ปีการศึกษา 2532

Z-80 EMULATOR

โดย

1. นาย ทอม ตันติสฤษดิ์ชัย 291068
2. นาย พรชัย พฤกษารัตนนนท์ 291142
3. นาย พิมล สว่างสมุทร 291148
4. นาย มานัส ปัญญาดิลก 291156

อาจารย์ที่ปรึกษา

1. ศ.ดร. ไพรัช ธัชยพงษ์
2. อ. วิชา ศรีปัญญาพงศ์

ปริญญานิพนธ์ปีการศึกษา 2532

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง Z80 EMULATOR

ผู้จัดทำ

1. นาย ทอม ตันติสฤษดิ์ชัย 291068
2. นาย พรชัย พฤกษารัตนนนท์ 291142
3. นาย พิมล สว่างสมุทร 291148
4. นาย มานัส ปัญญาดิลก 291156

..... อาจารย์ที่ปรึกษา

(ศ.ดร. ไพรัช ธัชยพงษ์)

..... อาจารย์ที่ปรึกษา

(อ. วิชา ศรีปัญญาพงศ์)

อีมูเลเตอร์สำหรับ Z-80

ทอม ตันติสฤษดิ์ชัย
พรชัย พฤกษารัตนนนท์
พิมล สว่างสมุทร
มานัส ปัญญาดิลก
ศ.ดร. ไพรัช ธัชยพงษ์ อาจารย์ที่ปรึกษา
อ. วิชา ศรีปัญญาพงศ์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2532

บทคัดย่อ

ปริญญานิพนธ์นี้กล่าวถึง การออกแบบและสร้างระบบต้นแบบของอีมูเลเตอร์สำหรับไมโครโพรเซสเซอร์ Z-80 โดยได้ประยุกต์เอา IBM PC/XT เป็น เทอร์มินัล ในการติดต่อกับผู้ใช้ โดยที่ วงจรภายนอก (Target System) สามารถทำงานแบบเวลาจริง (Real Time) ที่สัญญาณนาฬิกา 3.57166 MHz มีหน่วยความจำร่วม (Share Memory) ขนาด 48 กิโลไบท์พร้อมทั้งระบบซอฟแวร์ซึ่งประกอบไปด้วย อีดิเตอร์,แอสเซมเบลอร์และดีบักเกอร์ ที่สามารถทำ ซิงเกิลสะเต็ป (Single Step) และ สามารถกำหนดจุดหยุดของโปรแกรมได้

Z-80 EMULATOR

Tom Tantisalidchai
Pornchai Pruksaratananon
Pimon Sawangsamutr
Manat Panyadilok
Professor Pairash Thajchayapong Advisor
Wicha Sripanyapong Advisor
1989

Abstract

This thesis deals with the design and construction of emulator for the Z-80 microprocessor. User interface can be implemented on IBM PC/XT, to be terminal, by which the target systems can run at 3.57166 MHz and there are 48 Kbytes of static rams. Editor, Assembler and Debugger are included in the software system. It can do single step and mark break points.

สารบัญ

ภาคผนวก

สารบัญรูป

บทที่ 1
บทนำ

ในการทดสอบและออกแบบวงจรโดยใช้ไมโครโพรเซสเซอร์เป็นตัวควบคุม เช่น ใช้ซีพียู Z-80 ในการออกแบบวงจร ซึ่งในปัจจุบันต้องใช้ซิงเกิลบอร์ด (SINGLE BOARD) ในการเขียนโปรแกรมเพื่อการทดสอบ โดยในการเขียนโปรแกรมจะต้องป้อนคำสั่งเป็นเลขฐานสิบหก ทำให้เกิดความไม่สะดวกในการตรวจสอบและหาจุดบกพร่องหรือความผิดพลาดต่าง ๆ จึงเกิดแนวคิดที่ว่าจะนำเอาการเขียนโปรแกรม และการทดสอบมาทำบนเครื่องไมโครคอมพิวเตอร์ โดยการต่ออุปกรณ์เพิ่มภายนอก แล้วนำสัญญาณไปใช้ในการต่อวงจร โดยหลักการนี้ทำให้ผู้ใช้สามารถเขียนโปรแกรมและตรวจสอบการทำงานได้สะดวก ในที่นี้จะใช้ซีพียู Z-80 เนื่องจากเป็นเบอร์ที่ใช้กันอย่างกว้างขวาง การเขียนและแก้ไขโปรแกรมจะทำบนเครื่องไมโครคอมพิวเตอร์ มีการต่อสัญญาณจากขาของ Z-80 ไปใช้ในการทดสอบหรือออกแบบวงจร

ปริญญานิพนธ์นี้จะได้กล่าวถึงหลักการในการสร้างการทำงานเลียนแบบซีพียู Z-80 และซอฟต์แวร์สนับสนุนต่าง ๆ โดยแต่ละบทแบ่งเนื้อหาได้ดังนี้ ในบทที่ 2 ได้กล่าวถึงความเป็นมาและความหมายของ อีมูเลเตอร์ และทฤษฎีของระบบซอฟต์แวร์ บทที่ 3 กล่าวถึงภาษาชุดคำสั่ง และตัวแปลภาษา (Translators) บทที่ 4 กล่าวถึงระบบ Z-80 EMULATOR และ สถาปัตยกรรม บทที่ 5 กล่าวถึงแอสเซมเบลอร์ และ ดีบักเกอร์ ของโครงงานนี้ บทที่ 6 เป็นรายละเอียดของการทดลองที่พูดถึง ขั้นตอนในการพัฒนาวิทยานิพนธ์นี้ บทที่ 7 เป็นบทสรุปของปริญญานิพนธ์นี้ รวมไปถึงข้อจำกัดของปริญญานิพนธ์นี้ สำหรับภาคผนวกได้รวบรวมคุณสมบัติของซีพียู Z-80 ตลอดจนโปรแกรมที่เขียนขึ้น

1.1 วัตถุประสงค์

การทำปริญญานิพนธ์นี้มีวัตถุประสงค์ที่สำคัญ 3 ประการด้วยกัน คือ ประการแรกเพื่ออำนวยความสะดวกในการที่จะศึกษา โครงสร้าง การทำงานและการเขียนโปรแกรมของไมโครโพรเซสเซอร์เบอร์ Z-80 ซึ่งในปัจจุบันการศึกษาหรือการเรียนการสอนเกี่ยวกับโครงสร้างการทำงานของ ระบบไมโครโพรเซสเซอร์มักใช้ ไมโครโพรเซสเซอร์เบอร์ Z-80 เป็นตัวอย่าง ทั้งนี้อันเนื่องมาจากโครงสร้างของ Z-80 ง่ายต่อการศึกษาอีกทั้งยังมีเอกสารหรือหนังสือที่มากพอสำหรับใช้ในการค้นคว้า โดยการศึกษานั้นมักจะใช้ซิงเกิลบอร์ด (Single Board) เป็นเครื่องมือในการศึกษาและความยุ่งยากที่ประสบบ่อยมากก็คือ การเขียนโปรแกรมในซิงเกิลบอร์ดดังกล่าว ผู้ใช้จะต้องทำการแปลภาษา แอสเซมบลี

(Assembly) ให้เป็นรหัส (Code) ซึ่งเป็นเลขฐานสิบหกเสียก่อนแล้วจึงทำการป้อนหรือคีย์ (Key) ลงในชิงเกิลบอร์ด แต่ปริญญานิพนธ์นี้เอื้ออำนวยต่อผู้ใช้ในการศึกษาหรือพัฒนาระบบไมโครโพรเซสเซอร์เบอร์ Z-80 บนเครื่อง IBM PC/XT

ประการที่สองเพื่ออำนวยความสะดวก และร่นระยะเวลาของนักดิจิตอลอีเลคทรอนิคส์ในการพัฒนาระบบไมโครโพรเซสเซอร์ ที่ใช้ Z-80 เป็นซีพียู (CPU) และประการสุดท้ายเพื่อศึกษาและพัฒนาระบบต้นแบบในการที่จะประยุกต์ใช้กับไมโครโพรเซสเซอร์เบอร์อื่นๆ เช่น ไมโครโพรเซสเซอร์ตระกูล MCS-48 หรือตระกูล MCS-51

1.2 ความเป็นมา

ในการพัฒนาระบบที่ใช้ไมโครโปรเซสเซอร์เป็นส่วนที่คอยควบคุม (Micro-Processor Based System) อาจพอแบ่งได้เป็น 2 ส่วน คือ การพัฒนาวงจรหรือฮาร์ดแวร์ (Hardware) และการพัฒนาโปรแกรมหรือซอฟแวร์ ในยุคเริ่มต้นการพัฒนาวงจรและการพัฒนาโปรแกรมต้องดำเนินการแยกจากกัน ทำให้ต้องเสียเวลาดำเนินการมากต่อมาได้นำเอาระบบพัฒนาไมโครโปรเซสเซอร์ (Microprocessor Development System) มาช่วย ทำให้การพัฒนาวงจรและการพัฒนาโปรแกรมสามารถกระทำการร่วมกันได้ โดยเฉพาะอย่างยิ่งส่วนที่เกี่ยวกับการหาความผิดพลาดของวงจรและการหาความผิดพลาดของโปรแกรมหรือดีบัก (Debug)

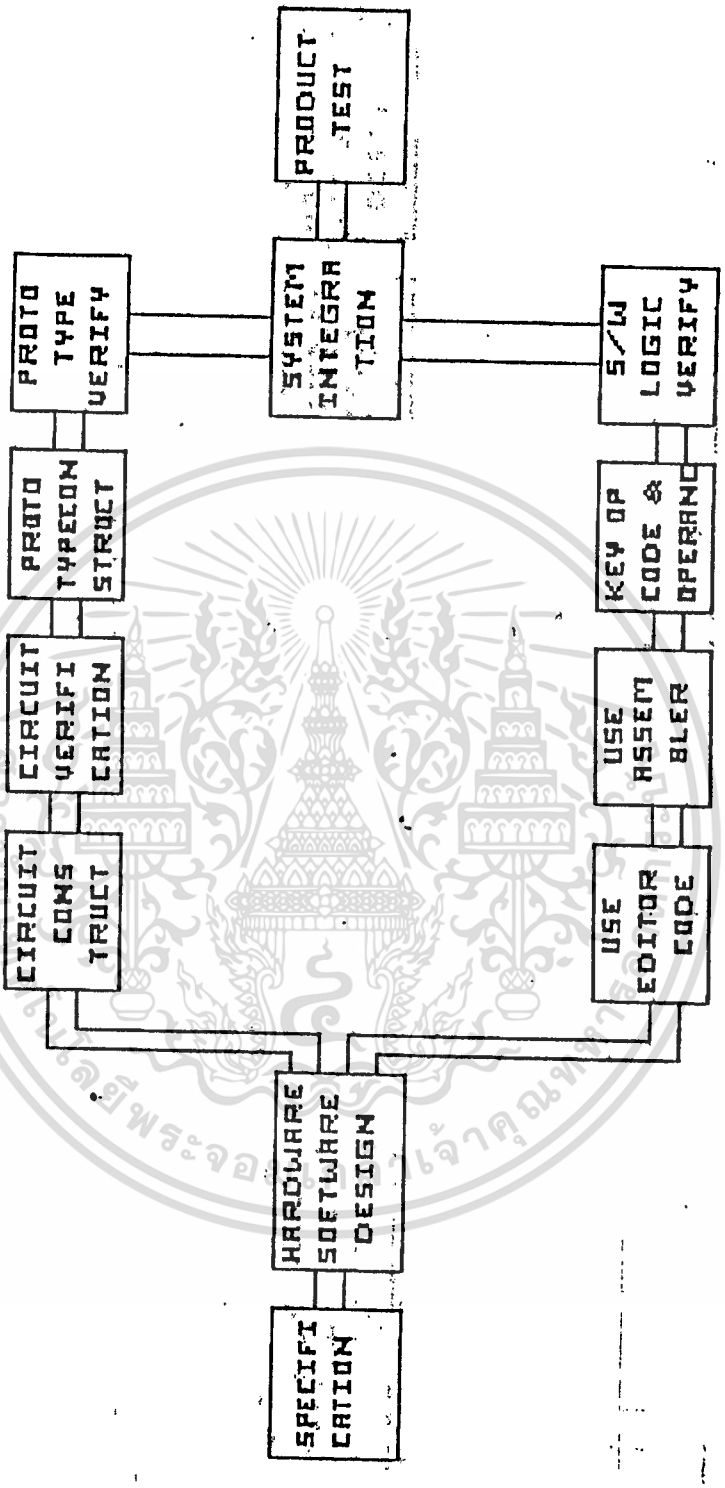
ระบบพัฒนาระบบไมโครโพรเซสเซอร์ที่ใช้กันอย่างแพร่หลายในปัจจุบัน อาจพอแบ่งได้เป็น 2 ประเภทคือ การใช้ In-Circuit Emulator (ICE) และ ชิงเกิลบอร์ด ระบบพัฒนาระบบไมโครโพรเซสเซอร์ประเภทแรกมีขีดความสามารถที่สูงมาก ทั้งทางด้านการพัฒนาวงจรและการพัฒนาโปรแกรม คือสามารถทำการดีบัก (Debug) และทดสอบวงจรที่ได้พัฒนาทุกส่วนโดยให้ผู้ใช้สามารถเขียนโปรแกรมที่แอสเดรสใดๆ ได้อย่างอิสระ ความสามารถในการป้องกันการเกิดสัญญาณรบกวนจากวงจรทดสอบ (Target System) เช่นในกรณีที่บัส (Bus) ข้อมูลที่ออกแบบไว้ต่อผิดและสามารถทำให้ ICE แฮงก์ (Hang) ได้ แต่อย่างไรก็ตามถึงแม้ว่าข้อดีและความสามารถของ ICE จะมีมากก็ตามแต่ราคานั้นสูงมากจึงมีผู้นำมาใช้ไม่มากนัก

ส่วนประเภทที่สองซึ่งมีราคาต่ำและมีขายกันอย่างแพร่หลาย จึงนิยมใช้กันมาก แต่ขีดความสามารถจำกัด เช่น ความสามารถของซีพียู และหน่วยความจำของระบบที่ทดสอบหรือระบบที่กำลังพัฒนา ถูกนำมาใช้ได้ไม่เต็มที่ การหาความผิดพลาดของโปรแกรมโดยเฉพาะในส่วนที่เกี่ยวกับการตั้งจุดหยุดอย่างมีเงื่อนไข (Conditional Breakpoint) และการตามรอยแบบเวลาจริง (Real Time Trace) ซึ่งส่วนใหญ่แล้วมักจะใช้ได้เฉพาะ

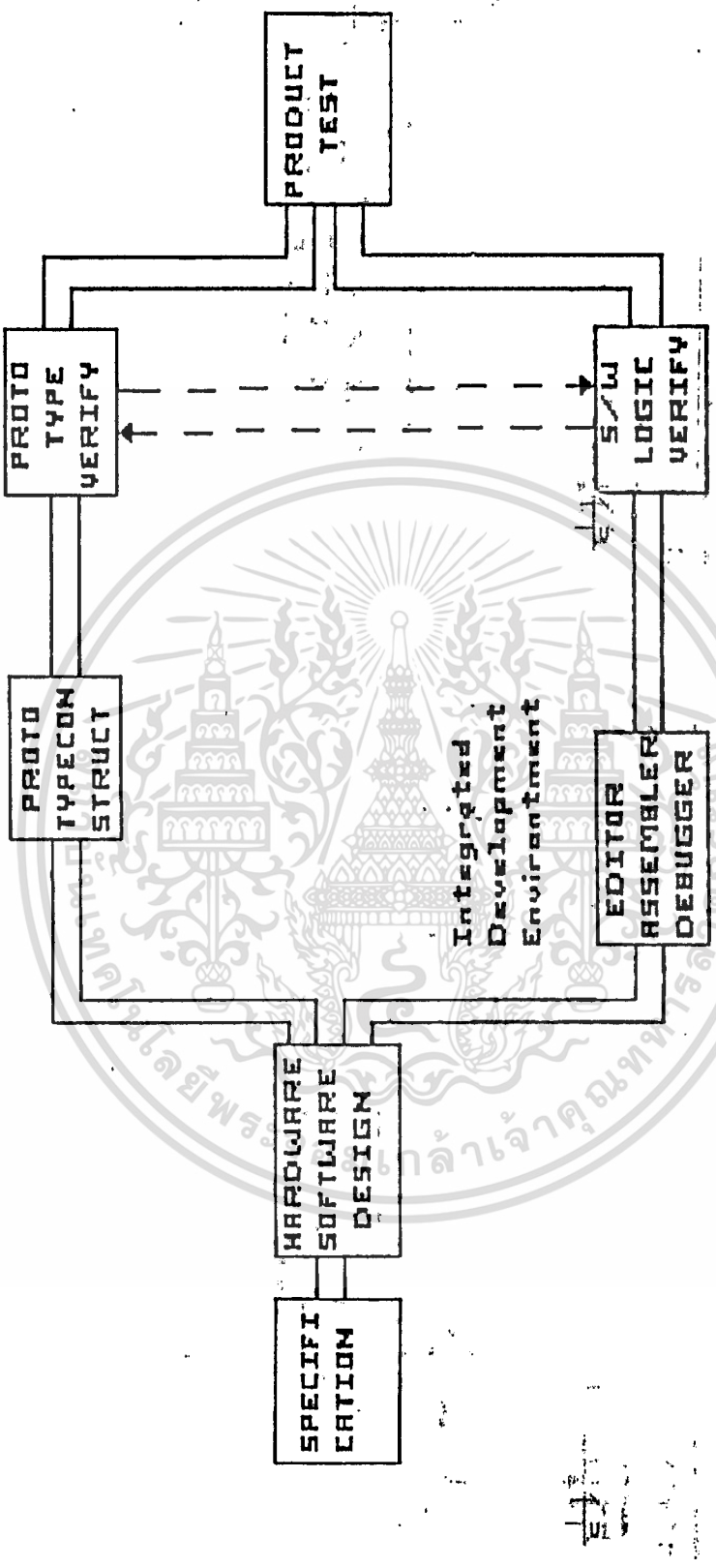
การดำเนินการทีละขั้น (Single Step) นอกจากนี้ยังมีขีดจำกัดในการหาความผิดพลาดของวงจร อีกทั้งยังมีความล่าช้าที่เกิดขึ้นเสมอในการใช้ชิงเกิลบอร์ดส่วนใหญ่ คือ การที่นักดิจิตอลอิเลคทรอนิคส์ต้องการพัฒนาโปรแกรมเพื่อทดสอบระบบ จะต้องทำการป้อนคำสั่งรหัสดำเนินการ (Opcode) ที่เป็นเลขฐานสิบหกลงในชิงเกิลบอร์ด ทำให้เกิดความไม่สะดวกและเกิดความล่าช้าเพราะจะต้องเสียเวลาให้กับกระบวนการดังกล่าว โดยเฉพาะถ้าโปรแกรมที่จะใช้ในการทดสอบระบบมีขนาดใหญ่มาก อีกทั้งยังทำให้เกิดความผิดพลาดอันเนื่องมาจากตัวของผู้ป้อนข้อมูลเอง หรือความยุ่งยากที่จะต้องทำการแบคอัพแรมไว้เสมอ ปัญหาดังกล่าวเหล่านี้จะแสดงดังรูปที่ 1.1

จากรูปที่ 1.1 เป็นการออกแบบระบบโดยทั่ว ๆ ไป ซึ่งเริ่มจากการหาข้อระบุเฉพาะหรือขอบเขตของระบบงาน (Specification) แล้วจึงเริ่มทำการออกแบบระบบฮาร์ดแวร์และซอฟท์แวร์ (Hardware Software Design) จากนั้นการพัฒนาระบบจะแยกกันทำไปพร้อม ๆ กันได้ คือทางด้านฮาร์ดแวร์ จากการออกแบบวงจรในแต่ละส่วนแล้วทำการสร้างวงจรขึ้นมา (Circuit Construction) ตัวอย่างวงจรแต่ละส่วนอย่างเช่น การออกแบบและสร้างในส่วนของหน่วยความจำ ซึ่งจะแยกจากส่วนของอินพุท เอาท์พุทพอร์ท และส่วนของการควบคุม จากนั้นขั้นต่อไปก็คือ การทดสอบในแต่ละส่วนว่ามีความถูกต้องตรงตามความต้องการมากน้อยเพียงใด (Circuit Verification) เมื่อทดสอบว่าส่วนต่าง ๆ ถูกต้องแล้ว ขั้นต่อไปก็จะนำแต่ละส่วนต่าง ๆ มารวมเป็นระบบต้นแบบ (Prototype Construction) แล้วทำการตรวจสอบ (Prototype Verification) ในขณะเดียวกันถ้ามองด้านซอฟท์แวร์ ผู้พัฒนาระบบก็จะทำการพัฒนาโปรแกรมที่จะใช้ระบบฮาร์ดแวร์ โดยใช้อิดิเตอร์ทำการเขียนโปรแกรม แล้วจะใช้ตัวแปรที่อยู่คนละระบบทำการแปล (Assembler) ให้เป็นรหัสดำเนินการ อย่างเช่น ในปัจจุบันมักจะพิมพ์รหัสดำเนินการแล้วนำมาป้อนลงชิงเกิ้ลบอร์ดด้วยมือ หรือถ้าจะให้รวดเร็วขึ้นอีกก็จะต้องทำการดาวน์โหลด (Download) ด้วยวิธีการส่งผ่านทางซีเรียลพอร์ท (Serial Port) หรือทำเป็นลักษณะใช้แรมสองทาง ซึ่งจำเป็นจะต้องระบบฮาร์ดแวร์เพิ่มเติมขึ้นมา เมื่อถึงขั้นตอนนี้ระบบต้นแบบของวงจร และซอฟท์แวร์จะต้องนำมารวมให้เป็นระบบรวมก่อน (System Integration) ที่จะทำการทดสอบขั้นสุดท้ายที่จะเป็นผลิตภัณฑ์ต่อไป (Product Test)

จากขั้นตอนทั้งหมดในรูป 1.1 จะสามารถลดขั้นตอนดังกล่าวในลักษณะดังแสดงดังรูป 1.2 โดยใช้ Z-80 อีมูเลเตอร์ช่วยในการพัฒนา โดยทางด้านฮาร์ดแวร์ผู้พัฒนาระบบสามารถออกแบบวงจรให้เป็นระบบต้นแบบบนกระดาษก่อนและส่วนทางด้านซอฟท์แวร์การพัฒนาจะทำบนระบบรวม ในระหว่างนี้ผู้พัฒนาสามารถทำการทดสอบระบบต้นแบบพร้อมกับทดสอบโปรแกรมได้อีกด้วยจนได้ผลิตภัณฑ์ออกมา



รูปที่ 1.1 การออกแบบระบบโดยทั่ว ๆ ไป



รูปที่ 1.2 การพัฒนาโดยใช้ Z-80 อีมูเลเตอร์

บทที่ 2

อีมูเลเตอร์ และ ระบบซอฟต์แวร์

(Emulator and Software System)

บทนี้จะกล่างถึงทฤษฎีและความหมายของคำว่า อีมูเลเตอร์ (EMULATOR) ซึ่งเป็นที่ถกเถียงกันอยู่ทั่วไปของบรรดานักดิจิตอลอิเลคทรอนิคส์และนักคอมพิวเตอร์ว่า คำว่า อีมูเลเตอร์ คืออะไรและทั่ว ๆ ไปใช้คำนี้ในทางใดบ้าง โดยนิยามหรือความหมายในบทนี้ได้อ้างอิงจากค้นคว้าวารสารทางวิชาการ ,สิ่งตีพิมพ์ (Paper) และหนังสือเป็นสำคัญ และเหตุใดปริญญานิพนธ์นี้จึงได้นำเอาคำ ๆ นี้มาใช้พร้อมทั้งนำเอาหลักการมาประยุกต์ให้เหมาะกับปัญหาที่พบเสมอในการพัฒนาระบบไมโครโพรเซสเซอร์ นอกจากนี้ยังกล่าวถึงทฤษฎีต่าง ๆ ที่เกี่ยวข้องกับระบบซอฟต์แวร์ที่ใช้ในปริญญานิพนธ์นี้

2.1 ความเป็นมา

ในปัจจุบันนี้ระบบคอมพิวเตอร์จะมีเทคโนโลยีในการสร้าง ระบบพัฒนาการที่ใหม่และเป็นรุ่นต่าง ๆ ออกมาเสมอ อย่างเช่น เมื่อสมัยหนึ่งคอมพิวเตอร์ของยุคนั้นจะมีเครื่องที่เป็นรุ่นของเครื่อง IBM 360 ,เครื่อง IBM 7090 หรือระบบไมโครคอมพิวเตอร์ที่มีบัสข้อมูล 8 บิท (Bit) หรือเป็นไมโครคอมพิวเตอร์ที่เป็นชิงเกิลบอร์ด ที่ใช้ไมโครโพรเซสเซอร์เบอร์ 8080, Z-80C แต่สมัยต่อมาระบบคอมพิวเตอร์ได้พัฒนาต่อจากนั้นมาอีกรุ่นหนึ่ง เช่น เครื่อง IBM 360/65 ,เครื่อง IBM 360/50 หรือแม้กระทั่งไมโครคอมพิวเตอร์ปัจจุบันเป็นเครื่อง 16 บิท, 32 บิท การที่เกิดพัฒนาการเช่นนี้ถึงแม้ว่าจะเป็นการดี แต่ข้อเสียย่อมมีคือ ในกรณีที่ว่าระบบหรือโปรแกรมของคอมพิวเตอร์รุ่นก่อนไม่สามารถใช้กับระบบคอมพิวเตอร์รุ่นใหม่ได้ จึงได้มีการค้นคิดหาวิถีทางในการที่จะแก้ไขปัญหาดังกล่าว เช่น วิธีการนำเอาภาษาระดับสูง (High-level Language) ในการปรับเปลี่ยนระบบ (Transition) หรือจากระบบหนึ่งให้สามารถทำงานในอีกระบบหนึ่งได้ การใช้การแปลโปรแกรมโดยอัตโนมัติ (Automatic Program Translation) จะใช้ได้เฉพาะโครงสร้างและสถาปัตยกรรมมีลักษณะคล้ายกันหรือวิธีการแปลด้วยมือ (Hand Translation) วิธีนี้เป็นวิธีที่ลำบากและเสียเวลาเป็นอย่างมาก จากวิธีการแก้ปัญหาต่างๆ เหล่านี้ ได้มีชื่อเรียกโดยเฉพาะขึ้นมาชื่อหนึ่งคือ อีมูเลชั่น (Emulation)

2.2 ความหมาย

อีมูเลชั่น คือ เครื่องมือหรือวิธีการใดวิธีการหนึ่งที่ใช้ในกระบวนการปรับเปลี่ยนระบบ ซึ่งอีมูเลชั่นนั้นเกิดจากการสร้างหรือขยายบางส่วนเพิ่มเติมในระบบคอมพิวเตอร์ระบบหนึ่งให้สามารถทำงานตามโปรแกรมที่เขียนขึ้นสำหรับระบบคอมพิวเตอร์อื่นได้ และระบบที่ทำได้เช่นนี้เราจึงเรียกว่า อีมูเลเตอร์ (EMULATOR)

อีมูเลเตอร์ยังอาจที่จะมีความหมายเพิ่มเติมว่าเป็นส่วนประกอบต่าง ๆ คือ ฮาร์ดแวร์ ,ไมโครโปรแกรม (Microprograms) และซอฟต์แวร์ ไปเพิ่มให้กับระบบคอมพิวเตอร์ระบบหนึ่ง เพื่อให้มีความสามารถในการที่จะทำงานตามโปรแกรมที่เขียนขึ้นสำหรับให้ระบบคอมพิวเตอร์อีกระบบหนึ่งทำงาน ซึ่งส่วนประกอบเหล่านี้อาจจะมีไม่ครบก็ได้แต่อย่างน้อยที่สุดจะต้องมีการต่อเติม ฮาร์ดแวร์หรือเพิ่มเติมไมโครโปรแกรม ถ้าไม่มีสิ่งหนึ่งสิ่งใดในสองสิ่งนี้เราเรียกว่า การจำลอง (Simulation)

การอีมูเลชันส่วนใหญ่มักกระทำในระดับคอมพิวเตอร์เมนเฟรม เช่น เครื่อง IBM 360/50 ทำการอีมูเลตเครื่อง IBM 360 เครื่อง IBM 360/65 เป็นอีมูเลเตอร์ของ IBM 7090 ,เครื่อง RCA Spectra/70 เป็นอีมูเลเตอร์ของเครื่อง RCA 301 และระบบของปริญญานิพนธ์นี้ เป็นการนำเครื่องไมโครคอมพิวเตอร์ IBM PC/XT ทำการอีมูเลตระบบไมโครโพรเซสเซอร์ คือ ซิงเกิลบอร์ด ซึ่งเรียกว่า Z-80 EMULATOR

In-Circuit Emulator (ICE) ก็เป็นอีมูเลชันอย่างหนึ่ง โดยระบบนี้นำไปใช้ประโยชน์ในด้านลดขั้นตอนในการพัฒนาระบบไมโครโพรเซสเซอร์ลงไปได้เป็นอย่างมาก ทั้งนี้เพราะผู้พัฒนาระบบไม่ต้องตรวจสอบวงจรที่ออกแบบในระดับส่วน ๆ ก่อน โดยผู้พัฒนาระบบจะรวมทุกส่วนเข้าด้วยกันให้เป็นระบบต้นแบบแล้วใช้ ICE เป็นตัวช่วยในการตรวจสอบ ให้เป็นระบบที่สมบูรณ์ต่อไป

2.3 ทฤษฎีของระบบซอฟต์แวร์

ในระบบ Z80-EMULATOR นอกจากจะมีการ์ดอีมูเลเตอร์ ซึ่งเป็นฮาร์ดแวร์ที่เป็นหัวใจสำคัญแล้ว ยังจะต้องประกอบด้วยซอฟต์แวร์หลายโมดูล ประกอบกันเข้าเป็นระบบโปรแกรมเหล่านี้นอกจากจะทำหน้าที่ในการควบคุมการทำงานของการ์ดแล้ว ยังมีส่วนประกอบที่ทำให้ซอฟต์แวร์ทั้งหมดประกอบเป็นระบบรวมที่อำนวยความสะดวกแก่ผู้ใช้ และเพิ่มประสิทธิภาพในการใช้งาน โดยที่ซอฟต์แวร์ทั้งหมดมีการทำงานร่วมกันและถูกรวบรวมเข้าเป็นระบบซอฟต์แวร์รวมเพียงโปรแกรมเดียว ผู้ใช้สามารถทำงานได้โดยไม่ต้องเรียกโปรแกรมย่อยในการทำงานแต่ละอย่าง

ซอฟต์แวร์ ของระบบนี้ประกอบด้วยโมดูลสำคัญ ดังต่อไปนี้

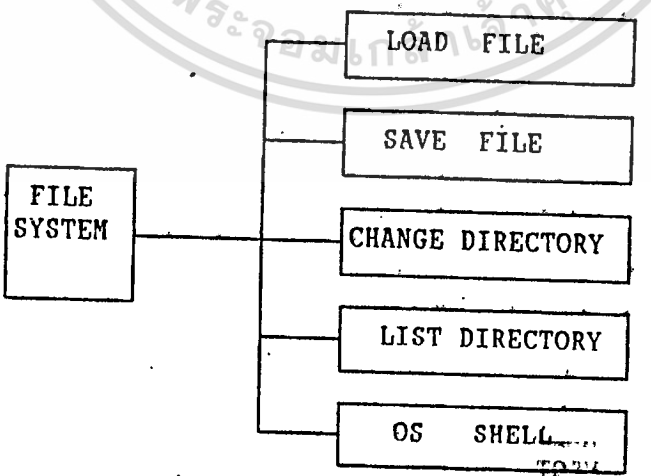
1. ระบบแฟ้มข้อมูล (File System)
2. อีดิเตอร์ (Editor)
3. แอสเซมเบลอร์ (Assembler)
4. ดีบักเกอร์ (Debugger)
5. มอนิเตอร์โปรแกรม (Monitor Program)

2.3.1 ระบบแฟ้มข้อมูล (File System)

เป็นโปรแกรมที่ประกอบด้วยฟังก์ชั่นการทำงานต่าง ๆ ที่ทำหน้าที่ จัดการเกี่ยวกับไฟล์ อาทิเช่น อ่านข้อมูลจากไฟล์ (load file) เก็บข้อมูลลงไฟล์ (save file) ซึ่งไฟล์ข้อมูลที่จะใช้ในระบบนี้มี 2 ประเภท คือ

- แฟ้มข้อมูลต้นฉบับ (Source Code File) โปรแกรมภาษาแอสเซมบลี Z-80 จะเก็บในดิสค์อยู่ในรูปของแฟ้มข้อความ (Text File)
- แฟ้มข้อมูลรหัสภาษาจุดหมาย (Object Code File) เป็นรหัสภาษาจุดหมาย (Object Code) ที่ได้จากการแปลชุดคำสั่งโปรแกรมภาษาแอสเซมบลี ด้วยโปรแกรม แอสเซมเบลอร์

นอกจากระบบไฟล์จะทำหน้าที่จัดการเกี่ยวกับไฟล์ แล้วยังทำงานเกี่ยวกับสารบบ (Directory) เช่น แสดงรายชื่อไฟล์ (listing filenames) ,ย้ายสารบบ (Change Directory) รวมทั้งทำหน้าที่ติดต่อกับระบบปฏิบัติการ ซึ่งฟังก์ชันการทำงานทั้งหมดแสดงได้ดังรูปที่ 2.1



รูป 2.1 แสดงฟังก์การทำงานของระบบไฟล์

2.3.2 อีดิเตอร์ (Editor)

โปรแกรมส่วนนี้ทำหน้าที่ในการสร้าง ,แก้ไข และเพิ่มเติมแฟ้มข้อมูลต้นฉบับ (Source File) ที่เป็นภาษาแอสเซมบลีของ Z-80 ข้อความที่จะทำการสร้างขึ้นใหม่หรือแก้ไขจะอยู่ในพื้นที่ของรหัสต้นฉบับ (Source Code Area) ซึ่งเป็นเนื้อที่ในหน่วยความจำที่ใช้สำหรับการแก้ไขแฟ้มข้อความ โปรแกรมอีดิเตอร์นี้สามารถเคลื่อนย้ายตำแหน่งของตัวชี้ตำแหน่ง (Cursor) ได้ตามแบบบรรณาธิกรเต็มจอภาพ (Full Screen Editor)

2.3.3 แอสเซมเบลอร์ (Assembler)

โปรแกรมส่วนนี้ ทำหน้าที่แปลงรหัสภาษาต้นฉบับให้เป็นรหัสภาษาจุดหมาย โปรแกรมจะอ่านอินพุตที่อยู่ในพื้นที่ของรหัสต้นฉบับมาประมวลผล แล้วให้เอาท์พุตเก็บไว้ในพื้นที่ของรหัสดำเนินการ (Op Code Area) รหัสภาษาจุดหมายที่อยู่ในพื้นที่ของรหัสดำเนินการ จะถูกโปรแกรมดีบักเกอร์เรียกใช้เพื่อทำการดำเนินการทีละขั้น (Single Step) หรือดำเนินงาน (Run) ต่อไป นอกจากแอสเซมเบลอร์จะอ่านเอารหัสภาษาจุดหมายเก็บไว้ในพื้นที่ของรหัสต้นฉบับแล้ว ยังอ่านเอาข้อมูลชุดเดียวกันนี้ไปไว้ในหน่วยความจำที่อยู่บนการ์ดอีกด้วย เพื่อให้ซีพียู Z-80 นำไปกระทำการต่อไป ในระหว่างการแปลภาษา ถ้าหากมีข้อผิดพลาดเกิดขึ้นแล้ว แอสเซมเบลอร์จะทำหน้าที่แจ้งข้อผิดพลาดให้ผู้ใช้รับทราบ และทำการแก้ไขให้ถูกต้องต่อไป

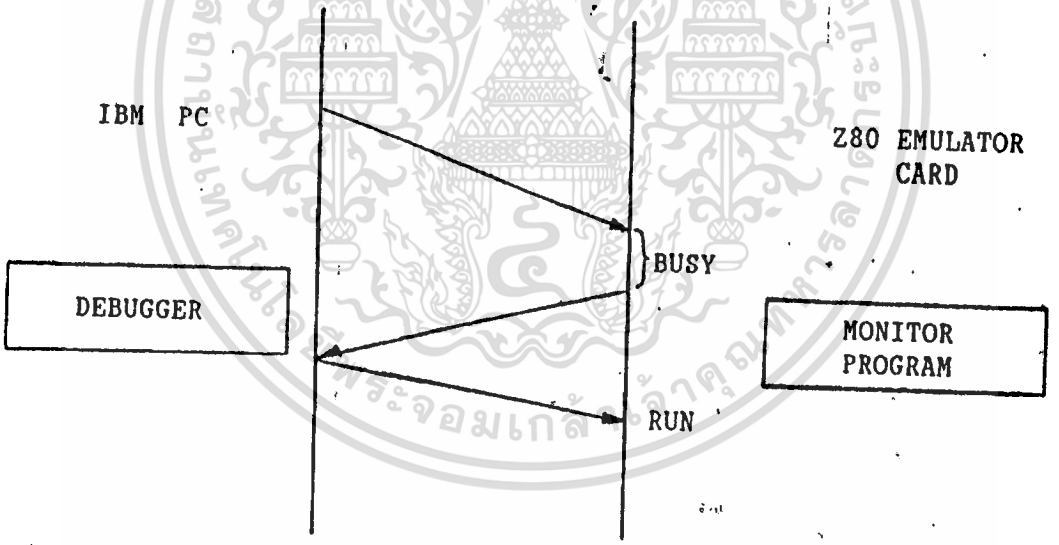
2.3.4 ดีบักเกอร์ (Debugger)

มีหน้าที่ในการกระทำ (Execute) คำสั่งในพื้นที่ของรหัสดำเนินการ และแสดงผลการทำงานเป็นขั้นตอน ซึ่งช่วยอำนวยความสะดวกแก่ผู้ใช้ในการศึกษาและตรวจสอบการทำงานของซีพียู Z-80 ทำให้ผู้ใช้สามารถตรวจสอบการทำงานของโปรแกรมที่ผู้ใช้เขียน

ดีบักเกอร์สามารถกระทำคำสั่งในโปรแกรมตามขั้นตอนที่ต้องการ สามารถตรวจสอบค่าของเรจิสเตอร์ในขณะใดขณะหนึ่ง และสามารถตรวจสอบข้อมูลในหน่วยความจำตำแหน่งต่าง ๆ

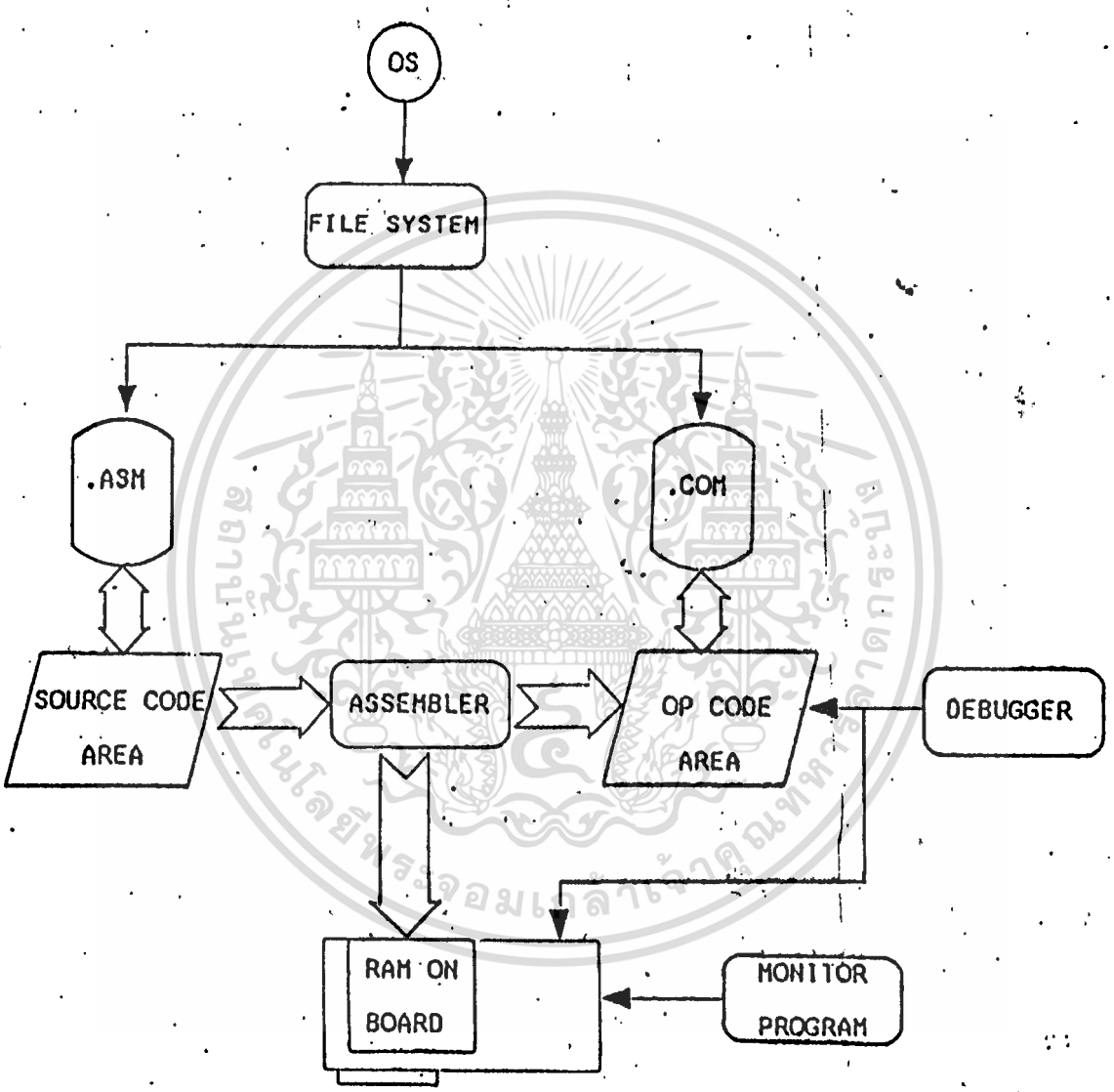
2.3.5 โปรแกรมมอนิเตอร์ (Monitor Program)

เป็นโปรแกรมควบคุมการทำงานของซีพียู Z-80 และควบคุมการติดต่อระหว่างการ์ดอีมูเลเตอร์กับไอบีเอ็มพีซี โปรแกรมมอนิเตอร์จะเก็บไว้ในรอม (ROM) บนการ์ด โปรแกรมมอนิเตอร์จะทำงานร่วมกันกับดีบักเกอร์ในการทำการดำเนินการทีละขั้น (Single Step) และดำเนินงาน (Run) โดยดีบักเกอร์จะส่งรหัสคำสั่งไปยังการ์ดอีมูเลเตอร์ ในขณะเดียวกันที่โปรแกรมมอนิเตอร์กำลังรอรับคำสั่ง เมื่อได้รับคำสั่งโปรแกรมมอนิเตอร์จะส่งสัญญาณตอบรับกลับไป และรอรับสัญญาณตอบรับกลับมาจากดีบักเกอร์อีกครั้งหนึ่ง เมื่อได้รับสัญญาณตอบรับจากดีบักเกอร์แล้วโปรแกรมมอนิเตอร์ จึงจะทำตามรหัสคำสั่งที่ได้รับ หลังจากทำงานตามคำสั่งแล้ว โปรแกรมมอนิเตอร์จะกลับมารอรับคำสั่งจากดีบักเกอร์ต่อไป การติดต่อกันระหว่างเครื่องพีซี กับโปรแกรมมอนิเตอร์แสดงได้ดังรูปที่ 2.2



รูป 2.2 แสดงการติดต่อระหว่างเครื่องพีซีและโปรแกรมมอนิเตอร์

โปรแกรมแต่ละโมดูลจะทำงานสัมพันธ์กันดังแสดงไว้ในรูปที่ 2.3



รูปที่ 2.3 แสดงความสัมพันธ์โมดูลการทำงานต่าง ๆ

ระบบแฟ้มข้อมูลจะทำหน้าที่ติดต่อกับระบบปฏิบัติการ และควบคุมการอ่าน-เขียนแฟ้มข้อมูลที่ใช้ในระบบแฟ้มข้อมูล จะอ่านข้อมูลรหัสต้นฉบับที่เป็นภาษาแอสเซมบลีจากแผ่นดิสค์มาเก็บไว้ในหน่วยความจำบริเวณพื้นที่ของรหัสต้นฉบับ (Source Code Area) ข้อมูลในพื้นที่ของรหัสต้นฉบับจะสามารถทำการแก้ไขเพิ่มเติมได้จากโปรแกรมอีดิเตอร์ โดยผู้ใช้สามารถป้อนโปรแกรมใหม่ได้ หรือแก้ไขโปรแกรมเดิมได้ เมื่อทำการแก้ไขเรียบร้อยแล้วระบบแฟ้มข้อมูล จะทำหน้าที่ในการเก็บข้อมูลลงแผ่นดิสค์เพื่อบันทึกไว้

หลังจากที่ได้ชุดคำสั่งภาษาต้นฉบับ (Source Program) ภาษาแอสเซมบลี Z-80 แล้ว โปรแกรมแอสเซมเบลอร์จะแปลรหัสต้นฉบับให้เป็นรหัสภาษาจุดหมาย ซึ่งก็คือภาษาเครื่องนั่นเอง และเก็บเอาท์พุตที่ได้ไว้ในหน่วยความจำบริเวณพื้นที่ของรหัสดำเนินการ แล้วนำไปถูกบันทึกเก็บไว้ในแผ่นดิสค์ นอกจากจะนำรหัสภาษาจุดหมายที่ได้จากการแปลเก็บไว้ในพื้นที่ของรหัสดำเนินการแล้ว โปรแกรมแอสเซมเบลอร์จะนำรหัสภาษาจุดหมาย นี้เก็บไว้ในหน่วยความจำบนการ์ด (RAM ON BOARD) เพื่อให้ ซีพียู Z-80 ใช้ดำเนินการต่อไป

ข้อมูลในพื้นที่ของรหัสดำเนินการที่เป็นเอาท์พุตจากแอสเซมเบลอร์ จะถูกใช้เป็นอินพุตของโปรแกรมดีบักเกอร์ โปรแกรมดีบักเกอร์จะต้องทำงานควบคู่ไปกับโปรแกรมมอนิเตอร์ ซึ่งอยู่บนการ์ด โดยที่ดีบักเกอร์จะเป็นผู้ส่งคำสั่งไปควบคุมการทำงานของการ์ดและรอรับผล หรือตรวจดูสถานะของเรจิสเตอร์จากโปรแกรมมอนิเตอร์ แล้วทำการแสดงผลให้กับผู้ใช้

ทางฝ่ายโปรแกรมมอนิเตอร์ เมื่อได้รับคำสั่งจากดีบักเกอร์แล้วจะทำงานตามคำสั่งกับข้อมูลในหน่วยความจำบนการ์ด โปรแกรมมอนิเตอร์จะรายงานผลการทำงานหรือสถานะของเรจิสเตอร์ไว้ในหน่วยความจำบนการ์ด ที่เป็นหน่วยความจำร่วมบริเวณหนึ่งที่ดีบักเกอร์สามารถอ่านข้อมูลจากหน่วยความจำบริเวณนี้ได้ แล้วนำไปแสดงผล ต่อจากนั้นโปรแกรมมอนิเตอร์จะกลับไปรอรับคำสั่งจากดีบักเกอร์ต่อไป

บทที่ 3
การแปลภาษา
(Translation)

3.1 ภาษาชุดคำสั่ง (Programming Languages)

การโต้ตอบสื่อสารระหว่างมนุษย์ด้วยกันเองจะเป็นไปอย่างมีประสิทธิภาพได้ โดยใช้ภาษาเป็นตัวกลาง ภาษาเป็นสิ่งที่ช่วยในการแสดงของความคิดที่อยู่ภายในมนุษย์ ซึ่งถ้าปราศจากภาษาแล้ว การสื่อสารกันจะเป็นสิ่งที่ยากลำบากมาก

ในด้านการเขียนชุดคำสั่งสำหรับคอมพิวเตอร์ (Computer Programming) นั้นก็มีภาษาชุดคำสั่ง (Programming Language) ซึ่งเป็นภาษาตัวกลางไว้ใช้ในการสื่อสารระหว่างบุคคลที่มีปัญหาต้องการจะแก้ไขกับเครื่องคอมพิวเตอร์ ซึ่งเป็นอุปกรณ์ไว้ช่วยในการแก้ปัญหานั้น

ภาษาชุดคำสั่งที่มีประสิทธิภาพจะช่วยส่งเสริมการพัฒนา และง่ายที่จะเขียนโปรแกรมเพื่อแก้ปัญหาที่ต้องการ ดังนั้นภาษาสำหรับโปรแกรมจึงเป็นสะพานที่เชื่อมช่องว่างระหว่างความคิดของมนุษย์ที่เป็นไปอย่างไม่มีโครงสร้าง กับการทำงานที่แน่นอนของเครื่องคอมพิวเตอร์

การเขียนโปรแกรมเพื่อแก้ปัญหาใดก็ตามจะเป็นไปอย่างสะดวกและง่ายดาย ถ้าภาษาที่ใช้มีลักษณะใกล้เคียงกับปัญหานั้น

ลำดับของภาษาสำหรับโปรแกรม ซึ่งเรียงตามความไม่ขึ้นอยู่กับเครื่องจากมากไปหาน้อย มีดังต่อไปนี้

1.ภาษาเครื่อง (Machine-Level Languages)

เป็นภาษาที่ต่ำที่สุดของภาษาในคอมพิวเตอร์ แต่ละคำสั่งในโปรแกรมจะถูกแสดงโดยรหัสตัวเลข และการอ้างถึงตำแหน่งต่าง ๆ ในหน่วยความจำก็อยู่ในรูปแบบตัวเลขด้วยเช่นเดียวกัน

2.ภาษาแอสเซมบลี (Assembly Languages)

เป็นภาษาที่แต่ละคำสั่งในภาษาเครื่องจะถูกแทนด้วยสัญลักษณ์ที่มีความหมาย เช่น ADD หมายถึงการบวก ,MUL หมายถึงการคูณ เป็นต้น

3.ภาษาขั้นสูง (Higher-Level or User-Oriented Languages)

เป็นภาษาที่มีลักษณะหลายอย่างเพิ่มเติมขึ้นมา แต่อาจจะขาดการเข้าถึงระบบโดยตรง ลักษณะที่เพิ่มเติมเข้ามา ได้แก่ มีคำสั่งแบบเป็นโครงสร้าง ,ประโยคที่ถูกซ้อนใน (Nested Statements) ,กลุ่มระเบียน (Blocks) และกระบวนการ (Procedures)

4.ภาษาตรงปัญหา (Problem-Oriented Languages)

เป็นภาษาที่แสดงอยู่ในรูปของปัญหาที่ตรงกันกับขอบเขตที่กำหนด

3.2 ตัวแปลภาษา (Translators)

ตัวแปลภาษา (Translator) มีทำหน้าที่ในการอ่านสิ่งเข้า (Input) ที่เป็นชุดคำสั่งภาษาต้นฉบับ (Source Program) เข้ามา จากนั้นทำการแปลงให้เป็นชุดคำสั่งภาษาจุดหมาย (Object หรือ Target Program) ซึ่งชุดคำสั่งภาษาต้นฉบับนั้นจะถูกเขียนอยู่ในรูปภาษาต้นฉบับ (Source Language) และชุดคำสั่งภาษาจุดหมายจะอยู่ในรูปภาษาจุดหมาย (Object Language)

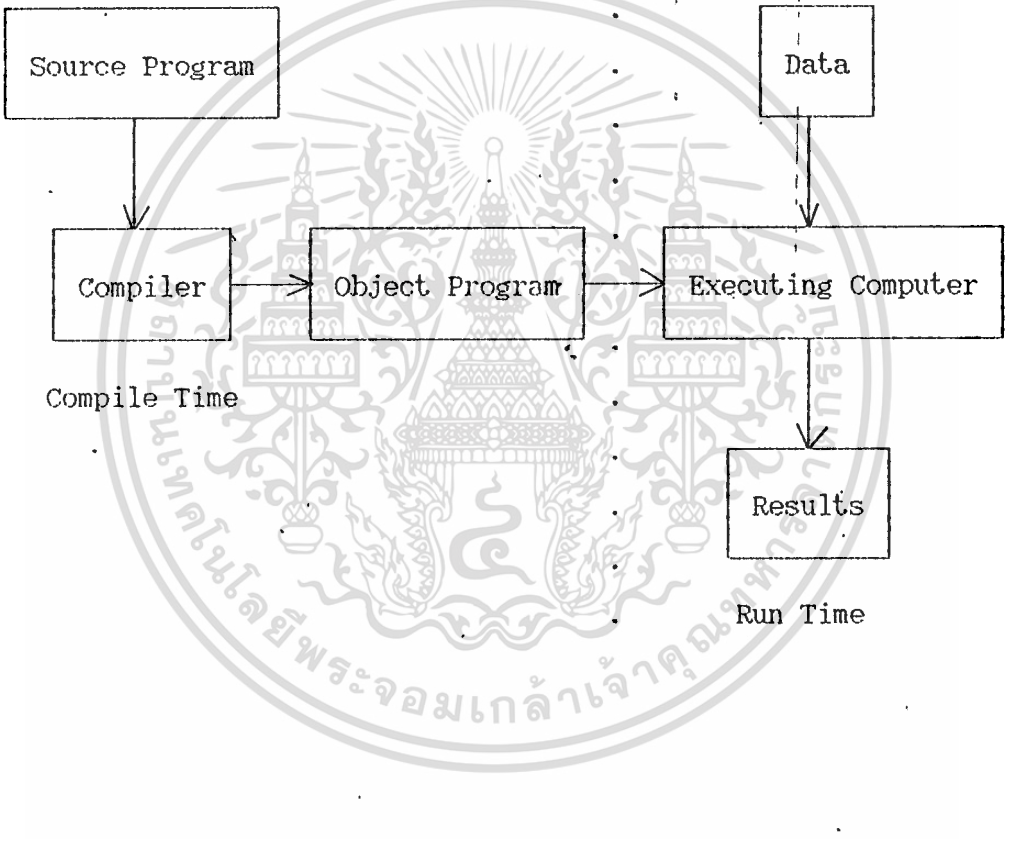
ถ้าภาษาต้นฉบับเป็นภาษาแอสเซมบลีและภาษาจุดหมายที่ได้เป็นภาษาเครื่องแล้ว ตัวตัวแปรภาษาที่ใช้แปลงนั้น จะเรียกว่า แอสเซมเบลอร์ (Assembler)

ภาษาแอสเซมบลีเป็นภาษาที่ใกล้ชิดกับภาษาเครื่องอย่างมาก เพราะแต่ละคำสั่งในภาษาแอสเซมบลีนั้น ส่วนมากจะเป็นการใช้แทนการทำงานของแต่ละคำสั่งในภาษาเครื่องโดยตรง และโครงสร้างของภาษาแอสเซมบลีจะอยู่ในรูปของเขต (Field) ที่เรียงลำดับกัน ตัวอย่างเช่น เขตแรกอาจจะแสดงถึงป้าย (Label) แล้วอาจจะตามด้วยเขตการปฏิบัติการ (Operation Field) และหลังป้ายนี้อาจจะมีเขตตัวถูกดำเนินการ (Operand Field) หนึ่งหรือสองเขตตามมา

แอสเซมเบลอร์ที่มีความสามารถสูงอาจจะเตรียมคำสั่งอื่น ๆ นอกเหนือจากคำสั่งที่ตรงกับคำสั่งเครื่อง เช่น คำสั่งมหัพภาค (Macroinstructions) โดยเป็นการประกอบคำสั่งต่าง ๆ ให้เป็นคำสั่งมหัพภาคเพียงคำสั่งเดียว แต่โครงสร้างของภาษาที่มีอยู่ในระดับสูง เช่น ข้อความสั่งแบบเคส (Case-Statement) ,ข้อความสั่งแบบซ้อนใน (Nested-Statement) และโครงสร้างแบบกลุ่มระเบียน (Block Structure) มักจะไม่พบในภาษาแอสเซมบลี

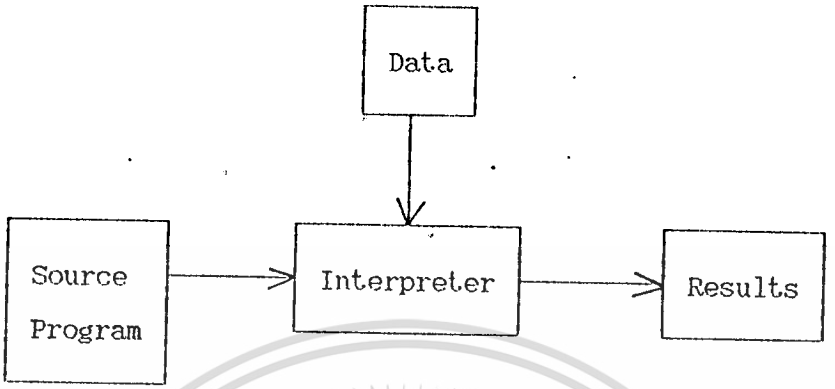
ส่วนตัวแปลภาษาที่ทำหน้าที่แปลงภาษาชั้นสูง เช่น ฟอร์แทรน (FORTRAN) ,ปาส-คาล (PASCAL) หรือโคบอล (COBOL) เป็นต้น ให้เป็นภาษาเครื่องหรือภาษาแอสเซมบลี ตัวแปลภาษานั้นจะเรียกว่า **ตัวแปลชุดคำสั่ง** (Compiler)

ภาพแสดงกระบวนการแปลชุดคำสั่ง (Compilation Process)



ตัวแปลคำสั่ง (Interpreter) เป็นตัวแปลภาษาอีกชนิดหนึ่งซึ่งจะทำการแปลชุดคำสั่งภาษาต้นฉบับให้เป็นรูปแบบภายใน (Internal Form) และการประมวลผลข้อมูลจะทำในเวลาเดียวกัน โดยจะไม่มีการสร้างชุดคำสั่งภาษาจุดหมายขึ้น

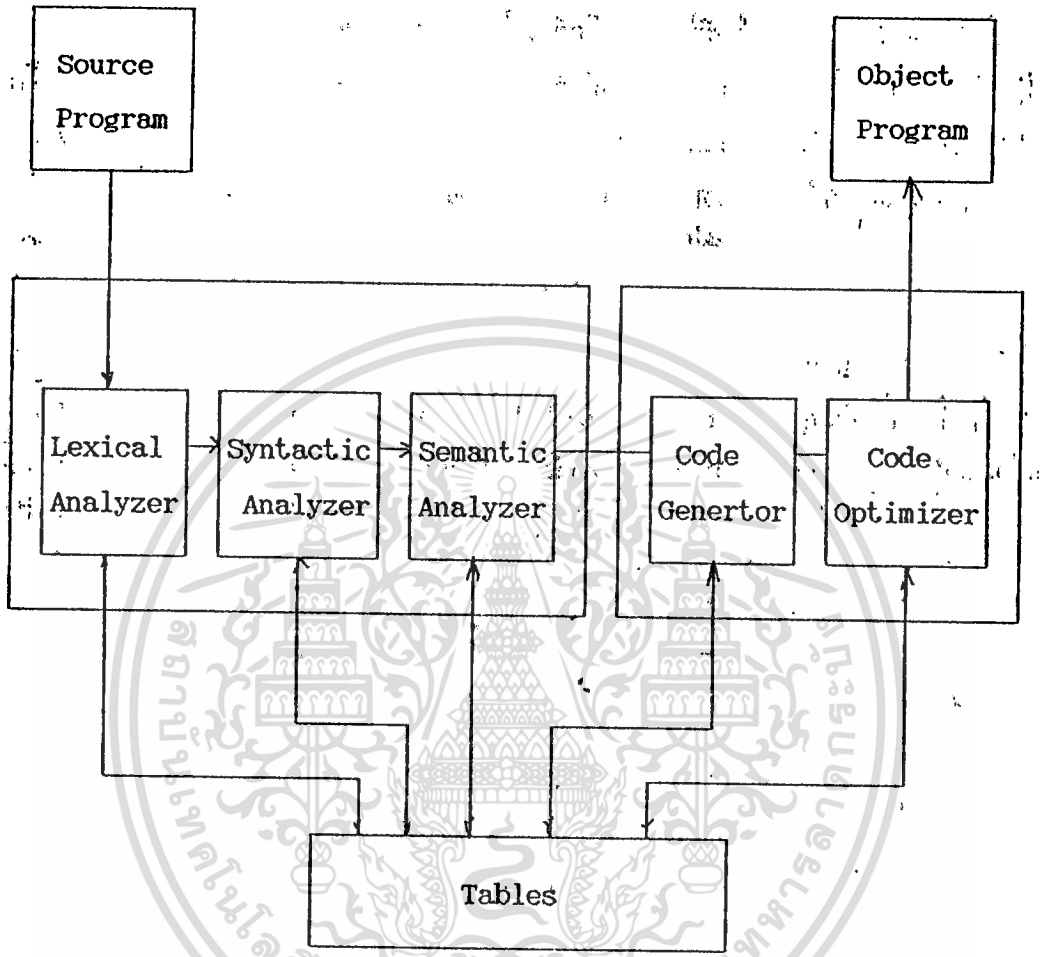
ภาพแสดงกระบวนการแปลคำสั่ง (Interpretive Process)



3.3 ตัวแบบของตัวแปลชุดคำสั่ง (Model of a Compiler)

ตัวแปลชุดคำสั่งจะประกอบด้วยการทำงานที่สำคัญ ๆ สองส่วน คือ วิเคราะห์ (Analysis) ชุดคำสั่งภาษาต้นฉบับออกเป็นส่วน ๆ จากนั้นนำแต่ละส่วนไปสังเคราะห์ (Synthesis) เพื่อสร้างชุดคำสั่งภาษาจุดหมายออกมา

ภาพแสดงส่วนประกอบของตัวแปลภาษา



ชุดคำสั่งภาษาต้นฉบับจะถูกส่งเข้าไปใน ตัววิเคราะห์ศัพท์ (Lexical Analyzer) หรือ ตัวกวาดตรวจ (Scanner) ซึ่งมีหน้าที่แยกข้อความที่เข้ามาให้เป็นส่วน ๆ หรือเป็นโทคเคน (Token) เช่น แยกออกเป็นค่าคงที่ (Constants) ,ชื่อตัวแปร (Variable Names) ,คำหลัก (Keywords) ได้แก่ DO ,IF ,THEN เป็นต้น และตัวปฏิบัติการ (Operators)

ซึ่งบางครั้งตัววิเคราะห์ศัพท์ยังอาจทำหน้าที่วิเคราะห์ไวยากรณ์ในระดับต่ำ และเพื่อประสิทธิภาพที่ดีขึ้น แต่ละชนิดโทคเคนจะถูกกำหนดโดย "ตัวเลขแสดงภายในอย่างเป็นหนึ่งเดียว" (Unique Internal Representation Number) ตัวอย่างเช่น ชื่อตัวแปรอาจถูกแทนด้วยเลข 1 ,ค่าคงที่แทนด้วย 2 ,ป้ายแทนด้วย 3,การบวก (+) แทนด้วย 4 เป็นต้น เช่นประโยค

```
TEST:    IF   A>B  THEN    X=Y;
```

(ใหม่)

หลังจากผ่านตัววิเคราะห์ศัพท์แล้วจะได้ลำดับของโทคเคนและตัวเลขแสดงภายในอย่างเป็นหนึ่งเดียว ดังต่อไปนี้

TEST	3
:	26
IF	20
A	1
>	15
B	1
THEN	21
X	1
=	10
Y	1
;	27

ตัววิเคราะห์ศัพท์จะทำการส่งโทคเคนเหล่านี้ไปให้ **ตัววิเคราะห์เชิงวากยสัมพันธ์ (Syntax Analyzer)** ซึ่งโทคเคนเหล่านี้อาจอยู่ในรูปแบบที่ประกอบด้วยสองส่วน คือส่วนแรกจะประกอบด้วยเลขที่อยู่ (Address) หรือตำแหน่ง (Location) ของโทคเคนนั้นในตารางสัญลักษณ์ (Symbol Table) และส่วนที่สองจะเป็นตัวเลขแสดงภายในอย่างเป็นหนึ่งเดียวของโทคเคนนั้น ที่ทำเช่นนี้เนื่องจากมีประโยชน์ต่อตัววิเคราะห์เชิงวากยสัมพันธ์เพราะทุกโทคเคนจะแสดงในรูปแบบข้อมูลที่มีความยาวคงที่

ตัววิเคราะห์เชิงวากยสัมพันธ์มีความซับซ้อนมากกว่าตัววิเคราะห์ศัพท์ ซึ่งการทำงานของตัววิเคราะห์เชิงวากยสัมพันธ์ จะทำการอ่านชุดคำสั่งภาษาต้นฉบับ (ในรูปของโทคเคน) ที่ผ่านตัววิเคราะห์ศัพท์มาแล้ว จากนั้นจะทำการวิเคราะห์โครงสร้างของประโยคว่าถูกต้องตามไวยากรณ์ของภาษาหรือไม่ สิ่งที่ได้จากตัววิเคราะห์เชิงวากยสัมพันธ์คือ รูปต้นไม้แสดงวากยสัมพันธ์ (Syntax Tree) หรือ บางสิ่งที่ใช้ได้เหมือนกัน

หลังจากนั้นรูปต้นไม้แสดงวากยสัมพันธ์ ที่ถูกสร้างจากตัววิเคราะห์เชิงวากยสัมพันธ์จะถูกนำไปใช้โดย **ตัววิเคราะห์ด้านความหมาย (Semantic Analyzer)** ซึ่งจะทำการวิเคราะห์ความหมายของชุดคำสั่งภาษาต้นฉบับ

ตัวอย่างประโยคเช่น (A+B)*(C+D) ตัววิเคราะห์ด้านความหมายทำการกำหนดว่าจะกระทำอะไรเมื่อพบตัวดำเนินการทางคณิตศาสตร์ (Arithmetic Operators) ได้แก่ การบวกและการคูณ และจะทำการตรวจสอบด้วยว่าตัวถูกดำเนินการ (Operand) ทั้งสองนั้นมีการกำหนดมาก่อนหรือไม่ ,มีชนิดเดียวกันหรือไม่ (ถ้าไม่ อาจจะมีการปรับให้เป็นชนิดเดียวกัน) และตัวถูกดำเนินการนั้นมีค่าอะไรอยู่

การทำงานของตัววิเคราะห์ด้านความหมายยังอาจรวมถึงการสร้าง รูปแบบชั่วคราว (Intermediate Form) ของรหัสต้นฉบับ (Source Code)

ตัวอย่างของรหัสต้นฉบับชั่วคราวของรหัสต้นฉบับ (Intermediate Source Code) ของประโยค (A+B)*(C+D) โดยใช้ควอดรัพเปิลโนเทชัน (Quadruple Notation) จะเป็นดังนี้

```
(+,A,B,T1)
(+,C,D,T2)
(*,T1,T2,T3)
```

จากตัวอย่างควอดรัพเปิลโนเทชัน จะประกอบด้วย 4 ส่วน คือ

(ตัวดำเนินการ,ตัวถูกดำเนินการ$_1$,ตัวถูกดำเนินการ$_2$,ผลลัพธ์)

ความหมายคือ จะนำตัวถูกดำเนินการ$_1$ และตัวถูกดำเนินการ$_2$ มากระทำอะไรบางอย่างแล้วแต่ตัวดำเนินการ จากนั้นนำผลที่ได้ไปเก็บไว้ที่ส่วนผลลัพธ์

ผลลัพธ์ที่ได้จากตัววิเคราะห์ด้านความหมายจะถูกนำไปยัง **ตัวก่อกำเนิดรหัส** (Code Generator) ที่จุดนี้รูปแบบชั่วคราวของภาษาของชุดคำสั่งภาษาต้นฉบับ (Source-Language Program) มักจะถูกแปลไปเป็นภาษาแอสเซมบลีหรือภาษาเครื่อง

ตัวอย่างของการแปลไปเป็นภาษาแอสเซมบลีของควอดรัพเปิลโนเทชันของตัวอย่างที่แล้ว จะเป็นดังนี้

```
LDA  A    ; นำค่าใน A ไปเก็บลงใน Accumulator
ADD  B    ; บวกค่าของ B เข้าไปใน Accumulator
STO  T1   ; เก็บค่าใน Accumulator ลงใน T1
LDA  C    ; นำค่าใน C ไปเก็บลงใน Accumulator
```

```
ADD  D    ; บวกค่าของ  D เข้าไปใน Accumulator
STO  T2   ; เก็บค่าใน Accumulator ลงใน T2
LDA  T1   ; นำค่าใน T1 ไปเก็บลงใน Accumulator
MUL  T2   ; คูณค่าของ T2 เข้าไปใน Accumulator
STO  T3   ; ; เก็บค่าใน Accumulator ลงใน T3
```

จากนั้นนำผลลัพธ์ที่ได้จากตัวก่อกำเนิดรหัสส่งเข้าไปยัง **ตัวปรับปรุงรหัส** (Code Optimizer) เพื่อที่จะทำการสร้างชุดคำสั่งของภาษาจุดหมายให้มีประสิทธิภาพเพิ่มขึ้นแล้วแต่จุดประสงค์ว่าต้องการประสิทธิภาพในทางใด เช่น ความรวดเร็ว หรือขนาดของชุดคำสั่งของภาษาจุดหมาย เป็นต้น ตัวอย่างภาษาแอสเซมบลีของตัวอย่างที่แล้วนำมาผ่านตัวปรับปรุงรหัสจะได้

```
LDA  A
ADD  B
STO  T1
LDA  C
ADD  D
MUL  T1
STO  T2
```

ซึ่งการทำงานของมัน จะเป็นในลักษณะ (C+D)*(A+B)

3.4 การค้นแบบทวิภาค (Binary Search)

มีขั้นตอนวิธี (Algorithm) ดังต่อไปนี้

ฟังก์ชัน BINARY_SEARCH (SARG)

กำหนดให้มีตารางที่เก็บค่าต่าง ๆ เรียงตามลำดับจากน้อยไปหามาก ดังนี้คือ $R_1, R_2, \ldots, R_n$ และใช้ตัวแปร $Name_i$ เป็นตัวชี้ค่าต่าง ๆ ในตาราง ฟังก์ชันนี้จะทำการหาค่าจากที่ส่งมา คือSARG ในตาราง ตัวแปรเฉพาะที่ในฟังก์ชันนี้ ได้แก่ B และ E ซึ่งแสดงค่าต่ำสุดและสูงสุดของช่วงสำหรับการค้นหาตามลำดับ ถ้า $Name_i$ ใด ๆ

ตรงกันกับ SARG ค่า i จะถูกส่งกลับไป แต่ถ้าไม่มีค่าใดที่ตรงกับ SARG เลย ฟังก์ชันนี้จะ ค่า -1 กลับไป

1.[กำหนดค่าเริ่มต้น]

B <- 1

E <- n

2.[เริ่มต้นค้นหา]

ขณะที่ B ยังน้อยกว่าหรือเท่ากับ E แล้ว ทำซ้ำถึงขั้นที่ 4

3.[หาตำแหน่งตรงกลาง]

i <- [(B+E)/2]

4.[เปรียบเทียบ]

ถ้า SARG < $NAME_i$ แล้ว

E <- i - 1

ถ้า SARG > $NAME_i$ แล้ว

B <- i + 1

ถ้า SARG = $NAME_i$ แล้ว

Return(i)

5.[การค้นหาไม่สำเร็จ]

Return(-1)

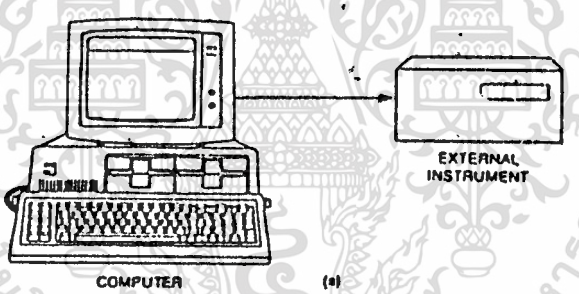
บทที่ 4

Z-80 EMULATOR และ สถาปัตยกรรม

จากความหมายของ อีมูเลชัน ในบทที่ 2 ในบทนี้จะกล่าวถึงสถาปัตยกรรมทั้งส่วนของฮาร์ดแวร์และซอฟต์แวร์ พร้อมทั้งบอกถึงรายละเอียดวิธีการออกแบบในส่วนต่าง ๆ ที่เกี่ยวข้อง จากระบบต่าง ๆ แล้วละเอียดไปจนถึงวงจรที่ใช้จริง นอกจากนี้ในส่วนของทางด้านซอฟต์แวร์ก็เป็นไปในทำนองเดียวกัน

4.1 สถาปัตยกรรมพื้นฐานของ Z-80 อีมูเลเตอร์

ลักษณะของ Z-80 อีมูเลเตอร์ที่ได้พัฒนาขึ้นจะอยู่ในรูปของบอร์ดที่เสียบในสล็อตของเครื่อง IBM PC/XT ซึ่งถ้านำมาใช้งานก็จะมีลักษณะดังรูปที่ 4.1



รูปที่ 4.1 ลักษณะการใช้งาน

ระบบพัฒนาไมโครโพรเซสเซอร์ที่ใช้การ์ด Z-80 อีมูเลเตอร์ประกอบด้วยส่วนสำคัญ 3 ส่วนคือ

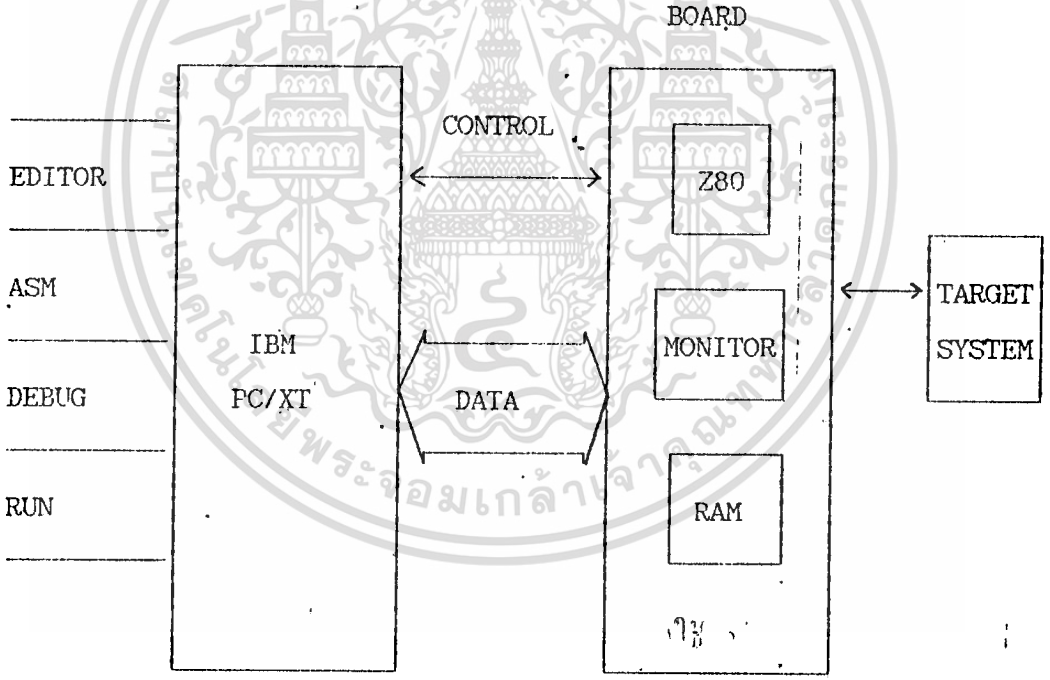
- IBM PC/XT ทำหน้าที่เปรียบเสมือนเป็นเครื่องปลายทาง (Terminal) ใช้ในการเขียนคำสั่งโดยอาศัยแผงแป้นอักขระ (Keyboard) และตัวจอภาพ (Monitor) รวมทั้งใช้ในการแอสเซมเบลอร์โปรแกรมให้เป็นรหัสดำเนินการ การทำงานของระบบทั้งหมดจะถูกควบคุมจากส่วนนี้

- การ์ด Z-80 อีมูเลเตอร์เป็นอุปกรณ์ที่มีไมโครโพรเซสเซอร์ Z-80 โดยทำหน้าที่แทนซีพียูของระบบเป้าหมาย (Target System) โดยการ์ดนี้ ยังมีความสามารถในการดีบักได้ด้วย

- ระบบเป้าหมาย (เป็นวงจรที่จะทดลอง) คือระบบที่ใช้ไมโครโพรเซสเซอร์เป็นส่วนประกอบในการพิจารณา ซึ่งส่วนนี้ถ้าไม่มีเราก็สามารถใช้การ์ดอีมูเลเตอร์เป็นส่วนช่วยในการพัฒนาและทดสอบโปรแกรม

4.2 บล็อกไดอะแกรมของระบบ Z-80 อีมูเลเตอร์

ในหัวข้อนี้จะแสดงส่วนประกอบต่าง ๆ ของ Z-80 อีมูเลเตอร์ที่กำหนดในขั้นต้นไว้ให้เป็นต้นแบบ และขอบเขตของระบบที่กำหนดไว้ในขั้นต้นนี้จะแสดงดังรูปที่ 4.2



รูปที่ 4.2 สถาปัตยกรรมของ Z80 EMULATOR

4.2.1 คุณลักษณะของอีมูเลเตอร์ Z-80

-ใช้กับไมโครโพรเซสเซอร์เบอร์ Z-80

-ใช้ร่วมกับ IBM PC/XT

-หน่วยความจำร่วม (Share Memory) มีขนาด 48 กิโลไบท์ โดยใช้สแตติกแรม (Static Ram)

-มีระบบซอฟท์แวร์รวม (Integrated Software System) บนเครื่องไอบีเอ็มพีซี ซึ่งผู้ใช้สามารถใช้อีดิเตอร์ทำการเขียนโปรแกรม แล้วใช้แอสเซมเบลอร์ทำการแปลรหัสต้นฉบับ (Source Code) ที่ได้ถ่ายลงหน่วยความจำ

-ผู้ใช้สามารถสั่งรีเซท (Reset) ,เอ็นเอ็มไอ (NMI) จากเครื่องปลายทางได้

-สามารถแสดงผลและทำการเปลี่ยนแปลงแก้ไขข้อมูลทั้งในเรจิสเตอร์และหน่วยความจำ

-สามารถทำการสั่งให้ซีพียู มีการทำงานทีละคำสั่ง (Single Step)

-ซีพียูสามารถทำงานตามเวลาจริงได้ (ยกเว้นการอินเตอร์รัพท์ จะเสียเวลาในการเข้าถึงแรมเล็กน้อย)

-สัญญาณที่ใช้มีความถี่สูง 3.5166 เมกะเฮิร์ต

-สามารถทำการกำหนดจุดหยุด (Break Point) โดยเครื่องปลายทางได้

-ผู้ใช้สามารถต่อระบบวงจรกับบอร์ดดังกล่าวได้

4.3 โครงสร้างทางด้านฮาร์ดแวร์

โครงสร้างของการ์ด Z-80 อีมูเลเตอร์ที่จะกล่าวถึงนี้เป็นระบบวงจรที่ทำหน้าที่ในการประสานการทำงานของไมโครโพรเซสเซอร์ 2 ฝ่ายด้วยกันคือฝ่ายซีพียูของเครื่องพีซีกับทางฝ่าย Z-80 โดยจะติดต่อกันตามกฎเกณฑ์ที่ตั้งเอาไว้หรือพิธีการ (Protocol) โดยรายละเอียดของการติดต่อจะกล่าวในเรื่องของโปรแกรมมอนิเตอร์ อุปกรณ์หรือส่วนที่จำเป็นในการติดต่อจะสามารถแสดงดังรูปที่ 4.3

ส่วนประกอบหลักๆ สามารถอธิบายได้ดังต่อไปนี้

- เครื่องไมโครคอมพิวเตอร์ไอบีเอ็มพีซี หรือเครื่องที่คอมแพทติเบิล (Compatible) ที่มีบัสแบบพีซีเอ็กซ์ที เป็นส่วนที่คอยติดต่อกับผู้ใช้ และขณะเดียวคอยควบคุมการทำงานหรือสั่งงานแก่ระบบบนบอร์ด คำสั่งต่าง ๆ และการรับคำตอบจะติดต่อผ่านทางพอร์ทที่ออกแบบไว้ซึ่งจะได้กล่าวในหัวข้อต่อไป ส่วนของเครื่องพีซีนี้ยังเป็นแหล่งจ่ายไฟ (Supply) ให้กับระบบบนบอร์ดอีกด้วย การที่ฝายของ Z-80 จะสามารถใช้หน่วยความจำร่วมได้หรือไม่นั้นก็ขึ้นกับส่วนของเครื่องพีซี

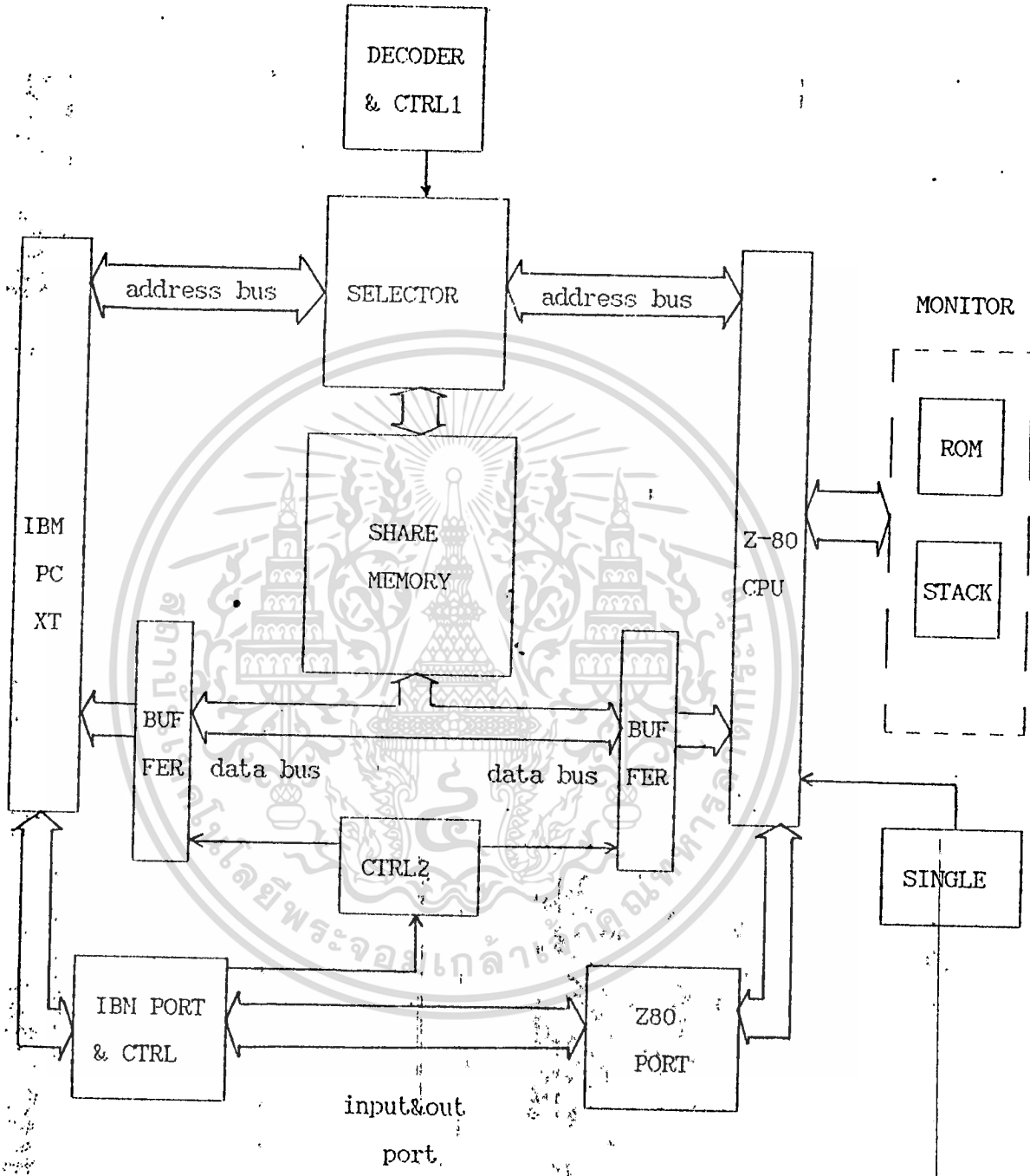
- ระบบของซีพียู Z-80 ก็จะเป็นส่วนที่หน้าที่คอยรับคำสั่งจากเครื่องพีซีไปทำงานในด้านต่าง ๆ ตามที่สั่ง ทำหน้าที่ควบคุมส่วนต่าง ๆ ของระบบบนบอร์ดที่เกี่ยวข้อง คือทำงานในโปรแกรมมอนิเตอร์ ควบคุมการทำงานแบบทีละคำสั่ง นอกจากนี้ถ้านำเอาไปประยุกต์ในการต่อกับอุปกรณ์ภายนอก สามารถนำสัญญาณจากขาซีพียู Z-80 ไปใช้ได้อีกด้วย

- หน่วยความจำร่วม หรือแรมสองทาง ที่ใช้ในโครงงานนี้จะใช้สแตติกแรม ในการที่จะให้เครื่องพีซีหรือทาง Z-80 อ่าน-เขียนข้อมูล โดยจะใช้ส่วนของตัวเลือก (Selector) ในการสับเปลี่ยนหน่วยความจำว่าจะให้ฝายใดอ้างหน่วยความจำได้ และในขณะเดียวกันจะมีส่วนของบัฟเฟอร์ (Buffer) ทั้งสองด้านที่มีการทำงานที่สัมพันธ์กัน ทำหน้าที่ในการที่จะสับเปลี่ยนบัสข้อมูลให้เป็นของฝายใดฝายหนึ่ง และในขณะนั้นจะมีส่วนของตัวถอดรหัส (Decoder) ทำหน้าที่ในการถอดรหัส ส่วนการที่ฝายไหนจะถอดรหัสได้นั้นก็จะมีการควบคุมในส่วนนี้ด้วย

หน่วยความจำร่วมนอกจากจะเป็นที่บรรจุโปรแกรมที่เขียนขึ้นจากทางด้านผู้ใช้แล้ว ก็จะมีเนื้อที่ส่วนหนึ่งในการส่งผ่านข้อมูลของระบบทั้งสองฝายโดยรายละเอียดจะอยู่ในหัวข้อโปรแกรมมอนิเตอร์

- มอนิเตอร์ (Monitor) ส่วนนี้จะประกอบด้วยรอม (ROM) ซึ่งเก็บโปรแกรมที่ใช้ควบคุมการทำงานของระบบ Z-80 และการติดต่อทั้งหมดระหว่างบอร์ดกับเครื่องพีซี

- ตัวถอดรหัสและตัวควบคุม1 (Decoder and CTRL1) ทำหน้าที่ในการถอดรหัสหน่วยความจำร่วม และตัวถอดรหัสอีกส่วนหนึ่งของ Z-80 จะใช้ทำการถอดรหัสหน่วยความจำที่เป็นของโปรแกรมมอนิเตอร์ อีกทั้งในส่วนนี้ยังเกี่ยวข้องกับสัญญาณการอ่านหรือเขียน



รูปที่ 4-3 แสดงโครงสร้างทางด้านฮาร์ดแวร์

- ตัวเลือก มีหน้าที่ในการที่จะให้ แอดเดรสบัสของฝายใดฝายหนึ่งติดต่อกับความจำร่วมได้ฝายเดียวเท่านั้น ซึ่งจะสังเกตได้ว่าส่วนของตัวเลือก จะมีอินพุทสองทางแต่จะมีเอาท์พุทออกมาแค่ทางเดียว

- บัฟเฟอร์ จะมีทั้งของทางด้านเครื่องพีซี และด้าน Z-80 ซึ่งบัฟเฟอร์ของทั้งสองฝายจะต้องทำงานให้สัมพันธ์กันเพื่อที่จะควบคุมสัญญาณบัสข้อมูลว่า ฝายใดสามารถที่จะทำการติดต่อกับหน่วยความจำร่วมได้

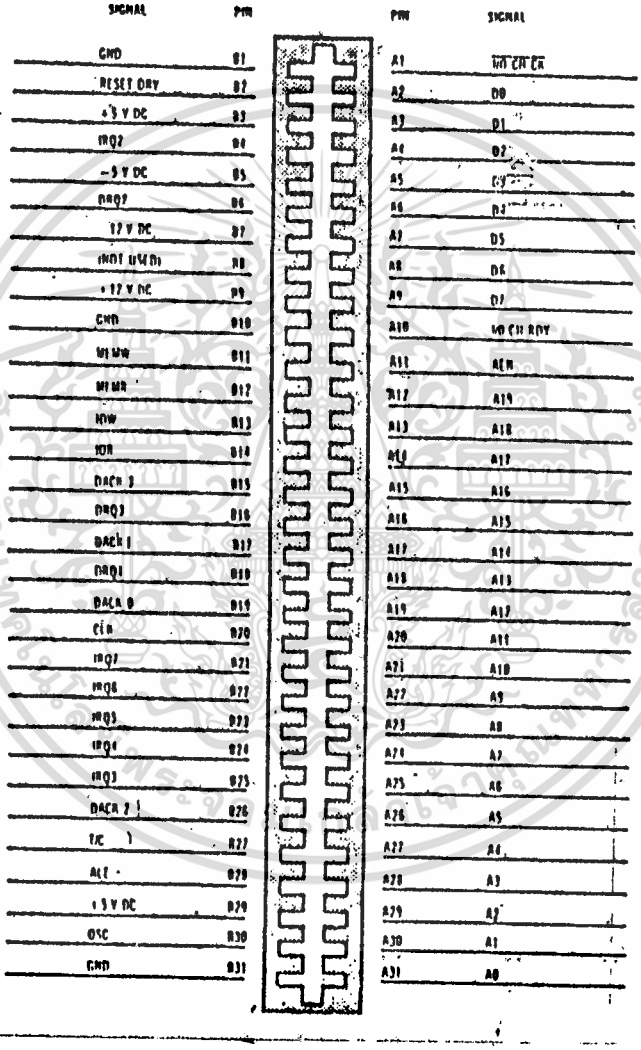
- ตัวควบคุม2 (CTRL2) ทำหน้าที่ควบคุมการทำงานของส่วนตัวเลือก และส่วนของบัฟเฟอร์ทั้งสองด้าน ให้สัมพันธ์กันก็คือ ถ้าในขณะที่ทางฝายเครื่องพีซีทำการอ้างแอดเดรสบัสข้อมูลของเครื่องพีซีก็จะต้องเชื่อมต่อกับบัสข้อมูลของส่วนหน่วยความจำร่วม และขณะเดียวกันต้องปิดหรือกันบัสข้อมูลและบัสแอดเดรสของฝาย Z-80 ไม่ให้ติดต่อกับบัสของส่วนหน่วยความจำร่วมได้ ในทางกลับกันก็จะกระทำเช่นนี้ในทางตรงข้าม เมื่อฝายของ Z-80 ทำการอ่านเขียนข้อมูลลงในส่วนของหน่วยความจำร่วม

- พอร์ทและตัวควบคุมของเครื่องพีซี (PC PORT & CTRL) ในส่วนนี้ก็คืออินพุทและเอาท์พุทพอร์ทของทางฝายเครื่องพีซี โดยทำหน้าที่ในการรับส่งข้อมูลระหว่างฝายเครื่องพีซีและทางฝาย Z-80 ข้อมูลต่างเหล่านี้ก็คือคำสั่งต่าง ๆ ที่จะให้ฝาย Z-80 ทำอะไร และการตอบรับจากทาง Z-80 นอกจากนี้ยังมีสัญญาณควบคุมที่ต่อออกจากเอาท์พุทพอร์ทนี้โดยตรง สัญญาณเหล่านี้ได้แก่ สัญญาณรีเซท ,สัญญาณเอ็นเอ็มไอ และ สัญญาณสเต็ป ที่ใช้ในกระบวนการทำงานทีละคำสั่ง

- พอร์ทของ Z-80 (Z-80 PORT) ในส่วนนี้ก็คืออินพุทและเอาท์พุทพอร์ทของทาง Z-80 ทำหน้าที่รับผิดชอบในการรับส่งข้อมูลระหว่างฝาย Z-80 และทางฝายเครื่องพีซี ซึ่งขั้นตอนในการรับส่งจะอยู่ในส่วนของโปรแกรมมอนิเตอร์

4.3.1 การออกแบบหน่วยความจำ

การติดต่อกับเครื่องพีซี เราจะอาศัยสล็อต ของเครื่องพีซี ซึ่งสัญญาณต่าง ๆ แสดงได้ในรูปที่ 4.4



รูปที่ 4.4 แสดงขาสัญญาณจากสล้อตของเครื่องพีซี

รายละเอียดของ หน่วยความจำ จะแสดงไว้ในรูปที่ 4.5 ซึ่ง ไอซี แต่ละตัวเป็นดังนี้

1) 2732 1 ตัว มีขนาด 4K โดย Z-80 จะมองเห็นที่ตำแหน่ง 0000-0FFFH เป็นที่เก็บของโปรแกรมมอนิเตอร์ซึ่ง หน่วยความจำ ที่ตำแหน่งนี้เครื่องพีซีจะมองไม่เห็น

2) 6116 1 ตัว มีขนาด 2K โดย Z-80 จะมองเห็นที่ตำแหน่ง 1000-17FFH เป็นที่เก็บของสแตกซึ่ง หน่วยความจำ ที่ตำแหน่งนี้เครื่องพีซีจะมองไม่เห็น

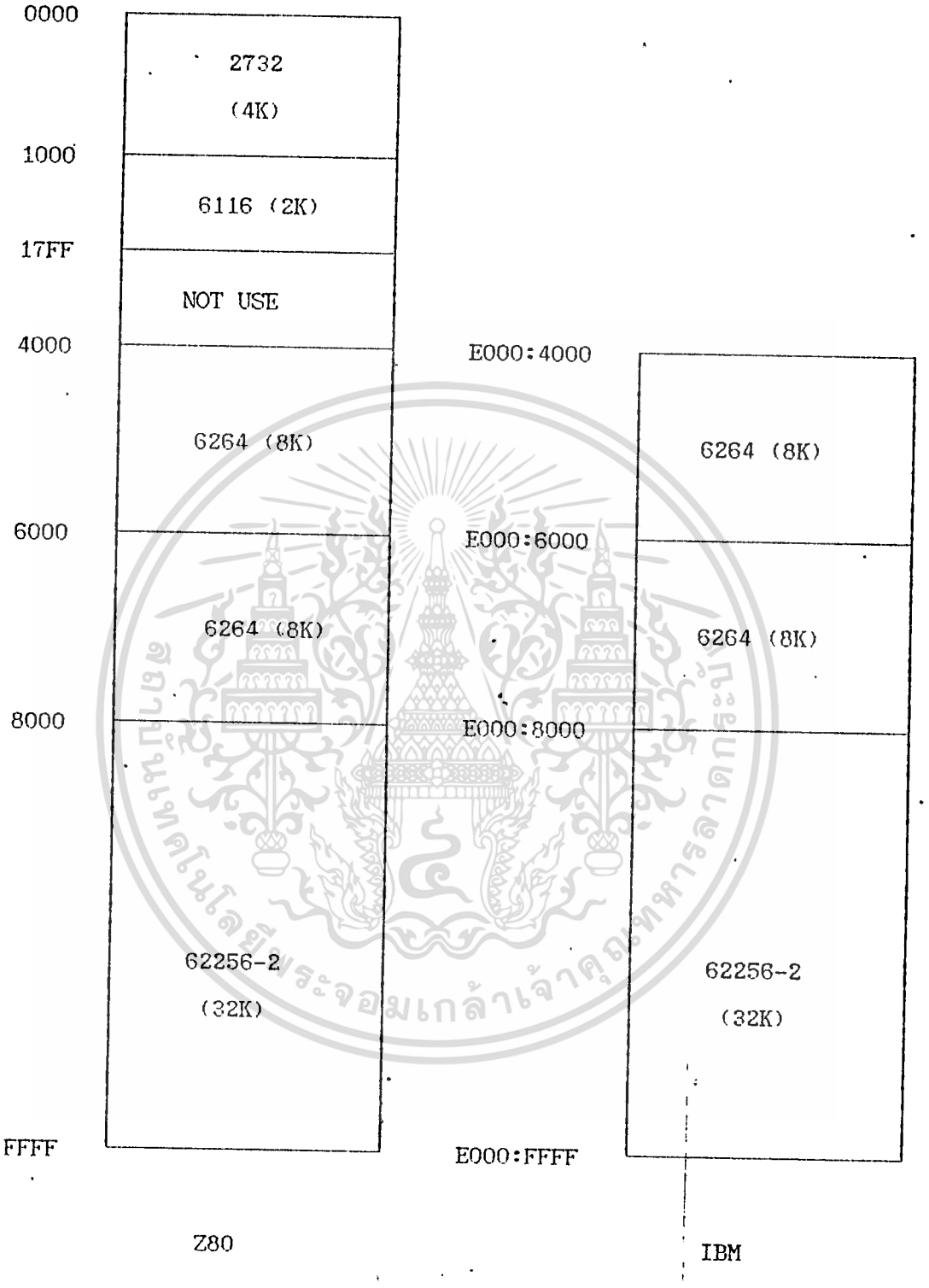
3) 6264 2 ตัว ตัวละ 8K ตัวแรก Z-80 มองเห็นที่ตำแหน่ง 4000-5FFFH ตัวที่-สองมองเห็นที่ตำแหน่ง 6000-7FFFH สำหรับเครื่องพีซีจะมองเห็นที่ตำแหน่ง E000:4000H-E000:5FFFH และที่ตำแหน่ง E000:6000H-E000:7FFFH

4) 62256-2 1 ตัว ตัวละ 32K Z-80 มองเห็นที่ตำแหน่ง 8000-FFFFH สำหรับเครื่องพีซีจะมองเห็นที่ตำแหน่ง E000:8000H-E000:FFFFH

การออกแบบวงจรที่ทำการถอดรหัสหน่วยความจำจะใช้ความสัมพันธ์ของขาแอดเดรสต่างๆ ทั้งของ Z-80 และของสล็อตจากเครื่องพีซีเองซึ่งวิธีการนี้สามารถแสดงได้ดังนี้

A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	-----	A0	HEX
1	1	1	0	0	0	0	0	0	0	-----	0	0000
1	1	1	0	0	0	0	0	1	1	-----	1	0FFF
1	1	1	0	0	0	0	1	0	0	-----	0	1000
1	1	1	0	0	0	0	1	0	1	-----	1	17FF
1	1	1	0	0	1	0	0	0	0	-----	0	4000
1	1	1	0	0	1	0	1	1	1	-----	1	5FFF
1	1	1	0	0	1	1	0	0	0	-----	0	6000
1	1	1	0	0	1	1	1	1	1	-----	1	7FFF
1	1	1	0	1	0	0	0	0	0	-----	0	8000
1	1	1	0	1	1	1	1	1	1	-----	1	FFFF

ซึ่งสามารถอธิบายได้ดังนี้คือ ทางด้านพีซี จะมีขาสัญญาณแอดเดรสทั้งหมด 20 ขาซึ่งในการออกแบบครั้งนี้จะใช้สัญญาณแอดเดรสขาที่ 19 - 16 เป็นเซกเมนต์ (Segment) คือกำหนดคงที่ให้เป็น E000 และขาแอดเดรสที่เหลือกำหนดให้เป็นออฟเซ็ท (Offset) ซึ่งจะมีการ ถอดรหัส ตรงกับทางด้าน Z-80 ที่มีขาสัญญาณแอดเดรส 16 ขา หน่วยความ-จำในส่วนนี้ก็คือการอ้างถึงหน่วยความจำสองทาง และการกระทำเช่นนี้ทำให้สามารถใช้ส่วนของวงจรถอดรหัสร่วมกันได้ แต่ในการอ้างแอดเดรสของมอนิเตอร์ในส่วนของ Z-80 ซีพียูนั้นจะแยกวงจรออกต่างหาก



รูปที่ 4-5 แสดงตำแหน่ง หน่วยความจำของระบบ

การออกแบบแอดเดรสของ หน่วยความจำบนเครื่องพีซีนั้นจะต้องทำการกำหนดแอดเดรสในช่วงที่ยังไม่ถูกใช้งานซึ่งจะแสดงได้ดังรูป 4.6 ดังจะเห็นได้ว่า ค่าตำแหน่งแอดเดรสที่ E0000H - 0EFFFFH ยังว่างอยู่จึงสามารถนำมาใช้งานได้ (เฉพาะเครื่อง IBM PC หรือ IBM PC/XT)

หน่วยความจำ ทั้ง 2 ส่วนนี้จะเป็นส่วนที่ร่วมกันของทั้งเครื่องพีซี และ Z-80 โดยให้เครื่องพีซีเขียน รหัส (CODE) ลงที่หน่วยความจำส่วนนี้ แล้วสวิทช์การทำงานไปที่ Z-80 ทำให้ สามารถอ่าน รหัสที่หน่วยความจำส่วนนี้ได้ แล้วนำค่ารหัสไปทำงานตามคำสั่งนั้น ๆ

4.3.2 การออกแบบส่วนของพอร์ท

การออกแบบส่วนของพอร์ทมีหลักการเช่นเดียวการกับการออกแบบแอสเดรสก็คือ บนเครื่องพีซีเราจะต้องคำนึงถึงแอดเดรสของพอร์ทที่สามารถนำมาใช้งานได้ซึ่งจะแสดงได้ดังรูปที่ 4.7 และ 4.8

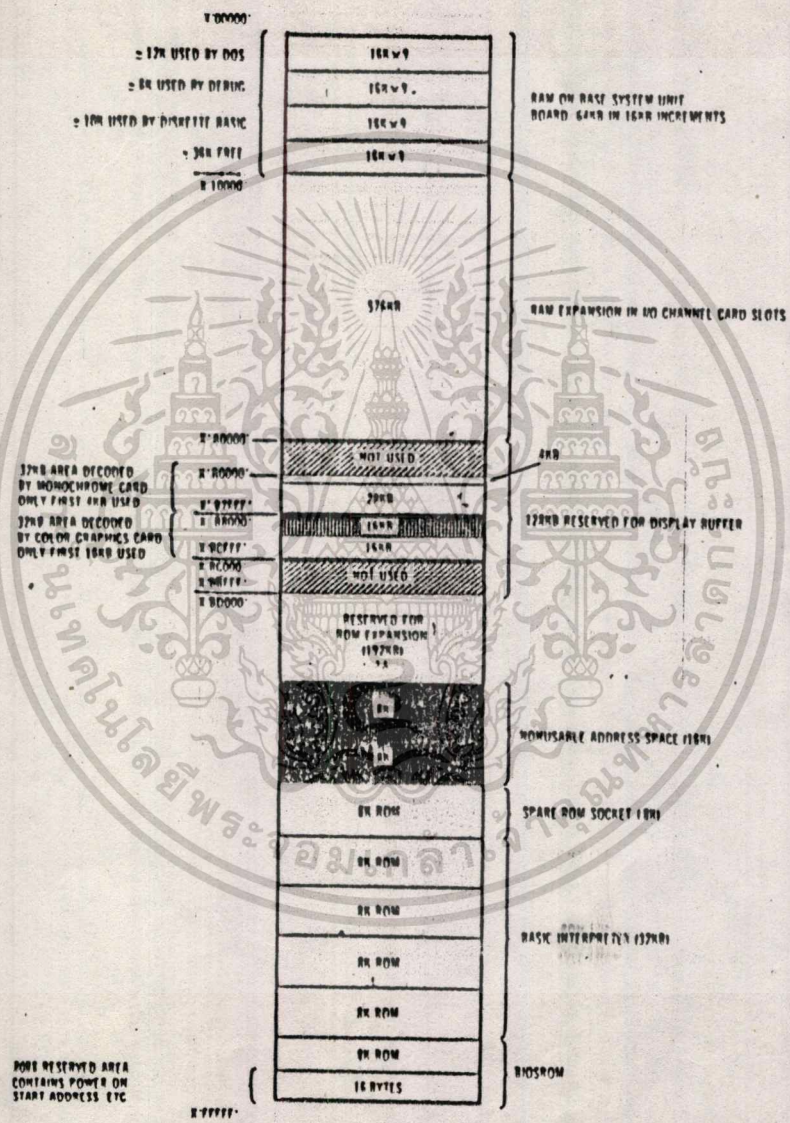
4.3.3 การออกแบบส่วนควบคุมการทำงานของ บัฟเฟอร์

ปัญหาใหญ่ของปริญญานิพนธ์ทางด้านฮาร์ดแวร์ที่ออกแบบยากที่สุดอยู่ที่จุดนี้ เพราะจะเกิดการรบกวนของบัสข้อมูลระหว่างระบบของสองฝ่ายและยังเกิดในกรณีเมื่อนำเอาระบบช่วยในการทดสอบมาใช้

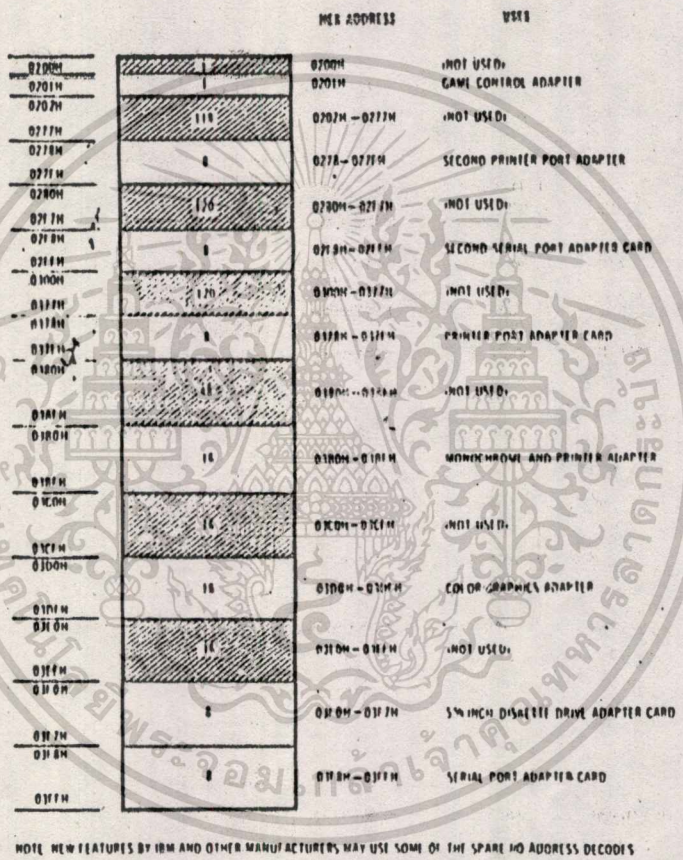
พีซี

จะขออธิบายถึงสัญญาณต่างๆ ที่เกี่ยวข้องกับการทำงานของวงจรส่วนนี้คือ

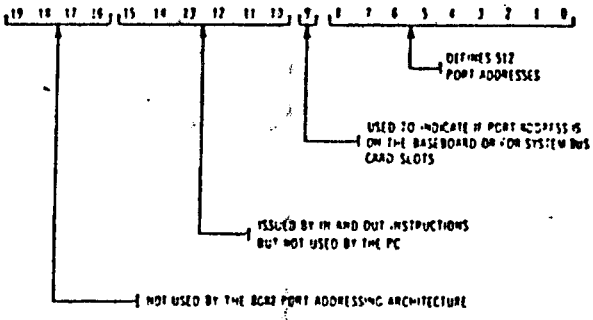
1) สัญญาณ CS คือสัญญาณที่แสดงการใช้งานของส่วนหน่วยความจำร่วม ถ้าฝ่ายใดฝ่ายหนึ่งของระบบอ้างถึงหน่วยความจำส่วนนี้ สัญญาณ CS จะมีสัญญาณเป็น 0 แต่ในตรงกันข้ามถ้าไม่มีการใช้งานของหน่วยความจำส่วนนี้ สัญญาณส่วนนี้ก็จะเป็น 1
2) สัญญาณ SW คือสัญญาณที่จะทำการควบคุมว่าจะให้ บัสแอดเดรสของฝ่ายใดเชื่อมต่อกับหน่วยความจำร่วม ถ้าสัญญาณนี้เป็น 0 จะทำให้บัสแอดเดรสของฝ่ายพีซีเชื่อมต่อกับระบบ หน่วยความจำร่วม ในทางตรงกันข้ามถ้าสัญญาณนี้เป็น 1 จะทำให้บัสแอดเดรสของฝ่าย Z-80 เชื่อมต่อกับระบบหน่วยความจำร่วม



รูปที่ 4.6 ตำแหน่งหน่วยความจำของเครื่องพีซี



รูปที่ 4.7 แสดงแอดเดรสพอร์ทของ สล๊อตเครื่องพีซี



รูปที่ 4.8 แสดงแอดเดรสพอร์ทของ 8088 MPU

3) สัญญาณ RD คือสัญญาณที่ควบคุมการอ่านและยังสามารถใช้เป็นสัญญาณควบคุมการเขียนได้อีกด้วย สัญญาณ RD นี้ถ้าเป็น 0 หมายถึงการที่จะทำการอ่านข้อมูลจากส่วนหน่วยความจำร่วม แต่ถ้าเป็น 1 แสดงว่าจะทำการเขียนข้อมูลลงในหน่วยความจำร่วม

จากความสัมพันธ์ของสัญญาณทั้งสามนำมาเขียนเป็นตารางลอจิก (Logic Table) เพื่อนำเอาไปสร้างวงจร ดังนี้

อินพุท			เอาท์พุท				ลักษณะการทำงาน
CS	SW	RD	G1	G2	DIR1	DIR2	
1	X	X	1	1	X	X	ไม่มีการอ่านเขียนข้อมูล
0	0	0	0	1	0	0	ฝาย IBM อ่านข้อมูล
0	0	1	0	1	1	1	ฝาย IBM เขียนข้อมูล
0	1	0	1	0	0	0	ฝาย Z-80 อ่านข้อมูล
0	1	1	1	0	1	1	ฝาย Z-80 เขียนข้อมูล

แอดเดรสพอร์ทบนเครื่องพีซีในปริญญานิพนธ์นี้ได้ใช้จำนวน 2 พอร์ทด้วยกันคือ

- อินพุทพอร์ท แอดเดรส 03E0H
- เอาท์พุทพอร์ท แอดเดรส 03E1H

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	PORT
0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	INPUT
0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	1	OUTPUT

ส่วนของทาง Z-80 มี 2 พอร์ทเช่นเดียวกัน แต่การออกแบบแอดเดรสพอร์ทจะใช้ขาสัญญาณแอดเดรสเพียง 8 ขาเท่านั้น พอร์ท 2 พอร์ทนี้คือ

- อินพุทพอร์ท แอดเดรส E1H
- เอาท์พุทพอร์ท แอดเดรส E0H

A7	A6	A5	A4	A3	A2	A1	A0	PORT
1	1	1	0	0	0	0	1	INPUT
1	1	1	0	0	0	0	0	OUTPUT

4.3.4 การออกแบบส่วนของการทำงานทีละคำสั่ง

การทำงานในส่วนนี้จะใช้วิธีการอินเตอร์รัพท์โดยการส่งสัญญาณลอจิก '0' ให้แก่ขา NMI ของ Z-80 การอินเตอร์รัพท์แบบนี้ระบบจะสามารถให้เกิดโดยวิธีการสองวิธีด้วยกันคือ

-ฝ่ายเครื่องพีซี ทำการเอาท์พอร์ทให้บิท 5 เป็นลอจิก '0'

-ฝ่าย Z-80 ตีความคำสั่งที่ได้มาว่าให้ทำการซิงเกิลสะเต็ปก็จะเอาท์พอร์ทบิท 7 เป็นลอจิก '1' ในขณะเดียวกันเมื่อ Z-80 ย้ายการทำงานมอนิเตอร์ไปยังแอดเดรสที่ต้องการทำการซิงเกิลสะเต็ป สัญญาณ MSLO จะเป็น '1' เช่นเดียวกันทำให้ขา 5 ของ U29B มีค่าลอจิก '0' ในขณะนั้น Z-80 ทำการเฟทช์คำสั่ง ดังนั้น M1,RD จะมีลอจิก '0' ทำให้ขา 5 ของ U29B เป็น '0' และเกิดการอินเตอร์รัพท์ลำดับ เมื่อเกิดการนอนมาสเคเบิ้ลมันโปรแกรมเคาท์เตอร์จะเปลี่ยนเป็น 0066 ก็ต่อเมื่อไซเคิลลูกสุดท้ายของคำสั่งนั้น ๆ ทำจนเสร็จ .

4.4 โปรแกรมมอนิเตอร์ (MONITOR PROGRAM)

โปรแกรมมอนิเตอร์ เป็นโปรแกรมที่ควบคุมการทำงานของ Z80 บนการ์ด โดยมีหน้าที่รับคำสั่งการทำงานจากผู้ใช้ผ่านทางเครื่องพีซีนำคำสั่งนั้นมาตีความหมาย แล้วไปทำงานตามคำสั่งนั้น ๆ เมื่อเสร็จสิ้นคำสั่งนั้นแล้วก็จะกลับมารอรับคำสั่งอื่นต่อไป

4.4.1 การทำงานของโปรแกรมมอนิเตอร์

ในส่วนของโปรแกรม มอนิเตอร์ จะใช้เนื้อที่ของแรมส่วนหนึ่งบนการ์ด เพื่อใช้ในการเก็บค่าเรจิสเตอร์ต่าง ๆ สำหรับการติดต่อกับเครื่องพีซี ซึ่งค่าตำแหน่งต่าง ๆ ที่ใช้มีดังนี้

SYSSTK	= 0179FH	ใช้เก็บค่าของ เรจิสเตอร์ SP
URAF	= 0FF00H	ใช้เก็บค่าของ เรจิสเตอร์ AF
URJAF	= 0FF02H	ใช้เก็บค่าของ เรจิสเตอร์ AF'
URBC	= 0FF04H	ใช้เก็บค่าของ เรจิสเตอร์ BC
URJBC	= 0FF06H	ใช้เก็บค่าของ เรจิสเตอร์ BC'
URDE	= 0FF08H	ใช้เก็บค่าของ เรจิสเตอร์ DE
URJDE	= 0FF0AH	ใช้เก็บค่าของ เรจิสเตอร์ DE'
URHL	= 0FF0CH	ใช้เก็บค่าของ เรจิสเตอร์ HL
URJHL	= 0FF0EH	ใช้เก็บค่าของ เรจิสเตอร์ HL'
URI	= 0FF10H	ใช้เก็บค่าของ เรจิสเตอร์ I
URR	= 0FF12H	ใช้เก็บค่าของ เรจิสเตอร์ R
URIX	= 0FF14H	ใช้เก็บค่าของ เรจิสเตอร์ IX
URIY	= 0FF16H	ใช้เก็บค่าของ เรจิสเตอร์ IY
URSP	= 0FF18H	ใช้เก็บค่าของ เรจิสเตอร์ SP
URPC	= 0FF2AH	ใช้เก็บค่าของ เรจิสเตอร์ PC
INPNUM	= 0FF2CH	ใช้เก็บค่าเบอร์ พอร์ท ที่ต้องการจะอ่าน
INPDATA	= 0FF2DH	ใช้เก็บค่าข้อมูลที่ได้จากคำสั่ง INPORT
OUTPNUM	= 0FF2EH	ใช้เก็บค่าเบอร์ พอร์ท ที่ต้องการจะเขียน
OUTPDATA	= 0FF2FH	ใช้เก็บค่าข้อมูลที่ต้องการจะเขียน
ADRST8	= 0FF32H	เก็บตำแหน่งที่จะไปทำงานของ RST 8H
ADRST10	= 0FF34H	เก็บตำแหน่งที่จะไปทำงานของ RST 10H

ADRST18 = 0FF36H เก็บตำแหน่งที่จะไปทำงานของ RST 18H

ADRST20 = 0FF38H เก็บตำแหน่งที่จะไปทำงานของ RST 18H

ADRST28 = 0FF3AH เก็บตำแหน่งที่จะไปทำงานของ RST 28H

ADRST30 = 0FF3CH เก็บตำแหน่งที่จะไปทำงานของ RST 30H

ADRST38 = 0FF3EH เก็บตำแหน่งที่จะไปทำงานของ RST 38H

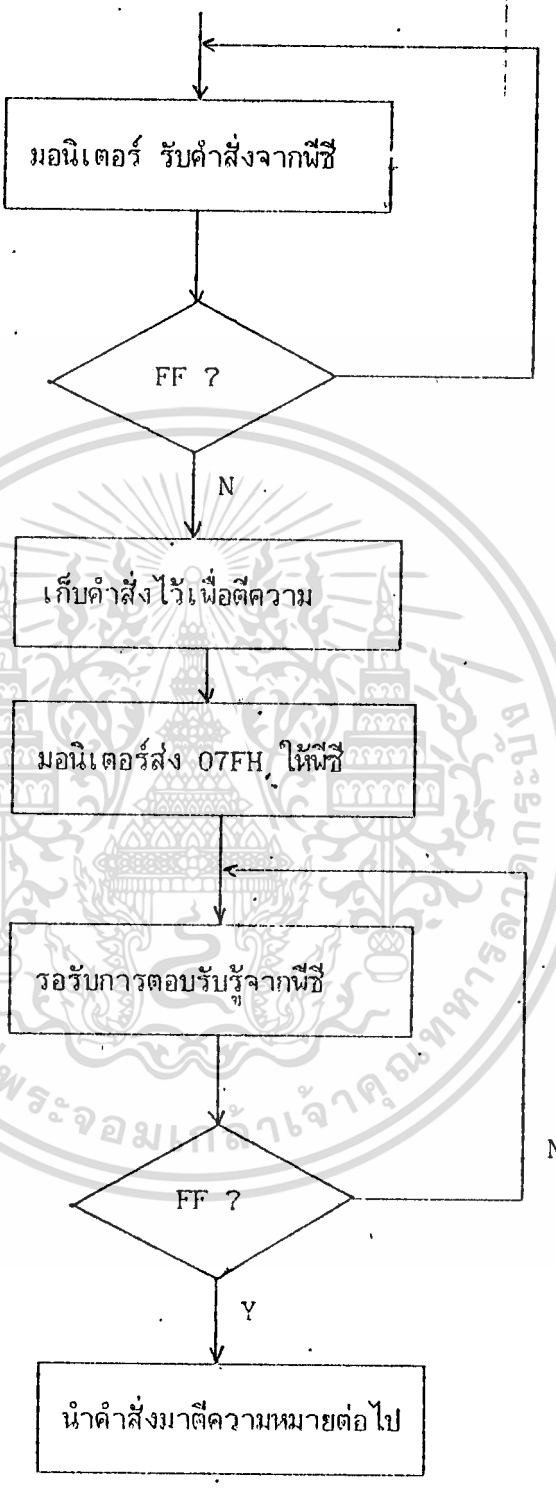
ADRST66 = 0FF40H เก็บตำแหน่งที่จะไปทำงานของ RST 66H

การติดต่อกันระหว่างเครื่องพีซีและ Z-80 จะติดต่อกันโดยผ่านพอร์ท ซึ่งเบอร์พอร์ทที่ใช้มีดังนี้

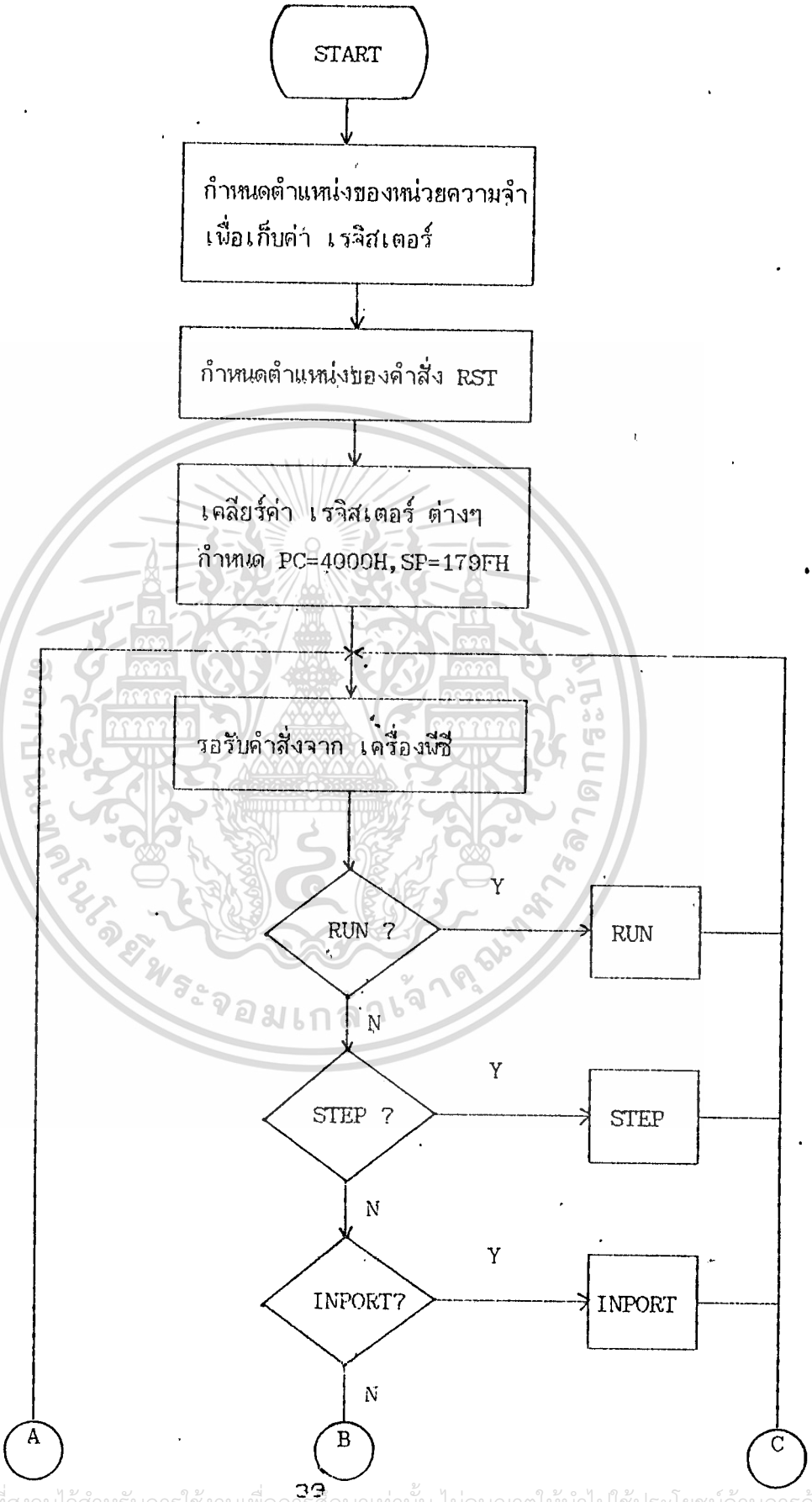
เครื่องพีซี	Z80
INPUT PORT = 3E0H	INPUT PORT = 0E1H
OUTPORT PORT = 3E1H	OUTPUT PORT = 0E0H

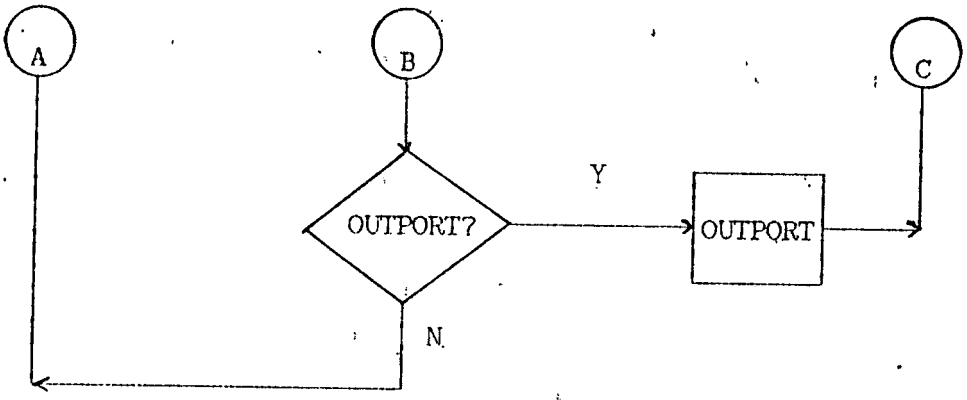
การติดต่อกันจะต้องมีการ แฮนด์เชค (HANDSHAKE) กันระหว่างพีซีและ Z-80 โดยกำหนดสถานะของพอร์ทในสภาพปรกติเป็น 80H โปรแกรมมอนิเตอร์ จะวนลูปรอรับคำสั่งจากพีซี เมื่อมีคำสั่งแรกเข้ามาโปรแกรม มอนิเตอร์ จะส่งค่า 07FH ออกไปให้พีซี ซึ่งพีซีจะได้รับค่าเป็น 0FFH เนื่องจากพีซีจะได้รับบิทที่ 7 เป็น 1 เสมอ เพื่อบอกให้พีซีรู้ว่าขณะนี้ โปรแกรมมอนิเตอร์ได้รับคำสั่งแล้ว จากนั้นพีซีจะส่งค่า 0FFH เพื่อบอกให้มอนิเตอร์รู้ว่าพีซีทราบแล้วว่า มอนิเตอร์ รับคำสั่งไว้แล้ว และพีซีพร้อมที่จะส่งคำสั่งต่อไปได้แล้ว การติดต่อกันระหว่างพีซี และ Z-80 แสดงได้ด้วยโฟว์ชาร์ตดังรูป 4.9

สำหรับโฟว์ชาร์ตแสดงการทำงานของโปรแกรมมอนิเตอร์ทั้งหมดแสดงได้ดังรูป 4.10



รูป 4.9 แสดงการติดต่อระหว่าง พีซี กับ Z80





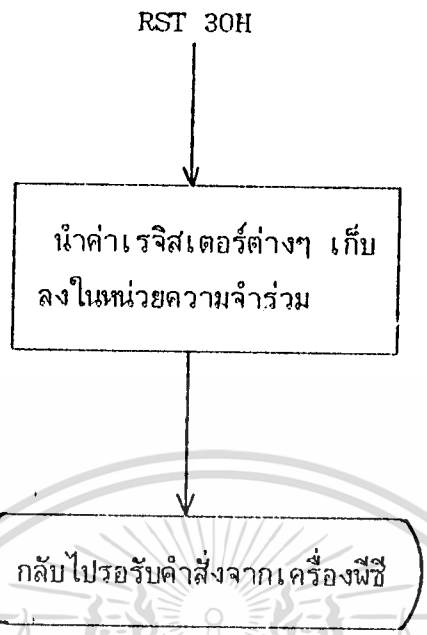
รูป 4.10 แสดงการทำงานของ โปรแกรมมอนิเตอร์ทั้งหมด

4.4.2 ความสามารถของโปรแกรมมอนิเตอร์

1. จุดหยุด (BREAK POINT)

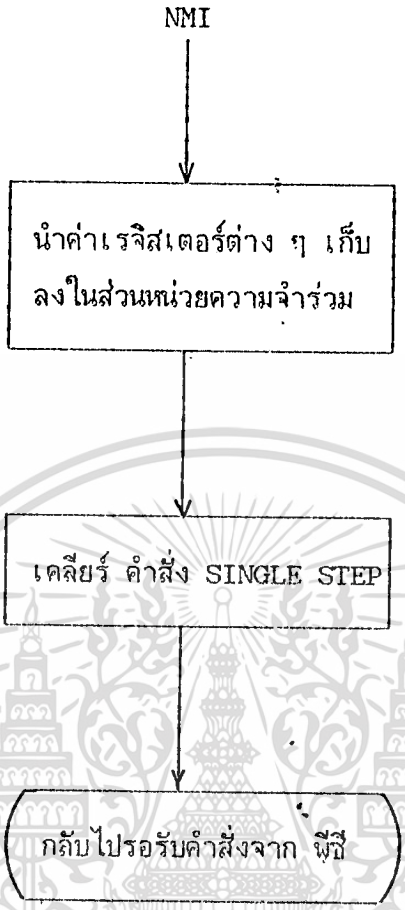
หลักการของ จุดหยุด จะใช้คำสั่ง RST 30H คือถ้าผู้ใช้ต้องการให้โปรแกรมหยุดทำงานที่คำสั่งไหน ก็ใส่คำสั่ง RST 30H ไว้ที่หลังคำสั่งนั้น โปรแกรมจะเริ่มทำงานตั้งแต่คำสั่งแรกที่เรจิสเตอร์ PC ชี้ไปจนกระทั่งถึงคำสั่ง RST 30H หลังจากนั้นจะแสดงค่าเรจิสเตอร์ ต่าง ๆ ที่ทำงานตั้งแต่ต้นจนถึงคำสั่งสุดท้ายก่อน RST 30H โปรแกรมที่เป็นอินเทอร์รัพท์เซอร์วิสรูทีน (Interrupt Service Routine หรือ ISR) ของ RST 30H จะทำหน้าที่เก็บค่าเรจิสเตอร์ทุกตัว รวมไปถึงเรจิสเตอร์สำรองด้วยลงในส่วนของหน่วยความจำร่วม เพื่อให้เครื่องพีซีสามารถอ่านค่า เรจิสเตอร์เหล่านี้เพื่อไปแสดงผลออกทางหน้าจอ

การตั้ง จุดหยุด สามารถทำได้ในขั้นตอนของการดีบัก โดยโปรแกรม ดีบัก จะใส่คำสั่ง RST 30H ให้ในตำแหน่งที่ผู้ใช้ต้องการให้โปรแกรมหยุดทำงาน และจะดึงคำสั่ง RST 30H ออกหลังจากเสร็จสิ้นการทำงานแล้ว การทำงานของคำสั่ง RST 30H สามารถแสดงได้ด้วยโฟว์ชาร์ตดังนี้



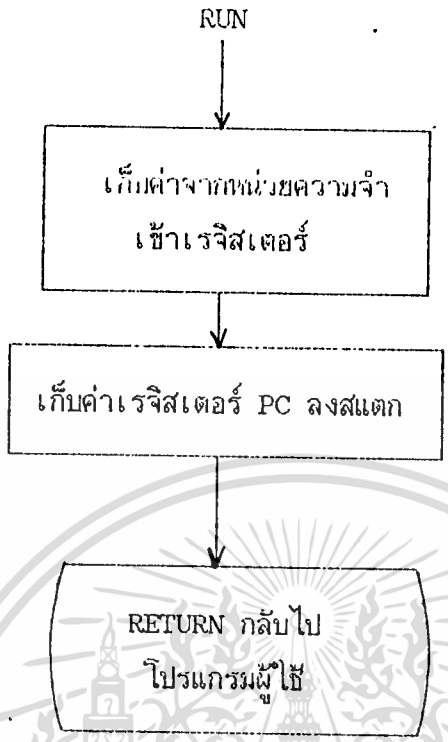
2. ทำงานทีละคำสั่ง (SINGLE STEP)

หลักการของการทำงานทีละคำสั่งจะใช้ NMI (NONMASKABLE INTERRUPT) เมื่อผู้ใช้สั่งให้ทำงานในโหมดนี้ โปรแกรม มอนิเตอร์ จะส่งคำสั่งที่ทำให้เกิด NMI ที่ขาของ Z-80 ในส่วนของ ISR ของ NMI จะคล้ายกับส่วนของจุดหยุด คือทำการเก็บค่าเรจิสเตอร์ ทุกตัวลงในส่วนหน่วยความจำร่วม เพื่อให้เครื่องพีซีสามารถอ่านค่า เรจิสเตอร์เหล่านี้เพื่อไปแสดงผลออกทางหน้าจอได้ หลังจากนั้นจะทำการเคลียร์สัญญาณ เพื่อให้ขา NMI ของ Z80 มีค่าโลจิกเป็น '1' การทำงานแบบทีละคำสั่งสามารถแสดงได้ด้วยโฟร์-ชาร์ตดังนี้



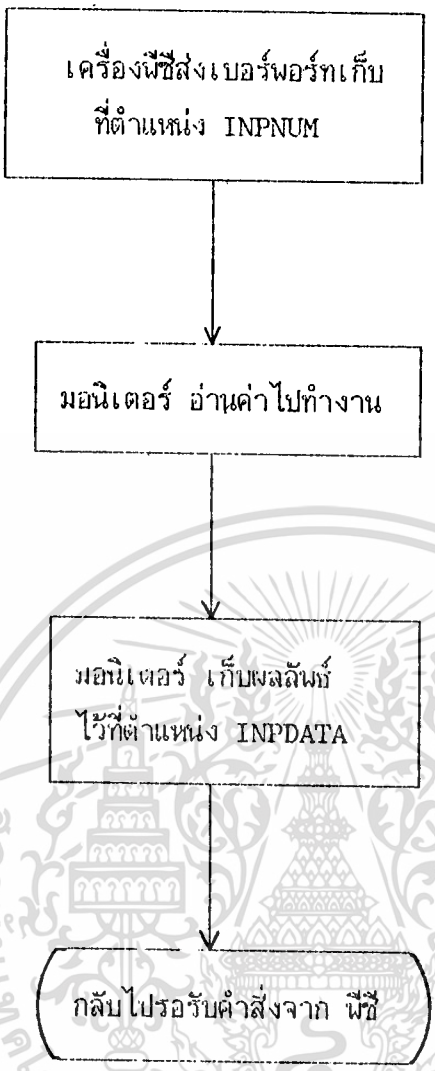
3. ทำงานรวดเดียว (RUN)

การทำงานของคำสั่งนี้ จะนำค่าที่เก็บไว้ในส่วนหน่วยความจำร่วมเข้าไปในเรจิสเตอร์ต่าง ๆ พร้อมกับเก็บค่าในเรจิสเตอร์ PC ปัจจุบัน ลงสแตก จากนั้นก็กลับไปยังโปรแกรม การทำงานจะเริ่มที่ตำแหน่งเรจิสเตอร์ PC ชี้อยู่ ไปจนจบโปรแกรม หรือเจอคำสั่ง RST 30H โดยหลักการนี้ ผู้ใช้สามารถกำหนดให้โปรแกรม เริ่มทำงานที่ตำแหน่งใดก็ได้ โดยการเปลี่ยนค่าที่ตำแหน่ง URPC ซึ่งจะเก็บค่าเรจิสเตอร์ PC การทำงานของคำสั่งนี้ แสดงได้ด้วยโฟว์ชาร์ตดังนี้



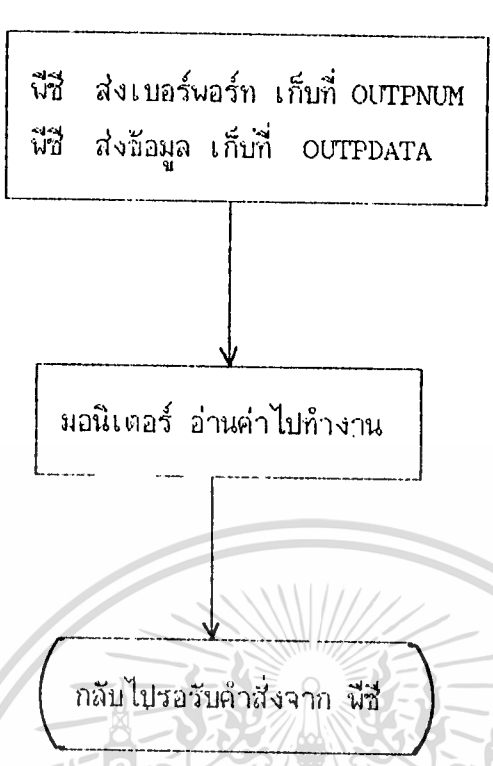
4. อ่านค่าจากพอร์ท (INPORT)

การทำงานของคำสั่งนี้ ผู้ใช้จะส่งหมายเลขพอร์ทที่ต้องการเข้ามาจากด้านเครื่องพีซี จากนั้นจะนำค่าเบอร์พอร์ทมาเก็บไว้ที่ตำแหน่ง INPNUM โปรแกรม มอนิเตอร์ จะอ่านค่าเบอร์พอร์ทจากตำแหน่ง INPNUM นี้ในการทำงาน ผลลัพธ์ที่ได้จากการทำคำสั่ง INPORT โปรแกรม มอนิเตอร์ จะนำไปเก็บไว้ที่ หน่วยความจำ ตำแหน่ง INPDATA จากนั้นเป็นหน้าที่ของเครื่องพีซีในการอ่านค่าที่ตำแหน่งนี้ไปแสดงผลบนจอ เพื่อให้ผู้ใช้ทราบต่อไป การทำงานสามารถแสดงได้ด้วยโฟว์ชาร์ตดังนี้



5. เทียนค่าออกพอร์ท (OUTPORT)

การทำงานของคำสั่ง เอาท์พอร์ท ผู้ใช้จะต้องส่งเบอร์พอร์ท และข้อมูลที่จะส่งออกไป โดยผ่านทางเครื่องพีซี จากนั้นจะนำค่าเบอร์พอร์ทเก็บไว้ที่ตำแหน่ง OUTPNUM และค่าที่จะส่งออกเก็บไว้ที่ตำแหน่ง OUTPDATA ทางด้านโปรแกรม มอนิเตอร์ จะใช้ค่าเหล่านี้ในการทำงาน ซึ่งการทำงานสามารถแสดงได้ด้วยโฟว์ชาร์ตดังนี้



6. รีสตาร์ท (RESTART)

สำหรับคำสั่ง รีสตาร์ท ที่ใช้ได้คือ RST 8H, RST 18H, RST 20H, RST 28H และ RST 38H ส่วนคำสั่ง RST 30H ใช้สำหรับการทำ จุดหยุด ในส่วนของ ISR ของแต่ละคำสั่งสามารถจะเขียนเข้นที่ตำแหน่งใดในแรมก็ได้ ซึ่งในแต่ละคำสั่งจะต้องเก็บค่าตำแหน่งที่จะไปทำงานไว้ที่หน่วยความจำส่วนเพิ่งเกสาร์ต ดังมีรายละเอียดดังนี้

- RST 8H เก็บค่าตำแหน่งที่จะไปทำงานที่ ADRST8 หรือ ตำแหน่ง 0FF32H
- RST 10H เก็บค่าตำแหน่งที่จะไปทำงานที่ ADRST10 หรือ ตำแหน่ง 0FF34H
- RST 18H เก็บค่าตำแหน่งที่จะไปทำงานที่ ADRST18 หรือ ตำแหน่ง 0FF36H
- RST 20H เก็บค่าตำแหน่งที่จะไปทำงานที่ ADRST20 หรือ ตำแหน่ง 0FF38H
- RST 28H เก็บค่าตำแหน่งที่จะไปทำงานที่ ADRST28 หรือ ตำแหน่ง 0FF3AH
- RST 38H เก็บค่าตำแหน่งที่จะไปทำงานที่ ADRST38 หรือ ตำแหน่ง 0FF3EH

ข้อจำกัดของการใช้คำสั่ง รีสตาร์ท เหล่านี้คือ ค่าเรจิสเตอร์ HL จะมีการเปลี่ยนค่าไป ดังนั้น ก่อนที่จะมีการใช้คำสั่ง รีสตาร์ท เหล่านี้ ต้องมีการเก็บค่าเรจิสเตอร์ HL ไว้ก่อน หลังจากคำสั่ง รีสตาร์ท นั้นแล้วค่อยนำค่า เรจิสเตอร์ HL ที่เก็บไว้คืนมา เช่น

```
PUSH HL
RST  10H
POP  HL
```

หลักการทำงานเมื่อมีคำสั่ง รีสตาร์ท เข้ามา เช่น RST 10H ที่ตำแหน่งแอดเดรส 10H จะนำค่าของแอดเดรสที่จะกระโดดไปทำงานคือค่าที่เก็บไว้ใน หน่วยความจำตำแหน่ง ADRST10 เก็บลงในเรจิสเตอร์ HL จากนั้นจึงกระโดดไปทำงานที่ตำแหน่ง HL ชี้อยู่ ส่วนคำสั่ง รีสตาร์ท อื่น ๆ มีหลักการทำงานเหมือนกัน ตัวอย่างการทำงานของ RST 10H แสดงได้ด้วยโฟว์ชาร์ตดังนี้

โปรแกรมมอนิเตอร์จะเขียนเก็บไว้ในอีพรอม (EPROM) เบอร์ 2732 Z-80 จะมองเห็นได้ที่ตำแหน่ง 0 - 2000H เมื่อรีเซ็ทซีพียู โปรแกรมจะเริ่มทำงานในส่วนของมอนิเตอร์ ซึ่งจะคอยวนลูปรอรับคำสั่งการทำงานจากเครื่องพีซี และไปทำงานตามคำสั่งนั้น ๆ

บทที่ 5
แอสเซมเบลอร์ ,ดีบักเกอร์ และอีดิเตอร์
(Assembler ,Debugger and Editor)

5.1 แอสเซมเบลอร์

โครงการนี้จะทำแอสเซมเบลอร์ไว้ใช้ในการแปลภาษาแอสเซมบลี ไปเป็นภาษาเครื่องของซีพียูเบอร์ Z-80 โดยส่วนประกอบต่าง ๆ ของแอสเซมเบลอร์จะมีดังต่อไปนี้

1.ตัววิเคราะห์ศัพท์ (Lexical Analyzer)

ทำหน้าที่แยกประโยคของภาษาแอสเซมบลีที่เข้ามาออกเป็นโทคเคน (Tokens) แต่ไม่มีการกำหนดตัวเลขแสดงภายในอย่างเป็นหนึ่งเดียว (Unique Internal Representation Number) สำหรับแต่ละชนิดของโทคเคน

2.ตัววิเคราะห์เชิงวากยสัมพันธ์ (Syntax Analyzer)

ทำหน้าที่ตรวจสอบประโยคภาษาแอสเซมบลีที่เป็นถูกแยกออกเป็นโทคเคนแล้วนั้น ว่าถูกต้องตามไวยากรณ์ของภาษาหรือไม่

3.ตัววิเคราะห์ด้านความหมาย (Semantic Analyzer)

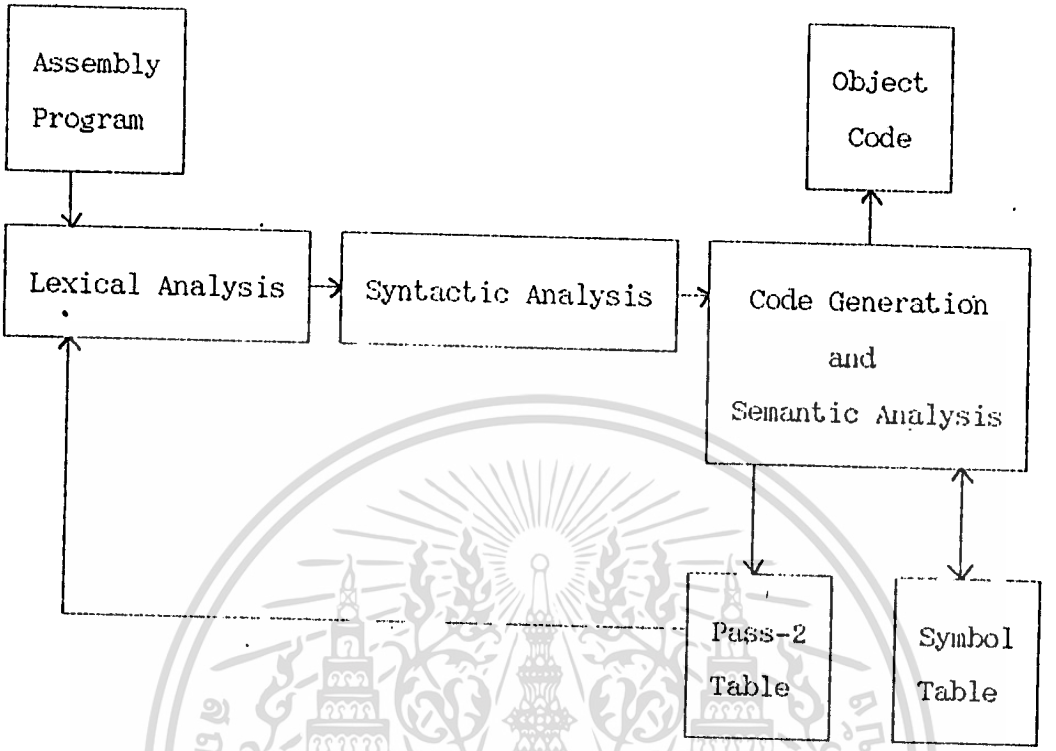
ทำหน้าที่ตรวจสอบชื่อป้าย (Label) และตัวแปรในตาราง และสร้างรูปแบบชั่วคราว (Intermediate Form) ของคำสั่งนั้นขึ้นมา

4.ตัวก่อกำเนิดรหัส (Code Generator)

ทำหน้าที่สร้างภาษาเครื่องของ Z-80 โดยนำรูปแบบชั่วคราวที่ถูกสร้างขึ้นจากตัววิเคราะห์ด้านความหมายไปค้นในตาราง เพื่อหารหัสเครื่อง (Machine Code) ของภาษาแอสเซมบลีนั้นออกมา

การทำงานของทั้ง 4 ส่วน จะทำงานต่อเนื่องกัน และไกล้ชิดกัน โดยแต่ละคำสั่งแอสเซมบลีจะถูกผ่านทั้ง 4 กระบวนการ และได้รหัสดำเนินการ (Operation Code) ออกมาก่อนที่จะแปลคำสั่งต่อไป

ภาพแสดงการทำงานของแอสเซมเบลอร์



5.1.1 ชุดคำสั่งภาษาแอสเซมบลี (Assembly Program)

เป็นโปรแกรมที่ถูกเขียนด้วยภาษาแอสเซมบลีของ Z-80 ซึ่งมีรูปแบบของภาษาดังต่อไปนี้

[<Label>:] <Operator> [<Operand>[,<Operand>]] [; <Comment>]

เช่น

```
        ORG     100h
        LD      IX,05FH
        LD      A,1
Test:   CP      (IX+02DH)       ; Test
        JR      Z,Quit
        INC     IX
        INC     A
```

```
                JR          Test
Quit:           HALT                        ; End
                END
```

5.1.2 การวิเคราะห์ศัพท์ (Lexical Analysis)

จะทำการการแยกประโยคของภาษาแอสเซมบลีออกเป็นโทคเคนย่อย โดยใช้

- ใช้ช่องว่างเพื่อใช้แยก ป้าย ,ตัวดำเนินการ ,ตัวถูกดำเนินการ และหมายเหตุ (Comment) ออกจากกัน
- ใช้เครื่องหมายจุดคู่ (':') เพื่อแยกป้ายออกมา
- ใช้ลูกน้ำ (',') เพื่อใช้แยกตัวถูกดำเนินการออกจากกัน
- ใช้เครื่องหมายบวก (+) เพื่อใช้แยกค่าในตัวถูกดำเนินการออกเป็นเรจิสเตอร์ (Register) กับค่า
- ใช้ เครื่องหมายอัฒภาค (';') เพื่อตัดหมายเหตุทิ้งไป

ข้อจำกัดจะมีดังต่อไปนี้

1) จะต้องมีช่องว่างระหว่าง ป้าย ,ตัวดำเนินการ ,ตัวถูกดำเนินการ และหมายเหตุเสมอ

2) ถ้ามีตัวถูกดำเนินการสองตัว จะต้องกั้นด้วยลูกน้ำ และต้องไม่มีช่องว่างระหว่างตัวถูกดำเนินการทั้งสองตัวนี้ และภายในตัวถูกดำเนินการจะต้องไม่มีช่องว่างอยู่

5.1.3 การวิเคราะห์วากยสัมพันธ์ (Syntactic Analysis)

นำโทคเคนที่ถูกแยกโดยตัววิเคราะห์ศัพท์มาวิเคราะห์ว่าถูกต้องตามรูปแบบของภาษาแอสเซมบลี ซึ่งการตรวจสอบจะค่อนข้างง่าย เนื่องจากโครงสร้างของภาษาเรียงลำดับไว้อย่างแน่นอนแล้ว ดังนั้นจึงแค่นำโทคเคนที่ได้มาตรวจสอบว่าเป็นไปตามลำดับของภาษาหรือไม่เท่านั้น และถ้าจำเป็นจะมีการเรียกตัววิเคราะห์ศัพท์อีกที เพื่อแยกลงเป็นส่วนย่อยที่สุดอีกสำหรับตรวจสอบ

5.1.4 การวิเคราะห์ด้านความหมาย (Semantic Analysis)

จะทำการตรวจสอบว่าป้ายที่จะกระโดดไป และตัวแปรที่นำมาใช้นั้น มีการกำหนดมาก่อนหรือไม่ โดยการตรวจสอบในตารางที่เก็บชื่อป้ายกับตารางที่เก็บชื่อตัวแปร ถ้าไม่มี

การกำหนดมาก่อนจะทำการเก็บหมายเลขบรรทัดนี้ไว้ใน Pass-2 Table เพื่อจะได้ถูกแปลเฉพาะบรรทัดนั้นหลังจากผ่าน Pass แรกไปแล้ว เป็นการประหยัดเวลา จากนั้นจะทำการสร้างรูปแบบชั่วคราว ซึ่งมีลักษณะคล้ายกันกับคำสั่งนั้น

โดยที่ตัวถูกดำเนินการที่เป็นตัวเลขจะถูกแปลงเป็น JN และตัวถูกดำเนินการที่เป็นตัวแปรหรือป้าย จะถูกแปลงเป็น (JN) เช่น

```
LD   A,20H
CALL Next
LD   A,Mem
```

เมื่อผ่านกระบวนการนี้ จะได้

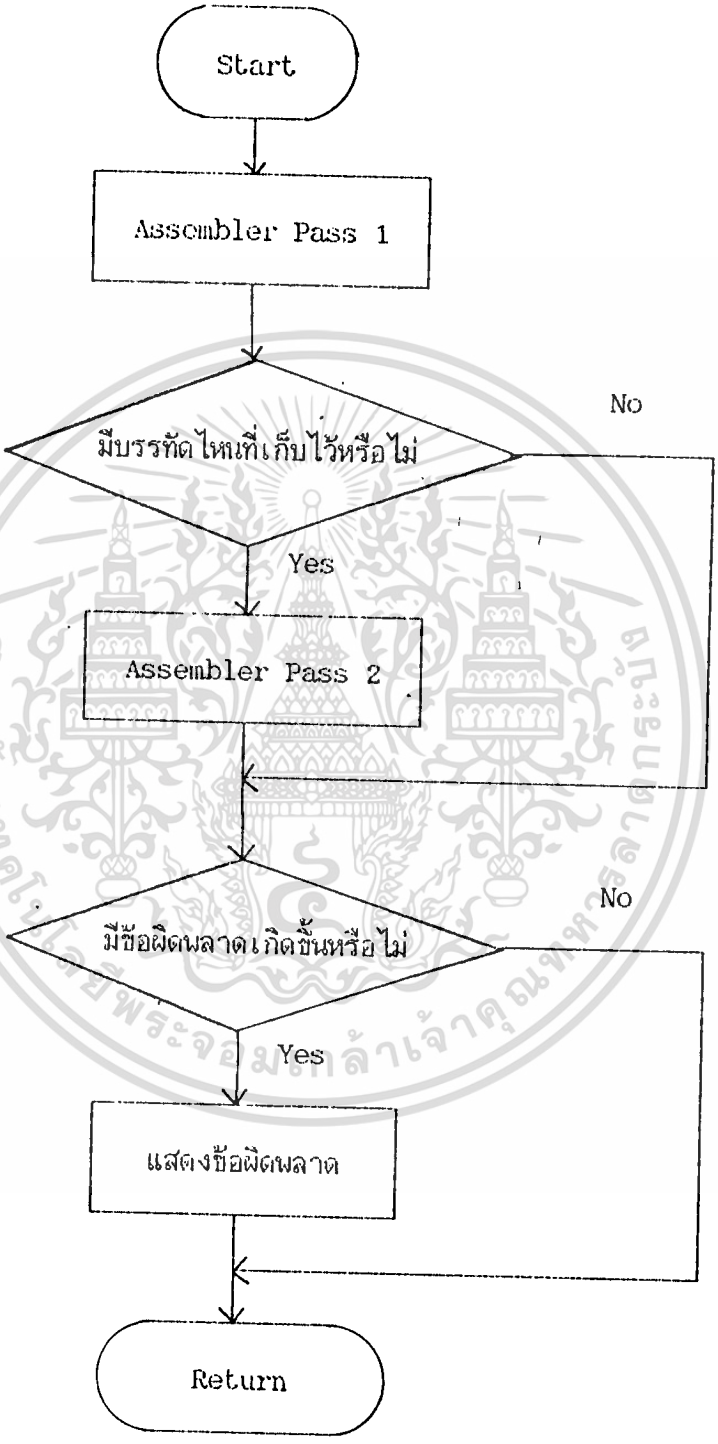
```
LD   A,JN
CALL (JN)
LD   A,(JN)
```

เป็นต้น

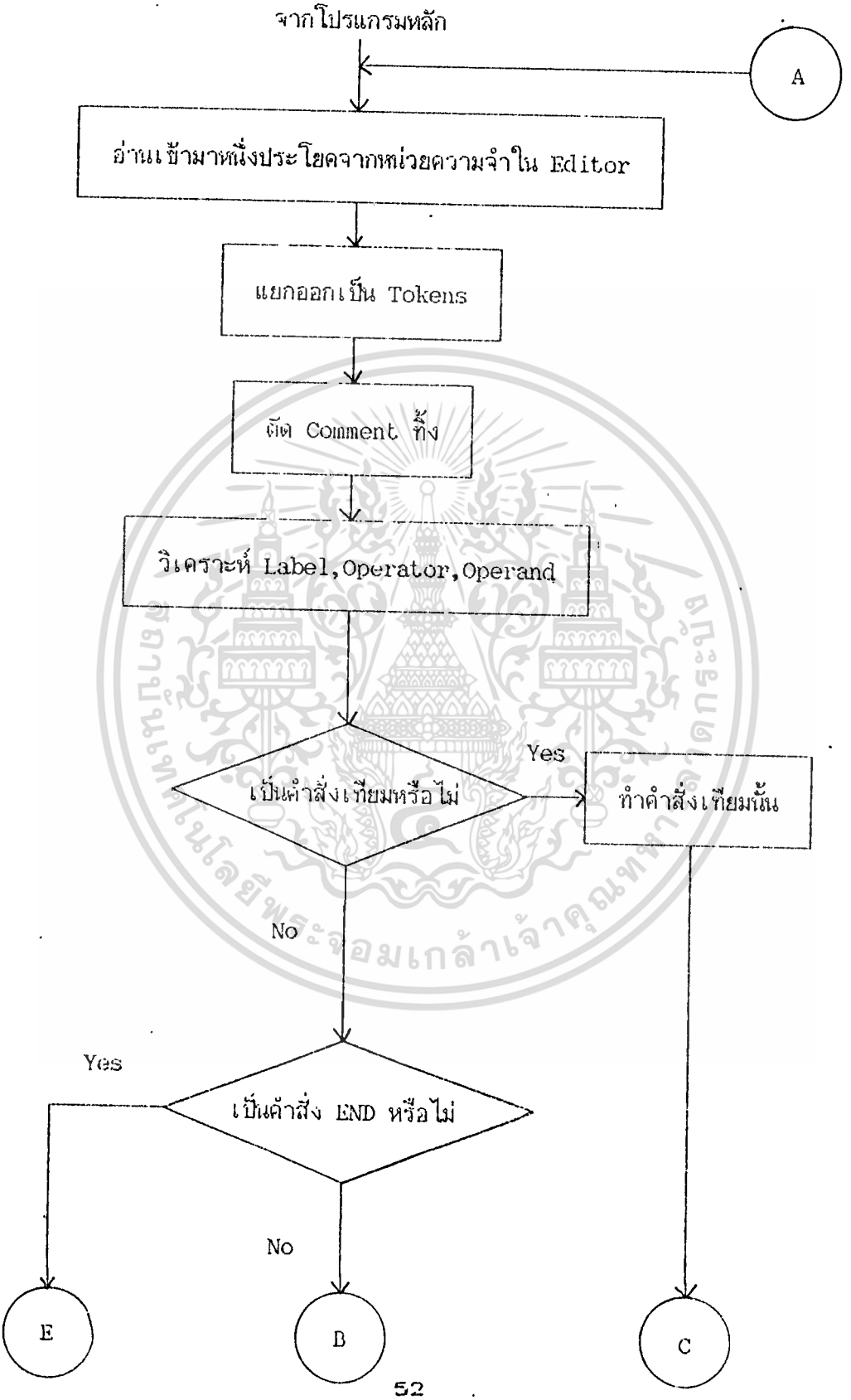
5.1.5 ตัวก่อกำเนิดรหัส (Code Generator)

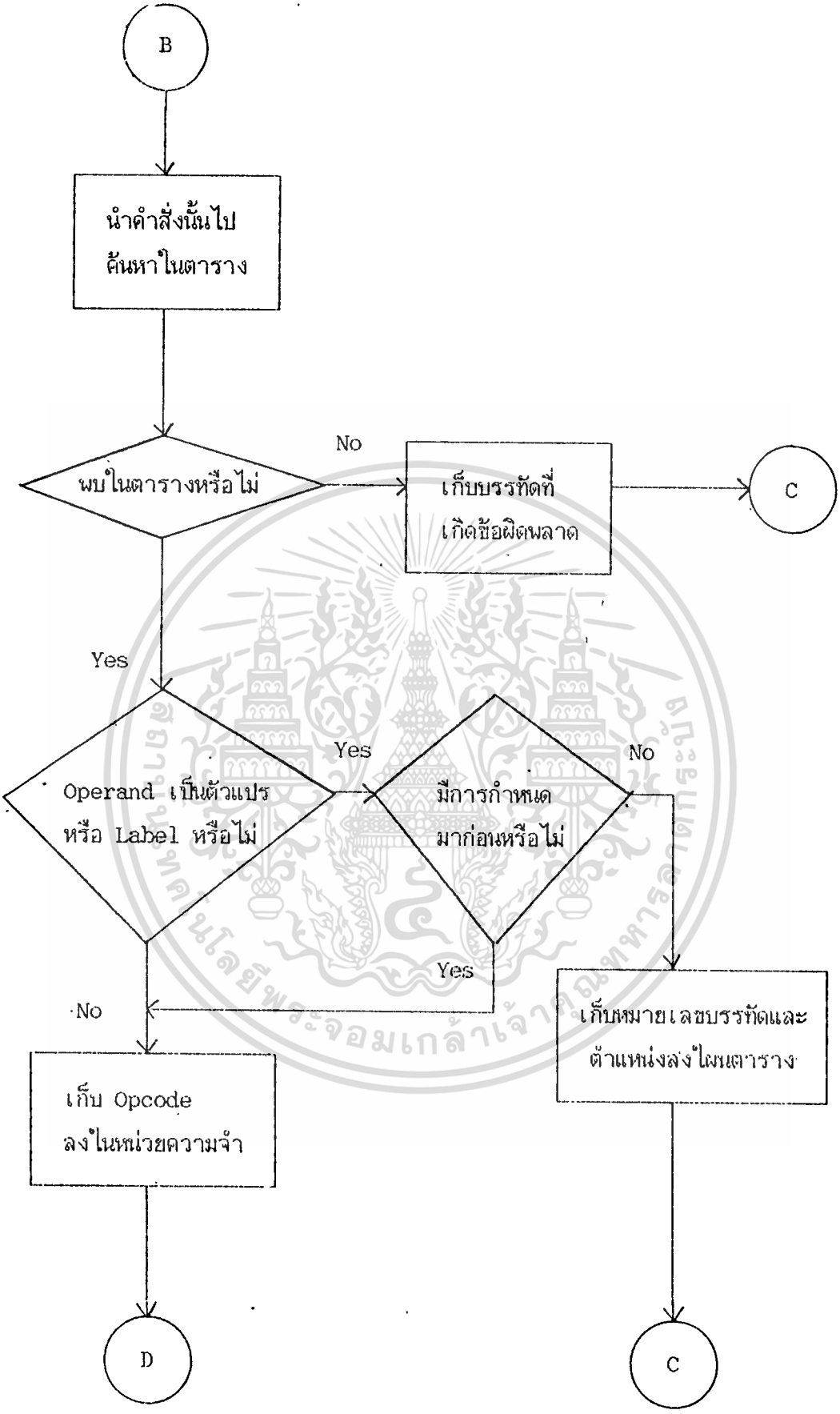
นำรูปแบบชั่วคราวนั้นไปค้นหาในตาราง เพื่อหารหัสดำเนินการ (Op Code) ของคำสั่งนั้นแล้วนำไปเก็บลงในหน่วยความจำ วิธีค้นหาจะใช้การค้นหาแบบทวิภาค (Binary Search)

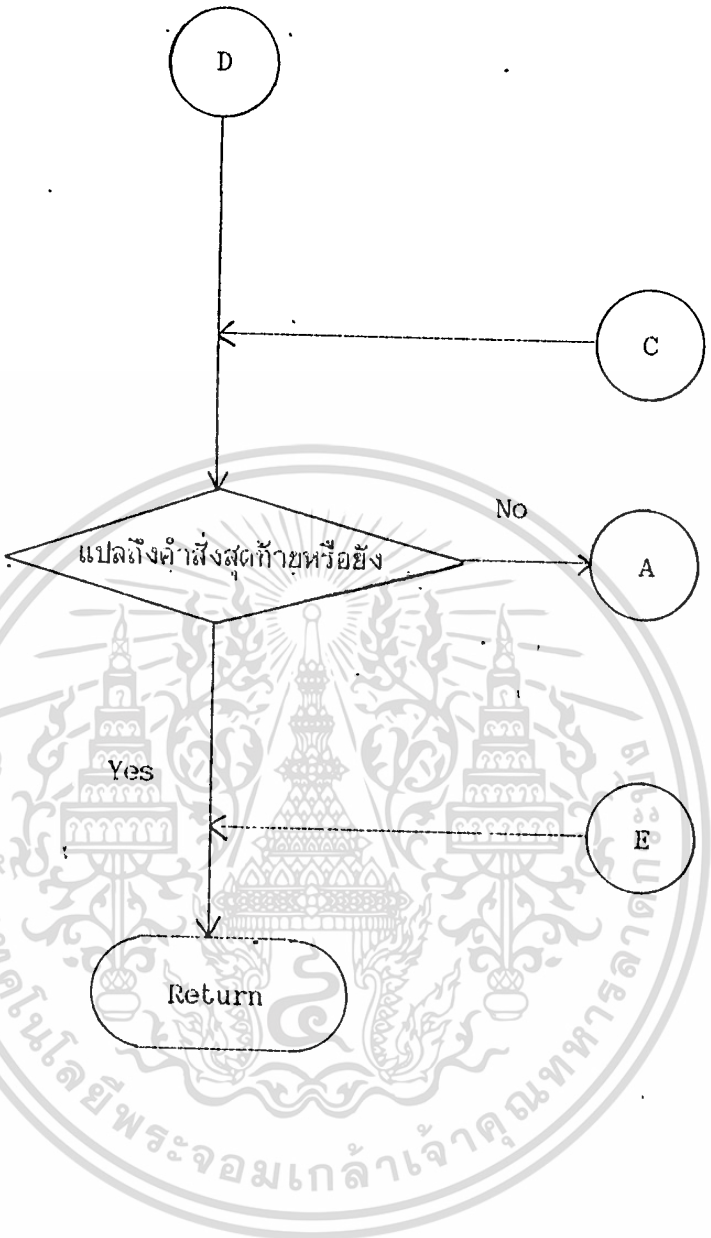
ผังงานแสดงการแอสเซมเบลอร์



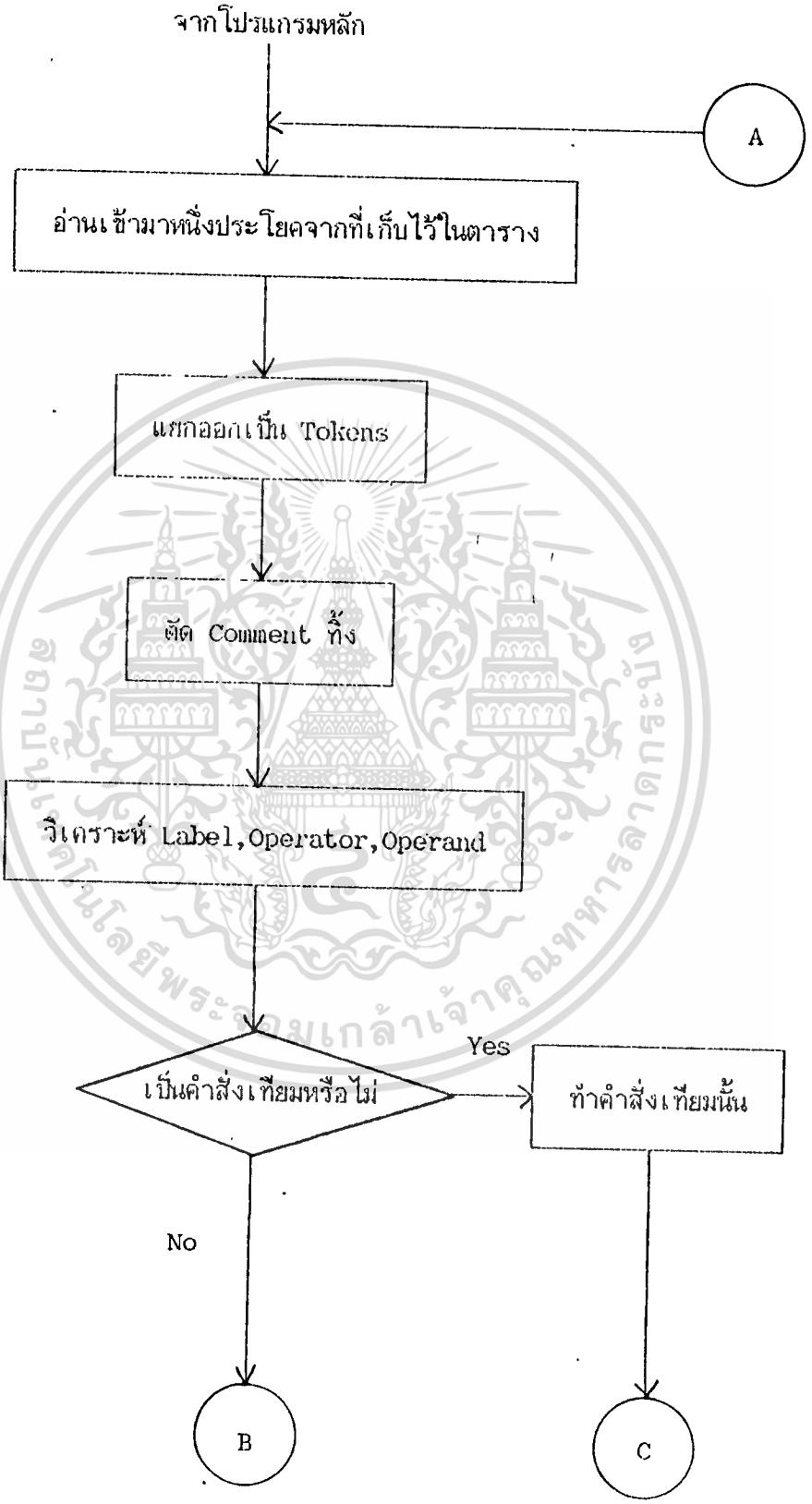
ผังงานแสดงการแอสเซมเบลอร์ Pass 1

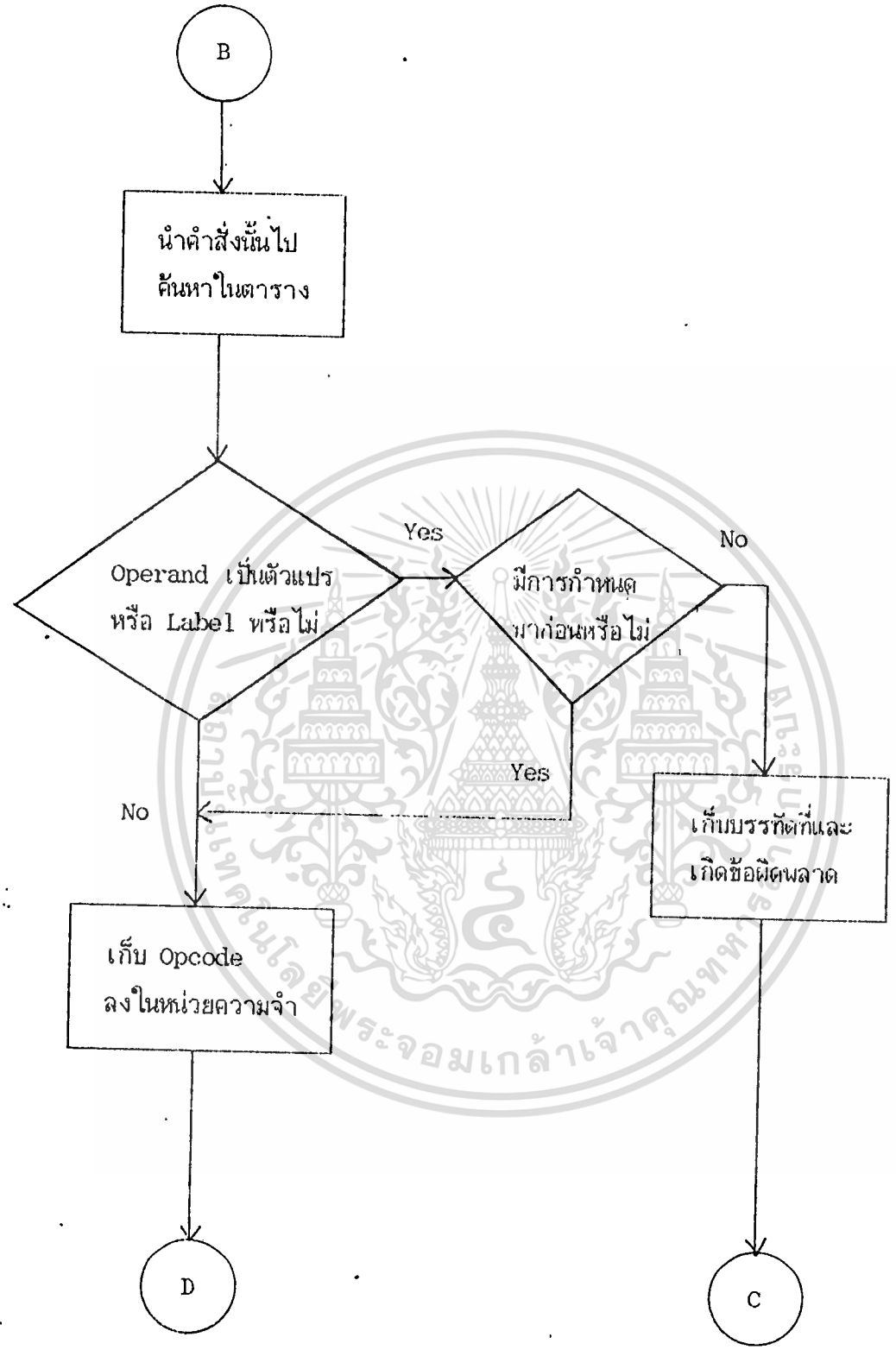


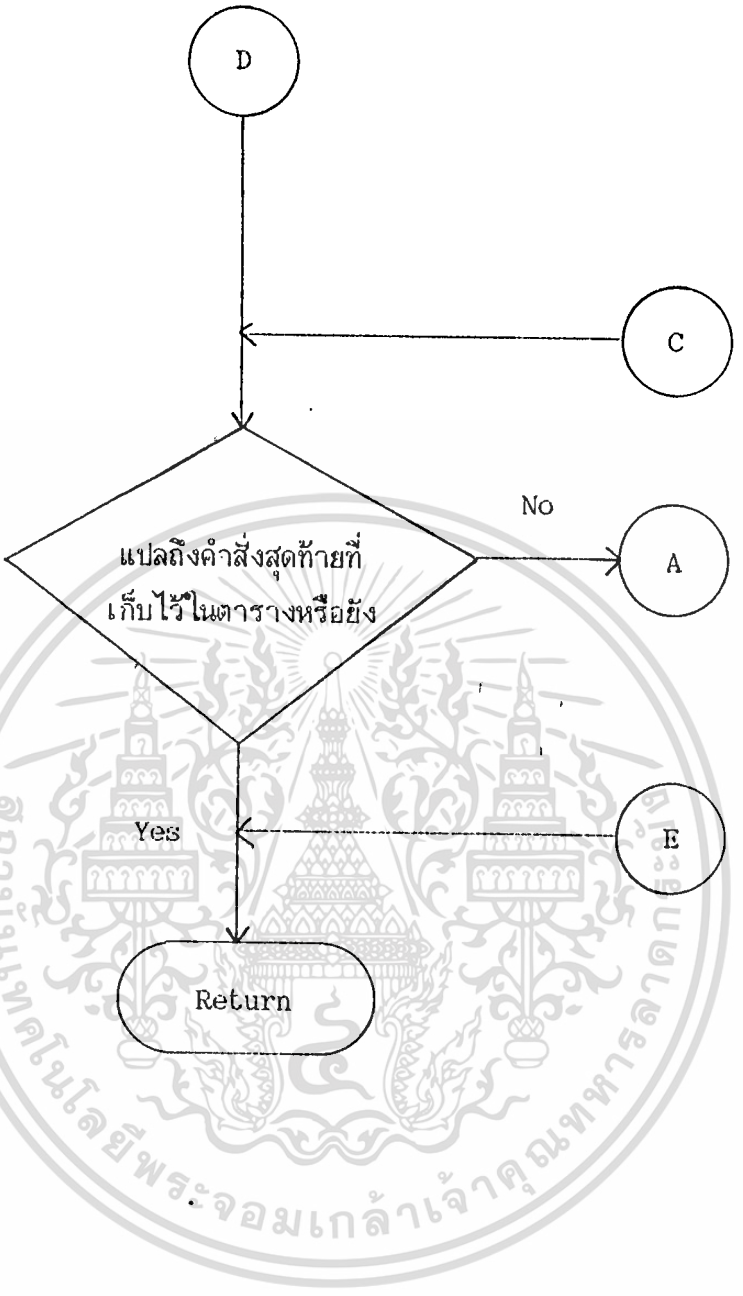




ผังงานแสดงการแอสเซมเบลอร์ Pass 2







5.2 การตรวจสอบการทำงานของโปรแกรม

การตรวจสอบการทำงานของโปรแกรมสามารถทำได้ โดยใช้เครื่องมือที่เรียกว่า ดีบักเกอร์ ซึ่งจะมีคุณสมบัติดังนี้

- ตรวจดูและเปลี่ยนค่าเรจิสเตอร์ต่าง ๆ ได้
- ตรวจดูและเปลี่ยนค่าในหน่วยความจำต่าง ๆ ได้
- ตั้งตำแหน่งที่จะหยุดการทำงาน เพื่อตรวจสอบได้
- สามารถที่หยุดการทำงานขณะที่โปรแกรมของผู้ใช้ทำงานอยู่ได้
- ทำงานทีละคำสั่งได้
- ทำงานทีละโพรซีเดอร์ (Procedure) ได้
- ทำงานรวดเดียวหมดได้
- สามารถทำอันแอสเซมเบลอร์ (Unassembler) เพื่อแปลงรหัสดำเนินการไปเป็นภาษาแอสเซมบลีได้

5.2.1 การติดต่อกับการ์ด Z-80

ตำแหน่งท้ายหน่วยความจำตั้งแต่ FF00H ถึง FFFFH จะสงวนไว้ให้โปรแกรมมอนิเตอร์บนการ์ด Z-80 ใช้เก็บค่าต่าง ๆ ที่จำเป็นที่ใช้ติดต่อกับเครื่องพีซี ดังตาราง

ตารางแสดงตำแหน่ง

ค่าที่เก็บ	ตำแหน่ง
คู่เรจิสเตอร์ AF	FF00H
คู่เรจิสเตอร์ AF'	FF02H
คู่เรจิสเตอร์ BC	FF04H
คู่เรจิสเตอร์ BC'	FF06H
คู่เรจิสเตอร์ DE	FF08H
คู่เรจิสเตอร์ DE'	FF0AH
คู่เรจิสเตอร์ HL	FF0CH
คู่เรจิสเตอร์ HL'	FF0EH

ตารางแสดงตำแหน่ง (ต่อ)

ค่าที่เก็บ	ตำแหน่ง
เรจิสเตอร์ I	FF10H
เรจิสเตอร์ R	FF12H
เรจิสเตอร์ IX	FF14H
เรจิสเตอร์ IY	FF16H
เรจิสเตอร์ SP	FF18H
เรจิสเตอร์ PC	FF2AH
หมายเลข Port ที่จะอ่าน	FF2CH
ค่า Port ที่อ่านได้	FF2DH
หมายเลข Port ที่จะเขียน	FF2EH
ค่า Port ที่จะเขียน	FF2FH
ตำแหน่งที่จะไปของ RST 8H	FF32H
ตำแหน่งที่จะไปของ RST 10H	FF34H
ตำแหน่งที่จะไปของ RST 18H	FF36H
ตำแหน่งที่จะไปของ RST 20H	FF38H
ตำแหน่งที่จะไปของ RST 28H	FF3AH
ตำแหน่งที่จะไปของ RST 30H	FF3CH
ตำแหน่งที่จะไปของ RST 38H	FF3EH
ตำแหน่งที่จะไปของ RST 66H	FF40H
RESERVED	FF42H-FFFFH

5.2.2 การตรวจดูและเปลี่ยนค่าเรจิสเตอร์

เราสามารถที่จะตรวจดูค่าและเปลี่ยนค่าเรจิสเตอร์ได้ โดยดูหรือเปลี่ยนค่าในหน่วยความจำจากตำแหน่งที่กำหนดในตาราง เมื่อส่งคำสั่งให้ทำงาน ตัวโปรแกรมมอนิเตอร์จะทำการอ่านจากค่าจากหน่วยความจำเข้าไปในเรจิสเตอร์ จากนั้นจะโอนการทำงานไปให้โปรแกรมที่จะดำเนินการ

5.2.3 การตรวจดูและเปลี่ยนค่าในหน่วยความจำ

หน่วยความจำเป็นแบบสองทาง คือ การ์ดและเครื่องพีซีสามารถที่จะมองหน่วยความจำนี้ได้ ดังนั้นถ้าต้องการดูหรือแก้ค่าในหน่วยความจำแล้ว ดีบักเกอร์จะส่งคำสั่งไปให้การ์ดเพื่อสับสวิทช์แรมมาให้เครื่องพีซี

การสวิทช์แรมกลับมาจะทำได้ก็ต่อเมื่อการ์ดอยู่ในสภาวะ Inactive คือในขณะที่โปรแกรมของผู้ใช้ไม่ได้ทำงาน โดยในขณะนั้นโปรแกรมมอนิเตอร์กำลังรอคำสั่งอยู่

การส่งคำสั่งเพื่อสวิทช์แรมนั้น ดีบักเกอร์จะทำให้โดยอัตโนมัติทุกครั้งที่โปรแกรมมอนิเตอร์ทำคำสั่งที่ดีบักเกอร์ส่งไปให้เสร็จแล้ว และกำลังรอคำสั่งใหม่อยู่ ผู้ใช้ไม่ต้องทำเอง

5.2.4 การตั้งตำแหน่งที่จะหยุดการทำงาน (Set Break Point)

การตั้งตำแหน่งที่จะหยุดทำงาน จะใช้คำสั่ง RST 30H มาเป็นตัวหยุด โดยดีบักเกอร์จะเก็บคำสั่งเดิมของตำแหน่งนั้นไว้ จากนั้นจะนำคำสั่ง RST 30H เข้าไปแทน หลังจากที่ทำงานมาถึงคำสั่งหยุดนี้แล้ว การทำงานจะถูกโอนไปที่โปรแกรมมอนิเตอร์ เพื่อทำการเก็บค่าเรจิสเตอร์ต่าง ๆ และการ์ดจะกลับมาอยู่ในสภาวะ Inactive เพื่อรอคำสั่งใหม่จากดีบักเกอร์ต่อไป

ดีบักเกอร์เมื่อตรวจสภาวะของการ์ดพบว่าเป็น Inactive แล้ว จะทำการสวิทช์หน่วยความจำมาให้เครื่องพีซี และนำคำสั่งเดิมไปใส่ที่ตำแหน่งที่กำหนดจุดหยุดไว้ จากนั้นก็รอคำสั่งจากผู้ใช้ต่อไป

5.2.5 การหยุดการทำงานขณะที่โปรแกรมของผู้ใช้ทำงานอยู่

ขณะที่โปรแกรมของผู้ใช้ทำงานอยู่ การ์ดจะอยู่ในสภาวะ Active แต่เราสามารถที่จะหยุดการทำงานของโปรแกรมได้ เพื่อตรวจสอบค่าต่าง ๆ โดยดีบักเกอร์ จะ

ส่งคำสั่ง Non-Maskable Interrupt (NMI) ไปให้การ์ด

เมื่อการ์ดรับคำสั่งนี้ การทำงานจะถูกโอนไปให้โปรแกรมมอนิเตอร์ เพื่อเก็บค่าเรจิสเตอร์ต่าง ๆ และการ์ดจะกลับมาอยู่ในสภาวะ Inactive อีกครั้ง ส่วนดีบักเกอร์ก็จะทำงานเช่นเดียวกันการตั้งจุดหยุด

5.2.6 การทำงานทีละคำสั่ง

การทำงานทีละคำสั่งนั้น ดีบักเกอร์จะส่งคำสั่งไปให้การ์ด เพื่อบอกว่าให้การ์ดทำงานทีละคำสั่งเมื่อการ์ดรับคำสั่งนี้แล้วจะคืนค่าเรจิสเตอร์ของโปรแกรมผู้ใช้กลับ จากนั้นจะส่งสัญญาณชิงเกิลสเตป ไปให้ซีพียู (รายละเอียดอยู่ในบทฮาร์ดแวร์) ซึ่งจะมี NMI เกิดขึ้นขณะที่โอนการทำงานไปให้โปรแกรมของผู้ใช้หลังจากที่ทำงานโปรแกรมของผู้ใช้ไปหนึ่งคำสั่ง

เมื่อเกิด NMI ขึ้นแล้ว การทำงานก็จะถูกโอนไปให้โปรแกรมมอนิเตอร์อีกที การทำงานของโปรแกรมมอนิเตอร์และดีบักเกอร์ หลังจากนี้จะเช่นเดียวกันกับการหยุดทำงานของโปรแกรมผู้ใช้ขณะที่ผู้ใช้ทำงานอยู่

5.2.7 การทำงานทีละโพรซีเดอร์ (Procedure)

การทำงานทีละโพรซีเดอร์แตกต่างจากการทำงานทีละคำสั่งตรงที่ว่า เมื่อมาถึงคำสั่ง CALL จะไม่เข้าไปทำในโพรซีเดอร์นั้น แต่จะถือว่าโพรซีเดอร์นั้นเป็นคำสั่ง ๆ เดียว แต่ถ้าเจอคำสั่งอื่น การทำงานก็จะเช่นเดียวกันกับการทำงานทีละคำสั่ง

การทำงานของดีบักเกอร์ เมื่อเจอคำสั่ง CALL นั้นก็เพียงแต่ตั้งจุดหยุดการทำงานไว้ที่คำสั่งหลังคำสั่ง CALL

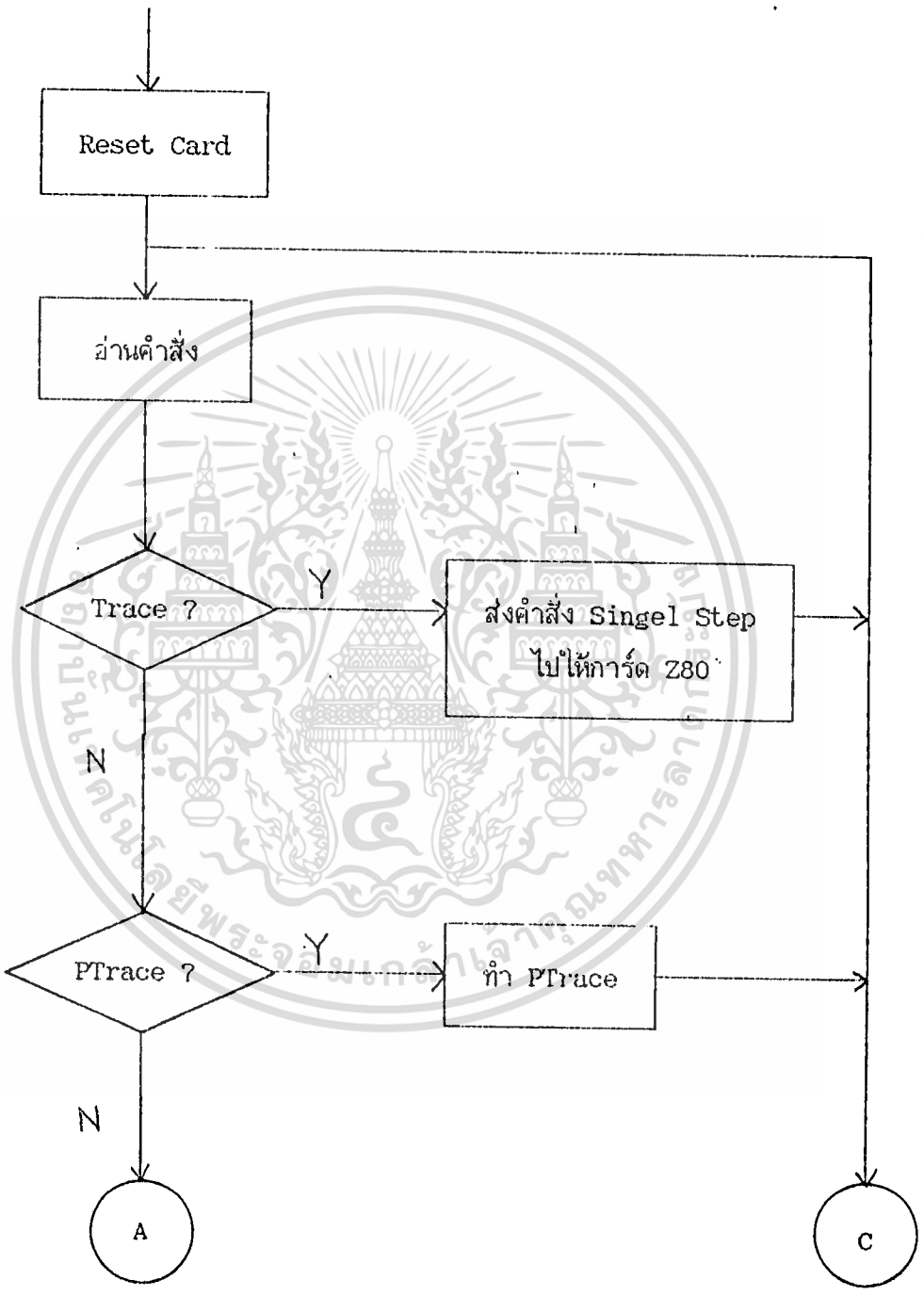
5.2.8 การทำงานรวดเดียวหมด

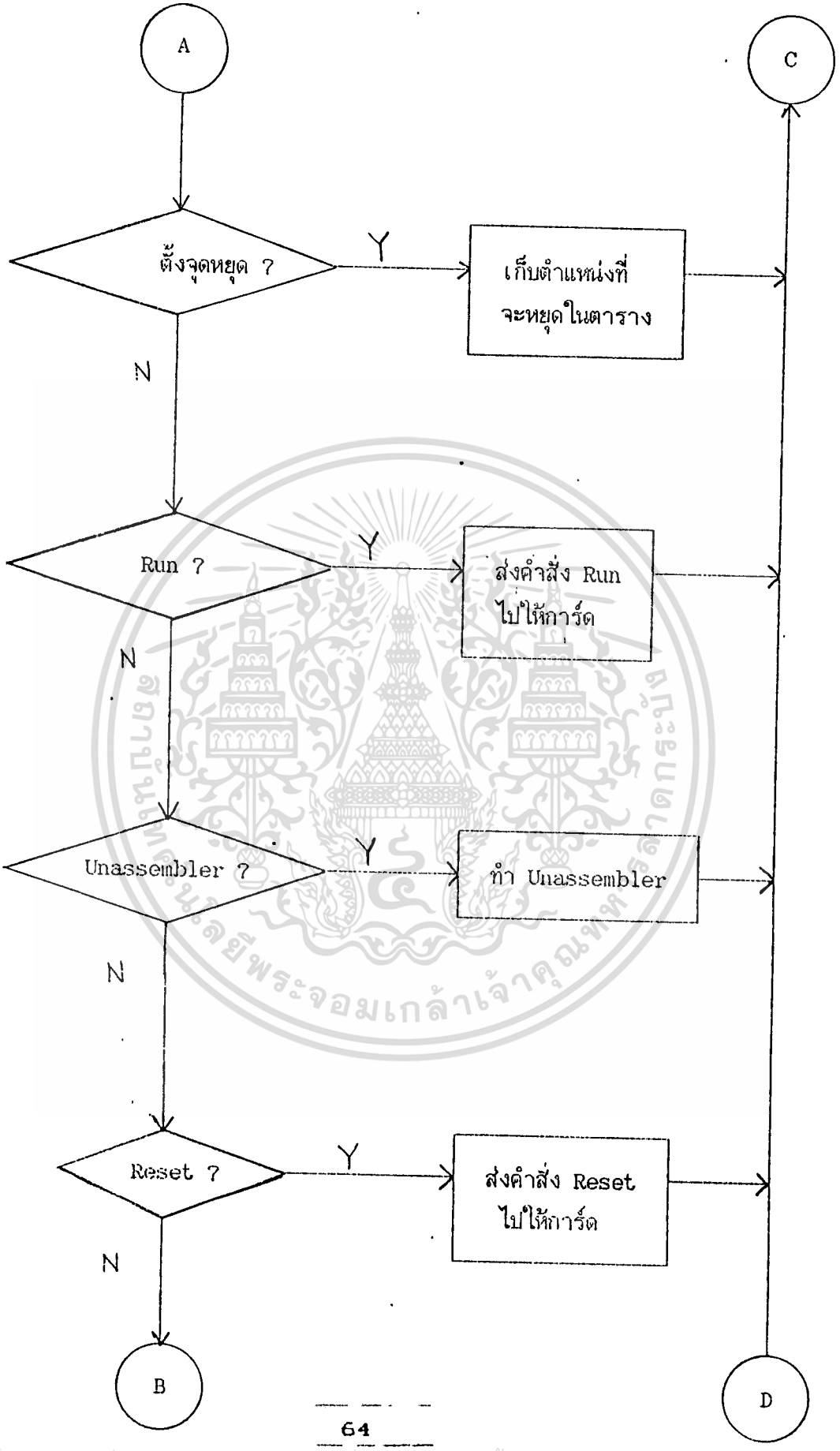
ดีบักเกอร์จะตรวจดูว่าผู้ใช้มีการกำหนดจุดหยุดการทำงานหรือไม่ ถ้ามีก็จะทำการใส่จุดหยุดการทำงานไว้ จากนั้นก็จะส่งคำสั่งไปให้การ์ดเพื่อโอนการทำงานไปให้โปรแกรมผู้ใช้

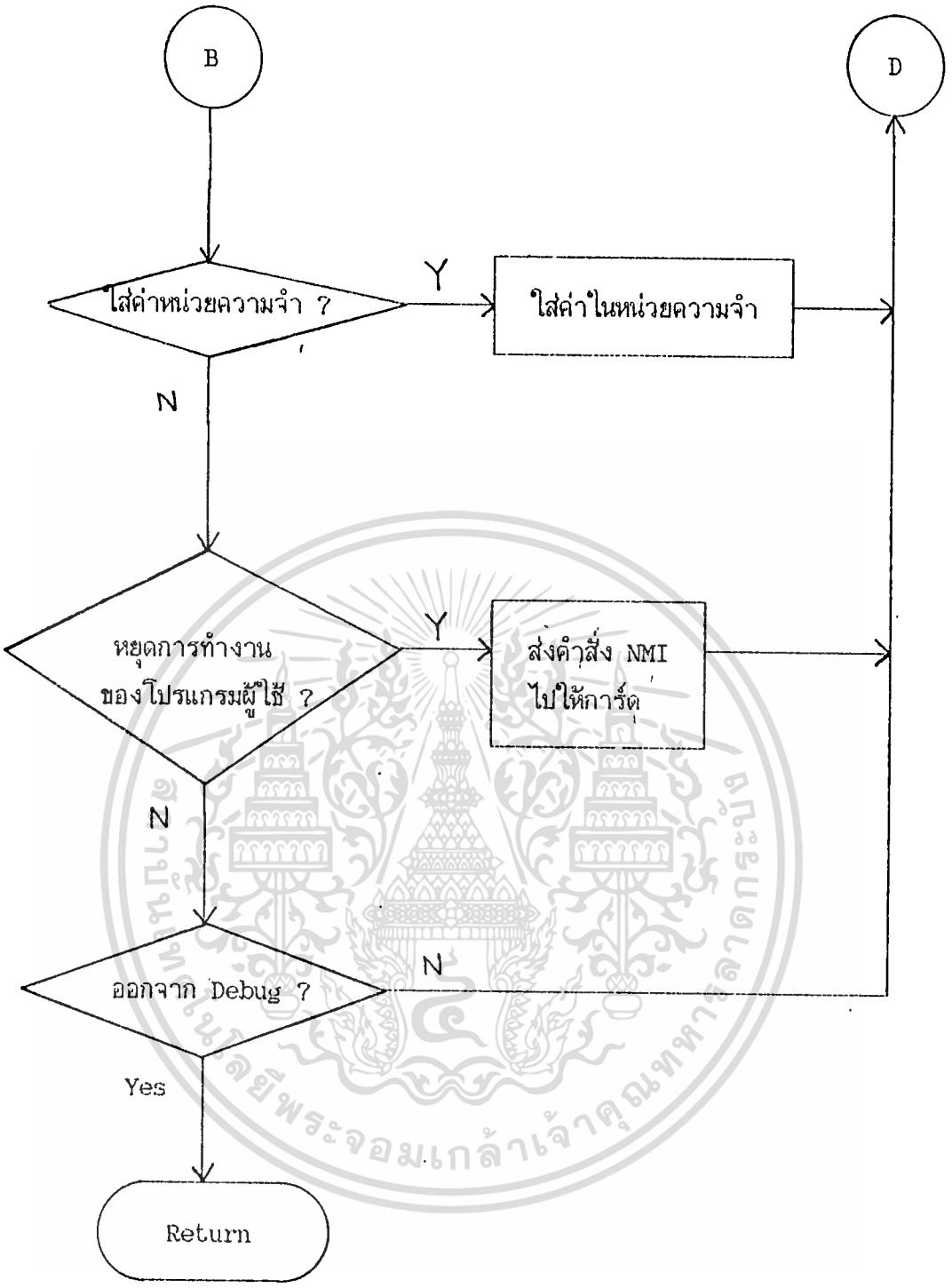
5.2.9 การทำอันแอสเซมเบลอร์ (Unassembler)

จะนำรหัสดำเนินการเข้าไปหาในตาราง เพื่อหาภาษาแอสเซมบลีของรหัสดำเนินการนั้นโดยจะส่งเข้าไปหาทีละห้าไบท์

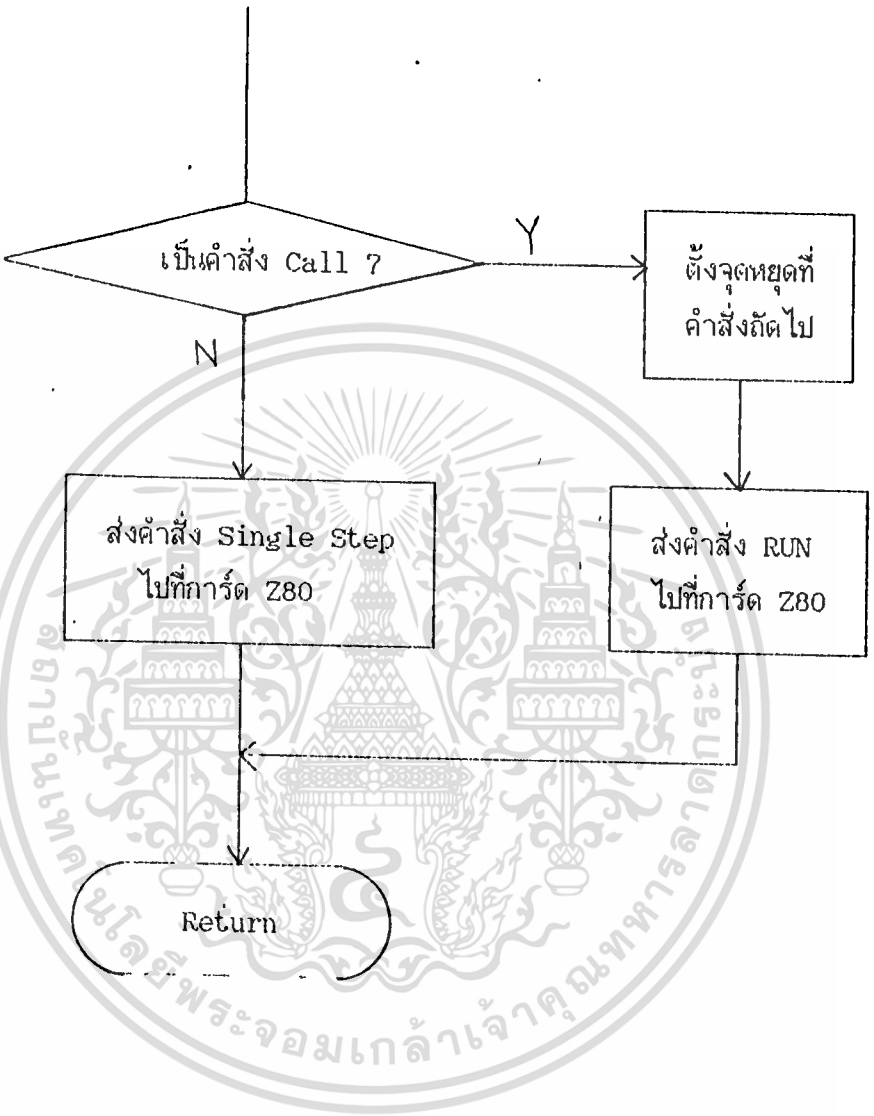
ผังงานแสดงการทำงานของดีบักเกอร์



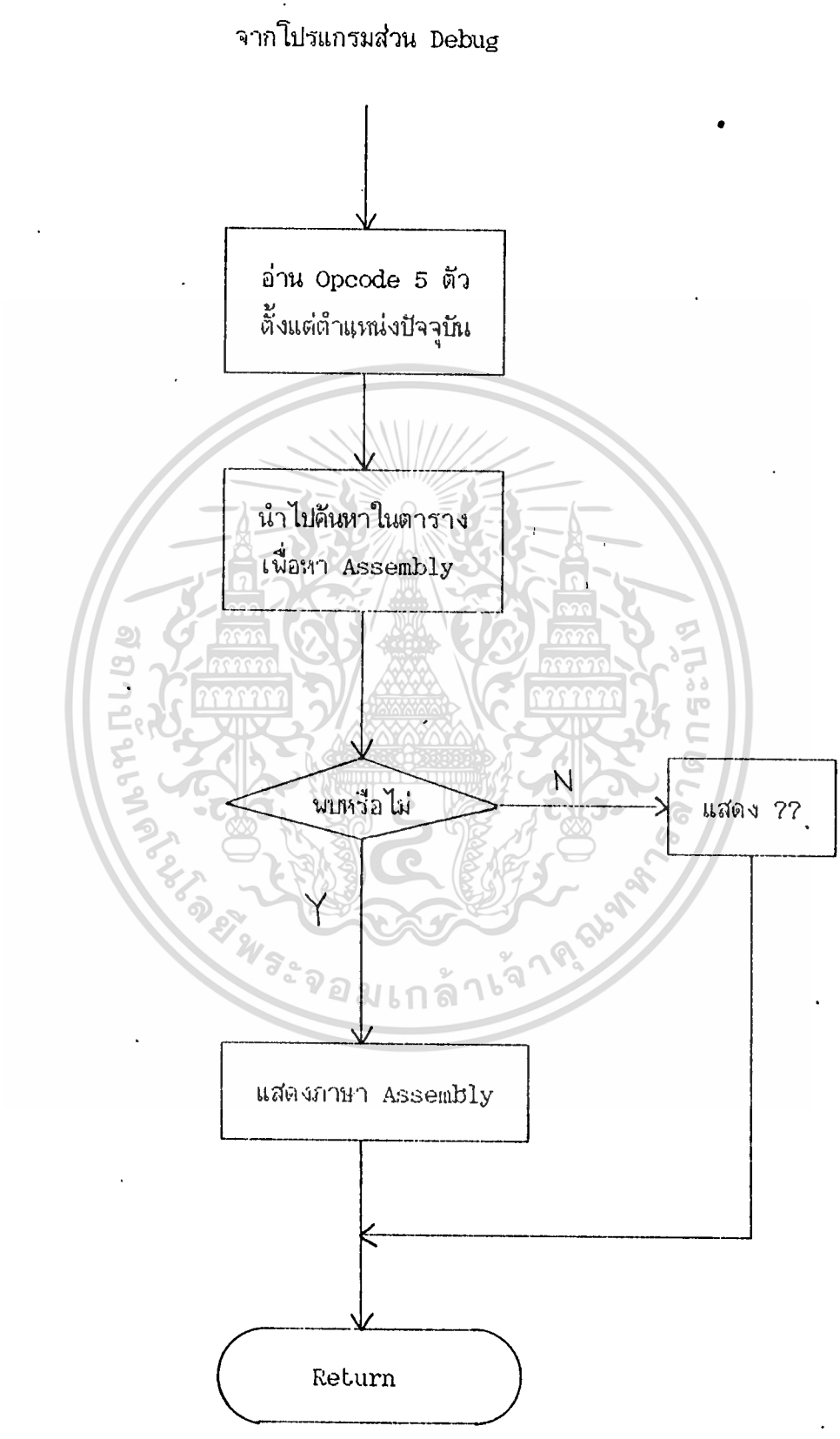




ผังงานแสดงการทำ PTrace (Procedure Trace)



ผังงานแสดงการทำแอสเซมเบลอร์



5.3 อีดิเตอร์ (Editor)

ซอฟต์แวร์ของระบบนี้รวมกันเป็นระบบรวม (Integrated System) ทำให้ผู้ใช้มีความสะดวกในการใช้งาน โปรแกรมอีดิเตอร์ทำหน้าที่เป็นบรรณาธิกรของระบบ ผู้ใช้สามารถสร้างรหัสต้นฉบับขึ้นได้ในรูปของแฟ้มข้อความ โดยไม่ต้องใช้อีดิเตอร์ตัวอื่น โดยที่อีดิเตอร์ที่ใช้ภายในระบบนี้ นอกจากจะทำให้ผู้ใช้มีความสะดวกในการเรียกใช้ ยังมีการทำงานที่สัมพันธ์กันกับแอสเซมเบลอร์ และการตรวจจับข้อผิดพลาด (Error Handling) ทำให้ผู้ใช้มีความสะดวกในการพัฒนาโปรแกรม สามารถแก้ไขข้อผิดพลาดได้อย่างสะดวก

จากที่กล่าวไว้ในบทก่อนหน้านี้ว่าอีดิเตอร์ จะทำการเปลี่ยนแปลงแก้ไขข้อมูลในบริเวณพื้นที่ของรหัสต้นฉบับ โปรแกรมจึงจำเป็นต้องสำรองเนื้อที่ในหน่วยความจำบริเวณนี้เอาไว้ ขนาดของชุดคำสั่งภาษาต้นฉบับที่สามารถสร้างขึ้นได้ ขึ้นอยู่กับขนาดของหน่วยความจำที่สำรองไว้ ในขั้นทดลองทำระบบต้นแบบนี้ ผู้จัดทำได้สำรองหน่วยความจำไว้จำกัด คือ สำรองไว้เพียง 64 กิโลไบต์ ฟังก์ชั่นย่อยของโปรแกรมอีดิเตอร์แสดงได้ดังรูปที่ 5.1

5.3.1 ฟังก์ชั่นย่อยของโปรแกรมอีดิเตอร์

โปรแกรมอีดิเตอร์ ประกอบด้วยฟังก์ชั่นการทำงานย่อยดังต่อไปนี้

- Window Functions

แบ่งจอภาพออกเป็นส่วนๆ โดยการกำหนดขอบเขตของวินโดว์ที่จำกัดขอบเขตของการแสดงผลบนจอภาพ ช่วยให้ฟังก์ชั่นการแสดงผลบนจอภาพ สามารถทำได้โดยไม่เหลื่อมล้ำกันกับส่วนแสดงผลของโปรแกรมส่วนอื่น

- Screen Display Functions

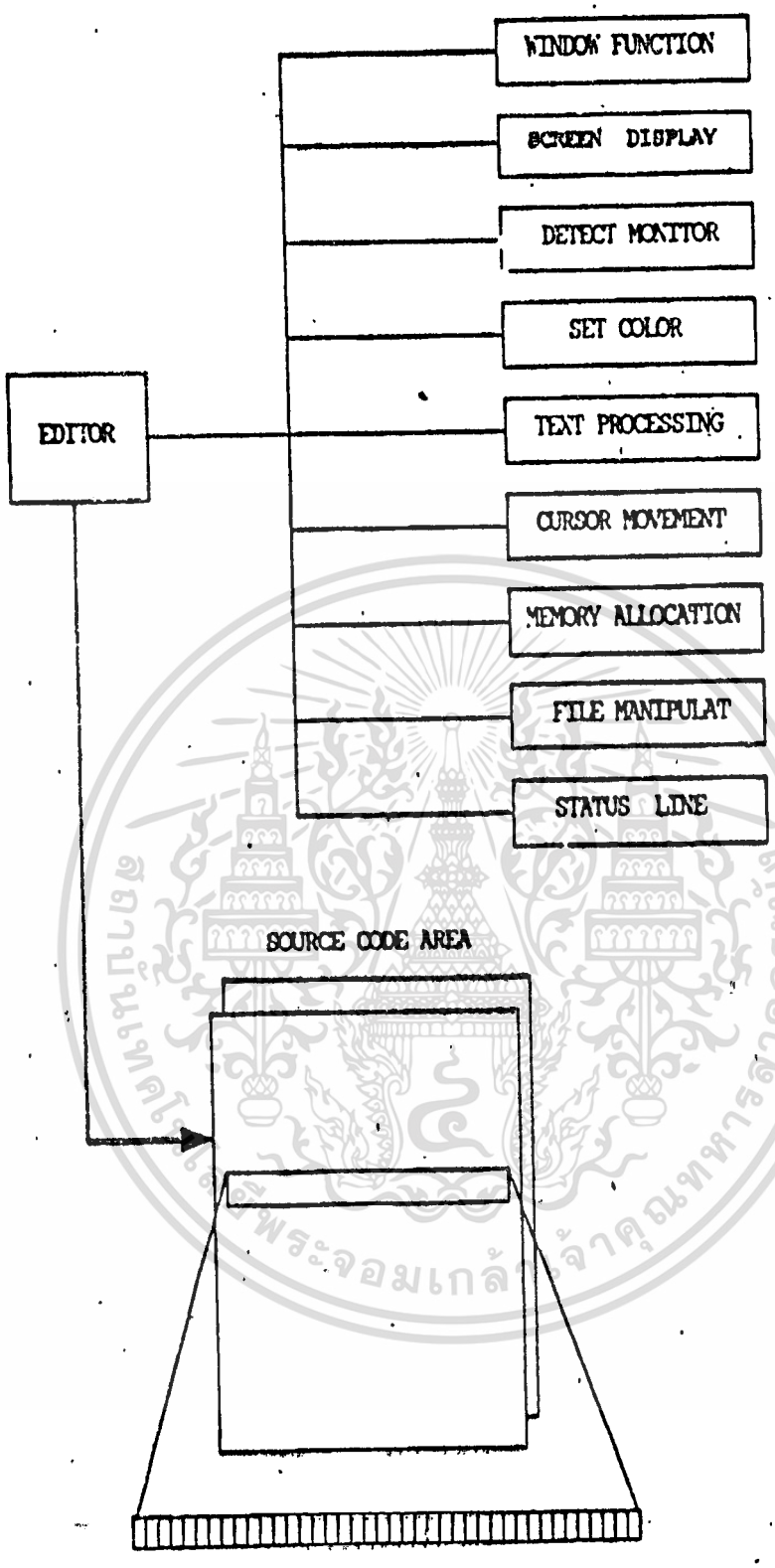
ทำหน้าที่ในการแสดงผลบนจอภาพให้ถูกต้อง และสัมพันธ์กันกับข้อมูลในพื้นที่ของรหัสต้นฉบับ ร่วมทั้งควบคุมลักษณะสีบนหน้าจอ

- Detect Monitor Functions

ตรวจสอบชนิดของการ์ดแสดงผลว่าเป็นชนิดใด (การ์ดโมโนโครมหรือการ์ดสี)

- Set Color

กำหนดสีของตัวอักษรและสีของพื้นหลัง (Buck Ground) ให้กับแต่ละวินโดว์ ในกรณีที่ฟังก์ชันการตรวจสอบจอภาพตรวจสอบพบว่าการ์ดแสดงผลเป็นการ์ดที่ใช้กับจอสี ถ้าหากเป็นการ์ดโมโนโครม จะกำหนดให้มีแค่ 2 สี (ขาว-ดำ)



รูป 5.1 แสดงฟังก์ชันย่อยของโปรแกรมอีดิเตอร์

- Text Processing

ตรวจสอบอินพุตจากแป้นพิมพ์แล้วจำแนกประเภทของคีย์ที่ผู้ใช้กดเข้ามา เพื่อทำการจัดการกับข้อมูล (text) ที่อยู่ในพื้นที่ของรหัสต้นฉบับ อินพุตที่ได้จากการกดคีย์-บอร์ด จะถูกนำไปประมวลผลด้วยวิธีต่าง ๆ กัน ตามประเภทของคีย์ที่กด ซึ่งโปรแกรมได้แบ่งประเภทของคีย์ที่กดเป็นดังนี้

- Normal Character

ตัวอักษรภาษาอังกฤษทั่วไป (a-z, A-Z) รวมทั้งเครื่องหมายพิเศษ (เช่น +, - , * , / , ! , @ เป็นต้น)

- Control Character

เป็นอินพุตที่ได้จากการกดปุ่มคอนโทรล (Ctrl) ควบคู่กับตัวอักษร ซึ่งมีรหัสเป็นคำสั่งต่าง ๆ กัน

ผู้ใช้สามารถควบคุมการใช้งานอีดิเตอร์ในบางส่วนได้โดยปุ่มคอนโทรล รหัสของปุ่มคอนโทรล จะบอกถึงคำสั่งที่ผู้ใช้ต้องการสั่งให้อีดิเตอร์ทำงาน อาทิเช่น

การควบคุมตำแหน่งของเคอร์เซอร์ (Cursor) เช่น

<Ctrl> + 'X' หมายถึง ย้าย เคอร์เซอร์ ลง 1 ตำแหน่ง

<Ctrl> + 'E' หมายถึง ย้าย เคอร์เซอร์ ขึ้น 1 ตำแหน่ง

<Ctrl> + 'S' หมายถึง ย้าย เคอร์เซอร์ ไปทางซ้าย 1 ตำแหน่ง

<Ctrl> + 'D' หมายถึง ย้าย เคอร์เซอร์ ไปทางขวา 1 ตำแหน่ง

<Ctrl> + 'R' หมายถึง ย้าย เคอร์เซอร์ ขึ้น 1 หน้า

<Ctrl> + 'C' หมายถึง ย้าย เคอร์เซอร์ ลง 1 หน้า

<Ctrl> + 'A' หมายถึง ย้าย เคอร์เซอร์ ไปทางซ้าย 1 คำ

<Ctrl> + 'F' หมายถึง ย้าย เคอร์เซอร์ ไปทางขวา 1 คำ

การควบคุมข้อความเป็นบล๊อค

<Ctrl> + 'K' + 'B' หมายถึง กำหนดตำแหน่งเริ่มต้นของบล๊อค

<Ctrl> + 'K' + 'K' หมายถึง กำหนดตำแหน่งสิ้นสุดของบล๊อค

<Ctrl> + 'K' + 'C' หมายถึง คัดลอกข้อความในบล๊อค

<Ctrl> + 'K' + 'V' หมายถึง ย้ายตำแหน่งของบล๊อค

<Ctrl> + 'K' + 'Y' หมายถึง ลบข้อความทั้งหมดในบล๊อค

การลบตัวอักษร ลบคำ ลบบรรทัด

<Ctrl> + 'G' หมายถึง ลบตัวอักษร

<Ctrl> + 'T' หมายถึง ลบคำ

<Ctrl> + 'Y' หมายถึง ลบทั้งบรรทัด

- Cursor Control Keys

เป็นปุ่มที่มีความหมายในการควบคุมตำแหน่งของ เคอร์เซอร์ โดยตรง

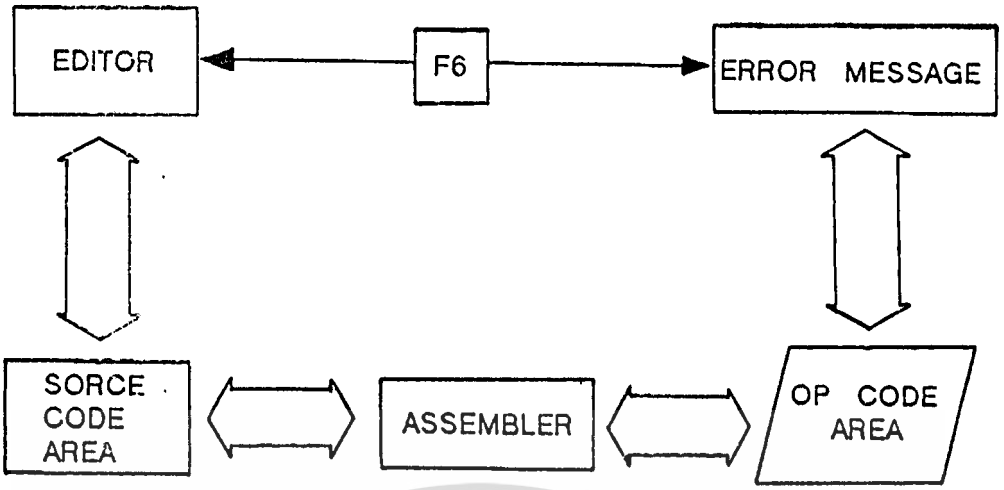
↓	หมายถึง ย้าย เคอร์เซอร์ ลง 1 ตำแหน่ง
↑	หมายถึง ย้าย เคอร์เซอร์ ขึ้น 1 ตำแหน่ง
←	หมายถึง ย้าย เคอร์เซอร์ ไปทางซ้าย 1 ตำแหน่ง
→	หมายถึง ย้าย เคอร์เซอร์ ไปทางขวา 1 ตำแหน่ง
<PgUp>	หมายถึง ย้าย เคอร์เซอร์ ขึ้น 1 หน้า
<PgDn>	หมายถึง ย้าย เคอร์เซอร์ ลง 1 หน้า
<Home>	หมายถึง ย้าย เคอร์เซอร์ ไปต้นบรรทัด
<End>	หมายถึง ย้าย เคอร์เซอร์ ไปท้ายบรรทัด

เมื่อมีอินพุตเข้ามาจัดอยู่ในกลุ่ม Cursor Control Key โปรแกรมจะเรียกใช้ฟังก์ชั่นการเลื่อนเคอร์เซอร์ เพื่อจัดการวางตำแหน่งที่ถูกต้องให้กับเคอร์เซอร์ แสดงภาพบนจอภาพให้สัมพันธ์กับตำแหน่งเคอร์เซอร์ และกำหนดตำแหน่งของข้อมูลที่จะทำการแก้ไขในพื้นที่ของรหัสต้นฉบับ

- Function Keys

จะเป็นกลุ่มของฟังก์ชันคีย์ (Function Keys) ได้แก่ F1-F12 ซึ่งใช้สำหรับแทนคำสั่งพิเศษที่ช่วยทำให้อีดิเตอร์มีความคล่องตัวในการใช้งานมากขึ้น หรือเป็นคำสั่งย้ายการทำงานไปยังส่วนอื่น ๆ ของซอฟต์แวร์ เช่น F6 จะเป็นปุ่มย้ายการทำงานของโปรแกรมระหว่างอีดิเตอร์ และข้อความแสดงข้อผิดพลาด (Error Message) เมื่อผู้ใช้อยู่ในวินโดว์ของอีดิเตอร์ จะสามารถดูข้อความแสดงข้อผิดพลาดที่ได้จากการแอสเซมเบลอร์ได้โดยกดปุ่ม F6 หรือเมื่อกำลังอยู่ในวินโดว์ของข้อความแสดงข้อผิดพลาดก็สามารถกลับมาสู่อีดิเตอร์โดยการกด F6 เช่นกัน

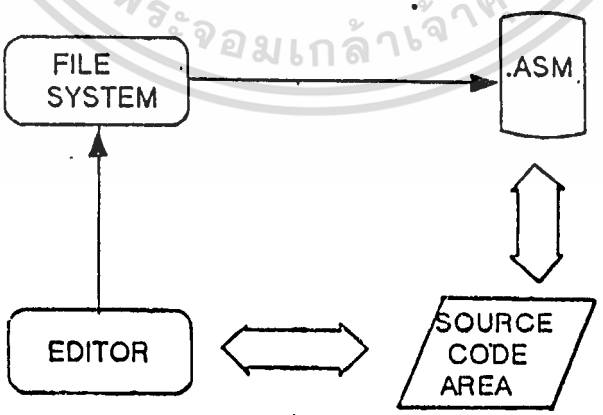
รูปที่ 5.2 แสดงการทำงานของปุ่ม F6 โดย จะเป็นปุ่มย้ายการทำงานของโปรแกรมระหว่างอีดิเตอร์ และข้อความแสดงข้อผิดพลาด



รูปที่ 5.2 แสดงการทำงานของปุ่ม F6

- File Handling

อีดิเตอร์จะสามารถทำงานกับแฟ้มข้อมูลต้นฉบับ และดิสค์ได้โดยไม่จำเป็นต้องออกไปที่รายการหลักแล้วจึงเรียกระบบแฟ้มข้อมูล (File System) โปรแกรมอีดิเตอร์สามารถเรียกใช้ฟังก์ชั่นบางตัวของระบบแฟ้มข้อมูลมาใช้ได้ เพื่อความคล่องตัวในการพัฒนาโปรแกรม เช่น ฟังก์ชั่น อ่านแฟ้มข้อมูล (Load File), เก็บแฟ้มข้อมูล (Save File) ซึ่งการทำงานสามารถแสดงได้ดังรูปที่ 5.3



รูปที่ 5.3 แสดงการทำงานของระบบไฟล์

- Status Line

บรรทัดล่างสุดของจอภาพจะเป็น บรรทัดที่บอกถึงรายละเอียดและสถานะบางอย่างของอีดิเตอร์ เช่น

- บรรทัดปัจจุบัน (Current Line)
- คอลัมน์ปัจจุบัน (Current Column)
- สถานะของปุ่มแทรก (Insert Key)
- ชื่อแฟ้มข้อมูลที่กำลังทำการแก้ไขอยู่

- Memory Allocation

โปรแกรมในส่วนของอีดิเตอร์ (รวมทั้งแอสเซมเบลอร์) มีความจำเป็นที่ต้องสำรองเนื้อที่ในหน่วยความจำ เพื่อไว้ใช้เก็บข้อมูล และมิให้โปรแกรมส่วนอื่นมาใช้เนื้อที่ในบริเวณดังกล่าว

การจองหน่วยความจำนี้ จะทำหน้าที่สำรองเนื้อที่ในหน่วยความจำเป็นบล๊อคที่ตำแหน่งต่าง ๆ กัน และตรวจดูว่ามีเนื้อที่สำรองเหลือพอหรือไม่ ถ้าหากมีหน่วยความจำที่ว่างเหลืออยู่ไม่พอ โปรแกรมก็ไม่อาจจะทำงานได้

เนื้อที่ในหน่วยความจำที่ถูกสำรองไว้ จะถูกใช้เป็นที่เก็บข้อมูลต่าง ๆ ดังนี้

- พื้นที่ของรหัสต้นฉบับ 64 กิโลไบต์
- พื้นที่ของรหัสดำเนินการ 64 กิโลไบต์
- ตารางสัญลักษณ์ที่ใช้ในแอสเซมเบลอร์ 64 กิโลไบต์
- พื้นที่สำหรับเก็บข้อความแสดงข้อผิดพลาด 2 กิโลไบต์

บทที่ 6
ขั้นตอนการพัฒนาและผลการทดลอง

6.1 ขั้นตอนการพัฒนาฮาร์ดแวร์

มีขั้นตอนการพัฒนาดังนี้

1. ออกแบบวงจรฮาร์ดแวร์ทั้งหมด
2. ลงแผ่นวงจรโดยการทำ Wire Wrapped
3. ทำการทดสอบในส่วนของหน่วยความจำ โดยเขียนโปรแกรมทดสอบให้ทำการอ่านและเขียนหน่วยความจำ ทั้งทางการ์ดและเครื่องพีซี
4. ทำการทดสอบในส่วนของพอร์ท โดยเขียนโปรแกรมทดสอบให้ทำการเขียนและอ่านพอร์ท
5. ต่อวงจรภายนอกเพื่อทดสอบการทำงานทั้งหมด

6.2 ขั้นตอนการพัฒนาซอฟต์แวร์

มีขั้นตอนดังต่อไปนี้

1. ออกแบบและเขียนโปรแกรมอีดิเตอร์ และแอสเซมเบลอร์ไปพร้อม ๆ กัน
2. ทำการเชื่อมโปรแกรมอีดิเตอร์ และแอสเซมเบลอร์เข้าด้วยกัน
2. ออกแบบและเขียนโปรแกรมมอนิเตอร์ โดยใช้โปรแกรมอีดิเตอร์และแอสเซมเบลอร์ที่เขียนขึ้นมาแล้ว มาช่วยในการพัฒนา
3. ออกแบบและเขียนโปรแกรมดีบักเกอร์
4. ทดสอบการทำงานร่วมกันระหว่างดีบักเกอร์ กับโปรแกรมมอนิเตอร์
5. เชื่อมการทำงานระหว่างระบบรวม กับดีบักเกอร์

6.3 สรุปการพัฒนา

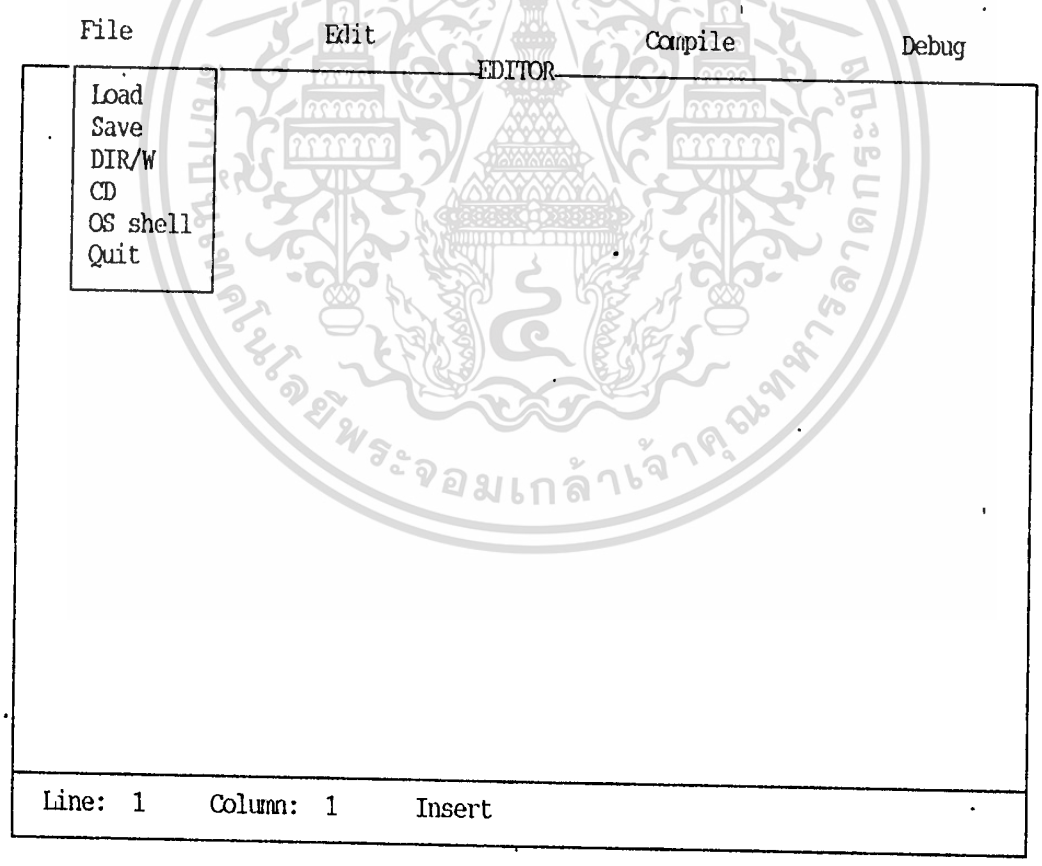
การออกแบบ พัฒนา และทดลองระบบในโครงงานนี้ได้ใช้เทคนิคแบบการพัฒนาโดยตัวเอง (Self Development) กล่าวคือจะทำการพัฒนาส่วนต่อไปโดยอาศัยส่วนที่พัฒนามาก่อน คือเริ่มแรกของโครงงานนี้ จะทำการพัฒนาระบบของหน่วยความจำขึ้นก่อนจากนั้นเมื่อส่วนนี้สามารถทำงานได้ก็จะทดลองในส่วนของพอร์ทโดยใช้ส่วนของหน่วยความจำ

ส่วนทางด้านซอฟต์แวร์นั้น การเขียนโปรแกรมอีดิเตอร์ แอสเซมเบลอร์ ดีบักเกอร์ ตลอดจนโปรแกรมมอนิเตอร์จะใช้ระบบซอฟท์แวร์ที่พัฒนาขึ้นเองทั้งหมด และประโยชน์ที่ได้จากการใช้เทคนิคนี้ คือ ทำให้สามารถพบข้อผิดพลาดของส่วนต่าง ๆ ได้ เพราะได้นำมาใช้งานจริงโดยตรงจากส่วนต่อไปที่พัฒนาต่อจากส่วนก่อน

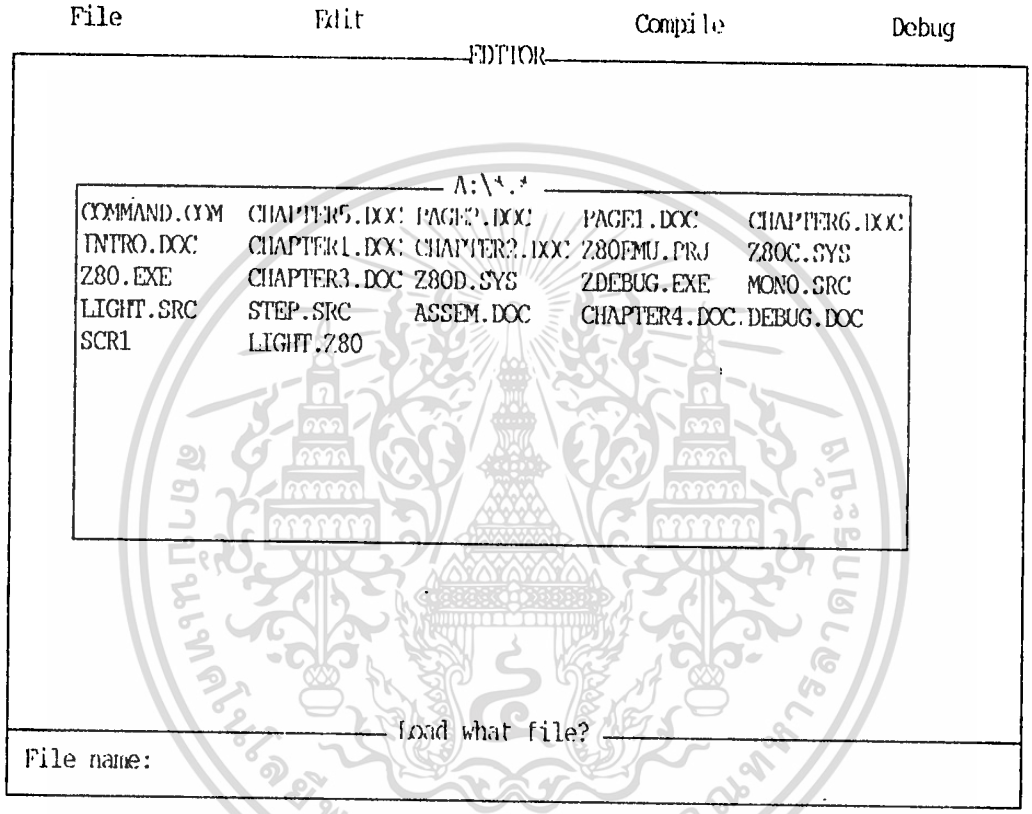
6.4 ผลการทดลอง

ผลการทดลองสามารถแสดงได้เป็นลำดับดังนี้

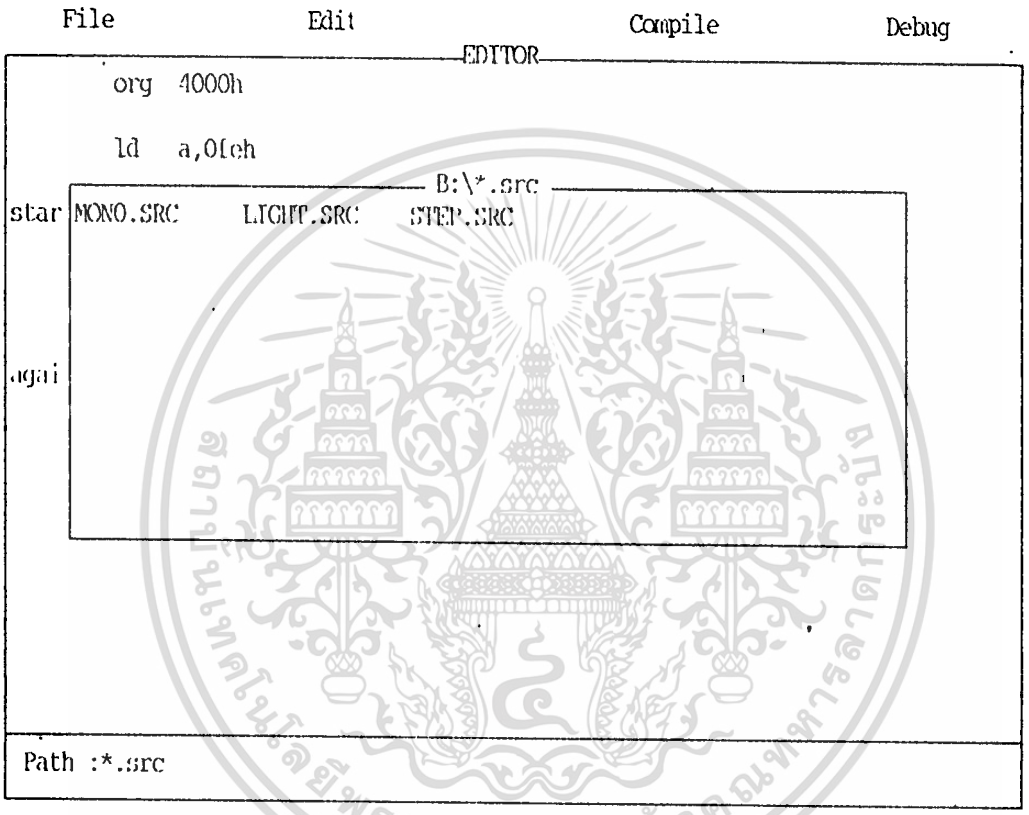
เมื่อเริ่มเข้าสู่โปรแกรม

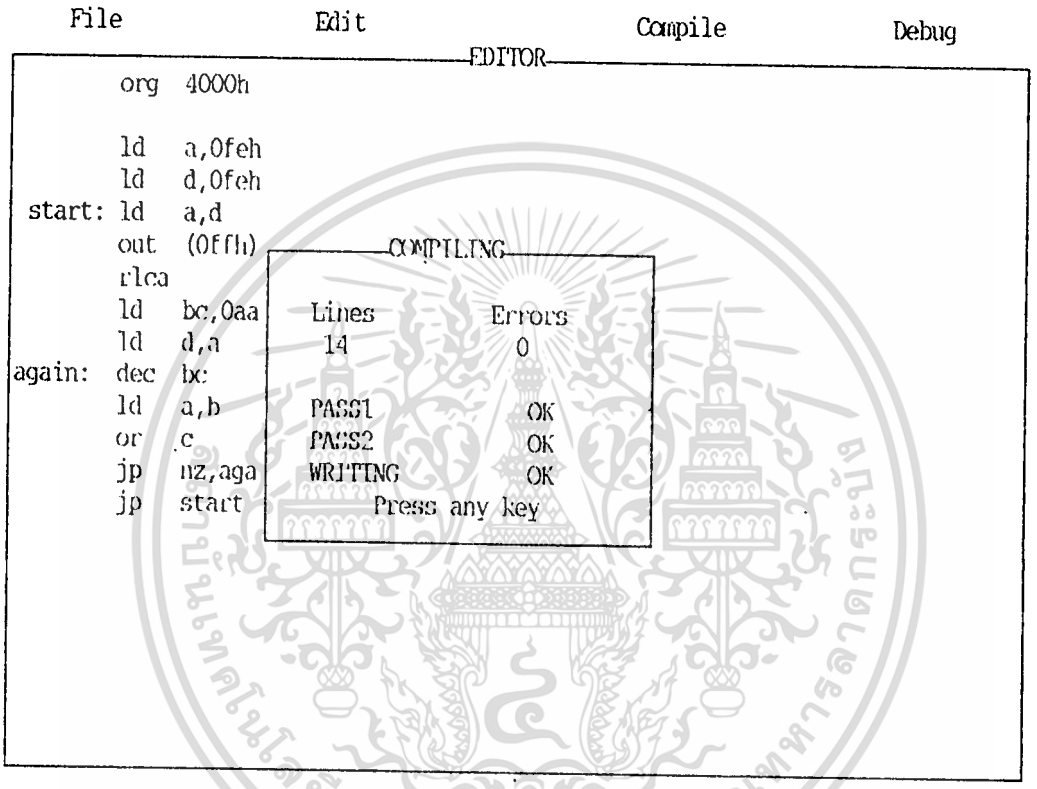


เมื่อเลือกรายการอ่านแฟ้มข้อมูล (Load)

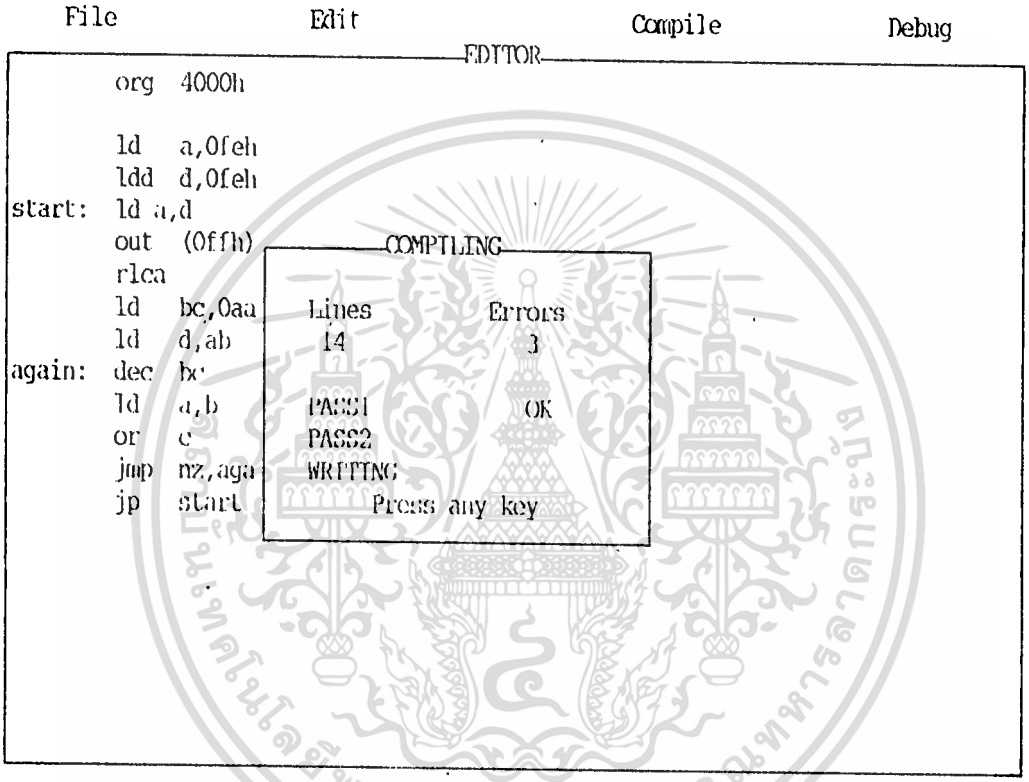


การทำงานของรายการแสดงสารบบ (Directory)

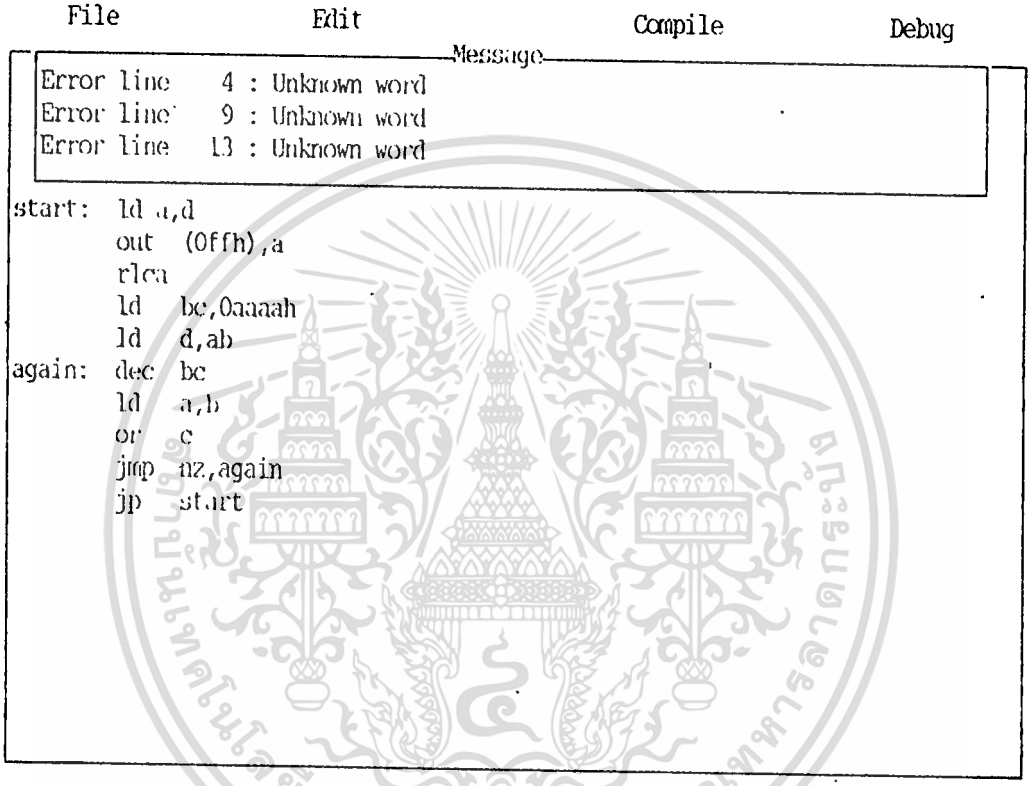




การแอสเซมเบลอร์รหัสต้นฉบับที่มีข้อผิดพลาด



ข้อความแสดงข้อผิดพลาด



ความช่วยเหลือของโปรแกรมดีบักเกอร์

```
Z80 Debugger version 1.0
Copyright (C) 1990
Written by . . . Tom Tantisalidchai
Computer Engineering of KMITL

Loading debug table . . .

Type ? for HELP
-?
?    Help          I    Inport
L  - Load          O    Outport
W  - Write         BP   Breakpoint Set
U    Unassembler   BL   Breakpoint List
E    Enter         BC   Breakpoint Clear
D    Dump          CR   Card Ram
F    Fill          PR   PCRam
R  - Register      RS   Reset card
T  - Trace         TR   Test Ram
P  - PTrace        IZ - Inport Z80
G  - Go            OZ - Outport Z80
S  - Stop          PL - PC Look ram
Q  - Quit          ZL - Z80 Look ram
-
```

การทำงานทีละคำสั่ง

```
-r
STATUS : Inactive
AF =0000  BC =0000  DE =0000  HL =0000  IX=0000  IY=0000  SP=FEFF
AF'=0000  BC'=0000  DE'=0000  HL'=0000  I =00    R =00    PC=4000
4000   3E FE                  LD        A,FE
-t
STATUS : Inactive
AF =FE00  BC =0000  DE =0000  HL =0000  IX=0000  IY=0000  SP=FEFF
AF'=0000  BC'=0000  DE'=0000  HL'=0000  I =00    R =35    PC=4002
4002   16 FE                  LD        D,FE
-t
STATUS : Inactive
AF =FE00  BC =0000  DE =FE00  HL =0000  IX=0000  IY=0000  SP=FEFF
AF'=0000  BC'=0000  DE'=0000  HL'=0000  I =00    R =6A    PC=4004
4004   7A                     LD        A,D
-t
STATUS : Inactive
AF =FE00  BC =0000  DE =FE00  HL =0000  IX=0000  IY=0000  SP=FEFF
AF'=0000  BC'=0000  DE'=0000  HL'=0000  I =00    R =1F    PC=4005
4005   D3 FF                  OUT       (FF),A
-t
STATUS : Inactive
AF =FE00  BC =0000  DE =FE00  HL =0000  IX=0000  IY=0000  SP=FEFF
AF'=0000  BC'=0000  DE'=0000  HL'=0000  I =00    R =54    PC=4007
4007   07                     RLCA
-
```

การทำงานของคำสั่งอันแอสเซมเบลอร์ (Unassembler)

```
-u 4000
4000   3E FE        LD      A,FE
4002   16 FE        LD      D,FE
4004   7A           LD      A,D
4005   D3 FF        OUT     (FF),A
4007   07           RLCA
4008   01 AA AA     LD      BC,AAAA
400B   57           LD      D,A
400C   0B           DEC     BC
400D   78           LD      A,B
400E   B1           OR      C
-u
400F   C2 0C 40     JP      NZ,(400C)
4012   C3 04 40     JP      (4004)
4015   DF           RST     18
4016   FF           RST     38
4017   FD           ??
4018   FF           RST     38
4019   FF           RST     38
401A   FF           RST     38
401B   7F           LD      A,A
401C   FF           RST     38
-
```

การทำงานของคำสั่งดัมพ์ (Dump)

```
-d 4000
4000    3E FE 16 FE 7A D3 FF 07 01 AA AA 57 0B 78 B1 C2    ....z......W.x..
4010    0C 40 C3 04 40 DF FF FD FF FF FF 7F FF FF FF FF    ................
4020    00 00 00 00 21 00 00 A0 00 00 05 00 01 40 80 00    ................
4030    82 00 00 02 18 00 00 08 00 50 00 00 00 88 80 00    .........P......
4040    00 00 00 21 05 00 00 00 00 00 00 00 00 40 00 00    ................
4050    10 08 00 00 00 00 00 00 08 02 46 00 00 22 00 00    ..........F.....
-d
4060    BA FF FF FF FF FF FF FF FF FF FF FE FF FF FF FE    ................
4070    FF 7F FF FF FF FF FF FF FF FF FF BF FF FF FF FF    ................
4080    80 FF FF FF FF EF F6 FF FF FF FF FF FF FF FF FF    ................
4090    FF FF FF FF FF FF FF FF FF FF FF F7 FF FF FF FF    ................
40A0    00 00 00 20 08 00 00 40 00 00 5A 00 00 00 00 00    ...........Z.....
40B0    00 00 00 00 0B 80 00 00 00 00 80 00 00 40 20 00    ................
-d
40C0    00 00 00 00 09 00 00 00 00 02 10 02 80 20 08 00    ................
40D0    C0 20 00 00 A0 00 00 82 00 08 02 08 00 40 28 00    ................
40E0    00 FF FF FF FF F7 FF FF FF FF FF FF FF FF FF F7    ................
40F0    FF FF FF FF FF FE FF FF FF FF FF FF FF FF FF FF    ................
4100    10 D8 EB 62 DE 69 5B 3E FF FF FF FF FF FD FF FF    ................
4110    FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF    ................
-
```

การตั้งจุดหยุด และการให้โปรแกรมของผู้ใช้ทำงาน (Run)

```
-bp 4000
-bp 4002
-bl
0    4000
1    4002
-bc 0
-bl
0    4002
-g
STATUS : Inactive
AF =FE00  BC =0000  DE =0000  HL =0000  IX=0015  IY=0000  SP=FEFF
AF'=0000  BC'=0000  DE'=0000  HL'=0000  I =00    R =6A    PC=4002
4002   16 FE              LD      D,FE
-g
STATUS : Active
-r
STATUS : Active
-s
STATUS : Inactive
AF =6624  BC =2065  DE =FE00  HL =0000  IX=0015  IY=0000  SP=FEFF
AF'=0000  BC'=0000  DE'=0000  HL'=0000  I =00    R =7B    PC=400D
400D   78              LD      A,B
-
```

บทที่ 7
บทสรุปและวิจารณ์

จากการพัฒนาและทดสอบในเครื่องต้นแบบโดยการทดลองเขียนโปรแกรมแอสเซมบลีของ Z-80 บน อีดิเตอร์ แล้วทำการแอสเซมเบลอร์ ลงในหน่วยความจำ ปรากฏว่าระบบสามารถทำงานถูกต้องอีกทั้งยังสามารถทำการดีบัก ได้ตรงตามที่ต้องการ ทำให้เกิดความสะดวกและความรวดเร็วในการที่จะพัฒนาโปรแกรม แต่ก็มีข้อจำกัดบางประการที่จะต้องพิจารณา เพื่อนำไปปรับปรุงในภายหลัง

7.1 ข้อจำกัดทางฮาร์ดแวร์

1. หน่วยความจำที่สามารถใช้ได้ ตั้งแต่ตำแหน่ง 4000H-FEFFH
2. วงจรภายนอกไม่สามารถใช้ Non-Maskable Interrupt ได้เนื่องจากการ์ดได้นำมาใช้ในการทำงานแบบทีละคำสั่งแล้ว
3. คำสั่ง RST 30H จะถูกสงวนไว้สำหรับการกำหนดจุดหยุดของโปรแกรมของผู้ใช้ แต่ผู้ใช้สามารถนำไปใช้อย่างอื่นได้ แต่ผู้ใช้ห้ามกำหนดจุดหยุดอีก
4. เนื่องจากมีการใช้ตำแหน่งหน่วยความจำในเครื่องพีซีตั้งแต่ E000:4000H ถึง E000:FFFFH เป็นหน่วยความจำร่วมในการ์ด ดังนั้นจะต้องไม่มีการ์ดอื่นที่มีหน่วยความจำทับกับตำแหน่งนี้
5. การ์ด Z-80 ได้ออกแบบพอร์ทที่ใช้ติดต่อกันระหว่างเครื่องพีซี คือ พอร์ทหมายเลข 3E0H กับ 3E1H ดังนั้นจะต้องไม่มีการ์ดอื่นที่มีพอร์ทตรงกัน
6. การออกแบบวงจรภายนอก จะต้องไม่ออกแบบพอร์ทซ้ำกับหมายเลขพอร์ท E0H กับ E1H เนื่องจากการ์ดนี้ได้นำพอร์ทนี้มาใช้ในการติดต่อกับเครื่องพีซี

7.2 ข้อจำกัดทางซอฟต์แวร์

7.2.1 ข้อจำกัดของโปรแกรมอีดิเตอร์

1. สามารถเขียนรหัสต้นฉบับได้ไม่เกิน 800 บรรทัด
2. แต่ละบรรทัดจุตัวอักษรได้ไม่เกิน 80 ตัวอักษร
3. ต้องการหน่วยความจำ ในการใช้งาน 64 กิโลไบต์

7.2.2 ข้อจำกัดของโปรแกรมแอสเซมเบลอร์

1. คำสั่งเทียมยังมีเพียงสามคำสั่ง คือ ORG ,DB (Define Byte) และ END

2. ไม่มีการสร้างแฟ้มข้อมูลแสดงการแอสเซมเบลอร์ (Listing File) กับแสดงตำแหน่งของตัวแปร (Map File)

3. ไม่มีการกระทำทางคณิตศาสตร์ให้ในการแปล

7.2.3 ข้อจำกัดของโปรแกรมดีบักเกอร์

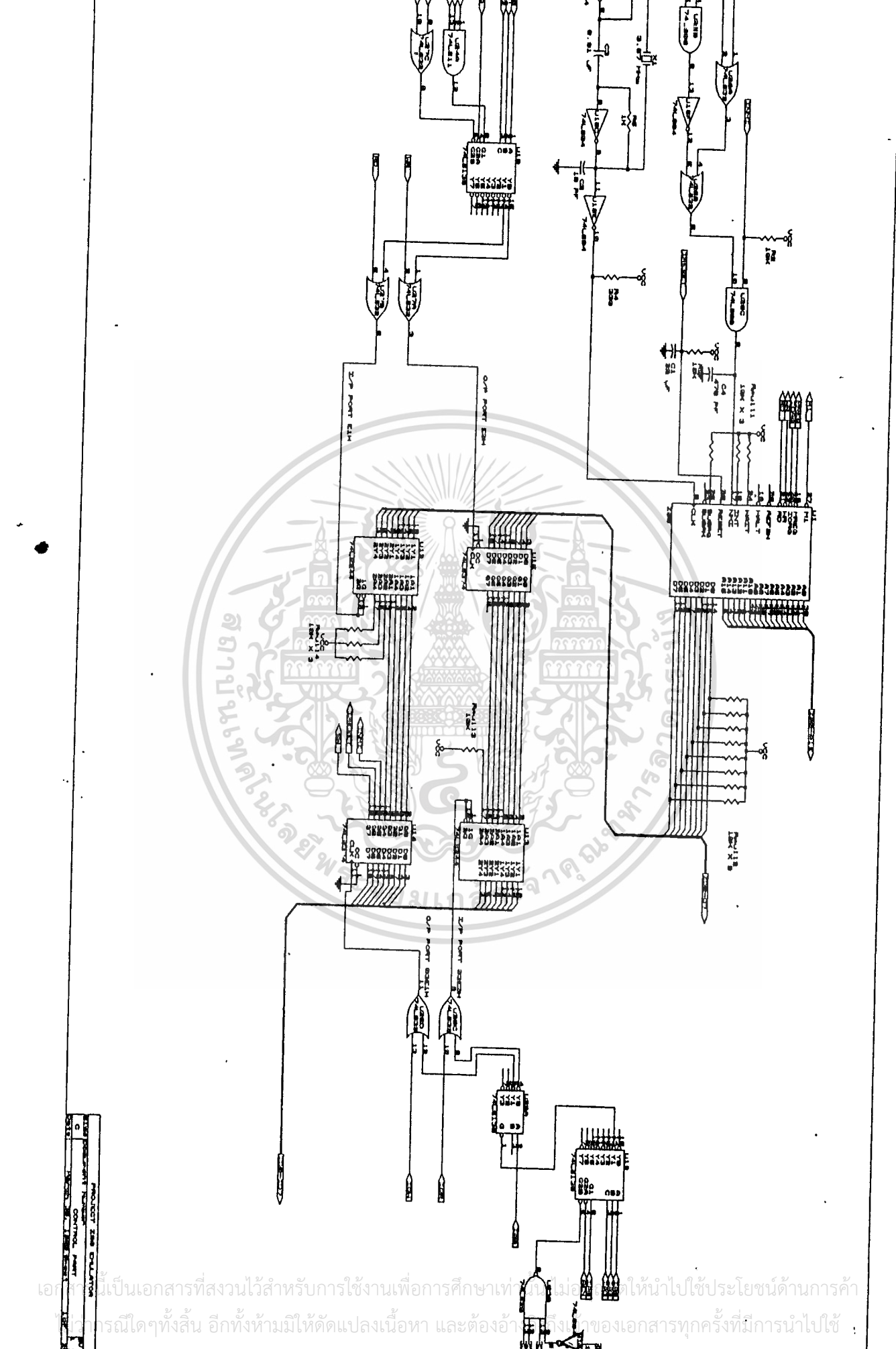
1. ไม่ได้เตรียมการทำแอสเซมเบลอร์แบบทีละบรรทัดให้
2. ไม่มีการกระทำทางคณิตศาสตร์ให้
3. ไม่ได้เตรียมการค้นหาข้อมูลที่ต้องการให้
4. การกำหนดจุดหยุด จะตั้งได้ไม่เกิน 15 จุด

7.3 บทสรุป

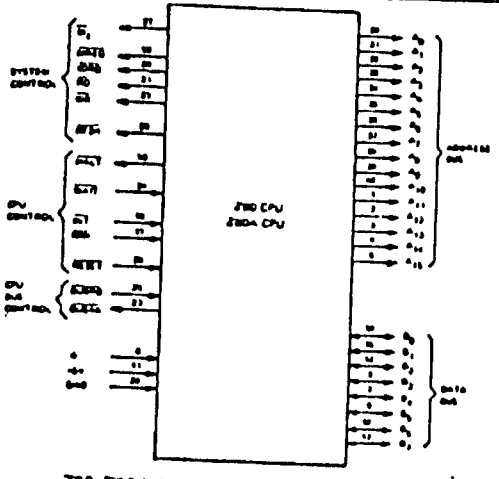
จากการที่นำเอาระบบรวมทั้งหมดมาทดลองต่อกับวงจรภายนอก ทำให้มีความสะดวกมากขึ้นในการทดสอบโปรแกรม และสามารถทำงานได้เป็นอย่างดี ยังผลให้การพัฒนาและการออกแบบวงจรในอนาคตที่ใช้ซีพียู Z-80 เกิดความสะดวกรวดเร็วมากยิ่งขึ้น

นอกจากนี้โครงสร้างทั้งหมดของระบบ Z80 อีมูเลเตอร์ ทั้งทางด้านฮาร์ดแวร์และซอฟแวร์เป็นแนวทางในการนำไปดัดแปลงโดยใช้ไมโครโพรเซสเซอร์เบอร์อื่น ๆ เช่น 8048 หรือ 8051 ได้โดยง่าย นอกจากจะช่วยในการพัฒนาโปรแกรมและวงจรตามที่ตั้งไว้แล้ว ยังช่วยประหยัดเวลาในการพัฒนาบุคลากรทางด้านนี้ด้วย ผลที่ได้คือประเทศชาติมีความเจริญรุ่งเรือง และในที่สุดก็สามารถพึ่งพาเทคโนโลยีของตัวเองได้

ภาคผนวก ก



PROJECT Z80 EMULATOR
MEMORY PART



Z80, Z80A CPU PIN CONFIGURATION

A_0-A_{15} (Address Bus) Tri-state output, active high. A_0-A_{15} constitute a 16-bit address bus. The address bus provides the address for memory (up to 64K bytes) data exchanges and for I/O device data exchanges.

D_0-D_7 (Data Bus) Tri-state input/output, active high. D_0-D_7 constitute an 8-bit bidirectional data bus. The data bus is used for data exchanges with memory and I/O devices.

$\overline{M_1}$ (Machine Cycle one) Output, active low. $\overline{M_1}$ indicates that the current machine cycle is the OP code fetch cycle of an instruction execution.

$\overline{MREQ}$ (Memory Request) Tri-state output, active low. The memory request signal indicates that the address bus holds a valid address for a memory read or memory write operation.

$\overline{IORQ}$ (Input/ Output Request) Tri-state output, active low. The $\overline{IORQ}$ signal indicates that the lower half of the address bus holds a valid I/O address for a I/O read or write operation. An $\overline{IORQ}$ signal is also generated when an interrupt is being acknowledged to indicate that an interrupt response vector can be placed on the data bus.

$\overline{RD}$ (Memory Read) Tri-state output, active low. $\overline{RD}$ indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the CPU data bus.

$\overline{WR}$ (Memory Write) Tri-state output, active low. $\overline{WR}$ indicates that the CPU data bus holds valid data to be stored in the addressed memory or I/O device.

$\overline{RFSH}$ (Refresh) Output, active low. $\overline{RFSH}$ indicates that the lower 7 bits of the address bus contain a refresh address for dynamic memories and the current $\overline{MREQ}$ signal should be used to do a refresh read to all dynamic memories.

$\overline{HALT}$ (Halt state) Output, active low. $\overline{HALT}$ indicates that the CPU has executed a HALT software instruction and is awaiting either a non-maskable or a maskable interrupt (with the mask enabled) before operation can resume. While halted, the CPU executes NOP's to maintain memory refresh activity.

$\overline{WAIT}$ (Wait) Input, active low. $\overline{WAIT}$ indicates to the Z-80 CPU that the addressed memory or I/O devices are not ready for a data transfer. The CPU continues to enter wait states for as long as this signal is active.

$\overline{INT}$ (Interrupt Request) Input, active low. The Interrupt Request signal is generated by I/O devices. A request will be honored at the end of the current instruction of the internal software controlled interrupt enable flip-flop (IFF) is enabled.

$\overline{NMI}$ (Non Maskable Interrupt) Input, active low. The non-maskable interrupt request line has a higher priority than $\overline{INT}$ and is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop. $\overline{NMI}$ automatically forces the Z-80 CPU to restart to location 0066_H.

$\overline{RESET}$ Input, active low. $\overline{RESET}$ initializes the CPU as follows: reset interrupt enable flip-flop, clear PC and registers I and R and set interrupt to 8080A mode. During reset time, the address and data bus go to a high impedance state and all control output signals go to the inactive state.

$\overline{BUSRQ}$ (Bus Request) Input active low. The bus request signal has a higher priority than $\overline{NMI}$ and is always recognized at the end of the current machine cycle and is used to request the CPU address bus, data bus and tri-state output control signals to go to a high impedance state so that other devices can control these buses.

$\overline{BUSAK}$ (Bus Acknowledge) Output, active low. Bus acknowledge is used to indicate to the requesting device that the CPU address bus, data bus and tri-state control bus signals have been set to their high impedance state and the external device can now control these signals.

Figure 2.13 Z-80 microprocessor pin numbers and definitions (Courtesy of Zilog, Inc.)

General-Purpose Arithmetic and CPU Control Groups

Mnemonic	Symbolic Operation	Flags S	Z	H	P/V	N	C	Opcode 76 543 210 Hex	Hex Bytes	Hex M Cycles	Hex T States	Comments
DAA	Converts acc. content into packed BCD following add or subtract with packed BCD operands.	↕	↕	X	↕	X	P	•	↕	00 100 111 27	1	1	4	Decimal adjust accumulator.
CPL	A ← $\bar{A}$	•	•	X	1	X	•	1	•	00 101 111 2F	1	1	4	Complement accumulator (one's complement).
NEG	A ← 0 − A	↕	↕	X	↕	X	V	1	↕	11 101 101 ED 01 000 100 44	2	2	8	Negate acc. (two's complement).
CCF	CY ← $\bar{CY}$	•	•	X	X	X	•	0	↕	00 111 111 3F	1	1	4	Complement carry flag.
SCF	CY ← 1	•	•	X	0	X	•	0	1	00 110 111 37	1	1	4	Set carry flag.
NOP	No operation	•	•	X	•	X	•	•	•.	00 000 000 00	1	1	4	
HALT	CPU halted	•	•	X	•	X	•	•	•	01 110 110 76	1	1	4	
DI	IFF ← 0	•	•	X	•	X	•	•	•	11 110 011 F3	1	1	4	
EI	IFF ← 1	•	•	X	•	X	•	•	•	11 111 011 FB	1	1	4	
IM 0	Set interrupt mode 0	•	•	X	•	X	•	•	•	11 101 101 ED 01 000 110 46	2	2	8	
IM 1	Set interrupt mode 1	•	•	X	•	X	•	•	•	11 101 101 ED 01 010 110 56	2	2	8	
IM 2	Set interrupt mode 2	•	•	X	•	X	•	•	•	11 101 101 ED 01 011 110 5E	2	2	8	

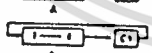
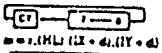
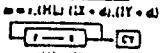
NOTES: IFF indicates the interrupt enable flip-flop
CY indicates the carry flip-flop
• indicates interrupts are not sampled at the end of EI or DI.

16-Bit Arithmetic Group

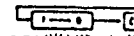
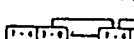
Mnemonic	Symbolic Operation	Flags S	Z	H	P/V	N	C	Opcode 76 543 210 Hex	Hex Bytes	Hex M Cycles	Hex T States	Comments
ADD HL, ss	HL ← HL + ss	•	•	X	X	X	•	0	↕	00 ss1 001	1	3	11	ss Reg 00 BC 01 DE 10 HL 11 SP
ADC HL, ss	HL ← HL + ss + CY	↕	↕	X	X	X	V	0	↕	11 101 101 ED 01 ss1 010	2	4	15	
SBC HL, ss	HL ← HL − ss − CY	↕	↕	X	X	X	V	1	↕	11 101 101 ED 01 ss0 010	2	4	15	
ADD IX, pp	IX ← IX + pp	•	•	X	X	X	•	0	↕	11 011 101 DD 00 pp1 001	2	4	15	pp Reg 00 BC 01 DE 10 IX 11 SP
ADD IY, rr	IY ← IY + rr	•	•	X	X	X	•	0	↕	11 111 101 FD 00 rr1 001	2	4	15	rr Reg 00 BC 01 DE 10 IY 11 SP
INC ss	ss ← ss + 1	•	•	X	•	X	•	•	•	00 ss0 011	1	1	6	
INC IX	IX ← IX + 1	•	•	X	•	X	•	•	•	11 011 101 DD 00 100 011 23	2	2	10	
INC IY	IY ← IY + 1	•	•	X	•	X	•	•	•	11 111 101 FD 00 100 011 23	2	2	10	
DEC ss	ss ← ss − 1	•	•	X	•	X	•	•	•	00 ss1 011	1	1	6	
DEC IX	IX ← IX − 1	•	•	X	•	X	•	•	•	11 011 101 DD 00 101 011 2B	2	2	10	
DEC IY	IY ← IY − 1	•	•	X	•	X	•	•	•	11 111 101 FD 00 101 011 2B	2	2	10	

NOTES: ss is any of the register pairs BC, DE, HL, SP
pp is any of the register pairs BC, DE, IX, SP
rr is any of the register pairs BC, DE, IY, SP

Rotate and Shift Group

Mnemonic	Symbolic Operation	Flags S	Z	H	P/V	N	C	Opcode 76 543 210 Hex	Hex Bytes	Hex M Cycles	Hex T States	Comments
RLCA		•	•	X	0	X	•	0	↕	00 000 111 07	1	1	4	Rotate left circular accumulator.
RLA		•	•	X	0	X	•	0	↕	00 010 111 17	1	1	4	Rotate left accumulator.
RRCA		•	•	X	0	X	•	0	↕	00 001 111 0F	1	1	4	Rotate right circular accumulator.
RRA		•	•	X	0	X	•	0	↕	00 011 111 1F	1	1	4	Rotate right accumulator.
RLC r	r, (HL), (IX + d), (IY + d)	↕	↕	X	0	X	P	0	↕	11 001 011 CB 00 000 r	2	2	8	Rotate left circular register r.
RLC (HL)		↕	↕	X	0	X	P	0	↕	11 001 011 CB 00 000 110	2	4	15	r Reg 000 B 001 C 010 D 011 E 100 H 101 L 111 A
RLC (IX + d)		↕	↕	X	0	X	P	0	↕	11 011 101 DD 11 001 011 CB — d — 00 000 110	4	6	23	
RLC (IY + d)		↕	↕	X	0	X	P	0	↕	11 111 101 FD 11 001 011 CB — d — 00 000 110	4	6	23	
RL m	 m = r, (HL), (IX + d), (IY + d)	↕	↕	X	0	X	P	0	↕	00 010 110				Instruction format and states are as shown for RLC's. To form new opcode replace 000 of RLC's with shown code
RRC m	 m = r, (HL), (IX + d), (IY + d)	↕	↕	X	0	X	P	0	↕	00 001 110				

Rotate and Shift Group (Continued)

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
RR m	 m ≡ r, (HL), (IX + d), (IY + d)	↕	↕	X	0	X	P	0	↕	CB 011					
SLA m	 m ≡ r, (HL), (IX + d), (IY + d)	↕	↕	X	0	X	P	0	↕	CB 100					
SRA m	 m ≡ r, (HL), (IX + d), (IY + d)	↕	↕	X	0	X	P	0	↕	CB 101					
SRL m	 m ≡ r, (HL), (IX + d), (IY + d)	↕	↕	X	0	X	P	0	↕	CB 111					
RLD	 A (HL)	↕	↕	X	0	X	P	0	•	ED 11 101 101 6F 01 101 111	2	5	18	Rotate digit left and right between the accumulator and location (HL). The content of the upper half of the accumulator is unaffected	
RRD	 A (HL)	↕	↕	X	0	X	P	0	•	ED 11 101 101 67 01 100 111	2	5	18		

Bit Set, Reset and Test Group

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
BIT b, r	$Z = \overline{r_b}$	X	↕	X	1	X	X	0	•	CB 01 b r	2	2	8	r: A 111, B 000, C 001, D 010, E 011, H 100, L 101	
BIT b, (HL)	$Z = \overline{(HL)_b}$	X	↕	X	1	X	X	0	•	CB 01 b 110	2	3	12	b: Bit Tested 000 0, 001 1, 010 2, 011 3, 100 4, 101 5, 110 6, 111 7	
BIT b, (IX + d)	$Z = \overline{(IX+d)_b}$	X	↕	X	1	X	X	0	•	DD CB d 01 b 110	4	5	20		
BIT b, (IY + d)	$Z = \overline{(IY+d)_b}$	X	↕	X	1	X	X	0	•	FD CB d 01 b 110	4	5	20		
SET b, r	$r_b = 1$	•	•	X	•	X	•	•	•	CB 11 b r	2	2	8		
SET b, (HL)	$(HL)_b = 1$	•	•	X	•	X	•	•	•	CB 11 b 110	2	4	15		
SET b, (IX + d)	$(IX+d)_b = 1$	•	•	X	•	X	•	•	•	DD CB d 11 b 110	4	6	23		
SET b, (IY + d)	$(IY+d)_b = 1$	•	•	X	•	X	•	•	•	FD CB d 11 b 110	4	6	23		
RES b, m	$m_b = 0$ m ≡ r, (HL), (IX + d), (IY + d)	•	•	X	•	X	•	•	•	10				To form new opcode replace 11 of SET b, s with 10. Flags and time states for SET instruction.	

NOTE: The notation m_b indicates bit b (0 to 7) of location m.

Jump Group

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
JP nn	PC ← nn	•	•	X	•	X	•	•	•	C3 — n — — n —	3	3	10		
JP cc, nn	If condition cc is true PC ← nn, otherwise continue	•	•	X	•	X	•	•	•	11 cc 010 — n — — n —	3	3	10	cc: 000 NZ non-zero, 001 Z zero, 010 NC non-carry, 011 C carry, 100 PO parity odd, 101 PE parity even, 110 P sign positive, 111 M sign negative	
JR e	PC ← PC + e	•	•	X	•	X	•	•	•	18 — e - 2 —	2	3	12		
JR C, e	If C = 0, continue If C = 1, PC ← PC + e	•	•	X	•	X	•	•	•	38 — e - 2 —	2 2	2 3	7 12	If condition not met If condition is met.	
JR NC, e	If C = 1, continue If C = 0 PC ← PC + e	•	•	X	•	X	•	•	•	30 — e - 2 —	2 2	2 3	7 12	If condition not met. If condition is met.	
JP Z, e	If Z = 0 continue If Z = 1, PC ← PC + e	•	•	X	•	X	•	•	•	28 — e - 2 —	2 2	2 3	7 12	If condition not met. If condition is met.	
JR NZ, e	If Z = 1 continue If Z = 0 PC ← PC + e	•	•	X	•	X	•	•	•	20 — e - 2 —	2 2	2 3	7 12	If condition not met. If condition is met.	
JP (HL)	PC ← HL	•	•	X	•	X	•	•	•	E9	1	1	4		
JP (IX)	PC ← IX	•	•	X	•	X	•	•	•	DD E9	2	2	8		

16-Bit Load Group

Mnemonic	Symbolic Operation	S	Z	Flags H	P/V	N	C	Opcode 76 543 210	Hex	Bytes	M Cycles	T States	Comments
LD dd, nn	dd ← nn	·	·	X	·	X	·	·	·	00 dd0 001 – n – – n –		3	3	10	dd: Pair 00 BC 01 DE 10 HL 11 SP
LD IX, nn	IX ← nn	·	·	X	·	X	·	·	·	11 011 101 DD 00 100 001 21 – n – – n –		4	4	14	
LD IY, nn	IY ← nn	·	·	X	·	X	·	·	·	11 111 101 FD 00 100 001 21 – n – – n –		4	4	14	
LD HL, (nn)	H ← (nn + 1) L ← (nn)	·	·	X	·	X	·	·	·	00 101 010 2A – n – – n –		3	5	16	
LD dd, (nn)	dd_H ← (nn + 1) dd_L ← (nn)	·	·	X	·	X	·	·	·	11 101 101 ED 01 dd1 011 – n – – n –		4	6	20	
LD IX, (nn)	IX_H ← (nn + 1) IX_L ← (nn)	·	·	X	·	X	·	·	·	11 011 101 DD 00 101 010 2A – n – – n –		4	6	20	
LD IY, (nn)	IY_H ← (nn + 1) IY_L ← (nn)	·	·	X	·	X	·	·	·	11 111 101 FD 00 101 010 2A – n – – n –		4	6	20	
LD (nn), HL	(nn + 1) ← H (nn) ← L	·	·	X	·	X	·	·	·	00 100 010 22 – n – – n –		3	5	16	
LD (nn), dd	(nn + 1) ← dd_H (nn) ← dd_L	·	·	X	·	X	·	·	·	11 101 101 ED 01 dd0 011 – n – – n –		4	6	20	
LD (nn), IX	(nn + 1) ← IX_H (nn) ← IX_L	·	·	X	·	X	·	·	·	11 011 101 DD 00 100 010 22 – n – – n –		4	6	20	
LD (nn), IY	(nn + 1) ← IY_H (nn) ← IY_L	·	·	X	·	X	·	·	·	11 111 101 FD 00 100 010 22 – n – – n –		4	6	20	
LD SP, HL	SP ← HL	·	·	X	·	X	·	·	·	11 111 001 F9		1	1	6	
LD SP, IX	SP ← IX	·	·	X	·	X	·	·	·	11 011 101 DD 11 111 001 F9		2	2	10	
LD SP, IY	SP ← IY	·	·	X	·	X	·	·	·	11 111 101 FD 11 111 001 F9		2	2	10	
PUSH qq	(SP – 2) ← qqL (SP – 1) ← qqH SP ← SP – 2	·	·	X	·	X	·	·	·	11 qq0 101		1	3	11	qq: Pair 00 BC 01 DE 10 HL 11 AF
PUSH IX	(SP – 2) ← IX_L (SP – 1) ← IX_H SP ← SP – 2	·	·	X	·	X	·	·	·	11 011 101 DD 11 100 101 E5		2	4	15	
PUSH IY	(SP – 2) ← IY_L (SP – 1) ← IY_H SP ← SP – 2	·	·	X	·	X	·	·	·	11 111 101 FD 11 100 101 E5		2	4	15	
POP qq	qq_H ← (SP + 1) qq_L ← (SP) SP ← SP + 2	·	·	X	·	X	·	·	·	11 qq0 001		1	3	10	
POP IX	IX_H ← (SP + 1) IX_L ← (SP) SP ← SP + 2	·	·	X	·	X	·	·	·	11 011 101 DD 11 100 001 E1		2	4	14	
POP IY	IY_H ← (SP + 1) IY_L ← (SP) SP ← SP + 2	·	·	X	·	X	·	·	·	11 111 101 FD 11 100 001 E1		2	4	14	

NOTES: dd is any of the register pairs BC, DE, HL, SP
qq is any of the register pairs AF, BC, DE, HL

(PAIR)$_H$, (PAIR)$_L$ refer to high order and low order eight bits of the register pair respectively, e.g., BC_L = C, AF_H = A

Exchange, Block Transfer, Block Search Groups

		S	Z	H	P/V	N	C					
EX DE, HL	DE ↔ HL	·	·	X	·	X	·	·	·	11 101 011 EB	1	1	4	
EX AF, AF'	$AF \leftrightarrow AF'$	·	·	X	·	X	·	·	·	00 001 000 08	1	1	4	
EXX	BC ↔ BC' DE ↔ DE' HL ↔ HL'	·	·	X	·	X	·	·	·	11 011 001 D9	1	1	4	Register bank and auxiliary register bank exchange
EX (SP), HL	H ↔ (SP + 1) L ↔ (SP)	·	·	X	·	X	·	·	·	11 100 011 E3	1	5	16	
EX (SP), IX	IX_H ↔ (SP + 1) IX_L ↔ (SP)	·	·	X	·	X	·	·	·	11 011 101 DD 11 100 011 E3	2	6	23	
EX (SP), IY	IY_H ↔ (SP + 1) IY_L ↔ (SP)	·	·	X	·	X	·	·	·	11 111 101 FD 11 100 011 E3	2	6	23	
LDI	(DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC – 1	·	·	X	①	X	0	·	·	11 101 101 ED 10 100 000 A0	2	4	16	Load (HL) into (DE), increment the pointers and decrement the byte counter (BC)
LDIR	(DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC – 1 Repeat until BC = 0	·	·	X	0	X	0	0	·	11 101 101 ED 10 110 000 B0	2 2	5 4	21 16	If BC ≠ 0 If BC = 0

NOTE: ① P/V flag = 0 if the result of BC – 1 = 0, otherwise P/V = 1.

Exchange, Block Transfer, Block Search Groups (Continued)

Mnemonic	Symbolic Operation	S	Z	Flags H	P/V	N	C	Opcode	Bytes	No. of M Cycles	No. of T States	Comments
LDD	(DE) ← (HL) DE ← DE - 1 HL ← HL - 1 BC ← BC - 1	•	•	X	0	X	①	0	•	11 101 101 10 101 000	ED A8	2	4	16	
LDDR	(DE) ← (HL) DE ← DE - 1 HL ← HL - 1 BC ← BC - 1 Repeat until BC = 0	•	•	X	0	X	0	0	•	11 101 101 10 111 000	ED B8	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
CPI	A - (HL) HL ← HL + 1 BC ← BC - 1	①	①	X	①	X	①	1	•	11 101 101 10 100 001	ED A1	2	4	16	
CPIR	A - (HL) HL ← HL + 1 BC ← BC - 1 Repeat until A = (HL) or BC = 0	①	①	X	①	X	①	1	•	11 101 101 10 110 001	ED B1	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)
CPD	A - (HL) HL ← HL - 1 BC ← BC - 1	①	①	X	①	X	①	1	•	11 101 101 10 101 001	ED A9	2	4	16	
CPDR	A - (HL) HL ← HL - 1 BC ← BC - 1 Repeat until A = (HL) or BC = 0	①	①	X	①	X	①	1	•	11 101 101 10 111 001	ED B9	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)

NOTES: ① P/V flag = 0 if the result of BC - 1 = 0, otherwise P/V = 1
② Z flag is 1 if A = (HL), otherwise Z = 0

8-Bit Arithmetic and Logical Group

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode	Bytes	M Cycles	T States	Comments
ADD A, r	A ← A + r	↕	↕	X	↕	V	0	↕	10 000 r	1	1	4	r: Reg 000 B 001 C 010 D 011 E 100 H 101 L 111 A
ADD A, n	A ← A + n	↕	↕	X	↕	V	0	↕	11 000 110 n	2	2	7	
ADD A, (HL)	A ← A + (HL)	↕	↕	X	↕	V	0	↕	10 000 110	1	2	7	
ADD A, (IX + d)	A ← A + (IX + d)	↕	↕	X	↕	V	0	↕	11 011 101 10 000 110 d	DD 3	5	19	
ADD A, (IY + d)	A ← A + (IY + d)	↕	↕	X	↕	V	0	↕	11 111 101 10 000 110 d	FD 3	5	19	
ADC A, s	A ← A + s + CY	↕	↕	X	↕	V	0	↕					s is any of r, n, (HL), (IX + d), (IY + d) as shown for ADD instruction. The indicated bits replace the 000 in the ADD set above
SUB s	A ← A - s	↕	↕	X	↕	V	1	↕					
SBC A, s	A ← A - s - CY	↕	↕	X	↕	V	1	↕					
AND s	A ← A ∧ s	↕	↕	1	X	P	0	0					
OR s	A ← A ∨ s	↕	↕	0	X	P	0	0					
XOR s	A ← A ⊕ s	↕	↕	0	X	P	0	0					
CP s	A - s	↕	↕	X	↕	V	1	↕					
INC r	r ← r + 1	↕	↕	X	↕	X	V	0	•	00 r 100	1	1	4	
INC (HL)	(HL) ← (HL) + 1	↕	↕	X	↕	X	V	0	•	00 110 100	1	3	11	
INC (IX + d)	(IX + d) ← (IX + d) + 1	↕	↕	X	↕	X	V	0	•	11 011 101 00 110 100 d	DD 3	6	23	
INC (IY + d)	(IY + d) ← (IY + d) + 1	↕	↕	X	↕	X	V	0	•	11 111 101 00 110 100 d	FD 3	6	23	
DEC m	m ← m - 1	↕	↕	X	↕	X	V	1	•					m is any of r, (HL), (IX + d), (IY + d) as shown for INC. DEC same format and states as INC. Replace 100 with 101 in opcode

Instruction Set

The Z80 microprocessor has one of the most powerful and versatile instruction sets available in any 8-bit microprocessor. It includes such unique operations as a block move for fast, efficient data transfers within memory or between memory and I/O. It also allows operations on any bit in any location in memory.

The following is a summary of the Z80 instruction set and shows the assembly language mnemonic, the operation, the flag status, and gives comments on each instruction. The *Z80 CPU Technical Manual* (03-0029-01) and *Assembly Language Programming Manual* (03-0002-01) contain significantly more details for programming use.

The instructions are divided into the following categories:

- 8-bit loads
- 16-bit loads
- Exchanges, block transfers, and searches
- 8-bit arithmetic and logic operations
- General-purpose arithmetic and CPU control
- 16-bit arithmetic operations
- Rotates and shifts
- Bit set, reset, and test operations
- Jumps
- Calls, returns, and restarts
- Input and output operations

A variety of addressing modes are implemented to permit efficient and fast data transfer between various registers, memory locations, and input/output devices. These addressing modes include:

- Immediate
- Immediate extended
- Modified page zero
- Relative
- Extended
- Indexed
- Register
- Register indirect
- Implied
- Bit

8-Bit Load Group

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Op code	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD r, r'	r ← r'	•	•	X	•	X	•	01 r r'	1	1	4	r, r' Reg
LD r, n	r ← n	•	•	X	•	X	•	00 r 110 – n –	2	2	7	000 B 001 C
LD r, (HL)	r ← (HL)	•	•	X	•	X	•	01 r 110	1	2	7	010 D
LD r, (IX + d)	r ← (IX + d)	•	•	X	•	X	•	11 011 101 DD 01 r 101 – d –	3	5	19	011 E 100 H 101 L
LD r, (IY + d)	r ← (IY + d)	•	•	X	•	X	•	11 111 101 FD 01 r 110 – d –	3	5	19	111 A
LD (HL), r	(HL) ← r	•	•	X	•	X	•	01 110 r	1	2	7	
LD (IX + d), r	(IX + d) ← r	•	•	X	•	X	•	11 011 101 DD 01 110 r – d –	3	5	19	
LD (IY + d), r	(IY + d) ← r	•	•	X	•	X	•	11 111 101 FD 01 110 r – d –	3	5	19	
LD (HL), n	(HL) ← n	•	•	X	•	X	•	00 110 110 36 – n –	2	3	10	
LD (IX + d), n	(IX + d) ← n	•	•	X	•	X	•	11 011 101 DD 00 110 110 36 – d – – n –	4	5	19	
LD (IY + d), n	(IY + d) ← n	•	•	X	•	X	•	11 111 101 FD 00 110 110 36 – d – – n –	4	5	19	
LD A, (BC)	A ← (BC)	•	•	X	•	X	•	00 001 010 0A	1	2	7	
LD A, (DE)	A ← (DE)	•	•	X	•	X	•	00 011 010 1A	1	2	7	
LD A, (nn)	A ← (nn)	•	•	X	•	X	•	00 111 010 3A – n – – n –	3	4	13	
LD (BC), A	(BC) ← A	•	•	X	•	X	•	00 000 010 02	1	2	7	
LD (DE), A	(DE) ← A	•	•	X	•	X	•	00 010 010 12	1	2	7	
LD (nn), A	(nn) ← A	•	•	X	•	X	•	00 110 010 32 – n – – n –	3	4	13	
LD A, I	A ← I	↕	↕	X	0	X	IFF	0 • 11 101 101 ED 01 010 111 57	2	2	9	
LD A, R	A ← R	↕	↕	X	0	X	IFF	0 • 11 101 101 ED 01 011 111 5F	2	2	9	
LD I, A	I ← A	•	•	X	•	X	•	• • 11 101 101 ED 01 000 111 47	2	2	9	
LD R, A	R ← A	•	•	X	•	X	•	• • 11 101 101 ED 01 001 111 4F	2	2	9	

NOTES: r, r' means any of the registers A, B, C, D, E, H, L
IFF the content of the IFF (interrupt enable flip-flop) is copied into the P/V flag
For an explanation of flag notation and symbols, see paragraph titled "Symbol Notation" following tables

Input and Output Group (Continued)

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76 543 210 Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
OTDR	(C) ← (HL) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1	X	X	1	X	11 101 101 ED 10 111 011	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to A_0 - A_7 B to A_8 - A_{15}

Summary of Flag Operation

Instruction	D_7 S	Z	H	P/V	N	D_0 C	Comments
ADD A, s; ADC A, s	↕	↕	X	V	0	↕	8 bit add or add with carry.
SUB s; SBC A, s; CP s; NEG	↕	↕	X	V	1	↕	8 bit subtract, subtract with carry, compare and negate accumulator.
AND s	↕	↕	1	P	0	0	Logical operations
OR s; XOR s	↕	↕	0	P	0	0	
INC s	↕	↕	↕	V	0	•	8 bit increment.
DEC s	↕	↕	↕	V	1	•	8 bit decrement.
ADD DD, ss	•	•	X	•	0	↕	16 bit add
ADC HL, ss	↕	↕	X	V	0	↕	16 bit add with carry.
SBC HL, ss	↕	↕	X	V	1	↕	16 bit subtract with carry.
RLA, RLCA, RRA, RRCA	•	•	0	•	0	↕	Rotate accumulator
RL m, RLC m, RR m, RRC m; SLA m; SRA m; SRL m	↕	↕	0	P	0	↕	Rotate and shift locations.
RLD, RRD	↕	↕	0	P	0	•	Rotate digit left and right.
DAA	↕	↕	↕	P	•	↕	Decimal adjust accumulator.
CPL	•	•	1	•	1	•	Complement accumulator.
SCF	•	•	0	•	0	1	Set carry.
CCF	•	•	X	•	0	↕	Complement carry.
IN r, (C)	↕	↕	0	P	0	•	Input register indirect.
INI, IND, OUTI, OUTD	X	↕	X	X	1	•	Block input and output. Z = 0 if B ≠ 0 otherwise Z = 0.
INIR, INDR, OTIR, OTDR	X	1	X	X	1	•	
LDI, LDD	X	X	0	↕	0	•	Block transfer instructions. P/V = 1 if BC ≠ 0, otherwise P/V = 0.
LDIR, LDDR	X	X	0	0	0	•	
CPI, CPIR, CPD, CPDR	X	↕	X	↕	1	•	Block search instructions. Z = 1 if A = (HL), otherwise Z = 0. P/V = 1 if BC ≠ 0, otherwise P/V = 0.
LD A, I; LD A, R	↕	↕	X	IFF	0	•	The content of the interrupt enable flip-flop (IFF) is copied into the P/V flag.
BIT b, s	X	↕	1	X	0	•	The state of bit b of location s is copied into the Z flag.

Symbolic Notation

Symbol	Operation
S	Sign flag. S = 1 if the MSB of the result is 1.
Z	Zero flag. Z = 1 if the result of the operation is 0.
P/V	Parity or overflow flag. Parity (P) and overflow (V) share the same flag. Logical operations affect this flag with the parity of the result while arithmetic operations affect this flag with the overflow of the result. If P/V holds parity, P/V = 1 if the result of the operation is even, P/V = 0 if result is odd. If P/V holds overflow, P/V = 1 if the result of the operation produced an overflow.
H	Half-carry flag. H = 1 if the add or subtract operation produced a carry into or borrow from bit 4 of the accumulator.
N	Add/Subtract flag. N = 1 if the previous operation was a subtract.
H & N	H and N flags are used in conjunction with the decimal adjust instruction (DAA) to properly correct the result into packed BCD format following addition or subtraction using operands with packed BCD format.
C	Carry/Link flag. C = 1 if the operation produced a carry from the MSB of the operand or result.

Symbol	Operation
↕	The flag is affected according to the result of the operation.
•	The flag is unchanged by the operation.
0	The flag is reset by the operation.
1	The flag is set by the operation.
X	The flag is a "don't care."
V	P/V flag affected according to the overflow result of the operation.
P	P/V flag affected according to the parity result of the operation.
r	Any one of the CPU registers A, B, C, D, E, H, L.
s	Any 8-bit location for all the addressing modes allowed for the particular instruction.
ss	Any 16-bit location for all the addressing modes allowed for that instruction.
ii	Any one of the two index registers IX or IY.
R	Refresh counter.
n	8-bit value in range < 0, 255 >.
nn	16-bit value in range < 0, 65535 >.

กิตติกรรมประกาศ

คณะผู้จัดทำขอขอบพระคุณ ท่านอาจารย์ที่ปรึกษาทั้งสองคือ ท่าน ศ.ดร. ไพรัช ธัชยพงษ์ และท่านอาจารย์ วิชา ศรีปัญญาพงศ์ ซึ่งคอยช่วยแนะนำหลักการ ,วางแนวทาง พร้อมทั้งวางแผนการทำงาน และคอยให้คำปรึกษาในเรื่องเทคนิคต่าง ๆ โดยเฉพาะอย่างยิ่งความรู้ทางด้านฮาร์ดแวร์ ตลอดจนท่านยังให้คำปรึกษาในทุก ๆ เรื่อง ทั้งยังเป็นผู้ที่ติดตามผลการทำงานและให้กำลังใจตลอดเวลาที่ผ่านมา จนทำให้ปริญญานิพนธ์ชิ้นนี้สำเร็จลุล่วงไปด้วยดี และท่านที่จะไม่กล่าวถึงไม่ได้เลยในที่นี้ คือท่านกรรมการสอบปริญญานิพนธ์ และท่านอาจารย์ภาควิชาวิศวกรรมคอมพิวเตอร์ซึ่งได้สละเวลาและให้อนุญาตคณะผู้จัดทำในการสอบปริญญานิพนธ์ก่อนกำหนด นอกจากนี้คณะผู้จัดทำยังขอขอบพระคุณทาง สำนักวิจัยและบริการคอมพิวเตอร์ ที่เอื้อเฟื้อสถานที่ตลอดจนอุปกรณ์ต่าง ๆ ที่ใช้ในการทำปริญญานิพนธ์

สุดท้ายนี้ขอขอบพระคุณบุคคลที่มีส่วนเกี่ยวข้องทางด้านเทคนิค และด้านการจัดพิมพ์ปริญญานิพนธ์ ตลอดจนผู้ที่ให้ความสนใจและช่วยเหลือมา ณ โอกาสนี้

เอกสารอ้างอิง

1. Adam Osborne ,Jerry Kane ,"An Introduction to Microcomputers Volume 2 Some Real Microprocessors" , Adam Osborne and Associate ,1978.
2. David C.Willen,Jeffrey I.Krantz,"8088 Assembler Language Programming: The IBM PC ", Howard W.Sams & Co., Inc., 1983.
3. Don Lancuster ,"TTL Cookbook" ,Howard W.sams & Co. ,Inc. ,335p. ,1974.
4. Efrem G.Mallach ,"Emulator Archictecture" ,Datamation , pp.24-32 ,August 1975.
5. "IBM Technical Reference" ,IBM Corp. ,1983.
6. James W. Coffron ,"Z80 Applications" ,SYBEX ,1983.
7. Jean-Paul Tremblay and Paul G.Sorenson ,"The Theory and Practice of Compiler Writing" ,McGRAW-HILL ,1985.
8. Kline ,B. ,M. Maerz ,and P.Rosenfeld,"The In-Circuit Approach to the Develoopment of Microcomputer-Based Products", Proceedings of The IEEE, Vol.64 (June 1976), pp.937-942, 1976.
9. Lance A.Leventhal ,"Z80 Assembly Language Programming" , Osborne Magraw Hill.
10. Lewis C. Eggebrecht , "Interfacing to the IBM Personal Computer" Howard W. Sams & Co.,Inc. 1983.
11. MOTORORA ,"MEMORY DATA".
12. "MPF-I User Manual" ,Multitech Industrial Corp. ,1981.
13. Russell Rector ,Grorge Alexy ,"The 8086 Book" , Osborne/McGraw-Hill ,1980.
14. คณะอนุกรรมการบัญญัติศัพท์คอมพิวเตอร์ราชบัณฑิตยสถาน ,"ศัพท์บัญญัติคอมพิวเตอร์ (ฉบับร่าง)" ,บริษัท เอ-อาร์ อินฟอร์เมชัน แอนด์ พับลิเคชัน จำกัด ,พ.ศ.2533
15. คู่มือเทียบเบอร์ไอซี TTL , ซีเอ็ดยูเคชั่น ,พิมพ์ครั้งที่ 6 ,พ.ศ.2531
16. คู่มือไอซีซีพียูพอร์ทและหน่วยความจำ ,ซีเอ็ดยูเคชั่น ,พ.ศ.2529
17. ดร.สมบูรณ์ จงชัยกิจ ,รศ.กฤษดา วิศวชีรานนท์ และสุภนันท์ หิรัญยพิสุทธิกุล,"เครื่องช่วยพัฒนาระบบไมโครโปรเซสเซอร์ระดับบอร์ด" ,การประชุมวิชาการวิศวกรรมไฟฟ้า 9 สถาบัน ครั้งที่ 12, 16-17 พฤศจิกายน 2532, หน้า 459-468

18. ดร.สมบูรณ์ จงชัยกิจ ,รศ.กฤษดา วิศวธีรานนท์ และเสกสิณห์ วัฒนะโชติ ,"อินเซอร์-กิตอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ Z-80 และ 8085" ,การประชุมวิชาการวิศวกรรมไฟฟ้า 9 สถาบัน ครั้งที่ 12, 16-17 พฤศจิกายน 2532, หน้า 469-478
19. ยืน ภู่วรรณ และวัฒนา เชียงกูล ,"ไมโครโปรเซสเซอร์ ไมโครคอมพิวเตอร์" ,ซี-เอ็ดยูเคชั่น ,หน้า 292, พ.ศ.2528
20. "อีที-บอร์ด 3.0" ,เซมิคอนดัคเตอร์ อิเลคโทนิคส์ ,พ.ศ.2532