(15);
    until (terminated);
end;
```

รูปที่ 3.16 โปรแกรมสร้างเทรดให้กับฟอร์มแสดงรีจิสเตอร์

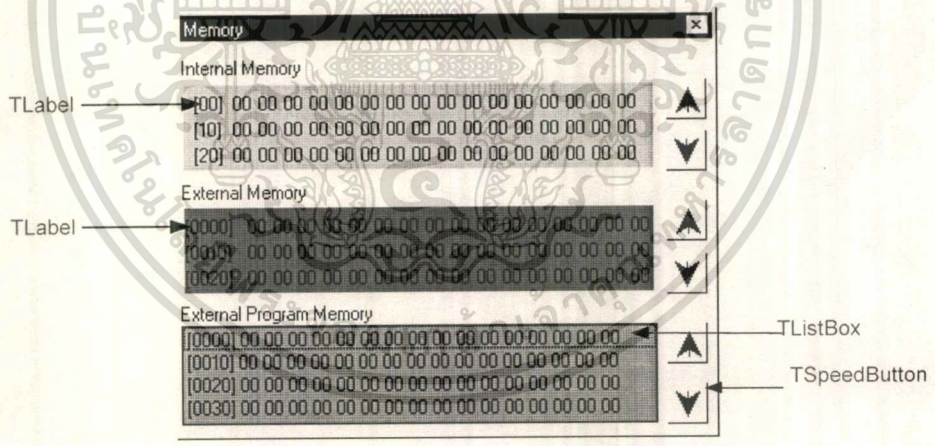
จากรูปที่ 3.16 ในการ Terminated ของ เทรด ตัวนี้จะกำหนดในสถานะแบบ Supped เท่านั้นจะไม่ใช้ Destroy เพราะต้องมาเสียเวลาในการทำ CreateThread ขึ้นมาใหม่ และการ Supped ก็ไม่กินเวลาในการทำงานของ CPU มากนัก

3.6 การสร้างฟอร์มแสดงข้อมูลของหน่วยความจำจำลอง

ในโปรแกรมจำลองการทำงานจะต้องมีการแสดงค่าของหน่วยความจำที่แอดเดรสต่างๆ โดยที่หน่วยความจำที่จะแสดงผล ประกอบด้วย

- Internal Data Memory ขนาด 256 byte
- External Data Memory ขนาด 65536 byte
- Program Data Memory ขนาด 65536 byte

ซึ่งหน่วยความจำทั้ง 3 ชนิดนี้ได้ประกาศเป็นตัวแปรอยู่ในส่วนของ Memory.pas ดังนั้นฟอร์มแสดงค่าของหน่วยความจำจำลองจึงเรียกใช้ข้อมูลจาก Memory.pas ในการออกแบบโครงสร้างฟอร์มได้ออกแบบให้ฟอร์มแสดงค่าของหน่วยความจำทั้ง 3 ชนิดพร้อมกันดังรูปที่ 3.17



รูปที่ 3.17 ฟอร์มแสดงค่าของหน่วยความจำจำลอง

ในส่วนของ Internal Data Memory ประกอบด้วยคอมโปเนนต์ Tlabel จัดวางเรียงกัน ทั้งหมด 48 ตัวแต่ละตัวมีชื่อไม่ซ้ำกัน ส่วนของ External Data Memory ประกอบด้วยคอมโปเนนต์ Tlabel จัดเรียงกัน ทั้งหมด 48 ตัวแต่ละตัวมีชื่อไม่ซ้ำกัน และส่วน Program Data Memory ใช้คอมโปเนนต์ TlistBox กำหนดการวางตัวอักษรจากข้อมูลสตริงในแต่ละบรรทัด มีทั้งหมด 4 บรรทัด จะสังเกตได้ว่า Internal, External กับ Program Data Memory จะใช้คอมโปเนนต์ในการ

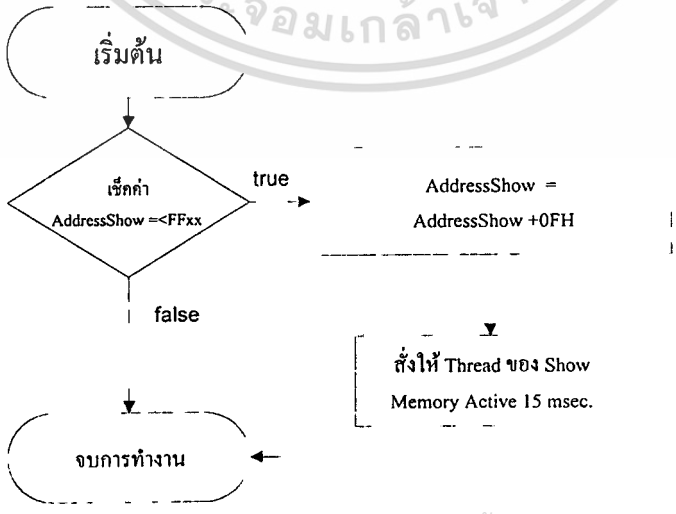
สร้างที่แตกต่างกัน สาเหตุเพราะข้อมูลที่ใช้ในการแสดงผลใน Tlabel ในชุดตัวอักษรแต่ละชุดจะมีระยะห่างที่เท่ากัน ส่วนใน TlistBox ข้อมูลในแต่ละชุดตัวอักษรจะถูกเลื่อนไม่ตามขนาดของตัวอักษรโดยเมื่อรันโปรแกรมข้อมูลที่จะเกิดการเปลี่ยนแปลงได้ Internal Data Memory กับ External Data Memory ส่วน Program Data Memory นั้น ผลของการรันจะไม่ทำให้ค่าที่แสดงผลของ Program Data Memory เปลี่ยน ในการแสดงผลเมื่อรันโปรแกรมตัวอักษรของ Internal,External Data Memory จึงไม่เกิดการเลื่อนของตัวอักษร จะสังเกตได้ว่าในการแสดงผลข้อมูลของหน่วยความจำนั้น จะแสดงเพียง 3 ถึง 4 บรรทัด เพื่อลดระยะเวลาในการทำงานของโปรแกรม และอีกทั้งไม่มีความจำเป็นที่ผู้ใช้จะต้องดูทั้งหมดของหน่วยความจำในเวลาเดียวกัน ถ้าหากผู้ใช้ต้องการเลื่อนแอดเดรสในการแสดงของหน่วยความจำก็ทำได้โดยการกดที่คอมโปเนนต์ TspeedButton ก็จะเลื่อนค่าบรรทัดในการแสดงผลของข้อมูล ซึ่งตามหลักการเลื่อนค่าการแสดงผลของข้อมูลจะต้องมีการเก็บค่า เพื่อใช้ในการตรวจสอบว่าจะแสดงอยู่ที่แอดเดรสที่เท่าไร โดยค่าทั้ง 3 ชนิดจะเก็บไว้ที่ Memory.pas ดังนี้

Internal Data Memory ตัวแปรที่ชี้ชื่อ Show_Internal_Mem ชนิด byte

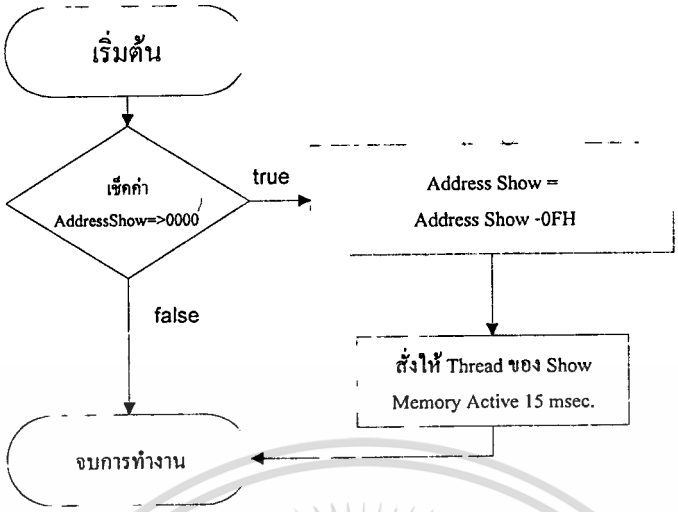
External Data Memory ตัวแปรที่ชี้ชื่อ Show_External_Mem ชนิด word

Program Data Memory ตัวแปรที่ชี้ชื่อ Show_Prog_Mem ชนิด word

โดยตัวแปรทั้ง 3 จะมีค่าแสดงแอดเดรสเริ่มต้นที่ 0H ส่วนการแสดง และสิ้นสุดที่ใน Internal Data Memory ที่ 0D0H ซึ่ง External Data Memory สิ้นสุดที่ 0FFDFH ส่วน External Data Memory สิ้นสุดที่ 0FFCFH ตามระยะห่างในการแสดงผลของแต่ละบรรทัด ดังนั้นเมื่อกดปุ่มให้เลื่อนบรรทัดในการแสดงหน่วยความจำ จึงมีลักษณะของการทำงานดังรูปที่ 3.18



รูปที่ 3.18 การทำงานเมื่อกดปุ่มเลื่อนหน่วยความจำขึ้น



รูปที่ 3.19 การทำงานเมื่อกดปุ่มเลื่อนหน่วยความจำลง

จากรูปที่ 3.18 และรูปที่ 3.19 แทน AddressShow ด้วยค่าของ Show_Internal_Mem, Show_External_Mem หรือ Show_Prog_Mem โดยมีลักษณะของโปรแกรมดังรูปที่ 3.20

```
(Internal Data Memory)
        procedure TShowMem.SpeedButton1Click(Sender: TObject);
        begin
                if Mcs_.Show_Internal_Data<($ff-$2f) then
                Mcs_.Show_Internal_Data:= Mcs_.Show_Internal_Data+$10;
                frameform.showrefresh;
        end;
        procedure TShowMem.SpeedButton2Click(Sender: TObject);
        begin
                if Mcs_.Show_Internal_Data>0 then
                Mcs_.Show_Internal_Data:=Mcs_.Show_Internal_Data-$10;
                frameform.showrefresh;
        end;
```

รูปที่ 3.20 โปรแกรมเลื่อนหน่วยความจำขึ้นลง

การนำค่าของหน่วยความจำไปแสดงผลนั้นจะแบ่งออกเป็น 2 ส่วนคือ ส่วนของ Program Data Memory กับส่วนของ Internal,External Data Memory โดยส่วนที่สองจะมีการทำงานเป็น เทรด ซึ่งจะกล่าวต่อไป ส่วนแรก Program Data Memory ใช้การแสดงผลโดยการนำค่าข้อมูลของโปรแกรมมาแสดงในแต่ละบรรทัด แล้วทำการส่งค่าออกไปเป็น String Line ของบรรทัดนั้นๆ มีลักษณะของโปรแกรมดังรูปที่ 3.21

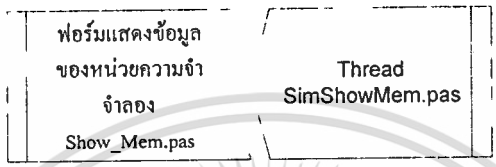
```
procedure showExp;
var
  i,l:word;
  st,space:string;
begin
with ShowMem do begin
i:=Mcs_.Show_External_Prog;
  ExProlist.items.Clear;
  for l:= 0 to $3 do
  begin
  space:=' ';
  st:= '['+inttohex(i,4)+']'+space
       +inttohex( Mcs_.External_prog_Mem[i],2)+space
       +inttohex(Mcs_.External_prog_Mem[i+1],2)+space
       +inttohex(Mcs_.External_prog_Mem[i+2],2)+space
       .....+inttohex(Mcs_.External_prog_Mem[i+15],2);
  i:= i+$10;
ExProlist.Items.Add(st);
   end;end;end;
```

รูปที่ 3.21 ฟังก์ชันแสดง External Data Memory

จากโปรแกรมในรูปที่ 3.21 จะสังเกตได้ว่าในหนึ่งบรรทัดที่ใช้ในการใส่ค่าไปใน TlistBox จะป้อนค่าของสตริงไปเป็นบรรทัดๆ โดยตรงทันที

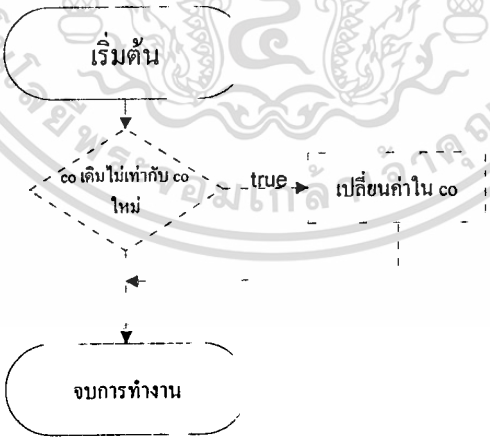
3.6.1 การทำเทรดของฟอร์มแสดงข้อมูลของหน่วยความจำจำลอง

ในการแสดงค่าของหน่วยความจำจำลองใช้เทรดในส่วนของการแสดงผลตรง Internal กับ External Data Memory เพราะส่วนนี้จะมีการเปลี่ยนแปลงของข้อมูลเมื่อมีการรันโปรแกรม เพื่อให้เกิดความเร็วในการทำงานและป้องกันการทำงานผิดพลาดจึงต้องมีการทำเทรดตรงส่วนนี้ โครงสร้างของเทรดในการแสดงผลหน่วยความจำจำลองมีโครงสร้างเป็นดังรูปที่ 3.22



รูปที่ 3.22 ส่วนประกอบของฟอร์มแสดงข้อมูลของหน่วยความจำจำลอง

ในเทรดชื่อ SimShowMem จะมีการเรียกใช้คอมโปเนนต์ที่สร้างโดยฟอร์มแสดงข้อมูลของหน่วยความจำจำลองผ่านทางคลาส (class)การแสดงผลในแต่ละตัวอักษรในแต่ละคอมโปเนนต์ในเทรดจะมีลักษณะการทำงานของโปรแกรมเป็นดังรูปที่ 3.23



รูปที่ 3.23 นำค่าจากตัวแปรไปแสดงผลในคอมโปเนนต์ผ่านทางเทรด

จากรูปที่ 3.23 ในส่วนของบล็อกที่เป็นเส้นประเราจะแทนให้มีจำนวนที่ต่อกันไปเท่ากับคอมโปเนนต์ ที่ใช้ในการแสดงผลซึ่งเขียนเป็นโปรแกรมได้ดังรูปที่ 3.24

```
procedure TVisualShowMem.showint;
        var
          i:word;
        begin
        with ShowMem do begin
           i:=Mcs_.Show_Internal_Data;
           ina.Caption:='['+inttohex(i,2)+']';
                if a0.Caption <> inttohex(Mcs_.In_Data_mem[i],2)
                then a0.Caption:=inttohex(Mcs_.In_Data_mem[i],2);
           ……..
        if xx.Caption <> inttohex(Mcs_.In_Data_mem[i+9],2)
                then a9.Caption:=inttohex(Mcs_.In_Data_mem[i+9],2);
end; end;
```

รูปที่ 3.24 ฟังก์ชันแสดงผล Internal Data Memory

ในส่วนของเทรดจะรวมเอา Internal กับ External ไว้ใน Procedure ตัวเดียวกันดังนั้นการรีเฟซในการแสดงผลจึงรีเฟสพร้อมกันทั้ง Internal และ External คุณลักษณะของเทรด SimShowMem.pas มีความสำคัญในการทำเทรดเป็น tpNormal ซึ่งเท่ากับ Primary Thread และมีส่วนแบ่งของเวลาในการทำงานอยู่ที่ 15 msec

3.7 การสร้างฟอร์มเซ็ตตำแหน่งแอดเดรสในหน่วยความจำจำลอง

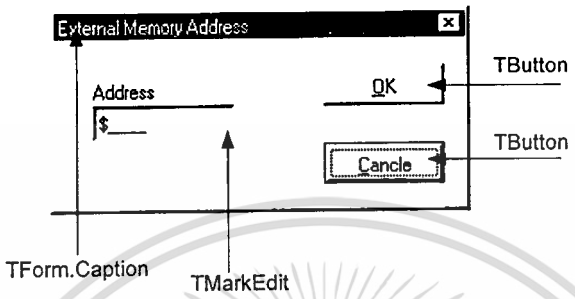
เมื่อผู้ใช้ต้องการดูแอดเดรสที่อยู่ห่างจากแอดเดรสที่แสดงอยู่มาก เช่น แสดงอยู่ที่ 0000 แต่ผู้ใช้ต้องการดูข้อมูลที่แอดเดรสที่ 0100 จะต้องกดปุ่มเลื่อนหน่วยความจำขึ้นหลายครั้งดังนั้นจึงต้องออกแบบให้สามารถกระโดดไปแสดงผลในหน่วยความจำที่แอดเดรสที่ต้องการได้ จากการออกแบบเริ่มจากกำหนดให้เรียกใช้ผ่านทางเมนูฟอร์มโดยเลือก Tool -> Address Show -> (Internal,External,Program Data Memory) และกำหนดให้มี คีย์ด่วน(hotkey)โดยกำหนดที่ TmainMemu ซึ่งกำหนดให้

CTRL+I คือ Internal Data Memory

CTRL+E คือ External Data Memory

CTRL+P คือ Program Data Memory

เมื่อกำหนดใน Tmainmenu ก็สร้างฟอร์มขึ้นมาใหม่โดยลักษณะของฟอร์มเป็นฟอร์มแบบ Normal กำหนดให้แสดงผลอยู่ที่กลางจอภาพลักษณะของฟอร์มเป็นดังรูปที่ 3.25



รูปที่ 3.25 ฟอร์มเปลี่ยนตำแหน่งแอดเดรสที่แสดงผล

การกำหนดชื่อของฟอร์มทำได้จากการเลือกของผู้ใช้ โดยที่ Tform.caption สามารถเลือกได้ 3 ชื่อคือ Internal Memory Address ,External Memory Address ,Program Memory Address และในแต่ละตัวเลือกจะกำหนดคุณสมบัติของ MarkEdit โดยเขียนโปรแกรมได้ดังรูปที่ 3.26

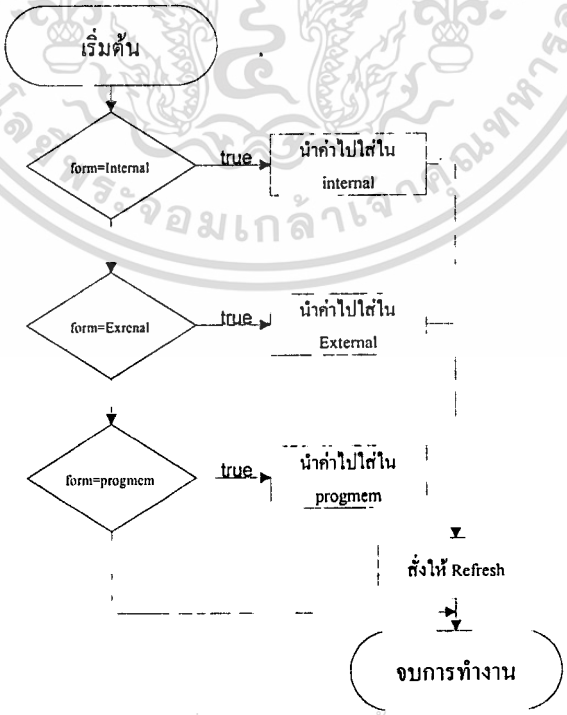
```
procedure TFrameForm.InternalDataMemory1Click(Sender: TObject);
begin
 Add_show.AddressShow.Caption:='Internal Memory Address';
 Add_Show.Addressshow.Address.EditMask:='$aa';
 Add_show.AddressShow.Tag:=0;
 Add_show.AddressShow.Show;
end;
procedure TFrameForm.ExternalDataMemory1Click(Sender: TObject);
begin
 Add_show.AddressShow.Caption:='External Memory Address';
 Add_Show.Addressshow.Address.EditMask:='$aaaa';
 Add_show.AddressShow.Tag:=0;
 Add_show.AddressShow.Show;
end;
```


รูปที่ 3.26 โปรแกรมเปลี่ยนตำแหน่งแอดเดรสที่แสดงผลใน Data Memory

```
procedure TFrameForm.ExternalProgramMemory1Click(Sender: TObject);
begin
  Add_show.AddressShow.Caption:='Program Memory Address';
  Add_Show.Addressshow.Address.EditMask:='$aaaa';
  Add_show.AddressShow.Tag:=0;
  Add_show.AddressShow.Show;
end;
```

รูปที่ 3.26 (ต่อ) โปรแกรมเปลี่ยนตำแหน่งแอดเดรสที่แสดงผลใน Data Memory

จากรูปที่ 3.26 ในส่วนของ procedure จะพบกับคำสั่ง Add_show.AddressShow.Tag คือคำสั่งที่ใช้กำหนดการเลือกอันดับที่ต้องการให้เอกทีฟเมื่อสร้างฟอร์มส่วนAdd_show.AddressShow. Show นั้นใช้ในการเรียกฟอร์มออกมาแสดง procedure ที่กล่าวมาทั้ง 3 จะต้องเรียกใช้ผ่านทางการกำหนด ClickEvent ในTManiMenu และเมื่อป้อนข้อมูลเสร็จแล้วกดปุ่ม Ok เพื่อนำข้อมูลที่ได้ไปใส่ในส่วนของ Show_Internal_Mem , Show_External_Mem หรือ Show_Prog_Mem ซึ่งคำสั่ง Ok จะทำการเลือกโดยเลือกจากชื่อของ Tform.caption โดยกระบวนการจะเป็นดังรูปที่ 3.27



รูปที่ 3.27 การทำงานเมื่อกดปุ่ม OK

จากรูปที่ 3.27 นำมาเขียนเป็นโปรแกรมได้ดังรูปที่ 3.28

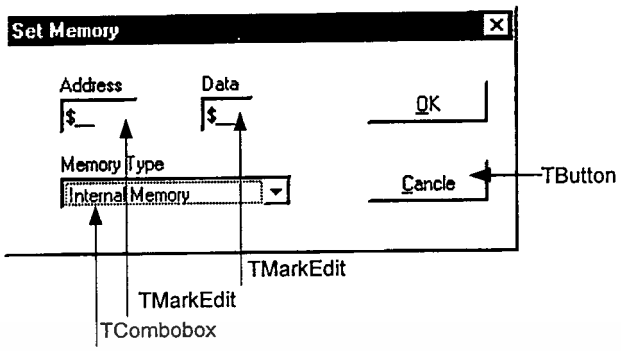
```
procedure TAddressShow.BitBtn1Click(Sender: TObject);
begin
  if Add_show.AddressShow.Caption =
  'Internal Memory Address' then //internal mem
     Mcs_.Show_Internal_Data :=Strtoint(address.Text);
  if Add_show.AddressShow.Caption =
  'External Memory Address' then//external mem
     Mcs_.Show_External_Mem :=Strtoint(address.Text);
  if Add_show.AddressShow.Caption =
  'Program Memory Address'then //external prog
     Mcs_.Show_External_Prog :=Strtoint(address.Text);
  FrameForm.Refreshall1.Click;
  AddressShow.Close; //close dialog
end;
```

รูปที่ 3.28 ฟังก์ชันเลือกชนิดของหน่วยความจำ

จากโปรแกรมในรูปที่ 3.28 จะสังเกตว่าฟังก์ชันที่สร้างขึ้นมาไม่ได้สร้างจาก Case เพราะว่าเพื่อให้ง่ายต่อการเขียนโปรแกรมอีกทั้งความยาวของคำสั่งไม่ได้มีความยาวมากจึงกำหนดโดยการใช้ คำสั่ง If ซ้อนกัน 3 ครั้ง

3.8 การสร้างฟอร์มเซ็ตค่าในหน่วยความจำ

เมื่อผู้ใช้ต้องการกำหนดค่าข้อมูลใดๆ ลงในหน่วยความจำไม่ว่าจะเป็น Internal ,External หรือ Program จึงมีความจำเป็นที่ต้องสร้างฟอร์มสำหรับนำข้อมูลไปใส่ในหน่วยความจำ กำหนดคีย์ด่วนคือ F2 หรือเลือกได้จาก Tool -> Set Memory โดยฟอร์มที่สร้างมีคุณสมบัติเป็น Normal กำหนดให้แสดงผลที่กลางจอภาพเป็นไดอะล็อกขนาดเล็ก มีลักษณะของฟอร์ม ดังรูปที่ 3.29



รูปที่ 3.29 คอมโปเนนต์ที่ใช้ในฟอร์มเซ็ทค่าในหน่วยความจำ

จากรูปที่ 3.29 ใช้คุณลักษณะของ Tcombobox กำหนดการเลือกชนิดของหน่วยความจำโดยผลลัพธ์ที่ได้จากการเลือกของ Tcombobox จะเป็นค่าตัวเลข 3จำนวนดังนี้

0 เท่ากับ Internal Memory

1 เท่ากับ External Memory

2 เท่ากับ Program Memory

เรานำค่าที่ได้จาก combobox ไปกำหนดคุณสมบัติของ Address ใน TmarkEdit โดยที่ Internal Memory มีความยาวที่ xxH ส่วน External,Program Memory มีความยาวอยู่ที่ xxxxH และยังนำค่าตัวเลขที่ได้จาก combobox ไปกำหนดชนิดของหน่วยความจำจำลองที่จะใส่ค่าไปในโปรแกรมโดยมีส่วนของการเลือกที่ Tcombobox เป็นดังโปรแกรมในรูปที่ 3.30

```
procedure TSMEM.MemTypeChange(Sender: TObject);
begin
        case Memtype.ItemIndex of
        0:Address.EditMask:='$aa'; //internal
        1:Address.EditMask:='$aaaa';// external
        2:Address.EditMask:='$aaaa';// prog
        end; //end case
end;
```


รูปที่ 3.30 ฟังก์ชันกำหนดขนาดความยาวของตัวอักษร

จากนั้นโปรแกรมในส่วนของ Click OK เป็นดังรูปที่ 3.31

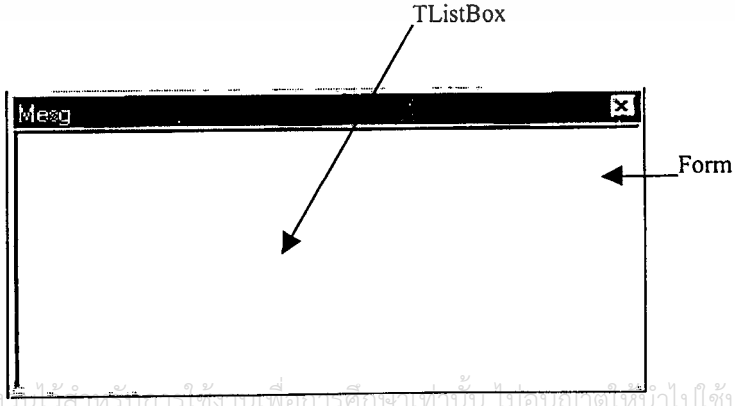
```
procedure TSMEM.BitBtn1Click(Sender: TObject);
begin
  case Memtype.ItemIndex of
        0:Mcs_.In_Data_Mem[Strtoint(address.Text)]:= StrToint(Data.Text);
        1: Mcs_.Ex_Data_Mem[Strtoint(address.Text)]:= StrToint(Data.Text);
        2: Mcs_.External_prog_Mem[Strtoint(address.Text)]:= StrToint(Data.Text);
end; //end case;
close; end;
```

รูปที่ 3.31 ฟังก์ชันส่งข้อมูลเข้าหน่วยความจำ

จากโปรแกรมในรูปที่ 3.31 จะนำค่าข้อมูลไปใส่ในคลาสของหน่วยความจำจำลองตามค่าของ index ใน combobox ที่เลือกไว้

3.9 การสร้างฟอร์มเม็สเส็จ

ฟอร์มเม็สเส็จเป็นฟอร์มที่ใช้ในการแสดงการทำงานที่ผิดพลาดของโปรแกรมซึ่งใน เวอร์ชั่น 0.2 นั้นฟอร์มเม็สเส็จนั้นไม่มีบทบาทมากนัก หากแต่เป็นอุปกรณ์ที่ผู้นำโปรแกรมไปพัฒนาต่อไปสามารถใช้ในการแสดงผลข้อความเกี่ยวกับการทำงานของระบบเพิ่มเติมจากเดิมได้ ดังนั้นฟอร์มเม็สเส็จจึงมีรายละเอียดในการสร้างค่อนข้างง่าย ดังแสดงในรูปที่ 3.32

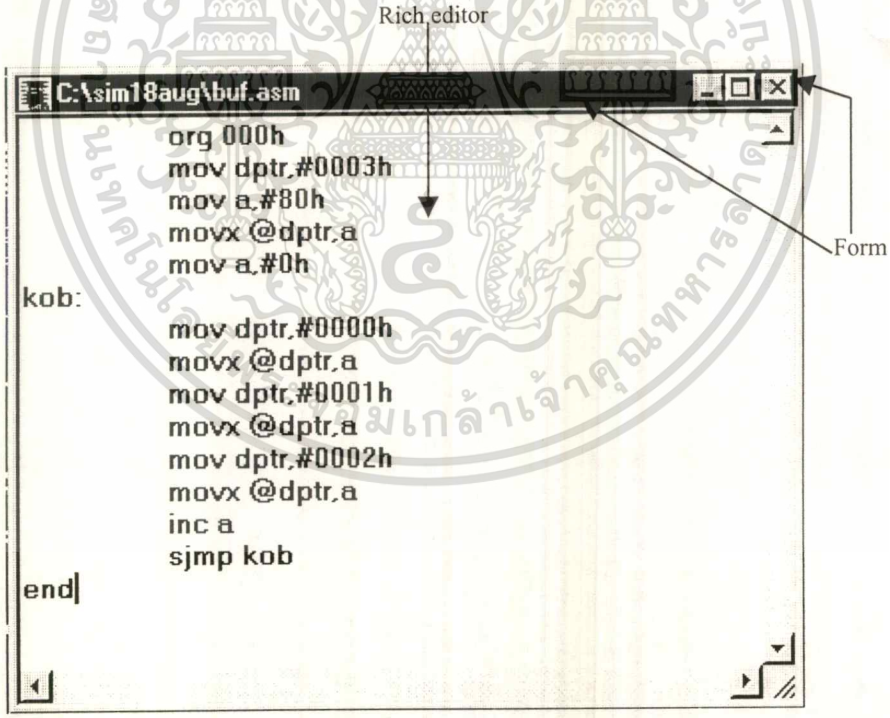


รูปที่ 3.32 ฟอร์มเม็สเส็จ

ส่วนประกอบของฟอร์มเม็สเส็จประกอบด้วย Tlistbox กับ Tform โดยการส่งผ่านค่าเข้าไปใน Tlistbox ส่งผ่านตามคุณสมบัติของ Tlistbox ซึ่งสามารถส่งมาจากส่วนใดๆ ของโปรแกรมก็ได้แต่ในส่วนของ Frameform โดยส่วนใหญ่จะส่งผ่านจาก procedure text_mesg(text:สตริง) โดยที่ procedure ตัวนี้จะรับข้อมูลเป็นสตริงและนำข้อมูลตัวนี้ส่งผ่านไป Tlistbox ในฟอร์มเม็สเส็จ

3.10 การสร้างฟอร์มอิดิเตอร์และการสั่งพิมพ์

โครงสร้างของฟอร์มอิดิเตอร์เหมือนกับฟอร์มเม็สเส็จ ในส่วนที่ใช้ในการพิมพ์ข้อความจะใช้ Trichedit แทนที่จะใช้ Tmemo สาเหตุเพราะว่า Tmemo ไม่สามารถแสดงตัวอักษรในรูปแบบฟร้อนอื่นๆได้ แต่ Trichedit สามารถแสดงตัวอักษรเป็นรูปแบบฟร้อนอื่นๆได้ แต่ในการทำงานของโปรแกรมได้กำหนดให้ Trichedit มีคุณสมบัติในการพิมพ์โดยใช้ฟร้อนมาตราฐาน เพื่อให้ข้อมูลที่บันทึกเป็นรูปแบบเท็กไฟล์ คือ ไม่มีในส่วนของการบอกว่าบรรทัดใดเป็นฟร็อนชนิดใด และอีกสาเหตุที่ใช้ Trichedit เพราะสามารถสั่งพิมพ์ได้ง่าย กว่า Tmemo มาก ดังแสดงในรูปที่ 3.33



รูปที่ 3.33 ฟอร์มอิดิเตอร์

การสั่งพิมพ์จะใช้คอมโปเนนต์ PrinterSetupDialog กับ PrintDialog โดยนำคอมโปเนนต์ทั้งสองมาวางที่ Frameform จากนั้นกำหนดในส่วนของ Memu bar จากคอมโปเนนต์ Mainmenu ให้สร้าง selectitem ชื่อ printer setup กับ print เมื่อเสร็จ พิมพ์คำสั่งในส่วนของ procedure printer setup item พิมพ์คำสั่ง PrinterSetupDialog.Execute โดยเมื่อผู้ใช้เลือก item จะไปติดต่อกับส่วนของการควบคุมเครื่องพิมพ์ในส่วนของ windows ซึ่งก็จะกำหนดคุณสมบัติของเครื่องพิมพ์ ต่อมาในส่วนของการสั่งพิมพ์จากที่สร้าง item print พิมพ์คำสั่งในส่วนของ procedure print item คำสั่งภายใน item เป็นดังรูปที่ 3.34

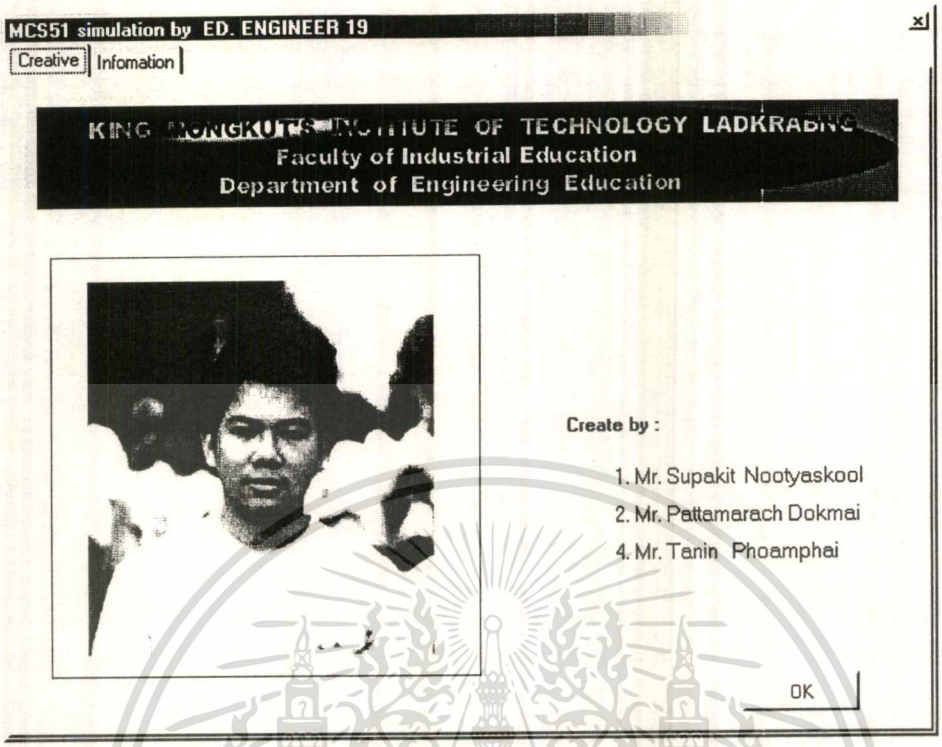
```
procedure TFrameForm.Print1Click(Sender: TObject);
begin
editor.text_edit.Lines.SaveToFile('printbuf.112');
sleep(3000);
if printdialog1.Execute then
  editor.text_edit.Print('printbuf.112');
end;
```

รูปที่ 3.34 ฟังก์ชันในการสั่งพิมพ์

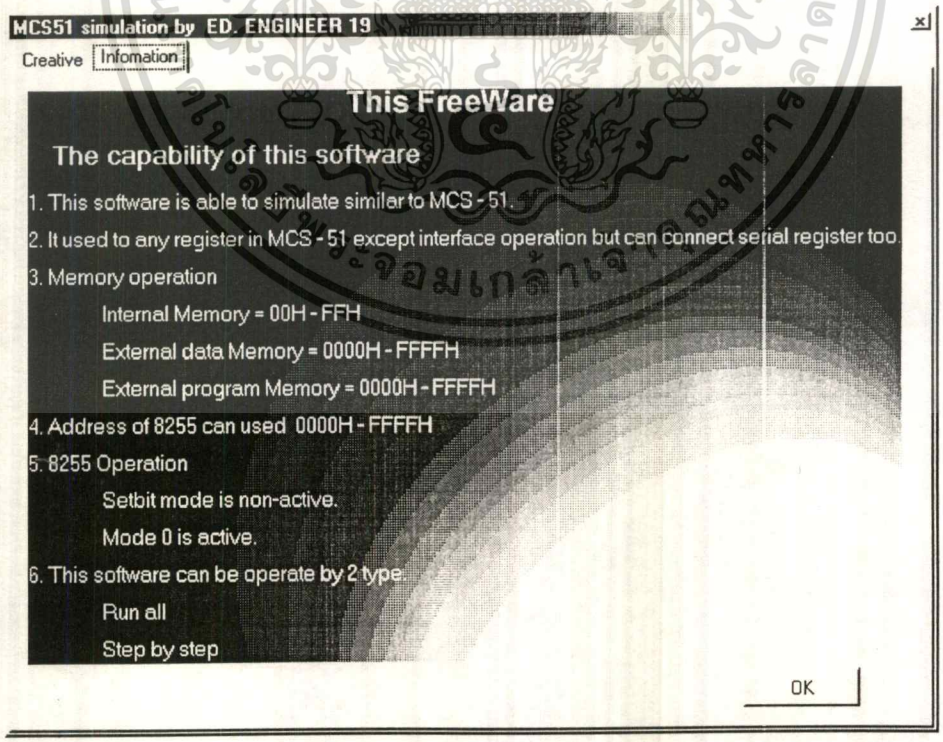
สังเกตได้ว่าจากคำสั่งภายใน procedure จะเก็บข้อมูลไว้ในไฟล์ชื่อ printbuf.112 เพราะว่าการสั่งพิมพ์จาก Trichedit นั้นจะเรียกจากเอกสารที่เป็นไฟล์เท่านั้น ซึ่งสังเกตได้จากบรรทัด editor.text_edit.print('Filename'); ดังนั้นจึงทำให้ต้องมีการเก็บข้อมูลก่อน และในช่วงของการพิมพ์จะต้องเรียก printdialog ด้วย

3.11 การสร้างฟอร์มแสดงรายละเอียดเกี่ยวกับโปรแกรม

ฟอร์มแสดงรายละเอียดของโปรแกรมจะกล่าวถึงผู้จัดทำและรายละเอียดเกี่ยวกับโปรแกรม ซึ่งมีรูปแบบฟอร์มและการวางคอมโปเนนต์ดังรูปที่ 3.35 และรูปที่ 3.36



รูปที่ 3.35 ฟอร์มแสดงรายละเอียดเกี่ยวกับโปรแกรมส่วนที่ 1



รูปที่ 3.36 ฟอร์มแสดงรายละเอียดเกี่ยวกับโปรแกรมส่วนที่ 2

จากรูปที่ 3.35 สังเกตได้ว่าในส่วนของชื่อของผู้จัดทำ ตัวอักษรจะสามารถทำ highlight ได้ โดยการกำหนดในส่วนของ event ที่ Onclick เรียก procedure showpic1

Mousemove เรียก procedure mousemove2

โดยที่ procedure ที่เรียกจากทั้งสองส่วนนั้นขึ้นอยู่กับ hightlight ที่เราเลือกด้วย ตัวอย่างข้อมูลที่มีอยู่ใน procedure ทั้งสองตัวและการกำหนดค่าภายในเป็นดังรูปที่ 3.37

```
procedure Tabout_form.showpic1(Sender: TObject);
begin
    hut1.Visible:=false;
    logo1.Visible:=false;
    kob1.Visible:=true;
end;

procedure Tabout_form.mousemove2;
begin
show2.Font.Color:= clyellow;
end;
```

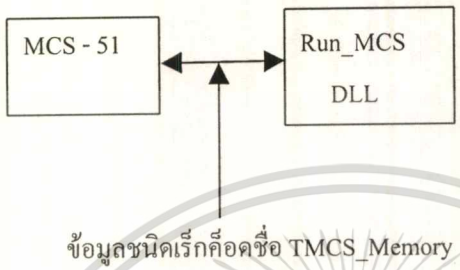
รูปที่ 3.37 ฟังก์ชันกำหนดสถานะการแสดงรูป

ในส่วนของ procedure showpic1 นั้นเมื่อผู้ใช้คลิกที่ส่วนของรูปภาพ จะทำให้มีการแสดงรูปภาพที่กำหนดให้แสดง ซึ่งทำให้รูปภาพที่แสดงใน image เกิดการเปลี่ยนแปลง ในส่วนของ procedure mousemove2 กำหนดให้แสดง highlight ขึ้นเมื่อเลื่อนเมาส์ผ่านคอมโปเนนต์ตัวดังกล่าว กำหนด hightlight เป็นสีเหลือง

3.12 การสร้างชุดคำสั่งของ MCS-51

เพื่อให้ง่ายต่อการเขียนโปรแกรมในส่วนของชุดคำสั่งของ MCS-51 จึงออกแบบให้เป็น DLL ซึ่งเป็นไลบารีแยกออกมาจากตัวโปรแกรมหลัก เพื่อให้สะดวกในการเขียนชุดคำสั่งของ MCS-51 การส่งค่าให้ไลบารี MCS-51 จะใช้ตัวแปรแบบเร็คคอร์ด ซึ่งภายในประกอบด้วย Array Internal Data Memory , Array External Data Memory , Array External Program Memory ใช้ใน

การส่งค่าทั้งหมดของโปรแกรมในการทำงาน ในการทำงานจะเรียกใช้ไลบารีชุดคำสั่ง MCS-51 เป็นฟังก์ชันตัวหนึ่งและจะเรียกใช้ฟังก์ชันตัวนี้ทุกๆ คำสั่งการทำงาน MCS-51 โดยที่ฟังก์ชันจะอ่านข้อมูลใน External Program Memory ซึ่งใช้ตัวแปร Address_Point_EXProg ในการชี้ว่าการทำงานอยู่ในแอดเดรสใด ดังรูปที่ 3.38



รูปที่ 3.38 การเชื่อมต่อระหว่าง Mcs51sim.exe กับ Run_Mcs.dll

3.12.1 ฟังก์ชันที่ใช้ใน Run_Mcs.dll ประกอบด้วย

1. function CY_Set(MCS_:TMcs_Memory):TMcs_Memory;
2. function CY_Clear(MCS_:TMcs_Memory):TMcs_Memory;
3. function AC_Set(MCS_:TMcs_Memory):TMcs_Memory;
4. function AC_Clear(MCS_:TMcs_Memory):TMcs_Memory;
5. function OV_Set(MCS_:TMcs_Memory):TMcs_Memory;
6. function OV_Clear(MCS_:TMcs_Memory):TMcs_Memory;
7. function P_Set(MCS_:TMcs_Memory):TMcs_Memory;
8. function P_Clear(MCS_:TMcs_Memory):TMcs_Memory;
9. function PSW_check(MCS_:TMcs_Memory;data:byte):TMcs_Memory;
10. function AJMPfunction(MCS_:TMcs_Memory;Pages:byte):TMcs_Memory;
11. function ACALLfunction(MCS_:TMcs_Memory;Pages:byte):TMcs_Memory;
12. function ADDC_A_Rn(MCS_:TMcs_Memory;Rn:byte):TMcs_Memory;
13. function SUBB_A_Rn(MCS_:TMcs_Memory;Rn:byte):TMcs_Memory;
14. function ORL_A_Rn(MCS_:TMcs_Memory;Rn:byte):TMcs_Memory;
15. function ANL_A_Rn(MCS_:TMcs_Memory;Rn:byte):TMcs_Memory;
16. function XRL_A_Rn(MCS_:TMcs_Memory;Rn:byte):TMcs_Memory;
17. function XCH_A_Rn(MCS_:TMcs_Memory;Rn:byte):TMcs_Memory;

18. function DJNZ_Rn(MCS_:TMcs_Memory;Rn:byte):TMcs_Memory;
19. function CJNE_Rn(Mcs_:TMcs_Memory;Rn:byte):TMCS_Memory;
20. function Bitaddress(bitadd:byte):TBitadd;

3.12.2 รายละเอียดการทำงานของฟังก์ชันต่างๆ มีดังนี้

1. function CY_Set(MCS_:TMcs_Memory):TMcs_Memory;

หน้าที่

เซ็ตให้บิต cy มีค่าเป็นหนึ่ง

การทำงาน

เมื่อเรียกใช้ฟังก์ชันจะนำค่าเดิมใน PSW มากำหนดให้บิต cy มีค่าเป็นหนึ่งทันที บิต cy อยู่ที่บิตที่ 7 ของ psw จากนั้นส่งค่าที่ได้ให้กับ หน่วยความจำ MCS-51 จำลอง ฟังก์ชันเป็นดังรูปที่ 3.39

```
function CY_Set(MCS_:TMcs_Memory):TMcs_Memory;
var i,j:byte;
begin
  i:=Mcs_.In_Data_Mem[_PSW];
  asm
    push ax
    mov al,i
    or  al,$80
    mov j,al
    pop ax
  end;
  Mcs_.In_Data_Mem[_PSW]:=j;
  CY_Set:=MCS_;
end;
```

รูปที่ 3.39 ฟังก์ชันเซ็ต carry แฟล็ก

2. function CY_Clear(MCS_:TMcs_Memory):TMcs_Memory;

หน้าที่

เคลียร์ให้บิต cy มีค่าเป็นศูนย์

การทำงาน

เมื่อเรียกใช้ฟังก์ชันจะนำค่าเดิมใน PSW มากำหนดให้บิต cy มีค่าเป็นศูนย์ทันที บิต cy อยู่ที่บิตที่ 7 ของ psw จากนั้นส่งค่าที่ได้ให้กับ หน่วยความจำ MCS-51 จำลอง ฟังก์ชันเป็นดังรูปที่ 3.40

```
function CY_Clear(MCS_:TMcs_Memory):TMcs_Memory;
var i,j:byte;
begin
  i:=Mcs_.In_Data_Mem[_PSW];
  asm
    push ax
    mov al,i
    and  al,$7f
    mov j,al
    pop ax
  end;
  Mcs_.In_Data_Mem[_PSW]:=j;
  CY_Clear:=MCS_;
end;
```

รูปที่ 3.40 ฟังก์ชัน clear carry แฟล็ก

3. function AC_Set(MCS_:TMcs_Memory):TMcs_Memory;

หน้าที่

เซ็ต ให้บิต ac มีค่าเป็นหนึ่ง

การทำงาน

เมื่อเรียกใช้ฟังก์ชันจะนำค่าเดิมใน PSW มากำหนดให้บิต ac มีค่าเป็นหนึ่งทันที บิต ac อยู่ที่บิตที่ 6 ของ psw จากนั้นส่งค่าที่ได้ให้กับ หน่วยความจำ MCS-51 จำลอง ฟังก์ชันเป็นดังรูปที่ 3.41

```
function AC_Set(MCS_:TMcs_Memory):TMcs_Memory;
var i,j:byte;
begin
   i:=Mcs_.In_Data_Mem[_PSW];
   asm
     push ax
     mov al,i
     or  al,$40
     mov j,al
     pop ax
    end;
    Mcs_.In_Data_Mem[_PSW]:=j;
    AC_Set:=MCS_;
end;
```

รูปที่ 3.41 ฟังก์ชันเซ็ต AC แฟล็ก

4. function AC_Clear(MCS_:TMcs_Memory):TMcs_Memory;

หน้าที่

เคลียร์ ให้บิต ac มีค่าเป็นศูนย์

การทำงาน

เมื่อเรียกใช้ฟังก์ชันจะนำค่าเดิมใน PSW มากำหนดให้บิต ac มีค่าเป็นศูนย์ทันที บิต ac อยู่ที่บิตที่ 6 ของ psw จากนั้นส่งค่าที่ได้ให้กับ หน่วยความจำ MCS-51 จำลอง ฟังก์ชันเป็นดังรูปที่ 3.42

```
function AC_Clear(MCS_:TMcs_Memory):TMcs_Memory;
var i,j:byte;
begin
   i:=Mcs_.In_Data_Mem[_PSW];
   asm
     push ax
     mov al,i
     and  al,$bf
     mov j,al
     pop ax
   end;
   Mcs_.In_Data_Mem[_PSW]:=j;
   AC_Clear:=MCS_;
end;
```

รูปที่ 3.42 ฟังก์ชัน clear AC แฟล็ก

5. function OV_Set(MCS_:TMcs_Memory):TMcs_Memory;

<u>หน้าที่</u>

เซ็ต ให้บิต ov มีค่าเป็นหนึ่ง

<u>การทำงาน</u>

เมื่อเรียกใช้ฟังก์ชันจะนำค่าเดิมใน PSW มากำหนดให้บิต ov มีค่าเป็นหนึ่งทันที บิต ov อยู่ที่บิตที่ 2 ของ psw จากนั้นส่งค่าที่ได้ให้กับ หน่วยความจำ MCS-51 จำลอง ฟังก์ชันเป็นดังรูปที่ 3.43

6. function OV_Clear(MCS_:TMcs_Memory):TMcs_Memory;

<u>หน้าที่</u>

เคลียร์ ให้บิต ov มีค่าเป็นศูนย์

การทำงาน

เมื่อเรียกใช้ฟังก์ชันจะนำค่าเดิมใน PSW มากำหนดให้บิต ov มีค่าเป็นศูนย์ทันที บิต ov อยู่ที่บิตที่ 2 ของ psw จากนั้นส่งค่าที่ได้ให้กับ หน่วยความจำ MCS-51 จำลอง ฟังก์ชันเป็นดังรูปที่ 3.44

```
function OV_Set(MCS_:TMcs_Memory):TMcs_Memory;
var i,j:byte;
begin
   i:=Mcs_.In_Data_Mem[_PSW];
   asm
     push ax
     mov al,i
     or  al,$04
     mov j,al
     pop ax
   end;
   Mcs_.In_Data_Mem[_PSW]:=j;
   OV_Set:=MCS_;
end;
```

รูปที่ 3.43 ฟังก์ชันเซ็ต OV แฟล็ก

7. function P_Set(MCS_:TMcs_Memory):TMcs_Memory;

หน้าที่

เซ็ต ให้บิต p มีค่าเป็นหนึ่ง

การทำงาน

เมื่อเรียกใช้ฟังก์ชันจะนำค่าเดิมใน PSW มากำหนดให้บิต p มีค่าเป็นหนึ่งทันที บิต p อยู่ที่บิตที่ 0 ของ psw จากนั้นส่งค่าที่ได้ให้กับ หน่วยความจำ MCS-51 จำลอง ฟังก์ชันเป็นดังรูปที่ 3.45

```
function OV_Clear(MCS_:TMcs_Memory):TMcs_Memory;
var i,j:byte;
begin
   i:=Mcs_.In_Data_Mem[_PSW];
   asm
     push ax
     mov al,i
     and  al,$fb
     mov j,al
     pop ax
   end;
   Mcs_.In_Data_Mem[_PSW]:=j;
   OV_Clear:=MCS_;
end;
```

รูปที่ 3.44 ฟังก์ชัน clear OV แฟล็ก

```
function P_Set(MCS_:TMcs_Memory):TMcs_Memory;
var i,j:byte;
begin
   i:=Mcs_.In_Data_Mem[_PSW];
   asm
     push ax
     mov al,i
     or  al,$01
     mov j,al
     pop ax
   end;
   Mcs_.In_Data_Mem[_PSW]:=j;
   P_Set:=MCS_;
end;
```

รูปที่ 3.45 ฟังก์ชันเซ็ตพาริตี้แฟล็ก

8. function P_Clear(MCS_:TMcs_Memory):TMcs_Memory;

หน้าที่

เคลียร์ ให้บิต p มีค่าเป็นศูนย์

การทำงาน

เมื่อเรียกใช้ฟังก์ชันจะนำค่าเดิมใน PSW มากำหนดให้บิต p มีค่าเป็นศูนย์ทันที บิต p อยู่ที่บิตที่ 0 ของ psw จากนั้นส่งค่าที่ได้ให้กับ หน่วยความจำ MCS-51 จำลอง ฟังก์ชันเป็นดังรูปที่ 3.46

```
function P_Clear(MCS_:TMcs_Memory):TMcs_Memory;
var i,j:byte;
begin
   i:=Mcs_.In_Data_Mem[_PSW];
   asm
     push ax
     mov al,i
     and  al,$fe
     mov j,al
     pop ax
   end;
   Mcs_.In_Data_Mem[_PSW]:=j;
   P_Clear:=MCS_;
end;
```

รูปที่ 3.46 ฟังก์ชันเคลียร์พาริตี้แฟล็ก

9. function PSW_check(MCS_:TMcs_Memory;data:byte):TMcs_Memory;

หน้าที่

เรียกใช้จากฟังก์ชันในการ เซ็ต และ เคลียร์ บิต cy,ac,p เพื่อลดขั้นตอนในการกำหนดสถานะของบิต และใช้ในชุดคำสั่งที่กระทำกันและไม่เกิด overflow แต่ในการทำงานของโปรแกรมได้สร้างการคำนวณเมื่อเกิด overflow ตามรูปแบบของคำสั่งนั้นๆเฉพาะตัว

การทำงาน

เมื่อเรียกใช้ฟังก์ชันค่าข้อมูล (data) ใช้กำหนดสถานะว่าบิต cy,ac,p ให้เป็น "0","1" โดยที่ cy อยู่ที่ตำแหน่งบิตที่ 7, ac อยู่ในตำแหน่งบิตที่ 6 และ p อยู่ในตำแหน่งบิตที่ 0 ฟังก์ชันเป็นดังรูปที่ 3.47

```
function PSW_check(MCS_:TMcs_Memory;data:byte):TMcs_Memory;
var cy,ac,p:byte;
begin
   asm
   pusha
   pushf   //carry check zone
   mov al,data
   and al,$01
  mov cy,al
   //parity check zone
   mov al,data
   and al,$04
   shr al,2
   mov p,al
   //auxcilary check zone
   mov al,data
   and al,$10
   shr al,4
   mov ac,al
   popf
   popa
   end;
   if cy=0 then MCS_:=CY_clear(MCS_) else MCS_:=CY_set(MCS_);
   if ac=0 then MCS_:=AC_clear(MCS_) else MCS_:=AC_set(MCS_);
   if p=0 then MCS_:=P_set(MCS_) else MCS_:=P_clear(MCS_);
   PSW_check:=MCS_;
end;
```


รูปที่ 3.47 ฟังก์ชันตรวจสอบรีจิสเตอร์ PSW

จากรูปภาพที่ 3.47 สังเกตได้ว่าพาริตี้ของฟังก์ชันจะตรวจสอบในบิตที่ 2, auxcilary จะตรวจสอบในบิตที่ 4 และ carry ตรวจสอบในบิตที่ 0 เพราะว่าในการคำนวณได้ออกแบบให้ใช้ชุดคำสั่งของ x86 ซึ่งจะมีคำสั่งที่ใช้ในการตรวจสอบแฟล็กคือคำสั่ง LAHF และการเรียงจึงใช้การเรียงตามบิตของ LAHF และผลลัพธ์ที่ได้ออกมาทาง PSW มีค่าถูกต้องเหมือนเดิม

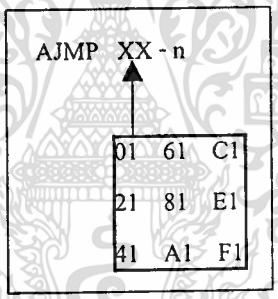
10. function AJMPfunction(MCS_:TMcs_Memory;Pages:byte):TMcs_Memory;

หน้าที่

เปลี่ยนค่าของ programcounter (pc) ให้ตรงตามแอดเดรสและเพจที่ต้องการ

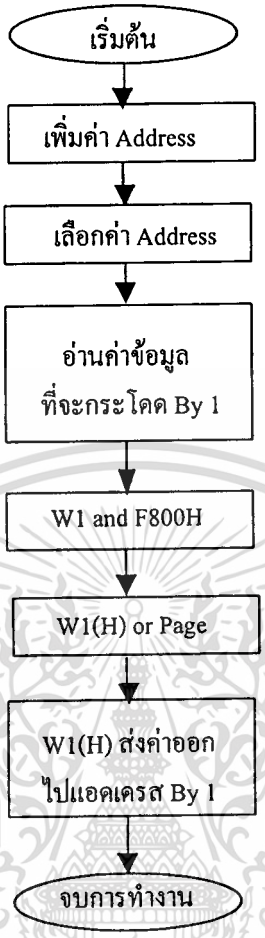
การทำงาน

ฟังก์ชันตัวนี้จะรับค่าเพจที่จะต้อง jump ไปโดยชุดคำสั่งของ MCS-51 ได้แบ่งแยกเพจเอาไว้ ซึ่งทำให้ชุดคำสั่งในการกระทำ Ajmp มี 8 ชุดคำสั่ง ซึ่งในฟังก์ชันได้แบ่งออกเป็นเพจ 0 ถึงเพจ 7 ดังรูปที่ 3.48



รูปที่ 3.48 คำสั่ง AJMP

ในฟังก์ชันการทำงานรับตัวแปร เพิ่มมาอีกหนึ่งตัวคือเพจซึ่งใช้ในการระบุเพจที่จะกระโดดไปตามที่ได้กล่าวไปแล้ว การทำงานของฟังก์ชันเป็นดังรูปที่ 3.49 (flowchart) คือ เพิ่มค่าของ pc เพื่ออ่านออปโค้ด (opcode) ตัวต่อไปได้ นำค่าแอดเดรสที่เพิ่มนั้นไปเก็บในตัวแปร w1 ซึ่งจะต้องมีขนาด 16 บิต เพื่อใช้ในการแยกระหว่าง 8 บิตบนกับ 8 บิตล่าง ต่อมาอ่านค่าข้อมูลที่ pc ใน External Program Memory



รูปที่ 3.49 กระบวนการทำงานของ AJMP

นำค่าที่ได้ไปเก็บในตัวแปร by1 ในตัวแปร by1 จะเป็นค่า n ที่ใช้ในการชี้ 8 บิตล่างกระโดดไปในตำแหน่งใด จากนั้นนำค่าของ w1 มาตัดเอาส่วนของ 8 บิตล่างออกโดยการนำค่า 0F8000H มา and แล้วนำค่าเพจไปใส่ใน w1 คือค่าบิตที่ 8 ถึงบิตที่ 11 และนำค่าจากตัวแปร by1 ไปใส่ในส่วนของ 8 บิตล่างของ w1 นำค่าที่ได้ใส่ใน pc เพื่อกระโดดไปในตำแหน่ง แอดเดรสที่ต้องการ ดังแสดงโปรแกรมในรูปที่ 3.50

```
function AJMPfunction(MCS_:TMcs_Memory;Pages:byte):TMcs_Memory;
var
  by1:byte;
  w1:word;
begin
   Mcs_.Address_point_ExProg:=Mcs_.Address_point_ExProg+1;
   w1:=Mcs_.Address_point_ExProg;
   by1:=Mcs_.External_prog_mem[Mcs_.Address_point_ExProg];
   asm
    push ax
    push bx
    mov bl,by1
    mov ax,w1
    and ax,$0f800  //delete data for define
    or  ah,Pages   //select page for jump
    mov al,bl
    mov w1,ax
    pop bx
    pop ax
   end;
   Mcs_.Address_point_ExProg:=w1;
   AJMPfunction:=MCS_;
end;
```

รูปที่ 3.50 ฟังก์ชัน AJMP

11.function ACALLfunction(MCS_:TMcs_Memory;Pages:byte):TMcs_Memory;

<u>หน้าที่</u>

เปลี่ยนค่า pc ให้อยู่ในตำแหน่งที่โปรแกรมย่อยตัวนั้นทำงานและนำค่าเก็บลงในสแต็ค (stack : sp)

การทำงาน

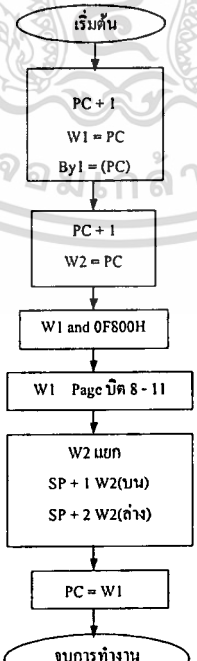
Acall เป็นคำสั่งที่แบ่งตำแหน่งของหน่วยความจำที่กระโดดไปเป็นเพจได้ 8 เพจ ดังนั้นในชุดคำสั่งจึงมี 8 ชุดคำสั่งแต่ลักษณะโครงสร้างการกระโดดเหมือนกันมีความแตกต่างตรงเพจของหน่วยความจำที่กระโดดไป ดังรูปที่ 3.51

ACALL

Code	XX	n
	11, 31, 51, 71, 91, B1, D1, F1	00H-FFH

รูปที่ 3.51 คำสั่ง ACALL

ดังนั้นในการออกแบบฟังก์ชัน (ดูรูปที่ 3.52) จึงสร้างฟังก์ชันขึ้นมาเพียงตัวเดียวแล้วให้มีความสามารถกระโดดไปในเพจต่างๆ ณ ตำแหน่งของหน่วยความจำต่างๆได้ โดยการทำงานของคำสั่งจะเริ่มจาก เพิ่มค่า pc แล้วนำค่า pc ที่ชี้ตำแหน่งของ External Program Memory อ่านค่าของตำแหน่งดังกล่าวเก็บลงในตัวแปร by1 ซึ่งเป็นตัวแปรขนาด 8 บิต จากนั้นเก็บค่าของตำแหน่ง pc ในตัวแปร w1 ซึ่งมีขนาด 16 บิต เพิ่มค่า pc อีกครั้งแล้วเก็บค่าของ pc ในตัวแปร w2 ขนาด 16 บิต



รูปที่ 3.52 กระบวนการทำงานของ ACALL

จากนั้นนำค่าตัวแปร w1 มาตัดเอาส่วนของ 8 บิตล่าง กับส่วนบิตที่ 8 ถึงบิตที่ 11 ออก โดยการนำค่า w1 and 0F800H นำค่าของตัวแปรเพจใส่ในตำแหน่งบิตที่ 8 ถึง บิตที่ 11 จากนั้นทำการแยกตัวแปร w2 ซึ่งมีขนาด 16 บิตแยกออกได้ 8 บิตบน กับ 8 บิตล่าง นำค่า 8 บิตบนเก็บลงใน สแต็คชั้นที่หนึ่ง จากนั้นนำ 8 บิตล่างเก็บตามลงในสแต็คนำค่า w1 ส่งให้กับ pc เพื่อกระโดดไปทำงานในตำแหน่งที่ต้องการ ฟังก์ชันจึงเป็นดังรูปที่ 3.53

```
function ACALLfunction(MCS_:TMcs_Memory;Pages:byte):TMcs_Memory;
var
  by1,by2:byte;
  w1,w2:word;
begin
   Mcs_.Address_point_ExProg:=Mcs_.Address_point_ExProg+1;
   w1:=Mcs_.Address_point_ExProg;
   by1:=Mcs_.External_prog_mem[Mcs_.Address_point_ExProg];//read page jump to
   Mcs_.Address_point_ExProg:=Mcs_.Address_point_ExProg+1;
   w2:=Mcs_.Address_point_ExProg; //address for save to stack
   asm
    push ax
    push bx
    mov bl,by1
    mov ax,w1
    and ax,S0f800  //delete data for define
    or  ah,Pages   //select page for jump
    mov al,bl
    mov w1,ax
    //save to stack
    mov ax,w2
    mov by1,ah
    mov by2,al
    pop bx
    pop ax
   end;
```


รูปที่ 3.53 ฟังก์ชัน ACALL

```
    Mcs_.In_Data_Mem[_SP]:=Mcs_.In_Data_Mem[_SP]+1; // stack +1
    Mcs_.In_Data_Mem[Mcs_.In_Data_Mem[_SP]]:=by2;   // save to stack memory low
    Mcs_.In_Data_Mem[_SP]:=Mcs_.In_Data_Mem[_SP]+1; // stack +1
    Mcs_.In_Data_Mem[Mcs_.In_Data_Mem[_SP]]:=by1;   // save to stack memory high
    Mcs_.Address_point_ExProg:=w1; //address jump to
    ACALLfunction:=MCS_;
end;
```

รูปที่ 3.53 (ต่อ) ฟังก์ชัน ACALL

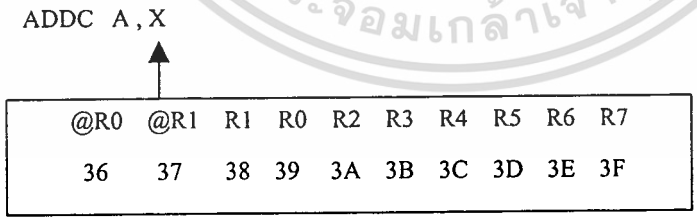
12. function ADDC_A_Rn(MCS_:TMcs_Memory;Rn:byte):TMcs_Memory;

หน้าที่

ทำการบวกค่าระหว่างรีจิสเตอร์ A กับ Rn หรือ (Rn) และคิดคำนวณค่าแฟล็ก

การทำงาน

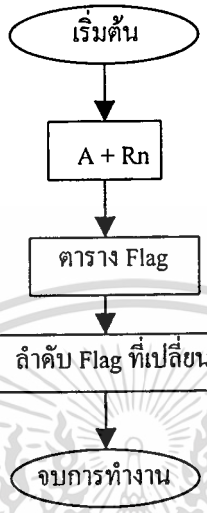
คำสั่ง ADDC เป็นคำสั่งที่ทำการบวกค่าระหว่าง A กับ Rn ซึ่งหากเป็นการเขียนโปรแกรมให้ตัวแปร สองตัวทำการบวกค่ากัน ก็สามารถกำหนดให้ตัวแปรบวกค่ากันได้ทันที แต่ฟังก์ชันนี้จำเป็นที่ต้องการค่าแฟล็ก ที่ได้จากการบวกค่าของตัวแปร ดังนั้นผู้ออกแบบโปรแกรมจึงอาศัยการบวกผ่านทางภาษาแอสเซมบลี้ ซึ่งสามารถบอกค่าแฟล็กได้ โดยการใช้คำสั่ง LAHF ดังรูปที่ 3.54



รูปที่ 3.54 คำสั่ง ADDC A,Rn

ในคำสั่ง ADDC ของ MCS-51 มีคำสั่งทั้งหมด 11 ตัวแต่ในฟังก์ชันตัวนี้สามารถใช้กับคำสั่ง @R0,@R1 และคำสั่ง R0 ถึง R7 ซึ่งทำได้ทั้งหมด 10 คำสั่ง การทำงานเมื่อเรียกใช้คำสั่ง ADDC

A,Rn ส่วนฟังก์ชัน ADD A,Rn จะเหมือนกับ ADDC A,Rn แตกต่างกันเพียงการนำ carry มาคิดเท่านั้น ดังนั้นวิธีการทำงานของคำสั่งแสดงได้ดังรูปที่ 3.55 และเขียนโปรแกรมได้ดังรูปที่ 3.36



รูปที่ 3.55 การทำงานของคำสั่ง ADDC A,Rn

```
function ADDC_A_Rn(MCS_:TMcs_Memory;Rn:byte):TMcs_Memory;
var by2,by1,by3,cy,ac,p:byte;
  ch1:char;
begin
   Mcs_.Address_point_ExProg:=Mcs_.Address_point_ExProg+1;
   by1:=Mcs_.In_Data_Mem[Rn];
   by2:=Mcs_.In_Data_Mem[_ACC];
   by3:=Mcs_.In_Data_Mem[_PSW];
    asm
      pusha
      pushf
      mov al,by3
      and al,$80          //check carryflag
      jz @flagzero
      stc
      jmp @other
```


รูปที่ 3.56 ฟังก์ชัน ADDC

```
@flagzero:  clc
@other:    mov al,by2
   mov bl,by1
   adc al,bl
   //check parity,carry,ac
   lahf
   mov by2,al
   mov by1,ah
   //check overflow
   jo @kob
   mov ch1,'0'
   jmp @exit
@kob:  mov ch1,'1'
@exit: popf
   popa
 end;
Mcs_.In_Data_Mem[_ACC]:= by2;
 asm
   pusha
   pushf
   //carry check zone
   mov al,by1
   and al,$01
   mov cy,al
   //parity check zone
   mov al,by1
   and al,$04
   shr al,2
   mov p,al
   //auxcilary check zone
   mov al,by1
   and al,$10
   shr al,4
   mov ac,al
   popf
   popa
 end;
   if cy=0 then MCS_:=CY_clear(MCS_)
else MCS_:=CY_set(MCS_);
   if ac=0 then MCS_:=AC_clear(MCS_)
else MCS_:=AC_set(MCS_);
   if p=0 then MCS_:=P_set(MCS_) else
MCS_:=P_clear(MCS_);
   if ch1='0' then Mcs_:=OV_clear(MCS_)
else Mcs_:=OV_set(MCS_);
 ADDC_A_Rn:=MCS_;
end;
```

รูปที่ 3.56 (ต่อ) ฟังก์ชัน ADDC

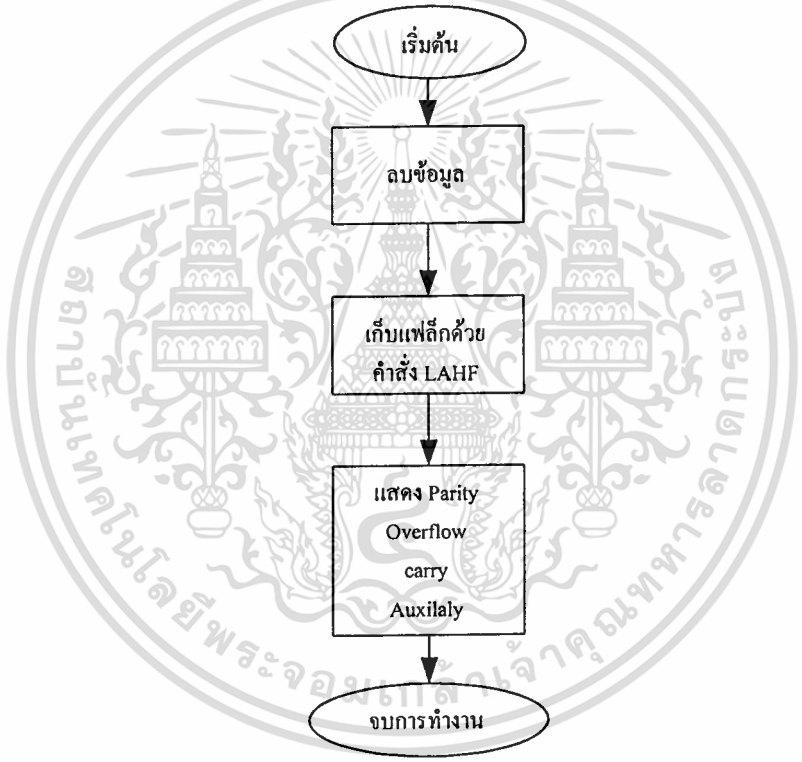
13. function SUBB_A_Rn(MCS_:TMcs_Memory;Rn:byte):TMcs_Memory;

<u>หน้าที่</u>

ลบข้อมูลระหว่างรีจิสเตอร์ A กับ Rn แล้วเก็บค่าลงในรีจิสเตอร์ A พร้อมกับคำนวณค่า แฟล็ก ซึ่งประกอบด้วย parity , carry , overflow , auxilary

การทำงาน

เมื่อทำการลบค่าระหว่าง รีจิสเตอร์ A จำลองกับรีจิสเตอร์ Rn จำลองค่าที่ได้ระหว่างการลบจะส่งให้กับคำสั่ง lahf ซึ่งคำสั่งนี้จะเก็บค่าแฟล็กไว้ที่รีจิสเตอร์ AH แล้วนำค่าที่ได้ในรีจิสเตอร์ AH มาแยกหาค่าของแฟล็กแต่ละตัวในแต่ละบิต นำค่าที่ได้ในแต่ละบิตเข้าสู่กลุ่มเงื่อนไข คำสั่ง IF ซึ่งในกลุ่มคำสั่ง IF จะเรียกใช้ฟังก์ชันที่มีหน้าที่ในการกำหนดแฟล็กให้มีค่าเป็น"0" หรือมีค่าเป็น "1" ซึ่งประกอบไปด้วย P_set , P_clear , CY_set , CY_clear , AC_set , AC_clear , OV_set , OV_clear ดังรูปที่ 3.57



รูปที่ 3.57 การทำงานของฟังก์ชัน SUBB

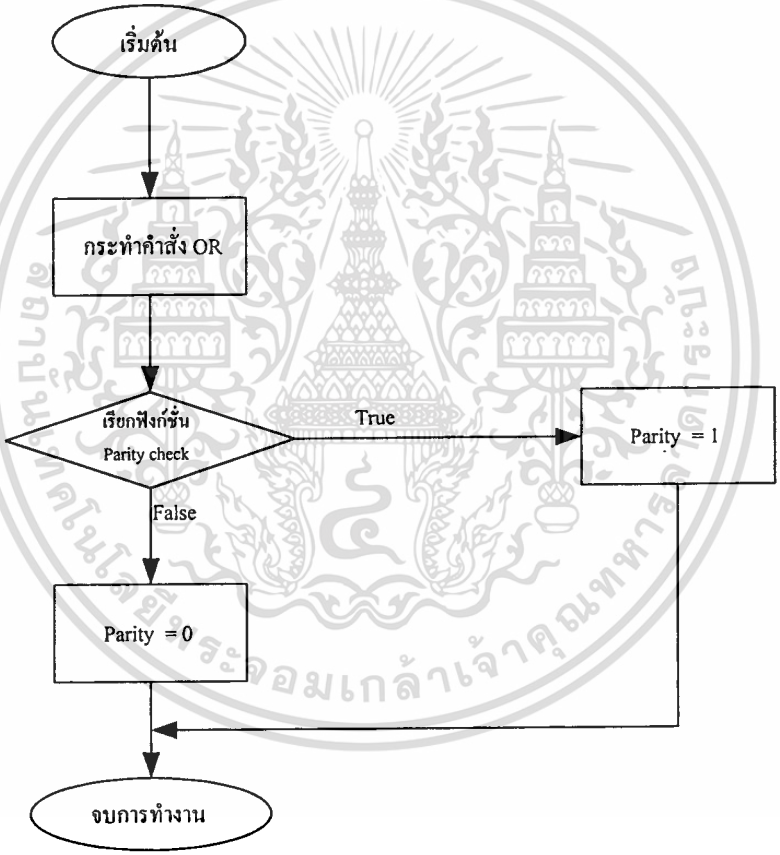
14. function ORL_A_Rn(MCS_:TMcs_Memory;Rn:byte):TMcs_Memory;

หน้าที่

OR ข้อมูลระหว่างรีจิสเตอร์ A กับ Rn ผลลัพธ์ที่ได้จะเก็บไว้ในรีจิสเตอร์ A และโปรแกรมจะทำการคิดพาริตี้แฟล็ก

การทำงาน

โดยในส่วนที่ทำการ OR กันผลที่ได้จะเก็บข้อมูลในรีจิสเตอร์ A จำลองจากนั้นนำค่าที่ได้จากรีจิสเตอร์ A จำลองส่งให้กับฟังก์ชัน pritycheck เพื่อตรวจสอบค่า parity แฟล็กโดยฟังก์ชันตัวนี้จะส่งค่ากลับเป็นตรรก ซึ่งถ้ามีค่าเป็นจริงจะเรียกใช้ฟังก์ชัน P_set เพื่อกำหนดให้ parity แฟล็กมีค่าเป็นหนึ่งแต่ถ้าเป็นเท็จจะเรียกใช้ฟังกชั่น P_clear เพื่อกำหนดให้ parity แฟล็ก มีค่าเป็น 0 ดังรูปที่ 3.58



รูปที่ 3.58 การทำงานของฟังก์ชัน ORL

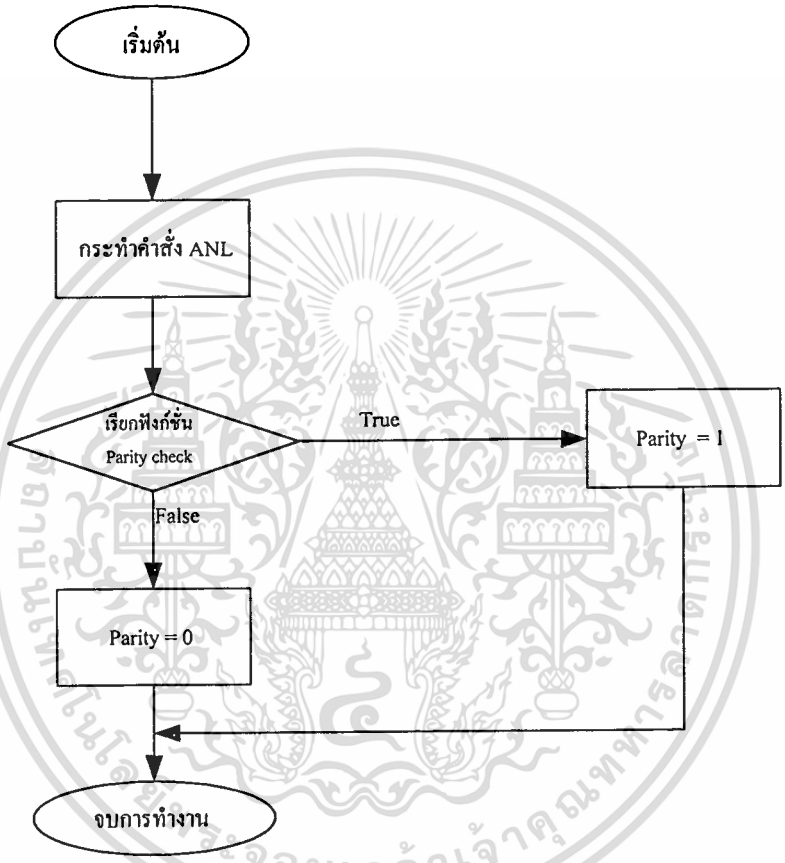
15. function ANL_A_Rn(MCS_:TMcs_Memory;Rn:byte):TMcs_Memory;

หน้าที่

AND ข้อมูลระหว่างรีจิสเตอร์ A กับรีจิสเตอร์ Rn ผลที่ได้เก็บข้อมูลในตัวแปร A และคำสั่ง AND นี้จะคิด parity แฟล็ก ด้วย

การทำงาน

จากโปรแกรมจะสังเกตได้ว่า โปรแกรมจะคิด parity แฟล็ก โดยการเรียกใช้ฟังก์ชัน paritycheck ซึ่งฟังก์ชันนี้จะส่งค่ากลับมาเป็นตรรกะ โดยที่ถ้าเป็นจริงจะเรียกใช้ฟังก์ชัน P_set ส่วนถ้าเป็นเท็จจะเรียกใช้ฟังก์ชัน P_clear ดังรูปที่ 3.59



รูปที่ 3.59 การทำงานของฟังก์ชัน ANL

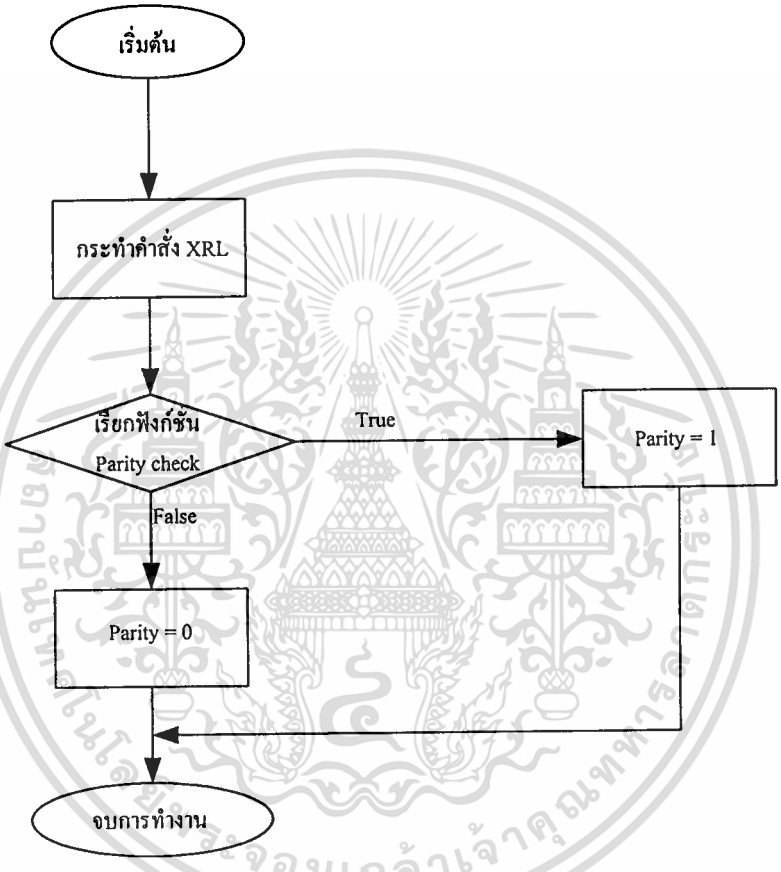
16. function XRL_A_Rn(MCS_:TMcs_Memory;Rn:byte):TMcs_Memory;

หน้าที่

ในการ XOR ระหว่างรีจิสเตอร์ A กับรีจิสเตอร์ Rn ผลที่ได้จะเก็บค่าไว้ที่รีจิสเตอร์ A พร้อมกับคิด parity แฟล็ก

การทำงาน

ในการทำ xor จะเรียกใช้จากคำสั่ง xor ซึ่งมีอยู่ในภาษาแอสแซมบลี้ ต่อจากนั้นเรียกใช้ฟังก์ชัน paritycheck โดยที่ที่ส่งกลับจากฟังก์ชันตัวนี้จะมีค่าเป็นตรรกะ โดยที่ถ้าเป็นจริงจะเรียกใช้ฟังก์ชัน P_set ส่วนถ้าเป็นเท็จจะเรียกใช้ฟังก์ชัน P_clear ดังรูปที่ 3.60



รูปที่ 3.60 การทำงานของฟังก์ชัน XRL

17. function XCH_A_Rn(MCS_:TMcs_Memory;Rn:byte):TMcs_Memory;

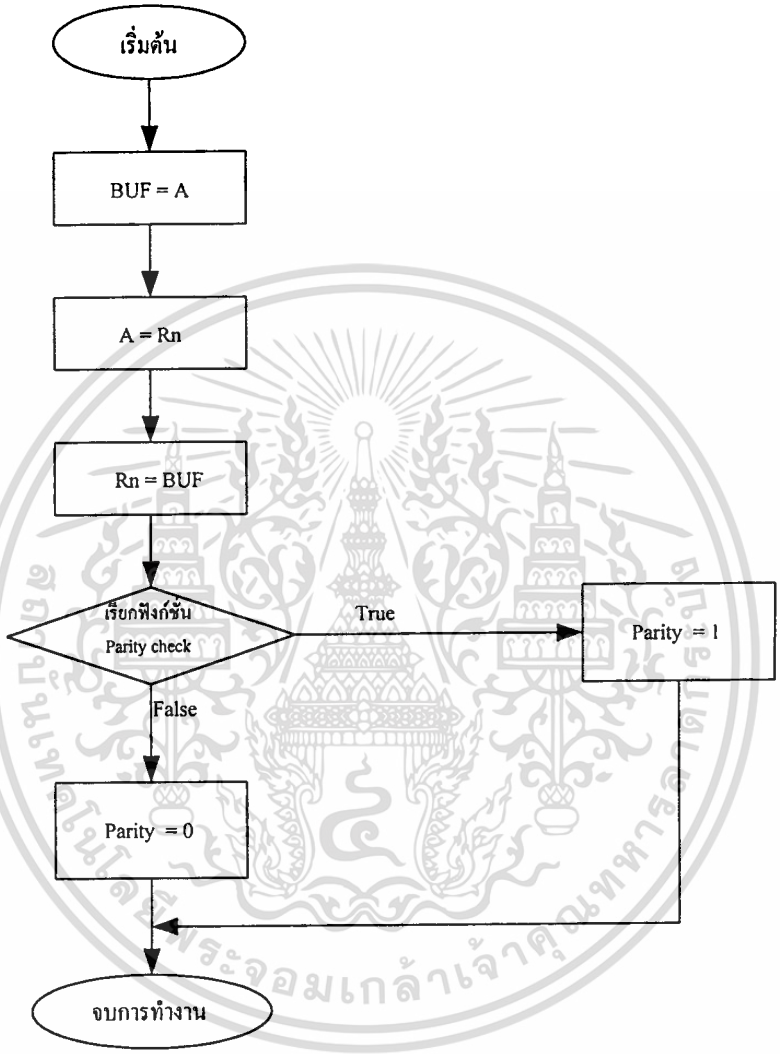
หน้าที่

ทำการสลับค่าระหว่างรีจิสเตอร์ A กับรีจิสเตอร์ Rn พร้อมกับคิด parity แฟล็กด้วย

การทำงาน

โปรแกรมจะรับค่าจากรีจิสเตอร์ A จำลองและเก็บค่าลงในบัฟเฟอร์ และนำค่าที่รีจิสเตอร์ Rn จำลองไปเก็บไว้ในรีจิสเตอร์ A จำลองต่อจากนั้น นำค่าจากบัฟเฟอร์ไปเก็บไว้ที่ รีจิสเตอร์ Rn โปรแกรมทำการตรวจสอบค่า parity โดยการนำค่ารีจิสเตอร์ A จำลองส่งให้กับ

ฟังก์ชัน paritycheck โดยที่ที่ส่งกลับจากฟังก์ชันตัวนี้จะมีค่าเป็นตรรกะ โดยที่ถ้าเป็นจริง จะเรียนใช้ฟังก์ชัน P_set ส่วนถ้าเป็นเท็จจะเรียกใช้ฟังก์ชัน P_clear ดังรูปที่ 3.61



รูปที่ 3.61 การทำงานของฟังก์ชัน XCH

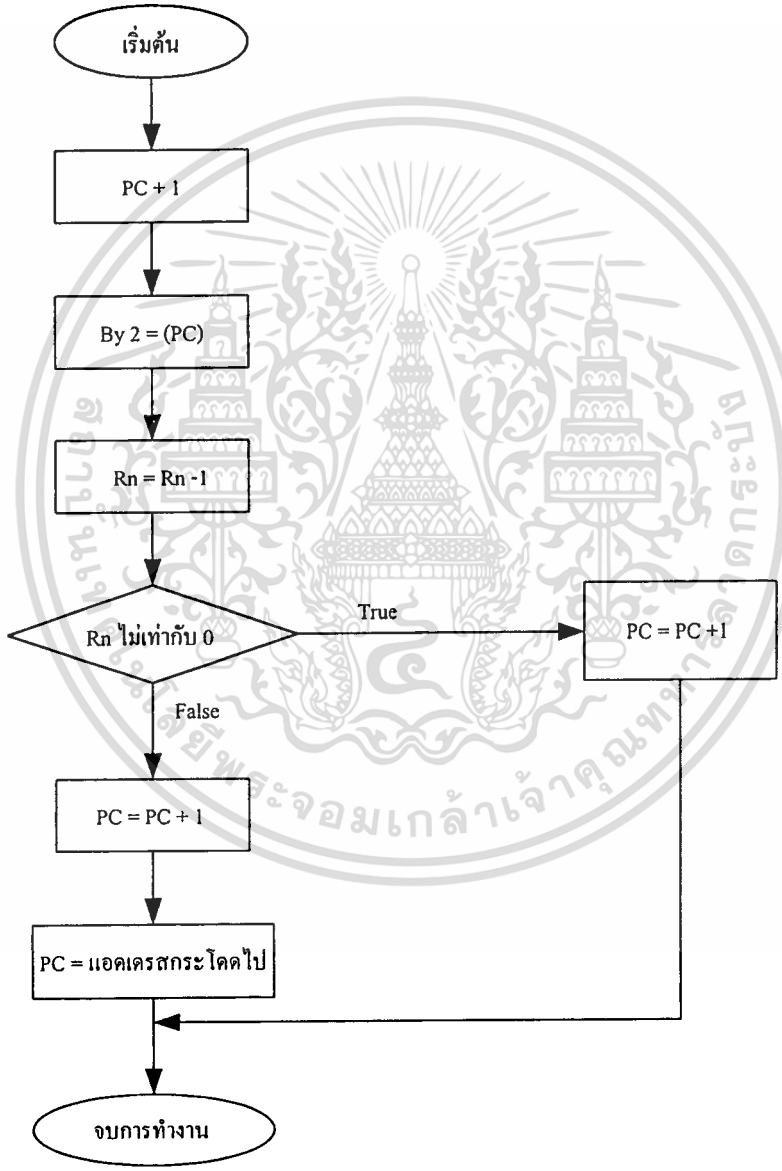
18. function DJNZ_Rn(MCS_:TMcs_Memory;Rn:byte):TMcs_Memory;

<u>หน้าที่</u>

DJNZ Rn ตรวจสอบค่ารีจิสเตอร์ Rn ว่าเมื่อนำมาลบ 1 มีค่าเป็น 0 หรือ ไม่ถ้ามีค่าเป็น 0 ให้กระโดดไปที่แอดเดรสที่ระบุไว้

การทำงาน

โปรแกรมจะรับค่ารีจิสเตอร์ Rn จำลองนำค่ารีจิสเตอร์ Rn มาลบหนึ่งพร้อมกับเก็บค่าลงในรีจิสเตอร์ Rn จำลอง นำค่าในรีจิสเตอร์ดังกล่าวไปตรวจสอบว่ามีค่าเป็น 0 หรือไม่ถ้ามีค่าไม่เท่ากับ 0 ให้เปลี่ยนค่าตัวชี้แอดเดรส Mcs_.Address_point_Exprog ให้เป็นแอดเดรสที่คำสั่ง djnz ระบบให้กระโดดไป ดังรูปที่ 3.62



รูปที่ 3.62 การทำงานของฟังก์ชัน DJNZ

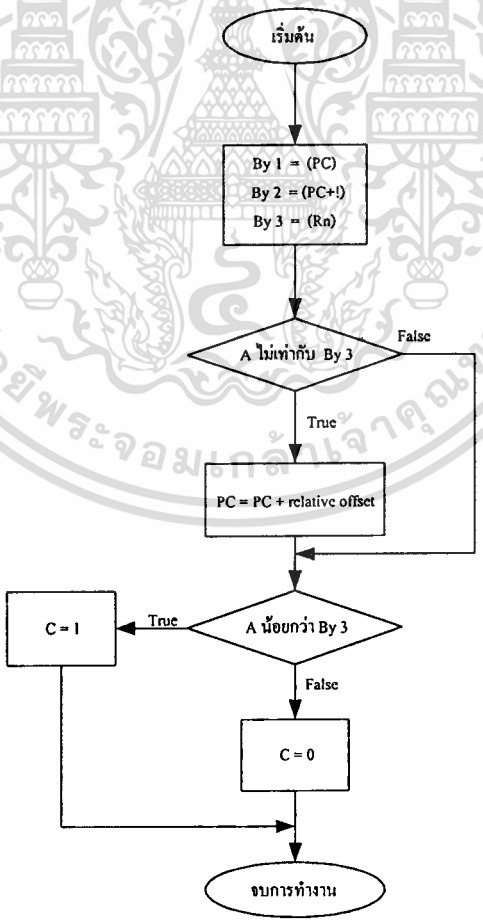
19. function CJNE_Rn(Mcs_:TMcs_Memory;Rn:byte):TMCS_Memory;

หน้าที่

เปรียบเทียบค่าระหว่างรีจิสเตอร์ A กับรีจิสเตอร์ Rn ถ้ามีค่าไม่เท่ากัน ให้กระโดดไปในตำแหน่งที่ CJNE ระบุ แต่ถ้ามีค่าเท่ากันให้ทำในคำสั่งถัดไป ต่อจากนั้นทำการเปรียบเทียบว่าถ้าค่า A มีค่าน้อยกว่า Rn ให้ CY มีค่าเป็น "1" แต่ถ้าไม่ใช้ให้ค่า CY เป็น "0"

การทำงาน

โปรแกรมจะรับข้อมูลจากรีจิสเตอร์ A จำลองนำไปเปรียบเทียบกับ ค่าที่อยู่ในรีจิสเตอร์ Rn จำลอง ถ้ามีค่าไม่เท่ากัน ให้กระโดดไปในตำแหน่งที่ CJNE ระบุ พร้อมกับตรวจสอบด้วยเงื่อนไขว่าถ้าค่ารีจิสเตอร์ A จำลองมีค่าน้อยกว่าค่ารีจิสเตอร์ Rn จำลอง ถ้าจริงให้ เรียกใช้ฟังก์ชัน CY_set แต่ถ้าเป็นเท็จให้เรียกฟังก์ชัน CY_clear จากเงื่อนไขแรกมีตรวจสอบคือ รีจิสเตอร์ A จำลองมีค่าไม่เท่ากับค่ารีจิสเตอร์ Rn จำลอง ถ้าเป็นเท็จให้ทำในตำแหน่งต่อไป ดังรูปที่ 3.63

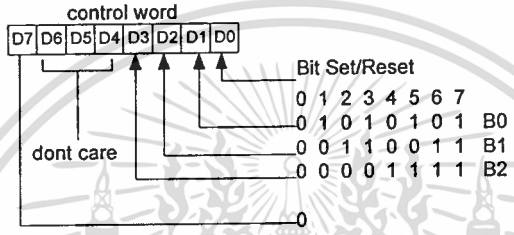


รูปที่ 3.63 การทำงานของฟังก์ชัน CJNE

3.13 การสร้างฟังก์ชันจำลองการทำงานของ 8255

3.13.1 การทำงานแบบ non-active

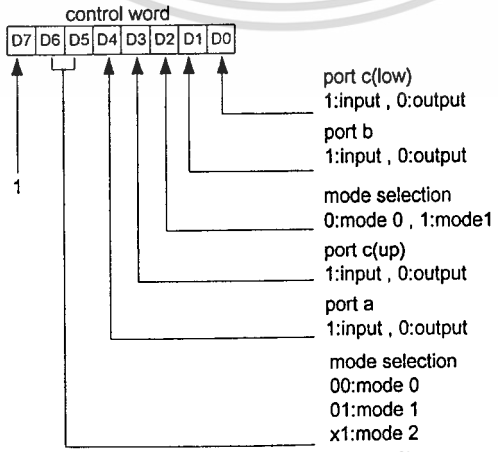
การทำงานแบบ non-active จะให้พอร์ต PC เป็นเอาต์พุตโดยการกำหนดให้พอร์ตใดมีสภาวะเป็น 0 หรือ 1 ได้จาก control word ดังรูปที่ 3.64โดยในการทำงานแบบ non-active ได้นั้นบิตที่ 7 ของ controlwod จะต้องมีค่าเป็น 0 ใช้บิตที่ 1-3 เป็นบิตที่กำหนดเลือกบิตของพอร์ต PC ตั้งแต่ PC.0 ถึง PC.7 และในบิตที่ 0 ใช้ในการกำหนดให้ บิตใดๆที่เลือกไว้มีสภาวะเป็น 0 หรือ 1



รูปที่ 3.64 การทำงานของ 8255 แบบ non-active

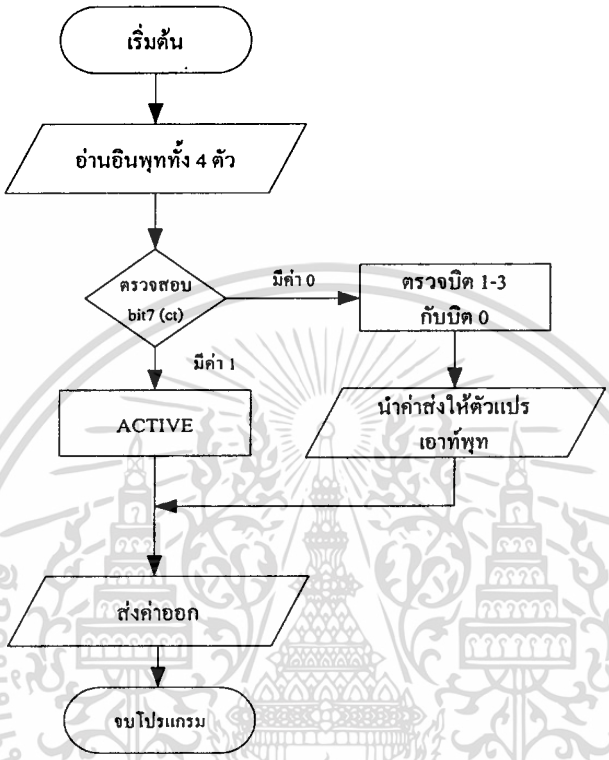
3.13.2 การทำงานแบบ active

ในการทำงานแบบ active 0 จะประกอบด้วย 3 โหมด คือ Mode0 , Mode1 และ Mode 2 แต่ในการจำลองการทำงานของ 8255 ออกแบบมาให้สามารถจำลองการทำงานได้เพียง Mode 0 เพียงอย่างเดียว เพราะหากออกแบบให้8255 สามารถทำงานได้หลายโหมดจะเกิดความซับซ้อนในโปรแกรมจำลอง มี control word ดังรูปที่ 3.65



รูปที่ 3.65 การทำงานแบบ active

จากลักษณะของ 8255 เรานำการทำงานในโหมด 0 กับการทำงานแบบ non-active มาเขียนเป็นโปรแกรมโดยเขียนเป็นโฟว์ชาร์ต ได้ดังรูปที่ 3.66



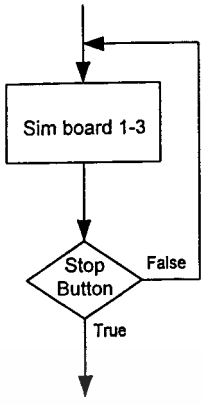
รูปที่ 3.66 โครงสร้างของฟังก์ชันการจำลอง 8255

3.14 การสร้างฟอร์มแสดงผลชุดอินเตอร์เฟสจำลอง

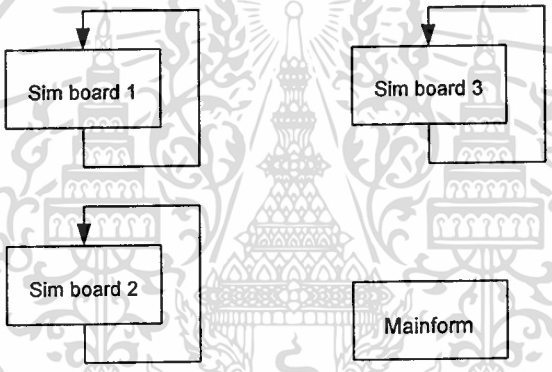
ชุดอินเตอร์เฟสจำลอง เป็นชุดที่จำลองการทำงานเสมือนกับการทำงานบนวงจรจริง ดังนั้นส่วนที่ชุดอินเตอร์เฟสจำลองจำเป็นต้องใช้ฟังก์ชันจำลองการทำงานของ 8255 โดยที่ฟังก์ชันจำลองการทำงานของ 8255 จะอยู่ในรูปของไฟล์ DLL ในชุดอินเตอร์เฟสจำลองได้ แบ่งเทรดโปรแกรมออกเป็น 3 ส่วนตามลักษณะการอินเตอร์เฟส โดยประกอบด้วย

1. Interface 1 ชื่อไฟล์เทรด simboard_1.pas
2. Interface 2 ชื่อไฟล์เทรด simboard_2.pas
3. Interface 3 ชื่อไฟล์เทรด simboard_3.pas

สาเหตุที่ต้องมีการใช้เทรดเพราะว่า ในการจำลองการทำงานหากผู้เขียนโปรแกรมใช้การทำลูปเพียงอย่างเดียวโปรแกรมไม่สามารถแบ่งส่วนเวลาในการทำงานไปทำในส่วนย่อยๆอื่นๆได้เพราะจะเสียเวลาในการประมวลผลในส่วนที่การทำงานลูปนั้นยังค้างอยู่จึงต้องเขียนโปรแกรมลูปดังรูปที่ 3.67



รูปที่ 3.67 การเขียนโปรแกรมโดยการทำลูปแบบเดิม

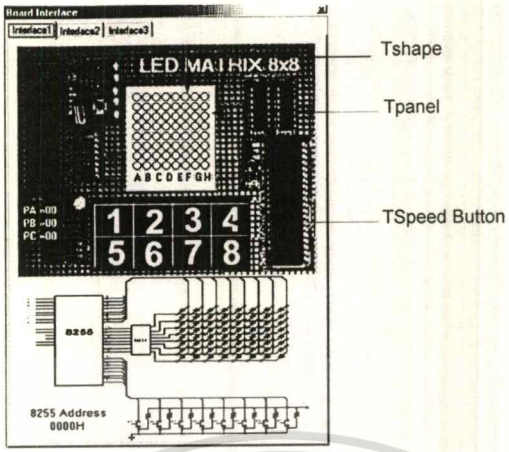


รูปที่ 3.68 การเขียนโปรแกรมโดยการใช้เทรดแบ่งเวลาในการทำลูป

จากรูปที่ 3.68 ดังนั้นจึงเห็นว่าโปรแกรมในส่วนของอินเตอร์เฟสจำลอง จำเป็นต้องมีการทำเทรดเพื่อช่วยให้เกิดความเร็วในการทำงาน

3.14.1 อินเตอร์เฟสจำลองตัวที่ 1

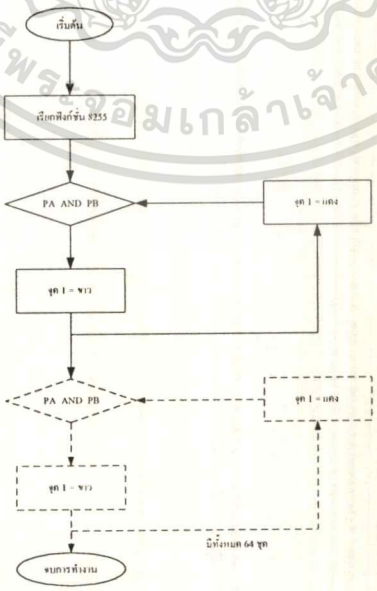
ในชุดอินเตอร์เฟสจำลองตัวที่ 1 (รูปที่ 3.69) จะเป็นโปรแกรมแสดงผล LED Dotmatrix 8x8 โดยการจำลองการทำงานจะต้องใช้ ฟังก์ชันจำลองการทำงานของ 8255 ในส่วนของตัว LED จำลองใช้คอมโปเนนต์ Tshape กำหนดให้มีลักษณะเป็นรูปวงกลม ในการแสดงผลใช้การกำหนดค Tshape ให้เปลี่ยนสีโดยการเปลี่ยนคุณสมบัติทาง Tshape.color ส่วนของปุ่มจะใช้คอมโปเนนต์ Tspeedbutton ส่วนรูปของปุ่มกำหนดให้เป็นไฟล์ฟอร์แมต BMP มีขนาดกว้าง 51 จุด สูง 59 จุด โดยมีทั้งหมด 8 ปุ่ม กำหนดให้เปิดเป็นชื่อ Key1.bmp ถึง Key8.bmp



รูปที่ 3.69 อินเตอร์เฟสจำลองตัวที่ 1

3.14.2 กระบวนการทำงานของ อินเตอร์เฟสจำลองตัวที่1

ในการทำเทรดของชุดอินเตอร์เฟสจำลอง เมื่อเข้าสู่วงรอบการทำเทรดโปรแกรมจะเรียกใช้ฟังก์ชันของ 8255 ฟังก์ชันของ 8255 จะส่งค่าของพอร์ต PA จำลอง, PBจำลอง และ PC จำลอง ค่าของ PA กับ PB จะถูกนำมา AND กันเมื่อค่า PA กับ PB มีค่าเป็นหนึ่ง จะทำให้ Tshape ซึ่งจำลองเป็น LED เปลี่ยนเป็นสีแดง ดังนั้นในการเช็คเงื่อนไขของ PA กับ PB จะต้องมีทั้งหมด 64 ตัวตามจำนวนของ Tshape ที่ใช้จำลองเป็น LED ดังแสดงการทำงานในรูปที่ 3.70 และเขียนโปรแกรมได้ดังรูปที่ 3.71



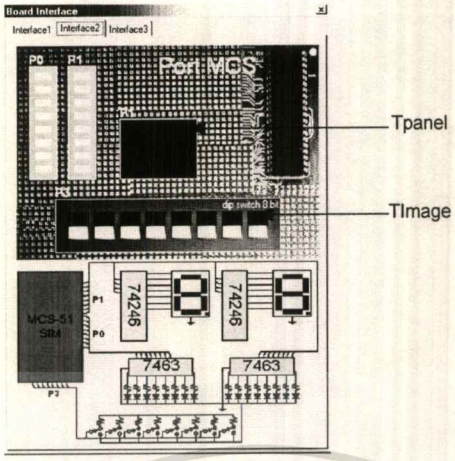
รูปที่ 3.70 การทำงานของ เทรด Simboard_.pas

```
procedure TSim_B1.UpdataShape;
  var Add8255_2:word;
  begin
     Add8255_2:=Simboard.Address8255_1;  //address 8255
     Simboard1.PC:= Key_interf1;
     Simboard1.Ex_Data_Mem[0]:=Mcs_.Ex_Data_Mem[Add8255_2];
     Simboard1.Ex_Data_Mem[1]:=Mcs_.Ex_Data_Mem[Add8255_2+1];
     Simboard1.Ex_Data_Mem[2]:=Mcs_.Ex_Data_Mem[Add8255_2+2];
     Simboard1.Ex_Data_Mem[3]:=Mcs_.Ex_Data_Mem[Add8255_2+3];
     Simboard1:=V8255(Simboard1); //updata and run 8255
     Mcs_.Ex_Data_Mem[Add8255_2]:=Simboard1.Ex_Data_Mem[0];
     Mcs_.Ex_Data_Mem[Add8255_2+1]:=Simboard1.Ex_Data_Mem[1];
     Mcs_.Ex_Data_Mem[Add8255_2+2]:=Simboard1.Ex_Data_Mem[2];
     Mcs_.Ex_Data_Mem[Add8255_2+3]:=Simboard1.Ex_Data_Mem[3];
      buf.interf1_t1.Caption:= 'PA ='+inttohex(Simboard1.PA,2);
      buf.interf1_t2.Caption:= 'PB ='+inttohex(Simboard1.PB,2);
      buf.interf1_t3.Caption:= 'PC ='+inttohex(Simboard1.PC,2);
      if Simboard1.PA_[0] and Simboard1.PB_[0] then Buf.DA0.Brush.Color:=clred
        else Buf.DA0.Brush.Color:= clWhite;
      if Simboard1.PA_[0] and Simboard1.PB_[1] then Buf.DA1.Brush.Color:=clred
        else Buf.DA1.Brush.Color:= clWhite;
  ..........
      if Simboard1.PA_[7] and Simboard1.PB_[7] then Buf.DH7.Brush.Color:=clred
        else Buf.DH7.Brush.Color:= clWhite;
  end;
```

รูปที่ 3.71 ฟังก์ชันเปลี่ยนสีคอมโปเนนต์ในอินเตอร์เฟส 1

3.14.3 อินเตอร์เฟสจำลองตัวที่ 2

การทำงานของอินเตอร์เฟสตัวนี้เป็นการจำลองการทำงานของพอร์ตของ MCS-51 ซึ่งพอร์ตที่ใช้ในอินเตอร์เฟสตัวนี้ประกอบด้วย P0,P1 และ P3 โดยที่ P3 จะเป็นอินพุตต่ออยู่กับ ดิพสวิตซ์จำลอง พอร์ต P1 ต่ออยู่กับ LEDจำลอง กับ SevenSegment จำลอง และพอร์ต P0ต่ออยู่กับ LED เพียงอย่างเดียว ดังรูปที่ 3.72 สำหรับคอมโปเนนต์ที่ใช้ใน SevenSegment จำลองคือ Tpanel ส่วนดิพสวิตซ์จำลอง ใช้ Timage ในการแสดงรูปภาพ โดยเมื่อมีการคลิกที่รูปภาพสวิตซ์จะเปลี่ยนรูปภาพที่ใช้แสดงสวิตซ์เป็นสถานะที่กลับกัน



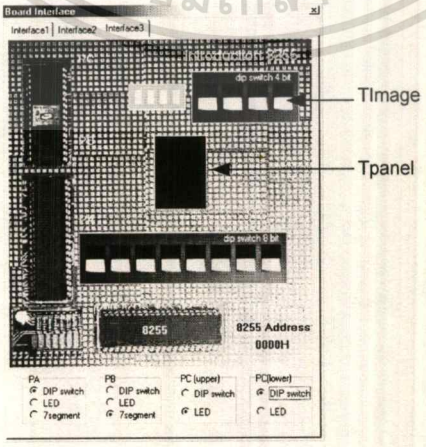
รูปที่ 3.72 อินเตอร์เฟสจำลองตัวที่ 2

3.14.4 กระบวนการทำงานของ อินเตอร์เฟสจำลองตัวที่2

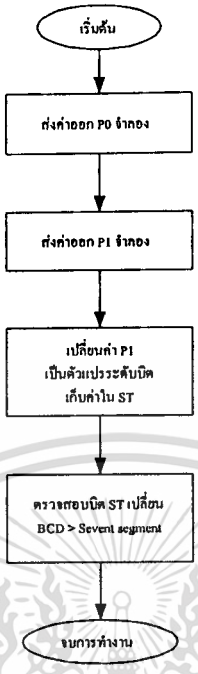
การทำงานในขั้นแรกจะส่งค่าข้อมูลออกไปยัง LED P0 จำลอง จากนั้นส่งไปยัง LED P1 จำลอง เมื่อเสร็จแล้วจะส่งค่าออกข้อมูล P1 ออกไปแปลงเป็น BCD to Seven และส่งออกไปยัง SevenSegment จำลอง ดังรูปที่ 3.74 และเขียนโปรแกรมได้ดังรูปที่ 3.75

3.14.5 อินเตอร์เฟสจำลองตัวที่ 3

อินเตอร์เฟสจำลองดังรูปที่ 3.73 ตัวนี้ใช้คอมโปเนนต์ Tpanel , Timage และใช้การกำหนดการ Visible ปิดเปิดการแสดงผล ทำให้สามารถเลือกอุปกรณ์ที่ใช้ในการอินเตอร์เฟสได้



รูปที่ 3.73 อินเตอร์เฟสจำลองตัวที่ 3



รูปที่ 3.74 การทำงานของ เทรด Simboard_2.pas

```
procedure TSim_B2.UpdataShape;
var
  st1,st2,st4,st5,st:string[8];
begin
   Mcs_.In_Data_Mem[_P3]:=Key_buf_interf2;
   st1:=inttobit(Mcs_.In_data_Mem[_P0]);
   st2:=inttobit(Mcs_.In_data_Mem[_P1]);
   with simboard do begin
    if st1[1]='1' then L10.Color:= clred else L10.Color:= clmenu;
    if st1[2]='1' then L11.Color:= clred else L11.Color:= clmenu;
    if st1[3]='1' then L12.Color:= clred else L12.Color:= clmenu;
    if st1[4]='1' then L13.Color:= clred else L13.Color:= clmenu;
    if st1[5]='1' then L14.Color:= clred else L14.Color:= clmenu;
    if st1[6]='1' then L15.Color:= clred else L15.Color:= clmenu;
    if st1[7]='1' then L16.Color:= clred else L16.Color:= clmenu;
    if st1[8]='1' then L17.Color:= clred else L17.Color:= clmenu;
    if st2[1]='1' then L20.Color:= clred else L20.Color:= clmenu;
    if st2[2]='1' then L21.Color:= clred else L21.Color:= clmenu;
    if st2[3]='1' then L22.Color:= clred else L22.Color:= clmenu;
    if st2[4]='1' then L23.Color:= clred else L23.Color:= clmenu;
    if st2[5]='1' then L24.Color:= clred else L24.Color:= clmenu;
```


รูปที่ 3.75 ฟังก์ชันแสดงผลออกคอมโปเนนต์ในอินเตอร์เฟส 2

```
 if st2[6]='1' then L25.Color:= clred else L25.Color:= clmenu;
 if st2[7]='1' then L26.Color:= clred else L26.Color:= clmenu;
 if st2[8]='1' then L27.Color:= clred else L27.Color:= clmenu;
end; //end with
st:=inttobit(Mcs_.in_data_mem[_P1]);
case bittoint(st[1]+st[2]+st[3]+st[4]+'0000') of
    $0:st4:='11111100';
    $1:st4:='01100000';
    $2:st4:='11011010';
    $3:st4:='11110010';
    $4:st4:='01100110';
    $5:st4:='10110110';
    $6:st4:='10111110';
    $7:st4:='11100000';
    $8:st4:='11111110';
    $9:st4:='11110110';
    $a:st4:='11101110';
    $b:st4:='00111110';
    $c:st4:='10011100';
    $d:st4:='01111010';
    $e:st4:='10011110';
    $f:st4:='10001110';
end; //end case
case bittoint(st[5]+st[6]+st[7]+st[8]+'0000') of
    $0:st5:='11111100';
    $1:st5:='01100000';
    $2:st5:='11011010';
    $3:st5:='11110010';
    $4:st5:='01100110';
    $5:st5:='10110110';
    $6:st5:='10111110';
    $7:st5:='11100000';
    $8:st5:='11111110';
    $9:st5:='11110110';
    $a:st5:='11101110';
    $b:st5:='00111110';
    $c:st5:='10011100';
    $d:st5:='01111010';
```

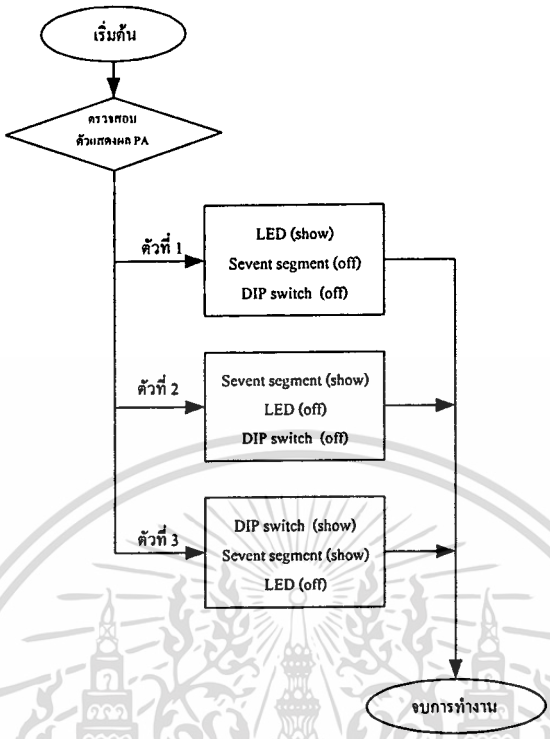

รูปที่ 3.75 (ต่อ) ฟังก์ชันแสดงผลออกคอมโปเนนต์ในอินเตอร์เฟส 2

```
            $e:st5:='10011110';
            $f:st5:='10001110';
        end; //end case
    with simboard do
     begin
        if st5[1]='1' then M1a.Color:= clred else M1a.Color:= clblack;
        if st5[2]='1' then M1b.Color:= clred else M1b.Color:= clblack;
        if st5[3]='1' then M1c.Color:= clred else M1c.Color:= clblack;
        if st5[4]='1' then M1d.Color:= clred else M1d.Color:= clblack;
        if st5[5]='1' then M1e.Color:= clred else M1e.Color:= clblack;
        if st5[6]='1' then M1f.Color:= clred else M1f.Color:= clblack;
        if st5[7]='1' then M1g.Color:= clred else M1g.Color:= clblack;
        if st5[8]='1' then M1p.Color:= clred else M1p.Color:= clblack;
        if st4[1]='1' then M2a.Color:= clred else M2a.Color:= clblack;
        if st4[2]='1' then M2b.Color:= clred else M2b.Color:= clblack;
        if st4[3]='1' then M2c.Color:= clred else M2c.Color:= clblack;
        if st4[4]='1' then M2d.Color:= clred else M2d.Color:= clblack;
        if st4[5]='1' then M2e.Color:= clred else M2e.Color:= clblack;
        if st4[6]='1' then M2f.Color:= clred else M2f.Color:= clblack;
        if st4[7]='1' then M2g.Color:= clred else M2g.Color:= clblack;
        if st4[8]='1' then M2p.Color:= clred else M2p.Color:= clblack;
     end;
end;
```

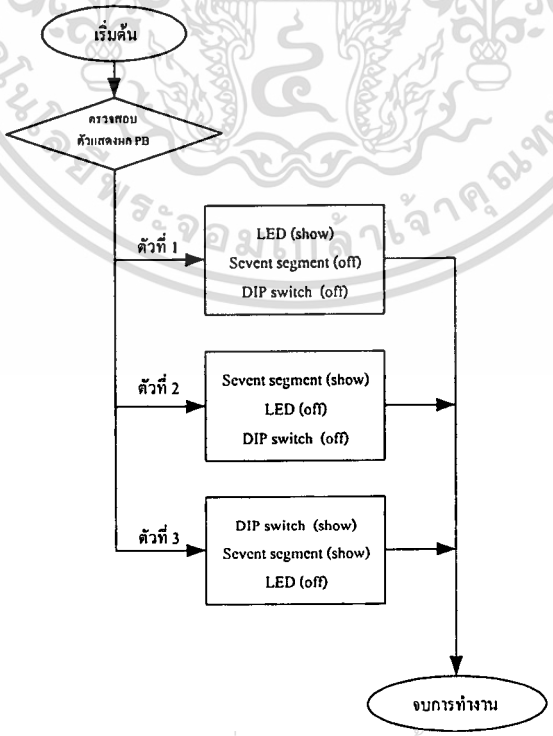
รูปที่ 3.75 (ต่อ) ฟังก์ชันแสดงผลออกคอมโปเนนต์ในอินเตอร์เฟส 2

3.14.6 กระบวนการทำงานของอินเตอร์เฟสจำลองตัวที่ 3

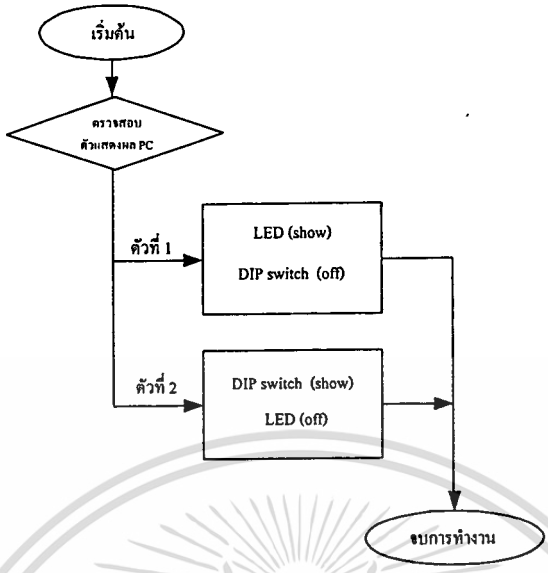
เป็นตัวอินเตอร์เฟสจำลองการทำงานของ 8255 ที่สามารถเปลี่ยนอินพุตเอาต์พุต ของพอร์ต PA,PB และ PC ได้ ดังนั้นในการออกแบบการทำงานจึงต้องมีส่วนที่สลับสถานะของพอร์ตเป็นอินพุตหรือเอาต์พุต พร้อมทั้งจะต้องสัมพันธ์กับค่าของ controlword ของ8255 ในโหมด 0 ด้วย อีกทั้งหาก 8255 ทำงานในโหมด non-active ได้ ดังรูปที่ 3.76 , รูปที่ 3.77 และรูปที่ 3.78



รูปที่ 3.76 กระบวนการทำงานเมื่อเลือก อินพุตเอาต์พุตของ PA



รูปที่ 3.77 กระบวนการทำงานเมื่อเลือก อินพุตเอาต์พุตของ PB



รูปที่ 3.78 กระบวนการทำงานเมื่อเลือก อินพุตเอาต์พุตของ PC ทั้งบนและล่าง

จะสังเกตได้จากฟังก์ชันที่ใช้ภายในเทรดจะพบว่ามีคำสั่ง IF เพื่อตรวจสอบการ Visible ของ คอมโปเนนต์ เพื่อใช้ในการกำหนดการแสดงผลเอาต์พุตของคอมโปเนนต์นั้นๆ ดังนั้นทำให้เกิดการเลือกในการแสดงผลทั้งทางด้านอินพุตและเอาต์พุต ดังโปรแกรมรูปที่ 3.79

```
procedure TSim_B3.UpdataShape;
var
   Add8255_2:word;
begin
    Add8255_2:=Simboard.Address8255_2;  //address 8255
    SimBoard3.Ex_Data_Mem[0]:=Mcs_.Ex_Data_Mem[Add8255_2];
    SimBoard3.Ex_Data_Mem[1]:=Mcs_.Ex_Data_Mem[Add8255_2+1];
    SimBoard3.Ex_Data_Mem[2]:=Mcs_.Ex_Data_Mem[Add8255_2+2];
    SimBoard3.Ex_Data_Mem[3]:=Mcs_.Ex_Data_Mem[Add8255_2+3];
    Simboard3:=V8255(Simboard3); //updata and run 8255
    Mcs_.Ex_Data_Mem[Add8255_2]:=Simboard3.Ex_Data_Mem[0];
    Mcs_.Ex_Data_Mem[Add8255_2+1]:=Simboard3.Ex_Data_Mem[1];
    Mcs_.Ex_Data_Mem[Add8255_2+2]:=Simboard3.Ex_Data_Mem[2];
    Mcs_.Ex_Data_Mem[Add8255_2+3]:=Simboard3.Ex_Data_Mem[3];
    with simboard do begin
      if  int3_led_pc1.Visible then begin
```


รูปที่ 3.79 ฟังก์ชันที่ใช้ในการแสดงผลของคอมโปเนนต์ในอินเตอร์เฟส 3

```
  if simboard3.PC_[0] then L30.Color:= clred else L30.Color:= clmenu;
  if simboard3.PC_[1] then L31.Color:= clred else L31.Color:= clmenu;
  if simboard3.PC_[2] then L32.Color:= clred else L32.Color:= clmenu;
  if simboard3.PC_[3] then L33.Color:= clred else L33.Color:= clmenu;
end;
if  int3_led_pc2.Visible then begin
  if simboard3.PC_[4] then L34.Color:= clred else L34.Color:= clmenu;
  if simboard3.PC_[5] then L35.Color:= clred else L35.Color:= clmenu;
  if simboard3.PC_[6] then L36.Color:= clred else L36.Color:= clmenu;
  if simboard3.PC_[7] then L37.Color:= clred else L37.Color:= clmenu;
end;
if  int3_led_pb.Visible then begin
  if simboard3.PB_[0] then L40.Color:= clred else L40.Color:= clmenu;
  if simboard3.PB_[1] then L41.Color:= clred else L41.Color:= clmenu;
  if simboard3.PB_[2] then L42.Color:= clred else L42.Color:= clmenu;
  if simboard3.PB_[3] then L43.Color:= clred else L43.Color:= clmenu;
  if simboard3.PB_[4] then L44.Color:= clred else L44.Color:= clmenu;
  if simboard3.PB_[5] then L45.Color:= clred else L45.Color:= clmenu;
  if simboard3.PB_[6] then L46.Color:= clred else L46.Color:= clmenu;
  if simboard3.PB_[7] then L47.Color:= clred else L47.Color:= clmenu;
end;
if  int3_led_pa.Visible then begin
  if simboard3.PA_[0] then L50.Color:= clred else L50.Color:= clmenu;
  if simboard3.PA_[1] then L51.Color:= clred else L51.Color:= clmenu;
  if simboard3.PA_[2] then L52.Color:= clred else L52.Color:= clmenu;
  if simboard3.PA_[3] then L53.Color:= clred else L53.Color:= clmenu;
  if simboard3.PA_[4] then L54.Color:= clred else L54.Color:= clmenu;
  if simboard3.PA_[5] then L55.Color:= clred else L55.Color:= clmenu;
  if simboard3.PA_[6] then L56.Color:= clred else L56.Color:= clmenu;
  if simboard3.PA_[7] then L57.Color:= clred else L57.Color:= clmenu;
end;
if  int3_seg_pb.Visible then begin
  if Simboard3.PB_[0] then M4a.Color:= clred else M4a.Color:= clBlack;
  if Simboard3.PB_[1] then M4b.Color:= clred else M4b.Color:= clBlack;
  if Simboard3.PB_[2] then M4c.Color:= clred else M4c.Color:= clBlack;
  if Simboard3.PB_[3] then M4d.Color:= clred else M4d.Color:= clBlack;
  if Simboard3.PB_[4] then M4e.Color:= clred else M4e.Color:= clBlack;
  if Simboard3.PB_[5] then M4f.Color:= clred else M4f.Color:= clBlack;
```


รูปที่ 3.79 (ต่อ) ฟังก์ชันที่ใช้ในการแสดงผลของคอมโปเนนต์ในอินเตอร์เฟส 3

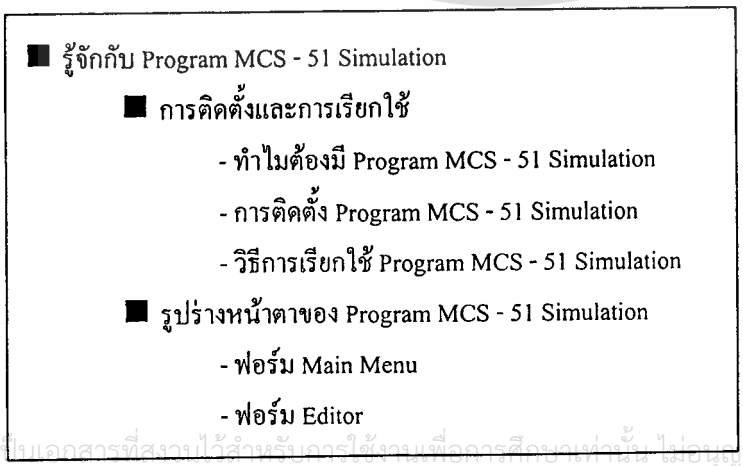
```
    if Simboard3.PB_[6] then M4g.Color:= clred else M4g.Color:= clBlack;
    if Simboard3.PB_[7] then M4p.Color:= clred else M4p.Color:= clBlack;
  end;
  if  int3_seg_pa.Visible then begin
    if Simboard3.PA_[0] then M5a.Color:= clred else M5a.Color:= clBlack;
    if Simboard3.PA_[1] then M5b.Color:= clred else M5b.Color:= clBlack;
    if Simboard3.PA_[2] then M5c.Color:= clred else M5c.Color:= clBlack;
    if Simboard3.PA_[3] then M5d.Color:= clred else M5d.Color:= clBlack;
    if Simboard3.PA_[4] then M5e.Color:= clred else M5e.Color:= clBlack;
    if Simboard3.PA_[5] then M5f.Color:= clred else M5f.Color:= clBlack;
    if Simboard3.PA_[6] then M5g.Color:= clred else M5g.Color:= clBlack;
    if Simboard3.PA_[7] then M5p.Color:= clred else M5p.Color:= clBlack;
  end;
  label8.Caption:='PA ='+inttohex(Simboard3.PA,2);
  label9.Caption:='PB ='+inttohex(Simboard3.PB,2);
  label10.Caption:='PC ='+inttohex(Simboard3.PC,2);
  label11.Caption:='CT ='+inttohex(Simboard3.ControlWord,2);
 end;
end;
```

รูปที่ 3.79 (ต่อ) ฟังก์ชันที่ใช้ในการแสดงผลของคอมโปเนนต์ในอินเตอร์เฟส 3

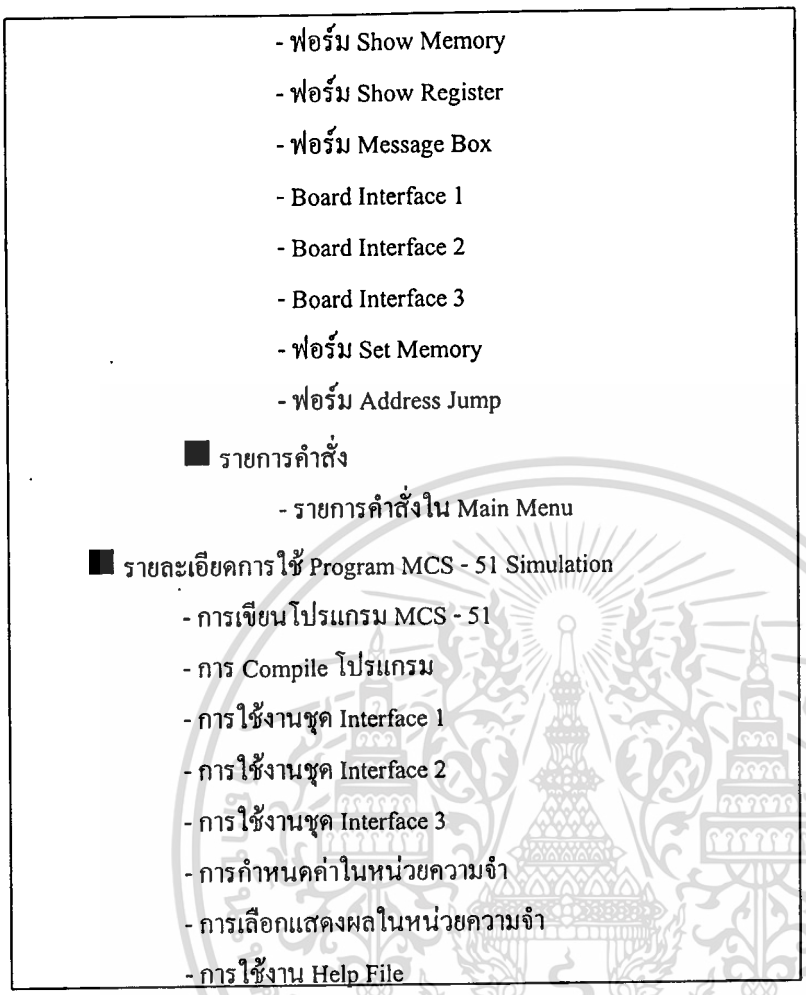
3.15 การสร้าง Help File

3.15.1 ขั้นตอนการสร้าง Help File

1. กำหนดรูปแบบของหัวข้อหลักและย่อย ดังรูปที่ 3.80

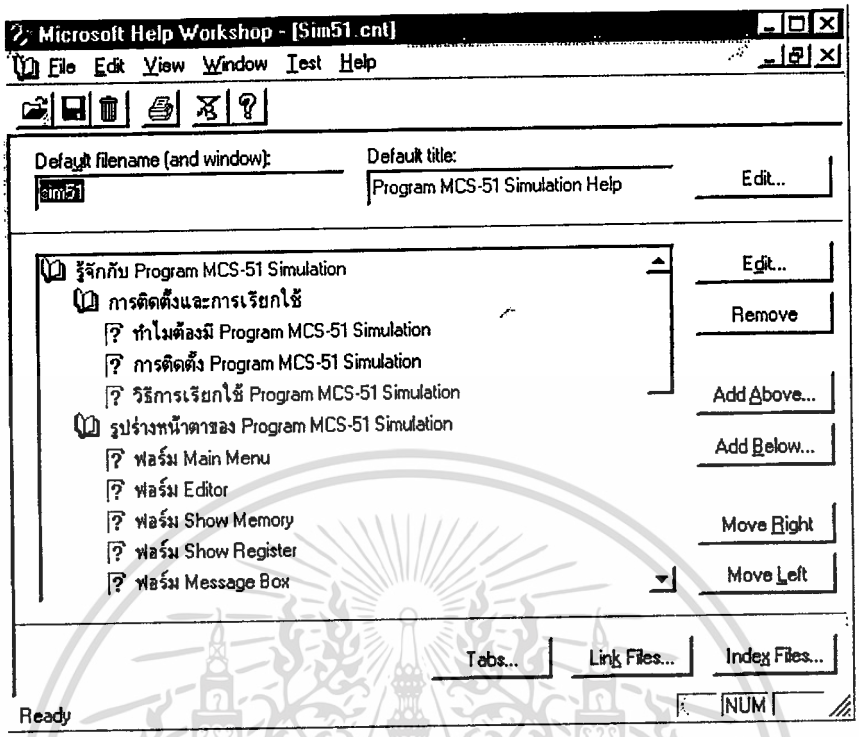


รูปที่ 3.80 รูปแบบของหัวข้อหลักและย่อย



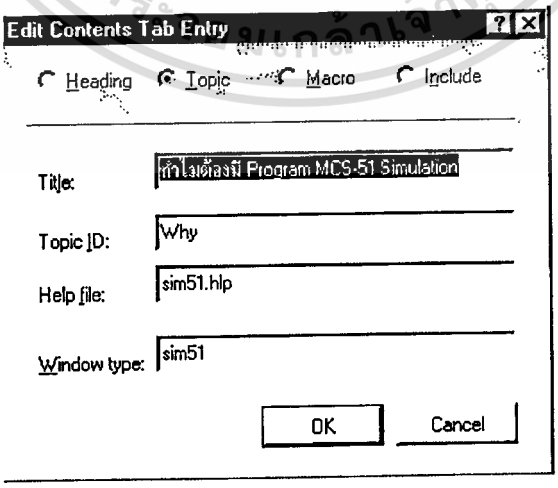
รูปที่ 3.80 (ต่อ) รูปแบบของหัวข้อหลักและย่อย

2. สร้างไฟล์ Help Contents โดยกำหนดหัวข้อหลักและย่อยตามข้อ 1. เริ่มจากเลือกไปที่ File New และเลือก Help Contents จะได้ดังรูปที่ 3.81 ตรงส่วนนี้จะเป็นการกำหนดชื่อไฟล์ Contents และ Title ที่แสดงขณะที่เปิด Help ขึ้นมา ส่วนด้านล่างจะเป็นรายการลำดับข้อมูลของหัวข้อหลักและย่อย และปุ่มทางด้านขวามือ จะเป็นตัวควบคุมลำดับก่อนหลังของหัวข้อรายการ หลังจากที่ได้กำหนดชื่อไฟล์และ Title เสร็จแล้วจึงทำการกำหนดลำดับของ Help โดยกดปุ่ม Edit อันที่สอง (อันล่าง)



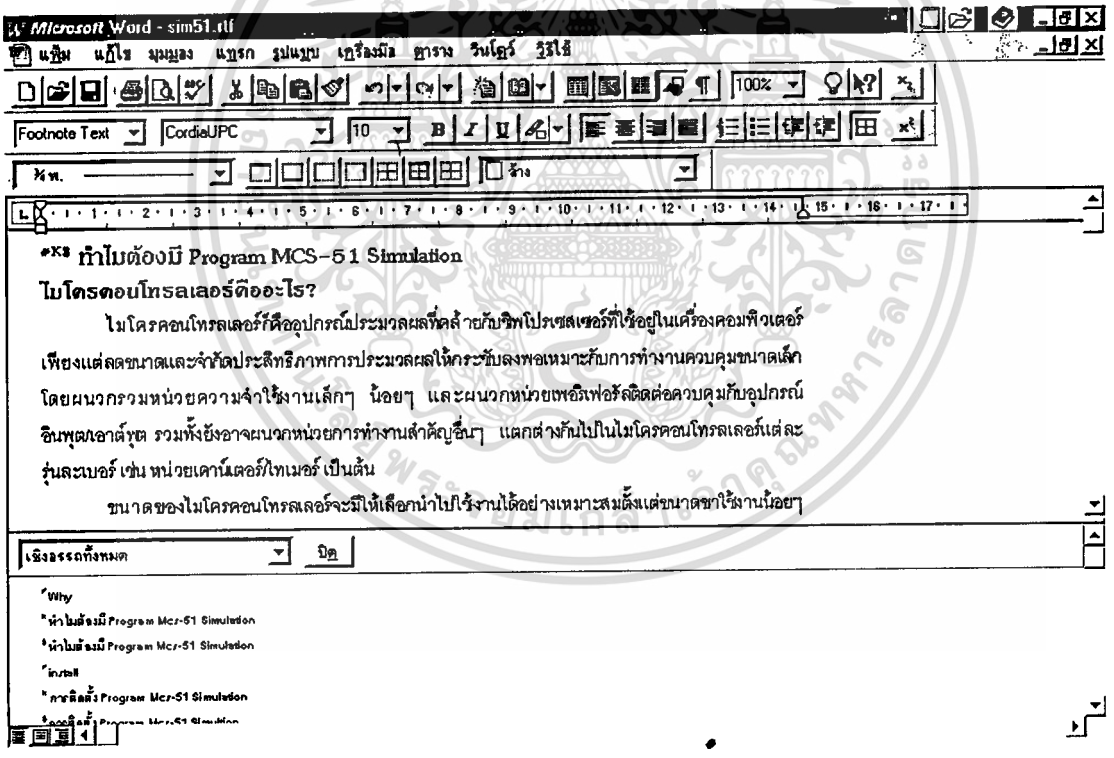
รูปที่ 3.81 การกำหนดรูปแบบของ Contents File

3. กำหนด Title และ Topic ID เพื่อที่จะระบุการเชื่อมไปยัง RTF (RICH TEXT FORMAT) ไฟล์ได้ โดยการระบุ Topic ID นั้นจะต้องพิจารณาถึงข้อกำหนดที่ได้กล่าวไว้แล้วในบทที่ 2 ในเรื่องของการสร้าง Help File ดังรูปที่ 3.82



รูปที่ 3.82 Edit Contents

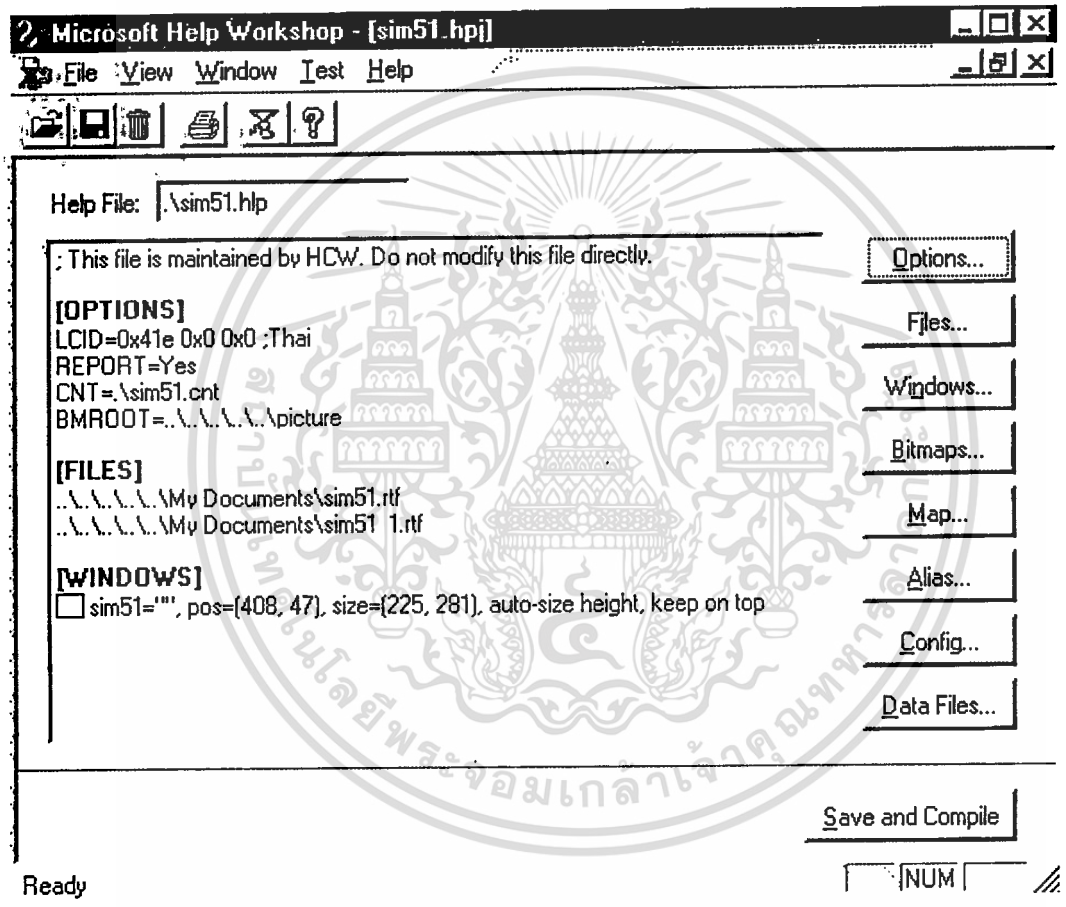
4. สร้างไฟล์ RTF โดยใช้โปรแกรม Microsoft Word ตั้งแต่ version 6 ขึ้นไป เริ่มจากการเปิดเอกสารเปล่าๆ ขึ้นมาหนึ่งเอกสาร เลือกไปที่เมนู View/Normal สำหรับกรณีที่อยู่ในโหมด Page Layout หรือ Outline เราจะมาเริ่มสร้าง Topic ID กัน การใช้ MS - Word ในการสร้างเอกสาร Help นี้ จะใช้ Footnote และ Page Break ร่วมกันในการกำหนด Topic และจุดสิ้นสุด จากนั้นทำการสร้าง Topic ID โดยเลือกไปที่ Insert/Footnote จากนั้นเลือกไปที่ Custom Mark เพื่อที่จะใส่รูปแบบและลำดับ Topic ID โดยจะต้องใส่เครื่องหมายต่างๆ ตามความต้องการในการสร้าง Help ตามต้องการ หลังจากที่ได้เพิ่ม Footnote (Topic ID) เรียบร้อยแล้ว จากนั้นก็ทำการพิมพ์ข้อความที่ต้องการได้ทันที และเมื่อพิมพ์ข้อมูลในส่วนของรายละเอียดต่างๆ เสร็จแล้ว ก็เลือกไปที่ Insert/Break เพื่อที่จะใส่ Page Break เพื่อระบุการจบข้อมูลในส่วนนี้ จากวิธีการที่ได้กล่าวมาข้างต้นสามารถนำไปใช้ในการเขียน Help File ในหัวข้ออื่นๆ ได้เช่นเดียวกัน ดังรูปที่ 3.83



รูปที่ 3.83 ตัวอย่างเอกสารนามสกุล RTF

5. การบันทึก เมื่อทำการพิมพ์ทุกอย่างเรียบร้อยแล้วก็ทำการบันทึกข้อมูล โดยเลือกไปที่ File/Save As เลือกไฟล์ Type ไปที่ Save As Type ไปที่ Rich Text Format และตั้งชื่อไฟล์

6. การสร้าง Help Project กลับมาที่ Microsoft Help Workshop ทำการสร้าง Help Project ขึ้นมา โดยตัวนี้จะเป็นตัวระบุรูปแบบของไฟล์ Help เช่นนำข้อมูลมาจาก RTF ไฟล์อะไร มีรูปแบบอะไรบ้าง เลือกไปที่ File/New/Help Project และใส่ชื่อไฟล์ลงไป ดังรูปที่ 3.84 ส่วนปุ่มทางด้านขวามือเป็นปุ่มที่ใช้ในการกำหนดค่าต่างๆ เช่น ที่ปุ่ม File เป็นตัวระบุว่าจะใช้ RTF ไฟล์อะไรในการนำมาผูก

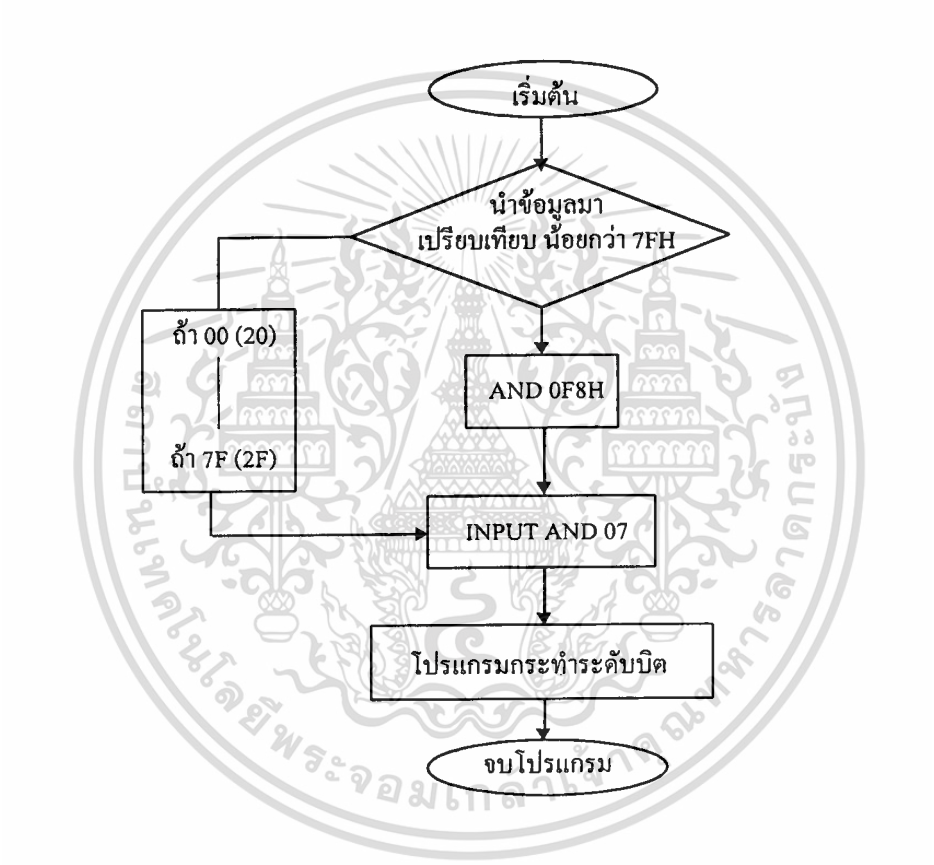


รูปที่ 3.84 การควบคุม Project File

7. การ Compiler ขั้นตอนต่อไปคือการคอมไพล์โปรแกรมเพื่อสร้างเป็นไฟล์ *.HLP ซึ่งจะเป็นไฟล์ที่สมบูรณ์ที่สามารถนำไปใช้แสดงข้อมูลได้ การคอมไพล์ทำได้โดยการเลือกไปที่ File/Compile หลังจากที่คอมไพล์เรียบร้อยแล้ว ตัวคอมไพล์จะแจ้งข้อความว่ามีการทำอะไรไปบ้างใน Help File เช่น การแสดงจำนวนของ Topic ID เป็นต้น

3.16 ฟังก์ชันในการกระทำระดับบิต

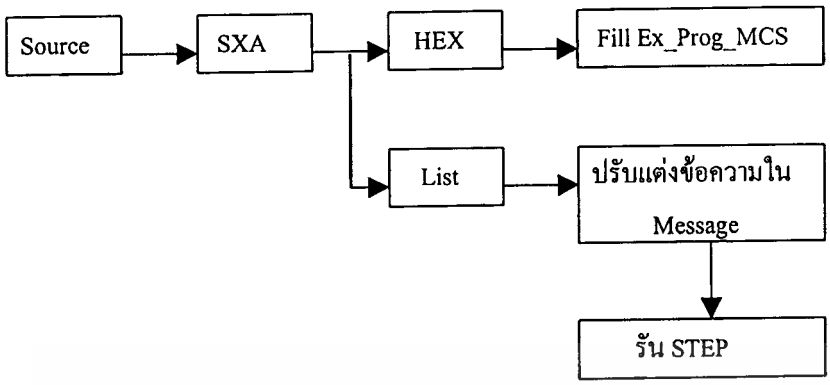
เป็นฟังก์ชันที่ช่วยในการคำนวณบิตแอดเดรสของโปรแกรม โดยใช้การเปรียบเทียบค่าบิตแอดเดรสก่อน 7F กับหลัง 7F โดยที่ก่อน 7F จะอยู่ช่วงแอดเดรส 20 ถึง 2F และหลัง 7F จะอยู่ตั้งแต่ 80ถึง FF กระจายกันอยู่ ซึ่งสามารถแยกและหาค่าได้โดยการ นำเอาส่วนของ 4 บิตบนไปเปรียบเทียบกับตำแหน่งของหน่วยความจำ Internal Data Memory และใช้ 4 บิตล่างในการเช็คว่าอยู่ในตำแหน่งบิตใด ดังรูปที่ 3.85



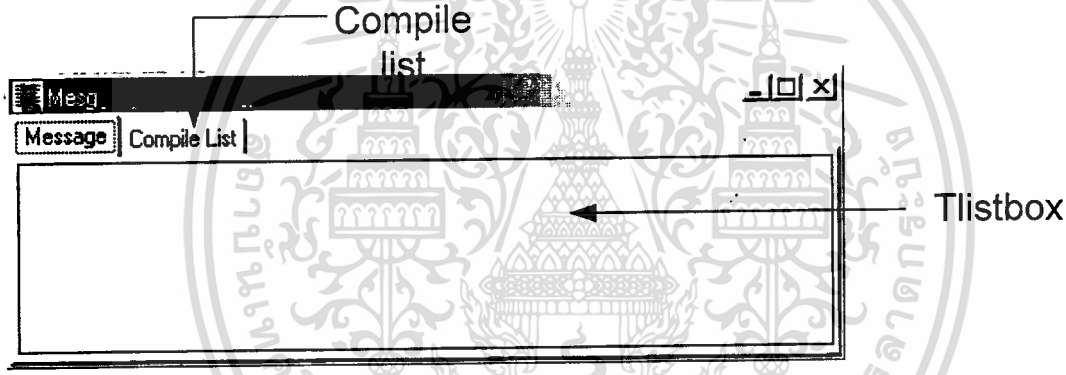
รูปที่ 3.85 กระบวนการทำงานของฟังก์ชันบิตแอดเดรส

3.17 การสร้างฟอร์มเม็สเส็จในส่วนของ Compile List

ฟอร์มเม็สเส็จเป็นฟอร์มที่ใช้ในการแสดงข้อความการทำงานของโปรแกรม และใช้ในการแสดงการของคำสั่งเมื่อทำการ รันในโหมด STEP ปกติเมื่อ SXA51 compile โปรแกรมผลที่ได้จะอยู่ในรูปไฟล์ 2 ตัวคือ Hex file format กับ List File ในการสร้างฟอร์มเม็สเส็จได้ใช้ในส่วนของ List file มาแสดงผล ลงบนคอมโปเนนต์ Tlistbox โดยคุณลักษณะของคอมโปเนนต์ Tlistbox สามารถแสดงเป็นบรรทัดได้ ดังรูปที่ 3.86 และมีส่วนประกอบของฟอร์มดังรูปที่ 3.87

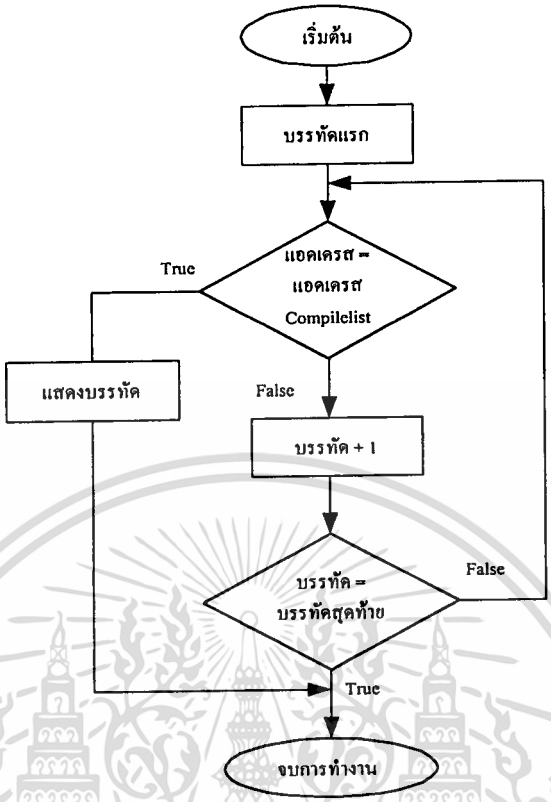


รูปที่ 3.86 การทำงานในการ Compile



รูปที่ 3.87 ส่วนประกอบของฟอร์มเม็สเส็จ Compile List

ในการหาว่าโปรแกรมทำงานอยู่ที่บรรทัดใด จะใช้การทำ เทรด แบบ Begin Thread ซึ่งเป็นเทรด ที่ไม่มีความซับซ้อนในการทำงาน ทำลูปวนรอบทุกบรรทัดของ Compile List และเช็คตัวอักษรตัวหน้า ซึ่งจะเป็นตัวบอกแอดเดรส และเมื่อพบก็ให้หยุดการทำลูป ดังรูปที่ 3.88 และสามารถเขียนโปรแกรมได้ดังรูปที่ 3.89



รูปที่ 3.88 การทำ เทรด ของ Compile List

```
procedure TMesg.StepSelect;
var ID:Dword;
begin
   BeginThread(nil,0,@Stepmesg,nil,0,ID);
end;
//end;
procedure Stepmesg;
var i:word;
   st,st2:string[100];
begin
sleep(100);
With Mesg do begin
for i:= 0 to (CompileList.Items.Count-1) do
   begin
      st:= CompileList.Items.strings[i];
      if not(st[10]='=') then
       begin
       st2:=Register_.Label22.Caption;
       if (st[6]+st[7]+st[8]+st[9] = st2[10]+st2[11]+st2
[12]+st2[13]) then
          begin
          CompileList.ItemIndex:=i;
          CompileList.Update;
          end;
       end;
    end;
  end;
end;
```

รูปที่ 3.89 ฟังก์ชันที่ใช้ตัดตัวอักษรใน list file

บทที่ 4

การทดลองและผลการทดลอง

4.1 กล่าวนำ

ในบทนี้กล่าวถึงวิธีทำการทดสอบชุดคำสั่งของไมโครคอนโทรลเลอร์ MCS-51 โดยแบ่งการทดสอบออกเป็นกลุ่มๆ คือ กลุ่มคำสั่งทางลอจิก, กลุ่มคำสั่งทางคณิตศาสตร์, กลุ่มคำสั่งควบคุมลำดับการทำงาน, กลุ่มคำสั่งการประมวลผลแบบบลูลีน

4.2 กลุ่มคำสั่งทางคณิตศาสตร์

4.2.1 การทดสอบคำสั่ง ADD

คำสั่ง ADD เป็นคำสั่งที่มีผลต่อแฟล็ก ซึ่งประกอบด้วย C, OV, AC, P คำสั่ง ADD ประกอบไปด้วย

1. ADD A,#n เป็นการบวกระหว่าง A กับค่าคงที่
2. ADD A,Rn เป็นการบวกระหว่าง A กับค่าใน Rn
3. ADD A,add เป็นการบวกระหว่าง A กับค่าที่แอดเดรส
4. ADD A,@Rn เป็นการบวกระหว่าง A กับค่าที่ Rn ชี้

1. ทดสอบคำสั่ง ADD A,#n

ทดสอบคำสั่ง ADD A,#n ด้วยการป้อนโปรแกรมดังรูปที่ 4.1 โดยโปรแกรมจะเก็บค่ารีจิสเตอร์ A ไว้ที่ตำแหน่ง 10H ถึง ตำแหน่ง 1EH และเก็บค่า PSW ไว้ที่ตำแหน่ง 20H ถึง 2FH

```
data1 equ 67h
org 000h
        mov r0,#20h
        mov r1,#10h
        mov r2,#0fh
test:   add a,#data1
        acall test2
        djnz r2,test
        sjmp $
test2:  mov @r0,psw
        mov @r1,a
        inc r1
        inc r0
        ret
end
```


รูปที่ 4.1 โปรแกรมทดสอบ ADD A,#n

ผลการทดสอบคำสั่ง ADD A,#n

1. ผลการทดสอบเมื่อรันด้วยโปรแกรม Sim51 ที่ถูกต้องโปรแกรมตำแหน่ง 10 ถึง 2F มีค่าดังรูปที่ 4.2

```
Int Ram ON 0010 67 CE 35 9C 03 6A D1 38
           0018 9F 06 6D D4 3B A2 09 00
           0020 01 05 C0 04 C0 00 44 81
           0028 04 C0 01 44 81 45 80 00
```

รูปที่ 4.2 ผลการรันคำสั่ง ADD A,#n ของ sim51

2. ผลการทดสอบเมื่อรันด้วยโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS-51 ดังรูปที่ 4.3

รูปที่ 4.3 ผลการรันคำสั่ง ADD A,#n ของ Mcs51Sim

2. ทดสอบคำสั่ง ADD A,Rn

โดยโปรแกรมจะเก็บค่ารีจิสเตอร์ A ไว้ที่ตำแหน่ง 10H ถึง ตำแหน่ง 1EH และเก็บค่า PSW ไว้ที่ตำแหน่ง 20H ถึง 2FH ใช้โปรแกรมทดสอบ ADD A,#n จากรูปที่ 4.1แก้ไขโปรแกรมในบรรทัดดังรูปที่ 4.4

```
        mov r2,#0fh
        mov r3,#67h
test:   add a,r3
        acall test2
```

รูปที่ 4.4 โปรแกรมทดสอบคำสั่ง ADD A,Rn

ผลการทดสอบคำสั่ง ADD A,Rn

1. ผลการทดสอบเมื่อรันด้วยโปรแกรม Sim51 ที่ถูกต้องโปรแกรมตำแหน่ง 10 ถึง 2F มีค่าดังรูปที่ 4.5

รูปที่ 4.5 ผลการรันคำสั่ง ADD A,Rn ของ sim51

2. ผลการทดสอบเมื่อรันด้วยโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS-51 ดังรูปที่ 4.6

รูปที่ 4.6 ผลการรันคำสั่ง ADD A,Rn ของ Mcs51Sim

3. ทดสอบคำสั่ง ADD A,add

โดยโปรแกรมจะเก็บค่ารีจิสเตอร์ A ไว้ที่ตำแหน่ง 10H ถึง ตำแหน่ง 1EH และเก็บค่า PSW ไว้ที่ตำแหน่ง 20H ถึง 2FH ใช้โปรแกรมทดสอบ ADD A,#n แก้ไขโปรแกรมในบรรทัดดังรูปที่ 4.7

```
        mov r2,#0fh
        mov r3,#67h
test:   add a,03h
        acall test2
```

รูปที่ 4.7 โปรแกรมทดสอบคำสั่ง ADD A,add

ผลการทดสอบคำสั่ง ADD A,add

1. ผลการทดสอบเมื่อรันด้วยโปรแกรม Sim51 ที่ถูกต้องโปรแกรมตำแหน่ง 10 ถึง 2F มีค่าดังรูปที่ 4.8

รูปที่ 4.8 ผลการรันคำสั่ง ADD A,add ของ sim51

2. ผลการทดสอบเมื่อรันด้วยโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS-51 ดังรูปที่ 4.9

รูปที่ 4.9 ผลการรันคำสั่ง ADD A,add ของ Mcs51Sim

4. ทดสอบคำสั่ง ADD A,@Rn

ในการทดสอบคำสั่ง ADD A,@Rn ไม่จำเป็นต้องทดสอบเพราะชุดคำสั่งในโปรแกรมที่เขียนขึ้นใช้แบบเดียวกับ ADD A,add เพียงแต่เปลี่ยนโปรแกรมภายในเล็กน้อย ผลที่ได้จึงถูกต้องเหมือนกัน

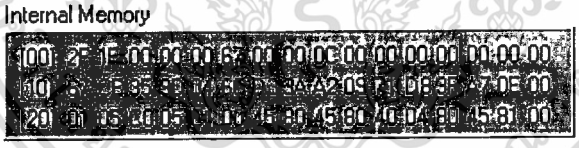
4.2.2 การทดสอบคำสั่ง ADDC

ในการทดสอบจะใช้โปรแกรม ADD นำมาแก้ไขโดยเปลี่ยนคำสั่ง ADD เป็น ADDC ซึ่งโปรแกรมที่ได้จะเป็นดังรูปที่ 4.10 ผลการทดสอบเมื่อนำโปรแกรมไปเทียบกับ sim51 ได้ดังรูปที่ 4.11 กับรูปที่ 4.12

```
data1 equ 67h
org 000h
        mov r0,#20h
        mov r1,#10h
        mov r2,#0fh
        mov r5,#67h
test:   addc a,05h
        acall test2
        djnz r2,test
        sjmp $
test2:  mov @r0,psw
        mov @r1,a
        inc r1
        inc r0
        ret
end
```

รูปที่ 4.10 โปรแกรมทดสอบคำสั่ง ADDC

รูปที่ 4.11 ผลการรันคำสั่ง ADDC ของ sim51



รูปที่ 4.12 ผลการรันคำสั่ง ADDCของ Mcs51Sim

ผลการทดสอบคำสั่ง SUBB

ผลการทำงานของคำสั่ง ADDC A,#n , ADDC A,Rn , ADDC A,@Rn และ ADD A,add ผลการทำงานที่ได้จะเป็นดังรูปที่ 4.12 เมื่อนำไปเทียบกับ ผลการทำงานของ Sim51 ซึ่งได้ผลดังรูปที่ 4.11 ผลที่ได้มีค่าเท่ากัน

4.2.3 การทดสอบคำสั่ง SUBB

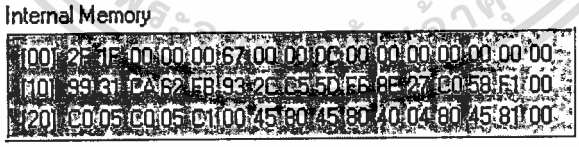
ในการทดสอบคำสั่ง SUBB จะใช้โปรแกรม ADD มาแก้ไขในบรรทัด ADD เป็น SUBB ซึ่งเป็นดังรูปที่ 4.13 ผลการทดสอบโปรแกรมนำไปเทียบกับ sim51 ได้ดังนี้

```
data1 equ 67h
org 0000h
        mov r0,#20h
        mov r1,#10h
        mov r2,#0fh
        mov r5,#67h
test:   subb a,05h
        acall test2
        djnz r2,test
        sjmp $
test2:  mov @r0,psw
        mov @r1,a
        inc r1
        inc r0
        ret
```

รูปที่ 4.13 โปรแกรมทดสอบคำสั่ง SUBB

ผลการทดสอบคำสั่ง SUBB

รูปที่ 4.14 ผลการรันคำสั่ง SUBB ของ sim51



รูปที่ 4.15 ผลการรันคำสั่ง SUBB ของ Mcs51Sim

ผลการทดสอบคำสั่ง SUBB A,#data , SUBB A,Rn , SUBB A,add และ SUBB A,@rn ผลการทำงานของโปรแกรม Sim51 เป็นดังรูปที่ 4.14 ส่วนผลการทำงานของโปรแกรม Mcs51Sim เป็นดังรูปที่ 4.15 เมื่อเทียบผลการทำงานจากรูปทั้งสองมีค่าเท่ากัน

4.2.4 การทดสอบคำสั่ง INC กับ DEC

คำสั่ง INC เป็นคำสั่งที่ใช้เพิ่มค่าให้กับรีจิสเตอร์ โดยเฉพาะ รีจิสเตอร์ A จะต้องคิด parity ส่วนคำสั่ง DEC เป็นคำสั่งทีใช้ในการลดค่า รีจิสเตอร์ A จะต้องคิด parity เช่นกัน การทดสอบจะเป็นการทดสอบการเพิ่มและลดค่าในแต่ละรีจิสเตอร์ โปรแกรมเป็นดังรูปที่ 4.16

```
org 000h
test1:  inc a
        acall test2
        setb psw.4
        acall test2
        setb psw.3
        acall test2
        clr psw.3
        clr psw.4
        sjmp test1
test2:
        inc dptr
        inc r0
        inc r1
        inc 30h
        inc 31h
        ret
end;
org 000h
test1:  dec a
        acall test2
        setb psw.4
        acall test2
        setb psw.3
        acall test2
        clr psw.3
        clr psw.4
        sjmp test1
test2:
        dec r0
        dec r1
        dec 30h
        dec 31h
        ret
end;
```

รูปที่ 4.16 โปรแกรมทดสอบคำสั่ง INC กับ DEC

ผลการทดสอบคำสั่ง INC กับ DEC

เมื่อสั่งให้โปรแกรมทำงาน ค่าของรีจิสเตอร์ R0 , R1 ,DPTR, ACC มีค่าเปลี่ยนแปลงเพิ่มขึ้นและลดลง

4.2.5 การทดสอบคำสั่ง MUL

คำสั่ง MUL เป็นคำสั่งคูณข้อมูลระหว่างรีจิสเตอร์ A กับ B เก็บค่าในรีจิสเตอร์ A พร้อมกับคิด flag ซึ่งประกอบด้วย CY,P,OV โปรแกรมที่ใช้ทดสอบการทำงานเป็นดังรูปที่ 4.17

```
org 000h
        mov r0,#20h
        mov r1,#10h
        mov r2,#10h
        mov r3,#67h
test:   add a,#89h
        mov b,r3
        mul ab
        acall test2
        djnz r2,test
        sjmp $
test2:  mov @r0,psw
        mov @r1,a
        inc r1
        inc r0
        ret
end
```

รูปที่ 4.17 โปรแกรมทดสอบคำสั่ง MUL AB

ผลการทดสอบคำสั่ง MUL

รูปที่ 4.18 ผลการรันคำสั่ง MUL ของ sim51

รูปที่ 4.19 ผลการรันคำสั่ง MUL ของ Mcs51Sim

ผลคูณที่ได้ รีจิสเตอร์ A แสดงอยู่ที่ตำแหน่ง 10 ถึง 1F และค่าของ PSW อยู่ที่ตำแหน่ง 20 ถึง 2F ผลที่ได้ของโปรแกรม Sim51 เป็นดังรูปที่ 4.18 และผลที่ได้จากการทำงานของ Mcs51Sim เป็นดังรูปที่ 4.19

4.3 กลุ่มคำสั่งทางตรรกศาสตร์

4.3.1 การทดสอบคำสั่ง ANL

คำสั่ง ANL เป็นคำสั่งที่ใช้ในการ AND ข้อมูล มีผลกับ Parity เมื่อ AND กับรีจิสเตอร์ A

ANL A,#n	And ข้อมูลระหว่าง A กับ #n
ANL A,Rn	And ข้อมูลระหว่าง A กับ Rn
ANL A,add	And ข้อมูลระหว่าง A กับ address
ANL A,@Rn	And ข้อมูลระหว่าง A กับ @Rn
ANL dataadd,#n	And ข้อมูลระหว่าง Dataaddress กับ #n
ANL dataadd,A	And ข้อมูลระหว่าง Dataaddress กับ A
ANL C,bitadd	And ข้อมูลระหว่าง C กับ bitaddress
ANL C,/bitadd	And ข้อมูลระหว่าง C กับ not bitaddress

ทดสอบคำสั่ง ANL

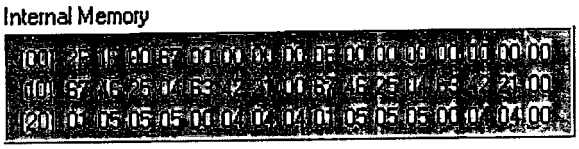
โปรแกรมที่ใช้ในการทดสอบเป็นโปรแกรม ADD แต่เพิ่มบรรทัด ANL A,#n , ANL A,R3 , ANL A,03hเข้าไป โปรแกรมเป็นดังรูปที่ 4.20

```
data1 equ 67h
org 000h
        mov r0,#20h
        mov r1,#10h
        mov r2,#0fh
        mov r3,#67h
test:   add a,03h
        anl a,#67h
        acall test2
        djnz r2,test
        sjmp $
test2;  mov @r0,psw
        mov @r1,a
        inc r1
        inc r0
        ret
end
```

รูปที่ 4.20 โปรแกรมทดสอบคำสั่ง ANL

```
Int Ram ON 0010 67 46 25 04 63 42 21 00
           0018 67 46 25 04 63 42 21 00
           0020 01 05 05 05 00 04 04 04
           0028 01 05 05 05 00 04 04 00
```

รูปที่ 4.21 ผลการรันคำสั่ง ANL ของ sim51



รูปที่ 4.22 ผลการทำงานคำสั่ง ANL ของ Mcs51Sim

ผลการทดสอบคำสั่ง ANL

จากรูปที่ 4.21 เป็นผลการทำงานของโปรแกรม sim51 เมื่อโปรแกรมเป็นดังรูปที่ 4.20 และรูปที่ 4.22 เป็นผลการทำงานจากโปรแกรมรูปที่ 4.20 เช่นกัน เมื่อนำรูปที่ 4.21 กับรูปที่ 4.22 มาเปรียบเทียบกัน พบว่ามีค่าเท่ากัน

ทดสอบคำสั่ง ANL C,bitaddress

คำสั่ง ANL เป็นคำสั่งที่มีการทำงานเป็นระดับบิตด้วย ดังนั้นจึงต้องทดสอบคำสั่งด้วย โดยโปรแกรมเป็นดังรูปที่ 4.23

```
        org 000h
test:   setb 0d7h
        anl c,/97h
        setb 0d7h
        setb 96h
        anl c,96h
        sjmp test
        end
```

รูปที่ 4.23 โปรแกรมทดสอบคำสั่ง ANL C,bitaddress

ผลการทดสอบคำสั่ง ANL C,bitaddress

ผลการทำงานของโปรแกรม ในช่วงแรกโปรแกรมจะ AND NOT กับบิตแอดเดรส 97H ซึ่ง C มีค่าเป็น 1 และเมื่อถึงบรรทัดที่ AND กับ 96H ผลที่ได้ C มีค่าเป็น 0 ซึ่งทำงานได้ถูก

4.3.2 การทดสอบคำสั่ง ORL

การทดสอบโดยการใช้โปรแกรมจากรูปที่ 4.20 นำมาแก้ไขโดยเปลี่ยนคำสั่ง ANL เป็นคำสั่ง ORL ซึ่งโปรแกรมเป็นดังรูปที่ 4.24

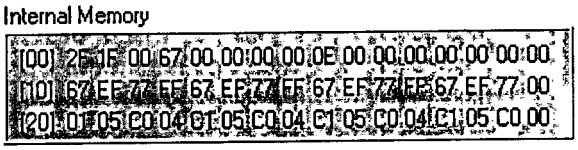
```
org 000h
        mov r0,#20h
        mov r1,#10h
        mov r2,#0fh
        mov r3,#67h
test:   add a,03h
        orl a,03
        acall test2
        djnz r2,test
        sjmp $
test2:  mov @r0,psw
        mov @r1,a
        inc r1
        inc r0
        ret
end
```

รูปที่ 4.24 โปรแกรททดสอบคำสั่ง ORL

ผลการทดสอบคำสั่ง ORL

ถ้าเกิด Error โปรแกรมจะไปเริ่มที่ตำแหน่ง 000hใหม่ ผลการทดสอบเมื่อเทียบกับโปรแกรม sim51 เป็นดังรูปที่ 4.25 และรูปที่ 4.26

รูปที่ 4.25 ผลการรันคำสั่ง ORL ของ sim51



รูปที่ 4.26 ผลการรันคำสั่ง ORL ของ Mcs51Sim

4.3.3 การทดสอบคำสั่ง XRL

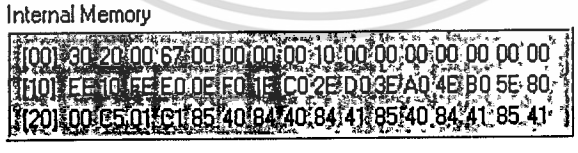
โปรแกรมทดสอบคำสั่ง XRL เป็นโปรแกรมที่นำมาจากรูปที่ 4.24 ซึ่งเป็นโปรแกรมทดสอบคำสั่ง ORL นำมาแก้ไขให้สามารถทดสอบคำสั่ง XRL ได้ โปรแกรมดังรูปที่ 4.27

```
org 000h
        mov r0,#20h
        mov r1,#10h
        mov r2,#10h
        mov r3,#67h
test:   add a,#89h
        mov b,r3
        xrl a,b
        acall test2
        djnz r2,test
        sjmp $
test2:  mov @r0,psw
        mov @r1,a
        inc r1
        inc r0
        ret
end
```

รูปที่ 4.27 โปรแกรมทดสอบคำสั่ง XRL

ผลการทดสอบ คำสั่ง XRL

รูปที่ 4.28 ผลการรันคำสั่ง XRL ของ sim51



รูปที่ 4.29 ผลการรันคำสั่ง XRL ของ Mcs51Sim

ผลการทำงานของโปรแกรมเมื่อป้อนโปรแกรมเข้าในโปรแกรมจำลอง Sim51 ผลที่ได้เป็นดังรปที่ 4.28 และผลการทำงานของโปรแกรม Mcs51Sim เป็นดังรูปที่ 4.29 เมื่อนำผลการทำงานจากโปรแกรมทั้งสองมาเปรียบเทียบกัน พบว่าค่าที่ได้มีค่าเท่ากัน

4.3.4 การทดสอบคำสั่ง CLR กับ CPL

คำสั่ง CLR กับ CLP เป็นคำสั่งที่ใช้กระทำกับบิตแอดเดรส ดังนั้นโปรแกรมจะต้องสุ่มการทำงานตามตำแหน่งต่างๆในบิตแอดเดรส โปรแกรมเป็นดังรูปที่ 4.30

```
org 000h
        mov a,#15h
        clr a
        setb psw.7
        clr c
        mov 20h,#01h
        mov 2fh,#80h
        mov 80h,#01h
        mov 0f8h,#80h
        clr 00h
        clr 7fh
        clr 80h
        clr 0ffh
        cpl c
        cpl c
        cpl 00h
        cpl 7fh
        cpl 80h
        cpl 0ffh
        sjmp $
end
```

รูปที่ 4.30 โปรแกรมทดสอบคำสั่ง CLR กับ CPL

ผลการทดสอบคำสั่ง CLR กับ CPL

เมื่อสั่งให้โปรแกรมทำงานในตำแหน่งต่างๆบิตแอดเดรส สามารถเซ็ตและเคลียร์บิตในตำแหน่งนั้นๆได้

4.3.5 การทดสอบคำสั่งในการหมุน

```
org 00h
test3:  mov psw,#80h
        mov r0,#7h
test1:  rlc a
        mov p1,a
        djnz r0,test1
        mov psw,#80h
        mov r0,#7h
test2:  rrc a
        mov p1,a
        djnz r0,test2
        sjmp test3
end
        org 00h
test3:  mov a,#01h
        mov r0,#7h
test1:  rl a
        mov p1,a
        djnz r0,test1
        mov a,#80h
        mov r0,#7h
test2:  rr a
        mov p1,a
        djnz r0,test2
        sjmp test3
```


รูปที่ 4.31 โปรแกรมทดสอบคำสั่งหมุนข้อมูล

ผลการทดสอบการหมุน

เมื่อสั่งให้โปรแกรมในรูปที่ 4.31 ซึ่งเป็นโปรแกรมทดสอบคำสั่งหมุนข้อมูล ผลที่ได้เมื่อโปรแกรมทำงาน ของคำสั่ง RR A , RL A , RRC A และคำสั่ง RLC A ลักษณะและทิศทางการหมุนเป็นไปตามรูปแบบลักษณะของคำสั่ง

4.3.6 การทดสอบคำสั่ง SWAP

```
        org 000h
test:   mov a,#15h
        swap a
        mov a,#39h
        swap a
        sjmp test
end
```

รูปที่ 4.32 โปรแกรมทดสอบคำสั่ง SWAP

ผลการทดสอบคำสั่ง SWAP

จากโปรแกรมในรูปที่ 4.32 ผลคือ สามารถสลับข้อมูลระหว่าง 4 บิตบนกับ 4 บิตล่างได้

4.4 กลุ่มคำสั่งควบคุมลำดับการทำงาน

4.4.1 การทดสอบคำสั่ง ACALL

คำสั่ง ACALL เป็นคำสั่งที่ ในเรียกโปรแกรมย่อย ในตำแหน่งต่างๆ พร้อมกับเก็บค่า PC เอาไว้ในสเต็ก โปรแกรมเป็นโปรแกรมขนาด 2 ไบท์ ในช่วงไบท์แรกบิตที่ 5 ถึงบิตที่ 7 จะเป็นตัวเก็บตำแหน่งแอดเรส A8 ถึง A10 และแอดเดรส A0 ถึง A7 จะถูกเก็บไว้ที่ไบท์ตัวที่ 2 คำสั่ง ACALL มีทั้งหมด 8 คำสั่ง หนึ่งคำสั่งกระโดดได้ 2 K คำสั่ง ACALL เป็นคำสั่งที่ไม่ผลต่อ flag

การทดสอบโดยการกระโดดไปทางด้านบวก

```
data1 equ  000h
data2 equ 7ffh
org data1
test1:    acall     test2
          inc r0
          sjmp test1
org data2
test2:    inc a
          ret
end
```

รูปที่ 4.33 โปรแกรมทดสอบคำสั่ง ACALL ช่วงกระโดดขึ้น

ตารางที่ 4.1 ข้อมูลใน Equ data1 data2 ที่ใช้ทดสอบ ACALL

	11 n	31 n	51 n	71 n	91 n	B1 n	D1 n	F1 n
Data1	000h	000h	000h	000h	000h	000h	000h	000h
Data2	0ffh	1ffh	2ffh	3ffh	4ffh	5ffh	6ffh	7ffh

จากโปรแกรมที่ 4.33 จะนำค่า Data1 กับ Data2 ซึ่งได้จากตารางที่ 4.1 แทนค่าลงไป แล้วทดสอบการทำงานของโปรแกรม

ผลการทดสอบ

การกระโดดไปในตำแหน่งต่างๆของหน่วยความ ทดสอบโดยการป้อนค่าจากตารางทั้ง 8 ตัวพบว่าสามารถกระโดดไปในตำแหน่งเพจต่างๆได้

การทดสอบโดยการกระโดดไปทางด้านลบ

```
data1 equ  07fch
data2 equ 05h
org 000h
        ljmp data1
org data2
test2:  inc a
        ret
org data1
test1:  acall   test2
        inc r0
        sjmp test1
end
```

รูปที่ 4.34 โปรแกรมทดสอบคำสั่ง ACALL ช่วงกระโดดลง

ตารางที่ 4.2 ข้อมูลใน Equ data1 data2 ที่ใช้ทดสอบ ACALL

	11 n	31 n	51 n	71 n	91 n	B1 n	D1 n	F1 n
Data1	7fch	7fch	7fch	7fch	7fch	7fch	7fch	7fch
Data2	05h	100h	200h	300h	400h	500h	600h	700h

ผลการทดสอบ

จากโปรแกรมรูปที่ 4.34 ป้อนข้อมูลดังตารางที่ 4.2 ได้ผลคือ การกระโดดไปในตำแหน่งต่างๆ ในหน่วยความจำ 8 ตำแหน่งพบว่าสามารถกระโดดไปในตำแหน่งเพจต่างๆได้

การทดสอบโดยการกระโดดระหว่างท้าย Page ต่อท้าย Page

```
data1 equ  0fffh
data2 equ 17ffh
org 000h
        ljmp data1
org data1
test1:  acall   test2
        inc r0
        sjmp test1
org data2
test2:  inc a
        ret
end
```


รูปที่ 4.35 โปรแกรมทดสอบคำสั่ง ACALL ช่วงกระโดดท้ายเพจต่อท้ายเพจ

ผลการทดสอบ

จากโปรแกรมที่ 4.35 โปรแกรมสามารถกระโดดข้ามระหว่างท้าย Page 01(0fffh) ไปยังท้าย Page 02(17ffh) ได้

4.4.2 การทดสอบคำสั่ง AJMP

คำสั่ง AJMP เป็นคำสั่งที่กระโดดไปในตำแหน่งสัมพันธ์ใดๆ เหมือนกับ ACALL แต่ AJMP เป็นการกระโดดไปในส่วนต่างๆของโปรแกรมไม่ได้กระโดดไปในโปรแกรมย่อยเหมือนอย่าง ACALL ดังนั้นลักษณะในการกระโดดจึงเหมือนกัน

```
org 000h
test1:  ajmp test2
org 0ffh
test3:  ajmp test4
org 1ffh
test4:  ajmp test5
org 2ffh
test5:  ajmp test6
org 3ffh
test6:  ajmp test7
org 4ffh
test7:  ajmp test8
org 5ffh
test8:  ajmp test9
org 6ffh
test9:  ajmp test1
org 7fch
test2:  ajmp test3
end
```

รูปที่ 4.36 โปรแกรมทดสอบคำสั่ง AJMP

ผลการทดสอบโปรแกรมกระโดดไปตำแหน่งต่างๆ

โปรแกรมที่ 4.36 จะเริ่มต้นทำงานที่ตำแหน่ง 0000H แล้วกระโดดไปท้ายเพจ 0 คือตำแหน่ง 7FCH ซึ่งเป็นท้ายเพจ แล้วกระโดดไปในตำแหน่ง 0FFH ซึ่งเป็นท้ายเพจอีกเพจหนึ่ง กระโดดไปเลยๆจนวนกลับมาที่ 0000H ใหม่

000h -> 7fch -> 0ffh -> 1ffh -> 2ffh -> 3ffh -> 4ffh -> 5ffh -> 6ffh -> 000h ->.....

ทดสอบ AJMP โดยการกระโดดระหว่างท้าย Page ไปยังท้าย Page

เป็นการกระโดยจากท้าย Page01 ไปยังท้าย Page02 ,03,04,05,06 และ 07

```
org 000h
test1:  ajmp test2
org 7ffh
test2:  nop
        nop
        ajmp test3
org 0fffh
test3:  nop
        nop
        ajmp test4
org 17ffh
test4:  nop
        nop
        ajmp test5
org 1fffh
test5:  nop
        nop
        ajmp test6
org 27ffh
test6:  nop
        nop
        ajmp test7
org 2fffh
test7:  nop
        nop
        ajmp test8
org 37ffh
test8:  nop
        nop
        ajmp test9
org 3fffh
test9:  Ljmp test1
end
```

รูปที่ 4.37 โปรแกรมทดสอบคำสั่ง AJMP

ผลการทดสอบ

เมื่อสั่งให้โปรแกรมที่ 4.37 ทำงานแอดเดรสที่แสดงผล จะกระโดดระหว่างท้ายเพจตัวที่หนึ่ง ไปยังท้ายเพจตัวที่สองได้

4.4.3 การทดสอบคำสั่ง CJNE

คำสั่ง CJNE เป็นคำสั่งเปรียบเทียบ ข้อมูลระหว่างรีจิสเตอร์ 2 ตัวถ้ามีค่าเท่ากัน ให้กระโดดไปในตำแหน่งสัมพันธ์ แต่ถ้าไม่เท่ากันให้เช็ค CY เป็น 0 หรือ 1

ทดสอบคำสั่ง CJNE A,#data,codeaddress

เมื่อรันโปรแกรมจะเพิ่มค่าในรีจิสเตอร์ A จาก 0fch ไปจนถึง 04h ซึ่งค่ารีจิสเตอร์ A แสดงในช่วง 10h ถึง 17h และค่าของ pswจะเก็บไว้ในช่วง 20h ถึง 27h

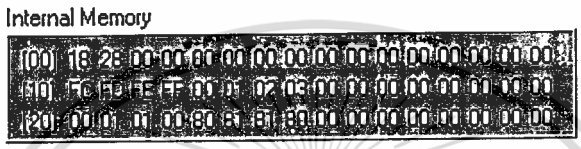
```
        org 000h
        mov r0,#10h
        mov r1,#20h
        mov a,#0fch
test1:  cjne a,#04h,test2
        inc a
        sjmp $
test2:  mov @r0,a
        mov @r1,psw
        inc a
        inc r0
        inc r1
        sjmp test1
end
```

รูปที่ 4.38 โปรแกรมทดสอบคำสั่ง CJNE

ผลการทดสอบคำสั่ง CJNE A,#data,codeaddress

```
Int Ram ON 0010 FC FD FE FF 00 01 02 03
           0018 67 46 25 04 63 42 21 00
           0020 00 01 01 00 80 81 81 80
           0028 01 05 05 05 00 04 04 00
```

รูปที่ 4.39 ผลการรันคำสั่ง CJNE ของ sim51



รูปที่ 4.40 ผลการรันคำสั่ง CJNE ของ Mcs51Sim

เมื่อพิมพ์โปรแกรมรูปที่ 4.38 แล้วสั่งให้ Sim51 จำลองการทำงานผลที่ได้เป็นดังรูปที่ 4.39 และเมื่อป้อนโปรแกรมให้ Mcs51Sim ผลที่ได้จะเป็นดังรูปที่ 4.40

4.4.4 การทดสอบคำสั่ง DJNZ

คำสั่ง DJNZ มี 2 แบบคือ DJNZ Rn,codeadd กับ DJNZ dataadd,codeadd ในการทดสอบจะใช้โปรแกรมดังรูปที่ 4.41 ตัวเดียวเพียงแต่ เปลี่ยนค่าของ R0 เป็น 0H ก็คือ dataaddress นั้นเอง

```
        org 000
test1:  mov r0,#0fh
test:   inc p0
        sjmp cont
org 30h
cont:   djnz  r0,test3
        inc p1
        sjmp test2
```

```
org 080h
test2:  djnz r0,test
        inc a
        ajmp test1
org 0b1h
test3:  inc p2
        ajmp test2
```

รูปที่ 4.41 โปรแกรมทดสอบคำสั่ง DJNZ

ผลการทดสอบ คำสั่ง DJNZ

คำสั่ง DJNZ Rn,codeadd กับ DJNZ dataadd,codeadd สามารถลบค่า R0 หรือ dataaddress ได้และสามารถกระโดดไปในตำแหน่งที่กำหนดได้

4.4.5 การทดสอบคำสั่ง JB กับ JNB

คำสั่ง JB เป็นคำสั่งกระโดดเมื่อ บิดแอดเดรสนั้นๆเป็น 1 ส่วน JNB กระโดดเมื่อบิตแอดเดรสนั้นเป็น 0

```
        org 000h
test1:  jnb  0h,test2
        clr 0h
        nop
        nop
test3:  sjmp test1
        org 082h
test2:  setb 0h
        jb 0h,test3
        end
```

รูปที่ 4.42 โปรแกรมทดสอบคำสั่ง JB กับ JNB

ผลการทดสอบคำสั่ง JB กับ JNB

เมื่อโปรแกรมรูปที่ 4.42 ทำงานมีผลคือโปรแกรมสามารถกระโดดเมื่อบิตแอดเดรสเป็น 1 หรือ 0 ได้ และกระโดดไปในตำแหน่งที่ระบุได้ระยะห่างช่วงสัมพันธ์เหมือนกับ SJMP

4.4.6 การทดสอบคำสั่ง JC กับ JNC

JC กับ JNC เป็นคำสั่งที่กระโดดเมื่อ carry flag เป็น 1 หรือ 0

```
org 000h                    org 081h
test1:  jnc  test2          test2:  setb psw.7
        clr psw.7                   nop
        nop                         jc test3
        nop                 end
test3:  sjmp test1
```

รูปที่ 4.43 โปรแกรมทดสอบคำสั่ง JC กับ JNC

ผลการทดสอบคำสั่ง JC กับ JNC

เมื่อโปรแกรมรูปที่ 4.43 ทำงานมีผลคือโปรแกรมสามารถกระโดดเมื่อ carry แฟล็ก เป็น 1 หรือ 0 ได้ และกระโดดไปในตำแหน่งที่ระบุได้ระยะห่างช่วงสัมพันธ์เหมือนกับ SJMP

4.4.7 การทดสอบคำสั่ง JZ กับ JNZ

JC กับ JNC เป็นคำสั่งที่กระโดดเมื่อ carry flag เป็น 1 หรือ 0

```
org 000h                    org 081h
test1:  jnc  test2          test2:  setb acc.0
        clr acc.0                   nop
        nop                         jc test3
        nop                 end
test3:  sjmp test1
```

รูปที่ 4.44 โปรแกรมทดสอบคำสั่ง JZ กับ JNZ

ผลการทดสอบคำสั่ง JZ กับ JNZ

เมื่อโปรแกรมรูปที่ 4.44 ทำงานมีผลคือโปรแกรมสามารถกระโดดเมื่อรีจิสเตอร์มีค่าเท่ากับ 0 หรือไม่เท่ากับ 0 ได้ และกระโดดไปในตำแหน่งที่ระบุได้ระยะห่างช่วงสัมพันธ์เหมือนกับ SJMP

4.4.8 การทดสอบคำสั่ง JBC

กระโดดไปในตำแหน่งสัมพันธ์ ถ้าบิตนั้น set และจะทำการ clear บิตนั้น โปรแกรมเป็นดังรูปที่ 4.45

```
        org 000h
test:   jbc psw.7,test1
test1:  nop
        setb psw.7
        nop
        sjmp test
        end
```

รูปที่ 4.45 โปรแกรมทดสอบคำสั่ง JBC

ผลการทดสอบคำสั่ง JBC

เมื่อโปรแกรมรูปที่ 4.45 ทำงานถึงคำสั่ง JBC ถ้าบิตนั้นเป็น 1 จะกระโดดไปตำแหน่งสัมพันธ์ และให้บิตนั้นเป็น 0

4.4.9 การทดสอบคำสั่ง LCALL กับ LJMP

```
org 000h
        ljmp test
org 010h
test1:  push acc
        inc r0
        inc r1
        pop acc
        ret
org 0fff0h
test:   inc a
        lcall test1
        sjmp test
end
```

รูปที่ 4.46 โปรแกรมทดสอบคำสั่ง LCALL กับ LJMP

ผลการทดสอบคำสั่ง LCALL กับ LJMP

โปรแกรมรูปที่ 4.46 คำสั่ง LCALL กับ LJMP สามารถทำงานกระโดดไปในตำแหน่งต่างๆ ในหน่วยความจำจำลองได้

4.4.10 การทดสอบคำสั่ง SJMP

```
        org 000h
        nop
        nop
kob1:   sjmp kob2
        mov a,#04h
        nop
kob3:   sjmp kob4
        org 030h
kob4:   mov a,#01h
        sjmp kob1

        org 083h
kob2:   mov a,#02h
        sjmp kob3
        end
```

รูปที่ 4.47 โปรแกรมทดสอบคำสั่ง SJMP

ผลการทดสอบคำสั่ง SJMP

โปรแกรมรูปที่ 4.47 สามารถกระโดดไปในตำแหน่งทางด้านบวกและลบได้

4.5 กลุ่มคำสั่งที่แบ่งตามวิธีการเข้าถึงข้อมูล

4.5.1 การทดสอบคำสั่ง XCHD

```
        org 000h
kob:    mov a,#15h
        mov r0,#10h
        mov r1,#11h
        mov @r0,#26h
        mov @r1,#27h
        xchd a,@r0
        mov p1,a
        xchd a,@r1
        mov p2,a
        sjmp kob
        end
```

รูปที่ 4.48 โปรแกรมทดสอบคำสั่ง XCHD

ผลการทดสอบคำสั่ง LCALL กับ LJMP

โปรแกรมรูปที่ 4.46 คำสั่ง LCALL กับ LJMP สามารถทำงานกระโดดไปในตำแหน่งต่างๆ ในหน่วยความจำจำลองได้

4.4.10 การทดสอบคำสั่ง SJMP

```
        org 000h
        nop
        nop
kob1:   sjmp kob2
        mov a,#04h
        nop
kob3:   sjmp kob4
        org 030h
kob4:   mov a,#01h
        sjmp kob1

        org 083h
kob2:   mov a,#02h
        sjmp kob3

        end
```

รูปที่ 4.47 โปรแกรมทดสอบคำสั่ง SJMP

ผลการทดสอบคำสั่ง SJMP

โปรแกรมรูปที่ 4.47 สามารถกระโดดไปในตำแหน่งทางด้านบวกและลบได้

4.5 กลุ่มคำสั่งที่แบ่งตามวิธีการเข้าถึงข้อมูล

4.5.1 การทดสอบคำสั่ง XCHD

```
        org 000h
kob:    mov a,#15h
        mov r0,#10h
        mov r1,#11h
        mov @r0,#26h
        mov @r1,#27h
        xchd a,@r0
        mov p1,a
        xchd a,@r1
        mov p2,a
        sjmp kob
        end
```

รูปที่ 4.48 โปรแกรมทดสอบคำสั่ง XCHD

บทที่ 5
บทสรุป ปัญหา แนวทางแก้ไขและพัฒนา

5.1 บทสรุป

ก่อนที่โครงงานนี้จะสำเร็จได้นั้นจำเป็นอย่างยิ่งที่จะต้องได้รับความช่วยเหลือจากท่านอาจารย์ในภาควิชา และบุคคลอื่นๆ ที่ผู้จัดทำได้รับความช่วยเหลือและข้อแนะนำต่างๆ ที่เป็นประโยชน์ต่อผู้จัดทำในการจัดทำโครงงานนี้ขึ้นมา นอกจากนั้นการศึกษา ค้นคว้า ในหลักการณ์และทฤษฎีต่างๆ ที่เกี่ยวข้องกับการทำโครงงานนี้ทำให้ผู้จัดทำได้ทราบถึงความรู้ต่างๆ อาจจะหาไม่ได้จากในห้องเรียน และจากการทำโครงงานนี้เองทำให้ผู้จัดทำมีความเห็นว่าการเรียนนั้นนอกจากที่เราจะเรียนภายในห้องเรียนเท่านั้นแล้ว เรายังต้องศึกษา ค้นคว้า จากภายนอกอีกด้วย และอีกสิ่งหนึ่งที่มีความสำคัญก็คือการได้เห็นว่าสิ่งที่เราได้เรียนไปนั้นสามารถที่จะนำไปใช้ประโยชน์ได้จริงๆ สำหรับโครงงานนี้ทางคณะผู้จัดทำได้มีการวางแผนในการจัดทำโครงงาน โดยมีระยะเวลาของแผนทั้งหมด 5 เดือน เริ่มตั้งแต่การรวบรวม ค้นคว้า หาเอกสารที่เกี่ยวข้อง ทั้งจากตำราภาษาไทย ภาษาอังกฤษ และจากวารสารต่างๆ และการขอคำปรึกษาจากอาจารย์ที่ปรึกษา ซึ่งได้วางแผนในการศึกษาไว้ประมาณ 2 สัปดาห์ จากนั้นก็เริ่มทำการวางแผนเกี่ยวกับการเขียนโปรแกรม การออกแบบหน้าจอของโปรแกรมในส่วนต่างๆ และเมื่อทราบส่วนประกอบต่างๆ แล้วก็จัดการเขียนโปรแกรมเป็นส่วนๆ โดยมีลักษณะของการทำงานเป็นแบบมัลติทาสกิง ซึ่งทำให้การพัฒนาในขั้นต่อไปสามารถทำได้ง่ายและมีประสิทธิภาพในการใช้งานดีกว่าแบบอื่นๆ และเมื่อเขียนโปรแกรมเป็นที่หน้าพอใจแล้วคือสามารถที่จะจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS-51 ได้แล้ว ก็ทำการตรวจสอบหาข้อผิดพลาดที่เกิดขึ้นจากตัวโปรแกรม โดยทำการทดสอบคำสั่งๆ ต่างที่สามารถเขียนโปรแกรมจำลองได้ ซึ่งผลการทดลองที่ออกมานั้นเป็นที่น่าพอใจว่า เราสามารถจำลองการทำงานของคำสั่ง ได้เกลือบทุกคำสั่ง มียกเว้นบ้างบางคำสั่ง เช่น คำสั่งในการอินเตอร์รัพ และหลังจากที่เวลาล่วงเลยมาประมาณ 3 เดือน เข้าสู่ช่วงของต้นเดือนที่ 4 โปรแกรมของเราได้มีความสมบูรณ์มากขึ้น แต่ทั้งนี้เราก็ยังไม่ได้ยุติเพียงเท่านั้นไม่ เรายังคงพัฒนาโปรแกรมต่อไปอย่างต่อเนื่องเพื่อให้เกิดประโยชน์อย่างสูงสุดต่อผู้ใช้แต่อีกสิ่งหนึ่งที่จะขาดไม่ได้ก็คือคู่มือในการใช้โปรแกรมซึ่งทางผู้จัดทำได้จัดทำขึ้นโดยมีจุดประสงค์ที่จะทำให้ผู้ใช้สามารถที่จะอ่าน ค้นคว้า ได้ด้วยตนเอง จากทั้งหมดที่ได้กล่าวมานั้นเป็นข้อสรุปที่เกิดขึ้นนับจากวันแรกที่ เริ่มสอบหัวข้อปริญญานิพนธ์ จนมาถึงวันสุดท้าย คือ การสอบปริญญานิพนธ์ และส่งผลให้เกิดความสำเร็จของปริญญานิพนธ์นี้ขึ้นมา

5.2 ประโยชน์ที่ได้รับจากการทำโครงงาน

ประโยชน์ที่ได้รับจากโครงงานมี ดังนี้

1. มีความรู้ความเข้าในเกี่ยวกับการเขียนโปรแกรมในแบบวิชวลมากขึ้น
2. รู้จักการวางแผนในการปฏิบัติงานให้เป็นระบบ
3. รู้จักหน้าที่ ความรับผิดชอบ ของแต่ละฝ่ายแต่ละคน
4. ความสามัคคีภายในกลุ่ม
5. การรู้จักแก้ไขปัญหาต่างๆ ได้ด้วยตนเอง หรือขอคำปรึกษาจากอาจารย์ที่ปรึกษา
6. สามารถบริหารเวลา ได้อย่างเหมาะสม
7. มีการเสนอความคิดเห็นต่างๆ ในทางสร้างสรรค์
8. ส่งเสริมการคิดที่มีระบบ ระเบียบ
9. เป็นแนวทางนำไปสู่การศึกษา ค้นคว้า วิจัย ในระดับการศึกษาที่สูงกว่าต่อไป

5.3 ปัญหา และแนวทางการแก้ไข

ปัญหาที่เกิดขึ้นจากการทำโครงงาน มีหลายข้อดังนี้

1. ปัญหา ที่พบเป็นข้อผิดพลาดของโปรแกรม คือ ในบางครั้งเมื่อนำโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS-51 ไปติดตั้งลงบนเครื่องคอมพิวเตอร์แล้วปรากฏว่า รูปภาพที่ปุ่มของ สปีดบาร์ไม่มี
   แนวทางแก้ไข คือ ทำการบันทึกรูปปุ่มเป็น DCR (Delphi Code Resource) แล้วใช้ Tlistbox เรียก
2. ปัญหา คือ คอมไพล์เลอร์ที่ใช้ในโปรแกรมเป็นคอมไพล์เลอร์ที่ทำงานบนดอส เมื่อนำเอาไปใช้กับระบบปฏิบัติการวินโดว์ 95 ทำให้ไม่สามารถเรียกผ่านไดเร็กทรอรี่ที่เก็บแบบวินโดว์ได้ จึงทำให้การคอมไพล์ไม่สามารถทำได้ภายนอกไดเร็กทรอรี่ของโปรแกรม MCS-51
   แนวทางแก้ไข คือ ทำการเก็บข้อมูลภายนอกเป็นไดเร็กทรอรี่ขนาด 16 บิต หรืออีกวิธีหนึ่งคือการเขียนคอมไพล์เลอร์ขึ้นเอง
3. ปัญหา คือ คอมโปเนนต์ที่ใช้ในการแสดงผลของชุดอินเตอร์เฟสเป็นคอมโปเนนต์ที่เป็นมาตฐานที่ให้มากับเดลไฟ ทำให้การแสดงผลไม่สวยงาม และจัดรูปแบบของการแสดงผลได้ยาก
   แนวทางแก้ไข คือ เขียนคอมโปเนนต์ขึ้นมาเอง

5.4 แนวทางการพัฒนาโครงงาน

แนวทางต่อไปที่ควรมีการพัฒนาโปรแกรมให้มีประสิทธิภาพสูงขึ้น มีดังนี้

1. ควรมีการเขียนคอมไพล์เลอร์ขึ้นใช้งานเอง
2. ควรมีชุดอินเตอร์เฟสจำลองเพิ่มขึ้น เช่น มีชุดจำลองการทำงานของสเต็ปปริ้งมอเตอร์
3. ควรมีการเพิ่มชุดอินเตอร์เฟสที่สามารถติดต่อโดยผ่านทางพอร์ตได้
4. ควรปรับปรุงโครงสร้างของหน้าจอ และรูปแบบให้มีรูปแบบที่สวยงามมากขึ้น

ภาคผนวก ก
คู่มือการใช้งานโปรแกรม

บทที่ 1
รู้จักกับโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์

โปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS-51 ที่สร้างขึ้นนี้เป็นโปรแกรมที่ทำงานบนระบบปฏิบัติการวินโดว์ซึ่งมีความสามารถในการจำลองการทำงานของไมโครคอนโทรลเลอร์เบอร์ 8051 ได้ ดังนั้นในบทนี้จะเป็นการกล่าวถึงลักษณะโดยทั่วไปของการทำงานบนระบบปฏิบัติการวินโดว์ ว่าเป็นอย่างไร มีโปรแกรมอะไรบ้างที่ใช้ได้กับวินโดว์ และมีข้อดีอย่างไรในการใช้งานบนระบบปฏิบัติการวินโดว์จากนั้นก็จะกล่าวถึงลักษณะและความสำคัญของโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS - 51 และประโยชน์ที่ได้รับ

1.1 ระบบปฏิบัติการ Windows คืออะไร

Windows หรือชื่อเต็มๆ ว่า Microsoft Windows คือโปรแกรมควบคุมระบบการทำงานของคอมพิวเตอร์ ตั้งแต่การแสดงผลบนจอภาพ, การป้อนข้อมูลด้วยคีย์บอร์ด หรือ mouse การเก็บข้อมูลที่เป็นตัวเลขตัวอักษรหรือรูปภาพ ตลอดจนถึงการพิมพ์ผลงานออกทางเครื่องพิมพ์ ผู้ใช้โปรแกรม Windows จะมีความรู้สึกเหมือนกับว่าจอคอมพิวเตอร์เป็นพื้นที่บนโต๊ะทำงานที่มีอุปกรณ์เครื่องมือช่วยในการทำงานเกือบทุกชนิด เช่น สมุดบันทึก, เครื่องคิดเลข, นาฬิกา, ปฏิทิน และเครื่องพิมพ์ดีดเป็นต้น การใช้เครื่องมือต่างๆ เหล่านี้ก็ทำได้ง่าย เพราะโปรแกรมได้จัดทำทุกอย่างไว้เป็นรูปภาพ ผู้ใช้ต้องการอะไรก็เพียงแต่เลือกรูปภาพนั้นก็จะได้สิ่งที่ต้องการทันที เช่น ต้องการบันทึกข้อความนัดหมายในปฏิทินก็เลือกรูปภาพปฏิทินแล้วใส่ข้อความได้ทันที ขณะเดียวกันหากต้องการดูเวลา ก็สามารถเลือกรูปนาฬิกาได้ทันทีเช่นกัน การใช้เครื่องมือหลายๆ อย่างพร้อมกันก็ไม่มีปัญหา เพียงแต่สภาพบนจอคอมพิวเตอร์จะเหมือนกับโต๊ะทำงานที่มีทั้ง เอกสาร, นาฬิกา, ปฏิทิน วางซ้อนทับผสมปนเปกันไปหมด ลักษณะเช่นนี้ระบบ Windows สามารถจำลองแบบได้ใกล้เคียงมากซึ่งเรียกว่า "Destop" และการใช้ระบบ Windows ก็ง่าย เพราะไม่ต้องจำคำสั่งมากนัก เนื่องจากคำสั่งทุกอย่างจะจัดทำเป็นรูปภาพ เพียงแต่ว่าผู้ใช้อาจลืมความหมายของแต่ละภาพ และส่วนที่เป็นคำสั่งก็จะเป็นแบบ menu เหมือนรายการอาหารให้เลือกใช้ได้อย่างสะดวก นอกจากนี้การใช้ระบบ Windows ยังสามารถควบคุมโปรแกรมอื่นๆ ได้หลายอย่าง ซึ่งมีคุณสมบัติโดดเด่นนานัปการ และในปัจจุบันมีผู้ใช้ Windows มากขึ้น รวมทั้งโปรแกรมที่อยู่ภายใต้ความควบคุมของWindows ก็มีมากขึ้น จึงเป็นส่วนที่ช่วยสนับสนุนให้งานที่ต้องใช้คอมพิวเตอร์พัฒนาเพิ่มขึ้น

1.2 โปรแกรมอะไรที่ใช้ในระบบ Windows ได้บ้าง

เนื่องจากระบบ Windows เป็นโปรแกรมที่ใช้ควบคุมระบบการทำงานคอมพิวเตอร์ดังกล่าวแล้วข้างต้น ดังนั้นจึงมีความสามารถควบคุมการทำงานของโปรแกรมต่างๆ ได้อีกด้วย แต่ระบบ Windows เพิ่มเริ่มนิยมใช้กันเมื่อไม่นานมานี้เอง ดังนั้นโปรแกรมรุ่นเก่าอย่าง WordStar, Lotus, dBase ที่เกิดขึ้นมาก่อนจึงไม่สามารถนำมาใช้ในระบบของ Windows ได้อย่างตรงๆ ต้องอาศัยคำสั่งและเทคนิคพิเศษพอสมควร ในที่นี้จึงขอแบ่งโปรแกรมออกเป็น 2 ชนิด คือ

1. Non-Windows Application หมายถึง โปรแกรมของ DOS ซึ่งไม่สามารถนำมาใช้ในระบบ Windows เช่น WordStar, Lotus, dBase เป็นต้น แต่โปรแกรมเหล่านี้สามารถใช้เทคนิคพิเศษเพื่อให้ทำงานร่วมกับระบบ Windows ได้เช่นกัน
2. Windows Application หมายถึง โปรแกรมที่ใช้กับระบบ Windows โดยเฉพาะ เช่น PageMaker, Excel Microsoft Word เป็นต้น โปรแกรมเหล่านี้ต้องพึ่งพาระบบ Windows เท่านั้น หากนำไปใช้ในคอมพิวเตอร์ที่มีแต่ระบบ DOS อย่างเดียวจะใช้งานไม่ได้เลย

1.3 สรุปได้ว่าใช้ Windows ดีอย่างไร

1. อำนวยความสะดวกให้ผู้ใช้ด้วยการเลือกคำสั่งจากรูปภาพหรือจากรายการคำสั่ง (menu)
2. ผู้ใช้สามารถจัดแต่งจอภาพได้ใหม่ตามความถนัด หรือความชอบของแต่ละคนได้
3. มีโปรแกรมพิเศษเฉพาะทาง ที่อยู่ภายใต้ความควบคุมของ Windows อีกมากมาย
4. สามารถถ่ายโอนข้อมูลทั้งที่เป็นรูปภาพหรือตัวเลข ตัวอักษร จากงานชิ้นหนึ่งไปยังอีกชิ้นหนึ่งได้อย่างสะดวก
5. ผู้ใช้สามารถทำงานได้หลายๆ งานพร้อมกัน โดยไม่จำเป็นต้องรอให้เสร็จงานใดงานหนึ่งก่อน
6. มีอุปกรณ์เครื่องมือที่จำเป็นต่อการทำงานให้อย่างครบครัน รวมทั้งรูปแบบตัวอักษรหรือที่เรียกว่า "font" ชนิดต่างๆ มากมายและสามารถพิมพ์ออกทางเครื่องพิมพ์ได้โดยไม่ต้องปรับแต่แก่ไขตัวเครื่องพิมพ์
7. มีคำแนะนำและคำอธิบายวิธีใช้คำสั่งต่างๆ อย่างละเอียด ทั้งภาษาไทยและอังกฤษ
8. มีเกมส์สนุกๆ ให้เล่นในยามพักผ่อน
9. เฉพาะตัวระบบWindows เองก็สามารถทำงานได้หลายอย่างโดยไม่ต้องพึ่งพาโปรแกรมพิเศษใดๆ เพียงแต่อาจไม่สมบูรณ์หรือไม่อาจสนองความต้องการของผู้ใช้ได้อย่างเต็มที่

1.4 โปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS - 51 คืออะไร

ไมโครคอนโทรลเลอร์ก็คืออุปกรณ์ประมวลผลที่คล้ายกับชิพโปรเซสเซอร์ที่ใช้อยู่ในเครื่องคอมพิวเตอร์ เพียงแต่ลดขนาดและจำกัดประสิทธิภาพการประมวลผลให้กระชับลงพอเหมาะกับการทำงานควบคุมขนาดเล็ก โดยผนวกรวมหน่วยความจำใช้งานเล็กๆ น้อยๆ และผนวกหน่วยเพอริเฟอรัลติดต่อควบคุมกับอุปกรณ์อินพุต/เอาต์พุต รวมทั้งยังอาจผนวกหน่วยการทำงานสำคัญอื่นๆ แตกต่างกันไปในไมโครคอนโทรลเลอร์แต่ละรุ่นละเบอร์ เช่น หน่วยเคาน์เตอร์/ไทเมอร์ เป็นต้น

ขนาดของไมโครคอนโทรลเลอร์จะมีให้เลือกนำไปใช้งานได้อย่างเหมาะสมตั้งแต่ขนาดขาใช้งานน้อยๆ 8 ขาสัญญาณ มีกระบวนการสั่งงานที่ไม่ซับซ้อน จนถึงขนาดใช้กับงานที่ต้องการคำสั่งงานซับซ้อนมากมาย ซึ่งจะมีขาสัญญาณใช้งานแบบกริดอาเรย์นับร้อยขารอบตัว

จากคุณสมบัติต่างๆ ที่เป็นประโยชน์ของตัวไมโครคอนโทรลเลอร์ จึงทำให้มีผู้นำไมโครคอนโทรลเลอร์มาใช้งานในด้านต่างๆ อย่างกว้างขวางมาโดยตลอด แต่สิ่งหนึ่งที่เกิดขึ้นจากการนำไมโครคอนโทรลเลอร์ไปใช้นั้นก็คือ การที่เราจะต้องทำการเขียนโปรแกรมเพื่อควบคุมการทำงานของไมโครคอนโทรเลอร์ตามที่เราต้องการ ซึ่งทำให้เกิดปัญหาในแง่ของการสร้างและการพัฒนาโปรแกรมเป็นไปอย่างไม่ค่อยมีประสิทธิภาพและมีความล่าช้าในการเขียนโปรแกรม ดังนั้นเพื่อให้การเขียนโปรแกรมมีความสะดวกรวดเร็วมากขึ้น และเพื่อง่ายต่อการใช้งาน การสร้างโปรแกรมเพื่อจำลองการทำงานของไมโครคอนโทรลเลอร์จึงเป็นส่วนหนึ่งที่ช่วยให้เราสามารถฝึกหัดและทดสอบโปรแกรมได้เหมาะสมกับการนำไปใช้งานจริงได้ และยังช่วยให้ผู้เขียนโปรแกรมสามารถที่จะพัฒนาโปรแกรมเพื่อให้เกิดประสิทธิภาพอย่างสูงสุดได้ ดังนั้นการศึกษาในวิธีการหรือหลักการในการใช้งานโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS - 51 จึงมีประโยชน์ทั้งต่อนักศึกษาและผู้สนใจโดยทั่วไปอย่างแท้จริง

1.5 ประโยชน์ที่ได้รับ

1. สามารถนำโปรแกรมไปใช้งานบนระบบปฏิบัติการ Windows ได้
2. สามารถช่วยให้การเรียนการสอนในวิชาไมโครคอนโทรลเลอร์ ได้มีการทดลองและฝึกหัดการเขียนโปรแกรมได้สะดวกขึ้น
3. ช่วยลดค่าใช้จ่ายในการจัดซื้อเครื่อง Single Board เพื่อใช้ในการทดลอง
4. สามารถที่จะพัฒนาโปรแกรมของ MCS-51 ได้สะดวก รวดเร็ว และมีความสามารถสูงขึ้น

1.6 เตรียมการเบื้องต้นอย่างไร

เครื่องคอมพิวเตอร์ที่จะนำโปรแกรมจำลองการทำงานของไมโครคอนโทรเลอร์ MCS - 51 ไปใช้ได้นั้น ควรมีส่วนประกอบที่สำคัญดังนี้คือ

1. เครื่องคอมพิวเตอร์ที่มี CPU รุ่น Pentium 100 MHz ขึ้นไป
2. จอภาพสีชนิด VGA หรือ Super VGA, 1024 X 468, 256 สีขึ้นไป
3. คีย์บอร์ด รุ่น 101 ปุ่มหรือมากกว่า
4. mouse ชนิด 2 หรือ 3 ปุ่ม
5. ฮาร์ดดิสก์ ที่มีความจุอย่างน้อย 10 MB พร้อมโปรแกรม SXA51
6. เครื่องพิมพ์ที่สามารถพิมพ์ได้ทั้งรูปภาพ (graphic) และตัวอักษร (text)
7. ความจุของหน่วยความจำ (CPU) อย่างน้อย 2MB และถ้ามีมากกว่านี้การทำงานของโปรแกรมจะเร็วขึ้นด้วย

1.7 กว่าจะมาเป็นโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS - 51

โปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS -51 เกิดขึ้นจากแนวความคิดของนักศึกษาสาขาอิเล็กทรอนิกส์และคอมพิวเตอร์ ห้อง 2 ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสหกรรม สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง โดยมีความคิดว่า ในการเรียนการสอนในวิชาไมโครคอนโทรลเลอร์นั้นนักศึกษาได้ศึกษาเกี่ยวกับทฤษฎีของไมโครคอนโทรลเลอร์ เบอร์ 8051 เข้าใจถึงหลักการในการเขียนโปรแกรม และส่วนประกอบต่างๆ ของตัวไมโครคอนโทรลเลอร์เป็นอย่างดี แต่ทั้งนี้นักศึกษายังขาดประสบการณ์ในเรื่องของการฝึกปฏิบัติในการเขียนโปรแกรมใช้งานจริงเท่าที่ควร อันเนื่องมาจากปัญหาต่างๆ ที่เกิดขึ้น ทั้งในด้านของเครื่อง single board ที่ใช้สำหรับในการเขียนโปรแกมมีน้อยไม่เพียงพอต่อนักศึกษาที่มีจำนวนมากกว่า และปัญหาในด้านอื่นๆ ที่เกิดขึ้นอีก

ถึงแม้ว่าในปัจจุบันเราสามารถที่จะใช้งานโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ที่มีผู้ผลิตหลายๆ บริษัทได้ผลิตขึ้นแล้วก็ตาม แต่ด้วยเหตุผลทางด้านเงินงบประมาณในการใช้จ่ายมีอยู่จำนวนจำกัด จึงทำให้ภาควิชาต้องนำเงินงบประมาณที่ได้รับไปใช้ในส่วนที่จำเป็นมากกว่า การซื้อซอฟแวร์ที่มีราคาแพง

นอกเหนือจากที่โปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ที่มีใช้อยู่ในปัจจุบันจะมีราคาที่ค่อนข้างสูงแล้ว ในการใช้งานยังเป็นการทำงานบนระบบปฏิบัติการ DOS ซึ่งทำให้การ

ใช้งานเป็นไปได้ยากกว่า บนโปรแกรมที่มีการทำงานบนระบบปฏิบัติการวิสโดว์ซึ่งเป็นโปรแกรมที่สร้างขึ้นจากคณะผู้จัดทำในขณะนี้แต่จากการเปรียบเทียบในเบื้องต้นนี้คณะผู้จัดทำมิได้หวังว่าโปรแกรมของคณะผู้จัดทำนั้นจะมีส่วนเกี่ยวข้องกับส่วนแบ่งในการตลาดของบริษัทนั้นๆ ไม่ เพียงแต่คณะผู้จัดทำมีความมุ่งหมายว่า โปรแกรมตัวนี้จะเป็นจุดเริ่มต้นของความคิดที่เกิดขึ้นจากคนไทยเพื่อการพัฒนาภูมิปัญญาที่มีอยู่ให้เกิดประโยชน์ และสร้างสรรค์สิ่งที่เป็นนวตกรรมและเทคโนโลยีต่อไปในอนาคต

จากเหตุผลข้างต้นจึงเป็นแรงผลักดันให้คณะผู้จัดทำได้เล็งเห็นถึงความสำคัญของการมีส่วนร่วมในการที่จะสร้างสรรค์ พัฒนากระบวนการเรียนการสอนของนักศึกษาให้มีประสิทธิภาพและเกิดผลสัมฤทธ์ทางการเรียนอย่างสูงสุด และสามารถนำประโยชน์ที่ได้รับจากการเรียนไปใช้ให้เกิดประโยชน์ทั้งต่อตนเองและประเทศชาติต่อไปในอนาคตได้อย่างแท้จริง

สำหรับอนาคตที่หวังไว้คือการพัฒนาโปรแกรมจำลองการทำงานของไมโครคอนโทรเลอร์ให้มีประสิทธิภาพ และมีความสามารถที่สูงขึ้น สามารถตอบสนองต่อผู้ใช้ได้ดีขึ้น แก้ไขเพิ่มเติมส่วนที่ขาดหายไปอันเนื่องจากเทคโนโลยีที่เปลี่ยนแปลงไปเพื่อให้โปรแกรมมีความทันสมัยอยู่ตลอดเวลา

1.8 Mouse กับคีย์บอร์ด ใช้เมื่อไรและอย่างไร

1.8.1 Mouse

เป็นอุปกรณ์สำคัญที่ช่วยอำนวยความสะดวกในการเลือกใช้คำสั่งและจัดการข้อมูล โดยเฉพาะส่วนที่เกี่ยวกับรูปภาพ ซึ่ง mouse จะมี 2 แบบ คือ 2 ปุ่มกับ 3 ปุ่ม ขึ้นอยู่กับรุ่นหรือยี่ห้อ แต่การใช้ส่วนใหญ่จะเป็นปุ่มซ้ายเพียงปุ่มเดียวเท่านั้น (สำหรับมือขวา) หากถนัดมือซ้ายต้องมีการสั่งไว้ก่อนโดยใช้ระบบ Windows ส่วนวิธีการกดปุ่มของ mouse มีจังหวะการกดปุ่มหลายอย่าง ดังนี้คือ

1. การเคลื่อน mouse อย่างเดียว จะทำให้สัญญลักษณ์รูปลูกศรบนจอภาพเลื่อนตามไปด้วย จุดประสงค์เพื่อต้องการไปเลือกคำสั่งหรือรูปภาพ หรือชี้ตำแหน่งที่จะป้อนข้อมูลในจอภาพ
2. Click mouse คือ กดปุ่มซ้ายเพียงปุ่มเดียวในลักษณะการเคาะโดยใช้นิ้วชี้
3. Double click คือ กดปุ่มซ้ายเพียงปุ่มเดียว ในลักษณะการเคาะด้วยนิ้วชี้ติดๆ กัน 2 ครั้ง (ต้องติดๆ กันด้วย ไม่เช่นนั้นจะกลายเป็นการ click ธรรมดา 2 ครั้งแทน)
4. Drag mouse คือ กดปุ่มซ้ายค้างไว้ในขณะที่เลื่อน mouse ไปด้วย

1.8.2 คีย์บอร์ด

เป็นอุปกรณ์สำหรับป้อนข้อมูลที่เป็นตัวอักษรหรือตัวเลข โดยมีลักษณะแป้นพิมพ์เหมือนเครื่องพิมพ์ดีด และมีมากกว่าด้วยปุ่มพิเศษเฉพาะงานอีกมากมาย โดยปกติการป้อนข้อมูลจะทำเหมือนกับใช้เครื่องพิมพ์ดีด ยกเว้นบางกรณีอาจมีการกด 2 ปุ่ม ควบกัน ดังตัวอย่างสัญลักษณ์ดังนี้คือ

<Shift+Tab> หมายถึง กดปุ่ม <Shift> ค้างไว้ก่อนแล้วกดปุ่ม <Tab> ด้วย เมื่อกดปุ่มได้ 2 ปุ่มแล้วก็ปล่อยมือได้ทันที อย่ากดค้างไว้ตลอดเพราะอาจมีผลต่อการทำงานของโปรแกรม ซึ่งบางตรั้งอาจทำให้ผลลัพธ์คลาดเคลื่อนหรือผิดไป

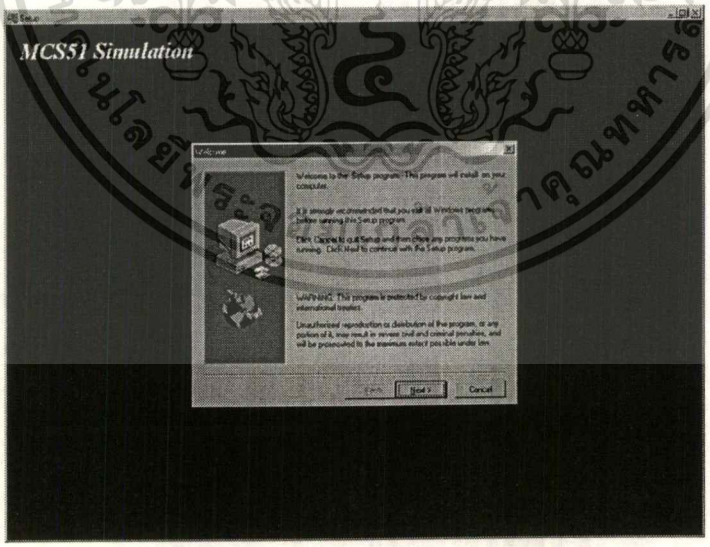
บทที่ 2

เริ่มต้นกับโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS - 51

2.1 วิธีติดตั้งโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS - 51

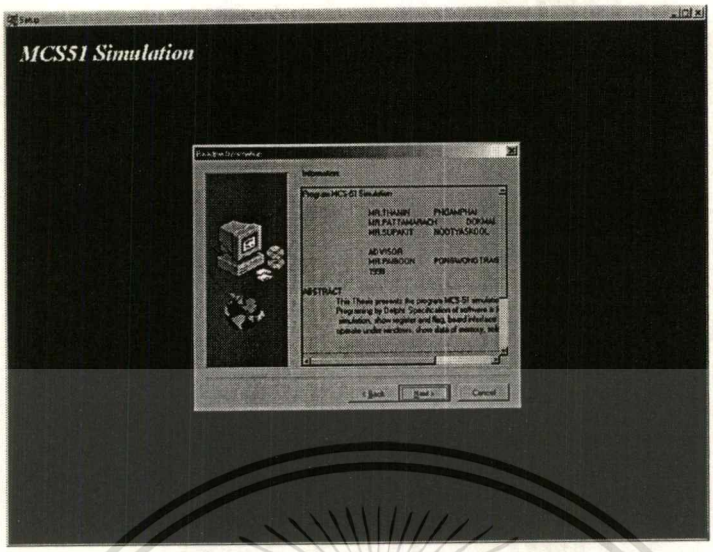
วิธีติดตั้งโปรแกรม ที่จะอธิบายต่อไปนี้เป็นการยกตัวอย่างวิธีการติดตั้ง ซึ่งอาจมีได้หลายวิธี นอกจากนี้

1. นำแผ่นที่เป็นชุดติดตั้งโปรแกรม(แผ่นที่ 1)ใส่เข้าไปในเครื่องคอมพิวเตอร์ของเรา
2. เลือกไปที่ Start เลือก คำสั่ง Run
3. จะมีหน้าต่าง Run ขึ้นมา ให้เลือกไปที่ Browse
4. ที่หน้าต่าง Browse ให้เลือกไดรฟ์ที่เราใส่ชุดติดตั้งโปรแกรมเอาไว้
5. เลือกไฟล์ setup แล้วเลือก open จะกลับมาที่หน้าจอ Run ให้เลือก OK
6. จะแสดงหน้าต่างโปรแกรมติดตั้งขึ้นมา ดังรูปต่อไปนี้



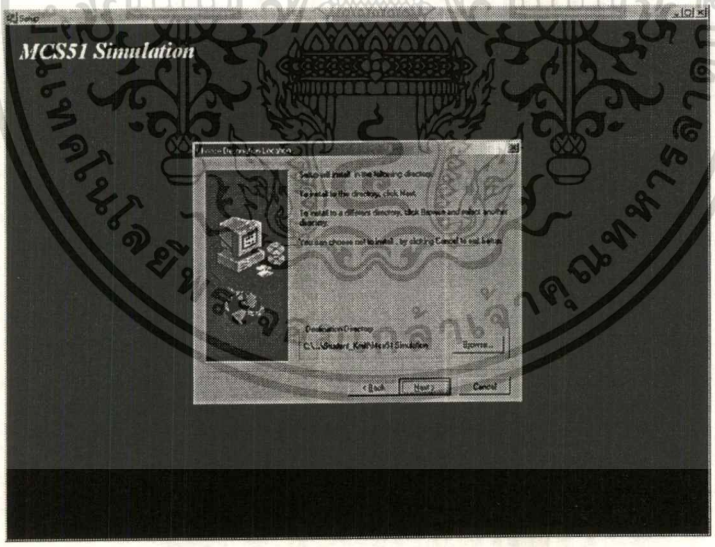
รูปที่ 2.1 หน้าต่างโปรแกรมติดตั้ง

7. ให้คลิกที่ปุ่ม Next



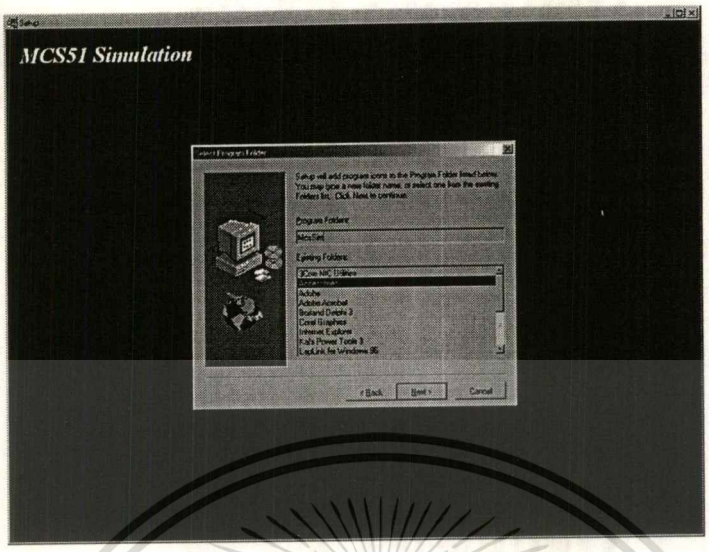
รูปที่ 2.2 รายละเอียดของผู้เขียนและตัวโปรแกรม

8. ให้คลิกที่ปุ่ม Next



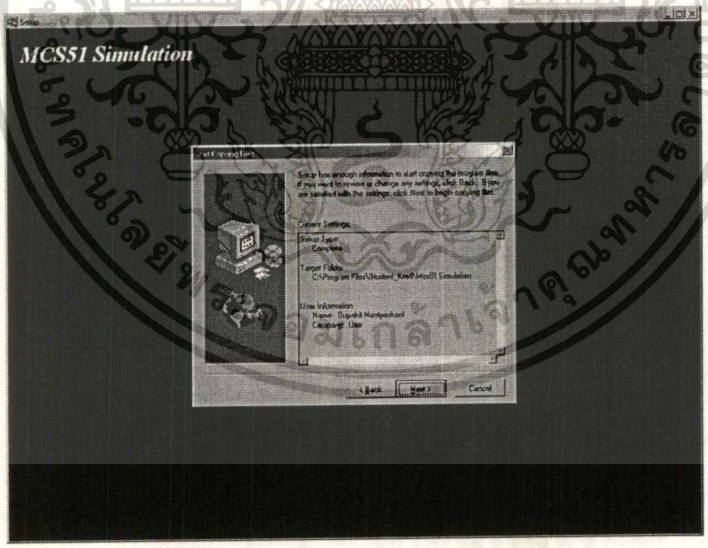
รูปที่ 2.3 ให้ผู้ใช้เลือก Directory

9. ถ้าผู้ใช้ต้องการเปลี่ยน Directory ไปไว้ที่อื่นให้คลิกที่ปุ่ม Browse แล้วทำการเลือก Directory ที่ต้องการ จากนั้นก็คลิกที่ปุ่ม Next



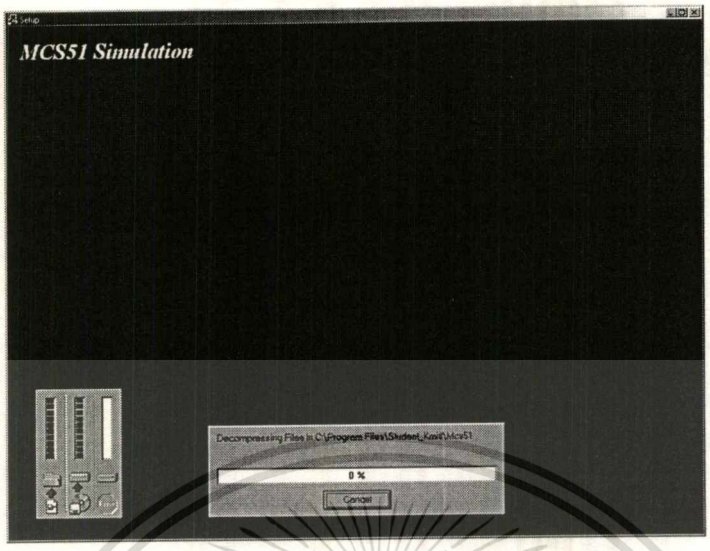
รูปที่ 2.4 การสร้างชื่อโฟลเดอร์

10. ให้คลิกที่ปุ่ม Next

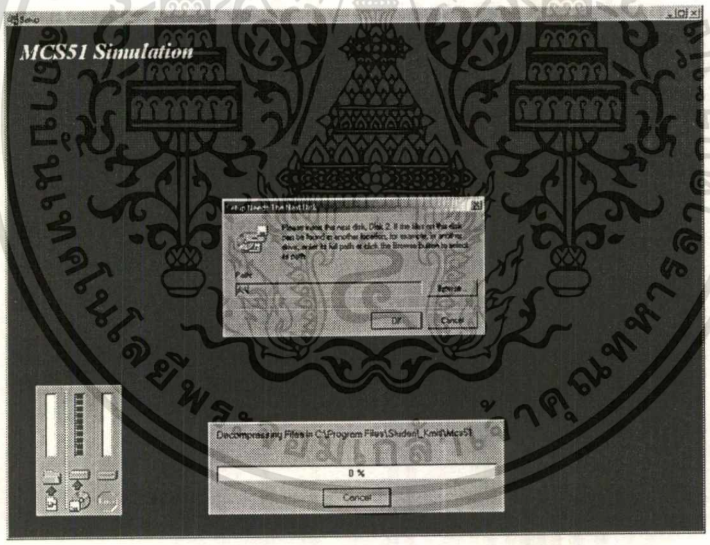


รูปที่ 2.5 แจ้งรายละเอียดการติดตั้งโปรแกรม

11. ให้คลิกที่ปุ่ม Next

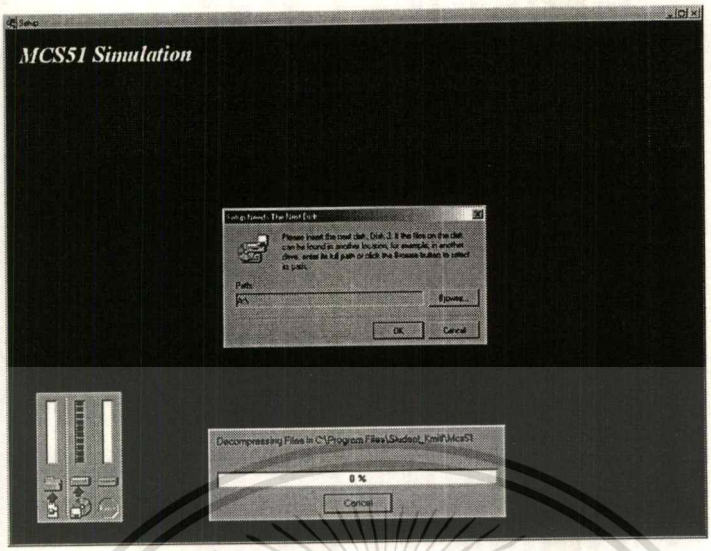


รูปที่ 2.6 โปรแกรมติดตั้งทำการอ่านข้อมูลบันทึกลงในฮาร์ดดิสก์



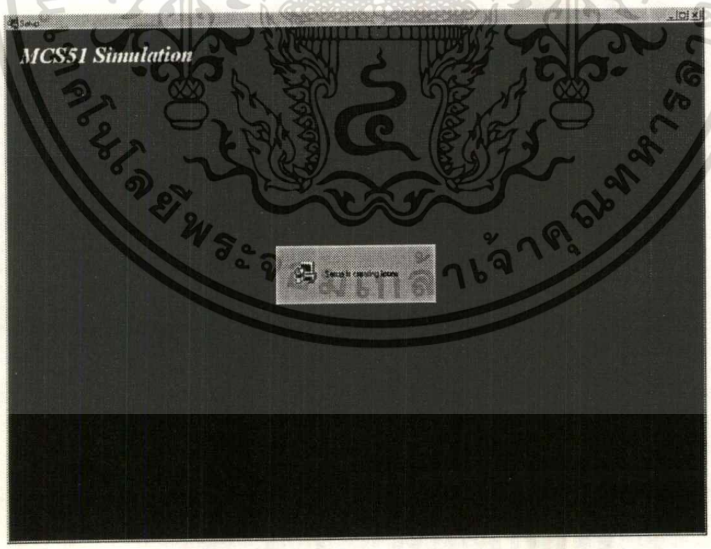
รูปที่ 2.7 ถามหาแผ่นดิสก์แผ่นที่ 2

12. โปรแกรมติดตั้งจะถามหาแผ่นดิสก์แผ่นที่ 2 ให้เราใส่แผ่นดิสก์แผ่นที่ 2 ในดิสก์ไดรฟ์ แล้วคลิก OK



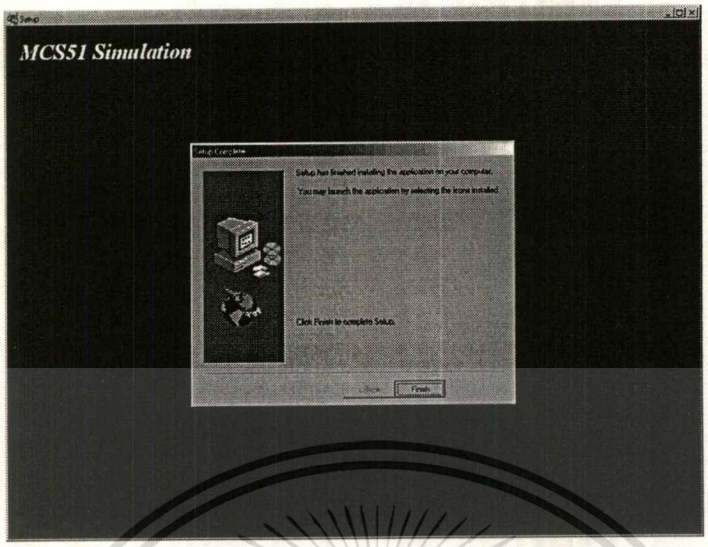
รูปที่ 2.8 ถามหาแผ่นดิสก์แผ่นที่ 3

13. โปรแกรมติดตั้งจะถามหาแผ่นดิสก์แผ่นที่ 3 ให้เราใส่แผ่นดิสก์แผ่นที่ 3 ในดิสก์ไดรฟ์ แล้วคลิก OK



รูปที่ 2.9 การติดตั้ง ICON ของโปรแกรมติดตั้ง

14. โปรแกรมติดตั้งจะทำการติดตั้ง icon เพื่อให้เราเรียกใช้งานโปรแกรมได้

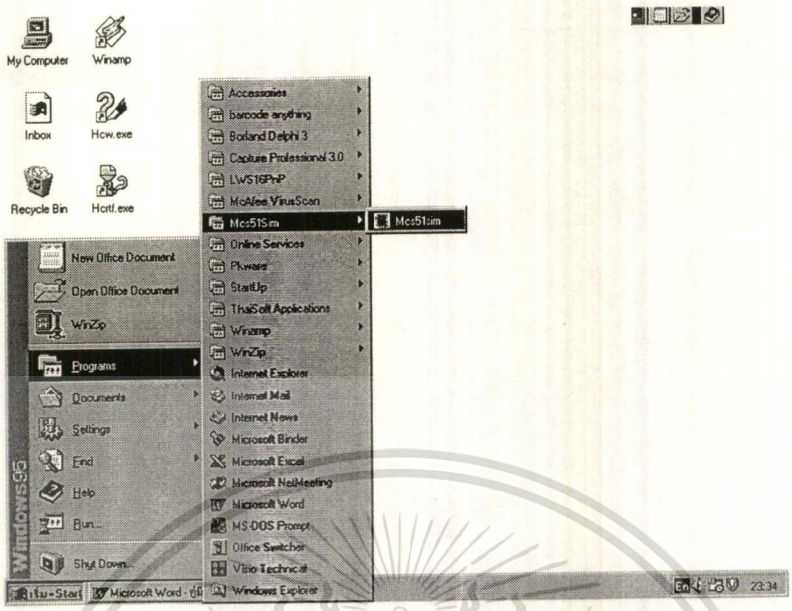


รูปที่ 2.10 เสร็จสิ้นการติดตั้งโปรแกรม

15. เมื่อโปรแกรมติดตั้ง ติดตั้งโปรแกรมจำลองการทำงานของไมโครคอนโทรเลอร์เรียบร้อยแล้วจะแสดงหน้าต่างดังรูปที่ ให้เราคลิกที่ปุ่ม Finish คือจบการติดตั้งโปรแกรม

2.2 การเรียกใช้งานโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS - 51

1. เลือกไปที่ Start
2. เลือกไปที่ Programs / Mcs51Sim ดังรูปที่ 2.11 ก็จะแสดงหน้าจอของโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS - 51 ออกมา



รูปที่ 2.11 การเปิดโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS - 51

2.3 หน้าจอของโปรแกรม

หน้าจอของโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS - 51 แบ่งได้คือ

1. หน้าต่างหลัก Main Menu
2. หน้าต่างหน่วยความจำ
3. หน้าต่างรีจิสเตอร์
4. หน้าต่างอิดิเตอร์
5. หน้าต่างอินเตอร์เฟส
6. หน้าต่าง Message

2.3.1 หน้าต่างหลัก

หน้าต่างหลักดังแสดงในรูปที่ 2.12 ประกอบด้วย

1. ไตเติลบาร์ (Title bar) จะแสดงชื่อโปรแกรม
2. เมนูบาร์ (Menu bar) แสดงรายการคำสั่งตั้งแต่ File จนถึง Help
3. สปีดบาร์ (Speed bar) ประกอบด้วย

3.1 New คือ ปุ่มที่ใช้เมื่อต้องการเริ่มต้นกรอกโปรแกรมใหม่ทั้งหมด โดยจะลบหรือยกเลิกโปรแกรมเก่าที่ปรากฏอยู่ในหน้าต่างอิดิเตอร์ทั้งหมด

3.2 Open คือ ปุ่มที่ใช้เรียกไฟล์ข้อมูลที่เก็บอยู่ในดิสก์กลับมาใช้งาน

3.3 Save คือ ปุ่มที่ใช้ในการเก็บบันทึกข้อมูลจากอิดิเตอร์ลงดิสก์โดยใช้ชื่อไฟล์เดิม แต่ถ้าเป็นการบันทึกครั้งแรกของข้อมูลนั้น โปรแกรมจะเปลี่ยนคำสั่งเป็น Save As ให้โดยอัตโนมัติ

3.4 Sim/Step คือ ปุ่มที่ใช้เลือกระหว่างการทำงานของโปรแกรมในโหมด Simulate หรือ Step

3.5 Compile คือ ปุ่มที่ใช้สั่งให้โปรแกรมทำการคอมไพล์โปรแกรมที่เขียนขึ้นที่อิดิเตอร์

3.6 Run คือ ปุ่มที่ใช้สั่งให้โปรแกรมทำการ Run โปรแกรมตามคำสั่งที่เขียน

3.7 Stop คือ ปุ่มที่ใช้สั่งให้โปรแกรมหยุดการทำงาน

3.8 Step คือ ปุ่มที่ใช้สั่งให้โปรแกรมทำงานทีละคำสั่ง

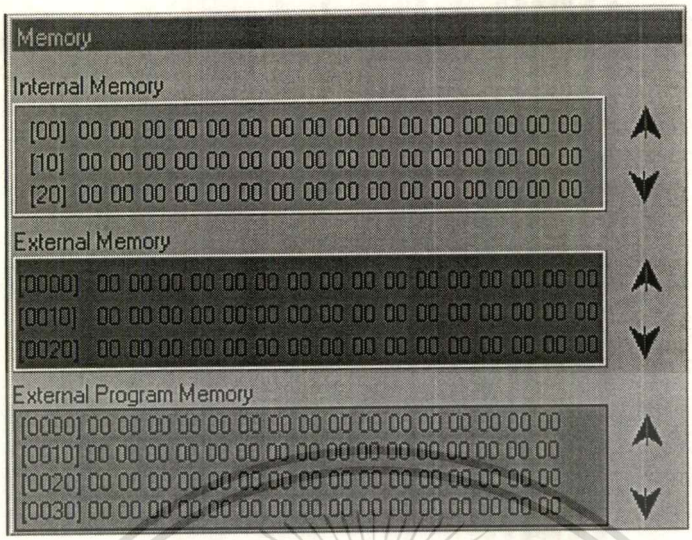
รูปที่ 2.12 หน้าต่างหลัก

2.3.2 หน้าต่างหน่วยความจำ

หน้าต่างหน่วยความจำเป็นหน้าต่างที่แสดงให้เห็นถึงข้อมูลต่างๆ ที่เก็บไว้ในหน่วยความจำที่ตำแหน่งต่างๆ โดยแบ่งเป็น 3 ส่วน ดังรูปที่ 2.13 คือ

1. Internal Memory
2. External Memory
3. External Program Memory

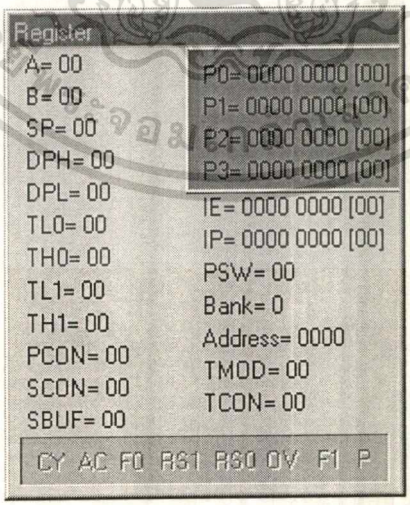
มีปุ่มลูกศรให้เลื่อนขึ้นเลื่อนลงเพื่อใช้ในการดูข้อมูลที่ตำแหน่งต่างๆ



รูปที่ 2.13 หน้าต่างหน่วยความจำ

2.3.3 หน้าต่างรีจิสเตอร์

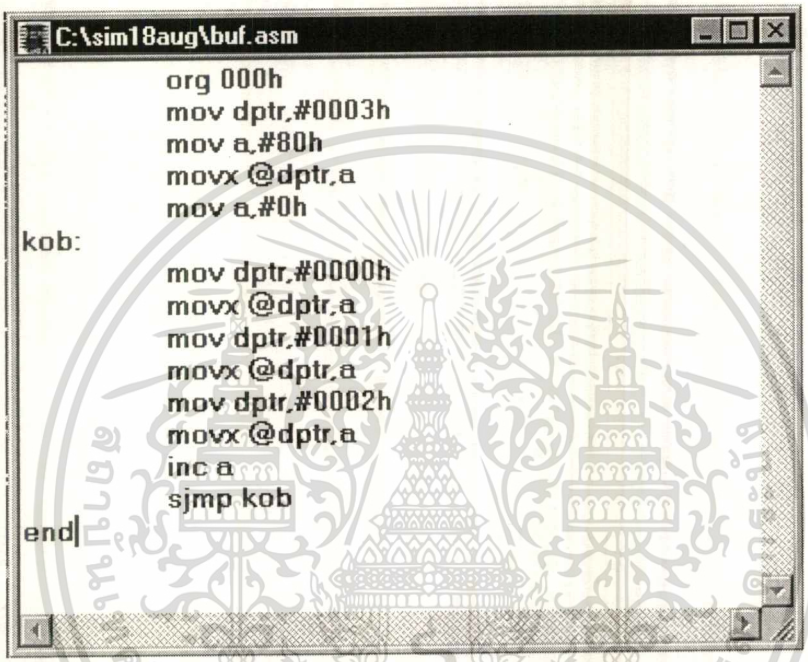
หน้าต่างรีจิสเตอร์เป็นหน้าต่างที่ใช้แสดงค่าของรีจิสเตอร์และสถานะของแฟลกต่างๆ ของไมโครคอนโทรลเลอร์ MCS-51 และข้อมูลภายในพอร์ต P0, P1, P3 ของ 8051 และตำแหน่งแอดเดรสในขณะรันโปรแกรม โดยแบ่งออกเป็นส่วนๆ แยกจากกัน ดังรูปที่ 2.14



รูปที่ 2.14 หน้าต่างรีจิสเตอร์

2.3.4 หน้าต่างอิดิเตอร์

หน้าต่างอิดิเตอร์เป็นหน้าต่างที่มีความสำคัญอย่างมากเพราะว่าเราจะใช้หน้าต่างนี้ในการเขียนโปรแกรมภาษาแอสเซมบลี้ของ 8051 เพื่อทดลอง โดยมีลักษณะของการใช้งานเหมือนกับการทำงานของอิดิเตอร์โดยทั่วไป คือสามารถลบ แทรก ตัด คำได้ ดังรูปที่ 2.15



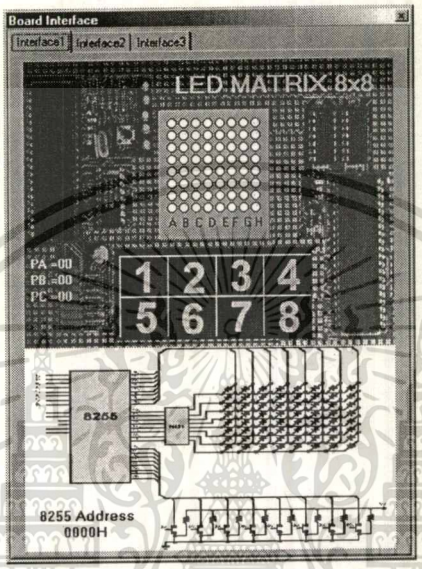
รูปที่ 2.15 หน้าต่างอิดิเตอร์

2.3.5 หน้าต่างอินเตอร์เฟส

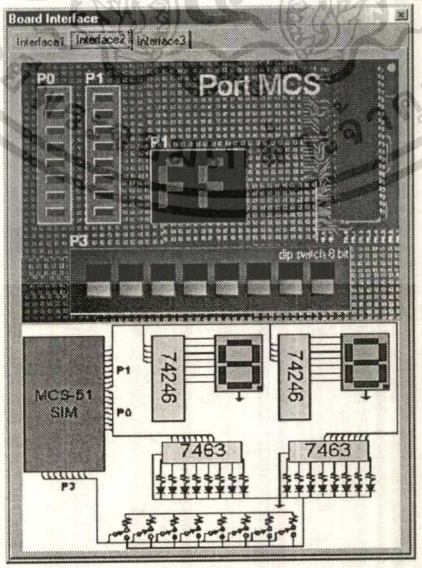
หน้าต่างอินเตอร์เฟสเป็นส่วนของการจำลองการทำงานทางด้านฮาร์ดแวร์ โดยแบ่งออกเป็น 3 ชุดอินเตอร์เฟส คือ

1. อินเตอร์เฟส 1(รูปที่ 2.16) คือ การอินเตอร์เฟสกับ 8255 โดยมีการต่อกับ LED Matrix ขนาด 8 X 8 และยังมีอินพุต คือ สวิทช์ 8 ตัว
2. อินเตอร์เฟส 2 (รูปที่ 2.17) คือ การอินเตอร์เฟสกับพอร์ตของ 8051 โดยมีการต่อ P0 กับ LED 8 ดวง, ต่อ P1 กับ LED 8 ดวง และ 7-segment, ต่อ P3 เป็นอินพุต คือ DIP Switch 8 bit
3. อินเตอร์เฟส 3 (รูปที่ 2.18) คือ การอินเตอร์เฟสกับ 8255 โดยที่ผู้ใช้สามารถที่จะเลือกได้ว่าให้พอร์ตนั้นเป็นอินพุตหรือเอาต์พุต โดยสามารถเลือกได้ดังนี้

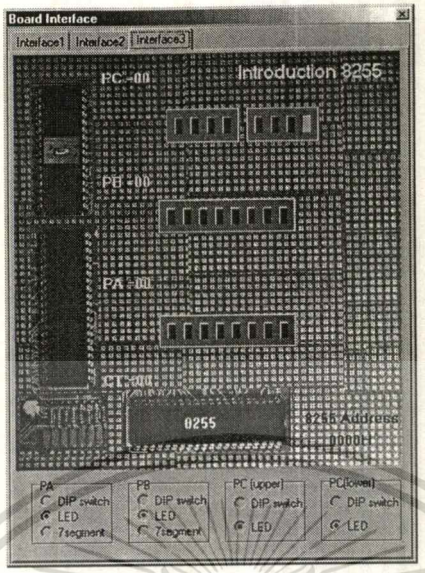
1. PA, PB เป็น อินพุต เลือก DIP Switch 8 bit
2. PA, PB เป็น เอาต์พุต เลือก LED หรือ 7-segment
3. $PC_{(lower)}$, $PC_{(upper)}$ เป็น อินพุต เลือก DIP Switch 8 bit
4. $PC_{(lower)}$, $PC_{(upper)}$ เป็น เอาต์พุต เลือก LED



รูปที่ 2.16 หน้าต่างอินเตอร์เฟส 1



รูปที่ 2.17 หน้าต่างอินเตอร์เฟส 2

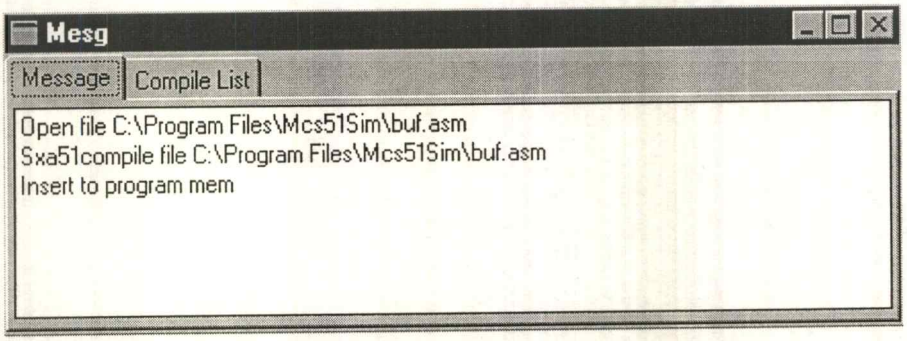


รูปที่ 2.18 หน้าต่างอินเตอร์เฟส 3

2.3.6 หน้าต่าง Message

หน้าต่าง Message ดังรูปที่ 2.19เป็นอีกหน้าต่างหนึ่งที่มีประโยชน์ต่อการตรวจแก้ไขส่วนที่ผิดพลาดของโปรแกรมได้ง่ายและสะดวกขึ้น โดยแบ่งออกเป็น 2 หน้าต่างคือ

1. หน้าต่าง Message คือ การแสดงในส่วนของข้อมูลต่างๆ ที่มีการทำงานที่เกิดขึ้นจากตัวโปรแกรม
2. หน้าต่าง Compile list คือ การแสดงรายการของคำสั่งในแต่ละบรรทัดให้ผู้ใช้สามารถสังเกตุการทำงานของโปรแกรมได้ (ถ้าเลือกการทำงานในโหมด Step)



รูปที่ 2.19 หน้าต่าง Message

2.4 รายการคำสั่งใน Main Menu

File		ในเมนูนี้จะเป็นการจัดการเกี่ยวกับไฟล์ดังมีเมนูย่อยดังนี้
	New	คือ การสร้างไฟล์ใหม่
	Open	คือ การเปิดไฟล์เก่าที่มีการบันทึกไว้แล้ว
	Import Hex file	คือ การโหลดไฟล์ที่มีนามสกุลเป็น Hex ไฟล์มาใช้
	Save	คือ การบันทึกไฟล์ที่เราเขียนขึ้นโดยจะบันทึกเป็นไฟล์ .asm
	Save as	คือ การบันทึกไฟล์โดยที่จะมีการให้ตั้งชื่อไฟล์ใหม่ได้
	Print	คือ การสั่งพิมพ์ไฟล์ที่เราเขียนขึ้น
	Print Setup	คือ การกำหนดค่าต่างๆ ในการสั่งพิมพ์
	Exit	คือ การออกจากโปรแกรม
Edit		ในเมนูนี้จะเป็นการแก้ไขไฟล์ที่ต้องการโดยมีเมนูย่อยดังนี้
	Cut	คือ การตัดส่วนที่ไม่ต้องการออก
	Copy	คือ การคัดลอกข้อความ
	Paste	คือ การวางข้อความที่ได้จากการคัดลอก
	Select all	คือ การเลือกข้อความทั้งหมด
Run		ในเมนูนี้จะเป็นการจัดการเกี่ยวกับการแปลงคำสั่งมีเมนูย่อยดังนี้
	Sxa51 Compile	คือ การสั่งให้แปลงคำสั่งของโปรแกรมที่เราเขียนขึ้น
	Run	คือ การสั่งให้โปรแกรมทำงานตามคำสั่ง
	Stop	คือ การสั่งให้โปรแกรมหยุดทำงาน
	Step	คือ การสั่งให้โปรแกรมทำงานทีละคำสั่ง
	Change Mode	คือ การเปลี่ยนโหมดการทำงานของโปรแกรม โดยมีให้เลือก 2 โหมด คือ 1. โหมด Sim คือทำทุกคำสั่งตลอดโปรแกรม 2. โหมด Step คือทำทีละคำสั่ง
Tools		ในเมนูนี้จะเป็นการจัดการเกี่ยวกับชุดเครื่องมือและฟอร์มต่างๆ
	Set Memory	คือ การกำหนดค่าของหน่วยความจำจำลองที่เราต้องการ
	Refresh all	คือ การ reset ค่าต่างๆ ทั้งหมด
	Address 8255	คือ การกำหนดค่าตำแหน่งแอดเดรสให้กับ 8255 ซึ่งมีเมนูย่อยให้เลือกอีก 2 ชุดอินเตอร์เฟส คือ อินเตอร์เฟส 1 กับ อินเตอร์เฟส 3

Address Show คือ การเลือกตำแหน่งของหน่วยความจำที่เราต้องการให้แสดงผล ซึ่งสามารถเลือกแสดงผลได้ 3 หน่วยความจำ คือ

1. Internal Data Memory
2. External Data Memory
3. External Program Memory

Editor คือ การเลือกหรือไม่เลือกการแสดงผลฟอร์ม Editor

Register คือ การเลือกหรือไม่เลือกการแสดงผลฟอร์ม Register

Memory คือการเลือกหรือไม่เลือกการแสดงผลฟอร์ม Memory

Message คือ การเลือกหรือไม่เลือกการแสดงผลฟอร์ม Message

Board Simulation คือ การเลือกหรือไม่เลือกการแสดงผลฟอร์ม Board Simulation

Help ในเมนูนี้เป็นส่วนของการแสดงวิธีการใช้โปรแกรม

Content คือ การเข้าสู่หน้าต่างสารบัญของ Help

Using Mcs51simคือ การเข้าสู่คู่มือในการใช้งานโปรแกรม

2.5 การเขียนโปรแกรม MCS-51

การเขียนโปรแกรม Mcs-51 มีขั้นตอนดังนี้ คือ

1. เมื่อเข้าสู่ Program MCS-51 Simulation แล้ว เลือกไปที่ File/New หรืออาจจะ Click ที่ปุ่ม New ใน speed bar ได้
2. ที่ฟอร์ม Editor ให้เราพิมพ์คำสั่งของโปรแกรม MCS-51 ตามรูปแบบของการเขียนโปรแกรม MCS-51 โดยกำหนดให้เริ่มต้นที่ตำแหน่ง 0000H
3. ทำการบันทึกไฟล์ที่เขียน โดยเลือกไปที่ File/Save หรือ click ที่ปุ่ม Save ใน speed bar ได้
4. ทำการตั้งชื่อไฟล์ โดยจะมีนามสกุลเป็น .asm

2.6 การ Compile โปรแกรม

การ Compile โปรแกรม มีขั้นตอนดังนี้ คือ

1. ทำการเปิดไฟล์ที่เขียนไว้ โดยเลือกที่ File/Open หรืออาจจะ Click ที่ปุ่ม Open ใน speed bar ได้
2. เลือกไปที่ Run/Sxa51 compile
3. เลือกไปที่ Run/Run หรือ Click ที่ปุ่ม Run ใน speed bar ได้
4. ตรวจสอบผลการ Run จากฟอร์ม Show Register หรือ Board Interface ได้

หมายเหตุ

การ Run สามารถที่จะเลือกทำได้ 2 โหมด คือ

1. Run ทีละคำสั่ง โดยเลือกไปที่ Run/Change Mode แล้วเลือกเป็นแบบ Stepหรือ Click ที่ปุ่ม Sim/Step ใน speed bar แล้วให้แสดงเป็น Step
2. Run ทั้งโปรแกรม โดยเลือกไปที่ Run/Change Mode แล้วเลือกเป็นแบบ Sim หรือ Click ที่ปุ่ม Sim/Step ใน speed bar แล้วให้แสดงเป็น Sim

2.7 การกำหนดค่าในหน่วยความจำ

การกำหนดค่าในหน่วยความจำ มีขั้นตอนดังนี้ คือ

1. เลือกไปที่ Tools/Set Memory
2. เลือกชนิดของหน่วยความจำที่ต้องการกำหนดค่าในช่อง Memory Type
3. ป้อนตำแหน่งหน่วยความจำที่ต้องการกำหนดค่าในช่อง Address
4. ป้อนข้อมูลที่ต้องการกำหนดค่าในช่อง Data
5. click ปุ่ม OK. เมื่อตั้งค่าเรียบร้อยแล้ว หรือ click ปุ่ม Cancle เมื่อต้องการยกเลิก

หมายเหตุ

ตำแหน่งของ Internal Memory คือ 0000H ถึง 0FFFH

ตำแหน่งของ External Memory คือ 0000H ถึง FFFFH

ตำแหน่งของ External Program Memory คือ 0000H ถึง FFFFH

2.8 การเลือกแสดงผลหน่วยความจำ

การเลือกแสดงผลหน่วยความจำ มีขั้นตอนดังนี้ คือ

สามารถที่จะแสดงผลของหน่วยความได้ 3 หน่วยความจำคือ

1. Internal Data Memory
2. External Data Memory
3. External Program Memory

มีวิธีการดังนี้

1. เลือกไปที่ Tools/Address Show
2. ที่ Address Show จะมีเมนูย่อยอีกว่าต้องการจะให้แสดงผลหน่วยความจำใด
3. เมื่อเลือกได้แล้วจะมี Dialog box แสดงขึ้นมา
4. ป้อนตำแหน่งของหน่วยความจำที่ต้องการให้แสดงในช่อง Address
5. click ปุ่ม OK เมื่อป้อนค่าถูกต้องแล้ว หรือ click ปุ่ม Cancle เมื่อต้องการยกเลิก

<u>หมายเหตุ</u>

ตำแหน่งของ Internal Memory คือ 0000H ถึง 0FFFH

ตำแหน่งของ External Memory คือ 0000H ถึง FFFFH

ตำแหน่งของ External Program Memory คือ 0000H ถึง FFFFH

2.9 การกำหนดตำแหน่งแอดเดรสของ 8255

1. เลือกไปที่ Tools/Address 8255
2. ที่ Address 8255 จะมีเมนูย่อย 2 เมนูคือ Interface 1 กับ Interface 3
3. ให้เลือกการกำหนดค่ากับชุดอินเตอร์เฟสที่ต้องการ
4. จะปรากฏหน้าต่างของการกำหนดค่าตำแหน่งแอดเดรสของ 8255
5. ให้ใส่ค่าแอดเดรสที่ต้องการจำนวน 4 ตำแหน่ง
6. เมื่อใส่ค่าเรียบร้อยแล้ว คลิกที่ปุ่ม OK

2.10 การใช้งานชุดอินเตอร์เฟส 1

1. กำหนดค่าตำแหน่งแอดเดรสของ 8255 ตามที่ต้องการ
2. จากวงจรกำหนดให้ที่ PA ต่อเป็น row และที่ PB ต่อเป็น column ของ LED Matrix ขนาด 8 X 8

3. และจากวงจรกำหนดให้ที่ PC ต่อเป็น Switch 8 ตัว
4. เขียนโปรแกรมตามที่ต้องการ

2.11 การใช้งานชุดอินเตอร์เฟส 2

การใช้งานชดอินเตอร์เฟส 2 เป็นการต่อใช้งานกับพอร์ตของ 8051 จำลอง โดยได้กำหนดให้มีการต่อของพอร์ตต่างๆ ดังนี้

1. P0 ต่อเป็น LED 8 ดวง
2. P1 ต่อเป็น LED 8 ดวง และ 7-Segment
3. P3 ต่อเป็น DIP Switch 8 bit

ผู้ใช้สามารถเขียนโปรแกรมสั่งให้เกิดการแสดงผลหรือรับค่าได้ตามการกำหนดของฮาร์ดแวร์ที่จำลองขึ้นในชุดอินเตอร์เฟสได้

2.12 การใช้งานชุดอินเตอร์เฟส 3

การใช้งานชุดอินเตอร์เฟส 3 เป็นการใช้งานติดต่อกับ 8255 มีวิธีการดังนี้

1. กำหนดค่าตำแหน่งแอดเดรสให้กับ 8255 ก่อน
2. กำหนดให้พอร์ตต่างๆ ของ 8255 เป็นอินพุตหรือเอาต์พุต โดยมีตัวเลือกดังนี้
    2.1 PA,PB ต่อเป็น อินพุต เลือก DIP Switch 8 bit
    2.2 PA,PB ต่อเป็น เอาต์พุต เลือก LED หรือ 7-segment
    2.3 PC(lower), PC(upper) ต่อเป็น อินพุต เลือก DIP Switch 8 bit
    2.4 PC(lower), PC(upper) ต่อเป็น เอาต์พุต เลือก LED
3. เขียนโปรแกรมสั่งงานตามที่ต้องการ

บทที่ 3
โปรแกรมพื้นฐาน

3.1 ตัวอย่างการใช้โปรแกรม

1. เปิดโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS - 51 โดยเลือกไปที่ Start / Program / Mcs51sim
2. เขียนโปรแกรมไฟวิ่ง จากบิต 0 ไป บิต 7 ที่พอร์ต P1 ดังนี้

```
        ORG     0000H
START:  MOV     A,#01H
LOOP:   MOV     P1,A
        ACALL DELAY
        RL      A
        SJMP    LOOP
DELAY:  MOV     R0,#20H
        DJNZ    R0,$
        RET
        END
```

3. ทำการบันทึกโปรแกรมโดยเลือกไปที่ File / Save หรือ คลิกที่ปุ่ม Save ที่ Speed bar
4. ทำการคอมไพล์โปรแกรมโดยเลือกไปที่ Run / Sax51 compile หรือคลิกที่ปุ่ม compile
5. เลือกโหมดในการรันโปรแกรม โดยเลือกไปที่ Run / Chang Mode หรือคลิกที่ปุ่ม compile ที่ Speed barมี 2 โหมด คือ Sim / Step
6. สั่ง Run โปรแกรม โดยการเลือกไปที่ Run / Run หรือ คลิกที่ปุ่ม Run ที่ Speed bar
7. ถ้าเลือกการทำงานในโหมด Step เมื่อสั่งรันโปรแกรมแล้วให้คลิกที่ปุ่ม Step เพื่อทำการรันโปรแกรมทีละคำสั่ง
8. สังเกตการเปลี่ยนแปลงที่เกิดขึ้นได้จาก หน้าต่างหน่วยความจำ, หน้าต่างรีจิสเตอร์, หน้าต่างอินเตอร์เฟสที่ 2

3.2 การตรวจแก้ข้อผิดพลาดของโปรแกรม

ในการตรวจแก้ไขข้อผิดพลาดหรือการดีบัค ที่เกิดขึ้นจากโปรแกรมสามารถตรวจแก้ไขได้ง่ายโดยการดูการแจ้งข้อความบรรทัดที่ผิดพลาดจากหน้าต่าง Mesg ในส่วนของ Compile list เมื่อทราบว่าบรรทัดใดผิดแล้ว ก็ให้กลับไปที่หน้าต่างอิดิเตอร์ ทำการแก้ไขโปรแกรมส่วนที่ผิดพลาด แล้วทำการคอมไพล์โปรแกรมใหม่อีกครั้งหนึ่ง จากนั้นก็ทำการรันโปรแกรมดูผลที่เกิดขึ้น

ภาคผนวก ข

ใบงานการทดลอง

การทดลองที่ 1
การใช้งานรีจิสเตอร์ของ MCS-51

วัตถุประสงค์

1. เพื่อให้รู้จักรีจิสเตอร์ของ MCS-51
2. สามารถนำรีจิสเตอร์ไปใช้งานในการเขียนโปรแกรมได้

ทฤษฎี

รีจิสเตอร์ คือ ตัวเก็บข้อมูลชั่วคราวเพื่อที่จะนำข้อมูลไปกระทำขั้นตอนในการคำนวนหรือกระทำอย่างอื่นกับข้อมูลในรีจิสเตอร์ ซึ่งรีจิสเตอร์มีหน้าที่ และ การทำงานแตกต่างกัน บางตัวเป็น ริจิสเตอร์ที่สามารถเฉพาะอย่าง บางตัวใช้งานได้ทั่วไป ใน MCS - 51 แล้วรีจิสเตอร์ที่ไว้ใช้งานดังนี้

รีจิสเตอร์ A (Accumulator)

เป็นรีจิสเตอร์ ขนาด 8 บิตซึ่งคำสั่งส่วนใหญ่จะอ้างถึงรีจิสเตอร์ตัวนี้ โดยถือค่าในรีจิสเตอร์ตัวนี้เป็นตัวตั้ง และ รับค่าผลลัพธ์ที่ได้จากการใช้คำสั่งทางคณิตศาสตร์ การบวก, การลบ, การคูณ และการหาร เข้ามาเก็บไว้ที่รีจิสเตอร์ตัวนี้ยังใช้เป็นตัวกระทำทางลอจิก และใช้เป็นตัวถ่ายเทข้อมูลในการติดต่อกับอุปกรณ์ภายนอก และหน่วยความจำ รวมถึงการใช้งานในการคำนวนค้นหา และตรวจสอบตารางข้อมูล

รีจิสเตอร์ B

เป็นรีจิสเตอร์ พิเศษใช้งานเฉพาะอย่าง ใช้สำหรับคำสั่งใน การคูณ และการหาร โดยใช้เป็นตัวคูณ และ ตัวหาร ตามูลำดบ และเมื่อทำการคูณ หรือ การหารเสร็จแล้วก็จะใช้เก็บผลลัพธ์ตัวที่สองของการคูณ และ ตัวเศษของการหาร

รีจิสเตอร์ PSW (Program Status Word)

เป็นรีจิสเตอร์ ที่จะบอกสถานะในการทำงานของคำสั่งต่างๆ และ ใช้เป็นตัวเลือก Banks ของ Register Bank ซึ่งมีอยู่ 4 Banks ด้วยกัน ความหมายของรีจิสเตอร์ PSW มีดังนี้

(MSB) (LSB)

CY	AC	F0	RS1	RS0	OV	-	P

CY เป็นแพล็กตัวทด โดยจะเป็น “1” หรือ “0” ได้ฮาร์ดแวร์ หรือ ซอฟต์แวร์ ระหว่างผลลัพธ์หลังจากการใช้คำสั่งทางคณิตศาสตร์ หรือ ทางลอจิก เช่น

	10001001	89
	+ 10010100	+94
CY[1]	00011101	1d

	10010100	94
	+01001001	+49
CY[0]	11011101	dd

CY ยังใช้เป็นตัวประมวลผลแบบบิตอีกด้วย

AC เป็นแพล็กตัวทดของ Auxiliar โดยจะเป็น “1” หรือ “0” ได้ฮาร์ดแวร์ หลังจากการบวกและ การลบ จะแสดงผลจากการทด หรือ ยืมจากบิตที่ 3 ของรีจิสเตอร์ A เช่น

	10010100	94
	+01101000	+68
AC[1]	11111100	FC

	11000100	94
	+00100000	+20
AC[0]	11100100	E4

F0 เป็นแพล็ก 0 จะเป็น “1” หรือ “0” ได้ด้วย ซอฟต์แวร์ซึ่งผู้ใช้สามารถกำหนดการใช้งานและ กำหนดสถานะที่แพล็กนี้

RS1,RS0 เป็นแพล็กที่เป็นตัวกำหนด Banks ของ Register Bank โดยการเปลี่ยนค่าโดยซอฟแวร์ ดังจะแสดงต่อไปนี้

RS1	RS0	BANKS	ADDRESS
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

Register Bank นี้ได้แบ่งออกเป็น 4 Banks โดยแต่ละ Banks จะแบ่งออกเป็นรีจิสเตอร์ขนาด 8 บิต เป็น 8 ตัวด้วยกันคือ R0, R1, R2, R3, R4, R5, R6, R7 โดยจะใช้หน่วยความจำภายในตามตางราง และ เมื่อเปิดเครื่อง หรือ ทำการรีเซ็ต จะชี้ที่ Banks 0 เสมอ

OV แฟล็ก Overflow จะเป็น "1" หรือ "0" ด้วยฮาร์ดแวร์ระหว่างการใช้งานคำสั่งนี้จะแสดงผลถึงการเกิดลักษณะ Overflow เช่น

	01111111	+127
	+01000000	+ 64
OV[1]	10000011	+191
	00100000	+32
	+00100000	+32
OV[0]	01000000	+64

P แฟล็ก Parity จะเป็น "1" หรือ "0" ด้วยฮาร์ดแวร์ จะเปลี่ยนไปตามคำสั่งที่ทำงานกับรีจิสเตอร์ A โดยจะตรวจสอบว่า มี "1" ใน Accumulator อยู่เป็นคู่ หรือ เป็น คี่ โดย

ถ้ามี "1" เป็นคู่ (EVEN) P = "0"

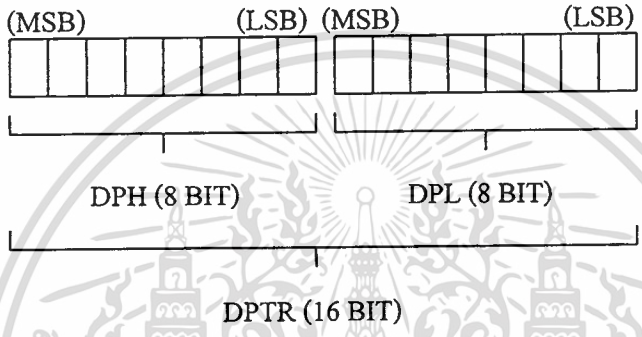
ถ้ามี "0" เป็นคี่ (ODD) P = "1"

รีจิสเตอร์ SP (Stack Pointer)

เป็นรีจิสเตอร์ขนาด 8 บิต ทำหน้าที่ใช้ชี้ตำแหน่งของสแตค ซึ่งสแตคมีไว้ใช้เพื่อเป็นที่เก็บข้อมูลชั่วขณะข้อมูลนี้ อาจจะมาจากรีจิสเตอร์ต่างๆ ข้อมูลที่จะเก็บลงในสแตคนี้อาจถูกเก็บเข้า หรือถูกนำออกโดยอัตโนมัติ เช่นเมื่อใช้คำสั่งในการเรียกโปรแกรมหลัก และ จะส่งค่าตำแหน่งเมื่อหลดจากโปรแกรมย่อย โดยใช้คำสั่ง RET หรือ RETI สามารถเก็บข้อมูลลงสแตค หรือ นำข้อมูลออกจากสแตคโดยใช้คำสั่ง PUSH และ POP

รีจิสเตอร์ DPTR (Data Pointer)

เป็นรีจิสเตอร์ขนาด 16 บิต และย่อยออกเป็น 2 ตัว คือ DPH และ DPL เป็นรีจิสเตอร์ขนาด 8 บิต ซึ่งทั้งสองตัวใช้ได้อย่างอิสระ หรือใช้รวมกันทั้งสองตัว 16 บิต ก็ได้ ทำให้สะดวกต่อการ Increment หรือ Decrement เพื่อใช้ในการเป็นฐานของตำแหน่งของข้อมูล ในการใช้คำสั่งเกี่ยวกับตารางข้อมูล และ ชี้ตำแหน่งของหน่วยความจำภายนอก



PORT 0 TO 3

เป็นรีจิสเตอร์ขนาด 8 บิต ซึ่งทำหน้าที่รีจิสเตอร์แลทช์ค่าของ P0, P1, P2 และ P3 ตามลำดับ

รีจิสเตอร์ SBUF (Serial Data Buffer)

เป็นรีจิสเตอร์ ทางผ่านในการติดต่อข้อมูลแบบอนุกรม ภายในแบ่งออกเป็นรีจิสเตอร์สองตัว ตัวหนึ่งเป็นบัฟเฟอร์การส่ง และอีกตัวเป็นบัฟเฟอร์การรับ เมื่อข้อมูลเข้า SBUF จะเป็น ข้อมูลในการส่งข้อมูลแบบอนุกรม และเมื่อนำข้อมูลออกจาก SBUF จะเป็นการรับข้อมูลจากบัฟเฟอร์ตัวรับข้อมูลอนุกรม

รีจิสเตอร์ CAPTURE

เป็นรีจิสเตอร์ที่เพิ่มเติมเข้ามาของ ไอซีเบอร์ 8032/8052 จะมีคู่รีจิสเตอร์ RCAP2H, RCAP2L ซึ่งเป็นรีจิสเตอร์ของตัวเวลาหมายเลข 2

รีจิสเตอร์ควบคุม (Control Register)

ในกลุ่ม SFR (Spactial Function Register) มีรีจิสเตอร์ที่ใช้ควบคุมการทำงานของ CPU ซึ่งมีใน CPU ได้ออกแบบให้มีวงจรในการติดต่อกับอุปกรณ์ภายนอก และอุปกรณ์อำนวยความสะดวกไว้ภายใน CPU อยู่แล้ว ก่อนที่จะใช้งานต้องทำการควบคุม หรือ กำหนดการทำงานของอุปกรณ์เหล่านั้น โดยการเปลี่ยนค่าภายในกลุ่ม โดยมีรีจิสเตอร์ที่จะควบคุมดังนี้

IP, IE เป็นการจัดสถานะของการใช้งานในระบบอินเตอร์รัพต์

TMOD, TCON, T2CON เป็นการจัดการทำงานของ Timer

SCON เป็นการจัดการทำงานของ การส่ง และการรับข้อมูลแบบอนุกรม

PCON เป็นการจัดโหมดการทำงานของ CPU ในการประหยัดพลังงาน

การทดลอง

1. เปิดโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ MCS - 51 เลือก

   Start -> Programs -> Mcs51Sim

2. เลือกไปที่หน้าต่างอิดิเตอร์
3. ให้ทำการป้อนโปรแกรมที่ 1 ดังนี้

```
        ORG   0000H
X       EQU   60H
Y       EQU   90H
LOOP:   MOV   A,#X    ; กำหนดค่าตัวตั้ง
        ADDC  A,#Y    ; ทำการบวกกันระหว่างตัวตั้งกับตัวบวก
        MOV   P1,A    ; ส่งผลการคำนวณออก Port 1
        SJMP  $
```

โปรแกรมที่ 1

4. ทำการบันทึกโปรแกรม โดยเลือกไปที่ เมนู File ของหน้าต่างหลัก แล้วเลือก Save (ถ้าไม่ต้องการเปลี่ยนชื่อไฟล์) หรือเลือก Save As (ถ้าต้องการเปลี่ยนชื่อไฟล์) จากนั้นให้ป้อนชื่อไฟล์ที่ต้องการ (เราอาจบันทึกโปรแกรมอีกวิธีหนึ่งคือการกดปุ่ม Save ที่ Speed bar)
5. เลือกโหมดการทำงานของโปรแกรม ซึ่งมีอยู่ 2 โหมดคือ Sim กับ Step โดยเลือกไปที่เมนู Run ของหน้าต่างหลัก แล้วเลือก Chang Mode ในที่นี้ให้เราเลือกไว้ที่โหมด Step (หรืออาจใช้อีกวิธีหนึ่งคือการกดปุ่ม Sim/Step ที่ Speed bar)
6. ทำการคอมไพล์โปรแกรมโดยเลือกไปที่ เมนู Run แล้วเลือก Sxa51 compile (หรืออีกวิธีหนึ่งคือกดปุ่ม Compile ที่ Speed bar)
7. ทำการสั่งให้โปรแกรมทำงานโดยเลือกไปที่ เมนู Run แล้วเลือก Run (หรืออีกวิธีหนึ่งคือกดปุ่ม Run ที่ Speed bar)

8. ถ้าต้องการยกเลิกโปรแกรมให้เลือกที่ เมนู Run แล้วเลือก Stop (หรืออีกวิธีหนึ่งคือกดปุ่ม Stop ที่ Speed bar)
9. ทำการทดลองให้โปรแกรมทำงานโดยกดที่ปุ่ม Step และ ลองเปลี่ยนค่าของตังตั้ง และตัวบวก แล้วสังเกตการเปลี่ยนแปลงของรีจิสเตอร์ A และ PSW แล้วบันทึกค่าลงในตาราง (เมื่อมีการแก้ไขโปรแกรมจะต้องทำการคอมไพล์โปรแกรมใหม่ทุกครั้ง)

X	Y	รีจิสเตอร์ A	PSW			
			CY	AC	OV	P
60H	90H					
50H	30H					
47H	30H					
60H	33H					
0AH	B0H					
65H	43H					
20H	11H					

10. ทำเหมือนขั้นตอนที่ 1 ถึง 8 แต่เปลี่ยนโปรแกรมใหม่ดังนี้

```
        ORG   0000H
        MOV   DPTR,#TABLE      ; กำหนดให้ DPTR = TABLE
LOOP:   INC   DPL              ; เพิ่มค่า DPL
        MOVX  A,@DPTR
        MOV   P1,A
        SJMP  LOOP
        ORG   8450H
TABLE:  DB    1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 0AH, 0BH, 0CH
```

โปรแกรมที่ 2

11. ให้ทำการทดลองให้โปรแกรมทำงานทีละSTEPแล้วสังเกตการเปลี่ยนแปลงของ Port 1
12. ให้ทำการป้อนโปรแกรมที่ 3

```
ORG   0000H
MOV   0D0H,#00H     ; กำหนดให้ PSW = 0000 0000B
MOV   R0,#01H       ; R0 = 01H
MOV   0D0H,#08H     ; กำหนดให้ PSW = 0000 1000B
MOV   R0,#02H       ; R0 = 02H
MOV   0D0H,#10H     ; กำหนดให้ PSW = 0001 0000B
MOV   R0,#03H       ; R0 = 03H
MOV   0D0H,#18H     ; กำหนดให้ PSW = 0001 1000B
MOV   R0,#04H       ; R0 = 04H
END
```

โปรแกรมที่ 3

13. ให้ทำการสั่งให้โปรแกรมทำทีละคำสั่งแล้วสังเกต การเปลี่ยนแปลงในหน่วยความจำภายในโดยสังเกตที่แอดเดรส 00, 08, 10, 1FH จะมีการเปลี่ยนแปลงในการใช้คำสั่งในช่วงใดบ้างบันทึกค่าลงตาราง

	00H	08H	10H	1FH	PSW
	00H	00H	00H	00H	00H
MOV 0D0H,#00H					
MOV R0,#01H					
MOV 0DOH,#08H					
MOV R0,#02H					
MOV 0D0H,#10H					
MOV R0,#03H					
MOV 0D0H,#18H					
MOV R0,#04H					

คำถาม

1. รีจิสเตอร์ตัวใดใช้งานในกลุ่มคำสั่งมากที่สุด
2. จงเขียนโปรแกรมที่ใช้กำหนดค่าต่อไปนี้

   DPTR = 6000H

   R0 (Regiater Banks 3) = 60H

   A = R0 (Register Banks 3)

การทดลองที่ 2
การอ้างตำแหน่งของ MCS - 51

วัตถุประสงค์

1. เพื่อศึกษาการอ้างตำแหน่งของ MCS - 51
2. เพื่อนำเอาหลักการของการอ้างตำแหน่งไปใช้ในการเขียนโปรแกรม

ทฤษฎี

คำสั่งภาษาแอสเซมบลี้ของ MCS - 51 จะประกอบด้วยนิวมอนิกรหัสการทำงานหนึ่งไบต์ และตัวโอเปอร์แรนด์อีก ตั้งแต่ 0 - 3 ไบต์ โดยการใช้คำสั่งสัญญลักษณ์ต่างๆ ที่ถูกแบ่งด้วย (,) คำสั่งที่ใช้ตัวโอเปอร์แรนด์สองตัว สัญญลักษณ์จะให้ตัวแรกเป็นตัวรับการถ่ายเทข้อมูล (Destination) ส่วนตัวหลังจะเป็นตัวส่งข้อมูล (Source) ส่วนที่ใช้คำสั่งต่างๆ ส่วนใหญ่นั้น จะทำงานโดยใช้ แอกคิวมูเลเตอร์ เป็นตัวหลักในการเป็นตัวส่งข้อมูลในโอเปอร์แรนด์ และ จะเป็นตัวรับผลลัพธ์ที่เกิดขึ้นหลังจากการทำงานคำสั่งนั้นๆ ด้วยการให้ตัวอักษร 'A' เป็นตัวกำหนดตัวส่งข้อมูลหรือรับการถ่ายเทข้อมูลในโอเปอร์แรนด์ เช่น

MOV (Destination) , (Source)

จากตัวอย่าง MOV เป็นคำสั่งที่จะเอาค่าจาก Destination ไปกระทำกับ Source ซึ่งที่ยกตัวอย่างนี้ เป็นการเอาข้อมูลจาก Source ไปเก็บไว้ที่ Destination ตัวอย่างเช่น

MOV A,R0

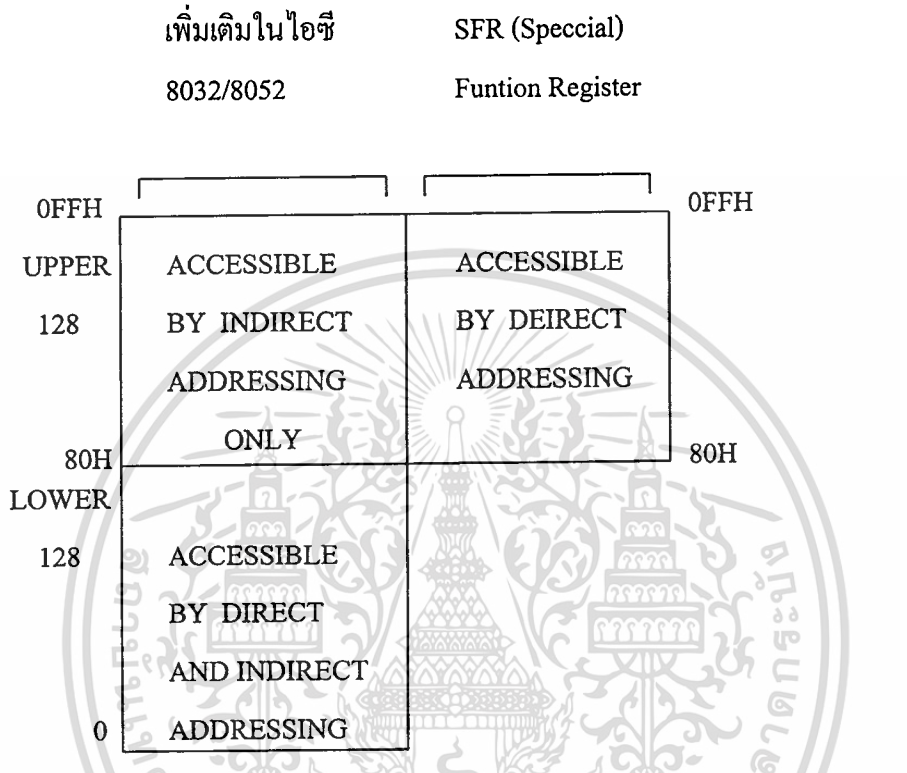
จากตัวอย่างเป็นการกำหนดให้ รีจิสเตอร์ A เป็น Destination และให้ R0 เป็น Source ซึ่งคำสั่งจากตัวอย่างเป็นการเอาข้อมูลที่ รีจิสเตอร์ R0 ไปเก็บไว้ที่ รีจิสเตอร์ A

การใช้ลักษณะของ Source ในคำสั่งตัวอย่างนี้ สามารถที่จะเลือกใช้ได้ตามลักษณะของการกำหนดเลขที่อยู่ได้ 4 โหมดด้วยกัน คือ

1. การกำหนดเลขที่อยู่รีจิสเตอร์ (Register Addressing)
2. การกำหนดเลขที่อยู่โดยตรง (Direct Addressing)
3. การกำหนดเลขที่อยู่รีจิสเตอร์โดยทางอ้อม (Indirect Addressing)
4. การกำหนดเลขทีอยู่ข้อมูลโดยตรง (Imediate Addressing)

โหมดสามโหมดแรกเป็นการเข้าถึง แรมภายใน (Internal RAM) ของ MCS-51 และรีจิสเตอร์ต่างๆ ภายใน ตามโครงสร้างของ MCS-51 นอกจากจะใช้เป็นตัวส่งข้อมูลในโอเปอร์แรนด์

ฟิลด์แล้ว ยังใช้เป็นตัวรับการถ่ายเทข้อมูลในโอเปอร์แรนด์ฟิลด์อีกด้วย ส่วนโหมดที่ 4 เนื่องจากโอเปอร์แรนด์ฟิลด์ข้อมูลคงที่ จึงเป็นตัวส่งข้อมูลคงที่ เพียงอย่างเดียวเท่านั้น



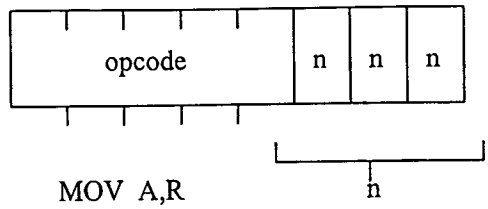
รูปที่ 2.1 การกำหนดตำแหน่งของ แรมภายใน (Internal RAM)

จากรูปที่ 2.1 แสดงการกำหนดเลขที่อยู่ของแต่ละโหมดซึ่ง แรมภายในจะอ้างแตกต่างกัน

1. การกำหนดเลขที่อยู่รีจิสเตอร์ (Register Addressing)

การกำหนดเลขที่อยู่รีจิสเตอร์ จะให้ข้อมูลการใช้รีจิสเตอร์กลุ่ม (Register Banks) ที่ถูกเลือกจากการกำหนด Banks ที่จะใช้งาน โดยแต่ละ Banks จะมีรีจิสเตอร์อยู่ 8 ตัว และใช้บิตต่ำสามบิตแรกของรหัสคำสั่งเป็นตัวกำหนดเลือกการใช้รีจิสเตอร์ที่จะใช้งานตัวใดตัวหนึ่งใน 8 ตัวทั้งสามบิตจะรวมอยู่ในรหัสออบโค้ดกับโอเปอร์แรนด์ การกำหนดเลขที่อยู่นี้จะเป็นคำสั่งแค่ 1 Byte เท่านั้น ดังรูปที่ 2.2

คำสั่ง MOV A , Rn

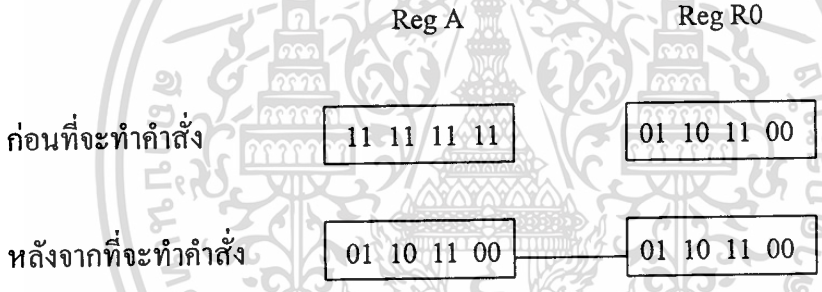


รูปที่ 2.2

<u>ตัวอย่าง</u>

ให้ R0 = 01101100b,A = 11111111b

ใช้คำสั่ง MOV A,R0

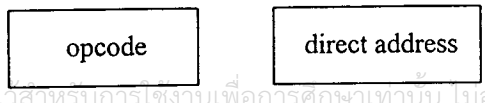


ในภาษาแอสเซมบลี้ของ MCS - 51 จะชี้การทำงานโหมดกำหนดเลขที่อยู่โดยรีจิสเตอร์ด้วยสัญญลักษณ์ Rn โดย n จะแทนค่า 0-7 (R0, R1, R2, R3, R4, R5, R6, R7)

2. การกำหนดเลขที่อยู่โดยตรง (Direct Addressing)

การกำหนดเลขที่อยู่โดยตรงเป็นวิธีเดียวที่จะเข้าถึงข้อมูลทางฮาร์ดแวร์รีจิสเตอร์ เช่น กลุ่มของ SFR และสามารถกำหนดโดยการกำหนดเลขที่อยู่โดยตรงบริเวณตำแหน่งต่างๆ ของหน่วยความจำภายใน (Internal RAM) จำนวน 128 Byte ด้วยการใช้ไบต์โอเปอร์แรนด์ต่อจากออปโค้ดของคำสั่งเป็นตัวกำหนดตำแหน่งที่ถูกใช้ ดังรูป

คำสั่ง ADD A , direct



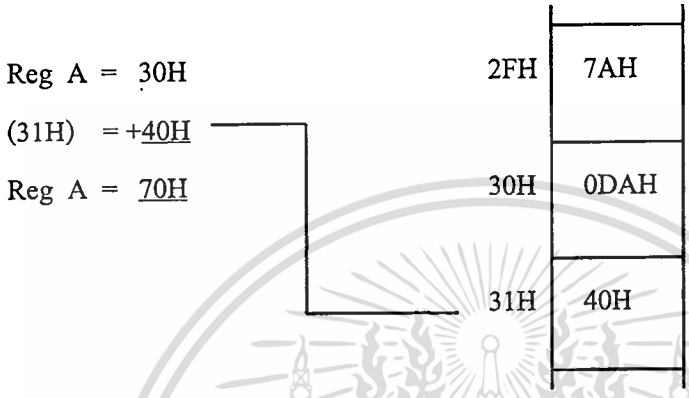
ADD A, direct

ตัวอย่าง

ใช้คำสั่ง ADD A,31H

ให้ A = 30H

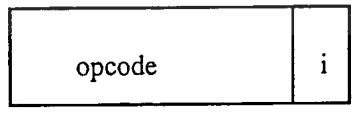
เมื่อทำคำสั่ง



3. การกำหนดเลขที่อยู่รีจิสเตอร์โดยทางอ้อม (Indirect Addressing)

การกำหนดเลขที่อยู่โดยทางอ้อม ของ MCS - 51 จะใช้ข้อมูลใน R0 หรือ R1 โดยแบงค์ที่ถูกเลือกทำงานเป็นดัชนี หรือ ตัวชี้ตำแหน่งข้อมูลภายใน 250 Byte แบ่งเป็นบล็อกต่ำ จำนวน 128 Byte ของแรมภายใน และ บล็อกสูงอีก 128 Byte ของแรมภายใน ซึ่งมีเฉพาะในไอซีเบอร์ 8032/8052 เท่านั้น หรือ ใช้เป็นตัวชี้จำนวน 256 Byte ของหน่วยความจำภายนอกการกำหนดใช้แรมภายในจะสามารถกำหนดภายในด้วย R0 และ R1 ซึ่งจะเป็นตัวเลือกด้วยรหัสคำสั่งออปโค้ดที่หลักต่ำสุด (LSB) ดังรูป

คำสั่ง ADD A,@Ri



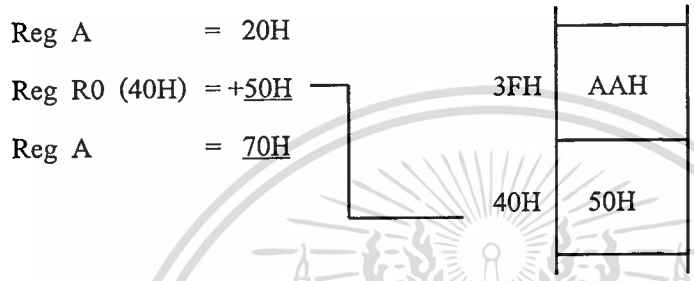
ADD A,@R I

ตัวอย่าง

ใช้คำสั่ง ADD A,@R0

ให้ R0 = 40H ก่อนที่จะคำสั่ง A = 20H

เมื่อทำคำสั่ง



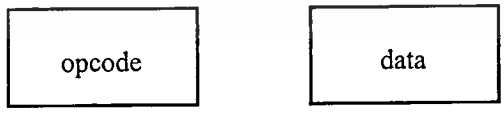
ใน MCS - 51 ภาษาแอสเซมบลี้ที่กำหนดการใช้ลักษณะนี้ จะใช้สัญลักษณ์ "@" นำหน้า R0 และ R1

การเข้าถึงข้อมูลภายนอกให้ได้ถึง 64 Kbyte จากหน่วยความจำข้อมูลภายนอกที่ว่างจะใช้รีจิสเตอร์คู่ของ Data Pointer (DPTR) ซึ่งมีขนาด 16 บิต

4. การกำหนดเลขที่อยู่ข้อมูลโดยตรง (Immediate Addressing)

การกำหนดเลขที่อยู่ข้อมูลโดยตรง เป็นการกำหนดโอเปอร์แรนด์เป็นค่าคงที่ซึ่งเป็นข้อมูลที่ต้องการนำมาใช้จริงโดยการกำหนดเลขที่อยู่นี้ใช้สัญญลักษณ์ "#" เป็นตัวกำหนดว่าภายในโอเปอร์แรนด์ฟิลด์ เป็นค่าคงที่ ดังรูป

คำสั่ง MOV A,#data

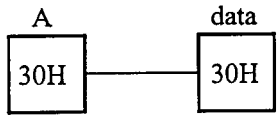


MOV A,# data

ตัวอย่าง

ใช้คำสั่ง MOV A,#30H

เมื่อทำคำสั่ง



การทดลอง

1. ให้ทำการป้อนโปรแกรมที่ 1

```
        ORG   0000H
        MOV   A,#60H      ; A = 60H
LOOP:   INC   A           ; A = A+1
        MOV   B,A         ; B = A
        MOV   R0,A        ; R0 = A
        INC   R0          ; R0 = R0+1
        MOV   DPL,R0      ; DPL = R0
        MOV   DPH,A       ; DPH = A
        SJMP  LOOP
```

โปรแกรมที่ 1

2. ให้ทำการสั่งให้โปรแกรมทำงานทีละ STEP และให้สังเกตค่าของรีจิสเตอร์
3. ป้อนโปรแกรมที่ 2

```
        ORG   0000H
        MOV   R0,#30H     ; R0 = 30H
        MOV   30H,#51H    ;(30H) = 51H
        MOV   31H,#60H    ;(31H) = 60H
        MOV   A,@R0       ; A = ((R0))
LOOP:   MOV   R7,A        ;R7 = A
        INC   @R0         ; (R0) = (R0) + 1
```

```
MOV   A,@R0        ; A = ((R0))
ADD   A,R7         ; A = A+R7
SJMP  LOOP
```

โปรแกรมที่ 2

4. ให้ทำการสั่งให้โปรแกรมทำงานทีละ STEP และให้สังเกตค่าของรีจิสเตอร์

คำถาม

1. จงเขียนโปรแกรมที่อ้างตำแหน่งของข้อมูลโดยทางอ้อมโดยใช้ในการกำหนดค่าในหน่วยความจำ 30H ถึง 35H โดยการเอาค่า 0, 1, 2, 3, 4, 5 ใส่ไว้ในหน่วยความจำตามลำดับ
2. จงยกตัวอย่างการอ้างตำแหน่งโดยตรง, รีจิสเตอร์, ทางอ้อม และ ข้อมูลโดยตรงมาอย่างละสองตัวอย่าง

การทดลองที่ 3
การใช้คำสั่งในการโอนย้ายข้อมูลของ MCS - 51

วัตถุประสงค์

1. เพื่อให้เข้าใจการโอนย้ายข้อมูลของ MCS-51
2. ให้เข้าใจลักษณะการใช้คำสั่งการโอนย้ายข้อมูลซึ่งมีอยู่หลายแบบคำส่ง
3. สามารถนำเอาคำสั่ง MOV, MOVC, MOVX, XCH, และ XCHD ใช้งานในการเขียนโปรแกรม

ทฤษฎี

คำสั่งโอนย้ายข้อมูลเป็นคำสั่งที่สำคัญคำสั่งหนึ่งซึ่งใน MCS-51 ได้แบ่งคำสั่งในการโอนย้ายข้อมูลออกตามหน้าที่ที่จะทำการโอนย้ายข้อมูลระหว่าง Destination กับ Source และการโอนย้ายข้อมูลกับหน่วยความจำ ใน MCS-51 ได้แบ่งหน่วยความจำออกเป็น สองส่วนด้วยกันคือ หน่วยความจำภายใน (Internal Memory) และหน่วยความจำภายนอก (External Memory) หน่วยความจำภายใน และหน่วยความจำภายนอก ยังแบ่งออกได้ดังนี้

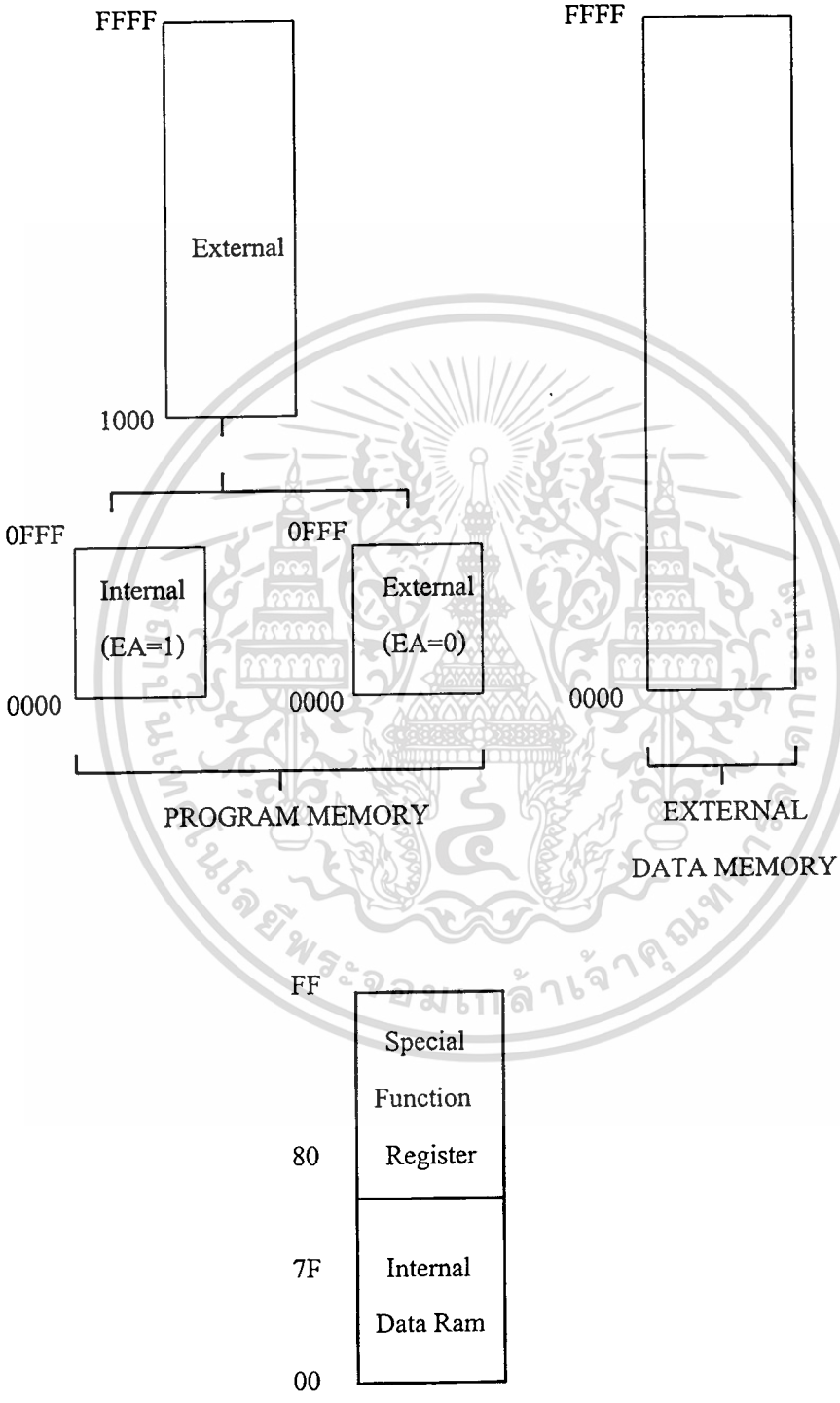
หน่วยความจำภายใน

1. หน่วยความจำที่ใช้เป็นข้อมูล (Internal Data RAM)
2. หน่วยความจำที่ใช้เป็นรีจิสเตอร์พิเศษ (SFR : Spacial Funtion Register)
3. หน่วยความจำที่เพิ่มเติมของไอซี 8032/8052

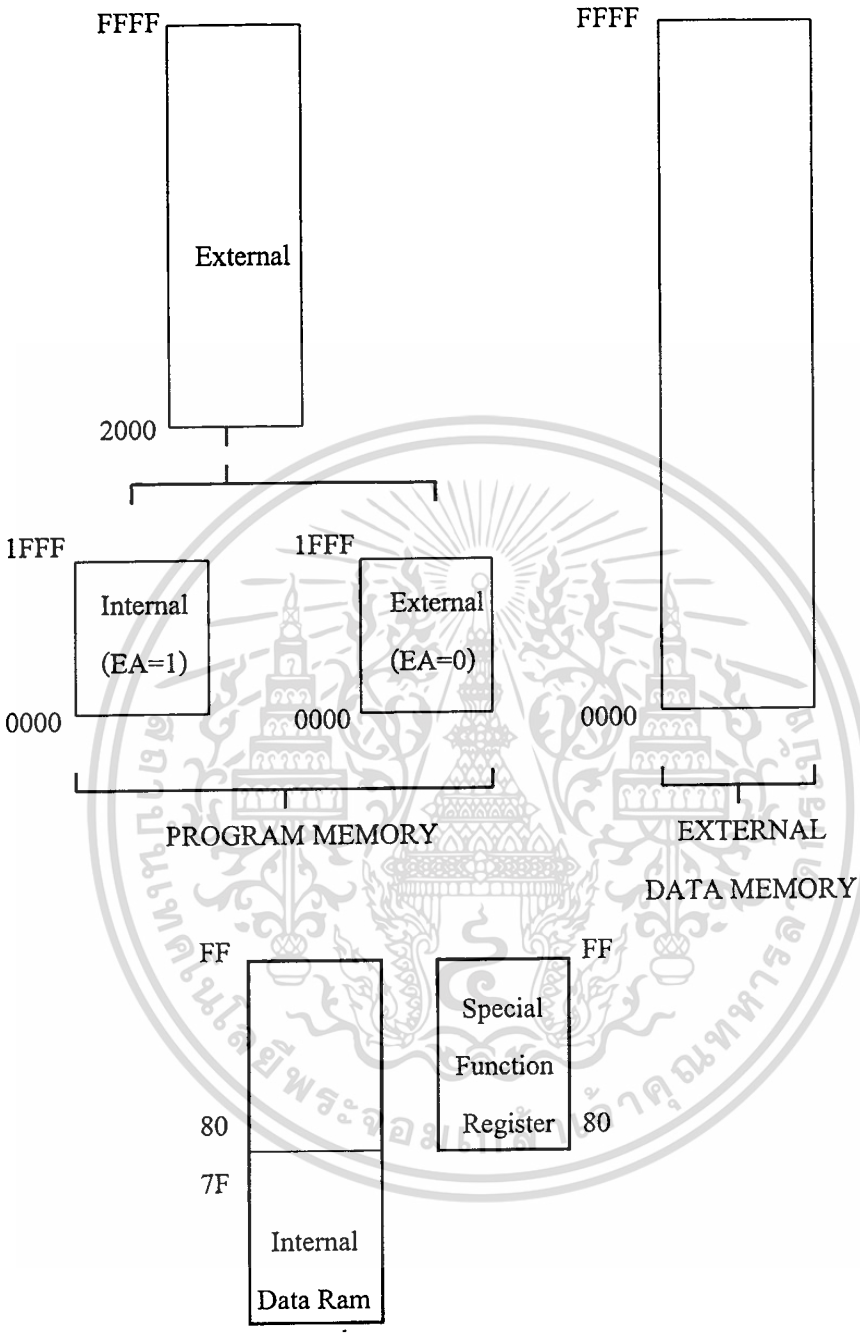
หน่วยความจำภายนอก

1. หน่วยความจำของโปรแกรม (Program Memory)
2. หน่วยความจำของข้อมูล (Data Memory)

เราสามารถแสดงรูปการแบ่งหน่วยความจำดังรูปต่อไปนี้



รูปที่ 3.1 หน่วยความจำของ MCS - 51 (ยกเว้น 8032/8052)



รูปที่ 3.2 หน่วยความจำของ 8032/8052

จากรูปของหน่วยความจำทั้งสองจะแยกหน่วยความจำออกเป็น 3 ส่วนด้วยกันคือ

1. หน่วยความจำภายใน (Internal Memory)
2. หน่วยความจำภายนอกที่ใช้เป็นโปรแกรม (Program Memory)
3. หน่วยความจำภายนอกที่ใช้เป็นข้อมูล (Data Memory)

ใน MCS - 51 ได้แยกการโอนย้ายข้อมูลที่จะใช้กับหน่วยความจำที่มีอยู่ดังนี้

เราจะใช้คำสั่ง MOV ในการใช้หน่วยความจำชนิดนี้ในการโอนย้ายข้อมผุลซึ่งการโอนย้ายข้อมูลยังแบ่งออกเป็น ดังนี้

1. การโอนย้ายข้อมูลระหว่าง รีจิสเตอร์ กับ รีจิสเตอร์

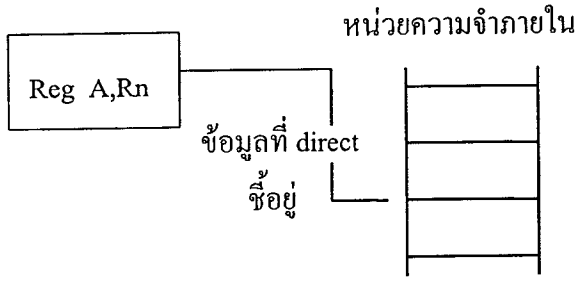
การโอนย้ายชนิดนี้เป็นการโอนย้ายระหว่างรีจิสเตอร์ กับ รีจิสเตอร์ ส่วนใหญ่จะใช้ รีจิสเตอร์ A กับรีจิสเตอร์ แบงค์ (Register Banks) รูปแบบคำสั่งมีดังนี้

MOV A,Rn เป็นการโอนย้ายข้อมูลจากรีจิสเตอร์แบงค์ที่เลือกแบงค์นั้นอยู่ไปเก็บไว้ที่ รีจิสเตอร์ A โดยที่ Rn แทนค่าด้วย R0, R1, R2, R3, R4, R5, R6, R7

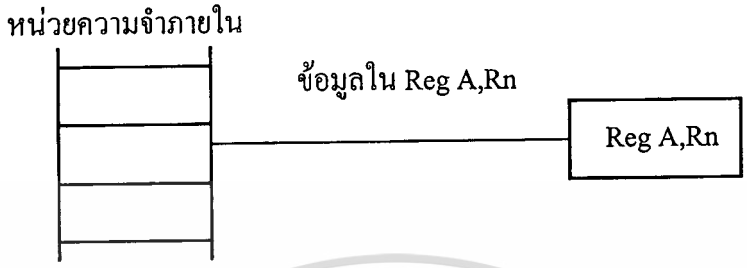
MOV Rn,A เป็นการโอนย้ายข้อมูลจากรีจิสเตอร์ A ไปเก็บไว้ที่รีจิสเตอร์แบงค์ที่เลือกแบงค์นั้นอยู่โดยที่ Rn แทนค่าด้วย R0, R1, R2, R3, R4, R5, R6, R7

2. การโอนย้ายข้อมูลระหว่าง รีจิสเตอร์ กับ หน่วยความจำ เป็นการย้ายข้อมูลระหว่าง รีจิสเตอร์ กับ หน่วยความจำภายในซึ่งมีรูปแบบคำสั่ง ดังนี้

MOV A,direct และ MOV Rn,direct เป็นการโอนย้ายข้อมูลจากหน่วยความจำที่ direct ชี้อยู่ภายในไปเก็บไว้ที่ รีจิสเตอร์ A หรือ รีจิสเตอร์ Rn (R0, R1, R2, R3, R4, R5, R, R7)



MOV direct,A และ MOV direct,Rn เป็นการโอนย้ายข้อมูลจากรีจิสเตอร์ A,Rn ไปเก็บที่หน่วยความจำภายใน ที่ direct ชี้อยู่



3. การโอนย้ายข้อมูลระหว่าง รีจิสเตอร์ กับ ค่าคงที่

เป็นการโอนย้ายข้อมูลกับค่าที่คงที่ไปไว้ที่ รีจิสเตอร์โดยที่คำสั่งนี้จะถือว่าค่าคงที่นี้เป็นข้อมูลซึ่งคำสั่งนี้ส่วนมากใช้เป็นคำสั่งในการกำหนดค่าให้รีจิสเตอร์ต่างๆ ก่อนที่จะใช้งานรูปแบบคำสั่งมีดังนี้

MOV A,#data และ MOV Rn,#data เป็นเอาค่าคงที่ของข้อมูล (data) ไปไว้ที่รีจิสเตอร์ A หรือ รีจิสเตอร์ Rn (Register banks R0, R1, R2, R3, R4, R5, R6, R7) ดังที่จะแสดงในรูป

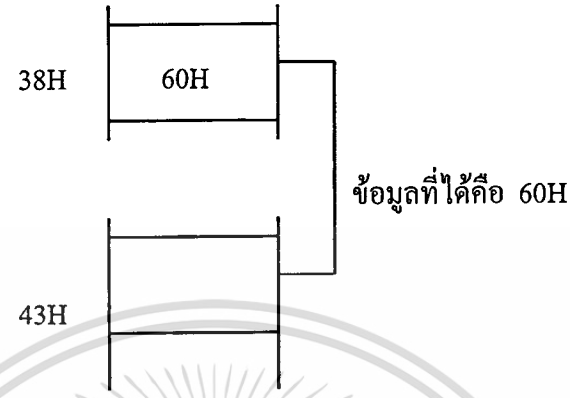
4. การโอนย้ายข้อมูลระหว่าง หน่วยความจำ กับ หน่วยความจำ

เป็นการโอนย้ายข้อมูลที่ MCS-51 ได้เพิ่มเข้ามาเพื่อสะดวกในการโอนย้ายข้อมูลระหว่างหน่วยความจำ กับ หน่วยความจำ รูปแบบคำสั่งมีดังนี้

MOV direct (destination),direct(source) เป็นการโอนย้ายข้อมูลโดยเอาข้อมูลจาก direct (source) ที่ชี้อยู่ ไปเก็บไว้ที่ direct (destination) ที่ชี้อยู่ดังตัวอย่าง

เมื่อทำงานคำสั่ง MOV 43H,38H

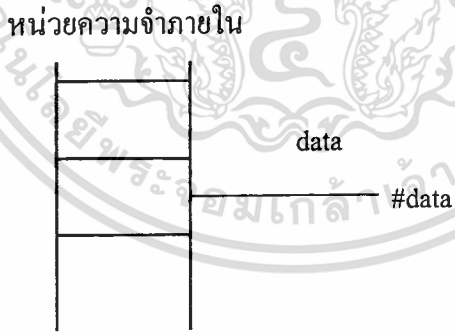
หน่วยความจำภายใน



5. การโอนย้ายข้อมูลระหว่าง หน่วยความจำ กับ ค่าคงที่

เป็นการโอนย้ายข้อมูลที่ MCS-51 ได้เพิ่มเข้ามาเพื่อสะดวกในการโอนย้ายข้อมูลระหว่างหน่วยความจำ กับ ค่าคงที่ของข้อมูล รูปแบบคำสั่งมีดังนี้

MOV direct,#data เป็นการโอนย้ายข้อมูลโดยเอาค่าคงที่ของข้อมูลไปเก็บไว้ที่หน่วยความจำที่ direct ชี้อยู่ ดังรูป



6. การโอนย้ายข้อมูลโดยทางอ้อม

เป็นการโอนย้ายข้อมูลโดยการเอาข้อมูลใน รีจิสเตอร์ แบงค์ แทนค่าด้วย Ri (R0, R1) ที่ใช้ในแบงค์นั้นอยู่ ไปชี้ตำแหน่งของหน่วยความจำภายใน ในคำสั่งโอนย้ายข้อมูลแบบนี้จะมีเครื่องหมาย "@" เป็นตัวบอกว่าเป็นการโอนย้ายข้อมูลโดยทางอ้อม รูปแบบคำสั่งมีดังนี้

MOV A,@Ri และ MOV direct,@Ri เป็นการโอนย้ายข้อมูลจากตำแหน่งในข้อมูลที่ Ri ชี้อยู่และ เอาข้อมูลจากตำแหน่งนั้น มาเก็บไว้ที่ รีจิสเตอร์ A หรือ ที่ direct ชี้อยู่

MOV @Ri,A MOV @Ri,direct และ MOV @Ri,#data เป็นการโอนย้ายข้อมูลจาก รีจิสเตอร์ A, ตำแหน่งที่ direct ชี้อยู่ และค่าของข้อมูลคงที่ไปเก็บไว้ที่ตำแหน่งที่ข้อมูลใน Ri ชี้ตำแหน่งที่จะเก็บ

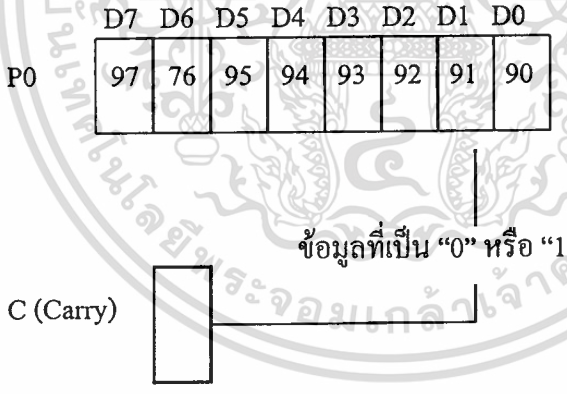
7. การโอนย้ายข้อมูลแบบ บิต

เป็นการโอนย้ายข้อมูลครั้งละบิตซึ่งใน MCS - 51 จะใช้ Carry อยู่ใน PSW (Program Status Word) เป็นตัวกลางในการโอนย้ายข้อมูลแบบบิตข้อมูลที่ได้จะเป็น "0" หรือ "1" เท่านั้น ซึ่งจะติดต่อกับหน่วยความจำภายในโดยอ้างเป็นตำแหน่งแบบบิต (Bit Addressing) ซึ่งแสดงในตารางใน Bit Address ในการอ้างมีรูปแบบคำสั่งดังนี้

MOV C,bit เป็นการโอนข้อมูล "0" หรือ "1" จากตำแหน่งของบิต (Bit Addressing) มาไว้ที่ Carry ดังตัวอย่าง

MOV C,P1.1 หรือ MOV C,91H

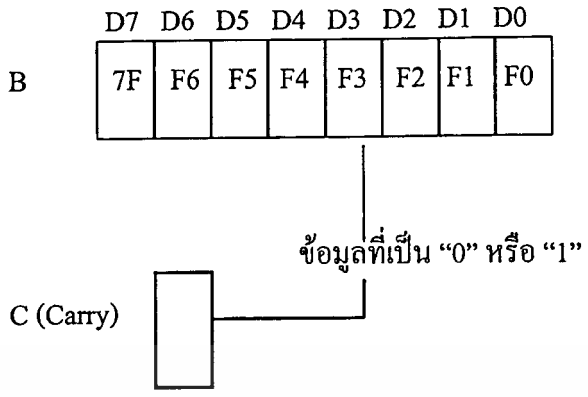
จากตัวอย่างเป็นการโอนย้ายข้อมูลจาก Port 0 บิตที่ 1 มาเก็บไว้ใน Carry ดังที่จะแสดงในรูป



MOV bit,C เป็นการโอนย้ายข้อมูล "0" หรือ "1" จาก Carry ไปไว้ที่ ตำแหน่งของ บิต (Bit Addressing) ดังตัวอย่าง

MOV B.3,C หรือ MOV 0F3H,C

จากตัวอย่างเป็นการโอนย้ายข้อมูลที่เป็น "0" หรือ "1" จาก Carry ไปเก็บไว้ที่รีจิสเตอร์ B บิตที่ 3 ดังที่ จะแสดงในรูป



หน่วยความจำภายนอกที่ใช้เป็นโปรแกรม (Program Memory)

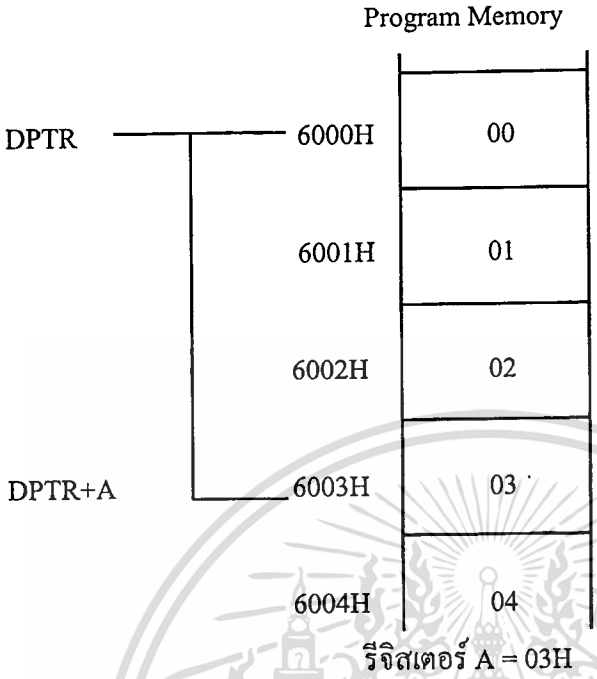
เราจะใช้คำสั่ง MOVC ในการอ้างหน่วยความจำชนิดนี้ส่วนมากคำสั่งชนิดนี้จะใช้ในการเปิดตารางที่มีอยู่แล้วโดยคำสั่งชนิดนี้จะอ้างข้อมูลเป็น Index จะมีตำแหน่งของฐานข้อมูล ซึ่งในคำสั่งนี้จะใช้ รีจิสเตอร์ PC(Program Counter) กับ DPTR(Data pointer) และ Offset จะใช้รีจิสเตอร์ A โดยก่อนที่จะนำข้อมูลมาได้ต้องทำการบวกค่า ของฐานข้อมูล กับ Offset และ ค่าที่ได้จะนำไปเป็นตำแหน่งชี้ข้อมูลเพื่อมาเก็บไว้ในรีจิสเตอร์ A ในคำสั่งนี้มีไว้ ใช้อยู่สองแบบดังนี้

MOVC A,@A+DPTR เป็นการโอนย้ายข้อมูลโดยใช้ รีจิสเตอร์ A เป็น Offset และ ใช้ DPTR เป็นตัวชี้ตำแหน่งของจุดเริ่มต้นของฐานข้อมูล แล้วมาเก็บไว้ที่ รีจิสเตอร์ A ดังที่จะแสดงใน

<u>ตัวอย่าง</u>

ตัวอย่างใช้คำสั่ง MOVC A,@A+DPTR

โดยที่กำ หนดให้ รีจิสเตอร์ A = 3 , รีจิสเตอร์ DPTR = 6000H



รีจิสเตอร์ A = 03H

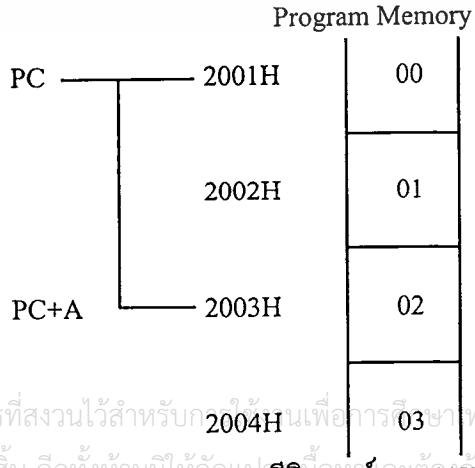
MOVC A,@A+PC เป็นคำสั่งโอนย้ายข้อมูลที่ใช้ รีจิสเตอร์ A เป็น Offset และ ใช้ PC เป็นจุดเริ่มต้นของตำแหน่งของข้อมูลโดยในคำสั่งนี้ค่าของ PC จะเป็นค่าถัดไปจากคำสั่งนี้ และ เอาค่าของ PC ไปบวกกับ รีจิสเตอร์ A ค่าที่ได้จากการบวกจะไปชี้ตำแหน่งที่ข้อมูลดังตัวอย่าง

ตัวอย่าง การใช้คำสั่ง MOVC A,@A+PC

โดยกำหนดให้ คำสั่ง MOVC A,@A+PC อยู่ที่ตำแหน่ง 2000H

รีจิสเตอร์ A = 2

2000H MOVC A,@A+PC



รีจิสเตอร์ A = 02H

หน่วยความจำภายนอกที่ใช้เป็นข้อมูล (Data Memory)

เราจะใช้คำสั่ง MOVX ในการอ้างหน่วยความจำชนิดนี้ซึ่งในคำสั่งนี้จะใช้รีจิสเตอร์ DPTR หรือ Ri (R0 หรือ R1) เป็นตัวชี้ข้อมูลโดยถ้าใช้ รีจิสเตอร์ DPTR จะเอาค่าใน DPTR ไปชี้ที่หน่วยความจำโดยเป็นตัวชี้ขนาด 16 Bit สามารถเข้าถึงข้อมูลในหน่วยความจำชนิดนี้ได้ถึง 64 Kbyte ส่วน Ri (R0 หรือ R1) จะข้อมูลใน (R0 หรือ R1) เป็นตัวชี้ข้อมูลทางด้านต่ำ คือ A0 ถึง A7 เป็นการอ้างแบบ 8 Bit ซึ่งก่อนที่เราจะใช้คำสั่งนี้ได้ก็ต้องทำการส่งตำแหน่งทางด้านสูงออกไปที่ Port 2 ก่อน โดยจะเป็นทางตำแหน่ง A8 ถึง A15 จะทำให้การอ้างครบ 16 Bit รูปแบบข้อความคำสั่งชนิดนี้มีดังนี้

MOVX A,Ri

เป็นคำสั่งโอนย้ายข้อมูลที่นำเอาค่าของข้อมูลใน Ri (R0 หรือ R1) เป็นตัวชี้ข้อมูลทางด้านตำแหน่งต่ำ (A0 ถึง A7) แล้วเอาข้อมูลจากตำแหน่งที่ชี้นั้นไปเก็บไว้ใน รีจิสเตอร์ A

MOVX A,@DPTR

เป็นคำสั่งโอนย้ายข้อมูล ที่นำเอาข้อมูลจากรีจิสเตอร์ DPTR ไปชี้ตำแหน่งของข้อมูลของ Data Memory แล้วเอาข้อมูลไปเก็บไว้ที่ รีจิสเตอร์ A

MOVX @Ri,A

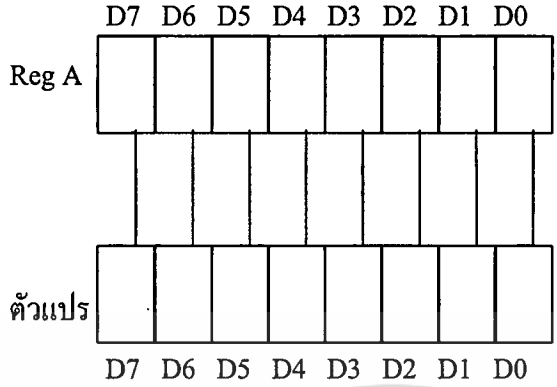
เป็นคำสั่งโอนย้ายข้อมูลที่นำเอาค่าของข้อมูลใน รีจิสเตอร์ A ไปเก็บไว้ที่ ตำแหน่งที่ข้อมูลของ Ri (R0 หรือ R1) เป็นตัวชี้ตำแหน่งของข้อมูลทางด้านต่ำ

MOVX @DPTR,A

เป็นคำสั่งโอนย้ายข้อมูลที่นำเอาข้อมูลใน รีจิสเตอร์ A ไปเก็บไว้ที่ข้อมูลที่ รีจิสเตอร์ DPTR ชี้อยู่

XCH

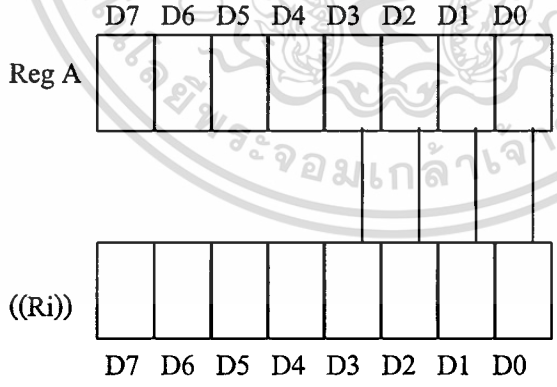
เป็นคำสั่งแลกเปลี่ยนข้อมูลแอคคิวมูเลเตอร์กับข้อมูลตัวแปร ซึ่ง คำสั่ง XCH จะโหลดค่าของข้อมูลที่ถูกกำหนดด้วยตัวแปร ไปเก็บไว้ที่ แอคคิวมูเลเตอร์ และในขณะเดียวกันก็จะโหลดค่าของ แอคคิวมูเลเตอร์ ไปเก็บในตำแหน่งของตัวแปรที่กำหนด โดยทั้งใน Source และ Destination ของโอเปอร์แรนด์สามารถให้ทำงานในโหมดรีจิสเตอร์ โดยตรง โดยอ้อม ด้วยรีจิสเตอร์ ลักษณะการทำงานของคำสั่ง XCH มีดังนี้



ลักษณะการทำงานของคำสั่ง XCH

XCHD (Exchange Digit)

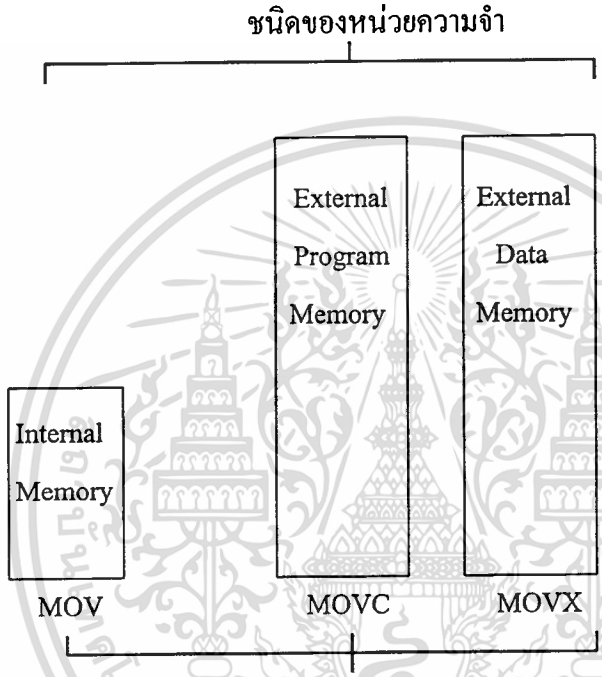
เป็นคำสั่งในการสลับค่าขนาดนิบเบิ้ล (nibble) กันระหว่างนิบเบิ้ลนัยต่ำของค่าข้อมูลของแอคคิวมูเลเตอร์ ซึ่งโดยทั่วไปจะเป็นการแทนค่าตัวเลข BCD (หรือตัวเลขฐาน 16) กับค่าข้อมูลนิบเบิ้ลนัยต่ำที่เก็บไว้ใน หน่วยความจำภายในโดยการกำหนดตำแหน่งที่อยู่ด้วยโหมดการกำหนดตำแหน่งโดยรีจิสเตอร์ ส่วนนิบเบิ้ลนัยสูง (บิตที่ 4-7) ของรีจิสเตอร์จะไม่มีผลต่อการเปลี่ยนแปลงนี้และไม่มีผลต่อค่าแฟล็กใดๆ ลักษณะการทำงานของคำสั่ง XCHD มีดังนี้



ลักษณะการทำงานของคำสั่ง XCHD A,Ri

สรุป

คำสั่งในการโอนย้ายข้อมูลเป็นคำสั่งที่สำคัญอย่างหนึ่งที่ควรจะเข้าใจก่อนที่จะศึกษาคำสั่งอื่น ซึ่งเป็นคำสั่งในการกำหนดค่าต่างๆ ก่อนที่จะไปกระทำข้อมูล การโอนย้ายข้อมูล และการแลกเปลี่ยนข้อมูลนี้ โอนย้ายข้อมูลที่จะใช้ เฉพาะชนิดของหน่วยความจำดังนี้



การทดลอง

1. ป้อนโปแกรมที่ 1 ซึ่งเป็นโปรแกรมทดลองการใช้งานคำสั่ง MOV

```
ORG   0000H
MOV   30H,#60H      ; (30H) = 60H
MOV   31H,#70H      ; (31H) = 70H
MOV   A,#30H        ; A = 30H
MOV   R0,A          ; R0 = A
MOV   A,@R0         ; A = ((R0))
MOV   R6,A          ; R6 = A
INC   R0            ; (R0) = (R0) + 1
MOV   A,@R0         ; A  = ((R0))
```

```
MOV   C,ACC.6        ; C = ACC.6
MOV   ACC.1,C        ; ACC.1 = C
SJMP  $
```

โปรแกรมที่ 1

2. สั่งให้โปรแกรมทำงานทีละ STEP โดยให้สังเกตค่าของรีจิสเตอร์ตามตารางแล้วทำการบันทึกค่าที่ได้ลงตาราง

รีจิสเตอร์	A	R0	R6	C
	00	00	00	00
MOV 30H,#60H				
MOV 31H,#70H				
MOV A,#30H				
MOV R0,A				
MOV A,@R0				
MOV R6,A				
INC R0				
MOV A,@R0				
MOV C,ACC.6				
MOV ACC.1,C				

3. ป้อนโปรแกรมที่ 2 ซึ่งเป็นโปรแกรมทดลองการใช้งานคำสั่ง MOVX

```
ORG   0000H
MOV   R0,#01H        ; R0 =01H
MOV   DPTR,#8900H    ; DPTR = 8900H
MOV   A,#40H         ; A = 40H
INC   A              ; A = A+1
MOVX  @DPTR          ; (DPTR) = A
INC   DPL            ; DPL = DPL+1
INC   A              ; A = A+1
```

```
MOVX @DPTR,A      ; ((DPTR)) = A
MOV  A,#50H       ; A = 50H
MOV  P2,#89H      ; P2(PORT 2) = 89H
MOVX A,@R0        ; A = ((R0))LSB
SJMP $
```

โปรแกรมที่ 2

4. เมื่อป้อนโปรแกรมเสร็จแล้วทำการสั่งให้โปแกรมทำงานแล้วให้ สังเกตการทำงานของคำสั่งต่างๆ และเมื่อทำงานคำสั่งจนเสร็จให้ตรวจดูว่าในรีจิสเตอร์ A มีค่าเท่ากับ
5. ป้อนโปรแกรมที่ 3 เข้าเครื่องซึ่งเป็นโปรแกรมทดลองการใช้งานคำสั่ง MOVC

```
LOOP:
      MOV  A,#1
      MOV  DPTR,#8400H
      MOVC A,@A+DPTR
      SJMP LOOP
```

โปรแกรมที่ 3

6. กำหนดให้ค่าในหน่วยความจำมีค่าดังนี้

8400H	44H	0AH	A4H	61H	81H	32H	44H	67H	54H	00H	8409H
840AH	11H	61H	65H	83H	91H	98H	87H	69H	11H	54H	8503H

7. ลองเปลี่ยนค่าในรีจิสเตอร์ A สังเกตว่า เมื่อทำงานตามคำสั่งเสร็จแล้วค่าของรีจิสเตอร์ A มีค่าเท่าไร เมื่อเปลี่ยนค่า A ก่อนที่จะสั่งให้โปรแกรมทำงาน

ถ้ารีจิสเตอร์ A = 1 เมื่อทำงานโปรแกรมเสร็จแล้วรีจิสตอร์ A เท่ากับเท่าไร

8. ป้อนโปรแกรมที่ 4 ซึ่งเป็นโปรแกรมทดลองการใช้งานคำสั่ง XCH

```
MOV  A,#60H       ; A = 60H
MOV  R0,#45H      ; R0 = 45H
XCH  A,R0         ; A <-> R0
```

โปรแกรมที่ 4

9. สั่งให้โปรแกรมทำงานตามโปรแกรมที่ 4 และให้สังเกตค่าที่รีจิสเตอร์ A,R0 ว่าจะเปลี่ยนค่าไปอย่างไร แล้วลองเปลี่ยนค่าของ รีจิสเตอร์ A,R0 ตามตารางแล้วบันทึกผลที่ได้ลงตาราง

ก่อนโปรแกรมทำงาน		หลังโปรแกรมทำงาน	
A	R0	A	R0
60H	45H		
57H	80H		
90H	0AFH		

10. ป้อนโปรแกรมที่ 5 เข้าเครื่อง ซึ่งเป็นโปรแกรมทดลองการใช้งานคำสั่ง XCHD

```
MOV   34H,#45H      ; (34) = 45H
MOV   R0,#34H       ; R0 = 34H
MOV   A,#60H        ; A = 60H
XCHD  A,@R0
```

โปรแกรมที่ 5

11. สั่งให้เครื่องทำงานตามโปรแกรมที่ 5 และ ให้สังเกตค่าที่รีจิสเตอร์ A กับตำแหน่งในหน่วยความจำ 34H ว่าจะเปลี่ยนค่าไปอย่างไร แล้วลองเปลี่ยนค่าตามตารางแล้วบันทึกผลที่ได้ลงตาราง

ก่อนโปรแกรมทำงาน		หลังโปรแกรมทำงาน	
A	34H	A	34H
60H	45H		
57H	80H		
90H	0AFH		

คำถาม

1. จงเขียนโปรแกรมในการโอนย้ายข้อมูลระหว่างหน่วยความจำของข้อมูลโดยใช้คำสั่ง MOVX A,@R1 โดยชี้ตำแหน่งของข้อมูลตำแหน่งที่ 957FH

การทดลองที่ 4
การใช้คำสั่งทางคณิตศาสตร์ของ MCS - 51

วัตถุประสงค์

1. เพื่อให้เราเข้าใจการใช้คำสั่งทางคณิตศาสตร์ของ MCS-51
2. เพื่อให้เราสามารถนำคณิตศาสตร์มาเขียนเป็นโปรแกรมได้

ทฤษฎี

ใน MCS-51 ได้มีคำสั่งเกี่ยวกับการทำงานทางคณิตศาสตร์ทางพื้นฐานสี่งานด้วยกัน และจะใช้ขนาดของข้อมูลในการคำนวน 8 Bit ที่ไม่คิดเครื่องหมายเป็นตัวคำนวนโดยตรง อย่างไรก็ตาม การใช้แฟล็ก OV (Overflow) ยังคงใช้งานในการบวก และ ลบ เพื่อทำให้ข้อมูลที่เป็นตัวคำนวนที่เป็นเลขลงตัวทางบวก และ ลบ ได้อยู่ทางคณิตศาสตร์ ซึ่งคำสั่งทางคณิตศาสตร์เป็นคำสั่งที่ใช้ในการคำนวนได้แบ่งตามพื้นฐานการคำนวนออกเป็น 4 แบบด้วยกันคือ

1. พื้นฐานของการบวก

เป็นการนำเอาค่าของตัวตั้งไปบวกเข้ากับตัวบวกโดยในการบวกในของไมโครโพรเซสเซอร์แล้วจะเป็นการบวกตัวเลขฐานสองเท่านั้นซึ่งต่างจากการบวกแบบตัวเลขฐานสิบ คือในการบวกเลขฐานสอง จะบวกค่าได้เฉพาะ “0” กับ “1” เท่านั้น ส่วนการบวกเลขฐานสิบนี้จะมีค่าที่จะทำการบวกได้ตั้งแต่ 0 ถึง 9 ดังตัวอย่างที่เปรียบเทียบการบวกเลขทั้งสองดังนี้

การบวกเลขฐานสิบ

81

+31

112

การบวกเลขฐาน สอง

1010

+1011

1 0101

เราจะสังเกตความแตกต่างกันระหว่างการบวกเลขฐานสิบ คือ 1+1 = 2 แต่ในการบวกเลขฐานสองจะเป็น 1+1 = 10 ซึ่งกฎของการบวกเลขฐานสองมีดังนี้

กฎของการบวกเลขฐานสอง

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 10

คำสั่งที่ใช้ในการบวกของ MCS-51 จะใช้รีจิสเตอร์ A เป็นตัวตั้งเสมอ และเมื่อทำการบวกผลลัพธ์จะเก็บไว้ที่ รีจิสเตอร์ A เสมอ และเมื่อมีตัวทดก็จะไปเก็บไว้ที่ CY (carry) ซึ่งคำสั่งที่เกี่ยวกับการบวกมีดังนี้

ADD เป็นคำสั่งบวกค่าในแอกคิวมูเลเตอร์กับค่าในแหล่งกำเนิดโอเปอร์แรนด์ โดยไม่คิดตัวทดแล้วส่งผลลัพธ์มาเก็บไว้ที่แอคคิวมูเลเตอร์

ADDC (Add with carry) เป็นคำสั่งบวกค่าแบบคิดตัวทด โดยทำการบวกค่าในแอคคิวมูเลเตอร์ กับค่าในแหล่งกำเนิดโอเปอร์แรนด์ แล้วบวกค่าที่ในบิตรตัวทด CY (carry) แล้วส่งผลลัพธ์มาเก็บไว้ที่แอคคิวมูเลเตอร์ การบวกชนิดนี้จะใช้ในการบวกค่าที่จำนวนมากกว่า 1Byte ขึ้นไป

DA A (Decimal Adjust Accumulator) เป็นคำสั่งที่ทำให้สามารถทำการ บวก ลบ เลขที่เป็น BCD (Binary Code Decimal) ได้ คำสั่ง DA A นี้จะวางไว้หลังคำสั่งทางคณิตศาสตร์ ADD, ADDC, INC, SUBB, DEC เพื่อทำการเปลี่ยนผลลัพธ์ที่อยู่ในรูปเลขฐานสองให้กลับมาเป็นผลลัพธ์ที่เป็น เลขฐานสิบ เมื่อการแปลงเลขฐานได้ค่ามาเกิน หนึ่งหลักก็จะไปเก็บไว้ที่ CY (carry) ดังตัวอย่างต่อไปนี้

ตัวอย่างการใช้คำสั่ง DA A

```
MOV   A,#30        ; A = 30
ADD   A,#80        ; A = A+80
DA    A            ; แปลงเลขให้อยู่ในตัวเลขฐานสิบ
```

จากโปรแกรมตัวอย่างจะเป็นการบวกค่า 80 กับรีจิสเตอร์ A ซึ่งเรากำหนดให้เท่ากับ 30 แต่ในไมโครโพรเซสเซอร์จะถือว่าเป็นเลขฐานสองจะทำการบวกค่าดังนี้

```
  0011 0000  (30)
+ 1000 0000  (80)
  1011 0000  (0B0)
```

ซึ่งค่าที่ได้คือ 1011 0000 ในเลขฐานสอง 0B0H ในเลขฐาน 10 แทนที่จะเป็น 110 ในเลขฐานสิบ แต่เมื่อทำคำสั่ง DA A แล้วค่าที่ได้ในรีจิสเตอร์ A จะมีค่าเท่ากับ 10 ซึ่งอีกหลักจะไปเก็บไว้ที่ CY (carry) คือ 1

INC (Increment) เป็นคำสั่งที่เพิ่มค่าในรีจิสเตอร์ หรือ หน่วยความจำ หนึ่ง ซึ่งคำสั่งนี้จะทำการบวกค่า หนึ่งเข้ากับค่าที่มีอยู่แล้วในรีจิสเตอร์ หรือหน่วยความจำ แล้วเก็บค่าที่ได้จากการเพิ่มนี้เข้าใส่ ในรีจิสเตอร์ หรือ หน่วยความจำที่อ้างถึงในคำสั่งนี้ดังตัวอย่าง

ตัวอย่างการใช้คำสั่ง INC

โดยกำหนดให้ รีจิสเตอร์ A ซึ่งก่อนที่จะใช้คำสั่ง INC A มีค่าเท่ากับ 0FEH

คำสั่ง	ค่าในรีจิสเตอร์ A
	0FEH
INC A	0FFH
INC A	00H
INC A	01H

ซึ่งจากตัวอย่างเมื่อใช้คำสั่ง INC เพิ่มค่าไปเรื่อยๆ จนถึงค่า 0FFH เมื่อใช้คำสั่ง INC อีกครั้งค่าที่ได้จากการเพิ่มจะเป็น 0

2. พื้นฐานของการลบ เป็นการลบค่าระหว่างตัวตั้ง และ ตัวลบในการลบเลขในฐานสิบ กับฐานสองจะคล้าย แต่จะแตกต่างกันที่ การลบเลข ฐานสิบจะมีค่าที่จะทำการลบได้ 0 ถึง 9 แต่ในฐานสอง จะมีเพียง 0 กับ 1 เท่านั้น และ เมื่อค่าตัวตั้งมีค่าน้อยกว่าตัวลบค่าที่ได้ในฐานสิบจะ เป็น ลบ ส่วน ในฐานสอง จะเป็นการยืม ซึ่งกฎของการลบเลขฐานสองมีดังนี้

0 - 0 = 0

0 - 1 = 1 มีการยืม 1

1 - 0 = 1

1 - 1 = 0

คำสั่งที่ใช้ในการลบจะใช้รีจิสเตอร์ A เป็นตัวตั้งซึ่งคำสั่งในการลบมีดังนี้

SUBB (Subtrac with borrow) เป็นการนำเอาค่าตัวเลขในแหล่งกำเนิดโอเปอร์แรนด์ ลบออกจากแอคคิวมูเลเตอร์ และหลังจากนั้น ก็นำค่าที่อยู่ในบิตทด CY (carry) ไปลบอีกครั้ง แล้วเอาผลลัพธ์ที่ได้จากการลบเข้าไปเก็บไว้ที่ แอคคิวมูเลเตอร์

DEC (Decrement) เป็นคำสั่งที่ลดค่าในรีจิสเตอร์ หรือ หน่วยความจำลงหนึ่ง ซึ่งคำสั่งนี้จะทำการบวกค่า หนึ่งเข้ากับค่าที่มีอยู่แล้วในรีจิสเตอร์ในรีจิสเตอร์หรือหน่วยความจำ แล้วเก็บค่าที่ได้จากการลดค่านี้เข้าไปเก็บไว้ที่รีจิสเตอร์ หรือ หน่วยความจำที่อ้างถึงในคำสั่งนี้ตัวอย่างเช่น

ตัวอย่างการใช้คำสั่ง DEC

โดยกำหนดให้ รีจิสเตอร์ A ซึ่งก่อนที่จะใช้คำสั่ง DEC A มีค่าเท่ากับ 02H

คำสั่ง	ค่าในรีจิสเตอร์ A
	02H
DEC A	01H
DEC A	00H
DEC A	0FFH

3. พื้นฐานของการคูณ

เป็นการนำเอาค่าตัวตั้ง และ ตัวคูณ มาทำการคูณกัน ซึ่งถ้าจะมองในการบวกแล้วก็คือ การเอาค่าตัวตั้งมาทำการบวกกันจำนวนครั้ง เท่ากับตัวคูณเช่นในการคูณเลขฐานสิบ

4 X 2 = 8 หรือ 4 + 4 = 8

8 X 4 = 32 หรือ 8 + 8 + 8 + 8 = 32

ในไมโครโพรเซสเซอร์แล้ว จะเป็นการคูณเลข BCD แบบไม่คิดเครื่องหมายนั้นทำได้ 2 วิธี คือ อาศัยการบวกกันของเลขหลายๆ ครั้ง และ การเลื่อนบิตของข้อมูลไปทางซ้ายมือ ซึ่งการเลื่อนบิตนี้ จะมีค่าเท่ากับการคูณด้วย 2^n N = จำนวนบิตที่ต้องการเลื่อน เช่น ต้องการคูณเลขด้วย 4 ก็ทำได้โดยการเลื่อนบิตของข้อมูลไปทางซ้าย 2 บิต ตัวอย่างเช่น ถ้าต้องการคูณเลขสองจำนวนที่ตัวตั้งเท่ากับ 5 ตัวคูณเท่ากับ 3 การคูณด้วยมือทำได้ดังนี้

(5) 1 0 1 ตัวตั้ง

(3) 0 1 0+ ตัวคูณ

(4) 1 0 1

1 0 1

0 0 0

0 1 1 1 1 ผลลัพธ์

ใน MCS - 51 ได้มีคำสั่งนี้ไว้ใช้อยู่แล้ว คำสั่งที่ใช้ในการคูณก็คือการใช้คำสั่ง MUL

MUL เป็นคำสั่งการคูณกันแบบไม่คิดเครื่องหมายของตัวเลขที่อยู่ในแอกคิวมูเลเตอร์ A กับตัวเลขที่อยู่ในรีจิสเตอร์ B แล้วนำผลลัพธ์ที่ได้จากการคูณซึ่งมีขนาดสูงสุด 2 Byte นำไปเก็บที่ AB โดยที่ รีจิสเตอร์ A จะรับหลักแรกส่วนรีจิสเตอร์ B จะรับหลักที่สองซึ่งเป็นหลักสูง ค่าของ OV

(Overflow) จะเป็น "0" ถ้าค่าในรีจิสเตอร์ B มีค่าเท่ากับ 0 จะเป็น "1" ถ้าค่าในรีจิสเตอร์ B ไม่เป็น 0 ส่วน CY (carry) จะเป็น "0" และไม่มี ผลต่อ AC (Auxiliary)

4. พื้นฐานของการหาร

เป็นการนำเอาค่าตัวตั้ง และ ตัวหาร มาทำการหารกัน ซึ่งถ้าจะมองในการคำนวณพื้นฐานก็คือการนำเอาตัวหารมาลบกับตัวตั้ง จะทำการลบไปเรื่อยๆ จนกว่าค่าตัวตั้งจะเป็นเครื่องหมายเป็นจำนวนครั้งในการลบ จะเป็นผลลัพธ์ในการหาร และ ผลของการลบก่อนที่ตัวตั้งจะเปลี่ยนเครื่องหมาย ก็คือเศษที่เหลือนั้นเอง และมีประสิทธิภาพมากกว่า คือการหารยาว ใน MCS - 51 มีคำสั่งที่ใช้ในการหาร คือใช้คำสั่ง DIV

DIV จะเป็นคำสั่งการหารกันด้วยตัวเลขที่ไม่คิดเครื่องหมายที่อยู่ในแอคคิวมูเลเตอร์ A (โดยถือว่าเป็นตัวตั้ง) ที่หารด้วยตัวเลขที่อยู่ในรีจิสเตอร์ B (ถือเป็นตัวหาร) และนำผลลัพธ์ที่ได้จากการหาร มาเก็บไว้ที่แอคคิวมูเลเตอร์ และเศษส่วนไปเก็บไว้ที่รีจิสเตอร์ B การหารด้วยค่า 0 จะไม่มีผลต่อข้อมูลในรีจิสเตอร์ A และ B และ OV (Overflow) จะเป็น "1" ส่วนการหารด้วยค่าอื่น OV (Overflow) จะเป็น "0" และไม่มีผลต่อ AC ถ้าไม่มีการหารกันด้วยตัวเลขที่ไม่เป็นแบบที่กล่าวตามย่อหน้าข้างบนนี้แล้ว ค่าบิตแฟล็กต่างๆ ใน PSW จะมีผลดังต่อไปนี้เนื่องจากการหารด้วยตัวเลขต่างๆ ดังนี้

CY (carry) จะเป็น "1" ถ้ามีการทำงานเนื่องจากผลของบิตสูงมีการทดเข้าสู่การยืมออกจากตัวทด

AC จะเป็น "1" ถ้าผลจากการทำงานเกิดมีการทดจากนิบเบิ้ลเบอร์ต่ำ ระหว่างการบวกกันหรือมีการยืมจากนิบเบิ้ลสูงเข้าสู่นิบเบิ้ลต่ำ ระหว่างการลบกันนอกเหนือจากนี้ บิต AC จะเป็น "0"

OV (Overflow) จะเป็น "1" ได้ดีต่อเมื่อเกิดการที่ค่าตัวหารมีค่า เป็น 0 เมื่อเกิดลักษณะนี้ขึ้น

P จะเป็น "1" ถ้าค่าฐานตัวเลข Modulo 2 (หลักที่ 2 ของผลลัพธ์ที่อยู่ในรีจิสเตอร์ B) รวมกับ 8 บิต ในแอคคิวมูเลเตอร์เป็นจำนวนคี่ และ บิต P จะเป็น "0" ถ้ารวมกันเป็นคู่

สรุป

คำสั่งทางคณิตศาสตร์ของ MCS - 51 เป็นคำสั่งที่ใช้คำนวณค่าตัวตั้ง กับ ตัวกระทำ คำสั่งทางคณิตศาสตร์นี้มีคำสั่งในการบวก การบวกพร้อมตัวทด การลบ การคูณ และการหาร การทำงานตามคำสั่งในการคำนวณนี้จะใช้รีจิสเตอร์ A ในการคำนวณโดยใช้เป็นตัวตั้ง ตัวเก็บผลลัพธ์ในการคำนวณ และ การคำนวณมีผลต่อการเปลี่ยนแปลงค่าของแฟล็ก เช่น เกิดการยืม เกิดการทด และ การคำนวณค่ามาก ทำให้ CY, OV ซึ่งมีการเปลี่ยนแปลงตลอดถ้าเกิดมีการคำนวณ

การทดลอง

1. ป้อนโปรแกรมที่ 1 ซึ่งเป็นโปรแกรมในการทดลองการบวกเลข และการแปลงเลขฐาน

```
MOV   A,R1    ; A = R1
ADD   A,R0    ; A = A + R0
```

โปรแกรมที่ 1

2. จากโปรแกรมที่ 1 เมื่อสั่งให้โปรแกรมทำงานโดยการแทนค่าใน R0, R1 โดยแทนค่าตามตารางต่อไปนี้

R1	R0	A	CY	OV	P
50H	40H				
7FH	9FH				
00H	40H				
80H	60H				

3. ป้อนโปรแกรมที่ 2 ซึ่งเป็นโปรแกรมในการทดลองการลบเลข โดยใช้

```
MOV   A,R1        ; A = R1
SUBB  A,R0        ; A = A+R0
```

โปรแกรมที่ 2

4. จากโปรแกรมที่ 2 เมื่อสั่งให้โปรแกรมทำงานโดยการแทนค่าใน R0, R1 โดยแทนค่าตามตารางต่อไปนี้

ค่าก่อนที่จะทำงาน

R1	R0	CY	CY	OV	A
50H	40H	0			
7FH	9FH	0			
00H	40H	1			
80H	60H	2			

5. ป้อนโปรแกรมที่ 3 เข้าเครื่องซึ่งเป็นโปรแกรมในการทดลองคำสั่งในการคูณเลข โดยให้

```
MOV   A,#10H
MOV   B,#20H
MUL   AB
```

โปรแกรมที่ 3

6. จากโปรแกรมที่ 3 เมื่อสั่งให้โปรแกรมทำงานโดยการแทนค่าใน A, B โดยแทนค่าตามตารางต่อไปนี้

ผลลัพธ์จากการทำคำสั่ง MUL

R1	R0	A	C	CY
50H	40H			
7FH	9FH			
00H	40H			
80H	60H			

7. ป้อนโปรแกรมที่ 4 เข้าเครื่องซึ่งเป็นโปรแกรมในการทดลองคำสั่งในการหารเลข โดยให้

```
MOV   A,#10H
MOV   B,#20H
DIV   AB
```

โปรแกรมที่ 4

8. จากโปรแกรมที่ 4 เมื่อสั่งให้โปรแกรมทำงานโดยการแทนค่าใน A, B โดยแทนค่าตามตารางต่อไปนี้

ผลลัพธ์จากการทำคำสั่ง DIV

R1	R0	A	B	CY	OV
50H	40H				
7FH	9FH				
00H	40H				
80H	60H				

คำถาม

1. จงเขียนโปรแกรมในการบวกเลข 16 บิต โดยกำหนดให้ R0, R1 เป็น ตัวตั้งให้ R2, R3 เป็นตัวบวกโดยที่ผลลัพธ์นำไปเก็บที่หน่วยความจำ 30H, 31H และ CY (Carry)

SETB เป็นคำสั่งที่ทำให้ค่าในตำแหน่งแบบบิต (Bit Addressing) เป็น “1” การเปลี่ยนแปลงแสดงได้ตามตารางดังนี้

ก่อนทำคำสั่ง SETB	หลังคำสั่ง SETB
1	1
0	1

ตัวอย่างการใช้คำสั่ง SETB

ก่อนทำคำสั่ง SETB 91H หรือ SETB P1.1

P1	1	1	1	0	1	0	0	1

Bit Address 97H 96H 95H 94H 93H 92H 91H 90H

หลังทำคำสั่ง SETB 91H หรือ SETB P1.1

P1	1	1	1	0	1	0	1	1

Bit Address 97H 96H 95H 94H 93H 92H 91H 90H

CPL เป็นคำสั่งที่เก็บ หรือ Complement ข้อมูลในแอคคิวมูเลเตอร์ หรือ ค่าในการกำหนดตำแหน่งแบบบิต โดยไม่มีผลใดต่อค่าแฟล็กใน PSW หรือการให้ตำแหน่งบิตซึ่งคำสั่งนี้จะเปลี่ยนตารางดังนี้

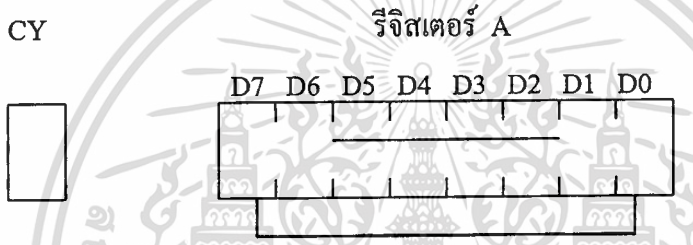
ก่อนทำคำสั่ง CPL	หลังคำสั่ง CPL
1	0
0	1

ตัวอย่างการใช้คำสั่ง CPL

รีจิสเตอร์ A

	ค่าตัวเลขฐาน 2	ค่าตัวเลขฐาน 10
ก่อนทำคำสั่ง CPL A	1001 1111	9F
หลังทำคำสั่ง CPL A	0110 0000	60

RL เป็นคำสั่งในการหมุนข้อมูลไปทางซ้ายใน แอคคิวมูเลเตอร์ โดยการหมุนข้อมูลแบบ 8 บิต ดังที่จะแสดงในรูปดังนี้

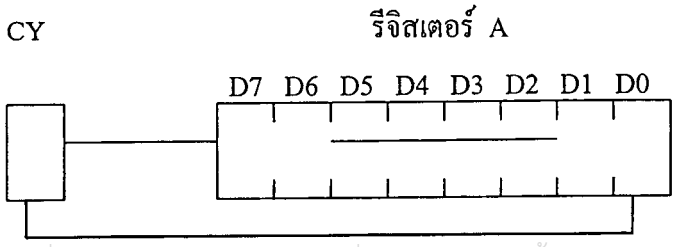


ตัวอย่างการใช้คำสั่ง RL

กำหนดให้รีจิสเตอร์ A = 1011 0001

	รีจิสเตอร์ A	CY (carry)
ก่อนทำคำสั่ง RL	1011 0001	0
หลังทำคำสั่ง RL	0110 0011	0

RLC เป็นคำสั่งในการหมุนข้อมูลไปทางซ้ายใน แอคคิวมูเลเตอร์ พร้อมทั้ง CY (carry) ซึ่งจะเป็นการหมุนข้อมูลแบบ 9 บิต

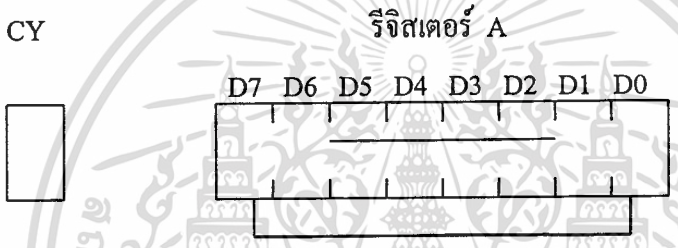


ตัวอย่างการใช้คำสั่ง RLC

กำหนดให้รีจิสเตอร์ A = 1000 0001

	รีจิสเตอร์ A	CY (carry)
ก่อนทำคำสั่ง RLC	1000 0001	0
หลังทำคำสั่ง RLC	0000 0010	1

RR เป็นคำสั่งในการหมุนข้อมูลไปทางขวาใน แอคคิวมูเลเตอร์ โดยการหมุนข้อมูลแบบ 8 บิต ดังที่จะแสดงในรูปดังนี้

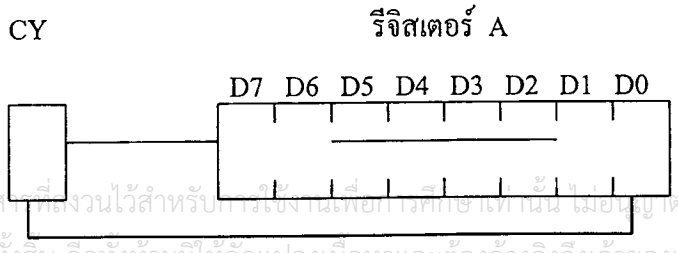


ตัวอย่างการใช้คำสั่ง RR

กำหนดให้รีจิสเตอร์ A = 1000 0001

	รีจิสเตอร์ A	CY (carry)
ก่อนทำคำสั่ง RR	1000 0001	0
หลังทำคำสั่ง RR	0100 0000	0

RRC เป็นคำสั่งในการหมุนข้อมูลไปทางขวาใน แอคคิวมูเลเตอร์ พร้อมทั้ง CY (carry) ซึ่งจะเป็นการหมุนข้อมูลแบบ 9 บิต

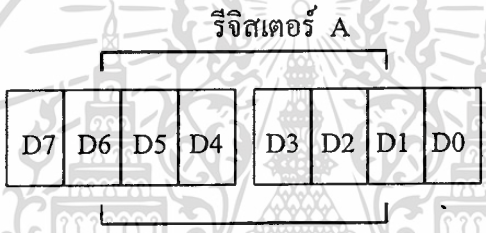


ตัวอย่างการใช้คำสั่ง RRC

กำหนดให้รีจิสเตอร์ A = 1001 0001

	รีจิสเตอร์ A	CY (carry)
ก่อนทำคำสั่ง RRC	1001 0001	0
หลังทำคำสั่ง RRC	0100 1000	1

SWAP เป็นคำสั่งสลับข้อมูลภายใน แอคคิวมูเลเตอร์ โดยการสลับข้อมูลระหว่างข้อมูล D7 - D4 กับ D3 - D0 โดยการสลับค่าของข้อมูล D7 กับ D3, D6 กับ D2, D5 กับ D1, D4 กับ D0 ดังที่จะแสดงในรูปดังนี้



ตัวอย่างการใช้คำสั่ง SWAP A

กำหนดให้รีจิสเตอร์ A = 1111 0000

	รีจิสเตอร์ A
ก่อนทำคำสั่ง SWAP A	1111 0000
หลังทำคำสั่ง SWAP A	0000 1111

ANL เป็นคำสั่งที่กระทำข้อมูลทางตรรกศาสตร์ ระหว่างแหล่งกำเนิดโอเปอร์แรนด์ ซึ่งจะสั่งให้ทำงานทางตรรกศาสตร์ หรือทางลอจิกของข้อมูลขนาดเป็น ไบต์ หรือ บิตก็ได้ และจะนำผลลัพธ์กลับไปเก็บไว้ที่ตำแหน่งตัวโอเปอร์แรนด์ที่สั่งในโอเปอร์แรนด์เป็นตัวแรก กฎของการทำคำสั่ง ANL มีดังนี้

กฎของการใช้คำสั่ง ANL

0 ANL 0 = 0

0 ANL 1 = 0

1 ANL 0 = 0

1 ANL 1 = 1

ตัวอย่างการใช้คำสั่ง ANL

กำหนดให้รีจิสเตอร์ A = 1011 0001

ANL A,#0011 1111

	รีจิสเตอร์ A
ก่อนทำคำสั่ง	1011 0001
หลังทำคำสั่ง	0011 0001

จากตัวอย่างการใช้คำสั่ง ANL จะแสดงการทำคำสั่งดังนี้

1011 0001

ANL

0011 1111

ผลลัพธ์ 0011 0001

ORL เป็นคำสั่งที่กระทำข้อมูลทางตรรกศาสตร์ ระหว่างแหล่งกำเนิดโอเปอร์แรนด์ ซึ่งจะสั่งให้ทำงานทางตรรกศาสตร์ หรือทางลอจิกของข้อมูลขนาดเป็น ไบต์ หรือ บิตก็ได้ และจะนำผลลัพธ์กลับไปเก็บไว้ที่ตำแหน่งตัวโอเปอร์แรนด์ที่สั่งในโอเปอร์แรนด์เป็นตัวแรก กฎของการทำคำสั่ง ORL มีดังนี้

กฎของการใช้คำสั่ง ORL

0 ORL 0 = 0

0 ORL 1 = 1

1 ORL 0 = 1

1 ORL 1 = 1

ตัวอย่างการใช้คำสั่ง ORL

กำหนดให้รีจิสเตอร์ A = 1111 0000

ORL A,#1001 0111

	รีจิสเตอร์ A
ก่อนทำคำสั่ง	1011 0001
หลังทำคำสั่ง	1011 1111

จากตัวอย่างการใช้คำสั่ง ORL จะแสดงการทำคำสั่งดังนี้

1011 1001

ORL

1001 0111

ผลลัพธ์ 1011 1111

XRL เป็นคำสั่งที่กระทำข้อมูลทางตรรกศาสตร์ ระหว่างแหล่งกำเนิดโอเปอร์แรนด์ ซึ่งจะสั่งให้ทำงานทางตรรกศาสตร์ หรือทางลอจิกของข้อมูลขนาดเป็น ไบต์ หรือ บิตก็ได้ และจะนำผลลัพธ์กลับไปเก็บไว้ที่ตำแหน่งตัวโอเปอร์แรนด์ที่สั่งในโอเปอร์แรนด์เป็นตัวแรก กฎของการทำคำสั่ง XRL มีดังนี้

กฎของการใช้คำสั่ง XRL

0 XRL 0 = 0

0 XRL 1 = 1

1 XRL 0 = 1

1 XRL 1 = 0

ตัวอย่างการใช้คำสั่ง XRL

กำหนดให้รีจิสเตอร์ A = 1111 0001

XRL A,#1001 1101

	รีจิสเตอร์ A
ก่อนทำคำสั่ง	1111 0001
หลังทำคำสั่ง	0110 1100

จากตัวอย่างการใช้คำสั่ง XRL จะแสดงการทำคำสั่งดังนี้

	1111 0001
XRL	
	1001 1101
ผลลัพธ์	0110 1100

สรุป

คำสั่งทางตรรกศาสตร์ หรือ ทางลอจิก เป็นคำสั่งพื้นฐานของไมโครโพรเซสเซอร์ทั่วไป ซึ่งมีคำสั่งในการ ทำค่าให้เป็น 0, ทำค่าให้เป็น 1, หมุนข้อมูลไปทางซ้ายหรือขวา และคำสั่งในการ AND, OR, XRL ซึ่งการกระทำข้อมูลที่คล้ายในวงจรดิจิตอล จะแตกต่างกันก็ที่เป็นคำสั่งที่ทำตามลำดับของคำสั่งเท่านั้นเอง

การทดลอง

1. ป้อนโปรแกรมที่ 1 เข้าเครื่อง

```
        ORG   0000H
        MOV   A,#00000001B  ;ACC = 00000001B
        MOV   R0,#9         ; LOOP = 9
LOOP:   RR    A             ; Rotate Right Accumultor
        DJNZ  R0,LOOP       ; Rotate Right 9 Loop
        SJMP  $
```

โปรแกรมที่ 1

2. สั่งให้เครื่องทำงานตามโปรแกรมสังเกตค่าของ Accumulator

3. ป้อนให้ทำการเปลี่ยนคำสั่งในโปรแกรมที่ 1 จาก RR A เป็น RL A, RRC A, RLC A ตามลำดับ แล้วทำตามข้อ 2 บันทึกค่า

คำสั่ง	ACCUMULATOR	CY
RR A		
RL A		
RRC A		
RLC A		

4. ป้อนโปรแกรมที่ 2 เข้าเครื่อง

```
        ORG   0000H
START:  MOV   A,#0000111B
        SWAP  A
        ANL   A,#10000000B
        ORL   A,#00001100B
        XRL   A,#10001100B
        CPL   A
        SJMP  $
```

โปรแกรมที่ 2

5. สั่งให้เครื่องทำงานตามโปรแกรมที่ป้อนเข้าไปโดยให้ทำงานทีละคำสั่งแล้วทำการเรียกดูค่าของรีจิสเตอร์บันทึกค่า

คำสั่ง	ACCUMULATOR	CY
SWAP A		
ANL A		
ORL A		
XRL A		

คำถาม

1. ผลที่ได้จากการใช้ XRL A,#0F0H โดยที่รีจิสเตอร์ A ก่อนที่จะทำคำสั่งนี้มีค่าเท่ากับ 0FH เมื่อทำคำสั่งนี้แล้วรีจิสเตอร์ A มีค่าเท่ากับเท่าไร

การทดลองที่ 6
การใช้คำสั่งในการกระโดดของ MCS - 51

วัตถุประสงค์

1. เพื่อเข้าใจคำสั่งในการกระโดด และ การเก็บค่าแสตค
2. สามารถสร้างเงื่อนไขในการกระโดดได้
3. สามารถใช้คำสั่งในการโปรแกรมย่อยได้

ทฤษฎี

คำสั่งในการกระโดด เป็นคำสั่งที่จะทำการย้ายตำแหน่งการทำงาน หรือ ตำแหน่งของโปรแกรม โดยการเปลี่ยนค่าใน PC (Program Counter) ไปที่ตำแหน่งของโปรแกรมที่จะย้ายตำแหน่งของโปรแกรม ที่จะทำงานในคำสั่งต่อไป คำสั่งในการกระโดดนี้จะแบ่งออกตามการใช้งานดังนี้

1. การกระโดดแบบไม่มีเงื่อนไข
2. การกระโดดแบบมีเงื่อนไข
3. การเรียกโปรแกรมย่อย โดยสามารถกลับมายังโปรแกรมหลักได้

การกระโดดแบบไม่มีเงื่อนไข

เป็นคำสั่งการกระโดดที่ย้ายการทำงานของโปรแกรมโดยการเปลี่ยนค่าใน PC (Program Counter) ไปที่ตำแหน่งที่ต้องการที่จะทำงานในคำสั่งต่อไป ในการกระโดดนี้จะไม่มีการตรวจสอบเงื่อนไข มีคำสั่งในการกระโดดนี้มีคำสั่งดังต่อไปนี้

AJMP (Absolute Jump) จะทำการกระโดดการทำงานของโปรแกรมไปยังแอดเดรสที่ถูกกำหนดอยู่ในรูปแบบออปโค้ดไบต์แรกที่บิต 7 - 5 และ ไบต์ ที่สองของชุดคำสั่ง โดยบิตที่เหลือที่อยู่นัยสำคัญสูงอีก 5 บิต ของ PC หลังจากที่ PC นี้เพิ่มค่าขึ้นครั้งละหนึ่งสองครั้ง แล้วจะยังคงที่ไม่ถูกทำให้เปลี่ยนแปลงจากคำสั่ง ฉะนั้นเป้าหมายของโปรแกรมที่จะกระโดยไปจะต้องอยู่ในขอบเขต 2 Kbyte ต่อจากแอดเดรสของคำสั่งที่ต่อจาก AJMP

LJMP (Long Jump) เป็นการกระโดดไปยังค่าแอดเดรสที่ถูกกำหนดโดยไม่มีเงื่อนไขโดยการโหลดค่าไบต์สูงและไบต์ต่ำด้วยรหัสคำสั่งของไบต์ตัวที่สอง และสามตามลำดับ เข้า PC ดังนั้น

ตำแหน่งแอดเดรสที่จะกระโดดไปสามารถที่ จะอยู่ตำแหน่งใดๆ ก็ได้ ภายในหน่วยความจำของโปรแกรม 64 Kbyte โดยที่ไม่มีผลต่อแฟล็กใดๆ

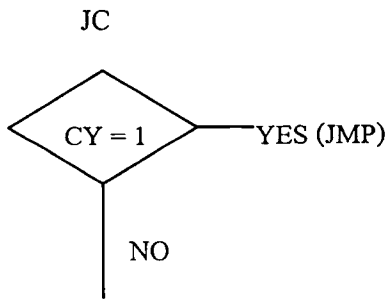
SJMP (Shot Jump) เป็นคำสั่งการกระโดดแบบไม่มีเงื่อนไขไปยังแอดเดรส ที่ถูกกำหนดโดยจะกระโดดไปตามค่าแอดเดรสที่ได้จากการบวกเอาข้อมูลแบบ sign relative displacement ของรหัสคำสั่งไบต์ที่สองกับตัวชี้โปรแกรม PC ที่มีค่าแอดเดรสที่ชี้ไบต์แรกของคำสั่งตัวต่อมา แล้วเก็บไว้ที่ PC เพราะฉะนั้นตำแหน่งของคำสั่งที่จะกระโดดไปอยู่ข้างหน้าไม่ห่างเกิน 127 ไบต์ หรือไปข้าหลังไม่เกิน 128 ไบต์

JMP @A+DPTR เป็นคำสั่งการกระโดดแบบไม่มีเงื่อนไขโดยกระโดดตามค่าผลลัพธ์ที่ได้จากการบวกโดยเอาค่าของข้อมูลในแอคคิวมูเลเตอร์ขนาด 8 Bit กับ DPTR ขนาด 16 Bit ผลจากการบวกกันของไบต์ต่ำถ้าเกิดตัวทดจะทดไปสู่ไบต์สูงผลที่ได้จะไปเก็บใน PC โดยที่ค่าของแอคคิวมูเลเตอร์ และ DPTR ไม่มีการเปลี่ยนแปลง และ ไม่มีผลต่อแฟล็ก

การกระโดดแบบมีเงื่อนไข

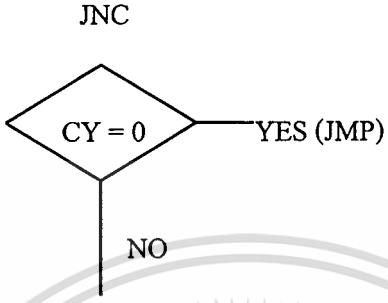
เป็นคำสั่งกระโดดที่จะทำการตรวจสอบเงื่อนไข ที่กำหนดไว้โดย ถ้าตรงตามเงื่อนไขให้ทำการกระโดด หรือ ผ่านคำสั่งกระโดดนั้นไป คำสั่งกระโดดแบบมีเงื่อนไขนี้ สามารถตรวจสอบเงื่อนไข แบบ บิต หรือ ไบต์ ตามแต่กำหนดคำสั่งกระโดดแบบมีเงื่อนไขมีดังนี้

JC ทำการกระโดดก็ต่อเมื่อ บิตทดเป็น “1” เป็นคำสั่งกระโดดถ้าบิตทดเป็น 1 จะกระโดดไปยังแอดเดรสที่กำหนดในคำสั่งด้วยค่าที่ได้จากการบวกกันของค่า signed relative displacement ไบต์ที่ 2 ของคำสั่งนี้กับค่าของ PC ที่ชี้ไบต์แรกของคำสั่งตัวต่อมา แล้วเก็บไว้ที่ PC หลังจากเพิ่มค่าแล้วจะเป็นการกระโดดตามค่าของ PC โดยไม่มีผลต่อแฟล็กใดๆ การทำงานของคำสั่ง JC จะแสดงตาม Flow Chart

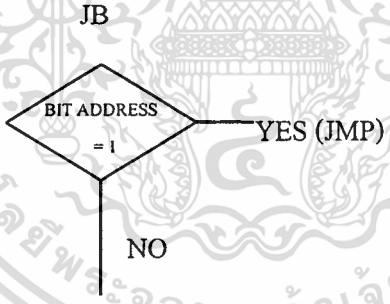


JNC ทำการกระโดดก็ต่อเมื่อ บิตทดเป็น “1” เป็นคำสั่งกระโดดถ้าบิตทดไม่เป็น 1 จะกระโดดไปยังแอดเดรสที่กำหนดในคำสั่งด้วยค่าที่ได้จากการบวกกันของค่า signed relative

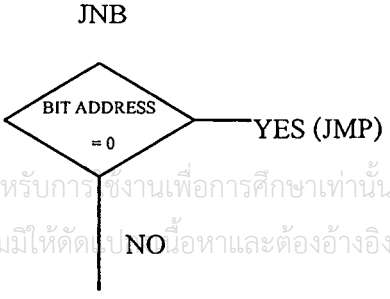
displacement ไบต์ที่ 2 ของคำสั่งนี้กับค่าของ PC ที่ชี้ไบต์แรกของคำสั่งตัวต่อมา แล้วเก็บไว้ที่ PC หลังจากเพิ่มค่าแล้วจะเป็นการกระโดดตามค่าของ PC โดยไม่มีผลต่อแฟล็กใดๆ การทำงานของคำสั่ง JNC จะแสดงตาม Flow Chart



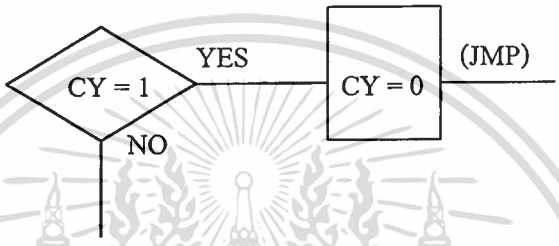
JB ทำการกระโดดก็ต่อเมื่อ บิตที่ชี้เป็น "1" เป็นคำสั่งกระโดด ถ้าบิตที่ถูกกำหนดมีค่าเป็น 1 คำสั่งจะกระโดดไปยังแอดเดรสที่กำหนด โดยมีจุดหมายปลายทางจากผลของการบวกกันด้วยค่า signed relative displacement ไบต์ที่ 3 ของคำสั่งนี้กับค่าของ PC ที่ชี้คำสั่งไบต์แรกของคำสั่งตัวต่อมา แล้วเก็บไว้ที่ PC หลังจากเพิ่มค่าแล้วจะเป็นการกระโดดตามค่าของ PC โดยไม่มีผลต่อแฟล็กใดๆ การทำงานของคำสั่ง JB จะแสดงตาม Flow Chart



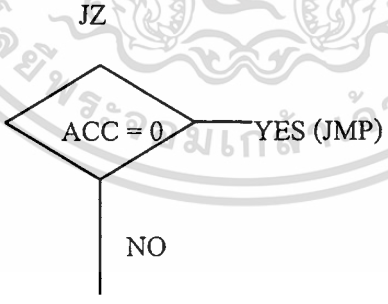
JNB ทำการกระโดดก็ต่อเมื่อ บิตที่ชี้ไม่เป็น "1" เป็นคำสั่งกระโดด ถ้าบิตที่ถูกกำหนดมีค่าเป็น 0 จะกระโดดไปยังแอดเดรสที่กำหนด ในคำสั่งด้วยค่าที่ได้จากการบวกกันของค่า signed relative displacement ไบต์ที่ 3 ของคำสั่งนี้กับค่าของ PC ที่ชี้ไบต์แรกของคำสั่งตัวต่อมา แล้วเก็บไว้ที่ PC หลังจากเพิ่มค่าแล้วจะเป็นการกระโดดตามค่าของ PC โดยที่ไม่มีผลต่อแฟล็กใดๆ การทำงานของคำสั่ง JNB จะแสดงตาม Flow Chart



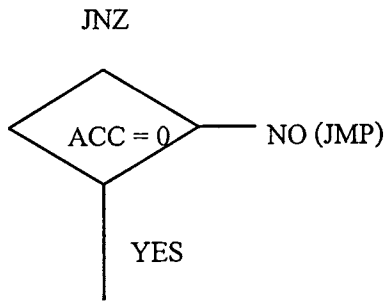
JBC ทำการกระโดดถ้าบิตที่ถูกกำหนดเป็น 1และจะทำการเคลียร์บิตนั้น เป็นคำสั่งกระโดดถ้าบิตที่ถูกกำหนดมีค่าเป็น 1 คำสั่งเคลียร์บิตตามตำแหน่งนั้นและ กระโดดไปยังแอดเดรสที่กำหนดไปยังจุดปลายที่ได้จากผลของการบวกกัน ด้วยค่า signed relative displacement ไบต์ที่ 3 ของคำสั่งนี้กับค่าของ PC ที่ชี้คำสั่งไบต์แรกของคำสั่งตัวต่อมา แล้วเก็บไว้ที่ PC หลังจากเพิ่มค่าแล้วจะเป็นการกระโดดตามค่าของ PC โดยไม่มีผลต่อแฟล็กใดๆ การทำงานของคำสั่ง JBC จะแสดงตาม Flow Chart



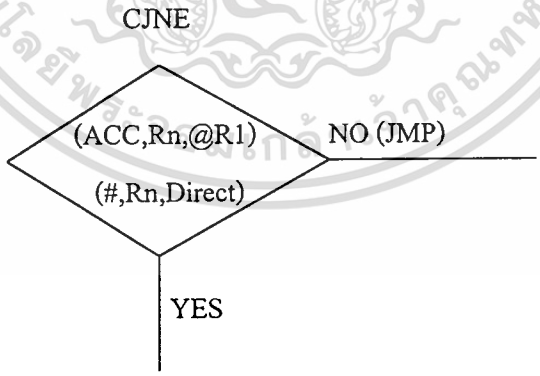
JZ เป็นคำสั่งกระโดด ถ้าค่าในแอคคิวมูเลเตอร์เป็น 0 คำสั่งจะกระโดดไปยังแอดเดรสที่กำหนด โดยมีจุดหมายปลายทางจากผลของการบวกกันด้วยค่า signed relative displacement ของไบต์ที่ 3 ของคำสั่งนี้ กับค่าของ PC ที่ชี้คำสั่งไบต์แรกของคำสั่งตัวต่อมา แล้วเก็บไว้ที่ PC หลังจาก PC เพิ่มค่าแล้วจะเป็นการกระโดดตามค่า PC โดยที่ไม่มีผลต่อแฟล็กใดๆ การทำงานของคำสั่ง JZ จะแสดงตาม Flow Chart



JNZ เป็นคำสั่งกระโดด ถ้าค่าในแอคคิวมูเลเตอร์ไม่มีค่าเป็น 0 คำสั่งจะกระโดดไปยังแอดเดรสที่กำหนด โดยมีจุดหมายปลายทางจากผลของการบวกกันด้วยค่า signed relative displacement ของไบต์ที่ 3 ของคำสั่งนี้ กับค่าของ PC ที่ชี้คำสั่งไบต์แรกของคำสั่งตัวต่อมา แล้วเก็บไว้ที่ PC หลังจาก PC เพิ่มค่าแล้วจะเป็นการกระโดดตามค่า PC โดยที่ไม่มีผลต่อแฟล็กใดๆ การทำงานของคำสั่ง JNZ จะแสดงตาม Flow Chart

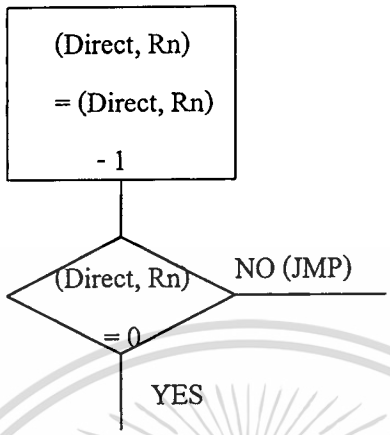


CJNE เป็นคำสั่งกระโดด โดยจะเปรียบเทียบค่าของข้อมูลโอเปอร์แรนด์สองตัวแรกซึ่งถ้าไม่เท่ากันก็จะกระโดดไปยังแอดเดรสที่ได้จากการรวมของค่าสัมพันธ์ที่คิดเครื่องหมายของไบต์ที่สามของชุดคำสั่งนี้กับค่าใน PC หลังจาก PC เพิ่มค่าไปเพื่อที่จะทำชุดคำสั่งต่อไป แฟล็กตัวทดจะเซตถ้าตัวเลขของจำนวนเต็มที่ไม่คิดเครื่องหมายของ ตำแหน่งของข้อมูล น้อยกว่าค่าตัวเลขจำนวนเต็มที่ไม่คิดเครื่องหมายของข้อมูลที่นำมาเปรียบเทียบ ถ้าเป็นอย่างอื่นคือ มากกว่า หรือ เท่ากับข้อมูลที่นำมาเปรียบเทียบ แฟล็กตัวทดจะ เคลียร์ ค่าข้อมูลโอเปอร์แรนด์จะไม่มีการเปลี่ยนค่าทั้งสองโอเปอร์แรนด์สามารถใช้โหมดการกำหนดเลขที่อยู่ได้ 4 โหมดโดยที่แอคคิวมูเลเตอร์อาจจะใช้เป็นตัวเปรียบเทียบตัวเลขที่ถูกกำหนดตำแหน่งในโหมดโดยตรงหรือโหมดโดยทันที และกำหนดตำแหน่งโดยทางอ้อมในตำแหน่ง RAM หรือตำแหน่งของข้อมูลในรีจิสเตอร์ทำงานกับข้อมูลโดยตรง การทำงานของคำสั่ง CJNE จะแสดงตาม Flow Chart



DJNZ เป็นการลดค่าตัวแปรลง 1 ค่า ถ้าผลลัพธ์ไม่เป็น 0 ก็จะทำการโดดไปยังแอดเดรสที่ถูกกำหนดจากโอเปอร์แรนด์ตัวที่ 2 หรือรหัสตัวสุดท้ายของคำสั่งที่คิดเครื่องหมายกับค่าใน PC ที่ชี้แอดเดรสของคำสั่งชุดต่อมา แต่ค่าที่ถูกกำหนดมีค่าเป็น 00H ก็จะถูกลดเป็น Underflow มีค่า 0FFH โดยที่ไม่มีผลต่อแฟล็กใดๆ ตำแหน่งที่ถูกกำหนดให้มีค่าลดลงอาจถูกกำหนดด้วยตัวรีจิสเตอร์ หรือโหมดการกำหนดเลขที่อยู่โดยตรงก็ได้ การทำงานของคำสั่ง DJNZ จะแสดงตาม Flow Chart

DJNZ

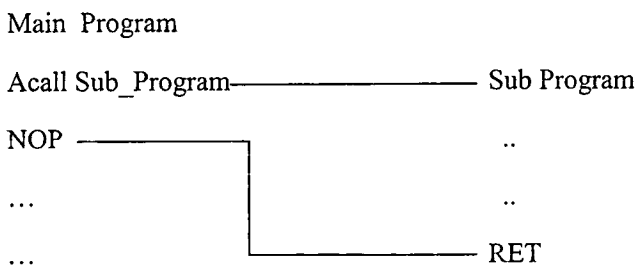


<u>หมายเหตุ</u> ถ้าใช้คำสั่งนี้ในการปรับค่าหรืออ่านจากพอร์ตเอาต์พุต ค่าที่ถูกใช้จะเป็นค่าพอร์ตข้อมูลที่แท้จริง ที่ถูกอ่านจากวงจรเอาต์พุตแลทช์ ไม่ใช่จากขาของ อินพุต

การเรียกโปรแกรมย่อย โดยสามารถกลับมายังโปรแกรมหลักได้

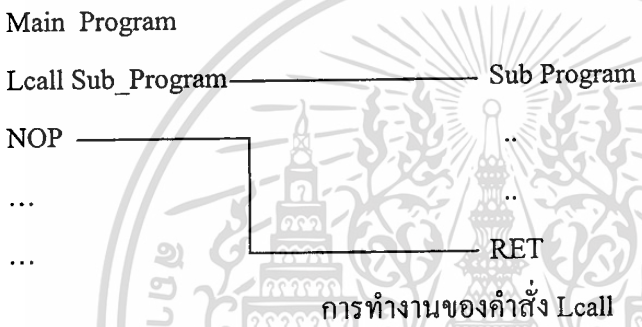
เป็นคำสั่งที่ใช้ในการเรียกโปรแกรมย่อยโดยที่ไม่มีเงื่อนไข แล้วสามารถกลับคืนสู่โปรแกรมหลักโดยคำสั่งในการเรียกโปรแกรมย่อย และ คำสั่งที่ใช้ในการกลับคืนสู่โปรแกรมหลักมีดังนี้

ACALL (Absolute Call) เป็นคำสั่งเรียกโปรแกรมย่อยในแอดเดรสที่กำหนดแบบไม่มีเงื่อนไข คำสั่งนี้จะเพิ่มค่าใน PC 2 ครั้ง เพื่อรับแอดเดรสจากคำสั่งต่อไป แล้วทำการเก็บค่าของ PC ของคำสั่งที่ต่อจากคำสั่งนี้ลงสแตค และเพิ่มค่าตัวชี้สแตค 2 ครั้ง โดยแอดเดรสที่ถูกเรียกเข้าใน PC จะได้จากรหัสบิตที่ 7 -5 และไบต์ที่สองของคำสั่ง Call ดังนั้นโปรแกรมย่อยจะอยู่ภายในขอบเขต 2 Kbyte ของหน่วยความจำโปรแกรม การทำงานของคำสั่ง Acall จะแสดงดังรูป



การทำงานของคำสั่ง Acall

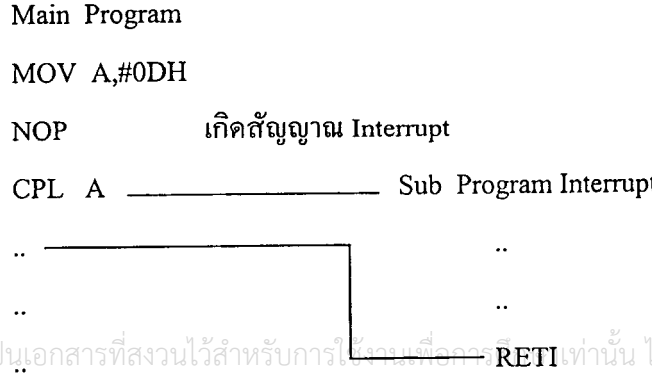
LCALL (Long Call) เป็นคำสั่งเรียกโปรแกรมย่อยในแอดเดรสที่กำหนดแบบไม่มีเงื่อนไข คำสั่งนี้จะเพิ่มค่าใน PC 2 ครั้ง เพื่อรับแอดเดรสจากคำสั่งต่อไปแล้วทำการเก็บค่าของ PC ของคำสั่งที่ต่อจากคำสั่งนี้ลงสแตค และเพิ่มค่าตัวชี้สแตค 2 ครั้ง โดยแอดเดรสที่ถูกโหลดเข้า PC ด้วยค่าไบต์สูง และไบต์ต่ำของแอดเดรสเริ่มต้นโปรแกรมย่อยที่ถูกเรียกด้วยคำสั่งนี้คือ ไบต์ตัวที่สอง และไบต์ตัวที่สามของคำสั่งรหัสคำสั่งตามลำดับ โปรแกรมก็จะทำงานอย่างต่อเนื่องด้วยคำสั่งที่แอดเดรสนี้ ดังนั้นตัวโปรแกรมย่อยสามารถจะอยู่ ณ ที่ใดก็ได้ของหน่วยความจำโปรแกรมขนาด 64 Kbyte โดยที่ไม่มี ผลต่อแฟล็กใดๆ การทำงานของคำสั่ง Lcall จะแสดงดังรูป



การทำงานของคำสั่ง Lcall

RET Return จาก โปรแกรมย่อยเป็นคำสั่งที่คืนค่าของ PC ที่เก็บไว้ใน สแตคซึ่งได้ถูกเก็บไว้เมื่อมีการคำสั่งในการเรียกโปรแกรมย่อยมาก่อน

RETI Return จาก Interrupt เป็นคำสั่งที่คืนค่าของ PC ที่เก็บไว้ในสแตค ได้ถูกเก็บไว้เมื่อเกิดการ Interrupt และจะกระโดดไปทำงานตามโปรแกรมย่อย ที่เกิดการ Interrupt เมื่อทำงานตามโปรแกรม Interrupt จำเป็นต้องมีคำสั่ง RETI จะคล้ายกับการใช้คำสั่ง RET เพื่อที่จะทำการกลับไปยังโปรแกรมหลักการทำงานของการใช้คำสั่ง RETI จะแสดงดังรูป



การทำงานของคำสั่ง RETI

PUSH จะทำงานโดยเพิ่มค่าในรีจิสเตอร์ SP ก่อน แล้วจึงถ่ายเทข้อมูลขนาด 1 ไบต์ จากตัวแหล่งกำเนิดที่ฟิลด์โอเปอร์แรนด์กำหนดไว้ ไปยังบริเวณสแตคตามตำแหน่งที่รีจิสเตอร์ SP กำหนด

POP การถ่ายเทข้อมูลขนาด 1 ไบต์ จากบริเวณสแตคตามตำแหน่งที่รีจิสเตอร์ที่โอเปอร์แรนด์กำหนด และหลังจากนั้น รีจิสเตอร์ SP จะลดค่าลงหนึ่ง

หมายเหตุ ข้อควรระวังในการใช้คำสั่ง PUSH และ POP คือ เมื่อทำการเก็บค่าด้วยคำสั่ง PUSH ค่าอะไรก่อนต้องทำการ POP ค่านั้นออกทีหลัง ซึ่งจะเห็นจากตัวอย่าง

```
PUSH   ACC
PUSH   DPL
PUSH   DPH
..
..
POP    DPH
POP    DPL
POP    ACC
```

และ เมื่อมีการเรียกโปรแกรมย่อยไม่ควรจะไปเปลี่ยนค่าของ SP (Stack Pointer) เพราะจะทำให้โปรแกรมไม่สามารถกลับไปยังโปรแกรมหลักได้

การทดลอง

1. ให้ทำการป้อนโปรแกรมที่ 1

```
        ORG    0000H
START:  MOV    R0,#20H        ; R0 = 20H
LOOP:   INC    R0             ; R0 = R0 + 1
        MOV    A,R0           ; A = R0
        CJNE   A,#40H,LOOP    ; กระโดด ถ้า A ไม่เท่ากับ 40H
        MOV    P1,A           ; P1 = A
        SJMP   $
```

โปรแกรมที่ 1

2. ให้ทำการสั่งให้เครื่องทำงานตามโปรแกรมแล้วให้ตรวจดูว่า PORT 1 มีค่าเท่ากับเท่าไร
3. ให้ทำการป้อนโปรแกรมที่ 2 เข้าเครื่อง

```
        ORG   0000H
        X     EQU   80H
START:  MOV   R0,#X        ; LOOP R0 = X
        CLR   A            ; A = 00H
LOOP:   INC   A            ; A = A+1
        DJNZ  R0,LOOP
        MOV   P1,A         ; PORT 1 = A
        SJMP  $
```

โปรแกรมที่ 2

4. ให้ทำการสั่งให้เครื่องทำตามโปรแกรมที่ 2 บันทึกค่าของ PORT 1

   PORT 1 =

5. ให้ทำการป้อนโปรแกรมที่ 3

```
        ORG   0000H
        X     EQU   80H
STSRT:  MOV   R0,#X        ; LOOP R0 = X
        CLR   A
LOOP:   LCALL T1           ; CALL SUB PROGRAM
        DJNZ  R0,LOOP
        MOV   P1,A
        SJMP  $
T1:     INC   A            ; A = A + 1
        RET
```

โปรแกรมที่ 3

6. ให้ทำการสั่งให้เครื่องทำงานตามโปรแกรมสังเกตค่าเปลี่ยนแปลงของ PORT 1 บันทึกค่า

   PORT 1 =

7. จากโปรแกรมที่ 2 และ 3 ให้ลองเปลี่ยนค่า X แล้วเปรียบเทียบการทำงานของทั้งสองโปรแกรม

คำถาม

1. จงเขียนโปรแกรมในการหน่วงเวลาโดยใช้คำสั่ง DJNZ, JNZ, JZ
2. จงอธิบายการทำงานของโปรแกรมที่ 2 โดยแสดงเป็น Flow Chart
3. จงอธิบายข้อแตกต่างในการใช้คำสั่ง Acall และ Lcall ในข้อดีและข้อเสียของทั้งสองคำสั่ง

ใบงานที่ 7
คำสั่งหมุนข้อมูล

วัตถุประสงค์

1. เพื่อให้นักเรียนได้รู้จักลักษณะของคำสั่งที่ใช้ในการหมุนข้อมูล
2. เพื่อให้สามารถอธิบายลักษณะของคำสั่งในการหมุนข้อมูลแต่ละแบบได้

ทฤษฎี

ในการเลื่อนข้อมูลหรือหมุนข้อมูลขนาด 4 บิต เราจะใช้คำสั่ง Rotate โดยคำสั่งนี้จะใช้สัญลักษณ "R" เป็นตัวบอกทิศทางการเคลื่อนที่ โดย "R" เป็นการเคลื่อนที่ไปทางขวา ส่วน "L" เป็นการเคลื่อนที่ไปทางซ้าย คำสั่งที่ใช้ในการหมุนข้อมูลนี้ กระทำได้กับรีจิสเตอร์ A เท่านั้น คำสั่งที่ใช้ในการหมุนมีทั้งหมด 4 คำสั่ง ได้แก่

RL A หมุนข้อมูลจากบิตต่ำไปหาบิตสูง

RLC A หมุนข้อมูลจากบิตต่ำไปหาบิตสูง นำ CY แฟล็กมาหมุนด้วย โดยอยู่ต่อจากบิต 7

RR A หมุนข้อมูลจากสูงต่ำไปหาบิตต่ำ

RRC A หมุนข้อมูลจากบิตสูงไปหาบิตต่ำ นำ CY แฟล็กมาหมุนด้วย โดยอยู่ต่อจากบิต 0

การทดลอง

1. พิมพ์โปรแกรมที่ 1 เข้าเครื่อง

```
        org 0000h
        mov a,#80h
        mov p1,a
loop:   acall delay
        rr a
        mov p1,a
        sjmp loop
        org 0100h
```

```
delay:  mov r0,#01h
dloop:  djnz r0,dloop
        ret
```

โปรแกรมที่ 1

2. ทำการ compile โปรแกรมแล้วเลือกอินเตอร์เฟส 2 รันโปรแกรมแล้วผลบันทึกผลที่จากโปรแกรม

ผลการทดลองโปรแกรมที่ 1

.....

.....

3. ป้อนโปรแกรมที่ 3 เข้าเครื่อง

```
org 0000h
        mov a,#01h
        mov p1,a
loop:   acall delay
        rl a
        mov p1,a
        sjmp loop
org 0100h
delay:
        mov r1,#01h
dloop:  djnz r1,dloop
        ret
```

โปรแกรมที่ 2

4. ทำการ compile โปรแกรมแล้วเลือกอินเตอร์เฟส 2 รันโปรแกรมแล้วผลบันทึกผลที่จากโปรแกรม

ผลการทดลองโปรแกรมที่ 2

.....

.....

5. ป้อนโปรแกรมที่ 3 เข้าเครื่อง

```
        org 0000h
loop2:  mov dptr,#0200h
        mov r1,#08h
loop:   mov a,#00h
        movc a,@a+dptr
        mov p1,a
        acall delay
        inc dptr
        djnz r1,loop
        mov dptr,#0200h
        sjmp loop2
org 0100h
delay:
        mov r0,#01h
dloop:  djnz r0,dloop
        ret
org 0200h
data:  db 81h,42h,24h,18h,18h,24h,42h,81h
```

โปรแกรมที่ 3

6. ทำการ compile โปรแกรมแล้วเลือกอินเตอร์เฟส 2 รันโปรแกรมแล้วผลบันทึกผลที่จากโปรแกรม

ผลการทดลองโปรแกรมที่ 3

.....

.....

คำถามท้ายการทดลอง

1. คำสั่ง RLC กับ RL ต่างกันอย่างไร

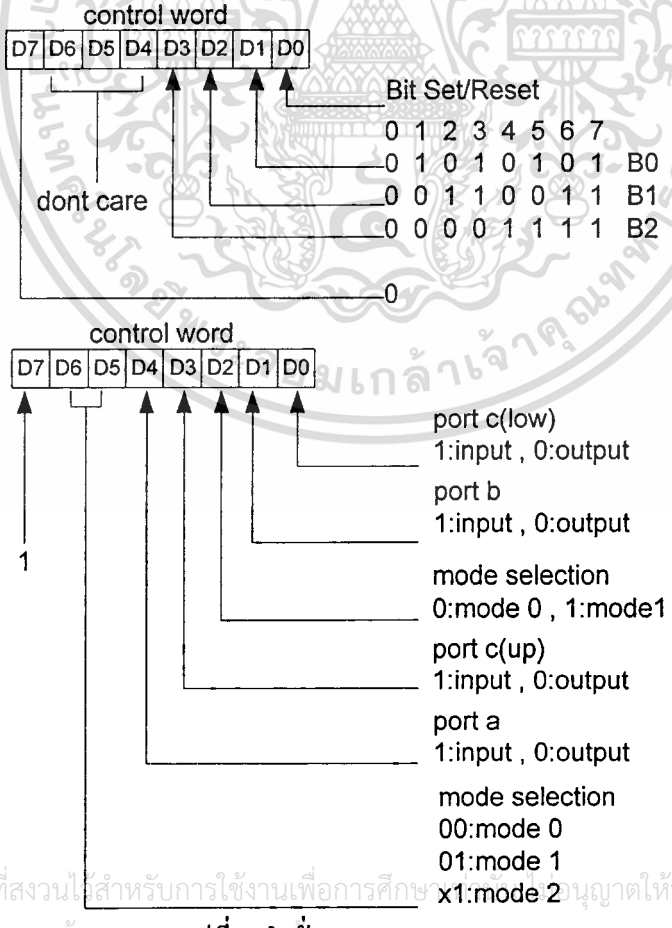
ใบงานที่ 8
การติดต่อกับ 8255

วัตถุประสงค์

1. เพื่อให้นักศึกษาสามารถเขียนโปรแกรมควบคุมการทำงานของ 8255 ได้
2. เพื่อให้นักศึกษาเข้าใจหลักการทำงานขอ 8255

ทฤษฎี

8255 สามารถทำงานได้ 3 โหมดด้วยกัน แต่ในใบงานฉบับนี้ใช้โปรแกรมจำลองการทำงาน MCS-51 ซึ่งโปรแกรมจำลองการทำงานนี้ 8255 จำลองสามารถทำงานได้ เพียงโหมดเดียว คือโหมด 0 เป็นโหมดที่ทำงานเป็นอุปกรณ์อินพุตเอาต์พุตแบบพื้นฐาน คือไม่มีการตรวจสอบ ในโหมด 0 ผู้ใช้สามารถกำหนดให้ พอร์ต PA, PB, PC เป็นอินพุตหรือเอาต์พุตก็ได้



รูปที่ 1 คำสั่งควบคุมของ 8255

ในโหมด 0 เป็นการทำงานแบบเอกทีฟ โดยจะต้องกำหนดให้บิตที่ 7 ของคำสั่งควบคุมมีค่าเป็น 1 หากกำหนดให้บิตที่ 7 มีค่าเป็น 0 จะเข้าสู่การทำงานในแบบนอนเอกทีฟ คือ สามารถใช้งานในพอร์ต PC เป็นเอาต์พุตได้เพียงอย่างเดียว และการกำหนดค่าเอาต์พุตของพอร์ต PC นั้นจะใช้การเลือกบิตและกำหนดให้บิตโดยกำหนดที่บิต 1ถึงบิต3 และกำหนดให้บิตที่เลือกมีสภาวะเซ็ต/เคลียร์

การทดลอง

1. พิมพ์โปรแกรมที่ 1 เข้าเครื่อง

```
porta  equ 2000h
portb  equ 2001h
portc  equ 2002h
ct     equ 2003h
       org 0000h
loop:  mov dptr,#data
       mov r0,#10h
ax1:   movc a,@a+dptr
       push dph
       push dpl
       mov dptr,#ct
       movx @dptr,a
       mov a,#0h
       pop dpl
       pop dph
       inc dptr
       djnz r0,ax1
       sjmp loop
data:  db 00h,02h,04h,06h,08h,0ah,0ch,0eh
       db 01h,03h,05h,07h,09h,0bh,0dh,0fh
```

โปรแกรมที่ 1

2. ทำการคอมไพโปรแกรมแล้วเลือก Interface3 แล้วเปลี่ยนตำแหน่งของ 8255 ให้อยู่ที่ตำแหน่ง 2000H
3. รันโปรแกรม แล้วบันทึกผลการทดลอง

ผลการทดลองโปรแกรมที่ 1

.....

.....

4. พิมพ์โปรแกรมที่ 2 เข้าเครื่อง

```
porta  equ 2000h
portb  equ 2001h
portc  equ 2002h
ct    equ 2003h
        org 0000h
        mov dptr,#ct
        mov a,#10010000b
        movx @dptr,a
loop:   mov dptr,#porta
        movx a,@dptr
        mov r1,a
        mov dptr,#portb
        movx a,#dptr
        add a,r1
        mov dptr,#portc
        mov @dptr,
        sjmp loop
end
```

โปรแกรมที่ 2

ผลการทดลองโปรแกรมที่ 2

.....

.....

คำถามท้ายการทดลอง

1. จากโปรแกรมที่ 2 จงนำมาแก้ไขให้เป็นโปรแกรมบวกเลข โดยรับค่าจากพอร์ต PA กับ PB เอาต์พุตออกที่พอร์ต PC
2. จากโปรแกรมที่ 2 จงนำมาแก้ไขให้เป็นโปรแกรมลบเลข โดยรับค่าจากพอร์ต PA กับ PB เอาต์พุตออกที่พอร์ต PC
3. จากโปรแกรมที่ 2 จงนำมาแก้ไขให้เป็นโปรแกรมคูณเลข โดยรับค่าจากพอร์ต PA กับ PB เอาต์พุตออกที่พอร์ต PC
4. จากโปรแกรมที่ 2 จงนำมาแก้ไขให้เป็นโปรแกรมหารเลข โดยรับค่าจากพอร์ต PA กับ PB เอาต์พุตออกที่พอร์ต PC

บรรณานุกรม

รัชนัย อินทุใส และไตรภพ อินทุใส. **ไมโครคอนโทรลเลอร์ 8051**. กรุงเทพ: หจก.สำนักพิมพ์ฟิสิกส์เซ็นเตอร์.

วิริ พงษ์แจ้ง. ผู้แปล **โปรแกรมย่อยภาษาแอสเซมบลี สำหรับ MS-DOS**. เขียนโดย Leo. J. Scanlon. กรุงเทพ: บริษัท ซีเอ็ดยูเคชั่น จำกัด(มหาชน), 2536.

สมยศ จุณณะปิยะ. **การใช้งานไมโครคอนโทรลเลอร์ ตระกูล MCS-51**. กรุงเทพ: สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังจัดพิมพ์, 2537.

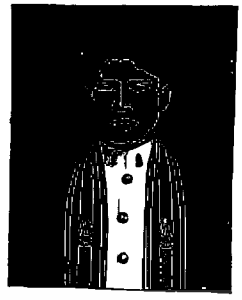
สุนทร วิทูสุรพจน์. **การใช้งานไมโครคอนโทรลเลอร์ ตระกูล 8051**. กรุงเทพ: บริษัท ซีเอ็ดยูเคชั่น จำกัด(มหาชน), 2537.

สุทธิศักดิ์ พงศ์ธนาพาณิช. ผู้แปล **การเขียนโปรแกรมบน 80386/80486**. เขียนโดย Ross P. Nelson. กรุงเทพ: บริษัท ซีเอ็ดยูเคชั่น จำกัด(มหาชน), 2537.

Osier, Dan.;Grobman, Steve.; and Batson, Stevw. **Teach yourself Delphi 2 in 21 days**. Sam Publishing, 1996.

Swan, Tom.,and Cogswell, Jeff. **Delphi 32-Bit Programming SECRETS**. IDG Books Worldwide, Inc. An International Data Group Company, 1996.

ประวัติผู้แต่ง



ชื่อผู้ทำปริญญานิพนธ์	นายธานินทร์ โพธิ์อำไพ
วันเดือนปีเกิด	5 กุมภาพันธ์ 2520
สถานที่เกิด	จังหวัดสมุทรปราการ
ภูมิลำเนาเดิม	จังหวัดสมุทรปราการ
ที่อยู่ปัจจุบัน	14 ม. 6 ต. บางเสาธง กิ่งอ. บางเสาธง
	จ. สมุทรปราการ 10540
โทรศัพท์	02-7083889

ประวัติการศึกษา

ประถมศึกษา	โรงเรียนวัดเสาธงนอก
มัธยมศึกษาตอนต้น	โรงเรียนบางบ่อวิทยาคม
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคฉะเชิงเทรา
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	วิทยาลัยเทคนิคฉะเชิงเทรา
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์
	ภาควิชาครุศาสตร์วิศวกรรม
	คณะครุสาสตร์อุตสาหกรรม
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	-
คติพจน์	ทำวันนี้ให้ดีที่สุด

ประวัติผู้แต่ง

ชื่อผู้ทำปริญญานิพนธ์	นายปัทมราช ดอกไม้
วันเดือนปีเกิด	6 ธันวาคม 2520
สถานที่เกิด	จังหวัดนครสวรรค์
ภูมิลำเนาเดิม	จังหวัดนครสวรรค์
ที่อยู่ปัจจุบัน	254/10 ม. 5 ต. พยุหะ อ. พยุหะคีรี จ. นครสวรรค์ 60130
โทรศัพท์	056-316337

ประวัติการศึกษา

ประถมศึกษา	โรงเรียนพยุหะวิทยา
มัธยมศึกษาตอนต้น	โรงเรียนพยุหะพิทยาคม
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคนครสวรรค์
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	สถาบันเทคโนโลยีราชมงคล วิทยาเขตเทคนิคกรุงเทพฯ
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุสาสตร์อุตสาหกรรม
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	-
คติพจน์	ใจเป็นนาย กายเป็นบ่าว

ประวัติผู้แต่ง

ชื่อผู้ทำปริญญานิพนธ์	นายสุภกิจ นุตยะสกุล
วันเดือนปีเกิด	22 เมษายน 2520
สถานที่เกิด	จังหวัดฉะเชิงเทรา
ภูมิลำเนาเดิม	จังหวัดฉะเชิงเทรา
ที่อยู่ปัจจุบัน	303/29 ซ.17 ถ. มหาจักรพรรดิ์ ต. หน้าเมือง อ. เมือง จ. ฉะเชิงเทรา 24000
โทรศัพท์	038-514545

ประวัติการศึกษา

ประถมศึกษา	โรงเรียนปัญจพิทยาคาร
มัธยมศึกษาตอนต้น	โรงเรียนผาณิตวิทยา
ประกาศนียบัตรวิชาชีพ (ปวช.)	วิทยาลัยเทคนิคฉะเชิงเทรา
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	วิทยาลัยเทคนิคฉะเชิงเทรา
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุสาสตร์อุตสาหกรรม
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	-
คติพจน์	หยุด