โปรแกรมจำลองการทำงานสำหรับ 8051 ไมโครคอนโทรลเลอร์

SIMULATOR PROGRAM FOR 8051 MICROCONTROLLER

โดย

นายพันธกร ชอบธรรม

นายสันติ แสงไกร

นายอำนาจ เชื้อนาค

ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมการวัดคุม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2542

ปริญญานิพนธ์ ปีการศึกษา 2542

ภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม

คณะ วิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง โปรแกรมจำลองการทำงานของ 8051 ไมโครคอนโทรลเลอร์

SIMULATOR PROGRAM FOR 8051 MICROCONTROLLER

ผู้จัดทำ

1. นายพันธกร	ชอบธรรม	40012093
2. นายสันติ	แสงไกร	40012108
3. นายอำนาจ	เชื้อนาค	40012115

.....อาจารย์ที่ปรึกษา

(ผศ. ทรงชัย วีระทวีมาศ)

.....อาจารย์ที่ปรึกษา

(อาจารย์ อาจินต์ น่วมสำราญ)

โดย นายพันธกร ชอบธรรม 40012093
นายสันติ แสงไกร 40012108
นายอำนาจ เชื้อนาค 40012115

อาจารย์ที่ปรึกษา ผ.ศ.ทรงชัย วีระทวีมาศ
อ.อาจินต์ น่วมสำราญ

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้เป็นการเสนอแนวทางการพัฒนา เพื่อใช้ในการจำลองการทำงานของไมโครคอนโทรลเลอร์ชิปเดี่ยวตระกูล 8051 ขึ้นมาใช้งาน โดยใช้โปรแกรมซิมูเลเตอร์เป็นเครื่องมือสำหรับช่วยตรวยสอบหรือแก้ไขระบบไมโครคอนโทลเลอร์ที่เกิดข้อผิดพลาดทางซอฟแวร์ ทั้งนี้เนื่องจากโปรแกรมซิมูเลเตอร์สามารถนำเอาข้อมูลที่มีความบกพร่องต่าง เช่น ข้อมูลภายในรีจีสเตอร์ ข้อมูลที่เก็บไว้ในหน่วยความจำสำหรับเก็บข้อมูลภายในหรือข้อมูลที่เก็บไว้ในหน่วยความจำสำหรับเก็บโปรแกรมขึ้นมาแสดงและตรวจสอบได้ และผู้ใช้สามารถตรวจสอบข้อมูลได้อย่างถูกต้อง การวิเคราะห์ปรับหาที่เกิดขึ้นก็สามารถทำได้อย่างรวดเร็วและถูกต้องแม่นยำทำให้ระยะเวลาที่แก้ปัญหาลดน้อยลง

Staff	Mr.Pantakon	Chobtum	40012093
	Mr. Santi	Sangkrai	40012108
	Mr. Umnat	Chuanak	40012115
Advisor	Asst.Prof.Songchai	Weerathaweemas	
	Mr.Arjin	Numsomran	

ABSTRACT

This paper presents the development of simulation for single chip microcontroller in 8051 family. A simulator program is frequency used for testing and solving trouble shoot in microcontroller systems having software error, because a simulator program can take some data form failure microcontroller systems sush data in registers, data in Internal memory or data in Program memory to show and check. After the user get the correct data, analysis of problem can be easily done and the problem solving time can be greatly reduced.

กิตติกรรมประกาศ

การจัดทำปริญญานิพนธ์ในครั้งนี้ สำเร็จลุล่วงไปได้ด้วยดี เพราะได้รับความเมตตาและความช่วยเหลือจาก ผศ. ทรงชัย วีระทวีมาศ และ อาจารย์อาจินต์ น่วมสำราญ ที่ได้ให้ความกรุณาแนะนำแก่ผู้ทำการวิจัยตลอดมา อีกทั้งยังเอื้อเฟื้อหนังสือและเอกสารต่าง ๆ ในการทำปริญญานิพนธ์ฉบับนี้ ผู้ทำการวิจัยขอกราบขอบพระคุณเป็นอย่างสูง

และที่ลืมเสียไม่ได้คือ ขอกราบขอบพระคุณ คุณพ่อ และคุณแม่ ที่เคารพรักยิ่ง ตลอดจนสมาชิกในครอบครัว พี่ ๆ น้อง ๆ และเพื่อนร่วมงานทุกคนที่ให้การสนับสนุนและเป็นกำลังใจที่ยิ่งใหญ่ในการทำปริญญานิพนธ์ฉบับนี้

สุดท้ายนี้ขอขอบคุณ ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่มอบโอกาสให้ได้มีการทำโครงงานวิจัย เพื่อเสริมสร้างทักษะและความรู้ ให้กับผู้ทำการวิจัยและให้การสนับสนุนจนทำให้โครงงานวิจัยและปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

สารบัญภาพ

หน้า

สารบัญภาพ (ต่อ)

หน้า

สารบัญภาพ (ต่อ)

หน้า

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมา

ปัจจุบันนี้มีการนำเอาระบบไมโครคอนโทรเลอร์เข้ามาใช้งานเป็นส่วนสำคัญในวงการอุตสาหกรรมกันอย่างแพร่หลาย โดยเริ่มตั้งแต่ Z80 ไมโครโปรเซสเซอร์ ขนาด 8 บิต ในอดีตเป็นต้นมา แม้กระทั่งในปัจจุบัน Z80ไมโครโปรเซสเซอร์ ยังคงเป็นที่นิยมสำหรับผู้ที่เริ่มต้นศึกษาเรื่องเกี่ยวกับไมโครโปรเซสเซอร์อยู่ ทั้งนี้เพราะ Z80 มีโครงสร้างที่ไม่ซับซ้อน และใช้ชุดคำสั่งที่เข้าใจง่าย และในขณะเดียวกันไมโครคอนโทรลเลอร์ขนาด 8 บิต ตระกูล MCS-51 ของบริษัท INTEL ก็เริ่มเข้ามามีบทบาทที่สำคัญต่อการศึกษา และการนำไปประยุกต์ใช้ในงานอุตสาหกรรม ดังจะเห็นได้จากการที่สถานศึกษาหลาย ๆ แห่งได้จัดให้มีการเรียนการสอนเกี่ยวกับไมโครคอนโทรลเลอร์ตระกูล MCS-51 ให้กับนักศึกษา เหตุผลประการสำคัญที่ไมโครคอนโทรลเลอร์ตระกูล MCS-51 เข้ามามีบทบาทอย่างรวดเร็ว และมีผู้นำไปประยุกต์ใช้งานกันอย่างแพร่หลายนั้น เนื่องจากไมโครคอนโทรลเลอร์ตระกูล MCS-51 นี้มีขอบเขตและความสามารถสูงกว่าไมโครโปรเซสเซอร์ธรรมดาทั่ว ๆ ไป

และเนื่องจากทางสถาบันก็ได้มีหลักสูตรการเรียนการสอนเกี่ยวกับไมโครคอนโทรลเลอร์ โดยใช้ซิงเกิ้ลบอร์ด (Single Board) ในการสอน วิธีนี้นักศึกษาจะต้องทำการแปลโปรแกรมที่เขียนขึ้นให้เป็นภาษาเครื่อง (Opcode) โดยการเปิดรหัสจากตารางชุดคำสั่ง เมื่อได้ Opcode แล้วจึงนำไปป้อนให้กับซิงเกิ้ลบอร์ด เพื่อทดสอบการทำงานของโปรแกรม ข้อดีของการพัฒนาด้วยซิงเกิ้ลบอร์ดคือ มีราคาถูก เหมาะสำหรับผู้ที่เริ่มต้นศึกษาไมโครคอนโทรลเลอร์ เนื่องจากโปรแกรมมอนิเตอร์ของซิงเกิ้ลบอร์ดจะประกอบด้วยโปรแกรมสนับสนุนในการเรียนรู้ทำให้ง่ายต่อการเขียนโปรแกรม และสามารถนำเอาซิงเกิ้ลบอร์ดไปประยุกต์ใช้เป็นเครื่องมือต่าง ๆ ได้นอกเหนือจากการพัฒนาโครงงาน ข้อเสียของการศึกษาด้วยซิงเกิ้ลบอร์ดได้แก่ ผู้เขียนไม่สามารถเขียนโปรแกรมใช้งานไมโครคอนโทรลเลอร์ได้ทุกส่วนเนื่องจาก โปรแกรมมอนิเตอร์ของซิงเกิ้ลบอร์ดได้ใช้งานบางส่วนของไมโคคอนโทรลเลอร์ไปแล้วเช่น พื้นที่หน่วยความจำโปรแกรม อินเตอร์รัพท์เวกเตอร์ เป็นต้น นอกจากนี้การเปิดรหัสจากตารางชุดคำสั่งเพื่อทำการแปลโปรแกรมนั้นใช้เวลานาน ทำให้ไม่สะดวกต่อการพัฒนาโปรแกรมที่ซับซ้อนหรือมีขนาดใหญ่

จากปัญหาที่เกิดขึ้นดังที่กล่าวมานั้น ด้วยเหตุนี้จึงมีการนำโปรแกรมซิมูเลเตอร์ (Program Simulator) เข้าใช้ประกอบการศึกษาด้วย โปรแกรมซิมูเลเตอร์เป็นการจำลองการทำงานของไมโคร

คอนโทรลเลอร์โดยใช้ซอฟแวร์ที่ให้ผลลัพท์ได้เช่นเดียวกับเมื่อทำงานด้วยฮาร์ดแวร์ เพียงแต่สภาวะของสัญญาณจากวงจรจริง ๆ นั้นเป็นการจำลองการทำงานด้วยซอฟแวร์ที่ทำงานบนเครื่องไมโครคอมพิวเตอร์ทั้งสิ้น ข้อดีของโปรแกรมซิมูเลเตอร์คือ มีราคาถูก เหมาะสำหรับผู้ที่ต้องการเริ่มต้นศึกษาไมโครคอนโทรลเลอร์ เช่นรีจีสเตอร์ ข้อมูลภายในหน่วยความจำ และส่วนอื่น ๆ ทำให้การทดสอบ และการตรวจสอบหาข้อผิดพลาดภายในโปรแกรมที่กำลังศึกษาอยู่ง่ายขึ้นโดยไม่ต้องใช้ฮาร์ดแวร์อื่นเพิ่มเติม

ภาพที่1.1 ตัวอย่างแสดงจอภาพการทำงานของโปรแกรม 8052 Simulator ของ Craig Steiner

1.2 วัตถุประสงค์ของปริญญานิพนธ์

1. เพื่อศึกษาโครงสร้างสถาปัตยกรรมภายใน ขั้นตอนการทำงาน รวมทั้งการใช้งานกลุ่มคำสั่งต่างของไมโครคอนโทรลเลอร์เบอร์ 8051
2. สามารถใช้โปรแกรมเคลไฟจำลองการทำงานของไมโครคอนโทรลเลอร์เบอร์ 8051ขึ้นมาเพื่อใช้งานภายใต้ระบบปฏิบัติการ Windows เพื่อที่จะสามารถนำโปรแกรมสำเร็จรูป ไปศึกษาและทดลองการทำงาน ในเครื่องคอมพิวเตอร์ส่วนบุคคลได้ เพื่อความสะดวกในการใช้งานของนักศึกษา ที่จะนำไปทดลอง หลังเวลาเรียนปกติได้ อีกทั้ง ยังสามารถตรวจสอบ ข้อผิดพลาดใน

การทำงานของโปรแกรม ในขณะที่ RUN โปรแกรม จะสามารถตรวจข้อมูลในหน่วยความจำต่าง ๆ ข้อมูลในรีจีสเตอร์ ข้อมูลในแฟลก และพอร์ตต่าง ๆ ได้อย่างระเอียด

1.3 ประโยชน์ที่ได้รับจากปริญญานิพนธ์

1. นักศึกษาที่ทำโครงงานได้รับความรู้เกี่ยวกับการทำงานของไมโครคอนโทรลเลอร์ และสามารถนำโปรแกรมเดลไฟมาเขียนโปรแกรมประยุกต์ใช้งานได้ตามต้องการ
2. มีเครื่องมือสำหรับศึกษา และพัฒนาเกี่ยวกับไมโครคอนโทรลเลอร์เบอร์ 8051 เพื่อนักศึกษาและผู้ที่สนใจจะศึกษาไมโครคอนโทรลเลอร์ขนาด 8 บิตเบอร์ 8051 และส่งเสริมความก้าวหน้าในการศึกษา เนื่องจากมีเครื่องมือพร้อมให้การสนับสนุนการวิจัย ค้นคว้าและการนำไปประยุกต์ใช้งาน

1.4 ขอบเขตของปริญญานิพนธ์

ทำการพัฒนาซอฟแวร์ของโปรแกรมซิมูเลเตอร์สำหรับไมโครคอนโทรลเลอร์เบอร์ 8051 โดยซอฟแวร์ที่พัฒนาขึ้นมานี้ทำงานบนระบบปฏิบัติการวินโดว์ซึ่งมีการติดต่อกับผู้ใช้ในรูปแบบกราฟฟิก จึงทำให้เป็นซอฟแวร์ที่ใช้งานง่าย และมีประสิทธิภาพ ซึ่งมีขอบเขตการทำงานได้ดังนี้

1. สามารถเขียนโปรแกรมและประมวลผลโปรแกรมคำสั่งต่าง ๆ ของไมโครคอนโทรลเลอร์เบอร์ 8051 ได้
2. สามารถประมวลผลโปรแกรมได้ทั้งแบบ Single และ แบบ Excecute Program ได้
3. สามารถตรวจสอบข้อมูลในหน่วยความจำต่าง ๆ ข้อมูลในรีจีสเตอร์ใช้งานต่าง ๆ รวมทั้งข้อมูลใน แฟลกและพอร์ตรับส่งข้อมูลต่าง ๆ ได้
4. สามารถหยุดการทำงานของโปรแกรม โดยแอดเดรสที่ถูกเรียกได้

บทที่2
หลักการและทฤษฎีที่เกี่ยวข้อง

ซอฟแวร์ของโปรแกรม 8051 ซิมูเลเตอร์นั้นประกอบด้วย 2 ส่วน ได้แก่ โปรแกรมการประมวลผลภายในของไมโครคอนโทรลเลอร์ ซึ่งจะทำหน้าที่รับค่าข้อมูลในรูปแบบของ File.Hex มาทำการประมวลผลตามเพื่อให้ได้ผลของการทำงานออกมาตามคำสั่งนั้น สำหรับการออกแบบโปรแกรมการประมวลผลภายในนั้น เลือกใช้ภาษาปาสคาล (Pascal) ในการพัฒนาโปรแกรม เพราะมีชุดคำสั่งที่เพียงพอต่อความต้องการอีกทั้งยังง่าย และสะดวกในการพัฒนา เมื่อทำการพัฒนาโปรแกรมประมวลผลได้สมบูรณ์เรียบร้อยแล้ว จึงพัฒนาส่วนของหน้าต่างของโปรแกรม 8051 ซิมูเลเตอร์ ส่วนรับค่าข้อมูลและส่วนแสดงผลด้วยโปรแกรม Delphi V.4 เพื่อเพิ่มความสะดวกและความสามารถในการใช้งาน เนื่องจากโปรแกรม 8051 ซิมูเลเตอร์นี้ทำงานภายใต้ระบบปฏิบัติการ Windows ที่เป็นการใช้งานแบบ Graphic Interface (GUI)

2.1 หลักการเบื้องต้นในการเขียนโปรแกรมด้วยเดลไฟ

การเขียนโปรแกรมด้วยเดลไฟเป็นเรื่องง่ายเมื่อเทียบกับการเขียนโปรแกรมกับวินโดวส์โดยตรงซึ่งก็คือการโปรแกรมด้วยวิธีเดิม จะขอกล่าวถึงลักษณะทั่ว ๆ ไปของเดลไฟ ซึ่งเป็นภาษาที่ผู้จัดทำได้นำมาเขียนโปรแกรมในโครงงานนี้

2.2 ยุคของการโปรแกรมบนวินโดวส์

ถึงแม้ยุคสมัยของวินโดวส์จะเริ่มมาไม่นาน แต่การเขียนโปรแกรมบนวินโดวส์ได้เปลี่ยนไปถึง 3 ยุคแล้ว คือ

1. ยุคแรกเขียนโปรแกรมด้วยภาษา C
2. ยุค โอ โอ พี เขียนโปรแกรมแบบ Opject-Oriented Programming ด้วยเทอร์โบปาสคาล บอร์แลนด์ปาสคาล หรือ C++ สำหรับการเขียนโปรแกรมบนวินโดวส์
3. ยุควิชวล (Visual) เป็นการเขียนโปรแกรมในลักษณะด้วยภาพโดยที่การเขียนโปรแกรมแบบ โอ โอ พี จะเขียนได้เร็วกว่าเมื่อใช้ภาษา C ประมาณ 5-10 เท่า แต่ทั้งนี้ไม่อาจเปรียบเทียบได้กับการโปรแกรมด้วยวิชวล โดยเฉพาะเมื่อใช้เดลไฟ

2.3 ชุดของเดลไฟ

บอร์แลนด์ได้ออกเดลไฟมา 2 ชุด คือ Delphi กับ Delphi Client/Server แตกต่างกันในเรื่องฐานข้อมูล ในส่วนของการเขียนโปรแกรมโดยทั่วไปจะเหมือนกัน

2.3.1 ความต้องการทางด้านฮาร์ดแวร์

ในคู่มือของเดลไฟไม่ได้แสดงความต้องการทางด้านฮาร์ดแวร์ แต่มีระบุอยู่ในเอกสารแจกจ่ายซึ่งกำหนดความต้องการไว้ดังนี้คือ

ซีพียู 386 ขึ้นไป (ควรเป็น 486 DX2-66 ขึ้นไป)

หน่วยความจำหลักอย่างน้อย 6 เมกกะไบต์ (ควรเป็น 8 เมกกะไบต์ขึ้นไป)

เนื้อที่ในฮาร์ดดิสก์ ตามการติดตั้ง (ควรมีเนื้อที่ว่างอย่างน้อย 80 เมกกะไบต์ขึ้นไป)

หากใช้ ซี พี ยู ที่มีความเร็วต่ำกว่า 486DX2-66 เมื่อดำเนินการใด ๆ แล้ว ในหลายกรณีจะต้องคอย ซึ่งอาจสร้างความสงสัยได้ เพราะไม่ได้แสดงเคอร์เซอร์เป็นรูปนาฬิกาทราย

2.3.2 การติดตั้ง

การติดตั้งกระทำบนวินโดวส์ โดยรันโปรแกรม INSTALL.EXE และควรติดตั้งตามคำแนะนำในการติดตั้ง ซึ่งจะเป็นการติดตั้งโดยสมบูรณ์ โดยกด ENTER ต่อทุกคำถาม และให้ปฏิบัติตามคำแนะนำที่ให้เปลี่ยนค่า

2.3.3 แอพพลิเคชั่นในชุดของเดลไฟ

เมื่อติดตั้งเดลไฟ โปรแกรมที่ติดตั้งจะสร้างหน้าต่างกลุ่ม (Group Window) ของเดลไฟ ซึ่งภายในประกอบด้วยไอคอนของแอพพลิเคชั่นต่าง ๆ ในชุดของเดลไฟจำนวนหนึ่ง กล่าวคือ

1.Delphi คือตัวเดลไฟ ซึ่งเมื่อต้องการรันเดลไฟ ให้เปิดจากไอคอนนี้

2.WinSight และ WinSpector เพื่อช่วยในการตรวจหาที่ผิด

ไอคอนโปรแกรมนั้นเป็นของแอพพลิเคชั่นทางด้านฐานข้อมูล

2.3.4 หน้าต่างของเดลไฟ

เมื่อเปิดใช้เดลไฟ จะแสดงหน้าต่างดังรูปที่ ซึ่งประกอบด้วยหน้าต่าง 4 บาน คือ

1. หน้าต่างหลัก อยู่ด้านบนสุด แบ่งออกเป็น 4 ส่วนคือ

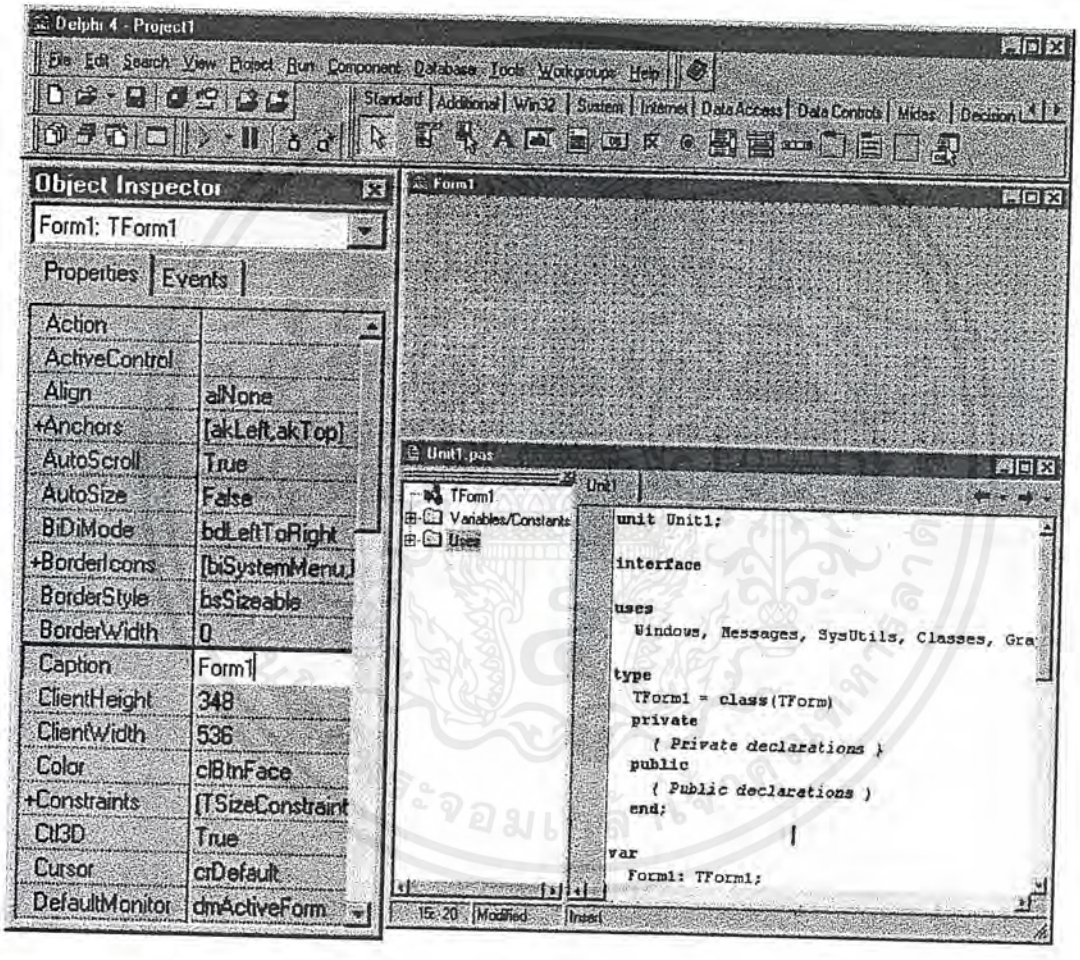
   -ไตเติลบาร์ขณะนี้แสดงข้อความว่า Delphi-Project1 ข้อความนี้จะเปลี่ยนไปตามสถานะ

   -เมนูบาร์ แสดงรายการ File ถึง Help

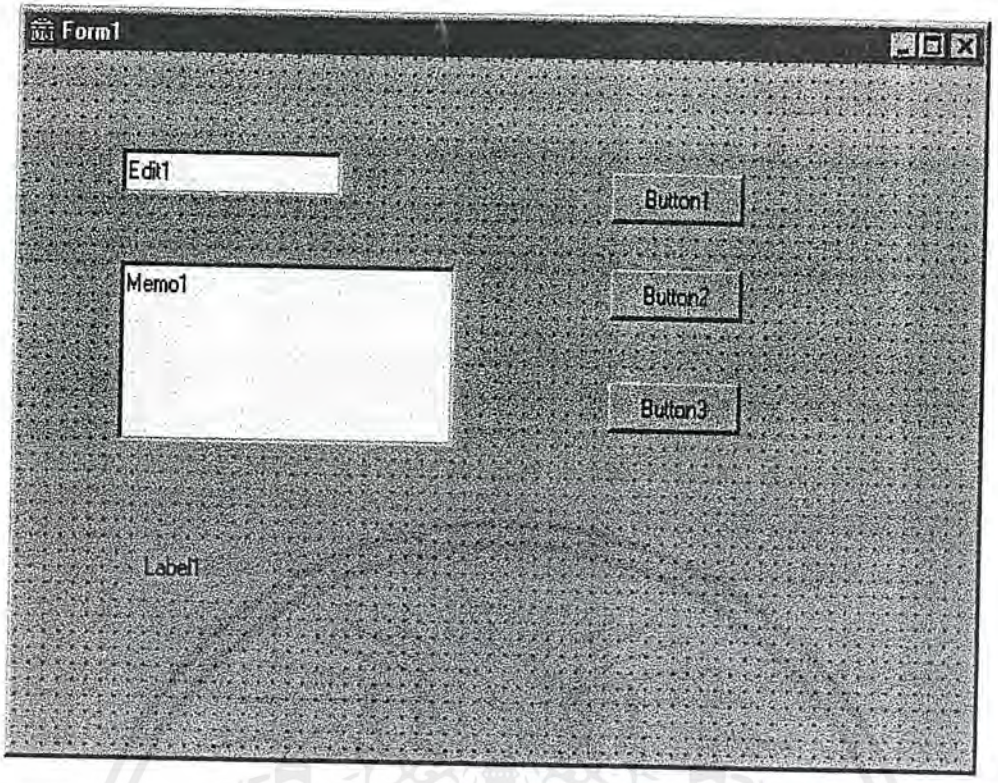
-สปีดบาร์ อยู่ใต้เมนูบาร์ระหว่าง File ถึง Help

-กล่องคอมโปเนนต์ (Component Palette) อยู่ทางด้านขวาของสปีดบาร์ แบ่งเป็นหน้าซ้อนทับกัน ดังที่ทำเป็นรูปที่คั่นหน้า (tab) ให้ชื่อเป็น Standard ถึง Samples เมื่อเลือกหน้าใด จะแสดงแถวของปุ่มคอมโปเนนต์ของหน้านั้น ในการอ้างถึงเพื่อช่วยให้หาปุ่มเหล่านี้ได้ง่าย จะอ้างชื่อหน้ากำกับ

กล่องคอมโปเนนต์มีไว้เพื่อให้เลือกคอมโปเนนต์ คือเลือกจากปุ่มคอมโปเนนต์ เช่นเป็น Bitbttn แล้วไปกำหนดในฟอร์ม เช่น ด้วยการคลิกเม้าส์ได้เป็นคอมโปเนนต์ฟอร์ม



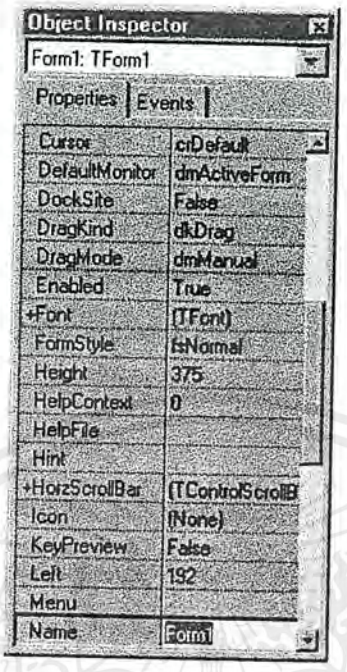
ภาพที่ 2.1 หน้าต่างของเดลไฟเมื่อเริ่มต้นรัน



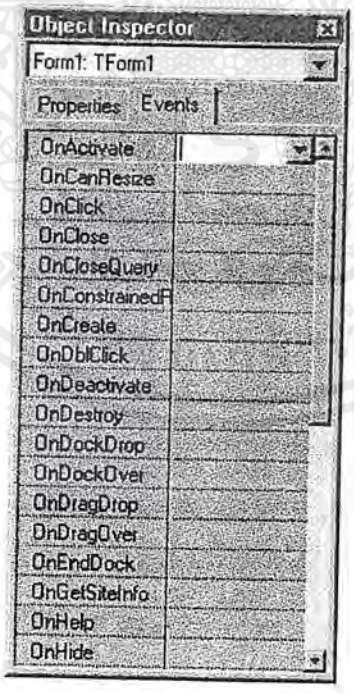
ภาพที่ 2.2 แสดงการกำหนดคอมโปเนนต์ในหน้าต่างฟอร์ม

2.หน้าต่างฟอร์ม (Form1) อยู่ด้านขวาใด้กล่องคอมโปเนนต์ เพื่อการวางรูปแบบของฟอร์ม ดังเช่นในรูปที่ ได้กำหนดคอมโปเนนต์เป็นปุ่มควบคุม Button จำนวน 3 ปุ่มและกรอบ Label จำนวน 1 กรอบ

3.หน้าต่างยูนิต (UNIT.PAS) ซ้อนทับกับหน้าต่างฟอร์ม เพื่อการป้อนโปรแกรม



ภาพที่ 2.3 หน้าต่าง Object Inspectorหน้า Properties



ภาพที่ 2.4 หน้าต่าง Object Inspector หน้า Events

4. หน้าต่าง Object Inspector อยู่ทางด้านซ้ายใต้สปีดบาร์ดังภาพที่ แบ่งออกเป็น 3 ส่วนคือ

-ไตเติลบาร์ แสดงข้อความว่า Object Inspector

-กรอบรายการออปเจ็กต์ เป็นกรอบคอมโบอยู่ใต้ไตเติลบาร์

-หน้า Properties และหน้า Events พื้นที่ที่เหลือของสองหน้าต่างนี้ ตามที่ทำที่ดั่นหน้าไว้ด้านล่าง เมื่อเลือกหน้าใดก็จะแสดงค่าของหน้านั้น คือ

-หน้า Properties คือ ตามภาพที่ 2.3 เพื่อเลือกให้ค่าพร็อพเพอร์ตี้ตามต้องการ

-หน้า Events ดังในภาพที่ 2.4 เพื่อเป็นการกำหนดกิจกรรมอีเวนต์ คือกิจกรรมตอบสนองอีเวนต์

2.4 หลักการทำงานของโปรแกรม 8051-Simlator

การทำงานของ เริ่มต้นโดยการโหลด Intel Stard File เข้ามเก็บในหน่วยความจำจำลองของโปรแกรมพร้อมกับทำการล้างข้อมูลที่มีอยู่ในหน่วยความจำเดิม จากนั้นโปรแกรมจะทำการถอดรหัสเพื่อตรวจสอบค่าแอสเดรสเริ่มต้นของคำสั่งจากนั้นโปรแกรมเริ่มทำงานตามรหัสของคำสั่งซึ่งภายในโปรแกรมได้มีเงื่อนไขการทำงานของกลุ่มคำสั่งต่าง ๆ ตามหลักการทำงานของ MCS-51 หลังจากทำงานตามเงื่อนไขของคำสั่งจะมีการตรวจสอบสภาวะต่างของรีจีสเตอร์และแฟลกต่างๆ

2.5 โครงสร้างของ Intel Standard File

รูปแบบของ Intel Standard-File ประกอบด้วยเรคคอร์ด 2 ชนิด คือ เรคคอร์ดข้อมูล (Data Recoord) กับเรคคอร์ดสิ้นสุด (End of Record) โดยรูปแบบของเรคคอร์ดจะขึ้นต้นด้วยรหัสนำ 9 ตัวอักษร ตามด้วยข้อมูลและรหัสปิดท้าย 2 ตัวอักษร โดยมีรูปแบบดังนี้

ตัวอย่างเช่น :09800000742A...1F

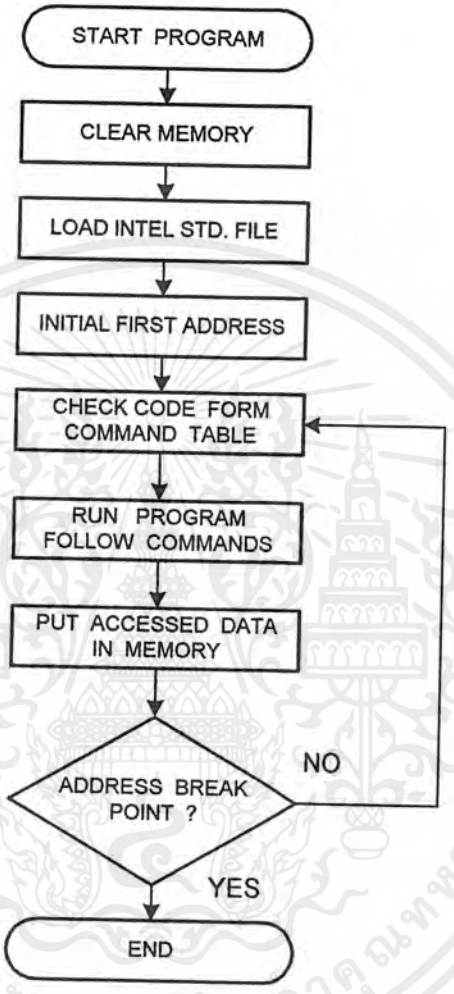
: เป็นตัวอักษรเริ่มต้น

09 เป็นจำนวนไบต์ของข้อมูลในเรคคอร์ด มีค่าเป็นเลขฐาน 16 (Hex)

8000 เป็นตำแหน่งข้อมูลไบต์เริ่มแรกของข้อมูลในเรคคอร์ด

00 แสดงชนิดของเรคคอร์ด เช่น จากตัวอย่าง 00 เป็นเรคคอร์ดของข้อมูล หรือ ถ้าเป็น 01 จะเป็นเรคคอร์ดสิ้นสุด

1F เป็นค่า CheckSum ซึ่งมีค่าเป็น 2'S Complement ของผลบวกข้อมูลทุกไบต์ในเรคคอร์ด

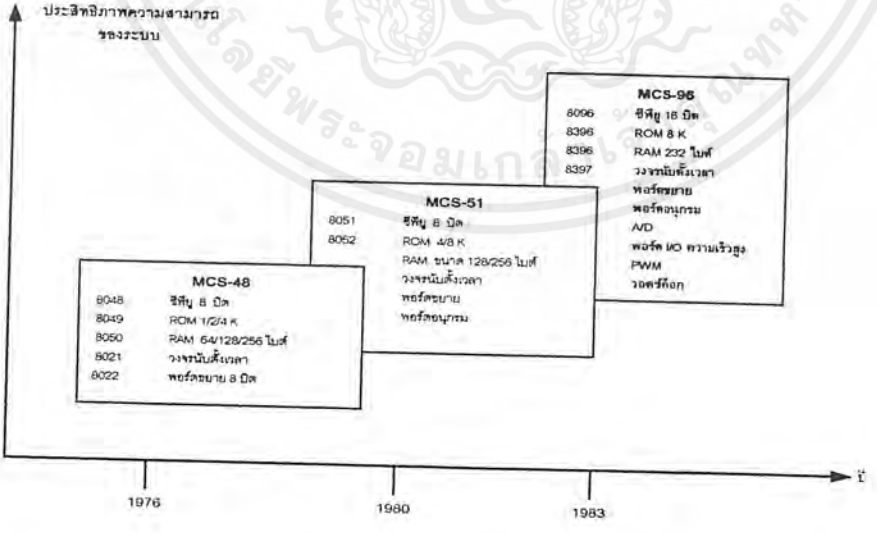


ภาพที่ 2.5 หลักการทำงานเบื้องต้นของโปรแกรม 8051 Simulator

บทที่ 3
ไมโครคอนโทรลเลอร์ตระกูล MCS – 51

3.1 การพัฒนาการทางเทคโนโลยีไมโครคอนโทรลเลอร์ของอินเทล

นับตั้งแต่บริษัท INTEL ได้ทำการผลิตไมโครคอนโทรลเลอร์ตระกูล MCS – 48 ขึ้นมาจนกระทั่งในปัจจุบันนี้ไมโครคอนโทรลเลอร์ในตระกูล MCS – 48 ได้มีการพัฒนาก้าวหน้าไปมากแต่ทางบริษัท INTEL ก็มิได้หยุดการพัฒนาไว้เพียงเท่านี้ INTEL ได้พยายามหาทางปรับปรุงไมโครคอนโทรลเลอร์ใหม่ ๆ ออกมาเพื่อหลีกเลี่ยงข้อจำกัดบางอย่างที่มีอยู่ในตระกูล MCS-48 และเพื่อประยุกต์ใช้ให้ทันกับความก้าวหน้าทางเทคโนโลยีที่มีการพัฒนาไปอย่างรวดเร็ว ดังนั้นไมโครคอนโทรลเลอร์ตระกูล MCS-51 จึงได้ถูกผลิตขึ้นมา โดยมีการใช้เทคโนโลยี HMOS ความจำมากกว่าในตระกูล MCS-48 ถึง 4 เท่า และยังเพิ่มความสามารถของวงจรภายในให้มีมากขึ้น เช่น มีพอร์ตสื่อสารข้อมูลแบบอนุกรมเพื่อรับหรือส่งข้อมูลได้ด้วยตัวเอง เพิ่มคำสั่งที่ใช้ในการเขียนโปรแกรมให้มากกว่าเดิมแต่ใช้เวลาแต่ละคำสั่งสั้นลง และคำสั่งที่เพิ่มขึ้นช่วยทำให้การเขียนโปรแกรมคล่องตัวและสะดวกขึ้นมากกว่าเดิมมาก ด้วยเหตุนี้การนำไมโครคอนโทรลเลอร์ตระกูล MCS-51 ไปประยุกต์ใช้งานต่าง ๆ จะสามารถทำได้สะดวกและคล่องตัวมากกว่าการใช้ไมโครคอนโทรลเลอร์ตระกูล MCS-48 มาก



ภาพที่ 3.1 แสดงการพัฒนาการทางเทคโนโลยีไมโครคอนโทรลเลอร์ของอินเทล

3.2 คุณสมบัติของไมโครคอนโทรลเลอร์ตระกูล MCS-51

โครงสร้างภายใน

-หน่วยความจำสำหรับเก็บโปรแกรม สูงสุด 64 กิโลไบต์ และหน่วยความจำสำหรับเก็บโปรแกรมภายใน 4 กิโลไบต์

-หน่วยความจำสำหรับเก็บข้อมูล

-หน่วยความจำสำหรับเก็บข้อมูลภายนอกชิป มีขนาดสูงสุด 64 กิโลไบต์ และใช้ PORT 0 เป็นแอสเดรสและดาต้าบัส และใช้ PORT 2 เพียง 3 เส้น เพื่อติดต่อกับหน่วยความจำที่ต้องการ

-หน่วยความจำสำหรับเก็บข้อมูลภายในชิป แบ่งเป็น 3 หน่วยย่อย

-หน่วยความจำสำหรับเก็บข้อมูลทั่วไป บริเวณ 128 ไบต์ล่าง

-หน่วยความจำสำหรับเก็บข้อมูลทั่วไป บริเวณ 128 ไบต์บน

-หน่วยความจำสำหรับเก็บข้อมูลของรีจีสเตอร์เฉพาะ (SFR)

- รีจีสเตอร์ใช้งานเฉพาะ การออกแบบสำหรับควบคุมระบบโดยเฉพาะ เช่น TCON, TMOD หรือรีจีสเตอร์ IE หรือ IP เป็นต้น

- รีจีสเตอร์ใช้งานทั่วไป เช่น รีจีสเตอร์ A,B หรือ R0-R7 อยู่ใน 128 ไบต์แรก มีด้วยกัน 4 กลุ่ม จึงมีด้วยกัน 32 ตัว

- โครงสร้างพอร์ต มีพอร์ต 4 บิต จำนวน 4 พอร์ต เพราะฉนั้นจึงใช้เป็นพอร์ต 1 บิต ได้ 32 พอร์ต

P0 ใช้งานเป็นดาต้าบัสและแอสเดรสบัสไบต์ต่ำ

P1 ใช้งานอินพุต เอาท์พุต พอร์ตทั่วไปได้

P2 ใช้งานอินพุต เอาท์พุต พอร์ตทั่วไปได้ และเป็นแอสเดรสบัสไบต์สูง

P3 ใช้งานสำหรับการเขียนและการอ่านข้อมูลจากภายนอก

- ไทม์เมอร์เคาน์เตอร์ มีขนาด 16 บิต ใช้สำหรับนับจำนวนสัญญาณนาฬิกาหรือแมชชีนไซเกิลของวงจรออสซิลเลเตอร์ภายใน หรือจำนวนครั้งของการเปลี่ยนแปลงสัญญาณจากภายนอก

การเข้าถึงข้อมูลมีหลายวิธี

- วิธีการเข้าถึงข้อมูลโดยตรง
- วิธีการเข้าถึงข้อมูลโดยทางอ้อม
- การเข้าถึงข้อมูลในรีจีสเตอร์ใช้งานทั่วไป

- การเข้าถึงข้อมูลในรีจีสเตอร์ใช้งานเฉพาะ
- วิธีการเข้าถึงข้อมูลาที่กำหนดโดยตรง
- วิธีการเข้าถึงข้อมูลโดยตัวอ้างอิง

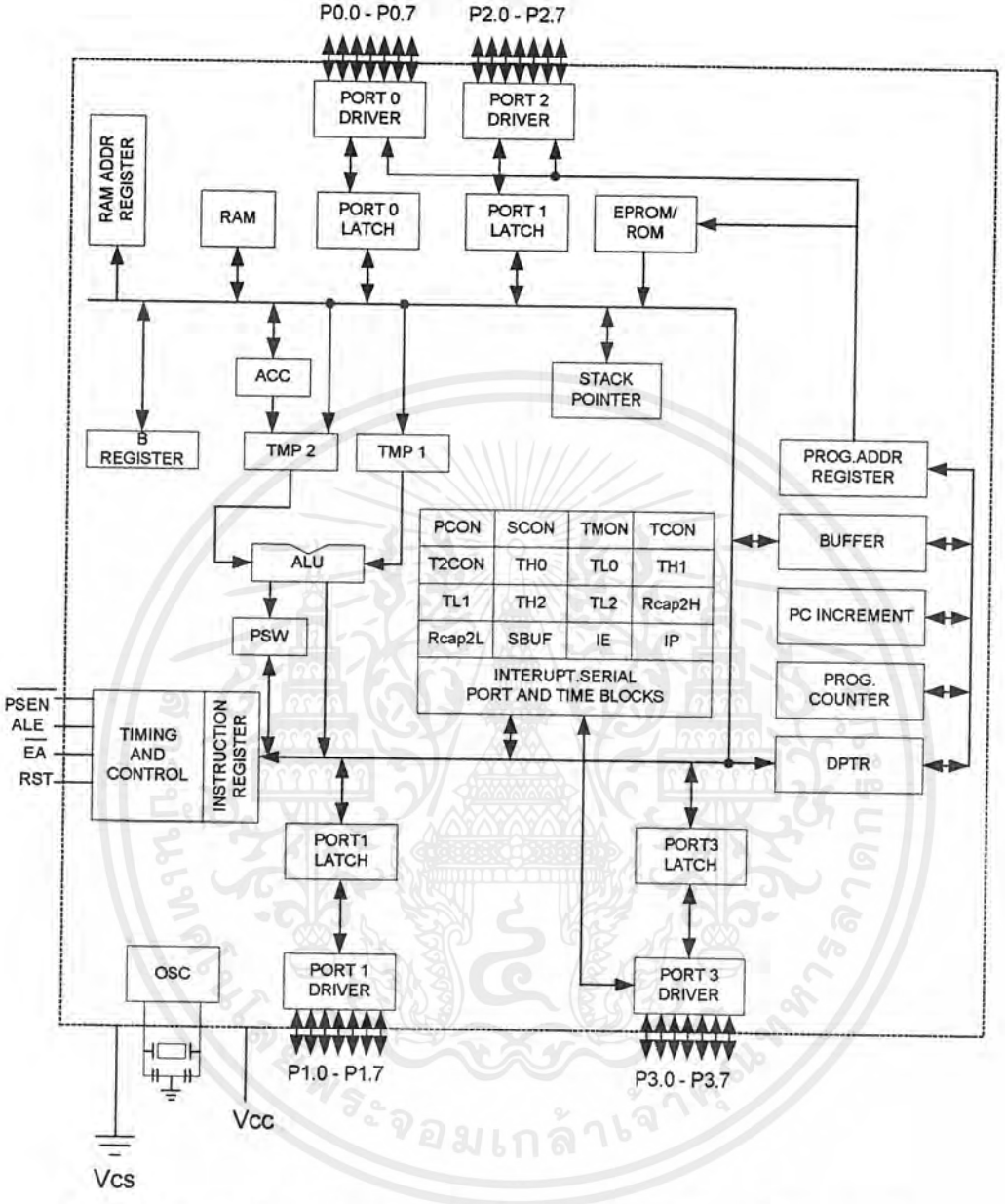
กลุ่มคำสั่งของ MCS-51

- กลุ่มคำสั่งทางคณิตศาสตร์
- กลุ่มคำสั่งทางตรรกศาสตร์
- กลุ่มคำสั่งเคลื่อนย้ายข้อมูล
- กลุ่มคำสั่งในการควบคุมลำดับการทำงานของโปรแกรม
- กลุ่มคำสั่งสำหรับการประมวลผลแบบบูลีน

3.3 โครงสร้างภายในของไมโครคอนโทรลเลอร์ตระกูล MCS–51

ในหัวข้อนี้จะกล่าวถึงรายละเอียดคร่าว ๆ ของไมโครคอนโทรเลอร์ตระกูล MCS–51 โดยมีจุดประสงค์เพื่อให้เข้าใจและมองเห็นภาพกว้าง ๆ ของไมโครคอนโทรลเลอร์ตระกูลนี้ เพื่อเป็นพื้นฐานในการศึกษารายละเอียดต่อไป หน่วยความจำสำหรับเก็บโปรแกรมจะเก็บโปรแกรมควบคุมการทำงานของชิป MCS-51 มีสมาชิกในตระกูลหลายเบอร์ด้วยกัน แต่ละเบอร์จะมีคุณสมบัติพิเศษบางอย่างแตกต่างกัน เช่น มีหน่วยความจำสำหรับเก็บโปรแกรมและข้อมูลภายในเพิ่มขึ้น มีวงจรเปลี่ยนค่าสัญญาณอะแนล็อกเป็นดิจิตอลในตัว สามารถรับสัญญาณอินเตอร์รัพต์ได้หลายชนิด ทำกระบวนการ DMA (Direct Memory Access) ได้ในตัว มีรีจีสเตอร์ใช้สำหรับเป็นไทม์เมอร์หรือเคาน์เตอร์เพิ่มขึ้น

ไมโครคอนโทรลเลอร์เบอร์ที่นับได้ว่าเป็นเบอร์พื้นฐานสำหรับตระกูล MCS-51 นี้ได้แก่เบอร์ 8051, 8031, 8751 โดยเบอร์ 8051 จัดเป็นสมาชิกตัวแรกในตระกูล ซึ่งมีหน่วยความจำสำหรับเก็บโปรแกรมภายในชิปเป็น ROM ขนาด 4 ไบต์ มีพอร์ตขนาด 8 บิต 4 พอร์ต และหน่วยความจำสำหรับเก็บข้อมูลทั่วไปภายใน MCS-51 (RAM) เองจำนวน 128 ไบต์ มีพอร์ตขนาด 8 บิต 4 พอร์ต มีรีจีสเตอร์สำหรับใช้เป็นไทม์เมอร์หรือเคาน์เตอร์ขนาด 16 บิตรวม 2 ตัว รับสัญญาณอินเตอร์รัพต์จากภายนอกได้ 2 ชนิด สามารถรับและส่งข้อมูลแบบอนุกรมผ่านทางพอร์ตสื่อสารข้อมูลแบบอนุกรมมีวงจรเพื่อสร้างสัญญาณนาฬิกาเพื่อควบคุมการทำงานในตัวเอง



ภาพที่ 3.2 แสดงโครงสร้างภายในของชิปไมโครคอนโทรลเลอร์ตระกูล MCS-51

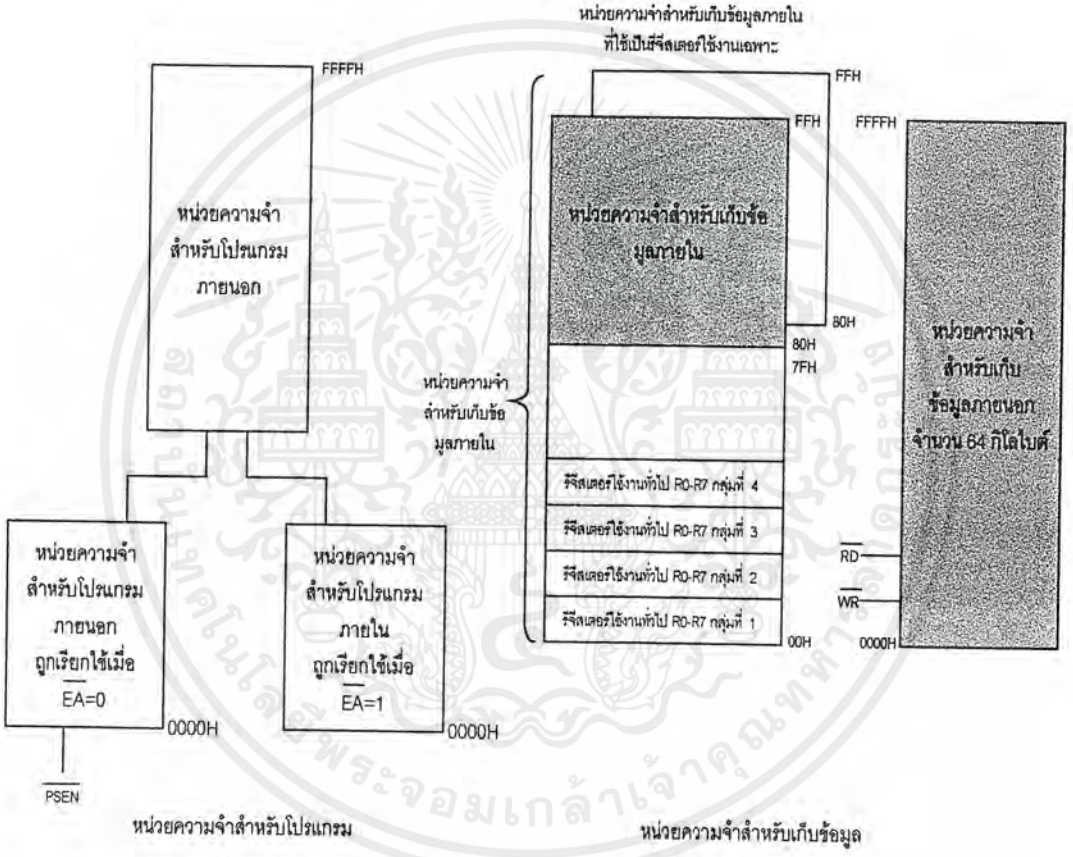
3.4 การแบ่งหน่วยความจำ

ในไมโครคอนโทรลเลอร์ตระกูล MCS – 51 แบ่งชนิดหรือหน้าที่ของหน่วยความจำออกเป็น 2 ส่วนคือ

หน่วยความจำโปรแกรม (Program memory) และหน่วยความจำข้อมูล (Data memory)

หน่วยความจำโปรแกรมจะใช้สำหรับเก็บโปรแกรมควบคุมการทำงานของไมโครคอนโทรลเลอร์ซึ่งบางเบอร์จะมีหน่วยความจำในส่วนนี้อยู่ภายในตัว โดยอาจจะมีขนาดไม่เท่ากัน

สำหรับหน่วยความจำข้อมูลจะใช้สำหรับเก็บข้อมูลค่าตัวแปรต่าง ๆ จากการทำงานของโปรแกรม ซึ่งใน MCS–51 ทุกตัวจะมีหน่วยความจำส่วนนี้อยู่จำนวนหนึ่ง แต่อาจมีขนาดมากหรือน้อยต่างกันไปในแต่ละเบอร์ สำหรับการจัดโครงสร้างของหน่วยความจำทั้งในส่วนของหน่วยความจำโปรแกรมและหน่วยความจำข้อมูลได้แสดงไว้ดังภาพที่ 3.3

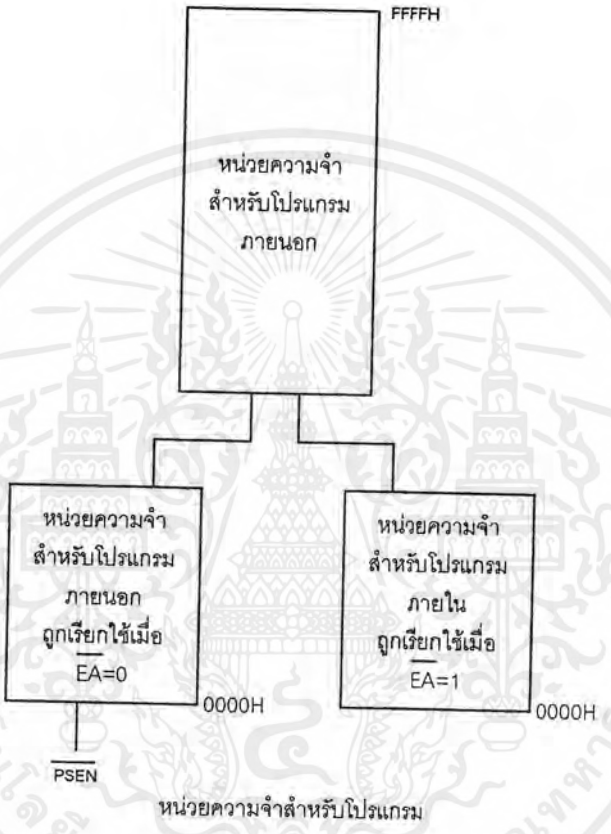


ภาพที่ 3.3 แสดงการจัดโครงสร้างของหน่วยความจำทั้งในหน่วยความจำโปรแกรม และหน่วยความจำข้อมูล

3.5 หน่วยความจำโปรแกรม

เป็นบริเวณหน่วยความจำ สำหรับเก็บข้อมูลและคำสั่งใช้งานต่าง ๆ หน่วยความจำโปรแกรมสามารถแบ่งออกได้เป็น 2 ส่วนคือ หน่วยความจำโปรแกรมภายในและหน่วยความโปรแกรมภายนอก หน่วยความจำโปรแกรมภายในนี้จะถูกเลือกใช้งานถ้าขาสัญญาณของ EA มีค่าเป็น 1 โดยจะถูกใช้งานในช่วงแอเดรส 0 – 0FFFH (หรือช่วง 0 – 1FFFF ในเบอร์ 8052)

นอกเหนือจากช่วงแอดเดรสไปจะใช้หน่วยความจำโปรแกรมภายนอกทั้งหมด ในกรณีตรงกันข้าม ถ้าขาสัญญาณ EA มีค่าเป็น 0 ในช่วงแอดเดรส 0 – 0FFFH (หรือช่วงแอดเดรส 0000H – 1FFFH ในเบอร์ 8052) จะถูกใช้จากหน่วยความจำภายนอก หรือกล่าวได้ว่า ถ้าขาสัญญา EA มีค่าเป็น 0 จะเป็นการเลือกหน่วยความจำโปรแกรมภายนอกทั้งหมดตลอดช่วงแอดเดรส

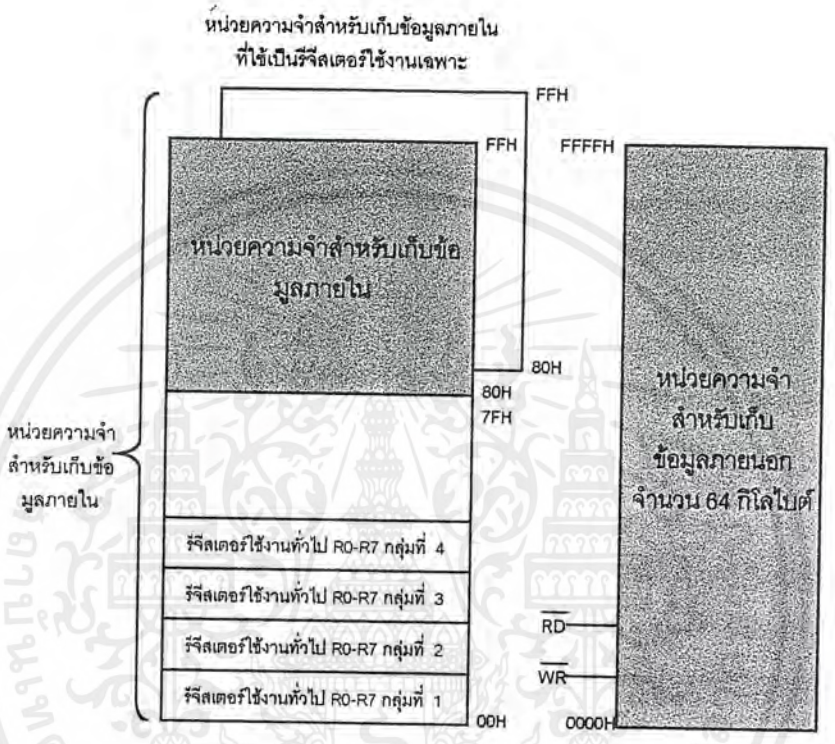


ภาพที่ 3.4 การจัดพื้นที่หน่วยความจำโปรแกรมสำหรับไมโครคอนโทรลเลอร์ 8051

3.6 หน่วยความจำสำหรับเก็บข้อมูล

ไมโครคอนโทรลเลอร์ตระกูล MCS – 51 สามารถทำการอ่านและเขียนข้อมูลจากหน่วยความจำข้อมูลที่มีขนาดสูงสุดได้ 64 กิโลไบต์ และหน่วยความจำในที่นี้ทำหน้าที่เก็บข้อมูลใช้งานจำนวนมากเป็นส่วนใหญ่ หน่วยความจำข้อมูลกำหนดให้มีตำแหน่งใช้งานได้ตั้งแต่ 00000H ถึง 08000H ซึ่งตามที่กล่าวมาแล้วหน่วยความจำสำหรับเก็บโปรแกรมภายนอกถูกกำหนดให้เริ่มที่ตำแหน่ง 04000H เป็นต้นไป นั่นคือโปรแกรมใช้งานจะต้องเริ่มประมวลผลที่ตำแหน่ง 04000H ขึ้นไปเสมอ หน่วยความจำข้อมูลภายในและหน่วยความจำข้อมูลภายนอก สำหรับหน่วยความจำข้อมูลภายในข้อมูลยังแบ่งออกได้เป็น 2 ส่วนย่อยคือ ส่วนที่เก็บข้อมูลทั่วไปและส่วนที่ใช้เป็นรีจิส

เตอร์หน้าที่พิเศษหรือ SFR (Speccial Function Register) โดยส่วนที่ใช้เก็บข้อมูลทั่วไปจะถูกใช้สำหรับเก็บข้อมูลหรือค่าตัวแปรต่าง ๆ จากการทำงานของโปรแกรม ส่วนรีจีสเตอร์หน้าที่พิเศษจะถูกใช้งานเป็นรีจีสเตอร์ควบคุมการทำงานและบอกสถานะการทำงานของไมโครคอนโทรลเลอร์

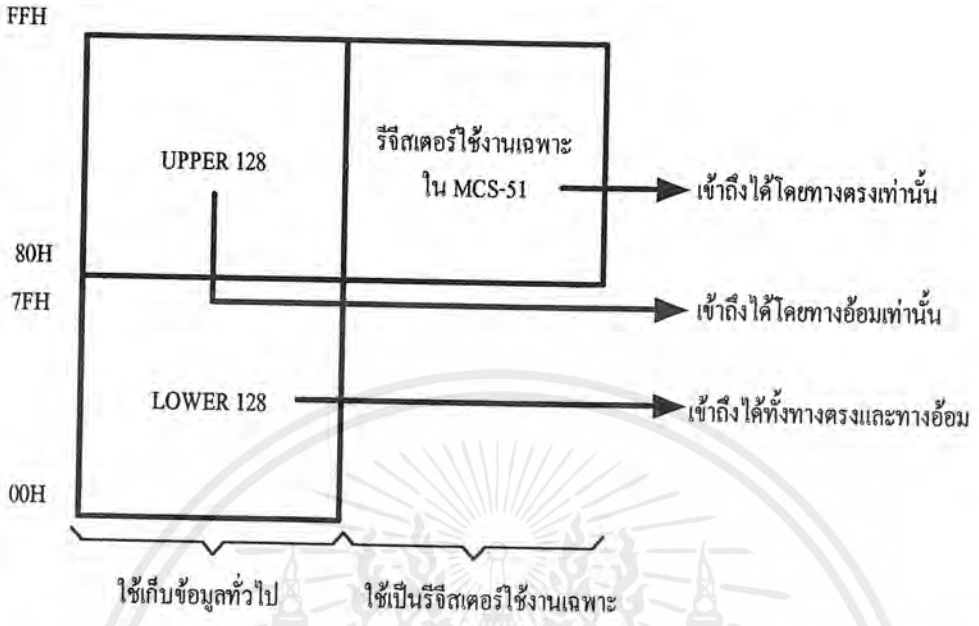


ภาพที่ 3.5 การจัดพื้นที่หน่วยความจำข้อมูลสำหรับไมโครคอนโทรลเลอร์ 8051

3.6.1 หน่วยความจำข้อมูลภายใน

หน่วยความจำข้อมูลในส่วนนี้จะถูกแบ่งออกเป็น 3 ส่วนย่อยดังนี้

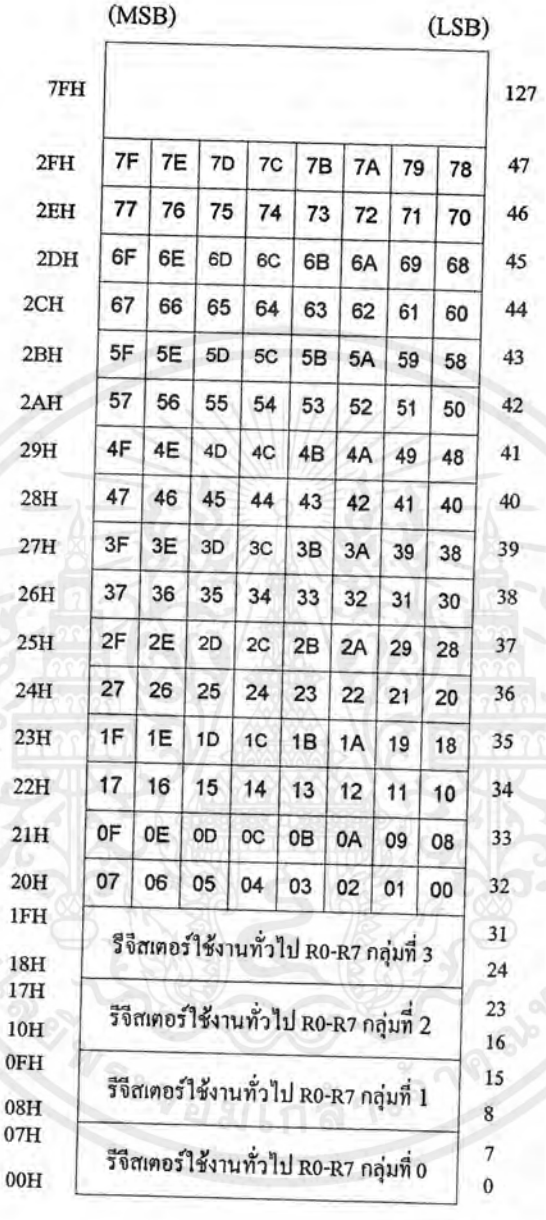
- หน่วยความจำสำหรับเก็บข้อมูลที่ใช้เก็บข้อมูลทั่วไปบริเวณ 128 ไบต์ล่าง (lower 128)
- หน่วยความจำสำหรับเก็บข้อมูลที่ใช้เก็บข้อมูลทั่วไปบริเวณ 128 ไบต์บน (upper 128)
- หน่วยความจำสำหรับเก็บข้อมูลที่ใช้เป็นรีจีสเตอร์ใช้งานเฉพาะ (SFR)



ภาพที่ 3.6 แสดงหน่วยความจำสำหรับเก็บข้อมูลทั้ง 3 ส่วน

บริเวณ 128 ไบต์บน (ตำแหน่ง 7FH-0FFH) จะใช้วิธีเข้าถึงข้อมูลโดยทางอ้อมเท่านั้น แต่สำหรับหน่วยความจำเก็บข้อมูลที่ใช้เป็นรีจีสเตอร์ใช้งานเฉพาะ (ตำแหน่ง 7FH-0FFH เช่นเดียวกัน) จะใช้วิธีเข้าถึงข้อมูลได้โดยทางตรงเท่านั้น ดังนั้นหน่วยความจำสำหรับเก็บข้อมูลทั้งสองบริเวณนี้จึงสามารถมีตำแหน่งซ้ำกันได้ โดยในระหว่างการทำงาน MCS-51 จะตรวจสอบจากรหัสคำสั่งเองว่าคำสั่งที่ต้องการทำงานมีการเข้าถึงข้อมูลในหน่วยความจำตำแหน่งใด และโดยวิธีไหน

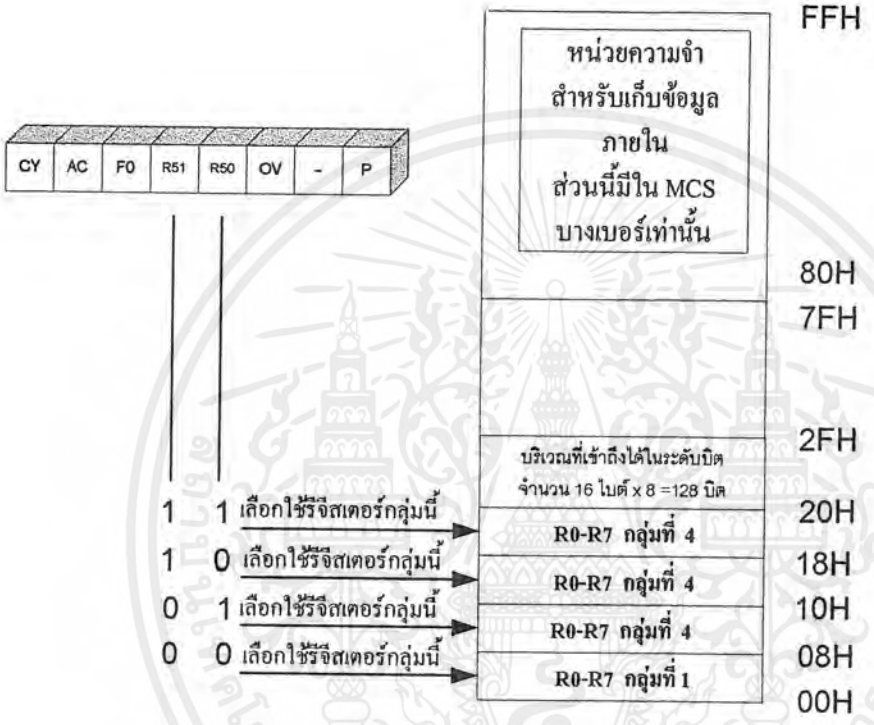
หน่วยความจำสำหรับเก็บข้อมูลภายใน บริเวณ 128 ไบต์ล่าง ตำแหน่ง 00H-1FH รวม 32 ไบต์ จะถูกกำหนดให้เป็นกลุ่มของรีจีสเตอร์ใช้งานทั่วไป 4 กลุ่ม กลุ่มละ 8 ตัว (R0-R7) กลุ่มรีจีเตอร์ใช้งานทั่วไปจะถูกเลือกใช้งานเพียงกลุ่มเดียวในขณะใดขณะหนึ่งเมื่อมีคำสั่งระบุให้ใช้ข้อมูลในรีจีเตอร์ใช้งานทั่วไป R0-R7 ซีพียูใน MCS-51 จะตรวจสอบเองว่าในขณะนั้นรีจีสเตอร์ใช้งานทั่วไปกลุ่มใดที่ถูกเลือกใช้งาน โดยดูจากบิต RS0,RS1 ในรีจีสเตอร์ใช้งานเฉพาะ PSW



ภาพที่ 3.7 แสดงตำแหน่งหน่วยความจำบริเวณ 128 ไบต์ล่างที่ใช้งานเป็นรีจีสเตอร์ทั่วไป และบริเวณที่สามารถเข้าถึงข้อมูลได้ในระดับบิต (bit addressable area)

หน่วยความจำสำหรับเก็บข้อมูลภายในที่อยู่ถัดจากกลุ่มรีจีสเตอร์ใช้งานทั่วไปทั้ง 4 กลุ่ม ตั้งแต่ตำแหน่ง 20H-2FH รวม 16 ไบต์ ได้ถูกออกแบบให้มีลักษณะ โครงสร้างพิเศษกว่าหน่วยความจำ

สำหรับเก็บข้อมูลบริเวณอื่น โดยแต่ละบิตของหน่วยของหน่วยความจำบริเวณนี้มีหมายเลขตำแหน่งกำหนดไว้แน่นอน เพื่อนำมาใช้เป็นข้อมูลขนาด 1 บิตในการทำงานของกลุ่มคำสั่งประมวลผลแบบบูลีนได้ แต่ละบิตของหน่วยความจำบริเวณนี้มีหมายเลขตำแหน่งเริ่มตั้งแต่ 00H-7FH รวม 128 ตำแหน่ง หรือ 128 บิต



ภาพที่ 3.8 แสดงการเลือกใช้กลุ่มรีจิสเตอร์ใช้งานทั่วไป โดยควบคุมจากบิต RS0,RS1

3.6.2 หน่วยความจำข้อมูลภายนอก

มีขนาดสูงสุด 64 กิโลไบต์ การใช้หน่วยความจำสำหรับเก็บข้อมูลภายนอก พอร์ต 0 จะถูกใช้เป็นแอสเดรสบัสและดาต้าบัส ส่วนพอร์ต 2 จะถูกใช้เพียง 3 เส้นเพื่อเป็นตัวเลือกช่วงหน่วยความจำที่ต้องการติดต่อเนื่องจากแอสเดรสที่ต้องการสำหรับหน่วยความจำขนาด 2 กิโลไบต์ คือ 11 เส้น (8 เส้นจากพอร์ต 0 รวมกับ 3 เส้นจากพอร์ต 2) ดังนั้นพอร์ต 2 ที่เหลือจากการติดต่อหน่วยความจำสำหรับเก็บข้อมูลสามารถนำไปใช้เป็นพอร์ตอินพุตหรือพอร์ตเอาท์พุตทั่วไปได้

หน่วยความจำสำหรับเก็บข้อมูลที่อยู่ภายนอกจำเป็นต้องมีสัญญาณควบคุมการอ่านและเขียนข้อมูล สัญญาณควบคุมการอ่านและการเขียนข้อมูลจะถูกส่งจาก MCS-51 ผ่านทางขา P3.7 และ P3.6 ตามลำดับ

3.7 พื้นที่หน่วยความจำที่เข้าถึงข้อมูลโดยทางตรงและทางอ้อม (Direct and Indirect Address Area)พื้นที่ 128 ไบต์ล่างสุดจะเป็น 3 ส่วนดังภาพที่ 3.9

1รีจิสเตอร์แบงค์ (Register Bank 0-3)

ตั้งแต่ตำแหน่ง (00h-1Fh) จะเป็นส่วนของรีจีสเตอร์แบงค์ (0-3) โดยแบ่งเป็นแบงค์ละ 8 ไบต์ (แต่ละแบงค์จะมีรีจีสเตอร์ R0,R1,R2,R3,R4,R5,R6,R7) ถ้าซีพียูทำงานอยู่ที่แบงค์ 3 เมื่อถูกรีเซ็ทจะกลับมาทำงานที่แบงค์ 0 เสมอ และ SP จะมาเริ่มต้นที่ตำแหน่ง 07h ทันที

2 บริเวณหน่วยความจำที่ใช้ตำแหน่งอ่านเขียนที่ละบิตได้ (Bit Address Area)

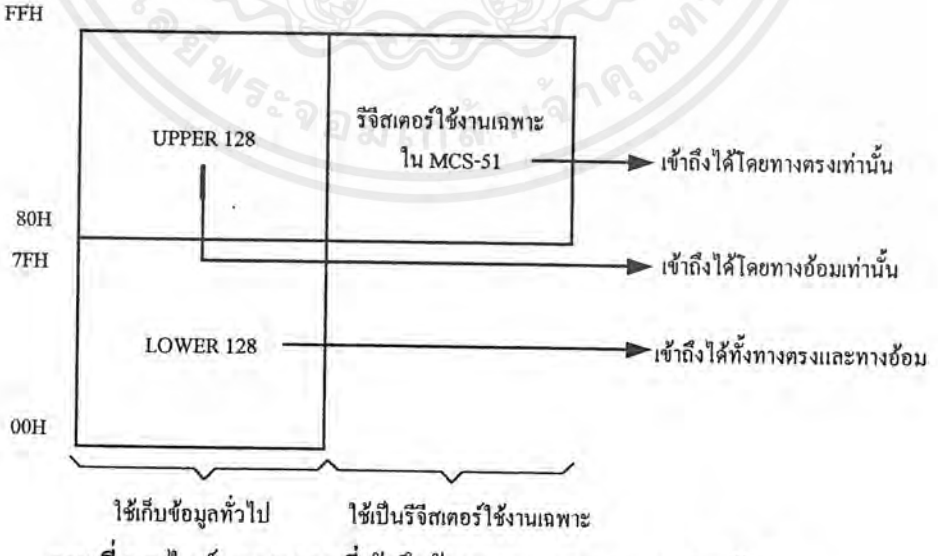
พื้นที่ตั้งแต่แอดเดรส (20h-7Fh) จำนวน 16 ไบต์หรือแบ่งเป็นบิตจะได้เท่ากับ 128 บิต ซึ่งตำแน่งบิตมีดังนี้ 00,01,02,03,04,05,06,07 จนถึง 7Fh

เช่น บิต 00 ก็คือ D0 ของหน่วยความจำตำแหน่ง 20h

บิต 01 ก็คือ D1 ของหน่วยความจำตำแหน่ง 20h

3 บริเวณหน่วยความจำที่ใช้งานทั่วไป (Scrath Pad Area)

พื้นที่ตั้งแต่ (30h-7Fh) จะเขียนข้อมูลได้ที่ละไบต์เท่านั้นไม่สามารถใช้คำสั่งเกี่ยวกับบิตได้ ถ้าย้ายเนื้อที่สแตคมายังบริเวณนี้โปรดระวังในการเขียนข้อมูลทับสแตค



ภาพที่ 3.9 ไบต์ของ RAM ที่เข้าถึงข้อมูลแบบทางตรงและทางอ้อม

3.8 การเข้าถึงข้อมูลในคำสั่ง

มีด้วยกันหลายวิธีดังนี้

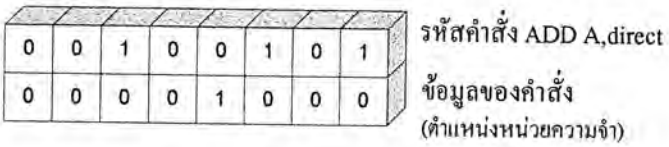
- วิธีการเข้าถึงข้อมูลโดยตรง (direct addressing)
- วิธีการเข้าถึงข้อมูลโดยทางอ้อม (indirect addressing)
- วิธีการเข้าถึงข้อมูลในรีจีสเตอร์ใช้งานทั่วไป (register instructions)
- วิธีการเข้าถึงข้อมูลในรีจีสเตอร์เฉพาะของคำสั่ง (register-specific instructions)
- วิธีการเข้าถึงข้อมูลที่กำหนดเองโดยตรง (immediate constants)
- วิธีการเข้าถึงข้อมูลที่มีตัวชี้อ้างอิง (indexed addressing)

คำสั่งแต่ละคำสั่งที่ต้องการโอเปอร์แรนด์อาจจะมีวิธีการเข้าถึงข้อมูลได้วิธีเดียวหรือหลายวิธีขึ้นอยู่กับชนิดของคำสั่งแต่ละคำสั่ง นอกจากนี้หน่วยความจำแต่ละบริเวณ ก็สามารถมีวิธีการเข้าถึงข้อมูลได้แตกต่างกัน

วิธีการเข้าถึงข้อมูลในแต่ละวิธีมีรายละเอียดดังนี้

3.8.1. วิธีการเข้าถึงข้อมูลโดยตรง วิธีนี้จะระบุค่าตำแหน่งหน่วยความจำที่เก็บข้อมูลโดยตรงในคำสั่งข้อมูลที่นำมาประมวลผลโดยวิธีนี้จะเป็นค่าของข้อมูลในหน่วยความจำสำหรับเก็บข้อมูลที่ใช้เก็บข้อมูลทั่วไปเฉพาะในบริเวณ 128 ไบต์ล่างและหน่วยความจำสำหรับเก็บข้อมูลที่ใช้เป็นรีจีสเตอร์ใช้งานเฉพาะเท่านั้น และเนื่องจากหน่วยความจำสำหรับเก็บข้อมูลที่ใช้เก็บข้อมูลทั่วไปในบริเวณ 128 ไบต์ล่างกับหน่วยความจำสำหรับเก็บข้อมูลที่ใช้เป็นรีจีสเตอร์ใช้งานเฉพาะมีขนาดรวมกันทั้งสิ้น 256 ไบต์ ดังนั้นค่าตำแหน่งหน่วยความจำที่ใช้ต้องเป็นเลขไบนารี่ขนาด 8 บิตเท่านั้น คำสั่งที่ใช้วิธีการเข้าถึงข้อมูลโดยตรงแบบนี้จะมีความยาวอย่างน้อยที่สุด 2 ไบต์ โดยไบต์แรกจะเป็นไบต์ที่บอกรหัสของคำสั่ง (opcode) ส่วนไบต์ที่สองซึ่งตามหลังไบต์ที่เป็นคำสั่งจะเป็นไบต์ที่ระบุค่าตำแหน่งหน่วยความจำที่มีข้อมูลที่ต้องการนำมาประมวลผล ดังตัวอย่างคำสั่งต่อไปนี้

ADD A,08H นำค่าในหน่วยความจำตำแหน่ง 08H มาบวกกับค่าในรีจีสเตอร์ A ผลลัพธ์ที่ได้เก็บไว้ในรีจีสเตอร์ A รหัสของคำสั่งมีดังนี้



3.8.2 วิธีการเข้าถึงข้อมูลโดยทางอ้อม ค่าตำแหน่งหน่วยความจำที่ต้องการจะติดต่อจะเก็บไว้ในรีจีสเตอร์เฉพาะของคำสั่ง ดังนั้นวิธีนี้เป็นการเข้าถึงข้อมูลโดยทางอ้อม นั่นคือ แทนที่ผู้เขียนโปรแกรมจะระบุค่าตำแหน่งข้อมูลโดยตรง วิธีนี้จะใช้ค่าที่เก็บไว้ในรีจีสเตอร์ที่ระบุในรหัสคำสั่งชี้ไปยังตำแหน่งของหน่วยความจำแทน หน่วยความจำที่สามารถใช้วิธีการเข้าถึงข้อมูลด้วยวิธีนี้จะเป็นหน่วยความจำสำหรับเก็บข้อมูลที่ใช้เก็บข้อมูลทั่วไปในบริเวณ 128 ไบต์ล่าง และ 128 ไบต์บน รวมทั้งหน่วยความจำสำหรับเก็บข้อมูลที่อยู่ภายนอกชิป รีจีสเตอร์ที่สามารถนำมาเป็นตัวชี้ (pointer) ตำแหน่งของหน่วยความจำมีดังนี้

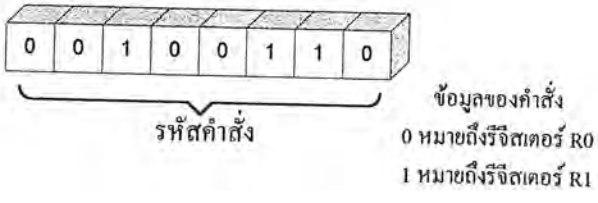
- รีจีสเตอร์ใช้งานทั่วไป R0, R1 ของแต่ละกลุ่มรีจีสเตอร์ทั้ง 4
- รีจีสเตอร์ใช้งานเฉพาะ SP (stack pointer)
- รีจีสเตอร์ใช้งานเฉพาะ DPTR (data pointer)

รีจีสเตอร์แต่ละตัวจะมีหน้าที่ไม่เหมือนกัน ซึ่งสามารถอธิบายหน้าที่การทำงานของรีจีสเ ตอร์ต่าง ๆ ข้างต้นดังนี้

รีจีสเตอร์ใช้งานทั่วไป R0, R1 เป็นรีจีสเตอร์ขนาด 8 บิต รีจีสเตอร์ทั้งสองสามารถชี้ตำแหน่งในหน่วยความจำได้เพียง 256 ตำแหน่งเท่านั้น โดยบริเวณหน่วยความจำของ 8051 ที่สามารถใช้รีจีสเตอร์ R0, R1 ระบุตำแหน่งได้นี้จะเป็นหน่วยความจำสำหรับเก็บข้อมูลที่ใชว้เก็บข้อมูลทั่วไปบริเวณ 128 ไบต์ล่าง ส่วนหน่วยความจำสำหรับเก็บข้อมูลที่ใช้เป็นรีจีสเตอร์ใช้งานเฉพาะไม่สามารถใช้รีจีสเตอร์ทั้งสองชี้ตำแหน่งนี้ได้ และต้องมีเครื่องหมาย @ นำหน้ารีจีสเตอร์ R0 หรือ R1 ด้วย

นอกจากจะสามารถใช้รีจีสเตอร์ใช้งานทั่วไป R0,R1 ชี้ตำแหน่งหน่วยความจำสำหรับเก็บข้อมูลที่ใช้เก็บข้อมูลทั่วไปที่อยู่ภายในชิปของ 8051 ได้แล้ว รีจีสเตอร์ทั้งสองยังสามารถชี้ตำแหน่งหน่วยความจำสำหรับเก็บข้อมูลที่อยู่ภายนอกชิปได้อีกด้วย แต่เนื่องจากรีจีสเตอร์ทั้งสองมีขนาดเพียง 8 บิต จึงมีข้อจำกัดตรงที่ไม่สามารถชี้ตำแหน่งหน่วยความจำสำหรับเก็บข้อมูลภายนอกได้เกิน 256 ไบต์ตัวอย่างคำสั่งที่ใช้รีจีสเตอร์ใช้งานทั่วไปเป็นตัวชี้ตำแหน่งข้อมูลมีดังนี้

ADD A,@R0 นำค่าในหน่วยความจำตำแหน่งที่ตรงกับค่าในรีจีสเตอร์ R0 มาบวกกับค่าในรีจีสเตอร์ A ผลลัพธ์ที่ได้เก็บไว้ในรีจีสเตอร์ A



รีจีสเตอร์ใช้งานเฉพาะ SP มีขนาด 8 บิท เหมือนรีจีสเตอร์ใช้งานทั่วไป R0, R1 แต่รีจีสเตอร์ใช้งานเฉพาะ SP ไม่สามารถชี้ตำแหน่งหน่วยความจำสำหรับเก็บข้อมูลที่อยู่ภายนอกชิปได้ รีจีสเตอร์ SP มีคำสั่งใช้เฉพาะคือ PUSH, POP, CALL ซึ่ง 8051 จะทำการเพิ่มหรือลดค่าในรีจีสเตอร์ SP ให้เอง ในสภาวะที่ 8051 ถูกรีเซตหรือเริ่มจ่ายพลังงาน ค่าในรีจีสเตอร์ SP จะมีค่าเท่ากับ 07H เสมอ

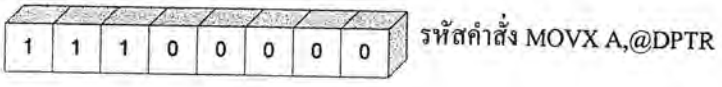
บริเวณหน่วยความจำที่รีจีสเตอร์ SP ชี้อยู่จะมีชื่อเรียกว่าสแตก ซึ่งเป็นบริเวณของหน่วยความจำที่เก็บไว้ใช้ปฏิบัติคำสั่งซึ่งใช้รีจีสเตอร์ SP เป็นตัวชี้ตำแหน่ง

ตัวรหัสคำสั่งของคำสั่งที่ใช้รีจีสเตอร์ใช้งานเฉพาะ SP เป็นตัวชี้ข้อมูล ไม่จำเป็นต้องระบุตำแหน่งของรีจีสเตอร์ SP เพราะ 8051 ทราบเองจากรหัสคำสั่งดังตัวอย่าง

PUSH ACC นำค่าในรีจีสเตอร์ A ไปเก็บไว้ในสแตก สมมติในขณะรีจีสเตอร์ใช้งานเฉพาะ SP มีค่า 15H

รีจีสเตอร์ใช้งานเฉพาะ DPTR เป็นรีจีสเตอร์ขนาด 16 บิต ดังนั้นรีจีสเตอร์ตัวนี้สามารถชี้ตำแหน่งหน่วยความจำสำหรับเก็บข้อมูลที่อยู่ภายนอกชิปได้มากถึง 64 กิโลไบต์ รีจีสเตอร์ตัวนี้ที่มีใช้ในคำสั่งที่ต้องการติดต่อกับหน่วยความจำสำหรับเก็บข้อมูลหรือโปรแกรมที่อยู่ภายนอกชิป 8051 ดังตัวอย่างต่อไปนี้

MOVX A,@DPTR นำค่าในหน่วยความจำตำแหน่งที่เท่ากับค่าในรีจีสเตอร์ DPTR ในขณะนั้นมาไว้ในรีจีสเตอร์ A



คำสั่งนี้ 8051 ทราบเองว่าใช้รีจีสเตอร์ DPTR

การเข้าถึงข้อมูลทางอ้อมโดยรีจีสเตอร์ทั้ง 4 ตัวนี้ โอเปอร์แรนด์ซึ่งเป็นรีจีสเตอร์ที่เก็บค่าตำแหน่งของข้อมูลที่ต้องประมวลผลจะต้องระบุเครื่องหมาย "@" ไว้ข้างหน้า ดังตัวอย่าง

MOV A,@R0 นำค่าในหน่วยความจำที่ชี้โดยรีจีสเตอร์ R0 ไปไว้ในรีจีสเตอร์ A

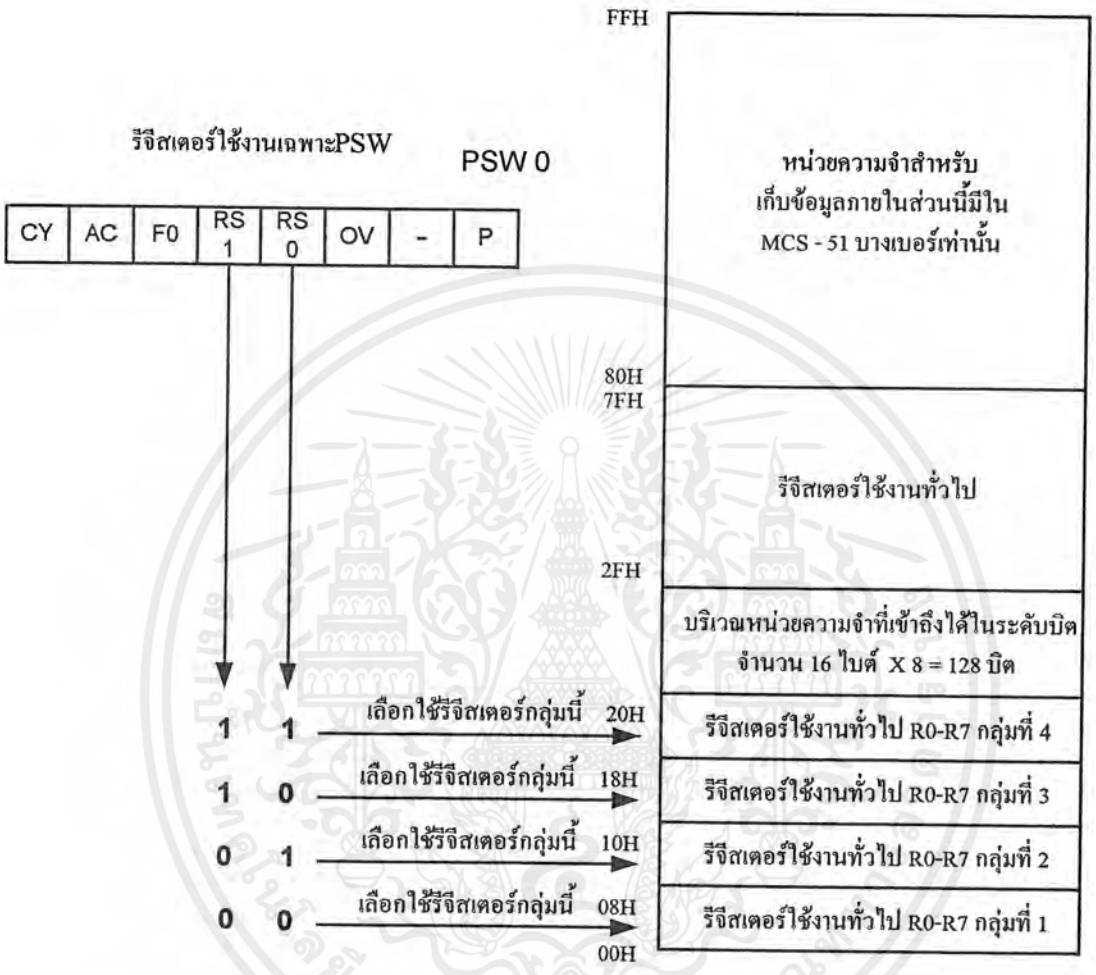
MOVX @DPTR,A นำค่าในรีจีสเตอร์ A ไปไว้ในหน่วยความจำที่ชี้โดยรีจีสเตอร์ DPTR การใช้รีจีสเตอร์ DPTR เป็นตัวชี้ตำแหน่งหน่วยความจำจะต้องโหลดค่าตำแหน่งที่ต้องการไปไว้ในรีจีสเตอร์ DPTR ก่อน

การใช้รีจีสเตอร์ SP เป็นตัวชี้ข้อมูลในหน่วยความจำมีวิธีการที่แตกต่างออกไปผู้เขียนโปรแกรมไม่จำเป็นต้องระบุค่าของรีจีสเตอร์ SP เพราะรีจีสเตอร์ตัวนี้จะถูกเพิ่มหรือลดค่าเองโดยอัตโนมัติ ดังตัวอย่าง

PUSH ACC จะเพิ่มค่าในรีจีสเตอร์ SP จากนั้นจะนำค่าในรีจีสเตอร์ A ไปไว้ในหน่วยความจำที่ชี้โดยรีจีสเตอร์ SP

POP ACC นำค่าในหน่วยความจำที่ชี้โดยรีจีสเตอร์ SP ไปไว้ในรีจีสเตอร์ A จากนั้นรีจีสเตอร์ SP จะลดค่าลงเอง

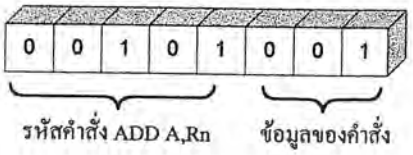
3.8.3 วิธีการเข้าถึงข้อมูลในรีจีสเตอร์ใช้งานทั่วไป วิธีนี้เป็นการเข้าถึงข้อมูลที่อยู่ในรีจีสเตอร์ R0-R7 ของรีจีสเตอร์ใช้งานทั่วไปในแต่ละกลุ่ม รีจีสเตอร์ใช้งานทั่วไปในแต่ละกลุ่มทั้ง 4 กลุ่มคือบริเวณหนึ่งของหน่วยความจำสำหรับเก็บข้อมูลที่ใช้เก็บข้อมูลบริเวณ 128 ไบต์ล่าง 00-1FH หรือตำแหน่งที่ 32 ไบต์ล่างสุดของหน่วยความจำสำหรับข้อมูลที่ใช้เก็บข้อมูลทั่วไป ดังนั้นหากผู้เขียนโปรแกรมต้องการอ่านหรือเขียนข้อมูลในรีจีสเตอร์ทั้ง 32 ตัวซึ่งแต่ละตัวจะมีตำแหน่งในหน่วยความจำที่แน่นอนก็สามารถชี้ตำแหน่งของหน่วยความจำที่ตรงกับรีจีสเตอร์ในแต่ละตัวด้วยวิธีการเข้าถึงข้อมูลโดยตรงหรือทางอ้อมดังที่กล่าวมาแล้วได้ เหตุที่ต้องมีวิธีการเข้าถึงข้อมูลในรีจีสเตอร์ใช้งานทั่วไปนี้ก็เพื่อลดขนาดของคำสั่งให้สั้นลง เพราะหากมีการเข้าถึงข้อมูลของรีจีสเตอร์ R0-R7 จะทราบเองว่าจะต้องเลือกรีจีสเตอร์ใช้งานกลุ่มใดใน 4 กลุ่ม โดยดูจากค่าในบิต RS0, RS1 ดังแสดงในภาพที่ 3.10



ภาพที่ 3.10 แสดงการเลือกใช้รีจิสเตอร์ใช้งานทั่วไปแต่ละกลุ่มจากบิต RS0, RS1

ดังนั้นคำสั่งที่มีการเข้าถึงข้อมูลใช้รีจิสเตอร์ใช้งานทั่วไปจึงต้องการเลขไบนารี่เพียง 3 บิต เป็นตัวบอกรีจิสเตอร์ที่ต้องการใช้เท่านั้น ดังนั้นคำสั่งที่มีการใช้รีจิสเตอร์ R0-R7 โดยวิธีนี้จะมีความยาวของคำสั่งต่ำที่สุด 1 ไบต์ (รหัสคำสั่ง 5 บิต รวมกับอีก 3 บิต ที่ต้องระบุรีจิสเตอร์) ดังแสดงในตัวอย่างต่อไปนี้

ADD A,R1 นำค่าในรีจิสเตอร์ทั่วไป R1 มาบวกกับค่าในรีจิสเตอร์ A ผลลัพธ์เก็บไว้ในรีจิสเตอร์ A



ระบุรีจีสเตอร์ที่ใช้ในคำสั่งดังนี้

0 0 0 R0	1 0 0 R4
0 0 1 R1	1 0 1 R5
0 1 0 R2	1 1 0 R6
0 1 1 R3	1 1 1 R7

3.8.4 วิธีการเข้าถึงข้อมูลในรีจีสเตอร์เฉพาะของคำสั่ง คำสั่งบางคำสั่งของ 8051 จะระบุไว้แล้วว่าต้องปฏิบัติการกับข้อมูลในรีจีสเตอร์ตัวใด เช่น รีจีสเตอร์ A รีจีสเตอร์ DPTR รีจีสเตอร์ SP ในรหัสคำสั่งของคำสั่งที่ใช้วิธีการเข้าถึงข้อมูลประเภทนี้ 8051 จะทราบเองว่าต้องทำงานกับรีจีสเตอร์ตัวใดโดยไม่จำเป็นต้องระบุตำแหน่งของรีจีสเตอร์ที่ใช้โดยคำสั่งเลย เช่น

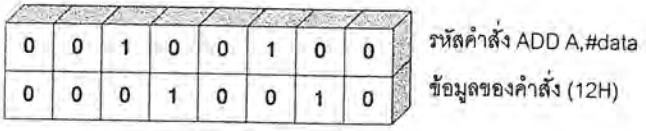
INC DPTR มีขนาด 1 ไบต์ เพราะไม่ต้องระบุตำแหน่งรีจีสเตอร์ DPTR

MUL AB มีขนาด 1 ไบต์ เพราะไม่ต้องระบุตำแหน่งรีจีสเตอร์ A, B

จากคำสั่งทั้งสองนี้จะเห็นได้ว่าโอเปอร์แรนด์ตัวแรกของคำสั่งเป็นรีจีสเตอร์ DPTR , A, B ตามลำดับ แต่ผู้ใช้ไม่จำเป็นต้องระบุตำแหน่งของรีจีสเตอร์ทั้งสอง เพราะ 8051 จะทราบเองจากรหัสคำสั่ง

3.8.5 วิธีการเข้าถึงข้อมูลที่กำหนดเองโดยตรง เป็นการกำหนดค่าข้อมูลที่จะนำไปประมวลผลโดยตรง ข้อมูลที่นำมาประมวลผลในคำสั่งจะอยู่ตามหลังรหัสคำสั่งโดยการใช้เครื่องหมาย " # " ระบุหน้าข้อมูลที่ต้องการ เช่น

ADD A,#12H นำค่า 12H มาบวกกับค่าในรีจีสเตอร์ A ผลลัพธ์เก็บไว้ในรีจีสเตอร์ A



3.8.6 วิธีการเข้าถึงข้อมูลโดยใช้ตัวอ้างอิง ข้อมูลที่ใช้วิธีการอ้างแบบนี้จะเป็นข้อมูลที่อยู่ในหน่วยความจำสำหรับเก็บโปรแกรมเท่านั้น นั่นแสดงว่าเราสามารถอ่านข้อมูลนี้ออกมาได้แต่ไม่สามารถนำข้อมูลไปเก็บโดยใช้วิธีนี้ได้ จุดประสงค์ของการอ้างข้อมูลแบบนี้มีไว้เพื่อใช้ในการเปิดหาค่าข้อมูลในหน่วยความจำสำหรับเก็บโปรแกรม ซึ่งเป็นหน่วยความจำชนิดถาวร (ROM) ข้อมูลในส่วนนี้จะไม่สูญหายแม้ไม่มีพลังงาน ในการทำงานของคำสั่งที่ใช้การเข้าถึงข้อมูลวิธีนี้จะใช้ค่าของรีจีสเตอร์ใช้งานเฉพาะ DPTR หรือ PC มารวมกับค่าในรีจีสเตอร์ A เพื่อชี้ไปยังตำแหน่งของหน่วยความจำสำหรับเก็บโปรแกรมที่เก็บข้อมูลไว้ ดังนั้นค่าในรีจีสเตอร์ใช้งานเฉพาะ DPTR,PC จะต้องมีค่าเท่ากับตำแหน่งต้นของหน่วยความจำสำหรับเก็บโปรแกรมในส่วนที่เก็บข้อมูล โดยใช้ค่าของรีจีสเตอร์ A เป็นตัวระบุว่าข้อมูลอยู่ห่างจากตำแหน่งเริ่มต้นในรีจีสเตอร์ใช้งานเฉพาะ DPTR หรือ PC เท่าใด จุดประสงค์ของวิธีการอ้างข้อมูลแบบนี้คือ ใช้ในการเปิดหาค่าข้อมูลในตารางซึ่งอยู่ในหน่วยความจำสำหรับเก็บโปรแกรมซึ่งเรียงต่อกันไป ตัวอย่างการใช้งานของวิธีการเข้าถึงข้อมูลชนิดนี้มีดังนี้คือ

```
MOVC A,@A+DPTR
MOVC A,@A+PC
```

การเข้าถึงข้อมูลวิธีนี้ยังมีที่ใช้ในกลุ่มคำสั่งควบคุมลำดับการทำงานของโปรแกรมโดยการบังคับให้โปรแกรมกระโดดไปทำงานที่ตำแหน่งใด ๆ ในความจำซึ่งมีค่าตำแหน่งเท่ากับผลรวมของค่าใช้รีจีสเตอร์ DPTR และ A ดังตัวอย่าง

```
JMP @A,+DPTR
```

3.9 ไทเมอร์ / เคาน์เตอร์

8051 จะมีตัวจับเวลา/ตัว (Timer/C0unter) นับขนาด 16 บิต จำนวน 2 ตัว คือไทเมอร์ / เคาน์เตอร์ 0 และ ไทเมอร์/เคาน์เตอร์ 1 โดยแต่ละตัวสามารถกำหนดให้เป็นตัวจับเวลาหรือตัวนับได้ โดยการเซตหรือเคลียร์บิต C/T ที่ตัวรีจีสเตอร์ควบคุม TMOD ซึ่งอยู่ในกลุ่มรีจีสเตอร์ฟังชันพิเศษ

ในการกำหนดให้ทำงานเป็นตัวจับเวลา ตัวรีจีสเตอร์ TH1 และ TH2 ซึ่งทำหน้าที่เป็นตัวเก็บค่าจำนวนพัลส์ที่เข้ามาจะเพิ่มค่าทุก ๆ แมชีนไซเกิล โดยแต่ละแมชีนไซเกิลจะประกอบด้วย 12 คาบออสซิลเลเตอร์ ดังนั้นอัตราการนับแต่ละครั้งจะใช้เวลาเท่ากับ 1/12 ของความถี่ออสซิลเลเตอร์ ซึ่งส่วนใหญ่จะใช้งานเป็นอินเตอร์รัพต์RTC (Real Time Clock)

และถ้าให้ทำงานเป็นตัวนับ รีจีสเตอร์ตัวนับจะเพิ่มค่าทุกครั้งที่มีการเปลี่ยนแปลงสถานะจาก "1" เป็น "0" ที่ขา T0 หรือ T1 โดยอัตราความถี่สูงสุดที่สามารถนับได้ต้องไม่เกิน 1/24 ของความถี่ออสซิลเลเตอร์

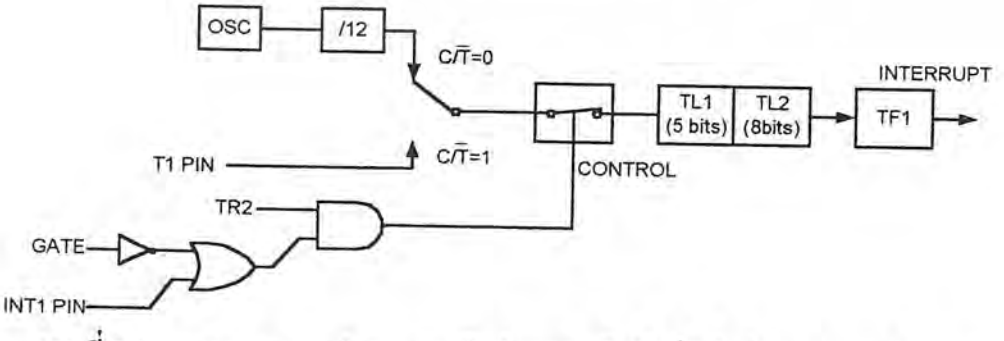
ตัวจับเวลา / ตัวนับสามารถโปรแกรมให้มันทำงานได้ต่างกันถึง 4 โหมด โดยการตั้งค่าในรีจีสเตอร์ TMOD ซึ่งสามารถกล่าวถึงการทำงานในแต่ละโหมดได้ดังนี้

โหมด 0 รีจีสเตอร์ตัวนับจะถูกกำหนดให้มี 13 บิต ประกอบด้วยรีจีสเตอร์ TH1 8 บิต และ TL1 อีก 5 บิตอันดับต่ำ ซึ่งสามารถกำหนดให้เป็นตัวจับเวลาหรือตัวนับได้โดยเซตหรือเคลียร์ที่บิต $C/\overline{T}$ ในตัวรีจีสเตอร์ TMOD การทำงานของรีจีสเตอร์จะนับขึ้นครั้งละ 1 เมื่อมีสัญญาณเข้ามา 1 ลูก และเมื่อนับจนเป็น "1" หมดทุกบิต ก็จะกลับมาเป็น "0" หมดทุกบิตใหม่ ซึ่งจะเป็นการเกิดโอเวอร์โฟลว์ (Overflow) ไปทดแฟลกอินเตอร์รัพต์ ให้เป็น "1"

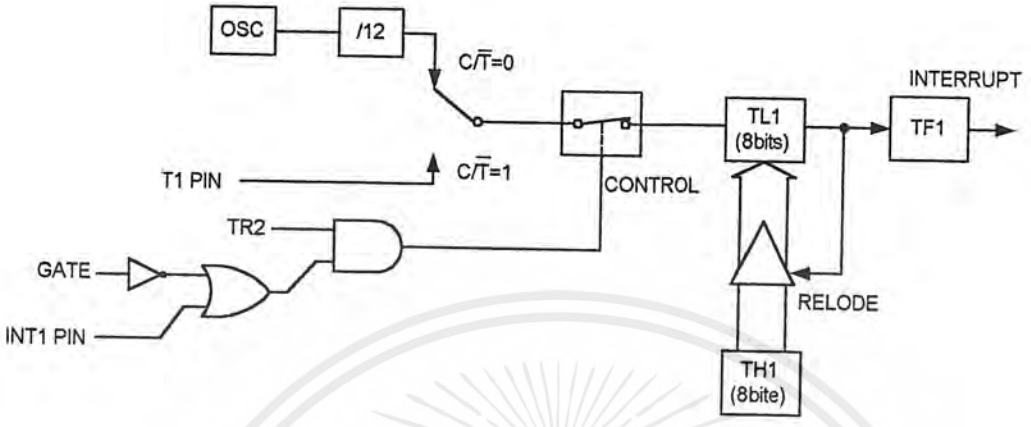
โหมด 1 การทำงานเหมือนกับโหมด 0 ทุกอย่างยกเว้นรีจีสเตอร์ตัวนับจะเป็นขนาด 16 บิต

โหมด 2 จะใช้รีจีสเตอร์ TL1 เป็นตัวนับเพียงตัวเดียวและเมื่อ TL1 นับจนเป็น "1" หมดทุกบิต ก็จะมีการโหลดค่าจากรีจีสเตอร์ TH1 เข้าไปไว้ใน TL1 โดยอัตโนมัติ และทำการทดลองแฟลกอินเตอร์รัพต์ TF1 ให้เป็น "1" ค่าใน TH1 นี้สามารถตั้งค่าด้วยซอฟแวร์

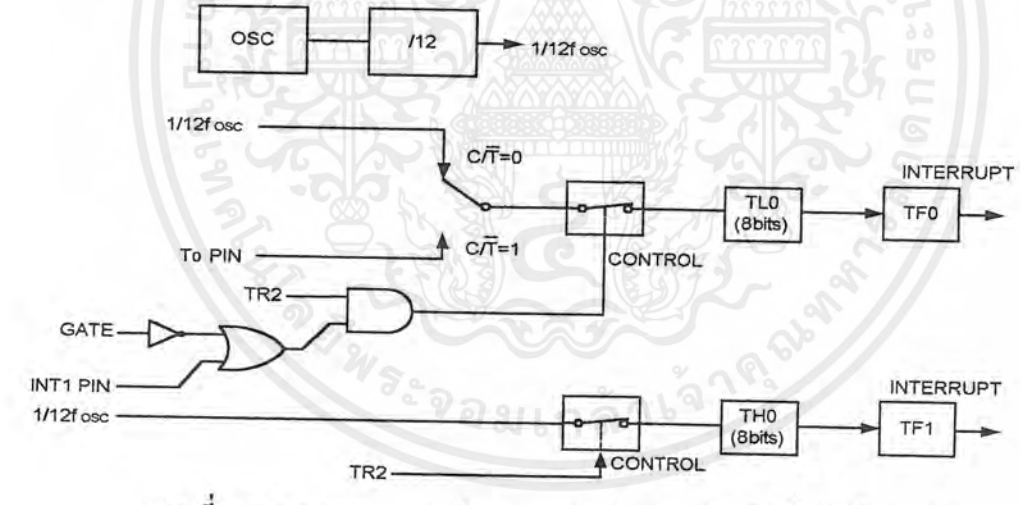
โหมด 3 เป็นการเพิ่มตัวจับเวลาขึ้นอีก 1 ตัว แต่จะเป็นขนาด 8 บิตทั้งคู่ ซึ่งลักษณะการทำงานอื่น ๆ จะเหมือนกับโหมด 0



ภาพที่ 3.11 แสดงวงจรการทำงานของตัวจับเวลา / ตัวนับที่ 1 ในโหมด 0 และโหมด 1



ภาพที่ 3.12 แสดงวงจรการทำงานของตัวจับเวลา / ตัวนับที่ 1 ในโหมด 2



ภาพที่ 3.13 แสดงวงจรการทำงานของตัวจับเวลา / ตัวนับที่ 0 ในโหมด 3

สำหรับรายละเอียดบิตควบคุมในรีจีสเตอร์ TMOD และรีจีสเตอร์ TCON แสดงไว้ดังภาพที่ 3.14 และ 3.15

TIMER/COUNTER 1				TIMER/COUNTER 0			
GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0

ภาพที่ 3.14 แสดงบิตควบคุมที่อยู่ในรีจีสเตอร์ TMOD

GATE : เซตเป็น "1" จะเป็นการอีนาเบิลตัวจับเวลา/ตัวนับให้ถูกควบคุมด้วยขา INTx ต้อง มี สถานะสูงและบิต TRx ใน TCON จะต้องเป็น "1" จึงจะเริ่มทำงาน แต่ถ้า GATE เป็น "0" ตัวจับเวลาตัวนับเวลาจะถูกควบคุมให้เริ่มทำงานด้วยบิต TRX เท่านั้น

C/T : เป็นบิตควบคุมการเลือกทำงานเป็นตัวจับเวลาหรือตัวนับ ถ้าเป็น "0"จะทำงานเป็นตัวจับเวลา ถ้าเป็น "1" ทำงานเป็นตัวนับ

M1, M0 : เป็นตัวเลือกโหมดการทำงานดังนี้

M1	M0	โหมดการทำงาน
0	0	โหมด 0
0	1	โหมด 1
1	0	โหมด 2
1	1	โหมด 3

ตารางที่ 3.1 แสดงการเลือกโหมดการทำงานของ TMER/COUNTER โดย บิต M1,M2

TF1	TR1	TF0	IE1	IT1	IE0	IT0

ภาพที่ 3.15 แสดงบิตควบคุมที่อยู่ในรีจีสเตอร์ TCON

TFx เป็นแฟลกอินเตอร์จะถูกเซตด้วยฮาร์ดแวร์เมื่อเกิดโอเวอร์โฟลว์ของตัวจับเวลา/ตัวนับ และจะเคลียร์ตัวเองโดยอัตโนมัติ เมื่อทำงานอินเตอร์รัพต์นั้นเสร็จเรียบร้อยแล้ว

TRx :เป็นบิตควบคุมให้ตัวจับเวลา/ตัวนับเวลาเริ่มทำงานโดยการเซตให้เป็น "1" และให้หยุดทำงานด้วยการเคลียร์ให้เป็น "0" โดยซอฟแวร์ล

IEx :เป็นแฟลกอินเตอร์รัพต์จากสัญญาณภายนอก เซตด้วยฮาร์ดแวร์เมื่อมีสัญญาณขอการอินเตอร์รัพต์ปรากฏที่ขา INTx และจะเคลียร์ตัวเองโดยอัตโนมัติ เมื่อกระโดดไปทำงานบริการอินเตอร์รัพต์ที่ขอเข้ามาเรียบร้อยแล้ว

Itx :เป็นบิตควบคุมรูปแบบสัญญาณอินเตอร์รัพต์ภายนอกจะเซต/เคลียร์ด้วยซอฟแวร์ โดยถ้าเซตเป็น "1" จะถูกอินเตอร์รัพต์ด้วยสัญญาณขอบขาลง และถ้าเคลียร์เป็น "0" จะถูกอินเตอร์รัพต์ด้วยสัญญาณระดับแรงดันต่ำ

3.10 โครงสร้างการอินเตอร์รัพต์

8051 รับสัญญาณอินเตอร์รัพต์จากอุปกรณ์อื่น ๆ ได้ 5 แหล่ง เราสามารถกำหนด ลำดับความสำคัญของอินเตอร์รัพต์ได้ 2 ระดับ โดยไม่ต้องอาศัยวงจรภายนอกช่วย แต่ละแหล่งของอินเตอร์รัพต์นั้นจะกำหนดเป็นเวกเตอร์เฉพาะ (ตัวชี้แอดเดรส) ดังนั้นเมื่อมีการอินเตอร์รัพต์เข้ามาแล้ว ตัวโปรเซสเซอร์จะกระโดดไปที่ส่วนของโปรแกรมที่ต้องทำงานตามวัตถุประสงค์ของอินเตอร์รัพต์นั้น หลังจากเก็บข้อมูลต่าง ๆ ของโปรแกรมเคาน์เตอร์ลงในสแต็ก

การจัดการและการควบคุมอินเตอร์รัพต์มีรีจิสเตอร์พิเศษ 2 ตัว สำหรับการกำหนดรูปแบบการทำงานต่าง ๆ ซึ่งได้แก่ รีจิสเตอร์พิเศษ IE (interrupt enable ,ที่อยู่แอดเดรส 0AH) และ IP (interrupt priority, ที่อยู่แอดเดรส 0B8H) การกำหนดรูปแบบอินเตอร์รัพต์ ทำได้โดยการเซตหรือรีเซตตำแหน่งบิตใน IE อย่างเหมาะสมดังแสดงความหมายของแต่ละบิตใน IE ในรูปที่ 2. สำหรับในรูปที่ 2. แสดงความหมายของแต่ละบิตในรีจิสเตอร์ TCON

EA	X	ET2	ES	ET1	EX1	ET0	EX0

ภาพที่ 3.16 หน้าที่การทำงานของแต่ละบิตในรีจิสเตอร์ IE

สัญลักษณ์	ตำแหน่ง	หน้าที่
EA	IE.7	ทำการดิสเอเบิลทุกอินเตอร์รัพต์ ถ้า EA = "0" จะไม่มีการตอบรับอินเตอร์รัพต์ใด ๆ ทั้งหมด ถ้า EA = "1" แต่ละอินเตอร์รัพต์จะถูกอีนาเบิลหรือดีสเอเบิลโดยการเซต หรือเคลียร์อีนาเบิลบิต
	IE.6	สงวนไว้ไม่ใช้งาน
ET2	IE.5	ทำการอีนาเบิลหรือดีสเอเบิลอินเตอร์รัพต์จากโอเวอร์โฟลว์ของไทเมอร์ 2 ถ้า ET2 =0 อินเตอร์รัพต์จากไทเมอร์ 2 จะถูกดีสเอเบิล
ES	IE.4	ทำการอีนาเบิลหรือดีสเอเบิลอินเตอร์รัพต์จากพอร์ตอนุกรมถ้า ES = "0" อินเตอร์รัพต์นี้จะถูกดีสเอเบิล
ET1	IE.3	ทำการอีนาเบิลหรือดีสเอเบิลอินเตอร์รัพต์จากโอเวอร์โฟลว์ของไทเมอร์1 ถ้า ET1 = 0อินเตอร์รัพต์จากไทเมอร์ 2 จะถูกดีสเอเบิล
EX1	IE.2	ทำการอีนาเบิลหรือดีสเอเบิลอินเตอร์รัพต์จากภายนอก 1 ถ้า EX1 = "0" อินตอร์รัพต์นี้จะถูกดีสเอเบิล
ET0	IE.1	ทำการอีนาเบิลหรือดีสเอเบิลอินเตอร์รัพต์จากโอเวอร์โฟลว์ของไทเมอร์ 0 ถ้าET0 = 0อินเตอร์รัพต์จากไทเมอร์ 2 จะถูกดีสเอเบิล
EX0	IE.0	ทำการอีนาเบิลหรือดีสเอเบิลอินเตอร์รัพต์จากภายนอก 0 ถ้า EX0 = "0"อินเตอร์รัพต์นี้จะถูกดีสเอเบิล

TF2	TR1	TF0	TR0	IE1	IT1	IE0	IT0

ภาพที่ 3.17 หน้าที่การงานของแต่ละบิตในรีจีสเตอร์ TCON

สัญลักษณ์	ตำแหน่ง	หน้าที่
TF1	TCON.7	เป็นโอเวอร์โฟลว์แฟลกของไทเมอร์ 1 จะถูกเซตโดยฮาร์ดแวร์เมื่อไทเมอร์/เคาท์เตอร์นี้เกิดโอเวอร์โฟลว์จะถูกเคลียร์โดยฮาร์ดแวร์เมื่อโปรเซสเซอร์เข้าสู่การอินเตอร์รัพต์รูทีน
TR1	TCON.6	บิตควบคุมการรันของไทเมอร์ 1 จะถูกเซตและเคลียร์โดยซอฟต์แวร์เพื่อให้ไทเมอร์/เคาน์นี้ทำงานหรือหยุดทำงาน
TF0	TCON.5	เป็นโอเวอร์โฟลว์แฟลกของไทเมอร์ 0 จะถูกเซตโดยฮาร์ดแวร์เมื่อไทเมอร์/เคาท์เตอร์นี้เกิดโอเวอร์โฟลว์จะถูกเคลียร์โดย

		ฮาร์ดแวร์เมื่อโปรเซสเซอร์เข้าสู่การอินเตอร์รัพต์รูทีน
TR0	TCON.4	บิตควบคุมการรันของไทเมอร์ 0 จะถูกเซตและเคลียร์โดยซอฟต์แวร์เพื่อให้ไทเมอร์/เคาน์นี้ทำงานหรือหยุดทำงาน
IE1	TCON.3	แฟลกแสดงการอินเตอร์รัพต์ที่ 1 ทำงานที่ขอบขา จะถูกเซตโดยฮาร์ดแวร์เมื่อตรวจพบสัญญาณอินเตอร์รัพต์จากภายนอกที่ขอบขา และจะถูกเคลียร์เมื่อเข้าสู่กระบวนการอินเตอร์รัพต์
IT1	TCON.2	บิตกำหนดรูปแบบการอินเตอร์รัพต์จากภายนอก 1 จะถูกเซตหรือเคลียร์โดยซอฟต์แวร์ เพื่อกำหนดให้เกิดอินเตอร์รัพต์ที่ขอบขาลงหรือที่ระดับโลว์ของสัญญาณอินเตอร์รัพต์ภายนอก
IE0	TCON.1	แฟลกแสดงการอินเตอร์รัพต์ที่ 0 ทำงานที่ขอบขา จะถูกเซตโดยฮาร์ดแวร์เมื่อตรวจพบสัญญาณอินเตอร์รัพต์จากภายนอกที่ขอบขา และจะถูกเคลียร์เมื่อเข้าสู่กระบวนการอินเตอร์รัพต์
IT0	TCON.0	บิตกำหนดรูปแบบการอินเตอร์รัพต์จากภายนอก 0 จะถูกเซตหรือเคลียร์โดยซอฟต์แวร์ เพื่อกำหนดให้เกิดอินเตอร์รัพต์ที่ขอบขาลงหรือที่ระดับโลว์ของสัญญาณอินเตอร์รัพต์ภายนอก

X	XP	PT2	PS	PT1	PXI	PT0	PX0

ภาพที่ 3.18 หน้าที่การทำงานของแต่ละบิตในรีจีสเตอร์ IP

สัญลักษณ์	ตำแหน่ง	หน้าที่
-	IP.7	สงวนไว้ไม่ใช้งาน
-	IP.6	สงวนไว้ไม่ใช้งาน
PT2	IP.5	บิตกำหนดระดับความสำคัญของอินเตอร์รัพต์ไทเมอร์ 2 ถ้า PT2 = "1" อินเตอร์รัพต์จะมีความสำคัญระดับสูง
PS	IP.4	บิตกำหนดระดับความสำคัญของอินเตอร์รัพต์ของพอร์ตอนุกรม ถ้า PS = "1" อินเตอร์รัพต์จะมีความสำคัญระดับสูง
PT1	IP.3	บิตกำหนดระดับความสำคัญของอินเตอร์รัพต์ไทเมอร์ 1 ถ้า PT1

= "1" อินเตอร์รัพต์จะมีความสำคัญระดับสูง

PX1	IP.2	บิตกำหนดระดับความสำคัญของอินเตอร์รัพต์จากภายนอก 1 ถ้า PX1 = "1" อินเตอร์รัพต์จะมีความสำคัญระดับสูง
PT0	IP.1	บิตกำหนดระดับความสำคัญของอินเตอร์รัพต์ไทเมอร์ 0 ถ้า PT0 = "1" อินเตอร์รัพต์จะมีความสำคัญระดับสูง
PX0	IP.0	บิตกำหนดระดับความสำคัญของอินเตอร์รัพต์จากภายนอก 0 ถ้า PX0 = "1" อินเตอร์รัพต์จะมีความสำคัญระดับสูง

ชนิดของอินเตอร์รัพต์	ลำดับความสำคัญ	ตำแหน่งเริ่มต้นของตำแหน่งบริการอินเตอร์รัพต์
อินเตอร์รัพต์ภายนอกชนิด 0 (IE0)	1	0003H
อินเตอร์รัพต์ของไทเมอร์ 0 (TF0)	2	000BH
อินเตอร์รัพต์ภายนอกชนิด 1 (IE1)	3	0013H
อินเตอร์รัพต์ของไทเมอร์ 1 (TF1)	4	001BH
อินเตอร์รัพต์ของสื่อสารอนุกรม (TI+RI)	5	0023H

ตารางที่ 3.2 แสดงตำแหน่งเริ่มต้นของโปรแกรมบริการอินเตอร์รัพต์แต่ละชนิดที่เกิดขึ้น

3.11 พอร์ตของ 8051

8051 เป็นไมโครคอลโทรลเลอร์ขนาด 40 ขา ซึ่งมีขาต่าง ๆ ดังนี้

- Vcc (ขา 40) ต่อกับ + 5 V
- Vss (ขา 20) เป็นขา GND
- พอร์ต 0 (ขา 32-39) มีทั้งหมด 8 บิต คือ (P0.7 – P0.0) มีโครงสร้างแบบ Open Drain Bi-Directional

พอร์ต 0 (ขา 32-39) มีทั้งหมด 8 บิต คือ (P0.7-P0.0) ใช้งานได้ 2 หน้าที่ คือเอดเดรสบัสและดาต้าบัสเมื่อต้องการติดต่อกับหน่วยความจำภายนอกหรือเป็นไอโอพอร์ต ถ้าต้องการให้ทำงานเป็นอินพุตพอร์ตจะต้องส่งลอจิก 1 ไปยังพอร์ตนี้ จะมีผลให้ Q ของ D-FF เป็น 0 ทำให้ FET ตัวล่างมี

สถานะ OFF สัญญาณที่ใช้อ่านอินพุตพอร์ทแลทช์โดยส่งสัญญาณ READ LATCH ไปกระตุ้นที่ Tri-State Buffer ตัวบนและการอ่าน Port (pin) จะใช้สัญญาณ Read (pin)

พอร์ต 1 (ขา 1-8) มีทั้งหมด 8 บิต คือ (P1.0-P1.7) มีโครงสร้างคล้ายพอร์ต 0 แต่จะใช้ความต้านทานภายในพูลอัพแทน Internal Pull up Register มีโครงสร้างดังรูป 2.4

พอร์ต 2 (ขา 21-28) มีทั้งหมด 8 บิต คือ (P2.7-2.0) มีโครงสร้างคล้ายพอร์ต 0 โดยมี FET ตัวล่างตัวเดียวส่วนตัวบนใช้ความต้านทานพูลอัพแทน (Internal Pull up) พอร์ตนี้ทำงาน 2 หน้าที่ คือ สามารถใช้เป็นเอดเดรสบัสขนาด 8 บิต (A15-A8) และเป็นไอโอพอร์ตที่ใช้งานทั่วไปเมื่อจะใช้งานเป็นอินพุตพอร์ตต้องส่งลอจิก 1 มาที่พอร์ตนี้ก่อนเพื่อบังคับให้ FET อยู่ในสภาวะ off ดังแสดงในรูป 2.5

พอร์ต 3 (ขา 10-17) มีทั้งหมด 8 บิต คือ ขา(P3.7-P3.0) มีโครงสร้างคล้ายพอร์ต 1 ทำงานได้ 2 หน้าที่คือเป็นไอโอพอร์ตถ้าจะโปรแกรมให้เป็นอินพุตพอร์ตต้องส่งลอจิก 1 มาที่พอร์ทนี้ก่อน และอีกหน้าที่หนึ่งคือใช้ส่งสัญญาณควบคุมออกมา และรับสัญญาณเข้าไปสัญญาณต่าง ๆ มีดังนี้

P3.0/RXD (Serial Input Port) เป็นขาที่ใช้รับข้อมูลแบบอนุกรม (UART)

P3.1/TXD (Serial Output Port) เป็นขาที่ใช้ส่งข้อมูลแบบอนุกรม (UART)

P3.2/INT0 (External Interupt 0) ใช้รับสัญญาณการขัดจังหวะจากภายนอกเบอร์ 0

P3.3/INT1(External Interupt 1) ใช้รับสัญญาณการขัดจังหวะจากภายนอกเบอร์ 1

P3.4/T0 (Counter 0 External Input) ขารับสัญญาณพัลซ์อินพุตเข้าไปยังวงจร Counter 0 (เป็นอินพุตโหมดเคาน์เตอร์)

P3.5/T1 (Counter 1 External Input) ขารับสัญญาณพัลซ์อินพุตเข้าไปยังวงจร Counter 1 (เป็นอินพุตโหมดเคาน์เตอร์)

P3.6/$\overline{WR}$ (External Data Memory Write Strope) ขาสัญญาณควบคุมการเขียนข้อมูลลงหน่วยความจำข้อมูลภายนอก

P3.7/$\overline{RD}$ (External Data Memory Read Strope) ขาสัญญาณควบคุมการอ่านข้อมูลลงหน่วยความจำข้อมูลภายนอก

บทที่ 4
ชุดคำสั่งของของ MCS-51

4.1 กลุ่มคำสั่งทางคณิตศาสตร์

กลุ่มคำสั่งกลุ่มแรกที่จะศึกษาในกลุ่มคำสั่งทั้งหมดของ MCS-51 คือ กลุ่มคำสั่งทางคณิตศาสตร์ กลุ่มคำสั่งทางคณิตศาสตร์ในMCS–51ประกอบด้วยกลุ่มคำสั่งที่เกี่ยวข้องกับการกระทำทางคณิตศาสตร์ ดังต่อไปนี้

- กลุ่มคำสั่งเพิ่มหรือลดค่าข้อมูลครั้งละหนึ่ง (Incrementing and Decrementing Instructions)
- กลุ่มคำสั่งบวกเลขจำนวนเต็ม (Addition Instructions)
- กลุ่มคำสั่งลบเลขจำนวนเต็ม (Subtraction Instructions)
- กลุ่มคำสั่งคูณเลขจำนวนเต็ม (Multiple Instructions)
- กลุ่มคำสั่งหารเลขจำนวนเต็ม (Divide Instructions)
- กลุ่มคำสั่งปรับค่าผลลัพธ์จากการบวกเลขรหัส BCD (Decimal Adjust Instructions)

4.1.1 กลุ่มคำสั่งเพิ่มหรือลดค่าข้อมูลครั้งละหนึ่ง

คำสั่งนี้เป็นคำสั่งพื้นฐานที่สุดของกลุ่มคำสั่งทางคณิตศาสตร์จากในตารางที่ 1 จะเห็นว่าคำสั่งนี้มีวิธีการเข้าถึงข้อมูลได้หลายวิธี ดังนั้นคำสั่งทั้งหมดในกลุ่มนี้ มีดังนี้

INCA	DEC A
INC Rn	DEC Rn
INC direct	DEC direct
INC @ Ri	DEC @ Ri
INC DPTR	ไม่มีคำสั่ง DEC DPTR

ตารางที่ 4.1 แสดงกลุ่มคำสั่งเพิ่มหรือลดค่าข้อมูลครั้งละหนึ่ง

หมายเหตุ จะสังเกตได้ว่าใน MCS – 51 ไม่มีคำสั่ง DEC DPTR ที่จะมาจับคู่คำสั่ง INC DPTR

คำสั่ง INC และ DEC จะใช้ในกรณีที่ต้องการเพิ่มหรือลดค่าข้อมูลในหน่วยความจำสำหรับเก็บข้อมูลทั่วไปภายในชิบหรือที่ใช้เป็นรีจิสเตอร์ ใช้งานเฉพาะครั้งละ 1 โดยไม่จำเป็นต้องใช้คำสั่งบวก หรือลบ ซึ่งต้องอาศัยรีจิสเตอร์ A เป็นการเฉพาะหรืออาจจะใช้คำสั่งกลุ่มนี้ในส่วนของโปรแกรมที่ต้องมีการวนรอบการทำงาน (loop) เป็นจำนวนครั้งตามที่ต้องการ โดยการเพิ่มหรือลดค่าข้อมูลที่ใช้เป็นตัวนับจำนวนรอบไปเรื่อย ๆ จนกระทั่งได้จำนวนรอบที่ต้องการ กลุ่มคำสั่งทางคณิตศาสตร์กลุ่มแรกที่กล่าวถึงนี้ไม่มีผลกระทบต่อเฟล็กใน MCS – 51 ใด ๆ ทั้งสิ้นยกเว้นหากข้อมูลถูกเพิ่มหรือลดค่ารีจิสเตอร์ PSW เอง

4.1.2 กลุ่มคำสั่งบวกเลขจำนวนเต็ม

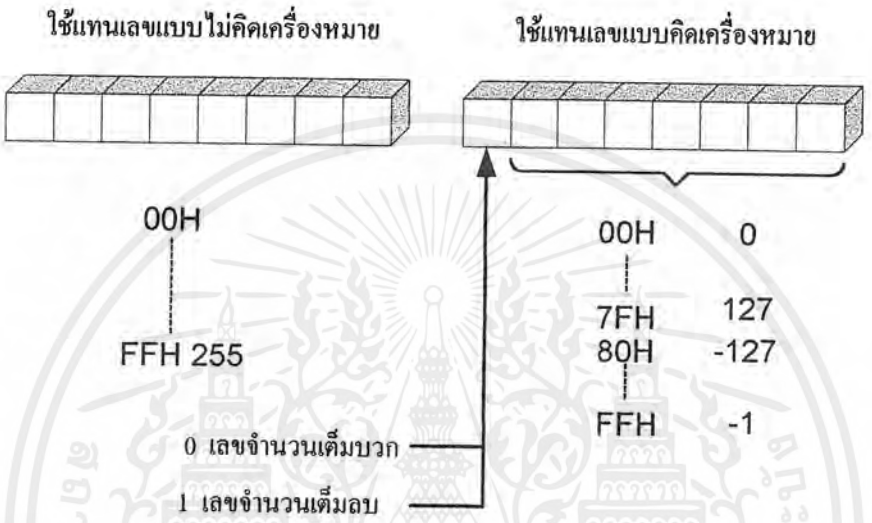
คำสั่งในกลุ่มนี้มีวิธีการในการเข้าถึงข้อมูลได้หลายวิธี ดังนั้นคำสั่งทั้งหมดในกลุ่มนี้จะประกอบไปด้วยคำสั่งย่อย ๆ ดังนี้

ADD A, Rn	ADDC A,Rn
ADD A, direct	ADDC A ,direct
ADD A, @Ri	ADDC A, @Ri
ADD A, # data	ADDC A,# data

หมายเหตุ สังเกตุว่าคำสั่งในการบวกนี้ ผลลัพธ์จะนำไปเก็บไว้ในรีจิสเตอร์ A เสมอ

เนื่องจากซีพียู MCS – 51 มีขนาด 8 บิต ดังนั้นการปฏิบัติคำสั่งบวกหรือลบ จะกระทำครั้งละ 8 บิตเสมอ และจากทฤษฎีหลักการนับ เราทราบว่าเลขไบนารีขนาด 8 หลักสามารถแทนจำนวนเต็มที่มีค่าแตกต่างกันได้ทั้งสิ้น 256 จำนวน ($2^8 = 256$) จำนวนนี้อาจเป็นเลขจำนวนเต็มบวกเพียงอย่างเดียว ก็จะมีค่าได้ตั้งแต่ 0 ถึง 255 (00 – OFFH) แต่หากต้องการแทนเลขไบนารีขนาด 8 หลัก ให้เป็นทั้งเลขจำนวนเต็มบวกและเลขจำนวนเต็มลบด้วย จำเป็นต้องใช้เทคนิคพิเศษเพื่อแยกความแตกต่างระหว่างเลขจำนวนเต็มบวกและเลขจำนวนเต็มลบ วิธีการทั่วไปที่ใช้แยกความแตกต่างระหว่างเลขจำนวนเต็มบวก และเลขจำนวนเต็มลบ คือการใช้บิตสูงสุด (MSB – Significant Bit) ของเลขไบนารีขนาด 8 หลัก เป็นการบอกเครื่องหมายของเลขจำนวนเต็มที่ต้องการแทนโดยมีเงื่อนไขว่าหากบิตนี้มีค่า เป็น 0 หมายความว่าเลขนั้นเป็นเลขจำนวนเต็มบวก หากบิตนี้มีค่าเป็น 1 หมายถึงเลขจำนวนเต็มลบ และเมื่อเราใช้บิตสูงสุดแทนเครื่องหมายแล้วจะเหลือเลขจำนวนเต็มลบ และเมื่อเราใช้บิตสูงแทนเครื่องหมายแล้วจะเหลือเลขไบนารีสำหรับใช้แทนเลขจำนวนเต็มฐาน 10 เพียง 7 หลัก ดังนั้นเลขจำนวนเต็มบวกที่สามารถแทนได้จะมีค่าอยู่ระหว่าง 0 – 127 (OO – 7FH) รวม 128 จำนวน ส่วนเลขจำนวนเต็มลบ จะมีได้ตั้งแต่ -1 ถึง -

128 (OFFH – 80H) รวม 128 จำนวนเช่นกัน เลขจำนวนลบนี้จะแทนอยู่ในรูป 2's complement ดังแสดงในภาพที่ 4.1



ภาพที่ 4.1 การใช้เลขไบนารี่ขนาด 8 บิตแทนจำนวนเต็มบวกอย่างเดียวและเลขจำนวนเต็มบวกและเลขจำนวนเต็มลบรวมกัน

เมื่อเราใช้เลขไบนารีขนาด 8 หลัก สำหรับใช้แทนจำนวนเต็มค่าต่าง ๆ ได้ทั้งสิ้น 256 ค่าซึ่งจะเป็นเลขที่มีเครื่องหมายหรือไม่นั้นขึ้นอยู่กับผู้เขียนโปรแกรมจะกำหนดไว้ ในการบวกเลขทั้งสองชนิด (เลขที่คิดและไม่คิดเครื่องหมาย) มักเกิดปัญหาเมื่อผลลัพธ์ที่ได้มีค่าเกินกว่าที่เลขไบนารี 8 หลักจะสามาถแทนได้ กรณีของปัญหาที่อาจเกิดขึ้นสามารถแยกกล่าวทีละหัวข้อ ได้ดังนี้

กรณีที่บวกเลขที่ไม่คิดเครื่องหมายบวก นั่นคือใช้เลขไบนารี ทั้ง 8 หลักแทนขนาดของจำนวน ซึ่งจำนวนสูงสุดที่สามารถแทนได้คือ 256 ดังนั้น ในการบวกเลขประเภทนี้หากผลลัพธ์ที่ได้มีค่าเกิน 255 จะทำให้เกิดปัญหา ดังแสดงในตัวอย่างต่อไปนี้

	150 +		10010110+	
	30		00011110	ผลลัพธ์ไม่เกิน 255
=	180	=	10110100	carry flag เคลียร์

	150+		10010110+	
	150		10010110	ผลลัพธ์เกิน 255
=	300	=	1 00101100	carry flag เซต

จากตัวอย่างจะเห็นว่าหากผลลัพธ์มีขนาดเกิน 255 จะทำให้เลขไบนารีที่ใช้แทนผลลัพธ์มีค่าผิดไปจากความเป็นจริง และเมื่อพิจารณาให้ดีจะพบว่า เมื่อผลลัพธ์จากการบวกเลขจำนวนเต็มที่ไม่คิดเครื่องหมายมีขนาดเกิน 255 จำเป็นที่จะต้องใช้เลขไบนารีเพิ่มอีก 1 บิต เพื่อใช้แทนผลลัพธ์เสมอ ดังนั้นเมื่อบวกเลขแล้วได้ผลลัพธ์เกินพิกัด เลขไบนารี 1 หลัก ที่เกินมานี้จะไปเซตบิต carry flag ของ รีจิสเตอร์ ใช้งานเฉพาะ PSW เพื่อให้สามารถตรวจสอบได้ในกรณีที่มีการบวกเลขเกินพิกัด ดังนั้น ในการบวกเลขประเภทนี้บิต carry flag จะเป็นตัวบอกสถานะของผลลัพธ์ว่ามีขนาดเกินพิกัดหรือไม่

กรณีที่บวกเลขที่คิดเครื่องหมาย การบวกเลขที่คิดเครื่องหมายมีความเป็นไปได้สองกรณีเท่านั้น คือ เลขที่นำมาบวกกันเป็นเครื่องหมายเดียวกันหรือเครื่องหมายต่างกัน หากเลขที่นำมาบวกกันมีเครื่องหมายต่างกัน ผลลัพธ์ที่ได้ย่อมถูกต้องเพราะสามารถแทนด้วยเลขฐานสองได้เสมอ ในกรณีนี้ overflow flag ถูกเคลียร์ เสมอ แค่บิต carry flag อาจถูกเซตหรือเคลียร์ขึ้นกับค่าของเลขที่นำมาบวกกัน ดังตัวอย่างต่อไปนี้

1101 1000	-40		
0111 1000	120 +		carry flag เซต
1 0101 0000	80		coverflow flag เคลียร์

ในกรณีเลขที่ต้องการนำมาบวกกันมีเครื่องหมายเหมือนกัน ผลลัพธ์ที่ได้อาจจะถูกต้องหรือไม่ถูกต้องอันเนื่องมาจากไม่สามารถใช้เลขฐานสองแทนผลลัพธ์ที่เกิดขึ้นได้ ดังนั้นการบวกเลขที่มีเครื่องหมายเหมือนกันอาจจะเกิดการที่แตกต่างกันได้ 4 กรณีดังนี้

- บวกเลขจำนวนเต็มบวก ได้ผลลัพธ์ไม่เกินพิกัด ดังตัวอย่าง

	0000 0010		2	
	0000 0101 +		5 +	
=	0000 0111	=	7	overflow flag ถูกเคลียร์

- บวกเลขจำนวนเต็มบวก ได้ผลลัพธ์เกินพิกัด ดังตัวอย่าง

0110 0100	100	
0110 0100 +	100 +	
1100 1000	200	overflow flag ถูกเซต

- บวกเลขจำนวนเต็มลบ ได้ผลลัพธ์ไม่เกินพิกัด ดังต่อไปนี้

1101 1000	-40	
1100 1100	-50 +	
1 1010 0110	-90	overflow flag ถูกเคลียร์

- บวกเลขจำนวนเต็มลบ ได้ผลลัพธ์เกินพิกัด ดังตัวอย่าง

1100 0000	-64	
1011 1010	-70 +	
1 0111 1010	-134	overflow flag เซต

จากตัวอย่างการบวกเลขที่มีเครื่องหมายเหมือนกันข้างต้นนี้ สรุปได้ว่ากรณีที่ overflow flag ถูกเซตหมายความว่าผลลัพธ์ที่ได้มีค่าเกินเลขจำนวนเต็มบวกสูงสุดหรือเลขจำนวนเต็บลบสูงสุดที่สามารถแทนได้ ดังนั้นกรณี ต่าง ๆ ที่อาจเกิดขึ้นในการบวกเลขที่มีเครื่องหมายสามารถตรวจสอบได้ จาก overflow flag ดังต่อไปนี้

เลขจำนวนเต็มบวก + เลขจำนวนเต็มบวก ได้ผลลัพธ์เกินพิกัด
overflow flag เซต
เลขจำนวนเต็มบวก + เลขจำนวนเต็มบวก ได้ผลลัพธ์ไม่เกินพิกัด
overflow flag เคลียร์
เลขจำนวนเต็มลบ + เลขจำนวนเต็มลบ ได้ผลลัพธ์เกินพิกัด
overflow flag เซต
เลขจำนวนเต็มลบ + เลขจำนวนเต็มลบ ได้ผลลัพธ์ไม่เกินพิกัด
overflow flag เคลียร์
เลขจำนวนเต็มบวก + เลขจำนวนเต็มลบ ผลลัพธ์ไม่เกินพิกัดแน่นอน
overflow flag เคลียร์เสมอ

การบวกเลขที่มีขนาดหลายไบต์ ในหัวข้อที่ผ่านมาเราใช้เลขไบนารีขนาด 8 บิต เพื่อแทนจำนวนที่ต้องการนำมาบวก เลขจำนวนเต็มที่แทนได้มีค่าแตกต่างกันรวมทั้งสิ้น 256 ค่าโดยอาจจะเป็นเลขที่คิดเครื่องหมายหรือไม่คิดเครื่องหมายขึ้นอยู่กับผู้ใช้เป็นคนกำหนด (0 – 256 ในแบบไม่คิดเครื่องหมาย –1 ถึง -128 และ 0 ถึง 127 ในแบบคิดเครื่องหมาย) แต่ในบางกรณีผู้เขียนโปรแกรมอาจต้องการความละเอียดของจำนวนที่มากกว่านี้ก็สามารถกระทำได้โดยการใช้เลขไบนารีที่มีขนาดมากขึ้นเพื่อแทนจำนวนที่ต้องการ เช่น ใช้เลขไบนารีขนาด 16 บิตแทนจำนวนต่าง ๆ ได้ทั้งสิ้น 65536 จำนวน เป็นต้น

การบวกเลขไบนารีที่มีขนาดหลายไลต์นี้ สามารถกระทำได้โดยนำมาบวกกันทีละไบต์ หากผลลัพธ์ที่ได้เกิด overflow จะต้องนำ carry flag ไปใช้ในเป็นตัวทดสำหรับการบวกในไบต์ถัดไป โดยการใช้คำสั่ง ADDC A,<byte>

4.1.3 กลุ่มคำสั่งลบเลขจำนวนเต็ม

กลุ่มคำสั่งที่ใช้ในการลบเลขจำนวนเต็มมีเพียงคำสั่งเดียวเท่านั้น คือ SUBB A, <byte> ดังแสดงในตารางที่ 1 ซึ่งจะเห็นได้ว่าสามารถใช้วิธีการเข้าถึงข้อมูลได้เหมือนกับคำสั่งบวก ดังนั้น คำสั่งในกลุ่มนี้จึงประกอบไปด้วยคำสั่งย่อย ดังนี้

SABB A, Rn	SUBB A, @ Ri
SABB A, direct	SUBB A, # data

คำสั่ง SABB A, <byte> นี้จะทำการลบบิต carry flag ออกจากผลลัพธ์ด้วยเสมอ ทั้งนี้เพื่อใช้ในการลบที่มีขนาดหลายไบต์ หากเราไม่ต้องการให้มีการลบลิต carry flag หรือ ต้องการลบเลขที่มีขนาดเพียง 12.1 ไบต์ ผู้เขียนโปรแกรมต้องเคลียร์ค่าของบิต carry flag ก่อน ปฏิบัติคำสั่งนี้เสมอ

คำสั่งในการลบมีผลกระทบต่อ flag บางตัวในรีจิสเตอร์ใช้งานเฉพาะ PSW ดังนี้

- บิต carry flag : ถูกเซตเมื่อมีตัวยืมเกิดขึ้นจากบิตที่ 7
- บิต auxilary flag : ถูกเซตเมื่อมีตัวยืมเกิดขึ้นในบิตที่ 3
- บิต overflow flag : ถูกเซตเมื่อมีตัวยืมเกิดขึ้นจากบิตที่ 7 แต่ไม่มีในบิตที่ 6 หรือตัวยืมเกิดขึ้นในบิตที่ 6 แต่ไม่มีในบิตที่ 7

การลบจำนวนเต็มที่มีลักษณะเช่นเดียวกับคำสั่งในการบวก ผู้เขียนในโปรแกรมสามารถแทนเลขไบนารีที่ใช้ในการให้เป็นเลขจำนวนเต็มที่คิดเครื่องคิดเลขหรือไม่คิดเครื่องหมายก็ได้ โดยในการลบเลขทั้งสองประเภทนี้รายละเอียดที่แตกต่างกันดังต่อไปนี้

กรณีที่ลบเลขที่ไม่คิดเครื่องหมาย ถ้าเป็นเลขที่มีขนาดเพียง 1 ไบต์ ผู้เขียนโปรแกรมจะต้องเคลียร์ค่าของบิต carry flag ก่อนเสมอ แต่ถ้าเป็นเลขที่มีขนาดมากกว่า 1 ไบต์ ก็ให้เคลียร์บิต carry flag ในไบต์แรก แต่ในไบต์ถัดไปไม่ต้องเคลียร์ เพื่อให้สามารถนำผลจากไบต์แรกมาคิดในไบต์ต่อไปได้

ในกรณีเลขที่ไม่มีเครื่องหมาย จะใช้เลขไบนารีขนาด 8 หลัก แทนขนาดของจำนวนโดย Carry flag จะถูกเซตในกรณีที่ตัวตั้งน้อยกว่าตัวลบ (ได้ผลลัพธ์เป็นเลขลบ) แต่ overflow flag มีค่าเป็น 0 เสมอ ดังแสดงในตัวอย่างต่อไปนี้

125	-	01111101		100	- 01100100
100		01100100		125	01111101
25		00011001	carry flag เคลียร์	-25	00011001 carry flag เซต

กรณีที่ลบเลขที่คิดเครื่องหมาย เช่นเดียวกับการบวกเลขที่คิดเครื่องหมาย เนื่องจากเลขที่นำมาลบกัน 2 ชนิด คือ เลขจำนวนเต็มบวก และจำนวนเต็มลบ ดังนั้น จะมีโอกาสเกิดเหตุการณ์ดังต่อไปนี้

- กรณีการลบเลขที่มีเครื่องหมายเดียวกัน เนื่องจากลบเลขที่มีเครื่องหมายเดียวกันผลลัพธ์ที่ได้ไม่มีโอกาสเกิน – 128 หรือ 127 ดังนั้นบิต overflow flag และ carry flag จะถูกเคลียร์เสมอ ดังตัวอย่างต่อไปนี้

- 100 -	- 25 -	100 -	25 -
-75	-100	25	100
25	75	125	-75

- กรณีลบเลขที่มีเครื่องหมายต่างกัน มีโอกาสที่ผลลัพธ์จะเกินหรือไม่เกินพิกัดของเลขไบนารีที่ใช้แทนได้ โดยหากผลลัพธ์มีค่าเกินพิกัด (เกิน - 128 หรือ +127) บิต overflow flag จะถูกเซตแต่ถ้าผลลัพธ์มีค่าไม่เกินพิกัดบิต overflow flag จะถูกเคลียร์ ดังแสดงในตัวอย่างต่อไปนี้

-100 -		-150 -	
25		50	
-125	overflow flag ถูกเคลียร์	-200	overflow flag ถูกเซต

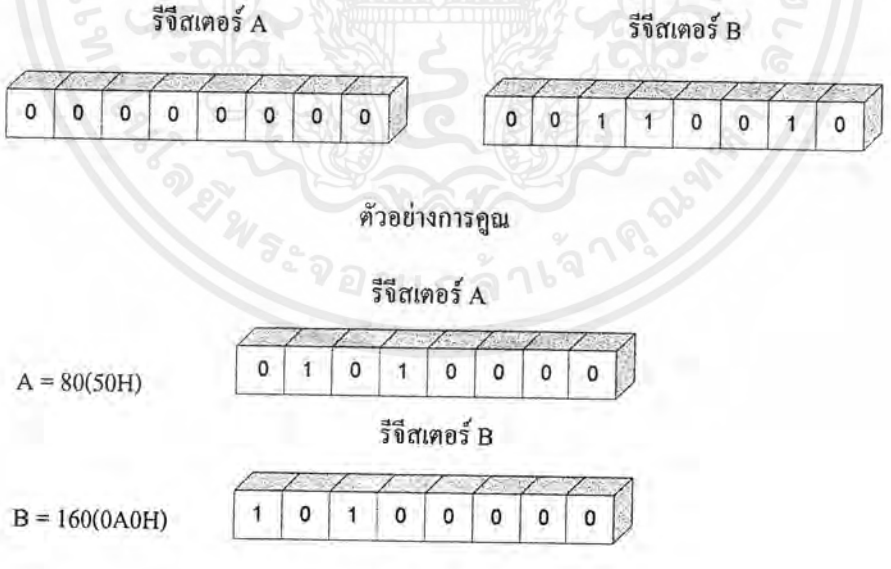
ดังนั้นในกรณีลบเลขที่คิดเครื่องหมาย overflow flag จะเป็นตัวบอกว่าขณะนี้ขณะนี้ผลลัพธ์ที่ได้มีค่าเกินค่าต่ำสุดที่มีได้ ส่วนในการลบเลขที่ไม่คิดเครื่องหมาย overflow flag จะถูกเคลียร์เสมอ แต่ carry flag อาจถูกเซตหรือเคลียร์ขึ้นกับค่าของตัวตั้งและตัวลบ

4.1.4 กลุ่มคำสั่งคูณและหาร

ในไมโครโปเซสเซอร์ บางเบอร์ไม่มีคำสั่งคูณและหาร ผู้เขียนโปรแกรมจะต้องเขียนโปรแกรมเพื่อทำงานแทนคำสั่งคูณหรือหารเลขเอง ซึ่งนับว่ายุ่งยากและไม่สะดวก อีกทั้งยังเสียเวลาในการทำงานที่นานกว่าปกติ เพราะต้องปฏิบัติตามคำสั่งหลายคำสั่ง แต่ใน MCS –50ได้อำนวยความสะดวกให้ผู้เขียนโปแกรมโดยบรรจุคำสั่งในการคูณหรือหารไว้ให้ผู้ใช้ด้วย คำสั่งทั้งสองนี้สามารถทำงานได้ภายในเวลา ประมาณ 4 ไมโครวินาที

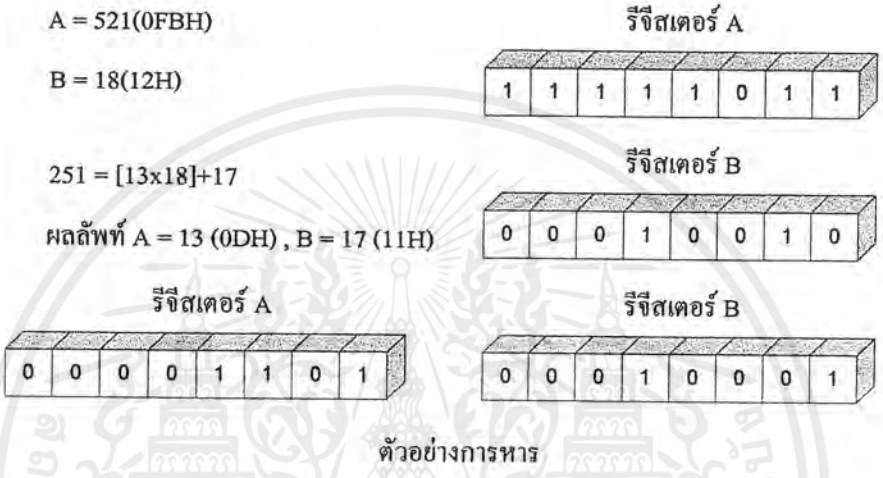
คำสั่งในการคูณหรือการหารจะกระทำที่รีจิสเตอร์ A และ B เสมอ ดังนี้

ในการคูณจะนำค่าในรีจิสเตอร์ A มาคูณกับค่าในรีจิสเตอร์ B ผลลัพธ์ที่เก็บไว้ในรีจิสเตอร์ Aและ B โดยรีจิสเตอร์ A เก็บไบต์ต่ำ ส่วนรีจิสเตอร์ B เก็บไบต์สูง overflow flag จะถูกเคลียร์หาค่าในไบต์สูงของผลลัพธ์ (เก็บไว้ในรีจิสเตอร์ B) มีค่าเป็นศูนย์ และถูกเซตเมื่อค่าในไบต์สูงของผลลัพธ์มีค่าไม่เป็นศูนย์ carry flag จะถูกเคลียร์เสมอ ส่วน asuxillary flag ไม่ถูกเปลี่ยนแปลงในภายหลังคำสั่งนี้



ภาพที่ 4.2 แสดงตัวอย่างการคูณ

ในการหารจะนำค่าในรีจิสเตอร์ Aมาหารด้วยค่าในรีจิตเตอร์ B ผลลัพธ์ที่ได้เก็บไว้ในรีจิสเตอร์ A และ B โดยรีจิสเตอร์ A เก็บจำนวนเต็มที่ได้จากการหาร ส่วนรีจิสเตอร์ B เก็บเศษที่ได้จากการหาร หากมีการหารด้วยศูนย์ (ค่าในรีจิสเตอร์ B เป็น 0) overflow flag จะถูกเซตและค่าในรีจิสเตอร์ A และ B จะมีค่าเป็นอะไรก็ได้ หากตัวหารไม่เป็นศูนย์ overflow flag และ carry flag จะถูกเคลียร์ส่วน auxilary flag ไม่ถูกเปลี่ยนค่าภายหลังกระทำการคำสั่งนี้ ดังแสดงในตัวอย่าง



ภาพที่ 4.3 แสดงตัวอย่างการหาร

4.1.5 กลุ่มคำสั่งที่ปรับค่าผลลัพธ์จากการบวกเลขรหัส

เนื่องจาก MCS – 51 บวกเลขในฐานสิบหกหรือฐานสองเท่านั้น แต่ในบางครั้งที่เราต้องการบวกเลขฐานสิบซึ่งแทนด้วยรหัส BCD การบวกเลขฐานสิบที่แทนด้วยรหัส BCD ใน MCS – 51 สามารถกระทำได้เช่นกันเพียงแต่ต้องใช้คำสั่ง DA A ต่อจากคำสั่งบวกตามปกติ คำสั่ง DA A จะทำงานโดยการเปลี่ยนค่าสัมผัสที่ได้จากการบวกซึ่งเป็นเลขฐานสิบหกให้กลายเป็นเลขฐานสิบซึ่งแทนด้วยรหัส BCD เอง

เลขฐานสิบมีวิธีการด้วยรหัส BCD โดยใช้เลขฐานสองหรือเลขไบนารีเพียง 4 บิต

0010 ใช้แทนเลข 2	0111 ใช้แทนเลข 7
0011 ใช้แทนเลข 3	1000 ใช้แทนเลข 8
0100 ใช้แทนเลข 4	1001 ใช้แทนเลข 9

จะเห็นได้ว่าเลขรหัส BCD ใช้เลขฐานสองเพียง 4 บิตมาแทนเลข 0 ถึง 9 ดังนั้นเลขไบนารีขนาด ไบต์สามารถแทนเลข BCD ได้ 2 หลัก ตั้งแต่ 00 –99 ถ้าใช้เลขไบนารี 8 หลักแทนเลข

ฐานสิบแบบปกติจะแทนได้ตั้งแต่ 0 ถึง 255 ตัวอย่างต่อไปนี้จะแสดงให้เห็นข้อแตกต่างระหว่างการใช้เลขไบนารีแทนเลขฐานสิบแบบปกติ และเมื่อใช้เลขไบนารีเลขฐานสิบโดยใช้รหัส BCD

00000010 = ในรหัส BCD	00000010 = 2 ในเลขฐานสิบ
01001000 = ในรหัส BCD	01001000 = 72 ในเลขฐานสิบ
10001000 = ในรหัส BCD	10001000 = 132 ในเลขฐานสิบ
11111111 ไม่มีค่าในรหัส BCD	11111111 = 255 ในเลขฐานสิบ

จะเห็นว่ารหัส BCD ที่ใช้แทนเลขฐานสิบสามารถทำความเข้าใจและหาค่าเป็นเลขฐานสิบได้ง่ายกว่า ดังนั้นบางครั้งเราจึงนิยมที่จะบวกหรือลบเลขที่ใช้รหัส BCD เพราะรหัสนี้สื่อความหมายมากกว่าเลขฐานสองมาก แต่เนื่องจากซีพียู ใน MCS – 51 บวกเลขได้ในฐานสองหรือฐานสิบหกเท่านั้น ดังนั้น หากเราแทนเลขที่จะทำการบวกด้วยรหัส BCD MCS - 51 จะไม่มีการทราบว่าเลขที่ต้องการบวกเป็นเลข BCD หรือไม่ และจะทำการบวกแบบเลขฐานสองปกติ ผลลัพธ์ที่ได้จึงเป็นเลขฐานสองหรือฐานสิบหก ทำให้ผลลัพธ์ไม่ตรงตามความต้องการของเรา ดังตัวอย่าง

00010010B	12 BCD
+ 00110100B	34 BCD +
01000110B = 46 ฐาน 16	46 BCD

ไม่ต้องใช้คำสั่ง DA A เพราะได้ผลลัพธ์เท่ากัน

00010101B	15 BCD
+ 00010101B	15 BCD +
00101010 B = 2A ฐาน 16	30 BCD

ต้องใช้คำสั่ง DA A เพื่อเปลี่ยน 2 A เป็น 30

จากตัวอย่างจะเห็นได้ว่าถ้าเลขที่นำมาบวกกันแต่ละหลักได้ผลลัพธ์เกิน 9 จะทำให้ผลลัพธ์ นี้มีค่าเกินกว่าที่รหัส BCD จะสามารถแทนได้ และ MCS – 51 มีคำสั่ง DA A เพื่อช่วยปรับค่าของเลขฐานสิบหกที่ได้จากการบวกเลข BCD ให้เป็นเลขฐานสิบที่ถูกต้อง

ในตัวอย่างของการบวก 15 กับ 15 ได้ผลลัพธ์เป็น 2A เมื่อใช้คำสั่ง DA A แล้ว จะทำให้ผลลัพธ์ที่ได้จากการบวกในรีจิสเตอร์ A ถูกเปลี่ยนเป็น 30

สิ่งหนึ่งที่ผู้เขียนโปรแกรมควรระวังไว้ คือ ก่อนใช้คำสั่ง DA A จำนวนที่จะนำมาบวกเข้าด้วยกันต้องเป็นเลข BCD เท่านั้น

4.2 กลุ่มคำสั่งทางตรรกศาสตร์

คำสั่งกลุ่มต่อไปนี้จะศึกษาต่อจากกลุ่มคำสั่งทางคณิตศาสตร์คือ กลุ่มคำสั่งทางตรรกศาสตร์ ตัวดำเนินการทางตรรกศาสตร์ (operator) ที่สำคัญ ๆ มีอยู่ด้วยกัน 4 ประเภท ซึ่งสามารถเทียบกับคำสั่งในกลุ่มคำสั่งทางตรรกศาสตร์ของ MCS – 51 ได้ดังนี้

AND เทียบได้กับคำสั่ง ANL (AND logical)

OR เทียบได้กับคำสั่ง ORL (OR logical)

XOR เทียบได้กับคำสั่ง XRL (exclusive - OR logical)

NOT เทียบได้กับคำสั่ง CPI (complement)

กลุ่มคำสั่งทางตรรกศาสตร์ที่มีใน MCS – 51 นอกจากจะมีคำสั่งในที่ใช้ในการกระทำทางตรรกศาสตร์โดยทั่วไปแล้ว ยังรวมเอาสั่งในการเลื่อนข้อมูลเป็นวงรอบครั้งละ บิต (rotate) คำสั่งในการสลับที่ข้อมูล 4 บิตบนและ 4 บิตล่างในรีจิสเตอร์ A ไว้เป็นประเภทหนึ่งของกลุ่มคำสั่งทางตรรกศาสตร์อีกด้วย

คำสั่งทั้งหมดในกลุ่มคำสั่งทางตรรกศาสตร์ที่เกี่ยวข้องกับการกระทำทางตรรกศาสตร์จริง ๆ ในกลุ่มนี้มีเพียง 4 คำสั่ง เท่านั้น คือ ANL, XRL, CPL โดยแต่ละคำสั่งจะกระทำการกับข้อมูลแต่ละบิตโดยตรงดังแสดงในตัวอย่างต่อไปนี้

- คำสั่ง ANL : AND ข้อมูลแต่ละบิต ดังนี้

โอเปอร์แรนด์ตัวแรก	10110110B	=	0B6H
โอเปอร์แรนด์ตัวที่สอง	01101101B	=	6DH
ผลลัพธ์	00100100B	=	24 H

- คำสั่ง ORL : OR ข้อมูลแต่ละบิต ดังนี้

โอเปอร์แรนด์ตัวแรก	10110110B	=	0B6H
โอเปอร์แรนด์ตัวที่สอง	01101101B	=	6DH
ผลลัพธ์	11011011B	=	0DBH

- คำสั่ง CPL : complement ข้อมูลแต่ละบิต ดังนี้

โอเปอร์แรนด์	10110110B	=	0B6H
ผลลัพธ์	010010011B	=	49 H

กลุ่มคำสั่งทางตรรกศาสตร์ ANL, ORL , XRL ที่แสดงในตารางที่ สามารถแยกออก

ได้เป็น 2 ประเภท ได้ดังนี้

คำสั่งที่ใช้รีจิสเตอร์ A เป็นที่เก็บผลลัพธ์ (destination register) คำสั่งนี้จะสามารถกระทำการทางตรรกศาสตร์กับโอเปอร์แรนด์อีกตัวหนึ่ง โอเปอร์แรนด์ตัวที่จะนำมากระทำการกับรีจิสเตอร์ A นี้สามารถใช้วิธีการเข้าถึงข้อมูลได้ทุกวิธี แต่ผลลัพธ์สุดท้ายจะนำไปเก็บไว้ในรีจิสเตอร์ A ตามเดิม คำสั่งในกลุ่มนี้ มีดังต่อไปนี้

ANL A, Direct	ORL A, Direct	XRL A, Direct
ANL A, @ Ri	ORL A, @Ri	XRL A, @Ri
ANL A,Rn	ORL A, Rn	XRL A,Rn
ANL A,# data	ORL A, #data	XRL A,#data

คำสั่งที่ใช้หน่วยความจำสำหรับเก็บข้อมูลภายใน MCS –51 เป็นที่เก็บผลลัพธ์ คำสั่งกลุ่มนี้สามารถใช้วิธีเข้าถึงข้อมูลของโอเปอร์แรนด์ที่จะนำมากระทำการทางตรรกศาสตร์กับข้อมูลในหน่วยความจำสำหรับเก็บข้อมูลภายใน MCS – 51 ได้เพียง 2 วิธีคือ ข้อมูลทางโอเปอร์แรนด์เป็นค่าคงที่ (immediate constants) ที่กำหนดเองในคำสั่ง หรือโอเปอร์แรนด์เป็นข้อมูลในรีจิสเตอร์ A คำสั่งทางตรรกศาสตร์ในกลุ่มนี้มีดังต่อไปนี้

ANL Direct , A	ORL Direct	XRL Direct ,A
ANL Direct , #data	ORL Direct, # data	XRL Direct ,#data

คำสั่งกลุ่มที่ใช้หน่วยความจำสำหรับเก็บข้อมูลภายใน MCS – 51 เป็นที่เก็บผลลัพธ์

เนื่องจากหน่วยความจำสำหรับเก็บข้อมูลภายใน MCS – 51 สามารถนำมากระทำการทางตรรกศาตร์ได้โดยตรงนี้ทำให้เกิดความสะดวกแก่ผู้ใช้มาก เพราะรีจิสเตอร์ ที่ใช้ควบคุมฮาร์ดแวร์ใน MCS – 51 ทุกตัวจะอยู่ในหน่วยความจำส่วนนี้ด้วย ดังนั้นการทำงานทางตรรกศาสตร์ใด ๆ กับข้อมูลในหน่วยความจำของ MCS – 51 สามารถกระทำได้โดยไม่ต้องผ่านรีจิสเตอร์ A ด้วย ดังนั้นการส่งข้อมูลหรือรับข้อมูลจากพอร์ตรวมทั้งกระทำการทางตรรกศาสตร์แล้วนำผลที่ได้ไปส่งออกไปที่พอร์ตสามารถกระทำได้โดยตรงดังตัวอย่าง

XRL PO, OFFH

จะนำค่าของพอร์ต 0 ไปทำการอินเวอรส์แล้วส่งกลับไปใหม่

คำสั่ง ANL, ORL , XRL ที่ได้กล่าวผ่านมานี้สามารถอ้างข้อมูลได้หลายวิธี แต่คำสั่ง CPL จะบังคับให้ทำงานที่รีจิสเตอร์ A เพียงตัวเดียวเท่านั้น นั่นคือคำสั่ง

CPL A

คำสั่งนี้ใช้เปลี่ยนค่าแต่ละบิตที่อยู่ในรีจิสเตอร์ A ให้เป็นตรงกันข้าม หากต้องการเปลี่ยนข้อมูลในหน่วยความจำตำแหน่งต่าง ๆ แต่ละบิตเป็นตรงกันข้าม ก็สามารถใช้คำสั่งนี้ผ่านรีจิสเตอร์ A โดยการนำข้อมูลในหน่วยความจำตำแหน่งที่ต้องการมาไว้ในรีจิสเตอร์ A จากนั้นก็กระทำคำสั่ง CPL A แล้วย้ายผลลัพธ์ที่ได้กลับไปยังตำแหน่งเดิม การเปลี่ยนค่าข้อมูลในหน่วยความจำแต่ละบิตเป็นตรงกันข้ามนี้อาจใช้คำสั่งแทนซึ่งจะสะดวกกว่ามาก นั่นคือคำสั่ง

XRL direct ,# OFFH

ซึ่งมีผลเดียวกับการนำข้อมูลในหน่วยความจำแต่ละบิตมาเปลี่ยนเป็นตรงกันข้าม แล้วผลลัพธ์ที่ได้ก็เก็บไว้ในหน่วยความจำตำแหน่งเดิม

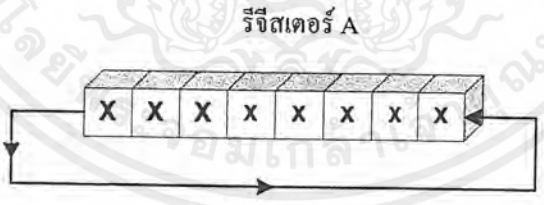
คำสั่งอื่นๆ ที่จัดอยู่ในกลุ่มคำสั่งทางตรรกศาสตร์ จะทำงานโดยไม่ได้กระทำการใด ๆ ทางตรรกศาสตร์เลย แต่เราจะจัดรวมไว้ในกลุ่มนี้ด้วย ได้แก่

- คำสั่ง CLR A จะเป็นการเคลียร์ค่าในรีจิสเตอร์ A โดยการโหลดรีจิสเตอร์ A ด้วย OOH แต่ถ้าผู้ใช้ต้องการเคลียร์ค่าในหน่วยความจำตำแหน่งต่างๆ ก็สามารถกระทำได้โดยใช้คำสั่งอื่นแทน ดังนี้

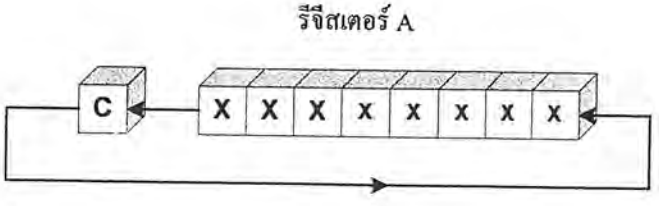
ANL direct .# OOH หรือ

MOV direct, # OOH

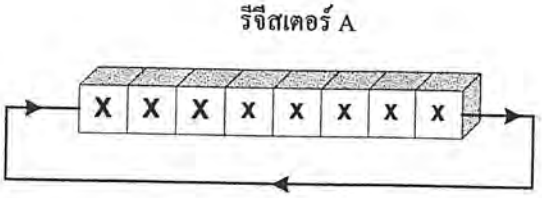
- คำสั่งในการเคลื่อนข้อมูลเป็นวงรอบ ซึ่งประกอบไปด้วยคำสั่งย่อย ๆ ดังนี้
- RL A มีการทำงานดังต่อไปนี้



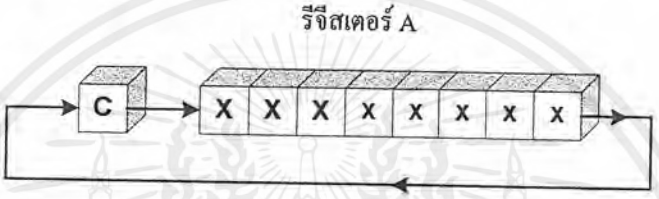
- RLC A มีการทำงานดังนี้



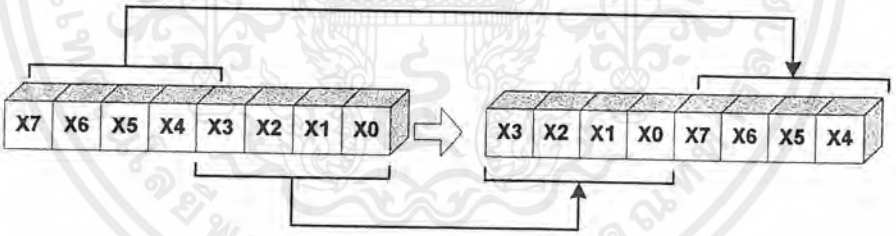
RR A มีการทำงานดังต่อไปนี้



RRC A มีการทำงานดังต่อไปนี้



SWAP A มีการทำงานดังต่อไปนี้



ภาพที่ 4.4 แสดงคำสั่งในการเคลื่อนข้อมูลเป็นวงรอบ

4.3 กลุ่มคำสั่งในการเคลื่อนย้ายข้อมูล

การเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำหรือรีจิสเตอร์นับเป็นสิ่งจำเป็นสำหรับการทำงานของไมโครโปรเซสเซอร์หรือไมโครคอนโทรลเลอร์ ทั้งนี้เพราะคำสั่งต่าง ๆ เช่น คำสั่งทางคณิตศาสตร์ คำสั่งทางตรรกศาสตร์ ฯลฯ ส่วนใหญ่จะทำงานกับรีจิสเตอร์ใช้งานเฉพาะบางตัวเท่านั้น เช่นรีจิสเตอร์ A, B ดังนั้นหากเราต้องการนำข้อมูลมาประมวลผลโดยคำสั่งเหล่านี้ จำเป็นจะต้องย้ายข้อมูลที่ต้องการไปยังรีจิสเตอร์เหล่านี้ และเมื่อได้ผลลัพธ์จากการคำนวณแล้วจึงเคลื่อนย้ายข้อมูลกลับมาเก็บไว้ในหน่วยความจำ ณ ตำแหน่งที่ต้องการ

คำสั่งในการเคลื่อนย้ายข้อมูล สามารถแบ่งออกได้ เป็น 3 กลุ่ม ตามประเภทของหน่วยความจำที่ใช้ในคำสั่ง ได้แก่

- คำสั่งในการเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำสำหรับเก็บข้อมูลภายในชิป
- คำสั่งในการเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำสำหรับเก็บข้อมูลภายนอกชิป
- คำสั่งในการเคลื่อนย้ายข้อมูลจากหน่วยความจำสำหรับเก็บโปรแกรมทั้งภายในและภายนอกชิป

รายละเอียดของกลุ่มคำสั่งสำหรับการเคลื่อนย้ายทั้ง 3 กลุ่ม มีดังนี้

คำสั่งแต่ละคำสั่งในกลุ่มนี้มีวิธีการเข้าถึงข้อมูลได้หลายวิธี เราสามารถแบ่งประเภทของคำสั่งในกลุ่มนี้ออกเป็น 3 กลุ่ม ดังนี้

- กลุ่มคำสั่ง MOV (เคลื่อนย้ายข้อมูล) ได้แก่คำสั่ง

MOV Rn, A	MOV @Ri, A
MOV Rn, direct	MOV @Ri, direct
MOV Rn, #data 8 bit	MOV @Ri, # data 8 bit
MOV Direct , A	MOV A, Rn
MOV direct , Rn	MOV A,direct
MOV direct,direct	MOV A, @ Ri
MOV direct , @ Ri	MOV A, data 16 bit

- กลุ่มคำสั่ง PUSH, POP ได้แก่ คำสั่ง

PUSH direct	POP direct

- กลุ่มคำสั่ง XCH (สลับที่ข้อมูล) ได้แก่ คำสั่ง

XCH A,Rn	XCH A, @Ri
XCH A,direct	XCHD A, @Ri

หมายเหตุ : Rn = RO – R7 = Ri

กลุ่มคำสั่ง MOV จะสังเกตได้ว่าไม่มีคำสั่ง MOV Ri, Ri และ MOV @ Ri, @ Ri และเนื่องจากรีจิสเตอร์ A อยู่ในหน่วยความจำสำหรับเก็บข้อมูลทั่วไปในชิปตำแหน่ง OEOH ดังนั้นการเคลื่อนย้ายข้อมูลที่เกี่ยวข้องกับรีจิสเตอร์ A โดยเฉพาะก็เพื่อลดขนาดของรหัสคำสั่งให้สั้นลงนั่นเอง นั่นคือ คำสั่งที่ใช้ในการเคลื่อนย้ายข้อมูลกับรีจิสเตอร์ A นี้ผู้ใช้ไม่จำเป็นจะต้องระบุ

ตำแหน่งของหน่วยความจำที่เป็นรีจิสเตอร์ A ในรหัสคำสั่งเลย เพราะ MCS – 51 จะเข้าใจว่าเป็นรีจิสเตอร์ A เองจากรหัสคำสั่ง

คำสั่ง MOV direct, direct มีไว้เพื่อใช้เคลื่อนย้ายข้อมูลภายในหน่วยความจำสำหรับเก็บข้อมูลทั่วไปภายในชิปบริเวณ 128 ไบต์ล่าง และหน่วยความจำสำหรับเก็บข้อมูลภายในชิปที่ใช้เป็นรีจิสเตอร์ใช้งานเฉพาะของ MCS – 51

กลุ่มคำสั่ง PUSH ,POP คำสั่ง PUSH , POP ทั้งสองนี้นับเป็นประเภทหนึ่งของกลุ่มคำสั่งในการเคลื่อนย้ายข้อมูล แต่การเคลื่อนย้ายข้อมูลของคำสั่ง PUSH , POP นี้มีจุดประสงค์ที่แตกต่างออกไปจากคำสั่ง MOV ทั่วไป คือ คำสั่งทั้งสองใช้เพื่อเคลื่อนย้ายข้อมูลไปเก็บไว้ในหน่วยความจำตำแหน่งที่ถูกระบบโดยค่าในรีจิสเตอร์ที่ใช้งานเฉพาะ SP ซึ่งเราเรียกบริเวณนี้ว่าสแตกแอเรีย และสามารถเรียกกลับคืนมาได้เมื่อต้องการ โดยมีรายละเอียด ดังนี้

PUSH direct คำสั่งนี้สามารถใช้วิธีการเข้าถึงข้อมูลได้เพียงวิธีเดียวเท่านั้น คือ วิธีการเข้าถึงข้อมูลโดยตรง การทำงานของคำสั่งสามารถเทียบได้กับคำสั่งต่อไปนี้

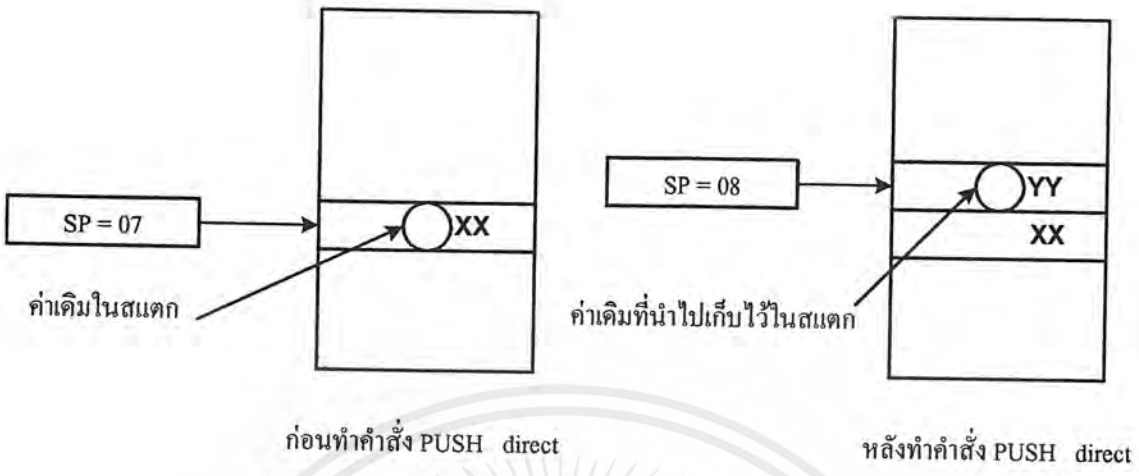
INC SP

MOV @ SP, direct ; คำสั่งนี้ไม่มีใน MCS – 51

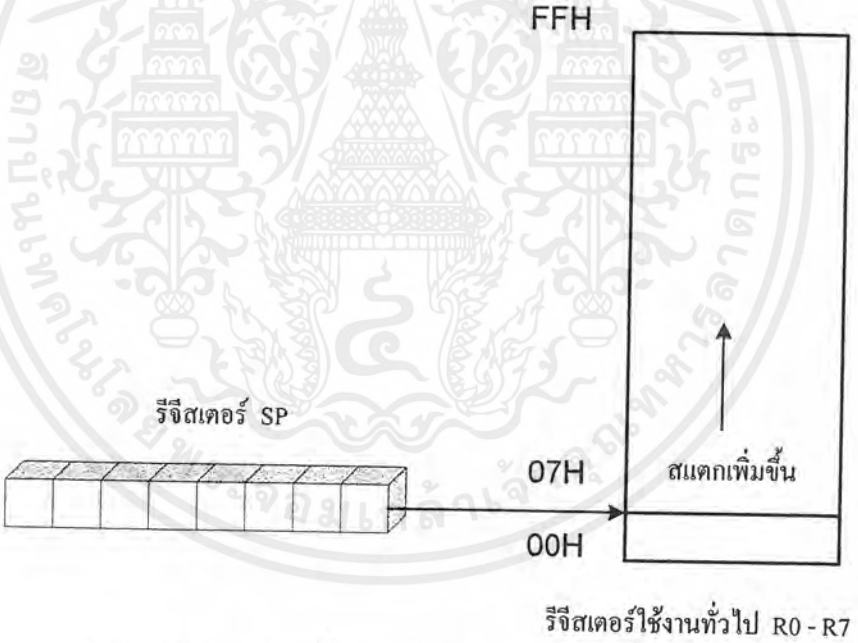
ในการทำงานของคำสั่ง ก่อนอื่นจะไปเพิ่มค่าของรีจิสเตอร์ใช้งานเฉพาะ SP ซึ่งจะทำหน้าที่เป็นตัวระบุตำแหน่งของหน่วยความจำที่จะนำข้อมูลไปเก็บ หลังจากนั้นก็นำเอาข้อมูลในหน่วยความจำ ณ ตำแหน่งที่ถูกระบุโดยการเข้าถึงข้อมูลโดยตรงไปไว้ในหน่วยความจำ ณ ตำแหน่งเดียวกับ รีจิสเตอร์ SP ชี้อยู่

เนื่องจาก MCS – 51 เพิ่มค่าของรีจิสเตอร์ SP ให้เองเมื่อกระทำคำสั่ง PUSH ดังนั้นผู้เขียนโปรแกรมไม่จำเป็นที่จะต้องไปยุ่งเกี่ยวกับค่าของรีจิสเตอร์นี้เลย ข้อมูลจะถูกเก็บในลักษณะที่ซ้อนกันไปเรื่อย ๆ ดังแสดงในภาพที่ 4.5 และภาพที่ 4.6

รีจิสเตอร์ SP มีขนาด 8 บิต ดังนั้นสามารถระบุตำแหน่งหน่วยงานความจำได้เพียง 256 ไบต์ เท่านั้น ทำให้บริเวณของาแตกแอเรียมีขนาดได้ไม่เกิน 256 ไบต์ด้วย แต่เนื่องจากบริเวณของสแตกแอเรีย คือ ส่วนหนึ่งของหน่วยความจำสำหรับเก็บข้อมูลทั่วไปภายในชิปซึ่งใช้รีจิสเตอร์ทั่วไป RO – R7 ด้วย ดังนั้นขนาดของสแตกแอเรีย จะมีขนาดไม่ถึง 256 ไบต์ โดยในตอนรีเซตหรือเริ่มจ่ายพลังงานให้ MCS- 51 ค่าของรีจิสเตอร์ SPจะถูกปรับให้เป็น 07H เองเพื่อไม่ให้ไปซ้อนกับรีจิสเตอร์ทั่วไป RO – R7 ดังแสดงในภาพที่ 4.5



ภาพที่ 4.5 แสดงการทำงานของ PUSH direct



ภาพที่ 4.6 แสดงค่าของ รีจีสเตอร์ SP เมื่อตอนรีเซ็ต หรือเริ่มจ่ายพลังงานให้ MCS – 51

จากภาพที่ 4.6 แสดงให้เห็นค่ารีจิสเตอร์ SP เมื่อตอนถูกรีเซตหรือเริ่มจ่ายพลังงานให้ MCS – 51 ซึ่งมีค่าเท่ากับ 07H ทำให้เราไม่สามารถใช้รีจิสเตอร์ทั่วไป R0 – R7 กลุ่มอื่นได้ หากต้องการใช้รีจิสเตอร์ทั่วไป R0 – R7 กลุ่มอื่น ผู้เขียนโปรแกรมจำเป็นต้องเปลี่ยนค่าของรีจิสเตอร์ใช้งานเฉพาะ SP ให้เป็นค่าอื่น เพื่อให้ข้อมูลที่ถูกนำมาเก็บไว้ในหน่วยความจำโดยคำสั่ง PUSH ไม่ไปซ้อนทับกับข้อมูลของรีจิสเตอร์เหล่านี้โดยใช้คำสั่งเคลื่อนย้ายข้อมูลไปไว้ในรีจิสเตอร์ SP ดังนี้

MOV SP, # ADDR คำสั่งนี้เป็นการเปลี่ยนค่าในรีจิสเตอร์ SP โดยตัวโปรแกรมแอสเซมเบลอร์ จะทราบตำแหน่งของรีจิสเตอร์ SP เอง ยกเว้นการแปลคำสั่งโดยการเปิดตาราง (ในตัวรหัสคำสั่งต้องระบุตำแหน่งของรีจิสเตอร์ SP ด้วยค่า 81H

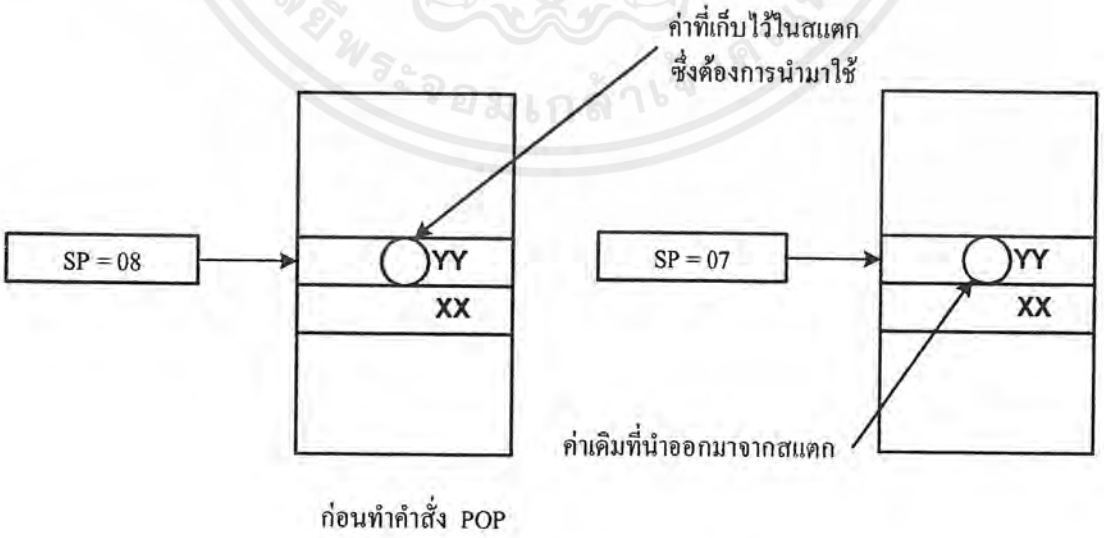
สแตกแอเรีย ส่วนที่มีค่าเกิน 128 ไบต์ จะอยู่ในหน่วยความจำสำหรับเก็บข้อมูลทั่วไปภายในชิปบริเวณ 128 ไบต์บน ซึ่งไม่มีใน MCS – 51 บางเบอร์ ดังนั้นหากเราใช้ MCS–51 เป็นเบอร์ที่ไม่มีหน่วยความจำส่วนนี้สแตกแอเรียจะมีขนาดได้มากที่สุด 128 ไบต์ เท่านั้นและเนื่องจากคำสั่ง PUSH จะเพิ่มค่าของรีจิสเตอร์ SP เองด้วยทุกครั้ง ดังนั้นหากก่อนกระทำคำสั่ง PUSH ค่าในรีจิสเตอร์ SP ขณะนั้น เป็น 7FH และผู้ใช้เลือก MCS–51 เบอร์ที่ไม่มีหน่วยความจำสำหรับเก็บข้อมูลทั่วไปภายในชิปบริเวณ 128ไบต์บน จะทำให้ข้อมูลที่นำไปเก็บไว้โดยคำสั่ง PUSH เกิดการสูญหายได้ แต่ถ้าใช้เบอร์ที่มีหน่วยความจำส่วนนี้และก่อนทำคำสั่ง PUSH ค่าในรีจิสเตอร์ SP มีค่าเป็น OFFH ข้อมูลที่จะนำมาเก็บจะถูกนำมาเก็บจะถูกนำไปไว้ที่ตำแหน่ง OOH ซึ่งเป็นตำแหน่งเริ่มต้นของหน่วยความจำสำหรับเก็บข้อมูลทั่วไปภายในชิป ทำให้ข้อมูลในรีจิสเตอร์ RO-R7 เปลี่ยนไปได้

POP direct คำสั่งนี้จะทำหน้าที่ตรงกันข้ามกับคำสั่ง PUSH โดยมีจุดประสงค์เพื่อนำข้อมูลที่ถูกเก็บไว้ให้หน่วยความจำโดยคำสั่ง PUSH กลับคืนมา

การทำงาน ของ POP direct สามารถเทียบได้กับคำสั่งต่อไปนี้

```
MOV direct , @ SP      ; คำสั่งนี้ไม่มีใน MCS – 51
DEC SP
```

การทำงานของคำสั่งอาจอธิบายด้วยแผนภาพการทำงานดังแสดงในภาพที่ 4.7



ภาพที่ 4.7 แสดงการทำงานของ POP direct

คำสั่ง POP จะทำงานคู่กับคำสั่ง PUSH โดยเมื่อโปรแกรมต้องการใช้รีจิสเตอร์ตัวใดตัวหนึ่งในการประมวลผลและจำเป็นที่ต้องนำค่าเดิมมาใส่คืนเมื่อเสร็จสิ้นการทำงานแล้ว จะสามารถใช้คำสั่ง PUSH, POP เพื่อเก็บรักษาค่าเดิมไว้และนำค่าเดิมมาใส่คืนเมื่อต้องการ ดังแสดงในตัวอย่างต่อไปนี้

```
PUSH 0D0H        ; เก็บค่าของหน่วยความจำตำแหน่ง 0D0H (รีจิสเตอร์ PSW)
                 ; คำสั่งที่มีการใช้รีจิสเตอร์ PSW หรือคำสั่งที่มีผลต่อค่าใน
                 รีจิสเตอร์ PSW
POP 0D0H         ; นำค่าที่เก็บไว้กลับคืนสู่รีจิสเตอร์ PSW
```

PUSH , POP ส่วนใหญ่จะใช้ในโปรแกรมย่อย ซึ่งถูกเรียกใช้จากโปรแกรมหลัก (main program) โดยในโปรแกรม่ย่อยอาจจะมีการใช้รีจิสเตอร์ตัวใดตัวหนึ่งในการประมวลผล หรือมีผลกระทบต่อค่าในรีจิสเตอร์บางตัว ในกรณีนี้จำเป็นต้องรักษาค่าเดิมของรีจิสเตอร์เหล่านี้ไว้เสียก่อน จากนั้นเมื่อทำงานในส่วนของโปรแกรมย่อยเสร็จสิ้นแล้วจึงค่อยนำค่าก็ได้เมื่อใดก็ได้โดยไม่ส่งผลกระทบต่อค่าในรีจิสเตอร์ที่โปรแกรมหลักใช้ในการประมวลผล ดังนั้น หากในโปรแกรมย่อยมีการเก็บรักษาค่าของรีจิสเตอร์ที่ใช้ในระหว่างการทำงาน ผู้เขียนโปรแกรมก็ไม่จำเป็นต้องกังวลกับการเปลี่ยนค่าในรีจิสเตอร์ที่ใช้ประมวลผลของโปรแกรมหลักเมื่อมีการเรียกใช้โปรแกรมย่อยแต่อย่างใด

กลุ่มคำสั่ง XCH คำสั่งนี้เป็นคำสั่งสุดท้ายของกลุ่มคำสั่งย้ายข้อมูลระหว่างหน่วยความจำสำหรับเก็บข้อมูลภายในชิป การทำงานในคำสั่งของคำสั่งจะมีลักษณะที่แตกต่างจากคำสั่ง MOV, PUSH , POP เพราะคำสั่งที่ผ่านมาทั้งหมดเป็นการเคลื่อนย้ายข้อมูลไปยังปลายทางนั่นเอง แต่คำสั่ง XCH (exchange) จะทำการสลับที่ข้อมูลระหว่างต้นทางและปลายทาง ทำให้ค่าของข้อมูลทั้งต้นทางและปลายทาง เปลี่ยนแปลงไปหลังกระทำคำสั่งนี้ ยกเว้นกรณีที่ข้อมูลที่ต้นทางและปลายทางมีค่าเท่ากัน คำสั่งนี้กระทำที่รีจิสเตอร์ A เสมอ ดังนั้น หากต้องการสลับที่ข้อมูลในหน่วยความจำส่วนที่เก็บข้อมูล 2 ตำแหน่ง ก็สามารถกระทำได้โดยการสลับข้อมูลผ่านรีจิสเตอร์ A

คำสั่ง XCH A, <byte> สามารถใช้วิธีการเข้าถึงข้อมูล ได้ 3 วิธี จึงมีคำสั่งนี้จึงมีคำสั่งย่อยอีก 3 คำสั่ง คือ

```
XHA A, Rn
XCH A, direct
XCH A, @ Ri
```

นอกจากคำสั่งทั้ง 3 ยังมีคำสั่งพิเศษซึ่งจะทำการสลับที่ข้อมูลเพียง 44 บิตล่างของไบต์ข้อมูลเท่านั้น แต่มีวิธีการเข้าถึงข้อมูลเพียงวิธีเดียวเท่านั้น คือใช้ค่ารีจิสเตอร์ RO, R1 เป็นตัวระบุตำแหน่งหน่วยความจำ (วิธีการเข้าถึงข้อมูลโดยทางอ้อม) คำสั่งนี้ คือ

XCHD A, @ Ri

4.3.1 กลุ่มคำสั่งในการเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำสำหรับเก็บข้อมูลภายนอกชิป

กลุ่มคำสั่งนี้ เป็นกลุ่มคำสั่งที่ใช้เคลื่อนย้ายข้อมูลภายในหน่วยความจำสำหรับเก็บข้อมูลอยู่ใน MCS 51 เท่านั้น แต่เนื่องจาก MCS–51 เองสามารถติดต่อกับหน่วยความ่จำส่วนที่เก็บข้อมูลภายนอกได้ถึง 64 กิโลไบต์ ดังนั้นจึงต้องมีกลุ่มคำสั่งเพื่อใช้ในการติดต่อกับหน่วยความจำส่วนนี้ อยู่ด้วย

คำสั่งที่ใช้ติดต่อกับหน่วยความจำที่เก็บข้อมูลภายนอก MCS – 51 นี้จะทำผ่านรีจีสเตอร์ A เสมอ และสามารถใช้วิธีการเข้าถึงข้อมูลได้เพียงวิธีเดียว คือ วิธีการเข้าถึงข้อมูลแบบอ้างอิง โดยใช้ค่ารีจิสเตอร์ RO, R1 หรือ DPTR (ไม่สามารถใช้รีจิสเตอร์ SP ได้) เป็นตัวระบุตำแหน่งที่อยู่ของหน่วยความจำภายนอก

เนื่องจากรีจิสเตอร์ RO, R1 มีขนาดเพียง 8 บิตทำให้สามารถระบุตำแหน่งที่อยู่ของหน่วยความจำภายนอกได้ในขอบเขต 256 ไบต์แรกเท่านั้น ส่วนรีจิสเตอร์ DPTR มีขนาด 16 บิต จึงสามารถระบุตำแหน่งหน่วยความจำได้ทั้งสิ้น 64 กิโลไบต์ ซึ่งเท่ากับความสามารถในการติดต่อของหน่วยความจำของ MCS – 51

4.3.2 กลุ่มคำสั่งในการเคลื่อนย้ายข้อมู่ลจากหน่วยความจำสำหรับเก็บโปรแกรมทั้งภายนอกและภายในชิป

คำสั่งในกลุ่มนี้มีอยู่ด้วยกัน 2 คำสั่ง

กลุ่มคำสั่งสองกลุ่มแรกที่ศึกษาผ่านมาเป็นกลุ่มคำสั่งที่ใช้ในการเคลื่อนย้ายข้อมูลกับหน่วยความจำที่เก็บข้อมูลภายในและภายนอก MCS – 51 หน่วยความจำที่เก็บข้อมูลนี้รักษาข้อมูลไว้ได้ก็ต่อเมื่อมีพลังงานเท่านั้น แต่ในบางกรณีเราจำเป็นต้องอาศัยข้อมูลที่มีค่าแน่นอนไม่เปลี่ยนแปลง เช่น รูปแบบของตัวอักษร ค่า sin , cos, tan ของมุมต่าง ๆ ทางตรีโกณมิติ ซึ่งข้อมูลเหล่านี้จะจำเป็นต้องเก็บรักษาไว้ตลอดเวลาถึงแม้ว่าจะไม่มีพลังงานจ่ายให้ก็ตาม เพื่อให้สามารถเรียกข้อมูลมาทำงานได้ทุกครั้งที่เปิดอุปกรณ์ซึ่งควบคุมด้วย MCS – 51 ดังนั้นเราจำเป็นต้องเก็บรักษาข้อมูลเหล่า

นี้ไว้ในหน่วยความจำซึ่งไม่ต้องอาศัยพลังงานในการเก็บรักษาข้อมูล นั่นคือหน่วยความจำที่เก็บโปรแกรม ซึ่งเป็น ROM, EPROM นั่นเอง

เมื่อผู้ออกแบบได้เก็บข้อมูลไว้ในหน่วยความจำที่เก็บโปรแกรมแล้ว จำเป็นที่จะต้องมีคำสั่งในการนำเอาข้อมูลจากหน่วยความ่จำส่วนนี้มาประมวลผลด้วย และใน MCS – 51 เองก็มีชุดคำสั่งในการเคลื่อนย้ายข้อมูลจากหน่วยความจำส่วนนี้มาประมวลผล

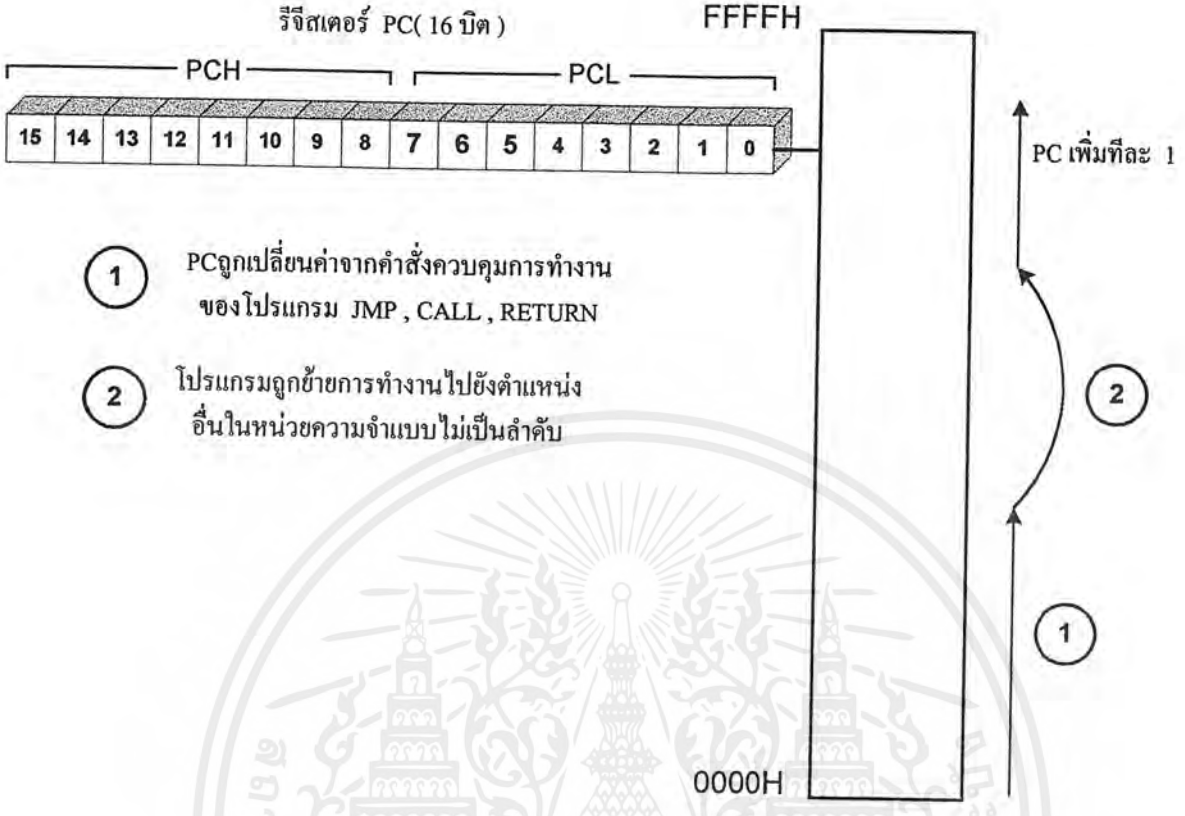
คำสั่งทั้งสองมีจุดมุ่งหมายเพื่อใช้ในการเปิดตาราง โดยข้อมูลในตารางจะอยู่ในหน่วยความจำที่เก็บโปรแกรม ซึ่งสามารถรักษาข้อมูลไว้โดยไม่ต้องอาศัยพลังงาน

4.4 กลุ่มคำสั่งควบคุมลำดับการทำงานของโปรแกรม

โปรแกรมซึ่งเป็นลำดับที่ใช้ในการควบคุมการทำงานของไมโครโทรลเลอร์ จะบรรจุอยู่ในหน่วยความจำถาวร ซึ่งอาจจะเป็น ROM, EPROM ฯลฯ ในการทำงานจริง ๆ ซีพียู่จะนำเอารหัสคำสั่งแต่ละคำสั่งในโปรแกรม ที่อยู่ในหน่วยความจำตำแหน่งต่าง ๆ (เฟตซ์คำสั่ง) มาตีความและปฏิบัติตามคำสั่งเป็นลำดับไปเรื่อย ๆ โดยปกติการเฟตซ์คำสั่งจากโปรแกรมซีพียูจะกระทำโดยการส่งค่าหมายเลขประจำตำแหน่งหรือแอดเดรสของหน่วยความจำที่เก็บคำสั่งแต่ละคำสั่งไว้ เมื่อหน่วยความจำได้รับค่าตำแหน่งจากซีพียู มันจะส่งข้อมูลที่เป็นรหัสคำสั่งที่ให้แก่ไมโครคอนโทรลเลอร์เพื่อปฏิบัติงานต่อไป ในทางปฏิบัติไมโครคอนโทรลเลอร์จะส่งค่าตำแหน่งของหน่วยงานความจำซึ่งกำหนดโดยใช้ค่าในรีจิสเตอร์ PC ค่าของรีจิสเตอร์ ตัวนี้จะเพิ่มขึ้นครั้งละหนึ่งหลังการทำงานแต่ละคำสั่ง ของซีพียู เพื่อจะได้นำข้อมูลที่เป็นรหัสคำสั่งซึ่งอยู่ในตำแหน่งถัดไปจากหน่วยงานความจำมาทำงาน หรือเพื่อนำข้อมูลของคำสั่งที่เก่บไว้ต่อจากรหัสคำสั่งมาประมวลผล ดังนั้นหากในขณะที่ซีพียูกำลังทำงานมีการเปลี่ยนแปลง ค่าในรีจิสเตอร์ PC ก็จะส่งผลให้ลำดับการทำงานของโปรแกรมเปลี่ยนไป การทำงานของรีจิสเตอร์ PC ในภาวะปกติ (เพิ่มครั้งละ หนึ่ง) และในภาวะที่ถูกเปลี่ยนค่าคำสั่งมีดังแสดงในรูปที่

คำสั่งที่ใช้ในการควบคุมลำดับการทำงานของโปรแกรมสามารถแยกออกเป็น 2 ประเภท ได้ดังนี้

- คำสั่งที่ใช้ควบคุมลำดับการทำงานของโปรแกรมแบบไม่มีเงื่อนไข (unconditonal jumps)
- คำสั่งที่ใช้ควบคุมลำดับการทำงานของโปรแกรมแบบมีเงื่อนไข (conditional jumps)



ภาพที่ 4.8 แสดงการทำงานของรีจีสเตอร์ PC

4.4.1 กลุ่มคำสั่งควบคุมโปรแกรมแบบไม่มีเงื่อนไข

กลุ่มคำสั่งที่ใช้ควบคุมลำดับการทำงานของโปรแกรมแบบไม่มีเงื่อนไขจะบังคับให้โปรแกรมย้ายการทำงานไปยังที่อื่นในหน่วยงานความจำซึ่งไม่ใช่ตำแหน่งถัดไปของคำสั่งนี้ทันที คำสั่งในกลุ่มนี้ประกอบไปด้วย 5 คำสั่ง คือ

- JMP addr
- CALL addr
- RET
- RETI
- NOP

คำสั่งทั้ง 5 คำสั่งล้วนแต่ใช้บังคับการทำงานของโปรแกรมอย่างไม่มีเงื่อนไข แต่จะมีวิธี

59

การใช้งานที่แตกต่างกันออกไป ดังนี้

- JMP addr คำสั่งนี้ไม่มีรหัสคำสั่งใน MCS – 51 โดยตรง แต่ใช้สำหรับเขียนโปรแกรมภาษาแอสแซมบลีโดยใช้แทนคำสั่ง JMP อื่น ๆ 3 คำสั่ง คือ
- LJMP addr
- SJMP addr
- AJMP addr

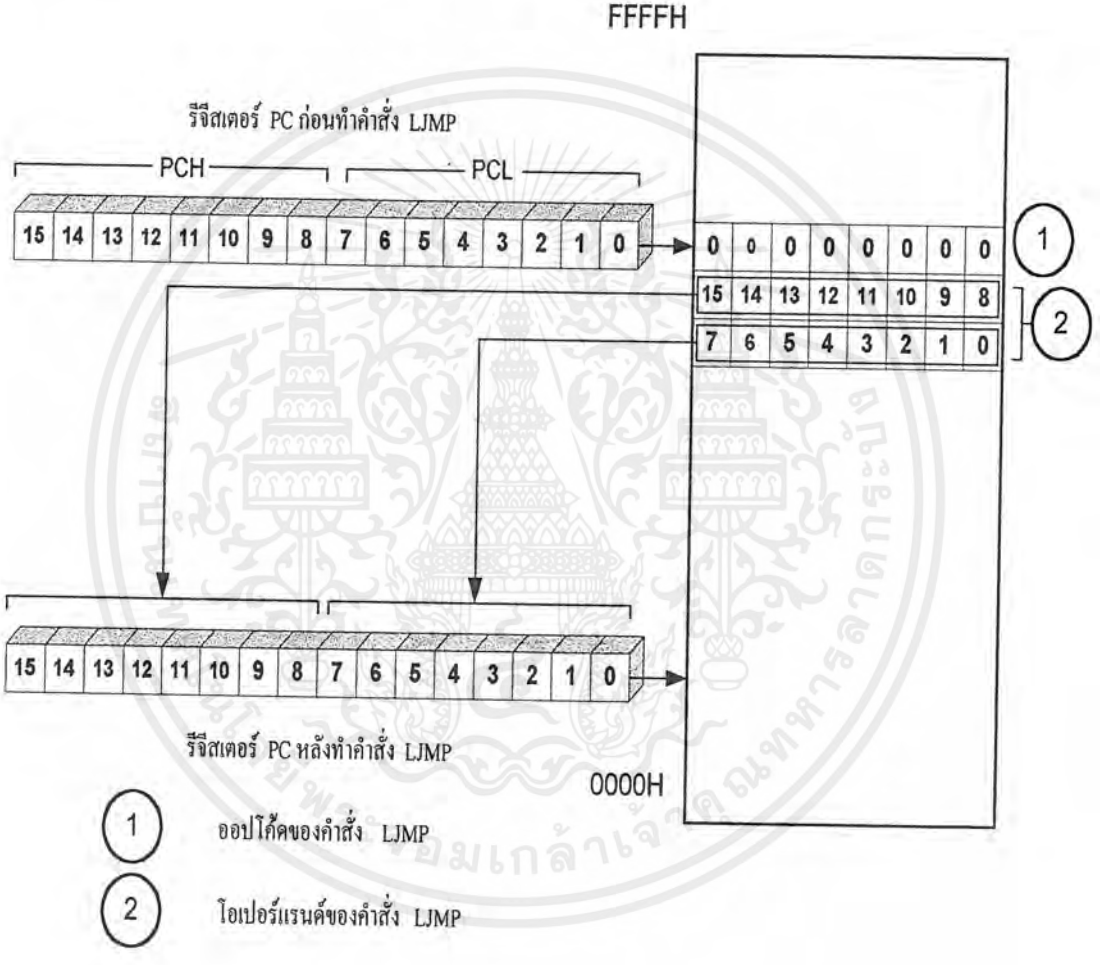
ในการใช้งานจริง ๆ เราสามารถเขียนโปรแกรมภาษาแอสแซมบลีโดยใช้คำสั่ง JMP addr ได้โดยเมื่อนำโปรแกรมภาษาแอสแซมบลีที่เขียนขึ้นมาไปทำการแอสแซมเบลอร์ (แปลคำสั่งภาษาแอสแซมบลีไปเป็นรหัสคำสั่งของ MCS – 51 โดยตรง) ตัวโปรแกรมแอสแซมเบลอร์จะทำการแปลรหัสนีโมนิกของคำสั่ง JMP ไปเป็นรหัสคำสั่งของคำสั่ง JMP ทั้งสามให้เอง ทั้งนี้โดยพิจารณาจากความเหมาะสมของแต่ละคำสั่ง แต่ผู้เขียนโปรแกรมก็สามารถระบุคำสั่งทั้งสามลงไปในโปรแกรมเองได้เช่นกัน ดังนั้น หากผู้เขียนโปรแกรมไม่สนใจรายละเอียดการทำงานของคำสั่งทั้งสามนี้ก็สามารถใช้คำสั่ง JMP addr หากผู้เขียนโปรแกรมไม่สนใจรายละเอียดการทำงานของคำสั่งทั้งสามนี้ก็สามารถใช้คำสั่ง JMP addr ได้โดยไม่จำเป็นต้องทราบถึงการทำงานของคำสั่งทั้งสาม แต่เพื่อให้การศึกษาเกี่ยวกับ MCS – 51 สมบูรณ์ เราจึงควรทราบรายละเอียดการทำงานของคำสั่งทั้งสามให้หมด เพื่อให้สามารถใช้งาน MCS – 51 ได้อย่างมีประสิทธิภาพยิ่งขึ้น

หมายเหตุ โปรแกรมแอสแซมเบลอร์ บางโปรแกรมไม่สามารถแทนคำสั่ง JMP addr ได้โดยตรง

คำสั่ง JMP ทั้งสามที่เราจะศึกษาต่อไปนี้ล้วนแต่มีจุดประสงค์ในการใช้งานเหมือนกัน คือ บังคับให้โปรแกรมไปทำงานที่ส่วนอื่น ของหน่วยความจำเหตุที่ต้องมีคำสั่งย่อยทั้งสามก็เพื่อให้การใช้งาน MCS – 51 มีประสิทธิภาพยิ่งขึ้น โดยคำสั่งทั้งสามจะมีขนาด ของคำสั่งไม่เท่ากัน แต่ความเร็วในการทำงานจะเท่ากัน การจะเลือกใช้คำสั่งหนึ่งคำสั่งใดขึ้นอยู่กับระยะห่างของหน่วยความจำที่โปรแกรมจะกระโดดไปทำงานกับตำแหน่งของตัวคำสั่ง JMP เอง รูปแบบของคำสั่งรายละเอียดการทำงานของคำสั่ง JMP ทั้งสามมีดังนี้

คำสั่ง LJMP add คำสั่งนี้มีขนาดของรหัสคำสั่งรวม 3 ไบต์ ตัวคำสั่งประกอบด้วยรหัสคำสั่งเอง 1 ไบต์ และโอเปอร์เรนต์อีก 2 ไบต์ โอเปอร์แรนต์ของคำสั่งซึ่งมีขนาด 16 ถึง 2 ไบต์ จะเป็นค่าของตำแหน่งหน่วยความจำที่จะบังคับให้โปรแกรมกระโดดไปทำงาน เมื่อ MCS – 51 กระทำด้วยคำสั่งค่าในรีจิสเตอร์ PC จะเพิ่มขึ้นอีกหนึ่ง จากนั้น MCS – 51 จะแทนที่ค่าในรีจิสเตอร์

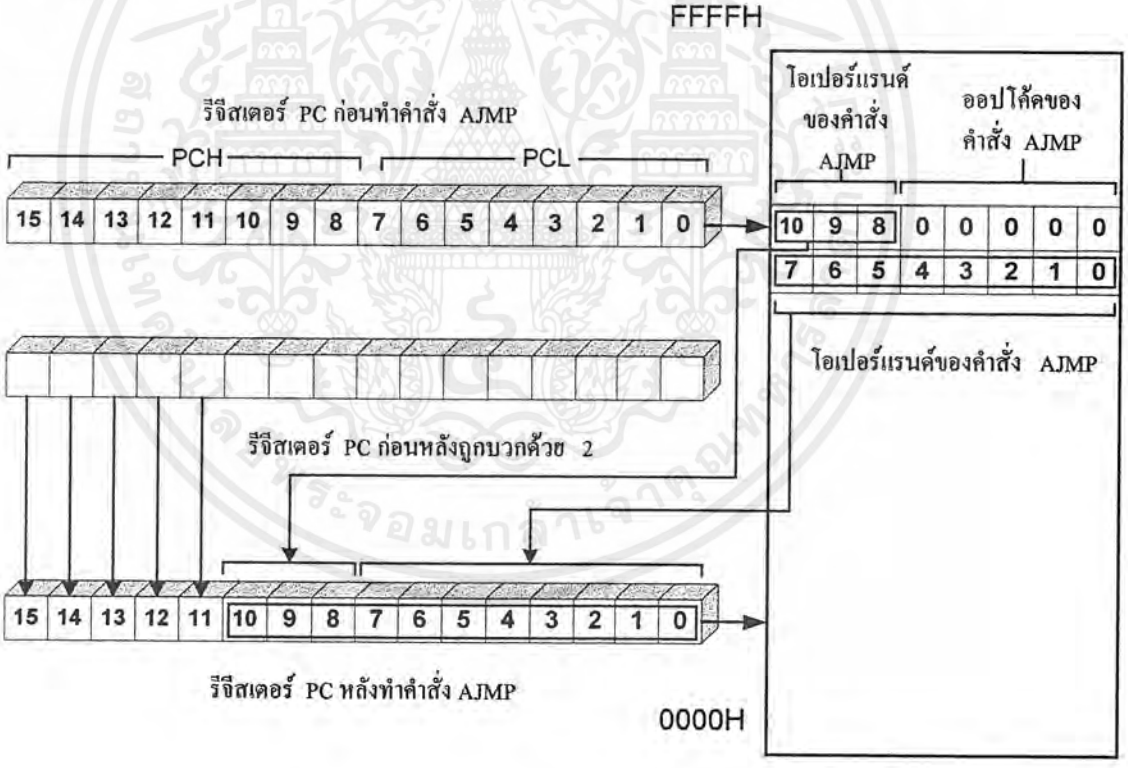
PC ด้วยค่าของโอเปอร์แรนด์ซึ่งมีขนาด 16 บิต ดังนั้นคำสั่งต่อไปนี้ที่ MCS – 51 จะทำงานคือคำสั่งซึ่งอยู่ในหน่วยความจำตำแหน่งที่ตรงกับค่าของรีจิสเตอร์ PC ซึ่งได้เปลี่ยนค่าไปโดยโอเปอร์แรนด์ซึ่งมีขนาด 16 บิต ดังนั้นคำสั่งต่อไปที่ MCS – 51 จะทำงานคือคำสั่งซึ่งอยู่ในหน่วยความจำตำแหน่งที่ตรงกับค่าของรีจิสเตอร์ PC ซึ่งได้เปลี่ยนค่าไปโดยโอเปอร์แรนด์ของคำสั่ง LJMP addr เรียบร้อยแล้วดังแสดงการทำงานในภาพที่ 4.8



ภาพที่ 4.9 แสดงการทำงานของคำสั่ง LJMP addr

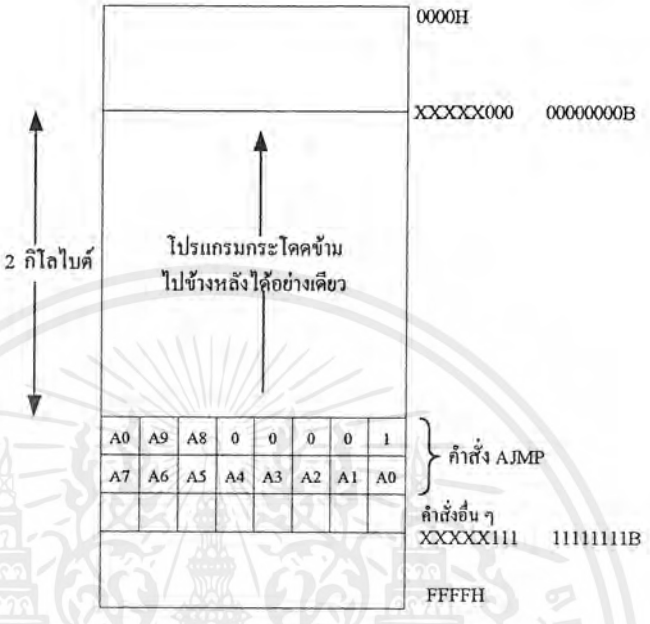
การที่รีจิสเตอร์ PC ถูกแทนที่ใหม่ด้วยโอเปอร์แรนด์ขนาด 16 บิต ทำให้ผู้เขียนโปรแกรมสามารบังคับให้โปรแกรมกระโดดไปทำงานบริเวณใดก็ได้ในหน่วยงานความจำส่วนที่เก็บโปรแกรม ซึ่งมีขนาดทั้งสิ้น 64 กิโลไบต์

คำสั่ง AJMP addr11 คำสั่งนี้มีขนาดของรหัสคำสั่งรวม 2 ไบต์ ตัวคำสั่งเองประกอบด้วยรหัสคำสั่งเอง 5 บิต รวมอยู่ในไบต์แรกของคำสั่ง (บิต 0 ถึง บิต 4) ส่วนบิตที่เหลือ 3 บิต ของไบต์แรกจะแทนด้วยค่าตำแหน่งหน่วยความจำบิต 8 ถึง บิต 10 ของตำแหน่งหน่วยความจำที่ต้องการย้ายไปทำงานภายหลังคำสั่งนี้ส่วนไบต์ที่สองของคำสั่งจะแทนด้วยค่าตำแหน่งหน่วยความจำบิต 0 ถึงบิต 7 ของหน่วยความจำที่ต้องการย้ายไปทำงานต่อจากคำสั่งนี้ ดังนั้นโอเปอร์แรนด์ของคำสั่ง AJMP จะเป็นค่าของตำแหน่งในหน่วยความจำจำนวน 11 บิต(บิต 0 ถึงบิต 10) โดยแยกกันอยู่ใน 2 ไบต์ ของคำสั่งเมื่อ MCS-51 กระทำคำสั่งAJMP ค่าในรีจิสเตอร์ PC จะเพิ่มขึ้นอีกสองจากนั้น MCS – 51 จะแทนที่ค่าของรีจิสเตอร์ PC ด้วยค่าตำแหน่งในหน่วยความจำเพียง 11 บิต (AO- A10) ส่วนตำแหน่งในหน่วยความจำบิต 11 ถึงบิต 15 จะยังคงมีค่าเดิมดังแสดงในรูปที่ จึงทำให้คำสั่ง AJMP สามารถบังคับให้โปรแกรมไปทำงานที่ส่วนอื่นของหน่วยความจำภายในขอบเขต 2 กิโลไบต์ (2^{11}= 2048) ซึ่งอยู่ติดกับตำแหน่งของคำสั่งดังแสดงให้เห็นในรูปที่ 4.10



รูปที่ 4.10 แสดงคำสั่ง AJMP addr

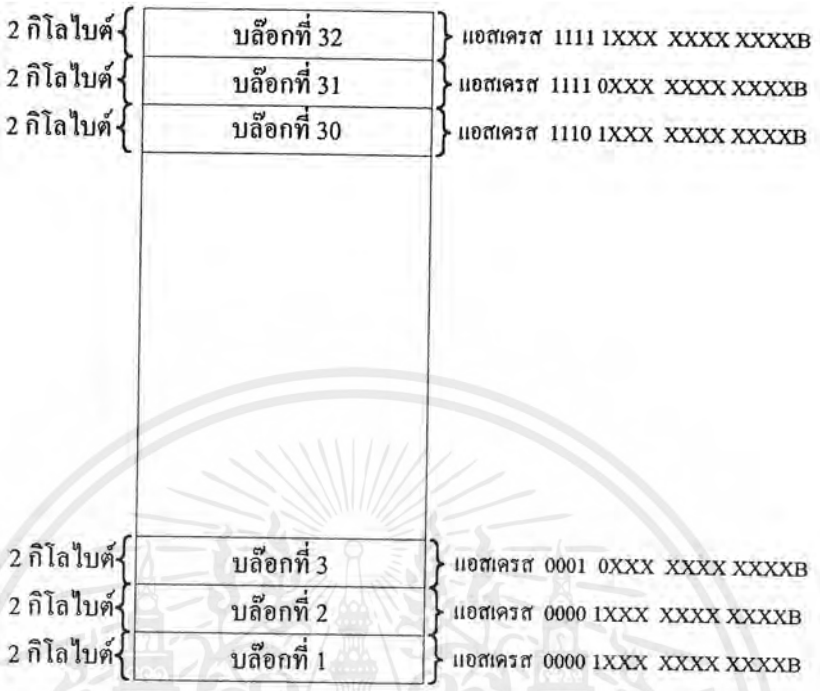
จากภาพที่ 4.10 จะแสดงให้เห็นตำแหน่งของคำสั่ง AJMP ซึ่งมีผลกระทบต่อขอบเขตการกระโดยข้ามไปทำงานที่ตำแหน่งอื่นในความจำภายในบริเวณ 2 กิโลไบต์ติดกับคำสั่ง



ภาพที่ 4.11 ขอบเขตในการกระโดดไปทำงานของคำสั่ง AJMP

ภาพที่ 4.11 (ต่อ) ขอบเขตในการกระโดดไปทำงานของคำสั่ง AJMP

หากเราแบ่งหน่วยความจำส่วนที่เก็บโปรแกรม ของ MCS – 51 ออกเป็นบล็อก ๆ โดยพิจารณาจาก 5 บิตบนของรีจิสเตอร์ PC จะทำให้สามารถแบ่งพื้นที่หน่วยความจำส่วนนี้ออกได้เป็น 32 บล็อกด้วยกัน ($2^5 = 32$) โดยเริ่มจากบล็อกแรกซึ่งมีค่า 5 บิตบนเป็น 0 หมด จนถึงบล็อกสุดท้ายที่มีค่า 5 บิต บนนี้เป็น1 หมด เพื่อความเข้าใจให้ดูรูปประกอบ ภาพที่ 4.12



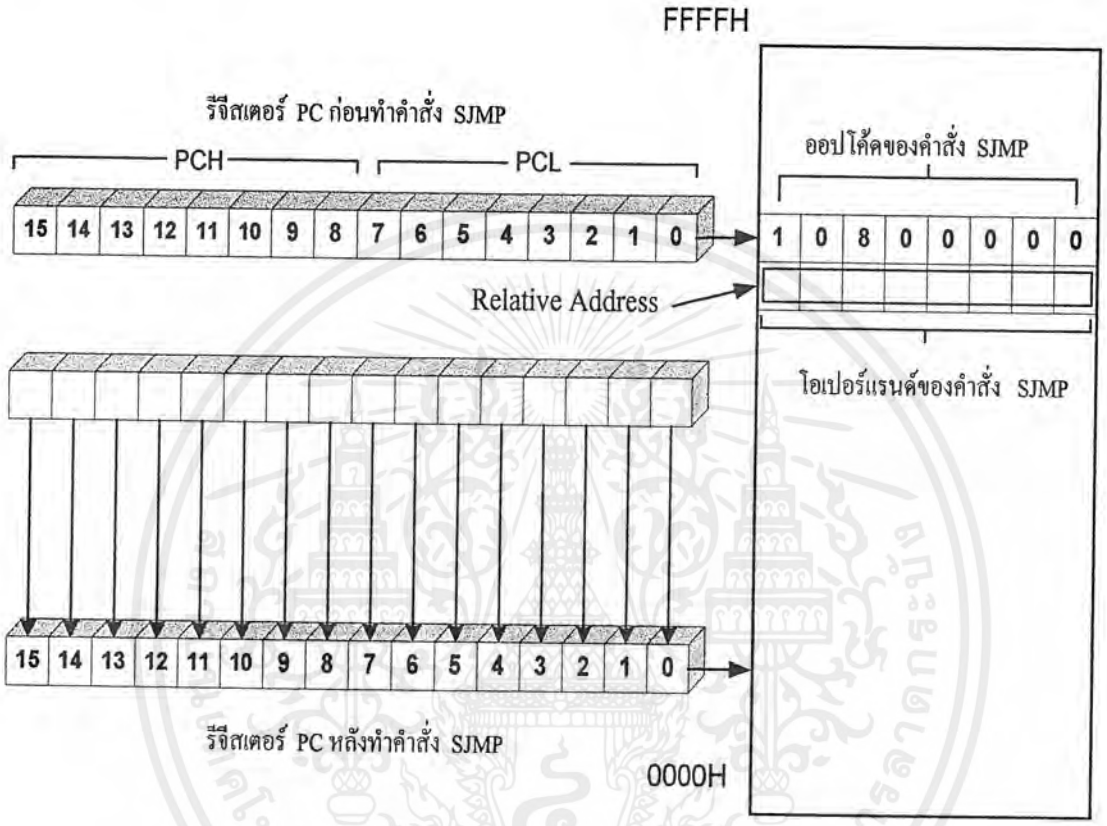
ภาพที่ 4.12 แสดงส่วนประกอบบล๊อกหน่วยความจำทั้ง 32 บล๊อก

คำสั่ง AJMP ในโปรแกรมซึ่งอยู่ในบล็อกใดบล็อกหนึ่งความจำเปลี่ยนค่าในรีจิสเตอร์ PC เพียง 11 บิต โดย 5 บิตที่เหลือมีค่าเหมือนเดิม นั่นคือ สามารถบังคับให้โปรแกรมกระโดดไปทำงานภายในบริเวณบล็อกใดบล็อกหนึ่งของหน่วยความจำเท่านั้น โดยหากคำสั่ง AJMP อยู่ในตำแหน่งที่ 1 หรือ 2 ในภาพที่ 4.11 โปรแกรมจะสามารถกระโดดไปข้างหน้าหรือถอยหลังได้เพียงอย่างเดียว โดยมีช่วงการกระโดดไปข้างหน้าและกระโดดถอยหลังไม่เท่ากัน ขึ้นอยู่กับตำแหน่งของคำสั่งในบล็อก นั้น ๆ

คำสั่ง SJMP rel address คำสั่งนี้มีขนาด 2 ไบต์ เช่นเดียวกับคำสั่ง AJMP addr 11 แต่ในคำสั่งนี้รหัสของคำสั่งจะมีขนาด 8 บิตอยู่ในไบต์แรกของคำสั่ง ส่วนโอเปอร์แรนด์มีขนาด 8 บิต เช่นกันแต่ว่าอยู่ในไบต์ที่สองตามหลังไบต์ที่เป็นรหัสคำสั่ง ค่าของโอเปอร์แรนด์ขนาด 8 บิต มีชื่อเรียกว่า relative address ซึ่งเป็นเลขฐาน 2 ที่มีบิต 7 เป็นตัวบอกเครื่องหมาย (มีค่าในช่วง –128 ถึง +127)

หลังจากกระทำคำสั่ง AJMP ค่าในรีจิสเตอร์ PC จะเพิ่มขึ้นอีก หนึ่งจากนั้น MCS – 51 จะบวกค่าในรีจิสเตอร์ PC ด้วยค่าโอเปอร์แรนด์ (relative address) แบบคิดเครื่องหมายแล้วนำผล

ลัพธ์ที่ได้กลับไปแทนที่รีจิสเตอร์ PC อีกครั้งดังนั้นรีจิสเตอร์ PC จะมีค่าใหม่ที่อยู่ในช่วง 128 ไบต์ ซึ่งอาจจะมากหรือน้อยกว่าค่าในรีจิสเตอร์ PC เดิม (อยู่ในช่วง 256 ไบต์ ติดกับตำแหน่งของคำสั่ง) ดังแสดงในรูปที่ 4.13



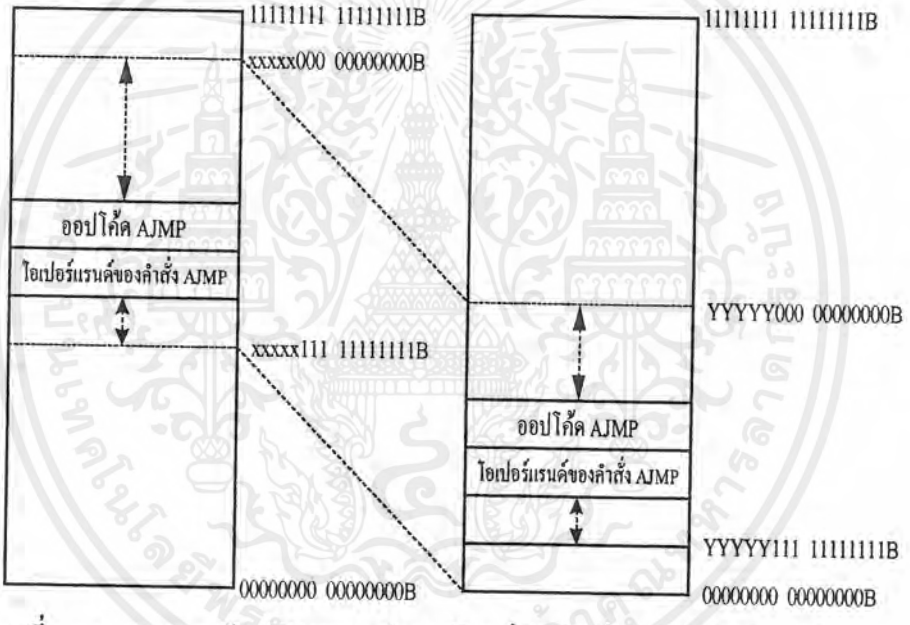
ภาพที่ 4.13 แสดงการทำงานของคำสั่ง SJMP

คำสั่ง SJMP นี้สามารถบังคับให้โปรแกรมกระโดดไปทำงานที่ส่วนอื่นในหน่วยความจำ ซึ่งมีขอบเขตในการกระโดยข้ามไปทำงานน้อยกว่าคำสั่ง LJMP และ AJMP ที่กล่าวมาแล้ว แต่คำสั่งนี้มีข้อดีเหนือกว่าคำสั่งอื่น ๆ ตรงที่ผู้เขียนโปรแกรมสามารถย้ายโปรแกรมที่มีคำสั่ง SJMP นี้ไปไว้ที่ใดก็ได้นั้นตำแหน่งของหน่วยความจำที่จะกระโดยข้างไปมาจากการคำนวณโดยนำค่าในรีจิสเตอร์ PC รวมกับค่าของโอเปอร์แรนด์ในคำสั่ง

คำสั่ง AJMP ถึงแม้จะสามารถบังคับให้โปรแกรมกระโดดไปทำงานที่อื่นได้ในบริเวณที่กว้างกว่าคำสั่ง SJMP แต่ในการย้ายโปรแกรมไปยังส่วนอื่นของหน่วยความจำจะกระทำได้ยากกว่า เพราะต้องแปลโปรแกรมแอสแซมบลีให้เป็นรหัสคำสั่งที่เป็นภาษาเครื่องใหม่ทั้งหมด ถึงอย่างไรก็

ตามโปรแกรมที่มีคำสั่ง ALMP ก็ยังสามารถย้ายไปไว้ที่อื่นในหน่วยความจำโดยไม่ต้องแปลโปรแกรมแอสแซมบลีใหม่ได้เช่นกัน เพียงแต่มีเงื่อนไขบางอย่างเพิ่มขึ้นมาเท่านั้น ดังจะได้กล่าวต่อไป

เนื่องจากตำแหน่ง ของคำสั่ง AJMP addr 11 มีผลต่อการกระโดดไปทำงานของโปรแกรม ดังได้แสดงในรูปที่ ซึ่งจะเห็นว่าตำแหน่งของคำสั่ง AJMP มีความสำคัญมาก หากต้องการย้ายโปรแกรมที่มีคำสั่ง AJMP นี้ไปไว้ในส่วนอื่น ๆ ของหน่วยความจำจะสามารถกระทำได้โดยการต้องย้ายไปทั้งบล็อกเพื่อให้ระยะห่างระหว่างต้นบล็อกหรือท้ายบล็อกของตัวคำสั่งเหมือนเดิมทั้งในบล็อกเดิมและในบล็อกใหม่ มิฉะนั้นโปรแกรมจะกระโดดไปทำงานในตำแหน่งที่ไม่ถูกต้อง ดังแสดงให้เห็นในรูปที่



ภาพที่ 4.14 แสดงการย้ายโปรแกรมเป็นบล๊อกเพื่อให้คำสั่งยังใช้ได้เหมือนเดิม

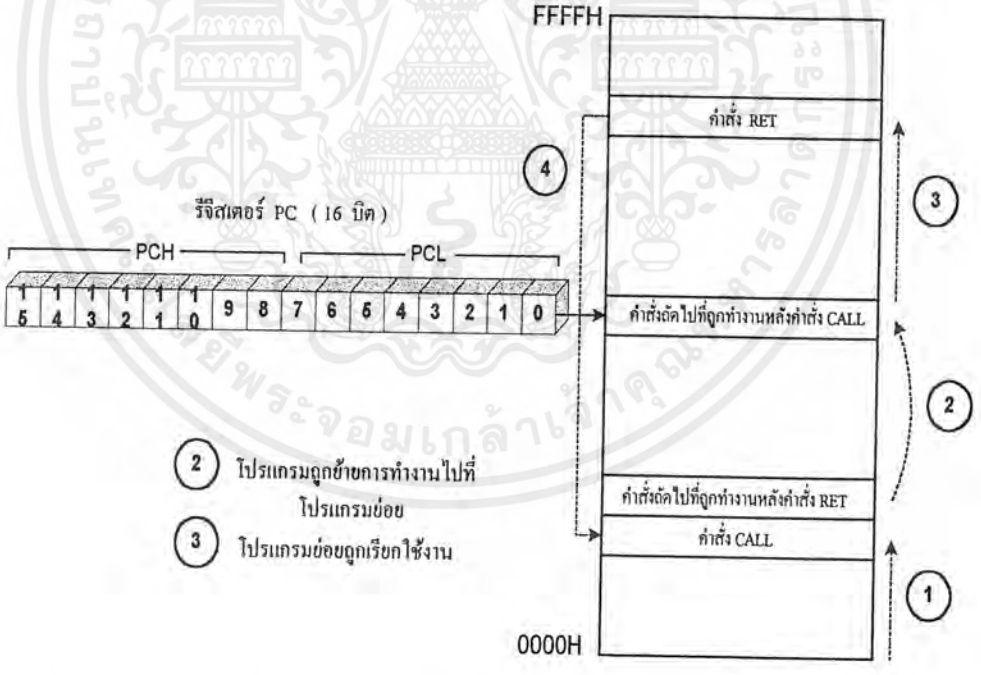
คำสั่ง JMP ในกลุ่มนี้ยังมีคำสั่ง JMP @ A + DPTR ซึ่งจะนำค่าในรีจิสเตอร์ A มารวมกับค่าในรีจิเตอร์ DPTR โดยไม่คิดเครื่องหมายของค่าในรีจิสเตอร์ A จากนั้นนำผลลัพธ์ที่ได้ไปไว้ในรีจิสเตอร์ PC จึงทำให้เราสามารถบังคับให้โปรแกรมไปทำงานบริเวณใดก็ได้ในหน่วยความจำขึ้นอยู่กับค่าในรีจิสเตอร์ DPTR และ A เท่านั้น

คำสั่ง CALL Addr และ RET คำสั่ง CALL Addr คำสั่งนี้มักใช้ควบคู่ไปกับคำสั่ง RET ดังนั้น จะกล่าวถึงรายละเอียดของคำสั่งทั้งสองพร้อมกันเลย

คำสั่ง CALL Addr มีไว้เพื่อควบคุมให้โปรแกรมหลักย้ายการทำงานไปที่โปรแกรมย่อยที่อยู่บริเวณใดบริเวณหนึ่งของหน่วยความจำ และเมื่อโปรแกรมย่อยทำงานเสร็จสิ้นแล้วก็สามารถคืนการทำงานมายังโปรแกรมหลักได้ด้วยการใช้คำสั่ง RET นั่นคือ คำสั่ง CALL และ RET นี้มีไว้เพื่อย้ายการทำงานไปที่อื่นชั่วคราว ซึ่งต่างจากคำสั่ง JMP ที่จะกระโดดไปทำงานต่อในตำแหน่งอื่นของหน่วยความจำเลย การทำงานของคำสั่ง CALL และ RET แสดงดังในรูปที่ 4.15

เนื่องจากรีจิสเตอร์ PC มีขนาด 16 บิต (2 ไบต์) ดังนั้นการทำคำสั่ง PUSH ,. POP โดยคำสั่ง CALL และ RET จึงต้องกระทำถึง 2 ครั้ง โดยครั้งแรกจะทำการ PUSH ค่าในรีจิสเตอร์ PC ไบต์ต่ำ(บิต 0 ถึง บิต 7)ก่อนแล้วจึง PUSH ค่าในรีจิสเตอร์ PC ไบต์ถัดไป

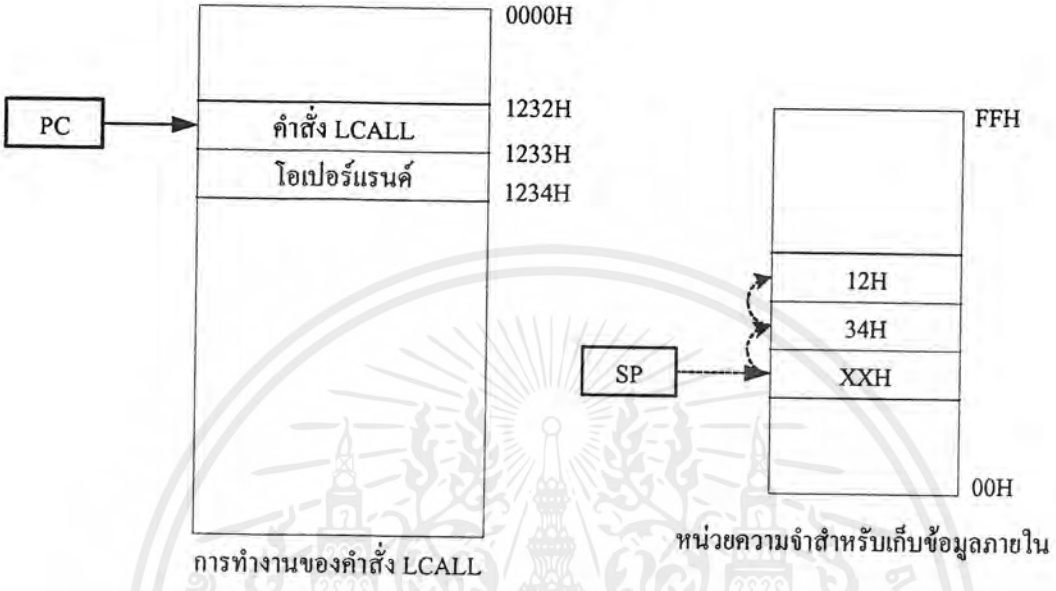
หากในโปรแกรมย่อยไม่มีคำสั่งRET จะทำให้ MCS – 51 ไม่สามารถกลับมาทำงานต่อในโปรแกรมหลักได้ เท่ากับเป็นการบังคับให้โปรแกรมกระโดดไปทำงานที่อื่นเลย ดังนั้น โปรแกรมย่อยทุกโปรแกรมจึงต้องมีคำสั่ง RET ปิดท้ายไว้เสมอ เพื่อให้ MCS – 51 ทราบว่าทำงานในโปรแกรมย่อยเสร็จแล้ว



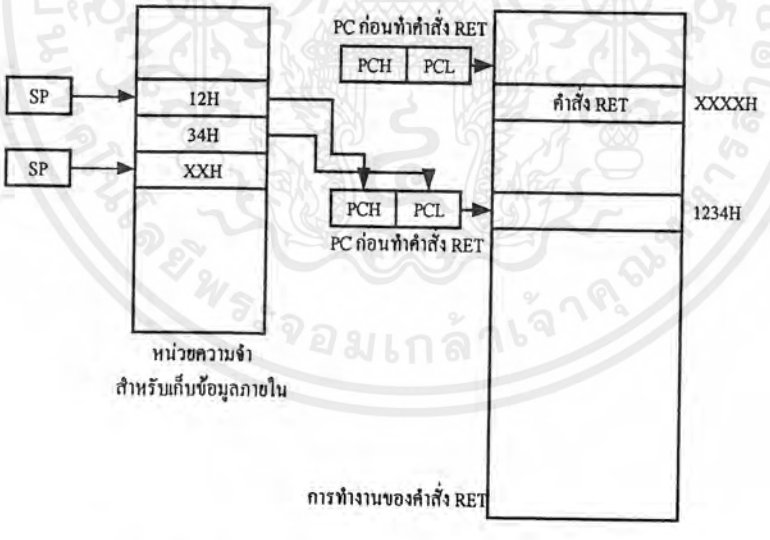
ภาพที่ 4.15 แสดงการทำงานของคำสั่ง CALL และ RET

หากในโปรแกรมย่อยไม่มีคำสั่งRET จะทำให้ MCS – 51 ไม่สามารถกลับมาทำงานต่อในโปรแกรมหลักได้ เท่ากับเป็นการบังคับให้โปรแกรมกระโดดไปทำงานที่อื่นเลย ดังนั้น โปรแกรม

ย่อยทุกโปรแกรมจึงต้องมีคำสั่ง RET ปิดท้ายไว้เสมอ เพื่อให้ MCS – 51 ทราบว่าทำงานในโปรแกรมย่อยเสร็จแล้ว



ภาพที่ 4.16 แสดงลำดับการทำงานของคำสั่ง CALL และ RET



ภาพที่ 4.16 (ต่อ) แสดงลำดับการทำงานของคำสั่ง CALL และ RET

คำสั่ง CALL ใน MCS – 51 มีถึง 2 คำสั่งด้วยกัน คือ

- LCALL addr 16
- ACALL addr 11

การใช้คำสั่ง CALL addr ตัวโปรแกรมแอสเซมเบลอร์ที่ดีจะทราบเองว่าสมควรแทนคำสั่งนี้ด้วยคำสั่ง CALL ชนิดใดที่เหมาะสม คำสั่ง CALL ทั้งสองมีรายละเอียดที่แตกต่างกัน ดังนี้

- LCALL addr 16

คำสั่งนี้มีขนาด 3 ไบต์ ไบต์แรกเป็นรหัสของคำสั่ง ส่วนไบต์ถัดไปอีกจะเป็นค่าตำแหน่งในหน่วยความจำซึ่ง MCS – 51 จะย้ายไปทำงานชั่วคราว คำสั่งนี้จะทำการ PUSH ค่าของหน่วยความจำในตำแหน่งถัดจากตัวคำสั่งไปไว้ในสแตก จากนั้นจึงแทนที่ค่าในรีจิสเตอร์ PC ด้วย ค่าของโอเปอร์แรนด์ทั้ง 16 บิต ดังแสดงการทำงานในรูปที่ 4.16

- ACALL addr 11

คำสั่งนี้มีขนาด 2 ไบต์แรกจะประกอบด้วยรหัสคำสั่งเพียง 5 บิต (10001) อยู่ในบิต 00 ถึง บิต 4 ของไบต์แรก บิตที่เหลือจะเป็นค่าตำแหน่งของหน่วยความจำที่ต้องการย้ายไปทำงานชั่วคราวเฉพาะบิต 10 ถึง บิต 8 ส่วนในไบต์ถัดไปจะเป็นค่าตำแหน่งของหน่วยความจำที่ต้องการย้ายไปทำงานชั่วคราวเฉพาะบิต 7 ถึง บิต 0 นั่นคือมีการแทนที่ค่าในรีจิสเตอร์ PC หลังกระทำคำสั่ง CALL เหมือนคำสั่ง AJMP addr 11 ดังนั้นโปรแกรมย่อยจึงต้องอยู่ในขอบเขตบริเวณ 2 กิโลไบต์ติดกับคำสั่ง ในการทำงานจะมีการกระทำคำสั่ง PUSH 2 ครั้ง เช่นเดียวกันกับคำสั่ง LCALL ทั้งนี้เพื่อให้สามารถนำค่าตำแหน่งถัดไปของคำสั่งมาไว้ในรีจิสเตอร์ PC เพื่อกลับมาทำงานยังโปรแกรมหลักต่อไป

คำสั่ง NOP คำสั่งนี้ซีพียูจะไม่ทำงานใด ๆ ทั้งสิ้น แต่จะปล่อยให้เวลาผ่านไป 1 แมชชีนไซเกิล พร้อมทั้งเปลี่ยนค่าในรีจิสเตอร์ PC ให้มาชี้ยังคำสั่งที่อยู่ในตำแหน่งถัดไปของคำสั่ง NOP นั่นคือเมื่อ MCS – 51 ทำคำสั่งใด ๆ มาจนถึงคำสั่ง NOP ก็จะข้ามไปทำงานที่คำสั่งซึ่งอยู่ถัดไปจากคำสั่ง NOP โดยปล่อยให้เวลาผ่านไป 1 แมชชินไซเกิลก่อนกระทำคำสั่งถัดไป

4.4.2 กลุ่มคำสั่งควบคุมโปรแกรมแบบมีเงื่อนไข

ในหัวข้อที่ผ่านมาเราได้กล่าวถึงกลุ่มคำสั่งในการควบคุมลำดับการทำงานของโปรแกรม ซึ่งได้ในการควบคุมให้โปรแกรมย้ายการทำงานไปยังส่วนอื่นโดยไม่มีเงื่อนไข ซึ่งมีข้อจำกัดในการเขียนโปรแกรมอยู่บ้าง เพราะโปรแกรมที่มีแต่คำสั่งเหล่านี้ไม่สามารถตัดสินใจโดยตัวเองได้ ดังนั้น

ในหัวข้อต่อไปนี้จะกล่าวถึงกลุ่มคำสั่งที่ใช้ในการควบคุมลำดับการทำงานของโปรแกรมที่มีความสามารถมากกว่าในกลุ่มที่ผ่านมา คำสั่งในกลุ่มนี้มีด้วยกันทั้งสิ้น 8 คำสั่ง คือ

JZ rel	jump if zero
JNZ rel	jump if not zero
CJNE A,direct rel	compare and if not equal
CJNE A,#data rel	compare and jump it not equal
CJNE Rn,#data rel	compare and jump it not equal
CJNE @ Ri,#data rel	compare and jump if not equal
DJNZ Rn, rel	decrement and jump if not zero
DJNZ direct ,rel	decrement and jump if not zero

จะเห็นว่าโอเปอร์แรนด์ของคำสั่งในกลุ่มนี้เป็นค่า relative address (ดิดเครื่องหมาย)ทั้งหมด ดังนั้น คำสั่งในกลุ่มนี้ทุกคำสั่งจะสามารถบังคับให้โปรแกรมทำงานบริเวณอื่นได้ในหน่วยความจำแบบมีเงื่อนไขภายในขอบเขตที่ห่างจากตัวคำสั่งได้เพียง 256 ไบต์เท่านั้น รายละเอียดแต่ละคำสั่งในกลุ่มนี้มีดังนี้

- คำสั่ง JZ rel จะตรวจสอบเงื่อนไขเฉพาะกับรีจิสเตอร์ A เท่านั้น หากรจิสเตอร์ A มีค่าเป็นศูนย์ คำสั่งนี้จะควบคุมให้โปรแกรมข้ามไปทำงานที่ตำแหน่งซึ่งได้จากค่าในรีจิสเตอร์ PC รวมกับค่า relative address ของโอเปอร์แรนด์ แต่หากค่าของรีจิสเตอร์ไม่เท่ากับศูนย์MCS – 51 จะกระทำคำสั่งซึ่งอยู่ถัดไปจากคำสั่งดังต่อไป
- คำสั่ง JNZ rel จะตรวจสอบรีจิตเตอร์ A เช่นเดียวกับคำสั่ง JZ rel แต่เงื่อนไขที่ใช้เป็นตัวตัดสินจะแตกต่างกัน นั่นคือ หากรีจิสเตอร์ A มีค่าเป็นศูนย์ MCS – 51 จะกระทำคำสั่งถัดไป แต่ถ้าค่าในรีจิสเตอร์ A ไม่เป็นศูนย์ โปรแกรมจะถูกบังคับให้ข้ามไปทำงานที่ตำแหน่งซึ่งได้จากค่าในรีจิสเตอร์ PC รวมกับค่า relative address ของโอเปอร์แรนด์
- คำสั่ง CJNE 4 คำสั่งต้องการข้อมูลหรือโอเปอร์แรนด์จำนวน 3 ตัว คำสั่งนี้ใช้ควบคุมลำดับการทำงานของโปรแกรมโดยมีเงื่อนไข คือ จะนำเองค่าของโอเปอร์แรนด์ทั้งสองของคำสั่งมาลบกัน ดังนี้
- โอเปอร์เรนต์ตัวแรกมากกว่าโอเปอร์เรนด์ตัวที่สอง carry flag ถูกเคลียร์
- โอเปอร์เรนตัวแรกน้อยกว่าโอเปอร์เรนต์ตัวที่สอง carry flag ถูกเซต

หากค่าของโอเปอร์เรนด์ตัวแรกมีค่าไม่เท่ากับโอเปอร์แรนด์ตัวที่สอง คำสั่งที่จะบังคับให้ MCS – 51 ข้ามไปทำงานที่ตำแหน่งได้จากค่าในรีจิสเตอร์ PC รวมกับค่า relative adress ที่ได้จากโอเปอร์แรนด์ตัวที่สาม แต่หากค่าของโอเปอร์แรนด์สองตัวแรกมีค่ากับเท่าโอเปอร์แรนด์ตัวที่สอง MCS – 51 จะข้ามคำสั่งนี้ไปกระทำคำสั่งถัดไปทันที

คำสั่งในการควบคุมลำดับการทำงานของโปรแกรมแบบมีเงื่อนไขยังมีส่วนหนึ่งที่จัดไว้ในกลุ่มคำสั่งประมวลผลแบบบูลีน เหตุที่ต้องแยกคำสั่งดังนี้เพราะกลุ่มคำสั่งสำหรับการประมวลผบแบบบูลีนมีการทำงานพิเศษที่ต่างจากไมโครโปรเซสเซอร์ทั่วไป โดยคำสั่งในกลุ่มประมวลผลแบบบูลีนจะมีกลุ่มคำสั่งย่อยที่ใช้ควบคุมลำดับการทำงานของโปรแกรมโดยตรวจสอบเงื่อนไขจากข้อมูลเพียงบิตเดียว รายละเอียดของกลุ่มคำสั่ง ประมวลผลแบบบูลีนจะได้ศึกษาต่อจากบทนี้ต่อไป

4.5 กลุ่มคำสั่งประมวลผลแบบบูลีน

กลุ่มีคำสั่งประมวลผลแบบบูลีนที่จะศึกษาต่อไปในบทนี้เป็นกลุ่มคำสั่งพิเศษที่มีใน MCS – 51 จุดประสงค์ของคำสั่งในกลุ่มนี้คือ อำนวยความสะดวกในการเขียนโปรแกรมที่จำเป็นต้องเกี่ยวข้องกับข้อมูลระดับบิต เช่น การตรวจสอบสถานะของพอร์ตแต่ละบิต การปฏิบัติคำสั่งทางตรรกศาสตร์ระดับบิตการเคลื่อนย้ายข้อมูลระดับบิต การเซตหรือเคลียร์บิต การควบคุมการทำงานของโปรแกรมโดยตัดสินเงื่อนไขจากสถานะบิต

เมื่อเราได้ศึกษาลักษณะของข้อมูลขนาด 1 บิตมาแล้ว ต่อไปจะเป็นการศึกษาคำสั่งของกลุ่มคำสั่งประมวลผลแบบบูลีนโดยแยกตามการทำงาน ดังนี้

4.5.1 กลุ่มคำสั่งควบคุมสถานะบิต

คำสั่งในกลุ่มนี้ใช้สำหรับควบคุมสถานะของบิตให้มีค่าตามที่เราต้องการ คำสั่งทั้งหมดในการควบคุมสถานะบิตมีด้วยกัน 6 คำสั่ง คือ

```
CLR   C
CLR   bit
SETB  C
SETB  bit
CPL   C
CPL   bit
```

ตัวอักษร B ในคำสั่ง SETB มีไว้เพื่อแยกความแตกต่างระหว่างคำสั่ง SET ซึ่งใช้ในโปรแกรมแอสแซมเบลอร์

ในการเขียนโปรแกรมภาษาแอสเซมบลีสำหรับ MCS – 51 โอเปอร์แรนด์ของคำสั่งที่เป็นหน่วยความจำขนาด 1 บิต สามารถระบุในตัวคำสั่งได้หลายวิธีดังนี้

- ระบุโดยค่าตำแหน่งระหว่าง 0 – 255 หรือ OOH – FFH ค่าตัวเลขที่ใช้อาจจะเป็นค่าตัวเลขที่ระบุในคำสั่งเลย หรือ เป็นนิพจน์ทางคณิตศาสตร์ที่ให้ค่าอยู่ในช่วง 0 – 255
- ในกรณีที่ของรีจิสเตอร์ซึ่งแต่ละบิต SCON.3 คือ บิตที่ 3 ของรีจิสเตอร์ SCON
- ในกรณีของรีจิสเตอร์ ซึ่งแต่ละบิตมีชื่อเฉพาะอยู่แล้ว โปรแกรมที่แอสเซมเบลอร์จะทราบเองว่าบิตที่อ้างด้วยชื่อนี้หมายถึงบิตอะไร เช่น บิต TI, RI หมายถึงบิตข้อมูลในรีจิสเตอร์ ใช้งานเฉพาะ SCON
-

นอกจากจะสามารถระบุตำแหน่งของหน่วยความจำขนาด 1 บิต ที่จะมากระทำการในคำสั่งด้วยวิธีการดังกล่าวข้างต้นแล้ว เรายังสามารถแทนชื่อของตำแหน่งหน่วยความจำได้ด้วยชุดของตัวอักษรได้แต่จะต้องใช้คำสั่ง SET หรือ EQU ในโปรแกรมแอสเซมเบลอร์ก่อน เช่น

```
SET  Label : 20H หรือ
Label EQU   20H
```

ในโปรแกรมแอสเซมเบลอร์บางโปรแกรมอาจไม่ได้ใช้คำสั่ง SETแต่จะใช้คำสั่ง EQU แทนดังนั้นผู้เขียนโปรแกรมจะต้องทราบเองว่า โปรแกรมแอสเซมเบลอร์ที่ใช้งานอยู่มีคำสั่งในการใช้งานอย่างไร

ตัวอย่างของการเข้าถึงข้อมูลขนาด 1 บิตจะแสดงให้เห็นทุกวิธี โดยสมมุติว่าเราต้องการเคลียร์บิตที่ 5 ของรีจิสเตอร์ PSW ซึ่งมีชื่อว่า บิต FO ดังนั้นคำสั่งต่อไปนี้จะให้ผลเช่นเดียวกัน

```
- CLR OD5H          ;ABSOLUTE ADDRESSING
- CLR FO            ;PRE – DEFINDED ASSEMBLER SYMBOL
- CLR PSW.5         ;DOT OPERATOR
```

หรือ อาจแทนบิตนี้ด้วยชื่อใหม่ว่า บิต STATUS ดังนี้

```
- STATUS BIT PSW.5  ; ตั้งชื่อบิต PSW.5 เป็น STATUS โดยใช้ชื่อคำสั่ง BIT หรือ
- STATUS EQUP SW.5  ;ตั้งชื่อบิต PSW. 5 เป็น STATUS โดยใช้คำสั่ง EQU
- CLR STATUR        ;เคลียร์บิตชื่อ STATUS ซึ่งเป็นบิต PSW.5
```

4.5.2 กลุ่มคำสั่งเคลื่อนย้ายข้อมูล คำสั่งกลุ่มนี้เป็นกลุ่มที่สองในกลุ่มคำสั่งประมวลผลแบบบูลีนจุดประสงค์ของคำสั่งมีไว้เพื่อเคลีท่อนย้ายข้อมูลขนาด 1 บิต จากที่หนึ่งไปยังอีกที่หนึ่ง โดยค่าข้อมูลที่ต้นทางยังคงมีค่าเดิม กลุ่มคำสั่งที่เคลื่อนย้ายข้อมูลในการประมวลผลแบบบูลีนต่างจากกลุ่มคำสั่งประมวลผลอื่น ๆ ที่เราได้ศึกษาผ่านมาแล้ว เพราะว่าข้อมูลที่เคลื่อนย้ายมีขนาดเพียง 1 บิต เท่านั้น ข้อจำกัดของคำสั่งอยู่ที่การเคลื่อนย้ายต้องใช้บิต C ในรีจิสเตอร์ PSW ทุกครั้ง นั่นคือ เราไม่สามารถย้ายข้อมูลระหว่างสองบิตใด ๆ โดยตรงได้ หากต้องการเคลื่อนย้ายบิตระหว่างหน่วยความจำ จำเป็นจะต้องกระทำผ่านบิต C เสมอ คำสั่งในกลุ่มนี้มีเพียงสองคำสั่งเท่านั้น คือ

- MOV C,bit
- MOV bit,c

4.5.3 กลุ่มคำสั่งทางตรรกศาสตร์ คำสั่งทางตรรกศาสตร์ในการประมวลผลแบบบูลีนจะกระทำการกับข้อมูลขนาด 1 บิต คำสั่งในกลุ่มนี้ทั้งหมดมีเพียง 4 คำสั่งเท่านั้น คือ

- ANL C,bit
- ANL N,/bit
- ORL C,bit
- ORL C,/bit

จะเห็นว่ามีเพียงคำสั่ง ANL และ ORL เท่านั้น ส่วนคำสั่ง XRL จะไม่มีกลุ่มนี้ซึ่งแตกต่างจากกลุ่มคำสั่งทางตรรกศาสตร์โดยทั่วไปที่ได้ศึกษาผ่านมาแล้ว

คำสั่ง ANL และ ORLที่แสดงข้างต้นมีด้วยกันอย่างละสองคำสั่ง สิ่งที่แตกต่างกันในคำสั่งทั้งสอง คือ การมีหรือไม่มีเครื่องหมาย" / " นำหน้าโอเปอร์แรนด์ตัวที่สอง โอเปอร์แรนด์ตัวที่สองของคำสั่งจะเป็นหน่วยความจำขนาด 1 บิต ตำแหน่งใดก็ได้ภายใน MCS – 51 ความหมายของเครื่องหมาย " / " คือ ระบุให้ซีพียูกระทำการตรรกศาสตร์กับค่าคอมพลีเมนต์ของโอเปอร์แรนด์ตัวที่สอง โดยหลังกระทำคำสั่งค่าข้อมูลในโอเปอร์แรนด์ตัวหลังจะไม่ถูกเปลี่ยนแปลงไป การที่ผู้ใช้สามารถระบุให้กระทำการกับข้อมูลที่อยู่ในรูปคอมพลีเมนต์นี้ ทำให้การเขียนโปรแกรมมีความสะดวกและกะทัดรัดมากขึ้น เนื่องจากไม่ต้องเปลี่ยนข้อมูลเป็นตรงกันข้ามก่อนแล้วจึงกระทำการคำสั่งทางตรรกศาสตร์

4.5.4 กลุ่มคำสั่งควบคุมโปรแกรมที่ขึ้นกับสถานะของบิต คำสั่งในกลุ่มนี้ใช้ในการควบคุมลำดับการทำงานของโปรแกรมโดยตรวจสอบเงื่อนไขของสถานะของบิตที่ระบุโดยโอเปอร์แรนด์ ในการทำงานคำสั่งกลุ่มนี้ MCS – 51 จะทำการตรวจสอบสถานะของบิตก่อน หากเงื่อนไขถูกต้อง

ตามคำสั่ง ค่าในรีจิสเตอร์ PC จะถูกเปลี่ยนแปลงไปเพื่อย้ายการทำงานไปยังส่วนอื่นของโปรแกรม คำสั่งในกลุ่มนี้มีด้วยกัน 2 ประเภท คือ คำสั่งที่ควบคุมลำดับของโปรแกรมโดยตรวจสอบสถานะของบิต C ในรีจิสเตอร์ PSW และคำสั่งที่ควบคุมลำดับของโปรแกรมโดยตรวจสอบสถานะของบิต C ในรีจิสเตอร์ PSW และคำสั่งที่ตรวจสอบสถานะของบิตใด ๆ ในหน่วยความจำขนาด 1 บิตที่กล่าวมาแล้ว คำสั่งทั้งหมดในกลุ่มนี้ มีดังนี้

- JC rel
- JNC rel
- JB bit,rel
- JNB bit,rel
- JBC bit,rel

คำสั่งทั้งหมดดังกล่าวใช้วิธีการเพิ่มค่าหรือลดค่าในรีจิสเตอร์ PC โดยการบวกค่าในโอเปอร์แรนด์ซึ่งเป็นเลขฐานสองที่คิดเครื่องหมายเข้ากับรีจิสเตอร์ PC เหมือนคำสั่ง SJMP ที่ได้ศึกษาผ่านมาแล้ว

คำสั่งที่ตรวจสอบสถานะของบิต C และหน่วยความจำขนาด 1 บิต ได้อำนวยความสะดวกให้ผู้เขียนโปรแกรมโดยสามารถใช้เงื่อนไขได้ทั้งสองอย่าง (ตรวจสอบเงื่อนไขได้ทั้งเป็นศูนย์ และหนึ่ง) ส่วนคำสั่งสุดท้ายจะตรวจสอบหน่วยความจำขนาด 1 บิต หากเงื่อนไขเป็นจริง (บิตที่อ้างโดยโอเปอร์แรนด์เป็น 1) ค่าในรีจิสเตอร์ PC จะถูกเปลี่ยนไปโดยค่า relative address และหน่วยความจำขนาด 1 บิตตำแหน่งก็จะถูกเคลียร์ด้วย

4.6 คำสั่งที่มีผลต่อการทำงานของกลุ่มคำสั่งประมวลผลแบบบูลีน

เนื่องจากการทำงานของกลุ่มคำสั่งประมวลผลแบบบูลีนจะมีการใช้บิต C ในรีจิสเตอร์ PSW เป็นเสมือนแอกคิวมูเลเตอร์ (accumulator) ขนาด 1 บิต ดังนั้น คำสั่งอื่นที่มีผลต่อบิตนี้จะส่งผลให้การทำงานของกลุ่มคำสั่งประมวลผลแบบบูลีนทำงานผิดพลาดได้ เช่น กลุ่มคำสั่งทางคณิตศาสตร์ตรรกศาสตร์ ซึ่งมีการเซตหรือเคลียร์บิตนี้หลังจากการกระทำคำสั่ง เพื่อป้องกันข้อผิดพลาดจากคำสั่งอื่น ๆ ที่มีผลกระทบต่อบิต C เราจึงควรที่จะเคลียร์บิต C ก่อนกระทำคำสั่งทุกครั้ง หรือหากต้องการใช้ค่าในบิต C อีกครั้งก็ควรใช้คำสั่ง CALL เพื่อเก็บค่ารีจิสเตอร์ PSW ไว้ใน สแตกเสียก่อน คำสั่งทั้งหมดที่มีผลกระทบต่อบิต C

บทที่ 5
การทดลองใช้งาน

การทำงานของเริ่มต้นโดยการโหลด Intel Stard File เข้ามเก็บในหน่วยความจำจำลองของโปรแกรมพร้อมกับทำการล้างข้อมูลที่มีอยู่ในหน่วยความจำเดิม จากนั้นโปรแกรมจะทำการถอดรหัสเพื่อตรวจสอบค่าแอสเดรสเริ่มต้นของคำสั่ง จากนั้นโปรแกรมเริ่มทำงานตามรหัสของคำสั่งซึ่งภายในโปรแกรมได้มีเงื่อนไขการทำงานของกลุ่มคำสั่งต่าง ๆ ตามหลักการทำงานของ MCS-51 หลังจากทำงานตามเงื่อนไขของคำสั่งจะมีการตรวจสอบสภาวะต่างของรีจีสเตอร์และแฟลกต่าง ๆ ดังเช่น โปรแกรมย่อยดังตัวอย่างต่อไปนี้

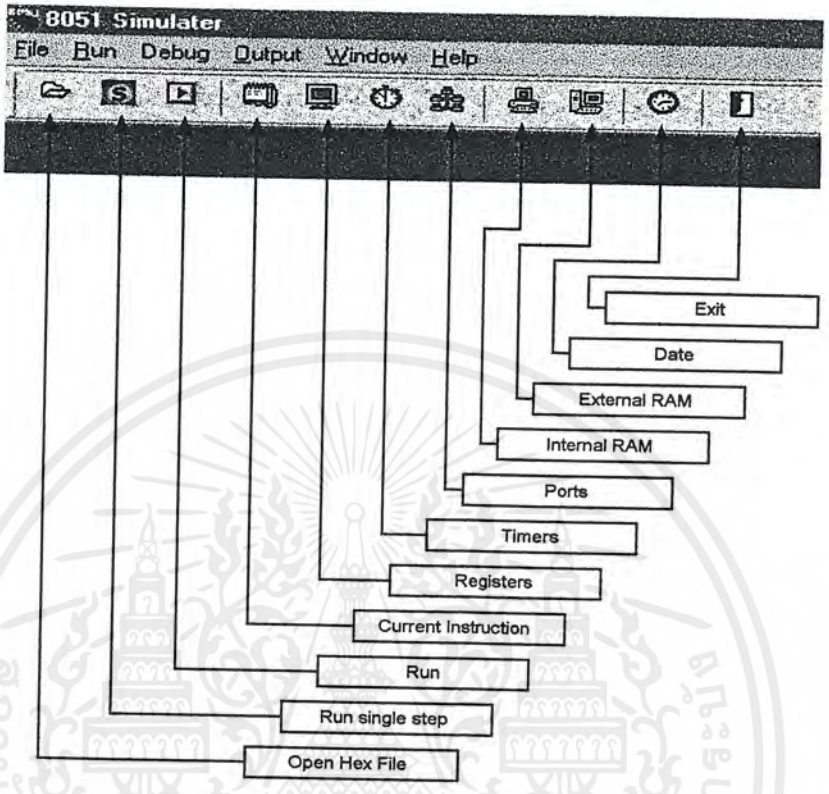
```
Procedure Tform1.AddA_dadd;
var An.Bn.Abn:byte;ans:word;
begin
        An := RegA and $0F;
        Bn := Address[Address[RegPc+1]] and $0F;
        Abn :=  An + Bn;
        AC_Flag(ABn);
        Ans := RegA + Address[Address[RegPC+1]];
        RegA := Io(ans);
        cy_flag(ans);
        ov_flag(ans);
        parity_flag(ans);
        RegPC := RegPC + 2;
        if SingleStep = true then exit; RunProgram;
end;
```

จากโปรแกรมย่อยตัวอย่าง จะเห็นได้ว่าโปรแกรมจะตรวจสอบค่าของแฟลกต่าง ๆ ที่เกี่ยวข้องและนำค่าที่ได้จากการประมวลผลไปเก็บไว้ในหน่วยความจำที่เป็นเป้าหมายและโปรแกรมจะทำการตรวจสอบค่าตำแหน่งต่อไปที่จะทำงาน โปรแกรมจะทำงานซ้ำวนรอบจนกระทั่งถึงคำสั่งให้

หยุดการทำงานหรือตำแหน่งที่จะทำงานต่อไปเป็นตำแหน่งที่ถูกกำหนดให้หยุดการทำงานโดยผู้ใช้โปรแกรม โปรแกรมจะหยุดการทำงาน

ภาพที่ 5.1 ไอคอนแสดงการเรียกใช้งานโปรแกรม 8501 Simulator ภายใต้ระบบปฏิบัติการWindow

ภาพที่ 5.2 แสดงหน้าต่าง Main ของ Program 8051 Simulater



ภาพที่ 5.3 แสดงหน้าที่ของปุ่มต่าง ๆ ใน Toolbar

5.1 File Dropdown Menu

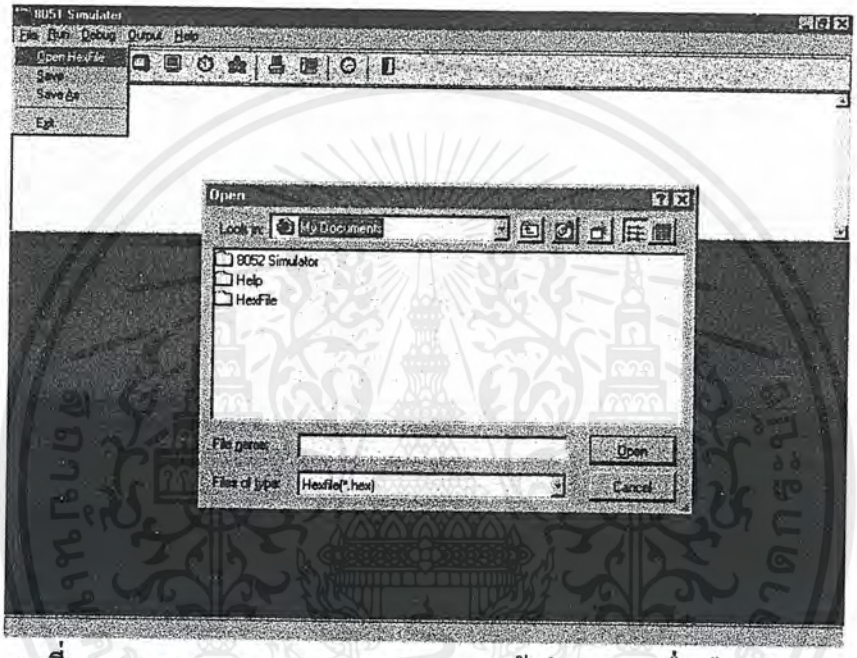
ตัวเลือกใน Menu นี้ประกอบด้วย

Open Hexfile

Save

Save As

Exit



ภาพที่5.4 แสดง File Dropdown Menu และ หน้าต่าง Open เพื่อเลือกเปิด File

5.1.1 File/Open Hexfile

ตัวเลือกนี้ใช้เพื่อโหลด Program 8051 เข้ามาในหน่วยความจำแบบจำลองเป็นครั้งแรก เมื่อเลือกตัวเลือกนี้แล้วคุณจะสามารถเลือกเปิด File ใน Format มาตราฐาน หรือ File ของ Intel ได้ (Hexfile) โดยเมื่อเปิดFile ใหม่ขึ้นมา Program ใด ๆ ที่มีอยู่ในหน่วยความจำจะถูกเขียนทับ Internal Ram, External Ram รวมถึง SFRs ทั้งหมดจะถูกล้างทิ้งข้อมูล Register ทั้งหมดจะถูกตั้งเป็นค่าดีฟอลท์

5.1.2 File/Save,Save As

ตัวเลือกนี้มีไว้ให้คุณสามารถ Save แบบจำลองที่ทำงานอยู่เก็บไว้เพื่อทำต่อในเวลาอื่น โดยข้อมูลทั้งหมดของแบบจำลองจะถูกบันทึกไว้และเมื่อเปิดขึ้นมาใหม่อีกครั้ง แบบจำลองจะสามารถทำงานต่อไปจากจุดที่ Save ทันทีเหมือนไม่เคยมีการหยุดแบบจำลองมาก่อน

5.1.3 File/Exit

ตัวเลือกนี้ใช้เมื่อต้องการออกจาก Program

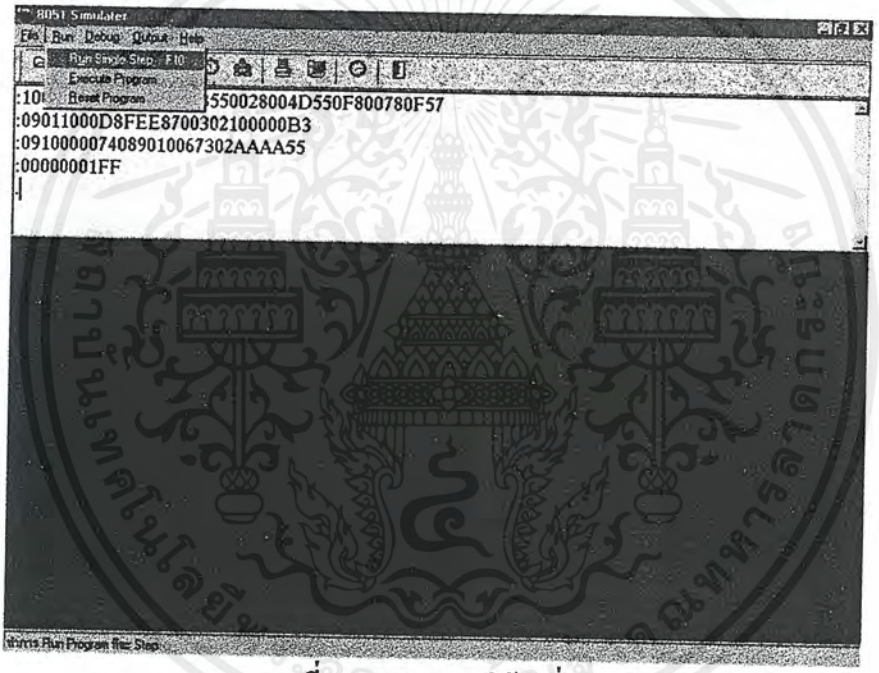
5.2 Run Dropdown Menu

ตัวเลือกนี้ใน Menu ประกอบด้วย

Run Single Step

Execute Program

Reset Program



ภาพที่ 5.5 แสดงการใช้คำสั่ง Run

5.2.1 Run/Run Single Step

ตัวเลือกนี้สั่งให้แบบจำลองทำงานเฉพาะที่กำหนดไว้ในคำสั่งถัดไป (คำสั่งที่แสดงในช่อง Next Instruction) แล้วก็จะหยุด Register ทุกค่าบนจอภาพย่อยทุกจอจะได้รับการอัพเดทตามความเปลี่ยนแปลงใด ๆ ที่เกิดขึ้นจากการทำงานตามคำสั่งในขั้นตอนนั้น ๆ

5.2.2 Run/Excute Program

ตัวเลือกนี้มีไว้เพื่อสั่งให้แบบจำลองสั่งให้ Program ที่โหลดขึ้นมาอยู่ในหน่วยความจำเริ่มทำงาน ในช่วงของคำสั่งที่บ่งชี้ไว้โดย PC การทำงานจะดำเนินไปเรื่อย ๆ จนกว่าจะถูกสั่งให้หยุด

5.2.3 Run/Reset Program

ตัวเลือกนี้จำลองการ Reset ระบบอย่างสมบูรณ์ของชิพ 8051 การ Reset Program จะ.....

1. ตั้งค่า Program Counter กลับเป็น 0 (การทำงานใด ๆ จะเริ่มใหม่ที่ Memory Address 0000)
2. กำหนดตำแหน่งของ Internal Ram ทั้ง 256 กลับไปที่ 0
3. ตั้ง External Ram ทั้ง 64 K กลับไปที่ 0
4. ตั้งค่า SFRs ทั้งหมดกลับเป็น 0 นอกจาก Stack Pointer ซึ่งจะถูกตั้งไปที่ 07 และพอร์ท SFRs จะถูกตั้งไว้ที่ FFh
5. ถ้าแบบจำลองถูกกำหนดไว้ให้วนรอบทับหน่วยความจำ Program /Code และ Program ที่ทำงานอยู่จะถูกเปลี่ยนแปลงในระหว่างการทำงานด้วยการเขียนข้อมูลทับตัวเองไป Program ที่โหลดเข้ามาเมื่อเริ่มแรกจะถูกนำกลับมาเหมือนเดิม

5.3 Debug Dorpdown Menu

ตัวเลือกนี้ใน Menu ประกอบด้วย

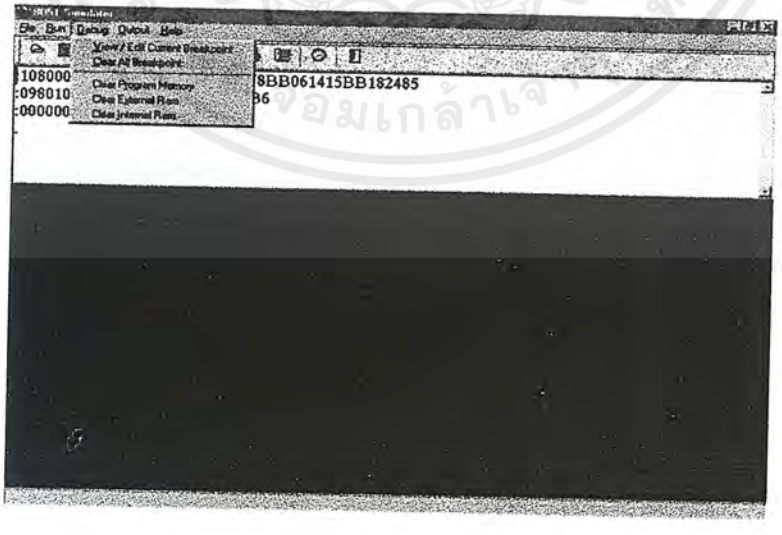
Clear All Break Point

View/Edit Current Break Point

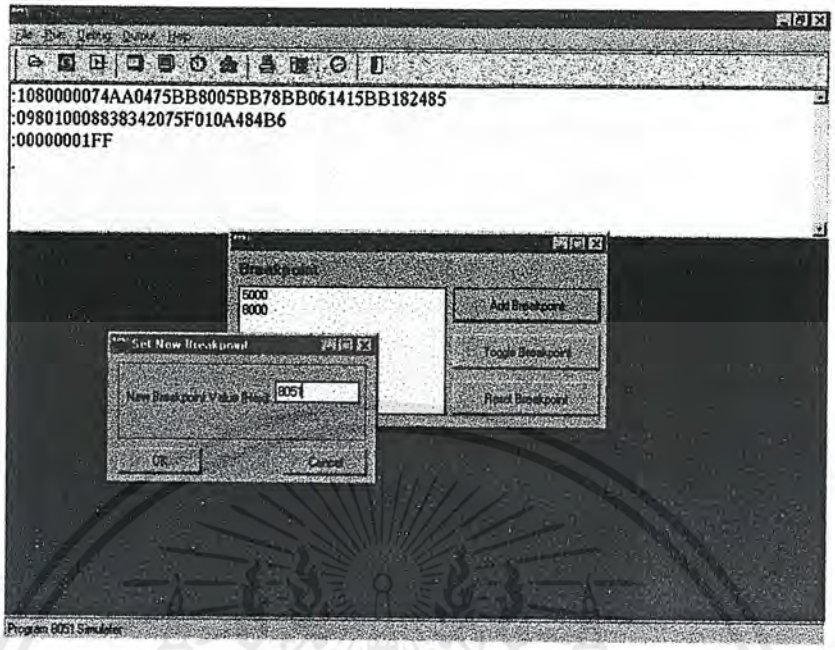
Clear Program Memory

Clear External Ram

Clear Internal Ram



ภาพที่ 5.6 แสดงเตัวเลือกในเมนู DebugDropdown Menu



ภาพที่ 5.7 แสดงการใส่ค่าตำแหน่งที่ต้องการให้เป็น Break Point

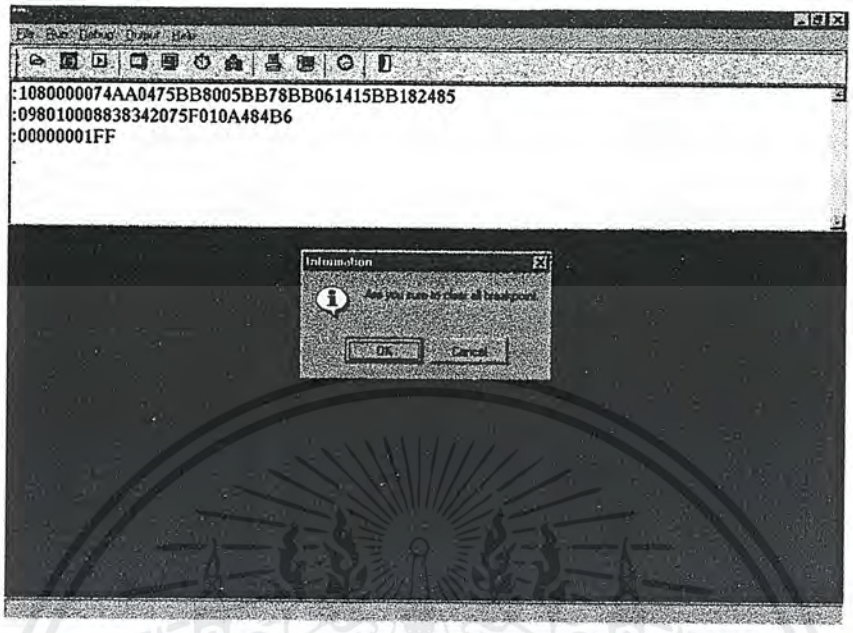
5.3.1 Debug/Toggle Break Point

ตัวเลือกนี้มีหน้าที่เพิ่มหรือถอดจุดตัด (Break Point) ใน Program จุดตัดคือจุดหนึ่งใน Memory ของ Program สั่ง Prograg ให้หยุดทำงาน เพื่อให้ผู้ใช้สามารถควบคุมระบบได้ด้วยตัวเอง หรือตรวจสอบค่าตัวแปรต่าง ๆ ในการทำงานของ Program นั้น โดยผู้ใช้สามารถสั่งให้ Program ทำงานต่อจากจุดที่หยุดได้ด้วยการใช้คำสั่ง Excute Program

ในการทำงานถ้าคำสั่ง Program ยังไม่ใช่จุดตัดการเลือกตัวเลือกนี้จะเพิ่มจุดตัดเข้าไปหน้าคำสั่ง และ Program จะหยุดทำงานก่อนถึงคำสั่งนั้นในการ Run Program ครั้งต่อไป ในทางกลับกันถ้าคำสั่งปัจจุบันมีจุดตัดอยู่แล้วการเลือกตัวเลือกนี้จะถอดจุดตัดที่มีออกจาก Program ซึ่งจะทำให้ Program ไม่หยุดทำงานในการ Run ครั้งต่อไป

5.3.2 Debug/Clear All Break Point

ใช้ตัวเลือกนี้เพื่อถอดจุดตัด (Break Point) ทุกจุดออกจาก Program ในการตรวจสอบ Program จะมีบ่อยครั้งเมื่อคุณทำงานไปได้ระยะหนึ่งแล้วพบว่า คุณได้ใส่จุดตัดไว้มากมายทั่ว Program ไปหมด แทนที่จะต้องไล่ถอดจุดตัดเหล่านั้นออกทีละจุด คุณสามารถเลือกตัวเลือกนี้เพื่อถอดจุดตัดทั้งหมดออกจาก Program ทั้งหมดได้ในครั้งเดียว



ภาพที่ 5.8 แสดงการใช้คำสั่ง Clear All Break Point

5.3.3 Debug/View/Edit Current Break Point

ตัวเลือกนี้จะเปิดหน้าต่างย่อยขึ้นมา หน้าต่างย่อยนี้จะแสดงรายชื่อของจุดตัด ที่ทำงานอยู่ทุกจุดใน Program

ณ จุดนี้ผู้ใช้สามารถลบจุดตัดที่มีอยู่ด้วยการคลิกที่ตำแหน่งที่แสดงอยู่บนรายชื่อ แล้วคลิกที่ Reset Break Point หรือเพิ่มจุดตัดเข้าไปใน Program ด้วยการคลิก Add Break Point แล้วใส่ตำแหน่ง (เลข Hexadecimal) ของ Memory Address ที่ต้องการวางจุดตัด

5.3.4 Debug /Clear Program Memory

Clear External Ram

Clear Internal Ram

ตัวเลือกทั้ง 3 ตัวนี้ มีหน้าที่ล้างข้อมูลในหน่วยความจำ 3 ประเภท ตามชื่อของตัวเลือกเมื่อสั่งล้างหน่วยความจำ สถานะส่วนอื่นของ Program ในแบบจำลองจะไม่ได้รับผลกระทบใด ๆ เช่นค่า SFRs จะไม่ถูก Reset เป็นต้น

5.4 Output Dropdown Menu

ตัวเลือกนี้ใน Menu ประกอบด้วย

Current Instruction

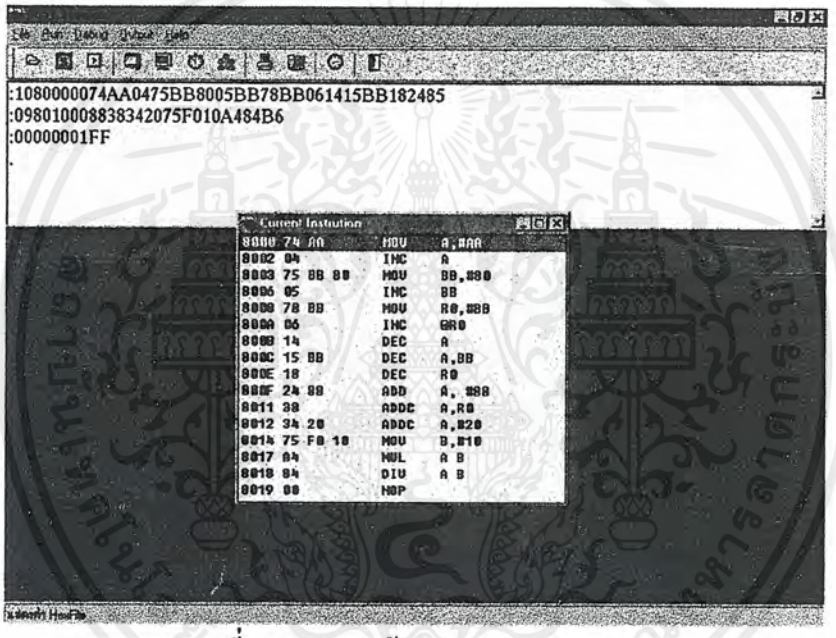
Registers

Timers

Ports

Internal Memory

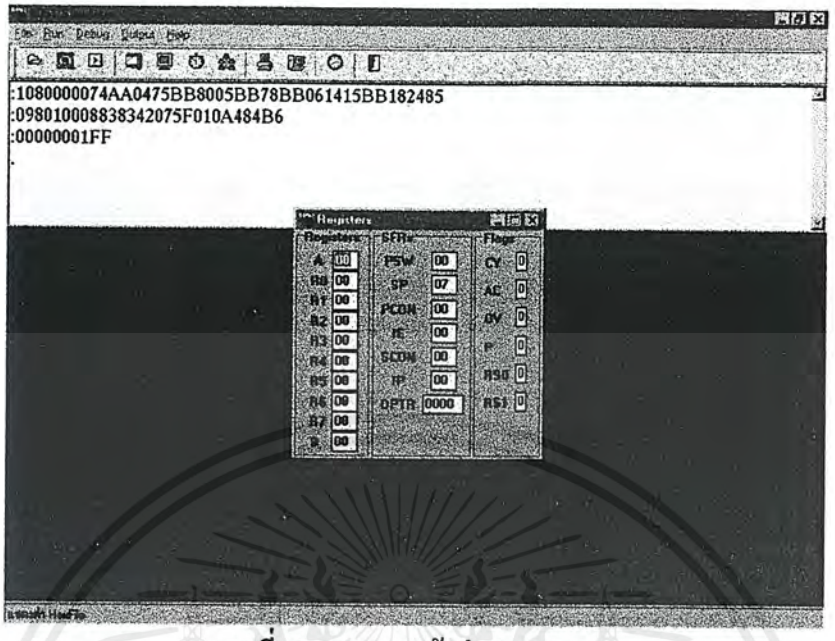
External Memory



ภาพที่ 5.9 แสดงหน้าต่าง Current Instruction

5.4.1 Output/Current Instruction

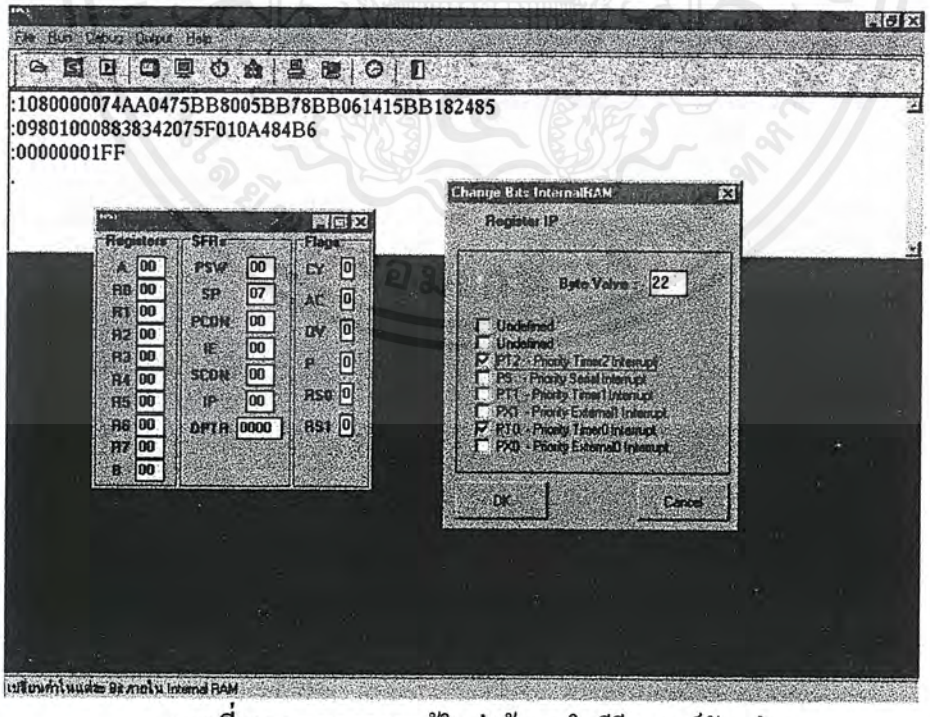
ตัวเลือกนี้ใช้สำหรับดูคำสั่งในภาษา Assembly จาก Hexfile ที่ได้ทำการโหลดมา อาจเปิดพร้อมกับหน้าต่าง Registers เพื่อดูความเปลี่ยนแปลงเมื่อทำการ Run Single Step



ภาพที่ 5.10 แสดงหน้าต่าง Register

5.4.2 Output/Register

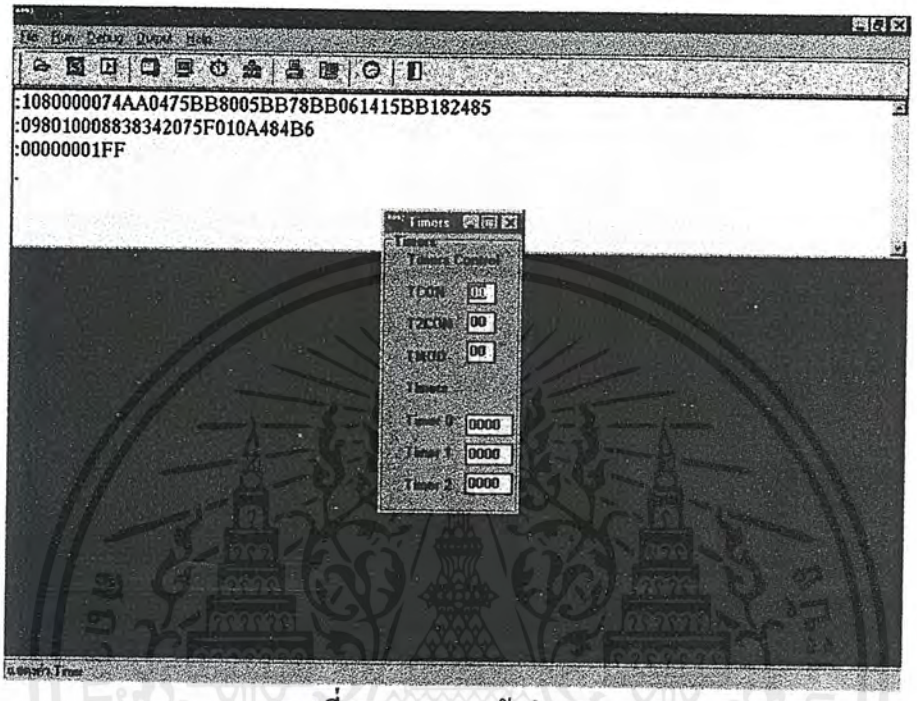
ตัวเลือกนี้จะแสดงหน้าต่างของ Registers และ SFRs ต่าง ๆ ซึ่งสามารถดูการเปลี่ยนแปลงและแก้ไขข้อมูลที่เกิดขึ้นกับรีจีสเตอร์ นั้น ๆ ได้



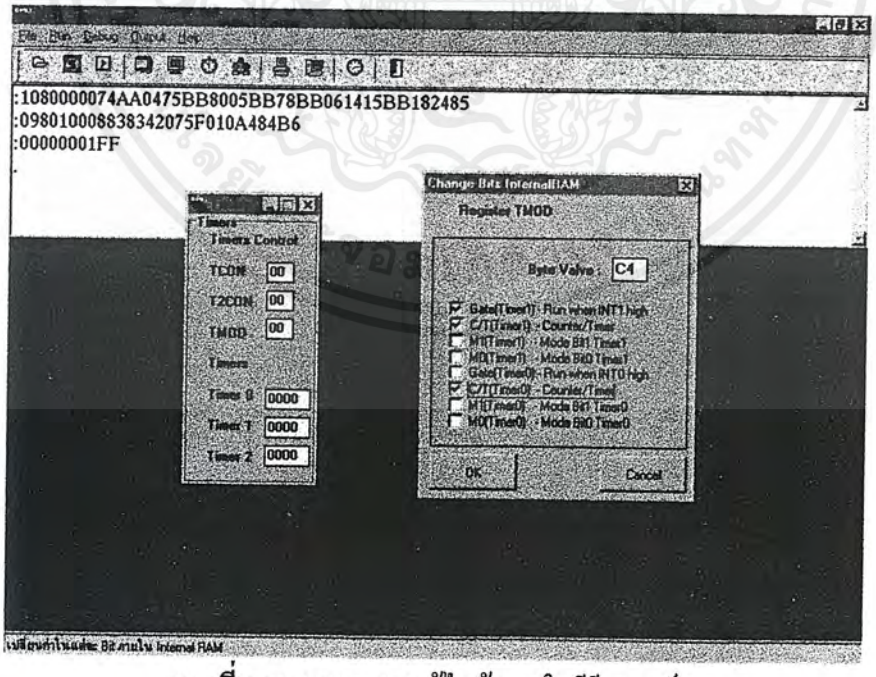
ภาพที่ 5.11 แสดงการแก้ไขค่าข้อมูลในรีจีสเตอร์ตัวอย่าง

5.4.3 Output/Timers

ตัวเลือกนี้จะใช้แสดงค่าข้อมูลและแก้ไขข้อมูลในรีจีสเตอร์ Timers ต่าง ๆ



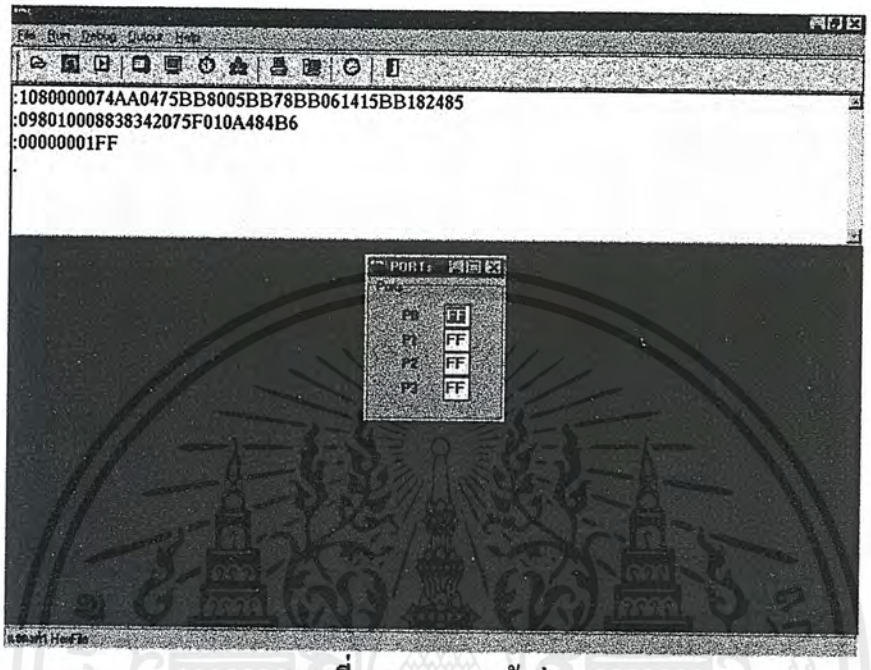
ภาพที่ 5.12 แสดงหน้าต่าง Timers



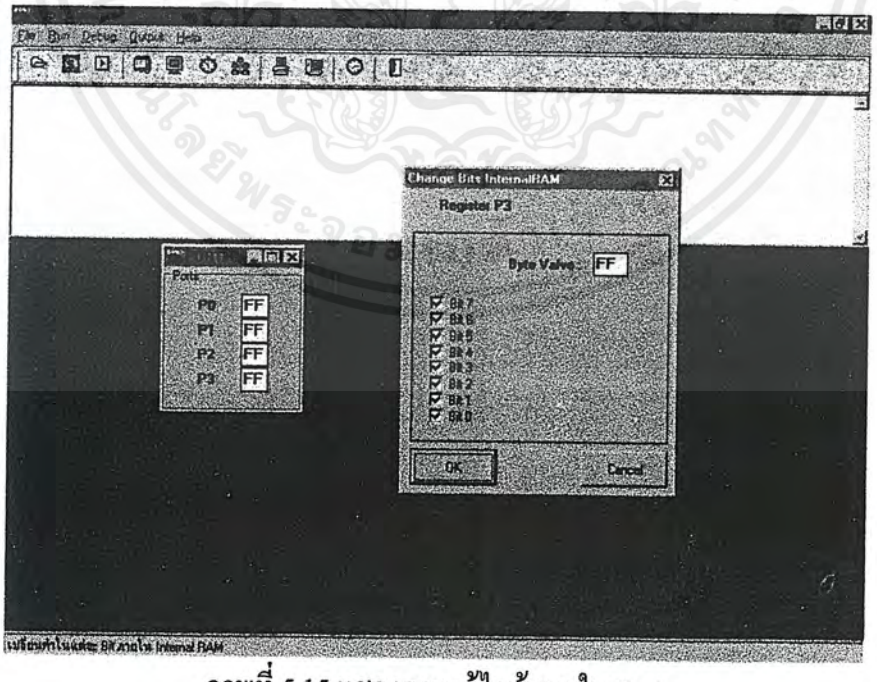
ภาพที่ 5.13 แสดงการแก้ไขข้อมูลในรีจีสเตอร์ Timer

5.4.4 Output/Ports

ตัวเลือกนี้จะแสดงค่าของ Ports ต่าง ๆ และสามารถแก้ไขข้อมูลในภายใน Port ได้



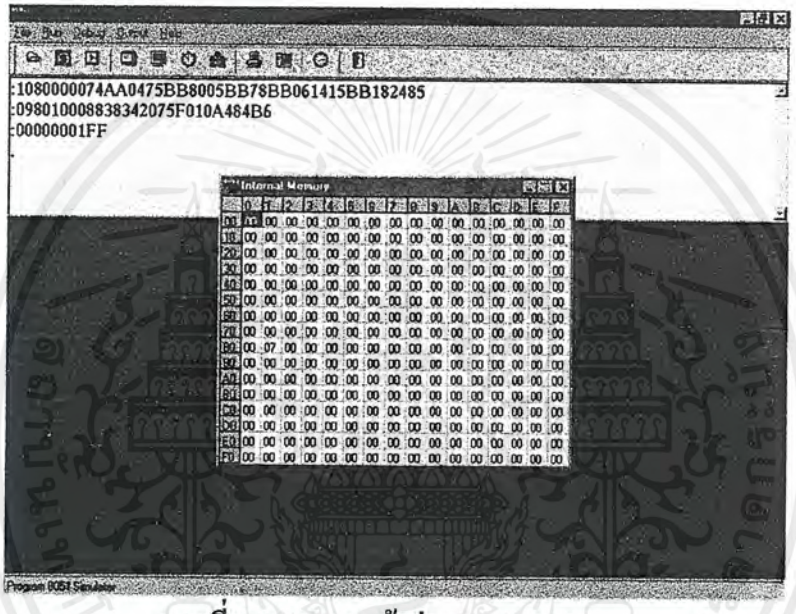
ภาพที่ 5.14 แสดงหน้าต่าง Ports



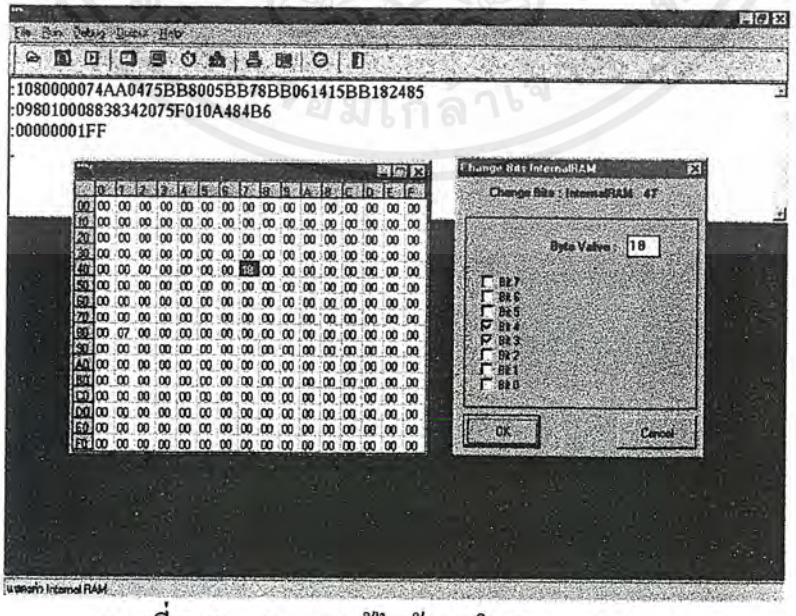
ภาพที่ 5.15 แสดงการแก้ไขข้อมูลใน Ports

5.4.5 Output/Internal Memory

ตัวเลือกนี้มีไว้สำหรับเปิดดูหน้าต่าง Internal Memory เพื่อดูและแก้ไขข้อมูลใน Ram 128 ไบต์ ที่มีอยู่ภายในโครงสร้างของ 8051 คุณสามารถทำให้ข้อมูลใน Internal Ram มีกี่อัพเดทตลอดเวลาในระหว่างการทำงานของ Program ด้วยการกำหนดให้แบบจำลองอัพเดทข้อมูล



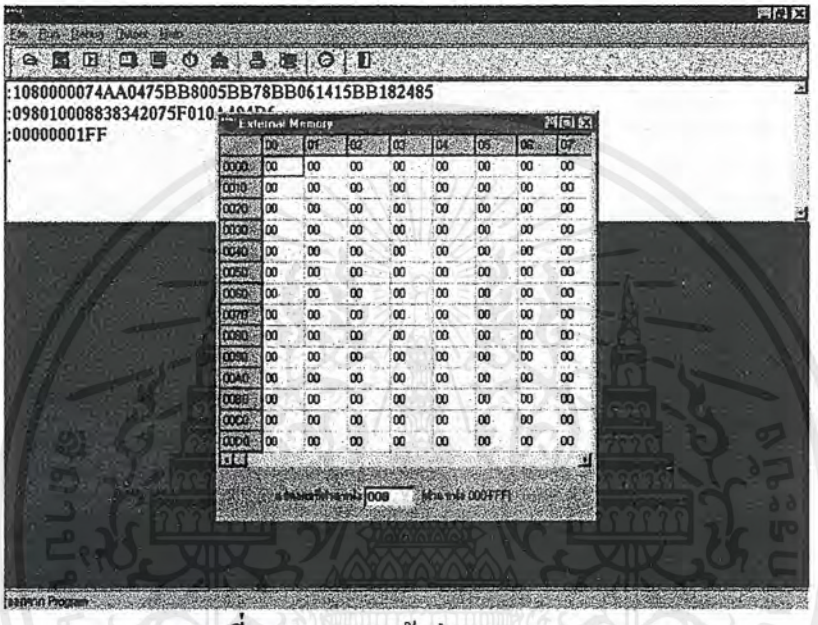
ภาพที่ 5.16 แสดงหน้าต่าง Internal Memory



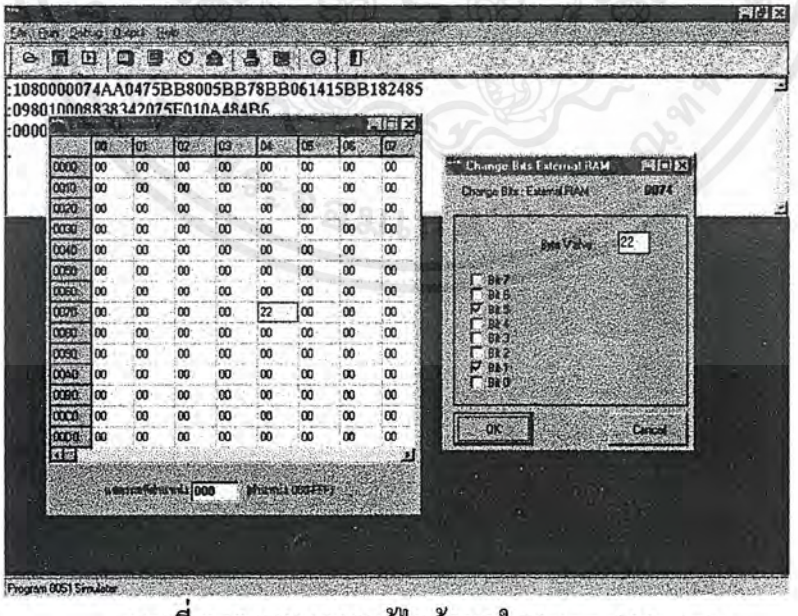
ภาพที่ 5.17 แสดงการแก้ไขข้อมูลใน Internal Memory

5.4.6 Output/External Ram

ตัวเลือกนี้ทำงานภายใต้หลักการเดียวกับ Internal Ram ต่างกันตรงที่หน้าต่างนี้ใช้ตรวจดูและแก้ไขค่าต่าง ๆ ในหน่วยความจำภายนอกของโครงสร้างแบบ 8051 เนื่องจากหน่วยความจำ External Ram มีขนาดใหญ่มาก ดังนั้นเพื่อความสะดวกผู้ใช้สามารถไปสู่ตำแหน่งที่ต้องการได้โดยกำหนดค่าตำแหน่งได้โดยใส่ค่าที่ต้องการใน Editblock ด้านล่าง

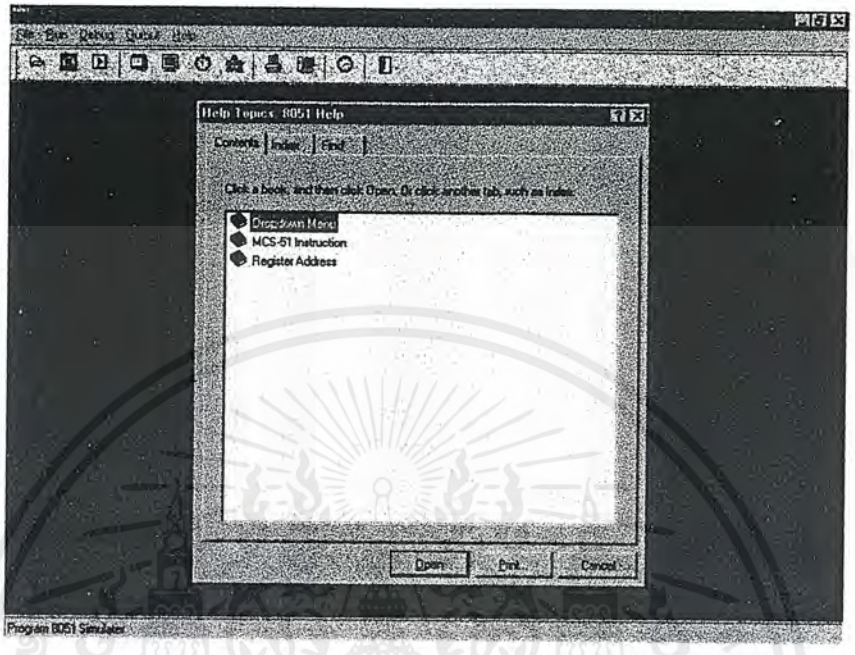


ภาพที่ 5.18 แสดงหน้าต่าง External Memory



ภาพที่ 5.19 แสดงการแก้ไขข้อมูลใน Internal Memory

5.5 Help Menu เมนูนี้มีไว้เพื่อช่วยแนะนำการใช้โปรแกรมแบบคร่าว ๆ



ภาพที่ 5.20 แสดงหน้าต่าง Help

ภายใน Help Menu จะมีตัวเลือกใช้งานอยู่ 3 ตัวเลือก ประกอบด้วย

5.5.1 Dropdown Menu Help

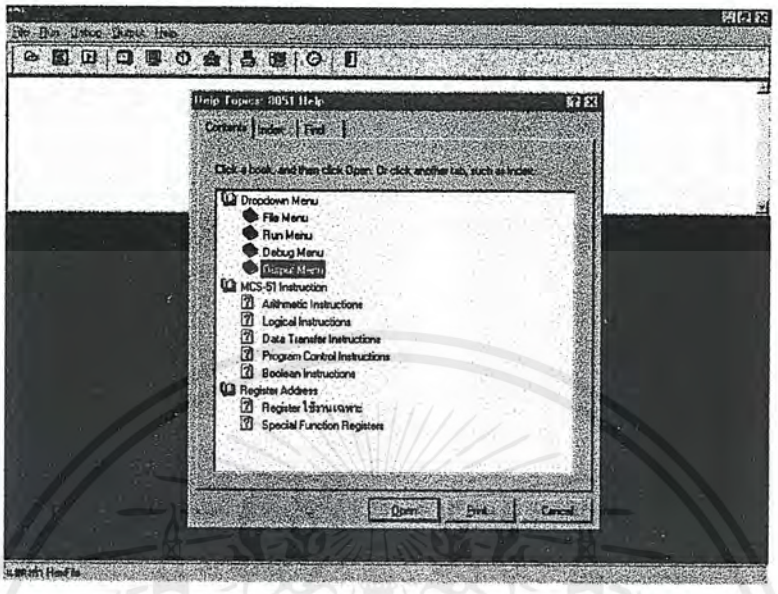
ตัวเลือกนี้จะแนะหน้าที่คำสั่งต่าง ๆ ของโปรแกรมโดยจะแบ่งตามกลุ่มของคำสั่งใช้งานโปรแกรม ได้แก่ File Menu, Run Menu, Debug Menu, Output Menu โดยจะอธิบายหน้าที่คำสั่งเหมือนดังที่กล่าวไปในหัวข้อของคำสั่งต่าง ๆ แล้ว

5.5.2 MCS-51 Instruction Help

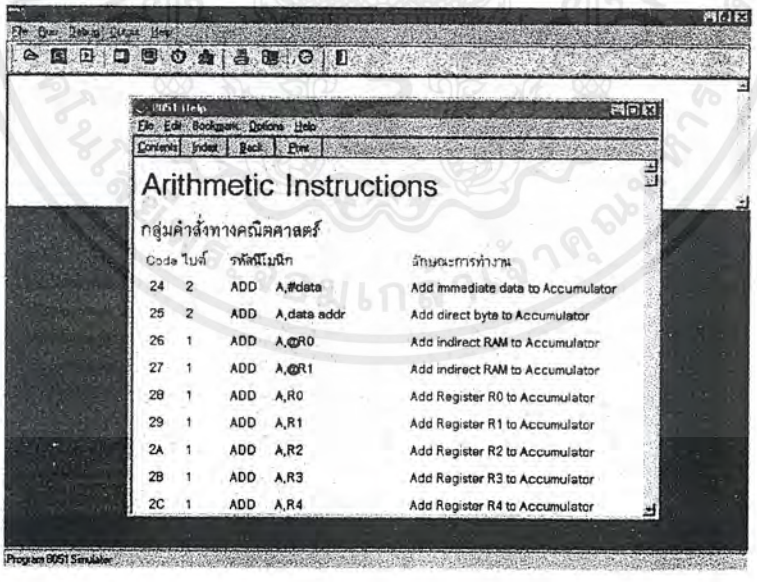
ตัวเลือกนี้จะแสดง Code จำนวนไบต์ รหัสนีโมนิก และลักษณะการทำงานของคำสั่งที่ใช้งานใน MCS-51 ทุกคำสั่ง โดยจะแบ่งกลุ่มตามลักษณะการทำงานของคำสั่ง

5.5.3 Register Address Help

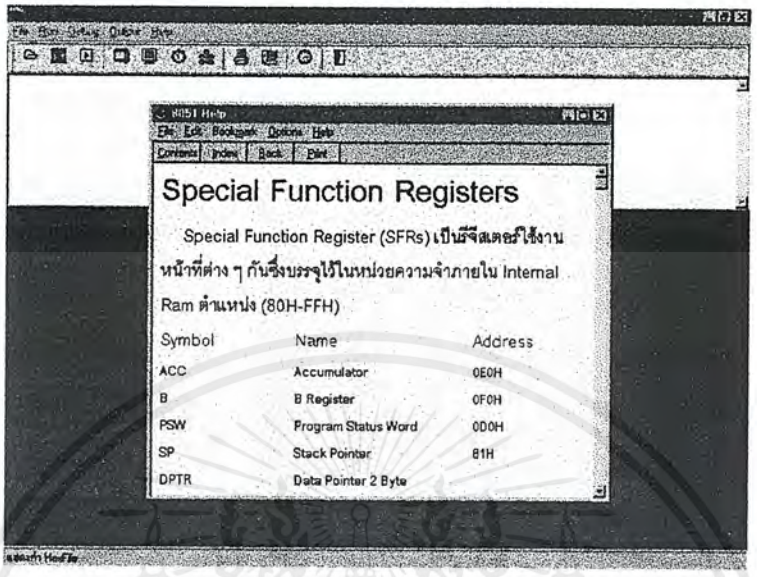
ตัวเลือกนี้จะแสดงตำแหน่งต่าง ๆ ของรีจีสเตอร์ทั้งหมดใน MCS-51 โดยจะแบ่งเป็น 2 กลุ่ม คือ รีจีสเตอร์ใช้งานทั่วไป และ Specail Function Registors



ภาพที่ 5.21 แสดงหัวข้อตัวเลือกต่าง ๆ ในหน้าต่าง Help



ภาพที่ 5.13 แสดงหน้าต่างรายละเอียดของคำสั่งเมื่อใช้คำสั่ง MCS-51 Instruction



ภาพที่ 5.24 หน้าต่างแสดงตำแหน่งของรีจิสเตอร์ เมื่อใช้คำสั่ง Register Address

สรุปผลและวิจารณ์

ปริญญานิพนธ์ฉบับนี้ มีวัตถุประสงค์ในการศึกษาและวิจัยเพื่อการสร้างโปรแกรมจำลองการทำงานของชิปไมโครคอนโทรลเลอร์เบอร์ 8051 ซึ่งเป็นซีพียูขนาด 16 บิต ในตระกูล MCS-51 โดยมีเป้าหมายให้สามารถจำลองการทำงานของชิปไมโครคอนโทรลเลอร์เบอร์ 8051 ลงบนระบบปฏิบัติการวินโดวส์ ซึ่งสามารถแสดงผลของการทำงานในรีจิสเตอร์ แฟลก และค่าในหน่วยความจำต่างๆในขณะที่โปรแกรมกำลังทำงานคำสั่งต่างๆได้

หลักการและแนวความคิดที่นำมาใช้ประกอบการวิจัยของปริญญานิพนธ์นี้คือ การศึกษาโครงสร้าง ภายใน รายละเอียดของหน่วยความจำ การส่งผ่านข้อมูลระหว่างส่วนต่างๆของหน่วยความจำ การทำงานของคำสั่งตลอดจนผลที่ได้จากการทำงานของคำสั่งต่างๆ ของชิปไมโครคอนโทรลเลอร์เบอร์ 8051 ด้วยเหตุนี้โปรแกรมจำลองการทำงานของ 8051 ไมโครคอนโทรลเลอร์ จึงมีความสะดวกและถูกต้องในการนำมาใช้งานในการทดลอง ปรับปรุง แก้ไขและพัฒนาโปรแกรมที่จะนำไปใช้งานกับ 8051 ไมโครคอนโทรลเลอร์ได้อย่างมีประสิทธิภาพ

ข้อเสนอแนะ

โปรแกรมจำลองการทำงานของ 8051 ไมโครคอนโทรลเลอร์นี้มีคุณสมบัติและฟังก์ชั่นการใช้งานที่มีความเหมาะสมและเพียงพอต่อการศึกษาและการนำไปประยุกต์ใช้งาน ได้อย่างมีประสิทธิภาพในระดับหนึ่งเท่านั้น ซึ่งในอนาคตจำเป็นต้องมีการพัฒนาให้มีขีดความสามารถ และมีประสิทธิภาพสูงขึ้นต่อไป โดยมีแนวทางในอนาคตดังนี้

1. พัฒนาโปรแกรมให้สามารถป้อนโปรแกรมคำสั่งเป็นแอสเซมบลีได้โดยตรง เพื่อ ความสะดวกในการพัฒนา แก้ไขปรับปรุงโปรแกรม

2. พัฒนาโปรแกรมให้สามารถมีการรับส่งข้อมูลและการทำงานติดต่อกับอุปกรณ์ภายนอกได้ด้วยการทำงานภายใต้เงื่อนไขเวลาที่ถูกต้อง

หนังสืออ้างอิง

1. ปรเมษฐ์ ประนยานันท์ และ ปิยพงศ์ เผ่าวณิช, "คู่มือและการประยุกต์ใช้งานไมโครคอนโทรลเลอร์ MCS-51" , บริษัท ซีเอ็ดยูเคชั่น จำกัด ,380 หน้า , 2536
2. ผศ. สมยศ จุณณปิยะ , "การประยุกต์ใช้งานไมโครคอนโทรลเลอร์ ตระกูล MCS-51" , คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง, 303 หน้า, 2541
3. แผนกหนังสือพิเศษด้านอิเล็กทรอนิกส์ , "ไมโครโปรเซสเซอร์ 1", บริษัท ซีเอ็ยูเคชั่นจำกัด, 219 หน้า , 2538
4. ไกรวุฒิ โรจน์ประเสริฐสุด, "ไมโครโปรเซสเซอร์ 2 ", บริษัท ซีเอ็ดยูเคชั่น จำกัด , 192 หน้า , 2539
5. นุกูล กระจาย , "การเขียนโปรแกรมแบบวิชวล Delphi 4" , บริษัท ซีเอ็ดยูเคชั่น จำกัด , 496 หน้า , 2542
6. กนก กุสุมาลย์ และ ไกรวุฒิ มั่นเสถียรสิน , "คู่มือการเขียนโปรแกรมด้วย Delphi" , บริษัท ซัคเซส จำกัด , 422 หน้า , 2542
7. วิมุติ วสะหลาย , "กลเม็ดเคล็ดลับ Borland Delphi 4" , บริษัท คอมกราฟ เพรส จำกัด, 233 หน้า, 2541
8. กิตติ เปรมพินิจ , "คู่มือการใช้งาน Borland Delphi 4" , ห้างหุ้นส่วนจำกัด เคพีเอน ซิสเต็มอินทิเกรเตอร์ , 234 หน้า , 2541