ปีการศึกษา 2532

INTERFACING TECHNIQUE

โดย

นายไพฑูรย์ เดชไพฑูรย์กุล

นายชูศักดิ์ พุกกะพันธ์

นายอุทัย โคตรวงศ์

อาจารย์ที่ปรึกษา

รศ.กิตติ ตรีเศรษฐ

อ.ธนิตย์ ตรีสุวรรณวัฒน์

ปริญญานิพนธ์ปีการศึกษา 2532

เรื่อง ชุดทดลองอินเทอร์เฟส (INTERFACING TECHNIQUE)

ผู้จัดทำ. . . . .

1.นายไพฑูรย์ เดชไพฑูรย์กุล

2.นายชูศักดิ์ พุกกะพันธ์

3.นายอุทัย โคตรวงค์

รศ.กิตติ ตีรเศรษฐ อาจารย์ที่ปรึกษา

(. . . . .)

นายธนิศย์ ตรีสุวรรณวัฒน์ อาจารย์ที่ปรึกษา

(. . . . .)

. . . . . อาจารย์ที่ปรึกษา

(. . . . .)

บทคัดย่อ

ปริญญานิพนธ์นี้ ได้ทำการทดลองออกแบบและสร้างชุดฝึก Interface ระหว่าง Single board Microcomputer ที่ใช้ Z-80 เป็น CPU กับ LED , DIP SWITCH LED SEVEN SEGMENT , DOT MATRIX , STEPPING MOTOR , DC MOTOR , KEYBOARD และการ INTERRUPT CPU เพื่อศึกษาการเขียนโปรแกรม Soft ware ในลักษณะ Machine Code และ Assembly Langauge โดยสามารถที่จะติดต่อกับอุปกรณ์ Peripheral ต่าง ๆ จากที่กล่าวมาแล้วให้เน้น การปฏิบัติทดลองเพื่อจักเกิดทักษะการเรียนรู้ได้จากชุดดลอง และมีใบงานประกอบการทดลอง 10 ใบงานพร้อมทฤษฎีบทนำ ชุดทดล่องนี้ได้ออกแบบ ให้สามารถศึกษาถึงหลักการของ Micro Processor ในการติดต่อกับอุปกรณ์ต่าง ๆ เพื่อที่จะนำไปประยุกต์ใช้งานในการควบคุมได้กับงานจริงในอุตสาหกรรมปัจจุบัน.

INTERFACING TECHNIQUE

BY...

Mr.Paitoon Dathpaitoonkul

Mr.Chusak Pukapan

Mr.Uthal Kotwong

ADVISOR

ASSOC. PROF. Kitti Tirasesth

Mr.Thanit Trisuwannawat

1989

ABSTRACT

This thesis is designed for training a course of Interfacing between Single Board Microcomputor and LED: Dip switch, Led seven Segment, Dot Matrix, Stepping Motor, DC Motor, Key Board and Interrupt CPU.

This training program is used to study Software in Machine Code and Assembly Language including 10 Experiments.

This training is designed for studying a principle of microprocessor and Interfacing technique which can be applied for Industrail Control for today.

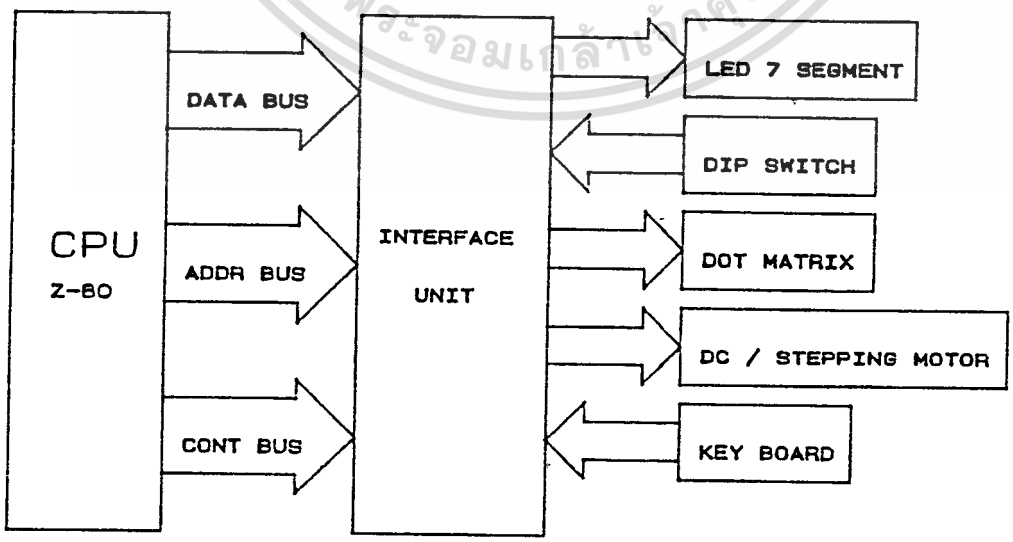
สารบัญ

บทที่ 1
บทนำ

ปัจจุบันนี้มนุษย์เรากำลังก้าวเข้าสู่ยุคของไมโครอิเล็กทรอนิกส์ ซึ่งเป็นปัจจัยที่ 5 ในการดำรงชีวิตในสังคมทุกวันนี้โดยไม่รู้ตัว เช่น เครื่องใช้ไฟฟ้า วิทยุ โทรศัพท์ รถยนต์ อื่น ๆ จะเป็นระบบกึ่งอัตโนมัติมีอุปกรณ์ไมโครอิเล็กทรอนิกส์เข้าไปร่วมด้วยเกือบทุกอย่าง สิ่งที่เป็นหัวใจในการควบคุมระบบอัตโนมัตินี้ คือ Microprocessor หรือ TTL, CMOS, อื่น ๆ นั่นเอง ดังนั้นเป็นสิ่งซึ่งจำเป็นสำหรับผู้ที่กำลังศึกษา จะต้องก้าวให้ทันกับเทคโนโลยีใหม่นี้ตลอดเวลา

ดังนั้น ปริญญานิพนธ์ฉบับนี้จึงขอเสนอเอกสาร และชุดทดลองประกอบใบงานการ Interface ระหว่าง Z - 80 ไมโครโปรเซสเซอร์ กับ อุปกรณ์ I/O ต่าง ๆ เพื่อใช้ในการเรียนการสอนของสถานศึกษา สังกัดกรมอาชีวะศึกษา กระทรวงศึกษาธิการโดยมุ่งเน้น ในการรู้จักนำเอาเทคโนโลยีสมัยใหม่มาใช้ ความสัมพันธ์ของชุดทดลองนี้กับหลักสูตรสาขาวิชาชีพชั้นสูง (ปวส) ในหัวข้อวิชา เทคนิคการอินเทอร์เฟส ไมโครคอมพิวเตอร์(MICROCOMPUTER INTERFACE TECHNIQUE) ซึ่งมีเนื้อหาเกี่ยวกับการอินเทอร์เฟส โครงสร้างอินพุท-เอาท์พุท,การอินเทอร์รัพท์ CPU , การเชื่อมต่อแบบขนาน, การเชื่อมต่อกับหน่วยความจำ, การเชื่อมต่อกับอุปกรณ์ต่าง ๆ เพื่อให้สามารถนำจุดทดลองนี้ มาใช้กับการเรียนการสอนตามหลักสูตรดังกล่าว

โครงสร้างชุดทดลอง MICROCOMPUTER INTERFACING TECHNIQUE ประกอบด้วยรายละเอียดดังนี้



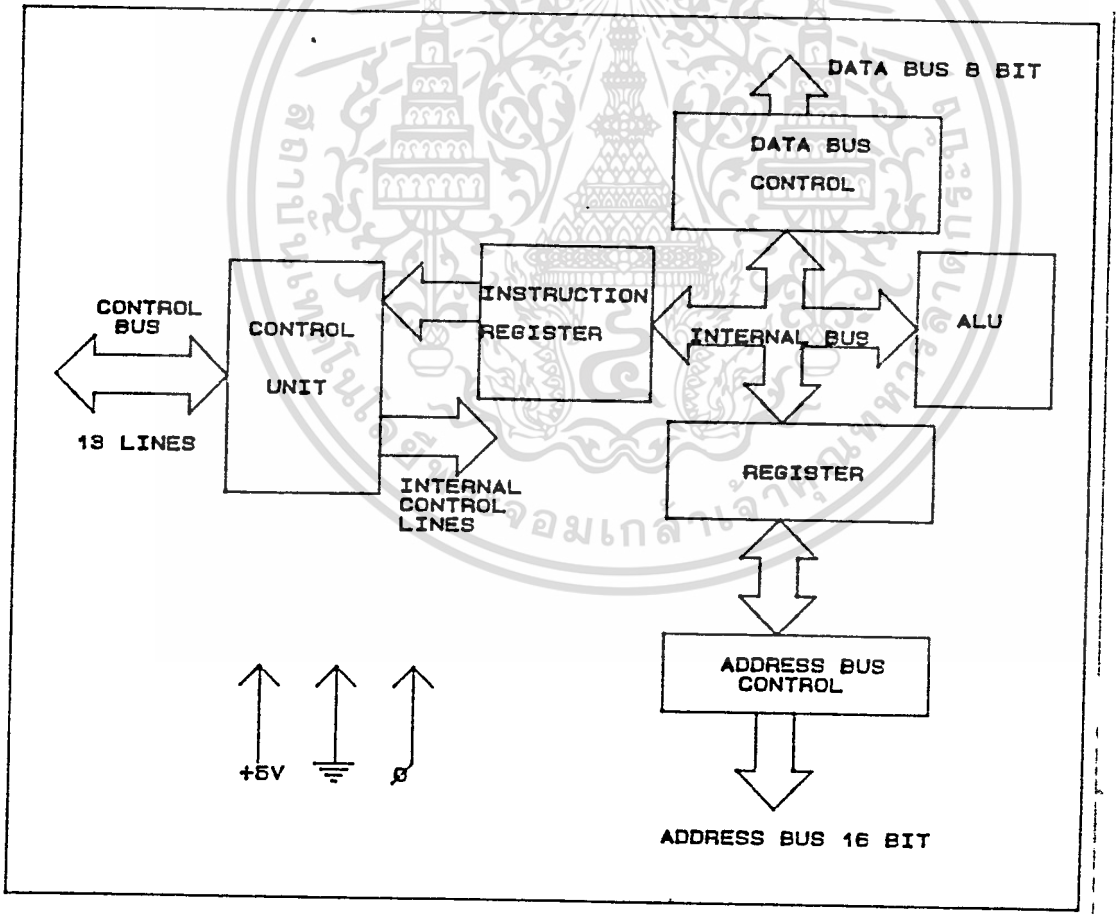
1. BASIC INPUT /OUTPUT PORT โดยใช้ IC 74373 จำนวน 2 ตัว คือ U1 # 74373 ให้ OUTPUT PORT DO ใช้ขับ LED 8 ดวง
   ส่วนอีกตัวคือ U2.#74373 เป็น INPUT PORT CO รับ DATA จาก DIP SWITCH
2. การใช้ IC 8255 PIA เป็น ไอซี ชิพเฉพาะในการ INTERFACE กับ INPUT/ OUTPUT PORT ต่าง ๆ จำนวน 2 ตัว
   ตัวที่ 1 (U3) ใช้ PORT EO ทำหน้าที่ ขับ DOT MATRIX
   ขับ DC MOTOR
   ขับ STEPPING MOTOR
   ตัวที่ 2 (U4) ใช้ PORT FO ทำหน้าที่ ขับ LED SEVEN SEGMENT และ SCAN KEYBOARD
3. ใช้IC 74LS138 ทำหน้าที่ DECODER เลือก PORT ต่าง ๆ ในการติดต่อกับอุปกรณ์ต่าง ๆ
4. BUFFER DATA ใช้ IC เบอร์ 74LS245 เป็น BIDIRECTIONAL BUFFER เพื่อให้อุปกรณ์ภายนอกดึงกระแสได้มาก

2.1 ไมโครโปรเซสเซอร์ Z-80

ไมโครเซสเซอร์ในปัจจุบันได้พัฒนาให้ก้าวหน้าตลอดเวลา โดยเฉพาะอย่างยิ่งในเรื่องของการประยุกต์ใช้งาน อย่างไรก็ตามหลักการทำงานของไมโครโปรเซสเซอร์ทุกเบอร์ก็มีหลักการคล้ายกัน จะแตกต่างกันก็เฉพาะในเรื่องของสัญญาณและชุดคำสั่ง ดังนั้นการเรียนรู้เกี่ยวกับไมโครโปรเซสเซอร์ ไม่ว่าจะเป็นเบอร์ใดก็ตามสามารถที่จะดัดแปลงให้ใช้งานแทนกันได้

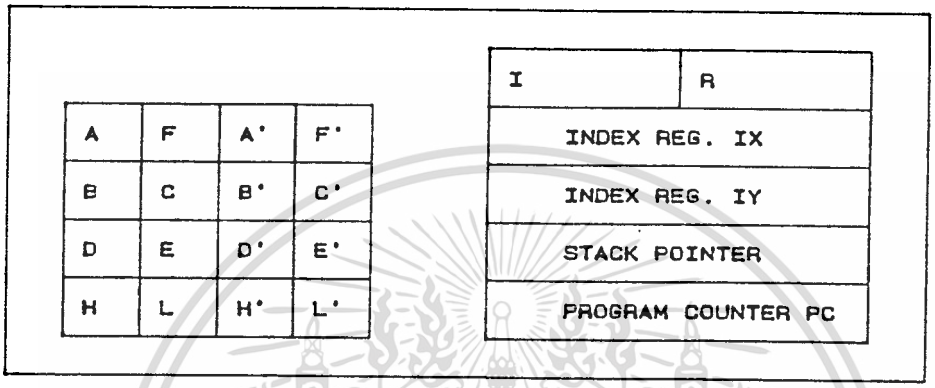
ไมโครโปรเซสเซอร์ Z-80 มีลักษณะพิเศษทางฮาร์ดแวร์หลายประการ เช่น มีโครงสร้างที่มีความสามารถรวมอยู่ในชิปตัวเดียว ใช้อัตราของสัญญาณนาฬิกาสูงถึง 4 MHz ในปัจจุบันสามารถเพิ่มอัตราความเร็วของสัญญาณนาฬิกาได้สูงกว่า 8 MHz ใช้แหล่งจ่ายไฟเลี้ยงเพียงชุดเดียว มีอุปกรณ์สำหรับอินเตอร์เฟสข้อมูลทั้งแบบอนุกรมและแบบขนาน.

โครงสร้างของ Z-80 โดยโครงสร้างของซีพียูนี้จะบรรจุลงในแอลเอสไอขนาด 40 ขา



รูป บล็อกไดอะแกรมซีพียู Z - 80

โครงสร้างภายในของ Z - 80 ซีพียูประกอบด้วยรีจีสเตอร์ภายในที่สามารถเขียนและอ่านได้ถึง 208 บิท โดยแยกเป็นกลุ่มของรีจีสเตอร์ขนาด 8 บิท และรีจีสเตอร์ขนาด 16 บิท อีก 4 รีจีสเตอร์ โดยมีชุดรีจีสเตอร์แสดงได้ดังรูป



รูป บล็อกไดอะแกรมซีพียู Z - 80

<u>รีจิสเตอร์หลักที่ใช้งานทั่วไป</u>

รีจีสเตอร์ในกลุ่มแรกคือ A,F,B,C,D,E,H,L เป็นรีจิสเตอร์ขนาด 8 บิท ที่ใช้งานทั่วไป โดยรีจิสเตอร์เหล่านี้สามารถประกอบรวมกันเป็นคู่รีจิสเตอร์ได้ คือ AF,BC,DE และ HL โดยคู่รีจิสเตอร์เหล่านี้จะได้รับการใช้งานในลักษณะของรีจิสเตอร์ขนาด 16 บิท การกระทำภายในซีพียูอาจจะอาศัยเพียงรีสิจเตอร์เดียวหรือกระทำเป็นคู่รีจิสเตอร์ได้ โดยที่ A คือ แอคคิวมูเลเตอร์ F คือ แฟลก แฟลกของ Z - 80 จะมีด้วยกันทั้งหมด 6 ตัว จึงใช้เพียง 6 บิท แต่ Z - 80 อาศัยการเพิ่มเติมบิทขึ้นอีก 2 บิท และกลายเป็นรีจิสเตอร์ F รีจิสเตอร์ F นี้สามารถได้รับการเซท รีเซทการกระทำตามคำสั่งทางคณิตศาสตร์ หรือลอจิกได้ และเราสามารถใช้ F เหมือนรีจิสเตอร์ตัวหนึ่ง ซึ่งเมื่อรวมกันกับ A แล้วจะกลายเป็นรีจิสเตอร์ขนาด 16 บิทได้

<u>กลุ่มรีจิสเตอร์สำรอง</u>

เป็นกลุ่มรีจิสเตอร์ที่สามารถเก็บข้อมูลได้ โดยเป็นตัวเก็บข้อมูลที่มาจากรีจิสเตอร์หลัก รีจิสเตอร์ชุดนี้จึงมีด้วยกัน 8 ตัว A',F',B',C',D',E',H',L' รีจิสเตอร์เหล่านี้เป็นรีจิสเตอร์ที่ใช้ในการเก็บข้อมูลชั่วคราว ในการที่ต้องการใช้รีจิสเตอร์หลักทำงานอย่างอื่นก่อน ดังนั้น รีจิสเตอร์กลุ่มนี้จึงไม่สามารถกระทำทางคณิตศาสตร์และลอจิกได้

กลุ่มรีจิสเตอร์ที่ใช้งานเฉพาะอย่าง

โปรแกรมเคาวน์เตอร์ (PC-Program Counter) โปรแกรมเคาวน์เตอร์เป็นรีจิสเตอร์ขนาด 16 บิทที่เป็นตัวกำหนดตำแหน่งของโปรแกรมในขณะสภาวะการกระทำการเฟทช์ โดยขณะทำการเฟทช์ค่าที่อยู่ในโปรแกรมเคาวน์เตอร์จะไปปรากฏอยู่ที่แอดเดรสบัสเพื่อชี้ไปยังตำแหน่งในหน่วยความจำให้ซีพียูอ่านคำสั่งมาดีความหมาย ค่าที่อยู่ในโปรแกรมเคาวน์เตอร์จะเพิ่มค่าขึ้นได้อย่างอัตโนมัติหลังการกระทำการเฟทช์ แต่ถ้าหากซีพียูกระทำคำสั่งให้ข้ามไปยังตำแหน่งอื่น (Jump) ค่าแอดเดรสที่จะกระโดดข้ามนั้นจะโหลดเข้ามายังโปรแกรมเคาวน์เตอร์ได้อย่างอัตโนมัติ

สแตคพอยท์เตอร์ (SP-Stack pointer) เป็นรีจิสเตอร์ที่มีขนาด 16 บิท ที่ใช้สำหรับชี้ไปยังแอดเดรสชั้นบนสุดของสแตคที่อยู่ใน RAM โดยส่วนของสแตคมีลักษณะโครงสร้างเป็นหน่วยความจำเป็นแบบเก็บทีหลังเรียกออกใช้ได้ก่อนข้อมูลในสแตคอาจได้รับการพุชหรือพอพมาจากข้อมูลรีจิสเตอร์ภายในซีพียู ลักษณะของสแตคที่นี้ยังเป็นส่วนช่วยในการกระทำอินเตอร์รัพท์ และการเรียกโปรแกรมย่อย กล่าวคือในการอินเตอร์รัพท์ค่าของโปรแกรมเคาวน์เตอร์จะได้รับการเก็บรักษาไว้ในชั้นสแตค ครั้งเมื่อโปรแกรมกลับจากอินเตอร์รัพท์ไปกระทำยังโปรแกรม หลักก็จะนำค่าจากสแตคกลับเข้ามายังโปรแกรมเคาวน์เตอร์ใหม่ ในทำนองเดียวกันการกระโดดไปกระทำยังโปรแกรมย่อย ก็เช่นเดียวกัน ดังนั้นการกระทำในรูปของอินเตอร์รัพท์หรือโปรแกรมย่อย สามารถซ้อนกันได้ไม่มีสิ้นสุด

อินเดครีจิสเตอร์ (IX,IY-index register) ซีพียู Z-80 มี อินเดครีจิสเตอร์ขนาด 16 บิท 2 ตัว แต่ละตัวใช้ประโยชน์หลักในการทำหน้าที่เป็นตัวเก็บแอดเดรสฐาน (base address) เพื่อทำหน้าที่อ้างแอดเดรสแบบอินเดคแอดเดรสชิ่ง (index addressing) ในโหมดของอินเดคแอดเดรสซึ่งมีข้อมูลที่อยู่ในอินเดครีจิสเตอร์นี้จะรวมกับข้อมูลที่ติดมากับคำสั่งอีก 8 บิท เพื่อเป็นตัวกำหนดแอดเดรสให้กับคำสั่งข้อมูลที่ติดมากับคำสั่งนี้เราเรียกว่าดิสเพลซเมนท์ (displacement) ซึ่งจะเก็บในรูปของตัวเลข 2's คอมพลีเมนท์

อินเตอร์รัพท์เพจแอดเดรสรีจิสเตอร์ (I-Interrupt pagee register) การอินเตอร์รัพท์ของ Z-80 มีหลายโหมด และโหมดหนึ่งที่ทำให้กจารอินเตอร์รัพท์ของ Z-80 มีประสิทธิภาพสูง กล่าวคือ เมื่อเกิดการอินเตอร์รัพท์ในโหมดนี้มันสามารถอ้างแอดเดรสโดยทางอ้อมไปกระทำโปรแกรมในที่ใดก็ได้ในหน่วยความจำ โดยอาศัยค่าในรีสเตอร์ I รวมกับค่าที่มาจากอุปกรณ์เพอริเฟอรัลอีก 8 บิท ชี้ไปยังค่าในหน่วยความจำ เพื่อนำค่านั้นมาโหลดเข้าไปในโปรแกรมเคาวน์เตอร์เพื่อกระทำต่อไป ด้วยวิธีการนี้เราจึงสามารถกระโดดเข้าไปทำที่ส่วนใดก็ได้ในหน่วยความจำ

รีจิสเตอร์รีเฟรชหน่วยความจำ (R-memory reflesh register) การต่อซีพียู

กับหน่วยความจำนั้น โดยแล้วปกติจะต่อกับหน่วยความจำชนิดสแตติคได้โดยง่าย แต่อย่างไรก็ดี ชนิดไดนามิคที่ต้องการรีเฟรชมีราคาถูกกว่า มีความหนาแน่นสูงกว่า Z-80 ให้ข้อดีว่าประการหนึ่งคือ มันสามารถให้การรีเฟรชหน่วยความจำได้อย่างอัตโนมัติ โดยค่าใน R นี้จะส่งออกไปยังแอดเดรสบัสในส่วนบิทที่มีนัยสำคัญต่ำกว่าจังหวะของการส่งนี้จะเป็นจังหวะเดียวกันกับที่ซีพียูส่งสัญญาณรีเฟรชออกมา ผู้โปรมแกรมสามารถกำหนดค่าให้กับรีจิสเตอร์ R นี้ได้ แต่ค่าในรีจิสเตอร์นี้จะเรียกใช้โดยผู้โปรแกรมทางคำสั่งโดยตรงไม่ได้

แอคคิวมูเลเตอร์ (accumulator) และแฟลก (flag) ซีพียูจะมีรีจิสเตอร์ที่ใช้เป็นหลักในการเป็นตัวโอเปอร์แรนด์สำหรับกระทำทางานคณิตศาสตร์และลอจิก โดยรีจิสเตอร์หลักนี้จะมีเพียง 8 บิท เรียกว่า "แอคคิวมูเลเตอร์" (accumulator) การกระทำในส่วนของหน่วยคณิตศาสตร์และลอจิกย่อมเกิดเงื่อนไขได้หลายอย่างที่จะต้องแสดงสถานะภาพของเงื่อนไขเหล่านั้น เช่น เงื่อนไขผลลัพธ์เป็นศูนย์ ผลลัพธ์เป็นบวกหรือลบ มีตัวทดหรือตัวขอยืมในการกระทำทางคณิตศาสตร์ แสดงเงื่อนไขพาริตี้คู่หรือคี่ ฯลฯ สิ่งเหล่านี้จะให้ผลลัพธ์แสดงสถานะได้ด้วยแฟลก (flag) เป็นริจิสเตอร์ขนาด 8 บิท ซึ่งสามารถรวมกับแอคคิวมูเลเตอร์ขนาด 16 บิทได้ ผู้โปรแกรมยังสามารถใช้คำสั่งในการเคลื่อนย้ายข้อมูลจากแอคคิวมูเลเตอร์ A และแฟลก F ไปเก็บไว้ใน A และ F ได้เพื่อทำให้การใช้งานของ A และ F มีประสิทธิภาพดียิ่งขึ้น

หน่วยคำนวณทางคณิตศาสตร์ (ALU - Arithmetic and Logic Unit) การประมวลผงที่สำคัญของซีพียูของคอมพิวเตอร์ยังขึ้นอยู่กับหน่วยคำนวณทางคณิตศาสตร์และลอจิก (ALU) บางส่วน ALU นี้จะนำข้อมูลซึ่งอาจจะมาจากภายนอกซีพียูหรือภายในซีพียูก็ได้มาประมวลผล การประมวลผลในส่วน ALU ที่สำคัญจะประกอบด้วย

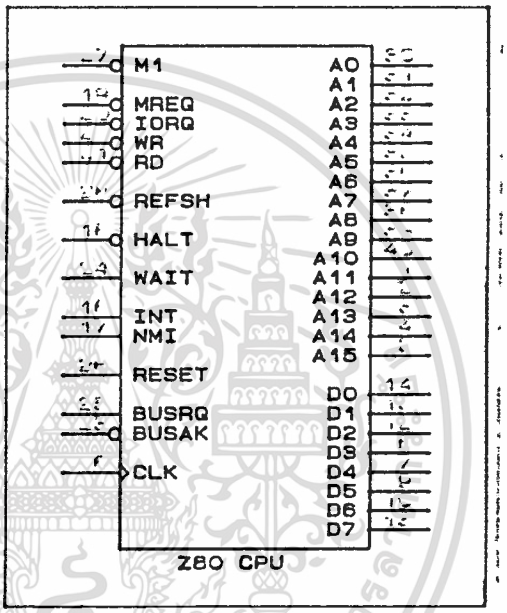
การบวก (add)	การเลื่อนบิททางซ้ายหรือขวา
การลบ (substract)	การเพิ่มค่า (increment)
ลอจิก AND	การลดค่า (decrement)
ลอจิก EX-OR	การเซทบิท (set-bit)
ลอจิก EX-OR	การรีเซทบิท (reset-bit)
เปรียบเทียบ (compare)	การทดสอบบิท (test bit)

รีจิสเตอร์คำสั่งและส่วนควบคุม (Instruction register and control) ในการกระทำการเฟทซ์พียูจะอ่านคำสั่งจากหน่วยความจำที่เป็นส่วนของโปรแกรมโดยรอคำสั่งนั้นมาเก็บไว้ใน IR เพื่อทำการอ่านรหัสคำสั่งและส่งสัญญาณควบคุมการทำงานภายในซีพียู หรือควบคุมการทำงานของระบบสัญญาณควบคุมเหล่านั้นจะออกมาในจังหวะต่าง ๆ กัน เพื่อใช้ควบคุมระบบในการทำงานต่อไป

การจัดขาของ Z-80

Z-80 ซีพียูเป็นไอซีไมโครโปรเซสเซอร์มีขาเพียง 40 ขา โดยหลักการแล้ว Z-80 เป็นซีพียูได้โดยสมบูรณ์ กล่าวคือ Z-80 ไม่ต้องประกอบกับอุปกรณ์ประกอบอื่นที่จะแยกการทำงานเพื่อรวมเป็น ซีพียู ส่วนของสัญญาณจะประกอบด้วยบัสแอสเดรส บัสข้อมูล และสัญญาณควบคุม

รายละเอียดของขาต่าง ๆ แสดงได้ดังนี้



A0-A15 บัสแอดเดรส สัญญาณที่ออกมาจากขาไอซีเหล่านี้จะให้แอดดีพซถะโดย ขาเหล่านี้เป็นเอาท์พุทแบบไตรสเตท บัสแอดเดรสมีด้วยกันทั้งหมด 16 สายเพื่อให้ซีพียูติดต่อกับหน่วยความจำได้มากถึง 2^{16} = 64 Kไบท์ นอกจากนี้ส่วนของแอดเดรสยังเป็นตัวกำหนดเบอร์พอร์ทของอุปกรณ์อินพุท-เอาท์พุทโดยขณะที่ซีพียูกระทำคำสั่งเกี่ยวกับอินพุทหรือเอาท์พุท ค่าของแอดเดรสบัสใน 8 บิทล่าง (A0-A7) จะแสดงค่าเบอร์พอร์ท ดังนั้นเราจึงมีอุปกรณ์อินพุทหรือเอาท์พุทได้ทั้งหมด 2^{8} = 256 พอร์ท และในขณะช่วงเวลารีเฟรช โดยเมื่อสัญญาณรีเฟรชปรากฏขึ้นที่ขารีเฟรชปรากฏขึ้นที่ขารีเฟรช (RFSH) ค่าในแอดเดรสบัส A0-A7 จะแสดงค่าแอดเดรสของหน่วยความจำที่ได้รับการกระทำการรีเฟรช

D0-D7 บัสข้อมูล (data bus) เป็นลักษณะบัสแบบสองทิศทาง Z-80 ซีพียู มีบัสข้อมูล 8 เส้น บัสข้อมูลเป็นเส้นทางผ่านของข้อมูลระหว่างซีพียู

กับหน่วยความจำ ซีพียูกับอุปกรณ์อินพุท-เอาท์พุท หรือการติดต่อระหว่างอุปกรณ์อินพุท-เอาท์พุทกับหน่วยความจำ

$\overline{M1}$ (machine cycle one) มีลักษณะเป็นแอคตีฟที่ลอจิก "0" $\overline{M1}$ เป็นส่วนที่จะบอกให้ทราบว่าขณะนี้ซีพียูกำลังอยู่สภาวะเฟทช์ ในขณะที่ซีพียูคำสั่งที่มีออพโค้ดสองไบท์ส่วนของ $\overline{M1}$ จะสร้างขึ้นขณะเฟทช์ในแต่ละไบท์ ลักษณะของคำสั่งที่มีออพโค้ด สองไบท์จะขึ้นต้นด้วย CBH,DDH,EDH,FDH,(H ต่อท้ายหมายถึงตัวเลขฐานสิบหก) นอกจากนี้ $\overline{M1}$ ยังสร้างสัญญาณร่วมกับ IORQ เพื่อบอกสถานะการตอบรับการอินเตอร์รัพท์

$\overline{MREQ}$ (memory request)เป็นลักษณะไตรสเตท ให้ลอจิกแอคตีฟที่ "0" เป็นสายสัญญาณที่บอกให้รู้ว่า ซีพียูต้องการเขียนหรืออ่านหน่วยความจำตามแอดเดรสที่ปรากฏอยู่ในแอดเดรสบัส

$\overline{IORQ}$ (input output request) เป็นเอาท์พุทลักษณะไตรสเตทให้ลอจิกเป็นลักษณะแอคตีฟที่ "0" เป็นสัญญาณที่บอกให้ทราบว่า ซีพียูต้องการติดต่อกับอุปกรณ์อินพุท–เอาท์พุทโดยแอดเดรสบัส 8 บิทล่างจะให้แสดงค่าเบอร์พอร์ท ส่วนบัสข้อมูลจะแสดงข้อมูลที่จะมีการส่งถ่ายระหว่างซีพียูกำลังตอบสนองผลการอินเตอร์รัพท์ โดยขณะนี้ส่วนของบัสข้อมูลจะมีการส่งผ่านเข้ามาด้วยค่าของอินเตอร์รัพท์เวคเตอร์

$\overline{RD}$ (memory read) เป็นเอาท์พุทไตรสเตทและแอคตีฟขณะลอจิก"0" $\overline{RD}$ เป็นตัวบอกว่าขณะนี้ซีพียูต้องการอ่านข้อมูลจากหน่วยความจำหรืออุปกรณ์ I/O

$\overline{WR}$ (memory write) เป็นขาเอาท์พุทแบบไตรสเตท และแอคตีฟขณะลอจิก"0" $\overline{RD}$ เป็นสัญญาณบอกว่าซีพียูต้องการเขียนข้อมูลโดยจะเขียนข้อมูลในตำแหน่งแอดเดรสบัสกำหนดขึ้นอาจเป็นหน่วยความจำหรืออุปกรณ์ I/O ก็ได้

$\overline{RFSH}$ (refresh) เป็นขาเอาท์พุทแอคตีฟขณะลอจิก"0" $\overline{RFSH}$ เป็นสัญญาณที่จะบอกให้ทราบว่าขณะนี้สัญญาณในแอดเดรสบัสในส่วน A0 - A6 เป็นแอดเดรสที่จะใช้ในการรีเฟรสหน่วยความจำชนิดไดนามิกส์ส่วน A7 จะเป็น "0" ส่วนบิท A16 - A8 จะแสดงค่ารีจิจเตอร์ I

$\overline{HALT}$ (halt state) เป็นเอาท์พุทที่แอคตีฟด้วยลอจิก"0" สัญญาณ $\overline{HALT}$ จะแสดงเมื่อซีพียูได้กระทำคำสั่ง $\overline{HALT}$ และจะหยุดรอจนกว่าจะมีการ

อินเตอร์รัพท์หรือรีเซท ขณะที่อยู่ในช่วง $\overline{\text{HALT}}$ ซีพียูจะเสมือนกำลังกระทำคำสั่ง NOP (no operation) เพื่อทำให้เกิดไซเกิ้ลในการทำงานเพื่อส่งสัญญาณไปกระทำการรีเฟรชหน่วยความจำชนิดไดนามิกส์

$\overline{\text{WAIT}}$ (wait) เป็นขาอินพุทจะแอคตีฟด้วยลอจิก"0" $\overline{\text{WAIT}}$ เป็นตัวกำหนดแสดงเพื่อบอกซีพียูให้หยุดรอ ในกรณีที่อินพุท - เอาท์พุท หรือหน่วยความจำไม่สามารถรับส่งข้อมูลได้ทันที $\overline{\text{WAIT}}$ จะเป็นตัวทำให้ซีพียูซิงค์ได้พอดีกับอุปกรณ์อินพุทเอาท์พุทที่ทำงานที่ความเร็วช้าๆ

$\overline{\text{INT}}$ (interrupt request) เป็นขาอินพุท แอคตีฟด้วยลอจิก "0" $\overline{\text{INT}}$ เป็นสัญญาณที่สร้างขึ้นจากอุปกรณ์อินพุทเอาท์พุท เพื่อต้องการที่จะอินเตอร์รัพท์ซีพียู ซีพียูจะทำการตรวจสอบสัญญาณนี้ทุก ๆ ครั้งที่จบการกระทำแต่ละครั้งแต่ละคำสั่ง การตอบสนองของตัวการอินเตอร์รัพท์สามารถควบคุมได้ด้วยซอฟแวร์ ด้วยการเซทค่าอินเตอร์รัพท์ฟลิบฟลอบ (IFF) การตอบสนองอินเตอร์รัพท์จะเกิดได้ยังต้องให้ $\overline{\text{BUSRQ}}$ ไม่แอคตีฟเมื่อซีพียูตอบสนองการอินเตอร์รัพท์ ซีพียูจะสร้างสัญญาณตอบด้วยการสร้างสัญญาณ $\overline{\text{IORQ}}$ ระหว่างช่วงเวลา $\overline{\text{MI}}$ การตอบสนองต่อการอินเตอร์รัพท์มีแยกแยะได้ 3 แบบ

$\overline{\text{NMI}}$ (nonmaskable interrput) เป็นขาอินพุท ที่จะทริกบอกซีพียูในขณะขอบพลัซ์ขาลง การอินเตอร์รัพท์ด้วยวิธีนี้ ซีพียูจะให้ความสำคัญสูงกว่า $\overline{\text{INT}}$ กล่าวคือมันจะตอบสนองและกระทำทันทีด้วยการเริ่มเอ็ก–ซีคิ้ว คำสั่งในตำแหน่ง 0066 H โดยอัตโนมัติ การกระโดดไปกระทำในกรณีนี้ ซีพียูจะเก็บค่าโปรแกรมเคาว์เตอร์เดิมไว้ในสแตค เพื่อจะได้กลับไปทำงานเดิมเมื่อเสร็จสิ้นการอินเตอร์รัพท์ได้

$\overline{\text{RESET}}$ (reset) เป็นขาอินพุทที่แอคตีฟด้วยลอจิก "0" การรีเซตในกรณีนี้จะมีผลดังนี้

1. ค่าของ PC มีค่าเป็น "0"
2. IFF จะได้รับการ Disable
3. รีจิสเตอร์ I จะมีค่า 00H
4. รีจิสเตอร์ R จะมีค่า 00H
5. จะมีการเซทอินเตอร์รัพท์โหมดมาอยู่ที่โหมด 0

ระหว่างการรีเซทสายแอดเดรสบัสข้อมูลจะได้รับการกระทำให้มีค่าอิมพีแดนซ์สูง เพื่อแยกออกจากซีพียู ส่วนสายสัญญาณควบคุมจะได้รับ

การทำให้เป็นสัญญาณไม่แอคตีฟการรีเฟรชจะไม่เกิดขึ้น

$\overline{BUSRQ}$ (bus request) เป็นขาอินพุทที่แอคตีฟด้วยลอจิก "0" $\overline{BUSRQ}$ เป็นสัญญาณที่ส่งบอกกับซีพียู เพื่อจต้องการให้ซีพียูควบคุมบัส กล่าวคือต้องการให้ซีพียูควบคุมบัส กล่าวคือต้องการให้ซีพียูทำให้บัสแอดเดรสและบัสข้อมูลอยู่ในสถานะอิมพีแดนซ์สูง คือต้องการแยกซีพียูออกจากบัสนั่นเอง

$\overline{BUSAK}$ (bus acknowledge) เป็นขาเอาท์พุท แอคตีฟขณะลอจิก "0" $\overline{BUSAK}$ เป็นสัญญาณตอบจากซีพียูว่าซีพียูได้แยกตัวเองออกจากแอดเดรสบัสและบัสข้อมูลเรียบร้อยแล้ว

0 (clock) สัญญาณนาฬิกาที่จะป้อนเข้าระบบ

ไดอะแกรมเวลาของซีพียู

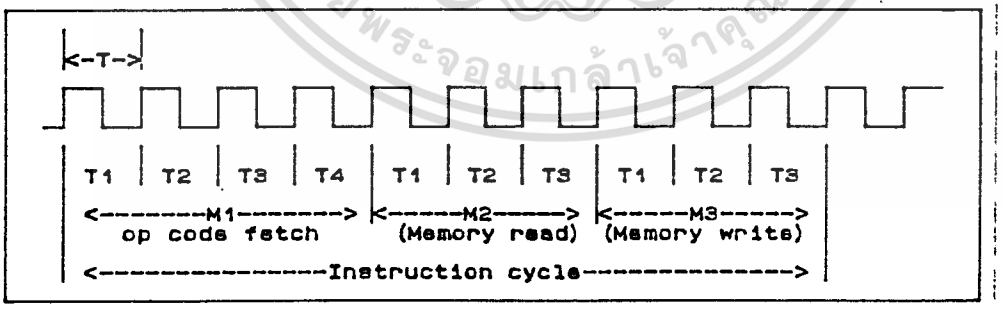
Z-80 ซีพียูจะทำงานในลักษณะพื้นฐานที่สำคัญประกอบด้วย

-การเขียน-อ่านหน่วยความจำ

-การเขียน-อ่านอุปกรณ์อินพุท-เอาท์พุท

-การตอบสนองต่อการอินเตอร์รัพท์

ลักษณะการทำงานจะสัมพันธ์กับสัญญาณนาฬิกา ของให้พิจารณาจากรูป



2.2 Z80 กับ อินพุท และ เอาท์พุท

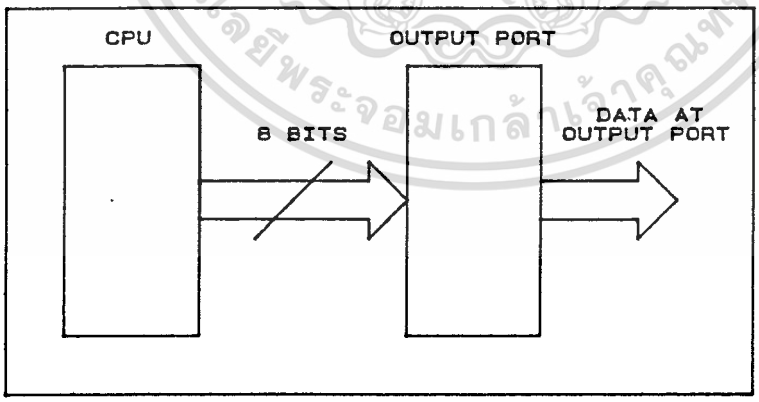
ในขบวนการทำงานของการ INTERFACE ในระบบไมโครโปรเซสเซอร์นั้น มีความจำเป็นอย่างยิ่งที่จะต้องทำการติดต่อกับอุปกรณ์อื่น ๆ เสมอนอกเหนือจากหน่วยความจำ แล้วในบางครั้ง CPU ยังมีความจำเป็นที่จะต้องทำการอ่านข้อมูลจากอุปกรณ์อินพุทและส่งข้อมูลออกไปทางอุปกรณ์เอาท์พุท จะศึกษาวิธีการเชื่อมต่อระหว่าง Z80 เข้ากับอุปกรณ์ I/O (INPUT/OUTPUT DEVICE) นอกจากนี้จะกล่าวถึงการสร้างพอร์ท I/O (INPUT/OUTPUT DEVICE) ทั่ว ๆ ไป โดยใช้อุปกรณ์ประเภท DISCRETE.

ขบวนการอินพุทและเอาท์พุท เนื่องจากการติดต่อกับอุปกรณ์ I/O นั้น มีลักษณะคล้ายคลึงกับการติดต่อกับ STATIC RAM จะเห็นว่าขบวนการติดต่อระหว่างCPU กับSTATIC RAM และการติดต่อระหว่างCPU กับ อุปกรณ์ I/O นั้นมีลักษณะคล้ายกันมาก

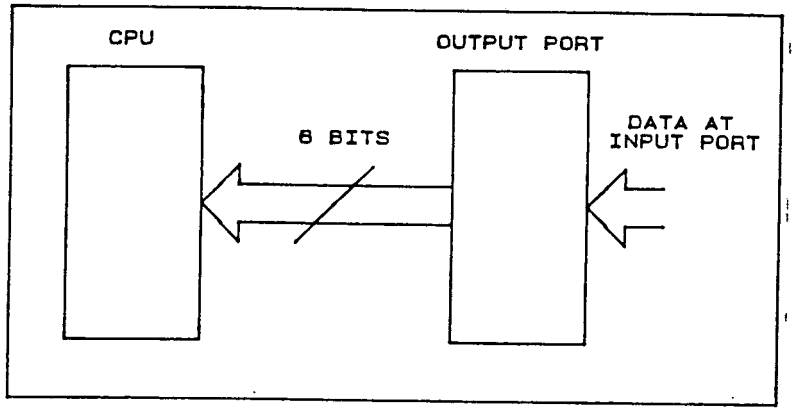
2.2.1 ขบวนการอินพุทและเอาท์พุทของ Z80.

สัญญาทางไฟฟ้าที่ใช้ในระบบฮาร์ดแวร์ในขณะที่ Z80 ทำการติดต่อกับอุปกรณ์ I/O นั้นจะประกอบไปด้วย สัญญาณจากบัสแอดเดรส (ADDRESS BUS). สัญญาณจากบัสข้อมูล (DATA BUS) และสัญญาณจากบัสควบคุม (CONTROL BUS)

รูป 2.2.1A และ 2.2.1B จะแสดงบล๊อกไดอะแกรมของการส่งข้อมูลขนาด 8 บิทให้กับอุปกรณ์เอาท์พุทและรับข้อมูลจากอุปกรณ์อินพุท



รูป 2.2.1A : แสดงบล๊อกไดอะแกรมของพอร์ทเอาท์พุท.



รูป 2.2.1B : แสดงบล็อกไดอะแกรมของพอร์ทอินพุท.

2.2.2 การบ่งบอกตำแหน่งของพอร์ท (PROT ADDRESS).

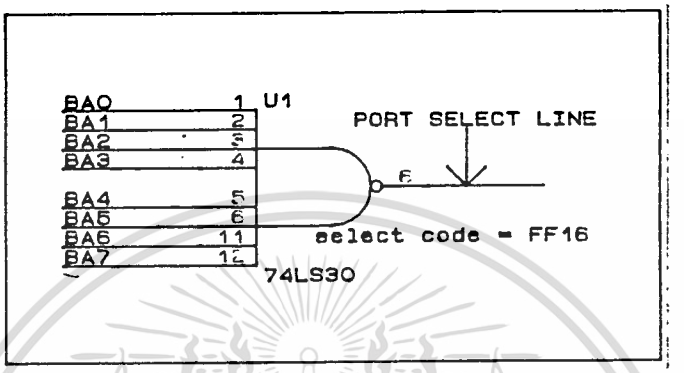
ในระบบของ Z80 นั้นจะมีบัสแอดเดรสอยู่ 16 เส้นแต่จะถูกใช้ในการบ่งบอกตำแหน่งของพอร์ท I/O เพียง 8 เส้นเท่านั้น ซึ่งก็คือ Z80 สามารถจะติดต่อกับ I/O ขนาด 8 บิทได้ถึง 256 พอร์ท (2^8=256) โดยที่ระบบที่ใช้บ่งบอกตำแหน่งของพอร์ทนั้นจะต้องเป็นลักษณะการวางผัง I/O แบบมาตรฐาน (STANDARD I/O MAP) และแต่ละพอร์ท ต้องใช้แอดเดรสเพียงแอดเดรสเดียวในการบ่งบอกตำแหน่ง (1 แอดเดรส = 1 พอร์ท) เท่านั้น แต่ถ้าเป็นการเลือกพอร์ท แบบ LINEAR SELECT I/O คือสายแอดเดรส 1 เส้นใช้เลือก I/O 1 พอร์ทก็จะมีพอร์ท I/O เพียง 8 พอร์ท LINEAR SELECT I/O นี้เหมาะกับระบบที่มีอุปกรณ์ I/O จำนวน น้อย.

(หมายเหตุ ในบทนี้จะกล่าวถึง I/O MAPPED I/O เท่านั้น หากเข้าใจการ ติดต่อกับ I/O ในลักษณะนี้แล้วก็จะเป็นการง่ายที่จะเข้าใจลักษณะการติดต่ออื่น ๆ).

เนื่องจากการบ่งตำแหน่งของ I/O และของหน่วยความจำในระบบของ Z80 ใช้บัสแอดเดรสชุดเดียวกัน ไมโครโปรเซสเซอร์จึงต้องมีสัญญาณ $\overline{\text{IORQ}}$ (INPUT/OUTPUT REQUEST) แยกออกจาก $\overline{\text{MREQ}}$ (MEMORY REQUEST) ทราบมาแล้วว่าในระบบหน่วยจำนั้นไม่จำเป็นต้องใช้บัสแอดเดรสทั้ง 16 เส้น (A0-A15) ในระบบของ I/O ก็ไม่จำเป็นที่ต้องใช้สายแอดเดรสทั้ง 8 เส้นเช่นกัน ตัวอย่างเช่น ระบบต้องการใช้พอร์ท I/O เพียง 5 พอร์ท จะใช้สายแอดเดรส 3 เส้นก็พอ เพราะสายแอดเดรส 3เส้นนั้นสามารถบ่งบอกตำแหน่งที่แตกต่างกันได้ถึง 8 ตำแหน่ง (2^3) แต่พอร์ท I/O ที่จะพิจารณาในบทนี้ จะใช้บัสแอดเดรสทั้ง 8 เส้นในการถอดรหัสเลือกพอร์ท.

แต่ละพอร์ท I/O ในระบบไมโครโปรเซสเซอร์จะตอบสนองเพียงตำแหน่งเดียว

ของระบบบัสแอดเดรส (A0-A7) ตำแหน่งของพอร์ท I/O ที่ไมโครโปรเซสเซอร์ติดต่อด้วยนั้น จะถูกเรียกว่า "พอร์ทแอดเดรส (PORT ADDRESS)" สำหรับในบทนี้จะใช้ตำแหน่ง OFFH เป็นพอร์ทแอดเดรสของอุปกรณ์ I/O รูป 2.2.2 แสดงวงจรที่ใช้ในการถอดรหัสเลือกพอร์ทแอดเดรส OFFH นี้.



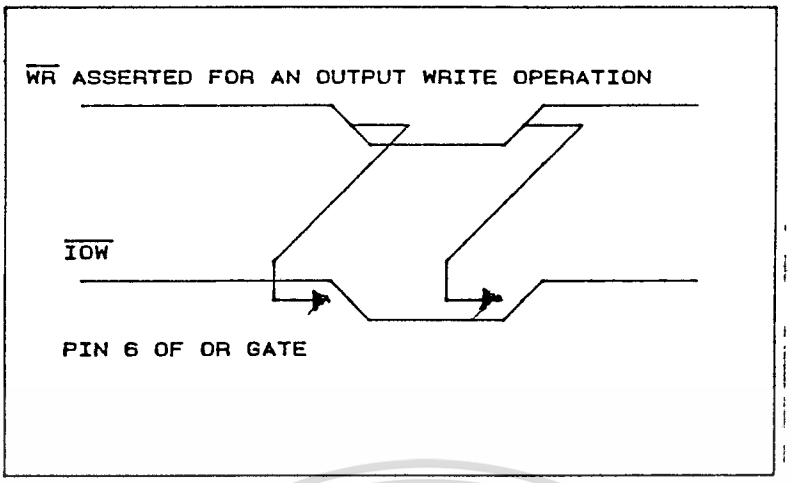
รูปที่ 2.2.2 : แสดงวงจรถอดรหัสเพื่อสร้างสัญญาณ PORT SELECT โดยมีตำแหน่ง PORT อยู่ที่ OFFH.

จากไดอะแกรมในรูป 2.2.2 นี้ จะเห็นว่าเอาท์พุทขา 8 ของ IC 74LS30 จะมีลอจิกเป็น "0" เมื่อขาอินพุทของ GATE ทั้งหมดเป็น "1" สังเกตุว่าเอาท์พุทของขา 8 ของ 74LS30 (PORT SELECT LINE) จะให้แอคทีฟที่ลอจิก "0" เมื่อบัสแอดเดรสมีลอจิกเท่ากับรหัสของพอร์ทที่จะเลือกเท่านั้น สำหรับตัวอย่างนี้จะให้ PORT SELECT LINES แอคทีฟที่ลอจิก "0".

อย่างไรก็ตาม PORT SELECT LINE สามารถที่จะแอคทีฟได้ ในขณะที่ Z80 ไม่ต้องการติดต่อกับ I/O ทั้งนี้ เนื่องจากระบบหน่วยความจำใช้บัสแอดเดรสชุดเดียวกับระบบของ I/O นั่นเอง ยกตัวอย่างเช่น ในกรณีที่ไมโครโปรเซสเซอร์ต้องการที่จะอ่านข้อมูลจากหน่วยความจำที่ตำแหน่ง XXFFH, PORT SELECT LINE ก็จะแอคทีฟด้วยเพราะบัสแอดเดรส A0-A7 จะมีค่า OFFH ซึ่งตรงกับรหัสของ PORT SELECT LINE ดังนั้นจึงจำเป็นต้องสร้างสัญญาณควบคุมขึ้นมา เพื่อเป็นการบ่งบอกว่าขณะนั้นไมโครโปรเซสเซอร์ต้องการติดต่อกับหน่วยความจำหรือ I/O.

2.2.3 การสร้างสัญญาณควบคุม $\overline{IOW}$ และ $\overline{IOR}$.

ในขณะที่ Z80 กระทำขบวนการติดต่อกับ I/O ขา $\overline{IORQ}$ ของ Z80 จะให้ลอจิก



รูปที่ 2.2.4B : แสดงไดอะแกรมเวลาของการเกิดสัญญาณ $\overline{IOW}$.

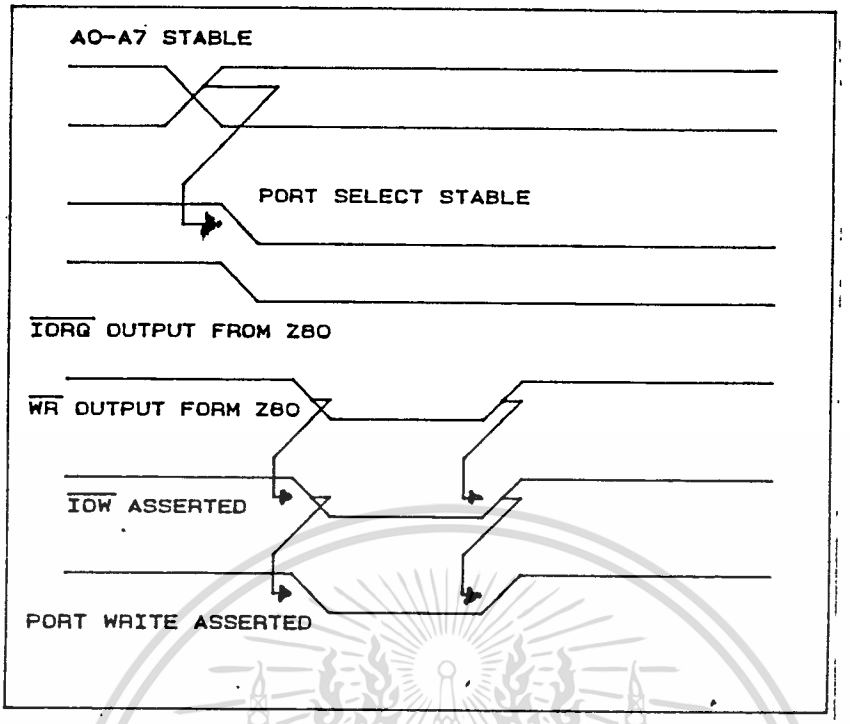
สัญญาณการเขียนพอร์ท (PORT WRITE) ของระบบ Z80 จะถูกใช้เป็นสัญญาณ WRITE ENABLE STROBE ของพอร์ทเอาท์พุทที่ถูกเลือก สัญญาณการเขียนพอร์ทนี้จะเกิดขึ้นได้ก็ต่อเมื่อ PORT SELECT LINE และ $\overline{IOW}$ แอคทีฟพร้อมกัน (มีลอจิกเป็น "0" พร้อมกัน) สัญญาณการเขียนพอร์ทจะเป็นสัญญาณที่กระตุ้นให้อุปกรณ์เอาท์พุทรับข้อมูลจากไมโครโปรเซสเซอร์ไปแลทช์(LATCH)เอาไว้ หรือเขียนข้อมูลลงบนพอร์ทเอาท์พุทที่ถูกเลือก

ไดอะแกรมเวลาในรูป 2.2.5 แสดงลำดับเวลาของสัญญาณที่ปรากฏขึ้นในขณะข้อมูลถูกส่งออกไปยังพอร์ทเอาท์พุท ในรูปนี้จะเห็นว่าทุก ๆ สัญญาณยกเว้นสัญญาณ $\overline{WR}$ จะเป็นสัญญาณซึ่งจะคงอยู่จนกว่าขบวนการติดต่อกับ I/O จะสิ้นสุดลง ดังนั้นจะเขียนวงจรได้ดังรูป 2.2.6 วงจรนี้แสดงวิธีหนึ่งที่ใช้ในการสร้างสัญญาณการเขียนพอร์ทเท่านั้น

มาพิจารณาไดอะแกรมของวงจรนี้ว่าทำงานได้อย่างไรการเข้าใจการทำงานของวงจรนี้จะช่วยให้เข้าใจสัญญาณต่าง ๆ ที่ Z80 ใช้ได้ดีขึ้น (สามารถที่จะสร้างสัญญาณ PORT STROBE โดยวิธีอื่นๆ ก็ได้)

จุดประสงค์ของวงจรในรูป 2.1.6 คือต้องการสร้าง TROBE ซึ่งแอคทีฟที่ลอจิก "0" เมื่อ CPU ต้องการส่งข้อมูลไปยังพอร์ทเอาท์พุทที่ถูกกำหนด ในรูปนี้ระบบบัสแอดเดรส A0-A7 เป็นอินพุททั้ง 8 เส้นของ NAND GATE ซึ่งเป็นวงจรลักษณะเดียวกับวงจรในรูป 2.2.2

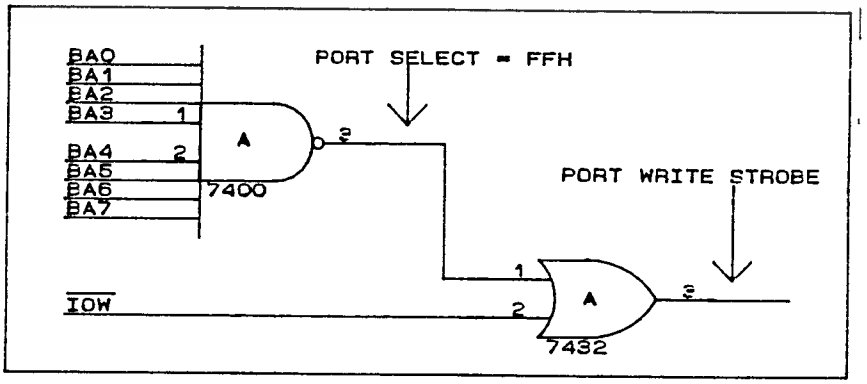
เมื่อบัสแอดเดรสทั้ง 8 เส้น (A0-A7) มีลอจิกเป็น "1" (รหัสเลือกพอร์ทคือ OFFH) เอาท์พุทของ NAND GATE จะมีลอจิกเป็น "0" PORT SELECT LINE นี้จะต่อเข้ากับอินพุทหนึ่งของ 74LS32 OR GATE สำหรับอีกอินพุทหนึ่งของ OR GATE นั้นมาจากสัญญาณ



รูปที่ 2.2.5 : แสดงไดอะแกรมเวลาของการเกิดสัญญาณ PORT WRITE. IOW STROBE (แสดงในรูป 2.2.3) เมื่อ IOW STROBE มีลอจิกเป็น "0" จะบ่งบอกให้รู้ว่า CPU กำลังกระทำการส่งข้อมูลไปยังพอร์ทเอาท์พุท.

ดังนั้นถ้าสัญญาณ IOW และ PORT SELECT LINE มีลอจิกเป็น "0" พร้อมกัน CPU ก็จะส่งข้อมูลให้พอร์ทเอาท์พุทที่ถูกเลือก การใช้สัญญาณ IOW และ PORT SELECT LINE มาผ่าน OR GATE ก็เพียงพอที่จะสร้างสัญญาณ STROBE กระตุ้นพอร์ทเอาท์พุทที่ถูกเลือกให้ทำการรับเอาข้อมูลไป สัญญาณนี้จะเรียกว่า "PORT WRITE STROBE".

วิธีการที่กล่าวถึงนี้เป็นวิธีการหนึ่งในการสร้างสัญญาณ STROBE ให้พอร์ทเอาท์พุทรับเอารับข้อมูลไปเท่านั้น ในระบบไมโครโปรเซสเซอร์ทั่วไปก็มักจะใช้วิธีดังกล่าว.

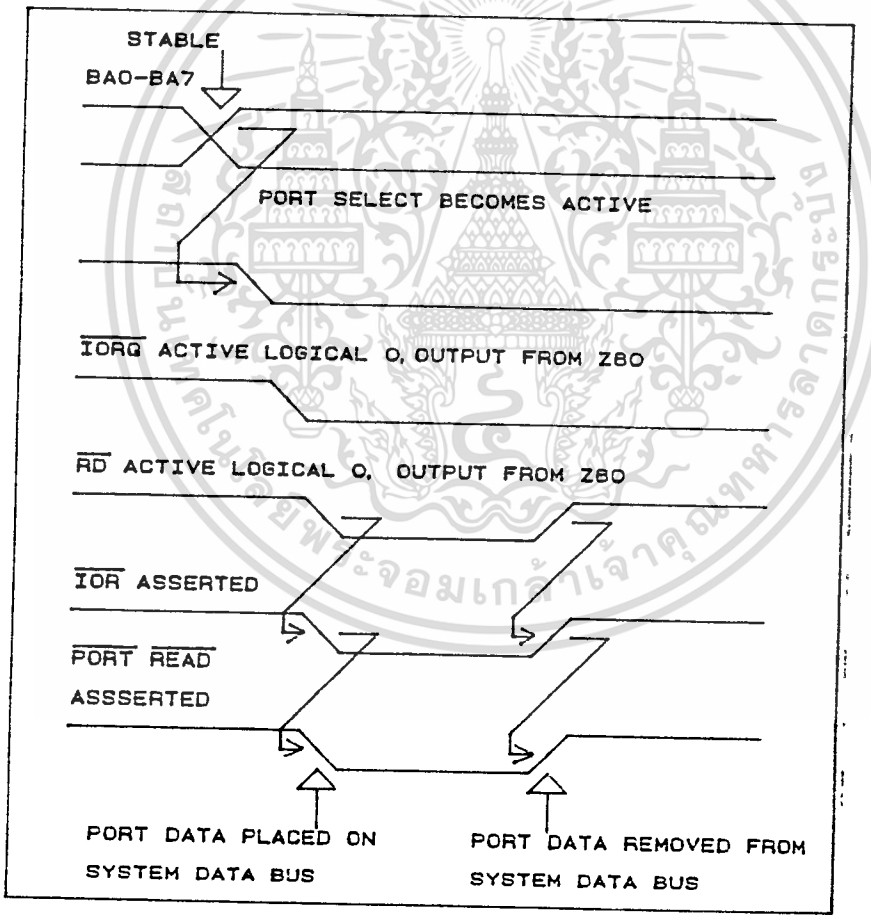


รูปที่ 2.2.6 : แสดงวงจรการสร้างสัญญาณ PORT WRITE.

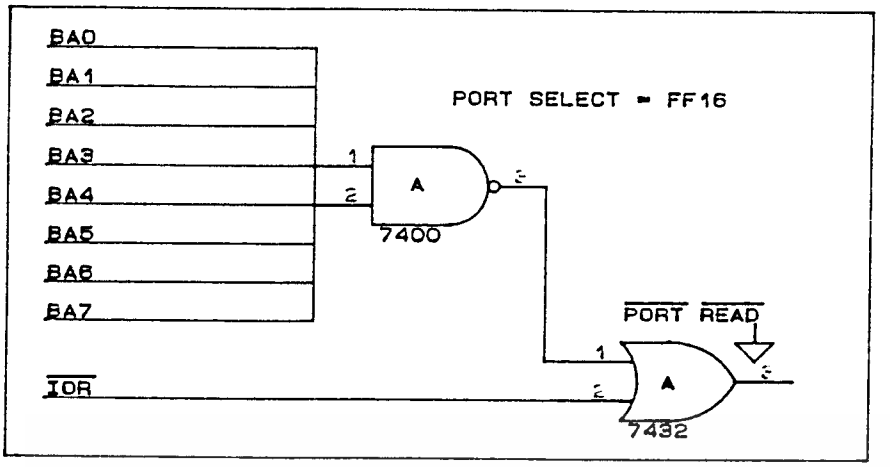
2.2.4 การสร้างสัญญาณการอ่านพอร์ท.

ที่จะกล่าวต่อไปนี้จะศึกษาขบวนการอ่านข้อมูลจากพอร์ทอินพุทของ Z80 ไดอะแกรมเวลาในรูป 2.2.7 แสดงลำดับของสัญญาณที่ปรากฏขึ้นระหว่างขบวนการอ่านข้อมูลจากพอร์ทอินพุท ในไดอะแกรมนี้สัญญาณ $\overline{IOR}$ จะมีลักษณะคล้ายกับสัญญาณ $\overline{IOW}$ ในขณะที่ทำการส่งข้อมูลไปยังพอร์ทเอาท์พุท.

จะมาพิจารณาผลของสัญญาณ $\overline{RD}$ ซึ่งแสดงในรูป 2.2.7 สัญญาณ $\overline{RD}$ นี้จะนำไปสร้างสัญญาณ $\overline{IOR}$ ให้แก่ฮาร์ดแวร์ส่วนพอร์ทอินพุท สัญญาณนี้จะทำให้เกิดการเคลื่อนย้ายข้อมูลจากพอร์ทอินพุทเข้าสู่ Z80 เมื่อ $\overline{RD}$ เปลี่ยนลอจิกมาเป็น "0" ข้อมูลจากพอร์ทอินพุทจะปรากฏบนระบบบัสข้อมูลและจะถูก STROBE เข้าไปในรีจิสเตอร์ภายใน (INTERNAL REGISTER) ของ Z80 สัญญาณ $\overline{RD}$ ก็จะกลับมาเป็นลอจิก "1" ใหม่ ขบวนการเคลื่อนย้ายข้อมูลก็จะสิ้นสุดลง



รูปที่ 2.2.7 : แสดงไดอะแกรมเวลาของสัญญาณที่เกิดขึ้นในวงจรรูป 2.2.6.



รูปที่ 2.2.8 : แสดงวงจงการสร้างสัญญาณ PORT READ.

รูป 2.2.8 แสดงวิธีหนึ่งในหลาย ๆ วิธีที่จะสร้างสัญญาณการอ่านพอร์ท ซึ่งมาจากเงื่อนไขทางลอจิกในรูป 2.2.7 จะเห็นว่ารูป 2.2.8 นั้นมีวงจรคล้ายกับวงจรในรูป 2.2.6 มากแตกต่างกันที่สัญญาณ $\overline{IOW}$ และ $\overline{IOR}$ เท่านั้น PORT READ STROBE จะแอคทีฟ (มีลอจิกเป็น "0") เมื่อ Z80 ทำการรับข้อมูลจากพอร์ทที่เลือก ในที่นี้คือ พอร์ท OFFH ในรูป 2.2.9 แสดงวงจรที่สมบูรณ์ในการสร้างพอร์ท I/O ขนาด 8 บิททั่ว ๆ ไป

ในหัวข้อต่อไปจะกล่าวถึงรายละเอียดของขบวนการเอาท์พุทและอินพุทจากพอร์ท I/O

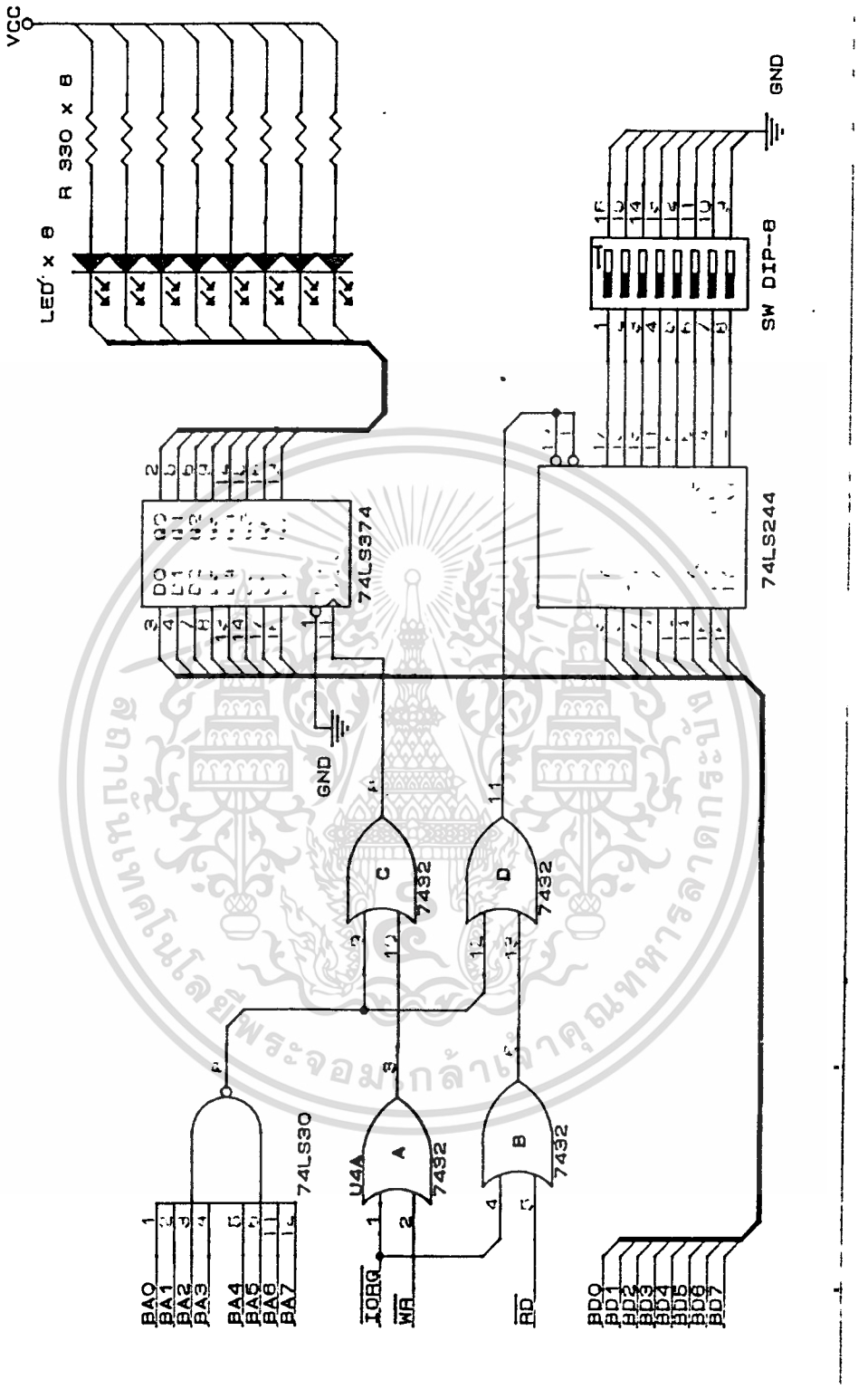
2.2.5 เหตุการณ์ที่เกิดขึ้นขณะกระบวนการส่งข้อมูลไปยังพอร์ทเอาท์พุท.

1. บัสแอดเดรส A0-A7 จะส่งตำแหน่งไปเอาท์พุทภายใต้การควบคุมของ Z80 ซึ่งขณะเดียวกันบัสแอดเดรส A0-A7 จะถูกถอดรหัสโดยฮาร์ดแวร์ส่วน PORT SELECT LINE ในรูป 2.2.9 เอาท์พุทขา 8 ของ IC1(74LS30) จะมีลอจิกเป็น "0" เมื่อ A0-A7 มีค่าตรงกับพอร์ทแอดเดรสที่กำหนด(ในที่นี้คือ OFFH).

2. Z80 จะส่งข้อมูลที่ต้องการส่งออกเข้าสู่บัสข้อมูล D0-D7 ในรูป 2.2.9 ข้อมูลจะรออยู่ที่ขาอินพุท D0-D7 ของ IC4 (74LS374 ; OCTAL LATCH).

3. ขา $\overline{IORQ}$ ของ Z80 จะถูกเซ็ทให้มีลอจิกเป็น "0" ซึ่งจะเป็นการบ่งบอกว่า Z 80 ต้องการที่จะติดต่อกับ I/O สัญญาณนี้ยังคงค้างไว้จนกว่าจะสิ้นสุดขบวนการ

4. ขา $\overline{WR}$ ของ Z 80 จะถูกเซ็ทให้มีลอจิกเป็น "0"ซึ่งจะทำให้สัญญาณ IOW มีลอจิกเป็น"0" ก็จะทำให้ PORT WRITE STROBE เปลี่ยนลอจิกเป็น "0" ขา $\overline{WR}$ ของ Z80 จะกลับมามีลอจิกเป็น"1" อีกครั้งหนึ่ง ขณะนี้ข้อมูลบนบัสข้อมูล D0-D7 จะถูกเขียนลงใน 74LS374 อันเป็นการสิ้นสุดการส่งข้อมูลไปยังพอร์ทเอาท์พุท



รูป 2.2.9 แสดงวงจรสมบูรณ์ของ พอร์ทอินพุท/เอาท์พุทขนาด 8 บิทซึ่งมีตำแหน่ง I/O อยู่ที่ OFFH

ในรูป 2.2.9 เป็นการนำข้อมูลของพอร์ทเอาท์พุทมาเปิด-ปิด LED บิทที่มีลอจิกเป็น "0" LED ตัวนั้นจะติด บิทที่มีลอจิกเป็น "1" LED จะดับ

2.2.6 ขบวนการอ่านข้อมูลจากพอร์ทอินพุท

ต่อไปจะพิจารณาขบวนการอ่านข้อมูลของ Z80 จากพอร์ทอินพุท ดังแสดงในรูป 2.2.9 ขบวนการส่วนใหญ่จะมีลักษณะคล้ายๆ กับการส่งข้อมูลไปยังพอร์ทเอาท์พุท ยกเว้นสัญญาณ $\overline{RD}$ ซึ่งจะถูกใช้แทนสัญญาณ $\overline{WR}$

1. สัญญาณ A0-A7 จะปรากฏขึ้นบนระบบบัสแอดเดรส IC1 จะทำการถอดรหัส ขา 8 เอาท์พุทของ IC1 จะมีลอจิกเป็น "0" ข้อมูลต่าง ๆ ของพอร์ทอินพุทจะยังไม่ปรากฏบนบัสข้อมูล จะรอจนกว่าสัญญาณ $\overline{IORQ}$ และสัญญาณ $\overline{RD}$ เกิดขึ้นพร้อมกันก่อน

2. $\overline{IORQ}$ จะถูกเซ็ทให้มีลอจิกเป็น "0" โดย Z80 การกระทำนี้เป็นการบ่งบอกแก่ระบบว่า Z80 ต้องการติดต่อกับ I/O ไม่ใช่หน่วยความจำ

3. สัญญาณ $\overline{RD}$ จะถูกเซ็ทให้มีลอจิกเป็น "0" โดย Z80 ซึ่งจะทำให้เกิดสัญญาณ $\overline{IOR}$ ขึ้นในระบบ

4. เมื่อสัญญาณ $\overline{IOR}$ ปรากฏขึ้นไมโครโปรเซสเซอร์จะทำการรับข้อมูลจากพอร์ทอินพุท เพราะว่าขา $\overline{RD}$ จาก Z80 มีลอจิกเป็น "0" ขา ENABLE (ขา 1.19) ของ IC5 (74LS244) จะถูกเซ็ทให้มีลอจิกเป็น "0" ลิกจิก "0" บนขา ENABLE ของ 74LS244จะทำให้ขาข้อมูลออก (DATA OUTPUT) มีข้อมูลส่งออกไปให้ระบบบัสข้อมูล เมื่อเหตุการณ์ทั้งหมดนี้เกิดขึ้นข้อมูลที่ถูกเซ็ทโดยสวิทช์หรือที่สัญญาณที่ส่งให้กับ 74LS244 ซึ่งเป็นบัฟเฟอร์ก็จะส่งข้อมูลให้กับบัสข้อมูลเข้าไปในรีจิสเตอร์ภายใน ของ Z80

5. ท้ายสุดขา $\overline{RD}$ จะถูกเซ็ทให้มีลอจิกเป็น "1" อีกครั้งหนึ่ง ภายใต้การควบคุมของ Z80 เมื่อ $\overline{RD}$ มีลอจิกเป็น "1" ข้อมูลจากพอร์ทอินพุทก็จะถูกเคลื่อนย้ายจากระบบบัสข้อมูลเข้าสู่ CPU ขบวนการอ่านข้อมูลจากพอร์ทอินพุทก็จะสิ้นสุดลง

2.2.7 สรุปขบวนการส่งข้อมูลออกไปยังพอร์ทเอาท์พุท.

1. A0-A7 จะถูกเซ็ทให้ตรงกับตำแหน่งของพอร์ทที่จะติดต่อด้วย
2. D0-D7 ถูกเซ็ทให้ตรงกับข้อมูลที่ต้องการส่งออกไป
3. $\overline{IORQ}$ จะมีลอจิกเป็น "0"
4. $\overline{WR}$ มีลอจิกเป็น "0"
5. $\overline{WR}$ จะถูกเซ็ทให้กลับมามีลอจิกเป็น "1"

2.2.8 สรุปขบวนการอ่านข้อมูลจากพอร์ทอินพุท

1. A0-A7 ถูกเซ็ทให้ตรงกับตำแหน่งของพอร์ทที่จะอ่านข้อมูล
2. $\overline{IORQ}$ ถูกเซ็ทให้มีลอจิกเป็น "0" ภายใต้การควบคุมของ Z80

3. $\overline{RD}$ ถูกเซ็ทให้มีลอจิกเป็น "0"

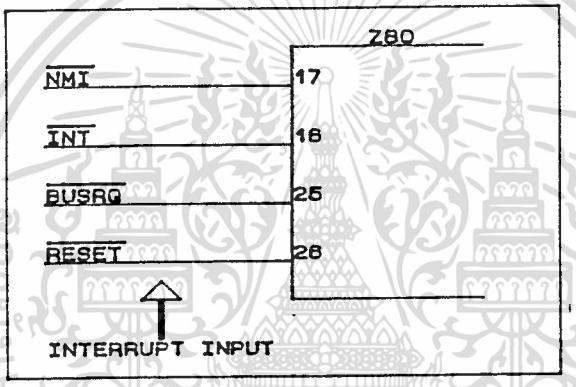
4. $\overline{RD}$ ถูกเซ็ทให้กลับมามีลอจิกเป็น "1"

2.3 Z80 กับการอินเทอร์รัพท์

การใช้งานไมโครโปรเซสเซอร์นั้น สิ่งหนึ่งที่ผู้ออกแบบระบบมักจะต้องคำนึงถึงอยู่เสมอก็คือ เรื่องของการอินเทอร์รัพท์ ดังนั้นในการที่จะนำเอาไมโครโปรเซสเซอร์ไปใช้งานได้อย่างมีประสิทธิภาพนั้น จึงจำเป็นต้องทำความเข้าใจเกี่ยวกับลักษณะ และ วิธีการอินเทอร์รัพท์เสียก่อน ในบทนี้จะกล่าวถึงลักษณะและวิธีการอินเทอร์รัพท์ในโหมดต่าง ๆ ของ Z80.

2.3.1 การอินเทอร์รัพท์ของ Z80

จุดประสงค์ของการทำขบวนการอินเทอร์รัพท์ก็คือ การทำให้ CPU พักจากการทำงานในโปรแกรมหลักไว้ชั่วคราว แล้วกระโดดไปทำงานในส่วนของโปรแกรมที่ตอบสนองต่อการขออินเทอร์รัพท์ (INTERRUPT SERVICE ROUTINE) นั้นโดยขึ้นกับชนิดและโหมดของการอินเทอร์รัพท์



รูปที่ 2.3.1สัญญาณที่ใช้เกี่ยวกับการอินเทอร์รัพท์

Z80 CPU ได้จัดการอินเทอร์รัพท์ไว้ใน 2 ลักษณะ คือการอินเทอร์รัพท์แบบไม่มีเงื่อนไข (NON-MASKABLE INTERRUPT) และแบบมีเงื่อนไข (MASKABLE INTERRUPT) ซึ่ง Z80 จะทำการตอบสนองต่อการขออินเทอร์รัพท์ทั้ง 2 แบบ ในลักษณะที่แตกต่างกัน

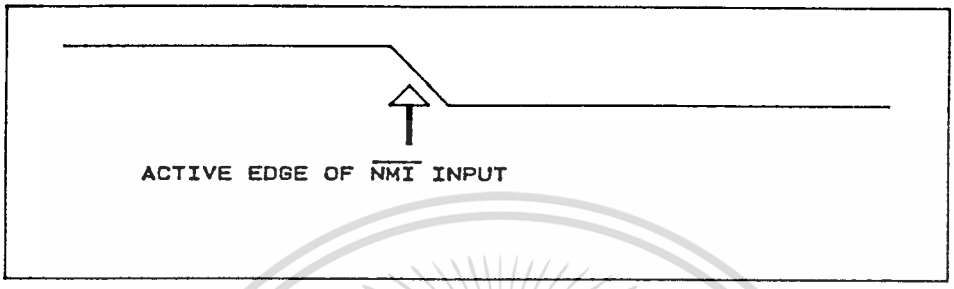
ยังมีวิธีการอื่นอีก 2 วิธี ซึ่งอาจจะกล่าวได้ว่าเป็นการอินเทอร์รัพท์ Z80 เช่นกัน คือ DMA (DIRECT MEMORY ACCESS) และ การรีเซ็ท.

สำหรับขบวนการ DMA นั้น เป็นขบวนการที่ค่อนข้างจะแตกต่างจากการอินเทอร์รัพท์และการรีเซ็ทอยู่มาก เนื่องจากขบวนการ DMA นี้แทนที่จะทำให้ Z80 กระโดดไปทำงานในโปรแกรมที่อยู่ในแอดเดรสอื่น กลับทำให้ Z 80 หยุดการทำงานไว้ชั่วขณะหนึ่ง

2.3.2 การอินเทอร์รัพท์แบบไม่มีเงื่อนไข (NON - MASKABLE INTERRUPT).

Z80 จะทำการตรวจสอบสัญญาณที่ขา $\overline{\text{NMI}}$ อยู่ตลอดเวลา ว่ามีการเปลี่ยนสถานะ

ลอจิกจาก "7" เป็น "0" (FALLING EDGE) หรือไม่ ถ้าพบว่ามีการเปลี่ยนแปลงดังกล่าวเกิดขึ้น FLIP-FLOP ภายในของ Z80 จะถูกเซ็ทไว้และ Z80 จะตรวจสอบค่าของ FLIP-FLOP ใน STATE สุดท้ายของการทำงานในทุก ๆ INSTRUCTION CYCLE ดังรูปที่ 2.3.2 ในกรณีที่ Z80 พบว่าค่าใน FLIP-FLOP นี้ถูกเซ็ทไว้ ก็จะเข้าสู่การทำงานในขบวนการที่ตอบสนองต่อการขออินเทอร์รัพท์ในแบบนี้



รูปที่ 2.3.2 : รูปสัญญาณที่ใช้ในการขออินเทอร์รัพท์แบบไม่มีเงื่อนไข

การขออินเทอร์รัพท์แบบนี้ Z80 จะทำการตอบสนองในทุกกรณี (ผู้ออกแบบไม่สามารถที่จะทำให้ Z80 DISABLE ต่อการขออินเทอร์รัพท์แบบนอน-มาสค์เคเบิลนี้ได้) ดังนั้นอุปกรณ์ภายนอกที่จะทำการขออินเทอร์รัพท์แบบนี้ได้นั้น จึงมักจะเป็นอุปกรณ์ที่มีความสำคัญมาก ๆ เท่านั้น

หลังจากที่ตรวจพบการขออินเทอร์รัพท์แบบนี้แล้ว Z80 จะนำเอาค่าแอดเดรสในรีจิสเตอร์ PC (PROGRAM COUNTER) ไปเก็บไว้ในสแตก (STACK) ซึ่งหลังจากที่ Z80 เสร็จจากการทำงานในโปรแกรมตอบสนองการอินเทอร์รัพท์ (INTERRUPT SERVICE ROUTINE) แล้ว จะนำเอาค่าที่เก็บไว้ในสแตกมาคืนให้กับ PC ตามเดิม เพื่อให้ Z80 สามารถที่จะกลับมาทำงานในโปรแกรมหลัก (MAIN PROGRAM) ต่อจากที่ทำค้างอยู่ก่อนที่จะถูกอินเทอร์รัพท์ได้อย่างถูกต้อง ซึ่งจะเห็นว่ามีลักษณะคล้ายคลึงกับคำสั่ง CALL ของ Z80 นั่นเอง.

เมื่อทำการเก็บค่าของ PC ไว้ในสแตกแล้วสถานะของ IFF1 ก็จะถูกนำไปเก็บไว้ใน IFF2 สำหรับชื่อของ IFF1 นั้นเรียกชื่อตามลักษณะการทำงานว่า INTERRUPT ENABLE FLIP-FLOP ซึ่งทำหน้าที่ของ IFF1 ก็คือ กำหนดว่าจะให้ Z80 ENABLE (สถานะของ IFF1 เป็นลอจิก "1") หรือ DISABBLE (สถานะของ IFF1 เป็นลอจิก "0") ต่อการขออินเทอร์รัพท์จากภายนอก ซึ่งจะใช้สำหรับกรณีของ มาสค์เคเบิลอินเตอร์รัพท์ (MASKABLE INTERRUPT) แต่สำหรับการอินเทอร์รัพท์แบบนอน-มาสค์เคเบิลนี้ สถานะของ IFF1 นี้จะไม่มีผลต่อการ ENABLE หรือ DISABLE ต่อการขออินเทอร์รัพท์แต่อย่างใด (เมื่อได้รับการขออิน

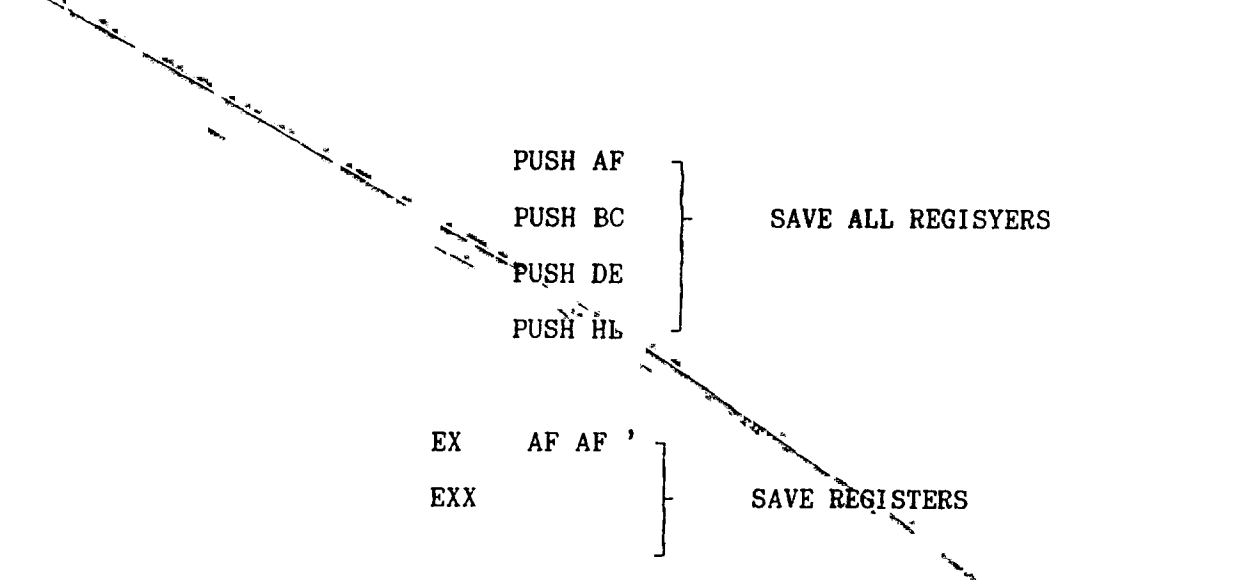
เทอร์รัพท์แบบนอน-มาสค์เคเบิล Z80 จะทำการตอบสนองในทุกกรณี) และก่อนเข้าสู่การทำงานในโปรแกรมหลัก (หลังจากสิ้นสุดการทำงานในโปรแกรมการตอบสนองการอินเทอร์รัพท์) นั้น IFF1 จะถูกปรับให้อยู่ในสถานะเดียวกับสถานะก่อนที่ Z80 จะถูกอินเทอร์รัพท์ (ซึ่งก็คือการนำเอาสถานะของ IFF2 มาคืนให้กับ IFF1 นั่นเอง)

หลังจากที่สถานะของ IFF1 ถูกนำไปเก็บไว้ใน IFF2 แล้ว IFF1 จะถูกปรับสถานะให้เป็นลอจิก "0" ชั่วคราว ในช่วงเวลาของ $\overline{NMI}$ เพื่อป้องกันการขออินเทอร์รัพท์ซ้อน (ซึ่งมีผลเฉพาะกับมาสค์เคเบิลอินเทอร์รัพท์เท่านั้น) เมื่อสถานะของ IFF1 ถูกปรับแล้ว Z80 จะกระโดดทำงานที่แอดเดรส 0066H ซึ่งที่แอดเดรสนี้จะเป็นแอดเดรสของโปรแกรมตอบสนองการอินเทอร์รัพท์ที่ผู้เขียนโปรแกรมเป็นผู้เขียนไว้ เพื่อกำหนดการทำงานของ Z80 เมื่อถูกอินเทอร์รัพท์แบบนอน-มาสค์เคเบิล

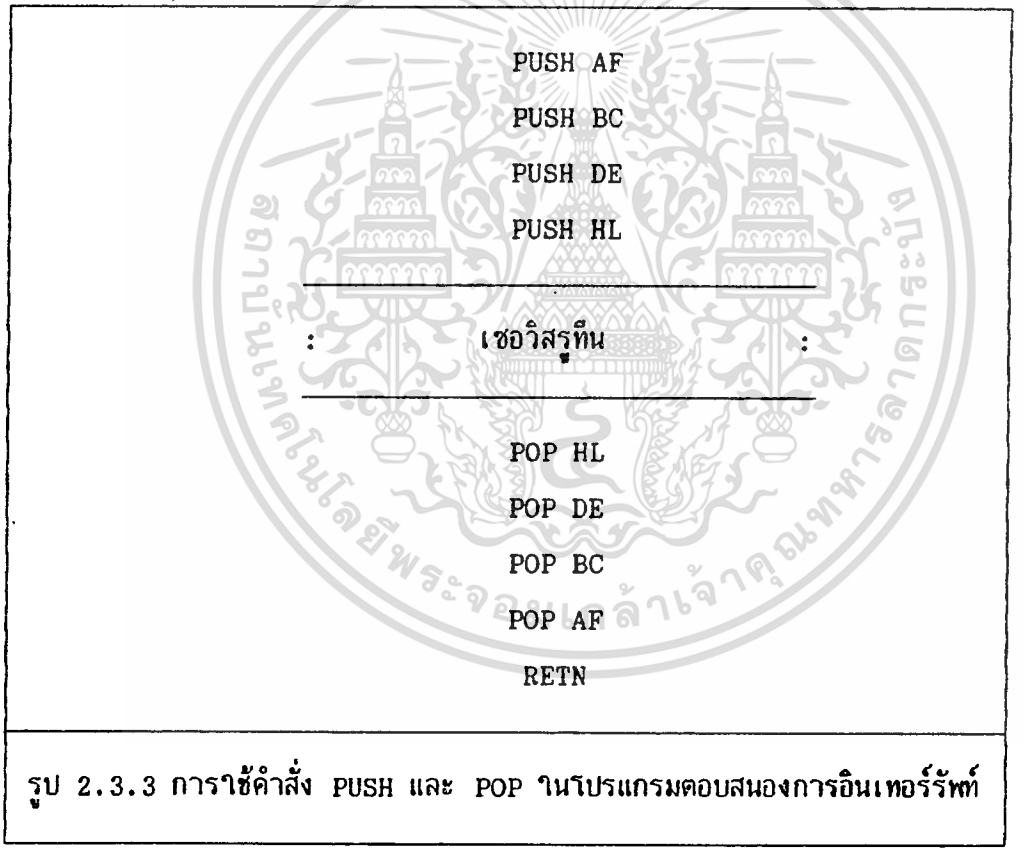
สามารถที่จะสรุปการทำงานของ Z80 ในขณะที่ถูกอินเทอร์รัพท์แบบนอนมาสค์เคเบิลได้ดังนี้คือ

1. นำเอาค่าแอดเดรสในรีจิสเตอร์ PC ไปเก็บไว้ในสแตก
2. นำเอาสถานะของ IFF1 ไปเก็บไว้ใน IFF2
3. ปรับสถานะของ IFF1 ให้เป็นลอจิก "0" เพื่อให้ Z80 ทำการ DISABLE ต่อการขออินเทอร์รัพท์จากภายนอก
4. Z80 จะกระโดดไปปฏิบัติตามโปรแกรมที่แอดเดรส 0066H

สิ่งที่สำคัญสิ่งหนึ่งที่จะต้องคำนึงถึงก็คือ ในขณะที่ Z80 ถูกอินเทอร์รัพท์นั้นค่าในรีจิสเตอร์ต่าง ๆ จะไม่ถูกเก็บเอาไว้ ดังนั้นในส่วนต้นของโปรแกรมตอบสนองการอินเทอร์รัพท์จึงควรที่จะเก็บค่ารีจิสเตอร์ต่าง ๆ ไว้ เนื่องจาก Z80 อาจจะถูกอินเทอร์รัพท์ที่ส่วนใดของโปรแกรมหลักก็ได้ ถ้าไม่ทำการเก็บค่ารีจิสเตอร์ต่าง ๆ ไว้ ค่าของรีจิสเตอร์ต่าง ๆ เหล่านั้นอาจจะถูกเปลี่ยนแปลงไป (ซึ่งเกิดจากการทำงานในโปรแกรมตอบสนองการอินเทอร์รัพท์) ทำให้เกิดความผิดพลาดขึ้นในระบบได้และอีกสิ่งหนึ่งที่สำคัญมากก็คือ ที่ส่วนท้ายของโปรแกรมตอบสนองการอินเทอร์รัพท์นั้น จะต้องทำการดึงเอาค่าที่เก็บไว้ออกมาคืนให้กับรีจิสเตอร์ของ Z80 ด้วย ในกรณีที่ใช้คำสั่ง PUSH ในการเก็บค่ารีจิสเตอร์แล้ว ไม่ใช้คำสั่ง POP คืนในตอนท้ายของโปรแกรมตอบสนองการอินเทอร์รัพท์ นอกจากจะทำให้ค่าในรีจิสเตอร์เกิดความผิดพลาดขึ้นแล้วยังจะทำให้ Z80 กลับมาทำงานที่แอดเดรสเดิมก่อนที่จะถูกอินเทอร์รัพท์ไม่ได้อีกด้วย เพราะค่าที่ถูกนำมาใส่คืนให้กับ PC นั้นไม่ใช่ค่าเดิม แต่จะเป็นค่าของรีจิสเตอร์แทน (ลักษณะของสแตกเป็นแบบเข้าก่อนออกทีหลังนั่นเอง)



(NOTE : THESE TWO INSTRUCTIONS WILL SAVE THE CONTENTS OF ALL CPU REGISTERS)



รูป 2.3.3 การใช้คำสั่ง PUSH และ POP ในโปรแกรมตอบสนองการอินเทอร์รัพท์

จากรูป 2.3.3 จะเห็นว่า ถ้าไม่ใช้คำสั่ง POP ก่อนคำสั่ง RETN (RETURN FROM NON-MASKABLE INTERRUPT) ค่าของ PC (หลังจากคำสั่งRETN) จะกลายเป็นค่าของคู่รีจิสเตอร์ HL แทนที่จะเป็นค่าเดิมของ PC ที่เก็บไว้ ซึ่งเป็นสาเหตุใหญ่ที่ทำให้เกิด

ความผิดพลาดขึ้นในระบบได้

สำหรับในส่วนของโปรแกรมตอบสนองการอินเทอร์รัพท์นั้น คำสั่งสุดท้ายของโปรแกรมจะเป็นคำสั่ง RETN <RETURN FROM NON-MASKABLE INTERRUPT> ซึ่งการทำงานของคำสั่งนี้จะเหมือนกับคำสั่ง RET และ RETN นั้น Z80 จะนำเอาข้อมูลในไบท์บนสุดของสแตกจำนวน 2 ไบท์ มาคืนให้กับ PC เพื่อใช้เป็นแอดเดรสที่ Z80 จะกลับไปทำโปรแกรมต่อจากที่ทำค้างไว้ก่อนที่จะถูกอินเทอร์รัพท์สิ่งที่แตกต่างระหว่างคำสั่ง RETN กับ RET ก็คือ ถ้าใช้คำสั่ง RETN แล้ว Z80 จะนำเอาสถานะของ IFF2 ส่งคืนให้กับ IFF1 (ซึ่งเป็นสถานะเดิมของ IFF1 ก่อนที่จะเข้าสู่การอินเทอร์รัพท์นั่นเอง) ความจริงแล้วสามารถที่จะใช้คำสั่ง RET แทนคำสั่ง RETN ก็ได้ เพียงแต่ว่าสถานะของ IFF2 จะไม่ถูกนำมาส่งคืนให้กับ IFF1 เท่านั้น

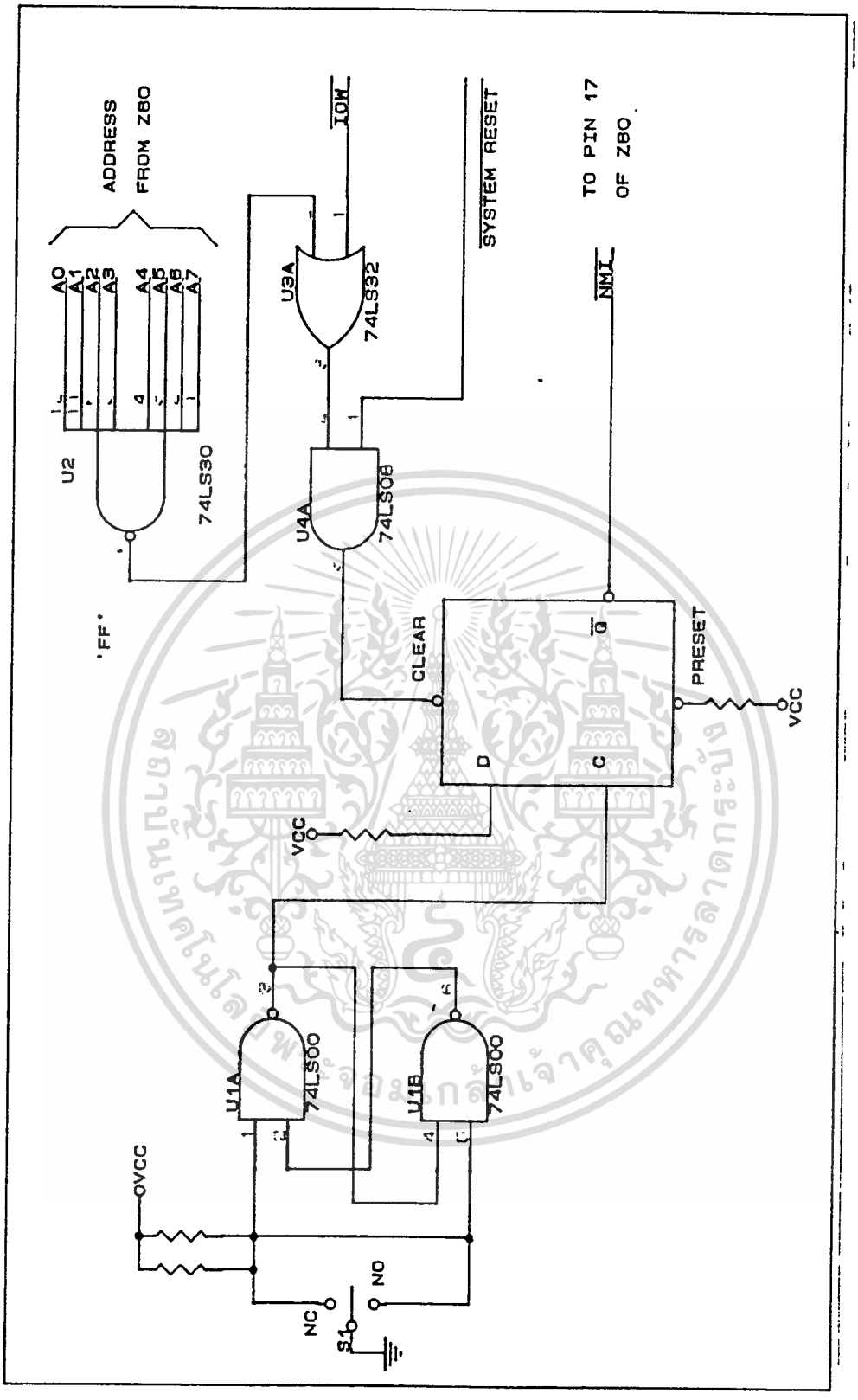
2.3.3 การสร้างสัญญาณ $\overline{\text{NMI}}$ (สัญญาณ NON-MASKABLE INTERRUPTS).

วิธีการในการสร้างสัญญาณ $\overline{\text{NMI}}$ นั้นมีอยู่หลายวิธี แต่ในทุกวิธีต่างก็ต้องยึดหลักที่ว่า เมื่อต้องการที่จะขออินเทอร์รัพท์ Z80 นั้น จะต้องทำให้ระดับสัญญาณที่ขา $\overline{\text{NMI}}$ ของ Z80 เปลี่ยนจากลอจิค "1" เป็น "0" (ขอบขาลง) และเมื่อเสร็จสิ้นจากขบวนการอินเตอร์รัพท์ไว้ ก็จะต้องทำการถอนสัญญาณ $\overline{\text{NMI}}$ โดยการทำให้ระดับสัญญาณที่ขา $\overline{\text{NMI}}$ ของ Z80 กลับเป็นลอจิก "1" อีกครั้ง.

วงจรข้างล่างนี้เป็นวิธีง่าย ๆวิธีหนึ่งในการที่จะสร้างสัญญาณ $\overline{\text{NMI}}$ ให้กับ Z80ได้โดยเพียงแต่กดสวิทช์เท่านั้น.

เมื่อสวิทช์ S1 ถูกกด เอาท์พุทที่ขา 3 ของ NAND GATE จะกลายเป็น "0" ทำให้ขาคล็อกของ D FLID - FLOP จะกลายเป็น "0" ตาม เมื่อปล่อยสวิทช์ขาคล็อกของ D FLID-FLOP จะกลายเป็น "1" ซึ่งจะมีผลให้เอาท์พุท Q กลายเป็น "0" (ขอบขาลง) ซึ่งจะทำให้ขา $\overline{\text{NMI}}$ ของ Z80 ได้รับสัญญาณอินเทอร์รัพท์.

หลังจากที่ Z80 ทำงานในส่วนที่เป็นโปรแกรมตอบสนองการอินเทอร์รัพท์จนเสร็จสิ้นแล้ว ก่อนที่ Z80 จะกลับเข้าสู่โปรแกรมหลัก จะต้องเขียนโปรแกรมให้ Z80 ส่งข้อมูลออกไปที่พอร์ท (PORT) OFFH ซึ่งจะทำให้ Q ของ D FLOD-FLOP กลับเป็น "1" อีกครั้ง ดังตัวอย่างในรูปที่ 2.3.5



รูป 2.3.4 ตัวอย่างวงจรสำหรับสร้างสัญญาณ $\overline{\text{NMI}}$ ให้กับ Z80

```
                ORG 0066H
                PUSH AF      : เก็บค่ารีจิสเตอร์ไว้ใน
                PUSH BC      : สแตก
                PUSH DE
                PUSH HL
-------------------------------------------------------
!   ส่วนที่เป็นโปรแกรมตอบสนองการอินเทอร์รัพท์          !
-------------------------------------------------------
                POP HL       : นำค่าที่เก็บไว้มาคืนให้
                POP DE       : กับรีจิสเตอร์
                POP BC
                POP AF
                OUT (OFFH) , A  : ทำการถอนสัญญาณ NMI
                RETN
```

รูปที่ 2.3.5 ตัวอย่างโปรแกรมสำเร็จรูปสำหรับวงจรในรูป 2.3.4

2.3.4 สรุปขั้นตอนการทำงาน

การอินเทอร์รัพท์แบบนอน-มาสค์เคเบิลที่อธิบายมาทั้งหมดนี้เชื่อว่าคงเพียงพอที่จะทำให้ผู้อ่านเข้าใจถึงวิธีการขออินเทอร์รัพท์แบบนี้ได้พอสมควร ในหัวข้อนี้จะขอนำลักษณะและวิธีการอินเทอร์รัพท์แบบนอน-มาสค์เคเบิลมากล่าวไว้เป็นข้อ ๆ ดังนี้

1. ทุกครั้งที่มีการขออินเทอร์รัพท์แบบนี้ Z80 จะทำการตอบสนองในทุกกรณีอย่างไม่มีเงื่อนไข

2. การอินเทอร์รัพท์แบบนี้จะเกิดขึ้นในกรณีที่ระดับสัญญาณที่ขา $\overline{\text{NMI}}$ เปลี่ยนจากลอจิก "1" เป็น "0" และ Z80 จะทำการตอบสนองต่อการขออินเทอร์รัพท์แบบนอน-มาสค์เคเบิลนี้ ใน STATE สุดท้ายของการทำงานใน INSTRUCTION CYCLE

3. Z80 จะนำเอาค่า PC ไปเก็บไว้ในสแตก

4. สถานะของ IFF1 จะถูกนำไปเก็บไว้ใน IFF2

5. สถานะของ IFF1 จะถูกนำเปลี่ยนให้เป็นลอจิก "0" (เพื่อทำการ

DISABLE ต่อการขออินเทอร์รัพท์แบบมาสค์เคเบิล)

6. Z80 จะกระโดดไปทำโปรแกรมที่แอดเดรส 0066H

2.3.5 มาสค์-เคเบิลอินเทอร์รัพท์ (MASKABLE INTERRUPT).

นอกจากการอินเทอร์รัพท์แบบนอน-มาสค์เคเบิลแล้ว Z80 ยังมีการอินเทอร์รัพท์อีกแบบหนึ่งคือ การอินเทอร์รัพท์แบบมาสค์เคเบิล (MASKABLE INTERURPT) การอินเทอร์รัพท์แบบหลังนี้สามารถที่จะสั่งให้ Z80 ENABLE หรือ DISABLE ต่อการของอินเทอร์รัพท์ได้ด้วยวิธีการทางซอฟท์แวร์ โดยใช้คำสั่ง EI (ENABLE INTERRUPT) หรือ DI (DISABLE INTERRUPT) ซึ่งการอินเทอร์รัพท์แบบแรกนั้นไม่สามารถที่จะทำให้ Z80 DISABLE ต่อการขออินเทอร์รัพท์ได้

สำหรับวิธีการที่จะขออินเทอร์รัพท์แบบนี้ สามารถที่จะทำได้โดยการทำให้ระดับลอจิกที่ขา $\overline{INT}$ เป็น "0" อยู่จนกระทั่ง Z80 สามารถที่จะตรวจจับสัญญาณได้ โดยที่ Z80 จะทำการตรวจสอบระดับสัญญาณที่ขา $\overline{INT}$ ใน STATE สุดท้ายของการทำงานในทุก ๆ INSTRUCTION CYCLE เมื่อ Z80 ตรวจพบสัญญาณ $\overline{INT}$ แล้วก็จะทำการตรวจสอบสถานะของ IFF1 ถ้าพบว่าสถานะของ IFF1 เป็น "0" Z80 ก็จะ DISABLE ต่อการขออินเทอร์รัพท์ แต่ในกรณีที่สถานะของIFF1 เป็น "1" Z80 ก็จะทำการตอบสนองต่อการขออินเทอร์รัพท์ และก่อนที่ Z80 จะกลับเข้าสู่การทำงานในโปรแกรมหลักนั้น จะต้องทำการถอนสัญญาณอินเทอร์รัพท์ที่ขา $\overline{INT}$ ของ Z80 ออกด้วย โดยการทำให้ระดับสัญญาณที่ขา $\overline{INT}$ ของ Z80 กลับเป็น "1" ใหม่

ในตอนนี้สามารถสรุปการอินเทอร์รัพท์แบบนี้ได้เป็น 2 หัวข้อ คือ

1. การขออินเทอร์รัพท์แบบมาสค์เคเบิล สามารถที่จะถูก DISABLE ได้โดยการใช้คำสั่ง DI ซึ่งจะทำให้สถานะของ IFF1 กลายเป็น "0"

2. สัญญาณ $\overline{INT}$ จะต้องมีลอจิกเป็น "0" อยู่จนกระทั่ง Z80 สามารถที่จะจับสัญญาณ $\overline{INT}$ ได้ และก่อนที่ Z80 จะกลับเข้าสู่การทำงานในโปรแกรมหลักนั้นจะต้องทำการถอนสัญญาณ $\overline{INT}$ ออกด้วย โดยการทำให้สัญญาณที่ขา $\overline{INT}$ กลับเป็น "1" อีกครั้ง

มีอุปกรณ์หรือซิพซัพพอร์ทอยู่หลายตัวที่ออกแบบไว้ใช้งานร่วมกับ Z80 โดยเฉพาะ เช่น Z80-PIO ซึ่งสามารถที่จะทำการถอนสัญญาณอินเทอร์รัพท์ออกได้เองในช่วงเวลาที่ Z80 ทำการถอนสัญญาณตอบรับการอินเทอร์รัพท์ (INTERRUPT ACKNOWLEDGE) และซิพซัพพอร์ทของ Z80 เหล่านี้จะทำการถอนสัญญาณที่ขา IEO (ทำให้ขานี้มีลอจิก "1") ของตัวซิพซัพพอร์ทออกเองในขณะที่ Z80 ทำการเฟ็ทช์คำสั่ง RETI สำหรับซิพซัพพอร์ทต่าง ๆ

การอินเทอร์รัพท์แบบมาสค์เคเบิลนี้ มีการทำงานในลักษณะต่าง ๆ กันถึง 3 โหมด (ถึงแม้จะมีขา $\overline{INT}$ เพียงขาเดียวก็ตาม) คือ โหมด 0 , โหมด 1 และ โหมด 2 ซึ่งจะ

ต้องทำการโปรแกรมเลือกโหมดก่อนที่ Z80 จะถูกอินเทอร์รัพท์โดยใช้คำสั่ง IM0 , IM1 หรือ IM2เพื่อเลือกการทำงานในโหมด 0,1 หรือ 2 ตามลำดับ

อย่างไรก็ตามเมื่อเริ่มจ่ายไฟหรือทำการรีเซ็ท Z80 โหมดของการอินเทอร์รัพท์จะถูกปรับให้อยู่ในโหมด 0 โดยอัตโนมัติ และ IFF1 จะถูกปรับให้มีสถานะเป็น "0" ซึ่งหมายความว่า Z80 จะปรับให้ IFF1 มีสถานะเป็น "1" เพื่อให้ Z80 พร้อมที่จะรับการขออินเทอร์รัพท์จากภายนอก

2.3.6 การอินเทอร์รัพท์ในโหมด 1

ดังที่ได้กล่าวมาแล้วว่าการอินเทอร์รัพท์แบบมาสค์เคเบิลนี้ สามารถที่จะเลือกให้ Z80 ทำงานในโหมดต่าง ๆ กันได้ถึง 3 โหมด คือ โหมด 0,1 หรือ 2 ได้โดยใช้คำสั่งทางซอฟแวร์เพื่อเลือกโหมดการทำงาน สำหรับในหัวข้อนี้จะศึกษาถึงการทำงานของ Z80 ในโหมด 1 ซึ่งสามารถที่จะเลือกโหมดการทำงานนี้ได้โดยการใช้คำสั่ง IM1 (INTERRUPT MODE 1).

หลังจากที่ได้ทำการอินเทอร์รัพท์ Z80 แล้ว และ Z80 ได้ถูกโปรแกรมให้ทำงานในโหมด 1,Z80 จะทำงานตามขั้นตอนต่อไปนี้คือ

1. สถานะของ IFF1 จะถูกปรับให้เป็นลอจิก "0" ซึ่งจะเป็นการทำให้ Z80 DISABLE ต่อการขออินเทอร์รัพท์ที่ขา $\overline{INT}$

2. สถานะของ IFF2 จะถูกปรับให้ลอจิก "0" เช่นเดียวกับ IFF1 (แต่ไม่ใช่การนำเอาสถานะของ IFF1 มาให้กับ IFF2)

3. ค่าของ PC จะถูกนำไปเก็บไว้บนสแตก ซึ่งจะถูกนำกลับมาใช้เป็นแอดเดรส (คือนำกลับมาคืนให้กับ PC นั่นเอง) ของชุดคำสั่งที่ Z80 จะทำหลังจากที่เสร็จจากการทำงานในโปรแกรมตอบสนองการอินเทอร์รัพท์แล้ว

4. Z80 จะกระโดดไปทำงานที่แอดเดรส 0038H

ซึ่งจะเห็นได้ว่าการทำงานในโหมดนี้มีลักษณะคล้ายกับการอินเทอร์รัพท์แบบนอนมาสค์เคเบิลมาก และการอินเทอร์รัพท์ในโหมดนี้นับได้ว่าง่ายที่สุดในการอินเทอร์รัพท์ทั้ง 3 โหมด เช่นเดียวกันที่ส่วนต้นของโปรแกรมตอบสนองการอินเทอร์รัพท์นั้น ควรที่จะเก็บค่ารีจิสเตอร์ต่าง ๆ ที่ต้องการใช้ในโปรแกรมตอบสนองการอินเทอร์รัพท์ที่นี้ไว้ด้วย โดยใช้คำสั่ง PUSH หรือ EX (EXCHANGE).

สิ่งหนึ่งที่จะต้องทำก็คือการถอนสัญญาณอินเทอร์รัพท์ที่ขา $\overline{INT}$ ของ Z80 ซึ่งจะต้องทำก่อนที่ Z80 จะกลับเข้าสู่การทำงานในโปรแกรมหลักหรือก่อนคำสั่ง EI ในโปรแกรมตอบสนองการอินเทอร์รัพท์ สำหรับซอฟท์แวร์ที่จะใช้ในส่วนนี้นั้นจะเปลี่ยนไปตามลักษณะของฮาร์ดแวร์ที่ได้ออกแบบไว้ในตอนนี้จะใช้วงจรในรูป 2.3.4 ซึ่งได้ใช้มาแล้วในการอินเทอร์รัพท์แบบ

นอนมาสค์เคเบิล โดยแทนที่จะเอาขา $\overline{Q}$ ของ D FLIP-FLOP ไปเข้าขา $\overline{NMI}$ ของ Z80 ก็เอาไปเข้าที่ขา $\overline{INT}$ แทน ซึ่งลักษณะการทำงานก็คล้ายกับที่ใช้กับ $\overline{NMI}$ เพียงแต่ $\overline{INT}$ นั้นจะ ENABLE ที่ระดับลอจิก "0" แต่ $\overline{NMI}$ จะ ENABLE ที่ขอบขาลงของสัญญาณเท่านั้น และสิ่งหนึ่งที่ไม่ได้กล่าวถึงในหัวข้อที่ 2.3.3 ก็คือ นอกจากจะทำการถอนสัญญาณอินเทอร์รัพท์ด้วยการใช้คำสั่งทางซอฟท์แวร์แล้วยังสามารถที่จะทำได้ด้วยการให้สัญญาณรีเซ็ทกับระบบอีกด้วย (เมื่อได้รับการรีเซ็ท Z80 จะถูกปรับให้อยู่ในโหมด 0 ทันที) สำหรับโปรแกรมส่วนที่เป็นโปรแกรมตอบสนองการอินเทอร์รัพท์นั้น ก็เพียงแต่เปลี่ยนคำสั่ง RETN เป็น RETI (RETURN FROM INTERRUPT) หรืออาจจะใช้คำสั่ง RET ธรรมดาก็ได้ ซึ่งทั้งคำสั่ง RETI และ RET ต่างก็ทำในสิ่งเดียวกันคือ นำเอาค่าของ PC ที่เก็บไว้ในตอนแรกมาคืนให้กับ PC เพื่อใช้เป็นแอดเดรสต่อไปที่ Z80 จะกลับไปทำงานในโปรแกรมหลักต่อจากที่ทำไว้ก่อนที่จะถูกอินเทอร์รัพท์ และคำสั่งทั้งสองนี้มีข้อแตกต่างกันอยู่คือ ชิพซัพพอร์ของ Z80 จะรับรู้ต่อการทำคำสั่ง RETI ของ Z80 ได้ แต่ไม่สามารถที่จะรับรู้การทำคำสั่ง RET ได้ นอกจากนี้ก่อนคำสั่ง RETI หรือ RET ในโปรแกรมตอบสนองการอินเทอร์รัพท์ ซึ่งควรจะเพิ่มคำสั่ง EI ไว้ด้วย เพื่อทำให้ Z80 กลับไปอยู่ในสภาวะที่พร้อมที่จะรับการอินเทอร์รัพท์ได้อีก อย่างไรก็ตามคำสั่ง EI นี้จะยังไม่มีผลจนกว่า Z80 จะทำคำสั่งที่ต่อจาก EI เสียก่อน ซึ่งจะเป็นผลดี คือ ทำให้แน่ใจได้ว่า Z80 จะไม่ถูกอินเทอร์รัพท์อีกก่อนที่จะกลับเข้าสู่โปรแกรมหลัก
อย่างไรก็ตามอาจจะใช้คำสั่ง EI ที่ส่วนใดของโปรแกรมตอบสนองการอินเทอร์รัพท์ก็ได้ซึ่งก็หมายความว่า ยอมให้มีการอินเทอร์รัพท์เกิดขึ้นได้ในระหว่างการทำงานในโปรแกรมตอบสนองการอินเทอร์รัพท์ได้ (ซึ่งอาจจะเป็นการอินเทอร์รัพท์ในโหมดอื่นก็ได้)

2.3.7 การอินเทอร์รัพท์ในโหมด 0.

ในหัวข้อนี้จะศึกษาถึงการอินเทอร์รัพท์ในโหมด 0 ของ Z80 ซึ่งดังที่ได้กล่าวไว้แล้วว่า เมื่อเริ่มจ่ายไฟให้หรือทำการรีเซ็ทบน Z80 โหมดของการอินเทอร์รัพท์จะถูกจัดให้อยู่ในโหมด 0 แต่อีกวิธีหนึ่งที่จะทำให้ Z80 เลือกการอินเทอร์รัพท์ในโหมด 0 ได้ก็คือการใช้คำสั่ง IM0 (บน 8080 ก็มีการอินเทอร์รัพท์ในรูปแบบเดียวกับโหมด 0 ของ Z80 ด้วย)

วิธีการขออินเทอร์รัพท์ในโหมดนี้ก็สามารถที่จะทำได้โดยใช้วิธีเดียวกันกับการอินเทอร์รัพท์ในโหมด 1 อย่างไรก็ตามในโหมด 0 นี้จะมีข้อที่ได้เปรียบกว่าในโหมด 1 อยู่ก็คือสามารถที่จะสั่งให้ Z80 กระโดยไปทำงานที่แอดเดรสใด ๆ หรือ ทำคำสั่งใด ๆ ก็ได้

```
                ORG   0038H
        PUSH AF
        PUSH HL     SAVE ALL REGISTERS OR ONLY THE ONES
        PUSH DE     USED IN THE ROUTINE
        PUSH BC

        (SERVICE ROUTINE)
            SECTION

        POP BC
        POP DE      RESTORE ALL REGISTERS
        POP HL
        POP AF

        OUT (OFFH)A                 CLEAR INTERRUPUT
            EI                      ENABLE INTERRUPUTS
        RET OR RETI
```

รูปที่ 2.3.6 ตัวอย่างโปรแกรมตอบสนองการอินเทอร์รัพท์สำหรับวงจรรูป 2.3.4

เมื่อได้รับการขออินเทอร์รัพท์จากอุปกรณ์ภายนอกและ Z80 ถูกสั่งให้ ENABLE การขออินเทอร์รัพท์แล้ว Z80 จะทำงานตามขั้นตอนต่อไปนี้ คือ

1. IFF1 และ IFF2 จะถูกปรับให้มีลอจิก "0" ซึ่งจะเป็นการทำให้ Z80 DISABLE ต่อการขออินเทอร์รัพท์ที่อาจเกิดขึ้นได้อีก

2. เอาท์พุทจากขา $\overline{M1}$ และ $\overline{IORQ}$ จะกลายเป็น "0" ทั้งคู่ ซึ่งจะเกิดขึ้นเฉพาะในขณะที่ Z80 ได้รับทราบการขออินเทอร์รัพท์แล้วเท่านั้น โดยทั่วไปแล้วสัญญาณจากขา $\overline{M1}$ และ $\overline{IORQ}$ นี้จะถูกนำมาผ่าน OR GATE (ซึ่งจะให้เอาท์พุทเป็น "0" เฉพาะในช่วงที่สัญญาณจากขา $\overline{M1}$ และ $\overline{IORQ}$ เป็น "0" พร้อมกันเท่านั้น) เพื่อนำไปใช้สำหรับการทำให้อุปกรณ์ภายนอกที่ขออินเทอร์รัพท์ทราบว่า Z80 ได้ทำการตอบรับการขออินเทอร์รัพท์ (INTERRUPT ACKNOWLADGE) แล้ว สำหรับชิพซัพพอร์ทของ Z80 นั้นจะมีขาสำหรับต่อกับขา

$\overline{M1}$ และ $\overline{IORQ}$ โดยตรง เพื่อตรวจสอบการตอบรับการขออินเทอร์รัพท์ของ Z80 อยู่แล้ว

3. เมื่ออุปกรณ์ภายนอกได้ตรวจสอบพบว่า Z80 ได้ทำการตอบรับการของอินเทอร์รัพท์ (INTERRUPT ACKNOWLEDGE) มาแล้ว ก็จะส่งข้อมูล 1 ไบท์ออกมาที่บัสข้อมูล (DATA BUS) ให้กับ Z80 เมื่อได้รับข้อมูลนี้แล้ว Z80 จะถือว่าข้อมูลนี้เป็นออฟโคดของคำสั่ง และทำงานตามออฟโคดนั้นทันที โดยทั่วไปแล้วข้อมูลไบท์นี้จะอยู่ในชุดคำสั่ง RST (RESTART) 0-7 ซึ่งจะมีผลทำให้ Z80 กระโดยไปทำงานในแอดเดรสต่าง ๆ ในกรณีที่ข้อมูลที่ส่งให้กับ Z80 เป็นออฟโคดของคำสั่งที่มากกว่า 1 ไบท์ เช่น ออฟโคดของคำสั่ง CALL ซึ่งต้องการถึง 3 ไบท์ ในกรณีนี้อุปกรณ์ภายนอกจะต้องทำการส่งข้อมูลที่เหลืออีก 2 ไบท์ให้กับ Z80 เอง (Z80 จะใช้อีก 2 MACHINE CYCLE ถัดมาในการอ่านข้อมูลที่เหลืออีก 2 ไบท์) แต่ในที่นี้ จะสมมติว่าข้อมูลที่อุปกรณ์ภายนอกส่งให้กับ Z80 นี้อยู่ในชุดคำสั่ง RST 0-7

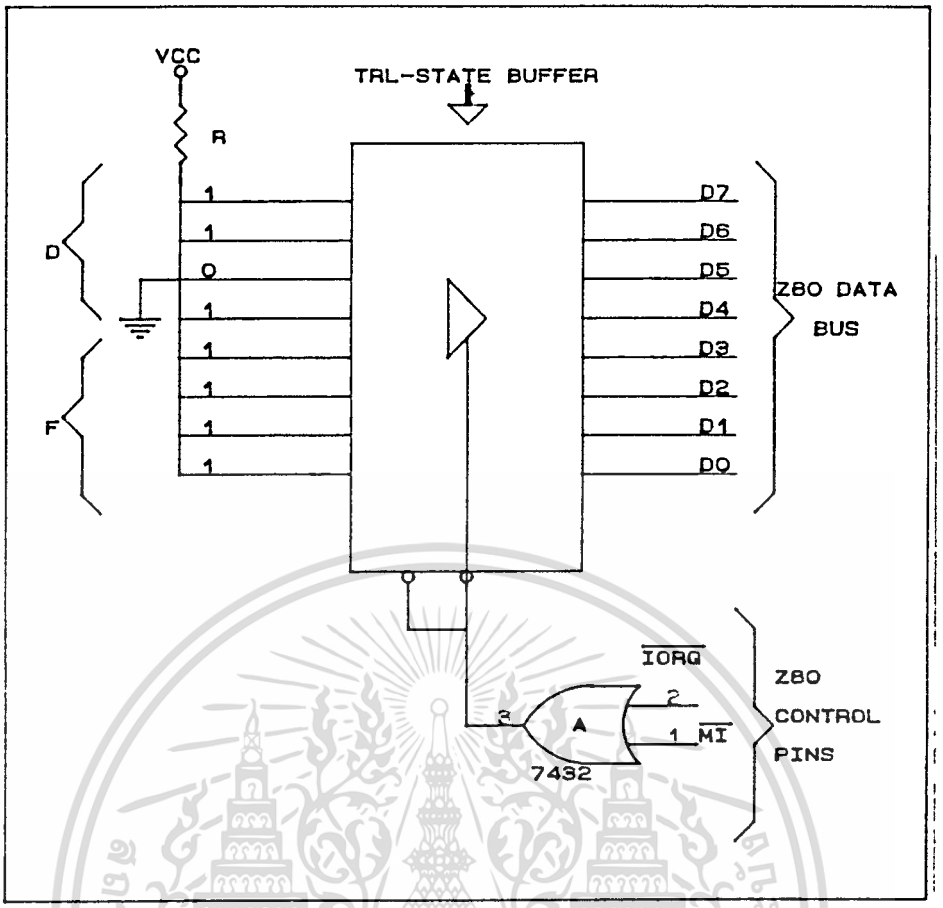
4. Z80 อ่านข้อมูลเข้ามาแล้วทำการถอดรหัสเช่นเดียวกับออฟโคดของคำสั่ง

5. ค่า PC จะถูกเก็บอยู่บนสแตก

6. Z80 จะกระโดดไปทำงานยังตำแหน่งแอดเดรสที่ถูกกำหนดโดยชุดคำสั่ง RST ตำแหน่งแอดเดรสที่ Z80 จะกระโดดไปทำเมื่อเทียบกับข้อมูลที่ได้รับเป็นดังนี้คือ

RST	ข้อมูลที่ส่งให้กับ Z80 (ฐาน 16)	ตำแหน่งแอดเดรส(ฐาน 16)
0	C7	0000
1	CF	0008
2	D7	0010
3	DF	0018
4	E7	0020
5	EF	0028
6	F7	0030
7	FF	0038

รูปที่ 2.3.7 นี้แสดงถึง วิธีการง่าย ๆ วิธีหนึ่งที่ใช้ในการส่งข้อมูลให้กับ Z80 ในรูปนี้นำเอาขา $\overline{M1}$ และ $\overline{IORQ}$ มาเข้าอินพุทของ OR GATE เมื่อ Z80 ตอบรับการอินเทอร์รัพท์เอาท์พุทที่ขา $\overline{M1}$ และ $\overline{IORQ}$ ของ Z80 จะกลายเป็น "0" ทั้งคู่ทำให้เอาท์พุทของ OR GATE กลายเป็น "0" ตาม ซึ่งจะเป็นการเปิด TRI-STATE GATE ซึ่งก็คือการทำให้ข้อมูลสามารถที่จะผ่านอินพุทไปสู่เอาท์พุทของ GATE ได้ทำให้ข้อมูล 0DFH ถูกส่งให้กับ Z80 เมื่อ Z80 ทำการถอดรหัสที่แอดเดรสแล้วได้เป็นออฟโคดของคำสั่ง RST 3 จะทำให้ Z80 กระโดยไปทำงานในโปรแกรมตอบสนองการอินเทอร์รัพท์ที่แอดเดรส 0018H สำหรับในตัวอย่างนี้สนใจแต่เฉพาะกรณีที่อุปกรณ์ภายนอกที่มีหลายชิ้น สำหรับคำสั่งสุดท้ายในโปรแกรมตอบสนองการอินเทอร์รัพท์นั้นใช้คำสั่ง RETI เช่นเดียวกับในโหมด 1 นั่นเอง



รูปที่ 2.3.7 วงจรง่าย ๆ สำหรับส่งข้อมูล RST ให้กับ Z80

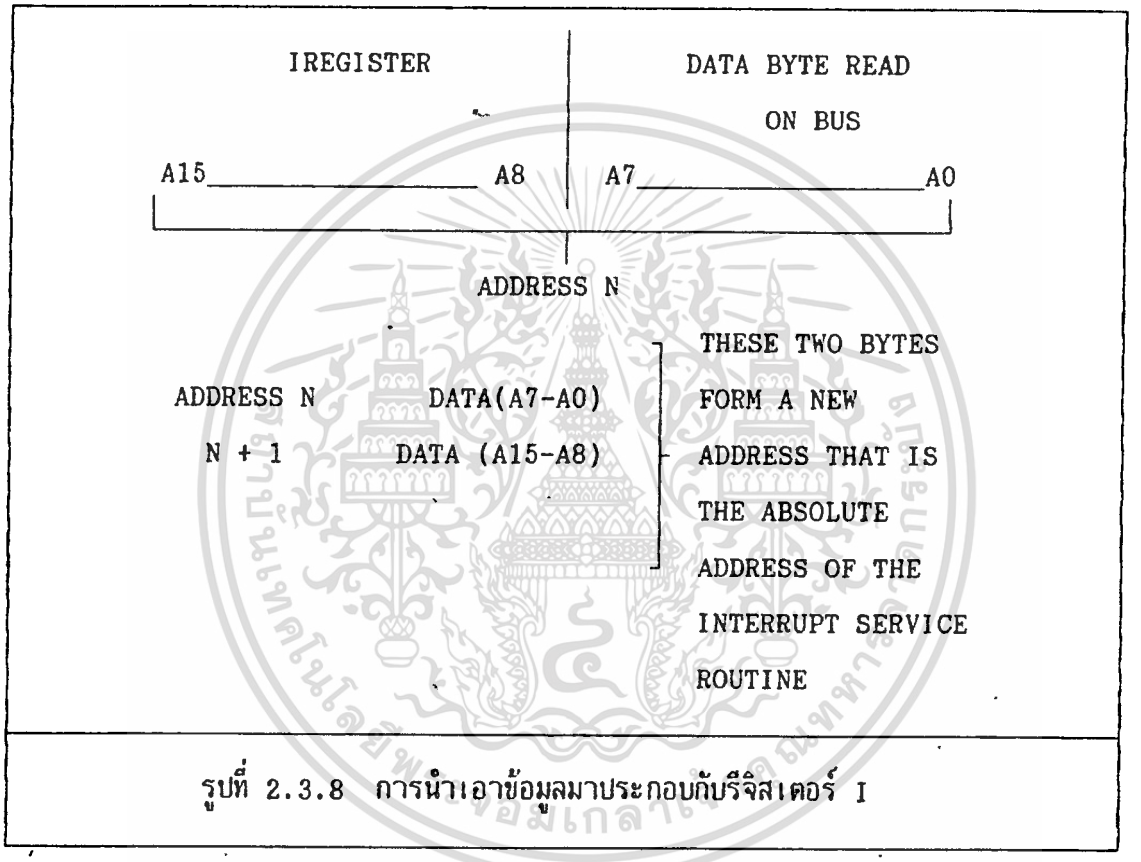
2.3.8 การอินเทอร์รัพท์ในโหมด 2

Z80 จะเลือกโหมด 2 เป็นโหมดในการอินเทอร์รัพท์เมื่อ Z80 ได้รับคำสั่ง IM 2 (INTERRUPT MODE 2) สำหรับวิธีการขออินเทอร์รัพท์ในโหมดนี้ก็ใช้วิธีเดียวกับการอินเทอร์รัพท์ในโหมด 1 และ 0 นั่นเอง.

การทำงาน ของ Z80 เมื่อถูกอินเทอร์รัพท์ในโหมด 2 นี้ก็คือ เมื่อ Z80 ตอบรับการขออินเทอร์รัพท์โดยการส่งสัญญาณ $\overline{M1}$ และ $\overline{IORQ}$ (ลอจิก "0") ให้กับอุปกรณ์ภายนอกที่ทำการขออินเทอร์รัพท์อุปกรณ์ภายนอกที่จะส่งข้อมูล 1 ไบท์ ผ่านบัสข้อมูลให้กับ Z80 ซึ่งในขั้นตอนนี้ก็จะมีลักษณะคล้ายกับการอินเทอร์รัพท์ในโหมด 0 นั่นเอง เพียงแต่ว่าในโหมด 2 นี้ เมื่อ Z80 ได้รับข้อมูลแล้วจะนำเอามาประกอบกับข้อมูลที่ถูกเก็บไว้ในรีจิสเตอร์ I (ดูรูปที่ 2.3.7 ประกอบ) โดยใช้ข้อมูลในรีจิสเตอร์ I เป็นแอดเดรส 8 บิทบน (A15-A18) และข้อมูลที่รับเข้ามาเป็นแอดเดรส 8 บิทล่าง (A7-A0) เมื่อได้แอดเดรสขนาด 16 บิทแล้ว (สมมติให้เป็นแอดเดรสที่ N) ก็จะใช้ข้อมูลในแอดเดรสที่ N นี้มาประกอบกับข้อมูลในแอดเดรสที่ N+1

เพื่อใช้เป็นแอดเดรสของ ROUTINE ที่ Z80 จะกระโดดไปทำ โดยจะใช้ข้อมูลในแอดเดรสที่ N เป็นแอดเดรส 8 บิทล่าง และใช้ข้อมูลในแอดเดรสที่ N+1 เป็นแอดเดรส 8 บิทบน

ตัวอย่างเช่น ในกรณีที่ข้อมูลที่อุปกรณ์ภายนอกส่งให้กับ Z80 เป็น 93H และกำหนดให้รีจิสเตอร์ I มีข้อมูล 10H อยู่ Z80 จะนำเอาข้อมูลในรีจิสเตอร์ I มาประกอบกับข้อมูลที่ได้รับเข้ามาซึ่งจะได้เป็นแอดเดรสค่า 1093H จากนั้น Z80 จะนำเอาข้อมูลในแอดเดรส 1093H กับ 1093+1H (1094H) มาประกอบกันเป็นค่าของแอดเดรสที่ Z80 จะกระโดดไปทำ เช่น



รูปที่ 2.3.8 การนำเอาข้อมูลมาประกอบกับรีจิสเตอร์ I

สมมติให้ แอดเดรส 1093H มีข้อมูลเป็น 40H
แอดเดรส 1094H มีข้อมูลเป็น 34H

ดังนั้นเมื่อนำข้อมูลในแอดเดรส 1093H และ 1094H มาประกอบกันจะได้ค่าแอดเดรสของโปรแกรมตอบสนองการอินเทอร์รัพท์เป็น 3440H นั่นเอง

บางครั้งในระบบอาจจะมีอุปกรณ์ภายนอกหลายตัวที่สามารถที่จะขออินเทอร์รัพท์ได้ และอุปกรณ์แต่ละตัวก็ต้องการการตอบสนองจาก Z80 ไม่เหมือนกัน ในกรณีนี้สามารถจะทำให้ Z80 กระโดดไปทำงานในแอดเดรสต่างๆ กันได้ โดยเพียงแต่ออกแบบให้อุปกรณ์ภายนอกส่งข้อมูลที่แตกต่างกัน (โดยให้สัมพันธ์กับแอดเดรสของโปรแกรมตอบสนองการอินเทอร์รัพท์ของอุ

ปกรณ์แต่ละชิ้น) ให้กับ Z80 ในช่วงที่ Z80 ตอบรับการขออินเทอร์รัพท์

สำหรับในกรณีนี้จะสมมติว่า มีอุปกรณ์ภายนอกที่สามารถที่จะขออินเทอร์รัพท์ได้อยู่ 5 ชิ้น สิ่งแรกที่ต้องการทำก็คือกำหนดว่าจะให้แอดเดรสของโปรแกรมตอบสนองการอินเทอร์รัพท์สำหรับอุปกรณ์แต่ละชิ้นนั้นอยู่ที่ใด ในกรณีนี้จะใช้ชื่อของอุปกรณ์ทั้ง 5 ชิ้นนี้ว่า DEV1-DEV5 และกำหนดให้แอดเดรสต่าง ๆ สำหรับอุปกรณ์ทั้ง 5 ดังนี้

อุปกรณ์ #	แอดเดรสในหน่วยความจำ (ฐาน 16)
DEV1	2754
DEV2	3015
DEV3	1774
DEV4	5367
DEV5	7741

สิ่งหนึ่งที่จะต้องคำนึงถึงในการส่งข้อมูลก็คือ ข้อมูลที่ส่งให้กับ Z80 นั้นจะต้องมีบิท D0 เป็น "0" เท่านั้นและข้อมูลไบท์นี้เรียกว่า "อินเทอร์รัพท์เวคเตอร์" (INTERRUPT VECTOR) สำหรับขั้นต่อไปก็คือจะต้องกำหนดว่าจะเก็บค่าแอดเดรสให้เก็บไว้ในแอดเดรส 1300H-1309H ดังตารางข้างล่าง

แอดเดรสในหน่วยความจำ (ฐาน 16)	ข้อมูล(ฐาน 16)	อุปกรณ์ #
1300	54	DEV1
1301	27	
1302	15	DEV2
1303	30	
1304	74	DEV3
1305	17	
1306	67	DEV4
1307	53	
1308	41	DEV5
1039	77	

จากตารางข้างต้น จะเห็นว่าแอดเดรสของโปรแกรมตอบสนองการอินเทอร์รัพท์ของอุปกรณ์แต่ละชิ้นที่ถูกเก็บไว้นั้น จะใช้เนื้อที่ในหน่วยความจำ 2 ไบท์ โดยเก็บค่าแอดเดรสล่าง (A7-A0) ไว้ในแอดเดรสคู่และค่าแอดเดรสบน (A15-A8) ไว้ในแอดเดรสคี่ และเรียกแอดเดรสที่เก็บค่าแอดเดรสของโปรแกรมตอบสนองการอินเทอร์รัพท์เหล่านี้ว่า VECTOR TABLE ADDRESS.

หลังจากที่ได้กำหนดแอดเดรสที่เก็บค่าแอดเดรสของโปรแกรมตอบสนองการอินเทอร์รัพท์เรียบร้อยแล้ว ต่อไปจะต้องทำการป้อนข้อมูลให้กับรีจิสเตอร์ I ซึ่งในที่นี้คือ 13H (แอดเดรสบนของ VECTOR TABLE) จากนั้นจึงจะกำหนดว่า จะให้อุปกรณ์ชิ้นใดส่งข้อมูลอะไรให้กับ Z80 ซึ่งข้อมูลนี้ก็คือแอดเดรสล่างของ VECTOR TABLE ดังตารางข้างล่าง

อุปกรณ์ #	ข้อมูลที่ต้องส่งออกมา(ฐาน 16)
DEV1	00
DEV2	02
DEV3	04
DEV4	06
DEV5	08

เมื่อได้ทำการออกแบบฮาร์ดแวร์ไว้เรียบร้อยแล้วสมมติว่าอุปกรณ์ DEVE3 ทำการขออินเทอร์รัพท์จาก Z80 เมื่อ Z80 ได้ทำการตอบรับอินเทอร์รัพท์โดยการทำให้ขา $\overline{M1}$ และ $\overline{IORQ}$ ของ Z80 เป็นลอจิก "0" แล้ว DEV3 ก็จะส่งข้อมูล 04H ให้กับ Z80 ซึ่ง Z80 จะนำเอาข้อมูลนี้ไปประกอบข้อมูลในรีจิสเตอร์ I (13H) ได้เป็นแอดเดรสค่า 1304H จากนั้นก็จะนำเอาค่าของข้อมูลในแอดเดรส 1304H (74H) และ 1305H (17H) มาประกอบกันเป็นแอดเดรสของโปรแกรมตอบสนองการอินเทอร์รัพท์ ซึ่งก็คือแอดเดรส 1774H.

ส่วนโปรแกรมที่ใช้ในการทำงานตามตัวอย่างข้างต้นนั้นแสดงดังรูปที่ 2.3.9

2.3.9 ขบวนการ DMA (DIRECT MEMORY ACCESS).

ในการส่งผ่านข้อมูลระหว่างอุปกรณ์ภายนอกกับหน่วยความจำของระบบนั้นโดยทั่วไปแล้วมักจะต้องกระทำผ่าน CPU เสมอ เช่น ในกรณีที่อุปกรณ์ภายนอกต้องการที่จะส่งข้อมูลให้กับหน่วยความจำ อุปกรณ์ภายนอกจะต้องทำการส่งข้อมูลให้กับ CPU ก่อน จากนั้น CPU จึงจะส่งข้อมูลนั้นให้กับหน่วยความจำอีกทีหนึ่ง ซึ่งจะเห็นว่าในการส่งผ่านข้อมูลระหว่างอุปกรณ์ภายนอกกับหน่วยความจำของระบบนั้นจะต้องผ่านการทำงานถึง 2 ขั้นตอนเสมอ ในกรณีที่ส่งผ่านข้อมูลจำนวนมากนั้นการทำงานโดยวิธีดังกล่าวจะทำให้เสียเวลาเป็นอย่างมาก

ดังนั้นเพื่อความรวดเร็วในการส่งผ่านข้อมูล จึงจำเป็นที่จะต้องใช้วิธีอื่นเข้าช่วย และวิธีการที่ใช้ในการนี้เรียกว่า DMA (DIRECT MEMORY ACCESS) ซึ่งก็คือขบวนการที่ช่วยให้การส่งผ่านข้อมูลระหว่างหน่วยความจำกับอุปกรณ์ภายนอกเป็นไปอย่างรวดเร็วยิ่งขึ้น

Z80 จะทำการตรวจสอบสัญญาณที่ขา $\overline{BUSRQ}$ (BUS REQUEST) ใน STATE สุดท้ายของทุก ๆ MACHINE CYCLE ในกรณีที่พบว่าสัญญาณที่ขา $\overline{BUSRQ}$ ได้รับลอจิก "0" (ซึ่งอุปกรณ์ภายนอกส่งเข้ามา) Z80 ก็จะหยุดการทำงานไว้ชั่วขณะจนกว่าระดับสัญญาณที่ขา $\overline{BUSRQ}$ จะกลับเป็น "1" อีกครั้ง และในช่วงเวลาที่

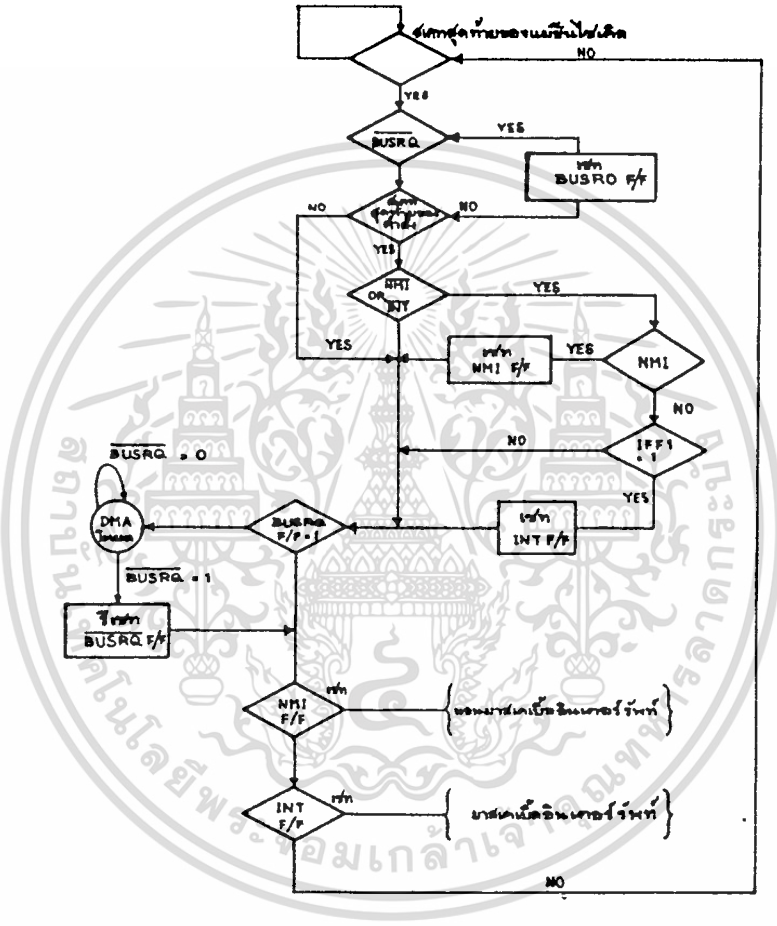
```
: โปรแกรมสำหรับจัดการทำงานของ Z80 ให้อยู่โหมด 2
0000                    CODE  0000
0000    ED5E            IM2             ; สั่งให้ Z80 เลือกอินเทอร์รัพท์
                                        ; โหมด 2
0002    3E11        LD A,13H            ; แอดเดรสบนของ VECTOR
                                        ; TABLE
0004    ED47        LD I,A
0006    FB          EI                  ; สั่งให้ Z80 ENABLE การ
                                        ; อินเทอร์รัพท์
;
;  การสร้าง VECTOR TABLE (อาจจะใช้วิธีอื่นก็ได้)
;
1300                    CODE  1300H     ; แอดเดรสของ  VECTOR TABLE
1300    5427            DEFW  2754H     ; DEV1
1302    1530            DEFW  3051H     ; DEV2
1304    7417            DEFW  1774H     ; DEV3
1306    6753            DEFW  5367H     ; DEV4
1308    4177            DEFW  7741H     ; DEV5
;
```

รูปที่ 2.3.9 โปรแกรมในการใช้งานสำหรับตัวอย่างข้างต้น

หยุดการทำงานนี้ Z80 ก็จะส่งสัญญาณ $\overline{\text{BUSAK}}$ (BUS ACKNOWLEDGE) ให้กับอุปกรณ์ภายนอก เพื่อบอกให้อุปกรณ์ภายนอกทราบว่า Z80 ได้รับทราบการขอใช้บัสแล้ว และ Z80 จะทำให้แอดเดรสบัส (A0-A15), บัสข้อมูล (D0-D7) และบัสควบคุมบางเส้นให้อยู่ในสภาวะ HIGH IMPEDANCE เพื่อให้อุปกรณ์ภายนอกที่ส่ง $\overline{\text{BUSRQ}}$ ให้กับ Z80 สามารถที่จะทำการติดต่อกับหน่วยความจำได้โดยตรง และเมื่อการติดต่อกับหน่วยความจำเสร็จสิ้นลงแล้ว อุปกรณ์ภายนอกจะต้องทำให้สัญญาณ $\overline{\text{BUSRQ}}$ กลับเป็นลอจิก "1" อีกครั้ง เพื่อให้ Z80 ออกจากขบวนการ DMA และกลับไปทำงานในโปรแกรมต่อจากที่ทำค้างไว้.

สำหรับในรูปที่ 2.3.10 จะแสดงโฟลว์ชาร์ตของลำดับการตอบสนองต่อการขออิน

เทอร์รัพท์ ($\overline{INT}$ และ $\overline{NMI}$) และ DMA (BUSRQ).



จากรูป 2.3.10 สามารถที่จะสรุปลำดับของการตอบสนองต่อการขออินเทอร์รัพท์และ DMA ได้ดังนี้

1. Z80 จะทำการตรวจสอบสัญญาณ $\overline{BUSRQ}$ ใน STATE สุดท้ายของทุก ๆ MACHINE CICLE ในกรณีที่ Z80 ตรวจพบสัญญาณ $\overline{BUSRQ}$ (มีลอจิกเป็น "0") ก็จะเข้าสู่ขบวนการ DMA จนกว่าสัญญาณ $\overline{BUSRQ}$ จะถูกถอนออกจากระบบ (สัญญาณ $\overline{BUSRQ}$ มีลอจิกเป็น "1")

2. Z80 จะทำการตรวจสอบการขออินเทอร์รัพท์จากอุปกรณ์ภายนอก ใน STATE สุดท้ายของทุก ๆ INSTRUCTION CYCLE

3. Z80 จะทำการตรวจสอบว่ามีการขออินเทอร์รัพท์แบบ $\overline{NMI}$ (ขอบขาลงของสัญญาณ) หรือไม่ ในกรณีที่พบว่ามีการขออินเทอร์รัพท์แบบ $\overline{NMI}$ นี้ ก็จะเข้าสู่การทำงานในขบวนการของนอน-มาสค์เคเบิลอินเทอร์รัพท์

4. Z80 จะทำการตรวจสอบว่ามีการขออินเทอร์รัพท์แบบ $\overline{INT}$ (ลอจิก"0") หรือไม่ ถ้าหากพบว่ามีการขออินเทอร์รัพท์แบบนี้ก็จะเข้าสู่การทำงานในขบวนการของมาสค์เคเบิลอินเทอร์รัพท์

จากโฟลว์ชาร์ตในรูป 2.3.10 จะสามารถที่จะจัดลำดับการตอบสนองต่อการอินเทอร์รัพท์และ DMA ได้ดังนี้คือ

ลำดับของการตอบสนอง	ขบวนการ
1	DIRECT MEMORY ACCESS
2	NON-MASKABLE INTERRUPT
3	MASKABLE INTERRUPT

2.3.10 การขออินเทอร์รัพท์จากอุปกรณ์หลายชิ้น

ในระบบบางระบบนั้นอาจจะมีอุปกรณ์ที่จะขออินเทอร์รัพท์ได้หลายชิ้น ดังนั้นจึงจำเป็นที่จะต้องทราบถึงวิธีการที่จะทำการเชื่อมต่ออุปกรณ์เหล่านี้เข้าด้วยกันเพื่อให้การขออินเทอร์รัพท์จากอุปกรณ์แต่ละชิ้นเป็นไปอย่างมีประสิทธิภาพ สำหรับในที่นี้จะกล่าวถึงวิธีการที่ใช้โดยทั่วๆ ไป คือPOLLING, PRIORITY INTERRUPT, และ DAISY CHAIN PRIORITY.

2.3.10.1 POLLING

วิธีการ POLLING นี้เป็นวิธีการที่ง่าย สะดวกและประหยัดที่สุด ในระบบที่มีอุปกรณ์หลายชิ้นที่ทำการขออินเทอร์รัพท์ได้ สำหรับวิธีการที่จะเชื่อมต่ออุปกรณ์เหล่านี้เข้าด้วยกันตามวิธี POLLING นี้ก็คือ นำเอาขา $\overline{INT}$ (แอกทีพ "0") ของอุปกรณ์แต่ละชิ้นมาเข้าอินพุทของ AND GATE แล้วนำเอาเอาท์พุทของ AND GATE ไปต่อเข้ากับขา $\overline{INT}$ ของ CPU, ด้วยวิธีนี้ CPU สามารถที่จะรับรู้การขออินเทอร์รัพท์ได้ (ไม่ว่าอุปกรณ์ชิ้นใดจะทำการขออินเทอร์รัพท์) และ

CPU จะตรวจสอบว่าอุปกรณ์ชิ้นใดที่ทำการขออินเทอร์รัพท์ โดยการตรวจสอบระดับลอจิกที่ขาอินพุทของ AND GATE ในกรณีที่มีอุปกรณ์มากกว่า 1 ชิ้นทำการขออินเทอร์รัพท์,CPU ก็จะต้องตัดสินเองว่าจะให้อุปกรณ์ชิ้นใดมีความสำคัญสูงกว่ากัน.

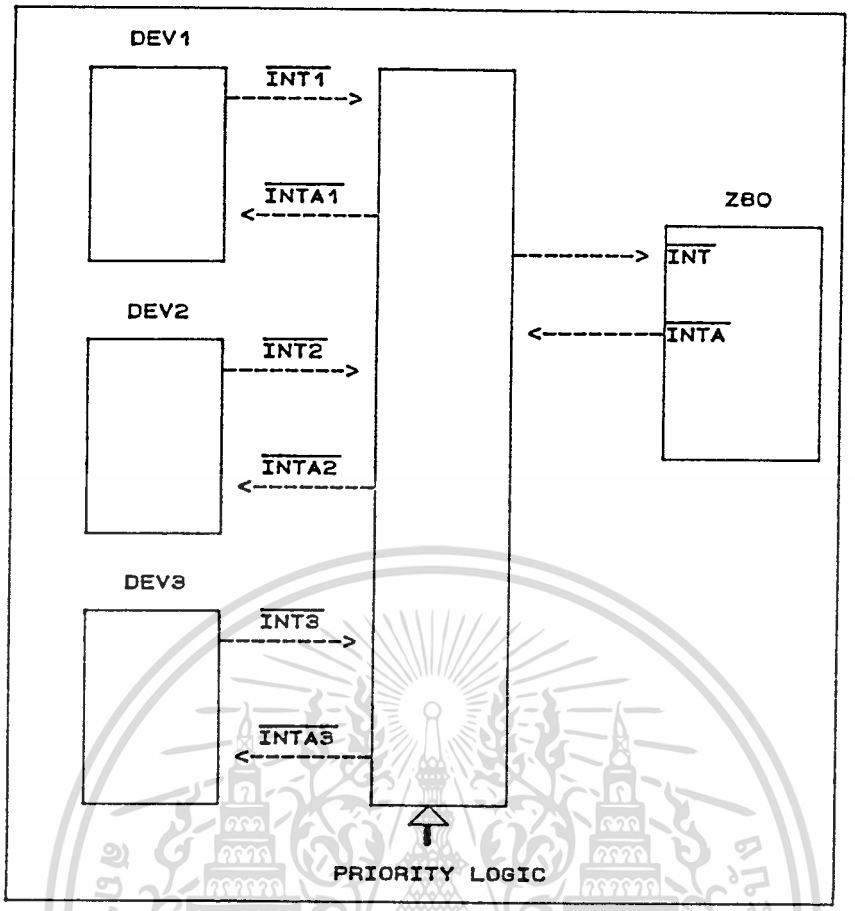
จะเห็นว่าวิธีการนี้ แม้จะง่ายและประหยัดแต่ก็ช้ากว่าวิธีอื่น และด้วยเหตุนี้ทำให้ในระบบที่ต้องการความเร็วในการทำงานสูงหรือในระบบใหญ่ ๆ มักจะไม่นิยมใช้วิธี POLLING นี้

สำหรับวิธีการ POLLING ในอีกรูปแบบหนึ่ง ที่ใช้กันอยู่ทั่วไปในระบบเล็ก ๆ นั้นจะเป็นลักษณะที่ CPU ต้องคอยอ่านข้อมูลจากอุปกรณ์ภายนอกตลอดเวลา เพื่อตรวจสอบว่ามีอุปกรณ์ชิ้นใดบ้างที่ต้องการจะติดต่อกับ CPU ซึ่งจะมีข้อเสียในแง่ที่ว่า CPU จะต้องคอยจนกว่าอุปกรณ์ภายนอกพร้อมที่จะติดต่อด้วย ในขณะที่วิธีการ POLLING ในแบบที่ใช้การอินเทอร์รัพท์ CPU ยังสามารถที่จะทำงานในโปรแกรมได้โดยจะทำการตรวจสอบสถานะของอุปกรณ์ภายนอกเฉพาะในขณะที่ถูกอินเทอร์รัพท์เท่านั้น

2.3.10.2 PRIORITY INTERRUPT.

วิธีนี้นับได้ว่าเป็นวิธีที่มีประสิทธิภาพมากวิธีหนึ่ง ซึ่งช่วยให้ CPU สามารถที่จะต่อกับอุปกรณ์ภายนอกหลาย ๆ ชิ้นที่ขออินเทอร์รัพท์ได้

เมื่ออุปกรณ์ภายนอกแต่ละชิ้นได้รับการตอบรับการอินเทอร์รัพท์จาก CPU แล้วก็จะส่งข้อมูลออกมาเพื่อที่จะส่งให้กับ CPU แต่ระบบที่มีอุปกรณ์ภายนอกหลาย ๆ ชิ้นถ้าปล่อยให้ข้อมูลจากอุปกรณ์ทุกตัวที่ขออินเทอร์รัพท์ (ในกรณีที่มีอุปกรณ์มากกว่า 1 ตัวทำการขออินเทอร์รัพท์) ถูกส่งออกมาพร้อม ๆ กันแล้ว ก็จะทำให้เกิดความผิดพลาดขึ้นได้ ดังนั้นจึงจำเป็นที่จะต้องใช้อุปกรณ์ที่สามารถช่วย CPU ในการที่จะเลือกให้สัญญาณ INTERRUPT ACKNOWLEDGE กับอุปกรณ์ภายนอกได้อย่างถูกต้อง สำหรับฮาร์ดแวร์ที่ใช้ในการทำหน้าที่นี้เรียกว่า PRIORITY LOGIC.



รูปที่ 2.3.11 บล็อกไดอะแกรมแสดงวิธีจัดการอินเทอร์รัพท์แบบ PRIORITY INTERRUPT

จากรูปที่ 2.3.11 จะเห็นว่าอุปกรณ์ภายนอกที่จะขออินเทอร์รัพท์จะต้องส่งสัญญาณ INTERRUPT REQUEST ให้กับ PRIORITY LOGIC ซึ่งจะพิจารณาว่าจะส่ง INTERRUPT REQUEST ของอุปกรณ์ตัวใดออกไปเมื่อใด และเมื่อ CPU ตอบรับการอินเทอร์รัพท์แล้ว PRIORITY LOGIC ก็จะทำการจัดส่ง INTERRUPT ACKNOWLEDGE ให้กับอุปกรณ์ตัวนั้น

ข้อได้เปรียบของวิธีนี้ก็คือ การจัดการอินเทอร์รัพท์ด้วยวิธีนี้ CPU สามารถที่จะรับการอินเทอร์รัพท์ได้อย่างรวดเร็วและมีประสิทธิภาพ แต่ข้อเสียก็คือ ฮาร์ดแวร์ที่ใช้มีความยุ่งยากมาก ซึ่งทำให้ค่าใช้จ่ายสูงตามไปด้วย

2.3.10.3 DAISY CHAIN PRIORITY

วิธีการจัดการอินเทอร์รัพท์ในแบบนี้นั้น เป็นรูปแบบการจัดลำดับความสำคัญที่ใช้กับชิพซัพพอร์ทของ Z80 โดยที่ชิพซัพพอร์ทของ Z80 จะมีขา IE1 และ IE0 ไว้สำหรับทำหน้าที่โดยเฉพาะ ซึ่งจะได้กล่าวถึงในบทต่อไป ขณะที่อุปกรณ์ที่มีลำดับความสำคัญสูงกว่ากำลังทำการขออินเทอร์รัพท์อยู่นั้น อุปกรณ์ที่มีลำดับความสำคัญต่ำกว่าก็จะถูก DISABLE การอินเทอร์รัพท์ไว้หมด เช่น ในกรณีที่ DEV1 ต้องการที่จะขออินเทอร์รัพท์ DEV2-DEV4 ก็จะถูก DISABLE

การอินเทอร์รัพท์ไว้หมด ทำให้ DEV2-DEV4 ขออินเทอร์รัพท์ไม่ได้ แต่อย่างไรก็ตามในกรณีที่อุปกรณ์ที่ถูก DISABLE ไว้ต้องการที่จะขออินเทอร์รัพท์ ก็สามารถที่จะทำได้ในทันทีที่ถูก ENABLE (อุปกรณ์ที่มีลำดับความสำคัญสูงกว่าเสร็จจากการอินเทอร์รัพท์แล้ว) ดังนั้นการอินเทอร์รัพท์ของอุปกรณ์ภายนอกจะไม่เกิดการสูญหายในระหว่างที่ถูก DISABLE เลย อีกกรณีหนึ่งที่สำคัญของการอินเทอร์รัพท์แบบ DAISY CHAIN ก็คือ ในขณะที่อุปกรณ์ที่มีลำดับความสำคัญในการขออินเทอร์รัพท์ต่ำกว่ากำลังขออินเทอร์รัพท์อยู่นั้น อุปกรณ์ที่มีลำดับความสำคัญสูงกว่าสามารถที่จะขออินเทอร์รัพท์แทรกขึ้นมาได้ และจะได้รับสัญญาณตอบรับการอินเทอร์รัพท์แทน

2.4 การใช้ 8255 PIA กับ Z80

ได้กล่าวถึงรายละเอียดของระบบไมโครโปรเซสเซอร์ การติดต่อกับอุปกรณ์ภายนอก (ขบวนการอินพุทและเอาท์พุท) และการอินเทอร์รัพท์มาแล้ว ต่อไปจะกล่าวถึงอุปกรณ์ซึ่งช่วยให้การติดต่อกับระบบภายนอกของระบบไมโครโปรเซสเซอร์มีประสิทธิภาพยิ่งขึ้น อุปกรณ์ชิ้นแรกที่จะกล่าวถึงนี้ ได้แก่ 8255 PIA (PROGRAMMABLE INTERFACE ADAPTER).

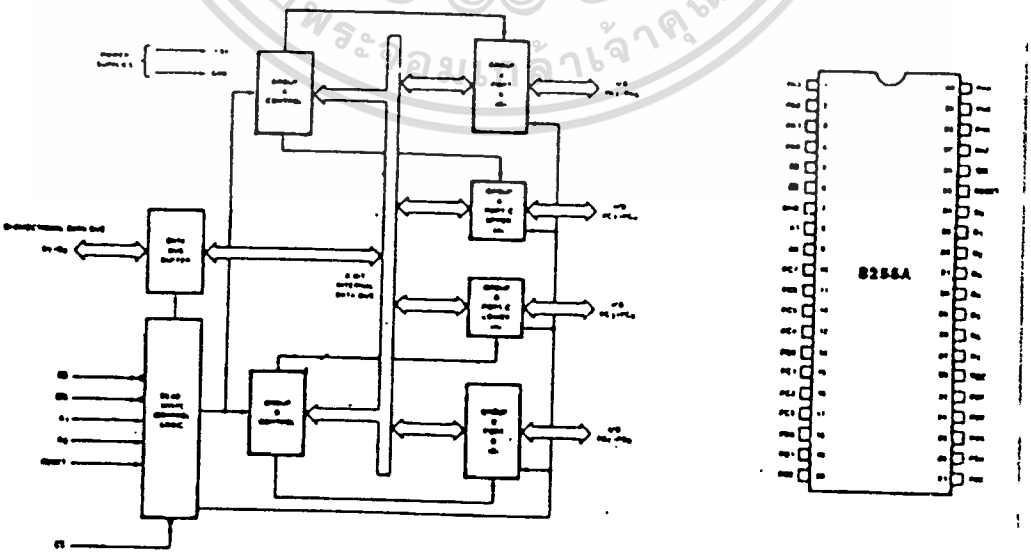
2.4.1 รายละเอียดเกี่ยวกับ 8255

8255 เป็นอุปกรณ์ LSI (LARGE SCALE INTEGRATED CIRCUIT) บรรจุอยู่ใน PACKAGE 40 ขาแบบ DIP (DUAL-IN-LINE PACKAGE) เริ่มผลิตโดยบริษัท INTEL COORPERATION ผู้ผลิตไมโครโปรเซสเซอร์เบอร์ 8080 จุดประสงค์เพื่อใช้งานร่วมกับ 8080 โดยเฉพาะแต่ในภายหลังได้มีการนำ 8255 ไป ประยุกต์ใช้งานร่วมกับไมโครโปรเซสเซอร์เบอร์อื่น ๆ รวมทั้ง Z80 ด้วย หากผู้อ่านเข้าใจการใช้งาน 8255 กับ Z80 ที่จะกล่าวถึงในบทนี้แล้วก็จะนำไปประยุกต์ใช้งานในลักษณะอื่น ๆ ได้ไม่ยากนัก

8255A/8255A-5
PROGRAMMABLE PERIPHERAL INTERFACE

- MCS-85™ Compatible 8255A-5
- 24 Programmable I/O Pins
- Completely TTL Compatible
- Fully Compatible with Intel® Microprocessor Families
- Improved Timing Characteristics
- Direct Bit Set/Reset Capability Easing Control Application Interface
- 40-Pin Dual In-Line Package
- Reduces System Package Count
- Improved DC Driving Capability

The Intel® 8255A is a general purpose programmable I/O device designed for use with Intel® microprocessors. It has 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. In the first mode (MODE 0), each group of 12 I/O pins may be programmed in sets of 4 to be input or output. In MODE 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining 4 pins, 3 are used for handshaking and interrupt control signals. The third mode of operation (MODE 2) is a bidirectional bus mode which uses 8 lines for a bidirectional bus, and 5 lines, borrowing one from the other group, for handshaking.



รูป 2.4.1 แสดงบล็กไดอะแกรมและการวางตำแหน่งของ 8255

รูป 2.4.1 นี้แสดงบล๊อกไดอะแกรม ของ 8255 ซึ่งหน้าที่ของแต่ละบล๊อกมีดังต่อไปนี้คือ

บล๊อกกลุ่มแรกที่จะพูดถึงนี้ ได้แก่ บล๊อกจำนวน 4 บล๊อก ที่อยู่ทางด้านขวาของรูป ซึ่งจะเป็นส่วนที่เชื่อมต่อกับอุปกรณ์ภายนอกอื่น ๆ โดยมีสาย PA0-PA7,PB0-PB7 และ PC0-PC7 เป็นทางผ่านของข้อมูลระหว่างอุปกรณ์ภายนอกกับ 8255 สายสัญญาณเหล่านี้จะถูกแบ่งออกเป็น 3 I/O พอร์ทได้แก่พอร์ท A (PA), พอร์ท B (PB) และพอร์ท C (PC) พอร์ทเหล่านี้แต่ละพอร์ทสามารถเป็นได้ทั้งพอร์ทอินพุทและเอาท์พุท และแต่ละบล๊อกจะมีสายสัญญาณเชื่อมเข้ากับบัสข้อมูลภายในของ 8255

บล๊อกกลุ่มถัดมา ได้แก่ GROUP A CONTROL และ GROUP B CONTROL ซึ่งจะเป็นตัวกำหนดลักษณะการทำงานของทั้ง 3 I/O พอร์ท (8255 มีลักษณะการทำงานที่แตกต่างกันอยู่ 3 โหมด สามารถกำหนดได้โดยการโปรแกรมส่ง CONTROL WORD ให้กับ 8255) จากรูป 2.4.1 จะเห็นว่าพอร์ท C นี้จะประกอบด้วยพอร์ทขนาด 4 บิท 2 พอร์ท กลุ่มหนึ่งจะถูกควบคุมโดย GROUP A CONTROL และอีกกลุ่มหนึ่งจะถูกควบคุมโดย GROUP B CONTROL

บล๊อกกลุ่มสุดท้ายที่จะกล่าวถึงได้แก่ DATA BUS BUFFER และ READ/WRITE CONTROL LOGIC ซึ่งบล๊อกเหล่านี้จะเป็นส่วนที่ติดต่อกับ CPU , DATA BUS BUFFER นี้จะเป็นบัฟเฟอร์ให้กับบัสข้อมูลของ CPU ส่วน READ/WRITE CONTROL LOGIC จะเป็นส่วนที่ควบคุมให้ข้อมูลเข้าหรือออกจากรีจิสเตอร์ภายใน ตัวที่ถูกต้องและในเวลาที่เหมาะสม

2.4.2 รายละเอียดการจัดเรียงขาของ 8255

ในส่วนนี้จะพิจารณาหน้าที่ของขาแต่ละขาของ 8255 ซึ่งข้อมูลเหล่านี้จะมีประโยชน์ในการเชื่อมต่อเข้ากับระบบบัสของ CPU สำหรับการจัดขาแสดงไว้ในรูปที่ 2.4.1 รายละเอียดของแต่ละมีดังนี้คือ

D0-D7 : เป็นสายข้อมูลอินพุท / เอาท์พุทแบบสองทิศทาง (BI-DIRECTIONAL BUS) จะเป็นทางผ่านของข้อมูลระหว่างพอร์ทต่าง ๆ ของ 8255 กับบัสข้อมูลของ Z80

$\overline{CS}$ (CHIP SELECT INPUT) : เมื่อขานี้มีสถานะลอจิกเป็น "0" CPU จะสามารถที่จะอ่านหรือเขียนข้อมูลกับ 8255 ได้

$\overline{RD}$ (READ INPUT) : เมื่อขานี้มีสถานะลอจิกเป็น "0" และสัญญาณ $\overline{CS}$ มีลอจิกเป็น "0" ข้อมูลจาก 8255 จะปรากฏสู่ระบบบัสข้อมูล CPU ก็จะสามารถอ่านข้อมูลออกไปได้ (ในการตั้งชื่อของขาสัญญาณนี้จะถือเอา CPU เป็นหลัก)

$\overline{WR}$ (WRTIE INPUT) : เมื่อขานี้มีสถานะลอจิกเป็น "0" และขาสัญญาณ $\overline{CS}$ มีลอจิกเป็น "0" ข้อมูลจากระบบบัสข้อมูลจะถูกเขียนเข้าไปยัง 8255 ได้

A0-A1 (ADDRESS INPUT) : จะเป็นตัวกำหนดการเลือกใช้รีจิสเตอร์ภายในของ 8255

RESET : เมื่อขานี้มีสถานะเป็น "1" 8255 จะอยู่ในสภาวะรีเซ็ททุก ๆ พอร์ทของ 8255 จะถูกเซ็ทให้อยู่ในโหมดอินพุท

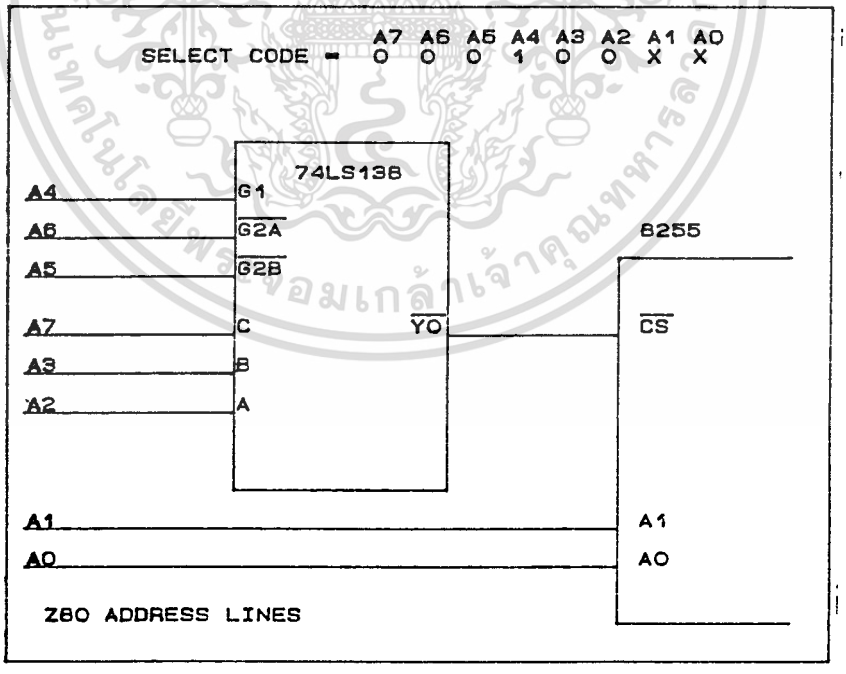
PA0-PA7,PB0-PB7 : ขาสัญญาณเหล่านี้จะถูกใช้เป็นพอร์ท I/O ขนาด 8 บิทใช้ต่อเข้ากับอุปกรณ์ภายนอกอื่น ๆ

PC0-PC7 : ขาสัญญาณนี้ถูกใช้เป็นพอร์ท I/O ขนาด 8 บิท เช่นเดียวกับ PA0-PA7 และ PB0-PB7 แต่กลุ่มของขาสัญญาณเหล่านี้สามารถแบ่งออกเป็น 2 กลุ่ม โดยแต่ละกลุ่มมีขนาด 4 บิทได้ กลุ่มแรกจะใช้ควบคุม PB0-PB7 และกลุ่มที่ 2 ใช้ควบคุม PA0-PA7

2.4.3 การต่อ 8255 เข้ากับ Z80

ในกาารต่อ 8255 เข้ากับระบบของ Z80 นั้น สัญญาณต่าง ๆ ที่เกิดขึ้นจะเหมือนกับขบวนการติดต่อกับ I/O โดยจะต้องเอา สัญญาณ A0-A7 จาก Z80 มาถอดรหัสเพื่อสร้างสัญญาณเลือกพอร์ท แต่เนื่องจาก 8255 มีขา ADDRESS INPUT อยู่แล้ว 2 ขา (A0,A1) ซึ่งโดยปกติแล้วขา A0,A1 นี้จะต่อเข้าโดยตรงกับ A0,A1 จากบัสแอดเดรส นั่นคือ 8255 หนึ่งตัวจะใช้ค่าพอร์ทแอดเดรสถึง 4 ค่า (2^2) ส่วนสัญญาณอีก 6 เส้น (A2-A7) จะนำไปถอดรหัสเพื่อทำสัญญาณเลือก ชิพ (CHIP SELECT) ให้แก่ 8255

ในที่นี้จะสมมติให้ 8255 มีพอร์ทแอดเดรสอยู่ที่ 10H,11H,12H และ 13H ซึ่งวิธีหนึ่งที่สามารถจะถอดรหัสพอร์ทเหล่านี้ได้ แสดงไว้ในรูป 2.4.2

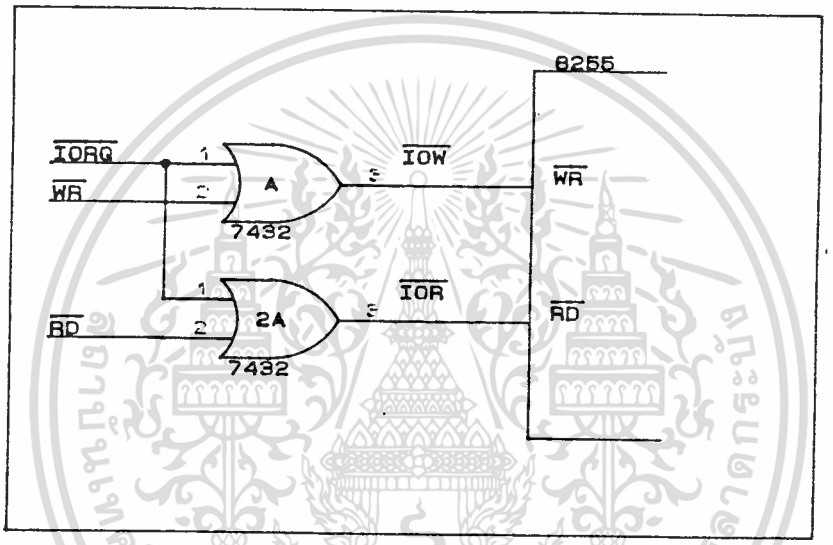


รูปที่ 2.4.2 : แสดงผังวงจรการถอดรหัสการเลือกพอร์ทที่ติดต่กับ 8255

จากรูปที่ 2.4.2 นี้จะเห็นว่าขาอินพุท $\overline{CS}$ จะแอคทีฟก็ต่อเมื่อ A7-A2 มีเท่ากับ

000400XXB (2 บิทล่างจะใช้เพื่อเลือกใช้รีจิสเตอร์ภายใน 4 ตัว)

ขั้นต่อไปที่จะต้องทำคือ การต่อขา $\overline{RD}$ และ $\overline{WR}$ ของ 8255 เข้ากับสัญญาณควบคุม $\overline{IOR}$ และ $\overline{IOW}$ ของระบบ การที่ไม่ต่อขา $\overline{RD}$ และ $\overline{WR}$ เข้าโดยตรง เพราะในตัวอย่างวิธีการถอดรหัสนี้ อาจจะเกิดกรณีที่ A7-A0 มีค่าตรงกับ 000100XXB ซึ่งจะทำให้เกิดการอ่านหรือเขียนข้อมูลกับ 8255 โดยไม่ต้องการได้ ในการแก้ปัญหานี้จึงใช้สัญญาณ $\overline{IORQ}$ จาก CPU มาทำเป็นสัญญาณ $\overline{IOR}$ และ $\overline{IOW}$ เพื่อแยกว่าเป็นการติดต่อกับ I/O ไม่ใช่หน่วยความจำ ดังแสดงในรูป 2.4.3



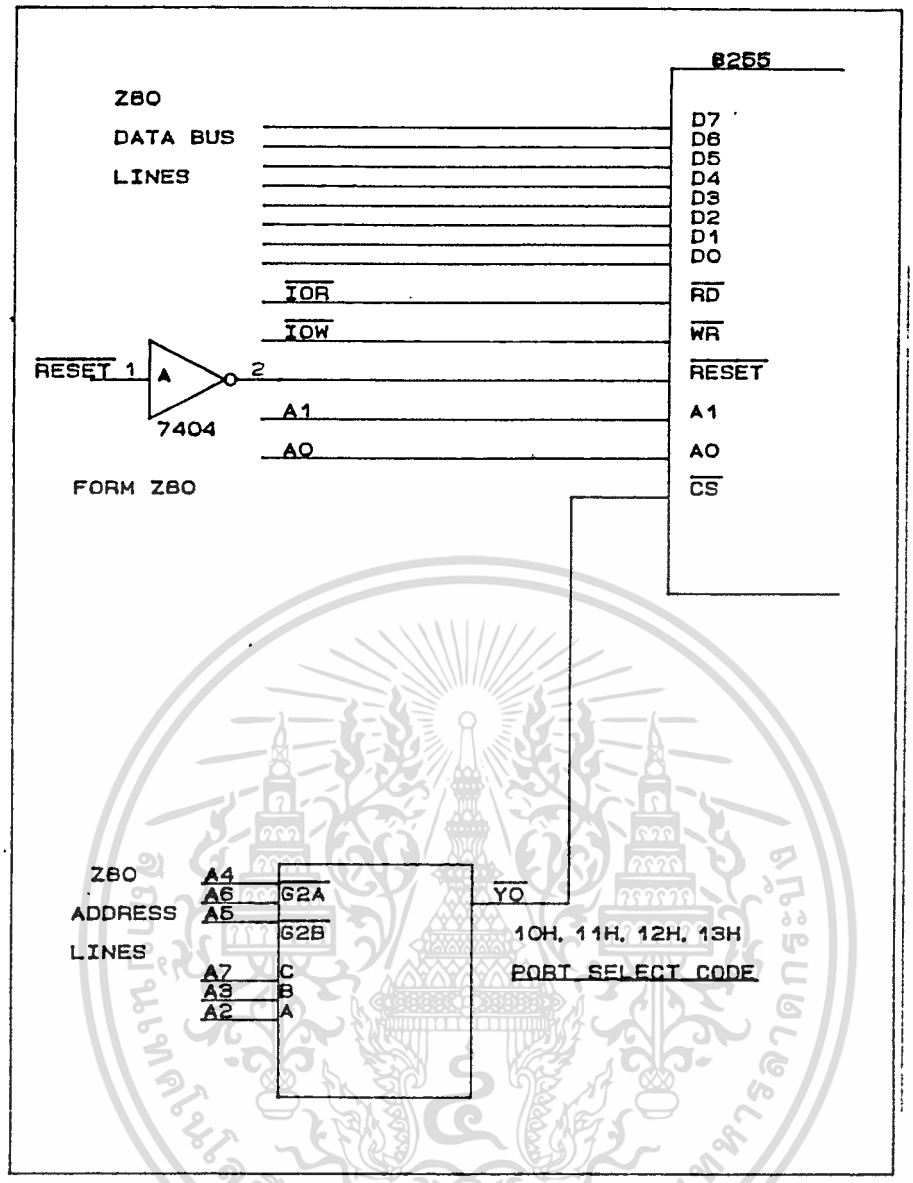
รูปที่ 2.4.3 : แสดงวิธีการต่อขา $\overline{WR}$ และ $\overline{RD}$ เข้ากับระบบของ Z80

ในการต่อขา RESET ของ 8255 ซึ่งจะแอคทีฟที่ลอจิก "1" เข้ากับขา RESET ของ Z80 ซึ่งแอคทีฟที่ลอจิก "0" นั้นจะต้องใช้ INVERTER คั่นกลางเสียก่อน

ในการต่อสายข้อมูล D0-D7 ของ 8255 เข้ากับระบบบัสข้อมูลของระบบจะสมมติว่าไม่มีการโหลดบนบัสข้อมูล ดังนั้นจึงสามารถต่อสายสัญญาณเหล่านี้เข้าโดยตรงกับระบบบัสข้อมูล ดังแสดงวงจรสมบูรณ์ของการเชื่อมต่อ 8255 เข้ากับระบบของ Z80 ในรูป 2.4.4

2.4.4 8255 READ และ WRITE REGISTER.

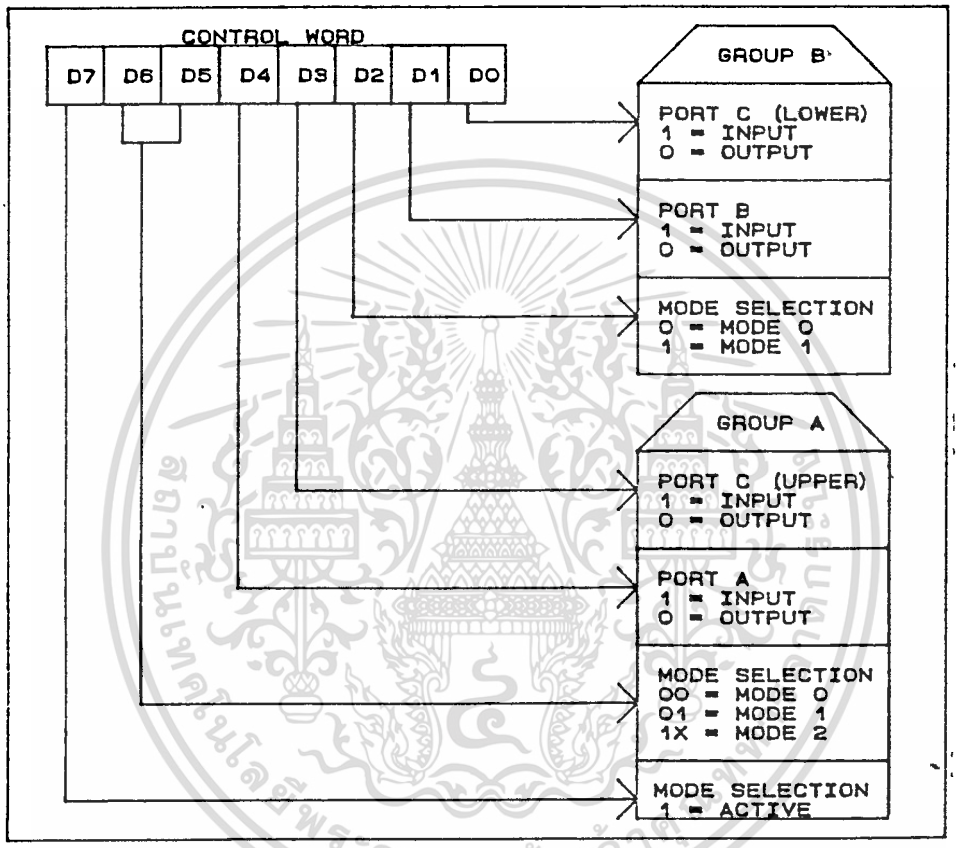
ขณะนี้ได้ทำการต่อ 8255 เข้ากับระบบของ Z80 แล้ว แต่ไปจะศึกษาการโปรแกรมใช้งาน 8255 เพื่อที่จะให้ทำงานตามที่ต้องการได้ จะเริ่มต้นพิจารณาที่รีจิสเตอร์ภายใน 4 ตัวของ 8255 สำหรับในตัวอย่างการถอดรหัสนี้ ตำแหน่งของรีจิสเตอร์จะอยู่ที่แอดเดรส 10H, 11H, 12H และ 13H ซึ่งรายละเอียดของรีจิสเตอร์เหล่านี้มีดังนี้คือ



รูปที่ 2.4.4 : แสดงผังวงจรสมบูรณ์ของการเชื่อมต่อ 8255 เข้ากับระบบของ Z80

DEVICE PIN				REGISTER NAME
RD	WR	A1	AO	
1	0	0	0	WRITE PORT A DATA
0	1	0	0	READ PORT A DATA
1	0	0	1	WRITE PORT B DATA
0	1	0	1	READ PORT B DATA
1	0	1	0	WRITE PORT C DATA
0	1	1	0	READ PORT C DATA
1	0	1	1	WRITE CONTROLA DATA
0	1	1	1	ILLEGAL READ REGISTER

หน้าที่ของรีจิสเตอร์หมายเลข 0-2 จะถูกกำหนดลักษณะการทำงานจากรีจิสเตอร์หมายเลข 3 (รีจิสเตอร์ควบคุม) รูปที่ 2.4.5 จะแสดงรายละเอียดของแต่ละบิทของรีจิสเตอร์ควบคุมนี้ต่อไป จะกล่าวถึงลักษณะการทำงานของ 8255 ทั้ง 3 โหมด และการโปรแกรมให้อยู่ในโหมดต่าง ๆ ได้ดังต่อไปนี้คือ



รูปที่ 2.3.5 แสดงรายละเอียดแต่ละบิทของรีจีสเตอร์ควบคุมของ 8255

2.4.5 โหมด 0 : BASIC REGISTER I/O.

ในการเซ็ท 8255 ให้อยู่ในโหมด 0 นั้นจะต้องส่งคำสั่งควบคุม (CONTROL WORD) ให้แก่รีจิสเตอร์ควบคุมก่อน คำสั่งควบคุมนี้จะกำหนดลักษณะการทำงานให้แก่แต่ละพอร์ทของ 8255 ตัวอย่างหนึ่งของคำสั่งควบคุมที่จะส่งให้ 8255 ทำงานอยู่ในโหมด 1 นี้ได้แก่

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0

จากรูปที่ 2.4.5 จะเห็นว่า

บิท D7 เป็นตัวกำหนดว่าเป็นคำสั่งควบคุม (CONTROL WORD)

บิท D6 และ D5 กำหนดโหมดการทำงานของ พอร์ท A D6,D5 มีค่าเป็น "0"

แสดงว่าอยู่ในโหมด 0

บิท D4 = "0" กำหนดให้ พอร์ท A เป็นพอร์ทเอาท์พุท

บิท D3 = "0" เซ็ทพอร์ท C 4 บิทบนเป็นพอร์ทเอาท์พุท

บิท D2 = "0" เซ็ทโหมดของพอร์ท B ให้พอร์ท B อยู่ในโหมด 0

บิท D1 = "0" เซ็ทพอร์ท B เป็นพอร์ทเอาท์พุท

บิท D0 = "0" เซ็ทพอร์ท C ให้ 4 บิทล่างเป็นพอร์ทเอาท์พุท

คำสั่งควบคุมนี้จะกำหนดให้พอร์ททั้ง 3 ของ 8255 ทำงานอยู่ในโหมด 0 และเป็นพอร์ทเอาท์พุทซึ่งจะได้สายสัญญาณซึ่งสามารถติดต่อกับอุปกรณ์ภายนอกได้ถึง 24 สาย คำสั่งของ Z80 ที่จะเซ็ทให้ 8255 อยู่ในลักษณะดังกล่าวได้แก่

```
LD A, 80J          :  เซ็ทคำสั่งควบคุม
OUT (13H),A        :  ส่งคำสั่งควบคุมให้ 8255
```

เมื่อ Z80 ทำการ EXECUTE คำสั่งข้างต้นแล้ว 8255 จะถูกเซ็ทให้พอร์ททุกพอร์ท เป็นพอร์ทเอาท์พุท และอยู่ในโหมด 0 จะสามารถส่งข้อมูลออกไปยังพรอ์ทต่าง ๆ ได้ด้วยคำสั่ง OUT ของ Z80 ตัวอย่างเช่น ต้องการส่ง 23H ไปยังพรอ์ท A , 41H ไปยังพอร์ท B และ 73H ไปยังพอร์ท C จะต้องให้ Z80 ทำตามโปรแกรมลักษณะดังนี้

```
LD A, 32H          : เซ็ทข้อมูลให้พอร์ท A
OUT (10H),A        : ส่งข้อมูลให้พอร์ท A
LD A, 41H          : เซ็ทข้อมูลให้พอร์ท B
OUT (11H),A        : ส่งข้อมูลให้พอร์ท B
LD A, 73H          : เซ็ทข้อมูลให้พอร์ท C
OUT (12H),A        : ส่งข้อมูลให้พอร์ท C
```

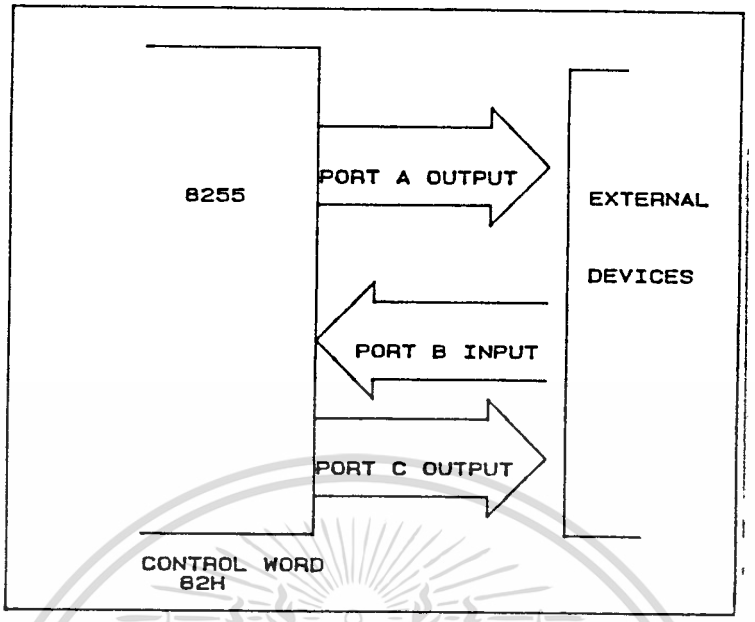
หลังจากที่คำสั่งเหล่านี้ถูก EXECUTE แล้วพอร์ท A,B และ C ของ 8255 จะมีข้อมูลต่าง ๆ ที่ส่งไปให้ปรากฏอยู่

ในการทำงานในโหมด 0 ของ 8255 นี้อาจจะสั่งให้พอร์ทของ 8255 เป็นอินพุทหรือเอาท์พุทได้อย่างเช่น ให้พอร์ท A และพอร์ C เป็นพอร์ทเอาท์พุทและพอร์ท B เป็นพอร์ทอินพุท จะต้องส่งคำสั่งควบคุมให้แก่รีจิสเตอร์ควบคุมในลักษณะดังนี้คือ

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	1	0

หลังจากที่ส่งคำสั่งควบคุมให้แก่รีจิสเตอร์ควบคุมแล้ว 8255 จะถูกเซ็ทให้มีลักษณะการทำงานดังรูป 2.4.6 จะใช้คำสั่ง IN อ่านข้อมูลมาจากพอร์ท B ได้

```
IN A, (11H)        :  อ่านข้อมูลจากพอร์ท B
```



รูปที่ 2.4.6 : บล็อกไดอะแกรมแสดงลักษณะการทำงานของ 8255 ในโหมด 0 หลังจากส่งคำสั่งควบคุมให้ 8255 แล้ว

ลักษณะการทำงานของพอร์ทต่าง ๆ ที่สามารถกำหนดได้ในโหมด 0 แสดงไว้ ในรูป 2.4.7

2.4.6 ตัวอย่างการใช้งาน 8255 ในโหมด 0

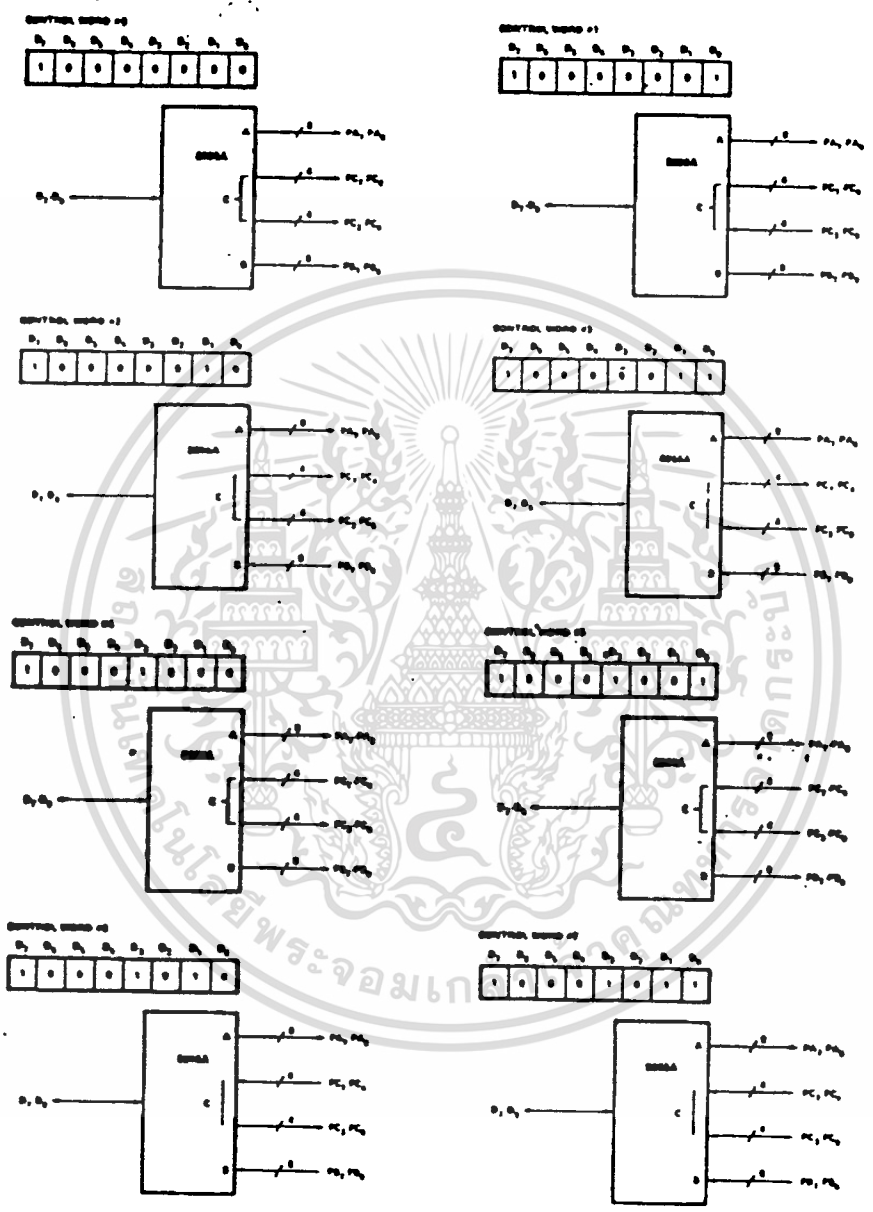
ตัวอย่างที่จะกล่าวถึงนี้เป็นการเชื่อมต่อ (INTERFACE) คีย์บอร์ดเข้ากับระบบของ Z80 ไมโครโปรเซสเซอร์ คีย์บอร์ดที่จะพิจารณาถึงนี้ประกอบด้วยสวิทช์แบบกดติดปล่อยดับ APAR (AINFLW POLW, SINGLE THROW) จัดอยู่ในลักษณะของแมททริกซ์ (MATRIX) ขนาด 4X4 ดังแสดงในรูป 2.4.8 จะมาพิจารณารายละเอียดของการทำงานดังต่อไปนี้

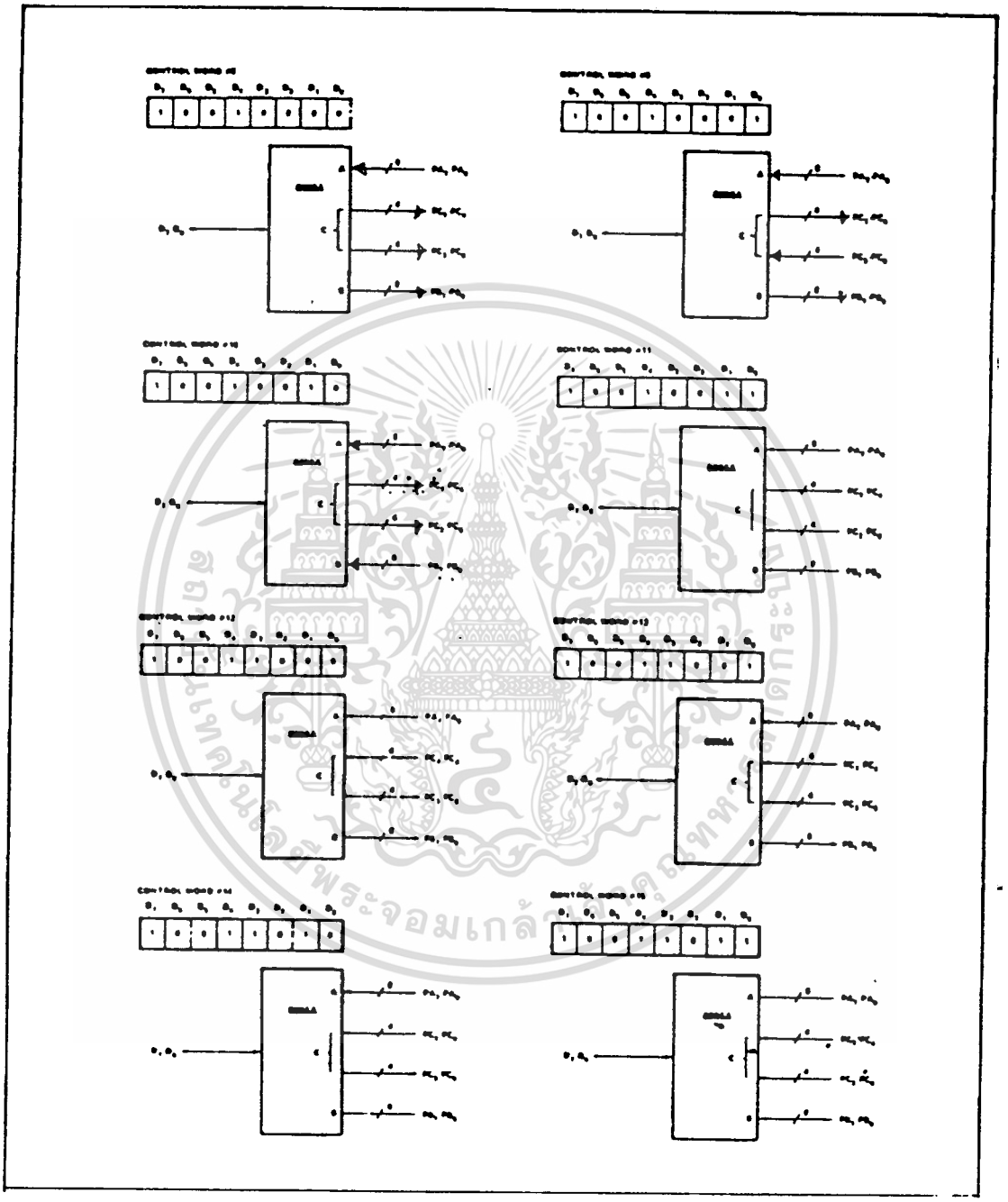
โดยนำเอา PB0-PB3 ของพอร์ท B ซึ่งเป็นพอร์ทเอาท์พุท ไปต่อเข้ากับแนวคอลัมน์ (COLUMN) ของคีย์บอร์ด และนำเอา PA0-PA3 ของพอร์ท A ซึ่งเป็นพอร์ทอินพุทต่อเข้ากับแนวโร (ROW) ของคีย์บอร์ด และทุก ๆ บิทของพอร์ท A ที่ต่อเข้ากับแนวโรจะต้องต่อความต้านทานขนาด 10K กับไฟบวกไว้ (PULL UP) ในการตรวจสอบการกดคีย์บอร์ดของนี้จะให้พอร์ท B สแกนลอจิก "0" ทีละบิทดังนี้

XXXX1110 -> XXXX1101 -> XXXX1011 -> XXXX0111 -> XXXX1110 ->.....

ก่อนที่จะมีการเปลี่ยนค่าข้อมูลเอาท์พุทของพอร์ท B พอร์ท A ก็จะทำการอ่านข้อมูลเข้ามาถ้าไม่มีการกดคีย์บอร์ดแล้วค่าที่ได้จากการอ่านจากพอร์ท A จะมี 4 บิทล่าง

MODE 0 Configurations





(PA0-PA3) เป็นลอจิก "1" หมด ทั้งนี้เพราะมีความต้านทานดึงขึ้นไฟบวกอยู่แต่ถ้ามีการกดคีย์บอร์ดคีย์ใดคีย์หนึ่งแล้วข้อมูลที่อ่านได้จะไม่มีค่าเป็น "1" หมดจะต้องมีบิทใดบิทหนึ่งมีลอจิกเป็น "0" ทั้งนี้เพราะสวิทช์ที่ถูกกดจะทำให้โรว (ROW) กับคอลัมน์ (COLUMN) แตะกันในขณะที่พอร์ท B สแกนลอจิก "0" มาถึงคอลัมน์ของสวิชท์ที่ถูกกดจะทำให้โรวที่ถูกกดมีลอจิกเป็น "0" ด้วย จะทราบว่าคีย์ใดถูกกดโดยการตรวจสอบว่าบิทใดของพอร์ท B และของพอร์ท A มีลอจิกเป็น "0"

รูปที่ 2.4.9 แสดงโปรแกรมการอ่านข้อมูลจากคีย์บอร์ดตามที่ได้อธิบายไว้แล้วในข้างต้น อย่างไรก็ตามโปรแกรมนี้ยังไม่สามารถนำไปใช้งานได้จริง ต้องเพิ่มโปรแกรมการตรวจสอบการกดคีย์บอร์ดค้าง การหน่วงเวลาเพื่อไม่ให้เกิดการผิดพลาดในการอานข้อมูลเข้ามาอีก เป็นต้น

```
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; PROGRAM FOR DETECTING A KEY PRESS ON THE HARDWARE
;  SHOWN IN FIGURE 6.8
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;  PORT B IS THE OUTPUT PORT
;  PORT A BND C ARE THE INPUT PORTS
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
0010            PORTA   EQU  10H
0011            PORTB   EQU  11H
0012            PORTC   EQU  12H
0013            CONP    EQU  13H
0099            CONWD   EQU  99H
;
1800                    CODE  1800H
;
1800  3E99          LD A,CONWD
1802  D313          OUT (CONP),A ;SET UP THE 8255
```

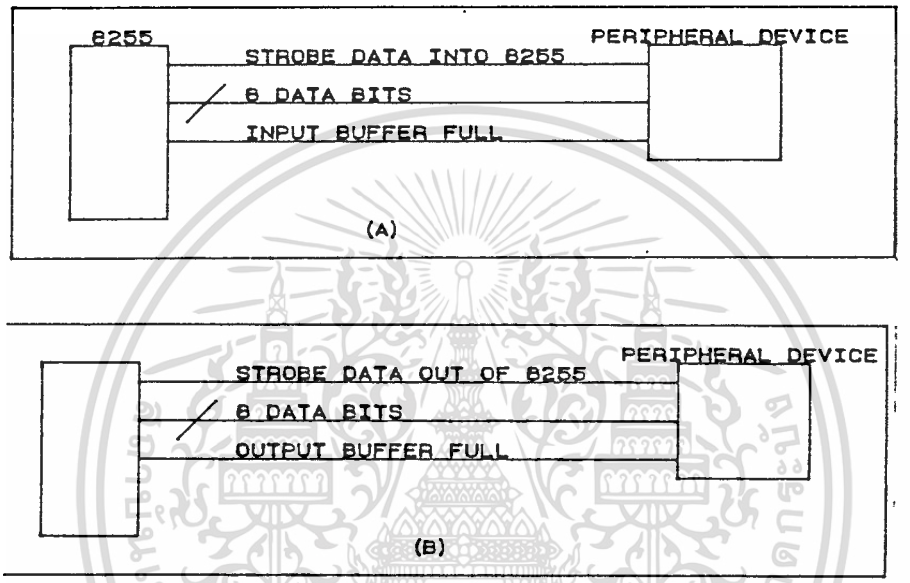
```
1804  0E11          LD C,PORTB    ;CREG = PORTB ADD
;
1806  06FE CLO      LD B,0FEH     ;COLUMN 0 ACTIVE
1808  ED41 COL1     OUT (C),B     ;ASSERT ACTIVE COLUMN
180A  D810          IN A,(PORTTA);READ THE ROWS
180C  2F            CPL           ;COMPLIMENT INPUT WORD
180D  E60F          AND 0FH       ;MASK OFF UNUSED BITS
180F  FE00          CP 00H        ;ANY BITS = 1?
1811  C21F18        JP NZ,KEYIN   ;IF YES THEN KEY PUSHED
1814  CB00          RLC B         ;SHIFT B LEFT, NEXT COL
                                    ACTIVE
1816  78            LD A,B
1817  FEEF          CP 0EFH       ;CHECK FOR LAST COLUMN ACTIVE
1819  C20818        JP NZ,COL1    ;NOT LAST, ASSERT NEXT COLUMN
181C  C30618        JP COL        ;LAST, ASSERT FIRST COLUMN
;
;   HANDLE A KEY PRESSER, NEED COLUMN AND ROW
;
181F  4F    KEYIN   LD C,A        ;ROW POS = 1 IN C REG
1820  78            LD A,H        ;COLUMN POS IN A REG
1821  2F            CPL           ;INVERT POS BYTE
1822  E60F          AND 0FH       ;UNUSED BITS = 0
1824  C9            RET
;
;   RETURN FROM ROUTINE WITH
;   A REG = COLUMN POS, 1 IN ACTIVE COLUMN BIT
;   C REG = ROW POS, 1 IN ACTIVE ROW BIT
;
                      END
```

รูปที่ 2.4.9 : แสดงโปรแกรมการตรวจสอบการกดคีย์บอร์ดตามวงจรในรูป 2.4.8

2.4.7 การใช้งาน 8255 ในโหมด 1.

การทำงานของ 8255 ในโหมด 1 นี้เป็นการทำงานในลักษณะของการ HANDSHAKE, พอร์ท A และพอร์ท B จะเป็นพอร์ทข้อมูล ส่วนพอร์ท C นี้จะถูกใช้เป็นสัญญาณ HANDSHAKE ให้กับพอร์ท B

หลักการรับส่งข้อมูลในวิธีการของ HANDSHAKE นี้ คือการให้อุปกรณ์ภายนอกส่งสัญญาณแสดงสภาวะความพร้อมให้กับ 8255 ดังแสดงในรูป 2.4.10



รูปที่ 2.4.10 : บล็กไดอะแกรมแสดงลักษณะการทำงานของการต่อต่อระหว่าง 8255 กับอุปกรณ์ภายนอกในลักษณะ HANDSHAKE

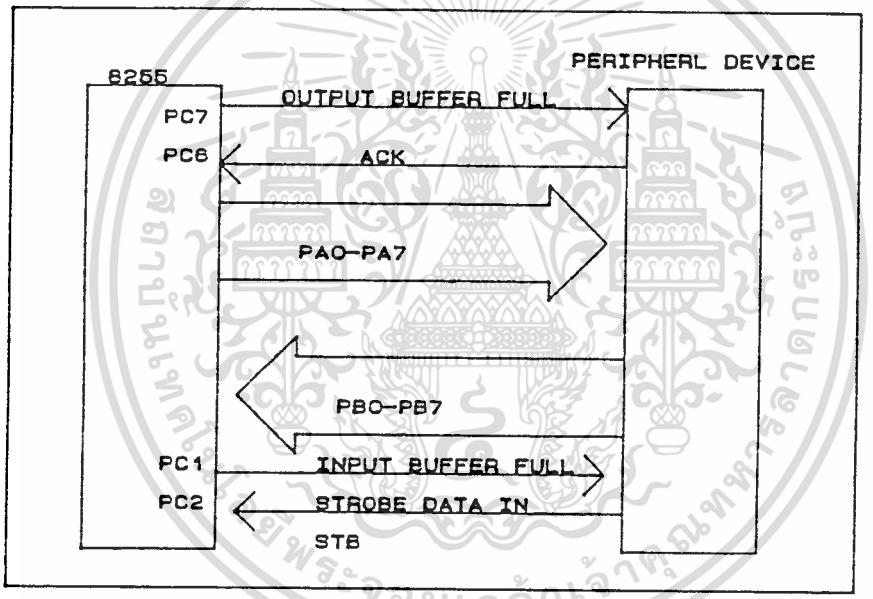
ในรูป 2.4.10A นี้ข้อมูลจะถูกส่งออกจากอุปกรณ์ภายนอกเข้าสู่ 8255 ก่อนที่อุปกรณ์ภายนอกจะเขียนข้อมูลให้แก่ 8255 จะต้องมีการตรวจสอบ INPUT BUFFER FULL FLAG เสียก่อน ถ้า FLAG นี้เป็นจริงแสดงว่าข้อมูลในบัฟเฟอร์ 8255 นี้ยังไม่ถูกอ่านโดย Z80 คือ ข้อมูลจากอุปกรณ์ภายนอกส่งข้อมูลให้กับ 8255 แล้ว Z80 ยังไม่ได้อ่านเอาข้อมูลเข้าไป แต่ถ้า FLAG นี้เป็นเท็จแสดงว่า Z80 อ่านข้อมูลออกไปแล้ว อุปกรณ์ภายนอกก็จะเขียนข้อมูลใหม่ให้ 8255 ได้ และเมื่ออุปกรณ์ภายนอกเขียนข้อมูลให้ 8255 แล้ว INPUT BUFFER FULL ก็จะกลับมาเป็นจริงอีกครั้งหนึ่ง

ในรูป 2.4.10B 8255 จะทำหน้าที่เป็นตัวส่งข้อมูลให้กับอุปกรณ์ภายนอกก่อนที่ 8255 จะส่งข้อมูลให้กับอุปกรณ์ภายนอกนั้นจะต้องเซ็ท OUTPUT BUFFER FULL FLAG เสียก่อนเพื่อบ่งบอกให้อุปกรณ์ภายนอกทราบว่าขณะนี้ 8255 มีข้อมูลพร้อมที่จะส่งออกไปให้แล้ว เมื่ออุปกรณ์ภายนอกส่งสัญญาณ STROBE รับเอาข้อมูลเข้าไปแล้ว INPUT BUFFER FULL FLAG

จะเปลี่ยนเป็นเท็จ เพื่อบ่งบอกให้อุปกรณ์ภายนอกรู้ว่าขณะนี้ไม่มีข้อมูลอยู่ใน 8255 ,Z80 สามารถส่งข้อมูลใหม่ออกไปให้ 8255 ได้

วิธีการทำ HANDSHAKE นี้มีประโยชน์มากในกรณีที่อุปกรณ์ภายนอกทำงานช้ากว่าระบบไมโครโปรเซสเซอร์ด้วยวิธีการนี้ไมโครโปรเซสเซอร์สามารถที่จะส่งข้อมูลให้กับ 8255 แล้วไปทำงานอื่นได้ จนกว่าข้อมูลภายใน 8255 ถูกส่งออกไปแล้ว Z80 ไมโครโปรเซสเซอร์จึงจะส่งข้อมูลใหม่ออกไปให้ ต่อไปนี้จะพิจารณารายละเอียดของการทำ HANDSHAKE ของ 8255 เพิ่มขึ้น

บล็อกไดอะแกรมที่แสดงในรูป 2.4.11 นี้ กำหนดให้พอร์ท A เป็นพอร์ทเอาท์พุท และพอร์ท B เป็นพอร์ทอินพุท (ซึ่งอาจจะอยู่ในลักษณะอื่น ๆ ก็ได้) เพื่อให้ 8255 ทำงานในลักษณะดังกล่าวจะต้องส่งคำสั่งควบคุมให้แก่ 8255 ดังนี้ 10100110B หรือ 0A3H



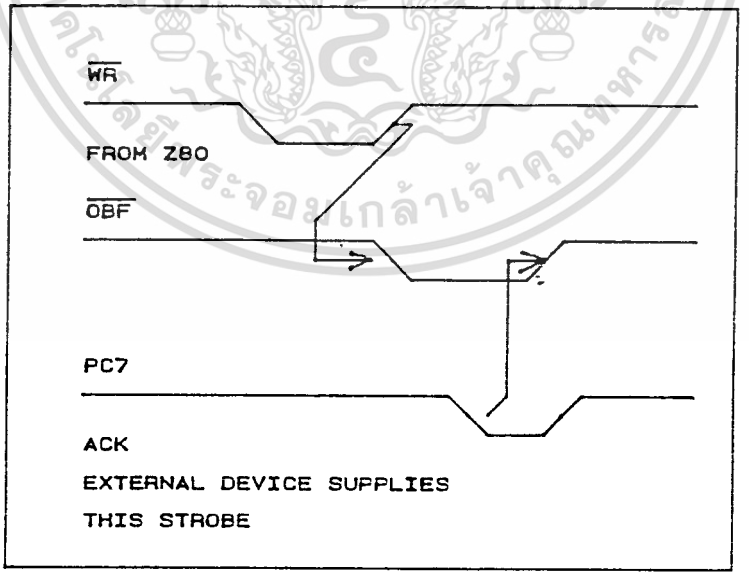
รูปที่ 2.4.11 : แสดงบล็อกไดอะแกรมของ 8255 ที่ให้พอร์ท A เป็นพอร์ทเอาท์พุท และพอร์ทอินพุทในลักษณะ HANDSHAKE

ในรูปนี้ข้อมูลเอาท์พุทจะอยู่บนขา PA0-PA7 ของ 8255,OUTPUT BUFFER FULL อยู่ที่ PC7 สัญญาณ ACKNOWLEDGE จากอุปกรณ์ภายนอกจะอยู่ที่ PC6, ข้อมูลอินพุทอยู่ที่ PB0-PB7, INPUT BUFFER FULL อยู่ที่ PC1 และสัญญาณ STROBE เพื่อให้ 8255 รับเอาข้อมูลเข้าไปอยู่ที่ PC2 รูปที่ 2.4.12 แสดงรายละเอียดของแต่ละบิทของพอร์ท C ในขณะที่ 8255 ทำงานในโหมด 1.

	OUT	IN
PC0	$INTR_B$	$INTR_B$
PC1	IBF_B	OBF_B
PC2	STB_B	ACK_B
PC3	$INTR_A$	$INTR_A$
PC4	STB_A	I/O
PC5	IBF_A	I/O
PC6	I/O	ACK_A
PC7	I/O	OBF_A

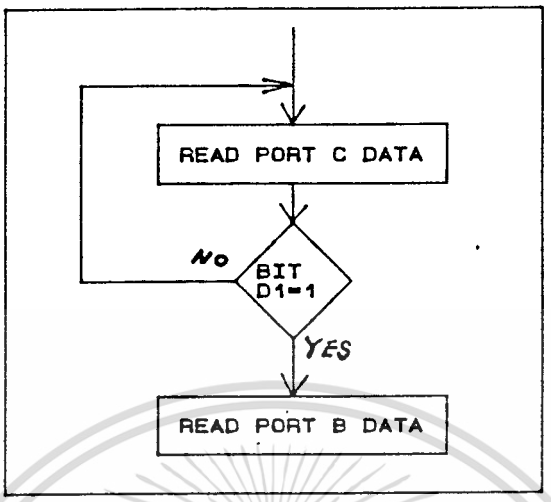
รูปที่ 2.4.12 : แสดงรายละเอียดและหน้าที่ของแต่ละขาของพอร์ท C ในโหมด 1.

ต่อไปจะพิจารณาซอฟท์แวร์ที่ทำให้ 8255 ทำงานในลักษณะดังกล่าวในวิธีการนี้จะไม่ใช้การอินเทอร์รัพท์แต่จะให้ 8255 มาคอยตรวจสอบสถานะของพอร์ท C แทน รูป 2.4.13A และรูป 2.4.13B นี้แสดงไดอะแกรมเวลาที่เกิดขึ้นขณะการรับส่งในขบวนการ HANDSHAKE.



รูปที่ 2.4.12A : แสดงไดอะแกรมเวลาของการส่งผ่านข้อมูลจากพอร์ท A ไปยังอุปกรณ์ภายนอก

แสดงโฟลว์ชาร์ตของขบวนการต่าง ๆ ดังกล่าว และสามารถเขียนเป็นโปรแกรมได้ดังรูป 2.4.17



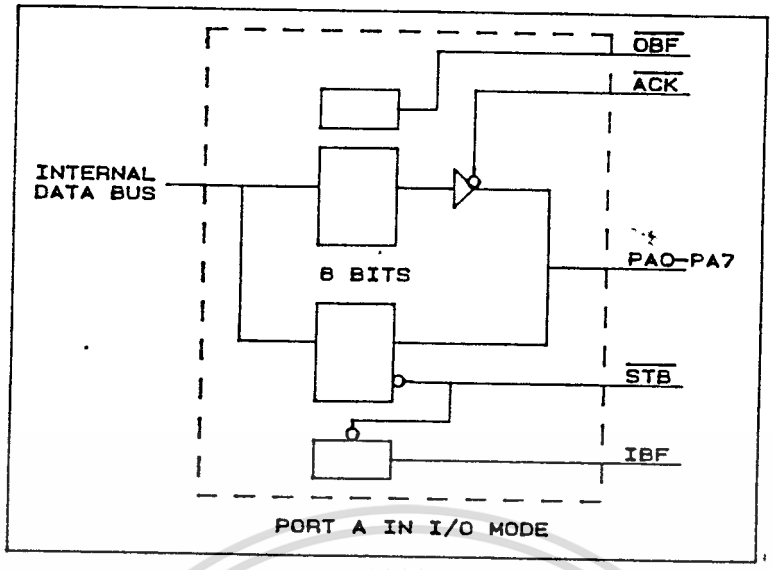
รูป 2.4.16 : FLOW CHART แสดงขบวนการอ่านข้อมูลจากอุปกรณ์ภายนอกเข้ามายังพอร์ท B โดยใช้ 4 บิทล่างของพอร์ท C เป็น HANDSHAKE

```
:
:
1800  DB 12        BACK    IN A, (12H)      ;READ PORT C
1802  CB 4F                BIT 1, C         ;TEST BIT 1=1
1804  CA 00 18             JP Z, BACK       ;DATA NOT STROBED IN YET
1807  DB 11                IN A, (11H)      ;DATA READY READ IT
1809  C9                   RET
                           END
```

รูปที่ 2.4.17 : แสดงโปรแกรมที่สอดคล้องกับโฟลว์ชาร์ตในรูป 2.4.16

2.4.8 การใช้งาน 8255 ในโหมด 2.

การทำงานของ 8255 ในโหมด 2 นี้จะเป็นการใช้งานในลักษณะที่ให้พอร์ท A เป็นพอร์ทข้อมูลแบบสองทิศทาง เมื่อ 8255 ถูกโปรแกรมให้พอร์ท A อยู่ในโหมด 2 นี้แล้วพอร์ท A จะมีลักษณะการทำงานตามบล็อกไดอะแกรมรูปที่ 2.4.18



รูปที่ 2.4.18 : บล็อกไดอะแกรมแสดงการทำงานของพอร์ท A ในโหมด 2

การทำงานในโหมดนี้ก็คือ การใช้พอร์ท A เป็นอินพุทและเอาท์พุทแลทช์ เอาท์พุทแลทช์ (OUTPUT LATCH) หมายถึง การเก็บเอาข้อมูลไว้เพื่อรออุปกรณ์ภายนอกรับเอาข้อมูลออกไป ส่วนอินพุทแลทช์ (INPUT LATCH) หมายถึงการเก็บข้อมูลที่อุปกรณ์ภายนอกส่งเข้ามาเพื่อรอให้ CPU อ่านเข้าไป ต่อไปจะพิจารณาการทำงานของบล็อกไดอะแกรมดังรูป 2.4.18 และศึกษาว่าจะมีการรับหรือส่งข้อมูลกับอุปกรณ์ภายนอกได้อย่างไร

ในการส่งข้อมูลให้อุปกรณ์ภายนอกนั้น ขั้นตอนแรกที่เกิดขึ้นก็คือ CPU จะต้องส่งข้อมูลไปแลทช์ไว้ในพอร์ท A ซึ่งจะทำให้ OUTPUT BUFFER FULL (OBF) ถูกเซ็ทให้เป็นจริง สัญญาณนี้จะบอกให้อุปกรณ์ภายนอกรู้ว่าขณะนี้พอร์ท A มีข้อมูลอยู่และบอกให้ CPU รู้ว่าข้อมูลของพอร์ท A ที่ส่งไปให้นั้นยังไม่ถูกอุปกรณ์ภายนอกอ่านออกไป อุปกรณ์ภายนอกจะต้องส่งสัญญาณ ACK (ACKNOWLEDGE) ให้กับ 8255 สัญญาณนี้จะเป็น ENABLE ให้ข้อมูลที่อยู่ในพอร์ท A ส่งออกไปยัง PA0-PA7 และเป็นการรีเซ็ท OBF เพื่อเป็นการบ่งบอกให้ CPU รู้ว่าข้อมูลที่อยู่ภายในพอร์ท A ถูกอุปกรณ์ภายนอกอ่านออกไปแล้วและสามารถส่งข้อมูลใหม่ไปให้พอร์ท A ได้อีก

ในการรับข้อมูลจากอุปกรณ์ภายนอกนั้น อุปกรณ์ภายนอกต้องตรวจสอบสถานะของ INPUT BUFFER FULL ($\overline{IBF}$) เสียก่อน ถ้า $\overline{IBF}$ มีลอจิกเป็น "1" แสดงว่าขณะนี้พอร์ท A อุปกรณ์ภายนอกจะส่งข้อมูลและสัญญาณ STROBE ($\overline{STB}$) ให้ 8255 สัญญาณนี้จะสั่งให้มีการนำข้อมูลไปแลทช์ไว้ในพอร์ท A และเซ็ทให้ $\overline{IBF}$ เป็นมีสถานะเป็นจริง และ CPU จะตรวจสอบสถานะของ $\overline{IBF}$ ได้โดยการอ่านข้อมูลจากพอร์ท C เมื่อ $\overline{IBF}$ จะมีสถานะลอจิกเป็น"0"

และอุปกรณ์ภายนอกสามารถส่งข้อมูลใหม่เข้ามาได้

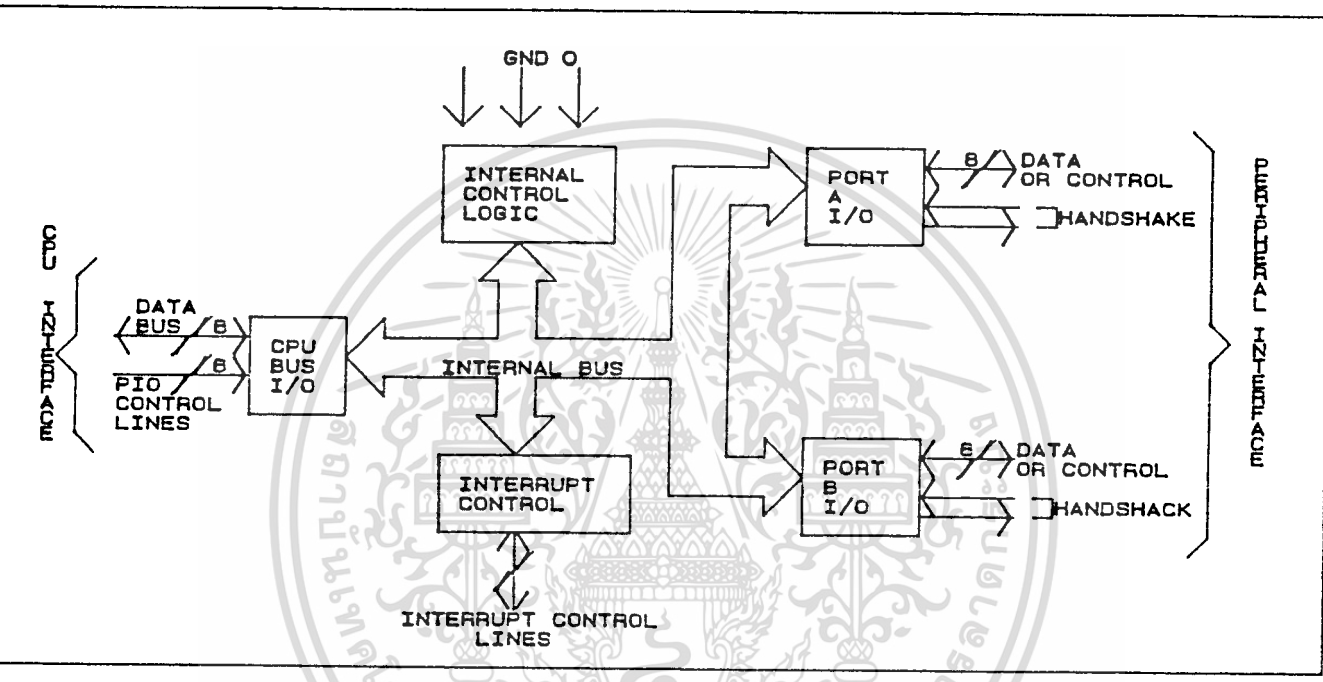
ในการใช้งานในโหมด 2 นี้พอร์ท C จะเป็นตัวแสดงสภาวะของสัญญาณดังกล่าว และรายละเอียดของแต่ละบิทจะแสดงดังรูป 2.4.19

PORT C LINE	DEFINTION
PC0	I/O
PC1	I/O
PC2	I/O
PC3	INTRA
PC4	STBA
PC5	IBFA
PC6	ACKA
PC7	OBFA

รูปที่ 2.4.19 :แสดงรายละเอียดของแต่ละขาของพอร์ท C ในการใช้งานในโหมด 2

2.5 Z80 – PIO

ในบทนี้จะศึกษาถึงชิพซัพพอร์ทอีกตัวหนึ่งของ Z80 คือ MOSTEK MK3881 หรือZ80-PIO นั่นเอง ซึ่งเป็นชิพซัพพอร์ทที่สร้างขึ้นมาสำหรับใช้กับ Z80 โดยเฉพาะ สำหรับ Z80-PIO นี้จะทำหน้าที่เป็นพอร์ท I/O (INPUT/OUTPUT PORT) ที่สามารถจะเลือกโปรแกรมให้ทำงานในลักษณะต่าง ๆ ได้คือ อาจจะให้ทำงานเป็นพอร์ทอินพุทมพอร์ทเอาท์พุท หรือเป็นทั้งพอร์ทอินพุทและพอร์ทเอาท์พุทในเวลาเดียวกันก็ได้ และจะให้มีการทำ HAND SHAKE หรือไม่ก็ได้ ขึ้นกับโหมดการทำงานที่โปรแกรมไว้ ซึ่งมีอยู่ถึง 4 โหมด คือ โหมด 0-โหมด 3.



รูปที่ 2.5.1 : บล็อกไดอะแกรมของ Z80-PIO.

2.5.1 บล็อกไดอะแกรมของ PIO

พิจารณาจากรูปที่ 2.5.1 จะเห็นว่า PIO ประกอบไปด้วยพอร์ท I/O 2 พอร์ท ซึ่งเรียกว่า พอร์ท A และ พอร์ท B โดยมีทั้ง 2 พอร์ทที่มีการทำงานคล้ายกันมากที่เอาท์พุทของทั้ง 2 พอร์ทจะมีบัสข้อมูลอยู่พอร์ทละ 8 เส้น, บัสควบคุมอีกพอร์ทละ 2 เส้น นอกจากนั้นยังมีสายควบคุมอินเทอร์รัพท์ (INTERRUPT CONTROL LINE) อีก 3 เส้น

ทั้งอินพุทและเอาท์พุทของทั้ง 2 พอร์ทสามารถที่จะต่อเข้ากับ TTL ได้โดยตรง (TTL COMPATABLE) โดยที่เอาท์พุทแต่ละเส้นสามารถที่จะขับกระแสได้ 2 mA ที่ลอจิก "0" และ 250 uA ที่ลอจิก "1" ซึ่งเพียงพอที่จะขับ TTL ซึ่งต้องการกระแส 1.6 mA ที่ลอจิก "0" และ 40 uA ที่ลอจิก "1" อย่างไรก็ตาม หากต้องการที่จะขับอุปกรณ์ที่ต้องการกระแสสูงว่าที่ PIO จะขับได้ ก็สามารถที่จะเพิ่มวงจรขับกระแสเข้าไปได้อีก.

บล็อกที่จะกล่าวถึงอีก 2 บล็อกก็คือ INTERNAL CONTROL CLOGIC และ INTERRUPT CONTROL สำหรับ INTERNAL CONTROL LOGIC จะเป็นบล็อกที่คอยจัดว่าข้อมูลภายในจะถูกส่งไปให้กับรีจิสเตอร์ใดที่เวลาใดได้อย่างถูกต้อง ส่วน INTERRUPT CONTROL จะทำหน้าที่คอยจัดการขออินเทอร์รัพท์และรับการตอบรับอินเทอร์รัพท์จาก Z80 ได้อย่างถูกต้อง.

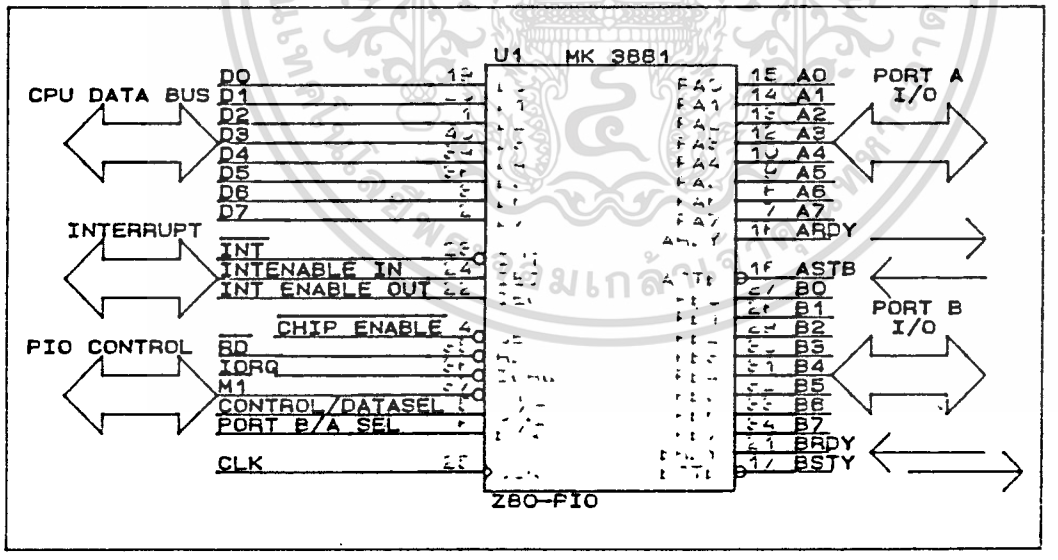
บล็อกสุดท้ายในบล็อกไดอะแกรมก็คือ CPU BUS I/O ซึ่งจะทำหน้าที่ในการติดต่อกับ CPU โดยตรง ที่เอาท์พุทจะประกอบด้วยบัสข้อมูล 8 เส้นและบัสควบคุม (PIO CONTROL LINES) อีก 6 เส้น.

2.5.2 การจัดเรียงขาบน Z80 - PIO

รูปที่ 2.5.2 จะแสดงการจัดเรียงขาบน Z80 - PIO ซึ่งเป็น ซึ่งเป็น PACKAG 40 ขาแบบ DIP ใช้ไฟเลี้ยง +5 V กับ GROUND ทำให้สะดวกต่อการใช้งาน เนื่องจากสามารถที่จะใช้ไฟเลี้ยงร่วมกับไอซีตระกูล TTL ได้พอดี พิจารณารูปที่ 2.5.2 สามารถที่จะแยกขาต่าง ๆ บน Z80 - PIO ออกเป็นกลุ่ม ๆ ตามลักษณะการใช้งานได้ดังนี้ คือ

1. กลุ่มที่ใช้เชื่อมต่อกับ CPU : หน้าที่ของขาในกลุ่มนี้จะทำการติดต่อกับ CPU โดยตรง ซึ่งจะประกอบไปด้วย

1.1) D7 - D0 : ทำหน้าที่ในการที่จะรับหรือส่งข้อมูลให้กับ CPU



1.2) B/$\overline{A}$ SELECT ; (PORT B/A SELECT) : หน้าที่ของขานี้ก็คือทำให้ PIO ทราบว่า CPU ต้องการที่จะติดต่อกับพอร์ท B (กรณีที่ได้รับลอจิก "1")หรือพอร์ทA (กรณีที่ได้รับลอจิก "0") โดยปรกติแล้วขา B/$\overline{A}$ SELECT นี้จะติดกับขา A0 ของ CPU โดยตรง

1.3) C/$\overline{D}$ SELECT ; (CONTROL/DATA SELECT) : ใช้ในการบอก PIO ว่าข้อมูลที่ CPU ส่งให้กับ PIO นั้น เป็นคำสั่งควบคุม (CONTROL WORK; ในกรณีที่ได้รับลอจิก "1") หรือ เป็นข้อมูลจริง (ACTUAL DATA; ในกรณีที่ได้รับลอจิก " 0" ขานี้โดยปกติจะต่อเข้ากับขา A1 ของ CPU โดยตรง

1.4) $\overline{CE}$; (CHIP ENABLE INPUT) : การทำงานของขา $\overline{CE}$ นี้ก็คือเมื่อขานี้ได้รับลอจิก "0" PIO ก็สามารถที่จะรับหรือส่งข้อมูลให้กับอุปกรณ์อินพุท,เอาท์พุทได้และเมื่อขา $\overline{CE}$ นี้ได้รับลอจิก "1" PIO ก็จะไม่สามารถที่จะรับหรือส่งข้อมูลได้

1.5) $\overline{M1}$; (MACHINE CYCLE 1) : สำหรับขานี้โดยปกติจะต่อกับขา M1 ของ Z80 โดยตรง ซึ่ง PIO จะใช้สัญญาณที่ขานี้มาควบคุมการทำงานหลาย ๆ อย่างภายในตัว PIO โดยที่เมื่อทั้ง $\overline{M1}$ และ $\overline{RD}$ เป็นลอจิก "0" ทั้งคู่ก็จะแสดงว่า Z80 กำลังอยู่ในช่วงเฟทช์ออพโคด (FETCH OPCODE) อยู่เมื่อ PIO ได้รับสัญญาณคู่นี้ก็จะคอยอ่านข้อมูลบนบัสข้อมูลว่าเป็นออพโคดของคำสั่ง RETI หรือไม่ถ้าใช่ก็จะไปทำให้ขา IEO มีสถานะเป็น "1" ซึ่งจะกล่าวถึงการใช้งานในภายหลังและในกรณีที่ทั้งขา $\overline{M1}$ และ $\overline{IORQ}$ ได้รับลอจิก "0" ทั้งคู่ก็แสดงว่า Z80 ส่งสัญญาณตอบสนองการอินเทอร์รัพท์ออกมา และเนื่องจาก PIO ไม่มีขาที่ใช้ในการรีเซ็ทโดยตรง ดังนั้นจึงได้มีการกำหนดไว้ว่า ถ้าขา M1 แอคทีฟเพียงขาเดียว ($\overline{IORQ}$ และ $\overline{RD}$ ไม่แอคทีฟตาม) PIO จะถือว่าเป็นการรีเซ็ท

1.6) $\overline{IORQ}$: สำหรับขานี้จะต่อโดยตรงกับขา $\overline{IORQ}$ ของ Z80 และใช้ร่วมกับ B/$\overline{A}$ SELECT, C/$\overline{D}$ SELECT และ $\overline{CE}$ เพื่อการแลกเปลี่ยนข้อมูลคือเมื่อ $\overline{IORQ}$ กับ $\overline{CE}$ ได้รับลอจิก "0" ทั้งคู่ (ACTIVE) ก็แสดงว่า Z80 ต้องการที่จะติดต่อกับ PIO ตัวนั้นนั่นเอง และเมื่อ $\overline{IORQ}$ กับ $\overline{M1}$ ได้รับลอจิก "0" (ACTIVE) ทั้งคู่ ก็แสดงว่า Z80 ได้ตอบรับการขออินเทอร์รัพท์มา ซึ่งถ้า PIO ถูกโปรแกรมไว้ให้ตอบสนองต่อการอินเทอร์รัพท์ PIO ก็จะส่งอินเทอร์รัพท์เวคเตอร์ไปบนบัสข้อมูลโดยอัตโนมัติ

1.7) $\overline{RD}$: ขา $\overline{RD}$ นี้ก็จะต่อโดยตรงกับขา $\overline{RD}$ ของ Z80 เช่นเดียวกันทราบมาแล้วว่าถ้าสัญญาณจากขา $\overline{RD}$ ของ Z80 นี้เป็น "0" ก็แสดงว่า Z80 ต้องการที่จะอ่านข้อมูลจากหน่วยความจำหรืออุปกรณ์ I/O ดังนั้นในกรณีที่ทั้งขา $\overline{IORQ}$ และ $\overline{CE}$ เป็น "0" ด้วยก็แสดงว่า Z80 ต้องการที่จะอ่านข้อมูลจาก PIO และเนื่องจากบน PIO ไม่มีขา $\overline{WR}$ ดังนั้น PIO จึงใช้หลักที่ว่าขา $\overline{RD}$ และ $\overline{WR}$ จะไม่แอคทีฟพร้อมกัน (ไม่เป็น "0" พร้อมกัน) คือถ้าขา $\overline{RD}$ เป็น "0" ขา $\overline{WR}$ จะเป็น "1" (ซึ่งจะเป็นช่วงเวลาที่ Z80 ทำการอ่านข้อมูลจากภายนอก) และ

ถ้าขา $\overline{RD}$ เป็น "1" ขา $\overline{WR}$ จะเป็น "0" (ซึ่งจะเป็นช่วงเวลาที่ Z80 ทำการส่งข้อมูลออก) ดังนั้น PIO จึงใช้วิธีตรวจสอบว่าถ้า $\overline{IORQ}$ และ $\overline{CE}$ แอคทีพ แต่ $\overline{RD}$ ไม่แอคทีพตาม ก็จะแสดงว่า Z80 ต้องการที่จะส่งข้อมูลให้กับ PIO นั่นเอง

1.8) 0 ; (CLOCK) : ขาคล็อกของ PIO จะต่อเข้ากับคล็อกของระบบซึ่งใช้สำหรับ Z80 ด้วย)

2. กลุ่มที่เกี่ยวกับการอินเทอร์รัพท์ สำหรับขาในกลุ่มนี้จะทำหน้าที่ในการควบคุมการขออินเทอร์รัพท์ของ PIO ซึ่งประกอบด้วย

2.1) $\overline{INT}$: สำหรับขา $\overline{INT}$ นี้เป็นขาเอาท์พุทของ PIO ซึ่งจะแอคทีพเมื่อ PIO ต้องการที่จะขออินเทอร์รัพท์ โดยสามารถที่จะควบคุมได้โดยวิธีการทางซอฟท์แวร์ และยังขึ้นอยู่กับขา IEI ของ PIO ด้วย

2.2) IEI ; (INTERRUPT ENABLE IN) และ IEO ; (INTERRUPT ENABLE OUT) : สำหรับขาทั้ง 2 นี้ จะใช้สำหรับการทำ DAISY CHAIN PRIORITY

3. กลุ่มที่เกี่ยวกับการติดต่อกับอุปกรณ์ภายนอก ขาในกลุ่มนี้จะทำหน้าที่ในการติดต่อ,รับหรือส่งข้อมูลให้กับอุปกรณ์ภายนอก ซึ่งสามารถที่จะแยกได้เป็น 2 พอร์ท คือ พอร์ท A และพอร์ท B

3.1) พอร์ท A ประกอบด้วย บัสข้อมูล 8 เส้น และบัสควบคุมอีก 2 เส้น คือ

3.1.1) A7 - A0 : ทำหน้าที่ในการรับหรือส่งข้อมูลให้กับอุปกรณ์ภายนอก โดยที่ A0 จะเป็นบิทที่มีนัยสำคัญต่ำ (LEAST-SIGNIFICANT BIT) ของข้อมูล

3.1.2) $\overline{ASTB}$ และ ARDY (PORT A STRODBE PULSE INPUT และ REGISTER A READY) เป็นขาที่ใช้สำหรับการทำงานในโหมดที่มีการทำ HANDSHAKE

3.2) พอร์ท B ประกอบด้วย บัสข้อมูล 8 เส้น และบัสควบคุม 2 เส้น เช่นกันคือ

3.2.1) B7 - B0 ทำหน้าที่รับหรือส่งข้อมูลของพอร์ท B กับอุปกรณ์ภายนอก

3.2.2) $\overline{BSTB}$; (PORT B STROBE PULSE INPUT) และ BRDY ; (REGISTER B READY) : ใช้สำหรับการทำงานในโหมดที่มีการทำ HANDSHAKE เช่นเดียวกับ $\overline{ASTB}$ และ ARDY

2.5.3 การต่อ Z80 - PIO กับ Z80.

ในหัวข้อนี้จะศึกษาถึงวิธีการที่จะต่อ Z80 - PIO กับ Z80 รูปที่ 2.5.3 จะแสดงวิธีหนึ่งในการที่จะต่อ PIO เข้ากับ Z80 โดยที่ขา B/A SELECT ของ Z80 - PIO จะต่อเข้าโดยตรงกับ A0 ของ Z80 ส่วนขา C/$\overline{D}$ SELECT จะต่อเข้ากับขา A1 ของ Z80 โดยวิธีนี้

จะเกิดกรณีต่าง ๆ ดังต่อไปนี้คือ

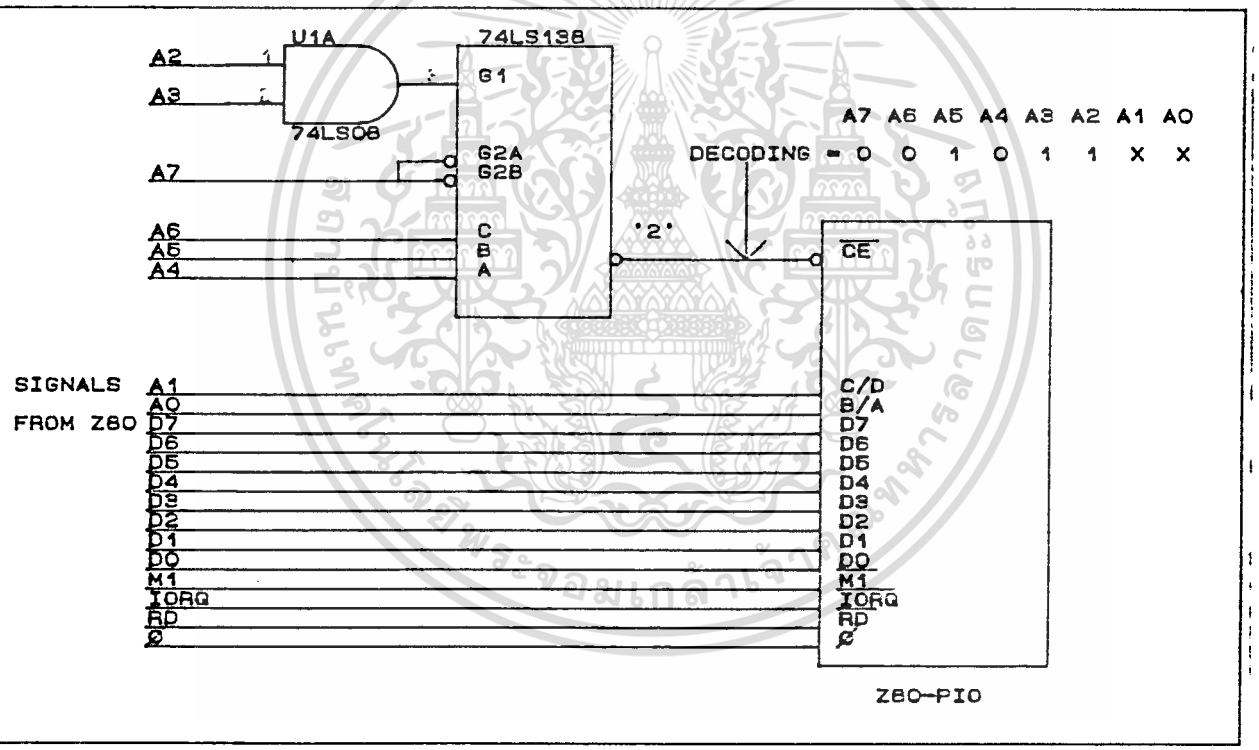
กรณีที่ A1 และ A0 เป็น "0" ทั้งคู่ จะเป็นการเลือก พอร์ท A DATA (ติดต่อรับส่งข้อมูลกับพอร์ท A)

กรณีที่ A1 เป็น "0" และ A0 เป็น "1" จะเป็นการเลือก พอร์ท B DATA (ติดต่อรับส่งข้อมูลกับพอร์ท B)

กรณีที่ A1 เป็น "1" และ A0 เป็น "0" จะเป็นการเลือกพอร์ท A CONTROL (สั่งคำสั่งควบคุมให้กับพอร์ท A)

กรณีที่ A1 เป็น "0" และ A0 เป็น "1" จะเป็นการเลือกพอร์ท B CONTROL (สั่งคำสั่งควบคุมให้กับพอร์ท B)

อย่างไรก็ตามการที่จะติดต่อกับพอร์ททั้ง 4 ข้างต้นนั้น จะต้องทำการ ENABLE PIO โดยการทำให้ขา $\overline{CE}$ ของ PIO ได้รับลอจิก "0"



รูปที่ 2.5.3 การต่อ PIO กับ Z80

จากรูปที่ 2.5.3 ขา $\overline{CE}$ ของ PIO นั้นจะต่อเข้ากับขาเอาท์พุทของตัวถอดรหัส

(DECODER) 74LS138 จะถูก ENABLE (ขา $\overline{CE}$ ได้รับลอจิก "0") ทุกครั้งที่ Z80 ส่งแอดเดรสที่ A7 - A2 เป็น 001011B ดังนั้นเมื่อนำเอา A7 - A2 ที่ได้ทำการถอดรหัสไว้นี้ไปประกอบกับ A1 - A0 ที่ทำการต่อกับ C/$\overline{D}$ และ A/$\overline{B}$ SELECT ในตอนต้นจะได้พอร์ทแอดเดรสดังต่อไปนี้คือ

พอร์ทแอดเดรส(ฐาน 16)	พอร์ทที่ถูกเลือก
2C	A DATA
2D	B DATA
2E	A CONTROL
2F	B CONTROL

2.5.4 การรีเซ็ทบน Z80 - PIO.

เมื่อเริ่มต้นจ่ายไฟให้กับระบบ PIO ก็จะเข้าสู่สภาวะการรีเซ็ทโดยอัตโนมัติดังนี้.

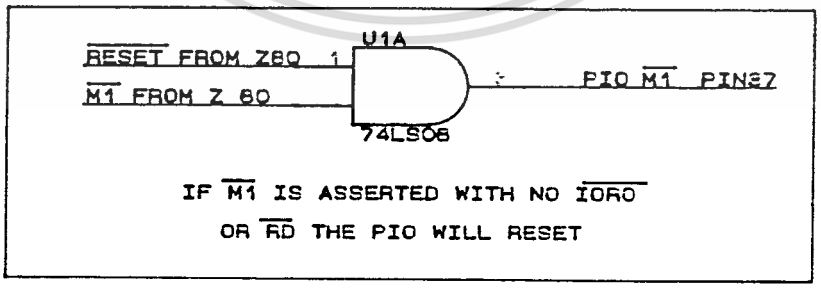
1. PORT MASK REGISTER ของทั้ง 2 พอร์ทจะถูกรีเซ็ท

2. PIO จะทำให้บัสข้อมูลของทั้ง 2 พอร์ทอยู่ในสภาวะ HIGH IMPEDANCE สัญญาณที่ขา READ ของทั้ง 2 พอร์ทจะไม่แอคทีฟ และ PIO จะเลือกการทำงานในโหมด 1 ให้กับทั้ง 2 พอร์ทโดยอัตโนมัติ

3. INTERRUPT VECTOR ADDRESS REGISTER จะไม่ถูกรีเซ็ทตาม สำหรับรีจิสเตอร์ตัวนี้จะส่งอินเทอร์รัพท์เวคเตอร์ออกไปบนบัสข้อมูล เมื่อ PIO ได้รับโปรแกรมตอบสนองการอินเทอร์รัพท์จาก Z80

4. PORT INTERRUPT ENABLE FLIP-FLOPS ของทั้ง 2 พอร์ทจะถูกรีเซ็ทซึ่งจะเป็นการทำให้ Z80 ไม่ทำการตอบสนองต่อการอินเทอร์รัพท์ จนกว่า PIO จะถูกโปรแกรมให้ตอบสนองต่อการอินเทอร์รัพท์

5. PORT OUTPUT REGISTER ของทั้ง 2 พอร์ทจะถูกรีเซ็ท



รูปที่ 2.5.4 : การต่อขา M1 ของ Z80-PIO เข้ากับสัญญาณ RESET และ M1 จาก Z80 CPU.

ดังที่ได้กล่าวไว้ในตอนต้นแล้วว่า นอกจาก PIO จะถูกรีเซ็ทในขณะที่เริ่มทำงาน (

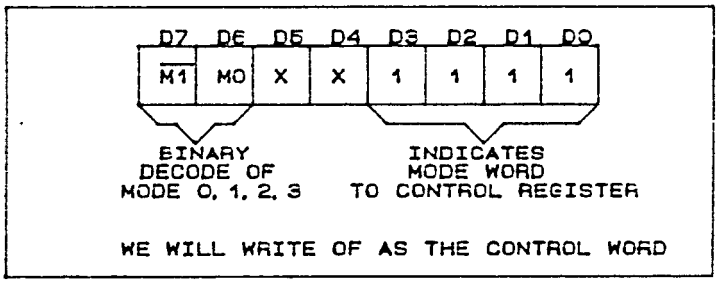
เมื่อเริ่มจ่ายไฟให้กับ PIO)แล้ว อีกวิธีหนึ่งที่จะทำการรีเซ็ท PIO ได้ก็ คือ การทำให้ขา $\overline{M1}$ ของ PIO แอคทีฟ (เป็นลอจิก "0") เพียงขอเดียว ดังนั้นแทนที่จะนำเอาขา M1 ของ PIO ต่อเข้ากับขา $\overline{M1}$ ของ Z80 โดยตรง ก็นำเอาขา $\overline{M1}$ กับขา RESET ของ Z80 มา AND กัน จากนั้นจึงนำเอาเอาท์พุทของ AND GATE ไปต่อกับขา $\overline{M1}$ ของ PIO ดังรูปที่ 2.5.4 แทน เมื่อมีการรีเซ็ท Z80 สัญญาณ $\overline{RESET}$ (ลอจิก "0") ก็จะมาเข้าที่อินพุทของ AND GATE (โดยที่ขา $\overline{M1}$ หรือ $\overline{IORQ}$ จะไม่แอคทีฟ) ทำให้ขา $\overline{M1}$ ได้รับลอจิก "0" (แอคทีฟ) เพียงขาเดียวจะทำให้ PIO ถูกรีเซ็ทตามไปด้วย และนอกจากในกรณีนี้แล้วขา $\overline{M1}$ จะไม่มีทางที่จะแอคทีฟเพียงขาเดียวได้เลย

2.5.5 การโปรแกรมและการทำงานของ Z80 ในโหมด 0

ลักษณะการทำงานของ PIO ในโหมดนี้ก็คือ การทำให้พอร์ทที่ถูกโปรแกรมทำหน้าที่เป็นพอร์ทเอาท์พุท คือ ใช้ส่งข้อมูลออกจากพอร์ทได้เพียงอย่างเดียว สามารถที่จะแยกโปรแกรมแต่ละพอร์ทได้โดยอิสระ กล่าวคืออาจจะเลือกโปรแกรมพอร์ท A ให้อยู่ในโหมด 1 และพอร์ท B ให้อยู่ในโหมด 0 แต่เนื่องจากการโปรแกรมในพอร์ททั้งสองมีความคล้ายคลึงกันมาก จึงจะอธิบายถึงวิธีการโปรแกรมในพอร์ท B เท่านั้น

ในขั้นแรกจะสมมติว่า PIO ถูกรีเซ็ท รีจิสเตอร์ตัวแรกที่จะต้องโปรแกรมก็คือ รีจิสเตอร์ควบคุม (CONTROL REGISTER) เพื่อใช้ในการเลือกโหมดการทำงาน ในกรณีนี้จะใช้วงจรดังในรูปที่ 2.5.3 ดังนั้นเพื่อที่จะเลือกโปรแกรมรีจิสเตอร์ควบคุมของพอร์ท B จะต้องเลือกโปรแกรมที่พอร์ทแอดเดรส 2FH ซึ่งเป็นแอดเดรสของพอร์ท B CONTROL (สำหรับพอร์ท A CONTROL คือแอดเดรส 2 EH) สำหรับวิธีการที่จะโปรแกรมนั้นจะทำได้โดยการส่งข้อมูลให้กับรีจิสเตอร์ควบคุมของพอร์ทนั้น ๆ (ในกรณีนี้คือพอร์ท B ; 2FH) และเรียกข้อมูลไปที่นี้ว่า "คำสั่งควบคุม" (CONTROL WORD) สำหรับลักษณะการจัดเรียงบิทในคำสั่งควบคุมนั้นได้แสดงไว้ในรูปที่ 2.5.5

จากรูปจะเห็นว่า 4 บิทล่าง(D3-D0) จะเป็น "1" หมดทุกบิทเพื่อทำให้ PIO ทราบว่าข้อมูลที่ส่งมานี้ใช้ในการเลือกโหมดทำงานของพอร์ท ส่วนบิท D7



รูปที่ 2.5.5 การจัดเรียงบิทในคำสั่งควบคุม

และ D6 นั้นจะใช้ในการแสดงว่าต้องการที่จะโปรแกรมพอร์ทที่เลือกไว้(ในที่นี้คือพอร์ทB) ให้อยู่ในโหมดใดโดยที่

D7	D6	โหมดการทำงาน
0	0	0
0	0	1
1	0	2 (BI-DIRECTIONAL)
1	1	3 (CONTROL MODE)

ส่วนบิท D4 และ D5 นั้นจะไม่มีผลต่อการโปรแกรมเลือกโหมด ดังนั้นจะให้บิททั้งสองมีค่าใดก็ได้ สำหรับในกรณีนี้จะให้เป็น "0" เหมือนกันทั้งคู่ ดังนั้นคำสั่งที่ใช้ในการเลือกโหมดจึงเป็น OFH สำหรับโหมด 0, 4FH สำหรับโหมด 1, 8FH สำหรับโหมด BI-DIRECTION หรือ CFH สำหรับโหมด 3 (CONTROL MODE) ในกรณีนี้ต้องการที่จะเลือกโหมด 0 สำหรับพอร์ท B ดังนั้นจึงต้องส่งข้อมูล OFH ไปยังพอร์ท 2FH เมื่อโปรแกรมพอร์ทนี้แล้วก็สามารถที่จะส่งข้อมูลผ่านพอร์ท B ไปยังอุปกรณ์ภายนอกได้โดยผ่านทางพอร์ทแอดเดรส 2DH (A0=1,A1=0 ; PORT B DATA) สำหรับตัวอย่างโปรแกรมของทั้ง 2 พอร์ทแสดงดังรูปที่ 2.5.6

ดังที่ได้กล่าวในตอนต้นแล้วว่าการทำ HANDSHAKE ของ Z80 นั้นจะต้องใช้งานขาARDY กับ $\overline{\text{ASTB}}$ (สำหรับ PORT A) หรือ BRDY กับ $\overline{\text{BSTB}}$ (สำหรับพอร์ท B) ซึ่งในโหมด 0 นี้การใช้งานขา RDY กับ STR (ของทั้ง 2 พอร์ท) จะเป็นดังนี้คือ

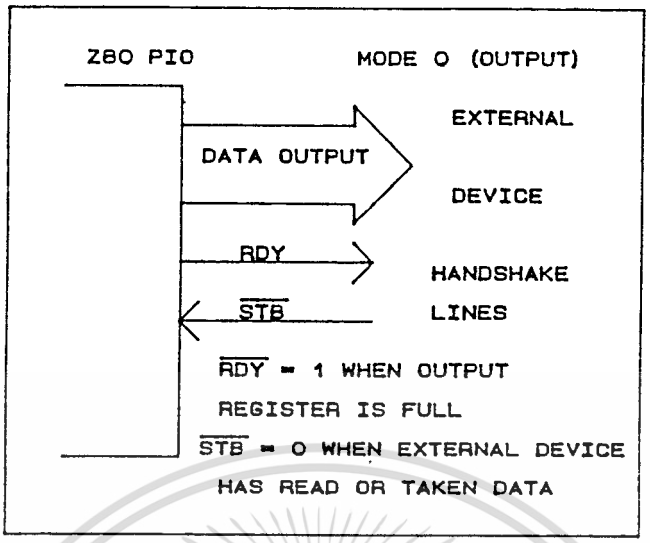
```
;
; โปรแกรมพอร์ท B เป็น เอาท์พุท , B/A = 1 , C/D = 1
; บิท D7 = 0 , D6 = 0
;
1500    3E0F        LD A,0FH
1502    D32F        OUT (2FH),A    ; ส่งคำสั่งควบคุมให้กับ
                                   ; พอร์ท B
;
; ส่งข้อมูล 54H ให้กับอุปกรณ์ภายนอกผ่านทางพอร์ท B
; B/A = 1 , C/D = 0
;
```

```
1504    3E54        LD A,54H
1506    D32D        OUT (2DH),A    ; ส่งข้อมูลผ่านทางพอร์ท B
;
; โปรแกรมพอร์ท A เป็นเอาท์พุท , B/A = 0 , C/D = 1
;
1508    3E0F        LD A,0FH
150A    D32C        OUT (2CH),A    ; ส่งคำสั่งควบคุมให้กับ
                                   ; พอร์ท A
;
; ส่งข้อมูล 90H ให้กับอุปกรณ์ภายนอกผ่านทางพอร์ท A
; B/A = 0 , C/D = 0
;
150C    3E90        LD A,90H
150E    D32C        OUT (2CH),A    ; ส่งข้อมูลผ่านพอร์ท A
;
```

รูปที่ 2.5.6 ตัวอย่างโปรแกรมที่ใช้ในการเลือกโหมดการทำงานในโหมด 0

เมื่อ Z80 ทำการส่งข้อมูลให้กับ PIO เรียบร้อยแล้ว ขา RDY ของ PIO จะมีลอจิกเป็น"1" และเมื่ออุปกรณ์ภายนอกได้รับสัญญาณนี้ ก็จะทำการอ่านข้อมูลจากพอร์ทเอาท์พุท (ที่ถูกโปรแกรมไว้ ในที่นี้คือทั้ง PORT A และ PORT B) และเมื่ออุปกรณ์ภายนอกอ่านข้อมูลจากพอร์ทเอาท์พุทแล้ว อุปกรณ์ภายนอกก็จะส่งสัญญาณ $\overline{STB}$ ให้กับ PIO เมื่อ PIO ได้รับสัญญาณ $\overline{STB}$ แล้วก็จะทำให้ขา $\overline{RDY}$ เป็น "0" จากนั้น PIO ก็จะส่ง $\overline{INT}$ ให้กับ Z80 เพื่อทำการขออินเทอร์รัพท์ ซึ่งจะทำให้ Z80 ทราบว่าอุปกรณ์จะส่งให้กับอุปกรณ์ภายนอกอีก Z80 ก็จะทำการส่งข้อมูลออกไปอีกโดยผ่านทาง PIO บล็อกไดอะแกรมในรูป 2.5.7 จะแสดงถึงการใช้บัสข้อมูล RDY และ $\overline{STB}$ กับอุปกรณ์ภายนอก

สำหรับในโหมด 0 นี้สามารถที่จะทำให้ PIO ส่งข้อมูลออกทางพอร์ทเอาท์พุท โดยไม่ทำ HANDSHAKE ได้โดยการทำให้ระดับลอจิกที่ขา $\overline{STB}$ เป็น "1" ซึ่งอาจจะทำได้โดยการนำเอาขา $\overline{STB}$ ของ PIO ต่อเข้ากับไฟเลี้ยงของระบบ (+5 V) เมื่อทำเช่นนี้แล้ว PIO ก็จะทำการส่งข้อมูลออกไปบนบัสข้อมูลของพอร์ทเอาท์พุท เพื่อส่งข้อมูลให้กับ

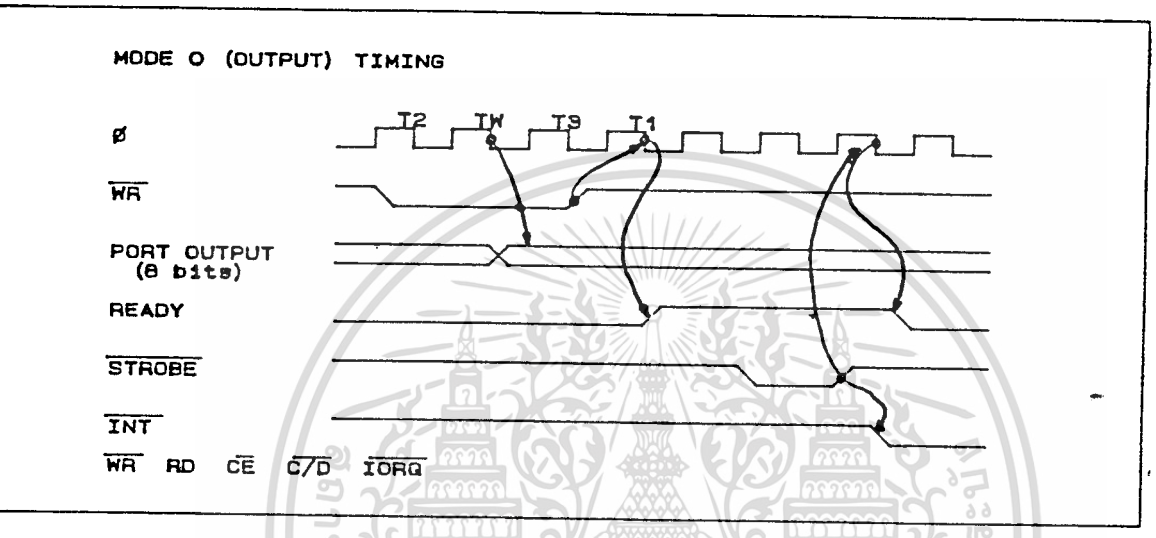


รูปที่ 2.5.7 บล็อกไดอะแกรมแสดงการใช้ขา RDY
และ STB กับอุปกรณ์ภายนอก

อุปกรณ์ภายนอกโดยไม่สนใจว่าอุปกรณ์ภายนอกจะทำการรับข้อมูลออกไปหรือยัง และ PIO จะไม่ทำการขออินเทอร์รัพท์จาก Z80 ด้วยสำหรับไดอะแกรมเวลา (TIMING DIAGRAM) การทำงานของ PIO ในโหมด 0 ได้แสดงไว้ดังรูปที่ 2.5.8

2.5.6 การโปรแกรมและการทำงานของ Z80-PIO ในโหมด 1.

เมื่อ PIO ถูกโปรแกรมให้ทำงานในโหมดนี้ PIO จะทำงานในลักษณะของพอร์ทอินพุท คือรับข้อมูลจากอุปกรณ์ภายนอกได้เพียงอย่างเดียว สำหรับคำสั่งควบคุมที่จะใช้ในการโปรแกรม PIO ให้ทำงานในโหมดนี้ก็คือ 4FH (D7=0,D6=1)โดยส่งไปยังพอร์ทแอดเดรส2EH หรือ 2FH (พอร์ท A หรือ B CONTROL) สำหรับในที่นี้จะให้พอร์ท B เป็นพอร์ทอินพุท ดังนั้นจะต้องส่งข้อมูล 4FH ไปยังพอร์ท 2FH ไปยังพอร์ท 2FH ในโหมดนี้ข้อมูลจากอุปกรณ์ภายนอกจะส่งให้กับ PIO ที่ขา B7-B0 ของ PIO (A7=A0 ในกรณีที่ใช้พอร์ท A เป็นพอร์ทอินพุท) ดังแสดงในรูปที่ 2.5.9

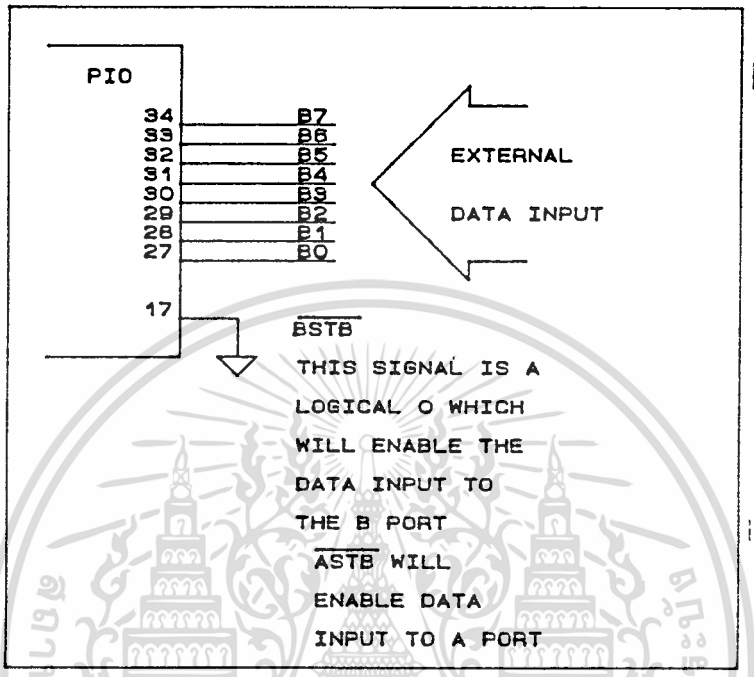


รูปที่ 2.5.8 ไดอะแกรมเวลาแสดงการทำงานของ PIO ในโหมด 0

ในกรณีที่ไม่ต้องการที่จะทำ HANDSHAKE ในโหมดนี้ ก็เพียงแต่ทำให้ขา $\overline{\text{BSTB}}$ ได้รับลอจิก "0" ซึ่งอาจจะทำได้โดยการต่อขา $\overline{\text{BSTB}}$ กับ GROUND ทำเช่นนี้จะทำให้ PIO รับข้อมูลที่อยู่บน B7-B0 (หรือ A7-A0) โดยไม่ต้องรอ $\overline{\text{BSTB}}$ จากอุปกรณ์ภายนอก รูปที่ 2.5.10 จะแสดงวิธีการโปรแกรม PIO ให้พอร์ท A เป็นพอร์ทเอาท์พุท และพอร์ท B เป็นพอร์ทอินพุท ซึ่งโปรแกรมนี้จะส่งข้อมูลให้กับพอร์ท A และรับข้อมูลเข้าทางพอร์ท B

สำหรับกรณีที่ต้องการที่จะทำ HANDSHAKE นั้น จะต้องใช้งานขา $\overline{\text{ASTB}}$ กับ ARDY (สำหรับพอร์ท A) และ $\overline{\text{BSTB}}$ กับ BRDY (สำหรับพอร์ท B) ในที่นี้จะใช้ขา $\overline{\text{BSTB}}$ กับ BRDY โดยเมื่อขา $\overline{\text{BSTB}}$ แอคทีฟ (ได้รับลอจิก "0" PIO ก็จำทำการอ่านข้อมูลจากบัสข้อมูลของพอร์ท

ที่ทำหน้าที่เป็นพอร์ทอินพุท ซึ่งในที่นี้ก็คือ B7-B0 (หรือ A7-A0 สำหรับพอร์ท A) นั่นเอง ส่วนขา BRDY นั้นจะ



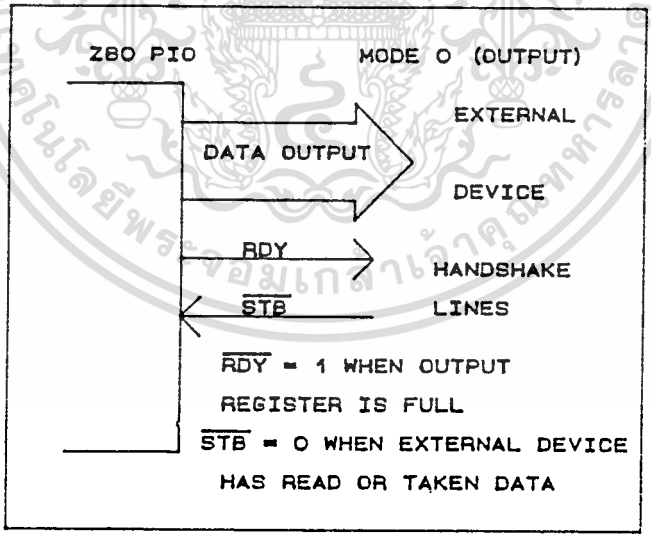
รูปที่ 2.5.9 การรับข้อมูลจากอุปกรณ์ภายนอกของ PIO.

```
;
; โปรแกรมพอร์ท A ให้เป็นพอร์ทเอาพุท
;
1600    3E0F        LD A,0FH
1602    D32E        OUT (2EH),A    ; ส่งคำสั่งควบคุมให้กับ
                                   ; พอร์ท B
;
; โปรแกรมพอร์ท B ให้เป็นพอร์ทอินพุท
;
1604    3E4F        LD A,4FH
1606    D32F        OUT (2FH),A    ;ส่งคำสั่งควบคุมให้กับ
```

```
                                        ; พอร์ท B
;
; การส่งข้อมูลออกและรับข้อมูลเข้าจากอุปกรณ์ภายนอกโดยผ่านทาง
; พอร์ท A และพอร์ท B ของ PIO
;
1608    DB2D        IN A,(2DH)      ; การอ่านข้อมูลจากพอร์ท B
160A    D32C        OUT (2CH)       ; การส่งข้อมูลให้กับพอร์ท A
```

รูปที่ 2.5.10 ตัวอย่างโปรแกรมใช้ในการเลือกการทำงานของ PIO ในโหมด 0 และ1

เป็นเอาท์พุทที่ทำหน้าที่ในการแสดงให้อุปกรณ์ภายนอกทราบว่า INPUT REGISTER นั้นพร้อมที่จะรับข้อมูลต่อไปได้ (CPU ได้ทำการอ่านข้อมูลใน INPUT REGISTER แล้ว)รูปที่ 2.5.11 จะแสดงบล็อกไดอะแกรมการต่อบัสข้อมูลม ,$\overline{BSTB}$ และ BRDY กับอุปกรณ์ภายนอก

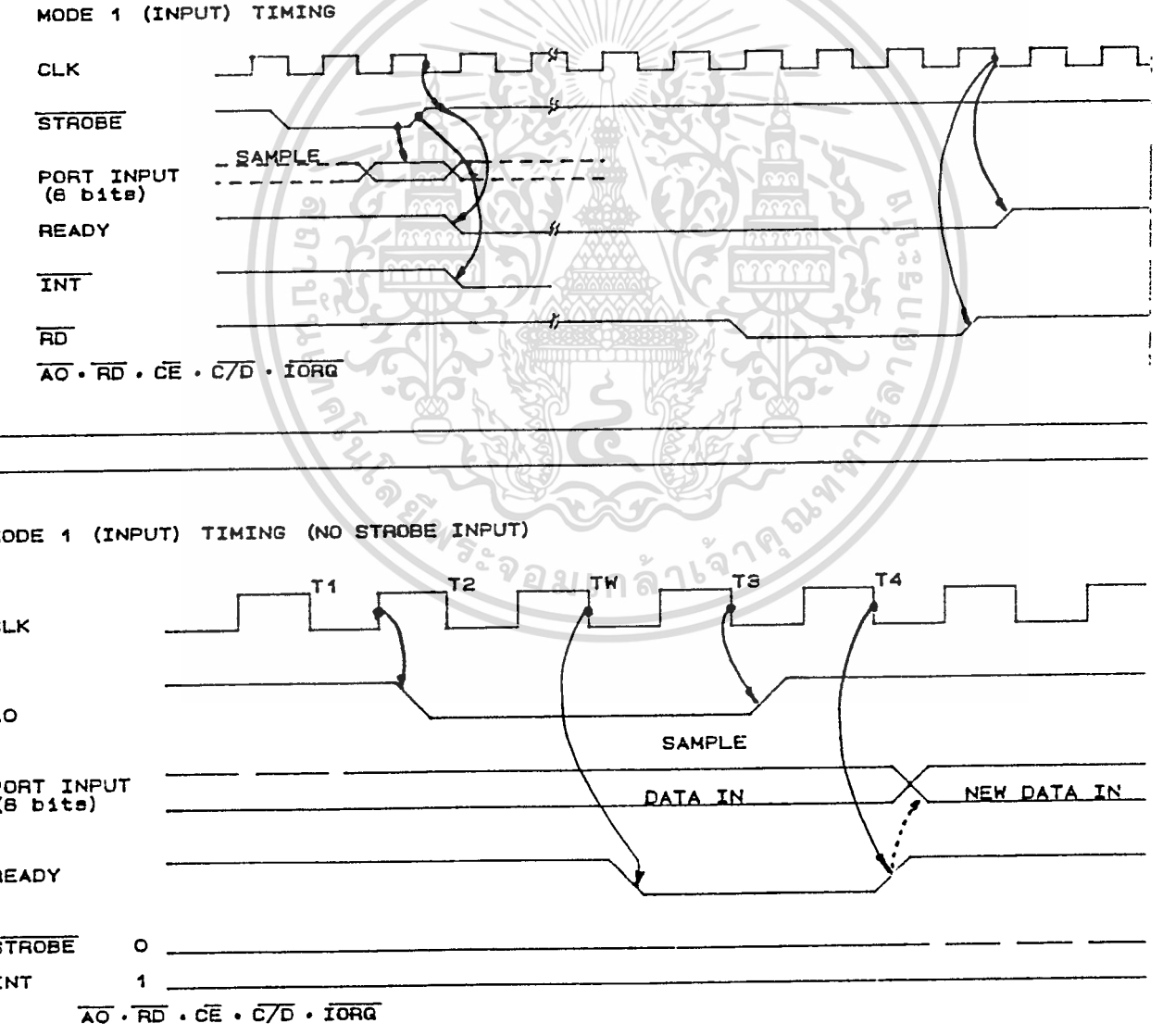


รูปที่ 2.5.11 บล็อกไดอะแกรมการต่อ PIO กับ อุปกรณ์ภายนอก.

เมื่ออุปกรณ์ภายนอกส่งข้อมูลให้กับ PIO แล้ว PIO จะทำการขออินเทอร์รัพท์ (ในกรณีที่

PIO ทำงานในลักษณะที่ไม่ทำ HANDSHAKE นั้น PIO จะไม่ทำการขออินเทอร์รัพท์จาก Z80) โดยทำให้ขา $\overline{INT}$ แอคทีฟ (แอคทีฟที่ลอจิก "0") จากนั้นก็จะเป็นหน้าที่ของโปรแกรมตอบสนองการอินเทอร์รัพท์ที่จะทำให้ Z80 อ่านข้อมูลจากพอร์ทอินพุทของ PIO (ในกรณีนี้คือ พอร์ท B) เมื่อ Z80 อ่านข้อมูลจากพอร์ทอินพุทแล้ว PIO ก็จะทำให้ขา BRDY ACTIVE (ลอจิก "1") เพื่อที่จำทำให้อุปกรณ์ภายนอกทราบว่า PIO พร้อมที่จะรับข้อมูลชุดใหม่แล้ว (อุปกรณ์ภายนอกจะไม่ทำการส่งข้อมูลให้กับ PIO จนกว่าขา BRDY ของ PIO จะมีระดับลอจิกเป็น "1")

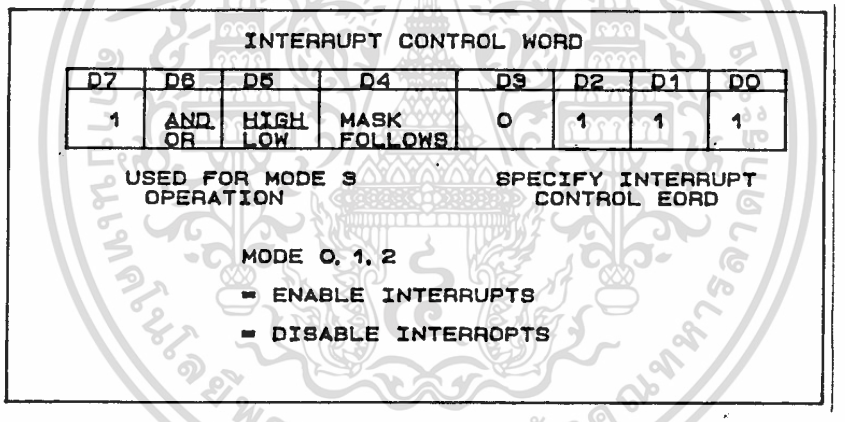
สิ่งหนึ่งที่จะต้องคำนึงถึงคือ โปรแกรมเลือกการทำงานในโหมดนี้แล้ว จะต้องทำการอ่านข้อมูลจาก PIO ในทันที เนื่องจากเมื่อ PIO ถูกรีเซ็ทนั้นขา BRDY จะถูกทำให้เป็น "0" ดังนั้นการอ่านข้อมูลจาก PIO ก็เพื่อที่จะทำให้ขา BRAY มีระดับลอจิกเป็น "1" นั่นเอง อย่างไรก็ตามข้อมูลที่อ่านเข้ามานี้เป็นข้อมูลที่ไม่สนใจว่าจะมีค่าเป็นอะไร เนื่องจากทำการอ่านข้อมูลนี้เพื่อทำให้ขา BRDY มีลอจิกเป็น "1" เท่านั้น



รูปที่ 2.5.12 ไดอะแกรมเวลาแสดงการทำงานของ PIO ในโหมด 1.

2.5.7 การโปรแกรมคำสั่งควบคุมการอินเทอร์รัพท์ (INTERRUPT CONTROL WORD)

สำหรับหัวข้อที่ผ่านมานั้น ได้ศึกษาถึงวิธีการที่จะโปรแกรม PIO ให้เลือกการทำงานในโหมด 0 หรือ โหมด 1 ให้กับพอร์ท A หรือ พอร์ท B รวมทั้งการอินเทอร์รัพท์ในทั้ง 2 โหมดอีกด้วย ในหัวข้อนี้จะศึกษาถึงวิธีที่จะโปรแกรมการอินเทอร์รัพท์ และวิธีการที่จะใช้กับการทำงานของ PIO ในโหมด 0 และ โหมด 1 เช่นเดียวกับการล็อกโหมด ขั้นแรกจะต้องทำการส่งข้อมูลให้กับรีจิสเตอร์ควบคุม (CONTROL REGISTER) เพื่อโปรแกรมทำงานให้กับ PIO เสียก่อน แต่ข้อมูลที่ส่งไปนี้ จะมีบิท D3–D0 เป็น 011B แทนที่จะเป็น 1111B สำหรับบิท D7-D4 นั้นจะถูกใช้ในการโปรแกรมการทำงานขอ PIO ในการอินเทอร์รัพท์และเรียกข้อมูลที่ถูกส่งให้กับรีจิสเตอร์ควบคุมในลักษณะนี้ว่า "INTERRUPT CONTROL WORD" รูปที่ 7.13 จะแสดงการจัดหน้าที่ของแต่ละบิทบน INTERRUPT CONTROL WORD



รูปที่ 2.5.13 การจัดเรียงบิทบน INTERRUPT CONTROL WORD.

ถ้าบิท D7 ของ INTERRUPT CONTROL WORD เป็น "1" จะทำให้ PIO พร้อมที่จะทำการอินเทอร์รัพท์ แต่ถ้าบิท D7 นี้เป็น "0" PIO จะไม่สามารถที่จะทำการขออินเทอร์รัพท์ได้

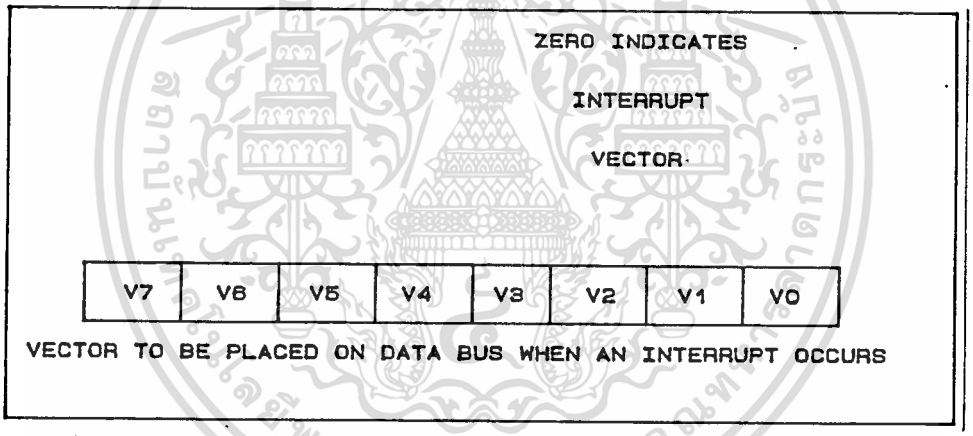
สำหรับบิท D6-D4 จะใช้เฉพาะการทำงานในโหมด 3 สำหรับในโหมด 0 และ โหมด 1 ของ PIO นั้น จะสนใจแต่เพียงการ ENABLE หรือ DISABLE การขออินเทอร์รัพท์เท่านั้น ดังนั้นนกรณีของโหมด 0 และ โหมด 1 ค่าของ บิท D6-D4 จะไม่มีผลต่อการขออินเทอร์รัพท์ของ PIO แต่อย่างใด ในกรณีนี้จะให้บิท D6-D4 เป็น "0" หมด ดังนั้นจึงมีข้อมูลเพียง 2 ค่าที่จะเลือกส่งให้กับรีจิสเตอร์ควบคุมคือ

87H สำหรับ ENABLE INTERRUPT

07H สำหรับ DISABLE INTERRUPT

ในกรณีที่ยอมให้ PIO ขออินเทอร์รัพท์ได้นั้น (ส่งข้อมูล 87H ให้ CONTROL

REGISTER) สิ่งที่จะต้องทำในขั้นต่อไปนี้ก็คือ กำหนดค่าของ INTERRUPT VECTOR ที่จะต้องส่งให้กับ Z80 ในการอินเทอร์รัพท์โหมด 2 ของ Z80 สำหรับ INTERRUPT VECTOR นี้สามารถที่จะส่งให้กับ PIO ได้ โดยวิธีเดียวกับการเลือกโหมดหรือการโปรแกรม INTERRUPT CONTROL WORD เพียงแต่ว่าในกรณีของ INTERRUPT VECTOR นั้นค่าของบิท D0 จะเป็น "0" (ค่าของ INTERRUPT VECTOR จะต้องเป็นเลขคู่เท่านั้น) สำหรับรูปแบบของบิทต่าง ๆ บน INTERRUPT VECTOR จะแสดงดังรูปที่ 2.5.14 สำหรับรูปที่ 2.5.15 นั้นจะเป็นการโปรแกรมให้พอร์ท A เป็นพอร์ทเอาท์พุทและทำการ ENABLE การขออินเทอร์รัพท์ของพอร์ท A รวมทั้งแสงวิธีการที่จะส่ง INTERRUPT VECTOR ให้กับ PIO ซึ่งในกรณีนี้กำหนดให้ INTERRUPT VECTOR มีค่า 76 H



รูปที่ 2.5.14 การจัดเรียงบิทบน INTERRUPT VECTOR.

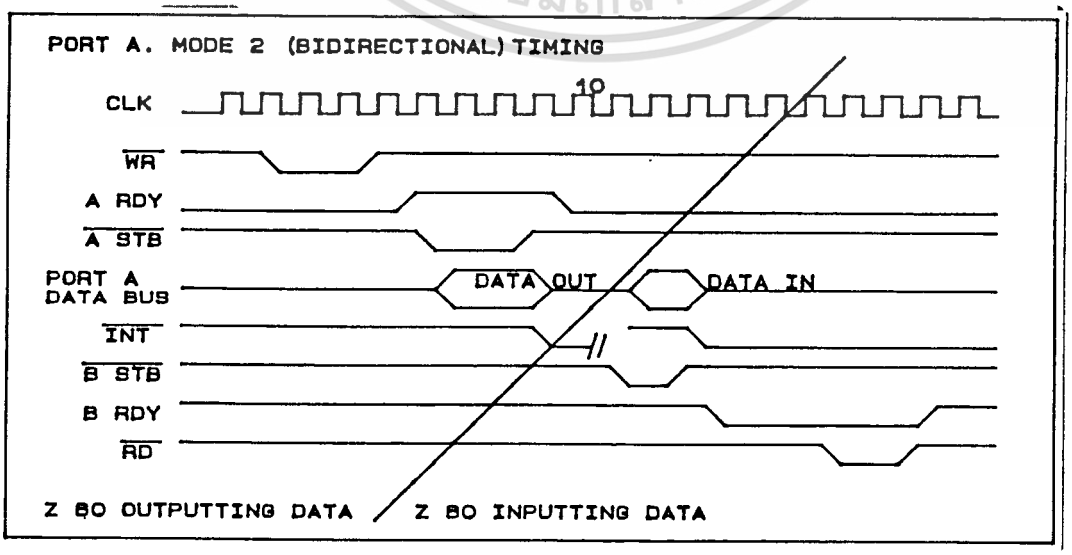
```
;
1700    3E76        LD A,76H        ; INTERRUPT VECTOR
                                    ; = 76H
1702    D32E        OUT (2EH),A     ; ส่งให้กับ PIO
1704    3E0F        LD A,0FH
1706    D32F        OUT (2EH),A     ;โปรแกรมพอร์ท A เป็น
                                    ; พอร์ทเอาท์พุท
; การโปรแกรม INTERRUPT CONTROL WORD สำหรับ พอร์ท A
```

```
;
170B    3E87        LD A,87H
1706    D32E        OUT (2EH),A    ; ENABLE การขอ
                                   ; อินเทอร์รัพท์
```

รูปที่ 2.5.15 ตัวอย่างโปรแกรมที่ใช้ในการ ENABLE INTERRUPT และการส่ง INTERRUPT VECTOR สำหรับ พอร์ท A

2.5.8 การใช้งาน PIO ในโหมด 2 (BI-DIRECTIONAL MODE).

การทำงานของ PIO ในโหมดนี้เป็นการรวมเอาลักษณะการทำงานใน โหมด 0 และ โหมด 1 ไว้ด้วยกัน คือ สามารถที่จะสั่งให้ PIO ทำงานเป็นทั้งพอร์ทอินพุทและพอร์ทเอาท์พุทได้ในพอร์ทเดียวกัน ดังนั้นในการทำ HANDSHAKE จึงจำเป็นที่จะต้องใช้บัสถึง 4 เส้น (2 เส้นสำหรับอินพุทและอีก 2 เส้นสำหรับเอาท์พุท) ด้วยเหตุนี้จึงทำให้สามารถที่จะงาน PIO ในโหมดนี้ได้เพียงพอร์ทเดียวเท่านั้น ซึ่งในโหมดนี้ PIO ได้กำหนดให้ใช้งานได้เฉพาะพอร์ท A เท่านั้น ส่วนพอร์ท B นั้นไม่ว่าจะใช้งานหรือไม่ก็ตามจะต้องถูกโปรแกรมให้อยู่ในโหมด 3 (CONTROL MODE) เท่านั้น และจะต้องทำการส่ง INTERRUPT VECTOR ที่แตกต่างกันจำนวน 2 ไบท์ให้กับ PIO ซึ่ง INTERRUPT VECTOR ทั้ง 2 ไบท์นี้จะใช้บ่งแอดเดรสของโปรแกรมตอบสนองการอินเทอร์รัพท์ที่ใช้ PIO ในโหมด 2 โดยจะทำการส่ง INTERRUPT VECTOR ทั้ง 2 ไบท์นี้ให้กับพอร์ท A (สำหรับกรณีที่เป็นเอาท์พุท) และพอร์ท B (สำหรับกรณีที่เป็นอินพุท) ตามลำดับ



รูปที่ 2.4.16 ไดอะแกรมเวลาสำหรับการรับส่งข้อมูลในโหมด 2

รูปที่ 2.5.16 จะแสดงถึงไดอะแกรมเวลาของการรับส่งข้อมูลของ PIO ในโหมดนี้ สำหรับขาที่ใช้ในการทำ HANDSHAKE ของพอร์ท A ($\overline{ASTB}$,ARDY) นั้น จะถูกใช้สำหรับกรณีที่พอร์ท A ทำหน้าที่เป็นพอร์ทเอาท์พุท และของพอร์ท B ($\overline{BSTB}$,BRDY) สำหรับกรณีที่พอร์ท A ทำหน้าที่เป็นพอร์ทอินพุท

ในขั้นนี้จะทำการศึกษาไดอะแกรมเวลาในรูปที่ 2.5.16 ดังนี้

1. เมื่อ Z80 ทำการส่งข้อมูลให้กับ OUTPUT REGISTER ของพอร์ท A PIO ก็จะทำการส่ง ARDY ออกไปให้กับอุปกรณ์ภายนอก เพื่อแสดงให้อุปกรณ์ภายนอกทราบว่า PIO พร้อมที่จะส่งข้อมูลให้กับอุปกรณ์ภายนอกแล้ว.

2. เมื่ออุปกรณ์ภายนอกได้รับ ARDY จาก PIO แล้วก็จะทำการส่ง $\overline{ASTB}$ กลับมาให้ PIO เมื่อ PIO ได้รับ $\overline{ASTB}$ แล้วก็จะส่งข้อมูลออกมาที่ DATA BUS (A0-A7) และอุปกรณ์ภายนอกก็จะรับข้อมูลออกไป จากนั้นอุปกรณ์ภายนอกก็จะทำให้ $\overline{ASTB}$ เป็นลอจิก "1"

3. หลังจากที่สัญญาณ $\overline{ASTB}$ เปลี่ยนจากลอจิก "0" เป็น "1" (อุปกรณ์ภายนอกได้รับข้อมูลไปแล้ว) PIO ก็จะส่ง $\overline{INT}$ ให้กับ Z80 เพื่อเป็นการแสดงว่า PIO พร้อมที่จะรับข้อมูลชุดใหม่แล้ว

4. สำหรับกรณีที่อุปกรณ์ภายนอกต้องการที่จะส่งข้อมูลให้กับ Z80 โดยผ่านทาง PIO นั้น อุปกรณ์ภายนอกก็จะส่ง $\overline{BSTB}$ ให้กับ PIO และทำการส่งข้อมูลออกมาที่ A0-A7 เพื่อให้ PIO ทำการรับข้อมูลเข้าไปเพื่อส่งให้กับ Z80 และเมื่อ $\overline{BSTB}$ เปลี่ยนลอจิกจาก "0" เป็น "1" (ขอบขาขึ้น) PIO ก็จะส่ง $\overline{INT}$ ให้กับ Z80 เพื่อแสดงให้ Z80 ทราบว่า PIO ได้รับข้อมูลจากอุปกรณ์ภายนอกและพร้อมที่จะส่งข้อมูลให้กับ Z80 แล้ว

5. หลังจากที่ PIO ทำการส่ง $\overline{INT}$ ให้กับ Z80 แล้ว ขา BRDY ของ PIO ก็จะกลายเป็น "0" เพื่อเป็นการบอกให้กับอุปกรณ์ภายนอกทราบว่า Z80 ยังไม่ได้รับข้อมูลจาก PIO และ PIO ยังไม่พร้อมที่จะรับข้อมูล และเมื่อ Z80 อ่านข้อมูลจาก พอร์ท A ของ PIO แล้ว PIO ก็จะส่ง BRDY ออกไปให้กับข้อมูลภายนอกเพื่อแสดงว่าขณะนี้ PIO พร้อมที่จะรับข้อมูลแล้ว

จากไดอะแกรมเวลาข้างต้น จะเห็นว่า PIO ทำการส่ง $\overline{INT}$ ให้กับ Z80 ทุกครั้งที่อุปกรณ์ภายนอกรับหรือส่งข้อมูลให้กับพอร์ท A ดังนั้น จึงจำเป็นที่จะต้องทำให้ Z80 ทราบว่าสัญญาณ $\overline{INT}$ ที่ PIO ส่งไปให้กับ Z80 นั้นเกิดขึ้นจากกรณีใดซึ่งวิธีที่ใช้โดยทั่วไปก็คือทำการส่ง INTERRUPT VECTOR ที่แตกต่างกันให้กับ Z80 (Z80 ทำการอินเทอร์รัพท์ในโหมด 2) สำหรับกรณีของ PIO นั้น สามารถที่จะทำได้โดยการส่ง INTERRUPT VECTOR ที่แตกต่างกันจำนวน 2 ไบท์ ให้กับ Z80

ดังนั้นเมื่ออุปกรณ์ภายนอกส่งข้อมูลให้กับ PIO, INTERRUPT VECTOR ที่อยู่บนพอร์ท B ก็จะถูกส่งออกไปบน DATA BUS ให้กับ Z80 และเมื่อ PIO ส่งข้อมูลให้กับอุปกรณ์ภายนอก INTERRUPT VECTOR ที่อยู่บนพอร์ท A ก็จะถูกส่งให้กับ Z80.

เนื่องจากพอร์ท B ถูกโปรแกรมให้ทำงานในโหมด 2 อาจจะเกิดผิดพลาดเนื่อง

นอก INTERRUPT VECTOR ที่อยู่บนพอร์ท A ก็จะถูกส่งให้กับ Z80.

เนื่องจากพอร์ท B ถูกโปรแกรมให้ทำงานในโหมด 2 อาจจะเกิดผิดพลาดเนื่องจากการทำงานของพอร์ท B ก็ได้ เพื่อป้องกันความผิดพลาดนี้ จึงควรที่จะทำการ DISABLE การขออินเทอร์รัพท์ของพอร์ท B ไว้ก่อน ดังโปรแกรมในรูปที่ 2.5.17 อย่างไรก็ตามการที่จะให้การรับส่งข้อมูลระหว่างอุปกรณ์ภายนอกกับ PIO เกิดขึ้นได้อย่างถูกต้องนั้น อุปกรณ์ภายนอกจะต้องทำการส่งสัญญาณ $\overline{\text{ASTB}}$ และ $\overline{\text{BSTB}}$ ได้ในช่วงเวลาที่ถูกต้องด้วย.

```
;
2700    OUTDAT EQU 2700H      ;MEM LOC FOR OUTPUT DATA
2701    INDAT  EQU OUTDAT+1   ;MEM LOC FOR INPUT DATA
;
;
;
;  THIS PROGRAM WILL SET UP PORT A AS AN I/O PORT
;
1800  3E8F     LD A 8FH       ;THIS IS THE MODE WORD
1802  D32E     OUT (2EH),A    ;WRITE MODE WORD TO PA
1804  3E00     LD A,OOH       ;INTERRUPT VECTOR FOR PA
1806  D32E     OUT (2EH),A    ;WRITE VECTOR TO PA
;
1808  3ECF     LD A,OCFH      ;MODE WORD FOR PORT B
180A  D32F     OUT (2FH),A    ;WRITE VECTOR TO PA
;
;    MODE WORD SET UP PORT B IN MODE 3
;    THE BIT DEFINITIONS FOR PORT B FOLLOW
;
180C  3EFF     LD A,OFFH      ;ALL BITS ARE INPUTS
180E  D32F     OUT (2FH),A    ;WRITE TO PORT B
1810  3E17     LD A,17H       ;PORT B INT CONTROL WORD
1812  D32E     OUT (2EH),A    ;DISABLE INT, MASK FOLLOWS
1814  3EFF     LD A,OFFH      ;ALL BITS DISABLED
1816  D32F     OUT (2FH),A    ;WRITE MASK TO PORT B
;
```

```
1818  3E02      LD A,02H        ;INTERRUPT VECTOR PORT B
181A  D32F      OUT (2FH),A     ;WRITE TO PORT B
;
181C  3E3A      LD A,3AH        ;SET VECTOR TABLE UPPER BYTE
181E  ED47      LD I,A
;
;    VECTOR TABLE IS AT LOCATION 3A00 FOR PORT A
;    AND 3A02 FOR PORT B
;    THE ADDRESS OF THE SERVICE ROUTINES ARE FC81 FOR
;    PORT A, AND FD00 FOR PORT B.
;
1820  ED5E      IM 2            ;SET INTERRUPT MODE 2
1822  3E83      LD A,83H        ;ENABLE INT ON PIO
1824  D32E      OUT (2EH),A     ;ENABLE PORT A
1826  D32F      OUT (2FH),A     ;ENABLE PORT B
1828  FB        EI              ;ENABLE INT ON Z80
;
1829  C32918 LOOP JP LOOP       ;WAIT FOR INTERRUPTS
```

รูปที่ 2.5.17 ตัวอย่างโปรแกรมแสดงการสั่งให้ PIO ทำงานในโหมด 2

2.5.9 การใช้งาน PIO ในโหมด 3 (CONTROL MODE)

การทำงานในโหมดนี้จะเป็นการทำงานในลักษณะที่ไม่มีการทำ HAND-SHAKE และสามารถที่จะโปรแกรมให้แต่ละบิทของแต่ละพอร์ทให้เป็นอินพุท หรือเอาท์พุทก็ได้ เช่น อาจจะให้บิท B0,B3 และ B6 เป็นอินพุทและบิทที่เหลือของพอร์ท B เป็นเอาท์พุทก็ได้ และที่สำคัญอีกอย่างหนึ่งก็คือ ในโหมดนี้สามารถที่จะใช้พอร์ททั้ง 2 แยกกันได้อย่างอิสระ.

เช่นเดียวกับในโหมดอื่น ๆ การที่จะสั่งให้ PIO ทำงานในโหมดนี้ได้นั้น จะต้องทำการส่งคำสั่งเลือกโหมด (MODE WORD) ไปยังรีจิสเตอร์ควบคุมของพอร์ทที่ต้องการ โดยคำสั่งเลือกโหมดสำหรับโหมด 3 นี้มีค่าเป็น 0CFH หลังจากที่ได้ทำการโปรแกรมเลือกโหมดนี้แล้วจะต้องทำการส่งข้อมูลไบท์ต่อไปให้กับ PIO ทันทีโดยที่ PIO จะถือว่าข้อมูลไบท์นี้เป็นข้อมูลที่

ใช้ในการเลือกว่าจะให้บิทใดของพอร์ทที่ถูกโปรแกรมไว้เป็นอินพุทและให้บิทใดเป็นเอาท์พุท คือ ถ้าบิทใดของข้อมูลในไบท์นี้เป็น "0" ก็แสดงว่าต้องการให้บิทนั้นเป็นเอาท์พุท และถ้าบิทใดเป็น "1" ก็แสดงว่าต้องการให้บิทนั้นเป็นอินพุท เช่นในกรณีที่ต้องการให้พอร์ท B ทำงานในโหมด 3 และให้บิท B0,B3 และ B6 เป็นอินพุทก็จะต้องส่งข้อมูลไบท์ที่ต่อจากคำสั่งเลือกโหมดเป็น 01001001B = 49H (บิท 0,3 และ 6 เป็น "1")

สำหรับตัวอย่างโปรแกรมข้างล่างนี้เป็นโปรแกรมที่ใช้ในการสั่งให้พอร์ท B ทำงานในโหมด 3 โดยที่บิท B0,B3 และ B6 เป็นอินพุท

```
LD A, OCFH
OUT (2FH),A          : โปรแกรมให้พอร์ท B ทำงานในโหมด 3
LD A, 49H
OUT (2FH),A          : โปรแกรมบิท
                     : B3,B3 และ B6 เป็นอินพุท
```

หลังจากการทำงานในโปรแกรมข้างต้นแล้วพอร์ท B ของ PIO ก็จะถูกโปรแกรมให้ทำงานในโหมด 3 โดยที่บิท B0,B3 และ B6 ใช้เป็นอินพุท ในตอนนี้ผู้เขียนโปรแกรมก็สามารถที่จะส่งข้อมูลออกไปยังบิทเอาท์พุท (บิท B1,B2,B4,B5 และ B7) และรับข้อมูลเข้าทางบิทอินพุทได้แล้ว โดยที่เมื่อทำการอ่านข้อมูลจากบิทอินพุทนั้น ข้อมูลจากบิทที่ถูกโปรแกรมให้เป็นอินพุทนั้นก็จะถูกส่งไปให้กับ Z80 ตามข้อมูลที่ได้รับเข้ามาจากภายนอก ส่วนข้อมูลจากบิทที่ถูกโปรแกรมให้เป็นเอาท์พุทนั้นก็จะเป็นข้อมูลเดิมที่ Z80 ส่งให้กับบิทเอาท์พุทเหล่านี้ เช่น ในกรณีที่ Z80 ส่งข้อมูล 00101B ให้กับอุปกรณ์ภายนอกโดยส่งผ่าน PIO ทางบิท B1,B2,B4,B5 และ B7 ตามลำดับ เมื่อ Z80 ทำการอ่านข้อมูลจากทางบิท B0,B3 และ B6 (ในที่นี้สมมติให้ข้อมูลที่อุปกรณ์ภายนอกส่งให้กับบิททั้งสามเป็น 011B ตามลำดับ) ดังนั้นข้อมูลจากพอร์ B ที่ PIO ทำการส่งออกมาบนบัสข้อมูล เพื่อส่งให้กับ Z80 จะเป็น 11011000B (OD8H) แต่ข้อมูลที่ Z80 ต้องการจะมีเพียง 3 บิทเท่านั้น ดังนั้นในกรณีนี้ ผู้เขียนโปรแกรมจะต้องทำการตรวจสอบข้อมูลที่รับเข้ามาเอง.

สำหรับการอินเทอร์รัพท์ในโหมดนี้จะขึ้นอยู่กับส่วนที่เป็นบิทอินพุทเท่านั้น และ PIO จะทำการส่งสัญญาณ $\overline{INT}$ ให้กับ Z80 ก็ต่อเมื่อข้อมูลที่บิทอินพุทที่ PIO รับเข้ามานั้นตรงกับเงื่อนไขที่กำหนด (โปรแกรม) ไว้ใน INTERRUPT CONTROL WORD เท่านั้น.

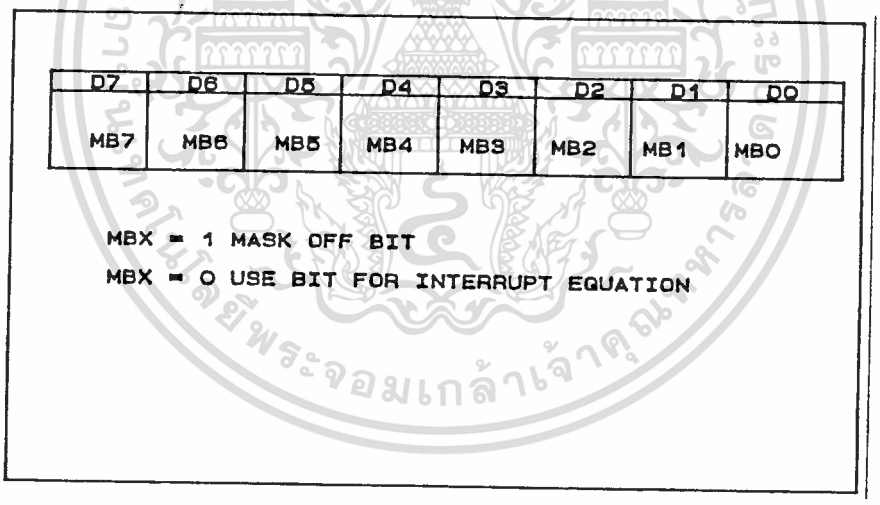
พิจารณาจากการจัดเรียงบิทของ INTERRUPT CONTROL WORD ในรูปที่ 2.5.13

1. บิท D6 ; (AND/OR) : ข้อมูลในบิทนี้จะใช้ในการเลือกระหว่างการ AND

หรือการ OR กันกับอินพุทที่ PIO รับเข้ามา โดยถ้าข้อมูลในบิทนี้เป็น "1" จะเป็นการเลือกการ AND กันของข้อมูลและ PIO จะทำาการส่ง INT ให้กับ Z80 เมื่อข้อมูลในบิทที่สนใจบิทใดบิทหนึ่งอยู่ในสภาวะแอคทีพ สำหรับสภาวะแอคทีพที่กล่าวถึงนี้อาจจะเป็น "0" หรือ "1" ก็ได้ และสามารถที่จะเลือกให้ความสนใจต่อบิทอินพุทใดก็ได้โดยการโปรแกรมในบิท D5 และ D4 ตามลำดับ

2. บิท D5 ; (HIGH/LOW) : บิทนี้จะใช้สำหรับการเลือกสภาวะแอคทีพที่อินพุทของ PIO โดยถ้าบิทนี้เป็น "1" สภาวะแอคทีพก็คือลอจิก "1" แต่ถ้าบิทนี้เป็น "0" สภาวะแอคทีพก็คือลอจิก "0"

3. บิท D4 ; (MASK FOLLOWS) : ถ้าข้อมูลในบิทนี้เป็น "1" จะเป็นการบอกให้ PIO ทราบว่า ข้อมูลที่จะถูกส่งตามมานั้นเป็น MASK WORD ซึ่งใช้ในการเลือกว่าอินพุทบิทใดเป็นบิทที่สนใจ รูปที่ 2.5.18 จะแสดงตำแหน่งของบิทต่าง ๆ บน MASK WORD ถ้าสนใจที่จะตรวจสอบิทใดก็ให้บิทนั้นเป็น "0" ส่วนบิทที่ไม่สนใจก็ "1" เช่นในกรณีที่สนใจที่จะตรวจสอบบิท B3 และ B4 ก็ให้ข้อมูลใน MASK WORD นี้มีค่าเป็น 11100111B หรือ OE7H เป็นต้น



รูปที่ 2.5.18 การจัดเรียงบิทบน MASK WORD

เพื่อที่จะอธิบายการทำงานในโหมดนี้ของ PIO จะขอยกตัวอย่างดังนี้คือ

1. ทำการโปรแกรมพอร์ท B ของ PIO ให้อยู่ในโหมด 3
2. โปรแกรมให้บิท B2,B3 และ B4 เป็นบิทอินพุท
3. กำหนด INTERRUPT VECTOR เป็น 0BH
4. โปรแกรม INTERRUPT CONTROL WORD โดยทำการ ENABLE

INTERRUPT (บิท D7 = "1") เลือกการ AND (บิท D6 = "1") ให้สภาวะแอคทีพเป็นลอจิก "0" (บิท D5 = "0"), มีการส่ง MASK WORD ตามมา (MASK FOLLOW ; บิท D4 = "1") และกำหนดให้บิท B3 และบิท B4 เป็นบิทที่ต้องการที่จะตรวจสอบ ดังนั้น MASK WORD จะมีข้อมูลเป็น OE7H

โปรแกรมข้างล่างนี้เป็นตัวอย่างโปรแกรมที่ใช้ในการโปรแกรม PIO ให้ทำงานตามตัวอย่างข้างต้น

```
LD A,0DFH
OUT (2FH),A          : โปรแกรมพอร์ท B ให้ทำงานในโหมด 3
LD A, 1CH
OUT (2FH),A          : ให้บิท B2,B3 และ B4 เป็นอินพุท
LD A,08H
OUT (2FH),A          : INTERRUPT VECTOR = 08H
LD A, 0D7H
OUT (2FH),A          : ENABLE INT, AND, MASK FOLLOWS
LD A,0E7H
OUT (2FH),A          : ทำการตรวจสอบบิท B3 และ B4
```

สำหรับโปรแกรมส่วนต่อไปที่จะต้องเขียนขึ้น คือ ส่วนของโปรแกรมตอบสนองการอินเทอร์รัพท์ที่ทำหน้าที่ ในการตอบสนองต่อการขออินเทอร์รัพท์ของ PIO และดังที่ได้กล่าวไว้แล้วว่า การใช้คำสั่ง RETI ในตอนสิ้นสุดของโปรแกรมตอบสนองการอินเทอร์รัพท์นั้น ชิพซัพพอร์ทของ Z80 สามารถที่จะรับรู้ต่อคำสั่งนี้ได้คือในขณะที่ Z80 อยู่ในช่วงการเฟทช์ออพโคดอยู่นั้น (ขา $\overline{M1}$ และ $\overline{RD}$ เป็น "0" พร้อมกัน) ชิพซัพพอร์ทของ Z80 จะทำการตรวจสอบออพโคดบนบัสข้อมูลว่าเป็นออพโคดของคำสั่ง RETI หรือไม่ ถ้าหากพบว่าเป็นออพโคดของคำสั่ง RETI ก็จะทำการถอนสัญญาณ $\overline{INT}$ ออกเอง ซึ่งเป็นการช่วยลดภาระของผู้เขียนโปรแกรมลงได้.

2.5.10 การ ENABLE และ DISABLE การขออินเทอร์รัพท์.

ในกรณีที่เริ่มการทำงานใหม่ PIO อาจสร้างสัญญาณ $\overline{INT}$ ขึ้นมา โดยที่ไม่ต้องการ ซึ่งอาจจะทำให้เกิดปัญหากับระบบได้ ดังนั้นในตอนเริ่มต้นการทำงานของระบบ จึงควรที่จะทำการ DISABLE การขออินเทอร์รัพท์ของ PIO เอาไว้ก่อน ดังโปรแกรมข้างล่าง (พอร์ทแอดเดรสที่ใช้เป็นค่าที่ได้จากการถอดรหัสในรูปที่ 2.5.3)

```
DI                      : ทำการ DISABLE INTERRUPT Z80
LD A,03H
OUT (2EH),A             : ทำการ DISABLE PORT A ของ PIO
OUT (2FH),A             : ทำการ DISABLE PORT B ของ PIO
-----------------------------------------
!     ส่วนของโปรแกรมที่ใช้ในการโปรแกรม PIO  !
-----------------------------------------
EI                      : ทำการ ENABLE INTERRUPT Z80
```

จากโปรแกรมข้างต้นจะเห็นว่าทำการ DISABLE ทั้ง Z80 และ PIO ไว้ และทำการ ENABLE INTERRUPT อีกครั้งหลังจากที่ได้ทำการโปรแกรม PIO เรียบร้อยแล้ว.

2.5.11 PRIORITY INTERRUPT ของ PIO.

ในหัวข้อนี้จะกล่าวถึงขาที่ใช้ในการอินเทอร์รัพท์ทั้ง 3 ขาคือ $\overline{INT}$,IEI และ IEO

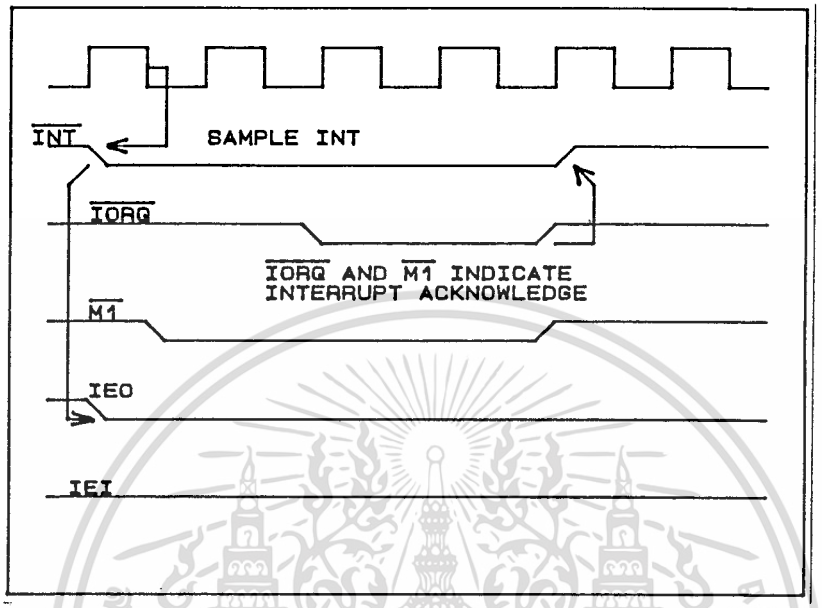
1. $\overline{INT}$: ขานี้เป็นเอาท์พุทของ PIO ซึ่งจะต่อโดยตรงกับขา $\overline{INT}$ ของ Z80

2. IEI : (INTERRUPT ENABLE INPUT) : ขานี้จะทำหน้าที่ในการ DISABLE หรือ ENABLE การขออินเทอร์รัพท์ของ PIO คือ ถ้าขา IEI นี้ได้รับลอจิก "1" PIO ก็สามารถที่จะทำการขออินเทอร์รัพท์ได้ แต่ถ้าหากว่าได้รับลอจิก "0" PIO จะไม่สามารถที่จะทำการขออินเทอร์รัพท์จาก Z80 ได้

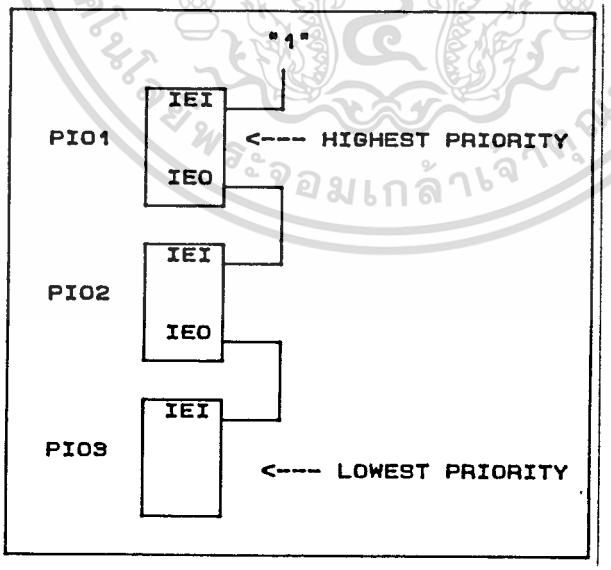
3. IEO : (INTERRUPT ENABLE OUTPUT) : ในขณะที่ PIO ทำการขออินเทอร์รัพท์อยู่นั้น PIO จะทำให้ขา IEO นี้เป็น "0" และขานี้จะกลับเป็น "1" อีกครั้ง เมื่อ Z80 ทำการเฟทช์คำสั่ง RETI จากหน่วยความจำ (ชิพซัพพอร์ทของ z80 จะคอยตรวจสอบออพโคดบนบัสข้อมูลทุกครั้งที่ Z80 ทำการเฟทช์ออพโคดจากหน่วยความจำ).

จากลักษณะการทำงานของ IEI และ IED สามารถที่จะนำเอาขาทั้งสองนี้มาใช้ในการจัดลำดับการอินเทอร์รัพท์แบบ DAISY CHAIN ได้ดังรูปที่ 2.5.19 โดยอุปกรณ์ที่มี PRIORITY สูงสุดนั้นขา IEI จะถูกต่อเข้ากับลอจิก "1" ซึ่งจะทำให้อุปกรณ์ตัวนี้สามารถที่จะทำการขออินเทอร์รัพท์ได้ตลอดเวลา ส่วนอุปกรณ์ที่มี PRIORITY ต่ำกว่านั้ ขา IEI จะถูกนำไปต่อเข้ากับขา IED ของตัวที่มี PRIORITY สูงกว่าด้วยเหตุนี้จึงทำให้อุปกรณ์ที่มี PRIORITY ต่ำกว่าไม่สามารถที่จะทำการขออินเทอร์รัพท์ได้ในกรณีที่อุปกรณ์ที่ PRIORITY สูงกว่ากำลังทำการขออินเทอร์รัพท์อยู่.

รูป 2.5.19 จะแสดงไดอะแกรมเวลาของการขออินเทอร์รัพท์ของ PIO (ชิพชัพพอร์ทเบอร์อื่น ๆ ของ Z80 จะมีลักษณะเช่นเดียวกัน).



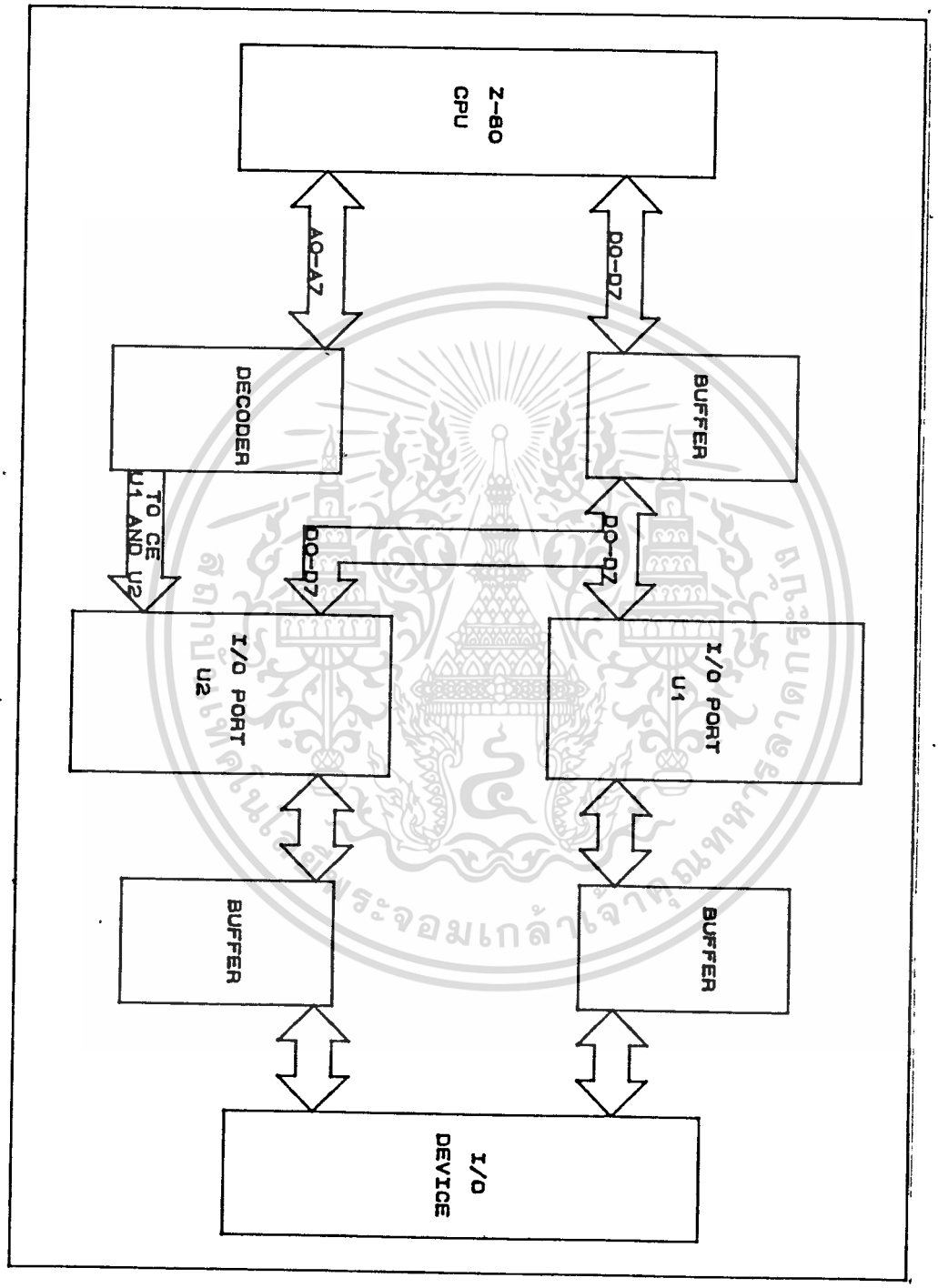
รูปที่ 2.5.19 ไดอะแกรมเวลาของการขออินเทอร์รัพท์ของ PIO.



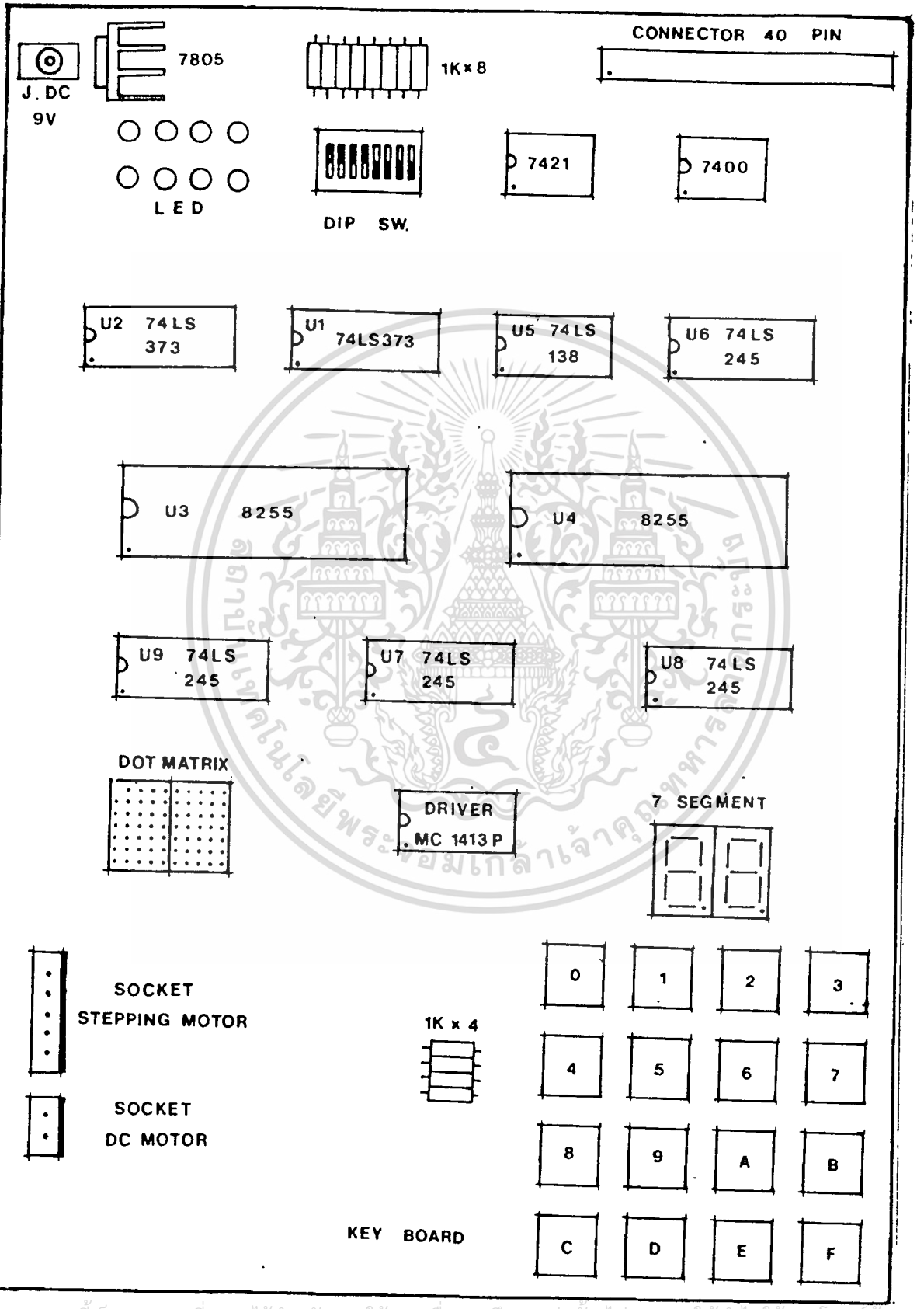
รูปที่ 2.5.20 การทำ DAISY CHAIN INTERRUPT

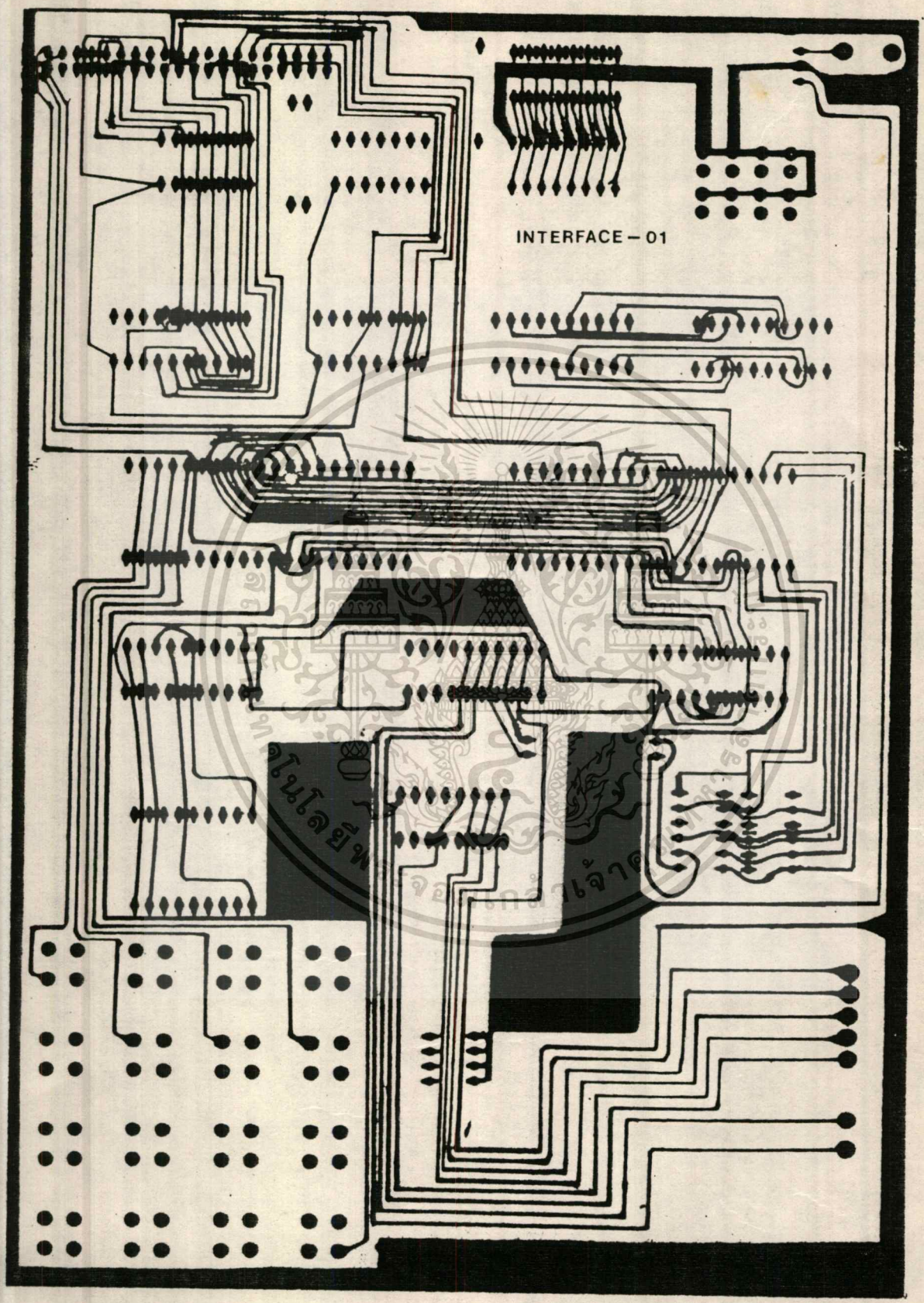
บทที่ 3.

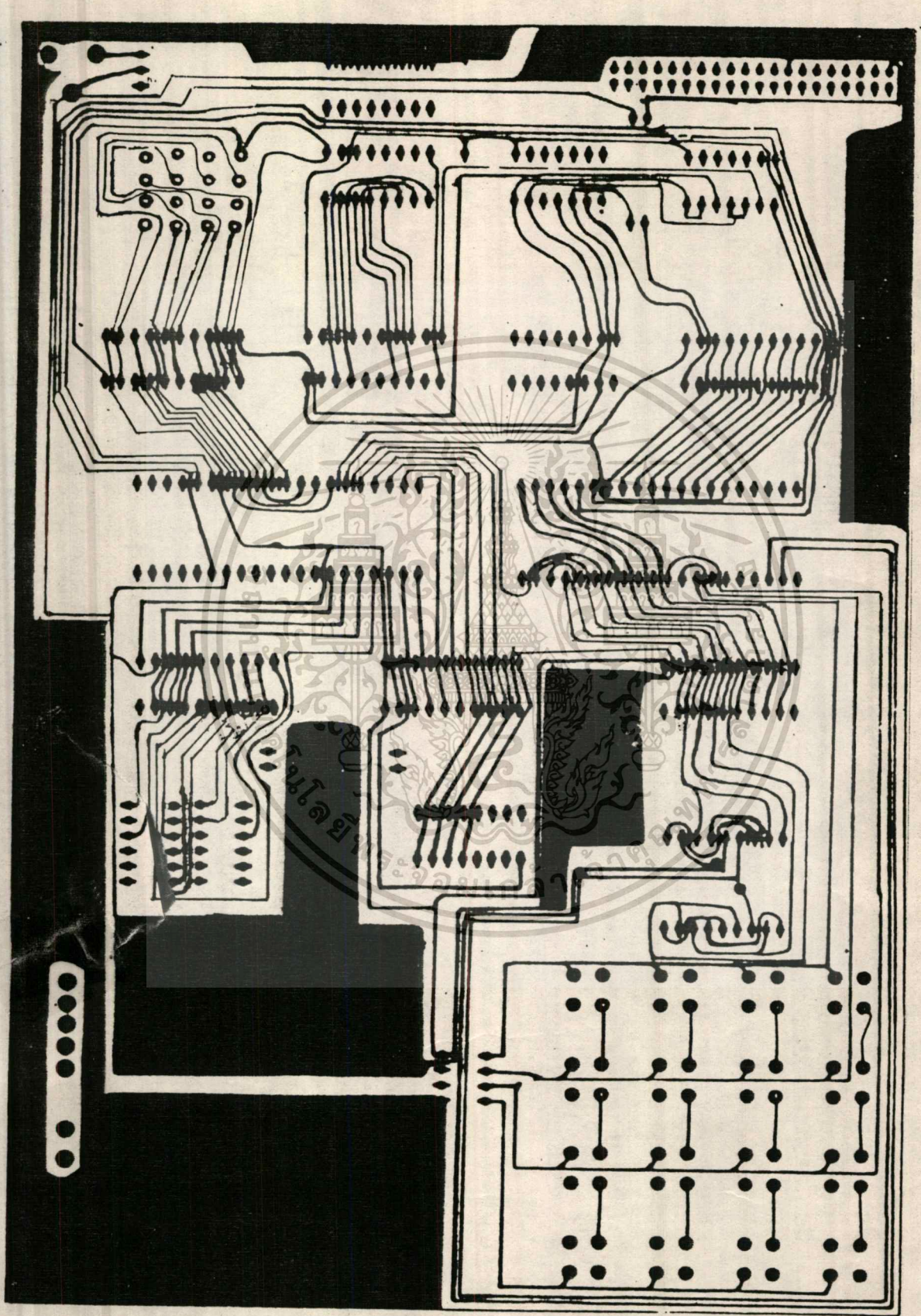
การออกแบบและการสร้างชุดผึก INTERFACE



รูปแสดง BLOCK DIAGRAM ของชุดผึก INTERFACE



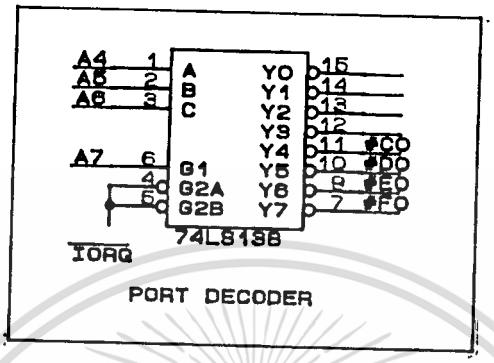




จากวงจร สามารถแยกส่วนต่าง ๆ ของวงจรได้ดังนี้

1. ส่วนวงจรดีโคด (DECODE)

จะเห็นได้ว่าจาก Z - 80 วงจรดีโคด 74138 ได้สัญญาณออกมาตามที่ต้องการดังรูป



2. ส่วนวงจร BASIC INPUT OUTPUT ประกอบด้วย

IC U1 74373 ใช้สำหรับ LED 8 ดวงโดยกำหนดให้เป็นพอร์ทเบอร์ D0

IC U2 74373 ใช้สำหรับเป็นอินพุท จาก DIP SWITCH โดยกำหนดให้เป็นพอร์ทเบอร์ C0

3. ส่วนของ IC U3 8255 PIA แบ่งออกเป็น 3 พอร์ท คือเริ่มจาก D0 - D3

#D0 ใช้ขับ DOT MATRIX

#D1 และD2 ใช้ขับ DOT MATRIX

#D2 (3,4) ใช้ขับ DC MOTOR

#D2 (5,8) ใช้ขับ STEPPING MOTOR

#D3 เป็น CONTROL WORD

4. ส่วนของ IC U4 8255 แบ่งออกเป็น 3 พอร์ท คือเริ่มจาก F0 - F3

F0 ใช้ขับ 7 SEGMENT

F1 (1,2)ใช้ขับ เอาท์พุทสำหรับ COMMON ของ 7 SEGMENT

F2 (5,8) ใช้เป็นอินพุท SCAN KEYBOARD

F3 เป็น CONTROL WORD

5. บัพเฟอร์ดาต้า (BUFFER DATA)

ใช้ IC U6 #74245 เป็นบัพเฟอร์ดาต้าให้กับ IC 8255 และ 74373 ทุกตัว

TRUTH TABLE IC # 74245

INPUT		OUTPUT
E	DIR	
L	L	B TO A
L	H	A TO B
H	X	ISOLATION

จากตารางข้างบนจะเห็นได้ว่า เมื่อเราต้องการรับอินพุทจากอุปกรณ์ภายนอกเราต้องป้อน LOW ที่ขา E และป้อน LOW ที่ขา DIR ซึ่งจะทำให้ตัวมันทำหน้าที่เป็นอินพุท คือส่งสัญญาณจากภายนอก (อุปกรณ์ภายนอก) ไปเข้า Z-80 นั่นเอง

แต่เมื่อเราต้องการให้มันเป็นเอาท์พุท คือต้องป้อน LOW ให้ E และป้อน HIGH ให้ DIR ซึ่งจะทำให้ตัวมันทำหน้าที่เป็น เอาท์พุท ส่งข้อมูลออกไปจาก Z-80 ไปสู่อุปกรณ์ภายนอกที่เราต้องการจะติดต่อด้วย

จากชุด DECODER ที่นำมาต่อกับขา 19 จะเห็นได้ว่า เมื่อส่งให้พอร์ทเบอร์ใดพอร์ทหนึ่งทำงานจะมีสัญญาณที่จะ CS ตัวใดตัวหนึ่ง (สำหรับ 74373 เราต้องต่อ NOT GATE อีก 1 ตัว เนื่องจาก 74373 ทำงานที่ LOGIC '1'

5. BUFFER O/P U7,U8 # 74245

เป็นตัวช่วยจ่ายกระแสให้อุปกรณ์ เอาท์พุท โดยเราใช้เบอร์ 74245 เป็น BUFFER โดยต่อขา 1 ไว้ที่ไฟบวกและขา 19 ให้ลงกราวด์ ซึ่งจะทำให้ 74245 เป็น BUFFER ตลอดเวลาซึ่งจะใช้ในกรณีของอุปกรณ์ภายนอกดึงกระแสมากเกินไป

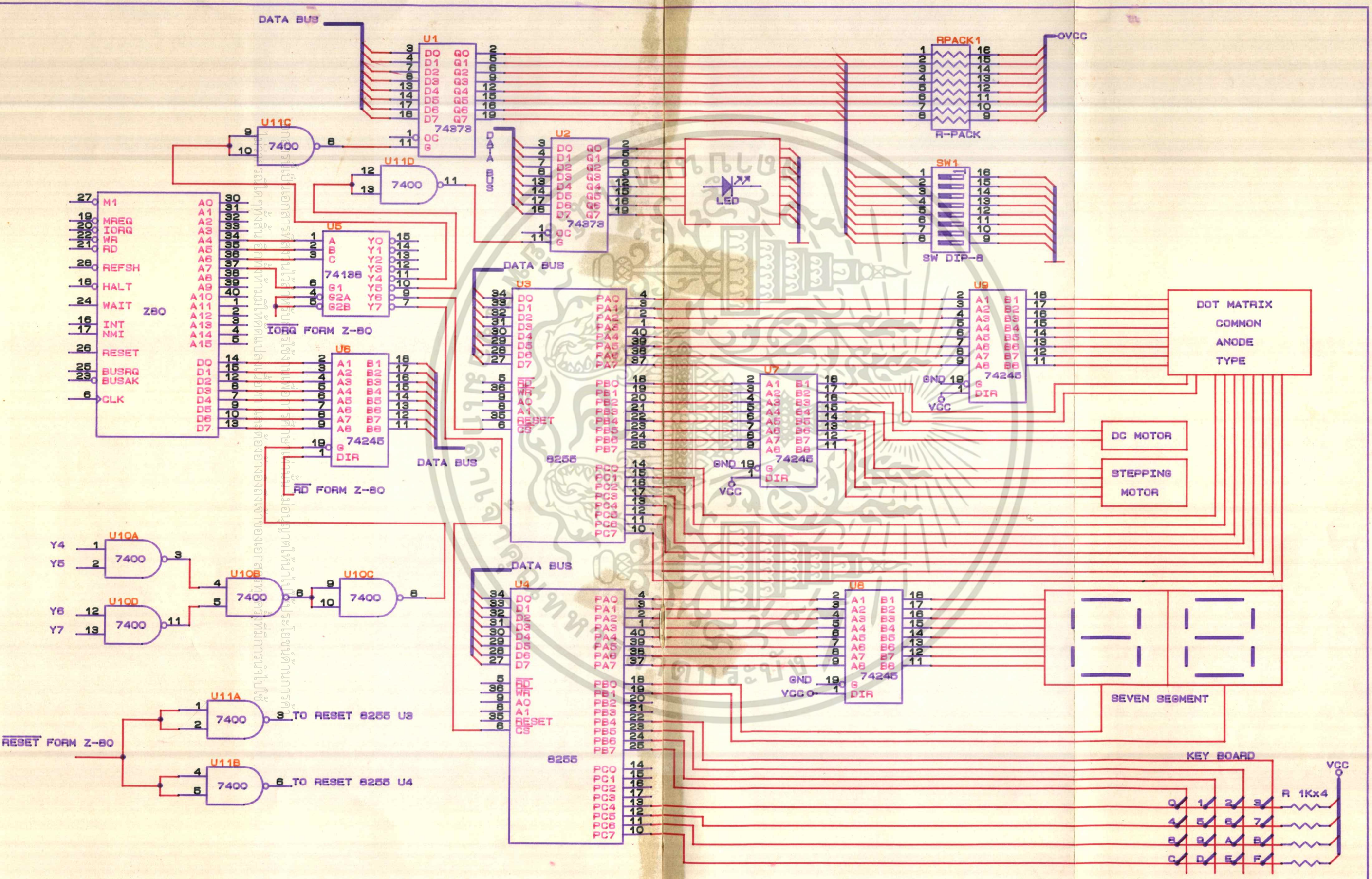
สรุปการทำงานของวงจร

เมื่อเริ่ม RUN PROGRAM สัญญาณต่าง ๆ จะมีลักษณะดังนี้

1. D0 - D7 จะมารออยู่ที่ 74245 BUFFER DATA
2. A4 A5 A6 A7 และ IORQ จะ DECODE ตามโปรแกรมที่เราออกแบบไว้ เมื่อได้ตามต้องการ ก็จะมีผลทำให้ 74245 BUFFER DATA ทำงาน (เมื่อ CPU ทำคำสั่ง อินพุทหรือเอาท์พุท สัญญาณ RD จาก CPU จะมีผลทำให้ 74245 ทำงานตามคำสั่งนั้น)

3. สมมุติเป็นโปรแกรมเอาท์พุท D0 - D7 จะผ่าน 74245 BUFFER ไปยัง IC O/P คือ 74373(U1-U2) และ 8255(U3-U4) แต่สัญญาณ CS จาก 74138 จะเป็นตัวเลือกให้ตัวไหนทำงาน ซึ่งทั้งนี้ขึ้นอยู่กับโปรแกรมที่กำหนดไว้ให้ตัวไหนทำงาน

<u>หมายเหตุ</u> โปรแกรมสำหรับ 8255 ต้องมี CONTROL WORD ให้กับ 8255 เพื่อเลือกว่าต้องการให้พอร์ทใดเป็น อินพุท หรือ เอาท์พุท โดยมีตารางบอกไว้


DATA BUS
U1
74373
U11C
7400
U11D
7400
U2
74373
DATA BUS
RPACK1
R-PACK
+VCC
LED
SW1
SW DIP-8
Z80
M1
MREQ
IORQ
WR
RD
REFSH
HALT
WAIT
INT
NMI
RESET
BUSRQ
BUSAK
CLK
A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 A10 A11 A12 A13 A14 A15
D0 D1 D2 D3 D4 D5 D6 D7
U5
74188
IORQ FORM Z-80
U6
74245
DIR
RD FORM Z-80
DATA BUS
U3
8255
PA0 PA1 PA2 PA3 PA4 PA5 PA6 PA7
PB0 PB1 PB2 PB3 PB4 PB5 PB6 PB7
PC0 PC1 PC2 PC3 PC4 PC5 PC6 PC7
RD WR A0 A1 RESET CS
U7
74245
GND
VCC
U9
74245
DOT MATRIX
COMMON
ANODE
TYPE
DC MOTOR
STEPPING
MOTOR
U10A
U10B
U10C
U10D
7400
Y4
Y5
Y6
Y7
DATA BUS
U4
8255
U8
74245
SEVEN SEGMENT
U11A
U11B
7400
RESET FORM Z-80
TO RESET 8255 U3
TO RESET 8255 U4
KEY BOARD
R 1Kx4
VCC

บทที่ 4

การทดลองและผลการทดลอง

จากวงจรที่ออกแบบและสร้างขึ้น สามารถทดสอบความถูกต้องของวงจรด้วยโปรแกรมง่ายๆ ดังนี้

โปรแกรมทดสอบ

โปรแกรมขับ LED

```
LD   A,FF
OUT  (C0),A
```

โปรแกรม LED โดยรับข้อมูลจาก DIP SW.

```
IN   A,(D0)
OUT  (C0),A
```

อาจเพิ่มโปรแกรมวนลูป เพื่อรับข้อมูลจาก DIP SW. ได้ตลอดเวลา

```
START : IN   A,(D0)
        OUT  (C0),A
        JR   START
```

โปรแกรมขับ 7 SEGMENT

จากวงจรได้โค๊ดตัวเลขและตัวหนังสือดังนี้

0 = 3F	9 = 6F
1 = 06	A = 77
2 = 5B	B = 7C
3 = 4F	C = 39
4 = 06	D = 5E
5 = 6D	E = 70
6 = 7D	F = 71
8 = 7F	

โปรแกรมทำให้ 7 SEGMENT แสดงผลเป็นเลข 0

ขั้นตอนแรกส่ง CONTROL WORD (80)เพื่อทำให้ PORT A ,B และ C เป็น OUTPUT PORT จะได้ PROGRAM เพื่อส่ง CONTROL WORD ดังนี้

```
LD  A,80
OUT (F3),A
```

หลังจากส่ง CONTROL WORD แล้ว ป้อนโปรแกรมเพื่อให้ 7 SEGMENT แสดงผลเป็นเลข 0 ดังนี้

```
LD  A,3F
OUT (F0),A
LD  A,0E
OUT (F1),A
```

ถ้าต้องการให้ 7 SEGMENT ตัวที่ 2 แสดงผลเป็นเลข 0 ให้ป้อน PROGRAM ดังนี้

```
LD  A,3F
OUT (F0),A
LD  A,0D
OUT (F1),A
```

ถ้าต้องการให้แสดงผล 2 ตัวพร้อมกัน ให้ใช้ PROGRAM วน LOOP

<u>หมายเหตุ</u> ควรทำให้ 7 SEGMENT ตัวที่ 1 ดับก่อน แล้วจึงให้ 7 SEGMENT ตัวที่ 2 ติด สลับกันไปโดยมีวงจรหน่วงเวลาในการแสดงผลของแต่ละตัวดังนี้

PROGRAM เพื่อส่ง CONTROL WORD

```
        LD  A,80
        OUT (F3),A
START:  LD  A,3F
        OUT (F0),A
        LD  A,0E
        OUT (F1),A
```

ส่วนหน่วงเวลาแสดงผล

```
        LD  D,FF
LOOP:   DEC D
        JR NZ,LOOP
```

ดับ SEGMENT ตัวที่ 1

```
        LD  A,0F
        OUT (F1),A
```

SEGMENT ตัวที่ 2 แสดงผลเลข 1

```
        LD  A,06
        OUT (F0), A
        LD  A,0D
        OUT (F1),A
```

หน่วงเวลาแสดงผล

ดับ Segemnt ตัวที่ 2

วนลูปแสดงผล

JP START

โปรแกรมขับ DOT MATRIX

จาก DOT MATRIX ที่ใช้เป็น COMMON ANODE

จากการทดลองได้โค๊ด

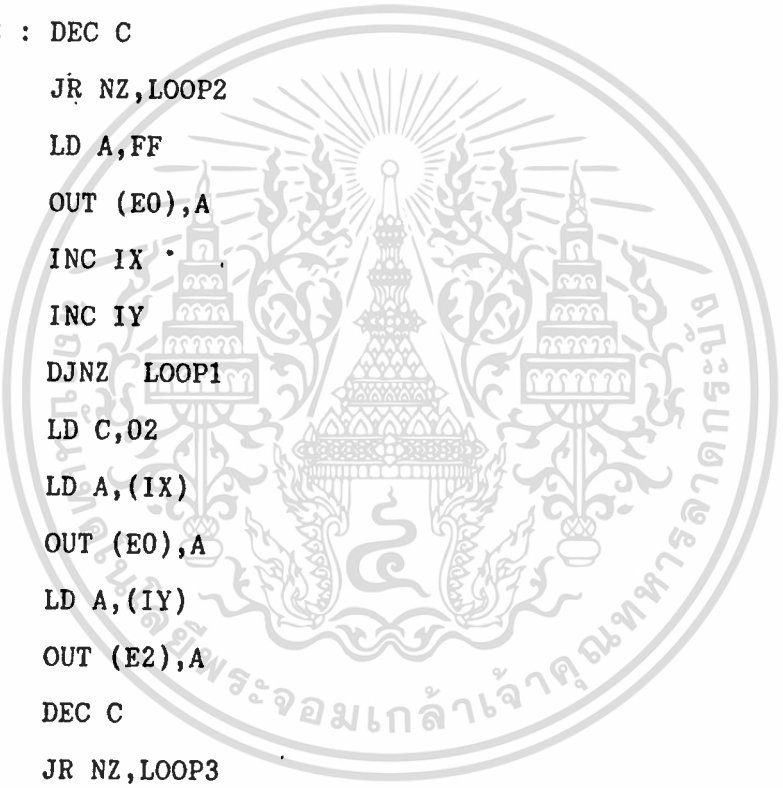
A=83 ED EE ED 83	B=00 B6 B6 B6 49
C=41 3E 3E 3E 5D	D=41 3E 3E 3E 5D
E=00 36 36 36 3E	F=00 76 76 76 7E
G=41 3E 3E 36 45	H=00 F7 F7 F7 00
I=FF 3E 00 3E FF	J=5F 3F 3E 40 FE
K=00 77 6B 5D 3E	L=00 30 30 30 30
M=01 7E 01 SE 01	N=00 FE FD FE 01
0=41 3E 3E 3E 41	P=00 76 76 76 79
Q=41 3E 3E 1F 01	R=00 76 66 56 39
S=19 36 36 36 4D	T=7E 7E 00 7E 7E
U=40 3F 3F 3F 40	V=60 5F 3F 5F 60
W=00 3F 43 3F 00	X 1C EB F7 ED 1C
Y=FC FB 07 FB FC	Z=1E A9 B9 BD BC
O=41 3E 3E 3E 41	I=FF FD 00 FF FF
2=1E 2E B9 3A BC	3=3E 39 39 39 00
4=FO F7 F7 OO F7	5=19 36 36 36 4D
6=4E 36 36 36 4D	7=1E 6E F9 FA FC
8=49 36 36 36 49	9=59 36 36 36 49

PROGRAM ส่ง CONTROL WORD ให้ IC U3 #8255 เป็นเอาท์พุท ทั้ง 3 พอร์ท

```
LD A,80
OUT (E3),A
```

ต้องการให้ Dot MATRIX แสดงผลเป็นตัวอักษร A และ B

```
        LD B,08
        LD IX,1900
        LD IY,1A00
LOOP1 : LD A,IX
        OUT E0,A
        LD A,(IY)
        OUT (E1),A
LOOP3 : LD C,FF
LOOP2 : DEC C
        JR NZ,LOOP2
        LD A,FF
        OUT (E0),A
        INC IX
        INC IY
        DJNZ  LOOP1
        LD C,02
        LD A,(IX)
        OUT (E0),A
        LD A,(IY)
        OUT (E2),A
        DEC C
        JR NZ,LOOP3
        JR START
```



โปรแกรม STEPPING MOTOR

```
CONTROL     LD A,80
            OUT (E3),A

โปรแกรม START: LD B,04
            LD D,10
```

```
LOOP1: LD A,D
       OUT (E1),A
       RLC D
       DJNZ  LOOP1
       JR  START
```

DC MOTOR

CONTROL

หมุนซ้าย
```
       LD A,04
       OUT (E1),A
```
หมุนขวา
```
       LD A,08
       OUT (E1),A
```

โปรแกรม รับคีย์บอร์ด

แยกโปรแกรมออกเป็น 2 ส่วน

1. ส่วนแสกนคีย์บอร์ด
2. ส่วนแสดงผล

ตัวอย่าง โปแกรมแสกนคีย์บอร์ด

```
       LD D,EE
       LD A,88
       OUT (F3), A
Loop : LD A,C
       OUT (F1),A
       IN  A,(F2)
       CPL
       AND F0H
       CP  00
       JR NZ,KEY
       RLC C
       JP LOOP
KEY :  LD D,A
```

```
        LD A,C
        AND A,0FH
        ADD A,D
```

ตัวอย่าง โปรแกรมแสดงผล

```
        LD HL,1900
        LD D,00
        LD E,A
        ADD HL,DE
        LD A,(HL)
        OUT (F0H),A
        LD A,FEH
        OUT (F1),A
        RST 38H
```

โค๊ดที่ได้จากการแสกนคีบอร์ด

C (39)-17	D (5E)-1B	E (79)-1D	F (71)-1E
8 (7F)-27	9 (6F)-2B	A (77)-2D	B (7C)-2E
4 (66)-47	5 (6D)-4B	6 (7D)-4D	7 (07)-4E
0 (3F)-87	1 (06)-8B	2 (5B)-8D	3 (4F)-8E

โปรแกรมใช้งาน

7 SEGMENT DISPLAY

```
Loop 2 : LD B, 10
         LD IX,1A50
         LD A, 80
         OUT (F3), A
Loop 1 : LD A,(IX+d)
         OUT (F0), A
         LD A,FE
         OUT (F1),A
         LD D,FF
Loop 4 : LD E,FF
Loop 3 : DEC E
         JR NZ,Loop 3
         DEC D
         JR NZ,Loop 4
         LD A,00
         OUT (F0),A
         INC IX
         DJNZ LOOP1
         JR LOOP2
```

1A50 = 3F 0

1A51 = 06 1

1A52 = 5B 2

1A53 = AF 3

1A54 = 66 4

1A55 = 6D 5

1A56 = 7D 6

1A57 = 07 7

1A58 = 7F 8

1A59 = 6F 9

1A5A = 77 A

1A5B = 7C B

1A5C = 39 C

1A5D = 5E D

1A5E = 79 E

1A5F = 71 F

การทดลองที่ 1

เรื่อง OUTPUT PORT

จุดประสงค์

1. เพื่อให้เข้าใจหลักการเบื้องต้นในการติดต่อระหว่างไมโครโปรเซสเซอร์กับอุปกรณ์ OUTPUT แบบง่าย
2. เพื่อให้สามารถออกแบบและประกอบวงจรเพื่อ DECODE PORT ตามต้องการ
3. เพื่อให้สามารถเขียนโปรแกรมควบคุมวงจรที่ออกแบบได้

ทฤษฎี

ขบวนการเอาท์พุทเป็นขบวนการในการส่งข้อมูลจาก MICRO PROCESSOR ออกมาทาง DATA BUS ไปยังอุปกรณ์ เอาท์พุท ซึ่งทำหน้าที่เป็นตัวรับข้อมูล ในระบบไมโครคอมพิวเตอร์ CPU จะต้องส่งสัญญาเลือกอุปกรณ์เอาท์พุท ตัวใดตัวหนึ่ง พร้อมกับส่งข้อมูลออกมารวมทั้งสัญญาณควบคุมที่จำเป็น

คำสั่งที่เกี่ยวกับ อินพุท-เอาท์พุท ของ Z-80 CPU

1. คำสั่งอินพุท-เอาท์พุทที่มีการอ้าง ADDRESS แบบ ABSOLUTE ADDRESSING

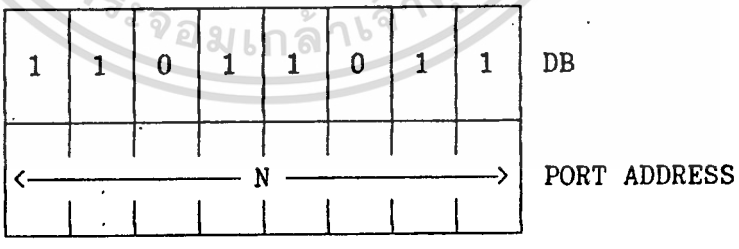
ได้แก่ คำสั่ง

```
IN A, (PORT)    ; INPUT
OUT (PORT),A    ; OUTPUT
```

IN A, (PORT)

FUNCTION : A <—— (PORT)

FORMAT :



* อุปกรณ์ประกอบที่ระบุ PORT ADDRESS โดยข้อมูลใน BYTE ที่ 2 จะนำข้อมูลไปเก็บไว้ใน ACCUMULATOR

ตัวอย่าง

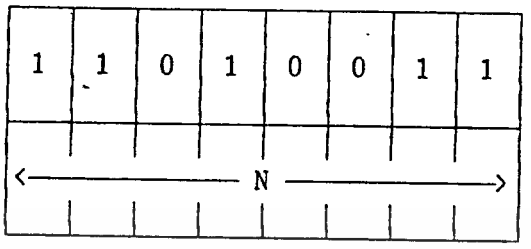
```
IN A, (0FFH)
```

หมายถึง การนำข้อมูลที่อยู่ที่ PORT หมายเลข FF มาเก็บไว้ใน ACCUMULATOR

OUT (PORT),A

FUNCTION : (PORT) <—— A

FORMAT :



* ข้อมูลใน ACCUMULATOR จะถูกนำออกจาก CPU ไปยังอุปกรณ์ภายนอก ซึ่งระบุตำแหน่งโดย (PORT)

ตัวอย่าง

OUT (00) , A

หมายถึง การนำเอาข้อมูลจาก ACCUMULATOR ส่งออกไปที่ PORT หมายเลข 00

2.คำสั่ง อินพุท-เอาท์พุท ที่มีการอ้าง ADDRESS แบบ INDIRECT ADDRESSING ได้แก่ คำสั่ง

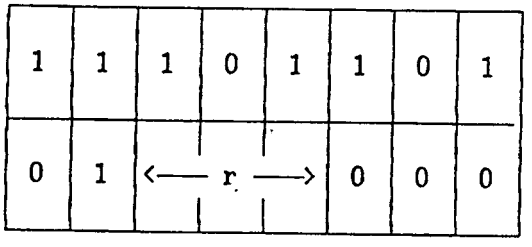
IN REG, (C) ; INPUT

OUT (C), REG ; OUTPUT

IN REG, (C)

FUNCTION : REG <—— (C)

FORMAT :



* ข้อมูลจากอุปกรณ์ภายนอกซึ่งระบุตำแหน่ง PORT โดยข้อมูลใน REGISTER C จะถูกนำไปเก็บไว้ใน REGISTER REG

BYTE CODE:

r:	A	B	C	D	E	H	L
ED	78	40	48	50	58	60	68

ตัวอย่าง

```
0E  00          LD C, 00H   ;  SET PORT NUMBER  = 00H
ED  40          IN B, (C)   ;  GET DATA FROM PORT TO
                               REGISTER B
```

เป็นคำสั่งในการนำข้อมูลจาก PORT หมายเลข 00 มาเก็บไว้ใน REGISTER B

OUT C, REG

FUNCTION : (C) <—— REG

FORMAT :

1	1	1	0	1	1	0	1	ED
0	1	<——	r	——>	0	0	1	

* ข้อมูลใน REGISTER REG จะถูกส่งไปยังอุปกรณ์ OUTPUT ที่ระบุ OUTPIT PORT โดยข้อมูลใน REGISTER C

BYTE CODE :

REG :	A	B	C	D	E	H	L
ED	79	41	49	51	59	61	69

ตัวอย่าง

```
0E  00          LD C, 00H   ;  SET PORT NUMERT = 00H
ED  59          OUT (C), E  ;  OUT DATA IN REGISTER E TO PORT
```

เป็นคำสั่งในการส่งข้อมูลจาก REGISTER E ไปยัง PORT หมายเลข 00

3.คำสั่ง อินพุท-เอาท์พุท แบบ BLOCK I/O

ได้แก่ คำสั่ง

INI

FUNCTION : (HL) <—— (C) ; B <—— B - 1

HL <—— HL + 1

FORMAT :

1	1	1	0	1	1	0	1	ED
1	0	1	0	0	0	1	0	A2

ตัวอย่าง

```
LD      HL, 19000H
LD      C,  00
LD      B,  10
INI
```

ข้อมูลจากอุปกรณ์ภายนอกซึ่งระบุตำแหน่ง INPUT PORT โดยข้อมูลใน REGISTER C (00) จะถูกนำไปเก็บไว้ในหน่วยความจำตำแหน่ง ซึ่งระบุ ADDRESS โดย REGISTER PAIR HL (ในที่นี้คือ ADDRESS 1900) จากนั้นข้อมูลใน REGISTER PAIR HL จะเพิ่มค่าขึ้นอีก 1 ในขณะที่ข้อมูลใน REGISTER B ลดค่าลง 1 ซึ่ง REGISTER B จะถูกใช้เป็น BYTE CONTER นับข้อมูลที่รับจากอุปกรณ์ภายนอกมายังหน่วยความจำ

INIR

FUNCTION : (HL) <—— (C) ; B <—— B - 1

HL <—— HL + 1

REPEAT UNTIL B = 0

FORMAT :

1	1	1	0	1	1	0	1	ED
1	0	1	1	0	0	1	0	B2

ตัวอย่าง

```
LD   HL,  1900H
LD   C,  00
LD   B,  10
INTR
```

ข้อมูลจากอุปกรณ์ภายนอกซึ่งระบุตำแหน่ง INPUT PORT โดยข้อมูลใน REGUTER C (00) จะถูกนำไปเก็บไว้ใน ADDRESS 1900 ถึง 1910 นั่นคือจนกว่าข้อมูลใน REGISTER B มีค่าเท่ากับ 0

OUTI

FUNCTION : (C) <—— (HL) ; B <—— B - 1 ;
HL <—— HL + 1

FORMAT :

1	1	1	0	1	1	0	1	ED
1	0	1	0	0	0	1	1	A3

* ข้อมูลในหน่วยความจำซึ่งระบุ ADDRESS โดยข้อมูลใน REGISTER PAIR HL จะถูกนำออกไปยังอุปกรณ์ เอาท์พุท ซึ่งระบุหมายเลข PORT โดยข้อมูลใน REGISTER C จากนั้นข้อมูลใน REGISTER PAIR HL จะเพิ่มค่าคำขึ้นอีก 1 ในขณะที่ข้อมูลใน REGISTER B ลดค่าลง 1

OTIR

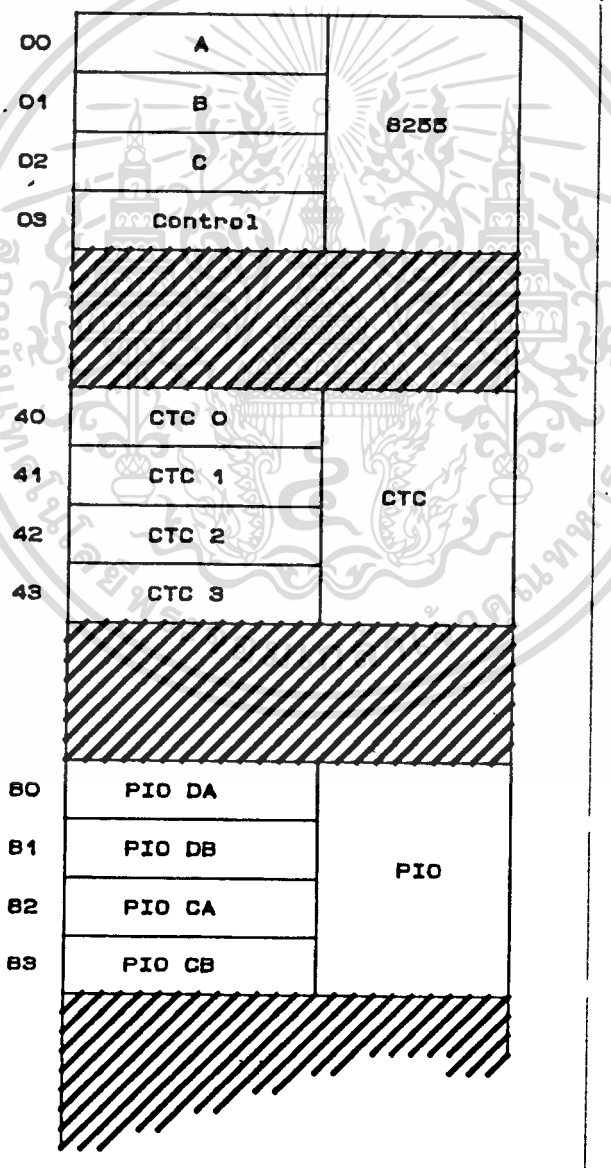
FUNCTION : (C) <—— (HL) ; B <—— B - 1 ;
HL <—— HL + 1
REPEAT UNTIL B = 0

FORMAT :

1	1	1	0	1	1	0	1	ED
1	0	1	1	0	0	1	1	B3

* เหมือนกับคำสั่ง OUTI แต่การส่งข้อมูลจะเป็นแบบต่อเนื่องจนกว่าข้อมูลใน REGISTER B = 0 จึงจะหยุด

ในการทดลองนี้จะอาศัยคุณสมบัติของ Z - 80 CPU เมื่อได้รับคำสั่งให้ติดต่อกับอุปกรณ์ภายนอก (คำสัง IN, OUT) Z- 80 จะส่งสัญญาณ $\overline{\text{IORQ}}$ (INPUT OUTPUT REQUEST) ออกมาพร้อมกับส่ง PORT ADDRESS ขนาด 8 BIT มาที่ ADDRESS BUS A0 - A7 และส่งข้อมูลออกมาที่ DATA BUS ให้กับอุปกรณ์ภายนอก ซึ่งเราสามารถนำเอา PORT ADDRESS มาทำการ DECODE เพื่อ กำหนดหมายเลข PORT ของอุปกรณ์ภายนอก ซึ่งการ DECODE จะต้องพิจารณา HARDWARE ของชุดฝึกที่ใช้ว่าได้ใช้ PORT หมายเลขใดไปแล้วบ้าง ในการทดลองนี้ใช้ชุดฝึก MPF-I ซึ่งมีแผนผัง I/O ดังนี้



I/O address map

การ DECODE PORT เราจะสามารถใช้ PORT หมายเลขใดก็ได้ที่ไม่ซ้ำกับแผนผัง I/O นี้

อุปกรณ์การทดลอง

1. ชุดฝึก MPF-I & SUPPLY
2. PROTOBOARD
3. IC 74138 , 7404 , 74373
4. LED
5. CONNECTOR

เตรียมการทดลอง

1. ศึกษาทฤษฎี และความเป็นมาของการทดลอง
2. ศึกษาคุณสมบัติของ IC 74138, 74373 ซึ่งใช้ในการทดลอง
3. ศึกษาตำแหน่ง CONNECTOR PIN # 1 ของ MPF-IB

ลำดับขั้นการทดลอง

1. ประกอบวงจรตามรูปที่ 1 โดยใช้ DATA SHEET ของ IC I4 373, 74138 และ 7404 ประกอบ

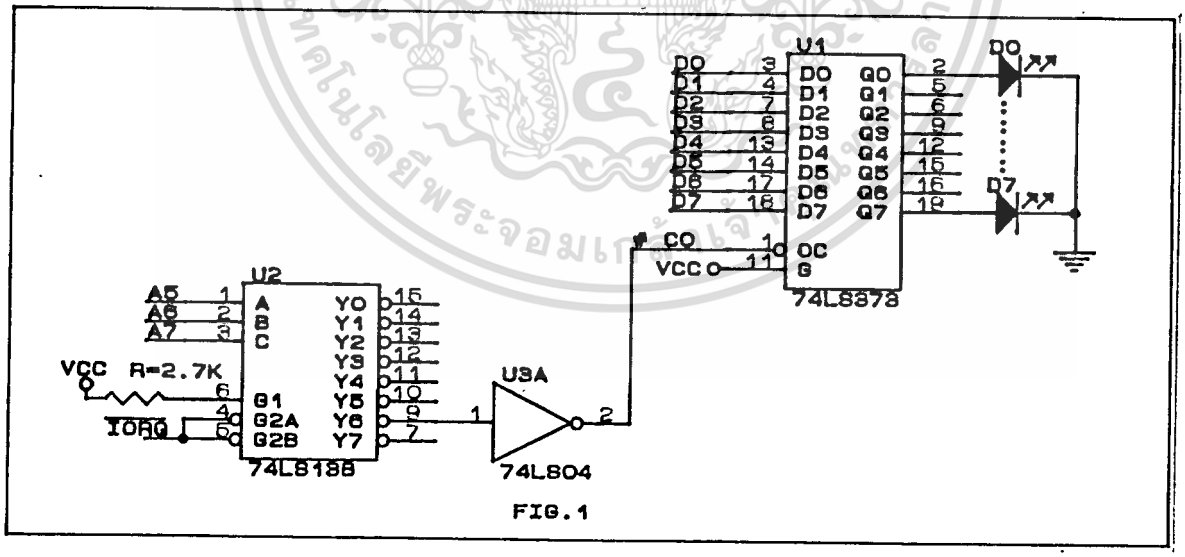


FIG.1

จากวงจรใช้ ADDRESS A_5, A_6, A_7 มาทำการ DECODE ใช้ OUTPUT Y_6 จะได้ว่า PORT ที่ออกแบบเป็น PORT หมายเลข CO

2. ทำการทดสองวงจรที่ต่อ ว่าถูกต้องหรือไม่ โดยการป้อนโปรแกรมดังนี้

ADDRESS	OP-CODE	MNEMONIC
1800	3E 0F	LD A, 0FFH
1802	D3 C0	OUT (C0), A
1804	FF	RST 38H

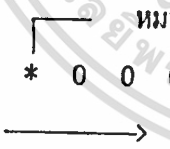
ถ้าวงจรถูกต้องหลังจาก EXECUTE PROGRAM แล้วจะทำให้ LED ติด 4 ดวง ที่ Q_3, Q_2, Q_1, Q_0 ทดลองเปลี่ยนข้อมูลที่ ADDRESS 1801 เป็น F0 แล้ว EXECUTE PROGRAM จะทำให้ LED ที่ Q_4, Q_5, Q_6, Q_7 ติด

หลังจากประกอบวงจรและทดสอบวงจรโดยโปรแกรมที่กำหนดให้ถ้าทุกอย่างถูกต้อง ติดไปเราจะศึกษาวิธีการออกแบบไฟวิ่งโดยใช้หลักการส่งข้อมูลที่ต้องการไปยัง OUTPUT PORT ที่ออกแบบไว้

หลักการของไฟวิ่งประกอบด้วยตัวแปร 2 ตัวคือ

1. จังหวะเวลา
2. รูปแบบของการติดดับของไป

จากวงจรเราได้ต่อ LED 8 ดวง และถ้าต้องการให้ติดทีละดวงเรียงจากขวาไปซ้ายดังรูป



ติดสว่างทีละดวงเรียงลำดับจากซ้ายไปขวา

เราสามารถกำหนดลักษณะการวิ่งดังกล่าว ในรูปแบบข้อมูลของไมโครโปรเซสเซอร์ได้ โดยกำหนดให้ดวงที่ติดสว่างมีลอจิก "1" และดวงที่ดับมีลอจิก "0" และทำการส่งข้อมูลที่กำหนดไว้นี้ออกมาทาง PORT โดยหน่วงเวลาในการส่งให้เท่า ๆ กัน เราก็จะได้ OUTPUT ที่สามารถจะนำไปขับหลอดไฟให้วิ่งได้ลักษณะข้อมูลเป็นดังนี้

ลำดับที่	หลอดไฟ	ลักษณะข้อมูล	ข้อมูลในแบบ HEX
1	* 0 0 0 0 0 0 0	1 0 0 0 0 0 0 0	80 H
2	0 * 0 0 0 0 0 0	0 1 0 0 0 0 0 0	40 H
3	0 0 * 0 0 0 0 0	0 0 1 0 0 0 0 0	20 H
4	0 0 0 * 0 0 0 0	0 0 0 1 0 0 0 0	10 H
5	0 0 0 0 * 0 0 0	0 0 0 0 1 0 0 0	08 H
6	0 0 0 0 0 * 0 0	0 0 0 0 0 1 0 0	04 H
7	0 0 0 0 0 0 * 0	0 0 0 0 0 0 1 0	02 H
8	0 0 0 0 0 0 0 *	0 0 0 0 0 0 0 1	01 H

จากนั้นเราก็สามารถทำโปรแกรมเพื่อกำหนดให้ไมโครโปรเซสเซอร์ส่งข้อมูลเหล่านี้ออกทาง PORT OUTPUT ที่กำหนดไว้

3. ป้อนโปรแกรมต่อไปนี้เข้าเครื่องและศึกษาการทำงานของโปรแกรม

```
Address  Machine Code   Label     Operand      Comment
1800     AF             START :   XOR A
1801     D3  C0                   OUT (C0),A   ;CLEAR DISDLAY
1803     06  08                   LD  B,08H    ; SET COUNTER
1805     21  00  19               LD  HL,1900H ; START DATA
                                                TABLE
1808     7E             LOOP  :   LD  A, (HL)
1809     D3  C0                   OUT (C0), A  ; DISPLAY DATA
180B     CD  20  18               CALL DELAY
180E     23                       INC  HL      ; NEXT TABLE
180F     10  F7                   DJNZ LOOP
1811     18  ED                   JR  START

1820     11  00  40   DELAY :    LD DE, 4000H ; DELAY =4000
1823     1B             DELAY 1:  DEC  DE
1824     7A                       LD  A,D
1825     B3                       OR  E
```

```
1826        20  FB                  JR  NZ, DELAY1
1828        C9                      RET
1900            80  40  20  10      TABLE
1904            08  04  02  01
```

4. เขียนโปรแกรมเพื่อให้ LED ติดกับตามรูปแบบต่อไปนี้

4.1 ติดทีละดวงจากขวาไปซ้าย

0 0 0 0 0 0 0 *

<———————

4.2 ติดทีละ 2 ดวง วิ่งออกจากกลางไปซ้ายและขวา

0 0 0 * * 0 0 0

<——— ———>

4.3 ติดทีละ 2 ดวง วิ่งจากริมขวาและซ้ายเข้าหากลาง

* 0 0 0 0 0 0 *

———> <———

5. สรุปและวิจารณ์ผลการทดลอง

<u>คำถาม</u>

1. จะมีวิธีการเปลี่ยนความเร็วของไฟวิ่งได้อย่างไร

2. ถ้าต้องการให้มีไฟวิ่งมากกว่า 8 ดวง จะมีวิธีการออกแบบวงจรอย่างไร

ผลการทดลอง

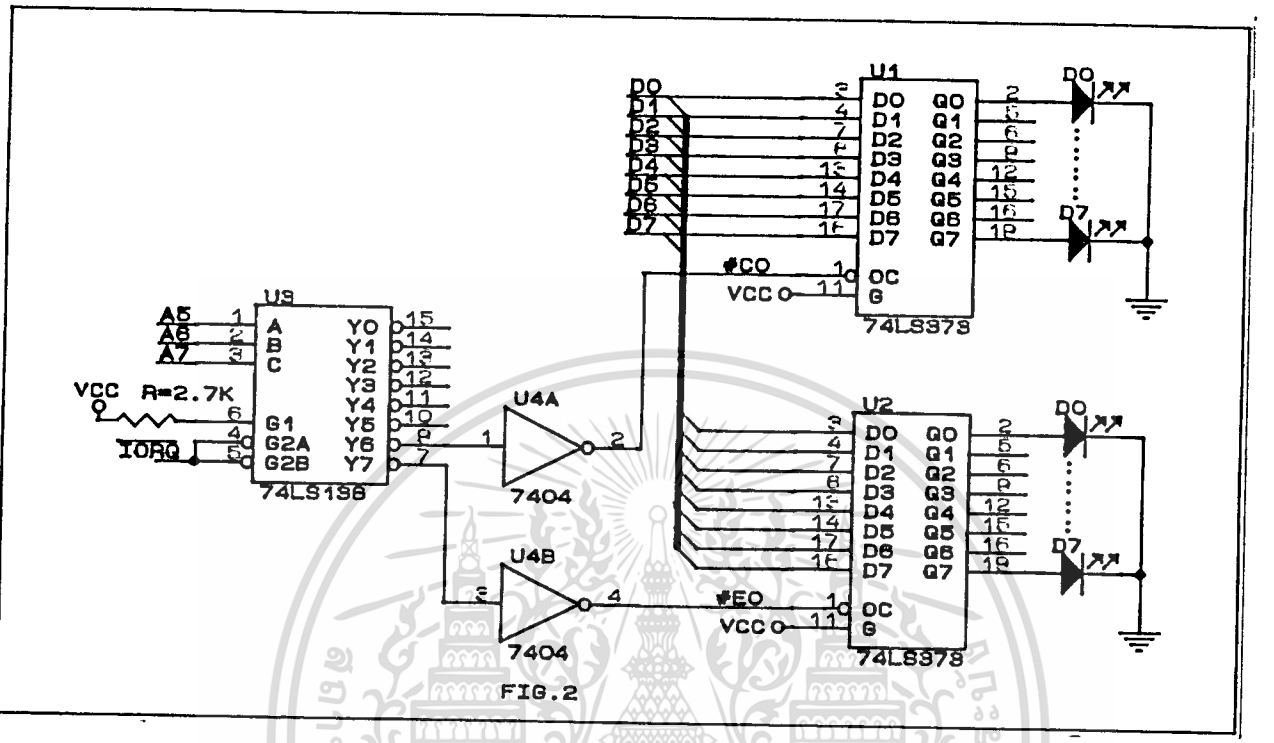
4.

Address	Machine Code	Label	Operand	Comment
1800	AF	START :	XOR A	
1810	D3 C0		OUT (C0),A	; CLEAR DISPLAY
1803	06 10		LD B,08H	; SET COUNTER
1805	21 00 19		LD HL,21900H	; START TABLE
1808	7E	LOOP:	LD A,(HL)	
1809	D3 C0		OUT (CO),A	; DISPLAY DATA
180B	CD 20 18		CALL DELAY	
180E	23		INC HL	; NEXT TABLE
180F	10 F7		DJNZ LOOP	
1811	18		JR START	
1820	11 00 40	DELAY:	LD DE, 4000H	
1823	1B	DELAY1:	DEC DE	
1824	7A		LD A,D	
1825	B3		OR E	
1826	20 FB		JR NZ , DELAY1	
1828	09		RET	
1900	01 02 04 08	TABLE		
1904	10 20 40 80			
1908	18 24 42 81			
190C	81 42 24 18			

เฉลยคำถาม

1. ในการเปลี่ยนความเร็วของไฟวิ่งสามารถจะทำได้โดยการเปลี่ยนค่าของ DELAY DATA ที่ ADDRESS 1821,1822 ซึ่งจากตัวอย่างกำหนดไว้ให้มีค่า 4000H แต่ถ้าต้องการเพิ่มความเร็วของไฟวิ่งก็ทำการลดค่าให้น้อยลงหรือถ้าต้องการให้ช้าลงก็ทำการเพิ่มค่า DELAY DATA นี้

2. ถ้าต้องการให้มีไฟวิ่งมากกว่า 8 ดวง สามารถทำได้โดยการเพิ่มวงจรอีกชุดหนึ่งดังรูป



การทดลองที่ 2

เรื่อง INPUT PORT

จุดประสงค์

1. เพื่อให้เข้าใจหลักการเบื้องต้นในการติดต่อระหว่างไมโครโปรเซสเซอร์กับอุปกรณ์ INPUT แบบง่าย
2. เพื่อให้สามารถออกแบบวงจร INPUT แบบง่ายได้
3. เพื่อให้สามารถเขียนโปรแกรมเพื่อรับข้อมูลจากอุปกรณ์ INPUT ที่ออกแบบได้

ทฤษฎี

ในการทดลองที่ผ่านมาได้กล่าวถึงขบวนการ OUTPUT และการใช้คำสั่งเกี่ยวกับ OUTPUT ซึ่งในการทดลองนี้จะกล่าวถึงขบวนการ INPUT และการใช้คำสั่งเกี่ยวกับ INPUT โดยอาศัยขบวนการ OUTPUT จากการทดลองที่ผ่านมาช่วยสนับสนุน

ขบวนการ INPUT เป็นขบวนการรับข้อมูลจากอุปกรณ์ภายนอกของ CPU โดยอุปกรณ์ภายนอกจะส่งข้อมูลผ่านทาง DATA BUS และคุณสมบัติของ INPUT PORT ที่สำคัญคือ อุปกรณ์ที่ทำหน้าที่เป็น INPUT PORT นั้น จะต้องไม่ส่งข้อมูล หรือสัญญาณใด,เข้าไปยัง DATA BUS จนกว่า CPU ต้องการ

คำสั่งที่ใช้เกี่ยวกับ INPUT นั้นได้แก่ IN,A (PORT) และคำสั่ง IN reg,(c) และคำสั่งในการส่งข้อมูลเข้าแบบเป็นกลุ่ม ซึ่งได้กล่าวรายละเอียดในการทดลองที่ผ่านมาแล้ว

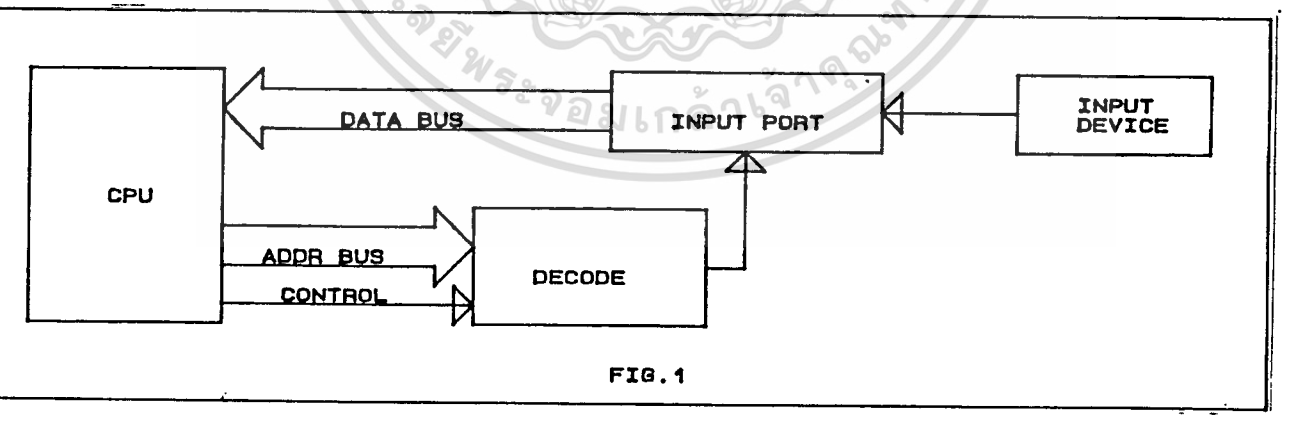


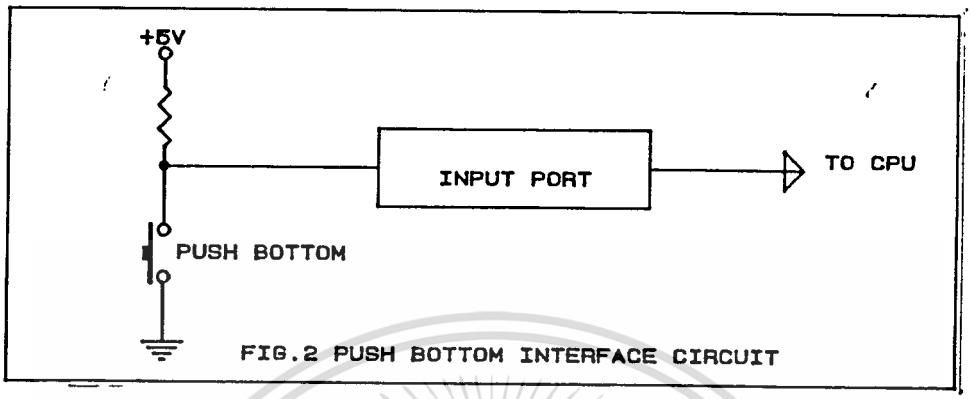
FIG.1

รูปที่ 1 ขบวนการ INPUT

จากรูป 1 จะพบว่า ลักษณะของ DIAGRAM จะคล้ายกันกับ DIAGRAM ในการติดต่อกับ OUTPUT ซึ่งกลไกในการรับข้อมูล ยังต้องเกี่ยวข้องกับ สัญญาณสำคัญ ๆ คือ สัญญาณ

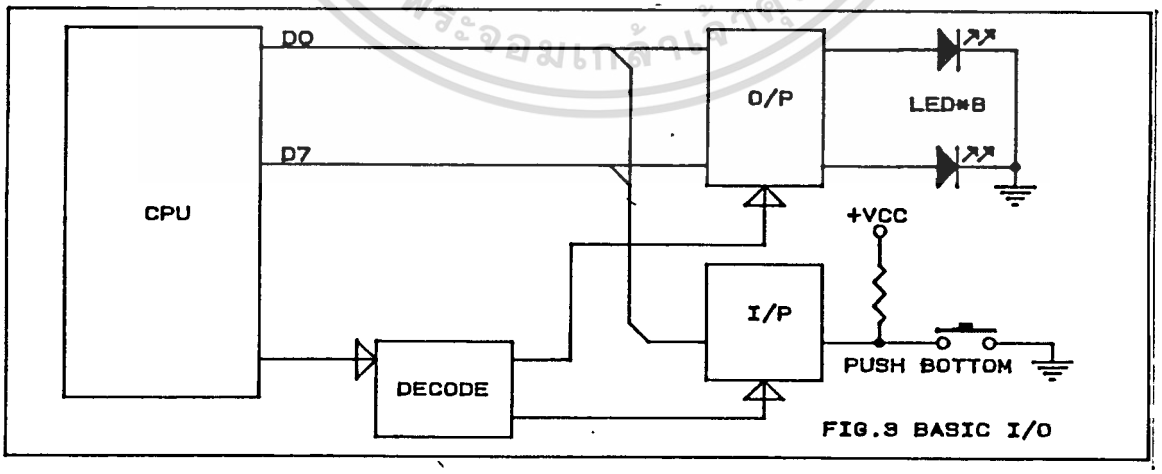
เลือก PORT, สัญญาณข้อมูลและสัญญาณควบคุม

ในการทดลองนี้จะเริ่มจากการ INTERFACE กับ PUSH BUTTON SWITCH ดังรูป



รูปที่ 2 PUH BUTTOM INTERFACE CIRCUIT

จากรูปที่ 2 แสดงการต่อ PUSH BUTTON SW.เข้ากับ INPUT PORT ซึ่ง PUSH BOTTON SW. เป็นอุปกรณ์ INPUT ประเภทความเร็วต่ำ ฉะนั้น INPUT PORT ที่เลือกใช้นี้จึงไม่จำเป็นจะต้อง LATCH ข้อมูลในการทดลองนี้เลือกใช้ 74L5245 ทำเป็น INPOT PORT ซึ่งเป็น IC OCTAL BUFFER การทำงานจะเป็นดังนี้ คือ เมื่อกด SW. ข้อมูล INPUT เป็น "0" และเมื่อปล่อยหรือยังไม่มีการกด ข้อมูลจะเป็น "1"



รูปที่ 3 BASIC I/O

อุปกรณ์การทดลอง

1. MPF-IB
2. PROBOARD พร้อม CONNECTOR
3. IC 74LS138 (DECODER) 1 ตัว
4. IC 74L5373 (OUTPUT PORT) 1 ตัว
5. IC 74LS245 (INPUT PORT) 1 ตัว
6. PUSH BOTTOM SW. 1 ตัว
7. DIP SWITCH 8 POSITION 1 ตัว
8. LED 8 ดวง
9. R 4.7K 1 ตัว
10. สายต่อวงจร และ POWER SUPPLY

เตรียมการทดลอง

1.ศึกษาทฤษฎีให้เข้าใจ

2.ศึกษาวิธีใช้ IC 74LS245 ซึ่งจะนำไปต่อเป็น INPUT PORT

3.เขียน WIRING DIAGRAM มีข้อกำหนดดังนี้

-ให้ OUTPUT PORT = C0H

-ให้ INPUT PORT = E0H

-ข้อมูลจาก INPUT PORT ต่อกัน D0 ขอ DATA BUS

ลำดับขั้นการทดลอง

1.ประกอบวงจรตามรูปที่ 4 โดยใช้ DATA SHEET IC 74138, 74373, 74245 ประกอบ

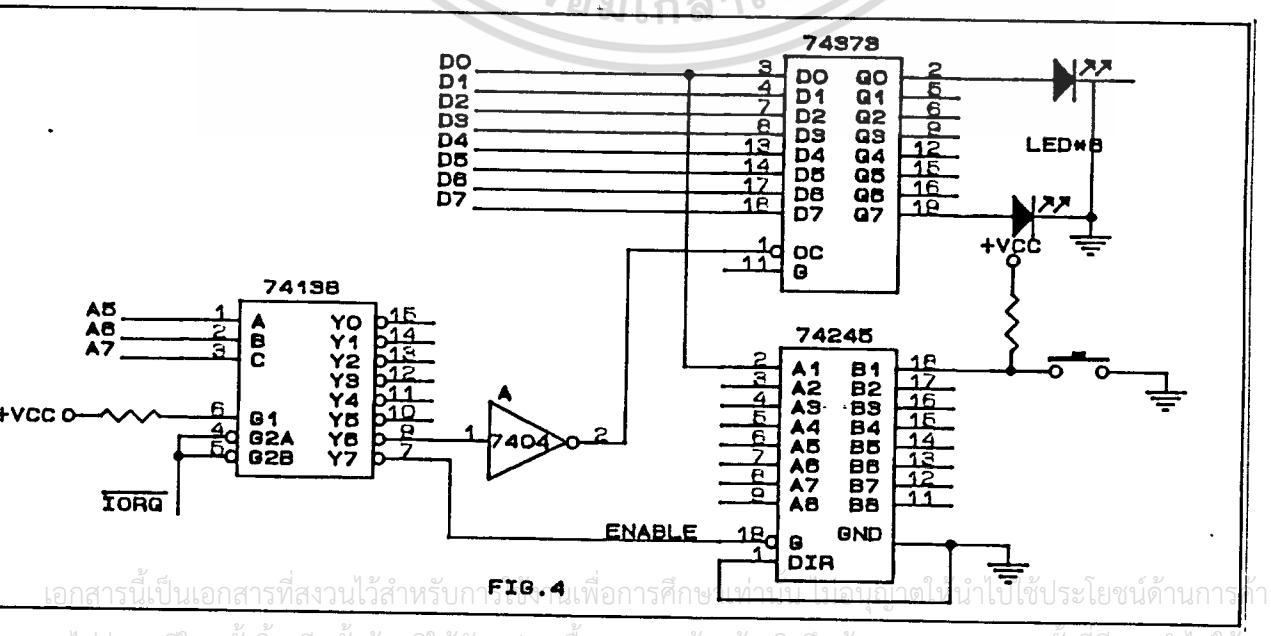


FIG.4

จงเขียนโปรแกรมให้ PUSH BOTTON ทำหน้าที่เป็น TOGGLE SW. ในการแสดงผลของ LED โดยให้รูปแบบของการแสดงผลอยู่ในรูปของ BINARY COUNTER ซึ่งในการกด PUSH BOTTON แต่ละครั้งจะเป็นการเปลี่ยนการทำงานของ COUNTER คือกลับไปกลับมาระหว่าง COUNT-UP กับ COUNT-DOWN

2.ประกอบวงจรตามรูปที่ 5

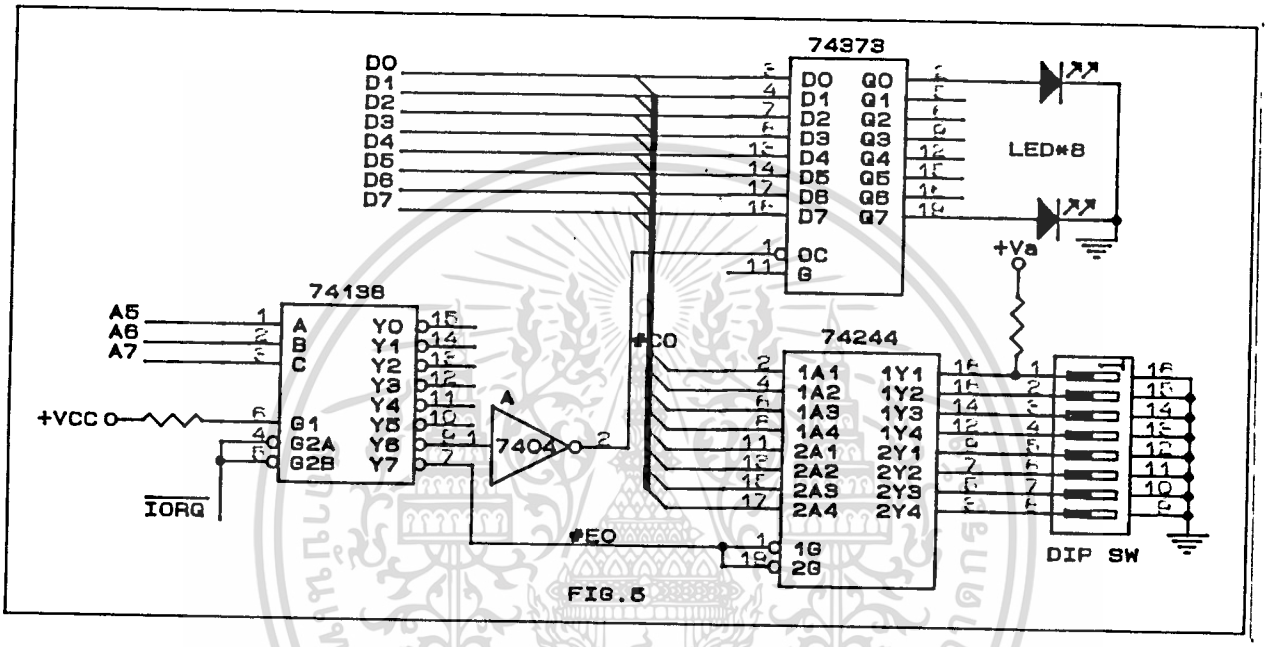


FIG.5

ทดสอบวงจรที่ต่อ ว่าถูกต้องหรือไม่โดยการป้อนโปรแกรมต่อไปนี้แล้วทดลอง EXECUTE โปรแกรม

```
1800    D3  E0    START : IN   A, (E0)
1802    D3  C0              OUT  (C0),A
1804    C3  00  18          JR   START
```

3.เขียนโปรแกรมโดยใช้ DIP SW. ตัวแรก (LSB) เพื่อควบคุมรูปแบบการติดดับของ LED ตามรูปข้างล่าง

```
0 0 0 0 0 0 0 *          LSB = 0
        <------
* 0 0 0 0 0 0 0          LSB = 1
------>
```

ถ้า LSB = "0" LED ติดวิ่งจาก ขวาไปซ้าย

ถ้า LSB = "1" LED ติดวิ่งจากซ้ายไปขวา

4.สรุปและวิจารณ์ผลการทดลอง

<u>คำถาม</u>

1.อธิบายข้อแตกต่างระหว่างขบวนการ INPUT กับ OUTPUT

2.ปัญหาของ SWITCH BOUNCE คืออะไร จงอธิบายและบอกวิธีการแก้ไข

ผลการทดลอง

โปรแกรมที่ทำให้ PUSH BOTTON ทำหน้าที่เป็น TOGGLE SW. ในการแสดงผลของ LED โดยให้รูปแบบของการแสดงผลอยู่ในรูปของ BINARY COUNTER ซึ่งในการกด PUSH BOTTON แต่ละครั้งจะเป็นการเปลี่ยนการทำงานของ COUNTER คือกลับไปกลับมาระหว่าง COUNT-UP กับ COUNT-DOWN

Address	Machine Code	Label	Operand	Comment
1800	1E 00		LD E,00	
1802	16 00		LD D,00	
1804	7A	MM :	LD A,D	
1805	D3 C0		OUT (C0),A	
1807	CD 50 18		CALL DELAY	
180A	DB E0		IN A,(E0)	
180C	CB 47		BIT 0,A	
180E	28 01		JR Z,KK	
1810	1C		INC E	
1811	CB 43	KK :	BIT 0,E	
1813	20 03		JR NZ,YY	
1815	14		INC D	
1816	18 EC		JR MM	
1818	15	YY :	DEC D	
1819	18 E9		JR MM	
1850	06 FF	DELAY:	LD B,FFH	
1852	1E FF	BB :	LD E,FFH	
1854	1D	DD :	DEC E	
1855	20 FD		JR NZ,DD	
1857	10 F9		DJNZ ,BB	
1859	C9		RET	

ตอบคำถาม

1. ข้อแตกต่างระหว่างขบวนการ INPUT กับ OUTPUT

-ขบวนการ INPUT เป็นขบวนการรับข้อมูลของ CPU โดยอุปกรณ์ภายนอกจะส่งข้อมูลผ่านทาง DATA BUS

-ขบวนการ OUT PUT เป็นขบวนการส่งข้อมูลจาก CPU ออกมาทาง DATA BUS ไปยังอุปกรณ์ภายนอก

2.ปัญหาของ SWITCH BOUNCE คือทำให้รับข้อมูลทาง INPUT ที่เป็น SWITCH ได้ข้อมูลที่ไม่แน่นอน ทั้งนี้เพราะช่วงที่เกิด BOUNCE ข้อมูลที่ได้รับอาจผิดพลาดได้ การแก้ไข

- ทาง HARDWARE โดยการเพิ่มวงจร MONOSTABLE เข้าไป
- ทาง SOFTWARE โดยการหน่วงเวลาให้รับ INPUT หลังจากเกิด BOUNCE แล้ว

การทดลองที่ 3

เรื่อง การใช้งาน 8255 เป็น INPUT และ OUTPUT PORT

จุดประสงค์

1. เพื่อให้สามารถใช้ 8255 PPI เป็น INPUT และ OUTPUT PORT ได้
2. เพื่อให้สามารถเขียนโปรแกรม เพื่อติดต่อกับ 8255 PPI ได้
3. เพื่อให้สามารถต่อวงจร DECODE PORT ได้

ทฤษฎี

8255 เป็น IC ที่มี 40 ขา ได้รับการออกแบบมาให้มีสัญญาณเพื่อเชื่อมต่อกับ 8080 แต่สัญญาณนี้ เหมาะที่จะใช้กับ Z-80 ได้ดี เช่น เดียวกัน 8255 เป็น IC ที่ต่อเป็น PORT ให้ ไมโครโปรเซสเซอร์ได้ 3 PORT โดยมีโครงสร้างพื้นฐาน ดังรูปที่ 1

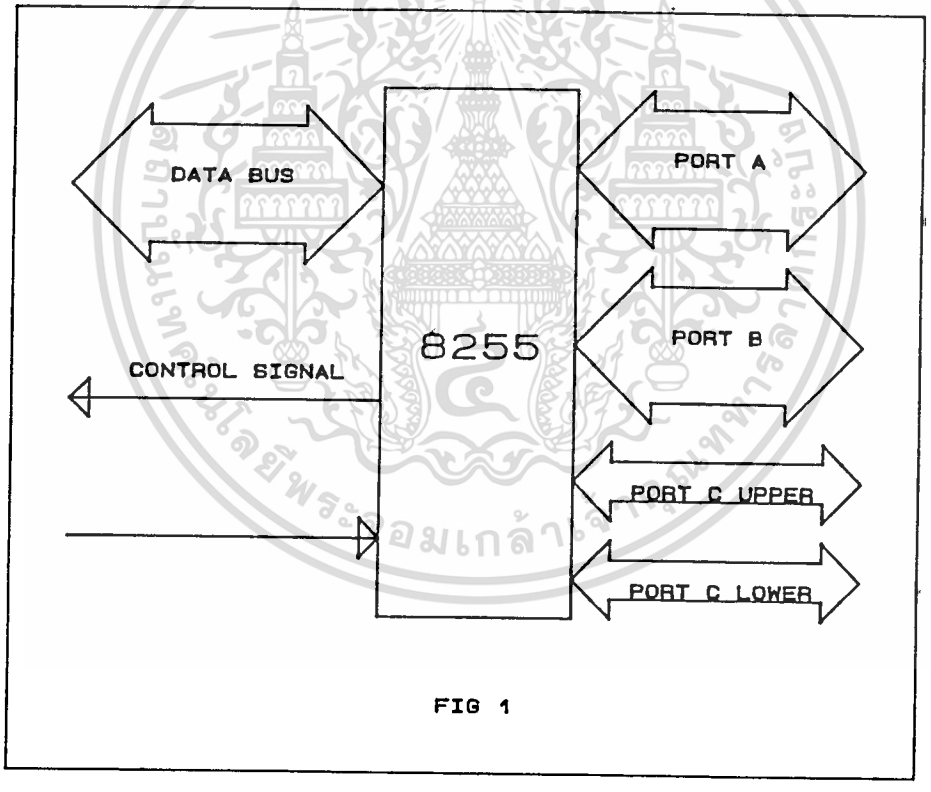


FIG 1

การเรียก PORT ของ 8255 จะเรียก PORT ต่าง ๆว่า PORT A, PORT B และ PORT C โดย PORT C แยกเป็น 2 ส่วนคือ PORT C ล่าง หรือตั้งแต่ PC0-PC3 มีจำนวน 4 บิท และ PORT C บน หรือตั้งแต่ PC4-PC7 ที่พิเศษคือ PORT ทุก PORT เป็นได้ทั้ง INPUT และ OUTPUT PORT

อุปกรณ์การทดลอง

1. MPF-IB & SUPPLY
2. PROTOBARD
3. IC 8255 PPI
4. IC 7404
5. RESISTOR ค่า 220 ohm x 16, 1k x 8, 2.7k
6. LED x 16

ลำดับขั้นการทดลอง

1.ประกอบวงจรตามรูปที่ 2

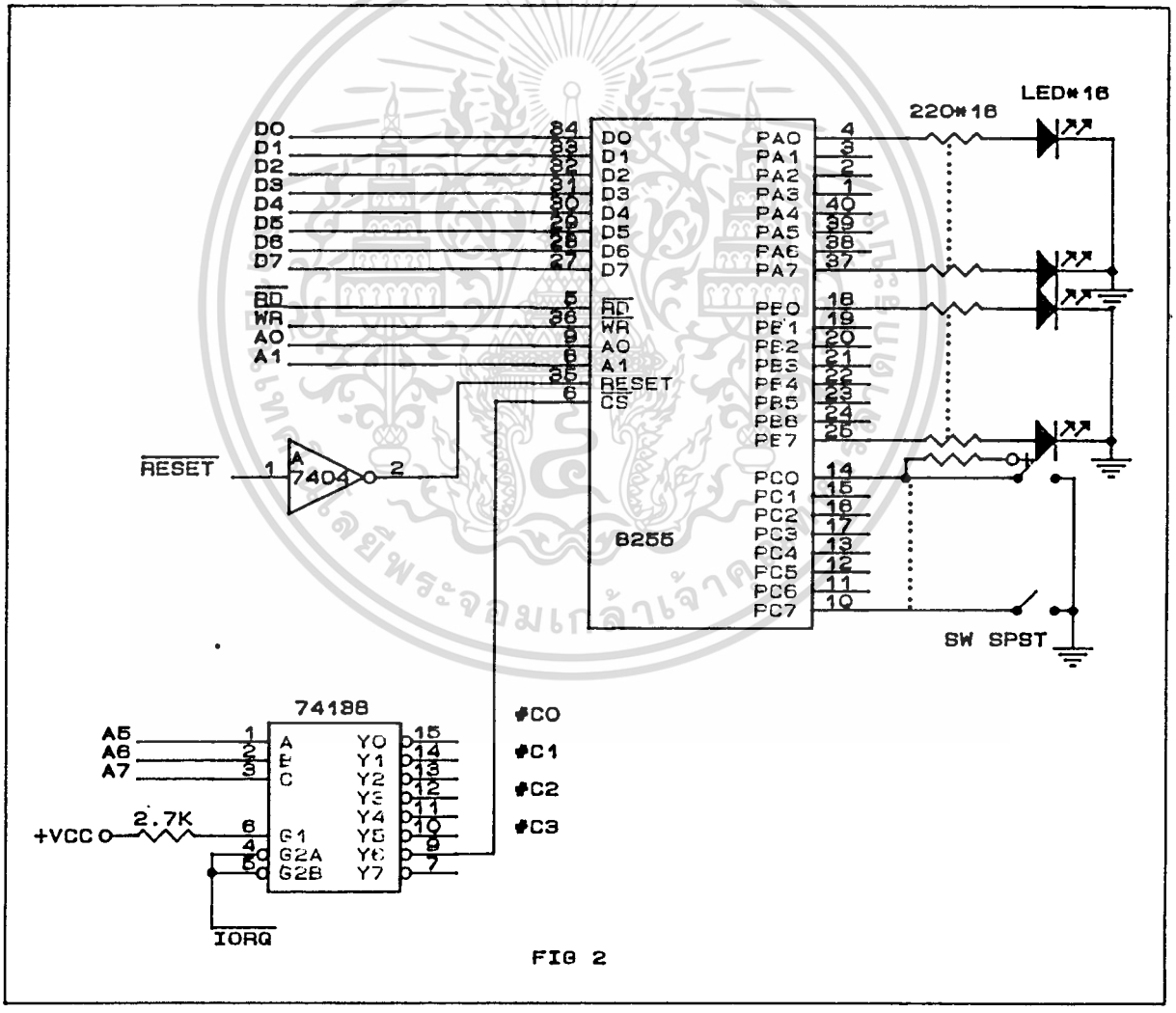


FIG 2

2.สั่ง CONTROL WORD เพื่อให้ 8255 ทำงานเป็น INPUT และ OUTPUT PORT ดังรูปที่ 2

PA0 - PA7	OUTPUT	PORT
PB0 - PB7	OUTPUT	PORT
DC0 - PC7	INPUT	PORT

3.เขียน FLOW CHART และ PROGRAM เพื่อให้ LED ติดกันตาม PATTERN ต่อไปนี้

3.1 จาก PORT A ไปยัง PORT B

PB7 PB0 PA7 PA0

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 *

<———————

4.เขียน PROGRAM เพื่อรับข้อมูล จาก DIP SWITCH ไปแสดงผลที่ PORT A และ PORT B พร้อมกัน

<u>คำถามท้ายการทดลอง</u>

1.เปรียบเทียบการใช้งานของ PORT ในการทดลองที่ 1 และ การทดลองที่ 2 กับ การใช้งาน 8255 ว่ามีข้อดีข้อเสียต่างกันอย่างไร

2.ในการใช้งาน 8255 ถ้าต้องการให้ 8255 เป็น PORT ดังข้างล่างนี้ จะต้องส่ง CONTROL WORD อย่างไร

PA0 - PA7	INPUT	PORT
PB0 - PB7	OUTPUT	PORT
PC0 - PC7	OUTPUT	PORT

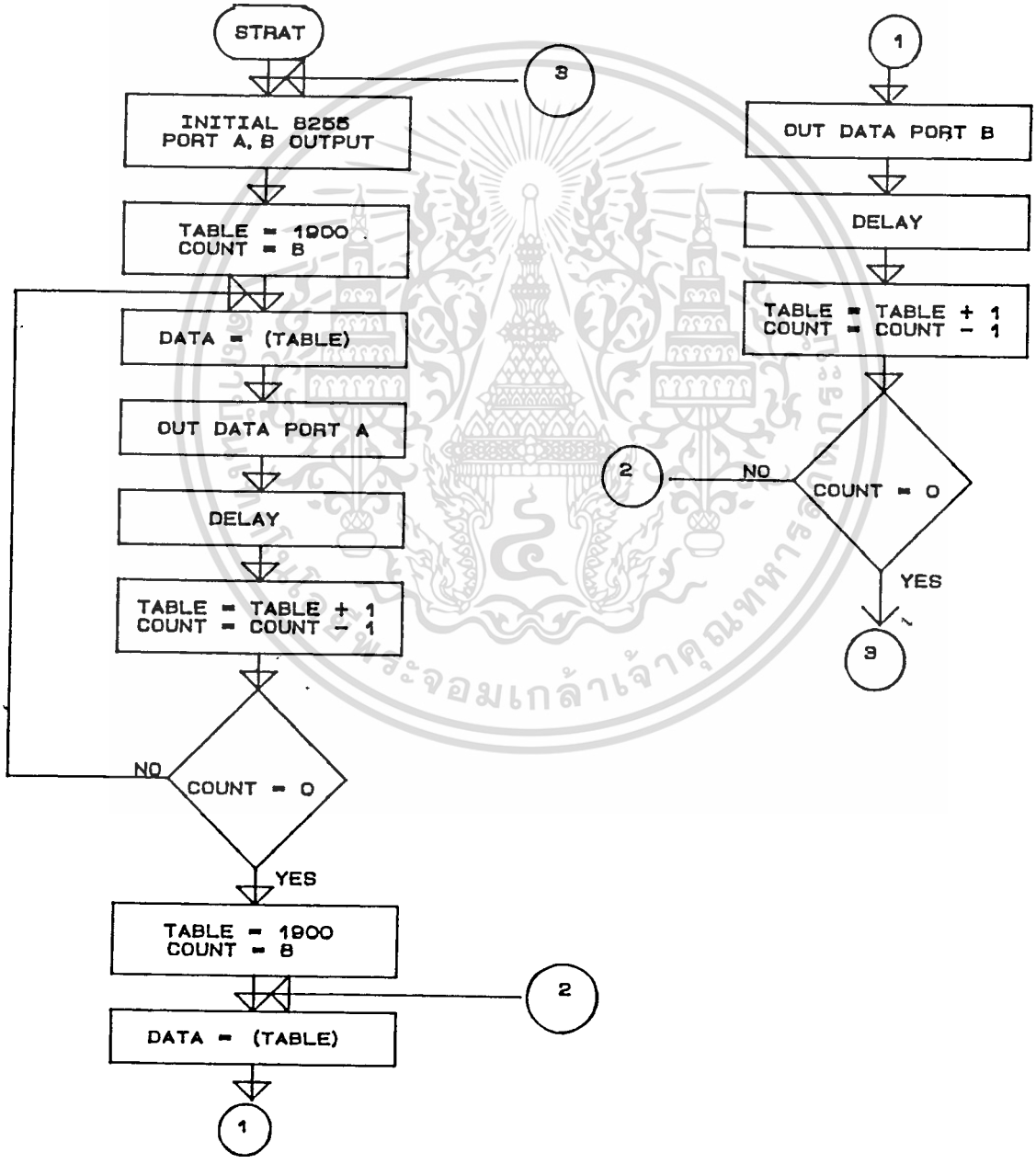
ผลการทดลอง

2. CONTROL WORD เพื่อให้ 8255 ทำงานเป็น INPUT และ OUTPUT PORT

PA0 - PA7		OUTPUT PORT
PB0 - PB7		OUTPUT PORT
PC0 - PC7		INPUT PORT

CONTROL WORD = 89

3. FLOW CHART



Address	Machine Code	Label	Operand	comment
1800	3E 89	START:	LD A, 89	;INITIAL 8255 PORT A,B
				;OUTPUT
1802	D3 C3		OUT (0C3),A	
1804	21 00 19		LD HL,1900H	;TABLE = 1900
1807	06 08		LD B,08H	;COUNTER =8
1809	7E	LOOP:	LD A, (HL)	
1804	D3 C0		OUT (0C0),A	;DISPLAY TO PORT A
180C	CD 30 18		CALL DELAY	
180F	23		INC HL	
1810	10 F7		DJNZ LOOP	
1812	21 00 19		LD HL, 1900H	
1815	06 08		LD B, 98H	
1817	7E	NEXT:	LD A,(HL)	
1818	D3 C1		OUT (0C1),A	
181A	CD 30 18		CALL DELAY	
181C	23		INC HL	
181D	10 F8		DJNZ NEXT	
181F	18 DF		JR START	
1830	11 00 40	DELAY:	LD DE, 4000H.	
1833	1B	DELAY1:	DEC DE	
1834	7A		LD A,D	
1835	B3		OR E	
1836	20 FB		JR NZ, DELAY 1	
1838	C9		RET	
1900	01 02 04 08	TABLE		
1904	10 20 40 80			

4.PROGRAM เพื่อรับข้อมูลจาก DIP SWITCH ไปแสดงผลที่ PORT A และ PORT B

Address	Machine Code	Label	Operand	Comment
1800	3E 89		LD A,89	;INITIAL 8255
1802	D3 C3		OUT (0C3),A	
1804	DB C2	LOOP:	IN A, (C2)	;GET DATA FROM DIP SW.
1806	D3 C0		OUT (C0),A	;OUT DATA TO PORT A
1808	D3 C1		OUT (C1),A	;OUT DATA TO PORT B
180A	18 F8		JR LOOP	

ตอบคำถามท้ายการทดลอง

1.ในการใช้งาน INPUT และ OUTPUT PORT ที่ใช้เกตลอจิก 3 สถานะหรือใช้ D FLIP FLOP ซึ่งจะใช้สัญญาณจากไมโครโปรเซสเซอร์ไปเปิดเกตให้ IC ทำงาน จุดอ่อนของการใช้งานในลักษณะนี้คือเรื่องของจำนวน IC ซึ่งอาจต้องใช้หลายตัวถ้าต้องการหลาย PORT และยากที่จะกำหนดลักษณะการทำงานให้แตกต่างไปจากวงจรเดิมที่ออกแบบไว้ 8255 เป็น LSI CHIP มีข้อดีกว่าที่ใช้งานง่าย มี PORT ทั้งหมดถึง 3 PORT สามารถเลือกรูปแบบการทำงานในแต่ละ PORT ให้เป็น INPUT หรือ OUTPUT ก็ได้ โดยการสั่งงานทาง SOFTWARE ซึ่งทำให้ง่ายต่อการเปลี่ยนแปลงวงจร และสะดวกในการใช้งาน

2.CONTROL WORD = 90

การทดลองที่ 4

เรื่อง SEVEN SEGMENT LED INTERFACING

จุดประสงค์

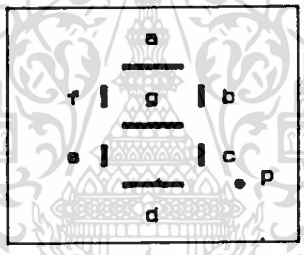
1.สามารถใช้งาน LED 7 SEGMENT เป็นอุปกรณ์ OUTPUT ได้

2.สามารถสร้าง CHARACTER GEN. เพื่อให้ LED 7 SEGMENT ติดเป็นอักขระตามต้องการ

3.สามารถเขียนโปรแกรมเพื่อควบคุมการทำงานของ LED 7 SEGMENT ได้

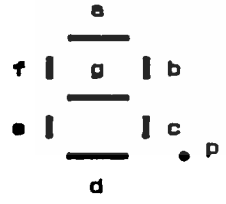
ทฤษฎี

LED 7 SEGMENT เป็นอุปกรณ์แสดงผลที่นิยมใช้มากสำหรับการแสดงผลของไมโครคอมพิวเตอร์ขนาดเล็กในระดับ SINGLE BOARD MICROCOMPUTER สาเหตุคงเนื่องมาจากใช้งานง่าย ราคาถูก ง่ายต่อการออกแบบ มักจะถูกใช้ในการแสดงผลในลักษณะตัวเลขหรือตัวอักขระแบบง่าย มีให้เลือกใช้ทั้งแบบ COMMON ANODE และ COMMON CATHODE โครงสร้างเป็นดังรูป



วิธีการสสร้าง CHARACTER GEN ให้กับ LED 7 SEGMENT

การต่อ DATA BUS กับ LED 7 SEGMENT



D7	D6	D5	D4	D3	D2	D1	D0
p	g	f	e	d	c	b	a

ตัวอย่าง ถ้าต้องการให้ LED 7 SEGMENT (COMMON CATHODE) ติดเป็นเลข 3 จะต้องป้อน DATA ดังนี้

อุปกรณ์การทดลอง

1.MPF-IB & SUPPLY

2.PROTOTBOARD

3.IC 74LS373, 74244

4.IC 7404, IC 74138

5.DIP SWITCH

6.LED 7 SEGMENT COMMON CATHODE

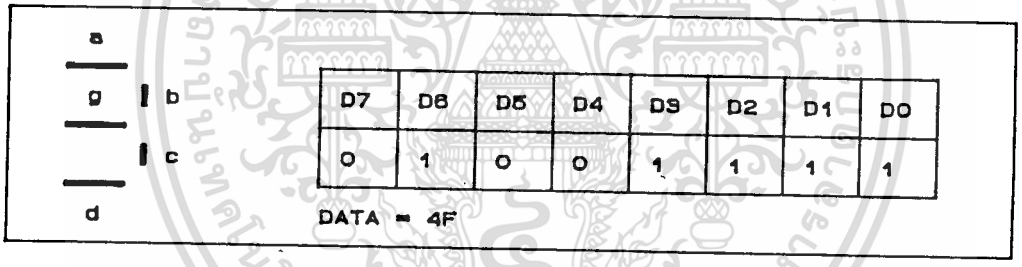
7.CONNECTOR พร้อมสาย PAIR

8.RESISTOR 1K

ลำดับขั้นการทดลอง

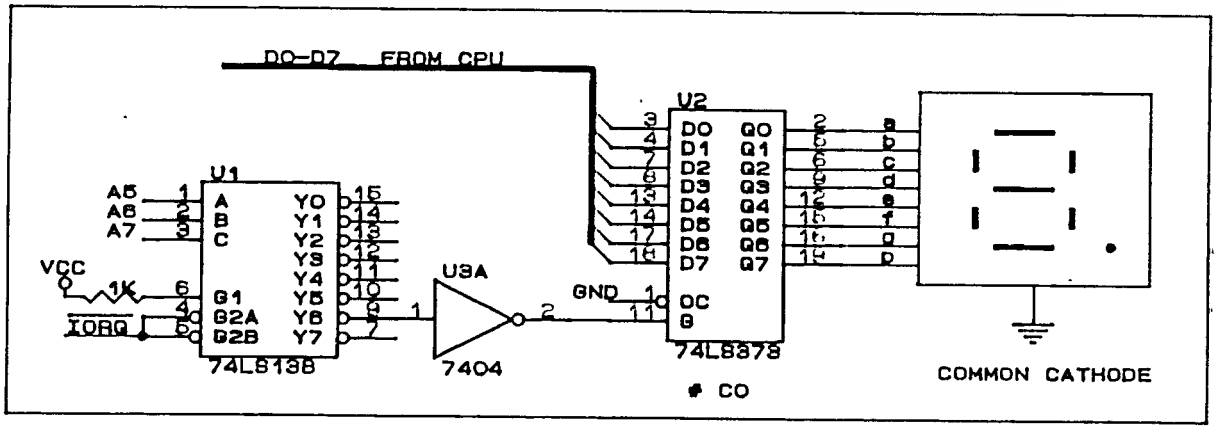
1.ให้นักศึกษาสร้าง CHARACTER GEN สำหรับ DATA ทุกตัวตั้งแต่ 0-F

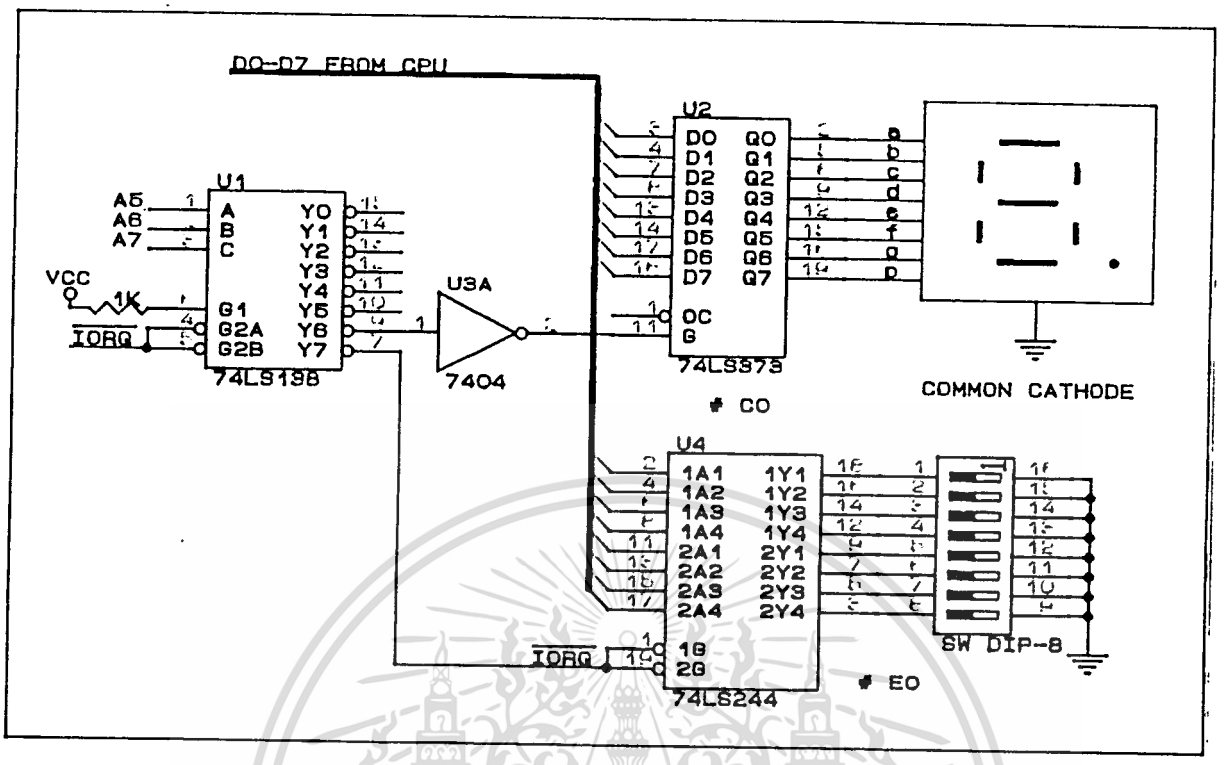
2.ประกอบวงจร ตามรูปที่ 1



3.จงเขียนโปรแกรมเพื่อให้ LED 7 SEGMENT แสดงตัวเลข ตั้งแต่ 0-F

4.ประกอบวงจรเพิ่มเติมตามรูปที่ 2





5.เขียน PROGRAM รับข้อมูลจาก DIP SWITCH แล้วแปลงข้อมูลที่รับได้จาก DIP SW. ให้ไปแสดงผลที่ 7 SEGMENT นั้น คือให้คอมพิวเตอร์แสดงตัวเป็น IC 7448 (BCD TO SEVEN SEGMENT DECODER)

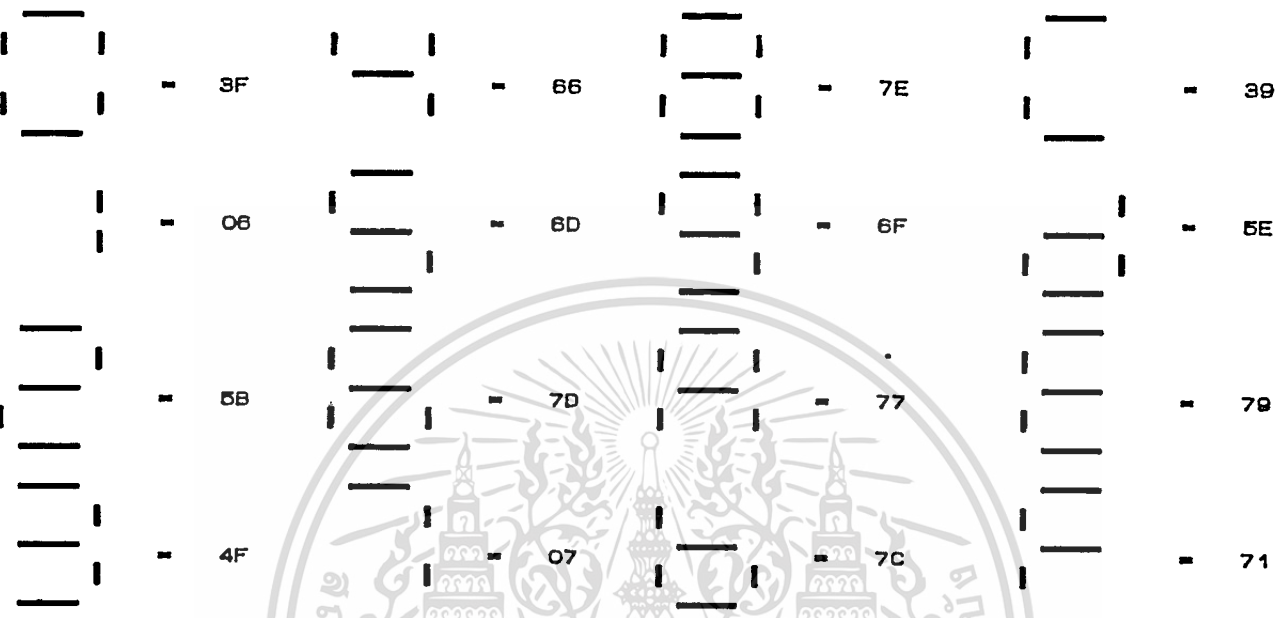
<u>คำถามท้ายการทดลอง</u>

1.ในกรณีที่ LED 7 SEGMENT ที่ใช้เป็นแบบ COMMON ANODE จะต้องทำการเปลี่ยนแปลงหรือแก้ไขอะไรบ้าง อย่างไร

ผลการทดลอง

การสร้าง CHARACTER GEN

CHARACTER GEN.



3.โปรแกรมเพื่อให้ LED 7 SEGMENT แสดง ตัวเลขตั้งแต่ 0-F

Address	Machine Code	Label	Operand	Comment
1800	21 00 s19	START:	LD HL,1900	;START TABLE
1803	06 10		LD B, 10	
1805	7E	LOOP:	LDA, (HL)	; GET DATA FROM TABLE
1806	D3 C0		OUT (C0),A	;DISPLAY DATA
1808	CD 20 18		CALL DELAY	
180B	23		INC HL	;NEXT DATA
180C	10 F7		DJNZ LOOP	
180E	18 F0		JR START	
1820	11 00 40	DELAY:	LD DE, 4000H	
1823	1B	DELAY1:	DEC DE	
1824	7A		LD A, D	
1825	B3		OR E	

```
1826    20 FB                   JR NZ, DELAY1
1828    C9                      RET

1900    3F 06 5B 4F    TABLE
1904    66 6D 7D 07
1908    7E 6F 77 7C
190C    39 5E 79 71
```

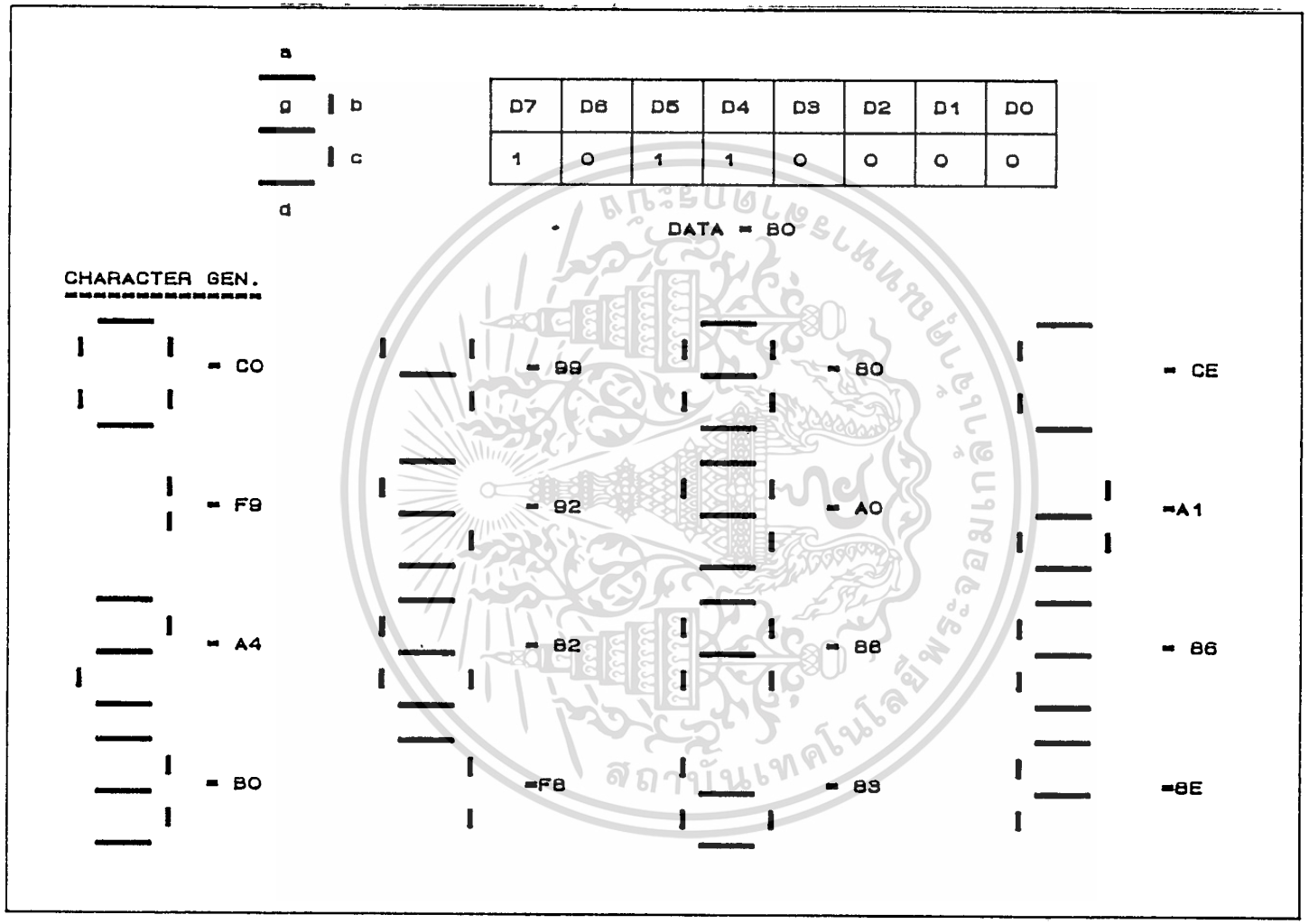
5.PROGRAM รับข้อมูลจาก DIP SW. แล้วแปรข้อมูลที่รับได้จาก DIP SW. ให้ไปแสดงผลที่ 7 SEGMENT

```
Address  Machine Code  Label    Operand        Comment
1800     21 00 19      START:   LD HL, 1900H   ;START OF TABLE
1803     16 00                  LD D,00
1805     DB E0                  IN A, (E0)     ;READ DATA FROM SW.
1807     5F                     LD E,A
1808     19                     ADD HL,DE

1809     7E                     LD A, (HL)     ;GET DATA FROM TABLE
180A     D3 C0                  OUT (C0), A    ;DISPLAY DATA
180C     18 F2                  JR START

1900     3F 06 5B 4F
1904     66 6D 7D 07
1908     7E 6F 77 7C
190C     39 5E 79 71
```

ตอบคำถาม

1.ในกรณีที่ LED 7 SEGMENT ที่ใช้เป็นแบบ COMMON ANODE จะต้องทำการเปลี่ยน CHARACTER GEN ใหม่ ดังนี้

เช่น ถ้าต้องการให้ LED ติดเป็นเลข 3 ต้องมี DATA ดังนี้


a
g
b
c
d
D7 D6 D5 D4 D3 D2 D1 D0
1 0 1 1 0 0 0 0
DATA = B0
CHARACTER GEN.
= C0
= F9
= A4
= B0
= 99
= 92
= 82
=F8
= 80
= A0
= 88
= 83
= CE
=A1
= 86
=8E

การทดลองที่ 5

การทดลองเรื่อง การใช้งาน LED 7 SEGMENT เป็นอุปกรณ์เอาท์พุท (ต่อ)

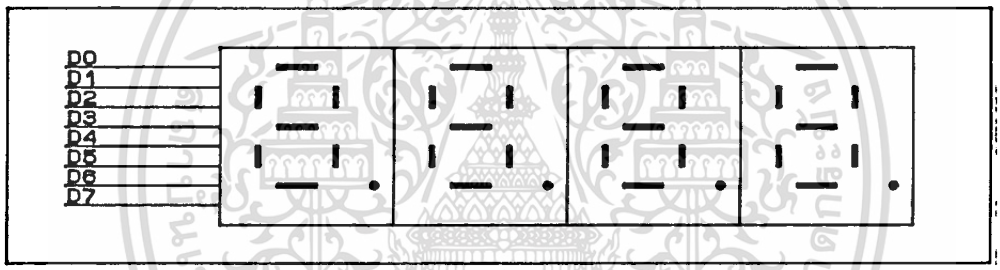
วัตถุประสงค์

1. เพื่อให้สามารถออกแบบ HARDWARE ในการใช้งาน 7 SEGMENT หลาย ๆ ตัวได้

2. เพื่อให้สามารถเขียนโปรแกรมเพื่อ SCAN DISPLAY ที่เป็น 7 SEGMENT ได้

ทฤษฎีก่อนการทดลอง

ในการต่อ LED 7 SEGMENT หลาย ๆ ตัว เป็นเอาท์พุทพอร์ท การต่อจะต่อ SEGMENT ของ แต่ละ DIGIT ร่วมกัน นั่นคือ SEGMENT A ของ DIGIT 1 จะต่อกับ SEGMENT A ของ DIGIT 2,3, และ 4 และการต่อจะเป็นเช่นเดียวกันนี้ทุก ๆ SEGMENT ของทุก DIGIT ดังรูป



การที่จะทำให้ DIGIT ไหนติดสามารถทำได้โดยการให้ COMMON ของ DIGIT นั้นเป็น LOGIC 0 ดังรูป

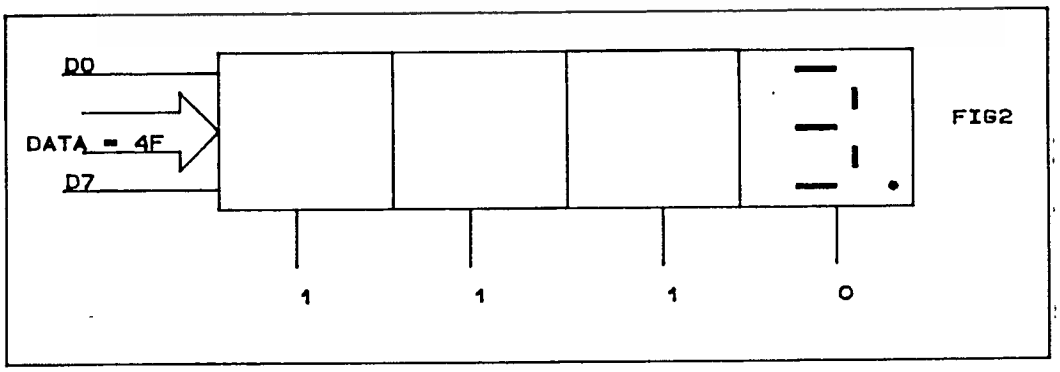


FIG2

อุปกรณ์การทดลอง

1. MPF-IB & SUPPLY
2. PROTOBOARD
3. IC 74373 , 7404 , 74138
4. LED 7 SEGMENT
5. CONNECTOR พร้อมสาย PAIR
6. RESISTOR

ลำดับขั้นการทดลอง

1. ประกอบวงจรตามรูปที่ 3

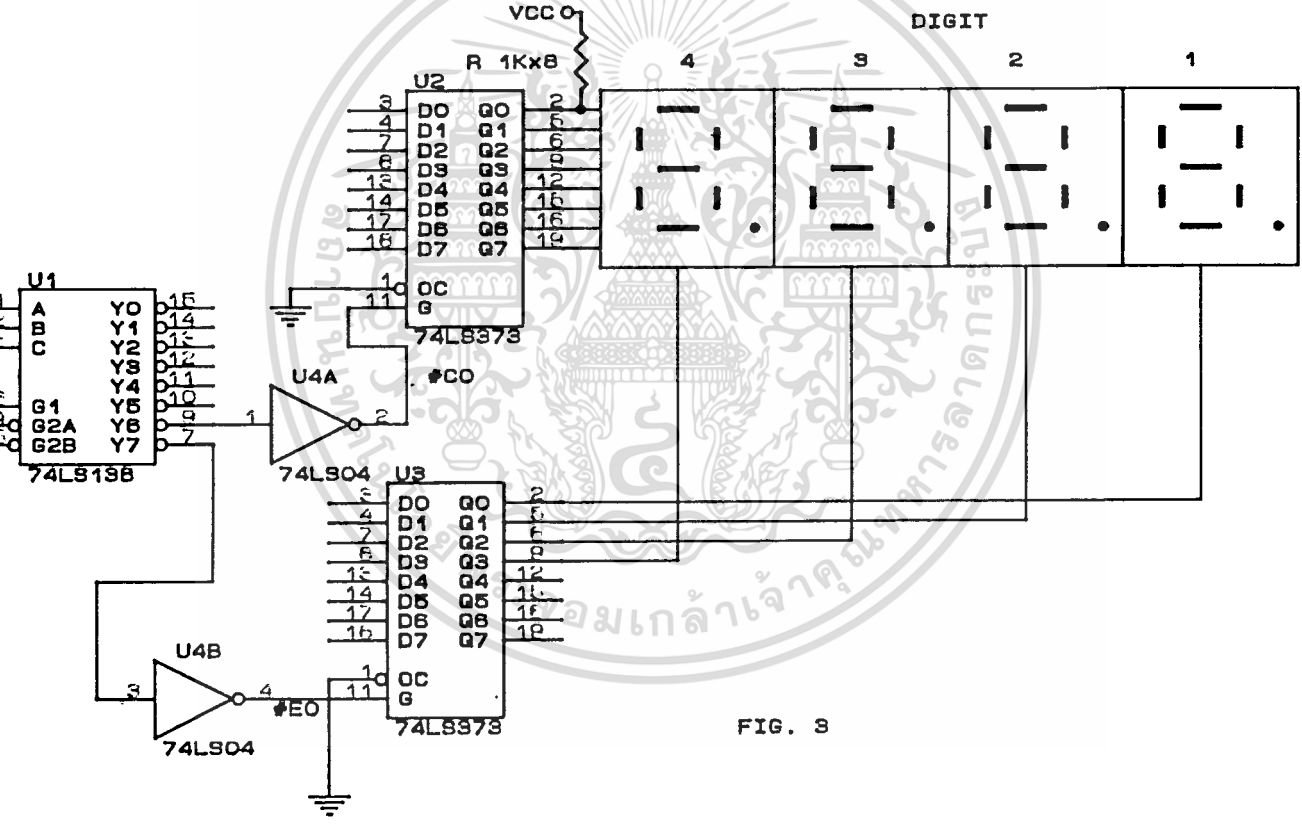
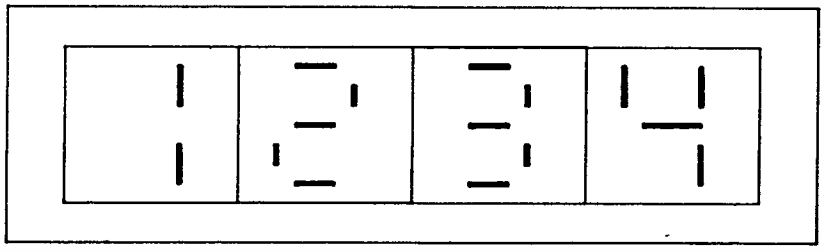


FIG. 3

2.จงเขียน FLOWCHART และ PROGRAM เพื่อทำให้ 7 SEGMENT แสดงผลเป็น ดังรูป



Address	Machine Code	Label	Operand	Comment
1800	21 00 19	START:	LD HL, 1900H	
1803	0E FF		LD C, FE	
1805	06 04		LD B,04	
1807	7E	LOOP:	LD A, (HL)	
1808	D3 C0		OUT (C0),A	
180A	79		LD A,C	
180B	D3 E0		OUT (E0),A	
180D	CD 30 18		CALL DELAY	
1810	CB 01		RLC C	
1812	3E 00		LD A, 00	
1814	D3 C0		OUT (C0),A	
1816	3E FF		LD A, FF	
1818	D3 E0		OUT (E0),A	
1814	23		INC HL	
181B	10 EA		DJNZ LOOP	
181D	18 E1		JR START	
1830	11 00 40	DELAY:	LD DE, 4000H	
1833	1B	DELAY1:	DEC DE	
1834	7A		LD A,D	
1835	B3		OR E	
1836	20 FB		JR NZ, DELAY1	

1838 C9 RET

1900 06 5B 4F 66

การทดลองที่ 6

เรื่อง Z-80 INTERRUPT

จุดประสงค์

1. เข้าใจวิธีการในการ INTERRUPT แบบต่าง ๆ
2. สามารถนำเทคนิคการ INTERRUPT ไปใช้งานได้

ทฤษฎี

การ INTERRUPT เป็นการขัดจังหวะการทำงานของ CPU เพื่อให้ CPU กระโดดไปทำงานในตำแหน่งที่ต้องการได้ โดยอาศัยสัญญาณขอขัดจังหวะ ซึ่งเป็นสัญญาณที่ CPU ได้รับจากอุปกรณ์ภายนอก เมื่อ CPU ได้รับสัญญาณ INTRRUPT และทำการส่งสัญญาณตอบรับการ INTERRUPT สัญญาณที่ CPU ตอบรับการ INTERRUPT นี้เรียกว่า INTERRUPT ACKNOWLEDGE ก่อนที่ CPU จะกระโดดไปทำคำสั่งในตำแหน่งที่ถูกกำหนดจากการ INTERRUPT CPU จะเก็บค่าของ PROGRAM COUNTER (PC) ขณะนั้นเอาไว้ใน STACK ก่อน ซึ่งจะทำให้ CPU สามารถกลับมาทำงานต่อจากเดิมก่อนได้รับการ INTERRUPT ได้ถูกต้อง ส่วน PROGRAM ที่ CPU กระโดดไปทำงาน เมื่อได้รับสัญญาณ INTERRUPT นั้นเรียกว่า INTERRUPT SERVICE ROUTINE

ประเภทของสัญญาณ INTERRUPT

1.NON-MASKABLE INTERRUPT เป็นสัญญาณ INTERRUPT ที่สามารถขัดจังหวะการทำงานของ CPU ได้โดยไม่มีข้อแม้ นั่นคือ เมื่อ CPU ได้รับสัญญาณ INTERRUPT ชนิดนี้แล้ว CPU จะต้องตอบสนองสัญญาณนี้ทุกครั้ง โดยการกระโดดไปทำงานที่ ADDRESS 0066H

2.MASKABLE INTERRUPT เป็นสัญญาณ INTERRUPT อีกแบบหนึ่งที่สามารถขัดจังหวะการทำงานของ CPU ได้ แต่มีข้อแม้ว่า CPU ในขณะนั้นจะต้องอยู่ในสภาวะที่ยอมรับการ INTERRUPT (ENABLE INTERRUPT) หาก CPU อยู่ในสภาวะไม่ยอมรับการ INTERRUPT (DISABLE INTERRUPT) สัญญาณ INTERRUPT ชนิดนี้จะไม่ได้รับการตอบสนองจาก CPU ในการกำหนดให้ CPU ยอมรับหรือไม่ยอมรับสัญญาณ INTERRUPT สามารถสั่งได้โดย SOFTWARE คือคำสั่ง ET (ENABLE INTERRUPT) หรือ DI (DISABLE INTERRUPT)

เตรียมการทดลอง

1.ศึกษาทฤษฎีของการ INTERRUPT ของ Z-80 CPU

2.เขียน MAIN PROGRAM และ INTERRUPT SERVICE ROUTINE จากปัญหาที่กำหนดในการทดลอง

3.ศึกษาวงจรการ INTERRUPT ของ MPF-IB

ใน MPF-IB นั้นสัญญาณ INTERRUPT ประเภท NON-MASKABLE จะถูกใช้โดย MONITOR PROGRAM ดังนั้นในการทดลองนี้ จะทดลองการ INTERRUPT กับสัญญาณ INTERRUPT ประเภท MASKABLE INTERRUPT ซึ่งบน BOARD มี KEY INIR ต่ออยู่กับขา 16 (ดูจากวงจร) ใน MONITOR PROGRAM ตำแหน่ง 0038H ซึ่งเป็น INTERRUPT SERVICE ROUTINE ที่ CPU จะกระโดดมาทำงานเมื่อ

1.CPU ตอบรับการ INTERRUPT MODE 1

2.CPU กระทำคำสั่ง RST 38H

3.CPU ตอบรับการ INTERRUPT MODE 0 และมี INTERRUPT VECTTOR เป็น RST 38H โดยปกติข้อมูลบน DATA BUS จะเป็น FFH

รายละเอียดของ MONITOR PROGRAM ในตำแหน่ง 0038H

```
LOC     OBJ     CODE            SOURCE      STAMENT
0038                    RST38   ORG     38H
                        ;Entry point of RST 38H (opcde FF) or
                        ;mode 1 interrupt. Fetch the address
                        ;stored in location 1FEE  and  1FEF,
                        ;then jump to this address. Initially,
                        ;1FEE and 1FEF are set to 0066. So
                        ;RST 38 will have the same effect as
                        ;software break.  Fy changing the
                        ;content of 1FEE and 1FEF, the user
                        ; can define his or her own service
                        ; routine.
                        ;The next three  instructions  push
                        ;the countents of 1FEE and 1FEF to
                        ;stack without changing any registers.

0038    E5                  PUSH    HL
0039    2AEE1F              LD      HL, (IM1AD)
003C    E3                  EX      (SP),HL
```

```
                    ;The top of the stack in now the
                    ;address of user defined service
                    ;routine.  Pop out this address then
                    ;branch to it.

003D    C9          RET
```

จาก PROGRAM (1M1AD) จะถูกกำหนดเป็นตำแหน่ง 1FEE และ 1FEE ดังนั้นคำสั่ง LD HL, (1M1D) จึงเป็นการนำข้อมูลตำแหน่ง 1FEE และ 1FEF ไปเก็บไว้ใน REGISTER PAIR HL เราจะพบว่า PROGRAM นี้ทั้งหมด ทำหน้าที่เพียงนำเอาข้อมูลที่อยู่ในตำแหน่ง 1FEE และ 1FEF เป็นตำแหน่งที่จะกระโดดไปทำงาน ซึ่งโดยปกติ ข้อมูลในตำแหน่ง 1FEE และ 1FEF จะมีค่าเป็น 00 และ 66 ซึ่งเป็นตำแหน่งของ PROGRAM INTERRUPT ประเภท NON-MASKABLE ดังนี้เมื่อเราเปลี่ยนข้อมูลในตำแหน่งทั้ง 2 นี้ เป็นข้อมูลตำแหน่ง INTERRUPT SERVICE ROUTINE ของเราเอง CPU ก็จะสามารถกระโดดไปทำงานในก INTERRUPT SERVICE ROUTINE ของเราได้ เมื่อได้รับการ INTERRUPT ดังกล่าว

ในการทดลองนี้จะทำการทดลอง INTERRUPT ใน MODE 1 โดยอาศัย KEY INTERRUPT ที่มีอยู่

<u>ข้อควรระวังในการเขียนโปรแกรม</u>

1. จะต้องทำการ SET MODE และทำการ ENABLE การ INTERRUPT ใน MAIN PROGRAM
2. ต้องทำการเก็บค่าของ REGISTER ต่าง ๆ ที่ใช้ใน MAIN PROGRAM ซึ่งจะต้องกำหนดไว้ตอนต้นของ INTERRUPT SERVICE ROUTINE

<u>ลำดับขั้นการทดลอง</u>

1. ศึกษาทฤษฎีและคำอธิบายการทดลองให้ละเอียด
2. เขียนโปรแกรมดังปัญหาต่อไปนี้

ปัญหา....ต้องการให้แสดงผลที่ SEGMENT ของ MPF-IB ดังรูปที่ 1 และเปลี่ยนรูปแบบของการแสดงผลเป็นดังรูปที่ 2 เมื่อได้รับสัญญาณ INTERRUPT และจะกลับไปแสดงผลดังรูปที่ 1 อีกครั้งหนึ่งเมื่อได้รับการ INTERRUPT ในครั้งต่อมา กล่าวคือ จะทำการแสดงผลสลับกันไปมาดังรูปแบบทั้ง 2 ทุกครั้ง เมื่อได้รับการ INTERRUPT และก่อนที่จะมีการเปลี่ยนรูปแบบของการแสดงผลทุกครั้งให้มีการสร้างเสียงขึ้นทุกครั้ง ตามความถี่ที่เหมาะสม

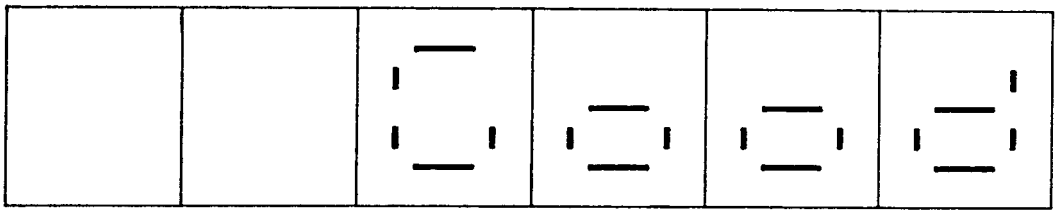


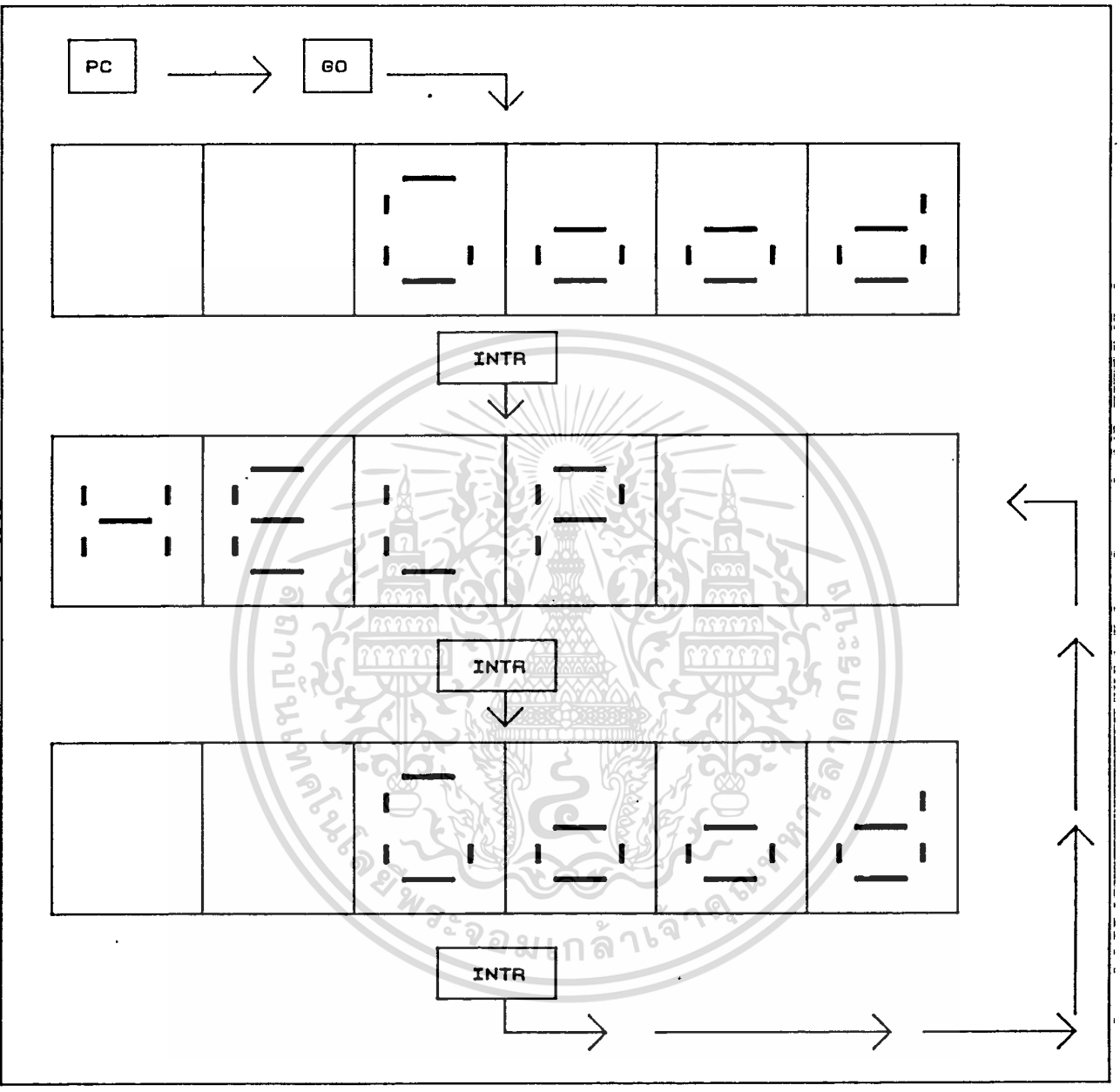
FIG.1

FIG.2

สรุปและวิจารณ์ผลการทดลอง

```
Address  Machine Code     Label    Operand            Comment
1800     ED 56            START :  IM 1               ;SET INTERRUPT MODE 1
1802     21 EE AF                  LD HL, 1FEEH       ;SET INTRRUPT SERVICE
1805     2A 50 18                  LD (HL), 1850H     ;ROUTINE ADDRESS
1808     DD 21 00 19               LD IX, 1900H
180C     F3               LOOP  :  DI
180D     CD 24 06                  CALL SCAN1
1810     FB                        EI
1811     18 F9                     JR,  LOOP

1850     3E 00                     LD A, 00
1852     D3 01                     OUT (01), A
1854     0C 0A                     LD  C, 0A          ;SET FREQUENCY
1856     21 C0 00                  LD HL, 0C0H        ;LENGTH OF FREQUENCY
1859     CD E4                     CALL TONE
185C     FB                        EI
185D     21 20 19                  LD HL, 1920H
1860     34                        INC (HL)
1861     CB 46                     BIT  0,(HL)
1863     28 05                     JR   Z, PATT2
1865     DD 21 10 19               LD  IX, 1010H
1869     C9                        RET
186A     DD 21 00 19      PATT2 :  LD IX, 1900H
186E     C9                        RET

1900     B3 A3 A3 AD 00 00
1910     00 00 1F 85 8F 37
1920     00
```

ผลการทดลอง



สรุปผลการทดลอง

การทดลองชุดนี้ เราสามารถเลือก MODE การทำงานโดยใช้ คำสั่ง IM1 (INTERRUPT MODE1) หลังจากที่มีการ INTERRUPT แล้ว Z-80 จะทำงานตามขั้นตอนดังนี้

1.สถานะของ IFF1 ถูกปรับให้เป็น "0" ซึ่งเป็นการทำให้ Z-80 DISABLE ต่อการขอ INTERRUPT ที่ขา $\overline{\text{INT}}$

2.สถานะของ 7FF2 ถูกปรับให้เป็นลอจิก "0"

3.ค่าของ PC ถูกเก็บไว้บน STACK

4.Z-80 จะกระโดดไปทำงานที่ ADDR 0038H

การกอนสัญญาณ INTERRUPT ที่ขา INT ของ Z-80 ซึ่งต้องทำก่อนที่ Z-80 จะกลับเข้าสู่การทำงานใน MAIN PROGRAM ด้วยคำสั่ง EI

การทดลองที่ 7

เรื่อง STEPPING MOTOR

จุดประสงค์

1. เพื่อให้นักศึกษาสามารถเข้าใจการทำงานของ STEPPING MOTOR

2. เพื่อให้นักศึกษาสามารถเขียนโปรแกรมเพื่อควบคุมการทำงานของ STEPPING MOTOR ได้

ทฤษฎี

Stepping motor เป็นมอเตอร์ไฟฟ้าแบบพิเศษ ซึ่งเมื่อป้อนไฟ DC ให้กับขด Stator จะเกิดแรงผลักต่อ rotor ด้วย electromagnetic force จะทำให้มอเตอร์หมุนตามค่าของ step angle บางครั้งเราจึงเรียกมอเตอร์แบบนี้ว่า Pulse motor Stepping motor แบบที่ใช้กันส่วนมากมีอยู่ 3 ชนิด คือ

1. แบบ Variable Reluctance (VR) ลักษณะของ rotor เป็นแบบ gear-shaped และทำด้วยเหล็กอ่อน

2. แบบ Permanent Magnel (PM) rotor ทำจาก permanent magnet จึงทำให้เกิด holding torgue แม้ว่าจะไม่มีการ excitation

3. แบบ hybrid เป็นแบบที่เกิดจากการรวม 2 ลักษณะแรกเข้าด้วยกันบางครั้งเราเรียกแบบนี้ว่า Step-Sync ลักษณะกราฟความสัมพันธ์ระหว่าง Torque กับความถี่ (position per second) เป็นดังรูปที่ 1

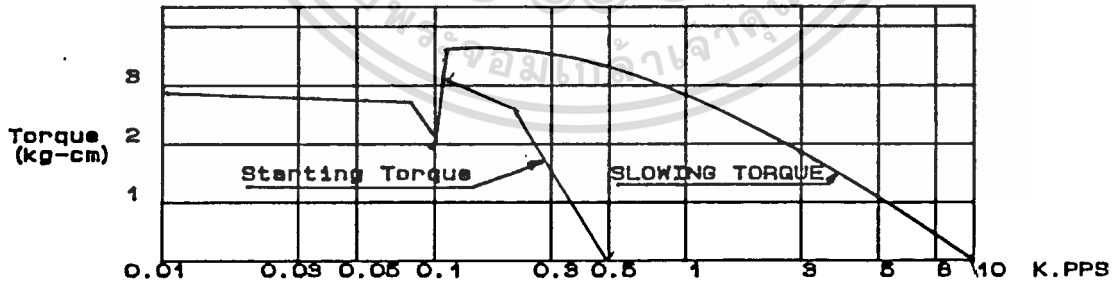


FIG 1

รูปที่ 1 กราฟแสดงความสัมพันธ์ระหว่าง TOURQE กับความถี่

Stepping motor เปรียบเสมือน electromachanical transducer ซึ่งมี input เป็นกลุ่มของ binary veltage และ output เป็นลักษณะของการเคลื่อนที่ในเชิง

มุม เป็น step ด้วยลักษณะการเคลื่อนที่ดังกล่าว stepping motor จึงได้รับการนำมาประยุกต์ใช้งานต่าง เช่น ใช้เป็นตัว carriage feed และ space feed ใน line printer, X-Y plotter, numerically controlled machine tool drives จนกระทั่งถึง robot โดยขนาดของ step อยู่ในช่วงตั้งแต่ 0.1 ถึง 30 องศา ซึ่งโครงสร้างของ stepping motor ยังมีลักษณะแตกต่างกันออกไปหลายชนิด ดังนี้คือ

1.แบบ solenoid-rachet

2.แบบ variable-reluctance (VR)

3.แบบ permanemt-magnet (PM)

4.แบบ hybrid (synchronous inductur)

5.แบบ electromechanical

6.แบบ electrohydrolic

ก่อนจะกล่าวถึงรายละเอียดต่อไป จะขออธิบายคำศัพท์ที่ใช้ในบทความนี้ที่สำคัญ จะยกตัวอย่างกับ stepping motor s1 phase แบบ permanent magnet ดังรูปที่ 2

PHASE เป็นส่วนของขดลวดระหว่างปลายสายกับ center tap หรือทั้งขดลวดค่าไม่มี center

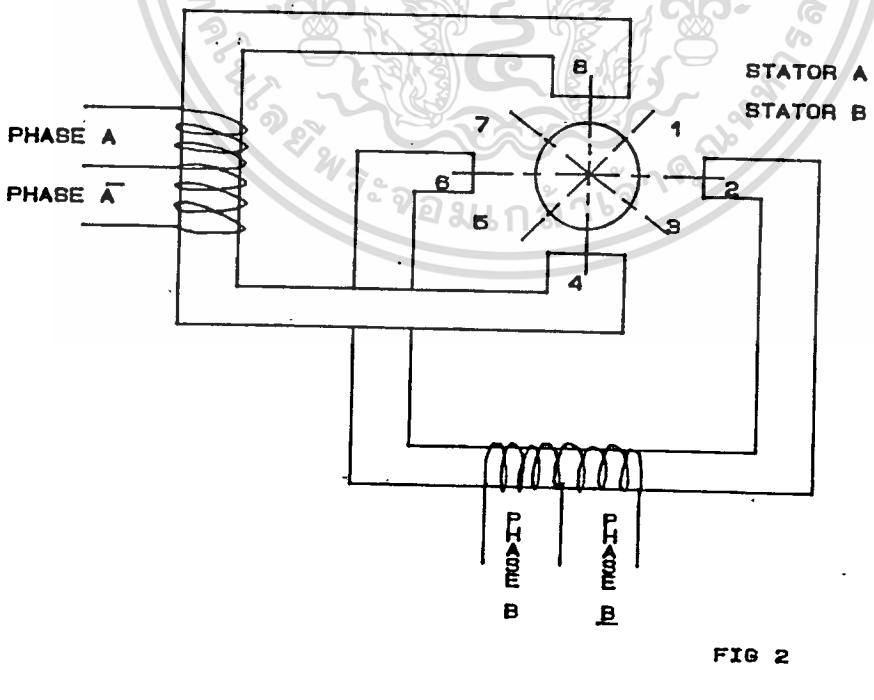


FIG 2

รูปที่ 2 STEPPING MOTOR 4 PHASE แบบ UNIPOLAR - PERMANENT MAGNET

FULL STEP MODE หนึ่ง phase บนแต่ละ stator จะถูก energised ที่เวลาหนึ่ง rotor จะหมุน"full steps" นั่นคือจากรูปที่ 2 จะหมุนตำแหน่งที่ 1 ไปยังตำแหน่งที่ 3,5,7 และกลับไปทวนซ้ำตั้งแต่ตำแหน่งที่ 1 อีก

HALF STEP MODE ทุก ๆ step ที่ 2 จะมีเพียง phase เดียวที่ถูก energised และระหว่าง step อื่น ๆ เป็น 1 phase บนแต่ละ stator rotor จะหมุน "half steps"นั่นคือ หมุนจากตำแหน่งที่ 1 ไปยังตำแหน่งที่ 2,3,4,5, และทวนซ้ำ

<u>การทำงานพื้นฐาน</u>

สเต็ปปิ้งมอเตอร์ เป็นมอเตอร์ที่ถูกใช้ทำงานโดยสัญญาณอินพุทที่เป็นรูปพัลส์ที่ให้เข้ามา จะให้การเปลี่ยนแปลงสถานะสนามแม่เหล็กไฟฟ้าที่เกิดขึ้น และให้การหมุนของมอเตอร์เป็นมุมที่คงที่ ซึ่งจะแตกต่างกับการหมุนของมอเตอร์ธรรมดาในระบบควบคุมที่ให้สัญญาณดิจิตอลนี้ เช่นการส่งข้อมูล การควบคุมข้อมูล โดยที่เป็นปริมาณค่าของสัญญาณดิจิตอลทั้งหมด จึงให้ควบคุมการทำงานของเต็ปมอเตอร์ได้เป็นอย่างดี

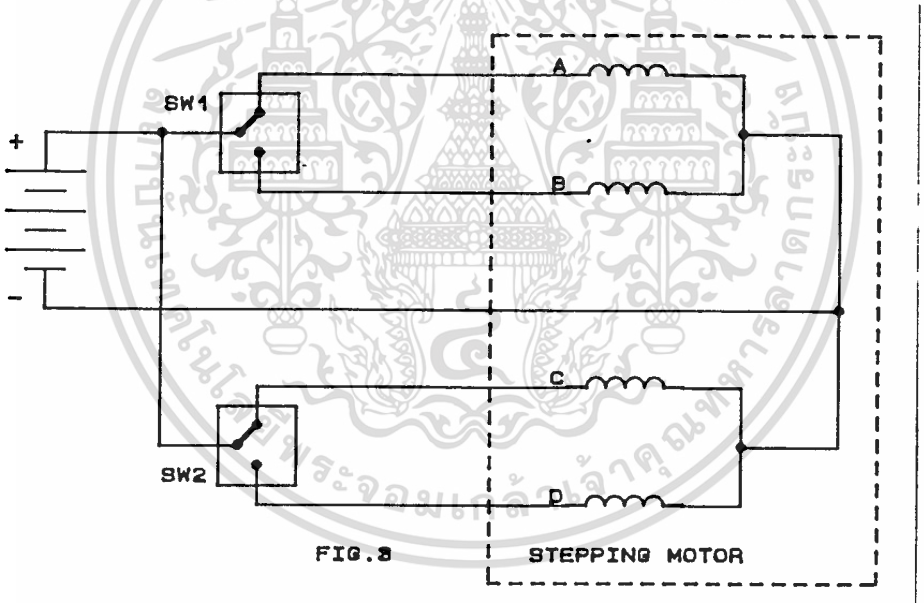


FIG.3

รูปที่ 3 BLOCK DIAGRAM การทำงานของ STEPPING MOTOR

การทำงานพื้นฐานของสเต็ปปิ้งมอเตอร์ ดังแสดงในรูปที่ 3 ซึ่งแสดง บล๊อกไดอะแกรมของวงจรขับสเต็ปปิงมอเตอร์ ถ้าให้กระแสไฟดีซีเรียงลำดับเข้าไปตามเฟส (PHASE) ต่าง ๆ ของมอเตอร์ก็จะทำให้มอเตอร์ที่หมุนไปตามองศาที่กำหนดไว้ การที่จะให้ไฟดีซีเรียงลำดับเข้าไปที่มอเตอร์ได้ก็โดยใช้การทำงานของสวิทช์ และเพื่อให้มอเตอร์หมุนจึงจำเป็นจะต้องมีวงจรขับสเต็ปปิงมอเตอร์ ในรูปที่ 3 สวิทช์ที่จะควบคุมการไหลของกระแสไฟคือสวิทช์ 1 และสวิทช์ 2 ถ้าให้การทำงานของสวิทช์ตามตารางที่ 1(a) ถ้าสวิทช์ทำงานโดยใช้รีเลย์หรือมือ

โยก สเต็ปปิงมอเตอร์ก็จะหมุนเหมือนกัน แต่จุหมุนช้าเนื่องจากรีเลย์หรือมือโยกให้การเปลี่ยนแปลงของสวิทช์ช้าซึ่งจะทำได้ไม่เร็วเท่ากับความถี่สูงสุดของสัญญาณพัลส์ที่มอเตอร์สามารถทำงานได้ (เป็นจำนวนพัลส์ต่อวินาที)

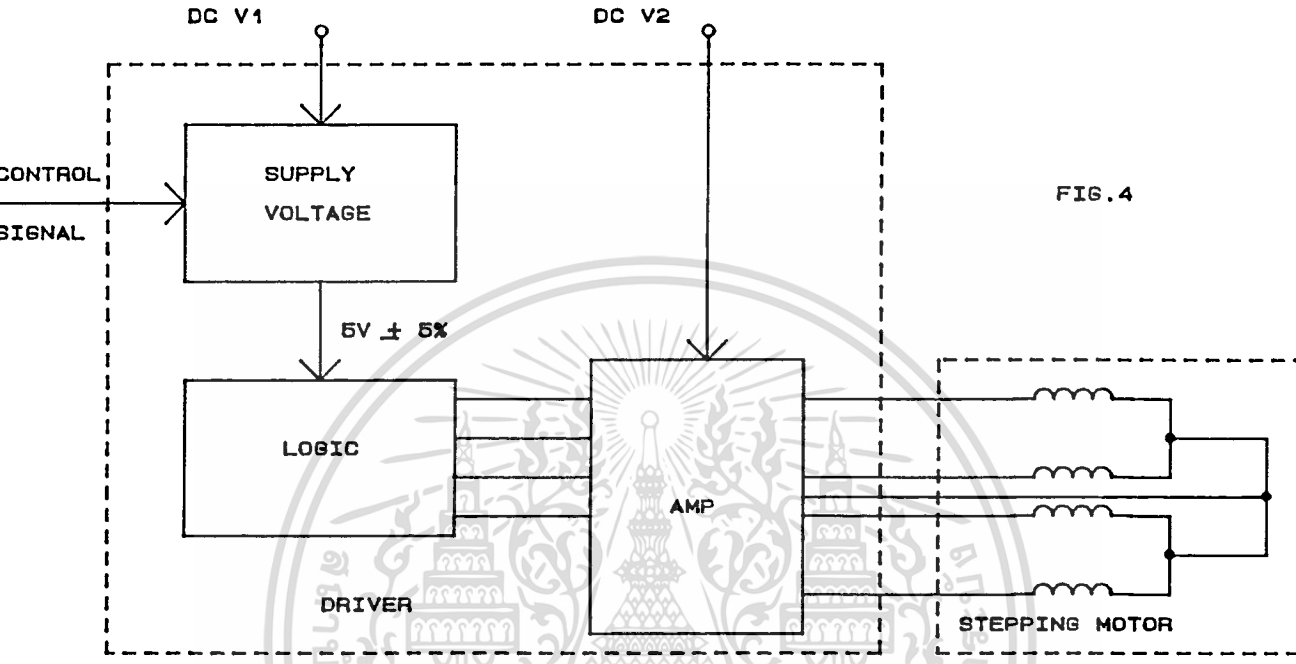


FIG.4

รูปที่ 4 บล๊อกไดอะแกรมของวงจรขับมอเตอร์

วงจรขับมอเตอร์มีหลายชนิดขึ้นอยู่กับชนิดของสเต็ปปิงมอเตอร์และชนิดของการให้เกิดสนามแม่เหล็ก ในที่นี้จะแสดงวงจรพื้นฐานที่ใช้ในการขับสเต็ปปิงมอเตอร์ชนิด 4 เฟส (4 ตำแหน่ง ในหนึ่งระยะ) และความสัมพันธ์ของคุณสมบัติของสเต็ปปิงมอเตอร์

<u>ระบบการเกิดสนามแม่เหล็ก</u>

จากคุณสมบัติพื้นฐานของสเต็ปปิงมอเตอร์ คุณสมบัติทางด้านสเตติคส์จะกำหนดด้วยคุณสมบัติของตัวมอเตอร์ ส่วนคุณสมบัติของทรานเซี้ยน จะเปลี่ยนแปลงอย่างมากตามระบบการเกิดสนามแม่เหล็กระบบการขับและ โครงสร้างของวงจรที่เอ้าท์พุท

ระบบการเกิดสนามแม่เหล็กของสเต็ปปิงมอเตอร์ มีอยู่ด้วยกัน 3 ระบบ คือระบบการเกิดสนามแม่เหล็กเฟสเดียว ระบบการให้เกิดสนามแม่เหล็ก 2 เฟส และการให้เกิดสนามแม่เหล็ก 1-2 เฟส ความสัมพันธ์และคุณสมบัติของระบบต่าง ๆ ดังแสดงในตารางที่ 3 และ 4

STEP	SW.1	SW.2
1	A	
2		C
3	B	
4		D
1	A	

1 PHASE

STEP	SW.1	SW.2
1	A	C
2	B	C
3	B	D
4	A	D
1	A	C

2 PHASE

STEP	SW.1	SW.2
1	A	C
2		C
3	B	C
4	B	
5	B	D
6		D
7	A	D
8	A	
1	A	C

1-2 PHASE

(a)ในกรณีที่ให้สนามแม่เหล็ก เฟสเดียว (b) ในกรณีที่ให้สนามแม่เหล็ก 2 เฟส (c) ในกรณีที่ให้สนามแม่เหล็ก 1-2 เฟส

ตารางที่ 1 ความสัมพันธ์ระหว่างการเกิดสนามแม่เหล็กกับสวิทช์

ระบบการเกิดสนามแม่เหล็ก 1 เฟส ระบบนี้จะมีประสิทธิภาพมากแต่เนื่องจากมีคุณสมบัติของการแดมปิ้งไม่ค่อยดี ดังนั้นจะใช้งานในย่านความถี่ที่เหมาะสมได้ดีเท่านั้นในกรณีที่ใช้ในช่วง Pulse rate ที่กว้าง จะต้องพิจารณาบริเวณที่จะเกิดการออสซิลเลท ซึ่งบริเวณนี้ไม่ถูกนำมาใช้งาน ตารางที่ 4 เปรียบเทียบคุณสมบัติทรานเซี้ยนกับแบบระบบการเกิดสนามแม่เหล็ก 2 เฟส

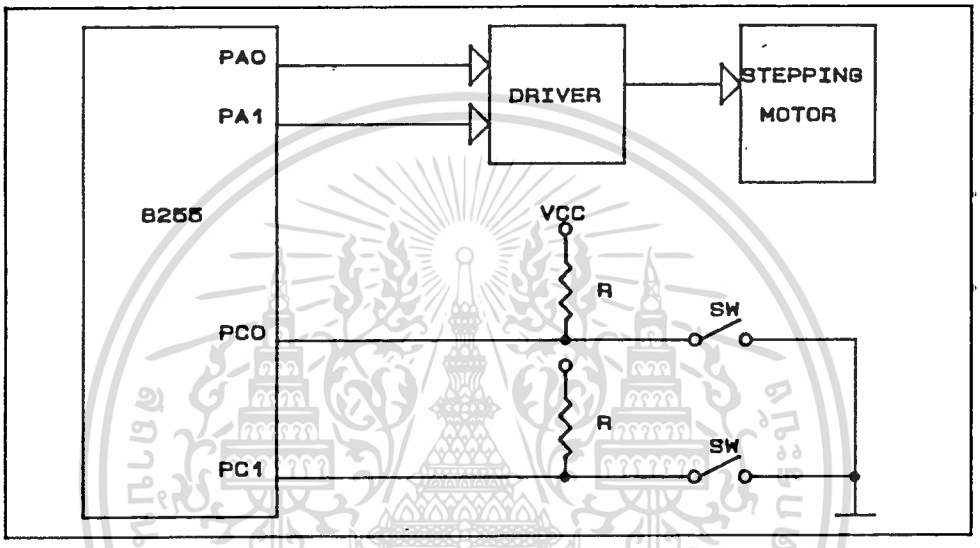
ระบบการเกิดสนามแม่เหล็ก 2 เฟส ระบบนี้จะใช้กันอย่างกว้างขวาง เมื่อเปรียบเทียบกับแบบแรกแล้ว จะใช้อินพุทสองเท่าเปรียบเทียบค่าทอร์ก (Torque) มากที่สุดของมอเตอร์ 4 เฟสแล้ว จะเพิ่มขึ้นแต่ 2 เท่า แต่ค่าคุณสมบัติทางด้านแดมปิ้งจะดีกว่าแบบแรก

ระบบการเกิดสนามแม่เหล็ก 1 เฟส ตามธรรมดาเป็นระบบที่ให้กระแสไหลผ่านเพียงเฟสเดียว ระบบการเกิดสนามแม่เหล็ก 2 เฟส ตามธรรมดาเป็นระบบที่ให้กระแสไหลผ่าน 2 เฟส ส่วนระบบการเกิดสนามแม่เหล็ก 1-2 เฟส จะให้กระแสไหลผ่านเฟสเดียวและ 2 เฟสสลับกันไป คุณสมบัติพิเศษของระบบนี้ก็คือ จะให้จำนวนการสเต็ปของมอเตอร์ลดลง 1/2 ของในระบบ 1 เฟส และ 2 เฟส ด้วยเหตุนี้ จำนวนรีโซลูชั่นของการสเต็ปจะเพิ่มขึ้น 2

เท่า และในกรณีของระบบ 1 เฟส กับ ระบบ 2 เฟส ที่ความถี่หนึ่ง ซึ่งให้การทำงานที่เกิดการออสซิลเลชั่นและเกิดเสียงดัง ในระบบนี้จะลดข้อเสียงที่เกิดขึ้นนี้ แต่ข้อเสียของระบบนี้ก็คือความหนาแน่นของการสเต็ปจะไม่ค่อยดี

ลำดับขั้นการทดลอง

1.ประกอบวงจรตามรูปที่ 5 โดยกำหนดให้ 8255 PA เป็น OUTPUT PORT และ PC_0,PC_1 เป็น INPUT PORT กำหนดให้อยู่ที่ PORT C_0,C_1 และ C_2

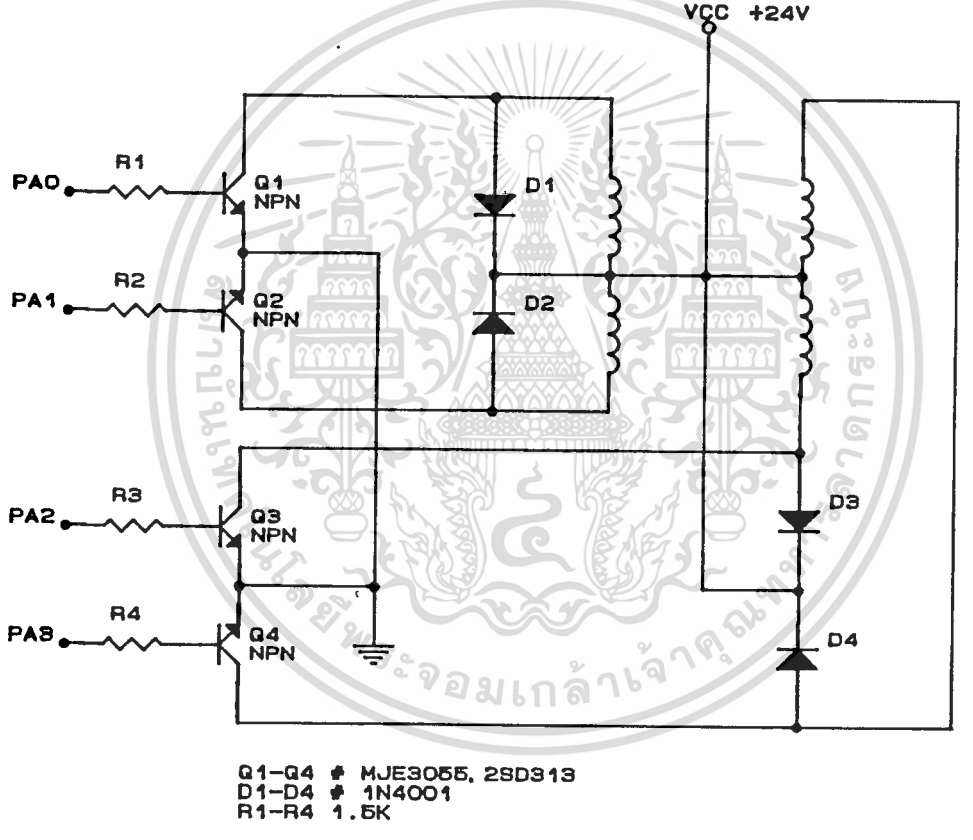


รูปที่ 5

การทำงานของวงจร STEPPING MOTOR DRIVER

จากวงจรในรูปที่ 6 แสดงภาค DRIVER ของ STEPPING MOTOR การ CONTROL ก็สามารถทำได้โดย ส่งข้อมูลผ่าน IC# 74373 เข้ามา เมื่อ Q1 ได้รับ LOGIC "1" แต่ Q2-Q4 ได้รับ LOGIC "0" ก็จะทำให้ Q1 "ON" เป็นเหตุให้ STEPPING MOROT หมุนไปหนึ่ง STEP คือ 1.8 องศา เมื่อต้องการให้มอเตอร์หมุนไปอีก หนึ่ง STEP ก็ให้ LOGIC A1 เป็น "0" ,QZ="1",Q3-Q4-"O" ก็ทำให้ STEPPING MOTOR สามารถหมุนไปอีก STEP ซึ่งสามารถเขียนเป็นตารางได้ดังนี้

STEP	Q1	Q2	Q3	Q4
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	1	0	0	0
6	0	1	0	0



DRIVER CIRCUIT FOR STEPPING MOTOR 4 ø

จากตารางจะเห็นว่า สถานะ LOGIC"1" จะ SHIGT ไปทีละ BIT เมื่อเราต้องการให้ STEPPING MOTOR หมุนไปกี่องศา เราก็สามารถกำหนดได้ โดยกำหนดจำนวน STEP (1STEP = 1.8 องศา) หรือ ถ้าต้องการให้ STEPPING MOTOR หมุนไปเรื่อย ๆ ก็สามารถทำได้ ดังตัวอย่างที่ 1

อุปกรณ์การทดลอง

1.MPF-IB & POWER SUPPLY

2.IC 8255

3. แผงวงจร DRIVER

4.DIP SW.

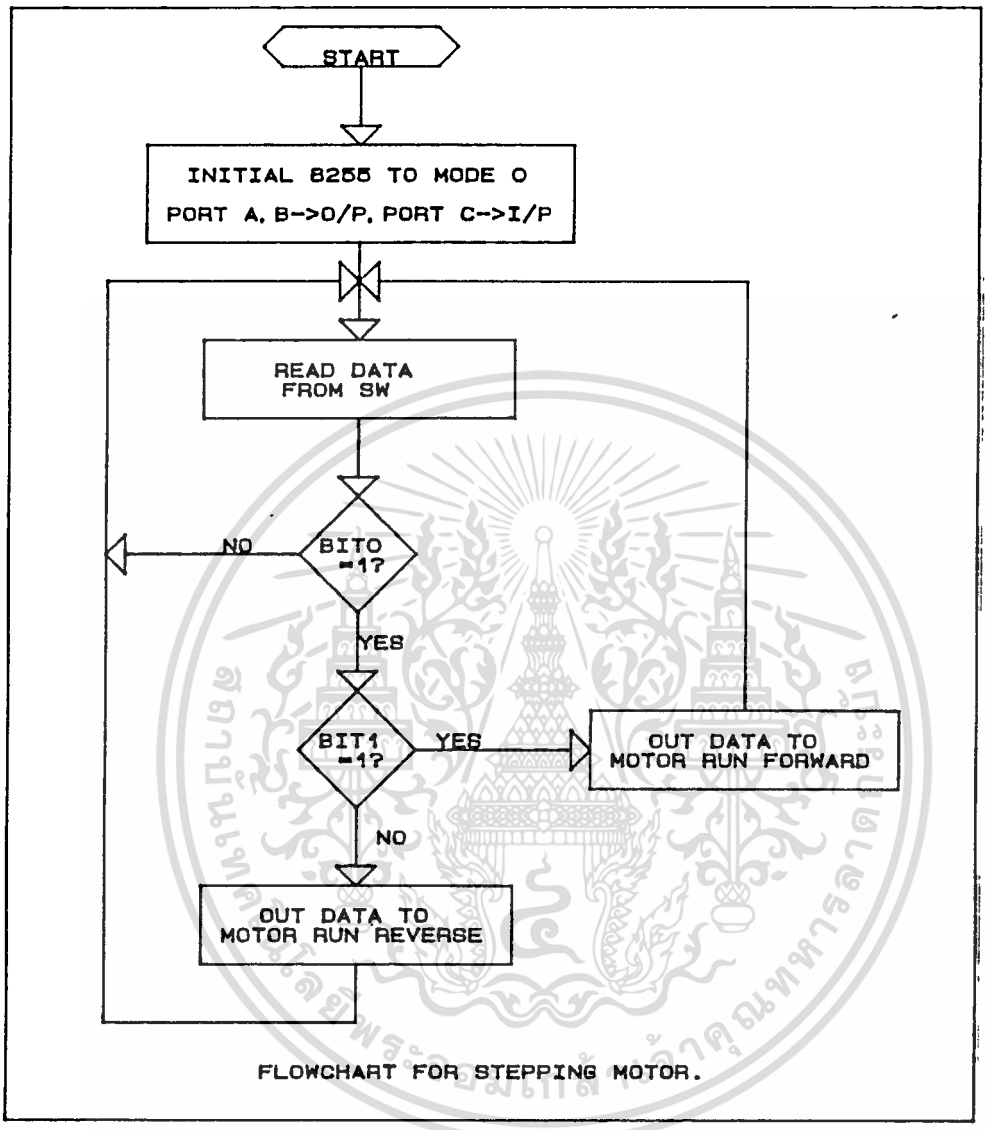
5.STEPPING MOTOR

6.POWER SUPPLY + 24V

2.เขียนโปรแกรมโดยรับข้อมูลจาก DIP SW. ที่ PORT C เพื่อควบคุมการหมุนของ STEPPING MOTOR โดยให้ PC0 ควบคุมให้ STEPPING หมุนหรือไม่หมุน PC1 ควบคุมทิศทางการหมุน FORWARD, REVERSE

3.สรุปผลการทดลอง

ผลการทดลอง



FLOWCHART FOR STEPPING MOTOR.

```
Address Machine Code  Label    Operand          Comment
1800    3E 89                  LD A, 89H        ;INITIAL 8255 MODE 0
                                                ;PORT A,B OUTPUT
                                                ;PORT  C  INPUT
1802    D3 C3                  OUT (C3),A
1804    AF                     XOR  A           ;CLEAR OUTPUT PORT
1805    D3 C0                  OUT (C0),A
1807    DB C2          MAIN :  IN   A,(C2)      ;READ DATA FROM PORT C
1809    CB 47                  BIT  0,A         ;IF BIT 0 OF DATA IS '1'
                                                ;MOTOR START OTHERWISE
                                                ;GOTO MAIN
180B    CA 07 18               JP  Z,MAIN
180E    CB 4F                  BIT 1,A          ;IF BIT 1 IS '1' MOTOR FORWARD
                                                ;IF BIT 1 IS '0' MOTOR REVERSE
1810    20 0C                  JR  NZ,FFD
1812    06 04                  LD  B,4          ;LOOP FOR MOTOR RUN REVERSE
1814    3E 08                  LD  A,08H
1816    D3 C0          REV:    OUT (C0),A
1818    CB 3F                  SRL A
181A    10 FA                  DJNZ REV
181C    18 E9                  JR MAIN
181E    06 04          FFD:    LD B,4           ;LOOP FOR MOTOR RUN FORWARD
1820    3E 01                  LD A, 01H
1822    D3 C0          LOOP:   OUT (C0),A
1824    CB 27                  SLA A
1826    10 FA                  DJNZ LOOP
1828    18 DD                  JR MAIN
```

สรุปผลการทดลอง

กำหนดให้ 8255 ทำงานใน MODE 0 ให้ FORT A และ PORT B เป็น O/P PORT และ PORT C เป็น I/P PORT โดยให้ STEPPING MOTOR BORD ต่ออยู่กับ PORT B

และ PODRT C ต่อกับ DIP SW. เมื่อเริ่ม RUN PROGRAM CPU จะอ่าน DATA จาก DIP SW. และตรวจสอบว่า BIT 0 เป็น "1" หรือ เป็น "0" ถ้าเป็น "1" ก็จะกลับไปอ่าน DATA ใหม่ จนกว่า BIT 0 จะเป็น "0" จึงมาตรวจสอบ BIT 1 ถ้าเป็น "1" ก็จะไปทำงานใน LOOP ของการ CONNOL FORWARD ถ้าเป็น "0" ก็จะทำงานใน LOOP ของการ REVERSE การทำงานในแต่ละ LOOP ของการ FORWARD และ REVERSE นั้น จะ SHIFT PHASE ของ STEPPING MOTOR แค่ 4 PHASE เท่านั้น ก็จะจบ LOOP แล้วก็ไปทำการอ่าน DATA จาก DIP SW. ใหม่

การทดลองที่ 8

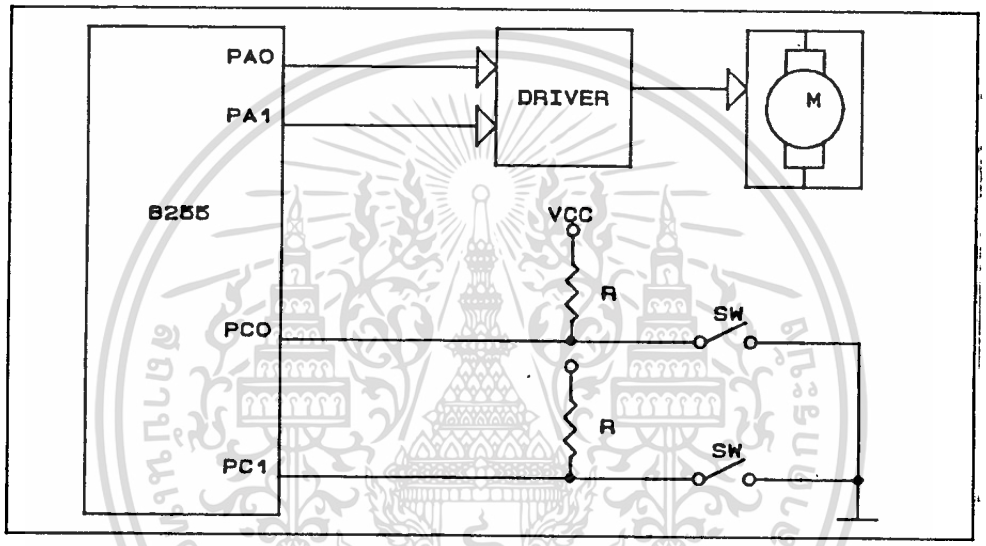
เรื่อง DC MOTOR

จุดประสงค์

1. เพื่อให้นักศึกษาสามารถเขียนโปรแกรมเพื่อควบคุมการทำงานของ DC MOTOR ได้

ทฤษฎี

การควบคุม DC MOTOR ด้วยไมโครโปรเซสเซอร์ให้สามารถหมุน FORWARD, REVERSE และ STOP ได้จะสามารถควบคุมได้โดยการเพิ่มชุด DRIVE MOTOR ส่วนการควบคุมทิศทางการหมุนจะควบคุมด้วย SW. ดัง Diagram ต่อไปนี้



รูปที่ 1

จะใช้ SW. เพียง 2 ตัว ก็สามารถควบคุม MOTOR ให้หมุน FORWARD, REVERSE และ STOP ได้

อุปกรณ์การทดลอง

1. MPF-IB & SUPPLY
2. ชุด DRIVER DC MOTOR
3. DIP SWITCH
4. CONNECTOR, WIRE

ลำดับขั้นการทดลอง

1. ประกอบวงจรตามรูปที่ 2 โดยการต่อ ชุด DRIVER เข้ากับ PA0 และ PA1 และต่อ DIP SW. เข้ากับ PC0 และ PC1

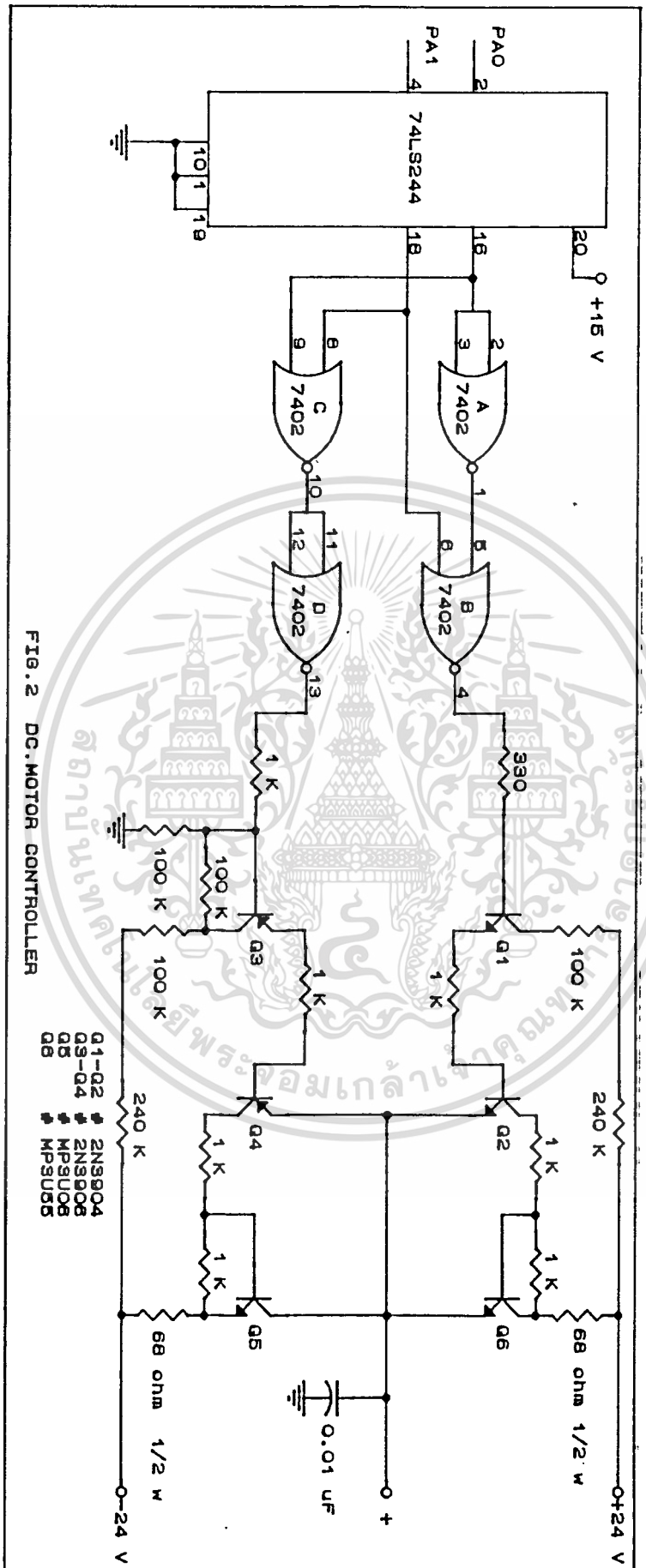


FIG.2 DC.MOTOR CONTROLLER

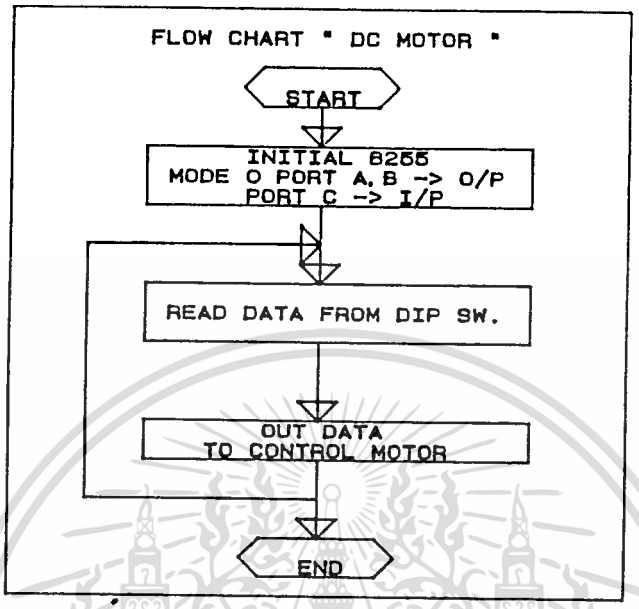
2. เขียนโปรแกรมรับข้อมูลจาก DIP SW. เพื่อควบคุม ทิศทางการหมุนของ MOTOR ดังต่อไปนี้

PC1	PC0	
0	0	REVERSE
0	1	STOP
1	0	FORWARD
1	1	STOP

3. อธิบายการทำงานของวงจร

ผลการทดลอง

FLOW CHART



Address	Machine Code	Label	Operand	Comment
1800	3E 89		LD A,89H	;INITIAL 8255 MODE 0
				PORT C-I/P
1802	D3 C3		OUT (C3),4	;PORT A,B
1804	DB C2	MAIN:	IN A, (0C2)	;READ DATA FROM SW.
1800	D3 C3		OUT (0C3),4	;OUT DATA TO CONTROL
				;MOTOR
1808	18 FA		JR MAIN	;GO BACK TO GET NEW DATA

การทำงาน

กำหนดให้ 8255 ทำงานใน MODE 0 โดยใช้ PORT A และ B เป็น OUTPUT PORT PORT C เป็น INPUT PORT เมื่อเริ่ม RUN PROGRAM CPU จะอ่าน DATA จาก DIP SW. และส่ง DATA ไปยัง 8255 PORT A เพื่อส่งให้วงจร DRIVER ควบคุมทิศทางการหมุนของ DC MOTOR ตามที่กำหนดไว้ในโปรแกรมนี้ จะทำงานเป็น LOOP ไม่มีการจบสิ้น คือเมื่อ READ DATA จาก DIP SW. มาแล้วจะส่ง DATA ไปควบคุม MOTOR แล้วหลังมา READ DATA ใหม่อีกเช่นนี้เรื่อย ๆ ไป

การทดลองที่ 9

เรื่อง DOT MATRIX LED

จุดประสงค์

1. เพื่อให้นักศึกษาสามารถสร้าง CHARACTIOR GEN. สำหรับ DOT MATRIX LED ได้

2. เพื่อให้นักศึกษาสามารถเขียนโปรแกรมเพื่อติดต่อกับ DOT MATRIX LED ได้

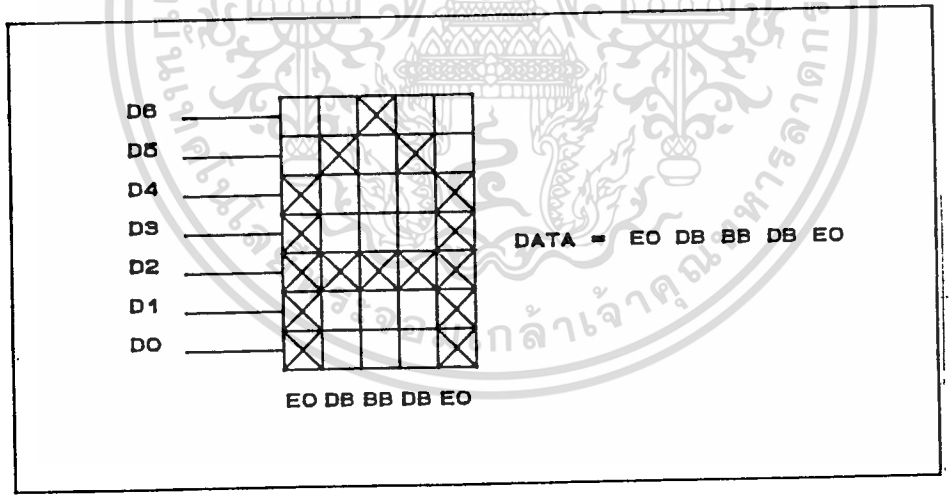
3. เพื่อให้นักศึกษาสามารถออกแบบวงจรเพื่อ INTERFACE กับ DOT MATRIX LED ได้

ทฤษฎี

การสร้าง CHARACTOR GEN สำหรับ DOT MATRIX LED

สำหรับการทดลองนี้ใช้ DOT MATRIX LED COMMON ANODE ขนาด 5 x 7

ตัวอย่าง CHARACTOR GEN. สำหรับตัวอักษร A



โครงสร้างของ DOT MATRIX LED

จากรูปที่ 1.1 และ 1.2 เป็นโครงสร้างของ DOT MATRIX LED COMMON CATHODE และ COMMON ANODE

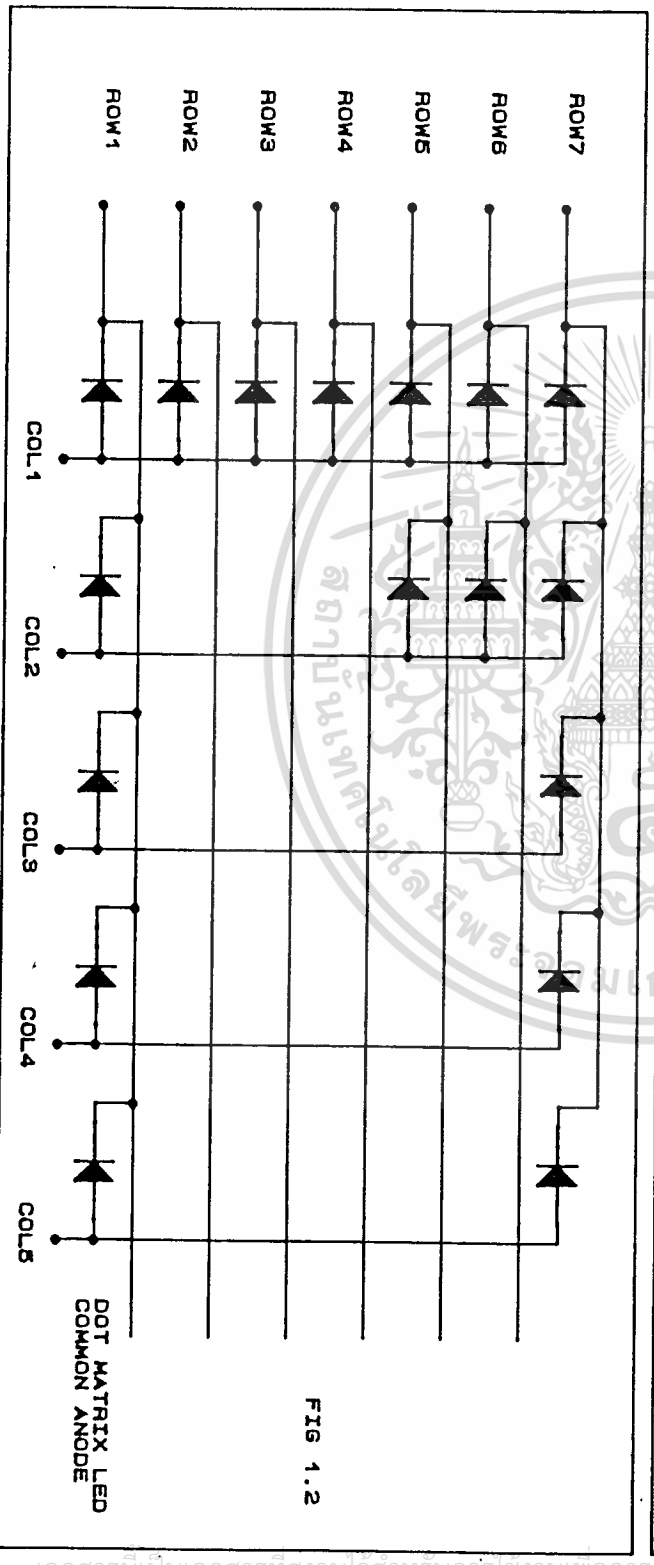


FIG 1.2

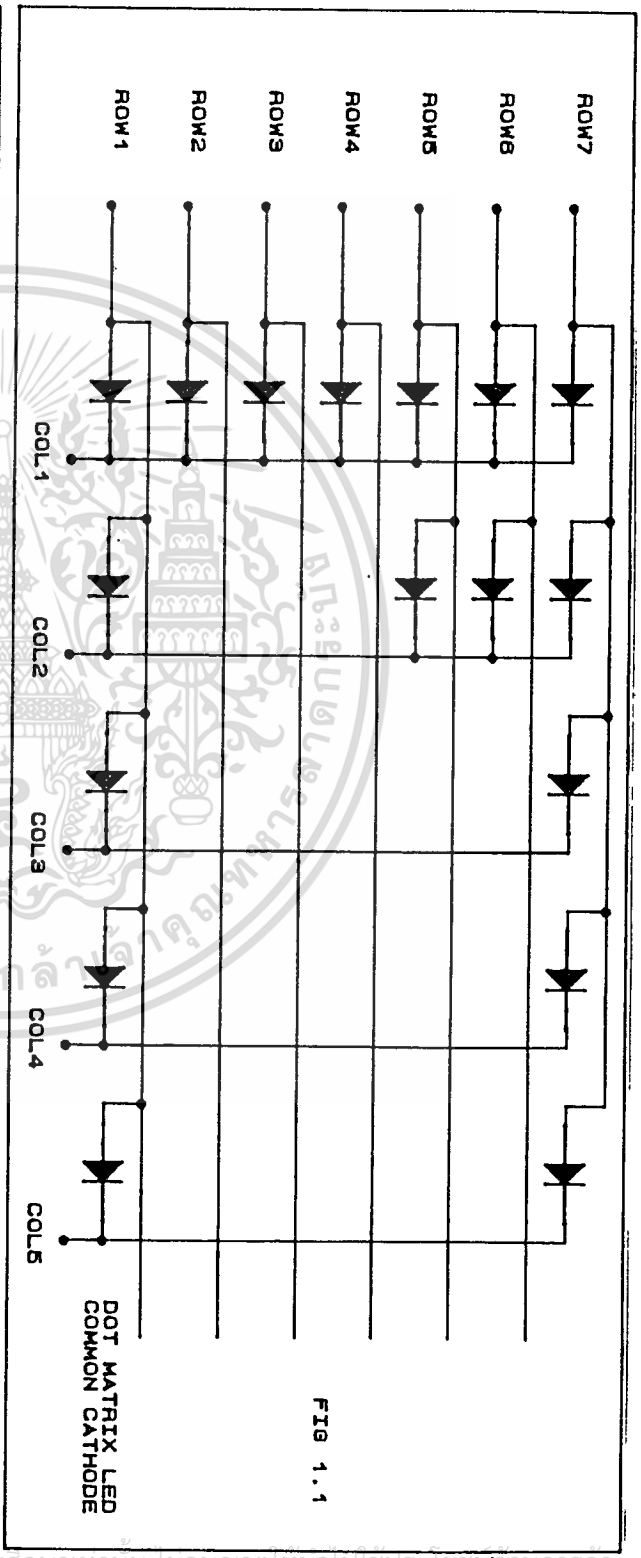


FIG 1.1

อุปกรณ์การทดลอง

1.MPF-IB & SUPPLY

2.DOT MATRIX LED

3.IC 74LS245

4.CONNECTOR

ลำดับขั้นการทดลอง

1.ประกอบวงจรตามรูปที่ 2

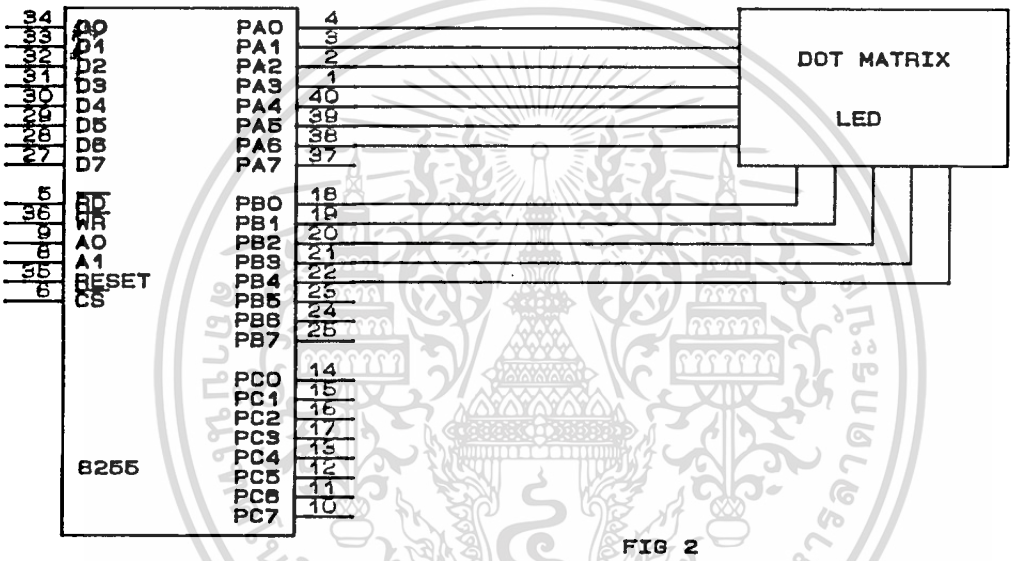


FIG 2

2.สร้าง CHARACTOR GEN. สำหรับตัวอักษร A - Z และ 0 - 9

3.เขียนโปรแกรม เพื่อให้ DOT MATRIX แสดงผลเป็นตัวอักษร A

ผลการทดลอง

Address	Machine Code	Label	Operand	Comment
1800	3E		LD A,89H	;INITIAL 8255 PORT A,B
1802	D3 C3		OUT (C3),A	;OUTPUT
1804	21 00 19	NEXT:	LD HL, 1900H	
1807	OE O1		LD C, 01	
1809	06 05		LD B, 05	
180B	7E	LOOP:	LD A,(HL)	
180C	D3 C0		OUT (C0),A	
180E	79		LD A,C	
180F	D3 C1		OUT (C1),A	
1811	CD 30 18		CALL DELAY	
1814	CB 01		RLC C	
1816	3E FF		LD A, FF	
1818	D3 C0		OUT (C0),A	
1814	3E 00		LD A, 00	
181C	D3 C1		OUT (C1),A	
181E	23		INC HL	
181F	10		DJNZ LOOP	
1821	18		JR NEXT	
1830	11 00 40	DELAY:	LD DE, 4000H	
1833	1B	DELAY1:	DEC DE	
1834	7R		LD A,D	
1835	B3		OR E	
1836	20 FB		JR NZ,DELAY1	
1838	C9		RET	
1900		TABLE		

สรุปผลการทดลอง

จากการทดลองเรื่องการ INTERFACE กับ DOT MATRIX LED จะเห็นว่าการสั่ง

งานให้DOT MATRIX LED แสดงผลเป็นตัวเลขหรือตัวอักษรใด จะต้องใช้ DATA ถึง 5 BYTE ในกรณีของ DOT MATRIX ขนาด 5 x 7 ซึ่งแตกต่างกับการแสดงผลด้วย LED 7 SEGMENT ซึ่งใช้ DATA เพียง 1 BYTE ในการแสดงตัวอักษร 1 ตัว แต่ข้อดีที่เหนือกว่าของ DOT MATRIX LED เพื่อเทียบกับ 7 SEGMENT คือ DOT MATRIX สามารถแสดงตัวอักษรได้ทุกตัว และมีความสวยงามกว่า

จากการทดลองนี้เราสามารถเพิ่มเติม DOT MATRIX เพื่อการแสดงตัวอักษรหรือข้อความที่มากกว่า 1 ตัวอักษรได้

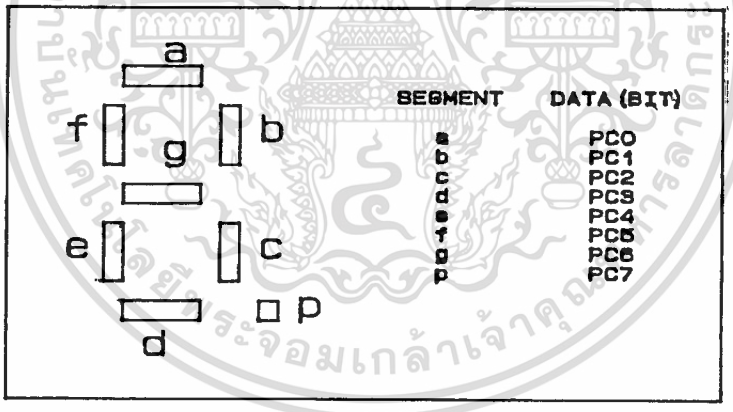
การทดลองที่ 10

INTERFACE KEYBOARD DISPLAY

การ INTERPACE ระหว่าง KEYBOARD & DISPLAY นั้น วิธีที่ง่ายที่สุด ก็คือ ใช้วิธีการ SCAN ในวงจรที่ใช้ในการทดลองนี้ เราใช้ 8255 เป็น อินพุท เอ๊าพุทพอร์ท ซึ่งรายละเอียดต่าง ๆ ของ 8255 เราได้กล่าวไว้ในตอนต้นของคู่มือเล่มนี้แล้วในวงจรนี้ เราได้กำหนดให้ 8255 อยู่ใน MODE 0 PORT C เป็น เอ๊าท์พุทพอร์ท สำหรับ SCAN SEGMENT ที่ ACTIVE HIGH "1" PORT A เป็น OUTPUT PORT สำหรับ SCAN DIGIT ที่ ACTIVE LOW "0" ส่วน PORT B เป็น INPUT PORT สำหรับการอ่านคีย์ ซึ่ง CONTROL WORD ที่ใช้มีค่าเท่ากับ 089H

การ SCAN DISPLAY

การที่จะให้ DISPLAY แสดงผลได้นั้น ก็ต้องส่งสัญญาณจาก PORT C มาทำการ SCAN SEGMENT โดยใช้ IC 74D244 เป็นทัฟเฟอร์ต่อระหว่าง PORT C และ SEGMENT ของ DISPLAY แต่ละตัว และแต่ละ SEGMENT ก็จะต่อร่วมกันกับ SEGMENT A ของ DISPLAY ตัวที่ 2,3,4,5 และ 6 ซึ่งตำแหน่ง และการต่อ SEGMENT กับ DATA ของ PORT C ดังรูป



ดังได้กล่าวมาแล้วว่า สัญญาณที่ส่งมา SCAN SEGMENT นั้น ACTIVE HIGH "1" ดังนั้น สมมุติว่า เราต้องการให้ DISPLAY แสดงเป็นตัวเลข "0" สัญญาณที่ส่งมา SCAN ก็ต้องให้ SEGMENT a,b,c,d,e และ f เป็น "1" นอกนั้นให้เป็น "0" ซึ่งจะได้ DATA เท่ากับ 3F ตารางข้างล่างนี้จะเป็นตารางของ DATA ซึ่งจะใช้ DISPLAY แสดงผลเป็นตัวอักษร 0 ถึง F

ตัวอักษร	DATA
0	3FH
1	06H

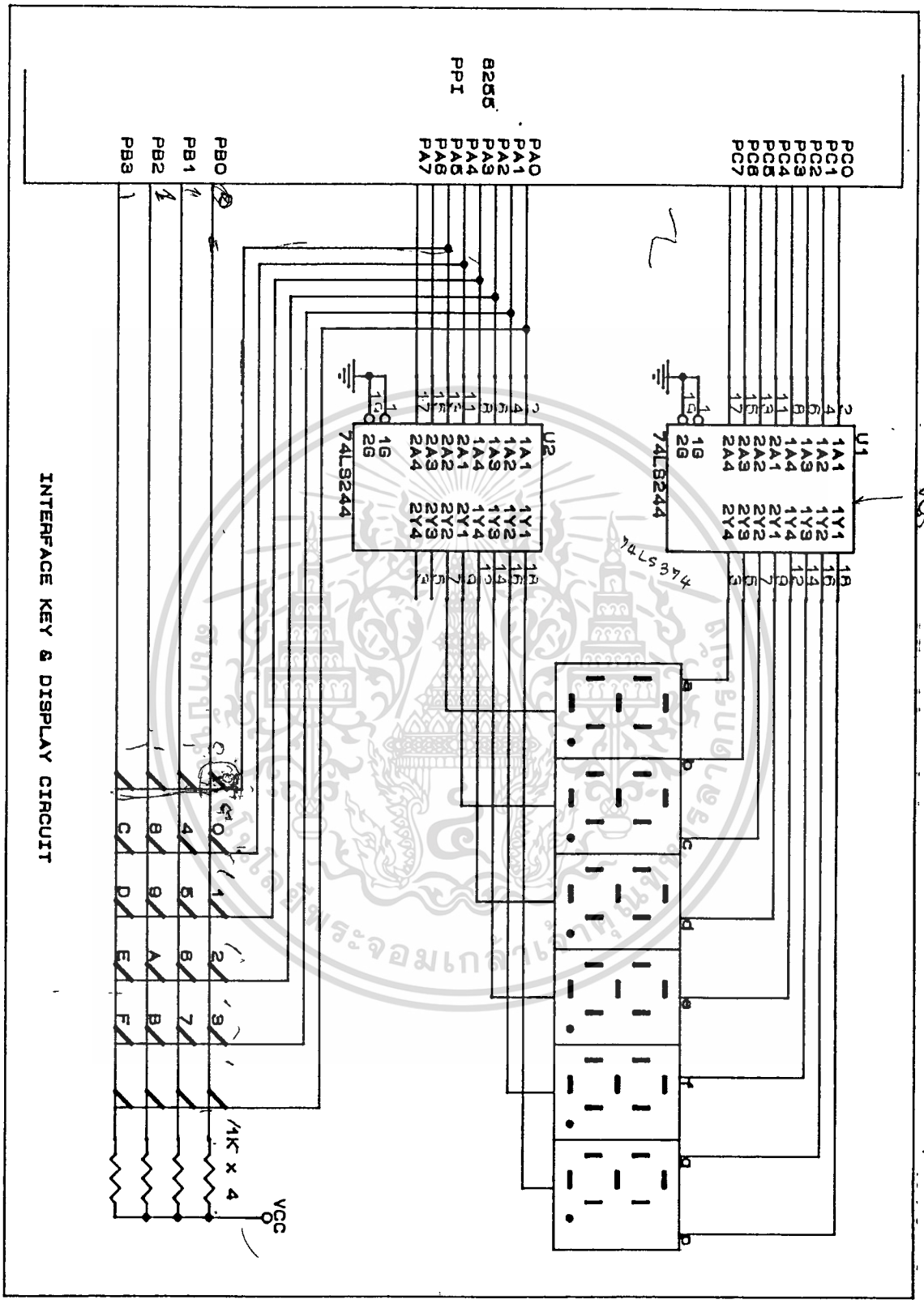
2	5BH
3	4FH
4	66H
5	6DH
6	7DH
7	07H
8	7FH
9	6FH
A	77H
B	7DH
C	39H
D	5FH
E	79H
F	71H

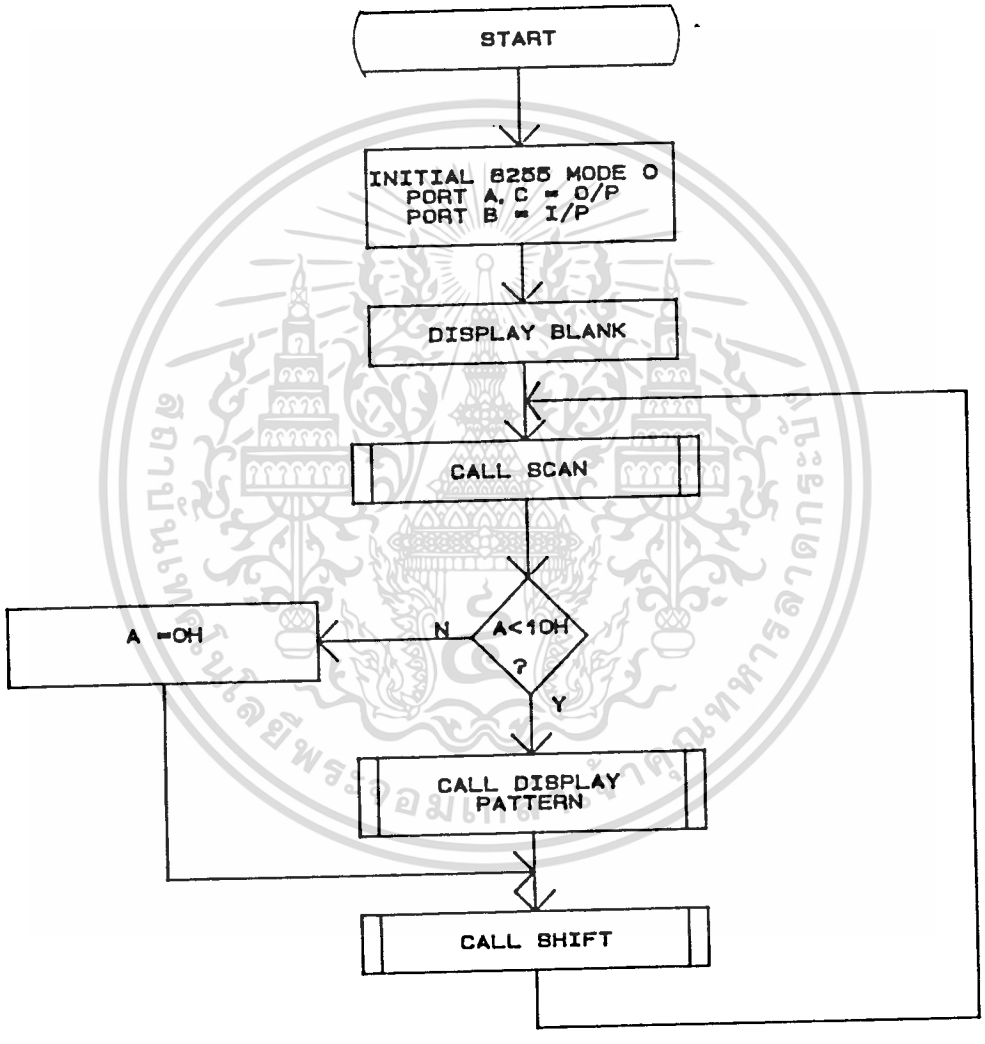
ในการเขียนโปรแกรมนั้น เราก็จะเก็บตารางนี้ไว้ในหน่วยความจำถ้าเราต้องการให้แสดงผงเป็นรูปอะไร ก็ทำได้โดยเปิดตารางหน่วยความจำ แล้วดึงเอา DATA มาใช้งาน

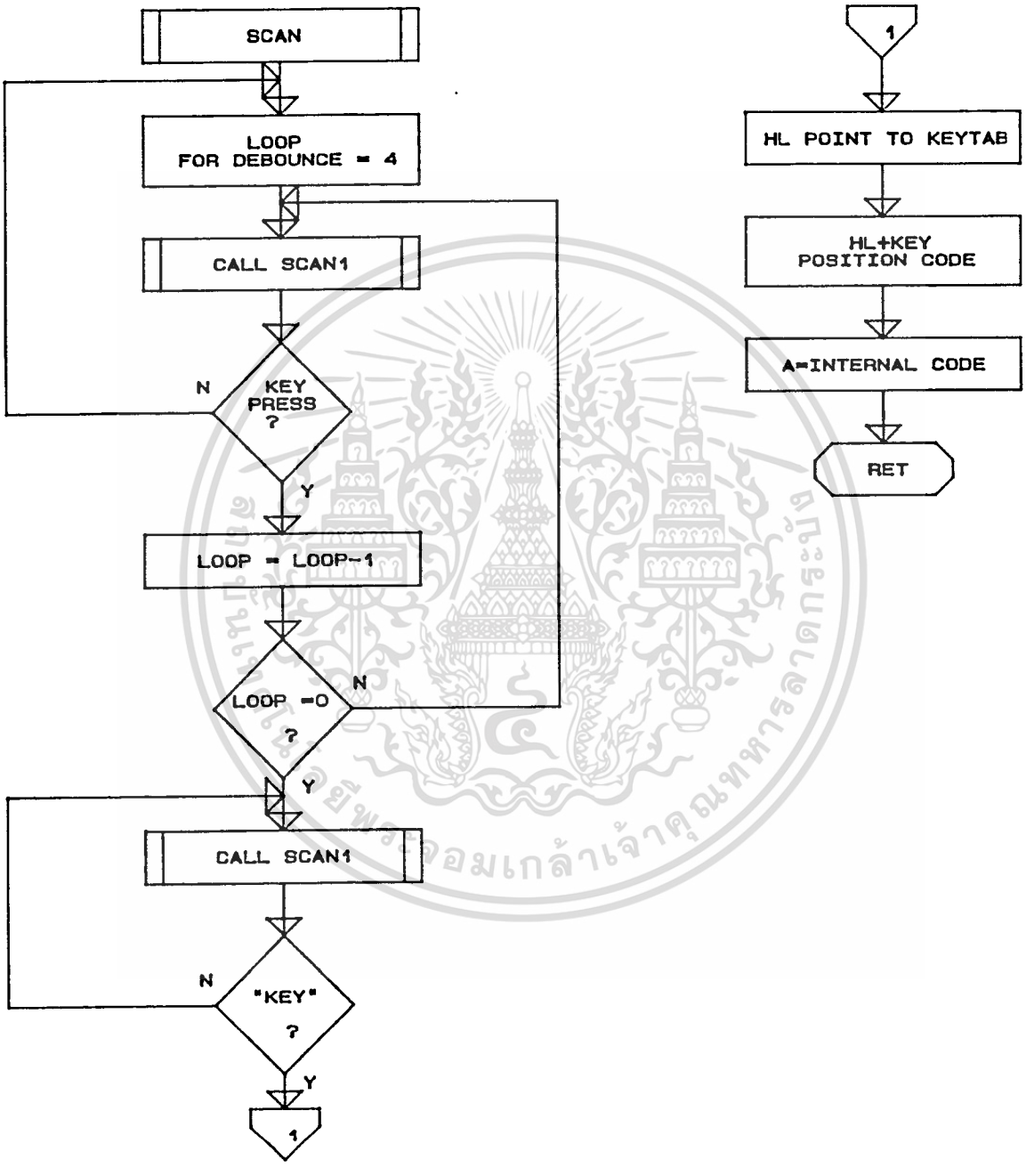
ในตอนต้นได้กล่าวถึงการ SCAN SEGMENT ไปแล้ว เราก็สามารถกำหนดให้ DISPLAY ตัวใดแสดงผลก็ได้ โดยการส่งสัญญาณจาก port A มาทำการ scan digit โดยใช้ IC 74244 เป็นบัฟเฟอร์เช่นกัน ในวงจรมีการใช้ DISPLAY 6 DIGIT ดังนั้นเราก็ใช้ data จาก port A เพียง 6 bil เท่านั้น คือใช้ BIT 0 ถึง BIT 5 ในการ SCAN DIGIT นั้น จะแอคตีฟ LOW "0" สมมิติว่า เราต้องการให้ DIGIT 3 แสดงผลเป็นเลข "5" ก็ทำได้โดยการให้ PORT C มีข้อมูลเป็น 6DH และ PORT A มีข้อมูลเป็น 11111011B (FBH)

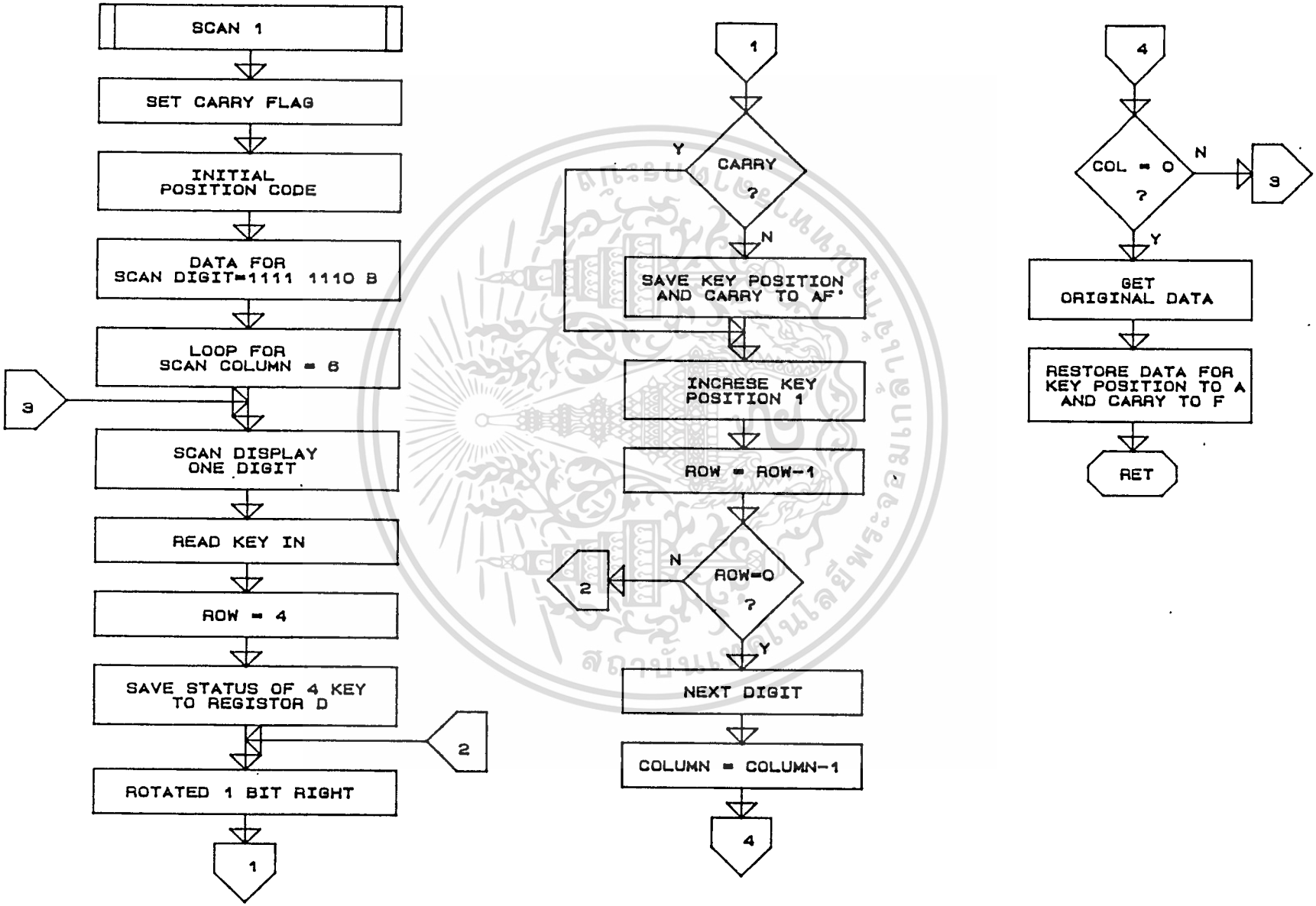
การอ่าน KEY

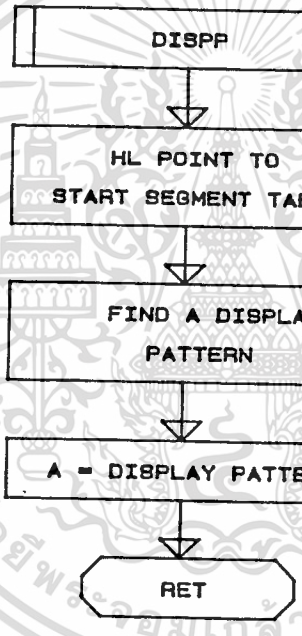
ในวงจรนี้เราสามารถต่อ KEY ได้ถึง 24 KEY (6x4) แต่ที่เราใช้ในขณะนี้เราใช้เพียง 20 key เท่านั้น ในการตรวจสอบ key ว่า มีการกดหรือไม่นั้น ก็ทำได้โดยอาศัย data จาก port A ที่ส่งให้ Scan digit (active low) มา scan ทาง column ของ key board ทีละ column โดยเริ่มจาก column ทางขวามือก่อน แล้วอ่าน I/P เข้าไปทาง PORT B แล้วเลื่อนข้อมูลไปทางขวา 1 บิท แล้วตรวจสอบดูว่า มี carry หรือไม่ ถ้ามี carry แสดงว่าไม่มีการกดคีย์ แต่ถ้าไม่มี carry แสดงว่ามีการกด key ดังนั้นต้องใช้โปรแกรม ตรวจสอบว่าเป็น key ใด แล้วไปเปิดตารางในหน่วยความจำ เพื่อเอา data มาแสดงผล ซึ่งความสามารถของวงจรก็ขึ้นอยู่กับ โปรแกรมที่ใช้ด้วยซึ่งรายละเอียดต่าง ๆ ของโปรแกรม จะอธิบายให้เข้าใจอีกครั้งหนึ่ง

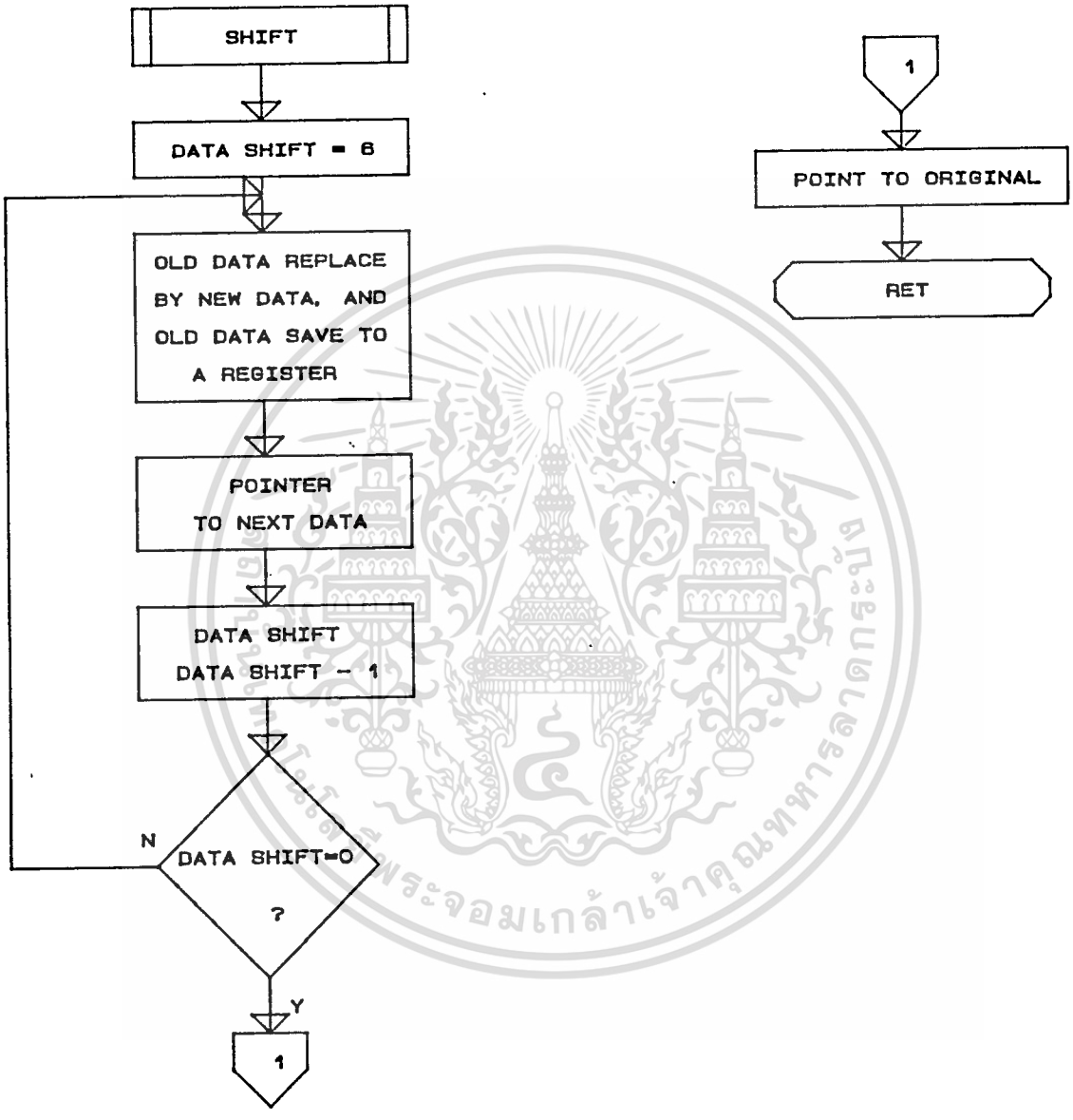







SCAN 1
SET CARRY FLAG
INITIAL POSITION CODE
DATA FOR SCAN DIGIT=1111 1110 B
LOOP FOR SCAN COLUMN = 8
3
SCAN DISPLAY ONE DIGIT
READ KEY IN
ROW = 4
SAVE STATUS OF 4 KEY TO REGISTOR D
2
ROTATED 1 BIT RIGHT
1
1
CARRY ?
Y
N
SAVE KEY POSITION AND CARRY TO AF'
INCRESE KEY POSITION 1
ROW = ROW-1
ROW=0 ?
N
2
Y
NEXT DIGIT
COLUMN = COLUMN-1
4
4
COL = 0 ?
N
3
Y
GET ORIGINAL DATA
RESTORE DATA FOR KEY POSITION TO A AND CARRY TO F
RET





```
;*************************************************************
;*************************************************************
;**       PROGRAM FOR CONTROL KEY BOARD AND DISPLAY.        **
;**                  PROGRAM  BY  CH.SAROT                  **
;*************************************************************
;*************************************************************
;
;*************************************************************
;*-WHEN EXECUTED THIS PROGRAM,DISPLAY WILL BLANK.WHEN USER  *
;* PRESS ANY DEY OF KEY 0-F DISPLAY WILL SHOW SAME THE KEY- *
;* PRESSED.IF PRESS ANOTHER KEY DISPLAY BILL BLANK.         *
;*************************************************************
      ORG     1800H          ;ORIGINAL ADDRESS.
      LD      IX,BLANK       ;DISPLAY BUFFER POINT TO BLANK.
      LD      A,82H          ;INITIAL 8255 MODEO,PORTB-I/P.
      OUT     (CTRL) .A      ;PCRTA AND C ARE OUTPUT.
MAIN: CALL    SCAN           ;CALL SUBROUTINE SCAN.
      CP      10H            ;IF KEY WAS PRESSED NOT KEY-
                             ;OF 0-F GOTO SPACE.
      JR      NC,SPACE
      CALL    DISPP
BACK: CALL    SHIFT
      JR      MAIN
SPACE:LD      A,0H           ;SET DISPLAY BLANK.
      JR      BACK
;
CTRL  EQU     0C3H           ;8255 CONTROL PORT
DIGIT EQU     0C0H           ;8255 PORT A
SEG   EQU     0C2H           ;8255 PORT C
KIN   EQU     0C1H           ;8255 PORT B
;*************************************************************
```

```
;FUNCTION SCAN: FOR SCAN KEY BOARD AND DISPLAY UNTIL AND KEY-
;WAS PRESSED.THIS SUBROUTINE SAVE KEY CODE TO REGISTER A.
;CALL SUBROUTINE SACN1
SCAN: LD      B,4            ;LOOP FOR DEBOUNCE KEY WHEN
                             ;ANY KEY IS DETECT.
SCNX: CALL    SCAN1
      JR      NC, SCAN
      DJNZ    SCNX
SCLOOP:CALL   SCAN1
      JR      C,SCLOOP
;CONVERT THE KEY-POSITION-CODE RETURN BY SCAN1 TO KEY INTERNAL
;CODE.THIS IS DONE BY IABLE LOOK-UP. THE TABLE USED IS KEYTAB.
      LD      HL,KEYTAB
      ADD     A,L
      LD      L,A
      LD      A,(HL)
      RET
;************************************************************
;FUNCTION SCAN1: THIS FUNCTION FOR SCAN KEY AND DISPLAY ONE TIME
;IF KEY WAS DETECTED BETWEEN SCAN CARRY FLAG IS '0'. IF NOKEY-
;CARRY FLAG WILL SET '1'
;DISPLAY RIGHTMOST POINT BY IX, AND DISPLSY LEFTMOST POINT BY-
;IX = 5
;THIS FUNCTION SAVE KEY POSTION TO REGISTER A.
SCAN1:SCF                    ;SET CARRY FLAG
      EXX
      EX      AF,AF:
      LD      C,OH           ;INITIAL POSITION CODE.
      LD      E,11111110B    ;SCAN FROM RIGHTMOST.
      LD      H,6            ;LOOP FOR SCAN COLUMN.
KCOL: LD      A,E
```

```
        OUT     (DIGIT),A       ;ACTIVE ONE COLUMN.
        LD      A,(IX)
        OUT     (SEG),A
        LD      B,201
        DJNZ    $               ;DELAY 1.5 ms PER DIGIT.
        XOR     A
        OUT     (SEG),A         ;DEACTIVE ALL DISPLAY SEGMENTS.
        LD      A,E
        OUT     (DIGIT),A
        LD      B,4             ;EACH COLUMN HAS 4 KEYS.
        IN      A,(KIN)         ;NOW BIT0-3 OF PORTB CONTAIN
                                ;THE STATUS OF 4 KEYS IN THE
                                ;ACTIVE COLUMN.
        LD      D,A             ;STORE A TO D.
KROW:   FF      D               ;ROTATE D 1 BIT RIGHT,BIT0 OF
                                ;D WILL BE ROTATE INTO CARRY.
        JR      C,NOKEY         ;SKIP NEXT 2 INSTRUCTIONS IF
                                ;THE KEY IS NOT PREXXED.
                                ;THE NEXT 2 INSTRUCTIONS STROE
                                ;THE CURRENT POSITION CODE
                                ;INTO A AND RESET CARRY FLAG
                                ;OF F REGISTER.
        LD      A,C             ;KEY-IN,GET KEY POSITION.
        EX      AF,AF'          ;SAVE A&CARRY INTO AF'.
NOKEY:  INC     C               ;INCREASE KEY CODE BY 1.
        DJNZ    KROW            ;LOP UNTIL 4 KEYS IN THE ACTIVE
                                ;CALUMN ARE ALL CHECKED.
        INC     IX
        LD      A,E
        RLC     A
        LD      E,A
```

```
        DEC     H
        JR      NA,KCOL
        KD      DE,-6
        ADD     IX,DE
        EXX
        EX      AF,AF'
        RET
;***************************************************************
;FUNCTION DISPP:THIS FUNCTION FOR CHANGE THE KEY CODE TO
;THE DISPLAY PATTERN.
DISPP:LD        HL,SEGTAB
        ADD     A,L
        LD      L,A
        LD      A,(HL)
;***************************************************************
;FUNCTION SHIFT : FOR SHIFT THE DISPLAY PATTERN WHEN KEY IS
;DETECTED.
SHIFT:LD        B,6             ;LOOP FOR SHIFT DATA 6 BYTE.
LOOP: EX        AF,AF'          ;SAVE NEW CODE.
        LD      A,(IX)
        EX      AF,AF'          ;RESTORE NEW CODE, SAVE (IX)
        LD      (IX),A          ;SAVE NEW DATA TO (IX).
        EX      AF,AF'          ;SVAE (IX) INTO A.
        INC     IX
        DJNZ    LOOP
        LDD     DE,-6
        ADD     IX,DE           ;GET ORIGINAL IX
;
KEYTAB:
KO      DEFB    20H             ;NOT USED
K:      DEFB    21H             ;NOT USED
```

```
K2      DEFB    22H             ;NOT USED
K3      DEFB    23H             ;NOT USED
K4      DEFB    03H             ;HEX-3
K5      DEFB    07H             ;HEX-7
K6      DEFB    0BH             ;HES-B
K7      DEFB    0FH             ;HEX-F
K8      DEFB    02H             ;HEX-2
K9      DEFB    06H             ;HEX-6
KOA     DEFB    0AH             ;HEX-A
KOB     DEFB    0EH             ;HEX-E
KOC     DEFB    01H             ;HEX-1
KOD     DEFB    05H             ;HEX-5
KOE     DEFB    09H             ;HEX-9
KOF     DEFB    0DH             ;HEX-D
K10     DEFB    0H              ;HEX-0
K11     DEFB    04H             ;HEX-4
K12     DEFB    08H             ;HEX-8
K13     DEFB    0CH             ;HEX-4
K14     DEFB    24H             ;NOT USED
K15     DEFB    25H             ;NOT USED
K16     DEFB    26H             ;NOT USED
K17     DEFB    27H             ;NOT USED
;
SEGTAB: DEFB    3FH             ;'0'
        DEFB    06H             ;'1'
        DEFB    8BH             ;'2'
        DEFB    4FH             ;'3'
        DEFB    66H             ;'4'
        DEFB    6DH             ;'5'
        DEFB    7DH             ;'6'
        DEFB    07H             ;'7'
```

```
        DEFB    7FH             ;'8'
        DEFB    8FH             ;'9'
        DEFB    77H             ;'A'
        DEFB    7CH             ;'B'
        DEFB    39H             ;'C'
        DEFB    5EH             ;'D'
        DEFB    79H             ;'E'
        DEFB    71H             ;'F'

BLANK:  DEFFB   0H
        DEFB    0H
        DEFB    0H
        DEFB    0H
        DEFB    0H
        DEFB    0H
        END
```

การทำงานของโปรแกรม KEY & DISPLAY

ถ้ากำหนดให้ ทำงานใน MODE 0 PORT A และ C เป็น O/P PURT PORT B เป็น I/P PORT โดนมหเ PORT A เป็น O/P สำหรับ SCAN DIGIT PORT C เป็น O/P สำหรับ SCAN SEGMENT และ PORT B เป็น I/P สำหรับอ่าน REY CODE เมื่อเริ่ม RUN PROGRAM DISPLAY จะ BLANK ต่อเมื่อมีการกด KEY ซึ่ง KEY ที่ใช้งานอยู่ มีอยู่ KEY เป็น KEY O-F KEY ใด KEY หนึ่งถูกกด DISPLAY ก็จะแสดงรูปแบบตาม KEY ที่ถูกกดและมีการกด KEY เพิ่มเข้ามาใหม่รูปแบบของ KEY ใหม่ จะเข้ามาแทนที่รูปแบบเก่า และรูปแบบเก่าก็จะเลื่อนขึ้นไป หลัก การแส่ดงผลของ DISPLAY จะเริ่มแสดงจากหลักทางขวาสุด ซึ่งการทำงานของแต่ละ FUNCTION จะกล่าวต่อไปนี้

SCAN -FUNCTION นี้จะเรียก FUNCTION SCAN1 และมี DELAY TIME สำหรับ DEBOUNC S/W FUNCTION นี้ จะทำการ SCAN KEY BOARD จนกว่าจะมีการกด KEY และจะเอาค่ารหัสของตำแหน่ง KEY ที่กด ที่ได้มาจาก FUNCTION SCAN1 มาแปลงเป็นค่ารหัสภายในของ KEY ที่กด แล้วเก็บไว้ใน REGISTER A.

SCAN1 -เป็น FUNCTION สำหรับ SCAN KEY และ DISPLAY 1 รอบ ถ้าตรวจสอบได้ว่ามีการกด KEY CARRY FLAG จะเป็น "0" แต่ถ้าไม่การกด KEY CARRY FLAG จะเป็น "1" ถ้าหากว่ามีการกด KEY ก็จะเก็บค่าของตำแหน่ง KEY ที่กดไว้ใน REGISTER A.

DISPP -เป็น FUNCTION ที่เปลี่ยนค่า รหัสภายใน ของ KEY ที่กด ที่เก็บไว้ใน REG.A เป็น DISPLAY PATTERN แล้วเก็บไว้ใน REG A.

SHIFT -เป็น FUNCTION สำหรับ เลื่อนตำแหน่งของการแสดงผลออกไป 1 หลัก เมื่อตรวจสอบพบว่ามีการกด KEY

SPACE -จะทำให้ DISPLAY PATTERN BLANK เมื่อ KEY ที่ถูกกดไม่ใช่ KEY 0-F

บทที่ 5
บทสรุป

จากการทดลองสร้างชุด INTERFACING TECHNIQUE ขึ้นมาเพื่อใช้ในการเรียนการสอนตามหลักสูตร สาขาวิชาชีพชั้นสูง (ปวส.) ในภาคปฏิบัติ วิชา เทคนิคการอินเทอร์เฟสไมโครคอมพิวเตอร์ (Micro computer Interfacing techique) โดยมีใบงานประกอบพอจะสรุปเป็นข้อดีข้อเสียออกเป็นส่วน ๆ ดังนี้คือ

1 เกี่ยวกับระบบการ อินเทอร์เฟส

เนื่องจากเป็นระบบพื้นฐานในการ INTERFACING กับ SINGLE BOARD MPF I-B จึงออกแบบให้แผ่น P.C.B มีขนาดเล็กสามารถบรรจุลงในฝาของกระเป๋าอีกข้างของ MPF-I ได้พอดี ผลที่ได้รับหลังการทดลองสร้างขึ้นมาใช้งานปรากฏผลดังนี้

1.1 ได้ทำการทดลองสอบตามใบงานทั้ง 10 หัวข้อดังนี้

- BASIC INPUT/OUTPUT PORT
- การใช้งาน 8255 เป็น INPUT/OUTPUT PORT
- SEVEN SEGMENT LED INTERFACING
- Z-80 INTERRUPT
- STEPPING MOTOR & DC MOTOR
- DOT MATRIX
- KEY BOARD AND DISPLAY

ผลการทดลองเป็นที่น่าพอใจ สามารถนำมาใช้ประกอบการเรียนการสอนได้

2 เกี่ยวกับอุปกรณ์ DRIVER พวก MOTOR ยังคงต้องใช้ PROTO BOARD ภายนอกมาต่อวงจรเพิ่มเติม ซึ่งยังเป็นข้อเสียของระบบอยู่ควรมีการพัฒนาต่อไป

3 เกี่ยวกับขนาดถูกบังคับอยู่ อุปกรณ์ DOT MATRIX , SEVEN SEGMENT มีจำนวนน้อยหลักไปควรเพิ่มเติมให้มากกว่านี้จะเห็นการแสดงผลหลังจากการ RUN PROGRAM ได้ชัดเจนขึ้น

4 จากชุด INTERFACE ที่สร้างขึ้นยังขาดการ INTERFACE แบบต่าง ๆ อีกเช่น การตรวจสอบสัญญาณซึ่งกันและกัน (HAND SHAKING) การเชื่อมต่อแบบอันดับ (SERIAL TRANSMISSION) และการนับกับการเป็นฐานเวลา (COUNTER/TIMER) ซึ่งจะต้องพัฒนาต่อไปให้สมบูรณ์ครอบคลุมเนื้อหาตามที่กล่าวทั้งหมดในโอกาสต่อไป

ภาคผนวก

5400/7400 Quadruple 2-Input Positive-NAND Gate

	Schottky TTL					High-Speed TTL					Low-Power Schottky TTL					Standard TTL					Low-Power TTL				
	Device Type	Package				Device Type	Package				Device Type	Package				Device Type	Package				Device Type	Package			
		C	P	M	CF		C	P	M	CF		C	P	M	CF		C	P	M	CF		C	P	M	CF
T. I.	SN54S00	J①W①	N①		W①	SN54H00	J①W①			W①	SN54LS00	J①W①			W①	SN5400	J①W①			W②	SN54L00	J①W①	N①	T①	T②
	SN74S00	J①W①	N①			SN74H00	J①W①	N①			SN74LS00	J①W①	N①			SN7400	J①W①	N①			SN74L00	J①W①	N①	T①	T②
FAIRCHILD	FJH5400/FJH5500	D①				FJH54H00/FJH74H00	D①			F②	74LS00 / 74SL00	W①			F①	FJH 5400/FJH 7400	D①			F②					
	FC74S00/FC95500	D①	P①			FC74H00/FC84H00	D①	P①			FC74LS00 / FC95500	W①	P①		F①	FC7400/FC94N00	D①	P①							
MOTOROLA						MC3100	L①			F②						MC5400	L①			F②					
						MC3000	L①	P①		F②	SN74LS00		P①			MC7400	L①	P①		F②					
N. S. C.						DM54H00	J①	N①			DM54LS00					DM5400	J①	N①		W②	DM54L00	J①	N①		F②
	DM74S00		N①			DM74H00	J①	N①			DM74LS00					DM7400	J①	N①			DM74L00	J①	N①		F②
PHILIPS	N74S00		①			OJH131/74H00		①			N74LS00		⑪			FJH131/7400		①							
SIGNETICS	N54S00	F①	A①		W①	S54H00	F①	A①		W②						S5400	F①	A①		W②					
	N74S00	F①	A①			N74H00	F①	A①			N74LS00		A①			N7400	F①	A①							
SIEMENS																FLH101		①							
FUJITSU						MB601	①	M①			74LS00		M①			MB400	①	M①							
HITACHI	HD74S00	①	P①								HD74LS00		P①			HD7400/HD7503	①	P①							
MITSUBISHI	M55000		P①								M74LS00		P①			M53200		P①							
NEC	μPB73S00	D①									74LS00		C①			μPB701	D①	C①							
TOSHIBA																TD3400A		P①							

Electrical Characteristics SN54LS00/SN74LS00

absolute maximum ratings over operating free-air temperature range

Supply voltage V_{CC}	7V	Operating free-air temperature range	SN54LS	−55°C to 125°C
Input voltage	7V		SN74LS	0°C to 70°C
Interemitter voltage	5.5V	Storage temperature range		−65°C to 150°C

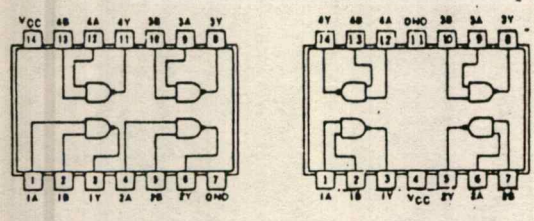
recommended operating conditions

	SN54LS00			SN74LS00			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage V_{CC}	4.5	5	5.5	4.75	5	5.25	V
High-level output current I_{OH}			−400			−400	μA
Low-level output current I_{OL}			4			8	mA
Operating free-air temperature T_A	−55		125	0		70	°C

electrical characteristics over recommended operating free-air temperature range

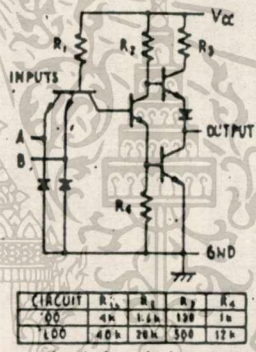
PARAMETER		TEST CONDITIONS†		MIN	TYP‡	MAX	UNIT
V_{IH}	High-level input voltage			2			V
V_{IL}	Low-level input voltage					0.8	V
V_I	Input clamp voltage	V_{CC} = MIN, I_I = −18mA				−1.5	V
V_{OH}	High-level output voltage	V_{CC} = MIN, I_{OH} = MAX, V_{IL} = V_{IL} max.		2.7	3.4		V
V_{OL}	Low-level output voltage	V_{CC} = MIN, I_{OL} = 4mA, V_{IH} = 2V.			0.2	0.4	V
I_I	Input current at maximum input voltage	V_{CC} = MAX, V_I = 7V				0.1	mA
I_{IH}	High-level input current	V_{CC} = MAX, V_{IH} = 2.7V				20	μA
I_{IL}	Low-level input current	V_{CC} = MAX, V_{IL} = 0.4V				−0.4	mA
I_{OS}	Short-circuit output current*	V_{CC} = MAX	54LS Family	−20		−100	mA
			74LS Family	−18		−100	
I_{CCH}	Supply current	V_{CC} = MAX	Total, outputs high		0.8	2	mA
I_{CCL}	Supply current		Total, outputs low		2.4	4.4	mA
I_{CC}	Supply current	V_{CC} = 5V	Average per gate (50% duty cycle)		0.4		mA
t_{PLH}	Propagation delay time, low-to-high-level output	V_{CC} = 5V,	T_A = 25°C,		9	15	ns
t_{PHL}	Propagation delay time, high-to-low-level output	C_L = 15pF,	R_L = 2kΩ		10	15	ns

Pin Assignments (Top View)



positive logic: $Y = \overline{AB}$

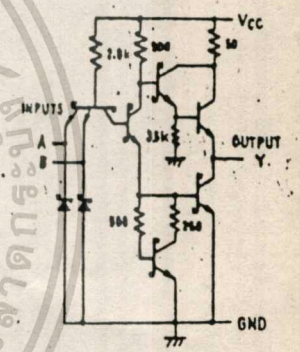
Schematics (each gate)



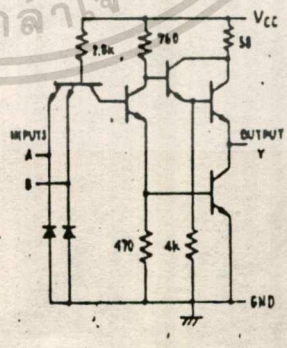
CIRCUIT	R_1	R_2	R_3	R_4
'00	4k	1.6k	130	1k
'L00	40k	20k	500	12k

Input clamp diodes not on SN54L'/SN74L' circuits.

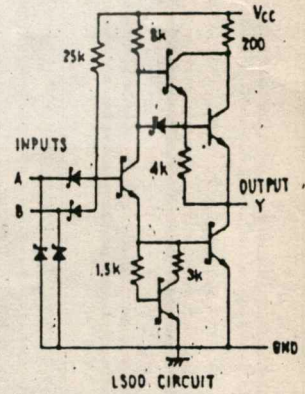
'00, 'L00 CIRCUITS



'S00 CIRCUIT



'H00 CIRCUIT



LS00 CIRCUIT

Resistor values shown are nominal and in ohms.

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.

‡ All typical values are at V_{CC} = 5V, T_A = 25°C.

* Not more than one output should be shorted at a time, and for SN54H'/SN74H' and SN54S'/SN74S', duration of short-circuit should not exceed 1 second.

54138/74138 3-Line-to-8-Line Decoder

	Schottky TTL					High-Speed TTL					Low-Power Schottky TTL					Standard TTL					Low-Power TTL				
	Device Type	Package C	P	M	CF	Device Type	Package C	P	M	CF	Device Type	Package C	P	M	CF	Device Type	Package C	P	M	CF	Device Type	Package C	P	M	CF
T.I.	SN54S138	J⑯			W⑯						SN54LS138	J⑯			W⑯										
	SN74S138	J⑯	N⑯								SN74LS138	J⑯	N⑯												
FAIRCHILD	/P93S138	P⑯									P93L138/P93LS138	P⑯			F⑯										
	PC74S138/PC93S138	P⑯									PC74LS138/PC93LS138	P⑯	P⑯		F⑯										
MOTOROLA																									
N.S.C.											SN74LS138		P⑯												
	DM74S138		⑯								DM74LS138		⑯												
PHILIPS											DM54LS138		⑯												
	N74S138		⑯																						
SIGNETICS	S54S138	F⑯	B⑯		W⑯						N74LS138		⑯												
	N74S138	F⑯	B⑯																						
SIEMENS											N74LS138		A⑯												
FUJITSU																									
HITACHI											74LS138		M⑯												
MITSUBISHI											HD74LS138		P⑯												
	M74S138		P⑯								M74LS138		P⑯												
NEC																									
											74LS138		C⑯												
TOSHIBA																									

Electrical Characteristics SN54LS138/SN74LS138

absolute maximum ratings over operating free-air temperature range

Supply voltage, V_{CC}	7V	Operating free-air temperature range	SN54LS	−55°C to 125°C
Input voltage	7V		SN74LS	0°C to 70°C
		Storage temperature range		−65°C to 150°C

recommended operating conditions

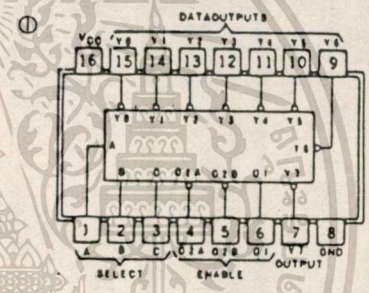
	SN54LS138			SN74LS138			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC}	4.5	5	5.5	4.75	5	5.25	V
High-level output current, I_{OH}			−400			−400	μA
Low-level output current, I_{OL}			4			8	mA
Operating free-air temperature, T_A	−55		125	0		70	°C

electrical characteristics over recommended operating free-air temperature range

	PARAMETER*	TEST CONDITIONS†		MIN	TYP‡	MAX	UNIT
V_{IH}	High-level input voltage			2			V
V_{IL}	Low-level input voltage					0.8	V
V_I	Input clamp voltage	V_{CC} = MIN, I_I = −18mA				−1.5	V
V_{OH}	High-level output voltage	V_{CC} = MIN, V_{IH} = 2V, V_{IL} = 0.8V, I_{OH} = −400μA	SN54LS	2.5	3.4		V
			SN74LS	2.7	3.4		
V_{OL}	Low-level output voltage	V_{CC} = MIN, V_{IH} = 2V, V_{IL} = 0.8V, I_{OL} = 8mA			0.35	0.5	V
I_I	Input current at maximum input voltage	V_{CC} = MAX, V_I = 7V				0.1	mA
I_{IH}	High-level input current	V_{CC} = MAX, V_I = 2.7V				20	μA
I_{IL}	Low-level input current	V_{CC} = MAX, V_I = 0.4V				−0.4	mA
I_{OS}	Short-circuit output current*	V_{CC} = MAX		−20		−100	mA
I_{CC}	Supply current	V_{CC} = MAX, Outputs enabled and open			0.3	18	mA
t_{PLH}	from Binary select to Any output	V_{CC} = 5V, T_A = 25°C, C_L = 15pF, R_L = 2kΩ	levels of delay 2		13	20	ns
t_{PHL}					27	41	
t_{PLH}			3		18	27	ns
t_{PHL}					26	39	
t_{PLH}	from Enable to Any output		2		12	18	ns
t_{PHL}					21	32	
t_{PLH}			3		17	26	ns
t_{PHL}					25	38	

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions for the applicable device type
‡ All typical values are at V_{CC} = 5V, T_A = 25°C
* Not more than one output should be shorted at a time, and duration of the short-circuit test should not exceed one second
t_{PLH} = propagation delay time, low-to-high-level output
t_{PHL} = propagation delay time, high-to-low-level output

Pin Assignment (Top View)



positive logic: see function table

Function Table

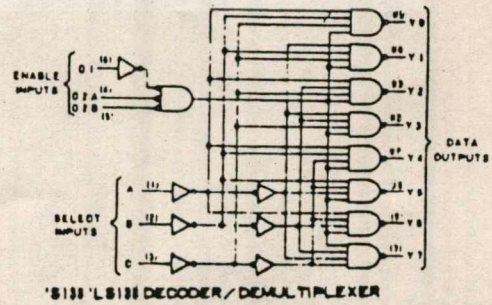
'S138 'LS138

INPUTS ENABLE G1	G2*	SELECT C	B	A	OUTPUTS Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

* G2 = G2A + G2B

H = high level, L = low level, X = irrelevant

Functional Block Diagram



'S138 'LS138 DECODER/DEMULTIPLEXER

54245/74245 Octal Bus Transceivers with 3-state Outputs

	Schottky TTL					High-Speed TTL					Low-Power Schottky TTL					Standard TTL					Low-Power T				
	Device Type	Package				Device Type	Package				Device Type	Package				Device Type	Package				Device Type	Pa			
		C	P	M	CF		C	P	M	CF		C	P	M	CF		C	P	M	CF		C	P		
T. I.											SN54LS245	J		W	W										
FAIRCHILD											SN74LS245	J	N												
MOTOROLA																									
N. S. C																									
PHILIPS																									
SIGNETICS																									
SIEMENS																									
FUJITSU																									
HITACHI																									
MITSUBISHI																									
NEC																									
TOSHIBA																									

Electrical Characteristics SN54LS245/SN74LS245

absolute maximum ratings over operating free-air temperature range

Supply voltage, V_{CC}	7 V	Operating free-air temperature range	SN54LS	−55°C to 125°C
Input voltage	7 V		SN74LS	0°C to 70°C
		Storage temperature range		−65°C to 150°C

recommended operating conditions

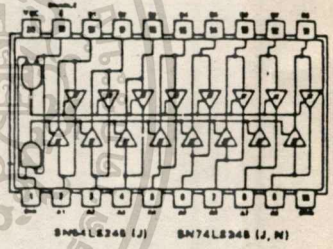
	SN54LS245			SN74LS245			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC}	4.5	5	5.5	4.75	5	5.25	V
High-level output current, I_{OH}			−12			−15	mA
Low-level output current, I_{OL}			12			24	mA
Operating free-air temperature, T_A	−55		125	0		70	°C

electrical characteristics over recommended operating free-air temperature range

PARAMETER		TEST CONDITIONS†		SN74LS245 MIN	SN74LS245 TYP‡	SN74LS245 MAX	UNIT
High-level input voltage				2			V
Low-level input voltage						0.8	V
Input clamp voltage		V_{CC} = MIN,	I_I = −18mA			−1.5	V
Hysteresis ($V_{T+} - V_{T-}$) A or B input		V_{CC} = MIN			0.2	0.4	V
High-level output voltage		V_{CC} = MIN, V_{IH} = 2V, V_{IL} = V_{IL} max	I_{OH} = −3mA	2.4	3.4		V
			I_{OH} = MAX	2			
Low-level output voltage		V_{CC} = MIN, V_{IH} = 2V, V_{IL} = V_{IL} max	I_{OL} = 12mA			0.4	V
			I_{OL} = 24mA			0.5	
Off-state output current, high-level voltage applied		V_{CC} = MAX, G at 2V	V_O = 2.7 V			10	μA
Off-state output current, low-level voltage applied			V_O = 0.4V			−200	
Input current at maximum input voltage	A or B	V_{CC} = MAX,	V_I = 5.5V			0.1	mA
	DIR or G		V_I = 7V			0.1	
High-level input current		V_{CC} = MAX,	V_{IH} = 2.7V			20	μA
Low-level input current		V_{CC} = MAX,	V_{IL} = 0.4V			−0.2	mA
Short-circuit output current		V_{CC} = MAX		−40		−225	mA
Supply current	Total, outputs high	V_{CC} = MAX, Outputs open			48	70	mA
	Total, outputs low				62	90	
	Outputs at Hi-Z				64	95	

switching characteristics, V_{CC} 5V, T_A 25°C

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
Propagation delay time, low-to-high-level output	C_L = 45pF, R_L = 667Ω, See Note 2		8	12	ns
Propagation delay time, high-to-low-level output			8	12	ns
Output enable time to low level			27	40	ns
Output enable time to high level			25	40	ns
Output disable time from low level	C_L = 5pF, R_L = 667Ω, See Note 2		15	25	ns
Output disable time from high level			15	25	ns



SN54LS245 (J) SN74LS245 (J, N)

conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions

typical values are at V_{CC} = 5V, T_A = 25°C.

not more than one output should be shorted at a time, and duration of the short-circuit should not not exceed one second

54373/74373 Octal D-Type Transparent Latches and Edge-Triggered Flip-Flops

	Schottky TTL					High-Speed TTL					Low-Power Schottky TTL					Standard TTL					Low-Power TTL				
	Device Type	Package				Device Type	Package				Device Type	Package				Device Type	Package				Device Type	Package			
		C	P	M	CF		C	P	M	CF		C	P	M	CF		C	P	M	CF		C	P	M	CF
	SN54S373	J①									SN54LS373	J①													
	SN74S373	J①N①									SN74LS373	J①N①													
FAIRCHILD																									
MOTOROLA																									
N.S.C.																									
PHILIPS																									
SIGNETICS																									
SIEMENS																									
MITSU																									
HITACHI																									
MITSUBISHI																									
?																									
TOSHIBA																									

Electrical Characteristics SN54LS373/SN74LS373

Absolute maximum ratings over operating free-air temperature range

Supply voltage, V_{CC}	7V	Operating free-air temperature range	SN54LS	−55°C to 125°C
Input voltage	7V		SN74LS	0°C to 70°C
		Storage temperature range		−65°C to 150°C

Recommend operating conditions

		SN54LS373			SN74LS373			UNIT
		MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC}		4.5	5	5.5	4.75	5	5.25	V
High-level output current, I_{OH}				−1			−2.6	mA
Low-level output voltage, V_{OH}				0.8			0.8	V
Pulse width, t_w	Clock enable high	15			15			ns
	Clock enable high	15			15			ns
Setup time, t_{SU}		0↑			0↑			ns
Hold time, t_{hold}		10↓			10↓			ns
Operating free-air temperature, T_A		−55		125	0		70	°C

Electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

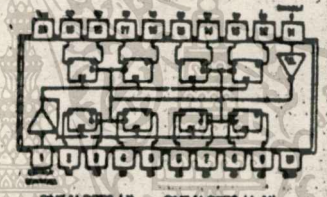
PARAMETER	TEST CONDITIONS‡		MIN	TYP‡	MAX	UNIT
High-level input voltage			2			V
Low-level input voltage					0.8	V
Input clamp voltage	V_{CC} = MIN, I_I = −18mA				−1.5	V
High-level output voltage	V_{CC} = MIN, V_{IH} = 2V, V_{IL} = V_{IL} max, I_{OH} = MAX		2.4	3.1		V
Low-level output voltage	V_{CC} = MIN, V_{IH} = 2V, V_{IL} = V_{IL} max	I_{OL} = 24mA		0.35	0.5	V
Off-state output current, high-level voltage applied	V_{CC} = MAX, V_{IH} = 2V, V_O = 2.7V				20	μA
Off-state output current, low-level voltage applied	V_{CC} = MAX, V_{IH} = 2V, V_O = 0.4V				−20	μA
Input current at maximum input voltage	V_{CC} = MAX, V_I = 7V				0.1	mA
High-level input current	V_{CC} = MAX, V_I = 2.7V				20	μA
Low-level input current	V_{CC} = MAX, V_I = 0.4V				−0.4	mA
Short-circuit output current §	V_{CC} = MAX		−30		−130	mA
Supply current	V_{CC} = MAX, Output control at 4.5V	'LS373		24	40	mA

Switching characteristics, V_{CC} = 5V, T_A = 25°C

PARAMETER	FROM (INPUT)	TO (OUTPUT)	TEST CONDITIONS	MIN	TYP	MAX	UNIT
f_{max}							MHz
t_{PLH}	Data	Any Q	C_L = 45pF, R_L = 667Ω, See Notes 2 and 3		12	18	ns
t_{PHL}					12	18	
t_{PLH}	Clock or enable	Any Q			20	30	ns
t_{PHL}					15	30	
t_{PZH}	Output Control	Any Q			15	35	ns
t_{PZL}					20	35	
t_{PHZ}	Output Control	Any Q	C_L = 5pF, R_L = 667Ω, See Note 3		12	20	ns
t_{PLZ}					15	25	

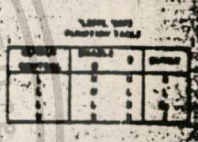
‡ For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.
§ Typical values are at V_{CC} = 5V, T_A = 25°C.
¶ Not more than one output should be shorted at a time and duration of the short circuit should not exceed one second.

Pin Assignments (Top View)

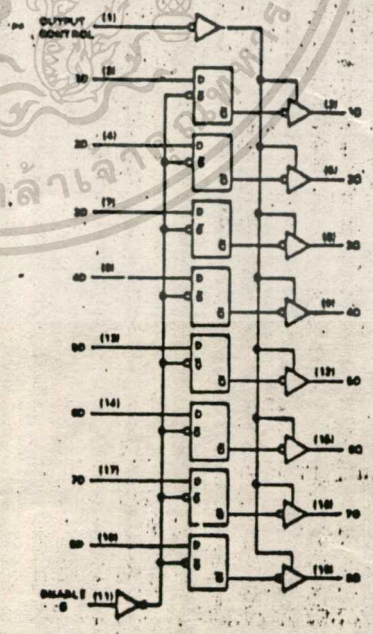


SN54LS373 (J)
SN54S373 (J)

SN74LS373 (J, N)
SN74S373 (J, N)



'LS373, 'S373
TRANSPARENT LATCHES



NOTES: 2. Maximum clock frequency is tested with all outputs loaded.
3. See load circuits and waveforms on page 3-11.

f_{max} = maximum clock frequency
t_{PLH} = propagation delay time, low-to-high-level output
t_{PHL} = propagation delay time, high-to-low-level output
t_{PZH} = output enable time to high level
t_{PZL} = output enable time to low level
t_{PHZ} = output disable time from high level
t_{PLZ} = output disable time from low level

intel

8255A/8255A-5
PROGRAMMABLE PERIPHERAL INTERFACE

- MCS-85™ Compatible 8255A-5
- 24 Programmable I/O Pins
- Completely TTL Compatible
- Fully Compatible with Intel® Microprocessor Families
- Improved Timing Characteristics
- Direct Bit Set/Reset Capability Easing Control Application Interface
- Reduces System Package Count
- Improved DC Driving Capability
- Available in EXPRESS
  - —Standard Temperature Range
  - —Extended Temperature Range

The Intel® 8255A is a general purpose programmable I/O device designed for use with Intel® microprocessors. It has 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. In the first mode (MODE 0), each group of 12 I/O pins may be programmed in sets of 4 to be input or output. In MODE 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining 4 pins, 3 are used for handshaking and interrupt control signals. The third mode of operation (MODE 2) is a bidirectional bus mode which uses 8 lines for a bidirectional bus, and 5 lines, borrowing one from the other group, for handshaking.

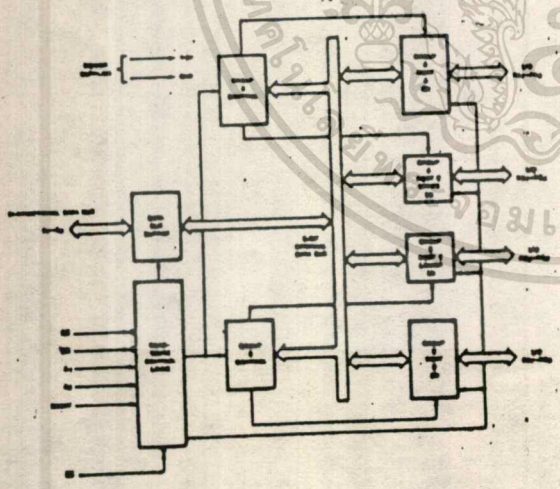


Figure 1. 8255A Block Diagram

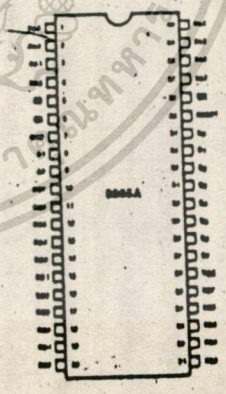


Figure 2. Pin Configuration

8255A FUNCTIONAL DESCRIPTION

General

The 8255A is a programmable peripheral interface (PPI) device designed for use in Intel® microcomputer systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the microcomputer system bus. The functional configuration of the 8255A is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

This 3-state bidirectional 8-bit buffer is used to interface the 8255A 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.

($\overline{CS}$)

Chip Select. A "low" on this input pin enables the communiction between the 8255A and the CPU.

($\overline{RD}$)

Read. A "low" on this input pin enables the 8255A to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to "read from" the 8255A.

($\overline{WR}$)

Write. A "low" on this input pin enables the CPU to write data or control words into the 8255A.

(A_0 and A_1)

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 registers. They are normally connected to the least significant bits of the address bus (A_0 and A_1).

8255A BASIC OPERATION

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	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
					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 – 3-STATE
1	1	0	1	0	ILLEGAL CONDITION
X	X	1	1	0	DATA BUS – 3-STATE

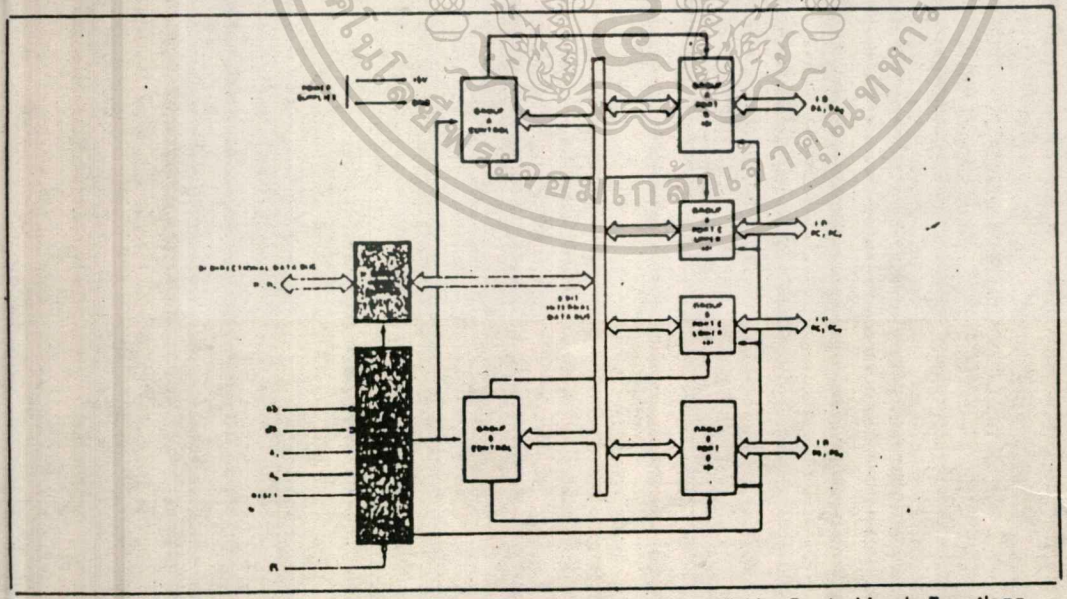


Figure 3. 8255A Block Diagram Showing Data Bus Buffer and Read/Write Control Logic Fupctions

intel

8255A/8255A-5
PROGRAMMABLE PERIPHERAL INTERFACE

- MCS-85™ Compatible 8255A-5
- 24 Programmable I/O Pins
- Completely TTL Compatible
- Fully Compatible with Intel® Microprocessor Families
- Improved Timing Characteristics
- Direct Bit Set/Reset Capability Easing Control Application Interface
- Reduces System Package Count
- Improved DC Driving Capability
- Available in EXPRESS
  - —Standard Temperature Range
  - —Extended Temperature Range

The Intel® 8255A is a general purpose programmable I/O device designed for use with Intel® microprocessors. It has 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. In the first mode (MODE 0), each group of 12 I/O pins may be programmed in sets of 4 to be input or output. In MODE 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining 4 pins, 3 are used for handshaking and interrupt control signals. The third mode of operation (MODE 2) is a bidirectional bus mode which uses 8 lines for a bidirectional bus, and 5 lines, borrowing one from the other group, for handshaking.

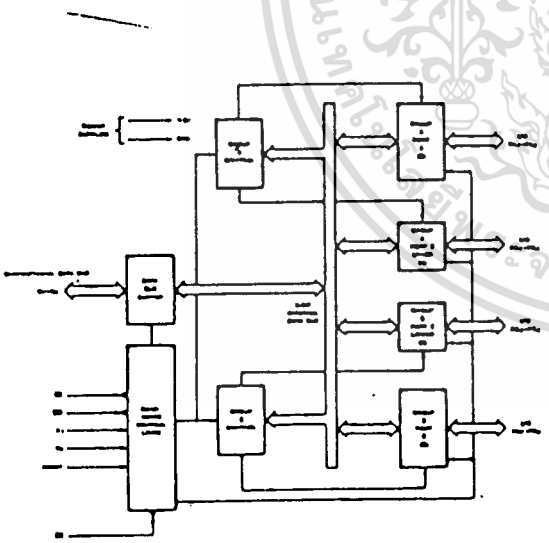


Figure 1. 8255A Block Diagram

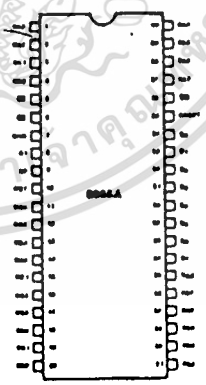


Figure 2. Pin Configuration

8255A FUNCTIONAL DESCRIPTION

General

The 8255A is a programmable peripheral interface (PPI) device designed for use in Intel® microcomputer systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the microcomputer system bus. The functional configuration of the 8255A is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

This 3-state bidirectional 8-bit buffer is used to interface the 8255A 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.

($\overline{CS}$)

Chip Select. A "low" on this input pin enables the communiction between the 8255A and the CPU.

($\overline{RD}$)

Read. A "low" on this input pin enables the 8255A to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to "read from" the 8255A.

($\overline{WR}$)

Write. A "low" on this input pin enables the CPU to write data or control words into the 8255A.

(A_0 and A_1)

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 registers. They are normally connected to the least significant bits of the address bus (A_0 and A_1).

8255A BASIC OPERATION

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	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
					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 – 3-STATE
1	1	0	1	0	ILLEGAL CONDITION
X	X	1	1	0	DATA BUS – 3-STATE

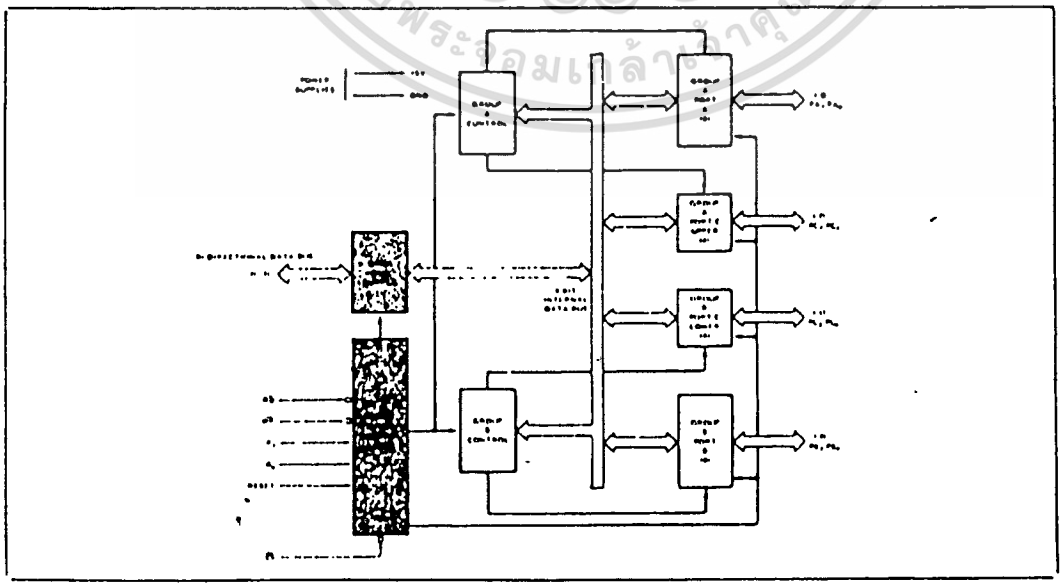


Figure 3. 8255A Block Diagram Showing Data Bus Buffer and Read/Write Control Logic Fuactions

(RESET)

Reset. A "high" on this input clears the control register and all ports (A, B, C) are set to the input mode.

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 8255A. The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 8255A.

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 Only be written into. No Read operation of the Control Word Register is allowed.

Ports A, B, and C

The 8255A contains three 8-bit ports (A, B, and C). All can be configured in 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 8255A.

Port A. One 8-bit data output latch/buffer and one 8-bit data input latch.

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

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 outputs and status signal inputs in conjunction with ports A and B.

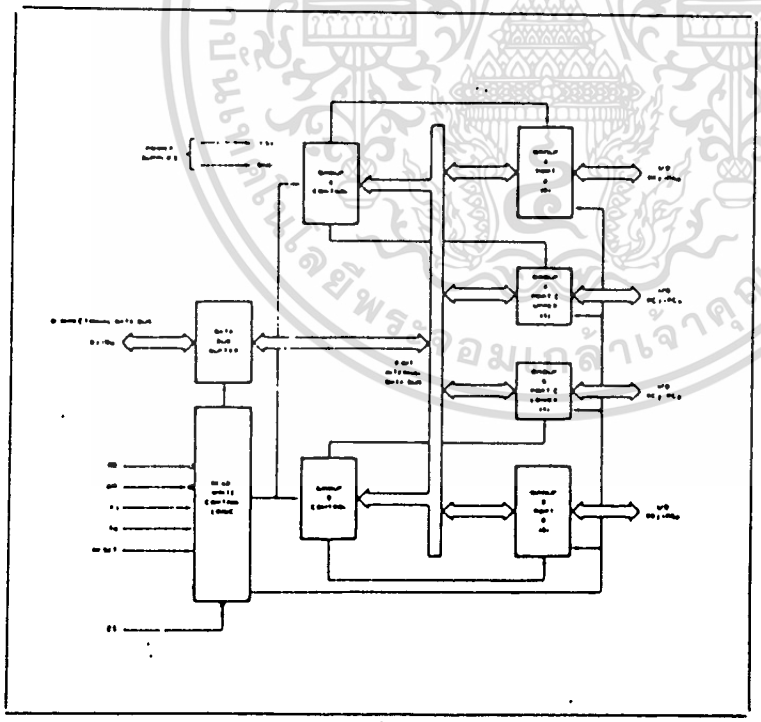
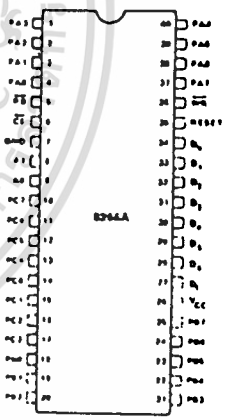


Figure 4. 8225A Block Diagram Showing Group A and Group B Control Functions

PIN CONFIGURATION



PIN NAMES

D_7-D_0	DATA BUS (BI-DIRECTIONAL)
RESET	RESET INPUT
$\overline{CS}$	CHIP SELECT
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A0, A1	PORT ADDRESS
PA7-PA0	PORT A (BIT)
PB7-PB0	PORT B (BIT)
PC7-PC0	PORT C (BIT)
V_{CC}	+5 VOLTS
GND	0 VOLTS

8255A 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 (i.e., all 24 lines will be in the high impedance state). After the reset is removed the 8255A can remain in the input mode with no additional initialization required. During the execution of the system program any of the other modes may be selected using a single output instruction. This allows a single 8255A to service a variety of peripheral devices with a simple software maintenance routine.

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.

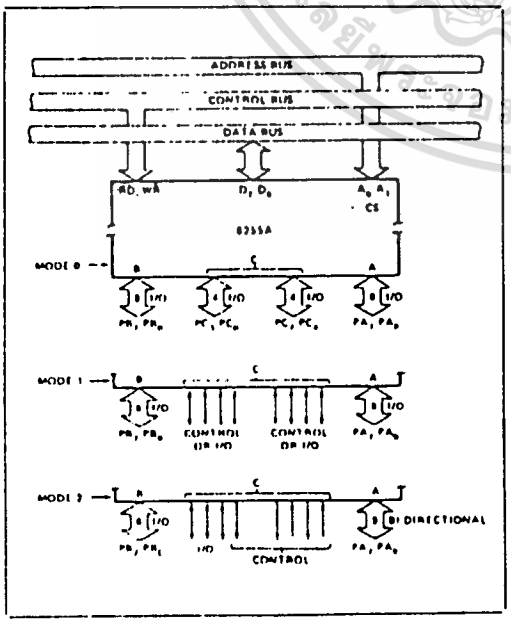


Figure 5. Basic Mode Definitions and Bus Interface

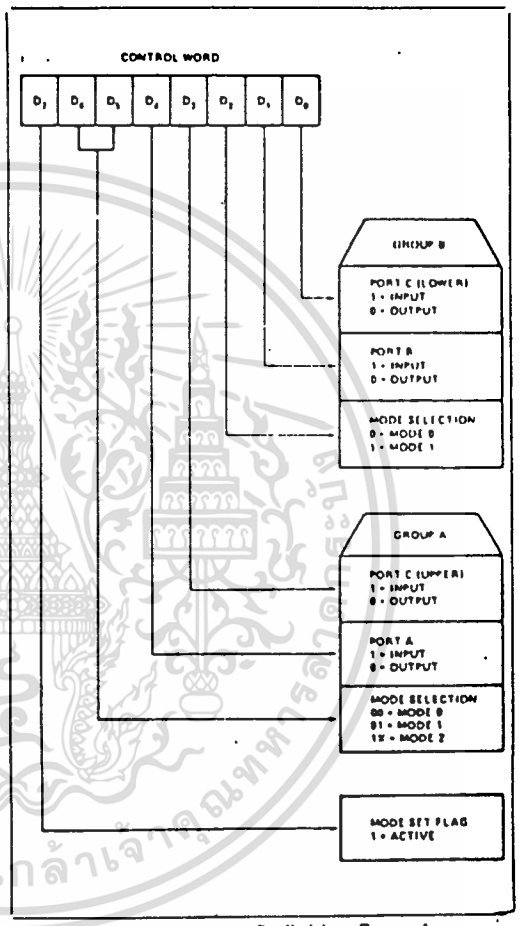


Figure 6. Mode Definition Format

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 8255A 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

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.

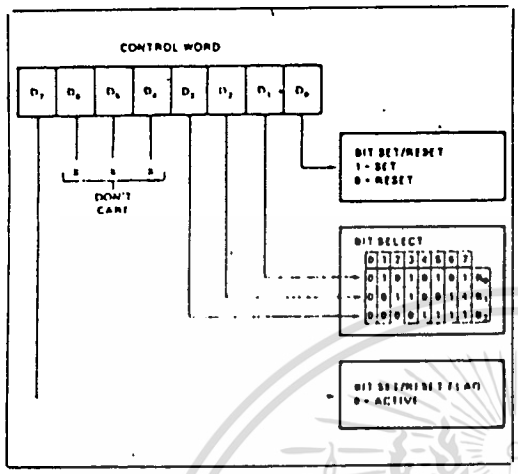


Figure 7. Bit Set/Reset Format

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 data output ports.

Interrupt Control Functions

When the 8255A 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 disallow or allow a specific I/O device to interrupt the CPU 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

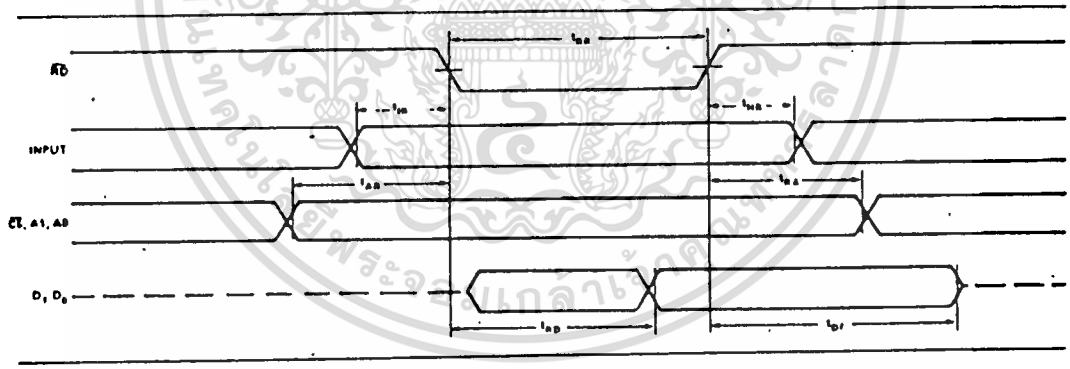
Note: All Mask flip-flops are automatically reset during mode selection and device Reset.

Operating Modes

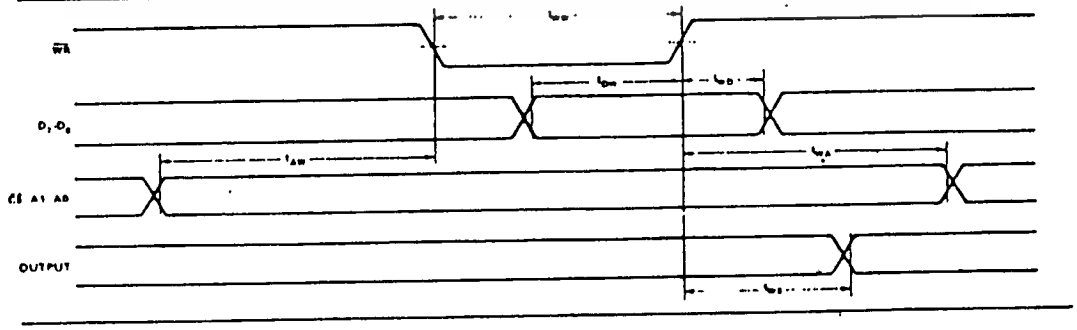
MODE 0 (Basic Input/Output). This functional configuration provides simple input and output operations for each of the three ports. No "handshaking" is required, data is simply written to or read from a specified port.

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 are possible in this Mode.



MODE 0 (Basic Input)

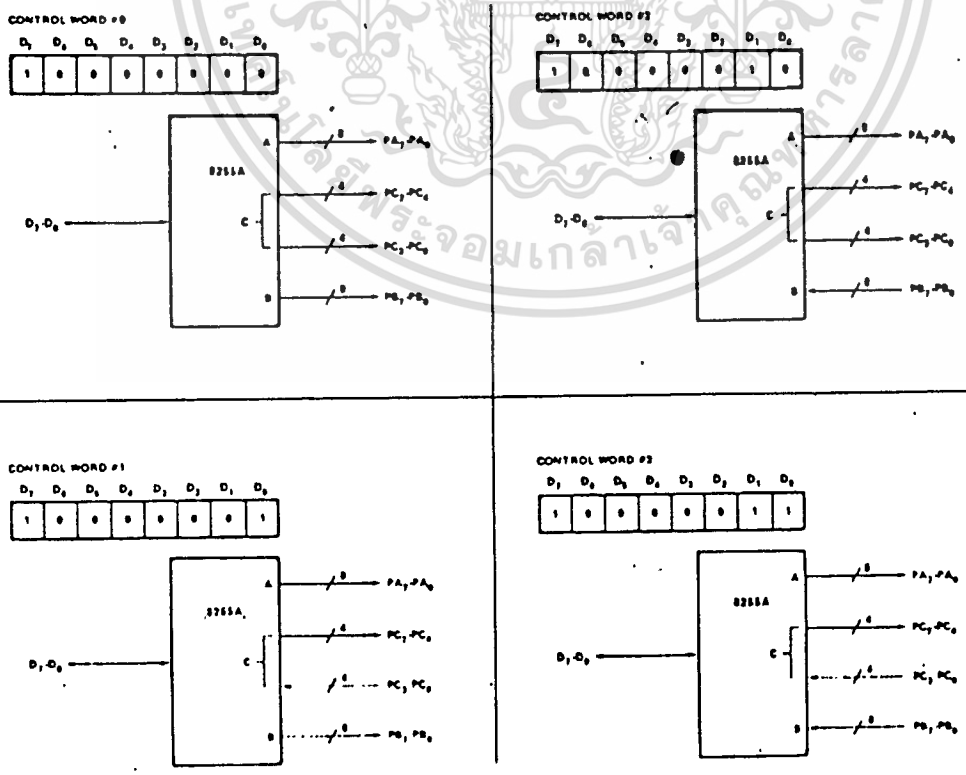


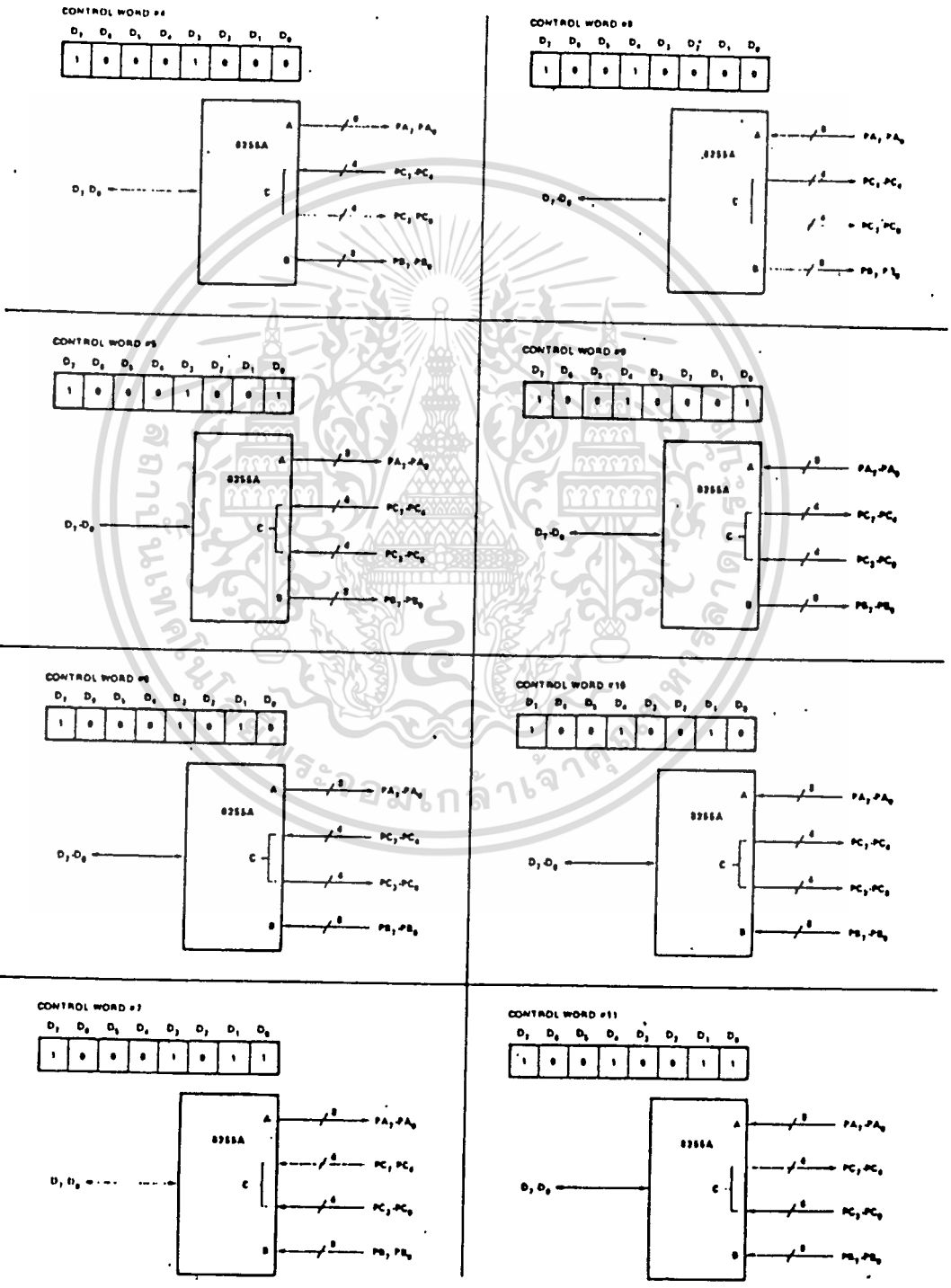
MODE 0 (Basic Output)

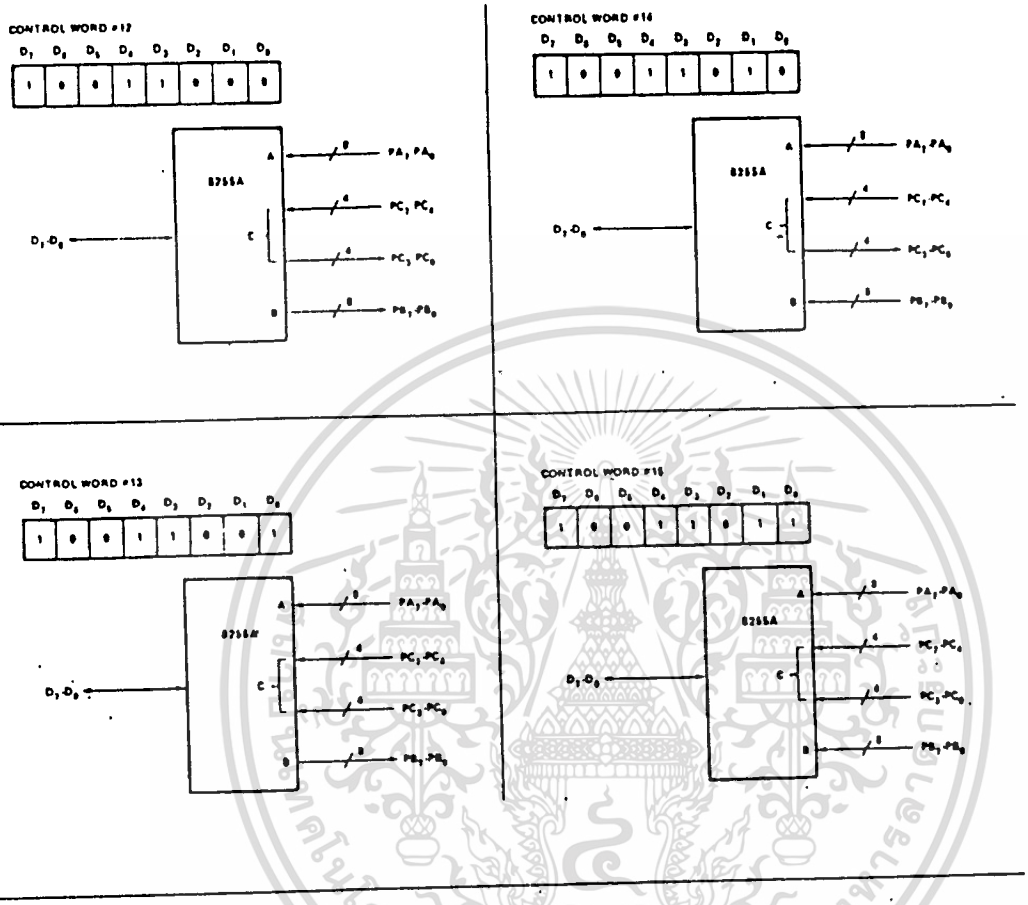
MODE 0 Port Definition

A		B		GROUP A			GROUP B	
D_4	D_3	D_1	D_0	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

MODE 0 Configurations







Operating Modes

MODE 1 (Strobed Input/Output). This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or "handshaking" signals. In mode 1, port A and Port B use the lines on port C to generate or accept these "handshaking" signals.

Mode 1 Basic Functional Definitions:

- Two Groups (Group A and Group B)
- Each group contains one 8-bit data port and one 4-bit control/data port.
- The 8-bit data port can be either input or output. Both inputs and outputs are latched.
- The 4-bit port is used for control and status of the 8-bit data port.

Input Control Signal Definition

$\overline{STB}$ (Strobe Input). A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F)

A "high" on this output indicates that the data has been loaded into the input latch; in essence, an acknowledgement. IBF is set by STB input being low and is reset by the rising edge of the RD input.

INTR (Interrupt Request)

A "high" on this output can be used to interrupt the CPU when an input device is requesting service. INTR is set by the $\overline{STB}$ is a "one", IBF is a "one" and INTE is a "one". It is reset by the falling edge of $\overline{RD}$. This procedure allows an input device to request service from the CPU by simply strobing its data into the port.

INTE A

Controlled by bit set/reset of PC_4.

INTE B

Controlled by bit set/reset of PC_2.

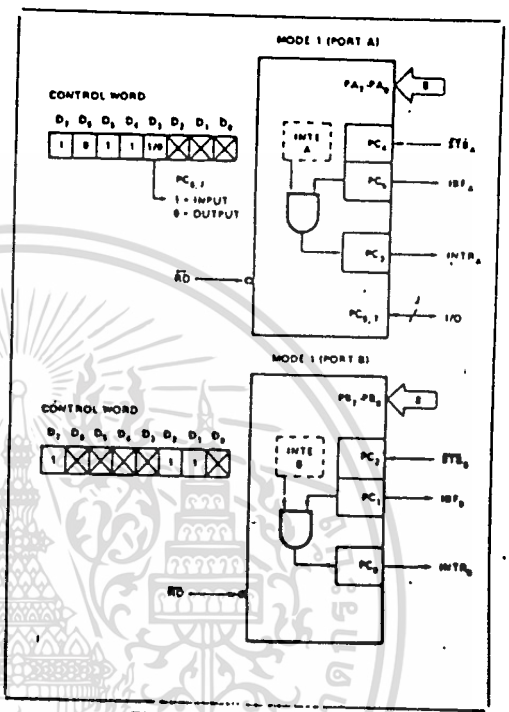


Figure 8. MODE 1 Input

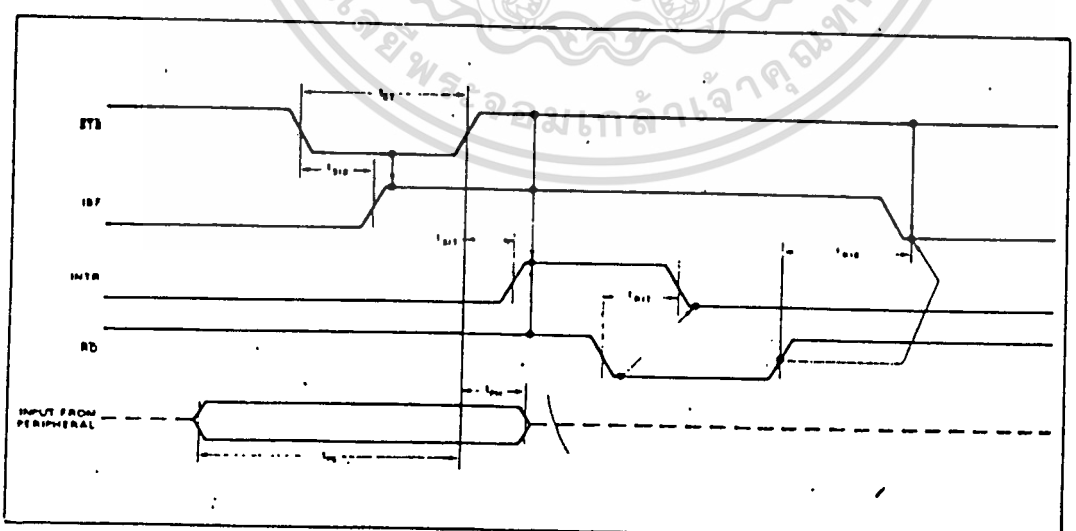


Figure 9. MODE 1 (Strobed Input)

Output Control Signal Definition

OBF (Output Buffer Full F/F). The OBF output will go "low" to indicate that the CPU has written data out to the specified port. The OBF F/F will be set by the rising edge of the WR input and reset by ACK input being low.

ACK (Acknowledge Input). A "low" on this input informs the 8255A that the data from port A or port B has been accepted. In essence, a response from the peripheral device indicating that it has recieved the data output by the CPU.

INTR (Interrupt Request). A "high" on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set when ACK is a "one", OBF is a "one", and INTE is a "one". It is reset by the falling edge of WR.

INTR (Interrupt Request). A "high" on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set when ACK is a "one", OBF is a "one", and INTE is a "one". It is reset by the falling edge of WR.

INTE A

Controlled by bit set/reset of PC_6.

INTE B

Controlled by bit set/reset of PC_2.

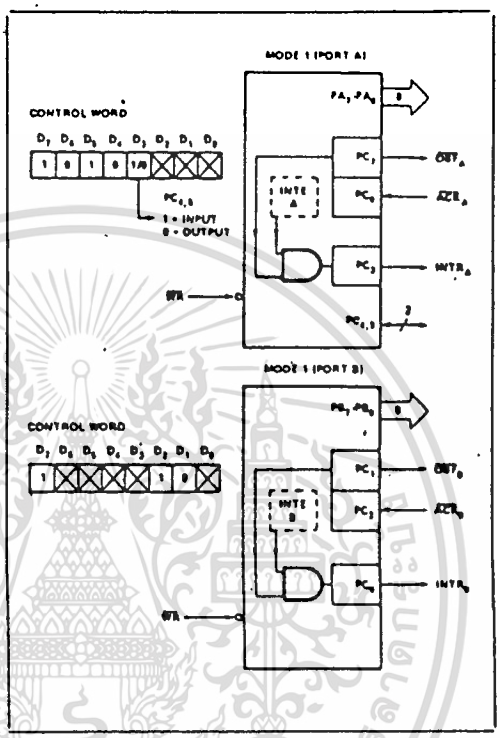


Figure 10. MODE 1 Output

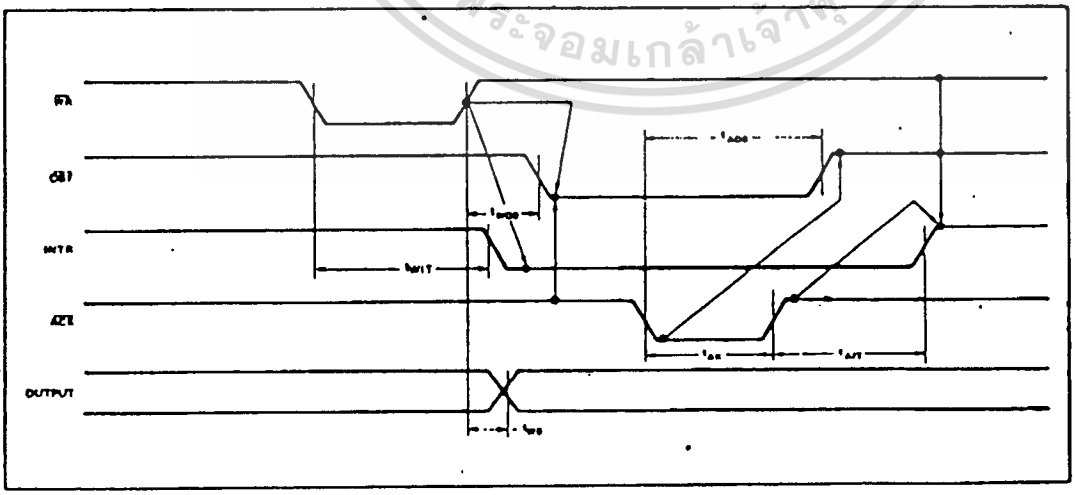


Figure 11. Mode 1 (Strobed Output)

Combinations of MODE 1

Port A and Port B can be individually defined as input or output in Mode 1 to support a wide variety of strobed I/O applications.

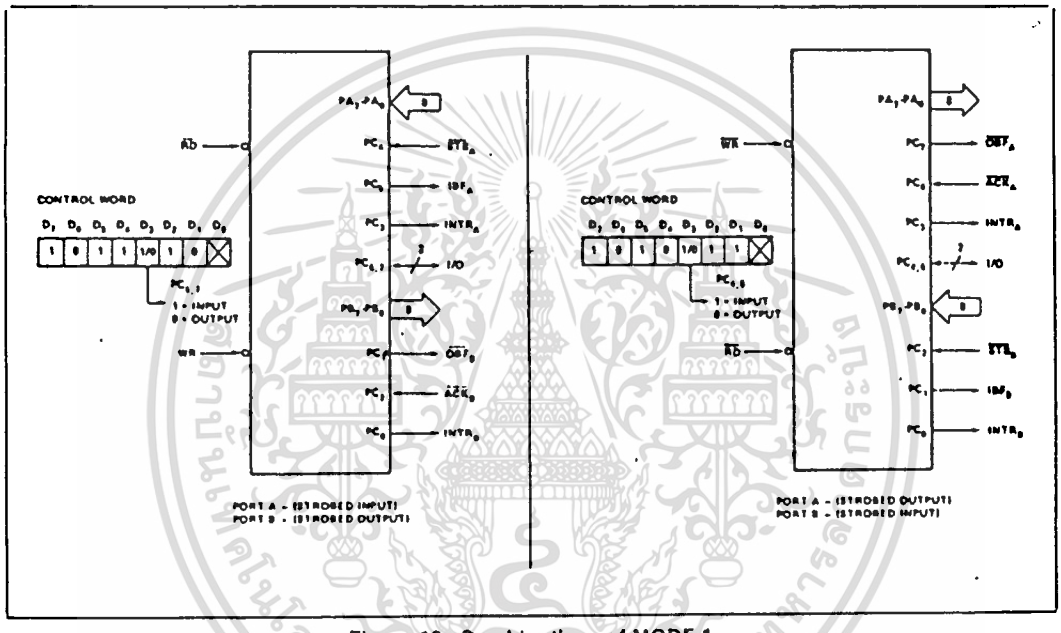


Figure 12. Combinations of MODE 1

Operating Modes

MODE 2 (Strobed Bidirectional Bus I/O). This functional configuration provides a means for communicating with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bidirectional bus I/O). "Handshaking" signals are provided to maintain proper bus flow discipline in a similar manner to MODE 1. Interrupt generation and enable/disable functions are also available.

MODE 2 Basic Functional Definitions:

- Used in Group A only.
- One 8-bit, bi-directional bus Port (Port A) and a 5-bit control Port (Port C).
- Both inputs and outputs are latched.
- The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A).

Bidirectional Bus I/O Control Signal Definition

INTR (Interrupt Request). A high on this output can be used to interrupt the CPU for both input or output operations.

Output Operations

$\overline{OBF}$ (Output Buffer Full). The $\overline{OBF}$ output will go "low" to indicate that the CPU has written data out to port A.

$\overline{ACK}$ (Acknowledge). A "low" on this input enables the tri-state output buffer of port A to send out the data. Otherwise, the output buffer will be in the high impedance state.

INTE 1 (The INTE Flip-Flop Associated with OBF). Controlled by bit set/reset of PC_6.

Input Operations

$\overline{STB}$ (Strobe Input). A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F). A "high" on this output indicates that data has been loaded into the input latch.

INTE 2 (The INTE Flip-Flop Associated with IBF). Controlled by bit set/reset of PC_4.

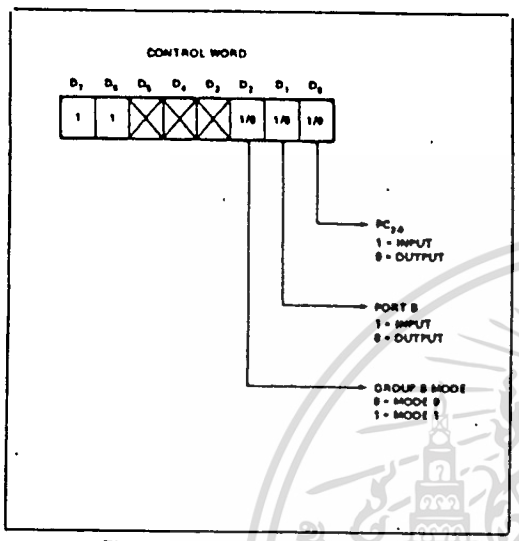


Figure 13. MODE Control Word

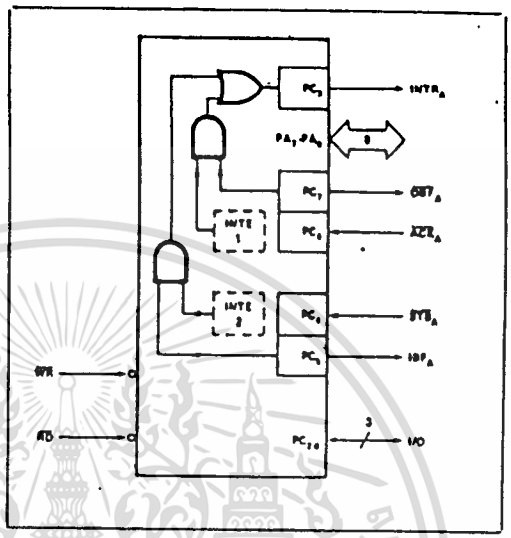


Figure 14. MODE 2

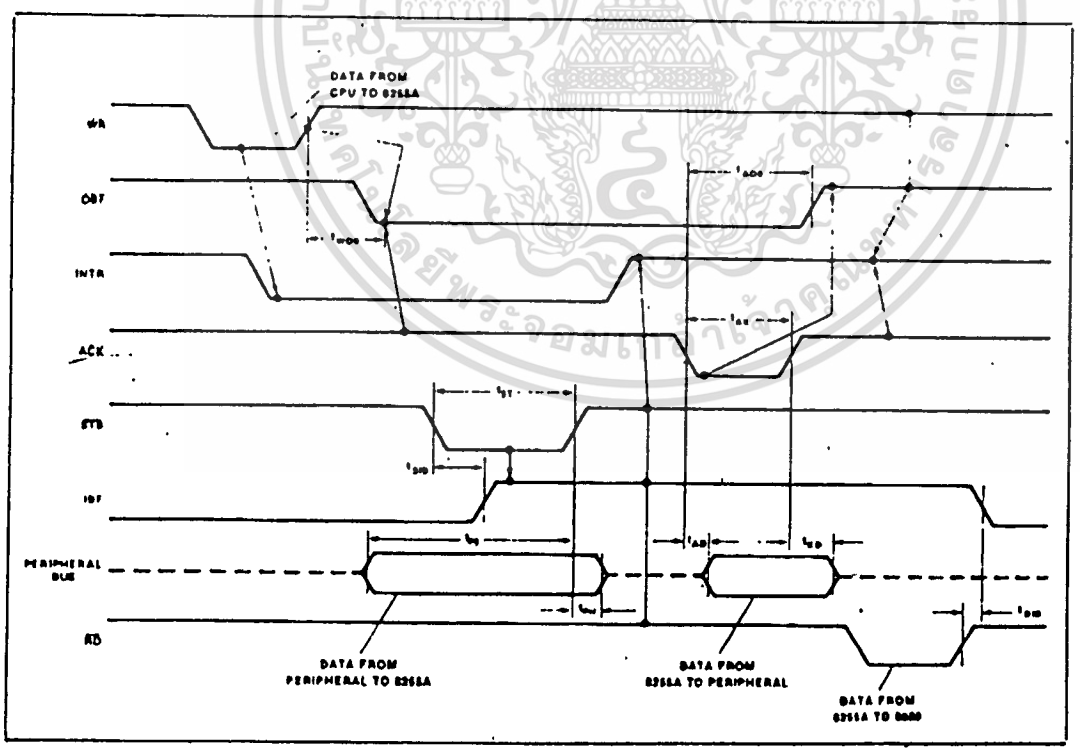


Figure 15. MODE 2 (Bidirectional)

NOTE Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ and $\overline{STB}$ occurs before $\overline{RD}$ is permissible.
(INTR = IBF · $\overline{MASK}$ · $\overline{STB}$ · $\overline{RD}$ + $\overline{OBF}$ · $\overline{MASK}$ · $\overline{ACK}$ · $\overline{WR}$)

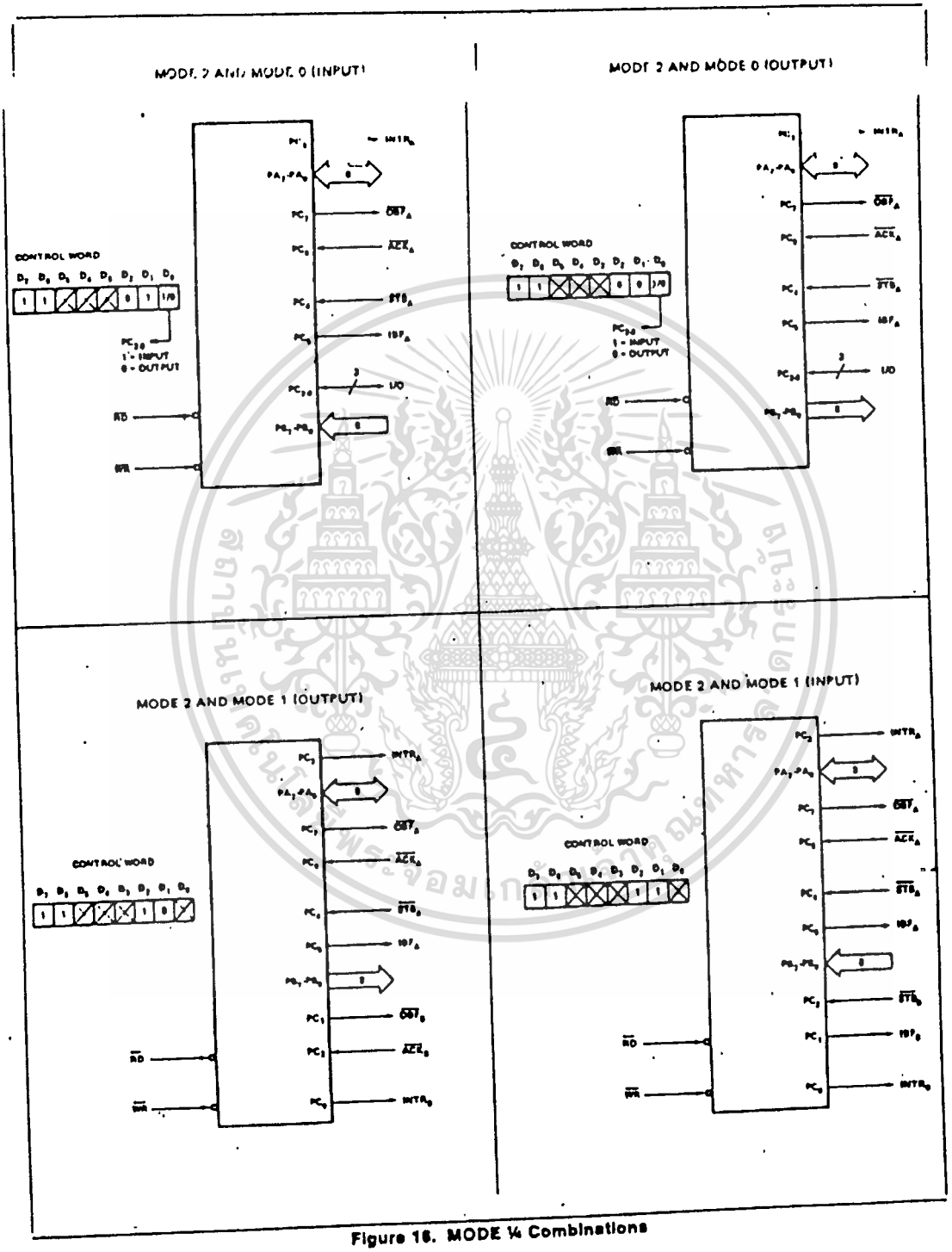


Figure 16. MODE 1/4 Combinations

Mode Definition Summary

	MODE 0		MODE 1		MODE 2	
	IN	OUT	IN	OUT	GROUP A ONLY	
PA_0	IN	OUT	IN	OUT	↔	
PA_1	IN	OUT	IN	OUT	↔	
PA_2	IN	OUT	IN	OUT	↔	
PA_3	IN	OUT	IN	OUT	↔	
PA_4	IN	OUT	IN	OUT	↔	
PA_5	IN	OUT	IN	OUT	↔	
PA_6	IN	OUT	IN	OUT	↔	
PA_7	IN	OUT	IN	OUT	↔	
PB_0	IN	OUT	IN	OUT	—	MODE 0 OR MODE 1 ONLY
PB_1	IN	OUT	IN	OUT	—	
PB_2	IN	OUT	IN	OUT	—	
PB_3	IN	OUT	IN	OUT	—	
PB_4	IN	OUT	IN	OUT	—	
PB_5	IN	OUT	IN	OUT	—	
PB_6	IN	OUT	IN	OUT	—	
PB_7	IN	OUT	IN	OUT	—	
PC_0	IN	OUT	$INTR_B$	$INTR_B$	I/O	
PC_1	IN	OUT	IBF_B	$\overline{OBF}_B$	I/O	
PC_2	IN	OUT	$\overline{STB}_B$	$\overline{ACK}_B$	I/O	
PC_3	IN	OUT	$INTR_A$	$INTR_A$	$INTR_A$	
PC_4	IN	OUT	$\overline{STB}_A$	I/O	$\overline{STB}_A$	
PC_5	IN	OUT	IBF_A	I/O	IBF_A	
PC_6	IN	OUT	I/O	$\overline{ACK}_A$	$\overline{ACK}_A$	
PC_7	IN	OUT	I/O	$\overline{OBF}_A$	$\overline{OBF}_A$	

Special Mode Combination Considerations

There are several combinations of modes when not all of the bits in Port C are used for control or status. The remaining bits can be used as follows:

If Programmed as Inputs –
All input lines can be accessed during a normal Port C read.

If Programmed as Outputs –
Bits in C upper (PC_7-PC_4) must be individually accessed using the bit set/reset function.

Bits in C lower (PC_3-PC_0) can be accessed using the bit set/reset function or accessed as a threesome by writing into Port C.

Source Current Capability on Port B and Port C

Any set of eight output buffers, selected randomly from Ports B and C can source 1mA at 1.5 volts. This feature allows the 8255 to directly drive Darlington type drivers and high-voltage displays that require such source current.

Reading Port C Status

In Mode 0, Port C transfers data to or from the peripheral device. When the 8255 is programmed to function in Modes 1 or 2, Port C generates or accepts "hand-shaking" signals with the peripheral device. Reading the contents of Port C allows the programmer to test or verify the "status" of each peripheral device and change the program flow accordingly.

There is no special instruction to read the status information from Port C. A normal read operation of Port C is executed to perform this function.

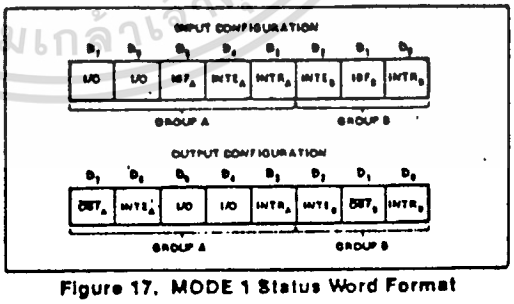


Figure 17. MODE 1 Status Word Format

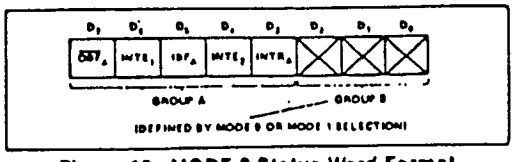


Figure 18. MODE 2 Status Word Format

APPLICATIONS OF THE 8255A

The 8255A is a very powerful tool for interfacing peripheral equipment to the microcomputer system. It represents the optimum use of available pins and is flexible enough to interface almost any I/O device without the need for additional external logic.

Each peripheral device in a microcomputer system usually has a "service routine" associated with it. The routine manages the software interface between the device and the CPU. The functional definition of the 8255A is programmed by the I/O service routine and becomes an extension of the system software. By examining the I/O devices interface characteristics for both data transfer and timing, and matching this information to the examples and tables in the detailed operational description, a control word can easily be developed to initialize the 8255A to exactly "fit" the application. Figures 19 through 25 present a few examples of typical applications of the 8255A.

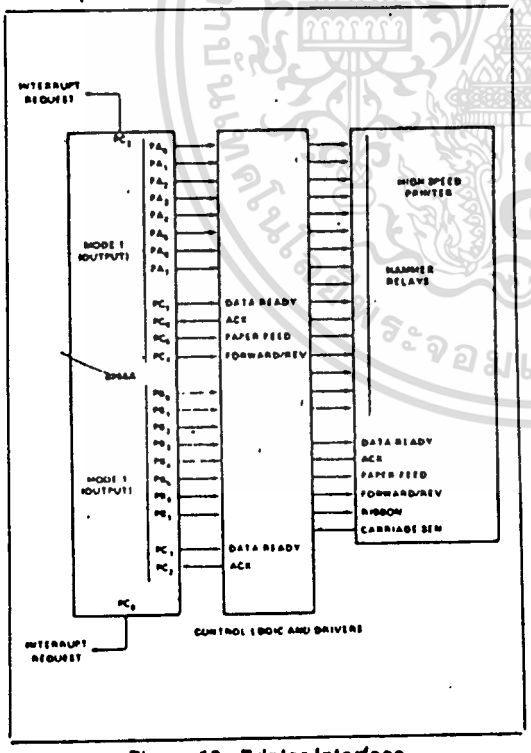


Figure 19. Printer Interface

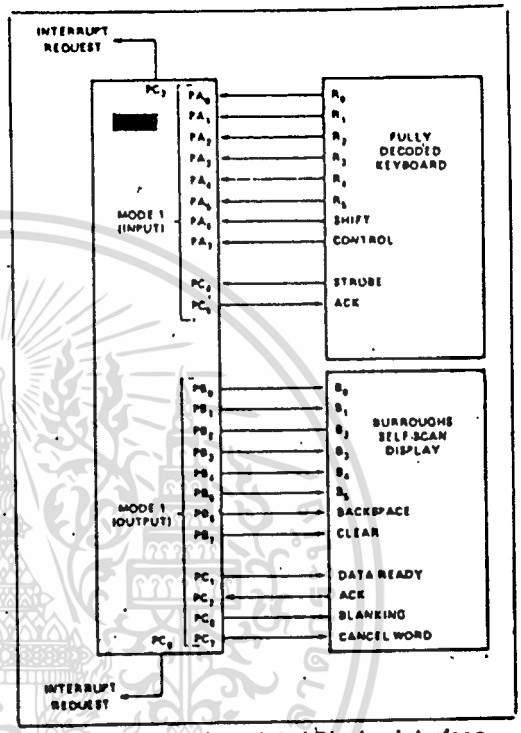


Figure 20. Keyboard and Display Interface

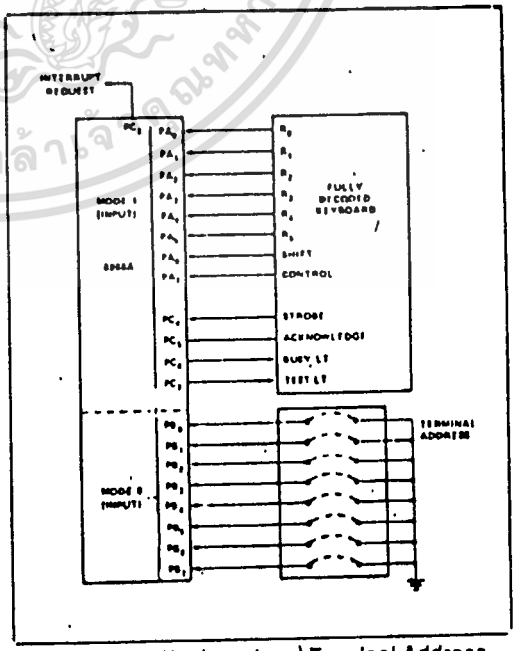


Figure 21. Keyboard and Terminal Address Interface

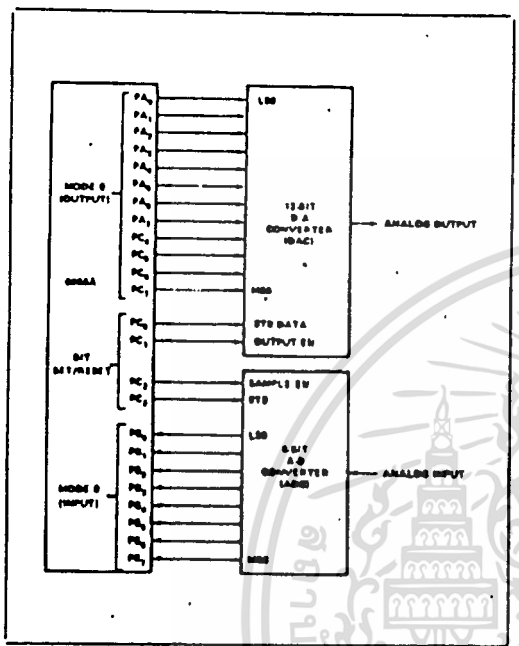


Figure 22. Digital to Analog, Analog to Digital

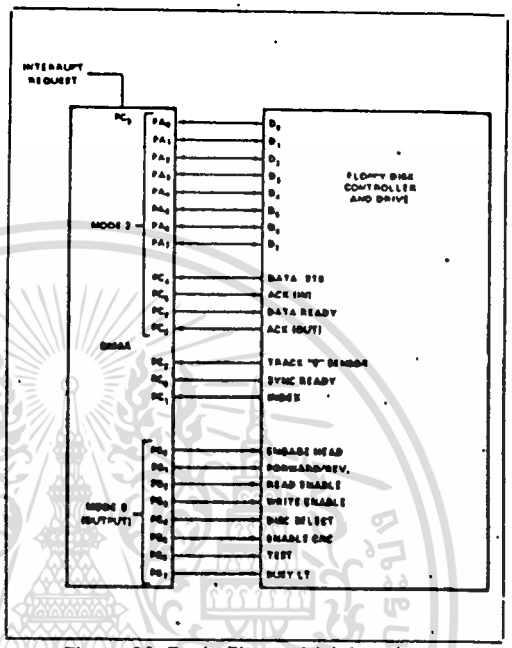


Figure 23. Basic Floppy Disk Interface

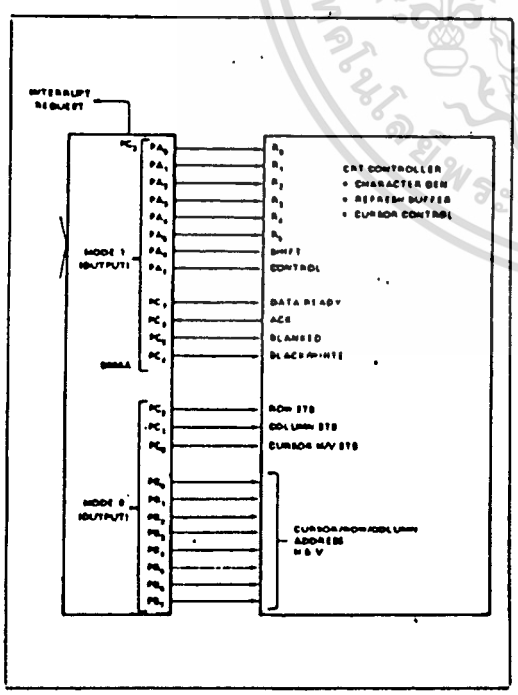


Figure 24. Basic CRT Controller Interface

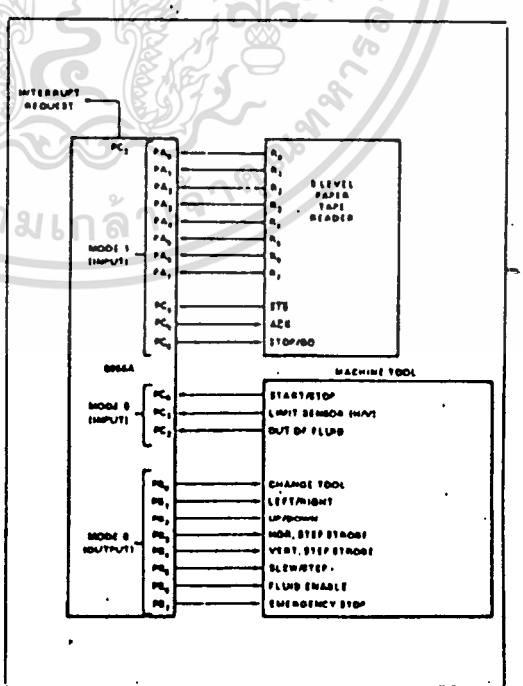


Figure 25. Machine Tool Controller Interface

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . . 0°C to 70°C
Storage Temperature . . . . . −65°C to +150°C
Voltage on Any Pin
With Respect to Ground . . . . . −0.5V to +7V
Power Dissipation . . . . . 1 Watt

NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

D.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5V ± 10%, GND = 0V)*

Symbol	Parameter	Min.	Max.	Unit	Test Conditions
V_{IL}	Input Low Voltage	-0.5	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{OL} (DB)	Output Low Voltage (Data Bus)		0.45*	V	I_{OL} = 2.5mA
V_{OL} (PER)	Output Low Voltage (Peripheral Port)		0.45*	V	I_{OL} = 1.7mA
V_{OH} (DB)	Output High Voltage (Data Bus)	2.4		V	I_{OH} = -400μA
V_{OH} (PER)	Output High Voltage (Peripheral Port)	2.4		V	I_{OH} = -200μA
I_{DAR}[1]	Darlington Drive Current	-1.0	-4.0	mA	R_{EXT} = 750Ω; V_{EXT} = 1.5V
I_{CC}	Power Supply Current		120	mA	
I_{IL}	Input Load Current		±10	μA	V_{IN} = V_{CC} to 0V
I_{OFL}	Output Float Leakage		±10	μA	V_{OUT} = V_{CC} to .45V

NOTES:
1. Available on any 8 pins from Port B and C.

CAPACITANCE (T_A = 25°C, V_{CC} = GND = 0V)

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Conditions
C_{IN}	Input Capacitance			10	pF	fc = 1MHz
$C_{I/O}$	I/O Capacitance			20	pF	Unmeasured pins returned to GND

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = +5V ± 10%, GND = 0V)*

Bus Parameters

READ

Symbol	Parameter	8255A		8255A-5		Unit
		Min.	Max.	Min.	Max.	
t_{AR}	Address Stable Before READ	0		0		ns
t_{RA}	Address Stable After READ	0		0		ns
t_{RR}	READ Pulse Width	300		300		ns
t_{RD}	Data Valid From READ[1]		250		200	ns
t_{DF}	Data Float After READ	10	150	10	100	ns
t_{RV}	Time Between READs and/or WRITEs	850		850		ns

A.C. CHARACTERISTICS (Continued)

WRITE

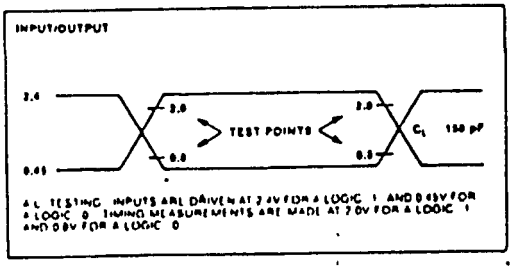
Symbol	Parameter	8255A		8255A-5		Unit
		Min.	Max.	Min.	Max.	
t_{AW}	Address Stable Before WRITE	0		0		ns
t_{WA}	Address Stable After WRITE	20		20		ns
t_{WW}	WRITE Pulse Width	400		300		ns
t_{DW}	Data Valid to WRITE (T.E.)	100		100		ns
t_{WD}	Data Valid After WRITE	30		30		ns

OTHER TIMINGS

Symbol	Parameter	8255A		8255A-5		Unit
		Min.	Max.	Min.	Max.	
t_{WB}	WR = 1 to Output[1]		350		350	ns
t_{IR}	Peripheral Data Before RD	0		0		ns
t_{HR}	Peripheral Data After RD	0		0		ns
t_{AK}	ACK Pulse Width	300		300		ns
t_{ST}	STB Pulse Width	500		500		ns
t_{PS}	Per. Data Before T.E. of STB	0		0		ns
t_{PH}	Per. Data After T.E. of STB	180		180		ns
t_{AD}	ACK = 0 to Output[1]		300		300	ns
t_{KD}	ACK = 1 to Output Float	20	250	20	250	ns
t_{WOB}	WR = 1 to OBF = 0[1]		650		650	ns
t_{AOB}	ACK = 0 to OBF = 1[1]		350		350	ns
t_{SIB}	STB = 0 to IBF = 1[1]		300		300	ns
t_{RIB}	RD = 1 to IBF = 0[1]		300		300	ns
t_{RIT}	RD = 0 to INTR = 0[1]		400		400	ns
t_{SIT}	STB = 1 to INTR = 1[1]		300		300	ns
t_{AIT}	ACK = 1 to INTR = 1[1]		350		350	ns
t_{WIT}	WR = 0 to INTR = 0[1,3]		450		450	ns

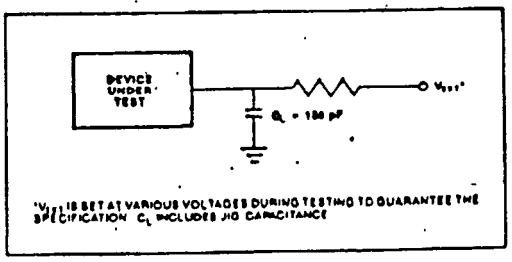
NOTES:
1. Test Conditions: C_L = 150 pF.
2. Period of Reset pulse must be at least 50μs during or after power on. Subsequent Reset pulse can be 500 ns min.
3. INTR↑ may occur as early as $\overline{WR}$↓.
* For Extended Temperature EXPRESS, use M8255A electrical parameters.

A.C. TESTING INPUT, OUTPUT WAVEFORM



A.C. TESTING INPUTS ARE DRIVEN AT 2.4V FOR A LOGIC "1" AND 0.45V FOR A LOGIC "0". TIMING MEASUREMENTS ARE MADE AT 2.0V FOR A LOGIC "1" AND 0.8V FOR A LOGIC "0".

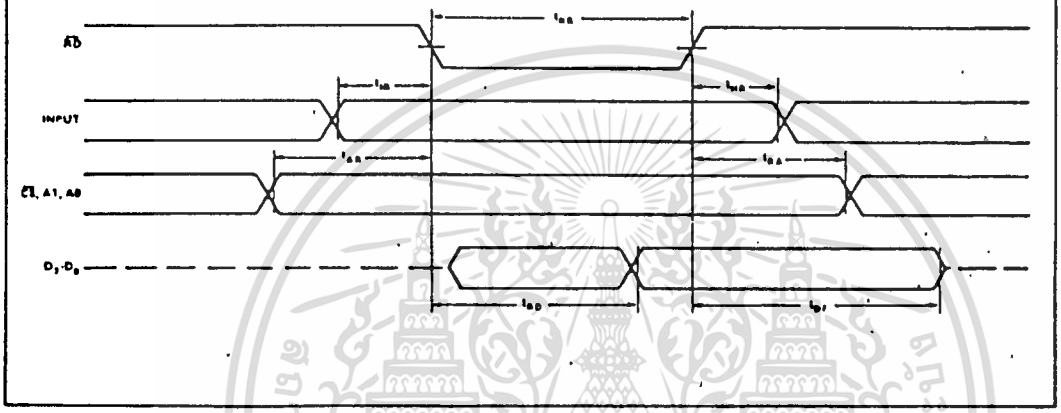
A.C. TESTING LOAD CIRCUIT



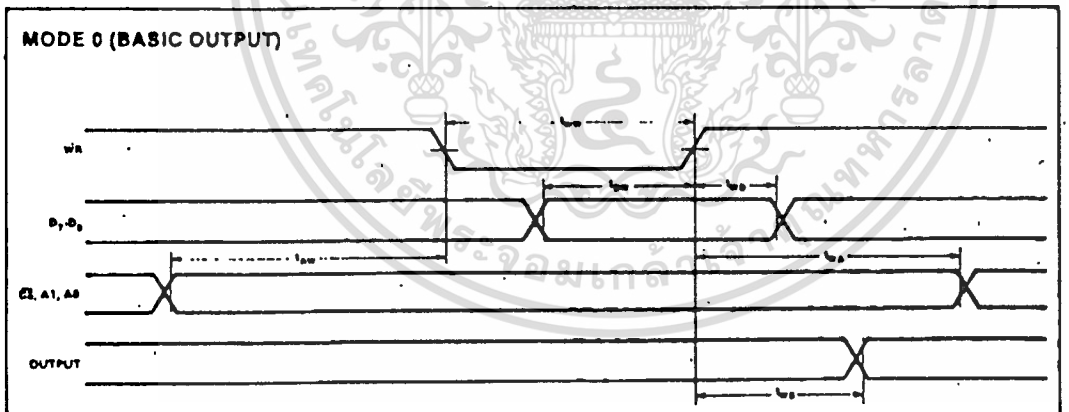
V_{CC}* IS SET AT VARIOUS VOLTAGES DURING TESTING TO GUARANTEE THE SPECIFICATION. C_L INCLUDES JIG CAPACITANCE.

WAVEFORMS

MODE 0 (BASIC INPUT)

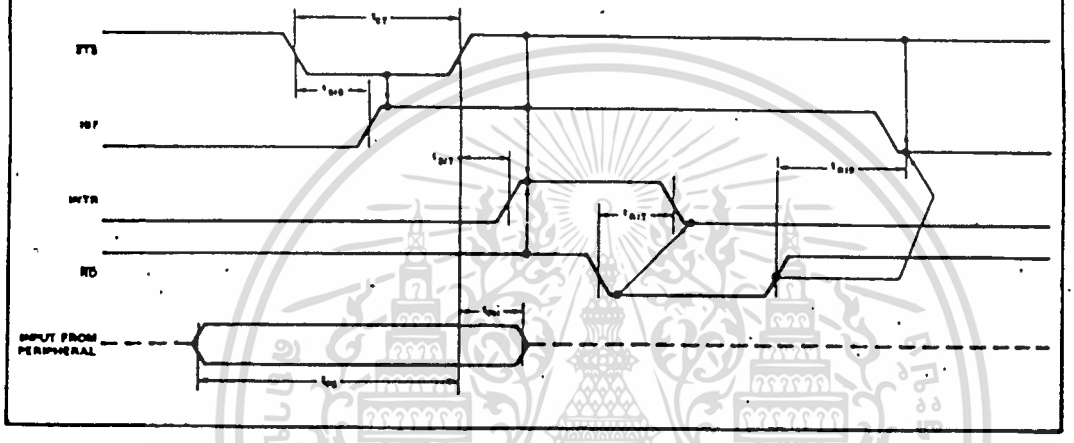


MODE 0 (BASIC OUTPUT)

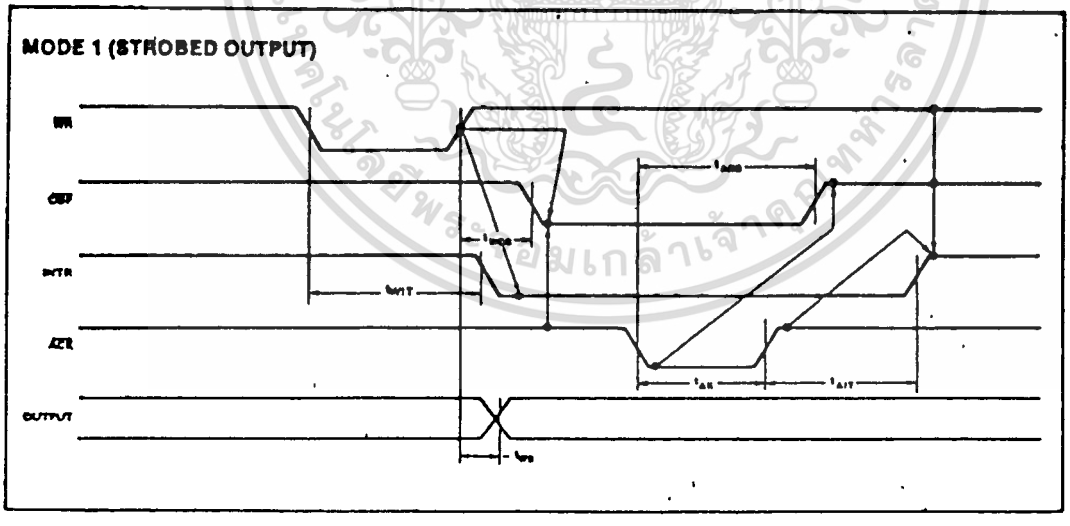


WAVEFORMS (Continued)

MODE 1 (STROBED INPUT)

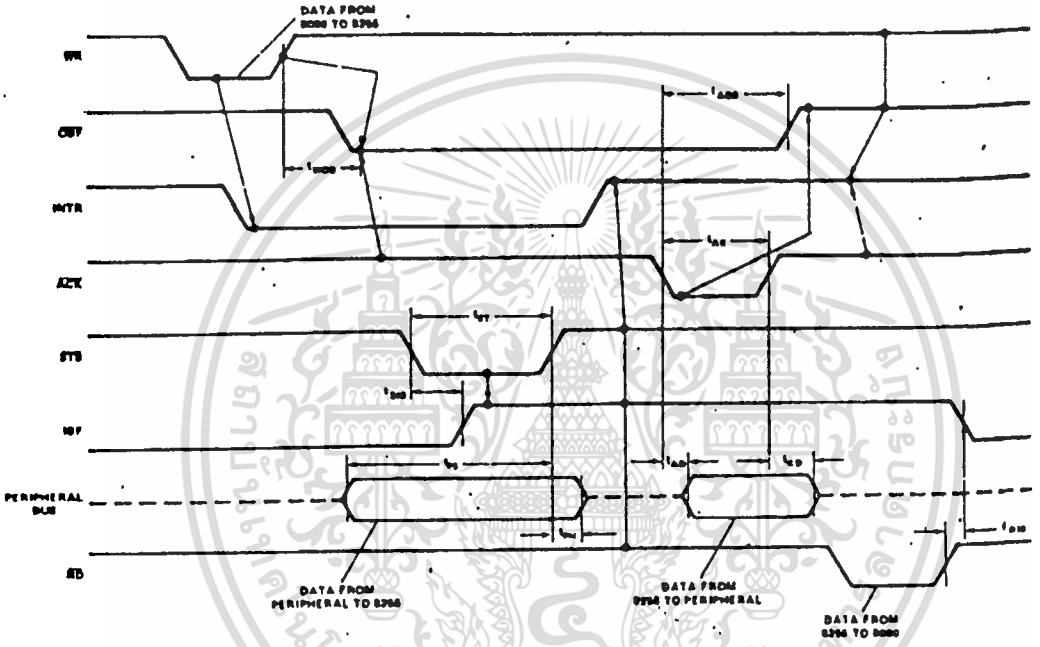


MODE 1 (STROBED OUTPUT)



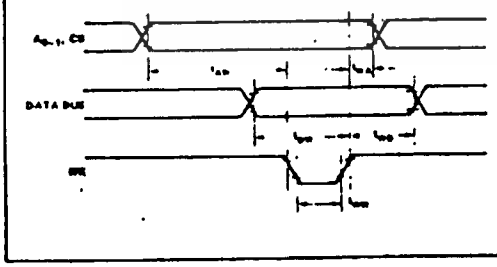
WAVEFORMS (Continued)

MODE 2 (BIDIRECTIONAL)

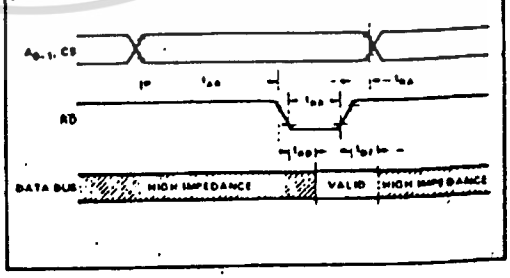


NOTE: Any sequence where $\overline{WR}$ occurs before $\overline{ACK}$ and $\overline{STB}$ occurs before $\overline{RD}$ is permissible.
(INTR = IBF • $\overline{MASK}$ • $\overline{STB}$ • $\overline{RD}$ + $\overline{OBF}$ • $\overline{MASK}$ • $\overline{ACK}$ • $\overline{WR}$)

WRITE TIMING



READ TIMING



Z8400
Z80® CPU Central Processing Unit

ilog

Product Specification

April 1985

Z80 CPU

URES

instruction set contains 158 instructions. The 78 uctions of the 8080A are included as a subset; 0A software compatibility is maintained.

t MHz, 6 MHz, 4 MHz, and 2.5 MHz clocks for the H, Z80B, Z80A, and Z80 CPU result in rapid uction execution with consequent high data ughput.

extensive instruction set includes string, bit, byte, word operations. Block searches and block sfers, together with indexed and relative addressing, lt in the most powerful data handling capabilities in microcomputer industry.

Z80 microprocessors and associated family of pheral controllers are linked by a vectored interrupt system. This system may be daisy-chained to allow implementation of a priority interrupt scheme. Little, if any, additional logic is required for daisy-chaining.

- Duplicate sets of both general-purpose and flag registers are provided, easing the design and operation of system software through single-context switching, background-foreground programming, and single-level interrupt processing. In addition, two 16-bit index registers facilitate program processing of tables and arrays.
- There are three modes of high speed interrupt processing: 8080 similar, non-Z80 peripheral device, and Z80 Family peripheral with or without daisy chain.
- On-chip dynamic memory refresh counter.

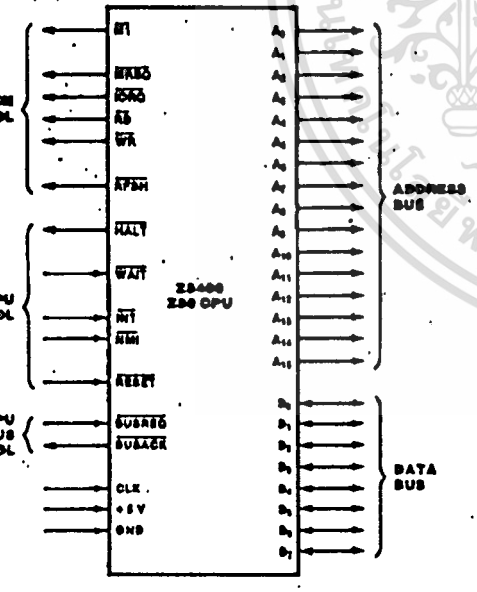


Figure 1. Pin Functions

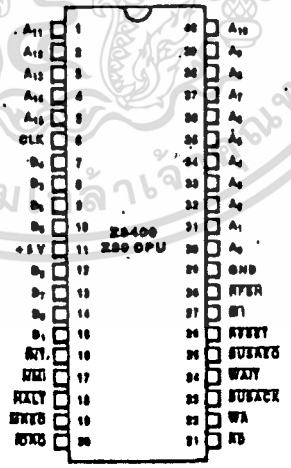


Figure 2a. 40-Pin Dual-In-Line Package (DIP) Pin Assignments

INSTRUCTION SET

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

The following is a summary of the Z80 instruction set which shows the assembly language mnemonic, the operation, the flag status, and gives comments on each instruction. For an explanation of flag notations and symbols for mnemonic tables, see the Symbolic Notations section which follows these tables. The *Z80 CPU Technical Manual* (03-0029-01), the *Programmer's Reference Guide* (03-0012-03), and *Assembly Language Programming Manual* (03-0002-01) contain significantly more details for programming use.

The instructions are divided into the following categories:

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

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

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

8-BIT LOAD GROUP

Mnemonic	Symbolic Operation	Flags S	Z	H	P/V	N	C	Opcode 76	543	210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD r, r'	r ← r'	•	•	X	•	X	•	•	•	01	r	r'		1	1	4	r, r' Reg.
LD r, n	r ← n	•	•	X	•	X	•	•	•	00	r	110		2	2	7	000 B
											← n →						001 C
LD r, (HL)	r ← (HL)	•	•	X	•	X	•	•	•	01	r	110		1	2	7	010 D
LD r, (IX + d)	r ← (IX + d)	•	•	X	•	X	•	•	•	11	011	101	DD	3	5	19	011 E
										01	r	110					100 H
											← d →						101 L
LD r, (IY + d)	r ← (IY + d)	•	•	X	•	X	•	•	•	11	111	101	FD	3	5	19	111 A
										01	r	110					
											← d →						
LD (HL), r	(HL) ← r	•	•	X	•	X	•	•	•	01	110	r		1	2	7	
LD (IX + d), r	(IX + d) ← r	•	•	X	•	X	•	•	•	11	011	101	DD	3	5	19	
										01	110	r					
											← d →						
LD (IY + d), r	(IY + d) ← r	•	•	X	•	X	•	•	•	11	111	101	FD	3	5	19	
										01	110	r					
											← d →						
LD (HL), n	(HL) ← n	•	•	X	•	X	•	•	•	00	110	110	36	2	3	10	
											← n →						
LD (IX + d), n	(IX + d) ← n	•	•	X	•	X	•	•	•	11	011	101	DD	4	5	19	
										00	110	110	36				
											← d →						
											← n →						

8-BIT LOAD GROUP (Continued)

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76	543	210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD (IY + d), n	(IY + d) ← n	•	•	X	•	X	•	•	•	11	111	101	FD	4	5	19	
								00	110	110	36				
									← d →						
									← n →						
LD A, (BC)	A ← (BC)	•	•	X	•	X	•	•	•	00	001	010	0A	1	2	7	
LD A, (DE)	A ← (DE)	•	•	X	•	X	•	•	•	00	011	010	1A	1	2	7	
LD A, (nn)	A ← (nn)	•	•	X	•	X	•	•	•	00	111	010	3A	3	4	13	
											← n →						
											← n →						
LD (BC), A	(BC) ← A	•	•	X	•	X	•	•	•	00	000	010	02	1	2	7	
LD (DE), A	(DE) ← A	•	•	X	•	X	•	•	•	00	010	010	12	1	2	7	
LD (nn), A	(nn) ← A	•	•	X	•	X	•	•	•	00	110	010	32	3	4	13	
											← n →						
											← n →						
LD A, I	A ← I	‡	‡	X	0	X	IFF	0	•	11	101	101	ED	2	2	9	
										01	010	111	57				
LD A, R	A ← R	‡	‡	X	0	X	IFF	0	•	11	101	101	ED	2	2	9	
										01	011	111	5F				
LD I, A	I ← A	•	•	X	•	X	•	•	•	11	101	101	ED	2	2	9	
										01	000	111	47				
LD R, A	R ← A	•	•	X	•	X	•	•	•	11	101	101	ED	2	2	9	
										01	001	111	4F				

NOTE: IFF, the content of the interrupt enable flip-flop, (IFF_2), is copied into the P/V flag.

16-BIT LOAD GROUP

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76	543	210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments	
LD dd, nn	dd ← nn	•	•	X	•	X	•	•	•	00	dd0	001		3	3	10	dd	Pair
											← n →						00	BC
											← n →						01	DE
LD IX, nn	IX ← nn	•	•	X	•	X	•	•	•	11	011	101	DD	4	4	14	10	HL
										00	100	001	21				11	SP
											← n →							
											← n →							
LD IY, nn	IY ← nn	•	•	X	•	X	•	•	•	11	111	101	FD	4	4	14		
										00	100	001	21					
											← n →							
											← n →							
LD HL, (nn)	H ← (nn + 1)	•	•	X	•	X	•	•	•	00	101	010	2A	3	5	16		
	L ← (nn)										← n →							
											← n →							
LD dd, (nn)	dd_H ← (nn + 1)	•	•	X	•	X	•	•	•	11	101	101	ED	4	6	20		
	dd_L ← (nn)									01	dd1	011						
											← n →							
											← n →							

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

16-BIT LOAD GROUP (Continued)

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	76	543	210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD IX, (nn)	$IX_H \leftarrow (nn+1)$	•	•	X	X	•	•	11	011	101	DD	4	6	20	
	$IX_L \leftarrow (nn)$							00	101	010	2A				
								← n →							
								← n →							
LD IY, (nn)	$IY_H \leftarrow (nn+1)$	•	•	X	X	•	•	11	111	101	FD	4	6	20	
	$IY_L \leftarrow (nn)$							00	101	010	2A				
								← n →							
								← n →							
LD (nn), HL	$(nn+1) \leftarrow H$	•	•	X	X	•	•	00	100	010	22	3	5	16	
	$(nn) \leftarrow L$								← n →						
									← n →						
LD (nn), dd	$(nn+1) \leftarrow dd_H$	•	•	X	X	•	•	11	101	101	ED	4	6	20	
	$(nn) \leftarrow dd_L$							01	dd0	011					
									← n →						
									← n →						
LD (nn), IX	$(nn+1) \leftarrow IX_H$	•	•	X	X	•	•	11	011	101	DD	4	6	20	
	$(nn) \leftarrow IX_L$							00	100	010	22				
									← n →						
									← n →						
LD (nn), IY	$(nn+1) \leftarrow IY_H$	•	•	X	X	•	•	11	111	101	FD	4	6	20	
	$(nn) \leftarrow IY_L$							00	100	010	22				
									← n →						
									← n →						
LD SP, HL	$SP \leftarrow HL$	•	•	X	X	•	•	11	111	001	F9	1	1	6	
LD SP, IX	$SP \leftarrow IX$	•	•	X	X	•	•	11	011	101	DD	2	2	10	
								11	111	001	F9				
LD SP, IY	$SP \leftarrow IY$	•	•	X	X	•	•	11	111	101	FD	2	2	10	
								11	111	001	F9				
PUSH qq	$(SP-2) \leftarrow qq_L$	•	•	X	X	•	•	11	qq0	101		1	3	11	
	$(SP-1) \leftarrow qq_H$														
	$SP \leftarrow SP-2$														
PUSH IX	$(SP-2) \leftarrow IX_L$	•	•	X	X	•	•	11	011	101	DD	2	4	15	
	$(SP-1) \leftarrow IX_H$							11	100	101	E5				
	$SP \leftarrow SP-2$														
PUSH IY	$(SP-2) \leftarrow IY_L$	•	•	X	X	•	•	11	111	101	FD	2	4	15	
	$(SP-1) \leftarrow IY_H$							11	100	101	E5				
	$SP \leftarrow SP-2$														
POP qq	$qq_H \leftarrow (SP+1)$	•	•	X	X	•	•	11	qq0	001		1	3	10	
	$qqL \leftarrow (SP)$														
	$SP \leftarrow SP+2$														
POP IX	$IX_H \leftarrow (SP+1)$	•	•	X	X	•	•	11	011	101	DD	2	4	14	
	$IX_L \leftarrow (SP)$							11	100	001	E1				
	$SP \leftarrow SP+2$														
POP IY	$IY_H \leftarrow (SP+1)$	•	•	X	X	•	•	11	111	101	FD	2	4	14	
	$IY_L \leftarrow (SP)$							11	100	001	E1				
	$SP \leftarrow SP+2$														

qq	Pair
00	BC
01	DE
10	HL
11	AF

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

EXCHANGE, BLOCK TRANSFER, BLOCK SEARCH GROUPS

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	76	543	210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
EX DE, HL	DE ↔ HL	•	•	X	•	X	•	•	•	11	101	011	EB	1	1	4	
EX AF, AF'	AF ↔ AF'	•	•	X	•	X	•	•	•	00	001	000	08	1	1	4	
EXX	BC ↔ BC' DE ↔ DE' HL ↔ HL'	•	•	X	•	X	•	•	•	11	011	001	D9	1	1	4	Register bank and auxiliary register bank exchange
EX (SP), HL	H ↔ (SP + 1) L ↔ (SP)	•	•	X	•	X	•	•	•	11	100	011	E3	1	5	19	
EX (SP), IX	IX_H ↔ (SP + 1) IX_L ↔ (SP)	•	•	X	•	X	•	•	•	11 11	011 100	101 011	DD E3	2	6	23	
EX (SP), IY	IY_H ↔ (SP + 1) IYL ↔ (SP)	•	•	X	•	X	•	•	•	11 11	111 100	101 011	FD E3	2	6	23	
LDI	(DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC − 1	•	•	X	0	X	①	0	•	11 10	101 100	101 000	ED A0	2	4	16	Load (HL) into (DE), increment the pointers and decrement the byte counter (BC)
LDIR	(DE) ← (HL) DE ← DE + 1 HL ← HL + 1 BC ← BC − 1 Repeat until BC = 0	•	•	X	0	X	②	0	•	11 10	101 110	101 000	ED B0	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
LDD	(DE) ← (HL) DE ← DE − 1 HL ← HL − 1 BC ← BC − 1	•	•	X	0	X	①	0	•	11 10	101 101	101 000	ED A8	2	4	16	
LDDR	(DE) ← (HL) DE ← DE − 1 HL ← HL − 1 BC ← BC − 1 Repeat until BC = 0	•	•	X	0	X	②	0	•	11 10	101 111	101 000	ED B8	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
CPI	A − (HL) HL ← HL + 1 BC ← BC − 1	③	③	X	①	X	1	•		11 10	101 100	101 001	ED A1	2	4	16	

NOTE: ① P/V flag is 0 if the result of BC − 1 = 0, otherwise P/V = 1.
② P/V flag is 0 only at completion of instruction.
③ Z flag is 1 if A = HL, otherwise Z = 0.

EXCHANGE, BLOCK TRANSFER, BLOCK SEARCH GROUPS (Continued)

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	76	543	210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
CPIR	A - (HL)	↕	③ ↕	X	① ↕	1	•	11	101	101	ED	2	5	21	If BC ≠ 0 and A ≠ (HL)
	HL ← HL + 1 BC ← BC - 1 Repeat until A = (HL) or BC = 0			X	X			10	110	001	B1	2	4	16	If BC = 0 or A = (HL)
CPD	A - (HL)	↕	③ ↕	X	① ↕	1	•	11	101	101	ED	2	4	16	
	HL ← HL - 1 BC ← BC - 1			X	X			10	101	001	A9				
CPDR	A - (HL)	↕	③ ↕	X	① ↕	1	•	11	101	101	ED	2	5	21	If BC ≠ 0 and A ≠ (HL)
	HL ← HL - 1 BC ← BC - 1 Repeat until A = (HL) or BC = 0			X	X			10	111	001	B9	2	4	16	If BC = 0 or A = (HL)

NOTE: ① P/V flag is 0 if the result of BC - 1 = 0, otherwise P/V = 1.
② P/V flag is 0 only at completion of instruction.
③ Z flag is 1 if A = (HL), otherwise Z = 0.

8-BIT ARITHMETIC AND LOGICAL GROUP

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	76	543	210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
ADD A, r	A ← A + r	↕	↕	X	↕	0	↕	10	000	r		1	1	4	r Reg.
ADD A, n	A ← A + n	↕	↕	X	V	0	↕	11	000	110		2	2	7	000 B
									← n →						001 C
															010 D
ADD A, (HL)	A ← A + (HL)	↕	↕	X	V	0	↕	10	000	110		1	2	7	011 E
ADD A, (IX + d)	A ← A + (IX + d)	↕	↕	X	V	0	↕	11	011	101	DD	3	5	19	100 H
								10	000	110					101 L
									← d →						111 A
ADD A, (IY + d)	A ← A + (IY + d)	↕	↕	X	V	0	↕	11	111	101	FD	3	5	19	
								10	000	110					
									← d →						
ADC A, s	A ← A + s + CY	↕	↕	X	V	0	↕		001						s is any of r, n, (HL), (IX + d), (IY + d) as shown for ADD instruction. The indicated bits replace the 000 in the ADD set above.
SUB s	A ← A - s	↕	↕	X	V	1	↕		010						
SBC A, s	A ← A - s - CY	↕	↕	X	V	1	↕		011						
AND s	A ← A ∧ s	↕	↕	X	1	X	P	0	0		100				
OR s	A ← A ∨ s	↕	↕	X	0	X	P	0	0		110				
XOR s	A ← A ⊕ s	↕	↕	X	0	X	P	0	0		101				
CP s	A - s	↕	↕	X	↕	X	V	1	↕		111				

8-BIT ARITHMETIC AND LOGICAL GROUP (Continued)

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76	543	210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
INC r	r ← r + 1	↕	↕	X	↕	V	0	•	00	r	100		1	1	4	
INC (HL)	(HL) ← (HL) + 1	↕	↕	X	↕	V	0	•	00	110	100		1	3	11	
INC (IX + d)	(IX + d) ← (IX + d) + 1	↕	↕	X	↕	V	0	•	11 00	011 110	101 100	DD	3	6	23	
									← d →							
INC (IY + d)	(IY + d) ← (IY + d) + 1	↕	↕	X	↕	V	0	•	11 00	111 110	101 100	FD	3	6	23	
									← d →							
DEC m	m ← m − 1	↕	↕	X	↕	V	1	•			101					

NOTE: m is any of r, (HL), (IX + d), (IY + d) as shown for INC. DEC same format and states as INC. Replace 100 with 101 in opcode

GENERAL-PURPOSE ARITHMETIC AND CPU CONTROL GROUPS

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

NOTES: ● converts accumulator content into packed BCD following add or subtract with packed BCD operands:
IFF indicates the interrupt enable flip-flop.
CY indicates the carry flip-flop.
★ indicates interrupts are not sampled at the end of EI or DI.

16-BIT ARITHMETIC GROUP

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	76	543	210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
ADD HL, ss	HL ← HL + ss	•	•	X	X	X	•	0	‡	00	ss1	001		1	3	11	ss Reg.
																	00 BC
ADC HL, ss	HL ←																01 DE
	HL + ss + CY	‡	‡	X	X	X	V	0	‡	11	101	101	ED	2	4	15	10 HL
										01	ss1	010					11 SP
SBC HL, ss	HL ←																
	HL − ss − CY	‡	‡	X	X	X	V	1	‡	11	101	101	ED	2	4	15	
										01	ss0	010					
ADD IX, pp	IX ← IX + pp	•	•	X	X	X	•	0	‡	11	011	101	DD	2	4	15	pp Reg.
										01	pp1	001					00 BC
																	01 DE
																	10 IX
																	11 SP
ADD IY, rr	IY ← IY + rr	•	•	X	X	X	•	0	‡	11	111	101	FD	2	4	15	rr Reg.
										00	rr1	001					00 BC
INC ss	ss ← ss + 1	•	•	X	•	X	•	•	•	00	ss0	011		1	1	6	01 DE
INC IX	IX ← IX + 1	•	•	X	•	X	•	•	•	11	011	101	DD	2	2	10	10 IY
										00	100	011	23				11 SP
INC IY	IY ← IY + 1	•	•	X	•	X	•	•	•	11	111	101	FD	2	2	10	
										00	100	011	23				
DEC ss	ss ← ss − 1	•	•	X	•	X	•	•	•	00	ss1	011		1	1	6	
DEC IX	IX ← IX − 1	•	•	X	•	X	•	•	•	11	011	101	DD	2	2	10	
										00	101	011	2B				
DEC IY	IY ← IY − 1	•	•	X	•	X	•	•	•	11	111	101	FD	2	2	10	
										00	101	011	2B				

ROTATE AND SHIFT GROUP

Mnemonic	Symbolic Operation	S	Z		H		P/V	N	C	76	543	210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
RLCA		•	•	X	0	X	•	0	‡	00	000	111	07	1	1	4	Rotate left circular accumulator.
RLA		•	•	X	0	X	•	0	‡	00	010	111	17	1	1	4	Rotate left accumulator.
RRCA		•	•	X	0	X	•	0	‡	00	001	111	0F	1	1	4	Rotate right circular accumulator.
RRA		•	•	X	0	X	•	0	‡	00	011	111	1F	1	1	4	Rotate right accumulator.

ROTATE AND SHIFT GROUP (Continued)

Symbolic Mnemonic	Operation	S	Z	H	P/V	N	C	76	543	210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
RLC r		↕	↕	X	0	X	P	0	↕	11	001	011	CB	2	2	8	Rotate left circular register r.
										00	[000]	r					
RLC (HL)		↕	↕	X	0	X	P	0	↕	11	001	011	CB	2	4	15	r: 000 B, 001 C, 010 D, 011 E, 001 H, 101 L, 111 A
										00	000	110					
RLC (IX + d)		↕	↕	X	0	X	P	0	↕	11	011	101	DD	4	6	23	
	r,(HL),(IX + d),(IY + d)									11	001	011	CB				
											← d →						
										00	[000]	110					
RLC (IY + d)		↕	↕	X	0	X	P	0	↕	11	111	101	FD	4	6	23	
										11	001	011	CB				
											← d →						
										00	[000]	110					
RL m		↕	↕	X	0	X	P	0	↕		[010]						Instruction format and states are as shown for RLCs. To form new opcode replace [000] of RLC with shown code.
	m = r,(HL),(IX + d),(IY + d)																
RRC m		↕	↕	X	0	X	P	0	↕		[001]						
	m = r,(HL),(IX + d),(IY + d)																
RR m		↕	↕	X	0	X	P	0	↕		[011]						
	m = r,(HL),(IX + d),(IY + d)																
SLA m		↕	↕	X	0	X	P	0	↕		[100]						
	m = r,(HL),(IX + d),(IY + d)																
SRA m		↕	↕	X	0	X	P	0	↕		[101]						
	m = r,(HL),(IX + d),(IY + d)																
SRL m		↕	↕	X	0	X	P	0	↕		[111]						
	m = r,(HL),(IX + d),(IY + d)																
RLD		↕	↕	X	0	X	P	0	•	11	101	101	ED	2	5	18	Rotate digit left and right between the accumulator and location (HL). The content of the upper half of the accumulator is unaffected.
										01	101	111	6F				
RRD		↕	↕	X	0	X	P	0	•	11	101	101	ED	2	5	18	
										01	100	111	67				

BIT SET, RESET AND TEST GROUP

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

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

JUMP GROUP

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	76	543	210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
JP nn	PC ← nn	•	•	X	•	X	•	•	•	11	000	011	C3	3	3	10	cc Condition
										← n →							000 NZ (non-zero)
										← n →							001 Z (zero)
JP cc, nn	If condition cc	•	•	X	•	X	•	•	•	11	cc	010		3	3	10	010 NC (non-carry)
	is true PC ← nn,									← n →							011 C (carry)
	otherwise									← n →							100 PO (parity odd)
	continue																101 PE (parity even)
JR e	PC ← PC + e	•	•	X	•	X	•	•	•	00	011	000	18	2	3	12	110 P (sign positive)
										← e − 2 →							111 M (sign negative)
JR C, e	If C = 0,	•	•	X	•	X	•	•	•	00	111	000	38	2	2	7	If condition not met.
	continue									← e − 2 →							
	If C = 1,													2	3	12	If condition is met.
	PC ← PC + e																
JR NC, e	IF C = 1,	•	•	X	•	X	•	•	•	00	110	000	30	2	2	7	If condition not met.
	continue									← e − 2 →							
	If C = 0,													2	3	12	If condition is met.
	PC ← PC + e																
JP Z, e	If Z = 0	•	•	X	•	X	•	•	•	00	101	000	28	2	2	7	If condition not met.
	continue									← e − 2 →							
	If Z = 1,													2	3	12	If condition is met.
	PC ← PC + e																
JR NZ, e	If Z = 1,	•	•	X	•	X	•	•	•	00	100	000	20	2	2	7	If condition not met.
	continue									← e − 2 →							
	If Z = 0,													2	3	12	If condition is met.
	PC ← PC + e																
JP (HL)	PC ← HL	•	•	X	•	X	•	•	•	11	101	001	E9	1	1	4	
JP (IX)	PC ← IX	•	•	X	•	X	•	•	•	11	011	101	DD	2	2	8	
										11	101	001	E9				
JP (IY)	PC ← IY	•	•	X	•	X	•	•	•	11	111	101	FD	2	2	8	
										11	101	001	E9				
DJNZ, e	B ← B − 1	•	•	X	•	X	•	•	•	00	010	000	10	2	2	8	If B = 0
	If B = 0,									← e − 2 →							
	continue																
	If B≠0,													2	3	13	If B≠0.
	PC ← PC + e																

NOTES: e represents the extension in the relative addressing mode.
e is a signal two's complement number in the range < − 126, 129 >.
e − 2 in the opcode provides an effective address of pc + e as PC is incremented by 2 prior to the addition of e. —

CALL AND RETURN GROUP

Mnemonic	Symbolic Operation	S	Z	H	P/V	N	C	Opcode 76	543	210	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
CALL nn	$(SP-1)\leftarrow PC_H$ $(SP-2)\leftarrow PC_L$ $PC\leftarrow nn$,	•	•	X	•	X	•	•	•	11	001 ← n → ← n →	101	CD	3	5	17	
CALL cc,nn	If condition cc is false	•	•	X	•	X	•	•	•	11	cc ← n →	100		3	3	10	If cc is false.
	continue, otherwise same as CALL nn										← n →			3	5	17	If cc is true.
RET	$PC_L\leftarrow(SP)$ $PC_H\leftarrow(SP+1)$	•	•	X	•	X	•	•	•	11	001	001	C9	1	3	10	
RET cc	If condition cc is false	•	•	X	•	X	•	•	•	11	cc	000		1	1	5	If cc is false.
	continue, otherwise same as RET													1	3	11	If cc is true.
RETI	Return from interrupt	•	•	X	•	X	•	•	•	11 01	101 001	101 101	ED 4D	2	4	14	
RETN[1]	Return from non-maskable interrupt	•	•	X	•	X	•	•	•	11 01	101 000	101 101	ED 45	2	4	14	
RST p	$(SP-1)\leftarrow PC_H$ $(SP-2)\leftarrow PC_L$ $PC_H\leftarrow 0$ $PC_L\leftarrow p$	•	•	X	•	X	•	•	•	11	t	111		1	3	11	

cc	Condition
000	NZ (non-zero)
001	Z (zero)
010	NC (non-carry)
011	C (carry)
100	PO (parity odd)
101	PE (parity even)
110	P (sign positive)
111	M (sign negative)

t	p
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

NOTE: [1]RETN loads $IFF_2 \rightarrow IFF_1$

INPUT AND OUTPUT GROUP

Mnemonic	Symbolic Operation	Flags S	Z	H	P/V	N	C	Opcode 7&6	5&4&3	2&1&0	Hex	No. of Bytes	No. of M Cycles	No. of T States	Comments
IN A, (n)	A ← (n)	•	•	X	•	X	•	•	•	11	011	01	DB	2	3	11	n to $A_0 \sim A_7$ Acc. to $A_8 \sim A_{15}$
											← n →						
IN r, (C)	r ← (C) if r = 110 only the flags will be affected	‡	‡	X	‡	X	P	0	•	11 01	101 r	101 000	ED	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INI	(HL) ← (C) B ← B – 1 HL ← HL + 1	X	‡ ①	X	X	X	X	1	X	11 10	101 100	101 010	ED A2	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INIR	(HL) ← (C) B ← B – 1 HL ← HL + 1 Repeat until B = 0	X	1 ②	X	X	X	X	1	X	11 10	101 110	101 010	ED B2	2 2	5 (If B≠0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
IND	(HL) ← (C) B ← B – 1 HL ← HL – 1	X	‡ ①	X	X	X	X	1	X	11 10	101 101	101 010	ED AA	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INDR	(HL) ← (C) B ← B – 1 HL ← HL – 1 Repeat until B = 0	X	1 ②	X	X	X	X	1	X	11 10	101 111	101 010	ED BA	2 2	5 (If B≠0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUT (n), A	(n) ← A	•	•	X	•	X	•	•	•	11	010	011	D3	2	3	11	n to $A_0 \sim A_7$ Acc. to $A_8 \sim A_{15}$
											← n →						
OUT (C), r	(C) ← r	•	•	X	•	X	•	•	•	11 01	101 r	101 001	ED	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTI	(C) ← (HL) B ← B – 1 HL ← HL + 1	X	‡ ①	X	X	X	X	1	X	11 10	101 100	101 011	ED A3	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTIR	(C) ← (HL) B ← B – 1 HL ← HL + 1 Repeat until B = 0	X	1 ②	X	X	X	X	1	X	11 10	101 110	101 011	ED B3	2 2	5 (If B≠0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTD	(C) ← (HL) B ← B – 1 HL ← HL – 1	X	‡ ①	X	X	X	X	1	X	11 10	101 101	101 011	ED AB	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTDR	(C) ← (HL) B ← B – 1 HL ← HL – 1 Repeat until B = 0	X	1 ②	X	X	X	X	1	X	11 10	101 111	101 011	ED	2 2	5 (If B≠0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$

NOTES: ① If the result of B – 1 is zero, the Z flag is set; otherwise it is reset.
② Z flag is set upon instruction completion only.

CPU TIMING

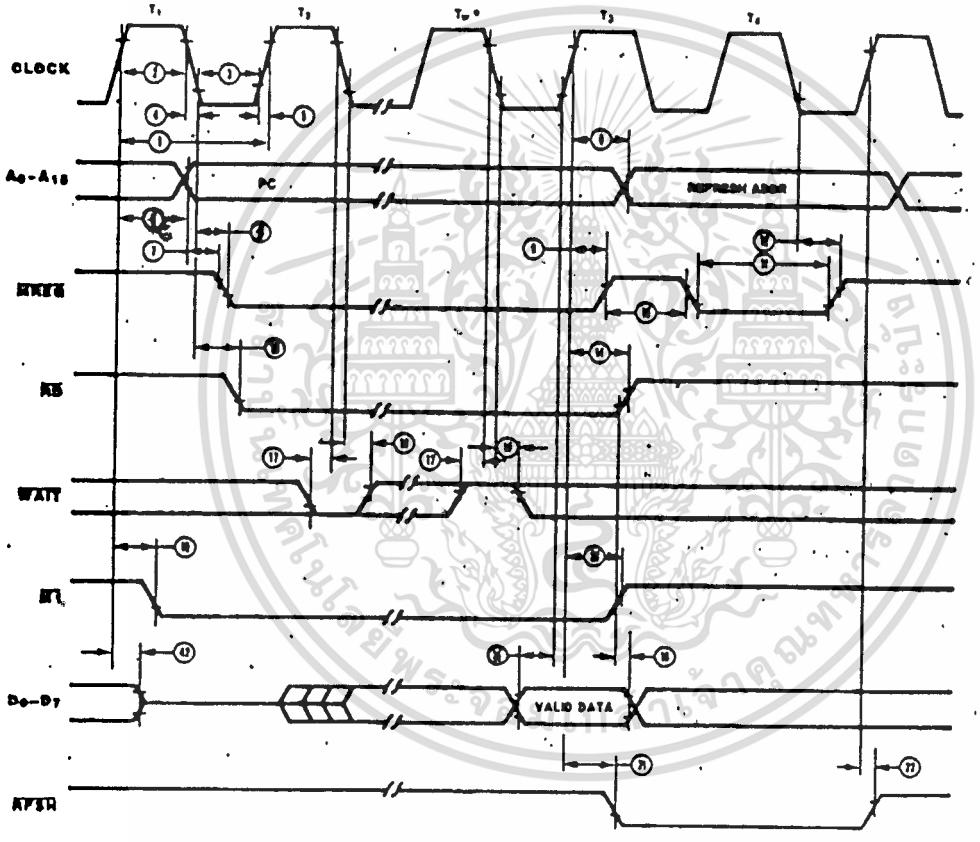
The Z80 CPU executes instructions by proceeding through a specific sequence of operations:

- Memory read or write
- I/O device read or write
- Interrupt acknowledge

The basic clock period is referred to as a T time or cycle, and three or more T cycles make up a machine cycle (M1, M2 or M3 for instance). Machine cycles can be extended either by the CPU automatically inserting one or more Wait states or by the insertion of one or more Wait states by the user.

Instruction Opcode Fetch. The CPU places the contents of the Program Counter (PC) on the address bus at the start of the cycle (Figure 5). Approximately one-half clock cycle later, $\overline{MREQ}$ goes active. When active, $\overline{RD}$ indicates that the memory data can be enabled onto the CPU data bus.

The CPU samples the $\overline{WAIT}$ input with the falling edge of clock state T_2. During clock states T_3 and T_4 of an $\overline{M1}$ cycle, dynamic RAM refresh can occur while the CPU starts decoding and executing the instruction. When the Refresh Control signal becomes active, refreshing of dynamic memory can take place.



*T_W = Wait cycle added when necessary for slow ancillary devices.

Figure 5. Instruction Opcode Fetch

Memory Read or Write Cycles. Figure 6 shows the timing of memory read or write cycles other than an opcode fetch ($\overline{M1}$) cycle. The $\overline{MREQ}$ and $\overline{RD}$ signals function exactly as in the fetch cycle. In a memory write cycle, $\overline{MREQ}$ also becomes active when the address bus is stable. The $\overline{WR}$ line is active when the data bus is stable, so that it can be used directly as an R/$\overline{W}$ pulse to most semiconductor memories.

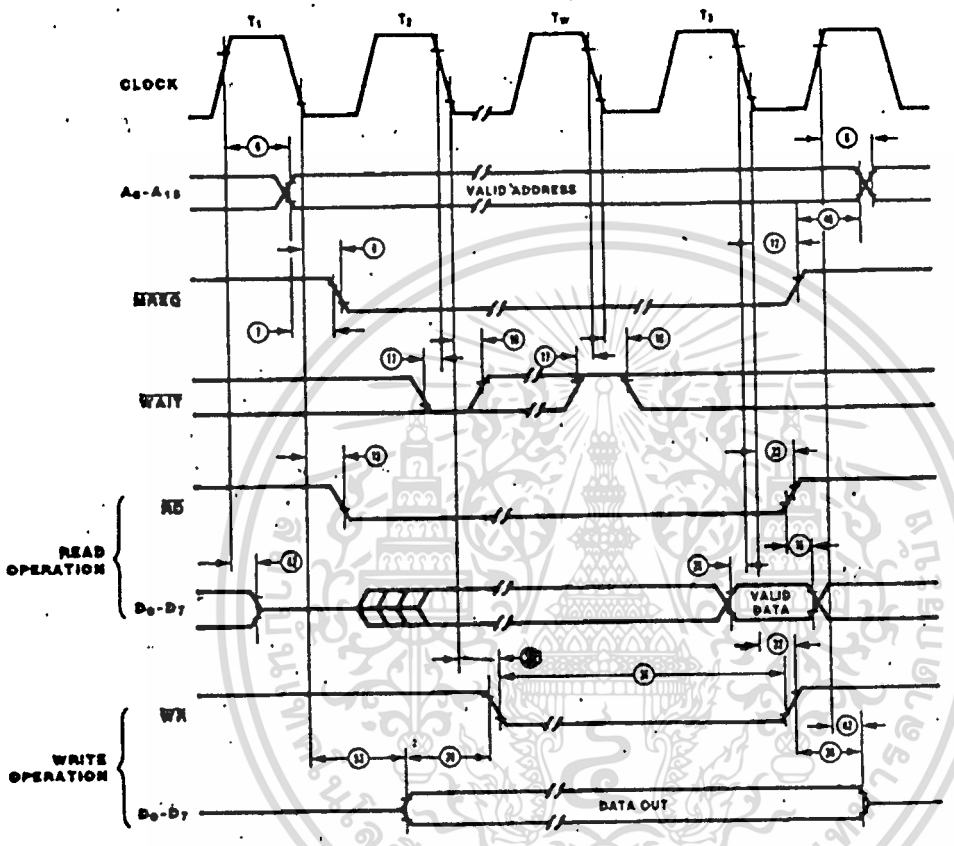
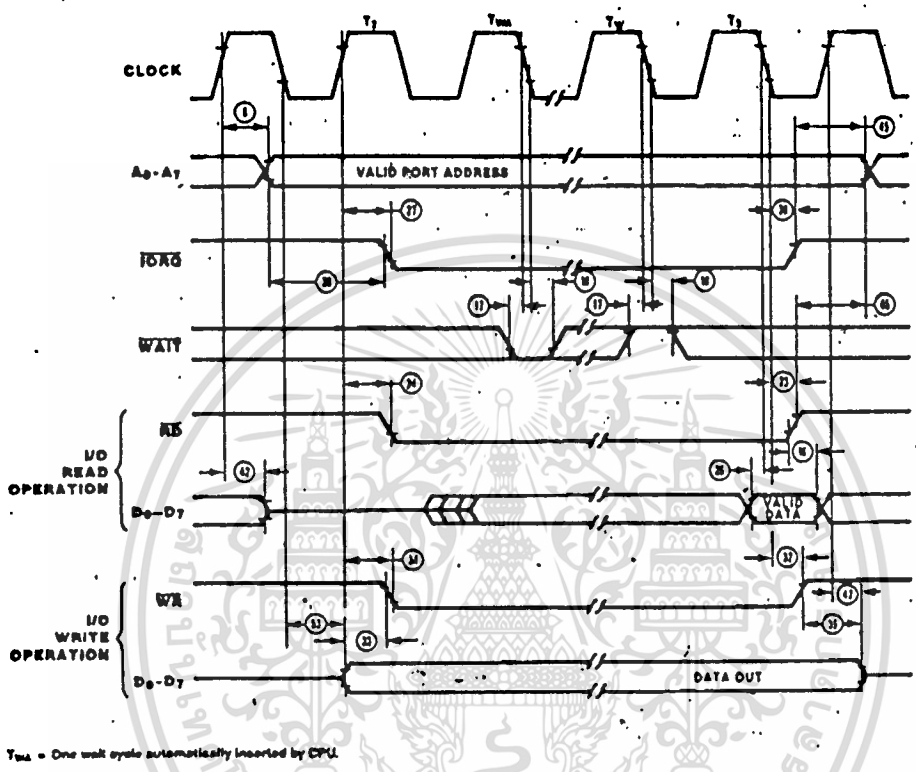


Figure 8. Memory Read or Write Cycles

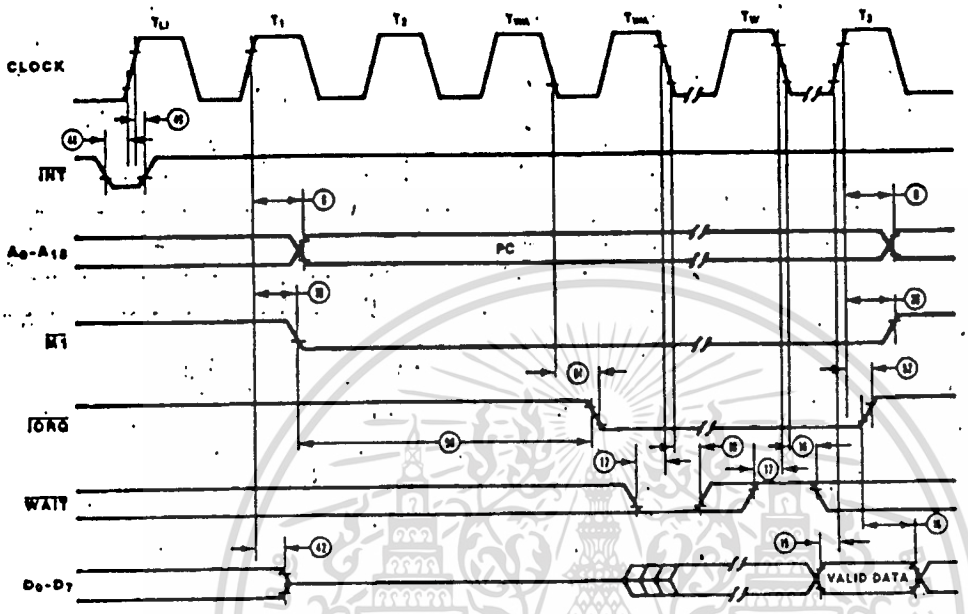
Input or Output Cycles. Figure 7 shows the timing for an I/O read or I/O write operation. During I/O operations, the CPU automatically inserts a single Wait state (T_{WA}). This extra Wait state allows sufficient time for an I/O port to decode the address from the port address lines.



T_{WA} = One wait cycle automatically inserted by CPU.

Figure 7. Input or Output Cycles

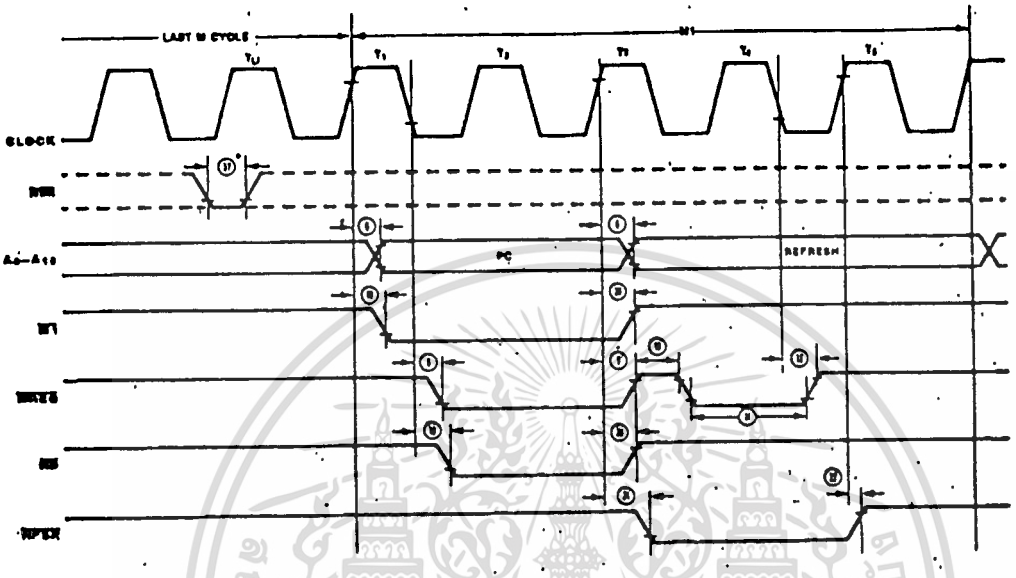
Interrupt Request/Acknowledge Cycle. The CPU samples the interrupt signal with the rising edge of the last clock cycle at the end of any instruction (Figure 8). When an interrupt is accepted, a special $\overline{M1}$ cycle is generated. During this $\overline{M1}$ cycle, $\overline{IORQ}$ becomes active (instead of $\overline{MREQ}$) to indicate that the interrupting device can place an 8-bit vector on the data bus. The CPU automatically adds two Wait states to this cycle.



NOTES: 1) T_{LI} = Last state of any instruction cycle.
2) T_{WA} = Wait cycle automatically inserted by CPU.

Figure 8. Interrupt Request/Acknowledge Cycle

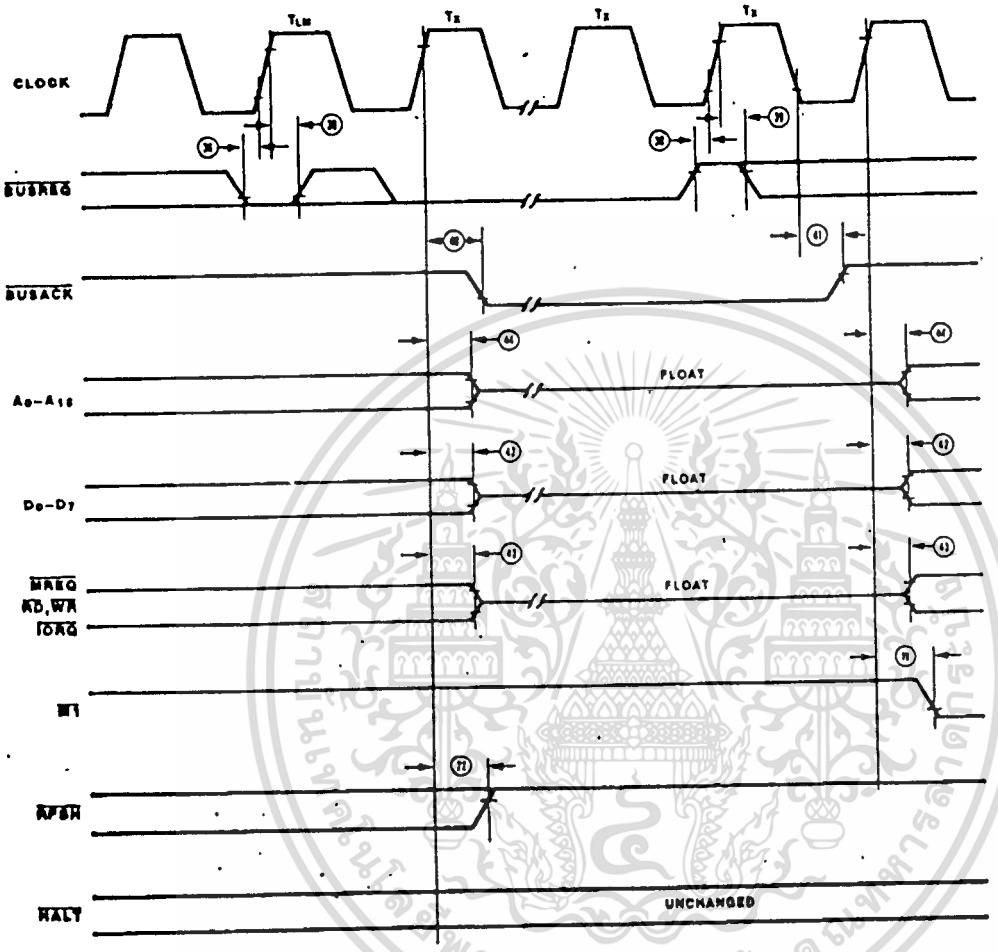
Non-Maskable Interrupt Request Cycle. $\overline{NMI}$ is sampled at the same time as the maskable interrupt input $\overline{INT}$ but has higher priority and cannot be disabled under software control. The subsequent timing is similar to that of a normal memory read operation except that data put on the bus by the memory is ignored. The CPU instead executes a restart (RST) operation and jumps to the $\overline{NMI}$ service routine located at address 0066H (Figure 9).



*Although $\overline{NMI}$ is an asynchronous input, to guarantee its being recognized on the following machine cycle, $\overline{NMI}$'s falling edge must occur no later than the rising edge of the clock cycle preceding the last state of any instruction cycle (T_{LS}).

Figure 9. Non-Maskable Interrupt Request Operation

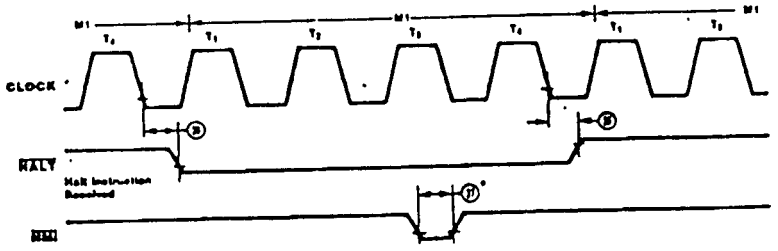
us Request/Acknowledge Cycle. The CPU samples $\overline{BUSREQ}$ with the rising edge of the last clock period of any achine cycle (Figure 10). If $\overline{BUSREQ}$ is active, the CPU ts its address, data, and $\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$, and $\overline{WR}$ lines to a high-impedance state with the rising edge of the next clock pulse. At that time, any external device can take control of these lines, usually to transfer data between memory and I/O devices.



NOTES: 1) T_{LM} = Last state of any M cycle.
2) T_X = An arbitrary clock cycle used by requesting device.

Figure 10. Z-BUS Request/Acknowledge Cycle

Halt Acknowledge Cycle. When the CPU receives a HALT instruction, it executes NOP states until either an $\overline{INT}$ or $\overline{NMI}$ input is received. When in the Halt state, the $\overline{HALT}$ output is active and remains so until an interrupt is received (Figure 11). INT will also force a Halt exit.



* Although $\overline{NMI}$ is an asynchronous input, to guarantee its being recognized on the following machine cycle, $\overline{NMI}$'s falling edge must occur no later than the rising edge of the clock cycle preceding the last state of any instruction cycle (T_{LA}).

Figure 11. Halt Acknowledge Cycle

Reset Cycle. $\overline{RESET}$ must be active for at least three clock cycles for the CPU to properly accept it. As long as $\overline{RESET}$ remains active, the address and data buses float, and the control outputs are inactive. Once $\overline{RESET}$ goes inactive, two internal T cycles are consumed before the CPU resumes normal processing operation. $\overline{RESET}$ clears the PC register, so the first opcode fetch will be to location 0000H (Figure 12).

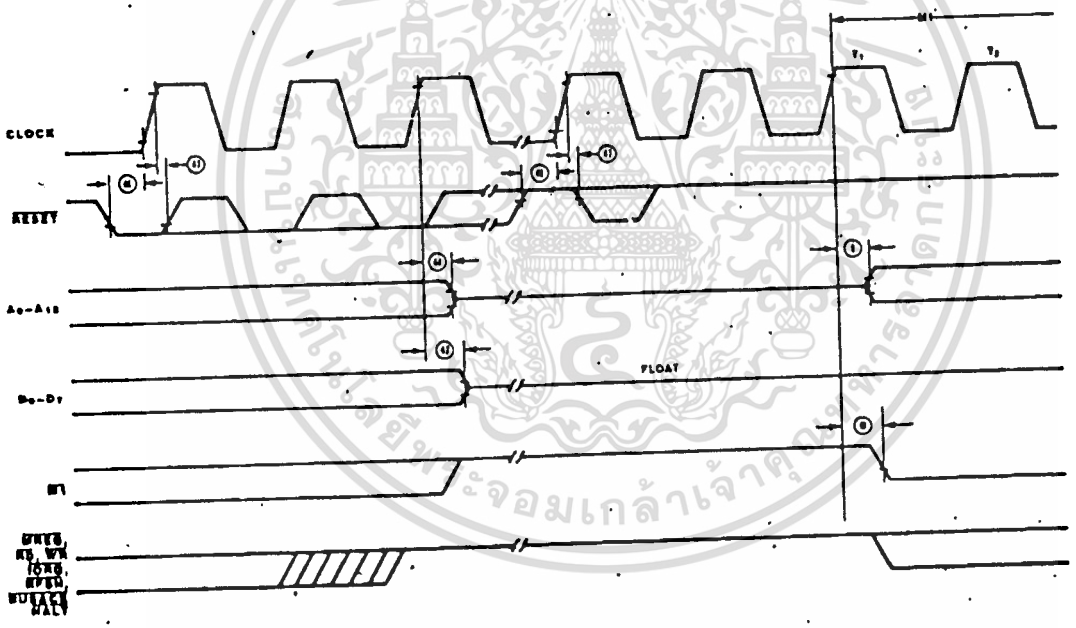


Figure 12. Reset Cycle

กิตติกรรมประกาศ

ผู้จัดทำขอขอบพระคุณ ผศ. กิตติ ตีระเศรษฐ และ อ.ธนิตย์ ตรีสุวรรณวัฒน์ อาจารย์ที่ปรึกษา ที่ให้คำแนะนำ และให้การสนับสนุนทั้งทางด้านเอกสารและ อุปกรณ์ทดลอง จนกระทั่งงานสำเร็จลุล่วง สามารถนำชุดทดลองนี้ไปใช้ทดลองประกอบการเรียน การสอน ได้ตามวัตถุประสงค์

หนังสืออ้างอิง

1. ชูชัย ธนสารดั้งเจริญ และ ดนัย แสงอุริยศิลป์, " การใช้งาน Z-80 " หน้า 51-141, พิสิกส์เซ็นเตอร์ การพิมพ์ , 2528.
2. รศ.ยืน ภู่วรวาณ, " ทฤษฎีและการประยุกต์ไมโครโปรเซสเซส Z-80 " หน้า 96-128, บริษัท ซีเอ็ดยูเคชั่น จำกัด, 2523.
3. COFFRON J.W, " Z-80 APPLICATIONS ", ZYBEX INC.,1983.
4. SOLLIBES, MARK GARETE, " INTERFACEING TO S-100/IEEE 696 MICRO COMPUTER ", OSBORNE/MCGRAW-HILL, BERKELEY, CALIFORNIA (USA), PP.139-173, 1987.
5. MULTITECH INDUSTRIAL CORPORATION " MICROPROFESSOR MPF-I MONITOR PROGRAM SOURCE LISTING ", 1981.
6. MULTITECH INDUSTRIAL CORPORATION." MICRO PROFESSOR USER MANUAL", 1981.