
**ภาควิชาครุศาสตร์วิศวกรรม**

**คณะครุศาสตร์อุตสาหกรรม**

**สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง**

**ใบรับรองปริญญานิพนธ์**

**ชื่อหัวข้อ** โปรแกรมจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80

Z-80 Simulator Program

**ชื่อนักศึกษา**

| | | | |
|---|---|---|---|
| 1. นายณรงค์ | สังขมาลัย | รหัสประจำตัว | 41031408 |
| 2. นายณัฐพงศ์ | ชิดพรมราช | รหัสประจำตัว | 41031409 |
| 3. นางสาวนิตยา | แผ่นใหญ่ | รหัสประจำตัว | 41031411 |

**หลักสูตร** ครุศาสตร์อุตสาหกรรมบัณฑิต **สาขาวิชา** อิเล็กทรอนิกส์และคอมพิวเตอร์

**อาจารย์ที่ปรึกษา** อาจารย์ไพบูลย์ พวงวงศ์ตระกูล

| คณะกรรมการสอบปริญญานิพนธ์ | | ลายมือชื่อ |
|---|---|---|
| 1. อาจารย์ไพบูลย์ | พวงวงศ์ตระกูล | ..... |
| 2. อาจารย์สุชิน | อาจหาญ | ..... |
| 3. อาจารย์อำพล | ทองระอา | ..... |
| 4. อาจารย์สุระชัย | พิมพ์สาลี | ..... |

**วัน/เดือน/ปีที่สอบ** วันเสาร์ที่ 13 พฤษภาคม พ.ศ. 2543 เวลา 13.00 น.

**สถานที่สอบ** ห้อง ค.301 คณะครุศาสตร์อุตสาหกรรม สจล.

ภาควิชารับรองแล้ว

ลงนาม.....

(ผศ.วิสุทธิ์ อธิพรธรรม)

หัวหน้าภาควิชาครุศาสตร์วิศวกรรม

วันที่ 3 เดือน ก.ค. พ.ศ. ....

# ปริญญานิพนธ์

## โปรแกรมจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80

## Z-80 SIMULATOR PROGRAM

นายณรงค์ สังขมาลัย
นายณัฐพงศ์ ชิดพรมราช
นางสาวนิตยา แผ่นใหญ่

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรครุศาสตร์อุตสาหกรรมบัณฑิต
สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์
ภาควิชาครุศาสตร์วิศวกรรม คณะครุศาสตร์อุตสาหกรรม
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2542

# ปริญญานิพนธ์

เรื่อง โปรแกรมจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80

Z-80 Simulator Program

## วัตถุประสงค์

1. ศึกษาการเขียนโปรแกรมบนวินโดวส์ด้วยวิชวลเบสิก
2. ศึกษารูปแบบคำสั่งและการทำงานของไมโครโปรเซสเซอร์ Z-80
3. ออกแบบโปรแกรมจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80
4. สร้างโปรแกรมจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80
5. ทดลองการใช้งานโปรแกรมจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80
6. นำเอาไปใช้ในการเรียนการสอนวิชาไมโครโปรเซสเซอร์ Z-80

## ประโยชน์ที่คาดว่าจะได้รับ

1. ได้รับความรู้ทักษะจากการเขียนโปรแกรมบนวินโดว์ด้วยวิชวลเบสิก
2. ได้รับความรู้จากการศึกษารูปแบบคำสั่งและการทำงานของไมโครโปรเซสเซอร์ Z-80
3. ได้รับความรู้ในการออกแบบโปรแกรมจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80
4. ได้รับความรู้ในสร้างโปรแกรมจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80
5. ทดลองการใช้งานโปรแกรมจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80
6. สามารถนำเอาโปรแกรมจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80 ไปใช้ในการเรียนการสอนวิชาไมโครโปรเซสเซอร์ Z-80 ได้จริง

# กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี เนื่องมาจากการได้รับความอนุเคราะห์ และช่วยเหลือทางด้านอุปกรณ์ เครื่องมือ ที่ใช้ในการปฏิบัติงาน และคำปรึกษาทางด้านเทคนิคจากท่านอาจารย์ที่ปรึกษาปริญญานิพนธ์ ที่คอยชี้แนะ รวมทั้งอาจารย์ภาควิชาครุศาสตร์วิศวกรรมทุกท่านที่คอยให้กำลังใจอยู่ตลอดเวลา ขอขอบพระคุณเพื่อนร่วมงานทุกคนภายในกลุ่มที่ร่วมแรงร่วมใจประสานความสามัคคีจนทำให้โครงงานนี้ประสบผลสำเร็จ และเพื่อน ๆ ร่วมชั้นเรียน ที่คอยให้ความช่วยเหลือทางด้านอุปกรณ์ และเครื่องมือเครื่องใช้ต่าง ๆ และที่สำคัญที่สุด ความสำเร็จครั้งนี้ขอมอบให้กับบุคคลที่ทำให้ได้รับการศึกษาตั้งแต่อดีตจนถึงปัจจุบัน คือ คุณพ่อ คุณแม่ ครู อาจารย์ ญาติพี่น้อง และผู้มีพระคุณทุกท่าน

| | |
|---|---|
| **ชื่อหัวข้อ** | โปรแกรมจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80 |
| **นักศึกษา** | นายณรงค์ สังขมาลัย |
| | นายณัฐพงศ์ ชิดพรมราช |
| | นางสาวนิตยา แผ่นใหญ่ |
| **อาจารย์ที่ปรึกษา** | อาจารย์ไพบูลย์ พวงวงศ์ตระกูล |
| **หลักสูตร** | ครุศาสตร์อุตสาหกรรมบัณฑิต |
| **สาขาวิชา** | อิเล็กทรอนิกส์และคอมพิวเตอร์ |
| **ปีการศึกษา** | 2542 |


**บทคัดย่อ**

ปริญญานิพนธ์ฉบับนี้นำเสนอโปรแกรมจำลองการทำงานไมโครโปรเซสเซอร์ Z-80 โดยมีคุณสมบัติเฉพาะของโปรแกรมดังนี้ สามารถจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80 ได้, สามารถแสดงค่าของรีจิสเตอร์ และสถานะแฟลกต่าง ๆ ได้, จำลองการทำงานของอุปกรณ์ต่อร่วมเพื่อการติดต่อได้, แสดงค่าข้อมูลของหน่วยความจำที่ตำแหน่งต่าง ๆ ได้ และมีการทำงานภายใต้ระบบปฏิบัติการวินโดวส์

| | |
|---|---|
| **Thesis Title** | Z-80 Simulator Program |
| **Student** | Mr. Narong Sungkhamalai |
| | Mr. Nathapong Chidpromrach |
| | Miss. Nittaya Panyai |
| **Advisor** | Mr. Paiboon Pongwongtragull |
| **Education Level** | Bachelor of Science in Industrial Education |
| **Program in** | Electronics and Computer |
| **Academic Year** | 1999 |


## ABSTRACT

This thesis presents the Z-80 simulator. The properties of a software thus able simulation Z-80 microprocessor, able to show register and status flag, simulation of board interface, show data of memory at any position and use by windows operating system.

# สารบัญ

# สารบัญ (ต่อ)

# สารบัญ (ต่อ)

# สารบัญ (ต่อ)

# สารบัญตาราง

# สารบัญรูป

# สารบัญรูป(ต่อ)

# สารบัญรูป(ต่อ)

# สารบัญรูป(ต่อ)

# บทที่ 1
# บทนำ

## 1.1 ความเป็นมา และความสำคัญของปริญญานิพนธ์

ในปัจจุบันได้มีการใช้เครื่องจักรมาควบคุมการทำงานต่างๆ มากมาย ไม่ว่าจะเป็นการควบคุมการผลิตให้มีประสิทธิภาพทั้งในด้านปริมาณและคุณภาพของสินค้า อุปกรณ์ที่ใช้ในการควบคุมที่นิยมมาใช้งานก็คือ ไมโครโปรเซสเซอร์ ซึ่งในบรรดาไมโครโปรเซสเซอร์นั้นมีอยู่หลายเบอร์ แต่ไมโครโปรเซสเซอร์ที่เป็นพื้นฐานคือ ไมโครโปรเซสเซอร์ Z-80 ซึ่งในระบบการเรียนการสอนได้มีการศึกษาการทำงานไมโครโปรเซสเซอร์ Z-80 แต่เนื่องจากในการศึกษาไมโครโปรเซสเซอร์ Z-80 นั้นจะมีการทดลองโปรแกรมการทำงานของไมโครโปรเซสเซอร์ โดยการต่อเข้ากับอุปกรณ์ต่างๆ อยู่เสมอ จึงทำให้ระบบการเรียนการสอนนั้นยุ่งยาก และใช้เวลาในการทดลองเป็นเวลานาน และในบางครั้งการใช้ชุดทดลองจริงนั้นก็มักจะมีปัญหาจากอุปกรณ์ชำรุดเสียหาย

ดังนั้นคณะผู้จัดทำจึงได้จัดทำโปรแกรมจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80 ขึ้นเพื่อใช้ศึกษาการทำงานของไมโครโปรเซสเซอร์ Z-80 ซึ่งโปรแกรมจำลองการทำงาน Z-80 นี้จะช่วยให้ผู้ที่สนใจเข้าใจหลักการทำงานของไมโครโปรเซสเซอร์ Z-80 ได้ง่ายยิ่งขึ้น

## 1.2 ขีดความสามารถของโครงงาน

โครงงานนี้มีความสามารถดังนี้

1) ใช้ประกอบการเรียนการสอนในวิชาไมโครโปรเซสเซอร์ Z-80 ได้
2) มีการจำลองคำสั่งในการทำงาน
3) มีการจำลองหน่วยความจำ
4) มีการจำลองรีจิสเตอร์
5) มีภาคแสดงผลจำลองแอลอีดีเมทริกซ์ 1 ชุด
6) มีภาคแสดงผลจำลองแบบเจ็ดส่วน 1 ชุด
7) มีการจำลองการทำงานของมอเตอร์ไฟตรง
8) มีการจำลองการทำงานของสเต็ปปิ้งมอเตอร์

## 1.3 เนื้อหาโดยสังเขป

เนื้อหาภายในปริญญานิพนธ์ฉบับนี้ได้แบ่งออกเป็นบทต่างๆ เพื่อความสะดวกต่อการศึกษาและทำความเข้าใจ โดยในแต่ละบทจะประกอบด้วยเนื้อหาดังนี้

บทที่ 2 ทฤษฎี และหลักการ ประกอบด้วยเนื้อหาดังนี้ คือ สถาปัตยกรรมของไมโครโปรเซสเซอร์ Z-80 ชุดคำสั่ง การเขียนโปรแกรมด้วยวิชวลเบสิก การประมวลระดับบิตด้วย Bits32.dll อุปกรณ์เชื่อมต่อกับไมโครโปรเซสเซอร์ Z-80 ซึ่งจะทำให้ผู้อ่านได้มีความรู้ความเข้าใจ เพื่อเป็นพื้นฐานในการศึกษา และเพื่อเป็นประโยชน์ในการทำความเข้าใจกับการทำงานของโปรแกรมที่ใช้งานจริงต่อไป

บทที่ 3 การออกแบบและการสร้าง และการทำงาน ประกอบด้วยเนื้อหาดังนี้ คือ โครงสร้างหลักของโปรแกรม การออกแบบโปรแกรม การสร้างหน้าต่างใช้งาน การสร้างชุดการเชื่อมต่อจำลองการทำงาน การออกแบบชุดคำสั่ง เพื่อทำให้ผู้อ่านได้เข้าใจวิธีการในการออกแบบ และการสร้าง เพื่อการทำการพัฒนา หรือปรับปรุงโปรแกรมให้มีประสิทธิภาพเพิ่มขึ้นในโอกาสต่อไป

บทที่ 4 การทดลอง และผลการทดลอง กล่าวถึงขั้นตอนในการทำการทดลอง และผลการทดสอบโปรแกรม ในชุดคำสั่งของโปรแกรมในส่วนต่างๆ

บทที่ 5 บทสรุป ปัญหา แนวทางแก้ไข และพัฒนา ขั้นการสรุปผล ในการจัดทำโครงงาน ปัญหาที่เกิดขึ้น และได้เสนอแนวทางการแก้ไขปัญหา รวมทั้งแนวทางในการพัฒนาโปรแกรมให้มีประสิทธิภาพมากยิ่งขึ้น

ภาคผนวก ก ผังการทำงานและโปรแกรม

ภาคผนวก ข ใบงานการทดลองและเฉลยใบงานการทดลอง

ภาคผนวก ค คู่มือการใช้งานโปรแกรม

# บทที่ 2
# ทฤษฎีและหลักการ

## 2.1 กล่าวนำ

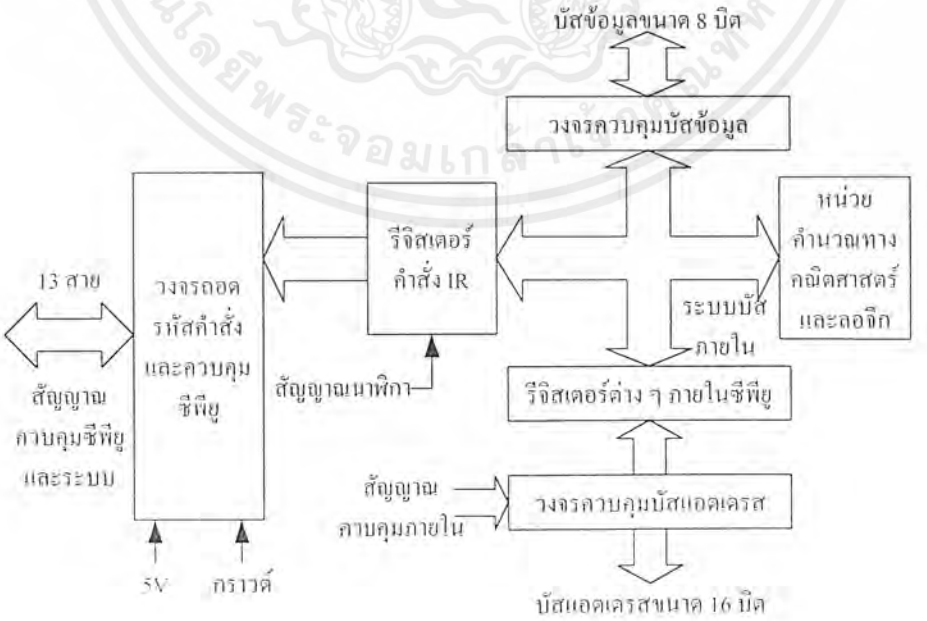
เนื้อหาของปริญญานิพนธ์ในบทนี้เป็นทฤษฎี และหลักการ ที่นำมาใช้ประกอบการสร้างโครงงานนี้ขึ้นมา ซึ่งประกอบด้วยทฤษฎี และหลักการเกี่ยวกับสถาปัตยกรรม และการทำงานของไมโครโปรเซสเซอร์ Z-80 คุณสมบัติและโครงสร้างของโปรแกรมวิชวลเบสิก (Visual Basic) ซึ่งหลักการเขียนโปรแกรมโดยใช้โปแกรมวิชวลเบสิก จะได้กล่าวถึงดังต่อไปนี้

## 2.2 สถาปัตยกรรมไมโครโปรเซสเซอร์ Z-80

ไมโครโปรเซสเซอร์ Z-80 เป็นไมโครโปรเซสเซอร์ที่มีจุดเริ่มต้นจากการพัฒนา และสร้างขึ้นของทีมวิศวกรบริษัทไซลอก (Zilog) ไมโครโปรเซสเซอร์ Z-80 เป็นไมโครโปรเซสเซอร์ขนาด 8 บิต ซึ่งมีลักษณะดังนี้

### 2.2.1 โครงสร้างทั่วไปของไมโครโปรเซสเซอร์ Z-80

ซึ่งลักษณะของโครงสร้างของไมโครโปรเซสเซอร์ Z-80 ดังแสดงในรูปที่ 2.1



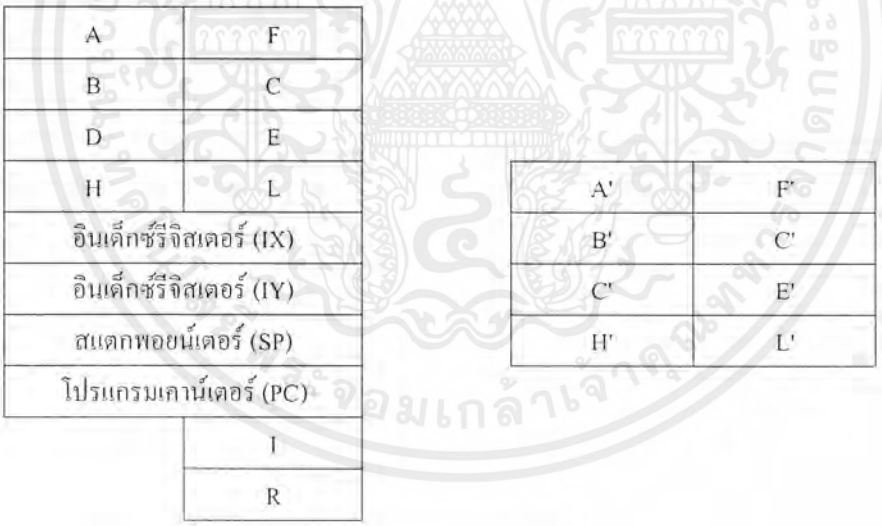

รูปที่ 2.1 โครงสร้างภายในของไมโครโปรเซสเซอร์ Z-80

โครงสร้างของ CPU Z-80 มีโครงสร้างที่พัฒนามาจาก 8080 ดังนั้นโครงสร้างพื้นฐานจะเหมือนกับซีพียู 8080 แต่เนื่องจากไมโครโปรเซสเซอร์ Z-80 มีการพัฒนามากขึ้นทางซอฟต์แวร์และฮาร์ดแวร์ จึงทำให้มีรายละเอียดแตกต่างและเพิ่มเติมอีกหลายประการด้วยกัน

จากรูปที่ 2.1 โครงสร้างภายในของไมโครโปรเซสเซอร์ Z-80 ประกอบด้วยบัสข้อมูลขนาด 8 บิต เป็นบัสชนิดสองทิศทาง ข้อมูลสามารถวิ่งเข้า หรือออกจากซีพียูได้ และบัสแอดเดรสซึ่งเป็นบัสขนาด 16 บิต จะมีขีดความสามารถในการอ้างถึงแอดเดรสได้โดยตรงถึง $2^{16}$ หรือ 64 กิโลไบต์ ซึ่งบัสแอดเดรสยังเป็นสายสำคัญในการอ้างถึงแอดเดรสของหน่วยที่เป็นอินพุต หรือเอาต์พุตด้วย

### 2.2.2 โครงสร้างภายในของไมโครโปรเซสเซอร์ Z-80

โครงสร้างภายในของไมโครโปรเซสอเซอร์ Z-80 ประกอบด้วย รีจิสเตอร์ภายในที่สามารถเขียน และอ่านข้อมูลได้ 208 บิต โดยแยกเป็นรีจิสเตอร์ขนาด 8 บิต มีทั้งหมด 18 รีจิสเตอร์ และมีรีจิสเตอร์ขนาด 16 บิต ทั้งหมด 4 รีจิสเตอร์ ซึ่งลักษณะโครงสร้างของรีจิสเตอร์ภายในทั้งหมดของไมโครโปรเซสเซอร์ Z-80 สามารถเขียนแผนผังได้ดังรูปที่ 2.2




**รูปที่ 2.2** โครงสร้างของรีจิสเตอร์ภายในไมโครโปรเซสเซอร์ Z-80

รีจิสเตอร์ในตัวซีพียูจะมีหน้าที่และการใช้งานที่แตกต่างกัน รีจิสเตอร์บางตัวใช้งานเฉพาะอย่าง และบางตัวเป็นรีจิสเตอร์ที่สามารถใช้งานได้ทั่วไป มีรายละเอียดหน้าที่ และการทำงานดังนี้

**1) รีจิสเตอร์หลัก** ที่ใช้งานทั่วไปรีจิสเตอร์ในกลุ่มแรกคือ A, F, B, C, D, E, H และ L ซึ่งจะเป็นรีจิสเตอร์ขนาด 8 บิต ที่ใช้งานทั่วไป รีจิสเตอร์เหล่านี้สามารถประกอบรวมกันเป็นคู่รีจิสเตอร์ได้คือ AF, BC, DE และ HL โดยคู่รีจิสเตอร์นี้ใช้งานในลักษณะของรีจิสเตอร์ขนาด 16 บิต ซึ่งการ

กระทำภายในซีพียูอาจจะอาศัยเพียงรีจิสเตอร์เดียวหรือกระทำเป็นคู่รีจิสเตอร์ได้ ซึ่งโครงสร้างของรีจิสเตอร์หลักแสดงได้ดังรูปที่ 2.3

| | |
|---|---|
| A | F |
| B | C |
| D | E |
| H | L |

**รูปที่ 2.3** รีจิสเตอร์หลัก

รีจิสเตอร์ A คือ แอคคิวมูเลเตอร์ (Accumulator) ซีพียูจะเป็นรีจิสเตอร์ที่ใช้เป็นหลักในการเป็นตัวกระทำสำหรับกระทำทางคณิตศาสตร์และทางลอจิก โดยรีจิสเตอร์หลักนี้จะมีเพียง 8 บิต การกระทำในส่วนของหน่วยคณิตศาสตร์และทางลอจิกย่อมเกิดเงื่อนไขได้หลายอย่าง ซึ่งจะต้องทำการแสดงสถานะภาพของเงื่อนไขเหล่านั้น เช่น เงื่อนไขผลลัพธ์เป็นศูนย์ ผลลัพธ์เป็นบวก หรือลบ มีตัวทด หรือตัวขอยืมในการกระทำทางคณิตศาสตร์ แสดงเงื่อนไขพาริตี้คู่หรือคี่สิ่งเหล่านี้จะให้ผลลัพธ์แสดงสถานะได้ด้วยแฟลก (Flag) เป็นรีจิสเตอร์ขนาด 8 บิต สามารถจะรวมกับแอคคิวมูเลเตอร์เป็นรีจิสเตอร์ขนาด 16 บิต และยังสามารถใช้คำสั่งในการเคลื่อนย้ายข้อมูลจากแอคคิวมูเลเตอร์ A และแฟลก F ไปเก็บใน A' และ F' ได้ เพื่อทำให้การใช้งานของ A และF มีประสิทธิภาพดียิ่งขึ้น

รีจิสเตอร์ F คือ แฟลก (Flag) แฟลกของไมโครโปรเซสเซอร์ Z-80 มีทั้งหมด 6 ตัว ซึ่งจะใช้เพียง 6 บิต แต่ไมโครโปรเซสเซอร์ Z-80 อาศัยการเพิ่มบิตขึ้นอีก 2 บิต และจะกลายเป็นรีจิสเตอร์ F รีจิสเตอร์ F นี้ จะได้รับการเซต (Set) และรีเซต (Reset) กระทำตามคำสั่งในทางคณิตศาสตร์หรือทางลอจิกได้ ลักษณะของรีจิสเตอร์ F ประกอบด้วยแฟลกแต่ละบิตดังรูปที่ 2.4

| S | Z | X | H | X | P/V | N | C |
|---|---|---|---|---|---|---|---|
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |

**รูปที่ 2.4** รีจิสเตอร์แฟลก

จากรูปที่ 2.4 ในบิตที่ 3 ($b_3$) และบิตที่ 5 ($b_5$) เป็นบิตที่ประกอบขึ้นโดยไม่มีความหมายในทางแฟลกแต่จะใช้ประกอบขึ้นเพื่อให้ครบจำนวน 8 บิต หน้าที่ของแฟลกแต่ละตัวมีดังนี้

**1.1) แฟลกตัวทด** (Carry Flag :C) เป็นแฟลกที่ใช้สำหรับทดข้อมูลในรีจิสเตอร์ A เช่น เมื่อมีการบวกข้อมูลขนาด 8 บิต ผลบวกเลยเป็น 9 บิต บิตที่เกินจะทดไปเก็บไว้ในที่แฟลกนี้ ในทำนองเดียวกันถ้าซีพียูมีการลบและมีการยืมค่าของแฟลกตัวนี้จะได้รับการเซตให้มีค่าเป็น 1 เช่นกัน

**1.2) แฟลกศูนย์** (Zero Flag :Z) แฟลกนี้จะได้รับการเซตให้มีค่าเป็น 1 ถ้าผลของการกระทำทำให้รีจิสเตอร์ A มีค่าเป็น 1 นอกจากนั้นมันจะทำการรีเซต

**1.3) แฟลกเครื่องหมาย** (Sign Flag :S) ลักษณะของแฟลกนี้จะบอกถึงการกระทำของซีพียูว่าผลลัพธ์ที่เกิดขึ้นมีค่าเป็นบวกหรือลบ ถ้าเครื่องหมายของตัวเลขเป็นลบจะปรากฏค่าเป็น 1

**1.4) แฟลกพาริตี้หรือโอเวอร์โฟลว์** (Parity Over Flow Flag :P/V) เป็นแฟลกที่ใช้สำหรับเป็นตัวบอกพาริตี้ของผลลัพธ์ในแอกคิวมูเลเตอร์ เมื่อให้มีการกระทำทางลอจิกและยังใช้แสดงสภาวะของค่าที่เกินกำหนดใน 8 บิต (1 บิตที่เป็นเครื่องหมายรวมกับ 7 บิตที่เป็นขนาด) นั่นคือ ค่าสูงสุดที่เป็นไปได้หรือเท่ากับ + 127 และ – 128 ถ้าค่ามากกว่า + 127 หรือน้อยกว่า – 128 ก็จะเซตค่าเป็น 1 ของบิตนี้ขึ้น ในกรณีที่แฟลกไม่ได้รับการเซตคือจะมีค่าเป็น 0 เพื่อบอกว่าผลลัพธ์ไม่เกิดการโอเวอร์โฟลว์ นอกจากนี้ในการกระทำทางลอจิก เช่น แอนด์ (AND), ออร์ (OR), เอ็กซ์ออร์ (XOR) เป็นต้น ผลลัพธ์ที่ได้จะได้รับการตรวจสอบว่ามีพาริตี้คู่หรือพาริตี้คี่ ถ้าเป็นพาริตี้คู่เซตมีค่าเป็น 1

**1.5) แฟลกตัวช่วยทด** (Haft Carry Flag :H) แฟลกนี้จะทำหน้าที่เป็นตัวทดหรือตัวยืมของตัวเลขฐานสอง (BCD)

**1.6) แฟลกการลบ** (Subtract Flag :N) ในการกระทำทางคณิตศาสตร์ของตัวเลขฐานสองเพื่อจะได้มีการรับรู้ในการปรับค่า เมื่อกระทำคำสั่ง DAA ได้ถูกต้อง แฟลกนี้จะเป็นตัวบอกว่าคำสั่งที่ถูกกระทำเป็นการบวกหรือการลบ โดยถ้ากระทำคำสั่งลบแฟลกนี้ได้รับการเซตให้มีค่าเป็น 1

**2) รีจิสเตอร์สำรอง** เป็นกลุ่มรีจิสเตอร์ที่สามารถเก็บข้อมูลได้ โดยจะเป็นตัวเก็บข้อมูลที่ได้มาจากรีจิสเตอร์หลัก รีจิสเตอร์ชุดนี้มี 8 ตัวคือ A', F', B', C', D', E', H' ,L' ซึ่งเป็นรีจิสเตอร์ที่ใช้ในการเก็บข้อมูลชั่วคราวในขณะที่ต้องการใช้รีจิสเตอร์หลักทำงานอย่างอื่นก่อน รีจิสเตอร์กลุ่มนี้ไม่สามารถกระทำทางคณิตศาสตร์และทางลอจิกได้ โครงสร้างรีจิสเตอร์สำรองแสดงได้ดังในรูปที่ 2.5

| | |
|---|---|
| A' | F' |
| B' | C' |
| D' | E' |
| H' | L' |

**รูปที่ 2.5** รีจิสเตอร์สำรอง

**3) กลุ่มรีจิสเตอร์ที่ใช้งานเฉพาะอย่าง** สำหรับรีจิสเตอร์ใช้งานเฉพาะอย่าง จะมีโครงสร้างของรีจิสเตอร์ใช้งานเฉพาะอย่างดังแสดงในรูปที่ 2.6 มีรายละเอียดหน้าที่และการทำงานดังนี้


**รูปที่ 2.6** รีจิสเตอร์ที่ใช้งานเฉพาะอย่าง

**3.1) โปรแกรมเคาน์เตอร์** (Program Counter :PC) เป็นรีจิสเตอร์ขนาด 16 บิต ที่เป็นตัวกำหนดตำแหน่งของโปรแกรมในขณะสภาวะของการกระทำการเฟตช์ (Fetch) โดยขณะทำการเฟตช์ค่าที่อยู่ในโปรแกรมเคาน์เตอร์จะไปปรากฏอยู่ที่แอดเดรสบัส เพื่อชี้ไปยังตำแหน่งในหน่วยความจำให้ซีพียูอ่านคำสั่งมาตีความหมาย ค่าที่อยู่ในโปรแกรมเคาน์เตอร์จะเพิ่มค่าขึ้นได้อย่างอัตโนมัติหลังการกระทำการเฟตช์ แต่ถ้าหากซีพียูกระทำคำสั่งข้ามไปยังตำแหน่งอื่น (Jump) ค่าแอดเดรสที่จะกระโดดข้ามนั้นจะโหลดเข้ามายังโปรแกรมเคาน์เตอร์ได้อย่างอัตโนมัติ

**3.2) สแตคพอยท์เตอร์** (Stack Pointer :SP) เป็นรีจิสเตอร์ที่มีขนาด 16 บิต ที่ใช้สำหรับชี้ไปยังแอดเดรสชั้นบนสุดของสแตค (Stack) ที่อยู่ในแรม (Read Access Memory :RAM) โดยส่วนของสแตคมีลักษณะโครงสร้างเป็นหน่วยความจำเป็นแบบเก็บทีหลังเรียกออกก่อน (Last In First Out) ข้อมูลในสแตคอาจได้รับการพุช (Push) หรือ ป็อบ (Pop) มาจากรีจิสเตอร์ภายในซีพียู ลักษณะของสแตคในที่นี้ยังเป็นส่วนช่วยในการกระทำขัดจังหวะ (Interrupt) และการเรียกโปรแกรมย่อย กล่าวคือ ในการขัดจังหวะค่าของโปรแกรมเคาน์เตอร์จะได้รับการรักษาไว้ในชั้นสแตค เมื่อโปรแกรมกลับจากขัดจังหวะไปกระทำยังโปรแกรมหลัก จะนำค่าจากสแตคกลับเข้ามายังโปรแกรมเคาน์เตอร์ใหม่ ในทำนองเดียวกันการกระโดดไปกระทำยังโปรแกรมย่อยก็เช่นเดียวกัน ดังนั้นการกระทำในรูปของการขัดจังหวะของโปรแกรมย่อยสามารถซ้อนกันได้ไม่มีสิ้นสุด

**3.3) อินเด็กซ์รีจิสเตอร์** (Index Register) ไมโครโปรเซสเซอร์ Z-80 มีอินเด็กซ์รีจิสเตอร์ขนาด 16 บิต 2 ตัว แต่ละตัวใช้ประโยชน์หลักในการทำหน้าที่เป็นตัวเก็บแอดเดรสฐาน (Base Address) เพื่อทำหน้าที่อ้างแอดเดรสแบบอินเด็กซ์แอดเดรสซิ่ง (Index Addressing) ซึ่งในโหมดของอินเด็กซ์แอดเดรสซิ่งมีข้อมูลที่อยู่ในอินเด็กซ์รีจิสเตอร์นี้จะรวมกับข้อมูลที่ติดมากับคำสั่งอีก 8

บิต เพื่อเป็นตัวกำหนดแอดเดรสให้กับคำสั่งข้อมูลที่ติดมากับคำสั่งนี้เรียกว่า ดีสเพลสเมนต์ (Displacement)

**3.4) อินเตอร์รัพท์เพจแอดเดรสรีจิสเตอร์** (I-Interrupt Page Address Register) การขัดจังหวะของไมโครโปรเซสเซอร์ Z-80 มีหลายโหมด และโหมดหนึ่งจะทำให้การขัดจังหวะของไมโครโปรเซสเซอร์ Z-80 มีประสิทธิภาพสูง กล่าวคือ เมื่อเกิดการขัดจังหวะในโหมดนี้ขึ้น สามารถอ้างแอดเดรสโดยทางอ้อม ไปกระทำโปรแกรมในที่ใดก็ได้ในหน่วยความจำ โดยจะอาศัยนำค่าของรีจิสเตอร์ I รวมกับค่าที่ส่งมาจากอุปกรณ์ต่อร่วมภายนอกอีก 8 บิต ชี้ไปยังค่าในหน่วยความจำเพื่อนำค่านั้นมาโหลดเข้าในโปรแกรมเคาน์เตอร์ เพื่อกระทำต่อไป

**3.5) รีจิสเตอร์รีเฟรชหน่วยความจำ** (R-Memory Refresh Register) การต่อซีพียูกับหน่วยความจำนั้นโดยปกติจะต่อกับหน่วยความจำชนิดสแตติค (Static) ได้ง่าย แต่หน่วยความจำชนิดไดนามิค (Dynamic) ที่ต้องการรีเฟรช (Refresh) มีราคาถูกกว่า และมีความหนาแน่นสูงกว่า ซึ่งในไมโครโปรเซสเซอร์ Z-80 ให้ข้อดีกว่าประการหนึ่ง คือ สามารถให้การรีเฟรชหน่วยความจำได้อย่างอัตโนมัติ โดยที่ค่าในรีจิสเตอร์ R จะเพิ่มค่าขึ้นอีก 1 ทุกครั้งที่กระทำการเฟตช์คำสั่ง และข้อมูลในรีจิสเตอร์ R ก็จะถูกส่งออกไปยังแอดเดรสบัสในส่วนบิตที่มีนัยสำคัญต่ำกว่า จังหวะของการส่งนี้เป็นจังหวะเดียวกันกับที่ซีพียูส่งสัญญาณรีเฟรชออกมา สามารถจะกำหนดค่าให้กับรีจิสเตอร์ R ได้ แต่ค่าในรีจิสเตอร์นี้จะเรียกใช้โดยผู้โปรแกรมทางคำสั่งโดยตรงไม่ได้

## 2.3 ชุดคำสั่งของไมโครโปรเซสเซอร์ Z-80

การพัฒนาไมโครโปรเซสเซอร์ Z-80 มีจุดมุ่งหมายหลักเพื่อให้สามารถใช้ชุดคำสั่งเดิมของ 8030 ได้ ดังนั้นทุกชุดคำสั่งที่เคยใช้ได้ในไมโครโปรเซสเซอร์เบอร์ 8080 จะยังคงใช้ได้ กรณีที่ใช้งานกับซีพียูไมโครโปรเซสเซอร์ Z-80 มีคำสั่งได้มากถึง 158 คำสั่ง ซึ่งรวมเอาทั้งหมด 78 คำสั่งที่มีอยู่ใน 8080 ไว้ด้วย กลุ่มของคำสั่งของไมโครโปรเซสเซอร์ Z-80 พอแบ่งแยกออกเป็นกลุ่มได้ดังนี้

1) กลุ่มคำสั่งการโหลด และเคลื่อนย้ายข้อมูล
2) กลุ่มคำสั่งการเคลื่อนย้าย และค้นหาข้อมูลเป็นกลุ่ม
3) กลุ่มคำสั่งทางคณิตศาสตร์ และลอจิก
4) กลุ่มคำสั่งการหมุนข้อมูลเป็นวงรอบ และการเลื่อนข้อมูล
5) กลุ่มคำสั่งการกระทำในส่วนบิต
6) กลุ่มคำสั่งการเรียกโปรแกรมย่อย
7) กลุ่มคำสั่งเกี่ยวกับอินพุต-เอาต์พุต
8) กลุ่มคำสั่งควบคุมการทำงานของซีพียู

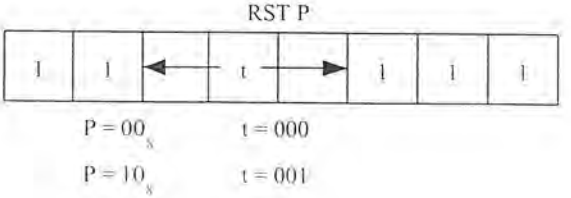
### 2.3.1 การอ้างตำแหน่งแอดเดรส

ภายในซีพียูของไมโครโปรเซสเซอร์ Z-80 มีรีจิสเตอร์หลายตัว การทำงานของซีพียูจึงคล่องตัวได้มาก การที่จะให้ไมโครโปรเซสเซอร์ Z-80 เป็นระบบที่สมบูรณ์จะต้องต่อกับหน่วยความจำ อุปกรณ์อินพุต และเอาต์พุต ในการต่อร่วมเพื่อให้ซีพียูทำงานได้สมบูรณ์ ในกรณีนี้ซีพียูจึงจำเป็นต้องมีการเรียกแอดเดรสที่ต้องการ การอ้างแอดเดรส (Addressing) ของซีพียูมีโหมด (Mode) การอ้างได้หลายวิธี ในแต่ละวิธีจะมีวิธีการอ้างแอดเดรสได้เป็นกลุ่ม ๆ คือ

**1) การอ้างแอดเดรสแบบอิมมีเดียต** (Immediate Addressing) ในโหมดนี้คำสั่งจะประกอบไปด้วยออปโค้ด (Opcode) 1-2 ไบต์ และตามด้วยโอเปอร์แรนด์ (Operand) อีก 1 ไบต์

**2) การขยายตัวโอเปอร์แรนด์ในกลุ่มคำสั่งการอ้างแอดเดรสแบบอิมมีเดียต** (Immediate Addressing Extended) การอ้างแอเดรสแบบนี้ก็จะเหมือนแบบแรก แต่จะแตกต่างกันเพียงส่วนของโอเปอร์แรนด์เป็นข้อมูลขนาด 16 บิต หรือ 2 ไบต์

**3) การอ้างแอดเดรสแบบโมดิไฟด์หน้าศูนย์** (Modified Page Zero) ในคำสั่งพิเศษอยู่คำสั่งหนึ่งที่ทำหน้าที่เหมือนกับคำสั่ง Call คือ คำสั่ง RST (Restart) ซึ่งมีลักษณะการทำงานในการอ้างแอดเดรส คือ สามารถกำหนดค่าให้กับโปรแกรมเคาน์เตอร์ได้โดยตรง ซึ่งค่าที่กำหนดได้จะเป็นแอดเดรสที่อยู่ในหน้าศูนย์ ลักษณะของคำสั่งนี้ประกอบขึ้นเพียงไบต์เดียวเท่านั้น จึงมีประโยชน์ในหลายอย่าง เช่น ใช้เป็นมาส์ค (Mask) สำหรับคำสั่งให้กระทำในขณะที่มีการขัดจังหวะข้อมูลเดิมในโปรแกรมเคาน์เตอร์ก่อนการเปลี่ยนแปลงจะได้รับการเก็บรักษาไว้ได้อีกด้วย แต่เงื่อนไขในการกระโดดไปยังแอดเดรสในหน้าศูนย์ยังมีขอบเขตจำกัด คือ จะกระโดดไปได้เพียง 8 แอดเดรสเท่านั้น ค่าแอดเดรสที่จะไปได้คือ 008, 108,...708 เท่านั้น ซึ่งลักษณะของคำสั่งจะเป็นดังรูปที่ 2.7 จะเห็นว่าการใช้คำสั่ง RST เพียงไบต์เดียวสามารถกำหนดคำสั่งในการทำให้เกิดการกระโดดไปยังแอดเดรสต่าง ๆ ได้ถึง 8 ค่าตามที่กำหนด ส่วนค่าในโปรแกรมเคาน์เตอร์เดิมจะเก็บรักษาไว้ในชั้นของแสตก




รูปที่ 2.7 การอ้างแอดเดรสแบบโมดิไฟด์หน้าศูนย์

**4) การอ้างแอดเดรสแบบเปรียบเทียบ** (Relative Addressing) วิธีการนี้จะใช้ข้อมูลในไบต์ที่อยู่ตามหลังออปโค้ด เพื่อบอกตำแหน่งว่าแอดเดรสที่อ้างถึงอยู่ห่างจากโปรแกรมเคาน์เตอร์เท่าใด

**5) การอ้างแอดเดรสแบบขยาย** (Extend Addressing) วิธีการนี้จะใช้ข้อมูล 2 ไบต์ที่ตามออปโค้ดเป็นตัวกำหนดค่าแอดเดรส การอ้างแอดเดรสแบบนี้ เป็นการเอาข้อมูลในตัวโอเปอร์แรนด์เป็นแอดเดรส โดยการนำเอาข้อมูลส่วนโอเปอร์แรนด์ไปเก็บไว้ในโปรแกรมเคาน์เตอร์โดยตรง ส่วนของข้อมูลเดิมจะเก็บไว้ในส่วนของแสตก

**6) การอ้างแอดเดรสแบบใช้รีจิสเตอร์** (Register Addressing) การออกแบบกลุ่มคำสั่งในขอบเขตของจำนวนบิตออปโค้ดที่จำกัดต้องหาวิธีที่ให้ได้ประสิทธิภาพดีที่สุด วิธีหนึ่งที่ใช้ คือ กำหนดรหัสของรีจิสเตอร์ เช่น มีรีจิสเตอร์ 8 ตัว ใช้รหัส 3 บิต กลุ่มคำสั่งโหลดสามารถใช้คำสั่งเพียง 1 ไบต์ เพื่อที่จะทำการโหลดข้อมูลระหว่างรีจิสเตอร์แสดงได้ดังรูปที่ 2.8 โดยที่ส่วนสามบิตใน $r_d$ (บิตที่ 3-6) คือ ชื่อรีจิสเตอร์ปลายทาง ส่วนสามบิตใน $r_s$ (บิตที่ 0-2) คือ รีจิสเตอร์ต้นทาง เช่น LD C, B เขียนได้เต็ม 01 001 000 เมื่อแทนรหัสสำหรับรีจิสเตอร์ต่าง ๆ เป็นดังตารางที่ 2.1


**รูปที่ 2.8** การอ้างแอดเดรสแบบใช้รีจิสเตอร์

**ตารางที่ 2.1** รหัสแทนรีจิสเตอร์ในการอ้างแอดเดรสแบบใช้รีจิสเตอร์

| รหัส | ชื่อรีจิสเตอร์ |
|---|---|
| 000 | B |
| 001 | C |
| 010 | D |
| 011 | E |
| 100 | H |
| 101 | L |
| 110 | หน่วยความจำที่แอดเดรสด้วยข้อมูลในรีจิเตอร์ HL |
| 111 | A |

7) **การอ้างแอดเดรสโดยใช้อินเด็กซ์รีจิสเตอร์** (Index Register) โดยที่ไมโครโปรเซสเซอร์มีอินเด็กซ์รีจิสเตอร์ 2 ตัว คือ รีจิสเตอร์ IX และ รีจิสเตอร์ IY วิธีการใช้อินเด็กซ์รีจิสเตอร์ร่วมในการอ้างแอดเดรสนั้นใช้ค่า รีจิสเตอร์ IX หรือ รีจิสเตอร์ IY เป็นฐานเพื่อรวมกับค่าที่ตามหลังออปโค้ดมารวมกันเป็นแอเดรสที่ต้องการ

8) **การอ้างแอดเดรสแบบอิมไพล์** (Implied Addressing) ในกรณีนี้ซีพียูจะตีความหมายเองโดยตรงว่า รีจิสเตอร์ภายในซีพียูตัวหนึ่งจะเป็นโอเปอร์แรนด์ กล่าวได้ว่า การทำงานของซีพียูจะนำเอารีจิสเตอร์ที่กำหนดในออปโค้ดมากระทำร่วมกับรีจิสเตอร์ A นั่นเอง

9) **การอ้างแอดเดรสเข้าสู่บิต** (Bit Addressing) ภายในตัวไมโครโปรเซสเซอร์จะมีคำสั่งพิเศษที่สามารถกระทำการเซต รีเซต หรือ การทดสอบบิตใดบิตหนึ่งใน 8 บิตได้

### 2.3.2 กลุ่มคำสั่งโหลด และเคลื่อนย้ายข้อมูล

กลุ่มของคำสั่งโหลด และเคลื่อนย้ายข้อมูลจะมีการเคลื่อนย้ายข้อมูลได้ 2 แบบ คือ การโหลด และเคลื่อนย้ายข้อมูลขนาด 8 บิต และขนาด 16 บิต เพื่อให้ทำความเข้าใจกลุ่มคำสั่งโหลด เช่น LD E, (IX+08) ชุดคำสั่งนี้ในหน่วยความจำจะเป็นคำสั่งการเคลื่อนย้ายข้อมูลจากหน่วยความจำที่หาค่ามาจากค่าของรีจิสเตอร์ IX รวมกับ ตัวเลข 08 เก็บไว้ในรีจิสเตอร์ E เป็นคำสั่งขนาด 3 ไบต์ และจากคำสั่ง 3 ไบต์ สามารถใช้วิธีการเพิ่มขนาดแอดเดรสได้

การโหลดข้อมูลลงในหน่วยความจำที่มีความสามารถสูงวิธีหนึ่ง คือ การโหลดใช้อินเด็กซ์รีจิสเตอร์เป็นตัวรับข้อมูล และใช้วิธีการอิมมิเดียทเป็นการกำหนดแหล่งเริ่มต้นของข้อมูล นอกจากนี้ยังสามารถใช้คำสั่งโหลดทำการโหลดข้อมูลในรูปแบบของการโหลดข้อมูลในรูปแบบของการโหลด 16 บิตได้อีกด้วย และเพื่อลดขนาดของคำสั่งให้สั้นขึ้น ตัวคำสั่งจึงเป็นลักษณะของการแลกเปลี่ยนข้อมูลของคู่รีจิสเตอร์ต่าง ๆ

### 2.3.3 กลุ่มคำสั่งเคลื่อนย้าย และค้นหาข้อมูลเป็นกลุ่ม

ไมโครโปรเซสเซอร์ Z-80 มีคำสั่งที่ทำให้การทำงานเป็นไปอย่างมีประสิทธิภาพโดยทำให้ลดขนาดของตัวโปรแกรมลงได้มาก มีลักษณะของกลุ่มคำสั่งในกลุ่มนี้จะอาศัยการทำงานร่วมกันของคู่รีจิสเตอร์ภายในซีพียู 3 คู่ คือ รีจิสเตอร์ HL ซึ่งเป็นรีจิสเตอร์ที่จะอ้างถึงแอดเดรสตำแหน่งจุดต้นทาง, รีจิสเตอร์ DE เป็นรีจิสเตอร์ที่อ้างถึงตำแหน่งแอดเดรสปลายทาง และรีจิสเตอร์ BC เป็นตัวนับจำนวนไบต์

ในการค้นหาข้อมูลใช้วิธีการ เช่น CPI (Compare With Increment) ซึ่งการทำงานจะเป็นไปดังนี้ ซีพียูจะนำข้อมูลจากหน่วยความจำที่มีแอดเดรสเป็นค่าอยู่ในคู่รีจิสเตอร์ HL มาเปรียบเทียบกับรีจิสเตอร์ A หลังจากนี้ค่าของคู่รีจิสเตอร์ HL จะเพิ่มค่าอีก 1 และค่าของคู่รีจิสเตอร์ BC จะลดค่าลงไปทีละ 1 ลักษณะการเปรียบเทียบจะให้ผลลัพธ์ที่แฟลก จากคำสั่งนี้จะให้การเซตค่ารีจิสเตอร์ BC

เป็นจำนวนไบต์ที่ต้องการเปรียบเทียบ ค่าของรีจิสเตอร์ BC จะลดลงจนเป็น 0 แล้วทดสอบค่าของ BC เท่ากับ 0 จะเห็นว่าทุกครั้งที่ทำคำสั่งนี้ค่าคู่รีจิสเตอร์ HL จะเพิ่มค่า ดังนั้นถ้าให้การกระทำคำสั่งนี้ใหม่แอดเดรสในหน่วยความจะเพิ่มขึ้นทีละหนึ่งเรื่อยไป

เพื่อเพิ่มประสิทธิภาพของไมโครโปรเซสเซอร์ Z-80 ยังมีคำสั่งที่สามารถให้การกระทำคำสั่งจนครบจำนวนไบต์ที่วางไว้ เช่น CPIR (Compare With Increment and Repeat) การกระทำคำสั่งนี้จะเหมือนกับคำสั่ง CPI แต่ซีพียูจะเพิ่มการทดสอบค่าในคู่รีจิสเตอร์ BC ว่าเป็น 0 แล้วหรือยัง ถ้ายังก็จะกระทำคำสั่งเดิมนี้ซ้ำอีก ซึ่งค่าในรีจิสเตอร์ BC จะลดลงมาทีละ 1 จนเป็น 0 จึงหยุดกระทำ

นอกจากนี้ยังสามารถให้ค่าแอดเดรสในคู่รีจิสเตอร์ HL ลดค่าทีละ 1 ได้เช่นกัน โดยใช้คำสั่ง CPD หรือ CPDR

สำหรับการเคลื่อนย้ายข้อมูลเป็นบล็อก ก็กระทำเช่นเดียวกับการเปรียบเทียบโดยการใช้คู่รีจิสเตอร์ HL เป็นตัวบอกแอดเดรสของหน่วยความจำที่ต้องการย้าย และคู่รีจิสเตอร์ DE เป็นตัวบอกค่าแอดเดรสที่จะนำไปเก็บอยู่ที่ใด การทำงานในลักษณะนี้ซีพียูนำข้อมูลในหน่วยความจำที่แอดเดรสไปเก็บไว้ในหน่วยความจำแอดเดรสที่รีจิสเตอร์ DE ชี้อยู่ และหลังจากนั้นค่าในรีจิสเตอร์ BC ลดลงไป 1 และค่าในรีจิสเตอร์ HL และ DE จะเพิ่มขึ้น 1 ในทำนองเดียวกันสามารถใช้ลำพังคำสั่งเดียวในการเคลื่อนย้ายข้อมูลทั้งบล็อกได้ด้วยการเอาจำนวนข้อมูลไว้ในรีจิสเตอร์ BC แล้วใช้คำสั่ง LDIR การทำงานของคำสั่งนี้จะคล้ายกับ LDI แต่จะทำซ้ำ ๆ ไปเรื่อย ๆ จนกระทั่งค่าในรีจิสเตอร์ BC เป็น 0 จึงหยุดกระทำคำสั่ง

### 2.3.4 กลุ่มคำสั่งทางคณิตศาสตร์และลอจิก

คำสั่งนี้จะกระทำกับรีจิสเตอร์ A ส่วนใหญ่ ลักษณะของกลุ่มคำสั่งประกอบด้วยการ ADD, SUB, ADC, SBC, INC, DEC, AND, OR และ XOR ในการอ้างแอดเดรสของตัวโอเปอร์แรนด์ทำได้หลายแบบขึ้นอยู่กับการเขียนและใช้งาน การกระทำในกลุ่มคำสั่งนี้จะให้ผลลัพธ์เก็บซ้ำที่รีจิสเตอร์ A และนอกจากนี้แล้วค่าในแฟลกจะมีผลต่อการกระทำคำสั่งด้วย เช่น ถ้าค่าในรีจิสเตอร์ A เป็นศูนย์ แฟลกตัวศูนย์ก็ได้รับการเซตค่าไว้ ยังมีคำสั่งที่กระทำทางคณิตศาสตร์ และลอจิกโดยเฉพาะบางอย่าง เช่น การกระทำการเปลี่ยนรหัสตัวเลขไบนารีให้เป็นตัวเลข BCD ได้ด้วยคำสั่ง DAA คำสั่งเกี่ยวกับการการทำคอมพลีเมนต์ การทำให้เป็นจำนวนลบ การเซต และการรีเซตแฟลกบางตัว และยังมีคำสั่งในกาควบคุมการทำงานของซีพียู เช่น NOP คือ ไม่ต้องทำอะไร HALT ให้หยุดโปแกรม และนอกจากนี้ก็มีคำสั่งเกี่ยวกับการขัดจังหวะของซีพียู

การกระทำทางคณิตศาสตร์ของไมโครโปรเซสเซอร์ Z-80 ยังสามารถการะทำในรูปของการบวกเลข 16 บิตได้อีกด้วย การบวกในกรณีนี้จะใช้การบวกระหว่างคู่รีจิสเตอร์ เช่น รีจิสเตอร์ HL กับ BC เป็นต้น ในการบวกเช่นนี้จะทำให้การกระทำทางคณิตศาสตร์มีประสิทธิภาพมากขึ้น

### 2.3.5 กลุ่มคำสั่งในการหมุนและการเลื่อนข้อมูล

ความสามารถพื้นฐานของไมโครโปรเซสเซอร์ Z-80 ในการทำข้อมูลและเลื่อนข้อมูลเหมือนกันในซีพียูทั่วไป เช่น มักเลื่อนบิตไปทางซ้ายและทางขวาเลื่อนเป็นวงรอบ โดยผ่านบิตทดหรือไม่ผ่านบิตทด แต่ไมโครโปรเซสเซอร์ Z-80 มีคำสั่งเกี่ยวกับการเลื่อนข้อมูลด้วยตัวเลข BCD อยู่ 2 คำสั่ง คือ RRD และ RLD ทั้งสองคำสั่งนี้จะให้ข้อมูล 1 ตัวเลข (BCD) ในแอคคิวมูเลเตอร์เลื่อนเป็นวงรวมกับข้อมูล 2 ตัวเลข ในหน่วยความจำที่อ้างแอดเดรสโดยรีจิสเตอร์ HL

การกระทำในกลุ่มคำสั่งนี้เกิดขึ้นภายใน ALU โดยมีการกระทำร่วมกับแฟลก ดังนั้นจังมีผลเกี่ยวข้องโดยตรงกับแฟลกบางตัว เช่น แฟลกการทด แฟลกเครื่องหมาย และแฟลกศูนย์เป็นต้น

### 2.3.6 กลุ่มคำสั่งเกี่ยวกับการกระทำภายในข้อมูลบิต

ในไมโครโปรเซสเซอร์ Z-80 มีความสามารถพิเศษในการเซต รีเซต หรือทดสอบบิตหนึ่งในรีจิสเตอร์ต่าง ๆ และในหน่วยความจำได้รายละเอียดของการเซต รีเซต และการทดสอบสามารถแยกคำสั่งย่อย ๆ ในกลุ่มนี้ได้มาก เพราะแต่ละคำสั่งจะเท่ากับมีคำสั่งย่อย ๆ ได้ถึง 8 คำสั่ง คือ การกระทำในแต่ละบิตต่าง ๆ ของรีจิสเตอร์ต่าง ๆ นั่นเอง

ในการทดสอบบิตต่าง ๆ ว่าเป็น 0 หรือ 1 นั้นใช้แฟลกศูนย์เป็นตัวเก็บข้อมูล เช่น ถ้าบิตที่ทดสอบเป็น 0 ก็จะเซตแฟลกศูนย์ให้เป็น 1

การกระทำในกรณีนี้กระทำได้ในทุกรีจิสเตอร์ ในหน่วยความจำ การอ้างแอดเดรสไปยังหน่วยความจำก็ทำได้ ทั้งใช้แบบทางอ้อมผ่านรีจิสเตอร์ HL หรือใช้อินเด็กซ์รีจิสเตอร์ IX, IY ร่วมด้วยก็ได้

### 2.3.7 กลุ่มคำสั่งการเรียกโปรแกรมย่อย

ไมโครโปรเซสเซอร์ Z-80 มีกลุ่มคำสั่งเกี่ยวกับการกระโดดที่มีประสิทธิภาพ คือ คำสั่ง JP และ JR การกระโดดไปยังที่ใดก็ได้ในที่นี้ คือ การโหลดเปลี่ยนค่าโปรแกรมเคาน์เตอร์นั่นเอง เช่น JP nn คือ การโหลดข้อมูล nn เข้าไปยังโปรแกรมเคาน์เตอร์

คำสั่ง JP ของกลุ่มคำสั่งที่มีพิเศษนอกเหนือจากไมโครโปรเซสเซอร์ 8080 มีคือ JR หรือการกระโดดไปยังแอดเดรสที่อ้างเปรียบเทียบกับค่าโปรแกรมเคาน์เตอร์เดิมของมัน ลักษณะคำสั่งนี้มีข้อดีคือ ใช้จำนวนไบต์น้อยกว่า คือ ใช้เพียง 2 ไบต์เท่านั้น

กลุ่มคำสั่งที่อ้างกับโปรแกรมย่อย เพื่อให้การเรียกโปรแกรมย่อยจะได้มีประสิทธิภาพสูงขึ้นไมโครโปรเซสเซอร์ Z-80 ใช้วิธีการเรียกโดยคำสั่ง CALL และ RET คำสั่ง CALL จะทำให้โปรแกรมเคาน์เตอร์เดิมที่มีอยู่ไปเก็บไว้ที่แสตกแล้วโหลดค่าแอดดรสของโปรแกรมย่อยกลับคืนมาให้โปรแกรมเคาน์เตอร์ และในการ RET ก็จะกระทำกลับกันนั่นคือ เป็นการป๊อบเอาข้อมูลในแสตกมาให้โปรแกรมเคาน์เตอร์

### 2.3.8 กลุ่มคำสั่งติดต่อกับอินพุต เอาต์พุต และควบคุมการทำงานของซีพียู

ในซีพียู 8080 กลุ่มคำสั่งติดต่อกับอินพุตและเอาต์พุตเพียง 2 คำสั่ง คือ IN และ OUT แต่ในไมโครโปรเซสเซอร์ Z-80 ได้มีการขยายการติดต่อกับอินพุตและเอาต์พุตมากขึ้น ซึ่งจะทำให้มีประสิทธิภาพการติดต่อไปเป็นได้ดียิ่งขึ้น เช่น คำสั่ง IN A, PORT เป็นต้น

กลุ่มคำสั่งการควบคุมการทำงานของซีพียู มีหลายคำสั่ง เช่น NOP ซึ่งเป็นคำสั่งที่ซีพียูไม่ต้องทำอะไรเลย HALT เป็นคำสั่งให้ซีพียูหยุด นอกจากนี้ยังมีคำสั่งเกี่ยวกับการขัดจังหวะ เช่น DI, EI. IM0, IM1 และ IM2 เป็นต้น

## 2.4 การเขียนโปรแกรมด้วยวิชวลเบสิก

การเขียนโปรแกรมคือ การสั่งให้คอมพิวเตอร์ทำงานตามที่ต้องการ เช่น โปรแกรมฝึกพิมพ์ดีด ซึ่งเป็นโปรแกรมที่สั่งให้เครื่องคอมพิวเตอร์โต้ตอบกับการกดแป้นคีย์บอร์ด (Keyboard) เพื่อสอนผู้ใช้พิมพ์ดีด เป็นต้น

สำหรับโปรแกรมวิชวลเบสิกเป็นเครื่องมือในการสร้างโปรแกรมบนระบบปฏิบัติการวินโดวส์ (Windows) ที่ใช้งานง่าย โดยการสร้างโปรแกรมในวิชวลเบสิก 6 นั้น เป็นการเลือกเครื่องมือต่าง ๆ มาออกแบบหน้าจอหน้าโปรแกรมที่จะสร้าง ซึ่งเรียกการเขียนโปรแกรมลักษณะนี้ว่า วิชวลโปรแกรมมิ่ง (Visual Programming) การเขียนโปรแกรมแบบนี้จะไม่จำเป็นต้องเขียนคำสั่งต่าง ๆ มากนัก ก็สามารถสร้างโปรแกรมได้อย่างรวดเร็ว

หลังจากที่ได้ออกแบบหน้าจอโปรแกรมตามวิธีที่กล่าวมา แล้วจะต้องเขียนโปรแกรมควบคุมการทำงานด้วย โดยใช้ภาษา BASIC (ย่อมาจาก Beginners All – Purpose Symbolic Instruction Code) ซึ่งเป็นภาษาที่ใช้งานง่าย เหมาะสำหรับผู้เริ่มต้นศึกษาการเขียนโปรแกรมบนวินโดวส์ สำหรับวิชวลเบสิก 6 นั้นเป็นเครื่องมือที่สร้างโปรแกรมต่าง ๆ ได้หลากหลาย ดังต่อไปนี้

1) โปรแกรมทั่วไปที่รันบนระบบปฏิบัติการวินโดวส์ โดยสามรถสร้างโปรแกรมทางด้านกราฟฟิก โปรแกรมจัดการไฟล์ โปรแกรมคำนวณเลขพื้นฐานให้ตรงกับความต้องการของการใช้งานเป็นต้น

2) โปรแกรมฐานข้อมูล วิชวลเบสิก 6 นั้นช่วยให้การสร้างโปรแกรมฐานข้อมูลเป็นเรื่องง่าย เนื่องจากกการมีเครื่องมือต่างๆ เกี่ยวกับฐานข้อมูลอย่างครบถ้วน เช่น เครื่องมือในการติดต่อกับฐานข้อมูลทั้งไมโครซอฟท์แอคเซส (Microsoft Access) หรือฐานข้อมูลบนระบบไคเอนท์เซฟเวอร์ (Client Sever) เช่น ไมโครซอฟท์เอสคิวแอลเซิร์ฟเวอร์ (Microsoft SQL Server) โดยการติดต่อกับฐานข้อมูลนั้น เพียงแต่กำหนดตำแหน่งของฐานข้อมูลนั้น สามารถติดต่อกับฐานข้อมูลนั้นได้ทันที

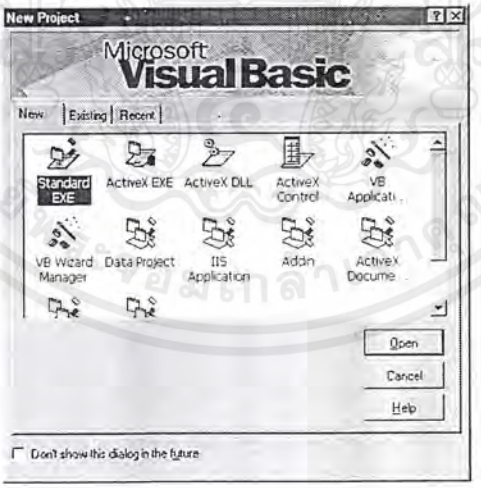
นอกจากนี้ในวิชวลเบสิก 6 ก็มีเครื่องมือในการสร้างรายงานสรุปข้อมูลจากฐานข้อมูลที่ใช้งานง่าย โดยการใช้เมาส์ (Mouse) ลากฟิลด์(Field) ข้อมูลที่ต้องการไปที่ที่ต้องการในรายงานที่ออกแบบได้เลย

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

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

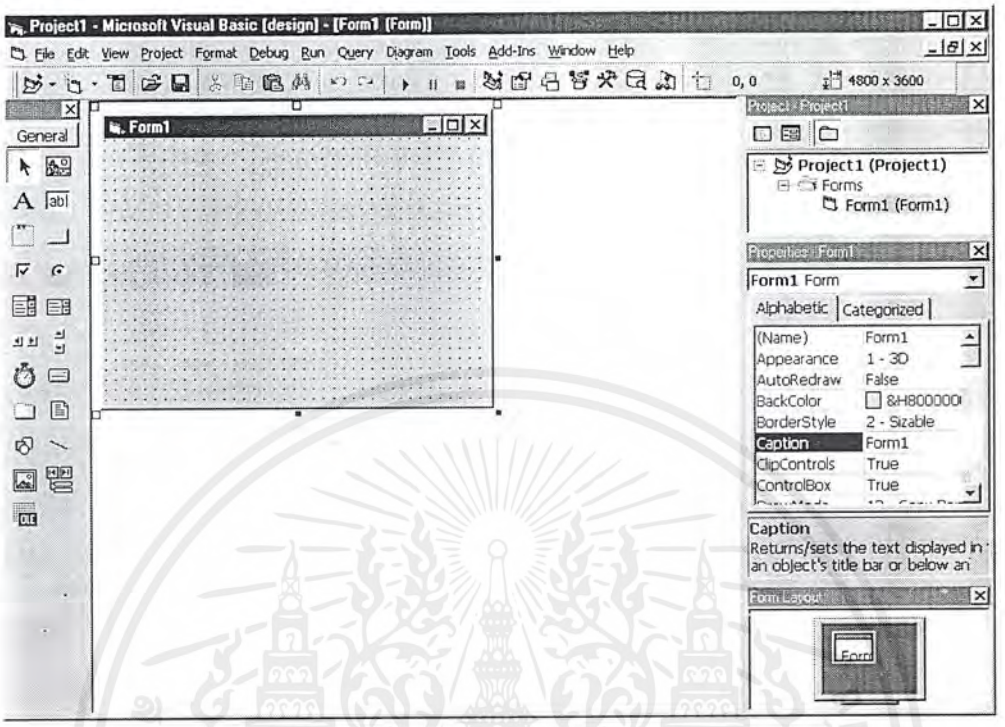
### 2.4.1 ส่วนประกอบของโปรแกรมวิชวลเบสิก

เมื่อทำการเปิดวิชวลเบสิกขึ้นมาจะปรากฏหน้าจอภาพดังรูปที่ 2.9 โดยทั่วไปแล้วจะเลือก Standard.EXE ซึ่งเป็นการใช้วิชวลเบสิกในการเขียนโปรแกรมที่ใช้บนในวินโดวส์ทั่วไป เนื่องจากโปรเจ็กต์ (Project) ชนิด Standard.EXE เป็นการเขียนโปแกรมทั่วไปที่ใช้บนวินโดวส์ โดยทั่วไปโปรเจ็กต์ คือ กลุ่มของไฟล์ที่จะนำมารวมกันเพื่อสร้างโปรแกรมในวิชวลเบสิก




**รูปที่ 2.9** หน้าต่างเริ่มต้นเข้าใช้งานโปแกรมวิชวลเบสิก

เมื่อเลือกเสร็จจะปรากฏหน้าจอของวิชวลเบสิก โดยมีชื่อของส่วนประกอบต่างๆ ที่จำเป็นจะต้องทราบดังรูปที่ 2.10




**รูปที่ 2.10** หน้าต่างของโปรแกรมวิชวลเบสิก

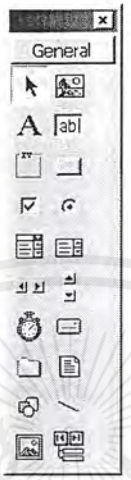
จากรูปที่ 2.10 หน้าต่างของโปรแกรมวิชวลเบสิกได้ประกอบไปด้วยส่วนต่าง ๆ ซึ่งแต่ละส่วนจะมีหน้าที่ และการทำงานที่แตกต่างกัน ซึ่งรายละเอียดมีดังนี้

**1) ทูลบาร์ (Tool bar)** เมื่อพิจารณาหน้าต่างของโปรแกรมวิชวลเบสิกจะมีปุ่มต่าง ๆ ที่วางเป็นแผงควบคุม ช่วยให้สามารถเรียกใช้งานคำสั่งได้อย่างสะดวกรวดเร็ว โดยเพียงแค่คลิกเมาส์ที่ปุ่มเท่านั้น ซึ่งดังแสดงรูปที่ 2.11


**รูปที่ 2.11** ทูลบาร์ของโปรแกรมวิชวลเบสิก

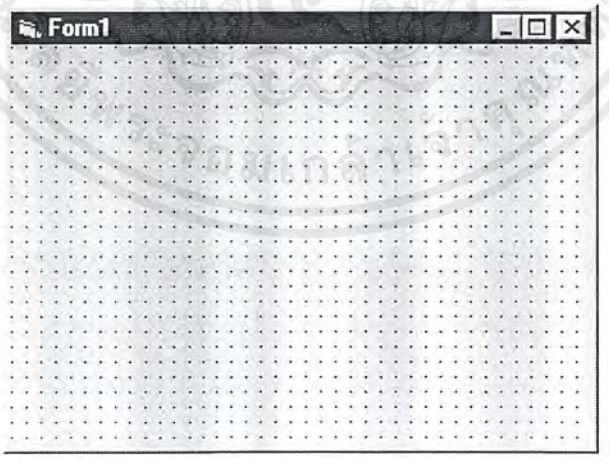
**2) ทูลบ็อกซ์ (Toolbox)** เป็นที่รวมออบเจ็ก (Object) ต่างที่จะนำมาประกอบกันเป็นโปรแกรม เมื่อใช้ออบเจ็กเหล่านี้ประกอบกันจะได้เป็นหน้าตาของโปรแกรมอาจเรียกให้ชัดเจนได้ว่า

คอนโทรลออบเจ็ก (Control Object) ซึ่งมีออบเจ็กหลักดังรูปที่ 2.12 นอกจากนี้สามารถที่จะทำการเพิ่มออบเจ็กต่างๆ เข้าไปในทูลบ็อกซ์ได้อีกมากมาย



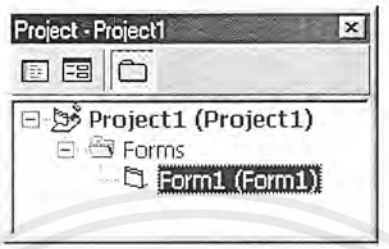

**รูปที่ 2.12** ทูลบ็อกซ์ของโปรแกรมวิชวลเบสิก 6

**3) หน้าต่างฟอร์ม (Form)** เป็นหน้าต่างเปล่าๆ หรือตัวฟอร์ม (Form) สำหรับในโครงสร้างองค์ประกอบของแอพพลิเคชัน (Application) เมื่อเริ่มเข้าสู่โปรแกรมวิชวลเบสิกจะปรากฏฟอร์มเปล่าให้เสมอ ซึ่งหน้าตาของหน้าต่างฟอร์มดังแสดงในรูปที่ 2.13




**รูปที่ 2.13** หน้าต่างฟอร์มของโปรแกรมวิชวลเบสิก

**4) หน้าต่างโปรเจ็กเอ็กซ์โพลเรอร์ (Project Explorer)** โปรแกรมต่างที่สร้างขึ้นมานั้นเรียกว่าโปรแกรมประยุกต์ หรือ แอพพลิเคชัน ซึ่งในวิชวลเบสิกจะเรียกโปรแกรมที่กำลังสร้างอยู่ว่าเป็นโครงการ หรือโปรเจ็ก หน้าต่างของโปรเจ็กเอ็กซ์โพลเรอร์มีรูปร่างหน้าตาดังแสดงในรูปที่ 2.14




**รูปที่ 2.14** หน้าต่างโปรเจ็กเอ็กซ์โพลเรอร์ของโปรแกรมวิชวลเบสิก

หน้าต่างโปรเจ็กเอ็กซ์โพลเรอร์จะใช้ควบคุมส่วนประกอบ และแฟ้มข้อมูลต่างๆ ที่อยู่ในโปรเจ็กต์ โครงสร้างแฟ้มข้อมูลส่วนต่างที่ประกอบขึ้นมาเป็นโปรเจ็กต์ เพื่อความสะดวกในที่จะการควบคุม และเปลี่ยนการทำงานระหว่างส่วนต่างๆ โดยแต่ละโปรเจ็กต์จะประกอบไปด้วยแฟ้มข้อมูลมากมายหลายประเภท ซึ่งจะแสดงรายละเอียดดังในตารางที่ 2.2

**ตารางที่ 2.2** 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<br>Vbx |

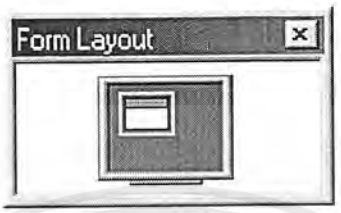
ตารางที่ 2.2 (ต่อ) ประเภทของแฟ้มข้อมูลในโปรเจ็กเอ็กโพลเรอร์

| ประเภทไฟล์ | รายละเอียด | นามสกุลไฟล์ |
|---|---|---|
| ไฟล์เอกสาร (ActiveX) | เหมือนกับฟอร์ม เพียงแต่ต้องเรียกผ่านโปรแกรมเว็บเบาว์เซอร์ | Dob |
| ไฟล์คลาสโมดุล (Class Module) | เก็บออบเจ็กต่างๆ ที่สร้างขึ้น เมื่อมีการเรียกใช้คลาสโมดูล โปรแกรมจะสร้างออบเจ็กนั้นขึ้นมาใหม่แทนที่จะใช้จากโปรแกรมหรือออบเจ็กโดยตรง อาจกล่าวได้ว่าคลาสโมดูลเปรียบเสมือนที่เก็บแผงหนังสือ หรือเทมเพลท (Template) ของออบเจ็กที่จะสร้างขึ้นมาใหม่นั่นเอง | Cls |
| ไฟล์ทรัพยากรอื่นๆ (Resource File) | เก็บภาพนามสกุล bmp ข้อความหรือข้อมูลใดๆ ที่สามารถแก้ไขได้โดยไดมต้องไปยุ่งเกี่ยวกับโปรแกรมในโมดูลหรือฟอร์มต่างๆ ในโปรเจ็กต์ | Res |

**5) หน้าต่างพรอบเพอร์ตี้ (Properties)** หน้าต่างนี้จะแสดงคุณสมบัติทั้งหมดของออบเจ็กที่ถูกเลือกอยู่ การคลิกเลือกที่ออบเจ็กใดในฟอร์มจะทำให้คุณสมบัติที่แสดงในหน้าต่างพรอบเพอร์ตี้เปลี่ยนไปตามออบเจ็กที่เลือก ซึ่งในการแก้ไขหรือตั้งค่าของคุณสมบัติต่างสามารถทำได้โดยตรงที่คุณสมบัติแต่ละค่า ซึ่งหน้าต่างพรอบเพอร์ตี้แสดงได้ดังรูปที่ 2.15


รูปที่ 2.15 หน้าต่างพรอบเพอร์ตี้

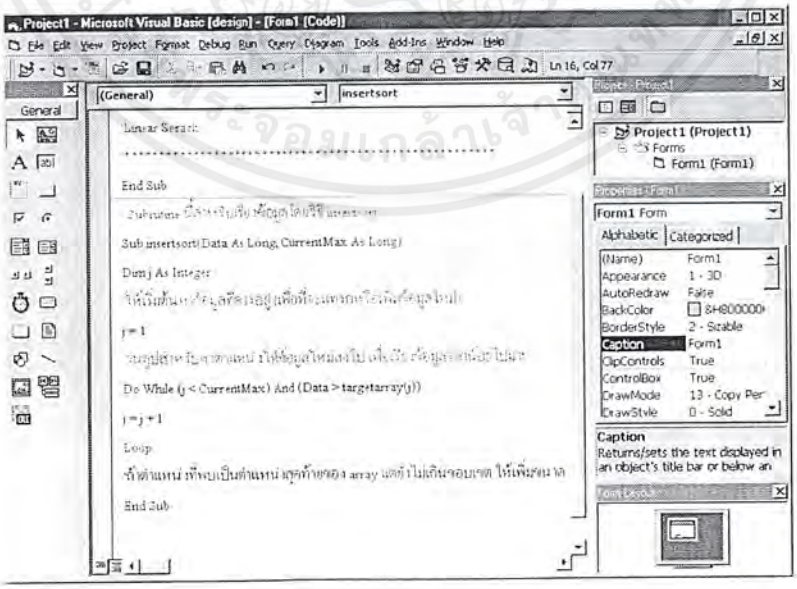
**6) หน้าต่างฟอร์มเลเอาต์ (Form Layout)** จะแสดงตำแหน่งฟอร์มของโปรแกรมที่กำลังสร้างให้ดูบนจอภาพ เพื่อกำหนดตำแหน่ง สำหรับตอนที่โปรแกรมทำงานจริง การย้ายตำแหน่งทำได้โดยการใช้เมาส์ลาก (Drag) รูปฟอร์มในจอภาพไปยังตำแหน่งที่ต้องการ ดังแสดงในรูปที่ 2.16




**รูปที่ 2.16** หน้าต่างฟอร์มเลเอาต์

**7) หน้าต่างโค้ดอิดิเตอร์ (Code Editor)** เป็นเนื้อที่สำหรับในเขียนโปรแกรม ซึ่งหน้าต่างโค้ดอิดิเตอร์จะแสดงขึ้นมาพร้อมสำหรับการเขียนโปรแกรมให้กับเหตุการณ์ (Event) หลักของออบเจ็กนั้น ส่วนที่สำคัญของหน้าต่างนี้คือ คอมโบบ็อกซ์ (Combo Box) ทั้งสองช่องที่อยู่ตรงส่วนบนของหน้าต่าง ซึ่งจะเป็นตัวควบคุมการเลือกออบเจ็ก และเหตุการณ์จะเกิดขึ้นกับออบเจ็กนั้น โดยโค้ดที่ปรากฏจะเป็นโปรแกรม หรือคำสั่งที่จะถูกเรียกใช้งานเมื่อมีเหตุการณ์นั้นเกิดขึ้นกับออบเจ็ก

หน้าต่างของโค้ดอิดิเตอร์จะดังแสดงในรูปที่ 2.17 ซึ่งสามารถเข้าไปเขียนโปรแกรมควบคุมนี้ได้โดยการคลิกคอนโทรลที่ต้องการแล้วจะปรากฏหน้าต่างนี้ให้เห็น




**รูปที่ 2.17** หน้าต่างโค้ดอิดิเตอร์

### 2.4.2 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 ที่ได้ทำการเลือกมาแล้วนำมาวาง คอนโทรลที่มีอยู่ในรูปที่ 2.2 จะเป็นเพียงคอนโทรลพื้นฐานในการเขียนโปรแกรม แต่ยังมีคอนโทรลอื่นๆ ที่น่าสนใจอีกมากมาย

การสร้างโปรแกรมในขั้นตอนแรกนั้นจะต้องเลือกคอนโทรลต่างๆ ให้เหมาะสมกับการทำงานของโปรแกรม เพราะว่าเป็นส่วนที่ผู้ใช้งานเห็น และทำงานด้วย 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 สำหรับกฎการตั้งชื่อของคำสั่งที่ตอบสนองต่อเหตุการณ์ในวิชวลเบสิก จะใช้ชื่อคอนโทรลตามด้วยเครื่องหมายขีดเส้นใต้ (_) และชื่อของเหตุการณ์ เช่น ปุ่มคำสั่งชื่อ Command1 ดังนั้นคำสั่งที่ตอบสนองต่อเหตุการณ์คลิก จะมีชื่อเป็น Command1_Click

หากต้องการที่จะทราบรายละเอียดในการเขียนโปรแกรมขั้นสูงขึ้นสามารถค้นหาข้อมูลได้จากในหนังสืออ้างอิงท้ายเล่มได้ ซึ่งมีรายละเอียดต่างๆ

## 2.5 การประมวลระดับบิตด้วย Bits32.dll

ปัจจุบันได้มีการนำเอาโปรแกรมวิชวลเบสิกมาใช้ในการสร้างโปรแกรมเพื่อควบคุมบอร์ดอิเล็กทรอนิกส์ประเภทซิงเกิ้ลบอร์ด (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) และนอกจากนี้ยังสนับสนุนการแปลงข้อมูลไบนารีให้อยู่ในรูปแบบของสติงข้อมูลฐานสิบหกได้อีกด้วย ซึ่งสามารถใช้ความสามารถในส่วนนี้เพื่อนำมาสร้างโปรแกรมที่ช่วยในการแสดงข้อมูลที่มีข้อมูลชนิดไบนารี

### 2.5.1 พื้นฐานการประมวลผลบิต

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

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

**2) ข้อมูลขนาดเวิร์ด** (Word Data Size) ข้อมูลขนาด 1 เวิร์ด จะมีขนาด 16 บิต ซึ่งเป็นมาตรฐานในการจัดเก็บข้อมูลตามรหัสยูนิโค้ด (Unicode) ในปัจจุบัน สำหรับโปรแกรมวิชวลเบสิกการประกาศข้อมูลชนิดอินทิเจอร์ (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 เป็นขนาดมาตรฐานในการอ้างอิงโค้ด หรือแบ่งเซกเมนต์ของในระบบปฏิบัติการวินโดวส์ 32 บิต เช่น วินโดวส์ 95, วินโดวส์ 98 และวินโดวส์ NT เป็นต้น สำหรับในโปรแกรมวิชวลเบสิก การประกาศข้อมูลชนิดลอง (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

### 2.5.2 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

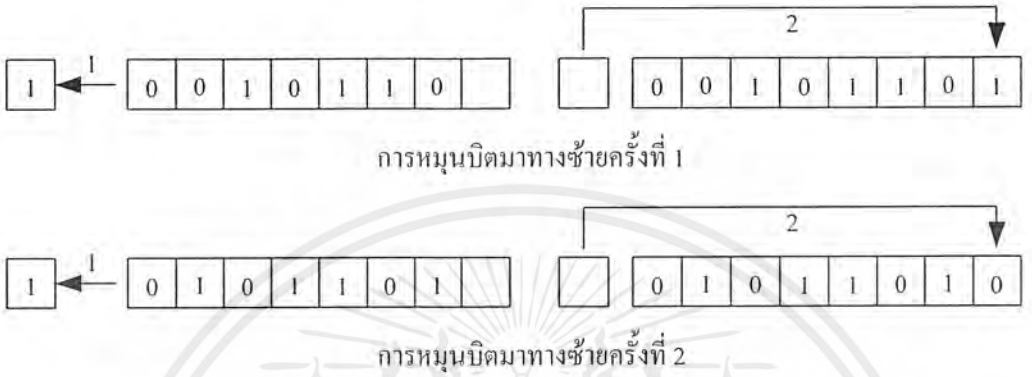
**2) การกำหนดค่าบิต (Set Bit)** หมายถึง การกำหนดให้บิตใดบิตหนึ่ง หรือ หลาย ๆ บิตมีค่าเท่ากับ 1 เช่น การกำหนดค่าลำดับที่ 3 ของค่าตัวเลขฐานสอง 10010110 จะเท่ากับ 10011110

**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)

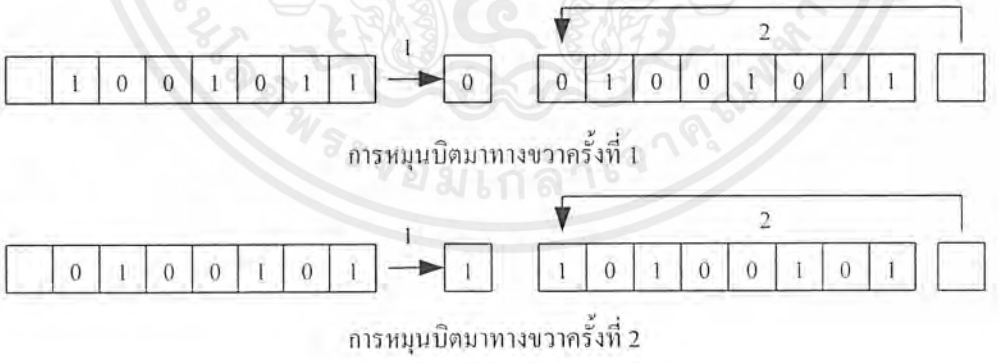
**5) การหมุนบิต (Rotate Bit)** ในการภาษาแอสเซมบลี การหมุนบิตซ้าย และบิตขวาจะเทียบเท่ากับคำสั่ง ROL และ ROR ตามลำดับ การหมุนบิตสามารถแบ่งออกได้ 2 ลักษณะดังนี้ การหมุน

บิตทางซ้าย (Rotate Bit Left) หมายถึง การเลื่อนทุก ๆ บิตของสายข้อมูลบิตมาทางซ้ายเท่ากับจำนวนบิตที่กำหนด และนำบิตที่ถูกเลื่อนมาทางซ้ายสุดดังกล่าวมาแทนที่บิตทางขวามือที่ถูกเลื่อน เช่น ต้องการหมุนบิตของตัวเลขฐานสอง 10010110 มาทางซ้ายจำนวน 2 บิต จะมีเป็นดังรูปที่ 2.18




**รูปที่ 2.18** การหมุนบิตทางซ้าย

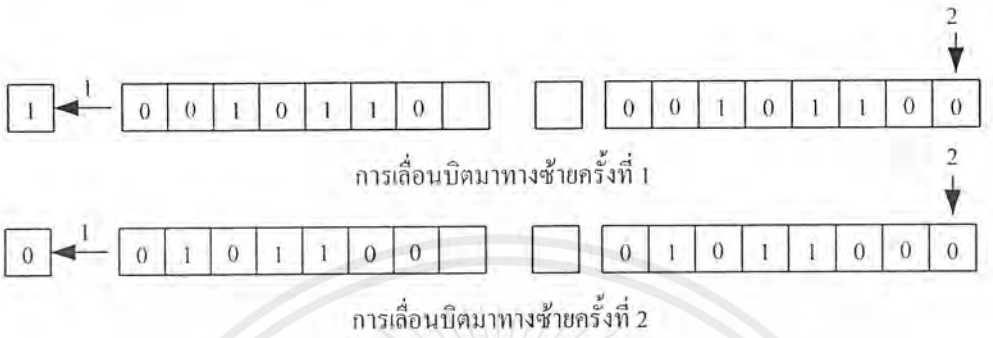
การหมุนบิตมาทางขวา (Rotate Bit Right) หมายถึง การเลื่อนทุกๆ บิตสายข้อมูลบิตมาทางขวาเท่ากับจำนวนบิตที่กำหนด และนำบิตที่ถูกเลื่อนดังกล่าวมาแทนที่บิตทางซ้ายมือที่ถูกเลื่อน เช่น ต้องการหมุนบิตของค่าตัวเลขฐานสอง 10010110 มาทางขวาจำนวน 2 บิต จะเป็นดังรูปที่ 2.19




**รูปที่ 2.19** การหมุนบิตทางขวา

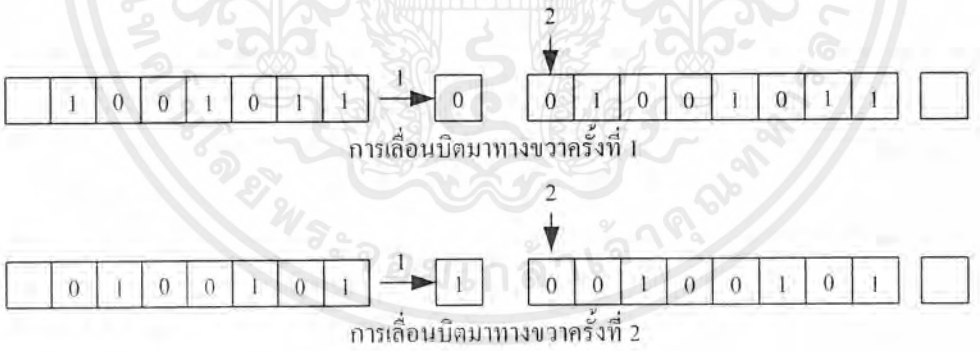
**6) การเลื่อนบิต (Shift Bit)** ในการภาษาแอสเซมบลี การเลื่อนบิตซ้าย และบิตขวาจะเทียบเท่ากับคำสั่ง SHL และ SHR ตามลำดับ การหมุนบิตสามารถแบ่งออกได้ 2 ลักษณะดังนี้ การเลื่อนบิตทางซ้าย (Shift Bit Left) หมายถึง การเลื่อนทุก ๆ บิตของสายข้อมูลบิตมาทางซ้ายเท่ากับจำนวนบิต

ที่กำหนด และแทนที่บิตทางขวามือที่ถูกเลื่อนด้วย 0 เช่น ต้องการเลื่อนบิตของตัวเลขฐานสอง 10010110 มาทางซ้ายจำนวน 2 บิต จะมีเป็นดังรูปที่ 2.20




**รูปที่ 2.20** การเลื่อนบิตทางซ้าย

การเลื่อนบิตทางขวา (Shift Bit Right) หมายถึง การเลื่อนทุกๆ บิตของสายข้อมูลบิตมาทางขวาเท่ากับจำนวนบิตที่กำหนด และแทนที่บิตทางซ้ายมือที่ถูกเลื่อนด้วย 0 เช่น ต้องการเลื่อนบิตของตัวเลขฐานสอง 10010110 มาทางขวาจำนวน 2 บิต จะมีเป็นดังรูปที่ 2.21




**รูปที่ 2.21** การเลื่อนบิตทางขวา

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

**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 2.22

| 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 2.22** การกลับลำดับตำแหน่งบิต

จากที่กล่าวมาข้างต้นทั้งหมดเป็นพื้นฐานข้อมูลที่เกี่ยวกับการประมวลผลบิตที่จำเป็นสำหรับการเขียนโปรแกรมที่ต้องเขียนติดต่อกับบอร์ดอิเล็กทรอนิกส์ หรือโปรแกรมประเภทบีบอัดข้อมูล

### 2.5.3 ไฟล์ไลบรารี Bits32.dll

ไฟล์ไลบารี Bits32.dll เป็นไฟล์ที่ถูกพัฒนาขึ้นมาสำหรับใช้งานร่วมกับวิชวลเบสิกรุ่น 5.0 และรุ่น 6.0 โดยจะประกอบด้วยกลุ่มฟังก์ชันที่เกี่ยวข้องกับประมวลผลบิตของข้อมูลขนาด 1 ไบต์ (สำหรับวิชวลเบสิกจะเทียบเท่ากับ Byte) ข้อมูลขนาด 2 ไบต์ (Word สำหรับวิชวลเบสิกจะเทียบเท่ากับ 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 2.3

**ตารางที่ 2.3** 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 |
| VbrcSHLB | เลื่อนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางซ้าย |
| VbrcSHRB | เลื่อนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางขวา |
| VbrcSwapBitB | กลับตำแหน่งของบิตของข้อมูลชนิดตัวเลขที่กำหนด |
| VbrcToggleBitB | กลับค่าของบิตตรงลำดับตำแหน่งของบิตในข้อมูลชนิดตัวเลขที่กำหนด |
| 3. ประมวลผลบิตชนิดข้อมูลตัวเลขขนาด 2 ไบต์ ประกอบด้วย | |
| VbrcBinStrW | แปลงข้อมูลตัวเลขให้เป็นข้อมูลสตริงของเลขไบนารี |
| 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 |

ตารางที่ 2.3 (ต่อ) ฟังก์ชันของไฟล์ไลบรารี Bits32.dll

| ชื่อฟังก์ชัน | หน้าที่ |
|---|---|
| VbrcSetBitD | กำหนดให้บิตมีค่า 1 หรือ 0 ตรงตำแหน่งของบิตที่กำหนด |
| VbrcROLD | หมุนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางซ้าย |
| VbrcRORD | หมุนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางขวา |
| VbrcSHLD | เลื่อนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางซ้าย |
| VbrcSHRD | เลื่อนข้อมูลบิตของข้อมูลชนิดตัวเลขที่กำหนดมาทางขวา |
| VbrcSwapBitD | กลับตำแหน่งของบิตของข้อมูลชนิดตัวเลขที่กำหนด |
| VbrcToggleBitD | กลับค่าของบิตตรงตำแหน่งของบิตในข้อมูลชนิดตัวเลขที่กำหนด |
| 5. แปลงข้อมูลชนิดตัวเลขกับข้อมูลชนิดสตริง | |
| VbrcCVBYT | แปลงข้อมูลชนิดสตริงเป็นชนิดตัวเลข |
| 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 |

## 2.6 โปรแกรมคอมไพเลอร์ไมโครโปรเซสเซอร์ Z-80

โปรแกรมคอมไพเลอร์ A180Z เป็นโปรแกรมที่ทำงานอยู่บนดอส โดยจะทำงานโดยการแปลงไฟล์ที่มีนามสกุล ASM ที่เขียนบนโปรแกรมอิดิเตอร์ทั่วไปแล้วจัดเก็บไฟล์เป็นนามสกุล ASM แล้วไปทำการคอมไพล์เป็นไฟล์นามสกุล OBJ และสร้างไฟล์นามสกุล LST โครงสร้างของไฟล์สกุล OBJ รูปแบบของข้อมูลแบบนี้มีลักษณะดังแสดงตารางที่ 2.4

ตารางที่ 2.4 รูปแบบของไฟล์นามสกุล OBJ

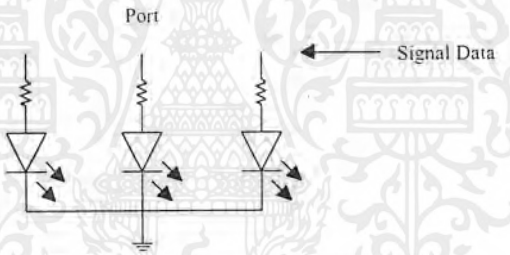
| ชื่อสัญลักษณ์ | จำนวนตัวอักษร | รายละเอียด |
|---|---|---|
| Record Mark | 1 | ในแต่ละบรรทัดของข้อมูลที่ต้องการบันทึกต้องเริ่มด้วยตัวโคลอน ( : ) |
| Record length | 2 | จำนวนข้อมูลที่ต้องการบันทึก |
| Address Field | 2 | แสดงแอดเดรสที่เก็บข้อมูลในไฟล์แรก ปกติมีค่า 00 |
| Record Type | 2 | ชนิดของข้อมูลที่ทำการบันทึก โดยถ้าเป็นข้อมูล<br>00 หมายถึง ข้อมูลของโปรแกรม<br>01 หมายถึง จบแฟ้มข้อมูล<br>02 หมายถึง แอดเดรสเพิ่มเติม<br>03 หมายถึง แอดเดรสเริ่มต้น |
| Datafield | เปลี่ยนแปลง | จะขึ้นอยู่กับรหัสชนิดของข้อมูลที่ทำการบันทึก<br>00 หมายถึง จำนวนข้อมูลที่ต้องการโปรแกรม<br>01 หมายถึง ไม่ใช้งาน (ว่าง)<br>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 |

## 2.7 อุปกรณ์ต่อร่วมกับไมโครโปรเซสเซอร์ Z-80

อุปกรณ์ต่างที่นำมาต่อเข้านำไมโครโปรเซสเซอร์ Z-80 จะเป็นส่วนอินพุต และเอาต์พุต ซึ่งจะเป็นอุปกรณ์ประเภทต่างๆ อย่าง เช่น ส่วนของอินพุต ก็จะมีส่วนของสวิตช์ และในส่วนของเอาต์พุตจะมี แอลอีดี ภาคแสดงผลแบบเจ็ดส่วน แอลซีดี หรือ มอเตอร์ เป็นต้น ซึ่งจะมีหลักการทำงานดังต่อไปนี้

### 2.7.1 แอลอีดีเมตริกซ์

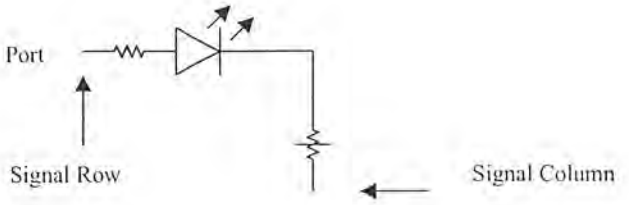
แอลอีดีเป็นอุปกรณ์แสดงผล โดยที่สามารถที่จะนำมาใช้ในการแสดงผล บอกสถานะต่าง ๆ ตามที่ต้องการ ซึ่งอุปกรณ์ประเภทนี้จะเป็นอุปกรณ์ที่พบเห็นโดยทั่วไป มีหลักการทำงานโดยการเปล่งแสงออกมา การต่อแอลอีดีมีหลายแบบขึ้นอยู่กับความต้องการ แต่ในที่นี้จะกล่าวถึงการต่อแอลอีดีเป็นการภาคแสดงผล (Display) อย่างง่ายๆ ก่อน คือ ภาคแสดงผลแบบแถวเดียว ลักษณะการต่อวงจรดังแสดงในรูปที่ 2.23




**รูปที่ 2.23** แอลอีดีแสดงผลแบบแถวเดียว

จากรูปที่ 2.23 มีหลักการทำงานโดยสามารถควบคุมสัญญาณเพียงด้านเดียว คือ ส่งแต่สัญญาณข้อมูล (Signal data) แล้วหน่วงเวลาไว้ระยะหนึ่ง แล้วจึงส่งสัญญาณตัดต่อไป ซึ่งจะทำให้สามารถมองเห็นเป็นลักษณะไฟวิ่งได้

นอกจากนี้สามารถที่ทำควบคุมไฟให้เห็นเป็นตัวอักษรหรือรูปภาพได้ การต่อแบบนี้จะพบเห็นกันมาก เนื่องจากปัจจุบันได้มีการนำไปใช้งานกันอย่างแพร่หลาย การต่อวงจรสามารถทำได้โดยทำการเพิ่มลอจิกการควบคุมให้กับแอลอีดี ซึ่งแทนที่จะควบคุมการติดดับเพียงด้านเดียวก็เป็นสองด้าน โดยมีลักษณะการต่อวงจรดังแสดงรูปที่ 2.24



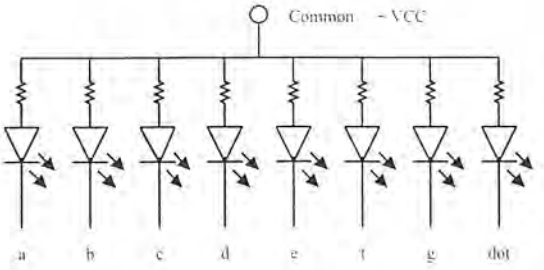

**รูปที่ 2.24** แอลอีดีแสดงผลแบบสองแถว

จากรูปที่ 2.24 เมื่อต่อสัญญาณสัญญาณตามจุดต่าง ๆ ได้เท่ากับแถวคูณคอลัมน์ ทำให้เกิดตัวอักษรวิ่งได้โดยส่งสัญญาณแถว และคอลัมน์ให้สอดคล้องกัน เรียกว่า วิธีการมัลติเพล็กซ์ (Multiplex) คือ ส่งข้อมูลตัวที่ 1 กำหนดให้คอลัมน์ที่ 1 ทำงาน แล้วหน่วงเวลาไว้ระยะหนึ่ง จากนั้นส่งข้อมูลตัวที่ 2 แล้วกำหนดให้คอลัมน์ตัวที่ 2 ทำงาน แล้วหน่วงเวลา ทำอย่างนี้ไปเรื่อยจนครบคอลัมน์สุดท้าย ซึ่งจะได้ตัวอักษรหรือรูปภาพ แต่ภาพที่เกิดจะเกิดในช่วงเวลาสั้น ๆ เพราะทำงานเร็วจำเป็นต้องกลับไปเริ่มส่งข้อมูลแบบเดิมระยะหนึ่ง เพื่อให้เห็นภาพชัดติดตา นั้นจะเป็นการทำให้ตัวอักษรเลื่อน โดยการเลื่อนตำแหน่งข้อมูลตัวแรกไป 1 ตำแหน่ง แล้วสแกนกลับไปวนข้อมูลระยะหนึ่ง แล้วเพิ่มตำแหน่งของข้อมูลที่เพิ่มแล้วขึ้นไปอีก 1 ทำจนถึงตำแหน่งของข้อมูลตัวอักษรสุดท้าย

### 2.7.2 ตัวแสดงผลแบบเจ็ดส่วน

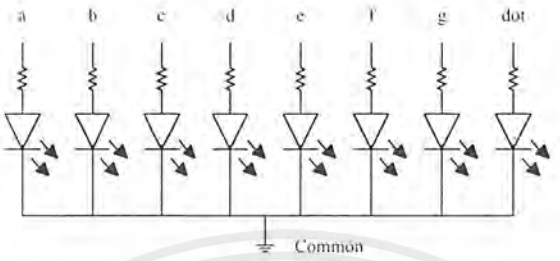
ตัวแสดงผลแบบเจ็ดส่วนเป็นอุปกรณ์แสดงผลอย่างหนึ่ง ซึ่งโครงสร้างของตัวแสดงผลแบบเจ็ดส่วนประกอบด้วยแอลอีดีจำนวน 7 ตัว หรือ 8 ตัว หรือมากกว่าในกรณีชนิดพิเศษซึ่ง ตัวแสดงผลแบบเจ็ดส่วนถ้าแบ่งตามชนิดของสารต่อแอลอีดีภายใน แบ่งได้เป็น 2 ชนิด คือ

**1) การต่อร่วมแอโนด** (Common Anode) คือ นำขาอาโนดของแอลอีดีมาต่อร่วมกัน ลักษณะการต่อแอลอีดีภายในของชนิดแอโนด ดังแสดงในรูปที่ 2.25




**รูปที่ 2.25** ตัวแสดงผลแบบเจ็ดส่วนชนิดแอลอีดีต่อร่วมแอโนด

**2) การต่อร่วมแคโทด** (Common Cathode) คือ นำขาคาโทดของแอลอีดีต่อร่วมกัน ลักษณะการต่อแอลอีดีภายในของชนิดการต่อร่วมแคโทด ดังแสดงในรูปที่ 2.26




**รูปที่ 2.26** ตัวแสดงผลแบบเจ็ดส่วนชนิดแอลอีดีต่อร่วมแอโนด

### 2.7.3 แอลซีดี

อุปกรณ์ในปัจจุบันนี้ในส่วนแสดงผลนั้นจะใช้แอลซีดีเสียเป็นส่วนใหญ่ ไม่ว่าจะเป็นเครื่องเล่นวีดีโอ วิทยุ เครื่องเสียง เครื่องถ่ายเอกสาร เป็นต้น จะนำเอาแอลซีดีนั้นไปเป็นส่วนประกอบในการนำไปแสดงผลบนอุปกรณ์เหล่านั้น ซึ่งตามลักษณะการนำจะแบ่งแอลซีดีนี้ออกเป็นส่วนต่างๆ ดังนี้

1) ส่วนของตัวอักษร (Character LCD Module)

2) ส่วนของแสดงกราฟิก (Graphic LCD Module)

3) ส่วนของภาคแสดงผลชนิดเจ็ดส่วน (Segment Display Type LCD Module)

ซึ่งในแต่ละส่วนของแอลซีดีนี้จะมีลักษณะหน้าที่การทำงานที่แตกต่างกัน โดยในแต่ละแบบนั้นก็จะมีส่วนประกอบใหญ่ ๆ แบ่งได้เป็นดังนี้

1) แบบแอลอีดีเมตริกซ์ (Dot matrix LCD) เป็นตัวแสดงผลให้มองเห็นในลักษณะการปิดและเปิดตัวเองกับแสง ก็คือ ส่วนของที่เป็นตัวกระจกบรรจุผลึก

2) ตัวขับ (Driver) เป็นตัวรับสัญญาณจากตัวควบคุมมาขับผลึกแอลซีดีอีกครั้งหนึ่ง

3) ตัวควบคุม (Controller) เป็นตัวรับข้อมูลจากอุปกรณ์ภายนอกมาและจัดการควบคุม แอลซีดีให้ทำงานแสดงผลต่าง ๆ เช่น การลบภาพ, การเกิดตัวอักษร เป็นต้น

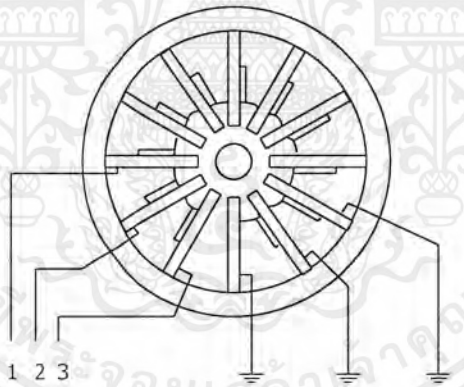
## 2.7.4 สเต็ปปิ้งมอเตอร์

**1) ประเภทของสเต็ปปิ้งมอเตอร์** เราสามารถแบ่งสเต็ปปิ้งมอเตอร์ตามพื้นฐานได้ 3 ชนิดคือ

**1.1) ชนิดวาริเอเบิลรีลักแตนซ์ (Variable Reluctance หรือ VR)** สเต็ปปิ้งมอเตอร์ชนิดนี้มีข้อเสียคือ เมื่อมีสเตปในการหมุนสูง จึงทำให้ความถูกต้องของตำแหน่ง และทำงานได้ดี สามารถทดสอบเพื่อให้ทราบว่าเป็นสเต็ปเปอร์มอเตอร์ชนิดนี้ได้ง่ายมาก โดยใช้มือหมุนที่เพลาของมอเตอร์ ซึ่งจะไม่เกิดปรากฏการณ์ทางแม่เหล็ก (Magnetism) จะทำให้หมุนได้โดยไม่ติดขัด แตกต่างจากชนิดอื่น คือเมื่อทำการหมุนจะรู้สึกขัด ๆ เหมือนเป็นฟันเฟือง

VR สเต็ปปิ้งมอเตอร์ที่มีสแต็คเดียวจะมีโรเตอร์เดียวเมื่อเทียบกับ VR สเต็ปปิ้งมอเตอร์แบบมีหลายสแต็ค หมายถึงมีหลายโรเตอร์ โรเตอร์และสเตเตอร์ทำจากสารแม่เหล็ก ขั้วของสเตเตอร์ที่อยู่ตรงกันข้ามจะพันด้วยขดลวดลักษณะที่ต่างกัน เพื่อให้มีความสมดุลย์ระหว่างเส้นแรงแม่เหล็กเข้า และออกจากโรเตอร์

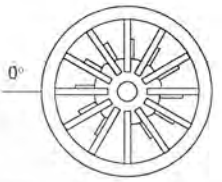
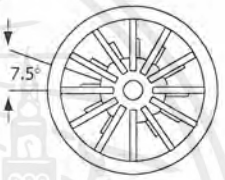
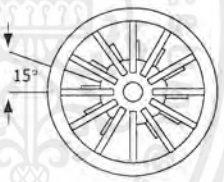
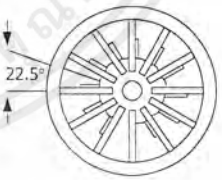
ตัวอย่างโครงสร้างของสเต็ปปิ้งมอเตอร์แบบค่ารีลักแตนซ์แปรค่าสแต็คเดียว หรือที่เรียกสั้นๆ ว่า VR สเต็ปปิ้งมอเตอร์ที่มีสแต็คเดียวแสดงได้ในรูปที่ 2.27




**รูปที่ 2.27** VR สเต็ปปิ้งมอเตอร์แบบมีสแต็คเดียว

ลำดับการทำงานที่สมบูรณ์แสดงในตารางที่ 2.5 เมื่อตำแหน่งเริ่มต้นของซี่ฟันของโรเตอร์จะเป็นสีดำเพื่อให้เราทำความเข้าใจได้ชัดเจนถึงการหมุนของโรเตอร์ในทิศทาง CW

**ตารางที่ 2.5** ลำดับของการสวิตช์ 3 สเต็ปของ VR สเต็ปปิ้งมอเตอร์แบบสแต็คเดียว และตำแหน่งของโรเตอร์

| การเรียงลำดับเฟส | ตำแหน่งของโรเตอร์และเส้นแรงแม่เหล็ก |
|---|---|
| ตำแหน่งโรเตอร์เริ่มต้น :<br>1) เฟส $\phi_1$ ได้รับพลังงาน<br>2) ซี่ฟันของโรเตอร์จะอยู่ในแนวซี่ฟันที่ 1, 4, 7, 10 ของสเตเตอร์ |  |
| สเต็ปที่ 1 :เฟส $\phi_3$ ได้รับพลังงาน<br>ซี่ฟันของโรเตอร์จะอยู่ในแนวซี่ฟันที่ 2, 5, 8, 11 ของสเตเตอร์<br>1) โรเตอร์จะเคลื่อนไปในทิศทาง CW เป็นมุม 7.5°<br>(1/3 ช่วงระหว่างซี่ฟันของโรเตอร์) |  |
| สเต็ปที่ 2 :เฟส $\phi_2$ ได้รับพลังงาน<br>ซี่ฟันของโรเตอร์จะอยู่ในแนวซี่ฟันที่ 3, 6, 9, 12ของสเตเตอร์<br>1) โรเตอร์จะเคลื่อนไปในทิศทาง CW รวมเป็นมุม 15° |  |
| สเต็ปที่ 3 :เฟส $\phi_1$ ได้รับพลังงาน<br>ซี่ฟันของโรเตอร์จะอยู่ในแนวซี่ฟันที่ 1, 4, 7, 10 ของสเตเตอร์<br>1) โรเตอร์จะเคลื่อนไปในทิศทาง CW เป็นมุม 7.5°<br>(เคลื่อนไปได้ 1 ช่วงระหว่างซี่ฟันของโรเตอร์) |  |

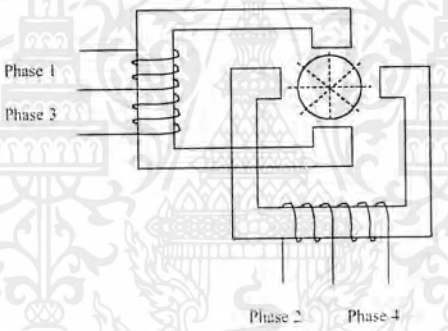
**1.2) ชนิดเพอร์มาเนนต์แม็กเน็ต (Permanent Magnet หรือ PM)** สเต็ปปิ้งมอเตอร์ชนิดนี้มีข้อดีคือ มีความถูกต้องของตำแหน่งเมื่อเปรียบเทียบกันกับสเต็ปปิ้งมอเตอร์ชนิดอื่น

**1.3) ชนิดไฮบริดจ์ (Hybrid)** เป็นชนิดที่นิยมใช้มากที่สุดในเครื่องคอมพิวเตอร์โดยจะใช้ในส่วนของการขับเคลื่อนหัวอ่านดิสก์ไดร์ว สเต็ปปิ้งมอเตอร์ชนิดนี้มีข้อดีคือ มีโครงสร้างภายในคือเป็นสเตเตอร์ซึ่งเป็นชนิดวาริเอเบิลรีลักแตนซ์ ส่วนโรเตอร์ที่เป็นชนิดเพอร์มาเนนต์แม็กเนตนำ

มาประกอบเข้าด้วยกัน ทำให้เป็นมอเตอร์ชนิดนี้มีแรงยึดหน่วงสูง มีแรงบิดดึง และผลักดี และยังคงทำงานได้ดีแม้ว่าจะมีจำนวนสเต็ปต่อรอบในการหมุนสูงก็ตาม

ไฮบริดจ์สเต็ปปิ้งมอเตอร์ (HSM) มีคุณลักษณะผสมของ PM และ VR สเต็ปปิ้งมอเตอร์ โครงสร้างของ HSM ประกอบด้วย 2 ตอนกับแกนแม่เหล็กอยู่ระหว่าง 2 ตอนแต่ละตอนประกอบด้วยซี่ฟันของโรเตอร์ และโพลของสเตเตอร์ที่มีซี่ฟันเช่นกัน และพันด้วยขดลวดรายละเอียดโครงสร้างของสเตเตอร์ และโรเตอร์ของแต่ละตอน

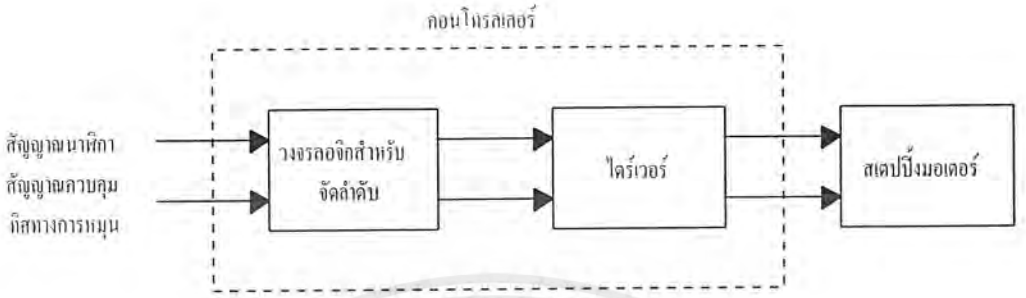
**1.4) ชนิดแรเอิร์ธเพอร์มาเนนต์แมกเน็ต (Rare Earth Permanent Magnet)** หรือที่เรียกกันว่าชนิดดิสก์แม็กเน็ต สเต็ปปิ้งมอเตอร์ (Disk Magnet Stepper) การทำงานจะเป็นแบบเดิมแต่โครงสร้างเป็นแบบใหม่จะทำให้เกิดความเฉื่อยต่ำมาก แต่มีอัตราเร่งสูง มอเตอร์ชนิดนี้จึงจัดเป็นมอเตอร์ที่มีประสิทธิภาพสูงทั้งในด้านแรงบิดดึง, กำลังทางกลที่ได้ของมอเตอร์, ความถูกต้องของตำแหน่งสูงมาก และความเร็วในการเริ่มหมุน และหยุดสูง อีกทั้งมีการสูญเสียของกำลังงานต่ำ




**รูปที่ 2.28** สเต็ปปิ้งมอเตอร์แบบ 4 เฟส (Phase) แบบยูนิโพลาร์เพอร์มาเนนแม็กเนต

**2) โครงสร้าง และการทำงานของสเต็ปปิ้งมอเตอร์**

สเปปิ้งมอเตอร์เป็นอุปกรณ์จำพวกเชิงกลไฟฟ้า ที่มีอินพุตเป็นกลุ่มไบนารี่โวลต์เตจ และเอาต์พุตเป็นลักษณะการเคลื่อนที่แบบเชิงมุม หรือหมุนไปเป็นสเต็ป (แต่ละสเต็ปอยู่ในช่วง 0.1 ถึง 30 องศา ขึ้นอยู่กับโครงสร้างของสเต็ปปิ้งมอเตอร์) ตามสัญญาณพัลซ์ที่ป้อนให้กับขดสเตเตอร์ซึ่งเกิดแรงผลักต่อโรเตอร์หมุนไป แต่ลักษณะของสเต็ปปิ้งมอเตอร์จะมีขดของสเตเตอร์อยู่หลายขดซึ่งเรียกว่า "เฟส" ฉะนั้นเมื่อป้อนสัญญาณที่เป็นพัลซ์ในลักษณะเป็นลำดับของเลขฐานสองผ่านวงจรขับ (Driver) จะทำให้โรเตอร์หมุนได้อย่างต่อเนื่องดังแผนผังในรูปที่ 2.29




**รูปที่ 2.29** แผนผังการควบคุมสเต็ปปิ้งมอเตอร์

จากแผนผังสเต็ปปิ้งมอเตอร์ในรูปที่ 2.29 จะเห็นว่าในการควบคุมสเต็ปปิ้งมอเตอร์ให้สามารถเคลื่อนที่ได้ตามที่ต้องการนั้น จะต้องทำการส่งสัญญาณทางด้านอินพุตซึ่งเป็นสัญญาณนาฬิกา (Clock Pulse) และส่งสัญญาณอินพุตซึ่งเป็นสัญญาณ ที่ใช้สำหรับควบคุมทิศทางการหมุนของสเตปปิ้งมอเตอ์เสียก่อนจึงจะสามารถทำการควบคุมสเต็ปปิ้งมอเตอร์ได้ สัญญาณที่ได้นั้นจะต้องผ่านวงจรลอจิกเพื่อทำการจัดลำดับเสียก่อน แล้วส่งเข้าไปขยายสัญญาณที่วงจรขับมอเตอร์เพื่อให้มีแรงดันเพียงพอในการขับสเตปปิ้งมอเตอร์ได้

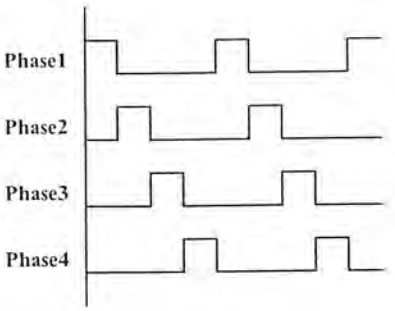
**3) การกระตุ้นและการควบคุมการหมุนของสเต็ปปิ้งมอเตอร์**

การทำให้สเต็ปปิ้งมอเตอร์เคลื่อนไปทีละสเต็ป ทำได้โดยการจ่ายกำลังไฟฟ้าไปยังขดลวดแต่ละขดบนสเตเตอร์ ซึ่งจะต้องป้อนแบบซีเควนเชียลในรูปแบบที่ถูกต้อง การป้อนพัลซ์กระตุ้นสเต็ปปิ้งมอเตอร์สามารถทำได้ 3 รูปแบบคือ

**3.1) แบบเวฟ (Wave)** เป็นการป้อนกระแสให้กับขดลวดแต่ละขดของสเต็ปปิ้งมอเตอร์ทีละขดเรียงลำดับกันได้ ลักษณะการขับแบบนี้จะทำให้แรงบิดน้อย

**ตารางที่ 2.6** การจ่ายกระแสแบบเวฟให้กับขดลวดแต่ละขดของสเต็ปปิ้งมอเตอร์

| Step | Phase | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| 1 | 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 1 |

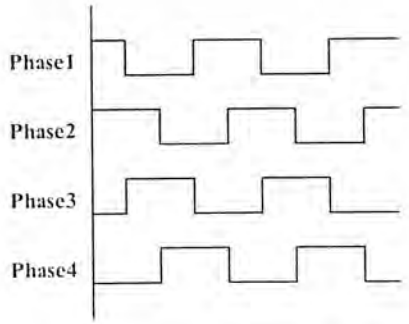



รูปที่ 2.30 การจ่ายกระแสแบบเวฟให้กับขดลวดแต่ละขดของสเต็ปปิ้งมอเตอร์

**3.2) แบบ 2 เฟส (Two Phase)** มีลักษณะคล้ายกับแบบเวฟ แต่การกระตุ้นโดย จ่ายกำลังไฟฟ้าไปที่ขดลวด 2 ขด ที่อยู่ใกล้กันในเวลาเดียวกันเรียงถัดกันไป เช่นเดียวกับแบบเวฟขึ้นอยู่กับทิศทางของการหมุน การเพิ่มจำนวนขดลวดของขดลวดที่ถูกกระตุ้นจะทำให้เพิ่มแรงบิดได้มากกว่าแบบเวฟ โรเตอร์จะเคลื่อนที่ด้วยแรงดึงเต็มที่ด้วยแรงดึงจากทั้งสองขดลวดที่ถูกกระตุ้นพร้อมกัน ข้อเสียของการกระตุ้นแบบนี้คือ ต้องจ่ายกำลังไฟฟ้ามากขึ้น การทำงานต่าง ๆ จะแสดงในรูปที่ 2.31

ตารางที่ 2.7 การจ่ายกระแสแบบ 2 เฟสให้กับขดลวดแต่ละขดของสเต็ปปิ้งมอเตอร์

| Step | Phase | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| 1 | 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 1 |

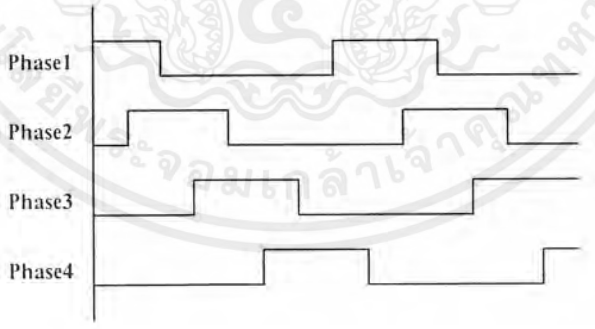



รูปที่ 2.31 การจ่ายกระแสแบบ 2 เฟสให้กับขดลวดแต่ละขดของสเต็ปปิ้งมอเตอร์

**3.3) แบบครึ่งสเต็ป (Half Step)** เป็นแบบที่ได้จากการผสมระหว่างการกระตุ้นแบบเวฟและแบบ 2 เฟส ดังแสดงในรูปที่ 2.15 เพื่อเพิ่มจำนวนสเต็ปต่อรอบอีกหนึ่งเท่าตัว แรงบิดที่ได้จากการกระตุ้นแบบนี้จะเพิ่มมากขึ้นอีกเพราะช่วงสเต็ปมีระยะสั้นลง และแต่ละสเต็ปเกิดจากแรงดึงของขดลวด 2 ขดที่กระตุ้นพร้อมกัน ความถูกต้องของตำแหน่งจึงมีเพิ่มขึ้น ที่สำคัญการกระตุ้นแบบนี้จะต้องทำการหมุน 2 สเต็ปจึงเท่ากับ 1 สเต็ปของ 2 แบบแรก ส่วนแหล่งจ่ายกำลังต้องใช้เหมือนกับแบบ 2 เฟส

ตารางที่ 2.8 การจ่ายกระแสแบบครึ่งสเต็ป ให้กับขดลวดแต่ละขดของสเต็ปปิ้งมอเตอร์

| STEP | PHASE | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| 1 | 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 |
| 4 | 0 | 0 | 1 | 0 |
| 5 | 0 | 0 | 1 | 1 |
| 6 | 0 | 0 | 0 | 1 |
| 7 | 1 | 0 | 0 | 1 |
| 8 | 1 | 0 | 0 | 0 |




รูปที่ 2.32 การจ่ายกระแสแบบครึ่งสเต็ป ให้กับขดลวดแต่ละขดของสเต็ปปิ้งมอเตอร์

**4) ประโยชน์ของสเต็ปปิ้งมอเตอร์**

ในระบบควบคุมตำแหน่งที่ใช้สเต็ปปิ้งมอเตอร์นั้นมีข้อดีอยู่หลายประการคือ

4.1) ในลักษณะการควบคุมแบบไม่ต้องการการป้อนกลับ ไม่ว่าจะเป็นการควบคุมความเร็วหรือตำแหน่ง

4.2) ความผิดพลาดที่เกี่ยวกับตำแหน่งแทบไม่มีเลย เนื่องจากการเคลื่อนที่ของสเต็ปปิ้งมอเตอร์นั้นเคลื่อนที่เป็นสเต็ปด้วยจำนวนองศาที่มีค่าแน่นอน

4.3) สเต็ปปิ้งมอเตอร์จะถูกนำมาใช้กับเครื่องมือที่ต้องการความละเอียดแม่นยำและใช้อยู่ในเครื่องมือประเภทดิจิตอล เช่น เครื่องวาดรูป เครื่องคอมพิวเตอร์ นิวเมอริคอลคอนโทรล (Computer Numerical Control) หรือ CNC

4.4) ไม่จำเป็นต้องใช้วงจรแปลงดิจิตอลเป็นแอนาลอก เมื่อทำการอินเตอร์เฟสกับไมโครคอมพิวเตอร์

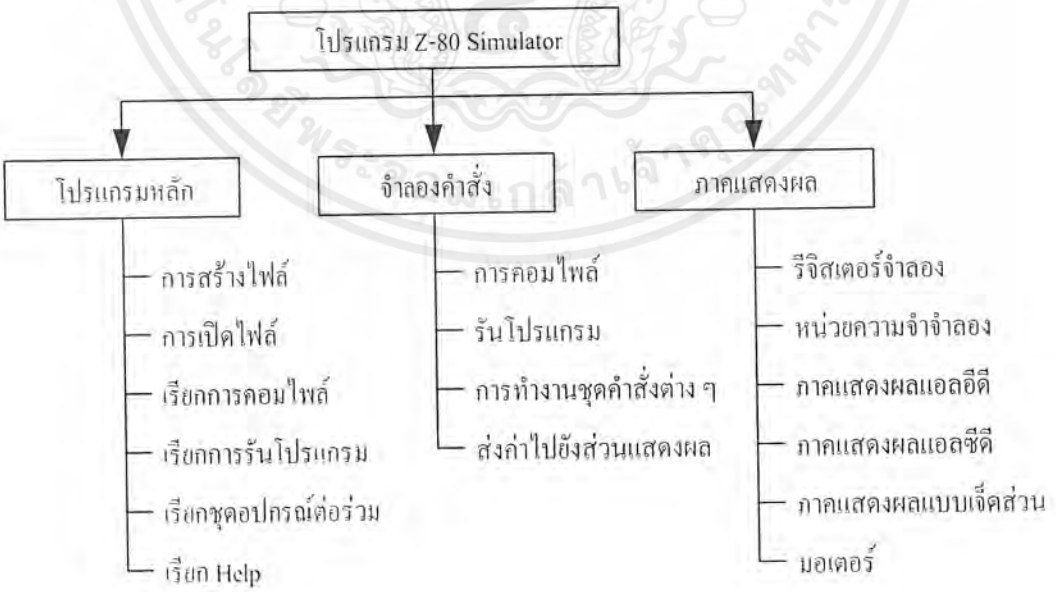
# บทที่ 3
# การออกแบบ การสร้าง และการทำงาน

## 3.1 กล่าวนำ

การออกแบบและการสร้างโปรแกรมได้แบ่งการทำงานออกเป็นส่วน ๆ เพื่อให้ง่ายต่อการเขียนโปรแกรมและพัฒนาโปรแกรมต่อไปในอนาคต ซึ่งได้แบ่งโปรแกรมออกเป็น 2 ส่วนคือ ส่วนของจำลองการทำงานของภายใน และส่วนของการแสดงผลติดต่อกับผู้ใช้งาน ซึ่งในการเขียนโปรแกรมของติดต่อกับผู้ใช้งานนั้นได้ใช้โปรแกรมวิชวลเบสิกเขียนเป็นตัวติดต่อกับผู้ใช้ รวมทั้งในส่วนของการเขียนจำลองคำสั่งต่าง ๆ ด้วย ซึ่งในรายละเอียดของการออกแบบและการสร้างสามารถอธิบายได้ดังต่อไปนี้

## 3.2 โครงสร้างโปรแกรม

โครงสร้างหลักของโปรแกรมได้แบ่งออกเป็น 3 ส่วน คือ ส่วนของโปรแกรมหลัก ส่วนของการจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80 และในส่วนของภาคแสดงผล ซึ่งโครงสร้างของโปรแกรมจะแสดงได้ดังรูปที่ 3.1

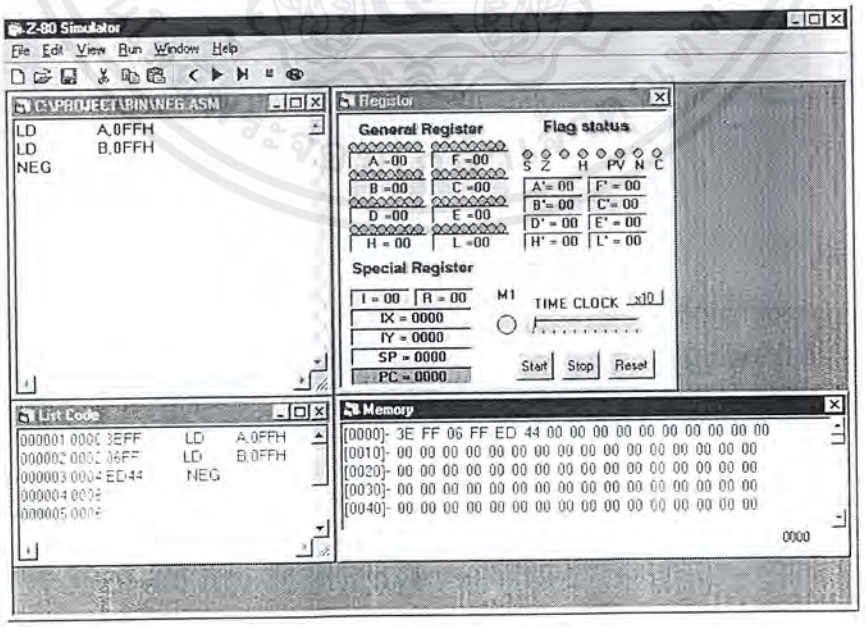



รูปที่ 3.1 โครงสร้างของโปรแกรม

จากรูปที่ 3.1 โครงสร้างของโปรแกรมแบ่งออกเป็น 3 ส่วน คือ ส่วนของโปรแกรมหลักจะเป็นส่วนที่ใช้ติดต่อกับผู้ใช้โดยตรง จะทำหน้าที่เกี่ยวกับการจัดการไฟล์ การเขียนเท็กซ์ไฟล์ ซึ่งจะทำหน้าที่เสมือนโปรแกรมอิดิเตอร์ทั่วไปนั่นเอง คือ มีการสร้างไฟล์ใหม่ การเปิดไฟล์ ซึ่งเมื่อมีการสร้างไฟล์ก็จะสามารถที่เรียกการคอมไพล์ให้อยู่ในรูปของโปรแกรมของไมโครโปรเซสเซอร์ Z-80 และเรียกการจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80 โดยการรันโปรแกรม ซึ่งในส่วนของโปรแกรมหลักนี้จะเป็นตัวเชื่อมต่อระหว่างการจำลองคำสั่ง และภาคแสดงผล ในส่วนของการจำลองคำสั่งจะมีหน้าที่ในการคอมไพล์ไฟล์เท็กซ์ที่ได้จากโปรแกรมหลักให้อยู่ในรูปแบบของการที่จะนำไปใช้จำลองการทำงานต่อไป และจะทำหน้าจำลองการทำงานของชุดคำสั่งต่างที่มีอยู่ในไมโครโปรเซสเซอร์ Z-80 ให้เหมือนกับทำงานด้วยไมโครโปรเซสเซอร์ Z-80 จริง โดยการรันโปรแกรม แล้วจะส่งผลไปยังภาคการแสดงผล ส่วนของภาคแสดงผลจะทำหน้าที่ในการจำลองหน่วยความจำ รีจิสเตอร์ และอุปกรณ์ต่อร่วมกับไมโครโปรเซสเซอร์ Z-80 ซึ่งในโปรแกรมนี้จะมีส่วนของภาคแสดงผลแอลอีดี ภาคแสดงผลแอลซีดี ภาคแสดงผลแบบเจ็ดส่วน และการทำงานของดีซีมอเตอร์ และสเต็ปปิ้งมอเตอร์

## 3.3 การออกแบบโปรแกรมหลัก

ส่วนของโปรแกรมหลักเป็นส่วนที่ติดต่อกับผู้ใช้โดยตรง ซึ่งจะทำการออกแบบให้มีความสะดวกต่อการเรียกใช้งาน ส่วนของโปรแกรมหลักแสดงได้ดังในรูปที่ 3.2




รูปที่ 3.2 หน้าต่างของโปรแกรมหลัก

จากรูปที่ 3.2 ส่วนของโปรแกรมหลักนั้นมีการทำงานดังในแผนผังโครงสร้างการทำงานในรูปที่ 3.1 ส่วนต่างของโปรแกรมจะมีหน้าที่และการทำงานที่แตกต่างกัน จะพบว่าโปรแกรมหลักได้แบ่งส่วนของการทำงานแสดงผลติดต่อกับผู้ใช้ของโปแกรมหลักแบ่งออกเป็น 4 ส่วนดังนี้

1) เมนูบาร์และทูลบาร์

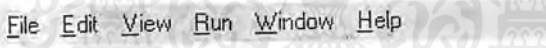
2) หน้าต่างรีจิสเตอร์

3) หน้าต่างหน่วยความจำ

4) หน้าต่างอิดิเตอร์

### 3.3.1 ส่วนของเมนูบาร์ และทูลบาร์ของโปรแกรม

ส่วนของเมนูบาร์ และทูลบาร์เป็นส่วนที่ทำให้ผู้ใช้สามารถใช้งานได้อย่างสะดวกสบาย โดยการคลิกปุ่ม หรือเมนูก็สามารถทำงานได้ โดยส่วนประกอบของเมนูบาร์ และทูลบาร์แสดงได้ดังรูปที่ 3.3 และ รูปที่ 3.4 ตามลำดับ




**รูปที่ 3.3** เมนูบาร์ของโปรแกรมหลัก

**รูปที่ 3.4** ทูลบาร์ของโปรแกรมหลัก

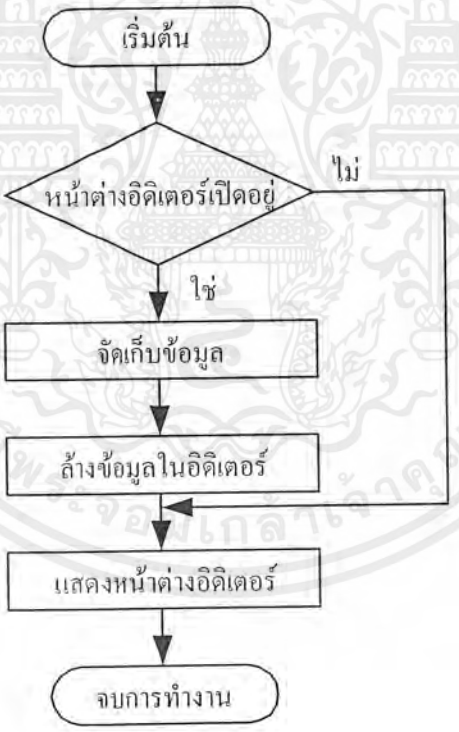
จากรูปที่ 3.3 เมนูบาร์จะประกอบไปด้วยเมนูหลักทั้งหมด 6 เมนูหลัก โดยแต่ละเมนูหลักจะมีเมนูย่อยอยู่ด้วย โดยจะแบ่งไปตามประเภทของคำสั่งของเมนูย่อยว่ามีการทำงานเป็นอย่างไร ซึ่งจะเป็นการง่ายต่อการที่จะเรียกใช้งานได้ง่าย และสะดวกต่อการเรียกใช้งานได้ ซึ่งลักษณะของเมนูบาร์นั้นจะเป็นลักษณะเหมือนกับเมนูของโปรแกรมโดยทั่วไป ประกอบไปด้วยส่วนเมนูไฟล์จะจัดการเกี่ยวกับการเปิดและสร้างไฟล์ใหม่ และการจบการทำงาน ส่วนของเมนูอิดิเตอร์จะเป็นการจัดการเกี่ยวกับข้อความในเท็กบอกซ์ เป็นต้น ในส่วนของเมนูบาร์ได้แบ่งเป็นเมนูหลัก และเมนูย่อยแต่ละเมนูนั้นทำหน้าที่ 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 |

โดยที่ส่วนของเมนูย่อยนั้นได้มีการออกแบบ และการทำงานในแต่ละส่วนของคำสั่งที่มีการทำงานที่สำคัญดังต่อไปนี้

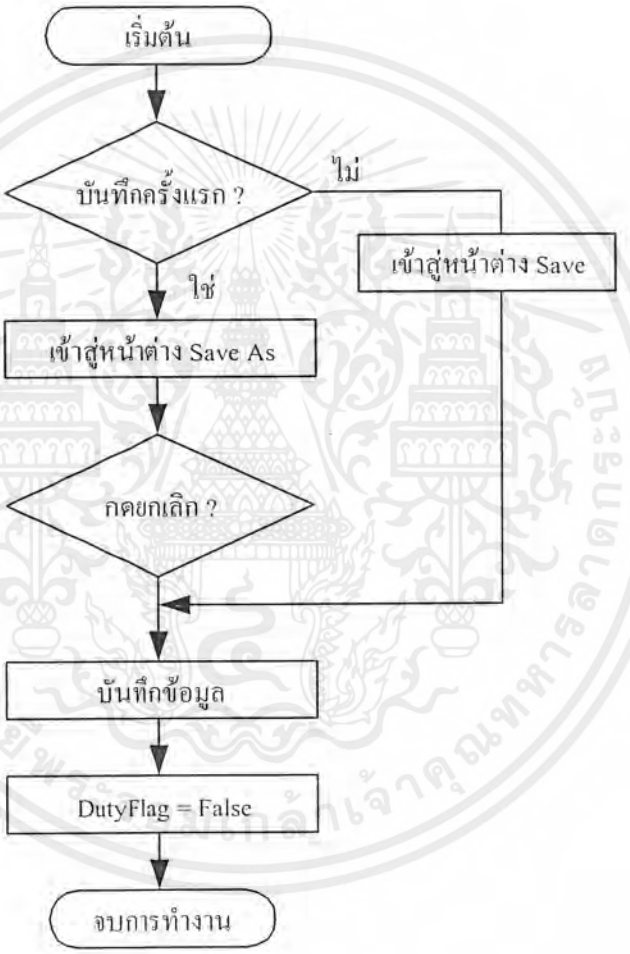
**1) คำสั่ง New** เป็นคำสั่งที่ให้มีการสร้างไฟล์ใหม่ โดยจะมีการตรวจสอบว่าในขณะนี้ ได้มีการเปิดหน้าต่างของการเขียนโปรแกรมอยู่หรือเปล่า หากมีการเปิดหรือสร้างไฟล์อยู่ ก็จะ ให้ทำการจัดเก็บข้อมูลก่อนที่จะทำการเปิดหน้าต่าง อิดิเตอร์ใหม่ ขั้นตอนในการเลือกคำสั่ง New ได้จากการคลิกไปที่เมนูบาร์ ซึ่งส่วน File แล้วคลิกเลือกลงมาก็จะเห็น ส่วนของเมนูย่อย ก็คลิกเลือก New ขั้นตอนในการทำงานดังแสดงในรูปที่ 3.4 โดยเริ่มจาก การตรวจสอบว่าในขณะนั้น ได้มีการเปิดการสร้างไฟล์อยู่หรือเปล่า หากพบว่าไม่มีการเปิดอิดิเตอร์อยู่ ก็จะทำให้มีการสร้างไฟล์ใหม่ขึ้นมา หากพบว่ามีการเปิดอิดิเตอร์ขึ้นมาแล้ว ก็จะทำการตรวจสอบว่าได้มีการจัดเก็บไฟล์ที่เปิดอยู่แล้วหรือไม่ หากไม่มีก็จะทำการให้มีการบันทึกข้อมูลเกิดขึ้น ถ้ามีการบันทึกข้อมูลไปแล้ว ก็จะทำการเปิด หน้าต่างอิดิเตอร์เพื่อสร้างไฟล์ใหม่ขึ้น โดยจะลบข้อมูลที่มีอยู่ในเท็กบ็อกซ์ออกก่อน




รูปที่ 3.5 แผนผังการทำงานของคำสั่ง New

**2) คำสั่ง Save** จะมีการตรวจสอบว่ามีการบันทึกข้อมูลไปก่อนหน้านี้หรือยัง หากยังไม่การบันทึกก็จะเข้าไปสู่ การ Save As ซึ่งเป็นการบันทึกไฟล์ครั้งแรก แต่ถ้าหากมีการบันทึกมาแล้วก็จะ

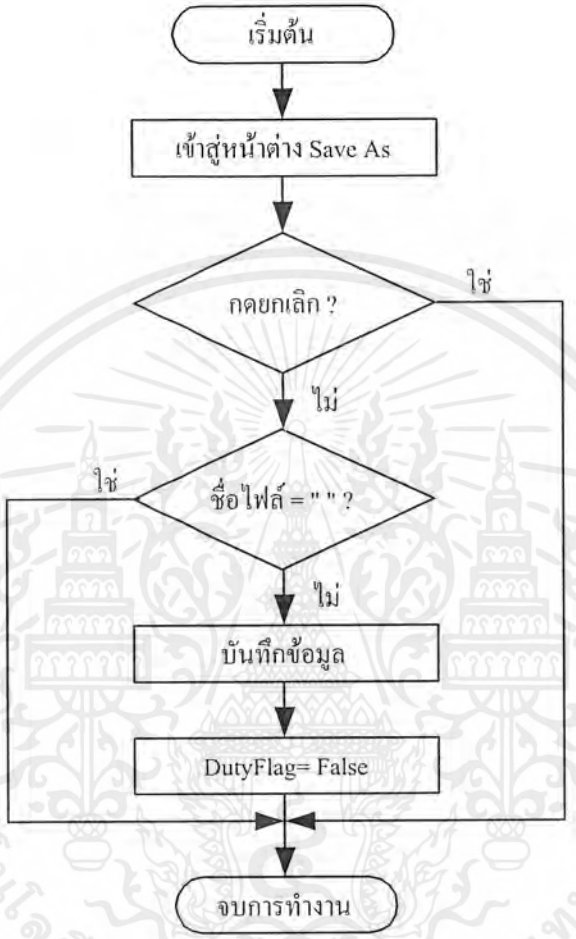
ทำการบันทึกไฟล์ทับข้อมูลเดิม คำสั่ง Save จะทำในกรณีที่มีการเปิดไฟล์ที่มีอยู่แล้วมาทำการแก้ไข ซึ่งถ้าหากไม่มีการแก้ไขก็จะทำการบันทึกไฟล์ข้อมูลที่ซึ่งเหมือนเดิมลงในไฟล์เดิมอีกครั้ง หากพบว่าเป็นการเปิดอิดิเตอร์ขึ้นมาเพื่อทำการสร้างไฟล์ใหม่นั้นจะทำการเตือนให้มีการบันทึกก่อน การเขียนโปรแกรมในส่วนนี้จะเป็นการเรียกใช้ในส่วนของ คอมมอนไดอะลอกของวิชวลเบสิกเข้ามาเป็นตัวจัดการ หากพบว่าเป็นการบันทึกไฟล์เป็นครั้งแรก ซึ่งกระบวนการดังกล่าวแสดงได้ ดังในรูปที่ 3.5




**รูปที่ 3.6** แผนผังการทำงานของคำสั่ง Save

**3) คำสั่ง Save As** จะมีขั้นตอนในการทำงานเหมือนกับการบันทึกโดยใช้คำสั่ง Save ซึ่งขั้นตอนในการทำงานจะเรียกหน้าต่าง Save As ขึ้นมา แล้วจะตรวจสอบว่าผู้ใช้ทำการยกเลิกการบันทึกหรือว่าชื่อไฟล์ที่ใส่นั้นว่าง หรือไม่ได้ใส่ชื่อที่ต้องการบันทึก จะไม่ทำการบันทึกให้ หากตรวจสอบ

ว่าไม่ได้ยกเลิก หรือ ใส่ชื่อไฟล์ที่ต้องการบันทึกแล้วก็จะทำการบันทึกข้อมูล แล้วกำหนดค่าให้ตัวแปร Dutyflag มีค่าเป็น False ซึ่งขั้นตอนการทำงานดังแสดงในรูปที่ 3.7




**รูปที่ 3.7** แผนผังการทำงานของคำสั่ง Save As

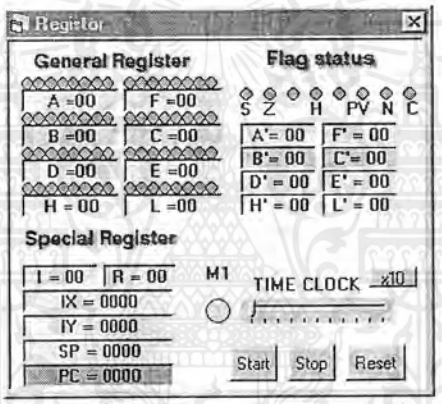
**4) คำสั่ง Assembler** เรียกส่วนของการทำการแอสเซมเบลอร์ ซึ่งอยู่ในส่วนของการจำลองชุดคำสั่ง โดยคำสั่งนี้จะกระทำได้เมื่อมีการสร้างไฟล์เท็กซ์นามสกุล Asm ที่เป็นโค้ดของการเขียนโปรแกรมไมโครโปรเซสเซอร์ Z-80 เรียบร้อยแล้ว ซึ่งถ้าหากว่าไม่มีการสร้างไฟล์นามสกุล Asm ก็จะไม่สามารถที่จะทำการแอสเซมเบลอร์ได้ ในคำสั่งนี้จะออกแบบให้มีการทำงานโดยไปเรียกโพซีเยอร์ของการแอสเซมเบลอร์ แล้วส่งการทำงานต่อไปให้ยังส่วนของการจำลองคำสั่ง

**5) คำสั่ง Start** เป็นคำสั่ง ที่จะไปกระทำ การจำลองการทำงานของ ไมโครโปรเซสเซอร์ Z-80 โดยที่คำสั่งนี้จะไปเรียกส่วนของการจำลองชุดคำสั่งให้เริ่มทำงาน ซึ่งคำสั่งนี้จะใช้ได้ก็ต่อ เมื่อมีการผ่านการการแอสเซมเบลอร์ เรียบร้อยแล้วจึงจะทำการใช้ได้ หากไม่ได้การแอสเซมเบลอร์ ก็

จะไม่สามารถเลือกใช้ได้ เนื่องจากได้ออกแบบให้ไม่แอกทีฟ คำสั่งนี้จะเรียก โพซีเยอร์ย่อยของการรันซึ่งอยู่ในส่วนของการจำลองชุดคำสั่ง แล้วส่งต่อการทำงานไปให้ยังส่วนการจำลองคำสั่ง

### 3.3.2 หน้าต่างรีจิสเตอร์

หน้าต่างของรีจิสเตอร์จะเป็นการจำลองรีจิสเตอร์ที่มีอยู่ในไมโครโปรเซสเซอร์ Z-80 หน้าต่างนี้จะแสดงเมื่อมีการเลือกที่เมนูหลัก View แล้วเลือกที่คำสั่ง Register Window ก็จะแสดงหน้าต่างนี้ และถ้าหากไม่ได้เลือกตามขั้นตอนดังกล่าวขณะที่ทำการรันโปรแกรม หน้าต่างรีจิสเตอร์จะแสดงขึ้นมา โดยหน้าต่างนี้มีหน้าที่ในการแสดงผลของการทำคำสั่งที่มีผลรีจิสเตอร์ทุกคำสั่ง ซึ่งจะแสดงค่าที่เปลี่ยนแปลงเพื่อที่จะทำให้สังเกตการเปลี่ยนแปลงได้ หน้าต่างของรีจิสเตอร์ดังแสดงในรูปที่ 3.8

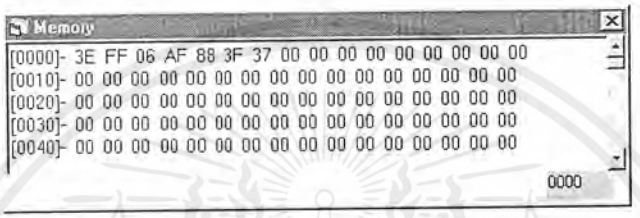



**รูปที่ 3.8** หน้าต่างรีจิสเตอร์

จากรูปที่ 3.8 หน้าต่างรีจิสเตอร์ประกอบด้วยส่วนของรีจิสเตอร์หลัก (General register) ซึ่งประกอบด้วยรีจิสเตอร์ A, B, C, D, E, F, H และ L รวมทั้งในส่วนของรีจิสเตอร์สำรองที่มีชื่อเหมือนกับรีจิสเตอร์หลัก แต่จะไม่สามารถที่จะกระทำการทางคณิตศาสตร์ และลอจิกได้ ซึ่งลักษณะของการวางนั้นจะวางให้เป็นแบบคู่กัน เพื่อเป็นการสร้างความเข้าใจถึงการจับคู่ของรีจิสเตอร์ที่ใช้ในกรณีที่ใช้งานรีจิสเตอร์แบบคู่ตามที่ได้กล่าวไว้ในทฤษฎีในบทที่ 2 ส่วนของรีจิสเตอร์เฉพาะ (Special register) ประกอบด้วยรีจิสเตอร์ขนาด 8 บิต 2 รีจิสเตอร์ คือ รีจิสเตอร์ R และ I รีจิสเตอร์ขนาด 16 บิต คือ รีจิสเตอร์ IX, IY, SPและ PC และในส่วนของสถานะของแฟลกที่เป็นไฟบอกสถานะต่าง ๆ นอกจากนั้นในส่วนของหน้าต่างรีจิสเตอร์จะมีส่วนของ Time Clock ที่เป็นการกำหนดค่าของเวลาที่ใช้ในการรันคำสั่ง ให้มีการรันเป็นทีละบรรทัด หรือ รันแบบต่อเนื่อง ได้ และมีปุ่มสต็อป และ ปุ่มรีเซต เพื่อหยุด การรัน และล้างข้อมูลในรีจิสเตอร์

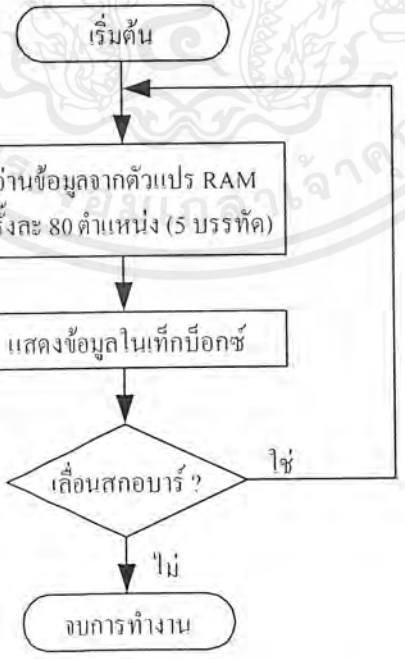
### 3.3.3 หน้าต่างหน่วยความจำ

หน้าต่างหน่วยความจำจะเป็นการจำลองหน่วยความจำให้กับไมโครโปรเซสเซอร์ Z-80 หน้าต่างนี้จะแสดงเมื่อมีการเลือกที่เมนูหลัก View แล้วเลือกที่ Memory Window จะแสดงหน้าต่างนี้ ถ้าหากไม่ได้เลือกตามขั้นตอนดังกล่าว ขณะที่รันโปรแกรมหน้าต่างหน่วยความจำจะแสดงขึ้นมา ตำแหน่งของหน่วยความจำมีตั้งแต่ 0000H - FFFFH สามารถที่จะเลือกดูข้อมูลในหน่วยความจำได้โดยการเลื่อนสกอร์บาร์ หน้าต่างของหน่วยความจำสามารถแสดงได้ดังในรูปที่ 3.9




**รูปที่ 3.9** หน้าต่างหน่วยความจำ

หน้าต่างหน่วยความจำทำงาน โดยการนำเอาข้อมูลจากตัวแปรหน่วยความจำ แล้วทำการอ่านข้อมูลเขียนลงในเท็กบ็อกซ์ 5 บรรทัด เมื่อมีการเลื่อนสกอร์บาร์จะไปการอ่านค่าในตัวแปรหน่วยความจำ 5 บรรทัด ซึ่งขั้นตอนในการทำงานดังแสดงในรูปที่ 3.10




**รูปที่ 3.10** แผนผังการแสดงผลของหน้าต่างหน่วยความจำ

### 3.3.4 หน้าต่างอิดิเตอร์

หน้าต่างอิดิเตอร์จะเป็นหน้าต่างที่ใช้สำหรับให้ผู้ใช้ทำการสร้างไฟล์ใหม่ หรือ ทำการเปิดไฟล์ที่เขียนไว้แล้วมาทำการแก้ไขใหม่อีกครั้ง ในหน้าต่างนี้จะประกอบไปด้วยเท็กบอกซ์เป็นตัวที่ใช้ในการเขียน และรับอ่านจากการอ่านไฟล์มาแสดงให้ผู้ใช้ได้ การทำงานของน้าต่างนี้จะเหมือนกับอิดิเตอร์ทั่วไป หน้าต่างอิดิเตอร์ดังแสดงในรูปที่ 3.11


**รูปที่ 3.11** หน้าต่างอิดิเตอร์


**รูปที่ 3.12** แผนผังการทำงานของหน้าต่างอิดิเตอร์

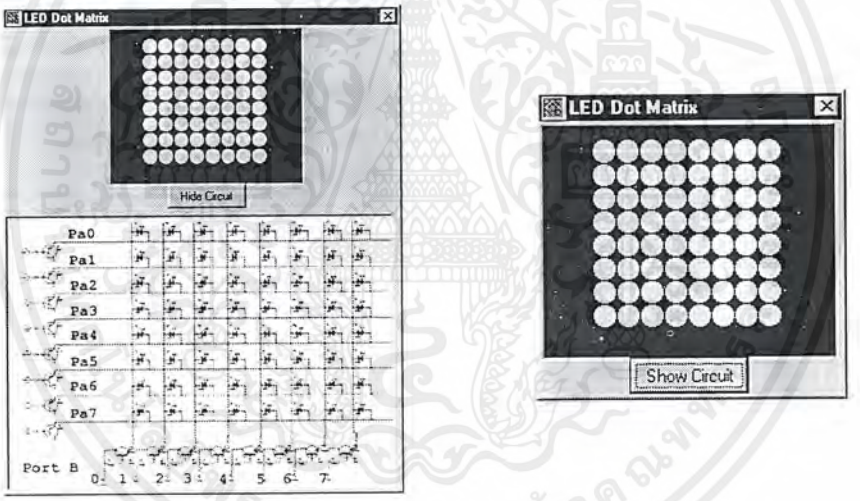
จากรูปที่ 3.12 ขั้นตอนการทำงานของหน้าต่างอิดิเตอร์จะรอรับการกดคีย์บอร์ด และจากการไฟล์เข้ามา หากมีการกดคีย์จะกำหนดให้ Dutyflag มีค่าเป็น True แล้วจึงทำการแสดงค่าที่รับมาในเท็กบ็อกซ์ แล้วจึงไปรอรับการกดคีย์อีกครั้ง

## 3.4 ภาคแสดงผล

ชุดอินเตอร์เฟสจำลองเป็นชุดที่จำลองการทำงานเสมือนกับการทำงานบนวงจรจริง ดังนั้น ส่วนที่ชุดอินเตอร์เฟสจำลองจำเป็นต้องใช้ คือ ฟังก์ชั่นจำลองการทำงานของ 8255 ซึ่งในการสร้างชุดอินเตอร์เฟสจะแบ่งชุดจำลองออกเป็น 3 ชุด คือ ภาคแสดงผลแอลอีดี, ภาคแสดงผลแบบแอลซีดี, ภาคแสดงผลแบบเจ็ดส่วน และชุดการจำลองการทำงานของมอเตอร์

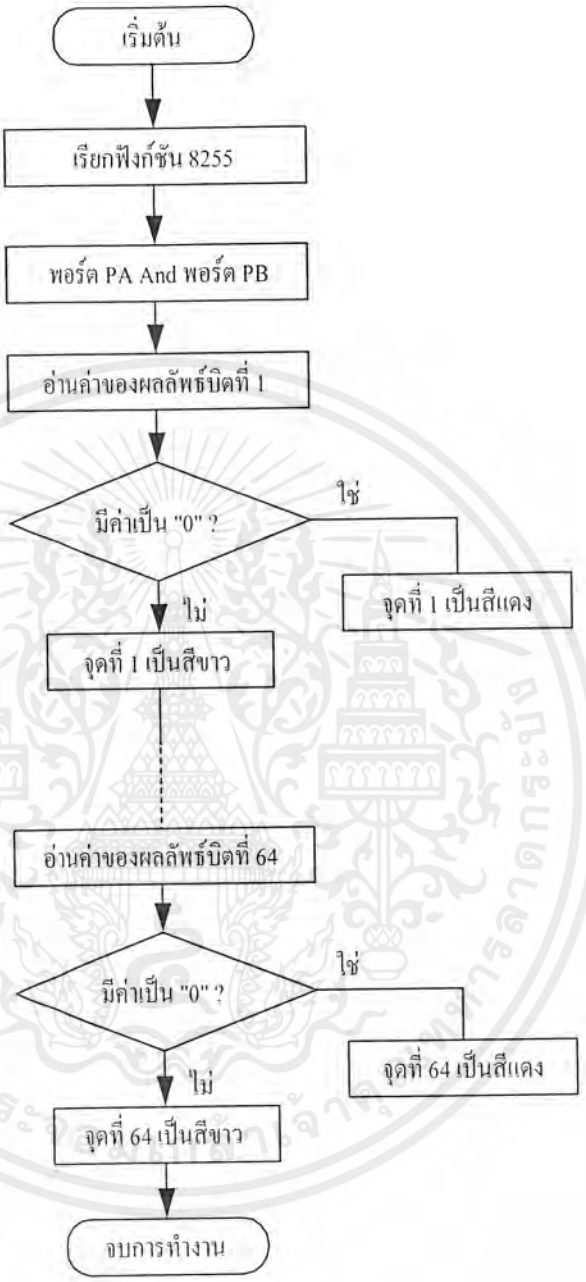
### 3.4.1 ภาคแสดงผลแอลอีดี

ในภาคแสดงผลแอลอีดีจะจำลองการทำงานการแสดงแสดงผลแอลอีดีเมตริกซ์ขนาด 8 x 8 โดยการจำลองการทำงานจะต้องใช้ ฟังก์ชันจำลองการทำงานของ 8255 ในส่วนของตัวแอลอีดีจำลองใช้คอมโปเนนต์รูปร่างกำหนดให้มีลักษณะเป็นรูปวงกลม ในการแสดงผลจะใช้การกำหนดรูปร่างให้เปลี่ยนสีโดยใช้คุณสมบัติ Shape.Fillcolor ภาคแสดงผลแอลอีดีดังแสดงในรูปที่ 3.13




**รูปที่ 3.13** 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 ดังนั้นในการตรวจสอบเงื่อนไขของ PA กับ PB จะต้องมีการตรวจสอบทั้งหมด 64 ครั้ง ซึ่งเท่ากับจำนวนของแอลอีดีมีทั้งหมด 64 ตัว แสดงการทำงานดังในรูปที่ 3.14 และโปรแกรมการทำงานได้ดังรูปที่ 3.15 ตามลำดับ




**รูปที่ 3.14** แผนผังการทำงานของภาคแสดงผลแอลอีดี

จากรูปที่ 3.14 การทำงานจะเรียกฟังก์ชัน 8255 ขึ้นมา เพื่อจะกำหนดพอร์ตที่ติดต่อด้วย แล้วอ่านค่าของพอร์ต PA และ พอร์ต PB เพื่อนำมาทำการแอนด์กัน จะตรวจว่าแต่ละบิตมีค่าเป็น 0 หรือ 1 ถ้าหากค่าเป็น 0 ก็จะไม่แสดง และถ้าเป็น 1 แอลอีดีติดนั่นเอง ซึ่งการตรวจสอบนี้จะสแกนทำทั้งหมด 64 ครั้งเท่ากับจำนวนของแอลอีดี

```
Private Function ledmatrix (pa0, pa1, pa2, pa3, pa4, pa5, pa6, pa7, pb0, pb1, pb2, pb3, pb4, pb5, pb6, pb7 As
Integer) As Integer
        Let red = QBColor(12)
        Let white = QBColor(15)
        onn = red
        off = white
Rem columm 1    If pa7 = 1 And pb7 = 1 Then led1.FillColor = onn Else led1.FillColor = off
                If pa7 = 1 And pb6 = 1 Then led2.FillColor = onn Else led2.FillColor = off
                If pa7 = 1 And pb5 = 1 Then led3.FillColor = onn Else led3.FillColor = off
                If pa7 = 1 And pb4 = 1 Then led4.FillColor = onn Else led4.FillColor = off
                If pa7 = 1 And pb3 = 1 Then led5.FillColor = onn Else led5.FillColor = off
                If pa7 = 1 And pb2 = 1 Then led6.FillColor = onn Else led6.FillColor = off
                If pa7 = 1 And pb1 = 1 Then led7.FillColor = onn Else led7.FillColor = off
                If pa7 = 1 And pb0 = 1 Then led8.FillColor = onn Else led8.FillColor = off
Rem columm 2    If pa6 = 1 And pb7 = 1 Then led9.FillColor = onn Else led9.FillColor = off
                If pa6 = 1 And pb6 = 1 Then led10.FillColor = onn Else led10.FillColor = off
                If pa6 = 1 And pb5 = 1 Then led11.FillColor = onn Else led11.FillColor = off
                If pa6 = 1 And pb4 = 1 Then led12.FillColor = onn Else led12.FillColor = off
                If pa6 = 1 And pb3 = 1 Then led13.FillColor = onn Else led13.FillColor = off
                If pa6 = 1 And pb2 = 1 Then led14.FillColor = onn Else led14.FillColor = off
                If pa6 = 1 And pb1 = 1 Then led15.FillColor = onn Else led15.FillColor = off
                If pa6 = 1 And pb0 = 1 Then led16.FillColor = onn Else led16.FillColor = off
Rem columm 3    If pa5 = 1 And pb7 = 1 Then led17.FillColor = onn Else led17.FillColor = off
                If pa5 = 1 And pb6 = 1 Then led18.FillColor = onn Else led18.FillColor = off
                If pa5 = 1 And pb5 = 1 Then led19.FillColor = onn Else led19.FillColor = off
                If pa5 = 1 And pb4 = 1 Then led20.FillColor = onn Else led20.FillColor = off
                If pa5 = 1 And pb3 = 1 Then led21.FillColor = onn Else led21.FillColor = off
                If pa5 = 1 And pb2 = 1 Then led22.FillColor = onn Else led22.FillColor = off
                If pa5 = 1 And pb1 = 1 Then led23.FillColor = onn Else led23.FillColor = off
                If pa5 = 1 And pb0 = 1 Then led24.FillColor = onn Else led24.FillColor = off
Rem columm 4    If pa4 = 1 And pb7 = 1 Then led25.FillColor = onn Else led25.FillColor = off
                If pa4 = 1 And pb6 = 1 Then led26.FillColor = onn Else led26.FillColor = off
                If pa4 = 1 And pb5 = 1 Then led27.FillColor = onn Else led27.FillColor = off
                If pa4 = 1 And pb4 = 1 Then led28.FillColor = onn Else led28.FillColor = off
                If pa4 = 1 And pb3 = 1 Then led29.FillColor = onn Else led29.FillColor = off
                If pa4 = 1 And pb2 = 1 Then led30.FillColor = onn Else led30.FillColor = off
                If pa4 = 1 And pb1 = 1 Then led31.FillColor = onn Else led31.FillColor = off
                If pa4 = 1 And pb0 = 1 Then led32.FillColor = onn Else led32.FillColor = off
```

รูปที่ 3.15 การเขียนฟังก์ชันการทำงานของภาคแสดงผลแอลอีดี

```
Rem column 5    If pa3 = 1 And pb7 = 1 Then led33.FillColor = onn Else led33.FillColor = off
                If pa3 = 1 And pb6 = 1 Then led34.FillColor = onn Else led34.FillColor = off
                If pa3 = 1 And pb5 = 1 Then led35.FillColor = onn Else led35.FillColor = off
                If pa3 = 1 And pb4 = 1 Then led36.FillColor = onn Else led36.FillColor = off
                If pa3 = 1 And pb3 = 1 Then led37.FillColor = onn Else led37.FillColor = off
                If pa3 = 1 And pb2 = 1 Then led38.FillColor = onn Else led38.FillColor = off
                If pa3 = 1 And pb1 = 1 Then led39.FillColor = onn Else led39.FillColor = off
                If pa3 = 1 And pb0 = 1 Then led40.FillColor = onn Else led40.FillColor = off
Rem columm 6    If pa2 = 1 And pb7 = 1 Then led41.FillColor = onn Else led41.FillColor = off
                If pa2 = 1 And pb6 = 1 Then led42.FillColor = onn Else led42.FillColor = off
                If pa2 = 1 And pb5 = 1 Then led43.FillColor = onn Else led43.FillColor = off
                If pa2 = 1 And pb4 = 1 Then led44.FillColor = onn Else led44.FillColor = off
                If pa2 = 1 And pb3 = 1 Then led45.FillColor = onn Else led45.FillColor = off
                If pa2 = 1 And pb2 = 1 Then led46.FillColor = onn Else led46.FillColor = off
                If pa2 = 1 And pb1 = 1 Then led47.FillColor = onn Else led47.FillColor = off
                If pa2 = 1 And pb0 = 1 Then led48.FillColor = onn Else led48.FillColor = off
Rem columm 7    If pa1 = 1 And pb7 = 1 Then led49.FillColor = onn Else led49.FillColor = off
                If pa1 = 1 And pb6 = 1 Then led50.FillColor = onn Else led50.FillColor = off
                If pa1 = 1 And pb5 = 1 Then led51.FillColor = onn Else led51.FillColor = off
                If pa1 = 1 And pb4 = 1 Then led52.FillColor = onn Else led52.FillColor = off
                If pa1 = 1 And pb3 = 1 Then led53.FillColor = onn Else led53.FillColor = off
                If pa1 = 1 And pb2 = 1 Then led54.FillColor = onn Else led54.FillColor = off
                If pa1 = 1 And pb1 = 1 Then led55.FillColor = onn Else led55.FillColor = off
                If pa1 = 1 And pb0 = 1 Then led56.FillColor = onn Else led56.FillColor = off
Rem columm 8    If pa0 = 1 And pb7 = 1 Then led57.FillColor = onn Else led57.FillColor = off
                If pa0 = 1 And pb6 = 1 Then led58.FillColor = onn Else led58.FillColor = off
                If pa0 = 1 And pb5 = 1 Then led59.FillColor = onn Else led59.FillColor = off
                If pa0 = 1 And pb4 = 1 Then led60.FillColor = onn Else led60.FillColor = off
                If pa0 = 1 And pb3 = 1 Then led61.FillColor = onn Else led61.FillColor = off
                If pa0 = 1 And pb2 = 1 Then led62.FillColor = onn Else led62.FillColor = off
                If pa0 = 1 And pb1 = 1 Then led63.FillColor = onn Else led63.FillColor = off
                If pa0 = 1 And pb0 = 1 Then led64.FillColor = onn Else led64.FillColor = off

End Function
```

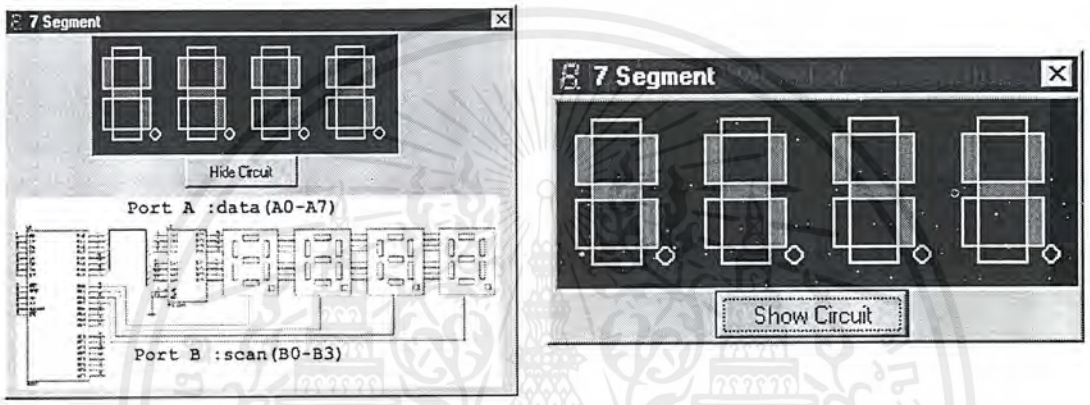
**รูปที่ 3.15 (ต่อ)** การเขียนฟังก์ชันการทำงานของภาคแสดงผลแอลอีดี

```
Rem columm 5    If pa3 = 1 And pb7 = 1 Then led33.FillColor = onn Else led33.FillColor = off
                If pa3 = 1 And pb6 = 1 Then led34.FillColor = onn Else led34.FillColor = off
                If pa3 = 1 And pb5 = 1 Then led35.FillColor = onn Else led35.FillColor = off
                If pa3 = 1 And pb4 = 1 Then led36.FillColor = onn Else led36.FillColor = off
                If pa3 = 1 And pb3 = 1 Then led37.FillColor = onn Else led37.FillColor = off
                If pa3 = 1 And pb2 = 1 Then led38.FillColor = onn Else led38.FillColor = off
                If pa3 = 1 And pb1 = 1 Then led39.FillColor = onn Else led39.FillColor = off
                If pa3 = 1 And pb0 = 1 Then led40.FillColor = onn Else led40.FillColor = off
Rem columm 6    If pa2 = 1 And pb7 = 1 Then led41.FillColor = onn Else led41.FillColor = off
                If pa2 = 1 And pb6 = 1 Then led42.FillColor = onn Else led42.FillColor = off
                If pa2 = 1 And pb5 = 1 Then led43.FillColor = onn Else led43.FillColor = off
                If pa2 = 1 And pb4 = 1 Then led44.FillColor = onn Else led44.FillColor = off
                If pa2 = 1 And pb3 = 1 Then led45.FillColor = onn Else led45.FillColor = off
                If pa2 = 1 And pb2 = 1 Then led46.FillColor = onn Else led46.FillColor = off
                If pa2 = 1 And pb1 = 1 Then led47.FillColor = onn Else led47.FillColor = off
                If pa2 = 1 And pb0 = 1 Then led48.FillColor = onn Else led48.FillColor = off
Rem columm 7    If pa1 = 1 And pb7 = 1 Then led49.FillColor = onn Else led49.FillColor = off
                If pa1 = 1 And pb6 = 1 Then led50.FillColor = onn Else led50.FillColor = off
                If pa1 = 1 And pb5 = 1 Then led51.FillColor = onn Else led51.FillColor = off
                If pa1 = 1 And pb4 = 1 Then led52.FillColor = onn Else led52.FillColor = off
                If pa1 = 1 And pb3 = 1 Then led53.FillColor = onn Else led53.FillColor = off
                If pa1 = 1 And pb2 = 1 Then led54.FillColor = onn Else led54.FillColor = off
                If pa1 = 1 And pb1 = 1 Then led55.FillColor = onn Else led55.FillColor = off
                If pa1 = 1 And pb0 = 1 Then led56.FillColor = onn Else led56.FillColor = off
Rem columm 8    If pa0 = 1 And pb7 = 1 Then led57.FillColor = onn Else led57.FillColor = off
                If pa0 = 1 And pb6 = 1 Then led58.FillColor = onn Else led58.FillColor = off
                If pa0 = 1 And pb5 = 1 Then led59.FillColor = onn Else led59.FillColor = off
                If pa0 = 1 And pb4 = 1 Then led60.FillColor = onn Else led60.FillColor = off
                If pa0 = 1 And pb3 = 1 Then led61.FillColor = onn Else led61.FillColor = off
                If pa0 = 1 And pb2 = 1 Then led62.FillColor = onn Else led62.FillColor = off
                If pa0 = 1 And pb1 = 1 Then led63.FillColor = onn Else led63.FillColor = off
                If pa0 = 1 And pb0 = 1 Then led64.FillColor = onn Else led64.FillColor = off

End Function
```

รูปที่ 3.15 (ต่อ) การเขียนฟังก์ชันการทำงานของภาคแสดงผลแอลอีดี

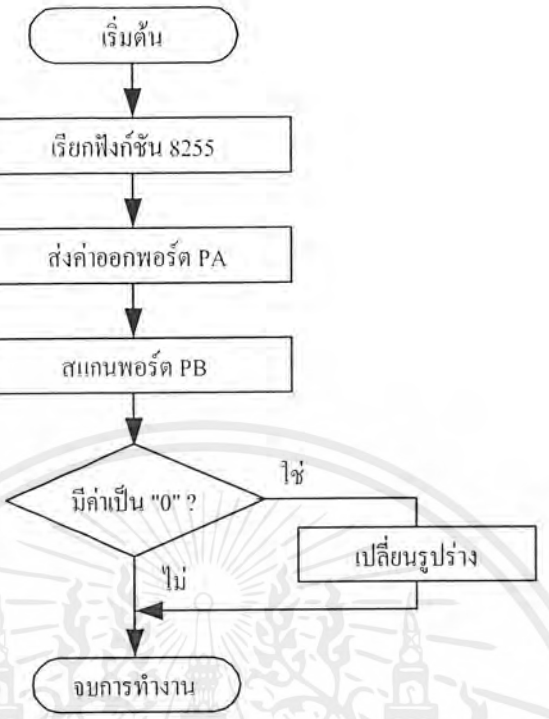
### 3.4.2 ภาคแสดงผลแบบเจ็ดส่วน

การทำงานของภาคแสดงผลแบบเจ็ดส่วนนี้ ต้องใช้ฟังก์ชันของ 8255 เหมือนกับภาคแสดงผลแบบแอลอีดี การใช้งานส่วนนี้จะต้องที่การส่งค่าผ่านฟังก์ชันของ 8255 ซึ่งภาคแสดงผลนี้ประกอบด้วยแบบเจ็ดส่วนทั้งหมด 4 ตัว โดยการออกแบบภาคแสดงผลแบบเจ็ดส่วนจำลอง คือ การทำรูปร่างเป็นรูปของอุปกรณ์อิเล็กทรอนิกส์แบบเจ็ดส่วนแอดโนดต่อร่วม ดังที่ได้กล่าวไว้ในส่วนของบทที่ 2 ซึ่งภาคแสดงผลแบบเจ็ดส่วนดังแสดงในรูปที่ 3.16




**รูปที่ 3.16** ภาคแสดงผลแบบเจ็ดส่วน

การทำงานของภาคแสดงผลแบบเจ็ดส่วนนี้ ในขั้นแรกจะส่งค่าข้อมูลออกไปยังฟังก์ชันของ 8255 จำลอง จากนั้นข้อมูลจะถูกส่งออกทางพอร์ต PA ส่วนพอร์ต PB0, PB1 และ PB2 ซึ่งจะใช้ในการสแกน ซึ่งขั้นตอนในการสแกนนั้นหากตัวใดที่มีค่าเป็น 0 รูปร่างของภาคตัวแสดงผลแบบเจ็ดส่วนจะไม่เปลี่ยนแปลงสี หากบิตใดมีค่าเป็น 1 ก็จะทำการเปลี่ยนแปลงสีให้เป็นสีแดง ขั้นตอนนี้จะทำอยู่ไปจนกว่าจะครบตามจำนวนที่ตั้งไว้ ซึ่งขั้นตอนการทำงานของภาคแสดงผลแบบเจ็ดส่วนดังแสดงในรูปที่ 3.17 และโปรแกรมการทำงานเขียนได้ในดังรูปที่ 3.18




**รูปที่ 3.17** การทำงานของภาคแสดงแบบเจ็ดส่วน

```
Private Sub Form_Load()
        Let data(0) = porta0
        Let data(1) = porta1
        Let data(2) = porta2
        Let data(3) = porta3
        Let data(4) = porta4
        Let data(5) = porta5
        Let data(6) = porta6
        Let data(7) = porta7

        Let scan(0) = portb0
        Let scan(1) = portb1
        Let scan(2) = portb2
        If data(0) = 1 And scan(0) = 0 Then a1.FillColor = QBColor(12) Else a1.FillColor = QBColor(15)
        If data(1) = 1 And scan(0) = 0 Then b1.FillColor = QBColor(12) Else b1.FillColor = QBColor(15)
        If data(2) = 1 And scan(0) = 0 Then c1.FillColor = QBColor(12) Else c1.FillColor = QBColor(15)
        If data(3) = 1 And scan(0) = 0 Then d_1.FillColor = QBColor(12) Else d_1.FillColor = QBColor(15)
```

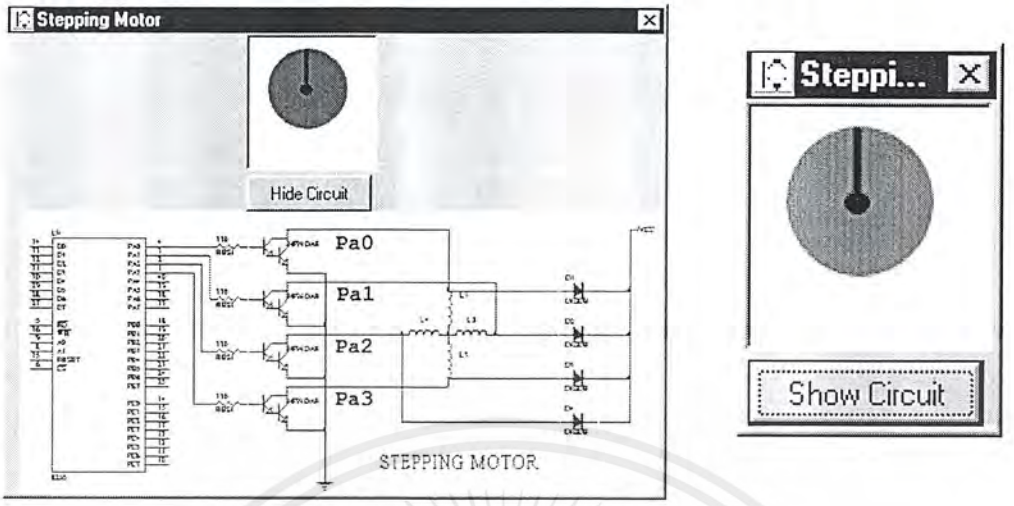
**รูปที่ 3.18** การเขียนฟังก์ชันการทำงานของภาคแสดงแบบเจ็ดส่วน

```
If data(4) = 1 And scan(0) = 0 Then e1.FillColor = QBColor(12) Else e1.FillColor = QBColor(15)
If data(5) = 1 And scan(0) = 0 Then f1.FillColor = QBColor(12) Else f1.FillColor = QBColor(15)
If data(6) = 1 And scan(0) = 0 Then g1.FillColor = QBColor(12) Else g1.FillColor = QBColor(15)
If data(7) = 1 And scan(0) = 0 Then dot1.FillColor = QBColor(12) Else dot1.FillColor = QBColor(15)

If data(0) = 1 And scan(1) = 0 Then a2.FillColor = QBColor(12) Else a2.FillColor = QBColor(15)
If data(1) = 1 And scan(1) = 0 Then b2.FillColor = QBColor(12) Else b2.FillColor = QBColor(15)
If data(2) = 1 And scan(1) = 0 Then c2.FillColor = QBColor(12) Else c2.FillColor = QBColor(15)
If data(3) = 1 And scan(1) = 0 Then d_2.FillColor = QBColor(12) Else d_2.FillColor = QBColor(15)
If data(4) = 1 And scan(1) = 0 Then e2.FillColor = QBColor(12) Else e2.FillColor = QBColor(15)
If data(5) = 1 And scan(1) = 0 Then f2.FillColor = QBColor(12) Else f2.FillColor = QBColor(15)
If data(6) = 1 And scan(1) = 0 Then g2.FillColor = QBColor(12) Else g2.FillColor = QBColor(15)
If data(7) = 1 And scan(1) = 0 Then dot2.FillColor = QBColor(12) Else dot2.FillColor = QBColor(15)

If data(0) = 1 And scan(2) = 0 Then a3.FillColor = QBColor(12) Else a3.FillColor = QBColor(15)
If data(1) = 1 And scan(2) = 0 Then b3.FillColor = QBColor(12) Else b3.FillColor = QBColor(15)
If data(2) = 1 And scan(2) = 0 Then c3.FillColor = QBColor(12) Else c3.FillColor = QBColor(15)
If data(3) = 1 And scan(2) = 0 Then D3.FillColor = QBColor(12) Else d_3.FillColor = QBColor(15)
If data(4) = 1 And scan(2) = 0 Then e3.FillColor = QBColor(12) Else e3.FillColor = QBColor(15)
If data(5) = 1 And scan(2) = 0 Then f3.FillColor = QBColor(12) Else f3.FillColor = QBColor(15)
If data(6) = 1 And scan(2) = 0 Then g3.FillColor = QBColor(12) Else g3.FillColor = QBColor(15)
If data(7) = 1 And scan(2) = 0 Then dot3.FillColor = QBColor(12) Else dot3.FillColor = QBColor(15)
End Sub
```

**รูปที่ 3.18 (ต่อ)** การเขียนฟังก์ชันการทำงานของภาคแสดงแบบเจ็ดส่วน

### 3.4.3 การจำลองการทำงานของสเต็ปปิ้งมอเตอร์

ชุดอินเตอร์เฟสจำลองสเต็ปปิ้งมอเตอร์ จะเป็นโปรแกรมจำลองการทำงานของสเต็ปปิ้งมอเตอร์ โดยการจำลองการทำงานจะใช้ฟังก์ชัน 8255 เป็นตัวติดต่อกับการแสดงผล ภาคแสดงผลแบบแอลซีดี และแบบเจ็ดส่วน การแสดงผลจะใช้คำสั่ง Line ของวิชวลเบสิก โดยทำเป็นสถานะต่าง ๆ ของมอเตอร์ไว้ ดังแสดงในตารางที่ 3.2 ส่วนของชุดอินเตอร์เฟสจำลองสเต็ปปิ้งมอเตอร์ ดังแสดงในรูปที่ 3.19

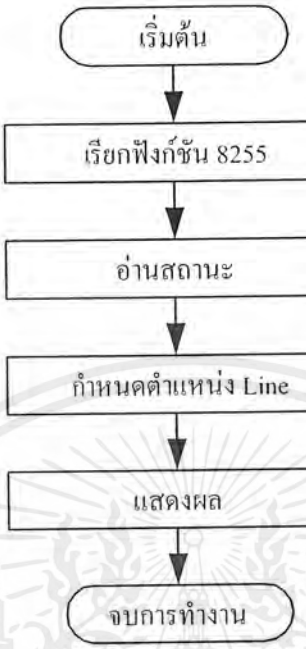



**รูปที่ 3.19** ชุดอินเตอร์เฟสจำลองสเต็ปปิ้งมอเตอร์

**ตารางที่ 3.2** มุมของโรเตอร์เทียบกับกระแสไฟฟ้าที่จ่ายแก่เฟสต่าง ๆ 8 ตำแหน่ง

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

การทำงานของชุดอินเตอร์เฟสจำลองสเต็ปปิ้งมอเตอร์ เมื่อมีการเรียกใช้โปรแกรมจะทำการรับค่าผ่านทางพอร์ต 8255 เมื่อรับค่ามาแล้วจะทำการตรวจสอบสถานะของข้อมูลที่ส่งมาว่าตรงกับสถานะใด แล้วต่อจากนั้นก็จะใช้คำสั่ง Line เพื่อกำหนดการจำลองการทำงานของสเต็ปปิ้งมอเตอร์ แสดงการขั้นตอนการทำงานดังในรูปที่ 3.20 และรูปการเขียนโปรแกรมการทำงานดังในรูปที่ 3.21




**รูปที่ 3.20** การทำงานของชุดอินเตอร์เฟสจำลองสเต็ปปิ้งมอเตอร์

```
Dim steps As Integer
Private Sub form_load()
'phase motor 8 status
        step1
        If porta0 = 1 Then step1
        If porta0 = 1 And porta1 = 1 Then step2
        If porta1 = 1 Then step3
        If porta1 = 1 And porta2 = 1 Then step4
        If porta2 = 1 Then step5
        If porta2 = 1 And porta3 = 1 Then step6
        If porta3 = 1 Then step7
        If porta2 = 1 And porta0 = 1 Then step8
End Sub
Private Function step1()
        Let off = QBColor(15)
        Image2.Picture = Image1(0)
nd Function
```

**รูปที่ 3.21** โปรแกรมการทำงานของชุดอินเตอร์เฟสจำลองสเต็ปปิ้งมอเตอร์

```
Private Function step2()
        Let off = QBColor(15)
        Image2.Picture = Image1(1)
End Function
Private Function step3()
        Let off = QBColor(15)
        Image2.Picture = Image1(2)
End Function
Private Function step4()
        Let off = QBColor(15)
        Image2.Picture = Image1(3)
End Function
Private Function step5()
        Let off = QBColor(15)
        Image2.Picture = Image1(4)
End Function
Private Function step6()
        Image2.Picture = Image1(5)
End Function
Private Function step7()
        Let off = QBColor(15)
        Image2.Picture = Image1(6)
End Function
Private Function step8()
        Let off = QBColor(15)
        Image2.Picture = Image1(7)
End Function
Private Sub Command2_Click()
     If steps = 1 Then step1
     If steps = 2 Then step2
     If steps = 3 Then step3
     If steps = 4 Then step4
     If steps = 5 Then step5
     If steps = 6 Then step6
     If steps = 7 Then step7
     If steps = 8 Then step8
     If steps > 8 Then step1  steps = steps + 1 If steps > 8 Then steps = 1
End Sub
```

**รูปที่ 3.21 (ต่อ)** โปรแกรมการทำงานของชุดอินเตอร์เฟสจำลองสเต็ปปิ้งมอเตอร์

## 3.5 การจำลองชุดคำสั่ง

### 3.5.1 การออกแบบหน่วยความจำ และรีจิสเตอร์จำลอง

ในการออกแบบหน่วยความจำและรีจิสเตอร์จำลอง มีการกำหนดตัวแปรต่าง ๆ ที่ใช้ในการจำลอง ดังต่อไปนี้

1) Public CARRYFLAG As Boolean เป็นตัวแปรแฟลกตัวทด เป็นตัวแปรชนิดแบบบูลีน

2) Public NEGATIVEFLAG As Boolean เป็นตัวแปรแฟลกลบ เป็นตัวแปรชนิดแบบบูลีน

3) Public PARITYOVERFLOW As Boolean เป็นตัวแปรแฟลกพาริตี้ หรือโอเวอร์โฟล์ เป็นตัวแปรชนิดแบบบูลีน

4) Public HALFCARRYFLAG As Boolean เป็นตัวแปรแฟลกช่วยทด ตัวแปรชนิดบูลีน

5) Public ZEROFLAG As Boolean เป็นตัวแปรแฟลกศูนย์ เป็นตัวแปรชนิดแบบบูลีน

6) Public SIGNFLAG As Boolean เป็นตัวแปรแฟลกเครื่องหมาย เป็นตัวแปรชนิดบูลีน

7) Public Flag As Byte เป็นตัวแปรกำหนดค่าของแฟลกทั้งหมด เป็นตัวแปรชนิดไบต์

8) Public CLOCK As Long เป็นตัวแปรกำหนดค่าของเวลาในการรัน หัวตัวไทเมอร์

9) Public GeneralRegis (17) As Long ตัวแปรรีจิสเตอร์ขนาด 8 บิต กำหนดให้มีทั้งหมด 18 รีจิสเตอร์ โดย แต่ละค่าของอินเด็กซ์จะหมายถึงรีจิสเตอร์ ดังแสดงในตารางที่ 3.3

**ตารางที่ 3.3** หมายเลขอินเด็กซ์ของตัวแปร GeneralRegis

| อินเด็กซ์ | รีจิสเตอร์ |
|---|---|
| 0 | A |
| 1 | B |
| 2 | C |
| 3 | D |
| 4 | E |
| 5 | F |
| 6 | H |
| 7 | L |
| 8 | I |
| 9 | R |

**ตารางที่ 3.3 (ต่อ)** หมายเลขอินเด็กซ์ของตัวแปร GeneralRegis

| อินเด็กซ์ | รีจิสเตอร์ |
|---|---|
| 10 | A' |
| 11 | B' |
| 12 | C' |
| 13 | D' |
| 14 | E' |
| 15 | F' |
| 16 | H' |
| 17 | L' |

10) Public SpecialRegis (3) As Long ตัวแปรรีจิสเตอร์เฉพาะขนาด 16 บิต ซึ่งตัวแปรในวิชวลเบสิกนั้นจะตรงกับตัวแปรชนิด Long ซึ่งแต่ละค่าของอินเด็กซ์จะหมายถึงรีจิสเตอร์ ดังแสดงในตารางที่ 3.4

**ตารางที่ 3.4** หมายเลขอินเด็กซ์ของตัวแปร SpecialRegis

| อินเด็กซ์ | รีจิสเตอร์ |
|---|---|
| 0 | PC |
| 1 | SP |
| 2 | IX |
| 3 | IY |

11) Public RAM(65535) As Byte ตัวแปรหน่วยความจำจำลองมีขนาด 64 กิโลไบต์ ซึ่งมีค่าเท่ากับ 65,536 ค่า แต่ละอินเด็กซ์บอกตำแหน่งหน่วยความจำ ซึ่งตัวไมโครโปรเซสเซอร์ Z-80 หน่วยความจำแต่ละตำแหน่งจะมีขนาดเท่ากับ 8 บิต ซึ่งในวิชวลเบสิกนั้นจะต้องประกาศตัวแบบเป็นชนิดไบต์

12) Public AF As Long ตัวแปรรีจิสเตอร์คู่ AF ขนาด 16 บิต ซึ่งตัวแปรในวิชวลเบสิกนั้นจะตรงกับตัวแปรชนิด Long

13) Public BC As Long ตัวแปรรีจิสเตอร์คู่ BC ขนาด 16 บิต ซึ่งตัวแปรในวิชวลเบสิกนั้นจะตรงกับตัวแปรชนิด Long

14) Public DE As Long ตัวแปรรีจิสเตอร์คู่ BE ขนาด 16 บิต ซึ่งตัวแปรในวิชวลเบสิกนั้นจะตรงกับตัวแปรชนิด Long

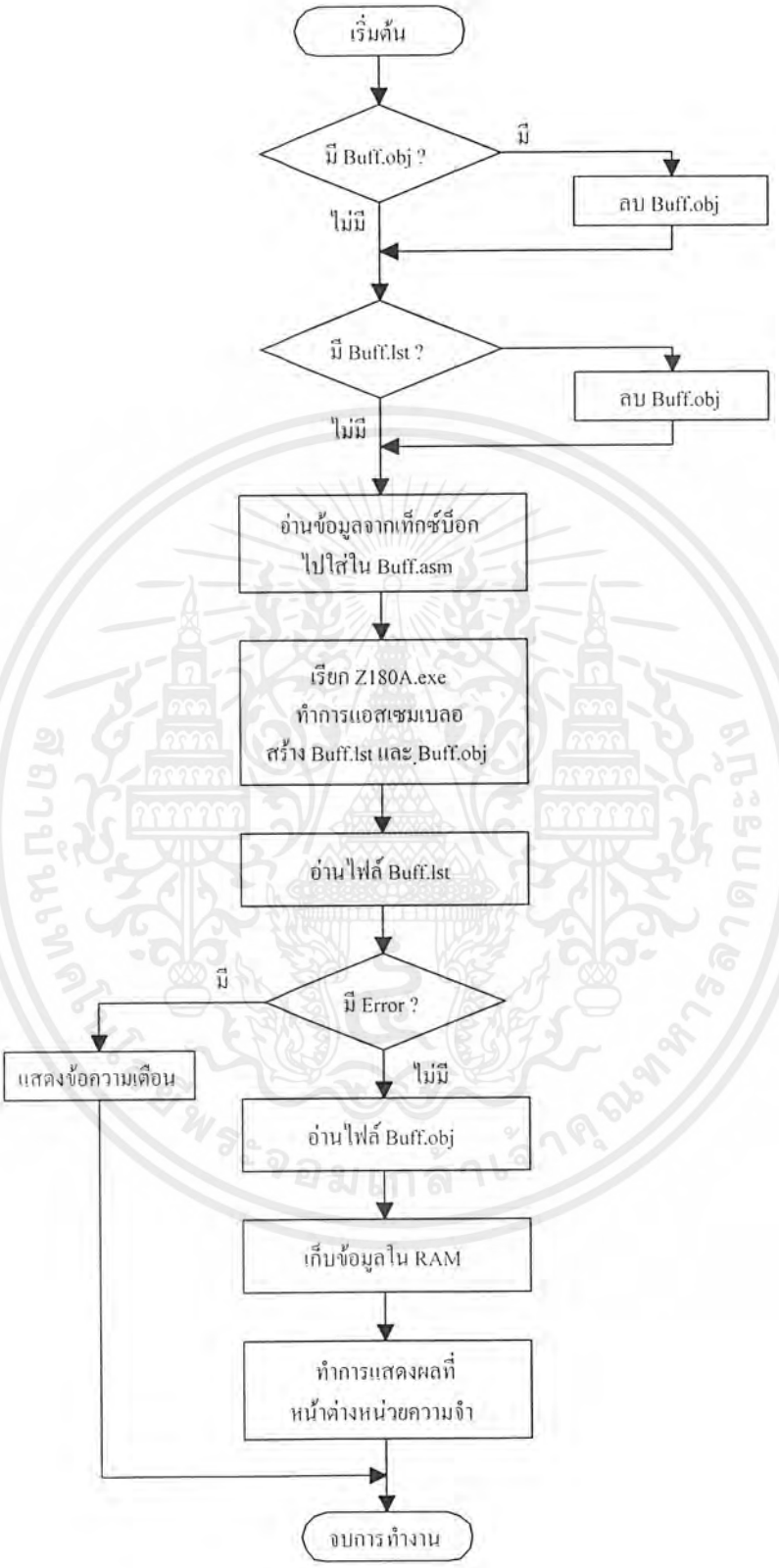
15) Public HL As Long ตัวแปรรีจิสเตอร์คู่ HL ขนาด 16 บิต ซึ่งตัวแปรในวิชวลเบสิกนั้นจะตรงกับตัวแปรชนิด Long

### 3.5.2 การจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80

การจำลองการทำงานของไมโครโปรเซสเซอร์นี้จะทำโดยการใช้โปรแกรมวิชวลเบสิกเขียนโปรแกรมควบคุมการทำงาน โดยออกแบบการทำงานนั้นจะแบ่งออกเป็น 2 ส่วน คือ ส่วนของการแอสเซมเบลอ และส่วนของการจำลองการทำงาน ซึ่งทั้งสองส่วนมีการทำงานดังนี้

**1) ส่วนของการแอสเซมเบลอ** คือส่วนของการแปลง ตัวโปรแกรมแอสแซมบลี้ ของ Z-80 ที่ได้สร้างขึ้นในส่วนของอิดิเตอร์ โดยจะทำการแปลงให้เป็นโค้ดเลขฐานสิบหก

การทำงานของส่วนของการแอสเซมเบลอ จะทำการตรวจสอบก่อนว่ามีไฟล์ Buf.obj และไฟล์ Buf.lst ไฟล์ทั้งสองนี้จะได้จากการแอสเซมเบลอ ซึ่งก่อนที่หน้าที่จะทำการแอสเซมเบลออาจจะมีการทำก่อนหน้านี้แล้ว ดังนั้นจึงต้องมีการตรวจสอบ หากพบว่ามีไฟล์ทั้งสองอยู่ หรือมีไม่ว่าจะเป็นไฟล์ใดไฟล์หนึ่งก็จะทำการลบไฟล์นั้นออก จากนั้นจะทำการอ่านข้อมูลที่อยู่ในเท็กบ็อกซ์ที่อยู่ในหน้าต่างอิดิเตอร์ที่ได้ทำการเขียนไว้แล้ว จากนั้นจะทำการเรียกตัวโปรแกรมแอสเซมเบลอมาทำการแปลงโค้ด ซึ่งตัวแอสเซมเบลอที่ใช้นั้น จะใช้ตัวคอมไพเลอร์ของไมโครโปรเซสเซอร์ Z180 ซึ่งเป็นตัวที่มีสนับสนุนคำสั่งของไมโครโปรเซสเซอร์ Z-80 จึงทำให้สามารถที่จะแอสเซมเบลอได้ โดยไฟล์ที่จะได้จากแปลงโค้ด จะมีอยู่ 2 ไฟล์ คือ ไฟล์ Buf.obj และ Buf.lst แล้วจากนั้นจะทำการอ่านไฟล์ Buf.lst เพื่อตรวจสอบว่ามีข้อผิดพลาดที่ไหน หากตรวจสอบว่ามีข้อผิดพลาดก็จะแสดงข้อความเตือน เพื่อให้ไปทำการแก้ไขแล้วจึงมาทำการแอสเซมเบลออีกครั้ง หากตรวจสอบว่าไม่มีข้อผิดพลาด ก็ทำการอ่านข้อมูลจากไฟล์ Buf.obj ซึ่งเป็นโค้ดเลขฐานสิบหก แล้วนำไปเก็บในตัวแปรหน่วยความจำจำลอง คือ RAM แล้วส่งผลแสดงออกทางหน้าต่างของหน่วยความจำว่า เมื่อทำการแปลงแล้วโค้ดที่เขียนไว้ มีคำสั่งอย่างไร เป็นเลขฐานสิบหกเท่าไร โดยจะเริ่มเก็บที่ตำแหน่งหน่วยความจำ 0000H ขึ้นไป แล้วจึงแสดงข้อความขึ้นมาบอกว่าไม่มีข้อผิดพลาด โดยอาจจะทำการรันคำสั่งเลย หรือว่ายังไม่ทำการรันก็ได้ ซึ่งแผนผังขั้นตอนการทำงานดังแสดงรูปที่ 3.22

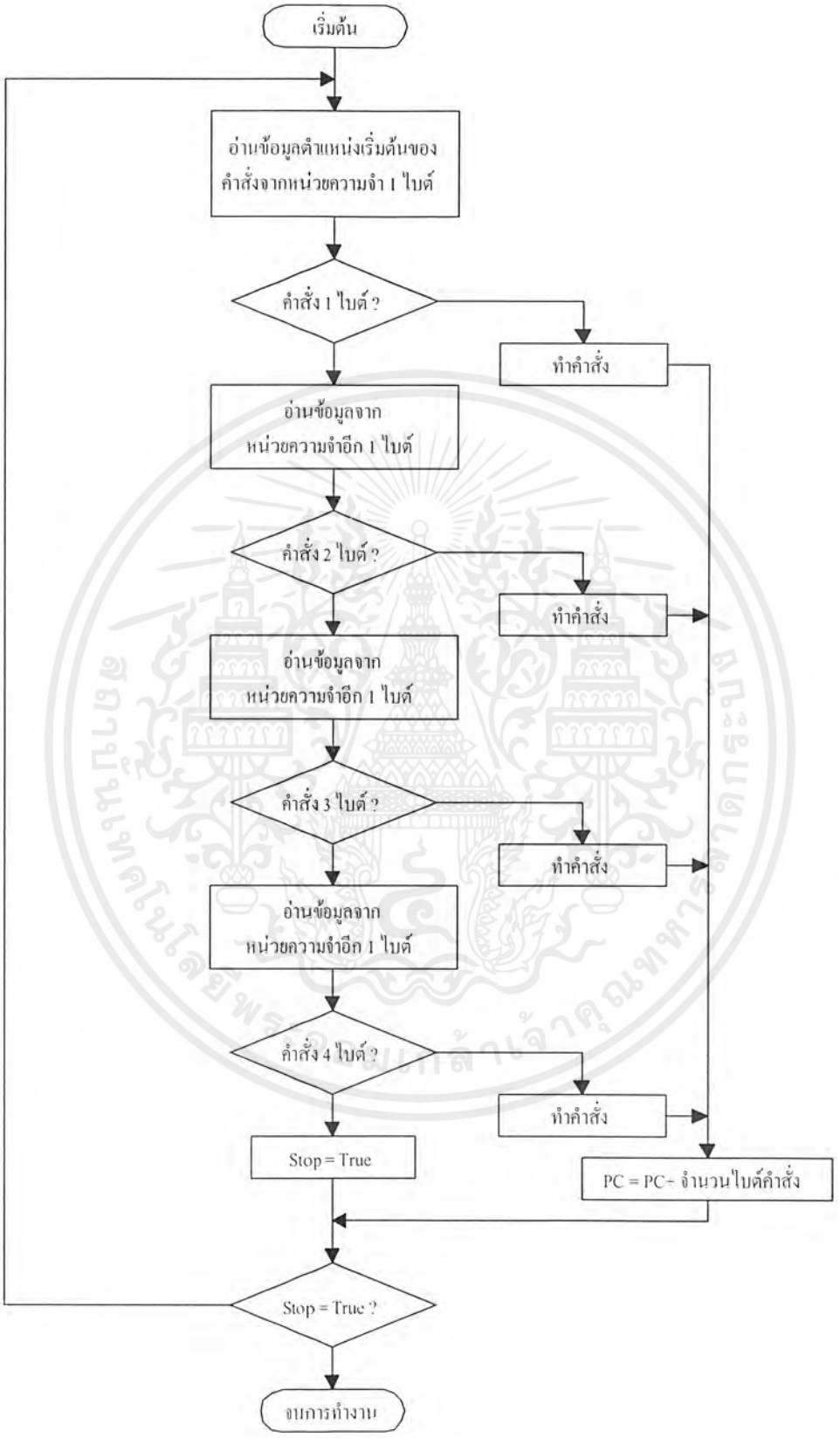



รูปที่ 3.22 แผนผังขั้นตอนการทำงานของส่วนการแอสเซมเบลอ

**2) ส่วนของการจำลองการทำงาน** คือ ส่วนของการอ่านโค้ดที่ได้จากการแอสเซมเบลอแล้วมาเป็นการกระทำตามโค้ดของทำสั่งที่สั่งให้ไมโครโปรเซสเซอร์ Z-80 ทำงาน โดยจะทำเหมือนกับการทำงานของไมโครโปรเซสเซอร์ Z-80 ซึ่งในส่วนนี้จะทำการจำลองชุดคำสั่งการทำงานทั้งหมดที่มีอยู่แล้วส่งผลออกสู่ส่วนของการแสดงผลต่าง ๆ

การทำงานของส่วนของการจำลองการทำงานจะอ่านข้อมูลที่ได้จากการแอสเซมเบลอ ซึ่งอยู่ในหน่วยความจำจำลองที่สร้างเอาไว้ โดยจะอ่านที่ตำแหน่ง 0000H ไมโครโปรเซสเซอร์ Z-80 จะเริ่มอ่านคำสั่งที่ตำแหน่ง 0000H เช่นกัน โดยในการเข้ามาจะอ่านข้อมูลขนาด 1 ไบต์ เข้ามา แล้วนำไปตรวจสอบว่า เป็นคำสั่งขนาด 1 ไบต์ ถ้าหากเป็นคำสั่งขนาด 1 ไบต์ จะทำคำสั่งนั้น หากไม่ใช่ก็จะไปอ่านข้อมูลจากหน่วยความจำมาอีก 1 ไบต์ แล้วตรวจสอบว่าเป็นคำสั่งขนาด 2 ไบต์ หรือไม่ หากเป็นคำสั่งขนาด 2 ไบต์ จะทำคำสั่งนั้น หากไม่ใช่ก็จะไปอ่านข้อมูลจากหน่วยความจำอีก 1 ไบต์ แล้วตรวจสอบว่าเป็นคำสั่งขนาด 3 ไบต์ หรือไม่ หากเป็นคำสั่งขนาด 3 ไบต์ จะทำคำสั่งนั้น หากไม่ใช่ก็จะไปอ่านข้อมูลจากหน่วยความจำอีก 1 ไบต์ แล้วตรวจสอบว่าเป็นคำสั่งขนาด 4 ไบต์ หรือไม่ หากเป็นคำสั่งขนาด 4 ไบต์ จะทำคำสั่งนั้น หากไม่ใช่จะไปสั่งให้สถานะ Stop มีค่า Ture ซึ่งจะทำให้หยุดการทำงานทันที

ในการตรวจสอบว่าเป็นคำสั่งขนาดกี่ไบต์ แล้วทำตามคำสั่งจะทำการเพิ่มค่าในโปรแกรมเคาน์เตอร์ขึ้นตามจำนวนไบต์ของคำสั่ง เมื่อเพิ่มค่าในโปรแกรมเคาน์เตอร์แล้ว จะตรวจสอบว่ามีการกำหนดสถานะของการ Stop ให้ True หรือไม่ หากพบว่ามีการกำหนดให้ Stop จะหยุดการทำงานทันที แต่ถ้าหากพบว่าไม่มีการสั่งให้ Stop จะกลับไปเริ่มอ่านข้อมูลในตำแหน่งต่อไป ซึ่งเป็นคำสั่ง แล้วทำงานตามขั้นตอนเดิมอีกครั้ง จนกระทั่งจะมีการกำหนดสถานะให้หยุดทำงาน แผนผังขั้นตอนการทำงานดังแสดงรูปที่ 3.23




**รูปที่ 3.23** แผนผังขั้นตอนการทำงานของส่วนการจำลองการทำงาน

# บทที่ 4

# การทดลองและผลการทดลอง

## 4.1 กล่าวนำ

ในบทนี้จะกล่าวถึงวิธีการทดสอบกลุ่มคำสั่งของไมโครโปรเซสเซอร์ Z-80 โดยแบ่งการทดสอบออกเป็นกลุ่ม ๆ คือ กลุ่มกลุ่มคำสั่งการโหลดและเคลื่อนย้ายข้อมูล, กลุ่มคำสั่งทางคณิตศาสตร์และลอจิก, คำสั่งการหมุนเป็นวงรอบและการเลื่อนข้อมูล, คำสั่งเกี่ยวกับการกระทำภายในข้อมูลบิต

## 4.2 กลุ่มคำสั่งการโหลดและเคลื่อนย้ายข้อมูล

กลุ่มคำสั่งการโหลดและเคลื่อนย้ายข้อมูล เป็นคำสั่งที่ไม่ผลต่อแฟลก คำสั่งการโหลดและเคลื่อนย้ายข้อมูลจะประกอบด้วย

1) คำสั่งเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับรีจิสเตอร์
2) คำสั่งเคลื่อนย้ายข้อมูลระหว่ารีจิสเตอร์กับหน่วยความจำ
3) คำสั่งเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับข้อมูลที่กำหนด
4) คำสั่งเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำกับข้อมูลที่กำหนด

### 4.2.1 การทดสอบคำสั่งเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับรีจิสเตอร์

คำสั่งการเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับรีจิสเตอร์ มีรูปแบบของคำสั่งดังแสดงในรูปที่ 4.1

LD r , s

r หมายถึง รีจิสเตอร์ต้นทาง

s หมายถึง รีจิสเตอร์ปลายทาง

**รูปที่ 4.1** รูปแบบคำสั่งเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับรีจิสเตอร์

การทดสอบ

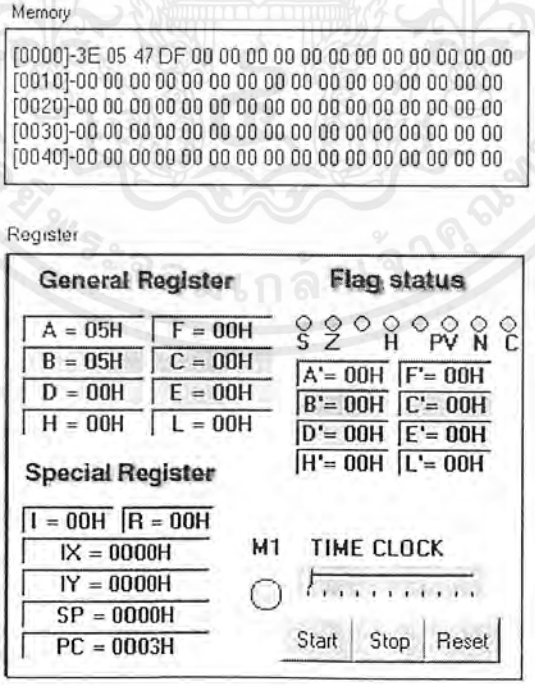
ทดสอบคำสั่งเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับรีจิสเตอร์ ด้วยการป้อนโปรแกรมดังแสดงในรูปที่ 4.2 โดยโปรแกรมจะเก็บค่า 05H ไว้ในรีจิสเตอร์ A แล้วนำค่าข้อมูลในรีจิสเตอร์ A ไปเก็บในรีจิสเตอร์ โดยที่โค้ดของโปรแกรมทั้งหมดเก็บที่หน่วยความจำตำแหน่ง 00H จนถึงตำแหน่ง 02H

```
LD      A,05H
LD      B,A
RST     18H
```

**รูปที่ 4.2** โปรแกรมทดสอบคำสั่งเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับรีจิสเตอร์

ผลการทดสอบคำสั่ง

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 03H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.3




**รูปที่ 4.3** ผลการทดสอบคำสั่งเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับรีจิสเตอร์

### 4.2.2 การทดสอบคำสั่งเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับหน่วยความจำ

การเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับหน่วยความจำจะสามารถทำได้ 2 วิธี คือ การย้ายข้อมูลจากรีจิสเตอร์ไปเก็บในหน่วยความจำ และย้ายข้อมูลจากหน่วยความจำไปเก็บในรีจิสเตอร์

**การทดสอบ**

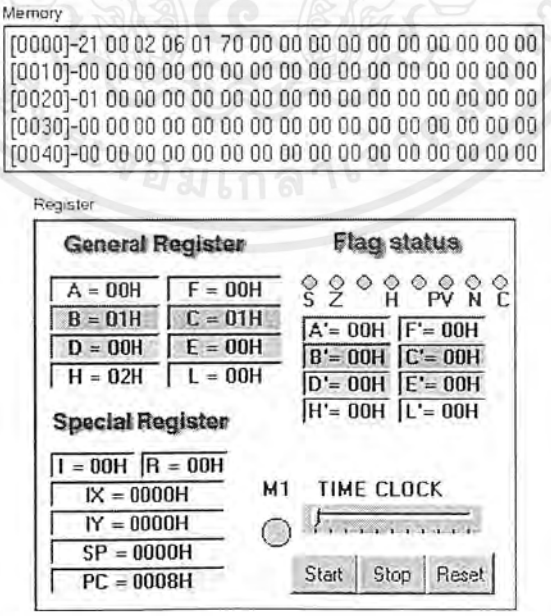
การทดสอบคำสั่งการเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับหน่วยความจำโดยการป้อนโปรแกรมดังรูปที่ 4.4 โค้ดโปรแกรมจะเก็บค่าไว้ที่หน่วยความจำตำแหน่ง 00H ถึง ตำแหน่ง 05H

```
LD      HL,20H
LD      B,01H
LD      (HL),B
LD      C,(HL)
RST     18H
```

รูปที่ 4.4 โปรแกรมทดสอบคำสั่งเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับหน่วยความจำ

**ผลการทดสอบ**

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 13H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.5




รูปที่ 4.5 ผลการทดสอบคำสั่งเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับหน่วยความจำ

### 4.2.3 การทดสอบคำสั่งเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับข้อมูลที่กำหนด

คำสั่งการเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับข้อมูลที่กำหนด วิธีนี้เป็นการอ้างแบบตรง คือ สามารถจะทำค่าอะไรใส่ในรีจิสเตอร์ไหนก็ได้ ทั้งแบบ 8 บิต และ 16 บิต

**การทดสอบ**

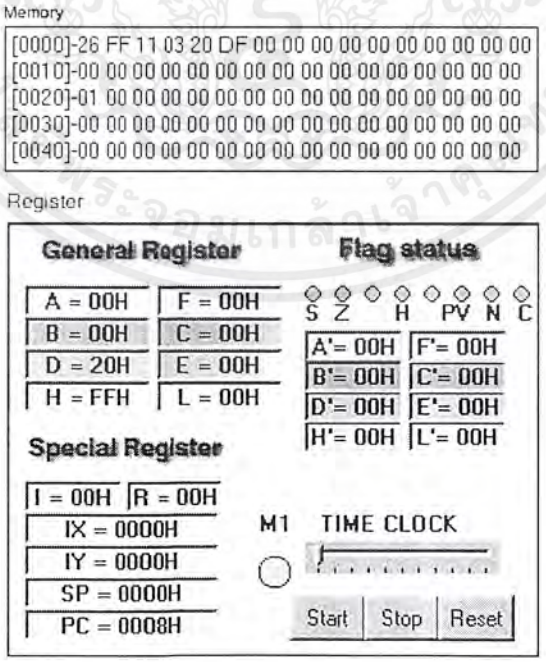
การทดสอบคำสั่งเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับรีจิสเตอร์ ด้วยการป้อนโปรแกรม ดังรูปที่ 4.6 โดยโค้ดโปรแกรมจะเก็บไว้ที่ตำแหน่งหน่วยความจำ 00H ถึง 02H

```
LD      H, 0FFH
LD      DE, 2000H
RST     18H
```

**รูปที่ 4.6** โปรแกรมทดสอบคำสั่งเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับข้อมูลที่กำหนด

**ผลการทดสอบคำสั่ง**

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 05H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.7




**รูปที่ 4.7** ผลการทดสอบคำสั่งเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับข้อมูลที่กำหนด

### 4.2.4 การทดสอบคำสั่งเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำกับข้อมูลที่กำหนด

คำสั่งการเคลื่อนย้ายข้อมูลระหว่างหน่วยความกับข้อมูลที่กำหนด คือ การย้ายข้อมูลไปเก็บหน่วยความจำ ซึ่งวิธีนี้คล้ายกับการเคลื่อนย้ายข้อมูลไปเก็บในรีจิสเตอร์ เพียงแต่แทนที่จะนำไปเก็บในรีจิสเตอร์จะนำไปเก็บในหน่วยความจำแทน

**การทดสอบ**

การทดสอบคำสั่งเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำกับข้อมูลที่กำหนด ด้วยการป้อนโปรแกรมดังรูปที่ 4.8 โปรแกรมจะนี้จะนำค่า 0030H ไปเก็บในรีจิสเตอร์ HL แล้วนำค่าข้อมูล FEH เก็บในตำแหน่งหน่วยความจำที่รีจิสเตอร์ HL แล้วนำค่า FFH เก็บไว้ในหน่วยความจำตำแหน่ง 0034H แล้วจะนำข้อมูล 15H ไปเก็บในตำแหน่ง 0305H และนำข้อมูล 08H เก็บในตำแหน่ง 0005H

```
LD      HL, 0030H
LD      (HL), 0FEH
LD      A, 0FFH
LD      (0034H), A
LD      IX, 0030H
LD      (IX+05H), 15H
LD      IY,0040H
LD      (IY+05H), 08H
RST     18H
```

**รูปที่ 4.8** โปรแกรมทดสอบคำสั่งเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์กับข้อมูลที่กำหนด

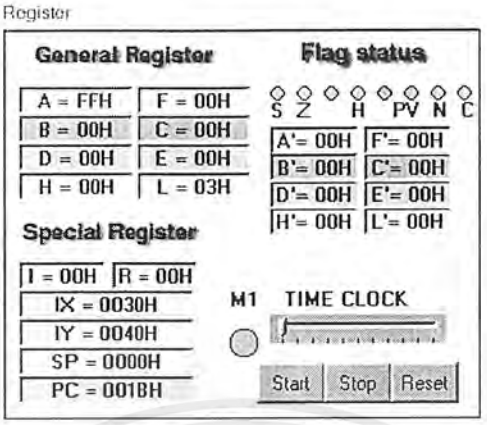
**ผลการทดสอบคำสั่ง**

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 0H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.9

Memory

```
[0000]-21 30 00 36 FF 3E FF 32 04 00 DD 21 00 30 DD 6E
[0010]-05 15 FD 21 00 40 FD 6E 05 08 00 00 00 00 00 00
[0020]-00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
[0030]-FE 00 00 00 FF 15 00 00 00 00 00 00 00 00 00 00
[0040]-00 00 00 00 00 08 00 00 00 00 00 00 00 00 00 00
```

**รูปที่ 4.9** ผลการทดสอบคำสั่งเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำกับข้อมูลที่กำหนด




รูปที่ 4.9 (ต่อ) ผลการทดสอบคำสั่งเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำกับข้อมูลที่กำหนด

## 4.2 กลุ่มคำสั่งทางคณิตศาสตร์และลอจิก

กลุ่มคำสั่งทางคณิตศาสตร์และลอจิก การทำงานมีผลต่อแฟลก ซึ่งกลุ่มคำสั่งนี้จะมีการทดสอบประกอบด้วย

1) คำสั่ง ADD

2) คำสั่ง ADC

3) คำสั่ง SUB

4) คำสั่ง SUBC

5) คำสั่งกระทำทางลอจิก

### 4.2.1 การทดสอบคำสั่ง ADD

คำสั่งบวกคำสั่งที่มีผลต่อแฟลก ซึ่งประกอบด้วย C Z P/V S N H

**การทดสอบ**

ทดสอบคำสั่งกลุ่มคำสั่ง ADD โดยการป้อนโปรแกรมตามในรูปที่ 4.10 ซึ่งโค้ดของโปรแกรมนั้นจะมีการ ADD นั้นอยู่ 5 รูปแบบ ซึ่งแต่ละรูปแบบจะมีการทำงานที่ไม่เหมือนกัน ในแบบแรกจะเป็นการบวกเลขโดยใช้รีจิสเตอร์ A กับ รีจิสเตอร์ B แบบที่ 2 จะเป็นการบวกเลขโดยใช้รีจิสเตอร์ A บวกกับข้อมูลโดยตรง แบบที่ 3 จนถึง แบบที่ 5 จะเป็นการบวกเลขโดยใช้รีจิสเตอร์ A กับตำแหน่งหน่วยความจำชี้อยู่ ซึ่งหากทำการป้อนโปรแกรมถูกต้องจะมีโค้ดโปรแกรมอยู่ ตั้งแต่ในตำแหน่งที่ 0000H ถึง

```
LD      A,05H
LD      B,01H
ADD     B
LD      A,05H
ADD     A,02H
LD      A,05H
LD      HL,10H
LD      B,02H
LD      (HL),B
ADD     A,(HL)
LD      A,05H
LD      B,02H
LD      IX,0200H
LD      (IX+01H),B
ADD     A,(IX+01H)
LD      A,05H
LD      B,02H
LD      IY,0200H
LD      (IY+01H),B
ADD     A,(IY+01H)
RST     18H
```

**รูปที่ 4.10** ผลการทดสอบคำสั่ง ADD

ผลการทดสอบ

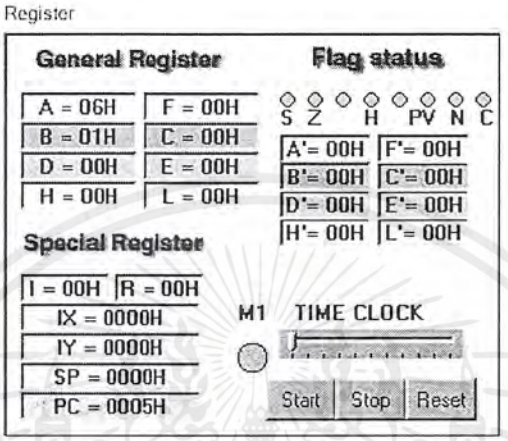
ผลการทดสอบคำสั่ง ADD เมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 30H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ ดังแสดงในรูปที่ 4.11

Memory

```
[0000]-3E 05 06 01 80 3E 05 C6 02 3E 05 21 10 00 06 02
[0010]-70 86 3E 05 02 DD 21 00 02 DD 70 01 DD 86 01 3E
[0020]-05 06 02 DD 21 00 02 DD 70 01 DD 86 01 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 08 00 00 00 00 00 00 00 00 00 00
```

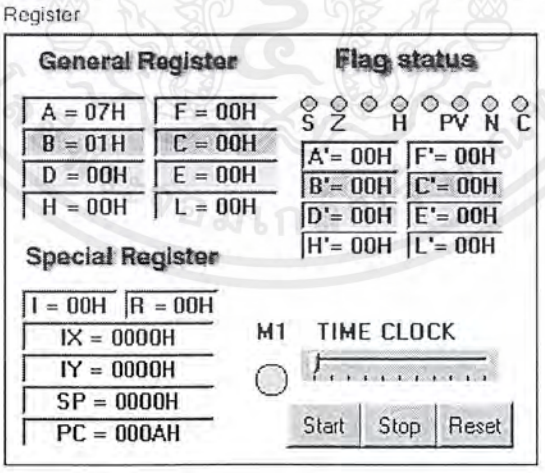
**รูปที่ 4.11** ผลการทดสอบคำสั่ง ADD ในหน้าต่างหน่วยความจำ

เนื่องจากโปรแกรมนั้นมีการทดสอบคำสั่ง ADD หลายรูปแบบดังนั้น เพื่อที่จะดูผลการรันนั้น จึงทำการรันแบบสเตป เพื่อดูว่านั้นการทำคำสั่งนั้นมีผลอย่างไรกับหน้าต่างรีจิสเตอร์ สำหรับในการทดสอบโปรแกรมตามบรรทัดที่ 1 ถึง บรรทัดที่ 3 มีผลการทดสอบดังแสดงในรูปที่ 4.12



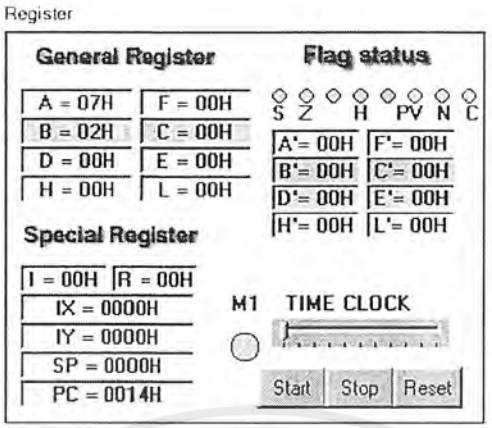

**รูปที่ 4.12** ผลการทดสอบสอบคำสั่ง ADD บรรทัดที่ 0 ถึง 3 ในหน้าต่างรีจิสเตอร์

ผลการทดสอบโปรแกรมตามบรรทัดที่ 4 ถึง บรรทัดที่ 5 จะมีผลการทดสอบดังแสดงในรูปที่ 4.13



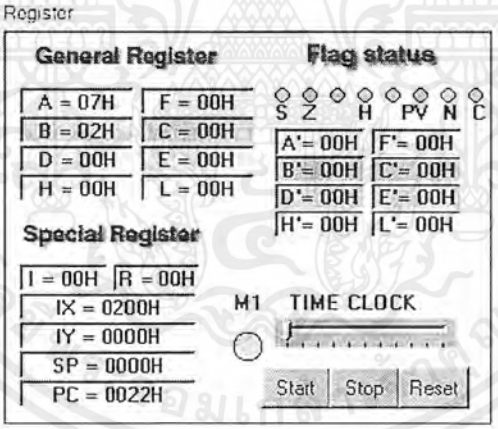

**รูปที่ 4.13** ผลการทดสอบสอบคำสั่ง ADD บรรทัดที่ 4 ถึง 5 ในหน้าต่างรีจิสเตอร์

ผลการทดสอบโปรแกรมตามบรรทัดที่ 6 ถึง บรรทัดที่ 10 จะมีผลการทดสอบดังแสดงในรูปที่ 4.14



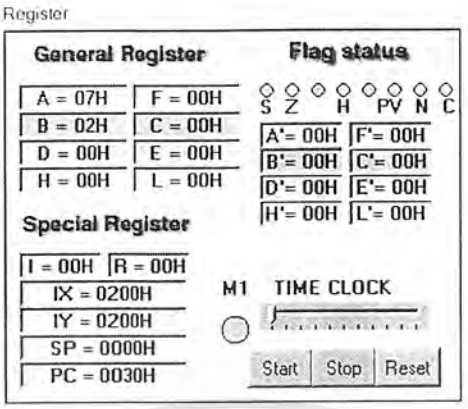

**รูปที่ 4.14** ผลการทดสอบสอบคำสั่ง ADD บรรทัดที่ 6 ถึง 10 ในหน้าต่างรีจิสเตอร์

ผลการทดสอบโปรแกรมตามบรรทัดที่ 11 ถึง บรรทัดที่ 15 จะมีผลการทดสอบดังแสดงใน รูปที่ 4.15




**รูปที่ 4.15** ผลการทดสอบสอบคำสั่ง ADD บรรทัดที่ 16 ถึง 10 ในหน้าต่างรีจิสเตอร์

ผลการทดสอบโปรแกรมตามบรรทัดที่ 16 ถึง บรรทัดที่ 21 จะมีผลการทดสอบดังแสดงใน รูปที่ 4.16




รูปที่ 4.16 ผลการทดสอบสอบคำสั่ง ADD บรรทัดที่ 16 ถึง 21 ในหน้าต่างรีจิสเตอร์

### 4.2.2 การทดสอบคำสั่ง ADC

คำสั่ง ADC เป็นการบวกอีกแบบหนึ่ง คือ แฟลกตัวทดจะถูกบวกเข้ามาทางหลักค่าต่ำสุด

**การทดสอบ**

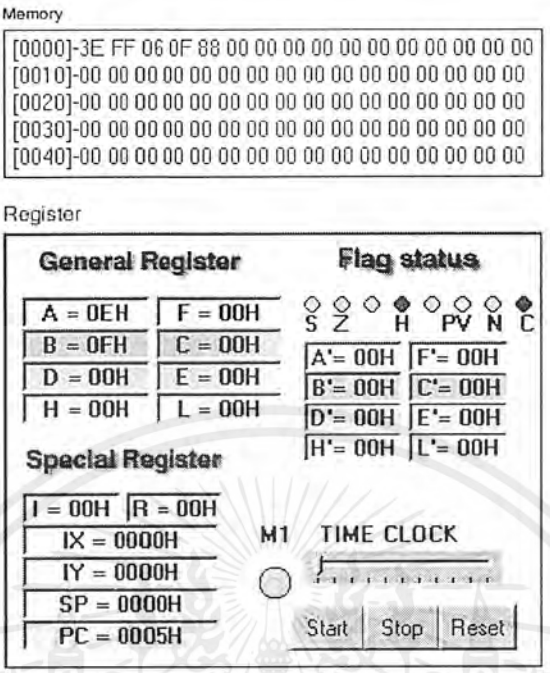
ทดสอบคำสั่ง ADC โดยการป้อนโปรแกรมดังรูปที่ 4.17 โปรแกรมจะเก็บค่าไว้ที่ 00H จนถึง 04H

```
LD      A,0FFH
LD      B,0FH
ADC     B
RST     18H
```

รูปที่ 4.17 โปรแกรมทดสอบคำสั่ง ADC

**ผลการทดสอบ**

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 04H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.18




รูปที่ 4.18 ผลการทดสอบคำสั่ง ADC

### 4.2.3 การทดสอบคำสั่ง SUB

คำสั่ง SUB เป็นการลบอีกแบบหนึ่ง คือ แฟลกตัวทดจะถูกบวกเข้ามาทางหลักค่าต่ำสุด

**การทดสอบ**

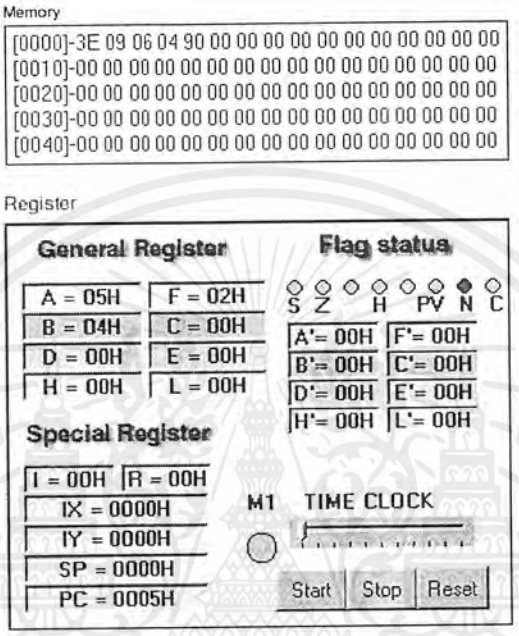
ทดสอบคำสั่ง SUB โดยการป้อนโปรแกรมดังรูปที่ 4.19 โปรแกรมจะเก็บค่าไว้ที่ 00H จนถึงตำแหน่ง 04H

```
LD    A,09H
LD    B,04H
SUB   B
LD    A,09H
LD    B,04H
SBC   B
```

รูปที่ 4.19 โปรแกรมทดสอบคำสั่ง SUB

**ผลการทดสอบ**

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 04H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.20




รูปที่ 4.20 ผลการทดสอบคำสั่ง SUB

### 4.2.3 การทดสอบคำสั่ง SBC

**การทดสอบ**

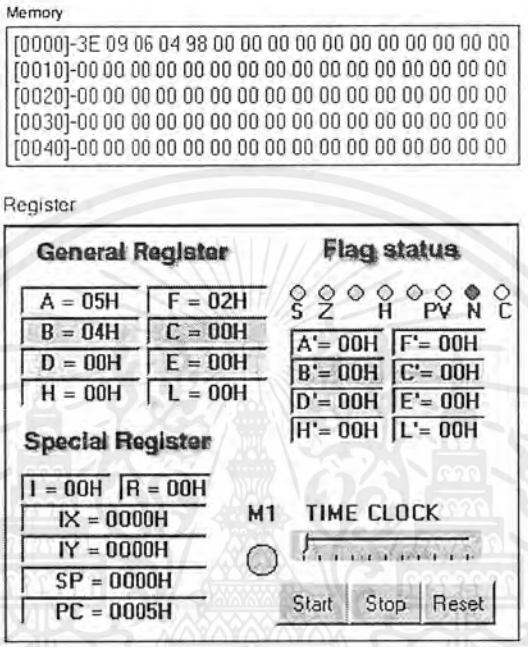
การทดสอบคำสั่ง SBC โดยการป้อนโปรแกรมทดสอบดังในรูปที่ 4.21 ซึ่งโปรแกรมจะเก็บค่าไว้ในตำแหน่งที่ 00H จนถึง 04H

```
LD    A,09H
LD    B,04H
SUC   B
```

รูปที่ 4.21 โปรแกรมการทดสอบคำสั่ง SBC

**ผลการทดสอบ**

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 04H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.22




รูปที่ 4.22 ผลการทดสอบคำสั่ง SBC

## 4.2.3 การทดสอบคำสั่ง AND

**การทดสอบ**

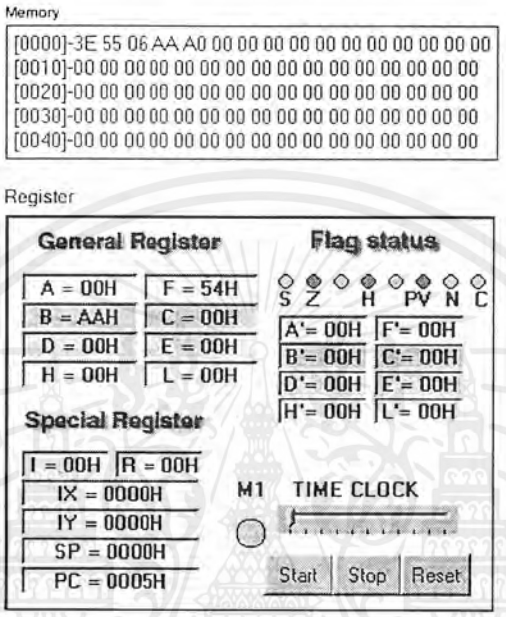
ทดสอบคำสั่ง AND ด้วยการป้อนโปรแกรมดังรูปที่ 4.23 โดยโปรแกรมจะเก็บค่าไว้ที่ 00H จนถึง 04H

```
LD      A,055H
LD      B,0AAH
AND     B
RST     18H
```

รูปที่ 4.23 โปรแกรมทดสอบคำสั่ง AND

**ผลการทดสอบ**

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 04H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.24




รูปที่ 4.24 ผลการทดสอบคำสั่ง AND

## 4.2.4 การทดสอบคำสั่ง OR

**การทดสอบ**

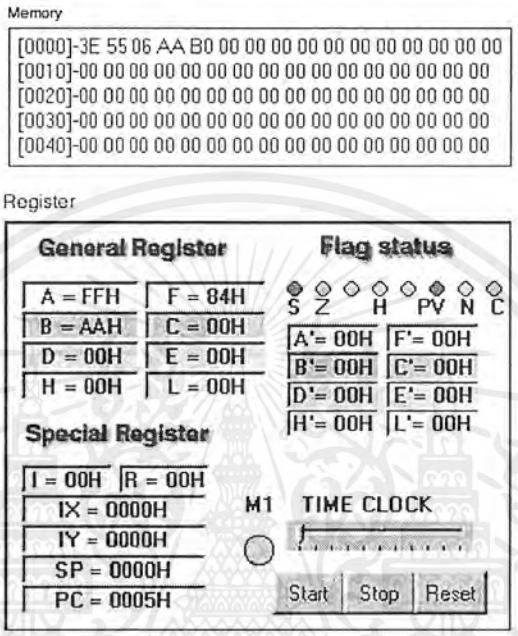
ทดสอบคำสั่ง OR ด้วยการป้อนโปรแกรมดังรูปที่ 4.25 โดยโปรแกรมจะเก็บค่าไว้ที่ 00H จนถึง 04H

```
        LD      A,055H
        LD      B,0AAH
        OR      B
        RST     18H
```

รูปที่ 4.25 โปรแกรมทดสอบคำสั่ง OR

**ผลการทดสอบ**

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 04H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 2.26




รูปที่ 4.26 ผลการทดสอบคำสั่ง OR

## 4.2.5 การทดสอบคำสั่ง XOR

**การทดสอบ**

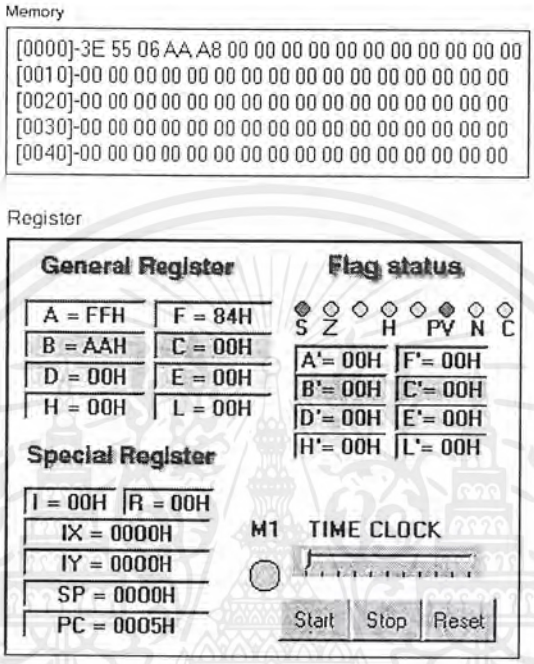
ทดสอบคำสั่ง XOR ด้วยการป้อนโปรแกรมดังรูปที่ 4.27 โดยโปรแกรมจะเก็บค่าไว้ที่ 00H จนถึง 04H

```
LD      A,055H
LD      B,0AAH
XOR     B
```

รูปที่ 4.27 โปรแกรมทดสอบคำสั่ง XOR

**ผลการทดสอบ**

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 04H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.28




รูปที่ 4.28 ผลการทดสอบคำสั่ง XOR

## 4.2.6 การทดสอบคำสั่ง CP

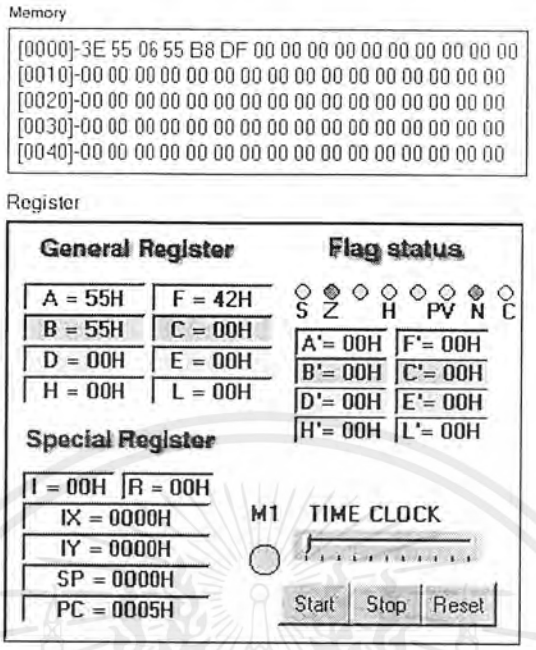
ทดสอบคำสั่ง CP ด้วยการป้อนโปรแกรมดังรูปที่ 4.29 โดยโปรแกรมจะเก็บค่าไว้ที่ 00H จนถึง 04H

```
LD    A,055H
LD    B,055H
CP    B
RST   18H
```

รูปที่ 4.29 โปรแกรมทดสอบคำสั่ง CP

**ผลการทดสอบคำสั่ง CP**

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 04H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.30




รูปที่ 4.30 ผลการทดสอบคำสั่ง CP

### 4.2.7 การทดสอบคำสั่ง INC

1) การทดสอบคำสั่ง INC r

การทดสอบ

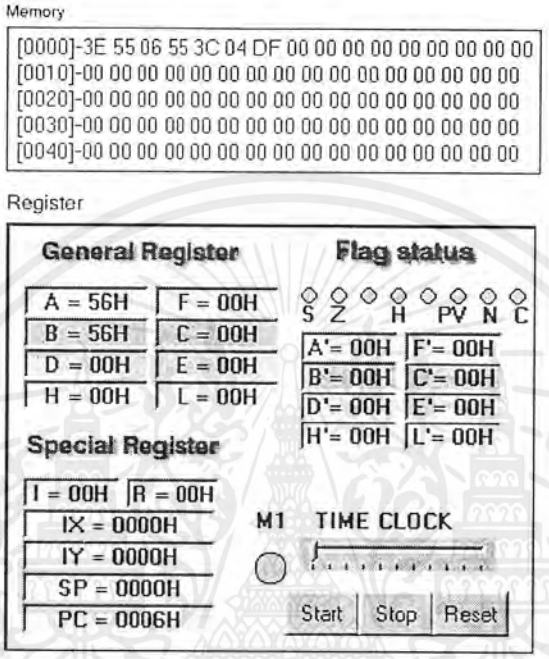
ทดสอบคำสั่ง INC r ด้วยการป้อนโปรแกรมดังรูปที่ 4.31 โดยโปรแกรมจะเก็บค่าไว้ที่ 00H จนถึง 05H

```
LD    A,055H
LD    B,055H
INC   A
INC   B
RST   18H
```

รูปที่ 4.32 โปรแกรมทดสอบคำสั่ง INC r

ผลการทดสอบ

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 05H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.32




รูปที่ 4.32 ผลการทดสอบคำสั่ง INC r

2) การทดสอบคำสั่ง INC (HL)

การทดสอบ

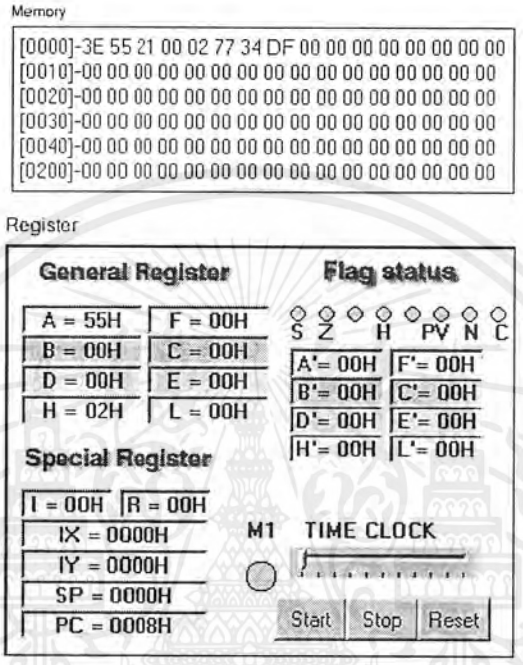
ทดสอบคำสั่ง INC (HL) ด้วยการป้อนโปรแกรมดังรูปที่ 4.33 โดยโปรแกรมจะเก็บค่าไว้ที่ 00H จนถึง 0200H

```
LD      A,055H
LD      HL,0200H
LD      (HL),A
INC     (HL)
RST     18H
```

รูปที่ 4.33 โปรแกรมทดสอบคำสั่ง INC (HL)

ผลการทดสอบ

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 0200H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.34




รูปที่ 4.34 ผลการทดสอบคำสั่ง INC (HL)

## 4.2.10 การทดสอบคำสั่ง DEC

การทดสอบ

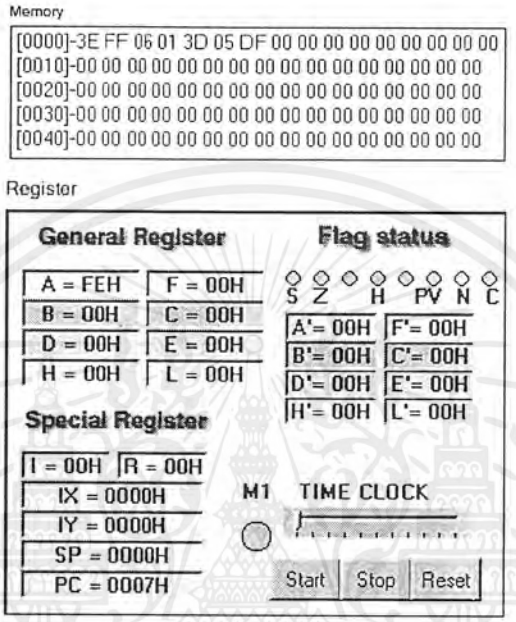
ทดสอบคำสั่ง DEC ด้วยการป้อนโปรแกรมดังรูปที่ 4.35 โดยโปรแกรมจะเก็บค่าไว้ที่ 00H จนถึง 06H

```
LD      A,0FFH
LD      B,01H
DEC     A
DEC     B
RST     18H
```

รูปที่ 4.35 โปรแกรมทดสอบคำสั่ง DEC

ผลการทดสอบ

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 06H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.36




รูปที่ 4.36 ผลการทดสอบคำสั่ง DEC

## 4.3 กลุ่มคำสั่งในการเลื่อนข้อมูลเป็นวงและการเลื่อนข้อมูล

### 4.3.1 การทดสอบคำสั่ง RLCA

การทดสอบ

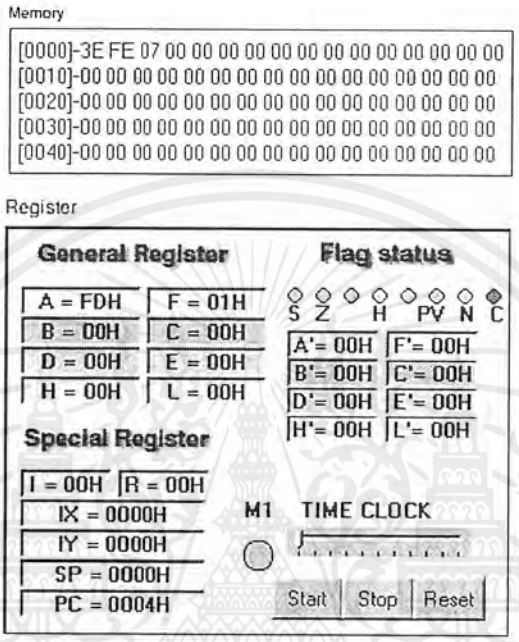
ทดสอบคำสั่ง RLCA ด้วยการป้อนโปรแกรมดังรูปที่ 4.37 โดยโปรแกรมจะเก็บค่าไว้ที่ 00H จนถึง 03H

```
LD      A,0FEH
RLCA
RST     18H
```

รูปที่ 4.37 โปรแกรมทดสอบคำสั่ง RLCA

ผลการทดสอบ

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 03H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.38




รูปที่ 4.38 ผลการทดสอบคำสั่ง RLCA

## 4.3.2 การทดสอบคำสั่ง RRCA

การทดสอบ

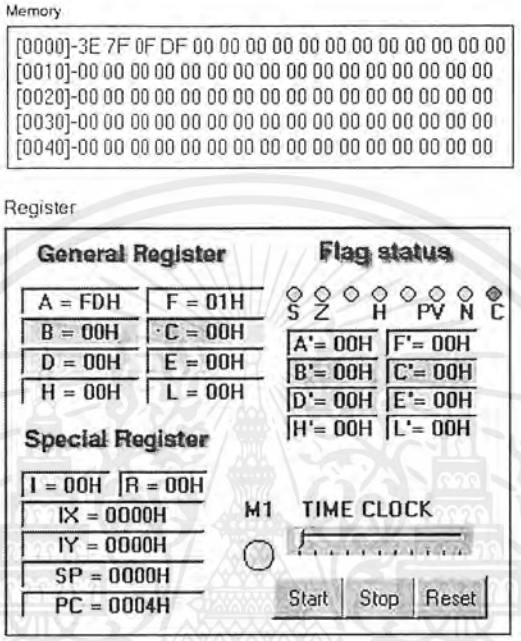
ทดสอบคำสั่ง RRCA ด้วยการป้อนโปรแกรมดังรูปที่ 4.39 โดยโปรแกรมจะเก็บค่าไว้ที่ 00H จนถึง 03H

```
LD      A,07FH
RRCA
RST     18H
```

รูปที่ 4.39 โปรแกรมทดสอบคำสั่ง RRCA

ผลการทดสอบ

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 03H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.40




รูปที่ 4.40 ผลการทดสอบคำสั่ง RRCA

## 4.4 กลุ่มคำสั่งเกี่ยวกับการกระทำภายในข้อมูลบิตต่าง ๆ

### 4.4.1 การทดสอบคำสั่ง BIT

1) การทดสอบคำสั่ง BIT b, r

การทดสอบ

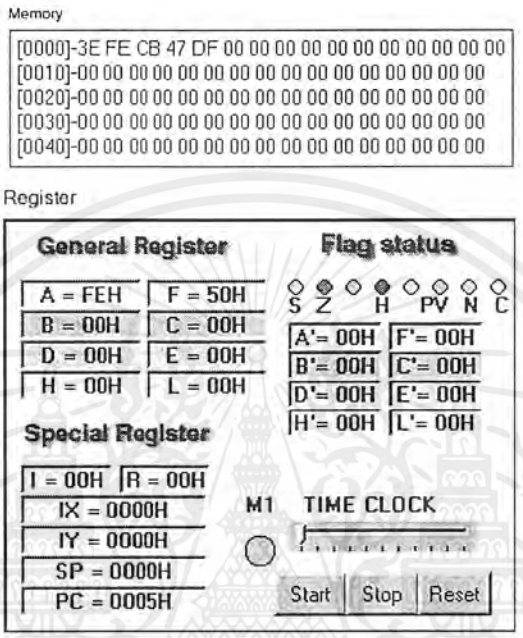
ทดสอบคำสั่ง BIT b, r ด้วยการป้อนโปรแกรมดังรูปที่ 4.41 โดยโปรแกรมจะเก็บค่าไว้ที่ 00H จนถึง 03H

```
LD      A,0FEH
BIT     0,A
RST     18H
```

รูปที่ 4.41 โปรแกรมทดสอบคำสั่ง BIT b, r

ผลการทดสอบ

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 03H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.42




รูปที่ 4.42 ผลการทดสอบคำสั่ง BIT b, r

**2) ทดสอบคำสั่ง BIT b, (IX+d)**

การทดสอบ

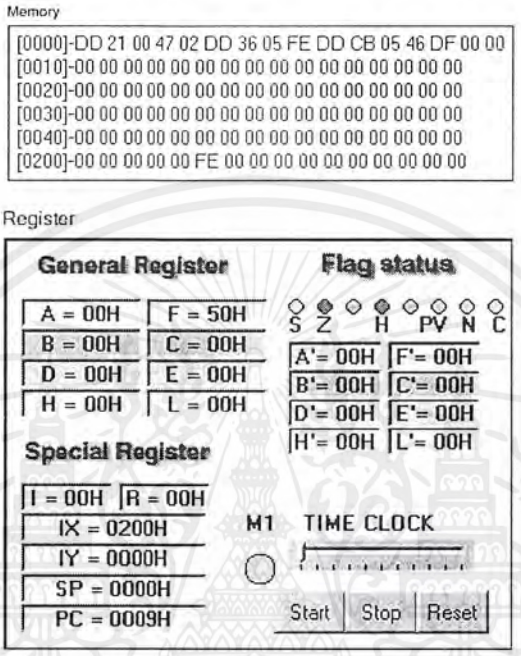
ทดสอบคำสั่ง BIT b, (IX+d) ด้วยการป้อนโปรแกรมดังรูปที่ 4.43 โดยโปรแกรมจะเก็บค่าไว้ที่ 00H จนถึง 0205H

```
        LD      IX,0200H
        LD      (IX+05H),0FEH
        BIT     0,(IX+05H)
        RST     18H
```

รูปที่ 4.43 โปรแกรมทดสอบคำสั่ง BIT b, (IX−d)

**ผลการทดสอบ**

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 0205H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.44




รูปที่ 4.44 ผลการทดสอบคำสั่ง BIT b, (IX+d)

## 4.4.2 การทดสอบคำสั่ง SET

**การทดสอบ**

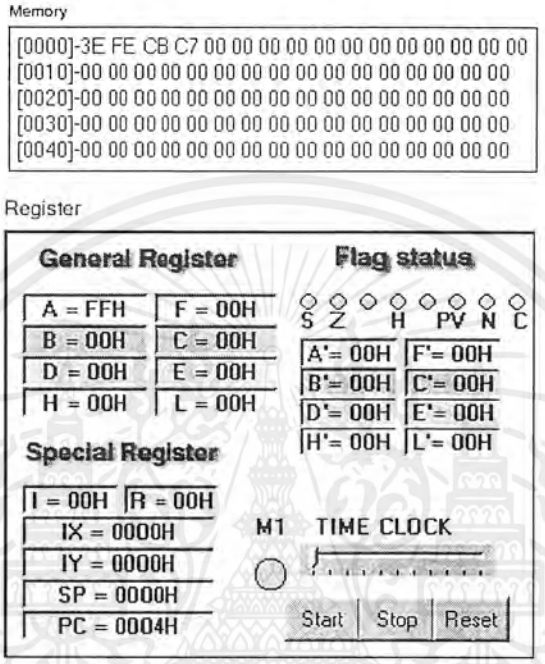
ทดสอบคำสั่ง SET ด้วยการป้อนโปรแกรมดังรูปที่ 4.45 โดยโปรแกรมจะเก็บค่าไว้ที่ 00H จนถึง 04H

```
LD      A,0FEH
SET     0,A
RST     18H
```

รูปที่ 4.45 โปรแกรมทดสอบคำสั่ง SET

ผลการทดสอบ

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 04H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.46




รูปที่ 4.46 ผลการทดสอบคำสั่ง SET

### 4.4.3 การทดสอบคำสั่ง RES

การทดสอบ

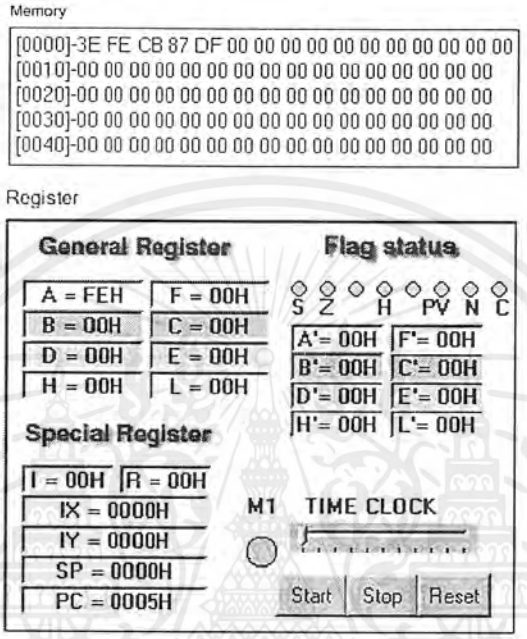
ทดสอบคำสั่ง RES ด้วยการป้อนโปรแกรมดังรูปที่ 4.47 โดยโปรแกรมจะเก็บค่าไว้ที่ 00H จนถึง 04H

```
LD      A,0FEH
RES     0,A
```

รูปที่ 4.47 โปรแกรมทดสอบคำสั่ง RES

ผลการทดสอบ

ผลการทดสอบเมื่อรันด้วยโปรแกรม Z-80 Simulator ที่ถูกต้องโปรแกรมตำแหน่ง 00H ถึง 04H จะต้องมีค่าดังในหน้าต่างหน่วยความจำ และหน้าต่างรีจิสเตอร์ ดังแสดงในรูปที่ 4.48




รูปที่ 4.48 ผลการทดสอบคำสั่ง RES

# บทที่ 5
# บทสรุป ปัญหา แนวทางแก้ไข และพัฒนา

## 5.1 บทสรุป

ก่อนที่โครงงานนี้จะสำเร็จได้นั้นจำเป็นอย่างยิ่งที่จะต้องได้รับความช่วยเหลือจากท่านอาจารย์ในภาควิชา และบุคคลอื่น ๆ ซึ่งผู้จัดทำได้รับความช่วยเหลือ และข้อแนะนำต่าง ๆ ที่เป็นประโยชน์ต่อผู้จัดทำในการจัดทำโครงงานนี้ขึ้นมา นอกจากนั้นการศึกษา ค้นคว้า ในหลักการและทฤษฎีต่าง ๆ ที่เกี่ยวข้องกับการทำโครงงานนี้ทำให้ผู้จัดทำได้ทราบถึงความรู้ต่าง ๆ อาจจะหาไม่ได้จากในห้องเรียน และจากการทำโครงงานนี้เองทำให้ผู้จัดทำมีความเห็นว่าการเรียนนั้นนอกจากที่เราจะเรียนภายในห้องเรียนเท่านั้นแล้ว เรายังต้องศึกษา ค้นคว้า จากภายนอกอีกด้วย และอีกสิ่งหนึ่งที่มีความสำคัญคือ การได้เห็นว่าสิ่งที่เราเรียนไปนั้นสามารถที่จะนำไปใช้ประโยชน์ได้จริง ๆ สำหรับโครงงานนี้ทางคณะผู้จัดทำได้มีการวางแผนในการจัดทำโครงงาน โดยมีระยะเวลาของแผนทั้งหมด 5 เดือน เริ่มตั้งแต่การรวบรวม ค้นคว้า หาเอกสารที่เกี่ยวข้อง ทั้งจากตำราภาษาไทย ภาษาอังกฤษ และจากวารสารต่าง ๆ และการขอคำปรึกษาจากอาจารย์ที่ปรึกษา ซึ่งได้วางแผนในการศึกษาไว้ประมาณ 2 สัปดาห์ จากนั้นก็เริ่มทำการวางแผนเกี่ยวกับการเขียนโปรแกรม การออกแบบหน้าจอของโปรแกรมในส่วนต่าง ๆ และเมื่อทราบส่วนประกอบต่าง ๆ แล้วก็จัดการเขียนโปรแกรมเป็นส่วนย่อย ๆ ซึ่งทำให้การพัฒนาในขั้นต่อไปสามารถทำได้ง่าย และมีประสิทธิภาพในการใช้งานดีกว่าแบบอื่น ๆ และเมื่อเขียนโปรแกรมเป็นที่หน้าพอใจแล้วคือ สามารถที่จะจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80 ได้แล้ว ก็ทำการตรวจสอบหาข้อผิดพลาดที่เกิดขึ้นจากตัวโปรแกรม โดยจะทำการทดสอบคำสั่ง ๆ ต่างที่สามารถเขียนโปรแกรมจำลองได้ ซึ่งผลการทดลองที่ออกมานั้นเป็นที่น่าพอใจว่า เราสามารถจำลองการทำงานของคำสั่ง ได้เกือบทุกคำสั่ง มียกเว้นบ้างบางคำสั่ง เช่น คำสั่งในการอินเตอร์รัพต์ แต่ทั้งนี้เราก็ยังไม่ได้ยุติเพียงเท่านั้นไม่ เรายังคงพัฒนาโปรแกรมต่อไปอย่างต่อเนื่องเพื่อให้เกิดประโยชน์อย่างสูงสุดต่อผู้ใช้แต่อีกสิ่งหนึ่งที่จะขาดไม่ได้ก็คือคู่มือในการใช้โปรแกรมซึ่งทางผู้จัดทำได้จัดทำขึ้นโดยมีจุดประสงค์ที่จะทำให้ผู้ใช้สามารถที่จะอ่าน ค้นคว้า ได้ด้วยตนเอง จากทั้งหมดที่ได้กล่าวมานั้นเป็นข้อสรุปที่เกิดขึ้นนับจากวันแรกที่ เริ่มสอบหัวข้อปริญญานิพนธ์ จนมาถึงวันสุดท้าย คือ การสอบปริญญานิพนธ์ และส่งผลให้เกิดความสำเร็จของปริญญานิพนธ์นี้ขึ้นมา

## 5.2ปัญหา

ปัญหาที่เกิดขึ้นจากการทำโครงงาน มีหลายข้อดังนี้

1) แอสเซมเบลอที่ใช้ในโปรแกรมเป็นแอสเซมเบลอที่ทำงานบนดอส แต่ส่วนของโปรแกรมหลักใช้งานในระบบวินโดว์ ทำให้เกิดปัญหา คือ เมื่อทำการคอมไพล์ ตัวโปรแกรมหลักจะเรียกตัวแอสเซมเบลอ เมื่อทำการคอมไพล์เสร็จโปรแกรมจะเรียก List ไฟล์มาแสดง เพื่อให้ทราบว่ามีการ Error หรือไม่ แต่เนื่องจากโปรแกรมที่ทำงานบนวินโดว์จะทำงานเร็วกว่าโปรแกรมที่ทำงานบนดอส ดังนั้นเมื่อโปรแกรมอ่าน List ไฟล์มาแสดงจะไม่สามารถแสดง List ไฟล์ได้ เนื่องจากโปรแกรมที่ทำการคอมไพล์สร้าง List ไฟล์ไม่เสร็จดังนั้นเมื่อโปรแกรมบนวินโดว์ทำการอ่านจึงไม่เจอไฟล์ ดังนั้นจึงแก้ปัญหาโดยการทำดีเลย์ ขึ้นมาเพื่อรอให้โปรแกรมแอสเซมเบลอสร้าง List ไฟล์ให้เสร็จก่อนแล้วจึงค่อยทำการอ่านไฟล์ขึ้นมาแสดง แต่หน่วงเวลาจะทำให้เกิดปัญหา คือ เมื่อนำโปรแกรมไปใช้กับเครื่องที่มีความเร็วจะไม่เท่ากัน

2) แอสเซมเบลอที่ใช้ในโปรแกรมเป็นแอสเซมเบลอที่ทำงานบนดอสไม่รองรับการแอสเซมเบลอไฟล์ .ASM ที่มีชื่อยาวเกินกว่า 8 ตัวอักษรจึงทำให้ตัวแอสเซมเบลอไม่สามารถสร้างไฟล์ .OBJ และ .LST

3) ออบเจ็กที่ใช้ในการแสดงผลของชุดอินเตอร์เฟสเป็นออบเจ็กที่เป็นมาตรฐานที่ให้มากับวิชวลเบสิก ทำให้การแสดงผลไม่สวยงาม และจัดรูปแบบของการแสดงผลได้ยาก

4) การใช้โปรแกรมวิชวลเบสิกเป็นตัวเขียนโปรแกรมจำลองของในส่วนของชุดคำสั่งนั้นค่อนข้างจะยุ่งยากเนื่องจากวิชวลเบสิกนั้นไม่คำสั่งที่สนับการกระทำระดับบิตจะต้องใช้การทำการเขียนโปรแกรมติดต่อกับไฟล์ Bits32.dll ซึ่งเป็นไฟล์ที่มีผู้เขียนขึ้นไว้แล้ว จะไม่สนับสนุนตัวแปรของวิชวลเบสิกที่เป็นแบบไบต์ หรือ เวิร์ด ซึ่งจะมีค่าที่ไม่ตรงกัน ซึ่งทำให้การเขียนโปรแกรมจะต้องเลือกใช้ตัวแปรให้เหมาะสม

5) ในการทดสอบในการออกแบบโครงสร้างในการทำงานของโปรแกรมครั้งแรก กำหนดให้การจำลองคำสั่งนั้นจะใช้ฐานข้อมูลเข้าใช้ร่วมในการค้น ซึ่งฐานข้อมูลจะทำการเก็บข้อมูลเกี่ยวกับข้อมูลของคำสั่ง เช่น ชื่อคำสั่ง รูปแบบของคำสั่ง ขนาดของคำสั่ง ซึ่งในการทดสอบนั้นสามารถที่ทำได้ โดยจำลองคำสั่งได้ เมื่อได้นำไปใช้ทดสอบกับเครื่องที่มีความเร็วต่ำ ก็จะทำให้การทำงานช้ามาก ซึ่งไม่เป็นไปตามลักษณะของการทำงานของไมโครโปรเซสเซอร์ Z-80 จึงทำให้การทำงานล่าช้า

## 5.3 แนวทางการแก้ไข

1) ควรจะเขียนโปรแกรมการดีเลย์ให้เป็นแบบของเรียวไทม์เพราะเมื่อนำโปรแกรมไปใช้กับเครื่องอื่นความเร็วจะได้เท่ากัน

2) เขียนแอสเซมเบลอซึ่งรองรับการทำงานของไฟล์ระบบวินโดว์

3) เขียนออบเจ็กขึ้นมาเอง

4) ใช้โปรแกรมที่มีการสนับสนุนการประมวลระดับบิตได้ เช่น โปรแกรมวิชวลซี หรือโปรแกรมเดลไฟล์ เป็นต้น

5) การแก้ไขนั้น ได้ทำการแก้ไขโดยการไม่นำข้อมูลจากฐานข้อมูลมาเปรียบเทียบ แต่ใช้การเขียนฟังก์ชันย่อยของแต่ละชุดคำสั่ง แล้วเมื่อทำการค้นหาก็จะตรวจสอบว่าเป็นโค้ดใดแล้วไปทำงานตามคำสั่ง

## 5.4 แนวทางการพัฒนาโครงงาน

แนวทางต่อไปที่ควรมีการพัฒนาโปรแกรมให้มีประสิทธิภาพสูงขึ้น มีดังนี้

1) ควรมีการเขียนแอสเซมเบลอขึ้นใช้งานเอง

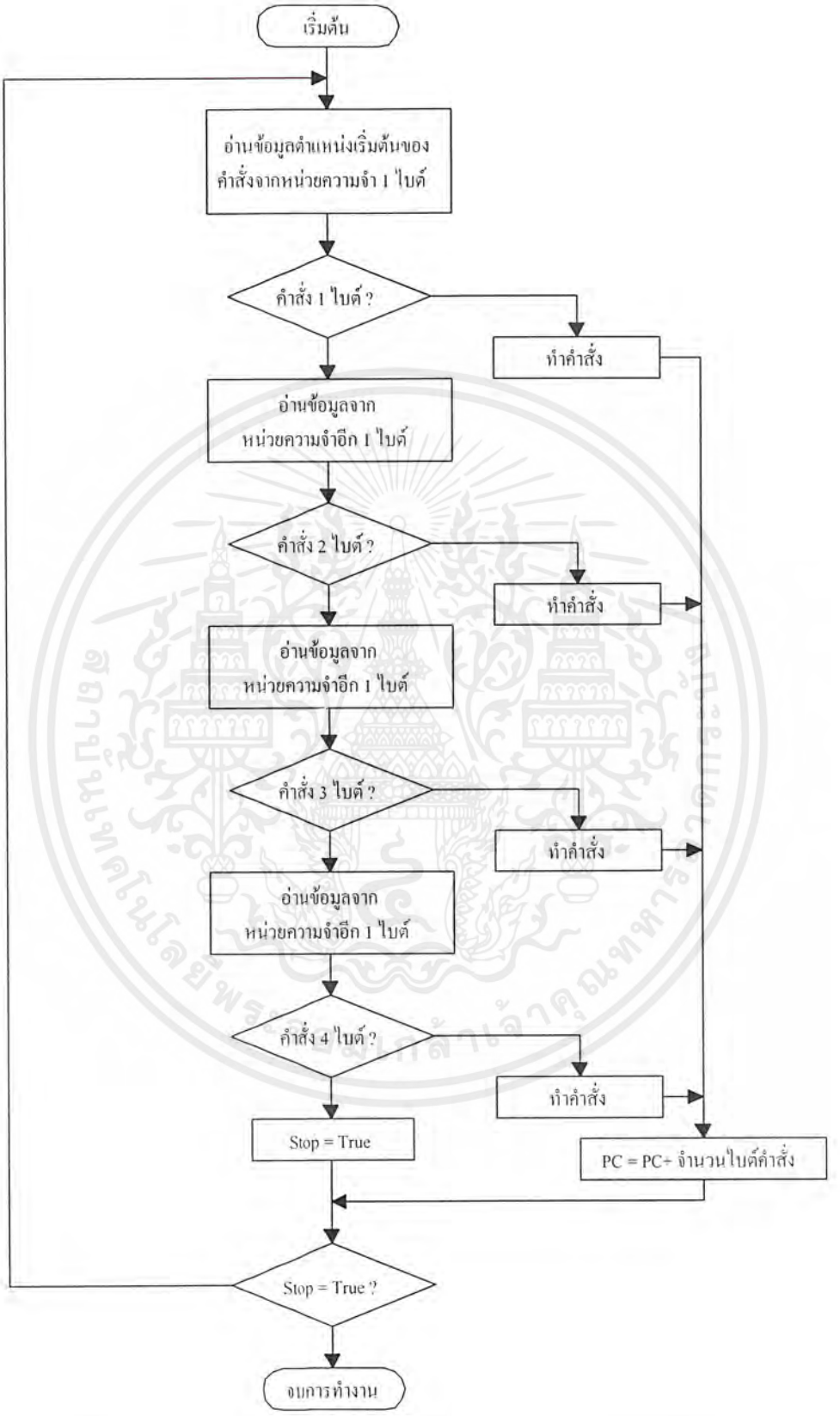
2) ควรมีการเพิ่มชุดอินเตอร์เฟสที่สามารถติดต่อโดยผ่านทางพอร์ตได้

3) ควรปรับปรุงโครงสร้างของหน้าจอ และรูปแบบให้มีรูปแบบที่สวยงามมากขึ้น

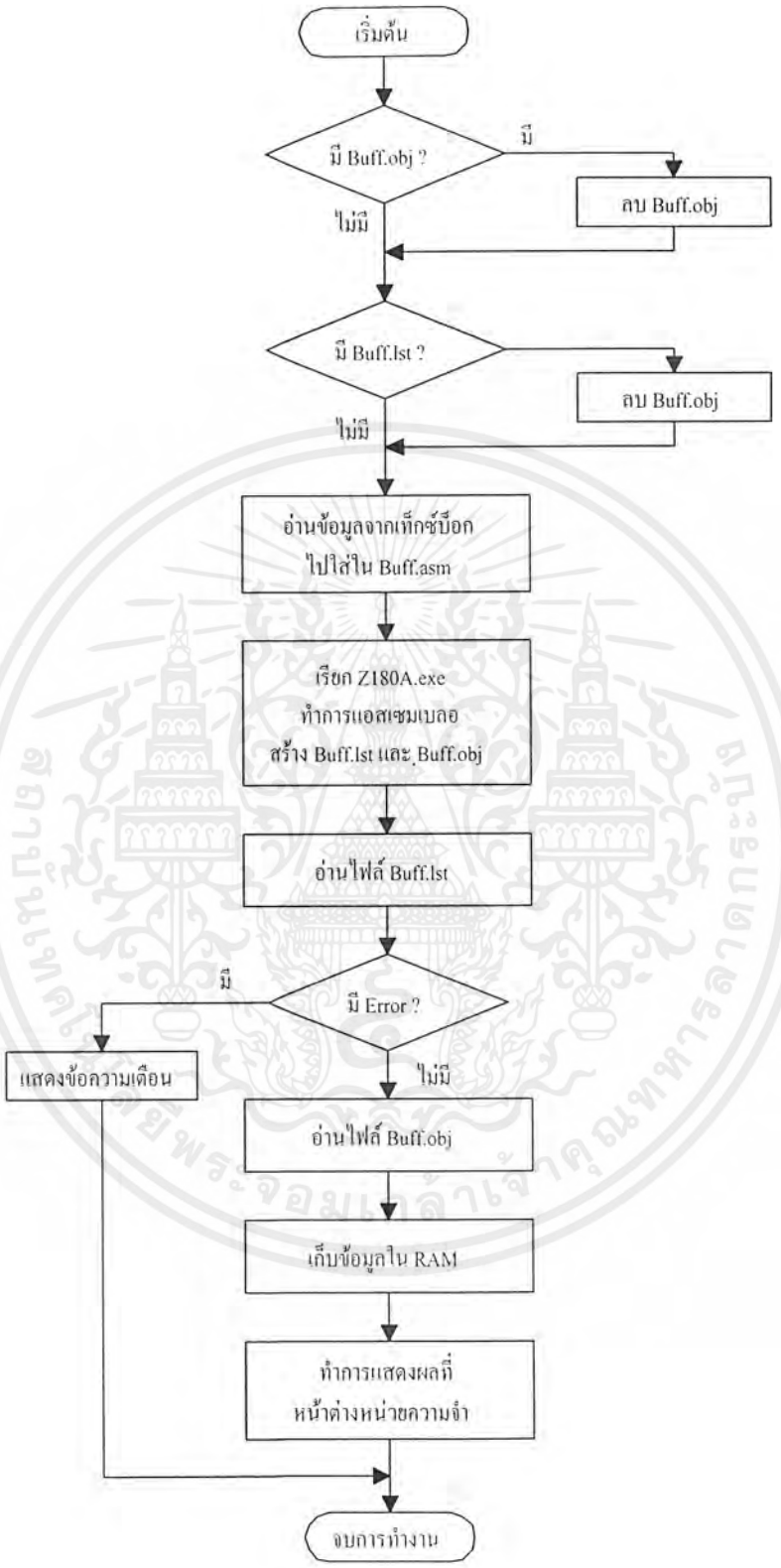
4) จำลองชุดของคำสั่งให้มากขึ้น

# ภาคผนวก ก

# ผังการทำงาน และโปรแกรม




**รูปที่ ก.1** แผนผังโปรแกรมจำลองการทำงานไมโครโปรเซสเซอร์ Z-80




**รูปที่ ก.2** แผนผังโปรแกรมการแอสเซมเบลอ

```
Option Explicit
Private Sub MDIForm_Load()
    ResWin = True
    MemWin = True
    ToolWin = True
    ResMove = False
    MemMove = False
    EditWin = False
    Me.Move 1441 * 0.3, 1441 * 0.1, 1441 * 7.8, 1441 * 5.5
    tips.Show
    frmSplash.Show
End Sub
Private Sub mnuAbout_Click()
    frmAbout.Show
End Sub
Private Sub mnuCascade_Click()
    Main.Arrange vbCascade
End Sub
Private Sub mnuCompile_Click()
    Unload Memo
    Unload List
    Regis.Hide
    Compile_asm
    Load List
    Load Memo
    List.Visible = True
    Regis.Visible = True
    Memo.Visible = True
    Regis.Show
End Sub
Private Sub mnuCopy_Click()
    Editor.Visible = False
    EditCopy
    Editor.Visible = True
End Sub
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.ActiveForm.ActiveControl.SelText <> "" 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 & "คุณต้องการบันทึกการเปลี่ยนแปลงนี้หรือไม่?"
      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$
          ' 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
        'Case 2    ' User chose Cancel. Cancels unload.
          'Cancel = True
          Exit Sub
      End Select
    End If
  End If
  'FileNew
```

```
    SetMenu_On
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 & "คุณต้องการบันทึกการเปลี่ยนแปลงนี้หรือไม่?"
            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$
                    ' 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
                    OpenFileDialog
                Case 2      ' User chose Cancel. Cancels unload.
                    'Cancel = True
                    Exit Sub
            End Select
        End If
    End If
    OpenFileDialog
    SetMenu_On
    Main.Toolbar1.Buttons(13).Enabled = True
End Sub
Private Sub mnuFileSave_Click()
    Dim strFilename$
        If UCase$(Left$(Editor.Caption, 8)) = "UNTITLED" Then
            strFilename$ = SaveFileAsDialog(CStr(Editor.Caption))
        Else
            strFilename$ = Editor.Caption
        End If
        If strFilename$ <> "" Then SaveTextFile strFilename$
```

```
            Main.Toolbar1.Buttons(13).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(13).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 mnuStart_Click()
      Editor.Visible = False
      List.Visible = False
      Runtum
      Editor.Visible = True
      List.Visible = True
End Sub
Private Sub Toolbar1_ButtonClick(ByVal Button As MSComctlLib.Button)
  Dim countnum As Integer
   Select Case Button.Index
      Case 1 'new
   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 = "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
            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$
            ' 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
          Case 2    ' User chose Cancel. Cancel the unload.
              'Cancel = True
              Exit Sub
        End Select
      End If
    End If
    FileNew
    SetMenu_On
      Case 2 'open
      If EditWin = True Then
      ' 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
          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$
            ' 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
            OpenFileDialog
          Case 2    ' User chose Cancel. Cancels unload.
              'Cancel = True
              Exit Sub
```

```
        End Select
    End If
Mnd If
OpenFileDialog
SetMenu_On
Main.Toolbar1.Buttons(13).Enabled = True
  Case 3 'save
  If UCaseS(LeftS(Editor.Caption, 8)) = "UNTITLED" Then
     strFilenameS = SaveFileAsDialog(CStr(Editor.Caption))
  Else
     strFilenameS = Editor.Caption
  End If
  If strFilenameS <> "" Then SaveTextFile strFilenameS
     Main.Toolbar1.Buttons(13).Enabled = True
  Case 5 'cut
           EditCut
  Case 6 'copy
           EditCopy
  Case 7 'paste
           EditPaste
  Case 8 'find
  Case 10 'undo
  Case 11 'redo
  Case 13 'compile
  Case 14 'run
           Main.mnuStart.Enabled = False
           Main.mnustep.Enabled = False
           Main.mnuStop.Enabled = True
           Main.Toolbar1.Buttons(14).Enabled = False
           Main.Toolbar1.Buttons(15).Enabled = False
           Main.Toolbar1.Buttons(16).Enabled = True
  Case 15 'step
  Case 16 'stop
           Main.mnuStart.Enabled = True
           Main.mnustep.Enabled = True
           Main.mnuStop.Enabled = False
           Main.mnuReset.Enabled = True
           Main.Toolbar1.Buttons(14).Enabled = True
           Main.Toolbar1.Buttons(15).Enabled = True
           Main.Toolbar1.Buttons(16).Enabled = False
           Main.Toolbar1.Buttons(17).Enabled = True
  Case 17 'reset
           Call Initial
           Main.mnuStart.Enabled = False
           Main.mnuStop.Enabled = False
           Main.mnustep.Enabled = False
           Main.mnuCompile.Enabled = True
```

```
            Main.Toolbar1.Buttons(14).Enabled = False
            Main.Toolbar1.Buttons(15).Enabled = False
            Main.Toolbar1.Buttons(16).Enabled = False
    End Select
End Sub
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
            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$
                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
Private Sub Form_Load()
   Me.Caption = "About " & App.Title
   lblVersion.Caption = "Version " & App.Major & "." & App.Minor & "." & App.Revision
   lblTitle.Caption = App.Title
End Sub
Public Sub StartSysInfo()
   On Error GoTo SysInfoErr
   Dim rc As Long
   Dim SysInfoPath As String
   ' Try To Get System Info Program Path\Name From Registry...
   If GetKeyValue(HKEY_LOCAL_MACHINE, gREGKEYSYSINFO, gREGVALSYSINFO, SysInfoPath) Then
   ' Try To Get System Info Program Path Only From Registry...
   ElseIf GetKeyValue(HKEY_LOCAL_MACHINE, gREGKEYSYSINFOLOC, gREGVALSYSINFOLOC, SysInfoPath) Then
      ' Validate Existance Of Known 32 Bit File Version
      If (Dir(SysInfoPath & "\MSINFO32.EXE") <> "") Then
         SysInfoPath = SysInfoPath & "\MSINFO32.EXE"
      ' Error - File can not be found...
      Else
         GoTo SysInfoErr
      End If
   ' Error - Registry Entry can not be found...
   Else
      GoTo SysInfoErr
   End If
   Call Shell (SysInfoPath, vbNormalFocus)
   Exit Sub
SysInfoErr:
   MsgBox "System Information Is Unavailable At This Time", vbOKOnly
End Sub
Public Function GetKeyValue(KeyRoot As Long, KeyName As String, SubKeyRef As String, ByRef KeyVal As String) As Boolean
   Dim i As Long                                    ' Loop Counter
   Dim rc As Long                                   ' Return Code
   Dim hKey As Long                                 ' Handle To An Open Registry Key
   Dim hDepth As Long
   Dim KeyValType As Long                           ' Data Type Of A Registry Key
   Dim tmpVal As String                             ' Tempory Storage For A Registry Key Value
   Dim KeyValSize As Long                           ' Size Of Registry Key Variable

' Open RegKey under KeyRoot {HKEY_LOCAL_MACHINE...}
   rc = RegOpenKeyEx(KeyRoot, KeyName, 0, KEY_ALL_ACCESS, hKey)    ' Open Registry Key
   If (rc <> ERROR_SUCCESS) Then GoTo GetKeyError                  ' Handle Error...
   tmpVal = String$(1024, 0)                                       ' Allocate Variable Space
   KeyValSize = 1024                                               ' Mark Variable Size
```

```
    '------------------------------------------------------------
    ' Retrieve Registry Key Value...
    '------------------------------------------------------------
    rc = RegQueryValueEx(hKey, SubKeyRef, 0,
                         KeyValType, tmpVal, KeyValSize)        ' Get/Create Key Value
    If (rc <> ERROR_SUCCESS) Then GoTo GetKeyError              ' Handle Errors
    If (Asc(Mid(tmpVal, KeyValSize, 1)) = 0) Then               ' Win95 Adds Null Terminated String...
        tmpVal = Left(tmpVal, KeyValSize - 1)                   ' Null Found, Extract From String
    Else                                                        ' WinNT Does NOT Null Terminate String...
        tmpVal = Left(tmpVal, KeyValSize)                       ' Null Not Found, Extract String Only
    End If
    '------------------------------------------------------------
    ' Determine Key Value Type For Conversion...
    '------------------------------------------------------------
    Select Case KeyValType                                      ' Search Data Types...
    Case REG_SZ                                                 ' String Registry Key Data Type
        KeyVal = tmpVal                                         ' Copy String Value
    Case REG_DWORD                                              ' Double Word Registry Key Data Type
        For i = Len (tmpVal) To 1 Step -1                       ' Convert Each Bit
            KeyVal = KeyVal + Hex (Asc(Mid(tmpVal, i, 1)))      ' Build Value Char. By Char.
        Next
        KeyVal = Format$("&h" + KeyVal)                         ' Convert Double Word To String
    End Select
    GetKeyValue = True                                          ' Return Success
    rc = RegCloseKey(hKey)                                      ' Close Registry Key
    Exit Function                                               ' Exit
GetKeyError:                                                    ' Cleanup After An Error Has Occured...
    KeyVal = ""                                                 ' Set Return Val To Empty String
    GetKeyValue = False                                         ' Return Failure
    rc = RegCloseKey(hKey)                                      ' Close Registry Key
End Function
Private Sub Form_KeyPress(KeyAscii As Integer)
    Unload Me
End Sub
Private Sub Form_Load()
    frmSplash.Show
    frmSplash.Refresh
     Timer1.Interval = 1000
     Timer1.Enabled = True
    Main.Show
End Sub
Private Sub Frame1_Click ()
    Unload Me
End Sub
Private Sub Timer1_Timer ()
    Unload Me
End Sub
```

```
Private Sub Command3_Click()
    Unload Me
End Sub
Private Sub Form_Load()
    Me.Move 2500, 1000, 5500, 4100
End Sub
Private Sub Label2_Click()
    tipsnew.Show
    Unload Me
End Sub
Private Sub Label5_Click()
    tipabi.Show
    Unload Me
End Sub
Private Sub t_Click()
    tipsnew.Show
    Unload Me
End Sub
Private Sub Form_Load()
    Me.Move 0, 4035, 1441 * 3, Main.Height - 1441 * 3 - 1441
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_Change()
List.Text1.Text = Txtlist
End Sub
Private Sub Form_Load()
  Me.Move Main.Width - 1441 * 4.5 - 400, 4035, 1441 * 4.65,(Main.Height - 1441 * 4
    Me.txtmem.Move 0, 0, 1441 * 4.5, 1441
    Me.VScroll1.Max = 4095
    Me.VScroll1.Move 1441 * 4.45, 0, 200, 1441
    Me.Label1.Move 1441 * 4, 1441, 700, 300
    Memo.txtmem.Visible = False
    'ใส่ข้อมูลลง Textbox
    Call displayMem
    Memo.Show
    Memo.Refresh
End Sub
Private Sub VScroll1_Change()
    ScaleM = Memo.VScroll1.Value
    Label1.Caption = Right(String(3, "0") & Hex(ScaleM * 16), 4)
    Call displayMem
End Sub
Private Sub VScroll1_Scroll()
```

```
        VScroll1_Change
        Call displayMem
End Sub
Private Sub Slider1_Change ()
    Slider1_Click
End Sub
Private Sub Slider1_Click ()
    Timer1.Interval = Slider1.Value * 100
End Sub
Private Sub Timer1_Timer ()
    If i = 0 Then
        ShPM1.FillColor = RGB (0, 255, 255)
        Regis.Refresh
        i = 2
    Else
        ShPM1.FillColor = RGB (255, 10, 0)
        Regis.Refresh
        i = 0
    End If
End Sub
Private Sub Command3_Click ()
    Unload Me
End Sub
Private Sub Form_Load()
    Me.Move 2500, 1000, 5500, 4100
End Sub
Private Sub t_Click()
    tipany.Show
    Unload Me
End Sub
Private Sub Command3_Click ()
    Unload Me
End Sub
Private Sub Form_Load()
    Me.Move 2500, 1000, 5500, 4100
End Sub
Private Sub t_Click()
    tipany.Show
    Unload Me
End Sub
Private Sub Command2_Click ()
    tiprun.Show
    Unload Me
End Sub
Private Sub Command3_Click ()
    Unload Me
End Sub
```

```
Private Sub Command4_Click ()
        tipsnew.Show
        Unload Me
End Sub
Private Sub Form_Load()
        Me.Move 2500, 1000, 5500, 4100
End Sub
Private Sub Command2_Click ()
        Unload Me
End Sub
Private Sub Command4_Click ()
        tiprun.Show
        Unload Me
End Sub
Private Sub Form_Load()
        Me.Move 2500, 1000, 5500, 4100
End Sub
Private Sub Command2_Click ()
        tipany.Show
        Unload Me
End Sub
Private Sub Command3_Click ()
      Unload Me
      End Sub
Private Sub Form_Load()
      Me.Move 2500, 1000, 5500, 4100
      End Sub
Private Sub Command2_Click ()
      tipflag.Show
      Unload Me
End Sub
Private Sub Command3_Click ()
      Unload Me
End Sub
Private Sub Command4_Click ()
      tipcompile.Show
      Unload Me
End Sub
Private Sub Form_Load()
      Me.Move 2500, 1000, 5500, 4100
End Sub
Private Sub Command1_Click ()
      Unload Me
End Sub
Private Sub Command2_Click ()
      Intro.Show
      Unload Me
```

```
End Sub
Private Sub Form_Load()
    Me.Move 2500, 1000, 5500, 4100
End Sub
Private Sub Command2_Click ()
    tipcompile.Show
    Unload Me
End Sub
Private Sub Command3_Click ()
    Unload Me
End Sub
Private Sub Command4_Click ()
    Intro.Show
    Unload Me
End Sub
Private Sub Form_Load()
    Me.Move 2500, 1000, 5500, 4100
End Sub

**************************************************ไฟล์โมดูล Bas_CmpZ80**************************************************
Option Explicit
Type FormStatusType
  Deleted As Boolean '
  DirtyFlag As Boolean 'save status writed on edit
End Type
Global InstrucDB As Database
Global InStruction As Recordset
Global WinStatusDB As QueryDef
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
Public Compileflag, Init, fstep As Boolean
Public Txtlist As String
Public InstionBYTE 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
Function GetDocIndex() As Integer 'Count File New
Dim i%, UB%
  UB% = UBound(TextEdit) 'Find dimension of an array
```

```
    'Find deleted index.
    For i% = 1 To UB%
      If fState(i%).Deleted Then
        GetDocIndex = i%
        fState(i%).Deleted = False
        Exit Function
      End If
    Next
    'If none have deleted,Create next index
    ReDim Preserve TextEdit(UB% + 1)
    ReDim Preserve fState(UB% + 1)
    InitVars (UB% + 1)
    fState(i%).Deleted = False
    GetDocIndex = UBound(TextEdit)
End Function
Sub InitVars(fIndex) 'Set status TextFile
    fState(fIndex).Deleted = True
    fState(fIndex).DirtyFlag = False
End Sub
Sub OpenFileDialog()
    Dim strFilename As String
    On Error Resume Next
    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
      strFilenameS = Main!CommonDialog.filename
      ASMFilename = Main!CommonDialog.FileTitle
      LoadTextFile strFilenameS
    End If
End Sub
Sub LoadTextFile(strFilenameS)
    Dim filenum%, fIndex%, txtS
    filenum% = FreeFile
    Open strFilenameS For Binary Access Read As filenum
    Screen.MousePointer = 11            'Set MousePointer to an hourglass
    Editor.Caption = UCaseS(strFilename)
    'Chang Form 's caption and display new text
    txtS = SpaceS(LOF(filenum%))
    Get filenum%, 1, txtS
    Editor.Text1.Text = txtS
    fState.DirtyFlag = False
    EditWin = True
    Editor.Show
ErrReturn01:
```

```
    Close #filenum
    Screen.MousePointer = 0
    Exit Sub
End Sub
Function InListMenuFile(strFilename$) As Integer
    Dim i As Integer
    For i = 1 To 4
        If MainForm!mnuFileLastOpen(i%).Caption = strFilename$ 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$) Then
        For i% = 4 To 2 Step -1
            'str = Mid(MainForm!mnuFileLastOpen(i% - 1).Caption, 3)
            MainForm!mnuFileLastOpen(i%).Caption = MainForm!mnuFileLastOpen(i% - 1).Caption
        Next i%
        MainForm!mnuFileLastOpen(1).Caption = strFilename$
    End If
    ShowFileFlag = False
    For i% = 1 To 4
        If MainForm!mnuFileLastOpen(i%).Caption <> "" Then
            MainForm!mnuFileLastOpen(i%).Visible = True
            ShowFileFlag = True
        Else
            MainForm!mnuFileLastOpen(i%).Visible = False
        End If
    Next i%
    If ShowFileFlag Then MainForm!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 ' เปิดไฟล์เพื่อทำการเขียนข้อมูล
    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:
    Select Case MsgBox(Error$, 64, App.Title)
        Case 3: Resume ErrReturn02
        Case 4: Resume
        Case 5: Resume Next
    End Select
End Sub
Sub EditCopy()
    Clipboard.SetText Main.ActiveForm.Text1.SelText
End Sub
Sub EditCut()
    Clipboard.SetText Main.ActiveForm.Text1.SelText
    Main.ActiveForm.Text1.SelText = ""
End Sub
Sub EditDelete()
    If Main.ActiveForm.Text1.SelLength = 0 Then
        Main.ActiveForm.Text1.SelLength = 1
        If Main.ActiveForm.Text1.SelLength <> "" Then
            'If cursor is on blank link,select 2 chars
            If Asc(Main.ActiveForm.Text1.SelText) = 13 Then Main.ActiveForm.Text1.SelLength = 2
        End If
    End If
    Main.ActiveForm.Text1.SelText = ""
End Sub
Sub EditPaste()
    If Clipboard.GetFormat(1) Then
        Main.ActiveForm.Text1.SelText = Clipboard.GetText()
    End If
End Sub
```

```
Sub ReplaceText()
   Dim start%, Offset%, pos%, FindLen%, ExitLoopflag As Boolean
   Dim strFindTxt$, strscr$
   ExitLoopflag = False
   Do
      If inFindCase% Then
         strFindTxt$ = strFind
         strsrc$ = MainForm.ActiveForm.Text1.Text
      Else
         strFindTxt$ = UCase(srtfind)
         str$ scr$ = UCase(MainForm.ActiveForm.Text1.SelStart)
      End If
      findlend% = Len(strFindTxt$)
      If (CurPos% = MainForm.ActiveForm.SelStart) Then
         Offset = 1
      Else
         Offset = 0
      End If
      start% = MainForm.ActiveForm.Text1.SelStart + Offset
      If inFindDir% = 1 Then
         pos% = InStr(start% + 1, strscr$, strFindTxt$)
      Else
         For pos% = start% - findlend% To 1 Step -1
            If MidS(strscr$, pos%, FindLen%) = strFindTxt$ Then Exit For
         Next
      End If
      'If string is found
      If pos% Then
         MainForm.ActiveForm.Text1.SelStart = pos% - 1
         MainForm.ActiveForm.Text1.SelLength = FindLen%
         Select Case MsgBox("Replaced this text?", 35, App.Title)
         Case 6 'yes
            Clipboard.SelText streplaced
            MainForm.ActiveForm.Text1.SelText = Clipboard.GetText()
            MainForm.ActiveForm.Text1.SelStart = pos% - 1
            MainForm.ActiveForm.Text1.SelLength = Len (strReplaced)
         Case 7 'no
         Case 2 'cancel
            ExitLoopflag = True
         End Select
      Else
         MsgBox "Search text not found.", 0, App.Title
         ExitLoopflag = True
      End If
      CurPos% = MainForm.ActiveForm.Text1.SelStart
   Loop Until ExitLoopflag
End Sub
```

```
Sub FindText()
    Dim start%, pos%, Offset%, FindLen%
    Dim strFindText As String, strscr As String
    If FirstFindFlag Then
        Offset% = 0
    Else
        If (CurPos% = MainForm.ActiveForm.Text1.SelStart) Then
            Offset% = 1
        Else
            Offset% = 0
        End If
    End If
    start% = MainForm.ActiveForm.Text1.SelStart + Offset%
    If inFindCase% Then
strFindTxt = strFind
        strscr$ = MainForm.ActiveForm.Text1.Text
    Else
        strFindTxt$ = UCase(strFind)
        str$ scr$ = UCase(MainForm.ActiveForm.Text1.SelStart)
    End If
    FindLen% = Len (strFindTxt$)
    If inFindDir% = 1 Then
            pos% = InStr(start% + 1, strscr$, strFindTxt$)
    Else
        For pos% = start% - findlend% To 1 Step -1
            If Mid$(strscr$, pos%, FindLen%) = strFindTxt$ Then Exit For
        Next
    End If
    'If string is found
        If pos% Then
            MainForm.ActiveForm.Text1.SelStart = pos% - 1
            MainForm.ActiveForm.Text1.SelLength = FindLen%
        Else
            MsgBox "Search text not found.", 0, App.Title
            ExitLoopflag = True
        End If
        CurPos% = MainForm.ActiveForm.Text1.SelStart
        FirstFindFlag = False
End Sub
Sub Compile_asm()
    Dim strPath, instrByte, CountCode, ADDRESS As String
    Dim filenum%, findex%, txt$, i, j, n As Integer
    Dim Inputdata, LSTchk As String, MyByte, bytestr, CountByte, MYADDRESS As Integer
    Dim strFilename$, LSTfilename$, OBJfilename$, PathDir$, Memfilename$
    Dim Cmp, RecNum
    Dim delaytime As Long
    Dim ErrorAsm, ErrorAsmCk As Integer
```

```
Dim a As String
    Compileflag = False
    Call INtitial
    Screen.MousePointer = 11   'ให้เมาส์เป็นรูปนาฬิกาทราย
      If Dir(("\\buf.asm")) <> "" Then
      Kill ("\\buf.asm")
    End If
   If Dir(("\\buf.lst")) <> "" Then
      Kill ("\\buf.lst")
    End If
    If Dir(("\\buf.obj")) <> "" Then
      Kill ("\\buf.obj")
    End If
  ChDir "\\"                                          'ไดเร็กทอรี่ปัจจุบันที่ a180Z.exe อยู่
     filenum% = FreeFile
 Open "\\BUF.ASM" For Output As filenum%
 txt$ = Editor.Text1.Text                             'อ่านข้อมูลจาก Editor
 Print #filenum%, txt$                                'เขียนข้อมูลลงในไฟล์
 Close #filenum%                                      'ปิด
    Cmp = Shell ("\\bin\A180z.EXE BUF.ASM", 0)        'รันคอมไพเลอร์ Z-80
   'เปิดไฟล์ List ที่เกิดจากการคอมไพเลอร์
    filenum% = FreeFile  'กำหนดหมายเลข Handdle ของไฟล์
     While Dir("\\buf.lst") = ""
      Wend
     While Dir("\\buf.obj") = ""
      Wend
    Open "\\buf.lst" For Input As filenum              'เปิดไฟล์
   ErrorAsm = 0
   Do while Not EOF (filenum)                          'ตรวจสอบตำแหน่งสิ้นสุดไฟล์
      Line Input #filenum, LSTchk                      'อ่านข้อมูลเป็นบรรทัด
      If Mid (LSTchk, 3, 21) = "***** Warning *****" Or Mid (LSTchk, 3, 21) = "****** Error ******" Then
        ErrorAsm = ErrorAsm + 1
      End If
   Loop
      If ErrorAsm > 0 Then
         ErrorAsmCk = MsgBox("โปรแกรมแอสแซมบลี้มีข้อผิดพลาด กรุณาแก้ไขโปรแกรมของท่าน", vbExclamation, "Error On Assembly Program")
      Else
        'ErrorAsmCk = MsgBox("..ผ่าน..สามารถสั่ง RUN ได้", vbOKOnly, "Compile Pass")
        Main.mnuStart.Enabled = True
        Main.mnustep.Enabled = True
        End If
   Close #filenum%
   filenum% = FreeFile                                 'กำหนดหมายเลข Handdle ของไฟล์
   Open "C:\Project\bin\buf.lst" For Binary Access Read As filenum   'เปิดไฟล์
   txt$ = Space$(LOF(filenum%))
```

```
Get filenum%, Seek (filenum%), txt$                    'อ่านข้อมูลจากไฟล์
    Screen.MousePointer = 0                            'กำหนดเมาส์ให้อยู่ในสภาวะปกติ
    Txtlist = txt$
    List.Text1.Text = Txtlist                          'แสดงใน Textbox
    Close #filenum%                                    'ปิดไฟล์ .LST ที่เปิด
    For j = 0 To 65535
      RAM (j) = 0
    Next
    filenum = FreeFile
    Open "\\buf.obj" For Input As filenum              'เปิดไฟล์ OBJ
    Do While Not EOF (filenum)                         'ตรวจสอบตำแหน่งสิ้นสุดไฟล์
      Line Input #filenum, Inputdata                   'อ่านข้อมูลเป็นบรรทัด
      CountCode = Mid (Inputdata, 2, 2)
      MyByte = Val ("&H" & CountCode)                  'จำนวน Byte ของคำสั่ง แปลงจาก Text เป็นตัวเลข
      'ตรวจสอบจำนวน Byte ของคำสั่ง
      If MyByte <> 0 Then                              'ถ้ามีจำนวนคำสั่งไม่เป็นศูนย์
        InstrCounter = InstrCounter + MyByte
        bytestr = 10
        For j = 0 To (MyByte - 1) 'อ่านเท่ากับจำนวนของคำสั่ง
          RAM (j) = Val (" &H" & Mid (Inputdata, bytestr, 2)) 'คำสั่ง 1 Byte
          bytestr = bytestr + 2
        Next
      End If
  Loop
  Close filenum                                        'ปิดไฟล์ OBJ
  Compileflag = True
End Sub
Sub Run ()
  Dim j, n, num, filenum%, bytestr, InstrOrder As Long
  Dim OBJfilename, Inputdata, opcode, instrByte, CountCode As String
  Dim varBookmark As Variant
  Dim MyByte As Byte
  Dim MyInsByte As Long
  If Compileflag = False Then Call Compile_asm
  Call INtitial
  'กำหนดค่าเริ่มต้น
  Inputdata = ""
  opcode = ""
  instrByte = ""
  bytestr = 10                                         'อินเด็กซ์ในการชี้ตำแหน่งของ Code ไฟล์ .OBJ
  j = 1                                                'ค่าเริ่มต้นในการนับจำนวนคำสั่ง
  InstrOrder = 1                                       'กำหนดของลำดับไบต์ของคำสั่งเป็นลำดับที่ 1
    Do While j <= 65535
      opcode = opcode & RightString(2, "0") & Hex(RAM(j - 1)), 2)   'นำข้อมูลใน Ram มาเปรียบเทียบ Opcode
      InstrOrder = InstrOrder + 1
       SourceN = .Fields("Source")
        SOURCE = SourceAddr(.Fields("Source")) 'อ่านค่าจากรีจิสเตอร์ต้นทางของคำสั่ง
```

```
    DEST = .Fields("Dest")
            Call SelectINSTR(.Fields("Instruction"))                    'เลือกว่าเป็นการใช้คำสั่งใด และ ทำงานตามคำสั่ง
            SpecialRegis(3) = SpecialRegis(3) + MyInsByte
            Regis.LbGRegis(3) = "PC = " & Right(String(2, "3") & Hex(SpecialRegis(3)), 4) & "H"
            opcode = ""
            InstrOrder = 1
          End If
          j = j + 1
        Loop
    End With
    Compileflag = False
End Sub
Sub RunSTEP()
    Dim j, n, num, filenum%, bytestr, InstrOrder As Integer
    Dim OBJfilename, Inputdata, opcode, instrByte, CountCode As String
    Dim varBookmark As Variant
    Dim MyByte As Byte
    Dim MyInsByte As Integer
    Dim match As Boolean
    fstep = False
    If Compileflag = False Then Call Compile_asm
    If Init = False Then
    Call INtitial
    End If
    'กำหนดค่าเริ่มต้น
    Inputdata = ""
    opcode = ""
    instrByte = ""
    bytestr = 10                                                        ' อินเด็กซ์ในการชี้ตำแหน่งของ Code ไฟล์ .OBJ
    j = Val("&H" & SpecialRegis(3)) + 1                                 ' ค่าเริ่มต้นในการนับจำนวนคำสั่ง
    InstrOrder = 1                                                      ' กำหนดของลำดับไบต์ของคำสั่งเป็นลำดับที่ 1
      Do Until fstep = True
            Call SelectINSTR(.Fields("Instruction"))                    ' เลือกว่าเป็นการใช้คำสั่งใด และ ทำงานตามคำสั่ง
            SpecialRegis(3) = Hex(Val("&H" & SpecialRegis(3)) + MyInsByte)
            Select Case Len(SpecialRegis(3))
              Case 1
                    SpecialRegis(3) = "000" & SpecialRegis(3)
              Case 2
                    SpecialRegis(3) = "00" & SpecialRegis(3)
              Case 3
                    SpecialRegis(3) = "0" & SpecialRegis(3)
            End Select
            Main.LbSpcRegi(3).Caption = "PC = " & SpecialRegis(3) & "H"
            opcode = ""
            InstrOrder = 1
            Match = True
            j = j + 1
```

```
    Loop
    fstep = True
    Loop
  End With
  If Val ("&H" & SpecialRegis(3)) = InstrCounter Then
    Main.mnustep.Enabled = False
  End If
End Sub

Public Sub Delay1 (HowLong As Integer)
    Dim F1 As Double
    Dim F2 As Long
    If HowLong > 20 Then HowLong = 20
      For F1 = 0 To HowLong
        For F2 = 0 To 300000
        Next
        Next
End Sub
 Public Sub Delay (HowLong As Integer)
    If HowLong > 40 Then HowLong = 40
    Dim F1 As Double
    Dim F2 As Long
      For F1 = 0 To HowLong
        For F2 = 0 To 300000
        Next
      Next
End Sub
Sub Runtum()
    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 MyInsByte, pc As Long

    If Compileflag = False Then Call Compile_asm 'Compile_asm ' ทำการคอมไพล์ก่อนถ้ายังไม่คอมไพล์
    Call INtitial
    pc = SpecialRegis(3)
    ' เฟทคำสั่งมาเก็บใน Instruction รีจิสเตอร์
    opcode = Hex(RAM(pc))
    ' ถอดรหัสคำสั่ง
    Select Case opcode
      'คำสั่ง 1 ไบต์ LOAD 8 บิต
          Case "7F", "78", "79", "7A", "7B", "7C", "7D", "7E": Call LdA_Regis(opcode) ' LD A,REGIS
          Case "47", "40", "41", "42", "43", "44", "45", "46": Call LdB_Regis(opcode)  ' LD B,REGIS
          Case "4F", "48", "49", "4A", "4B", "4C", "4D", "4E": Call LdC_Regis(opcode) 'LD C,REGIS
          Case "57", "50", "51", "52", "53", "54", "55", "56": Call LdD_Regis(opcode)  ' LD D,REGIS
          Case "5F", "58", "59", "5A", "5B", "5C", "5D", "5E": Call LdE_Regis(opcode) 'LD E,REGIS
```

```
        Case "67", "60", "61", "62", "63", "64", "65", "66": Call LdH_Regis(opcode)   ' LD H,REGIS
        Case "6F", "68", "69", "6A", "6B", "6C", "6D", "6E": Call LdL_Regis(opcode) 'LD L,REGIS
        Case "77", "70", "71", "72", "73", "74", "75": Call LdaddHL_Regis(opcode)     'LD(HL),REGIS
'คำสั่ง Load 16 bit 1 ไบต์
        Case "0A", "1A", "02", "12": Call Ld16_1B(opcode) 'LD A, (BC), LD A, (DE), LD (BC), A, LD (DE), A
'กลุ่มคณิตศาสตร์และลอจิก 8บิต 1 ไบต์
    Case "87", "80", "81", "82", "83", "84", "85", "86": Call ADDA_REGIS(opcode)  'ADD A,REGIS
    Case "8F", "88", "89", "8A", "8B", "8C", "8D", "8E": Call ADCA_REGIS(opcode) ' ADC A,REGIS
    Case "97", "90", "91", "92", "93", "94", "95", "96": Call SUB_REGIS(opcode)   ' SUB
    Case "9F", "98", "99", "9A", "9B", "9C", "9D", "9E": Call SBCA_REGIS(opcode) 'SBC A,REGIS
    Case "A7", "A0", "A1", "A2", "A3", "A4", "A5", "A6": Call AND_REGIS(opcode)   ' AND
    Case "AF", "A8", "A9", "AA", "AB", "AC", "AD", "AE": Call XOR_REGIS(opcode) 'XOR
    Case "B7", "B0", "B1", "B2", "B3", "B4", "B5", "B6": Call OR_REGIS(opcode) ' OR
    Case "BF", "B8", "B9", "BA", "BB", "BC", "BD", "BE": Call COMPARE(opcode) 'CP
    Case "27": Call DAA 'DAA
    Case "2F": Call CPL 'CPL
    Case "3F": Call CCF  'CCF
    Case "37": Call SCF 'SCF
  ' กลุ่มคณิตศาสตร์และลอจิก 16 บิต 1 ไบต์
    Case "09", "19", "29", "39": Call ADDHL_Regis(opcode)  'ADD HL, REGIS 16
  ' กลุ่มคำสั่งเลื่อนและหมุน 1 ไบต์'RLCA,RRCA,RLA,RRA
    Case "07": Call RLCA 'RLCA
    Case "0f": Call RRCA 'RRCA
    Case "17": Call RLA 'RLA
    Case "1f": Call RRA 'RRA
  ' กลุ่มคำสั่งเพิ่ม-ลดค่า
    Case "3C", "04", "0C", "14", "1C", "24", "2C", "03", "13", "23", "34": Call INC (opcode) ' INC
    Case "3D", "05", "0D", "15", "1D", "25", "2D", "0B", "1B", "2B", "35": Call DEC (opcode) 'DEC
'กลุ่มคำสั่งการเก็บและเลี่ยนข้อมูล
    Case "F5", "C5", "D5", "E5" 'push
    Case "F1", "C1", "D1", "E1" 'pop
    Case "08" 'EX AF, AF'
    Case "EB" 'EX DE, HL
    Case "E3" 'EX (SP), HL
    Case "D9" 'EXX
    Case ""
    Case ""
End Select
' เพิ่มค่า Program counter
pc = pc + InstionBYTE
SpecialRegis(3) = pc
Unload Regis
Unload Memo
Load Regis
Load Memo
Regis.Refresh
Memo.Refresh
```

```
End Sub
Public Sub LdA_Regis(ByVal opcode As String)
   Dim HL As Long
   HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
   Select Case opcode "7F", "78", "79", "7A", "7B", "7C", "7D", "7E" '
          Case "7F": GeneralRegis(0) = GeneralRegis(0) 'A
          Case "78": GeneralRegis(0) = GeneralRegis(1) 'B
           Case "79": GeneralRegis(0) = GeneralRegis(2) 'C
           Case "7A": GeneralRegis(0) = GeneralRegis(3) 'D
           Case "7B": GeneralRegis(0) = GeneralRegis(4) 'E
           Case "7C": GeneralRegis(0) = GeneralRegis(6) 'H
           Case "7D": GeneralRegis(0) = GeneralRegis(7) 'L
           Case "7E": GeneralRegis(0) = RAM(HL) '(HL)
     End Select
     InstionBYTE = 1
End Sub
Public Sub LdB_Regis(ByVal opcode As String)
   Dim HL As Long
   HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
   Select Case opcode "47", "40", "41", "42", "43", "44", "45", "46"
           Case "47": GeneralRegis(1) = GeneralRegis(0) 'A
           Case "40": GeneralRegis(1) = GeneralRegis(1) 'B
           Case "41": GeneralRegis(1) = GeneralRegis(2) 'C
           Case "42": GeneralRegis(1) = GeneralRegis(3) 'D
           Case "43": GeneralRegis(1) = GeneralRegis(4) 'E
           Case "44": GeneralRegis(1) = GeneralRegis(6) 'H
           Case "45": GeneralRegis(1) = GeneralRegis(7) 'L
           Case "46": GeneralRegis(1) = RAM(HL) '(HL)
    End Select
    InstionBYTE = 1
End Sub
Public Sub LdC_Regis(ByVal opcode As String)
   Dim HL As Long
   HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
        Select Case opcode " 4F", "48", "49", "4A", "4B", "4C", "4D", "4E"
           Case "4F": GeneralRegis(2) = GeneralRegis(0) 'A
 (         Case "48": GeneralRegis(2) = GeneralRegis(1) 'B
           Case "49": GeneralRegis(2) = GeneralRegis(2) 'C
           Case "4A": GeneralRegis(2) = GeneralRegis(3) 'D
           Case "4B": GeneralRegis(2) = GeneralRegis(4) 'E
           Case "4C": GeneralRegis(2) = GeneralRegis(6) 'H
           Case "4D": GeneralRegis(2) = GeneralRegis(7) 'L
           Case "4E": GeneralRegis(2) = RAM(HL) '(HL)
    End Select
    InstionBYTE = 1
End Sub
Public Sub LdD_Regis(ByVal opcode As String)
```

```
    Dim HL As Long
    HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
    Select Case opcode ' "57", "50", "51", "52", "53", "54", "55", "56"
            Case "57": GeneralRegis(3) = GeneralRegis(0) 'A
            Case "50": GeneralRegis(3) = GeneralRegis(1) 'B
            Case "51": GeneralRegis(3) = GeneralRegis(2) 'C
            Case "52": GeneralRegis(3) = GeneralRegis(3) 'D
            Case "53": GeneralRegis(3) = GeneralRegis(4) 'E
            Case "54": GeneralRegis(3) = GeneralRegis(6) 'H
            Case "55": GeneralRegis(3) = GeneralRegis(7) 'L
            Case "56": GeneralRegis(3) = RAM(HL) '(HL)
    End Select
    InstionBYTE = 1
End Sub
Public Sub LdE_Regis(ByVal opcode As String)
    Dim HL As Long
    HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
    Select Case opcode ""5F", "58", "59", "5A", "5B", "5C", "5D", "5E"
            Case "5F": GeneralRegis(4) = GeneralRegis(0) 'A
            Case "58": GeneralRegis(4) = GeneralRegis(1) 'B
            Case "59": GeneralRegis(4) = GeneralRegis(2) 'C
            Case "5A": GeneralRegis(4) = GeneralRegis(3) 'D
            Case "5B": GeneralRegis(4) = GeneralRegis(4) 'E
            Case "5C": GeneralRegis(4) = GeneralRegis(6) 'H
            Case "5D": GeneralRegis(4) = GeneralRegis(7) 'L
            Case "5E": GeneralRegis(4) = RAM(HL) '(HL)
    End Select
    InstionBYTE = 1
End Sub
Public Sub LdH_Regis(ByVal opcode As String)
    Dim HL As Long
    HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
    Select Case opcode ' "67", "60", "61", "62", "63", "64", "65", "66"
            Case "67": GeneralRegis(6) = GeneralRegis(0) 'A
            Case "60": GeneralRegis(6) = GeneralRegis(1) 'B
            Case "61": GeneralRegis(6) = GeneralRegis(2) 'C
            Case "62": GeneralRegis(6) = GeneralRegis(3) 'D
            Case "63": GeneralRegis(6) = GeneralRegis(4) 'E
            Case "64": GeneralRegis(6) = GeneralRegis(6) 'H
            Case "65": GeneralRegis(6) = GeneralRegis(7) 'L
            Case "66": GeneralRegis(6) = RAM(HL) '(HL)
    End Select
    InstionBYTE = 1
End Sub
Public Sub LdL_Regis(ByVal opcode As String)
    Dim HL As Long
    HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
```

```
  Select Case opcode ""6F", "68", "69", "6A", "6B", "6C", "6D", "6E"
        Case "6F": GeneralRegis(7) = GeneralRegis(0) 'A
        Case "68": GeneralRegis(7) = GeneralRegis(1) 'B
        Case "69": GeneralRegis(7) = GeneralRegis(2) 'C
        Case "6A": GeneralRegis(7) = GeneralRegis(3) 'D
        Case "6B": GeneralRegis(7) = GeneralRegis(4) 'E
        Case "6C": GeneralRegis(7) = GeneralRegis(6) 'H
        Case "6D": GeneralRegis(7) = GeneralRegis(7) 'L
        Case "6E": GeneralRegis(7) = RAM(HL) '(HL)
  End Select
  InstionBYTE = 1
End Sub
Public Sub LdaddHL_Regis(ByVal opcode As String)
  Dim HL As Long
  HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
  Select Case opcode ""77", "70", "71", "72", "73", "74", "75"
        Case "77": RAM (HL) = GeneralRegis(0)  'A
        Case "70": RAM (HL) = GeneralRegis(1) 'B
        Case "71": RAM (HL) = GeneralRegis(2) 'C
        Case "72": RAM (HL) = GeneralRegis(3) 'D
        Case "73": RAM (HL) = GeneralRegis(4) 'E
        Case "74": RAM (HL) = GeneralRegis(6) 'H
        Case "75": RAM (HL) = GeneralRegis(7) 'L
  End Select
  InstionBYTE = 1
End Sub
Public Sub Ld16_1B(ByVal opcode As String)
  Dim BC As Long
  Dim DE As Long
  BC = Val (Right (String (2, "0") & Hex (GeneralRegis(1)), 2) & Right(String(2, "0") & Hex(GeneralRegis(2)), 2))
  DE = Val (Right (String (2, "0") & Hex (GeneralRegis(3)), 2) & Right(String(2, "0") & Hex(GeneralRegis(4)), 2))
  Select Case opcode ' "0A", "1A", "02", "12" '
        Case "0A": GeneralRegis(0) = RAM(BC) 'A,(BC)
        Case "1A": GeneralRegis(0) = RAM(DE) 'A,(DE)
        Case "02": RAM (BC) = GeneralRegis(0) '(BC),A
        Case "12": RAM (DE) = GeneralRegis(0) '(DE),A
  End Select
  InstionBYTE = 1
End Sub
Public Sub ADDA_REGIS (ByVal opcode As String) ""87", "80", "81", "82", "83", "84", "85", "86"  'ADD A, REGIS
 Dim HL As Long
  HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
  Select Case opcode
        Case "87": GeneralRegis(0) = GeneralRegis(0) + GeneralRegis(0) 'ADD A,A
        Case "80": GeneralRegis(0) = GeneralRegis(0) + GeneralRegis(1) 'ADD A,B
        Case "81": GeneralRegis(0) = GeneralRegis(0) + GeneralRegis(2) 'ADD A,C
        Case "82": GeneralRegis(0) = GeneralRegis(0) + GeneralRegis(3) 'ADD A,D
```

```
            Case "83": GeneralRegis(0) = GeneralRegis(0) + GeneralRegis(4) 'ADD A,E
            Case "84": GeneralRegis(0) = GeneralRegis(0) + GeneralRegis(6) 'ADD A,H
            Case "85": GeneralRegis(0) = GeneralRegis(0) + GeneralRegis(7) 'ADD A,L
            Case "86": GeneralRegis(0) = GeneralRegis(0) + RAM(HL) 'ADD A,(HL)
    End Select
    Call arismatic8flagcheck (0)
End Sub
Public Sub ADCA_REGIS (ByVal opcode As String) ' "8F", "88", "89", "8A", "8B", "8C", "8D", "8E" ' ADC A,REGIS
 Dim HL As Long, C As Long
    If (CARRYFLAG = True) Then
     C = 1
     Else
     C = 0
    End If
    HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
    Select Case opcode
            Case "8F": GeneralRegis(0) = GeneralRegis(0) + GeneralRegis(0) + C 'ADC A,A
            Case "88": GeneralRegis(0) = GeneralRegis(0) + GeneralRegis(1) + C 'ADC A,B
            Case "89": GeneralRegis(0) = GeneralRegis(0) + GeneralRegis(2) + C 'ADC A,C
            Case "8A": GeneralRegis(0) = GeneralRegis(0) + GeneralRegis(3) + C 'ADC A,D
            Case "8B": GeneralRegis(0) = GeneralRegis(0) + GeneralRegis(4) + C 'ADC A,E
            Case "8C": GeneralRegis(0) = GeneralRegis(0) + GeneralRegis(6) + C 'ADC A,H
            Case "8D": GeneralRegis(0) = GeneralRegis(0) + GeneralRegis(7) + C 'ADC A,L
            Case "8E": GeneralRegis(0) = GeneralRegis(0) + RAM(HL) + C 'ADD A,(HL)
    End Select
    Call arismatic8flagcheck (0)
End Sub

Public Sub SUB_REGIS (ByVal opcode As String) '"97", "90", "91", "92", "93", "94", "95", "96"    ' SUB
 Dim HL As Long
    HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
    Select Case opcode
            Case "87": GeneralRegis(0) = GeneralRegis(0) - GeneralRegis(0) " SUB A,A
            Case "80": GeneralRegis(0) = GeneralRegis(0) - GeneralRegis(1) " SUB A,B
            Case "81": GeneralRegis(0) = GeneralRegis(0) - GeneralRegis(2) " SUBA,C
            Case "82": GeneralRegis(0) = GeneralRegis(0) - GeneralRegis(3) " SUBA,D
            Case "83": GeneralRegis(0) = GeneralRegis(0) - GeneralRegis(4) " SUBA,E
            Case "84": GeneralRegis(0) = GeneralRegis(0) - GeneralRegis(6) " SUBA,H
            Case "85": GeneralRegis(0) = GeneralRegis(0) - GeneralRegis(7) " SUBA,L
            Case "86": GeneralRegis(0) = GeneralRegis(0) - RAM(HL) " SUB A,(HL)
    End Select
    Call arismatic8flagcheck (1)
End Sub
Public Sub SBCA_REGIS (ByVal opcode As String) ' "9F", "98", "99", "9A", "9B", "9C", "9D", "9E"  'SBC A, REGIS
 Dim HL As Long, C As Long
    If (CARRYFLAG = True) Then
     C = 1
```

```
    Else
    C = 0
  End If
  HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
  Select Case opcode
        Case "8F": GeneralRegis(0) = GeneralRegis(0) - GeneralRegis(0) - C 'SBCA,A
        Case "88": GeneralRegis(0) = GeneralRegis(0) - GeneralRegis(1) - C 'SBC A,B
        Case "89": GeneralRegis(0) = GeneralRegis(0) - GeneralRegis(2) - C 'SBC A,C
        Case "8A": GeneralRegis(0) = GeneralRegis(0) - GeneralRegis(3) - C 'SBC A,D
        Case "8B": GeneralRegis(0) = GeneralRegis(0) - GeneralRegis(4) - C 'SBCA,E
        Case "8C": GeneralRegis(0) = GeneralRegis(0) - GeneralRegis(6) - C 'SBC A,H
        Case "8D": GeneralRegis(0) = GeneralRegis(0) - GeneralRegis(7) - C 'SBC A,L
        Case "8E": GeneralRegis(0) = GeneralRegis(0) - RAM(HL) - C 'SBCA,(HL)
  End Select
  Call arismatic8flagcheck (1)
End Sub
' ชุดคำสั่งคำลอง 8 บิต   ** กระทำทางตรรกะ **
Public Sub AND_REGIS (ByVal opcode As String)  ' "A7", "A0", "A1", "A2", "A3", "A4", "A5", "A6"   'AND A,REGIS
  Dim HL As Long
  HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
  Select Case opcode
        Case "A7": GeneralRegis(0) = GeneralRegis(0) And GeneralRegis(0) 'AND A,A
        Case "A0": GeneralRegis(0) = GeneralRegis(0) And GeneralRegis(1) 'AND A,B
        Case "A1": GeneralRegis(0) = GeneralRegis(0) And GeneralRegis(2) 'AND A,C
        Case "A2": GeneralRegis(0) = GeneralRegis(0) And GeneralRegis(3) 'AND A,D
        Case "A3": GeneralRegis(0) = GeneralRegis(0) And GeneralRegis(4) 'AND A,E
        Case "A4": GeneralRegis(0) = GeneralRegis(0) And GeneralRegis(6) 'AND A,H
        Case "A5": GeneralRegis(0) = GeneralRegis(0) And GeneralRegis(7) 'AND A,L
        Case "A6": GeneralRegis(0) = GeneralRegis(0) And RAM(HL) 'AND A,(HL)
  End select
  Call SETBOOLEEN (GeneralRegis(0))
End Sub
Public Sub XOR_REGIS (ByVal opcode As String)  ' Case "AF", "A8", "A9", "AA", "AB", "AC", "AD", "AE"  'XOR'"
 Dim HL As Long
  HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
      Select Case opcode
        Case "AF": GeneralRegis(0) = GeneralRegis(0) Xor GeneralRegis(0) 'XOR A,A
        Case "A8": GeneralRegis(0) = GeneralRegis(0) Xor GeneralRegis(1) 'XOR A,B
        Case "A9": GeneralRegis(0) = GeneralRegis(0) Xor GeneralRegis(2) 'XORA,C
        Case "AA": GeneralRegis(0) = GeneralRegis(0) Xor GeneralRegis(3) 'XOR A,D
        Case "AB": GeneralRegis(0) = GeneralRegis(0) Xor GeneralRegis(4) 'XOR A,E
        Case "AC": GeneralRegis(0) = GeneralRegis(0) Xor GeneralRegis(6) 'XOR A,H
        Case "AD": GeneralRegis(0) = GeneralRegis(0) Xor GeneralRegis(7) 'XOR A,L
        Case "AE": GeneralRegis(0) = GeneralRegis(0) Xor RAM(HL) 'XOR A,(HL)
  End Select
  Call SETBOOLEEN (GeneralRegis(0))
End Sub
```

```
Public Sub OR_REGIS (ByVal opcode As String) 'Case "B7", "B0", "B1", "B2", "B3", "B4", "B5", "B6"   ' OR
 Dim HL As Long
   HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
        Select Case opcode
              Case "B7": GeneralRegis(0) = GeneralRegis(0) Or GeneralRegis(0) 'OR A,A
              Case "B0": GeneralRegis(0) = GeneralRegis(0) Or GeneralRegis(1) 'OR A,B
              Case "B1": GeneralRegis(0) = GeneralRegis(0) Or GeneralRegis(2) 'ORA,C
              Case "B2": GeneralRegis(0) = GeneralRegis(0) Or GeneralRegis(3) 'OR A,D
              Case "B3": GeneralRegis(0) = GeneralRegis(0) Or GeneralRegis(4) 'OR A,E
              Case "B4": GeneralRegis(0) = GeneralRegis(0) Or GeneralRegis(6) 'OR A,H
              Case "B5": GeneralRegis(0) = GeneralRegis(0) Or GeneralRegis(7) 'OR A,L
              Case "B6": GeneralRegis(0) = GeneralRegis(0) Or RAM(HL) 'OR A,(HL)
    End Select
    Call SETBOOLEEN (GeneralRegis(0))
End Sub
Public Function COMPARE (ByVal opcode As String) ' Case "BF", "B8", "B9", "BA", "BB", "BC", "BD", "BE"  'CP
    Dim ZERO As Long
    Dim HL As Long
    HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
    Select Case opcode
              Case "BF": ZERO = GeneralRegis(0) And GeneralRegis(0) 'CP A,A
              Case "B8": ZERO = GeneralRegis(0) And GeneralRegis(1) 'CP A,B
              Case "B9": ZERO = GeneralRegis(0) And GeneralRegis(2) 'CP A,C
              Case "BA": ZERO = GeneralRegis(0) And GeneralRegis(3) 'CP A,D
              Case "BB": ZERO = GeneralRegis(0) And GeneralRegis(4) 'CP A,E
              Case "BC": ZERO = GeneralRegis(0) And GeneralRegis(6) 'CP A,H
              Case "BD": ZERO = GeneralRegis(0) And GeneralRegis(7) 'CP A,L
              Case "BE": ZERO = GeneralRegis(0) And RAM(HL) 'CP A,(HL)
    If (ZERO = GeneralRegis(0)) Then
       ZEROFLAG = True
    Else
       ZEROFLAG = False
    End If
    NEGATIVEFLAG = True
    Flag = FLAGSET
End Function


Public Function DAA ()
    Dim ah, al, H As Byte
    Dim BUF, BUF1, hexA As String
          hexA = Right("00" & Hex(GeneralRegis(0)), 2)
          Al = Val ("&H" & Right (hexA, 1))
            If (al > 9 Or HALFCARRYFLAG = True) Then al = al + 6
            If al > 15 Then
              HALFCARRYFLAG = True
              H = 1
            Else
              HALFCARRYFLAG = True
```

```
            H = 0
          End If
          BUF = Right (Hex (al), 1)
          Ah = Val ("&H" & Left (hexA, 1))
          Ah = ah + H
          If (ah > 9) Then ah = ah + 6
            If ah > 15 Then
              CARRYFLAG = True
            Else
              CARRYFLAG = False
            End If
          BUF1 = Right (Hex (ah), 1)
          BUF = BUF1 & BUF
          If Val (BUF) = 0 Then
            ZEROFLAG = True
          Else
            ZEROFLAG = False
          End If
           If vbrcGetBitB(Val(BUF), 7) Then
            NEGATIVEFLAG = True
          Else
            NEGATIVEFLAG = False
          End If

          GeneralRegis(0) = Val(BUF)
          Flag = FLAGSET
End Function
Public Sub CPL ()
 Dim a As Byte
     A = GeneralRegis(0)
     HALFCARRYFLAG = True
     NEGATIVEFLAG = True
     Flag = FLAGSET
     vbrcToggleBitB a, 0
     vbrcToggleBitB a, 1
     vbrcToggleBitB a, 2
     vbrcToggleBitB a, 3
     vbrcToggleBitB a, 4
     vbrcToggleBitB a, 5
     vbrcToggleBitB a, 6
     vbrcToggleBitB a, 7
     GeneralRegis(0) = a
End Sub
 Public Sub CCF ()
  NEGATIVEFLAG = False
  If CARRYFLAG = False Then
    CARRYFLAG = True
  Else
```

```
    CARRYFLAG = False
  End If
  Flag = FLAGSET
  End Sub
Public Sub SCF ()
  NEGATIVEFLAG = False
  HALFCARRYFLAG = False
  CARRYFLAG = True
  Flag = FLAGSET
  End Sub
  Public Sub ADDHL_Regis(ByVal opcode As String) 'Case "09", "19", "29", "39" 'ADD HL, REGIS 16
   Dim HL As Long, BC As Long, DE As Long, SP As Long
   Dim HLstr As String
  HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
  BC = Val (Right (String (2, "0") & Hex (GeneralRegis(1)), 2) & Right(String(2, "0") & Hex(GeneralRegis(2)), 2))
  DE = Val (Right (String (2, "0") & Hex (GeneralRegis(3)), 2) & Right(String(2, "0") & Hex(GeneralRegis(4)), 2))
  SP = SpecialRegis(2)
       Select Case opcode
         Case "09": HL = HL + BC
         Case "19": HL = HL + DE
         Case "29": HL = HL + HL
         Case "39": HL = HL + SP
       End Select
       HLstr = Right ("000" & Hex (HL), 4)
       GeneralRegis(6) = Right(HLstr, 2) 'H
       GeneralRegis(7) = Left(HLstr, 2) 'L
       NEGATIVEFLAG = False
      If (HL > 65535) Then
           CARRYFLAG = True
       Else
           CARRYFLAG = False
       End If
       Flag = FLAGSET
  End Sub
 'กลุ่มคำสั่งเลื่อนและหมุน 1 ไบต์'RLCA.RRCA.RLA.RRA
      ' Case "07", "0f", "17", "1f"
Public Function RLCA () '07
  A = GeneralRegis(0)
  CARRYFLAG = vbrcGetBitB(a, 7)
  Call vbrcROLB(a, 1)
  GeneralRegis(0) = a
  NEGATIVEFLAG = False
  HALFCARRYFLAG = False
  Flag = FLAGSET
End Function
```

```
Public Function RRCA () '0F
   A = GeneralRegis(0)
   CARRYFLAG = vbrcGetBitB(a, 7)
   Call vbrcRORB(a, 1)
   GeneralRegis(0) = a
   NEGATIVEFLAG = False
   HALFCARRYFLAG = False
   Flag = FLAGSET
End Function
Public Function RLA ()
   Dim a, C (8), b (8), n As Integer
   Dim f As Integer
   A = GeneralRegis(0)
   If a > 255 Then
   A = a - 256
   Else
   End If
   If CARRYFLAG = True Then
   C (8) = 1
   Else
   C (8) = 0
   End If
   C (0) = vbrcGetBitB(a, 0)
   C (1) = vbrcGetBitB(a, 1)
   C (2) = vbrcGetBitB(a, 2)
   C (3) = vbrcGetBitB(a, 3)
   C (4) = vbrcGetBitB(a, 4)
   C (5) = vbrcGetBitB(a, 5)
   C (6) = vbrcGetBitB(a, 6)
   C (7) = vbrcGetBitB(a, 7)
   B (0) = C (0)
   B (1) = C (1)
   B (2) = C (2)
   B (3) = C (3)
   B (4) = C (4)
   B (5) = C (5)
   B (6) = C (6)
   B (7) = C (7)
   B (8) = C (8)
   'หมุนซ้าย'
   C (0) = B (8)
   C (1) = B (0)
   C (2) = B (1)
   C (3) = B (2)
   C (4) = b(3)
   C (5) = B (4)
   C (6) = B (5)
```

```
C (7) = B (6)
CARRYFLAG = B (7)
'CONVERT BINARY TO INTEGER'
A = 0
For n = 0 To 7
    A = a + ((2 ^ n) * C (n))
Next
NEGATIVEFLAG = False
HALFCARRYFLAG = False
Flag = FLAGSET
   GeneralRegis(0) = a
End Function
Public Function RRA ()
Dim a, C (8), b (8), n As Integer
Dim f As Integer
a = GeneralRegis(0)
If a < 0 Then
    a = a + 256
Else
End If
If CARRYFLAG = True Then
   C (8) = 1
Else
   C (8) = 0
End If
C (0) = vbrcGetBitB(a, 0)
C (1) = vbrcGetBitB(a, 1)
C (2) = vbrcGetBitB(a, 2)
C (3) = vbrcGetBitB(a, 3)
C (4) = vbrcGetBitB(a, 4)
C (5) = vbrcGetBitB(a, 5)
C (6) = vbrcGetBitB(a, 6)
C (7) = vbrcGetBitB(a, 7)
B (0) = C (0)
B (1) = C (1)
B (2) = C (2)
B (3) = C (3)
B (4) = C (4)
B (5) = C (5)
B (6) = C (6)
B (7) = C (7)
B (8) = C (8)
'หมุนบิต
C(0) = b(1)
C (1) = b (2)
C (2) = b (3)
C (3) = b (4)
```

```
C (4) = b (5)
C (5) = b (6)
C (6) = b (7)
C (7) = b (8)
CARRYFLAG = b (0)
'CONVERT BINARY TO INTEGER'
a = 0
For n = 0 To 7
   a = a + ((2 ^ n) * C(n))
Next
NEGATIVEFLAG = False
HALFCARRYFLAG = False
Flag = FLAGSET
   GeneralRegis(0) = a
End Function
Public Function INC(ByVal opcode As String) 'Case "3C", "04", "0C", "14", "1C", "24", "2C", "03", "13", "23", "34" 'INC
'Case "3D", "05", "0D", "15", "1D", "25", "2D", "0B", "1B", "2B", "35" 'DEC
Dim HL As Long, BC As Long, DE As Long, SP As Long
Dim HLstr As String
Dim BUF As Long
Dim BCstr As String
HL = Val (Right (String (2, "0") & Hex (GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
BC = Val (Right (String (2, "0") & Hex (GeneralRegis(1)), 2) & Right(String(2, "0") & Hex(GeneralRegis(2)), 2))
DE = Val (Right (String (2, "0") & Hex (GeneralRegis(3)), 2) & Right(String(2, "0") & Hex(GeneralRegis(4)), 2))
Select Case opcode
   Case "3C" 'INC A
         BUF = GeneralRegis(0) + 1
         If BUF > 255 Then
            BUF = BUF - 256
            PARITYOVERFLOW = True
           Else
            PARITYOVERFLOW = False
            End If
          GeneralRegis(0) = BUF
   Case "04" 'INC B
          BUF = GeneralRegis(1) + 1
         If BUF > 255 Then
            BUF = BUF - 256
            PARITYOVERFLOW = True
           Else
            PARITYOVERFLOW = False
            End If
          GeneralRegis(1) = BUF
   Case "0C" 'INC C
          BUF = GeneralRegis(2) + 1
         If BUF > 255 Then
            BUF = BUF - 256
```

```
            PARITYOVERFLOW = True
            Else
            PARITYOVERFLOW = False
            End If
          GeneralRegis(2) = BUF
Case "14" 'INC D
          BUF = GeneralRegis(3) + 1
          If BUF > 255 Then
            BUF = BUF - 256
            PARITYOVERFLOW = True
            Else
            PARITYOVERFLOW = False
            End If
          GeneralRegis(3) = BUF
Case "1C" 'INC E
          BUF = GeneralRegis(4) + 1
          If BUF > 255 Then
            BUF = BUF - 256
            PARITYOVERFLOW = True
          Else
            PARITYOVERFLOW = False
          End If
          GeneralRegis(4) = BUF
Case "24" 'INC H
          BUF = GeneralRegis(6) + 1
          If BUF > 255 Then
            BUF = BUF - 256
            PARITYOVERFLOW = True
          Else
            PARITYOVERFLOW = False
          End If
          GeneralRegis(6) = BUF
Case "2C" 'INC L
          BUF = GeneralRegis(7) + 1
          If BUF > 255 Then
            BUF = BUF - 256
            PARITYOVERFLOW = True
          Else
            PARITYOVERFLOW = False
          End If
          GeneralRegis(7) = BUF
Case "03" 'INC BC
          BC = BC + 1
          BCstr = Right("000" & Hex(BC), 4)
          GeneralRegis(1) = Right(HLstr, 2) 'B
          GeneralRegis(2) = Left(HLstr, 2) 'C
Case "13" 'INC DE
```

```
            DE = DE + 1
            Dim DEstr As String
            DEstr = Right("000" & Hex(DE), 4)
            GeneralRegis(3) = Right(DEstr, 2) 'D
            GeneralRegis(4) = Left(DEstr, 2) 'E
        Case "23" 'INC HL
            HL = HL + 1
            Dim HLstr As String
            HLstr = Right("000" & Hex(HL), 4)
            GeneralRegis(6) = Right(HLstr, 2) 'H
            GeneralRegis(7) = Left(HLstr, 2) 'L
        Case "34" 'INC( HL)
            RAM(HL) = RAM(HL) + 1
    End Select
    NEGATIVEFLAG = False
    Flag = FLAGSET
End Function
Public Function DEC(ByVal opcode As String)
'Case "3D", "05", "0D", "15", "1D", "25", "2D", "0B", "1B", "2B", "35" 'DEC
    Dim HL As Long, BC As Long, DE As Long, SP As Long
    Dim HLstr As String
    Dim BUF As Long
    Dim BCstr As String
    Dim DEstr As String
    Dim HLstr As String
    HL = Val(Right(String(2, "0") & Hex(GeneralRegis(6)), 2) & Right(String(2, "0") & Hex(GeneralRegis(7)), 2))
    BC = Val(Right(String(2, "0") & Hex(GeneralRegis(1)), 2) & Right(String(2, "0") & Hex(GeneralRegis(2)), 2))
    DE = Val(Right(String(2, "0") & Hex(GeneralRegis(3)), 2) & Right(String(2, "0") & Hex(GeneralRegis(4)), 2))
    Select Case opcode
        Case "3D" 'DEC A
            BUF = GeneralRegis(0) - 1
            If BUF < 0 Then
                BUF = BUF + 256
                PARITYOVERFLOW = True
            Else
                PARITYOVERFLOW = False
            End If
            GeneralRegis(0) = BUF
        Case "05" 'DEC B
            BUF = GeneralRegis(1) - 1
            If BUF < 0 Then
                BUF = BUF + 256
                PARITYOVERFLOW = True
            Else
                PARITYOVERFLOW = False
            End If
            GeneralRegis(1) = BUF
```

```
Case "0D" 'DEC C
        BUF = GeneralRegis(2) - 1
        If BUF < 0 Then
            BUF = BUF + 256
            PARITYOVERFLOW = True
        Else
            PARITYOVERFLOW = False
        End If
        GeneralRegis(2) = BUF
Case "15" 'DEC D
        BUF = GeneralRegis(3) - 1
        If BUF < 0 Then
            BUF = BUF + 256
            PARITYOVERFLOW = True
        Else
            PARITYOVERFLOW = False
        End If
        GeneralRegis(3) = BUF
Case "1D" 'DEC E
        BUF = GeneralRegis(4) - 1
        If BUF < 0 Then
            BUF = BUF + 256
            PARITYOVERFLOW = True
        Else
            PARITYOVERFLOW = False
        End If
        GeneralRegis(4) = BUF
Case "25" 'DEC H
        BUF = GeneralRegis(6) - 1
        If BUF < 0 Then
            BUF = BUF + 256
            PARITYOVERFLOW = True
        Else
            PARITYOVERFLOW = False
        End If
        GeneralRegis(6) = BUF
Case "2D" 'DEC L
        BUF = GeneralRegis(7) - 1
        If BUF < 0 Then
            BUF = BUF + 256
            PARITYOVERFLOW = True
        Else
            PARITYOVERFLOW = False
        End If
        GeneralRegis(7) = BUF
Case "0B" 'DEC BC
        BC = BC - 1
```

```
            BCstr = Right("000" & Hex(BC), 4)
            GeneralRegis(1) = Right(HLstr, 2) 'B
            GeneralRegis(2) = Left(HLstr, 2) 'C
    Case "1B" 'DEC DE
            DE = DE - 1
            DEstr = Right("000" & Hex(DE), 4)
            GeneralRegis(3) = Right(DEstr, 2) 'D
            GeneralRegis(4) = Left(DEstr, 2) 'E
    Case "2B" 'DEC HL
            HL = HL - 1
            HLstr = Right("000" & Hex(HL), 4)
            GeneralRegis(6) = Right(HLstr, 2) 'H
            GeneralRegis(7) = Left(HLstr, 2) 'L
     Case "35" 'DEC( HL)
          RAM(HL) = RAM(HL) - 1
    End Select
    NEGATIVEFLAG = False
    Flag = FLAGSET
    End Function
**********************************************ไฟล์โมดูล bas_menu**********************************************
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
Sub SetMenu_On()
    'Menu File
    Main.mnuFileSave.Enabled = True
    Main.mnuFileSaveAs.Enabled = True
    Main.mnuPrint.Enabled = True
    Main.mnuPrintSetup.Enabled = True
    'Menu Edit
    Main.mnuPaste.Enabled = True
    Main.mnuFind.Enabled = True
    Main.mnuReplace.Enabled = True
    Main.mnuCompile.Enabled = True
End Sub
Sub Setmenu_Off()
    'Menu File
    Main.mnuFileSave.Enabled = False
    Main.mnuFileSaveAs.Enabled = False
    Main.mnuPrint.Enabled = False
    Main.mnuPrintSetup.Enabled = False
    Main.mnuPaste.Enabled = False
    Main.mnuFind.Enabled = False
```

```
    Main.mnuReplace.Enabled = False
    Main.mnuCompile.Enabled = False
End Sub
Public Sub ShowRegis()
    Regis.LbGRegis(0).Caption = "A =" & Right(String(2, "0") & Hex(GeneralRegis(0)), 2)
    Regis.LbGRegis(1).Caption = "B =" & Right(String(2, "0") & Hex(GeneralRegis(1)), 2)
    Regis.LbGRegis(2).Caption = "C =" & Right(String(2, "0") & Hex(GeneralRegis(2)), 2)
    Regis.LbGRegis(3).Caption = "D =" & Right(String(2, "0") & Hex(GeneralRegis(3)), 2)
    Regis.LbGRegis(4).Caption = "E =" & Right(String(2, "0") & Hex(GeneralRegis(4)), 2)
    Regis.LbGRegis(5).Caption = "F =" & Right(String(2, "0") & Hex(GeneralRegis(5)), 2)
    Regis.LbGRegis(6).Caption = "H = " & Right(String(2, "0") & Hex(GeneralRegis(6)), 2)
    Regis.LbGRegis(7).Caption = "L =" & Right(String(2, "0") & Hex(GeneralRegis(7)), 2)
    Regis.LbGRegis(8).Caption = "I = " & Right(String(2, "0") & Hex(GeneralRegis(8)), 2)
    Regis.LbGRegis(9).Caption = "R = " & Right(String(2, "0") & Hex(GeneralRegis(9)), 2)
    Regis.LbGRegis(10).Caption = "A'= " & Right(String(2, "0") & Hex(GeneralRegis(10)), 2)
    Regis.LbGRegis(11).Caption = "B'= " & Right(String(2, "0") & Hex(GeneralRegis(11)), 2)
    Regis.LbGRegis(12).Caption = "C'= " & Right(String(2, "0") & Hex(GeneralRegis(12)), 2)
    Regis.LbGRegis(13).Caption = "D' = " & Right(String(2, "0") & Hex(GeneralRegis(13)), 2)
    Regis.LbGRegis(14).Caption = "E' = " & Right(String(2, "0") & Hex(GeneralRegis(14)), 2)
    Regis.LbGRegis(15).Caption = "F' = " & Right(String(2, "0") & Hex(GeneralRegis(15)), 2)
    Regis.LbGRegis(16).Caption = "H' = " & Right(String(2, "0") & Hex(GeneralRegis(16)), 2)
    Regis.LbGRegis(17).Caption = "L' = " & Right(String(2, "0") & Hex(GeneralRegis(17)), 2)
    Regis.LbSpcReg(0).Caption = "IX = " & Right(String(4, "0") & Hex(SpecialRegis(0)), 4)
    Regis.LbSpcReg(1).Caption = "IY = " & Right(String(4, "0") & Hex(SpecialRegis(1)), 4)
    Regis.LbSpcReg(2).Caption = "SP = " & Right(String(4, "0") & Hex(SpecialRegis(2)), 4)
    Regis.LbSpcReg(3).Caption = "PC = " & Right(String(4, "0") & Hex(SpecialRegis(3)), 4)
    If vbrcGetBitB(GeneralRegis(0), 0) = True Then 'A0
        Regis.ShapeA(0).FillColor = RGB(255, 0, 0)
    Else
        Regis.ShapeA(0).FillColor = RGB(0, 200, 255)
    End If
    If vbrcGetBitB(GeneralRegis(0), 1) = True Then 'A1
        Regis.ShapeA(1).FillColor = RGB(255, 0, 0)
    Else
        Regis.ShapeA(1).FillColor = RGB(0, 200, 255)
    End If
    If vbrcGetBitB(GeneralRegis(0), 2) = True Then 'A2
        Regis.ShapeA(2).FillColor = RGB(255, 0, 0)
    Else
        Regis.ShapeA(2).FillColor = RGB(0, 200, 255)
    End If
    If vbrcGetBitB(GeneralRegis(0), 3) = True Then 'A3
        Regis.ShapeA(3).FillColor = RGB(255, 0, 0)
    Else
        Regis.ShapeA(3).FillColor = RGB(0, 200, 255)
    End If
    If vbrcGetBitB(GeneralRegis(0), 4) = True Then 'A4
```

```
    Regis.ShapeA(4).FillColor = RGB(255, 0, 0)
  Else
    Regis.ShapeA(4).FillColor = RGB(0, 200, 255)
  End If
  If vbreGetBitB(GeneralRegis(0), 5) = True Then 'A5
    Regis.ShapeA(5).FillColor = RGB(255, 0, 0)
  Else
    Regis.ShapeA(5).FillColor = RGB(0, 200, 255)
  End If
  If vbreGetBitB(GeneralRegis(0), 6) = True Then 'A6
    Regis.ShapeA(6).FillColor = RGB(255, 0, 0)
  Else
    Regis.ShapeA(6).FillColor = RGB(0, 200, 255)
  End If
  If vbreGetBitB(GeneralRegis(0), 7) = True Then 'A7
    Regis.ShapeA(7).FillColor = RGB(255, 0, 0)
  Else
    Regis.ShapeA(7).FillColor = RGB(0, 200, 255)
  End If
  If vbreGetBitB(GeneralRegis(1), 0) = True Then 'B0
    Regis.ShapeB(0).FillColor = RGB(255, 0, 0)
  Else
    Regis.ShapeB(0).FillColor = RGB(0, 200, 255)
End If
  If vbreGetBitB(GeneralRegis(1), 1) = True Then 'B1
    Regis.ShapeB(1).FillColor = RGB(255, 0, 0)
  Else
    Regis.ShapeB(1).FillColor = RGB(0, 200, 255)
  End If
  If vbreGetBitB(GeneralRegis(1), 2) = True Then 'B2
    Regis.ShapeB(2).FillColor = RGB(255, 0, 0)
  Else
    Regis.ShapeB(2).FillColor = RGB(0, 200, 255)
  End If
  If vbreGetBitB(GeneralRegis(1), 3) = True Then 'B3
    Regis.ShapeB(3).FillColor = RGB(255, 0, 0)
  Else
    Regis.ShapeB(3).FillColor = RGB(0, 200, 255)
  End If
  If vbreGetBitB(GeneralRegis(1), 4) = True Then 'B4
    Regis.ShapeB(4).FillColor = RGB(255, 0, 0)
  Else
    Regis.ShapeB(4).FillColor = RGB(0, 200, 255)
  End If
  If vbreGetBitB(GeneralRegis(1), 5) = True Then 'B5
    Regis.ShapeB(5).FillColor = RGB(255, 0, 0)
  Else
```

```
Regis.ShapeB(5).FillColor = RGB(0, 200, 255)
    End If
    If vbrcGetBitB(GeneralRegis(1), 6) = True Then 'B6
      Regis.ShapeB(6).FillColor = RGB(255, 0, 0)
Else
      Regis.ShapeB(6).FillColor = RGB(0, 200, 255)
    End If
    If vbrcGetBitB(GeneralRegis(1), 7) = True Then 'B7
      Regis.ShapeB(7).FillColor = RGB(255, 0, 0)
   Else
      Regis.ShapeB(7).FillColor = RGB(0, 200, 255)
    End If
    If vbrcGetBitB(GeneralRegis(2), 0) = True Then 'C0
      Regis.ShapeC(0).FillColor = RGB(255, 0, 0)
   Else
      Regis.ShapeC(0).FillColor = RGB(0, 200, 255)
    End If
    If vbrcGetBitB(GeneralRegis(2), 1) = True Then 'C1
      Regis.ShapeC(1).FillColor = RGB(255, 0, 0)
   Else
      Regis.ShapeC(1).FillColor = RGB(0, 200, 255)
    End If
    If vbrcGetBitB(GeneralRegis(2), 2) = True Then 'C2
      Regis.ShapeC(2).FillColor = RGB(255, 0, 0)
   Else
      Regis.ShapeC(2).FillColor = RGB(0, 200, 255)
    End If
    If vbrcGetBitB(GeneralRegis(2), 3) = True Then 'C3
      Regis.ShapeC(3).FillColor = RGB(255, 0, 0)
   Else
      Regis.ShapeC(3).FillColor = RGB(0, 200, 255)
    End If
    If vbrcGetBitB(GeneralRegis(2), 4) = True Then 'C4
      Regis.ShapeC(4).FillColor = RGB(255, 0, 0)
   Else
      Regis.ShapeC(4).FillColor = RGB(0, 200, 255)
    End If
    If vbrcGetBitB(GeneralRegis(2), 5) = True Then 'C5
      Regis.ShapeC(5).FillColor = RGB(255, 0, 0)
   Else
      Regis.ShapeC(5).FillColor = RGB(0, 200, 255)
    End If
    If vbrcGetBitB(GeneralRegis(2), 6) = True Then 'C6
      Regis.ShapeC(6).FillColor = RGB(255, 0, 0)
   Else
      Regis.ShapeC(6).FillColor = RGB(0, 200, 255)
    End If
```

```
If vbrcGetBitB(GeneralRegis(2), 7) = True Then 'C7
   Regis.ShapeC(7).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeC(7).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(3), 0) = True Then 'D0
   Regis.ShapeD(0).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeD(0).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(3), 1) = True Then 'D1
   Regis.ShapeD(1).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeD(1).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(3), 2) = True Then 'D2
   Regis.ShapeD(2).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeD(2).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(3), 3) = True Then 'D3
   Regis.ShapeD(3).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeD(3).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(3), 4) = True Then 'D4
   Regis.ShapeD(4).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeD(4).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(3), 5) = True Then 'D5
   Regis.ShapeD(5).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeD(5).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(3), 6) = True Then 'D6
   Regis.ShapeD(6).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeD(6).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(3), 7) = True Then 'D7
   Regis.ShapeD(7).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeD(7).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(4), 0) = True Then 'E0
   Regis.ShapeE(0).FillColor = RGB(255, 0, 0)
```

```
Else
    Regis.ShapeE(0).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(4), 1) = True Then 'E1
    Regis.ShapeE(1).FillColor = RGB(255, 0, 0)
Else
    Regis.ShapeE(1).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(4), 2) = True Then 'E2
    Regis.ShapeE(2).FillColor = RGB(255, 0, 0)
Else
    Regis.ShapeE(2).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(4), 3) = True Then 'E3
    Regis.ShapeE(3).FillColor = RGB(255, 0, 0)
Else
    Regis.ShapeE(3).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(4), 4) = True Then 'E4
    Regis.ShapeE(4).FillColor = RGB(255, 0, 0)
Else
    Regis.ShapeE(4).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(4), 5) = True Then 'E5
    Regis.ShapeE(5).FillColor = RGB(255, 0, 0)
Else
    Regis.ShapeE(5).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(4), 6) = True Then 'E6
    Regis.ShapeE(6).FillColor = RGB(255, 0, 0)
Else
    Regis.ShapeE(6).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(4), 7) = True Then 'E7
    Regis.ShapeE(7).FillColor = RGB(255, 0, 0)
Else
    Regis.ShapeE(7).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(5), 0) = True Then 'F0
    Regis.ShapeF(0).FillColor = RGB(255, 0, 0)
Else
    Regis.ShapeF(0).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(5), 1) = True Then 'F1
    Regis.ShapeF(1).FillColor = RGB(255, 0, 0)
Else
    Regis.ShapeF(1).FillColor = RGB(0, 200, 255)
```

```
End If
If vbrcGetBitB(GeneralRegis(5), 2) = True Then 'F2
   Regis.ShapeF(2).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeF(2).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(5), 3) = True Then 'F3
   Regis.ShapeF(3).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeF(3).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(5), 4) = True Then 'F4
   Regis.ShapeF(4).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeF(4).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(5), 5) = True Then 'F5
   Regis.ShapeF(5).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeF(5).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(5), 6) = True Then 'F6
   Regis.ShapeF(6).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeF(6).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(5), 7) = True Then 'F7
   Regis.ShapeF(7).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeF(7).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(6), 0) = True Then 'H0
   Regis.ShapeH(0).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeH(0).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(6), 1) = True Then 'H1
   Regis.ShapeH(1).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeH(1).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(6), 2) = True Then 'H2
   Regis.ShapeH(2).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeH(2).FillColor = RGB(0, 200, 255)
End If
If vbrcGetBitB(GeneralRegis(6), 3) = True Then 'H3
   Regis.ShapeH(3).FillColor = RGB(255, 0, 0)
```

```
Else
    Regis.ShapeH(3).FillColor = RGB(0, 200, 255)
End If
 If vbrcGetBitB(GeneralRegis(6), 4) = True Then 'H4
    Regis.ShapeH(4).FillColor = RGB(255, 0, 0)
Else
    Regis.ShapeH(4).FillColor = RGB(0, 200, 255)
End If
 If vbrcGetBitB(GeneralRegis(6), 5) = True Then 'H5
    Regis.ShapeH(5).FillColor = RGB(255, 0, 0)
Else
    Regis.ShapeH(5).FillColor = RGB(0, 200, 255)
End If
 If vbrcGetBitB(GeneralRegis(6), 6) = True Then 'H6
    Regis.ShapeH(6).FillColor = RGB(255, 0, 0)
Else
    Regis.ShapeH(6).FillColor = RGB(0, 200, 255)
End If
 If vbrcGetBitB(GeneralRegis(6), 7) = True Then 'H7
    Regis.ShapeH(7).FillColor = RGB(255, 0, 0)
Else
    Regis.ShapeH(7).FillColor = RGB(0, 200, 255)
End If
 If vbrcGetBitB(GeneralRegis(7), 0) = True Then 'L0
    Regis.ShapeL(0).FillColor = RGB(255, 0, 0)
Else
    Regis.ShapeL(0).FillColor = RGB(0, 200, 255)
End If
 If vbrcGetBitB(GeneralRegis(7), 1) = True Then 'L1
    Regis.ShapeL(1).FillColor = RGB(255, 0, 0)
Else
    Regis.ShapeL(1).FillColor = RGB(0, 200, 255)
End If
 If vbrcGetBitB(GeneralRegis(7), 2) = True Then 'L2
    Regis.ShapeL(2).FillColor = RGB(255, 0, 0)
Else
    Regis.ShapeL(2).FillColor = RGB(0, 200, 255)
End If
 If vbrcGetBitB(GeneralRegis(7), 3) = True Then 'L3
    Regis.ShapeL(3).FillColor = RGB(255, 0, 0)
Else
    Regis.ShapeL(3).FillColor = RGB(0, 200, 255)
End If
 If vbrcGetBitB(GeneralRegis(7), 4) = True Then 'L4
    Regis.ShapeL(4).FillColor = RGB(255, 0, 0)
Else

    Regis.ShapeL(4).FillColor = RGB(0, 200, 255)
```

```
End If
If vbreGetBitB(GeneralRegis(7), 5) = True Then 'L5
   Regis.ShapeL(5).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeL(5).FillColor = RGB(0, 200, 255)
End If
If vbreGetBitB(GeneralRegis(7), 6) = True Then 'L6
   Regis.ShapeL(6).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeL(6).FillColor = RGB(0, 200, 255)
End If
If vbreGetBitB(GeneralRegis(7), 7) = True Then 'L7
   Regis.ShapeL(7).FillColor = RGB(255, 0, 0)
Else
   Regis.ShapeL(7).FillColor = RGB(0, 200, 255)
End If
'CARRY FLAG
If vbreGetBitB(GeneralRegis(5), 0) = True Then
   Regis.shapeflag(0).FillColor = RGB(255, 0, 0)
Else
   Regis.shapeflag(0).FillColor = RGB(0, 200, 255)
End If
'NEGATIVE FLAG
If vbreGetBitB(GeneralRegis(5), 1) = True Then
   Regis.shapeflag(1).FillColor = RGB(255, 0, 0)
Else
   Regis.shapeflag(1).FillColor = RGB(0, 200, 255)
End If
'PARITY OVERFLOW FLAG
If vbreGetBitB(GeneralRegis(5), 2) = True Then
   Regis.shapeflag(2).FillColor = RGB(255, 0, 0)
Else
   Regis.shapeflag(2).FillColor = RGB(0, 200, 255)
End If
'NO USE
If vbreGetBitB(GeneralRegis(5), 3) = True Then
   Regis.shapeflag(3).FillColor = RGB(255, 0, 0)
Else
   Regis.shapeflag(3).FillColor = RGB(0, 200, 255)
End If
'HALF CARRY FLAG
If vbreGetBitB(GeneralRegis(5), 4) = True Then
   Regis.shapeflag(4).FillColor = RGB(255, 0, 0)
Else
   Regis.shapeflag(4).FillColor = RGB(0, 200, 255)
End If

'NO USE
```

```
    If vbreGetBitB(GeneralRegis(5), 5) = True Then
      Regis.shapeflag(5).FillColor = RGB(255, 0, 0)
    Else
      Regis.shapeflag(5).FillColor = RGB(0, 200, 255)
    End If
    'ZERO FLAG
    If vbreGetBitB(GeneralRegis(5), 6) = True Then
      Regis.shapeflag(6).FillColor = RGB(255, 0, 0)
    Else
      Regis.shapeflag(6).FillColor = RGB(0, 200, 255)
    End If
    'SIGN FLAG
    If vbreGetBitB(GeneralRegis(5), 7) = True Then
      Regis.shapeflag(7).FillColor = RGB(255, 0, 0)
    Else
      Regis.shapeflag(7).FillColor = RGB(0, 200, 255)
    End If
End Sub
Public Sub displayMem()
    Dim row As Long
    Dim strBuf As String
    Dim startbyte As Long
      Dim a As Long
      Dim b As Long
      Dim BUF As String
      ScaleM = Memo.VScroll1.Value
      startbyte = ScaleM * 16
     'นำข้อมูลใน Ram(64K) มาแสดง ในฟอร์ม Memory
      For b = 0 To 5
      If startbyte < 65521 Then
        For a = 0 To 15
          BUF = BUF & "  " & Right(String(2, "0") & Hex(RAM(startbyte + a)), 2)
        Next
          strBuf = strBuf & "[" & Right(String(3, "0") & Hex(startbyte), 4) & "]-" & BUF & Chr(13) & Chr(10)
          startbyte = startbyte + 16
          BUF = ""
      End If
      Next
      Memo.txtmem.Text = strBuf
      Memo.txtmem.Visible = True
      Memo.Refresh
End Sub

'**********************************************ไฟล์โมดูลSim80**********************************************
Option Explicit
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 Function vbrcBinStrB Lib "Bits32.dll" _
   (ByVal ByteVal As Byte) As String
Public Declare Function vbrcGetBitB Lib "Bits32.dll" _
   (ByVal ByteVal As Byte, ByVal BitNum As Integer) As Integer
Public Declare Function vbrcSwapBitB Lib "Bits32.dll" _
   (ByVal ByteVal As Byte) As Byte
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" _
   (WordVal As Integer, ByVal BitNum As Integer)
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)
' ---------- DWORD SIZE BIT MANIPULATION ----------
Public Declare Function vbreBinStrD Lib "Bits32.dll" _
    (ByVal DWordVal As Long) As String
Public Declare Function vbreGetBitD Lib "Bits32.dll" _
    (ByVal DWordVal As Long, ByVal BitNum As Integer) As Integer
Public Declare Function vbreSwapBitD Lib "Bits32.dll" _
    (ByVal DWordVal As Long) As Long
Public Declare Sub vbreResetBitD Lib "Bits32.dll" _
    (DWordVal As Long, ByVal BitNum As Integer)
Public Declare Sub vbreROLD Lib "Bits32.dll" _
    (DWordVal As Long, ByVal BitNum As Integer)
Public Declare Sub vbreRORD Lib "Bits32.dll" _
    (DWordVal As Long, ByVal BitNum As Integer)
Public Declare Sub vbreSetBitD Lib "Bits32.dll" _
    (DWordVal As Long, ByVal BitNum As Integer)
Public Declare Sub vbreSHLD Lib "Bits32.dll" _
    (DWordVal As Long, ByVal BitNum As Integer)
Public Declare Sub vbreSHRD Lib "Bits32.dll" _
    (DWordVal As Long, ByVal BitNum As Integer)
Public Declare Sub vbreToggleBitD Lib "Bits32.dll" _
    (DWordVal As Long, ByVal BitNum As Integer)
Public CARRYFLAG As Boolean
Public NEGATIVEFLAG As Boolean
Public PARITYOVERFLOW As Boolean
Public HALFCARRYFLAG As Boolean
Public ZEROFLAG As Boolean
Public SIGNFLAG As Boolean
Public Flag As Byte
Public CLOCK As Boolean
Public GeneralRegis(18) As Byte      'ตัวแปรกำหนดรีจิสเตอร์ขนาด 8 Bit (0 is A,1 is B, 2 is C, 3 is D,4 is E, 5 is F, 6 is H,7 is L,8 is I, 9 is R)
Public SpecialRegis(3) As Long       'ตัวแปรกำหนดรีจิสเตอร์เฉพาะ ขนาด16 Bit (0 is PC, 1 is SP, 2 is IX,3 is IY)
Public RAM(65535) As Byte            'ตัวแปรกำหนดหน่วยความจำ RAM 64 Kbyte
Public AF As Long
Public BC As Long
Public DE As Long
Public HL As Long
Global InstrCounter As Long          'ตัวแปรที่กำหนดจำนวนไบต์ของชุดคำสั่ง
Sub INtitial()
  CARRYFLAG = False
  NEGATIVEFLAG = False
  PARITYOVERFLOW = False
  HALFCARRYFLAG = False
  ZEROFLAG = False
  SIGNFLAG = False
  Flag = FLAGSET
End Sub
Function SetFlag(ByVal PosFlag As Integer)
```

```
  Select Case PosFlag
    Case CARRY_FLAG
        vbrcSetBitB GeneralRegis(5), 0
    Case NEGATIVE_FLAG
        vbrcSetBitB GeneralRegis(5), 1
    Case ParityOver_FLAG
        vbrcSetBitB GeneralRegis(5), 2
    Case HALF_FLAG
        vbrcSetBitB GeneralRegis(5), 4
    Case ZERO_FLAG
        vbrcSetBitB GeneralRegis(5), 6
    Case SIGN_FLAG
        vbrcSetBitB GeneralRegis(5), 7
End Function
' ฟังก์ชั่นในการตรวจเช็ครีจิสเตอร์ แฟล็ก '
Public Function FLAGSET() As Byte
    If (CARRYFLAG = True) Then
    vbrcSetBitB Flag, 0
  Else
    vbrcResetBitB Flag, 0
  End If
  If (NEGATIVEFLAG = True) Then
    vbrcSetBitB Flag, 1

  Else
    vbrcResetBitB Flag, 1
  End If
  If (PARITYOVERFLOW = True) Then
    vbrcSetBitB Flag, 2
  Else
    vbrcResetBitB Flag, 2
  End If
  If (HALFCARRYFLAG = True) Then
    vbrcSetBitB Flag, 4
  Else
    vbrcResetBitB Flag, 4
  End If
  If (ZEROFLAG = True) Then
    vbrcSetBitB Flag, 6
    Else
    vbrcResetBitB Flag, 6
  End If
  If (SIGNFLAG = True) Then
    vbrcSetBitB Flag, 7

  Else
    vbrcResetBitB Flag, 7
  End If
```

```
    End Function
' ชุดคำสั่งจำลอง 8 บิต    ** กระทำทางตรรกะ **  '
Public Sub BOOLEEN8(SOURCE As Byte, FN As String)
    Dim Ab, So, OUT8 As Byte
    Ab = GeneralRegis(0)
    So = SOURCE
    Select Case FN
    Case "AND"
        OUT8 = (Ab And So)
        HALFCARRYFLAG = True
    Case "OR"
        OUT8 = (Ab Or So)
        HALFCARRYFLAG = False
    Case "XOR"
        OUT8 = (Ab Xor So)
        HALFCARRYFLAG = False
    End Select
    GeneralRegis(0) = OUT8
SETBOOLEEN (OUT8)
End Sub
Public Function SETBOOLEEN(OUT As Byte)
  Dim O(7) As Byte
  'PARITY FLAG CHECK'
    O(0) = vbreGetBitB(OUT, 0)
    O(1) = vbreGetBitB(OUT, 1)
    O(2) = vbreGetBitB(OUT, 2)
    O(3) = vbreGetBitB(OUT, 3)
    O(4) = vbreGetBitB(OUT, 4)
    O(5) = vbreGetBitB(OUT, 5)
    O(6) = vbreGetBitB(OUT, 6)
    O(7) = vbreGetBitB(OUT, 7)
    If (O(0) + O(1) + O(2) + O(3) + O(4) + O(5) + O(6) + O(7)) Mod 2 Then
       PARITYOVERFLOW = False
    Else
       PARITYOVERFLOW = True
    End If
    'SIGN CHECK
    If O(7) = 0 Then
       SIGNFLAG = False
    Else
       SIGNFLAG = True
    End If
    'ZERO CHECK
    If OUT = 0 Then

       ZEROFLAG = True
    Else
       ZEROFLAG = False
```

```
    End If
    CARRYFLAG = False
    NEGATIVEFLAG = False
    Flag = FLAGSET
End Function
'OR 8 BIT'
Public Function ORbyte(a As Byte, b As Byte) As Byte
    Dim a0 As Integer
    Dim a1 As Integer
    Dim a2 As Integer
    Dim a3 As Integer
    Dim a4 As Integer
    Dim a5 As Integer
    Dim a6 As Integer
    Dim a7 As Integer
    Dim B0 As Integer
    Dim B1 As Integer
    Dim B2 As Integer
    Dim B3 As Integer
    Dim B4 As Integer
    Dim B5 As Integer
    Dim B6 As Integer
    Dim B7 As Integer
    Dim C0 As Integer
    Dim C1 As Integer
    Dim C2 As Integer
    Dim C3 As Integer
    Dim C4 As Integer
    Dim C5 As Integer
    Dim C6 As Integer
    Dim C7 As Integer
    Dim Cc As Integer
    Dim f As Integer
    a0 = vbreGetBitB(a, 0)
    a1 = vbreGetBitB(a, 1)
    a2 = vbreGetBitB(a, 2)
    a3 = vbreGetBitB(a, 3)
    a4 = vbreGetBitB(a, 4)
    a5 = vbreGetBitB(a, 5)
    a6 = vbreGetBitB(a, 6)
    a7 = vbreGetBitB(a, 7)
    B0 = vbreGetBitB(b, 0)
    B1 = vbreGetBitB(b, 1)
    B2 = vbreGetBitB(b, 2)

    B3 = vbreGetBitB(b, 3)
    B4 = vbreGetBitB(b, 4)
    B5 = vbreGetBitB(b, 5)
```

```
    B6 = vbreGetBitB(b, 6)
    B7 = vbreGetBitB(b, 7)
    'AND BIT OPERATE'
    C0 = a0 Or B0
    C1 = a1 Or B1
    C2 = a2 Or B2
    C3 = a3 Or B3
    C4 = a4 Or B4
    C5 = a5 Or B5
    C6 = a6 Or B6
    C7 = a7 Or B7
    'PARITY FLAG CHECK'
    f = C0 + C1 + C2 + C3 + C4 + C5 + C6 + C7
    If (f / 2 = 1) Or (f / 4 = 1) Or (f / 6 = 1) Or (f / 8 = 1) Then
      PARITYOVERFLOW = True
      Else
      PARITYOVERFLOW = False
    End If
    HALFCARRYFLAG = False
    CARRYFLAG = False
    NEGATIVEFLAG = False
    'CONVERT BINARY TO INTEGER'
    Cc = ((2 ^ 7 * C7) + (2 ^ 6 * C6) + (2 ^ 5 * C5) + (2 ^ 4 * C4) + (2 ^ 3 * C3) + (2 ^ 2 * C2) + 2 * C1 + 1 * C0)

    If Cc = 0 Then
       ZEROFLAG = True
    Else
      ZEROFLAG = False
    End If
    Flag = FLAGSET
    ORbyte = Cc
End Function
'XOR 8 BIT'
Public Function XORbyte(a As Byte, b As Byte) As Byte
    Dim a0 As Integer
    Dim a1 As Integer
    Dim a2 As Integer
    Dim a3 As Integer
    Dim a4 As Integer
    Dim a5 As Integer
    Dim a6 As Integer
    Dim a7 As Integer
    Dim B0 As Integer
    Dim B1 As Integer
    Dim B2 As Integer
    Dim B3 As Integer
    Dim B4 As Integer
    Dim B5 As Integer
```

```
    Dim B6 As Integer
    Dim B7 As Integer
    Dim C0 As Integer
    Dim C1 As Integer
    Dim C2 As Integer
    Dim C3 As Integer
    Dim C4 As Integer
    Dim C5 As Integer
    Dim C6 As Integer
    Dim C7 As Integer
    Dim Cc As Integer
    Dim f As Integer
    a0 = vbreGetBitB(a, 0)
    a1 = vbreGetBitB(a, 1)
    a2 = vbreGetBitB(a, 2)
    a3 = vbreGetBitB(a, 3)
    a4 = vbreGetBitB(a, 4)
    a5 = vbreGetBitB(a, 5)
    a6 = vbreGetBitB(a, 6)
    a7 = vbreGetBitB(a, 7)
    B0 = vbreGetBitB(b, 0)
    B1 = vbreGetBitB(b, 1)
    B2 = vbreGetBitB(b, 2)
    B3 = vbreGetBitB(b, 3)
    B4 = vbreGetBitB(b, 4)
    B5 = vbreGetBitB(b, 5)
    B6 = vbreGetBitB(b, 6)
    B7 = vbreGetBitB(b, 7)
    'XOR BIT OPERATE'
    C0 = a0 Xor B0
    C1 = a1 Xor B1
    C2 = a2 Xor B2
    C3 = a3 Xor B3
    C4 = a4 Xor B4
    C5 = a5 Xor B5
    C6 = a6 Xor B6
    C7 = a7 Xor B7
    'PARITY FLAG CHECK'
    f = C0 + C1 + C2 + C3 + C4 + C5 + C6 + C7
    If (f \ 2 = 1) Or (f \ 4 = 1) Or (f \ 6 = 1) Or (f / 8 = 1) Then
      PARITYOVERFLOW = True
    Else
  PARITYOVERFLOW = False
    End If
    HALFCARRYFLAG = False
    CARRYFLAG = False
    NEGATIVEFLAG = False
  'CONVERT BINARY TO INTEGER'
```

```
    Cc = ((2 ^ 7 * C7) + (2 ^ 6 * C6) + (2 ^ 5 * C5) + (2 ^ 4 * C4) + (2 ^ 3 * C3) + (2 ^ 2 * C2) + 2 * C1 + 1 * C0)
    If Cc = 0 Then
       ZEROFLAG = True
    Else
       ZEROFLAG = False
    End If
    Flag = FLAGSET
    XORbyte = Cc
End Function
'XOR A 8 BIT'
Public Function XORAbyte(a As Byte) As Byte
    CARRYFLAG = False
    NEGATIVEFLAG = False
    PARITYOVERFLOW = True
    HALFCARRYFLAG = False
    ZEROFLAG = True
    SIGNFLAG = False
    Flag = FLAGSET
    XORAbyte = 0
End Function
Public Function ADD8(SOURCE As String)
    Dim Ab, So As Integer
    Dim a As Long
    Ab = Val("&H" & GeneralRegis(0))
    So = Val("&H" & SOURCE)
    a = Ab + So
    If a > 255 Or a < 0 Then
       CARRYFLAG = True
    Else
       CARRYFLAG = False
    End If
    If Len(Hex(a)) < 2 Then
       GeneralRegis(0) = "0" & Hex(a)
    Else
       GeneralRegis(0) = Right(Hex(a), 2)
    End If
    Main.LbGRegis(0).Caption = "A = " & GeneralRegis(0) & "H"
    arismatic8flagcheck (0)
End Function
Public Function SUB8(SOURCE As String)
    Dim Ab, So As Integer
    Dim a As Long
    Ab = Val("&H" & GeneralRegis(0))
    So = Val("&H" & SOURCE)
    a = Ab - So
    If a > 255 Or a < 0 Then
       CARRYFLAG = True
    Else
```

```
        CARRYFLAG = False
    End If
    If Len(Hex(a)) < 2 Then
        GeneralRegis(0) = "0" & Hex(a)
    Else
        GeneralRegis(0) = Right(Hex(a), 2)
    End If
    Main.LbGRegis(0).Caption = "A = " & GeneralRegis(0) & "H"
    arismatic8flagcheck (1)
End Function
Public Function ADC8(SOURCE As String)
    Dim Ab, So As Integer
    Dim a As Long
    Ab = Val("&H" & GeneralRegis(0))
    So = Val("&H" & SOURCE)
    If (CARRYFLAG = True) Then
        a = (Ab + So + 1)
    Else
        a = (Ab + So)
        'CARY CHECK
        If a > 255 Or a < 0 Then
            CARRYFLAG = True
        Else
            CARRYFLAG = False
        End If
    End If
    If Len(Hex(a)) < 2 Then
        GeneralRegis(0) = "0" & Hex(a)
    Else
        GeneralRegis(0) = Right(Hex(a), 2)
    End If
    Main.LbGRegis(0).Caption = "A = " & GeneralRegis(0) & "H"
    arismatic8flagcheck (0)
End Function
Public Function arismatic8flagcheck(use As Byte)
    'use= 0 by add function
    'use = 1 by sub function
    'OVER FLOW CHECK'
    If GeneralRegis(0) > 255 Or GeneralRegis(0) < 0 Then
        CARRYFLAG = True
    Else
        CARRYFLAG = False
    End If
    If GeneralRegis(0) < -128 Or GeneralRegis(0) > 127 Then
        'a = C + 256
        PARITYOVERFLOW = True
    Else
        PARITYOVERFLOW = False
```

```
    End If
    ' Half carry check
    If Val("&H" & Right(Hex(SOURCE), 1)) + Val("&H" & Right(Hex(GeneralRegis(0)), 1)) > 15 Or Val("&H" & Right(Hex(GeneralRegis(0)),
          1)) - Val("&H" & Right(Hex(SOURCE), 1)) < 0 Then
        HALFCARRYFLAG = True
    Else
        HALFCARRYFLAG = False
    End If
'Sign flag check
    If vbrcGetBitD(GeneralRegis(0), 7) = 0 Then
        SIGNFLAG = False
    Else
        SIGNFLAG = True
    End If
'Zero flag check
    If GeneralRegis(0) = 0 Then
      ZEROFLAG = True
    Else
      ZEROFLAG = False
    End If
    If use = 0 Then
      NEGATIVEFLAG = False
    Else
      NEGATIVEFLAG = True
    End If
    Flag = FLAGSET
End Function
Public Function ADCHL(SOURCE As String)
    Dim Ab, So As Long
    Dim HL, HL_ As Double
    Dim BUF As String
    Ab = Val("&H" & GeneralRegis(6) & GeneralRegis(7))
    If Ab < 0 Then Ab = 65536 + Ab
    So = Val("&H" & SOURCE)
    If So < 1 Then So = 65536 + So
    If (CARRYFLAG = True) Then
      HL = (Ab + So + 1)
      Else
      HL = (Ab + So)
    End If
    If HL > 65535 Then
        HL_ = HL - 65536
    Else
        HL_ = HL
    End If
  If Len(Hex(HL_)) = 3 Then
      BUF = "0" & Hex(HL_)
    ElseIf Len(Hex(HL_)) = 2 Then
```

```
    BUF = "00" & Hex(HL_)
   ElseIf Len(Hex(HL_)) = 1 Then
    BUF = "000" & Hex(HL_)
   Else
        BUF = Hex(HL_)
   End If
   GeneralRegis(6) = Left(BUF, 2)
   GeneralRegis(7) = Right(BUF, 2)
   Main.LbGRegis(6).Caption = "H = " & GeneralRegis(6) & "H"
   Main.LbGRegis(7).Caption = "L = " & GeneralRegis(7) & "H"
  'CARY CHECK
   If (HL > 65535) Then
       CARRYFLAG = True
    Else
       CARRYFLAG = False
    End If
    'OVER FLOW CHECK'
    If Val(GeneralRegis(6) & GeneralRegis(7)) < -32768 Or Val(GeneralRegis(6) & GeneralRegis(7)) > 32767 Then
       PARITYOVERFLOW = True
   Else
       PARITYOVERFLOW = False
    End If
    'Half carry check
    If Val("&H" & GeneralRegis(6) & GeneralRegis(7)) = 0 Then
      ZEROFLAG = True
    Else
      ZEROFLAG = False
   End If
   NEGATIVEFLAG = False
   Flag = FLAGSET
End Function
Public Function ADDHL(SOURCE As String)
   Dim Ab, So As Long
   Dim HL, HL_ As Double
   Dim BUF As String
   Ab = Val("&H" & GeneralRegis(6) & GeneralRegis(7))
   If Ab < 0 Then Ab = 65536 - Ab
   So = Val("&H" & SOURCE)
    If So < 0 Then So = 65536 - So
       HL = (Ab + So)
   If HL > 65535 Then
        HL_ = HL - 65536
   Else
        HL_ = HL
   End If
   If Len(Hex(HL_)) = 3 Then
     BUF = "0" & Hex(HL_)
   ElseIf Len(Hex(HL_)) = 2 Then
```

```
    BUF = "00" & Hex(HL_)
   ElseIf Len(Hex(HL_)) = 1 Then
    BUF = "000" & Hex(HL_)
   Else
        BUF = Hex(HL_)
   End If
   GeneralRegis(6) = Left(BUF, 2)
   GeneralRegis(7) = Right(BUF, 2)
   Main.LbGRegis(6).Caption = "H = " & GeneralRegis(6) & "H"
   Main.LbGRegis(7).Caption = "L = " & GeneralRegis(7) & "H"
   'CARY CHECK
   If (HL > 65535) Then
      CARRYFLAG = True
   Else
      CARRYFLAG = False
   End If
   'OVER FLOW CHECK'
   NEGATIVEFLAG = False
   Flag = FLAGSET
End Function
Public Function ADDX(SOURCE As String)
   Dim Ab, So As Long
   Dim IX, IX_ As Double
   Dim BUF As String
   Ab = Val("&H" & SpecialRegis(0))
   If Ab < 0 Then Ab = 65536 + Ab
   So = Val("&H" & SOURCE)
   If So < 0 Then So = 65536 + So
        IX = (Ab + So)
   If IX > 65535 Then
        IX_ = IX - 65536
   Else
        IX_ = IX
   End If
   If Len(Hex(IX_)) = 3 Then
      BUF = "0" & Hex(IX_)
   ElseIf Len(Hex(IX_)) = 2 Then
    BUF = "00" & Hex(IX_)
   ElseIf Len(Hex(IX_)) = 1 Then
    BUF = "000" & Hex(IX_)
   Else
      BUF = Hex(IX_)
   End If
   SpecialRegis(0) = BUF
   Main.LbSpcReg(0).Caption = "IX = " & SpecialRegis(0) & "H"
   'CARY CHECK
   If (IX > 65535) Then
      CARRYFLAG = True
```

```
    Else
        CARRYFLAG = False
    End If
    NEGATIVEFLAG = False
    Flag = FLAGSET
End Function

Public Function ADDY(SOURCE As String)
    Dim Ab, So As Long
    Dim IY, IY_ As Double
    Dim BUF As String
    Ab = Val("&H" & SpecialRegis(1))
    If Ab < 0 Then Ab = 65536 + Ab
    So = Val("&H" & SOURCE)
    If So < 0 Then So = 65536 + So
        IY = (Ab + So)
    If IY > 65535 Then
        IY_ = IY - 65536
    Else
        IY_ = IY
    End If
    If Len(Hex(IY_)) = 3 Then
      BUF = "0" & Hex(IY_)
    ElseIf Len(Hex(IY_)) = 2 Then
     BUF = "00" & Hex(IY_)
    ElseIf Len(Hex(IY_)) = 1 Then
     BUF = "000" & Hex(IY_)
      Else
      BUF = Hex(IY_)
    End If

    SpecialRegis(1) = BUF
    Main.LbSpcReg(1).Caption = "IY = " & SpecialRegis(1) & "H"
    'CARY CHECK
    If (IY > 65535) Then
        CARRYFLAG = True
    Else
        CARRYFLAG = False
    End If
    NEGATIVEFLAG = False
    Flag = FLAGSET
End Function
Public Function SBC8(SOURCE As String)
    Dim Ab, So As Integer
    Dim a As Long
    Ab = Val("&H" & GeneralRegis(0))
    If Ab < 0 Then Ab = 65536 + Ab
    So = Val("&H" & SOURCE)
```

```
    If So < 0 Then So = 65536 + So
    If (CARRYFLAG = True) Then
      a = (Ab - So - 1)
      Else
      a = (Ab - So)
    End If
     If a > 255 Or a < 0 Then
        CARRYFLAG = True
      Else
        CARRYFLAG = False
    End If
    If Len(Hex(a)) < 2 Then
      GeneralRegis(0) = "0" & Hex(a)
    Else
      GeneralRegis(0) = Right(Hex(a), 2)
    End If
    Main.LbGRegis(0).Caption = "A = " & GeneralRegis(0) & "H"
    arismatic8flagcheck (1)
End Function
Public Function SBCHL(SOURCE As String)
    Dim Ab, So As Long
    Dim HL, HL_ As Double
    Dim BUF As String
    Ab = Val("&H" & GeneralRegis(6) & GeneralRegis(7))
    If Ab < 0 Then Ab = 65536 + Ab
    So = Val("&H" & SOURCE)
    If So < 1 Then So = 65536 - So
    If (CARRYFLAG = True) Then
      HL = (Ab - So - 1)
      Else
      HL = (Ab - So)
    End If
     If HL < 0 Then
        HL_ = HL + 65536
    Else
        HL_ = HL
    End If
  If Len(Hex(HL_)) = 3 Then
      BUF = "0" & Hex(HL_)
    ElseIf Len(Hex(HL_)) = 2 Then
     BUF = "00" & Hex(HL_)
    ElseIf Len(Hex(HL_)) = 1 Then
     BUF = "000" & Hex(HL_)
    Else
        BUF = Hex(HL_)
    End If
    GeneralRegis(6) = Left(BUF, 2)
    GeneralRegis(7) = Right(BUF, 2)
```

```
    Main.LbGRegis(6).Caption = "H = " & GeneralRegis(6) & "H"
    Main.LbGRegis(7).Caption = "L = " & GeneralRegis(7) & "H"
    'CARY CHECK
    If (HL < 0) Then
        CARRYFLAG = True
    Else
        CARRYFLAG = False
    End If
    'OVER FLOW CHECK'
    If Val(GeneralRegis(6) & GeneralRegis(7)) < -32768 Or Val(GeneralRegis(6) & GeneralRegis(7)) > 32767 Then
        PARITYOVERFLOW = True
    Else
        PARITYOVERFLOW = False
    End If
    'Half carry check
    If Val("&H" & Right(SOURCE, 2)) + Val("&H" & Right(GeneralRegis(7), 2)) > 255 Then
        HALFCARRYFLAG = True
    Else
        HALFCARRYFLAG = False
    End If
    'SIGN CHECK
    If vbrcGetBitB(Val("&H" & GeneralRegis(6)), 7) Then
        SIGNFLAG = True
    Else
        SIGNFLAG = False
    End If
    'ZEROCHECK
    If Val("&H" & GeneralRegis(6) & GeneralRegis(7)) = 0 Then
        ZEROFLAG = True
    Else
        ZEROFLAG = False
    End If
    NEGATIVEFLAG = True
    Flag = FLAGSET
End Function
'SUBA 8 BIT'
Public Function SUBA() As Byte
    CARRYFLAG = False
    NEGATIVEFLAG = True
    PARITYOVERFLOW = True
    HALFCARRYFLAG = False
    ZEROFLAG = True
    SIGNFLAG = False
    Flag = FLAGSET
End Function
Public Function BIT(ByVal pos As String)
    Dim Databyte As Byte
    Databyte = Val("&H" & SOURCE)
```

```
    If vbreGetBitB(Databyte, Int(pos)) = 1 Then
        ZEROFLAG = False
    Else
        ZEROFLAG = True
    End If
    NEGATIVEFLAG = False
    HALFCARRYFLAG = True
    Flag = FLAGSET
End Function
Public Function SETB(a As String)
    Dim pos, row, col, Strlen As Integer
    Dim bufsource As Byte
    bufsource = Val("&H" & SOURCE)
    Select Case a
    Case "0"
        Call vbreSetBitB(bufsource, 0)
    Case "1"
        Call vbreSetBitB(bufsource, 1)
    Case "2"
        Call vbreSetBitB(bufsource, 2)
    Case "3"
        Call vbreSetBitB(bufsource, 3)
    Case "4"
        Call vbreSetBitB(bufsource, 4)
    Case "5"
        Call vbreSetBitB(bufsource, 5)
    Case "6"
        Call vbreSetBitB(bufsource, 6)
    Case "7"
        Call vbreSetBitB(bufsource, 7)
    End Select
    bituse (Hex(bufsource))
End Function
Public Function RES(a As String)
    Dim pos, row, col, Strlen As Integer
    Dim bufsource As Byte
    Dim BUF As String
    bufsource = Val("&H" & SOURCE)
    Select Case a
    Case "0"
        Call vbreResetBitB(bufsource, 0)
    Case "1"
        Call vbreResetBitB(bufsource, 1)
    Case "2"
        Call vbreResetBitB(bufsource, 2)
    Case "3"
        Call vbreResetBitB(bufsource, 3)
    Case "4"
```

```
    Call vbrcResetBitB(bufsource, 4)
  Case "5"
    Call vbrcResetBitB(bufsource, 5)
  Case "6"
    Call vbrcResetBitB(bufsource, 6)
  Case "7"
    Call vbrcResetBitB(bufsource, 7)
  End Select
  bituse (Hex(bufsource))
End Function
  Public Function bituse(BUF As String)
  Dim pos, row, col, Strlen As Integer
  If Len(BUF) < 2 Then BUF = "0" & BUF
  Select Case SourceN
   Case "A"
          GeneralRegis(0) = BUF
          Main.LbGRegis(0).Caption = "A = " & GeneralRegis(0) & "H"
   Case "B"
          GeneralRegis(1) = BUF
          Main.LbGRegis(1).Caption = "B = " & GeneralRegis(1) & "H"
   Case "C"
          GeneralRegis(2) = BUF
          Main.LbGRegis(2).Caption = "C = " & GeneralRegis(2) & "H"
   Case "D"
          GeneralRegis(3) = BUF
          Main.LbGRegis(3).Caption = "D = " & GeneralRegis(3) & "H"
   Case "E"
          GeneralRegis(4) = BUF
          Main.LbGRegis(4).Caption = "E = " & GeneralRegis(4) & "H"
   Case "H"
          GeneralRegis(6) = BUF
          Main.LbGRegis(6).Caption = "H = " & GeneralRegis(6) & "H"
   Case "L"
          GeneralRegis(7) = BUF
          Main.LbGRegis(7).Caption = "L = " & GeneralRegis(7) & "H"
   Case "(HL)"
          pos = Val("&H" & GeneralRegis(6) & GeneralRegis(7))
          If pos < 0 Then pos = 65536 + pos
          MEMory(pos) = BUF
          row = Val("&H" & "0" & GeneralRegis(6))
          col = Val("&H" & "0" & GeneralRegis(7))
          Main!MSFlexGrid1.row = row + 1
          Main!MSFlexGrid1.col = col + 1
          Main!MSFlexGrid1.Text = MEMory(pos)
   Case "(IX+dd)"
          pos = Val("&H" & SpecialRegis(0)) + Val("&H" & Right(instrData, 2))
          If pos < 0 Then pos = 65536 + pos
          MEMory(pos) = Left(BUF, 2)
```

```
            col = Val("&H" & "0" & Right(Hex(pos), 2))
            Strlen = Len(Hex(pos))
        Select Case Strlen
          Case 0, 1, 2
              row = 0
          Case 3
              row = Val("&H" & "0" & Left(Hex(pos), 1))
          Case 4
              row = Val("&H" & "0" & Left(Hex(pos), 2))
          Case 5
              row = Val("&H" & "0" & Left(Hex(pos), 3))
        End Select
            Main!MSFlexGrid1.row = row + 1
            Main!MSFlexGrid1.col = col + 1
            Main!MSFlexGrid1.Text = MEMory(pos)
    Case "(IY+dd)"
            pos = Val("&H" & Left(instrData, 2)) + Val("&H" & SpecialRegis(1))
            If pos < 0 Then pos = 65536 + pos
            MEMory(pos) = Left(BUF, 2)
             col = Val("&H" & "0" & Right(Hex(pos), 2))
            Strlen = Len(Hex(pos))
        Select Case Strlen
          Case 0, 1, 2
              row = 0
          Case 3
              row = Val("&H" & "0" & Left(Hex(pos), 1))
          Case 4
              row = Val("&H" & "0" & Left(Hex(pos), 2))
          Case 5
              row = Val("&H" & "0" & Left(Hex(pos), 3))
        End Select
            Main!MSFlexGrid1.row = row + 1
            Main!MSFlexGrid1.col = col + 1
            Main!MSFlexGrid1.Text = MEMory(pos)
    End Select
End Function
Public Function RLAold(SOURCE As String)
  Dim a, C(8), b(8), n As Integer
  Dim f As Integer
  a = Val("&H" & SOURCE)
  If a > 255 Then
  'CARRYFLAG = True
  a = a - 256
  Else
  'CARRYFLAG = False
  End If
  If CARRYFLAG = True Then
  C(8) = 1
```

```
    Else
     C(8) = 0
    End If
    C(0) = vbrcGetBitB(a, 0)
    C(1) = vbrcGetBitB(a, 1)
    C(2) = vbrcGetBitB(a, 2)
    C(3) = vbrcGetBitB(a, 3)
    C(4) = vbrcGetBitB(a, 4)
    C(5) = vbrcGetBitB(a, 5)
    C(6) = vbrcGetBitB(a, 6)
    C(7) = vbrcGetBitB(a, 7)
    b(0) = C(0)
    b(1) = C(1)
    b(2) = C(2)
    b(3) = C(3)
    b(4) = C(4)
    b(5) = C(5)
    b(6) = C(6)
    b(7) = C(7)
    b(8) = C(8)
    'หมุนซ้าย'
    C(0) = b(8)
    C(1) = b(0)
    C(2) = b(1)
    C(3) = b(2)
    C(4) = b(3)
    C(5) = b(4)
    C(6) = b(5)
    C(7) = b(6)
    CARRYFLAG = b(7)
    'CONVERT BINARY TO INTEGER'
    a = 0
    For n = 0 To 7
        a = a + ((2 ^ n) * C(n))
    Next
    NEGATIVEFLAG = False
    HALFCARRYFLAG = False
    Flag = FLAGSET
    If Len(Hex(a)) = 1 Then
       GeneralRegis(0) = "0" & Hex(a)
    Else
       GeneralRegis(0) = Hex(a)
    End If
    Main.LbGRegis(0).Caption = "A - " & GeneralRegis(0) & "H"
End Function
Public Function RRAold(SOURCE As String)
    Dim a, C(8), b(8), n As Integer
    Dim f As Integer
```

```
a = Val("&H" & SOURCE)
If a < 0 Then
  a = a + 256
End If
If CARRYFLAG = True Then
   C(8) = 1
Else
   C(8) = 0
End If
C(0) = vbrcGetBitB(a, 0)
C(1) = vbrcGetBitB(a, 1)
C(2) = vbrcGetBitB(a, 2)
C(3) = vbrcGetBitB(a, 3)
C(4) = vbrcGetBitB(a, 4)
C(5) = vbrcGetBitB(a, 5)
C(6) = vbrcGetBitB(a, 6)
C(7) = vbrcGetBitB(a, 7)

b(0) = C(0)
b(1) = C(1)
b(2) = C(2)
b(3) = C(3)
b(4) = C(4)
b(5) = C(5)
b(6) = C(6)
b(7) = C(7)
b(8) = C(8)
'หมุนซ้าย'
C(0) = b(1)
C(1) = b(2)
C(2) = b(3)
C(3) = b(4)
C(4) = b(5)
C(5) = b(6)
C(6) = b(7)
C(7) = b(8)
CARRYFLAG = b(0)
'CONVERT BINARY TO INTEGER'
 a = 0
 For n = 0 To 7
       a = a + ((2 ^ n) * C(n))
Next
NEGATIVEFLAG = False
HALFCARRYFLAG = False
Flag = FLAGSET
If Len(Hex(a)) = 1 Then
   GeneralRegis(0) = "0" & Hex(a)
Else
```

```
    GeneralRegis(0) = Hex(a)
  End If
  Main.LbGRegis(0).Caption = "A = " & GeneralRegs(0) & "H"
End Function
Public Function RRC(a As Byte)
  Dim C0 As Integer
  Dim C1 As Integer
  Dim C2 As Integer
  Dim C3 As Integer
  Dim C4 As Integer
  Dim C5 As Integer
  Dim C6 As Integer
  Dim C7 As Integer
  Dim Cf As Integer
  Dim Cc As Integer
  Dim f As Integer

  Call vbcRORB(a, 1)
  CARRYFLAG = vbcGetBitB(a, 0)
  C0 = vbcGetBitB(a, 0)
  C1 = vbcGetBitB(a, 1)
  C2 = vbcGetBitB(a, 2)
  C3 = vbcGetBitB(a, 3)
  C4 = vbcGetBitB(a, 4)
  C5 = vbcGetBitB(a, 5)
  C6 = vbcGetBitB(a, 6)
  C7 = vbcGetBitB(a, 7)
  'PARITY FLAG CHECK'
  f = C0 + C1 + C2 + C3 + C4 + C5 + C6 + C7
  If (f / 2 = 1) Or (f / 4 = 1) Or (f / 6 = 1) Or (f / 8 = 1) Then
    PARITYOVERFLOW = True
  Else
    PARITYOVERFLOW = False
  End If
  NEGATIVEFLAG = False
  HALFCARRYFLAG = False
  Flag = FLAGSET
  RRC = a
End Function

Public Function RRCAold(SOURCE As String)
  Dim a As Byte
  a = Val("&H" & SOURCE)
  CARRYFLAG = vbcGetBitB(a, 0)
  Call vbcRORB(a, 1)
  If Len(Hex(a)) = 1 Then
    GeneralRegis(0) = "0" & Hex(a)
  Else
```

```
    GeneralRegis(0) = Hex(a)
  End If
  Main.LbGRegis(0).Caption = "A =" & GeneralRegis(0) & "H"
  NEGATIVEFLAG = False
  HALFCARRYFLAG = False
  Flag = FLAGSET
End Function
Public Function RLCAold(SOURCE As String)
  Dim a As Byte
  a = Val("&H" & SOURCE)
  CARRYFLAG = vbreGetBitB(a, 7)
  Call vbreROLB(a, 1)
  If Len(Hex(a)) = 1 Then
    GeneralRegis(0) = "0" & Hex(a)
  Else
    GeneralRegis(0) = Hex(a)
  End If
  Main.LbGRegis(0).Caption = "A =" & GeneralRegis(0) & "H"
  NEGATIVEFLAG = False
  HALFCARRYFLAG = False
  Flag = FLAGSET
End Function
Public Function RLC(a As Byte)
  Dim C0 As Integer
  Dim C1 As Integer
  Dim C2 As Integer
  Dim C3 As Integer
  Dim C4 As Integer
  Dim C5 As Integer
  Dim C6 As Integer
  Dim C7 As Integer
  Dim Cf As Integer
  Dim Ce As Integer
  Dim f As Integer
  CARRYFLAG = vbreGetBitB(a, 7)
  Call vbreROLB(a, 1)
  C0 = vbreGetBitB(a, 0)
  C1 = vbreGetBitB(a, 1)
  C2 = vbreGetBitB(a, 2)
  C3 = vbreGetBitB(a, 3)
  C4 = vbreGetBitB(a, 4)
  C5 = vbreGetBitB(a, 5)
  C6 = vbreGetBitB(a, 6)
  C7 = vbreGetBitB(a, 7)
  'PARITY FLAG CHECK'
  f = C0 + C1 + C2 + C3 + C4 + C5 + C6 + C7
  If (f = 2) Or (f = 4) Or (f = 6) Or (f = 8) Then
    PARITYOVERFLOW = True
```

```
    Else
        PARITYOVERFLOW = False
    End If
    NEGATIVEFLAG = False
    HALFCARRYFLAG = False
    Flag = FLAGSET
    RLC = a
End Function
Public Function RR8(a As Byte) As Byte
    Dim C0 As Integer
    Dim C1 As Integer
    Dim C2 As Integer
    Dim C3 As Integer
    Dim C4 As Integer
    Dim C5 As Integer
    Dim C6 As Integer
    Dim C7 As Integer
    Dim Cf As Integer
    Dim Cc As Integer
    Dim bC0 As Integer
    Dim bC1 As Integer
    Dim bC2 As Integer
    Dim bC3 As Integer
    Dim bC4 As Integer
    Dim bC5 As Integer
    Dim bC6 As Integer
    Dim bC7 As Integer
    Dim bCf As Integer
    Dim bCc As Integer
    Dim f As Integer
    If CARRYFLAG = True Then
        Cc = 1
    Else
        Cc = 0
    End If

    C0 = vbreGetBitB(a, 0)
    C1 = vbreGetBitB(a, 1)
    C2 = vbreGetBitB(a, 2)
    C3 = vbreGetBitB(a, 3)
    C4 = vbreGetBitB(a, 4)
    C5 = vbreGetBitB(a, 5)
    C6 = vbreGetBitB(a, 6)
    C7 = vbreGetBitB(a, 7)
    bC0 = C0
    bC1 = C1
    bC2 = C2
    bC3 = C3
```

```
    bC4 = C4
    bC5 = C5
    bC6 = C6
    bC7 = C7
    bCc = Cc
    'หมุนขวา'
    C0 = bC1
    C1 = bC2
    C2 = bC3
    C3 = bC4
    C4 = bC5
    C5 = bC6
    C6 = bC7
    C7 = bCc
    CARRYFLAG = bC0
    'PARITY FLAG CHECK'
    f = C0 + C1 + C2 + C3 + C4 + C5 + C6 + C7
    If (f / 2 = 1) Or (f / 4 = 1) Or (f / 6 = 1) Or (f / 8 = 1) Then
       PARITYOVERFLOW = True
    Else
       PARITYOVERFLOW = False
    End If
    'CONVERT BINARY TO INTEGER'
    Cf = ((2 ^ 7 * C7) + (2 ^ 6 * C6) + (2 ^ 5 * C5) + (2 ^ 4 * C4) + (2 ^ 3 * C3) + (2 ^ 2 * C2) + 2 * C1 + 1 * C0)
    NEGATIVEFLAG = False
    HALFCARRYFLAG = False
    Flag = FLAGSET
    RR8 = Cf
End Function
Public Function RL8(a As Byte)
    Dim C0 As Integer
    Dim C1 As Integer
    Dim C2 As Integer
    Dim C3 As Integer
    Dim C4 As Integer
    Dim C5 As Integer
    Dim C6 As Integer
    Dim C7 As Integer
    Dim Cf As Integer
    Dim Cc As Integer
    Dim bC0 As Integer
    Dim bC1 As Integer
    Dim bC2 As Integer
    Dim bC3 As Integer
    Dim bC4 As Integer
    Dim bC5 As Integer
    Dim bC6 As Integer
    Dim bC7 As Integer
```

```
Dim bCf As Integer
Dim bCc As Integer
Dim f As Integer
If CARRYFLAG = True Then
    Cc = 1
 Else
    Cc = 0
End If
C0 = vbrcGetBitB(a, 0)



C1 = vbrcGetBitB(a, 1)
C2 = vbrcGetBitB(a, 2)
C3 = vbrcGetBitB(a, 3)
C4 = vbrcGetBitB(a, 4)
C5 = vbrcGetBitB(a, 5)
C6 = vbrcGetBitB(a, 6)
C7 = vbrcGetBitB(a, 7)
bC0 = C0
bC1 = C1
bC2 = C2
bC3 = C3
bC4 = C4
bC5 = C5
bC6 = C6
bC7 = C7
bCc = Cc
'หมุนซ้าย'
C0 = bCc
C1 = bC0
C2 = bC1
C3 = bC2
C4 = bC3
C5 = bC4
C6 = bC5
C7 = bC6
CARRYFLAG = bC7
'PARITY FLAG CHECK'
f = C0 + C1 + C2 + C3 + C4 + C5 + C6 + C7
If (f / 2 = 1) Or (f  4 = 1) Or (f  6 = 1) Or (f  8 = 1) Then
   PARITYOVERFLOW = True
Else
   PARITYOVERFLOW = False
End If
'CONVERT BINARY TO INTEGER'
Cf = (2 ^ 7 * C7) + (2 ^ 6 * C6) + (2 ^ 5 * C5) + (2 ^ 4 * C4) + (2 ^ 3 * C3) + (2 ^ 2 * C2) + 2 * C1 + 1 * C0
NEGATIVEFLAG = False
HALFCARRYFLAG = False
```

```
    Flag = FLAGSET
    RL8 = Cf
End Function
Public Function SLA(a As Byte)
    Dim C0 As Integer
    Dim C1 As Integer
    Dim C2 As Integer
    Dim C3 As Integer
    Dim C4 As Integer
    Dim C5 As Integer
    Dim C6 As Integer
    Dim C7 As Integer
    Dim Cf As Integer
    Dim Cc As Integer
    Dim bC0 As Integer
    Dim bC1 As Integer
    Dim bC2 As Integer
    Dim bC3 As Integer
    Dim bC4 As Integer
    Dim bC5 As Integer
    Dim bC6 As Integer
    Dim bC7 As Integer
    Dim bCf As Integer
    Dim bCc As Integer
    Dim f As Integer
    If CARRYFLAG = True Then
        Cc = 1
    Else
        Cc = 0
    End If
    C0 = vbreGetBitB(a, 0)
    C1 = vbreGetBitB(a, 1)
    C2 = vbreGetBitB(a, 2)
    C3 = vbreGetBitB(a, 3)
    C4 = vbreGetBitB(a, 4)
    C5 = vbreGetBitB(a, 5)
    C6 = vbreGetBitB(a, 6)
    C7 = vbreGetBitB(a, 7)
    bC0 = C0
    bC1 = C1
    bC2 = C2
    bC3 = C3
    bC4 = C4
    bC5 = C5
    bC6 = C6
    bC7 = C7
    bCc = Cc
    'ชิฟซ้าย
```

```
    C0 = 0
    C1 = bC0
    C2 = bC1
    C3 = bC2
    C4 = bC3
    C5 = bC4
    C6 = bC5
    C7 = bC6
    CARRYFLAG = bC7
    'PARITY FLAG CHECK'
    f = C0 + C1 + C2 + C3 + C4 + C5 + C6 + C7
    If (f / 2 = 1) Or (f / 4 = 1) Or (f / 6 = 1) Or (f / 8 = 1) Then
       PARITYOVERFLOW = True
    Else
       PARITYOVERFLOW = False
    End If
    'CONVERT BINARY TO INTEGER'
    Cf = ((2 ^ 7 * C7) + (2 ^ 6 * C6) + (2 ^ 5 * C5) + (2 ^ 4 * C4) + (2 ^ 3 * C3) + (2 ^ 2 * C2) + 2 * C1 + 1 * C0)
    NEGATIVEFLAG = False
    HALFCARRYFLAG = False
    Flag = FLAGSET
    SLA = Cf
End Function
Public Function SRA(a As Byte)
    Dim C0 As Integer
    Dim C1 As Integer
    Dim C2 As Integer
    Dim C3 As Integer
    Dim C4 As Integer
    Dim C5 As Integer
    Dim C6 As Integer
    Dim C7 As Integer
    Dim Cf As Integer
    Dim Cc As Integer
    Dim bC0 As Integer
    Dim bC1 As Integer
    Dim bC2 As Integer
    Dim bC3 As Integer
    Dim bC4 As Integer
    Dim bC5 As Integer
    Dim bC6 As Integer
    Dim bC7 As Integer
    Dim bCf As Integer
    Dim bCc As Integer
    Dim f As Integer
    If CARRYFLAG = True Then
        Cc = 1
    Else
```

```
    Cc = 0
  End If
  C0 = vbreGetBitB(a, 0)
  C1 = vbreGetBitB(a, 1)
  C2 = vbreGetBitB(a, 2)
  C3 = vbreGetBitB(a, 3)
  C4 = vbreGetBitB(a, 4)
  C5 = vbreGetBitB(a, 5)
  C6 = vbreGetBitB(a, 6)
  C7 = vbreGetBitB(a, 7)
  bC0 = C0
  bC1 = C1
  bC2 = C2
  bC3 = C3
  bC4 = C4
  bC5 = C5
  bC6 = C6
  bC7 = C7
  bCc = Cc
  ' ชิฟขวา
  C0 = bC1
  C1 = bC2
  C2 = bC3
  C3 = bC4
  C4 = bC5
  C5 = bC6
  C6 = bC7
  C7 = bCc
  CARRYFLAG = 0
  C7 = bC7
  'PARITY FLAG CHECK'
  f = C0 + C1 + C2 + C3 + C4 + C5 + C6 + C7
  If (f / 2 = 1) Or (f  4 = 1) Or (f  6 = 1) Or (f / 8 = 1) Then
     PARITYOVERFLOW = True
  Else
     PARITYOVERFLOW = False
  End If
  'CONVERT BINARY TO INTEGER'
  Cf = ((2 ^ 7 * C7) + (2  6 * C6) + (2 ^ 5 * C5) + (2 ^ 4 * C4) + (2 ^ 3 * C3) + (2 ^ 2 * C2) + 2 * C1 + 1 * C0)
  NEGATIVEFLAG = False
  HALFCARRYFLAG = False
  Flag = FLAGSET
  SRA = Cf
End Function
Public Function SRL(a As Byte)
  Dim C0 As Integer
  Dim C1 As Integer
  Dim C2 As Integer
```

```
Dim C3 As Integer
Dim C4 As Integer
Dim C5 As Integer
Dim C6 As Integer
Dim C7 As Integer
Dim Cf As Integer
Dim Cc As Integer
Dim bC0 As Integer
Dim bC1 As Integer
Dim bC2 As Integer
Dim bC3 As Integer
Dim bC4 As Integer
Dim bC5 As Integer
Dim bC6 As Integer
Dim bC7 As Integer
Dim bCf As Integer
Dim bCc As Integer
Dim f As Integer
If CARRYFLAG = True Then
    Cc = 1
Else
    Cc = 0
End If
C0 = vbrcGetBitB(a, 0)
C1 = vbrcGetBitB(a, 1)
C2 = vbrcGetBitB(a, 2)
C3 = vbrcGetBitB(a, 3)
C4 = vbrcGetBitB(a, 4)
C5 = vbrcGetBitB(a, 5)
C6 = vbrcGetBitB(a, 6)
C7 = vbrcGetBitB(a, 7)
bC0 = C0
bC1 = C1
bC2 = C2
bC3 = C3
bC4 = C4
bC5 = C5
bC6 = C6
bC7 = C7
bCc = Cc
'ชิฟขวา
C0 = bC1
C1 = bC2
C2 = bC3
C3 = bC4
C4 = bC5
C5 = bC6
C6 = bC7
```

```
    C7 = bCc
    CARRYFLAG = 0
    'PARITY FLAG CHECK'
    f = C0 + C1 + C2 + C3 + C4 + C5 + C6 + C7
    If (f / 2 = 1) Or (f / 4 = 1) Or (f / 6 = 1) Or (f / 8 = 1) Then
        PARITYOVERFLOW = True
    Else
        PARITYOVERFLOW = False
    End If
    'CONVERT BINARY TO INTEGER'
    Cf = ((2 ^ 7 * C7) + (2 ^ 6 * C6) + (2 ^ 5 * C5) + (2 ^ 4 * C4) + (2 ^ 3 * C3) + (2 ^ 2 * C2) + 2 * C1 + 1 * C0)
    NEGATIVEFLAG = False
    HALFCARRYFLAG = False
    Flag = FLAGSET
    SRL = Cf
End Function
Public Sub CCFold()
    NEGATIVEFLAG = False
    If CARRYFLAG = False Then
        CARRYFLAG = True
    Else
        CARRYFLAG = False
    End If
    Flag = FLAGSET
    End Sub
Public Sub SCFold()
    NEGATIVEFLAG = False
    HALFCARRYFLAG = False
    CARRYFLAG = True
    Flag = FLAGSET
End Sub
Public Sub CPLold()
    Dim a As Byte
    a = Val("&H" & GeneralRegist(0))
    HALFCARRYFLAG = True
    NEGATIVEFLAG = True
    Flag = FLAGSET
    vbrcToggleBitB a, 0
    vbrcToggleBitB a, 1
    vbrcToggleBitB a, 2
    vbrcToggleBitB a, 3
    vbrcToggleBitB a, 4
    vbrcToggleBitB a, 5
    vbrcToggleBitB a, 6
    vbrcToggleBitB a, 7
    If Len(a) < 2 Then
        GeneralRegist(0) = "0" & Hex(a)
    Else
```

```
        GeneralRegis(0) = Hex(a)
    End If
    Main.LbGRegis(0).Caption = "A = " & GeneralRegis(0) & "H"
End Sub
Public Sub NEG()
    Dim a As Byte
    a = Val("&H" & GeneralRegis(0))
    vbrcToggleBitB a, 0 'complement's 1
    vbrcToggleBitB a, 1
    vbrcToggleBitB a, 2
    vbrcToggleBitB a, 3
    vbrcToggleBitB a, 4
    vbrcToggleBitB a, 5
    vbrcToggleBitB a, 6
    vbrcToggleBitB a, 7
    a = a + 1 'complement 's 2
    ' OVER FLOW CHECK'
    If (a > 255) Then
        a = a - 256
        PARITYOVERFLOW = True
    Else
        PARITYOVERFLOW = False
    End If
    'HALF CARRY CHECK
    If (a > 127) Or (a < -128) Then
        HALFCARRYFLAG = True
    Else
        HALFCARRYFLAG = False
    End If
    'SIGN CHECK
    If vbrcGetBitB(a, 7) = 0 Then
        SIGNFLAG = False
    Else
        SIGNFLAG = True
    End If
    'ZERO CHECK
    If a = 0 Then
        ZEROFLAG = True
    Else
        ZEROFLAG = False
    End If
    NEGATIVEFLAG = True
    Flag = FLAGSET
     If Len(a) < 2 Then
        GeneralRegis(0) = "0" & Hex(a)
     Else
        GeneralRegis(0) = Hex(a)
     End If
```

```
    Main.LbGRegis(0).Caption = "A = " & GeneralRegis(0) & "H"
End Sub
Public Function CP(SOURCE As String)
   Dim a, b As Byte
   a = Val("&H" & GeneralRegis(0))
   b = Val("&H" & SOURCE)
   If (a And b) = a Then
      ZEROFLAG = True
   Else
      ZEROFLAG = False
   End If
   NEGATIVEFLAG = True
   Flag = FLAGSET
End Function
Public Function INCold(DEST As String)
Dim pos, row, col, Strlen As Integer
Dim BUF As String
   If Len(Hex(Val("&H" & SOURCE) + 1)) < 2 Then
      BUF = "0" & Hex(Val("&H" & SOURCE) + 1)
   Else
      BUF = Hex(Val("&H" & SOURCE) + 1)
   End If
   Select Case DEST
      Case "A"
            GeneralRegis(0) = BUF
            Main.LbGRegis(0).Caption = "A = " & GeneralRegis(0) & "H"
      Case "B"
            GeneralRegis(1) = BUF
            Main.LbGRegis(1).Caption = "B = " & GeneralRegis(1) & "H"
      Case "C"
            GeneralRegis(2) = BUF
            Main.LbGRegis(2).Caption = "C = " & GeneralRegis(2) & "H"
      Case "D"
            GeneralRegis(3) = BUF
            Main.LbGRegis(3).Caption = "D = " & GeneralRegis(3) & "H"
      Case "E"
            GeneralRegis(4) = BUF
            Main.LbGRegis(4).Caption = "E = " & GeneralRegis(4) & "H"
      Case "H"
            GeneralRegis(6) = BUF
            Main.LbGRegis(6).Caption = "H = " & GeneralRegis(6) & "H"
      Case "L"
            GeneralRegis(7) = BUF
            Main.LbGRegis(7).Caption = "L = " & GeneralRegis(7) & "H"
     Case "BC"
            GeneralRegis(1) = Left(BUF, 2)
            GeneralRegis(2) = Right(BUF, 2)
            Main.LbGRegis(1).Caption = "B = " & GeneralRegis(1) & "H"
```

```
            Main.LbGRegis(2).Caption = "C = " & GeneralRegis(2) & "H"
    Case "DE"
            GeneralRegis(3) = Left(BUF, 2)
            GeneralRegis(4) = Right(BUF, 2)
            Main.LbGRegis(3).Caption = "D = " & GeneralRegis(3) & "H"
            Main.LbGRegis(4).Caption = "E = " & GeneralRegis(4) & "H"
    Case "HL"
            GeneralRegis(6) = Left(BUF, 2)
            GeneralRegis(7) = Right(BUF, 2)
            Main.LbGRegis(6).Caption = "H = " & GeneralRegis(6) & "H"
            Main.LbGRegis(7).Caption = "L = " & GeneralRegis(7) & "H"
      Case "(HL)"
            pos = Val("&H" & GeneralRegis(6) & GeneralRegis(7))
            If pos < 0 Then pos = 65536 + pos
            MEMory(pos) = BUF
            row = Val("&H" & "0" & GeneralRegis(6))
            col = Val("&H" & "0" & GeneralRegis(7))
            Main!MSFlexGrid1.row = row + 1
            Main!MSFlexGrid1.col = col + 1
            Main!MSFlexGrid1.Text = MEMory(pos)
  End Select
  If (SOURCE + 1) > 255 Then
      PARITYOVERFLOW = True
  Else
      PARITYOVERFLOW = False
  End If
  NEGATIVEFLAG = False
  Flag = FLAGSET
End Function
Public Function DECOLD(DEST As String)
  Dim pos, row, col, Strlen, BFF As Integer
  Dim BUF As String
  BFF = Val("&H" & SOURCE) - 1
  If BFF < 0 Then BFF = 256 + BFF
  If Len(Hex(BFF)) < 2 Then
      BUF = "0" & Hex(BFF)
  Else
    BUF = Hex(BFF)
  End If
  Select Case DEST
    Case "A"
            GeneralRegis(0) = BUF
            Main.LbGRegis(0).Caption = "A = " & GeneralRegis(0) & "H"
    Case "B"
            GeneralRegis(1) = BUF
            Main.LbGRegis(1).Caption = "B = " & GeneralRegis(1) & "H"
    Case "C"
            GeneralRegis(2) = BUF
```

```
            Main.LbGRegis(2).Caption = "C = " & GeneralRegis(2) & "H"
    Case "D"
            GeneralRegis(3) = BUF
            Main.LbGRegis(3).Caption = "D = " & GeneralRegis(3) & "H"
    Case "E"
            GeneralRegis(4) = BUF
            Main.LbGRegis(4).Caption = "E = " & GeneralRegis(4) & "H"
    Case "H"
            GeneralRegis(6) = BUF
            Main.LbGRegis(6).Caption = "H = " & GeneralRegis(6) & "H"
    Case "L"
            GeneralRegis(7) = BUF
            Main.LbGRegis(7).Caption = "L = " & GeneralRegis(7) & "H"
    Case "BC"
        ' If SourceN <> "(adr)" Then
            GeneralRegis(1) = Left(BUF, 2)
            GeneralRegis(2) = Right(BUF, 2)
            Main.LbGRegis(1).Caption = "B = " & GeneralRegis(1) & "H"
            Main.LbGRegis(2).Caption = "C = " & GeneralRegis(2) & "H"
    Case "DE"
            GeneralRegis(3) = Left(BUF, 2)
            GeneralRegis(4) = Right(BUF, 2)
            Main.LbGRegis(3).Caption = "D = " & GeneralRegis(3) & "H"
            Main.LbGRegis(4).Caption = "E = " & GeneralRegis(4) & "H"
    Case "HL"
            GeneralRegis(6) = Left(BUF, 2)
            GeneralRegis(7) = Right(BUF, 2)
            Main.LbGRegis(6).Caption = "H = " & GeneralRegis(6) & "H"
            Main.LbGRegis(7).Caption = "L = " & GeneralRegis(7) & "H"
     Case "(HL)"
            pos = Val("&H" & GeneralRegis(6) & GeneralRegis(7))
            If pos < 0 Then pos = 65536 + pos
            MEMory(pos) = BUF
            row = Val("&H" & "0" & GeneralRegis(6))
            col = Val("&H" & "0" & GeneralRegis(7))
            Main!MSFlexGrid1.row = row + 1
            Main!MSFlexGrid1.col = col + 1
            Main!MSFlexGrid1.Text = MEMory(pos)
    End Select
    If (SOURCE - 11) < 0 Then
        PARITYOVERFLOW = True
    Else
        PARITYOVERFLOW = False
    End If
    NEGATIVEFLAG = False
    Flag = FLAGSET
End Function
```

```
Public Function DAAodl(SOURCE As String)
    Dim a, ah, al, H As Byte
    Dim BUF, BUF1 As String


    a = Val("&H" & SOURCE)
    al = Val("&H" & Right(SOURCE, 1))

    If (al > 9 Or HALFCARRYFLAG = True) Then al = al + 6
    If al > 15 Then
            HALFCARRYFLAG = True
            H = 1
    Else
            HALFCARRYFLAG = True
            H = 0
     End If
            BUF = Right(Hex(al), 1)
            ah = Val("&H" & Left(SOURCE, 1))
            ah = ah + H
            If (ah > 9) Then ah = ah + 6
              If ah > 15 Then
                 CARRYFLAG = True
              Else
                 CARRYFLAG = False
              End If

             BUF1 = Right(Hex(ah), 1)
            BUF = BUF1 & BUF
            If Val(BUF) = 0 Then
              ZEROFLAG = True
            Else
              ZEROFLAG = False
            End If
             If vbrcGetBitB(Val(BUF), 7) Then
              NEGATIVEFLAG = True
            Else
              NEGATIVEFLAG = False
            End If
            If Len(BUF) = 1 Then BUF = "0" & BUF
            GeneralRegis(0) = BUF
            Main.LbGRegis(0) = "A = " & GeneralRegis(0) & "H"
            Flag = FLAGSET
End Function

Public Function ld16(inlong As Long) As Long
   ld16 = inlong
End Function
```

```
Public Function in8(inbyte As Byte) As Byte
   in8 = inbyte
End Function
Public Function OUT8(outbyte As Byte) As Byte
   OUT8 = outbyte
End Function
```

**รูปที่ ก.3** โปรแกรมการจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80

# ภาคผนวก ข

# ใบงานการทดลอง

# ใบงานที่ 1
# คำสั่งการโอนย้ายข้อมูล

## วัตถุประสงค์

1) เพื่อให้เข้าใจการใช้คำสั่งเกี่ยวกับการโอนย้ายข้อมูล

2) รู้จักการใส่ค่าเริ่มต้นของรีจิสเตอร์ต่าง ๆ

3) รู้จักการป้อนข้อมูล และคำสั่งให้โปรแกรมจำลองการทำงานไมโครโปรเซสเซอร์ Z-80

## ทฤษฎี

การโอนย้ายข้อมูลถือเป็นความสามารถที่สำคัญที่สุด การที่คอมพิวเตอร์สามารถประมวลผลโปรแกรมได้รวดเร็วก็ขึ้นอยู่กับความสามารถในการโอนย้ายข้อมูลนั่นเอง การโอนย้ายข้อมูลในไมโครโปรเซสเซอร์นั้นสามารถแบ่งได้เป็น 5 ชนิดคือ

1) การโอนย้ายข้อมูลระหว่างรีจิสเตอร์ด้วยกัน

2) การโอนย้ายข้อมูลระหว่างรีจิสเตอร์กับหน่วยความจำ

3) การโอนย้ายข้อมูลระหว่างรีจิสเตอร์กับหน่วยอินพุต เอาต์พุต

4) การโอนย้ายข้อมูลระหว่างรีจิสเตอร์กับข้อมูลที่กำหนดขึ้น

5) การโอนย้ายข้อมูลระหว่างหน่วยความจำกับหน่วยอินพุต เอาต์พุต

6) การโอนย้ายข้อมูลระหว่างหน่วยความจำกับข้อมูลที่กำหนดขึ้น

**1. การโอนย้ายข้อมูลระหว่างรีจิสเตอร์**

ภายใน Z-80 จะประกอบด้วยรีจิสเตอร์ขนาด 8 บิต 18 ตัวคือ A, F, B, C, D, E, H, L, A', F', B', C', D', E', H', L', R และขนาด 16 บิตอีก 4 ตัวคือ IX, IY, SP, PC ซึ่งในแต่ละตัวจะทำหน้าที่ในการเก็บข้อมูลชั่วคราว ใน Z-80 นี้จะใช้คำสั่ง LD การที่จะเคลื่อนย้ายข้อมูลระหว่างรีจิสเตอร์นั้น ๆ ในคำสั่งต่าง ๆ นั้นจะใช้ตัวกระทำ (operand) 2 ตัว คือ ซอร์ท (SOURCE) หมายถึง ตำแหน่งข้อมูลเดิมที่จะถูกย้ายและตำแหน่งที่ข้อมูลเดิมจะถูกย้ายไปเก็บ หรือรับข้อมูลจะเรียกว่า "DESTINATION" รูปแบบของคำสั่งกลุ่มนี้คือ LD r, s

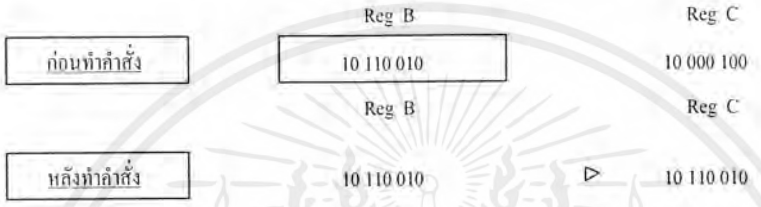
r คือ รีจิสเตอร์ต่าง ๆ ของ Z-80 ที่เป็นตัวรับข้อมูล (DESTINATION)

s คือ รีจิสเตอร์ต่าง ๆ ของ Z-80 ที่จะเป็นตำแหน่งที่จะถูกย้ายไป

สมมุติว่าต้องการนำข้อมูลในรีจิสเตอร์ B ไปเก็บไว้ในรีจิสเตอร์ C สามารถเขียนได้ดังนี้ หลังจากใช้คำสั่งแล้วข้อมูลเดิมในรีจิสเตอร์ C จะถูกแทนที่ด้วยข้อมูลในรีจิสเตอร์ B และข้อมูลในรีจิสเตอร์ B จะเหมือนเดิมไม่เปลี่ยนแปลง

ข้อควรระวัง คือ ลำดับการเขียนรีจิสเคอร์ LD C, B รีจิสเตอร์ C จะมาก่อนรีจิสเตอร์ B แต่มีความหมายว่านำเอาข้อมูลในรีจิสเตอร์ B ไปเก็บไว้ในรีจิสเตอร์ C

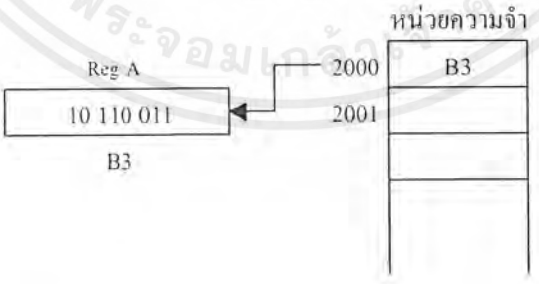
(B) = > (C)




**รูปที่ ข.1** การทำคำสั่ง LD C, B

**2. การโอนย้ายข้อมูลระหว่างรีจิสเตอร์กับหน่วยความจำ**

การโอนย้ายข้อมูลแบบนี้จะทำได้ 2 วิธีคือ การย้ายในรีจิสเตอร์ไปเก็บในหน่วยความจำ และการย้ายข้อมูลจากหน่วยความจำไปเก็บในรีจิสเตอร์ ซึ่งการโอนย้ายด้วยวิธีนี้คล้ายคลึงกับวิธีแรกแต่แทนที่จะเป็นรีจิสเตอร์ ก็เปลี่ยนเป็นแอดเดรสของหน่วยความจำตำแหน่งใดตำแหน่งหนึ่ง เช่น LD A, (2000H) หมายความว่า ข้อมูลที่อยู่ในแอดเดรส 2000H จะถูกนำไปเก็บในรีจิสเตอร์ A




**รูปที่ ข.2** การทำคำสั่ง LD A, (2000H)

**3. การโอนย้ายข้อมูลระหว่างรีจิสเตอร์กับหน่วยอินพุต เอาต์พุต**

การโอนย้ายข้อมูลในลักษณะนี้จะใช้คำสั่งต่างจากที่ได้ใช้มาแล้วคือ ใช้คำสั่ง IN, OUT ซึ่งจะทำหน้าที่อ่านข้อมูลจาก PORT ที่เรากำหนดมาเก็บไว้ในรีจิสเตอร์ หรือนำข้อมูลจากรีจิสเตอร์ออกไปยัง PORT ที่ต้องการเช่น ต้องการอ่านข้อมูลจาก PORT เบอร์ 63H เข้ามาในรีจิสเตอร์ A จะทำได้ดังนี้ IN A, (63H) ดังนั้นหลังจากทำคำสั่งนี้แล้วข้อมูลที่อยู่ใน PORT เบอร์ 63H จะเข้ามาเก็บในรีจิสเตอร์ A ทันที

การโอนย้ายข้อมูลโดยวิธีนี้สามารถกระทำเป็นบล็อกได้โดยใช้คำสั่ง INI, INIR ซึ่งคำสั่ง INI ย่อมาจากคำสั่ง Input และ Increment คือ หมายเลขพอร์ตจะถูกกำหนดโดยรีจิสเตอร์ C และตำแหน่งที่เก็บจะถูกชี้โดย HL โดยเมื่อทำคำสั่งเสร็จแล้ว B = B+1 และ HL = HL+1 ส่วนคำสั่ง INIR นั้นมาจาก Input Increment Repeat ซึ่งคล้ายกับคำสั่ง INI แต่เพิ่มการตรวจสอบว่า B = 0 หรือยัง ถ้ายังจะกลับไปอินพุต ใหม่จนกว่ารีจิสเตอร์ B = 0 สรุปว่า

1) ฟังก์ชันของ INI คือ (HL) <= (C); B<=B-1; HL<HL+1

2) ฟังก์ชันของ INIR คือ (HL) <= (C); B<=B-1; HL<HL+1

3) ทำซ้ำจนกระทั่งรีจิสเตอร์ B = 0

**4. การโอนย้ายข้อมูลระหว่างหน่วยความจำกับอินพุต เอาต์พุต**

การโอนย้ายลักษณะนี้ไม่สามารถที่จะกระทำตรง ๆ กับหน่วยความจำได้เพราะคำสั่งในตัวไมโครโปรเซสเซอร์ Z-80 ไม่ได้มีไว้ให้ แต่สามารถกระทำแบบอินไดเร็ค (Indirect) โดยจะผ่านทางรีจิสเตอร์ได้ เช่น ต้องการอ่านข้อมูลจากหมายเลข 63H มาเก็บไว้ในหน่วยความจำแอดเดรส 1000H สามารถทำได้ดังนี้

```
IN   A,63H          ; อ่านค่าในพอร์ทเบอร์ 63H เก็บใน A
LD  (2000H),A       ; ค่าใน A เก็บในตำแหน่ง 2000H
```

**รูปที่ ข.3** โปรแกรมการโอนย้ายข้อมูลระหว่างหน่วยความจำกับอินพุต

**5. การโอนย้ายข้อมูลระหว่างรีจิสเตอร์กับข้อมูลที่กำหนดขึ้น**

การโอนย้ายข้อมูลโดยวิธีนี้เป็นการอ้างแอดเดรสแบบตรง (Direct Addressing) คือเราสามารถทำการนำค่าอะไรไปใส่ในรีจิสเตอร์ไหนก็ได้ในทั้งแบบ 8 บิตและแบบ 16 บิต เช่น

8 บิต เช่น LD A, 86H

16 บิต เช่น LD HL, 1234H

**6. การโอนย้ายข้อมูลระหว่างหน่วยความจำกับข้อมูลที่กำหนดขึ้น**

การโอนย้ายวิธีนี้คล้ายกับการโอนย้ายระหว่างรีจิสเตอร์กับข้อมูลที่กำหนดขึ้น เพียงแต่แทนที่จะนำไปเก็บในรีจิสเตอร์ก็นำเอาไปเก็บในหน่วยความจำแทน เช่น LD (HL), 36H

## สรุป

จากที่กล่าวมาแล้วจะพบว่า คำสั่งในไมโครโปรเซสเซอร์ Z-80 สามารถที่จะทำการโอนย้ายข้อมูลได้หลายวิธีเช่นการโอนย้ายข้อมูลระหว่างรีจิสเตอร์กับรีจิสเตอร์ หรือรีจิสเตอร์กับหน่วยความจำเป็นต้น คำสั่งในการโอนย้ายข้อมูลเกือบทุกครั้งจะใช้คำว่า "LD" เป็นตัวกระทำ (Operand) จะยกเว้นก็เฉพาะคำสั่งที่เกี่ยวกับหน่วย INPUT, OUTPUT เท่านั้นที่ใช้คำสั่ง IN, OUT

## ลำดับขั้นการทดลอง

1) เขียนโปรแกรมภาษา Assembly เพื่อทำการกำหนดข้อมูลในรีจิสเตอร์ต่าง ๆ ดังต่อไปนี้ A=00H, B=01H, C=02H, D=03H, E=04H, โดยใช้คำสั่ง LD แบบ 8 บิต

1.1) เขียนโปรแกรมภาษา Assembly และ Machine Code ลงในช่องว่าง

| Memory address | Machine Code | | Assembly |
|---|---|---|---|
| 0000 | 3E | 00 | LD A,00H |
| _____ | __ | __ | _____ |
| _____ | __ | __ | _____ |
| _____ | __ | __ | _____ |
| _____ | __ | __ | _____ |
| 000A | FF | | RST 38H |

1.2) ป้อนโปรแกรมเข้าโปรแกรมการจำลองการทำงานของไมโครเซสเซอร์ Z-80 ซึ่งต้องเขียนในหน้าต่างอิดิเตอร์

1.3) ทำการรันโปรแกรมการทำงานส่วนภาคผนวก และทำการรันโปรแกรม

1.4) ทำการตรวจสอบรีจิสเตอร์ต่าง ๆ ว่าตรงกับความเป็นจริงหรือไม่

2) เขียนโปรแกรมภาษา Assembly เพื่อทำการกำหนดข้อมูลในรีจิสเตอร์ต่าง ๆ ให้สามารถทำงานได้เหมือนกับโปรแกรมข้อ 1.11 แต่ใช้คำสั่งโอนย้ายข้อมูลแบบ 16 บิต

2.1) เติมโปรแกรมลงในช่องว่างข้างล่าง

| Memory address | Machine code | Assembly |
|---|---|---|
| 0000 | 3E 00 | LD A,00H |
| ——— | —— —— | ——————— |
| ——— | —— —— | ——————— |
| 0008 | FF | RST 38H |

2.2) ทำการ Execute Program และตรวจสอบข้อมูลว่า ถูกต้องหรือไม่ ข้อมูล 16 บิต จะประกอบด้วยข้อมูล 2 ไบต์ โดยมีไบต์สูง (High Order Byte) จะเก็บที่แอดเดรสสูงและไบต์ต่ำ (Low Order Byte) จะถูกเก็บที่แอดเดรสต่ำกว่า เช่น ข้อมูล 00FF ถูกเก็บที่หน่วยความจำตั้งแต่ 0000H จะได้เป็นดังนี้

0000H จะมีข้อมูล FFH

0001H จะมีข้อมูล 00H

3) ทำการทดลองโดยการป้อนโปรแกรมตามที่กำหนด ลงในหน้าต่างอิดิเตอร์ แล้วทำการแอสเซมเบลอ ดูผลของการแอสเซมเบลอในหน้าต่าง แล้วนำมาบันทึกลงในผลการทดลอง

| **Address** | **Machine Code** | **Operand** |
|---|---|---|
| 0000 | 3E 01 | LD A,01H |
| 0002 | 06 20 | LD B, 20H |
| 0004 | 0E 60 | LD C, 60H |
| 0006 | 26 20 | LD H, 20H |
| 0008 | 2E 61 | LD L, 61H |
| 000A | DD5600 | LD D, (IX+00H) |
| 000D | DD7005 | LD (IY+05), B |
| 0010 | 74 | LD (HL), H |
| 0011 | 5E | LD E, (HL) |
| 0012 | DD36105E | LD (IX+10H), 5EH |
| 0016 | 0A | LD A, (BC) |
| 0017 | 326220 | LD (2062H), A |
| 001A | 5F | LD E, A |
| 001B | FF | RST 18H |

4) ทำการรันโปรแกรมในข้อที่ 3 แล้วดูผลการทดลอง โดยในการรันนั้นทำการเลือกการรันแบบ Step เพื่อดูการเปลี่ยนแปลงในแต่ละรีจิสเตอร์ แล้วอธิบายการทำงานของโปรแกรม

## สรุปผลการทดลอง

## คำถามท้ายการทดลอง

1) ในการอ้างตำแหน่งข้อมูลแบบอินเด็กซ์ สามารถระบุตำแหน่งของข้อมูลห่างจากตำแหน่งฐานได้เท่าใด

2) ในการโอนย้ายข้อมูลแบบใดที่ไม่ได้ใช้คำสั่ง LD (Load)

# ใบงานที่ 2
# คำสั่งเกี่ยวกับคณิตศาสตร์ และการเปรียบเทียบข้อมูล

## วัตถุประสงค์

1) เพื่อให้เข้าถึงวิธีการในการอ้างแอดเดรสแบบต่าง ๆ ในไมโครโปรเซสเซอร์ Z-80
2) เพื่อให้เข้าใจความหมายของ FLAG ต่าง ๆ ที่มีอยู่ในในไมโครโปรเซสเซอร์ Z-80
3) เพื่อให้เข้าใจเกี่ยวกับคำสั่งทางคณิตศาสตร์และคำสั่งตรรกะ
4) สามารถนำคำสั่งต่าง ๆ มาเขียนเป็นโปรแกรมทางคณิตศาสตร์ได้
5) เข้าใจเทคนิคการเขียนโปรแกรมบวก และลบเลขไบนารีได้

## ทฤษฎี

**การอ้าง Address ของ Z-80 (Addressing Mode)**

ภายในในไมโครโปรเซสเซอร์ Z-80 มีรีจิสเตอร์หลายตัวการทำงานของ Z-80 จึงคล่องตัวมาก การที่จะใช้ในไมโครโปรเซสเซอร์ Z-80 เป็นระบบที่สมบูรณ์ จะต้องมีการติดต่อกับหน่วยความจำและอุปกรณ์อินพุต และเอาต์พุต ในการติดต่อนี้ในไมโครโปรเซสเซอร์ Z-80 จะต้องมีการเรียก แอดเดรสที่ต้องการซึ่งเรียกว่าโหมดของการอ้างแอดเดรส (Addressing Mode) ซึ่งการอ้างแอดเดรสของในไมโครโปรเซสเซอร์ Z-80 มีหลายวิธีแบ่งออกได้ดังนี้

1) การอ้างแอดเดรสแบบอิมมิเดียท (Immediate Address) ในโหมดนี้จะประกอบด้วยออบโค้ด 1-2 ไบต์ และโอเปอแรนด์ อีก 1 ไบต์ ลักษณะของคำสั่งจะเป็นดังแสดงในรูปที่ ข.4

| OPCODE |
| --- |
| OPERAND |

**รูปที่ ข.4** ลักษณะของคำสั่งการอ้างแอดเดรสแบบอิมมิเดียท

ตัวอย่างการอ้างแอดเดรสแบบนี้ คือ การโหลดข้อมูลค่าคงที่เข้าไปเก็บไว้ในรีจิสเตอร์ หรือหน่วยความจำ เช่น

```
LD      A,0FH
```

2) การขยายตัวโอเปอแรนด์ ในกลุ่มคำสั่งการอ้างแอดเดรสแบบอิมมิเดียท (Immediate) การอ้างแอดเดรสแบบนี้จะเหมือนกับแบบแรก แตกต่างกันเพียง Operand เป็นข้อมูลขนาด 16 บิต ลักษณะของคำสั่งจะเป็นดังนี้

| Opcode |
|---|
| Operand |
| Operand |

**รูปที่ ข.5** ลักษณะของคำสั่งการขยายโอเปอแรนด์ในการแอสเดรสแบบอิมมิเดียท

ตัวอย่างเช่น

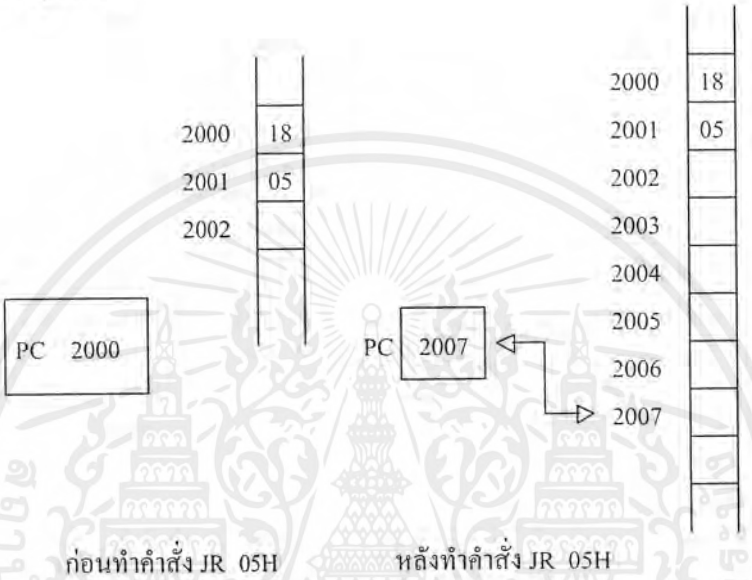
```
LD      HL,1000H
LD      ED,2000H
```

3) การอ้างแอดเดรสแบบ Modify Page ใน Z-80 มีคำสั่งพิเศษอยู่คำสั่งหนึ่ง คือ คำสั่ง RST หรือ RESTART ลักษณะการทำงานของคำสั่งคือมันสามารถที่จะกำหนดให้โปรแกรมเคาน์เตอร์ได้โดยตรง คำสั่งจะเป็นค่าแอดเดรส และโปรแกรมถูกนำค่าใหม่เข้าไปใส่ตามคำสั่ง RST แต่ละตัวการอ้างแบบนี้มีเงื่อนไขที่จำกัดคือ จะสามารถกระโดดไปได้ เพียง 8 แอดเดรสดังแสดงในตารางที่ ข.1

**ตารางที่ ข.1** แอสเดรสที่คำสั่ง RST กระไปทำงาน

| คำสั่ง RST | แอดเดรสที่กระโดดไปทำ |
|---|---|
| RST 0 | 00 H |
| RST 1 | 08 H |
| RST 2 | 10 H |
| RST 3 | 18H |
| RST 4 | 20 H |
| RST 5 | 28 H |
| RST 6 | 30 H |
| RST 7 | 38 H |

4) การอ้างแอดเดรสแบบเปรียบเทียบ (Relative Addressing) การอ้างแอดเดรสแบบนี้จะใช้ข้อมูลไบต์ที่อยู่ตามหลังออปโค้ด เพื่อจะบอกว่าตำแหน่งแอดเดรสที่อ้างถึงนั้นอยู่ห่างจากโปรแกรมเคาน์เตอร์ไปเท่าใด ซึ่งตำแหน่งที่ห่างนี้จะเรียกว่า ออฟเซต (Offset) ซึ่งค่าออฟเซตจะมีค่าเป็นตัวเลข 2' Complement ค่าที่อ้างจะอยู่ระหว่าง +127 ถึง -128 จากตำแหน่ง pc+2 เช่น JR 05H ลักษณะของคำสั่งจะเป็นดังรูปที่ ข.6




**รูปที่ ข.6** การอ้างแอดเดรสแบบเปรียบเทียบ

5) การอ้างแอดเดรสแบบ Extend วิธีการนี้จะใช้ข้อมูลอยู่ 2 ไบต์ที่ตามหลังออปโค้ดเป็นตัวกำหนดค่าแอดเดรส เช่นคำสั่ง CALL nn หมายถึงการเรียกโปรแกรมย่อยที่ตำแหน่ง nn ซึ่งการอ้างแอดเดรสแบบนี้จะนำเอาตัวโอเปอแรนด์ไปเป็นแอดเดรสนั่นเอง เช่นคำสั่ง CALL 2000H

6) การอ้างแอดเดรสโดยใช้อินเดกรีจิสเตอร์ (Index Register) ใน Z-80 มี Index Register อยู่ 2 ตัว คือ IX, IY การอ้างแอดเดรสแบบนี้จะใช้ IX หรือ IY เป็น Base รวมกับค่าที่ตามหลังออปโค้ดเป็นแอดเดรสที่ต้องการ ค่าที่ตามหลังออปโค้ดจะเป็นค่าของเลข 2' Complement คือ 127 ถึง -128 เช่น LD (IX+3),A ค่าของ A จะถูกนำไปเก็บในตำแหน่งที่ถูกใช้โดย (IX+3)

7) การอ้างแอดเดรสแบบใช้รีจิสเตอร์ (Register Address) การอ้างแบบนี้ใช้รีจิสเตอร์เป็นตัวอ้างแอดเดรสเช่น ADD A,B

8) การอ้างแอดเดรสแบบ Implied Addressing การอ้างแอดเดรสแบบนี้ซีพียูจะตีความหมายเองโดยอัตโนมัติว่ารีจิสเตอร์ในซีพียูตัวหนึ่งจะเป็น Operand เช่น ADD A,B ซีพียูจะนำข้อมูลใน B บวกเข้าไปใน A หรือ AND 03H ซีพียูจะตีความหมายว่า นำค่า 03H ไป AND กับข้อมูล A

9) การอ้างแอดเดรสเข้าสู่บิตต่าง ๆ (Bit Addressing) ใน Z-80 มีคำสั่งพิเศษที่สามารถทำการเซต หรือรีเซต หรือทดสอบบิตใดบิตหนึ่ง เช่น RES 7,B (หมายถึง การทำการรีเซตบิตที่ 7 ของรีจิสเตอร์ B ให้เป็น 0

**แฟลกในไมโครโปรเซสเซอร์ Z-80**

หลังจากที่ซีพียูมีการทำคำสั่งเกี่ยวกับทางคณิตศาสตร์และตรรกะ แล้วผลที่ได้จากการกระทำเก็บในรีจิสเตอร์ A และจะมีผลต่อแฟลกบางตัว แฟลกต่าง ๆ เหล่านี้อยู่ในฟลิปฟลอบของไมดครโปรเซสเซอร์ Z-80 ซึ่งฟลิปฟลอบเหล่านี้ จะประกอบกันเป็นรีจิสเตอร์ตัวหนึ่ง มีชื่อเรียกว่า "แฟลกรีจิสเตอร์" ซึ่งข้อมูลในรีจิสเตอร์แฟลกนี้สามารถที่จะย้ายไปอยู่ในหน่วยความจำได้เช่นเดียวกับรีจิสเตอร์ทั่ว ๆ ไป โดยใช้คำสั่ง PUSH รายละเอียดของแฟลกต่าง ๆ มีดังต่อไปนี้

1) แฟลกตัวทด (C: Carry Flag) แฟลกตัวทดนี้จะเป็นบิตที่ใช้สำหรับในการทดซึ่งเกิดจากคำสั่งทางคณิตศาสตร์ จากข้อมูลในรีจิสเตอร์ A เช่น เมื่อมีการบวกข้อมูลขนาด 8 บิต ผลของการบวกจะกลายเป็น 9 บิต บิตที่เกิน จะถูกเก็บไว้ในแฟลกตัวทด และในทำนองเดียวกัน ถ้ามีการกระทำคำสั่งลบ และมีการขอยืมค่า แฟลกตัวนี้ก็จะถูกเซตให้เป็น "1"

2) แฟลกศูนย์ (Z: Zero Flag) ถ้าข้อมูลอยู่ในแอกคิวมูลเลเตอร์มีค่าเป็น "0" แฟลกนี้จะถูกเซตให้มีค่าเป็น "1" นอกเหนือจากนั้น แฟลกนี้จะถูกรีเซตให้มีค่าเป็น "0"

3) แฟลกเครื่องหมาย (S: Sign Flag) ในกรณีของเลขที่มีการคิดเครื่องหมาย (127 ถึง –128) แฟลกตัวนี้จะเป็นตัวแสดงว่าข้อมูลใน A เป็นเลขบวก หรือลบ คือ ถ้า S = 1 จะมีเครื่องหมายเป็นลบ ถ้า S = 0 จะมีเครื่องหมายเป็นบวก และแฟลกนี้จะไม่ถูกคำนึงถึงมีการกำหนดข้อมูลเป็นลบแบบไม่คิดเครื่องหมาย

4) แฟลกพาริตี้และค่าเกิน (P/V : Parity/Over Flow Flag) แฟลกนี้จะทำหน้าที่ 2 อย่าง คือ ในกรณีของการทำคำสั่งทางลอจิก เช่น AND, OR, XOR ผลที่ได้รับจะมีการตรวจสอบว่ามีพาริตี้คู่หรือคี่ โดยถ้าเป็นคู่แฟลกนี้จะมีการเซตค่าเท่ากับ "1"

ส่วนกรณีมีการทำคำสั่งทางคณิตศาสตร์ แฟลกนี้จะเป็นตัวบอกค่าที่เกินค่ากำหนดใน 8 บิต แบบคิดเครื่องหมาย (1 บิตเครื่องหมายรวมกับอีก 7 บิตที่เป็นขนาด) นั่นคือค่าสูงสุดที่เป็นไปได้ 127 และ –128 ถ้าค่ามากกว่า +127 หรือน้อยกว่า –128 จะมีการเซตค่าเป็น "1" ที่แฟลกนี้

5) แฟลกตัวทดช่วย (H: Half Carry) แฟลกนี้จะเป็นบิตที่ทำหน้าที่เป็นตัวทด หรือตัวยืมของเลข BCD

6) แฟลกการลบ (N: Subtract Flag) เนื่องจากในคำสั่งทางคณิตศาสตร์ของตัวเลข BCD เพื่อจะได้มีการรับรู้ การปรับค่า เมื่อกระทำ DAA (Decimal Adjust) ได้ถูกต้อง แฟลกตัวนี้เป็นตัวบอกว่าคำสั่งที่ถูกกระทำเป็นการบวก หรือลบ โดยถ้ากระทำคำสั่งลบ จะได้รับการเซตให้มีค่าเป็น "1"

ลักษณะของแฟลกรีจิสเตอร์ ประกอบด้วยแฟลกแต่ละบิตดังต่อไปนี้

| S | Z | X | H | X | P/V | N | C |
|---|---|---|---|---|---|---|---|

X หมายถึง บิตที่ไม่ได้ใช้แต่จะประกอบกันเป็นแฟลกรีจิสเตอร์ให้ครบ 8 บิต

**รูปที่ ข.7** บิตต่าง ๆ ของแฟลกรีจิสเตอร์

## กลุ่มคำสั่งทางคณิตศาสตร์ตรรกะ 8 บิต

คำสั่งกลุ่มนี้เป็นกลุ่มคำสั่งเกี่ยวกับการคำนวณทางคณิตศาสตร์ เช่น การบวก ลบ การเปรียบเทียบและการคำนวณทางตรรกะ เช่น การ AND, OR, XOR ตลอดจนการ Complement สำหรับการคำนวณจะประกอบด้วยตัวตั้ง และตัวคำนวณ โดยตัวตั้งจะถูกเก็บไว้ในรีจิสเตอร์ A ส่วนตัวคำนวณ อาจจะมาจากรีจิสเตอร์ตัวใดตัวหนึ่งจากค่าตัวที่เขียนเอาไว้ในคำสั่ง หรือจากค่าในหน่วยความจำก็ได้สำหรับผลการคำนวณจะถูกนำไปเก็บไว้ในแอกคิวมูเลเตอร์ และสถานะต่าง ๆ ของผลลัพธ์ จะถูกนำไปเซตที่แฟลกซึ่งทำหน้าที่ต่าง ๆ ดังที่ได้กล่าวมาแล้ว ต่อไปนี้จะเป็นความหมายของคำสั่งต่าง ๆ บางตัวที่เกี่ยวกับคำสั่งทางคณิตศาสตร์และตรรกะ เช่น

**ตารางที่ ข.2** ความหมายของคำสั่งทางคณิตศาสตร์และตรรกะ

| คำสั่ง | ความหมายของคำสั่ง |
|---|---|
| ADC A,B | ค่าในรีจิสเตอร์ B ถูกบวกเข้ากับ A พร้อมกับคิดแฟลกตัวทด |
| ADD A,A | ข้อมูลในรีจิสเตอร์ A ถูกบวกเข้ากับตัวมันเอง ผลลัพธ์เก็บเอาไว้ในรีจิสเตอร์ A |
| SUB C | ข้อมูลใน A ถูกนำไปลบออกจาก A ผลลัพธ์ที่ได้เก็บไว้ใน A (ไม่คิดแฟลกตัวทด) |
| AND B | ข้อมูลใน B ถูกนำไปลบออกจาก A คิดแฟลกตัวทดผลลัพธ์เก็บไว้ในรีจิสเตอร์ A |
| OR 0FFH | นำค่า FFH ไป OR กับ A เก็บค่าใน A ซึ่งจะมีผลต่อแฟลกเครื่องหมาย และศูนย์ |
| CP (HL) | เปรียบเทียบข้อมูลตำแหน่งที่ถูกชี้โดย HL กับA ข้อมูลใน A ไม่เปลี่ยนแปลง แต่จะมีผลต่อแฟลกเครื่องหมาย แฟลกศูนย์ แฟลกช่วยทด และแฟลกตัวทด |
| INC B | เพิ่มค่าให้ B ทีละ 1 มีผลต่อแฟลกแฟลกเครื่องหมาย แฟลกศูนย์ แฟลกช่วยทด |
| DEC (IX+3) | ข้อมูลของตำแหน่งที่ถูกชี้โดย IX+3 จะถูกลดค่าลงหนึ่ง |

**การบวกลบเลขไบนารี่**

ในส่วนนี้จะกล่าวถึงการบวกและการลบเลขจำนวนเต็ม BINARY แบบไม่คิดเครื่องหมาย (Unsign Binary Integer) ซึ่งถ้าผลลัพธ์ของการบวกมีค่ามากกว่าค่าสูงสุดของเลขจำนวน n Bit จะเกิดตัวทด หรือแฟลกตัวทดจะถูกเซต และในการลบ ถ้าตัวลบมีค่ามากกว่าตัวตั้ง จะเกิดการยืม หรือแฟลกตัวทดจะเซตเป็นหนึ่งซึ่งแสดงว่าผลลัพธ์ที่ได้ไม่ถูกต้อง

ตัวอย่างการบวกเลขแบบไบนารี่ สมมุติ 7FH + ADH

```
      0111   1111   (7FH)    ค่าผลลัพธ์จะเป็น 012CH ซึ่งเป็นค่าที่ถูกต้อง
      1010   1101   (ADH)
[1]   0010   1100   (012CH)
Carry
```

สมมุติ ถ้าเปลี่ยนกลับเป็นการลบเลขแบบไบนารี่ 7FH - ADH

```
      0111   1111   (7FH)
      1010   1101   (ADH)      เกิดการยืม ผลลัพธ์ที่ได้ไม่ถูกต้อง
[1]          1101   (0010)
Carry
```

สมมุติ ถ้าเปลี่ยนกลับเป็นการลบเลขแบบไบนารี่ ADH - FH

```
      1010   1101   (ADH)
      0111   1111   (FH)              ผลลัพธ์ที่ได้ถูกต้อง
[0]   0010   1110
Carry
```

## ลำดับขั้นการทดลอง

1) โปรแกรมต่อไปนี้ ใช้บวกลบ ข้อมูลในรีจิสเตอร์ B กับ รีจิสเตอร์ C เข้าด้วยกัน ผลที่ได้เก็บใน DE ทำการเปิดหน้าต่างอิดิเตอร์ขึ้นมาแล้วทำการป้อนโปรแกรม และบันทึกไฟล์ จากนั้นทำการแอสเซมเบลอโปรแกรม แล้วทำรันโปรแกรมบันทึกผลการทดลองลงในตารางที่ ข.3

| Address | Machine Code | Label | Operand | Comment |
|---|---|---|---|---|
| 0000 | 78 | | LD A, B | ; นำ (B) => เก็บใน (A) |
| 0001 | 81 | | ADD A, C | ; (A+B) => A |
| 0002 | 5F | | LD E, A | ; ผลใน A เก็บใน E |
| 0003 | 3E 00 | | LD A, 0H | ; เคลียร์ A เป็น 0 |
| 0005 | CE 00 | | ADC A,0H | ; A+0+C |
| 0007 | 57 | | LD D, A | นำ (A) =>(D) |
| 0008 | DF | | RST 38H | |

ตารางที่ ข.3 บันทึกผลการทดลอง

| ค่าเริ่มต้นของรีจิสเตอร์ | | ผลของโปรแกรมเมื่อมีการรัน | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | รีจิสเตอร์ | ผลของแฟลก | | | | | |
| B | C | DE | Sign | Zero | Half | P/V | Neg | Carry |
| 54H | 6BH | | | | | | | |
| 8F | A0H | | | | | | | |
| 37H | 7E | | | | | | | |
| 87H | 91H | | | | | | | |

2) โปรแกรมต่อไปนี้ ใช้บวกข้อมูล 16 บิต ในคู่รีจิสเตอร์ BC กับข้อมูลในแอดเดรส 2200H-2201H ผลที่ได้เก็บใน DE ป้อนและรันโปรแกรม แล้วบันทึกผลการทดลองในตารางที่ ข.4

| Address | Machine Code | Label | Operand | Comment |
|---|---|---|---|---|
| 0000 | 3A 00 22 | | LD A, B | ; นำ (B) => เก็บใน (A) |
| 0001 | 81 | | ADD A, C | ; (A+B) => A |
| 0002 | 5F | | LD E, A | ; ผลใน A เก็บใน E |
| 0003 | 3A 01 22 | | LD A, (2201H) | ; ข้อมูลใน 2201H เก็บ A |
| 0005 | 88 | | ADC A,B | ; A+B = A |
| 0007 | 57 | | LD D, A | นำ (A) =>(D) |
| 0008 | DF | | RST 38H | |

ตารางที่ ข.4 บันทึกผลการทดลอง

| ค่าเริ่มต้นของ รีจิสเตอร์ และหน่วยความจำ | | ผลของโปรแกรมเมื่อมีการรัน | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | รีจิสตอร์ | ผลของแฟลก | | | | | |
| BC | 2200-2201H | DE | Sign | Zero | Half | P/V | Neg | Carry |
| 0354 H | 1001 H | | | | | | | |
| 116A H | 1021 H | | | | | | | |
| 3110 H | 0010 H | | | | | | | |
| 0A01 H | 3101 H | | | | | | | |

3) จากโปรแกรมในข้อ 2 เขียนโปรแกรมใหม่เพื่อให้เป็นโปรแกรมสำหรับการลบเลข ผลลัพธ์เก็บในหน่วยความจำตำแหน่ง 2202-2203H (แอดเดรสสูงเก็บไบต์สูง แอดเดรสไบต์ต่ำเก็บไบต์ต่ำ) บันทึกผลการทดลองลงในตารางที่ ข.5

ตารางที่ ข.5 บันทึกผลการทดลอง

| ค่าเริ่มต้นของ รีจิสเตอร์ และหน่วยความจำ | | ผลของโปรแกรมเมื่อมีการรัน | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | หน่วยความจำ | ผลของแฟลก | | | | | |
| BC | 2200-2201H | 2202-2203H | Sign | Zero | Half | P/V | Neg | Carry |
| 354 H | 1001 H | | | | | | | |
| 116A H | 1021 H | | | | | | | |
| 3110 H | 0010 H | | | | | | | |
| A01 H | 3101 H | | | | | | | |

## สรุปผลการทดลอง

## คำถามท้ายการทดลอง

1) Over Flow หมายถึง อะไร จะเกิดขึ้นเมื่อใด

2) แฟลกรีจิสเตอร์มีประโยชน์อย่างไร

# ใบงานที่ 3
# กลุ่มคำสั่งการกระโดด และการทำซ้ำ

## วัตถุประสงค์

1) เข้าใจการทำงานของกลุ่มคำสั่งการกระโดดแบบมีเงื่อนไข และไม่มีเงื่อนไข

2) รู้จักวิธีการสั่งให้โปรแกรมเคาน์เตอร์กระโดดไปยังตำแหน่งหน่วยความจำที่ต้องการได้ตามเงื่อนไขของแฟลกต่าง ๆ

3) เข้าใจเทคนิคการออกแบบโปรแกรมทำซ้ำลูป และหน่วงเวลา

4) ฝึกหัดการใช้คำสั่งกระโดด และโปรแกรมทำซ้ำ

## ทฤษฎี

ตามปกติคอมพิวเตอร์จะทำงานตามคำสั่งที่เขียนในโปรแกรมทีละคำสั่งเรียงลำดับกันไป แต่ในบางคำสั่งเราต้องการให้คอมพิวเตอร์ทำงานอย่างหนึ่งซ้ำ ๆ กันหลาย ๆ ครั้ง คอมพิวเตอร์สามารถทำงานดังกล่าวได้ซึ่งขึ้นอยู่กับเงื่อนไขที่ผู้เขียนตั้งโปรแกรมไว้โดยที่ไม่ต้องเขียนโปรแกรมซ้ำกันหลายครั้ง นอกจากนั้นแล้วยังมีความสามารถในการตัดสินใจที่จะทำหรือไม่ทำ โปรแกรมได้ตามเงื่อนไขที่ตั้งไว้ การที่คอมพิวเตอร์มีความสามารถดังกล่าวเนื่องมาจากมีคำสั่งเกี่ยวกับการกระโดดและการตัดสินใจนั่นเอง

คำสั่งการกระโดด (Jump Instruction) ในไมโครโปรเซสเซอร์ Z-80 มีกลุ่มคำสั่งเกี่ยวกับการกระโดดที่มีประสิทธิภาพซึ่งคำสั่งเหล่านี้แบ่งออกได้เป็น 2 ประเภท คือ

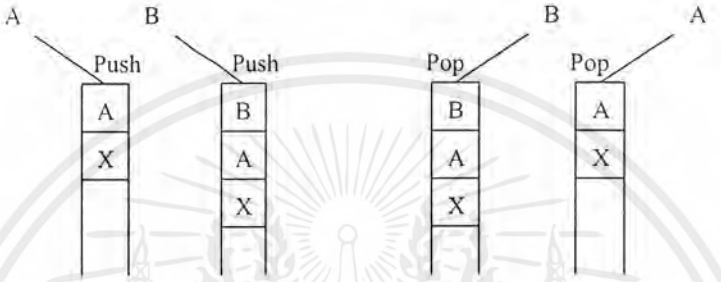
1) คำสั่งกระโดดแบบธรรมดา แบ่งได้เป็น

1.1) กระโดดแบบไม่มีเงื่อนไข การทำคำสั่งในกลุ่มนี้ ซีพียูจะถูกสั่งให้ทำโปรแกรมตามตำแหน่งที่ต้องการทันทีโดยไม่มีเงื่อนไข เช่น JP 2200H หลังจากทำคำสั่งนี้ค่าในโปรแกรมเคาน์เตอร์จะเปลี่ยนเป็นค่า 2200H

โปรแกรมเคาน์เตอร์ (PC: Program Counter) เป็นรีจิสเตอร์มีขนาด 16 บิต เป็นรีจิสเตอร์ของไมโครโปรเซสเซอร์ Z-80 ที่เป็นตัวกำหนดตำแหน่งของโปรแกรมในขณะเฟทซ์คำสั่งค่าในโปรแกรมเคาน์เตอร์จะปรากฏที่แอดเดรสบัสเพื่อจะเป็นตัวชี้ตำแหน่งไปยังหน่วยความจำที่ให้ซีพียูอ่านคำสั่งและแผลความหมายในโปรแกรมเคาน์เตอร์ จะเพิ่มค่าโดยอัตโนมัติเมื่อทำคำสั่ง ในแต่ละ

คำสั่ง เสร็จแต่ถ้าซีพียูพบคำสั่งกระโดดค่าแอดเดรสที่จะกระโดดไปนั้นจะถูกนำมาใส่ใน โปรแกรมเคาน์เตอร์

แสตคพอยน์เตอร์เป็นรีจิสเตอร์ขนาด 16 บิตใช้สำหรับเป็นตัวชี้ตำแหน่งแอดเดรสชั้นบนสุดของแสตคที่อยู่ในหน่วยความจำลักษณะของแสดคจะมีโครงสร้างแบบ เข้าก่อนออกทีหลัง กล่าวคือ การนำข้อมูลเข้าไปเก็บในแสตคจะถูกเก็บซ้อน ๆ กัน โดยข้อมูลอันแรกจะถูกนำไปเก็บไว้ชั้นล่างสุดและเมื่อต้องการนำข้อมูลที่จะใช้ออกมา ก็ต้องนำข้อมูลที่นำไปเก็บไว้หลังสุดออกมาก่อน




**รูปที่ ข.7** ลักษณะของแสตคเป็นหน่วยความจำแบบเข้าก่อนออกทีหลัง

1.2) การกระโดดแบบมีเงื่อนไข (Condition jump) จะมีการกำหนดเงื่อนไขในการกระโดดไปยังตำแหน่งที่ต้องการ ถ้าเงื่อนไขเป็นเท็จก็จะไม่กระโดดแต่จะทำคำสั่งในตำแหน่งถัดไป ซึ่งเงื่อนไขดังกล่าวจะขึ้นอยู่กับข้อมูลที่อยู่ในแฟลกรีจิสเตอร์ เช่น

JP C.2000H ถ้า Cy = "1" จะกระโดด ถ้า Cy = "0" จะทำคำสั่งถัดไป

JP Z.2500H ถ้า C = "1" หรือผลจากการทำคำสั่งก่อนมีค่าเป็น "0" จะกระโดด

2. คำสั่งการกระโดดแบบสัมพันธ์ (Jump Relative)

ในไมโครโปรเซสเซอร์ Z-80 นั้นจะมีคำสั่งการกระโดดแบบธรรมดาแล้ว ยังมีคำสั่งการกระโดดแบบที่มีประสิทธิภาพและใช้หน่วยความจำน้อยกว่า คือ จะใช้หน่วยความจำเพียง 2 ไบต์ แทนที่จะใช้ 3 ไบต์ การกระโดดแบบนี้เรียกว่า (Jump relative) โดยจะอาศัยค่า displacement เป็นตัวกำหนดตำแหน่งที่ซีพียูจะไปทำงาน ซึ่งค่านี้ก็คือ ค่าของช่วงห่างระหว่างตำแหน่งที่ซีพียูจะกระโดดและตำแหน่งที่ซีพียูจะกระโดดไป โดยค่า displacement จะเป็นเลขแบบ 2' complement แบบคิดเครื่องหมายคืออยู่ในช่วง 127 ถึง -128 ซึ่งลักษณะในการกระโดดก็จะมีแบบที่มีเงื่อนไขและแบบที่ไม่มีเงื่อนไขเหมือนกับการกระโดดแบบธรรมดา แต่จะแตกต่างกันตรงที่ การกระโดดแบบนี้ไม่สามารถใช้ Sign flag และ Parity flag ได้ ซึ่งเงื่อนไขในการกระโดดจะถูกกำหนดโดยการใช้ค่าออฟเซต ซึ่งถ้าค่าออฟเซตมีค่าเป็นบวกก็จะทำให้การกระโดดไปข้างหน้า แต่ถ้าเป็นลบก็จะทำให้กระโดดไปข้างหลัง ซึ่งค่าที่มากที่สุดที่จะกระโดดได้คือ 7FH หรือ 127

**โปรแกรมการทำงานซ้ำ (Program Loop)**

เครื่องคอมพิวเตอร์นอกจากจะมีข้อดีในการที่จะสามารถจดจำข้อมูลได้เป็นจำนวนมาก ๆ ยังมีข้อดีในการทำงานซ้ำ ๆ กันได้มากมายตามความต้องการของผู้ใช้ เช่น การคำนวณเงินเดือนของพนักงานสูตรการคำนวณเงินเดือนของพนักงานแต่ละคนย่อมไม่เหมือนกัน ดังนั้นเราจำเป็นที่จะต้องใช้โปรแกรมการทำงานซ้ำ มาช่วยในการคำนวณเพื่อให้ลดเวลาในการทำงานและลดการสิ้นเปลืองทรัพยากรและหน่วยความจำอีกด้วย

## ลำดับขั้นการทดลอง

1) ทำการทดลองตามโปรแกรม ในคำสั่งนี้เป็นการนำค่าในรีจิสเตอร์ C ใส่ในหน่วยความจำที่รีจิสเตอร์ HL ชี้โดยมีการทำซ้ำจนกว่ารีจิสเตอร์ B จะมีค่าเป็น 0 โดยดูจากแฟลกรีจิสเตอร์

| Address | Machine Code | Label | Operand | Comment |
|---|---|---|---|---|
| 0000 | 21 00 22 | START: | LD HL, 2000H | ; กำหนดตำแหน่ง |
| 0003 | 06 FF | | LD B, 10 H | ; รีจิสเตอร์ B เป็นตัวนับ |
| 0005 | 0E 11 | | LD C, 11H | ; ให้ค่าข้อมูลเป็น 11H |
| 0007 | 71 | LOOP: | LD (HL), C | ; ใส่ค่าใน C ใน (HL) |
| 0008 | 23 | | INC HL | ; HL<= HL+1 |
| 0009 | 10 FC | | DJNZ LOOP | ; B-1 วนถ้า B < > 0 |
| 000B | DF | | RST 38H | ; หยุดโปรแกรม |

2) กำหนดให้ C = 11H แล้วทำการรันโปรแกรม แล้วทำการบันทึกผลการทดลอง

ผลลัพธ์ในหน่วยความจำ 2200-220F เป็น_____

3) ทดลองเปลี่ยนค่าในบรรทัดที่ 3 ของโปรแกรมใหม่ ซึ่งเดิม C = 11H เป็น 00H แล้วรันโปรแกรมใหม่อีกครั้ง แล้วทำการบันทึกผลการทดลอง

ผลลัพธ์ในหน่วยความจำ 2200-220F เป็น_____

4) จงทำการเขียนโปรแกรมที่มีลักษณะดังข้อ 1 แต่ให้ค่าข้อมูลที่เก็บในรีจิสเตอร์ C นั้นเพิ่มค่าทุกครั้งเมื่อเมื่อเพิ่มค่าตำแหน่งหน่วยความจำ ซึ่งถ้าโปรแกรมถูกต้องจะได้ข้อมูลใน ตำแหน่ง 2000 – 200F เป็น 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

สามารถเขียนโปรแกรมได้ดังนี้คือ

| Address | Machine Code | Label | Operand | Comment |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

## สรุปผลการทดลอง

# ใบงานที่ 4

# การแสดงผลด้วย LED Metrix

## วัตถุประสงค์

1) เพื่อให้รู้จักการแสดงผลอย่างง่าย ด้วย LED Metrix

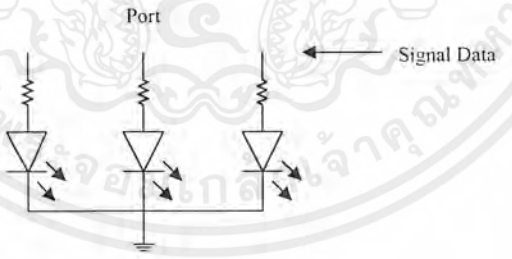
2) เพื่อให้รู้จักการสแกน

3) เพื่อให้เกิดความคิดในการออกแบบ

4) เพื่อให้เข้าใจการต่ออุปกรณ์ร่วมกับไมโครโปรเซสเซอร์ Z-80

## ทฤษฎี

แอลอีดีเป็นอุปกรณ์แสดงผล โดยที่สามารถที่จะนำมาใช้ในการแสดงผล บอกสถานะต่าง ๆ ตามที่ต้องการ ซึ่งอุปกรณ์ประเภทนี้จะเป็นอุปกรณ์ที่พบเห็นโดยทั่วไป มีหลักการทำงานโดยการเปล่งแสงออกมา การต่อแอลอีดีมีหลายแบบขึ้นอยู่กับความต้องการ แต่ในที่นี้จะกล่าวถึงการต่อแอลอีดีเป็นการภาคแสดงผล (Display) อย่างง่าย ๆ ก่อน คือ ภาคแสดงผลแบบแถวเดียว ลักษณะการต่อวงจรดังแสดงในรูปที่ ข.8

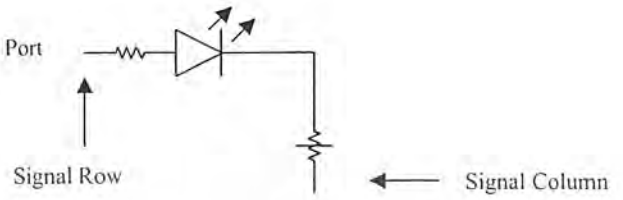



**รูปที่ ข.8** แอลอีดีแสดงผลแบบแถวเดียว

จากรูปที่ ข.8 มีหลักการทำงานโดยสามารถควบคุมสัญญาณเพียงด้านเดียว คือ ส่งแต่สัญญาณข้อมูล (Signal data) แล้วหน่วงเวลาไว้ระยะหนึ่ง แล้วจึงส่งสัญญาณตัดต่อไป ซึ่งจะทำให้สามารถมองเห็นเป็นลักษณะไฟวิ่งได้

นอกจากนี้สามารถที่ทำควบคุมไฟให้เห็นเป็นตัวอักษรหรือรูปภาพได้ การต่อแบบนี้จะพบเห็นกันมาก เนื่องจากปัจจุบันได้มีการนำไปใช้งานกันอย่างแพร่หลาย การต่อวงจรสามารถทำได้

โดยทำการเพิ่มลอจิกการควบคุมให้กับแอลอีดี ซึ่งแทนที่จะควบคุมการติดดับเพียงด้านเดียวก็เป็นสองด้าน โดยมีลักษณะการต่อวงจรดังแสดงรูปที่ ข.9




**รูปที่ ข.9** แอลอีดีแสดงผลแบบสองแถว

จากรูปที่ ข.9 เมื่อต่อสัญญาณสัญญาณตามจุดต่าง ๆ ได้เท่ากับแถวคูณคอลัมน์ ทำให้เกิดตัวอักษรวิ่งได้โดยส่งสัญญาณแถว และคอลัมน์ให้สอดคล้องกัน เรียกว่า วิธีการมัลติเพล็กซ์ (Multiplex) คือ ส่งข้อมูลตัวที่ 1 กำหนดให้คอลัมน์ที่ 1 ทำงาน แล้วหน่วงเวลาไว้ระยะหนึ่ง จากนั้นส่งข้อมูลตัวที่ 2 แล้วกำหนดให้คอลัมน์ตัวที่ 2 ทำงาน แล้วหน่วงเวลา ทำอย่างนี้ไปเรื่อยจนครบคอลัมน์สุดท้าย ซึ่งจะได้ตัวอักษรหรือรูปภาพ แต่ภาพที่เกิดจะเกิดในช่วงเวลาสั้น ๆ เพราะทำงานเร็วจำเป็นต้องกลับไปเริ่มส่งข้อมูลแบบเดิมระยะหนึ่ง เพื่อให้เห็นภาพชัดติดตา นั้นจะเป็นการทำให้ตัวอักษรเลื่อน โดยการเลื่อนตำแหน่งข้อมูลตัวแรกไป 1 ตำแหน่ง แล้วสแกนกลับไปวนข้อมูลระยะหนึ่ง แล้วเพิ่มตำแหน่งของข้อมูลที่เพิ่มแล้วขึ้นไปอีก 1 ทำจนถึงตำแหน่งของข้อมูลตัวอักษรสุดท้าย

ในการที่จะให้ทำงานได้นั้นจะต้องมีการกำหนดให้ 8255 พอร์ต A เป็นคอลัมน์ และพอร์ต B เป็นแถว โดยการสั่ง Initial ที่คอนโทรลพอร์ตของ 8255 (ทางซอฟต์แวร์) การที่จะให้เมตริกซ์แถวใดสว่างก็ส่งแถวให้มีค่าเป็น 0 และคอลัมน์มีค่าเป็น 1 ด้วย

## ลำดับขั้นการทดลอง

1) ทำการทดลองโดยเขียนโปรแกรม การสั่งให้แอลอีดีมุมซ้ายติด

2) เขียนโปรแกรมให้แอลอีดีสแกนทางหลัก

ตามนี้คือ

| Address | Machine Code | Operand | | Comment |
|---|---|---|---|---|
| 0000 | 3E 80 | LD | A,80H | ; Initial 8255 |
| 0002 | D3 83 | OUT | (83H),A | ; Initial 8255 |
| 0004 | 3E FF | LD | A,11111111B | ;.ให้ข้อมูลเป็น 11111111 B |
| 0006 | D3 81 | OUT | (81H),A | ; ส่งค่าข้อมูลออก พอร์ต A ของ 8255 |
| 0008 | 3E 01 | LD | A,00000001B | ; ให้ข้อมูลเป็น 00000001 B |
| 000A | D3 80 | OUT | (80H),A | ; ส่งค่าข้อมูลออก พอร์ต B ของ 8255 |
| 000C | CB17 | RL | A | ;หมุนข้อมูลไปทางซ้าย |
| 000E | C30A00 | JP | 0AH | ; กระโดดไปส่งค่าออกพอร์ต B |

แล้วทำการแอสเซมเบลอร์และรันดูผลการทดลอง

3) เขียนโปรแกรมให้แอลอีดีสแกนทางแถว

| Address | Machine Code | Operand | Comment |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

หมายเหตุ อุปกรณ์ทุกตัวต่อกับ 8255 จำลองอยู่ที่ตำแหน่ง 80H

## สรุปผลการทดลอง

## คำถามท้ายการทดลอง

1) อธิบายการทำงานของโปรแกรมไฟอักษรวิ่ง ตามความเข้าใจ

# ใบงานที่ 5

# การแสดงผลด้วยตัวแสดงผลเจ็ดส่วน

## วัตถุประสงค์

1) เพื่อให้รู้จักการแสดงผลอย่างง่าย ด้วย 7-segment

2) เพื่อให้รู้จักการสแกน

3) เพื่อให้เกิดความคิดในการออกแบบ

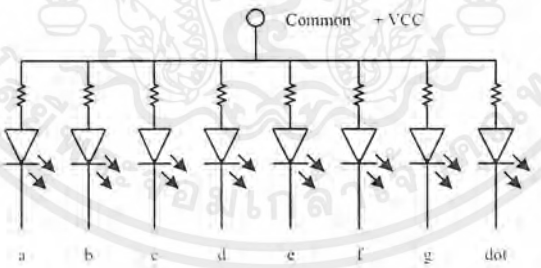
4) เพื่อให้เข้าใจการต่ออุปกรณ์ร่วมกับไมโครโปรเซสเซอร์ Z-80

## ทฤษฎี

**ตัวแสดงผลแบบเจ็ด**

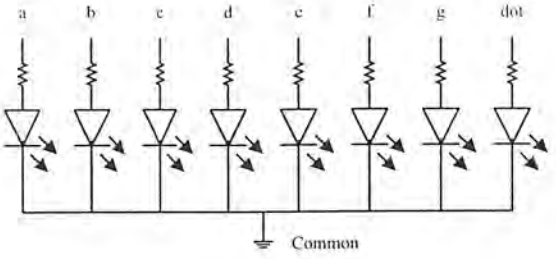
ตัวแสดงผลแบบเจ็ดส่วนเป็นอุปกรณ์แสดงผลอย่างหนึ่ง ซึ่งโครงสร้างของตัวแสดงผลแบบเจ็ดส่วนประกอบด้วยแอลอีดีจำนวน 7 ตัว หรือ 8 ตัว หรือมากกว่าในกรณีชนิดพิเศษซึ่ง ตัวแสดงผลแบบเจ็ดส่วนถ้าแบ่งตามชนิดของสารต่อแอลอีดีภายใน แบ่งได้เป็น 2 ชนิด คือ

**1) การต่อร่วมแอโนด** (Common Anode) คือ นำขาอาโนดของ แอลอีดีมาต่อร่วมกัน ลักษณะการต่อแอลอีดีภายในของชนิดแอโนด ดังแสดงในรูปที่ ข.10

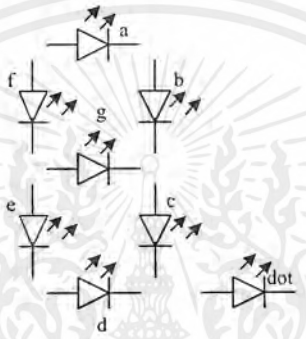



**รูปที่ ข.10** ตัวแสดงผลแบบเจ็ดส่วนชนิดแอลอีดีต่อร่วมแอโนด

**2) การต่อร่วมแคโทด** (Common Cathode) คือ นำขาแคโทดของแอลอีดีต่อร่วมกัน ลักษณะการต่อแอลอีดีภายในของชนิดการต่อร่วมแคโทด ดังแสดงในรูปที่ ข.11



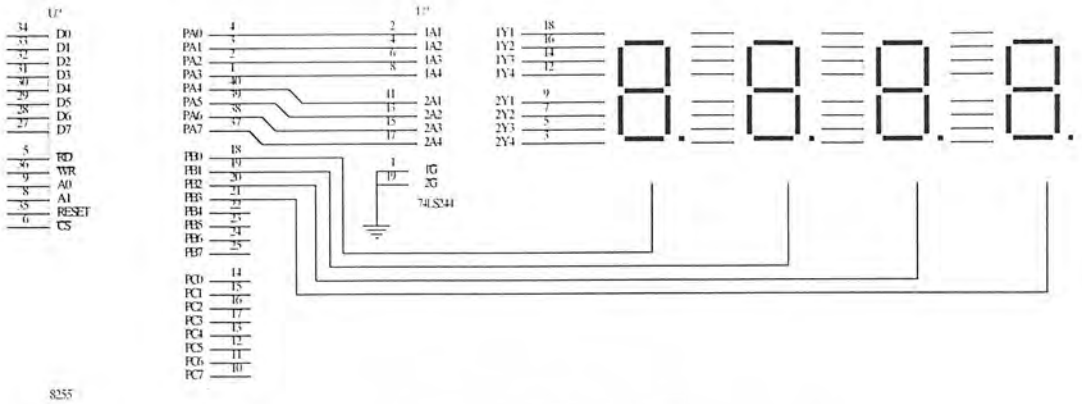

**รูปที่ ข.11** ตัวแสดงผลแบบเจ็ดส่วนชนิดแอลอีดีต่อร่วมแคโถด




**รูปที่ ข.12** ลักษณะการวางตำแหน่งของแอลอีดีบนตัวแสดงผลเจ็ดส่วน

ถ้ากำหนดให้ตัวแสดงผลเจ็ดส่วนนี้เป็นแบบถาโถดร่วมการทำให้เซกเมนต์ใดๆ ติดได้โดยให้ระดับบิตเป็น 1 ดังนั้นการแสดงตัวเลข สามารถเขียนเป็นเลขฐาน 16 ได้ดังตารางที่ ข.5 วิธีการคือเมื่อ ต้องการให้แสดงผลเป็นเลขใด ก็ส่งรูปแบบนั้น ออกทางพอร์ตเอาต์พุต

**3) การนำข้อมูลออกที่ตัวแสดงผลเจ็ดส่วนหลายตัว** การส่งข้อมูลออกที่ตัวแสดงผลเจ็ดส่วนหลายๆ ตัวนั้น โดยทั่วไปนิยมใช้วิธีการมัลติเพล็ก แทนที่จะใช้พอร์ตเท่ากับจำนวนของตัวแสดงผล คือจะใช้พอร์ตหนึ่งต่อเข้ากับเซกเมนต์ของทุกตัวแสดงผล และอีกพอร์ตหนึ่งจะต่อเข้ากับขาร่วม (Common) ของตัวแสดงผลนั้นๆ และการจะทำให้ตัวแสดงผลตัวใดติด ก็ทำการควบคุมที่ขาร่วมของตัวแสดงผลแต่ละตัวนั้นซึ่งวิธีการต่อวงจรแสดงดังรูปที่ 4




รูปที่ ข.13 วงจรการต่อตัวแสดงผลเจ็ดส่วนแบบ 4 หลัก

ตารางที่ ข.6 รูปแบบของรหัสการแสดงตัวเลขที่ ตัวแสดงผลเจ็ดส่วน

| เลข | . | G | f | e | d | c | b | a | รหัส |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 3F |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 06 |
| 2 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 5B |
| 3 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 4F |
| 4 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 66 |
| 5 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 6D |
| 6 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 7D |
| 7 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 07 |
| 8 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 7F |
| 9 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 6F |
| A | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 77 |
| B | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 7C |
| C | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 39 |
| D | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 5E |
| E | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 79 |
| F | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 71 |

## ลำดับขั้นการทดลอง

1) ทำการทดลองโดยการเขียนโปรแกรมให้ตัวแสดงผลเจ็ดส่วนติดเป็นเลข 0 แล้วสแกนพอร์ตไปด้วยได้ตามโปรแกรมด้านล่างคือ

| Address | Machine Code | Operand | | Comment |
|---|---|---|---|---|
| 0000 | 3E 80 | LD | A,80H | ; Initial 8255 |
| 0002 | D3 83 | OUT | (83H),A | ; Initial 8255 |
| 0004 | 37 | SCF | | ;.ให้แคร์รี่แฟลกเป็น 1 |
| 0005 | 3E 3F | LD | A,3F H | ; ให้แสดงเลข 0 |
| 0007 | D3 80 | OUT | (80H),A | ; ส่งค่าข้อมูลออก พอร์ต A ของ 8255 |
| 0009 | 3E EE | LD | A,11101110B | ;กำหนดตำแหน่งหลักที่จะติดเป็น 0 |
| 000B | D3 81 | OUT | (81H),A | ; ส่งค่าข้อมูลออก พอร์ต B ของ 8255 |
| 000D | CB 17 | RL | A | ;หมุนค่าตำแหน่งที่สแกนไปทางขวา |
| 000F | C3 0B 00 | JP | 0B H | :วนลูป ส่งค่าออกพอร์ตอีกครั้ง |

ดูผลการทดลองจากหน้าต่าง Seven Segment

2) จงทำการเขียนโปรแกรมแสดงค่า 0 – 9 ที่ตัวแสดงผลหลักขวามือสุด แล้ววนแสดงค่าจนกว่าจะหยุดโปรแกรมด้วยปุ่ม Stop

| Address | Machine Code | Operand | Comment |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

| Address | Machine Code | Operand | Comment |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

หมายเหตุ อุปกรณ์ทุกตัวต่อกับ 8255 จำลองอยู่ที่ตำแหน่ง 80H

## สรุปผลการทดลอง

คำถามท้ายการทดลอง

1) ถ้าใช้ตัวแสดงผลเป็นแบบคอมมอนแอโนด แล้วรูปแบบรหัสการแสดงผลของตัวแสดงผลเจ็ดส่วนจะเป็นอย่างไร

| เลข | . | g | f | e | d | c | b | a | รหัส |
|---|---|---|---|---|---|---|---|---|---|
| 0 | | | | | | | | | |
| 1 | | | | | | | | | |
| 2 | | | | | | | | | |
| 3 | | | | | | | | | |
| 4 | | | | | | | | | |
| 5 | | | | | | | | | |
| 6 | | | | | | | | | |
| 7 | | | | | | | | | |
| 8 | | | | | | | | | |
| 9 | | | | | | | | | |
| A | | | | | | | | | |
| B | | | | | | | | | |
| C | | | | | | | | | |
| D | | | | | | | | | |
| E | | | | | | | | | |
| F | | | | | | | | | |

# ใบงานที่ 6
# การควบคุมมอเตอร์ไฟฟ้ากระแสตรง

## วัตถุประสงค์

1) เพื่อให้รู้จักการควบคุมให้มอเตอร์ไฟฟ้ากระแสตรงด้วยไมโครโปรเซสเซอร์ Z-80

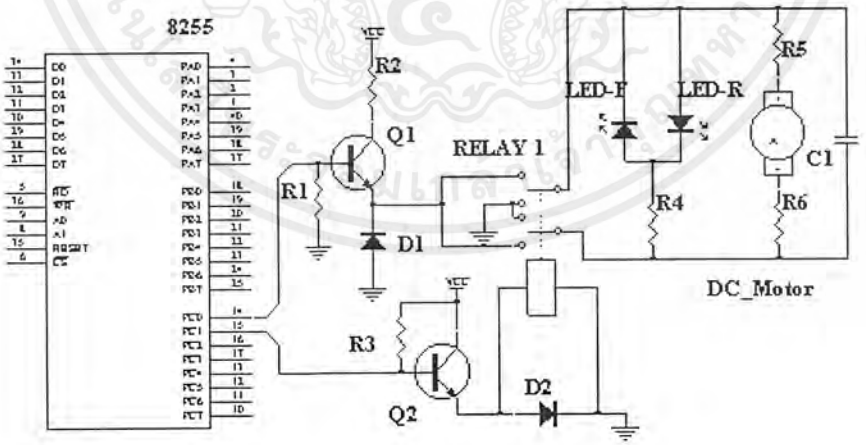
2) เพื่อให้รู้จักการควบคุมทิศทางของมอเตอร์ไฟฟ้ากระแสตรง

3) เพื่อให้เกิดความคิดในการออกแบบ

4) เพื่อให้เข้าใจการต่ออุปกรณ์ร่วมกับไมโครโปรเซสเซอร์ Z-80

## ทฤษฎี

มอเตอร์ไฟฟ้ากระแสตรงมีการทำงานที่มีสถานะการทำงานอยู่ 2 สถานะใหญ่ ก็คือ หมุนและไม่หมุน ซึ่งจะหมุนก็ต่อเมื่อจ่ายไฟเข้าที่มอเตอร์ และหยุดหมุนเมื่อ ไม่มีไฟเลี้ยง แต่จะแตกต่างกับอุปกรณ์ประเภทแสดงผลเช่น แอลอีดี หรือ ตัวแสดงผลแบบเจ็ดส่วนคือ ในสถานะหมุนนั้นเรายังสามารถกำหนดได้อีกว่าจะให้มอเตอร์นั้นหมุนไปในทิศทางใด คือสถานะหมุนก็ยังแบ่งได้ออกเป็น สถานะย่อยได้อีก 2 สถานะคือ หมุนตามเข็มนาฬิกา และหมุนทวนเข็มนาฬิกา




รูปที่ ข.14 วงจรควบคุมมอเตอร์ไฟฟ้ากระแสตรง

และในการควบคุมว่าจะให้หมุนทวนเข็มนาฬิกาหรือหมุนตามเข็มนาฬิกานั้น กำหนดได้จากว่าจ่ายไฟเลี้ยงเข้าที่มอเตอร์เป็นแบบใดซึ่งถ้าจ่าย ไฟบวกเข้าที่ขั้วบวก และลบเข้าที่ขั้วลบ มอเตอร์ก็จะหมุนตามเข็มนาฬิกา แต่ถ้าจ่ายไฟกลับขั้วกันแล้วก็จะทำให้มอเตอร์หมุน ทวนเข็มนาฬิกา ซึ่งเราสามารถออกแบบ วงจรควบคุมการจ่ายไฟ และ การควบคุมทิศทางได้ดังในวงจรตามรูปที่ ข.14

จากวงจรในรูปที่ 1 ต่อวงจรควบคุมมอเตอร์ไฟฟ้ากระแสตรงอยู่ที่พอร์ต C0 และ C1 โดยการทำงานของวงจรสามารถอธิบายได้ดังนี้คือ พอร์ต C0 จะเป็นพอร์ตที่ทำหน้าที่ควบคุมการจ่ายไฟเลี้ยงเข้าที่มอเตอร์ โดยผ่านทาง รีเลย์ เมื่อจ่าย ลอจิก 1ให้พอร์ต C0 ก็จะทำให้มีไฟเลี้ยงจ่ายไปยังมอเตอร์ทำให้ มอเตอร์หมุน และถ้าพอร์ต C0 เป็นลอจิก 0 ก็ไม่มีไฟเลี้ยงจ่ายให้มอเตอร์ ก็จะทำให้มอเตอร์หยุดหมุน พอร์ต C1 จะเป็นพอร์ตที่ทำหน้าที่ควบคุมขั้วไฟฟ้าที่จะจ่ายให้กับมอเตอร์โดยควบคุมผ่านรีเลย์

## ลำดับขั้นการทดลอง

1) ทำการทดลองโดยเขียนโปรแกรม การสั่งให้มอเตอร์หมุนตามเข็มนาฬิกาโดยโปรแกรมดังนี้คือ

| Address | Machine Code | Label | Operand | | Comment |
|---|---|---|---|---|---|
| 0000 | 3E 80 | | LD | A,80H | ;Initial 8255 |
| 0002 | D3 83 | | OUT | (83H),A | |
| 0004 | 06 3F | | LD | B,3FH | ;กำหนดช่วงเวลาที่ให้หมุน |
| 0006 | 3E FF | | LD | A,11111111B | ;ให้หมุนตามเข็มนาฬิกา |
| 0008 | D3 82 | | OUT | (82H),A | ;ส่งข้อมูลไปควบคุม |
| 000A | 10 FE | LOOP: | DJNZ | LOOP | ;ลดค่าเวลาลงจนครบ |
| 000C | 3E 00 | | LD | A,00000000B | ;ให้มอเตอร์หยุดหมุน |
| 000E | D3 82 | | OUT | (82H),A | |
| 0010 | FF | | RST | 38H | ;จบการทำงาน |

แล้วทำการแอสเซมเบลอละรันดูผลการทดลอง

2) จงเขียนโปรแกรมให้มอเตอร์หมุนตามเข็มนาฬิกา แล้วให้หมุนทวนเข็มนาฬิกา จึงจบการทำงาน

| Address | Machine Code | Label | Operand | Comment |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

หมายเหตุ อุปกรณ์ทุกตัวต่อกับ 8255 จำลองอยู่ที่ตำแหน่ง 80H

## สรุปผลการทดลอง

## คำถามท้ายการทดลอง

1) อธิบายการทำงานของโปรแกรมควบคุมมอเตอร์ไฟฟ้ากระแสตรงตามข้อ 2 มาพอเข้าใจ

# ใบงานที่ 7
# การควบคุมสเต็ปปิ้งมอเตอร์

## วัตถุประสงค์

1) เพื่อให้รู้เกี่ยวกับ เกี่ยวกับ สเต็ปปิ้งมอเตอร์แบบต่าง ๆ

2) เพื่อให้รู้เข้าใจการสั่งงาน สเต็ปปิ้งมอเตอร์

3) เพื่อให้เกิดแนวคิดที่จะนำ สเต็ปปิ้งมอเตอร์ไปใช้งาน

## ทฤษฎี

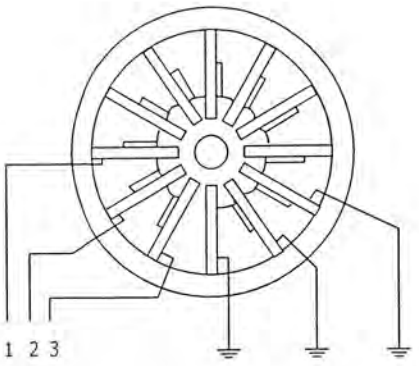
### สเต็ปปิ้งมอเตอร์

**1) ประเภทของสเต็ปปิ้งมอเตอร์** เราสามารถแบ่งสเต็ปปิ้งมอเตอร์ตามพื้นฐานได้ 3 ชนิดคือ

**1.1) ชนิดวาริเอเบิลรีลักแตนซ์ (Variable reluctance หรือ VR)** สเต็ปปิ้งมอเตอร์ชนิดนี้มีข้อเสียคือ เมื่อมีสเตปในการหมุนสูง จึงทำให้ความถูกต้องของตำแหน่ง และทำงานได้ดี สามารถทดสอบเพื่อให้ทราบว่าเป็นสเต็ปเปอร์มอเตอร์ชนิดนี้ได้ง่ายมาก โดยใช้มือหมุนที่เพลาของมอเตอร์ ซึ่งจะไม่เกิดปรากฏการณ์ทางแม่เหล็ก (Magnetism) จะทำให้หมุนได้โดยไม่ติดขัด แตกต่างจากชนิดอื่น คือเมื่อทำการหมุนจะรู้สึกขัด ๆ เหมือนเป็นฟันเฟือง

VR สเต็ปปิ้งมอเตอร์ที่มีสแต็คเดียวจะมีโรเตอร์เดียวเมื่อเทียบกับ VR สเต็ปปิ้งมอเตอร์แบบมีหลายสแต็ค หมายถึงมีหลายโรเตอร์ โรเตอร์และสเตเตอร์ทำจากสารแม่เหล็ก ขั้วของสเตเตอร์ที่อยู่ตรงกันข้ามจะพันด้วยขดลวดลักษณะที่ต่างกัน เพื่อให้มีความสมดุลย์ระหว่างเส้นแรงแม่เหล็กเข้า และออกจากโรเตอร์

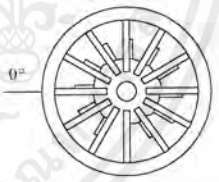
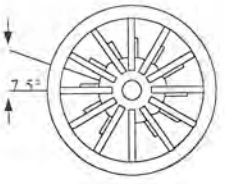
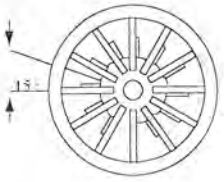
ตัวอย่างโครงสร้างของสเต็ปปิ้งมอเตอร์แบบค่ารีลักแตนซ์แปรค่าสแต็คเดียว หรือที่เรียกสั้นๆ ว่า VR สเต็ปปิ้งมอเตอร์ที่มีสแต็คเดียวแสดงได้ในรูปที่ ข.15



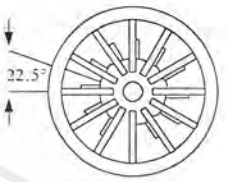

**รูปที่ ข.15** VR สเต็ปปิ้งมอเตอร์แบบมีสแต็คเดียว

ลำดับการทำงานที่สมบูรณ์แสดงในตารางที่ 1 เมื่อตำแหน่งเริ่มต้นของซี่ฟันของโรเตอร์ จะเป็นสีดำเพื่อให้เราทำความเข้าใจได้ชัดเจนถึงการหมุนของโรเตอร์ในทิศทาง CW

**ตารางที่ ข.6** ลำดับของการสวิตช์ 3 สเต็ปของ VR สเต็ปปิ้งมอเตอร์แบบสแต็คเดียว และตำแหน่งของโรเตอร์

| การเรียงลำดับเฟส | ตำแหน่งของโรเตอร์และเส้นแรงแม่เหล็ก |
|---|---|
| ตำแหน่งโรเตอร์เริ่มต้น :<br>1) เฟส $\phi_1$ ได้รับพลังงาน<br>2) ซี่ฟันของโรเตอร์จะอยู่ในแนวซี่ฟันที่ 1, 4, 7, 10 ของสเตเตอร์ |  |
| สเต็ปที่ 1 :เฟส $\phi_3$ ได้รับพลังงาน<br>ซี่ฟันของโรเตอร์จะอยู่ในแนวซี่ฟันที่ 2, 5, 8, 11 ของสเตเตอร์<br>1) โรเตอร์จะเคลื่อนไปในทิศทาง CW เป็นมุม $7.5^\circ$ (1/3 ช่วงระหว่างซี่ฟันของโรเตอร์) |  |
| สเต็ปที่ 2 :เฟส $\phi_2$ ได้รับพลังงาน<br>ฟันของโรเตอร์จะอยู่ในแนวซี่ฟันที่ 3, 6, 9, 12ของสเตเตอร์<br>1) โรเตอร์จะเคลื่อนไปในทิศทาง CW รวมเป็นมุม $15^\circ$ |  |

**ตารางที่ ข.6(ต่อ)** ลำดับของการสวิตซ์ 3 สเต็ปของ VR สเต็ปปิ้งมอเตอร์แบบสแต็คเดียว และตำแหน่งของโรเตอร์

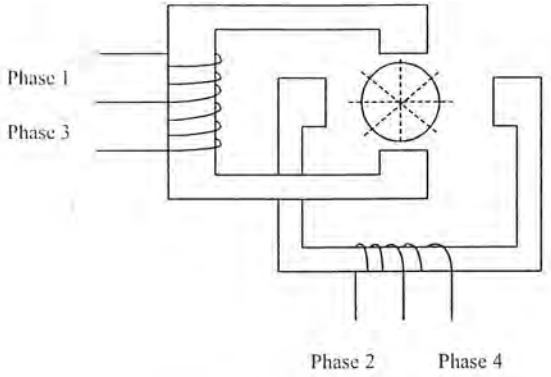
| การเรียงลำดับเฟส | ตำแหน่งของโรเตอร์และเส้นแรงแม่เหล็ก |
|---|---|
| สเต็ปที่ 3 :เฟส $\phi_1$ ได้รับพลังงาน<br>ซี่ฟันของโรเตอร์จะอยู่ในแนวซี่ฟันที่ 1, 4, 7, 10 ของสเตเตอร์<br>1) โรเตอร์จะเคลื่อนไปในทิศทาง CW เป็นมุม $7.5^\circ$<br>(เคลื่อนไปได้ 1 ช่วงระหว่างซี่ฟันของโรเตอร์) |  |

**1.2) ชนิดเพอร์มาเนนต์แม็กเน็ต (Permanent magnet หรือ PM)** สเต็ปปิ้งมอเตอร์ชนิดนี้มีข้อดีคือ มีความถูกต้องของตำแหน่งเมื่อเปรียบเทียบกันกับสเต็ปปิ้งมอเตอร์ชนิดอื่น

**1.3) ชนิดไฮบริดจ์ (Hybrid)** เป็นชนิดที่นิยมใช้มากที่สุดในเครื่องคอมพิวเตอร์โดยจะใช้ในส่วนของการขับเคลื่อนหัวอ่านดิสก์ไดร์ว สเต็ปปิ้งมอเตอร์ชนิดนี้มีข้อดีคือ มีโครงสร้างภายในคือเป็นสเตเตอร์ซึ่งเป็นชนิดวาริเอเบิลรีลักแตนซ์ ส่วนโรเตอร์ที่เป็นชนิดเพอร์มาเนนต์แม็กเนตนำมาประกอบเข้าด้วยกัน ทำให้เป็นมอเตอร์ชนิดนี้มีแรงยึดหน่วงสูง มีแรงบิดดึง และผลักดี และยังคงทำงานได้ดีแม้ว่าจะมีจำนวนสเต็ปต่อรอบในการหมุนสูงก็ตาม

ไฮบริดจ์สเต็ปปิ้งมอเตอร์ (HSM) มีคุณลักษณะผสมของ PM และ VR สเต็ปปิ้งมอเตอร์ โครงสร้างของ HSM ประกอบด้วย 2 ตอนกับแกนแม่เหล็กอยู่ระหว่าง 2 ตอนแต่ละตอนประกอบด้วยซี่ฟันของโรเตอร์ และโพลของสเตเตอร์ที่มีซี่ฟันเช่นกัน และพันด้วยขดลวดรายละเอียดโครงสร้างของสเตเตอร์ และโรเตอร์ของแต่ละตอน

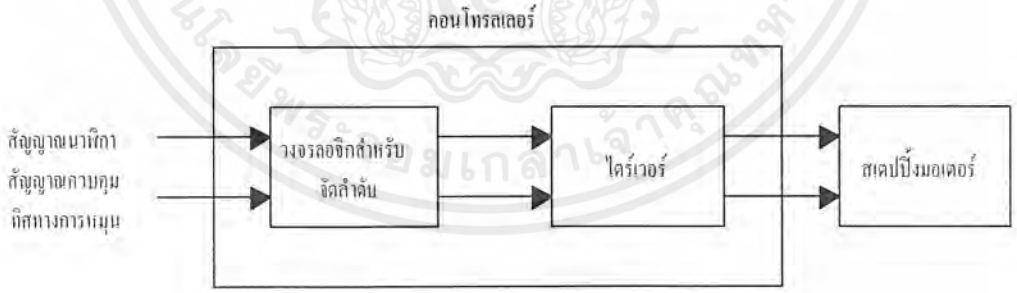
**1.4) ชนิดแรเอิร์ธเพอร์มาเนนต์แมกเน็ต (Rare earth permanent magnet)** หรือที่เรียกกันว่าชนิดดิสก์แม็กเน็ต สเต็ปปิ้งมอเตอร์ (Disk magnet stepper) การทำงานจะเป็นแบบเดิมแต่โครงสร้างเป็นแบบใหม่จะทำให้เกิดความเฉื่อยต่ำมาก แต่มีอัตราเร่งสูง มอเตอร์ชนิดนี้จึงจัดเป็นมอเตอร์ที่มีประสิทธิภาพสูงทั้งในด้านแรงบิดดึง, กำลังทางกลที่ได้ของมอเตอร์, ความถูกต้องของตำแหน่งสูงมาก และความเร็วในการเริ่มหมุน และหยุดสูง อีกทั้งมีการสูญเสียของกำลังงานต่ำ




**รูปที่ ข.16** สเต็ปปิ้งมอเตอร์แบบ 4 เฟส (Phase) แบบยูนิโพลาร์เพอร์มาเนนแม็กเนต

**2) โครงสร้าง และการทำงานของสเต็ปปิ้งมอเตอร์**

สเปปิ้งมอเตอร์เป็นอุปกรณ์จำพวกเชิงกลไฟฟ้า ที่มีอินพุตเป็นกลุ่มไบนารี่โวลต์เตจ และเอาต์พุตเป็นลักษณะการเคลื่อนที่แบบเชิงมุม หรือหมุนไปเป็นสเต็ป (แต่ละสเต็ปอยู่ในช่วง 0.1 ถึง 30 องศา ขึ้นอยู่กับโครงสร้างของสเต็ปปิ้งมอเตอร์) ตามสัญญาณพัลซ์ที่ป้อนให้กับขดสเตเตอร์ซึ่งเกิดแรงผลักต่อโรเตอร์หมุนไป แต่ลักษณะของสเต็ปปิ้งมอเตอร์จะมีขดของสเตเตอร์อยู่หลายขดซึ่งเรียกว่า "เฟส" ฉะนั้นเมื่อป้อนสัญญาณที่เป็นพัลซ์ในลักษณะเป็นลำดับของเลขฐานสองผ่านวงจรขับ (Driver) จะทำให้โรเตอร์หมุนได้อย่างต่อเนื่องดังแผนผังในรูปที่ ข.17




**รูปที่ ข.17** แผนผังการควบคุมสเต็ปปิ้งมอเตอร์

จากแผนผังสเต็ปปิ้งมอเตอร์ในรูปที่ ข.17 จะเห็นว่าในการควบคุมสเต็ปปิ้งมอเตอร์ให้สามารถเคลื่อนที่ได้ตามที่ต้องการนั้น จะต้องทำการส่งสัญญาณทางด้านอินพุตซึ่งเป็นสัญญาณนาฬิกา (Clock Pulse) และส่งสัญญาณอินพุตซึ่งเป็นสัญญาณ ที่ใช้สำหรับควบคุมทิศทางการหมุนของสเตปปิ้งมอเตอร์เสียก่อนจึงจะสามารถทำการควบคุมสเต็ปปิ้งมอเตอร์ได้ สัญญาณที่ได้นั้นจะ

ต้องผ่านวงจรลอจิกเพื่อทำการจัดลำดับเสียก่อน แล้วส่งเข้าไปขยายสัญญาณที่วงจรขับมอเตอร์เพื่อให้มีแรงดันเพียงพอในการขับสเตปปิ้งมอเตอร์ได้

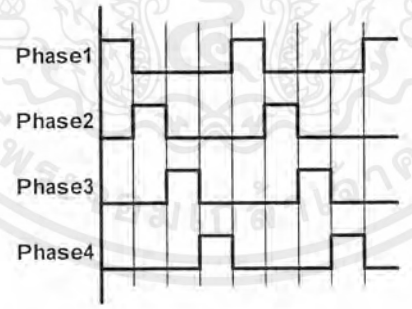
**3) การกระตุ้นและการควบคุมการหมุนของสเต็ปปิ้งมอเตอร์**

การทำให้สเต็ปปิ้งมอเตอร์เคลื่อนไปทีละสเต็ป ทำได้โดยการจ่ายกำลังไฟฟ้าไปยังขดลวดแต่ละขดบนสเตเตอร์ ซึ่งจะต้องป้อนแบบซีเควนเชียลในรูปแบบที่ถูกต้อง การป้อนพัลซ์กระตุ้นสเต็ปปิ้งมอเตอร์สามารถทำได้ 3 รูปแบบคือ

**3.1) แบบเวฟ (Wave)** เป็นการป้อนกระแสให้กับขดลวดแต่ละขดของสเต็ปปิ้งมอเตอร์ทีละขดเรียงลำดับกันได้ ลักษณะการขับแบบนี้จะทำให้แรงบิดน้อย

ตารางที่ ข.7 การจ่ายกระแสแบบเวฟให้กับขดลวดแต่ละขดของสเต็ปปิ้งมอเตอร์

| Step | Phase | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| 1 | 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 1 |

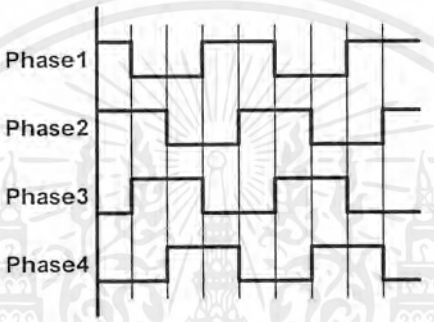



รูปที่ ข.18 การจ่ายกระแสแบบเวฟให้กับขดลวดแต่ละขดของสเต็ปปิ้งมอเตอร์

**3.2) แบบ 2 เฟส (Two Phase)** มีลักษณะคล้ายกับแบบเวฟ แต่การกระตุ้นโดย จ่ายกำลังไฟฟ้าไปที่ขดลวด 2 ขด ที่อยู่ใกล้กันในเวลาเดียวกันเรียงถัดกันไป เช่นเดียวกับแบบเวฟขึ้นอยู่กับทิศทางของการหมุน การเพิ่มจำนวนขดลวดของขดลวดที่ถูกกระตุ้นจะทำให้เพิ่มแรงบิดได้มากกว่าแบบเวฟ โรเตอร์จะเคลื่อนที่ด้วยแรงดึงเต็มที่ด้วยแรงดึงจากทั้งสองขดลวดที่ถูกกระตุ้นพร้อมกัน ข้อเสียของการกระตุ้นแบบนี้คือ ต้องจ่ายกำลังไฟฟ้ามากขึ้น การทำงานต่าง ๆ จะแสดงในรูปที่ ข.19

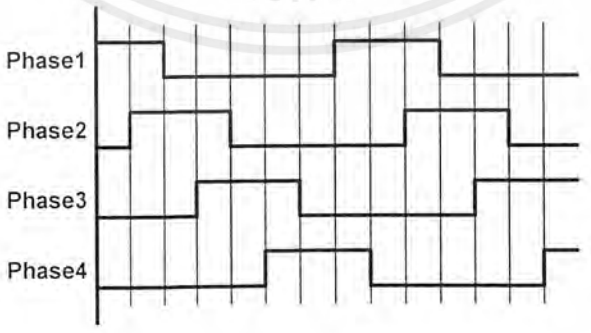
ตารางที่ ข.8 การจ่ายกระแสแบบ 2 เฟสให้กับขดลวดแต่ละขดของสเต็ปปิ้งมอเตอร์

| Step | Phase | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| 1 | 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 1 |




รูปที่ ข.19 การจ่ายกระแสแบบ 2 เฟสให้กับขดลวดแต่ละขดของสเต็ปปิ้งมอเตอร์

**3.3) แบบครึ่งสเต็ป (Half Step)** เป็นแบบที่ได้จากการผสมระหว่างการกระตุ้นแบบเวฟและแบบ 2 เฟส ดังแสดงในรูปที่ ข.20 เพื่อเพิ่มจำนวนสเต็ปต่อรอบอีกหนึ่งเท่าตัว แรงบิดที่ได้จากการกระตุ้นแบบนี้จะเพิ่มมากขึ้นอีกเพราะช่วงสเต็ปมีระยะสั้นลง และแต่ละสเต็ปเกิดจากแรงดึงของขดลวด 2 ขดที่กระตุ้นพร้อมกัน ความถูกต้องของตำแหน่งจึงมีเพิ่มขึ้น ที่สำคัญการกระตุ้นแบบนี้จะต้องทำการหมุน 2 สเต็ปจึงเท่ากับ 1 สเต็ปของ 2 แบบแรก ส่วนแหล่งจ่ายกำลังต้องใช้เหมือนกับแบบ 2 เฟส




รูปที่ ข.20 การจ่ายกระแสแบบครึ่งสเต็ป ให้กับขดลวดแต่ละขดของสเต็ปปิ้งมอเตอร์

ตารางที่ ข.9 การจ่ายกระแสแบบครึ่งสเต็ป ให้กับขดลวดแต่ละขดของสเต็ปปิ้งมอเตอร์

| STEP | PHASE | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| 1 | 1 | 1 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 |
| 4 | 0 | 0 | 1 | 0 |
| 5 | 0 | 0 | 1 | 1 |
| 6 | 0 | 0 | 0 | 1 |
| 7 | 1 | 0 | 0 | 1 |
| 8 | 1 | 0 | 0 | 0 |

## ลำดับขั้นการทดลอง

1) ทำการทดลองโดยเขียนโปรแกรม การสั่งให้สเต็ปปิ้งมอเตอร์หมุนตามเข็มนาฬิกาโดยจ่ายกระแสแบบเวฟโปรแกรมดังนี้คือ

| Address | Machine Code | Label | Operand | | Comment |
|---|---|---|---|---|---|
| 0000 | 3E 80 | | LD | A,80H | ;Initial 8255 |
| 0002 | D3 83 | | OUT | (83H),A | |
| 0004 | 3E 88 | | LD | A,10001000B | ;ให้หมุนแบบเวฟ |
| 0006 | D3 80 | | OUT | (80H),A | ;ส่งข้อมูลออกพอร์ต |
| 0008 | CB 17 | | RL | A | |
| 000A | C3 06 00 | | JP | 06 H | |

แล้วทำการแอสเซมเบลอละรันดูผลการทดลอง

2) จงเขียนโปรแกรมให้สเต็ปปิ้งมอเตอร์หมุนตามเข็มนาฬิกา โดยจ่ายกระแสแบบ 2 เฟส

| Address | Machine Code | Label | Operand | Comment |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

หมายเหตุ อุปกรณ์ทุกตัวต่อกับ 8255 จำลองอยู่ที่ตำแหน่ง 80H

## สรุปผลการทดลอง

## คำถามท้ายการทดลอง

1) อธิบายการทำงานของโปรแกรมควบคุมสเต็ปปิ้งมอเตอร์ไฟฟ้ากระแสตรงตามข้อ 2 มาพอเข้าใจ

# เฉลยใบงานที่ 1
# คำสั่งการโอนย้ายข้อมูล

## วัตถุประสงค์

1) เพื่อให้เข้าใจการใช้คำสั่งเกี่ยวกับการโอนย้ายข้อมูล

2) รู้จักการใส่ค่าเริ่มต้นของรีจิสเตอร์ต่าง ๆ

3) รู้จักการป้อนข้อมูล และคำสั่งให้โปรแกรมจำลองการทำงานไมโครโปรเซสเซอร์ Z-80

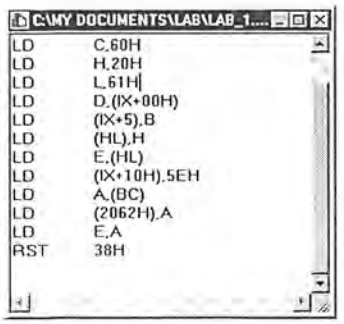
## ลำดับขั้นการทดลอง

1) เขียนโปรแกรมภาษา Assembly เพื่อทำการกำหนดข้อมูลในรีจิสเตอร์ต่าง ๆ ดังต่อไปนี้ A=00H, B=01H, C=02H, D=03H, E=04H, โดยใช้คำสั่ง LD แบบ 8 บิต

1.1) เขียนโปรแกรมภาษา Assembly และ Machine Code ลงในช่องว่าง

| Memory address | Machine Code | | Assembly |
|---|---|---|---|
| 0000 | 3E | 00 | LD A,00H |
| 0002 | 06 | 01 | LD B,01H |
| 0004 | 0E | 02 | LD C,02H |
| 0006 | 16 | 03 | LD D,03H |
| 0008 | 1E | 04 | LD E,04H |
| 000A | FF | | RST 38H |

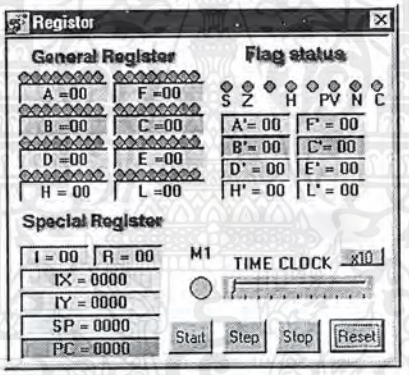
1.2) หลังจากที่ป้อนโปรแกรมแอสเซมบลี้ดังข้อที่ 1.1 เข้าโปรแกรมจำลองการทำงาน-ไมโครเซสเซอร์ Z-80 ในหน้าต่างอิดิเตอร์แล้ว สามารถแสดงได้ดังรูปที่ ข.21



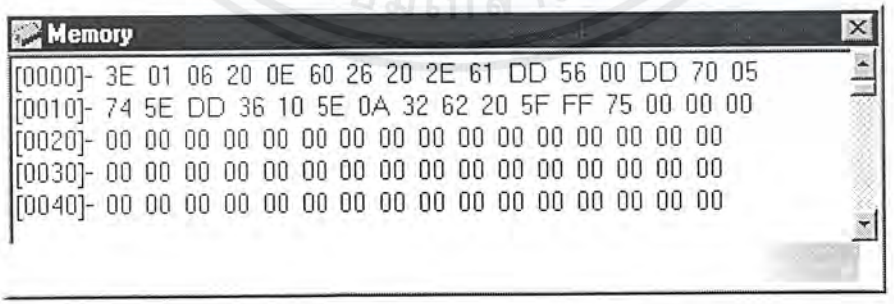

**รูปที่ ข.21** โปรแกรมที่เขียนในหน้าต่างอิดิตเตอร์

1.3) ทำการแอสเซมเบลอโปรแกรมโดยการคลิกที่ เมนู หรือ ทูลบาร์จะปรากฏหน้าจอแสดงได้ดังในรูปที่ ข.22


ก) ข้อมูลในหน้าต่าง List หนังจากทำการแอสเซมเบลอ แล้ว




ข) ข้อมูลในหน้าต่าง Registor หลังจากทำการแอสเซมเบลอ แล้ว

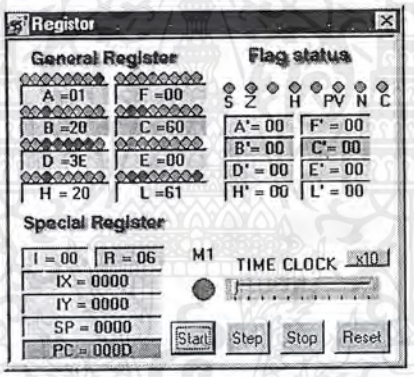



ค) ข้อมูลในหน้าต่าง Memory หนังจากทำการแอสเซมเบลอ แล้ว

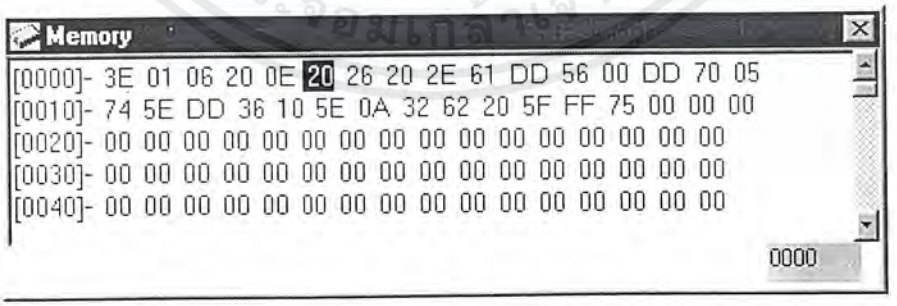
**รูปที่ ข.22** ข้อมูลในหน้าต่างโปรแกรมหลังจากได้ทำการแอสเซมเบลอแล้ว

1.4) ทำการรันโปรแกรมแบบสเต็ปแล้วตรวจสอบรีจิสเตอร์ต่าง ๆ ว่าตรงกับความเป็นจริงหรือไม่โดยตรวจตามลำดับจะได้ข้อมูลแสดงได้ดังรูปที่ข.23


ก) ข้อมูลในหน้าต่าง List หลังจากทำการรันไปจนถึงบรรทัดที่ 7 แล้ว

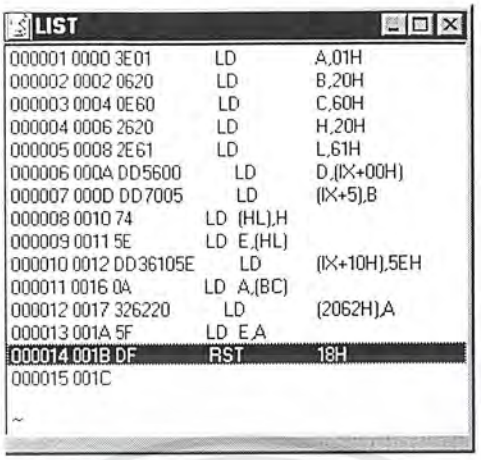



ข) ข้อมูลในหน้าต่าง Registor หลังจากทำการรันไปจนถึงบรรทัดที่ 7 แล้ว

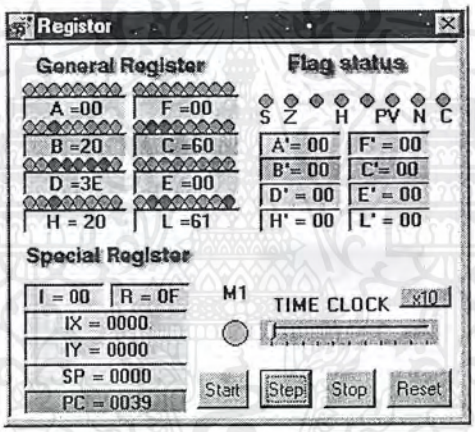



ค) ข้อมูลในหน้าต่าง Memory หลังจากทำการรันไปจนถึงบรรทัดที่ 7 แล้วแถบสีแสดงถึงข้อมูลที่เปลี่ยนแปลงไปหลังจากทำคำสั่ง LD (IX+5H),B

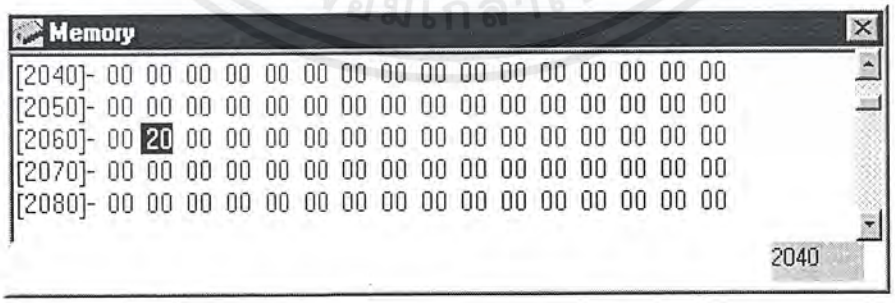
**รูปที่ ข.23** ข้อมูลในหน้าต่างโปรแกรมขณะทำการรันโปรแกรมแบบสเต็ป




ง) ข้อมูลในหน้าต่าง List หลังจากทำการรันไปจนถึงบรรทัดที่ 14 แล้ว




ฉ) ข้อมูลในหน้าต่าง Registor หลังจากทำการรันไปจนถึงบรรทัดที่ 14 แล้ว




ช) ข้อมูลในหน้าต่าง Memory หลังจากทำการรันไปจนถึงบรรทัดที่ 14 แล้ว แถบสีแสดงค่าข้อมูลในหน่วยความจำหลังทำคำสั่ง LD (2062),A

**รูปที่ ข.23(ต่อ)** ข้อมูลในหน้าต่างโปรแกรมขณะทำการรันโปรแกรมแบบสเต็ป

2) เขียนโปรแกรมภาษา Assembly เพื่อทำการกำหนดข้อมูลในรีจิสเตอร์ต่าง ๆ ให้สามารถทำงานได้เหมือนกับโปรแกรมข้อ 1.11 แต่ใช้คำสั่งโอนย้ายข้อมูลแบบ 16 บิต

2.1) เติมโปรแกรมลงในช่องว่างข้างล่าง

| Memory address | Machine code | | Assembly |
|---|---|---|---|
| 0000 | 3E | 00 | LD A,00H |
| 0002 | 01 | 0201 | LD BC,0102H |
| 0004 | 11 | 0403 | LD DE,0304H |
| 0008 | FF | | RST 38H |

2.2) ทำการ Execute Program และตรวจสอบข้อมูลว่า ถูกต้องหรือไม่ ข้อมูล 16 บิต จะประกอบด้วยข้อมูล 2 ไบต์ โดยมีไบต์สูง (High Order Byte) จะเก็บที่แอดเดรสสูงและไบต์ต่ำ (Low Order Byte) จะถูกเก็บที่แอดเดรสต่ำกว่า เช่น ข้อมูล 00FF ถูกเก็บที่หน่วยความจำตั้งแต่ 0000H จะได้เป็นดังนี้

0000H จะมีข้อมูล FFH

0001H จะมีข้อมูล 00H

3) ทำการทดลองโดยการป้อนโปรแกรมตามที่กำหนด ลงในหน้าต่างอิดิเตอร์ แล้วทำการแอสเซมเบลอ ดูผลของการแอสเซมเบลอในหน้าต่าง แล้วนำมาบันทึกลงในผลการทดลอง

| Address | Machine Code | Operand |
|---|---|---|
| 0000 | 3E 01 | LD A,01H |
| 0002 | 06 20 | LD B, 20H |
| 0004 | 0E 60 | LD C, 60H |
| 0006 | 26 20 | LD H, 20H |
| 0008 | 2E 61 | LD L, 61H |
| 000A | DD5600 | LD D, (IX+00H) |
| 000D | DD7005 | LD (IY+05), B |
| 0010 | 74 | LD (HL), H |
| 0011 | 5E | LD E, (HL) |
| 0012 | DD36105E | LD (IX+10H), 5EH |
| 0016 | 0A | LD A, (BC) |
| 0017 | 326220 | LD (2062H), A |
| 001A | 5F | LD E, A |
| 001B | FF | RST 18H |

4) ทำการรันโปรแกรมในข้อที่ 3 แล้วดูผลการทดลอง โดยในการรันนั้นทำการเลือกการรันแบบ Step เพื่อดูการเปลี่ยนแปลงในแต่ละรีจิสเตอร์ แล้วอธิบายการทำงานของโปรแกรม

**สามารถอธิบายได้คือ**

จากโปรแกรมบรรทัดที่ 1 ถึง 5 เป็น การนำข้อมูลใส่ในรีจิสเตอร์ โดยการอ้างตำแหน่งแบบทันที บรรทัดที่ 6 และ 7 เป็นการเคลื่อนย้ายข้อมูล โดยอ้างตำแหน่งดัชนี ซึ่งจะนำข้อมูลในหน่วยความจำตำแหน่งที่ IX+00H ชี้อยู่ มาเก็บในรีจิสเตอร์ D ในบรรทัดที่ 8,9 เป็นการ โอนย้ายข้อมูลโดยการอ้างตำแหน่งแบบผ่านรีจิสเตอร์ ซึ่งจากโปรแกรมใช้รีจิสเตอร์ HL เป็นตัวชี้ตำแหน่งหน่วยความจำ ในบรรทัดที่ 10 เป็นการใช้อินเด็กซ์รีจิสเตอร์ชี้ตำแหน่งหน่วยที่จำนำข้อมูล ไปเก็บ บรรทัดที่ 11 เป็นการใช้รีจิสเตอร์ BC ไปชี้ตำแหน่งข้อมูลในหน่วยความจำ ที่ จะมาเก็บในรีจิสเตอร์ A บรรทัดที่ 13 เป็นการอ้างตำแหน่งแบบรีจิสเตอร์ และบรรทัดทุดท้ายคือการให้โปรแกรมกระโดด ไปทำงานในตำแหน่งที่ต้องการโดยอ้างตำแหน่งในหน้าศูนย์

## สรุปผลการทดลอง

จากการทดลองเขียนโปรแกรมในข้อที่ 3 แต่ละคำสั่งนั้นจะมี วิธีการเข้าถึงข้อมูลอยู่หลายแบบที่แตกต่างกันไป ซึ่งการที่เราจะเลือกใช้วิธีการ เข้าถึงข้อมูลแบบใดนั้นขึ้นอยู่กับว่าเราจะนำข้อมูลนั้นไปใช้งานอะไร และ ขึ้นอยู่กับวิธีการที่จะให้ได้มาซึ่งผลลัพธ์ ซึ่งอาจจะใช้คำสั่งที่แตกต่างกันแต่ได้ผลลัพธ์ที่เหมือนกัน ดังนั้นเราก็ควรเลือกวิธีการที่ง่ายและรวดเร็วที่สุดมาเขียนโปรแกรม

## คำถามท้ายการทดลอง

1) ในการอ้างตำแหน่งข้อมูลแบบอินเด็กซ์ สามารถระบุตำแหน่งของข้อมูลห่างจากตำแหน่งฐานได้เท่าใด

**ตอบ:**-128 ถึง 127

2) ในการโอนย้ายข้อมูลแบบใดที่ไม่ได้ใช้คำสั่ง LD (Load)

**ตอบ:**คำสั่งการติดต่อกับเกี่ยวกับหน่วย INPUT และ OUTPUT

# เฉลยใบงานที่ 2

# คำสั่งเกี่ยวกับคณิตศาสตร์ และการเทียบข้อมูล

## วัตถุประสงค์

1) เพื่อให้เข้าถึงวิธีการในการอ้างแอดเดรสแบบต่าง ๆ ในไมโครโปรเซสเซอร์ Z-80

2) เพื่อให้เข้าใจความหมายของ FLAG ต่าง ๆ ที่มีอยู่ในในไมโครโปรเซสเซอร์ Z-80

3) เพื่อให้เข้าใจเกี่ยวกับคำสั่งทางคณิตศาสตร์และคำสั่งตรรกะ

4) สามารถนำคำสั่งต่าง ๆ มาเขียนเป็นโปรแกรมทางคณิตศาสตร์ได้

5) เข้าใจเทคนิคการเขียนโปรแกรมบวก และลบเลขไบนารีได้

## ลำดับขั้นการทดลอง

1) โปรแกรมต่อไปนี้ ใช้บวกลบ ข้อมูลในรีจิสเตอร์ B กับ รีจิสเตอร์ C เข้าด้วยกัน ผลที่ได้เก็บใน DE ทำการเปิดหน้าต่างอิดิเตอร์ขึ้นมาแล้วทำการป้อนโปรแกรม และบันทึกไฟล์ จากนั้นทำการแอสเซมเบลอโปรแกรม แล้วทำรันโปรแกรมบันทึกผลการทดลองลงในตารางที่ ข.10

| Address | Machine Code | Label | Operand | Comment |
|---|---|---|---|---|
| 0000 | 78 | | LD A, B | ; นำ (B) => เก็บใน (A) |
| 0001 | 81 | | ADD A, C | ; (A+B) => A |
| 0002 | 5F | | LD E, A | ; ผลใน A เก็บใน E |
| 0003 | 3E 00 | | LD A, 0H | ; เคลียร์ A เป็น 0 |
| 0005 | CE 00 | | ADC A,0H | ; A+0+C |
| 0007 | 57 | | LD D, A | นำ (A) =>(D) |
| 0008 | DF | | RST 38H | |

ตารางที่ ข.10 บันทึกผลการทดลอง

| ค่าเริ่มต้นของรีจิสเตอร์ | | ผลของโปรแกรมเมื่อมีการรัน | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | รีจิสเตอร์ | ผลของแฟลก | | | | | |
| B | C | DE | Sign | Zero | Half | P/V | Neg | Carry |
| 54H | 6BH | 00BF H | 0 | 1 | 0 | 0 | 0 | 0 |
| 8F | A0H | 012F H | 0 | 0 | 0 | 0 | 0 | 0 |
| 37H | 7E | 00B5 H | 0 | 1 | 0 | 0 | 0 | 0 |
| 87H | 91H | 0118 H | 0 | 0 | 0 | 0 | 0 | 0 |

2) โปรแกรมต่อไปนี้ ใช้บวกข้อมูล 16 บิต ในคู่รีจิสเตอร์ BC กับข้อมูลในแอดเดรส 2200H-2201H ผลที่ได้เก็บใน DE ป้อนและรันโปรแกรม แล้วบันทึกผลลงในตารางที่ ข.11

| Address | Machine Code | Label | Operand | Comment |
|---|---|---|---|---|
| 0000 | 3A 00 22 | | LD A, B | ; นำ (B) => เก็บใน (A) |
| 0001 | 81 | | ADD A, C | ; (A+B) => A |
| 0002 | 5F | | LD E, A | ; ผลใน A เก็บใน E |
| 0003 | 3A 01 22 | | LD A, (2201H) | ; ข้อมูลใน 2201H เก็บ A |
| 0005 | 88 | | ADC A,B | ; A+B = A |
| 0007 | 57 | | LD D, A | นำ (A) =>(D) |
| 0008 | DF | | RST 38H | |

ตารางที่ ข.11 บันทึกผลการทดลอง

| ค่าเริ่มต้นของรีจิสเตอร์ และหน่วยความจำ | | ผลของโปรแกรมเมื่อมีการรัน | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | รีจิสตอร์ | ผลของแฟลก | | | | | |
| BC | 2200-2201H | DE | Sign | Zero | Half | P/V | Neg | Carry |
| 0354 H | 1001 H | 1355 H | 0 | 0 | 0 | 0 | 0 | 0 |
| 116A H | 1021 H | 218B H | 0 | 0 | 0 | 0 | 0 | 0 |
| 3110 H | 0010 H | 3120 H | 0 | 0 | 0 | 0 | 0 | 0 |
| 0A01 H | 3101 H | 3B02 H | 0 | 0 | 1 | 0 | 0 | 0 |

3) จากโปรแกรมในข้อ 2 เขียนโปรแกรมใหม่เพื่อให้เป็นโปรแกรมสำหรับการลบเลข ผลลัพธ์เก็บในหน่วยความจำตำแหน่ง 2202-2203H (แอดเดรสสูงเก็บไบต์สูง แอดเดรสไบต์ต่ำเก็บไบต์ต่ำ) บันทึกผลการทดลองลงในตารางที่ ข.12

**ตารางที่ ข.12** บันทึกผลการทดลอง

| ค่าเริ่มต้นของ รีจิสเตอร์ และหน่วยความจำ | | ผลของโปรแกรมเมื่อมีการรัน | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | หน่วยความจำ | ผลของแฟลก | | | | | |
| BC | 2200-2201H | 2202-2203H | Sign | Zero | Half | P/V | Neg | Carry |
| 354 H | 1001 H | **F344 H** | **0** | **0** | **0** | **1** | **1** | **1** |
| 116A H | 1021 H | **0149 H** | **0** | **0** | **0** | **0** | **1** | **0** |
| 3110 H | 0010 H | **3100 H** | **0** | **0** | **0** | **0** | **1** | **0** |
| A01 H | 3101 H | **D900 H** | **0** | **0** | **0** | **1** | **1** | **1** |

## สรุปผลการทดลอง

จากการทดลอง การใช้คำสั่งทางคณิตศาสตร์ เช่นบวก หรือลบ นั้น ล้วนมีผลต่อค่า Flag Registor ซึ่งแล้วแต่ว่าผลจากการใช้คำสั่งนั้นแล้วเกิดการเปลี่ยนแปลงอย่างไรบ้างกับค่าข้อมูลในแอคคิวมูเลเตอร์

## คำถามท้ายการทดลอง

1) Over Flow หมายถึง อะไร จะเกิดขึ้นเมื่อใด

Over Flow ก็คือ เมื่อซีพียูกระทำคำสั่งทางคณิตศาสตร์แล้วเกิดค่าเกินขึ้น ซึ่งค่าใน แอคคิวมูเลเตอร์มีขนาด 8 บิต มันสามารถแสดงค่าของเลขฐานสองได้จาก –128 ถึง 127 ดังนั้นถ้าการกระทำทางผลคณิตศาสตร์มีค่ามากกว่า 127 หรือน้อยกว่า –128 แล้วก็จะทำให้เกิดค่า Over Flow ขึ้น

2) แฟลกรีจิสเตอร์มีประโยชน์อย่างไร

แฟลกรีจิสเตอร์มีประโยชน์ในการแสดง ผลการกระทำคำสั่งทางคณิตศาสตร์ และ ลอจิก และคำสั่งที่มีผลต่อแอคคิวมูเลเตอร์

# เฉลยใบงานที่ 3
# กลุ่มคำสั่งการกระโดด และการทำซ้ำ

## วัตถุประสงค์

1) เข้าใจการทำงานของกลุ่มคำสั่งการกระโดดแบบมีเงื่อนไข และไม่มีเงื่อนไข

2) รู้จักวิธีการสั่งให้โปรแกรมเคาน์เตอร์กระโดดไปยังตำแหน่งหน่วยความจำที่ต้องการได้ตามเงื่อนไขของแฟลกต่าง ๆ

3) เข้าใจเทคนิคการออกแบบโปรแกรมทำซ้ำลูป และหน่วงเวลา

4) ฝึกหัดการใช้คำสั่งกระโดด และโปรแกรมทำซ้ำ

## ลำดับขั้นการทดลอง

1) ทำการทดลองตามโปรแกรม ในคำสั่งนี้เป็นการนำค่าในรีจิสเตอร์ C ใส่ในหน่วยความจำที่รีจิสเตอร์ HL ชี้โดยมีการทำซ้ำจนกว่ารีจิสเตอร์ B จะมีค่าเป็น 0 โดยดูจากแฟลกรีจิสเตอร์

| Address | Machine Code | Label | Operand | Comment |
|---|---|---|---|---|
| 0000 | 21 00 22 | START | LD HL, 2000H | ; กำหนดตำแหน่ง |
| 0003 | 06 FF | | LD B, 10 H | ; รีจิสเตอร์ B เป็นตัวนับ |
| 0005 | 0E 11 | | LD C, 11H | ;.ให้ค่าข้อมูลเป็น 11H |
| 0007 | 71 | LOOP | LD (HL), C | ; ใส่ค่าใน C ใน (HL) |
| 0008 | 23 | | INC HL | ; HL<= HL+1 |
| 0009 | 10 FC | | DJNZ LOOP | ; B-1 วนถ้า B < > 0 |
| 000B | DF | | RST 38H | ; หยุดโปรแกรม |

2) กำหนดให้ C = 11H แล้วทำการรันโปรแกรม แล้วทำการบันทึกผลการทดลอง

ผลลัพธ์ในหน่วยความจำ 2200-220F เป็น 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 00

3) ทดลองเปลี่ยนค่าในบรรทัดที่ 3 ของโปรแกรมใหม่ ซึ่งเดิม C = 11H เป็น 00H แล้วรันโปรแกรมใหม่อีกครั้ง แล้วทำการบันทึกผลการทดลอง

ผลลัพธ์ในหน่วยความจำ 2200-220F เป็น 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

4) จงทำการเขียนโปรแกรมที่มีลักษณะดังข้อ 1 แต่ให้ค่าข้อมูลที่เก็บในรีจิสเตอร์ C นั้นเพิ่มค่าทุกครั้งเมื่อเมื่อเพิ่มค่าตำแหน่งหน่วยความจำ ซึ่งถ้าโปรแกรมถูกต้องจะได้ข้อมูลใน ตำแหน่ง 2000 – 200F เป็น **00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F**

สามารถเขียนโปรแกรมได้ดังนี้

| **Address** | **Machine Code** | **Label** | **Operand** | **Comment** |
|---|---|---|---|---|
| 0000 | 21 00 22 | START | LD HL, 2000H | ; กำหนดตำแหน่ง |
| 0003 | 06 FF | | LD B, 10 H | ; รีจิสเตอร์ B เป็นตัวนับ |
| 0005 | 0E 11 | | LD C, 00 H | ;.ให้ค่าข้อมูลเป็น 11H |
| 0007 | 71 | LOOP | LD (HL), C | ; ใส่ค่าใน C ใน (HL) |
| 0008 | 0C | | INC C | ;เพิ่มค่าข้อมูลใน C |
| 0009 | 23 | | INC HL | ; HL<= HL+1 |
| 000A | 10 FC | | DJNZ LOOP | ; B-1 วนถ้า B < > 0 |
| 000C | DF | | RST 38H | ; หยุดโปรแกรม |

**สรุปผลการทดลอง**

จากการทดลองสามารถทราบถึงการวนลูปอย่างง่ายโดยการใช้ DJNZ แต่การกระโดด และการทำซ้ำนั้น มีหลายคำสั่ง มากซึ่ง DJNZ ก็เป็นเพียงแค่ส่วนหนึ่งของการทำซ้ำและกระโดดอย่างง่ายผู้ใช้ควรศึกษาคำสั่งอื่นๆ เช่น JP JR ด้วย

# เฉลยใบงานที่ 4

# การแสดงผลด้วย LED Metrix

## วัตถุประสงค์

1) เพื่อให้รู้จักการแสดงผลอย่างง่าย ด้วย LED Metrix

2) เพื่อให้รู้จักการสแกน

3) เพื่อให้เกิดความคิดในการออกแบบ

4) เพื่อให้เข้าใจการต่ออุปกรณ์ร่วมกับไมโครโปรเซสเซอร์ Z-80

## ลำดับขั้นการทดลอง

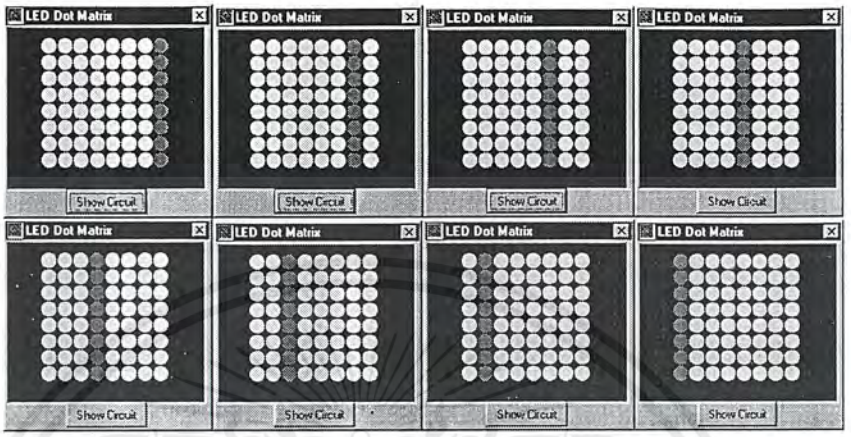
1) ทำการทดลองโดยเขียนโปรแกรม การสั่งให้แอลอีดีมุมซ้ายติด สามารถเขียนโปรแกรมได้ดังนี้

```
ld      a,80h
out     (83h),a
ld      a,10000000b
out     (80h),a
ld      a,10000000b
out     (81h),a
rst     38h
```

2) เขียนโปรแกรมให้แอลอีดีสแกนทางหลัก ตามนี้คือ

```
ld      a,80h
out     (83h),a
ld      a,11111111b
out     (81h),a
ld      a,00000001b
out     (80h),a
rl      a
jp      0ah
```

แล้วทำการแอสเซมเบลอร์และรันดูผลการทดลองจะได้ผลการทดลองในหน้าต่าง LED Metrix แสดงได้ดังรูปที่ ข.24




**รูปที่ ข.24** ผลการทดลองโปรแกรมแสดงทางหลัก

5) เขียนโปรแกรมให้แอลอีดีสแกนทางแถว
สามารถเขียนได้ดังนี้

```
ld      a,80h
out     (83h),a
ld      a,11111111b
out     (80h),a
ld      a,00000001b
out     (81h),a
rl      a
jp      0ah
```

หมายเหตุ อุปกรณ์ทุกตัวต่อกับ 8255 จำลองอยู่ที่ตำแหน่ง 80H

## สรุปผลการทดลอง

จากการทดลองการที่จะทำให้ LED แต่ละดวงติดนั้นต้องให้ข้อมูลทางแถวเป็น 1 และข้อมูลในทางสดมภ์เป็น 1 ด้วย LED จึงจะติด

## คำถามท้ายการทดลอง

1) อธิบายการทำงานของโปรแกรมไฟวิ่ง ตามความเข้าใจ

หลักการทำให้ไฟวิ่งนั้นทำได้โดยส่งข้อมูลไปยังพอร์ตที่ LED นั้นอยู่ตามลักษณะของวงจรของ LED ว่าจะสั่งให้ LED ติดนั้นต้องส่งข้อมูลอย่างไร เมื่อทราบว่าจะส่งข้อมูลอะไรแล้วเราทำการส่งข้อมูลนั้นออกไปยังพอร์ต จากนั้นก็มาทำการหมุนข้อมูลที่จะส่งไป แล้วส่งข้อมูลไปที่พอร์ตอีกครั้งและวนลูปหมุนข้อมูลและส่งค่าออกพอร์ต ก็จะทำให้ LED ที่ต่อ อยู่กับพอร์ตติดเป็นไฟวิ่ง

# เฉลยใบงานที่ 5

# การแสดงผลด้วยตัวแสดงผลเจ็ดส่วน

## วัตถุประสงค์

1) เพื่อให้รู้จักการแสดงผลอย่างง่าย ด้วย ตัวแสดงผลแบบเจ็ดส่วน

2) เพื่อให้รู้จักการสแกน

3) เพื่อให้เกิดความคิดในการออกแบบ

4) เพื่อให้เข้าใจการต่ออุปกรณ์ร่วมกับไมโครโปรเซสเซอร์ Z-80

## ลำดับขั้นการทดลอง

1) ทำการทดลองโดยการเขียนโปรแกรมให้ตัวแสดงผลเจ็ดส่วนติดเป็นเลข 0 แล้วสแกนพอร์ตไปด้วยได้ตามโปรแกรมด้านล่างคือ

| Address | Machine Code | Operand | | Comment |
|---|---|---|---|---|
| 0000 | 3E 80 | LD | A,80H | ; Initial 8255 |
| 0002 | D3 83 | OUT | (83H),A | ; Initial 8255 |
| 0004 | 37 | SCF | | ;.ให้แครี่แฟลกเป็น 1 |
| 0005 | 3E 3F | LD | A,3F H | ; ให้แสดงเลข 0 |
| 0007 | D3 80 | OUT | (80H),A | ; ส่งค่าข้อมูลออก พอร์ต A ของ 8255 |
| 0009 | 3E EE | LD | A,11101110B | ;กำหนดตำแหน่งหลักที่จะติดเป็น 0 |
| 000B | D3 81 | OUT | (81H),A | ; ส่งค่าข้อมูลออก พอร์ต B ของ 8255 |
| 000D | CB 17 | RL | A | ;หมุนค่าตำแหน่งที่สแกนไปทางขวา |
| 000F | C3 0B 00 | JP | 0B H | ;วนลูป ส่งค่าออกพอร์ตอีกครั้ง |

ดูผลการทดลองจากหน้าต่าง Seven Segment

2) จงทำการเขียนโปรแกรมแสดงค่า 0 – 9 ที่ตัวแสดงผลหลักขวามือสุด แล้ววนแสดงค่าจนกว่าจะหยุดโปรแกรมด้วยปุ่ม Stop

| Address | Machine Code | Operand | Comment |
|---|---|---|---|
| 0000 | 3E80 | LD A,80H | ;Initial |
| 0002 | D3 83 | OUT (83H),A | |
| 0004 | 3E F7 | LD A,11110111B | ;กำหนดตำแหน่งหลักขวามือสุดติด |
| 0006 | D3 81 | OUT (81H),A | |
| 0008 | 3E 3F | LD A,3F H | ;เลข 0 |
| 000A | D3 80 | OUT (80H),A | |
| 000C | 3E 06 | LD A,06 H | ;เลข 1 |
| 000E | D3 80 | OUT (80H),A | |
| 0010 | 3E 5B | LD A,5B H | ;เลข 2 |
| 0012 | D3 80 | OUT (80H),A | |
| 0014 | 3E 4F | LD A,4F H | ;เลข 3 |
| 0016 | D3 80 | OUT (80H),A | |
| 0018 | 3E 66 | LD A,66 H | ;เลข 4 |
| 001A | D3 80 | OUT (80H),A | |
| 001C | 3E 6D | LD A,6D H | ;เลข 5 |
| 001E | D3 80 | OUT (80H),A | |
| 0020 | 3E 7D | LD A,7D H | ;เลข 6 |
| 0022 | D3 80 | OUT (80H),A | |
| 0024 | 3E 07 | LD A,07 H | ;เลข 7 |
| 0026 | D3 80 | OUT (80H),A | |
| 0028 | 3E 7F | LD A,7F H | ;เลข 8 |
| 002A | D3 80 | OUT (80H),A | |
| 002C | 3E 6F | LD A,6F H | ;เลข 9 |
| 002E | D3 80 | OUT (80H),A | |
| 0030 | C3 08 00 | JP 08H | ;กระโดดไปทำที่เลข 0 อีกครั้ง |

**หมายเหตุ** อุปกรณ์ทุกตัวต่อกับ 8255 จำลองอยู่ที่ตำแหน่ง 80H

## สรุปผลการทดลอง

จากผลการทดลองที่ได้ทำให้ทราบว่าเราสามารถกำหนดให้ เซกเมนต์ใดๆ ใน ตัวแสดงผลแบบเจ็ดส่วนติดได้ไม่จำกัดรูปแบบ แล้วแต่การออกแบบของผู้เขียนโปรแกรม และชนิดของตัวแสดงผลแบบเจ็ดส่วนว่าเป็น คอมมอนแอโนด หรือ คอมมอนแคโทด

## คำถามท้ายการทดลอง

1) ถ้าใช้ตัวแสดงผลเป็นแบบคอมมอนแอโนด แล้วรูปแบบรหัสการแสดงผลของตัวแสดงผลเจ็ดส่วนจะเป็นอย่างไร

| เลข | . | g | f | e | d | c | b | A | รหัส |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | C0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | F9 |
| 2 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | A4 |
| 3 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | B0 |
| 4 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 99 |
| 5 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 92 |
| 6 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 82 |
| 7 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | F8 |
| 8 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 80 |
| 9 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 90 |
| A | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 88 |
| B | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 83 |
| C | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | C6 |
| D | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | A1 |
| E | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 86 |
| F | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 8E |

# เฉลยใบงานที่ 6

# การควบคุมมอเตอร์ไฟฟ้ากระแสตรง

## วัตถุประสงค์

1) เพื่อให้รู้จักการควบคุมให้มอเตอร์ไฟฟ้ากระแสตรงด้วยไมโครโปรเซสเซอร์ Z-80

2) เพื่อให้รู้จักการควบคุมทิศทางของมอเตอร์ไฟฟ้ากระแสตรง

3) เพื่อให้เกิดความคิดในการออกแบบ

4) เพื่อให้เข้าใจการต่ออุปกรณ์ร่วมกับไมโครโปรเซสเซอร์ Z-80

## ลำดับขั้นการทดลอง

1) ทำการทดลองโดยเขียนโปรแกรม การสั่งให้มอเตอร์หมุนตามเข็มนาฬิกาโดยโปรแกรมดังนี้คือ

| Address | Machine Code | Label | Operand | | Comment |
|---|---|---|---|---|---|
| 0000 | 3E 80 | | LD | A,80H | ;Initial 8255 |
| 0002 | D3 83 | | OUT | (83H),A | |
| 0004 | 06 3F | | LD | B,3FH | ;กำหนดช่วงเวลาที่ให้หมุน |
| 0006 | 3E FF | | LD | A,11111111B | ;ให้หมุนตามเข็มนาฬิกา |
| 0008 | D3 82 | | OUT | (82H),A | ;ส่งข้อมูลไปควบคุม |
| 000A | 10 FE | LOOP: | DJNZ | LOOP | ;ลดค่าเวลาลงจนครบ |
| 000C | 3E 00 | | LD | A,00000000B | ;ให้มอเตอร์หยุดหมุน |
| 000E | D3 82 | | OUT | (82H),A | |
| 0010 | FF | | RST | 38H | ;จบการทำงาน |

แล้วทำการแอสเซมเบลอร์และรันดูผลการทดลอง

2) จงเขียนโปรแกรมให้มอเตอร์หมุนตามเข็มนาฬิกา แล้วให้หมุนทวนเข็มนาฬิกา จึงจบการทำงาน

| Address | Machine Code | Label | Operand | | Comment |
|---|---|---|---|---|---|
| 0000 | 3E 80 | | LD | A,80H | ;Initial 8255 |
| 0002 | D3 83 | | OUT | (83H),A | |
| 0004 | 06 3F | | LD | B,3FH | ;กำหนดช่วงเวลาที่ให้หมุน |
| 0006 | 3E FF | | LD | A,11111111B | ;ให้หมุนตามเข็มนาฬิกา |
| 0008 | D3 82 | | OUT | (82H),A | ;ส่งข้อมูลไปควบคุม |
| 000A | 10 FE | LOOP: | DJNZ | LOOP | ;ลดค่าเวลาลงจนครบ |
| 000C | 3E FD | | LD | A,11111101B | ;ให้หมุนทวนเข็มนาฬิกา |
| 000E | D3 82 | | OUT | (82H),A | |
| 0010 | 06 3F | | LD | B,3F H | ;กำหนดช่วงเวลาที่หมุน |
| 0012 | 10 FE | LOOP1: | DJNZ | LOOP1 | ;ลดค่าเวลาลง จนครบ |
| 0014 | 3E 00 | | LD | A,00000000B | ;.ให้มอเตอร์หยุดหมุน |
| 0016 | D3 82 | | OUT | (82 H),A | |
| 0018 | FF | | RST | 38 H | ;จบการทำงาน |

หมายเหตุ อุปกรณ์ทุกตัวต่อกับ 8255 จำลองอยู่ที่ตำแหน่ง 80H

## สรุปผลการทดลอง

การทำให้มอเตอร์ หมุนหรือหยุดหมุนควบคุมได้ที่ พอร์ต C0

การควบคุมทิศทางการหมุนของมอเตอร์ทำได้โดยควบคุมที่พอร์ต C1

## คำถามท้ายการทดลอง

1) อธิบายการทำงานของโปรแกรมควบคุมมอเตอร์ไฟฟ้ากระแสตรงตามข้อ 2 มาพอเข้าใจ

โปรแกรมควบคุมมอเตอร์ไฟฟ้ากระแสตรงนั้นจะทำการส่งข้อมูลออกไปที่พอร์ต C0 และ C1 เพื่อทำการควบคุมให้มอเตอร์หยุด หมุนตามเข็มนาฬิกา หรือหมุนทวนเข็มนาฬิกา ซึ่งก็สามารถควบคุมการจ่ายไฟได้จากพอร์ต C0 และควบคุมทิศทางได้จาก พอร์ต C1

# ใบงานที่ 7
# การควบคุมสเต็ปปิ้งมอเตอร์

## วัตถุประสงค์

1) เพื่อให้รู้เกี่ยวกับ เกี่ยวกับ สเต็ปปิ้งมอเตอร์แบบต่าง ๆ

2) เพื่อให้รู้เข้าใจการสั่งงาน สเต็ปปิ้งมอเตอร์

3) เพื่อให้เกิดแนวคิดที่จะนำ สเต็ปปิ้งมอเตอร์ไปใช้งาน

## ลำดับขั้นการทดลอง

1) ทำการทดลองโดยเขียนโปรแกรม การสั่งให้สเต็ปปิ้งมอเตอร์หมุนตามเข็มนาฬิกาโดยจ่ายกระแสแบบเวฟโปรแกรมดังนี้คือ

| Address | Machine Code | Label | Operand | | Comment |
|---|---|---|---|---|---|
| 0000 | 3E 80 | | LD | A,80H | ;Initial 8255 |
| 0002 | D3 83 | | OUT | (83H),A | |
| 0004 | 3E 88 | | LD | A,10001000B | ;ให้หมุนแบบเวฟ |
| 0006 | D3 80 | | OUT | (80H),A | ;ส่งข้อมูลออกพอร์ต |
| 0008 | CB 17 | | RL | A | |
| 000A | C3 06 00 | | JP | 06 H | |

แล้วทำการแอสเซมเบลอละรันดูผลการทดลองจะได้ดังรูปด้านล่างคือ

สเต็ปที่ 1 สเต็ปที่ 2 สเต็ปที่ 3 สเต็ปที่ 4

**รูปที่ ข.25** ผลการรันโปรแกรมควบคุมสเต็ปปิ้งมอเตอร์แบบเวฟ

ลักษณะการหมุนจะเป็นไปตามสเต็บที่ 1 – 4 แล้ววนไปเรื่อยๆ จนกว่าจะกดปุ่ม Stop หรือปุ่ม Reset

2) จงเขียนโปรแกรมให้สเต็ปปิ้งมอเตอร์หมุนตามเข็มนาฬิกา โดยจ่ายกระแสแบบ 2 เฟส

| Address | Machine Code | Label | Operand | Comment |
|---|---|---|---|---|
| 0000 | 3E80 | | ld a,80h | ;Initial 8255 |
| 0002 | D383 | | out (83h),a | |
| 0004 | 3E01 | | Ld a,00000001b | ;กำหนดให้มอเตอร์ |
| 0006 | D380 | | out (80h),a | ;หมุนแบบ 2 เฟส |
| 0008 | 3E03 | | ld a,00000011b | |
| 000A | D380 | | out (80h),a | |
| 000C | 3E02 | | ld a,00000010b | |
| 000E | D380 | | out (80h),a | |
| 0010 | 3E06 | | ld a,00000110b | |
| 0012 | D380 | | out (80h),a | |
| 0014 | 3E04 | | ld a,00000100b | |
| 0016 | D380 | | out (80h),a | |
| 0018 | 3E0C | | ld a,00001100b | |
| 001A | D380 | | out (80h),a | |
| 001C | 3E08 | | ld a,00001000b | |
| 001E | D380 | | out (80h),a | |
| 0020 | 3E09 | | ld a,00001001b | |
| 0022 | D380 | | out (80h),a | |
| 0024 | C30400 | | jp 04h | ;กลับไปทำอีกครั้ง |

**หมายเหตุ** อุปกรณ์ทุกตัวต่อกับ 8255 จำลองอยู่ที่ตำแหน่ง 80H

## สรุปผลการทดลอง

การทำให้สเต็ปปิ้งมอเตอร์หมุนในรูปแบบใดๆ นั้นต้องทราบรูปแบบของสัญญาณที่จะจ่ายให้กับขาต่าง ๆ ของมอเตอร์ก่อนแล้วส่งสัญญาณออกพอรต์ที่ต่ออยู่ วงจรควบคุมมอเตอร์แล้วมอเตอร์ก็จะหมุนตามที่เรากำหนด

## คำถามท้ายการทดลอง

1) อธิบายการทำงานของโปรแกรมควบคุมสเต็ปปิ้งมอเตอร์ตามข้อ 2 มาพอเข้าใจ

จากโปรแกรมในข้อที่ 2 เมื่อเริ่มต้นเราต้องทำการ Initial 8255 ให้พร้อมที่จะใช้งานว่าจะใช้งานเป็นพอร์ตเอาต์พุตก็คือ 80 H หลังจากนั้นก็ส่งข้อมูลลักษณะสัญญาณควบคุมมอเตอร์ในแบบ 2 เฟส ออกไปที่พอร์ต ก็สามารถควบคุมให้สเต็ปปิ้งมอเตอร์หมุนตามแบบ 2 เฟสได้

# ภาคผนวก ค
# คู่มือการใช้งานโปรแกรม

# คู่มือการใช้งานโปรแกรมจำลองการทำงานของ Z-80

## 1. บทนำ

โปรแกรมจำลองการทำงานไมโครโปรเซสเซอร์ Z-80 (Z-80 Simulator Program) ที่สร้างขึ้นนี้เป็นโปรแกรมที่ทำงานบนระบบปฏิบัติการวินโดวส์ ซึ่งมีความสามารถในการจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80 ได้ ดังนั้นในบทนี้จะเป็นการกล่าวถึงความเป็นมาของโปรแกรม และลักษณะการทำงานทั่วไปบนระบบปฏิบัติการบนวินโดวส์ ว่ามีลักษณะเป็นอย่างไร มีข้อดี และข้อเสียอย่างไร นอกจากนี้จะกล่าวถึงประโยชน์ของโปรแกรมจำลองการทำงานของไมโครโปรเซสเซอร์ Z-80 อีกด้วย

ไมโครโปรเซสเซอร์ Z-80 เป็นพื้นฐานของไมโครโปรเซสเซอร์และไมโครคอนโทรลเลอร์ตัวอื่น ๆ แม้ว่าไมโครโปรเซสเซอร์ Z-80 จะเสื่อมความนิยมในการที่จะนำไปใช้งานแล้วก็ตาม แต่ไมโครโปรเซสเซอร์ Z-80 ก็ยังคงรับใช้คนทั่วไปอย่างเดิม อาจกล่าวได้ว่าไมโครโปรเซสเซอร์ Z-80 เป็นไมโครโปรเซสเซอร์อมตะ พบว่ายังในการใช้งานกันอยู่ เริ่มตั้งแต่สถาบันการศึกษาระดับการการเรียน ปวช. จนถึง ปวส. หรือระดับที่สูงขึ้น ได้มีการบรรจุเข้าไว้เป็นวิชาหนึ่งของหลักสูตรการเรียนการสอนทั้งในระดับ ปวช. และปวส. หรือไม่ก็นำไปเป็นเครื่องมือสำหรับนักศึกษาเพื่อใช้ทำโครงงาน

การที่ได้บรรจุไมโครโปรเซสเซอร์ Z-80 ไว้ในหลักสูตรของการเรียนการสอนนั้น จะต้องมีการเรียนการสอนและฝึกหัดทดลองให้เป็นไปตามวัตถุประสงค์ของการเรียนการสอน ซึ่งในการเรียนการสอนนั้นจะมีชุดของการทดลองเพื่อให้ผู้เรียนได้เกิดความรู้ความเข้าใจ แต่มักจะประสบกับปัญหา เช่น ชุดทดลองนั้นไม่เพียงพอ หรือการทดลองไม่ประสบผลสำเร็จเนื่องจากการทดลองประกอบวงจรเพื่อใช้ใช้งานไมโครโปรเซสเซอร์Z-80 นั้นมีปัญหาเนื่องจากการต่อวงจร ซึ่งจะทำให้การเรียนการสอนนั้นเป็นไปได้ช้า จึงมักจะไม่ประสบผลสำเร็จในการเรียนการสอนวิชานี้ เป็นต้น

โปรแกรมจำลองทำงานของไมโครโปรเซสเซอร์ Z-80 นั้นได้มีหลายบริษัทผลิตขึ้นมา แต่โปรแกรมที่สร้างเคยมีผู้สร้างขึ้นมาแล้วนั้น บางโปรแกรมจะมีการทำงานบนดอส แต่เนื่องจากเราต้องการที่จะพัฒนาให้ทำงานบนวินโดวส์ สามารถที่จะจำลองการทำงานได้ โดยการจำลองหน่วยเก็บข้อมูล การติดต่อกับอุปกรณ์ต่อร่าม อีกทั้งยังสร้างขึ้นเพื่อนำไปใช้ในการเรียนการสอน เพื่อช่วยลดปัญหาที่เกิดขึ้น ไม่ว่าจะเป็นการลดเวลาในการเรียนการสอน และค่าใช้จ่ายในการซื้อชุดอุปกรณ์

## 2. ความสามารถของโปรแกรมจำลองการทำงานของ Z-80

1) สามารถจำลองคำสั่งในการทำงานของไมโครโปรเซสเซอร์ Z-80

2) สามารถจำลองหน่วยความจำที่ต่อกับไมโครโปรเซสเซอร์ Z-80

3) สามารถจำลองรีจิสเตอร์ของไมโครโปรเซสเซอร์ Z-80

4) มีภาคแสดงผลจำลองแอลอีดีเมทริกซ์ 1 ชุด

5) มีภาคแสดงผลจำลองแอลซีดี 1 ชุด

6) มีภาคแสดงผลจำลองแบบเจ็ดส่วน 1 ชุด

7) มีการจำลองการทำงานของมอเตอร์

## 3. ความต้องการทางด้านฮาร์ดแวร์ และระบบปฏิบัติการ

ความต้องการทางด้านฮาร์ดแวร์ และระบบปฏิบัติในการทำงานของโปรแกรมจำลองการของโมโครโปรเซสเซอร์ Z-80 ใช้งานนั้น ควรมีส่วนประกอบดังต่อไปนี้

1) เครื่องคอมพิวเตอร์รุ่นเพนเทียม 166 MHz เป็นอย่างต่ำ

2) หน่วยความจำอย่างต่ำ 16 MB

3) ขนาดความจุฮาร์ดดิสก์อย่างน้อย 10 MB

4) จอภาพสี VGA หรือสูงกว่า

5) เมาส์ชนิด 2 ปุ่ม หรือ 3 ปุ่ม

6) เครื่องพิมพ์ที่สามารถพิมพ์ได้ทั้งตัวอักษร และรูปภาพ

## 4. ระบบปฏิบัติการวินโดวส์

วินโดวส์ (Windows) คือ โปรแกรมที่ควบคุมการทำงานของคอมพิวเตอร์ ตั้งแต่การแสดงผลบนจอภาพ การป้อนข้อมูลด้วยคีย์บอร์ด หรือการใช้เมาส์ การเก็บข้อมูลที่เป็นตัวเลข ตัวอักษร และรูปภาพ ตลอดจนการพิมพ์ผลงานออกทางเครื่องพิมพ์ ผู้ใช้ดปรแกรมวินโดวส์จะมีความรู้สึกเหมือนกับว่าจอคอมพิวเตอร์เป็นพื้นที่บนโต๊ะทำงานที่มีอุปกรณ์เครื่องมือช่วยในการทำงานเกือบทุกชนิด เช่น สมุดจดบันทึก เครื่องคิดเลข นาฬิกา ปฏิทิน และเครื่องพิมพ์ดีด เป็นต้น การใช้เครื่องมือต่าง ๆ เหล่านี้ทำได้ง่าย เพราะโปรแกรมได้จัดทำทุกอย่างไว้เป็นรูปภาพ ผู้ใช้ต้องการอะไรก็เพียงแต่เลือกรูปภาพนั้น ก็จะได้สิ่งที่ต้องการทันที เช่น ต้องการบันทึกข้อความนัดหมายในปฏิทินก็เลือกรูปภาพที่เป็นรูปภาพปฏิทินก็สามารถใส่ข้อความได้ทันที การใช้เครื่องมือหลาย ๆ อย่างพร้อมกันก็ไม่มีปัญหา เพียงแต่สภาพบนจอภาพคอมพิวเตอร์จะเหมือนกับโต๊ะที่มีทั้งอกสาร เครื่องมือ

เครื่องใช้วางซ้อนกันอยู่ ปนเปกันไปหมด ซึ่งลักษณะนี้วินโดวส์สามารถทำได้ใกล้เคียงมากซึ่งเรียกว่า เดสทอป (Destop) และการใช้งานวินโดวส์นั้นทำได้ไม่ยากนัก ไม่ต้องจำคำสั่งมากมาย เนื่องจากคำสั่งจะเป็นลักษณะของรูปภาพ เพียงแต่ว่าผู้ใช้อาจจะลืมความหมายของแต่ละภาพ และส่วนที่เป็นคำสั่งก็จะเป็นแบบเมนู เหมือนรายการอาหารให้เลือกใช้ได้อย่างสะดวกสบาย นอกจากนี้ระบบปฏิบัติการวินโดวส์ยังสามารถคบคุมโปรแกรมอื่น ๆ ได้หลายอย่าง ซึ่งมีคุณสมบัติโดดเด่น และในปัจจุบันมีผู้ใช้วินโดวส์มากขึ้น รวมทั้งโปรแกรมที่อยู่ภายใต้การความควบคุมของวินโดวส์ก็มีมากขึ้น จึงเป็นส่วนที่ช่วยสนับสนุนให้งานที่ต้องใช้คอมพิวเตอร์เพิ่มขึ้น

เนื่องจากระบบปฏิบัติการวินโดวส์เป็นโปรแกรมที่ใช้ควบคุมระบบการทำงานของคอมพิวเตอร์ ดังที่ได้กล่าวมาแล้วในข้างต้น ดังนั้นจึงมีความสามารถควบคุมการทำงานของโปรแกรมอื่น ๆ ได้อีกด้วย แต่ระบบปฏิบัติการวินโดวส์ มีการพัฒนาอย่างต่อเนื่อง จึงทำให้โปรแกรมที่เป็นโปรแกรมรุ่นเก่าบางโปรแกรมก็ไม่สามารถที่จะทำงานบนวินโดวส์ได้ อย่างเช่น Wordstar, Lotus และ dBase ที่เกิดขึ้นมาก่อนจึงไม่สามารถนำมาใช้ในระบบของวินโดวส์ ได้อย่างตรง ๆ ต้องอาศัยคำสั่ง และเทคนิคพิเศษพอสมควร ในที่นี้จึงขอแบ่งโปรแกรมออกเป็น 2 ชนิด คือ

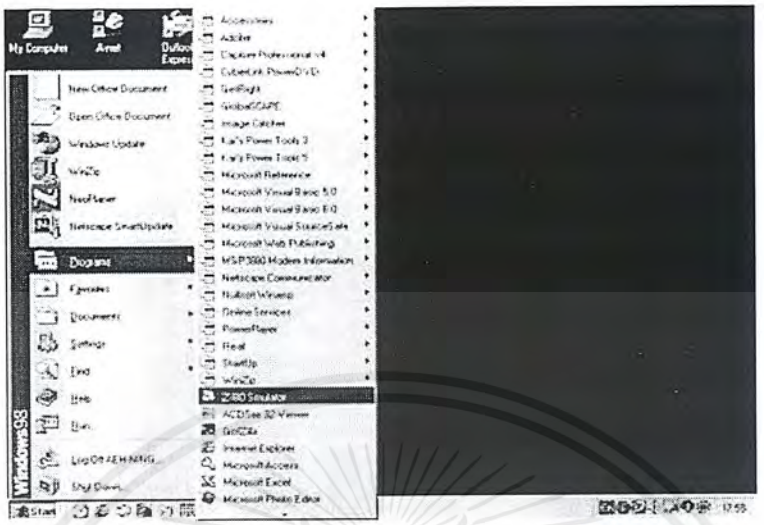
**1) Non-Windows Application** หมายถึง โปรแกรมของดอส (Dos) ซึ่งไม่สามารถที่จะนำไปใช้ในระบบวินโดวส์ เช่น Wordstar, Lotus และ dBase เป็นต้น แต่สามารถใช้เทคนิคพิเศษเพื่อให้ใช้งานร่วมกันได้

**2) Windows Application** หมายถึง โปรแกรมที่ใช้กับระบบวินโดวส์โดยเฉพาะ เช่น Microsoft Excel, Microsoft Word เป็นต้น โปรแกรมเหล่านี้ต้องพึ่งพาระบบปฏิบัติการวินโดวส์ เท่านั้น หากนำไปใช้ ในคอมพิวเตอร์ที่มีแต่ระบบดอสอย่างเดียวก็จะใช้งานไม่ได้

## 5. การติดตั้งโปรแกรม

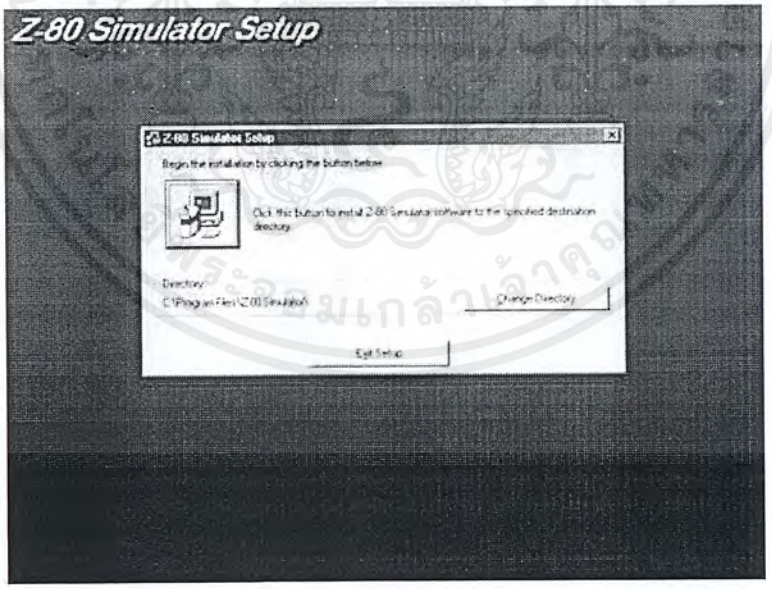
การติดตั้งโปรแกรม Z-80 Simulator Program ทำได้โดยการใส่แผ่นซีดีรอม (CD-ROM) ลงไปในไดรฟ์ D: หรือจะทำการสั่งแผ่นติดตั้งที่เป็นแผ่นดิสก์ที่ 1 ลงในไดรฟ์ A: หลังจากนั้น ทำการเรียกไฟล์ที่มีชื่อ Setup.exe จากนั้นดำเนินการตามขั้นตอนที่โปรแกรมติดตั้งกำหนดไว้ และจะแสดงไดอะล็อกบ็อกซ์ต่าง ๆ ขึ้นมาเพื่อให้กรอกข้อมูลตามลำดับ ดังจะอธิบายดังต่อไปนี้

1) เมื่อทำการรันไฟล์ Setup.exe แล้วจะปรากฏจอภาพดังแสดงในรูปที่ ค.1 ซึ่งจะบอกถึงการติดตั้งโปรแกรม ว่าต้องการที่จะติดตั้งหรือไม่ โดยจะมีปุ่ม OK และปุ่ม Exit Setup



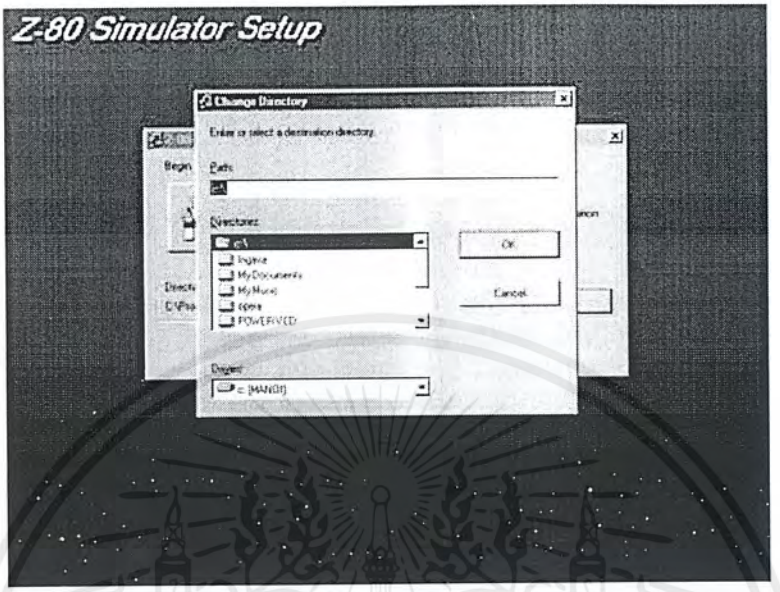

รูปที่ ค.1 หน้าต่างเข้าสู่การติดตั้งโปรแกรม

2) ให้คลิกที่ปุ่ม OK จะปรากฏจอภาพดังแสดงในรูปที่ ค.2 บอกถึงไดเรกทอรีที่ต้องการจะทำการติดตั้งไว้ ซึ่งค่าเริ่มต้นที่กำหนดให้นั้น คือ C:\Program Files\Z-80 Simulator



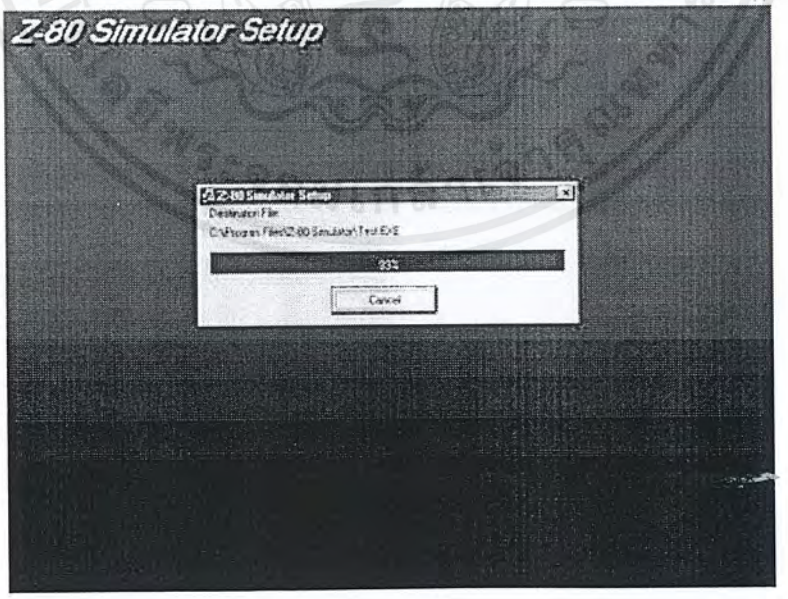

รูปที่ ค.2 ไดเรกทอรีที่จะติดตั้งโปรแกรม

3) ให้คลิกที่ปุ่ม Change Directory จะปรากฏจอภาพดังแสดงในรูปที่ ค.3 ซึ่งแสดงหน้าต่างให้ทำการเลือกเปลี่ยนไดเรกทอรี ตามที่ต้องการ



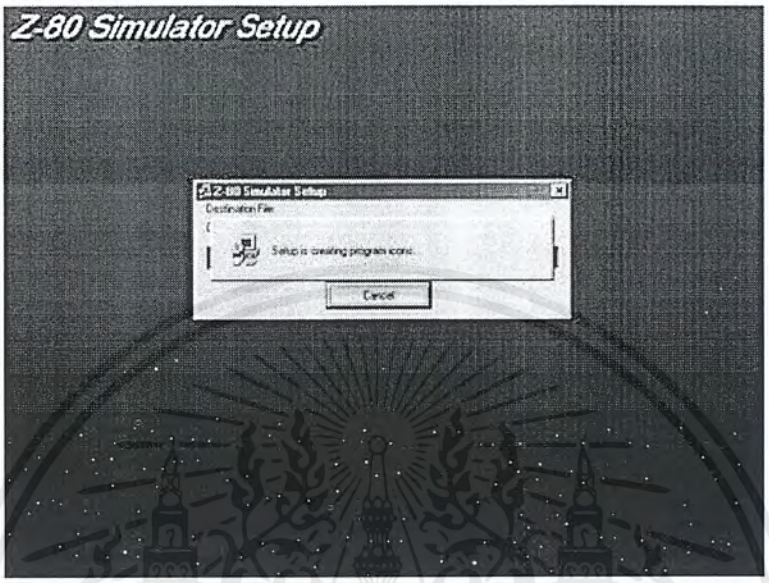

**รูปที่ ค.3** การเปลี่ยนไดเรกทอรีที่จะติดตั้ง

4) เมื่อกลับจากการเปลี่ยนไดเรกทอรีเรียบร้อยแล้ว จะกลับเข้าสู่จอภาพดังในรูปที่ ค.2 ให้คลิกปุ่มอยู่ทางซ้ายมือด้านบน เป็นปุ่มที่มีรูปคอมพิวเตอร์อยู่ จะปรากฏจอภาพดังแสดงในรูปที่ ค.4



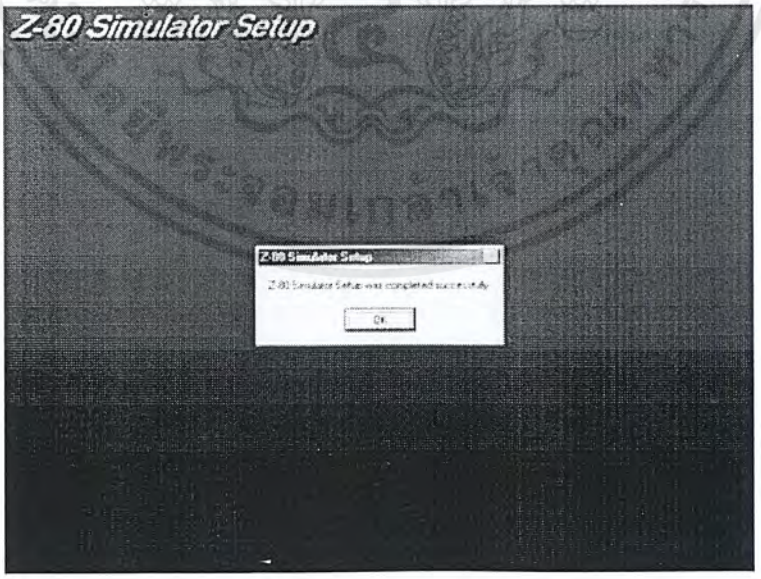

**รูปที่ ค.4** การติดตั้งไฟล์ต่าง ๆ ของโปรแกรม

5) โปรแกรมจะทำการติดตั้งไฟล์ต่างลงในไดเรกทอรีที่กำหนดครบแล้ว จะทำการติดตั้งไฟล์ไอคอนของโปรแกรม ซึ่งจะปรากฏจอภาพดังในรูปที่ ค.5




**รูปที่ ค.5** การติดตั้งไอคอนของโปรแกรม

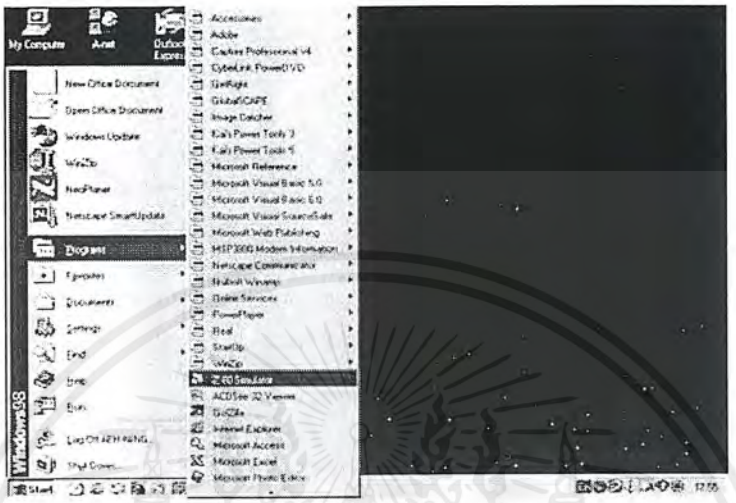
6) เมื่อทำการติดตั้งสมบูรณ์แล้ว จะปรากฏจอภาพแสดงดังในรูปที่ ค.6 คลิกที่ปุ่ม OK การติดตั้ง Z-80 Simulator Program เสร็จสมบูรณ์




**รูปที่ ค.6** การติดตั้งเสร็จสมบูรณ์

## 6. การเข้าสู่โปแกรมจำลองการทำงานของ Z-80

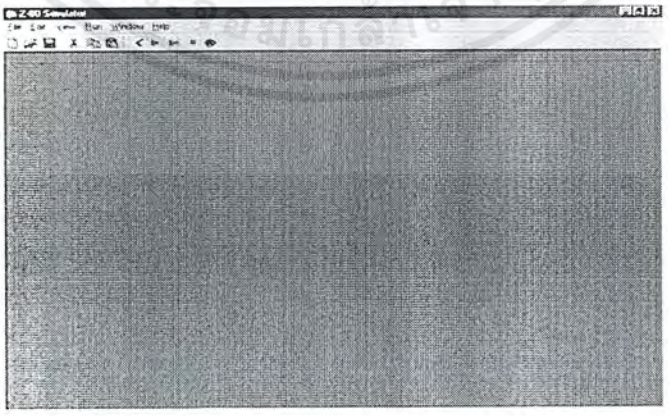
การเข้าสู่การรัน Z-80 Simulator Program สามารถทำได้ดังแสดงในรูปที่ ค.7



**รูปที่ ค.7** การเข้าสู่ Z-80 Simulator Program

จากรูปที่ ค.7 สามารถที่จะเข้าสู่การรัน Z-80 Simulator Program ได้โดยการคลิกที่ Start แล้วเลือกเข้าไปที่โปรแกรมไฟล์ (Program Files) คลิก Z-80 Simulator Program หรือจะคลิกที่ไอคอนของโปรแกรม

เมื่อเข้าสู่โปรแกรม Z-80 Simulator Program จะปรากฏหน้าต่างดังแสดงในรูปที่ ค.8 ซึ่งในเข้าสู่โปรแกรมครั้งแรกนั้น หน้าต่างของโปรแกรมหลักนั้นจะมีเพียงเมนูบาร์และทูลบาร์



**รูปที่ ค.8** หน้าต่าง Z-80 Simulator Program เริ่มเข้าสู่โปรแกรม

## 7. ส่วนประกอบของโปรแกรมจำลองการทำงานของ Z-80

ส่วนประกอบต่าง ๆ ของ Z-80 Simulator Program ประกอบด้วย

1) เมนูบาร์ และทูลบาร์

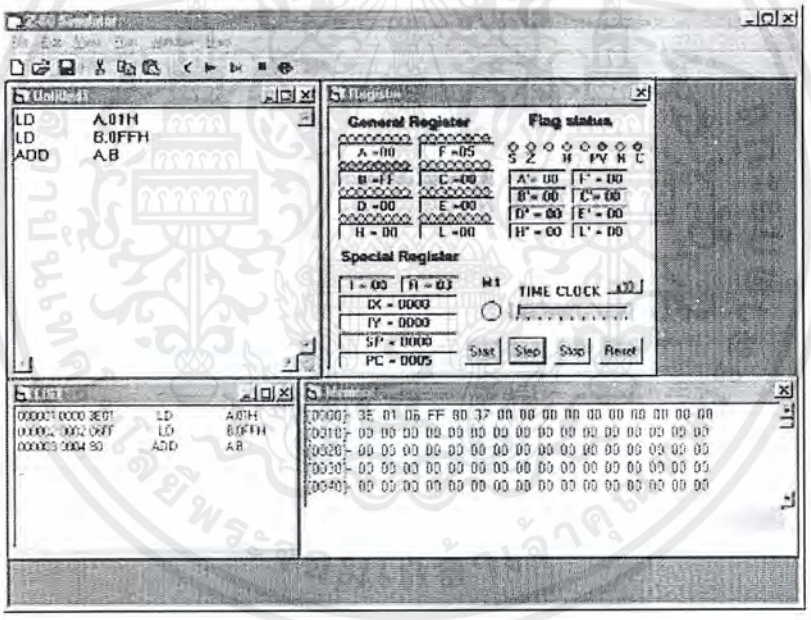
2) หน้าต่างรีจิสเตอร์

3) หน้าต่างหน่วยความจำ

4) หน้าต่างอิดิเตอร์

5) หน้าต่างลิสต์โค้ด

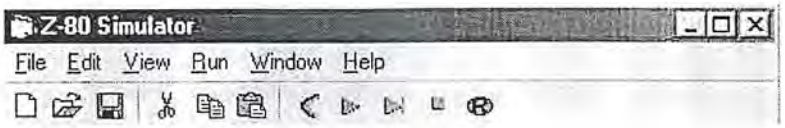
ส่วนประกอบทั้งหมดนี้จะทำงานเมื่อมีการรัน หรือเรียกผ่านส่วนของเมนูบาร์ จะสามารถแสดงหน้าต่างเหล่านี้ออกมาให้เห็น ซึ่งหน้าต่างหลักของโปรแกรมที่ประกอบด้วย 4 ส่วนแสดงดังในรูปที่ ค.9




**รูปที่ ค.9** หน้าต่างโปรแกรมหลักของ Simulator Program

### 1) เมนูบาร์ และทูลบาร์

ส่วนของเมนูบาร์ และทูลบาร์เป็นส่วนที่ทำให้ผู้ใช้สามารถใช้งานได้อย่างสะดวกสบาย โดยการคลิกปุ่ม หรือเมนูก็สามารถทำงานได้ โดยส่วนประกอบของเมนูบาร์ และทูลบาร์ดังแสดงในรูปที่ ค.10




**รูปที่ ค.10** เมนูบาร์ และทูลบาร์

จากรูปที่ ค.10 เมนูบาร์จะประกอบไปด้วยเมนูหลักทั้งหมด 6 เมนูหลัก โดยแต่ละเมนูหลักจะมีเมนูย่อยอยู่ด้วย โดยจะแบ่งไปตามประเภทของคำสั่งของเมนูย่อยว่ามีการทำงานเป็นอย่างไร ซึ่งจะเป็นการง่ายต่อการที่จะเรียกใช้งานได้ง่าย และสะดวกต่อการเรียกใช้งานได้ ซึ่งลักษณะของเมนูบาร์นั้นจะเป็นลักษณะเหมือนกับเมนูของโปรแกรมโดยทั่วไป ประกอบไปด้วยส่วนเมนูไฟล์จะจัดการเกี่ยวกับการเปิดและสร้างไฟล์ใหม่ และการจบการทำงาน ส่วนของเมนูอิดิเตอร์จะเป็นการจัดการเกี่ยวกับข้อความในเท็กบอกซ์ เป็นต้น ในส่วนของเมนูบาร์ได้แบ่งเป็นเมนูหลัก และเมนูย่อยแต่ละเมนูนั้นทำหน้าที่ และเรียกใช้งานที่แตกต่างกัน ดังแสดงในตารางที่ ค.1

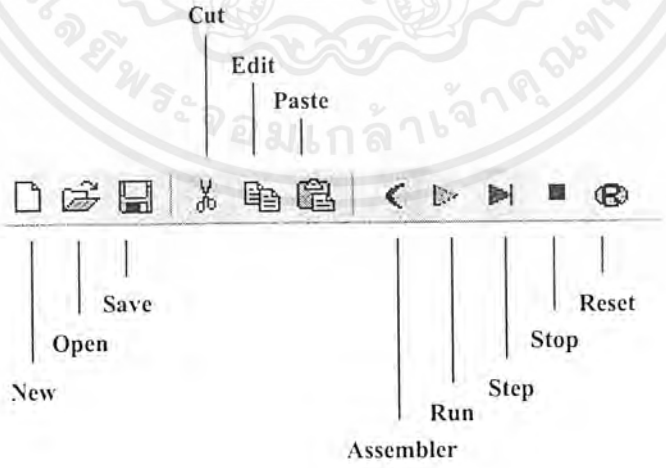
**ตารางที่ ค.1** ส่วนประกอบของเมนูบาร์

| 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 |

ตารางที่ ค.1 (ต่อ) ส่วนประกอบของเมนูบาร์

| 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 |

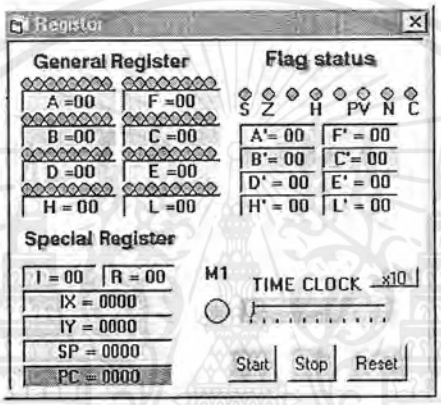
ทูลบาร์จะเป็นลักษณะของรูปภาพเล็ก ๆ ที่แทนคำสั่งของเมนูบาร์ ซึ่งที่สามารถทำงานแทนเมนูบาร์ได้บางคำสั่ง หากต้องการจะเลือกให้ทำงานได้โดยการใช้เมาส์คลิกรูปภาพ ตามคำสั่งในงานที่ต้องการ ซึ่งได้ทูลบาร์ดังแสดงในรูปที่ ค.11




รูปที่ ค.11 เมนูบาร์

### 2) หน้าต่างรีจิสเตอร์

หน้าต่างของรีจิสเตอร์ คือ หน้าต่างแสดงรีจิสเตอร์ที่มีอยู่ในไมโครโปรเซสเซอร์ Z-80 หน้าต่างนี้จะแสดงเมื่อมีการเลือกที่เมนูหลัก View แล้วเลือกที่คำสั่ง Register Window ก็จะแสดงหน้าต่างนี้ และถ้าหากไม่ได้เลือกตามขั้นตอนดังกล่าวขณะที่ทำการรันโปรแกรม หน้าต่างรีจิสเตอร์จะแสดงขึ้นมา โดยหน้าต่างนี้มีหน้าที่ในการแสดงผลของการทำคำสั่งที่มีผลรีจิสเตอร์ทุกคำสั่ง ซึ่งจะแสดงค่าที่เปลี่ยนแปลงเพื่อที่จะทำให้สังเกตการเปลี่ยนแปลงได้ หน้าต่างของรีจิสเตอร์ดังแสดงในรูปที่ ค.12




**รูปที่ ค.12** หน้าต่างรีจิสเตอร์

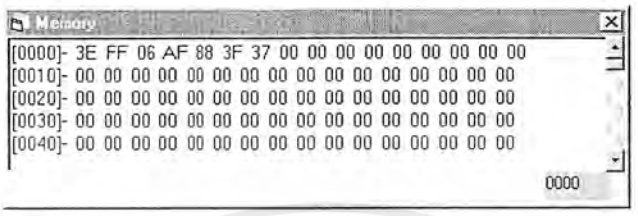
จากรูปที่ ค.12 หน้าต่างรีจิสเตอร์ประกอบด้วยส่วนของรีจิสเตอร์หลัก (General Register) ซึ่งประกอบด้วยรีจิสเตอร์ A, B, C, D, E, F, H และ L ส่วนจิสเตอร์สำรอง และส่วนรีจิสเตอร์เฉพาะ (Special Register) ประกอบด้วยรีจิสเตอร์ขนาด 8 บิต 2 รีจิสเตอร์ คือ รีจิสเตอร์ R และ I รีจิสเตอร์ขนาด 16 บิต คือ รีจิสเตอร์ IX, IY, SPและ PC และในส่วนของสถานะของแฟลกเป็นไฟบอกสถานะต่าง ๆ

นอกจากนั้นในส่วนของหน้าต่างรีจิสเตอร์จะมีส่วนของTime Clock ที่เป็นการกำหนดค่าของเวลาที่ใช้ในการรันคำสั่ง ให้มีการรันเป็นแบบทีละสเต็ป หรือว่าเป็นการรันแบบครั้งเดียวต่อเนื่องกัน และส่วนของปุ่มในการกำหนดการรันหรือหยุดการรันได้โดยที่ไม่ต้องไปเรียกที่เมนูบาร์

### 3) หน้าต่างหน่วยความจำ

หน้าต่างหน่วยความจำ คือ หน้าต่างแสดงข้อมูลหน่วยความจำ หน้าต่างนี้จะแสดงเมื่อมีการเลือกที่เมนูหลัก View แล้วเลือกที่ Memory Window จะแสดงหน้าต่างนี้ ถ้าหากไม่ได้เลือกตามขั้น

ตอนดังกล่าว ขณะที่รันโปรแกรมหน้าต่างหน่วยความจำจะแสดงขึ้นมา ตำแหน่งของหน่วยความจำมีตั้งแต่ 0000H - FFFFH สามารถที่จะเลือกดูข้อมูลในหน่วยความจำได้ โดยการเลื่อนสกอร์บาร์ หน้าต่างของหน่วยความจำสามารถแสดงได้ดังในรูปที่ ค.13




**รูปที่ ค.13** หน้าต่างหน่วยความจำ

### 4) หน้าต่างอิดิเตอร์

หน้าต่างอิดิเตอร์จะเป็นหน้าต่างที่ใช้สำหรับให้ผู้ใช้ทำการสร้างไฟล์ใหม่ หรือ ทำการเปิดไฟล์ที่เขียนไว้แล้วมาทำการแก้ไขใหม่อีกครั้ง ในหน้าต่างนี้จะประกอบไปด้วยเท็กบอกซ์เป็นตัวที่ใช้ในการเขียน และรับอ่านจากการอ่านไฟล์มาแสดงให้ผู้ใช้ได้ การทำงานของน้าต่างนี้จะเหมือนกับอิดิเตอร์ทั่วไป หน้าต่างอิดิเตอร์ดังแสดงในรูปที่ ค.14

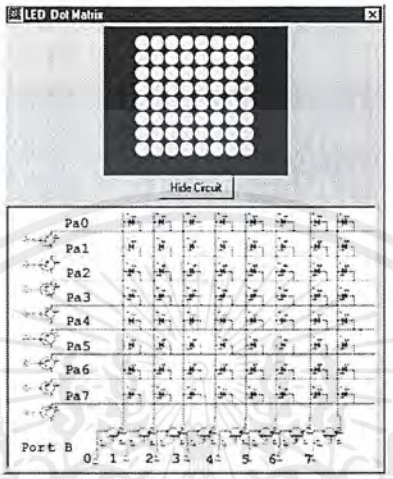
**รูปที่ ค.14** หน้าต่างอิดิเตอร์

## 8. หน้าต่างการอินเตอร์เฟส

หน้าต่างของการอินเตอร์เฟส คือ การต่ออุปกรณ์จำลองเข้ากับไมโครโปรเซสเซอร์ Z-80 ซึ่งสามารถที่จะเรียกใช้งานได้โดยการตั้งค่าของพอร์ต แล้วเขียนโปรแกรมให้แสดงออกสู่หน้าต่างของการอินเตอร์เฟส หน้าต่างอินเตอร์เฟสจะแบ่งออกเป็น 4 ชุด คือ หน้าต่างแสดงผลแอลอีดี, หน้าต่างแสดงผลแบบเจ็ดส่วน และหน้าต่างจำลองการทำงานของมอเตอร์

### 1) หน้าต่างแสดงผลแอลอีดี

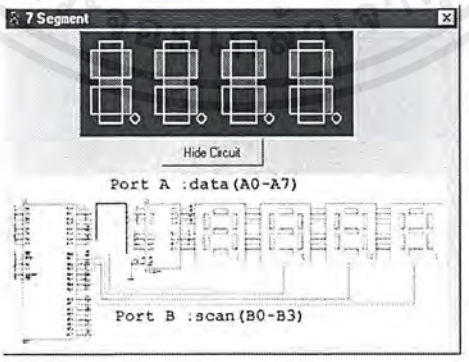
ในภาคแสดงผลแอลอีดีจะจำลองการทำงานการแสดงแสดงผลแอลอีดีเมตริกซ์ขนาด 8 x 8 โดยการสั่งให้ทำงาน จะต้องกำหนดค่าให้กับ 8255 หน้าต่างแสดงผลแอลอีดีดังแสดงในรูปที่ ค.15




**รูปที่ ค.15** หน้าต่างแสดงผลแอลอีดี

### 2) หน้าต่างแสดงผลแบบเจ็ดส่วน

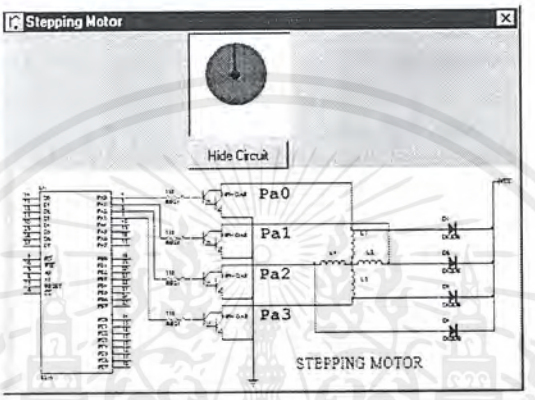
หน้าต่างแสดงผลแบบเจ็ดส่วนนี้ ต้องกำหนดค่าให้กับ 8255 เหมือนหน้าต่างแสดงผลแบบแอลอีดี การใช้งานส่วนนี้จะต้องที่การส่งค่าผ่านฟังก์ชันของ 8255 ซึ่งภาคแสดงผลนี้ประกอบด้วยแบบเจ็ดส่วนทั้งหมด 4 ตัว ซึ่งหน้าต่างแสดงผลแบบเจ็ดส่วนดังแสดงในรูปที่ ค.16




**รูปที่ ค.16** หน้าต่างแสดงผลแบบเจ็ดส่วน

### 3) หน้าต่างจำลองการทำงานของมอเตอร์

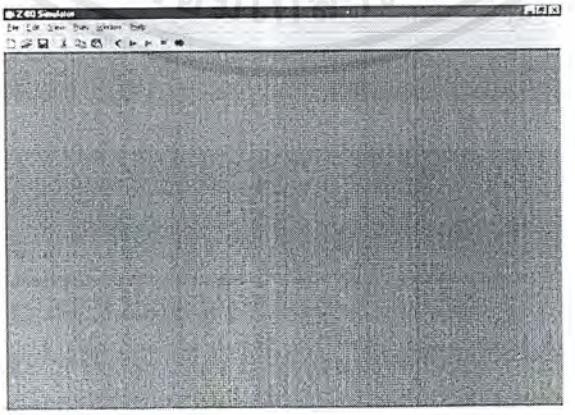
หน้าต่างจำลองมอเตอร์เป็นหน้าต่างจำลองการทำงานของมอเตอร์ โดยการจำลองการทำงานจะต้องกำหนดค่าให้กับ 8255 เป็นตัวติดต่อกับการแสดงผลส่วนของหน้าต่างอินเตอร์เฟสจำลองสเต็ปปิ้งมอเตอร์ดังแสดงในรูปที่ ค.17




**รูปที่ ค.17** หน้าจำลองการทำงานของมอเตอร์

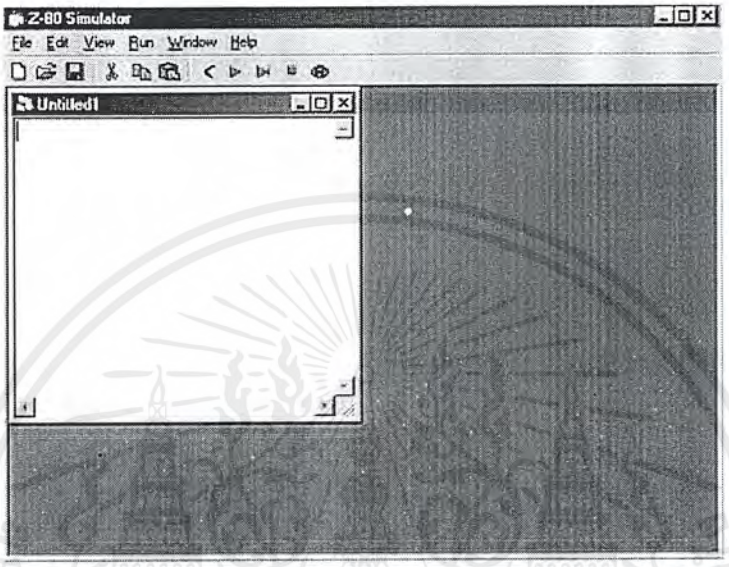
## 9. การใช้งานโปรแกรม

1) การเข้าสู่การใช้งานโปรแกรม สามารถได้โดยขั้นตอนในการเลือกไอคอนของโปรแกรม Z-80 Simulator Program ซึ่งได้กล่าวไว้ในบทที่ เมื่อเลือกเปิดโปรแกรมแล้วจะปรากฏหน้าต่างดังรูปที่ ค.18



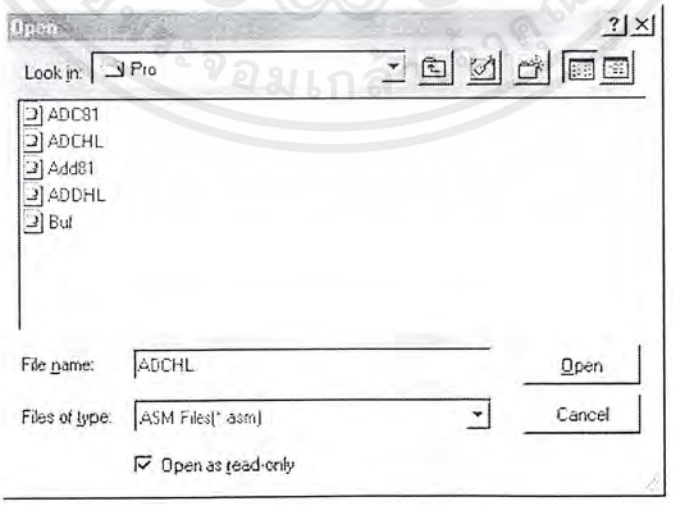
**รูปที่ ค.18** หน้าต่างเริ่มเข้าสู่โปรแกรม

2) เมื่อเข้าสู่โปรแกรมจะไม่เป็นหน้าต่างเปล่าที่ไม่ปรากฏหน้าต่างอื่น ๆ จนกว่าจะไปทำการเลือก การสร้างไฟล์ใหม่ที่เมนูบาร์ คือ New หรือ ที่ทูลบาร์ เมื่อทำการเลือกแล้วจะปรากฏหน้าต่างดังแสดงในรูปที่ ค.19



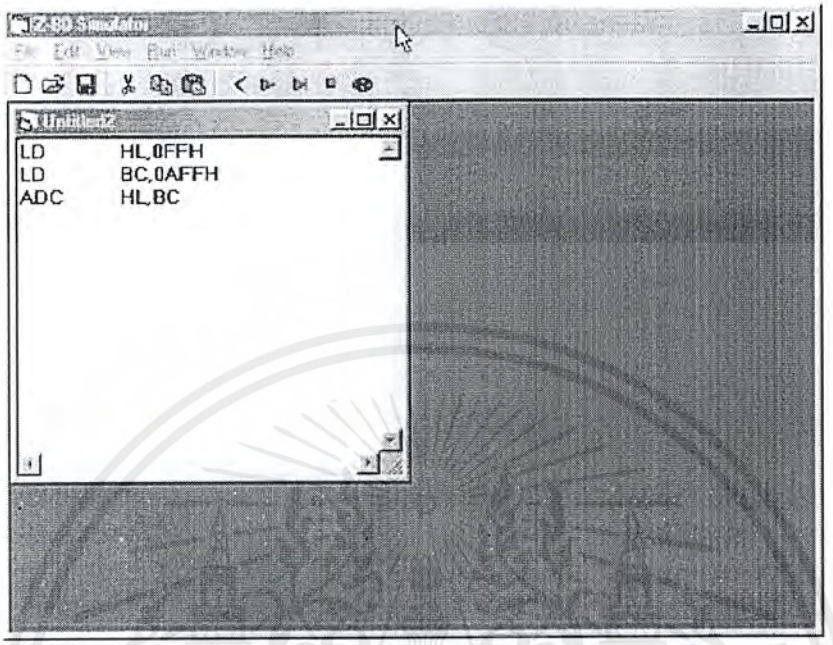

**รูปที่ ค.19** การสร้างไฟล์ใหม่

3) ถ้าหากต้องการที่จะเปิดไฟล์ที่สร้างไว้แล้ว ก็สามารถเลือกการเปิดไฟล์ โดยการเข้าไปที่เมนูไฟล์ แล้วเลือก Open หรือว่าจะเลือกจากทูลบาร์ ซึ่งจะปรากฏหน้าต่างดังแสดงในรูปที่ ค.20



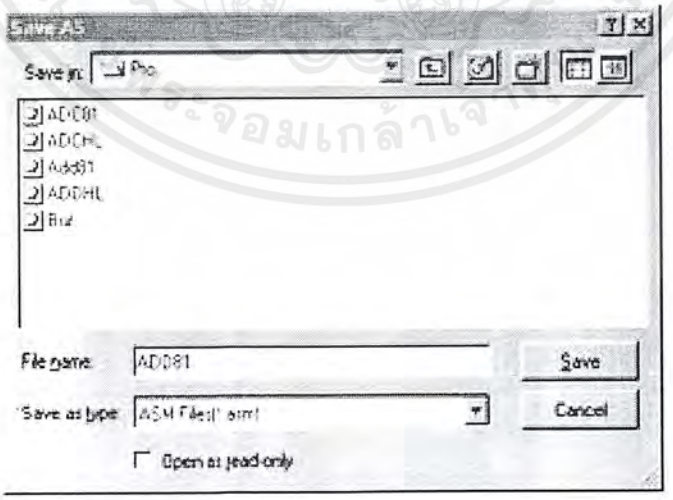

**รูปที่ ค.20** หน้าต่างการเปิดไฟล์

4) เมื่อทำการสร้างไฟล์ โดยการป้อนโปรแกรมที่หน้าต่างอิดิเตอร์ ดังแสดงในรูปที่ ค.21




**รูปที่ ค.21** การป้อนโปรแกรมในหน้าต่างอิดิเตอร์

5) เมื่อทำการป้อนโปรแกรมเรียบร้อยแล้วก็ทำการจัดเก็บไฟล์ ให้มีนามสกุล ASM ซึ่งหน้าต่าง Save As แสดงเมื่อการบันทึกนั้นเป็นการบันทึกไฟล์ครั้งแรก ดังแสดงในรูปที่ ค.22

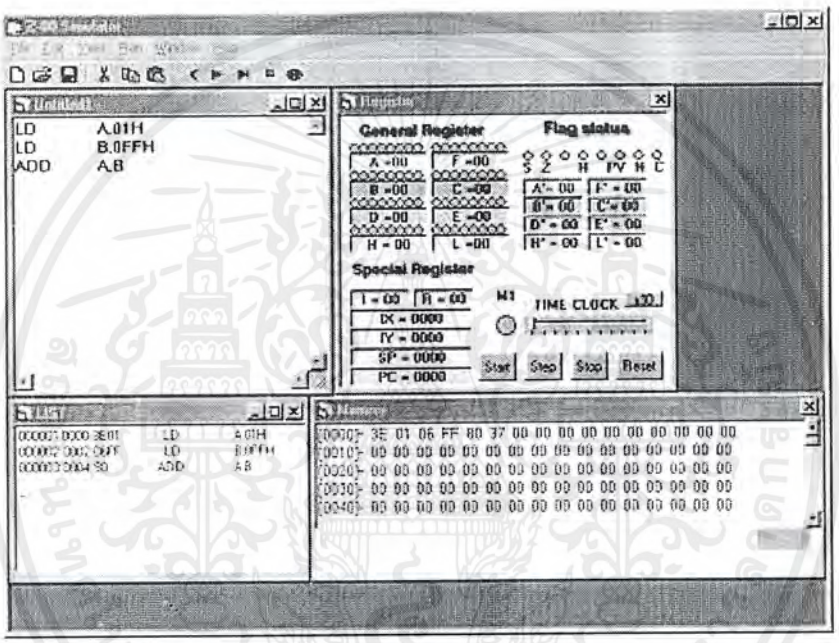



**รูปที่ ค.23** หน้าต่าง Save As

## 10. การ Assembler ไฟล์

เมื่อทำการสร้างไฟล์นามสกุล ASM เรียบร้อยแล้วทำการตรวจสอบว่ามีข้อผิดพลาดหรือไม่ แล้วจึงจะทำการ Assembler เพื่อแปลงไฟล์โปรแกรมที่เป็นลักษณะของเท็กซ์ไฟล์มาเป็นไฟล์ตัวเลขสิบหก ซึ่งขั้นตอนในการ Assembler มีดังนี้

1) ไปที่เมนู Run แล้วทำการเลือก Assembler หรือจะทำการกดคีย์ลัด F7 ก็ได้ ซึ่งจะปรากฏหน้าต่างดังแสดงดังในรูปที่ ค.23



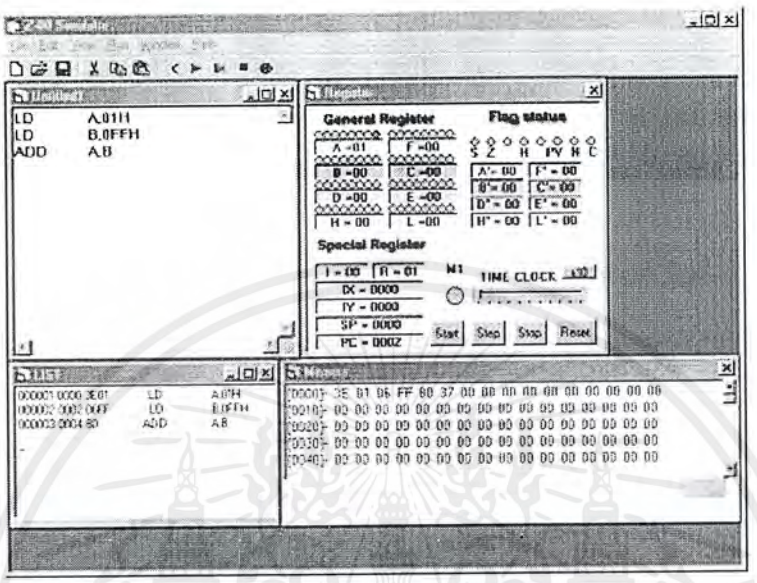

รูปที่ ค.23 การ Assembler

ซึ่งในการ Assembler จะได้ผลแสดงในหน้าต่างของหน่วยความจำ ซึ่งจะแสดงออกมาเป็นเลขฐานสิบหก และแสดงที่หน้าต่าง List จะบอกถึงข้อความผิดพลาดต่าง ๆ หรือผลการแปลงคำสั่งในแต่ละบรรทัด ซึ่งจากที่มีการ Assembler แล้ว ถ้าไม่มีข้อผิดพลาดก็จะทำให้สามารถรันโปรแกรมต่อไปได้

## 11. การรันโปรแกรม

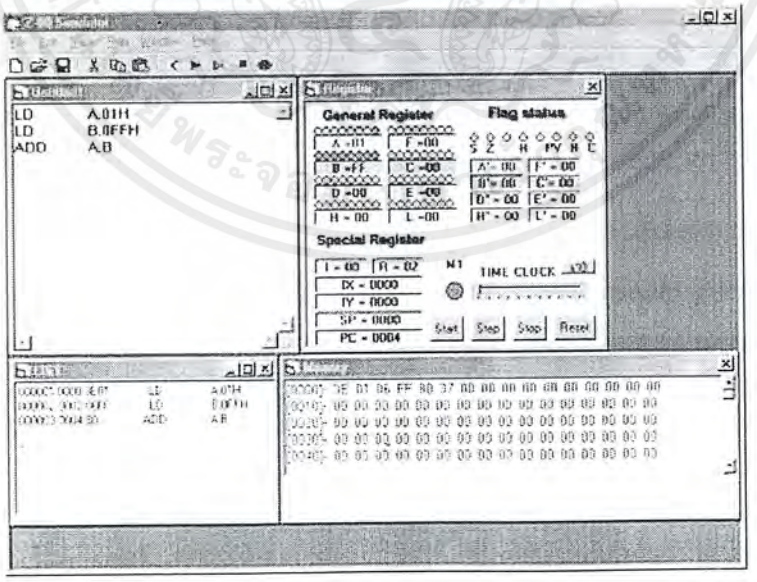
เมื่อทำการ Assemble เรียบร้อยแล้ว ก็จะเข้าสู่ขั้นตอนในการรันโปรแกรม เพื่อดูผลการทำงาน ซึ่งการรันโปรแกรมนั้นสามารถทำได้ตามขั้นตอนดังต่อไปนี้

1) เลือกไปที่เมนู Run เลือก Start จะปรากฏจอภาพดังแสดงในรูปที่ ค.24 ซึ่งในที่นี้เป็นการป้อนโปรแกรมดังในรูปที่ ค.21 ดังนั้นมีผลการรันดังในรูปที่ ค.24



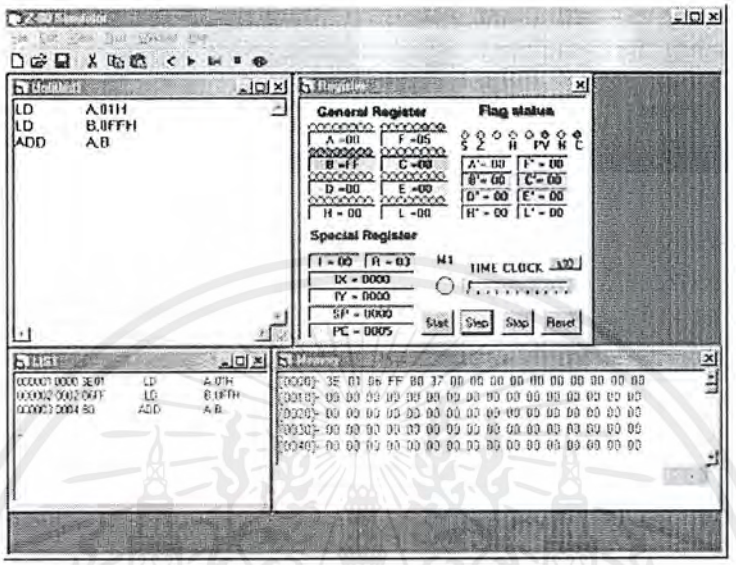

**รูปที่ ค.24** ผลการรันโปรแกรมเมื่อเลือก Start

2) หากทำการเลือก Start อีกครั้งก็จะมีผลการรันดังแสดงในรูปที่ ค.25




รูปที่ ค.25 ผลการรันโปรแกรม

3) หากต้องการที่จะการรันโปรแกรมต่อ และเลือกการรันแบบ Step จะมีผลการรันดังแสดงในรูปที่ ค.26




**รูปที่ ค.26** ผลการรันแบบ Step

4) เมื่อต้องการที่จะจบการรัน ไปที่เมนู Run เลือก Stop หรือจะเลือกจากทูลบาร์ หรือจากหน้าต่างรีจิสเตอร์ก็ได้ ก็จะหยุดการทำงาน

5) หากต้องการที่จะหยุดการรัน แล้วรีเซตโปรแกรมให้ก็ให้เลือกการ Reset ก็จะไปหยุดการทำการรัน แล้วเคลียร์ค่าต่างในหน้าต่างรีจิสเตอร์

# ภาคผนวก ง
# วิธีสร้างหน้าต่างวิธีใช้

# วิธีการสร้างหน้าต่างวิธีใช้ในโปรแกรมจำลองการทำงานไมโครโปรเซสเซอร์ Z-80

## 1.การออกแบบ

วิธีใช้ หรือ Help ในโปรแกรมจำลองการทำงานไมโครโปรเซสเซอร์ Z-80 นั้นได้ออกแบบเป็นการใช้ฟอร์มของวิชวลเบสิก เรียกเปิดเท็กซ์ไฟล์ ซึ่งได้เขียนข้อมูลไว้แล้วในไฟล์ชื่อ helpzz.txt และ helpz801.txt เป็นข้อมูลเกี่ยวกับ ความรู้เบื้องต้นเกี่ยวกับโปรแกรมและวิธีการใช้โปรแกรมเบื้องต้น

### 1.1 การออกแบบฟอร์มวิธีใช้

ในการออกแบบฟอร์มวิธีใช้นั้นออกแบบโดยใช้ คอนโทรลอยู่ 2 อย่างด้วยกันคือ

1) เท็กซ์บ็อก

2) คอมมานด์บ็อตทอม

ซึ่งลักษณะการวางคอลโทรลทั้ง 3 ตัวแสดงได้ดังรูปที่ ง.1


รูปที่ ง.1 การวางคอนโทรลลงบนฟอร์มวิธีใช้

ลักษณะการใช้งานเท็กซ์บ็อก นั้นเราต้องคำนึงถึงทรัพยากรของระบบด้วย เพราะเท็กซ์บ็อกนั้นใช้ทรัพยากรของวินโดวส์ สูงพอสมควร ยิ่งถ้าใช้เท็กซ์บ็อก ขนาดใหญ่ยิ่งต้องใช้ทรัพยากรมาก และมันจะทำให้โปรแกรมเราช้าลง ดังนั้นข้อมูลในเท็กซ์บ็อกจึงไม่ควรมากเกินไป จึงได้แบ่งข้อมูลสำหรับใช้เป็นไฟล์ช่วยเหลือเป็น 2 ชุดคือ ข้อมูลทั่วไปเกี่ยวกับโปรแกรมและข้อมูลวิธีการใช้งานโปรแกรม โดยแยกออกเป็นไฟล์ 2 ไฟล์อย่างชัดเจนเพื่อที่ เท็กซบ็อกจะไม่ได้ใช้ทรัพยากรของวินโดวส์มากไป

**1.2 ข้อมูลในไฟล์ helpzz.txt**

ข้อมูลในไฟล์ helpzz.txt นั้นเป็นข้อมูลวิธีการใช้งานเบื้องต้นของโปรแกรมจำลองการทำงานไมโครโปรเซสเซอร์ Z-80 ซึ่งเป็นประโยชน์สำหรับช่วยเหลือเบื้องต้นสำหรับผู้ไม่เคยใช้งานโปรแกรมจำลองการทำงานไมโครโปรเซสเซอร์ Z-80 มาก่อนในการใช้งานโปรแกรมนี้

**1.3 ข้อมูลในไฟล์ helpz801.txt**

ข้อมูลในไฟล์ helpz801.txt นั้นเป็นข้อมูลทั่วไปเกี่ยวกับโปรแกรมจำลองการทำงานไมโครโปรเซสเซอร์ Z-80 ซึ่งเป็นประโยชน์สำหรับช่วยเหลือเบื้องต้นสำหรับผู้ไม่เคยใช้งานโปรแกรมจำลองการทำงานไมโครโปรเซสเซอร์ Z-80 มาก่อน เพื่อให้ทราบรายละเอียดความเป็นมาและวัตถุประสงค์ตลอดจนประโยชน์ของโปรแกรมจำลองการทำงานไมโครโปรเซสเซอร์ Z-80

## 2. วิธีการสร้างฟอร์มวิธีใช้

วิธีการสร้างฟอร์ม วิธีใช้นั้น ทำได้โดย

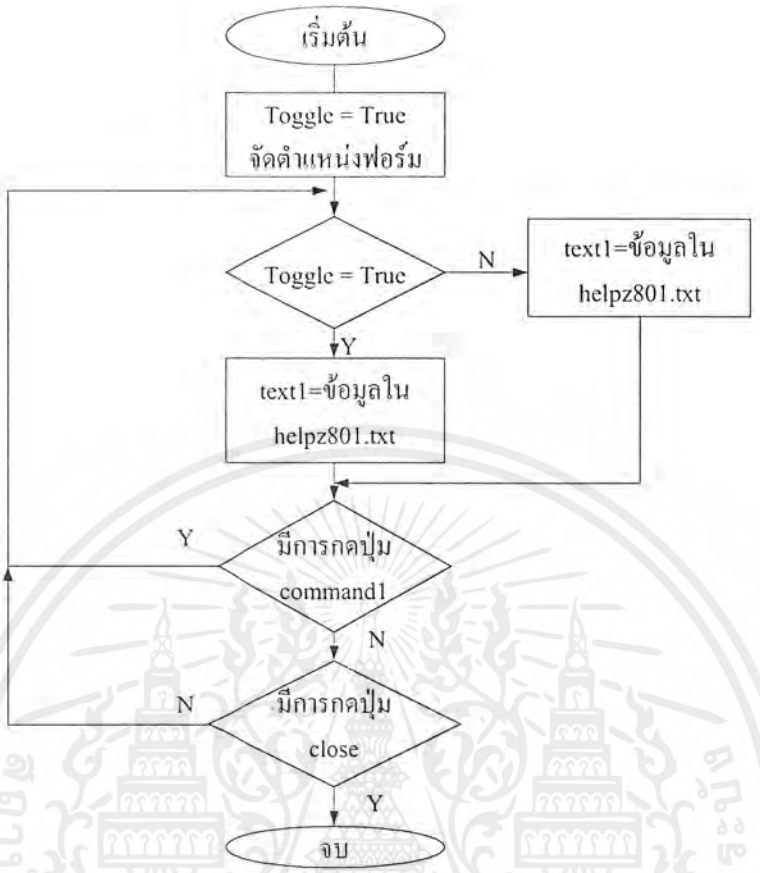
1) สร้างฟอร์มใหม่ และวางคอนโทรลเท็กซ์บ็อก และ คอมมานด์บ็อตทอม ดังรูปที่ ง.1

2) สร้างไฟล์ helpz810.txt และ helpzz.txt

3) ทำการเขียนโปรแกรมที่ฟอร์ม วิธีช่วยเหลือ เพื่อจัดหน้าต่างให้สวยงาม และ เรียกข้อมูลในไฟล์ helpz801.txt หรือ helpzz.txt มาแสดงในเท็กซ์บ็อก เมื่อฟอร์มถูกโหลด

**2.1 วิธีการเขียนโปรแกรมที่ฟอร์มวิธีใช้**

วิธีการเขียนโปรแกรมที่ฟอร์มช่วยเหลือนั้นสามารถอธิบายได้ดังผังงานในรูปที่ ง.2




รูปที่ ง.2 ผังการทำงานของฟอร์มวิธีใช้

โปรแกรมที่เขียนในการสร้างฟอร์มวิธีใช้นั้นแสดงได้ดังรูปที่ ง.3

```
Private Sub Form_Load()
toggle = True
 Call Display
Me.Move 1441 * 3.5, 0, 1441 * 4, 1441 * 4.7
Command1.Move 1441 * 1.2, 1441 * 4.1, Command1.Width, Command1.Height
End Sub

Private Sub Form_Resize()
helpfrm.Text1.Move 5, 5, helpfrm.Width - 100, helpfrm.Height
End Sub
```

รูปที่ ง.3 โปรแกรมที่ใช้แสดงฟอร์มวิธีใช้

```
Private Sub Display()
   Dim buf As String
   help = ""
      If toggle = False Then
      FileName = "C:\Program Files\Z_80sim\helpzz.txt"
      helpfrm.Caption = "วิธีใช้ เกี่ยวกับการใช้งานโปรแกรม"
      toggle = True
       Else
      FileName = "C:\Program Files\Z_80sim\helpz801.txt "
       helpfrm.Caption = "วิธีใช้ ทั่วไปเกี่ยวกับโปรแกรม"
      toggle = False
       End If
       filenum = FreeFile
      Open FileName For Input As filenum ' เปิดไฟล์
        Do While Not EOF(filenum)                    ' อ่านไฟล์ที่ละ 14 บรรทัด
        Line Input #filenum, buf ' อ่านข้อมูลเป็นบรรทัด
        buf = buf & Chr(13) & Chr(10)
                help = help & buf
         Loop
        Close #filenum%
      Text1.Text = help

End Sub
```

**รูปที่ ง.3(ต่อ)** โปรแกรมที่ใช้แสดงฟอร์มวิธีใช้

หน้าตาของหน้าต่างวิธีใช้เมื่อเปิดมาครั้งแรกแสดงได้ในรูปที่ ง.4


**รูปที่ ง.4** หน้าต่างวิธีใช้เมื่อแสดงไฟล์ helpz801.txt

หน้าตาของหน้าต่างวิธีใช้เมื่อทำการคลิกที่ปุ่ม "เกี่ยวกับการใช้งานโปรแกรม"แสดงดังรูปที่ ง.5


**รูปที่ ง.5** หน้าต่างวิธีใช้เมื่อแสดงไฟล์ helpzz.txt

# บรรณานุกรม

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

บริษัท อีทีที จำกัด. **"คู่มือการทดลอง ET-HARDWARE LAB EXPERMENT FOR ET-BOARD VERSION3.0,3.5".** กรุงเทพฯ : อีทีที จำกัด,2538.

บริษัท อีทีที จำกัด. **"คู่มือการทดลอง ET-SOFTWARE LAB EXPERMENT FOR ET-BOARD VERSION3.0,3.5".** กรุงเทพฯ : อีทีที จำกัด,2538.

ยืน ภู่วรรณ. **"ทฤษฎีและการประยุกต์ไมโครโปรเซสเซอร์ Z-80".** กรุงเทพฯ : ซีเอ็ดยูเคชั่น จำกัด, 2521.

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 | นายณรงค์ สังขมาลัย |
| วัน เดือน ปีเกิด | 26 พฤศจิกายน พ.ศ. 2521 |
| สถานที่เกิด | จังหวัด กรุงเทพมหานครฯ |
| ภูมิลำเนาเดิม | 208/8 หมู่ 1 ตำบลหมากแข้ง |
| | อำเภอเมือง จังหวัด อุดรธานี 41000 |
| ที่อยู่ปัจจุบัน | 208/8 หมู่ 1 ตำบล หมากแข้ง |
| | อำเภอ เมือง จังหวัด อุดรธานี 41000 |
| โทรศัพท์ | (042) 321-939 |

**ประวัติการศึกษา**

| | |
|---|---|
| ประถมศึกษา | โรงเรียนเทศบาล 7 |
| มัธยมศึกษาตอนต้น | โรงเรียนอุดรพิทยานุกูล |
| ประกาศนียบัตรวิชาชีพ (ปวช.) | วิทยาลัยเทคนิคอุดรธานี |
| ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.) | วิทยาลัยเทคนิคอุดรธานี |
| ปริญญาตรี | สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ |
| | ภาควิชาครุศาสตร์วิศวกรรม |
| | คณะครุศาสตร์อุตสาหกรรม |
| ผลงานที่ได้รับรางวัล | รางวัลชนะเลิศโครงงานวิทยาศาสตร์ |
| | ระดับภาคตะวันออกเฉียงเหนือ ปี พ.ศ. 2538 |
| ทุนการศึกษา | - |
| คติพจน์ | ทำดีไว้ ไม่เสียหลาย |

# ประวัติผู้แต่ง

| | |
|---|---|
| ชื่อผู้ทำปริญญานิพนธ์ | นายณัฐพงศ์ ชิดพรมราช |
| วัน เดือน ปีเกิด | 21 สิงหาคม พ.ศ. 2521 |
| สถานที่เกิด | จังหวัด กรุงเทพมหานครฯ |
| ภูมิลำเนาเดิม | 33/882 ถนนงามวงศ์วาน<br>แขวงลาดยาว เขตจตุจักร<br>จังหวัด กรุงเทพมหานครฯ 10900 |
| ที่อยู่ปัจจุบัน | 33/882 ถนนงามวงศ์วาน<br>แขวงลาดยาว เขตจตุจักร<br>จังหวัด กรุงเทพมหานครฯ 10900 |
| โทรศัพท์ | (02) 953-3625, (01) 696-8770 |

**ประวัติการศึกษา**

| | |
|---|---|
| ประถมศึกษา | โรงเรียนรัตนโกสินทร์สมโภช |
| มัธยมศึกษาตอนต้น | โรงเรียนราชานิตบางเขน |
| ประกาศนียบัตรวิชาชีพ (ปวช.) | - |
| ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.) | สถาบันเทคโนโลยีราชมงคล<br>วิทยาเขตนนทบุรี |
| ปริญญาตรี | สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์<br>ภาควิชาครุศาสตร์วิศวกรรม<br>คณะครุศาสตร์อุตสาหกรรม |
| ผลงานที่ได้รับรางวัล | - |
| ทุนการศึกษา | - |
| คติพจน์ | ปล่อยวาง ไม่ยึดติด |

# ประวัติผู้แต่ง

| | |
|---|---|
| ชื่อผู้ทำปริญญานิพนธ์ | นางสาวนิตยา แผ่นใหญ่ |
| วัน เดือน ปีเกิด | 2 เมษายน พ.ศ. 2521 |
| สถานที่เกิด | จังหวัด นครราชสีมา |
| ภูมิลำเนาเดิม | 764 ถนนราชนิกูล |
| | ตำบลในเมือง อำเภอเมือง |
| | จังหวัด นครราชสีมา 30000 |
| ที่อยู่ปัจจุบัน | 11/9 หมู่ที่ 3 ซอยวัดพุทธบูชา |
| | ถนนพุทธบูชา แขวงบางมด เขตทุ่งครุ |
| | จังหวัด กรุงเทพมหานครฯ 10140 |
| โทรศัพท์ | (02) 870-6376 |

**ประวัติการศึกษา**

| | |
|---|---|
| ประถมศึกษา | โรงเรียนบ้านหัวทะเล |
| มัธยมศึกษาตอนต้น | โรงเรียนสุรนารีวิทยา |
| ประกาศนียบัตรวิชาชีพ (ปวช.) | - |
| ประกาศนียบัตรวิชาชีพชั้นสูง (ปวส.) | สถาบันเทคโนโลยีราชมงคล |
| | วิทยาเขตภาคตะวันออกเฉียงเหนือ |
| ปริญญาตรี | สาขาวิชาอิเล็กทรอนิกส์และคอมพิวเตอร์ |
| | ภาควิชาครุศาสตร์วิศวกรรม |
| | คณะครุศาสตร์อุตสาหกรรม |
| ผลงานที่ได้รับรางวัล | - |
| ทุนการศึกษา | - |
| คติพจน์ | ชีวิต คือการค้นหา รางวัล คือการค้นพบ |