ปีการศึกษา 2534

# อินเซอร์กิตอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ Z-80

# In-Circuit Emulator for Z-80 Microprocessor


โดย

นาย วิทศล วัชรทน รหัส 31.1015

นาย เชิดชัย ปิยกุลมาลา รหัส 31.1069

นาย พีรเชษฐ์ เชิดสถิรกุล รหัส 31.1188


อาจารย์ที่ปรึกษา

อาจารย์ สุรพันธ์ เอื้อไพบูลย์

# อินเซอร์กิตอีมูเลเตอร์สำหรับไมโครโปรเซสเซอร์ Z-80

โดย

นาย วิทวัส วัชโรทน รหัส 31.1015

นาย เชิดชัย ปิยกุลมาลา รหัส 31.1069

นาย พีรเชษฐ์ เชิดสถิรกุล รหัส 31.1188

อาจารย์ที่ปรึกษา

อาจารย์ สุรพันธ์ เอื้อไพบูลย์

บทคัดย่อ

ปริญญานิพนธ์นี้กล่าวถึง การออกแบบและพัฒนา In-Circuit Emulator สำหรับไมโครโปรเซสเซอร์ Z-80 โดยมีการติดต่อกับผู้ใช้ผ่านทาง IBM PC หรือ Terminal โดยอาศัย พอร์ท RS-232C พร้อมทั้งระบบซอฟท์แวร์ ซึ่งประกอบไปด้วย อิดิเตอร์, แอสเซมเบอร์, และ ดีบักเกอร์ ซึ่งสามารถทำ ซิงเกิลสเต็ป (Single Step) และกำหนดจุดหยุด (Break Point) ของโปรแกรมได้ และวงจรภายนอก (Target System) สามารถทำงานแบบ เวลาจริง (Real Time) ได้ มีหน่วยความจำ Emulation Memoryขนาด64KBytes โดยกำหนดMemoryMap ได้ด้วยซอฟท์แวร์

# In-Circuit Emulator for Microprocessor Z-80

By

Koson Vacharothone 31.1015

Cherdchai Piyakilmala 31.1069

Perached Cherdsatirakul 31.1188

Advisor:

Suraphun Airphaiboon

ABSTRACT

This thesis is about the design and the developement of the in-circuit emulator for Z-80 microprocessor. User interface can bemadethrough serial port (RS-232C) of an IBM PC or a terminal. The software system consists of Editor, Assembler, and Debugger which can do single step and mark breakpoint. The target system can run under realtime condition. The 64 KBytes of emulation memory can be located by the memory map through the software system.

# สารบัญ

# บทที่ 1
# บทนำ

การพัฒนาระบบที่ใช้ในไมโครโปรเซสเซอร์เป็นส่วนประกอบ (Microprocessor Based System) อาจพอแบ่งได้เป็น 2 ส่วน คือ การพัฒนาฮาร์ดแวร์ (Hardware) และ การพัฒนาซอร์ฟแวร์ (Software) ในยุคเริ่มต้น การพัฒนาวงจรและการพัฒนาโปรแกรมต้องดำเนินการแยกกัน ทำให้ต้องใช้เวลาดำเนินการมาก ต่อมาได้นำเอาระบบพัฒนาไมโครโปรเซสเซอร์ (Microprocessor Development System) มาช่วย ทำให้การพัฒนาวงจรและการพัฒนาโปรแกรมสามารถกระทำร่วมกันได้ โดยเฉพาะอย่างยิ่ง ส่วนที่เกี่ยวกับการหาความผิดพลาดของวงจร และการหาความผิดพลาดของโปรแกรม (Debug)

ระบบพัฒนาไมโครโปรเซสเซอร์ที่ใช้กันอย่างแพร่หลายในปัจจุบัน อาจพอแบ่งได้เป็น 2 ประเภท คือ การใช้ซิงเกิลบอร์ด (Single Board Microcomputer) และการใช้เครื่องอีมูเลเตอร์ (In-circuit Emulator) ระบบพัฒนาไมโครโปรเซสเซอร์ประเภทแรกมีขีดความสามารถค่อนข้างจำกัด เช่น ความสามารถของซีพียูและหน่วยความจำของระบบที่กำลังพัฒนาถูกนำมาใช้ได้ไม่เต็มที่ การหาความผิดพลาดของโปรแกรม (Debug) ทำได้จำกัดโดยเฉพาะในส่วนที่เกี่ยวกับ การทำเบรกพอยท์แบบมีเงื่อนไข (Conditional Breakpoint) และการเทรสแบบเรียลไทมน์ (Real Time Trace) นอกจากนั้นยังมีขีดจำกัดในการหาความผิดพลาดของวงจร อีกทั้งยังมีความล่าช้าที่เกิดขึ้นเสมอในการใช้ซิงเกิลบอร์ดส่วนใหญ่ คือ การที่นักคิจิตอลอิเล็คทรอนิกส์ต้องการพัฒนาโปรแกรมเพื่อทดสอบระบบ จะต้องทำการป้อนคำสั่งฮ๊อบโค๊ด (Opcode) ที่เป็นเลขฐานสิบหกลงในซิงเกิลบอร์ด ทำให้เกิดความไม่สะดวกและเกิดความล่าช้า เพราะจะต้องเสียเวลาให้กับกระบวนการดังกล่าว โดยเฉพาะถ้าโปรแกรมที่จะใช้ในการทดสอบระบบมีขนาดใหญ่มาก อีกทั้งยังทำให้เกิดความผิดพลาดอันเนื่องมาจากตัวของผู้ป้อนข้อมูลเอง หรือความยุ่งยากที่จะต้องทำการแบคอัพเรมาไว้เสมอ ส่วนระบบพัฒนาไมโครโปรเซสเซอร์ประเภทหลังมีขีดความสามารถสูงกว่าทั้งทางด้านการพัฒนาวงจรและการพัฒนาโปรแกรม คือสามารถทำการดีบัก (debug) และทดสอบวงจรที่ได้พัฒนาทุกส่วนโดยให้ผู้ใช้สามารถเขียนโปรแกรมที่เอดเตรสใดๆได้อย่างอิสระ ความสามารถในการป้องกันการเกิดสัญญาณรบกวนจากวงจรทาร์เก็ตซีสเตมท์

(Target System) เช่นในกรณีที่บัสข้อมูล (data bus) ที่ออกแบบไว้ต่อผิดผิดจะสามารถทำให้เครื่องอีมูเลเตอร์แฮงค์ได้ แต่อย่างไรก็ตามถึงแม้ว่าข้อดีและความสามารถของเครื่องอีมูเลเตอร์จะมีมากก็ตามแต่ราคานั้นสูงมาก จึงมีผู้นำมาใช้ไม่มากนัก

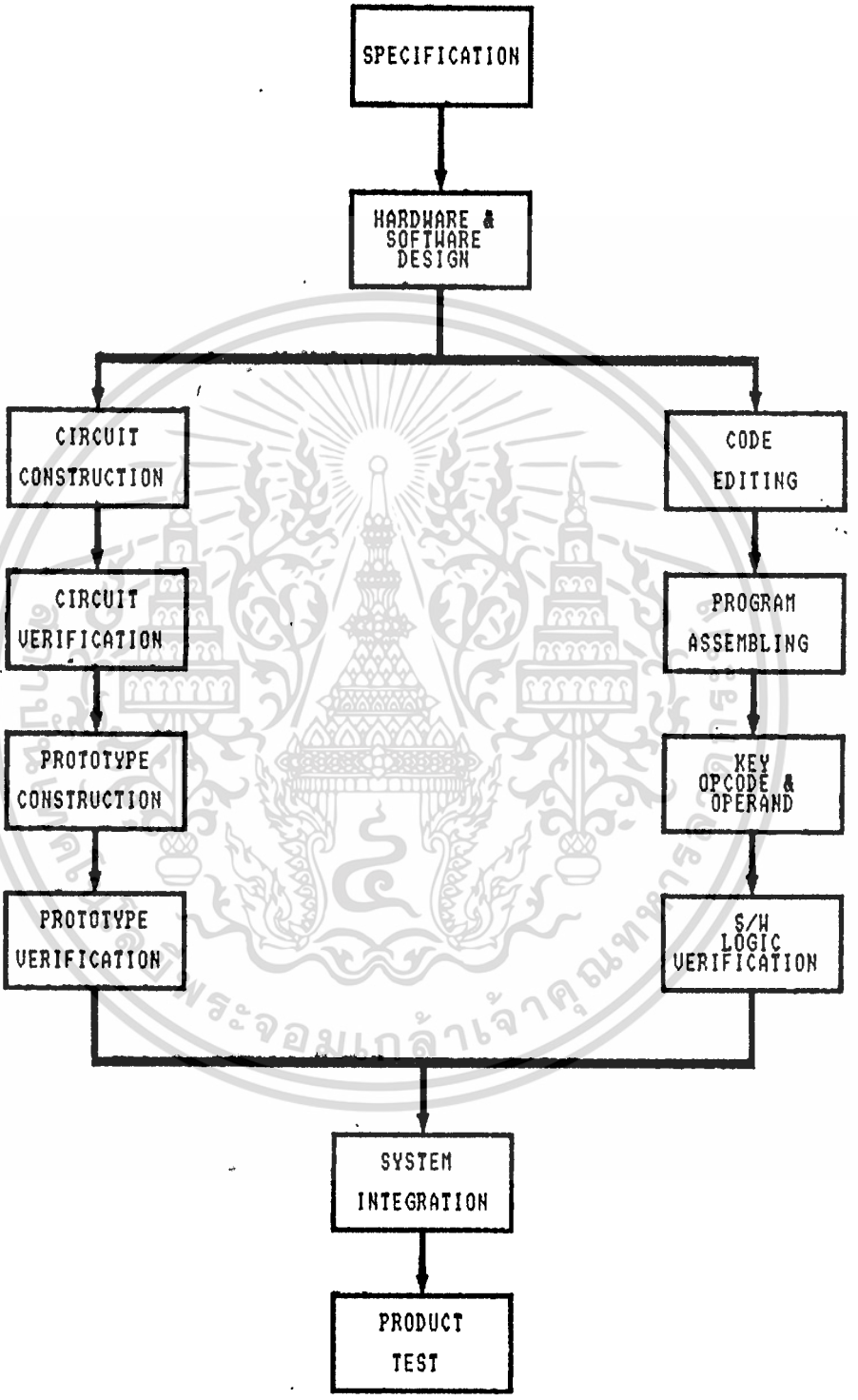
อนึ่ง แต่เดิมอีมูเลเตอร์จะเป็นระบบค่อนข้างใหญ่ทำให้มีราคาแพง ต่อมาได้มีการนำเครื่องคอมพิวเตอร์ (Personal Computer) ซึ่งมีราคาถูกมาประยุกต์ใช้งาน ทำให้ เครื่องอีมูเลเตอร์มีราคาถูกลง แต่กระนั้นก็ยังราคาค่อนข้างแพงอยู่

โครงงานพัฒนา เครื่องอีมูเลเตอร์นี้เป็นการพัฒนาเพื่อสร้าง เครื่องต้นแบบที่สามารถใช้งานร่วมกับเครื่องคอมพิวเตอร์ที่วางขายได้

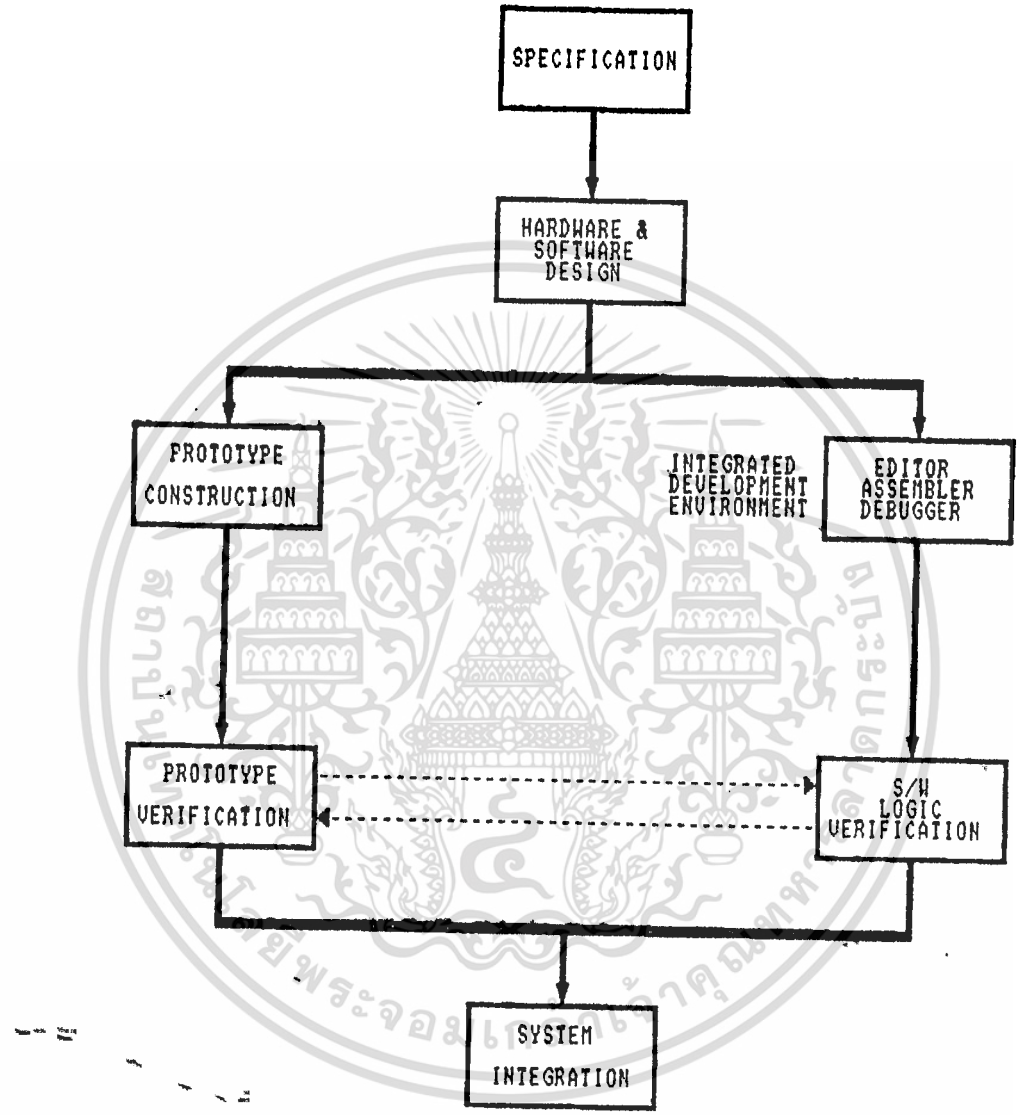
# บทที่ 2
# หลักการของ อีมูเลเตอร์

รูปที่ 2.1 เป็นการออกแบบระบบโดยทั่วๆไป ซึ่งเริ่มจากการหาข้อระบุเฉพาะหรือขอบเขตของระบบงาน (Specification) แล้วจึงเริ่มทำการออกแบบระบบฮาร์ดแวร์และซอฟท์แวร์ (Design) จากนั้นการพัฒนาระบบจะแยกกันทำไปพร้อมๆกันได้ คือทางด้านฮาร์ดแวร์จากการออกแบบวงจรในแต่ละส่วนแล้วทำการสร้างวงจรขึ้นมา (Circuit Construction) ตัวอย่างวงจรแต่ละส่วนอย่าง เช่น การออกแบบและสร้างในส่วนของหน่วยความจำ ซึ่งจะแยกจากส่วนของอินพุท เอาท์พุทพอร์ท และส่วนของการควบคุม จากนั้นขั้นต่อไปก็คือ การทดสอบในแต่ละส่วนว่ามีความถูกต้องตรงตามความต้องการมากน้อยเพียงใด (Circuit Verification) เมื่อทดสอบว่าส่วนต่างๆถูกต้องแล้ว ขั้นต่อไปก็จะนำส่วนต่างๆ มารวมเป็นระบบต้นแบบ (Prototype Construction) แล้วทำการตรวจสอบ (Prototype Verification) ในขณะเดียวกันถ้ามองด้านซอฟท์แวร์ ผู้พัฒนาระบบก็จะทำการพัฒนาโปรแกรมที่จะใช้ระบบฮาร์ดแวร์โดยใช้อิดิเตอร์ทำการเขียนโปรแกรม แล้วจะใช้ตัวแปลที่อยู่คนละระบบทำการแปล(Assembler)ให้เป็นรหัสคำเนินการ อย่างเช่น ในปัจจุบันมักจะพิมพ์รหัสคำเนินการแล้วนำมาป้อนลงชิงเกิลบอร์ดด้วยมือ หรือถ้าจะให้รวดเร็วขึ้นอีกก็จะต้องทำการดาวน์โหลด (Download) ด้วยวิธีการส่งทางซีเรียลพอร์ท (Serial Port) หรือทำเป็นลักษณะใช้แรมสองทาง ซึ่งจะต้องมีระบบฮาร์ดแวร์เพิ่มเติมขึ้นมา เมื่อถึงขั้นตอนนี้ระบบต้นแบบของวงจร และซอฟท์แวร์จะต้องนำมารวมให้เป็นระบบรวมก่อน (System Integration) ที่จะทำการทดสอบขั้นสุดท้ายที่จะเป็นผลิตภัณฑ์ต่อไป (Product Test)

จากขั้นตอนทั้งหมดในรูป 2.1 จะสามารถลดขั้นตอนดังกล่าวในลักษณะดังแสดงในรูปที่ 2.2 โดยใช้เครื่องอีมูเลเตอร์ช่วยในการพัฒนา โดยทางด้านฮาร์ดแวร์ผู้พัฒนาระบบสามารถออกแบบวงจรให้เป็นระบบต้นแบบบนกระดาษก่อน และส่วนทางด้านซอฟท์แวร์การพัฒนาจะทำบนระบบรวม ในระหว่างนี้ผู้พัฒนาสามารถทำการทดสอบระบบต้นแบบพร้อมกับทดสอบโปรแกรมได้อีกด้วยจนได้ผลิตภัณฑ์ออกมา




รูปที่ 2.1 การออกแบบระบบโดยทั่วๆไป



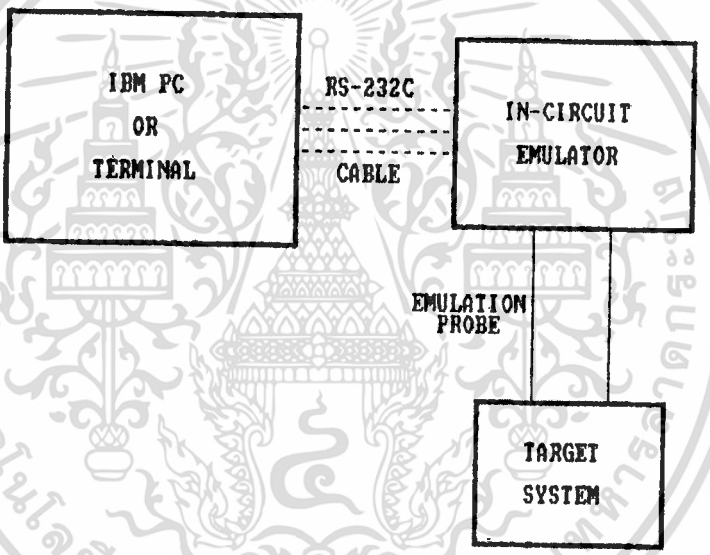

รูปที่ 2.2 การพัฒนาโดยใช้ เครื่องอีมูเลเตอร์ Z80

# บทที่ 3

# การออกแบบ (DESIGN)

**ลักษณะและคุณสมบัติของเครื่อง อีมูเลเตอร์ ที่พัฒนา**

ก่อนที่จะกล่าวถึงคุณสมบัติของเครื่องอีมูเลเตอร์ที่ได้พัฒนาขึ้น ลองมาพิจารณาการนำอีมูเลเตอร์ไปใช้ดังแสดงดังรูป 3.1




รูป 3.1 ภาพรวมการต่อ ICE เข้ากับระบบ

ระบบพัฒนาไมโครโปรเซสเซอร์ ซึ่งใช้ เครื่องอีมูเลเตอร์ประกอบด้วยส่วนสำคัญ 3 ส่วนคือ

-เครื่องคอมพิวเตอร์ที่มีพอร์ท RS-232C ใช้ในการเขียนคำสั่ง โดยอาศัยคีย์บอร์ดและจอภาพ ในการติดต่อกับเครื่องอีมูเลเตอร์ผ่านพอร์ท RS-232C

-ทาร์เก็ตซีสเต็มท์ คือระบบที่ใช้ไมโครโปรเซสเซอร์เป็นส่วนประกอบ ที่จะพัฒนา

-เครื่องอีมูเลเตอร์เป็นอุปกรณ์ที่ทำหน้าที่แทนซีพียูของทาร์เก็ตซีสเต็มท์ และ เครื่องอีมูเลเตอร์ยังมีความสามารถในการดีบั๊กทั้งซอร์ฟแวร์และฮาร์ดแวร์ และกรณีที่ทาร์เก็ตซีสเต็มท์ยังมีฮาร์ดแวร์ไม่ครบ เครื่องอีมูเลเตอร์จะมีสัญญาณนาฬิกาภายใน (Internal Clock) และหน่วยความจำอีมูเลชั่น (Emulation memory) ให้ใช้ในขณะทดลองแทนอุปกรณ์บนทาร์เก็ตซีสเต็มท์

**เครื่องอีมูเลเตอร์ที่พัฒนา มีลักษณะสมบัติดังนี้**

-ใช้ร่วมกับไมโครโปรเซสเซอร์เบอร์ Z-80

-ใช้ร่วมกับเครื่องคอมพิวเตอร์ไอบีเอ็มพีซี (IBM PC) หรือเครื่องคอมแพทิเบิ้ล(compatible ทั่วไปที่มีพอร์ทการส่งข้อมูลอนุกรม RS-232C

-สามารถเลือกใช้สัญญานนาฬิกาจากภายในเครื่องอีมูเลเตอร์หรือจากทาร์เก็ตซิสเต็มท์ได้

-หน่วยความจำอีมูเลชั่นมีขนาด 64 kB และผู้ใช้สามารถกำหนด เมมโมรีแมพ์ (memory map)ด้วยคำสั่งได้ช่วงละ 4 kB กรณีที่โปรแกรมทำงานผิดพลาดจากข้อกำหนดในเมมโมรีแมพ์ ระบบจะหยุดทำงานทันที และบอกความผิดพลาดให้ผู้ใช้ทราบ

-ผู้ใช้สามารถกำหนดให้รับหรือไม่รับสัญญาน INT, NMI, BUDRQ, RESET จากทาร์เก็ตซิสเต็มท์ได้

-แสดงและแก้ไขเปลี่ยนแปลงข้อมูลใน รีจิสเตอร์ ,หน่วยความจำ และพอร์ทได้

-แสดงข้อมูลในหน่วยความจำ เป็นภาษาแอสเซมบลี (disassembler)

-ป้อนโปรแกรมเป็นภาษาแอสเซมบลีทีละบรรทัด (line assembler) ให้เป็นข้อมูลในหน่วยความจำ เพื่อการแก้ไขหรือทดลองโปรแกรมสั้นๆ

-มีการถ่ายโอนข้อมูลในรูป Intel Hex File จากเครื่องคอมพิวเตอร์ลงในหน่วยความจำของเครื่องอีมูเลเตอร์ (download) และส่งข้อมูลจากหน่วยความจำของเครื่องอีมูเลเตอร์ไปให้เครื่องคอมพิวเตอร์เก็บ (up load)

-สั่งให้ Z80 ทำงานทีละคำสั่ง (single step)และทีละแมชชีนไซเคิล (cycle step)

-สั่งให้ Z80 ทำงานในระบบเวลาจริง (real time)

-การกำหนดจุดหยุด (breakpoint) แบบมีเงื่อนไข และไม่มีเงื่อนไข ได้ 65536จุด

**เทคนิคการสร้างเครื่องอีมูเลเตอร์**

ปกติซีพียู Z80 จะทำงานตลอดเวลาโดยให้แอดเดรสของโปรแกรมกับหน่วยความจำที่เก็บโปรแกรม แล้วอ่านคำสั่งมาวิเคราะห์และทำงานตามคำสั่งนั้น วิธีที่จะหยุดซีพียูจากการทำงานตามปกตินั้นมีอยู่หลายวิธี แต่วิธีที่ใช้ควบคุมซีพียูอีมูเลชั่นในงานนี้คือ ทำให้ไปหยุดรอที่สถานะ Wait เพราะสถานะของบัสจะคงที่ ทำให้วงจรควบคุมอ่านสถานะของบัสได้ง่าย และซีพียูยังไม่รับคำสั่งหรือข้อมูลจากบัส ทำให้เปลี่ยนคำสั่งหรือข้อมูลเพื่อควบคุมซีพียูได้ทุกๆแมชชีนไซเคิล

โดยปกติซีพียูอีมูเลชั่นจะทำงานตามโปรแกรมที่ผู้ใช้เขียนไว้ในหน่วยความจำอีมูเลชั่น หรือหน่วยความจำในทาร์เก็ตซิสเต็มท์ แต่เมื่อเครื่องอีมูเลเตอร์ได้รับคำสั่งให้แสดงหรือแก้ไขข้อมูลในรีจิสเตอร์ ระบบจะมีการทำงานตามลำดับดังนี้

-ซีพียูควบคุม (control cpu) ส่งสัญญาณให้ซีพียูอีมูเลชั่นหยุดรอคำสั่ง

-ตัดการติดต่อระหว่างบัสข้อมูลของซีพียูอีมูเลชั่นกับหน่วยความจำ

-ซีพียูควบคุมส่งคำสั่งเกี่ยวกับรีจิสเตอร์ให้กับซีพียูอีมูเลชั่นทางบัสข้อมูล

-ซีพียูอีมูเลชั่นทำงานตามคำสั่งนั้นและแสดงผลบนบัสข้อมูล

-ซีพียูควบคุมอ่านข้อมูลของรีจิสเตอร์จากบัสข้อมูลของซีพียูอีมูเลชั่น

-ให้มีการติดต่อบัสข้อมูลของซีพียูอีมูเลชั่นกับหน่วยความจำ

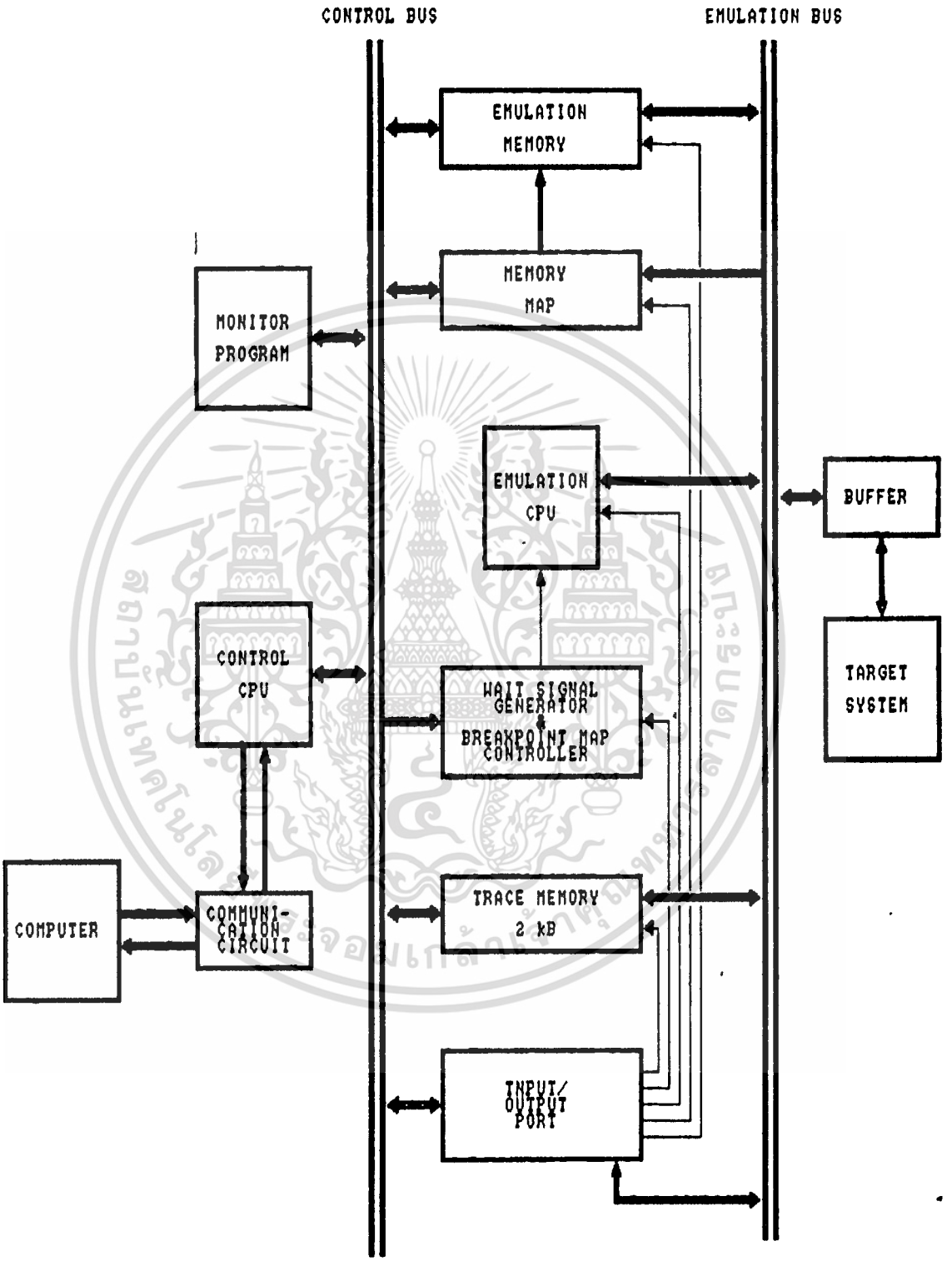
-ให้ซีพียูอีมูเลชั่นทำงานจากโปรแกรมในหน่วยความจำตามปกติ

รูปที่ 3.2 แสดงแผนภาพการเชื่อมต่อบัสแอดเดรส, ข้อมูลและสัญญาณควบคุมจากซีพียูควบคุมและทาร์เก็ตซิสเต็มท์กับหน่วยความจำอีมูเลชั่น ซึ่งสัญญาณควบคุมที่สำคัญ คือ สัญญาณ Wait ที่ควบคุมให้ซีพียูหยุดรอที่สถานะ Wait

กรณีมีการทำงานทีละคำสั่ง จะให้ซีพียูอีมูเลชั่นหยุดรอให้พอร์ทอินพุทของซีพียูควบคุมที่ต่อกับบัสของซีพียูอีมูเลชั่นอ่านสถานะของบัสเพื่อแสดงผล แล้วเลิกสัญญาณ Wait 2 รอบสัญญานนาฬิกา (Clock Cycle) เพื่อให้ซีพียูอีมูเลชั่นทำงาน แล้วหยุดในแมชชีนไซเคิลถัดไปเพื่ออ่านสถานะแล้วเลิกสัญญาณ Wait อีก จนเริ่มคำสั่งใหม่จึงหยุดรอคำสั่งจากผู้ใช้

ถ้าเป็นคำสั่งให้แสดงหรือแก้ไขข้อมูลในรีจิสเตอร์ วงจรควบคุมจะให้ซีพียูอีมูเลชั่นหยุดแล้วตัดการติดต่อระหว่างบัสข้อมูลของซีพียูอีมูเลชั่นกับหน่วยความจำและกับบัสข้อมูลของทาร์เก็ตติมท์ซิสเต็มท์ โดย disable Data Bus Buffer แล้วส่งอ็อบเจ็คของชุดคำสั่งที่จำเป็นให้ซีพียูอีมูเลชั่นทางพอร์ทเอาท์พุทของซีพียูควบคุมที่ต่อกับบัสข้อมูลของซีพียูอีมูเลชั่น แล้วยกเลิกสัญญาณ Wait 1 สัญญานนาฬิกา เพื่อให้ซีพียูอีมูเลชั่นรับคำสั่งนั้นเหมือนกับเป็นโปรแกรมมอนิเตอร์ (Monitor Program) สำหรับซีพียูอีมูเลชั่น แต่ไม่อยู่ในหน่วยความจำ สำหรับการแสดงและแก้ไขข้อมูลในหน่วยความจำและพอร์ทสามารถทำได้โดยตรงจากพอร์ทเอาท์พุทของซีพียูควบคุม

ถ้าเป็นการกำหนดจุดหยุด ซีพียูควบคุมจะส่งข้อมูลไปเก็บที่หน่วยความจำในเมมโมรีแมพคอนโทรลเลอร์เพื่อเปรียบเทียบกับสถานะของบัสของซีพียูอีมูเลชั่น ถ้ามีค่าตรงกันจะทำให้ ซีพียูอีมูเลชั่นเป็นสถานะ Wait ในแมชชีนไซเคิลแรกของคำสั่งต่อไปเพื่อให้ซีพียูควบคุมมาอ่านสถานะ




รูปที่ 3.2 แผ่นภาพการเชื่อมต่อบัสแอดเดรส ข้อมูล
และสัญญาณควบคุมของซีพียูอีมูเลชั่น

## 3.1 ฮาร์ดแวร์ (HARDWARE)

ในการออกแบบสร้างโครงงานชุดนี้ได้แบ่งรูปร่างภายนอกออกเป็นบอร์ด (board) 2 แผ่นซึ่งบอร์ดแต่ละแผ่นทำหน้าที่ดังนี้คือ บอร์ดควบคุม (controller board) และบอร์ดอีมูเลชั่น (emulation board) ในส่วนของบอร์ดควบคุมและ อีมูเลชั่นบอร์ดลักษณะการออกแบบวงจรในโครงงานชุดนี้ ใช้การออกแบบจากภายในบล็อกไดอะแกรม แล้วจึงนำสัญญาณต่างๆทั้งอินพุทและเอาท์พุทมาต่อเชื่อมเข้าด้วยกัน ดังนั้นจะเห็นว่าเริ่มแรกก่อนที่จะออกแบบวงจรนั้นจะต้องสร้างบล็อกไดอะแกรมในแต่ละส่วนขึ้นมาก่อน แล้วนำมาประกอบกัน หลังจากนั้นก็คือการออกแบบวงจร การอธิบายส่วนต่างๆของวงจรจึงแยกการอธิบายเป็นบล็อกไดอะแกรม โดยจะสามารถเปิดดูวงจรจริงๆได้จากภาคผนวกท้ายเล่ม วงจรในภาคผนวกท้ายเล่มนั้นจะไม่แบ่งเป็นบล็อกไดอะแกรม แต่จะต่อรวมวงจรทั้งหมดภายในบอร์ด โดยแยกออกมาเป็นแผ่นๆ การที่จะทราบว่าบล็อกไดอะแกรมที่ต้องการจะดูนั้นอยู่ภายในส่วนไหนของวงจรจึงต้องอ่านรายละเอียดจากการอธิบายในแต่ละบล็อกไดอะแกรมในหัวข้อหลังจากนี้

### 3.1.1 บอร์ดควบคุม

บอร์ดควบคุมจะประกอบด้วยส่วนต่างๆดังแสดงในรูปที่ 3.3 อธิบายส่วนต่างๆดังนี้

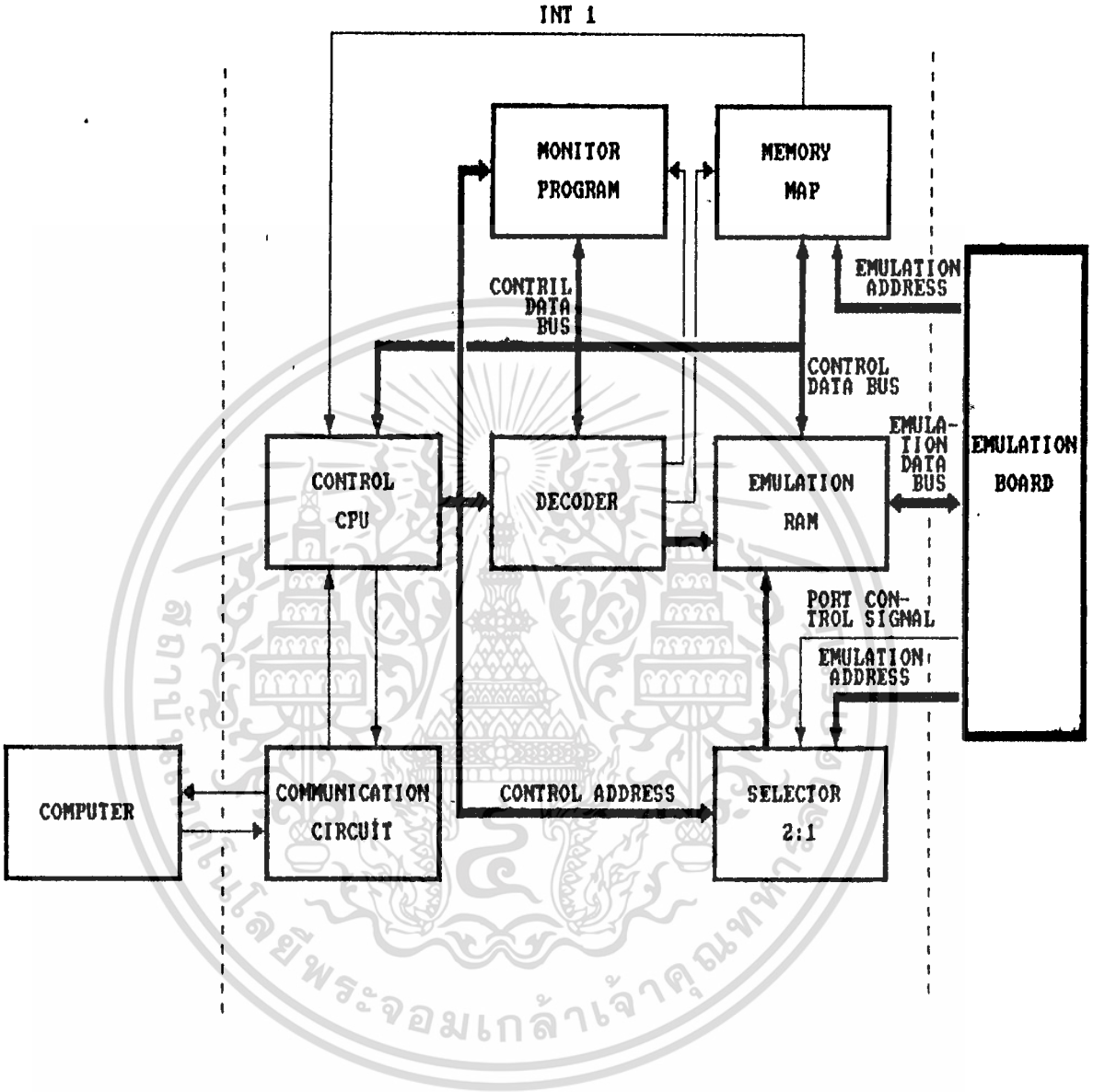
<u>คอนโทรลซีพียู</u>

เป็นส่วนที่ทำหน้าที่ควบคุมการทำงานของระบบ ซึ่งในโครงงานนี้ได้เลือกใช้ซีพียูเบอร์ MCS-51 ซึ่งจากการศึกษาซีพียูหลายๆเบอร์ พบว่าซีพียูเบอร์ MCS-51 มีข้อเหมาะสมสำหรับโครงงานนี้หลายประการคือ

- MCS-51 นั้นต้องการ ship support น้อย วงจรฮาร์แวร์นั้นไม่ยุ่งยาก อีกทั้งส่วนซอร์ฟแวร์ที่ควบคุมการทำงานของ MCS51 นั้นก็ง่ายและสะดวกในการใช้งาน

- MCS-51 นั้นสามารถอ้างหน่วยความจำได้ถึง 2 ส่วน คือ ส่วนหน่วยความจำโปรแกรม (Program Memory) และส่วนหน่วยความจำข้อมูล (Data Memory) ซึ่งส่วนหน่วยความจำโปรแกรมก็จะใช้ในการเก็บโปรแกรมควบคุมการทำงานของระบบ (ROM) และส่วนหน่วยความจำข้อมูลก็ใช้ในส่วนหน่วยความจำอีมูเลชั่นของระบบ

- ภายใน MCS-51 นั้น ยังมีหน่วยความจำต่างหากอีก ซึ่งส่วนนี้สามารถใช้เป็นที่เก็บข้อมูลชั่วคราวของซีพียูควบคุมได้




รูปที่ 3.3 บล็อกไดอะแกรมของบอร์ดควบคุม

- MCS-51 มีวงจรจับเวลา/นับเวลา (timer/counter) อยู่ในตัว ซึ่งสามารถควบคุมได้ง่ายโดยซอฟท์แวร์ ในส่วนฮาร์ดแวร์นั้นก็ไม่ยุ่งยาก การใช้ต่ออินเตอร์เฟสตามมาตรฐาน RS232C ก็เพียงแต่ต่อวงจรแปลงสัญญาณระดับ TTL เป็นระดับมาตรฐาน RS232C เท่านั้น

- MCS-51 สามารถทำงานแบบบิท (BIT) ได้ ซึ่งเหมาะกับงานควบคุมเป็นอย่างยิ่ง และจากการที่ MCS-51 นี้สามารถทำงานแบบบิทได้ จึงนำเอามาประยุกต์ให้ทำการอ้างหน่วยความจำได้มากกว่าปกติ

<u>มอนิเตอร์โปรแกรม</u> (monitor program) เป็นโปรแกรมมอนิเตอร์ของไมโครโปรเซสเซอร์ 8032 ใช้หน่วยความจำรอม (ROM) ขนาด 8 kB

<u>ดีโคดเดอร์</u> (decoder) เนื่องจากว่าไมโครโปรเซสเซอร์เบอร์ 8032 ไม่มีการจัดสัญญาณพอร์ตให้ใช้งาน มีแต่เพียงหน่วยความจำภายนอกเท่านั้น ดังนั้นถ้าต้องการใช้พอร์ตเพิ่มเติมจากที่จัดไว้ให้ จึงต้องมีการใช้เทคนิคการดีโคด แบบที่เรียกกันว่า เมมโมรีแมพ (memory-mapped) คือการดีโคดสัญญาณพอร์ตบนช่วงของหน่วยความจำที่มีอยู่ ในเมื่อต้องกันเนื้อที่ของหน่วยความจำภายนอกส่วนหนึ่งไปใช้งานเป็นพอร์ตเสีย จึงไม่มีหน่วยความจำเหลือพอสำหรับใช้เป็นอีมูเลชั่นเมมโมรี (อีมูเลชั่นเมมโมรีใช้เนื้อที่ขนาด 64 kB) จึงต้องมีการเพิ่มจำนวนของหน่วยความจำภายนอกให้มากขึ้นไปอีกโดยการเพิ่มบิตที่ 16 ออกไปอีก 1 บิต ทำให้มีขนาดของหน่วยความจำภายนอกทั้งหมด 128 kB การแบ่งหน่วยความจำและพอร์ตในโครงงานนี้มีดังนี้ คือ แหน่ง 00000-10000H ใช้งานเป็น อินพุท(input) และ เอาท์พุท (output)พอร์ต โดยแบ่งเป็นเอาท์พุทพอร์ต 8 พอร์ต อินพุทพอร์ต 4 พอร์ต ในวงจรใช้ไอซีหมายเลข 74LS138 74LS139 อย่างละ 1 ตัว

<u>วงจรสื่อสารรับส่งข้อมูล</u> การติดต่อระหว่างผู้ใช้กับโครงงานอีมูเลเตอร์นั้นกระทำผ่านทางพอร์ตอนุกรม RS-232 ของคอมพิวเตอร์ ดังนั้นจึงจำเป็นต้องมีวงจรสำหรับปรับระดับสัญญาณมาตรฐาน RS-232 ลงมาให้อยู่ในระดับสัญญาณที่ทีทีแอลลอจิก (TTL LOGIC) เพื่อให้สามารถใช้ร่วมกันได้กับโครงงานอีมูเลเตอร์ รูปวงจรดูได้จากภาคผนวกในแผ่นวงจรที่มีหมายเลข เอกสารชื่อว่า RS-232C

เซเลคเตอร์ 2:1 (selector 2:1) ทำหน้าที่เลือกสัญญาณแอดเดรสสำหรับอีมูเลชั่นแรม (emulation RAM) โดยเลือกระหว่างสัญญาณแอดเดรสจากไมโครโปรเซสเซอร์เบอร์ 8032 กับ สัญญาณแอดเดรสจากไมโครโปรเซสเซอร์เบอร์ Z-80 ภายในวงจรประกอบไปด้วยไอซีหมายเลข 74LS157 ทั้งหมด 5 ตัว และ 74LS32

อีมูเลชั่นแรม ใช้สแตติกแรม (static RAM) เบอร์ 62256 ขนาด 32 KB จำนวน 2 ตัว ต่อเป็นหน่วยความจำ ขนาด 64 KB ในหน่วยความจำส่วนนี้สามารถอ้างได้ทั้งจากคอนโทรบซีพียู และจาก อีมูเลชั่นซีพียู ทำให้ถูกใช้เป็นหน่วยความจำส่วนกลางสำหรับใช้งาน

เมมโมรีแมพ (memory map) เนื่องจากในขณะใช้งานเครื่องอีมูเลเตอร์นั้น ผู้ใช้อาจจะอยากให้หน่วยความจำ ณ.ที่ช่วงใดช่วงหนึ่งถูกกำหนดเป็น รอม, แรมหรือใช้หน่วยความจำภายนอก ดังนั้นจึงออกแบบส่วนนี้ขึ้นมาเพื่อให้กำหนดรูปแบบของอีมูเลชั่นเมมโมรี ในแต่ละช่วงลงไปได้ และในขณะที่โปรแกรมทำงานนอกเหนือจากข้อกำหนด เช่นกำหนดอีมูเลชั่นเมมโมรีช่วง 0000-1FFFH เป็นรอม แต่ในขณะเดียวกัน โปรแกรมกลับสั่งให้เขียนค่าลงไปในอีมูเลชั่นเมมโมรีช่วงดังกล่าว อีมูเลเตอร์ก็จะหยุดการทำงานของ อีมูเลชั่นซีพียูทันที และบอกให้ผู้ใช้ทราบว่าเกิดความผิดพลาดขึ้นในกรณีใหน ในวงจรจะใช้ไอซีหน่วยความจำขนาด 16x 4B เบอร์ 74S189, 74LS157, 74LS244

ข้อมูลการกำหนดชนิดของหน่วยความจำ

| D0 | D1 | NOT Y0 | NOT Y1 | หน่วยความจำ |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | แรม |
| 1 | 0 | 0 | 1 | รอม |
| 0 | 0 | 1 | 1 | หน่วยความจำภายนอก |

### 3.1.2 บอร์ดอีมูเลชั่น

ในส่วนของบอร์ดอีมูเลชั่นนี้จะมีสถาปัตยกรรมตามบล็อกไดอะแกรมดังรูปที่ 3.4 ในแต่ละบล็อกไดอะแกรมสามารถอธิบายได้โดยสังเขปดังนี้

อีมูเลชั่นซีพียู เป็นไมโครโปรเซสเซอร์เบอร์เดียวกับที่ต้องการจะอีมูเลต (emulate) ในโครงงานนี้คือไมโครโปรเซสเซอร์ตระกูล Z-80

บัฟเฟอร์ (buffer) เป็นส่วนของวงจรขับสัญญาณต่างๆของไมโครโปรเซสเซอร์ Z-80 เพื่อใช้กับการ์เก็ตซิสเต็ม ประกอบด้วยไอซีหมายเลข 74S244 และ 74S245 สัญญาณต่างๆ ของไมโครโปรเซสเซอร์ที่จำเป็นต้องผ่านวงจรขับคือ

1 บัสข้อมูล (data bus)

2 บัสแอดเดรส (address bus)

3 สัญญาณควบคุม ประกอบด้วย ฮอล์ท(halt), ไอโอรีเควส(IORQ), เมมโมรีรีเควส (MERQ), ไรท์(WR), รีด(RD), บัสรีเควส(BUSRQ), เอ็ม1(M1), รีเฟรช(RFSH)

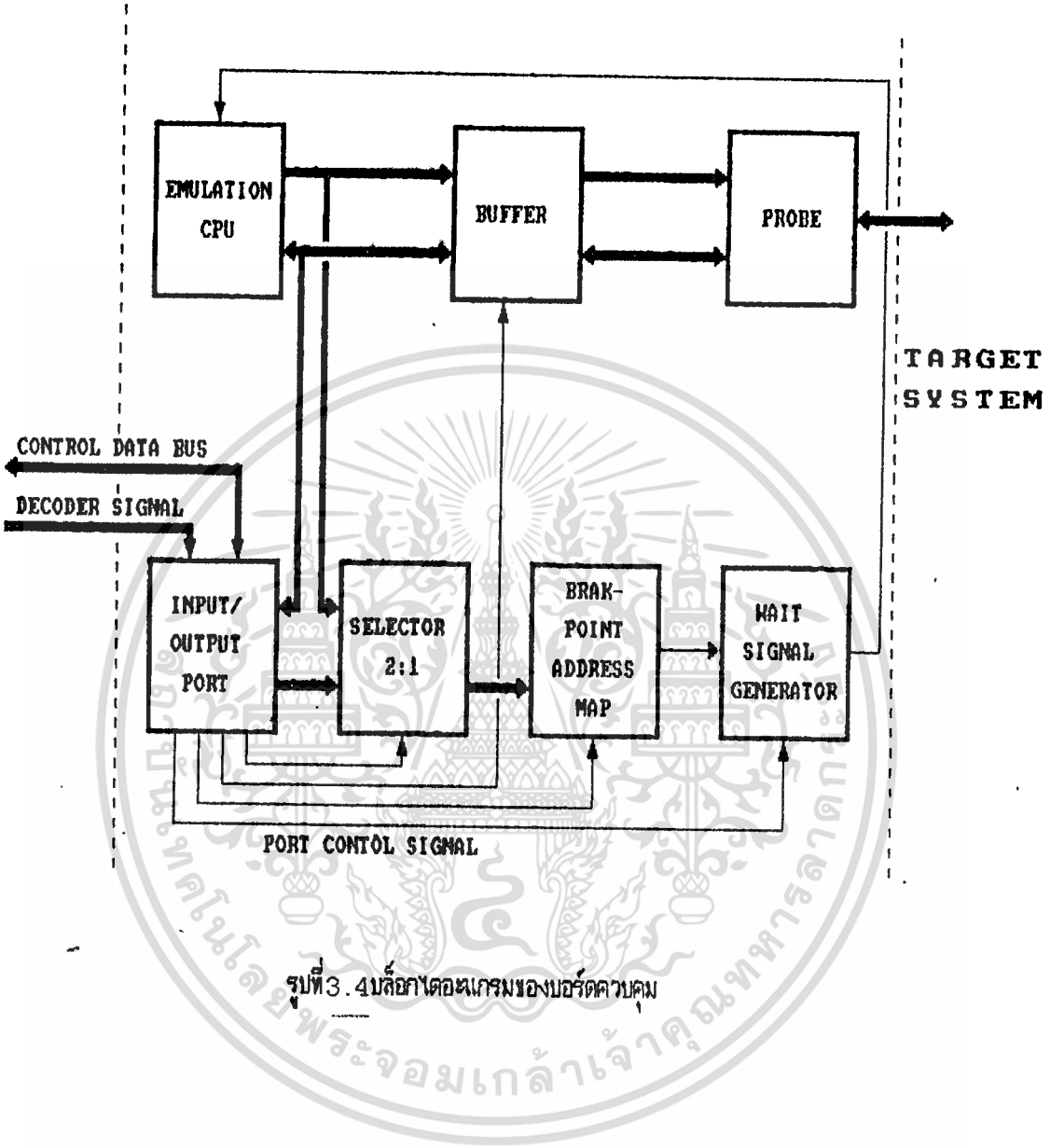
โพร์บ (PROBE) ดิพจัมเปอร์(dip jumper) 40 ขา มีลักษณะการจัดวางขาเหมือนกับไมโครโปรเซสเซอร์ Z-80 ใช้เสียบที่การ์เก็ตซิสเต็มแทนตำแหน่งของไมโครโปรเซสเซอร์ Z-80

อินพุทและเอาท์พุทพอร์ต (input/output port) ใช้รับส่งสัญญาณต่างๆระหว่างอีมูเลชั่นซีพียู กับคอนโทรลซีพียู อาทิเช่น รีจิสเตอร์ม โอเปอร์แรน (operant) แอดเดรส ฯลฯ ในวงจรใช้ไอซีหมายเลข 74LS373, 74LS244

สร้างสัญญาณควบคุมพอร์ต และสัญญาณทริก(trig)พอร์ต เพื่อใช้ควบคุมส่วนต่างๆของอีมูเลเตอร์ให้ทำงานสัมพันธ์กัน ในวงจรจะประกอบด้วย ไอซีเบอร์ 74LS75 74LS123 74LS32 74LS08 กำเนิดสัญญาณแอดเดรสที่ถูกนำไปใช้ในการอ้างจุดหยุดของโปรแกรม โดยจะมีขาสัญญาณแอดเดรสทั้งหมด 16 เส้น ในวงจรใช้ไอซีแลทช์ 74LS374ล



เซเลคเตอร์ 2:1 เลือกสัญญาณแอดเดรสสำหรับ เบรกพอยแอดเดรสแมพ (break




รูปที่ 3.4 บล็อกไดอะแกรมของบอร์ดควบคุม

point address map)โดยเลือกระหว่างสัญญาณแอดเดรสจากไมโครโปรเซสเซอร์ Z-80 และสัญญาณแอดเดรสที่มาจากอินพุทเอาท์พุทพอร์ต ที่กล่างถึงตอนต้น ภายในวงจรประกอบด้วย ไอซีหมายเลข 74LS257 ถึง 4 ตัว เพราะมีสัญญาณแอดเดรสทั้งหมด 16 เส้น

เบรกพอยแอดเดรสเมพ ในการตั้งหรือลบค่าของเบรคพอยแอดเดรส โครงงานชุดนี้ใช้วิธีการกำหนดแอดเดรส 1 ตำแหน่งของไมโครโปรเซสเซอร์ Z-80 ถูกแทนด้วย 1 บิตของแรมดังนั้นจะเห็นว่าไมโครโปรเซสเซอร์มีตำแหน่งแอดเดรสทั้งหมด $2^{16}$ ตำแหน่งจึงต้องใช้จำนวนบิตทั้งหมด $2^{16}$ บิต ซึ่งก็คือ แรมขนาด 64x1 KB ภายในวงจรประกอบด้วยแรมหมายเลข 6264 ต่อร่วมกับ 74S251, 74LS244, 74LS259 รวมกันเป็นสเตติกแรมขนาด 64x1 kB

ส่วนกำเนิดสัญญาณเวท(wait) เนื่องจากว่าในโครงงานชุดนี้ใช้วิธีการสร้างสัญญาณเวทเพื่อหยุดการทำงานของไมโครโปรเซสเซอร์ Z-80 ดังนั้นการที่จะให้ไมโครโปรเซสเซอร์ทำงานต่อไปได้จำเป็นจะต้องมีการยกเลิกสัญญาณเวท ซึ่งนั่นก็คือหน้าที่ของส่วนกำเนิดสัญญาณเวท ในส่วนนี้จะมีโหมดการทำงาน 2 โหมด คือโหมดแรก เป็นโหมดการทำงานทีละคำสั่ง(single step) และโหมดหลัง เป็นโหมดการทำงานทีละมาชีนไซเคิล (cycle step) ในวงจรประกอบด้วยไอซีหมายเลข 74S74, 74LS175, 74LS02, 74LS125

## 3.2 โครงสร้างระบบซอฟท์แวร์ของเครื่องอีมูเลเตอร์

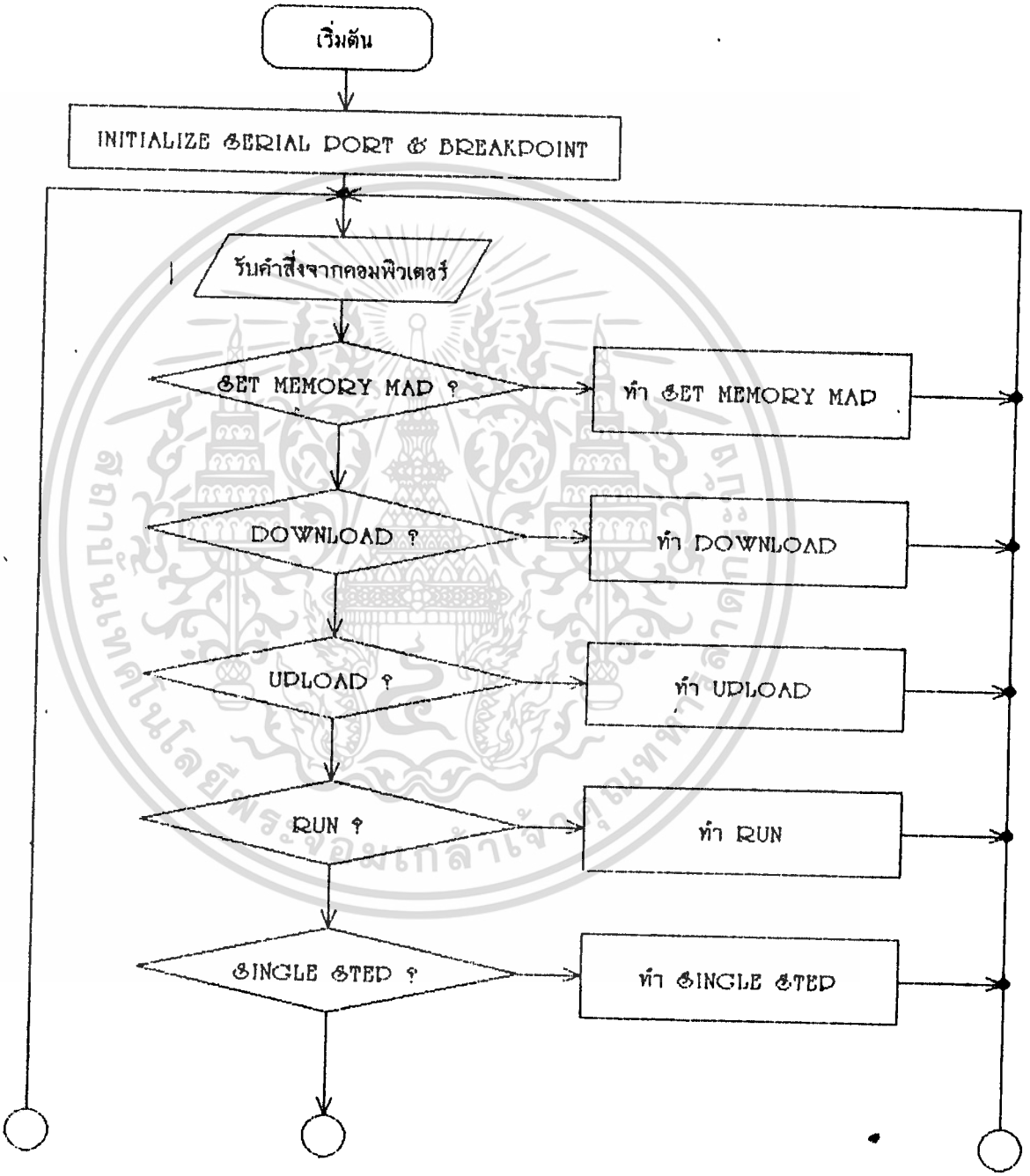
ประกอบด้วยสองส่วนคือ

### 3.2.1 โปรแกรมควบคุมการทำงาน (Monitor Program)

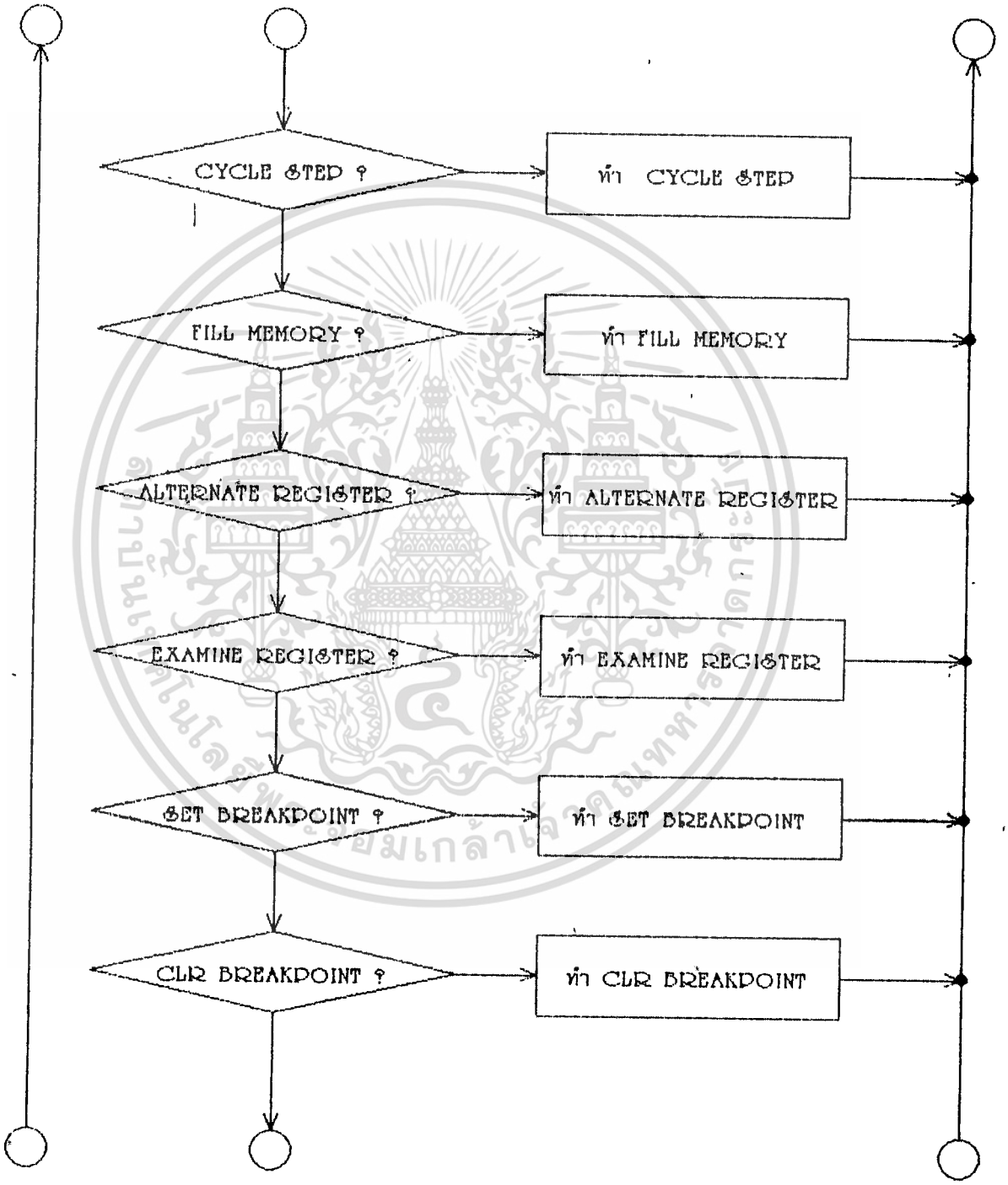
ส่วนโปรแกรมมอนิเตอร์นี้จะเป็นส่วนที่ควบคุมการทำงานของซีพียู MCS-51 โดยจะทำหน้าที่เป็นตัวกลางรับคำสั่งที่มาจากเครื่องคอมพิวเตอร์ผ่านทางพอร์ทRS-232 ทำการวิเคราะห์และตีความหมาย แล้วจึงไปทำงานตามคำสั่งนั้น เมื่อทำงานจนเสร็จแล้วก็จะหลับมารอรับคำสั่งต่อไปจากเครื่องคอมพิวเตอร์อีก อีกหน้าที่หนึ่งของโปรแกรมมอนิเตอร์คือการกำหนดสถานะเริ่มต้น (initialize) ของฮาร์ดแวร์ส่วนต่างๆ



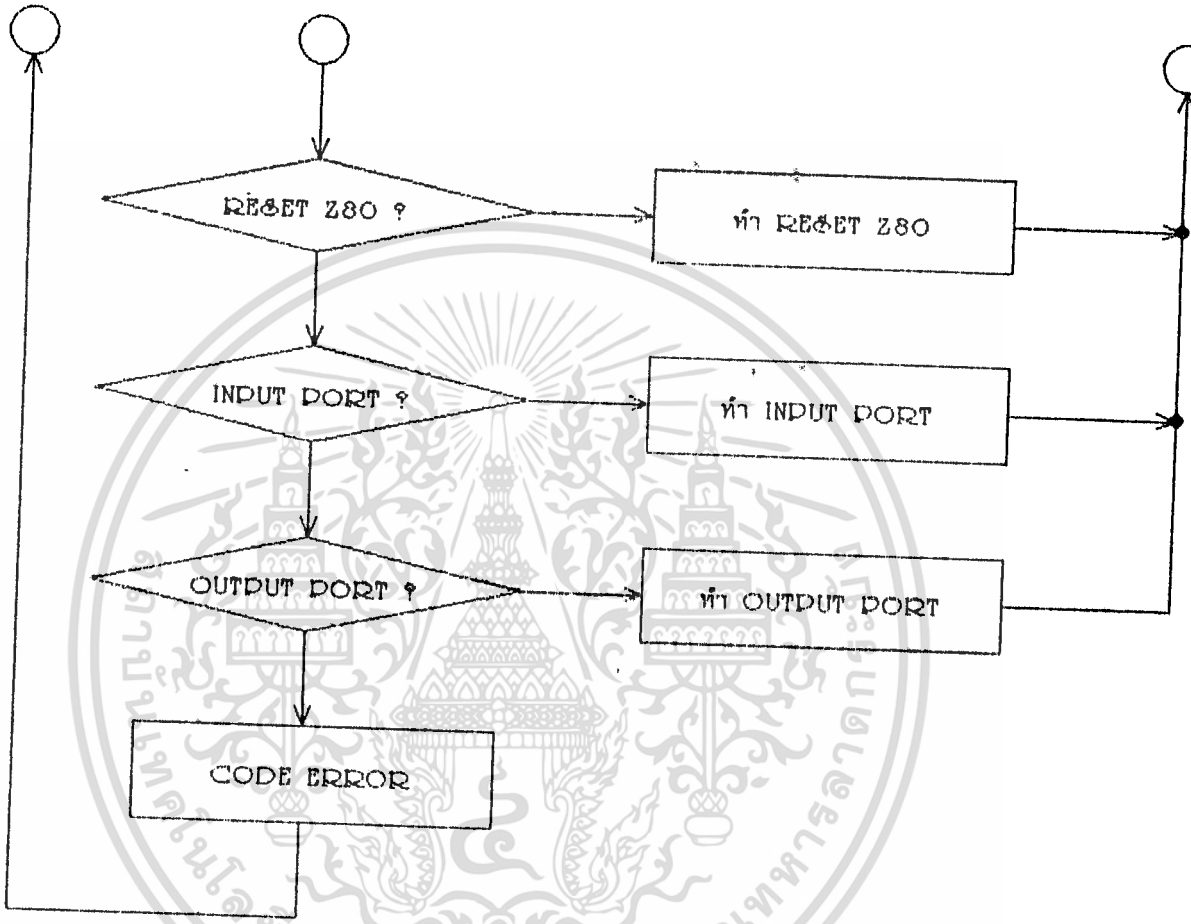
โปรแกรมมอนิเตอร์สามารถอธิบายเป็นส่วนต่างๆตามหน้าที่และตามคำสั่งที่รับมาจากเครื่องคอมพิวเตอร์ดังผังการทำงานรูปที่ 3.5 และอธิบายได้ดังนี้




รูปที่ 3.5
โปรแกรมหลัก




รูปที่ 3.5 (ต่อ)




รูปที่ 3.5 (ต่อ)

3.2.1.1 การกำหนดสถานะเริ่มต้น (initialize)

ในการเปิดเครื่องครั้งแรกหรือทำการรีเซท จำเป็นจะต้องมีการกำหนดสถานะให้แก่ฮาร์ดแวร์ส่วนต่างๆก่อนดังนี้

3.2.1.1.1 การกำหนดสถานะของการติดต่อกับเครื่องคอมพิวเตอร์

การติดต่อกับเครื่องคอมพิวเตอร์จะต้องมีการกำหนดอัตราการส่งและโปรโตคอล (protocal) การส่งซึ่งบนชิพ 8032 นั้นมีทั้งพอร์อนุกรมและตัวตั้งเวบาอยู่แล้ว ดังนั้นโปรแกรมมอนิเตอร์จะงเป็นการกำหนดค่าให้ SFR ให้ทำงานในโหมดต่างๆเท่านั้น โดยในที่นี้กำหนดอัตราการส่งไว้ 4800 bps โปรโตคอลการส่งให้มี 1 บิตเริ่มต้น 1 บิตหยุด บิตข้อมูล 8 บิต และไม่มีพาริตี้บิต

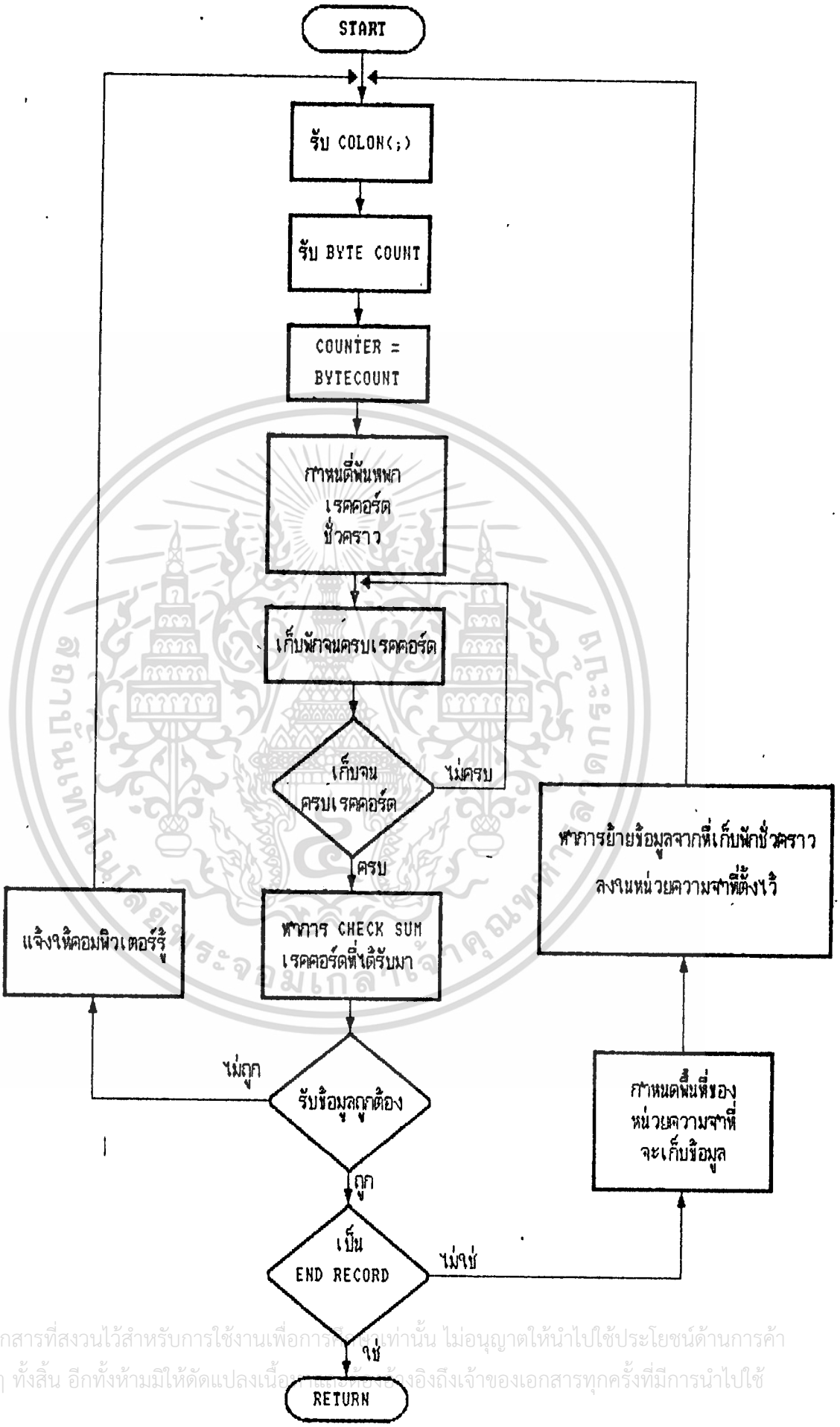
3.2.1.1.2 การกำหนดสถานะของส่วนควบคุมและส่วนหน่วยความจำ

ระบบฮาร์ดแวร์ของเครื่องอีมูเลเตอร์ที่สร้างขึ้นนั้นจะมีการรีเซ็ทเมื่อเปิดเครื่องหรือเมื่อกดปุ่มรีเซ็ทอยู่แล้ว แต่ก็มีบางส่วนที่ไม่ถูกรีเซ็ทจึงต้องสั่งโดยซอฟท์แวร์ เช่น ค่าของพอร์ทควบคุม (control port) และค่าในหน่วยความจำที่เก็บตำแหน่งของเบรกพอยท์ (break point map)

3.2.1.2 การรับแฟ้มข้อมูลจากเครื่องคอมพิวเตอร์ (download)

ในการรับข้อมูลจากเครื่องคอมพิวเตอร์นั้นจะมีข้อแม้อยู่ว่าแฟ้มข้อมูลนั้นจะต้องเป็นไฟล์แบบ INTEL-HEX FORMAT (ดูในภาคผนวก) เครื่องคอมพิวเตอร์จะทำการส่งข้อมูลมาทีละเรคคอร์ด ส่วนควบคุมจะทำการรับข้อมูลแล้วนำไปเก็บพักชั่วคราวไว้ในส่วนบัฟเฟอร์ภายใน (internal RAM) แล้วจึงทำการตรวจสอบผลรวมของไบต์ (check sum) ซึ่งจะมีการตอบรับกับเครื่องคอมพิวเตอร์ เมื่อตรวจสอบผลรวมถูกต้องก็จะนำข้อมูลจากแรมภายในไปเก็บไว้ในแรมภายนอก (external RAM) ตามตำแหน่งที่ระบุ แล้วจึงไปรอรับเรคคอร์ดต่อไป ถ้าการตรวจสอบผลรวมไม่ถูกต้อง ก็จะไม่มีการย้ายข้อมูลจากแรมภายในไปแรมภายนอก แต่จะให้เครื่องคอมพิวเตอร์ส่งเรคคอร์ดเดิมนี้ใหม่ จนกว่าการตรวจสอบถูกต้องจึงจะย้ายข้อมูล

ในกรณีที่เป็นเรคคอร์ดสิ้นสุดไฟล์ (end record) ก็จะทำการจรวจสอบก่อนแล้วจึงออกจากโปรแกรม การทำงานของส่วนนี้อธิบายได้ดังผังงานตามรูปที่ 3.6






รูปที่ 3.6 แสดงการ DOWNLOAD

3.2.1.3 การส่งแฟ้มข้อมูลให้เครื่องคอมพิวเตอร์ (upload)

เครื่องคอมพิวเตอร์จะทำการกำหนดตำแหน่งเริ่มต้น พร้อมทั้งจำนวนไบต์ที่ต้องการมาให้ จำนวนไบต์สูงสุดที่จะส่งได้แต่ละครั้งนั้นจะต้องไม่เกิน 128ไบต์ ส่วนควบคุมจะทำการส่งโดยเครื่องจะย้ายข้อมูลจากแรมภายนอกตามตำแหน่งที่ระบุเสร็จแล้วมาเก็บไว้ในแรมภายในก่อน แล้วจึงส่งไปให้เครื่องคอมพิวเตอร์พร้อมกับทำการตรวจสอบผลรวมไปด้วย ถ้าการตรวจสอบผลรวมไม่ถูกต้อง เครื่องคอมพิวเตอร์ก็จะสั่งให้ส่วนควบคุมทำการส่งข้อมูลใหม่ การทำงานแสดงได้ดังรูปที่ 3.7

3.2.1.4 คำสั่งรีเซ็ท

จะเป็นการสั่งให้พอร์ทอินพุท/เอาท์พุท กำเนิดสัญญาณให้เท่ากับรีเซ็ทของ Z-80

3.2.1.5 คำสั่ง GO

เครื่องคอมพิวเตอรจะส่งตำแหน่งเริ่มต้นมาให้ เมื่อซีพียูควบคุมรับมาแล้ว ก็จะทำการตัดการทำงานของซีพียู Z-80 ออกจากบาร์เกิดชิสเต็ม แล้วสั่งให้ซีพียู Z-80 กระโดดไปรออยู่ที่ตำแหน่งเริ่มต้นแต่ขณะนี้หน่วยความจำยังมีมูเลชั่นยังอยู่ในการควบคุมของ MCS-51 อยู่จึงต้องโอนการควบคุมนี้ไปให้ Z-80 จากนั้นจึงยกเลิกสัญญาณรอเพื่อให้ Z-80 ทำงานต่อไปโดยเริ่มที่ตำแหน่งที่ตั้งไว้จากนั้น MCS-51 ก็จะวนรอรับคำสั่งให้หยุดจากคอมพิวเตอร์

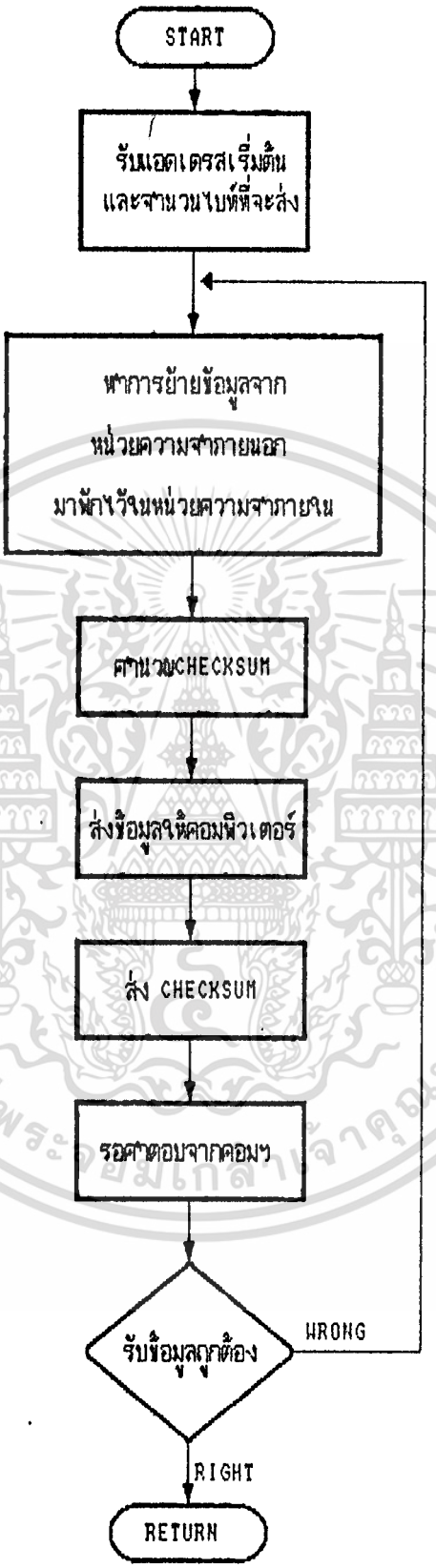
การที่จะสั่งให้ Z-80 ทำคำสั่งตามที่ซีพียูควบคุมสั่งนั้น ระบบฮาร์ดแวร์จะมีการเชื่อมต่อสายข้อมูลของทั้ง Z-80 และซีพียูควบคุม โดยมีพอร์ทอินพุทเอาท์พุทเป็นตัวเลือก เมื่อซีพียูควบคุมจะสั่ง Z-80 โปรแกรมมอนิเตอร์ก็จะส่งชุดคำสั่งของ Z-80 ไปให้ จากนั้นจึงสั่งให้อินพุท/เอาท์พุทพอร์ แลทช์ไว้ที่ตาต้าบัสรอให้ Z-80 อ่านชุดคำสั่งไปใช้ ผังการทำงานแสดงดังรูปที่ 3.8

3.2.1.6 คำสั่งซิงเกิลสเต็ป (SINGLE STEP)

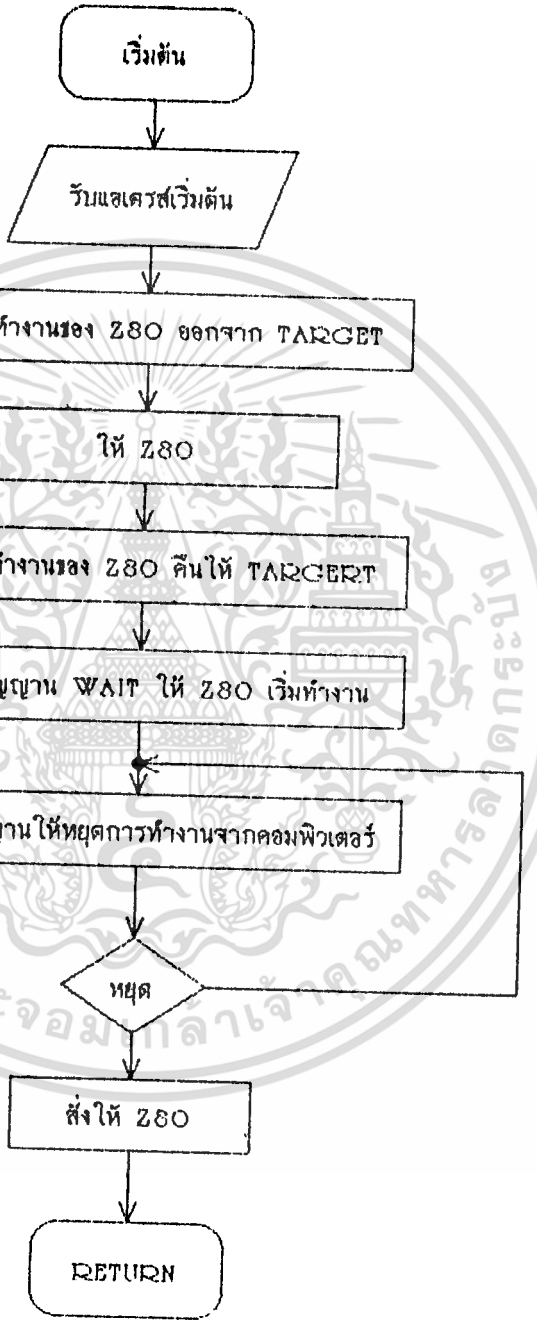
เครื่องคอมพิวเตอร์จะส่งตำแหน่งที่จะทำการซิงเกิลสเต็ป จากนั้นซีพียูควบคุมจะสั่งให้ Z-80 กระโดดไปที่ตำแหน่งนั้นและทำการอ่านค่ารีจิสเตอร์และสถานะบัสต่างๆก่อนเริ่มทำชุดคำสั่งนั้น เมื่อทำชุดคำสั่งแล้วจึงแสดงผลอีกครั้ง แสดงผังงานดังรูปที่ 3.9



การอ่านค่ารีจิสเตอร์ภายใน Z-80 นั้นจะมีเทคนิคคือ โปรแกรมมอนิเตอร์จะสั่งให้

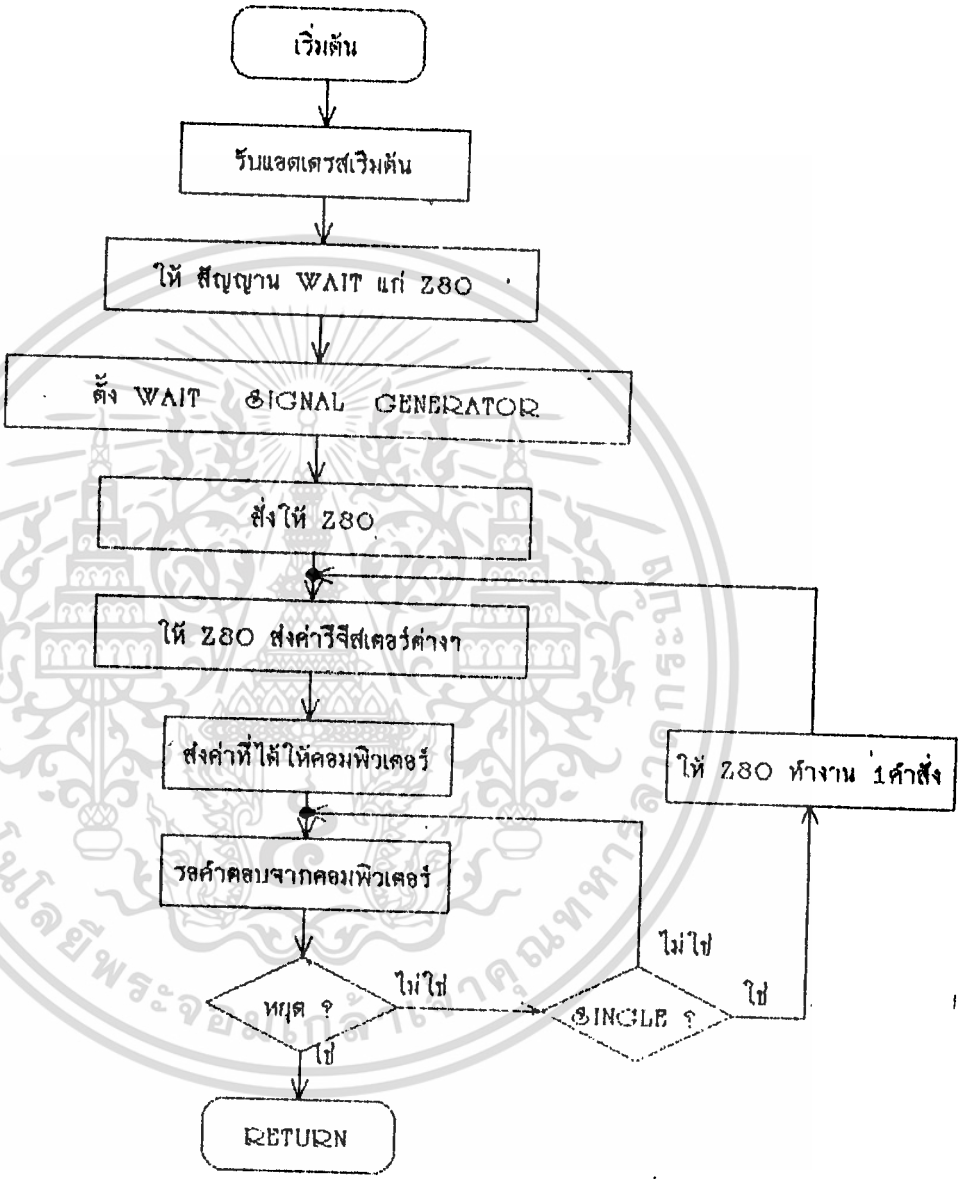



รูปที่ 3.7 แสดงการ UPLOAD




รูปที่ 3.8

คำสั่ง GO




รูปที่ 3.9 แสดงการทำชิง เกิลสเต็ป

Z-80 ทำคำสั่งจำพวกที่มีการย้ายข้อมูลจากรีจิสเตอร์ภายในผ่านบัสข้อมูลออกสู่ภายนอก เช่นคำสั่ง LD (FFF0),AF ขณะที่ค่ารีจิสเตอร์ถูกส่งผ่านดาต้าบัสนั้น ก็จะมีสัญญาณรอให้ Z-80 ทำให้บัสสถานะบัสจะคงอยู่อย่างนั้น จากนั้นส่วนพอรทอินพุท/เอาท์พุทก็จะอ่านบัสส่งให้ซีพียูควบคุม

### 3.2.1.7 คำสั่งไซเคิลสเต็ป

เป็นการให้ Z-80 ทำงานทีละ 1 แมชชีนไซเคิล ผังงานการทำงานเป็นเช่นเดียวกับคำสั่งชิงเกิลสเต็ป จะต่างกันตรงที่แบบไซเคิลเต็ปจะอ่านได้เฉพาะสถานะของบัสเท่านั้น ไม่สามารถอ่านค่ารีจิสเตอร์ขณะนั้นได้ และส่วนกำเนิดสัญญาณเวทจะอยู่ในโหมดไซเคิลสเต็ปคือจะให้ Z-80 ทำงานทีละ 1 แมชชีนไซเคิล ผังการทำงานแสดงดังรูปที่ 3.10

### 3.2.1.8 คำสั่งเบรกพอย์ท (break point)

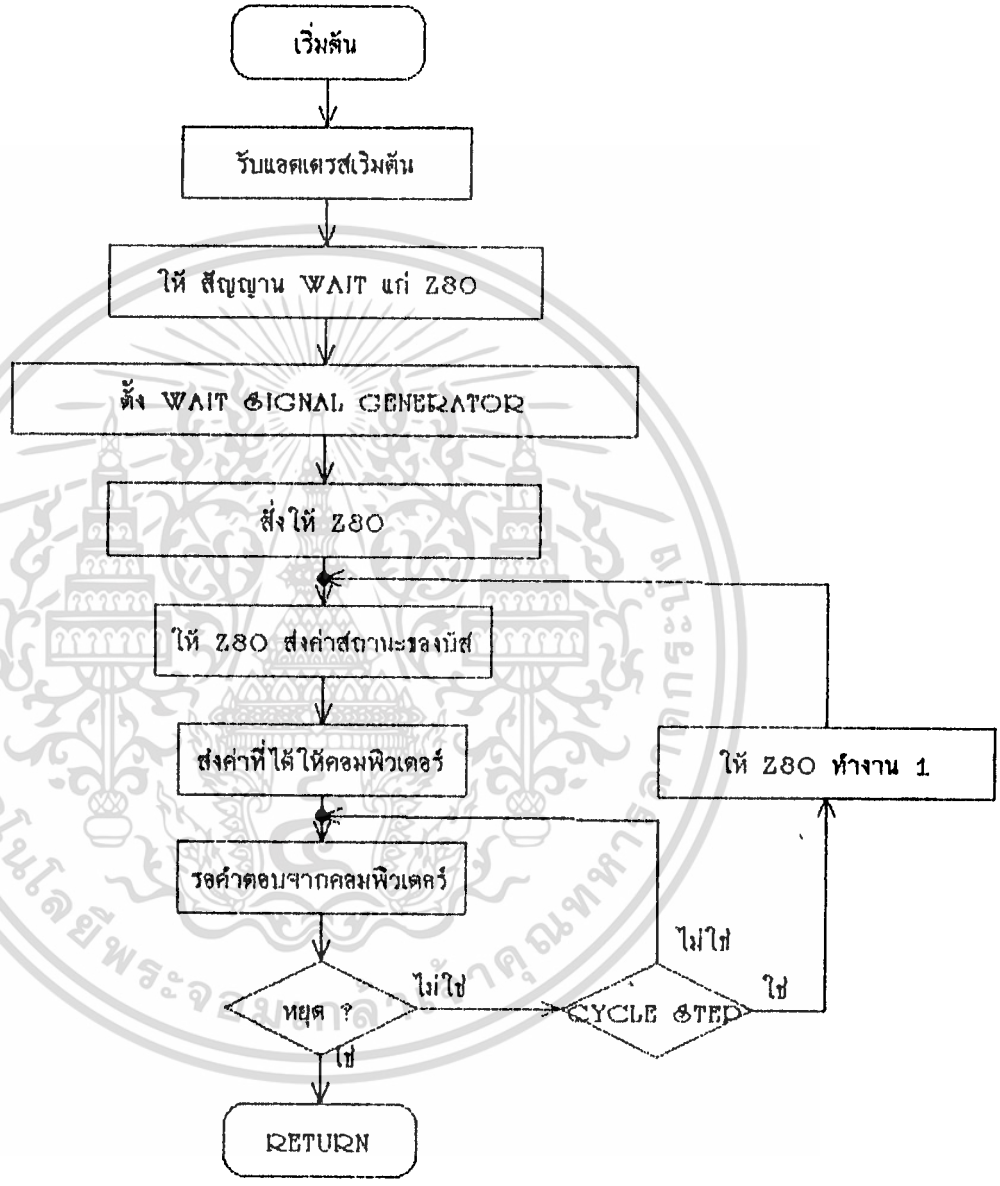
มีอยู่สองส่วนคือ ส่วนเขียนตั้งเบรคพอย์ท และส่วนลบเบรกพอย์ต ส่วนนี้จะเขียนค่า 1 บิตลงใน เบรคพอย์ทตำแหน่งตามที่ผู้ใช้กำหนด

### 3.2.1.9 คำสั่งเกี่ยวกับรีจิสเตอร์

จะแยกได้ 2 ชุด ชุดแรกจะเป็นชุดคำสั่งในรีจิสเตอร์ ชุดที่สองเป็นชุดคำสั่งแก้ไขค่าในรีจิสเตอร์ ชุดคำสั่งดูค่าในรีจิสเตอร์ จะเป็นการอ่านค่ารีจิสเตอร์ทั้งหมดของ Z-80 ในขณะนั้นจะคล้ายกับการขอดูค่ารีจิสเตอร์ ในคำสั่งชิงเกิ้ลสเต็ป ส่วนการแก้ไขค่าในรีจิสเตอร์นั้น ก็จะมีอยู่ 2 กลุ่มคือ กลุ่มแรกเป็นการแก้ไขค่ารีจิสเตอร์ขนาด 8 บิต อีกกลุ่มคือการแก้ไขค่ารีจิสเตอร์ขนาด 16 บิต เช่น IX,IY,PC

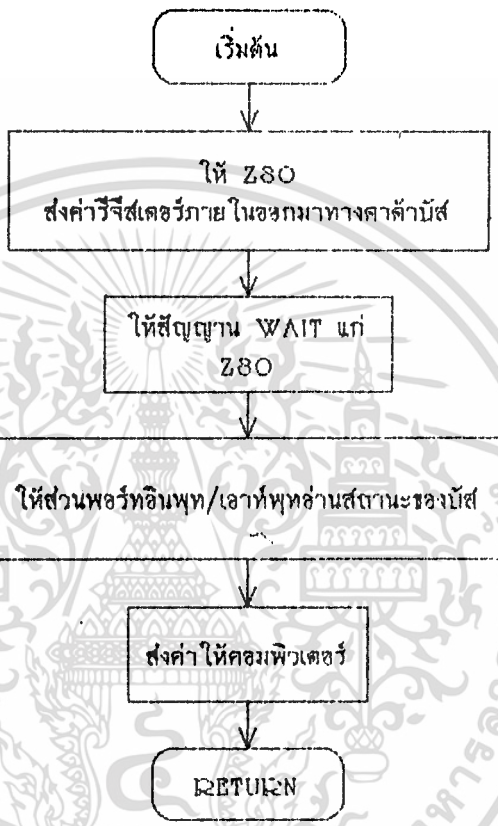
การแก้ไขทำได้เช่นเดียวกับการดู่าคือใช้เทคนิคการให้ Z-80 อ่านค่าข้อมูลภายนอกไปยังรีจิสเตอร์ภายใน ซึ่งข้อมูลภายนอกนั้นแทนที่ Z-80 จะอ่านมาจากส่วน บัฟเฟอร์ก็จะให้อ่านจากพอร์ทอินพุท/เอาท์พุทแทน

ผังงานการทำงานแสดงดังรูปที่ 3.11 , 3.12

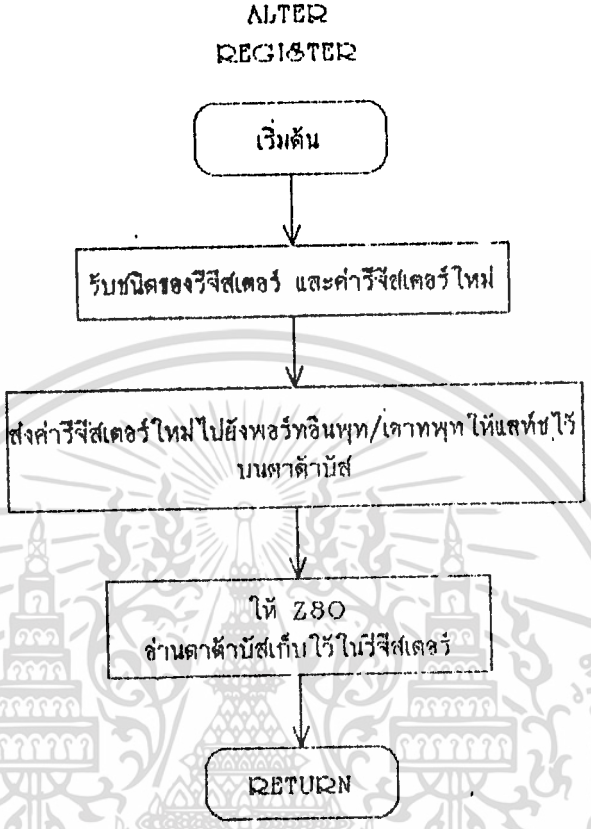



รูปที่ 3.10
แสดงการทำ
ไซเคิลสเต็ป

ดูค่าวีจีเอสเตอร์




รูปที่ 3.11
แสดงการดูค่าในวีจีเอสเตอร์




รูปที่ 3.13
แสดงการเปลี่ยนค่าในรีจีสเตอร์

3.2.1.10 คำสั่งเติมค่าลงในหน่วยความจำ (fill memory)

เนื่องจากรีจิสเตอร์ต่างๆในซีพียู MCS นั้นจะเป็นขนาดเป็น 8 บิท แต่ตำแหน่งเริ่มต้นและตำแหน่งสุดท้ายที่จะเติมนั้นเป็นขนาด 16 บิต ดังนั้นจึงมีเทคนิคโดยการนำเอาตัวชี้ข้อมูล (DPTR) มาช่วย

3.2.1.11 คำสั่งอินพุทพอร์ท/เอาท์พุทพอร์ท (input/output port)

จะเป็นการสั่งให้ซีพียู Z-80 ทำคำสั่ง IN หรือคำสั่ง OUT เพื่ออ่านค่าหรือเขียนค่าให้กับพอร์ท

3.2.2 โปรแกรมผู้ใช้ (USER PROGRAM)

เนื่องจากว่าอินเซอร์กิตอีมูเลเตอร์ไมโครโปรเซสเซอร์นี้เป็นอุปกรณ์ที่ช่วยและอำนวยความสดวกในการพัฒนาระบบที่ทำงานโดยใช้ไมโครโปรเซสเซอร์ ดังนั้นการติดต่อกับผู้ใช้จึงเป็นสิ่งที่สำคัญมาก เพราะว่าผู้ที่ใช้งานอีมูเลเตอร์ชุดนี้ย่อมต้องการติดต่อกับระบบที่กำลังพัฒนา (target system) อย่างใกล้ชิด ต้องการทราบค่าต่างๆในระบบที่กำลังพัฒนา สามารถอ่านค่าต่างๆออกมาได้อย่างสดวกในรูปแบบของข้อมูลที่เข้าใจง่าย อินเซอร์กิตอีมูเลเตอร์ชุดนี้ทำการติดต่อกับผู้ใช้งานโดยผ่านทางคอมพิวเตอร์ส่วนบุคคลในตระกูลไอบีเอ็มพีซี (IBM PC) การติดต่อกับคอมพิวเตอร์ส่วนบุคคลของอีมูเลเตอร์ชุดนี้ จะทำการติดต่อผ่านทางพอร์ทอนุกรม (Serial port) คอมพิวเตอร์ส่วนบุคคลในตระกูลไอบีเอ็มพีซีสามารถติดตั้งพอร์อนุกรมได้ถึง 4 พอร์ท แต่โดยทั่วๆไปจะทำการติดตั้งเพียงแค่สองพอร์ทเท่านั้น และซอฟท์แวร์ชุดนี้ก็สามารถทำการจัดการเลือกใช้พอร์ทได้ว่าจะให้ชุดอีมูเลเตอร์นี้ทำการติดต่อกับผู้ใช้ผ่านทางคอมพิวเตอร์ส่วนบุคคลทางพอร์ทอนุกรมที่เท่าไร

3.2.2.1 การติดต่อกับอีมูเลเตอร์

<u>พอร์ทอนุกรม</u> (SERIAL PORT หรือ COM PORT)

พอร์ตสื่อสารอนุกรมของไอบีเอ็มพีซี(IBM PC) พอร์สื่อสารเป็นส่วนสำคัญส่วนหนึ่งที่มีอยู่บนไมโครคอมพิวเตอร์ 16 บิต พอร์สื่อสารนี้มีชื่อเรียกอีกอย่างหนึ่งว่า คอม-พอร์ท (com-port) พอรต์นี้เป็นทางออกของข้อมูลที่ผู้ใช้มักจะกำหนดอยู่ในสเปกด้วย เสมอ

พอร์ตสื่อสาร RS232 บนเครื่องไมโครคอมพิวเตอร์ 16 บิต มีโครงสร้างที่สามารถโปรแกรมด้วยการส่งรหัสคำสั่งให้กับชิปหลักได้ อย่างไรก็ตามผู้ออกแบบได้ให้ทางเลือกในการสื่อสารด้วยกระแสวนรอบ (current loop) หรือแบบแรงดันคือ RS232 โดยใช้จัมเปอร์เพื่อเลือกระบบ สำหรับตัว RS232 นี้เป็นมาตรฐานแบบอะซิงโครนัสที่โปรแกรมสตาร์ทบิตสต็อปและพาริตี้บิต อัตราการส่งก็สามารถกำหนดได้ตั้งแต่ 50 บอดถึง 9600 บอด ลักษณะพิเศษของวงจรคือสามารถส่งสัญญาณมาอินเตอร์รัพซีพียูตามเงื่อนไขได้ และยังมีโครงสร้างฮาร์ดแวร์ป้อนกลับเพื่อใช้ในการตรวจสอบระบบว่าทำงานปกติดีหรือไม่ ได้อีกด้วย วงจรพอร์สื่อสารของไมโครคอมพิวเตอร์ 16 บิตนี้ใช้ไอซีหมายเลข 8250 เป็นตัวสำคัญของระบบ 8250 เป็นไอซีขนาด 40 ขา มีขีดความสามารถพิเศษดังนี้

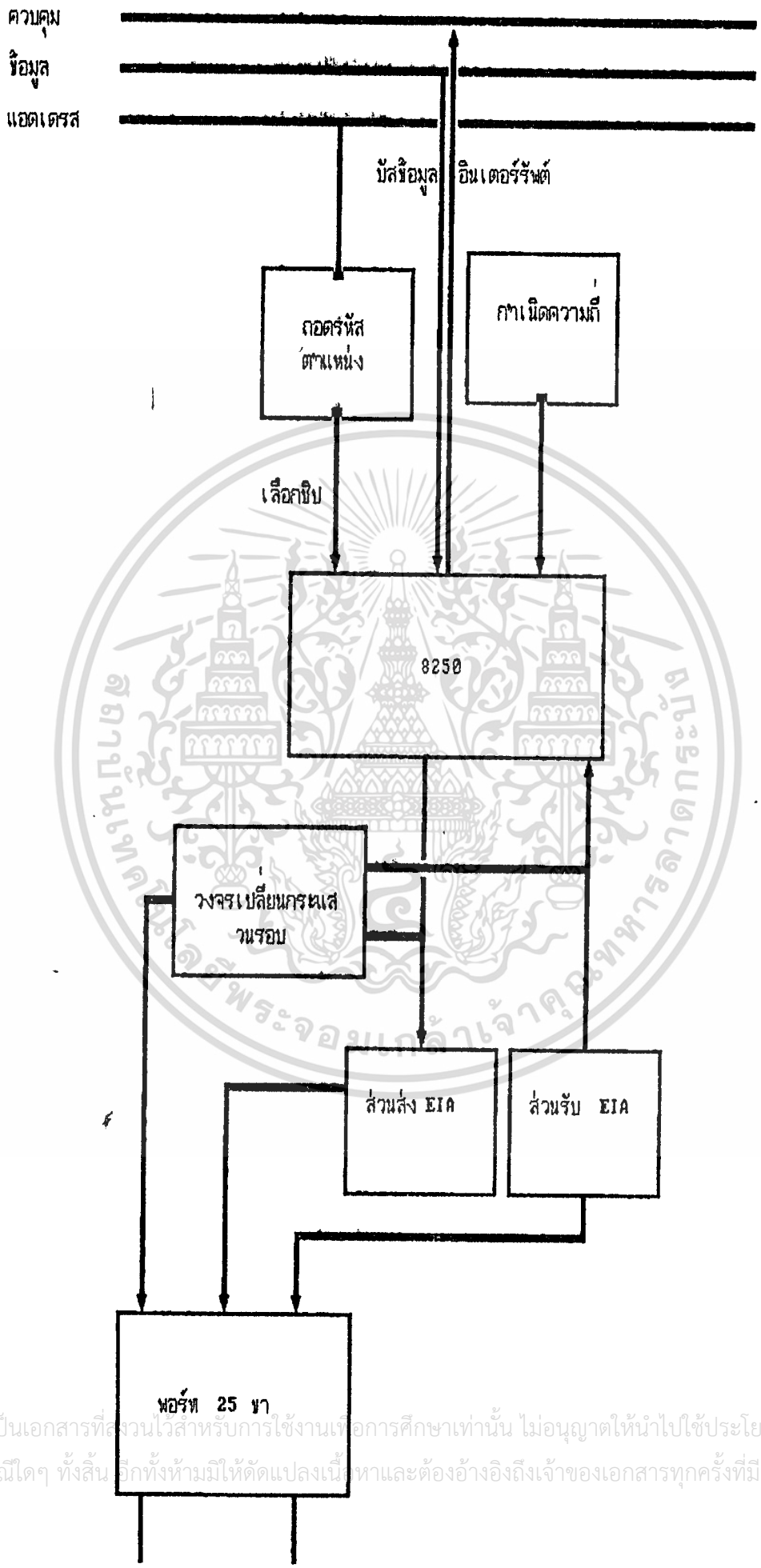
- มีบัฟเฟอร์ในตัวเพื่อทำให้ไม่จำเป็นต้องซิงโครนัสการรับส่ง
- ใช้สัญญาณนาฬิกาอิสระต่างหากไม่ขึ้นกับสัญญาณนาฬิกาของระบบ
- มีสัญญาณตอบรับควบคุมโมเดมทั้ง CTS (clear to send), RTS (request to send), DSR (data set ready), DTR (data terminal ready), RC (ring indicator) และสัญญาณดีเทคตัวพาหะ (carrier detector)
- ตรวจสอบสตาร์ทบิตที่ผิดพลาด
- ตรวจสอบและสร้างสัญญาณสายขาด เพื่อใช้ในการตรวจสอบการทำงานของระบบ

<u>โครงสร้างของพอร์สื่อสาร</u>

พอร์ตสื่อสารของเครื่องไมโครคอมพิวเตอร์ 16 บิตที่กล่าวถึงนี้เป็นระบบมาตรฐานตามแบบเครื่องไอบีเอ็มพีซีเอ็กซ์ที โครงสร้างบล็อกไดอะแกรมแสดงดังรูปที่ 3.13

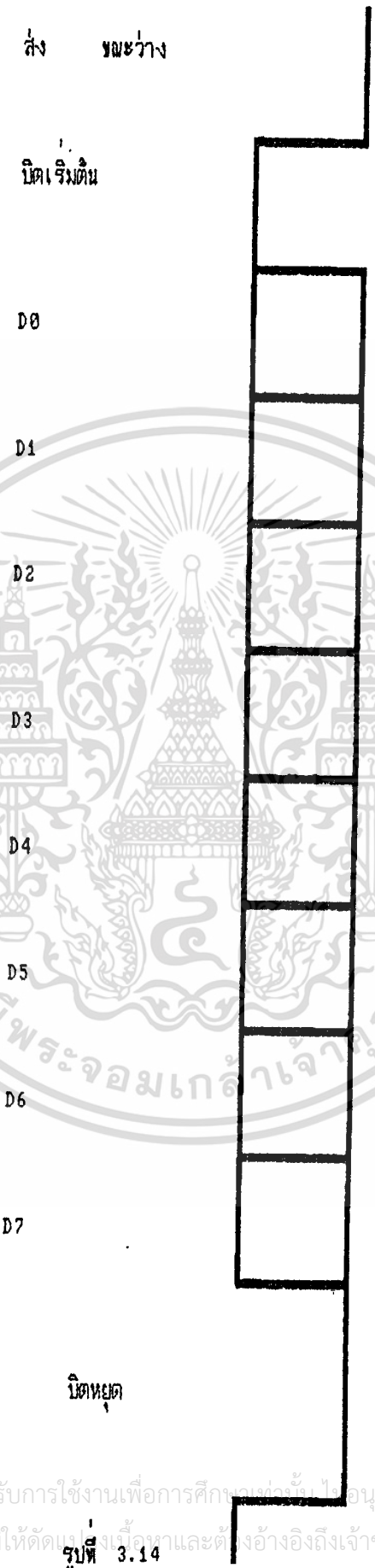
<u>รูปแบบของข้อมูลที่รับหรือส่ง</u>

การสื่อสารข้อมูลของระบบนี้เป็นการสื่อสารแบบอะซิงโครนัส รูปแบบของข้อมูลจะมีสตาร์ทบิต บิตตรวจสอบพาริตี้ และสต็อปบิต โครงสร้างของข้อมูลแต่ละเฟรมเป็นดังรูปที่ 3.14






รูปที่ 3.13 รูปตัวอย่างการต่อ 8250




รูปที่ 3.14

รูปแบบข้อมูล 1 เฟรม

เนื่องจากว่าพอร์ทอนุกรมนี้มีหัวใจการทำงานอยู่ที่ชิพ (chip) หมายเลข 8250 ซึ่งชิพหมายเลขนี้เป็นชิพที่สามารถโปรแกรมได้ เพราะว่ามีรีจิสเตอร์ภายใน รีจิสเตอร์ภายในของ 8250 มีดังนี้คือ

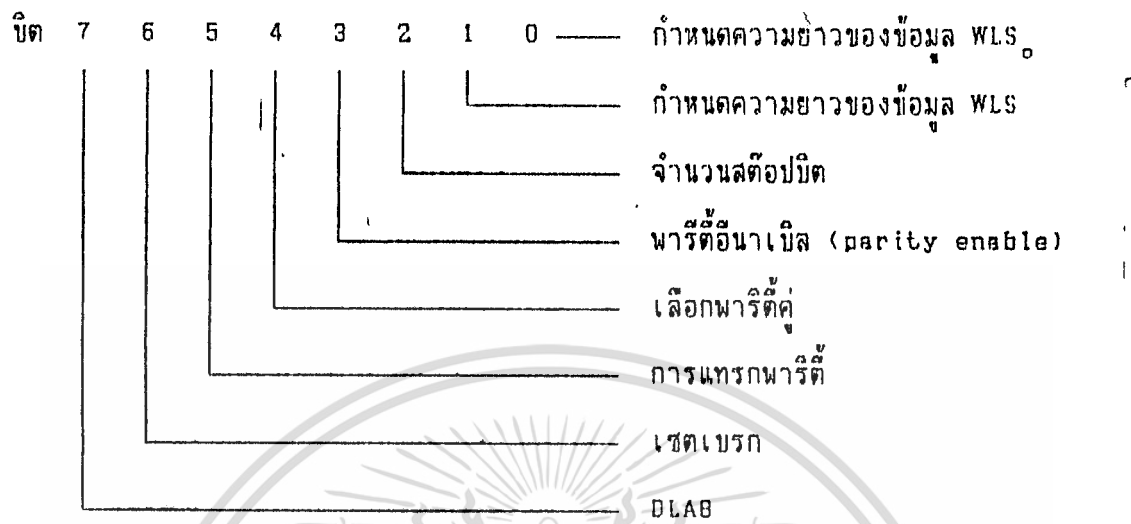
- บัฟเฟอร์สำหรับตัวรับข้อมูล (อ่าน)
- โฮลดิ้งสำหรับตัวส่งข้อมูล (เขียน)
- อีนาเบิลอินเตอร์รัพต์
- กำหนดอินเตอร์รัพต์
- ควบคุมสายสื่อสาร
- ควบคุมโมเดม
- แสดงสถานะสายสื่อสาร
- แสดงสถานะโมเดม
- แลตช์ตัวหาร (LSB)
- แลตช์ตัวหาร (MSB)

<u>การใช้งานรีจิสเตอร์บางตัวของ 8250</u>

ความหมายของรีจิสเตอร์ต่างๆ ของ 8250

**รีจิสเตอร์ควบคุมสายสื่อสาร** (line control register) ในการควบคุมรูปแบบของข้อมูลแบบอะซิงโครนัสนั้น ผู้โปรแกรมจะต้องกำหนดค่าลงในรีจิสเตอร์ควบคุมสายสื่อสาร รีจิสเตอร์ตัวนี้มี 8 บิต โดยแต่ละบิตมีความหมายดังนี้

พอร์ทแอดเดรสหมายเลข 3FB




รูปที่ 3.15 แสดงค่าของบิตในรีจิสเตอร์ควบคุมสายสื่อสาร

| บิต 0 | บิต 1 | ความหมาย |
|---|---|---|
| 0 | 0 | ข้อมูลขนาด 5 บิต |
| 0 | 1 | ข้อมูลขนาด 6 บิต |
| 1 | 0 | ข้อมูลขนาด 7 บิต |
| 1 | 1 | ข้อมูลขนาด 8 บิต |

รูปที่ 3.16 แสดงความหมายของบิตที่ 0 และ บิตที่ 1

- **บิต 2** เป็นบิตที่ใช้ในการกำหนดจำนวนสต๊อปบิต ถ้าเป็น "0" หมายถึงใช้สต๊อปบิต 1 บิต แต่ถ้าบิต 2 เป็น "1" ในกรณีส่งแบบ 5 บิตจะมีความยาวของสต๊อปบิตเป็น 1.5 บิต แต่ถ้าส่งแบบ 6,7,8 บิตความยาวของสต๊อปบิตจะเป็น 2
- **บิต 3** บิตนี้เป็นบิตแสดงการอีนาเบิลให้มีการตรวจสอบพาริตี้ โดยค่าบิตนี้มีค่าเป็น "1" จะมีการเพิ่มพาริตี้



- **บิต 4** มีค่าเป็น "0" และบิต 3 มีค่าเป็น "1" จะมีการกำหนดเป็นพาริตี้คี่ แต่ถ้าบิตนี้

มีค่าเป็น "1" จะเป็นพาริตี้คู่

- บิต 5 เมื่อบิต 3 มีค่าเป็น "1" และบิต 5 มีค่าเป็น "1" และบิต 4 มีค่าเป็น "1" จะมีการแทรกหรือตรวจสอบพาริตี้ ด้วยเงื่อนไขกำหนดให้เป็น "0" และถ้าบิต 4 มีค่าเป็น "0" บิต 3 มีค่าเป็น "1" และบิต 5 มีค่าเป็น "1" จะมีการกำหนดบิตพาริตี้เป็น "1"

- บิต 6 เป็นบิตที่ควบคุมการเบรก เมื่อบิต 6 มีค่าเป็น "1" ส่วนของ SOUT จะได้รับการกำหนดให้เป็น "0" ตลอด

- บิต 7 บิตนี้ทำหน้าที่เป็น DLAB บิตที่จะมีผลต่อการแลตช์ตัวหาร

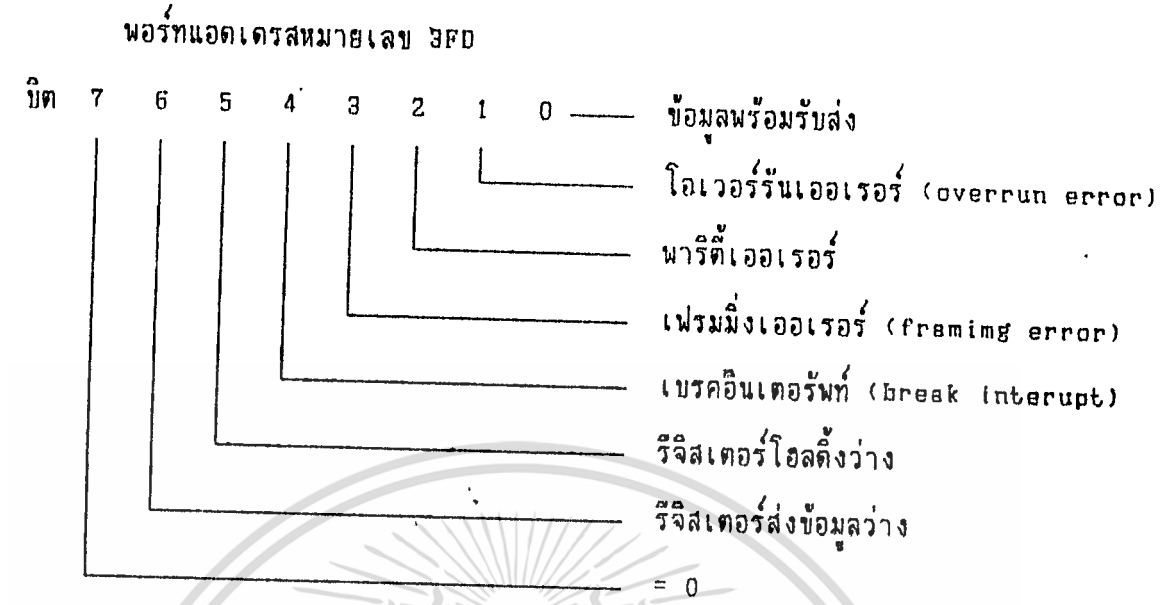
**รีจิสเตอร์แสดงสถานะสายสื่อสาร** (line status register)

รีจิสเตอร์ตัวนี้เป็นรีจิสเตอร์ที่จะให้ข้อมูลแก่ซีพียูเกี่ยวกับการสื่อสารข้อมูลในสายสื่อสาร ค่าของบิตต่างๆ ในรีจิสเตอร์เป็นดังนี้

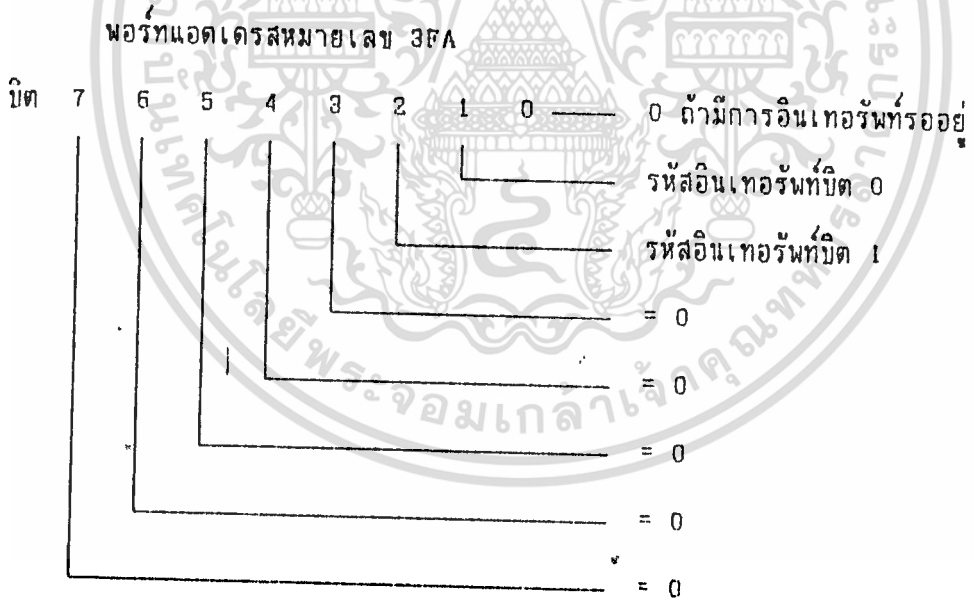
- บิต 0 บิตนี้เป็นบิตที่บอกสถานะการรับข้อมูล ถ้าบิตนี้เป็น "1"แสดงว่าการรับข้อมูลเข้ามาในบัฟเฟอร์ได้ครบทุกบิตแล้ว บิตนี้จะได้รับการรีเซตให้เป็น "0" เมื่อซีพียูได้อ่านข้อมูลในบัฟเฟอร์ไปแล้ว หรือจะให้ซีพียูเขียนข้อมูลกลับมายังรีจิสเตอร์นี้ก็ได้

- บิต 1 บิตนี้ถ้ามีค่าเป็น "1" แสดงว่าเกิด overrun error กล่าวคือขณะที่มีข้อมูลที่บัฟเฟอร์แต่ซีพียูยังไม่ได้อ่านไป ปรากฏว่ามีข้อมูลชุดใหม่มาเขียนทับบนบัฟเฟอร์นี้ บิตนี้จะรีเซตโดยซีพียู เมื่อซีพียูอ่านค่าจากรีจิสเตอร์นี้ไปแล้ว

- บิต 2 บิตนี้ถ้ามีค่าเป็น "1" แสดงว่าเกิดพาริตี้เออเรอร์ (parity error) กล่าวคือถ้ามีการตรวจสอบบิตพาริตี้แล้วไม่เป็นไปตามที่กำหนดไว้ บิตนี้จะได้รับการรีเซตโดยซีพียู เมื่อซีพียูอ่านค่าจากรีจิสเตอร์นี้ไปแล้ว




รูปที่ 3.17 ค่าของบิตในรีจิสเตอร์แสดงสถานะสายสื่อสาร




รูปที่ 3.18 ค่าของบิตในรีจิสเตอร์กำหนดอินเตอร์รัพต์

- **บิต 3** บิตนี้ถ้ามีค่าเป็น "1" แสดงว่าเฟรมของข้อมูลไม่เป็นไปตามที่กำหนด เช่นตรวจสอบจำนวนบิตโดยดูที่พาริตี้และสต๊อปบิตไม่เป็นไปตามที่กำหนด
- **บิต 4** บิตนี้เรียกว่า break interupt บิตนี้จะได้รับการเซตให้มีค่าเป็น "1" ถ้าหาก

ว่ารับข้อมูลอินพุตเป็น "0" เป็นเวลายาวนานกว่าเวิร์ดของการสื่อสาร

- บิต 5 บิตนี้เป็นบิตที่บอกว่า 8250 พร้อมที่จะรับข้อมูลจากสายสื่อสาร บิตนี้จะได้รับการเซตให้มีค่าเป็น "1" บิตนี้ยังคงสร้างสัญญาณอินเตอร์รัพต์เพื่อส่งไปบอกซีพียูด้วย บิตนี้จะมีสถานะเซตเมื่อมีการส่งถ่ายข้อมูลจากโฮลดรีจิสเตอร์ไปยังชิปรีจิสเตอร์เพื่อพร้อมที่จะส่ง
- บิต 6 เป็นบิตที่จะบอกว่าชิปรีจิสเตอร์ว่างเปล่า บิตนี้จะได้รับการเซตให้มีค่าเป็น 1 เพื่อบอกว่าพร้อมส่งแล้ว
- บิต 7 จะเป็น "0" ตลอด

<u>การโปรแกรมอัตราบอด</u> (boud rate generator)

อัตราบอดคืออัตราการเปลี่ยนแปลงของข้อมูลต่อหนึ่งหน่วยเวลาในการส่งข้อมูล ซึ่งต่างจากอัตราบิต เนื่องจากว่าอัตราบิตคืออัตราการผ่านจำนวนบิตของข้อมูลต่อหนึ่งหน่วยเวลา โดยที่อัตราบิตจะมีค่าต่ำกว่าอัตราบอด เนื่องจากว่าในข้อมูลหนึ่งชุดการเปลี่ยนแปลงของข้อมูลย่อมน้อยกว่าจำนวนของข้อมูลที่ส่งออกไป

อัตราบอดได้รับการกำหนดเทียบกับสัญญาณนาฬิกา 1.8432 MHz ซึ่งสัญญาณนาฬิกานี้เป็นสัญญาณนาฬิกาอิสระต่างจากสัญญาณนาฬิกาที่มาจากเมนบอร์ด (main board) ของคอมพิวเตอร์ส่วนบุคคล (personal computer) แต่จะมาจากวงจรสร้างสัญญาณนาฬิกาภายในชิป 8250 เอง และอัตราบอดนี้สามารถโปรแกรมได้โดยการกำหนดค่าของตัวหารได้ตั้งแต่ 1 ถึง $(2^{16}-1)$ค่าความถี่เอาต์พุทของตัวกำหนดอัตราบอดมีค่าเท่ากับ 16xอัตราบอด ดังนั้น ตัวหาร = ความถี่สัญญาณนาฬิกา หาร (อัตราบอดx16) การกำหนดอัตราบอดด้วยการกำหนดตัวหารนี้ตัวหารจึงเป็นค่ากำหนดในรีจิสเตอร์ 2 ตัว ตัวหารนี้จะต้องถูกกำหนดค่าก่อนแล้วโปรแกรมลงมาในรีจิสเตอร์นี้ การกำหนดต้องให้ DLAB = "1" แล้วให้ลงมาในรีจิสเตอร์ 3F8 ซึ่งเรียงกันเป็น LSB ของตัวหารส่วน 3F9 เมื่อ DLAB = "1" จะเป็นค่าของตัวหาร MSB ค่าของตัวหารเมื่อเทียบกับสัญญาณ 1.8432 MHz เป็นดังตารางในรูปที่ 3.19

| อัตราบอด | ตัวหาร | | ค่าผิดพลาด |
|---|---|---|---|
| | ฐานสิบ | ฐานสิบหก | |
| 50 | 2304 | 900 | - |
| 75 | 1536 | 600 | - |
| 110 | 1047 | 417 | 0.026 |
| 134.5 | 857 | 359 | 0.058 |
| 150 | 768 | 300 | - |
| 300 | 384 | 180 | - |
| 600 | 192 | 0C0 | - |
| 1200 | 96 | 060 | - |
| 1800 | 64 | 040 | - |
| 2000 | 58 | 03A | 0.69 |
| 2400 | 48 | 030 | - |
| 3600 | 32 | 020 | - |
| 4800 | 24 | 018 | - |
| 7200 | 16 | 010 | - |
| 9600 | 12 | 00C | - |

รูปที่ 3.19 ค่าตัวหารสำหรับการกำหนดอัตราบอด

<u>การใช้งานพอร์ตอนุกรม</u>

พอร์ทสื่อสารนี้จะทำหน้าที่ในการส่งและรับข้อมูลแบบอนุกรมอซิงโครนัส ก่อนจะทำการรับส่งข้อมูลนั้นจะต้องมีการโปรแกรมการทำงานของพอร์ทก่อน โดยการโปรแกรมรีจิสเตอร์ภายในชิปหมายเลข 8250 สิ่งที่จะต้องทำการโปรแกรมก็คือ ขนาดของข้อมูล ขนาดของสตาร์ท

บิต ขนาดของสต๊อปบิต พาริตี้ และบอดเรท โฟลว์ชาร์ทแสดงการทำงานของพอร์ตอนุกรมนี้แสดงได้ดังรูปที่ 3.20

**การโปรแกรมพอร์ทอนุกรม**

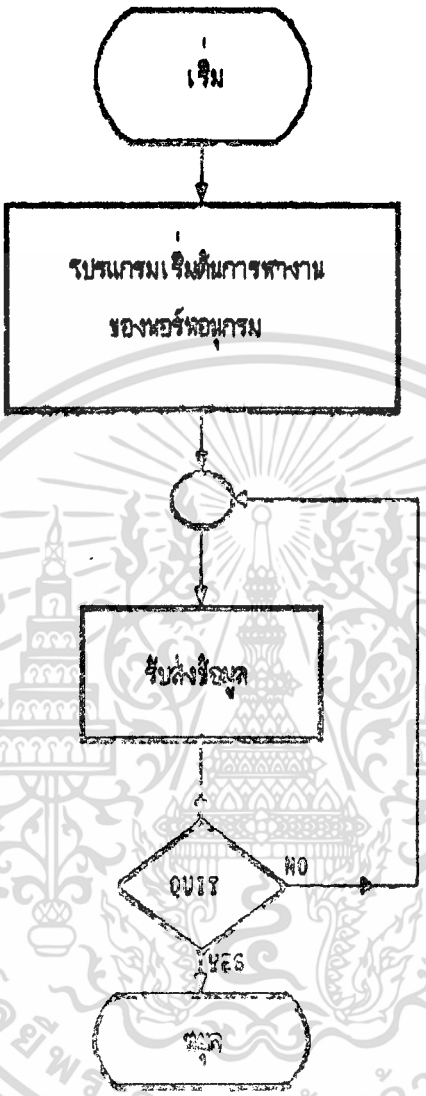
- ***การโปรแกรมอัตราเบด*** เนื่องจากว่าตัวหาร เป็นข้อมูลขนาด 2 ไบท์ (byte) ดังนั้นจึงต้องใช้ที่เก็บตัวหารขนาด 2 ไบท์ หรือรีจิสเตอร์ 2 ตัว โดยที่พอร์ท 3F8 และ 3F9 ในขณะที่ DLAB = "1" ก็คือทางที่จะติดต่อกับรีจิสเตอร์ที่เก็บค่าตัวหาร ซึ่ง พอร์ท 3F8 จะเก็บค่าในไบท์ที่มีนัยสำคัญต่ำกว่า และ พอร์ท 3F9 จะเก็บค่าในไบท์ที่มีนัยสำคัญสูงกว่า

- ***การโปรแกรมจำนวนสตาร์ทบิต*** จะโปรแกรมได้โดยการให้ค่ากับบิตที่ 2 ของรีจิสเตอร์ควบคุมสายสื่อสารตามความหมายของบิตต่างๆ ของรีจิสเตอร์นี้ดังกล่าวแล้วข้างต้น ซึ่งมีความสัมพันธ์กันกับการเลือกใช้ขนาดของข้อมูลด้วย

- ***การโปรแกรมขนาดของข้อมูล*** จะโปรแกรมได้โดยการให้ค่ากับบิตที่ 0 และ 1 ของรีจิสเตอร์ควบคุมสายสื่อสาร ซึ่งค่าของบิตทั้งสองและจำนวนสต๊อปบิตแสดงดังตาราง

| บิต 0 | บิต 1 | ความหมาย |
|---|---|---|
| 0 | 0 | ข้อมูลขนาด 5 บิต |
| 0 | 1 | ข้อมูลขนาด 6 บิต |
| 1 | 0 | ข้อมูลขนาด 7 บิต |
| 1 | 1 | ข้อมูลขนาด 8 บิต |

- ***การโปรแกรมขนาดของสต๊อปบิต*** กำหนดโดยการให้ค่ากับรีจิสเตอร์ควบคุมสายสื่อสารในบิตที่ 2




รูปที่ 3.20 แสดงผังการทำงานของโปรแกรมเชิงงานพอร์ตอนุกรม

- ***การโปรแกรมการเลือกใช้พาริตี้*** มีลักษณะการให้ค่ากับบิตต่างๆ ของรีจิสเตอร์ควบคุมสายสื่อสารตามความหมายของบิตต่างๆของรีจิสเตอร์นี้ดังที่ได้กล่าวไว้แล้วข้างต้น

การที่จะติดต่อกับรีจิสเตอร์ควบคุมสายสื่อสารนี้ได้ ก็คือการติดต่อผ่านทางพอร์ตของเครื่องคอมพิวเตอร์ โดยที่ในการติดต่อกับพอร์ตอนุกรมจะมีการติดต่อผ่านทางหมายเลขพอร์ตตั้งแต่ พอร์ตหมายเลข 3F8 ถึง 3FE สำหรับพอร์ตคอม 1 และหมายเลขพอร์ต 2F8 ถึง 2FE สำหรับพอร์ตคอม 2 ในการติดต่อกับรีจิสเตอร์ควบคุมสายสื่อสารนั้นจะติดต่อผ่านทางหมายเลขพอร์ตที่มี สายแอดเดรส $A_2$ $A_1$ $A_0$ เป็น 0 1 1 ตามลำดับ ดังนั้นการจะติดต่อกับรีจิสเตอร์ควบคุมสายสื่อสารได้นั้นก็โดยการติดต่อผ่านทางพอร์ตหมายเลข 3F8 + 3 สำหรับพอร์ตคอม 1 และผ่านทางพอร์ตหมายเลข 3F9 + 3 สำหรับพอร์ตคอม 2

- ***ตัวอย่างโปรแกรมการเริ่มต้นคอมพอร์ต*** ตัวอย่างนี้เป็นการโปรแกรมเพื่อควบคุมพอร์ตสื่อสาร แบบง่าย ๆ โดยไม่มีการตรวจสอบสัญญาณ โปรแกรมตัวอย่างนี้เริ่มจากการกำหนดค่าเริ่มต้นให้กับ 8250 ใหม่ โดยการกำหนดอัตราบอดให้ใหม่ ในที่นี้ใช้ตัวหารเป็น 384 คือกำหนดเป็น 300 บอด สังเกตว่าโปรแกรมเซตพอร์ต 3FB ซึ่งเป็นรีจิสเตอร์ควบคุมให้บิต 7 เป็น 1 ก่อนที่จะส่งตัวหารไป ส่วนพอร์ต 3FB ส่วนสุดท้ายที่ส่งรหัส 00000011 ไปก็คือกำหนดพอร์ตแมตของข้อมูลที่ส่งเป็นแบบ 8 บิตไม่มีพาริตี้

```
MOV   DX,3F8 + 3 ; CONTROL REGISTER
MOV   AL,80H
OUT   DX,AL
MOV   AX,384
MOV   DX,3F8
OUT   DX,AL
MOV   AL,AH
INC   DX
OUT   DX,AL
```

```
MOV   DX,3F8 + 3
MOV   AL,00000011B
OUT   DX,AL
```

- ***ตัวอย่างการโปรแกรมการส่งข้อมูล*** รีจิสเตอร์แสดงสถานะของสายสื่อสารคือ พอร์ต 3FD ในเลขฐานสิบหก มีข้อมูลแสดงสถานะของบัฟเฟอร์ตัวส่ง ในการใช้งานจะส่งข้อมูลออกไปเมื่อบัฟเฟอร์ว่างเท่านั้น ซึ่งจะทราบได้ว่าบัฟเฟอร์นั้นว่างหรือไม่ได้จากการตรวจสอบบิตที่ 5 ตัวอย่างโปรแกรมซึ่งทำการส่งอักษร "A" มีดังนี้

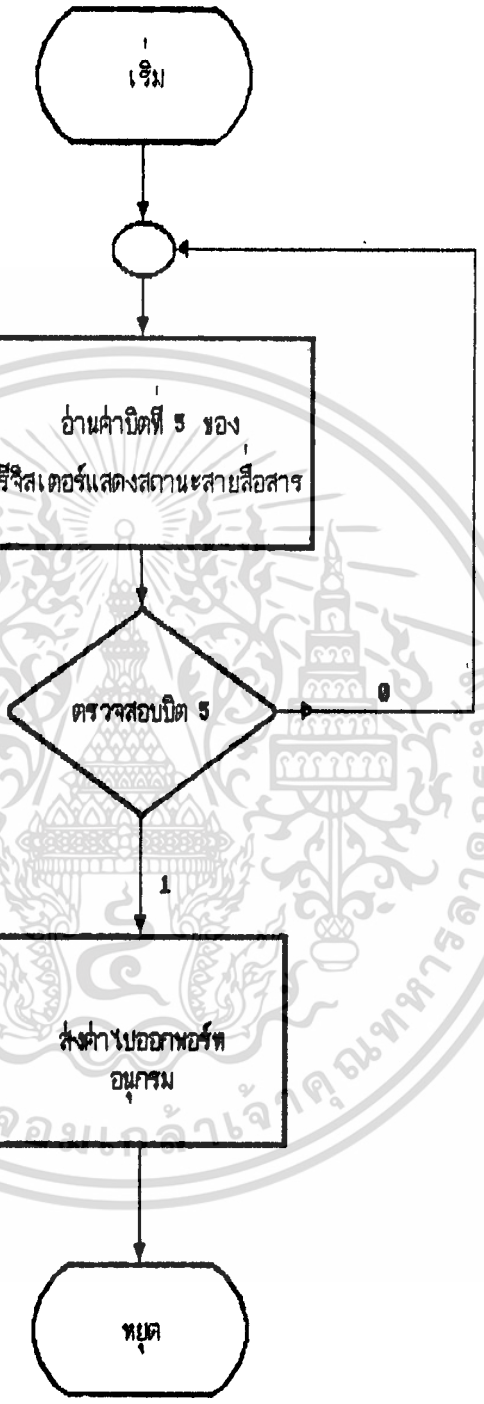
```
SEND  :   IN    AL,DX
          TEST  AL,20H
          JZ    SEND
          MOV   AL,'A'
          MOV   DX,3F8
          OUT   DX,AL
```

- ***ตัวอย่างการโปรแกรมการรับข้อมูล*** ในการรับข้อมูลจะทำการรับก็ต่อเมื่อมีข้อมูลเข้ามาในบัฟเฟอร์แล้วเท่านั้น จะทราบได้จากการตรวจสอบบิต 0 ของรีจิสเตอร์แสดงสถานะสายสื่อสาร ตัวอย่างโปรแกรมที่ทำหน้าที่รับข้อมูล 1 ตัวอักษรมีดังนี้

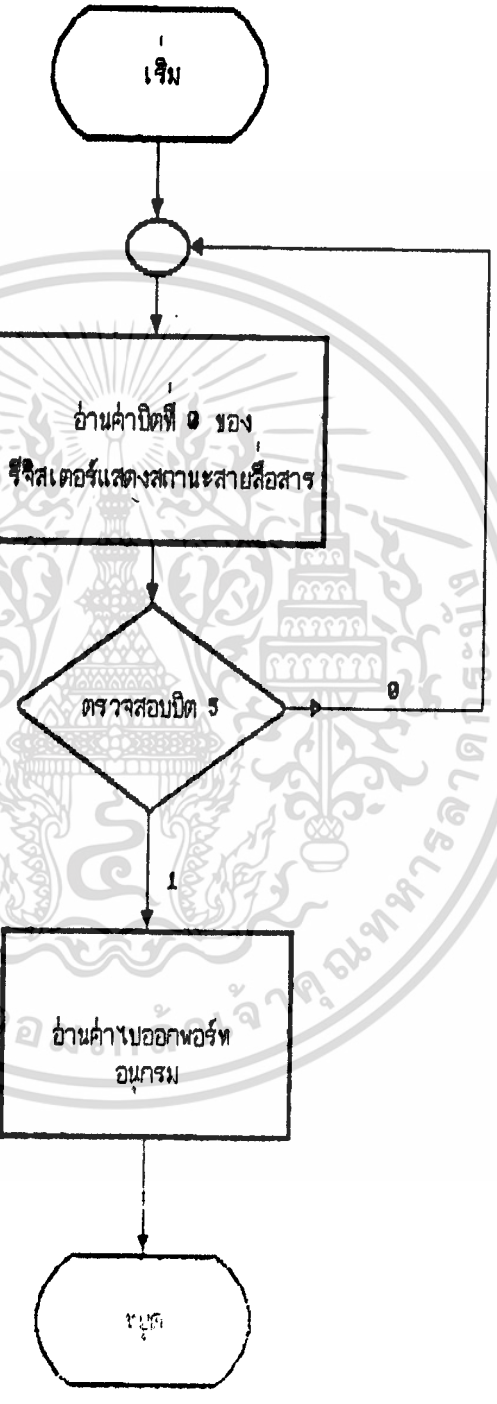
```
RECV  :
          IN    AL,DX
          TEST  AL,1
          JZ    RECV
          MOV   DX,3F8
          IN    AL,DX
```



โฟล์วชาร์ทแสดงการใช้งานพอร์ทอนุกรมในการส่งข้อมูลและการรับข้อมูล แสดงดังรูปที่ 3.21 และ 3.22




รูปที่ 3.21 แสดงแผนผังการทำงานของโปรแกรมส่งข้อมูล




รูปที่ 3.22 แสดงแผนผังการทำงานของโปรแกรมรับข้อมูล

3.2.2.2 การติดต่อกับผู้ใช้ (User interface)

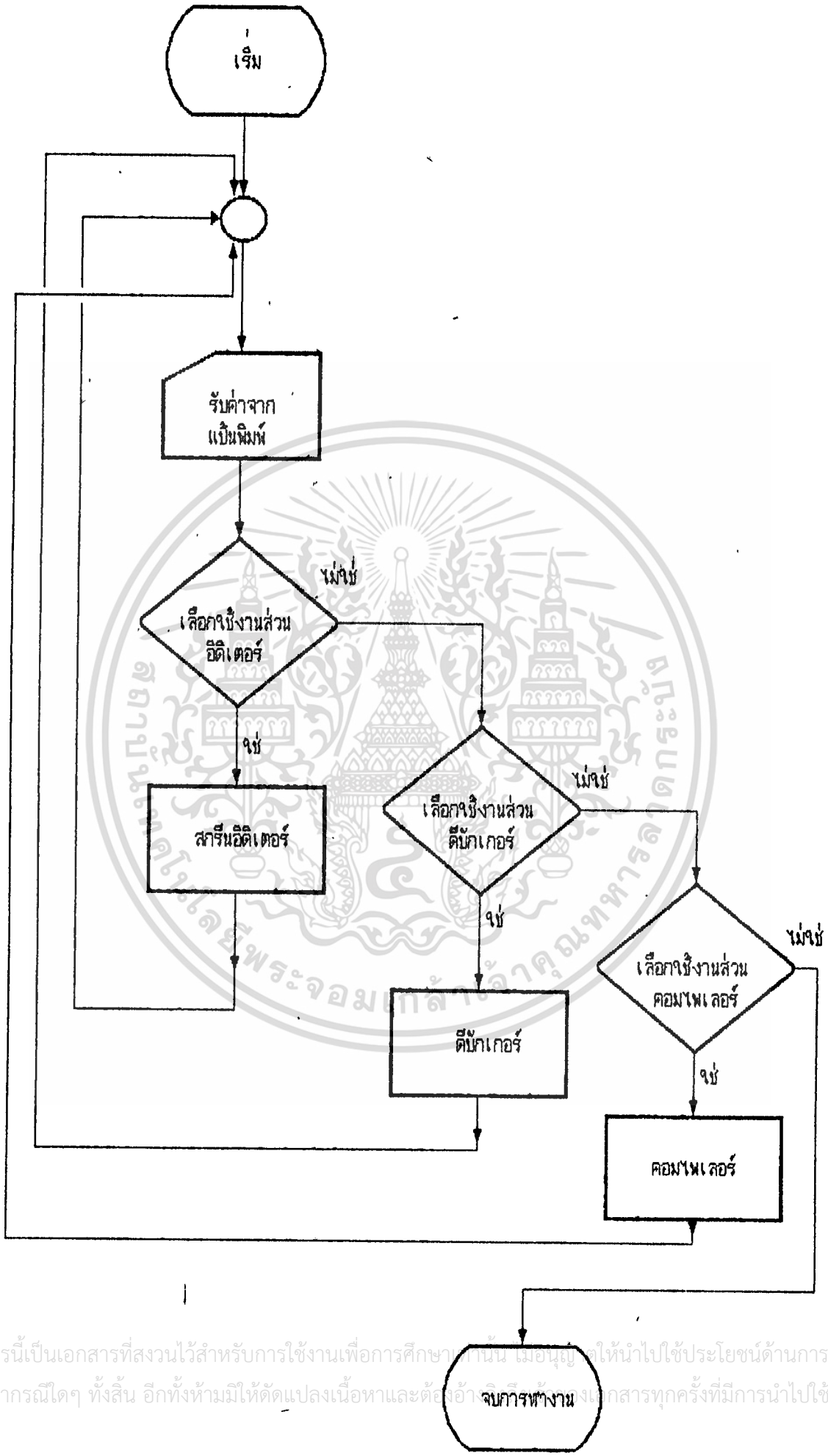
เนื่องจากว่าอีมูเลเตอร์นี้เป็นอุปกรณ์ที่ทำหน้าที่ช่วย และอำนวยความสดวกในงานพัฒนาระบบที่ทำงานโดยใช้ไมโครโปรเซสเซอร์ส่วน ที่ทำหน้าที่ในการอำนวยความสดวกในการติดต่อกับชุดอีมูเลเตอร์นี้ก็คือซอฟท์แวร์ที่ทำการติดต่อกับผู้ใช้ เป็นส่วนที่ทำให้ผู้ใช้สามารถอ่านและตีความหมายค่าต่างๆภายในระบบที่กำลังพัฒนา หรือผลลัพธ์ที่เกิดขึ้นได้โดยง่าย และเพื่อเป็นการอำนวยความสดวกอย่างแท้จริง ซอฟท์แวร์นี้จึงได้รับการพัฒนาขึ้นมาโดยมีอรรถประโยชน์หลายอย่างคลอบคลุมการทำงานทั้งหมดในการพัฒนาระบบที่ทำงานโดยใช้ไมโครโปรเซสเซอร์ ซึ่งซอฟแวร์นี้มีโครงสร้างดังในรูปที่ 3.23 ซึ่งประกอบไปด้วย

- อีดิเตอร์ (EDITOR)
- คอมไพเลอร์ (COMPILER)
- ดีบักเกอร์ (DEBUGGER)

โดยที่ซอฟท์แวร์ทั้งสามส่วนนี้จะวางตัวอยู่ภายใต้โปรแกรมหลักที่ทำหน้าที่ในการเลือกใช้โปรแกรมย่อยเหล่านี้อีกทีหนึ่ง รายละเอียดของส่วนต่างๆ มีดังนี้

<u>อีดิเตอร์</u> ส่วนนี้เป็นสกรีนอีดิเตอร์ (screen editor) หรือ อีดิเตอร์แบบเต็มจอภาพ จุดประสงค์ก็เพื่อใช้ในการเขียนซอร์สโคด (source code) ซึ่งเป็นภาษาแอสเซมบลี (assembly) โดยทั่วไปการเขียนซอร์สโคดจะต้องใช้สกรีนอีดิเตอร์จากภายนอก แล้วจึงนำซอร์สโคดที่เขียนขึ้นไปทำการคอมไพล์ (compile) โดยคอมไพเลอร์ที่ทำหน้าที่คอมไพล์ซอร์สโคดนั้นๆ จากนั้นจึงทำการโหลด (load) ผลที่ได้จากการคอมไพล์ลงในระบบที่กำลังทำการพัฒนา ซึ่งเป็นการยุ่งยากและมีหลายขั้นตอน แต่สำหรับระบบซอฟแวร์ตัวนี้ที่มีอีดิเตอร์ภายในตัว สามารถที่จะทำการเขียนซอร์ซโคท และทำการคอมไพล์ได้ทันที

หลักการของการทำงานของสกรีนอีดิเตอร์ตัวนี้ก็คือ การมองข้อมูลในหน่วยความจำโดยมีพอยเตอร์ (pointer) เป็นตัวชี้ตำแหน่งของข้อมูล และการเรียงต่อกันของข้อมูลในหน่วยความจำสามารถที่จะอ้างถึงได้โดยการใช้ตัวแปรแบบพอยเตอร์ในภาษาชั้นสูงต่างๆ เช่น ภาษาซี (C LANGUAGE) , ภาษาปาสคาล (PASCAL) ฯลฯ หรืออาจใช้วิธีการสร้างพอยเตอร์ขึ้นเองโดยใช้ภาษาแอสเซมบลี และให้การกดคีย์ (key) ทุกตัวเป็นคำสั่งในการควบคุมการทำงานของโปรแกรม เช่น เมื่อมีการกดคีย์ตัวอักษรก็ให้หมายถึงคำสั่งให้เก็บค่าของคีย์ที่กดนั้นเข้าไว้เป็นส่วนหนึ่งของข้อ






รูปที่ 3.23 โครงสร้างซอร์ฟแวร์การติดต่อกับผู้ใช้

มูลในตำแหน่งการเรียงตัวของข้อมูลในหน่วยความจำที่เหมาะสม หรือถ้ามีการกดคีย์ เอสเคป (escape) ก็ให้หมายถึงให้จบการทำงานของโปรแกรม หรือถ้ามีการกดคีย์คอนโทรลเอส (control s) ก็หมายถึงการให้เก็บข้อมูลใส่ไว้ในแฟ้มข้อมูล ฯลฯ

ดีบัก เกอร์ เป็นส่วนที่สำคัญที่สุดของซอฟแวร์ชุดนี้ เนื่องจากว่าผู้ใช้จะทำการติดต่อกับระบบที่กำลังทำการพัฒนาอยู่นั้นโดยผ่านทางซอฟแวร์ส่วนดีบัก เกอร์นี้ หน้าที่หลักของดีบัก เกอร์นี้จะทำการส่งผ่านข้อมูลระหว่างผู้ใช้กับระบบปลายทางโดยผ่านทางพอร์ทอนุกรมเข้าไปสู่ ส่วนที่ทำหน้าที่รับข้อมูลซึ่งมีไมโครคอนโทรลเลอร์หมายเลข 8032 ทำหน้าที่รับข้อมูลและส่งผ่านข้อมูลไปยังระบบปลายทางอีกทีหนึ่ง หรืออาจคอยรับข้อมูลจากระบบปลายทางมาแสดงผลให้กับผู้ใช้

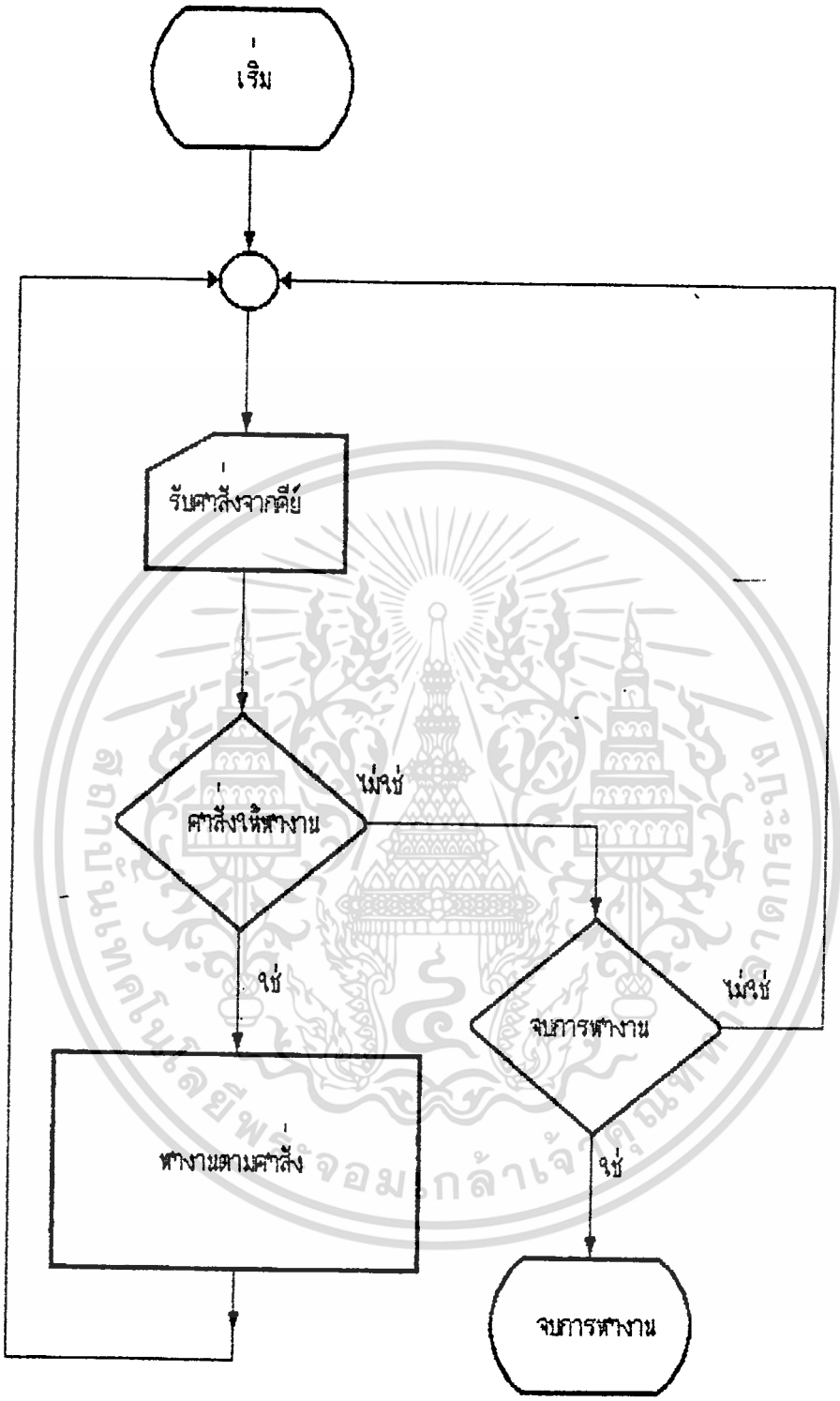
ซอฟแวร์ชุดนี้สามารถเลือกติดต่อได้กับพอร์ตอนุกรมทั้งคอม1 และคอม2 โครงสร้างของส่วนดีบัก เกอร์นี้แสดงได้ดังรูปที่ 3.24 ซึ่งประกอบด้วยส่วนต่างๆที่สำคัญดังนี้

- รีเซท (RESET) คือคำสั่งให้ทำการรีเซททาร์เก็ทซิสเต็ม ทำงานโดยการส่งรหัสรีเซทขนาด 1 ไบต์ไปยังไมโครคอนโทรลเลอร์หมายเลข 8032 ซึ่งทำหน้าที่ในการรับและ 8032 นี้จะทำการรีเซ็ททาร์เก็ตซิสเต็ม
- อัพโหลด (UPLOAD) คือการส่งข้อมูลจากทาร์เก็ทซิสเต็ม มายังคอมพิวเตอร์ ทำงานโดยการส่งรหัสขนาด 1 ไบต์ไปยังไมโครคอนโทรลเลอร์ 8032 ให้ทำการสั่งให้ทาร์เก็ตซิสเต็มส่งข้อมูลในหน่วยความจำออกมาและ 8032 นี้จะส่งผ่านข้อมูลนี้มายังคอมพิวเตอร์เพื่อแสดงผลให้กับผู้ใช้
- ดาวน์โหลด (DOWNLOAD) คือการส่งแฟ้มข้อมูลลงไปยังทาร์เก็ทซิสเต็ม ทำงานโดยที่คอมพิวเตอร์ส่งโคด (code) ขนาด 2 ไบต์ให้กับไมโครคอนโทรลเลอร์ โดยที่ไบต์แรกคือคำสั่งให้รอดาวน์โหลด อีกไบต์คือจำนวนไบต์ของข้อมูลที่จะส่งไป
- รีจิสเตอร์ (REGISTER) คือการตรวจสอบค่าในรีจิสเตอร์ต่างๆ ในทาร์เก็ทซิสเต็ม ทำงานโดยการสั่งให้ทาร์เก็ตซิสเต็มส่งค่าต่างๆในรีจิสเตอร์ออกมาโดยผ่านทางไมโครคอนโทรลเลอร์ในรูปของกลุ่มข้อมูล เมื่อคอมพิวเตอร์รับกลุ่มข้อมูลนี้ได้ก็จะทำการแยกแยะและนำมาแสดงผลให้ได้ตรงกับชนิดของข้อมูลนั้นๆ
- ซิงเกิลสเต็พ (SINGLE STEP) คือการสั่งให้ทาร์เก็ทซิสเต็มทำงานที่ละคำสั่ง การทำงานของคำสั่งนี้ก็คือ คอมพิวเตอร์จะส่งคำสั่งไปยังไมโครคอนโทรลเลอร์เพื่อให้ไม

โครคอนโทรลเลอร์รับรู้และสั่งให้ทาร์เก็ตซิสเต็มทำการรีเซ็ตซิสเต็ม โดยการไปเซ็ทอินเทอรัพท์ของทาร์เก็ตซิสเต็ม

- ไซเคิลสเต็พ (CYCLE STEP) คือการสั่งให้ทาร์เก็ตซิสเต็มทำงานที่ละมาชีนไซเคิล (machine cycle)
- ดัมพ์ (DUMP) คือการนำค่าของหน่วยความจำในตำแหน่งที่ต้องการออกมาแสดงผลทางจอภาพ คำสั่งนี้จะทำงานโดยการส่ง คำสั่งขนาด 1 ไบต์ออกไปเพื่อให้ไมโครคอนโทรลเลอร์ทราบว่าเป็นคำสั่งดัมพ์หน่วยความจำ จากนั้นจะรอรับข้อมูลอีก 3 ไบต์จากคอมพิวเตอร์ ซึ่ง 3 ไบต์นี้ก็คือ จำนวนไบต์ของข้อมูลที่ต้องการดัมพ์ และตำแหน่งของหน่วยความจำที่เป็นตำแหน่งเริ่มต้นของการดัมพ์
- เบรก (BREAK) คือการสั่งให้ทาร์เก็ตซิสเต็มหยุดทำงานชั่วคราว
- โก (GO) เป็นคำสั่งที่สั่งให้ทาร์เก็ตซิสเต็มทำงานตามคำสั่งที่มีอยู่ในหน่วยความจำในตำแหน่งที่อินสรักชั่นพอยเตอร์ (instruction pointer) ชี้อยู่
- เอาท์พุทพอร์ท (OUTPUT PORT) คือคำสั่งให้ส่งค่าจากคอมพิวเตอร์ไปยังพอร์ทของทาร์เก็ตซิสเต็ม
- อินพุทพอร์ต (INPUT PORT) คือการรับค่าจากพอร์ทของทาร์เก็ตซิสเต็มเข้ามาแสดงผลบนไมโครคอมพิวเตอร์
- แอสเซมบลี คือการแปลรหัสนีโมนิค (MNEMONIC) ที่รับค่าเข้าไปทางเทอร์มินอลของไมโครคอมพิวเตอร์ให้เป็นออปเจ็กโคด (Object code) และทำการเก็บค่าไว้ในหน่วยความจำของไมโครคอมพิวเตอร์
- อันแอสเซมบลี (UNASSEMBLY) คือการแสดงผลของข้อมูลในหน่วยความจำในรูปของเลขฐานสิบหกและรหัสแอสกี้ (ASCII)

จะเห็นว่าการทำงานหลักของซอฟท์แวร์นี้คือการติดต่อรับส่งข้อมูลผ่านทางพอร์ตอนุกรม




รูปที่ 3.24

ผังงานแสดงการทำงานของส่วนดีบักเกอร์

# บทที่ 4
# การทดลองและผลการทดลอง

เพื่อให้ผลการทดลองเป็นผลที่สามารถเห็นได้ชัดเจน จึงสร้างวงจรชุด target system ขึ้นมา ดังวงจร

หลังจากที่ทำการทดสอบ วงจรที่ออกแบบเป็นส่วน ๆ จนได้ผลเป็นที่พอใจแล้ว ผลการทดลองที่จะกล่าวต่อไปนี้ จึงเป็นการทดลองกับ target board ที่ออกแบบขึ้นมาเพื่อใช้ทดสอบ สำหรับโครงงานชุดนี้

(D)DOWNLOAD

การโหลดไฟล์แบบอินเทล์ฟอร์แมท์ลงบนหน่วยความจำอีมูเลชั่น (emulation memory) การทดสอบส่วนการ dump memory นั้นจะมีโปรแกรมทดสอบ 2 ชุด ในชุดแรกคือโปรแกรมส่วนการทำงานของซีพียูควบคุมหรือที่เรียกทั่วไปว่า โปรแกรมมอนิเตอร์ ชุดที่สองเป็นโปรแกรมส่วนแสดงผลบนจอภาพที่อยู่บนเครื่องคอมพิวเตอร์ เมื่อรันโปรแกรมทั้งสองชุดแล้ว บนจอคอมพิวเตอร์จะแสดงผลดังรูปที่ 4.1 เพื่อแสดงผลว่าโปรแกรมที่ต้องการถูกโหลดลงบนหน่วยความจำของอีมูเลเตอร์แล้ว จะสามารถตรวจสอบผลได้ด้วยการ ใช้คำสั่ง upload

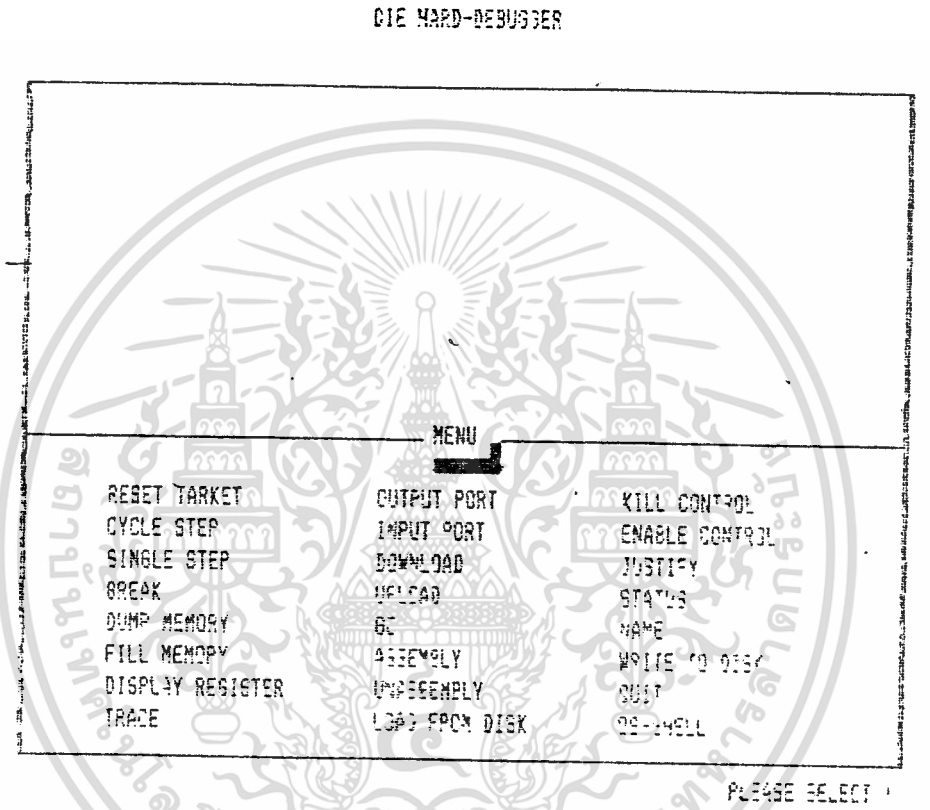
-D TEST.HEX

(U)UPLOAD

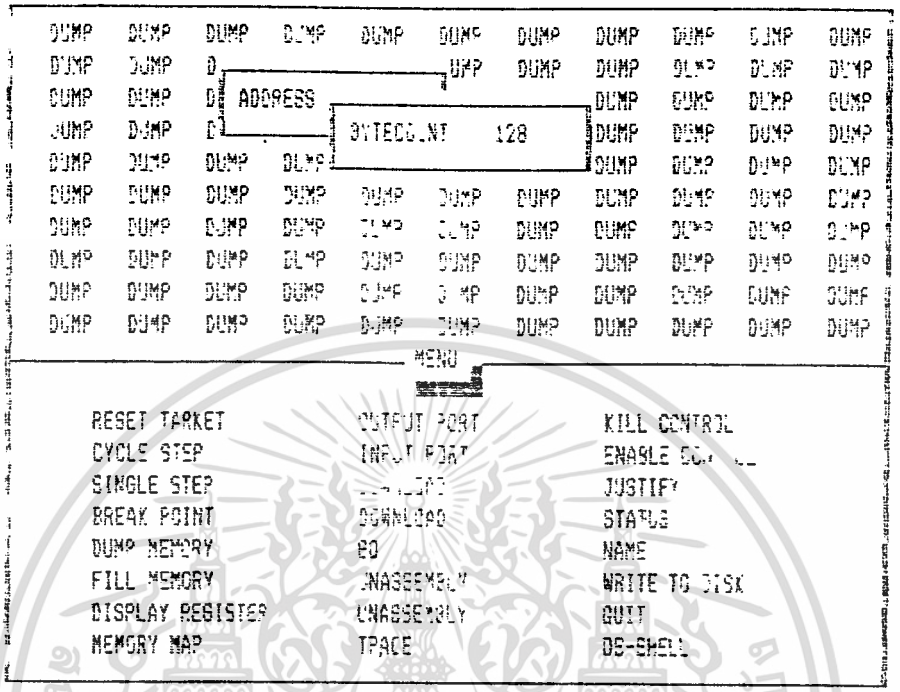
การเก็บค่าข้อมูลจากหน่วยความจำไปเก็บไว้ในไฟล์ข้อมูล, ก่อนการทดลองทำการอัพโหลดนั้นจะต้องทำการดาวน์โหลดค่าที่แน่นอนลงบนหน่วยความจำก่อน จึงจะสามารถรู้ได้ว่าค่าที่อัพโหลดที่ได้มานั้นถูกต้องตามค่าที่ได้ดาวน์โหลดลงไป แสดงผลการทดลองได้ดังรูปที่ 4.2

-U 0000 128

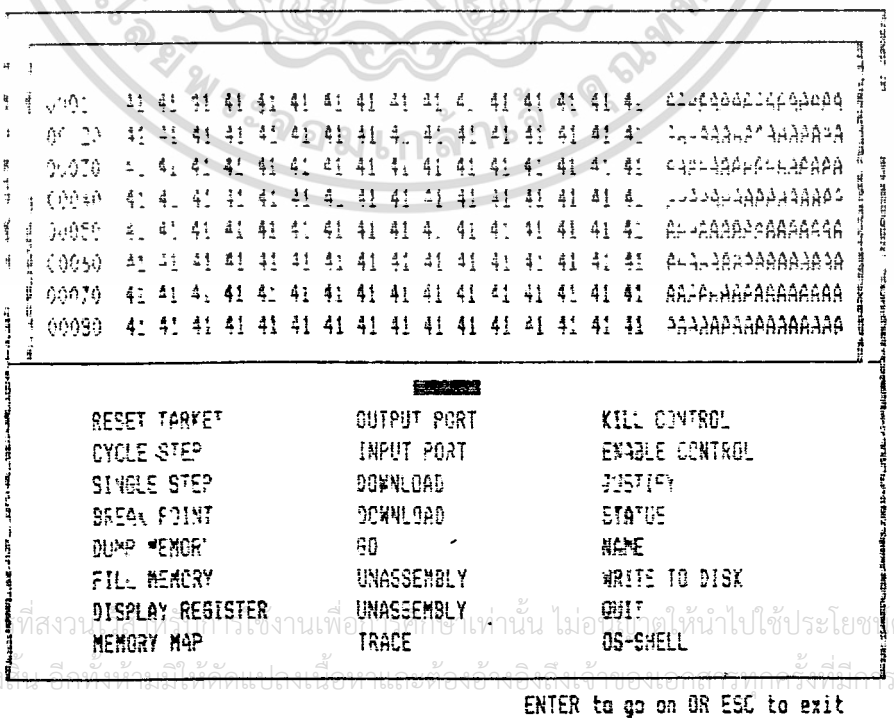
ในช่วงแรกของการทดสอบ มีปัญหาเกี่ยวกับการรับส่งข้อมูลได้ไม่ถูกต้อง เนื่องจากความเร็วและลำดับของการรับส่งข้อมูล หลังจากที่แก้ปัญหาดังกล่าว ผลของการรับส่งข้อมูลจึงถูกต้องดัง




รูปที่ 4.1 ผลจากการทำคำสั่ง Download




รูปที่ 4.2 ผลจากการทำคำสั่ง Upload



ปรากฏบนจอภาพ

(G)RUN

การสั่งให้ไมโครโปรเซสเซอร์ Z80 ทำงานตามตำแหน่งแอดเดรสที่ต้องการ จากการทดลองสั่งให้ ไมโครโปรเซสเซอร์ Z80 ทำงานในช่วงกลางของโปรแกรม

-G 001B

-

(S)SINGLE STEP

การสั่งให้ไมโครโปรเซสเซอร์ ทำงานทีละคำสั่ง โดยแสดงผลค่ารีจิสเตอร์ ในขณะนั้นด้วย

-S 0012

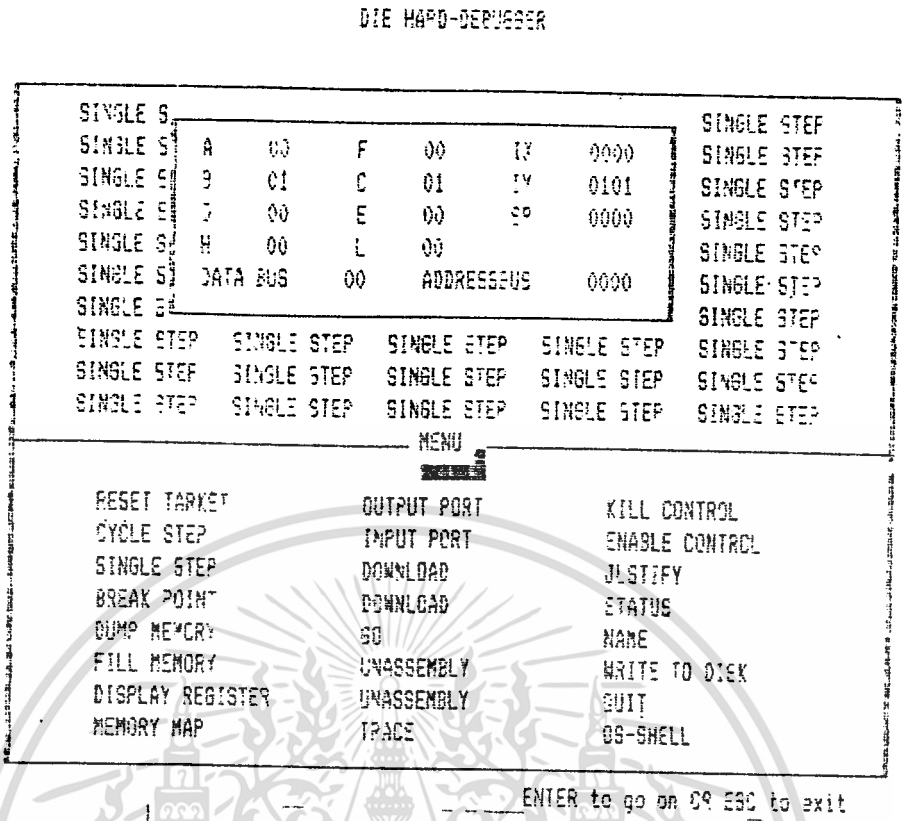
-

(C)CYCLE STEP

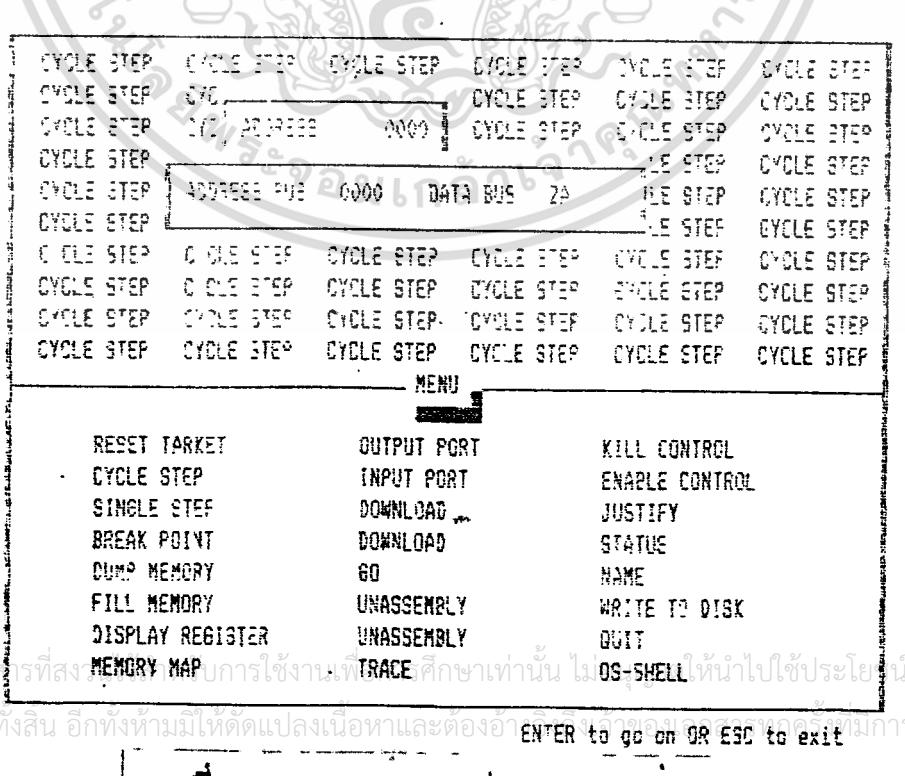
การสั่งให้ไมโครโปรเซสเซอร์ ทำงานทีละรอบแมชชีนไซเคิล

-C

IP DATA BUS




รูปที่ 4.3 ผลจากการทำคำสั่ง Single Step




รูปที่ 4.4 ผลจากการทำคำสั่ง Cycle Step

# บทที่ 5
# สรุปและวิจารณ์

ในบทนี้ จะเป็นการสรุปขั้นตอนการทำปริญญานิพนธ์ที่ได้ทำมา โดยจะกล่าวถึงจุดประสงค์ในการทำการวิจัย ปัญหาที่เกิดขึ้นและการแก้ไข ผลที่ได้รับจากการทำวิจัย รวมทั้งแนวทางในการที่จะพัฒนาประสิทธิภาพของอิมูเลเตอร์ แซด-80 ต่อไป

จุดประสงค์ของการทำ อิมูเลเตอร์นี้คือ การทำอุปกรณ์ที่สามารถช่วยในการพัฒนาระบบที่ทำงานโดยอาศัยไมโครโปรเซสเซอร์ แซด-80 ซึ่งช่วยในการพัฒนา แก้ไขโปรแกรม ซึ่งส่วนโครงงานตัวแบบนี้สามารถที่จะทำงานต่างๆ และแสดงผลให้กับผู้ใช้โดยผ่านทางจอภาพของเครื่องคอมพิวเตอร์ส่วนบุคคล

ความสามารถของอิมูเลเตอร์ชุดนี้ที่สำคัญคือ ความสามารถในการส่งผ่านโปรแกรมจากแฟ้มข้อมูลบนแผ่นดิสเก็ต (diskette) ของเครื่องคอมพิวเตอร์ไปยังไมโครโปรเซสเซอร์ แซด-80, สามารถที่จะแสดงค่าของรีจิสเตอร์ได้, สามารถเติมค่าต่างๆในหน่วยความจำตำแหน่งต่างๆได้, สามารถทำงานแบบทีละคำสั่งได้, สามารถทำงานทีละหนึ่งแมชชีนไซเคิลได้ ฯลฯ

ในการติดต่อระหว่างเครื่องอิมูเลเตอร์กับเครื่องคอมพิวเตอร์นั้นจะต้องติดต่อผ่านทางไมโครคอนโทรลเลอร์หมายเลข 8031 อีกที ปัญหาที่เกิดขึ้นส่วนใหญ่ในการพัฒนาชุดอิมูเลเตอร์ชุดนี้ก็คือ ปัญหาในการติดต่อสื่อสารกันระหว่างอิมูเลเตอร์และไมโครคอนโทรลเลอร์ 8031 และส่วนปัญหาที่เกิดจากการติดต่อกันระหว่างไมโครคอนโทรลเลอร์ 8031 กับเครื่องคอมพิวเตอร์ ในส่วนปัญหาในการติดต่อระหว่างอิมูเลเตอร์กับไมโครคอนโทรลเลอร์ก็คือ ปัญหาเรื่องความพร้อมกันของเวลาของสัญญาณของระบบทั้งสอง ซึ่งปัญหานี้เป็นปัญหาที่แก้ไขได้โดยการออกแบบแต่เริ่มต้น ในส่วนของการติดต่อระหว่างไมโครคอนโทรลเลอร์กับคอมพิวเตอร์นั้นปัญหาที่เกิดขึ้นก็คือความต่างกันของระบบการรับส่งข้อมูลระหว่างเครื่องคอมพิวเตอร์กับไมโครคอนโทรลเลอร์ ซึ่งปัญหานี้สามารถแก้ไขได้โดยการเขียนซอฟท์แวร์ให้มีความยืดหยุ่นในการรับส่งข้อมูลในแบบต่างๆ เพื่อรองรับการรับส่งสัญญาณจากทั้งสองระบบ ปัญหาที่สำคัญอีกอย่างก็คือปัญหาเรื่องสัญญาณรบกวนในสายต่างๆที่มีความยาว เช่นสายไฟเลี้ยงระบบ สายส่งสัญญาณ ซึ่งปัญหาจะออกมาในลักษณะการผิดเพี้ยนของสัญญาณที่ระบบ วิธีการแก้ปัญหาในกรณีแรกก็คือการลดความยาวของสายจ่ายไฟเลี้ยง และการใช้ตัวเก็บประจุต่อคร่อมที่ขั้วของสายจ่ายไฟเลี้ยงวงจรโดยให้ตัวเก็บประจุอยู่ใกล้กับระบบมากที่สุด

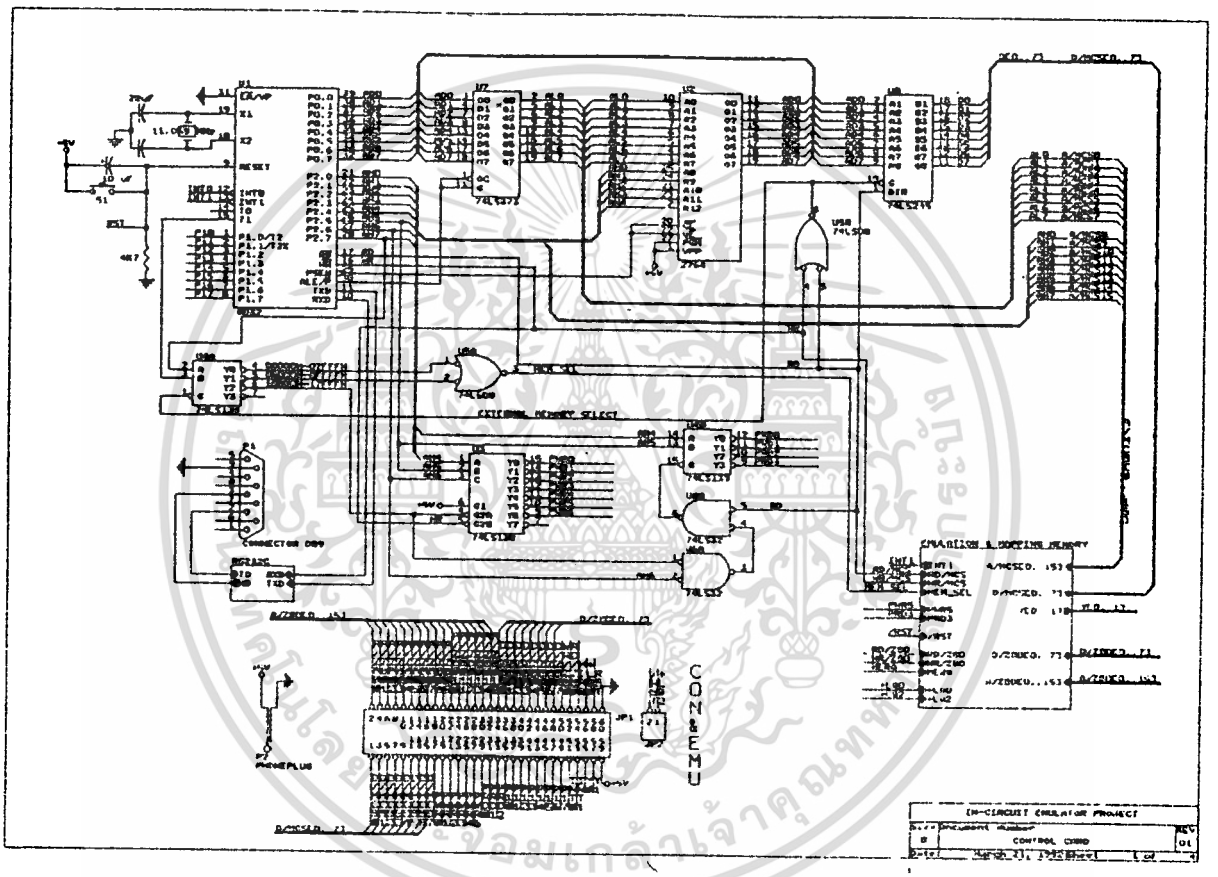
ส่วนในกรณีที่สองจะทำให้สัญญาณที่ส่งออกไปนั้นจะมีการเพี้ยนที่ปลายทาง วิธีการแก้ปัญหานี้ทำได้โดยการลดความยาวของสายส่ง และ การเขียนซอฟท์แวร์ที่มีการตรวจสอบผลรวมของข้อมูล เมื่อพบว่าการส่งสัญญาณเกิดความผิดพลาดให้ทำการส่งสัญญาณใหม่

แนวทางในการพัฒนาต่อ จากโครงงานในชุดนี้ เพื่อให้มีความสะดวกในการทำงานมากขึ้น ซอฟท์แวร์ในส่วนที่ทำการติดต่อกับผู้ใช้ควรจะมีลักษณะที่มีอรรถประโยชน์ครบทุกอย่างในการใช้งานในการพัฒนาระบบไมโครโปรเซสเซอร์ เช่นมีอิดิเตอร์ที่มีประสิทธิภาพสูงขึ้นสามารถเรียกใช้งานได้อย่างรวดเร็ว มีคอมไพเลอร์ในตัว มีดีบักเกอร์ที่สามารถติดต่อส่งผ่านข้อมูลกับอิดิเตอร์ และคอมไพเลอร์ได้อย่างสะดวกโดยถือว่า ทั้งคอมไพเลอร์ อิดิเตอร์ และ ดีบักเกอร์นี้เป็นส่วนเดียวกัน อย่างเช่นสามารถทำการเติมค่าต่างลงในหน่วนความจำได้ ซึ่งการเติมค่านี้ก็เป็นการเติมค่าลงในอิดิเตอร์ไปในตัวด้วย ฯลฯ

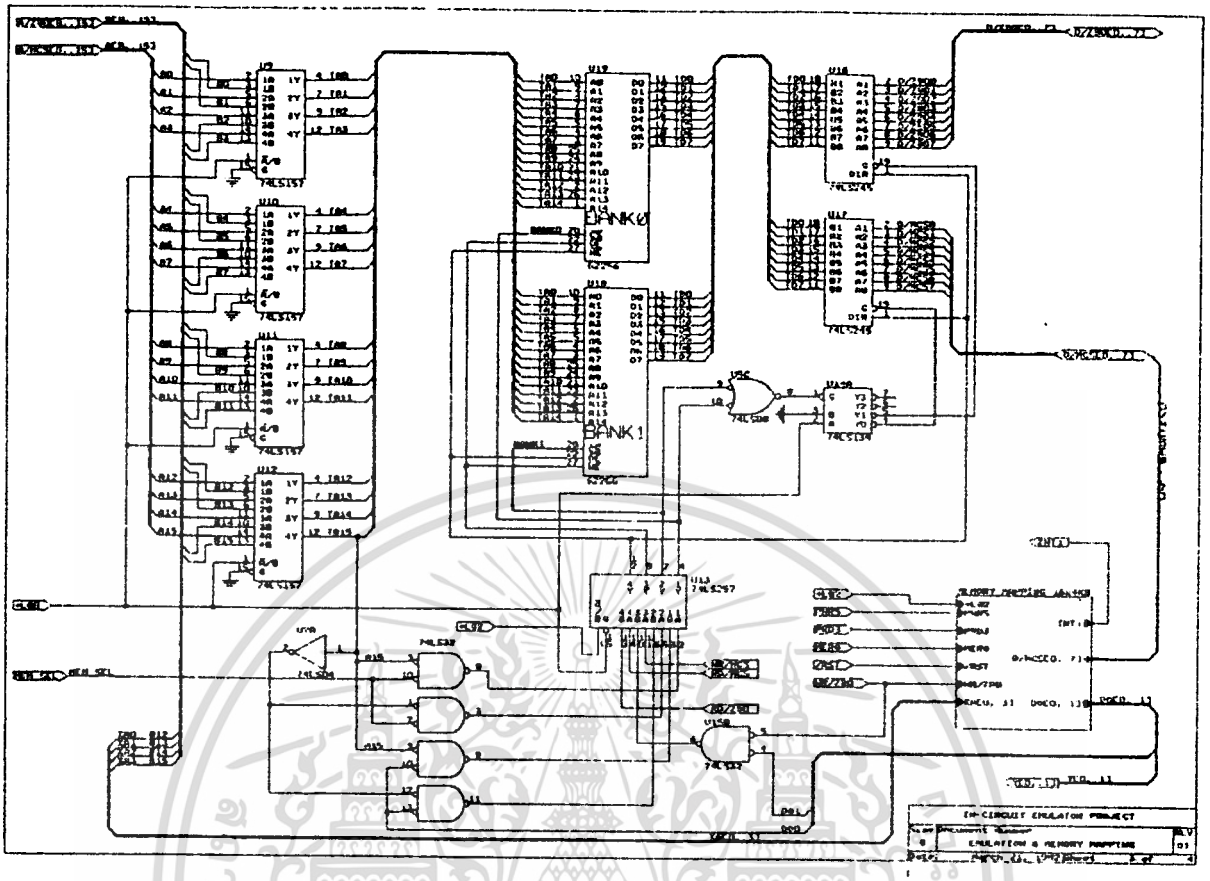
ผลจากการทำวิจัยในครั้งนี้ทำให้เข้าใจการทำงานของไมโครโปรเซสเซอร์ แซด-80, ไมโครคอนโทรลเลอร์ เอ็มซีเอส-51 และการทำงานของพอร์ทอนุกรมบนเครื่องคอมพิวเตอร์มากยิ่งขึ้น ซึ่งความเข้าใจนี้สามารถที่จะนำไปประยุกต์ใช้งานในระบบอื่นๆได้อย่างมีประสิทธิภาพ

# ภาคผนวก ก

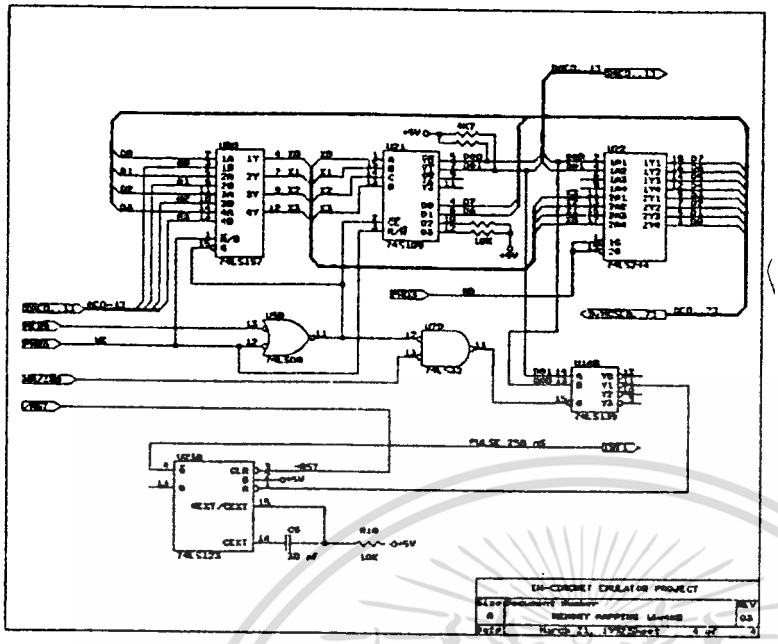
## วงจรต่างๆ



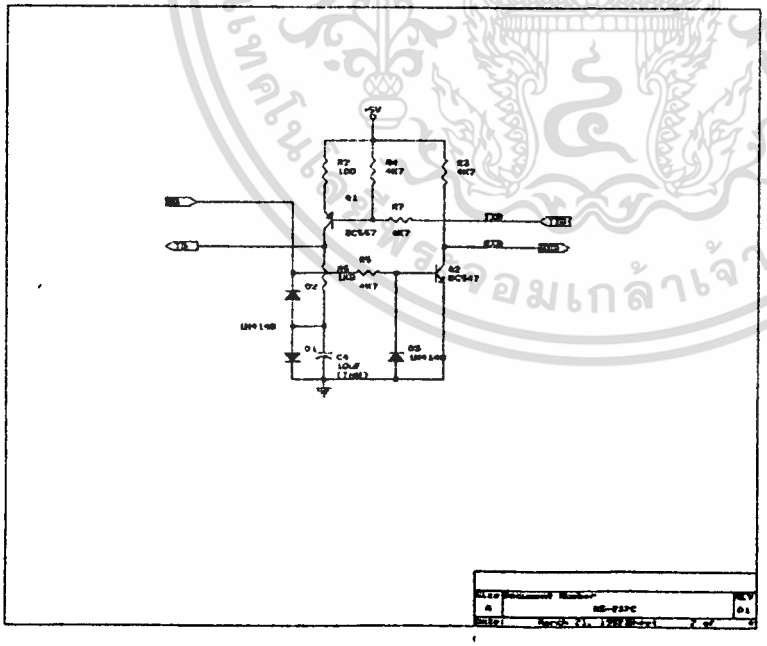

วงจรส่วนบอร์ดควบคุม



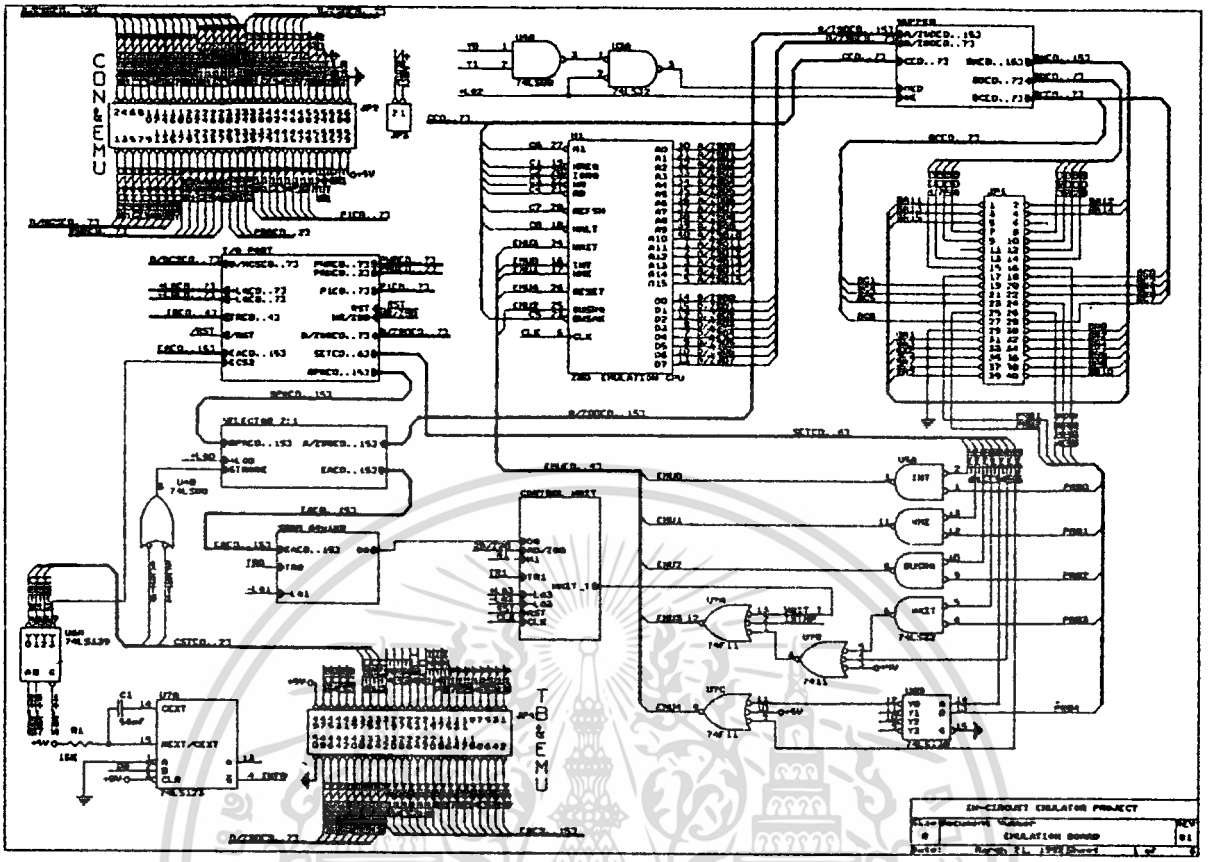

วงจรย่อยส่วนหน่วยความจำอีมูเลชั่น



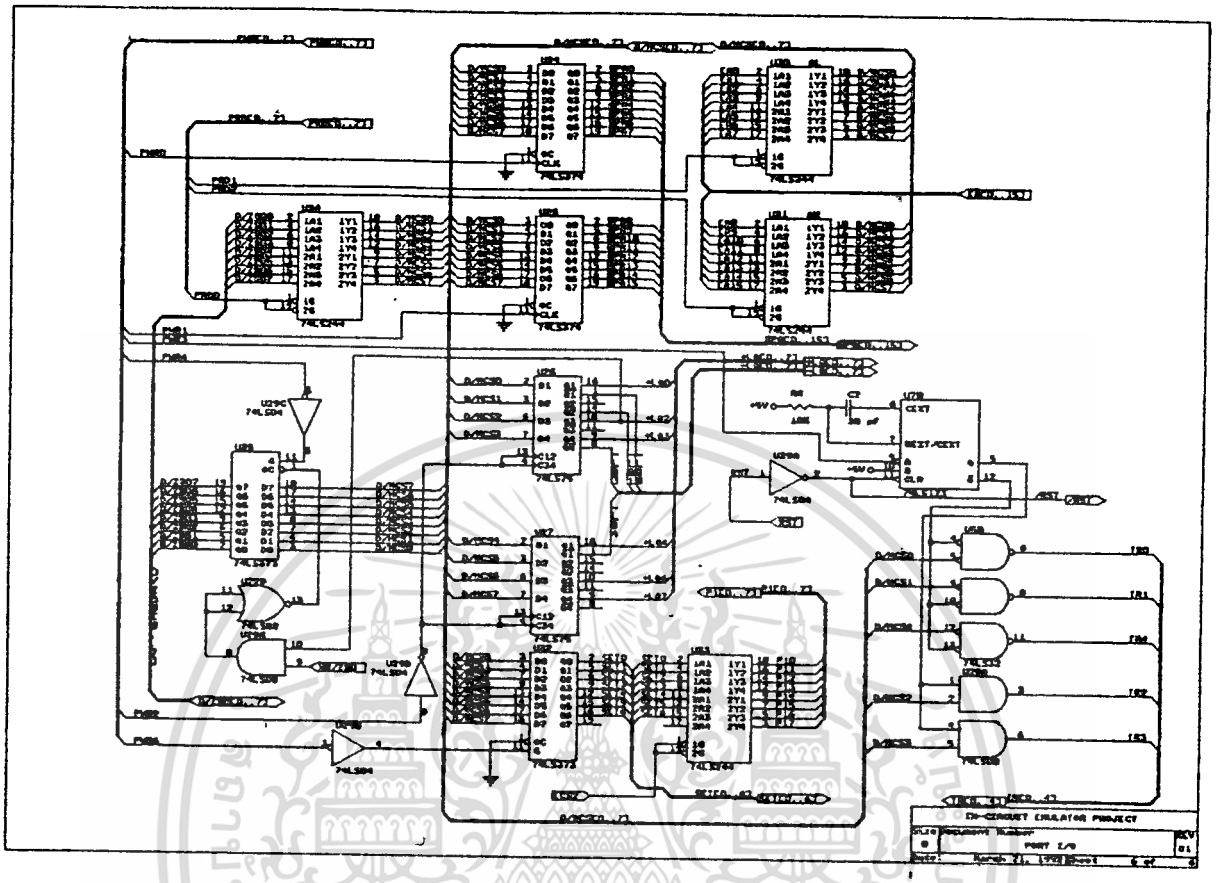
วงจรย่อยส่วนเมมโมรีแมพ



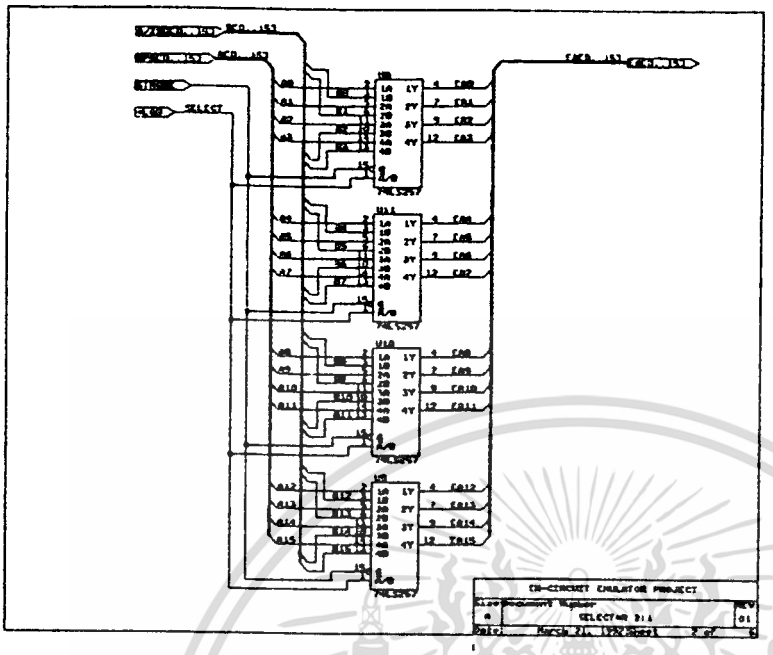
วงจรย่อยส่วนการสื่อสารแบบ RS232C



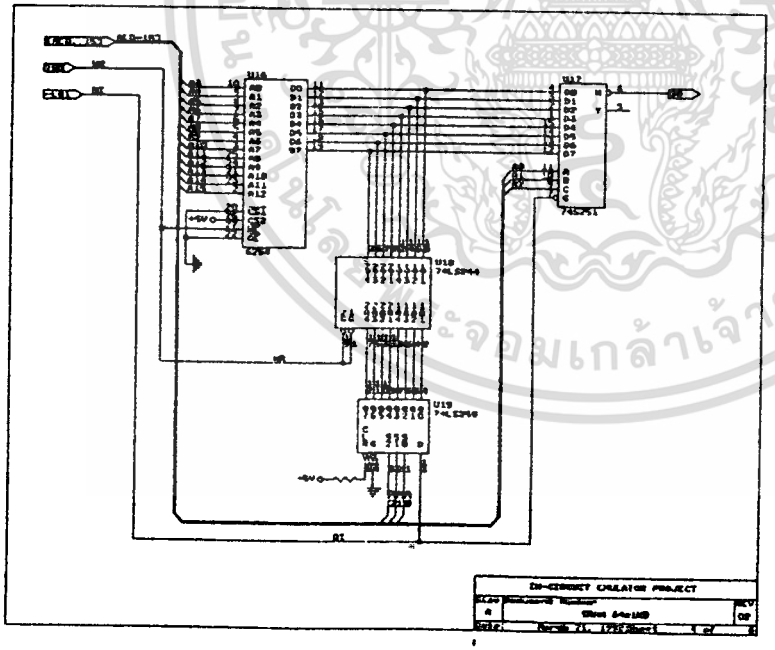

วงจรสวนบอร์ดอีมูเลชั่น



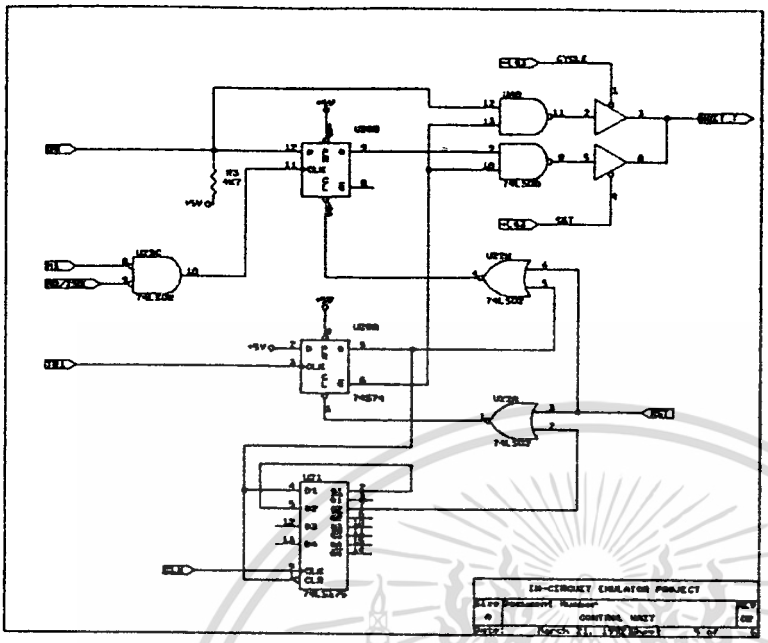
วงจรย่อยส่วนพอร์ทอินพุท/เอาท์พุท



วงจรย่อยส่วนเซเล็กเตอร์ 2:1



วงจรย่อยส่วนแอดเดรสแมพ



วงจรย่อยส่วนการเกิดสัญญาณ WAIT

# ภาคผนวก | ข

INTEL-HEX FILE FORMAT

รูปแบบของ INTEL HEX FILE ประกอบด้วยเรคคอร์ด 2 ชนิดคือ เรคคอร์ดข้อมูล (DATA RECORD) และเรคคอร์ดสิ้นสุดแฟ้มข้อมูล (end-of-file record) โดยรูปแบบของเรคคอร์ดจะเริ่มต้นด้วยรหัสนำ 9 ตัวอักษร ตามด้วยข้อมูล(ถ้ามี) และรหัสปิดท้าย 2 ตัวอักษร โดยมีรูปแบบดังนี้

:BCAAATTHH....HHCC

: - ตัวอักษรเริ่มต้น

BC - byte count จำนวนไบท์ของข้อมูลในเรคคอร์ด มีค่าเป็นเลขฐาน 16 (HEX)
(BC เป็น 00 ถ้าเป็นเรคคอร์ดสิ้นสุดแฟ้มข้อมูล)

AAAA - คือตำแหน่งของข้อมูลไบท์แรกในเรคคอร์ด

TT - แสดงชนิดของเรคคอร์ด
(TT เป็น 00 ถ้าเป็นเรคคอร์ดข้อมูล)
(TT เป็น 01 ถ้าเป็นเรคคอร์ดสิ้นสุดแฟ้ม)

HH - ข้อมูล 1 ไบท์

CC - ค่าของ check sum ซึ่งมีค่าเป็น 2's complement ของผลบวกของข้อมูลทุกไบท์ในเรคคอร์ด รวมทั้งผลบวกของจำนวนไบท์ข้อมูล (BC) ,AAAA,TT

# ภาคผนวก ค

## ความรู้ทั่วไปเกี่ยวกับ MCS-51

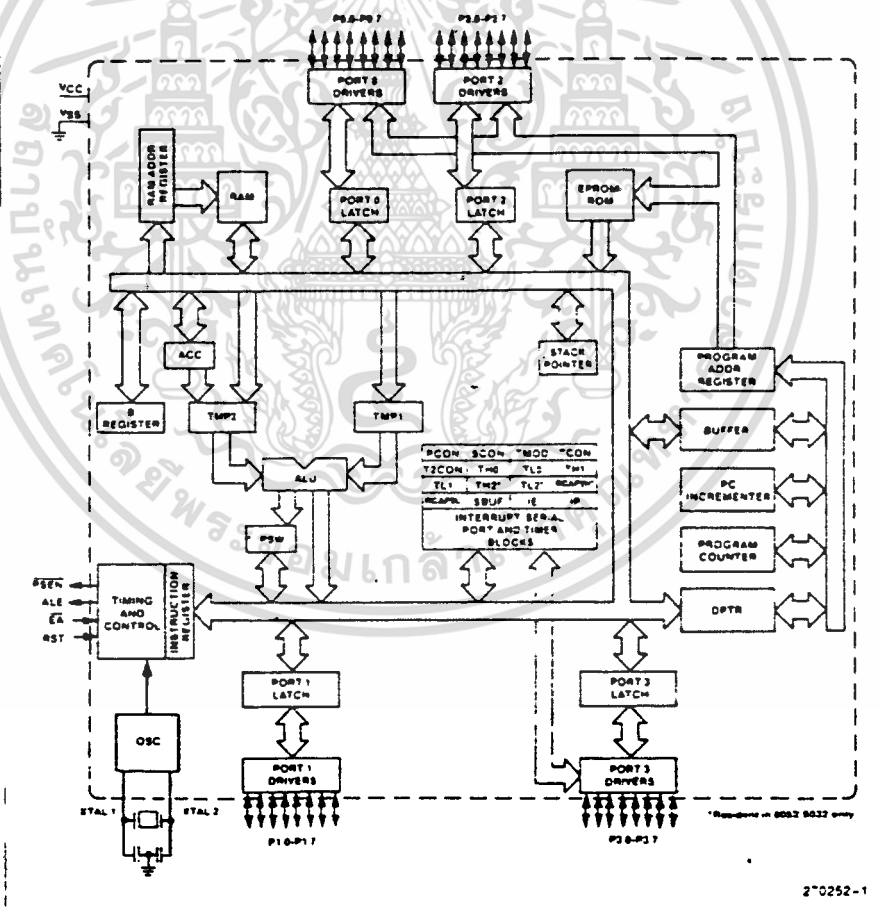
ในการสร้างไมโครคอนโทรลเลอร์นั้น ส่วนสำคัญที่สุดก็คือ ตัวชิพไมโครโปรเซสเซอร์ ซึ่งชิพในแต่ละตระกูลนั้น ก็มีข้อแตกต่างกันอยู่หลายอย่างตามสายตระกูลนั้น ๆ สำหรับชิพในตระกูล MCS-51 นั้นมีอยู่ด้วยกันหลายเบอร์ เช่น 8051,8751,8031 ฯลฯ ลักษณะโดยทั่วไปของชิพตระกูลนี้ พอจะสรุปได้ดังนี้คือ

- ซีพียูมีขนาด 8 บิท
- มีวงจรออสซิเลเตอร์ (oscillator) และวงจรนาฬิกาบนชิพ
- ชุด รีจิสเตอร์ใช้งาน (bank register) มีอยู่ 4 ชุด
- มีตัวจับเวลา/ตัวนับ (timer/counter) ขนาด 16 บิท 2 ชุด
- มีพอร์ตอินพุท/เอาท์พุท แบบขนาน 2 ทิศทาง จำนวน 4 พอร์ต พอร์ตละ 8 บิท รวมทั้งหมด 32 เส้น แต่สำหรับเบอร์ 8031 จะเหลือเพียง 16 เส้น อีก 16 เส้นที่เหลือใช้ในการเข้าถึงทางแอดเดรสและข้อมูล
- พอร์ตแบบอนุกรมสามารถที่จะโปรแกรมการรับส่งแบบ Full Duplex ที่ความเร็วสูง
- หนึ่งวัฏจักรคำสั่งจะใช้เวลา 1 ไมโครวินาที ด้วยการใช้คริสตัล 12 MHz
- อ้างแอดเดรสข้อมูลภายนอกได้ 64 kBytes
- อ้างแอดเดรสโปรแกรมภายนอกได้ 64 kBytes
- สามารถกำหนดเข้าถึงที่อยู่ข้อมูลขนาดไบต์หรือบิตได้โดยตรง (Direct Bit & Byte Addressing)
- มีซอฟต์แวร์แฟลกสำหรับผู้ใช้ที่จะกำหนดเองได้ถึง 128 ตำแหน่งบิท
- โครงสร้างอินเตอร์รัพท์ได้ 5 แหล่งและ 6 แหล่ง สำหรับ 8032/8052 พร้อมด้วยการจัด priority ได้ 2 ระดับ
- ตัวโปรเซสเซอร์สามารถใช้งานแบบบูลีน (boolean) ได้
- มีคำสั่งคูณและหารทางฮาร์ดแวร์ ทำได้ภายใน 4 ไมโครวินาที
- ตัวเลขทางคณิตศาสตร์ใช้ได้ทั้งแบบไบนารีและเดซิมัล

- การใช้พื้นที่สแตกค์ (stack) สำหรับโปรแกรมย่อยต่าง ๆ ทำได้กว้างขึ้น

## สถาปัตยกรรมของ MCS-51

ในรูปที่ 1 เป็นรูปสถาปัตยกรรมภายในของ MCS-51 ซึ่งประกอบไปด้วย central processing unit, program และ data memory, input/output ports, registers สำหรับ mode status และ data ซึ่งแต่ละส่วนนี้เชื่อมโยงถึงกันโดยผ่าน data bus ขนาด 8 บิท และติดต่อกับภายนอกโดยผ่าน I/O port ซึ่งแต่ละส่วนสามารถสรุปย่อได้ดังนี้




รูปที่ 1

**Central Processing unit**

CPU อาจกล่าวได้ว่าเป็น "สมอง" ของไมโครคอมพิวเตอร์ (microcomputer) เป็นตัวประมวลผลคำสั่งที่ป้อนเข้ามา ซึ่งในซีพียูนี้ก็จะประกอบไปด้วย Arithmetic/Logic Unit (ALU), รีจีสเตอร์(Register) A, B,PSW,SP, Program Counter และ Data Pointer ขนาด 16 บิต ในแต่ละส่วนทำหน้าที่ดังนี้

***Arithmetic Logic Unit***

ALU เป็นตัวทำหน้าที่จัดการคำสั่งต่าง ๆ ที่เกี่ยวกับตัวเลข เช่นการบวก, การลบ, การคูณ, การหาร และการคำนวณทางตรรก (Logical operation) เช่น AND, OR, Exclusive-ORและรวมทั้งโรเทท์(rotate),เครียร์(clear),คอมพลีเมนท์(completement) ALU ยังทำหน้าที่เป็นทางผ่าน และเป็นรีจีสเตอร์ชั่วคราว สำหรับการส่งผ่านข้อมูลภายในระบบด้วย ALU สามารถที่จะตัดสินใจในการกระโดดไปทำคำสั่งของโปรแกรมในส่วนอื่น ๆ ตามเงื่อนไขที่ตั้งขึ้น

ข้อเด่นของALU ตระกูล MCS-51 คือ มันสามารถจัดการข้อมูลทั้งขนาด 1 บิทและ 8 บิทได้ ข้อมูลขนาด 1 บิท สามารถถูกเซ็ท, เคลียร์ หรือคอมพลีเมนท์ (complement) , เคลื่อนย้าย, ตรวจสอบ และใช้ในการประมวลผลทางตรรกได้ ซึ่งในการจัดข้อมูลขนาด 1 บิทนี้ ทำได้ลำบาก สำหรับไมโครโปรเซสเซอร์ทั่ว ๆ ไป งานในลักษณะนี้จึงได้ชื่ออีกอย่างหนึ่งว่า ตัวประมวลผลบูลีน (Boolean Processor)

***Accumulator และ PSW***

Accumulator (A) เป็นรีจิสเตอร์ขนาด 8 บิท ทำหน้าที่เป็นที่สำหรับ source operand และที่รับผลลัพธ์จากการประมวลผลทั้งทางคณิตศาสตร์และทางตรรก บางคำสั่งก็ใช้ทางานโดยตรงกับเอกคิวมูเลเตอร์ (Accumulator) เลยเช่น การโรเทท์, การจัดการด้านพาริตี้ (parity) ซึ่งคำสั่งบางคำสั่งก็มีผลในการเปลี่ยนค่าสถานะแฟลกที่อยู่ในรีจีสเตอร์ Program Status Word (PSW) ซึ่งแต่ละแฟลกแสดงได้ดังรูปที่ 2 สำหรับ Carry flag นั้น ยังอาจใช้เป็น Boolean Accumulator สำหรับคำสั่งในการจัดการข้อมูลขนาด 1 บิทด้วย

*CPU Register ตัวอื่น ๆ*

รีจีสเตอร์ B ใช้ในการทำงานของคำสั่งคูณและหาร รีจีสเตอร์นี้ใช้คู่กันแอกคูมูเลเตอร์ โดยเป็น operand ตัวที่สอง

Stack Pointer (SP) ชิพในตระกูล MCS-51 นี้จะใช้เนื้อที่ของแรมภายใน ในการสร้างสแตกค์ ซึ่งสแตกค์นี้สามารถสร้างได้ถึง 128 Bytes แต่ในการสร้างงานจริงจะต้องใช้สแตกค์ในการใช้เก็บค่าตัวแปรต่าง ๆ ด้วย ดังนั้นค่านี้ก็จะลดหลั่นลงไปด้วย สำหรับค่า defaults ของ SP เมื่อทำการรีเซ็ท คือ 07H

(MSB) (LSB)

| CY | AC | F0 | RS1 | RS0 | OV | - | P |
|---|---|---|---|---|---|---|---|

| สัญลักษณ์ | ตำแหน่ง | ข้อกำหนดการทำงาน |
|---|---|---|
| CY | PSW7 | แฟลกตัวทด จะเซ็ท/เคลียร์ด้วยฮาร์ดแวร์หรือซอฟต์แวร์ ระหว่างผลลัพธ์หลังการใช้คำสั่งทางคณิตศาสตร์ หรือตรรกศาสตร์ที่เน่นอน |
| AC | PSW6 | แฟลกตัวทดของ Auxilliary จะเซ็ท/ เคลียร์ด้วยฮาร์ดแวร์ ระหว่างการบวกและลบ ที่แสดงผลจากการทดหรือยืมจากบิทที่ 3 ของ ACC |
| F0 | PSW5 | แฟลก 0 จะเซ็ท/เคลียร์ด้วยซอฟต์แวร์ที่ผู้ใช้กำหนดสถานะแฟลกนี้เอง |
| RS1 | PSW4 | รีจิสเตอร์ตัวควบคุมการเลือกแบงค์ ด้วยค่า RS1 และ RS0 |
| RS0 | PSW3 | จะเซ็ทหรือเคลียร์ด้วยซอฟต์แวร์ เพื่อเลือกกลุ่มรีจีสเตอร์ทำงานในแต่ละแบงค์ โดยปรับค่าใน RS1 และ RS0 ให้อีนาเบิ้ลคุมลักษณะการเลือกแบงค์ต่อไปนี้ |

| RS1 | RS0 | เลือกแบงค์ | ค่าแอดเดรส |
|---|---|---|---|
| 0 | 0 | แบงค์ 0 | 00H - 07H |
| 0 | 1 | แบงค์ 1 | 08H - 0FH |
| 1 | 0 | แบงค์ 2 | 10H - 17H |
| 1 | 1 | แบงค์ 3 | 18H - 1FH |

| | | |
|---|---|---|
| OV | PSW2 | <u>แฟลก overflow</u> จะเซ็ทหรือเคลียร์ด้วยฮาร์แวร์ระหว่างการใช้คำสั่งที่แสดงผลถึงการเกิดลักษณะ overflow ทางคณิตศาสตร์ |
| - | PSW1 | <u>บิทสำรอง</u> จะไม่สามารถเซ็ท/เคลียร์ด้วยผู้ใช้ เพราะสำรองไว้สำหรับโรงงานผู้สร้าง |
| P | PSW0 | <u>แฟลกพาริตี้</u> จะเซ็ท/เคลียร์ด้วยฮาร์ดแวร์ในแต่ละวัฏจักรคำสั่ง แสดงถึงตัวเลขคำ "1"ในแต่ละบิทของแอกคิวมูเลเตอร์ เช่น "1"มี 6 ตัว จะเป็นพาริตี้คู่ P บิทจะเท่ากับ 0 |

<u>ข้อมูลชุดที่ 1</u>

Data pointer (DPTR) เป็นรีจีสเตอร์ขนาด 16 บิท โดยแบ่งเป็นรีจีสเตอร์ 8 บิท 2ตัวคือ ไบต์สูง (DPH) และไบต์ต่ำ (DPL) ทำหน้าที่อ้างตำแหน่งของข้อมูลใน external data memory และยังใช้เป็นฐานในคำสั่งการกระโดดแบบ indirect ด้วย

**Memory Space**

ในส่วนหน่วยความจำของซีพียูตระกูล MCS-51 นั้น แยกออกเป็น 2 ชนิด คือหน่วยความจำโปรแกรม (Program Memory) และหน่วยความจำข้อมูล (data memory) แต่ละชนิดสามารถอ้างได้สูงสุดชนิดละ 64 Kbytes



หน่วยความจำโปรแกรมนั้นจะเก็บในรูปของรอม หรือ อีพรอม (EPROM) ซึ่งในส่วนนี้จะทำ

งานทันทีเมื่อมีการเปิดเครื่อง ในการอ้างแอดเดรสนั้นสามารถทำได้โดย โปรแกรมเค้าท์เตอร์ Program Counter (PC) ซึ่งเป็นรีจีสเตอร์ขนาด 16 บิท ผ่านทาง bus 16 bit ส่วนหน่วยความจำข้อมูลนั้นจะเป็นแรม ซึ่งยังแบ่งแรมออกเป็น 2 ชนิด คือ แรมภายใน กับ แรมภายนอกซึ่งในการอ้างแอดเดรสของแรมภายในนั้นจะอ้างแบบโดยตรง โดยใช้ ALU เป็นตัวชี้ตำแหน่ง ส่วนแรมภายนอกนั้นเป็นแบบ indirect

พอร์ต 0 ถึง 3 (P0, P1, P2, P3)

รีจีสเตอร์ P0, P1, P2, P3 ของกลุ่มรีจีสเตอร์ฟังก์ชันกลุ่มพิเศษ (Special Function Register : SFR) จะเป็นตัวรีจีสเตอร์ที่เลทช์ค่าของพอร์ท 0,1,2,3 ตามลำดับ ในขณะใช้งาน

**บัฟเฟอร์ข้อมูลอนุกรม (Serial Data Buffer : SBUF)**

บัฟเฟอร์ข้อมูลอนุกรม แบ่งออกเป็นรีจีสเตอร์สองตัว ตัวหนึ่งเป็นบัฟเฟอร์การส่ง และอีกตัวเป็นบัฟเฟอร์การรับ เมื่อข้อมูลถ่ายเทเข้า SBUF มันจะถ่ายเข้าบัฟเฟอร์ส่งซึ่งเป็นตัวจัดการส่งข้อมูลอนุกรม วิธีการเคลื่อนย้ายเข้า SBUF ขึ้นอยู่กับการเริ่มแรก (initial) การส่ง เมื่อข้อมูลย้ายออกจาก SBUF จะเป็นการรับข้อมูลบัฟเฟอร์ตัวรับ

**รีจีสเตอร์ควบคุม (Control Register)**

เป็นกลุ่ม SFR ที่เป็น IP, IE, TMOD, TCON, T2CON, SCON, PCON จะประกอบด้วย บิทที่ใช้ในการควบคุมและแสดงสถานะของการทำงานในระบบอินเตอร์รัพท์ ตัวตั้งเวลา/ตัวนับ และพอร์ตอนุกรม ซึ่งตัว MCS-51 จะมีเนื้อที่ในหน่วยความจำบนชิพจำนวน 128 ไบท์ สำหรับรีจีสเตอร์ฟังก์ชันพิเศษนี้ จะสามารถจัดแบ่งตำแหน่งสำหรับ SFR ให้ทำงานเป็นรีจีสเตอร์ต่าง ๆ ได้ดังข้อมูลชุดที่ 3

| | | ตำแหน่ง |
|---|---|---|
| * ACC | Accumulator | 0E0H |
| * B | B รีจีสเตอร์ | 0F0H |
| * PSW | Program Status Word | 0D0H |
| SP | Stock Pointer | 081H |
| DPTR | ตัวชี้ข้อมูล ประกอบด้วย DPH และ | 083H |
| | DPL | 082H |
| * P0 | พอร์ต 0 | 080H |
| * P1 | พอร์ต 1 | 090H |
| * P2 | พอร์ต 2 | 0A0H |
| * P3 | พอร์ต 3 | 0B0H |
| * IP | ตัวควบคุมการอินเตอร์รัพต์ตามลำดับ | 0B8H |
| * IE | ตัวควบคุมการอินเตอร์รัพต์อีนาเบิ้ล | 0A8H |
| TMOD | ตัวควบคุมการเลือกโหมดการตั้งเวลา/ตัวนับ | 089H |
| * T2CON | ตัวควบคุมการตั้งเวลา/ตัวนับ 2 | 088H |
| TCON | ตัวควบคุมการตั้งเวลา/ตัวนับ | 0C8H |
| TH0 | รีจีสเตอร์ตัวตั้งเวลา/ตัวนับ 0 (ไบต์สูง) | 08CH |
| TL0 | รีจีสเตอร์ตัวตั้งเวลา/ตัวนับ 0 (ไบต์ต่ำ) | 08AH |
| TH1 | รีจีสเตอร์ตัวตั้งเวลา/ตัวนับ 1 (ไบต์สูง) | 08DH |
| TL1 | รีจีสเตอร์ตัวตั้งเวลา/ตัวนับ 1 (ไบต์ต่ำ) | 08BH |
| + TH2 | รีจีสเตอร์ตัวตั้งเวลา/ตัวนับ 2 (ไบต์สูง) | 0CDH |
| + TL2 | รีจีสเตอร์ตัวตั้งเวลา/ตัวนับ 2 (ไบต์ต่ำ) | 0CCH |
| + RLDH | รีจีสเตอร์ตัวตั้งเวลา/ตัวนับ 2 ประจุใหม่อัตโนมัติ(ไบต์สูง) | 0CBH |
| + RLDL | รีจีสเตอร์ตัวตั้งเวลา/ตัวนับ 2 ประจุใหม่อัตโนมัติ(ไบต์ต่ำ) | 0CAH |
| * SCON | ควบคุมการส่งข้อมูลอนุกรม | 098H |
| SBUF | บัฟเฟอร์ข้อมูลการส่งอนุกรม | 099H |
| PCON | ควบคุมการใช้พลังงาน (Power) | 097H |

เครื่องหมาย * หน้าตัวรีจีสเตอร์แสดงว่า รีจีสเตอร์นั้นสามารถที่จะแอดเดรสข้อมูลได้ทั้งข้อมูลขนาดไบต์และบิท และเครื่องหมาย + นั้นจะมีเฉพาะในเบอร์ 8032/8052 เท่านั้น

**การจัดขาลักษณะภายนอกของ MCS-51**

รูปที่ 2 แสดงการจัดขาตามลักษณะภายนอกของชิพ MCS-51 ซึ่งมีรายละเอียดดังนี้


รูปที่ 2 Pinout Diagram

- ขา VSS (ขา 20) เป็นขาสำหรับต่อลงดิน

- ขา VCC (ขา 40) เป็นขาที่ต่อแรงดันไฟ DC ขนาด 5 V และใช้สำหรับการโปรแกรม

- ขา Port0 (P0.0-P0.7/ AD0-AD7)(ขา 32-39)เป็นพอร์ต I/O 8 บิทแบบสองทิศทาง (bidirectional) สามารถที่จะรับโหลดทีทีแอลได้ 8 ตัว สำหรับเบอร์ 8031 จะใช้พอร์ต 0 เป็น I/O โดยจะทำงานเป็นมัลติเพลสเซอร์ด้วยสัญญาณแอดเดรสบิทต่ำกับบัสข้อมูลในการติดต่อกับหน่วยความจำภายนอก

- ขา Port1 (P1.0-P1.7) (ขา 1-8) เป็นพอร์ต I/O 8 บิท แบบสองทิศทาง ในเบอร์ 8052 ขา P1.0 และ P1.7 จะใช้งานเป็น T2 และ T2EX โดยขา T2 จะทำหน้าที่รับสัญญาณจากภายนอกให้ตัวตั้งเวลา 2 ทำงาน และขา T2EX จะเป็นตัวอินพุทผ่าน

เข้าตัวตั้งเวลา 2 ถูกกระตุ้นให้ทำงานแบบปกติตามโปรแกรมที่ตั้งไว้

-ขาPort2 (P2.0-P2.7) (ขา 21-28) เป็น พอร์ต I/O 8 บิท แบบสองทิศทาง นอกจากจะเป็นพอร์ต I/O แล้ว ยังใช้ทำงานพิเศษได้ดังตารางนี้

| ขาพอร์ต | ขา | การทำงานตามฟังก์ชันพิเศษ |
|---|---|---|
| P3.0 | 10 | RxD พอร์ตอนุกรมอินเตอร์รัพท์ |
| P3.1 | 11 | TxD พอร์ตอนุกรมเอาท์พุท |
| P3.2 | 12 | INT0 อินเตอร์รัพท์ภายนอกตัวที่ 1 |
| P3.3 | 13 | INT1 อินเตอร์รัพท์ภายนอกตัวที่ 2 |
| P3.4 | 14 | T0 สัญญาณกระตุ้นเข้าที่ตัวตั้งเวลา และตัวนับ 0 |
| P3.5 | 15 | T1 สัญญาณกระตุ้นเข้าที่ตัวตั้งเวลา และตัวนับ 1 |
| P3.6 | 16 | WR สัญญาณควบคุมการเขียน |
| P3.7 | 17 | RD สัญญาณควบคุมการอ่าน |

การใช้งานตามฟังก์ชันนี้ จะต้องเริ่มต้นโปรแกรมด้วยการส่งค่า '1' ไปเลท์ชไว้ที่รีจีสเตอร์ P3 ก่อนที่จะให้ทำงานตามฟังก์ชันนั้น ๆ

- ขา RST (Reset) (ขา 9) ต้องคงสถานะค่าสูงไว้เป็นอย่างน้อย สองรอบ ระหว่างที่ออสซิเลเตอร์ทำงาน ขณะที่ต้องการรีเซ็ทระบบ

- ขา ALE/PROG (ขา 30) เป็นขา address latch enable ด้วยการส่งพัลซ์ออกไปใช้สำหรับเลท์ชค่าแอดเดรสไบต์ต่ำจากพอร์ต 0 ขานี้ยังใช้เป็นสัญญาณพัลซ์เข้าสำหรับการควบคุมการโปรแกรม EPROM ภายในชิพ

- ขา PSEN (ขา 29) Program storage enable เป็น strobe การอ่านข้อมูลจากหน่วยความจำภายนอก เมื่อชิพทำงานด้วยโปรแกรมภายนอก ขา PSEN จะสร้าง strobe ค่าสองครั้งภายในแต่ละแมชชีนไซเคิล และเมื่อทำงานในช่วงการอ่านหรือเขียนข้อมูลจากหน่วยความจำข้อมูลภายนอกสัญญาณจะมีสถานะสูง และจะไม่มีพัลซ์ส่งออกถ้าชิพทำงานด้วยโปรแกรมหน่วยความจำภายใน



- ขา EA/Vpp (ขา 31) ถ้ามีสถานะสูง ตัวซีพียูจะทำงานตามโปรแกรมที่อยู่ในหน่วย

ความจำภายใน การทำให้ EA มีสถานะต่ำจะควบคุมให้ซีพียูทำงานตามโปรแกรมหน่วยความจำภายนอก ฉะนั้นในชิพที่ไม่มีรอมภายใน ก็จะต้องต่อขานี้ลง ground

- ขา XTAL1 (ขา 19) ใช้เป็นตัวอินพุทเข้าออสซิเลเตอร์
- ขา XTAL2 (ขา 18) ใช้เป็นตัวเอาท์พุทจากออสซิเลเตอร์

## คำสั่งการใช้งาน MCS-51

ชุดคำสั่งที่ใช้ควบคุม MCS-51 สามารถสรุปได้โดยย่อดังตารางต่อไปนี้

Interrupt Response Time: Refer to Hardware Description Chapter.

**Instructions that Affect Flag Settings(1)**

| Instruction | Flag C | OV | AC | Instruction | Flag C | OV | AC |
|---|---|---|---|---|---|---|---|
| ADD | X | X | X | CLR C | 0 | | |
| ADDC | X | X | X | CPL C | X | | |
| SUBB | X | X | X | ANL C,bit | X | | |
| MUL | 0 | X | | ANL C,/bit | X | | |
| DIV | 0 | X | | ORL C,bit | X | | |
| DA | X | | | ORL C,/bit | X | | |
| RRC | X | | | MOV C,bit | X | | |
| RLC | X | | | CJNE | X | | |
| SETB C | 1 | | | | | | |

(1)Note that operations on SFR byte address 208 or bit addresses 209-215 (i.e., the PSW or bits in the PSW) will also affect flag settings.

**Note on instruction set and addressing modes:**

- **Rn** — Register R7–R0 of the currently selected Register Bank.
- **direct** — 8-bit internal data location's address. This could be an Internal Data RAM location (0–127) or a SFR [i.e., I/O port, control register, status register, etc. (128–255)].
- **@Ri** — 8-bit internal data RAM location (0–255) addressed indirectly through register R1 or R0.
- **#data** — 8-bit constant included in instruction.
- **#data 16** — 16-bit constant included in instruction.
- **addr 16** — 16-bit destination address. Used by LCALL & LJMP. A branch can be anywhere within the 64K-byte Program Memory address space.
- **addr 11** — 11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction.
- **rel** — Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditional jumps. Range is −128 to +127 bytes relative to first byte of the following instruction.
- **bit** — Direct Addressed bit in Internal Data RAM or Special Function Register.

| Mnemonic | | Description | Byte | Oscillator Period |
|---|---|---|---|---|
| **ARITHMETIC OPERATIONS** | | | | |
| ADD | A,Rn | Add register to Accumulator | 1 | 12 |
| ADD | A,direct | Add direct byte to Accumulator | 2 | 12 |
| ADD | A,@Ri | Add indirect RAM to Accumulator | 1 | 12 |
| ADD | A,#data | Add immediate data to Accumulator | 2 | 12 |
| ADDC | A,Rn | Add register to Accumulator with Carry | 1 | 12 |
| ADDC | A,direct | Add direct byte to Accumulator with Carry | 2 | 12 |
| ADDC | A,@Ri | Add indirect RAM to Accumulator with Carry | 1 | 12 |
| ADDC | A,#data | Add immediate data to Acc with Carry | 2 | 12 |
| SUBB | A,Rn | Subtract Register from Acc with borrow | 1 | 12 |
| SUBB | A,direct | Subtract direct byte from Acc with borrow | 2 | 12 |
| SUBB | A,@Ri | Subtract indirect RAM from ACC with borrow | 1 | 12 |
| SUBB | A,#data | Subtract immediate data from Acc with borrow | 2 | 12 |
| INC | A | Increment Accumulator | 1 | 12 |
| INC | Rn | Increment register | 1 | 12 |
| INC | direct | Increment direct byte | 2 | 12 |
| INC | @Ri | Increment direct RAM | 1 | 12 |
| DEC | A | Decrement Accumulator | 1 | 12 |
| DEC | Rn | Decrement Register | 1 | 12 |
| DEC | direct | Decrement direct byte | 2 | 12 |
| DEC | @Ri | Decrement indirect RAM | 1 | 12 |

| Mnemonic | | Description | Byte | Oscillator Period |
|---|---|---|---|---|
| **ARITHMETIC OPERATIONS (Continued)** | | | | |
| INC | DPTR | Increment Data Pointer | 1 | 24 |
| MUL | AB | Multiply A & B | 1 | 48 |
| DIV | AB | Divide A by B | 1 | 48 |
| DA | A | Decimal Adjust Accumulator | 1 | 12 |
| **LOGICAL OPERATIONS** | | | | |
| ANL | A,Rn | AND Register to Accumulator | 1 | 12 |
| ANL | A,direct | AND direct byte to Accumulator | 2 | 12 |
| ANL | A,@Ri | AND indirect RAM to Accumulator | 1 | 12 |
| ANL | A,# data | AND immediate data to Accumulator | 2 | 12 |
| ANL | direct,A | AND Accumulator to direct byte | 2 | 12 |
| ANL | direct,# data | AND immediate data to direct byte | 3 | 24 |
| ORL | A,Rn | OR register to Accumulator | 1 | 12 |
| ORL | A,direct | OR direct byte to Accumulator | 2 | 12 |
| ORL | A,@Ri | OR indirect RAM to Accumulator | 1 | 12 |
| ORL | A,# data | OR immediate data to Accumulator | 2 | 12 |
| ORL | direct,A | OR Accumulator to direct byte | 2 | 12 |
| ORL | direct,# data | OR immediate data to direct byte | 3 | 24 |
| XRL | A,Rn | Exclusive-OR register to Accumulator | 1 | 12 |
| XRL | A,direct | Exclusive-OR direct byte to Accumulator | 2 | 12 |
| XRL | A,@Ri | Exclusive-OR indirect RAM to Accumulator | 1 | 12 |
| XRL | A,# data | Exclusive-OR immediate data to Accumulator | 2 | 12 |
| XRL | direct,A | Exclusive-OR Accumulator to direct byte | 2 | 12 |
| XRL | direct,# data | Exclusive-OR immediate data to direct byte | 3 | 24 |
| CLR | A | Clear Accumulator | 1 | 12 |
| CPL | A | Complement Accumulator | 1 | 12 |

| Mnemonic | | Description | Byte | Oscillator Period |
|---|---|---|---|---|
| **LOGICAL OPERATIONS (Continued)** | | | | |
| RL | A | Rotate Accumulator Left | 1 | 12 |
| RLC | A | Rotate Accumulator Left through the Carry | 1 | 12 |
| RR | A | Rotate Accumulator Right | 1 | 12 |
| RRC | A | Rotate Accumulator Right through the Carry | 1 | 12 |
| SWAP | A | Swap nibbles within the Accumulator | 1 | 12 |
| **DATA TRANSFER** | | | | |
| MOV | A,Rn | Move register to Accumulator | 1 | 12 |
| MOV | A,direct | Move direct byte to Accumulator | 2 | 12 |
| MOV | A,@Ri | Move indirect RAM to Accumulator | 1 | 12 |
| MOV | A,# data | Move immediate data to Accumulator | 2 | 12 |
| MOV | Rn,A | Move Accumulator to register | 1 | 12 |
| MOV | Rn,direct | Move direct byte to register | 2 | 24 |
| MOV | Rn,# data | Move immediate data to register | 2 | 12 |
| MOV | direct,A | Move Accumulator to direct byte | 2 | 12 |
| MOV | direct,Rn | Move register to direct byte | 2 | 24 |
| MOV | direct,direct | Move direct byte to direct | 3 | 24 |
| MOV | direct,@Ri | Move indirect RAM to direct byte | 2 | 24 |
| MOV | direct,# data | Move immediate data to direct byte | 3 | 24 |
| MOV | @Ri,A | Move Accumulator to indirect RAM | 1 | 12 |

| Mnemonic | | Description | Byte | Oscillator Period |
|---|---|---|---|---|
| **DATA TRANSFER (Continued)** | | | | |
| MOV | @Ri,direct | Move direct byte to indirect RAM | 2 | 24 |
| MOV | @Ri, #data | Move immediate data to indirect RAM | 2 | 12 |
| MOV | DPTR, #data16 | Load Data Pointer with a 16-bit constant | 3 | 24 |
| MOVC | A,@A + DPTR | Move Code byte relative to DPTR to Acc | 1 | 24 |
| MOVC | A,@A + PC | Move Code byte relative to PC to Acc | 1 | 24 |
| MOVX | A,@Ri | Move External RAM (8-bit addr) to Acc | 1 | 24 |
| MOVX | A,@DPTR | Move External RAM (16-bit addr) to Acc | 1 | 24 |
| MOVX | @Ri,A | Move Acc to External RAM (8-bit addr) | 1 | 24 |
| MOVX | @DPTR,A | Move Acc to External RAM (16-bit addr) | 1 | 24 |
| PUSH | direct | Push direct byte onto stack | 2 | 24 |
| POP | direct | Pop direct byte from stack | 2 | 24 |
| XCH | A,Rn | Exchange register with Accumulator | 1 | 12 |
| XCH | A,direct | Exchange direct byte with Accumulator | 2 | 12 |
| XCH | A,@Ri | Exchange indirect RAM with Accumulator | 1 | 12 |
| XCHD | A,@Ri | Exchange low-order Digit indirect RAM with Acc | 1 | 12 |

| Mnemonic | | Description | Byte | Oscillator Period |
|---|---|---|---|---|
| **BOOLEAN VARIABLE MANIPULATION** | | | | |
| CLR | C | Clear Carry | 1 | 12 |
| CLR | bit | Clear direct bit | 2 | 12 |
| SETB | C | Set Carry | 1 | 12 |
| SETB | bit | Set direct bit | 2 | 12 |
| CPL | C | Complement Carry | 1 | 12 |
| CPL | bit | Complement direct bit | 2 | 12 |
| ANL | C,bit | AND direct bit to CARRY | 2 | 24 |
| ANL | C,/bit | AND complement of direct bit to Carry | 2 | 24 |
| ORL | C,bit | OR direct bit to Carry | 2 | 24 |
| ORL | C,/bit | OR complement of direct bit to Carry | 2 | 24 |
| MOV | C,bit | Move direct bit to Carry | 2 | 12 |
| MOV | bit,C | Move Carry to direct bit | 2 | 24 |
| JC | rel | Jump if Carry is set | 2 | 24 |
| JNC | rel | Jump if Carry not set | 2 | 24 |
| JB | bit,rel | Jump if direct Bit is set | 3 | 24 |
| JNB | bit,rel | Jump if direct Bit is Not set | 3 | 24 |
| JBC | bit,rel | Jump if direct Bit is set & clear bit | 3 | 24 |
| **PROGRAM BRANCHING** | | | | |
| ACALL | addr11 | Absolute Subroutine Call | 2 | 24 |
| LCALL | addr16 | Long Subroutine Call | 3 | 24 |
| RET | | Return from Subroutine | 1 | 24 |
| RETI | | Return from interrupt | 1 | 24 |
| AJMP | addr11 | Absolute Jump | 2 | 24 |
| LJMP | addr16 | Long Jump | 3 | 24 |
| SJMP | rel | Short Jump (relative addr) | 2 | 24 |



| Mnemonic | | Description | Byte | Oscillator Period |
|---|---|---|---|---|
| **PROGRAM BRANCHING (Continued)** | | | | |
| JMP | @A + DPTR | Jump indirect relative to the DPTR | 1 | 24 |
| JZ | rel | Jump if Accumulator is Zero | 2 | 24 |
| JNZ | rel | Jump if Accumulator is Not Zero | 2 | 24 |
| CJNE | A,direct,rel | Compare direct byte to Acc and Jump if Not Equal | 3 | 24 |
| CJNE | A,#data,rel | Compare immediate to Acc and Jump if Not Equal | 3 | 24 |

| Mnemonic | | Description | Byte | Oscillator Period |
|---|---|---|---|---|
| **PROGRAM BRANCHING (Continued)** | | | | |
| CJNE | Rn,#data,rel | Compare immediate to register and Jump if Not Equal | 3 | 24 |
| CJNE | @Ri,#data,rel | Compare immediate to indirect and Jump if Not Equal | 3 | 24 |
| DJNZ | Rn,rel | Decrement register and Jump if Not Zero | 2 | 24 |
| DJNZ | direct,rel | Decrement direct byte and Jump if Not Zero | 3 | 24 |
| NOP | | No Operation | 1 | 12 |

# ภาคผนวก ง

## ความรู้ทั่วไปเกี่ยวกับ ไมโครโปรเซสเซอร์ Z80

### 1.1 บทนำ

หลังจากที่ประสบผลสำเร็จทั้งในด้านการขายและการพัฒนาระบบไมโครโปรเซสเซอร์เบอร์ 8080 แล้ว ด้านซอฟต์แวร์ก็ได้รับการพัฒนาจนทำให้เครื่องไมโครคอมพิวเตอร์ที่ใช้ซีพียู 8080 สามารถทำงานได้อย่างกว้างขวางจนถึงระดับภาษาขั้นสูง เช่น BASIC, FORTRAN ฯลฯ นอกจากนี้การพัฒนาในระบบการจัดการทำงาน (operating system) ที่ใช้ร่วมกับอุปกรณ์เพอริเฟอรัล (peripheral) ต่าง ๆ ก็ได้รับการพัฒนามาถึงจุดที่ทำให้เครื่องไมโครคอมพิวเตอร์ที่ใช้ซีพียู 8080 เป็นระบบที่สมบูรณ์มากที่สุดระบบหนึ่ง และเพื่อให้ประสบผลสำเร็จทางด้านการตลาดในการขายไมโครโปรเซสเซอร์ บริษัทผู้ผลิตจึงต้องคำนึงถึงระบบซอฟต์แวร์ที่มีการพัฒนาไปมากแล้ว วิธีการหนึ่งก็ด้วยการสร้างไมโครโปรเซสเซอร์เบอร์ใหม่ที่มีโครงสร้างทางสถาปัตยกรรมดีกว่าตัวเดิมที่มีอยู่ แต่ยังสามารถใช้ชุดคำสั่งเดิมได้เพื่อจะได้ใช้กับระบบซอฟต์แวร์ที่มีอยู่แล้ว Z-80 เป็นไมโครโปรเซสเซอร์ที่มีจุดเริ่มต้นมาจากการพัฒนาและสร้างขึ้นของทีมวิศวกรบริษัทไซลอก (Zilog) Z-80 เป็นไมโครโปรเซสเซอร์ขนาด 8 บิต ซึ่งมีลักษณะที่น่าสนใจดังนี้

1. Z-80 มีลักษณะทางซอฟต์แวร์ที่สามารถนำไปใช้แทนไมโครโปรเซสเซอร์เบอร์ 8080 ได้

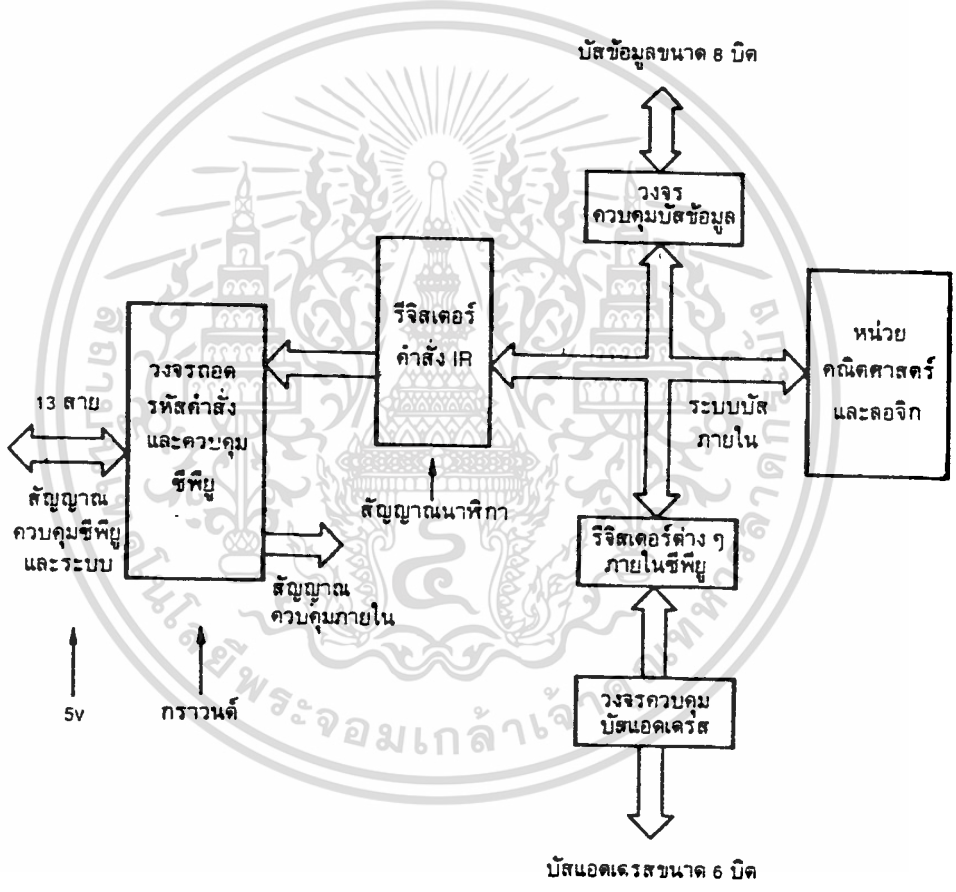
2. Z-80 มีลักษณะพิเศษทางฮาร์ดแวร์หลายประการ เช่น มีโครงสร้างที่มีความสามารถรวมอยู่ในชิปเดียว ใช้อัตราของสัญญาณนาฬิกาสูงถึง 4 เมกะเฮิรตซ์ (MHz) ในปัจจุบันสามารถเพิ่มอัตราเร็วของสัญญาณนาฬิกาได้สูงกว่า 8 เมกะเฮิรตซ์ ใช้แหล่งจ่ายไฟเลี้ยงเพียงชุดเดียวคือ 5 โวลต์ และต้องการสัญญาณนาฬิกาเพียงเฟสเดียว

3. อุปกรณ์ที่เป็นชิปประกอบหาได้ง่าย เช่น ชิปที่ใช้ต่อกับเพอริเฟอรัลต่าง ๆ ซึ่งใช้ในการอินเตอร์เฟสข้อมูลทั้งแบบอนุกรมและแบบขนาน

4. ปัจจุบันมีผู้ผลิตชิป Z-80 ด้วยเทคโนโลยีอื่น ๆ อีก เช่น CMOS ทำให้สิ้นเปลืองกำลังงานน้อยมาก

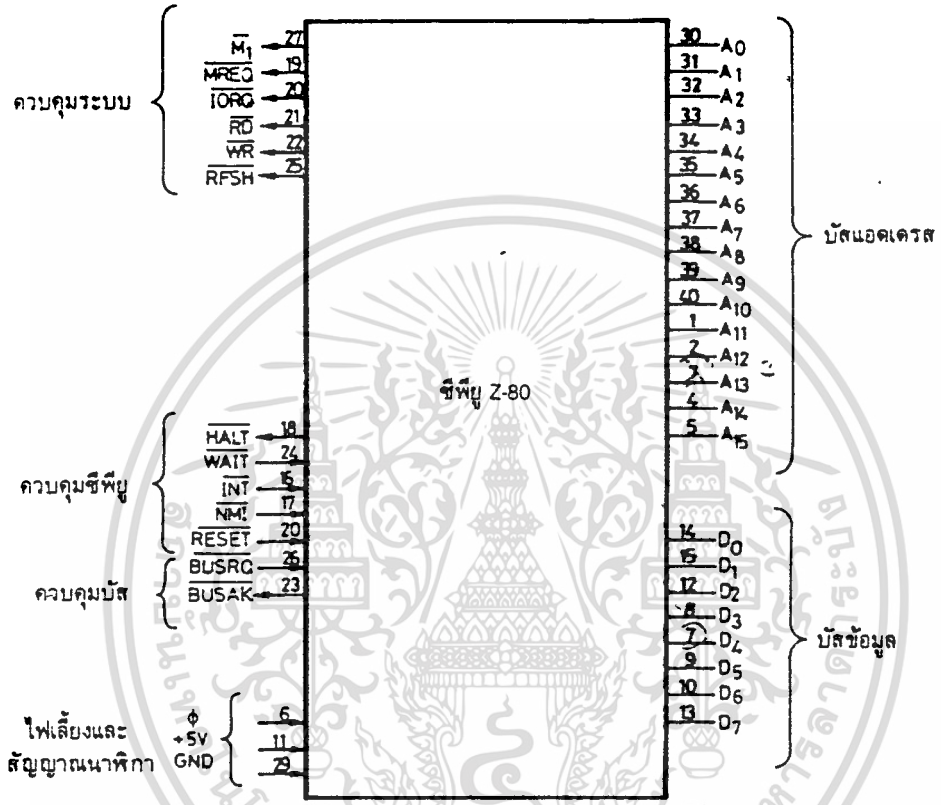
## 1.2 โครงสร้างทั่วไปของซีพียู

โครงสร้างทั่วไปของซีพียู Z-80 มีลักษณะคล้ายคลึงกับไมโครโปรเซสเซอร์ 8080 มาก แต่มีข้อแตกต่างกันในรายละเอียดบางอย่างที่ Z-80 มีมากกว่า ลักษณะโครงสร้างของ Z-80 แสดงได้ดังรูปที่ 1.1




**รูปที่ 1.1** โครงสร้างภายในของซีพียู Z-80

โครงสร้างภายในของซีพียูจะประกอบด้วยบัสข้อมูลขนาด 8 บิต ซึ่งเป็นบัสชนิดสองทิศทางคือ ข้อมูลสามารถวิ่งเข้าหรือออกจากซีพียูได้ และบัสแอดเดรสซึ่งเป็นบัสขนาด 16 บิต ที่จะทำให้ขีดความสามารถในการอ้างถึงแอดเดรสได้โดยตรงถึง $2^{16}$ หรือ 64 กิโลไบต์ (KB) นอกจากนี้บัสแอดเดรสยังเป็นสายสำคัญในการอ้างถึงแอดเดรสของหน่วยเพอริเฟอรัลที่เป็นอินพุตหรือเอาต์พุตด้วย สายบัสควบคุมนี้จะประกอบไปด้วยสัญญาณควบคุมต่างๆ ซึ่งมีอยู่ทั้งหมด 13 สาย ลักษณะการจัดสายสัญญาณแสดงได้ดังรูปที่ 1.2




**รูปที่ 1.2** แสดงลักษณะการจัดสัญญาณไอซีของ Z-80

รายละเอียดและหน้าที่ที่สำคัญของสายสัญญาณต่าง ๆ มีดังนี้

$A_0 - A_{15}$ เป็นสายของบัสแอดเดรสจำนวน 16 สาย ซึ่งส่วนภายในของเอาต์พุตเป็นลอจิก 3 สถานะ (tristate output) และจะถูกอีนาเบิล (enable) เลือกว่าเวลาใดเป็นสัญญาณแอดเดรสของหน่วยความจำหรือของอุปกรณ์อินพุตเอาต์พุตใด ทั้งนี้เพราะสายของบัสแอดเดรสยังทำหน้าที่เป็นตัวอ้างแอดเดรสสำหรับอุปกรณ์อินพุตเอาต์พุตอีกด้วย

$D_0 - D_7$ เป็นสายของบัสข้อมูลจำนวน 8 สาย ลักษณะของสายนี้เป็นลอจิก 3 สถานะสองทิศทาง (tristate input/output) เพื่อเลือกทิศทางการไหลของข้อมูลระหว่างซีพียูกับหน่วยความจำหรืออุปกรณ์อินพุตเอาต์พุต

$\overline{M_1}$ ลักษณะจะเป็นสัญญาณเอาต์พุต โดยส่งสัญญาณออกมาเพื่อบอกให้ทราบว่ากำลังอยู่ในสภาวะเฟตช์ (fetch) โดยแอคตีฟที่ลอจิก "0"

$\overline{\text{MREQ}}$ เป็นสายสัญญาณเอาต์พุตลอจิก 3 สถานะซึ่งจะบอกว่า ขณะนี้สัญญาณที่บัสแอดเดรสมีค่าแอดเดรสเพื่อเขียนหรืออ่านในหน่วยความจำ การแอดดีฟจะแอดดีฟที่ลอจิก "0"

$\overline{\text{IORQ}}$ เป็นสายสัญญาณเอาต์พุตซึ่งจะบอกว่า ขณะนี้สัญญาณในบัสแอดเดรสจาก $A_0-A_7$ มีค่าของอินพุตเอาต์พุตอยู่ ประโยชน์ของสัญญาณนี้เพื่อดีเท็กรหัสแอดเดรสในการเขียนหรือการอ่านข้อมูลจากเพอริเฟอรัล

$\overline{\text{RD}}$ เป็นสายสัญญาณเอาต์พุตที่จะบอกให้ทราบว่า ขณะนี้ซีพียูต้องการจะอ่านข้อมูลจากหน่วยความจำหรือจากอุปกรณ์อินพุตเอาต์พุต

$\overline{\text{WR}}$ เป็นสายสัญญาณเอาต์พุตเพื่อจะบอกให้ทราบว่า ขณะนี้ซีพียูต้องการจะเขียนข้อมูลในหน่วยความจำหรือในอุปกรณ์อินพุตเอาต์พุต

$\overline{\text{RFSH}}$ เป็นสายที่ส่งสัญญาณเพื่อจะบอกว่า ขณะนี้สายของแอดเดรสจะบรรจุข้อมูลแอดเดรสสำหรับการรีเฟรชหน่วยความจำชนิดไดนามิก

$\overline{\text{HALT}}$ เป็นสายสัญญาณที่จะแอดดีฟเมื่อซีพียูกระทำคำสั่ง HALT โดยจะแอดดีฟให้ลอจิก "0"

$\overline{\text{WAIT}}$ เป็นสายสัญญาณที่จะบอกให้ทราบว่า ขณะนี้หน่วยความจำหรืออุปกรณ์อินพุตเอาต์พุตยังไม่พร้อมที่จะรับหรือส่งผ่านข้อมูล คือ เมื่อส่งสัญญาณนี้เข้าไป ซีพียูจะหยุดรอจนกว่าเลิกสัญญาณ $\overline{\text{WAIT}}$

$\overline{\text{INT}}$ เป็นสัญญาณจากอุปกรณ์อินพุตเอาต์พุตที่จะอินเตอร์รัพต์ซีพียู การอินเตอร์รัพต์จะมีหลายโหมดซึ่งจะเรียกการอินเตอร์รัพต์แบบนี้ว่า มาสเคเบิลอินเตอร์รัพต์ (maskable interrupt)

$\overline{\text{RESET}}$ เป็นสัญญาณที่จะส่งเข้าไปรีเซตซีพียู หรือทำให้โปรแกรมเคาน์เตอร์มีค่าเป็น 0

$\overline{\text{NMI}}$ เป็นสัญญาณอินเตอร์รัพต์แบบที่เรียกว่า นอนมาสเคเบิลอินเตอร์รัพต์ (non-maskable interrupt)

$\overline{\text{BUSRQ}}$ เป็นการส่งสัญญาณเพื่อบอกซีพียูว่า ขณะนี้จะต้องการใช้บัส ซึ่งทำให้ซีพียูควบคุมบัสโดยใช้หลักการลอจิก 3 สถานะในการทำให้บัสแอดเดรสและบัสข้อมูลแยกออกจากระบบในซีพียู เพื่อให้หน่วยความจำและอุปกรณ์อินพุต-เอาต์พุตใช้บัสในการเคลื่อนย้ายข้อมูลระหว่างกัน

$\overline{\text{BUSAK}}$ เป็นสัญญาณที่ส่งออกไปจากซีพียู เพื่อบอกว่าขณะนี้ซีพียูไม่ได้ใช้บัสแล้ว

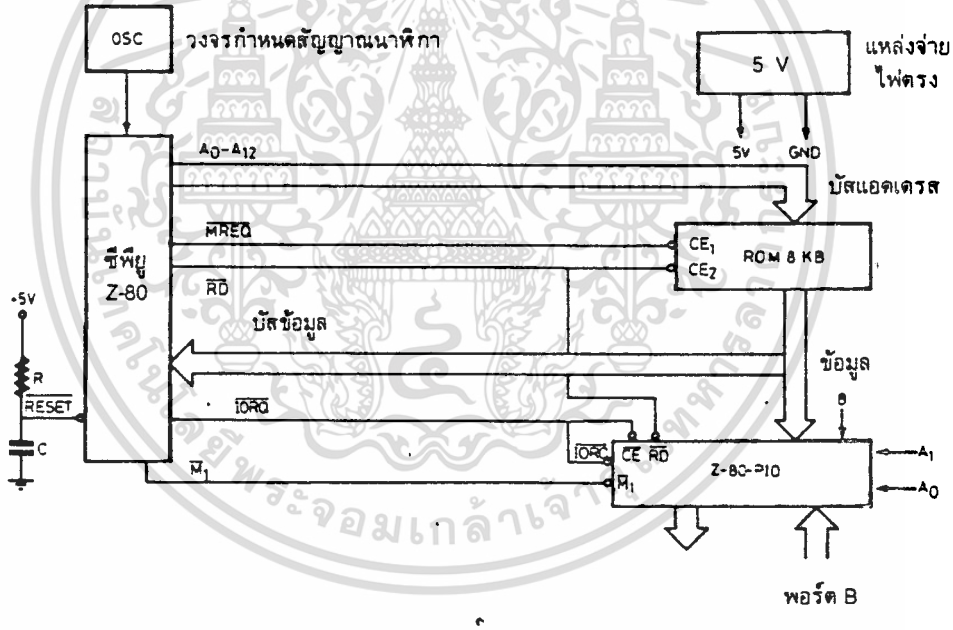
*หมายเหตุ* การใช้เครื่องหมายบาร์ (—) อยู่เหนือสัญลักษณ์ แสดงว่าเป็นการแอดดีฟที่ลอจิก "0"

## 1.3 ระบบทางฮาร์ดแวร์ของไมโครคอมพิวเตอร์ Z-80

ระบบฮาร์ดแวร์พื้นฐานของไมโครคอมพิวเตอร์ Z-80 ประกอบด้วย

1. แหล่งจ่ายไฟตรงขนาด 5 โวลต์
2. วงจรกำเนิดสัญญาณนาฬิกา
3. อุปกรณ์หน่วยความจำ RAM (random access memory) หรือ ROM (read only memory)
4. วงจรอินพุตเอาต์พุต
5. ซีพียู Z-80

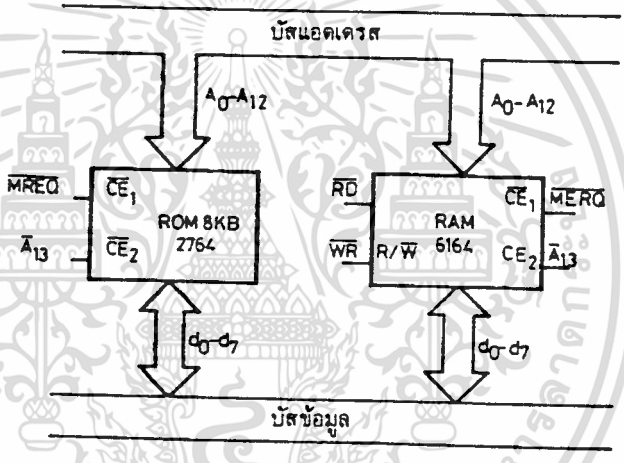
ระบบที่ประกอบด้วยสิ่งต่าง ๆ เหล่านี้ สามารถเขียนเป็นแผนผังได้ดังแสดงในรูปที่ 1.3




**รูปที่ 1.3** ระบบไมโครคอมพิวเตอร์พื้นฐานของ Z-80

จากรูปที่ 1.3 จะเห็นว่า ซีพียูต้องการแหล่งจ่ายไฟขนาด 5 โวลต์ กราวนด์ และวงจรกำเนิดสัญญาณนาฬิกา สำหรับการต่อหน่วยความจำ ROM นั้น สามารถต่อบัสแอดเดรสเข้ากับแอดเดรสของหน่วยความจำโดยตรงได้ และบัสข้อมูลก็จะต่อโดยตรงกับข้อมูลอินพุตและข้อมูลเอาต์พุตของระบบ ส่วนสัญญาณที่จะควบคุมหน่วยความจำประกอบด้วย สัญญาณ $\overline{\text{MREQ}}$ ซึ่งเป็นสัญญาณเพื่อบอกว่าซีพียูต้องการติดต่อกับหน่วยความจำ และเนื่องจากหน่วยความจำในรูปที่ 1.3 เป็น ROM จึงต้องมีสัญญาณ $\overline{\text{RD}}$ เพื่อบอกว่าเมื่อไรจึงจะเป็นการอ่าน

สำหรับ Z-80-PIO ชิปไอซีนี้จะทำหน้าที่ในการแยกอินพุตและเอาต์พุต ซึ่งจะได้พอร์ตเบอร์ต่าง ๆ คือ พอร์ต A พอร์ต B พอร์ต C และพอร์ต D และจากการกำหนดของ $A_0$ และ $A_1$ สัญญาณควบคุม Z-80-PIO คือ สัญญาณ $\overline{IORQ}$ ซึ่งเป็นสัญญาณที่จะบอกว่า ขณะนี้ซีพียูต้องการติดต่อกับอุปกรณ์อินพุตเอาต์พุต และเราต่อสัญญาณ $\overline{RD}$ เข้ามาควบคุมเพื่อจะกำหนดว่า เมื่อไรจึงจะเป็นการอ่านหรือการเขียน ส่วนตัวอุปกรณ์ $\overline{RC}$ ที่ต่ออยู่ที่ขารีเซตจะทำหน้าที่หน่วงเวลาหรือเริ่มต้นการรีเซตซีพียูในขณะเปิดไฟเลี้ยงเข้าวงจร เราสามารถเพิ่มขยายการต่อหน่วยความจำ RAM ได้ดังแสดงในรูปที่ 1.14

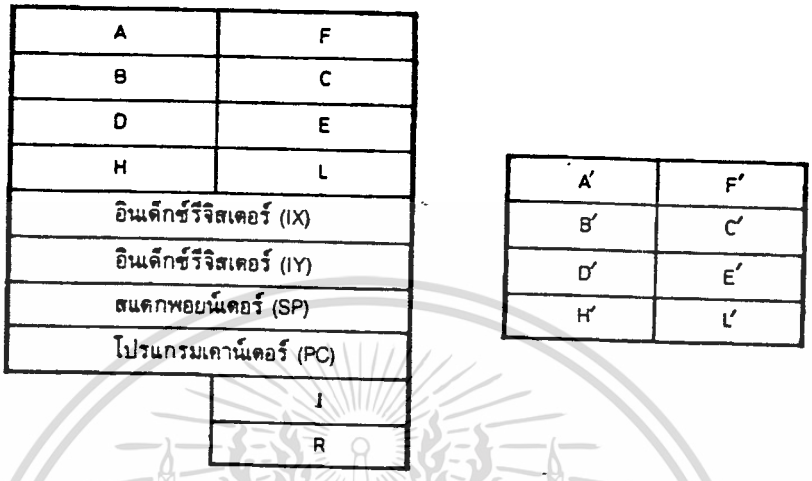



**รูปที่ 1.4** การต่อ RAM และ ROM ขนาด 8 KB

ในระบบคอมพิวเตอร์นั้น การควบคุมการทำงานเกี่ยวกับการอ่านหรือการเขียนจะใช้สัญญาณ $\overline{RD}$ และสัญญาณ $\overline{WR}$ และจากระบบในรูปที่ 1.4 จะทำการเซตส่วนของหน่วยความจำ ROM ให้อยู่ในแอดเดรส 0000H จนถึงแอดเดรส 1FFFH หรืออยู่ใน 8 กิโลไบต์แรกจาก 0 นั่นเอง สำหรับหน่วยความจำ RAM นั้นจะเริ่มที่แอดเดรส 2000H เป็นต้นไป เราสามารถควบคุมด้วยสัญญาณแอดเดรส $A_{13}$ นั่นเอง

## 1.4 โครงสร้างภายในซีพียู

สิ่งที่น่าสนใจและควรทำความเข้าใจเกี่ยวกับโครงสร้างของซีพียูในขั้นแรกก็คือ ส่วนของรีจิสเตอร์ต่าง ๆ และหน้าที่ที่สำคัญของแต่ละรีจิสเตอร์ ซึ่ง Z-80 จะประกอบด้วยรีจิสเตอร์ภายในซีพียูที่เป็นขนาด 8 บิต มีทั้งหมด 18 ตัว และเป็นรีจิสเตอร์ขนาด 16 บิตอีก 4 ตัว รูปโครงของรีจิสเตอร์ภายในซีพียูสามารถเขียนเป็นแผนผังได้ดังรูปที่ 1.5




**รูปที่ 1.5** โครงสร้างของรีจิสเตอร์ที่มีอยู่ในซีพียู Z-80

รีจิสเตอร์ในซีพียูมีหน้าที่และการใช้งานที่แตกต่างกัน บางตัวใช้งานเฉพาะอย่าง บางตัวเป็นรีจิสเตอร์ที่สามารถใช้งานได้ทั่วไป สำหรับรีจิสเตอร์ที่ใช้งานเฉพาะอย่างมีรายละเอียดหน้าที่และการทำงานดังนี้

1. **โปรแกรมเคาน์เตอร์หรือ PC** (program counter) เป็นรีจิสเตอร์สำหรับเก็บแอดเดรสขนาด 16 บิต มีหน้าที่สำหรับให้ซีพียูเฟตช์คำสั่งในหน่วยความจำได้อย่างถูกต้อง และหลังจากที่ซีพียูได้กระทำคำสั่งเสร็จแล้ว ค่าในโปรแกรมเคาน์เตอร์จะเพิ่มค่าขึ้นโดยอัตโนมัติ

2. **สแตกพอยน์เตอร์หรือ SP** (stack pointer) เป็นรีจิสเตอร์ขนาด 16 บิต โดยซีพียูจะใช้สแตกพอยน์เตอร์เป็นตัวชี้ไปยังหน่วยความจำ เพื่อบอกว่าชั้นบนสุดของสแตกอยู่ที่ใด และถ้าซีพียูมีการกระทำตามคำสั่ง POP หรือคำสั่ง PUSH ก็จะเปลี่ยนค่าสแตกพอยน์เตอร์ไปโดยอัตโนมัติ เพื่อชี้ตำแหน่งบนสุดของสแตกในหน่วยความจำ

3. **อินเด็กซ์รีจิสเตอร์หรือ IX, IY** (index register) ทั้ง IX และ IY เป็นรีจิสเตอร์อิสระที่มีขนาด 16 บิต โดยปกติจะใช้เป็นฐานในการชี้ไปยังบริเวณหน่วยความจำที่เป็นทางผ่านเข้าออกของข้อมูล การอ้างแอดเดรสจะใช้วิธีเปรียบเทียบกับค่าใน IY เช่น ถ้าอ้างถึง ADD A,(IY + D) จะหมายความว่า ให้นำค่า D ไปบวกกับค่าใน IY เป็นค่าที่จะอ้างถึงหน่วยความจำ แล้วนำข้อมูลจากหน่วยความจำมาบวกกับรีจิสเตอร์ A

4. **รีจิสเตอร์อินเตอร์รัพต์เพจแอดเดรสหรือ I** (interrupt page address register) รีจิสเตอร์ประเภทนี้มีประโยชน์สำหรับการกระโดดไปกระทำโปรแกรมอื่นในขณะที่มีการอินเตอร์รัพต์ โดยรีจิสเตอร์ $I_7$ จะเป็นตัวกำหนดแอดเดรสในบิต $A_8 - A_{15}$ ส่วนแอดเดรส $A_0 - A_7$ จะมาจากลักษณะของคำสั่งอินเตอร์รัพต์ ซึ่งโดยปกติของ 8080 การอินเตอร์รัพต์ค่าแอดเดรสที่

ซีพียูจะกระโดดไปกระทำมักจะอยู่ที่แอดเดรสคัน ๆ แต่ถ้าเลือกโหมดการอินเตอร์รัพต์ของ Z-80 ที่ใช้รีจิสเตอร์ I จะสามารถกระโดดไปที่ใด ๆ ก็ได้

**5. รีจิสเตอร์รีเฟรชหน่วยความจำหรือ R** (memory refresh register) ซีพียูของ Z-80 จะมีรีจิสเตอร์ที่ใช้ในการรีเฟรชหน่วยความจำชนิดไดนามิก (DRAM) ได้ รีจิสเตอร์นี้เป็นรีจิสเตอร์ขนาด 8 บิต ซึ่งจะเก็บค่าแอดเดรสของหน่วยความจำที่ต้องการรีเฟรช โดยส่วนของแอดเดรสจะส่งไปในลักษณะแถวหรือคอลัมน์ของหน่วยความจำชนิด DRAM

สำหรับรีจิสเตอร์ PC และรีจิสเตอร์ SP จะมีลักษณะเหมือนกับ 8080 แต่รีจิสเตอร์ IX, IY, I และรีจิสเตอร์ R เป็นรีจิสเตอร์ที่เพิ่มขึ้นจาก 8080

**6. รีจิสเตอร์ที่ใช้งานทั่วไป** การใช้รีจิสเตอร์ทั่วไปใน Z-80 มีลักษณะคล้ายกับการใช้ใน 8080 มาก แต่มีรายละเอียดที่แตกต่างกันเล็กน้อย รีจิสเตอร์ทั่วไปของ Z-80 ที่เป็นขนาด 8 บิตมีทั้งหมด 14 ตัว และสามารถใช้รวมเป็นคู่รีจิสเตอร์ได้ 7 คู่ ลักษณะของรีจิสเตอร์และรหัสที่ใช้เป็นดังนี้

000 หมายถึง รีจิสเตอร์ B
001 หมายถึง รีจิสเตอร์ C
010 หมายถึง รีจิสเตอร์ D
011 หมายถึง รีจิสเตอร์ F
100 หมายถึง รีจิสเตอร์ H
101 หมายถึง รีจิสเตอร์ L
110 หมายถึง รหัสใช้แทนหน่วยความจำ
111 หมายถึง รีจิสเตอร์ A

ในการอ้างคู่รีจิสเตอร์ เราใช้รีจิสเตอร์ A ร่วมกับรีจิสเตอร์ F (F ในที่นี้คือแฟล็ก) แฟล็กของ Z-80 มีขนาด 8 บิต และจะเป็นเสมือนรีจิสเตอร์ทั่วไปหนึ่งรีจิสเตอร์ นอกจากนี้การใช้รีจิสเตอร์ B จะใช้คู่กับรีจิสเตอร์ C รีจิสเตอร์ D คู่กับรีจิสเตอร์ E และรีจิสเตอร์ H คู่กับรีจิสเตอร์ L รหัสที่จะใช้แทนคู่รีจิสเตอร์เหล่านี้คือ

00 ใช้แทนคู่ รีจิสเตอร์ BC
01 ใช้แทนคู่ รีจิสเตอร์ DE
10 ใช้แทนคู่ รีจิสเตอร์ HL
11 ใช้แทนคู่ รีจิสเตอร์ SP

ในการใช้งานเราอาจมองรูปข้อมูลในลักษณะ 8 บิตเพื่อกระทำโดยใช้รีจิสเตอร์ทั่วไปได้ แต่ถ้าเป็นข้อมูลขนาด 16 บิต เราอาจใช้คู่รีจิสเตอร์แทน

**7. รีจิสเตอร์ที่ใช้เก็บข้อมูลชั่วคราว** ใน Z-80 มีกลุ่มรีจิสเตอร์ที่นอกเหนือจากของ 8080 อยู่ชุดหนึ่งที่ทำหน้าที่เก็บข้อมูลชั่วคราว กลุ่มรีจิสเตอร์นี้ประกอบด้วย รีจิสเตอร์ A', F', B', C',

D', E', H', L' รีจิสเตอร์กลุ่มนี้ไม่สามารถกระทำในลักษณะทางลอจิกหรือทางคณิตศาสตร์ได้ แต่จะมีเพียงคำสั่งย้ายข้อมูลไปกลับระหว่างรีจิสเตอร์ A, F, B, C, D, E, H, L กับรีจิสเตอร์ A', F', B', C', D', E', H', L' ประโยชน์ของรีจิสเตอร์กลุ่มนี้ที่เห็นได้ชัดคือ สามารถเก็บรักษาสถานะข้อมูลของรีจิสเตอร์ที่สำคัญในขณะที่มีการอินเตอร์รัพต์ไว้ได้ โดยไม่ต้องอาศัยสแตกในหน่วยความจำเหมือน 8080

## 1.5 แฟล็ก

ซีพียูใน Z-80 จะประกอบด้วยแฟล็ก (flag) จำนวน 6 แฟล็ก และมีบิตที่ไม่ได้แสดงเป็นแฟล็กอีก 2 บิต รวมเป็น 8 บิตเพื่อประกอบเป็นรีจิสเตอร์ F ส่วนของแฟล็กแต่ละแฟล็กสามารถที่จะเซตหรือรีเซตตามการกระทำของคำสั่งที่ซีพียูกำลังทำงาน นอกจากนี้ซีพียูยังสามารถใช้แฟล็กในการตรวจสอบเพื่อกระทำเงื่อนไขต่าง ๆ ลักษณะการใช้แฟล็กจะใช้อักษรย่อแทนแฟล็กดังนี้

| | | |
|---|---|---|
| C | หมายถึง | แฟล็กตัวทด |
| N | หมายถึง | แฟล็กแสดงการบวกหรือการลบ |
| P/V | หมายถึง | แฟล็กแสดงพาริตี้และโอเวอร์โฟลว์ |
| H | หมายถึง | แฟล็กตัวทดช่วย |
| Z | หมายถึง | แฟล็กแสดงค่าศูนย์ |
| S | หมายถึง | แฟล็กเครื่องหมาย |
| X | หมายถึง | แฟล็กที่ไม่ได้ใช้ |

ในการกระทำคำสั่งต่าง ๆ ของ Z-80 บางคำสั่งอาจจะมีผลต่อแฟล็ก แต่บางคำสั่งก็ไม่มีผลต่อแฟล็ก การที่คำสั่งบางคำสั่งมีผลต่อแฟล็กทำให้เราสามารถตรวจสอบเงื่อนไขได้จากแฟล็ก เช่น การใช้คำสั่ง INC A หมายถึงให้เพิ่มค่ารีจิสเตอร์ A อีก 1 ผลที่เกี่ยวกับแฟล็กคือ

| | | |
|---|---|---|
| ถ้าผลลัพธ์เป็น 0 | Z | คือ 1 |
| ถ้าผลลัพธ์เป็นลบ | S | คือ 1 |
| ถ้ามีตัวทดในบิตที่ 3 | H | คือ 1 |
| ถ้าผลลัพธ์มีจำนวนเลข 1 เป็นเลขคู่ | P/V | คือ 1 |

*หมายเหตุ* คำสั่งนี้จะไม่เกี่ยวข้องกับแฟล็ก N และแฟล็ก C

ลักษณะของรีจิสเตอร์ F จะประกอบด้วยแฟล็กแต่ละบิตดังนี้

| S | Z | X | H | X | P/V | N | C |
|---|---|---|---|---|---|---|---|
| $b_7$ | $b_6$ | $b_5$ | $b_4$ | $b_3$ | $b_2$ | $b_1$ | $b_0$ |

ในบิต $b_5$ และ $b_3$ จะเป็นบิตที่ประกอบขึ้นมาโดยไม่มีความหมายในทางแฟล็ก แต่จะประกอบเพื่อให้รีจิสเตอร์ F ครบ 8 บิต การทำงานของซีพียูจึงสามารถโหลดข้อมูลจากรีจิสเตอร์ F ไปยังรีจิสเตอร์ A หรือโหลดจากรีจิสเตอร์ A กลับมายังรีจิสเตอร์ F ได้ รายละเอียดและหน้าที่ของแฟล็กแต่ละตัวมีดังนี้

1. **แฟล็กตัวทดหรือ C** (carry flag) แฟล็กตัวนี้เป็นแฟล็กที่ใช้สำหรับทดข้อมูลในรีจิสเตอร์ A เช่น เมื่อมีการบวกข้อมูลขนาด 8 บิต ผลบวกอาจเลยเป็น 9 บิต บิตที่เกินเลยจะทดเข้าไปเก็บไว้ที่แฟล็ก C นี้ ในทำนองเดียวกัน ถ้าซีพียูกระทำคำสั่งลบและมีการยืมค่าของแฟล็กตัวนี้ แฟล็ก C ก็จะได้รับการเซตให้มีค่าเป็น "1" เช่นกัน

2. **แฟล็กศูนย์หรือ Z** (zero flag) ในแฟล็กบิตนี้จะได้รับการเซตให้มีค่าเป็น "1" ถ้าผลของการกระทำทำให้รีจิสเตอร์ A มีค่าเป็น "1" นอกเหนือจากนี้มันจะกระทำการรีเซต

3. **แฟล็กเครื่องหมายหรือ S** (sign flag) ลักษณะของแฟล็กนี้จะบอกถึงการกระทำของซีพียูว่า ผลลัพธ์ที่เกิดขึ้นมีค่าเป็นบวกหรือลบ ถ้าเครื่องหมายของตัวเลขเป็นลบจะปรากฏค่า "1" ในบิตนี้

4. **แฟล็กพาริตี้หรือโอเวอร์โฟลว์หรือ P/V** (parity or over flow flag) เป็นแฟล็กที่ใช้สำหรับเป็นตัวบอกพาริตี้ของผลลัพธ์ในแอกคูมิวเลเตอร์เมื่อให้มีการกระทำทางลอจิก และยังใช้แสดงสภาวะของค่าที่เกินกำหนดใน 8 บิต (1 บิตที่เป็นเครื่องหมายรวมกับ 7 บิตที่เป็นขนาด) นั่นคือค่าสูงสุดที่เป็นไปได้หรือเท่ากับ +127 และ −128 ถ้าค่ามากกว่า +127 หรือน้อยกว่า −128 ก็จะเกิดการเซตค่าเป็น 1 ของบิตนี้ขึ้น ดังตัวอย่าง

| | | | |
|---|---|---|---|
| + 120 | 0111 | 1000 | |
| + 105 | 0110 | 1001 | + |
| 0 | 1110 | 0001 = −95 (ผิด) | เกิดโอเวอร์โฟลว์ |

บิตพาริตี้จะได้รับการเซตให้มีค่าเป็น "1"

ลองพิจารณาเลขจำนวนลบ

| | | | |
|---|---|---|---|
| − 5 | 1111 | 1011 | |
| −16 | 1111 | 0000 | |
| 1 | 1110 | 1011 = −21 (ถูก) | ไม่เกิดโอเวอร์โฟลว์ |

ในกรณีนี้แฟล็กจะไม่ได้รับการเซตคือ จะมีค่าเป็น "0" เพื่อบอกว่าผลลัพธ์ไม่เกิดโอเวอร์โฟลว์ นอกจากการใช้ในกรณีนี้แล้ว ในเรื่องการกระทำทางลอจิก เช่น AND, OR, XOR ผลลัพธ์ที่ได้จะได้รับการตรวจสอบว่ามีพาริตี้คู่หรือพาริตี้คี่ โดยถ้าเป็นพาริตี้คู่ แฟล็กนี้จะได้รับการเซตให้มีค่าเป็น "1"

5. **แฟล็กตัวทดช่วยหรือ H** (half carry flag) แฟล็กตัวนี้จะเป็นแฟล็กที่ทำหน้าที่เป็นตัวทดหรือตัวยืมของตัวเลข BCD

**6. แฟล็กการลบหรือ N** (subtract flag) เนื่องจากในการกระทำทางคณิตศาสตร์ของตัวเลข BCD เพื่อจะได้มีการรับรู้ในการปรับค่าเมื่อกระทำคำสั่ง DAA ได้ถูกต้อง แฟล็กนี้จะเป็นตัวบอกว่าคำสั่งที่ถูกกระทำเป็นการบวกหรือลบ โดยถ้ากระทำคำสั่งลบ แฟล็กบิตนี้จะได้รับการเซตให้มีค่าเป็น "1"

## 1.6 วิธีการอ้างแอดเดรสของ Z-80

ในการทำงานของ Z-80 เกือบทั้งหมดของคำสั่งที่ใช้ใน Z-80 จะทำงานโดยการกระทำร่วมระหว่างรีจิสเตอร์ต่าง ๆ ในซีพียูกับหน่วยความจำอุปกรณ์อินพุตเอาต์พุต วิธีการกระทำร่วมกันนี้จำเป็นจะต้องอ้างตำแหน่ง วิธีการอ้างตำแหน่งหรือแอดเดรสมีดังต่อไปนี้

**1.6.1 การอ้างแอดเดรสแบบอิมมีเดียต** (immediate addressing) การอ้างแอดเดรสแบบนี้จะอาศัยไบต์ที่ตามออปโค้ดเป็นข้อมูลหรือเป็นตัวโอเปอร์แรนด์โดยตรง เช่น

```
LD A,5FH
```

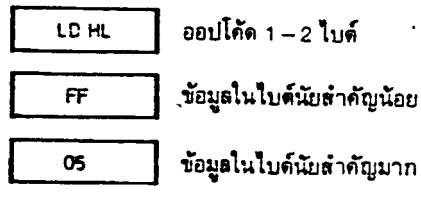
ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ


ตัวอย่างของคำสั่งนี้แสดงให้เห็นได้โดยการโหลดรีจิสเตอร์ด้วยค่าคงที่ ซึ่งจะเก็บค่าคงที่นี้ไว้ในไบต์ที่อยู่ถัดมา

**1.6.2 การอ้างแอดเดรสแบบขยายข้อมูลชนิดอิมมีเดียต** (immediate extended addressing) การอ้างแอดเดรสแบบนี้เป็นการขยายแอดเดรสในแบบที่ 1 คือ ใช้โอเปอร์แรนด์ที่เป็นข้อมูลขนาด 2 ไบต์ ดังตัวอย่าง

```
LD HL,05FFH
```

ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ




จากตัวอย่างนี้เป็นการโหลดข้อมูลขนาด 2 ไบต์ไปที่คู่รีจิสเตอร์ภายในซีพียู

**1.6.3 การอ้างแอดเดรสแบบมอดิฟายด์เพจศูนย์** (modified page zero addressing) ใน Z-80 มีคำสั่งพิเศษอยู่คำสั่งหนึ่งที่ทำหน้าที่เหมือนคำสั่ง CALL คือ คำสั่ง RST หรือ RESTART ลักษณะการทำงานในการอ้างแอดเดรสคือ คำสั่งนี้สามารถกำหนดค่าให้กับโปรแกรมเคาน์เตอร์ได้โดยตรง ค่าที่กำหนดให้จะเป็นแอดเดรสที่อยู่ในเพจศูนย์ ลักษณะของคำสั่งนี้ประกอบขึ้นจากไบต์เดียวเท่านั้น จึงใช้ประโยชน์ได้หลายอย่าง เช่น ใช้เป็นคำสั่งให้กระทำในขณะที่มีการอินเตอร์รัพต์ นอกจากนี้ข้อมูลเดิมในโปรแกรมเคาน์เตอร์ก่อนการเปลี่ยนแปลงจะได้รับการเก็บรักษาไว้ได้อีกด้วย แต่เงื่อนไขในการกระโดดไปที่แอดเดรสในเพจศูนย์ยังมีขอบเขตจำกัดคือ จะกระโดดไปได้เพียง 8 แอดเดรสเท่านั้น ค่าแอดเดรสที่มันจะไปได้คือ แอดเดรส $b_5$, $b_4$, $b_3$, 000 นั่นคือ $00_8$, $10_8$, ..., $70_8$ เท่านั้น ลักษณะของคำสั่งคือ

RST P

ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ


เมื่อ P = 00$_8$ t = 000

P = 10$_8$ t = 001

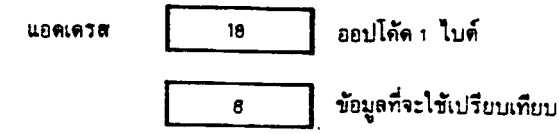
ฯลฯ

จะเห็นว่าการใช้คำสั่ง RST P เพียงไบต์เดียวสามารถกำหนดคำสั่งในการทำให้เกิดการกระโดดไปยังแอดเดรสต่าง ๆ ได้ถึง 8 ค่าตามที่กำหนด ส่วนค่าในโปรแกรมเคาน์เตอร์เดิมจะเก็บรักษาไว้ในชั้นของสแตก

**1.6.4 การอ้างแอดเดรสแบบเปรียบเทียบ** (relative addressing) การอ้างแอดเดรสแบบนี้จะใช้ข้อมูลไบต์ที่อยู่ตามหลังออปโค้ด เพื่อบอกตำแหน่งว่าแอดเดรสที่อ้างถึงอยู่ห่างจากค่าในโปรแกรมเคาน์เตอร์เท่าใด เช่น

JR e (jump relative)

ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ

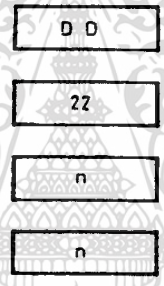


เมื่อซีพียูกระทำคำสั่งนี้เสร็จ ค่าในโปรแกรมเคาน์เตอร์จะมีค่าเป็น PC+2+e นั่นคือ ค่าใหม่ที่จะอ้างถึงอยู่ห่างจากคำสั่งที่กระทำแล้วด้วยค่า e นั่นเอง ค่า e ที่ซีพียูมองเห็นจะเป็นลักษณะของตัวเลข 2's คอมพลีเมนต์ ดังนั้นค่าที่จะอ้างถึงได้จึงอยู่ระหว่าง +127 กับ −128 จาก PC+2

**1.6.5 การอ้างแอดเดรสแบบขยายข้อมูลแอดเดรส** (extended addressing) วิธีการนี้จะใช้ข้อมูล 2 ไบต์ตามที่ออปโค้ด ข้อมูลนี้คือ nn ซึ่งจะเป็นตัวกำหนดค่าของแอดเดรสที่จะกระทำใหม่ เช่น CALL nn หมายถึง การเรียกโปรแกรมย่อยที่ตำแหน่ง nn เช่น

LD (nn). IX

ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ

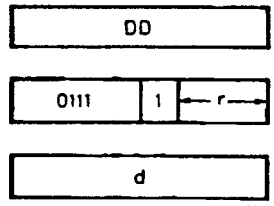



ความหมายของคำสั่งนี้คือ นำค่า $IX_H$ ไปเก็บไว้ในหน่วยความจำที่ตำแหน่งแอดเดรส nn+1 และค่าของ $IX_L$ เก็บไว้หน่วยความจำที่ตำแหน่งแอดเดรส nn

**1.6.6 การอ้างแอดเดรสโดยใช้อินเด็กซ์รีจิสเตอร์** (index register addressing) โดยที่ Z-80 มีอินเด็กซ์รีจิสเตอร์ถึง 2 ตัว คือ IX และ IY วิธีการใช้อินเด็กซ์รีจิสเตอร์ร่วมในการอ้างแอดเดรสนั้น จะใช้ค่า IX หรือ IY เป็นฐานเพื่อรวมกับค่าที่ตามหลังออปโค้ด แล้วนำมารวมเป็นแอดเดรสที่ต้องการ เช่น

LD (IX+d). A

ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ



ลักษณะของคำสั่งนี้เป็นการนำเอาข้อมูลของรีจิสเตอร์ A ไปเก็บไว้ในหน่วยความจำที่อ้างแอดเดรส โดยที่ค่า IX รวมกับค่า d ปกติค่า d จะได้รับการรวมโดยอยู่ในรูป 2's คอมพลีเมนต์ ดังนั้นจึงทำให้ค่า d แปรได้จากค่า +127 ถึง −128

**1.6.7 การอ้างแอดเดรสโดยใช้รีจิสเตอร์** (register addressing) การออกแบบกลุ่มคำสั่งในขอบเขตของจำนวนบิตออปโค้ดที่จำกัด ต้องหาวิธีที่ให้ได้ประสิทธิภาพดีที่สุด วิธีหนึ่งที่ใช้คือ การกำหนดรหัสของรีจิสเตอร์ เช่น ถ้าใช้วิธีรีจิสเตอร์ 8 ตัว ก็ใช้รหัส 3 บิต ดังนั้นกลุ่มคำสั่งโหลดสามารถใช้คำสั่งเพียง 1 ไบต์หรือ 8 บิตเพื่อทำการโหลดข้อมูลระหว่างรีจิสเตอร์ได้ เช่น คำสั่ง

LD C, B

ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ


**1.6.8 การอ้างแอดเดรสแบบอิมไพล์** (implied addressing) ในกรณีนี้ซีพียูจะตีความหมายเองโดยตรงว่า รีจิสเตอร์ตัวหนึ่งในซีพียูจะเป็นโอเปอร์แรนด์ เช่น คำสั่ง

ADD A,r

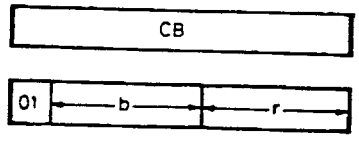
ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ


จากคำสั่งนี้จะเห็นว่า เราไม่ต้องกำหนดรหัสของรีจิสเตอร์ A เลย แต่เครื่องจะตีความแล้วทราบเองว่าเป็นคำสั่งที่ต้องกระทำร่วมกับรีจิสเตอร์ A

**1.6.9 การอ้างแอดเดรสเพื่อเข้าสู่บิตต่าง ๆ** (bit addressing) Z-80 มีคำสั่งพิเศษที่สามารถกระทำการเซตหรือรีเซต หรือตรวจสอบบิตใดบิตหนึ่งใน 8 บิตได้ เช่น คำสั่ง

RES b, r (reset bit b of operand r)

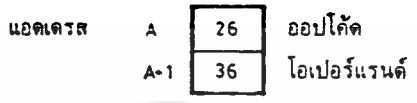
ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ



เรายังสามารถใช้วิธีการโหลดข้อมูลแบบอิมมีเดียตได้โดยใช้คำสั่ง

LD H, 36

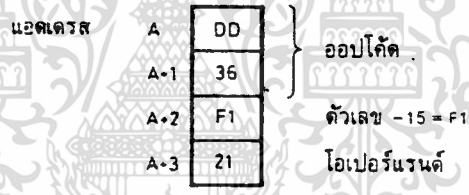
ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ




แต่การโหลดข้อมูลลงในหน่วยความจำอีกวิธีหนึ่งที่มีประสิทธิภาพสูงคือ การโหลดโดยใช้อินเด็กซ์รีจิสเตอร์เป็นตัวรับข้อมูล และใช้วิธีการอิมมีเดียตเป็นการกำหนดแหล่งเริ่มต้น (source) ของข้อมูล เช่น

LD (IX-15), 21

ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ




จากลักษณะคำสั่งนี้จะเป็นการนำตัวเลข 21 ไปเก็บไว้ในหน่วยความจำที่ตำแหน่งแอดเดรส IX-15 นอกจากนี้เรายังสามารถใช้คำสั่งโหลดทำการโหลดข้อมูลในรูปแบบของการโหลดข้อมูลขนาด 16 บิตได้อีกด้วย และเพื่อลดขนาดของคำสั่งให้สั้นขึ้น ตัวคำสั่งจึงเป็นลักษณะของการแลกเปลี่ยนข้อมูลของคู่รีจิสเตอร์ต่าง ๆ

**1.7.2 กลุ่มคำสั่งในการค้นหาข้อมูลและเคลื่อนย้ายข้อมูลเป็นกลุ่ม** Z-80 มีคำสั่งที่ทำให้การทำงานเป็นไปอย่างมีประสิทธิภาพ โดยสามารถลดขนาดของตัวโปรแกรมลงได้มาก ลักษณะของกลุ่มคำสั่งในกลุ่มนี้จะอาศัยการทำงานร่วมกันของคู่รีจิสเตอร์ภายในซีพียู 3 คู่ คือ

HL เป็นคู่รีจิสเตอร์ที่อ้างถึงตำแหน่งจุดต้นทาง

DE เป็นคู่รีจิสเตอร์ที่อ้างถึงตำแหน่งจุดปลายทาง

BC เป็นตัวนับจำนวนไบต์

ในการค้นหาข้อมูล เราใช้กลุ่มคำสั่งดังนี้คือ

CPI (compare with increment)

ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ

ในที่นี้จะใช้รหัส b แทนโอเปอร์แรนด์ว่าเป็นบิตใดในรีจิสเตอร์ เช่น

b เท่ากับ 000 หมายถึง บิต 0

b เท่ากับ 111 หมายถึง บิต 7

ดังนั้นจึงสามารถทำให้บิตใด ๆ ในรีจิสเตอร์เป็น 0 ได้ เช่น คำสั่ง RES 0, A จะหมายถึง ทำให้บิต 0 ของรีจิสเตอร์ A เป็น 0

*หมายเหตุ* เราสามารถใช้การอ้างแอดเดรสตามที่กล่าวมาแล้วหลาย ๆ แบบมารวมกันได้

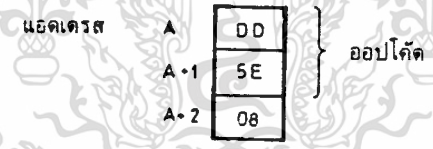
## 1.7 กลุ่มคำสั่ง

คำสั่งของ Z-80 มีถึง 158 คำสั่ง โดยคำสั่งส่วนหนึ่งเป็นคำสั่งที่มีใน 8080 (78 คำสั่ง) คำสั่งเหล่านี้สามารถแบ่งออกเป็นกลุ่ม ๆ ได้ดังนี้

**1.7.1 กลุ่มคำสั่งเกี่ยวกับการโหลดข้อมูลและแลกเปลี่ยนข้อมูล** การโหลดเป็นการเคลื่อนย้ายข้อมูลที่สำคัญ ใน Z-80 นั้นเราสามารถเคลื่อนย้ายข้อมูลได้อย่างมีประสิทธิภาพ และเพื่อทำความเข้าใจกับกลุ่มคำสั่งโหลด เราพิจารณาจากตัวอย่างต่าง ๆ ดังนี้

LD E, (IX + 08)

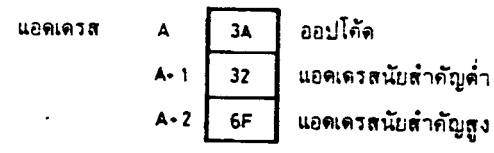
ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ




จากกลุ่มคำสั่งที่กล่าวมาแล้ว จะเห็นว่าเป็นคำสั่งขนาด 3 ไบต์เท่านั้น และจากคำสั่งขนาด 3 ไบต์นี้ เราสามารถใช้วิธีการเพิ่มขนาดของแอดเดรสได้ โดยใช้คำสั่ง

LD A, (6F 32)

ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ




ลักษณะคำสั่งเหล่านี้จะเป็นการโหลดข้อมูลในหน่วยความจำตำแหน่งแอดเดรส 6F 32 สิ่งที่น่าสังเกตคือ 32 จะอยู่ในไบต์ที่ 2 ส่วน 6F จะอยู่ในไบต์ที่ 3

การทำงานในลักษณะนี้ ซีพียูจะนำข้อมูลจากหน่วยความจำที่แอดเดรส HL ไปเก็บไว้ในหน่วยความจำที่แอดเดรส DE และหลังจากนั้นค่าในคู่รีจิสเตอร์ BC, HL และ DE จะเพิ่มค่าขึ้น 1 และจะลดลงไป 1 ในทำนองเดียวกันเราสามารถใช้เพียงคำสั่งเดียวในการเคลื่อนย้ายข้อมูลทั้งบล็อกได้ โดยการนำเอาจำนวนข้อมูลไปไว้ในคู่รีจิสเตอร์ BC แล้วใช้คำสั่ง LDIR การทำงานของคำสั่งนี้จะคล้ายกับคำสั่ง LDI แต่จะทำซ้ำ ๆ ไปเรื่อย ๆ จนกระทั่งค่าในคู่รีจิสเตอร์ BC เป็น 0 ก็จะหยุดและไปกระทำคำสั่งถัดไป

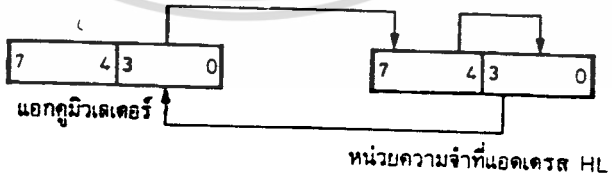
**1.7.3 กลุ่มคำสั่งการกระทำทางคณิตศาสตร์และลอจิก** คำสั่งนี้จะกระทำด้วยรีจิสเตอร์ A เป็นส่วนใหญ่ ลักษณะของกลุ่มคำสั่งนี้ประกอบด้วยลักษณะการ ADD, SUB, ADC, SBC, INC, DEC, AND, OR, XOR ในการอ้างแอดเดรสของตัวโอเปอร์แรนด์นั้น เราทำได้หลายแบบซึ่งขึ้นอยู่กับผู้เขียนและผู้ใช้งาน

การกระทำในกลุ่มคำสั่งนี้จะให้ผลลัพธ์เก็บซ้ำที่ในรีจิสเตอร์ A และนอกจากนี้แล้วค่าของแฟล็กจะมีผลต่อการกระทำของคำสั่งด้วย เช่น ถ้าค่าในรีจิสเตอร์ A เป็น 0 แฟล็กตัวศูนย์ (Z flag) ก็จะได้รับการเซตค่าไว้

**1.7.4 กลุ่มคำสั่งในการเลื่อนข้อมูลเป็นวง** (rotate) **และการชิฟต์** (shift) ความสามารถพื้นฐานของ Z-80 ในการเลื่อนข้อมูลและการชิฟต์ก็เหมือนกับในซีพียูของคอมพิวเตอร์ทั่วไป แต่ใน Z-80 มีคำสั่งที่เกี่ยวกับการเลื่อนข้อมูลตัวเลข BCD อยู่ 2 คำสั่งคือ คำสั่ง RRD และคำสั่ง RLD ทั้ง 2 คำสั่งนี้จะทำให้ข้อมูล 1 (ตัวเลข BCD) ในแอกคูมิวเลเตอร์เลื่อนเป็นวงรวมกับข้อมูล 2 (ตัวเลข BCD) ในหน่วยความจำที่อ้างแอดเดรสโดยคู่รีจิสเตอร์ HL ลักษณะการกระทำจะเป็นดังนี้

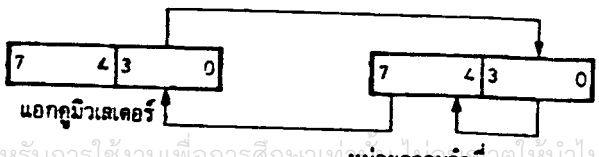
1. **คำสั่ง RRD**

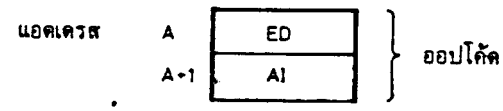
ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ




2. **คำสั่ง RLD**

ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ



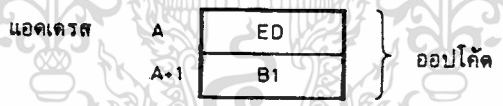



ลักษณะการทำงานจะเป็นดังนี้ คือ ซีพียูจะนำข้อมูลจากหน่วยความจำที่มีค่าแอดเดรสอยู่ในคู่รีจิสเตอร์ HL มาเปรียบเทียบกับรีจิสเตอร์ A หลังจากนี้ค่าของคู่รีจิสเตอร์ HL จะเพิ่มค่าอีก 1 และค่าของคู่รีจิสเตอร์ BC จะลดค่าตัวเองลงไปทีละ 1 ลักษณะการเปรียบเทียบจะให้ผลลัพธ์ที่แฟล็ก จากคำสั่งนี้เราจะให้การเซตค่าของคู่รีจิสเตอร์ BC เป็นจำนวนไบต์ที่ต้องการเปรียบเทียบ ค่าของคู่รีจิสเตอร์ BC จะลดลงมาจนเป็น 0 แล้วเราทดสอบค่าของคู่รีจิสเตอร์ BC ได้เท่ากับ 0 จะเห็นว่าทุกครั้งที่ทำคำสั่งนี้ ค่าของคู่รีจิสเตอร์ HL จะเพิ่มค่า ดังนั้นถ้าให้กระทำคำสั่งนี้ใหม่แอดเดรสในหน่วยความจำจะเพิ่มค่าครั้งละ 1 เสมอ

เพื่อเพิ่มประสิทธิภาพของ Z-80 เรามีคำสั่งที่สามารถให้ซีพียูกระทำคำสั่งจนครบตามจำนวนไบต์ที่วางไว้ เช่น คำสั่ง

CPIR (compare with increment and repeat)

ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ



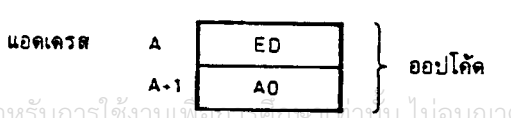

การทำคำสั่งนี้จะเหมือนกับคำสั่ง CPI แต่ซีพียูจะเพิ่มการทดสอบค่าในคู่รีจิสเตอร์ BC ว่าเป็น 0 แล้วหรือยัง ถ้ายังก็จะกระทำคำสั่งเดิมนี้ซ้ำอีก ซึ่งค่าในคู่รีจิสเตอร์ BC จะลดลงมาทีละ 1 จนเป็น 0 จึงหยุดกระทำ นอกจากนี้รายังสามารถให้ค่าแอดเดรสในคู่รีจิสเตอร์ HL ลดค่าลงทีละ 1 ได้เช่นกัน โดยใช้คำสั่ง CPD หรือคำสั่ง CPDR

สำหรับการเคลื่อนย้ายข้อมูลเป็นบล็อก (block) ก็กระทำเช่นเดียวกับการเปรียบเทียบ โดยใช้คู่รีจิสเตอร์ HL เป็นตัวบอกแอดเดรสของหน่วยความจำที่ต้องการย้าย และคู่รีจิสเตอร์ DE เป็นตัวบอกว่าค่าแอดเดรสที่จะนำไปเก็บอยู่ที่ใด ตัวอย่างของคำสั่งคือ

LDI (load location (DE) with location (HL) increment)

LDI (DE). (HL)

ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ



**ตารางที่ 1.1 (ต่อ)** แสดงลักษณะของรหัสแทนเงื่อนไข

| cc | เงื่อนไข | ดูจากแฟล็ก |
|---|---|---|
| 101 | พาริตี้เป็นคู่ | P |
| 110 | มีเครื่องหมายบวก | S |
| 111 | มีเครื่องหมายลบ | S |

ในทำนองเดียวกัน Z-80 ก็มีคำสั่งที่ใช้ RET แบบมีเงื่อนไข เช่น

RET cc

เมื่อ cc เป็นจริงจะกระทำการกระโดดกลับโปรแกรมหลัก แต่ถ้า cc ไม่เป็นจริงก็จะกระทำคำสั่งถัดไป

นอกจากการเรียกโปรแกรมย่อยแล้ว Z-80 ยังมีคำสั่งในลักษณะ 8080 แบบต่าง ๆ อีก ซึ่งอาจเป็นได้ทั้งแบบมีเงื่อนไขหรือไม่มีเงื่อนไข

**1.7.7 กลุ่มคำสั่งเกี่ยวกับการติดต่อระหว่างอินพุตเอาต์พุตกับซีพียู** ใน 8080 กลุ่มคำสั่งนี้มีเพียง 2 คำสั่งเท่านั้นคือ IN และ OUT แต่ใน Z-80 ได้เพิ่มขยายการติดต่อกับอินพุตเอาต์พุตมากยิ่งขึ้น ทำให้ประสิทธิภาพของการติดต่อเป็นไปได้ดียิ่งขึ้น

คำสั่งที่ซ้ำกับ 8080 คือ

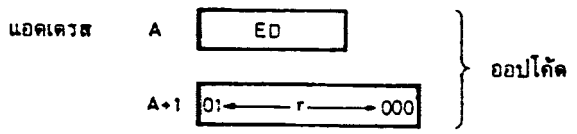
IN A, PORT

ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ


การกระทำคำสั่งนี้ จะเป็นการนำข้อมูลจากพอร์ตที่เป็นอินพุตโหลดข้อมูลมาที่รีจิสเตอร์ A สิ่งที่เพิ่มมาของ Z-80 คือ ความสามารถในการเลือกโหลดให้กับข้อมูลที่รีจิสเตอร์ใดก็ได้ โดยใช้คำสั่ง

IN r, (C)

ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ



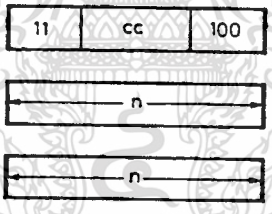
**1.7.5 กลุ่มคำสั่งเกี่ยวกับการกระทำภายในข้อมูลบิตต่าง ๆ** Z-80 มีความสามารถพิเศษในการเซต รีเซต หรือทดสอบบิตใดบิตหนึ่งในรีจิสเตอร์ต่าง ๆ และในหน่วยความจำได้ รายละเอียดของการเซต การรีเซต และการทดสอบสามารถแยกเป็นคำสั่งย่อย ๆ ในกลุ่มนี้ได้มากมาย เพราะแต่ละคำสั่งจะเท่ากับมีคำสั่งย่อย ๆ ได้ถึง 8 คำสั่ง

ในการทดสอบบิตต่าง ๆ ว่าเป็น 0 หรือ 1 นั้น เราใช้แฟล็ก 0 เป็นตัวเก็บข้อมูล เช่น ถ้าบิตที่ทดสอบเป็น 0 ก็จะเซตแฟล็กศูนย์ให้มีค่าเป็น 1

**1.7.6 กลุ่มคำสั่งที่อ้างกับโปรแกรมย่อย** เพื่อให้การเรียกโปรแกรมย่อยมีประสิทธิภาพสูง Z-80 ใช้วิธีการเรียกโดยคำสั่ง CALL และคำสั่ง RET คำสั่ง CALL จะกระทำให้ค่าของโปรแกรมเคาน์เตอร์เดิมที่มีอยู่ไปเก็บไว้ที่สแตก แล้วโหลดค่าแอดเดรสของโปรแกรมย่อยกลับคืนมาให้โปรแกรมเคาน์เตอร์ และในการกระทำคำสั่ง RET ก็จะกระทำกลับกัน นั่นคือจะเป็นการดึง (pop) ข้อมูลในสแตกมาให้โปรแกรมเคาน์เตอร์ นอกจากนี้ Z-80 ยังมีคำสั่งพิเศษอีกคือ

CALL cc, nn

ลักษณะการจัดวางรูปคำสั่งในหน่วยความจำ คือ




คำสั่งนี้จะเป็นการทดสอบเงื่อนไขใน cc ถ้าเงื่อนไขเป็นจริง การกระทำตามคำสั่ง CALL ก็จะเกิดขึ้น คือเครื่องจะเลื่อน (push) ข้อมูลในโปรแกรมเคาน์เตอร์ไปไว้ในสแตก แล้วนำค่า nn ไปไว้ในโปรแกรมเคาน์เตอร์ ถ้าเงื่อนไขใน cc ไม่เป็นจริง ซีพียูจะกระทำคำสั่งถัดไป การใช้รหัสแทนเงื่อนไขขนาด 3 บิต จะสามารถแทนเงื่อนไขได้ 8 แบบ ดังแสดงในตารางที่ 1.1

**ตารางที่ 1.1** แสดงลักษณะของรหัสแทนเงื่อนไข

| cc | เงื่อนไข | ดูจากแฟล็ก |
|---|---|---|
| 000 | ไม่เป็นศูนย์ | Z |
| 001 | เป็นศูนย์ | Z |
| 010 | ไม่มีตัวทด | C |
| 011 | มีตัวทด | C |
| 100 | พาริตี้เป็นคี่ | P′ |

พิจารณาการทำงานของคำสั่งเกี่ยวกับการอินเตอร์รัพต์ต่าง ๆ ดังต่อไปนี้


(ก) คำสั่ง EI (ข) คำสั่ง DI

ความหมายของคำสั่งในรูป (ก) นี้จะกระทำการรีเซตค่า IFF1 และ IFF2 ให้เป็น "0" ทั้งคู่ เพื่อแสดงลักษณะของการดิสเอเบิล ดังนั้นการอินเตอร์รัพต์แบบมาสเคเบิลไม่ว่าแบบใดจะไม่สามารถรับผลตอบสนองได้

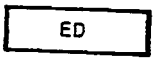
ความหมายของคำสั่งในรูป (ข) เป็นคำสั่งไบต์เดียว ทำให้ IFF1 และ IFF2 เป็น "1" เสมอ แต่การกระทำคำสั่งนี้เพียงคำสั่งเดียวไม่สามารถอีนาเบิลได้ การอินเตอร์รัพต์จะเกิดขึ้นได้เมื่อซีพียูกระทำคำสั่งถัดไปให้เสร็จก่อน เพราะในคำสั่งที่ตามหลังอาจใช้เป็นคำสั่ง RETI ได้ ซึ่งกรณีนี้จะต้องกระทำคำสั่ง RETI ให้เสร็จก่อน

ผลตอบสนองต่อการอินเตอร์รัพต์แบบต่าง ๆ มีรายละเอียดดังนี้

**1.8.1 แบบนอนมาสเคเบิลอินเตอร์รัพต์** ถ้าสัญญาณ $\overline{\text{NMI}}$ เข้ามาจะมีการตอบสนองทันที โดยจะกระโดดไปกระทำยังตำแหน่งแอดเดรส $0066_{16}$ ทันที และจะเก็บค่าโปรแกรมเคาน์เตอร์เดิมไว้บนสแตก ซึ่งถ้าต้องการให้กลับมาทำโปรแกรมหลักเมื่อเสร็จการอินเตอร์รัพต์แล้ว ผู้โปรแกรมสามารถใช้คำสั่ง RETN (return from nonmaskable interrupt) ค่าในสแต็กจะได้รับการดึงมาไว้ในโปรแกรมเคาน์เตอร์เพื่อกระทำกลับไปยังโปรแกรมหลัก

**1.8.2 แบบมาสเคเบิลอินเตอร์รัพต์** การอินเตอร์รัพต์แบบนี้แบ่งออกได้ 3 แบบตามวิธีทางซอฟต์แวร์ ซึ่งผู้โปรแกรมเป็นผู้กำหนด นั่นคือ เมื่อสัญญาณ $\overline{\text{INT}}$ เข้ามา ซีพียูจะตรวจดูว่าระบบซอฟต์แวร์ของผู้โปรแกรมให้ทำการตอบสนองต่อสัญญาณ $\overline{\text{INT}}$ นี้แบบใด ซึ่งถ้าเป็นแบบของซอฟต์แวร์ เราสามารถเซตได้ด้วยคำสั่งของการอินเตอร์รัพต์โหมดต่าง ๆ ดังนี้

**1. การอินเตอร์รัพต์โหมด 0** ลักษณะของคำสั่งที่ใช้ในการเซตคือ IM0 รูปแบบของคำสั่งในหน่วยความจำมีดังนี้



ลักษณะของการอินเตอร์รัพต์ในโหมดนี้จะเหมือนกับของ 8080 นั่นเอง คือเมื่อซีพียูอยู่ในโหมดนี้ ถ้าเพอริเฟอรัลต้องการอินเตอร์รัพต์และส่งสัญญาณ $\overline{INT}$ มา การตอบสนองของซีพียูจะกระทำได้ด้วยการหยุดชั่วคราว แล้วอ่านคำสั่งที่สามารถแทรกเข้ามาในบัสข้อมูลได้ ซีพียูนำคำสั่ง 1 ไบต์ที่ได้จากเพอริเฟอรัลนี้ไปดีความหมายใน IR (instruction register) คำสั่งที่จะแทรกเข้าไปก็คือ คำสั่ง 1 ไบต์ โดยใช้ RST b นั่นเอง ดังนั้นเมื่อซีพียูกระทำคำสั่ง RST b ข้อมูลของโปรแกรมเคาน์เตอร์เดิมจะได้รับการเก็บไว้ในสแตก โดยที่โปรแกรมเคาน์เตอร์ใหม่จะเก็บค่า b ที่มาจากคำสั่ง RST แล้วกระทำคำสั่งถัดไปตามโปรแกรมเคาน์เตอร์

2. **การอินเตอร์รัพต์โหมด 1** การอินเตอร์รัพต์แบบนี้จะไม่มีใน 8080 คำสั่งที่ใช้ในการเซตคือ IM1 ซึ่งมีรูปแบบดังนี้

| ED |
|---|
| 56 |

ในโหมดนี้ ซีพียูจะให้ผลตอบสนองต่อสัญญาณอินเตอร์รัพต์ โดยทำการเอ็กซิคิวต์คำสั่ง RST 0038 โดยอัตโนมัติ นั่นคือ ถ้าซีพียูอยู่ในโหมดนี้และมีการอินเตอร์รัพต์เกิดขึ้น ซีพียูจะกระโดดไปกระทำที่ตำแหน่ง 0038 โดยอัตโนมัติ

3. **การอินเตอร์รัพต์โหมด 2** การอินเตอร์รัพต์แบบนี้จะได้รับการเซตโดยคำสั่ง IM2 ซึ่งมีรูปแบบดังนี้

| ED |
|---|
| 5E |

การอินเตอร์รัพต์โหมดนี้จะเป็นแบบที่มีประสิทธิภาพสูงสุด เพราะจะทำให้ซีพียูสามารถกระโดดไปบริเวณส่วนใดของโปรแกรมก็ได้ การกระทำในกรณีนี้จะใช้วิธีการเรียกแอดเดรสโดยทางอ้อมไปยังหน่วยความจำที่ใด ๆ โดยใช้ข้อมูลขนาด 8 บิตที่ได้มาจากเพอริเฟอรัล ซึ่งจะเป็นส่วนของบิตที่เป็นนัยสำคัญต่ำรวมกับรีจิสเตอร์ I ที่เป็นบิตที่มีนัยสำคัญสูง โดยความจริงแล้วบิตนัยสำคัญต่ำสุดคือ $A_0$ จะต้องเป็น 0 และจากการรวมของรีจิสเตอร์ I กับ 7 บิต รวมบิต 0 ที่มีค่าเป็น 0 จะชี้ไปยังแอดเดรสหนึ่งในหน่วยความจำ แล้วนำเอาค่าในหน่วยความจำนั้น 2 ไบต์มาเรียงกันเป็นแอดเดรสของการกระโดดไปทำงาน

# สรุปคำสั่งของ Z80 ที่เกี่ยวข้อง

| MNEMONIC | OP CODE | COMMENT | FLAGS S | Z | H | P/V | N | C | M CYCLES | T STATES |
|---|---|---|---|---|---|---|---|---|---|---|
| INIR | EDB2 | Input data from input port pointed to by C into memory pointed to by HL. Decrement B. Increment HL. Repeat B = 0 | X | 1 | X | X | 1 | X | 5 (if B ≠ 0) | 12 |
| | | | | | | | | | 4 (if B = 0) | 16 |
| JP (HL) | E9 | Jump to the address pointed to by HL | . | . | . | . | . | . | 1 | 4 |
| JP (IX) | DDE9 | Jump to the address pointed to by IX | . | . | . | . | . | . | 2 | 8 |
| JP (IY) | FDE9 | Jump to the address pointed to by IY | . | . | . | . | . | . | 2 | 8 |
| JP C,nn | DAnn | If C = 0, disregard instruction. If C = 1, jump to address pointed to by bytes 3 and 2 | . | . | . | . | . | . | 3 | 10 |
| JP M,nn | FAnn | If sign flag = 0, disregard instruction. If sign flag = 1, jump to address pointed to by bytes 3 and 2 | . | . | . | . | . | . | 3 | 10 |
| JP NC,nn | D2nn | If carry flag = 1, disregard instruction. If carry flag = 0, jump to address pointed to by bytes 3 and 2 | . | . | . | . | . | . | 3 | 10 |
| JP nn | C3nn | Unconditional jump to address pointed to by bytes 3 and 2 | . | . | . | . | . | . | 3 | 10 |
| JP NZ,nn | C2nn | If zero flag = 1 disregard instruction. If zero flag = 0, jump to address pointed to bytes 3 and 2 | . | . | . | . | . | . | 3 | 10 |
| JP P,nn | F2nn | If sign flag = 1, disregard instruction. If sign flag = 0, jump to address pointed to by bytes 3 and 2 | . | . | . | . | . | . | 3 | 10 |
| JP PE,nn | EAnn | If parity flag = 0, disregard instruction. If parity flag = 1, jump to address pointed to by bytes 3 and 2 | . | . | . | . | . | . | 3 | 10 |
| JP PO,nn | E2nn | If parity flag = 1, disregard instruction. If parity flag = 0, jump to address pointed to by bytes 3 and 2 | . | . | . | . | . | . | 3 | 10 |
| JP Z,nn | CAnn | If zero flag = 0, then disregard instruction. If zero flag = 1, then jump to address pointed to by bytes 3 and 2 | . | . | . | . | . | . | 3 | 10 |
| JR C,e | 38e | If carry flag = 0, disregard instruction. If carry flag = 1, jump to PC + e | . | . | . | . | . | . | 2 (if C = 0) | 7 |
| | | | | | | | | | 3 (if C = 1) | 12 |
| JR e | 18e | Unconditional relative jump to PC + e | . | . | . | . | . | . | 3 | 12 |
| JR NC,e | 30e | If carry flag = 1, disregard jump. If C = 0, jump to PC + e | . | . | . | . | . | . | 2 (if C = 1) | 7 |
| | | | | | | | | | 3 (if C = 0) | 12 |
| JR NZ,e | 20e | If Z = 1, disregard instructions. If Z = 0, jump to PC + e | . | . | . | . | . | . | 2 (if Z = 1) | 7 |
| | | | | | | | | | 3 (if Z = 0) | 12 |

| MNEMONIC | OP CODE | COMMENT | FLAGS S | Z | H | P/V | N | C | M CYCLES | T STATES |
|---|---|---|---|---|---|---|---|---|---|---|
| LD (IY + d),D | FD72d | Load contents of register D into memory pointed to by IY + d | . | . | . | . | . | . | 5 | 19 |
| LD (IY + d),E | FD73d | Load contents of register E into memory pointed to by IY + d | . | . | . | . | . | . | 5 | 19 |
| LD (IY + d),H | FD74d | Load contents of register H into memory pointed to by IY + d | . | . | . | . | . | . | 5 | 19 |
| LD (IY + d),L | FD75d | Load contents of register L into memory pointed to by IY + d | . | . | . | . | . | . | 5 | 19 |
| LD (IY + d),n | FD36dn | Load data in byte 4 into memory pointed to by IY + d | . | . | . | . | . | . | 5 | 19 |
| LD (nn),A | 32nn | Load contents of A into memory pointed to by bytes 3 and 2 | . | . | . | . | . | . | 4 | 13 |
| LD (nn),BC | ED43nn | Load contents of C into memory pointed to by bytes 4 and 3. Load B into the following address | . | . | . | . | . | . | 6 | 20 |
| LD (nn),DE | ED53nn | Load contents of E into memory pointed to by bytes 4 and 3. Load D into following address | . | . | . | . | . | . | 6 | 20 |
| LD (nn),HL | 22nn | Load L into memory pointed to by bytes 3 and 2. Load H into the next higher address | . | . | . | . | . | . | 5 | 16 |
| LD (nn),IX | DD22nn | Load the LOW byte of the IX register into memory pointed to by bytes 4 and 3. Load the HIGH byte of IX into the next higher address | . | . | . | . | . | . | 6 | 20 |
| LD (nn),IY | FD22nn | Load the LOW byte of the IY register into memory pointed to by bytes 4 and 3. Load the HIGH byte of IY into the next higher address | . | . | . | . | . | . | 6 | 20 |
| LD (nn),SP | ED73nn | Load the LOW byte of the stack pointer into the address pointed to by bytes 4 and 3. Load the HIGH byte of SP into the next higher address | . | . | . | . | . | . | 6 | 20 |
| LD A,(BC) | 0A | Load contents of memory pointed to by BC into the accumulator | . | . | . | . | . | . | 2 | 7 |
| LD A,(DE) | 1A | Load contents of memory pointed to by DE into the accumulator | . | . | . | . | . | . | 2 | 7 |
| LD A,(HL) | 7E | Load contents of memory pointed to by HL into the accumulator | . | . | . | . | . | . | 2 | 7 |
| LD A,(IX + d) | DD7Ed | Load contents of memory pointed to by IX + d into the accumulator | . | . | . | . | . | . | 5 | 19 |
| LD A,(IY + d) | FD7Ed | Load contents of memory pointed to by IY + d into the accumulator | . | . | . | . | . | . | 5 | 19 |
| LD A,(nn) | 3Ann | Load contents of memory pointed to by bytes 3 and 2 into the accumulator | . | . | . | . | . | . | 4 | 13 |
| LD A,A | 7F | Load contents of accumulator into the accumulator | . | . | . | . | . | . | 1 | 4 |

| MNEMONIC | OP CODE | COMMENT | FLAGS S | Z | H | P/V | N | C | M CYCLES | T STATES |
|---|---|---|---|---|---|---|---|---|---|---|
| LD D,(IX + d) | DD56d | Load contents of memory pointed to by IX + d into D | . | . | . | . | . | . | 5 | 19 |
| LD D,(IY + d) | FD56d | Load contents of memory pointed to by IY + d into D | . | . | . | . | . | . | 5 | 19 |
| LD D,A | 57 | Load contents of accumulator into D | . | . | . | . | . | . | 1 | 4 |
| LD D,B | 50 | Load contents of register B into D | . | . | . | . | . | . | 1 | 4 |
| LD D,C | 51 | Load contents of register C into D | . | . | . | . | . | . | 1 | 4 |
| LD D,D | 52 | Load contents of register D into D | . | . | . | . | . | . | 1 | 4 |
| LD D,E | 53 | Load contents of register E into D | . | . | . | . | . | . | 1 | 4 |
| LD D,H | 54 | Load contents of register H into D | . | . | . | . | . | . | 1 | 4 |
| LD D,L | 55 | Load contents of register L into D | . | . | . | . | . | . | 1 | 4 |
| LD D,n | 16n | Load data in byte 2 into D | . | . | . | . | . | . | 2 | 7 |
| LD DE,(nn) | ED5Bnn | Load contents of memory pointed to by bytes 4 and 3 into E. Load the next byte in memory into D | . | . | . | . | . | . | 6 | 20 |
| LD DE,nn | 11nn | Load data in byte 2 into register E. Load data in byte 3 into D | . | . | . | . | . | . | 3 | 10 |
| LD E,(HL) | 5E | Load contents of memory pointed to by HL into E | . | . | . | . | . | . | 2 | 7 |
| LD E,(IX + d) | DD5Ed | Load contents of memory pointed to by IX + d into E | . | . | . | . | . | . | 5 | 19 |
| LD E,(IY + d) | FD5Ed | Load contents of memory pointed to by IY + d into E | . | . | . | . | . | . | 5 | 19 |
| LD E,A | 5F | Load contents of accumulator into E | . | . | . | . | . | . | 1 | 4 |
| LD E,B | 58 | Load contents of register B into E | . | . | . | . | . | . | 1 | 4 |
| LD E,C | 59 | Load contents of register C into E | . | . | . | . | . | . | 1 | 4 |
| LD E,D | 5A | Load contents of register D into E | . | . | . | . | . | . | 1 | 4 |
| LD E,E | 5B | Load contents of register E into E | . | . | . | . | . | . | 1 | 4 |
| LD E,H | 5C | Load contents of register H into E | . | . | . | . | . | . | 1 | 4 |
| LD E,L | 5D | Load contents of register L into E | . | . | . | . | . | . | 1 | 4 |
| LD E,n | 1En | Load data in byte 2 into E | . | . | . | . | . | . | 2 | 7 |
| LD H,(HL) | 66 | Load contents of memory pointed to by HL into H | . | . | . | . | . | . | 2 | 7 |
| LD H,(IX + d) | DD66d | Load contents of memory pointed to IX + d into H | . | . | . | . | | | 5 | 19 |
| LD H,(IY + d) | FD66d | Load contents of memory pointed to IY + d into H | . | . | . | . | | | 5 | 19 |
| LD H,A | 67 | Load contents of accumulator into H | . | . | . | | | | 1 | 4 |
| LD H,B | 60 | Load contents of register B into H | . | . | . | | | | 1 | 4 |
| LD H,C | 61 | Load contents of register C into H | . | . | . | | | | 1 | 4 |
| LD H,D | 62 | Load contents of register D into H | . | . | . | . | | | 1 | 4 |
| LD H,E | 63 | Load contents of register E into H | . | . | . | | | | 1 | 4 |
| LD H,H | 64 | Load contents of register H into H | . | | . | | | | 1 | 4 |
| LD H,L | 65 | Load contents of register L into H | . | | | | | | 1 | 4 |
| LD H,n | 26n | Load data in byte 2 into H | . | . | | | | | 2 | 7 |
| LD HL,(nn) | 2Ann | Load contents of memory pointed to by bytes 3 and 2 into L. Load the following byte into H | | | | | | | 5 | 16 |

| MNEMONIC | OP CODE | COMMENT | FLAGS S | Z | H | P/V | N | C | M CYCLES | T STATES |
|---|---|---|---|---|---|---|---|---|---|---|
| LD HL,nn | 21nn | Load byte 2 into L and load byte 3 into H | . | . | . | . | . | . | 3 | 10 |
| LD I,A | ED47 | Load accumulator into interrupt register | . | . | . | . | . | . | 2 | 9 |
| LD IX,(nn) | DD2Ann | Load contents of memory pointed to by bytes 4 and 3 into the LOW byte of IX. Load the following byte into the HIGH byte of IX | . | . | . | . | . | . | 6 | 20 |
| LD IX,nn | DD21nn | Load byte 3 into LOW byte of IX and byte 4 into the HIGH byte of IX | . | . | . | . | . | . | 4 | 14 |
| LD IY,(nn) | FD2Ann | Load contents of memory pointed to by bytes 4 and 3 into the LOW byte of IY. Load the following byte into the HIGH byte of IY | . | . | . | . | . | . | 6 | 20 |
| LD IY,nn | FD21nn | Load byte 3 into LOW byte of IY and byte 4 into the HIGH byte of IY | . | . | . | . | . | . | 4 | 14 |
| LD L,(HL) | 6E | Load contents of memory pointed to by HL into L | . | . | . | . | . | . | 2 | 7 |
| LD L,(IX+d) | DD6Ed | Load contents of memory pointed to by IX+d into L | . | . | . | . | . | . | 5 | 19 |
| LD L,(IY+d) | FD6Ed | Load contents of memory pointed to by IY+d into L | . | . | . | . | . | . | 5 | 19 |
| LD L,A | 6F | Load contents of accumulator into L | . | . | . | . | . | . | 1 | 4 |
| LD L,B | 68 | Load contents of register B into L | . | . | . | . | . | . | 1 | 4 |
| LD L,C | 69 | Load contents of register C into L | . | . | . | . | . | . | 1 | 4 |
| LD L,D | 6A | Load contents of register D into L | . | . | . | . | . | . | 1 | 4 |
| LD L,E | 6B | Load contents of register E into L | . | . | . | . | . | . | 1 | 4 |
| LD L,H | 6C | Load contents of register H into L | . | . | . | . | . | . | 1 | 4 |
| LD L,L | 6D | Load contents of register L into L | . | . | . | . | . | . | 1 | 4 |
| LD L,n | 2En | Load data in byte 2 into L | . | . | . | . | . | . | 1 | 4 |
| LD R,A | ED4F | Load contents of accumulator into R | . | . | . | . | . | . | 1 | 4 |
| LD SP,(nn) | ED7Bnn | Load contents of memory pointed to by bytes 3 and 2 into the LOW byte of SP and the contents of the next location in memory into the HIGH byte of SP | . | . | . | . | . | . | 6 | 20 |
| LD SP,HL | F9 | Load contents of HL into stack pointer | . | . | . | . | . | . | 1 | 6 |
| LD SP,IX | DDF9 | Load contents of IX into stack pointer | . | . | . | . | . | . | 2 | 10 |
| LD SP,IY | FDF9 | Load contents of IY into stack pointer | . | . | . | . | . | . | 2 | 10 |
| LD SP,nn | 31nn | Load byte 2 into the LOW byte of the stack pointer and byte 3 into the HIGH byte | . | . | . | . | . | . | 3 | 10 |
| LDD | EDA8 | Load contents of memory pointed to by HL into memory pointed to by DE. Decrement HL. Decrement BC. Decrement DE. P/V Flag = 0 if BC becomes 0 as a result of the decrement. Otherwise P/V = 1 | . | . | 0 | ↕ | 0 | . | 4 | 16 |

| MNEMONIC | OP CODE | COMMENT | FLAGS S | Z | H | P/V | N | C | M CYCLES | T STATES |
|---|---|---|---|---|---|---|---|---|---|---|
| OUT (C),B | ED41 | Output contents of register B to output port pointed to by C | . | . | . | . | . | . | 3 | 12 |
| OUT (C),C | ED49 | Output contents of register C to output port pointed to by C | . | . | . | . | . | . | 3 | 12 |
| OUT (C),D | ED51 | Output contents of register D to output port pointed to by C | . | . | . | . | . | . | 3 | 12 |
| OUT (C),E | ED59 | Output contents of register E to output port pointed to by C | . | . | . | . | . | . | 3 | 12 |
| OUT (C),H | ED61 | Output contents of register H to output port pointed to by C | . | . | . | . | . | . | 3 | 12 |
| OUT (C),L | ED69 | Output contents of register L to output port pointed to by C | . | . | . | . | . | . | 3 | 12 |
| OUT (n),A | D3n | Output contents of accumulator to output port pointed to by byte 2 | . | . | . | . | . | . | 3 | 11 |
| OUTD | EDAB | Output contents of memory pointed to by HL to the output port pointed to by register C. Decrement B. Decrement HL. If the result of B-1 is zero, then Z = 1. Otherwise Z = 0 | X | ↕ | X | X | 1 | X | 4 | 16 |
| OUTI | EDA3 | Output contents of memory pointed to by HL to the output port pointed to by register C. Decrement B. Increment HL. If the result of B-1 is zero, then Z = 1. Otherwise Z = 0 | X | ↕ | X | X | 1 | X | 4 | 16 |
| POP AF | F1 | Load contents of memory pointed to by SP into F and contents of memory pointed to by SP + 1 into accumulator | . | . | . | . | . | . | 3 | 10 |
| POP BC | C1 | Load contents of memory pointed to by stack pointer into C and contents of memory pointed to by SP + 1 into B | . | . | . | . | . | . | 3 | 10 |
| POP DE | D1 | Load contents of memory pointed to by SP into E and contents of memory pointed to by SP + 1 into D | . | . | . | . | . | . | 3 | 10 |
| POP HL | E1 | Load contents of memory pointed to by SP into L and contents of memory pointed to by SP + 1 into H | . | . | . | . | . | . | 3 | 10 |
| POP IX | DDE1 | Load contents of memory pointed to by stack pointer into LOW byte of IX and contents of memory pointed to by SP + 1 into HIGH byte of IX | . | . | . | . | . | . | 4 | 14 |
| POP IY | FDE1 | Load contents of memory pointed to by stack pointer into LOW byte of IY and contents of memory pointed to by SP + 1 into HIGH byte of IY | . | . | . | . | . | . | 4 | 14 |
| PUSH AF | F5 | Load contents of accumulator into memory pointed to by SP − 1 and flags register into memory pointed to by SP − 2 | . | . | . | . | . | . | 3 | 11 |

| MNEMONIC | OP CODE | COMMENT | FLAGS S | Z | H | P/V | N | C | M CYCLES | T STATES |
|---|---|---|---|---|---|---|---|---|---|---|
| PUSH BC | C5 | Load B into memory pointed to by SP – 1 and C into memory pointed to by SP – 2 | . | . | . | . | . | . | 3 | 11 |
| PUSH DE | D5 | Load contents of D into memory pointed to by SP – 1 and contents of E into memory pointed to by SP – 2 | . | . | . | . | . | . | 3 | 11 |
| PUSH HL | E5 | Load contents of H into address pointed to by SP – 1 and load L into address pointed to by SP – 2 | . | . | . | . | . | . | 3 | 11 |
| PUSH IX | DDE5 | Load contents of HIGH byte of IX into memory pointed to by SP – 1 and contents of LOW byte into memory pointed to by SP – 2 | . | . | . | . | . | . | 4 | 15 |
| PUSH IY | FDE5 | Load contents of HIGH byte of IY into memory pointed to by SP – 1 and contents of LOW byte into memory pointed to by SP – 2 | . | . | . | . | . | . | 4 | 15 |
| RES 0,(HL) | CB86 | Reset bit zero in memory pointed to by HL to 0 | . | . | . | . | . | . | 4 | 15 |
| RES 0,(IX + d) | DDCBd86 | Reset bit zero in memory pointed to by IX + d to 0 | . | . | . | . | . | . | 6 | 23 |
| RES 0,(IY + d) | FDCBd86 | Reset first bit in memory pointed to by IY + d to 0 | . | . | . | . | . | . | 6 | 23 |
| RES 0,A | CB87 | Reset bit zero of accumulator to 0 | . | . | . | . | . | . | 2 | 8 |
| RES 0,B | CB80 | Reset bit zero of B register to 0 | . | . | . | . | . | . | 2 | 8 |
| RES 0,C | CB81 | Reset bit zero of C register to 0 | . | . | . | . | . | . | 2 | 8 |
| RES 0,D | CB82 | Reset bit zero of D register to 0 | . | . | . | . | . | . | 2 | 8 |
| RES 0,E | CB83 | Reset bit zero of E register to 0 | . | . | . | . | . | . | 2 | 8 |
| RES 0,H | CB84 | Reset bit zero of H register to 0 | . | . | . | . | . | . | 2 | 8 |
| RES 0,L | CB85 | Reset bit zero of L register to 0 | . | . | . | . | . | . | 2 | 8 |
| RES 1,(HL) | CB8E | Reset bit 1 in memory pointed to by HL to 0 | . | . | . | . | . | . | 4 | 15 |
| RES 1,(IX + d) | DDCBd8E | Reset bit 1 in memory pointed to by IX + d to 0 | . | . | . | . | . | . | 6 | 23 |
| RES 1,(IY + d) | FDCBd8E | Reset bit 1 in memory pointed to by IY + d to 0 | . | . | . | . | . | . | 6 | 23 |
| RES 1,A | CB8F | Reset bit 1 in accumulator to 0 | . | . | . | . | . | . | 2 | 8 |
| RES 1,B | CB88 | Reset bit 1 in B register to 0 | . | . | . | . | . | . | 2 | 8 |
| RES 1,C | CB89 | Reset bit 1 in C register to 0 | . | . | . | . | . | . | 2 | 8 |
| RES 1,D | CB8A | Reset bit 1 in D register to 0 | . | . | . | . | . | . | 2 | 8 |
| RES 1,E | CB8B | Reset bit 1 in E register to 0 | . | . | . | . | . | . | 2 | 8 |
| RES 1,H | CB8C | Reset bit 1 in H register to 0 | . | . | . | . | . | . | 2 | 8 |
| RES 1,L | CB8D | Reset bit 1 in L register to 0 | . | . | . | . | . | . | 2 | 8 |
| RES 2,(HL) | CB96 | Reset bit 2 in memory pointed to by HL to 0 | . | . | . | . | . | . | 4 | 15 |
| RES 2,(IX + d) | DDCBd96 | Reset bit 2 in memory pointed to by IX + d to 0 | . | . | . | . | . | . | 6 | 23 |
| RES 2,(IY + d) | FDCBd96 | Reset bit 2 in memory pointed to by IY + d to 0 | . | . | . | . | . | . | 6 | 23 |

# เอกสารอ้างอิง


1. ดร.สมบูรณ์ จงชัยกิจ, รศ.กฤษดา วิศวธีรานนท์, และ เสกสัณห์ วัฒนะโชติ, "In-Circuit Emulator for Z-80 Microprocessor," นิทรรศการวิชาการทางวิศวกรรม ครั้งที่ 9 คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย, หน้า 97-99, 2533.
2. Lewis C. Eggebrecht, "Interfacing to the IBM Personal Computer", Howard W. Sams & Co., Inc., 1983.
3. "International Bussiness Machines Corporation, Technical Reference", IBM Corporation, 1983.
4. "คู่มือเทียบเบอร์ไอซี TTL," ซีเอ็ดยูเคชั่น, พิมพ์ครั้งที่ 6, พ.ศ. 2531.
5. "ICEPET User's Guide," Microtime Computer Inc., June 1986.
6. James Bignnell and Robert Donovan, "Z80 Microprocessor Technology Hardware, Software and Interfacing",Delmar Publishers Inc., 1986.
7. "Intel Microcontroller Handbook", INTDK CORPORATION, 1985.
8. "IC TTL, Databook", Science, Engineering & Education Co., Ltd.
9. รศ. ยืน ภู่วรวรรณ, "ทฤษฎีและการประยุกต์ ไมโครโปรเซสเซอร์ Z80", บริษัท ซีเอ็ดยูเคชั่น จำกัด, พ.ศ. 2532.

# กิตติกรรมประกาศ

ในการทำปริญญานิพนธ์นี้ ผู้จัดทำได้รับความเอื้อเฟื้อจาก อาจารย์ มนัส สังวรศิลป์ อาจารย์ สุรพันธ์ เอื้อไพบูลย์ รุ่นพี่ รุ่นน้อง และเพื่อนๆ ทุกๆคน ในการจัดหาอุปกรณ์ เครื่องมือ แหล่งข้อมูล สถานที่ คำแนะนำ ให้คำปรึกษา และช่วยอำนวยความสะดวกต่างๆ ตลอดจนกำลังใจในระหว่างการทำปริญญานิพนธ์นี้จนสำเร็จลุล่วงด้วยดี

คณะผู้จัดทำขอขอบคุณทุกๆท่านมา ณ ที่นี้ด้วย

คณะผู้จัดทำ