ชุดพัฒนาระบบไมโครโปรเซสเซอร์

MICROPROCESSOR DEVELOPMENT SYSTEM

โดย

นาย พยุงศักดิ์ หลำรอด

นาย อานนท์ ศรีวิไลย

อาจารย์ที่ปรึกษา

รศ. สุเชียร เกียรติสุนทร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมระบบควบคุม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

ปริญญานิพนธ์ปีการศึกษา 2545

ภาควิชา วิศวกรรมระบบควบคุม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ชุดพัฒนาระบบไมโครโปรเซสเซอร์

(Microprocessor Development System)

ผู้จัดทำ

นาย พยุงศักดิ์ หลำรอด 43015318

นาย อานนท์ ศรีวิไลย 43015346

.....อาจารย์ที่ปรึกษา

(รศ.สุเธียร เกียรติสุนทร)

ชุดพัฒนาระบบไมโครโปรเซสเซอร์

MICROPROCESSOR DEVELOPMENT SYSTEM

โดย

นาย พยุงศักดิ์ หลำรอด

นายอานนท์ ศรีวิไลย

อาจารย์ที่ปรึกษา

รศ. สุเธียร เกียรติสุนทร

ปีการศึกษา 2545

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ จะกล่าวถึงการสร้างชุดพัฒนาระบบไมโครโปรเซสเซอร์และการประยุกต์ใช้งานไมโครโปรเซสเซอร์ Z-80 โดยมีการต่อชุดบอร์ดอินพุต/เอาต์พุต อินเตอร์เฟส และเขียนโปรแกรมเพื่อแสดงผลการทำงาน รวมทั้งเขียนโปรแกรมการทำงานพื้นฐาน ซึ่งสามารถพัฒนาโปรแกรมให้ใช้งานได้ในระดับที่สูงขึ้น

Abstract

This project refers to Microprocessor Development System and Microprocessor Z-80. It is I/O Interface Board and program to display operation including basic program that can develop to using in higher operation.

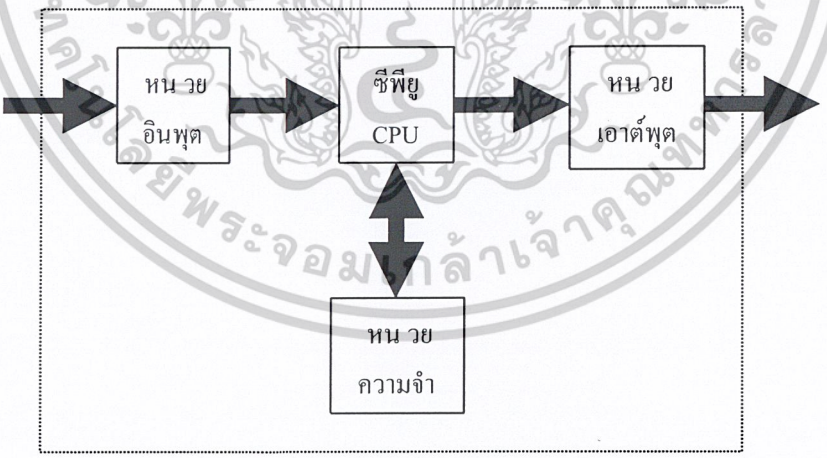
สารบัญ

สารบัญรูปภาพ

บทที่ 1
บทนำ

ในปัจจุบันคอมพิวเตอร์ได้เข้ามามีบทบาทในชีวิตประจำวันของเรามากขึ้นทุกที ถ้าเราได้ศึกษาเกี่ยวกับคอมพิวเตอร์แล้วเราจะพบว่ามีสิ่งหนึ่งซึ่งเป็นหัวใจหลักในเครื่องคอมพิวเตอร์นั่นก็คือ "ไมโครโปรเซสเซอร์" ไมโครโปรเซสเซอร์นี้จะทำหน้าที่ควบคุมการทำงานทั้งหมดของเครื่องคอมพิวเตอร์ เพราะฉะนั้นถ้าเราอยากจะเข้าใจการทำงานของเครื่องคอมพิวเตอร์ เราก็จะต้องทำความเข้าใจเกี่ยวกับการทำงานของไมโครโปรเซสเซอร์ก่อน

ระบบไมโครโปรเซสเซอร์ได้พัฒนาขึ้นอย่างรวดเร็ว ซึ่งเกิดจากการพัฒนาสิ่งประดิษฐ์ที่เป็นสารกึ่งตัวนำก็คือ ทรานซิสเตอร์ ต่อมาได้พัฒนามาเป็น วงจรรวม (Integrated Circuit) หรือ IC ซึ่งสามารถนำไปแทนที่ทรานซิสเตอร์ในวงจรอิเล็กทรอนิกส์ของระบบคอมพิวเตอร์ ผลก็คือทำให้เกิดคอมพิวเตอร์ขนาดเล็กกว่าเดิม และหลังจากนั้นได้มีการพัฒนาโดยใช้เทคโนโลยีของการผลิตวงจรร่วมแบบ Large Scale Integrated Circuit หรือ LSI ทำการรวมเอาวงจรที่ใช้เป็นหน่วยประมวลผลกลาง (Central Processing Unit หรือ CPU) ของคอมพิวเตอร์มาบรรจุอยู่ในแผ่นไอซีเพียงตัวเดียว ซึ่งไอซีนี้เรียกว่า ไมโครโปรเซสเซอร์ (Microprocessor) และได้นำไมโครโปรเซสเซอร์มาต่อร่วมกับ หน่วยความจำ (Memory) หน่วยอินพุตและเอาต์พุต (Input Output Unit) ก็จะทำให้ได้เป็นระบบคอมพิวเตอร์ขึ้นมา ดังแสดงในรูปบล็อกไดอะแกรมของระบบไมโครคอมพิวเตอร์



รูปที่ 1.1 บล็อกไดอะแกรมของระบบไมโครคอมพิวเตอร์

จากรูปที่ 1.1 เห็นได้ว่าโครงสร้างของระบบไมโครคอมพิวเตอร์ก็เหมือนโครงสร้างของคอมพิวเตอร์ทั่วๆ ไป ดพียงแต่หน่วยประมวลผลกลางเป็นไมโครโปรเซสเซอร์ ดังนั้นขนาดหน่วยความจำและจำนวนของหน่วยอินพุต/เอาต์พุตจะมีจำกัด โดยขึ้นอยู่กับชนิดและขนาดของไมโครโปรเซสเซอร์ ดังนั้นไมโครคอมพิวเตอร์จะไม่สามารถไปแทนที่คอมพิวเตอร์ขนาดใหญ่ๆ ได้

บทที่ 2
ทฤษฎี

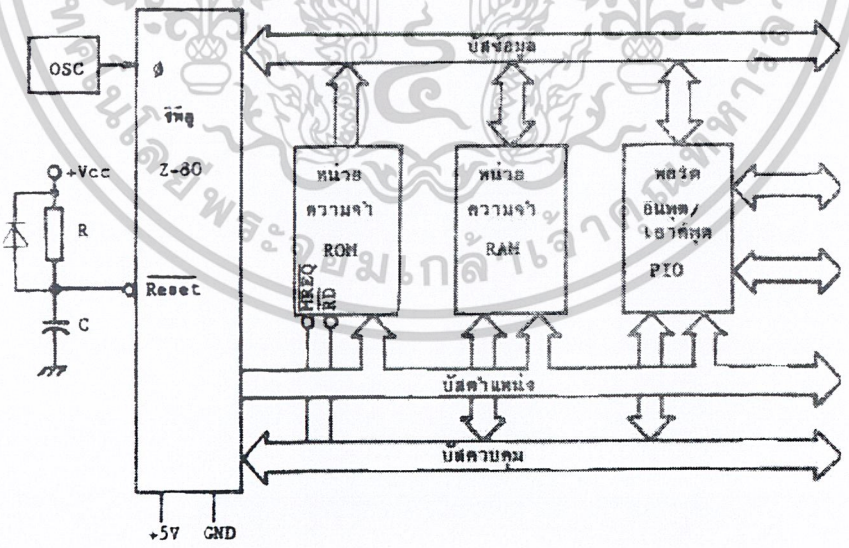
พื้นฐานของไมโครโปรเซสเซอร์

สถาปัตยกรรมของระบบไมโครคอมพิวเตอร์

สถาปัตยกรรมของระบบไมโครคอมพิวเตอร์ แสดงดังรูปที่ 2.1 ในที่นี้หน่วย ไมโครโปรเซสเซอร์(Microprocessor Unit หรือ MPU)ใช้เบอร์ Z-80 ซึ่งมีหน้าที่คือ เป็นหน่วยประมวลผลกลาง (Central Processing Unit หรือ CPU) และไมโครโปรเซสเซอร์นี้ อาจเรียกแทนได้ด้วย ซีพียูภายในไมโครโปรเซสเซอร์จะประกอบด้วยหน่วยพื้นฐานต่างๆ เช่น หน่วยกระทำทางคณิตศาสตร์และลอจิก (Arithmetic Logical Unit หรือ ALU) รีจิสเตอร์ภายในต่างๆ หน่วยควบคุม (control หรือ CU) และวงจรลอจิกต่างๆ อีกมากมาย

ไมโครคอมพิวเตอร์ จะมีทางเดินของสัญญาณต่างๆ ซึ่งเราเรียกเส้นทางเดินของสัญญาณที่ใช้เชื่อมโยงระหว่างหน่วยต่างๆ ของระบบนี้ว่า บัส (Bus) บัสที่มีใช้ในระบบไมโครคอมพิวเตอร์นี้มีด้วยกัน 3 ชนิด คือ

- บัสข้อมูล (Data Bus)
- บัสตำแหน่ง (Address Bus)
- บัสควบคุม (Control Bus)



รูปที่ 2.1 ระบบไมโครคอมพิวเตอร์พื้นฐาน

หน้าที่ของบัสต่างๆ จะอธิบายได้ดังนี้

บัสข้อมูล (Data Bus) เป็นบัส 2 ทิศทาง ที่ใช้เป็นเส้นทางสำหรับนำพาข้อมูลที่เคลื่อนย้ายระหว่าง ส่วนต่างๆ ของระบบ ซึ่งส่วนมากจะเป็นการเคลื่อนย้ายข้อมูล ระหว่างหน่วยความจำกับไมโครโปรเซสเซอร์ หรือระหว่างไมโครโปรเซสเซอร์กับอุปกรณ์ อินพุต / เอาต์พุต

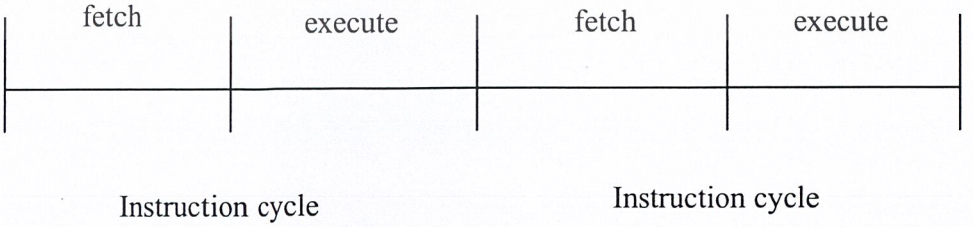
บัสตำแหน่ง (Address Bus) เป็นบัสทิศทางเดียว เป็นเส้นทางที่ใช้สำหรับนำพาสัญญาณเพื่อกำหนดตำแหน่งที่สร้างขึ้นโดยไมโครโปรเซสเซอร์ สัญญาณนี้ใช้ในการกำหนดตำแหน่งของหน่วยความจำเพื่อจะนำข้อมูลออกมาสู่ไมโครโปรเซสเซอร์ หรือนำข้อมูลจากไมโครโปรเซสเซอร์เข้าไปเก็บ หรือใช้ในการกำหนดตำแหน่งของอุปกรณ์ อินพุต / เอาต์พุต

บัสควบคุม (Control Bus) เป็นเส้นทางผ่านของสัญญาณควบคุมต่างๆ ที่ใช้ในระบบ เพื่อให้การทำงานของระบบสอดคล้องกัน

โครงสร้างของโปรเซสเซอร์

หน้าที่หลักอันหนึ่งของโปรเซสเซอร์ที่ใช้เป็นหน่วยประมวลผลกลางของคอมพิวเตอร์ คือ ปฏิบัติ (execute) คำสั่งที่เก็บไว้ในหน่วยความจำหลัก (Main memory) ตามลำดับของคำสั่ง ซึ่งหน่วยความจำนี้โดยปกติจะอยู่นอกโปรเซสเซอร์ ในขั้นแรกโปรเซสเซอร์ต้องอ่านคำสั่ง (fetch) มาจากหน่วยความจำก่อน จากนั้นจะถอดรหัสคำสั่ง และปฏิบัติตามคำสั่งที่อ่านมาได้ ลำดับของการกระทำก่อให้เกิดขบวนการในการทำงานของคำสั่ง เรียกว่า รอบคำสั่ง (instruction cycle) ซึ่งสามารถแบ่งออกเป็น 2 ส่วนใหญ่ๆ ได้คือ ช่วงอ่านคำสั่ง (fetch cycle) และ ช่วงการปฏิบัติคำสั่ง (execution cycle) **fetch** หมายถึง ขบวนการที่โปรเซสเซอร์นำคำสั่งมาจากหน่วยความจำหลัก และถอดรหัสคำสั่ง **execute** หมายถึง ขบวนการที่โปรเซสเซอร์ทำคำสั่งนั้นจนเสร็จ

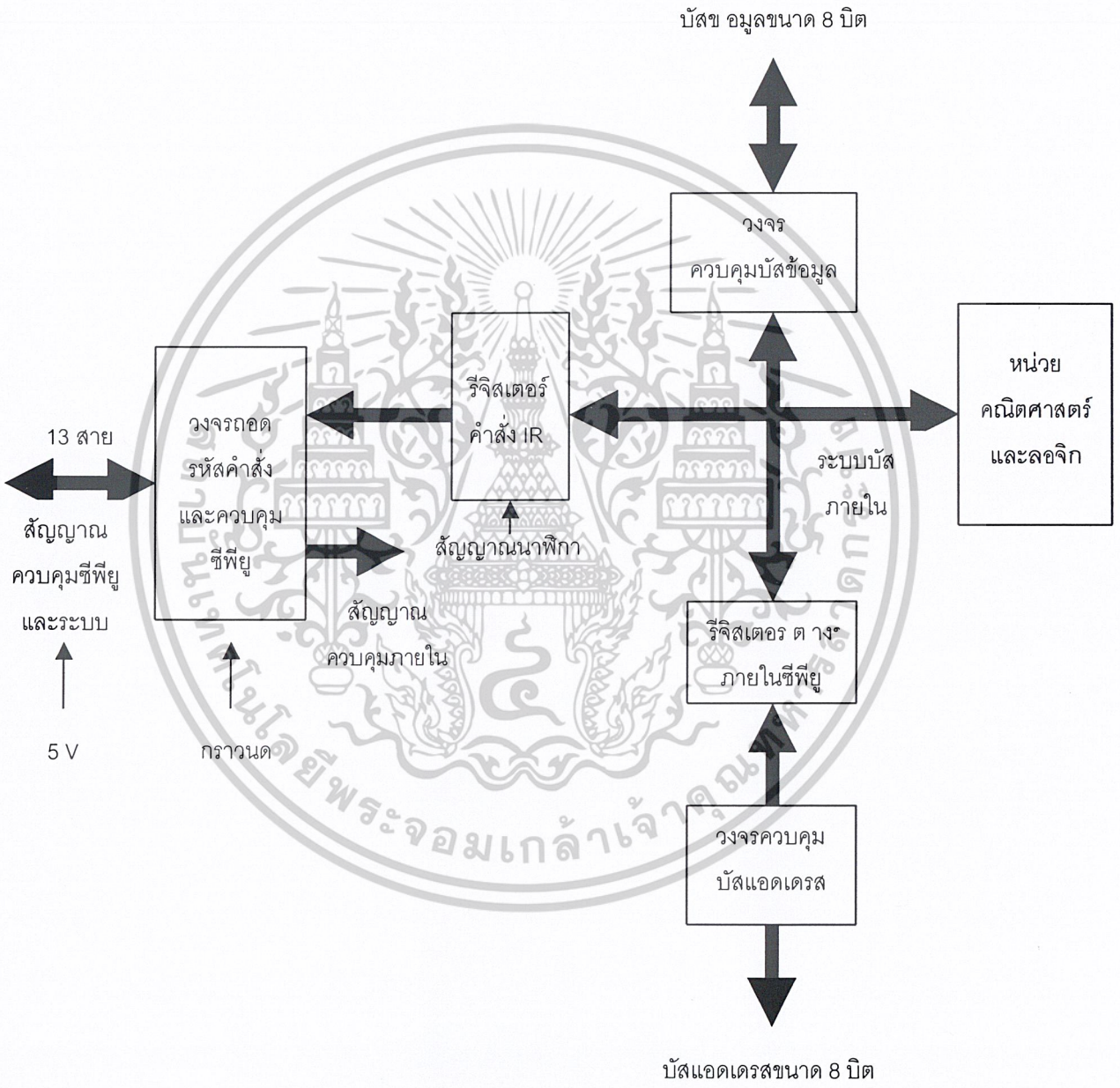
เมื่อทำคำสั่งหนึ่งเสร็จเรียบร้อยแล้ว ก็จะ fetch และ execute คำสั่งถัดไปตามลำดับ ซึ่งอาจอธิบายได้ดังรูปที่ 2.2



รูปที่ 2.2 รอบคำสั่ง Instruction cycle

ลักษณะโครงสร้างของไมโครโปรเซสเซอร์ Z – 80

Z-80 เป็นไมโครโปรเซสเซอร์ที่พัฒนามาจากไมโครโปรเซสเซอร์เดิมคือเบอร์ 8080 ฉะนั้นลักษณะของโครงสร้าง, คำสั่ง, การใช้งานจึงคล้ายกันหากแต่ว่า Z-80 ได้มีความสามารถมากขึ้นหลายอย่าง ทั้งในด้านของคำสั่งและการอินเตอร์รัพต์ เป็นต้น



รูปที่ 2.3 ลักษณะโครงสร้างภายในของซีพียู Z-80

การทำงานอย่างคร่าวๆของตัวซีพียูที่แสดงในรูปที่ 2.3

ส่วนควบคุมซีพียู จะสั่งให้ส่วนวงจรควบคุมบัสแอดเดรสให้สัญญาณการอ้างอิงแอดเดรสออกไปสู่หน่วยความจำภายนอกเพื่อนำคำสั่งที่เก็บในหน่วยความจำนั้นมาเข้าสู่วงจรควบคุมบัสข้อมูลแล้วเข้าสู่ รีจิสเตอร์คำสั่ง IR รอส่งต่อไปถอดรหัสคำสั่งที่วงจรถอดรหัสคำสั่งและควบคุมซีพียู หลังจากถอดรหัสแล้วก็จะมีสัญญาณควบคุมออกมาจากส่วนควบคุมเพื่อทำตามคำสั่งที่ถอดรหัสได้นั้น เช่น หากเป็นการคำนวณส่วนของหน่วยคำนวณก็จะทำการคำนวณโดยใช้หน่วยความจำชั่วคราวคือรีจิสเตอร์ต่างๆ ช่วยคำนวณ แล้วจึงส่งผลที่ได้ออกไปทางบัสข้อมูลต่อไป

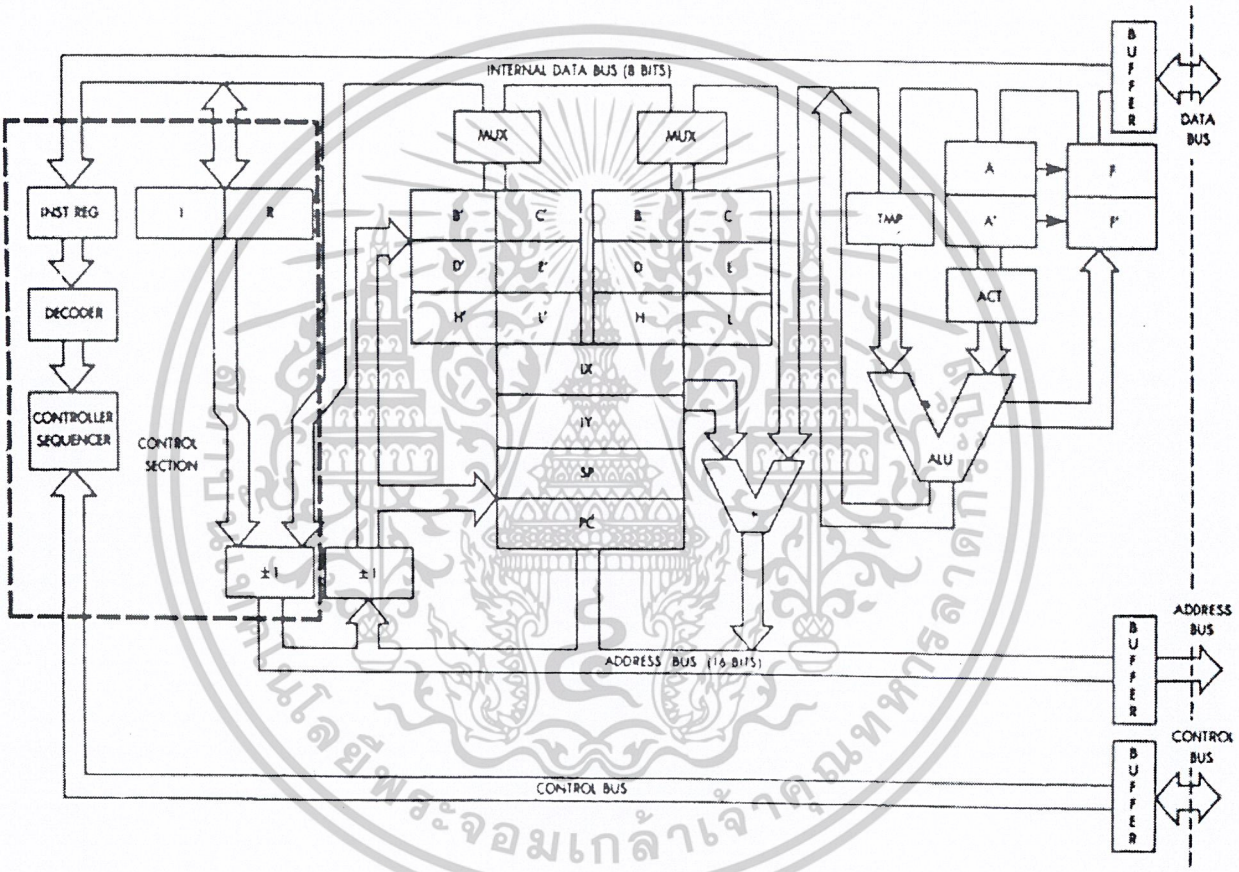
โครงสร้างซีพียู Z-80ที่ละเอียดขึ้นและการทำงานภายในในบางคำสั่ง

โครงสร้างของซีพียู Z-80 แสดงดังรูปที่ 2.4 จากรูปเห็นได้ว่าซีพียูมีบัสอยู่ 3 ชนิด คือ บัสตำแหน่ง บัสข้อมูล และบัสควบคุม ซึ่งบัสเหล่านี้ใช้สำหรับทำการเชื่อมต่อซีพียูกับอุปกรณ์ภายนอก และภายในไมโครโปรเซสเซอร์จะประกอบด้วยวงจรพื้นฐานต่างๆ คือ ALU แอคคิวมูเลเตอร์ รีจิสเตอร์ตำแหน่งและแฟลกบอกจากนี้ยังมีรีจิสเตอร์ ใช้งานต่าง ๆ อีก คือ B,C,E,H,L,B',C',E',H' และ L' ส่วนรีจิสเตอร์ตำแหน่ง ก็คือ PC,SP,IX,IY ซึ่งรีจิสเตอร์ต่าง ๆเหล่านี้ จะได้กล่าวถึงโดยละเอียดต่อไป ส่วนทางด้านซ้ายสุดคือส่วนของหน่วยควบคุม ซึ่งส่วนนี้ที่ถอดรหัสคำสั่งแล้วส่งสัญญาณควบคุมไปตามส่วนต่างๆ ทั้งภายในซีพียูและภายนอกซีพียู รวมทั้งรับสัญญานควบคุมจากภายนอกเข้ามาด้วย สัญาณควบคุมของซีพียู Z-80 ภายนอกมี 13 สัญญาณซึ่งสัญญาน ซึ่ง อาจแบ่งได้ 2 อย่างคือสัญญานควบคุมตัวซีพียูและสัญญาณควบคุมระบบ (CPU and system control) ซึ่งสัญญาณต่าง ๆ จะทำให้มีผลต่อซีพียูและมีผลต่อระบบไมโครคอมพิวเตอร์ด้วย บัสข้อมูลเป็นบัสที่มีขนาด 8 บิต ที่ใช้เป็นทางเดินของข้อมูลระหว่างไมโคโปรเซสเซอร์กับหน่วยความจำ หรืออุปกรณ์อินพุต/เอาต์พุตต่างๆ บัสตำแหน่งเป็นที่มีขนาด16 บิต เพื่อใช้ในการอ้างอิงถึงตำแหน่งของหน่วยความจำ ดังนั้นจะทำให้สามารถอ้างถึงหน่วยความจำภายนอกได้ ตำแหน่ง (65536 ตำแหน่ง หรือ 64k) คือตั้งแต่ตำแหน่งที่ 0-65535 บล็อกที่มีเครื่องหมาย +/- ที่อยู่ทางด้านล่างซ้ายของรีจิสเตอร์หมายถึงการเพิ่มหรือลดข้อมูล (Increment Decrement) ที่มีอยู่ในรีจิสเตอร์ตำแหน่งหรือคู่ของรีจิสเตอร์ต่างๆ คือ SP,PC,BC,DE,HL ซึ่งรีจิสเตอร์ต่างๆ เหล่านี้เป็นรีจิสเตอร์ในการกำหนดตำแหน่งโดยตรง (Pure address register) ของการอ้างถึงตำแหน่งของหน่วยความจำแบบตรง (Direct address mode) ซีพียู Z-80 มีคำสั่งที่เกี่ยวกับอินพุต/เอาต์พุตโดยเฉพาะไม่ใช้ลักษณะการอินพุต/เอาพุตแบบ Memory-mepped (Memory-map I/O ใช้ส่วนหนึ่งของหน่วยความจำเพื่อเป็นตำแหน่งของอุปกรณ์อินพุต/เอาต์พุต) ในการให้ไมโครโปรเซสเซอร์ทำตามหน้าที่ต้องการ ทำได้โดยการเขียนโปรแกรมเก็บไว้ในหน่วยความจำ จากนั้นให้ไมโครโปรเซสเซอร์อ่านคำสั่งมาจากหน่วยความจำเพื่อมาปฏิบัติการปฏิบัติคำสั่งต่างๆ นั้นเราไม่จำเป็นต้องเข้าถึงส่วนรายละเอียดต่างๆ ของไมโครโปรเซสเซอร์ทั้งหมดโดยเราจะสนใจเฉพาะรีจิสเตอร์ต่างๆ ที่เกี่ยวข้องกับ

การเขียนโปรแกรมเท่านั้นรีจิสเตอร์ภายในที่สามารถอ่านหรือเขียนได้มีถึง 208 บิต โดยแยกเป็นกลุ่มของรีจิสเตอร์ขนาด 8 บิต 18 รีจิสเตอร์และรีจิสเตอร์ขนาด 16 บิตอีก 4 รีจิสเตอร์ รีจิสเตอร์ต่างๆ ภายใน Z-80 เป็นลักษณะของสแตติกแรม และรีจิสเตอร์เหล่านั้น แบ่งออกเป็น 3 ประเภทคือ

1. รีจิสเตอร์ใช้งานทั่วไป (General purpose register)
2. แอคิวมูลเลเตอร์และรีจิสเตอร์สถานะ (Accumulator and Flag register)
3. รีจิสเตอร์ใช้งานเฉพาะอย่าง (Special purpos register)

การจัดรีจิสเตอร์ภายในของซีพียู Z-80 แสดงดังรูปที่ 3-2



รูปที่ 2.4 แสดงถึงโครงสร้างภายในของ Z-80 ที่ละเอียดขึ้น

IX, IY Register เรียกว่า "Index Register" ใช้ในการเก็บค่าของแอดเดรสหน่วยความจำหลักที่ใดที่หนึ่งเพื่อเก็บไว้อ้างอิงถึงตำแหน่งของหน่วยความจำที่อื่นได้ ในรูปจะเห็นได้ว่ามีเอาต์พุตที่นำไปบวกเพิ่มกับข้อมูลในบัสข้อมูลได้ นั้นหมายความว่าค่าของ IX, IY ที่ส่งออกไปบนแอดเดรสบัสสามารถที่จะเปลี่ยนแปลงได้แล้วแต่ผู้ใช้จะกำหนดข้อมูลที่นำมาบวก (แต่ค่าในตัวของ IX, IY ไม่เปลี่ยนแปลง) จึงสะดวกในการอ้างแอดเดรสหลายพื้นที่เมื่อใช้ IX, IY เป็นแอดเดรสหลัก (index) ค่าที่นำมาบวกเข้านี้จะเรียกว่า "Displacement"

เครื่องหมาย +, -1 ที่ย้อนจากสายแอดเดรสบัสเข้าสู่ตัวรีจิสเตอร์ต่างๆ นั้นเป็นการบอกว่าค่าของรีจีสเตอร์ต่างๆ นี้เมื่อนำมาใช้กำหนดแอดเดรสแล้วสามารถป้อนกลับมาเพื่อเพิ่มหรือลดค่าของมันได้อีกอันนี้จะมีประโยชน์มากในการทำคำสั่งพวกวนลูป (loop) เช่น การย้ายข้อมูลเป็นกลุ่มเป็นต้น

I register เรียกว่า "Interrupt page address register" ซึ่งใช้ในการเก็บค่าของแอดเดรสไบต์สูงของการอินเตอร์รัพท์ โดยการใช้งานจะนำไปรวมกับค่าของแอดเดรสไบต์ต่ำที่ส่งมาจากอุปกรณ์อินเตอร์รัพท์ให้สังเกตุในรูปว่าจะมีสัญญาณข้อมูลจากบัส มารวม (บล็อก +/-1) ก่อนออกสู่แอดเดรสบัส (ใช้ในโหมด 2 ของการอินเตอร์รัพ Z-80)

R register คือ Memory-refresh register ใช้เก็บค่าของแอดเดรสที่จะถูก รีเฟรช ของหน่วยความจำแบบ ไดนามิก (dynamic ram) ซึ่ง R register จะให้ค่าแอดเดรสออกมาที่เพิ่มค่าโดยอัตโนมัติทุกครั้งที่อยู่ในช่วงแมทชีนไซเกิ้ลของการเฟทคำสั่ง

INST register เป็น รีจีสเตอร์ ที่ใช้เก็บข้อมูลคำสั่งที่ถูกเฟทเข้ามาใช้ช่วงของ แมทชีนเฟทคำสั่งเพื่อรอส่งต่อไปให้กับส่วนวงจรถอดรหัสคำสั่งต่อไป

Decode เป็นวงจรที่ใช้ในการถอดรหัสคำสั่งที่เก็บใน **INST register** จากนั้นส่งสัญญาณให้ส่วนวงจร CONTROL SEQUENCER ทำสัญญาณออกมาควบคุมอุปกรณ์ต่างๆ เพื่อให้ทำงานตามคำสั่งนั้นๆ

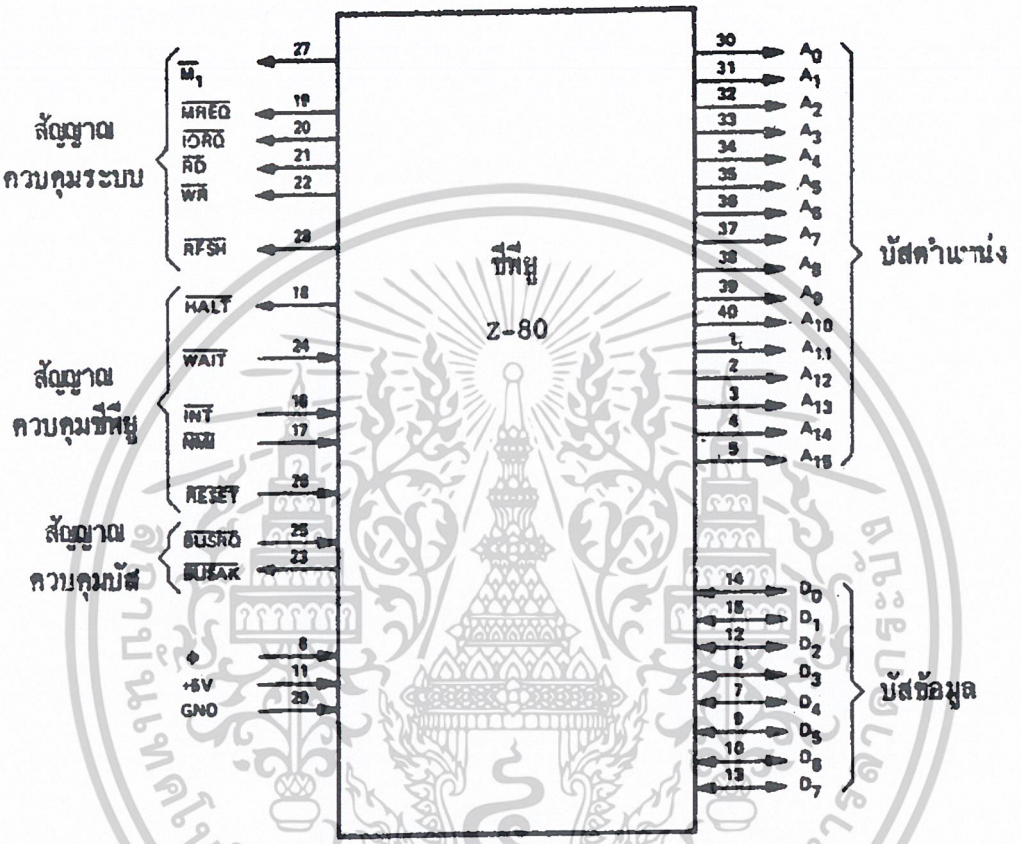
Program counter (PC) เป็นรีจีสเตอร์ 16 บิต ที่ใช้ในการให้ค่าแอดเดรสหน่วยความจำเพื่อทำงานตามลำดับโปรแกรม โดย PC จะเพิ่มค่าตัวเองอัตโนมัติ ดังที่กล่าวมาบ้างแล้วในตอนต้น

Stack pointer (SP) เป็นรีจีสเตอร์ 16 บิต ใช้ในการชี้ค่าแอดเดรสของพื้นที่หน่วยความจำที่กำหนดเป็นพื้นที่ของ Stack ค่าของแอดเดรสสามารถเพิ่มหรือลดเองอัตโนมัติ ซึ่งกล่าวมาแล้วในข้างต้น

จากรูป 2.4 จะเห็นได้ว่า เอ้าพุตจากโครงสร้างภายในออกสู่ภายนอกเป็น ชุดของสัญญาณที่มีด้วยกัน 3 ชุดคือ ชุดของสัญญาณควบคุม, ชุดของสัญญาณแอดเดรสหน่วยความจำ และชุดของสัญญาณข้อมูล ซึ่งจะได้กล่าวถึงรายละเอียดของสัญญาณที่นำออกมาใช้, ตำแหน่งขาสัญญาณต่างๆ ในหัวข้อ 2.2 ต่อไป

ขาและสัญญาณสำหรับการเชื่อมต่อ

ไมโครโปรเซสเซอร์ Z-80 บรรจุอยู่ในไอซีขนาดมาตรฐานอุตสาหกรรม (Industry Standrard) แบบ In-Line Package (DIP) หรือที่เรียกว่าแบบตีนตะขาบ 40 ขา ขาต่างๆแสดงไว้ในรูปที่ 2.5



รูปที่ 2.5 แสดงขาต่างๆ ของซีพียู Z-80

ตัวชิพ ซีพียู เป็นลักษณะตัวแบนสีดำที่มีขนาด 40 ขา ซึ่งแยกออกเป็นกลุ่มต่างๆ คือ บัสข้อมูล ขนาด 8 บิต ซึ่งเป็นบัสชนิดสองทิศทาง คือสามารถวิ่งเข้าออกจากตัวซีพียูได้ และบัสแอดเดรสที่มีขนาด 16 บิต ทำให้สามารถอ้างแอดเดรสได้ถึง 64 Kbytes และบัสแอดเดรสนี้ยังมีส่วนใช้ในการอ้างอิงแอดเดรสของ พอร์ทอินพุตเอาต์พุตด้วย และอีกกลุ่มคือกลุ่มของสัญญาณควบคุมการทำงาน ซึ่งจะมีทั้งหมด 13 สาย ที่เหลือเป็นขาสัญญาณนาฬิกาและแรงจ่ายไฟเลี้ยง

รายละเอียดและหน้าที่ที่สำคัญของสายสัญญาณต่างๆ มีดังนี้

A0-A15 บัสแอดเดรส สัญญาณที่ออกมาจากขาไอซีเหล่านี้จะให้แอคตีฟขณะ high โดยขาเหล่านี้เป็น เอาต์พุตแบบไตรสเตท บัสแอดเดรสมีด้วยกันทั้งหมด 16 สายเพื่อให้ซีพียูติดต่อกับหน่วย ความจำได้มากถึง 216 = 64 K ไบต์นอกจากนี้ส่วนของแอดเดรสยังเป็นตัวกำหนดเบอร์พอร์ตของอุปกรณ์อินพุต-เอาต์พุตโดยขณะที่ซีพียูกระทำคำสั่งเกี่ยวกับอินพุตหรือเอาต์พุต ค่าของแอดเดรสบัสใน 8 บิตล่าง (A0 - A7) จะแสดงค่าเบอร์พอร์ตดังนั้นเราจึงมีอุปกรณ์อินพุตหรือเอาต์พุตได้ ทั้งหมด 28 = 256 พอตท และในขณะช่วงเวลารีเฟรชโดยเมื่อสัญญาณรีเฟรช ปรากฏขึ้นที่ขารีเฟรช (RFSH) ค่าในแอดเดรสบัส A0 - A7จะแสดงค่าแอดเดรสของหน่วยความจำที่จะได้รับการกระทำการรีเฟรช

D0-D7 บัสข้อมูล (data bus) เป็นลักษณะบัสแบบสองทิศทาง Z-80 ซีพียูมีบัสข้อมูล 8 เส้น บัสข้อมูล เป็นเส้นทางผ่านของข้อมูลระหว่าง ซีพียูกับหน่วยความจำซีพียู กับอุปกรณ์อินพุต-เอาต์พุตหรือการติดต่อระหว่างอุปกรณ์อินพุต-เอาต์พุตกับหน่วยความจำ

M1 (Machine Cycle One) ลักษณะเป็นสัญญาณเอาต์พุต โดยส่วนสัญญาณออกมาให้ทราบว่าตอนนี้ทำงานอยู่ในสภาวะเฟทช์ (fetch) (สภาวะการเอาข้อมูลคำสั่งจากหน่วยความจำสู่ตัวซีพียู) โดยแอคตีฟที่โลจิก "0"

$\overline{\textbf{MREQ}}$ (Memory Request) เป็นสายสัญญาณเอาต์พุตแบบสามสถานะ และแอคตีฟที่โลจิก 0 เมื่อสายสัญญาณนี้แอคตีฟ บอกให้ทราบว่า ขณะนี้ไมโครโปรเซสเซอร์ต้องการติดต่อกับหน่วยความจำเพื่ออ่านหรือเขียนข้อมูล โดยที่ตำแหน่งของหน่วยความจำจะปรากฏอยู่บัสตำแหน่งแล้ว

$\overline{\textbf{IORQ}}$ (Input/Output Request) เป็นเอาต์พุตแบบสามสถานะและแอคตีฟที่ลอจิก 0 เมื่อสายสัญญาณนี้แอคตีฟ บอกให้ทราบว่า ขณะนี้ทางด้านไบต์ต่ำ (A0-A7) ของบัสตำแหน่งบรรจุตำแหน่งของพอร์ต ที่จะส่งถ่ายข้อมูลระหว่างไมโครโปรเซสเซอร์กับอุปกรณ์อินพุต/เอาต์พุต

$\overline{\textbf{RD}}$ (Memory Read) เป็นขาสัญญาณเอาต์พุตแบบสามสถานะ และแอคตีฟที่ลอจิก 0 สัญญาณนี้แอคตีฟเพื่อชี้ว่า ขณะนี้ไมโครโปรเซสเซอร์ต้องการอ่านข้อมูลจากหน่วยความจำ หรือจากอุปกรณ์อินพุต/เอาต์พุต

$\overline{\textbf{WR}}$ (Memory Write) เป็นขาสัญญาณเอาต์พุตแบบสามสถานะ และแอคตีฟที่ลอจิก 0 สัญญาณนี้แอคตีฟเพื่อชี้ว่าขณะนี้ไมโครโปรเซสเซอร์ต้องการเขียนข้อมูลจากหน่วยความจำ หรือเข้าอุปกรณ์อินพุต/เอาต์พุต

$\overline{\textbf{RFSH}}$ (Refresh) เป็นขาเอาต์พุตแอคตีฟที่ลอจิก 0 ส่งสัญญาณเพื่อจะบอกว่า ขณะนี้บัสตำแหน่งทางด้านต่ำ 7 บิต (A0-A6) บรรจุตำแหน่งหน่วยความจำแบบไดนามิกแรมที่จะรีเฟรชและสัญญาณ MERQ ในช่วงนี้จะนำไปใช้เป็นสัญญาณสำหรับอ่านเพื่อรีเฟรช (Refresh Read) ไดนามิกแรมทั้งหมดที่ใช้ในระบบ

$\overline{\textbf{HALT}}$ (Halt State) เป็นขาเอาต์พุต แอคตีฟที่ลอจิก 0 เป็นสัญญาณเพื่อชี้ว่าขณะนี้ไมโครโปรเซสเซอร์ปฏิบัติคำสั่ง HALT จากโปรแกรม และกำลังรอสัญญาณการอินเตอร์รัพต์ชนิด

นอนมาสเคเบิลหรือมาสเคเบิล จากอุปกรณ์ภายนอก ถ้าได้รับสัญญาณการอินเตอร์รัพต์แล้วจึงจะทำงานต่อไป

$\overline{\text{WAIT}}$ (wait) เป็นขาอินพุต แอคตีฟที่ที่ลอจิก 0 เป็นสัญญาณเพื่อชี้ว่า การส่งถ่ายข้อมูลระหว่างไมโครโปรเซสเซอร์และหน่วยความจำ หรืออุปกรณ์อินพุต/เอาต์พุต ยังไม่เรียบร้อย และให้ไมโครโปรเซสเซอร์หยุดรอ ตราบเท่าที่ขานี้ยังแอคตีฟอยู่ ดังนั้นสัญญาณนี้จะใช้เพื่อให้หน่วยความจำหรืออุปกรณ์อินพุต/เอาต์พุตที่มีความเร็วใดๆ สามารถทำงานให้เข้าจังหวะกันได้พอดี (Synchronized) กับไมโครโปรเซสเซอร์

$\overline{\text{INT}}$ (Interrupt Request) เป็นขาอินพุต แอคตีฟที่ลอจิก 0 สัญญาณ INT นี้ เป็นสัญญาณที่สร้างมาจากอุปกรณ์อินพุต/เอาต์พุต เพื่อต้องการอินเตอร์รัพต์การทำงานตามปกติของไมโครโปรเซสเซอร์ สัญญาณร้องขอนี้จะถูกตรวจสอบเมื่อถึงสเตทสุดท้ายของคำสั่ง และไมโครโปรเซสเซอร์จะจดจำไว้ ถ้าหากว่าโปรแกรมกำหนดให้มีการยอมรับสัญญาณการอินเตอร์รัพต์ได้ (Enable Interrupt) โดย IFF1 ถูกเซตเป็น 1 และไม่มีการร้องขอใช้บัสเสียก่อน คือขา BUSRQ ต้องไม่แอคตีฟ เมื่อไมโครโปรเซสเซอร์รับสัญญาณอินเตอร์รัพต์ มันจะตอบสนองโดยการส่งสัญญาณ IORQ ออกมาในช่วงเวลา M1 เพื่อเป็นการตอบรับการอินเตอร์รัพต์ (Interrupt Acknowledge) ในช่วงไซเกิลของคำสั่งต่อมา

$\overline{\text{NMI}}$ (Non Maskable Interrupt) เป็นขาอินพุตและแอคตีฟที่ขอบพัลส์ขาลง (Negative edge Trigger) สัญญาณที่ขา NMI นี้มีลำดับความสำคัญสูงกว่าสัญญาณที่ขา INT ไมโครโปรเซสเซอร์ จะทำการตรวจสอบขานี้ที่สเตทสุดท้ายของคำสั่ง

$\overline{\text{RESET}}$ เป็นขาสัญญาณที่รับสัญญาณจากอุปกรณ์ภายนอกที่ต้องการรีเซตตัวซีพียู หรือต้องการให้โปรแกรมเคาน์เตอร์มีค่าเป็น “0”

$\overline{\text{BUSRQ}}$ (Bus Request) เป็นขาสัญญาณอินพุตแอคตีฟที่ระดับ 0 สัญญาณ BUSRQ นี้มีผลทำให้บัสตำแหน่งบัสข้อมูล และสัญญาณควบคุมที่เป็นขาเอาต์พุตแบบสามสถานะ อยู่ในสภาวะอิมพีแดนซ์สูง จากนั้นบัสต่างๆ จะถูกควบคุมโดยอุปกรณ์ภายนอก ไมโครโปรเซสเซอร์จะตรวจสอบสัญญาณการขอใช้บัสนี้ทุกๆ สเตทสุดท้ายของแมทชีนไซเกิลของคำสั่ง และเมื่อพบการขอใช้บัสซีพียูจะตอบสนองในไซเกิลถัดไป

$\overline{\text{BUSAK}}$ (Bus Acknowledge) เป็นขาเอาต์พุตแอคตีฟที่ระดับ 0 สัญญาณนี้ใช้สำหรับตอบรับการขอใช้บัส และแสดงว่าขณะนี้บัสตำแหน่ง บัสข้อมูล และสัญญาณควบคุมที่เป็นเอาต์พุตแบบสามสถานะอยู่ในสภาวะอิมพีแดนซ์สูงแล้ว อุปกรณ์ควบคุมภายนอกสามารถเข้ามาควบคุมบัสได้

$\varnothing$ เป็นขาที่รับสัญญาณนาฬิกาเพื่อเป็นฐานเวลาหลักในการทำงานของระบบ

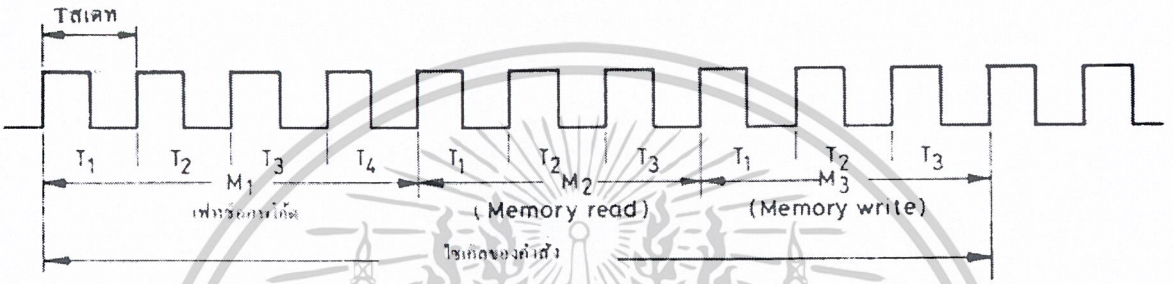
+5 เป็นแรงดันไฟเลี้ยงตัวซีพียู

GND เป็นกราวด์ของตัวซีพียู

ไดอะแกรมเวลาของไมโครโปรเซสเซอร์

ลักษณะของสัญญาณไดอะแกรมเวลาในการทำงานในไซเกิ้ลต่าง

ในการทำงานของไมโครโปรเซสเซอร์ทั้งหลายนั้น จะมีการทำงานอยู่สองจังหวะคือ เฟทซ์และเอ็กซีคิ้ว ซึ่งในสภาวะของการเฟทซ์จะมีเพียง แมทชีนไซเกิ้ลเดียวที่เรียกว่า M1 ส่วนสภาวะของการเอ็กซีคิ้วนั้นจะมีแมทชีนไซเกิ้ลคือ M2, M3, M4... ซึ่งแต่ละคำสั่งก็จะใช้จำนวนของแมทชีนไซเกิ้ลไม่เท่ากัน ยกตัวอย่างดังรูปที่ 2.6



รูปที่ 2.6 ตัวอย่างไดอะแกรมเวลาของไมโครโปรเซสเซอร์

ในแมทชีนไซเกิ้ลของ Z-80 นั้นจะมีอยู่ด้วยกัน 8 แมทชีนไซเกิ้ล

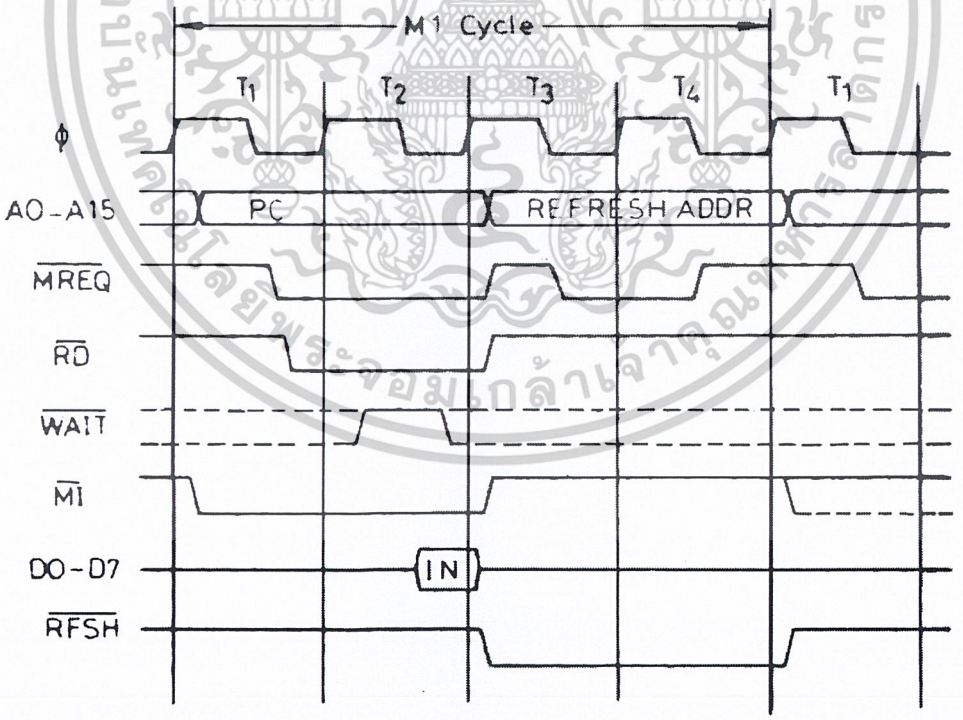
1. ไซเกิ้ลการเฟทซ์ (fecth) รหัสคำสั่ง
2. ไซเกิ้ลการอ่านหรือเขียนหน่วยความจำ
3. ไซเกิ้ลการอ่านหรือเขียนข้อมูลกับอุปกรณ์ อินพุต / เอาต์พุต
4. ไซเกิ้ลการขอใช้บัสหรือการตอบสนองการขอใช้บัส
5. ไซเกิ้ลการขออินเตอร์รัพต์และการตอบรับการอินเตอร์รัพต์
6. ไซเกิ้ลการขอและการตอบรับการอินเตอร์รัพต์แบบนอนมาสเคเบิล
7. ไซเกิ้ลการออกจากคำสั่ง HALT
8. ไซเกิ้ลของการรีเซท (Reset)

การทำงานไซเกิ้ลการเฟทซ์ (fecth) รหัสคำสั่ง

ทุกไซเกิ้ลของคำสั่งแต่ละคำสั่งที่ทำงานจะต้องเริ่มต้นที่ไซเกิ้ลของ M1 ก่อนเสมอ (และในบางคำสั่งก็จะใช้เฉพาะไซเกิ้ลของ M1 ก็จะทำงานเสร็จสิ้นได้เช่นคำสั่ง INC R โดย R : Any registor)

เมื่อเข้าสู่ไซเกิ้ลของการทำงาน M1 ดังแสดงในรูปที่ 2.7 โดยสัญญาณ M1 จะแอคตีฟเป็น "0" ก่อนเป็นระยะนานตลอดช่วงของการเฟทซ์ข้อมูลคำสั่ง (ทั้งยังเป็นการแสดงว่าตอนนี้อยู่ในสภาวะของการเฟทซ์) ค่าของ PC จะเข้าสู่แอดเดรสบัสในช่วงขอบขาลงของ T1 , สัญญาณ MREQ และ RD จะเป็น "0" เป็นตัวชี้บอกการติดต่อเพื่ออ่านข้อมูลกับหน่วยความจำภายนอก ค่าของข้อมูล (คำสั่ง) ที่ได้เข้าสู่ตัวไมโครโปรเซสเซอร์ก็จะถูกแลทช์เข้ามาที่ช่วงขอบขาขึ้นของ T3 เท่านั้น และในช่วงของ T3, T4 ที่เหลือนี้ก็จะเป็นช่วงของการรีเฟรชหน่วยความจำแบบไดนามิก ซึ่งค่าของ PC ตอนนี้จะเป็นค่าของแอดเดรสรีเฟรชที่ได้จะรีจิสเตอร์ R โดยเมื่ออยู่ในช่วงของการรีเฟรชนั้นจะมีการแสดงให้ทราบโดยสัญญาณที่ขารีเฟรชจะมีโลจิก "0" ตลอดช่วงของการรีเฟรชหน่วยความจำ ถึงตรงนี้ก็จะสิ้นสุดสภาวะเฟทซ์หรือ M1

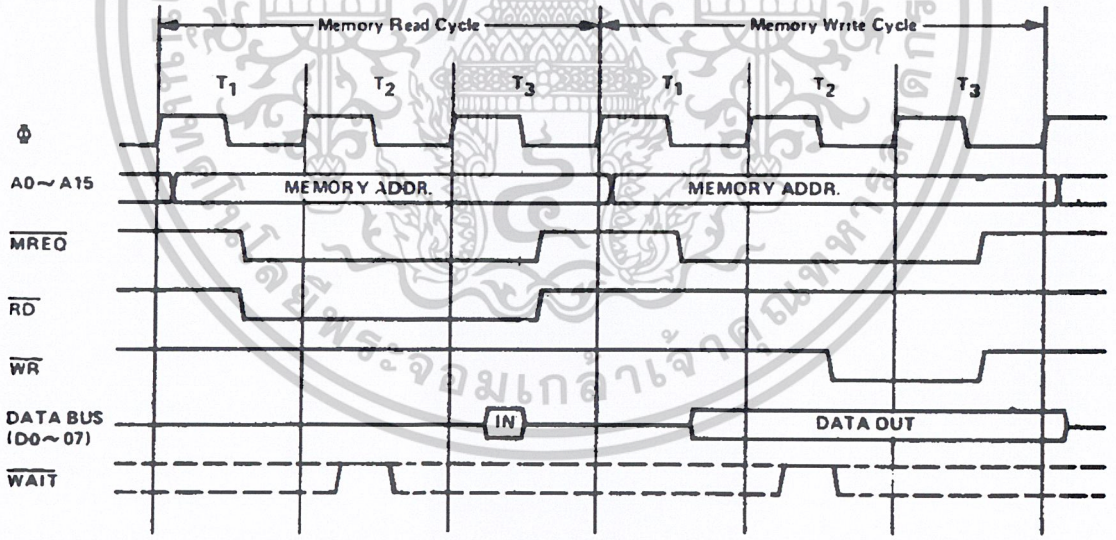
เราสามารถที่จะยืดเวลาของช่วง M1 ออกไปโดยการให้สัญญาณที่ขา WAIT โดยตัวของ CPU จะมีการตรวจสัญญาณ WAIT จากภายนอกในช่วงของขอบขาลงของ T2 ซึ่งหากพบก็จะยืดสัญญาณควบคุมต่างๆ ที่เกิดขึ้นในตอนนั้นออกไปตราบเท่าที่ยังมีสัญญาณ WAIT อยู่ ในรูปช่วงที่จะมีการยืดสัญญาณจะแสดงเป็นเส้นขาดต่อไว้



รูปที่ 2.7 ไซเกิ้ลการเฟทซ์รหัสคำสั่ง

ไซเกิ้ลการอ่านหรือเขียนหน่วยความจำ

ไดอะแกรมของเวลาการอ่านหรือเขียนหน่วยความจำแสดงดังรูปที่ 2.8 ไซเกิ้ลการทำงานทั้งสองนี้ใช้สัญญาณนาฬิกาอย่างละ 3 คาบ ไม่รวมคาบเวลาการรอคอยที่เกิดขึ้นที่ขา WAIT พิจารณาไซเกิ้ลในการอ่าน เห็นว่าขั้นแรก CPU จะส่งตำแหน่งที่ต้องการอ่านลงในบัสตำแหน่ง และหลังจากขอบพัลส์ขาลงของ T1 ก็ส่งสัญญาณ MREQ และ RD ออกมา (เหมือนช่วงการเฟทช์) จากนั้นหน่วยความจำจะส่งข้อมูลในตำแหน่งนั้นลงมาในบัสข้อมูล เมื่อถึงขอบขาลงของ T2 ซีพียู จะตรวจสอบที่ขา WAIT ว่ามีสัญญาณการรอคอยหรือไม่ ถ้ามี ซีพียูจะสร้าง Tw ขึ้นในคาบเวลาต่อไป ถ้าไม่มี ซีพียูจะทำงานต่อไป คือเมื่อถึงช่วงขอบพัลส์ขาลงของ T3 มันจะทำการอ่านข้อมูลจากบัสข้อมูลในระยะเวลาสั้นๆ เข้าไปเพื่อมิให้เกิดความผิดพลาด ส่วนไซเกิ้ลของการเขียนข้อมูลลงหน่วยความจำพิจารณาได้จากรูปที่ 2.8 ซึ่งลักษณะการทำงานจะคล้ายกับช่วงการอ่าน คือ หลังจากส่งตำแหน่งที่ต้องการเขียนลงในบัสตำแหน่งแล้วก็จะส่งสัญญาณ MREQ ออกมา จากนั้นข้อมูลของรีจิสเตอร์ที่ต้องการจะเขียนลงไปในหน่วยความจำจะถูกถ่ายลงในบัสข้อมูลที่ขอบพัลส์ขาลงของ T1 และขอบพัลส์ขาลงของ T2 สัญญาณ WR จะแอคตีฟ ซึ่งทำให้หน่วยความจำภายนอกเขียนข้อมูลลงในบัสข้อมูล เข้าไปในหน่วยความจำตามตำแหน่งที่กำหนดที่บัสตำแหน่งขอบพัลส์ขาลงของ T3 สัญญาณ MREQ และ WR จะหายไป หลังจากนั้นข้อมูลบนบัสข้อมูลก็สามารถเปลี่ยนแปลงต่อไปได้



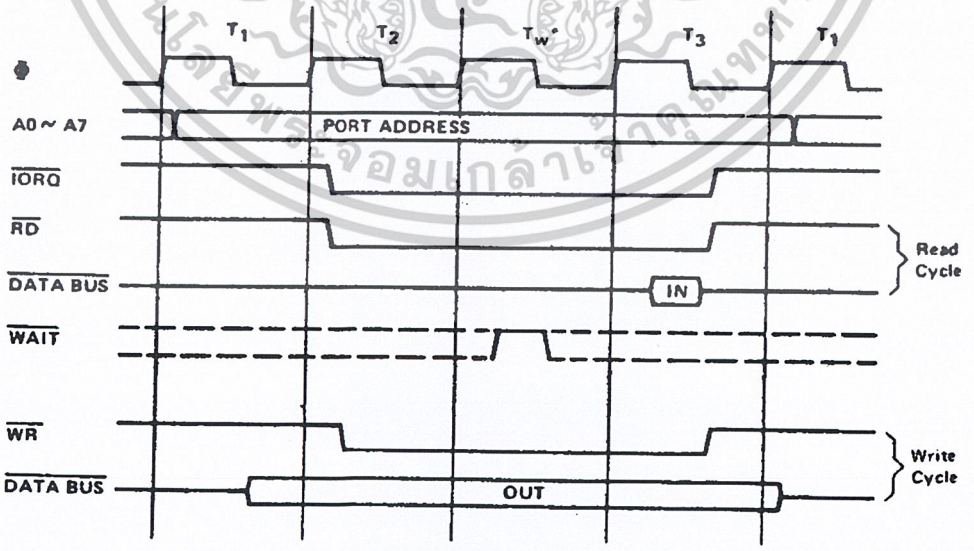
รูปที่ 2.8 ไซเกิ้ลการอ่านหรือเขียนหน่วยความจำ

ไซเกิ้ลการอ่านหรือเขียนข้อมูลกับอุปกรณ์ อินพุต / เอาต์พุต

ไซเกิ้ลการอ่านหรือเขียนข้อมูลกับอุปกรณ์ อินพุต / เอาต์พุตนี้ เกิดขึ้นในขณะที่ซีพียูปฏิบัติคำสั่งเกี่ยวกับการอินพุตหรือเอาต์พุต เช่น IN A,(n) หรือ OUT (n),A เป็นต้น ปกติคำสั่งกลุ่มนี้ใช้ 3 หรือ 4 แมทชีนไซเกิ้ล แต่คำสั่งที่มีความสามารถสูงบางคำสั่ง เช่น INTR INDR OTTR หรือ OTDR ที่สามารถส่งถ่ายข้อมูลได้ถึง 256 ไบต์ จะใช้แมทชีนไซเกิ้ลบางอันซ้ำๆกัน จนกระทั่งส่งถ่ายข้อมูลเสร็จ ผลก็คือเวลาในการทำงานของคำสั่งจะขึ้นอยู่กับจำนวนไบต์ที่จะส่งถ่าย และความเร็วของอุปกรณ์อินพุต/เอาต์พุตด้วย

ไดอะแกรมเวลาของการอ่านข้อมูลจากอุปกรณ์อินพุตและเขียนข้อมูลลงอุปกรณ์เอาต์พุต แสดงดังรูปที่ 2.9 จากรูปจะเห็นได้ว่า เมื่อเริ่มต้นแมทชีนไซเกิ้ลตำแหน่งของอุปกรณ์อินพุต / เอาต์พุต จะถูกแทนลงในบัสตำแหน่งทางด้านไบต์ต่ำ (A0-A7) หลังจากที่ขอบพัลส์ขาขึ้นของ T2 สัญญาณ IORQ แอคตีฟ ถ้าเป็นไซเกิ้ลของการอ่าน สัญญาณ RD จะแอคตีฟพร้อมกับสัญญาณ IORQ ตัวควบคุมอุปกรณ์ภายนอกจะรับรู้สัญญาณการขออ่านนี้ และนำข้อมูลส่งไปบนบัสข้อมูล จากนั้นซีพียูจะสร้าง Tw ขึ้มา 1 คาบเวลา ทั้งๆที่ไม่สัญญาณ WAIT และการตรวจสอบสัญญาณ WAIT จะทำที่ขอบพัลส์ขาลงของ Tw แทนการอ่านข้อมูลจากบัสข้อมูลเข้ายังซีพียูจะทำที่ขอบพัลส์ขาลงของ T3

ไซเกิ้ลของการเขียนข้อมูล สัญญาณ WR จะแอคตีฟในเวลาเดียวกับสัญญาณ IORQ ดังรูปที่ 2.9 และข้อมูลจากรีจิสเตอร์ที่จะส่งไปยังอุปกรณ์อินพุต/เอาต์พุต จะถูกถ่ายลงบัสข้อมูลที่ขอบพัลส์ขาลงของ T1 คือก่อนที่มีสัญญาณ IORQ และ WR หลังจาก T2 ซีพียูจะสร้าง Tw ขึ้นมาอัตโนมัติ 1 คาบ เช่นเดียวกับอ่าน และจากนั้นในช่วงเวลาที่เหลือคือ T3 ตัวควบคุมอุปกรณ์อินพุต / เอาต์พุตจะนำข้อมูลจากบัสข้อมูลเข้าไป



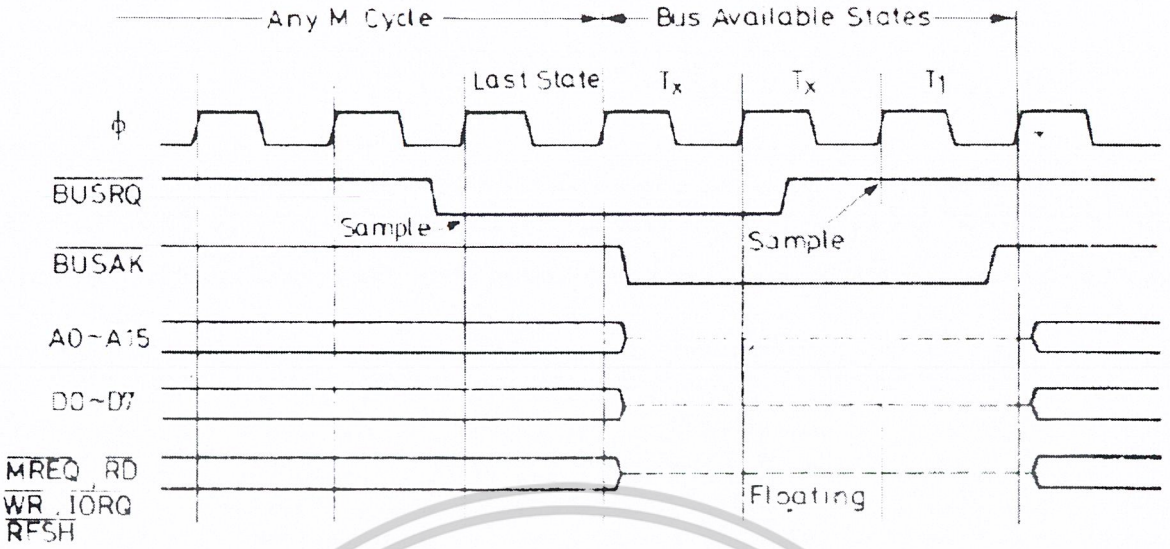
รูปที่ 2.9 ไซเกิ้ลการอ่านหรือเขียนข้อมูลกับอุปกรณ์ อินพุต / เอาต์พุต

จากการทำงานของการอ่านหรือเขียนอุปกรณ์อินพุต / เอาต์พุตมีข้อสังเกตอย่างหนึ่ง คือ หลัง T2 ซีพียูจะใส่คาบการรอคอย(Tw)เข้ามาอัตโนมัติ 1 คาบ ที่เป็นเช่นนี้เพราะว่าในช่วงการทำงานอินพุต / เอาต์พุตนี้ ระยะเวลาจากที่สัญญาณ IORQ เริ่มเกิดขึ้นจนถึงเวลาที่ซีพียูจะต้องตรวจสอบสัญญาณการรอคอยนั้นสั้นมาก หากไม่ใส่ Tw เข้าไปเป็นกรณีพิเศษแล้วช่วงเวลานี้อาจจะสั้นเกินกว่าที่ อินพุต/เอาต์พุตใดๆ จะถอดรหัสตำแหน่งของตัวเอง แล้วส่งสัญญาณ WAIT ได้ทัน นอกจากนี้หากไม่มีการใส่ Tw นี้เข้าไป ยังเป็นการยากมากที่จะออกแบบอุปกรณ์อินพุต / เอาต์พุตที่เป็นพวก MOS ให้ทำงานพอดีกับการทำงานของซีพียู

ไซเกิ้ลการขอใช้บัสหรือการตอบสนองการขอใช้บัส

ที่เวลาใด ๆ ของการทำงานของซีพียู อุปกรณ์ภายนอกสามารถส่งสัญญาณมาควบคุมบัสตำแหน่ง A15 – A0 บัสข้อมูล D7 – D0 และสัญญาณควบคุม MREQ RD WR และ RFSH ได้ โดยการส่งสัญญาณ BUSRQ เข้ามาที่ซีพียู เหตุผลที่ทำเช่นนี้ เนื่องจากเพื่อให้อุปกรณ์ภายนอกสามารถควบคุมการติดต่อกันโดยตรงระหว่างหน่วยความจำภายนอก และอุปกรณ์อินพุต/เอาต์พุตซึ่งทำงานด้วยความเร็วสูงโดยไม่ต้องมีซีพียู วิธีการแบบนี้เรียกว่า Direct Memory Access (DMA) สัญญาณ BUSRQ จะถูกตรวจสอบโดยซีพียูที่ขอบพัลล์ขาขึ้นที่คาบเวลาสุดท้ายของแมทชีนไซเกิ้ลใดๆ เมื่อพบว่าที่ขา BUSRQ นี้ มีระดับ “0” แสดงว่ามีการขอใช้บัสจากภายนอก ดังรูปที่ 2.10

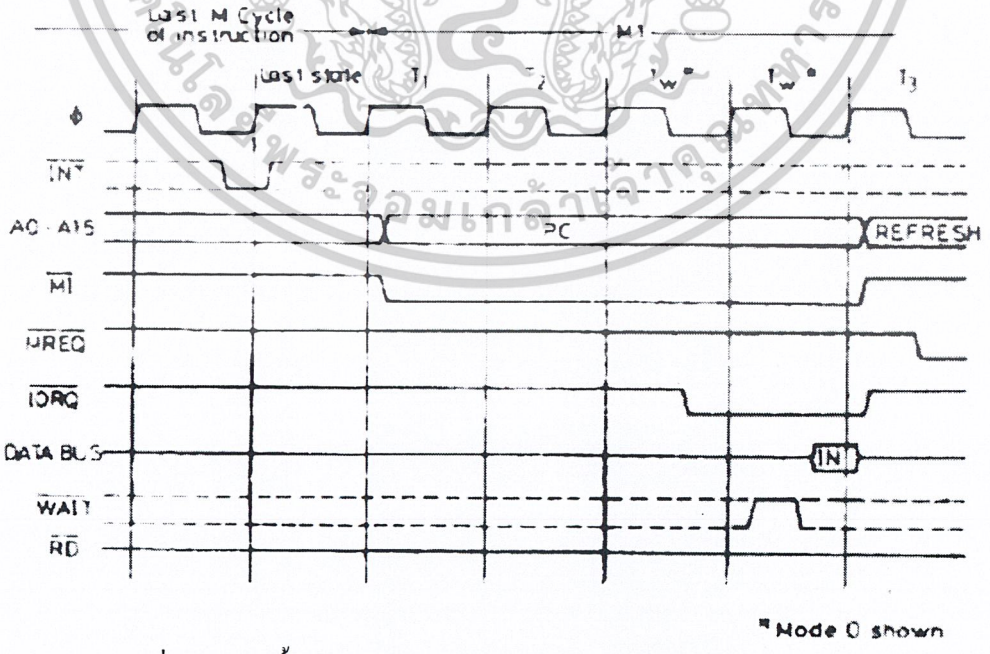
เมื่อซีพียูได้รับสัญญาณ BUSRQ มันจะส่งสัญญาณตอบรับ (BUSAK) เมื่อถึงขอบพัลล์ขาขึ้นขาคาบเวลาต่อมาและในขณะเดียวกัน บัสตำแหน่ง บัสข้อมูล รวมทั้งตำแหน่งควบคุมที่ MREQ RD WR และ RFSH มีสภาวะเป็นอิมพีแดนซ์สูง ดังนั้นการเปลี่ยนแปลงใดๆ บนบัสขณะนี้จะไม่มีผลต่อซีพียู และบัสต่างๆ จะอยู่ในสภาพอิมพีแดนซ์สูงตราบเท่าที่ขา BUSRQ นี้แอคตีฟอยู่ การยกเลิกการขอใช้บัสเกิดขึ้นเมื่อสัญญาณ BUSRQ นี้ไม่แอคตีฟ ซึ่งซีพียูตรวจสอบขอบพัลล์ขาขึ้นของคาบเวลาต่อไปหลังจากที่มีการขอใช้บัสแล้ว จากนั้นคาบเวลาต่อไปสัญญาณ BUSAK จะหายไป และที่ขอบพัลล์ขาขึ้นของคาบเวลาต่อไป บัสต่างๆ ก็กลับสู่สภาพเดิม ข้อสังเกตในการทำ DMA อย่างหนึ่งก็คือ ถ้าระบบใช้หน่วยความจำแบบไดนามิกแรม และช่วงเวลาการทำ DMA นานวงจรภายนอกจะต้องสร้างสัญญาณรีเฟรชหน่วยความจำแบบไดนามิกแรมนี้ และในขณะที่อยู่ในไซเกิ้ลของ BUSRQ ซีพียูจะไม่รับการอินเตอร์รัพทั้งแบบ NMI และ INT



รูปที่ 2.10 ไซเกิ้ลการขอใช้บัสหรือการตอบสนองการขอใช้บัส

ไซเกิ้ลการขออินเตอร์รัพต์และการตอบรับการอินเตอร์รัพต์

ไซเกิ้ลนี้เป็นการอินเตอร์รัพต์แบบมาสเคเบิล ที่อาจเกิดขึ้นที่ขา INT ถ้าอินเตอร์รัพต์ฟลิบฟลอบ (IFF) ถูกเซตให้ยอมรับการอินเตอร์รัพต์ และในขณะนั้นไม่มีการขอใช้บัสโดยสัญญาณ BUSRQ แล้ว แสดงว่าซีพียูสามารถที่จะรับสัญญาณ INT จากอุปกรณ์ภายนอกได้ สัญญาณการขออินเตอร์รัพต์จากอุปกรณ์สัญญาณภายนอกที่ขา INT นี้ ซีพียูจะทำการตรวจสอบที่ขอบพัลส์ขาขึ้นของคาบเวลาสุดท้ายของแมทชีนไซเกิ้ลสุดท้ายของคำสั่ง ถ้าพบว่าที่ขา INT มีระดับ “0” แสดงว่าเกิดการขออินเตอร์รัพต์ และจากนั้นก็เริ่มเข้าสู่ไซเกิ้ลของการอินเตอร์รัพต์ ดังรูปที่ 2.11



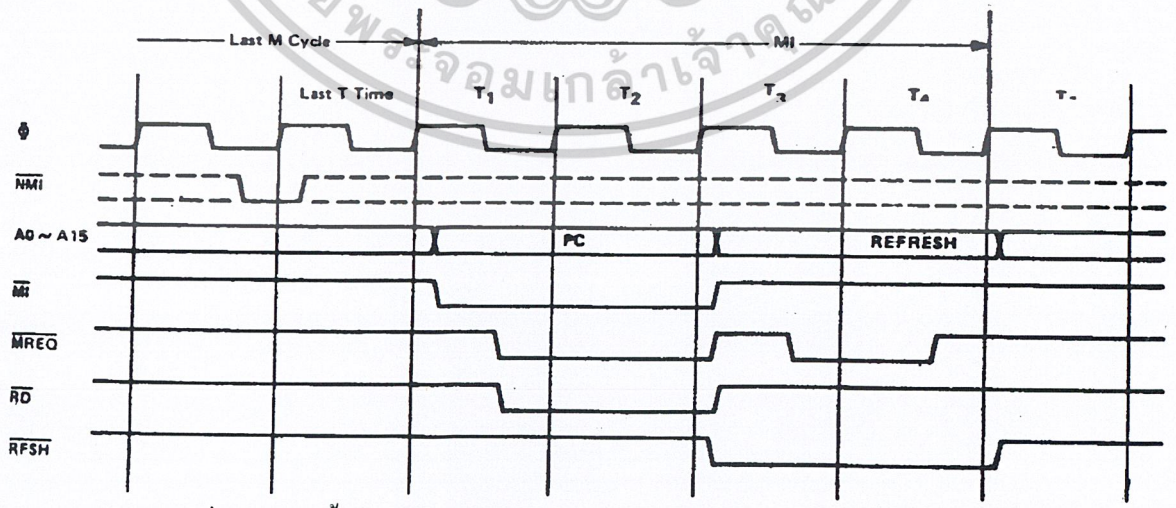
รูปที่ 2.11 ไซเกิ้ลการขออินเตอร์รัพต์และการตอบรับการอินเตอร์รัพต์

เมื่อถึงเวลา T1 ของไซเกิ้ลการอินเตอร์รัพต์ สัญญาณ M1 จะแอคตีฟ และหลังจากคาบเวลา T2 ซีพียูจะสร้าง Tw ขึ้นมา 2 คาบ Tw ที่ใส่เข้ามานี้เพื่อให้สามารถออกแบบวงจรจัดการการอินเตอร์รัพต์ชนิด Ripple Priority ได้โดยง่าย และทำให้มีเวลาเพียงพอที่สัญญาณ Ripple จะอยู่ตัวได้และพอที่จะระบุได้ว่า อุปกรณ์อินพุต/เอาต์พุตใดต้องส่งเวกเตอร์มาบนบัสข้อมูล อุปกรณ์อินพุต/เอาต์พุตภายนอกสามารถรับรู้สัญญาณตอบรับการอินเตอร์รัพต์ได้จากสัญญาณคอมไบเนชั่นระหว่าง M1 กับ IORQเมื่ออุปกรณ์อินพุต / เอาต์พุตได้รับสัญญาณการตอบรับแล้ว มันจะส่งข้อมูลลงในบัสข้อมูล ซึ่งซีพียูจะอ่านขอมูลนี้เข้าไปในขณะช่วงขอบพัลล์ขาขึ้นของ T3 เพื่อเป็นเวกเตอร์ที่ใช้ชี้ตำแหน่งของโปรแกรมบริการการอินเตอร์รัพต์ ในช่วงเวลา T3 สัญญาณ IORQ และ M1 หยุดแอคตีฟและการรีเฟรชก็เริ่มขึ้นส่วนขบวนการบริการการอินเตอร์รัพต์จะขึ้นอยู่กับโมดของการอินเตอร์รัพต์ที่เลือก

ไซเกิ้ลการขอและการตอบรับอินเตอร์รัพท์แบบนอนมาสเคเบิล (Non Maskable Interrupt)

การกระทำของซีพียูในไซเกิ้ลการขอและการตอบรับอินเตอร์แบบนอนมาสเคเบิล (Non Maskable Interrupt) นี้แสดงดังรูปที่ 2.12

สัญญาณการขออินเตอร์รัพต์แบบนี้จะถูกส่งมาจากอุปกรณ์ภายนอกเข้ามาทางขา NMI ของซีพียู และซีพียูไม่สามารถที่จะกันไม่ให้รับการอินเตอร์รัพต์แบบนี้ได้ เนื่องจากสัญญาณที่ขานี้ จัดให้มีลำดับความสำคัญสูงกว่าสัญญาณที่ขา INT สัญญาณ NMI นี้จะแอคตีฟที่ขอบพัลล์ขาลง แต่ซีพียูจะทำการตรวจสอบสัญญาณที่ขานี้ในเวลาเดียวกับตรวจสอบที่ขา INT ถ้าขา NMI มีระดับ 0 แสดงว่ามีการขออินเตอร์รัพต์แบบ Non-Maskable และซีพียูจะทำการตอบสนองในลักษณะเดียวกับการอ่านข้อมูล ข้อแตกต่างมีเพียงแต่ ซีพียูจะไม่ส่งใจข้อมูลจากบัสข้อมูล แต่จะส่งค่าของโปรแกรมเคาน์เตอร์ไปเก็บไว้ในสแตคภายนอก และกระโดดไปทำงานที่ตำแหน่ง 0066H โดยอัตโนมัติ ดังนั้นโปรแกรมบริการการอินเตอร์รัพต์แบบนี้จะต้องเริ่มต้นที่ตำแหน่ง 0066H นี้เท่านั้น

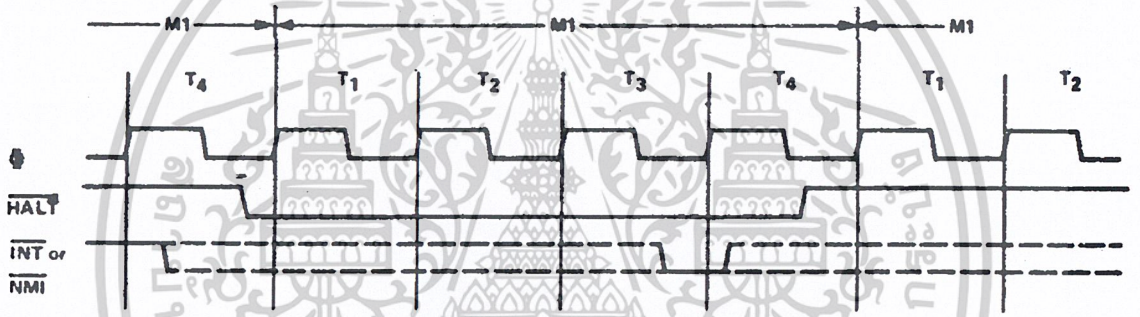


รูปที่ 2.12 ไซเกิ้ลการขอและการตอบรับอินเตอร์รัพท์แบบนอนมาสเคเบิล

ไซเกิ้ลการออกจากคำสั่ง HALT

ในการกระทำโปรแกรม เมื่อใดก็ตามที่ซีพียูปฏิบัติคำสั่ง HALT สัญญาณ HALT จะแอคตีฟ และจากนั้นซีพียูจะสร้างไซเกิ้ล M1 และปฏิบัติคำสั่งในลักษณะ NOP แต่ไม่มีการเพิ่มค่าโปรแกรมเคาน์เตอร์ การปฏิบัติคำสั่ง NOP นี้จะเกิดเรื่อยไปจนกระทั่ง ซีพียูได้รับสัญญาณอินเตอร์รัพต์ที่ขา RESET NMI หรือ INT (เมื่อ IFF=1) เมื่อสัญญาณอินเตอร์รัพต์ NMI และ INT จะถูกตรวจสอบที่ขอบพัลล์ขาขึ้นของ T4 ดังรูปที่ 2.13 ถ้าซีพียูได้รับสัญญาณอินเตอร์รัพต์ มันจะออกจากสภาวะของการ HALT ที่ขอบพัลล์ขาขึ้นของคาบเวลาถัดไป จากนั้นซีพียูจะตอบรับและกระทำการบริการการขออินเตอร์รัพต์ตามชนิดของการอินเตอร์รัพต์ ที่เกิดขึ้น

จุดประสงค์ของการทำคำสั่ง NOP ในช่วง HALT ก็เพื่อที่จะให้สัญญาณรีเฟรชยังคงทำงานต่อไป ในแต่ละไซเกิ้ลของ HALT ซึ่งก็คือ ไซเกิ้ล M1 ธรรมดา ยกเว้นแต่ว่ามันจะไม่สนใจข้อมูลจากหน่วยความจำและรหัสคำสั่ง NOP จะเกิดขึ้นภายในเอง



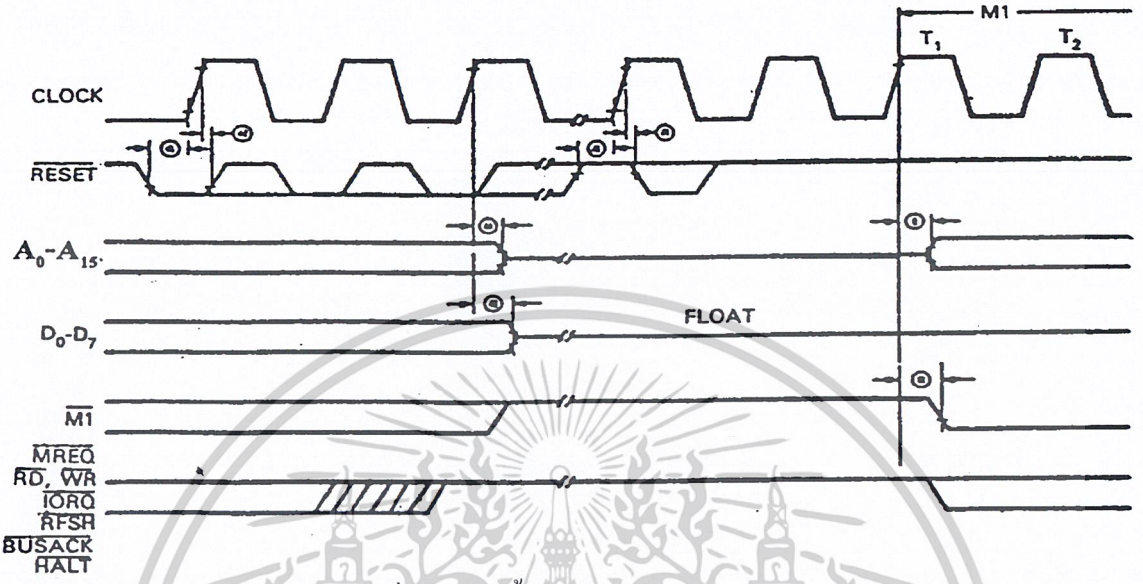
รูปที่ 2.13 ไซเกิ้ลการออกจากคำสั่ง HALT

ไซเกิ้ลของการรีเซท (Reset)

ที่ขาของซีพียู มีขารับสัญญาณรีเซทอยู่ ซึ่งเมื่อต้องการรีเซทตัว ซีพียู ก็จะต้องให้ขานี้แอคตีฟโลจิก “0” อยู่อย่างน้อง 3 ลูกสัญญาณนาฬิกา (ขอบขาขึ้น) เพื่อให้ซีพียูทำงานได้ถูกต้อง สัญญาณแอดเดรสและข้อมูลก็จะถูกปล่อยลอยและสัญญาณควบคุมก็จะควบคุมไม่ได้ และค่าภายในก็จะถูกกำหนดใหม่ดังนี้

- Interrupt enable flip-flop จะ disable เป็นการป้องการ interrupt (INT)
- รีจิสเตอร์ I (interrupt vector register) เป็น 00H
- รีจิสเตอร์ R (Refresh register) เป็น 00H
- ตั้งโหมด interrupt ไปที่ MODE 0
- ไม่มีการรีเฟรช ทั้ง แอดเดรสบัสและบัสข้อมูลถูกปล่อยลอยไว้
- โปรแกรม PC จะเป็น 0000H

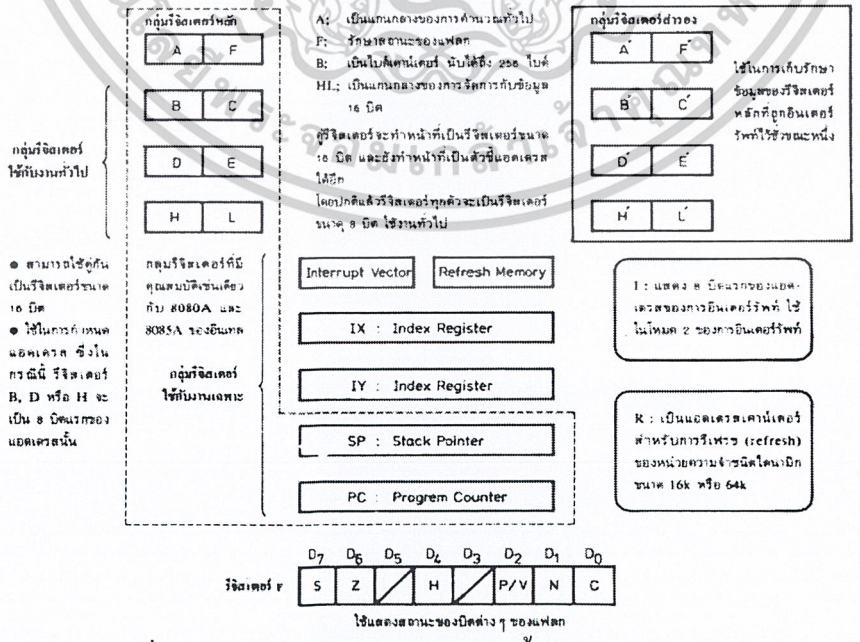
แต่ว่าหลังจากที่เลิกรีเซทนั่นคือเมื่อสัญญาณรีเซทเป็น “1” สัญญาณนาฬิกาก็จะทิ้งระยะไป 2T แล้วตัวซีพียู ก็จะเริ่มทำงานที่ 0000H (PC=0000H) เป็นการเริ่มเฟทช์เอาคำสั่งตำแหน่งนั้นมาตีความหมายทำงานต่อไป เหมือนกับภาวะเริ่มเปิดเครื่องไมโครโปรเซสเซอร์ทำงานอีกครั้ง



รูปที่ 2.14 ไซเกิ้ลของการรีเซท (Reset)

รีจิสเตอร์ และ ALU

โครงสร้างภายในของ Z-80 CPU ประกอบด้วย รีจิสเตอร์ภายในที่สามารถ เขียนและอ่านได้ถึง 208 บิต โดยแยกเป็นกลุ่มของรีจิสเตอร์ขนาด 8 บิต 18 รีจิสเตอร์และรีจิสเตอร์ขนาด 16 บิต อีก 4 รีจิสเตอร์



รูปที่ 2.15 ภาพแสดงรีจิสเตอร์ภายในทั้งหมดของ CPU Z-80

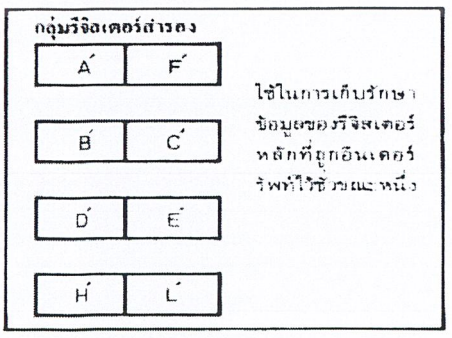
รีจิสเตอร์หลักที่ใช้งานทั่วไป

รีจิสเตอร์ในกลุ่มแรก คือ A,F,B,C,D,E,H,L เป็นรีจิสเตอร์ขนาด 8 บิต ที่ใช้งานทั่วไปโดยรีจิสเตอร์ เหล่านี้สามารถประกอบรวมกันเป็นคู่รีจิสเตอร์ได้ คือ AF,BC,DE และ HL โดยคู่รีจิสเตอร์เหล่านี้ได้รับการใช้งานในลักษณะของรีจิสเตอร์ ขนาด 16 บิต การกระทำภายใน CPU อาจจะอาศัยเพียงรีจิสเตอร์เดียวหรือกระทำเป็นคู่รีจิสเตอร์ได้ โดยที่ A คือ Accumulator , F คือ flag , flag ของ Z-80 จะมีด้วยกันทั้งหมด 6 ตัว จึงใช้เพียง 6 บิต แต่ Z-80 อาศัยการเพิ่มบิตขึ้นอีก 2 บิต และกลายเป็นรีจิสเตอร์ F รีจิสเตอร์ F นี้ ได้รับการ set , reset การกระทำตามคำสั่งทาง คณิตศาสตร์ หรือ ลอจิกได้และเราสามารถใช้ F เหมือนรีจิสเตอร์หนึ่ง ซึ่งเมื่อรวมกับ A แล้ว จะกลายเป็นรีจิสเตอร์ขนาด 16บิตได้

รูปที่ 2.16 ภาพแสดงกลุ่มรีจิสเตอร์หลักใช้งานทั่วไป

กลุ่มรีจิสเตอร์สำรอง

เป็นกลุ่มรีจิสเตอร์ที่สามารถเก็บข้อมูลได้ โดยเป็นตัวเก็บข้อมูลที่มาจากรีจิสเตอร์หลัก รีจิสเตอร์ ชุดนี้จึงมีด้วยกัน 8 ตัว คือ A', F', B', C', D', E', H' ,L' รีจิสเตอร์เหล่านี้เป็นรีจิสเตอร์ที่ใช้ในการ เก็บข้อมูลชั่วคราว ในการที่ต้องการใช้รีจิสเตอร์หลักทำงานอย่างอื่นก่อน ดังนั้นรีจิสเตอร์กลุ่ม นี้จึงไม่สามารถกระทำทางคณิตศาสตร์และลอจิกได้



รูปที่ 2.17 ภาพแสดงกลุ่มรีจิสเตอร์สำรอง

กลุ่มรีจิสเตอร์ที่ใช้งานเฉพาะอย่าง

โปรแกรมเคาน์เตอร์ (PC - Program counter) โปรแกรมเคาน์เตอร์เป็นรีจิสเตอร์ขนาด 16 บิต ที่เป็นตัวกำหนดตำแหน่งของโปรแกรมในขณะสภาวะการกระทำการเฟทช์โดยขณะทำ การเฟทช์ค่าที่อยู่ในโปรแกรมเคาน์เตอร์จะไปปรากฏอยู่ที่แอดเดรสบัสเพื่อชี้ไปยังตำแหน่งในหน่วยความจำ ให้ CPU อ่านคำสั่งมาตีความหมายค่าที่อยู่ในโปรแกรมเคาน์เตอร์จะเพิ่มค่าขึ้นได้อย่างอัตโนมัติ หลังการกระทำการเฟทช์ แต่ถ้าหากซีพียูกระทำคำสั่งให้ข้ามไปยังตำแหน่งอื่น (Jump) ค่าแอดเดรสที่จะกระโดดข้ามนั้นจะโหลดเข้ามายังโปรแกรมเคาน์เตอร์ได้อย่างอัตโนมัติ

สแตคพอยท์เตอร์ (SP - Stack pointer) เป็นรีจิสเตอร์ที่มีขนาด 16 บิตที่ใช้สำหรับชี้ไปยังแอดเดรสชั้นบนสุดของสแตคที่อยู่ใน RAM โดยส่วนของสแตคมีลักษณะโครงสร้างเป็นหน่วยความจำเป็นแบบเก็บทีหลังเรียกออกก่อน (last in first out) ข้อมูลในสแตค อาจได้รับการ push หรือ pop มาจากรีจิสเตอร์ภายในCPU ลักษณะของสแตคในที่นี้ยังเป็น ส่วนช่วยในการกระทำ interrupt และการเรียกโปรแกรมย่อย กล่าวคือ ในการ interrupt ค่าของโปรแกรมเคาน์เตอร์จะได้รับการรักษาไว้ในชั้นสแตคครั้นเมื่อโปรแกรมกลับจาก interrupt ไปกระทำยังโปรแกรมหลักก็จะนำค่าจากสแตคกลับเข้ามายังโปรแกรมเคาน์เตอร์ ใหม่ ในทำนองเดียวกัน การกระโดดไปกระทำยังโปรแกรมย่อยก็เช่นเดียวกัน ดังนั้น การกระทำในรูปของ interrupt ของ โปรแกรมย่อยสามารถ ซ้อนกันได้ไม่มีสิ้นสุด

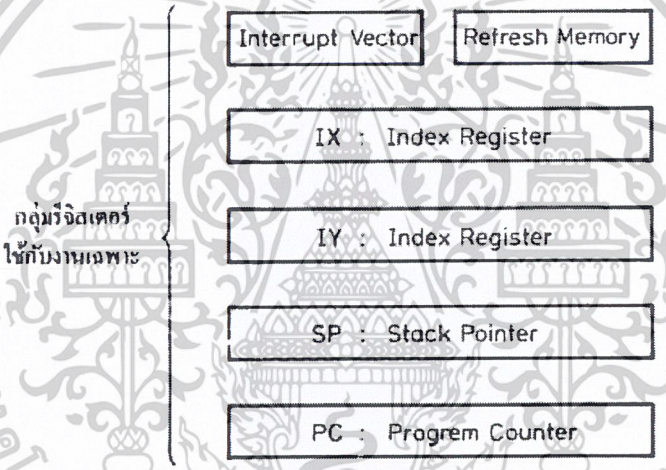
อินเดครีจิสเตอร์ (IX,IY - index register) CPU Z-80 มีอินเดครีจิสเตอร์ขนาน 16 บิต 2 ตัว แต่ละตัวใช้ประโยชน์หลักในการทำหน้าที่เป็นตัวเก็บแอดเดรสฐาน (base address) เพื่อทำหน้าที่อ้างแอดเดรสแบบอินเดคแอดเดรสซิ่ง (index addressing)ในโหมดของอินเดคแอดเดรส ซึ่งมีข้อมูลที่อยู่ในอินเดครีจิสเตอร์นี้จะรวมกับข้อมูลที่ติดมากับคำสั่งอีก 8 บิต เพื่อเป็นตัวกำหนดแอดเดรสให้กับ คำสั่ง ข้อมูลที่ติดมากับคำสั่งนี้เราเรียกว่า displacement ซึ่งจะเก็บในรูปของตัวเลข 2's complement

อินเตอร์รัพท์เพจแอดเดรสรีจิสเตอร์ (I-Interrupt page address register) การอินเตอร์รัพท์ของ Z-80 มี หลายโหมดและโหมดหนึ่งที่ทำให้การอินเตอร์รัพท์ของ Z-80 มีประสิทธิภาพ สูง กล่าวคือ เมื่อเกิดการอินเตอร์รัพท์ในโหมดนี้ขึ้น มันสามารถอ้างแอดเดรสโดยทางอ้อม ไปกระทำโปรแกรมใน ที่ใดก็ได้ในหน่วยความจำ โดยอาศัยค่าในรีจิสเตอร์ รวมกับค่าที่ส่งมาจากอุปกรณ์เพอริเฟอรัลอีก 8 บิต ชี้ไปยังค่าในหน่วยความจำเพื่อนำค่านั้นมาโหลดเข้าใน โปรแกรมเคาน์เตอร์เพื่อกระทำต่อไป ด้วยวิธีการนี้เราจึงสามารถกระโดดเข้าไปทำที่ส่วนใดก็ได้ในหน่วยความจำ

รีจิสเตอร์รีเฟรชหน่วยความจำ (R-memory reflesh register) การต่อซีพียูกับหน่วยความจำนั้นโดย ปกติจะต่อกับหน่วยความจำชนิด static ได้โดยง่าย แต่ชนิด dynamic ที่ต้องการรีเฟรช มีราคาถูกกว่ามีความหนาแน่นสูงกว่า Z-80 ให้ข้อดีกว่าประการหนึ่งคือมันสามารถให้การรีเฟรชหน่วยความจำได้อย่างอัตโนมัติ โดยค่าใน R รีจิสเตอร์จะเพิ่มค่าขึ้นอีก 1 ทุกครั้งที่มีการกระทำการ fetch คำสั่ง และข้อมูลในรีจิสเตอร์ R นี้ จะถูกส่งออกไปยังแอดเดรสบัสในส่วนบิตที่มีนัยสำคัญต่ำกว่าจังหวะของการส่งนี้จะ

เป็นจังหวะเดียวกันกับที่ซีพียูส่งสัญญาณรีเฟรชออกมา ผู้โปรแกรมสามารถกำหนดค่าให้กับรีจิสเตอร์ R นี้ได้แต่ค่าในรีจิสเตอร์ นี้จะเรียกใช้โดยผู้โปรแกรมทางคำสั่งโดยตรงไม่ได้

แอคคิวมูเลเตอร์ (accumulator) และแฟลก (flag) ซีพียูจะมีรีจิสเตอร์ที่ใช้เป็นหลักในการ เป็นตัว operand สำหรับกระทำทางคณิตศาสตร์และลอจิก โดยรีจิสเตอร์หลักนี้จะมีเพียง 8 บิต เรียกว่า "แอคคิวมูเลเตอร์ (accumulator)" การกระทำในส่วนของหน่วยคณิตศาสตร์ และลอจิกย่อมเกิดเงื่อนไขได้หลายอย่างที่จะต้องแสดงสถานะภาพของเงื่อนไขเหล่านั้น เช่น เงื่อนไขผลลัพธ์เป็นศูนย์ผลลัพธ์เป็นบวกหรือลบมีตัวทดหรือตัวขอยืมในการกระทำทางคณิตศาสตร์ แสดงเงื่อนไขพาริตี้คู่หรือคี่สิ่งเหล่านี้จะให้ผลลัพธ์แสดงสถานะ ได้ด้วย แฟลก (flag) แฟลกเป็นรีจิสเตอ์ขนาด 8 บิต ซึ่ง สามารถรวมกับแอคคิวมูเลเตอร์เป็น รีจิสเตอร์ขนาด 16 บิต ได้ผู้โปรแกรมยังสามารถใช้คำสั่งในการเคลื่อนย้ายข้อมูลจาก แอคคิวมูเลเตอร์ A และแฟลก F ไปเก็บไว้ใน A' และ F' ได้ เพื่อทำให้การใช้งาน ของ A และ F มีประสิทธิภาพดียิ่งขึ้น



รูปที่ 2.18 ภาพแสดงกลุ่มรีจิสเตอร์ใช้งานเฉพาะอย่าง

การอินเตอร์รัพต์ (INTERRUPT)

Z-80 มีขาอินพุตสำหรับอินเตอร์รัพท์ได้ 2 ขา คือ อินเตอร์รัพท์แบบมาสเคเบิ้ล (INT) และแบบนอนมาสเคเบิ้ล (MNI)สำหรับการอินเตอร์รัพท์แบบมาสเคเบิ้ล สามารถให้ CPU กระโดดข้ามไปทำงานที่ส่วนต่าง ๆ ของโปรแกรมได้ส่วนการ อินเตอร์รัพท์แบบนอนมาสเตเบิ้ลนั้นแตกต่างกับแบบมาสเคเบิ้ล คือการอินเตอร์รัพท์แบบนี้จะเกิดขึ้นทันทีโดยที่ผู้ใช้ไม่สามารถหยุดยั้งหรือดิสสเบิ้ลได้ด้วยซอฟท์แวร์

นอนมาสเคเบิ้ลอินเตอร์รัพท์(NON-MASKABLE INTERRUPT) การอินเตอร์รัพท์จะป้อนเข้าทางขา NMI ของซีพียูโดยที่ลอจิกของการแอคตีฟจะเกิดขึ้นด้วยลอจิก "0" เมื่อซีพียูรับสัญญาณนี้แล้ว ซีพียูจะไม่กระทำคำสั่งถัดไป โดยจะตอบสนองต่อการอินเตอร์รัพท์โดยการเปลี่ยนข้อมูลให้ PC เป็น 0066H เพื่อให้ภาวะการเฟทช์ครั้งต่อไปเกิดขึ้นที่แอดเดรสนี้ การตอบสนองในกรณีของนอนมาสเคเบิ้ลนี้ ซีพียูถือว่าเป็นส่วนสำคัญที่สุดที่จะต้องกระทำโดยคำสั่งทางซอฟท์แวร์หรือขบวนการอื่นใดไม่

สามารถเข้ามาหยุดยั้งการอินเตอร์รัพท์ได้ การอินเตอร์รัพท์ด้วยวิธีนี้จึงมักใช้ในกรณีว่ามีเหตุการณ์สำคัญที่สุด จากหลักการทั่วไปของการอินเตอร์รัพท์ ค่าเดิมของ PC ที่ซีพียูจะกระทำต่อไปในโปรแกรมหลักจะได้รับการเก็บรักษาไว้ในสแตคและการกลับคืนสู่โปรแกรมหลักเดิมนั้นกระทำได้ด้วยคำสั่ง RETN (Return from Nonmaskable)

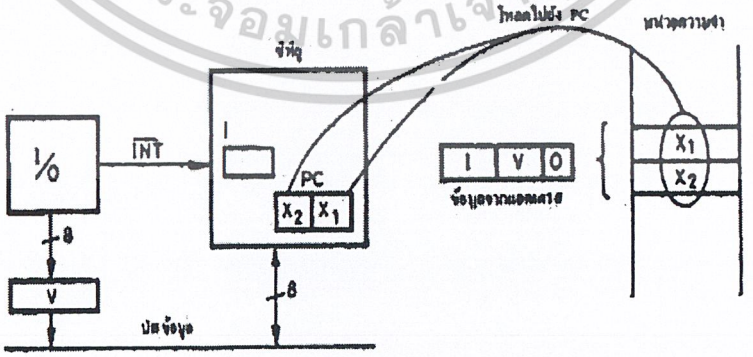
การอินเตอร์รัพท์แบบมาสเคเบิ้ล (Maskable Interrupt) ในการอินเตอร์รัพท์แบบนี้ ผู้ใช้ต้องอินเตอร์รัพท์ผ่านเข้ามาทางขา INT ของซีพียู เมื่อซีพียูได้รับสัญญาณนี้แล้วซีพียูจะตรวจสอบสถานะของตัวเองว่าจะตอบสนองต่อการอินเตอร์รัพท์หรือไม่ การที่จะตอบสนองหรือไม่สามารถโปรแกรมได้ด้วยซอฟท์แวร์ ดังนั้นผู้โปรแกรมจึงสามารถกำหนดสภาวะการอินเตอร์รัพท์ให้ได้ รับการตอบสนองตรงส่วนใดของโปรแกรมได้การอินเตอร์รัพท์ด้วยวิธีนี้สามารถแยกแยะการอินเตอร์รัพท์ออกเป็น 3 โหมด ซึ่งการแยกโหมดก็ทำได้ด้วยการใช้ซอฟท์แวร์หรือคำสั่งนั่นเอง โดยแยกเป็นโหมด 0 (IM0), โหมด 1 (IM1), โหมด 2 (IM2)

การอินเตอร์รัพท์โหมด 0 ในโหมดนี้ผู้ออกแบบซีพียู Z-80 ได้ออกแบบมาเพื่อให้ Z-80 ทำการตอบสนองต่อสัญญาณอินเตอร์รัพท์เหมือน 8080 ทุกประการ กล่าวคือเมื่อมีการอินเตอร์รัพท์ เกิดขึ้นและโปรแกรมทางซอฟท์แวร์ได้เซทโหมดการรับอินเตอร์รัพท์เป็นโหมด 0 (IM0) และอีนาเบิ้ลการอินเตอร์รัพท์ไว้การทำงานของซีพียูจะหยุดการเฟทช์คำสั่ง ถัดไป แต่จะตอบรับการอินเตอร์รัพท์ด้วยการส่งสัญญาณตอบรับด้วย /M1 กับ /IORQ อ่านข้อมูล 1 ไบต์ เข้ามาทางบัสข้อมูล ข้อมูล 1 ไบต์นี้ได้รับการส่งมาจาก อุปกรณ์ I/O ที่อินเตอร์รัพท์มา เมื่อซีพียูอ่านข้อมูลไบต์นี้มาซีพียูจะถือว่าเป็นออพโค้ดทันทีและจะตีความหมายในการทำงาน คำสั่งขนาด 1 ไบต์ที่เหมาะ ในการใช้ในการอินเตอร์รัพท์ก็คือคำสั่ง RST เมื่อซีพียูเอ็กซีคิ้วคำสั่ง RST ซีพียูจะเก็บข้อมูลเดิมใน PC ไว้ที่สแตค แล้วเปลี่ยนค่า PC ใหม่ตามลักษณะของการ RST นั้น ๆดังนั้นซีพียูจะกระโดดข้ามไปทำงานตามที่ต้องการของการอินเตอร์รัพท์ได้ การตอบสนองต่อการอินเตอร์รัพท์ด้วยการส่งสัญญาณ M1 และ IORQ ทำการเฟทช์ข้อมูลจาก I/O มาเอ็กซีคิ้วนี้อาจทำได้โดยการให้ I/O ส่งคำสั่งอื่นที่ไม่ใช่คำสั่ง RST (คำสั่งไบต์เดียว) แต่เป็น คำสั่งหลายไบต์ เช่นคำสั่ง CALL ซึ่งเป็นคำสั่งขนาด 3 ไบต์ โดย I/O ส่งข้อมูลขณะตอบรับการ อินเตอร์รัพท์(/M1+/IORQ) ด้วยข้อความ CDH (CALL) เมื่อซีพียูเอ็กซีคิ้วคำสั่งนี้ก็จะทราบว่าเป็นคำสั่ง CALL ซึ่งยังต้องการข้อมูลเพิ่มอีก 2 ไบต์ ซีพียูจะสร้างแมชีน ไซเคิลต่อไปในการอ่านหน่วยความจำเหมือนกระทำคำสั่ง CALL จริง ๆในการนี้เราจะต้องใช้วงจรทางฮาร์ดแวร์ประกอบเพื่อให้ I/O ส่งข้อมูลอีก 2 ไบต์ ตามไปให้ได้ซึ่งการอินเตอร์รัพท์ด้วยวิธีนี้อาจกำหนดได้ด้วยเทคนิคที่แตกต่างกันตามการออกแบบของแต่ละบุคคล บริษัทอินเทลได้ออกแบบไอซีที่ใช้ในการควบคุมการอินเตอร์รัพท์และจัดลำดับความสำคัญในการอินเตอร์รัพท์ขึ้น เช่น IC เบอร์ 8214 การใช้เวลาในการตอบสนองต่อการอินเตอร์รัพท์ด้วยการ ส่ง M1 และ IORQ นี้จะกระทำเหมือนการกระทำ ในช่วงเวลาแมชชีนไซเคิล /M1 แต่ซีพียูจะใช้เวลายาวนานกว่าแมชชีนไซเคิล M1โดยปกติ โดยการเพิ่ม Tw ขึ้นก็เพื่อให้เวลาแก่อุปกรณ์ I/Oในการกระทำขบวนการเดซี่เชน (Daisy Chain) ในการจัดลำดับความสำคัญของการอินเตอร์รัพท์ โดยปกติการเซทให้อยู่ใน

โหมดนี้กระทำได้ด้วยการสั่งทางซอฟท์แวร์ด้วยคำสั่ง IM1 และการกำหนดให้รับการอินเตอร์รัพท์ได้หรือไม่กระทำได้ด้วยคำสั่ง EI และ DI และเพื่อให้การทำงานเหมือนอยู่ในโหมดของ 8080 ทุกประการ ดังนั้นหลังจากการรีเซทซีพียู ซีพียูจะเซทตัวเองให้อยู่ในโหมด 0 นี้โดยอัตโนมัติ

การอินเตอร์รัพท์โหมด 1 ในโหมดนี้ผู้โปรแกรมสามารถกำหนดได้ด้วยคำสั่ง IM1 การอินเตอร์รัพท์ในโหมด นี้จะกระทำเหมือนกันกับนอนมาสเคเบิ้ลอินเตอร์รัพท์ทุกประการแต่จะแตกต่างกันก็เพียงการรีสตาร์ทมาที่ตำแหน่ง 0038H (กรณีนอนมาสเคเบิ้ลไปกระทำที่ 0066H) และจำนวนคาบเวลาที่ใช้ในโหมด 1 นี้มีมากกว่าในนอนมาสเคเบิ้ลทั้งนี้ เพราะในโหมดนี้ซีพียูต้องเพิ่ม Tw ขึ้นอีก 2 สเตทการอินเตอร์รัพท์ในโหมดนี้สามารถดิสเอเบิ้ลได้ด้วยซอฟแวร์

การอินเตอร์รัพท์โหมด 2 ในโหมดนี้ทำให้ Z-80 มีขีดความสามารถเกี่ยวกับการอินเตอร์รัพท์สูงขึ้นมาก การอินเตอร์รัพท์ในโหมดนี้กำหนดได้ด้วยคำสั่ง IM2 และการจะให้ซีพียูตอบสนองหรือไม่ก็ยังใช้คำสั่งโปรแกรมได้เช่นกันโดยใช้คำสั่ง EI และ DI การกระโดดไปยังโปรแกรมอื่นในขณะเมื่อซีพียูตอบสนองต่อการอินเตอร์รัพท์ ในกรณีนี้จะไปที่ใดก็ได้ โดยใช้แอดเดรสในการกระโดดนี้ได้ถึง 16 บิต ซึ่งทำให้การอินเตอร์รัพท์ทำได้สะดวกและรวดเร็วขึ้นอีกมาก กรรมวิธีการตอบสนองต่อการอินเตอร์รัพท์ในกรณีนี้คือ เมื่อมีสัญญาณ /INT เข้ามาและซีพียูตรวจสอบได้ในตอนสุดท้ายของคำสั่ง ซีพียูจะตอบสนองด้วยการส่ง /M1 กับ /IORQ ออกไป สัญญาณ /M1 กับ /IORQ จะเป็นตัวบอกอุปกรณ์ที่ ส่ง /INT มาให้ส่งข้อมูลขนาด 1 ไบต์เข้าทางบัสข้อมูล สำหรับในโหมดนี้ข้อมูลที่ส่งจาก I/O ขนาด 1 ไบต์ที่เข้าทางบัสข้อมูลนี้ซีพียูถือว่าเป็น เวคเตอร์ของ การอินเตอร์รัพท์ โดยข้อมูลในบิต D0 จะต้องเป็น "0" ส่วนบิตอื่นจะเป็นอะไรก็ได้ซีพียูจะนำเวคเตอร์นี้ไปเป็นข้อมูลแอดเดรสไบต์ที่มีนัยสำคัญต่ำและข้อมูลจากรีจิสเตอร์ 1 ภายในซีพียูเป็นข้อมูลไบต์ที่มีนัยสำคัญสูง เรียกหาไปยังข้อมูลในหน่วยความจำเพื่ออ่านข้อมูลในหน่วยความจำ 2 ไบต์ติดกันนั้นมาโหลดใส่ PC หรือเป็นการอ้างแอดเดรสให้ PC แบบโดยทางอ้อมนั่นเอง



รูปที่ 2.19 การอินเตอร์รัพท์โหมด 2

สำหรับกรณีนี้จะเห็นว่าเราสามารถเซทค่าในรีจิสเตอร์ I ให้เป็นอะไรก็ได้และ I/O จะส่งข้อมูลเวคเตอร์มาประกอบรวมเพื่อบอกถึงค่าตารางในหน่วยความจำที่ต้องการ ด้วยวิธีการเช่นนี้จะทำให้การกระโดดไปยังโปรแกรมย่อยเกิดขึ้นที่ใดก็ได้

อุปกรณ์ไอซีที่ทำงานร่วมโดยใช้อินเตอร์รัพท์โหมดนี้มีหลายเบอร์ด้วยกัน เช่น Z-80PIO, Z-80CTC, Z-80SIO ฯลฯ ซึ่งอุปกรณ์อินเตอร์เฟสเหล่านี้สามารถส่งเวคเตอร์ให้กับซีพียูได้อย่างมีประสิทธิภาพ

การอินเตอร์รัพท์ในโหมดนี้ซีพียูต้องการเวลาถึง 19 สเตท โดยใช้ 7 สเตทใน การเฟทช์เวคเตอร์ 6 สเตทในการเก็บข้อมูล PC เดิมในสเตท และ 6 เป็นการอ่านข้อมูลจากหน่วยความจำมายัง PC

การอีนาเบิ้ลและดิสเอเบิ้ลอินเตอร์รัพท์ สถานะภาพต่อการตอบสนองอินเตอร์รัพท์ ซีพียูจะทำการตรวจสอบที่ IFF หรือ อินเตอร์รัพท์ฟลิบฟลอบในกรณีของ Z-80 จะมีฟลิบฟลอบที่แสดงสถานะภาพ ในการอินเตอร์รัพท์อยู่ 2 บิต คือ IFF1 และ IFF2 โดยทั้งสองบิตนี้จะได้รับการเกี่ยวข้องด้วยจากการกระทำของซีพียูหรือของผู้โปรแกรมโดยโปรแกรมคำสั่ง เข้ามาเซทหรือรีเซทฟลิบฟลอบ โดยหลักการ IFF1ทำหน้าที่เป็นตัวกำหนดการอีนาเบิ้ลหรือดิสเอเบิ้ลการอินเตอร์รัพท์ โดยที่ IFF2 จะมีหน้าที่หลักในการเก็บข้อมูลชั่วคราวของ IFF1 ในขณะที่มีการรีเซทซีพียูทางขารีเซท ทั้ง IFF1 และ IFF2 จะได้รับการรีเซทไปด้วย การแสดงสถานะ "0" ของ IFF1 จะเป็นการดิสเอเบิ้ลการอินเตอร์รัพท์ กล่าวคือ IFF1 = "0" ซีพียูยะไม่รับรู้ต่อการอินเตอร์รัพท์ที่เข้ามาทาง INT การเซท IFF สามารถกระทำได้ด้วยคำสั่ง EI โดยสถานะภาพของ IFF1 และ IFF2 ที่จะเปลี่ยนแปลงเนื่องจากการกระทำต่าง ๆ สรุปได้ดังตารางข้างล่างนี้

การกระทำ	IFF1	IFF2	
รีเซทซีพียู	0	0	
DI	0	0	
EI	1	1	
LD A, I	•	•	IFF2 ⟶ แฟลกพาริตี้
LD A, R	•	•	IFF2 ⟶ แฟลกพาริตี้
เมื่อกระทำ NMI	0	•	
RETN	IFF2	•	IFF2 ⟶ IFF1
เมื่อกระทำ INT	0	0	
RETI	•	•	

"•" หมายถึงไม่เปลี่ยนแปลง

รูปที่ 2.20 รูปแสดงตารางการอีนาเบิ้ลและดิสเอเบิ้ลอินเตอร์รัพท์

จากตารางพอสรุปได้ว่า การกำหนดอีนาเบิ้ลจะต้องทำการเซทฟลิบฟลอบ IFF1หรือกล่าวอีกนัยหนึ่ง การยอมให้เกิดการอินเตอร์รัพท์ได้ก็ต่อเมื่อซีพียูตรวจสอบ IFF1 ว่าอยู่ในสภาวะอีนาเบิ้ลหรือไม่ การตรวจสอบสภาวะอีนาเบิ้ลว่าได้รับการอีนาเบิ้ลหรือดิสเอเบิ้ล ในบางกรณีทำได้โดยการตรวจสอบทางบิตพาริตี้นั่นคือ การกระทำคำสั่ง LD A, I และ LD A, R จะมีผลให้ค่าของ IFF2 ไปเก็บยังพาริตี้แฟลก

เมื่อมีการอินเตอร์รัพท์แบบ NMI ขึ้นจะเกิดสภาวะดิสเอเบิ้ลทันทีที่ IFF1 กล่าวคือมันจะได้ รับการรีเซท นั่นคือระหว่างการอินเตอร์รัพท์แบบ NMI นี้ การอินเตอร์รัพท์แบบอื่นจะเข้ามาอีกไม่ได้ ซีพียูจะไม่รับรู้ทั้งสิ้นสถานะเดิมก่อนการ อินเตอร์รัพท์แบบ NMI (สถานะการดิสเอเบิ้ลหรืออีนาเบิ้ล)จะได้รับการเก็บรักษาไว้ที่ IFF2 ซึ่งระหว่างนี้จะได้รับการตรวจสอบได้เช่นกันว่า ก่อนการเข้าไปสู่โหมด NMI สภาวะการเป็นอย่างไร และเมื่อกลับเข้าโปรแกรมหลักด้วยคำสั่ง RETN จะทำให้สถานะเดิมเก็บรักษาไว้ใน IFF1 ใหม่การตอบสนองต่อ INT ก็จะทำให้ IFF1 และ IFF2 ได้รับการรีเซทเช่นกัน ดังนั้นเมื่อมีการ INT สัญญาณ INT ครั้งต่อไปจะไม่สามารถได้รับการตอบสนองจนกว่าจะมีคำสั่ง EI /INT จึงจะได้รับการตอบสนองนั่นเอง ส่วนการเอ็กซีคิ้วคำสั่ง RETI จะไม่มีผลทำให้ IFF1 และ IFF2 เกิดการเปลี่ยนแปลง

ไดอะแกรมเวลาสำหรับการตอบสนองต่อการอินเตอร์รัพท์ของซีพียู การสนองต่อการอินเตอร์รัพท์ของซีพียูมีลักษณะสำคัญที่เราจะต้องพิจารณาในแง่ของไดอะแกรมเวลา เพื่อการเชื่อมต่อกับอุปกรณ์ I/O ในแง่ทางฮาร์ดแวร์จะได้เป็นไปอย่างถูกต้อง

สรุปการตอบสนองจะเกิดตามลำดับดังนี้

1. อุปกรณ์ I/O ส่งสัญญาณอินเตอร์รัพท์มา โดยการทำให้ขา /INT อยู่ในลอจิก "0"

2. CPU จะตอบสนองต่อการอินเตอร์รัพท์ด้วยการส่ง M1 ลอจิก "0" ก่อนเพื่อให้อุปกรณ์ I/O เตรียมจัดการเกี่ยวกับขบวนการจัดลำดับก่อน แล้ว IORQจากซีพียูจึงส่งตามมา การให้ IORQ มาทีหลัง M1ก็เนื่องจากให้ช่วงเวลาดีเลย์ระหว่างนี้เป็นตัวกำหนด IEI และ IEO ของการกระทำเดซีเชน เมื่อ IORQ ออกไปที่ I/O และถ้า IEI ของอุปกรณ์นั้นเป็นลอจิก "1" ตัว I/O นั้นก็จะส่งเวคเตอร์เข้ามาทางบัสข้อมูล และซีพียูจะใช้ IORQ และ M1 เป็นพัลซ์สั่งอ่านเข้าทางบัสข้อมูลและการทำเดซีเชน อุปกรณ์ที่ส่งอินเตอร์รัพท์ได้ จะให้ IEO เป็นลอจิก "0" เพื่อป้อน เข้า IEI ของตัวอื่นเป็นการบล็อกการส่งเวคเตอร์จาก I/O ตัวอื่น

3. การเคลียร์อินเตอร์รัพท์ โดยปกติอุปกรณ์I/O จะแอคตีฟในการส่งอินเตอร์รัพท์ ได้ต้องให้ IEI = 1 และ IEO = 0 ดังนั้นเมื่อเสร็จสิ้นการกระทำการอินเตอร์รัพท์ แล้วสถานะของการ I/O จะต้อง เคลียร์ตัวเองได้ นั้นคือเมื่อมีการกระทำคำสั่ง RETI (ED 4D) ก็จะมีการบอกให้อุปกรณ์ I/O ทราบว่าจบสิ้นการอินเตอร์รัพท์ แล้ว เพื่อทำให้ IEO เป็น "1" เพื่อการอีนาเบิ้ลตัวI/O ที่มีความสำคัญน้อยกว่า ความสัมพันธ์ของ /INT, /NMI และ /BUSRQ

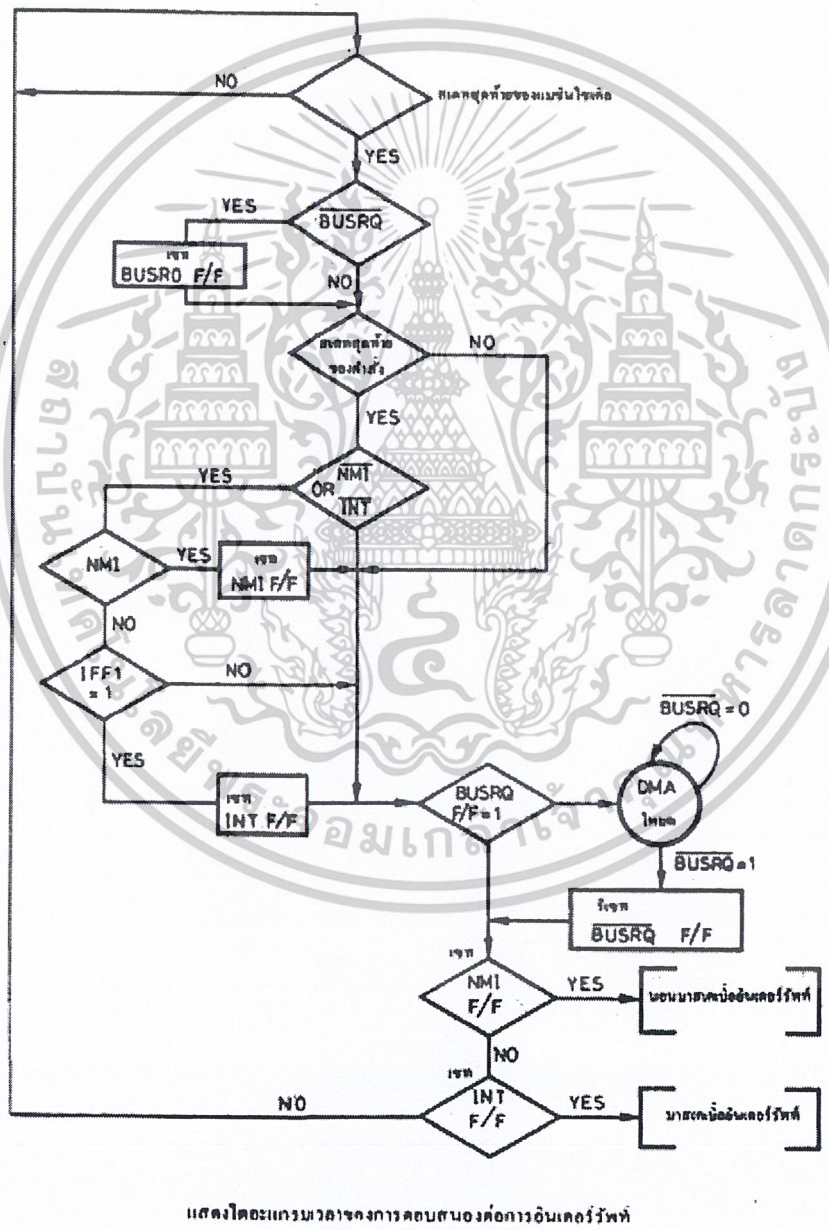
ไดอะแกรมเวลาข้างล่างนี้เป็นไดอะแกรมการจัดลำดับการทำงานภายใน ในการตรวจสอบการอินเตอร์รัพท์ทั้งแบบ INT และ NMI รวมทั้งขบวนการขอใช้บัส BUSRQ โดยมีลักษณะการทำงานที่น่าสนใจดังนี้

1. การตรวจสอบ INT และ NMI ซีพียูจะทำการตรวจสอบทุกๆ ไซเคิลสุดท้ายของสัญญาณนาฬิกาของแต่ละคำสั่ง

2. BUSRQ จะได้รับการตรวจสอบทุก ๆ ตอนสุดท้ายของแมชีนไซเคิล

3. เมื่อ CPU อยู่ในสภาวะ DMA จะไม่ยอมรับรู้การขอ INT หรือ NMI

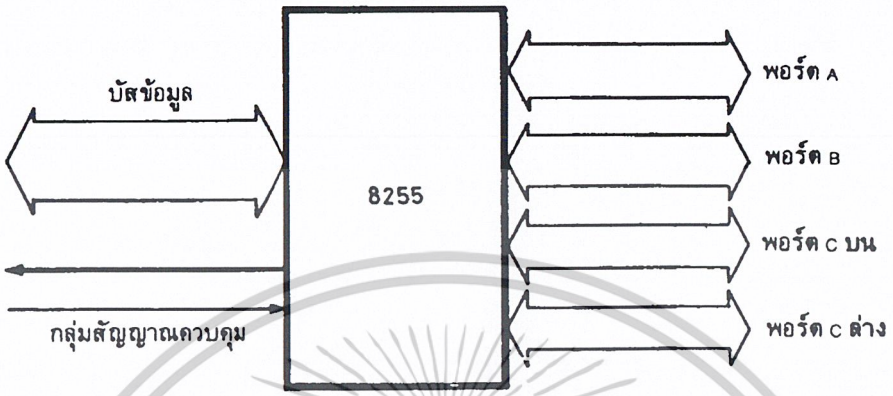
4. การจัดการตอบสนองจะเป็นไปตามลำดับดังนี้ BUSRQ /NMI หรือ INT



รูปที่ 2.21 ไดอะแกรมการจัดลำดับการทำงานภายในการตรวจสอบการอินเตอร์รัพท์

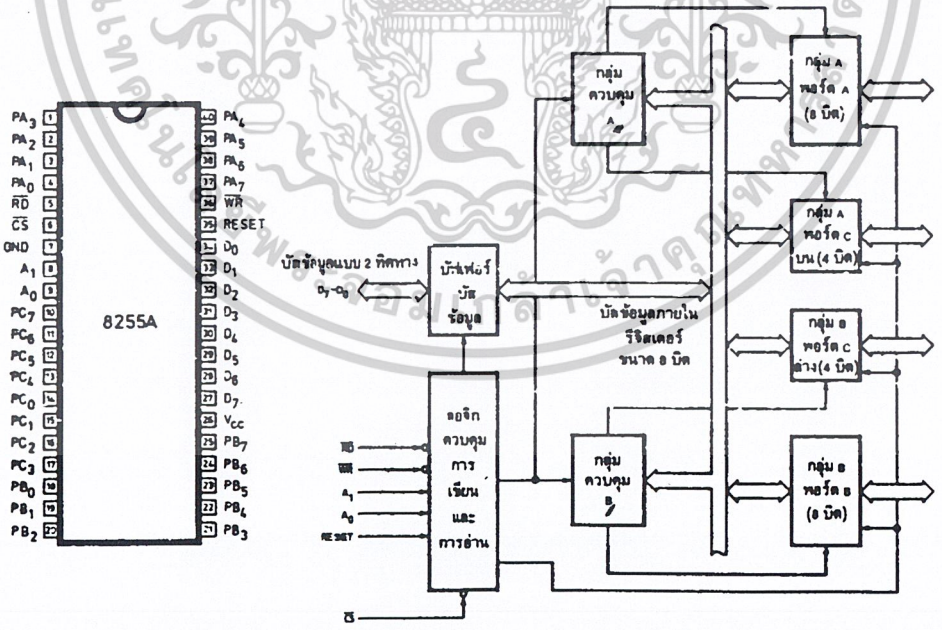
ไอซี 8255

8255 เป็นไอซีที่มี 40 ขา ได้รับการออกแบบมาให้มีสัญญาณเพื่อเชื่อมต่อกับ 8080 แต่ สัญญาณนี้พอเหมาะที่จะใช้กับ Z-80 ได้ดีเช่นเดียวกัน 8255 เป็นไอซีที่ต่อพอร์ตให้ไมโครโปรเซสเซอร์ได้ 3 พอร์ต โดยมีโครงสร้างพื้นฐานแสดงดังรูปที่ 2.22



รูปที่ 2.22 แสดงโครงสร้างพื้นฐานของ ไอซี 8255

การเรียกพอร์ตของ 8255 จะเรียกพอร์ตต่างๆว่า พอร์ต A พอร์ต B และพอร์ต C โดยพอร์ต C แยกเป็น 2 ส่วนคือ พอร์ต C ล่างหรือตั้งแต่ $PC_0 - PC_3$ มีจำนวน 4 บิต และพอร์ต C บนหรือตั้งแต่ $PC_4 - PC_7$ ที่พิเศษคือ พอร์ตทุกพอร์ตเป็นได้ทั้งพอร์ตอินพุตและพอร์ตเอาต์พุต



รูปที่ 2.23 แสดงแผนผังวงจรภายในและการจัดขาของ ไอซี 8255

รูปที่ 2.23 เป็นแผนผังภายในของไอซีและการจัดวางขาของไอซี 8255 การทำงานของวงจรจะใช้สัญญาณควบคุมจากไมโครโปรเซสเซอร์มาควบคุมการทำงาน โดยไมโครโปรเซสเซอร์จะส่งคำสั่งมาโปรแกรมการทำงานหรือกำหนดรูปแบบของพอร์ตให้เป็นอินพุตหรือเอาต์พุตได้

ขาต่างๆของ 8255

เพื่อให้เข้าใจวิธีการต่อใช้งานระหว่าง Z-80 กับ 8255 จึงจำเป็นต้องเข้าใจความหมายและตำแหน่งของขาต่างๆเสียก่อน ขาทั้ง 40 ขาของไอซีประกอบด้วย

$D_0 - D_7$ เป็นขาที่ข้อมูลอินพุต/เอาต์พุตจะต้องผ่านเข้าออกจากส่วนนี้ $D_0 - D_7$ จึงต่อเข้ากับระบบบัสของไมโครโปรเซสเซอร์ เพื่อให้ไมโครโปรเซสเซอร์ สามารถอ่านหรือเขียนข้อมูลจากพอร์ตผ่านทางบัสนี้

$\overline{CS}$ (สัญญาณเลือกชิป) ขานี้เป็นขาอินพุตที่จะรับสัญญาณจากภายนอกเพื่อเลือกชิป 8255 โดยเมื่อขานี้เป็น "0" จะทำให้ 8255 ต่อเข้ากับระบบบัสของไมโครโปรเซสเซอร์ เพื่อให้ไมโครโปรเซสเซอร์เขียนหรืออ่านข้อมูลจากพอร์ตได้

$\overline{RD}$ (สัญญาณการอ่าน) เป็นสัญญาณอินพุตที่ต้องส่งมาจากซีพียูเมื่อสัญญาณที่ขานี้เป็น "0" และสัญญาณ $\overline{CS}$ เป็น "0" ด้วย ไอซี 8255 จะทำตัวให้ซีพียูอ่านข้อมูลจากบัสในขณะที่เป็นพอร์ตอินพุต

$\overline{WR}$ เป็นสัญญาณการเขียน จะแอคตีฟเมื่อสัญญาณ $\overline{WR}$ และสัญญาณ $\overline{CS}$ เป็น "0" สัญญาณนี้จะมาจากซีพียูเมื่อต้องการเขียนข้อมูลลงบนพอร์ตที่กำหนด

$A_0 - A_1$ (สัญญาณแอดเดรส) ลอจิกของสัญญาณทั้งสองจะถอดรหัสออกเป็น 4 รหัส เพื่อกำหนดรีจิสเตอร์ภายในที่เชื่อมต่อกับพอร์ตอินพุตเอาต์พุตของ 8255

$\overline{RESET}$ (สัญญาณรีเซต) เป็นสัญญาณที่ส่งจากภายนอกเข้ามาทำการรีเซต 8255 เพื่อเคลียร์สถานะต่างๆ ของ 8255 เมื่อ 8255 ได้รับการรีเซต ก็จะกลับเข้าสู่โหมดอินพุตหรือทุกพอร์ตที่เป็นพอร์ตอินพุต

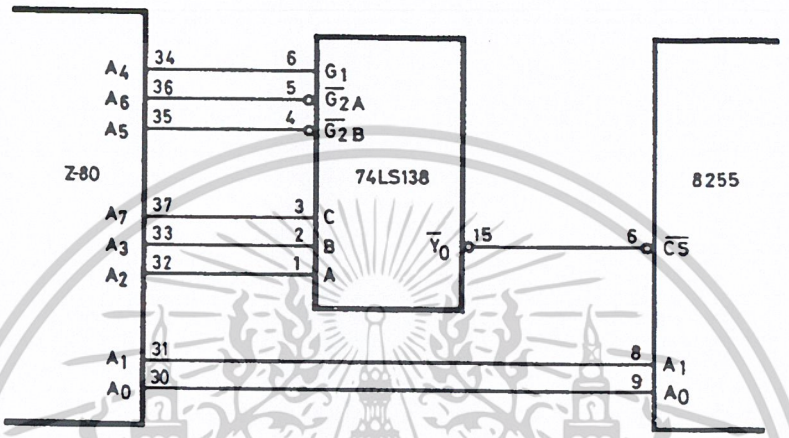
$PA_0 - PA_7$ เป็นสายสัญญาณที่เป็นพอร์ตของ 8255 ที่ชื่อพอร์ต A การเลือกพอร์ตจะเลือกโดยสัญญาณแอดเดรส $A_0 - A_1$

$PB_0 - PB_7$ เป็นสายสัญญาณที่เป็นพอร์ต B ของ 8255 ถูกเลือกโดยสัญญาณแอดเดรส $A_0 - A_1$

$PC_0 - PC_7$ เป็นสายสัญญาณที่เป็นพอร์ต C ของ 8255 การกำหนดพอร์ตนี้จะได้รับการกำหนดโดยสัญญาณแอดเดรส $A_0 - A_1$ พอร์ต C นี้แบ่งเป็น 2 กลุ่มคือ กลุ่ม $PC_0 - PC_3$ และกลุ่ม $PC_4 - PC_7$

การเชื่อมต่อ 8255 กับ Z-80

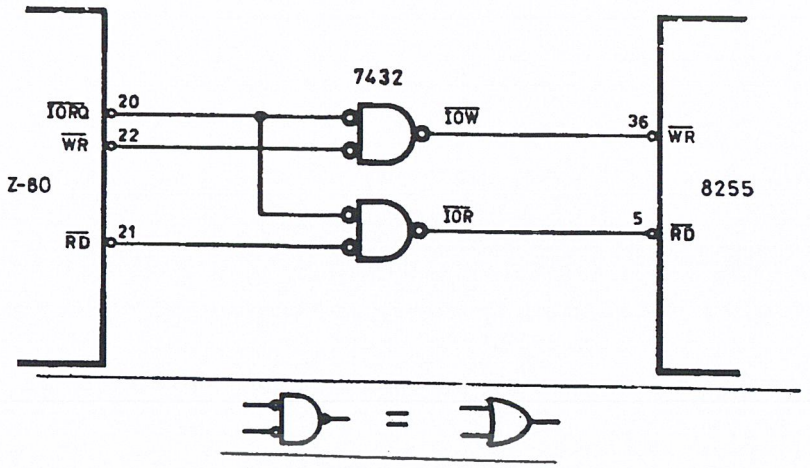
หากพิจารณาจากขาต่างๆของ 8255 จะเห็นว่า ส่วนขาควบคุมที่จะเชื่อมต่อเข้ากับบัสของไมโครโปรเซสเซอร์นั้นสามารถเชื่อมต่อกับบัสได้ง่าย ในที่นี้จะลองต่อ 8255 เป็นพอร์ตให้กับ Z-80 สมมติว่าต้องการให้ Z-80 มองเห็น 8255 เป็นพอร์ตหมายเลข 10H, 11H, 12H และ 13H การเชื่อมต่อสายสัญญาณการเลือกแอดเดรสของพอร์ตแสดงได้ดังรูปที่ 2.24



รูปที่ 2.24 การเชื่อมต่อสายสัญญาณการเลือกแอดเดรสของพอร์ต

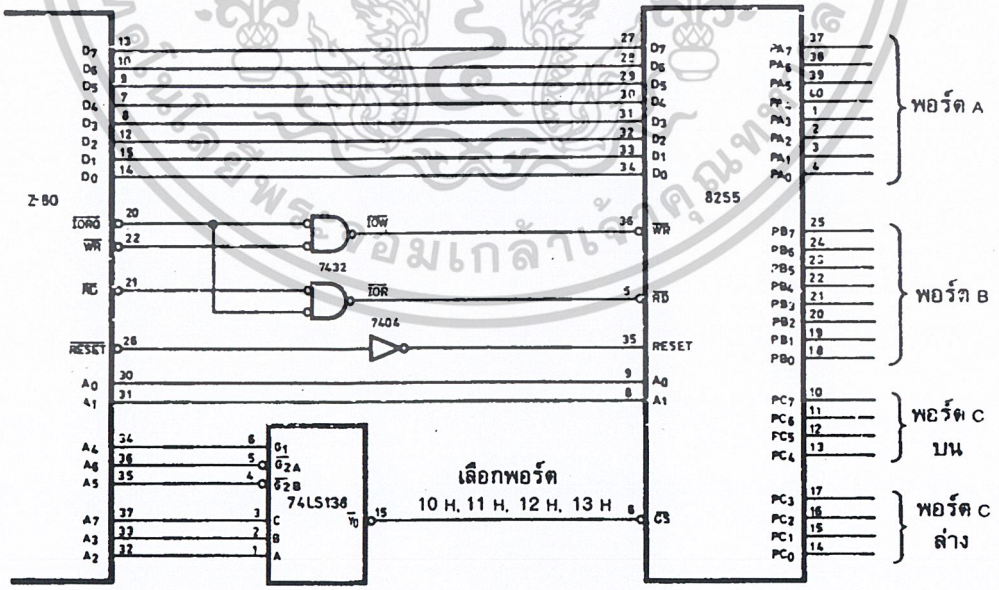
สังเกตว่า ขณะสัญญาณ CS แอคตีฟนั้น สัญญาณแอดเดรส A_7, A_6, A_5, A_4, A_3, A_2 จะต้องมีข้อมูล 000100 และเมื่อรวมกับ A_1, A_0 จะเป็น 000100XX พอร์ตที่เกิดขึ้นเมื่อ A_3, A_2 เป็น 00 คือ พอร์ต 10H และถ้า A_3, A_2 เป็น 11 พอร์ตจะเป็น 13H การกำหนดพอร์ตของ Z-80 จะใช้ข้อมูลบนบัสแอดเดรส 8 เส้น คือ A_3, A_2 เท่านั้น สัญญาณที่จะควบคุม 8255 อีกชุดหนึ่งคือ สัญญาณควบคุมการเขียนและการอ่าน หากสัญญาณ WR แอคตีฟเป็น "0" จะหมายถึง การเขียนพอร์ตหรือส่งข้อมูลให้พอร์ตเอาต์พุต แต่ถ้าสัญญาณ RD แอคตีฟเป็น "0" จะหมายถึง การอ่านพอร์ตหรือรับข้อมูลอินพุต

เพื่อให้แยกกันระหว่างการเขียนและการอ่านหน่วยความจำกับการเขียนและการอ่านพอร์ตอินพุต/เอาต์พุต จึงต้องใช้สัญญาณ IORQ ร่วมด้วย กล่าวคือ ถ้าสัญญาณ WR เกิดขึ้นพร้อมสัญญาณ IORQ จะหมายถึง สัญญาณ IOW หรือสัญญาณเขียนพอร์ต และถ้าให้สัญญาณ IORQ แอคตีฟพร้อมกับสัญญาณ RD จะหมายถึงสัญญาณ IOR หรือสัญญาณอ่านพอร์ต ซึ่งการเชื่อมต่อสายสัญญาณควบคุมการเขียนและการอ่านหน่วยความจำแสดงได้ดังรูปที่ 2.25



รูปที่ 2.25 การเชื่อมต่อสายสัญญาณควบคุมการเขียนและการอ่านหน่วยความจำ ไอซี 8255

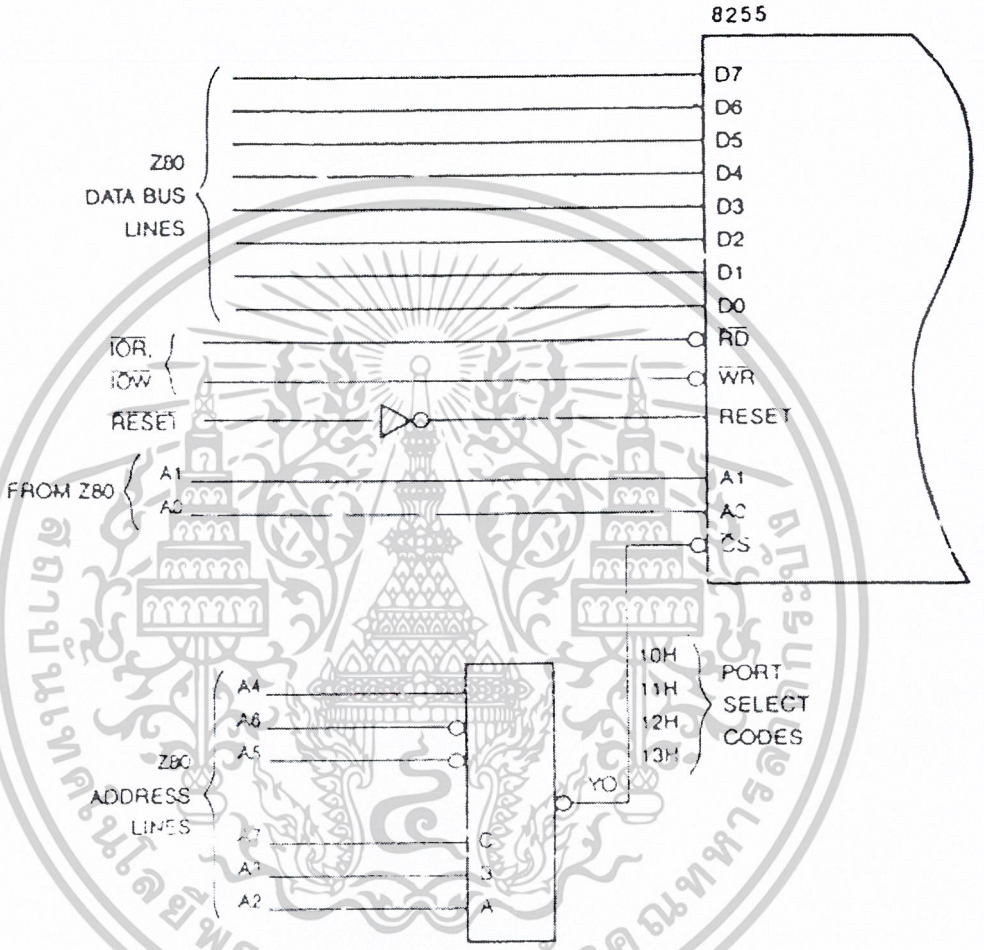
เมื่อเชื่อมต่อเป็นระบบ จะต้องมีการเชื่อมสายสัญญาณ RESET ของ Z-80 มายังขา RESET ของ 8255 การรีเซตของ 8255 ใช้ "1" ซึ่งตรงข้ามกับ Z-80 ดังนั้นจำเป็นต้องมีอินเวอร์เตอร์เปลี่ยนลอจิกก่อน การที่ต้องรีเซต 8255 พร้อมกับ Z-80 ก็เนื่องจากว่า ขณะที่ Z-80 รีเซต เราจะเริ่มจากให้พอร์ตทุกพอร์ตของ 8255 เป็นอินพุต เพื่อว่าอาจมีข้อมูลบ้างส่วนไปออกที่พอร์ตเอาต์พุตในขณะที่เรายังไม่ต้องการ ซึ่งอาจจะทำให้ระบบอินเตอร์เฟสภายนอกมีปัญหาได้ เพราะเราไม่รู้สถานะที่แน่นอนของ 8255 ก่อนการโปรแกรมโหมดการทำงาน ระบบการเชื่อมต่อของ 8255 กับ Z-80 ทั้งระบบแสดงได้ดังรูปที่ 2.26



รูปที่ 2.26 การเชื่อมต่อ ไอซี 8255 กับ Z-80 ทั้งระบบ

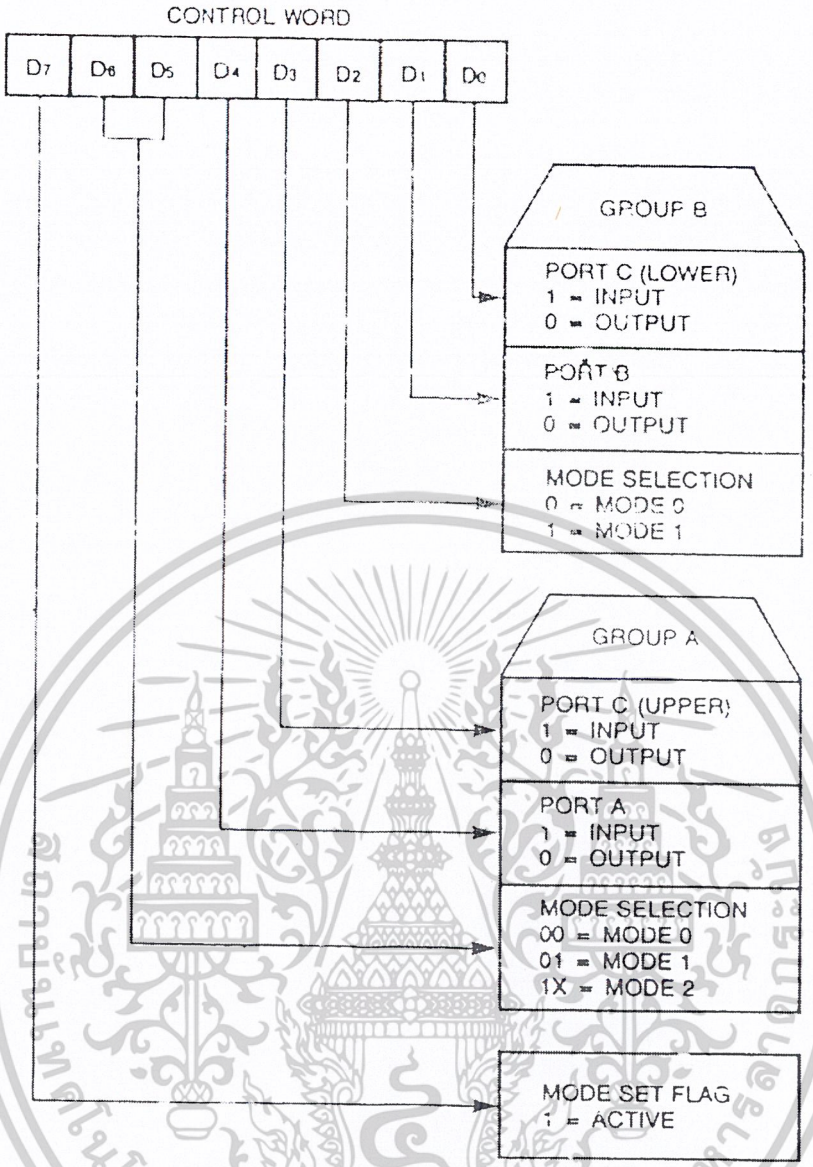
8255 รีค (REAK) และ ไรท์ รีจิสเตอร์ (WRITE REGISTER)

ขณะนี้เราได้ทำการต่อ 8255 เข้ากับระบบของ Z-80 แล้ว ต่อไปเราจะศึกษาการโปรแกรมใช้งาน 8255 เพื่อที่จะให้ทำงานตามที่เราต้องการได้ จะเริ่มต้นพิจารณาที่รีจิสเตอร์ภายใน 4 ตัวของ 8255 สำหรับในตัวอย่างการถอดรหัสของเรานี้ ตำแหน่งรีจิสเตอร์จะอยู่ที่แอดเดรส 10H , 11H , 12H และ 13H ซึ่งรายละเอียดของรีจิสเตอร์ เหล่านี้มีดังนี้คือ



รูปที่ 2.27 แสดงผังวงจรสมบูรณ์ของการเชื่อมต่อ 8255 เข้ากับระบบของ Z-80

หน้าที่ของรีจิสเตอร์หมายเลข 0-2 จะถูกกำหนดลักษณะการทำงานจากรีจิสเตอร์หมายเลข 3 (รีจิสเตอร์ควบคุม) รูปที่ 2.21 จะแสดงรายละเอียดของแต่ละบิตของรีจิสเตอร์ควบคุมนี้ ต่อไปเราจะกล่าวถึง ลักษณะการทำงานของ 8255 ทั้ง 3 โหมด และการโปรแกรมให้อยู่ในโหมดต่างๆ ได้ดังต่อไปนี้คือ



รูปที่ 2.28 แสดงรายละเอียดแต่ละบิตของรีจิสเตอร์ควบคุมของ 8255

การใช้งาน 8255 ในโหมด 0

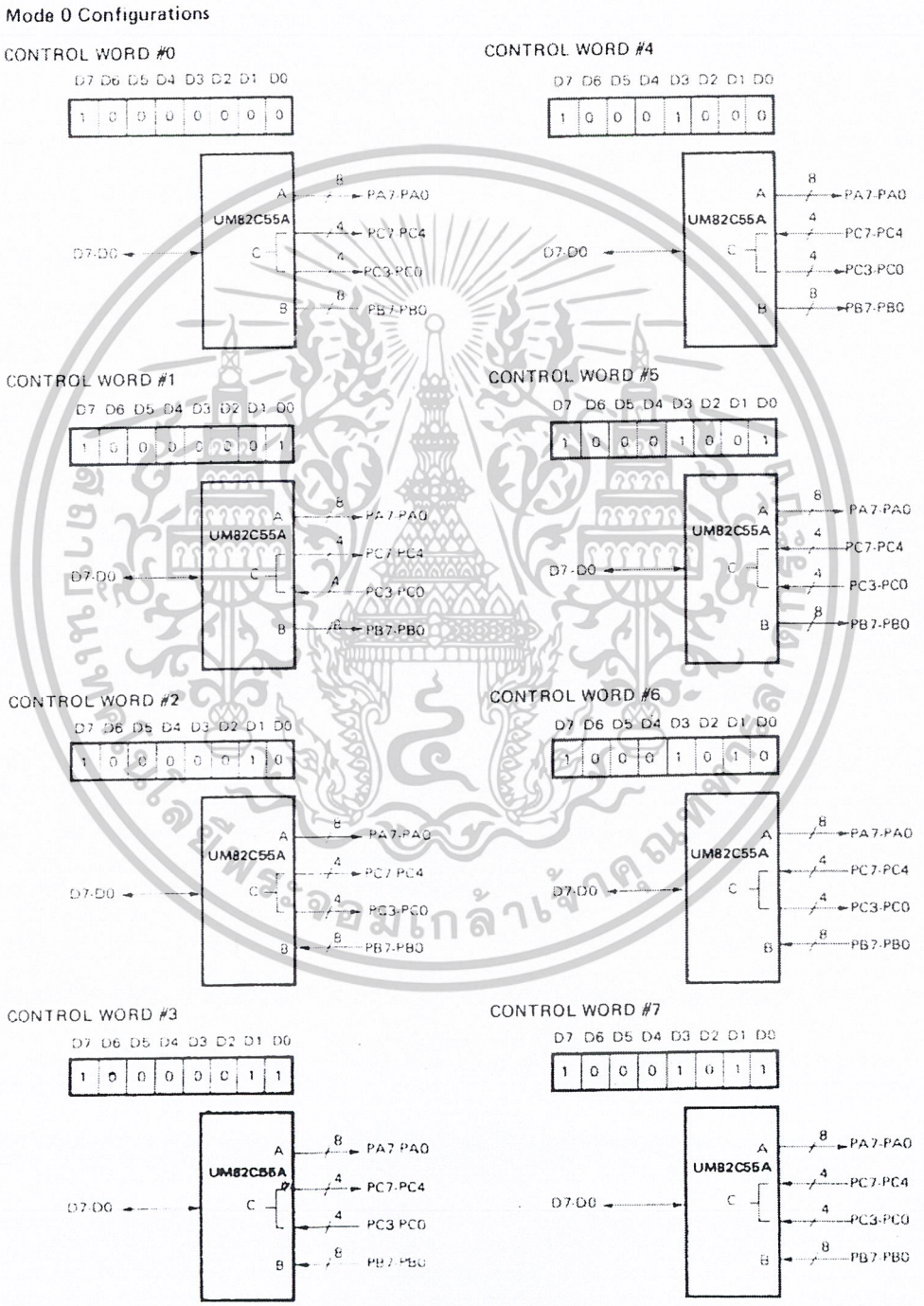
จากรูปที่ 2.21 เราจะเห็นว่า

- บิต D7 เป็นตัวกำหนดว่าเป็นคำสั่งควบคุม (CONTROL WORD)
- บิต D6 และ D5 กำหนดโหมดการทำงานของ พอร์ต A D6 , D5 มีค่าเป็น “0” แสดงว่า อยู่ในโหมด 0
- บิต D4 = “0” กำหนดให้ พอร์ต A เป็น พอร์ต เอาต์พุต
- บิต D3 = “0” เซตพอร์ต C 4 บิต บนเป็น พอร์ต เอาต์พุต
- บิต D2 = “0” เซตโหมดของพอร์ต B ให้พอร์ต B อยู่ในโหมด 0

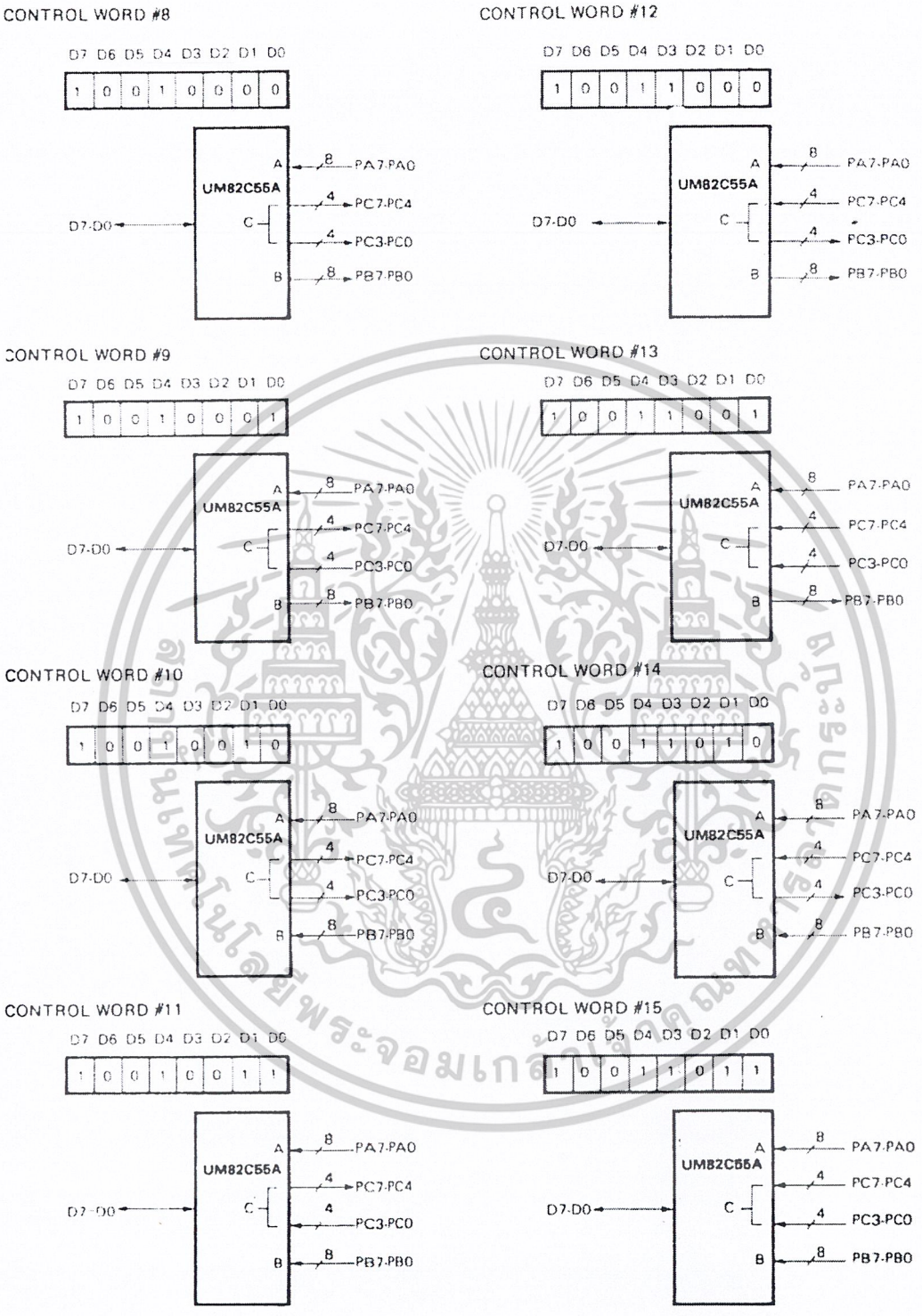
- บิต D1 = “0” เซตพอร์ต B เป็นพอร์ต เอาต์พุต
- บิต D0 = “0” เซตพอร์ต C ให้ 4 บิต ล่างเป็น พอร์ต เอาต์พุต

คำสั่งควบคุมนี้จะกำหนดให้ พอร์ตทั้ง 3 ของ 8255 ทำงานในโหมด 0 และเป็น พอร์ต เอาต์พุต ซึ่งจะได้สายสัญญาณซึ่งสามารถติดต่อกับอุปกรณ์ภายนอกได้ถึง 24 สาย

ลักษณะการทำงานของพอร์ตต่างๆ ที่สามารถกำหนดได้ในโหมด 0 แสดงไว้ในรูปที่ 2.29



รูปที่ 2.29 แสดงลักษณะต่างๆ ในการใช้งาน 8255 ในโหมด 0

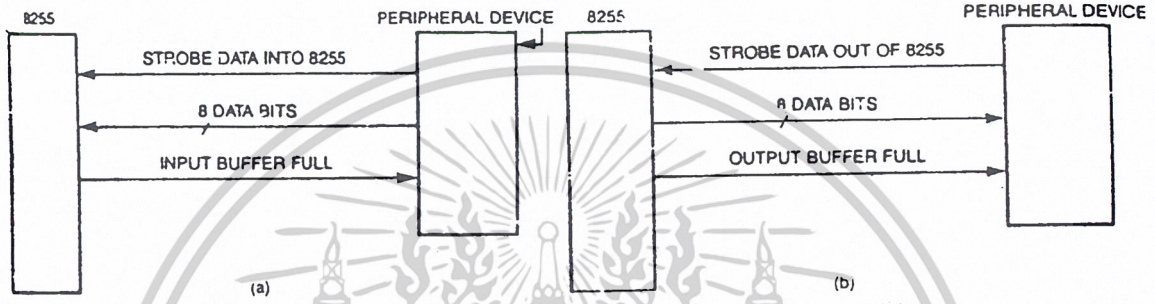


รูปที่ 2.29 (ต่อ) แสดงลักษณะต่างๆ ในการใช้งาน 8255 ในโหมด 0

การใช้งาน 8255 ในโหมด 1

การทำงานของ 8255 ในโหมด 1 นี้เป็นการทำงานในลักษณะของการ HANDSHAKE , พอร์ต A และพอร์ต B จะเป็นพอร์ตข้อมูล ส่วนพอร์ต C นี้จะถูกใช้เป็นสัญญาณ HANDSHAKE โดย 4 บิตบนจะเป็นสัญญาณ HANDSHAKE ให้กับพอร์ต A และ 4 บิตล่างจะเป็นสัญญาณ HANDSHAKE ให้กับพอร์ต B

หลักการรับส่งข้อมูลในวิธีการของ HANDSHAKE นี้ คือ การให้อุปกรณ์ส่งสัญญาณแสดงสภาวะความพร้อมให้กับ 8255 ดังแสดงในรูปที่ 2.31



รูปที่ 2.30 บล็อกไดอะแกรมแสดงลักษณะการทำงานของการติดต่อระหว่าง 8255 กับอุปกรณ์ภายนอก

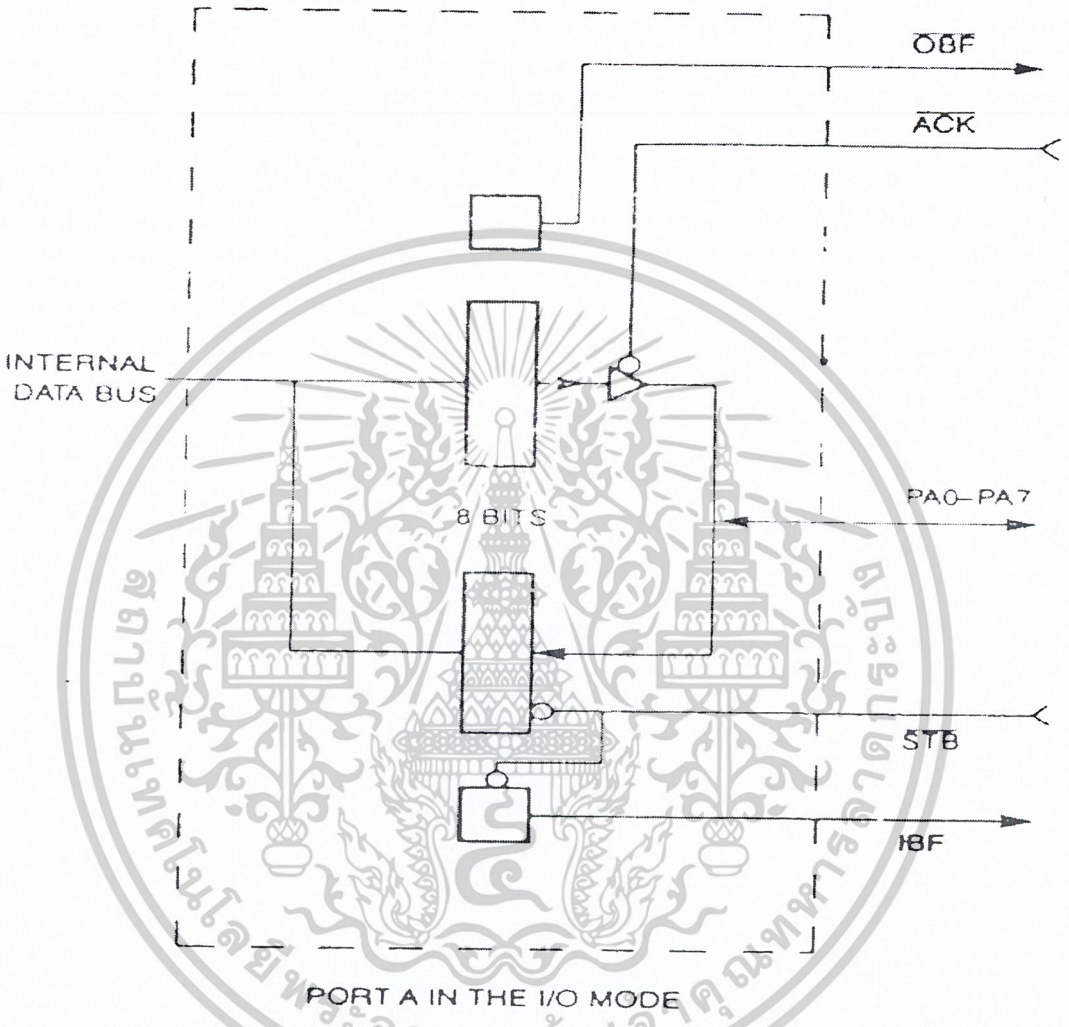
ในรูปที่ 2.31 (a) นี้ข้อมูลจะถูกส่งออกจากอุปกรณ์ภายนอกเข้าสู่ 8255 ก่อนที่อุปกรณ์ภายนอกจะเขียนข้อมูลให้แก่ 8255 จะต้องมีการตรวจสอบ อินพุตบัฟเฟอร์ ฟูลแฟค (INPUT BUFFER FULL FLAG) เสียก่อน ถ้า FLAG นี้ เป็นจริงแสดงว่าข้อมูลในบัฟเฟอร์ 8255 นี้ยังไม่ถูกอ่านโดย Z-80 แต่ถ้า FLAG เป็นเท็จแสดงว่า Z-80 อ่านข้อมูลออกไปแล้ว อุปกรณ์ภายนอกก็จะเขียนข้อมูลใหม่ให้ 8255 ได้

ในรูปที่ 2.31 (b) จะทำหน้าที่เป็นตัวส่งข้อมูลให้อุปกรณ์ภายนอก ก่อนที่ 8255 จะส่งข้อมูลให้อุปกรณ์ภายนอกนั้นจะต้องเซต OUTPUT BUFFER FULL FLAG เสียก่อน เพื่อบอกให้อุปกรณ์ภายนอกทราบว่า ขณะนี้ 8255 มีข้อมูลพร้อมที่จะส่งออกไปแล้ว เมื่ออุปกรณ์ภายนอกส่งสัญญาณ Strobe รับเอาข้อมูลเข้าไปแล้ว INPUT BUFFER FULL FLAG จะเปลี่ยนเป็นเท็จ เพื่อบอกให้อุปกรณ์ภายนอกรู้ว่าขณะนี้ไม่มีข้อมูลอยู่ใน 8255 Z-80 สามารถส่งข้อมูลใหม่ออกไปให้ 8255 ได้

วิธีการทำ HANDSHAKE นี้ มีประโยชน์มากในกรณีที่อุปกรณ์ภายนอกทำงานช้ากว่าระบบไมโครโปรเซสเซอร์ ด้วยวิธีนี้ไมโครโปรเซสเซอร์สามารถที่จะส่งข้อมูลให้กับ 8255 แล้วไปทำงานอื่นได้ จนกว่าข้อมูลภายใน 8255 ถูกส่งออกไปแล้ว Z-80 ไมโครโปรเซสเซอร์จึงจะส่งข้อมูลใหม่ออกไปให้

การใช้งาน 8255 ในโหมด 2

การทำงานของ 8255 ในโหมด 2 นี้เป็นการใช้งานในลักษณะที่ให้ พอร์ต A เป็น พอร์ตข้อมูลแบบ 2 ทิศทาง เมื่อ 8255 ถูกโปรแกรมให้ พอร์ต A อยู่ในโหมด 2 นี้ แล้วพอร์ต A จะมีลักษณะการทำงานตามบล็อกไดอะแกรมรูปที่ 2.32



รูปที่ 2.31 บล็อกไดอะแกรมแสดงการทำงานของพอร์ต A ในโหมด 2

การทำงานในโหมดนี้คือ การใช้พอร์ต A เป็นอินพุตและเอาต์พุตแลทช์ (OUTPUT LATCH) หมายถึง การเก็บข้อมูลไว้เพื่อรออุปกรณ์ภายนอกรับเอาข้อมูลออกไป ส่วนอินพุตแลทช์ (INPUT LATCH) หมายถึง การเก็บข้อมูลที่อุปกรณ์ภายนอกส่งเข้ามาเพื่อรอให้ CPU อ่านเข้าไป

บทที่ 3

การออกแบบการทดลอง

3.1 กล่าวนำ

ในการทดลองนี้ได้แยกการทดลองออกเป็น 2 ส่วน คือ ส่วนทางด้านฮาร์ดแวร์ และส่วนทางด้านซอฟท์แวร์ เพื่อทำการทดลองเกี่ยวกับการติดต่อระหว่างเครื่องคอมพิวเตอร์ การ์ด(Card) อินเตอร์เฟส ET-PC8255 และ บอร์ด Z-80 Controller สำหรับรับและส่งข้อมูลและแสดงผลการทดลอง ซึ่งทางด้านฮาร์ดแวร์ จะกล่าวถึงการสร้างวงจรบัฟเฟอร์ (Buffer) ของขาสัญญาณต่างๆ และการใช้งาน การ์ดอินเตอร์เฟส ET-PC8255 ส่วนทางด้านซอฟแวร์ จะใช้โปรแกรมภาษา Borland C++ Builder ในการทดลอง

3.2 ทางด้านฮาร์ดแวร์ (Hardware)

ในการออกแบบฮาร์ดแวร์ เพื่อทำการทดลอง โดยได้จัดทำวงจรบัฟเฟอร์ ของขาสัญญาณต่างๆ ดังรูปที่ 3.2

ในส่วนของวงจรเลือกใช้ไอซี ที่ทำหน้าที่บัฟเฟอร์ คือ เบอร์ 74LS365A ทำหน้าที่บัฟเฟอร์ขาสัญญาณต่างๆ ทั้งสัญญาณ อินพุต และเอาต์พุต การต่อใช้งานไอซี เบอร์ 74LS365A มีคุณสมบัติการต่อใช้งานตามรูปที่ 3.2

FUNCTION TABLE, '365A, '366A

INPUTS			OUTPUTS	
$\overline{OE}_1$	$\overline{OE}_2$	I	Y	$\overline{Y}$
L	L	L	L	H
L	L	H	H	L
X	H	X	(Z)	(Z)
H	X	X	(Z)	(Z)

FUNCTION TABLE, '367A, '368A

INPUTS		OUTPUTS	
$\overline{OE}$	I	Y	$\overline{Y}$
L	L	L	H
L	H	H	L
H	X	(Z)	(Z)

L = LOW voltage level
H = HIGH voltage level
X = Don't care
(Z) = HIGH impedance (off) state

TYPE	TYPICAL PROPAGATION DELAY	TYPICAL SUPPLY CURRENT (TOTAL)
74365A, 367A	10ns	65mA
74LS365A, 367A	10ns	14mA
74366A, 368A	9ns	59mA
74LS366A, 368A	10ns	12mA

ORDERING CODE

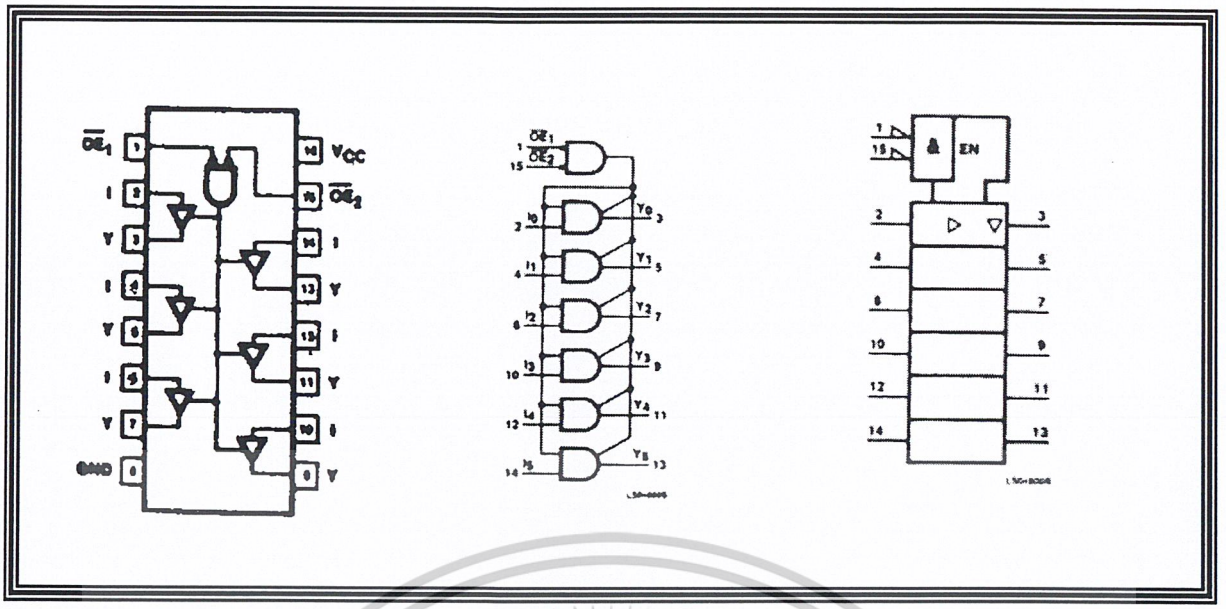
PACKAGES	COMMERCIAL RANGE V_{CC} = 5V ±5%; T_A = 0°C to +70°C
Plastic DIP	N74365AN, N74LS365AN, N74366AN, N74LS366AN N74367AN, N74LS367AN, N74368AN, N74LS368AN
Plastic SO-16	N74LS365AD, N74LS367AD, N74LS368AD

NOTE:
For information regarding devices processed to Military Specifications, see the Signetics Military Products Data Manual

INPUT AND OUTPUT LOADING AND FAN-OUT TABLE

PINS	DESCRIPTION	74	74LS
All	Inputs	1ul	1LSul
All	Outputs	20ul	30LSul

รูปที่ 3.1 ขาสัญญาณของ ไอซี เบอร์ 74LS365A

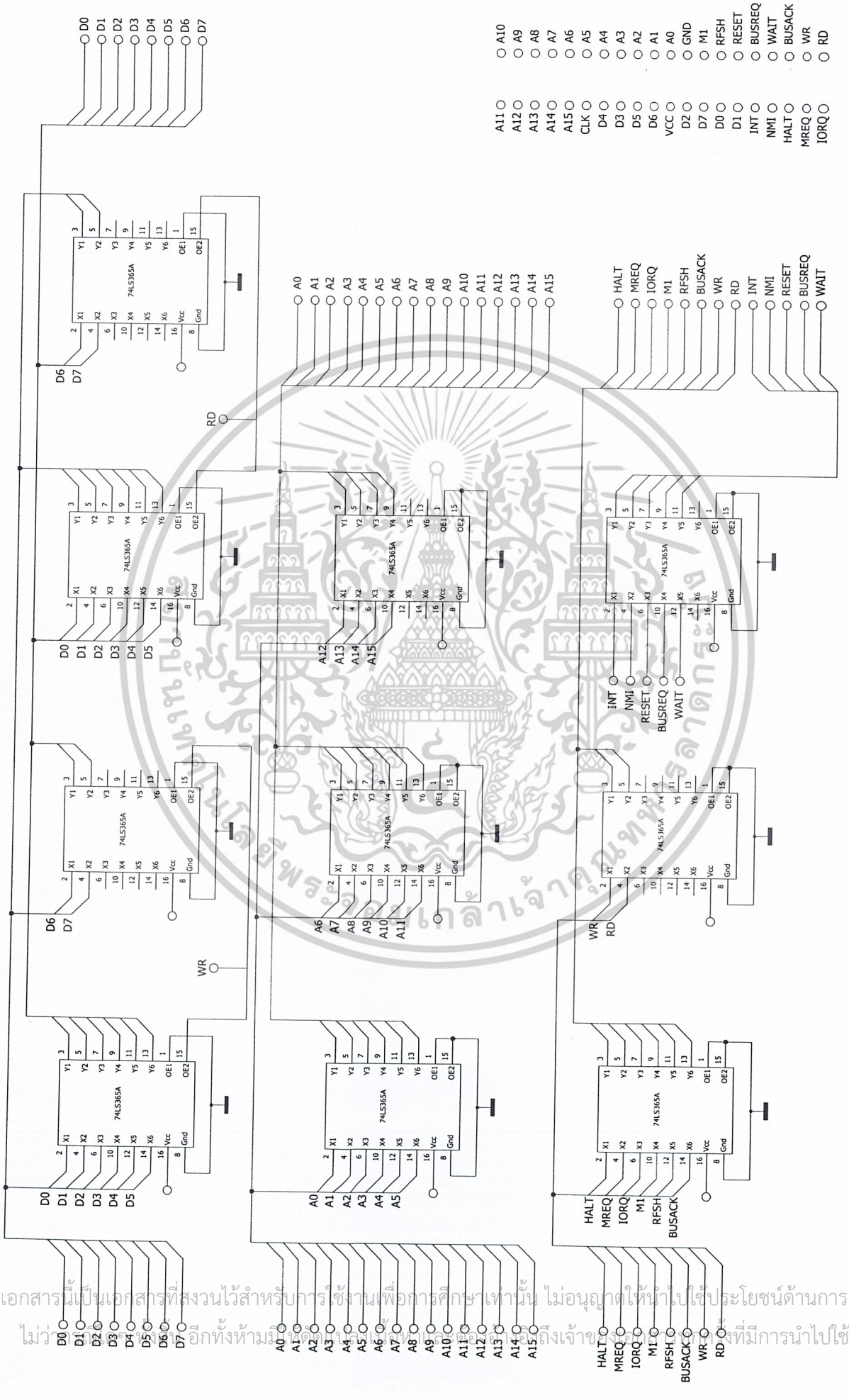


รูปที่ 3.1 (ต่อ) ขาสัญญาณของ ไอซี เบอร์ 74LS365A

จากรูปที่ 3.1 จะให้ ลอจิก “0” กับ OE1 และ OE2 ซึ่งทั้งสองขานี้ Active low ขาเอาต์พุตจะได้ผลลัพท์ เหมือนขาอินพุต แต่ถ้าขาใดขาหนึ่งเป็น “1” เอาต์พุตจะเป็น High Impedance ไม่ว่า อินพุตจะเป็นลอจิก “0” หรือ“1”

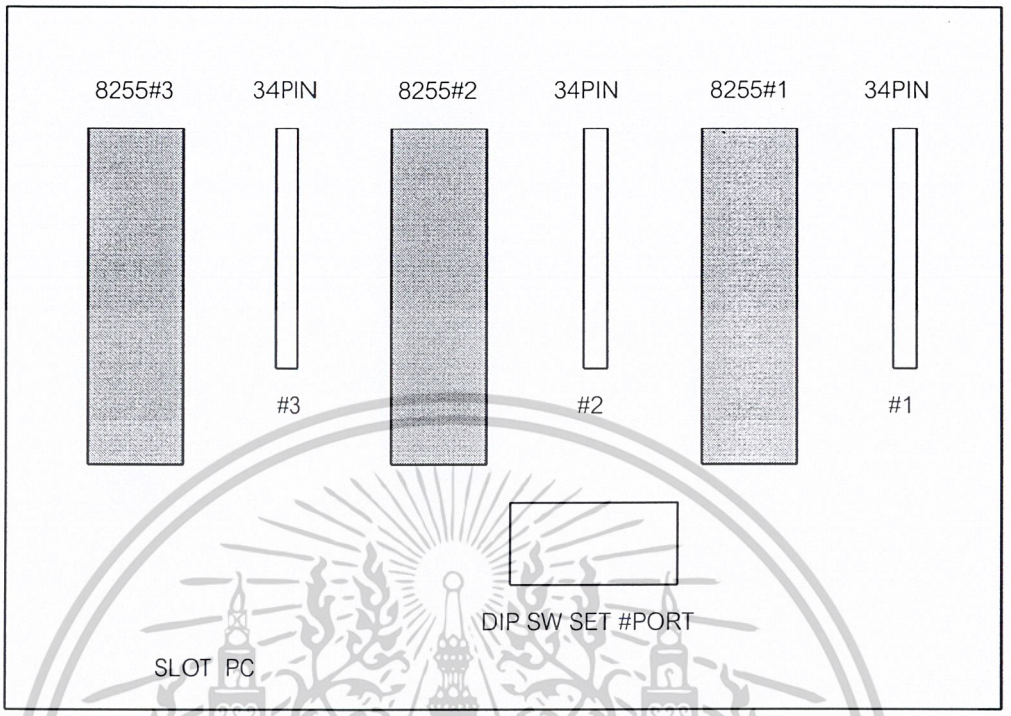
เมื่อนำมาต่อใช้งานจะได้วงจรดังรูปที่ 3.2 ในการต่อใช้งานนั้นเนื่องจากขา D0-D7 เป็นขาสัญญาณที่ข้อมูลสามารถวิ่งได้ทั้ง 2 ทิศทาง คือเป็นทั้งอินพุตและเอาต์พุต สัญญาณที่จะเลือกว่าเป็นอินพุต หรือเอาต์พุต ก็คือสัญญาณ WR และ RD ดังนั้น จึงนำขา OE1 ของไอซี 1 และ ไอซี 2 มาต่อกับขา WR และ ขา OE2 ของ ไอซี 3 และ ไอซี 4 มาต่อกับขา RD เพราะลักษณะการทำงานของทั้งสองขานี้ เมื่อขา WR แอคตีฟ ซึ่งเป็นการ Active low ทำให้ ไอซี 1 และ ไอซี 2 ทำงานตามคุณสมบัติ ขณะเดียวกัน ขา RD จะเป็น High ทำให้ ไอซี 3 และ ไอซี 4 เป็น High Impedance และตัดตัวเองออกจากระบบบัส ตามคุณสมบัติ และเช่นกัน เมื่อ ขา RD Active low ขา WR ก็จะเป็น High ลักษณะ การทำงานก็จะทำงานสลับกับสภาวะการทำงานของสภาวะ WR

ส่วนขาสัญญาณอื่นๆ นั้น มีลักษณะ ในทิศทางเดียวกันทั้งหมด กล่าวคือ จะมีขา เอาต์พุต 24 ขา ได้แก่ A0-A15 , HALT , MREQ , IORQ , RFSH , M1 , BUSAK , WR RD และมีขา อินพุต อีก 5 ขา ได้แก่ INT , NMI , RESET , BUSRQ , WAIT การต่อใช้งานจะต่อตามวงจรในรูปที่ 3.2



3.3 CADR ET-PC8255

ลักษณะของ ET-PC 8255



รูปที่ 3.3 แสดงลักษณะของ CARD ET-PC 8255

ET-PC 8255 จะเป็น CARD ต่อขยายระบบเครื่อง PC ให้มีส่วนของพอร์ต อินพุต/เอาต์พุต ใช้งานมากขึ้น โดยจะมีพอร์ตใช้งานจำนวน 9 พอร์ต หรือ 72 บิต I/O (TTL 0-5 V)

การทำงานของ ET-PC 8255

CARD ET-PC 8255 จะประกอบไปด้วย 2 ส่วนใหญ่ๆก็คือ ส่วนไอซี 8255 ซึ่งเป็น ไอซี ทำหน้าที่เป็นพอร์ตอินพุต/เอาต์พุต และส่วนของวงจรไอซีดีโค้ด (เลือกตำแหน่งของพอร์ต 8255) คือ ไอซี 74LS688 , 74LS139 และ DIP SW.

การ DECODE PORT

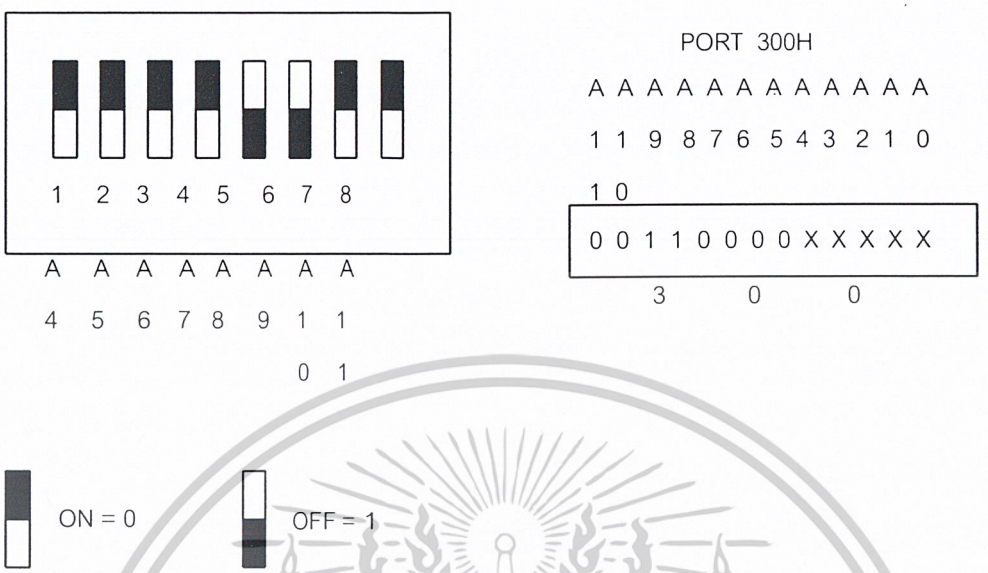
โดย CARD ET-PC 8255 จะใช้ตำแหน่ง PORT 12 PORT ต่อ CARD DECODE PORT เราจะได้ตำแหน่ง PORT ทั้ง 12 port ดังนี้

ตารางการกำหนดตำแหน่งของพอร์ตสำหรับ Card I/O 8255 ที่มี 3 Connector

Connector No.1	Connector No.2	Connector No.3
PortA=XX0H	PortA=XX4H	PortA=XX8H
PortB=XX1H	PortB=XX5H	PortB=XX9H
PortC=XX2H	PortC=XX6H	PortC=XXAH
Control Port=XX3H	Control Port=XX7H	Control Port=XXBH

รูปที่ 3.4 แสดงตำแหน่งของ PORT

สามารถตั้งเบอร์ DECODE PORT ได้โดยการปรับ DIP SW. ซึ่งก็มีค่าเท่ากับค่า ADDRESS นั้นๆ การออกแบบการทดลองครั้งนี้จะตั้งที่ตำแหน่ง 300H จะ SET DIP SW. ดังนี้



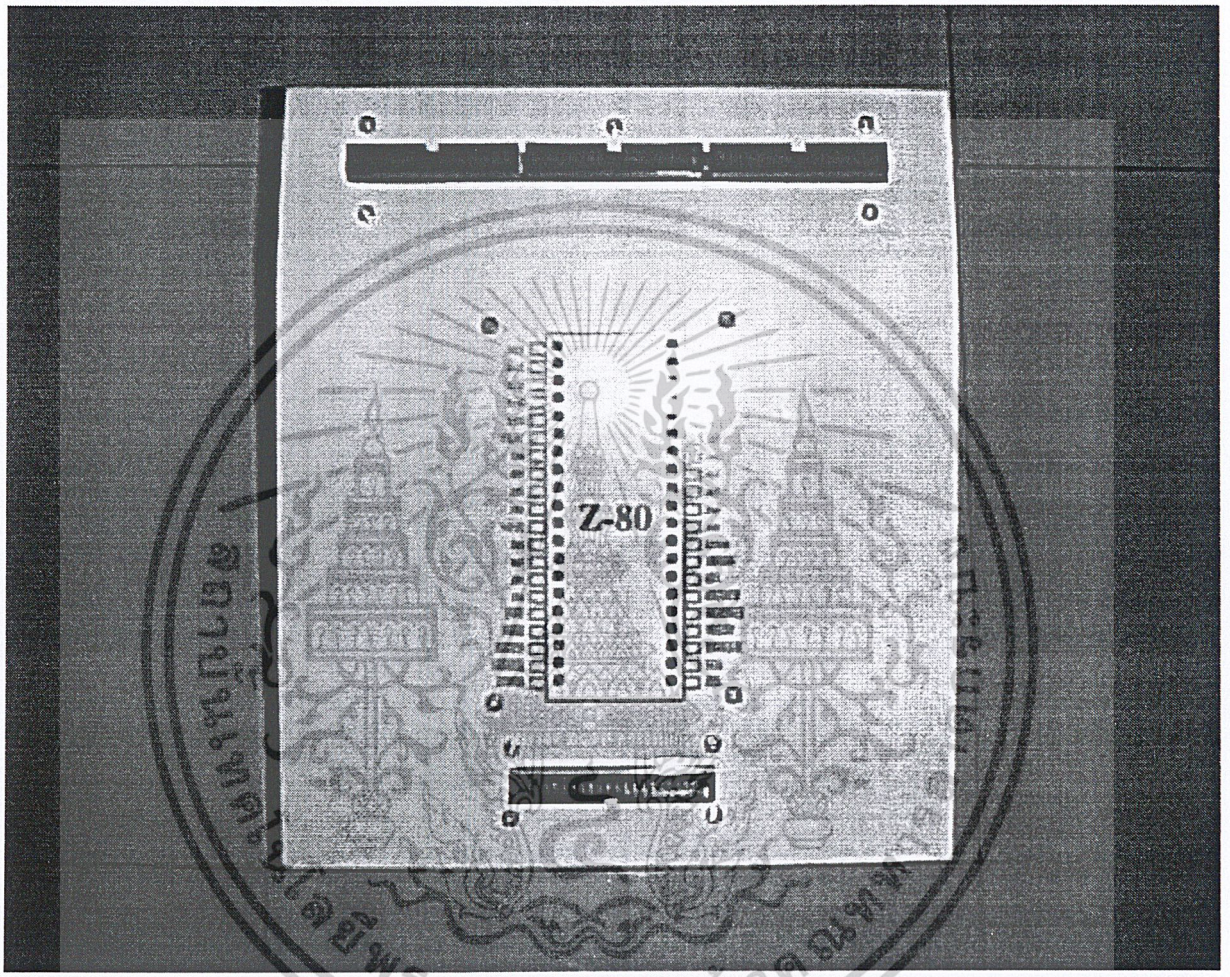
รูปที่ 3.5 แสดงการตั้งตำแหน่งพอร์ตโดย Dip SW.

การอินเตอร์เฟสระหว่างคอมพิวเตอร์กับบอร์ด Z-80 Controller ผ่าน CARD ET-PC 8255 นั้น มีการกำหนดขาสัญญาณดังนี้ D0-D7 กำหนดไว้ที่ PORT A ของ 8255 ตัวที่ 1 A0-A15 กำหนดไว้ที่ PORT A และPORT B ของ 8255 ตัวที่ 2 และขา Control Bus ต่างๆ กำหนดขาที่เป็น เอาต์พุต ไว้ที่ PORT A และขาที่เป็นอินพุต ไว้ที่ PORT B ของ 8255 ตัวที่ 3

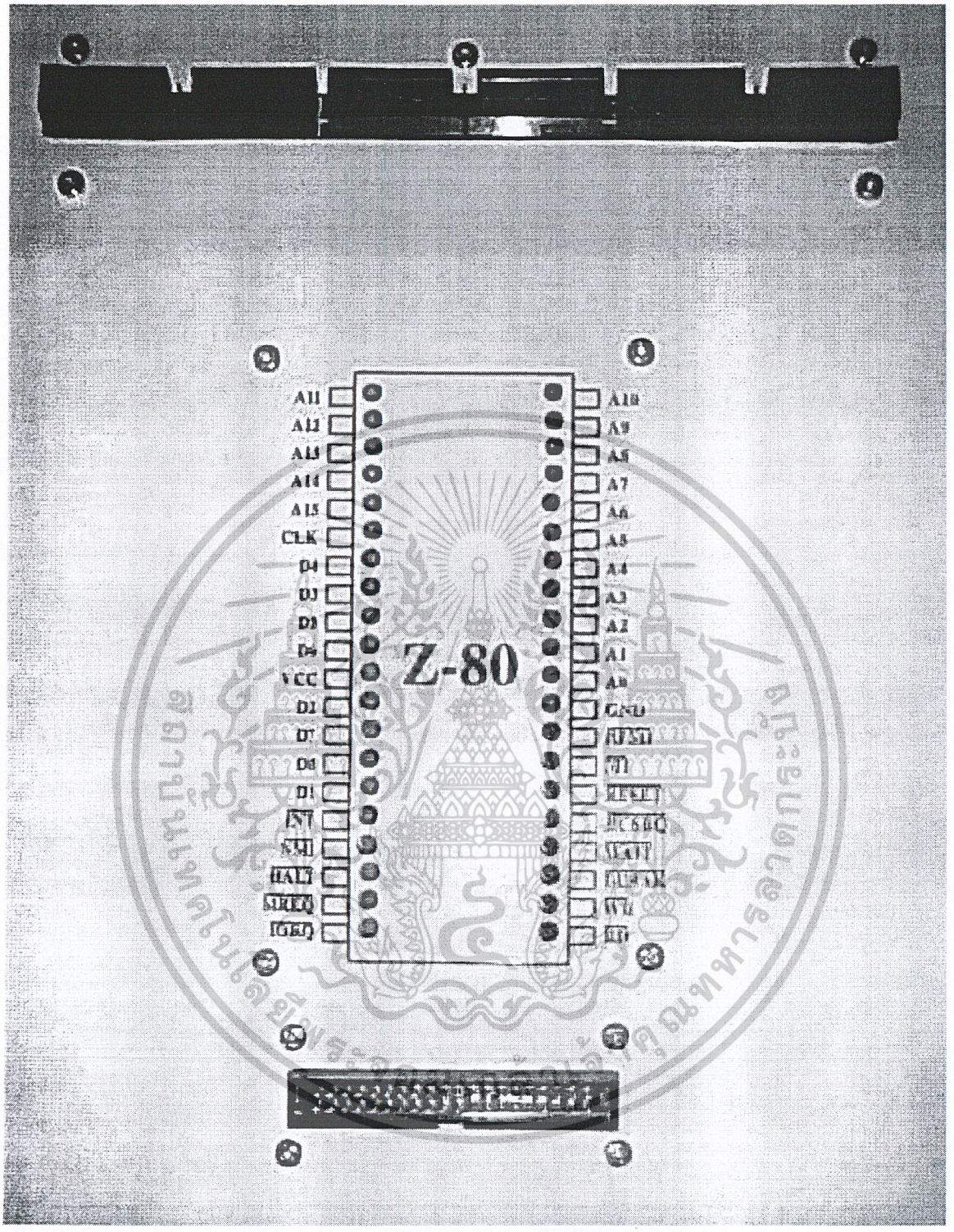
SPECIFICATIONS

LOGIC INPUT AND OUTPUT	: MIN	MAX	
INPUT LOGIT LOW	: -0.5	0.8	VOLTS
INPUT LOGIT HIGH	: 2.0	5	VOLTS
OUTPUT LOW VOLTAGE PORTS	: -	0.45	VOLTS
(I-SINK = 1.7 MA)			
OUTPUT LOW VOLTAGE PORTS	: 2.4	-	VOLTS
(I-SINK = 200 UA)			
POWER CONSUMPTION	: 300 MA +5 V		
SIZE	: HALF SLOT (13 X 11 CM)		
PORT	: 3 (8255 I/O PORT 3 X 8 BIT)		
CONNECTOR	: 3 (34 PIN HEADER-STRIP ETT IO BUS)		
DECODE PORT	: 8 POSITION DIP SW.		

3.4 แสดงรูปสำเร็จของอุปกรณ์ที่ใช้สำหรับการทดลอง



รูปที่ 3.6 แสดงอุปกรณ์ที่ใช้สำหรับการทดลอง



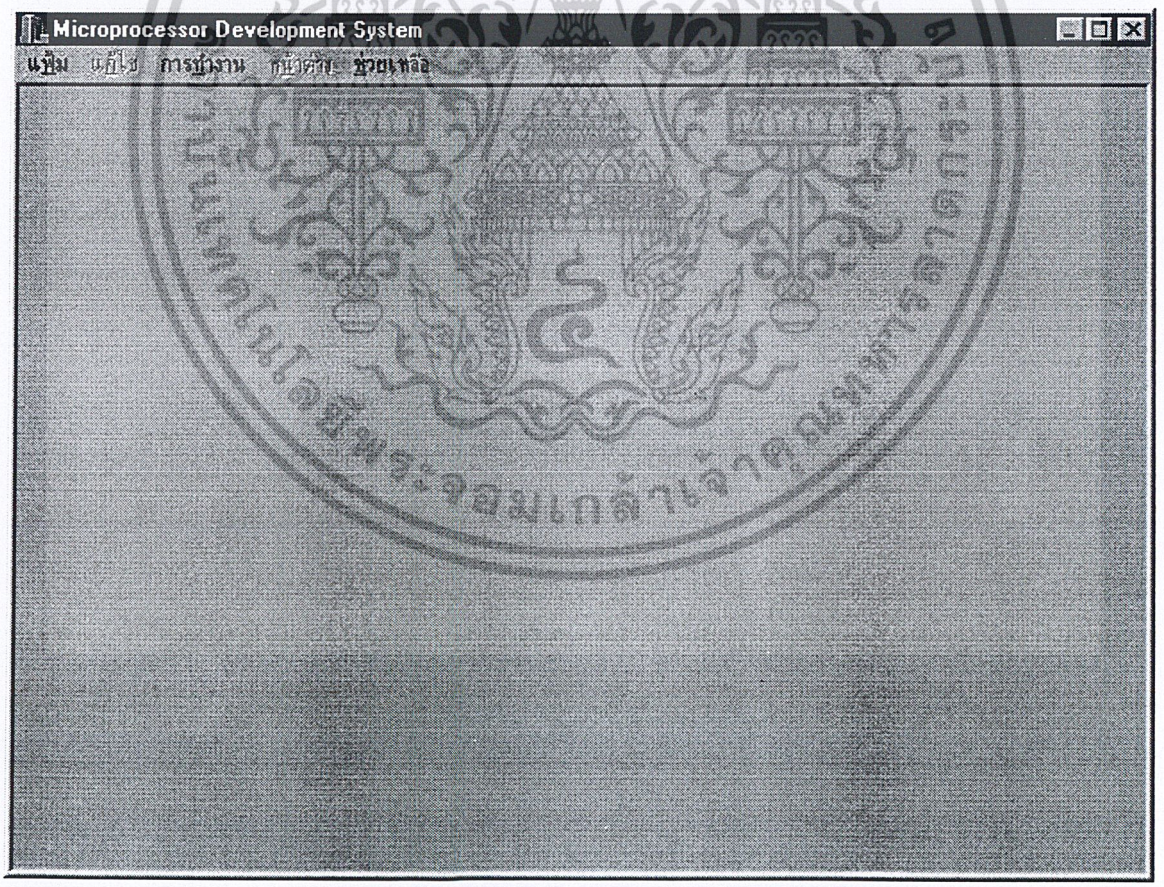
รูปที่ 3.6 (ต่อ) แสดงอุปกรณ์ที่ใช้สำหรับการทดลอง

3.5 ส่วนของซอฟท์แวร์ (Software)

ในการทดลองเลือกใช้งานโปรแกรม Borland C++ Builder เพราะมีแนวคิดว่าในคอมพิวเตอร์ในปัจจุบัน มีระบบปฏิบัติการส่วนใหญ่เป็นในลักษณะวินโดว์ และเป็นที่ยอมรับกันว่า วินโดว์มีประสิทธิภาพในการทำงานสูงกว่าดอส ดังนั้นในคอมพิวเตอร์ที่ใช้วินโดว์ ถ้าเราเขียนโปรแกรมสำหรับดอสเพื่อรันในวินโดว์ก็เท่ากับว่าเราไม่ได้ใช้ความสามารถของวินโดว์ที่มีอยู่เลย เพราะโปรแกรมจะลดความสามารถของวินโดว์ให้เหลือเท่ากับคอมพิวเตอร์ที่ใช้ดอสเท่านั้น

สำหรับคอมพิวเตอร์ที่ใช้วินโดว์ในการเขียนโปรแกรมเพื่อรันเพื่อรันในวินโดว์ จะทำให้เกิดประโยชน์สูงสุด เพราะโปรแกรมสำหรับวินโดว์จะนำความสามารถของวินโดว์มาใช้อย่างเต็มที่ และการเขียนโปรแกรมบนวินโดว์ทำได้ง่ายกว่าการเขียนโปรแกรมสำหรับดอส

Borland C++ Builder เป็นการเขียนโปรแกรมแบบวิชวล (การเขียนโปรแกรมแบบวิชวลเป็นการเขียนโปรแกรมด้วยส่วนประกอบต่างๆที่เรามองเห็นทันทีในขณะที่เรากำลังเขียนโปรแกรม) โดยการนำส่วนประกอบต่างๆที่ต้องใช้มาวางในวินโดว์ของโปรแกรม Borland C++ Builder จะสร้างซอร์สโค้ดพื้นฐานให้ ต่อจากนั้นเราจึงเขียนสเตตเมนท์เพิ่มเติมให้เข้ากับซอร์สโค้ดที่มีอยู่เพื่อให้โปรแกรมสามารถทำงานตามที่เราต้องการได้



รูปที่ 3.7 แสดงหน้าจอโปรแกรมที่เขียนขึ้นโดยโปรแกรม Borland C++ Builder

บทที่ 4

การทดลองและผลการทดลอง

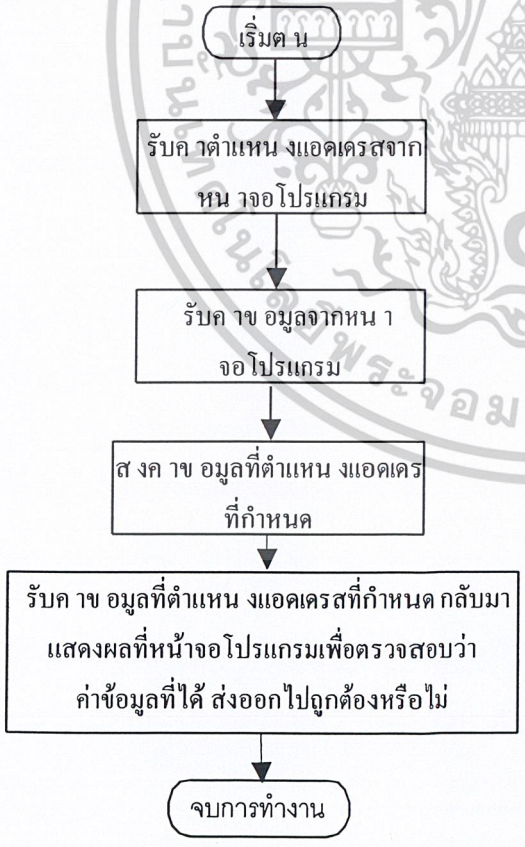
การทดลองสามารถทำการติดต่อกับ บอร์ด Z-80 Controller ได้โดยทำการอินเตอร์เฟสระหว่างคอมพิวเตอร์ กับ บอร์ดทดลอง ผ่าน Card ET-PC8255 โดยมีการเขียนโปรแกรมด้วยภาษา Borland C++ Builder ให้แสดงผลการทดลองตามที่ต้องการ โดยแบ่งการทดลองออกเป็น 4 การทดลอง ซึ่งได้ผลการทดลอง คือ

4.1 การทดลองที่ 1 การรับและส่งข้อมูล

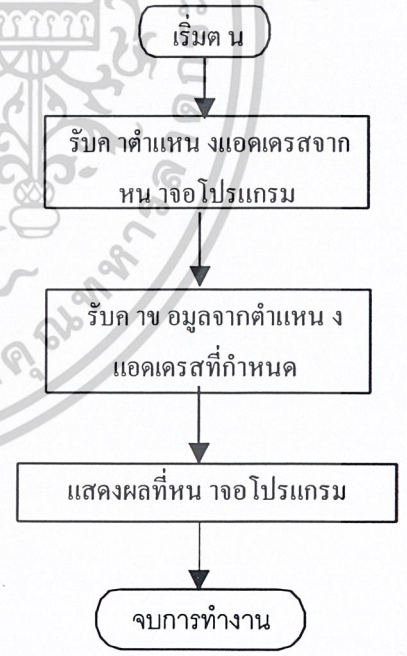
การทดลองนี้จะเป็นการรับและส่งข้อมูลไปเก็บไว้ที่หน่วยความจำบนบอร์ดZ-80 Controller กล่าวคือ จะกำหนดตำแหน่งแอดเดรส และระบุค่าข้อมูล แล้วส่งข้อมูลออกไป ในการรับข้อมูลกลับมาแสดงนั้น จะมี 2 รูปแบบ คือ

- กำหนดค่าตำแหน่งแอดเดรส แล้วรับค่าข้อมูลนั้นมาแสดง
- กำหนดแอดเดรสเป็นช่วงของหน่วยความจำ แล้วจึงรับค่าข้อมูลที่ถูกกำหนดเป็นช่วงตำแหน่งแอดเดรสมาแสดง

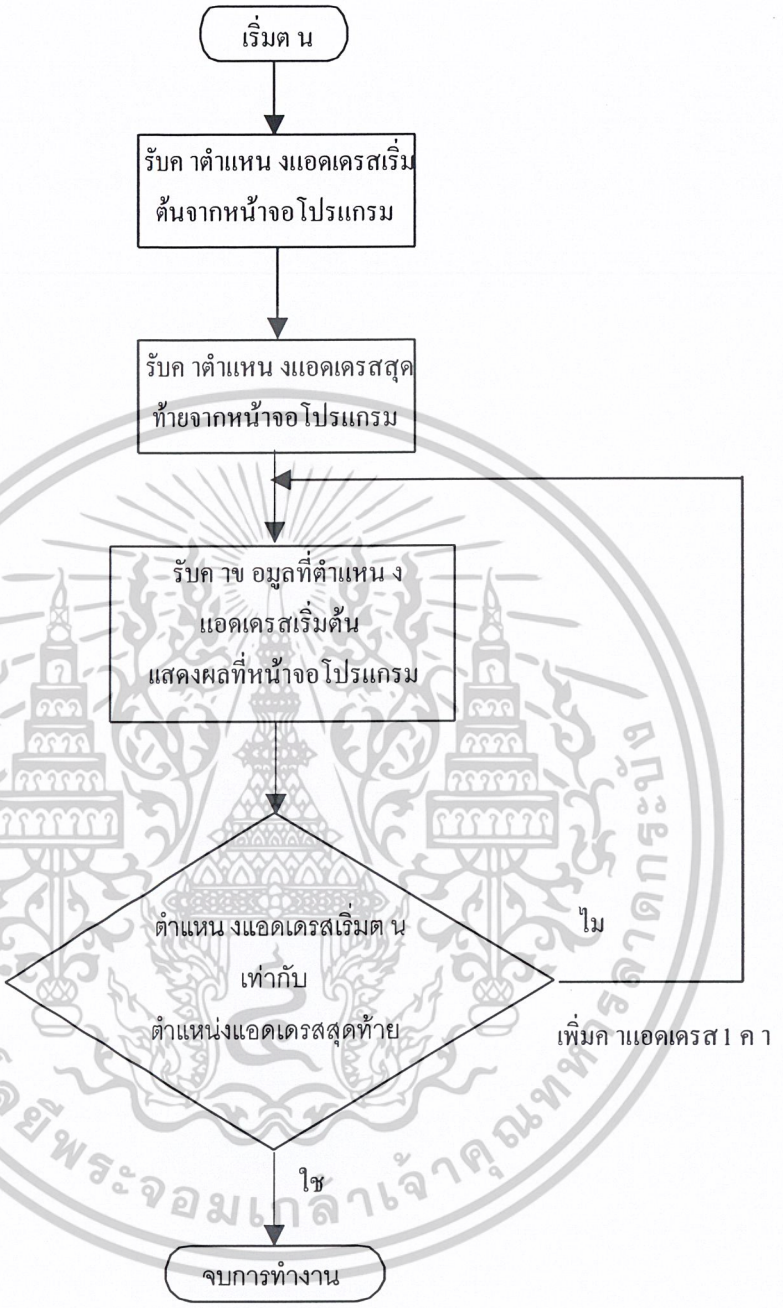
โดยมีขั้นตอนการทำงานดังนี้



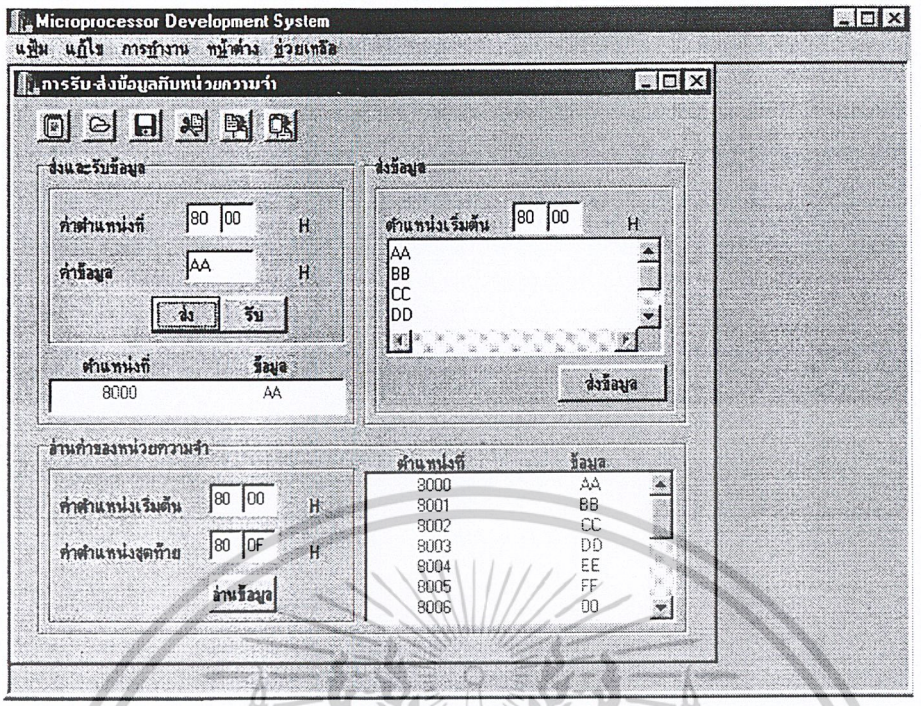
รูปที่ 4.1 แสดงการส่งข้อมูลในตำแหน่งที่กำหนด



รูปที่ 4.2 แสดงการรับข้อมูลในตำแหน่งที่กำหนด



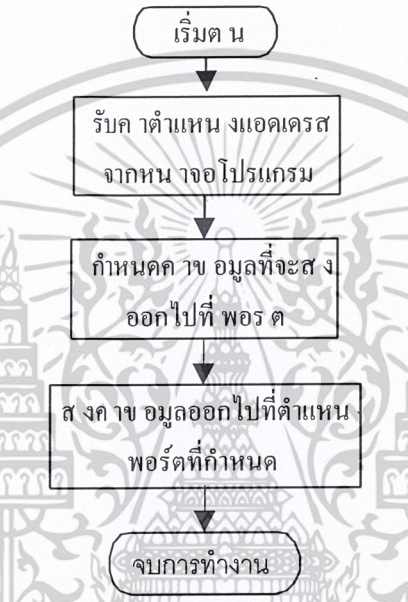
รูปที่ 4.3 แสดงการรับข้อมูลเป็นช่วงของตำแหน่งแอดเดรส



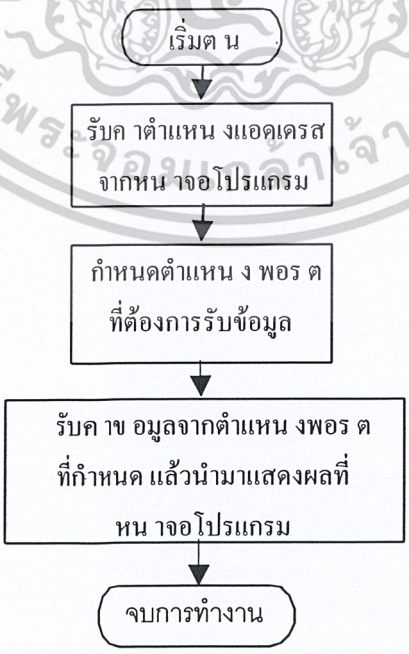
รูปที่ 4.4 แสดงหน้าจอโปรแกรมการรับและส่งข้อมูลไปเก็บไว้ที่หน่วยความจำ

4.2 การทดลองที่ 2 การรับส่งข้อมูลผ่าน พอร์ต I / O ของ บอร์ด Z-80 Controller

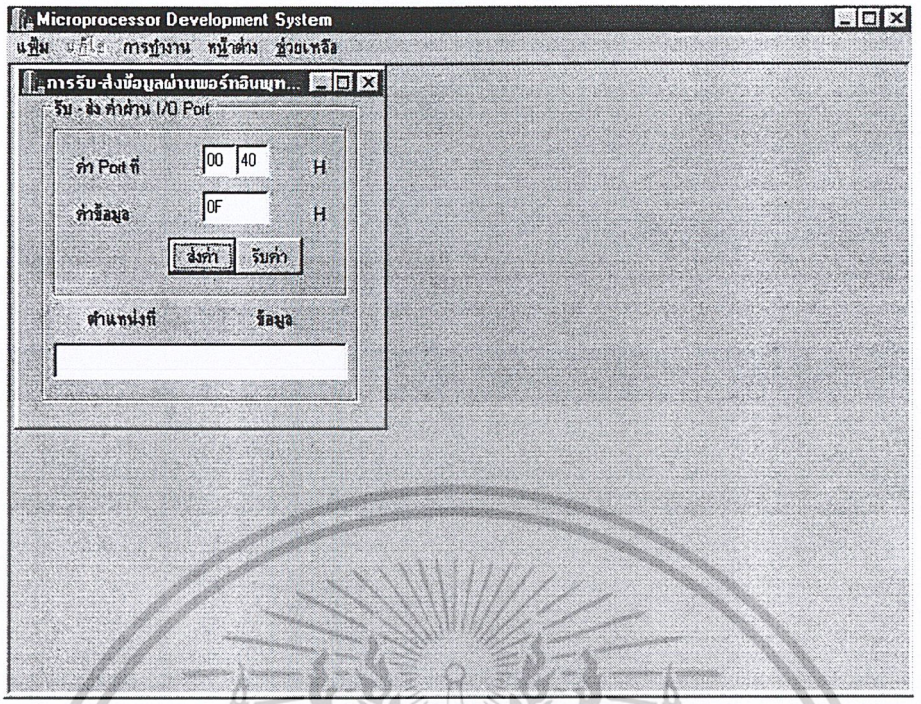
การทดลองนี้จะเป็นการรับและส่งค่าข้อมูล ผ่านทาง I / O พอร์ต ของ บอร์ด Z-80 Controller ซึ่ง บอร์ด Z-80 Controller ที่ใช้ในการทดลองนี้ ใช้ ไอซี เบอร์ 8255 มาเป็น I / O พอร์ต การทำงานกล่าวคือ จะกำหนดค่าแอดเดรส ซึ่งค่านี้จะเป็นตัวบอกตำแหน่ง พอร์ต ในการส่งค่าข้อมูลออกไปก็จะต้องกำหนดค่าข้อมูลที่หน้าจอโปรแกรมด้วย และในการรับค่าข้อมูลมาจาก พอร์ต นั้น ก็จะสามารถแสดงผลได้ที่หน้าจอโปรแกรม โดยมีขั้นตอนการทำงานดังนี้



รูปที่ 4.5 แสดงการส งข อมูลผ าน I / O พอร ต



รูปที่ 4.6 แสดงการรับข้อมูลผ่าน I / O พอร์ต

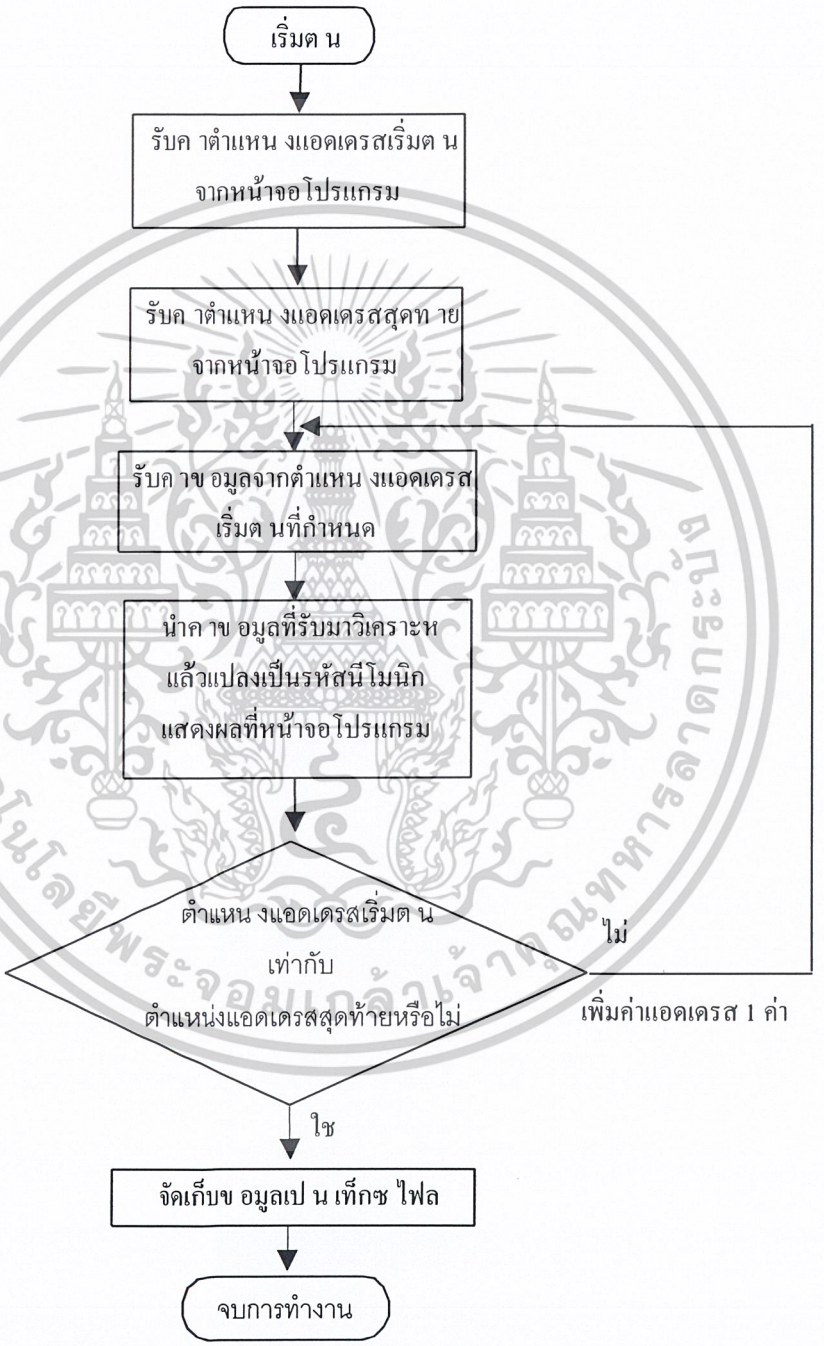


รูปที่ 4.7 แสดงหน้าจอโปรแกรมการรับและส่งค่าข้อมูล ผ่านทาง I / O พอร์ต

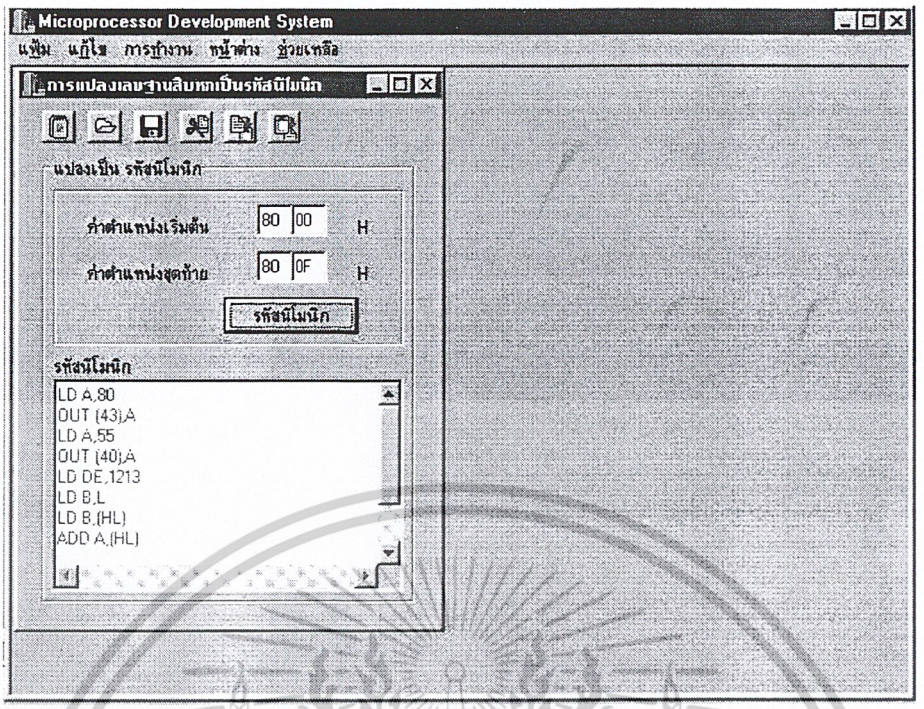
4.3 การทดลองที่ 3 การรับค่าข้อมูลแล้วนำมาแปลงเป็นรหัสนีโมนิก

ในการทดลองนี้จะเป็นการรับค่าข้อมูลเข้ามาแล้วให้โปรแกรมที่เขียนขึ้นมาวิเคราะห์ แปลงกลับเป็นรหัสนีโมนิก

การทำงานกล่าว คือ กำหนดตำแหน่งแอดเดรสเริ่มต้น และตำแหน่งแอดเดรสสุดท้าย โปรแกรมจะแปลงเป็นรหัสนีโมนิก แล้วจัดเก็บเป็นเท็กซ์ไฟล์ เพื่อนำไปใช้งานต่อไปโดยมีขั้นตอนการทำงานดังนี้



รูปที่ 4.8 แสดงการรับค่าข้อมูลแล้วนำมาแปลงเป็นรหัสนีโมนิก

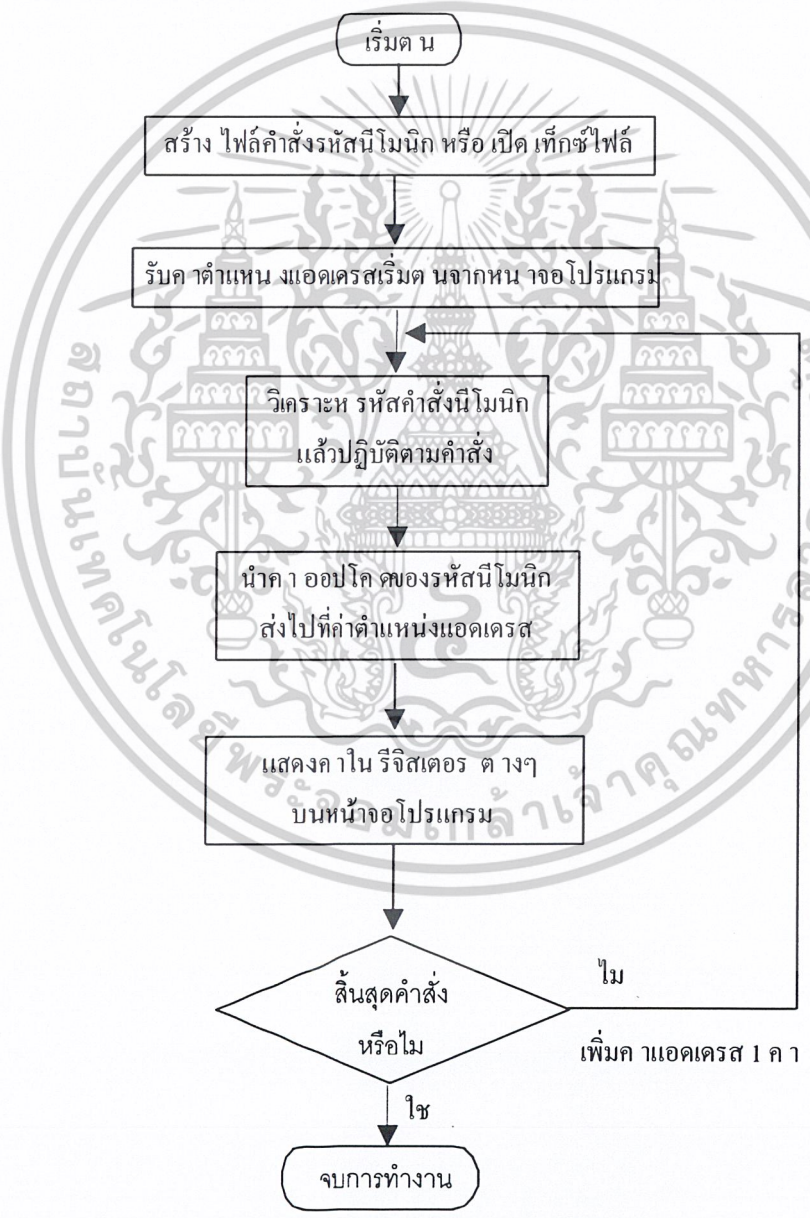


รูปที่ 4.9 แสดงหน้าจอโปรแกรมการรับค่าข้อมูลแล้วนำมาแปลงเป็นรหัสนีโมนิก

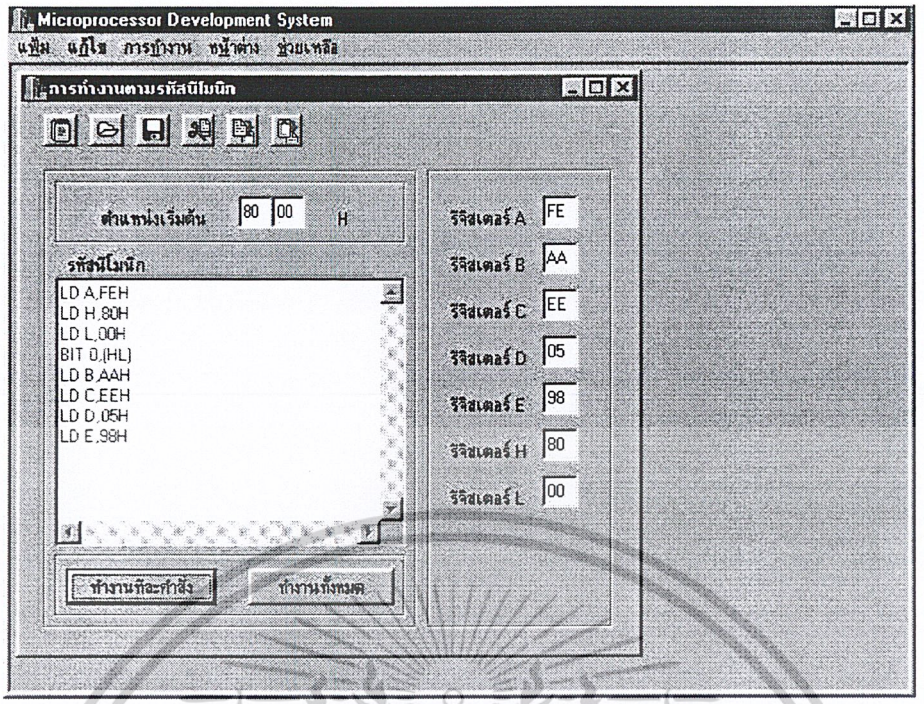
4.4 การทดลองที่ 4 การทำการตามคำสั่งรหัสนิวโมนิก

การทดลองนี้เป็นการใช้โปรแกรมที่พัฒนาขึ้น อ่านและวิเคราะห์เท็กซ์ไฟล์ ที่มีรหัสคำสั่งนีโมนิก แล้วนำมาปฏิบัติตามคำสั่งที่ได้รับมา

การทำงานกล่าวคือ กำหนดค่าแอดเดรสเริ่มต้นของโปรแกรม (Origin) ที่สามารถพิมพ์รหัสคำสั่งนีโมนิกที่ต้องการหรือแก้ไขได้ที่หน้าจอโปรแกรม หรือ เปิด เท็กซ์ไฟล์ ที่มีรหัสคำสั่งนีโมนิก อ่านคำสั่งในเท็กซ์ไฟล์ ทำการวิเคราะห์ แล้วปฏิบัติตามที่ละบรรทัด พร้อมทั้งส่งค่า ออปโค๊ด (OP CODE) ของคำสั่งนั้นๆ ไปไว้ ณ ตำแหน่งที่กำหนดไว้ โดยมีขั้นตอนการทำงานดังนี้



รูปที่ 4.10 แสดงการทำการตามคำสั่งรหัสนีโมนิก



รูปที่ 4.11 แสดงหน้าจอโปรแกรมการทำการตามคำสั่งรหัสนีโมนิก

บทที่ 5
บทวิจารณ์และสรุปผลการทดลอง

ผลการทดลอง

1. การทดลองที่ 1 การรับ-ส่งข้อมูลกับหน่วยความจำ

ในการทดลองนี้ได้มีการเขียนโปรแกรมเพื่อจำลองสภาวะการรับและส่งข้อมูลให้หน่วยความจำของไมโครโปรเซสเซอร์เบอร์ Z – 80 โดยมีลักษณะการทำงานตามแผนผังลำดับการทำงานของสภาวะการรับและส่งข้อมูลในการทดลองสามารถส่งข้อมูลขนาด 8 บิต ไปเก็บไว้ในหน่วยความจำบนบอร์ดทดลอง ที่ตำแหน่งแอดเดรสที่กำหนดได้ และสามารถรับข้อมูลจากตำแหน่งแอดเดรสที่กำหนดไว้ มาแสดงบนหน้าจอโปรแกรมได้เช่นกัน

2. การทดลองที่ 2 การรับ-ส่งข้อมูลผ่านพอร์ต อินพุต/เอาต์พุต

ในการทดลองนี้ได้มีการเขียนโปรแกรมเพื่อจำลองสภาวะการรับและส่งข้อมูลผ่านพอร์ตอินพุต/เอาต์พุต ของไมโครโปรเซสเซอร์เบอร์ Z – 80 โดยมีลักษณะการทำงานตามแผนผังลำดับการทำงาน ของสภาวะการรับและส่งข้อมูลผ่านพอร์ตอินพุต/เอาต์พุต ในการทดลองนี้สามารถส่งข้อมูลขนาด 8 บิต ออกไปที่พอร์ตเอาต์พุตที่ตำแหน่งแอดเดรสที่กำหนดพอร์ตแล้วแสดงผลด้วยไดโอดเปล่งแสงได้ และสามารถรับข้อมูลจากตำแหน่งแอดเดรสกำหนดพอร์ตแล้วนำมาแสดงที่หน้าจอโปรแกรมได้

3. การทดลองที่ 3 การแปลงเลขฐานสิบหกเป็นรหัสนีโมนิก

ในการทดลองนี้ได้เขียนโปรแกรมขึ้นมาโดยมีหลักการทำงานตามสภาวะการรับและส่งข้อมูลของไมโครโปรเซสเซอร์เบอร์ Z – 80 เมื่อรับข้อมูลมาแล้วโปรแกรมจะทำการวิเคราะห์แล้วแปลงเป็นรหัสนีโมนิกแสดงผลที่หน้าจอโปรแกรม

4. การทดลองที่ 4 การทำงานตามรหัสนีโมนิก

ในการทดลองนี้เป็นการนำรหัสนีโมนิกที่อยู่ในลักษณะเท็กซ์ไฟล์ แล้วโปรแกรมที่เขียนขึ้นมาจะอ่านรหัสนีโมนิกในเท็กซ์ไฟล์แล้วนำมาวิเคราะห์ตีความหมายว่ารหัสคำสั่งนั้นให้ปฏิบัติอะไรแล้วปฏิบัติตาม รวมทั้งถอดรหัสคำสั่งเป็นเลขฐานสิบหกแล้วส่งเลขฐานสิบหกนั้นไปไว้ที่หน่วยความจำภายนอก และมีการจำลองรีจิสเตอร์ต่างๆของไมโครโปรเซสเซอร์เบอร์ Z – 80 ขึ้นมาแล้วแสดงที่หน้าจอเพื่อแสดงค่าต่างๆในรีจิสเตอร์ ว่ามีการเปลี่ยนแปลงตามรหัสคำสั่งนั้นอย่างไรบ้าง

ปัญหาที่เกิดขึ้น

1. จากการทดลองที่ 1 ไม่เกิดปัญหาใดๆ

2. จากการทดลองที่ 2 ไม่เกิดปัญหาใดๆ

3. จากการทดลองที่ 3 เนื่องจากในการแปลงเลขฐานสิบหกกลับมาเป็นรหัสนีโมนิกนั้นเป็นการรับค่าจากหน่วยความจำภายนอกบนบอร์ดทดลองมาตีความหมาย เพราะฉะนั้นจึงจะทำให้สามารถแสดงได้แต่เฉพาะส่วนของรหัสนีโมนิก แต่ส่วนประกอบอื่นเช่น ส่วนลาเบล (LABEL) หรือ ส่วนคอมเม้นต์ (COMMENT) ไม่สามารถนำมาแสดงที่หน้าจอโปรแกรมได้

4. จากการทดลองที่ 4 เพื่อให้สอดคล้องกับข้อจำกัดของการทดลองที่ 3 ดังนั้นการทดลองที่ 4 จึงเป็นการนำเฉพาะส่วนของรหัสนีโมนิกเท่านั้นเข้ามาในหน้าจอโปรแกรม ถ้านำส่วนประกอบอื่นเข้ามาด้วย เช่น ส่วนลาเบล หรือ ส่วนคอมเม้นต์ เข้ามาด้วยโปรแกรมจะไม่สามารถกระทำตามคำสั่งได้

แนวทางการแก้ไข

ปัญหาที่เกิดขึ้นในการทดลองที่ 3 และ การทดลองที่ 4 นั้นเป็นข้อจำกัดของโปรแกรม ที่ไม่สามารถนำส่วนของ ลาเบล และ คอมเม้นต์ มาแสดงร่วมกับ ส่วนของรหัสนีโมนิกได้ เนื่องจากในการทดลองนี้ เป็นการรับค่าจากหน่วยความจำภายนอกบนบอร์ดทดลอง แล้วนำมาแปลงเป็นรหัสนีโมนิก เพราะฉะนั้น ไม่มีสิ่งที่จะแสดงให้เห็นว่าในส่วนของลาเบล และส่วนคอมเม้นต์ ว่าคืออะไร จึงไม่สามารถทำการแก้ไขปัญหาที่เกิดขึ้นนี้ได้

ภาคผนวก

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
8E	ADC A,(HL)	Add location (HL) to Acc
DD8E05	ADC A,(IX+d)	Add location (IX+d)to Acc
FD8E05	ADC A,(IY+d)	Add location (IY+d)to Acc
8F	ADC A,A	Add with carry operand s to Acc
88	ADC A,B	”
89	ADC A,C	”
8A	ADC A,D	”
8B	ADC A,E	”
8C	ADC A,H	”
8D	ADC A,L	”
CE20	ADC A,n	Add value n to Acc
ED4A	ADC HL,BC	Add with carry Reg. pair ss to HL
ED5A	ADC HL,DE	”
ED6A	ADC HL,HL	”
ED7A	ADC HL,SP	”
86	ADD A,(HL)	Add location (HL) to Acc
DD8605	ADD A,(IX+d)	Add location (IX+d)to Acc
FD8605	ADD A,(IY+d)	Add location (IY+d)to Acc
87	ADD A,A	Add with carry operand s to Acc
80	ADD A,B	”
81	ADD A,C	”
82	ADD A,D	”
83	ADD A,E	”
84	ADD A,H	”
85	ADD A,L	”
C620	ADD A,n	Add value n to Acc
09	ADD HL,BC	Add Reg. pair ss to HL
19	ADD HL,DE	”
29	ADD HL,HL	”
39	ADD HL,SP	”
DD09	ADD IX,BC	Add Reg. pair pp to IX
DD19	ADD IX,DE	”

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
DD29	ADD IX,IX	Add Reg. pair pp to IX
DD39	ADD IX,SP	”
FD09	ADD IY,BC	Add Reg. pair rr to IY
FD19	ADD IY,DE	”
FD29	ADD IY,IY	”
FD39	ADD IY,SP	”
A6	AND (HL)	-
DDA605	AND (IX+d)	-
FDA605	AND (IY+d)	-
A7	AND A	Logical ‘AND’ of operand s and Acc
A0	AND B	”
A1	AND C	”
A2	AND D	”
A3	AND E	”
A4	AND H	”
A5	AND L	”
E620	AND n	”
CB46	BIT 0,(HL)	Test BIT b of location (HL)
DDCB0546	BIT 0,(IX+d)	Test BIT b of location (IX+d)
FDCB0546	BIT 0,(IY+d)	Test BIT b of location (IY+d)
CB47	BIT 0,A	Test BIT b of Reg. r
CB40	BIT 0,B	”
CB41	BIT 0,C	”
CB42	BIT 0,D	”
CB43	BIT 0,E	”
CB44	BIT 0,H	”
CB45	BIT 0,L	”
CB4E	BIT 1,(HL)	Test BIT b of location (HL)
DDCB054E	BIT 1,(IX+d)	Test BIT b of location (IX+d)
FDCB054E	BIT 1,(IY+d)	Test BIT b of location (IY+d)
CB4F	BIT 1,A	Test BIT b of Reg. r
CB48	BIT 1,B	”

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
CB4C	BIT 1,H	Test BIT b of Reg. r
CB4D	BIT 1,L	”
CB56	BIT 2,(HL)	Test BIT b of location (HL)
DDCB0556	BIT 2,(IX+d)	Test BIT b of location (IX+d)
FDCB0556	BIT 2,(IY+d)	Test BIT b of location (IY+d)
CB57	BIT 2,A	Test BIT b of Reg. r
CB50	BIT 2,B	”
CB51	BIT 2,C	”
CB52	BIT 2,D	”
CB53	BIT 2,E	”
CB54	BIT 2,H	”
CB55	BIT 2,L	”
CB5E	BIT 3,(HL)	Test BIT b of location (HL)
DDCB055E	BIT 3,(IX+d)	Test BIT b of location (IX+d)
FDCB055E	BIT 3,(IY+d)	Test BIT b of location (IY+d)
CB5F	BIT 3,A	Test BIT b of Reg. r
CB58	BIT 3,B	”
CB59	BIT 3,C	”
CB5A	BIT 3,D	”
CB5B	BIT 3,E	”
CB5C	BIT 3,H	”
CB5D	BIT 3,L	”
CB66	BIT 4,(HL)	Test BIT b of location (HL)
DDCB0566	BIT 4,(IX+d)	Test BIT b of location (IX+d)
FDCB0566	BIT 4,(IY+d)	Test BIT b of location (IY+d)
CB67	BIT 4,A	Test BIT b of Reg. r
CB60	BIT 4,B	”
CB61	BIT 4,C	”
CB62	BIT 4,D	”
CB63	BIT 4,E	”
CB64	BIT 4,H	”
CB65	BIT 4,L	”

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
CB6E	BIT 5,(HL)	Test BIT b of location (HL)
DDCB056E	BIT 5,(IX+d)	Test BIT b of location (IX+d)
FDCB056E	BIT 5,(IY+d)	Test BIT b of location (IY+d)
CB6F	BIT 5,A	Test BIT b of Reg. r
CB68	BIT 5,B	”
CB69	BIT 5,C	”
CB6A	BIT 5,D	”
CB6B	BIT 5,E	”
CB6C	BIT 5,H	”
CB6D	BIT 5,L	”
CB76	BIT 6,(HL)	Test BIT b of location (HL)
DDCB0576	BIT 6,(IX+d)	Test BIT b of location (IX+d)
FDCB0576	BIT 6,(IY+d)	Test BIT b of location (IY+d)
CB77	BIT 6,A	Test BIT b of Reg. r
CB70	BIT 6,B	”
CB71	BIT 6,C	”
CB72	BIT 6,D	”
CB73	BIT 6,E	”
CB74	BIT 6,H	”
CB75	BIT 6,L	”
CB7E	BIT 7,(HL)	Test BIT b of location (HL)
DDCB057E	BIT 7,(IX+d)	Test BIT b of location (IX+d)
FDCB057E	BIT 7,(IY+d)	Test BIT b of location (IY+d)
CB7F	BIT 7,A	Test BIT b of Reg. r
CB78	BIT 7,B	”
CB79	BIT 7,C	”
CB7A	BIT 7,D	”
CB7B	BIT 7,E	”
CB7C	BIT 7,H	”
CB7D	BIT 7,L	”
DC8405	CALL C,nn	Call subroutine at location nn if condition cc if ture
FC8405	CALL M,nn	Call subroutine at location nn if condition cc if ture

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
D48405	CALL NC,nn	Call subroutine at location nn if condition cc if ture
C48405	CALL NZ,nn	”
F48405	CALL P,nn	”
EC8405	CALL PE,nn	”
E48405	CALL PO,nn	”
CC8405	CALL Z,nn	”
CD8405	CALL nn	Unconditional call subroutine at location nn
3F	CCF	Complement carry flag
BE	CP (HL)	-
DDBE05	CP (IY+d)	-
FDBE05	CP (IY+d)	-
BF	CP A	Compare operand s with Acc.
B8	CP B	”
B9	CP C	”
BA	CP D	”
BB	CP E	”
BC	CP H	”
BD	CP L	”
FE20	CP n	”
EDA9	CPD	Compare location (HL) and Acc. Decrement HL and BC
EDB9	CPDR	Compare location (HL) and Acc. Decrement HL and BC repeat until BC=0
EDB1	CPIR	Compare location (HL) and Acc. Increment HL and Decrement BC repeat until BC=0
EDA1	CPI	Compare location (HL) and Acc. Increment HL and Decrement BC
2F	CPL	Complement Acc. (1’s comp.)
27	DDA	Decimal adjust Acc.
35	DEC (HL)	-
DD3505	DEC (IX+d)	Decrement (IX+d)
FD3505	DEC (IY+d)	Decrement (IY+d)
3D	DEC A	Decrement operand m
05	DEC B	”
0B	DEC BC	Decrement Reg. pair ss
0D	DEC C	Decrement operand m

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
15	DEC D	Decrement operand m
1B	DEC DE	Decrement Reg. pair ss
1D	DEC E	Decrement operand m
25	DEC H	Decrement operand m
2B	DEC HL	Decrement Reg. pair ss
DD2B	DEC IX	Decrement IX
FD2B	DEC IY	Decrement IY
2D	DEC L	Decrement operand m
3B	DEC SP	Decrement Reg. pair sp
F3	DI	Disable interrupts
102E	DJNZ e	Decrement B and Jump relative if B ≠ 0
FB	EI	Enable interrupts
E3	EX (SP),HL	Exchange the location (SP) and HL
DDE3	EX (SP),IX	Exchange the location (SP) and IX
FDE3	EX (SP),IY	Exchange the location (SP) and IY
08	EX AF,AF'	Exchange the contents of AF and AF'
EB	EX DE,HL	Exchange the contents of DE and HL
D9	EXX	Exchange the contents of BC,DE,HL with contents of Bc',DE',HL'
76	HALT	HALT (wait for interrupt or reset)
ED46	IM 0	Set interrupt mode 0
ED56	IM 1	Set interrupt mode 1
ED5E	IM 2	Set interrupt mode 2
ED78	IN A,(C)	Load the Reg. r with input from device (C)
ED40	IN B,(C)	"
ED48	IN C,(C)	"
ED50	IN D,(C)	"
ED58	IN E,(C)	"
ED60	IN H,(C)	"
ED68	IN L,(C)	"
34	INC (HL)	Increment location (HL)
DD3405	INC (IX+d)	Increment location (IX+d)
FD3405	INC (IY+d)	Increment location (IY+d)

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
3C	INC A	Increment Reg. r
04	INC B	”
03	INC BC	”
0C	INC C	”
14	INC D	”
13	INC DE	”
1C	INC E	”
24	INC H	”
23	INC HL	”
DD23	INC IX	Increment IX
FD23	INC IY	Increment IY
2C	INC L	Increment Reg. r
33	INC SP	Increment Reg. pair ss
DB20	IN A,(n)	Load the Reg. Acc. with input from device n
EDAA	IND	Load location (HL) with input from port (C) decrement HL and B
EDBA	INDR	Load location (HL) with input from port (C) decrement HL and B repeat B=0
EDA2	INI	Load location (HL) with input from port (C) increment HL and decrement B
EDB2	INIR	Load location (HL) with input from port (C) increment HL and decrement B
C38405	JP nn	Unconditional jump to location nn
E9	JP (HL)	Unconditional jump to (HL)
DDE9	JP (IX)	Unconditional jump to (IX)
FDE9	JP (IY)	Unconditional jump to (IY)
DA8405	JP C,nn	Jump to location nn if condition cc is ture
FA8405	JP M,nn	”
D28405	JP NC,nn	”
C28405	JP NZ,nn	”
F28405	JP P,nn	”
EA8405	JP PE,nn	”
E28405	JP PO,nn	”
CA8405	JP Z,nn	”
382E	JR C,e	Jump relative to PC+e if carry=1
302E	JR NC,e	Jump relative to PC+e if carry=0

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
202E	JR NZ,e	Jump relative to PC+e if non zero (Z=0)
282E	JR Z,e	Jump relative to PC+e if zero (Z=1)
182E	JR e	Unconditional Jump relative to PC+e
02	LD (BC),A	Load location (BC) with Acc.
12	LD (DE),A	Load location (DE) with Acc.
77	LD (HL),A	Load location (HL) with Acc.
70	LD (HL),B	Load location (BC) with Reg. r
71	LD (HL),C	”
72	LD (HL),D	”
73	LD (HL),E	”
74	LD (HL),H	”
75	LD (HL),L	”
3620	LD (HL),n	”
DD7705	LD (IX+d),A	Load location (IX+d) with Reg. r
DD7005	LD (IX+d),B	”
DD7105	LD (IX+d),C	”
DD7205	LD (IX+d),D	”
DD7305	LD (IX+d),E	”
DD7405	LD (IX+d),H	”
DD7505	LD (IX+d),L	”
DD360520	LD (IX+d),n	Load location (IX+d) with value n
FD7705	LD (IY+d),A	Load location (IY+d) with Reg. r
FD7005	LD (IY+d),B	”
FD7105	LD (IY+d),C	”
FD7205	LD (IY+d),D	”
FD7305	LD (IY+d),E	”
FD7405	LD (IY+d),H	”
FD7505	LD (IY+d),L	”
FD360520	LD (IY+d),n	Load location (IY+d) with value n
328405	LD (nn),A	Load location (nn) with Acc.
ED438405	LD (nn),BC	Load location (nn) with Reg. pair dd
ED538405	LD (nn),DE	”

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
228405	LD (nn),HL	Load location (nn) with HL
DD228405	LD (nn),IX	Load location (nn) with IX.
FD228405	LD (nn),IY	Load location (nn) with IY
ED738405	LD (nn),SP	-
0A	LD A,(BC)	Load Acc. with location (BC)
1A	LD A,(DE)	Load Acc. with location (DE)
7E	LD A,(HL)	Load Acc. with location (HL)
DD7E05	LD A,(IX+d)	Load Acc. with location (IX+d)
FD7E05	LD A,(IY+d)	Load Acc. with location (IY+d)
3A8405	LD A,(nn)	Load Acc. with location (nn)
7F	LD A,A	Load Acc. with Reg. r
78	LD A,B	”
79	LD A,C	”
7A	LD A,D	”
7B	LD A,E	”
7C	LD A,H	”
ED57	LD A,I	Load Acc. with I
7D	LD A,L	Load Acc. with Reg. r
3E20	LD A,n	Load Acc. with location n
ED5F	LD A,R	Load Acc. with Reg. r
46	LD B,(HL)	Load Reg. r with location (HL)
DD4605	LD B,(IX+d)	Load Reg. r with location (IX+d)
FD4605	LD B,(IY+d)	Load Reg. r with location (IY+d)
47	LD B,A	Load Reg. r with location Reg. r’
40	LD B,B	”
41	LD B,C	”
42	LD B,D	”
43	LD B,E	”
44	LD B,H	”
45	LD B,L	”
0620	LD B,n	Load Reg. r with value n
ED4B8405	LD BC,(nn)	-

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
018405	LD BC,nn	-
4E	LD C,(HL)	Load Reg. r with location (HL)
DD4E05	LD C,(IX+d)	Load Reg. r with location (IX+d)
FD4E05	LD C,(IY+d)	Load Reg. r with location (IY+d)
4F	LD C,A	Load Reg. r with location Reg. r'
48	LD C,B	"
49	LD C,C	"
4A	LD C,D	"
4B	LD C,E	"
4C	LD C,H	"
4D	LD C,L	"
0E20	LD C,n	Load Reg. r with value n
56	LD D,(HL)	Load Reg. r with location (HL)
DD5605	LD D,(IX+d)	Load Reg. r with location (IX+d)
FD5605	LD D,(IY+d)	Load Reg. r with location (IY+d)
57	LD D,A	Load Reg. r with location Reg. r'
50	LD D,B	"
51	LD D,C	"
52	LD D,D	"
53	LD D,E	"
54	LD D,H	"
55	LD D,L	"
1620	LD D,n	Load Reg. r with value n
ED5B8405	LD DE,(nn)	"
118405	LD DE,nn	"
5E	LD E,(HL)	Load Reg. r with location (HL)
DD5E05	LD E,(IX+d)	Load Reg. r with location (IX+d)
FD5E05	LD E,(IY+d)	Load Reg. r with location (IY+d)
5F	LD E,A	Load Reg. r with location Reg. r'
58	LD E,B	"
59	LD E,C	"
5A	LD E,D	"

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
5B	LD E,E	Load Reg. r with location Reg. r'
5C	LD E,H	"
5D	LD E,L	"
1E20	LD E,n	Load Reg. r with value n
66	LD H,(HL)	Load Reg. r with location (HL)
DD6605	LD H,(IX+d)	Load Reg. r with location (IX+d)
FD6605	LD H,(IY+d)	Load Reg. r with location (IY+d)
67	LD H,A	Load Reg. r with location Reg. r'
60	LD H,B	"
61	LD H,C	"
62	LD H,D	"
63	LD H,E	"
64	LD H,H	"
65	LD H,L	"
2620	LD H,n	Load Reg. r with value n
2A8405	LD HL,(nn)	Load HL with location (nn)
218405	LD HL,nn	Load HL with value nn
ED47	LD I,A	Load I with Acc.
DD2A8405	LD IX,(nn)	Load IX with location (nn)
DD218405	LD IX,nn	Load IX with value nn
FD2A8405	LD IY,(nn)	Load IY with location (nn)
FD218405	LD IY,nn	Load IY with value nn
6E	LD L,(HL)	Load Reg. r with location (HL)
DD6E05	LD L,(IX+d)	Load Reg. r with location (IX+d)
FD6E05	LD L,(IY+d)	Load Reg. r with location (IY+d)
6F	LD L,A	Load Reg. r with location Reg. r'
68	LD L,B	"
69	LD L,C	"
6A	LD L,D	"
6B	LD L,E	"
6C	LD L,H	"
6D	LD L,L	"

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
2E20	LD L,n	Load Reg. r with value n
ED4F	LD R,A	Load R with Acc.
ED7B8405	LD SP,(nn)	Load SP with (nn)
F9	LD SP,HL	Load SP with HL
DDF9	LD SP,IX	Load SP with IX
FDF9	LD SP,IY	Load SP with IY
318405	LD SP,nn	Load SP with nn
EDA8	LDD	Load location (DE) with location (HL) ,decrement DE ,HL and BC
EDB8	LDDR	Load location (DE) with location (HL) ,decrement DE ,HL and BC repeat until
EDA0	LDI	Load location (DE) with location (HL) ,increment DE ,HL decrement BC
EDB0	LDIR	Load location (DE) with location (HL) ,increment DE ,HL decrement BC repeat
ED44	NEG	Negate Acc. (2's complement)
00	NOP	No operation
B6	OR (HL)	Logical 'OR' or operand s and Acc.
DDB605	OR (IX+d)	,,
FDB605	OR (IY+d)	,,
B7	OR A	,,
B0	OR B	,,
B1	OR C	,,
B2	OR D	,,
B3	OR E	,,
B4	OR H	,,
B5	OR L	,,
F620	OR n	,,
ED8B	OTDR	,,
EDB3	OTIR	Load output port (C) with location (HL) decrement HL and B ,repeat until B=0
ED79	OUT (C),A	Load output port (C) with location (HL) increment HL and B ,repeat until B=0
ED41	OUT (C),B	Load output port (C) with Reg. r
ED49	OUT (C),C	,,
ED51	OUT (C),D	,,
ED59	OUT (C),E	,,
ED61	OUT (C),H	,,

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
ED69	OUT (C),L	Load output port (C) with Reg. r
D320	OUT (n),A	Load output port (n) with Acc.
EDAB	OUTD	Load output port (C) with location (HL) decrement HL and B
EDA3	OUTI	Load output port (C) with location (HL) increment HL and B
F1	POP AF	Load Reg. pair qq with top of stack
C1	POP BC	”
D1	POP DE	”
E1	POP HL	”
DDE1	POP IX	”
FDE1	POP IY	Load IX with top of stack
F5	PUSH AF	Load IY with top of stack
C5	PUSH BC	Load Reg. pair qq onto stack
D5	PUSH DE	”
E5	PUSH HL	”
DDE5	PUSH IX	”
FDE5	PUSH IY	Load IX onto stack
CB86	RES 0,(HL)	Load IY onto stack
DDCB0586	RES 0,(IX+d)	Reset bit b of operand m
FDCB0586	RES 0,(IY+d)	”
CB87	RES 0,A	”
CB80	RES 0,B	”
CB81	RES 0,C	”
CB82	RES 0,E	”
CB83	RES 0,H	”
CB84	RES 0,L	”
CB85	RES 1,(HL)	”
CB8E	RES 1,(IX+d)	”
DDCB058E	RES 1,(IY+d)	”
FDCB058E	RES 1,A	”
CB8F	RES 1,B	”
CB88	RES 1,C	”
CB89	RES 1,E	”

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
CB8A	RES 1,D	Reset bit b of operand m
CB8B	RES 1,E	”
CB8C	RES 1,H	”
CB8D	RES 1,L	”
CB96	RES 2,(HL)	”
DDCB0596	RES 2,(IX+d)	”
FDCB0596	RES 2,(IY+d)	”
CB97	RES 2,A	”
CB90	RES 2,B	”
CB91	RES 2,C	”
CB92	RES 2,D	”
CB93	RES 2,E	”
CB94	RES 2,H	”
CB95	RES 2,L	”
CB9E	RES 3,(HL)	”
DDCB059E	RES 3,(IX+d)	”
FDCB059E	RES 3,(IY+d)	”
CB9F	RES 3,A	”
CB98	RES 3,B	”
CB99	RES 3,C	”
CB9A	RES 3,D	”
CB9B	RES 3,E	”
CB9C	RES 3,H	”
CB9D	RES 3,L	”
CBA6	RES 4,(HL)	”
DDCB05A6	RES 4,(IX+d)	”
FDCB05A6	RES 4,(IY+d)	”
CBA7	RES 4,A	”
CBA0	RES 4,B	”
CBA1	RES 4,C	”
CBA2	RES 4,D	”
CBA3	RES 4,E	Reset bit b of operand m

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
CBA4	RES 4,H	Reset bit b of operand m
CBA5	RES 4,L	”
CBAE	RES 5,(HL)	”
DDCB05AE	RES 5,(IX+d)	”
FDCB05AE	RES 5,(IY+d)	”
CBAF	RES 5,A	”
CBA8	RES 5,B	”
CBA9	RES 5,C	”
CBAA	RES 5,D	”
CBAB	RES 5,E	”
CBAC	RES 5,H	”
CBAD	RES 5,L	”
CBB6	RES 6,(HL)	”
DDCB05B6	RES 6,(IX+d)	”
FDCB05B6	RES 6,(IY+d)	”
CBB7	RES 6,A	”
CBB0	RES 6,B	”
CBB1	RES 6,C	”
CBB2	RES 6,D	”
CBB3	RES 6,E	”
CBB4	RES 6,H	”
CBB5	RES 6,L	”
CBBE	RES 7,(HL)	”
DDCB05BE	RES 7,(IX+d)	”
FDCB05BE	RES 7,(IY+d)	”
CBBF	RES 7,A	”
CBB8	RES 7,B	”
CBB9	RES 7,C	”
CBBA	RES 7,D	”
CBBB	RES 7,E	”
CBBC	RES 7,H	”
CBBD	RES 7,L	Reset bit b of operand m

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
C9	RET	Return from subroutine
D8	RET C	Return from subroutine if condition cc is ture
F8	RET M	”
D0	RET MC	”
C0	RET MZ	”
F0	RET P	”
E8	RET PE	”
E0	RET P0	”
C8	RET Z	”
ED4D	RETI	Return from interrupt
ED45	RETN	Return from non maskable interrupt
CB16	RL (HL)	Rotate left though carry operand m
DDCB0516	RL (IX+d)	”
FDCB0516	RL (IY+d)	”
CB17	RL A	”
CB10	RL B	”
CB11	RL C	”
CB12	RL D	”
CB13	RL E	”
CB14	RL H	”
CB15	RL L	”
17	RLA	Rotate left Acc. though carry
CB06	RLC (HL)	Rotate location (HL) left circular
DDCB0506	RLC (IX+d)	Rotate location (IX+d) left circular
FDCB0506	RLC (IY+d)	Rotate location (IY+d)left circular
CB07	RLC A	Rotate Reg. r left circular
CB00	RLC B	”
CB01	RLC C	”
CB02	RLC D	”
CB03	RLC E	”
CB04	RLC H	”
CB05	RLC L	”

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
07	RLCA	Rotate left circular Acc.
ED6F	RLD	Rotate digit left and right between Acc. and location (HL)
CB1E	RR (HL)	Rotate right though carry operand m
DDCB051E	RR (IX+d)	”
FDCB051E	RR (IY+d)	”
CB1F	RR A	”
CB18	RR B	”
CB19	RR C	”
CB1A	RR D	”
CB1B	RR E	”
CB1C	RR H	”
CB1D	RR L	”
1F	RRA	Rotate right Acc. though carry
CB0E	RRC (HL)	Rotate operand m right circular
DDCB050E	RRC (IX+d)	”
FDCB050E	RRC (IY+d)	”
CB0F	RRC A	”
CB0F	RRC B	”
CB09	RRC C	”
CB0A	RRC D	”
CB0B	RRC E	”
CB0C	RRC H	”
CB0D	RRC L	”
OF	RRCA	Rotate right circular Acc.
ED67	RRD	Rotate digit right and left between Acc. and location (HL)
C7	RST 00H	Restart to location p
CF	RST 08H	”
D7	RST 10H	”
DF	RST 18H	”
E7	RST 20H	”
EF	RST 28H	”
F7	RST 30H	”

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
FF	RST 38H	Restart to location p
DE20	SBC A,n	Subtract operand s from Acc. with carry
9E	SBC A,(HL)	”
DD9E05	SBC A,(IX+d)	”
FD9E05	SBC A,(IY+d)	”
9F	SBC A,A	”
98	SBC A,B	”
99	SBC A,C	”
9A	SBC A,D	”
9B	SBC A,E	”
9C	SBC A,H	”
9D	SBC A,L	”
ED42	SBC HL,BC	Substract Reg. pair ss from HL with carry
ED52	SBC HLDE	”
ED62	SBC HL,HL	”
ED72	SBC HL,SP	”
37	SCF	Set carry flag (C=1)
CBC6	SET 0,(HL)	Set bit B of location (HL)
DDCB05C6	SET 0,(IX+d)	Set bit B of location (IX+d)
FDCB05C6	SET 0,(IY+d)	Set bit B of location (IY+d)
CBC7	SET 0,A	Set bit B of Reg.r
CBC0	SET 0,B	”
CBC1	SET 0,C	”
CBC2	SET 0,D	”
CBC3	SET 0,E	”
CBC4	SET 0,H	”
CBC5	SET 0,L	”
CBCE	SET 1,(HL)	Set bit B of location (HL)
DDCB05CE	SET 1,(IX+d)	Set bit B of location (IX+d)
FDCB05CE	SET 1,(IY+d)	Set bit B of location (IY+d)
CBCF	SET 1,A	Set bit B of Reg.r
CBC8	SET 1,B	”

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
CBC9	SET 1,C	Set bit B of Reg.r
CBCA	SET 1,D	”
CBCB	SET 1,E	”
CBCC	SET 1,H	”
CBCD	SET 1,L	”
CBD6	SET 2,(HL)	Set bit B of location (HL)
DDCB05D6	SET 2,(IX+d)	Set bit B of location (IX+d)
FDCB05D6	SET 2,(IY+d)	Set bit B of location (IY+d)
CBD7	SET 2,A	Set bit B of Reg.r
CBD0	SET 2,B	”
CBD1	SET 2,C	”
CBD2	SET 2,D	”
CBD3	SET 2,E	”
CBD4	SET 2,H	”
CBD5	SET 2,L	”
CBDE	SET 3,(HL)	Set bit B of location (HL)
DDCB05DE	SET 3,(IX+d)	Set bit B of location (IX+d)
FDCB05DE	SET 3,(IY+d)	Set bit B of location (IY+d)
CBDF	SET 3,A	Set bit B of Reg.r
CBD9	SET 3,C	”
CBDA	SET 3,D	”
CBDB	SET 3,E	”
CBDC	SET 3,H	”
CBDD	SET 3,L	”
CBE6	SET 4,(HL)	Set bit B of location (HL)
DDCB05E6	SET 4,(IX+d)	Set bit B of location (IX+d)
FDCB05E6	SET 4,(IY+d)	Set bit B of location (IY+d)
CBE7	SET 4,A	Set bit B of Reg.r
CBE0	SET 4,B	”
CBE1	SET 4,C	”
CBE2	SET 4,D	”
CBE3	SET 4,E	”

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
CBE4	SET 4,H	Set bit B of Reg.r
CBE5	SET 4,L	”
CBEE	SET 5,(HL)	Set bit B of location (HL)
DDCB05EE	SET 5,(IX+d)	Set bit B of location (IX+d)
FDCB05EE	SET 5,(IY+d)	Set bit B of location (IY+d)
CBEF	SET 5,A	Set bit B of Reg.r
CBE8	SET 5,B	”
CBE9	SET 5,C	”
CBEA	SET 5,D	”
CBEB	SET 5,E	”
CBEC	SET 5,H	”
CBED	SET 5,L	”
CBF6	SET 6,(HL)	Set bit B of location (HL)
DDCB05F6	SET 6,(IX+d)	Set bit B of location (IX+d)
FDCB05F6	SET 6,(IY+d)	Set bit B of location (IY+d)
CBF7	SET 6,A	Set bit B of Reg.r
CBF0	SET 6,B	”
CBF1	SET 6,C	”
CBF2	SET 6,D	”
CBF3	SET 6,E	”
CBF4	SET 6,H	”
CBF5	SET 6,L	”
CBFE	SET 7,(HL)	Set bit B of location (HL)
DDCB05FE	SET 7,(IX+d)	Set bit B of location (IX+d)
FDCB05FE	SET 7,(IY+d)	Set bit B of location (IY+d)
CBFF	SET 7,A	Set bit B of Reg.r
CBF8	SET 7,B	”
CBF9	SET 7,C	”
CBFA	SET 7,D	”
CBFB	SET 7,E	”
CBFC	SET 7,H	”
CBFD	SET 7,L	”

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
CB26	SLA (HL)	Shift operand m left arithmetic
DDCB0526	SLA (IX+d)	”
FDCB0526	SLA (IY+d)	”
CB27	SLA A	”
CB20	SLA B	”
CB21	SLA C	”
CB22	SLA D	”
CB23	SLA E	”
CB24	SLA H	”
CB25	SLA L	”
CB2E	SRA (HL)	Shift operand m right arithmetic
DDCB052E	SRA (IX+d)	”
FDCB052E	SRA (IY+d)	”
CB2F	SRA A	”
CB28	SRA B	”
CB29	SRA C	”
CB2A	SRA D	”
CB2B	SRA E	”
CB2C	SRA H	”
CB2D	SRA L	”
CB3E	SRL (HL)	Shift operand m right logical
DDCB053E	SRL (IX+d)	”
FDCB053E	SRL (IY+d)	”
CB3F	SRL A	”
CB38	SRL B	”
CB39	SRL C	”
CB3A	SRL D	”
CB3B	SRL E	”
CB3C	SRL H	”
CB3D	SRL L	”
96	SUB (HL)	Subtract operand s from Acc.
DD9605	SUB (IX+d)	”

Z-80 INSTRUCTION CODES

OBJ CODE	SOURCE STATEMENT	Z-80 CPU INSTRUCTION SET
FD9605	SUB (IY+d)	Subtract operand s from Acc.
97	SUB A	”
90	SUB B	”
91	SUB C	”
92	SUB D	”
93	SUB E	”
94	SUB H	”
95	SUB L	”
D620	SUB n	”
AE	XOR (HL)	Exclusive ‘OR’ operand s and Acc.
DDAE05	XOR (IX+d)	”
FDAE05	XOR (IY+d)	”
AF	XOR A	”
A8	XOR B	”
A9	XOR C	”
AA	XOR D	”
AB	XOR E	”
AC	XOR H	”
AD	XOR L	”
EE20	XOR n	Exclusive ‘OR’ operand s and Acc.

intersil

82C55A

CMOS Programmable Peripheral Interface

June 1998

Features

- Pin Compatible with NMOS 8255A
- 24 Programmable I/O Pins
- Fully TTL Compatible
- High Speed, No "Wait State" Operation with 5MHz and 8MHz 80C86 and 80C88
- Direct Bit Set/Reset Capability
- Enhanced Control Word Read Capability
- L7 Process
- 2.5mA Drive Capability on All I/O Ports
- Low Standby Power (ICCSB) 10µA

Description

The Intersil 82C55A is a high performance CMOS version of the industry standard 8255A and is manufactured using a self-aligned silicon gate CMOS process (Scaled SAJI IV). It is a general purpose programmable I/O device which may be used with many different microprocessors. There are 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. The high performance and industry standard configuration of the 82C55A make it compatible with the 80C86, 80C88 and other microprocessors.

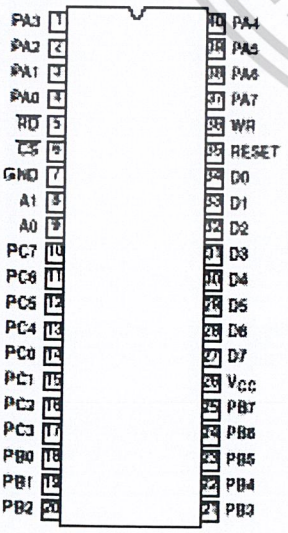
Static CMOS circuit design insures low operating power. TTL compatibility over the full military temperature range and bus hold circuitry eliminate the need for pull up resistors. The Intersil advanced SAJI process results in performance equal to or greater than existing functionally equivalent products at a fraction of the power.

Ordering Information

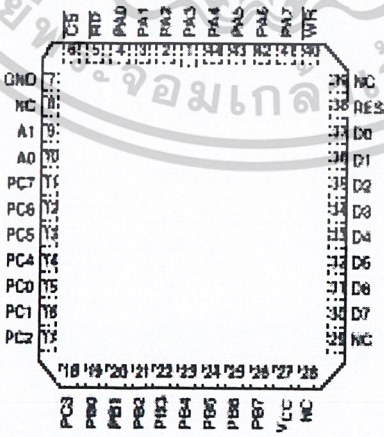
PART NUMBERS		PACKAGE	TEMPERATURE RANGE	PKG. NO.
5MHz	8MHz			
CP82C55A-5	CP82C55A	40 Ld PDIP	0°C to 70°C	E40.6
IP82C55A-5	IP82C55A		-40°C to 85°C	E40.6
CS82C55A-5	CS82C55A	44 Ld PLCC	0°C to 70°C	N44.65
IS82C55A-5	IS82C55A		-40°C to 85°C	N44.65
CD82C55A-5	CD82C55A	40 Ld CERDIP	0°C to 70°C	F40.6
ID82C55A-5	ID82C55A		-40°C to 85°C	F40.6
MD82C55A-5/B	MD82C55A/B		-55°C to 125°C	F40.6
84066010A	84066020A	SMD#		F40.6
MR82C55A-5/B	MR82C55A/B	44 Pad CLCC	-55°C to 125°C	J44.A
84066010A	84066020A	SMD#		J44.A

Pinouts

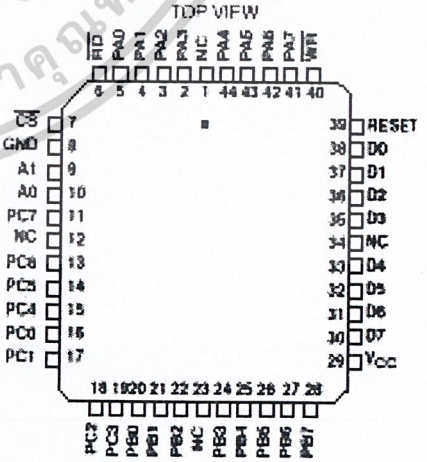
82C55A (DIP)
TOP VIEW



82C55A (CLCC)
TOP VIEW



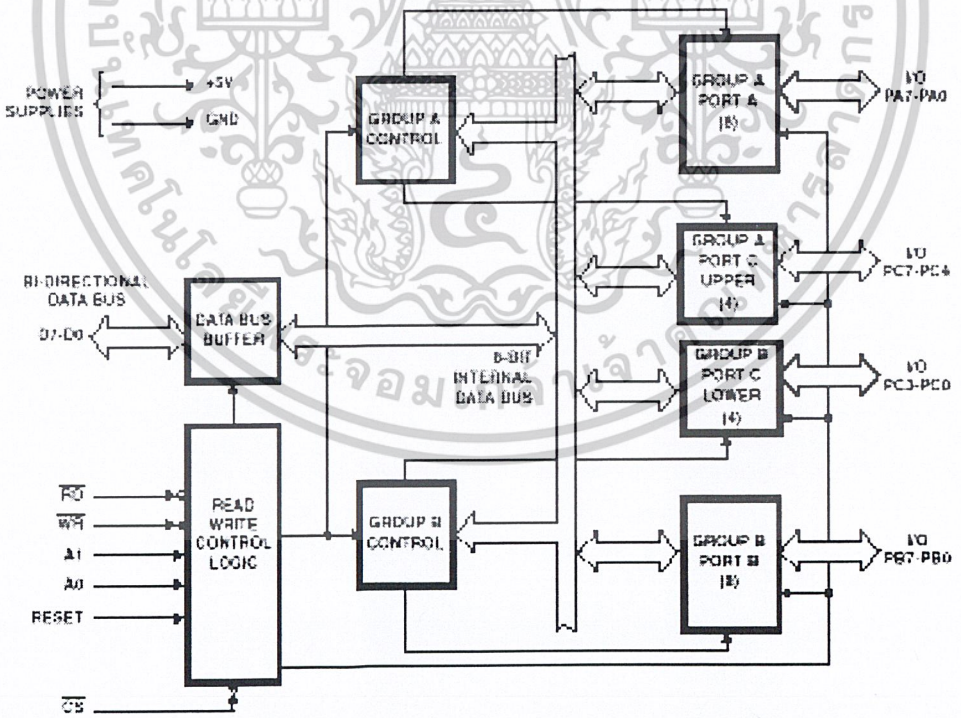
82C55A (PLCC)
TOP VIEW



Pin Description

SYMBOL	PIN NUMBER	TYPE	DESCRIPTION
V_{CC}	26		V_{CC}: The +5V power supply pin. A 0.1μF capacitor between pins 26 and 7 is recommended for decoupling
GND	7		GROUND
D0-D7	27-34	I/O	DATA BUS: The Data Bus lines are bidirectional three-state pins connected to the system data bus
RESET	35	I	RESET: A high on this input clears the control register and all ports (A, B, C) are set to the input mode with the "Bus Hold" circuitry turned off.
$\overline{CS}$	6	I	CHIP SELECT: Chip select is an active low input used to enable the 82C55A onto the Data Bus for CPU communications.
$\overline{RD}$	5	I	READ: Read is an active low input control signal used by the CPU to read status information or data via the data bus.
$\overline{WR}$	36	I	WRITE: Write is an active low input control signal used by the CPU to load control words and data into the 82C55A.
A0, A1	8, 9	I	ADDRESS: These input signals, in conjunction with the $\overline{RD}$ and $\overline{WR}$ inputs, control the selection of one of the three ports or the control word register. A0 and A1 are normally connected to the least significant bits of the Address Bus (A0, A1).
PA0-PA7	1-4, 37-40	I/O	PORT A: 8-bit input and output port. Both bus hold high and bus hold low circuitry are present on this port.
PB0-PB7	18-25	I/O	PORT B: 8-bit input and output port. Bus hold high circuitry is present on this port.
PC0-PC7	10-17	I/O	PORT C: 8-bit input and output port. Bus hold circuitry is present on the port.

Functional Diagram



Functional Description

Data Bus Buffer

This three state bi-directional 8 bit buffer is used to interface the 82C55A to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

(CS) Chip Select. A "low" on this input pin enables the communication between the 82C55A and the CPU.

(RD) Read. A "low" on this input pin enables 82C55A to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to "read from" the 82C55A.

(WR) Write. A "low" on this input pin enables the CPU to write data or control words into the 82C55A.

(A0 and A1) Port Select 0 and Port Select 1. These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word register. They are normally connected to the least significant bits of the address bus (A0 and A1).

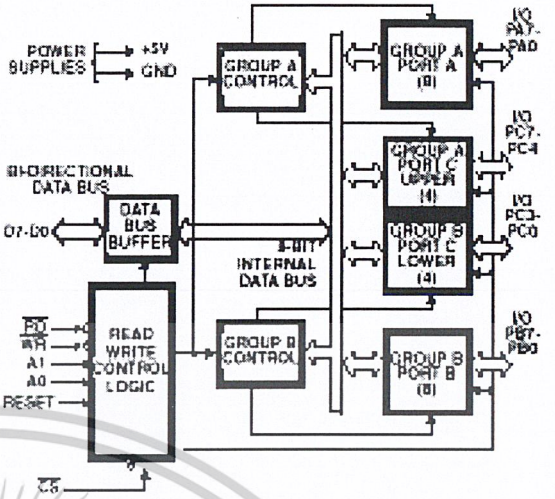


FIGURE 1. 82C55A BLOCK DIAGRAM, DATA BUS BUFFER, READ/WRITE, GROUP A & B CONTROL LOGIC FUNCTIONS

82C55A BASIC OPERATION

A1	A0	RD	WR	CS	INPUT OPERATION (READ)
0	0	0	1	0	Port A → Data Bus
0	1	0	1	0	Port B → Data Bus
1	0	0	1	0	Port C → Data Bus
1	1	0	1	0	Control Word → Data Bus
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	Data Bus → Port A
0	1	1	0	0	Data Bus → Port B
1	0	1	0	0	Data Bus → Port C
1	1	1	0	0	Data Bus → Control
					DISABLE FUNCTION
X	X	X	X	1	Data Bus → Three-State
X	X	1	1	0	Data Bus → Three State

(RESET) Reset. A "high" on this input initializes the control register to 9Bh and all ports (A, B, C) are set to the input mode. "Bus hold" devices internal to the 82C55A will hold the I/O port inputs to a logic "1" state with a maximum hold current of 400μA.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU "outputs" a control word to the 82C55A. The control word contains information such as mode, "bit set", "bit reset", etc., that initializes the functional configuration of the 82C55A.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Control Group A - Port A and Port C upper (C7 - C4)

Control Group B - Port B and Port C lower (C3 - C0)

The control word register can be both written and read as shown in the "Basic Operation" table. Figure 4 shows the control word format for both Read and Write operations. When the control word is read, bit D7 will always be a logic "1", as this implies control word mode information.

Ports A, B, and C

The 82C55A contains three 8-bit ports (A, B, and C). All can be configured to a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 82C55A.

Port A One 8-bit data output latch/buffer and one 8-bit data input latch. Both "pull-up" and "pull-down" bus-hold devices are present on Port A. See Figure 2A.

Port B One 8-bit data input/output latch/buffer and one 8-bit data input buffer. See Figure 2B.

Port C One 8-bit data output latch buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal output and status signal inputs in conjunction with ports A and B. See Figure 2B.

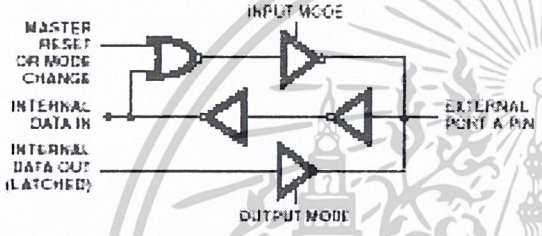


FIGURE 2A. PORT A BUS-HOLD CONFIGURATION

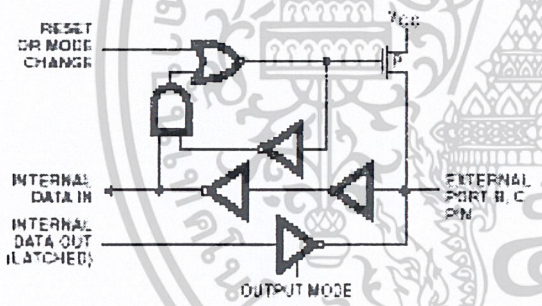


FIGURE 2B. PORT B AND C BUS HOLD CONFIGURATION

FIGURE 2. BUS-HOLD CONFIGURATION

Operational Description

Mode Selection

There are three basic modes of operation that can be selected by the system software:

Mode 0 - Basic Input/Output
Mode 1 - Strobed Input/Output
Mode 2 - Bi-directional Bus

When the reset input goes "high" all ports will be set to the input mode with all 24 port lines held at a logic "one" level by internal bus hold devices. After the reset is removed, the 82C55A can remain in the input mode with no additional initialization required. This eliminates the need to pullup or pull-down resistors in all-CMOS designs. The control word register will contain 9Bh. During the execution of the system program, any of the other modes may be selected using a single output instruction. This allows a single 82C55A to service a variety of peripheral devices with a simple software maintenance routine. Any port programmed as an output port is initialized to all zeros when the control word is written.

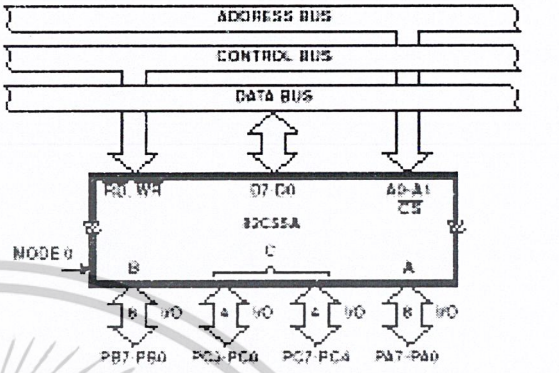


FIGURE 3. BASIC MODE DEFINITIONS AND BUS INTERFACE

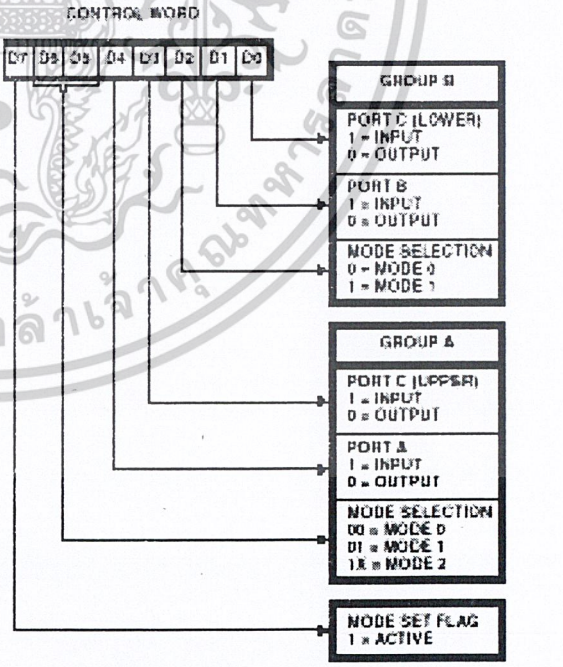


FIGURE 4. MODE DEFINITION FORMAT

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be "tailored" to almost any I/O structure. For instance: Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.

The mode definitions and possible mode combinations may seem confusing at first, but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 82C55A has taken into account things such as efficient PC board layout, control signal definition vs. PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature (Figure 5)

Any of the eight bits of Port C can be Set or Reset using a single Output instruction. This feature reduces software requirements in control based applications.

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were output ports.

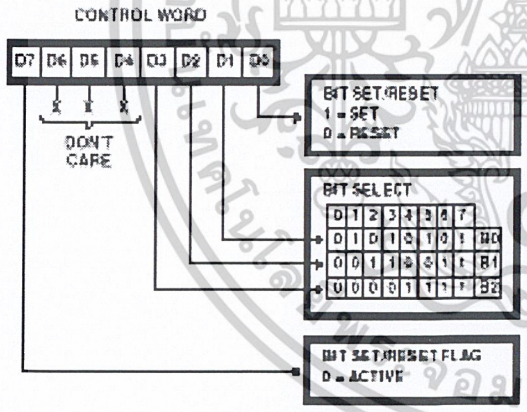


FIGURE 5. BIT SET/RESET FORMAT

Interrupt Control Functions

When the 82C55A is programmed to operate in mode 1 or mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the bit set/reset function of port C.

This function allows the programmer to enable or disable a CPU interrupt by a specific I/O device without affecting any other device in the interrupt structure.

INTE Flip-Flop Definition

(BIT-SET)-INTE is SET - Interrupt Enable

(BIT-RESET)-INTE is Reset - Interrupt Disable

NOTE: All Mask flip-flops are automatically reset during mode selection and device Reset.

Operating Modes

Mode 0 (Basic Input/Output) This functional configuration provides simple input and output operations for each of the three ports. No handshaking is required, data is simply written to or read from a specific port.

Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports
- Any Port can be input or output
- Outputs are latched
- Inputs are not latched
- 16 different Input/Output configurations possible

MODE 0 PORT DEFINITION

A		B		GROUP A			GROUP B	
D4	D3	D1	D0	PORT A	PORT C (Upper)	#	PORT B	PORT C (Lower)
0	0	0	0	Output	Output	0	Output	Output
0	0	0	1	Output	Output	1	Output	Input
0	0	1	0	Output	Output	2	Input	Output
0	0	1	1	Output	Output	3	Input	Input
0	1	0	0	Output	Input	4	Output	Output
0	1	0	1	Output	Input	5	Output	Input
0	1	1	0	Output	Input	6	Input	Output
0	1	1	1	Output	Input	7	Input	Input
1	0	0	0	Input	Output	8	Output	Output
1	0	0	1	Input	Output	9	Output	Input
1	0	1	0	Input	Output	10	Input	Output
1	0	1	1	Input	Output	11	Input	Input
1	1	0	0	Input	Input	12	Output	Output
1	1	0	1	Input	Input	13	Output	Input
1	1	1	0	Input	Input	14	Input	Output
1	1	1	1	Input	Input	15	Input	Input

MOTOROLA

SN54/74LS365A
SN54/74LS366A
SN54/74LS367A
SN54/74LS368A

3-STATE HEX BUFFERS

These devices are high speed hex buffers with 3 state outputs. They are organized as single 6-bit or 2-bit/4-bit, with inverting or non-inverting data (D) paths. The outputs are designed to drive 15 TTL Unit Loads or 60 Low Power Schottky loads when the Enable (E) is LOW.

When the Output Enable (E) is HIGH, the outputs are forced to a high impedance "off" state. If the outputs of the 3 state devices are tied together, all but one device must be in the high impedance state to avoid high currents that would exceed the maximum ratings. Designers should ensure that Output Enable signals to 3-state devices whose outputs are tied together are designed so there is no overlap.

3-STATE HEX BUFFERS

LOW POWER SCHOTTKY

J SUFFIX
CERAMIC
CASE 620-09

N SUFFIX
PLASTIC
CASE 648-08

D SUFFIX
SOIC
CASE 751B-05

ORDERING INFORMATION

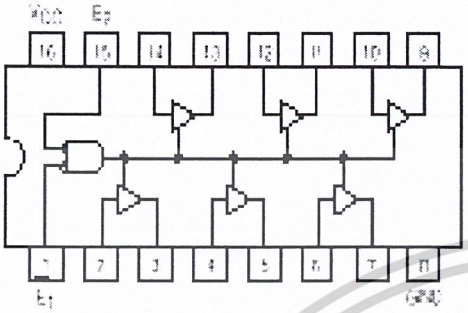
SN54LSXXXJ	Ceramic
SN74LSXXXN	Plastic
SN74LSXXXD	SOIC

GUARANTEED OPERATING RANGES

Symbol	Parameter		Min	Typ	Max	Unit
V_{CC}	Supply Voltage	54 74	4.5 4.75	5.0 5.0	5.5 5.25	V
T_A	Operating Ambient Temperature Range	54 74	−55 0	25 25	125 70	°C
I_{OH}	Output Current — High	54 74			−1.0 −2.6	mA
I_{OL}	Output Current — Low	54 74			12 24	mA

SN54/74LS365A • SN54/74LS366A
SN54/74LS367A • SN54/74LS368A

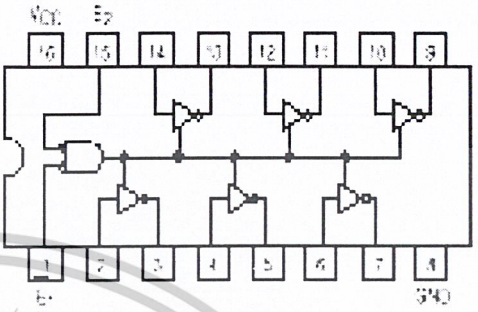
SN54/74LS365A
HEX 3-STATE BUFFER WITH
COMMON 2-INPUT NOR ENABLE



TRUTH TABLE

INPUTS			OUTPUT
$\overline{E_1}$	$\overline{E_2}$	D	
L	L	L	L
L	L	H	H
H	X	X	(Z)
X	H	X	(Z)

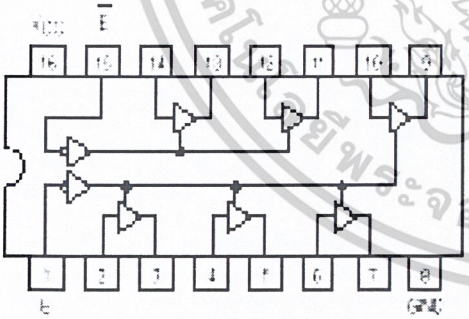
SN54/74LS366A
HEX 3-STATE INVERTER BUFFER
WITH COMMON 2-INPUT NOR ENABLE



TRUTH TABLE

INPUTS			OUTPUT
$\overline{E_1}$	$\overline{E_2}$	D	
L	L	L	H
L	L	H	L
H	X	X	(Z)
X	H	X	(Z)

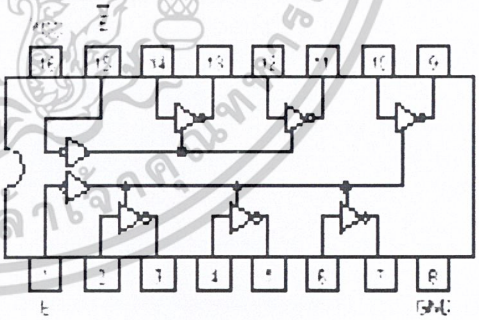
SN54/74LS367A
HEX 3-STATE BUFFER
SEPARATE 2-BIT AND 4-BIT SECTIONS



TRUTH TABLE

INPUTS		OUTPUT
$\overline{E}$	D	
L	L	L
L	H	H
H	X	(Z)

SN54/74LS368A
HEX 3-STATE INVERTER BUFFER
SEPARATE 2-BIT AND 4-BIT SECTIONS



TRUTH TABLE

INPUTS		OUTPUT
$\overline{E}$	D	
L	L	H
L	H	L
H	X	(Z)

SN54/74LS365A • SN54/74LS366A
SN54/74LS367A • SN54/74LS368A

DC CHARACTERISTICS OVER OPERATING TEMPERATURE RANGE (unless otherwise specified)

Symbol	Parameter		Limits Min	Typ	Max	Unit	Test Conditions	
V_{IH}	Input HIGH Voltage		2.0			V	Guaranteed Input HIGH Voltage for All Inputs	
V_{IL}	Input LOW Voltage	54			0.7	V	Guaranteed Input LOW Voltage for All Inputs	
		74			0.8			
V_{IK}	Input Clamp Diode Voltage			0.65	−1.5	V	V_{CC} = MIN, I_{IN} = −18 mA	
V_{OH}	Output HIGH Voltage	54	2.4	3.4		V	V_{CC} = MIN, I_{OH} = MAX, V_{IN} = V_{IH} or V_{IL} per Truth Table	
		74	2.4	3.1		V		
V_{OL}	Output LOW Voltage	54, 74		0.25	0.4	V	I_{OL} = 12 mA	V_{CC} = V_{CC} MIN, V_{IN} = V_{IL} or V_{IH} per Truth Table
		74		0.35	0.5	V	I_{OL} = 24 mA	
I_{OZH}	Output Off Current HIGH				20	μA	V_{CC} = MAX, V_{OUT} = 2.7 V	
I_{OZL}	Output Off Current LOW				−20	μA	V_{CC} = MAX, V_{OUT} = 0.4 V	
I_{IH}	Input HIGH Current				20	μA	V_{CC} = MAX, V_{IN} = 2.7 V	
					0.1	mA	V_{CC} = MAX, V_{IN} = 7.0 V	
I_{IL}	Input LOW Current E Inputs				−0.4	mA	V_{CC} = MAX, V_{IN} = 0.4 V	
	D Inputs				−20	μA	V_{CC} = MAX, V_{IN} = 0.5 V Other D Input at 2.0 V	
					−0.3	mA	V_{CC} = MAX, V_{IN} = 0.4 V Both E Inputs at 0.4 V	
I_{OS}	Short Circuit Current (Note 1)		−40		−225	mA	V_{CC} = MAX	
I_{CC}	Power Supply Current LS365A, 367A				24	mA	V_{CC} = MAX	
	LS366A, 368A				21			

Note 1: Not more than one output should be shorted at a time, nor for more than 1 second.

AC CHARACTERISTICS (T_A = 25°C, V_{CC} = 5.0 V)

Symbol	Parameter	Limits LS365A/LS367A Min	Typ	Max	LS366A/LS368A Min	Typ	Max	Unit	Test Conditions
t_{PLH} t_{PHL}	Propagation Delay		10 9.0	15 22		7.0 12	15 18	ns	C_L = 45 pF, R_L = 667 Ω
t_{PZH} t_{PZL}	Output Enable Time		19 24	35 45		18 28	35 45	ns	
t_{PHZ} t_{PLZ}	Output Disable Time			30 35			30 35	ns	C_L = 5.0 pF

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยดีก็เพราะได้รับความกรุณาเมตตา และความอนุเคราะห์ จาก รศ.สุเธียร เกียรสุนธร อาจารย์ที่ปรึกษาโครงงานนี้ ที่คอยให้ความช่วยเหลือ คำแนะนำ ชี้แนะ พร้อมทั้งให้ข้อมูลต่างๆ ซึ่งเป็นประโยชน์ต่อการทำโครงงานนี้เป็นอย่างยิ่ง

ขอขอบพระคุณคณาจารย์ทุกท่านที่ได้ให้คำชี้แนะ คำปรึกษาและประสิทธิ์ประสาทวิชาความรู้ให้กับผู้จัดทำ

ขอขอบพระคุณภาควิชาวิศวกรรมระบบควบคุม สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่ได้เอื้อเฟื้อเรื่องเครื่องมือและอุปกรณ์ต่างๆ ในการทำโครงงานนี้ให้ประสบผลสำเร็จลุล่วงไปด้วยดี

ขอขอบพระคุณคุณพ่อ คุณแม่ และคุณพี่ของผู้จัดทำ ที่ได้อุปการะผู้จัดทำและเป็นกำลังใจให้แก่ผู้จัดทำ

ขอขอบคุณเพื่อนๆ พี่ๆ ทุกคนที่คอยให้คำปรึกษาและคำแนะนำต่างๆ ตลอดจนอุปกรณ์ในการทำงานมาโดยตลอด

ผู้จัดทำ

นาย พยุงศักดิ์ หลำรอด

นาย อานนท์ ศรีวิไลย

หนังสืออ้างอิง

1. Barden W,"The Z-80 Microcomputer Handbook" , Howard W. Sams Et Co.,Inc.,1979
2. ยืน ภู่วรวรรณ "ทฤษฎีและการประยุกต์ไมโครโปรเซสเซอร์ Z-80" , บริษัท ซีเอ็ดยูเคชั่น จำกัด พิมพ์ครั้งที่ 1 , 1989
3. พันจันทร์ ธนวัฒนเสถียร , คงสวัสดิ์ ลอรัตนเรืองกิต "ทฤษฎีและการประยุกต์ไมโครโปรเซสเซอร์" , บริษัท ซีเอ็ดยูเคชั่น จำกัด ,พ.ศ.2537
4. วิบูลย์ ชื่นแขก "ไมโครโปรเซสเซอร์" , สำนักพิมพ์สถาบันเทคโนโลยีพระจอมเกล้าพระนครเหนือ , พ.ศ.2532
5. บริษัท ซีเอ็ดยูเคชั่น จำกัด "คู่มือไอซีไมโครโปรเซสเซอร์และไอซีที่เกี่ยวข้อง" , บริษัท ซีเอ็ดยูเคชั่น จำกัด , พ.ศ.2536