ปริญญานิพนธ์

โปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11

68HC11 Simulator

นางสาวพนัตฎาร์ ผิวผ่อง
นางสาวรุ่งทิพย์ ดวงกลาง

ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรครุศาสตร์อุตสาหกรรมบัณฑิต
สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์
ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2543

ภาควิชาครุศาสตร์วิศวกรรม

คณะครุศาสตร์อุตสาหกรรม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองปริญญานิพนธ์

ชื่อหัวข้อ โปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11

68HC11 Simulator

ชื่อนักศึกษา 1. นางสาวพนัตฎาร์ ผิวผ่อง รหัสประจำตัว 42035341

2. นางสาวรุ่งทิพย์ ดวงกลาง รหัสประจำตัว 42035347

หลักสูตร ครุศาสตร์อุตสาหกรรมบัณฑิต **สาขาวิชา** อิเล็กทรอนิกส์และคอมพิวเตอร์

อาจารย์ที่ปรึกษา อาจารย์ปิยะ ศุภวราสุวัฒน์

อาจารย์ที่ปรึกษาร่วม อาจารย์ไพบูลย์ พวงวงศ์ตระกูล

คณะกรรมการสอบปริญญานิพนธ์		ลายมือชื่อ
1. อาจารย์ปิยะ	ศุภวราสุวัฒน์	.....
2. อาจารย์ไพบูลย์	พวงวงศ์ตระกูล	.....
3. อาจารย์สุชิน	อาจหาญ	.....
4. อาจารย์ปิยะ	จิตธรรมมาภิรมย์	.....
5. อาจารย์อำพล	ทองระอา	.....

วัน/เดือน/ปีที่สอบ วันเสาร์ที่ 9 ธันวาคม พ.ศ. 2543 เวลา 13.30 น.

สถานที่สอบ ห้อง ค.311 คณะครุศาสตร์อุตสาหกรรม สจล.

ภาควิชารับรองแล้ว

ลงนาม.....

(ผศ.วิสุทธิ์ อธิพรธรรม)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

วันที่.....เดือน.....พ.ศ.....

ปริญญานิพนธ์

เรื่อง โปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11
68HC11 Simulator

วัตถุประสงค์

1. ศึกษาการเขียนโปรแกรมบนวินโดวส์ด้วยวิชวลเบสิก
2. ศึกษารูปแบบคำสั่งและการทำงานของไมโครคอนโทรลเลอร์ 68HC11
3. ออกแบบโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11
4. สร้างโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11
5. ทดลองการใช้งานโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11
6. นำเอาไปใช้ในการเรียนการสอนวิชาไมโครคอนโทรลเลอร์ 68HC11

ประโยชน์ที่คาดว่าจะได้รับ

1. ได้รับความรู้ทักษะจากการเขียนโปรแกรมบนวินโดวส์ด้วยวิชวลเบสิก
2. ได้รับความรู้จากการศึกษารูปแบบคำสั่งและการทำงานของไมโครคอนโทรลเลอร์ 68HC11
3. ได้รับความรู้ในการออกแบบโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11
4. ได้รับความรู้ในการสร้างโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11
5. ทดลองการใช้งานโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11
6. สามารถนำโปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11 ไปใช้ในการเรียนการสอนวิชาไมโครคอนโทรลเลอร์ 68HC11 ได้จริง

ชื่อหัวข้อ	โปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11
นักศึกษา	นางสาวพนัตฎาร์ ผิวผ่อง
	นางสาวรุ่งทิพย์ ดวงกลาง
อาจารย์ที่ปรึกษา	อาจารย์ปิยะ ศุภวราสุวัฒน์
อาจารย์ที่ปรึกษาร่วม	อาจารย์ไพบูลย์ พวงวงศ์ตระกูล
หลักสูตร	ครุศาสตร์อุตสาหกรรมบัณฑิต
สาขาวิชา	อิเล็กทรอนิกส์และคอมพิวเตอร์
ปีการศึกษา	2543

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้นำเสนอโปรแกรมจำลองการทำงานไมโครคอนโทรลเลอร์ 68HC11 โดยมีคุณสมบัติเฉพาะของโปรแกรมดังนี้ สามารถจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11ได้, สามารถแสดงค่าของรีจิสเตอร์ และสถานะแฟลกต่างๆ ได้, จำลองการทำงานของอุปกรณ์ต่อร่วมเพื่อการติดต่อได้, แสดงค่าข้อมูลของหน่วยความจำที่ตำแหน่งต่างๆ ได้ และมีการทำงานภายใต้ระบบปฏิบัติการวินโดวส์

Thesis Title	68HC11 Simulator
Students	Miss Panutdar Phiwpong
	Miss Rungtip Daungklang
Advisor	Mr. Piya Supavarasuwat
Co-Advisor	Mr. Paiboon Pongwongtragull
Education Level	Bachelor of Science in Industrial Education
Program in	Electronics and Computer
Academic Year	2000

ABSTRACT

This thesis presents the 68HC11 simulator. The properties of a software thus able simulation 68HC11 microcontroler, able to show register and status flag, simulation of board interface, show data of memory at any position and use by windows operating system.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปด้วยดี เนื่องจากความร่วมมือของสมาชิกภายในกลุ่มผู้จัดทำขอกราบขอบพระคุณอาจารย์ปิยะ ศุภวราสุวัฒน์ อาจารย์ไพบูลย์ พวงวงศ์ตระกูล อาจารย์ปิยะ จิตธรรมมาภิรมย์ อาจารย์สุชิน อาจหาญ รวมทั้งคณาจารย์ภาควิชาครุศาสตร์วิศวกรรมทุกท่านที่กรุณาให้คำแนะนำ แนวความคิด ความรู้ ตลอดจนแนวทางในการแก้ไขปัญหาในการจัดทำปริญญานิพนธ์ ขอขอบคุณเพื่อนๆ ทุกคนที่ให้ความอนุเคราะห์ในการช่วยเหลือด้านต่างๆ สุดท้ายที่ควรระลึกถึงอย่างยิ่งคือ บิดา และมารดา ที่เป็นผู้ให้ความสนับสนุนทางด้านการศึกษา เงินทุน และให้กำลังใจด้วยดีตลอดมาตั้งแต่อดีตจนถึงปัจจุบัน จนทำให้ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี

สารบัญ

สารบัญ (ต่อ)

สารบัญ (ต่อ)

สารบัญตาราง

ตาราง **หน้า**

สารบัญตาราง (ต่อ)

ตาราง **หน้า**

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

รูป **หน้า**

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปริญญานิพนธ์

ในกระบวนการทำงานหลายๆ ด้านเราจะสังเกตเห็นว่าได้มีการนำไมโครคอนโทรลเลอร์ 68HC11 มาใช้งานทางด้านการควบคุมหรืองานอื่นๆ ตามแต่ผู้ใช้ต้องการอยู่หลายๆ งานด้วยกัน เช่น การควบคุมการเปิดปิดสวิตช์ไฟ การควบคุมการแสดงผลของตัวอักษรวิ่ง เป็นต้น ดังนั้นจึงแสดงให้เราเห็นว่าไมโครคอนโทรลเลอร์มีประโยชน์และความสำคัญอย่างมาก แต่กระบวนการเรียนรู้นอกจากการศึกษาในหนังสือหรือตำราเรียนแล้ว สิ่งหนึ่งที่จะช่วยให้นักศึกษามีความเข้าใจมากยิ่งขึ้นก็คือการได้ฝึกหัด การเขียนโปรแกรมและทำการทดลอง แต่ปัญหาที่เกิดขึ้นคือนักศึกษาไม่มีชุดทดลองแบบแผ่นพิมพ์เดี่ยวที่จะนำมาใช้ศึกษาหรือทดลองให้เพียงพอต่อความต้องการ และเนื่องจากชุดทดลองแบบแผ่นพิมพ์เดี่ยวมีราคาแพง ดังนั้นจึงมีความคิดที่จะเขียนโปรแกรมเพื่อจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11 และมีการจำลองส่วนของอุปกรณ์ทางด้านฮาร์ดแวร์ ไว้ด้วยเพื่อให้เห็นถึงผลของการทำงานตามคำสั่งของไมโครคอนโทรลเลอร์ 68HC11 ซึ่งทำให้ผู้ใช้มีความสะดวกในการใช้งานได้ดีขึ้น

1.2 ขีดความสามารถของโครงงาน

โครงงานนี้มีขีดความสามารถดังต่อไปนี้

1) สามารถจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11 ได้
2) สามารถแสดงค่ารีจิสเตอร์ต่างๆ ได้ เช่น A, B และ D เป็นต้น
3) สามารถแสดงผลการทำงานของโปรแกรมบนจอคอมพิวเตอร์ได้
4) สามารถบันทึก (Save) และเปิด (Open File) ไฟล์ที่เขียนขึ้นแล้วได้
5) สามารถอ่านคู่มือ (Help) เพื่อช่วยในการใช้โปรแกรมได้
6) โปรแกรมทำงานภายใต้ระบบปฏิบัติการวินโดว์ 95/98
7) มีชุดอินเตอร์เฟสจำลอง 8255

1.3 เนื้อหาโดยสังเขป

เนื้อหาภายในปริญญานิพนธ์ฉบับนี้แบ่งออกเป็นบทต่างๆ เพื่อความสะดวกต่อการศึกษาและทำความเข้าใจ ในแต่ละบทจะประกอบด้วยเนื้อหาที่สำคัญดังนี้

บทที่ 2 ทฤษฎีและหลักการ ประกอบด้วยเนื้อหาในทางทฤษฎีที่เกี่ยวข้องคือ ข้อมูลทางทฤษฎีของไมโครคอนโทรลเลอร์ 68HC11 ทั้งสถาปัตยกรรมโครงสร้าง การทำงาน ตลอดจนรายละเอียดการทำงานของชุดคำสั่งที่ใช้กับไมโครคอนโทรลเลอร์เพื่อให้สามารถทำงานได้ตามความต้องการ และวิธีการใช้งานโปรแกรมวิชวลเบสิกในการออกแบบโปรแกรม ซึ่งจะทำให้ผู้อ่านได้มีความรู้ความเข้าใจพื้นฐานเพื่อเป็นประโยชน์ต่อการทำความเข้าใจกับการทำงานของโปรแกรมที่ใช้งานจริงต่อไป

บทที่ 3 การออกแบบ การสร้างและการทำงาน ประกอบด้วยเนื้อหาในเรื่องของการออกแบบโปรแกรม การสร้างหน้าจอต่างๆ การสร้างชุดอินเตอร์เฟสจำลองการทำงาน การสร้างไฟล์ช่วยเหลือ และอื่นๆ เพื่อให้ผู้อ่านได้เข้าใจวิธีการในการออกแบบและการสร้างเพื่อทำการพัฒนาหรือปรับปรุงโปรแกรมให้มีประสิทธิภาพเพิ่มขึ้นในโอกาสต่อไป

บทที่ 4 การทดลองและผลการทดลอง กล่าวถึงขั้นตอนในการทำการทดลอง และผลการทดสอบโปรแกรม ในชุดคำสั่งของโปรแกรมในส่วนต่างๆ

บทที่ 5 บทสรุป ปัญหา แนวทางแก้ไขและพัฒนา กล่าวถึงปัญหาที่เกิดขึ้นจากการทำงานของผู้ศึกษา และแนวทางในการแก้ไขที่ได้รับจากอาจารย์ที่ปรึกษาปริญญานิพนธ์ ในด้านต่างๆ

ในภาคผนวกแสดงรายละเอียดของโปรแกรม และส่วนต่างๆ ที่ใช้จัดทำโครงงาน

บทที่ 2
ทฤษฎีและหลักการ

2.1 กล่าวนำ

เนื้อหาของปริญญานิพนธ์ในบทนี้เป็นทฤษฎี และหลักการที่นำมาใช้ประกอบการสร้างโครงงานนี้ขึ้นมา ซึ่งประกอบด้วยทฤษฎี และหลักการเกี่ยวกับสถาปัตยกรรม และการทำงานของไมโครคอนโทรลเลอร์ 68HC11 คุณสมบัติและโครงสร้างของโปรแกรมวิชวลเบสิก (Visual Basic) ซึ่งหลักการเขียนโปรแกรมโดยใช้โปรแกรมวิชวลเบสิก จะได้กล่าวถึงดังต่อไปนี้

2.2 ไมโครคอนโทรลเลอร์ 68HC11

2.2.1 ลักษณะโดยทั่วไปของไมโครคอนโทรลเลอร์ 68HC11

ไมโครคอนโทรลเลอร์ 68HC11 เป็นชิปไมโครคอนโทรลเลอร์เบอร์หนึ่ง ของบริษัทโมโตโรลาที่ได้ออกแบบมาโดยรวมอุปกรณ์ที่จำเป็นในการใช้งานของระบบไมโครคอนโทรลเลอร์ไว้ภายใน เช่น หน่วยความจำรอม แรม และอีอีพรอม พอร์ตอินพุต/เอาต์พุต วงจรตั้งเวลา ตัวนับ ระบบการอินเตอร์รัปต์ วงจรแปลงสัญญาณแอนะลอกเป็นดิจิตอล ระบบป้องกันความผิดพลาดที่จะเกิดขึ้นในโปรแกรม และส่วนติดต่อสื่อสารข้อมูลผ่านทางพอร์ตอนุกรมสามารถ สรุปคุณสมบัติโดยทั่วไปของ 68HC11 ได้ 2 ลักษณะดังนี้

1) คุณสมบัติทางฮาร์ดแวร์ มีรายละเอียด ดังนี้

1.1) ซีพียูขนาด 8 บิต

1.2) มีหน่วยความจำรอมภายในสำหรับโปรแกรมขนาด 4, 8 หรือ 12 กิโลไบต์ (บางเบอร์จะไม่มีส่วนนี้)

1.3) มีหน่วยความจำอีอีพรอมภายในขนาด 512 ไบต์ หรือ 2 กิโลไบต์

1.4) มีหน่วยความจำแรมภายในขนาด 192, 256 หรือ 512 ไบต์

1.5) มีตัวตั้งเวลาและตัวนับขนาด 8 บิต

1.6) มีวงจรแปลงสัญญาณแอนะลอกเป็นดิจิตอล 8 บิต 8 ช่อง

1.7) มีวงจรพัลส์แอกคิวมูเลเตอร์ขนาด 8 บิต

1.8) มีส่วนติดต่อสื่อสารข้อมูลผ่านทางพอร์ตอนุกรม (Serial Communication Interface SCI)

1.9) มีวงจรเชื่อมต่ออุปกรณ์อนุกรม (Serial Peripheral Interface SPI)

1.10) มีวงจรรีลไทม์อินเตอร์รัปต์

1.11) มีระบบวอตช์ด็อก

1.12) สามารถขยายตัวตั้งเวลาเป็นระบบ 16 บิตได้ โดยมีฟังก์ชันการทำงานเพิ่มเติม คือ วงจรปรีสเกลเลอร์ที่สามารถโปรแกรมได้ 4 สเกล อินพุตสำหรับตรวจจับสัญญาณ 3 อินพุต และเอาต์พุตสำหรับวงจรเปรียบเทียบ 5 เอาต์พุต

1.13) บรรจุในตัวถังหลายแบบ คือ DIP ขนาด 48 ขา, PLCC ขนาด 52 ขา หรือ QFT ขนาด 64 ขา

1.14) มีระบบอินเตอร์รัปต์ 2 ระดับ จาก 21 แหล่ง

1.15) สามารถติดต่อหน่วยความจำภายนอกได้ 64 กิโลไบต์

1.16) สามารถต่อเชื่อมกันทำงานเป็นมัลติโปรเซสเซอร์ได้

2) คุณสมบัติทางซอฟต์แวร์ มีรายละเอียด ดังนี้

2.1) มีชุดคำสั่งเพิ่มเติมมากกว่าไมโครโปรเซสเซอร์ เบอร์ 6800 และ 6801 จึงสามารถใช้ชุดคำสั่งเดียวกับ 6800 หรือ 6801 ได้

2.2) สามารถทำการหารเลข 16 บิตโดยได้ผลลัพธ์เป็นตัวเลข 16 บิตและเศษขนาด 16 บิต

2.3) สามารถประมวลผลข้อมูลละเอียดถึงระดับบิต

2.4) มีโหมดการทำงาน WAIT และโหมด STOP เพื่อประหยัดพลังงาน 68HC11 เป็นชิปไมโครคอนโทรลเลอร์ ที่ได้รับการผลิตโดยใช้เทคโนโลยี HCMOS (High – density CMOS) จึงทำให้มีความเร็วในการทำงานสูง ในขณะที่กินกำลังงานไฟฟ้าต่ำ

2.2.2 ตระกูลของไมโครคอนโทรลเลอร์ 68HC11

บริษัทโมโตโรลาได้ผลิตชิป 68HC11 ออกมาหลายเวอร์ชัน โดยแต่ละเวอร์ชันจะมีความแตกต่างกันไป ในด้านการรวมอุปกรณ์ภายนอกเข้าไปไว้ในชิป เช่น ขนาดและชนิดของหน่วยความจำ จำนวนวงจรแปลงสัญญาณ แอนะลอกเป็นดิจิตอล ตัวตั้งเวลา ตัวนับ เป็นต้น โดยผู้ใช้สามารถสังเกตได้จากรหัสเบอร์ที่ต่อท้ายเบอร์ 68HC11 ในตารางที่ 2.1 เป็นตารางสรุปตระกูลของ 68HC11

ตารางที่ 2.1 ตระกูลของไมโครคอนโทรลเลอร์ 68HC11 ทั้งแบบ 48, 52 และ 64 ขา

รูปร่างตัวถัง	เบอร์	คำอธิบาย	CONFIG
แบบ DIP (Dual-in-line Package) 48 ขา	MC68HC11AOCP	ไม่มีรอมและอีอีพรอม	$0C
	MC68HC11AOCP3	ทำงานที่ 3 เมกะเฮิรตซ์ ไม่มีรอม และอีอีพรอม	$0C
	MC68HC11A1CP	ไม่มีรอม	$0D
	MC68HC11A1CP3	ทำงานที่ 3 MH_z ไม่มีรอม	$0D
	MC68HC11A1VP	ไม่มีรอม	$0D
	MC68HC11A1MP	ไม่มีรอม	$0D
	MC68HC11A1CP	มีวงจรวอตช์ด็อก ไม่มีรอม	$09
	MC68HC11A1CP3	ทำงานที่ 3 เมกะเฮิรตซ์ มีวงจรวอตช์ด็อก ไม่มีรอม	$09
	MC68HC11A1VP	มีวงจรวอตช์ด็อก ไม่มีรอม	$09
	MC68HC11A1MP	มีวงจรวอตช์ด็อก ไม่มีรอม	$09
	MC68HC1148CP1	มีบัฟฟาโลรอม**	$0F
	MC68HC11A8VP1	มีบัฟฟาโลรอม**	$0F
	MC68HC11A8MP1	มีบัฟฟาโลรอม**	$0F
	XC68HC11B8***	มีรอม 8 กิโลไบต์ เป็นเวอร์ชันที่พัฒนาเป็นรุ่นแรก	$0F
	XC68HC11B1***	ไม่มีรอม	$0D
	XC68HC11B0	ไม่มีรอมและอีอีพรอม	$0C
	PLC68HC11E9	มีรอม 12 กิโลไบต์ แรม 512 ไบต์ และอินพุตแคปเจอร์ 4 ช่อง	$0F
	MC68HC11E1	ไม่มีรอม	$0D
	MC68HC11E0	ไม่มีรอมและอีอีพรอม	$0C
	MC68HC811E2	มีอีอีพรอม 2 กิโลไบต์ ****	$FF
	MC68HC11D3	มีรอม 4 กิโลไบต์ เป็นรุ่นประหยัด ไม่มีวงจร ADC	N/A

ตารางที่ 2.1 (ต่อ) ตระกูลของไมโครคอนโทรลเลอร์ 68HC11 ทั้งแบบ 48, 52 และ 64 ขา

รูปร่างตัวถัง	เบอร์	คำอธิบาย	CONFIG
แบบ PLCC (Plastic Leaded Chip Carrie) 52 ขา	MC68HC11A0CFN	ไม่มีรอมและอีอีพรอม	$0C
	MC68HC11A0CFN3	ทำงานที่ 3 เมกะเฮิรตซ์ ไม่มีรอม และอีอีพรอม	$0C
	MC68HC11A1CEN	ไม่มีรอม	$0D
	MC68HC11A1CFN3	ทำงานที่ 3 เมกะเฮิรตซ์ ไม่มีรอม	$0D
	MC68HC11A1VFN	ไม่มีรอม	$0D
	MC68HC11A1MFN	ไม่มีรอม	$0D
	MC68HC11A1CFN	มีวงจรวอตช์ด็อก ไม่มีรอม	$09
	MC68HC11A1CFN3	ทำงานที่ 3 เมกะเฮิรตซ์ มีวงจรวอตช์ด็อก ไม่มีรอม	$09
	MC68HC11A1VFN	มีวงจรวอตช์ด็อก ไม่มีรอม	$09
	MC68HC11A1MFN	มีวงจรวอตช์ด็อก ไม่มีรอม	$09
	MC68HC11A8CFN1	มีบัฟฟาโลรอม**	$0F
	MC68HC11A8VFN1	มีบัฟฟาโลรอม**	$0F
	MC68HC11A8MFN1	มีบัฟฟาโลรอม**	$0F
แบบ QFP (Quad Flat Pack) 64 ขา	MC68HC11AU	ไม่มีรอมและอีอีพรอม	$0C
	MC68HC11A0CFU3	ทำงานที่ 3 เมกะเฮิรตซ์ ไม่มีรอม และอีอีพรอม	$0C
	MC68HC11A1CFU	ไม่มีรอม	$0D
	MC68HC11A1CFU3	ทำงานที่ 3 เมกะเฮิรตซ์ ไม่มีรอม	$0D
	MC68HC11A1VPU	ไม่มีรอม	$0D
	MC68HC11A1MFU	ไม่มีรอม	$0D
	MC68HC11A8CFU1	มีบัฟฟาโลรอม **	$0F
	MC68HC11A8VFU1	มีบัฟฟาโลรอม **	$0F

จากตารางที่ 2.1

* CONFIG หมายถึง ค่าในรีจิสเตอร์ CONFIG เป็นค่าที่บ่งบอกถึงลำดับก่อนหลังของการผลิตออกมาจากโมโตโลรา

** บัฟฟาโลรอม คือ โปรแกรมมอนิเตอร์ ของบอร์ดพัฒนาชิปไมโครคอนโทรลเลอร์ 68HC11 (Evaluation board) ซึ่งทางโมโตโรลาเขียนขึ้น แล้วบรรจุลงในหน่วยความจำรอมภายในชิป

*** ใน 68HC11 เวอร์ชันนี้ การโปรแกรมอีอีพรอมต้องใช้แรงดันภายนอก 19 โวลต์

**** สามารถเปลี่ยนตำแหน่งได้ภายในขอบเขต 4 กิโลไบต์ โดยการกำหนดที่บิตภายในรีจิสเตอร์ CONFIG

ตารางที่ 2.1 สามารถสรุปตระกูลของ 68HC11 เป็นอนุกรมได้ดังนี้

1) อนุกรม A

จะปรากฏรหัสอักษร A แล้วตามด้วยตัวเลขต่อท้ายเบอร์ 68HC11 เป็นการบ่งบอกว่า 68HC11 ในเวอร์ชันนี้มีรอมหรืออีอีพรอมอยู่หรือไม่ ถ้ามีขนาดเท่าไร ในอนุกรมนี้มีด้วยกัน 3 เบอร์ คือ 68HC11A8, 68HC11A1 และ 68HC11A0

2) อนุกรม B

จะมีอักษร B และตัวเลขต่อท้ายเบอร์ 68HC11 ส่วนรหัสอักษร 2 ตัวหน้า คือ XC68HC11 อนุกรมนี้ เป็นเวอร์ชันทดลองใช้งานรุ่นแรก ดังนั้นในการโปรแกรมอีอีพรอมในชิปต้องจ่ายแรงดัน 19 โวลต์ จากแหล่งจ่ายไฟภายนอกให้แก่ชิปด้วย

3) อนุกรม E

จะมีอักษร E และตัวเลขต่อท้ายเบอร์68HC11 ในอนุกรมนี้มีด้วยกัน 4 เบอร์ คือ 68HC11E9, 68HC11E2, 68HC11E1 และ 68HC11E0 โดย 68HC11 ในอนุกรมนี้ได้รับการพัฒนาขีดความสามารถ เพิ่มมากขึ้น ได้แก่ มีอินพุตสำหรับตรวจจับสัญญาณสำหรับตัวตั้งเวลา 4 ชุด มีขนาดของหน่วยความจำรอม และแรมภายในมากกว่า โดยมีขนาดของรอมสูงถึง 12 กิโลไบต์ ในเบอร์ 68HC11E9 และขนาดของแรม 512 ไบต์ เสริมด้วยเอาต์พุตของฟังก์ชันการเปรียบเทียบอีก 5 ชุด

4) อนุกรม D

68HC11 ในอนุกรมนี้เป็นรุ่นราคาถูก จะไม่มีวงจรแปลงสัญญาณแอนะลอกเป็นดิจิตอล ทำให้ 68HC11 อนุกรมนี้มีเพียง 40 ขา และขนาดของหน่วยความจำภายในก็น้อยกว่าในอนุกรมอื่นๆ โดยมีเพียง 4 กิโลไบต์สำหรับรอม และ 192 ไบต์ สำหรับแรม

2.2.3 แผนผังการทำงานภายในของไมโครคอนเลอร์ 68HC11

ไมโครคอนโทรลเลอร์ 68HC11 ที่จะนำมาใช้อ้างอิงเพื่อนำเสนอคือ เบอร์ MC68HC11A8 โดย มีแผนผังการทำงานส่วนประกอบภายในของ MC68HC11A8 ภายในชิปประกอบด้วยส่วนหลัก ๆ 6 ส่วน คือ

1) ซีพียู

2) หน่วยความจำ

3) แอกคิวมูเลเตอร์-วอตช์ด็อก-ตัวตั้งเวลาหลัก

4) พอร์ตอินพุต เอาต์พุต

5) วงจรแปลงสัญญาณแอนะลอกเป็นดิจิตอล

6) สโตรบ/แฮนด์เชก

1) ซีพียู ในส่วนนี้มีส่วนประกอบย่อยอีก 3 ส่วนคือ ส่วนควบคุมโหมดการทำงานของชิป (MODE) ส่วนกำเนิดสัญญาณนาฬิกา (CLK) และส่วนลอจิกอินเตอร์รัปต์ (INT) โดยทั้งสามส่วนนี้จะได้รับการควบคุมการทำงานจากแก่นซีพียู

2) หน่วยความจำ ภายในชิป MC68HC11A8 จะมีหน่วยความจำครบทั้ง 3 แบบ คือ รอมแรม และ อีอีพรอม ซึ่งการต่อและเรียกใช้หน่วยความจำเบอร์นี้จะขึ้นอยู่กับส่วนซีพียูเป็นหลัก

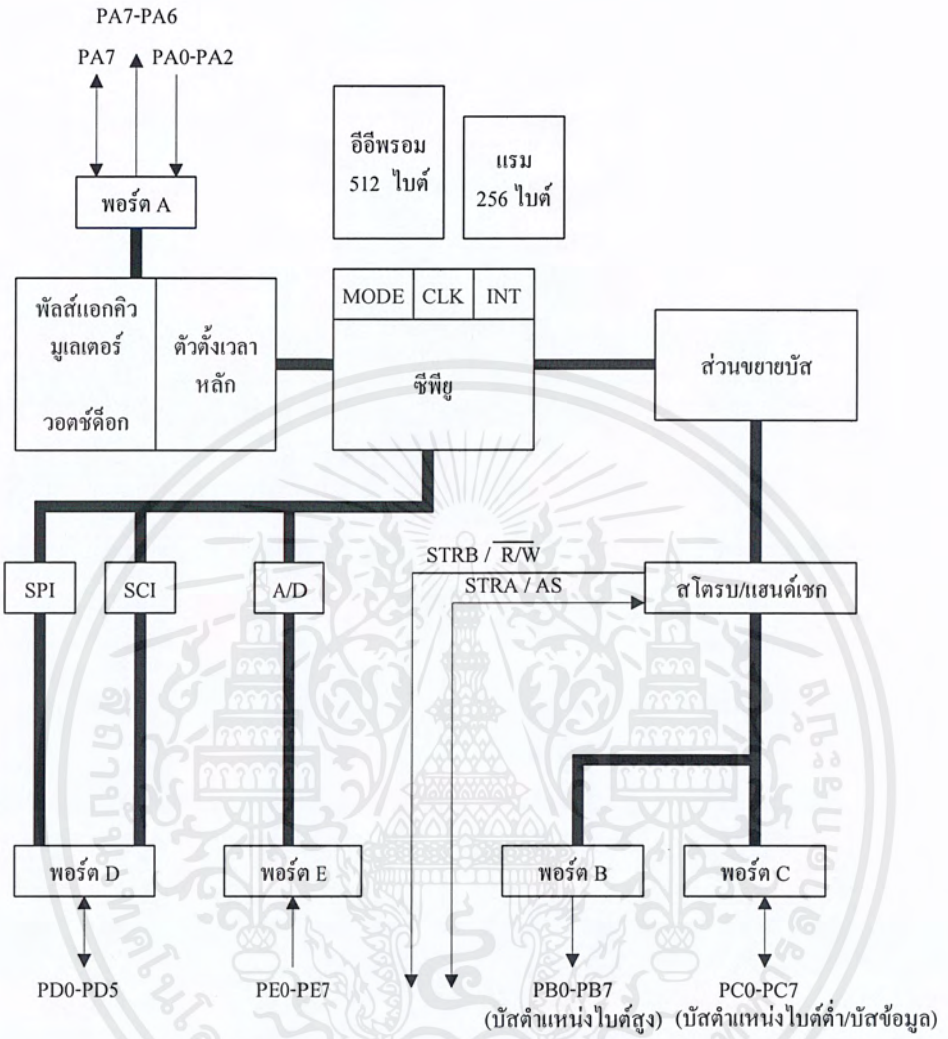
3) แอกคิวมูเลเตอร์-วอตช์ด็อก-ตัวตั้งเวลาหลัก ในส่วนนี้จะมีการติดต่อกับพอร์ต A โดยใช้พอร์ต A เป็นทางผ่านของข้อมูล พัลส์แอกคิวมูเลเตอร์จะใช้พอร์ต PA7 ในขณะที่ส่วนตัวตั้งเวลาหลักจะใช้ PA3-PA6 ในชิปยังมีวงจรวอตช์ด็อก เพื่อช่วยให้ชิปสามารถทำงานได้อย่างต่อเนื่อง

4) พอร์ตอินพุตเอาต์พุต ใน MC68HC11A1 จะมีพอร์ตอินพุตเอาต์พุตด้วยกัน 5 พอร์ต ดังนี้

พอร์ต A เป็นทั้งพอร์ตอินพุตและเอาต์พุต โดยมีการทำงานแยกกันคือ PA7 สามารถส่งผ่านข้อมูลได้สองทิศทาง ในขณะที่ PA3-PA6 เป็นพอร์ตเอาต์พุตของตัวตั้งเวลาหลักและ PA0 - PA2 เป็นพอร์ตอินพุต

พอร์ต B เป็นพอร์ตเอาต์พุต ในขณะที่พอร์ต C เป็นพอร์ตอินพุตเอาต์พุต สามารถส่งข้อมูลได้ 2 ทิศทางโดยที่พอร์ต C นี้จะทำหน้าที่เป็นบัสตำแหน่งไบต์ต่ำ และบัสข้อมูลด้วยโดยได้รับการควบคุมการทำงานแบบมัลติเพล็กซ์

พอร์ต D เป็นพอร์ตที่ใช้ในการถ่ายทอดสัญญาณจากส่วนเชื่อมต่ออุปกรณ์อนุกรมและส่วนสื่อสารข้อมูลอนุกรม จึงมีลักษณะเป็นพอร์ตอินพุตเอาต์พุตที่สามารถส่งผ่านข้อมูลได้สองทิศทางในขณะที่พอร์ต E เป็นพอร์ตอินพุตสำหรับวงจรแปลงสัญญาณแอนะลอกเป็นดิจิตอล (Analog Digital Converter)

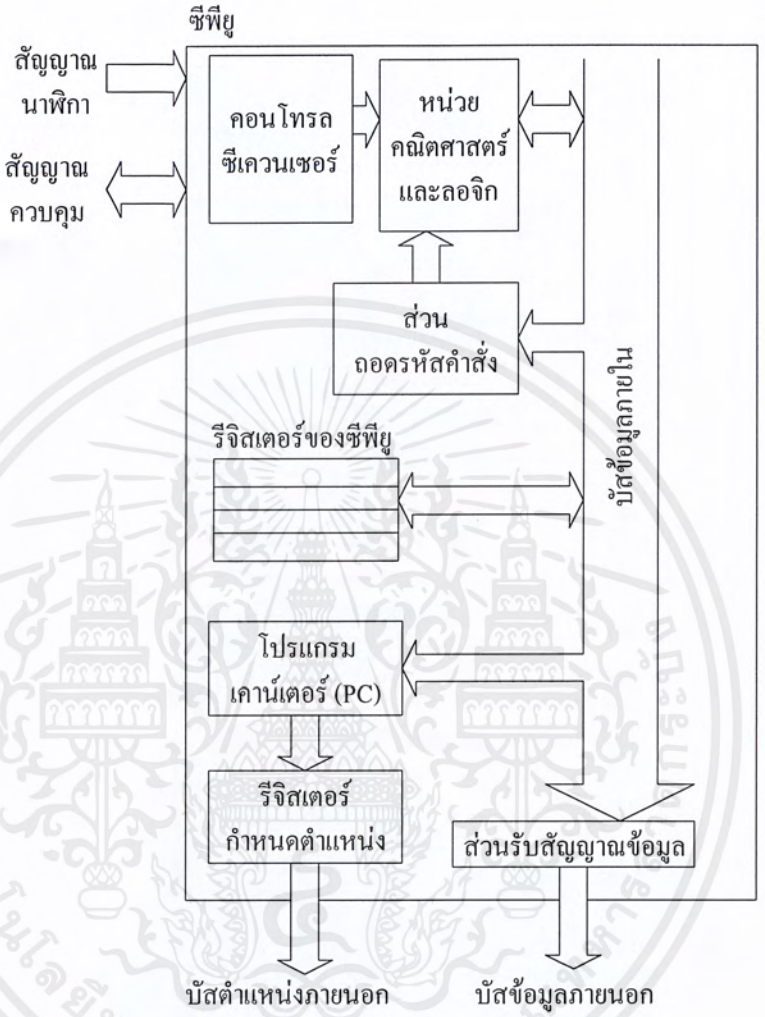


รูปที่ 2.1 แผนผังการทำงานภายในของ MC68HC11A8

5) วงจรแปลงสัญญาณแอนะลอกเป็นดิจิตอล (ADC) เป็นวงจรที่ช่วยเสริมให้ชิปไมโครคอนโทรลเลอร์มีประสิทธิภาพมากขึ้น ปกติแล้ว 68HC11A8CFN1 ก็จะมีวงจร ADC ครบ 8 ช่อง การเรียกใช้งานวงจรส่วนนี้จะได้รับการควบคุมจากซีพียู

6) สโตรบ/แฮนด์เชก ในส่วนนี้จะทำงานร่วมกับส่วนขยายบัสด้วยทั้งนี้เพื่อให้ซีพียูสามารถทำงานได้กับบัสตำแหน่งถึง 16 บิต ที่ส่วนนี้จะมีสัญญาณที่สำคัญๆ อยู่ 2 สัญญาณคือ STRB/R/W และ STRA/AS โดยทั้งสองสัญญาณคือ สัญญาณสโตรบเพื่อให้ชิปทำการอ่านเขียนข้อมูลได้ และที่ส่วนนี้ ยังมีวงจรแฮนด์เชกเพื่อตรวจสอบความพร้อม ในการรับส่งข้อมูลของชิป กับอุปกรณ์ภายนอก

2.2.4 การทำงานของซีพียูภายใน 68HC11

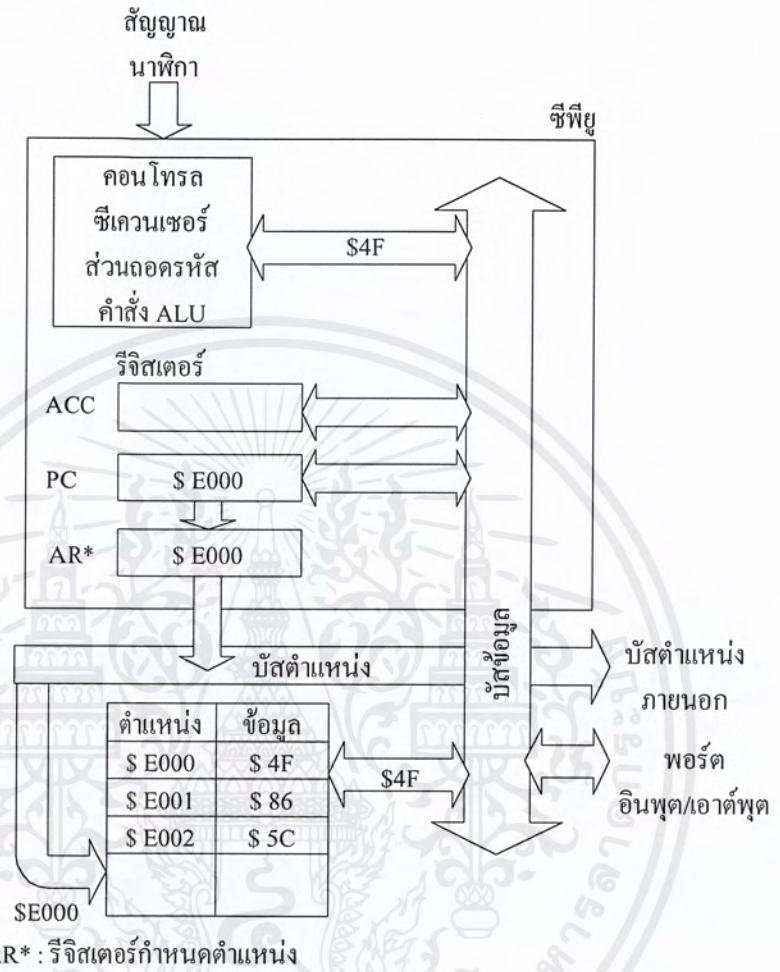


รูปที่ 2.2 แผนผังการทำงานของซีพียู

จากรูปที่ 2.2 จะเห็นว่าซีพียูจะมีรีจิสเตอร์ไว้ใช้งานเป็นของตนเองอยู่หลายตัว รีจิสเตอร์โปรแกรมเคาน์เตอร์ (Program Counter Register PC) จะเป็นตัวบอกให้ซีพียูทราบว่าขณะนี้กำลังจัดการกับคำสั่งหรือข้อมูลที่ตำแหน่งใด ส่วนรีจิสเตอร์ตัวอื่นๆ ก็จะมีหน้าที่จัดการและเก็บข้อมูลหรือ กำหนดตำแหน่งแตกต่างกันออกไป

ส่วนถอดรหัสคำสั่ง จะทำหน้าที่ถอดรหัสคำสั่งที่ได้รับจากบัสข้อมูล เพื่อกำหนดให้ส่วนคณิตศาสตร์และลอจิก (ALU) จัดการกับข้อมูล ขณะที่ส่วนคอนโทรลซีเควนเซอร์ จะทำหน้าที่กำหนดจังหวะในการถ่ายทอดคำสั่งและข้อมูล ไปยังบัสข้อมูลภายในซีพียู ส่วนรีจิสเตอร์กำหนด

ตำแหน่งทำหน้าที่กำหนดสภาวะของบัสตำแหน่ง



รูปที่ 2.3 ลักษณะการทำงานที่เกิดขึ้นภายในซีพียู เมื่อทำการเฟตช์คำสั่ง CLRA

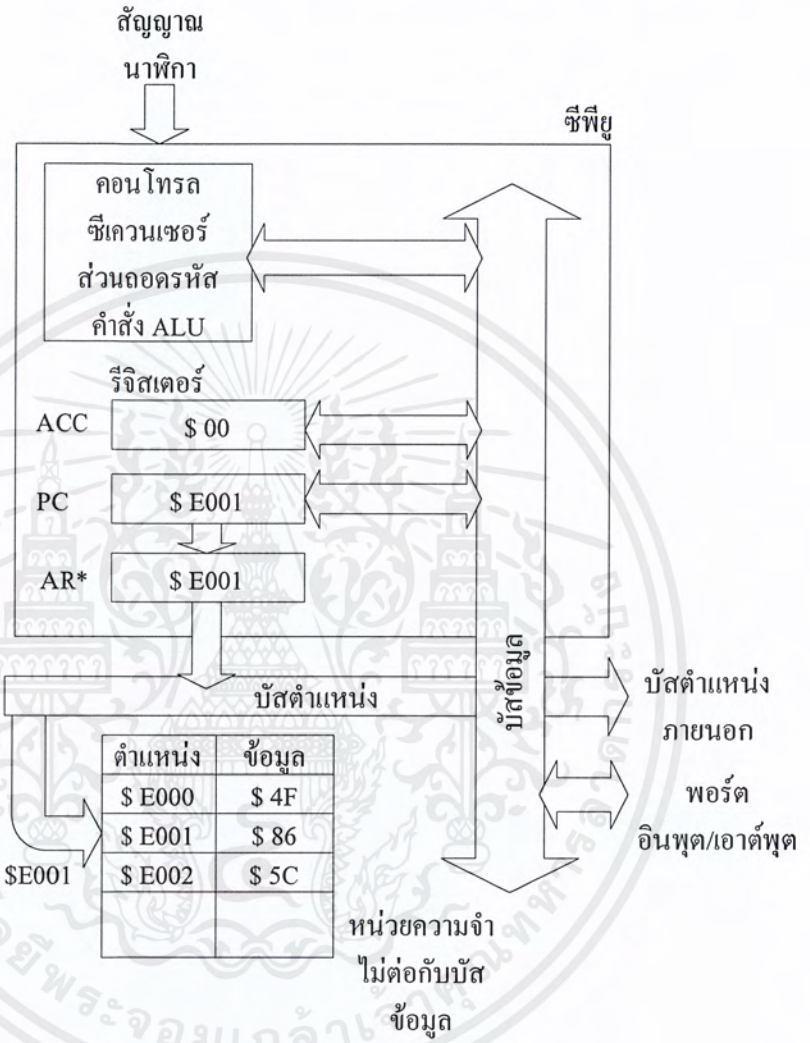
บัสตำแหน่งภายนอกจะเป็นตัวเลือกตำแหน่งของหน่วยความจำเพื่อเรียกหรือถ่ายทอดข้อมูล ส่วนรับสัญญาณข้อมูล จะเป็นตัวกำหนดสภาวะของสัญญาณข้อมูล ที่รับเข้าหรือส่งออกไปยังหน่วยความจำหรือรีจิสเตอร์เอาต์พุต ลักษณะการทำงานของซีพียูสามารถอธิบาย โดยใช้ตัวอย่างประกอบดังนี้

สมมติเขียนโปรแกรมลงในหน่วยความจำตำแหน่ง $E000-$E002 ดังนี้

CLRA ; เป็นการเคลียร์ค่าในแอกคิวมูเลเตอร์ให้เป็น $00

LDAA $5C ; นำค่า $5C ไปเก็บไว้ในแอกคิวมูเลเตอร์ ดังนั้นที่ตำแหน่ง $E000 จะเก็บรหัสคำสั่งของคำสั่ง CLRA คือ $4F ในขณะที่ตำแหน่ง $E001 เก็บรหัสคำสั่งของคำสั่ง LDAA คือ

$86 ส่วนตำแหน่ง $E002 เก็บค่าตัวดำเนินการของคำสั่ง LDAA คือ $5C ในกระบวนการทำงานของซีพียูจะแบ่งเป็นช่วงเฟตช์และเอ็กซีคิวต์ช่วงเฟตช์ คือ ช่วงที่ซีพียูทำงานตามคำสั่งนั้นๆ

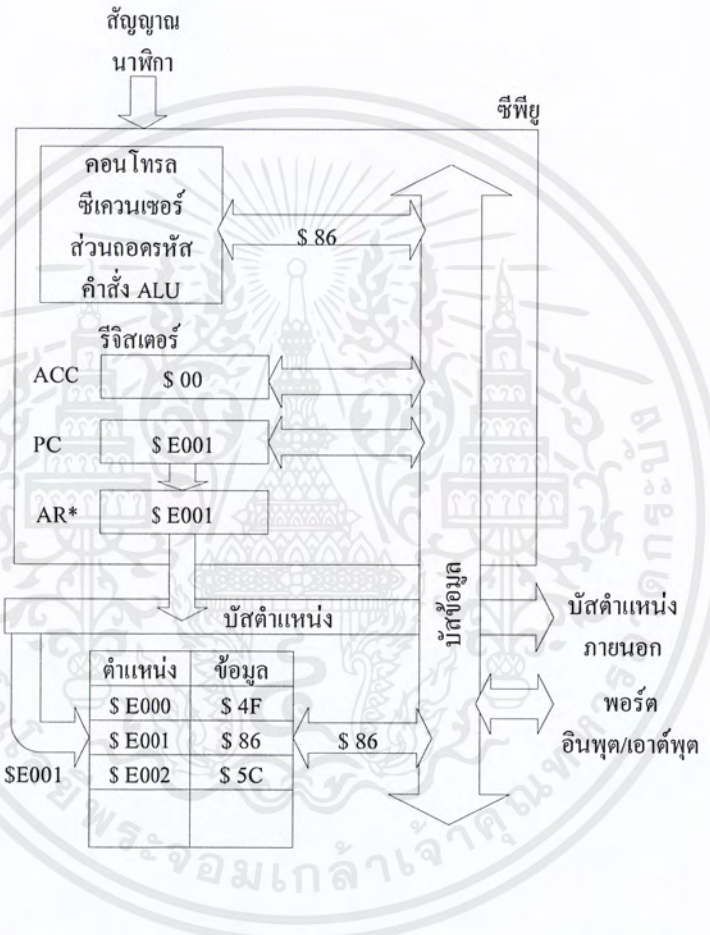


รูปที่ 2.4 การทำงานของซีพียูเมื่อทำการเอ็กซีคิวต์คำสั่ง CLRA

ในรูปที่ 2.3 แสดงการทำงานในช่วงเฟตช์ข้อมูล $4F ซึ่งเป็นรหัสคำสั่งของคำสั่ง CLRA รีจิสเตอร์ PC จะเป็นคำสั่ง $E000 นั่นคือ ตำแหน่งที่ซีพียูไปทำงาน และค่า $E000 นี้จะถูกส่งต่อไปยังรีจิสเตอร์กำหนดตำแหน่งด้วย เพื่อกำหนดตำแหน่งของหน่วยความจำที่จะไปติดต่อด้วย ถึงตอนนี้ส่วนคอนโทรลซีเควนซ์จะทำการดึงข้อมูล $4F เข้าไปในส่วนถอดรหัสคำสั่งจะรู้ว่าเป็นคำสั่งทำให้ค่าแอกคิวมูเลเตอร์เป็น $00

ในรูปที่ 2.4 จะแสดงสภาวะหลังจากที่ซีพียูเอ็กซีคิวต์ คำสั่ง CLRA แล้ว จะเห็นว่าค่าใน

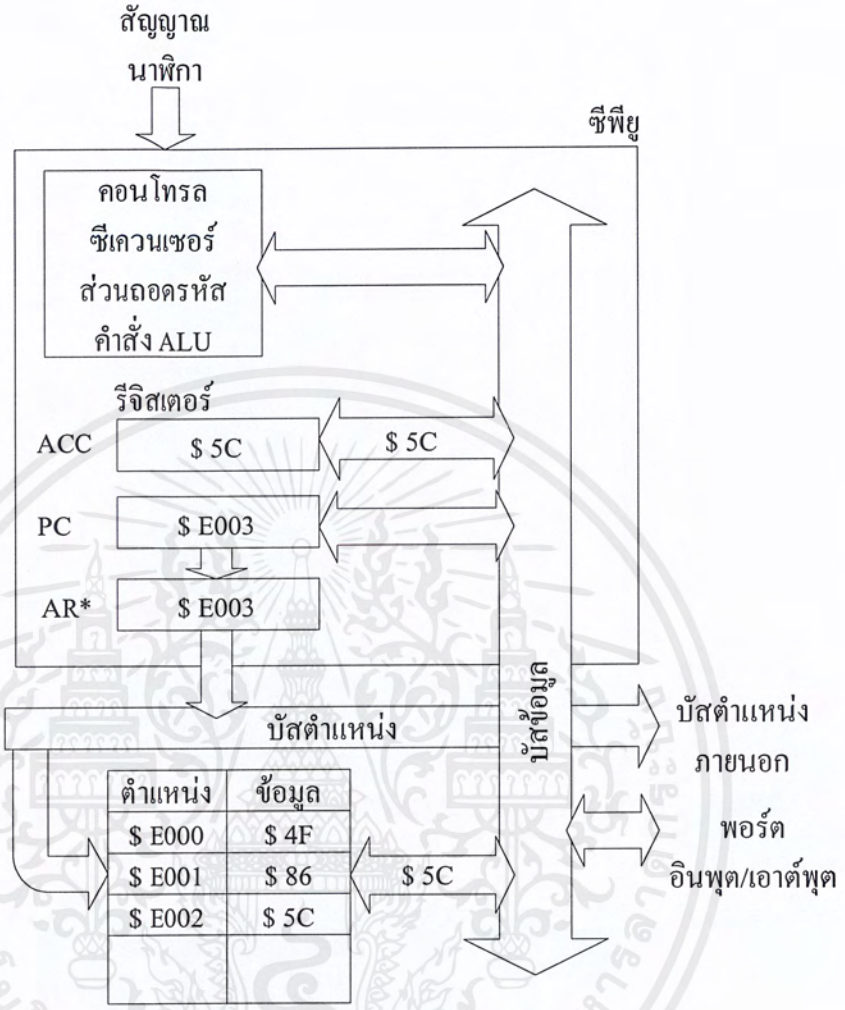
แอกคิวมูเลเตอร์จะเป็น $00 ในขณะที่ PC เพิ่มค่าไปเป็น $E001 ซึ่งก็คือตำแหน่งต่อไปที่ซีพียูจะไปทำงาน เช่นเดียวกันค่า $E001 จะถูกเก็บไว้ในรีจิสเตอร์กำหนดตำแหน่งด้วยเพื่อเตรียมกำหนดค่าตำแหน่งต่อไปหลังจากเอ็กซีคิวต์คำสั่งไปแล้วหน่วยความจำก็จะไม่เชื่อมต่อเข้ากับบัสข้อมูลอีก จนกว่าจะมีการเฟตช์คำสั่งใหม่



รูปที่ 2.5 การทำงานของซีพียู เมื่อทำการเฟตช์ข้อมูลของรหัสคำสั่ง LDAA

จากนั้นซีพียูจะทำงานในคำสั่งต่อไปคือ คำสั่ง LDAA #$5C ก็เริ่มจากเฟตช์ข้อมูล $86 ก็คือรหัสคำสั่งของคำสั่ง LDAA จากหน่วยความจำตำแหน่ง $E001 ดังในรูปที่ 2.5 เมื่อเอ็กซีคิวต์แล้วก็จะไปอ่านค่าข้อมูลตัวดำเนินการของคำสั่ง LDAA คือ $5C ที่ตำแหน่ง $E002 ต่อด้วยคำสั่ง LDAA #$5C คือ การนำค่า $5C เข้าไปเก็บในแอกคิวมูเลเตอร์ จากรูปที่ 2.6 เป็นการแสดงสภาวะข้อมูล

ภายในซีพียูค่า $5C จะถูกเก็บเข้าไปในแอกคิวมูเลเตอร์ หลังจากที่เอ็กซีคิวต์คำสั่งแล้วค่า PC จะเพิ่มเป็น $E003 เพื่อเตรียมทำงานที่ตำแหน่งต่อไป



รูปที่ 2.6 หลังจากที่ซีพียูเอ็กซีคิวต์คำสั่ง LDAA

2.2.5 โหมดการทำงานของไมโครคอนโทลเลอร์ 68HC11

68HC11 มีโหมดการทำงาน 4 โหมด ดังนี้

1) โหมดชิปเดี่ยว (Single-chip Operating Mode)

2) โหมดมัลติเพล็กซ์ขยาย (Expanded Multiplexed Operating Mode)

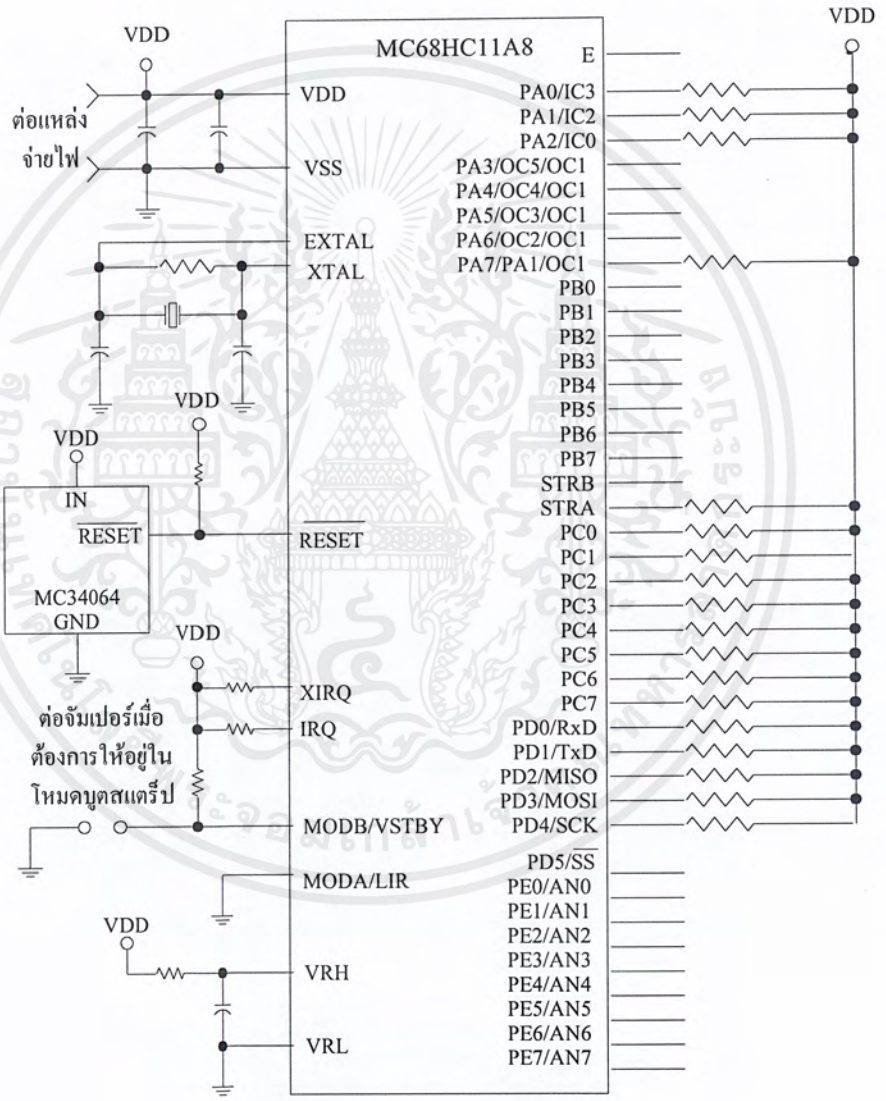
3) โหมดบูตสแตร์ปพิเศษ (Special Bootstrap Operating Mode)

4) โหมดทดสอบพิเศษ (Special test Operating Mode)

1) โหมดชิปเดี่ยว

โหมดชิปเดี่ยว ในโหมดนี้ 68HC11 จะทำงานโดยลำพังตัวเดียว จากที่ได้กล่าวมาแล้ว

ว่าภายใน 68HC11 มีหน่วยความจำและพอร์ตถึง 5 พอร์ต ดังนั้นเมื่อทำงานโหมดชิปเดี่ยวตัวซีพียูก็จะทำการเรียกข้อมูลที่อยู่ในหน่วยความจำภายใน ออกมาใช้งานและใช้พอร์ตที่มีอยู่รับหรือส่งค่าออกไป ดังนั้นในโหมดนี้ 68HC11 จะใช้หน่วยความจำและพอร์ตภายในชิป ทำให้ไม่สามารถเพิ่มหน่วยความจำหรือพอร์ตเพื่อขยายระบบได้ การต่อวงจรของ 68HC11 เมื่อทำงานในโหมดนี้แสดงดังในรูปที่ 2.7

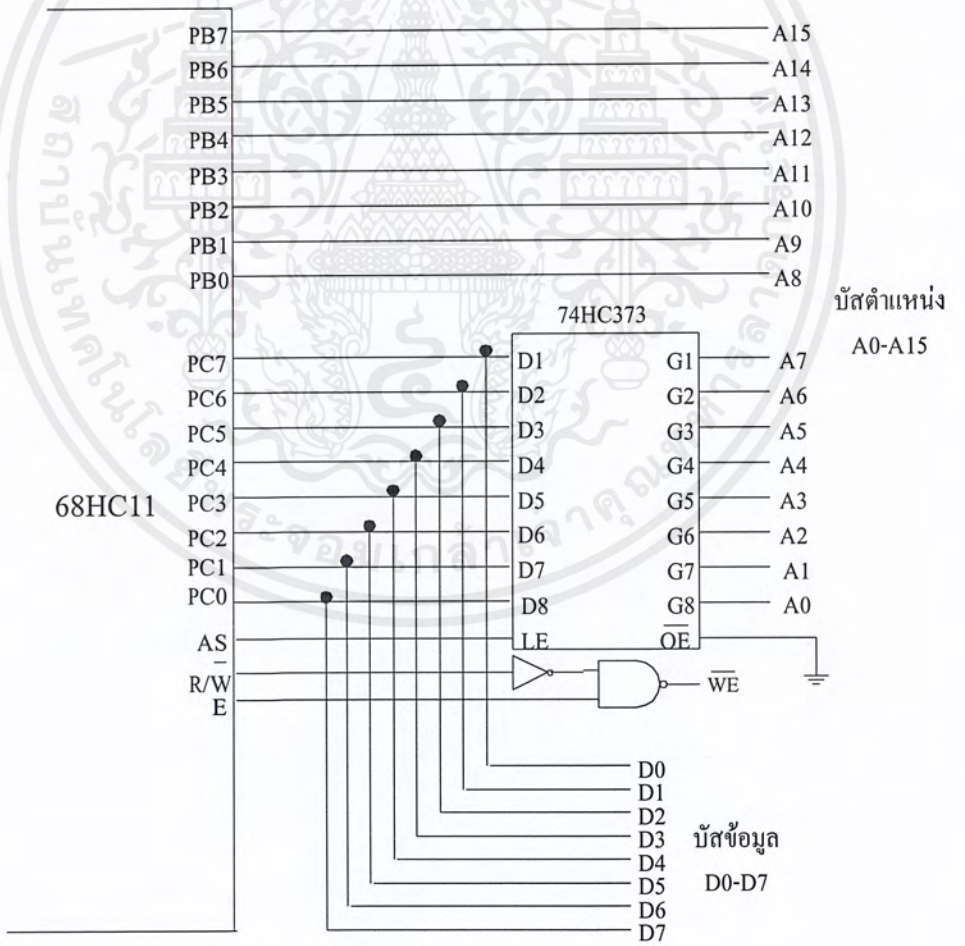


รูปที่ 2.7 วงจรการต่อ 68HC11 เมื่อทำงานในโหมดชิปเดี่ยว

2) โหมดมัลติเพล็กซ์ขยาย

การทำงานของ 68HC11 ในโหมดนี้ จะแตกต่างกับโหมดชิปเดี่ยวโดยโปรแกรมควบคุมการทำงานของซีพียูจะอยู่ในหน่วยความจำภายนอก โดยที่พอร์ต B และ C ของ 68HC11 ทำหน้าที่เป็นบัสตำแหน่งและข้อมูลสังเกตจากแผนผังการทำงานภายในของ 68HC11 ในรูปที่ 2.1 พอร์ต B ทำหน้าที่เป็นบัสตำแหน่งไบต์สูง ส่วนพอร์ต C ทำหน้าที่เป็นบัสตำแหน่งไบต์ต่ำและบัสข้อมูล การที่พอร์ต C จะสามารถเป็นทั้งบัสตำแหน่งและบัสข้อมูลได้นั้น ต้องใช้กระบวนการที่เรียกว่า การมัลติเพล็กซ์ (Multiplexed) การแยกสัญญาณตำแหน่งและข้อมูลมาใช้งาน จะต้องต่อวงจรเพื่อทำการดีมัลติเพล็กซ์ภายนอก ดังมีตัวอย่างวงจรตามรูปที่ 2.8

สัญญาณควบคุมที่นำมาใช้ดีมัลติเพล็กซ์สัญญาณตำแหน่งและข้อมูล ได้แก่ สัญญาณ STROBE /R/W (STRB/R/W) และสัญญาณ STROBE A/ADDRESS STROBE (STRA/A)



รูปที่ 2.8 วงจรดีมัลติเพ็กซ์

ในโหมดนี้ 68HC11 จะสามารถติดต่อกับหน่วยความจำภายนอกเพิ่มได้สูงสุดถึง 84 กิโลไบต์ แต่ก็มีข้อเสีย คือต้องใช้อุปกรณ์ต่อเพิ่มมากไม่ว่าจะเป็นหน่วยความจำชิปพอร์ตอินพุตเอาต์พุต ทำให้ค่าใช้จ่ายของระบบสูงขึ้น แต่อย่างไรก็ตาม โหมดนี้เป็นโหมดที่นิยมใช้งานมากที่สุด เพราะสามารถต่อขยายหน่วยความจำและพอร์ตได้มากมายนั่นเอง

3) โหมดบูตสแตร์ปพิเศษ

โหมดบูตสแตร์ปพิเศษเป็นโหมดการทำงานที่คล้ายกับแบบชิปเดี่ยวคือตัวไมโครคอนโทรลเลอร์สามารถทำงานได้โดยลำพัง แต่จะมีข้อพิเศษกว่าแบบซิงเกิลชิปอยู่หลายประการ สามารถเขียนโปรแกรมแล้วโหลดเข้าไปในแรมภายในชิปได้เลย โปรแกรมที่ใช้ในการบูตจะถูกบรรจุไว้ในบูตสแตร์ปรอมจำนวน 192 ไบต์ รอมนี้จะสามารถเรียกข้อมูลออกมาใช้งานได้ก็ต่อเมื่อ 68HC11 ทำงานในโหมดบูตสแตร์ปนี้เท่านั้น โดยโปรแกรมบูตนี้จะเก็บไว้ที่ตำแหน่ง $BF40-$BFFF เมื่อเรียกใช้งานโปรแกรมบูต ส่วน SCI (Serial Communication Interface) จะส่งข้อมูลโปรแกรมไปยังแรมภายในชิปที่ตำแหน่ง $0000-$00FF หลังจากที่ส่งข้อมูลที่ตำแหน่ง $00FF เรียบร้อยแล้ว ซีพียูจะเริ่มมาทำงานที่ตำแหน่ง $0000 ต่อไป ในโหมดการทำงานแบบนี้ 68HC11 จะส่งผ่านข้อมูลผ่านพอร์ต SCI หลังจากที่ทำการรีเซตซีพียูแล้ว ส่วน SCI จะทำงานที่สัญญาณนาฬิกา E/16 (หรือมีอัตราบอดเท่ากับ 7,812 สำหรับสัญญาณนาฬิกาที่มีความถี่ 2 เมกะเฮิรตซ์) ในโหมดนี้ยังมีลักษณะการทำงานที่ค่อนข้างพิเศษอีกประการหนึ่งคือ การป้องกันการคัดลอก โดยจะมีบิตป้องกัน (Security bit) เข้ามาเกี่ยวข้อง ถ้าหากบิตป้องกันนี้ถูกเซตเป็น "1" ที่เอาต์พุตของส่วน SCI จะส่งค่า $FF ออกไป ในทางตรงข้ามถ้าไม่มีการเซตบิตป้องกัน ข้อมูลที่เป็นการหยุดการทำงานวงจรชิปจะถูกส่งออกไปภายนอก โดยผ่านส่วน SCI ผู้ใช้งานต้องทำการดาวน์โหลดข้อมูลโปรแกรม 256 ไบต์ ไปยังแรมภายใน โดยให้เริ่มทำงานที่แอดเดรส $0000 ถ้าทุกอย่างเรียบร้อย ก็จะไปเริ่มทำงานที่แอดเดรส $0000 เพื่อเตรียมทำงาน อินเตอร์รัปต์เวกเตอร์ของ 68HC11 เมื่อถูกใช้งานในโหมดนี้สามารถเข้าถึงหน่วยความจำแรมได้เร็วขึ้น การใช้อินเตอร์รัปต์จะทำให้ผู้ใช้งานสามารถกระโดดไปทำงานในส่วนอื่นได้ อินเตอร์รัปต์เวกเตอร์ที่มีใน 68HC11 ถ้าหากต้องการใช้งานอินเตอร์รัปต์ SWI ก็ต้องระบุคำสั่งกระโดด เพื่อให้ซีพียูสามารถเรียกใช้งานได้ในแรมที่ตำแหน่ง $00F4, $00F5 และ $00F6 เมื่อเกิดอินเตอร์รัปต์ SWI ขึ้น ค่าเวกเตอร์นี้จะเป็นตัวบอกให้ซีพียูทำการอ่านข้อมูล เพื่อประมวลผลที่ $00F4 ในแรม และที่นั่นก็จะมีคำสั่ง JUMP เพื่อให้ซีพียูสามารถตอบสนองการอินเตอร์รัปต์ได้

อีกลักษณะพิเศษหนึ่ง คือ สามารถที่จะเข้าไปทำงานที่รอมแอดเดรส $0000 ได้ โดยไม่ต้องทำการดาวน์โหลดก่อน วิธีการนี้จะใช้ได้กับเฉพาะเมื่อใช้สัญญาณนาฬิกา E/16 เริ่มด้วยการส่ง $55

ไปแทนค่า $FF ด้วยคำสั่งดังกล่าวนี้ จะทำให้ซีพียูกระโดดไปทำงานที่แอดเดรส $0000 โดยไม่ต้องผ่านการดาวน์โหลด

4) โหมดทดสอบพิเศษ

โหมดทดสอบพิเศษเป็นโหมดการทำงานที่โรงงานผู้ผลิตเป็นผู้กำหนดการทำงาน ในโหมดนี้จะคล้ายคลึงกับโหมดมัลติเพล็กซ์ขยาย การรีเซตและอินเตอร์รัปต์เวกเตอร์จะได้รับการเฟตช์จากหน่วยความจำภายนอกที่ตำแหน่ง $BFC0-$BFFF มากกว่าที่ตำแหน่ง $FFC0-$FFFF รีจิสเตอร์ TEST1 จะได้รับการอนาเบิลเพื่อเป็นการบ่งบอกให้ทราบว่า ขณะนี้ซีพียูพร้อมรับการตรวจสอบแล้ว โหมดการทำงานนี้เป็นโหมดที่ผู้ใช้งานปกติไม่ควรเลือกใช้ เพราะจะทำให้ระบบป้องกันข้อมูลภายในชิปด้อยลงไป

2.2.6 รายละเอียดสัญญาณและการจัดขาของ 68HC11

รายละเอียดของขาสัญญาณทั้งหมดมีดังนี้

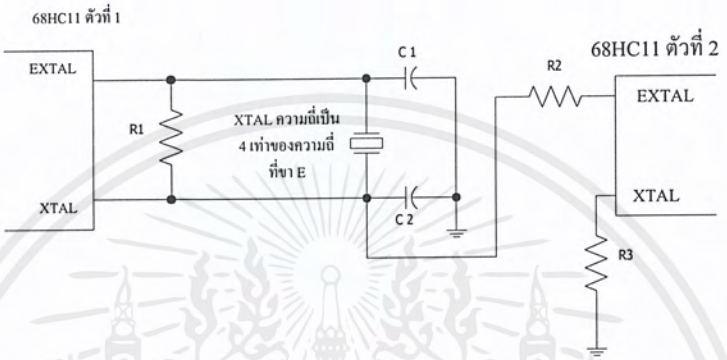
1) ขาไฟเลี้ยงและกราวด์ (Vdd, Vss) เป็นขาที่ใช้สำหรับจ่ายแรงดันเพื่อให้ 68HC11 ทำงาน โดยขา Vdd ต้องป้อนแรงดัน +5 โวลต์ ในขณะที่ขา Vss เป็นกราวด์ เนื่องจาก 68HC11 มีโครงสร้างมาจากอุปกรณ์จำพวกซีมอส ดังนั้นจะต้องระมัดระวังอย่างมาก ในการจ่ายแรงดันไฟเลี้ยงแหล่งจ่ายไฟ ต้องมีคุณภาพค่อนข้างดี แรงดันคงที่ และต้องต่อตัวเก็บประจุแบบเซรามิกค่า 0.1 ไมโครฟารัด คร่อมระหว่างขา Vdd และ Vss และต้องติดตั้งตัวเก็บประจุนี้ให้ใกล้ขาใดขาหนึ่งมากที่สุด

2) ขารีเซต เป็นขาอินพุตรับสัญญาณเพื่อให้ 68HC11 เริ่มต้นทำงานใหม่อันอาจเกิดจากสภาวะที่ซีพียูเพิ่งได้รับไฟเลี้ยงให้เริ่มทำงาน หรือเกิดจากการทำงานภายในวงจรเกิดความผิดพลาดแล้วถูกตรวจจับได้ โดยวงจรวอตช์ด็อกจากนั้น วงจรวอตช์ด็อกก็จะป้อนสัญญาณมาที่ขานี้เพื่อ

รูปที่ 2.9 การจัดวงจรเพื่อต่อคริสตอลให้แก่ 68HC11

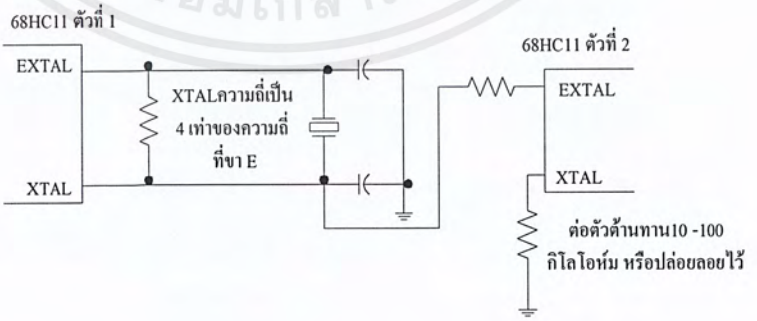
กระตุ้นให้ซีพียูเริ่มทำงานใหม่ โดยสัญญาณที่ป้อนเข้าขานี้ต้องมีสภาวะลอจิก “0” สัญญาณรีเซต

ถ้าหากใช้วงจรกำเนิดสัญญาณนาฬิกาภายนอก ขา XTAL ต้องปล่อยไว้ หรือต่อตัวต้านทานค่าตั้งแต่ 10 กิโลโอห์ม - 100 กิโลโอห์มลงกราวด์ แล้วป้อนสัญญาณนาฬิกาเข้าที่ขา EXTAL โดยความถี่ที่ป้อนนี้ต้องสูงกว่าที่ขา E (ซึ่งจะอธิบายถึงขานี้ในภายหลัง) ประมาณ 4 เท่า ดังใน รูปที่ 2.10



รูปที่ 2.10 การต่อวงจรกำเนิดสัญญาณนาฬิกาภายนอกให้แก่ 68HC11

ในการต่อวงจรเพื่อกำเนิดสัญญาณนาฬิกานั้น ควรต่อบัฟเฟอร์ที่มีอิมพีแดนซ์สูง เช่น ใช้ไอซีเบอร์ 74HC04 ไว้สำหรับในกรณีที่ต้องการต่อวงจรกำเนิดสัญญาณนาฬิกานี้จ่ายให้แก่ไมโครคอนโทรลเลอร์ตัวอื่นด้วย อย่างไรก็ดี ถ้าหากต้องการต่อวงจรกำเนิดสัญญาณนาฬิกาเข้ากับไมโครคอนโทรลเลอร์เพียง 2 ตัว สามารถทำได้โดยต่อสัญญาณจากขา XTAL ผ่านตัวต้านทาน 220 โอห์ม เข้าที่ขา EXTAL ของไมโครคอนโทรลเลอร์อีกตัวหนึ่งก็ได้ ดังในรูปที่ 2.11



รูปที่ 2.11 การต่อวงจรกำเนิดสัญญาณนาฬิกาภายนอกให้แก่ไมโครคอนโทรลเลอร์ 2 ตัว

อีกประการหนึ่งที่ต้องคำนึงถึงในการออกแบบแผ่นวงจรพิมพ์ในส่วนที่เกี่ยวข้องกับขาทั้งสองนี้ ค่าความจุของตัวเก็บประจุที่แสดงในวงจรรูปที่ 2.9 และ 2.11 เป็นค่าที่รวมกับค่าความจุแฝงที่เกิดขึ้นบนแผ่นวงจรพิมพ์ด้วย ดังนั้นการเลือกใช้ค่าตัวเก็บประจุต้องใช้ค่าที่ลดลงจากวงจรที่แสดงเล็กน้อย

4) ขา E เป็นขาเอาต์พุตของวงจรกำเนิดสัญญาณนาฬิกาภายในชิป สามารถใช้เป็นฐานเวลาอ้างอิงได้ โดยความถี่ที่ออกจากขานี้จะมีค่า 1/4 เท่าของความถี่อินพุตที่ขา XTAL และ EXTAL เมื่อใดที่ขา E นี้มีสภาวะลอจิกเป็น "0" หมายความว่า ขณะนี้ซีพียูกำลังทำการประมวลผลภายในอยู่ ถ้าหากเป็น "1" หมายถึง ขณะนี้ข้อมูลได้รับการเรียกใช้จากซีพียู ถ้าหากไมโครคอนโทรลเลอร์อยู่ในโหมด STOP สัญญาณที่ขา E จะไม่มีออกมาหรือเกิดสภาวะ High Impedance นั่นเอง

5) ขา IRQ (Interupt Request) เป็นขาอินพุตสำหรับเรียกการอินเตอร์รัปต์แบบอะซิงโครนัสของชิป 68HC11 โดยการรับสัญญาณเพื่ออินเตอร์รัปต์นี้สามารถจะเลือกได้ว่า จะรับสัญญาณขอบขาลง (Negative Edge) หรือจะรับการทริกเป็นแบบระดับสัญญาณ โดยปกติจะกำหนดให้รับการอินเตอร์รัปต์ด้วยการทริกแบบระดับสัญญาณ นั่นคือ ทำงานที่ระดับลอจิก "0" เมื่อต่อใช้งานปกติต้องต่อตัวต้านทานค่า 4.7 กิโลโอห์มพูลอัปเข้ากับไฟเลี้ยง

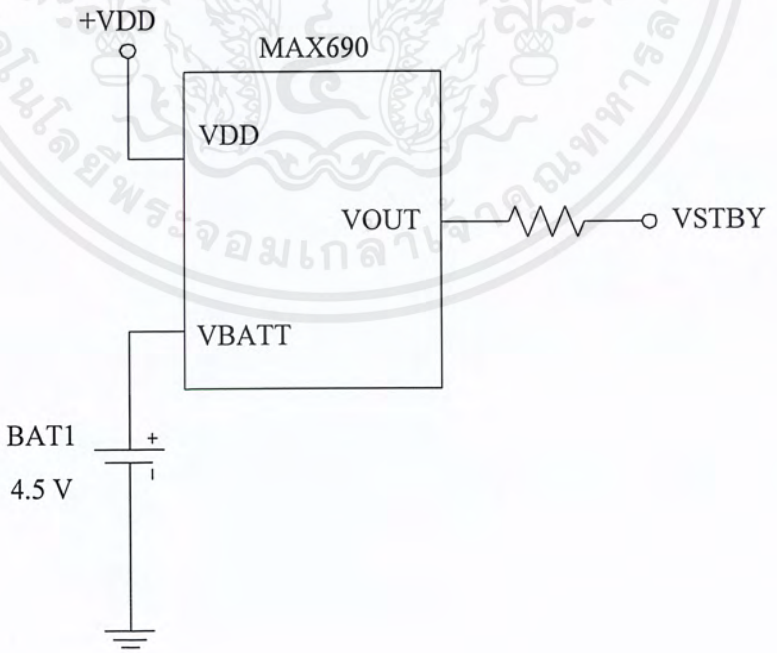
6) ขา XIRQ (Non-Maskable Interrupt) เป็นขาอินพุตสำหรับตอบรับการอินเตอร์รัปต์แบบนอนมาสเคเบิลหลังจากที่มีการรีเซตซีพียู สามารถรับการอินเตอร์รัปต์ด้วยการทริกแบบระดับสัญญาณ โดยทำงานที่ลอจิก "0" และต้องต่อตัวต้านทานพูลอัปกับไฟเลี้ยงเข้าที่ขา XIRQ นี้ด้วย ในขณะที่ไม่มีการส่งสัญญาณอินเตอร์รัปต์

7) ขา MODA/LIR และ MODB/Vstby (Mode A/load instruction register และ Mode B/Standby voltage) หลังจากที่มีการรีเซตไมโครคอนโทรลเลอร์ ขานี้ (ทั้ง MODA และ MODB) จะถูกใช้ให้เป็นขาสำหรับเลือกโหมดการทำงานของ 68HC11 ซึ่งเลือกได้เพียง 1 โหมด จาก 4 โหมดที่ได้อธิบายไปแล้ว โดยแสดงการเลือกโหมดตามตารางที่ 2.2

ตารางที่ 2.2 การเลือกโหมดการทำงานของ 68HC11

MODB	MODA	โหมดการทำงาน
1	0	โหมดชิปเดี่ยว
1	1	โหมดมัลติเพล็กซ์ขยาย
0	0	โหมดบูตสแตร์ปพิเศษ
0	1	โหมดทดสอบพิเศษ

หลังจากเลือกโหมดการทำงานแล้ว ขา MODA จะเปลี่ยนลักษณะการทำงานเป็นขา LIR แทนซึ่งจะมีลักษณะเป็นขาเอาต์พุตใช้ในการแสดงสภาวะว่าขณะนี้ได้เริ่มจัดการกับคำสั่งแล้ว โดยจะแอกทีฟเป็นลอจิก “0” ในทันทีที่ขา E เริ่มส่งสัญญาณนาฬิกาพัลส์แรกของแต่ละคำสั่งออกมา ซึ่งก็คือ เมื่อซีพียูเริ่มเฟตช์ข้อมูลคำสั่ง ขา LIR นี้จะแอกทีฟทันที สัญญาณที่ขา LIR จะมีไว้ช่วยในการแก้ไขโปรแกรม (Program debugging) เช่นเดียวกับขา MODA เมื่อขา MODB ถูกใช้ในการเลือกโหมดการทำงานไปแล้ว ที่ขา MODB นี้จะเปลี่ยนหน้าที่เป็นขาอินพุต Vstby แทนโดยอินพุต Vstby นี้เป็นอินพุตสำหรับจ่ายแรงดันเพื่อสแตนด์บายให้แก่แรมภายในชิป ถ้าหากแรงดันที่ขานี้มากกว่าแรงดันไฟเลี้ยงชิปประมาณ 0.7 โวลต์แล้ว หน่วยความจำแรมภายในชิปทั้ง 256 ไบต์ และส่วนลอจิกรีเซต จะรับแรงดันจากขานี้แทนแรงดันที่จ่ายเข้าที่ขา Vdd นั่นคือ ใช้เป็นขาที่จ่ายแรงดันเพื่อเลี้ยงหน่วยความจำแรม เพื่อป้องกันข้อมูลภายในชิปสูญหายเมื่อไม่มีการจ่ายแรงดันไฟเลี้ยงปกติ สำหรับส่วนลอจิกรีเซตจะต้องกำหนดให้มีสภาวะ “0” ก่อนที่จะปลดแรงดันออกจากขา Vdd และต้องรักษาสภาวะลอจิก “0” นี้ไปจนกว่าที่ขา Vdd จะได้รับแรงดันไฟเลี้ยงในระดับปกติ ในรูปที่ 2.12 เป็นวงจรที่ใช้ในการเก็บรักษาข้อมูลในแรมของ 68HC11 โดยการใช้ชิป MAX690 และแบตเตอรี่ผ่านตัวต้านทาน 4.7 กิโลโอห์ม ป้อนเข้าที่ขา MODB/Vstby ของ 68HC11 ก็จะสามารถเก็บรักษาข้อมูลในแรมได้



รูปที่ 2.12 การต่อ MAX690

8) ขา Vrl และ Vrh (A/D Converter Reference Voltages) เป็นขาที่ใช้สำหรับต่อแรงดันอ้างอิงสำหรับวงจรแปลงสัญญาณแอนะลอกเป็นดิจิตอลภายในชิป

9) ขา STRB/R/W (Strobe B และ Read/Write) เป็นขาที่สามารถทำงานได้หลายหน้าที่ ขึ้นอยู่กับโหมดการทำงานของไมโครคอนโทรลเลอร์ ถ้าหาก 68HC11 อยู่ในโหมดชิปเดี่ยวขานี้จะเป็นขาเอาต์พุต STRB หรือเรียกว่าขาสโตรบสำหรับการแฮนด์เชคกับอุปกรณ์ภายนอกที่ต่อเข้าที่พอร์ตขนานอินพุตเอาต์พุต แต่ถ้าอยู่ในโหมดมัลติเพล็กซ์ขยาย ขานี้จะแสดงตนเป็นขา R/W ใช้ในการควบคุมทิศทางของการถ่ายเทข้อมูลกับบัสข้อมูลภายนอกชิป ถ้าขานี้เป็นลอจิก "0" จะหมายความว่าขณะนี้ซีพียูกำลังทำการเขียนข้อมูลลงไปในบัสข้อมูลภายนอก แต่ถ้าเป็นลอจิก "1" จะเป็นการแสดงว่าขณะนี้กำลังอ่านข้อมูลเข้ามา เพื่อทำการประมวลผล ถ้าหากมีการเขียนข้อมูลอย่างต่อเนื่อง ขา R/W ก็จะเป็นลอจิก "0" ตลอดเวลา จนกว่าจะทำการเขียนข้อมูลเสร็จ นอกจากนี้เมื่อใช้งานขา R/W ร่วมกับขา E จะสามารถใช้เป็นสัญญาณอีนาเบิลสำหรับหน่วยความจำสแตติกแรมภายนอก

10) ขา STRA/AS (Strobe A และ Address Strobe) เป็นขาอินพุตโดยรับสัญญาณเป็นแบบขอบขาของสัญญาณ ลักษณะการใช้งานขานี้จะขึ้นอยู่กับการกำหนดโหมดการทำงานของ 68HC11 สำหรับโหมดซิงเกิลชิปขานี้จะเป็นขา STRA คือ เป็นขาสโตรบสำหรับการแฮนด์เชกกับอุปกรณ์ภายนอกที่ต่อเข้ากับพอร์ตขนานอินพุตเอาต์พุต เช่นเดียวกับขา STRB ในโหมดมัลติเพล็กซ์ขยายขานี้จะเปลี่ยนลักษณะการทำงานเป็นขาเอาต์พุต AS โดยใช้ในวงจรดีมัลติเพล็กซ์สัญญาณข้อมูลกับสัญญาณแอดเดรสที่พอร์ต C

11) ขาสัญญาณของพอร์ต พอร์ต A, D และ E จะเป็นพอร์ตที่ไม่ขึ้นกับโหมดการทำงานของ 68HC11 นั่นคือ ไม่ว่า 68HC11 จะทำงานในโหมดใดก็ตาม พอร์ตทั้ง 3 ยังคงมีลักษณะสัญญาณและการทำงานเช่นเดิม ในขณะที่พอร์ต B จะถูกกำหนดให้เป็นพอร์ตเอาต์พุต หากทำงานในโหมดชิปเดี่ยว แต่ถ้าเป็นโหมดมัลติเพล็กซ์ขยายจะกลายเป็นขาตำแหน่งไบต์สูง ส่วนพอร์ต C จะเป็นพอร์ตอินพุตเอาต์พุต ถ้า 68HC11 ทำงานในโหมดชิปเดี่ยว ถ้าหากทำงานในโหมดมัลติเพล็กซ์ขยายพอร์ต C นี้จะใช้เป็นบัสมัลติเพล็กซ์ระหว่างตำแหน่ง 8 บิตต่ำกับบัสข้อมูล

12) พอร์ต A (PA0-PA7) จะเห็นว่า พอร์ต A นี้แต่ละสัญญาณสามารถทำงานได้ตั้งแต่ 2-3 ฟังก์ชัน โดยสามารถเป็นได้ทั้งอินพุตแคปเจอร์ (Input capture) อินพุต คือ IC1, IC2 และ IC3 เป็นเอาต์พุตของฟังก์ชันเปรียบเทียบ (Output compare) 4 ช่อง คือ OC2, OC3, OC4 และ OC5 หรือจะเป็นเอาต์พุตของฟังก์ชันเปรียบเทียบ ลำดับที่ 5 (OC1) อีกลักษณะหนึ่ง คือเป็นอินพุตของพัลส์แอกคิวมูเลเตอร์ ตัวตั้งเวลาโปรแกรมได้ ถ้าหากขาสัญญาณของพอร์ต A ไม่ได้ถูกใช้งานในฟังก์ชันการตั้งเวลา สามารถใช้งานเป็นพอร์ตอินพุตเอาต์พุต เพื่อเชื่อมต่อในงานทั่วๆ ไปได้

13) พอร์ต B (PB0-PB7) เมื่อ 68HC11 ทำงานในโหมดชิปเดี่ยว ขาสัญญาณทั้งหมดจะเป็นขาเอาต์พุตของพอร์ตเอาต์พุต และใช้งานร่วมกับสัญญาณสโตรบเอาต์พุต จะมีพัลส์เอาต์พุตมาปรากฏที่ขา STRB ในทุกๆ ช่วงเวลาที่มีการเขียนข้อมูลออกมาที่พอร์ต B ถ้าหากทำงานในโหมดมัลติเพล็กซ์ขยาย ขาสัญญาณทั้งหมดของพอร์ต B จะทำหน้าที่เป็นขาตำแหน่ง 8 บิตสูง (A8-A15) พอร์ต C (PC0-PC7) ถ้า 68HC11 ทำงานในโหมดชิปเดี่ยว ขาของพอร์ต C ทั้งหมดจะเป็นขาพอร์ตอินพุตเอาต์พุต ถ้าเป็นพอร์ตอินพุต พอร์ต C สามารถที่จะแลตช์สัญญาณอินพุตที่ป้อนเข้ามานี้ได้ โดยการป้อนสัญญาณเข้าที่ขา STRA สัญญาณพอร์ต C มักใช้ในการแฮนด์เชกของการเชื่อมต่อพอร์ตขนาน โดยในขณะเดียวกันขา STRB จะต้องถูกใช้เป็นสายควบคุมการแฮนด์เชก แต่ถ้าทำงานในโหมดมัลติเพล็กซ์ขยาย ขาสัญญาณพอร์ต C จะสามารถเป็นได้ทั้งขาของตำแหน่งและข้อมูลโดยการมัลติเพล็กซ์ ปกติขานี้จะเป็นสัญญาณตำแหน่ง A0-A7 ในแต่ละไซเคิลการทำงานของไมโครคอนโทรลเลอร์แต่ถ้าหากขา E แอกทีฟเป็น “1” สัญญาณที่ขานี้จะกลับเป็นขาสัญญาณข้อมูล D0-D7 โดยทิศทางการเข้าออกของข้อมูลที่พอร์ต C จะแสดงออกมาที่ขาสัญญาณ R/W

14) พอร์ต D (PD)-PD7) ขาสัญญาณพอร์ต D ทั้งขาปกติจะใช้เป็นขาสัญญาณอินพุต เอาต์พุต และสามารถรองรับในการใช้งานเชื่อมต่อสื่อสารข้อมูลอนุกรม (SCI) และเชื่อมต่ออุปกรณ์อนุกรม (SPI) ด้วย ถ้าหากมีการอีนาเบิลในส่วนนี้ให้ทำงาน ขา PD0 เป็นขาอินพุตรับข้อมูล (RxD) เมื่อใช้ในการเชื่อมต่อสื่อสารข้อมูลอนุกรม (SCI) ขา PD1 เป็นเอาต์พุตส่งข้อมูล (TxD) เมื่อใช้ในการเชื่อมต่อสื่อสารข้อมูลอนุกรม (SCI) ขา PD2-PD5 ใช้สำหรับเชื่อมต่ออุปกรณ์อนุกรม (SPI) โดย PD2 เป็นขาสัญญาณ มาสเตอร์อินสเลฟเอาต์ (Master In Slave Out : MISO) PD3 เป็นขาสัญญาณมาสเตอร์เอาต์สเลฟอิน (Master Out Slave In : MOSI) PD4 เป็นขาสัญญาณนาฬิกาอนุกรม (SCK) และขา PD5 เป็นขาอินพุตเลือกสเลฟ (SS)

15) พอร์ต E (PE0-PE7) พอร์ต E เป็นพอร์ตอินพุต และเป็นขาอินพุตสำหรับวงจรแปลงสัญญาณแอนะลอกเป็นดิจิตอล ใน 68HC11 แบบ 48 ขา พอร์ต E จะมีเพียง 4 ขา แต่ถ้าเป็นแบบ 52 ขา จะมีครบ 8 ขา สัญญาณอินพุตที่จะป้อนเข้ามาที่พอร์ต E นี้ ต้องมีความแน่นอนของสัญญาณสูงพอสมควร เพราะความแม่นยำของการแปลงสัญญาณแอนะลอกเป็นดิจิตอล จะขึ้นอยู่กับความเที่ยงตรงของสัญญาณอินพุตเป็นหลัก

2.2.7 รีจิสเตอร์ของซีพียู

ซีพียูของไมโครคอนโทรลเลอร์ 68HC11 จะมีรีจิสเตอร์ใช้งานอยู่ 7 ตัว อันได้แก่

1) แอกคิวมูเลเตอร์ A และ B

2) แอกคิวมูเลเตอร์ D

3) รีจิสเตอร์อินเด็กซ์ IX

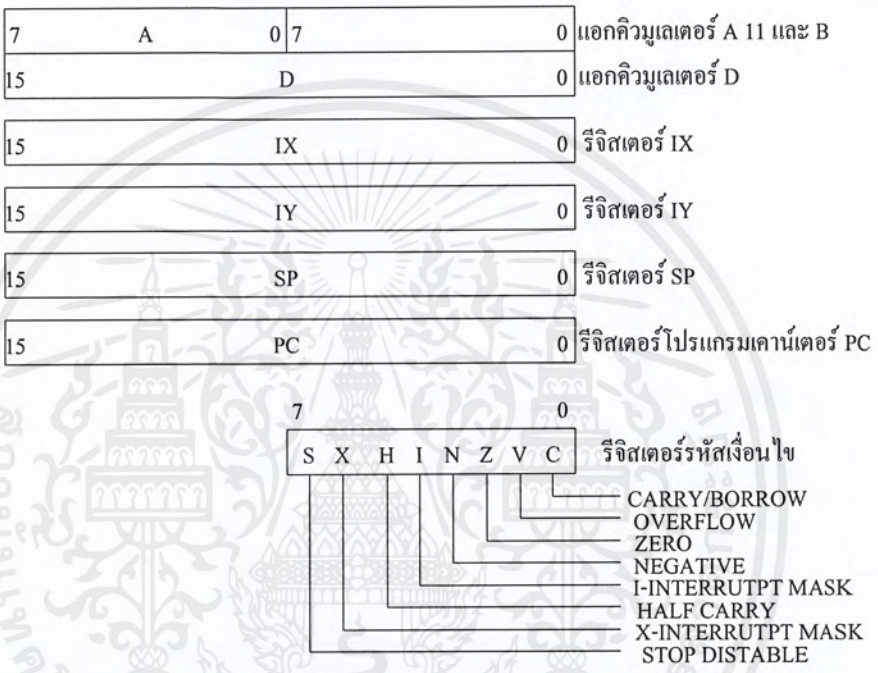
4) รีจิสเตอร์อินเด็กซ์ IY

5) รีจิสเตอร์ตัวชี้สแต็ก (Stack Pointer : SP)

6) รีจิสเตอร์โปรแกรมเคาน์เตอร์ (Program Counter : PC)

7) รีจิสเตอร์รหัสเงื่อนไข (Condition Code Register : CCR)

โดยมีลักษณะของรูปแบบการจัดจำนวนบิตและความหมายดังในรูปที่ 2.13



รูปที่ 2.13 รูปแบบการจัดจำนวนบิตและความหมายของรีจิสเตอร์ของซีพียู

1) แอกคิวมูเลเตอร์ A และ B เป็นรีจิสเตอร์ขนาด 8 บิต ที่ใช้ในการเก็บค่าตัวดำเนินการและผลของการคำนวณทางคณิตศาสตร์หรือผลจากการจัดการข้อมูลโดยตัวซีพียูในการประมวลผลทางคณิตศาสตร์หรือลอจิก จะต้องนำข้อมูลเหล่านั้นมาเก็บในรีจิสเตอร์ทั้งสองตัวนี้ จึงจะสามารถประมวลผลได้

2) แอกคิวมูเลเตอร์ D เป็นรีจิสเตอร์ขนาด 16 บิต ใช้ในการประมวลผลและเก็บค่าจากการคำนวณทางคณิตศาสตร์และลอจิกแอกคิวมูเลเตอร์ D ก็เกิดมาจากการรวมกันของแอกคิวมูเลเตอร์ A และ B จึงทำให้มีขนาด 16 บิต

3) รีจิสเตอร์อินเด็กซ์ IX เป็นรีจิสเตอร์ขนาด 16 บิต ใช้ในการชี้ตำแหน่ง เพื่อเข้าไปจัดการประมวลผลกับข้อมูลในตำแหน่งนั้นๆ นอกจากนั้น IX สามารถใช้เป็นตัวนับหรือรีจิสเตอร์เก็บข้อมูลชั่วคราวได้ด้วย

4) รีจิสเตอร์อินเด็กซ์ IY เป็นรีจิสเตอร์ขนาด 16 บิต มีหน้าที่เหมือนกับ IX แต่จะแตกต่างกันตรงที่ในทุกคำสั่งที่ต้องเกี่ยวข้องกับ IY จะต้องมีข้อมูลไบต์พิเศษของรหัสแมชชีน และมีไซเกิลพิเศษของช่วงเวลาในการเอ็กซีคิวต์คำสั่งด้วย จึงทำให้คำสั่งที่มี IY ไปเกี่ยวข้องต้องมีขนาดเพิ่มขึ้นอย่างน้อย 2 ไบต์ขึ้นไปหรือเรียกว่า พรีไบต์ (Prebyte)

5) รีจิสเตอร์ตัวชี้สแต็ก (SP) เป็นรีจิสเตอร์ขนาด 16 บิต ใช้เก็บตำแหน่งบนสแต็ก (Stack) โดยสแต็กใน 68HC11 นี้จะมีลักษณะการเก็บข้อมูลเข้าและนำข้อมูลออกมาเป็นแบบ LIFO (Last-In-First-Out) หรือข้อมูลที่เข้าไปเก็บในสแต็กหลังสุด เมื่อจะเรียกออกมาจะถูกเรียกออกมาก่อน สแต็กใช้เก็บข้อมูลของรีจิสเตอร์ เมื่อต้องมีการนำรีจิสเตอร์ตัวนั้นไปทำงานในโปรแกรมย่อยอื่นหรือต้องไปใช้ในการตอบสนองการอินเตอร์รัปต์ เพื่อเป็นการป้องกันข้อมูลเดิมสูญหายจึงต้องเก็บข้อมูลนั้นไว้ในหน่วยความจำสำรองแห่งหนึ่ง ซึ่งก็คือสแต็กนั่นเอง ทุกครั้งที่มีการเก็บข้อมูลลงสแต็กค่าของ SP จะลดลง ในทางตรงข้ามถ้าเรียกข้อมูลออกจากสแต็กค่าของ SP ก็จะเพิ่มขึ้น

6) รีจิสเตอร์โปรแกรมเคาน์เตอร์ (PC) เป็นรีจิสเตอร์16บิตใช้เก็บค่าของแอดเดรสของคำสั่งถัดไปที่ซีพียูจะไปทำการเอ็กซีคิวต์

7) รีจิสเตอร์รหัสเงื่อนไข (CCR) เป็นรีจิสเตอร์ขนาด 8 บิต ในแต่ละบิตจะแสดงความหมายของผลจากการกระทำคำสั่งที่เพิ่งจะเอ็กซีคิวต์ไปของซีพียู ดังแสดงในรูปที่ 2.12 แต่ละบิตของ CCR เป็นอิสระต่อกัน จึงสามารถตรวจสอบสถานะได้โดยใช้โปรแกรม และยังสามารถนำผลการตรวจสอบนั้นไปดำเนินการต่อได้ด้วย

รายละเอียดของแต่ละบิตใน CCR มีดังนี้

7.1) บิต Carry/Borrow (C) บิตนี้จะเซตเมื่อซีพียูทำการประมวลผลทางคณิตศาสตร์แล้วเกิดการทดค่า หรือยืมค่า บิตนี้สามารถที่จะใช้งานร่วมกับคำสั่งการหมุน และเลื่อนข้อมูลได้ หรือที่เรียกว่า บิตทด

7.2) บิต Over Flow (V) บิตนี้จะเซตเป็น “1” เมื่อซีพียูกระทำคำสั่งคณิตศาสตร์แล้วเกิดค่าที่เกินออกมานอกเหนือจากเงื่อนไขดังกล่าว บิตนี้จะเป็น “0”

7.3) บิต Zero (Z) บิตนี้จะเซตเมื่อผลของการกระทำทางคณิตศาสตร์ หรือลอจิกหรือการประมวลผลข้อมูลแล้วทำให้เกิดเป็นค่าศูนย์ขึ้นมา

7.4) บิต Negative (N) ถ้าหากผลของการกระทำทางคณิตศาสตร์หรือลอจิกหรือการประมวลผลข้อมูลแล้ว ทำให้เกิดค่าเป็นลบบิตนี้จะเซต ผลลัพธ์ที่เป็นลบสามารถสังเกตได้จากบิตที่มีนัยสำคัญสูงสุด (MSB) มีค่าเป็น “1”

7.5) บิต I-Interrupt Mask (I) สามารถเซตบิตนี้ให้เป็น “1” ได้ 2 วิธีคือ โดยวิธีการทาง

ฮาร์ดแวร์ และโดยการใช้คำสั่งที่ใช้ในการดิสเอเบิลการอินเตอร์รัปต์แบบมาสเคเบิลทั้งภายในและภายนอก

7.6) บิต Half Carry (H) จะเซตเป็น "1" เมื่อซีพียูกระทำคำสั่งทางคณิตศาสตร์ เช่น ADD, ABA หรือ ADC แล้วเกิดการทดข้ามจากบิตที่ 3 มายังบิตที่4 บิต X-Interrupt Mask (X) บิตนี้จะถูกเซตด้วยวิธีการทางฮาร์ดแวร์เท่านั้น โดยการป้อนสัญญาณเข้าที่ขา RESET และ XIRQ และจะรีเซตบิตนี้ได้ด้วยการใช้คำสั่ง TAP หรือ RTI เท่านั้น

7.7) บิต Stop Disable (S) บิตนี้จะเซตเมื่อต้องการดิสเอเบิลคำสั่ง STOP และถ้ารีเซตบิตนี้ ก็จะเป็นการอีนาเบิลคำสั่ง STOP บิตนี้ถูกควบคุมโดยโปรแกรม และเมื่อบิตนี้ถูกเซตจะทำให้คำสั่ง STOP มีผลเช่นเดียวกับคำสั่ง NOP

2.2.8 ระบบฮาร์ดแวร์ของไมโครคอนโทรลเลอร์ 68HC11

ในหัวข้อนี้ จะอธิบายถึงระบบของฮาร์ดแวร์ของไมโครคอนโทรลเลอร์ 68HC11 โดยจะเริ่มต้นจาก หน่วยความจำภายในชิป รีจิสเตอร์ควบคุมภายใน 68HC11 การจัดการเกี่ยวกับการบันทึกและลบข้อมูลในหน่วยความจำอีอีพรอม การทำงานของระบบสัญญาณนาฬิกาใน 68HC11 และเรื่องสุดท้ายเป็นเรื่องการเชื่อมต่อกับหน่วยความจำภายนอก

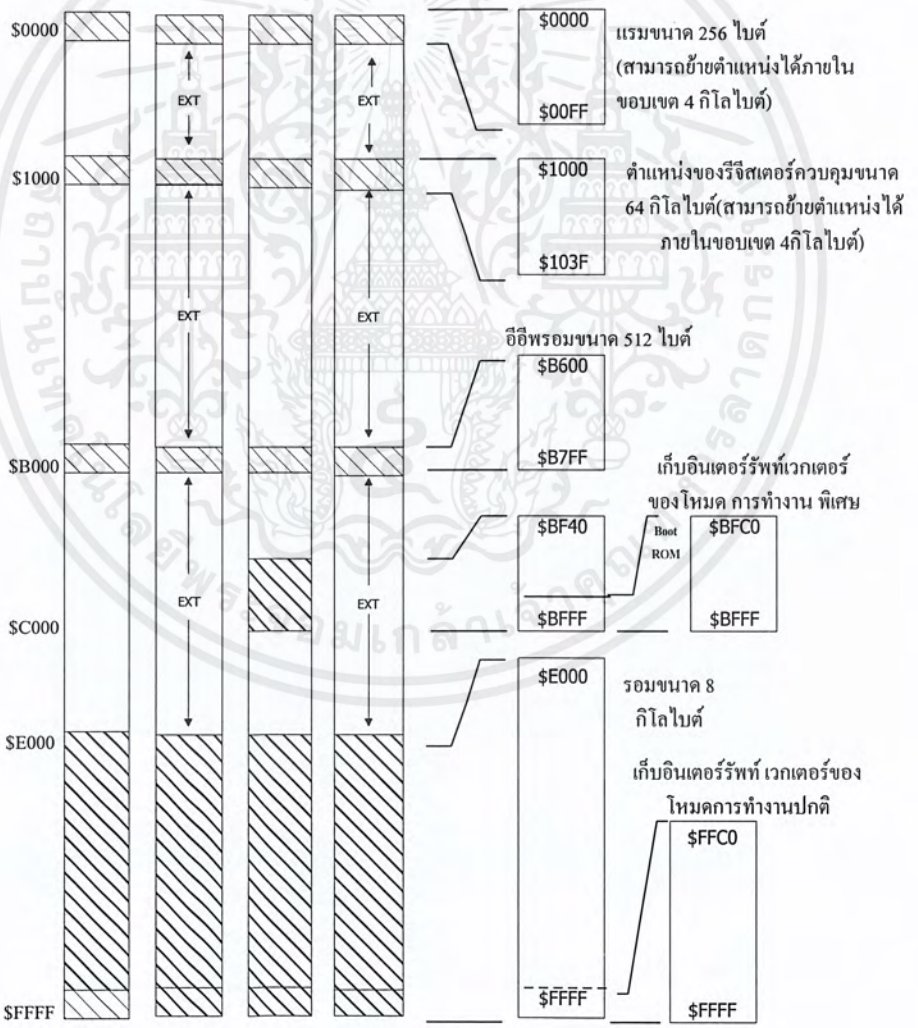
1) หน่วยความจำภายในชิป

จะมีหน่วยความจำภายในโดยจะมีเป็นชนิดใดขนาดเท่าใดขึ้นอยู่กับรุ่นหรือเวอร์ชันของ 68HC11 แต่ที่จะนำมาอธิบายนี้เป็นเบอร์ MC68HC11A8 ซึ่งในเบอร์นี้จะมีหน่วยความจำภายในค่อนข้างสมบูรณ์ กล่าวคือมีหน่วยความจำทุกแบบ และมีขนาดแตกต่างกันไปคือ ใน MC68HC11A8 มีรอมขนาด 8 กิโลไบต์ แรมขนาด 256 ไบต์ และอีอีพรอมขนาด 512 ไบต์ โดยมีการจัดหน่วยความจำภายในชิป ดังรูปที่ 2.14 เป็นการจัดสรรหน่วยความจำ หรือที่เรียกว่า แผนผังหน่วยความจำ (Memory maps) จะขึ้นอยู่กับว่า MC68HC11A8 นี้ทำงานอยู่ในโหมดใด

จากรูปที่ 2.14 ส่วนที่แรเงาจะถูกแสดงรายละเอียดเอาไว้ในรูปด้านขวามือเรียบร้อยแล้ว เพื่อบอกให้กับผู้ใช้งานทราบว่า พื้นที่ในส่วนใดถูกจัดสรรไว้เพื่อรองรับงานอย่างใดบ้าง แรมขนาด 256 ไบต์ ถูกจัดสรรไว้ในตำแหน่ง $0000-$00FF ที่ตำแหน่ง $1000-$103F เป็นพื้นที่สำหรับรีจิสเตอร์ควบคุมต่างๆ ขนาด 64 ไบต์ ทั้งแรมและรีจิสเตอร์สามารถที่จะมีการจัดสรรตำแหน่งกันใหม่ ถ้าหากมีการใช้งานในลักษณะ 4 เพจ ซึ่งสามารถกำหนดได้โดยรีจิสเตอร์ INIT อันจะได้กล่าวถึงต่อไป ที่ตำแหน่ง $B600-$B7FF ขนาด 512 ไบต์เป็นพื้นที่ของหน่วยความจำอีอีพรอม ที่ตำแหน่ง $BP40-$BFFF ถ้า 68HC11 ทำงานในโหมดบูตสแตร์ปพิเศษ พื้นที่นี้จะสงวนไว้เป็นรอมที่เก็บข้อมูลสำหรับเริ่มต้นการทำงานของระบบ (Boot ROM) ถ้าหาก 68HC11 ทำงานในโหมดทดสอบพิเศษที่ตำแหน่ง $BFC0-$BFFF ซึ่งมีขนาดของหน่วยความจำเป็น 192 ไบต์ จะถูกจัดสรร

ไว้สำหรับเก็บอินเตอร์รัปต์เวกเตอร์ของการทำงานโหมดบูตสแตร็ปพิเศษ (Special modes interrupt vectors) สำหรับตำแหน่ง $E000-$FFFF จะถูกจัดสรรไว้สำหรับเป็นพื้นที่ของรอมขนาด 8 กิโลไบต์ แต่ถ้าหาก 68HC11 ทำงานในโหมดชิปเดี่ยวและมัลติเพล็กซ์ขยาย ที่ตำแหน่ง $FFC0-$FFFFจะถูกจัดสรรไว้เพื่อเก็บค่าของอินเตอร์รัปต์เวกเตอร์ปกติ (Normal interrupt vectors)

ส่วนบริเวณที่เขียนว่า EXT เป็นพื้นที่ที่จัดสรรไว้สำหรับต่อหน่วยความจำภายนอกเพิ่มขึ้น จะเห็นว่ามีด้วยกันเพียง 2 โหมดเท่านั้นที่สามารถจะต่อกับหน่วยความจำภายนอกได้ ซึ่งก็คือโหมดมัลติเพล็กซ์ขยายและโหมดการทดสอบพิเศษ และทั้ง 2 โหมดนั้นก็ได้กำหนดตำแหน่งสำหรับหน่วยความจำไว้ 3 ช่วงคือ $0100-$0FFF มีขนาด 3,840 ไบต์, $1040-$B5FF ขนาด 42,432 ไบต์ (41.4375 กิโลไบต์) และ $B800-$DFFF ขนาด 10 กิโลไบต์



รูปที่ 2.14 การจัดสรรหน่วยความจำของ 68HC11 แบ่งตามโหมดการทำงาน

ส่วนบริเวณที่เขียนว่า EXT เป็นพื้นที่ที่จัดสรรไว้สำหรับต่อหน่วยความจำภายนอกเพิ่มขึ้นจะเห็นว่ามีด้วยกันเพียง 2 โหมดเท่านั้นที่สามารถจะต่อกับหน่วยความจำภายนอกได้ ซึ่งก็คือโหมดมัลติ-เพล็กซ์ขยายและโหมดการทดสอบพิเศษ และทั้ง 2 โหมดนั้นก็ได้กำหนดตำแหน่งสำหรับหน่วยความจำไว้ 3 ช่วงคือ $0100-$0FFF มีขนาด 3,840 ไบต์, $1040-$B5FF ขนาด 42,432 ไบต์ (41.4375 กิโลไบต์) และ $B800-$DFFF ขนาด 10 กิโลไบต์

2.2.9 การอ้างตำแหน่งและชุดคำสั่งของ 68HC11

1) การอ้างตำแหน่งของ 68HC11

68HC11 มีกระบวนการอ้างตำแหน่ง (Addressing) ทั้งสิ้น 6 โหมดด้วยกัน ประกอบด้วย การอ้างตำแหน่งแบบทันทีทันใด (Immediat), แบบโดยตรง (Direct), แบบขยาย (Extended), แบบอินเด็กซ์ (Index), แบบอินเฮียแรนท์ (Inherent) และแบบสัมพัทธ์ (Relative) ในบางคำสั่งเมื่อถูกใช้ในการอ้างตำแหน่งบางแบบ จำเป็นต้องเพิ่มข้อมูลเข้าไปอีก 1 ไบต์ก่อนรหัสคำสั่ง เพื่อปรับให้ 68HC11 สามารถทำงานในลักษณะมัลติเพจออปโค้ดแมป (Multi-page opcode map) ได้เหมาะสมขึ้น ข้อมูลไบต์นั้นจะเรียกว่า พรีไบต์ (Prebyte)

1.1) การอ้างตำแหน่งแบบทันทีทันใด

ในการอ้างตำแหน่งแบบนี้ เป็นการติดต่อเพื่อจัดการข้อมูล ซึ่งเป็นค่าใดๆ โดยตรงในทันทีทันใด จำนวนไบต์ของคำสั่งในการอ้างตำแหน่งแบบนี้จะมีขนาดตั้งแต่ 2-4 ไบต์ ขึ้นอยู่กับขนาดของรีจิสเตอร์ที่เกี่ยวข้องด้วย เช่น ถ้าเป็นแอกคิวมูเลเตอร์ A ก็จะมีจำนวนไบต์ของคำสั่ง 2 ไบต์ (ไบต์แรกเป็นรหัสคำสั่ง อีก 1 ไบต์หลังเป็นขนาดของตัวดำเนินการ ซึ่งเท่ากับขนาดของรีจิสเตอร์แอกคิวมูเลเตอร์ A) รูปแบบของคำสั่งที่มีการอ้างตำแหน่งแบบนี้ หลังจากคำสั่งแล้วต้องตามด้วยเครื่องหมาย # เสมอเพื่อเป็นการบ่งให้ซีพียูทราบว่า คำสั่งที่จะกระทำต่อไปนี้อ้างตำแหน่งแบบทันทีทันใด

1.2) การอ้างตำแหน่งแบบโดยตรง

เป็นการติดต่อเพื่อประมวลผลข้อมูลขนาด 8 บิต ที่อยู่ในหน่วยความจำแรมภายในชิป ซึ่งมีอยู่ 256 ไบต์ โดยมีตำแหน่งตั้งแต่ $0000-$00FF ดังนั้นค่าตัวดำเนินการที่ตามหลังรหัสคำสั่งคำสั่งก็คือ ค่าตำแหน่งของแรมนั่นเอง

ในการอ้างตำแหน่งแบบนี้บางทีเรียกว่า การอ้างตำแหน่งเพจศูนย์ (Zero page addressing)

1.3) การอ้างตำแหน่งแบบขยาย

ในการอ้างตำแหน่งแบบนี้ ข้อมูลในไบต์ที่ 2 และ 3 ตามหลังรหัสคำสั่ง จะเก็บค่าตำแหน่งจริงของหน่วยความจำที่ต้องการนำข้อมูลในหน่วยความจำออกมาประมวลผลหรือจัด

เก็บข้อมูลลงไปใหม่ เมื่อใช้การอ้างตำแหน่งแบบนี้คำสั่งจะมีขนาด 3-4 ไบต์ โดยเป็นรหัสคำสั่งไบต์ที่ 1 หรือ 2 (กรณี ถ้ามีขนาด 4 ไบต์) ส่วน 2 ไบต์หลัง จะเป็นค่าตำแหน่งที่อ้างถึงในหน่วยความจำ

1.4) การอ้างตำแหน่งแบบอินเด็กซ์

การอ้างตำแหน่งแบบนี้จะมีการนำรีจิสเตอร์ชี้ข้อมูล (Index register) ทั้ง 2 ตัว คือ IX และ IY มาใช้ในการคำนวณค่าตำแหน่งที่ต้องการเรียกใช้ข้อมูล ดังนั้นค่าตำแหน่งที่ต้องการใช้งานจะเปลี่ยนแปลง หรือขึ้นอยู่กับองค์ประกอบหลัก 2 ประการ คือ

1. ค่าตำแหน่งที่ถูกกำหนดอยู่ในปัจจุบัน
2. ค่าออฟเซ็ต ที่บรรจุอยู่ในคำสั่ง (หรือค่าโอเปอร์แรนด์)

ในการอ้างตำแหน่งแบบนี้ สามารถที่จะใช้หน่วยความจำที่ตำแหน่งใดก็ได้ในจำนวน 64 กิโลไบต์เป็นจุดอ้างอิง

ส่วนในด้านขนาดของคำสั่งในการอ้างตำแหน่งแบบนี้ ถ้าใช้รีจิสเตอร์ IXจะมีขนาด 2 ไบต์ โดยไบต์แรกเป็นออปโค้ด ส่วนไบต์ที่สองเป็นค่าออฟเซต หากใช้รีจิสเตอร์ IY จะมีขนาด 3 ไบต์ ไบต์แรกเป็นพรีไบต์ ตามด้วยรหัสคำสั่งและค่าออฟเซตขนาด 8 บิต

1.5) การอ้างตำแหน่งแบบอินเฮียแรนท์

การอ้างตำแหน่งแบบนี้ จะไม่ยุ่งเกี่ยวกับข้อมูลในหน่วยความจำแต่อย่างใด แต่จะเข้าไปจัดการในรีจิสเตอร์แทน ดังนั้นขนาดของคำสั่งในการอ้างตำแหน่งแบบนี้จะมีเพียง 1-2 ไบต์ โดยเป็นรหัสคำสั่งทั้งสิ้นไม่มีตัวดำเนินการ

1.6) การอ้างตำแหน่งแบบสัมพัทธ์

การอ้างตำแหน่งแบบนี้จะใช้ในชุดคำสั่งกระโดด (Jump and branch instructions) ถ้าหากเงื่อนไขในการกระโดดเป็นจริง ค่าออฟเซตขนาด 8 บิต ที่อยู่ตามหลังรหัสคำสั่ง ก็จะถูกบวกเข้าไปในริจิสเตอร์โปรแกรมเคาน์เตอร์ (PC) เพื่อกำหนดตำแหน่งต่อไปที่จะข้ามไปทำงานของซีพียู ปกติแล้วขนาดของคำสั่งในการอ้างตำแหน่งแบบนี้จะเท่ากับ 2 ไบต์โดยไบต์แรกเป็นรหัสคำสั่ง ส่วนไบต์ที่ 2 เป็นค่าออฟเซตเพื่อบอกจำนวนที่จะเพิ่มเข้าไปใน PC

1.7) พรีไบต์

เป็นข้อมูลขนาด 8 บิต (1 ไบต์) ที่ใส่เข้าไปก่อนหน้ารหัสคำสั่ง เพื่อประโยชน์ในการบอกให้ซีพียูทราบถึงลักษณะของการจัดเพจของรหัสคำสั่งโดยถ้าหาก 68HC11 ทำงานในเพจ 1 ข้อมูลในพรีไบต์ก็ไม่ต้องมี แต่ถ้าทำงานในเพจ 2 จะต้องเพิ่มค่าพรีไบต์เท่ากับ $18 เข้าไปก่อนรหัสคำสั่งเสมอ แล้วตามด้วยรหัสคำสั่ง ในกรณีเพจ 3 จะใช้ค่า $1A และใช้ค่า $CD สำหรับเพจ 4

ค่าพรีไบต์จะเข้าไปเกี่ยวข้อง เมื่อมีคำสั่งใดๆก็ตามกำหนดให้รีจิสเตอร์ IY ทำงาน

2) ชุดคำสั่งของ 68HC11

ซีพียูภายในไมโครคอนโทรลเลอร์ 68HC11 มีลักษณะการทำงานคล้ายๆ กับซีพียู MC6801 แต่ได้มีการเพิ่มเติมความสามารถของ 68HC11 นี้ด้วยการเพิ่มรีจิสเตอร์และคำสั่งให้มากขึ้น เช่น เพิ่มรีจิสเตอร์ขนาด 16 บิต อีก 1 ตัวคือ IY เพิ่มคำสั่งเกี่ยวกับการหารเลข 16 บิตอีก 2 แบบ เพิ่มคำสั่งควบคุมและเพิ่มคำสั่งจัดการข้อมูลระดับบิต (Bit manipulation instruction)

ในที่นี้จะแบ่งชุดคำสั่งของ 68HC11 ออกเป็น 7 กลุ่มย่อยๆ ดังนี้

1) กลุ่มคำสั่งจัดการเกี่ยวกับข้อมูล (Data handling instruction)

2) กลุ่มคำสั่งทางคณิตศาสตร์ (Arithmatics instruction)

3) กลุ่มคำสั่งทางลอจิก (Logic instruction)

4) กลุ่มคำสั่งการกระโดด (Jump and branch instruction)

5) กลุ่มคำสั่งเปรียบเทียบและตรวจสอบข้อมูล (Data test instruction)

6) กลุ่มคำสั่งจัดการกับรีจิสเตอร์รหัสเงื่อนไข (CCR instruction)

7) กลุ่มคำสั่งควบคุม (Control instruction)

2.1) กลุ่มคำสั่งจัดการเกี่ยวกับข้อมูล (Data handling instruction)

ในชุดคำสั่งกลุ่มนี้ยังแบ่งแยกย่อยได้อีก 3 กลุ่มคือ กลุ่มเคลื่อนย้ายข้อมูล กลุ่มเปลี่ยนแปลงแก้ไขข้อมูล เลื่อนและหมุนข้อมูล

2.1.1) กลุ่มเคลื่อนย้ายข้อมูล

กลุ่มคำสั่งชุดนี้แบ่งออกเป็น กลุ่มเคลื่อนย้ายข้อมูลระดับบิต ระดับไบต์ และเวิร์ด โดยมีรายละเอียดของคำสั่งในแต่ละกลุ่มดังนี้

1) กลุ่มเคลื่อนย้ายข้อมูลระดับบิต

มีด้วยกัน 2 คำสั่ง คือ BSET (Set Bit) และ BCLR (Clear Bit)

BSET เป็นคำสั่งที่ใช้ในการทำให้บิตใดๆ ของข้อมูลที่อ้างเป็น "1" แล้วนำข้อมูลที่ทำการเซตบิตแล้วกลับเข้าไปเก็บในตำแหน่งเดิม รูปแบบการทำงานเป็นดังนี้ $M + mm \rightarrow M$ (M คือ ค่าของข้อมูลที่อ้างถึงในหน่วยความจำ) mm คือ บิตใดๆ ที่ต้องการเซต ส่วนเครื่องหมาย + หมายถึง การออร์ (OR)

BCLR มีลักษณะตรงข้ามกับ BSET คือ ทำให้บิตใดๆ ของข้อมูลที่อ้างถึงเป็น "0" มีรูปแบบการทำงานดังนี้ $\overline{M}.mm \rightarrow M$ (. คือการแอนด์ (AND))

2) กลุ่มคำสั่งเคลื่อนย้ายข้อมูลระดับไบต์

ในกลุ่มนี้มีด้วยกัน 15 คำสั่ง สามารถอธิบายรายละเอียดได้ดังนี้

LDAA (Load Accumulator A) เป็นคำสั่งที่ใช้โหลดข้อมูล ที่อ้างถึงมาเก็บไว้ในแอกคิวมูเลเตอร์ A

LDAB (Load Accumulator B) เป็นคำสั่งที่ใช้โหลดข้อมูล ที่อ้างถึงมาเก็บไว้ในแอกคิวมูเลเตอร์ B

STAA (Store Accumulator A) เป็นคำสั่งที่ใช้ในการอ่านค่าจากแอกคิวมูเลเตอร์ A มาเก็บไว้ในหน่วยความจำ จะมีลักษณะการทำงานตรงข้ามกับ LDAA

STAB (Store Accumulator B) เหมือนกับ STAA แต่จะอ่านค่าจากแอก-คิวมูเลเตอร์ B แทน

TAB (Transfer A to B) เป็นคำสั่งในการถ่ายเทข้อมูลจากแอกคิวมูเลเตอร์ A ไป B

TBA (Transfer B to A) เป็นคำสั่งในการถ่ายเทข้อมูลจากแอกคิวมูเลเตอร์ B ไป A

TAP (Transfer A to CCR) เป็นคำสั่งในการถ่ายเทข้อมูลจากแอกคิวมูเลเตอร์ A ไปยังรีจิสเตอร์รหัสเงื่อนไข (CCR)

TPA (Transfer CCR to A) เป็นคำสั่งในการถ่ายเทข้อมูล จากรีจิสเตอร์รหัสเงื่อนไข (CCR) มายังแอกคิวมูเลเตอร์ A

CLRA และ CLRB (Clear Accumulator A & Clear Accumulator B) เป็นคำสั่งที่ใช้ในการเคลียร์ข้อมูลในแอกคิวมูเลเตอร์ A และ B

CLR (Clear memory byte) เป็นคำสั่งในการเคลียร์ข้อมูล ในหน่วยความจำ 1 ไบต์ นั่นคือนำข้อมูล “0” เขียนลงในหน่วยความจำนั้นเอง

PSHA และ PSHB (Push A onto Stack & Push B onto Stack) เป็นคำสั่งที่ใช้เก็บค่าข้อมูลในแอกคิวมูเลเตอร์ A หรือ B ไปไว้ในสแต็ก เมื่อทำคำสั่งนี้แล้วค่าของรีจิสเตอร์ชี้สแต็ก (SP) จะลดลง

PULA และ PULB (Pull A from Stack & Pull B from Stack) เป็นคำสั่งที่ใช้เรียกข้อมูลของแอกคิวมูเลเตอร์ A หรือ B ที่เก็บไว้ในสแต็ก กลับมาเก็บไว้ในแอกคิวมูเลเตอร์ A หรือ B อย่างเดิม ค่าของ SP จะเพิ่มขึ้นเมื่อทำคำสั่งนี้แล้ว

3) กลุ่มคำสั่งเคลื่อนย้ายข้อมูลระดับเวิร์ด

LDD (Load Double Accumulator D) เป็นการโหลดข้อมูล 16 บิต มาเก็บไว้ในแอกคิวมูเลเตอร์ D โดยไบต์ต่ำจะเก็บไว้ในแอกคิวมูเลเตอร์ A ส่วนไบต์สูงจะเก็บไว้ในแอกคิวมูเลเตอร์ B (แอกคิวมูเลเตอร์ D เกิดจากการรวมกันของ A และ B)

STD (Store Accumulator D) เป็นคำสั่งในการเก็บข้อมูลจาก D ไปไว้ในหน่วยความจำ โดยมีการแยกเก็บคือ ข้อมูลใน A จะถูกเก็บไว้ในหน่วยความจำแอดเดรสต่ำ ส่วนข้อมูลใน B จะเก็บไว้ในหน่วยความจำแอดเดรสถัดไป

ตารางที่ 2.3 คำสั่งในกลุ่มเคลื่อนย้ายข้อมูลระดับบิตและไบต์

คำสั่ง	ลักษณะการทำงาน	แฟลกของรีจิสเตอร์รหัสเงื่อนไข							
		S	X	H	I	N	Z	V	C
BSET	M ← M OR mm	-	-	-	-	T	T	0	-
BCLR	M ← M AND ($\overline{mm}$)	-	-	-	-	T	T	0	-
LDAA (LDAB)	A ← M	-	-	-	-	T	T	0	-
STAA (STAB)	M ← M	-	-	-	-	T	T	0	-
TAB	B ← A	-	-	-	-	T	T	0	-
TBA	A ← B	-	-	-	-	T	T	0	-
TAP	CC ← A	T	T	T	T	T	T	T	T
TPA	A ← CC	-	-	-	-	-	-	-	-
CLAA (CLAB)	A ← 0	-	-	-	-	0	1	0	0
CLR	M ← 0	-	-	-	-	0	1	0	0
PSHA (PSHB)	Stack ← M	-	-	-	-	-	-	-	-
PULA (PULB)	M ← Stack	-	-	-	-	-	-	-	-

XGDX และ XGDY (Exchange D with X & Exchange D with Y) เป็นคำสั่งในการเปลี่ยนข้อมูลระหว่างรีจิสเตอร์ข้อมูล IX และ IY กับแอกคิวมูเลเตอร์ D โดย XGDX เป็นการเปลี่ยนระหว่าง IX กับ D และ XGDY เป็นการแลกเปลี่ยนระหว่าง IY กับ D

LDX และ LDY (Load Index Register X & Load Index Register Y) เป็นคำสั่งใช้ในการโหลดข้อมูลขนาด 16 บิต จากหน่วยความจำ 2 แอดเดรส (แอดเดรสละ 8 บิต) มาเก็บไว้ใน IX และ IY

STX และ STY (Store Index Register X & Store Index Register Y) เป็นคำสั่งในการเก็บข้อมูลจาก IX และ IY ลงในหน่วยความจำ โดยไบต์ต่ำเก็บในแอดเดรสต่ำ ส่วนไบต์สูงเก็บในแอดเดรสถัดไป

LDS (Load Stack Pointer) เป็นคำสั่งใช้ในการกำหนดค่าแอดเดรสของสแต็กให้แก่รีจิสเตอร์ตัวชี้สแต็ก (SP)

PSHX และ PSHY (Push X onto Stack & Push Y onto Stack) เป็นการเก็บค่า IX และ IY ลงในสแต็ก โดยเก็บข้อมูลไบต์ต่ำของ IX และ IY เข้าไปก่อนหลังจากทำงานแล้วค่าของ SP จะลดลง 2 ค่า (SP = SP-2)

PULX และ PULY (Pull X from Stack & Pull Y form Stack) เป็นการดึงค่า IX และ IY กลับคืนมาจากสแต็ก โดยเก็บข้อมูลไบต์สูงของ IX และ IY จะออกมาก่อนหลังจากทำงานแล้วค่าของ SP จะเพิ่มขึ้น 2 ค่า (SP = SP+2)

ตารางที่ 2.4 คำสั่งกลุ่มเคลื่อนย้ายข้อมูลระดับเวิร์ด

คำสั่ง	ลักษณะการทำงาน	แฟลกของรีจิสเตอร์เงื่อนไข							
		S	X	H	I	N	Z	V	C
LDD	A ← M, B ← M+1	-	-	-	-	T	T	0	-
STD	M ← A, M+1 ← B	-	-	-	-	T	T	0	-
XGDX	D ↔ X	-	-	-	-	-	-	-	-
XGDY	D ↔ Y	-	-	-	-	-	-	-	-
LDX	X ← M:M+1	-	-	-	-	T	T	0	-
STX	M:M+1 ← X	-	-	-	-	T	T	0	-
LDY	Y ← M:M+1	-	-	-	-	T	T	0	-
STY	M:M+1 ← Y	-	-	-	-	T	T	0	-
LDS	S ← M:M+1	-	-	-	-	T	T	0	-
STS	M:M+1 ← S	-	-	-	-	T	T	0	-
PSHX	Stack ← X	-	-	-	-	-	-	-	-
PULX	X ← Stack	-	-	-	-	-	-	-	-
PSHY	Stack ← Y	-	-	-	-	-	-	-	-

ตารางที่ 2.4 (ต่อ) คำสั่งกลุ่มเคลื่อนย้ายข้อมูลระดับเวิร์ด

คำสั่ง	ลักษณะการทำงาน	แฟลกของรีจิสเตอร์เงื่อนไข							
		S	X	H	I	N	Z	V	C
PULY	Y ← Stack	-	-	-	-	-	-	-	-
TSX	X ← S+1	-	-	-	-	-	-	-	-
TXS	S ← X-1	-	-	-	-	-	-	-	-
TSY	Y ← S+1	-	-	-	-	-	-	-	-
TYS	X ← Y-1	-	-	-	-	-	-	-	-

TSX และ TSY (Transfer Stack pointer to X & Transfer Stack pointer to Y) เป็นการนำค่าของตัวชี้สแต็กที่เพิ่มขึ้น 1 ค่า (SP+1) มาเก็บไว้ใน IX หรือ IY

TXS และ TYS (Transfer X to Stack pointer & Transfer Y to Stack pointer) เป็นการนำค่าของ IX หรือ IY ที่ลดลง 1 ค่าแอดเดรส (IX-1 หรือ IY-1) ไปเก็บไว้ใน SP

2.1.2) กลุ่มคำสั่งเปลี่ยนแปลงและแก้ไขข้อมูล

แบ่งออกเป็น 3 ส่วนคือ กลุ่มคำสั่งเพิ่มค่า ลดค่า และแปลงค่า มีรายละเอียดดังนี้

1) กลุ่มคำสั่งเพิ่มค่า (Increment)

มีทั้งสิ้น 6 คำสั่ง ดังนี้

INCA และ INCB (Increment Accumulator A & Increment Accumulator B) เป็นคำสั่งที่ใช้เพิ่มค่าแอกคิวมูเลเตอร์ A หรือ B ครั้งละ 1 ค่า

INC (Increment Memory byte) เป็นคำสั่งที่ใช้เพิ่มค่าของข้อมูลที่อ้างถึง 1 ค่า

INCX และ INCY (Increment index Register X & Increnment Indexregister Y) เป็นคำสั่งใช้เพิ่มค่า IX หรือ IY ครั้งละ 1 ค่า

INS (Increment stack pointer) เป็นคำสั่งใช้เพิ่มค่ารีจีสเตอร์ตัวชี้แสต็ก SP ขึ้นครั้งละ 1 ค่า

2) กลุ่มคำสั่งลดค่า (Decrement)

DECA และ DECB (Decrement Accumulator A & Decrement Accumulator B) เป็นคำสั่งที่ใช้ลดค่าแอกคิวมูเลเตอร์ A หรือ B

DEC (Decrement Memory byte) เป็นคำสั่งที่ใช้ลดค่าของข้อมูลที่อ้างถึง

DECX และ DECY (Decrement index Register X & Decrement Indexregister Y) เป็นคำสั่งใช้ลดค่า IX หรือ IY ครั้งละ 1 ค่า

DES (Decrement stack pointer) เป็นคำสั่งใช้ลดค่ารีจีสเตอร์ตัวชี้แสต็ก SP ขึ้นครั้งละ 1 ค่า

3) กลุ่มคำสั่งแปลงค่า (Complement)

มีด้วยกัน 3 คำสั่งคือ

COM A COM B (1's Complement A & 1's Complement B) เป็นคำสั่งที่ใช้ในการแปลงค่าข้อมูลในแอกคิวมูเลเตอร์ A หรือ B เป็น 1's คอมพลีเมนท์ โดยนำค่า $FF ลบด้วยค่าในแอกคิวมูเลเตอร์ แล้วนำมาเก็บในแอกคิวมูเลเตอร์

ตารางที่ 2.5 คำสั่งกลุ่มเปลี่ยนแปลงและแก้ไขข้อมูล

คำสั่ง	ลักษณะการทำงาน	แฟลกของรีจิสเตอร์เงื่อนไข							
		S	X	H	I	N	Z	V	C
INCA (INCB)	A ← A+1	-	-	-	-	T	T	T	-
INC	M ← M+1	-	-	-	-	T	T	T	-
INX	X ← X+1	-	-	-	-	-	T	-	-
INY	Y ← Y+1	-	-	-	-	-	T	-	-
INS	S ← S+1	-	-	-	-	-	-	-	-
DECA (DECB)	A ← A-1	-	-	-	-	T	T	T	-
DEC	M ← M-1	-	-	-	-	T	T	T	-
DEX	X ← X-1	-	-	-	-	-	T	-	-
DEY	Y ← Y-1	-	-	-	-	-	T	-	-
DES	S ← S-1	-	-	-	-	-	-	-	-
COMA (COMB)	A ← FFH-A	-	-	-	-	T	T	0	1
COM	M ← FFH-M	-	-	-	-	T	T	0	1

COM (1's Complement memory byte) เช่นเดียวกับ COM A และ COM B หากแต่จะเปลี่ยนจากข้อมูลที่อยู่ในหน่วยความจำ แล้วเก็บกลับเข้าไปในตำแหน่งเดิม รายละเอียดของกระบวนการต่างๆ ของการแปลงข้อมูลทำเช่นเดียวกับคำสั่ง COM A และ COM B

2.1.3) กลุ่มคำสั่งเลื่อนและหมุนข้อมูล

1) กลุ่มคำสั่งหมุนข้อมูล

มีด้วยกัน 6 คำสั่ง มีรายละเอียดดังนี้

ROLA (Rotate Left A)

ROLB (Rotate Left B)

ROL (Rotate Left)

ทั้ง 3 คำสั่ง เป็นคำสั่งให้หมุนข้อมูลไปทางซ้าย แต่จะต่างกันที่ว่า คำสั่ง ROLA เป็นการหมุนข้อมูลในแอกคิวมูเลเตอร์ A และคำสั่ง ROLB เป็นการหมุนข้อมูลใน B หรือ ROL เป็นคำสั่งที่ใช้ในการหมุนข้อมูลในหน่วยความจำ

RORA (Rotate Right A)

RORB (Rotate Right B)

ROR (Rotate Right)

เป็นคำสั่งให้หมุนข้อมูลไปทางขวา แต่จะต่างกันที่ว่า คำสั่ง RORA เป็นการหมุนข้อมูลในแอกคิวมูเลเตอร์ A และคำสั่ง RORB เป็นการหมุนข้อมูลใน หรือ ROR เป็นคำสั่งที่ใช้ในการหมุนข้อมูลในหน่วยความจำ

2) กลุ่มคำสั่งเลื่อนข้อมูลคณิตศาสตร์

เหตุผลที่เรียกกลุ่มคำสั่งต่อไปนี้ว่า กลุ่มคำสั่งเลื่อนข้อมูลคณิตศาสตร์ คือ ในทุกๆ ครั้งของการเลื่อนข้อมูลจะเกิดการคูณหรือหาร 2 เสมอ กลุ่มคำสั่งเลื่อนข้อมูลคณิตศาสตร์มีด้วยกัน 7 คำสั่ง ดังนี้

ASLA (Arithmatic Shift Left A) เป็นคำสั่งเลื่อนข้อมูลใน A ไปทางซ้ายโดยบิต LSB ข้อมูล "0" เลื่อนเข้ามาแทนที่ บิต MSB จะเลื่อนไปที่บิตทด

นั่นคือ จากข้อมูล $40 เมื่อกระทำคำสั่ง ASLA ค่าจะกลายเป็น $80 ซึ่งก็คือเกิดการคูณ 2 ให้แก่ข้อมูลเดิม

ASLB (Arithmatic Shift Left B) มีหลักการทำงานเช่นเดียวกับ ASLA แต่เป็นการเลื่อนข้อมูลในแอกคิวมูเลเตอร์ B

ASL (Arithmatic Shift Left) เป็นคำสั่งเลื่อนข้อมูลในหน่วยความจำไปทางซ้าย มีรูปแบบการทำงานเช่นเดียวกับ ASLA และ ASLB

ASLD (Arithmatic Shift Left D) เป็นคำสั่งเลื่อนข้อมูล ของแอกคิวมูเลเตอร์ D ขนาด16 บิต มีรูปแบบการทำงานเช่นเดียวกับ ASLA และ ASLB หากแต่จำนวนข้อมูลเพิ่มจาก 8 บิต เป็น 16 บิต

ASRA (Arithmatic Shift Right A) เป็นคำสั่งเลื่อนข้อมูลใน A ไปทางขวา นั่นคือ จากข้อมูล $40 เมื่อกระทำคำสั่ง ASRA ค่าจะกลายเป็น $20 ซึ่งก็คือเกิดการหาร 2 ให้แก่ข้อมูลเดิม

ASRB (Arithmatic Shift Right B) มีหลักการทำงานเช่นเดียวกับ ASRA แต่เป็นการเลื่อนข้อมูลในแอกคิวมูเลเตอร์ B

ASR (Arithmatic Shift Right) เป็นคำสั่งเลื่อนข้อมูลในหน่วยความจำ มีรูปแบบการทำงานเช่นเดียวกับ ASRA และ ASRB

3) กลุ่มคำสั่งเลื่อนข้อมูลลอจิก

ในกลุ่มนี้คำสั่งในการเลื่อนข้อมูลปกติ ซึ่งก็มีทั้งเลื่อนข้อมูลไปทางซ้าย หรือขวา โดยจะให้ค่าศูนย์เลื่อนเข้ามาแทนที่บิต LSB ในกรณีเลื่อนข้อมูลไปทางซ้าย และเลื่อนค่าศูนย์เข้าที่บิต MSB ในกรณีเลื่อนข้อมูลไปทางขวา กลุ่มคำสั่งนี้มีด้วยกัน 8 คำสั่ง มีรายละเอียดดังนี้

LSLA (Logic Shift Left A)

ตารางที่ 2.6 คำสั่งกลุ่มเลื่อนและหมุนข้อมูล

คำสั่ง	ลักษณะการทำงาน	แฟลกของรีจิสเตอร์เงื่อนไข							
		S	X	H	I	N	Z	V	C
ROLA (ROLB)	ROTATE LEFT A	-	-	-	-	T	T	T	T
ROL	ROTATE LEFT M	-	-	-	-	T	T	T	T
RORA (RORB)	ROTATE RIGHT A	-	-	-	-	T	T	T	T
ROR	ROTATE RIGHT M	-	-	-	-	T	T	T	T
ASLA (ASLB)	ARITHAMATIC SHIFT LEFT A	-	-	-	-	T	T	T	T
ASL	ARITHAMATIC SHIFT LEFT M	-	-	-	-	T	T	T	T
ASLD	ARITHAMATIC SHIFT LEFT D	-	-	-	-	T	T	T	T
ASRA (ASLB)	ARITHAMATIC SHIFT RIGHT A	-	-	-	-	T	T	T	T
ASR	ARITHAMATIC SHIFT RIGHT M	-	-	-	-	T	T	T	T

ตารางที่ 2.6 (ต่อ) คำสั่งกลุ่มเลื่อนและหมุนข้อมูล

คำสั่ง	ลักษณะการทำงาน	แฟลกของรีจิสเตอร์เงื่อนไข							
		S	X	H	I	N	Z	V	C
LSLA (LSLB)	LOGIC SHIFT LEFT A	-	-	-	-	T	T	T	T
LSL	LOGIC SHIFT LEFT M	-	-	-	-	T	T	T	T
LSLD	LOGIC SHIFT LEFT D	-	-	-	-	T	T	T	T
LSRA (LSRB)	LOGIC SHIFT RIGHT A	-	-	-	-	T	T	T	T
LSR	LOGIC SHIFT RIGHT M	-	-	-	-	0	T	T	T
LSRD	LOGIC SHIFT RIGHT D	-	-	-	-	0	T	T	T

LSLB (Logic Shift Left B)

LSL (Logic Shift Left)

ทั้ง 3 คำสั่ง เป็นคำสั่งเลื่อนข้อมูลไปทางซ้าย แต่จะต่างกันที่ข้อมูลที่นำมากระทำ

LSLD (Logic Shift Left D) เป็นคำสั่งเลื่อนข้อมูลในแอกคิวมูเลเตอร์ D ไปทางซ้าย

ทุกคำสั่งการเลื่อนข้อมูลไปทางซ้าย จะมีผลต่อแฟลก N, Z, V และ C ทั้งสิ้น

LSRA (Logic Shift Right A)

LSRB (Logic Shift Right B)

LSR (Logic Shift Right)

เป็นคำสั่งให้หมุนข้อมูลไปทางขวา แต่จะต่างกันที่ข้อมูลที่นำมากระทำ

LSRD (Logic Shift Right D) เป็นคำสั่งเลื่อนข้อมูลในแอกคิวมูเลเตอร์ D ไปทางขวา

ทุกคำสั่งการเลื่อนข้อมูลไปทางขวา จะมีผลต่อแฟลก Z, V และ C ในขณะที่แฟลก N จะมีค่าเป็น "0"

2.2) กลุ่มคำสั่งทางคณิตศาสตร์ (Arithmatics instruction)

ในกลุ่มคำสั่งแบ่งได้อีก 3 กลุ่มคือ กลุ่มคำสั่งการบวก ลบ คูณและหาร

2.2.1) กลุ่มคำสั่งการบวก

มีด้วยกัน 9 คำสั่ง มีรายละเอียดดังนี้

ADDA และ ADDB (Add memory to A & Add memory to B) เป็นคำสั่งใช้ในการบวกค่าของแอกคิวมูเลเตอร์ A หรือ B เข้ากับค่าข้อมูลในหน่วยความจำ แล้วนำผลลัพธ์มาเก็บไว้ในแอกคิวมูเลเตอร์ A หรือ B การบวกข้อมูลทั้ง 2 คำสั่งนี้ เป็นการบวกข้อมูลขนาด 8 บิต

ABA (Add Accumulators) คำสั่งนี้เป็นการบวกค่าของแอกคิวมูเลเตอร์ ทั้ง 2 ตัวคือ A กับ B เข้าด้วยกัน ผลลัพธ์จะเก็บไว้ที่แอกคิวมูเลเตอร์ A

ADCA และ ADCB (Add With Carry to A & Add With Carry to B) เป็นคำสั่งใช้ในการบวกค่าของแอกคิวมูเลเตอร์ A หรือ B เข้ากับค่าข้อมูลขนาด 8 บิต โดยมีการคิดตัวทดด้วย แล้วนำผลลัพธ์มาเก็บไว้ในแอกคิวมูเลเตอร์ A หรือ B

ADDD (Add 16 bit to D) เป็นคำสั่ง ที่ใช้ในการบวกข้อมูล ขนาด 16 บิต ระหว่างแอกคิวมูเลเตอร์ D กับข้อมูลในหน่วยความจำ ผลลัพธ์เก็บไว้ที่ D

ABX และ ABY (Add B to X & Add B to Y) เป็นคำสั่ง ที่ใช้รวมค่าของแอกคิวมูเลเตอร์ B กับรีจิสเตอร์ IX และ IY โดย 8 บิตแรกของ IX และ IY จะถูกบวกด้วย 00 ส่วน 8 บิตหลังจะถูกรวมเข้ากับค่าของ B ผลลัพธ์จะเก็บไว้ที่ IX หรือ IY

DAA (Decimal Adjust A) เป็นคำสั่งใช้ในการปรับค่าในแอกคิวมูเลเตอร์ A ซึ่งเป็นไบนารี 8 บิตปกติเป็นเลขฐานสิบหรือรหัส BCD (Binary Code Decimal)

2.2.2) กลุ่มคำสั่งการลบ

มีทั้งสิ้น 9 คำสั่งดังนี้

SUB A และ SUB B (Subtractor memory to A & Subtractor memory to B) เป็นคำสั่งใช้ในการลบค่าของแอกคิวมูเลเตอร์ A หรือ B เข้ากับค่าข้อมูลในหน่วยความจำ แล้วนำผลลัพธ์มาเก็บไว้ในแอกคิวมูเลเตอร์ A หรือ B การลบข้อมูลทั้ง 2 คำสั่งนี้ เป็นการลบข้อมูลขนาด 8 บิต

SBA (Subtractor B from A) คำสั่งนี้เป็นการลบค่าของแอกคิวมูเลเตอร์ทั้ง 2 ตัวคือ A กับ B เข้าด้วยกัน ผลลัพธ์จะเก็บไว้ที่แอกคิวมูเลเตอร์ A

SBCA และ SBCB (Subtractor With Carry to A & Subtractor With Carry to B) เป็นคำสั่งใช้ในการลบค่าของแอกคิวมูเลเตอร์ A หรือ B เข้ากับค่าข้อมูลขนาด 8 บิต โดยมีการคิดตัวทดด้วย แล้วนำผลลัพธ์มาเก็บไว้ในแอกคิวมูเลเตอร์ A หรือ B

SUBD (Subtractor 16 bit to D) เป็นคำสั่ง ที่ใช้ในการลบข้อมูล ขนาด 16 บิตระหว่างแอกคิวมูเลเตอร์ D กับข้อมูลในหน่วยความจำ ผลลัพธ์เก็บไว้ที่ D

NEGA และ NEGB (2 's Complement A & 2 's Complement B) เป็นคำสั่งที่ใช้การจัดการข้อมูลในแอกคิวมูเลเตอร์ A และ B ให้เป็นค่า 2 's คอมพลีเมนต์โดยจะถูกลบด้วย 00 ตั้งลบด้วยค่าใน A หรือ B ผลลัพธ์จะเก็บไว้ที่ A หรือ B

ตารางที่ 2.7 คำสั่งทางคณิตศาสตร์

คำสั่ง	ลักษณะการทำงาน	แฟลกของรีจิสเตอร์เงื่อนไข							
		S	X	H	I	N	Z	V	C
ADDA (ADDB)	A ← A+M	-	-	T	-	T	T	T	T
ABA	A ← A+B	-	-	T	-	T	T	T	T
ADCA (ADCB)	A ← A+M+C	-	-	T	-	T	T	T	T
DAA	ปรับเลขใน A เป็นเลขฐานสิบ	-	-	-	-	T	T	T	T
ADDD	D ← D+(M:M+1)	-	-	-	-	T	T	T	T
ABX	X ← X+00:B	-	-	-	-	-	-	-	-
ABY	Y ← Y+00:B	-	-	-	-	-	-	-	-
SUBA (SUBB)	A ← A-M	-	-	-	-	T	T	T	T
SBA	A ← A-B	-	-	-	-	T	T	T	T
SBCA (SBCB)	A ← A-M-C	-	-	-	-	T	T	T	T
NEGA (NEGB)	A ← 0-A	-	-	-	-	T	T	T	T
NEG	M ← 0-M	-	-	-	-	T	T	T	T
SUBD	D ← D-(M:M+1)	-	-	-	-	T	T	T	T
MUL	D ← A*B	-	-	-	-	-	-	-	T
IDIV	X ← D/X:D ← r	-	-	-	-	-	T	0	T
FDIV	X ← D/X:D ← r	-	-	-	-	-	T	T	T

NEG (2 's Complement memory byte) เป็นคำสั่ง ที่ใช้การจัดการข้อมูลในหน่วยความจำ ให้เป็นค่า 2 's คอมพลีเมนต์ โดยจะถูกลบด้วย 00 ผลลัพธ์จะเก็บไว้ที่หน่วยความจำ

2.2.3) กลุ่มคำสั่งการคูณและหาร

มีด้วยกัน 3 คำสั่งดังนี้

MUL (Multiply 8 by 8) เป็นคำสั่งคูณข้อมูลขนาด 8 บิต ระหว่างข้อมูลในแอกคิวมูเลเตอร์ A กับ B ผลคูณจะถูกเก็บไว้ในแอกคิวมูเลเตอร์ D

IDIV (Integer Divide 16 by 16) เป็นคำสั่งการหารเลขจำนวนเต็มขนาด 16 บิต โดยตัวตั้งจะถูกเก็บไว้ใน D ส่วนตัวหารเก็บไว้ใน IX ผลหารจะถูกเก็บไว้ใน IX เศษของการหารเก็บใน D

FDIV (Fraction Divide 16 by 16) เป็นคำสั่งการหารเลขเศษส่วนขนาด 16 บิต โดยตัวตั้งจะถูกเก็บไว้ใน D ส่วนตัวหารเก็บไว้ใน IX ผลหารจะถูกเก็บไว้ใน IX

2.3) กลุ่มคำสั่งทางลอจิก (Logic instruction)

มีด้วยกันสามกระบวน คือ แอนด์ ออร์ และ เอกซ์คลูซีพออร์ รวมแล้วมีคำสั่งทั้งสิ้น 6 คำสั่งดังนี้

ANDA และ ANDB (AND A with memory & AND B with memory) เป็นคำสั่งใช้ในการแอนด์ข้อมูล 8 บิต ระหว่าง A หรือ B กับค่าข้อมูลใดๆ ผลของการแอนด์เก็บไว้ใน A หรือ B

ORAA และ ORAB (OR Accumulator A & OR Accumulator B) เป็นคำสั่งใช้ในการออร์ข้อมูล 8 บิต ระหว่าง A หรือ B กับค่าข้อมูลใดๆ ผลของการแอนด์เก็บไว้ใน A หรือ B

EORA และ EORB (Exclusive OR Accumulator A & Exclusive OR Accumulator B) เป็นคำสั่งใช้ในการออร์ข้อมูล 8 บิต ระหว่าง A หรือ B กับค่าข้อมูลใดๆ ผลของการแอนด์เก็บไว้ใน A หรือ B

ตารางที่ 2.8 คำสั่งทางลอจิก

คำสั่ง	ลักษณะการทำงาน	แฟลกของรีจิสเตอร์เงื่อนไข							
		S	X	H	I	N	Z	V	C
ANDA (ANDB)	A ← A AND M	-	-	-	-	T	T	0	-
ORAA (ORAB)	A ← A OR M	-	-	-	-	T	T	0	-
EORA (EORAB)	A ← A EOR M	-	-	-	-	T	T	0	-

2.4) กลุ่มคำสั่งเปรียบเทียบและตรวจสอบข้อมูล (Data test instruction)

แบ่งลักษณะและทดสอบได้สองลักษณะตามขนาดของข้อมูล 8 บิต และ 16 บิต ดังนี้

2.4.1) กลุ่มคำสั่งเปรียบเทียบและตรวจสอบข้อมูลขนาด 8 บิต

มีด้วยกัน 8 คำสั่ง และแต่ละคำสั่งเมื่อกระทำไปแล้ว จะมีผลต่อการเปลี่ยนแปลงของแฟล็ก N, Z, V และ C

CMPA และ CMPB (Compare A to memory & Compare B to memory) เป็นคำสั่งในการเปรียบเทียบระหว่าง แอกคิวมูเลเตอร์ A กับ B กับข้อมูลในหน่วยความจำ

CBA (Compare A to B) เป็นคำสั่งในการเปรียบเทียบระหว่าง แอกคิวมูเลเตอร์ A กับ B

TSTA และ TSTB (Test for zero or minus of A & Test for zero or minus of B) เป็นคำสั่งในการตรวจสอบว่าแอคคิวมูเลเตอร์ A และ B เป็นศูนย์หรือติดลบ

TST (Test for zero or minus of memory) เป็นคำสั่งในการตรวจสอบข้อมูลในหน่วยความจำว่าเป็นศูนย์หรือติดลบ

BITA และ BITB (Bit test A with memory & Bit test B with memory) เป็นการเปรียบเทียบข้อมูลในระดับบิตของแอคคิวมูเลเตอร์ A หรือ B กับหน่วยความจำหรือข้อมูลใดๆ นั่นเอง

ตารางที่ 2.9 คำสั่งเปรียบเทียบและตรวจสอบข้อมูล

คำสั่ง	ลักษณะการทำงาน	แฟลกของรีจิสเตอร์เงื่อนไข							
		S	X	H	I	N	Z	V	C
CMPA (CMPB)	A – M	-	-	-	-	T	T	T	T
CBA	A - B	-	-	-	-	T	T	T	T
TSTA (TSTB	A - 0	-	-	-	-	T	T	0	0
TST	M – 0	-	-	-	-	T	T	0	0
BITA (BITB)	A AND M	-	-	-	-	T	T	0	-
CPD	D - (M:M+1)	-	-	-	-	T	T	T	T
CPX	X - (M:M+1)	-	-	-	-	T	T	T	T
CPY	Y – (M:M+1)	-	-	-	-	T	T	T	T

2.4.2) กลุ่มคำสั่งการเปรียบเทียบและตรวจสอบข้อมูลขนาด 16 บิต

คำสั่งในกลุ่มนี้มีด้วยกัน 3 คำสั่ง และแต่ละคำสั่งเมื่อกระทำไปแล้ว จะมีผลต่อการเปลี่ยนแปลงของแฟล็ก N,Z,V และ C เพื่อใช้ประโยชน์ในเงื่อนไขของการกระโดดต่อไป

CPD (Compare D to memory 16 bit) เป็นคำสั่งในการเปรียบเทียบข้อมูลในแอคคิวมูเลเตอร์ D กับข้อมูลขนาด 16 บิต ในหน่วยความจำ

CPX (Compare IX to memory 16 bit) เป็นคำสั่งในการเปรียบเทียบข้อมูลในแอคคิวมูเลเตอร์ IX กับข้อมูลขนาด 16 บิต ในหน่วยความจำ

CPY (Compare IY to memory 16 bit) เป็นคำสั่งในการเปรียบเทียบข้อมูลในแอคคิวมูเลเตอร์ IY กับข้อมูลขนาด 16 บิต ในหน่วยความจำ

2.5) กลุ่มคำสั่งการกระโดด (Jump and branch instruction)

2.5.1) กลุ่มคำสั่งกระโดดแบบไม่มีเงื่อนไข

ในกลุ่มมีด้วยกัน 7 คำสั่ง ดังนี้

JMP (Jump) เป็นคำสั่งที่ให้ซีพียูข้ามไปจัดการข้อมูล หรือคำสั่งในหน่วยความจำตำแหน่งใดๆ ที่ถูกกำหนดไว้ในโอเปอร์แรนด์ ค่า PC จะเท่ากับแอดเดรสปลายทางที่ระบุให้กระโดดไป

JSR (Jump to Subroutine) เป็นคำสั่งที่ให้ซีพียูกระโดดไปทำงานยังโปรแกรมย่อย เมื่อทำคำสั่งค่าของ PC เดิมจะถูกเก็บไว้ในสแต็ก จากนั้นค่า PC จะถูกเปลี่ยนเป็นค่าแอดเดรสที่ต้องการกระโดดไป จากนั้น ซีพียูก็จะทำงานตามที่โปรแกรมย่อยกำหนด จะกลับมายังโปรแกรมหลักได้ก็ต่อเมื่อพบคำสั่ง RTS

BRA (Branch Always) เป็นคำสั่งให้ซีพียูข้ามไปทำงานที่แอดเดรสปลายทางที่ถูกระบุด้วยค่าสัมพัทธ์

BRN (Branch Never) เมื่อทำคำสั่งนี้ค่า PC จะเพิ่ม 2 ค่า นั่นคือ เป็นการสั่งให้ให้ซีพียูกระโดดข้ามไปทำงานที่แอดเดรสปลายทางที่อีก 2 แอดเดรสข้างหน้า

BSR (Branch to Subroutine) เป็นคำสั่งให้ซีพียูกระโดดไปทำงานที่โปรแกรมย่อย โดยค่าของ PC เดิมจะถูกเก็บไว้ในสแต็ก และค่า PC จะเปลี่ยนไปเป็นแอดเดรสกำหนดปลายทาง ซึ่งได้มาจากการคำนวณค่าสัมพัทธ์ และจะกลับมายังโปรแกรมหลักเมื่อพบคำสั่ง RTS

RTS (Return from Subroutine) เมื่อพบคำสั่งนี้ซีพียูจะกระโดดกลับไปทำงานที่โปรแกรมหลัก โดยเรียกค่า PC เดิมที่ถูกเก็บไว้ในสแต็กออกมา

RTI (Return from Interrupt) เมื่อมีการอินเตอร์รัปต์ ซีพียูจะทำการเก็บค่าของรีจิสเตอร์รหัสเงื่อนไข (CCR) แอกคิวมูเลเตอร์ A และ B ค่า IX, IY และค่า PC ไว้ในสแต็กทั้งหมด และเมื่อกระทำโปรแกรมตอบสนองการอินเตอร์รัปต์เรียบร้อยแล้ว พบคำสั่งนี้ซีพียูจะทำการดึงค่าของ CR, A, B, IX, IY และ PC คืนกลับมาจากสแต็กเพื่อทำงานปกติต่อไป

ตารางที่ 2.10 คำสั่งกระโดดแบบไม่มีเงื่อนไข

คำสั่ง	ลักษณะการทำงาน	แฟลกของรีจิสเตอร์เงื่อนไข							
		S	X	H	I	N	Z	V	C
JMP	PC ← M:M+1	-	-	-	-	-	-	-	-
JSR	Stack ← PC: PC← M:M+1	-	-	-	-	-	-	-	-
BRA	PC ← PC + M	-	-	-	-	-	-	-	-
BRN	PC ← PC + 2	-	-	-	-	-	-	-	-
BSR	Stack ← PC; PC ← PC + M	-	-	-	-	-	-	-	-
RTS	PC ← Stack	-	-	-	-	-	-	-	-
RTI	CC,B,A,X,Y,PC ← Stack	T	L	T	T	T	T	T	T

ความแตกต่างของการ Jump และ Branch

ในการพิจารณาว่าจะใช้คำสั่ง Jump หรือ Branch นั้น จะขึ้นอยู่กับผู้เขียนโปรแกรมโดยใช้คำสั่ง Jump อันได้แก่ JMP, JSR จะสามารถกระโดดไปที่ใดในหน่วยความจำก็ได้โดยกำหนดค่าแอดเดรสลงไปชัดเจน ทำให้ต้องเปลืองหน่วยความจำ แต่ถ้าเป็นคำสั่ง BRA หรือ BSR จะสามารถกระโดดไปในขอบเขต +127 และ –128 ตำแหน่งจากจุดที่กระทำคำสั่ง ทำให้ใช้หน่วยความจำในการเขียนโปรแกรมน้อยกว่าและซีพียูทำงานเร็วกว่า แต่ก็มีข้อด้อยตรงที่ขอบเขตการกระโดด

2.5.2) กลุ่มคำสั่งกระโดดแบบมีเงื่อนไข

สามารถแบ่งแยกย่อยได้อีก 4 กลุ่มย่อยดังนี้

1) กลุ่มคำสั่งกระโดดแบบทดสอบบิตในไบต์

มีด้วยกัน 2 คำสั่งคือ BRSET และ BRCLR

BRSET (Branch if Bit Set) เป็นคำสั่งที่ให้ซีพียูกระโดดไปทำงาน ยังแอดเดรสปลายทาง หลังจากที่มีการตรวจสอบข้อมูลบิตใดๆ ในหน่วยความจำแล้วพบว่าบิตที่ตรวจสอบนั้นเป็น “1” แต่ถ้าบิตที่ตรวจสอบนั้นเป็น “0” ก็จะไม่มีการกระโดด

BRCLR (Branch if Bit Clear) เป็นคำสั่งที่มีลักษณะการทำงานตรงข้ามกับ BRSET คือให้ซีพียูกระโดดไปแอดเดรสปลายทาง หลังจากที่มีการตรวจสอบข้อมูลบิตใดๆ ในหน่วยความจำแล้วพบว่าบิตที่ตรวจสอบนั้นเป็น “0” แต่ถ้าบิตที่ตรวจสอบนั้นเป็น “1” ก็จะไม่มีการกระโดด

ตารางที่ 2.11 กลุ่มคำสั่งกระโดดแบบทดสอบบิตในไบต์

คำสั่ง	ลักษณะการทำงาน	แฟลกของรีจิสเตอร์เงื่อนไข							
		S	X	H	I	N	Z	V	C
BRSET	Branch if (NOT M)	-	-	-	-	-	-	-	-
	AND mm = 00H	-	-	-	-	-	-	-	-
BRCLR	Branch if M AND mm = 00H	-	-	-	-	-	-	-	-

2) กลุ่มคำสั่งกระโดดแบบทดสอบแฟลก มีด้วยกัน 8 คำสั่ง

BCC (Branch if Carry Clear) ซีพียูจะกระโดดเมื่อตรวจสอบบิตตัวทดแล้วพบว่าเป็น “0”

BCS (Branch if Carry Set) ซีพียูจะกระโดดเมื่อตรวจสอบบิตตัวทดแล้วพบว่าเป็น “1”

BNE (Branch if Not Zero) เป็นคำสั่งให้ซีพียูกระโดดเมื่อตรวจสอบบิตศูนย์ (Zero) แล้วพบว่าเป็น “0”

BEQ (Branch if Zero) เป็นคำสั่งให้ซีพียูกระโดดเมื่อตรวจสอบบิตศูนย์ (Zero) แล้วพบว่าเป็น “1” นั่นคือ เกิดค่าศูนย์ในกระบวนการประมวลผลข้อมูล

BPL (Branch if Plus) เป็นคำสั่งให้ซีพียูกระโดดเมื่อตรวจบิตลบแล้วพบว่าเป็น “0” นั่นคือ ค่าข้อมูลเป็นบวก

BMI (Branch if Minus) เป็นคำสั่งให้ซีพียูกระโดดเมื่อตรวจบิตลบแล้วพบว่าเป็น “1” นั่นคือ ค่าข้อมูลเป็นลบ

BVC (Branch if Overflow Clear) เป็นคำสั่งให้ซีพียูกระโดดเมื่อตรวจสอบบิตเกิน (Overflow) แล้วพบว่าเป็น “0” นั่นคือ ไม่เกิดค่าเกินย่านที่กำหนดในกระบวนการประมวลผลข้อมูล (ค่าเกินคือ ค่าที่มากหรือน้อยกว่า +127 ถึง –128)

BVS (Branch if Overflow Set) เป็นคำสั่งให้ซีพียูกระโดดเมื่อตรวจสอบบิตเกิน (Overflow) แล้วพบว่าเป็น “1” นั่นคือ ไม่เกิดค่าเกินย่าน +127 ถึง –128 ในการประมวลผลข้อมูล

ตารางที่ 2.12 กลุ่มคำสั่งกระโดดแบบทดสอบแฟลก

คำสั่ง	ลักษณะการทำงาน	แฟลกของรีจิสเตอร์เงื่อนไข							
		S	X	H	I	N	Z	V	C
BCS	Branch if Carry Set	-	-	-	-	-	-	-	-
BCC	Branch if Carry Clear	-	-	-	-	-	-	-	-
BEQ	Branch if Zero	-	-	-	-	-	-	-	-
BNE	Branch if Not Zero	-	-	-	-	-	-	-	-
BMI	Branch if Minus	-	-	-	-	-	-	-	-
BPL	Branch if Plus	-	-	-	-	-	-	-	-
BVS	Branch if Overflow Set	-	-	-	-	-	-	-	-
BVC	Branch if Overflow Clear	-	-	-	-	-	-	-	-

3) กลุ่มคำสั่งกระโดดแบบเปรียบเทียบข้อมูลศูนย์

ในกลุ่มคำสั่งนี้จะมีการเปรียบเทียบข้อมูลกับค่าศูนย์มากกว่า น้อยกว่าหรือเท่ากับ โดยมีการคำนึงถึงเครื่องหมายของข้อมูลด้วยว่าเป็นลบหรือบวกมีด้วยกันทั้งสิ้น 6 คำสั่งดังนี้

BLT (Branch if < zero) เมื่อทำคำสั่งนี้ซีพียูจะตรวจสอบการเอกซ์คลูซีฟออร์ของบิตลบ (N) กับบิตเกิน (V) ใน CCR ว่าเป็น “1” หรือไม่ ถ้าเป็นนั่นแสดงว่าข้อมูลที่ทำการเปรียบเทียบน้อยกว่า “0” ซีพียูจะทำการกระโดด

BLE (Branch if < zero) เมื่อทำคำสั่งนี้ซีพียูจะตรวจสอบข้อมูล ว่าน้อยกว่าหรือเท่ากับ “0” หรือไม่ ถ้าใช่ซีพียูก็กระโดดไปทำงานตามแอดเดรสที่กำหนดต่อไป

BEQ (Branch if = zero) นอกจากจะใช้คำสั่งนี้ ในการกระโดดแบบทดสอบแฟลกแล้ว ยังสามารถใช้โดยกลุ่มกลุ่มคำสั่งกระโดแบบเปรียบเทียบ โดยคำนึงถึงเงื่อนไขได้ด้วยนั่นคือ ถ้าเปรียบเทียบข้อมูลแล้วพบว่าเท่ากับ “0” ซีพียูก็กระโดดไปทำงานตามแอดเดรสที่กำหนด

BEE (Branch if > zero) เป็นคำสั่งที่ให้ซีพียูกระโดด เมื่อผลการเปรียบเทียบมีค่ามากกว่า “0”

BGT (Branch if > zero) เป็นคำสั่งที่ให้ซีพียูกระโดด เมื่อทำการเปรียบเทียบมีค่ามากกว่า “0”

BNE (Branch if not = zero) เป็นคำสั่งที่ทำให้ซีพียูกระโดด เมื่อเปรียบเทียบแล้วพบว่าข้อมูลนั้นไม่ใช่ค่า “0”

ตารางที่ 2.13 กลุ่มคำสั่งกระโดดแบบเปรียบเทียบข้อมูลศูนย์

คำสั่ง	ลักษณะการทำงาน	แฟลกของรีจิสเตอร์เงื่อนไข							
		S	X	H	I	N	Z	V	C
BLT	Branch if < Zero	-	-	-	-	-	-	-	-
BLE	Branch if <= Zero	-	-	-	-	-	-	-	-
BEQ	Branch if Zero	-	-	-	-	-	-	-	-
BGT	Branch if > Zero	-	-	-	-	-	-	-	-
BNE	Branch if not Zero	-	-	-	-	-	-	-	-

4) กลุ่มคำสั่งแบบเปรียบเทียบข้อมูล 2 ข้อมูล

เป็นกลุ่มคำสั่งที่ใช้กำหนดเงื่อนไขการกระโดดหลังจากการเปรียบเทียบข้อมูลมากกว่า น้อยกว่า หรือเท่ากัน ที่เปรียบเทียบกับค่าศูนย์ที่เป็นหลักมีด้วยกัน 6 คำสั่งดังนี้

BLO (Branch if lower) เป็นคำสั่งที่ให้ซีพียูกระโดดเมื่อ ผลการเปรียบเทียบของข้อมูลหลักปรากฏว่าน้อยกว่า โดยบิตทดจะเซตเป็น “1”

BLS (Branch if lower 0r same) เป็นคำสั่งให้ซีพียูกระโดดเมื่อผลการเปรียบเทียบข้อมูล ปรากฏว่าข้อมูลหลักน้อยกว่าหรือเท่ากับข้อมูลที่นำมาเปรียบเทียบโดยซีพียูจะตรวจสอบผลได้จากการเซตของบิตศูนย์หรือบิตทด

BHS (Branch if Higher or Same) เป็นคำสั่งให้ซีพียูกระโดดเมื่อข้อมูลหลักมากกว่าหรือเท่ากับข้อมูลที่นำมาเปรียบเทียบ โดย:ซีพียูจะตรวจสอบผลจากการเคลียร์ของบิตทด (C=0)

BHI (Branch if Higher) เป็นคำสั่งให้ซีพียูกระโดดเมื่อข้อมูลมากกว่าข้อมูลที่นำมาเปรียบเทียบโดย:ซีพียูจะตรวจสอบผลจากการออร์กันของบิตศูนยีและบิตทดต้องได้ศูนย์ (C+Z=0)

BNE (Branch if not = zero) เป็นคำสั่งให้ซีพียูกระโดด หากเปรียบเทียบข้อมูลทั้งสองแล้วไม่เท่ากันโดยซีพียูตรวจสอบได้จากบิตศูนย์ต้องเป็น “0” (Z = 0)

ข้อสังเกต คำสั่ง BNE สามารถนำไปใช้ในกลุ่มกระโดดแบบมีเงื่อนไขได้ทุกแบบ ทั้งที่ผลของการทำงานก็เหมือนกัน ทั้งนี้เนื่องจากขอบเขตของการทำงานค่อนข้างกว้างมากนั่นเอง คำสั่ง BNE จะให้ซีพียูกระโดด เมื่อผลการเปรียบเทียบแล้วข้อมูลไม่เท่ากัน ไม่เท่ากับศูนย์ ในขณะที่คำสั่ง BEQ จะให้ซีพียูกระโดดเมื่อผลการเปรียบเทียบแล้วข้อมูลเท่ากันหรือเท่ากับศูนย์

ตารางที่ 2.14 กลุ่มคำสั่งกระโดดแบบเปรียบเทียบข้อมูล 2 ข้อมูล

คำสั่ง	ลักษณะการทำงาน	แฟลกของรีจิสเตอร์เงื่อนไข							
		S	X	H	I	N	Z	V	C
BLO	Branch if Lower	-	-	-	-	-	-	-	-
BLS	Branch if Lower or Same	-	-	-	-	-	-	-	-
BEQ	Branch if Zero	-	-	-	-	-	-	-	-
BHS	Branch if Higher or Same	-	-	-	-	-	-	-	-
BHI	Branch if Higher	-	-	-	-	-	-	-	-
BNE	Branch if Not Zero	-	-	-	-	-	-	-	-

2.6) กลุ่มคำสั่งจัดการกับรีจิสเตอร์รหัสเงื่อนไข (CCR instruction)

ในกลุ่มนี้เป็นคำสั่งในการเคลียร์บิตต่างๆในรีจิสเตอร์รหัสเงื่อนไข มีทั้งหมด 6 คำสั่ง คือ

SEC (Set Carry) เป็นคำสั่งที่ใช้ในการเซตบิตทด (Carry) ให้เป็น “1”

CLC (Clear Carry) เป็นคำสั่งเคลียร์บิตทดให้เป็นศูนย์

SEV (Set two’s complement overflow bit) เป็นคำสั่งเซตบิตโอเวอร์โฟลวให้เป็น “1”

CLV (Clear two’s complement overflow bit) เป็นคำสั่งเคลียร์บิตโอเวอร์-โฟลวให้เป็น “0”

SEI (Set interrupt mask) เป็นคำสั่งเซตอินเตอร์รัพท์มาร์กใน CCR ให้เป็น “1” เพื่อเป็นการดิสเอเบิลอินเตอร์รัปต์

CLI (Clear interrupt mask) เป็นคำสั่งเคลียร์อินเตอร์รัพท์มาร์กใน CCR ให้เป็น “1” เพื่อเป็นการอินาเบิลอินเตอร์รัปต์

ตารางที่ 2.15 คำสั่งจัดการกับรีจิสเตอร์รหัสเงื่อนไข (CCR instruction)

คำสั่ง	ลักษณะการทำงาน	แฟลกของรีจิสเตอร์เงื่อนไข							
		S	X	H	I	N	Z	V	C
SEC	C ← 1	-	-	-	-	-	-	-	1
CLC	C ← 0	-	-	-	-	-	-	-	0
SEV	V ← 1	-	-	-	-	-	-	1	-
CLV	V ← 0	-	-	-	-	-	-	0	-
SEI	I ← 1 (distable interupts)	-	-	-	1	-	-	-	-
CLI	I ← 0 (enable interupts)	-	-	-	0	-	-	-	-

2.6) กลุ่มคำสั่งควบคุม (Control instruction)

เป็นกลุ่มคำสั่งที่ใช้ในการควบคุมการทำงานหลักของไมโครคอนโทรลเลอร์ซึ่งจะเกี่ยวข้องกับการอินเตอร์รัปต์ และโหมดการทำงานประหยัดพลังงานมีด้วยกัน 4 คำสั่งคือ

SWI (Solfware Interrupt) เป็นคำสั่งอินเตอร์รัปต์ 68HC11 โดยใช้ซอฟแวร์ซึ่งตามปกติการอินเตอร์รัปต์มักเกิดจากสัญญาณทางฮาร์แวร์ แต่ใน 68HC11 สามารถอินเตอร์รัปต์ทางซอฟแวร์ได้ด้วยคำสั่งนี้ เมื่อกระทำคำสั่งซีพียูจะทำการเก็บค่ารีจีสเตอร์ต่างๆไว้ในแสต็ก ชนิดบิต I ใน CCR เพื่อบอกสถานะการอินเตอร์รัพท์ สถานะของการอินเตอร์รัปต์

WAIT (Wait for Interrupt) เป็นคำสั่งให้ 68HC11 รอรับการอินเตอร์รัปต์โดย เมื่อกระทำคำสั่งนี้ ซีพียูจะเก็บค่ารีจีสเตอร์ทุกตัวไว้ในแสต็ก แล้วจะ Halt เพื่อหยุดการทำงาน จากนั้นรอการอินเตอรัปต์หรือจะเรียกสภาวะนี้ว่า “Wait State”

STOP (Stop internal clock) เมื่อกระทำคำสั่งนี้ระบบสัญญาณนาฬิกาภายในชิปจะหยุดทำงาน ทำให้ซีพียูอยู่ในสภาวะแสตนด์บาย กินกำลังไฟฟ้าต่ำ คำสั่งนี้จะไม่มีผลต่อข้อมูลภายในรีจีสเตอร์หรือแรมภายในชิป 68HC11 จะตอบสนองคำสั่งนี้หรือไม่ขึ้นอยู่กับการกำหนดบิต S ใน CCR ถ้าหากเป็น “1” จะเป็นการดิสเอเบิลคำสั่ง STOP โดยเมื่อเอ็กซีคิวต์คำสั่งนี้ 68HC11 จะมองเหมือนคำสั่ง NOP คือไม่มีการทำงานอะไรทั้งสิ้น เพียงแต่เพิ่มค่า PC เท่านั้น ในทางตรงข้ามถ้าบิต S เป็น “0” จะเป็นการอินาเบิลคำสั่งนี้แก่ซีพียู ซึ่งเมื่อเอ็กซีคิวต์แล้ว ต้องทำงานตามที่กำหนด

เหมือนคำสั่ง NOP คือไม่มีการทำงานอะไรทั้งสิ้น เพียงแต่เพิ่มค่า PC เท่านั้น ในทางตรงข้ามถ้าบิต S เป็น “0” จะเป็นการอินาเบิลคำสั่งนี้แก่ซีพียู ซึ่งเมื่อเอกซีคิวต์แล้ว ต้องทำงานตามที่กำหนด

NOP (No Operation) เป็นคำสั่งที่ทำให้ค่า PC เพิ่มขึ้น ไม่มีรีจีสเตอร์ตัวใดได้รับผลกระทบจากคำสั่งนี้ มักใช้คำสั่งนี้เมื่อต้องการหน่วงเวลาการทำงานของ 68HC11

ตารางที่ 2.17 คำสั่งควบคุม (Control instruction)

คำสั่ง	ลักษณะการทำงาน	แฟลกของรีจิสเตอร์เงื่อนไข							
		S	X	H	I	N	Z	V	C
SWI	Stack ← PC,Y,X,A B, CC; 1 ← 1; PC ← M:M+1	-	-	-	1	-	-	-	-
WAI	Stack ←PC, Y ,X, A, B, CC; HALT	-	-	-	-	-	-	-	-
STOP		-	-	-	-	-	-	-	-
NOP	PC ← PC + 1	-	-	-	-	-	-	-	-

2.3 การเขียนโปรแกรมด้วยวิชวลเบสิก

การเขียนโปรแกรมคือ การสั่งให้คอมพิวเตอร์ทำงานตามที่ต้องการ เช่น โปรแกรมฝึกพิมพ์ดีด ซึ่งเป็นโปรแกรมที่สั่งให้เครื่องคอมพิวเตอร์โต้ตอบกับการกดแป้นคีย์บอร์ด (Keyboard) เพื่อสอนผู้ใช้พิมพ์ดีด เป็นต้น

สำหรับโปรแกรมวิชวลเบสิกเป็นเครื่องมือในการสร้างโปรแกรมบนระบบปฏิบัติการวินโดวส์ (Windows) ที่ใช้งานง่าย โดยการสร้างโปรแกรมในวิชวลเบสิก 6 นั้น เป็นการเลือกเครื่องมือต่าง ๆ มาออกแบบหน้าจอหน้าโปรแกรมที่จะสร้าง ซึ่งเรียกการเขียนโปรแกรมลักษณะนี้ว่าวิชวลโปรแกรมมิ่ง (Visual Programming) การเขียนโปรแกรมแบบนี้ จะไม่จำเป็นต้องเขียนคำสั่งต่าง ๆ มากนัก ก็สามารถสร้างโปรแกรมได้อย่างรวดเร็ว

หลังจากที่ได้ออกแบบหน้าจอโปรแกรมตามวิธีที่กล่าวมาแล้ว จะต้องเขียนโปรแกรมควบคุมการทำงานด้วย โดยใช้ภาษา BASIC (ย่อมาจาก Beginners All – Purpose Symbolic

Instruction Code) ซึ่งเป็นภาษาที่ใช้งานง่าย เหมาะสำหรับผู้เริ่มต้นศึกษาการเขียนโปรแกรมบนวินโดวส์สำหรับวิชวลเบสิก 6 นั้นเป็นเครื่องมือที่สร้างโปรแกรมต่างๆ ได้หลากหลายดังต่อไปนี้

1) โปรแกรมทั่วไปที่รันบนระบบปฏิบัติการวินโดวส์ โดยสามารถสร้างโปรแกรมทางด้านกราฟฟิก โปรแกรมจัดการไฟล์ โปรแกรมคำนวณเลขพื้นฐาน ให้ตรงกับความต้องการของการใช้งานเป็นต้น

2) โปรแกรมฐานข้อมูล วิชวลเบสิก 6 นั้นช่วยให้การสร้างโปรแกรมฐานข้อมูลเป็นเรื่องง่าย เนื่องจากกการมีเครื่องมือต่างๆ เกี่ยวกับฐานข้อมูลอย่างครบถ้วน เช่น เครื่องมือในการติดต่อกับฐานข้อมูลทั้งไมโครซอฟท์แอคเซส (Microsoft Access) หรือฐานข้อมูลบนระบบไคลเอนท์เซฟเวอร์ (Client Sever) เช่น ไมโครซอฟท์เอสคิวแอลเซิร์ฟเวอร์ (Microsoft SQL Server) โดยการติดต่อกับฐานข้อมูลนั้น เพียงแต่กำหนดตำแหน่งของฐานข้อมูลก็จะสามารถติดต่อกับฐานข้อมูลนั้นได้ทันที นอกจากนี้ในวิชวลเบสิก 6 ก็มีเครื่องมือในการสร้างรายงานสรุปข้อมูลจากฐานข้อมูลที่ใช้งานง่าย โดยการใช้เมาส์ (Mouse) ลากฟิลด์ (Field) ข้อมูลที่ต้องการไปที่ที่ต้องการในรายงานที่ออกแบบได้เลย

3) คอมโพเนนต์ (Component) ทางด้านแอกทีฟเอกซ์ (ActiveX) ซึ่งก็ได้แก่ ActiveX Component, ActiveX Control และ ActiveX Document เป็นเครื่องมือที่ช่วยให้สามารถนำส่วนของโปรแกรมที่สร้างแล้วไปใช้ในโปรแกรมอื่นๆ ได้ เช่น ไมโครซอฟท์เอกซ์เซล (Microsoft Excel) เป็นต้น

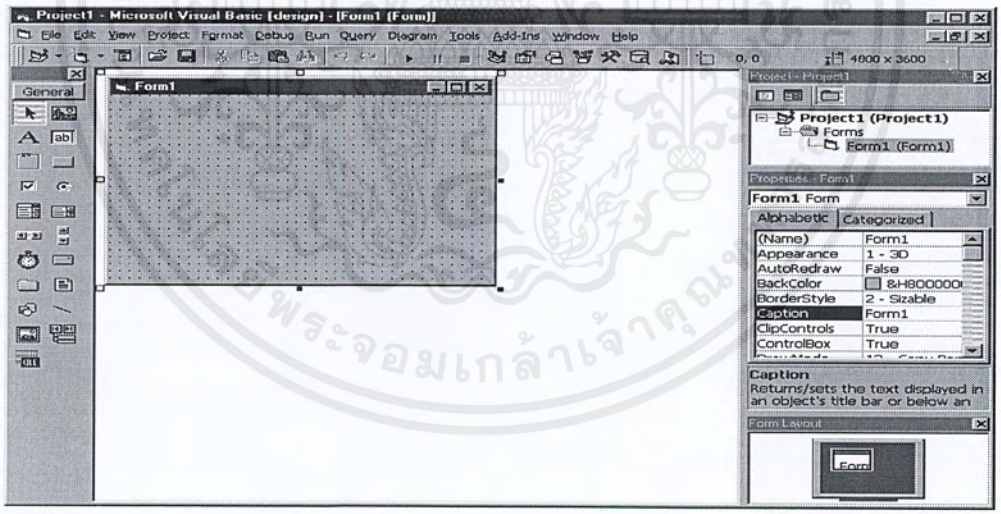
4) โปรแกรมที่รันบนอินเตอร์เนต (Internet) ด้วยความสามารถของวิชวลเบสิก 6 ช่วยให้สามารถสร้างโปรแกรมที่ใช้งานบนอินเตอร์เนตได้อย่างง่าย โดยที่ไม่ต้องเรียนรู้การเขียนคำสั่งด้วยภาษา HTML (Hypertext Markup Language) หรือภาษาสคริปต์ (Script) ที่ใช้งานบนอินเตอร์เนต

2.3.1 ส่วนประกอบของโปรแกรมวิชวลเบสิก

เมื่อทำการเปิดวิชวลเบสิกขึ้นมาจะปรากฏหน้าจอภาพดังรูปที่ 2.15 โดยทั่วไปแล้วจะเลือก Standard.EXE ซึ่งเป็นการใช้วิชวลเบสิกในการเขียนโปรแกรมที่ใช้บนในวินโดวส์ทั่วไป เนื่องจากโปรเจ็กต์ (Project) ชนิด Standard.EXE เป็นการเขียนโปแกรมทั่วไปที่ใช้บนวินโดวส์ โดยทั่วไปโปรเจ็กต์ คือ กลุ่มของไฟล์ที่จะนำมารวมกันเพื่อสร้างโปรแกรมในวิชวลเบสิก

รูปที่ 2.15 หน้าต่างเริ่มต้นเข้าใช้งานโปรแกรมวิชวลเบสิก

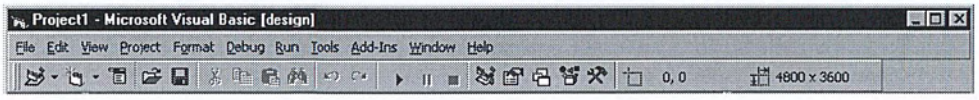
เมื่อเลือกเสร็จจะปรากฏหน้าจอของวิชวลเบสิกโดยมีชื่อของส่วนประกอบต่างๆ ที่จำเป็นจะต้องทราบดังรูปที่ 2.16



รูปที่ 2.16 หน้าต่างของโปรแกรมวิชวลเบสิก

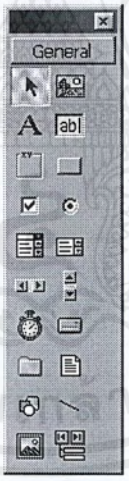
จากรูปที่ 2.16 หน้าต่างของโปรแกรมวิชวลเบสิกได้ประกอบไปด้วยส่วนต่างๆ ซึ่งแต่ละส่วนจะมีหน้าที่ และการทำงานที่แตกต่างกัน ซึ่งรายละเอียดมีดังนี้

1) ทูลบาร์ (Tool bar) เมื่อพิจารณาหน้าต่างของโปรแกรมวิชวลเบสิกจะมีปุ่มต่างๆ ที่วางเป็นแผงควบคุม ช่วยให้สามารถเรียกใช้งานคำสั่งได้อย่างสะดวกรวดเร็ว โดยเพียงแค่คลิกเมาส์ที่ปุ่มเท่านั้น ซึ่งดังแสดงรูปที่ 2.17



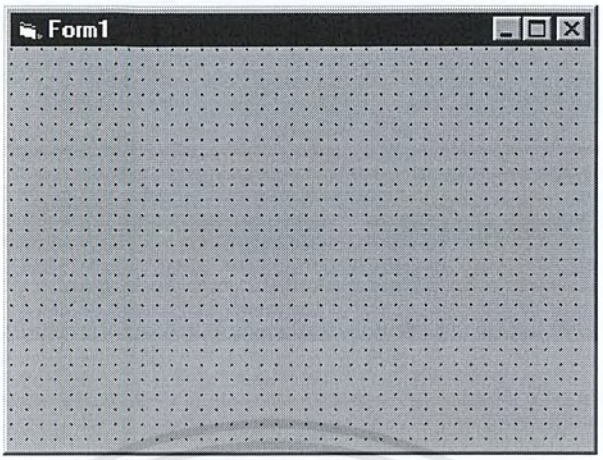
รูปที่ 2.17 ทูลบาร์ของโปรแกรมวิชวลเบสิก

2) ทูลบ็อกซ์ (Toolbox) เป็นที่รวมออบเจ็ก (Object) ต่างที่จะนำมาประกอบกันเป็นโปรแกรม เมื่อใช้ออบเจ็กเหล่านี้ประกอบกันจะได้เป็นหน้าตาของโปรแกรมอาจเรียกให้ชัดเจนได้ว่าคอนโทรลออบเจ็ก (Control Object) ซึ่งมีออบเจ็กหลักดังรูปที่ 2.18 นอกจากนี้สามารถที่จะทำการเพิ่มออบเจ็กต่างๆ เข้าไปในทูลบ็อกซ์ได้อีกมากมาย



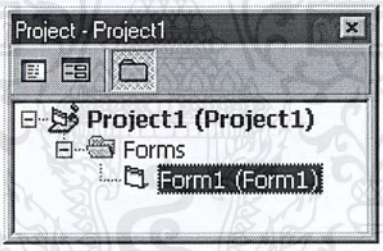
รูปที่ 2.18 ทูลบ็อกซ์ของโปรแกรมวิชวลเบสิก 6

3) หน้าต่างฟอร์ม (Form) เป็นหน้าต่างเปล่าๆ หรือตัวฟอร์ม (Form) สำหรับในโครงสร้างองค์ประกอบของแอพพลิเคชัน (Application) เมื่อเริ่มเข้าสู่โปรแกรมวิชวลเบสิกจะปรากฏฟอร์มเปล่าให้เสมอ ซึ่งหน้าตาของหน้าต่างฟอร์มดังแสดงในรูปที่ 2.19



รูปที่ 2.19 หน้าต่างฟอร์มของโปรแกรมวิชวลเบสิก

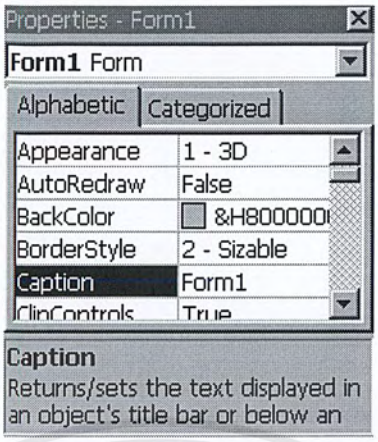
4) หน้าต่างโปรเจ็กเอ็กซ์โพลเรอร์ (Project Explorer) โปรแกรมต่างที่สร้างขึ้นมานั้น เรียกว่า โปรแกรมประยุกต์ หรือ แอพพลิเคชัน ซึ่งในวิชวลเบสิกจะเรียกโปรแกรมที่กำลังสร้างอยู่ว่าเป็นโครงการหรือโปรเจ็ก หน้าต่างของโปรเจ็กเอ็กซ์โพลเรอร์มีลักษณะดังแสดงในรูปที่ 2.20



รูปที่ 2.20 หน้าต่างโปรเจ็กเอ็กซ์โพลเรอร์ของโปรแกรมวิชวลเบสิก

หน้าต่างโปรเจ็กเอ็กซ์โพลเรอร์จะใช้ควบคุมส่วนประกอบ และแฟ้มข้อมูลต่างๆ ที่อยู่ในโปรเจ็กต์ โครงสร้างแฟ้มข้อมูลส่วนต่างที่ประกอบขึ้นมาเป็นโปรเจ็กต์ เพื่อความสะดวกในการที่จะควบคุม และเปลี่ยนการทำงานระหว่างส่วนต่างๆ โดยแต่ละโปรเจ็กต์จะประกอบไปด้วยแฟ้มข้อมูลมากมายหลายประเภท ซึ่งจะแสดงรายละเอียดดังในตารางที่ 2.18

5) หน้าต่างพรอบเพอร์ตี้ (Properties) หน้าต่างนี้จะแสดงคุณสมบัติทั้งหมดของออบเจ็กที่ถูกเลือกอยู่ การคลิกเลือกที่ออบเจ็กใดในฟอร์ม จะทำให้คุณสมบัติที่แสดงในหน้าต่างพรอบเพอร์ตี้เปลี่ยนไปตามออบเจ็กที่เลือก ซึ่งในการแก้ไขหรือตั้งค่าของคุณสมบัติต่างสามารถทำได้โดยตรงที่คุณสมบัติแต่ละค่า ซึ่งหน้าต่างพรอบเพอร์ตี้แสดงได้ดังรูปที่ 2.21



รูปที่ 2.21 หน้าต่างพรอบเพอร์ตี้

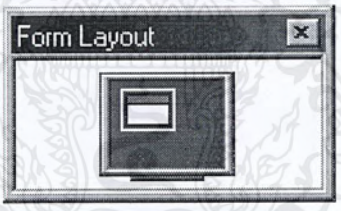
ตารางที่ 2.17 ประเภทของแฟ้มข้อมูลในโปรเจ็กเอ็กโพลเรอร์

ประเภทไฟล์	รายละเอียด	นามสกุลไฟล์
ไฟล์โปรเจ็ก (Project File)	เก็บข้อมูลต่างๆ ของโปรเจ็กรวมทั้งรายชื่อไฟล์ที่ประกอบขึ้นมาเป็นโปรเจ็ก	Vbp
ไฟล์ฟอร์ม (Form File)	เก็บข้อมูลที่ออกแบบไว้โดยในไฟล์นี้รวมคำสั่งต่างๆ ที่เขียนโปรแกรมไว้ให้กับแต่ละออบเจ็กทีอยู่ในฟอร์ม	Frm
ไฟล์ไบนารีฟอร์ม	จะเก็บข้อมูลที่เป็นแฟ้มไบนารีของฟอร์ม เช่น รูปภาพ หรือ ไอคอน เป็นต้น	Frx
ไฟล์โมดูลแบบปกติ (Standard Module)	เก็บโปรแกรมย่อยและตัวแปรต่างๆ ที่เขียนแยกจากฟอร์มเพื่อให้ฟอร์มหรือโมดูลอื่นเรียกใช้งานได้	Bas
ไฟล์ออบเจ็กคอลโทรล (Object Control)	นามสกุลลงท้ายด้วย ocx (Active control) หรือ vbx เป็นออบเจ็กที่เพิ่มเข้าไปในโปรเจ็กนอกเหนือจากคอนโทรลพื้นฐาน	Ocx Vbx
ไฟล์เอกสาร (ActiveX)	เหมือนกับฟอร์ม เพียงแต่ต้องเรียกผ่านโปรแกรมเว็บเบาว์เซอร์	Dob
ไฟล์คลาสโมดุล (Class Module)	เก็บออบเจ็กต่างๆ ที่สร้างขึ้น เมื่อมีการเรียกใช้คลาสโมดูล โปรแกรมจะสร้างออบเจ็กนั้นขึ้นมาใหม่แทนที่	Cls

ตารางที่ 2.17 (ต่อ) ประเภทของแฟ้มข้อมูลในโปรเจ็กเอ็กโพลเรอร์

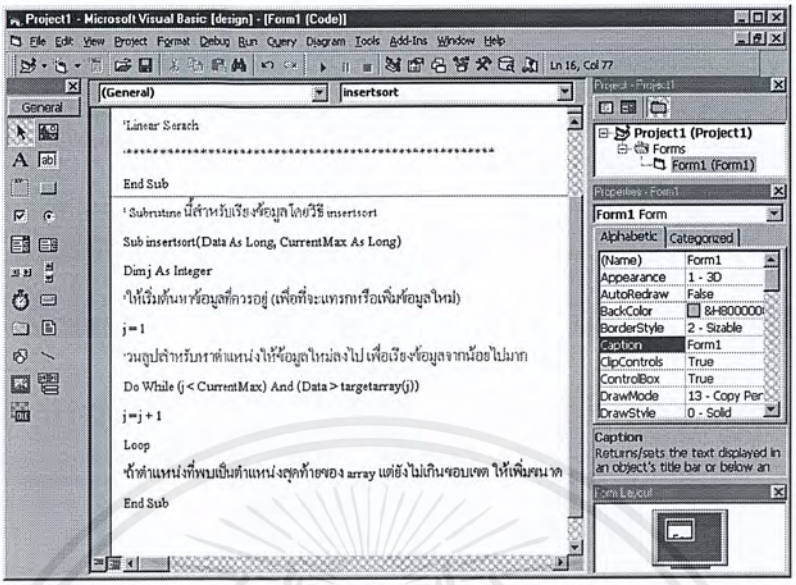
ประเภทไฟล์	รายละเอียด	นามสกุลไฟล์
	จะใช้จากโปรแกรมหรือออบเจ็กโดยตรง อาจกล่าวได้ว่าคลาสโมดูลเปรียบเสมือนที่เก็บแผงหนังสือ หรือเทมเพลท (Template) ของออบเจ็กที่จะสร้างขึ้นมาใหม่นั่นเอง	
ไฟล์ทรัพยากรอื่นๆ (Resource File)	เก็บภาพนามสกุล bmp ข้อความหรือข้อมูลใดๆ ที่สามารถแก้ไขได้โดยไดมต้องไปยุ่งเกี่ยวกับโปรแกรมในโมดูลหรือฟอร์มต่างๆ ในโปรเจ็กต์	Res

6) หน้าต่างฟอร์มเลเอาต์ (Form Layout) จะแสดงตำแหน่งฟอร์มของโปรแกรมที่กำลังสร้างให้ดูบนจอภาพ เพื่อกำหนดตำแหน่ง สำหรับตอนที่โปรแกรมทำงานจริง การย้ายตำแหน่งทำได้โดยการใช้เมาส์ลาก (Drag) รูปฟอร์มในจอภาพไปยังตำแหน่งที่ต้องการ ดังแสดงในรูปที่ 2.22



รูปที่ 2.22 หน้าต่างฟอร์มเลเอาต์

7) หน้าต่างโค้ดอิดิเตอร์ (Code Editor) เป็นเนื้อที่สำหรับในเขียนโปรแกรม ซึ่งหน้าต่างโค้ด อิดิเตอร์จะแสดงขึ้นมาพร้อมสำหรับการเขียนโปรแกรมให้กับเหตุการณ์ (Event) หลักของออบเจ็กนั้น ส่วนที่สำคัญของหน้าต่างนี้คือ คอมโบบ็อกซ์ (Combo Box) ทั้งสองช่องที่อยู่ตรงส่วนบนของหน้าต่าง ซึ่งจะเป็นตัวควบคุมการเลือกออบเจ็ก และเหตุการณ์จะเกิดขึ้นกับออบเจ็กนั้น โดยโค้ดที่ปรากฏจะเป็นโปรแกรม หรือคำสั่งที่จะถูกเรียกใช้งานเมื่อมีเหตุการณ์นั้นเกิดขึ้นกับออบเจ็กหน้าต่างของโค้ดอิดิเตอร์จะดังแสดงในรูปที่ 2.23 ซึ่งสามารถเข้าไปเขียนโปรแกรมควบคุมนี้ได้โดยการคลิกคอนโทรลที่ต้องการแล้วจะปรากฏหน้าต่างนี้ให้เห็น



รูปที่ 2.23 หน้าต่างโค้ดอิดิเตอร์

2.3.2 หลักการในการเขียนโปรแกรมด้วยวิชวลเบสิก

พื้นฐานเกี่ยวกับหลักการที่ควรทราบในการพัฒนาโปรแกรมด้วยวิชวลเบสิก คือ ต้องรู้จักคอนโทรล คุณสมบัติ และการเขียนโปรแกรมแบบตอบสนองต่อเหตุการณ์ (Event Driven) ซึ่งมีความสำคัญมากในการเขียนโดยใช้วิชวลเบสิก สามารถจะแบ่งขั้นตอนในการสร้างโปรแกรมในวิชวลเบสิกได้เป็น 2 ขั้นตอนหลักได้แก่ การออกแบบหน้าจอโปรแกรม และการเขียนโปรแกรม

1) ออกแบบหน้าจอของโปรแกรม ซึ่งเป็นส่วนที่ทำหน้าที่ติดต่อกับผู้ใช้ เรียกว่า ยูสเซอร์อินเตอร์เฟส (User Interface) ซึ่งในเป็นการออกแบบหน้าจอของโปรแกรมด้วยคอนโทรล คอนโทรลเป็นเครื่องมือในการออกแบบหน้าจอโปรแกรม ซึ่งเครื่องมือต่างๆ เหล่านี้วิชวลเบสิกได้เตรียมไว้ให้ในทูลบ็อกซ์ โดยให้เลือกคอนโทรลที่ตรงกับจุดประสงค์ในการใช้งาน และนำมาวางบนฟอร์มว่างที่ปรากฏอยู่ คอนโทรลต่างๆ ที่ได้ทำการเลือกมาแล้วนำมาวางคอนโทรลที่มีอยู่ในรูปที่ 2.23 จะเป็นเพียงคอนโทรลพื้นฐานในการเขียนโปรแกรม แต่ยังมีคอนโทรลอื่นๆ ที่น่าสนใจอีกมากมาย

การสร้างโปรแกรมในขั้นตอนแรกนั้นจะต้องเลือกคอนโทรลต่างๆ ให้เหมาะสมกับการทำงานของโปรแกรม เพราะว่าเป็นส่วนที่ผู้ใช้งานเห็นและทำงานด้วย เมื่อได้เลือกคอนโทรลจากทูลบ็อกซ์มาวางบนฟอร์ม วิชวลเบสิกจะตั้งชื่อให้กับคอนโทรลโดยอัตโนมัติ

2) การเขียนโปรแกรม ซึ่งในโปรแกรมวิชวลเบสิกเป็นการกำหนดคุณสมบัติของคอนโทรลบนฟอร์มให้เหมาะสม และการเขียนคำสั่งตอบสนองต่อเหตุการณ์

การกำหนดคุณสมบัติ คุณสมบัติ (Properties) คือ ลักษณะต่างๆ ของคอนโทรลที่ถูกนำมาวางบนฟอร์มที่สามารถกำหนดได้ เช่น ข้อความที่ปรากฏบนคอนโทรล รูปแบบตัวอักษรของคอนโทรล เป็นต้น ซึ่งในการกำหนดคุณสมบัติทำได้โดยการกำหนดในหน้าต่างของพรอบเพอร์ตี้ ซึ่งในการกำหนดจะต้องกำหนดให้เหมาะสม และตรงกับการใช้งาน

นอกจากการกำหนดคุณสมบัติแล้ว ควรจะรู้จักกับการเขียนโปรแกรมแบบตอบสนองต่อเหตุการณ์ คือ การใช้คำสั่งกำหนดให้คอนโทรลตอบสนองต่อเหตุการณ์บางอย่างที่เกิดขึ้น (Event) เมื่อโปรแกรมที่สร้างถูกนำมาใช้งาน เช่น ถ้าต้องการให้โปรแกรมเกิดการตอบสนองเมื่อปุ่มคำสั่งถูกคลิกเมาส์ หรือตอบสนองเมื่อค่าในเท็กบ็อกซ์ (Textbox) ถูกเปลี่ยนแปลง สำหรับกฎการตั้งชื่อของคำสั่งที่ตอบสนองต่อเหตุการณ์ในวิชวลเบสิกจะใช้ชื่อคอนโทรลตามด้วยเครื่องหมายขีดเส้นใต้ (_) และชื่อของเหตุการณ์ เช่น ปุ่มคำสั่งชื่อ Command1 ดังนั้นคำสั่งที่ตอบสนองต่อเหตุการณ์คลิก จะมีชื่อเป็น Command1_Click

หากต้องการที่จะทราบรายละเอียดในการเขียนโปรแกรมขั้นสูงขึ้นสามารถค้นหาข้อมูลได้จากในหนังสืออ้างอิงท้ายเล่มได้ ซึ่งมีรายละเอียดต่างๆ

2.4 การประมวลระดับบิตด้วย Bits32.dll

ปัจจุบันได้มีการนำเอาโปรแกรมวิชวลเบสิกมาใช้ในการสร้างโปรแกรมเพื่อควบคุมบอร์ดอิเล็กทรอนิกส์ประเภทชุดทดลองแบบแผ่นพิมพ์เดี่ยว (Single Board) หรือ วงจรควบคุมดาวเทียม โดยการต่อผ่านพอร์ตอนุกรม (Serial Port) หรือพอร์ตขนาน (Parallel Port) กันมากขึ้น ซึ่งมีการประมวลข้อมูลทั้งด้านอินพุต และเอาต์พุตในรูปแบบของบิตเป็นส่วนใหญ่ เนื่องมาจากวิชวลเบสิกในด้านของความสะดวก และความง่ายต่อการใช้งาน แต่อย่างไรก็ตามโปรแกรมเมอร์มักจะประสบปัญหาความล่าช้าของวิชวลเบสิกในการทำงานด้านการประมวลผลระดับบิต (Bit Manipulation) เนื่องจากไม่มีฟังก์ชัน และตัวแปรที่สนับสนุนก็ไม่เหมาะกับการทำงานด้านการประมวลผลระดับบิตโดยตรง ด้วยเหตุผลข้างต้นจึงทำให้วิชวลเบสิกไม่เหมาะกับการสร้างโปรแกรมด้านการประมวลผลบิต

ดังนั้นเพื่อเป็นการเพิ่มขีดความสามารถในด้านการประมวลผลบิตได้อย่างรวดเร็วให้กับวิชวลเบสิก จึงได้มีผู้ทำการพัฒนาไฟล์ไลบารี Bits32.dll ที่สามารถประมวลผลระดับบิตได้อย่างรวดเร็ว ซึ่งประกอบด้วยฟังก์ชันในการกำหนดค่า ลบค่า หรืออ่านค่าบิตที่กำหนด ฟังก์ชันในการเลื่อนบิต (Bit Shift) และหมุนบิต (Bit Rotation) และนอกจากนี้ยังสนับสนุนการแปลงข้อมูลไบนารีให้อยู่ในรูปแบบของสตริงข้อมูลฐานสิบหกได้อีกด้วย ซึ่งสามารถใช้ความสามารถในส่วนนี้เพื่อนำมาสร้างโปรแกรมที่ช่วยในการแสดงข้อมูลที่มีข้อมูลชนิดไบนารี

2.4.1 พื้นฐานการประมวลผลบิต

บิตเป็นตัวเลขฐานข้อมูล 2 (Binary Number) ที่ประกอบด้วยตัวเลข "0" และ "1" หรือสามารถเรียกอีกอย่างว่าสถานะ ปิด (Off) และเปิด (On) ตามลำดับ ซึ่งจะเป็นระบบเลขฐานของระบบคอมพิวเตอร์ และระบบโทรคมนาคมแบบดิจิตอลทั้งหมดในปัจจุบัน โดยที่ขนาดของการจัดเก็บที่สำคัญสามารถแบ่งออกได้เป็นดังนี้

1) ข้อมูลขนาดไบต์ (Byte Data Size) ข้อมูลขนาด 1 ไบต์ จะมีขนาด 8 บิต ซึ่งเป็นมาตรฐานในการจัดเก็บข้อมูลตามรหัสแอสกีในปัจจุบัน สำหรับโปรแกรมวิชวลเบสิก การประกาศข้อมูลชนิดไบต์ (Byte) จะคอมแพททิเบิลกับข้อมูลขนาดไบต์ โดยจะมีค่าตั้งแต่ 0 ถึง 255

2) ข้อมูลขนาดเวิร์ด (Word Data Size) ข้อมูลขนาด 1 เวิร์ด จะมีขนาด 16 บิต ซึ่งเป็นมาตรฐานในการจัดเก็บข้อมูลตามรหัสยูนิโค้ด (Unicode) ในปัจจุบัน สำหรับโปรแกรมวิชวลเบสิก การประกาศข้อมูลชนิดอินทิเจอร์ (Integer) จะคอมแพททิเบิลกับข้อมูลขนาดเวิร์ด แต่จะต่างกันตรงที่ข้อมูลถูกจัดเก็บจะถูกตีความเป็นได้ทั้งเลขจำนวนเต็มบวก และลบมีค่าตั้งแต่ –32,768 ถึง 32,767 แต่ข้อมูลขนาดเวิร์ดในด้านการประมวลผลบิตจะถูกตีความเป็นเลขจำนวนเต็มบวก มีค่าตั้งแต่ 0 ถึง 65,535 เท่านั้น

3) ข้อมูลขนาดดับเบิลเวิร์ด (Double-Word Data Size) ข้อมูลขนาด 1 ดับเบิลเวิร์ด จะมีขนาด 32 บิต เป็นขนาดมาตรฐานในการอ้างอิงรหัส หรือแบ่งเซกเมนต์ของในระบบปฏิบัติการวินโดวส์ 32 บิต เช่น วินโดวส์ 95, วินโดวส์ 98 และวินโดวส์ NT เป็นต้น สำหรับในโปรแกรมวิชวลเบสิกการประกาศข้อมูลชนิดลอง (Long) จะคอมแพททิเบิลกับข้อมูลขนาดดับเบิลเวิร์ด แต่ต่างกันตรงที่ข้อมูลถูกจัดเก็บจะถูกตีความเป็นได้ทั้งเลขจำนวนเต็มบวก และลบมีค่าตั้งแต่ -2,147,483,648 ถึง 2,147,483,647 แต่ข้อมูลขนาดดับเบิลเวิร์ดในด้านการประมวลผลบิตจะถูกตีความเป็นเลขจำนวนเต็มบวก มีค่าตั้งแต่ 0 ถึง 4,294,967,295 เท่านั้น ดังนั้นในการประมวลผลบิตต้องเลือกขนาดตัวแปรที่เหมาะสมกับขนาดของข้อมูลที่ได้จากการประมวลผล เพื่อป้องกันไม่ให้เกิดปัญหาโอเวอร์โฟลว์ หรือค่าที่ได้จากการคำนวณโตกว่าขนาดของตัวแปร

สำหรับในวิชวลเบสิกจะสนับสนุนการกำหนดตัวเลข หรือค่าคงที่ เฉพาะในรูปแบบของระบบตัวเลขฐานสิบ ตัวเลขฐานแปด และตัวเลขฐานสิบหก เท่านั้น โดยที่จะต้องกำหนดตัวอักษร O และตัวอักษร H ต่อท้ายตัวเลขฐานแปด และตัวเลขฐานสิบหก ตามลำดับ เช่น f = 32O จะหมายถึงการกำหนดค่าตัวเลขฐานแปด 32 (26_{10}) ให้กับตัวแปร f และ f = 9EH จะหมายถึงการกำหนดค่าตัวเลขฐานแปด 9E (158_{10}) ให้กับตัวแปร f

2.4.2 เทคนิคเกี่ยวกับการประมวลผล

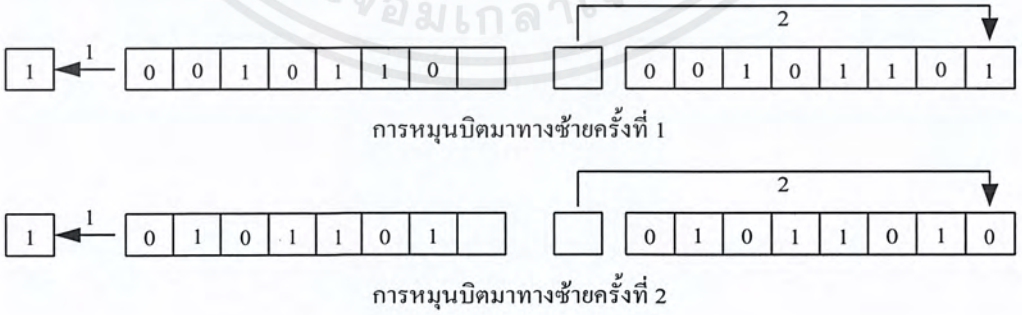
1) การอ่านค่าของบิต (Get Bit) หมายถึง การอ่านสถานะของบิตตำแหน่งที่กำหนด โดยที่บิตแรกสุดของค่าตัวเลขจะหมายถึง บิตลำดับที่ 0 และบิตสุดท้ายของค่าตัวเลขจะหมายถึง บิตลำดับที่ N-1 โดยที่ N หมายถึงขนาดของข้อมูล (8, 16 หรือ 32 บิตเป็นต้น) เช่น ค่าของบิตลำดับที่ 4 และ 5 ของค่าตัวเลขฐานสอง 10010110 จะมีค่าเท่ากับ 1 และ 0 ตามลำดับ

2) การกำหนดค่าบิต (Set Bit) หมายถึง การกำหนดให้บิตใดบิตหนึ่ง หรือ หลาย ๆ บิตมีค่าเท่ากับ 1 เช่น การกำหนดค่าลำดับที่ 3 ของค่าตัวเลขฐานสอง 10010110 จะเท่ากับ 10011110

3) การรีเซตค่าบิต (Reset Bit) หมายถึง การกำหนดให้บิตใดบิตหนึ่ง หรือหลายบิตมีค่าเท่ากับ 0 เช่น การกำหนดค่าบิตลำดับที่ของค่าตัวเลขฐานสอง 10010110 จะเท่ากับ 10010010

4) การกลับค่าบิต (Toggle Bit) หมายถึง การกำหนดให้บิตใดบิตหนึ่ง หรือ หลาย ๆ บิตมีค่าตรงข้ามสถานะเดิม เช่น การกลับค่าบิตลำดับที่ 2 ของค่าตัวเลขฐานสอง 10010110 และ 10010010 และกลับค่าบิตตามลำดับที่ 2 ของค่าตัวเลขฐานสอง 10010010 จะเท่ากับ 10010110 ซึ่งมีค่าเหมือนเดิมก่อนการกลับค่าบิต และสามารถเรียกสถานะปฏิบัติการดังกล่าวว่า การปฏิบัติการตรรกะ Xor (Exclusive OR)

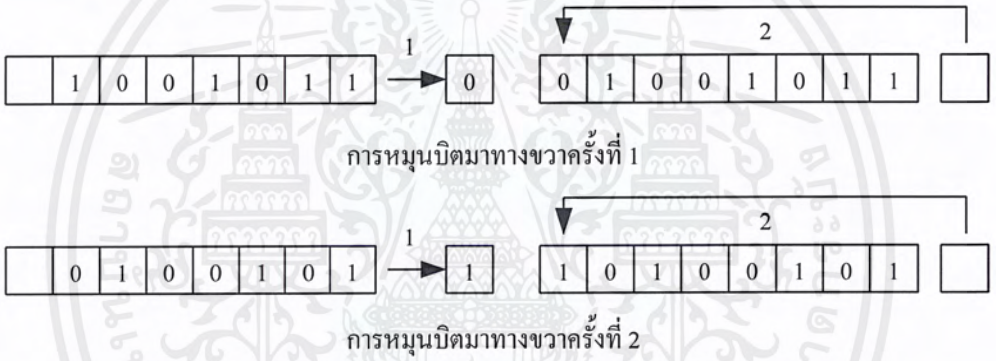
5) การหมุนบิต (Rotate Bit) ในการภาษาแอสเซมบลี การหมุนบิตซ้าย และบิตขวาจะเทียบเท่ากับคำสั่ง ROL และ ROR ตามลำดับ การหมุนบิตสามารถแบ่งออกได้ 2 ลักษณะดังนี้ การหมุนบิตทางซ้าย (Rotate Bit Left) หมายถึง การเลื่อนทุกๆ บิตของสายข้อมูลบิตมาทางซ้ายเท่ากับจำนวนบิตที่กำหนด และนำบิตที่ถูกเลื่อนมาทางซ้ายสุดดังกล่าวมาแทนที่บิตทางขวามือที่ถูกเลื่อน เช่น ต้องการหมุนบิตของตัวเลขฐานสอง 10010110 มาทางซ้ายจำนวน 2 บิต จะมีเป็นดังรูปที่ 2.24



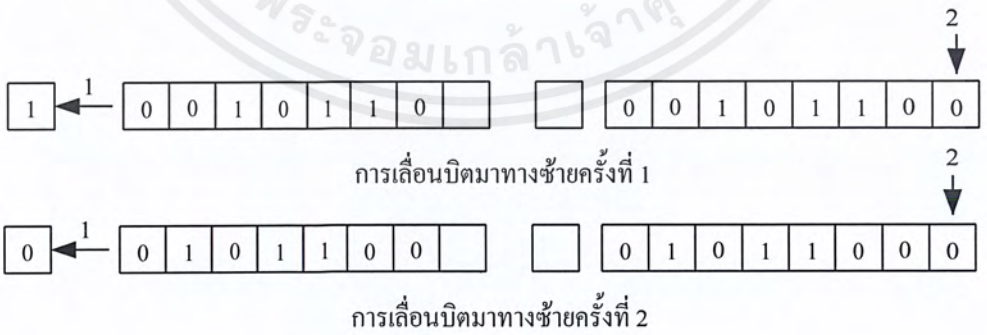
รูปที่ 2.24 การหมุนบิตทางซ้าย

การหมุนบิตมาทางขวา (Rotate Bit Right) หมายถึง การเลื่อนทุกๆ บิตสายข้อมูลบิตมาทางขวาเท่ากับจำนวนบิตที่กำหนด และนำบิตที่ถูกเลื่อนดังกล่าวมาแทนที่บิตทางซ้ายมือที่ถูกเลื่อน เช่นต้องการหมุนบิตของค่าตัวเลขฐานสอง 10010110 มาทางขวาจำนวน 2 บิต จะเป็นดังรูปที่ 2.25

6) การเลื่อนบิต (Shift Bit) ในการภาษาแอสเซมบลีการเลื่อนบิตซ้าย และบิตขวาจะเทียบเท่ากับคำสั่ง SHL และ SHR ตามลำดับ การหมุนบิตสามารถแบ่งออกได้ 2 ลักษณะดังนี้ การเลื่อนบิตทางซ้าย (Shift Bit Left) หมายถึง การเลื่อนทุก ๆ บิตของสายข้อมูลบิตมาทางซ้ายเท่ากับจำนวนบิตที่กำหนด และแทนที่บิตทางขวามือที่ถูกเลื่อนด้วย 0 เช่น ต้องการเลื่อนบิตของตัวเลขฐานสอง 10010110 มาทางซ้ายจำนวน 2 บิต จะมีเป็นดังรูปที่ 2.26



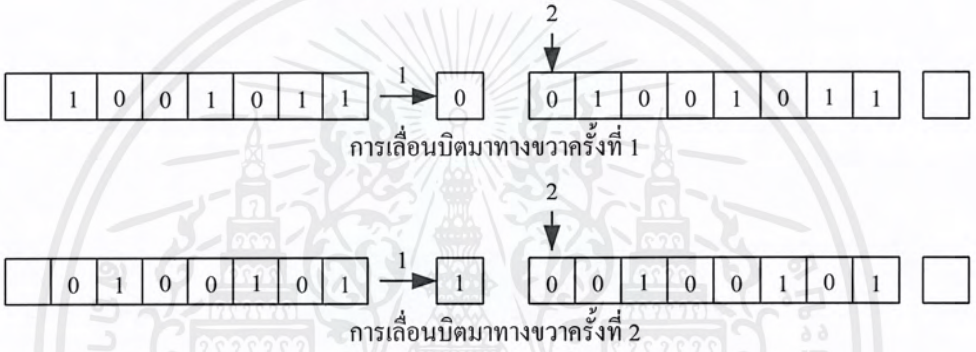
รูปที่ 2.25 การหมุนบิตทางขวา



รูปที่ 2.26 การเลื่อนบิตทางซ้าย

การเลื่อนบิตทางขวา (Shift Bit Right) หมายถึง การเลื่อนทุกๆ บิตของสายข้อมูลบิตมาทางขวาเท่ากับจำนวนบิตที่กำหนด และแทนที่บิตทางซ้ายมือที่ถูกเลื่อนด้วย "0" เช่น ต้องการเลื่อนบิตของตัวเลขฐานสอง 10010110 มาทางขวาจำนวน 2 บิต จะมีเป็นดังรูปที่ 2.27

สำหรับในทางพีชคณิตบูลีนการเลื่อนบิตมาทางขวา 1 บิต จะเท่ากับการหารเลขจำนวนด้วยสอง และในทางกลับกันการเลื่อนบิตมาทางซ้าย 1 บิตจะเท่ากับการคูณเลขด้วยจำนวนดังกล่าวด้วย 2 เช่น กันการเลื่อนบิตของตัวเลข 00000110 (6_{10}) มาทางขวา 1 บิต จะเท่ากับ 00000011 (3_{10}) เป็นต้น



รูปที่ 2.27 การเลื่อนบิตทางขวา

7) การกลับลำดับตำแหน่งบิต (Swap Bit) หมายถึง การสลับตำแหน่งของสายในข้อมูล เช่น สมมุติว่าสายประกอบด้วย n บิต ก็จะเป็นการสลับตำแหน่งระหว่างบิตลำดับที่ n กับ 0 และ n-1 กับ 1 และ n-2 กับ 2 เรื่อยๆ ไปจนกว่าจะหมดทุกตำแหน่งของบิต เช่น การกลับลำดับของค่าตัวเลขฐานสอง 10010110 ก็จะเป็นดังรูปที่ 2.28

ตำแหน่งลำดับของบิต	7	6	5	4	3	2	1	0
ค่าตัวเลขฐานสอง ก่อนการกลับลำดับบิต	1	0	0	1	0	1	1	0
	↓	↓	↓	↓	↓	↓	↓	↓
ค่าตัวเลขฐานสอง หลังการกลับลำดับบิต	0	1	1	0	1	0	0	1

รูปที่ 2.28 การกลับลำดับตำแหน่งบิต

จากที่กล่าวมาข้างต้นทั้งหมดเป็นพื้นฐานข้อมูลที่เกี่ยวกับการประมวลผลบิตที่จำเป็นสำหรับการเขียนโปรแกรมที่ต้องเขียนติดต่อกับบอร์ดอิเล็กทรอนิกส์ หรือโปรแกรมประเภทบีบอัดข้อมูล

2.4.3 ไฟล์ไลบรารี Bits32.dll

ไฟล์ไลบารี Bits32.dll เป็นไฟล์ที่ถูกพัฒนาขึ้นมาสำหรับใช้งานร่วมกับวิชวลเบสิกรุ่น 5.0 และรุ่น 6.0 โดยจะประกอบด้วยกลุ่มฟังก์ชันที่เกี่ยวข้องกับประมวลผลบิตของข้อมูลขนาด 1 ไบต์ (สำหรับวิชวลเบสิกจะเทียบเท่ากับ Byte) ข้อมูลขนาด 2ไบต์ (Word สำหรับวิชวลเบสิกจะเทียบ-เท่ากับ Integer) และข้อมูลขนาด 4 ไบต์ (Double-Word สำหรับวิชวลเบสิกจะเทียบเท่ากับ Long) และกลุ่มฟังก์ชันที่เกี่ยวข้องกับการเปลี่ยนแปลงข้อมูลชนิดตัวเลข สำหรับฟังก์ชันการเปลี่ยนแปลงข้อมูลนั้น จะเป็นการเปลี่ยนแปลงในรูปของตัวเลขเป็นข้อมูลสตริงในรูปของเลขฐานสิบหก หรือจะเป็นการเปลี่ยนแปลงข้อมูลเป็นเลขฐานสิบ หรือการแปลงให้อยู่ในรูปของตัวเลขฐานสอง เป็นต้น สำหรับประโยคการประกาศฟังก์ชัน และค่าคงที่ทั้งหมดของไฟล์ไลบารี Bits32.dll จะถูกจัดเก็บไว้ในไฟล์โมดูล Bits32.bas และก่อนที่จะทำการใช้งานต้องทำการคัดลอกไฟล์ไลบรารี Bits32.dll ลงในไดเรคทรอรีชื่อ System ของวินโดวส์ (เช่น C:\Windows\System) ก่อน โดยที่ค่าคงที่ และฟังก์ชันทั้งหมดของไฟล์ไลบรารี Bits32.dll แสดงได้ดังในตารางที่ 2.19

ตารางที่ 2.18 ฟังก์ชันของไฟล์ไลบรารี Bits32.dll

ชื่อฟังก์ชัน	หน้าที่
1. รายงานเวอร์ชัน และสงวนสิทธิ์ ประกอบด้วย	
VbrcBitGetCopyright	รายงานชื่อผู้เขียน และการสงวนสิทธิ์
VbrcBitGetVertion	รายงานหมายเลขเวอร์ชันของไฟล์ไลบรารี Bits32.dll
2. ประมวลผลบิตชนิดข้อมูลตัวเลขขนาด 1 ไบต์ ประกอบด้วย	
VbrcBinStrB	แปลงข้อมูลตัวเลขให้เป็นข้อมูลสตริงของเลขไบนารี
VbrcGetBitB	อ่านค่าของบิตจากตำแหน่งของบิตข้อมูลตัวเลขที่กำหนด
VbrcResetBitB	กำหนดให้บิตมีค่าเท่ากับ 0 ตรงตำแหน่งของบิตที่กำหนด
VbrcSetBitB	กำหนดให้บิตมีค่า 1 หรือ 0 ตรงตำแหน่งของบิตที่กำหนด
VbrcROLB	หมุนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางซ้าย
VbrcRORB	หมุนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางขวา

ตารางที่ 2.18 (ต่อ) ฟังก์ชันของไฟล์ไลบรารี Bits32.dll

ชื่อฟังก์ชัน	หน้าที่
VbrcSHLB	เลื่อนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางซ้าย
VbrcSHRB	เลื่อนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางขวา
VbrcSwapBitB	กลับตำแหน่งของบิตของข้อมูลชนิดตัวเลขที่กำหนด
VbrcToggleBitB	กลับค่าของบิตตรงลำดับตำแหน่งของบิตในข้อมูลชนิดตัวเลขที่กำหนด
3. ประมวลผลบิตชนิดข้อมูลตัวเลขขนาด 2 ไบต์ ประกอบด้วย	
VbrcGetBitBW	อ่านค่าของบิตจากตำแหน่งของบิตข้อมูลตัวเลขที่กำหนด
VbrcResetBitBW	กำหนดให้บิตมีค่าเท่ากับ 0 ตรงตำแหน่งของบิตที่กำหนด
VbrcSetBitBW	กำหนดให้บิตมีค่า 1 หรือ 0 ตรงตำแหน่งของบิตที่กำหนด
VbrcROLBW	หมุนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางซ้าย
VbrcRORBW	หมุนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางขวา
VbrcSHLBW	เลื่อนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางซ้าย
VbrcSHRBW	เลื่อนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางขวา
VbrcSwapBitBW	กลับตำแหน่งของบิตของข้อมูลชนิดตัวเลขที่กำหนด
VbrcToggleBitBW	กลับค่าของบิตตรงตำแหน่งของข้อมูลชนิดตัวเลขที่กำหนด
4. ประมวลผลบิตชนิดข้อมูลตัวเลขขนาด 4 ไบต์ ประกอบด้วย	
VbrcBinStrD	แปลงข้อมูลตัวเลขให้เป็นข้อมูลสตริงของเลขไบนารี
VbrcGetBitD	อ่านค่าของบิตจากตำแหน่งของบิตข้อมูลตัวเลขที่กำหนด
VbrcResetBitD	กำหนดให้บิตมีค่าเท่ากับ 0 ตรงตำแหน่งของบิตที่กำหนด
VbrcSetBitD	กำหนดให้บิตมีค่า 1 หรือ 0 ตรงตำแหน่งของบิตที่กำหนด
VbrcROLD	หมุนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางซ้าย
VbrcRORD	หมุนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางขวา
VbrcSHLD	เลื่อนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางซ้าย
VbrcSHRD	เลื่อนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางขวา
VbrcSwapBitD	กลับตำแหน่งของบิตของข้อมูลชนิดตัวเลขที่กำหนด
VbrcToggleBitD	กลับค่าของบิตตรงตำแหน่งของบิตในข้อมูลชนิดตัวเลขที่กำหนด
5. แปลงข้อมูลชนิดตัวเลขกับข้อมูลชนิดสตริง	
VbrcCVBYT	แปลงข้อมูลชนิดสตริงเป็นชนิดตัวเลข

ตารางที่ 2.18 (ต่อ) ฟังก์ชันของไฟล์ไลบรารี Bits32.dll

ชื่อฟังก์ชัน	หน้าที่
VbrcCVD	แปลงข้อมูลชนิดสตริงเป็นชนิดตัวเลข
VbrcCVI	แปลงข้อมูลชนิดสตริงเป็นชนิดตัวเลข
VbrcCVL	แปลงข้อมูลชนิดสตริงเป็นชนิดตัวเลข
VbrcCVS	แปลงข้อมูลชนิดสตริงเป็นชนิดตัวเลข
VbrcMKBYT	แปลงข้อมูลชนิดตัวเลขให้ชนิดเป็นสตริง
VbrcMKD	แปลงข้อมูลชนิดตัวเลขให้ชนิดเป็นสตริง
VbrcMKI	แปลงข้อมูลชนิดตัวเลขให้ชนิดเป็นสตริง
VbrcMKL	แปลงข้อมูลชนิดตัวเลขให้ชนิดเป็นสตริง
VbrcMKS	แปลงข้อมูลชนิดตัวเลขให้ชนิดเป็นสตริง
6. การแยกหรือรวมไบต์ของข้อมูลชนิดตัวเลข	
VbrcB2W	สร้างตัวเลขขนาด 2 ไบต์ จากข้อมูลตัวเลข 1 ไบต์ 2 ตัว
VbrcW2B	แยกตัวเลขขนาด 2 ไบต์ ออกเป็นข้อมูลชนิดตัวเลข 1 ไบต์ 2 ตัว
VbrcW2D	สร้างตัวเลขขนาด 4 ไบต์ ออกเป็นข้อมูลชนิดตัวเลข 2 ไบต์ 2 ตัว
VbrcD2W	แยกตัวเลขขนาด 4 ไบต์ ออกเป็นข้อมูลชนิดตัวเลข 2 ไบต์ 2 ตัว
VbrcRGB	แปลงข้อมูลตัวเลข 1 ไบต์ของแม่สีแดง สีเขียว และสีน้ำเงินให้เป็นข้อมูลตัวเลขขนาด 4 ไบต์
7. แปลงข้อมูลไบนารีเป็นสตริงของเลขฐานสิบหก	
VbrcBin2Hex	แปลงข้อมูลชนิดสตริงไบนารีเป็นสตริงของตัวเลขฐานสิบหก
VbrcUnprintableCharFilter	แทนที่แอสกีที่ไม่สามารถแสดงผลได้ เช่น แอสกี 10 หรือ 13

2.5 โปรแกรมคอมไพเลอร์ไมโครคอนโทรลเลอร์ 68HC11

โปรแกรมคอมไพเลอร์ AS11 เป็นโปรแกรมที่ทำงานอยู่บนดอส โดยจะทำงานโดยการแปลงไฟล์ที่มีนามสกุล ASM ที่เขียนบนโปรแกรมอิดิเตอร์ทั่วไปแล้วจัดเก็บไฟล์เป็นนามสกุล ASM แล้วไปทำการคอมไพล์เป็นไฟล์นามสกุล S19 และสร้างไฟล์นามสกุล LST โครงสร้างของไฟล์สกุล S19 รูปแบบของข้อมูลแบบนี้มีลักษณะดังแสดงตารางที่ 2.20

ตารางที่ 2.19 รูปแบบของไฟล์นามสกุล S19

ชื่อสัญลักษณ์	จำนวนตัวอักษร	รายละเอียด
Record Mark	1	ในแต่ละบรรทัดของข้อมูลที่ต้องการบันทึกต้องเริ่มด้วย S
Record length	2	จำนวนข้อมูลที่ต้องการบันทึก
Address Field	2	แสดงแอดเดรสที่เก็บข้อมูลในไฟล์แรก ปกติมีค่า 00
Record Type	2	ชนิดของข้อมูลที่ทำการบันทึก โดยถ้าเป็นข้อมูล 00 หมายถึง ข้อมูลของโปรแกรม 01 หมายถึง จบแฟ้มข้อมูล 02 หมายถึง แอดเดรสเพิ่มเติม 03 หมายถึง แอดเดรสเริ่มต้น
Datafield	เปลี่ยนแปลง	จะขึ้นอยู่กับรหัสชนิดของข้อมูลที่ทำการบันทึก 00 หมายถึง จำนวนข้อมูลที่ต้องการโปรแกรม 01 หมายถึง ไม่ใช้งาน (ว่าง) 02 หมายถึง เซกเมนต์สำหรับแอดเดรสที่มากกว่า 64 กิโลไบต์ ข้อมูลจะถูกเริ่มต้นเก็บที่เซกเมนต์ x 10h บวกกับค่าของพื้นที่แอดเดรส ตัวอย่าง แอดเดรสเพิ่มเติมมีค่า F800h พื้นที่ของแอดเดรสเท่ากับ 1000h ดังนั้นข้อมูลจะไปเริ่มต้นบันทึกที่ตำแหน่ง F900h
Checksum	2	การคำนวณค่า Checksum จะรวมค่าของข้อมูลทั้งหมดที่ทำการบันทึก จากนั้นทำทูคอมพลีเมนต์ค่า Check Sum คือ ไบต์ค่าของข้อมูลสุดท้ายหลัง ทำทูคอมพลีเมนต์

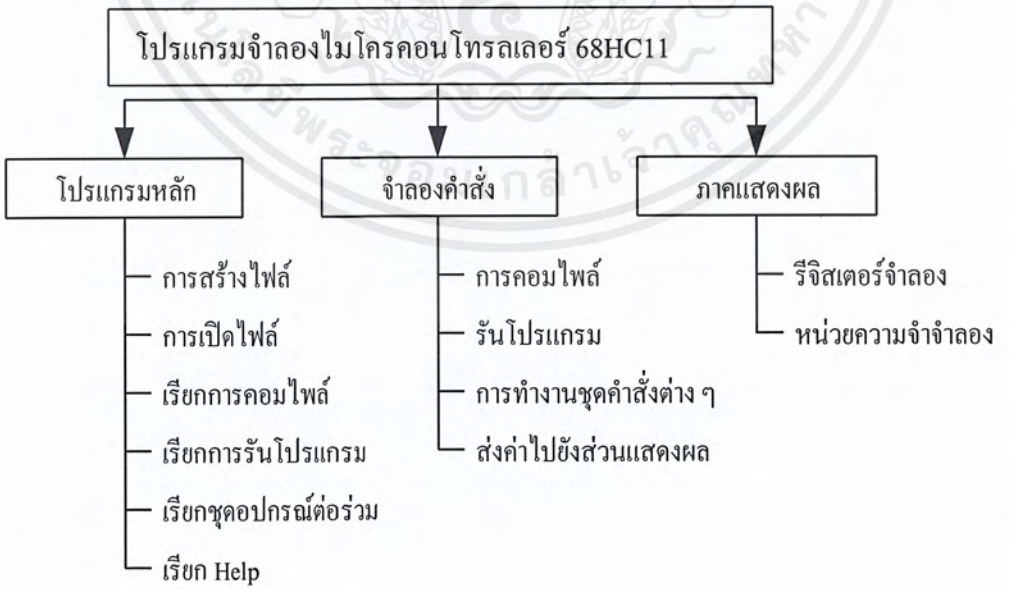
บทที่ 3
การออกแบบ การสร้าง และการทำงาน

3.1 กล่าวนำ

การออกแบบและการสร้างโปรแกรมได้แบ่งการทำงานออกเป็นส่วนๆ เพื่อให้ง่ายต่อการเขียนโปรแกรมและพัฒนาโปรแกรมต่อไปในอนาคต ซึ่งได้แบ่งโปรแกรมออกเป็น 2 ส่วน คือ ส่วนของการจำลองการทำงานภายในของไมโครคอนโทรลเลอร์ 68HC11 และส่วนของการแสดงผลติดต่อกับผู้ใช้งาน ซึ่งในการเขียนโปรแกรมของการติดต่อกับผู้ใช้งานได้ใช้โปรแกรมวิชวลเบสิก รวมทั้งในส่วนของการเขียนโปรแกรมจำลองคำสั่งต่างๆ ด้วย ซึ่งในรายละเอียดของการออกแบบและการสร้างอธิบายได้ดังต่อไปนี้

3.2 โครงสร้างโปรแกรม

โครงสร้างหลักของโปรแกรมแบ่งออกเป็น 3 ส่วน คือ ส่วนของโปรแกรมหลัก ส่วนของการจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11 และในส่วนของภาคแสดงผล ซึ่งโครงสร้างของโปรแกรมจะแสดงได้ดังรูปที่ 3.1

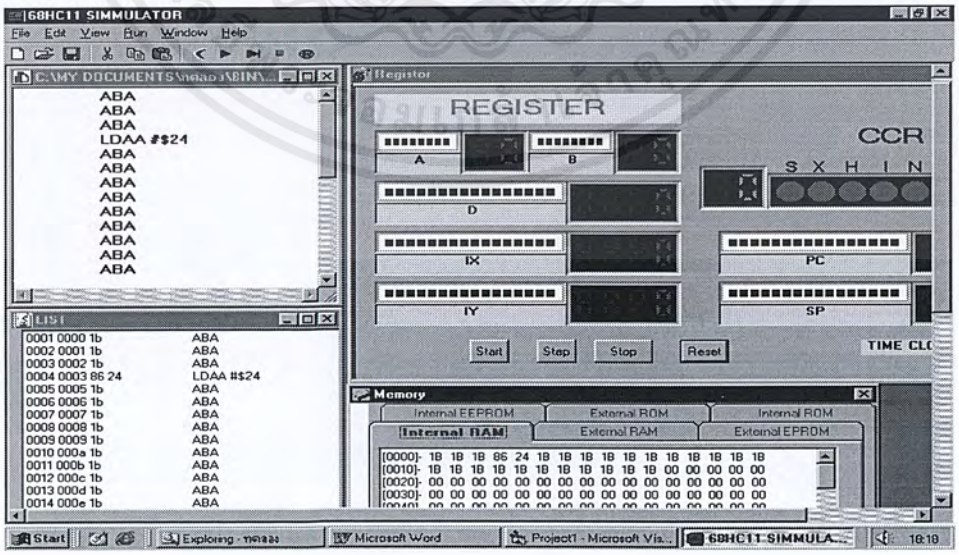


รูปที่ 3.1 โครงสร้างของโปรแกรม

จากรูปที่ 3.1 โครงสร้างของโปรแกรมแบ่งออกเป็น 3 ส่วน คือ ส่วนของโปรแกรมหลักจะเป็นส่วนที่ใช้ติดต่อกับผู้ใช้โดยตรง จะทำหน้าที่เกี่ยวกับการจัดการไฟล์ การเขียนเท็กซ์ไฟล์ ซึ่งจะทำหน้าที่เสมือนโปรแกรมอิดิเตอร์ทั่วไปนั่นเอง คือ มีการสร้างไฟล์ใหม่ การเปิดไฟล์ ซึ่งเมื่อมีการสร้างไฟล์ใหม่ ก็จะสามารถที่เรียกการคอมไพล์ให้อยู่ในรูปของโปรแกรมของไมโครคอนโทรลเลอร์ 68HC11 และเรียกการจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11 โดยการรันโปรแกรม ซึ่งในส่วนของโปรแกรมหลักนี้จะเป็นตัวเชื่อมต่อระหว่างการจำลองคำสั่งและภาคแสดงผล ในส่วนของการจำลองคำสั่งจะทำหน้าที่คอมไพล์ไฟล์เท็กซ์ที่ได้จากโปรแกรมหลักให้อยู่ในรูปแบบของการที่จะนำไปใช้จำลองการทำงานต่อไป และจะจำลองการทำงานของชุดคำสั่งต่างๆ ที่มีอยู่ในไมโครคอนโทรลเลอร์ 68HC11 เหมือนกับทำงานด้วยไมโครคอนโทรลเลอร์ 68HC11 จริงๆ เมื่อรันโปรแกรม แล้วจะส่งผลไปยังภาคการแสดงผล ส่วนของภาคแสดงผลจะทำหน้าที่ในการจำลองหน่วยความจำ รีจิสเตอร์ และอุปกรณ์ต่อร่วมกับไมโครคอนโทรลเลอร์ 68HC11 ซึ่งในโปรแกรมนี้จะมีส่วนของภาคแสดงผลแอลอีดี ภาคแสดงผลแอลซีดี ภาคแสดงผลแบบเจ็ดส่วน และการทำงานของดีซีมอเตอร์ และสเต็ปปิ้งมอเตอร์

3.3 การออกแบบโปรแกรมหลัก

ส่วนของโปรแกรมหลักเป็นส่วนที่ติดต่อกับผู้ใช้โดยตรง ซึ่งจะทำการออกแบบให้มีความสะดวกต่อการใช้งาน ส่วนของโปรแกรมหลักแสดงดังรูปที่ 3.2



รูปที่ 3.2 หน้าต่างของโปรแกรมหลัก

จากรูปที่ 3.2 ส่วนของโปรแกรมหลักนั้นมีการทำงานดังในแผนผังโครงสร้างการทำงานในรูปที่ 3.1 ส่วนต่างๆ ของโปรแกรมจะมีหน้าที่และการทำงานที่แตกต่างกัน โดยโปรแกรมหลักได้แบ่งส่วนของการแสดงผลในการติดต่อกับผู้ใช้โปแกรมหลักออกเป็น 4 ส่วนคือ

1) เมนูบาร์และทูลบาร์

2) หน้าต่างรีจิสเตอร์

3) หน้าต่างหน่วยความจำ

4) หน้าต่างอิดิเตอร์

3.3.1 ส่วนของเมนูบาร์ และทูลบาร์ของโปรแกรม

ส่วนของเมนูบาร์ และทูลบาร์เป็นส่วนที่ทำให้ผู้ใช้สามารถใช้งานได้อย่างสะดวกสบาย โดยการคลิกปุ่ม หรือเมนู โดยส่วนประกอบของเมนูบาร์ และทูลบาร์แสดงได้ดังรูปที่ 3.3 และรูปที่ 3.4 ตามลำดับ

รูปที่ 3.3 เมนูบาร์ของโปรแกรมหลัก

รูปที่ 3.4 ทูลบาร์ของโปรแกรมหลัก

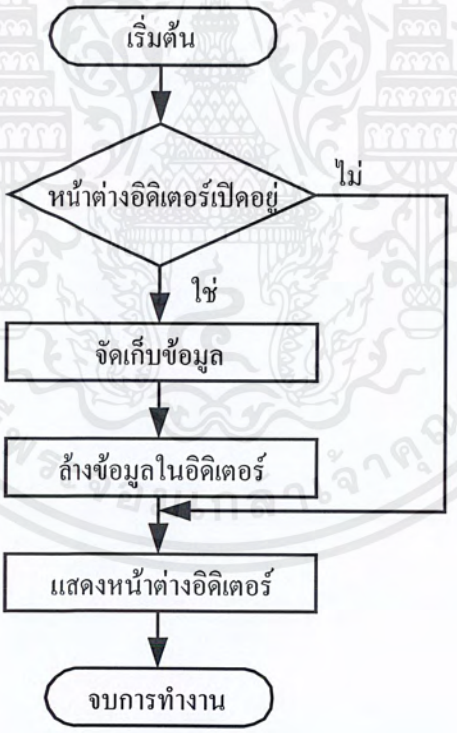
จากรูปที่ 3.3 เมนูบาร์จะประกอบไปด้วยเมนูหลักทั้งหมด 6 เมนูหลัก โดยแต่ละเมนูหลักจะมีเมนูย่อยอยู่ด้วย โดยจะแบ่งตามประเภทของคำสั่งของเมนูย่อยว่ามีการทำงานเป็นอย่างไร ซึ่งจะเป็นการง่ายและสะดวกต่อการใช้งาน ซึ่งลักษณะของเมนูบาร์นั้นจะเป็นลักษณะเหมือนกับเมนูของโปรแกรมโดยทั่วไป ประกอบไปด้วยส่วนเมนูไฟล์จะจัดการเกี่ยวกับการเปิดและสร้างไฟล์ใหม่ และการจบการทำงาน ส่วนของเมนูอิดิเตอร์จะเป็นการจัดการเกี่ยวกับข้อความในเท็กบอกซ์เป็นต้น ในส่วนของเมนูบาร์ได้แบ่งเป็นเมนูหลัก และเมนูย่อย แต่ละเมนูนั้นทำหน้าที่ และเรียกใช้งานที่แตกต่างกัน ดังแสดงในตารางที่ 3.1

ตารางที่ 3.1 ส่วนประกอบของเมนูบาร์

ชื่อเมนูหลัก	หน้าที่
File	จัดการเกี่ยวกับการจัดการไฟล์
New	เปิดหน้าต่างของอิดิเตอร์เพื่อเริ่มการสร้างใหม่
Open	เปิดไฟล์ที่มีการจัดเก็บไว้แล้ว
Save	จัดเก็บไฟล์ที่ทำการสร้างขึ้นมา
Save As	จัดเก็บไฟล์ที่สร้างขึ้นมาโดยจะจัดเก็บให้มีการเปลี่ยนชื่อเป็นชื่ออื่น ๆ
Exit	จบการทำงานของโปรแกรม
Edit	จัดการเกี่ยวกับการจัดการข้อมูล คัดลอก ตัด วาง และค้นหา
Cut	ตัดข้อความที่ต้องการ
Copy	ทำการคัดลอกสำเนา
Paste	วางข้อความลงในตำแหน่งที่กำหนด
Delete	ลบข้อความที่เลือก
Find	ค้นหาข้อความ
Replace	วางข้อความหนึ่งแทนข้อความที่เลือกไว้
View	แสดงหน้าต่างของรีจิสเตอร์ หน้าต่างของหน่วยความจำ
Code window	แสดงหน้าต่างของอิดิเตอร์
List window	แสดงหน้าต่างผลที่ได้จากการคอมไพล์
Register window	แสดงหน้าต่างของรีจิสเตอร์
Memory window	แสดงหน้าต่างของหน่วยความจำ
I/O Toolbox	แสดงหน้าต่างของทูลบ็อกซ์
Run	เกี่ยวการคอมไพล์ การรันโปรแกรม
Assembler	ทำการแปลงไฟล์นามสกุล ASM เป็นไฟล์ในรูปแบบไฟล์ OBJ
Start	รันคำสั่ง
Break	หยุดรันคำสั่งชั่วคราว
Stop	ออกจากรันคำสั่ง
Window	จัดรูปแบบของหน้าต่างที่แสดงในโปรแกรม
Help	แสดงเกี่ยวกับโปรแกรม และคู่มือโปรแกรม

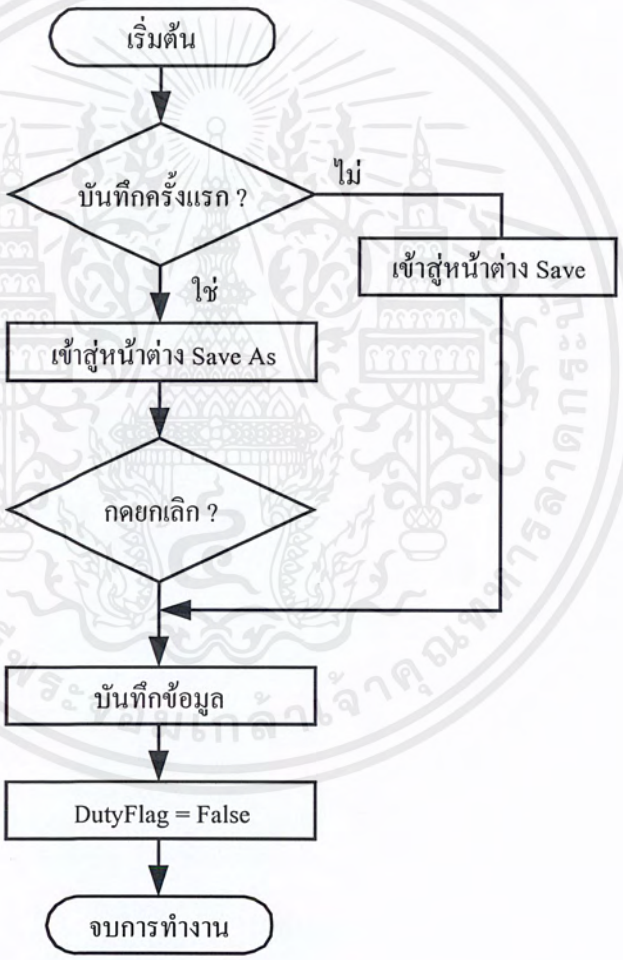
โดยที่ส่วนของเมนูย่อยนั้นได้มีการออกแบบการทำงานในแต่ละส่วนของคำสั่งซึ่งมีการทำงานที่สำคัญดังต่อไปนี้

1) คำสั่ง New เป็นคำสั่งที่ใช้ในการสร้างไฟล์ใหม่ โดยจะมีการตรวจสอบว่าในขณะนี้ ได้มีการเปิดหน้าต่างของการเขียนโปรแกรมหรือไม่ ถ้ามีการสร้างไฟล์หรือเปิดอยู่ ก็จะทำการจัดเก็บข้อมูล ก่อนที่จะทำการเปิดหน้าต่างอิดิเตอร์ใหม่ ขั้นตอนในการเลือกคำสั่ง New เลือกได้โดยการคลิกไปที่เมนูบาร์ เลือกเมนู File แล้วคลิกเลือกลงมาในส่วนของเมนูย่อย เลือก New ขั้นตอนการทำงานแสดงดังในรูปที่ 3.4 โดยเริ่มจากการตรวจสอบว่าในขณะนั้น ได้มีการเปิดไฟล์ค้างอยู่หรือไม่ ถ้าไม่มีการเปิดไฟล์ค้างอยู่ก็จะทำการสร้างไฟล์ใหม่ขึ้นมา แต่ถ้าพบว่ามีการเปิดไฟล์ค้างอยู่แล้วก็จะทำการตรวจสอบว่าได้มีการจัดเก็บไฟล์ที่เปิดอยู่แล้วหรือไม่หากไม่มีก็จะให้มีการบันทึกข้อมูล ถ้ามีการบันทึกข้อมูลไปแล้วก็จะทำการเปิดหน้าต่างอิดิเตอร์เพื่อสร้างไฟล์ขึ้นมาใหม่ โดยจะลบข้อมูลที่มีอยู่ในเท็กบ็อกซ์ออกไป



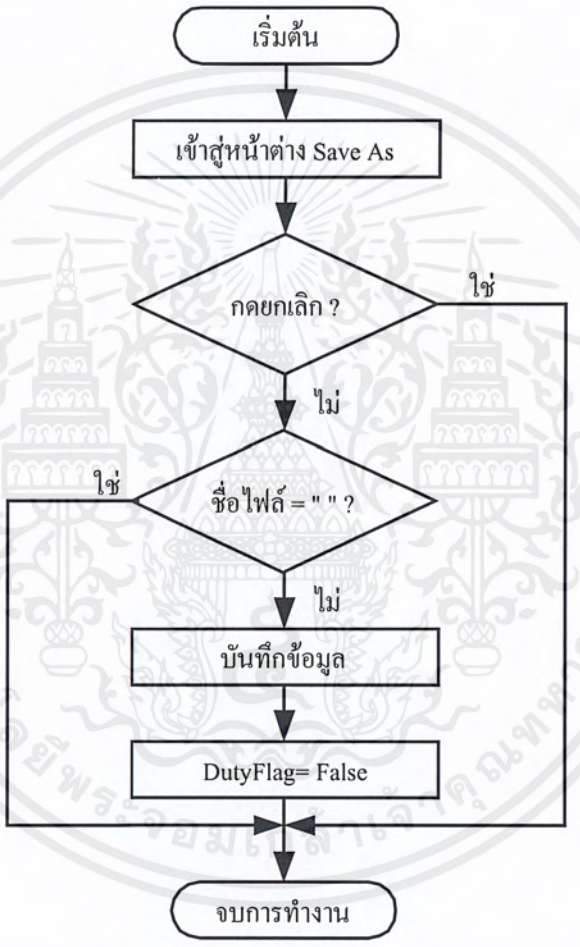
รูปที่ 3.5 แผนผังการทำงานของคำสั่ง New

2) คำสั่ง Save จะมีการตรวจสอบว่าเคยมีการบันทึกข้อมูลหรือไม่ หากยังไม่มีการบันทึกก็จะเข้าไปสู่ การ Save As ซึ่งเป็นการบันทึกไฟล์ครั้งแรก แต่ถ้าหากมีการบันทึกมาแล้วก็จะทำการบันทึกไฟล์ทับข้อมูลเดิม คำสั่ง Save จะทำในกรณีที่มีการเปิดไฟล์ที่มีอยู่แล้วมาทำการแก้ไข ซึ่งถ้าหากไม่มีการแก้ไขก็จะทำการบันทึกไฟล์ข้อมูลเดิมลงในไฟล์เดิมอีกครั้ง แต่หากพบว่าเป็นการเปิดอิดิเตอร์ขึ้นมาเพื่อทำการสร้างไฟล์ใหม่นั้น โปรแกรมจะเตือนให้มีการบันทึกข้อมูลก่อน การเขียนโปรแกรมในส่วนนี้จะเป็นการเรียกใช้คอมมอนไดอะลอกของวิชวลเบสิกเข้ามาเป็นตัวจัดการ หากพบว่าเป็นการบันทึกไฟล์เป็นครั้งแรก ซึ่งกระบวนการดังกล่าวแสดงได้ดังรูปที่ 3.5



รูปที่ 3.6 แผนผังการทำงานของคำสั่ง Save

3) คำสั่ง Save As จะมีขั้นตอนในการทำงานเหมือนกับการบันทึกโดยใช้คำสั่ง Save ซึ่งขั้นตอนในการทำงานจะเรียกหน้าต่าง Save As ขึ้นมา แล้วจะตรวจสอบว่าผู้ใช้ทำการยกเลิกการบันทึกหรือว่าชื่อไฟล์ที่ใส่นั้นว่าง หรือไม่ได้ใส่ชื่อที่ต้องการบันทึก จะไม่ทำการบันทึกให้ หากตรวจสอบว่าไม่ได้ยกเลิก หรือ ใส่ชื่อไฟล์ที่ต้องการบันทึกแล้วก็จะทำการบันทึกข้อมูล แล้วกำหนดค่าให้ตัวแปร Dutyflag มีค่าเป็น False ซึ่งขั้นตอนการทำงานดังแสดงในรูปที่ 3.7



รูปที่ 3.7 แผนผังการทำงานของคำสั่ง Save As

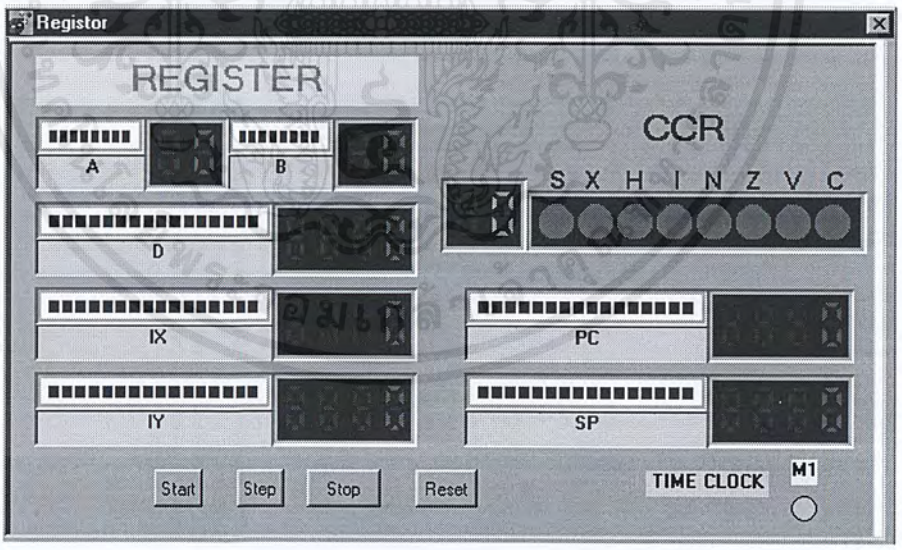
4) คำสั่ง Assembler เรียกส่วนของการทำการแอสเซมเบลอร์ ซึ่งอยู่ในส่วนของการจำลองชุดคำสั่ง โดยคำสั่งนี้จะกระทำได้เมื่อมีการสร้างไฟล์เท็กซ์นามสกุล Asm ที่เป็นโค้ดของการเขียนโปรแกรมไมโครคอนโทรลเลอร์ 68HC11 เรียบร้อยแล้ว ซึ่งถ้าหากว่าไม่มีการสร้างไฟล์นามสกุล

Asm ก็จะไม่สามารถที่จะทำการแอสเซมเบลอร์ได้ ในคำสั่งนี้จะออกแบบให้มีการทำงานโดยไปเรียกโพรซีเยอร์ของการแอสเซมเบลอร์ แล้วส่งการทำงานต่อไปให้ยังส่วนของการจำลองคำสั่ง

5) คำสั่ง Start เป็นส่วนเริ่มต้นการทำงานในส่วนของการจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11 ซึ่งคำสั่งนี้จะไปเรียกส่วนของการจำลองชุดคำสั่งให้เริ่มทำงาน คำสั่งนี้จะใช้ได้ก็ต่อ เมื่อโปรแกรมคำสั่งผ่านการแอสเซมเบลอร์เรียบร้อยแล้ว หากยังไม่แอสเซมเบลอร์จะยังสามารถเลือกใช้ได้ เนื่องจากได้ออกแบบไม่ให้แอกทีฟ คำสั่งนี้จะเรียกโพรซีเยอร์ย่อยของการรันซึ่งอยู่ในส่วนของการจำลองชุดคำสั่ง แล้วส่งต่อการทำงานไปยังส่วนการจำลองคำสั่งแต่ละคำสั่ง

3.3.2 หน้าต่างรีจิสเตอร์

หน้าต่างของรีจิสเตอร์จะเป็นการจำลองรีจิสเตอร์ที่มีอยู่ในไมโครคอนโทรลเลอร์ 68HC11 หน้าต่างนี้จะแสดงเมื่อมีการเลือกที่เมนูหลัก View แล้วเลือกที่คำสั่ง Register Window ก็จะแสดงหน้าต่างนี้ และถ้าหากไม่ได้เลือกตามขั้นตอนดังกล่าวขณะที่ทำการรันโปรแกรม หน้าต่างรีจิสเตอร์จะแสดงขึ้นมา โดยหน้าต่างนี้มีหน้าที่ในการแสดงผลของการทำคำสั่งที่มีผลรีจิสเตอร์ทุกคำสั่ง ซึ่งจะแสดงค่าที่เปลี่ยนแปลงเพื่อที่จะสังเกตการเปลี่ยนแปลงได้ หน้าต่างของรีจิสเตอร์แสดงได้ดังรูปที่ 3.8



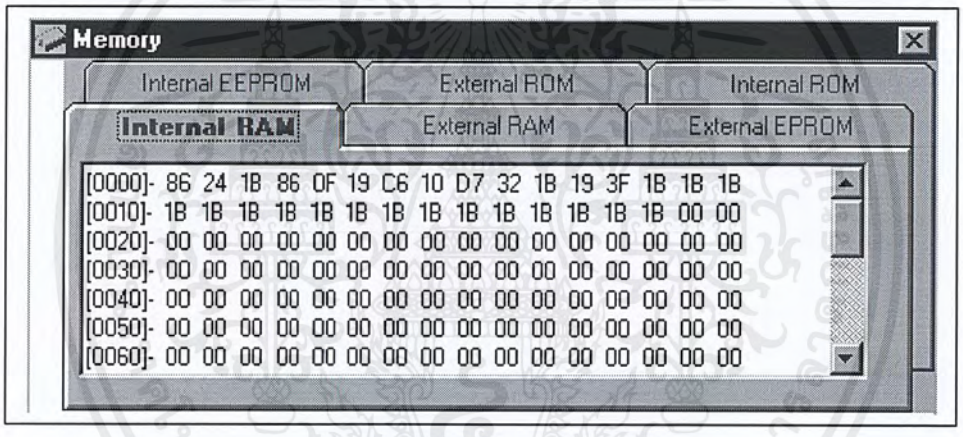
รูปที่ 3.8 หน้าต่างรีจิสเตอร์

จากรูปที่ 3.8 หน้าต่างรีจิสเตอร์ประกอบด้วยส่วนของรีจีสเตอร์ใช้งานทั่วไป ประกอบด้วยรีจิสเตอร์ขนาด 8 บิต 3 รีจิสเตอร์ คือ รีจิสเตอร์ A, B และ C, CCR รีจิสเตอร์ขนาด 16 บิต คือ รีจิสเตอร์ D, IX, IY, SP และ PC และในส่วนสถานะของแฟลกจะเป็นไฟบอกสถานะต่างๆ

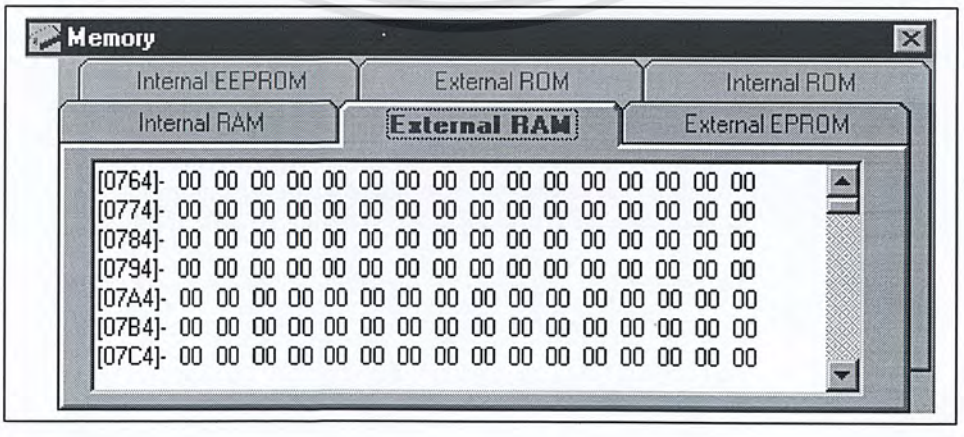
นอกจากนั้นในส่วนของหน้าต่างรีจิสเตอร์จะมีการรันเป็นทีละบรรทัด หรือ รันแบบต่อเนื่องได้ และมีปุ่มหยุด และ ปุ่มรีเซต เพื่อหยุด การรัน และล้างข้อมูลในรีจิสเตอร์

3.3.3 หน้าต่างหน่วยความจำ

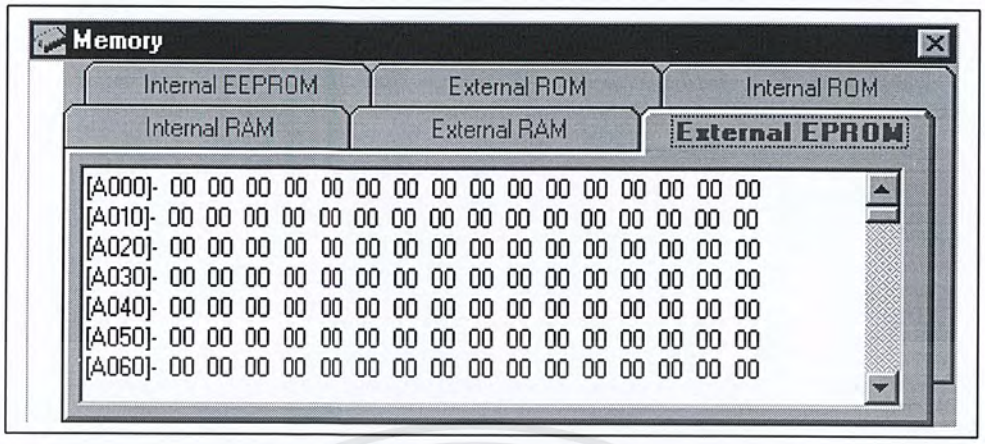
หน้าต่างหน่วยความจำจะเป็นการจำลองหน่วยความจำให้กับไมโครคอนโทรลเลอร์ 68HC11 หน้าต่างนี้จะแสดงเมื่อมีการเลือกที่เมนูหลัก View แล้วเลือกที่ Memory Window จะแสดงหน้าต่างนี้ ถ้าหากไม่ได้เลือกตามขั้นตอนดังกล่าว ขณะที่รันโปรแกรมหน้าต่างหน่วยความจำจะแสดงขึ้นมา ตำแหน่งของหน่วยความจำมีตั้งแต่ 0000H - FFFFH สามารถที่จะเลือกดูข้อมูลในหน่วยความจำได้โดยการเลื่อนสกอร์บาร์ หน้าต่างของหน่วยความจำสามารถแสดงได้ดังในรูปที่ 3.9-3.14.



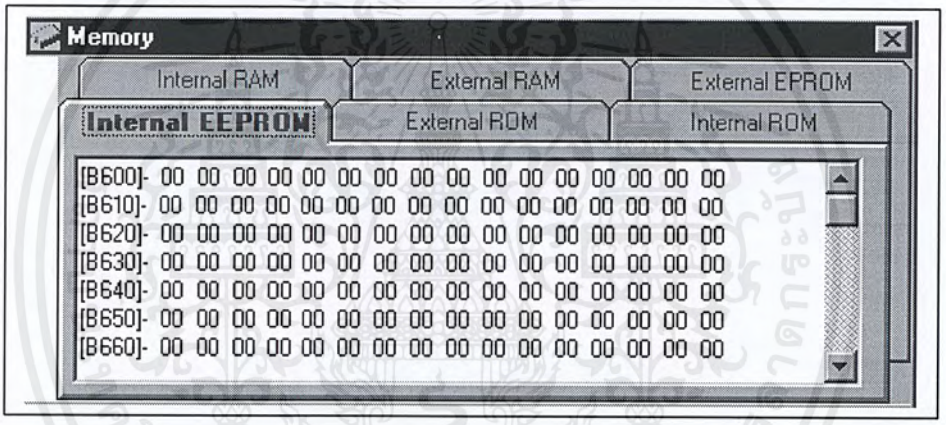
รูปที่ 3.9 หน้าต่างหน่วยความจำ (แรมภายใน)



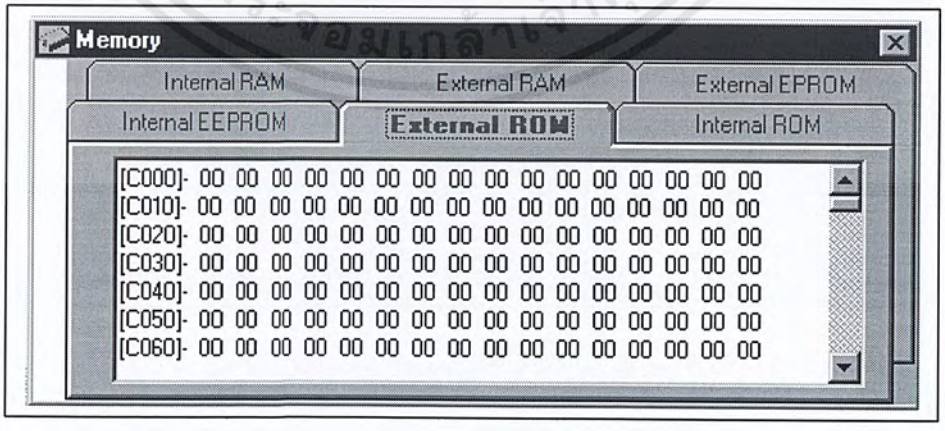
รูปที่ 3.10 หน้าต่างหน่วยความจำ (แรมภายนอก)



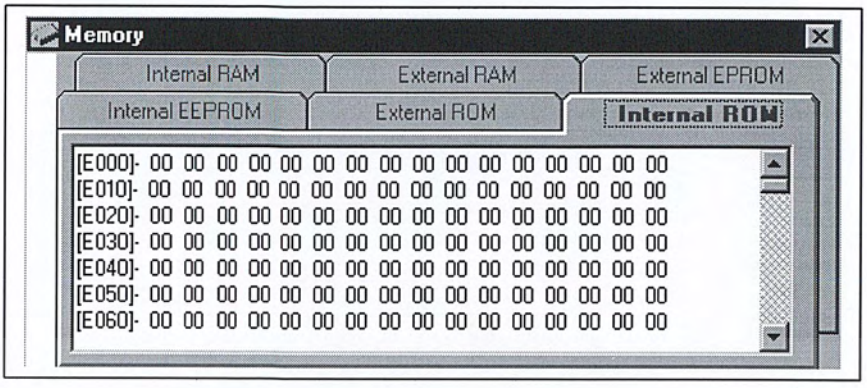
รูปที่ 3.11 หน้าต่างหน่วยความจำ (อีพรอมภายนอก)



รูปที่ 3.12 หน้าต่างหน่วยความจำ (ออีพรอมภายใน)

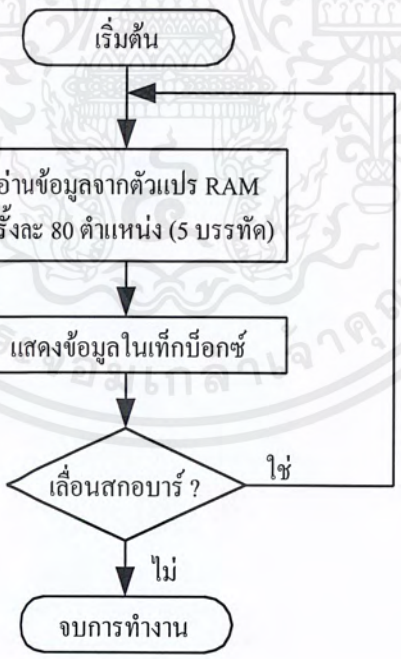


รูปที่ 3.13 หน้าต่างหน่วยความจำ (รอมภายนอก)



รูปที่ 3.14 หน้าต่างหน่วยความจำ (รอมภายใน)

หน้าต่างแสดงผลหน่วยความจำจะแสดงผลข้อมูล โดยการอ่านข้อมูลจากตัวแปรหน่วยความจำ แล้วทำการเขียนข้อมูลลงในเท็กบ็อกซ์เพื่อทำการแสดงข้อมูลทีละ 5 บรรทัด เมื่อมีการเลื่อนสกอร์บาร์ก็จะทำการแสดงค่าในตัวแปรหน่วยความจำ 5 บรรทัด ซึ่งขั้นตอนในการทำงานดังแสดงในรูปที่ 3.15



รูปที่ 3.15 แผนผังการแสดงผลของหน้าต่างหน่วยความจำ

3.3.4 หน้าต่างอิดิเตอร์

หน้าต่างอิดิเตอร์จะเป็นหน้าต่างที่ใช้สำหรับให้ผู้ใช้ทำการสร้างไฟล์ใหม่ หรือ ทำการเปิดไฟล์ที่เขียนไว้แล้วมาทำการแก้ไขใหม่อีกครั้ง ในหน้าต่างนี้จะประกอบไปด้วยเท็กบอกซ์เป็นตัวที่ใช้ในการเขียน และรับไฟล์จากการอ่านไฟล์มาแสดงให้ผู้ใช้ได้ การทำงานของหน้าต่างนี้จะเหมือนกับอิดิเตอร์ทั่วไป หน้าต่างอิดิเตอร์ดังแสดงในรูปที่ 3.16

รูปที่ 3.16 หน้าต่างอิดิเตอร์

รูปที่ 3.17 แผนผังการทำงานของหน้าต่างอิดิเตอร์

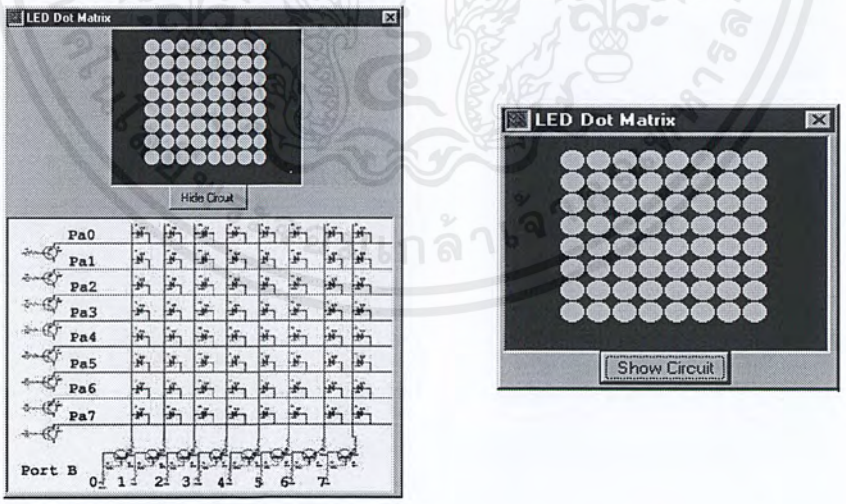
จากรูปที่ 3.17 ขั้นตอนการทำงานของหน้าต่างอิดิเตอร์จะรอรับการกดคีย์ใดๆ และจากการอ่านไฟล์เข้ามา หากมีการกดคีย์จะกำหนดให้ Dutyflag มีค่าเป็น True แล้วจึงทำการแสดงค่าที่รับมาในเท็กบ็อกซ์ แล้วจึงไปรอรับการกดคีย์อีกครั้ง

3.4 ภาคแสดงผล

ภาคแสดงผลจะมีลักษณะเป็นชุดอินเตอร์เฟสจำลอง ซึ่งจะทำการจำลองการทำงานเสมือนกับการทำงานของวงจรจริง ดังนั้นส่วนของชุดอินเตอร์เฟสจำลองในภาคแสดงผลนี้จำเป็นต้องใช้ฟังก์ชั่นจำลองการทำงานของ 8255 ซึ่งในการสร้างชุดอินเตอร์เฟสจะแบ่งชุดจำลองออกเป็น 3 ภาค คือ ภาคแสดงผลแอลอีดี, ภาคแสดงผลแบบแอลซีดี, ภาคแสดงผลแบบเจ็ดส่วน และภาคการจำลองการทำงานของมอเตอร์

3.4.1 ภาคแสดงผลแอลอีดี

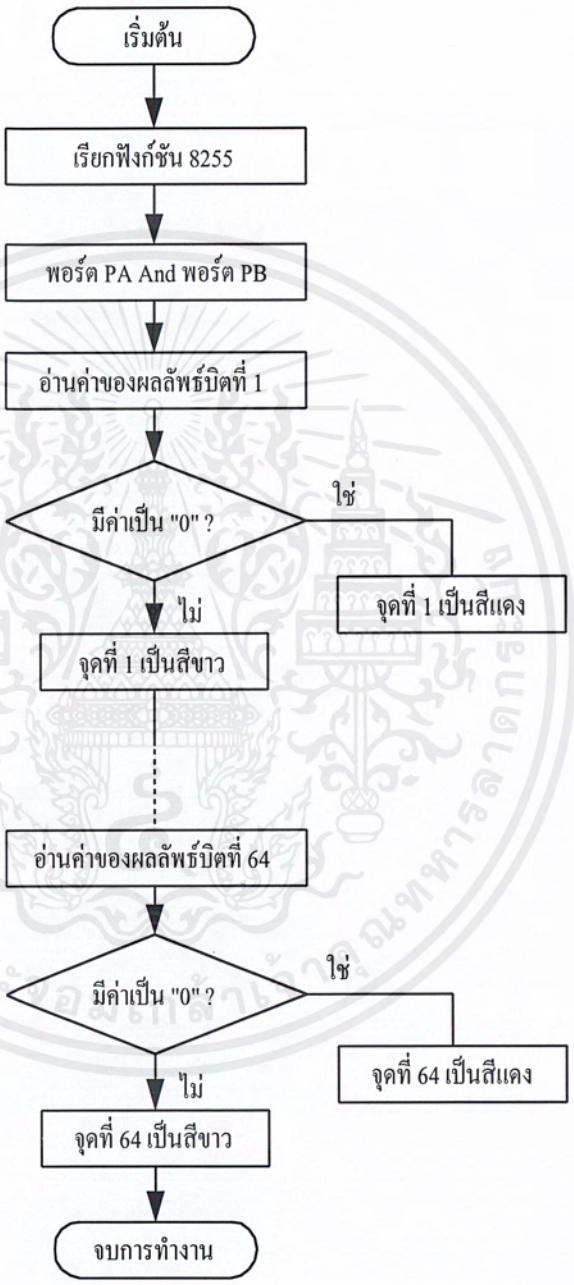
ในภาคแสดงผลแอลอีดีจะจำลองการทำงานของแสดงผลแอลอีดีเมตริกซ์ขนาด 8 x 8 โดยในการจำลองการทำงานจะต้องใช้ ฟังก์ชันจำลองการทำงานของ 8255 ในส่วนของตัวแอลอีดีจำลองใช้คอมโปเนนต์รูปร่างกำหนดให้มีลักษณะเป็นรูปวงกลม ในการแสดงผลจะใช้การกำหนดรูปร่างให้เปลี่ยนสีโดยใช้คุณสมบัติ Shape.Fillcolor ภาคแสดงผลแอลอีดีดังแสดงในรูปที่ 3.18



รูปที่ 3.18 ภาคแสดงผลแอลอีดี

การทำงานของภาคแสดงผลแอลอีดี เมื่อมีการเรียกใช้โปรแกรมจะทำการรับค่าผ่านทางพอร์ต 8255 ซึ่งผู้ใช้ต้องทำการเขียนโปรแกรมเพื่อทำการกำหนดค่าเริ่มต้นให้ 8255 ฟังก์ชันของ

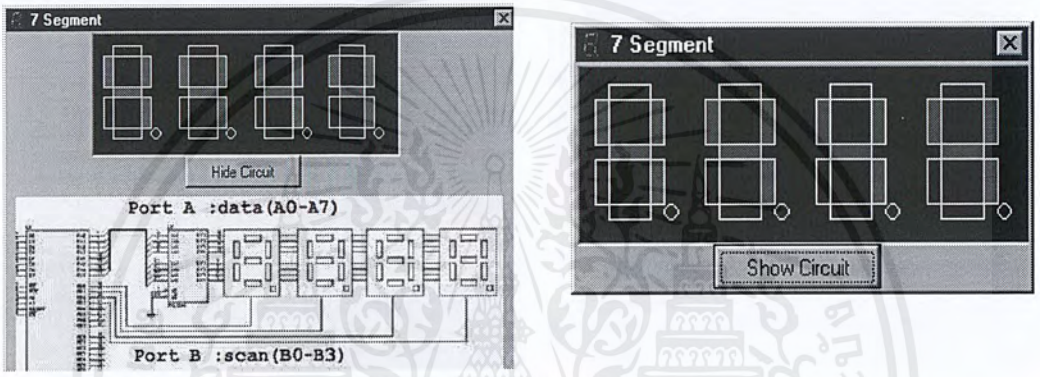
8255 จะส่งค่าของพอร์ต PA จำลอง, PB จำลอง และ PC จำลอง ค่าของ PA กับ PB จะถูกนำมา AND กันเมื่อค่า PA กับ PB มีค่าเป็น 1 ทำให้รูปร่างซึ่งจำลองเป็นแอลอีดีเปลี่ยนเป็นสีแดง



รูปที่ 3.19 แผนผังการทำงานของภาคแสดงผลแอลอีดี

3.4.2 ภาคแสดงผลแบบเจ็ดส่วน

การทำงานของภาคแสดงผลแบบเจ็ดส่วนนี้ ต้องใช้ฟังก์ชันของ 8255 เหมือนกับภาคแสดงผลแบบแอลอีดี การใช้งานส่วนนี้จะต้องที่การส่งค่าผ่านฟังก์ชันของ 8255 ซึ่งภาคแสดงผลนี้ประกอบด้วยแบบเจ็ดส่วนทั้งหมด 4 ตัว โดยการออกแบบภาคแสดงผลแบบเจ็ดส่วนจำลอง คือ การทำรูปร่างเป็นรูปของอุปกรณ์อิเล็กทรอนิกส์แบบเจ็ดส่วนแอดโนดต่อร่วม ดังที่ได้กล่าวไว้ในส่วนของบทที่ 2 ซึ่งภาคแสดงผลแบบเจ็ดส่วนดังแสดงในรูปที่ 3.20

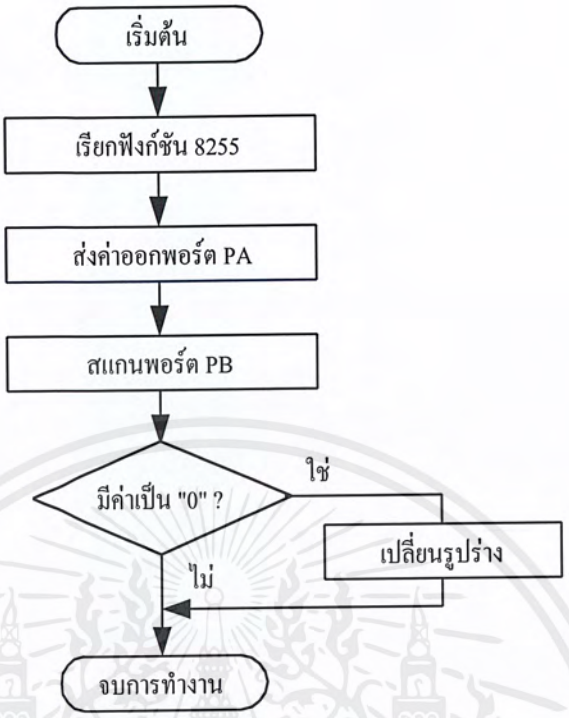


รูปที่ 3.20 ภาคแสดงผลแบบเจ็ดส่วน

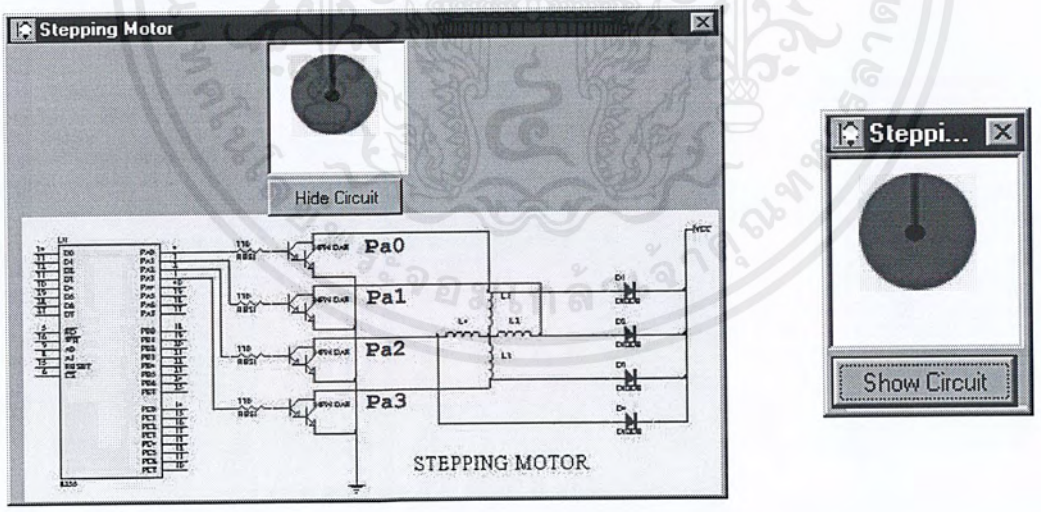
การทำงานของภาคแสดงผลแบบเจ็ดส่วนนี้ ในขั้นแรกจะส่งค่าข้อมูลออกไปยังฟังก์ชันของ 8255 จำลอง จากนั้นข้อมูลจะถูกส่งออกทางพอร์ต PA ส่วนพอร์ต PB0, PB1 และ PB2 ซึ่งจะใช้ในการสแกน ซึ่งขั้นตอนในการสแกนนั้นหากตัวใดที่มีค่าเป็น “0” รูปร่างของภาคตัวแสดงผลแบบเจ็ดส่วนจะไม่เปลี่ยนแปลงสี หากบิตใดมีค่าเป็น “1” ก็จะทำการเปลี่ยนแปลงสีให้เป็นสีแดง ขั้นตอนนี้จะทำอยู่ไปจนกว่าจะครบตามจำนวนที่ตั้งไว้ ซึ่งขั้นตอนการทำงานของภาคแสดงผลแบบเจ็ดส่วนดังแสดงในรูปที่ 3.21

3.4.3 การจำลองการทำงานของสเต็ปปิ้งมอเตอร์

ชุดอินเตอร์เฟสจำลองการทำงานของสเต็ปปิ้งมอเตอร์ จะเป็นโปรแกรมจำลองการทำงานของสเต็ปปิ้งมอเตอร์ โดยการจำลองการทำงานจะใช้ฟังก์ชัน 8255 เป็นตัวติดต่อภาคแสดงผลแบบแอลซีดี และแบบเจ็ดส่วน การแสดงผลจะใช้คำสั่ง Line ของวิชวลเบสิก โดยทำเป็นสถานะต่างๆ ของมอเตอร์ไว้ ดังแสดงในตารางที่ 3.2 ส่วนของชุดอินเตอร์เฟสจำลองสเต็ปปิ้งมอเตอร์ ดังแสดงในรูปที่ 3.22



รูปที่ 3.21 การทำงานของภาคแสดงแบบเจ็ดส่วน

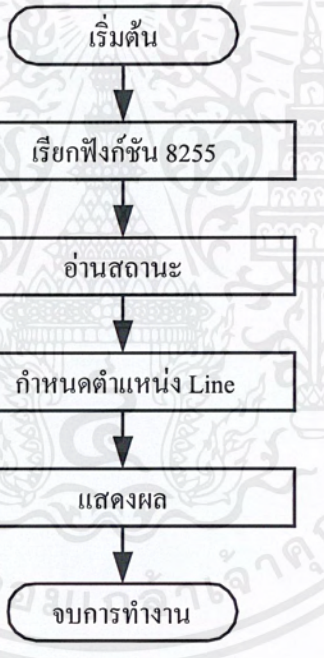


รูปที่ 3.22 ชุดอินเตอร์เฟสจำลองการทำงานของสเต็ปปิ้งมอเตอร์

ชุดอินเตอร์เฟสจำลองการทำงานของสเต็ปปิ้งมอเตอร์ จะมีการเรียกใช้โปรแกรมเพื่อทำการรับค่าผ่านทางพอร์ต 8255 เมื่อรับค่ามาแล้วจะทำการตรวจสอบสถานะของข้อมูลที่ส่งมาว่าตรงกับสถานะใด

ตารางที่ 3.2 มุมของโรเตอร์เทียบกับกระแสไฟฟ้าที่จ่ายแก่เฟสต่างๆ 8 ตำแหน่ง

เฟสที่จ่ายกระแสไฟฟ้า	ตำแหน่งโรเตอร์
$\varnothing_1$	
$\varnothing_1\varnothing_2$	
$\varnothing_2$	
$\varnothing_2\varnothing_3$	
$\varnothing_3$	
$\varnothing_3\varnothing_4$	
$\varnothing_4$	
$\varnothing_4\varnothing_1$	



รูปที่ 3.23 การทำงานของชุดอินเตอร์เฟสจำลองการทำงานของสเต็ปปิ้งมอเตอร์

3.5 การจำลองชุดคำสั่ง

3.5.1 การออกแบบหน่วยความจำ และรีจิสเตอร์จำลอง

ในการออกแบบหน่วยความจำและรีจิสเตอร์จำลอง มีการกำหนดตัวแปรต่าง ๆ ที่ใช้ในการจำลอง ดังต่อไปนี้

1) Public flag As Byte เป็นตัวแปร CCR แฟลกตัว ชนิดไบต์

2) Public RegisA As Long เป็นตัวแปรรีจิสเตอร์ A แทนข้อมูลขนาด 8 บิตชนิด Long

3) Public Regisb As Long เป็นตัวแปรรีจิสเตอร์ B แทนข้อมูลขนาด 8 บิตชนิด Long

4) Public Regisd As Long เป็นตัวแปรรีจิสเตอร์ D แทนข้อมูลขนาด 16 บิตชนิด Long

5) Public Regisy As Long เป็นตัวแปรรีจิสเตอร์ IY แทนข้อมูลขนาด 16 บิตชนิด Long

6) Public Regisx As Long เป็นตัวแปรรีจิสเตอร์ IX แทนข้อมูลขนาด 16 บิตชนิด Long

7) Public RegisSP As Long เป็นตัวแปรรีจิสเตอร์ SP แทนข้อมูลขนาด 16 บิตชนิด Long

8) Public RegisPC As Long เป็นตัวแปรรีจิสเตอร์ PC แทนข้อมูลขนาด 16 บิตชนิด Long

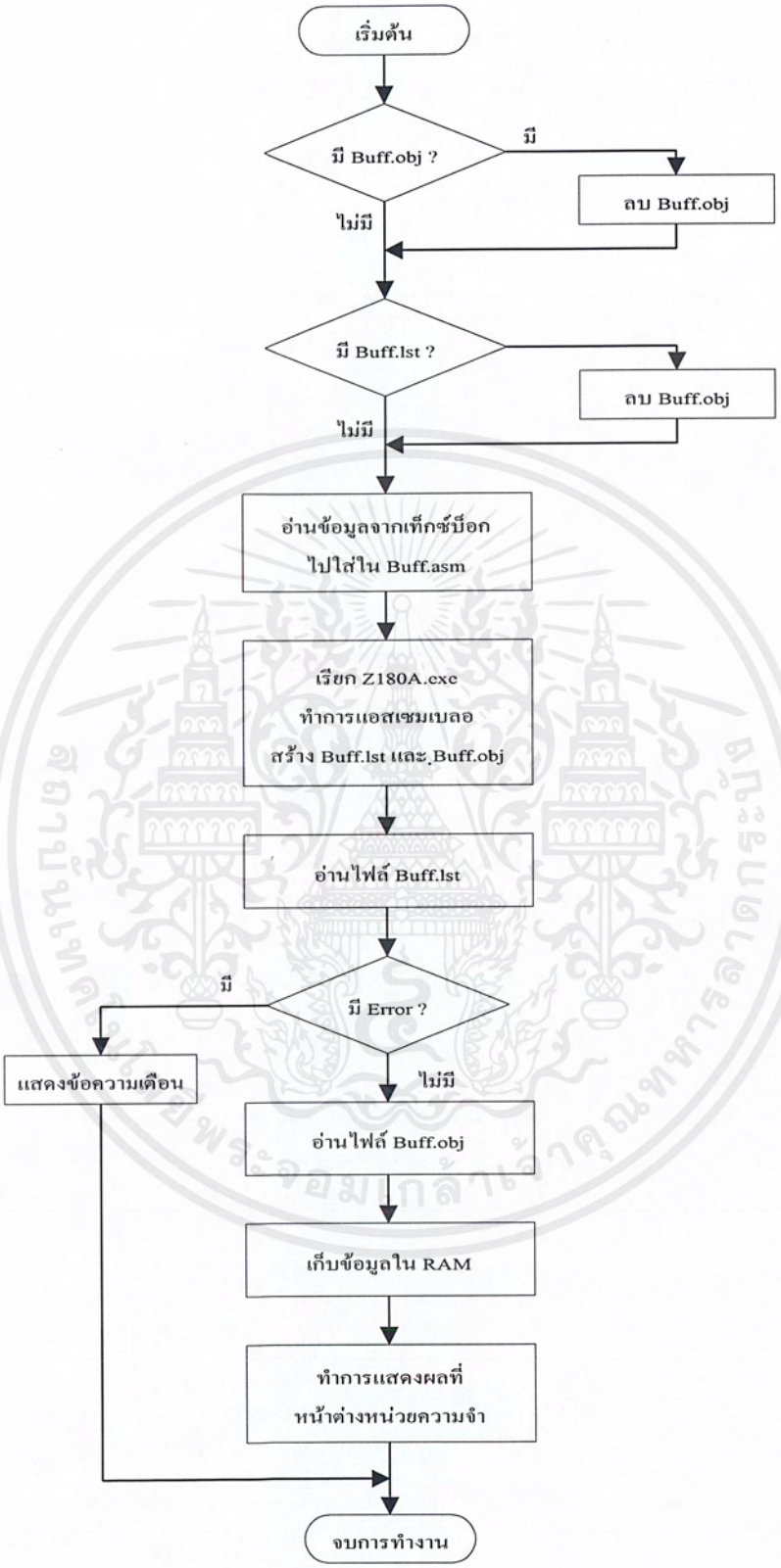
9) Public RAM (65535) As Long ตัวแปรหน่วยความจำจำลองมีขนาด 64 กิโลไบต์ ซึ่งมีค่าเท่ากับ 65,536 ค่า แต่ละอินเด็กซ์บอกตำแหน่งหน่วยความจำ ซึ่งตัวไมโครคอนโทรลเลอร์ 68HC11 หน่วยความจำแต่ละตำแหน่งจะมีขนาดเท่ากับ 8 บิต ซึ่งในวิชวลเบสิกนั้นจะต้องประกาศตัวแปรเป็นชนิด Long

3.5.2 การจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11

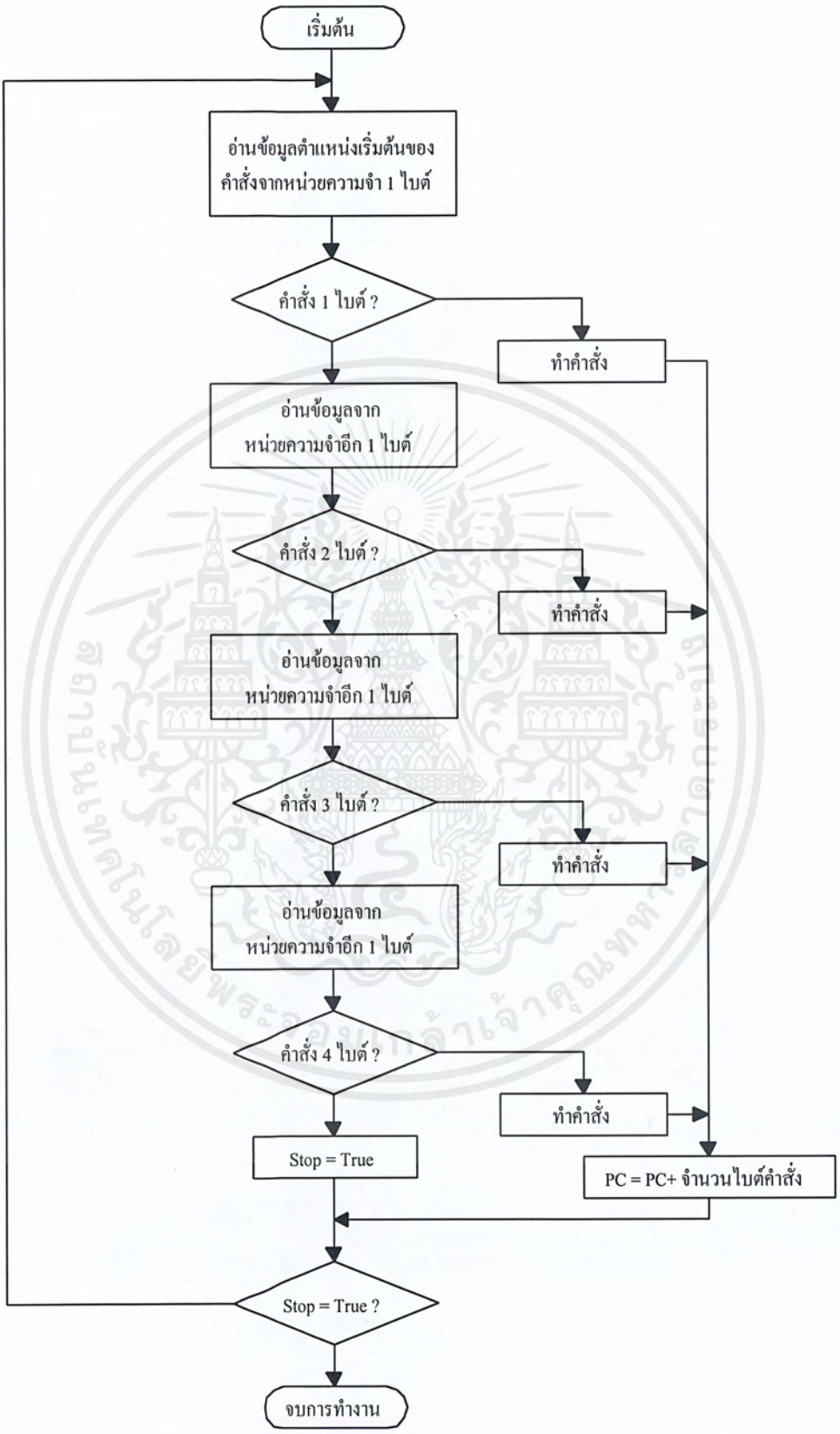
การจำลองการทำงานของไมโครโปรเซสเซอร์นี้จะทำโดยการใช้โปรแกรมวิชวลเบสิกเขียนโปรแกรมควบคุมการทำงาน โดยออกแบบการทำงานนั้นจะแบ่งออกเป็น 2 ส่วน คือ ส่วนของการแอสเซมเบลอร์ และส่วนของการจำลองการทำงาน ซึ่งทั้งสองส่วนมีการทำงานดังนี้

1) ส่วนของการแอสเซมเบลอร์ คือส่วนของการแปลงโปรแกรมแอสแซมบลี้ ที่ได้สร้างขึ้นในส่วนของอิดิเตอร์ เป็นรหัสเลขฐานสิบหก

การทำงานของส่วนของการแอสเซมเบลอร์ จะทำการตรวจสอบก่อนว่ามีไฟล์ Buf.obj และไฟล์ Buf.lst ไฟล์ทั้งสองนี้จะได้จากการแอสเซมเบลอร์ ซึ่งก่อนที่จะทำการแอสเซมเบลอร์ในครั้งนี้อาจจะได้ทำการแอสเซมเบลอร์ก่อนหน้านี้แล้ว ดังนั้นจึงต้องมีการตรวจสอบ หากพบว่ามีไฟล์ทั้งสองอยู่ หรือมีไม่ว่าจะเป็นไฟล์ใดไฟล์หนึ่งก็จะทำการลบไฟล์นั้นออก จากนั้นจะทำการอ่านข้อมูลที่อยู่ในเท็กบ็อกซ์ที่อยู่ในหน้าต่างอิดิเตอร์ที่ได้ทำการเขียนไว้แล้ว จากนั้นจะทำการเรียกตัวโปรแกรมแอสเซมเบลอร์มาทำการแปลงรหัส ซึ่งตัวแอสเซมเบลอร์ที่ใช้นั้น จะใช้ตัวคอมไพเลอร์ของไมโครคอนโทรลเลอร์ AS11 ซึ่งเป็นตัวที่สนับสนุนคำสั่งของไมโครคอนโทรลเลอร์ 68HC11 จึงทำให้สามารถที่จะแอสเซมเบลอร์ได้โดยไฟล์ที่จะได้จากแปลงรหัสนั้นจะมีอยู่ 2 ไฟล์คือ



รูปที่ 3.24 แผนผังขั้นตอนการทำงานของส่วนการแอสเซมเบลอร์



รูปที่ 3.25 แผนผังขั้นตอนการทำงานของส่วนการจำลองการทำงาน

คือไฟล์ Buf.obj และ Buf.lst จากนั้นจะอ่านไฟล์ Buf.lst เพื่อตรวจสอบว่ามีข้อผิดพลาดที่ใด หากตรวจสอบว่ามีข้อผิดพลาดก็จะแสดงข้อความเตือน เพื่อให้ไปทำการแก้ไขแล้วจึงมาทำการแอสเซม-เบลอร์อีกครั้ง หากตรวจสอบแล้วไม่มีข้อผิดพลาด จะอ่านข้อมูลจากไฟล์ Buf.obj ซึ่งเป็นรหัสเลขฐานสิบหก นำไปเก็บในตัวแปรหน่วยความจำจำลอง คือ RAM แล้วแสดงผลออกทางหน้าต่างของหน่วยความจำว่า เมื่อทำการแปลงแล้วรหัสที่เขียนไว้ มีคำสั่งอย่างไร เป็นเลขฐานสิบหกเท่าไร และจะเริ่มเก็บข้อมูลที่ตำแหน่งหน่วยความจำ 0000H ขึ้นไป แล้วจึงแสดงข้อความขึ้นมาบอกว่าไม่มีข้อผิดพลาด และสามารถทำการรันคำสั่งเพื่อจำลองการทำงานได้ทันที ซึ่งแผนผังขั้นตอนการทำงานดังแสดงรูปที่ 3.24

2) ส่วนของการจำลองการทำงาน คือ ส่วนของการอ่านรหัสที่ได้จากการแอสเซมเบลอร์แล้วมาทำตามรหัสของทำสั่งที่สั่งให้ไมโครคอนโทรลเลอร์ 68HC11 ทำงาน ซึ่งในส่วนนี้จะทำการจำลองชุดคำสั่งการทำงานทั้งหมดที่มีอยู่แล้วส่งผลออกสู่ส่วนของการแสดงผลต่างๆ

การทำงานในส่วนของการจำลองการทำงาน จะอ่านข้อมูลที่ได้จากการแอสเซมเบลอร์ ซึ่งอยู่ในหน่วยความจำจำลองที่สร้างเอาไว้ โดยจะเริ่มอ่านที่ตำแหน่งเริ่มต้นของโปรแกรม โดยจะเริ่มอ่านข้อมูลเข้ามาทีละ 1 ไบต์ แล้วตรวจสอบว่าเป็นคำสั่งขนาด 1 ไบต์, 2 ไบต์, 3 ไบต์, 4 ไบต์ หรือ 5 ไบต์ ถ้าไม่ใช่จะสั่งให้หยุดทำงานทันทีถ้าพบว่าเป็นคำสั่งก็จะทำงานตามคำสั่งนั้นและเพิ่มค่าในโปรแกรมเคาท์เตอร์ขึ้นตามจำนวนไบต์ของคำสั่ง และตรวจสอบว่ามีการสั่งให้หยุดทำงานหรือไม่ หากพบจะสั่งให้หยุดทำงานทันที ถ้าไม่พบก็จะเริ่มอ่านข้อมูลเข้ามาใหม่ทำเช่นนี้ไปเรื่อยๆ จนพบคำสั่งให้หยุดทำงาน แผนผังขั้นตอนการทำงานดังแสดงรูปที่ 3.25

บทที่ 4

การทดลองและผลการทดลอง

4.1 กล่าวนำ

ในบทนี้จะกล่าวถึงวิธีการทดสอบกลุ่มคำสั่งของไมโครคอนโทรลเลอร์ 68HC11 โดยแบ่งการทดสอบออกเป็นกลุ่มๆ คือ กลุ่มคำสั่งการจัดการเกี่ยวกับข้อมูล, กลุ่มคำสั่งทางคณิตศาสตร์, กลุ่มคำสั่งทางลอจิก, กลุ่มคำสั่งเปรียบเทียบและตรวจสอบข้อมูล, กลุ่มคำสั่งจัดการเกี่ยวกับรีจิสเตอร์เงื่อนไข และกลุ่มคำสั่งควบคุม

4.2 กลุ่มคำสั่งการจัดการเกี่ยวกับข้อมูล

กลุ่มคำสั่งการจัดการเกี่ยวกับข้อมูลจะประกอบด้วย

1) กลุ่มคำสั่งเคลื่อนย้ายข้อมูล แบ่งออกเป็นกลุ่มเคลื่อนย้ายข้อมูลระดับบิต ระดับไบต์ และเวิร์ด

2) คำสั่งเปลี่ยนแปลงแก้ไขข้อมูล

4.2.1 การทดสอบคำสั่งเคลื่อนย้ายข้อมูล

กลุ่มคำสั่งการเคลื่อนย้ายข้อมูล มีรูปแบบของคำสั่งดังแสดงในรูปที่ 4.1

LDAA #$XX #$XX หมายถึง ค่าที่ต้องการเก็บไว้ในรีจิสเตอร์ A

รูปที่ 4.1 รูปแบบคำสั่งเคลื่อนย้ายข้อมูล

การทดสอบ

ทดสอบคำสั่งเคลื่อนย้ายข้อมูล ด้วยการป้อนโปรแกรมดังแสดงในรูปที่ 4.2 โดยโปรแกรมจะเก็บค่า 05H ไว้ในรีจิสเตอร์ A โดยที่โค้ดของโปรแกรมทั้งหมดเก็บที่หน่วยความจำตำแหน่ง 00H

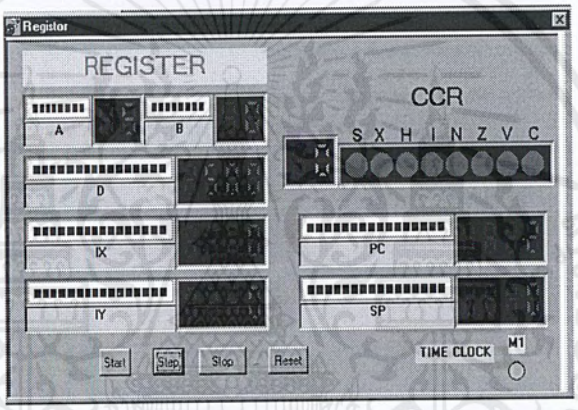
```
LDAA    #$05
```

รูปที่ 4.2 โปรแกรมทดสอบคำสั่งเคลื่อนย้ายข้อมูล

ผลการทดสอบคำสั่ง

ผลการทดสอบเมื่อรันด้วยโปรแกรม 68HC11 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.3

```
[0000]- 86 05 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0010]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0020]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0030]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0040]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0050]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0060]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```



รูปที่ 4.3 ผลการทดสอบคำสั่งเคลื่อนย้ายข้อมูลในหน้าต่างรีจิสเตอร์และหน้าต่างหน่วยความจำ

4.2.2 การทดสอบคำสั่งเปลี่ยนแปลงแก้ไขข้อมูล

การเปลี่ยนแปลงแก้ไขข้อมูลสามารถทำได้ 3 รูปแบบ คือ การเพิ่มค่า, การลดค่า และการแปลงค่า

การทดสอบ

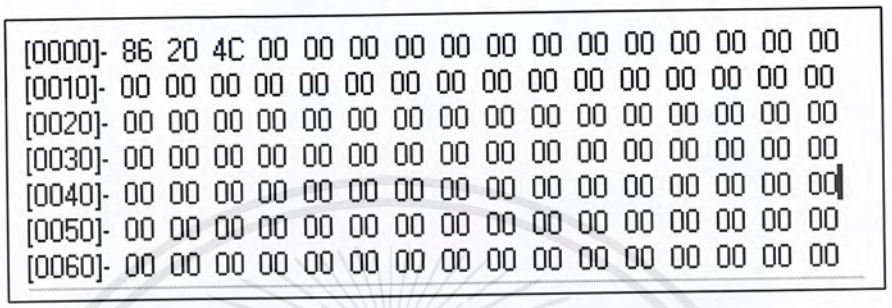
การทดสอบคำสั่งการเพิ่มค่าโดยการป้อนโปรแกรมดังรูปที่ 4.4 โค้ดโปรแกรมจะเก็บค่าไว้ที่หน่วยความจำตำแหน่ง 00H ถึง ตำแหน่ง 02H

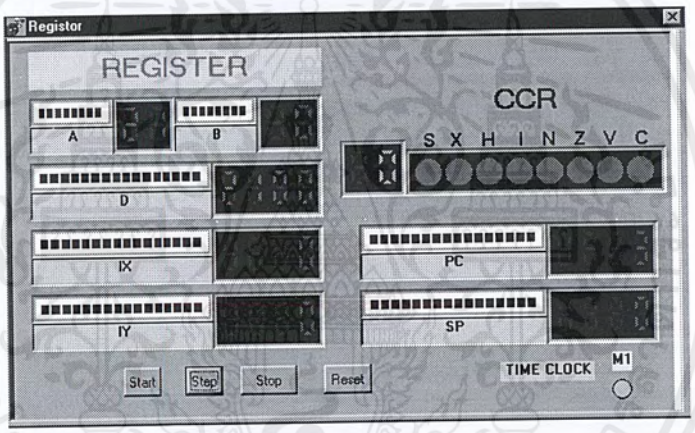
```
LDAA   #$20
INCA
```

รูปที่ 4.4 โปรแกรมทดสอบคำสั่งเพิ่มค่า

ผลการทดสอบ

ผลการทดสอบเมื่อรันด้วยโปรแกรม 68HC11 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 02H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.5





รูปที่ 4.5 ผลการทดสอบคำสั่งเพิ่มค่าในหน้าต่างรีจิสเตอร์และหน้าต่างหน่วยความจำ

4.3 กลุ่มคำสั่งทางคณิตศาสตร์

กลุ่มคำสั่งทางคณิตศาสตร์จะประกอบด้วย

1) กลุ่มคำสั่งการบวก

2) กลุ่มคำสั่งการลบ

3) กลุ่มคำสั่งการคูณและหาร

4.3.1 การทดสอบคำสั่งการบวก

การทดสอบ

การทดสอบคำสั่งการบวก ด้วยการป้อนโปรแกรมดังรูปที่ 4.6 โดยโค้ดโปรแกรมจะเก็บไว้ที่ตำแหน่งหน่วยความจำ 00H

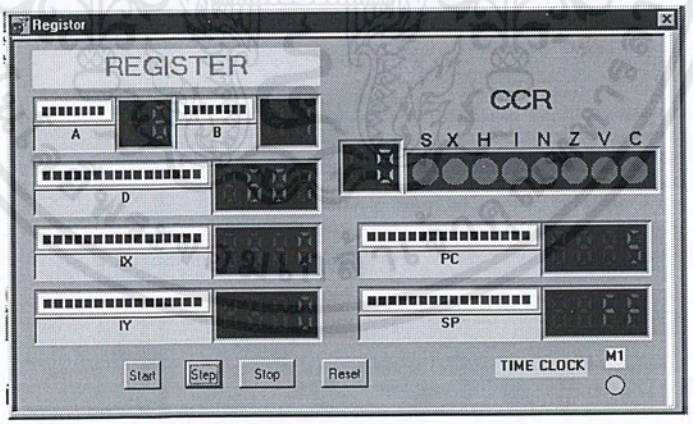
```
LDAA   #$05
LDAB   #$01
ABA
```

รูปที่ 4.6 โปรแกรมทดสอบคำสั่งการบวก

ผลการทดสอบคำสั่ง

ผลการทดสอบเมื่อรันด้วยโปรแกรม 68HC11 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 05H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.7

```
[0000]- 86 05 C6 01 1B 00 00 00 00 00 00 00 00 00 00 00
[0010]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0020]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0030]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0040]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0050]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0060]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```



รูปที่ 4.7 ผลการทดสอบคำสั่งการบวกในหน้าต่างรีจิสเตอร์และหน้าต่างหน่วยความจำ

4.3.2 การทดสอบคำสั่งการลบ

การทดสอบ

การทดสอบคำสั่งการลบ ด้วยการป้อนโปรแกรมดังรูปที่ 4.8 โดยโค้ดโปรแกรมจะเก็บไว้ที่ตำแหน่งหน่วยความจำ 00H

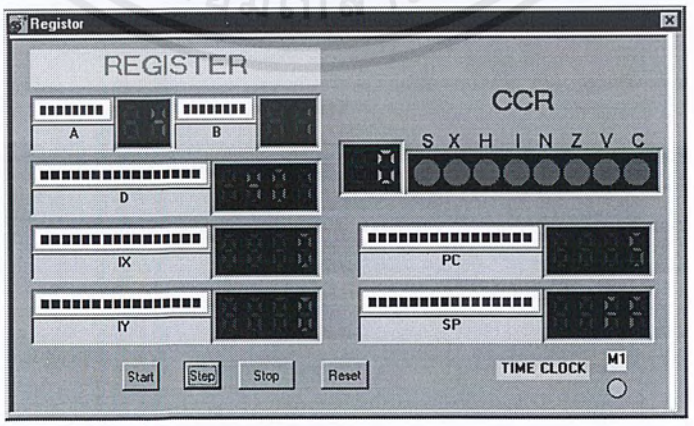
```
LDAA    #$05
LDAB    #$01
SBA
```

รูปที่ 4.8 โปรแกรมทดสอบคำสั่งการลบ

ผลการทดสอบคำสั่ง

ผลการทดสอบเมื่อรันด้วยโปรแกรม 68HC11 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 05H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.9

```
[0000]- 86 05 C6 01 10 00 00 00 00 00 00 00 00 00 00 00
[0010]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0020]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0030]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0040]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0050]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0060]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```



รูปที่ 4.9 ผลการทดสอบคำสั่งการลบในหน้าต่างรีจิสเตอร์และหน้าต่างหน่วยความจำ

4.3.3 การทดสอบคำสั่งคูณและหาร

การทดสอบ

การทดสอบคำสั่งการคูณ ด้วยการป้อนโปรแกรมดังรูปที่ 4.8 โดยโค้ดโปรแกรมจะเก็บไว้ที่ตำแหน่งหน่วยความจำ 00H

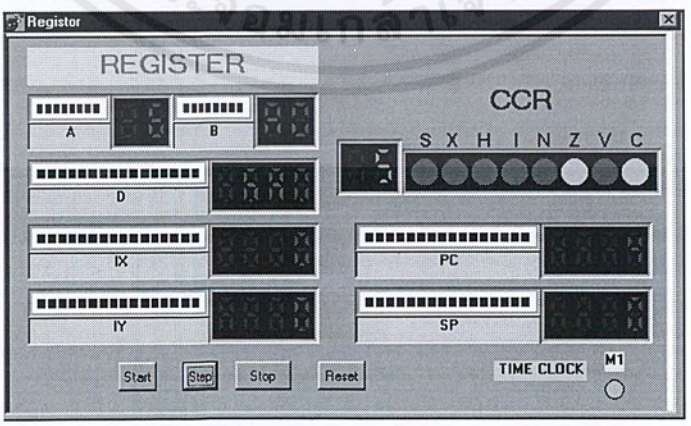
```
LDD     #$2035
MUL
ADCA    #$00
```

รูปที่ 4.10 ผลการทดสอบคำสั่งการคูณ

ผลการทดสอบ

ผลการทดสอบคำสั่งการคูณ เมื่อรันด้วยโปรแกรม 68HC11 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 05H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ ดังแสดงในรูปที่ 4.11

```
[0000]- CC 20 35 3D 89 00 00 00 00 00 00 00 00 00 00 00
[0010]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0020]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0030]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0040]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0050]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0060]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```



รูปที่ 4.11 ผลการทดสอบสอบคำสั่งการคูณในหน้าต่างรีจิสเตอร์ และหน้าต่างหน่วยความจำ

4.4 กลุ่มคำสั่งทางลอจิก

มีด้วยกัน 3 กระบวนการคือ แอนด์, ออร์ และเอ็กซ์คลูซีฟออร์

การทดสอบ

การทดสอบคำสั่งการแอนด์ ด้วยการป้อนโปรแกรมดังรูปที่ 4.12 โดยโค้ดโปรแกรมจะเก็บไว้ที่ตำแหน่งหน่วยความจำ 00H

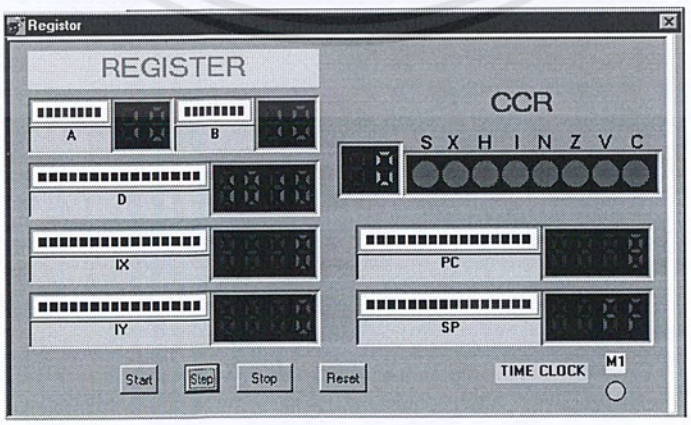
```
LDAA   #$20
LDAB   #$10
STAB   #$11
ANDA   $11
```

รูปที่ 4.12 ผลการทดสอบคำสั่งการแอนด์

ผลการทดสอบ

ผลการทดสอบคำสั่งการแอนด์ เมื่อรันด้วยโปรแกรม 68HC11 Simulator ที่ถูกต้อง โปรแกรมตำแหน่ง 00H ถึง 07H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ ดังแสดงในรูปที่ 4.13

```
[0000]- 86 20 C6 10 D7 11 84 11 00 00 00 00 00 00 00 00
[0010]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0020]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0030]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0040]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0050]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0060]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```



รูปที่ 4.13 ผลการทดสอบสอบคำสั่งการแอนด์

4.5 กลุ่มคำสั่งการเปรียบเทียบและทดสอบข้อมูล

แบ่งลักษณะการเปรียบเทียบและทดสอบได้ 2 ลักษณะตามขนาดของข้อมูล ดังนี้

1) กลุ่มคำสั่งการเปรียบเทียบและทดสอบข้อมูลขนาด 8 บิต

2) กลุ่มคำสั่งการเปรียบเทียบและทดสอบข้อมูลขนาด 16 บิต

การทดสอบ

การทดสอบคำสั่งเปรียบเทียบด้วยการป้อนโปรแกรมดังรูปที่ 4.14 โดยโค้ดโปรแกรมจะเก็บไว้ที่ตำแหน่งหน่วยความจำ 00H

```
LDAA   #$20
LDAB   #$10
CBA    #$11
```

รูปที่ 4.14 โปรแกรมทดสอบคำสั่งการเปรียบเทียบ

ผลการทดสอบ

ผลการทดสอบคำสั่งการคูณ เมื่อรันด้วยโปรแกรม 68HC11 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 05H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ ดังแสดงในรูปที่ 4.16

4.6 กลุ่มคำสั่งการกระโดด

แบ่งออกเป็น 2 ประเภท คือ

1) กลุ่มคำสั่งการกระโดดแบบไม่มีเงื่อนไข

2) กลุ่มคำสั่งการกระโดดแบบมีเงื่อนไข

4.6.1 การทดสอบคำสั่งการกระโดดแบบไม่มีเงื่อนไข

การทดสอบ

ทดสอบคำสั่งการกระโดดแบบไม่มีเงื่อนไข ด้วยการป้อนโปรแกรมดังแสดงในรูปที่ 4.15 โดยที่โค้ดของโปรแกรมทั้งหมดเก็บที่หน่วยความจำตำแหน่ง 00H

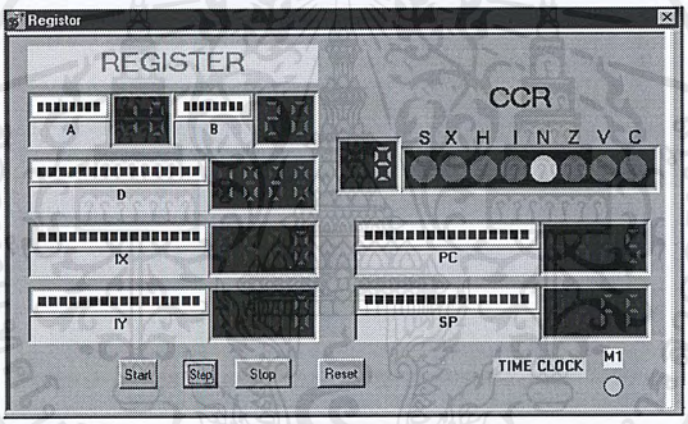
ผลการทดสอบคำสั่ง

ผลการทดสอบเมื่อรันด้วยโปรแกรม 68HC11 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.17

```
LDAA #03
LDAB $05
JMP  LOOP
```

รูปที่ 4.15 โปรแกรมทดสอบคำสั่งกระโดดแบบไม่มีเงื่อนไข

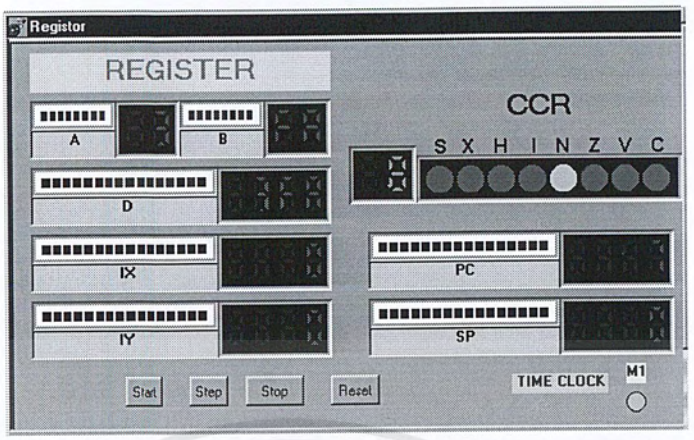
```
[0000]- 86 10 C6 20 11 00 00 00 00 00 00 00 00 00 00 00
[0010]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0020]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0030]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0040]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0050]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0060]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```



รูปที่ 4.16 ผลการทดสอบสอบคำสั่งเปรียบเทียบในหน่วยความจำจำลองและหน้าต่างรีจิสเตอร์

```
[0000]- 86 03 D6 05 21 FA 00 00 00 00 00 00 00 00 00 00
[0010]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0020]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0030]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0040]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0050]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0060]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

รูปที่ 4.17 (ก) ผลการทดสอบคำสั่งกระโดดแบบไม่มีเงื่อนไข ในหน้าต่างรีจิสเตอร์ และหน่วยความจำ



รูปที่ 4.17 (ข) ผลการทดสอบคำสั่งกระโดดแบบไม่มีเงื่อนไข ในหน้าต่างรีจิสเตอร์และหน่วยความจำ

4.6.2 การทดสอบคำสั่งการกระโดดแบบมีเงื่อนไข

การทดสอบ

ทดสอบคำสั่งการกระโดดแบบมีเงื่อนไข ด้วยการป้อนโปรแกรมดังแสดงในรูปที่ 4.18 โดยที่โค้ดของโปรแกรมทั้งหมดเก็บที่หน่วยความจำตำแหน่ง 00H

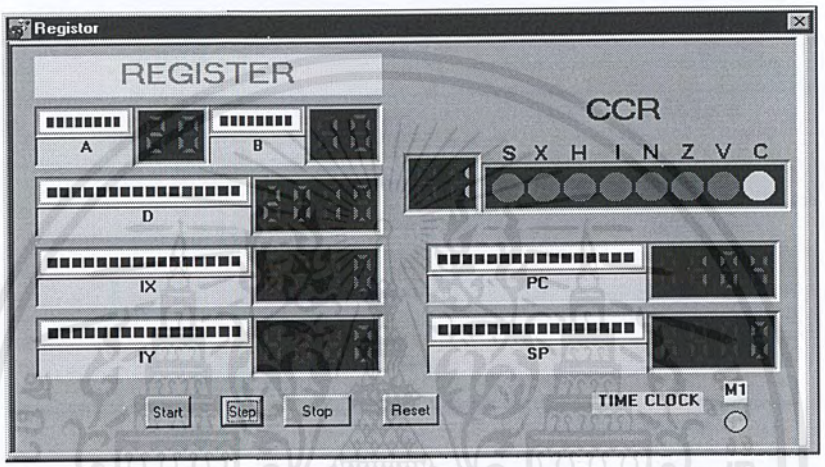
```
LDAA #$30
SEC
BCS #12
```

รูปที่ 4.18 โปรแกรมทดสอบคำสั่ง

ผลการทดสอบคำสั่ง

ผลการทดสอบเมื่อรันด้วยโปรแกรม 68HC11 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.19

```
'[A000]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[A010]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[A020]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[A030]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[A040]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[A050]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[A060]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```



รูปที่ 4.19 ผลการทดสอบสอบคำสั่งกระโดดแบบมีเงื่อนไขในหน้าต่างหน่วยความจำจำลอง และหน้าต่างรีจิสเตอร์

4.7 กลุ่มคำสั่งจัดการกับรีจิสเตอร์รหัสเงื่อนไข

ในกลุ่มนี้เป็นคำสั่งที่ทำการเซตหรือเคลียร์บิตต่างๆ ในรีจิสเตอร์รหัสเงื่อนไข

การทดสอบ

ทดสอบคำสั่งจัดการกับรีจิสเตอร์เงื่อนไข ด้วยการป้อนโปรแกรมดังแสดงในรูปที่ 4.20 โดยที่โค้ดของโปรแกรมทั้งหมดเก็บที่หน่วยความจำตำแหน่ง 00H

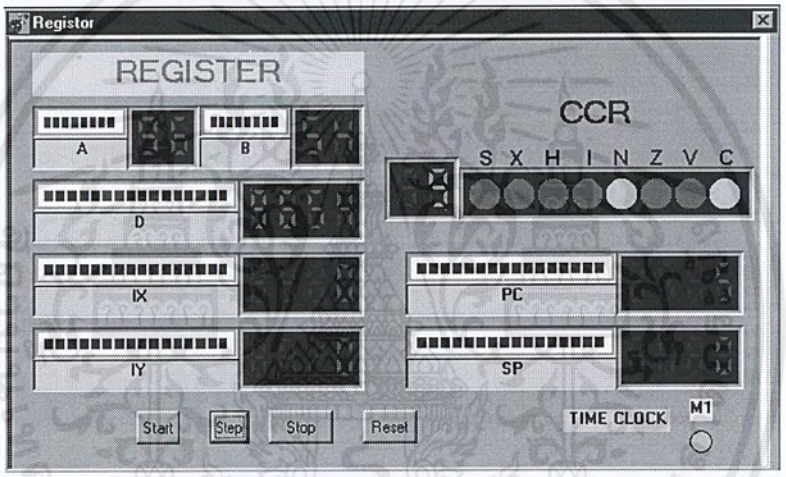
```
LDDA  #$55
LDAB  #$65
SEC
```

รูปที่ 4.20 โปรแกรมทดสอบคำสั่ง SEC

ผลการทดสอบคำสั่ง

ผลการทดสอบเมื่อรันด้วยโปรแกรม 68HC11 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.21

```
[0000]- C6 54 96 02 0D 00 00 00 00 00 00 00 00 00 00 00
[0010]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0020]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0030]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0040]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0050]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0060]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```



รูปที่ 4.21 ผลการทดสอบคำสั่ง SEC ในหน้าต่างหน่วยความจำจำลองและหน้าต่างรีจีสเตอร์

4.8 กลุ่มคำสั่งควบคุม

เป็นกลุ่มคำสั่งที่ใช้ควบคุมการทำงานหลักของไมโครคอนโทรลเลอร์ ซึ่งจะเกี่ยวข้องกับการอินเตอร์รัปต์ และโหมดการทำงานประหยัดพลังงาน

การทดสอบ

ทดสอบคำสั่งจัดการกับรีจิสเตอร์เงื่อนไข ด้วยการป้อนโปรแกรมดังแสดงในรูปที่ 4.22 โดยที่โค้ดของโปรแกรมทั้งหมดเก็บที่หน่วยความจำตำแหน่ง 00H

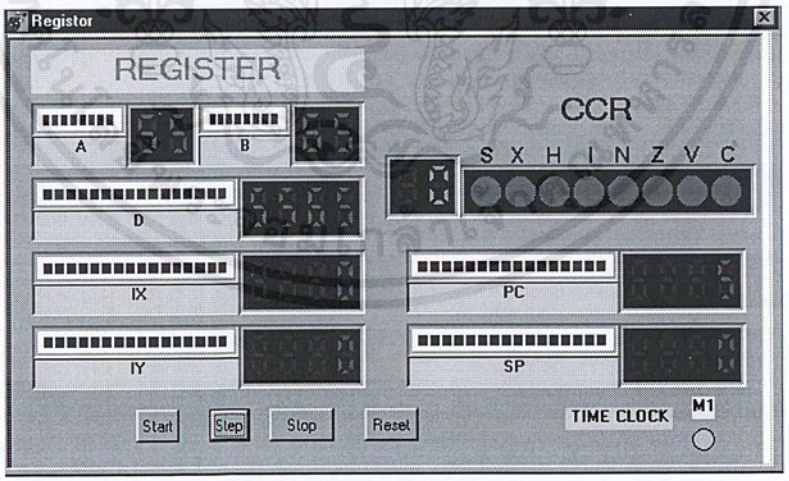
```
LDAA  #$55
LDAB  #$65
NOP
```

รูปที่ 4.22 โปรแกรมทดสอบคำสั่ง NOP

ผลการทดสอบคำสั่ง

ผลการทดสอบเมื่อรันด้วยโปรแกรม 68HC11 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.23

```
[0000]- 86 55 C6 65 01 00 00 00 00 00 00 00 00 00 00 00
[0010]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0020]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0030]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0040]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0050]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0060]- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```



รูปที่ 4.23 ผลการทดสอบคำสั่ง NOPในหน้าต่างหน่วยความจำจำลองและหน้าต่างรีจีสเตอร์

บทที่ 5
บทสรุป ปัญหา แนวทางแก้ไข และพัฒนา

5.1 บทสรุป

โครงงานนี้มีลักษณะเป็นโปรแกรมที่จำลองการทำงานตามคำสั่งของไมโครคอนโทรลเลอร์ 68HC11 ซึ่งเป็นการออกแบบโดยใช้การเขียนโปรแกรมในรูปแบบของวิชวล เพื่อให้สามารถทำการติดต่อกับผู้ใช้ได้ง่ายและมีการแสดงผลที่สวยงาม ลักษณะการจำลองการทำงานตามชุดคำสั่งแต่ละคำสั่งของไมโครคอนโทรลเลอร์ 68HC11 นั้น จะมีการแสดงค่าของรีจิสเตอร์แต่ละตัว ค่าของข้อมูลที่มีอยู่ในหน่วยความจำส่วนต่างๆ ตลอดจนสถานะการทำงานของคำสั่งตามโปรแกรมของไมโครคอนโทรลเลอร์ 68HC11

5.2 ปัญหา และแนวทางแก้ไข

ในการทำโครงงานมีปัญหาต่างๆ เกิดขึ้นและได้ทำการแก้ไขด้วยวิธีการต่างๆ จนทำให้สามารถที่จะจัดทำโครงงานได้สำเร็จตามวัตถุประสงค์ ซึ่งปัญหาและวิธีการที่ใช้ในการแก้ไขปัญหานั้นมีดังต่อไปนี้

1) **ปัญหา** แอสเซมเบลอร์ที่ใช้ในโปรแกรมเป็นแอสเซมเบลอร์ที่ทำงานบนดอส แต่ส่วนของโปรแกรมหลักใช้งานในระบบวินโดว์ ทำให้เมื่อทำการคอมไพล์ ตัวโปรแกรมหลักจะเรียกตัวแอสเซมเบลอร์ เมื่อทำการคอมไพล์เสร็จโปรแกรมจะเรียก List ไฟล์มาแสดง เพื่อให้ทราบว่ามีการ Error หรือไม่ แต่เนื่องจากโปรแกรมที่ทำงานบนวินโดว์จะทำงานเร็วกว่าโปรแกรมที่ทำงานบนดอส ดังนั้นเมื่อโปรแกรมอ่าน List ไฟล์มาแสดงจะไม่สามารถแสดง List ไฟล์ได้ เนื่องจากโปรแกรมที่ทำการคอมไพล์สร้าง List ไฟล์ไม่เสร็จดังนั้นเมื่อโปรแกรมบนวินโดว์ทำการอ่านจึงไม่เจอไฟล์

การแก้ไข ทำการหน่วงเวลา ขึ้นมาเพื่อรอให้โปรแกรมแอสเซมเบลอร์สร้าง List ไฟล์ให้เสร็จก่อนแล้วจึงค่อยทำการอ่านไฟล์ขึ้นมาแสดง แต่การหน่วงเวลาจะทำให้เกิดปัญหา เมื่อนำโปรแกรมไปใช้กับเครื่องที่มีความเร็วในการทำงานไม่เท่ากัน

2) **ปัญหา** แอสเซมเบลอร์ที่ใช้ในโปรแกรม เป็นแอสเซมเบลอร์ที่ทำงานบนดอสและไม่รองรับการแอสเซมเบลอร์ไฟล์ .ASM ที่มีชื่อยาวเกินกว่า 8 ตัวอักษรจึงทำให้ตัวแอสเซมเบลอร์ไม่สามารถสร้างไฟล์ .S19 และ .LST

การแก้ไข กำหนดรูปแบบการจัดเก็บชื่อไฟล์ได้ไม่เกิน 8 ตัวอักษร และใช้คำสั่งของดอสและดีบัก เพื่อทำการสร้างไฟล์ .S19 และ .LST

3) **ปัญหา** ออบเจ็กที่ใช้ในการแสดงผลของชุดอินเตอร์เฟสเป็นออบเจ็กที่เป็นมาตรฐานที่ให้มากับวิชวลเบสิก ทำให้การแสดงผลไม่สวยงาม และจัดรูปแบบของการแสดงผลได้ยาก

การแก้ไข จัดหาออบเจ็กแบบพิเศษมาทำการติดตั้งเพิ่มเติม

4) **ปัญหา** โปรแกรมวิชวลเบสิกไม่มีคำสั่งที่สนับสนุนการกระทำระดับบิต

การแก้ไข ทำการเขียนโปรแกรมติดต่อกับไฟล์ Bits32.dll ซึ่งเป็นไฟล์ที่มีผู้เขียนขึ้นไว้

5) **ปัญหา** ในการทดสอบในการออกแบบโครงสร้างในการทำงานของโปรแกรมครั้งแรก กำหนดให้การจำลองคำสั่งนั้นจะใช้ฐานข้อมูลเข้าใช้ร่วมในการค้น ซึ่งฐานข้อมูลจะทำการเก็บข้อมูลเกี่ยวกับข้อมูลของคำสั่ง เช่น ชื่อคำสั่ง รูปแบบของคำสั่ง ขนาดของคำสั่ง ซึ่งในการทดสอบนั้นสามารถที่ทำได้ โดยจำลองคำสั่งได้ เมื่อได้นำไปใช้ทดสอบกับเครื่องที่มีความเร็วต่ำ ก็จะทำให้การทำงานช้ามาก ซึ่งไม่เป็นไปตามลักษณะของการทำงานของไมโครคอนโทรลเลอร์ 68HC11 จึงทำให้การทำงานล่าช้า

การแก้ไข สร้างตัวแปรหน่วยความจำในการจำลองการเก็บข้อมูล

5.3 แนวทางการพัฒนาโครงงาน

แนวทางต่อไปที่ควรมีการพัฒนาโปรแกรมให้มีประสิทธิภาพสูงขึ้น มีดังนี้

1) ควรมีการเขียนแอสเซมเบลอขึ้นใช้งานเอง

2) ควรมีการเพิ่มชุดอินเตอร์เฟสที่สามารถติดต่อโดยผ่านทางพอร์ตได้

3) ควรปรับปรุงโครงสร้างของหน้าจอ และรูปแบบให้มีรูปแบบที่สวยงามมากขึ้น

ภาคผนวก

โปรแกรมจำลองการทำงานของไมโครคอนโทรลเลอร์ 68HC11

```
MEMO
Private Sub Form_Load()
    Me.Move 1441 * 3 + 20, 5500, 1441 * 4.65, Main.Height - 1200 '
ใส่ข้อมูลลง textbox
End Sub

'DISPLAY
Option Explicit
Global ResWin As Boolean
Global MemWin As Boolean
Global ResMove As Boolean
Global MemMove As Boolean
Global EditWin As Boolean
Global ToolWin As Boolean
Public ScaleM As Long
Public step As Boolean
Sub SetMenu_On()
  Main.mnuFileSave.Enabled = True
  Main.mnuFileSaveAs.Enabled = True
  Main.mnuPaste.Enabled = True
  Main.mnuCompile.Enabled = True
End Sub
Public Sub ShowRegis()
 Regisd = (RegisA * 256) + Regisb
  Regis.ledFlag(1).Value = Flag
  Regis.NumLEDflag.Value = ("&H" & Hex(Regis.ledFlag(1).Value))
  Regis.LEDPC.Value = PC
  Regis.NumLED8.AlphaNumeric = Hex(PC)
  Regis.LEDA.Value = RegisA
  Regis.NumLED3.AlphaNumeric = Hex(RegisA)
  Regis.LEDB.Value = Regisb
  Regis.NumLED2.AlphaNumeric = Hex(Regisb)
  Regis.LEDD.Value = Regisd
  Regis.NumLED4.AlphaNumeric = Hex(Regisd)
  Regis.LEDX.Value = Regisx
  Regis.NumLED5.AlphaNumeric = Hex(Regisx)
  Regis.LEDIY.Value = Regisy
  Regis.NumLED7.AlphaNumeric = Hex(Regisy)
  Regis.LEDSP.Value = SP
  Regis.NumLED6.AlphaNumeric = Hex(SP)
End Sub

Public Sub start()
 If Regis.Timer1.Interval = 0 Then
        If step = True Then
        Regis.ShPM1.FillColor = RGB(255, 0, 0) 'runstep
Else
        Regis.ShPM1.FillColor = RGB(0, 200, 255) 'runstep
        Regis.Refresh
        step = True
    End If
    Runtime
    ShowRegis
  Else
  Regis.Timer1.Enabled = True
End If
End Sub

'User Edit
Option Explicit
Type FormStatusType
  Deleted As Boolean '
  DirtyFlag As Boolean 'save status writed on edit
End Type
Global fState As FormStatusType 'save status file new
Global ASMFilename As String
Global strFind As String, strReplaced As String, inFindCase%
Global inFindDir%, CurPos%, FirstFindFlag As Boolean
Global getregis(6) As Long
Public Compileflag, Init, fstep As Boolean
Public Txtlist As String
Public InstionBYTE As Long
Public Line1 As Long

Sub FileNew()
Dim fIndex As Integer, NewNum%
  fState.DirtyFlag = False
  Main.Tag = Main.Tag + 1
  Editor.Caption = "Untitled" & Main.Tag
  Editor.Text1.Text = ""
  Editor.Show
  EditWin = True
End Sub
Sub OpenFileDialog()
  Dim strFilename As String
  On Error Resume Next
   ChDir App.Path & "\bin"
  Main!CommonDialog.DialogTitle = "Open"
  Main!CommonDialog.FileName = ""
  Main!CommonDialog.Flags = &H1& Or &H1000&
  Main!CommonDialog.Filter = "ASM Files(*.asm)|*.asm|All Files
(*.*)|*.*|"
```

```
Main!CommonDialog.Action = 1

  If Err <> 32755 Then    'User Press Cancal
    strFilename$ = Main!CommonDialog.FileName
    ASMFilename = Main!CommonDialog.FileTitle
    LoadTextFile strFilename$
  End If
End Sub
Sub LoadTextFile(strFilename$)
  Dim filenum%, fIndex%, txt$
  filenum% = FreeFile
  Open strFilename$ For Binary Access Read As filenum
  Screen.MousePointer = 11              'Set MousePointer to an
hourglass
  Editor.Caption = UCase$(strFilename)

  'Chang Form 's caption and display new text
  txt$ = Space$(LOF(filenum%))
  Get filenum%, 1, txt$
  Editor.Text1.Text = txt$
  fState.DirtyFlag = False
  EditWin = True
  Editor.Show
ErrReturn01:
  Close #filenum
  Screen.MousePointer = 0
  Exit Sub
End Sub
Function InListMenuFile(strFilename$) As Integer
  Dim i As Integer
  For i = 1 To 4
    If Main.mnuFileLastOpen(i%).Caption = strFilename$ Then
      InListMenuFile = True
      Exit Function
    End If
  Next i%
  InListMenuFile = False
End Function
Sub UpdateDynamicFileMenu(strFilename$)
  Dim i%, j%, str$, ShowFileFlag As Boolean
  If Not InListMenuFile(strFilename$) Then
    For i% = 4 To 2 Step -1
      'str = Mid(MainForm!mnuFileLastOpen(i% - 1).Caption, 3)
     Main!mnuFileLastOpen(i%).Caption = Main!mnuFileLastOpen
(i% - 1).Caption
    Next i%
ShowFileFlag = False
  For i% = 1 To 4
    If Main!mnuFileLastOpen(i%).Caption <> "" Then
      Main!mnuFileLastOpen(i%).Visible = True
      ShowFileFlag = True
    Else
      Main!mnuFileLastOpen(i%).Visible = False
    End If
  Next i%
  If ShowFileFlag Then Main!mnuFileLastOpen(0).Visible = True
End Sub
Function SaveFileAsDialog(strFilename$) As String
  On Error Resume Next
  Dim strContents As String
  Main!CommonDialog.DialogTitle = "Save AS"
  Main!CommonDialog.FileName = strFilename$
  Main!CommonDialog.Flags = &H2&
  Main!CommonDialog.Filter = "ASM Files(*.asm)|*.asm|Text Files
(*.txt)|(*.txt)"
  Main!CommonDialog.Action = 2
  If Err <> 32755 Then
    SaveFileAsDialog = Main!CommonDialog.FileName
    ASMFilename = Main!CommonDialog.FileTitle
  Else
    SaveFileAsDialog = ""
  End If

End Function
Sub SaveTextFile(strFilename$)
  Dim txt$, filenum%
  On Error GoTo ErrTrap02
  Screen.MousePointer = 11 ' กำหนดให้เมาส์เป็นรูปนาฬิกาทราย
  filenum% = FreeFile
  Open strFilename$ For Output As #FreeFile ' เปิดไฟล์เพื่อทำการ
เขียนข้อมูล
  txt$ = Editor.Text1.Text  'อ่านข้อมูลจาก Editor
  Print #filenum%, txt$  ' เขียนข้อมูลลงในไฟล์
  fState.DirtyFlag = False
  ErrReturn02:
  Close #filenum% ' ปิดไฟล์
  If Err = 0 Then
    Editor.Caption = UCase$(strFilename$)
  End If
  Screen.MousePointer = 0
 Exit Sub
 ErrTrap02:
```

```
Case 4: Resume
      Case 5: Resume Next
   End Select
End Sub
Sub EditCopy()
   Clipboard.SetText Editor.Text1.SelText
End Sub
Sub EditCut()
   Clipboard.SetText Editor.Text1.SelText
   Editor.Text1.SelText = ""
End Sub
Sub EditDelete()
   If Editor.Text1.SelLength = 0 Then
      Editor.Text1.SelLength = 1
      If Editor.Text1.SelLength <> 0 Then
         If Asc(Editor.Text1.SelText) = 13 Then
Editor.Text1.SelLength = 2
      End If
   End If
   Editor.Text1.SelText = ""
End Sub
Sub EditPaste()
   If Clipboard.GetFormat(1) Then
      Editor.Text1.SelText = Clipboard.GetText()
   End If
End Sub

'Editor
Private Sub Form_Load()
      Me.Move 0, 0, 1441 * 3, 4035
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, UnloadMode
As Integer)
   Dim strMsg As String
   Dim strFilename As String
   Dim intResponse As Integer

   ' Check to see if the text has been changed.
   If fState.DirtyFlag Then
      strFilename = Editor.Caption
      strMsg = "The text in " & strFilename & " has changed."
      strMsg = strMsg & vbCrLf
      strMsg = strMsg & "Do you want to save the changes?"
      intResponse = MsgBox(strMsg, 51, Main.Caption)
      Select Case intResponse
If UCase$(Left$(Editor.Caption, 8)) = "UNTITLED" Then
            strFilename$ = SaveFileAsDialog(CStr
(Editor.Caption))
         Else
            strFilename$ = Editor.Caption
         End If
         If strFilename$ <> "" Then SaveTextFile strFilename$
         EditWin = False
      ' Call the save procedure. If strFilename = Empty, then
      ' the user chose Cancel in the Save As dialog box; otherwise,
      ' save the file.
      Case 7      ' User chose No. Unload the file.
         Cancel = False
         EditWin = False
      Case 2      ' User chose Cancel. Cancel the unload.
         Cancel = True
      End Select
   End If
   'Setmenu_Off
End Sub

Private Sub Form_Resize()
   If WindowState <> 1 Then
      Text1.Move 0, 0, Me.ScaleWidth, Me.ScaleHeight
   End If
End Sub

Private Sub Text1_KeyPress(KeyAscii As Integer)
   'fState.DirtyFlag = True
End Sub
```

```
Display
Dim temp$
Private Sub Form_Load()
  Me.Enabled = True
 Me.Move 0, 4035, 1441 * 3, (Main.Height - 1441)
  List1.Height = ListPROG.Height - 400
  List1.Width = ListPROG.Width - 100
  'List1.Selected(0) = True
 ' Me.Refresh
End Sub

Private Sub Form_Resize()
  If ListPROG.Height > 400 And ListPROG.Width > 100 Then
  List1.Height = ListPROG.Height - 400
  List1.Width = ListPROG.Width - 100

 'List1.ListCount
  End If
End Sub

Main
Option Explicit
Dim i As Long
Private Sub mnclearMEM_Click()
      Call clearRAM
End Sub

Private Sub mnuAbout_Click()
'frmAbout.Show
End Sub

Private Sub mnuCascade_Click()
   Main.Arrange vbCascade
End Sub

Private Sub mnuCompile_Click()
    Dim i As Integer
   Dim strMsg As String
   Dim strFilename As String
   Dim intResponse As Integer
   ins = 0
   PC = 0
   SP = 255
   RegisA = 0
   Regisb = 0
   Regisd = 0
'Regis.CmdRESET_Click
  '*
 ' For i = 0 To 6
  'getregis(i) = 0
  'Next i
     If EditWin = True Then       ' Check to see if the text has been
changed.
      If fState.DirtyFlag Then
        strFilename = Editor.Caption
        strMsg = "มีการเปลี่ยนแปลงใน " & strFilename & ""
        strMsg = strMsg & vbCrLf
        strMsg = strMsg & "คุณต้องการบันทึกการเปลี่ยนแปลงนี้หรือ
ไม่?"
        intResponse = MsgBox(strMsg, 51, Main.Caption)
        Select Case intResponse
          Case 6     ' User chose Yes.
           ' The file hasn't been saved yet.
           If UCase$(Left$(Editor.Caption, 8)) = "UNTITLED"
Then
             strFilename$ = SaveFileAsDialog(CStr
(Editor.Caption))
           Else
             strFilename$ = Editor.Caption
           End If
           If strFilename$ <> "" Then SaveTextFile strFilename$
          Case 7     ' User chose No. Unload the file.
           OpenFileDialog
          Case 2     ' User chose Cancel. Cancel the unload.
            Exit Sub
        End Select
      End If
     End If
    'OpenFileDialog
    SetMenu_On
    Main.Toolbar1.Buttons(9).Enabled = True
     ListPROG.List1.Clear
     For i = 1 To 6
     Memo.txtmem(i).Text = ""
     Next i
      Compile_asm
       Main.Toolbar1.Buttons(10).Enabled = True 'start
       Main.Toolbar1.Buttons(11).Enabled = True 'step
End Sub

Private Sub mnuCopy_Click()
   Editor.Visible = False
```

```
Private Sub mnuCut_Click()
   Editor.Visible = False
   EditCut
   Editor.Visible = True
End Sub
Private Sub mnuDelete_Click()
   Editor.Visible = False
   EditDelete
   Editor.Visible = True
End Sub
Private Sub mnuEdit_Click()
   If EditWin = True Then
    ' If Main.ActiveControl <> "" Then
        Main.mnuCut.Enabled = True
        Main.mnuCopy.Enabled = True
        Main.mnuDelete.Enabled = True
      Else
        Main.mnuCut.Enabled = False
        Main.mnuCopy.Enabled = False
        Main.mnuDelete.Enabled = False
    ' End If
   End If
End Sub
Private Sub mnuexit_Click()
   Unload Me
End Sub
Private Sub mnuFileNew_Click()
   Dim strMsg As String
   Dim strFilename As String
   Dim intResponse As Integer
   If EditWin = True Then
     ' Check to see if the text has been changed.
     If fState.DirtyFlag Then
       strFilename = Editor.Caption
       strMsg = "มีการเปลี่ยนแปลงใน " & strFilename & " "
       strMsg = strMsg & vbCrLf
       strMsg = strMsg & "คุณต้องการบันทึกการเปลี่ยนแปลงนี้หรือ
ไม่?"
       intResponse = MsgBox(strMsg, 51, Main.Caption)
       Select Case intResponse
         Case 6     ' User chose Yes.
           ' The file hasn't been saved yet.
           If UCase$(Left$(Editor.Caption, 8)) = "UNTITLED"
Then
             strFilename$ = SaveFileAsDialog(CStr
(Editor.Caption))
If strFilename$ <> "" Then SaveTextFile strFilename$
             Exit Sub
         End Select
       End If
   End If
   FileNew
   SetMenu_On
   Main.Toolbar1.Buttons(9).Enabled = True
End Sub
Private Sub mnuFileOpen_Click()
   Dim strMsg As String
   Dim strFilename As String
   Dim intResponse As Integer

   If EditWin = True Then      ' Check to see if the text has been
changed.
     If fState.DirtyFlag Then
       strFilename = Editor.Caption
       strMsg = "มีการเปลี่ยนแปลงใน " & strFilename & ""
       strMsg = strMsg & vbCrLf
       strMsg = strMsg & "คุณต้องการบันทึกการเปลี่ยนแปลงนี้หรือ
ไม่?"
       intResponse = MsgBox(strMsg, 51, Main.Caption)
       Select Case intResponse
         Case 6     ' User chose Yes.
           If UCase$(Left$(Editor.Caption, 8)) = "UNTITLED"
Then
             strFilename$ = SaveFileAsDialog(CStr
(Editor.Caption))
           Else
             strFilename$ = Editor.Caption
           End If
           If strFilename$ <> "" Then SaveTextFile strFilename$
         Case 7     ' User chose No. Unload the file.
           OpenFileDialog
         Case 2     ' User chose Cancel. Cancel the unload.
           Exit Sub
       End Select
     End If
   End If
   OpenFileDialog
   SetMenu_On
   Main.Toolbar1.Buttons(9).Enabled = True
End Sub

Private Sub mnuFileSave_Click()
```

```
Else
        strFilename$ = Editor.Caption
      End If
      If strFilename$ <> "" Then SaveTextFile strFilename$
      Main.Toolbar1.Buttons(9).Enabled = True
End Sub

Private Sub mnuFileSaveAs_Click()
   Dim strFilename$
     strFilename$ =
SaveFileAsDialog(CStr(Main.ActiveForm.Caption))
   If strFilename <> "" Then
     SaveTextFile strFilename$
   End If
   Main.Toolbar1.Buttons(9).Enabled = True
End Sub
Private Sub mnuHorizontal_Click()
  Main.Arrange vbTileHorizontal
End Sub

Private Sub mnuMem_Click()
     Memo.Show
   MemWin = True
End Sub

Private Sub mnuPaste_Click()
   EditPaste
End Sub
Private Sub mnuReg_Click()
          Regis.Show
End Sub

Private Sub mnuReset_Click()
  'Call INtitial
  Regis.CmdRESET_Click
   If PC = 0 Then ListPROG.List1.Selected(0) = True
   Regis.Timer1.Enabled = False
   Main.mnuStart.Enabled = True
   Main.mnuSTEP.Enabled = True
   Main.mnuStop.Enabled = False
   Main.mnuCompile.Enabled = True
   Main.Toolbar1.Buttons(10).Enabled = True 'start
   Main.Toolbar1.Buttons(11).Enabled = True 'step
   Main.Toolbar1.Buttons(12).Enabled = False 'stop
End Sub

Private Sub mnuStart_Click()
     Call start
     Main.mnuSTEP.Enabled = False
     Main.mnuStop.Enabled = True
     Main.Toolbar1.Buttons(11).Enabled = False 'step
     Main.Toolbar1.Buttons(12).Enabled = True 'stop
End Sub

Private Sub mnustep_Click()
     Main.mnuStart.Enabled = False
     Main.mnuSTEP.Enabled = True
     Main.mnuStop.Enabled = True
     Main.Toolbar1.Buttons(10).Enabled = False 'start
     Main.Toolbar1.Buttons(11).Enabled = True 'step
     Main.Toolbar1.Buttons(12).Enabled = True 'stop
End Sub

Private Sub mnuSTEP_mortor_Click()
StepMT.Show
End Sub

Private Sub mnuStop_Click()
     Regis.Timer1.Enabled = False
     Main.mnuStart.Enabled = True
     Main.mnuSTEP.Enabled = True
     Main.mnuStop.Enabled = False
     Main.mnuReset.Enabled = True
     Main.Toolbar1.Buttons(10).Enabled = True 'start
     Main.Toolbar1.Buttons(11).Enabled = True 'step
     Main.Toolbar1.Buttons(12).Enabled = False 'stop
     Main.Toolbar1.Buttons(13).Enabled = True 'reset
End Sub

Private Sub mnuTopic_Click()
'helpfrm.Show
End Sub

Private Sub MnuViewLIST_Click()
ListPROG.Show
End Sub

Private Sub Toolbar1_ButtonClick(ByVal Button As
MSComctlLib.Button)
  Dim countnum As Integer
   Select Case Button.Index
     Case 1: mnuFileNew_Click 'new
```

```
Case 9: mnuCompile_Click 'compile
    Case 10: mnuStart_Click 'start
    Case 11: mnustep_Click 'step
    Case 12: mnuStop_Click 'stop
    Case 13: mnuReset_Click 'reset
  End Select
End Sub

Compile
Global ss As Integer
Public Const BIT_COPYRIGHT_DEFAULT = 0
Public Const BIT_COPYRIGHT_CAPITAL = 1
Public Const BIT_COPYRIGHT_SMALL = 2
' Unprintable string filter.
Public Const BIT_UNSTR_DEFAULT = 1
Public Const BIT_UNSTR_CTRL = 1
Public Const BIT_UNSTR_EXTD = 2
Public Const BIT_UNSTR_CTRLANDEXTD = 3
' BIT32.DLL API Functions declaration.
' ---------- BYTE SIZE BIT MANIPULATION ----------
Public Declare Sub vbrcGetBitB Lib "Bits32.dll" _
  (ByVal ByteVal As Byte, ByVal BitNum As Integer)
Public Declare Sub vbrcResetBitB Lib "Bits32.dll" _
  (ByteVal As Byte, ByVal BitNum As Integer)
Public Declare Sub vbrcROLB Lib "Bits32.dll" _
  (ByteVal As Byte, ByVal BitNum As Integer)
Public Declare Sub vbrcRORB Lib "Bits32.dll" _
  (ByteVal As Byte, ByVal BitNum As Integer)
Public Declare Sub vbrcSetBitB Lib "Bits32.dll" _
  (ByteVal As Byte, ByVal BitNum As Integer)
Public Declare Sub vbrcSHLB Lib "Bits32.dll" _
  (ByteVal As Byte, ByVal BitNum As Integer)
Public Declare Sub vbrcSHRB Lib "Bits32.dll" _
  (ByteVal As Byte, ByVal BitNum As Integer)
Public Declare Sub vbrcToggleBitB Lib "Bits32.dll" _
  (ByteVal As Byte, ByVal BitNum As Integer)
' ---------- WORD SIZE BIT MANIPULATION ----------
Public Declare Function vbrcBinStrW Lib "Bits32.dll" _
  (ByVal WordVal As Integer) As String
Public Declare Function vbrcGetBitW Lib "Bits32.dll" _
  (ByVal WordVal As Integer, ByVal BitNum As Integer) As Integer
Public Declare Function vbrcSwapBitW Lib "Bits32.dll" _
  (ByVal WordVal As Integer) As Integer
Public Declare Sub vbrcResetBitW Lib "Bits32.dll" _
  (WordVal As Integer, ByVal BitNum As Integer)
Public Declare Sub vbrcROLW Lib "Bits32.dll" _
Public Declare Sub vbrcRORW Lib "Bits32.dll" _
  (WordVal As Integer, ByVal BitNum As Integer)
Public Declare Sub vbrcSetBitW Lib "Bits32.dll" _
  (WordVal As Integer, ByVal BitNum As Integer)
Public Declare Sub vbrcSHLW Lib "Bits32.dll" _
  (WordVal As Integer, ByVal BitNum As Integer)
Public Declare Sub vbrcSHRW Lib "Bits32.dll" _
  (WordVal As Integer, ByVal BitNum As Integer)
Public Declare Sub vbrcToggleBitW Lib "Bits32.dll" _
  (WordVal As Integer, ByVal BitNum As Integer)
Public Declare Function vbrcBinStrD Lib "Bits32.dll" _
  (ByVal DWordVal As Long) As String
Public Declare Function vbrcGetBitD Lib "Bits32.dll" _
  (ByVal DWordVal As Long, ByVal BitNum As Integer) As Integer
Public Declare Function vbrcSwapBitD Lib "Bits32.dll" _
  (ByVal DWordVal As Long) As Long
Public Declare Sub vbrcResetBitD Lib "Bits32.dll" _
  (DWordVal As Long, ByVal BitNum As Integer)
Public Declare Sub vbrcROLD Lib "Bits32.dll" _
  (DWordVal As Long, ByVal BitNum As Integer)
Public Declare Sub vbrcRORD Lib "Bits32.dll" _
  (DWordVal As Long, ByVal BitNum As Integer)
Public Declare Sub vbrcSetBitD Lib "Bits32.dll" _
  (DWordVal As Long, ByVal BitNum As Integer)
Public Declare Sub vbrcSHLD Lib "Bits32.dll" _
  (DWordVal As Long, ByVal BitNum As Integer)
Public Declare Sub vbrcSHRD Lib "Bits32.dll" _
  (DWordVal As Long, ByVal BitNum As Integer)
Public Declare Sub vbrcToggleBitD Lib "Bits32.dll" _
  (DWordVal As Long, ByVal BitNum As Integer)
Public Flag As Byte
Public MEM(65535) As Long
Public RAM(65535) As Long     'ตัวแปรกำหนดหน่วยความจำ RAM
64 Kbyte
Public startaddr As Long
Public addram As Long
Dim IR As Byte
Dim IdentifyLen As Byte
Global instrData As Byte              'ตัวแปรข้อมูลที่กำหนดข้อ
มูลจากอ่านในคำสั่ง
Global InstrCounter As Long           'ตัวแปรที่กำหนดจำนวนไบต์
ของชุดคำสั่ง
Public LINEnum(255) As Long
Global PC As Long
Global mybte As Long
Global PCH As Long
```

```
Global IXL As Long
Global IYH As Long
Global IYL As Long
Global OPCODE As String
Global OPERAND As String
Global Data As Long
Global SimPrC As Long
Global Caseregis As Long
Global RegisA As Long
Global a(8) As Long
Global B(8) As Long
Global IX(8) As Long
Global IY(8) As Long
Global aSP(8) As Long
Global aPC(8) As Long
Global AA As Long
Global Regisb As Long
Global Regisd As Long
Global Regisx As Long
Public Stoped As Boolean
Global Regisy As Long
Global ADR As Long
Global SP As Long
Global n As Long
Global DH As Long
Global DL As Long
Global ins As Long

Sub Compile_asm()
   Dim strBuf As String
   Dim a As Long
   Dim L As Long
   Dim B As Long
   Dim BUF As String
   Dim intxt As Long
   Dim STARTBYTE As Long
   Dim LASTBYTE As Long
   Dim MYFSO As New FileSystemObject
   Dim TXTSTREAM As TextStream
   Dim txtfile As File
   Dim tt As Long '*
   Dim CountCodea, CountCodeb, strPath, instrByte, CountCode, ttt
As String, myadd
   Dim filenum%, fIndex%, txt$, i, j, n As Integer
   Dim inpd, Inputdata, LSTchk As String, mybyt, myby, MyByte,
bytestr, CountByte As Integer
Dim delaytime, AdRes As Long
   Dim ErrorAsm, ErrorAsmCk As Integer
   Dim filnum As Integer
   Stoped = False
   SP = 255
   PC = 0
   Flag = 0
   ins = 0
      Compileflag = False
      ShowRegis
      ChDir App.Path & "\bin\"
      If Dir(App.Path & ("\bin\BUF.ASM")) <> "" Then
        Kill (App.Path & "\bin\BUF.ASM")
      End If
      If Dir(App.Path & ("\bin\BUF.s19")) <> "" Then
        Kill (App.Path & "\bin\buf.s19")
      End If
      ChDir App.Path & "\bin\" 'ไดเร็กทอรี่ปัจจุบันที่ as11.exe อยู่
        filenum% = FreeFile
      Open App.Path & "\bin\BUF.ASM" For Output As filenum%
        txt$ = Editor.Text1.Text  'อ่านข้อมูลจาก Editor
        Print #filenum%, txt$  ' เขียนข้อมูลลงในไฟล์
      Close #filenum% ' ปิด

      While Dir(App.Path & ("\bin\buf.asm")) = ""
      Wend
      ChDir App.Path & "\bin\"
      Call Shell(App.Path & "\bin\command.com /c AS11.exe buf.asm
-l > buf.lst", vbHide)
      ' รันคอมไพเลอร์as11
      ' เปิดไฟล์ List ที่เกิดจากการคอมไพเลอร์
      Dim ii As Long
      For ii = 0 To 255
      LINEnum(ii) = 0
       Next ii
       ii = 0
      filenum% = FreeFile  'กำหนดหมายเลข Handdle ของไฟล์
      Screen.MousePointer = 0  'กำหนดเมาส์ให้อยู่ในสภาวะปกติ
      For j = 0 To 65535
         RAM(j) = 0
         MEM(j) = 0
       Next
     While Dir(App.Path & ("\bin\buf.lst")) = ""
     Wend
      Open App.Path & "\bin\buf.lst" For Input As #1 'เปิดไฟล์
      ErrorAsm = 0
```

```
Line Input #1, LSTchk                    ' อ่านข้อมูลเป็นบรรทัด
      If ii > 2 Then
      sw$ = Mid(LSTchk, 1, 10)
      sa$ = Mid(LSTchk, 1, 3)
    If ((sw$ = "buf.asm") Or (sa$ = "C:\")) Then
       ErrorAsm = ErrorAsm + 1
       ErrorAsmCk = MsgBox("โปรแกรมแอสแซมบลี้มีข้อผิดพลาด
กรุณาแก้ไขโปรแกรมของท่าน", vbExclamation, "Error On Assembly
Program")
     End If
     End If
     ii = ii + 1
     Loop
Close #1
   If ErrorAsm = 0 Then
      ii = 0
       ErrorAsmCk = MsgBox("..ผ่าน..สามารถสั่ง RUN ได้",
vbOKOnly, "Compile Pass")
       ListPROG.Enabled = True
       Memo.Enabled = True
       Regis.Enabled = True
   Open App.Path & "\bin\buf.lst" For Input As #1 'เปิดไฟล์
    ErrorAsm = 0
      Do While Not EOF(1)                ' ตรวจสอบ
ตำแหน่งสิ้นสุดไฟล์
      Line Input #1, LSTchk              ' อ่านข้อมูลเป็นบรรทัด
      If ii > 2 Then
      ListPROG.List1.AddItem LSTchk
      LINEnum(ii) = Val("&H" & Mid(LSTchk, 8, 4))
     End If
     ii = ii + 1
     Loop
     filenum = FreeFile
     DoEvents
     InstrCounter = 0
     While Dir(App.Path & "\bin\buf.s19") = ""
     Wend
     Open App.Path & "\bin\buf.s19" For Input As filenum        '
เปิดไฟล์ S19
     Do While Not EOF(filenum)          ' ตรวจสอบ
ตำแหน่งสิ้นสุดไฟล์
       Line Input #filenum, Inputdata  'inpd = Inputdata ' อ่านข้อมูล
เป็นบรรทัด
       CountCode = Mid(Inputdata, 3, 2)
       MyByte = Val(" &H" & CountCode) - 3 ' จำนวน Byte ของคำ
สั่ง แปลงจาก Text เป็นตัวเลข
mybyt = Val(" &H" & CountCodea) * 256
       CountCodeb = (Mid(Inputdata, 7, 2))
       myby = Val(" &H" & CountCodeb)
       myadd = mybyt + myby

       If MyByte <> 0 Then 'ถ้ามีจำนวนคำสั่งไม่เป็นศูนย์
         mybte = MyByte
         InstrCounter = InstrCounter + MyByte
         bytestr = 9 '(MYADDRESS + InstrCounter - MyByte)
       For j = myadd To (myadd + InstrCounter - 1)  'อ่านเท่ากับ
จำนวนของคำสั่ง
           startaddr = myadd
           RAM(j) = Val(" &H" & Mid(Inputdata, bytestr, 2)) 'คำสั่ง
1 Byte
           MEM(j) = RAM(j)
           bytestr = bytestr + 2
       Next
     End If
     Loop
Close filenum  'ปิดไฟล์
For i = 1 To 6
Select Case i
     Case 1: STARTBYTE = 0
          L = 15
     Case 2: STARTBYTE = 1892
          L = 127
     Case 3:  STARTBYTE = 40960
          L = 255
     Case 4:  STARTBYTE = 46592
          L = 31
     Case 5:  STARTBYTE = 49152
          L = 511
     Case 6:  STARTBYTE = 57344
          L = 511
End Select
  For j = 0 To L
  BUF = ""
       For a = 0 To 15
            BUF = BUF & "  " & Right(String(2, "0") & Hex(RAM
(STARTBYTE + a)), 2)
       Next a
          strBuf = strBuf & "[" & Right(String(3, "0") & Hex
(STARTBYTE), 4) & "]-" & BUF & Chr(13) & Chr(10)
          STARTBYTE = STARTBYTE + 16
Next j
  Memo.txtmem.Item(i).Text = strBuf
```

```
Main.mnuStart.Enabled = True
        Main.mnuSTEP.Enabled = True
        Compileflag = True
ListPROG.List1.Selected(0) = True
Close #1
End If

 End Sub'คอมไพล์

Sub FetchIR() 'แยกออบโค้ดกับ โอเปอร์แรน
'If stoped = False Then
OPCODE = Right("00" & Hex(RAM(startaddr + PC)), 2)
Select Case OPCODE
'กลุ่มคำสั่ง 1 ไบต์
'**************ABA************************
Case "1B"
        PC = PC + 1
        n = RegisA
        RegisA = Regisb + RegisA
 If vbrcGetBitD(n, 3) And vbrcGetBitD(Regisb, 3) Or vbrcGetBitD
(Regisb, 3) And Not vbrcGetBitD(RegisA, 3) Or Not vbrcGetBitD
(RegisA, 3) And vbrcGetBitD(n, 3) = 1 Then
   vbrcSetBitB Flag, 5
 Else
   vbrcResetBitB Flag, 5
 End If
 If vbrcGetBitD(RegisA, 7) Then
    vbrcSetBitB Flag, 3
 Else
    vbrcResetBitB Flag, 3
 End If
 cc = 1
                For i = 0 To 7
                    cc = (Not vbrcGetBitD(RegisA, i)) And cc
                Next i
                If cc = 0 Then
                   vbrcResetBitB Flag, 2
                Else
                    vbrcSetBitB Flag, 2
                End If
 If vbrcGetBitD(n, 7) And vbrcGetBitD(Regisb, 7) And Not
vbrcGetBitD(RegisA, 7) Or Not vbrcGetBitD(n, 7) And Not
vbrcGetBitD(Regisb, 7) And vbrcGetBitD(RegisA, 7) = 1 Then
   vbrcSetBitB Flag, 1
 Else
   vbrcResetBitB Flag, 1
If vbrcGetBitD(n, 7) And vbrcGetBitD(Regisb, 7) Or vbrcGetBitD
(Regisb, 7) And Not vbrcGetBitD(RegisA, 7) Or Not vbrcGetBitD
(RegisA, 7) And vbrcGetBitD(RegisA, 7) = 1 Then
   vbrcSetBitB Flag, 0
 Else
   vbrcResetBitB Flag, 0
 End If

' คำสั่ง ABX
Case "3A": PC = PC + 1
      Regisx = Regisx + Regisb

' คำสั่ง LSRD
 Case "04": PC = PC + 1
 n = Regisd
 Call vbrcSHLD(Regisd, 1)
 vbrcResetBitB Flag, 3
 cc = 1
 For i = 0 To 15
    cc = (Not vbrcGetBitD(Regisd, i)) And cc
 Next i
 If cc = 0 Then
  vbrcResetBitB Flag, 2
 Else
  vbrcSetBitB Flag, 2
 End If
 If vbrcGetBitD(Regisd, 0) Then
    vbrcSetBitB Flag, 1
 Else
    vbrcResetBitB Flag, 1
 End If
 If vbrcGetBitD(n, 0) = 1 Then
    vbrcSetBitB Flag, 0
Else
    vbrcResetBitB Flag, 0
End If
              If Regisd > 255 Then
                RegisA = (Regisd / 256)
                Regisb = Regisd - (RegisA * 255)
              Else
                RegisA = 0
                Regisb = Regisd
              End If

 '**************ASLA ************************
Case "48": PC = PC + 1
```

```
If n Mod 2 Then
vbrcSetBitB Flag, 0
Else
vbrcResetBitB Flag, 0
End If
If vbrcGetBitD(RegisA, 7) = 0 Then
 vbrcResetBitB Flag, 3
Else
  vbrcSetBitB Flag, 3
 End If
                cc = 1
                For i = 0 To 7
                    cc = (Not vbrcGetBitD(RegisA, i)) And cc
                Next i
                If cc = 0 Then
                   vbrcResetBitB Flag, 2
                Else
                   vbrcSetBitB Flag, 2
                End If
               If vbrcGetBitD(Flag, 3) Xor vbrcGetBitD(Flag, 0) = 1
Then
                vbrcSetBitB Flag, 1
                Else
                vbrcResetBitB Flag, 1
                End If

 '**************ASL B*************************
 Case "58": PC = PC + 1
 n = Regisb
 Regisb = Regisb * 2
 If n Mod 2 Then
vbrcSetBitB Flag, 0
Else
vbrcResetBitB Flag, 0
End If
 If vbrcGetBitD(Regisb, 7) = 0 Then
  vbrcResetBitB Flag, 3
Else
  vbrcSetBitB Flag, 3
 End If
                cc = 1
                For i = 0 To 7
                    cc = (Not vbrcGetBitD(Regisb, i)) And cc
                Next i
                If cc = 0 Then
                   vbrcResetBitB Flag, 2
End If
                If vbrcGetBitD(Flag, 3) Xor vbrcGetBitD(Flag, 0) = 1
Then
                vbrcSetBitB Flag, 1
                Else
                vbrcResetBitB Flag, 1
                End If
 '**************ASL D*************************
 Case "05": PC = PC + 1
 n = Regisd
 Call vbrcSHRD(Regisd, 1)
          If vbrcGetBitD(n, 15) = 1 Then
             vbrcSetBitB Flag, 0
           Else
             vbrcResetBitB Flag, 0
           End If
         If vbrcGetBitD(Regisd, 15) = 1 Then
             vbrcSetBitB Flag, 3
           Else
             vbrcResetBitB Flag, 3
           End If

           cc = 1
            For i = 0 To 15
                cc = (Not vbrcGetBitD(Regisd, i)) And cc
            Next i
            If cc = 0 Then
               vbrcResetBitB Flag, 2
            Else
               vbrcSetBitB Flag, 2
             End If
            If vbrcGetBitD(Flag, 3) Xor vbrcGetBitD(Flag, 0) = 1
Then
            vbrcSetBitB Flag, 1
            Else
            vbrcResetBitB Flag, 1
            End If
            If Regisd > 255 Then
             RegisA = (Regisd / 256)
             Regisb = Regisd - (RegisA * 256)
           Else
             RegisA = 0
             Regisb = Regisd
           End If
 '***********ASLA*************************
 Case "47": PC = PC + 1
```

```
If (Data Mod 2) Then
        vbrcSetBitB Flag, 0
      Else
        vbrcResetBitB Flag, 0
      End If
      If vbrcGetBitD(RegisA, 7) = 0 Then
        vbrcResetBitB Flag, 3
      Else
        vbrcSetBitB Flag, 3
      End If
        cc = 1
        For i = 0 To 7
          cc = (Not vbrcGetBitD(RegisA, i)) And cc
        Next i
        If cc = 0 Then
          vbrcResetBitB Flag, 2
        Else
          vbrcSetBitB Flag, 2
        End If
        If vbrcGetBitD(Flag, 3) Xor vbrcGetBitD(Flag, 0) = 1 Then
          vbrcSetBitB Flag, 1
        Else
          vbrcResetBitB Flag, 1
        End If
'************ASR B*********************
Case "57": PC = PC + 1
        Data = Regisb
      Regisb = Data / 2
      If (Data Mod 2) Then
        vbrcSetBitB Flag, 0
      Else
        vbrcResetBitB Flag, 0
      End If
      If vbrcGetBitD(Regisb, 7) = 0 Then
        vbrcResetBitB Flag, 3
      Else
        vbrcSetBitB Flag, 3
      End If
        cc = 1
        For i = 0 To 7
          cc = (Not vbrcGetBitD(Regisb, i)) And cc
        Next i
        If cc = 0 Then
          vbrcResetBitB Flag, 2
        Else
          vbrcSetBitB Flag, 2
vbrcSetBitB Flag, 1
        Else
          vbrcResetBitB Flag, 1
        End If
  '**************CBA*************************
 Case "11":
 PC = PC + 1
result% = RegisA - Regisb
 If vbrcGetBitD(result%, 7) Then
    vbrcSetBitB Flag, 3
 Else
    vbrcResetBitB Flag, 3
 End If
 cc = 1
          For i = 0 To 7
              cc = (Not vbrcGetBitD(result%, i)) And cc
          Next i
          If cc = 0 Then
            vbrcResetBitB Flag, 2
          Else
            vbrcSetBitB Flag, 2
          End If
 If vbrcGetBitD(RegisA, 7) And (Not vbrcGetBitD(Regisb, 7)) And
 (Not vbrcGetBitD(result%, 7)) Or (Not vbrcGetBitD(RegisA, 7)) And
 vbrcGetBitD(Regisb, 7) And vbrcGetBitD(result%, 7) = 1 Then
 vbrcSetBitB Flag, 1
 Else
 vbrcResetBitB Flag, 1
 End If
 If (Not vbrcGetBitD(RegisA, 7)) And vbrcGetBitD(Regisb, 7) Or
 vbrcGetBitD(Regisb, 7) And vbrcGetBitD(result%, 7) Or
 vbrcGetBitD(result%, 7) And (Not vbrcGetBitD(RegisA, 7)) = 1 Then
 vbrcSetBitB Flag, 0
 Else
 vbrcResetBitB Flag, 0
 End If
 '****คำสั่ง CLC
 Case "0C": PC = PC + 1
 vbrcResetBitB Flag, 0
  '************CLEAR I FLAG*************
 Case "0E": PC = PC + 1
 vbrcResetBitB Flag, 4
 '************CLEAR REG A*************
 Case "4F": PC = PC + 1
 RegisA = 0
vbrcSetBitB Flag, 2
```

```
vbrcResetBitB Flag, 3
'*************CLEAR REG B*************
Case "5F"
PC = PC + 1
Regisb = 0
vbrcSetBitB Flag, 2
vbrcResetBitB Flag, 0
vbrcResetBitB Flag, 1
vbrcResetBitB Flag, 3
' คำสั่ง CLV
Case "0A": PC = PC + 1
vbrcResetBitB Flag, 1
'คำสั่ง COMA
Case "43"
PC = PC + 1
RegisA = &HFF - RegisA
If vbrcGetBitD(RegisA, 7) = 0 Then
 vbrcResetBitB Flag, 3
Else
  vbrcSetBitB Flag, 3
End If
              cc = 1
              For i = 0 To 7
                  cc = (Not vbrcGetBitD(RegisA, i)) And cc
              Next i
              If cc = 0 Then
                 vbrcResetBitB Flag, 2
              Else
                 vbrcSetBitB Flag, 2
               End If
    vbrcResetBitB Flag, 1
    vbrcSetBitB Flag, 0
'****คำสั่ง COMB*******
Case "53"
PC = PC + 1
st$ = Right("00" & Hex(RAM(startaddr + PC)), 2)
Regisb = &HFF - Regisb
If vbrcGetBitD(Regisb, 7) = 0 Then
 vbrcResetBitB Flag, 3
Else
  vbrcSetBitB Flag, 3
End If
              cc = 1
              For i = 0 To 7
                  cc = (Not vbrcGetBitD(Regisb, i)) And cc
              Next i
Else
                 vbrcSetBitB Flag, 2
               End If
    vbrcResetBitB Flag, 1
    vbrcSetBitB Flag, 0
'**************DAA*************************
Case "19":
PC = PC + 1
S$ = Right("00" & Hex(RegisA), 2)
st$ = S$
DH = Val("&H" + Mid(S$, 1, 1))
DL = Val("&H" + Mid(st$, 2, 2))
If vbrcGetBitD(Flag, 0) Then
  If (DH <= 2) And (vbrcGetBitD(Flag, 5) = 0) Then
     If (DL <= 9) Then
     RegisA = RegisA + Hex(60)
      vbrcSetBitB Flag, 0
     Else
     RegisA = RegisA + Hex(66)
     vbrcSetBitB Flag, 0
     End If
  End If
  If (DH <= 3) And (vbrcGetBitD(Flag, 5) = 1) And (DL <= 3) Then
    RegisA = RegisA + Hex(66)
     vbrcSetBitB Flag, 0
  End If
Else 'carry flag เป็น 0
  If (DH <= 9) And (vbrcGetBitD(Flag, 5) = 0) And (DL <= 9) Then
     RegisA = RegisA
     vbrcResetBitB Flag, 0
  End If
  If (DH <= 9) And (vbrcGetBitD(Flag, 5)) And (DL <= 3) Then
     RegisA = RegisA + 6
     vbrcResetBitB Flag, 0
 End If
 If (DH <= 8) And (vbrcGetBitD(Flag, 5) = 0) And (DL >= 10) Then
     RegisA = RegisA + 6
     vbrcResetBitB Flag, 0
End If
If (DH >= 10) And (vbrcGetBitD(Flag, 5) = 0) And (DL <= 9) Then
     RegisA = RegisA + Hex(60)
     vbrcSetBitB Flag, 0
End If
If (DH >= 9) And (vbrcGetBitD(Flag, 5) = 0) And (DL >= 10) Then
     RegisA = RegisA + Hex(66)
     vbrcSetBitB Flag, 0
```

```
RegisA = RegisA + Hex(66)
        vbrcSetBitB Flag, 0
End If
If vbrcGetBitD(RegisA, 7) Then
    vbrcSetBitB Flag, 3
Else
    vbrcResetBitB Flag, 3
End If
 cc = 1
            For i = 0 To 7
                cc = (Not vbrcGetBitD(RegisA, i)) And cc
            Next i
            If cc = 0 Then
              vbrcResetBitB Flag, 2
            Else
              vbrcSetBitB Flag, 2
             End If
End If

'**************DECA************************
 Case "4A": PC = PC + 1
n = RegisA
 RegisA = RegisA - 1
  If vbrcGetBitD(RegisA, 7) = 0 Then
  vbrcResetBitB Flag, 3
Else
   vbrcSetBitB Flag, 3
End If
            cc = 1
            For i = 0 To 7
                cc = (Not vbrcGetBitD(RegisA, i)) And cc
            Next i
            If cc = 0 Then
              vbrcResetBitB Flag, 2
            Else
              vbrcSetBitB Flag, 2
             End If
        cc = 1
            For i = 0 To 7
                cc = (vbrcGetBitD(n, i) And cc) And Not
vbrcGetBitD(n, 7)
            Next i
            If cc = 0 Then
              vbrcResetBitB Flag, 1
            Else
              vbrcSetBitB Flag, 1
Case "5A": PC = PC + 1
 n = Regisb
 Regisb = Regisb - 1
  If vbrcGetBitD(Regisb, 7) = 0 Then
  vbrcResetBitB Flag, 3
Else
   vbrcSetBitB Flag, 3
 End If
            cc = 1
            For i = 0 To 7
                cc = (Not vbrcGetBitD(Regisb, i)) And cc
            Next i
            If cc = 0 Then
              vbrcResetBitB Flag, 2
            Else
              vbrcSetBitB Flag, 2
             End If
        cc = 1
            For i = 0 To 7
                cc = (vbrcGetBitD(n, i) And cc) And Not
vbrcGetBitD(n, 7)
            Next i
            If cc = 0 Then
              vbrcResetBitB Flag, 1
            Else
              vbrcSetBitB Flag, 1
             End If
'**************DECS************************
Case "34": PC = PC + 1
SP = SP - 1
'**************DECX************************
 Case "09"
Regisx = Regisx - 1
cc = 1
            For i = 0 To 15
                cc = (Not vbrcGetBitD(Regisx, i)) And cc
            Next i
            If cc = 0 Then
              vbrcResetBitB Flag, 2
            Else
              vbrcSetBitB Flag, 2
             End If
'*************
Case "15": PC = PC + 1
      Data = RAM(startaddr + PC)
```

```
Regisb = RegisA
                cc = 1
                For i = 0 To 7
                    cc = (Not vbrcGetBitD(Regisb, i)) And cc
                Next
                If vbrcGetBitD(RegisA, 7) = 0 Then
                  vbrcResetBitB Flag, 3
                Else
                   vbrcSetBitB Flag, 3
                End If
                If cc = 0 Then
                   vbrcResetBitB Flag, 2
                Else
                   vbrcSetBitB Flag, 2
                End If
                   vbrcResetBitB Flag, 1
'************TBA**********************
Case "17": PC = PC + 1
RegisA = Regisb
                cc = 1
                For i = 0 To 7
                    cc = (Not vbrcGetBitD(RegisA, i)) And 1 And cc
                Next
                If vbrcGetBitD(RegisA, 7) = 0 Then
                  vbrcResetBitB Flag, 3
                Else
                   vbrcSetBitB Flag, 3
                End If
                If cc = 0 Then
                   vbrcResetBitB Flag, 2
                Else
                   vbrcSetBitB Flag, 2
                End If
                   vbrcResetBitB Flag, 1
'************TAP**********************
Case "06": PC = PC + 1
Flag = RegisA
'************TPA**********************
Case "07": PC = PC + 1
RegisA = Flag
'************TSX**********************
Case "30": PC = PC + 1
Regisx = SP + 1
'**********TXS**************************
Case "35": PC = PC + 1
SP = Regisx - 1
RAM(SP) = RegisA
SP = SP - 1
'**************PSHB**********************
Case "37": PC = PC + 1
RAM(SP) = Regisb
SP = SP - 1
'*************PSHX**********************
Case "3C": PC = PC + 1
RAM(SP) = Regisx - ((Regisx - 256) / 256)
SP = SP - 1
RAM(SP) = Regisx - (RAM(SP + 1) * 256) - 255  'UPDATE NEW
FROM 256
SP = SP - 1
'***********PUL A**************************
Case "32": PC = PC + 1
RegisA = RAM(SP)
SP = SP + 1
'************PUL B**************************
Case "33": PC = PC + 1
Regisb = RAM(SP)
SP = SP + 1
'*************PUL X**************************
Case "38": PC = PC + 1
DH = RAM(SP + 1)
DL = RAM(SP + 2)
Regisx = (DH * 256) + DL
SP = SP + 2
'**************INCB**************************
Case "5C": PC = PC + 1
n = Regisb
Regisb = Regisb + 1
If vbrcGetBitD(Regisb, 7) = 0 Then
  vbrcResetBitB Flag, 3
Else
vbrcSetBitB Flag, 3
End If
                cc = 1
                For i = 0 To 7
                    cc = (Not vbrcGetBitD(Regisb, i)) And cc
                Next i
                If cc = 0 Then
                   vbrcResetBitB Flag, 2
                Else
                   vbrcSetBitB Flag, 2
                End If
                    cc = 1
```

```
cc = (vbrcGetBitD(n, i) And cc) And Not vbrcGetBitD(n, 7)
                Next i
                If cc = 0 Then
                  vbrcResetBitB Flag, 1
                Else
                   vbrcSetBitB Flag, 1
                 End If
'**************INS*************************
Case "31": PC = PC + 1
SP = SP + 1
'**************INX*************************
Case "08": PC = PC + 1
Regisx = Regisx + 1
            cc = 1
                For i = 0 To 15
                     cc = (Not vbrcGetBitD(Regisx, i)) And cc
                Next i
                If cc = 0 Then
                  vbrcResetBitB Flag, 2
                Else
                   vbrcSetBitB Flag, 2
                End If

Case "6A", "6C": PC = PC + 1
ADR = Regisx + RAM(startaddr + PC)
Data = RAM(ADR)
PC = PC + 1
'**************ROL A*************************
Case "49": PC = PC + 1
n = RegisA
Call vbrcROLD(RegisA, 1)
If vbrcGetBitD(RegisA, 7) = 0 Then
 vbrcResetBitB Flag, 3
Else
  vbrcSetBitB Flag, 3
End If
                cc = 1
                For i = 0 To 7
                     cc = (Not vbrcGetBitD(RegisA, i)) And cc
                Next i
                If cc = 0 Then
                   vbrcResetBitB Flag, 2
                Else
                   vbrcSetBitB Flag, 2
                 End If
                If vbrcGetBitD(Flag, 3) Xor vbrcGetBitD(Flag, 0) = 1
vbrcSetBitB Flag, 1
                Else
                vbrcResetBitB Flag, 1
                End If
                If vbrcGetBitD(n, 7) = 1 Then
                  vbrcSetBitB Flag, 0
                Else
                  vbrcResetBitB Flag, 0
                End If
'**************ROL B*************************
Case "59": PC = PC + 1
n = Regisb
Call vbrcROLD(Regisb, 1)
If vbrcGetBitD(Regisb, 7) = 0 Then
 vbrcResetBitB Flag, 3
Else
 vbrcSetBitB Flag, 3
End If
                cc = 1
                For i = 0 To 7
                     cc = (Not vbrcGetBitD(Regisb, i)) And cc
                Next i
                If cc = 0 Then
                  vbrcResetBitB Flag, 2
                Else
                  vbrcSetBitB Flag, 2
                 End If
                If vbrcGetBitD(Flag, 3) Xor vbrcGetBitD(Flag, 0) = 1
Then
                vbrcSetBitB Flag, 1
                Else
                vbrcResetBitB Flag, 1
                End If
                If vbrcGetBitD(n, 7) = 1 Then
                  vbrcSetBitB Flag, 0
                Else
                  vbrcResetBitB Flag, 0
                End If
'**************ROR A*************************
Case "46": PC = PC + 1
n = RegisA
Call vbrcRORD(RegisA, 1)
If vbrcGetBitD(RegisA, 7) = 0 Then
 vbrcResetBitB Flag, 3
Else
  vbrcSetBitB Flag, 3
```

```
For i = 0 To 7
                cc = (Not vbrcGetBitD(RegisA, i)) And cc
            Next i
            If cc = 0 Then
              vbrcResetBitB Flag, 2
            Else
               vbrcSetBitB Flag, 2
             End If
            If vbrcGetBitD(Flag, 3) Xor vbrcGetBitD(Flag, 0) = 1
Then
            vbrcSetBitB Flag, 1
            Else
            vbrcResetBitB Flag, 1
            End If
            If vbrcGetBitD(n, 0) = 1 Then
              vbrcSetBitB Flag, 3
            Else
               vbrcResetBitB Flag, 3
            End If
'**************ROR B*************************
 Case "56": PC = PC + 1
n = Regisb
Call vbrcRORD(Regisb, 1)
If vbrcGetBitD(Regisb, 7) = 0 Then
  vbrcResetBitB Flag, 3
Else
   vbrcSetBitB Flag, 3
 End If
            cc = 1
            For i = 0 To 7
                cc = (Not vbrcGetBitD(Regisb, i)) And cc
            Next i
            If cc = 0 Then
               vbrcResetBitB Flag, 2
            Else
               vbrcSetBitB Flag, 2
             End If
            If vbrcGetBitD(Flag, 3) Xor vbrcGetBitD(Flag, 0) = 1
Then
            vbrcSetBitB Flag, 1
            Else
            vbrcResetBitB Flag, 1
            End If
            If vbrcGetBitD(n, 0) = 1 Then
               vbrcSetBitB Flag, 0
            Else
'**************LSR A*************************
 Case "44": PC = PC + 1
 n = RegisA
Call vbrcSHRD(RegisA, 1)
  vbrcResetBitB Flag, 3
   cc = 1
            For i = 0 To 7
                cc = (Not vbrcGetBitD(RegisA, i)) And cc
            Next i
            If cc = 0 Then
              vbrcResetBitB Flag, 2
            Else
               vbrcSetBitB Flag, 2
             End If
            If vbrcGetBitD(Flag, 3) Xor vbrcGetBitD(Flag, 0) = 1
Then
            vbrcSetBitB Flag, 1
            Else
            vbrcResetBitB Flag, 1
            End If
            If vbrcGetBitD(n, 0) = 1 Then
              vbrcSetBitB Flag, 0
            Else
               vbrcResetBitB Flag, 0
            End If
'**************LSR B*************************
 Case "54": PC = PC + 1
 n = Regisb
 Call vbrcSHRD(Regisb, 1)
 cc = 1
            For i = 0 To 7
                cc = (Not vbrcGetBitD(Regisb, i)) And cc
            Next i
            If cc = 0 Then
              vbrcResetBitB Flag, 2
            Else
               vbrcSetBitB Flag, 2
             End If
            If vbrcGetBitD(Flag, 3) Xor vbrcGetBitD(Flag, 0) = 1
Then
            vbrcSetBitB Flag, 1
            Else
            vbrcResetBitB Flag, 1
            End If
            If vbrcGetBitD(n, 7) = 1 Then
              vbrcSetBitB Flag, 0
```

```
End If
  '**************ABX*************************
'**************SBA*************************
Case "10": PC = PC + 1
n = RegisA
RegisA = RegisA - Regisb
If vbrcGetBitD(RegisA, 7) Then
   vbrcSetBitB Flag, 3
Else
   vbrcResetBitB Flag, 3
End If
cc = 1
            For i = 0 To 7
                cc = (Not vbrcGetBitD(RegisA, i)) And cc
            Next i
            If cc = 0 Then
               vbrcResetBitB Flag, 2
            Else
               vbrcSetBitB Flag, 2
            End If
If vbrcGetBitD(n, 7) And (Not vbrcGetBitD(Regisb, 7)) And (Not
vbrcGetBitD(RegisA, 7)) Or Not (vbrcGetBitD(n, 7)) And
vbrcGetBitD(Regisb, 7) And vbrcGetBitD(RegisA, 7) = 1 Then
   vbrcSetBitB Flag, 1
Else
   vbrcResetBitB Flag, 1
End If
If (Not vbrcGetBitD(n, 7)) And vbrcGetBitD(Regisb, 7) Or
vbrcGetBitD(Regisb, 7) And vbrcGetBitD(RegisA, 7) Or
vbrcGetBitD(RegisA, 7) And (Not vbrcGetBitD(RegisA, 7)) = 1 Then
   vbrcSetBitB Flag, 0
Else
   vbrcResetBitB Flag, 0
End If
  '**************NEGA*************************
Case "40": PC = PC + 1
RegisA = 0 - RegisA
If vbrcGetBitD(RegisA, 7) Then
   vbrcSetBitB Flag, 3
Else
   vbrcResetBitB Flag, 3
End If
cc = 1
            For i = 0 To 7
                cc = (Not vbrcGetBitD(RegisA, i)) And cc
            Next i
Else
            vbrcSetBitB Flag, 2
         End If
cc = 1
         For i = 0 To 6
             cc = ((Not vbrcGetBitD(RegisA, i)) And cc) And
vbrcGetBitD(RegisA, 7)
         Next i
         If cc = 0 Then
            vbrcResetBitB Flag, 1
         Else
            vbrcSetBitB Flag, 1
         End If
For i = 0 To 7
            cc = (vbrcGetBitD(RegisA, i)) Or cc
         Next i
         If cc = 0 Then
            vbrcResetBitB Flag, 0
         Else
            vbrcSetBitB Flag, 0
         End If
'**********************NEGB*************************
Case "50": PC = PC + 1
Regisb = 0 - Regisb
If vbrcGetBitD(Regisb, 7) Then
   vbrcSetBitB Flag, 3
Else
   vbrcResetBitB Flag, 3
End If
cc = 1
         For i = 0 To 7
             cc = (Not vbrcGetBitD(Regisb, i)) And cc
         Next i
         If cc = 0 Then
            vbrcResetBitB Flag, 2
         Else
            vbrcSetBitB Flag, 2
         End If
cc = 1
         For i = 0 To 6
             cc = ((Not vbrcGetBitD(Regisb, i)) And cc) And
vbrcGetBitD(Regisb, 7)
         Next i
         If cc = 0 Then
            vbrcResetBitB Flag, 1
         Else
```

```
For i = 0 To 7
                    cc = (vbrcGetBitD(Regisb, i)) Or cc
                Next i
                If cc = 0 Then
                  vbrcResetBitB Flag, 0
                Else
                  vbrcSetBitB Flag, 0
                End If
'***************MUL************************
Case "3D": PC = PC + 1
ss = RegisA * Regisb
              If ss > 255 Then
               L = (ss Mod 256)
               RegisA = (ss / 256) - (L / 256)
               Regisb = ss - (RegisA * 256)
              Else
               RegisA = 0
               Regisb = ss
              End If

If vbrcGetBitD(Regisb, 7) Then
   vbrcSetBitB Flag, 0
Else
   vbrcResetBitB Flag, 0
End If
'***************IDIV************************
Case "02": PC = PC + 1
x = Regisx
n = Regisd / Regisx
Regisd = Regisd - (n * Regisx)
Regisx = n
cc = 1
      For i = 0 To 15
        cc = (Not vbrcGetBitD(Regisx, i)) And cc
      Next i
      If cc = 0 Then
        vbrcResetBitB Flag, 2
      Else
        vbrcSetBitB Flag, 2
      End If
        vbrcResetBitB Flag, 1
        cc = 1
    For i = 0 To 15
      cc = (Not vbrcGetBitD(Regisx, i)) And cc
    Next i
    If cc = 0 Then
vbrcSetBitB Flag, 0
   End If
'***************FDIV************************
Case "10": PC = PC + 1
n = Regisx
Regisx = Regisd / Regisx
 cc = 1
            For i = 0 To 15
              cc = (Not vbrcGetBitD(Regisx, i)) And cc
            Next i
            If cc = 0 Then
              vbrcResetBitB Flag, 2
            Else
              vbrcSetBitB Flag, 2
            End If
           If Regisx <= Regisd Then
           vbrcSetBitB Flag, 1
           Else
           vbrcResetBitB Flag, 1
           cc = 1
            For i = 0 To 15
              cc = (Not vbrcGetBitD(n, i)) And cc
            Next i
            If cc = 0 Then
              vbrcResetBitB Flag, 0
            Else
              vbrcSetBitB Flag, 0
            End If
           End If
'***************TSTA************************
 Case "4D": PC = PC + 1
result% = RegisA - 0
If vbrcGetBitD(RegisA, 7) Then
  vbrcSetBitB Flag, 3
 Else
   vbrcResetBitB Flag, 3
 End If
 cc = 1
            For i = 0 To 7
              cc = (Not vbrcGetBitD(RegisA, i)) And cc
            Next i
            If cc = 0 Then
              vbrcResetBitB Flag, 2
            Else
              vbrcSetBitB Flag, 2
            End If
```

```
'***************TSTB************************
 Case "8F": PC = PC + 1
        Data = Regisx
        Regisx = Regisd
              If Data > 255 Then
               RegisA = (Data / 256)
               Regisb = Data - (RegisA * 255)
              Else
               RegisA = 0
               Regisb = Data
              End If
 Case "5D": PC = PC + 1
result% = Regisb - 0
If vbrcGetBitD(Regisb, 7) Then
   vbrcSetBitB Flag, 3
 Else
   vbrcResetBitB Flag, 3
 End If
 cc = 1
              For i = 0 To 7
                  cc = (Not vbrcGetBitD(Regisb, i)) And cc
              Next i
              If cc = 0 Then
               vbrcResetBitB Flag, 2
              Else
               vbrcSetBitB Flag, 2
              End If
               vbrcResetBitB Flag, 1
               vbrcResetBitB Flag, 0
 '***************RTS************************
 Case "39": PC = PC + 1
S$ = Right("0000" & Hex(startaddr + PC), 4)
DH = Val(Mid(S$, 1, 2))
DL = Val(Mid(S$, 3, 2))
 RAM(SP + 1) = DH
 RAM(SP + 2) = DL
 SP = SP + 2
 '***************RTI************************
 Case "3B": PC = PC + 1
RAM(SP) = OPCODE
 RAM(SP + 1) = Regisb
 RAM(SP + 2) = RegisA

S$ = Right("0000" & Hex(Regisx), 4)
DH = Val(Mid(S$, 1, 2))
DL = Val(Mid(S$, 3, 2))
S$ = Right("0000" & Hex(Regisy), 4)
DH = Val(Mid(S$, 1, 2))
DL = Val(Mid(S$, 3, 2))
 RAM(SP + 6) = DH
 RAM(SP + 7) = DL

 S$ = Right("0000" & Hex(PC + startaddr), 4)
DH = Val(Mid(S$, 1, 2))
DL = Val(Mid(S$, 3, 2))
 RAM(SP + 8) = DH
 RAM(SP + 9) = DL
 SP = SP + 9
 '***************SEC************************
 Case "0D": PC = PC + 1
 vbrcSetBitB Flag, 0
 '***************SEV************************
 Case "0B": PC = PC + 1
 vbrcSetBitB Flag, 1
 '***************SEI************************
 Case "0F": PC = PC + 1
 vbrcSetBitB Flag, 4
 '***************SWI************************
 Case "3F": PC = PC + 1
 DH = SP
 RAM(SP - 8) = Val(OPCODE)
 RAM(SP - 7) = Regisb
 RAM(SP - 6) = RegisA
S$ = Right("0000" & Hex(Regisx), 4)
DH = Val(Mid(S$, 1, 2))
DL = Val(Mid(S$, 3, 2))
 RAM(SP - 5) = DH
 RAM(SP - 4) = DL

S$ = Right("0000" & Hex(Regisy), 4)
DH = Val(Mid(S$, 1, 2))
DL = Val(Mid(S$, 3, 2))
 RAM(SP - 3) = DH
 RAM(SP - 2) = DL

S$ = Right("0000" & Hex(startaddr + PC), 4)
DH = Val(Mid(S$, 1, 2))
DL = Val(Mid(S$, 3, 2))
 RAM(SP - 1) = DH
 RAM(SP) = DL
 vbrcSetBitB Flag, 4
SP = SP - 8
```

```
DH = Regisx
        Regisx = Regisd
        Regisd = DH

'**************NOP*************************
Case "01": PC = PC + 1

'Opcode 1 Operand1 Immediate Mode 20 คำสั่ง
Case "89", "C9", "8B", "CB", "84", "C4", "85", "C5", "81", "C1",
"88", "C8", "86", "C6", "8A", "CA", "82", "C2", "80", "C0":
PC = PC + 1
Data = RAM(startaddr + PC)
ADR = startaddr + PC
PC = PC + 1
'Opcode 1 Operand1 DIR Mode 32 คำสั่ง
Case "99", "D9", "9B", "DB", "D3", "94", "D4", "95", "D5", "91",
"D1", "9C", "98", "D8", "96", "D6", "DC", "9E", "9A", "DA", "92",
"D2", "97", "D7", "DD", "9F", "DF", "90", "D0", "93", "9D":
PC = PC + 1
ADR = RAM(startaddr + PC)
Data = RAM(ADR)
PC = PC + 1
'Opcode 1 Operand1 Ind,X  Mode 45 คำสั่ง
Case "A9", "E9", "AB", "EB", "E3", "A4", "E4", "68", "67", "A5",
"E5", "6F", "A1", "E1", "63", "AC", "6A", "A8", "E8", "6C", "6E",
"AD", "A6", "E6", "EC", "AE", "EE", "68", "64", "60", "AA", "EA",
"69", "66", "A2", "E2", "A7", "E7", "ED", "AF", "EF", "A0", "E0",
"A3", "6D":
PC = PC + 1
ADR = Regisx + RAM(startaddr + PC)
Data = RAM(ADR)
PC = PC + 1
'Opcode 1 Operand1 REL Mode 18 คำสั่ง
Case "25", "27", "2C", "2E", "22", "24", "2F", "25", "23", "2D", "2B",
"26", "2A", "20", "21", "8D", "28", "29":
PC = PC + 1
DH = RAM(startaddr + PC)
PC = PC + 1

'Opcode 1 Operand 2 Immedaite Mode 6 คำสั่ง
Case "C3", "8C", "CC", "8E", "CE", "83":
PC = PC + 1
DH = RAM(startaddr + PC)
DL = RAM(startaddr + PC + 1)
Data = (DH * 256) + DL
PC = PC + 2
ADR = RAM(startaddr + PC)
DH = RAM(startaddr + PC + 1)
Data = RAM(ADR)
PC = PC + 2
'Opcode 1 Operand 2 EXT Mode 45 คำสั่ง
Case "89", "F9", "BB", "FB", "F3", "B4", "F4", "78", "77", "B5",
"F5", "7F", "B1", "F1", "73", "BC", "7A", "B8", "F8", "7C", "BE",
"FE", "78", "74", "70", "BA", "FA", "79", "76", "B2", "F2", "B7",
"F7", "FD", "BF", "FF", "B0", "F0", "B3", "7D", "F6", "BD", "B6":
PC = PC + 1
DH = RAM(startaddr + PC)
DL = RAM(startaddr + PC + 2)
ADR = (DH * 256) + DL
Data = RAM(ADR)
'Opcode 1 Operand 2 Ind,X  Mode 2 คำสั่ง
Case "1D", "1C": PC = PC + 1
ADR = Regisx + RAM(startaddr + PC)
Data = RAM(ADR)
DH = RAM(startaddr + PC + 1)
PC = PC + 2
'OPCODE 1 OPERAND 3
'DIR 2 คำสั่ง
Case "13", "12": PC = PC + 1
ADR = RAM(startaddr + PC)
Data = RAM(ADR)
DH = RAM(startaddr + PC + 1)
DL = RAM(startaddr + 2)
PC = PC + 3
'IND,X  2 คำสั่ง
Case "1E", "1F": PC = PC + 1
ADR = Regisx + RAM(startaddr + PC)
Data = RAM(ADR)
DH = RAM(startaddr + PC + 1)
DL = RAM(startaddr + 2)
PC = PC + 3
'OPCODE 2
Case "18", "1A", "CD":
        PC = PC + 1
        OPCODE = OPCODE + Right("00" & Hex(RAM(startaddr +
PC)), 2)
    Select Case OPCODE
        'OPCODE 2 OPERAND INH  Mode
        Case "183A": ' คำสั่ง ABY
            PC = PC + 1
            Regisy = Regisy + Regisb
        Case "1809": 'คำสั่ง DEY
```

```
cc = 1
                For i = 0 To 15
                    cc = (Not vbrcGetBitD(Regisy, i)) And cc
                Next i
                If cc = 0 Then
                  vbrcResetBitB Flag, 2
                Else
                  vbrcSetBitB Flag, 2
                End If
            Case "183C": 'PSHY
             PC = PC + 1
             RAM(SP) = Regisy - ((Regisy - 256) / 256)
             SP = SP - 1
             RAM(SP) = Regisy - (RAM(SP + 1) * 256) - 255  'UPDATE
NEW FROM 256
            SP = SP - 1
            Case "1838": 'PULY
            PC = PC + 1
             Regisy = RAM(SP) - ((RAM(SP) - 256) / 256)
            SP = SP + 1
            Regisy = RAM(SP - 1) - (Regisy * 256) - 256
            SP = SP + 1
            Case "1830" 'TSY
            PC = PC + 1
            Regisy = SP + 1
            Case "1835" 'TYS
            PC = PC + 1
            SP = Regisy - 1
            Case "188F":
            DH = Regisy
            Regisy = Regisd
            Regisd = DH
            'OPCODE 2 OPERAND 1 DIR Mode 4 คำสั่ง
            Case "1A93", "18DE", "18DF", "189C":
                PC = PC + 1
                ADR = RAM(startaddr + PC)
                Data = RAM(ADR)
                PC = PC + 1
            'OPCODE 2 OPERAND 1 Ind,X  Mode 4 คำสั่ง
             Case "1AA3", "1AAC", "1AEE", "1AEF":
                PC = PC + 1
                ADR = Regisx + RAM(startaddr + PC)
                Data = RAM(ADR)
                PC = PC + 1
            'OPCODE 2 OPERAND 1 Ind,Y Mode 48 คำสั่ง
              Case "18A9", "18E9", "18AB", "18E3", "18A4", "18E4",
"18AD", "18A6", "18E6", "18EC", "18AE", "CDEE", "18EE",
"1868", "1864", "1860", "18AA", "18EA", "1869", "1866", "18A2",
"18E2", "18A7", "18E7", "18ED", "18AF", "CDEF", "18EF", "18A0",
"18E0", "18A3", "186D":
                PC = PC + 1
                ADR = Regisy + RAM(startaddr + PC)
                Data = RAM(ADR)
                PC = PC + 1
            'OPCODE 2 OPERAND 2 Imm Mode มี 3 คำสั่ง
            Case "18CE", "188C", "1A83":
                PC = PC + 1
                DH = RAM(startaddr + PC)
                DL = RAM(startaddr + PC + 1)
                Data = (DH * 256) + DL
                PC = PC + 2
            'OPCODE 2 OPERAND 2 EXT Mode มี 4 คำสั่ง
             Case "1AB3", "18BC", "18FE", "18FF":
                PC = PC + 1
                DH = RAM(startaddr + PC)
                DL = RAM(startaddr + PC + 1)
                ADR = (DH * 256) + DL
                Data = RAM(ADR)
                PC = PC + 2
            'OPCODE 2 OPERAND 2 Ind,Y Mode มี 2 คำสั่ง
             Case "181D", "181C":
                PC = PC + 1
                ADR = Regisy + RAM(startaddr + PC)
                Data = RAM(ADR)
                DH = RAM(startaddr + PC + 1)
            'OPCODE 2 OPERAND 3   Ind,Y Mode มี 2 คำสั่ง
              Case "181F", "181E":
                PC = PC + 1
                ADR = Regisy + RAM(startaddr + PC)
                Data = RAM(ADR)
                DH = RAM(startaddr + PC + 1)
                DL = RAM(startaddr + PC + 2)
                PC = PC + 3
        End Select
    End Select
    Call LOAD
    Screen.MousePointer = 0
    'กำหนดเมาส์ในสภาวะปกติ
    End Sub

    Sub LOAD()
    Dim r As Long
```

```
Dim DD As Integer
Select Case OPCODE
        'กลุ่มคำสั่งบวก
        'ADCA with carry
        Case "89", "99", "B9", "A9", "18A9"
          ss = RegisA + Data
          If vbrcGetBitD(ss, 7) Then
            vbrcSetBitB Flag, 3
          Else
            vbrcResetBitB Flag, 3
          End If
            B(1) = vbrcGetBitD(RegisA, 3) And vbrcGetBitD
(Data, 3) Or vbrcGetBitD(Data, 3) And Not (vbrcGetBitD(ss, 3)) Or
Not (vbrcGetBitD(ss, 3)) And vbrcGetBitD(RegisA, 3)
          If B(1) Then
            vbrcSetBitB Flag, 5
          Else
            vbrcResetBitB Flag, 5
          End If
            cc = 1
          For i = 0 To 7
            cc = (Not vbrcGetBitD(ss, i)) And cc
          Next
          If cc = 0 Then
            vbrcResetBitB Flag, 2
          Else
            vbrcSetBitB Flag, 2
          End If
            B(1) = vbrcGetBitD(RegisA, 7) And
vbrcGetBitD(Data, 7) And Not (vbrcGetBitD(ss, 7)) Or Not
(vbrcGetBitD(RegisA, 7)) And Not (vbrcGetBitD(Data, 7)) And
vbrcGetBitD(ss, 7)
          If B(1) Then
            vbrcSetBitB Flag, 1
          Else
            vbrcResetBitB Flag, 1
          End If
        B(1) = vbrcGetBitD(RegisA, 7) And vbrcGetBitD(Data, 7)
Or vbrcGetBitD(Data, 7) And Not (vbrcGetBitD(ss, 7)) Or Not
(vbrcGetBitD(ss, 7)) And vbrcGetBitD(RegisA, 7)
        If B(1) Then
          vbrcSetBitB Flag, 0
        Else
          vbrcResetBitB Flag, 0
        End If
        Regisb = ss + B(1)
If vbrcGetBitD(ss, 7) Then
          vbrcSetBitB Flag, 3
        Else
          vbrcResetBitB Flag, 3
        End If

        B(1) = vbrcGetBitD(Regisb, 3) And vbrcGetBitD(Data,
3) Or vbrcGetBitD(Data, 3) And Not (vbrcGetBitD(ss, 3)) Or Not
(vbrcGetBitD(ss, 3)) And vbrcGetBitD(Regisb, 3)
        If B(1) Then
          vbrcSetBitB Flag, 5
        Else
          vbrcResetBitB Flag, 5
        End If
        cc = 1
        For i = 0 To 7
          cc = (Not vbrcGetBitD(ss, i)) And 1 And cc
        Next
        If cc = 0 Then
          vbrcResetBitB Flag, 2
        Else
          vbrcSetBitB Flag, 2
        End If
          B(1) = vbrcGetBitD(Regisb, 7) And vbrcGetBitD
(Data, 7) And Not (vbrcGetBitD(ss, 7)) Or Not (vbrcGetBitD(Regisb,
7)) And Not (vbrcGetBitD(Data, 7)) And vbrcGetBitD(ss, 7)

        If B(1) Then
          vbrcSetBitB Flag, 1
        Else
          vbrcResetBitB Flag, 1
        End If

          B(1) = vbrcGetBitD(Regisb, 7) And vbrcGetBitD
(Data, 7) Or vbrcGetBitD(Data, 7) And Not (vbrcGetBitD(ss, 7)) Or
Not (vbrcGetBitD(ss, 7)) And vbrcGetBitD(Regisb, 7)

        If B(1) Then
          vbrcSetBitB Flag, 0
        Else
          vbrcResetBitB Flag, 0
        End If
          Regisb = ss + B(1)
    ' คำสั่ง ADDD
    Case "C3", "D3", "F3", "E3", "18E3":
          Data = Regisd + RAM(ADR)
```

```
RegisA = (Data - 255) / 15
                Registb = Data - RegisA
               Else
                RegisA = 0
                Registb = Data
               End If
                If vbrcGetBitD(RegisA, 7) Then
                   vbrcSetBitB Flag, 3
                Else
                   vbrcResetBitB Flag, 3
                End If
               cc = 1
               For i = 0 To 7
                  cc = cc And Not (vbrcGetBitD(RegisA, i)) And
Not (vbrcGetBitD(Registb, i))
               Next i
               If cc Then
                   vbrcSetBitB Flag, 2
               Else
                   vbrcResetBitB Flag, 2
               End If
               B(1) = vbrcGetBitD(RegisA, 7) And vbrcGetBitD
(ADR, 15) And Not (vbrcGetBitD(Data, 15)) Or Not (vbrcGetBitD
(RegisA, 7))
  'คำสั่ง ANDA
          Case "84", "94", "B4", "A4", "18A4":
            RegisA = Hex(RegisA) And Data
              vbrcResetBitB Flag, 1 'V=0
            If vbrcGetBitD(RegisA, 15) Then 'N
              vbrcSetBitB Flag, 3
            Else
              vbrcResetBitB Flag, 3
            End If
             cc = 1 'Z
             For i = 0 To 7
             cc = cc And Not vbrcGetBitD(RegisA, i)
             Next i
  'คำสั่ง ANDB
          Case "C4", "D4", "F4", "E4", "18E4":
           Registb = Hex(Registb) And Data
           vbrcResetBitB Flag, 1 'V=0
            If vbrcGetBitD(Registb, 15) Then 'N
              vbrcSetBitB Flag, 3
            Else
              vbrcResetBitB Flag, 3
            End If
cc = cc And Not vbrcGetBitD(Registb, i)
            Next i
 'คำสั่ง ASL
  Case "78", "68", "1868":
    n = Data
    Data = Data * 2
    RAM(ADR) = Data
    If n Mod 2 Then
      vbrcSetBitB Flag, 0
    Else
      vbrcResetBitB Flag, 0
   End If
    If vbrcGetBitD(Data, 7) = 0 Then
      vbrcResetBitB Flag, 3
   Else
      vbrcSetBitB Flag, 3
   End If
      cc = 1
      For i = 0 To 7
        cc = (Not vbrcGetBitD(Data, i)) And cc
      Next i
      If cc = 0 Then
        vbrcResetBitB Flag, 2
      Else
        vbrcSetBitB Flag, 2
      End If
      If vbrcGetBitD(Flag, 3) Xor vbrcGetBitD(Flag, 0) = 1 Then
        vbrcSetBitB Flag, 1
      Else
        vbrcResetBitB Flag, 1
      End If
 'คำสั่ง ASR
 Case "77", "67", "1867":
  n = Data
  Data = Data / 2
  If (n Mod 2) Then
    vbrcSetBitB Flag, 0
  Else
    vbrcResetBitB Flag, 0
  End If
  If vbrcGetBitD(Data, 7) = 0 Then
    vbrcResetBitB Flag, 3
  Else
    vbrcSetBitB Flag, 3
  End If
    cc = 1
```

```
Next i
        If cc = 0 Then
          vbrcResetBitB Flag, 2
        Else
          vbrcSetBitB Flag, 2
        End If
        If vbrcGetBitD(Flag, 3) Xor vbrcGetBitD(Flag, 0) = 1 Then
          vbrcSetBitB Flag, 1
        Else
          vbrcResetBitB Flag, 1
        End If
' คำสั่ง BCLR
    Case "15", "1D", "181D":
        RAM(startaddr + PC) = Data And Not (RAM(startaddr + PC
+ 1))
        PC = PC + 1
        vbrcResetBitB Flag, 1
        If vbrcGetBitD(RAM(startaddr + PC), 7) Then
          vbrcSetBitB Flag, 7
        Else
          vbrcResetBitB Flag, 7
        End If
        cc = 1
        For i = 1 To 7
        cc = Not vbrcGetBitD(RAM(startaddr + PC), i) And cc
        Next i
        If cc Then
          vbrcSetBitB Flag, 2
        Else
          vbrcResetBitB Flag, 2
        End If
'คำสั่ง BITA
    Case "85", "95", "B5", "A5", "18A5":
        result% = RegisA And Data
        If vbrcGetBitD(RegisA, 7) Then
          vbrcSetBitB Flag, 3
        Else
          vbrcResetBitB Flag, 3
        End If
        cc = 1
        For i = 0 To 7
            cc = cc And Not vbrcGetBitD(result%, i)
        Next i
        If cc Then
          vbrcSetBitB Flag, 2
        Else
          vbrcResetBitB Flag, 1
'คำสั่ง BITB
    Case "C5", "D5", "F5", "E5", "18E5":
        result% = Regisb And Data
        If vbrcGetBitD(Regisb, 7) Then
          vbrcSetBitB Flag, 3
        Else
          vbrcResetBitB Flag, 3
        End If
        cc = 1
        For i = 0 To 7
            cc = cc And Not vbrcGetBitD(result%, i)
        Next i
        If cc Then
          vbrcSetBitB Flag, 2
        Else
          vbrcResetBitB Flag, 2
        End If
        vbrcResetBitB Flag, 1
'คำสั่ง BRCLR
        Case "13", "1F", "181F":
        If Not (Data And (PC + 2)) Then
          PC = PC + 4 + Data
        End If
        If Not (Data And (PC + 3)) Then
          PC = PC + 5 + Data
        End If
'คำสั่ง BRSET
        Case "12", "1E", "181E":
          If Not ((Not Data) And (PC + 2)) Then
            PC = PC + 4 + Data + startaddr + PC
          End If
          If Not ((Not Data) And (PC + 3)) Then
            PC = PC + 5 + Data + startaddr + PC
          End If
'คำสั่ง CLR
        Case "7F", "6F", "186F":
          RAM(ADR) = 0
          RegisA = 0
          vbrcSetBitB Flag, 2
          vbrcResetBitB Flag, 0
          vbrcResetBitB Flag, 1
          vbrcResetBitB Flag, 3
'คำสั่ง CMPA
          Case "81", "91", "B1", "A1", "18A1":
            result% = RegisA - Hex(Data)
```

```
Else
                vbrcResetBitB Flag, 7
            End If
            cc = 1
            For i = 0 To 7
            cc = cc And Not vbrcGetBitD(result%, i)
            Next i
            If cc Then
              vbrcSetBitB Flag, 2
            Else
               vbrcResetBitB Flag, 2
            End If
            cc = vbrcGetBitD(RegisA, 7) And Not vbrcGetBitD
(Data, 7) And Not vbrcGetBitD(result%, 7)
'คำสั่ง CMPB
          Case "C1", "D1", "F1", "E1", "18E1":
           result% = Regisb - Hex(Data)
           If vbrcGetBitD(result%, 7) Then
              vbrcSetBitB Flag, 7
          Else
             vbrcResetBitB Flag, 7
          End If
            cc = 1
            For i = 0 To 7
            cc = cc And Not vbrcGetBitD(result%, i)
            Next i
            If cc Then
              vbrcSetBitB Flag, 2
            Else
               vbrcResetBitB Flag, 2
            End If
            cc = vbrcGetBitD(Regisb, 7) And Not vbrcGetBitD
(Data, 7) And Not vbrcGetBitD(result%, 7)
' คำสั่ง COM
Case "9F", "BF", "AF", "18AF":
 If SP > 255 Then
  RAM(ADR) = (SP / 256)
  RAM(ADR + 1) = SP - (RAM(ADR) * 255)
Else
  RAM(ADR) = 0
  RAM(ADR + 1) = SP
End If
   vbrcResetBitB Flag, 1
If vbrcGetBitD(SP, 15) Then
    vbrcSetBitB Flag, 3
Else
cc = 1
For i = 0 To 15
cc = cc And Not vbrcGetBitD(SP, i)
Next i
If cc Then
   vbrcSetBitB Flag, 2
Else
   vbrcResetBitB Flag, 2
End If

Case "76", "66", "1866":
n = Data
Call vbrcRORD(Data, 1)

If vbrcGetBitD(Flag, 0) Then
   Data = Data Or 8
   vbrcResetBitB Flag, 3
Else
  Data = Data And 7
  vbrcSetBitB Flag, 3
End If
cc = 1
For i = 0 To 7
    cc = (Not vbrcGetBitD(Data, i)) And cc
Next i
If cc = 0 Then
   vbrcResetBitB Flag, 2
Else
   vbrcSetBitB Flag, 2
End If
If vbrcGetBitD(Flag, 3) Xor vbrcGetBitD(Flag, 0) = 1 Then
   vbrcSetBitB Flag, 1
Else
   vbrcResetBitB Flag, 1
End If
If vbrcGetBitD(n, 0) = 1 Then
   vbrcSetBitB Flag, 0
Else
   vbrcResetBitB Flag, 0
End If
 RAM(ADR) = Data
Case "70", "60", "1860":
'***
 Data = 0 - Data
 If vbrcGetBitD(Data, 7) Then
    vbrcSetBitB Flag, 3
```

```
End If
 cc = 1
                For i = 0 To 7
                    cc = (Not vbrcGetBitD(Data, i)) And cc
                Next i
                If cc = 0 Then
                  vbrcResetBitB Flag, 2
                Else
                  vbrcSetBitB Flag, 2
                End If
 cc = 1
                For i = 0 To 6
                    cc = ((Not vbrcGetBitD(Data, i)) And cc) And
vbrcGetBitD(Data, 7)
                Next i
                If cc = 0 Then
                  vbrcResetBitB Flag, 1
                Else
                  vbrcSetBitB Flag, 1
                End If
For i = 0 To 7
                    cc = (vbrcGetBitD(Data, i)) Or cc
                Next i
                If cc = 0 Then
                  vbrcResetBitB Flag, 0
                Else
                  vbrcSetBitB Flag, 0
                End If
RAM(ADR) = Data
'****
Case "74", "64", "1864":
 n = Data
 Call vbrcSHRD(Data, 1)
 cc = 1
                For i = 0 To 7
                    cc = (Not vbrcGetBitD(Data, i)) And cc
                Next i
                If cc = 0 Then
                  vbrcResetBitB Flag, 2
                Else
                  vbrcSetBitB Flag, 2
                End If
                If vbrcGetBitD(Flag, 3) Xor vbrcGetBitD(Flag, 0) = 1
Then
                vbrcSetBitB Flag, 1
                Else
                If vbrcGetBitD(n, 7) = 1 Then
                    vbrcSetBitB Flag, 0
                  Else
                    vbrcResetBitB Flag, 0
                  End If
RAM(ADR) = Data
      Case "14", "1C":
        RAM(ADR) = Data + (PC + 2)
        If vbrcGetBitD(RAM(ADR), 7) Then
          vbrcSetBitB Flag, 7
        Else
          vbrcResetBitB Flag, 7
        End If
        cc = 1
        For i = 0 To 7
        cc = cc And (Not vbrcGetBitD(RAM(ADR), i))
        Next i
        vbrcResetBitB Flag, 1
      Case "181C":
        RAM(ADR) = Data + (PC + 3)

      Case "2A":
       If Not vbrcGetBitD(Flag, 2) Then
         PC = PC + 2 + Data
       End If

      Case "26":
       If vbrcGetBitD(Flag, 2) Then
        PC = PC + 2 + Data
       End If
      Case "2B":
       If vbrcGetBitD(Flag, 3) Then
         PC = PC + 2 + Data
       End If
    Case "28":
      If Not vbrcGetBitD(Flag, 1) Then
        PC = PC + 2 + ADR
      End If
    Case "29":
      If vbrcGetBitD(Flag, 1) Then
        PC = PC + 2 + ADR
      End If

    Case "25": 'BLO REL
       If vbrcGetBitD(Flag, 0) Then
```

```
Case "27":
        If vbrcGetBitD(Flag, 1) Then
            PC = PC + 2 + Data
        End If
 Case "2C":
        If (vbrcGetBitD(Flag, 3) Xor vbrcGetBitD(Flag, 1)) Then
            PC = PC + 2 + Data
        End If
Case "2E":
       If (vbrcGetBitD(Flag, 2) And (vbrcGetBitD(Flag, 3) Xor
vbrcGetBitD(Flag, 1))) Then
            PC = PC + 2 + Data
       End If
   Case "03":
     ' PC = PC + 1
      QUOTIENT% = Regisd / Regisx
      REMEND% = Regisd - (REMEND% * Regisx)
  Regisd = QUOTIENT%
  Regisx = REMEND%
  cc = 1
  For i = 0 To 15
      cc = cc And vbrcGetBitD(Regisd, i)
  Next i
  If cc = 0 Then
    vbrcResetBitB Flag, 3
  Else
    vbrcSetBitB Flag, 3
  End If
   cc = 1
   For i = 0 To 15
      cc = cc And vbrcGetBitD(Regisx, i)
   Next i
        If cc = 0 Then
            vbrcResetBitB Flag, 0
        Else
            vbrcSetBitB Flag, 0
        End If
   If Regisx <= Regisd Then
    vbrcSetBitB Flag, 1
   Else
    vbrcResetBitB Flag, 1
   End If
'กลุ่มคำสั่ง โหลด LDD
       Case "86", "96", "B6", "A6", "18A6":
            RegisA = Data
            cc = 1
Next
    If vbrcGetBitD(RegisA, 7) = 0 Then
      vbrcResetBitB Flag, 3
    Else
      vbrcSetBitB Flag, 3
    End If
    If cc = 0 Then
      vbrcResetBitB Flag, 2
    Else
      vbrcSetBitB Flag, 2
    End If
      vbrcResetBitB Flag, 1

Case "C6", "D6", "F6", "E6", "18E6":
      Regisb = Data
 cc = 1
For i = 0 To 7
    cc = (Not vbrcGetBitD(Regisb, i)) And 1 And cc
Next
 If vbrcGetBitD(Regisb, 7) = 0 Then
    vbrcResetBitB Flag, 3
Else
    vbrcSetBitB Flag, 3
End If

 If cc = 0 Then
   vbrcResetBitB Flag, 2
 Else
 vbrcSetBitB Flag, 2
 End If
 vbrcResetBitB Flag, 1

Case "CC", "DC", "FC", "EC", "18EC":

  If Data > 255 Then
   RegisA = (Data / 256)
   Regisb = Data - (RegisA * 256)
  Else
   RegisA = 0
   Regisb = Data
  End If
   cc = 1
   DD = 1
  For i = 0 To 7
     cc = (Not vbrcGetBitD(RegisA, i)) And 1 And cc
     DD = (Not vbrcGetBitD(Regisb, i)) And 1 And DD
```

```
If vbrcGetBitD(RegisA, 7) = 0 Then
        vbrcResetBitB Flag, 3
       Else
        vbrcSetBitB Flag, 3
       End If

       If cc And DD Then
         vbrcResetBitB Flag, 2
       Else
         vbrcSetBitB Flag, 2
       End If
         vbrcResetBitB Flag, 1

      Case "8E", "9E", "BE", "AE", "18AE":
           SP = Data
           cc = 1
      For i = 0 To 15
           cc = (Not vbrcGetBitD(RegisSP, i)) And 1 And cc
      Next
       If vbrcGetBitD(RegisSP, 15) = 0 Then
         vbrcResetBitB Flag, 3
       Else
         vbrcSetBitB Flag, 3
       End If

       If cc = 0 Then
        vbrcResetBitB Flag, 2
        Else
         vbrcSetBitB Flag, 2
       End If
       vbrcResetBitB Flag, 1

      Case "CE", "DE", "FE", "EE", "CDEE":
           Regisx = Data
           cc = 1
           For i = 0 To 15
           cc = (Not vbrcGetBitD(Regisx, i)) And 1 And cc
           Next
           If vbrcGetBitD(Regisx, 15) = 0 Then
             vbrcResetBitB Flag, 3
           Else
             vbrcSetBitB Flag, 3
           End If

           If cc = 0 Then
             vbrcResetBitB Flag, 2
```

```
If vbrcGetBitD(Regisb, 7) = 0 Then
             vbrcResetBitB Flag, 3
           Else
             vbrcSetBitB Flag, 3
           End If
           If cc = 0 Then
             vbrcResetBitB Flag, 2
           Else
             vbrcSetBitB Flag, 2
           End If
             vbrcResetBitB Flag, 1
      Case "DD", "FD", "ED", "18ED":
           RAM(ADR) = RegisA
           RAM(ADR + 1) = Regisb
           cc = 1
           DD = 1
           For i = 0 To 7
             cc = (Not vbrcGetBitD(RegisA, i)) And 1 And
cc
             DD = (Not vbrcGetBitD(Regisb, i)) And 1 And
DD
           Next
         If vbrcGetBitD(RegisA, 7) = 0 Then
           vbrcResetBitB Flag, 3
         Else
          vbrcSetBitB Flag, 3
         End If

         If cc And DD Then
           vbrcResetBitB Flag, 2
          Else
         vbrcSetBitB Flag, 2
         End If
         vbrcResetBitB Flag, 1

        Case "DF", "FF", "EF", "CDEF":
           If Regisx > 256 Then
            r = Regisx / 256
           Else
           r = 0
           End If
           n = Regisx - (r * 256)
           RAM(Data) = r
           RAM(Data + 1) = n
                cc = 1
      For i = 0 To 15
```

```
Next
        If vbrcGetBitD(Regisx, 15) = 0 Then
            vbrcResetBitB Flag, 3
        Else
            vbrcSetBitB Flag, 3
        End If

        If cc = 0 Then
         vbrcResetBitB Flag, 2
         Else
          vbrcSetBitB Flag, 2
        End If
        vbrcResetBitB Flag, 1

    Case "18DF", "18FF", "1AEF", "18EF":
            If Regisy > 256 Then
             r = Regisy / 256
             Else
             r = 0
             End If
             n = Regisy - (r * 256)
             RAM(Data) = r
             RAM(Data + 1) = n
                        cc = 1
      For i = 0 To 15
          cc = (Not vbrcGetBitD(Regisy, i)) And 1 And cc
      Next
        If vbrcGetBitD(Regisy, 15) = 0 Then
            vbrcResetBitB Flag, 3
        Else
            vbrcSetBitB Flag, 3
        End If

        If cc = 0 Then
         vbrcResetBitB Flag, 2
         Else
          vbrcSetBitB Flag, 2
        End If
        vbrcResetBitB Flag, 1

        'กลุ่ม INC  DEC

        Case "7C", "6C", "186C"
           Data = Data + 1
           cc = 1
         For i = 0 To 7
If vbrcGetBitD(Data, 7) = 0 Then
            vbrcResetBitB Flag, 3
         Else
            vbrcSetBitB Flag, 3
        End If

        If cc = 0 Then
         vbrcResetBitB Flag, 2
         Else
          vbrcSetBitB Flag, 2
        End If
         cc = 1
        For i = 0 To 6
            cc = vbrcGetBitD(Data, i) And 1 And cc
        Next
         If cc And (Not (vbrcGetBitD(Data, 7))) Then
         vbrcSetBitB Flag, 1
         Else
         vbrcResetBitB Flag, 1
         End If

         Case "4C": 'INCA
         RegisA = RegisA + 1
        cc = 1
        For i = 0 To 7
            cc = (Not vbrcGetBitD(RegisA, i)) And 1 And cc
        Next
         If vbrcGetBitD(RegisA, 7) = 0 Then
            vbrcResetBitB Flag, 3
         Else
            vbrcSetBitB Flag, 3
         End If
        If cc = 0 Then
         vbrcResetBitB Flag, 2
         Else
          vbrcSetBitB Flag, 2
        End If
         cc = 1
        For i = 0 To 6
            cc = vbrcGetBitD(RegisA, i) And 1 And cc
        Next
          If cc And (Not (vbrcGetBitD(RegisA, 7))) Then
          vbrcSetBitB Flag, 1
          Else
          vbrcResetBitB Flag, 1
          End If
```

```
cc = 1
        For i = 0 To 15
            cc = (Not vbrcGetBitD(Regisy, i)) And 1 And cc
        Next
         If cc = 0 Then
         vbrcResetBitB Flag, 2
         Else
          vbrcSetBitB Flag, 2
         End If

         Case "7A", "6A", "186A":
          Data = Data - 1
          cc = 1
        For i = 0 To 6
            cc = (Not vbrcGetBitD(Data, i)) And 1 And cc
        Next
          If Not (cc) = 0 Then
         vbrcResetBitB Flag, 1
         Else
          vbrcSetBitB Flag, 1
         End If

         If vbrcGetBitD(Data, 7) Then
          vbrcSetBitB Flag, 3
         Else
         vbrcResetBitB Flag, 3
         End If
        cc = (Not vbrcGetBitD(Data, 7)) And 1 And cc
           If cc = 0 Then
         vbrcResetBitB Flag, 2
         Else
          vbrcSetBitB Flag, 2
         End If

         Case "1809":
         Regisy = Regisy - 1
           cc = 1
           For i = 0 To 15
           cc = (Not vbrcGetBitD(Regisy, i)) And 1 And cc
           Next
           If cc = 0 Then
              vbrcResetBitB Flag, 2
           Else
              vbrcSetBitB Flag, 2
           End If
         'กลุ่มบวก
         If vbrcGetBitD(ss, 7) Then
                vbrcSetBitB Flag, 3
              Else
                vbrcResetBitB Flag, 3
              End If
                B(1) = vbrcGetBitD(RegisA, 3) And vbrcGetBitD
(Data, 3) Or vbrcGetBitD(Data, 3) And Not (vbrcGetBitD(ss, 3)) Or
Not (vbrcGetBitD(ss, 3)) And vbrcGetBitD(RegisA, 3)
                If B(1) Then
                  vbrcSetBitB Flag, 5
                Else
                  vbrcResetBitB Flag, 5
                End If
                cc = 1
                For i = 0 To 7
                cc = (Not vbrcGetBitD(ss, i)) And 1 And cc
                Next
                If cc = 0 Then
                vbrcResetBitB Flag, 2
                Else
                vbrcSetBitB Flag, 2
                End If
                   B(1) = vbrcGetBitD(RegisA, 7) And vbrcGetBitD
(Data, 7) And Not (vbrcGetBitD(ss, 7)) Or Not (vbrcGetBitD(RegisA,
7)) And Not (vbrcGetBitD(Data, 7)) And vbrcGetBitD(ss, 7)
                If B(1) Then
                  vbrcSetBitB Flag, 1
                Else
                  vbrcResetBitB Flag, 1
                End If
                  B(1) = vbrcGetBitD(RegisA, 7) And vbrcGetBitD
(Data, 7) Or vbrcGetBitD(Data, 7) And Not (vbrcGetBitD(ss, 7)) Or
Not (vbrcGetBitD(ss, 7)) And vbrcGetBitD(RegisA, 7)
                 If B(1) Then
                   vbrcSetBitB Flag, 0
                Else
                  vbrcResetBitB Flag, 0
                End If
                  RegisA = ss

            Case "CB", "DB", "FB", "EB", "18EB"
                ss = Regisb + Data
               If vbrcGetBitD(ss, 7) Then
                   vbrcSetBitB Flag, 3
               Else
                vbrcResetBitB Flag, 3
```

```
B(1) = vbrcGetBitD(Regisb, 3) And vbrcGetBitD(Data, 3) Or
vbrcGetBitD(Data, 3) And Not (vbrcGetBitD(ss, 3)) Or Not
(vbrcGetBitD(ss, 3)) And vbrcGetBitD(Regisb, 3)
        If B(1) Then
          vbrcSetBitB Flag, 5
        Else
           vbrcResetBitB Flag, 5
        End If
           cc = 1
        For i = 0 To 7
           cc = (Not vbrcGetBitD(ss, i)) And 1 And cc
        Next
        If cc = 0 Then
           vbrcResetBitB Flag, 2
        Else
           vbrcSetBitB Flag, 2
        End If
           B(1) = vbrcGetBitD(Regisb, 7) And vbrcGetBitD(Data,
7) And Not (vbrcGetBitD(ss, 7)) Or Not (vbrcGetBitD(Regisb, 7))
And Not (vbrcGetBitD(Data, 7)) And vbrcGetBitD(ss, 7)
        If B(1) Then
           vbrcSetBitB Flag, 1
        Else
           vbrcResetBitB Flag, 1
        End If
           B(1) = vbrcGetBitD(Regisb, 7) And vbrcGetBitD(Data,
7) Or vbrcGetBitD(Data, 7) And Not (vbrcGetBitD(ss, 7)) Or Not
(vbrcGetBitD(ss, 7)) And vbrcGetBitD(Regisb, 7)
        If B(1) Then
           vbrcSetBitB Flag, 0
        Else
           vbrcResetBitB Flag, 0
        End If
          Regisb = ss

        Case "80", "90", "B0", "A0", "18A0":
              n = RegisA - Data
              If vbrcGetBitD(n, 7) Then
              vbrcSetBitB Flag, 3
              Else
              vbrcResetBitB Flag, 3
              End If
              cc = 1
              For i = 1 To 7
              cc = cc And Not (vbrcGetBitD(n, i))
              Next i
If vbrcGetBitD(Flag, 0) Then
              cc = 1
              Else
              cc = 0
              End If
               n = Regisb - Data - cc
              If vbrcGetBitD(n, 7) Then
              vbrcSetBitB Flag, 3
              Else
              vbrcResetBitB Flag, 3
              End If
             cc = 1
              For i = 1 To 7
              cc = cc And Not (vbrcGetBitD(n, i))
              Next i
              If cc Then
              vbrcSetBitB Flag, 2
              Else
              vbrcResetBitB Flag, 2
              End If
              cc = vbrcGetBitD(Regisb, 7) And Not
vbrcGetBitD(Data, 7) And Not vbrcGetBitD(n, 7) Or Not
vbrcGetBitD(Regisb, 7) And vbrcGetBitD(Data, 7) And vbrcGetBitD
(n, 7)
             If cc Then
              vbrcSetBitB Flag, 1
              Else
              vbrcResetBitB Flag, 1
              End If
               cc = Not vbrcGetBitD(Regisb, 7) And
vbrcGetBitD(Data, 7) Or vbrcGetBitD(Data, 7) And vbrcGetBitD(n,
7) Or vbrcGetBitD(n, 7) And Not vbrcGetBitD(Regisb, 7)
             If cc Then
              vbrcSetBitB Flag, 0
              Else
              vbrcResetBitB Flag, 0
              End If
            Regisb = n
      Case "83", "93", "B3", "A3", "18A3":
        n = Regisd - Data

        If n > 255 Then
        RegisA = (n - 255) / 15
        Regisb = n - RegisA
        Else
        RegisA = 0
```

```
vbrcSetBitB Flag, 3
                Else
                  vbrcResetBitB Flag, 3
                End If
                cc = 1
                For i = 1 To 15
                cc = cc And Not (vbrcGetBitD(n, i))
                Next i
                If cc Then
                vbrcSetBitB Flag, 2
                Else
                vbrcResetBitB Flag, 2
                End If
                cc = vbrcGetBitD(Regisd, 15) And Not
vbrcGetBitD(Data, 15) And Not vbrcGetBitD(n, 15) Or Not
vbrcGetBitD(Regisd, 15) And vbrcGetBitD(Data, 15) And
vbrcGetBitD(n, 15)
              If cc Then
                vbrcSetBitB Flag, 1
                Else
                vbrcResetBitB Flag, 1
                End If
                 cc = Not vbrcGetBitD(Regisd, 15) And
vbrcGetBitD(Data, 15) Or vbrcGetBitD(Data, 15) And vbrcGetBitD
(n, 15) Or vbrcGetBitD(n, 15) And Not vbrcGetBitD(Regisd, 15)
              If cc Then
                vbrcSetBitB Flag, 0
                Else
                vbrcResetBitB Flag, 0
                End If
             Regisd = n

        'กลุ่มคำสั่งทางตรรกะ

        Case "8A", "9A", "BA", "AA", "18AA":
            RegisA = Hex(RegisA) Or Hex(Data)
        vbrcResetBitB Flag, 1
        If vbrcGetBitD(RegisA, 7) Then
          vbrcSetBitB Flag, 3
        Else
          vbrcResetBitB Flag, 3
        End If
        cc = 1
        For i = 0 To 7
          cc = cc And Not (vbrcGetBitD(RegisA, i))
        Next i
Else
          vbrcResetBitB Flag, 2
        End If
      Case "CA", "DA", "FA", "EA", "18EA":
        Regisb = Hex(Regisb) Or Hex(Data)
        vbrcResetBitB Flag, 1
        If vbrcGetBitD(Regisb, 7) Then
          vbrcSetBitB Flag, 3
        Else
          vbrcResetBitB Flag, 3
        End If
        cc = 1
        For i = 0 To 7
          cc = cc And Not (vbrcGetBitD(Regisb, i))
        Next i
          If cc Then
          vbrcSetBitB Flag, 2
        Else
          vbrcResetBitB Flag, 2
        End If
      Case "88", "98", "B8", "A8", "18A8":
        RegisA = Hex(RegisA) Xor Hex(Data)
        vbrcResetBitB Flag, 1 'V=0
        If vbrcGetBitD(RegisA, 15) Then 'N
          vbrcSetBitB Flag, 3
        Else
          vbrcResetBitB Flag, 3
        End If
          cc = 1 'Z
          For i = 0 To 7
          cc = cc And Not vbrcGetBitD(RegisA, i)
          Next i

      Case "C8", "D8", "F8", "E8", "18E8":
        Regisb = Hex(Regisb) Xor Hex(Data)
        vbrcResetBitB Flag, 1 'V=0
        If vbrcGetBitD(REGIRSB, 15) Then 'N
          vbrcSetBitB Flag, 3
        Else
          vbrcResetBitB Flag, 3
        End If
          cc = 1 'Z
          For i = 0 To 7
          cc = cc And Not vbrcGetBitD(Regisb, i)
          Next i
```

```
vbrcSetBitB Flag, 3
                    Else
                      vbrcResetBitB Flag, 3
                    End If
                    cc = 1
                    For i = 1 To 15
                    cc = cc And Not (vbrcGetBitD(n, i))
                    Next i
                    If cc Then
                    vbrcSetBitB Flag, 2
                    Else
                    vbrcResetBitB Flag, 2
                    End If
                    cc = vbrcGetBitD(Regisd, 15) And Not
vbrcGetBitD(Data, 15) And Not vbrcGetBitD(n, 15) Or Not
vbrcGetBitD(Regisd, 15) And vbrcGetBitD(Data, 15) And
vbrcGetBitD(n, 15)
                  If cc Then
                    vbrcSetBitB Flag, 1
                    Else
                    vbrcResetBitB Flag, 1
                    End If
                     cc = Not vbrcGetBitD(Regisd, 15) And
vbrcGetBitD(Data, 15) Or vbrcGetBitD(Data, 15) And vbrcGetBitD
(n, 15) Or vbrcGetBitD(n, 15) And Not vbrcGetBitD(Regisd, 15)
                  If cc Then
                    vbrcSetBitB Flag, 0
                    Else
                    vbrcResetBitB Flag, 0
                    End If
                 Regisd = n

        'กลุ่มคำสั่งทางตรรกะ

       Case "8A", "9A", "BA", "AA", "18AA":
          RegisA = Hex(RegisA) Or Hex(Data)
       vbrcResetBitB Flag, 1
       If vbrcGetBitD(RegisA, 7) Then
         vbrcSetBitB Flag, 3
       Else
         vbrcResetBitB Flag, 3
       End If
       cc = 1
       For i = 0 To 7
         cc = cc And Not (vbrcGetBitD(RegisA, i))
       Next i
Else
         vbrcResetBitB Flag, 2
       End If
     Case "CA", "DA", "FA", "EA", "18EA":
       Regisb = Hex(Regisb) Or Hex(Data)
       vbrcResetBitB Flag, 1
       If vbrcGetBitD(Regisb, 7) Then
         vbrcSetBitB Flag, 3
       Else
         vbrcResetBitB Flag, 3
       End If
       cc = 1
       For i = 0 To 7
         cc = cc And Not (vbrcGetBitD(Regisb, i))
       Next i
         If cc Then
         vbrcSetBitB Flag, 2
       Else
         vbrcResetBitB Flag, 2
       End If
     Case "88", "98", "B8", "A8", "18A8":
       RegisA = Hex(RegisA) Xor Hex(Data)
       vbrcResetBitB Flag, 1 'V=0
       If vbrcGetBitD(RegisA, 15) Then 'N
         vbrcSetBitB Flag, 3
       Else
         vbrcResetBitB Flag, 3
       End If
         cc = 1 'Z
         For i = 0 To 7
         cc = cc And Not vbrcGetBitD(RegisA, i)
         Next i

     Case "C8", "D8", "F8", "E8", "18E8":
       Regisb = Hex(Regisb) Xor Hex(Data)
       vbrcResetBitB Flag, 1 'V=0
       If vbrcGetBitD(REGIRSB, 15) Then 'N
         vbrcSetBitB Flag, 3
       Else
         vbrcResetBitB Flag, 3
       End If
         cc = 1 'Z
         For i = 0 To 7
         cc = cc And Not vbrcGetBitD(Regisb, i)
       Next i
```

```
If vbrcGetBitD(result%, 7) Then
            vbrcSetBitB Flag, 3
          Else
            vbrcResetBitB Flag, 3
          End If
          cc = 1
          For i = 0 To 7
          cc = cc And Not vbrcGetBitD(result%, i)
          Next i
          If cc Then
             vbrcSetBitB Flag, 2
          Else
            vbrcResetBitB Flag, 2
          End If
              vbrcResetBitB Flag, 1

         Case "C5", "D5", "F5", "E5", "18E5":
            result% = Regisb And Hex(Data)
         If vbrcGetBitD(result%, 7) Then
            vbrcSetBitB Flag, 3
          Else
            vbrcResetBitB Flag, 3
          End If
          cc = 1
          For i = 0 To 7
          cc = cc And Not vbrcGetBitD(result%, i)
          Next i
          If cc Then
             vbrcSetBitB Flag, 2
          Else
            vbrcResetBitB Flag, 2
          End If
              vbrcResetBitB Flag, 1

         Case "1A83", "1A93", "1AB3", "1AA3", "CDA3":
            result% = Regisd - Data
            If vbrcGetBitD(result%, 7) Then
              vbrcSetBitB Flag, 3
            Else
              vbrcResetBitB Flag, 3
            End If
            cc = 1
            For i = 0 To 15
            cc = Not vbrcGetBitD(result%, i) And cc
            Next i
            If cc Then
vbrcResetBitB Flag, 2
            End If
            cc = vbrcGetBitD(Regisx, 15) And Not vbrcGetBitD
(Data, 15) And Not vbrcGetBitD(result%, 15) Or Not vbrcGetBitD
(Regisd, 15) And vbrcGetBitD(Data, 15) And vbrcGetBitD
(result%, 15)
            If cc Then
              vbrcSetBitB Flag, 1
            Else
              vbrcResetBitB Flag, 1
            End If
            cc = Not vbrcGetBitD(Regisx, 15) And vbrcGetBitD
(Data, 15) Or vbrcGetBitD(Data, 15) And vbrcGetBitD(result%,
15) Or vbrcGetBitD(result%, 15) And Not vbrcGetBitD(Regisd,
15)
            If cc Then
              vbrcSetBitB Flag, 0
            Else
              vbrcResetBitB Flag, 0
            End If
          Case "188C", "189C", "18BC", "1AAC", "18AC":
            result% = Regisy - Hex(Data)
          Case "7E", "6E", "186E":
            PC = RAM(Data)
          Case "9D", "BD", "AD", "18AD":
             If Data > 255 Then
             PCH = (Data - 255) / 15
             PCL = Data - PCH
             Else
             PCH = 0
             PCL = Data
            End If
            RAM(SP) = PCL
            RAM(SP - 1) = PCH
            SP = SP - 2

         Case "20":
            PC = PC + 2 + Data
        Case "8D"
          ' PC = PC + 2
            If Data > 255 Then
              PCH = (Data - 255) / 15
              PCL = Data - PCH
              Else
              PCH = 0
              PCL = Data
```

```
RAM(SP) = PCH
            SP = SP - 1
            PC = PC + Hex(Data)
        Case "39"
            PC = SP
        Case "21":
            PC = PC + 3

    End Select
End Sub
Sub Runtime()
    Dim n, num, filenum%, bytestr, InstrOrder As Long
    Dim OBJfilename, Inputdata, OPCODE, instrByte, CountCode As
String
    Dim varBookmark As Variant
    Dim MyByte As Byte
    Dim cc As Integer
    Dim MyInsByte As Long
    Dim Data(7) As Long
    Dim G As Long
    Dim B As Long
    Dim STARTBYTE As Long
    Dim LASTBYTE As Long
    Dim instxt As Integer
    Dim test As Byte
    Dim PCH As Long
    Dim a As Long
    Dim L, S As Long
    Dim i As Integer
    Dim temp As Byte
    If Compileflag = False Then Call Compile_asm 'Compile_asm ทำ
การคอมไพล์ก่อนถ้ายังไม่คอมไพล์
      If Stoped = False Then
'****************************************** เฟทคำสั่งมาเก็บใน
instruction รีจิสเตอร์
        Call FetchIR
        S = 0
         For G = 0 To 65535
            If RAM(G) <> MEM(G) Then
              PCH = G
              S = S + 1
               If (PCH >= 0) And (PCH <= 255) Then
                  intxt = 1
                  STARTBYTE = 0
                  LASTBYTE = 255
               End If
STARTBYTE = 8192
                  LASTBYTE = 10239
               End If
               If (PCH >= 40960) And (PCH <= 45055) Then
                  intxt = 3
                  STARTBYTE = 40960
                  LASTBYTE = 45055
               End If
               If (PCH >= 46592) And (PCH <= 47103) Then
                  intxt = 4
                  STARTBYTE = 46592
                  LASTBYTE = 47103
               End If
               If (PCH >= 49152) And (PCH <= 57343) Then
                  intxt = 5
                  STARTBYTE = 49152
                  LASTBYTE = 57343
               End If
               If (PCH >= 57344) And (PCH <= 65535) Then
                  intxt = 6
                  STARTBYTE = 57344
                  LASTBYTE = 65535
               End If
'นำข้อมูลใน Ram(64K) มาแสดง ในฟอร์ม Memory
               L = (LASTBYTE - STARTBYTE) / 15
               For B = 0 To L
                  Screen.MousePointer = 11
                  If STARTBYTE < LASTBYTE Then
                     For a = 0 To 15
                        BUF$ = BUF$ & "  " & Right(String
(2, "0") & Hex(RAM(STARTBYTE + a)), 2)
                     Next a
                        strBuf$ = strBuf$ & "[" & Right
(String(3, "0") & Hex(STARTBYTE), 4) & "]-" & BUF & Chr(13)
& Chr(10)
                        STARTBYTE = STARTBYTE + 16
                        BUF$ = ""

                  End If
               Next B
               If intxt <> 0 Then
                  Screen.MousePointer = 0 'กำหนดเมาส์ใน
สภาวะปกติ

                  Memo.txtmem.Item(intxt).Text = strBuf$
               End If
            End If
```

```
Memo.Show
 End If

        If ListPROG.List1.SelCount <> 0 Then
          If PC = 0 Then
          ListPROG.List1.Selected(0) = True
          End If
       End If

'End If
   ins = ins + 1
        '*************************************************
เพิ่มค่า program
counter*************************************************
***************
       If (ins = 0) Then
            ListPROG.List1.Selected(0) = True
       End If
       If (ins > 0 And (ins < ListPROG.List1.ListCount)) Then
            ListPROG.List1.Selected(ins) = True
       End If
      ShowRegis
End Sub

Regis
Dim i As Long
Dim speed As Long
Dim Toggle As Boolean

Private Sub CmdRESET_Click()
   Stoped = False
   PC = 0
   SP = 255 '&HCFFF
   ins = 0
   Flag = Flag And &H0
   Regisb = 0
   Regisd = 0
   Regisx = 0
   Regisy = 0
   RegisA = 0
   'ListPROG.List1.Selected(0) = True
   ShowRegis
End Sub

Private Sub Cmdstart_Click()
   Dim tmpAS As Integer
STARTBYTE = 8192
            LASTBYTE = 10239
          End If
          If (PCH >= 40960) And (PCH <= 45055) Then
            intxt = 3
            STARTBYTE = 40960
            LASTBYTE = 45055
          End If
          If (PCH >= 46592) And (PCH <= 47103) Then
            intxt = 4
            STARTBYTE = 46592
            LASTBYTE = 47103
          End If
          If (PCH >= 49152) And (PCH <= 57343) Then
            intxt = 5
            STARTBYTE = 49152
            LASTBYTE = 57343
          End If
          If (PCH >= 57344) And (PCH <= 65535) Then
            intxt = 6
            STARTBYTE = 57344
            LASTBYTE = 65535
          End If
          'นำข้อมูลใน Ram(64K) มาแสดง ในฟอร์ม Memory
          L = (LASTBYTE - STARTBYTE) / 15
          For B = 0 To L
            Screen.MousePointer = 11
            If STARTBYTE < LASTBYTE Then
              For a = 0 To 15
                BUF$ = BUF$ & "  " & Right(String
(2, "0") & Hex(RAM(STARTBYTE + a)), 2)
              Next a
                strBuf$ = strBuf$ & "[" & Right
(String(3, "0") & Hex(STARTBYTE), 4) & "]-" & BUF & Chr(13)
& Chr(10)
                STARTBYTE = STARTBYTE + 16
                BUF$ = ""

            End If
          Next B
          If intxt <> 0 Then
            Screen.MousePointer = 0 'กำหนดเมาส์ใน
สภาวะปกติ
            Memo.txtmem.Item(intxt).Text = strBuf$
          End If
        End If
```

```
For SimPrC = 0 To tmpAS
      If Stoped = True Then
        SimPrC = tmpAS + 1
      Else
        CmdStep_Click
        'Regis.Show
        Cmdstop.SetFocus
      End If
      DoEvents
   Next SimPrC
   Timer1.Interval = 1000
   Timer1.Enabled = True
End Sub

Private Sub CmdStep_Click()
   Call Runtime
    Screen.MousePointer = 0 'กำหนดเมาส์ในสภาวะปกติ

End Sub

Private Sub cmdstop_Click()
   Stoped = True
End Sub

Private Sub Form_Load()
    Dim i As Byte
   Toggle = True
   Me.Move 1441 * 3 + 20, 0, Me.Width, Me.Height

   PC = 0
   SP = 255 '&HCFFF
   ins = 0
   Flag = Flag And &H0
   Regisb = 0
   Regisd = 0
   Regisx = 0
   Regisy = 0
   RegisA = 0
  ' ListPROG.List1.Selected(0) = True
   ShowRegis

  CmdRESET_Click
   Regis.Refresh
   Timer1.Interval = 1000
   Timer1.Enabled = True
End Sub

Slider1_Click
End Sub

Private Sub Slider1_Click()
      If speed < 1 Then speed = 1
      CLOCK = Slider1.Value * 100 / speed
      Timer1.Interval = CLOCK
End Sub

'Private Sub Timer1_Timer()
'     ShowRegis
'End Sub
```

บรรณานุกรม

กิตติ ภักดีวัฒนะกุล และ จำลอง ครุอุตสาหะ. “ **Visual Basic 6.0 ฉบับโปรแกรมเมอร์ 6**”. พิมพ์ครั้งที่ 3. กรุงเทพฯ : เคทีพี คอมพ์ แอนด์ คอนซัลน์ จำกัด,2540.

ชัยวัฒน์ ลิ้มพรจิตรวิลัย. **“ไมโครคอนโทรลเลอร์ 68HC11”.** กรุงเทพฯ : ซีเอ็ดยูเคชั่น จำกัด,2538.

บริษัท โมโตโรล่า จำกัด.“**คู่มือการใช้งานไมโครคอนโทรลเลอร์ HC11 MC68HC11A8 TECHNICAL DATA”.** กรุงเทพฯ : โมโตโรล่า จำกัด,2534.

สุชาย ธนวเสถียร. **“การเขียนโปรแกรมด้วยวิชวลเบสิก 6.0”.** กรุงเทพฯ : Sum Publishing, 2540.

สุทธิศักดิ์ พงศ์ธนาพาณิช. **“การเขียนโปรแกรมด้วย Visual Basic 6.0 ระดับสูง การใช้งานฟังก์ชันวินโดวส์ API-32 บิต”.** กรุงเทพฯ : 2540.

วิบูลย์ ชื่นแขก. “ **ไมโครโปรเซสเซอร์**”. กรุงเทพฯ : สถานบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ, 2525.

ประวัติผู้แต่ง

ชื่อผู้ทำปริญญานิพนธ์	นางสาวพนัตฎาร์ ผิวผ่อง
วัน เดือน ปีเกิด	14 ตุลาคม พ.ศ. 2521
สถานที่เกิด	จังหวัด กาฬสินธุ์
ภูมิลำเนาเดิม	174 หมู่ 10 ตำบล สมเด็จ อำเภอสมเด็จ จังหวัด กาฬสินธุ์ 46150
ที่อยู่ปัจจุบัน	174 หมู่ 10 ตำบล สมเด็จ อำเภอสมเด็จ จังหวัด กาฬสินธุ์ 46150
โทรศัพท์	(043) 823575

ประวัติการศึกษา

ประถมศึกษา	โรงเรียนบ้านสี่แยกสมเด็จ
มัธยมศึกษาตอนต้น	โรงเรียนกาฬสินธุ์พิทยาสรรพ์
มัธยมศึกษาตอนปลาย	โรงเรียนกาฬสินธุ์พิทยาสรรพ์
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	สถาบันเทคโนโลยีราชมงคล วิทยาเขตขอนแก่น
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	ทุนผลการเรียนวิทยาศาสตร์คะแนนสูงสุด
คติพจน์	ไม่มีใครได้อะไรมาง่ายๆ

ประวัติผู้แต่ง

ชื่อผู้ทำปริญญานิพนธ์	นางสาวรุ่งทิพย์ ดวงกลาง
วัน เดือน ปีเกิด	24 มกราคม พ.ศ. 2522
สถานที่เกิด	จังหวัด นครราชสีมา
ภูมิลำเนาเดิม	10 หมู่ 5 ตำบล ขามเฒ่า อำเภอโนนสูง จังหวัดนครราชสีมา 30160
ที่อยู่ปัจจุบัน	10 หมู่ 5 ตำบล ขามเฒ่า อำเภอโนนสูง จังหวัดนครราชสีมา 30160
โทรศัพท์	(044) 383396

ประวัติการศึกษา

ประถมศึกษา	โรงเรียนไตรคามสามัคคี
มัธยมศึกษาตอนต้น	โรงเรียนโนนสูงศรีธานี
มัธยมศึกษาตอนปลาย	โรงเรียนโนนสูงศรีธานี
ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.)	สถาบันเทคโนโลยีราชมงคล วิทยาเขตภาคตะวันออกเฉียงเหนือ
ปริญญาตรี	สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
ผลงานที่ได้รับรางวัล	-
ทุนการศึกษา	-
คติพจน์	ใจเป็นนาย กายเป็นบ่าว