เครื่องควบคุมตำแหน่ง X-Y โดยอัตโนมัติ

AUTOMATIC X-Y POSITION CONTROL

โดย

นาย โชคชัย สว่างญาติ

นาย ปิยะวุฒิ ช่างสุวรรณ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

เครื่องควบคุมตำแหน่ง X-Y โดยอัตโนมัติ
AUTOMATIC X-Y POSITION CONTROL

โดย

นาย โชคชัย สว่างญาติ

นาย ปิยะวุฒิ ช่างสุวรรณ

อาจารย์ที่ปรึกษา

ผศ.ดร. สุรพันธ์ เอื้อไพบูลย์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

ปริญญานิพนธ์ปีการศึกษา 2541

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องควบคุมตำแหน่ง X-Y โดยอัตโนมัติ

ผู้จัดทำ

นาย โชคชัย สว่างญาติ เลขประจำตัว 39013156

นาย ปิยะวุฒิ ช่างสุวรรณ เลขประจำตัว 39013166

.....อาจารย์ที่ปรึกษา

(ผศ.ดร. สุรพันธ์ เอื้อไพบูลย์)

เครื่องควบคุมตำแหน่ง X-Y โดยอัตโนมัติ

นาย โชคชัย สว่างญาติ
นาย ปิยะวุฒิ ช่างสุวรรณ
ดร. สุรพันธ์ เอื้อไพบูลย์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2541

บทคัดย่อ

โครงการนี้จะนำเอาหลักการของ ไมโครคอนโทรลเลอร์ หลักการของการควบคุมชุดสเตปปิ้งมอเตอร์ เอามาประยุกต์ใช้ในการควบคุมตำแหน่งที่ต้องการ ซึ่งเป็นที่มาของโครงงาน " เครื่องควบคุมตำแหน่ง X-Y โดยอัตโนมัติ "

ในโครงการนี้จะใช้ผลที่ได้จากการประมวลผลจากภาพ และนำผลที่ได้นี้มาควบคุมไมโครคอนโทรลเลอร์ ด้วยการอินเตอร์เฟสกับคอมพิวเตอร์ โดยจะใช้การเคลื่อนที่ด้วยสเตปปิ้งมอเตอร์ ซึ่งทำหน้าที่เป็นต้นกำลัง และจะเคลื่อนที่ไปตามรางเหล็ก ที่เป็นลักษณะในแนวแกน X และแกน Y ซึ่งจะเป็นหลักในการควบคุมตำแหน่งต่างๆได้

AUTOMATIC X-Y POSITION CONTROL

Chokchai Swangyart

Piyavot Changsuwan

Dr. Surapan Airphaiboon advisor

1998

Abstract

This project takes principle of microcontroller and stepping motor control to apply in controlling position by Image processing Method.

We will use result from image processing and take it to control microcontrolling by interfacing . Stepping motor is power ' s source for moving along steel groove by X axis and Y axis.

สารบัญ

บทที่ 1
บทนำ

ปัจจุบันได้มีการนำเอาไมโครโปรเซสเซอร์มาใช้งานกันอย่างกว้างขวาง ส่วนใหญ่จะนำมาใช้ควบคุมอุปกรณ์ต่างๆ อาทิเช่น เตาอบไมโครเวฟ วิทยุติดรถยนต์ วิทยุมือถือ ตลอดจนถึงรีโมทเครื่องปรับอากาศ เหตุที่นำมาประยุกต์เพราะมีขีดความสามารถสูงทำงานได้รวดเร็วมีขนาดเล็ก การเปลี่ยนแปลงฟังก์ชันในการทำงานเพียงแต่เปลี่ยนโปรแกรมในการทำงานเท่านั้น โดยไม่ต้องเปลี่ยนแปลงฮาร์ดแวร์เลย โครงสร้างไมโครโปรเซสเซอร์ที่นำมาใช้ในการทำงานจะต้องต่อร่วมกับหน่วยความจำและอุปกรณ์อินพุทเอ้าท์พุทพอร์ท

ปัจจุบันได้มีการรวมวงจรทั้งหมดไว้ในชิพเดี่ยวขนาด 40 ขา ซึ่งทำให้ลักษณะงานของการควบคุมเฉพาะอย่างของเครื่องมือ หรือเครื่องวัดเฉพาะอย่างนั้น ไม่จำเป็นจะต้องใช้องค์ประกอบที่มีอยู่ในไมโครคอมพิวเตอร์ทั้งหมด จึงทำให้เกิดมีการนำเอาหน่วยต่างๆที่มีองค์ประกอบที่จำเป็นเฉพาะในการทำงานควบคุมชนิดนั้นๆ มารวมเป็นหน่วยเดียวกัน ทำให้ขนาดลดลงเหลือขนาดเท่ากับชิบไมโครโปรเซสเซอร์ทั่วๆไป ที่มีขนาดตั้งแต่ 40-62 ขา เป็นชิบตัวเดียวที่มีลักษณะการใช้งานส่วนใหญ่จะใช้ในด้านการควบคุมงานอัตโนมัติต่างๆ ซึ่งโดยพื้นฐานก็ยังคงมีลักษณะการทำงานเช่นเดียวกับไมโครคอมพิวเตอร์ แต่เมื่อมีการนำไปใช้งานด้านการควบคุมเป็นหลักจึงมีชื่อเรียกใหม่ว่า MICROCONTROLER หรือที่ทั่วๆไปเรียกว่า SINGLE CHIP

1.1 ลักษณะทั่วไปของ MICROCONTROLER

1. สร้างโดยใช้ CHMOS เทคโนโลยีการทำงานด้วยแหล่งจ่ายไฟขนาด 5V เพียงแหล่งเดียว
2. ซีพียูมีขนาด 8บิท
3. มีวงจรออสซิลเลเตอร์และนาฬิกาบนชิบ
4. ชุดแบงค์ (BANK)รีจีสเตอร์มี 4 ชุด แต่ละชุดมีรีจีสเตอร์ 8 ตัว ทำงานเช่นเดียวกับ MCS-48
5. มีตัวจับเวลา/ตัวนับ ขนาด 16 บิต 2 จุด และสำหรับเบอร์ 8032/8052 มี 3 ชุด
6. มีพอร์ตไอโอแบบขนานสองทิศทางจำนวน 4 พอร์ต พอร์ตละ 8 บิต รวมทั้งหมดเป็น 32 เส้น แต่จะเหลือเพียง 16 เส้น สำหรับเบอร์ 8031 อีก 16 เส้น ใช้ในการเข้าถึงทางแอดเดรสและข้อมูล

7. พอร์ตแบบอนุกรมสามารถที่จะโปรแกรมการรับส่งแบบ Full Duplex ที่ความเร็วสูง
8. หนึ่งวัฏจักรคำสั่งจะใช้เวลา 1 ไมโครวินาที ด้วยการใช้คริสทัล 12 เมกะเฮิรตซ์
9. แอดเดรสข้อมูลภายนอกได้ 64 กิโลไบต์
10. แอดเดรสโปรแกรมภายนอกได้ 64 กิโลไบต์
11. สามารถกำหนดเลขที่อยู่ข้อมูลขนาดไบต์หรือบิตได้โดยตรง
12. มีซอฟต์แวร์แฟลกสำหรับผู้ใช้ที่จะกำหนดเองได้ถึง 128 ตำแหน่งบิต
13. โครงสร้างอินเตอร์รัพต์ทำได้ 5 แหล่ง และ 6 แหล่ง สำหรับ 8092/8052 พร้อมด้วยการจัดโพรโอริตี้ (Priority) ได้ 2 ระดับ
14. ตัวโพรเซสเซอร์สามารถใช้งานแบบบูลลิน (Boolean) ได้เหมาะสำหรับการใช้งานควบคุม
15. มีคำสั่งคูณ และหารทางฮาร์ดแวร์ทำได้ภายใน 4 ไมโครวนาที
16. ตัวเลขทางคณิตศาสตร์ ใช้ได้ทั้งแบบไบนารี และเดซิมัล
17. การใช้พื้นที่สแต็กสำหรับโปรแกรมย่อยต่างๆ ทำได้ง่ายและกว้าง
18. ชุดคำสั่งของ MCS-51 โดยการกำหนดเลขที่ (ADDRESSING MODE) ได้มากกว่าชุดคำสั่งของ MCS-51

ตระกูล MSC-51 จะมีทั้งแบบมี ROM ในตัว หรือไม่มี ROM หรือมี EPROM บนชิปเดียวกัน และจะมีตำแหน่งขาที่เหมือนกัน ตารางที่ 1 แสดงถึงรายละเอียดของเบอร์ต่างๆ ในตระกูล MCS-51 ที่มีจำหน่ายในท้องตลาด

เบอร์	หน่วยความจำภายใน		ตัวตั้งเวลา	อินเตอร์รัพต์
	โปรแกรม	ข้อมูล	ตัวนับจำนวน	
8052 AH	8K X 8 ROM	256 X 8 RAM	3 X 16 BIT	6
8051 AH	4K X 8 ROM	128 X 8 RAM	2 X 16 BIT	5
8051	4K X8 ROM	128 X 8 RAM	2 X 16 BIT	5
8052 AH	ไม่มี ROM	256 X 8 RAM	3 X 16 BIT	6
8031 AH	ไม่มี ROM	128 X 8 RAM	2 X16 BIT	5
8031	ไม่มี ROM	128 X 8 RAM	2 X 16 BIT	5
8751 H	4K X 8 EPROM	128 X 8 RAM	2 X 16 BIT	5
8751 H-12	4K X 8 EPROM	128 X 8 RAM	2 X 16 BIT	5

ตารางที่ 1 รายละเอียดของตระกูล MSC-51

8751 H อยู่ในกลุ่มรุ่นเดียวกับ 8051 AH ที่เราสามารถโปรแกรมได้ด้วยระบบไฟ และสามารถลบโปรแกรมออกได้ด้วยแสงอุลตร้าไวโอเล็ต นอกเหนือจากไอซีที่แสดงในตารางข้างบนที่ใช้เทคโนโลยี HMOS แล้วยังมีตระกูลอื่นที่ใช้เทคโนโลยี CHMOS ที่ประหยัดพลังงานได้มากกว่า 4 เท่าของ HMOS ที่มีจำหน่ายขณะนี้คือ เบอร์ 80C51 และ 87C51

สำหรับ Project นี้ได้ประยุกต์ใช้งาน MICROCONTROLER เบอร์ 8951 เพื่อใช้ในการควบคุมการเคลื่อนที่ในแนวแกน X และแกน Y โดยได้นำเอา Stepping Motor มาใช้ในการขับ ให้ทำงาน ซึ่งจะทำให้เราสามารถควบคุมการเคลื่อนที่ในแนวแกน X และแกน Y ได้

1.2 ลักษณะทั่วไปของ STEPPING MOTOR

Stepping Motor เป็นมอเตอร์ที่หมุนทีละ Step โดยแต่ละ Step มอเตอร์จะหมุนด้วยมุมคงที่ค่าหนึ่ง ซึ่งในการควบคุมการหมุนของ Stepping Motor นั้นจะอาศัยวงจรควบคุมทางดิจิตอล โดยวงจรทางดิจิตอลนี้จะทำหน้าที่ในการจัดลำดับการกระตุ้นในแต่ละเฟสของ Stepping motor ซึ่งจะทำให้สามารถกำหนดทิศทางการหมุน ความเร็วในการหมุนและตำแหน่งที่ต้องการจะเลื่อนไปของ Stepping Motor ได้อย่างถูกต้องแม่นยำ

เนื่องจากวงจรทางดิจิตอลที่ใช้ในการควบคุมการหมุนของ Stepping Motor สามารถกำหนดความเร็วในการหมุนและตำแหน่งที่ต้องการจะเลื่อนไปของ Stepping Motor ได้อย่างถูกต้องแม่นยำ ดังนั้นจึงไม่จำเป็นต้องมีการ Feedback เพื่อควบคุมความเร็วและตำแหน่งเพราะฉะนั้นอาจจะกล้าวได้ว่าระบบควบคุมการหมุนของ Stepping Motor เป็นแบบ open loop control system แต่ในการหมุนของ Stepping Motor นั้นมีข้อเสียคือในบางช่วงของความเร็วในการหมุนของ Stepping Motor อาจจะทำให้เกิดการออสซิเลทที่แกนเพราของมอเตอร์ได้ และอาจจะทำให้มอเตอร์เกิดการออสซิเลทได้

สำหรับการประยุกต์การใช้งานของ Stepping Motor มีดังนี้คือ

1. อุปกรณ์ประกอบการทำงานของคอมพิวเตอร์
   - Serial printer
   - X-Y plotter
   - Floppy disk driver
2. ระบบ Numerical control
   - X-Y table and index table
   - Driller machine
   - Automatic drafting machine
3. การประยุกต์ใช้งานอื่น
   - Facsimiles
   - Semiautomatic wiring unit
   - Application in spec science

สำหรับ Project นี้ได้ประยุกต์ใช้งาน Stepping Motor เพื่อใช้ในการควบคุมการเคลื่อนที่ในแนวแกน X และแกน Y โดยได้นำเอา Stepping Motor มาใช้ในการขับ Power Screw ให้ทำงาน ซึ่งจะทำให้เราสามารถควบคุมการเคลื่อนที่ในแนวแกน X และแกน Y ได้

1.3 โปรแกรมจากคอมพิวเตอร์

สำหรับในส่วนของโปรแกรมที่ใช้สำหรับการประมวลผล ในการรับภาพจากกล้องวีดีโอ ซึ่งมีการ

บันทึกในลักษณะของ File.bmp มาทำการประมวลผลนั้นในโครงงานชิ้นนี้ได้เลือกที่จะใช้ Delphi 3.0 เป็นโปรแกรมสำหรับการพัฒนา Application program ใน window ซึ่งจากการพิจารณาแล้วพบว่าเหมาะกับการใช้ด้วยสาเหตุหลายประการ

1.3.1 การติดต่อกับผู้ใช้งาน

โครงงานนี้เขียนโดยใช้ Delphi 3.0 ซึ่งเป็นการเขียนโปรแกรมบน window ดังนั้นลักษณะของการติดต่อกับผู้ใช้งานก็จะใช้ลักษณะเช่นเดียวกันกับ window เช่น

-Form

-Button Click

-Edit Text

-Scroll bar

ซึ่งทำให้ผู้ใช้งานที่เคยใช้ระบบ windows ง่ายต่อการเข้าใจในการใช้งานไม่ต้องเสียเวลาในการทำความเข้าใจกับส่วนติดต่อกับผู้ใช้งานมากนัก และในการออกแบบส่วนติอต่อกับผู้ใช้งานโดยการใช้ Delphi ก็สามารถที่จะทำได้ง่าย เลือกรูปแบบได้หลายรูปแบบทำให้ติดต่อกับผู้ใช้มีลักษณะที่สวยงามเป็นการดึงดูดผู้ใช้งานให้มาสนใจได้อย่างดี

1.3.2 สภาพแวดล้อมในการเขียนโปรแกรม

- การใช้ Delphi เป็นการเขียนโปรแกรมในลักษณะของ Object ภาษาที่ใช้ใน Delphi คือ ภาษา Pascal (Object Pascal) ในการเขียนโปรแกรมและ Delphi ยังสนับสนุนการพัฒนาโปแกรมแบบ Visual Programming ซึ่งเราสามารถที่จะเลือก Component ที่ได้ทำการสร้างไว้แล้วมาวางบน Form และเพียงแต่กำหนดคุณสมบัติบางอย่างของคอมโพแนนต์ตามที่เราต้องการเท่านั้น

- การเขียนโปรแกรมใน Delphi จะเป็น Event คือขึ้นอยู่กับเหตุการณ์ที่เรากำหนด ซึ่งเป็นการเขียนโปรแกรม Event เช่น

Botton Click

Formcreate

OnTimer

โปรแกรมใด ๆ ที่เราเขียนขึ้นมาก็จะมีการตอบสนองต่อ Event ใด ๆ ไม่เหมือนกันขึ้นอยู่กับการกำหนดของผู้เขียน

- สนับสนุนการสร้าง Component หรือการสร้าง Object ขึ้นมา ทำให้การเขียนโปรแกรมทำได้ง่าย โดยการเขียนแยกกันในแต่ละส่วนให้สำเร็จไปทีละส่วน โดยตัวโปรแกรม

หลักจะมองเห็น Component หรือ Object เป็นเพียงวัตถุหนึ่ง Component หรือ object ที่เขียนขึ้นมาสำเร็จแล้ว ในตัวโปรแกรมทุกส่วนจะอยู่ภายในโปรแกรมเดียวกันหมด ซึ่งอาจจะวุ่นวายในการแก้ปัญหา โดยเฉพาะถ้าเป็นโปรแกรมที่มีขนาดใหญ่ ๆ เราก็สามารถที่จะแยกโปรแกรมเป็นส่วนต่างๆแยกจากกัน โดยการเขียนเป็น Object หรือ Component ขึ้นมา

- สามารถตรวจสอบการทำงานของโปรแกรมหรือข้อผิดพลาดในการเขียนโปรแกรมได้ง่าย ซึ่ง Delphi สามารถที่จะแสดงข้อผิดพลาดของเราได้ทุกส่วน เช่น การใช้เครื่องหมายไม่ถูกต้อง ตัวแปรที่ไม่ทราบ หรือการเขียนโปรแกรมผิดหลักเกณฑ์ต่าง ๆ ในส่วนของการตรวจสอบการทำงานก็จะมีส่วนของการตรวจสอบค่านิพจน์ หรือสมการต่าง ๆ ของเราได้ทำให้ง่ายต่อการพัฒนา

1.3.3 คำสั่งที่สนับสนุนโปรแกรมในโครงงาน

สำหรับในส่วนของ soft ware ในโครงงานนี้ ส่วนหลัก ๆ จะแบ่งได้เป็น 3 ส่วน คือ

-การรับภาพจากกล้องแล้วทำการ save ภาพนั้นเป็น File.bmp. ในส่วนนี้ได้ใช้โปรแกรมสำเร็จรูป

-การนำภาพนั้นมาประมวลผล (image processing) อันได้แก่ การทำระดับภาพให้มีเพียง 2 คือ ขาวและดำ การหาขอบภาพและการทำ contour คำสั่งหลักที่สนับสนุนการทำงานในส่วนนี้คือ Canvas.Pixels[x,y] เป็นการอ้างถึง Pixels ที่ตำแหน่ง [x,y] บนจอภาพซึ่ง Canvas.Pixels[x,y] นี้จะเก็บค่าสีที่ตำแหน่ง Pixels[x,y] ไว้ดังนี้ก็จะมีลักษณะ 2 ลักษณะนี้ การนำค่าสี Pixels[x,y] นั้นมาตรวจสอบและการกำหนดค่าสี Pixels[x,y] ให้กับ

- การนำค่าจุด [x,y] ที่ได้จากการ Contour แล้วส่งออกทาง Serial port ของ computer ในส่วนนี้ได้เลือกใช้ Component RS232 ซึ่งได้ทำการ Download มาใช้งานเป็น freeware ซึ่งสามารถส่งค่าออกทาง Serial port ได้อย่างดี

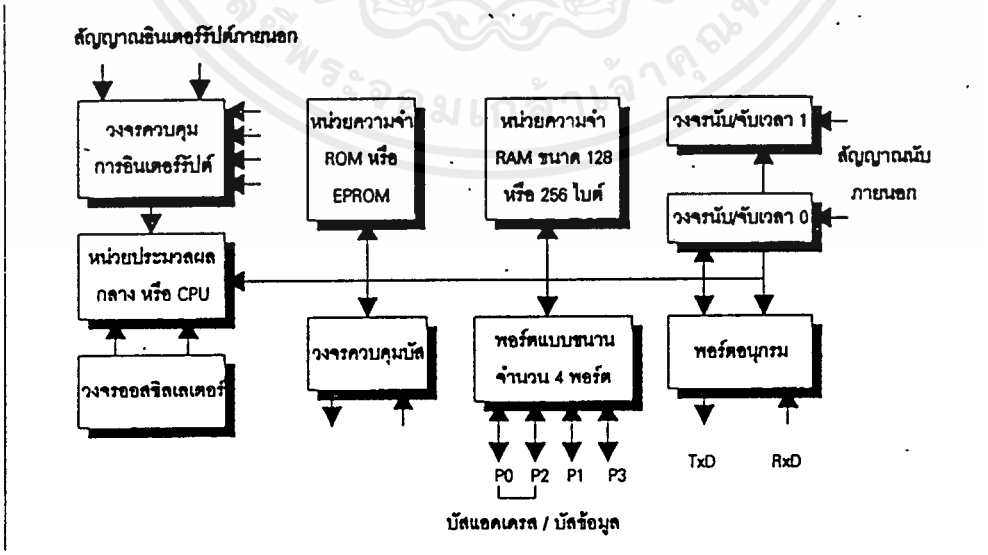
บทที่ 2
ทฤษฎีการทำงาน

2.1 ไมโครคอนโทรลเลอร์ตระกูล 8051

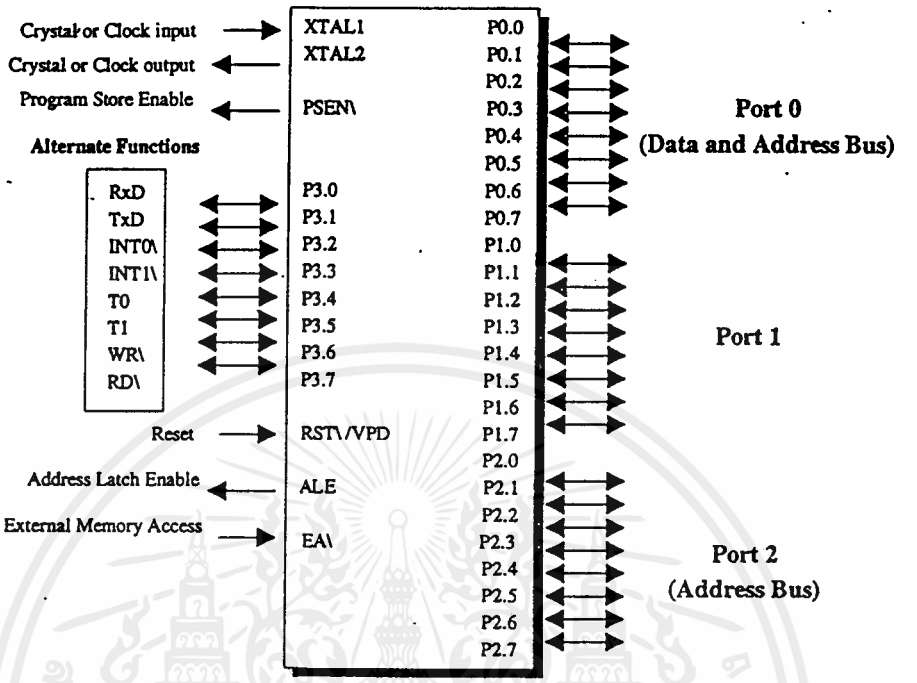
ไมโครคอนโทรลเลอร์เป็นไมโครโปรเซสเซอร์ประเภทหนึ่งที่ได้รับการออกแบบมาเพื่อใช้งานกับระบบควบคุมที่มีขนาดเล็ก โดยภายในไอซีไมโครคอนโทรลเลอร์หนึ่งตัวจะประกอบด้วยหน่วยการทำงานหลักของระบบคอมพิวเตอร์ครบถ้วน เช่น หน่วยประมวลผลกลางหรือซีพียู (CPU) หน่วยความจำ พอร์ตในการติดต่อหรือควบคุมอุปกรณ์ต่างๆ เป็นต้น ซึ่งหากว่าเป็นการใช้งานไมโครโปรเซสเซอร์ทั่วไปก็จะต้องใช้ไอซีภายนอกมาประกอบเพื่อทำหน้าที่ เหล่านี้ ดังนั้นจึงอาจกล่าวได้ว่าไมโครคอนโทรลเลอร์เป็นระบบคอมพิวเตอร์เพื่องานควบคุมที่สมบูรณ์ โดยบรรจุอยู่ภายในไอซีเพียงหนึ่งตัวเท่านั่น ในบางครั้งจึงอาจพบว่ามีการเรียกไมโครคอนโทรลเลอร์ว่าเป็น ระบบไมโครคอมพิวเตอร์ชิปเดี่ยว (1 chip microcomputer)

ไมโครคอนโทรลเลอร์ตระกูล MCS-51 ประกอบด้วย ไมโครคอนโทรลเลอร์หลายรุ่น (Version) ซึ่งมีสถาปัตยกรรมพื้นฐานที่เหมือนกัน เพียงแต่มีขนาดหรือจำนวนของหน่วยทำงานภายในที่ต่างกันออกไป เพื่อความเหมาะสมในงานประยุกต์ต่างๆตามความต้องการ โดยมีทั้งลักษณะที่ใช้เทคโนโลยีการผลิตไอซีวงจรรวมความจุสูงมาก (LSI) แบบ HMOS หรือ CHMOS ซึ่งมีคุณลักษณะที่สูงมากขึ้น และสิ้นเปลืองกำลังไฟฟ้าน้อยกว่ามาก

2.2 คุณลักษณะพื้นฐานของ 8051



รูปที่ 2.1 แสดงให้เห็นถึงหน่วยการทำงานพื้นฐานของไมโครคอนโทรลเลอร์เบอร์ต่างๆที่จัดอยู่ภายใน



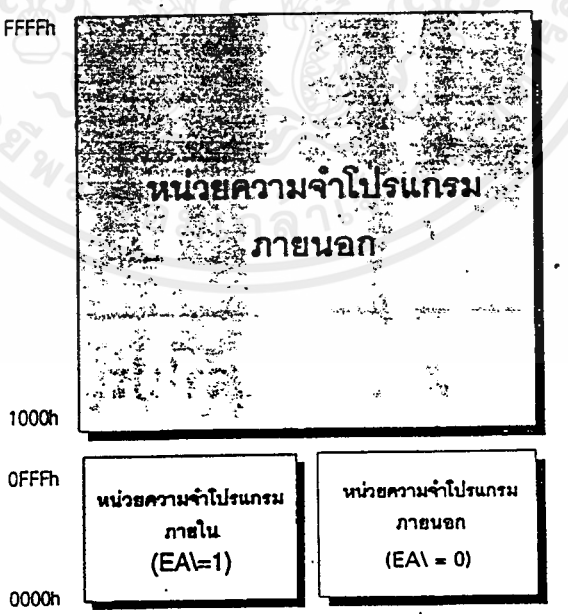
รูปที่ 2.2 การกำหนดขาสัญญาณของไอซี 8051

โดยมากแล้วไมโครคอนโทรลเลอร์ตระกูลนี้ มักจะมีรูปร่างของไอซีเป็นแบบ DIP ขนาด 40 ขา ดังแสดงเป็นแผนภาพในรูปที่ 4 ซึ่งแต่ละขาสัญญาณจะมีหน้าที่ที่ระบุชัดเจนตามสัญลักษณ์ ชื่อย่อที่กำกับในแต่ละขา อย่างไรก็ตามจะมีบางขาสัญญาณที่อาจจะมีหน้าที่ได้มากกว่าหนึ่งอย่าง (ซึ่งเขียนกำกับไว้ว่า Alternate Function ในรูปที่ 2.2 ซึ่งจะไม่สามารถใช้งานในเวลาเดียวกันได้ ตัวอย่างเช่น ขาสัญญาณบิต 0 ของพอร์ต 3 (ใช้ตัวย่อเป็น P3.0) อาจจะใช้เป็นขาสัญญาณเอาต์พุต หรืออินพุตตามปกติ หรืออาจทำหน้าที่เป็นขาสัญญาณอินพุตของข้อมูลสื่อสารแบบอนุกรม (RxD) ให้กับวงจรสื่อสารแบบอนุกรมของ 8051 ได้ ซึ่งการจะกำหนดว่าจะทำงานในลักษณะใดก็ขึ้นอยู่กับ การเชื่อมต่อวงจรเข้ากับขาสัญญาณและโปรแกรมควบคุมของระบบนั้น

2.3 หน่วยความจำโปรแกรมของ 8051

ในระบบของไมโครคอนโทรลเลอร์ 8051 จำเป็นต้องมีหน่วยความจำสำหรับบรรจุคำสั่งหรือโปรแกรมที่ผู้ใช้พัฒนาขึ้นจัดเก็บไว้ภายในหน่วยความจำ ที่เรียกว่า หน่วยความจำโปรแกรม (Program Memory) โดยอาจจะประกอบอยู่ภายในตัวไอซีของ 8051 เองหรือเป็นไอซีหน่วยความจำ EPROM หรือ ROM แยกออกต่างหากได้

หน่วยความจำโปรแกรมของ 8051 เป็นบริเวณหน่วยความจำ สำหรับเก็บข้อมูลและคำสั่งใช้งานต่างๆซึ่งแม้ว่าจะไม่มีการจ่ายกระแสไฟฟ้าให้กับระบบ ข้อมูลเหล่านี้ก็ยังคงอยู่ไม่สูญหายโครงสร้างของหน่วยความจำโปรแกรม มีลักษณะเช่นเดียวกับหน่วยความจำที่บรรจุอยู่ในไอซีหน่วยความจำประเภทต่างๆ เช่น หน่วยความจำแบบ ROM หรือ EPROM 8051 สามารถอ่านข้อมูลหน่วยความจำโปรแกรมนี้สูงสุดได้ไม่เกิน 64 กิโลไบต์ และแยกประเภทของหน่วยความจำโปรแกรมเป็นสองลักษณะ ตามตำแหน่งของหน่วยศวามจำโปรแกรมนั้น คือ หน่วยความจำโปรแกรมภายใน(Internal Program Memory)ซึ่งเป็นหน่วยความจำ ROM หรือ EPROM ที่อยู่ภายในตัวไอซีไมโครคอนโทรลเลอร์เอง และ หน่วยความจำโปรแกรมภายนอก (External Program Memory) ซึ่งเป็นการใช้ไอซีหน่วยความจำมาทำหน้าที่เป็นหน่วยความจำ โปรแกรมของระบบโดยการจัดพื้นที่หน่วยความจำโปรแกรมสำหรับไมโครคอนโทรลเลอร์ 8051 สามารถแสดงได้ดังรูปที่ 2.3



รูปที่ 2.3 การจัดพื้นที่หน่วยความจำโปรุแกรมสำหรับ MCS-51

2.3.1หน่วยความจำโปรแกรมภายใน

ไมโครคอนโทรลเลอร์เบอร์ต่างๆ ที่จัดอยู่ในตระกูล 8051 นี้มีขนาดของหน่วยความจำโปรแกรมภายในแตกต่างกันออกไป เพื่อความเหมาะสมกับการนำไปใช้งานในลักษณะต่างๆกัน ดังนี้

8051 และ 8052 มีหน่วยความจำแบบ ROM ขนาด 4 และ 8 กิโลไบต์ ตามลำดับ ประกอบอยู่ภายในไอซี และมีความเหมาะสมกับการนำไปใช้ในวงจรทางอุตสาหกรรมที่มีจำนวนการผลิตมาก เนื่องจากจะมีผลทำให้ต้นทุนค่าใช้จ่ายการผลิตต่อหน่วยลดลงได้มาก

8751 มีหน่วยความจำแบบ EPROM ขนาด 4 กิโลไบต์อยู่ภายในไอซี ข้อมูลที่จัดเก็บอยู่ภายในนี้สามารถใช้แสงอัลตร้าไวโอเลตลบได้ และนำไปบรรจุโปรแกรมใหม่ได้อีกครั้งหนึ่ง คล้ายครึ่งกับไอซีหน่วยความจำ EPROM ไมโครคอนโทรลเลอร์เบอร์นี้เหมาะสมกับงานด้านอุตสาหกรรมที่มีจำนวนการผลิตคราวละไม่มากนัก หรืออาจจะเป็นงานประเภทต้นแบบภายในห้องปฏิบัติการ

8031 และ 8032 ไม่มีหน่วยความจำโปรแกรมอยู่ภายในตัวไอซีเลย ดังนั้นในการสำไปใช้งานจึงจำเป็นต้องอาศัยหน่วยความจำโปรแกรมภายนอก ซึ่งการใช้งานในลักษณะนี้จะมีผลทำให้ต้องเสียความสามารถบางประการ เกี่ยวกับพอร์ตอินพุต/เอาพุตของไมโครคอนโทรลเลอร์ไป เนื่องจากต้องนำไปใช้เป็นสัญญาณควบคุม เกี่ยวกับการจัดการติดต่อหน่วยความจำภายนอกแทน

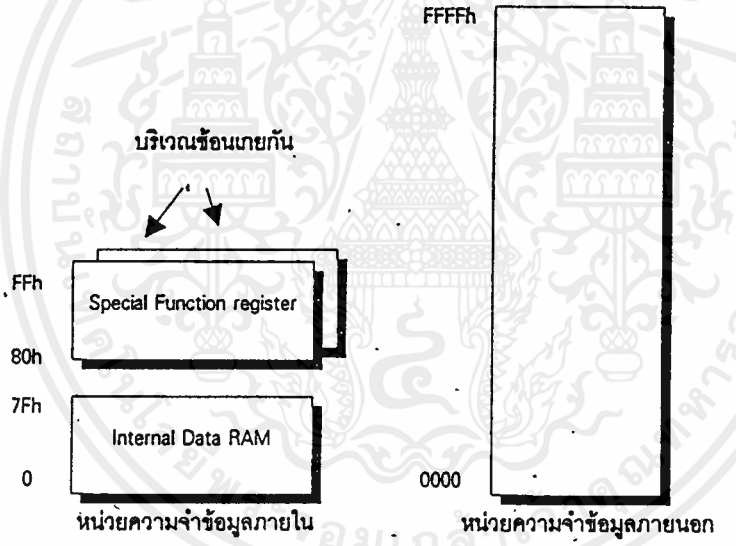
2.3.2 หน่วยความจำโปรแกรมภายนอก

หน่วยความจำโปรแกรมภายนอกเป็นการใช้หน่วยความจำ EPROM (หรือ ROM) เชื่อมต่อเข้ากับระบบของ 8051 โดยอาจจะมีสาเหตุได้หลายประการ เช่น เป็นการทดลองทำระบบต้นแบบจำนวนน้อย หรืออาจลดต้นทุนการผลิต เพราะราคาของไมโครคอนโทรลเลอร์แบบไม่มีหน่วยความจำโปรแกรมภายในราคาจะต่ำกว่าแบบที่มีหน่วยความจำโปรแกรมภายในมาก เป็นต้น ในบางครั้งอาจจะมีสาเหตุจากความจำเป็นอื่นๆ ที่ไม่อาจหลีกเลี่ยงได้ เช่น การที่หน่วยความจำภายในของไมโครคอนโทรลเลอร์มีขนาดความจุที่ไม่เพียงพอกับโปรแกรม หรืออาจจะเป็นว่าการที่ใช้ไอซีหน่วยความจำจะทำให้สามารถจัดหาเครื่องมือ(Tools) ช่วยการพัฒนาระบบที่ใช้งานกันโดยแพร่หลายและราคาถูกได้ ซึ่งจะช่วยลดเวลาในการพัฒนาระบบลงได้มากเป็นต้น

ไมโครคอนโทรลเลอร์เบอร์ต่างๆ ของตระกูล 8051 นี้ สามารถขยายให้ใช้งานหน่วยความจำภายนอกได้ทั้งสิ้น โดยในกรณีที่มีหน่วยความจำโปรแกรมภายในอยู่แล้ว การอ้างตำแหน่งแอดเดรสที่มีทั้งในหน่วยความจำโปรแกรมภายในและภายนอกนั้น จะต้องทำการพิจารณาระดับลอจิกของสัญญาณของสัญญาณ EA

2.3.3 หน่วยความจำข้อมูลของ 8051

หน่วยความจำข้อมูลมีหน้าที่สำหรับเก็บข้อมูล หรือตัวแปรที่เกิดขึ้นในขณะที่กำลังประมวลผลโปรแกรมไว้เป็นการชั่วคราว โดยพื้นฐานแล้วหน่วยความจำข้อมูลจัดเป็นหน่วยความจำ RAM แบบสเตติก ดังนั้นเมื่อไม่มีการจ่ายกระแสไฟฟ้าให้กับระบบ ก็จะมีผลทำให้ข้อมูลที่จัดเก็บไว้ภายนหน่วยความจำนี้สูญหายไป พื้นที่ของหน่วยความข้อมูลของ 8051 สามารถมีได้สูงสุดไม่เกิน 64 กิโลไบต์ และแยกประเภทออกเป็นสองลักษณะตามตำแหน่งที่ตั้งของหน่วยความจำนั้น ดังแสดงในแผนภาพในรูปที่ 6 คือ หน่วยความจำมูลภายใน (Internal Data Memory) ซึ่งเป็น RAM ที่อยู่ภายในตัวของไมโครคอนโทรลเลอร์เอง และหน่วยความจำข้อมูลภายนอก (External Data Memory) ซึ่งเป็นการใช้ไอซีหน่วยความจำ RAM มาเพิ่มเติมเข้าในวงจร ลักษณะเดียวกับการนำไอซี EPROM มาใช้งานเป็นหน่วยความจำโปรแกรมนั้นเอง



รูปที่ 2.4 การจัดพื้นที่หน่วยความจำข้อมูลสำหรับ MCS-51

2.3.4 หน่วยความจำข้อมูลภายใน

หน่วยความจำข้อมูลภายในของ 8051 มีจำนวนทั้งหมด 256 ไบต์ โดยจำแนกออกได้เป็นสองลักษณะ คือ พื้นที่เฉพาะสำหรับตัวประมวลผลกลาง (หรือซีพียู) ใช้งานเท่านั้น ซึ่งเรามักจะเรียกกันอีกชื่อหนึ่งว่า รีจิสเตอร์ และพื้นที่ใช้งานทั่วไปสำหรับโปรแกรมใช้งานที่ผู้ใช้สร้างขึ้นมา

จากรูปที่ 2.5 แสดงให้เห็นถึงการจัดสรรพื้นที่ของหน่วยความจำข้อมูลภายในของ 8051 ซึ่งจำแนกออกเป็นสองส่วนดังนี้

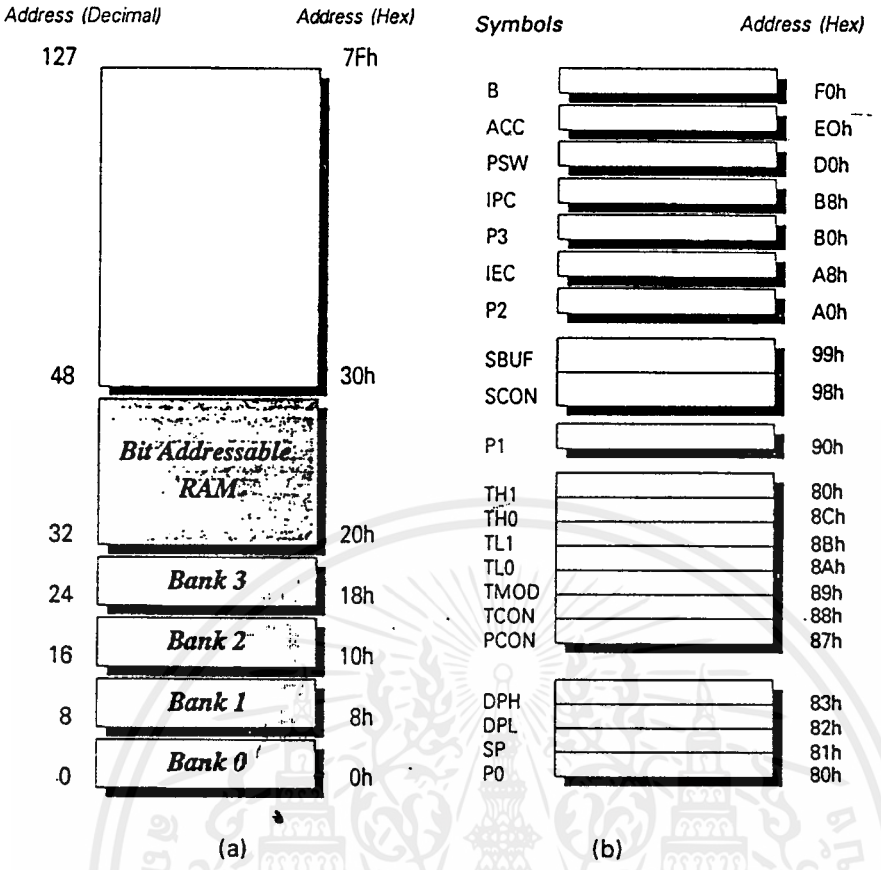
หน่วยความจำขนาด 128 ไบต์แรก บริเวณนี้จะมีตำแหน่งแอดเดรสอยู่ในช่วง 00H-7FH ซึ่งยังได้มีการจำแนกย่อยไปอีกเป็นสามส่วนตามประเภทของการใช้งาน ดังนี้

บริเวณแอดเดรส 00H-1FH จำนวน 32 ไบต์

แอดเดรส	รีจิสเตอร์แบงก์	ชื่อรีจิสเตอร์ใช้งาน
00H-07H	0	R0-R7
08H-0FH	1	R0-R7
10H-17H	2	R0-R7
18H-1FH	3	R0-R7

จะเห็นได้ว่าชื่อของรีจิสเตอร์ไม่ว่าจะอยู่ในรีจิสเตอร์แบงก์ใด ก็จะมีชื่อ R0-R7 เหมือนกันทั้งสิ้น (ดูรูปที่ 2.5) ดังนั้นในการใช้งานผู้ใช้จะต้องให้ความระมัดระวังว่าต้องการรีจิสเตอร์นั้นๆจากแบงก์ใด การสวิตช์เลือกแต่ละกลุ่มของรีจิสเตอร์นี้ก็ทำได้ง่าย เพียงการกำหนดค่าของบิตที่อยู่ภายในรีจิสเตอร์ PSW เท่านั้นตามตารางต่อไปนี้

รีจิสเตอร์	บิต RS0	บิต RS1	ตำแหน่งหน่วยความจำ
แบงก์ 0	0	0	0000H
แบงก์ 1	1	1	0008H
แบงก์ 2	2	0	0010H
แบงก์ 3	3	1	0018H



รูปที่ 7 การจัดพื้นที่หน่วยข้อมูลภายใน

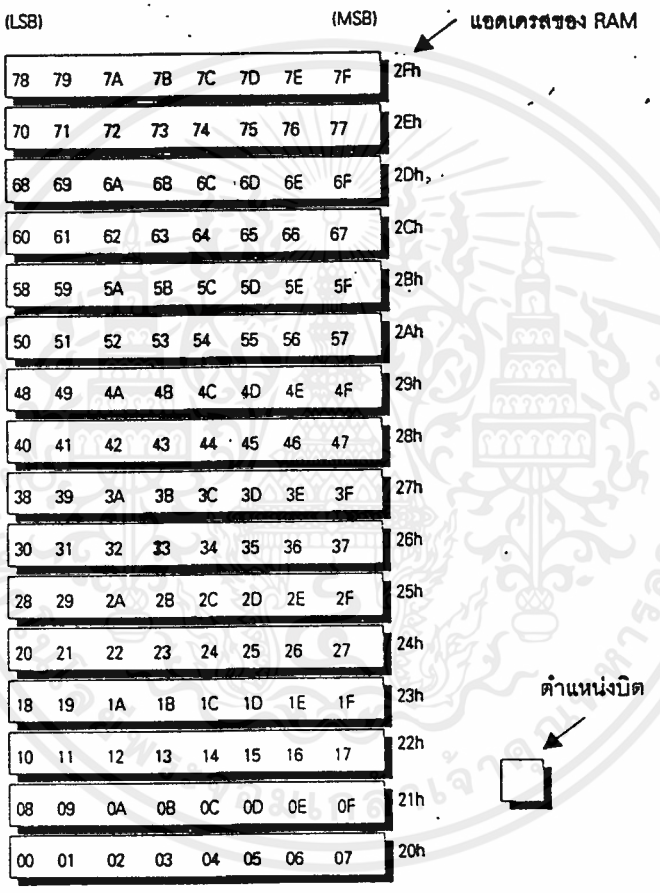
อย่างไรก็ตามโดยทั่วไปก็มักจะมีการใช้งานรีจิสเตอร์ R0-R7 เฉพาะในแบงก์ 0 เท่านั้น ดังนั้นพื้นที่ของแบงก์อื่นๆ ที่เหลือก็นำมาใช้ในลักษณะหน่วยความจำข้อมูลปกติด้วยการอ้างถึงหมายเลขภายในปกติด้วยการอ้างถึงหมายเลขของแอดเดรสนั้นๆ โดยตรง

บริเวณแอดเดรส 20H-2FH จำนวน จำนวน 16 ไบต์บริเวณพื้นที่เป็นส่วนสำคัญผู้ใช้ซึ่งจะมีความพิเศษต่างไปจากหน่วยความจำส่วนอื่นๆ เนื่องจากผู้ใช้อาจสามารถอ้างถึงหน่วยความจำบริเวณนี้ได้ทั้งในลักษณะของ ไบต์ข้อมูล เช่นปกติ หรืออาจเป็น บิตข้อมูล ได้โดยตรง ดังนั้นหากว่ามองในลักษณะบิตข้อมูลแล้ว ก็จะมีพื้นที่ที่ตัวแปรแบบบิตให้ใช้งานได้มากถึง 128 บิต โดยตำแหน่งแรกของบิตจะเป็นบิตซึ่งเริ่มต้นนับจากบิตนัยสำคัญต่ำสุด (LSB) ของแอดเดรส 2FH (ดูรูปที่ 2.6)

ความสามารถในการใช้งานพื้นที่ส่วนนี้แบบบิตข้อมูลโดยตรงนี้นับว่าน่าสนใจมาก และถือเป็นการใช้งาน 8051 อย่างเต็มประสิทธิภาพทีเดียว เนื่องจากว่า 8051 ได้รับการออกแบบมาสำหรับงานควบคุมเป็นพื้นฐานอยู่แล้ว ซึ่งส่วนมากงานลักษณะเช่นนี้หากเป็นการนำเข้าข้อมูลก็มักจะเป็นเพียงการอ่านค่าสถานะลอจิกของเส้นสัญญาณ หรือกรณีการส่งออกข้อมูลก็จะการ

กำหนดสภาวะลอจิกให้กับวงจรภายนอกผ่านทางบิตใดบิตหนึ่งอยู่แล้ว ดังนั้นหากว่ามีการกำหนดบิตหรืออ่านค่าของบิตมาโดยตรง แทนที่จะต้องทำลอจิกขั้นต้นกับข้อมูลทั้งไบต์เพื่อต้องการทราบผลเพียงหนึ่งบิตเช่นที่กระทำกันในโปรเซสเซอร์โดยทั่วไป ก็จะเพิ่มความสะดวกและรวดเร็วในการเขียนโปรแกรมควบคุมมาก รายละเอียดในส่วนนี้จะได้กล่าวถึงอีกครั้งหนึ่งเมื่อศึกษาถึงการใช้งานพอร์ตอินพุต/เอาต์พุตต่อไป

บริเวณแอดเดรส 30H-7FH เป็นบริเวณที่สามารถนำไปใช้งานได้อย่างอิสระ โดยสามารถอ้างถึงได้เฉพาะในลักษณะของไบต์ข้อมูลตามปกติเท่านั้น



รูปที่ 2.6 หน่วยความจำข้อมูลภายในบริเวณที่อ้างถึงได้แบบบิต

หน่วยความจำขนาด 128 ไบต์ถัดไป

พื้นที่ตั้งแต่บริเวณตั้งแต่แอดเดรส 80H-FFH เป็นบริเวณของหน่วยความจำที่มีการใช้งานเฉพาะจาก 8051 เท่านั้น โดยจะนำมาใช้เป็นตำแหน่งของ **รีจิสเตอร์หน้าที่พิเศษ** (Special Function Register หรือ SFR) จำนวน 20 ตำแหน่ง ดังแสดงเป็นแผนภาพในรูปที่ 3.4

สำหรับไมโครคอนโทรลเลอร์เบอร์ 8051 จะมีหน่วยความจำข้อมูลภายในสำหรับการใช้งานเพิ่มมากขึ้นกว่าเบอร์อื่นๆ เช่น 8031 หรือ 8751 อีก 128 ไบต์ โดยจะอยู่บริเวณช่วงแอดเดรส 80H ถึง FFH เช่นกัน ซึ่งแม้ว่าจะเป็นพื้นที่ที่มีหมายเลขแอดเดรสเดียวกับส่วนของรีจิสเตอร์หน้าที่พิเศษ แต่ในความเป็นจริงแล้วจะเป็นพื้นที่หน่วยความจำอีกบริเวณหนึ่ง ซึ่งมีการซ้อนเกย (Overlap) กันให้อยู่ในบริเวณแอดเดรสส่วนนี้ ซึ่งหากว่าผู้ใช้งานต้องการจะเก็บข้อมูลในพื้นที่บริเวณนี้ก็จะต้องใช้การอ้างถึงหน่วยความจำแบบโดยอ้อม (Indirect Addressing) เท่านั้น

รีจิสเตอร์หน้าที่พิเศษ

รีจิสเตอร์หน้าที่พิเศษ (SFR) เป็นรีจิสเตอร์สำหรับการควบคุมหน้าที่และการทำงานของอุปกรณ์หรือพอรต์ของ 8051 ทั้งหมด โดยมีตำแหน่งอยู่ในบริเวณแอดเดรส 80H-FFH (อ้างถึงรูปที่ 3.4) การใช้งานรีจิสเตอร์หน้าที่พิเศษเหล่านี้สามารถทำได้ทั้งการระบุถึงชื่ อของรีจิสเตอร์หรือตำแหน่งแอดเดรสที่เป็นของรีจิสเตอร์นั้นก็ได้

แอกคิวมูเลเตอร์ (Accumulator) หรือ ACC

เป็นรีจิสเตอร์ขนาด 8 บิต ทำหน้าที่ในการเก็บข้อมูลที่จะส่งให้กับหน่วยทำงานภายในซีพียูและเก็บผลลัพธ์ที่ได้จากการทำงานนั้น การทำงานของรีจิสเตอร์นี้มีลักษณะเช่นเดียวกับตัวแอกคิวมูเลเตอร์ของโปรเซสเซอร์ทั่วไป การใช้งานภายในโปรแกรมจะเรียกว่า รีจิสเตอร์ A

รีจิสเตอร์ B

เป็นรีจิสเตอร์ที่ใช้สำหรับการทำคำสั่งการคูณและการหารตัวเลข ในกรณีที่ไม่ใช้ในการคำนวนทางด้านคณิตสตร์ ก็สามารถนำไปใช้งานเช่นเดียวกับรีจิสเตอร์ทั่วไป

โปรแกรมเคาน์เตอร์ (Program Counter)

เป็นรีจิสเตอร์ที่ใช้ในการชี้ตำแหน่งแอดเดรสของหน่วยความจำโปรแกรม ซึ่งจะต้องไปทำงานในลำดับต่อไป การใช้งานภายในโปรแกรมจะเรียกว่า รีจิสเตอร์ PC

สแต็กพอยน์เตอร์ (Stack Pointer)

เป็นรีจิสเตอร์ขนาด 8 บิต ทำหน้าที่เก็บตำแหน่งของตัวชี้หรือพอยน์เตอร์ (Pointer) ของบริเวณสแตก (Stack) สำหรับเก็บข้อมูลแอกคิวมูเลเตอร์ รีจิสเตอร์ต่างๆ รวมทั้งข้อมูลจากโปรแกรมโดยปกติแล้วเมื่อทำการเริ่มต้นระบบใหม่ภายหลังจากการเริ่มจ่ายกระแสไฟฟ้า หรือมีการรีเซต (Reset) เกิดขึ้นค่าภายในสแต็กพอยน์เตอร์จะมีค่า 07H ซึ่งเป็นตำแหน่งแอดเดรสภายในบริเวณพื้นที่ 128 ไบต์แรกของหน่วยความจำข้อมูลภายใน การใช้งานภายในโปรแกรมจะเรียกว่า รีจิสเตอร์ SP

ตัวชี้ข้อมูล หรือ ดาต้าพอยน์เตอร์ (Data Pointer)

เป็นรีจิสเตอร์ขนาด 16 บิต ซึ่งเรียกว่า รีจิสเตอร์ DPTR และสามารถใช้งานแยกออกเป็นรีจิสเตอร์ขนาด 8 บิตสองตัว คือ รีจิสเตอร์ DPH และ DPL เพื่อเก็บค่าแอดเดรสของหน่วยความจำที่จะต้องใช้งานภายในโปรแกรม หรืออาจจะเป็นแอดเดรสของอุปกรณ์ภายนอกซึ่งกำหนดให้ติดต่อกันโดยใช้ตำแหน่งของหน่วยความจำนั้นภายในโปรแกรม

โปรแกรมสเตตัสเวิร์ด (PSW)

รีจิสเตอร์นี้ทำหน้าที่บอกถึงแฟล็กสภาวะการทำงานต่างๆ รวมทั้งบิตสำหรับการกำหนดเลือกแบงก์(Bank) ของรีจิสเตอร์ที่ใช้งานด้วย

รีจิสเตอร์ที่เกี่ยวข้องกับพอร์ต(Port Register)

รีจิสเตอร์เหล่านี้มีความเกี่ยวข้องกับการทำงานของพอร์ตอินพุต/เอาร์พุตโดยตรง ซึ่งแต่ละตัวจะเป็นรีจิสเตอร์ขนาด 8 บิต สามารถใช้งานได้ทั้งในลักษณะการอินพุต หรือการเอาต์พุตข้อมูลได้ การดำเนินการใดๆ ที่เกี่ยวข้องกับพอร์ตทั้งสี่นี้จะมีผลทำให้ข้อมูลที่ตำแหน่งของพอร์ตเหล่านี้เปลี่ยนแปลงไปเช่นกัน นอกจากนี้พอร์ต P0 และ P2 ยังสามารถนำมาใช้ในการติดต่อกับหน่วยความจำโปรแกรมหรือหน่วยความจำข้อมูลภายนอกได้ โดยพอร์ต P2 จะเป็นค่าของแอดเดรส 8 บิตบนของหน่วยความจำ ช่วงเวลาต่อมาจึงจะนำพอร์ต P0 ไปใช้เป็นบัสสำหรับการรับหรือส่งข้อมูลกับหน่วยอุปกรณ์ภายนอก สำหรับพอร์ต P3 นั้นนอกเหนือจะใช้ในลักษณะของพอร์ตอินพุต/เอาต์พุตเช่นปกติแล้ว ยังนำมาใช้ในฐานะบัสควบคุมเกี่ยวกับสัญญาณอินเตอร์รัปต์ได้อีกด้วย

รีจิสเตอร์ SBUF

เป็นบัฟเฟอร์ขนาด 8 บิต สำหรับการสื่อสารข้อมูลแบบอนุกรมทั้งการรับและส่งข้อมูล ซึ่งตามความเป็นจริงแล้วบัฟเฟอร์นี้มีอยู่ด้วยกันสองชุดและแยกออกจากกันอย่างชัดเจน สำหับการส่งและการรับ โดยซีพียูจะทำการเลือกบัฟเฟอร์ที่เหมาะสมให้โดยอัตโนมัติ

รีจิสเตอร์ PCON

เป็นรีจิสเตอร์ที่ใช้ในการควบคุมหน้าที่การทำงานในสามลักษณะ ซึ่งได้แก่ การควบคุมการทำงานของโปรเซสเซอร์ (บิต IDL และ PD) การกำหนดอัตราการทวีคูณของอัตราเร็วในการสื่อสารข้อมูลอนุกรม(บิต SMOD) และแฟล็กสภาวะสำหรับการใช้งานทั่วไป (บิต GR0 และ GR1)

บิต PD(Power down)

เป็นการกำหนดให้ลดกำลังไฟฟ้าที่จ่ายให้กับส่วนของโปรเซสเซอร์ภายในลง โดยยังคงมีกำลังไฟฟ้าจ่ายให้กับหน่วยความจำข้อมูลภายในผ่านทางขาสัญญาณ RST วิธีการนี้มักนำมาใช้

ในกรณีที่มีการตรวจสอบการไม่มีกำลังไฟฟ้า (Power failure) โดยวงจรตรวจสอบภายนอกจะต้องมีการอิเตอร์รัปต์เข้ามา เพื่อทำการเก็บข้อมูลที่กำลังประมวลผลอยู่ก่อนและเมื่อมีกระแสไฟฟ้านจ่ายให้เป็นปกติแล้ว จึงค่อยนำข้อมูลนั้นมาประมวลผลต่อไป

บิต ILD(Idle Mode)

เป็นการกำหนดให้โปรเซสเซอร์หยุดทำงานชั่วขณะ (Sleep) และจะกลับมาอยู่ในสภาพปกติอีกครั้งเมื่อทำการรีเซตทางฮาร์แวร์ หรือมีการอินเตอร์รัปต์อย่างใดอย่างหนึ่งเกิดขึ้น การทำงานในลักษณะนี้สามารถเกิดขึ้นได้ก็เนื่องจากว่าสภาวะการหยุดการทำงานชั่วขณะนั้น เป็นเพียงการห้ามไม่ให้มีสัญญาณนาฬิกาจ่ายให้ส่วนของโปรเซอร์เท่านั้น ส่วนของวงจรการอินเตอร์รัปต์พอร์ตอนุกรมและวงจรนับ/จับเวลา ยังคงมีสัญญาณนาฬิกาอยู่เป็นปกติ

รีจิสเตอร์ IP,IE,TMOD,SMOD,SCON

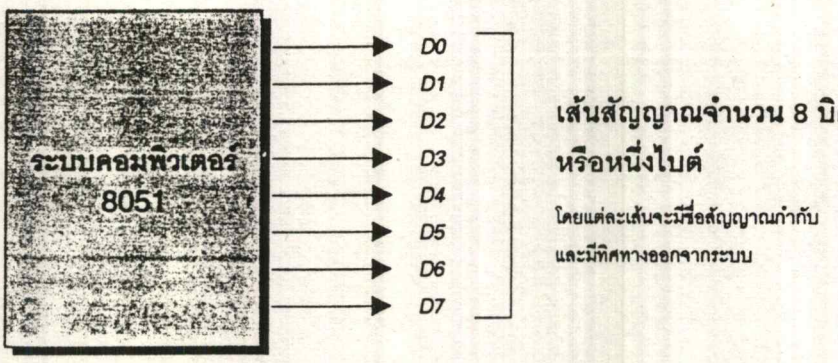
เป็นกลุ่มของรีจิสเตอร์ที่ทำหน้าที่กำหนดการควบคุม และการทำงานของการอินเตอร์รัปต์ต่างๆ ของ 8051

2.3.5 หน่วยความจำข้อมูลภายนอก

การใช้หน่วยความจำข้อมูลภายนอกเป็นวิธีการแก้ปัญหาอย่างหนึ่ง ในกรณีที่มีความต้องการหน่วยความจำสำหรับการเก็บข้อมูลชั่วคราว หรือหรือตัวแปรของโปรแกรมมากเกินกว่าขนาดความจำของหน่วยความจำข้อมูลภายใน ซึ่งมีขนาดเพียง 128 หรือ 256 ไบต์เท่านั้น บางครั้งการใช้หน่วยความจำข้อมูลภายนอกยังเหมาะกับงานประยุกต์บางอย่างที่จำเป็น ต้องมีการเก็บสำรองข้อมูลบางอย่างไว้ไม่ให้สูญหายแม้ว่าจะไม่มีการจ่ายไฟให้กับระบบ ก็สามารถทำได้โดยการใช้ไอซีหน่วยความจำ RAM พร้อมแบตเตอรี่สำรองประเภทลิเทียมหรือนิเกิล-แคดเมียมเป็นตัวเก็บข้อมูลเหล่านี้ไว้แทน อย่างไรก็ตามไม่ว่าสาเหตุการนำไอซีหน่วยความจำภายนอกมาใช้จะเป็นอะไรจะมีผลทำให้พอร์พอินพุต/เอาต์พุตข้อมูลของ 8051 ถูกนำไปใช้เพื่อการติดต่อกับหน่วยความจำเหล่านี้แทน ดังนั้นจึงอาจจำเป็นต้องมีการใช้วงจรประกอบอื่นๆ เพื่อชดเชยความสามารถเหล่านั้นของ 8051 แทน

2.4 พอร์ตอินพุต/เอาต์พุตของ 8051

พอร์ต มีความหมายถึงแอดเดรสหนึ่งที่ได้รับการกำหนดไว้เพื่อการโอนย้ายข้อมูลระหว่างไมโครคอนโทรลเลอร์กับอุปกรณ์ภายนอก การกำหนดประเภทของการติดต่อขึ้นอยู่กับทิศทางการไหลของข้อมูลเมื่อพิจารณาจากไมโครคอนโทรลเลอร์เป็นหลัก (รูปที่ 2.7) ดังนั้นการนำเข้าข้อมูลจากวงจรภายนอกจึงเรียกว่า การอินพุต(Input) และในกรณีตรงกันข้ามเพื่อส่งออกข้อมูลก็จะเรียกว่า การเอ้าต์พุต(Output)



รูปที่ 2.7 แสดงการส่งข้อมูลผ่านทางพอร์ตเอาต์พุต

เมื่อพิจารณาถึงการส่งข้อมูลภายในพอร์ตจะสามารถแยกประเภทของพอร์ตออกได้เป็นสองลักษณะคือ พอร์ตแบบขนาน(Parallel port) ซึ่งทำการส่งบิต ข้ออมูลทั้งหมดออกมาหรือนำเข้าไปพร้อมกันทีเดียว และ พอร์ตแบบอนุกรม(Serial Port) ซึ่งทำการโอนย้ายข้อมูลคราวละบิตๆ จนครบจำนวน

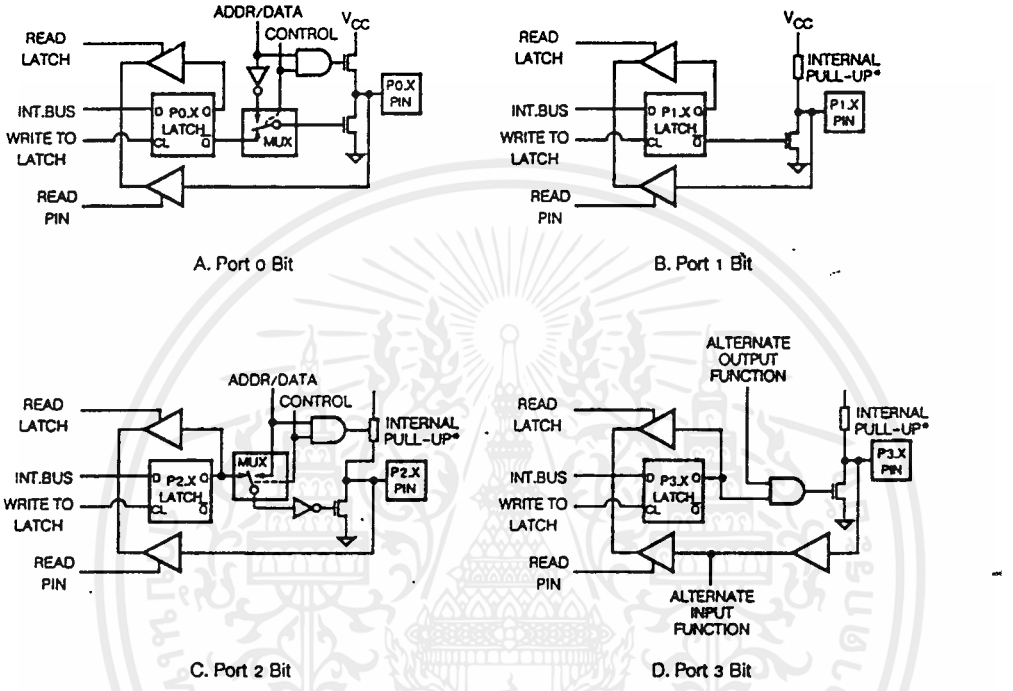
2.4.1 พอร์ตแบบขนานของ 8051

8051 มีโครงสร้างของพอร์ตเรียกชื่อเรียงตามลำดับว่า พอร์ต 0,1,2 และ 3 และเป็นพอร์ตขนาด 8 บิตทั้งหมด การใช้งานพอร์ตสามารถทำได้ทั้งในลักษณะของสัญญาณเดี่ยวๆหรือกลุ่มของสัญญาณได้ นอกจากนี้พอร์ต 0,2 และ 3 ยังสามารถนำไปใช้งานอื่นๆที่ไม่ใช่เป็นพอร์ตอินพุต/เอาต์พุตได้ โดยพอร์ต 0 จะทำหน้าที่มัลติเพล็กซ์ ระหว่างบัสแอดเดรสไบต์ต่ำและบัสข้อมูลสำหรับการติดต่อกับวงจรประกอบร่วมกับข้อมูลบัสแอดเดรสไบต์สูงซึ่งจะส่งออกมาทางพอร์ต 2 สำหรับพอร์ต 3 นั้น นอกเหนือไปจากความสามารถเช่นปกติแล้ว สามารถนำไปเป็นขาสัญญาณของการอินเตอร์รัปต์ต่างๆ ซึ่งรวมทั้งการสร้างสัญญาณควบคุม RD\ และ WR\ เพื่อทำหน้าที่อ่านหรือเขียนหน่วยความจำข้อมูลภายนอกด้วย การใช้งานพอร์ตลักษณะอื่นๆ ที่ไม่ใช่เป็นพอร์ตแบบอินพุต/เอาต์พุตนี้จะดำเนินการโดย 8051 เองโดยอัตโนมัติ

2.4.2 โครงสร้างการทำงานของพอร์ต 8051

จากลักษณะโครงสร้างของแต่ละบิตภายในพอร์ตทั้งหมดของ 8051 ซึ่งแสดงได้ในรูปที่ 2.8 นั้น จะเห็นว่ามีความคล้ายคลึงกันตามลักษณะโครงสร้างที่เรียกว่า Quasi-bidirectional port ยกเว้นพอร์ต 0 ซึ่งเพียงแต่ไม่มีตัวต้านทานทำหน้าที่ Pull-up สัญญาณไว้ภายในเท่านั้น วงจรประกอบอื่นภายในยังมีฟลิปฟลอปแบบ D ซึ่งมีผลทำให้พอร์ตสามารถแลตซ์หรือค้างสภาวะของสัญญาณได้ นอกจากนี้ในส่วนเอาต์พุตของฟลิปฟลอปเฉพาะของพอร์ต 0 และ

พอร์ต 2 จะมีโครงสร้างที่ทำหน้าที่คล้ายคลึงกับสวิตช์เพิ่มเติมขึ้น เพื่อควบคุมให้เอาต์พุตนี้ต่อเข้ากับส่วนของทรานซิสเตอร์ในระหว่างที่ไม่มีการทำงานในลักษณะของบัสแอดเดรสหรือบัสข้อมูลด้วย สำหรับบัฟเฟอร์จำนวนสองตัวของทุกบิตในพอร์ตนั้นมีการทำงานแยกกันอย่างอิสระ โดยตัวที่อยู่ทางด้านบนจะยอมให้สัญญาณผ่านได้ก็ต่อเมื่อได้มีการอ่านสถานะของขาสัญญาณเท่านั้น



A. Port 0 Bit B. Port 1 Bit

C. Port 2 Bit D. Port 3 Bit

2.4.3 การใช้งานพอร์ตเป็นการอินพุต

การใช้งานพอร์ตเป็นการอินพุตข้อมูลจะต้องเริ่มด้วยการส่งข้อมูลที่มีค่าเป็น 1 ออกมาทางบิตของพอร์ตนั้นก่อนเป็นลำดับแรก เพื่อหยุดการทำงานของทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณเอาต์พุตของบิตนั้น ทำให้ขาสัญญาณของบิตถูกต่อเข้ากับตัวต้านทานซึ่งทำหน้าที่ Pull-up ภายในซึ่งมีผลให้บิตนั้นๆ ของพอร์ต 1,2 และ 3 เป็นสภาวะของลอจิกสูง ตัวต้านทานซึ่งมีค่าประมาณ 50 KΩ ซึ่งเป็นค่าที่สูงมาก และทำให้อุปกรณ์ภายนอกสามารถขับสัญญาณของพอร์ตเหล่านี้เป็นลอจิกต่ำได้ง่าย สำหรับบิตของพอร์ต 0 นั้น แม้ว่าจะมีหลักการทำงานที่คล้ายคลึงกันกับบิตของพอร์ตอื่นๆ แต่เนื่องจากการที่ไม่มีตัวต้านทานทำหน้าที่ Pull-up ภายในไว้ ทำให้เมื่อทรานซิตเตอร์ที่ทำหน้าที่ขับสัญญาณเอรต์พุตนั้นหยุดการทำงาน ก็จะเป็นผลให้ขาสัญญาณนี้อยู่ในสภาวะอิมพีแดนซ์สูงแทน

2.4.4 การใช้งานพอร์ตเป็นการเอาต์พุต

เมื่อมีการส่งข้อมูลที่มีค่าเป็น 0 ให้กับแต่ละบิตของพอร์ตทุกพอร์ต ข้อมูลนี้จะถูกส่งให้กับฟลิปฟลอปซึ่งจะค้างค่านี้ไว้ และมีผลทำให้ทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณเอาต์พุตนั้นทำงานดังนั้นขาสัญญาณก็จะมีสภาวะลอจิกเป็นลอจิกต่ำด้วย

ส่วนการส่งข้อมูลที่มีค่าเป็นหนึ่งออกมานั้น ในกรณีที่มีการทำงานในแต่ละบิตของพอร์ต 1,2 และ 3 จะทำให้ทรานซิสเตอร์ที่ทำหน้าที่ขับสัญญาณเอาต์พุตนั้นหยุดการทำงาน มีผลทำให้ขาของสัญญาณเป็นลอจิกสูงด้วยตัวต้านทานที่ Pull-up อยู่ภายในนั้น แต่สำหรับการทำงานในแต่ละบิตทางพอร์ต 0 เป็นการเอาต์พุตข้อมูล จึงจำเป็นต้องใช้ตัวต้านทานภายนอก Pull-up สัญญาณไว้กับลอจิกสูงแทน

ความสามารถอีกประการหนึ่งเกี่ยวกับพอร์ตอินพุต/เอาต์พุตของ 8051 เป็นวิธีอ่านค่าลอจิกจากพอร์ตซึ่งมีได้สองวิธีคือ การอ่านค่าลอจิกที่ขาสัญญาณ (Port pin) แลการอ่านค่าลอจิกของการแลตซ์ที่พอร์ต (Port latch) ดังจะสังเกตุได้ดังรูปที่ 2.8 วิธีการอ่านค่าจากพอร์ตทั้งสองแบบนี้จะช่วยให้ระบบทำงานได้ด้วยความถูกต้องมากยิ่งขึ้น

2.5 การอินเตอร์รัปต์ของการสื่อสารอนุกรม

เนื่องจากการส่งหรือรับข้อมูลอนุกรมในการส่งข้อมูลไบต์หนึ่งๆ ค่อนข้างจะใช้เวลานานหลายมิลลิวินาที ดังนั้นเพื่อให้การจัดการเกี่ยวกับการสื่อสารแบบนี้เป็นไปอย่างมีประสิทธิภาพ 8051 จึงได้กำหนดให้บิตหรือแฟล็กสถานะที่เกี่ยวข้องทั้งหมด จัดรวมอยู่ภายในรีจิสเตอร์ SCON เท่านั้น เช่น แฟล็ก T1 ซึ่งจะมีค่าเป็นหนึ่งเมื่อข้อมูลได้ส่งออกไปภายนอกเสร็จสิ้นแล้ว และแฟล็ก RI ซึ่งจะมีค่าเป็น 1 เพื่อแจ้งให้ทราบว่าได้รับข้อมูลผ่านมาทางพอร์ตอนุกรม เมื่อแฟล็กตัวใดตัวหนึ่งมีค่าเป็น 1 จะมีผลทำให้เกิดการอินเตอร์รัปต์ขึ้น ดังนั้นภายในโปรแกรมจะต้องทำการตรวจสอบสภาวะของแฟล็กเหล่านี้เองว่ามีการอินเตอร์รัปต์เกิดขึ้นเพราะสาเหตุใดจากนั้นจึงค่อยทำการกำหนดค่า 0 ให้กับแฟล็กนั้น ลักษณะดังกล่าวนี้จะมีความแตกต่างไปจากการอินเตอร์รัปต์จากสัญญาณอื่นๆ เช่น วงจรนับ/จับเวลา เป็นต้น ซึ่งจะมีการกำหนดค่า 0 ให้กับแฟล็กสถานะที่เกี่ยวข้องโดยอัตโนมัติ ภายหลักจากที่ได้เข้าไปทำงานยังส่วนโปรแกรมย่อยบริการอินเตอร์รัปต์

2.5.1 กระบวนการรับและส่งข้อมูลอนุกรมของ 8051

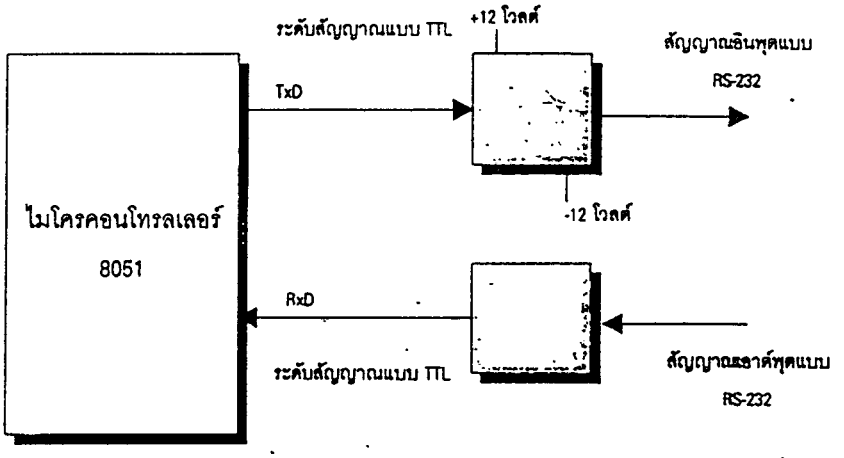
การส่งข้อมูลออกทางพอร์ตอนุกรมของ 8051 จะเริ่มต้นขึ้น ภายหลังเมื่อมีการเขียนข้อมูลลงในรีจิสเตอร์ SBUF ข้อมูลนี้จะถูกจัดการด้วยวิธีการทางด้านฮาร์ดแวร์ในการเลื่อนบิตและส่งสัญญาณออกไปภายนอกโดยอัตโนมัติ เมื่อข้อมูลเหล่านี้ได้ส่งออกครบถ้วนแล้ว จึงทำการ

กำหนดค่าของแฟล็ก TI ให้เป็น 1 เพื่อแจ้งให้ทราบว่าขณะนี้รีจิสเตอร์ SUBF ว่างและพร้อมที่จะส่งข้อมูลไบต์ต่อไปแล้ว ในกรณีที่ผู้ใช้เขียนข้อมูลใหม่ลงในรีจิสเตอร์ SBUF โดยไม่รอให้แฟล็ก TI มีค่าเป็น 1 ก่อนจะมีผลทำให้ข้อมูลที่ส่งออกไปผิดพลาดได้

สำหรับการรับข้อมูลจากพอร์ตอนุกรมจะเริ่มต้นโดยการกำหนดค่าบิต REN(Reciever Enable) ให้มีค่าเป็น 1 ก่อน หลังจากนั้นเมื่อมีบิตของข้อมูลถูกส่งเข้ามาจากภายนอกระบบฮาร์ดแวร์ของ 8051 จึงจะทำการเลื่อนบิตเหล่านี้เข้ามาโดยอัตโนมัติ และเมื่อบิตสุดท้ายถูกเลื่อนเข้ามาเรียบร้อยแล้ว ข้อมูลนั้นจะถูกย้ายเข้ามาเก็บไว้ยังรีจิสเตอร์ SBUF และทำการกำหนดให้แฟล็ก RI ให้มีค่าเป็น 1 ซึ่งมีผลทำให้เกิดการอินเตอร์รัปต์โปรแกรมขึ้น

2.5.2 การเชื่อมต่อแบบมาตรฐาน RS-232C

ในการเชื่อมต่อแบบอนุกรมเข้ากับอุปกรณ์คอมพิวเตอร์ต่างๆ เช่น คอมพิวเตอร์ เทเล็กซ์ หรือ โทรพิมพ์ เป็นต้น มักกำหนดใช้การเชื่อมต่อตามมาตรฐาน RS-232C ทั้งนี้เพื่อให้มีการใช้งานเส้นสัญญาณหรือรูปแบบของตัวเชื่อมต่อที่สอดคล้องกัน จะได้ลดปัญหาการเข้ากันไม่ได้ระหว่างสัญญาณของอุปกรณ์ที่มาเชื่อมต่อกันทั้งสองด้านให้น้อยลง เนื่องจากระดับโวลเตจที่ใช้และการแทนความหมายของระดับลอจิกตามมาตรฐานนี้แตดต่างไปจากที่ใช้งานกันในระบบดิจิตอลทั่วไปโดยระดับสัญญาณของ RS-232 เป็นแบบไบโพลาร์(Bipolar) ระดับโวลเตจทางด้านลบช่วง -3V ถึง -20V แทนค่าลอจิก 1 และโวลเตจทางด้านบวกช่วง +3V ถึง +20V แทนค่าลอจิก 0 ดังนั้นจะเห็นได้ว่ามีความจำเป็นต้องเพิ่มเติมอุปกรณ์หรือวงจรพิเศษเข้าไป เพื่อเปลี่ยนระดับโวลเตจจากระบบ 0V ถึง +5V จากขาสัญญาณของ 8051 เป็นระดับโวลเตจที่สูงกว่าค่า +3.0V หรือต่ำกว่า -3.0V ซึ่งแสดงให้เห็นว่าระดับสัญญาณแบบ TTL จากขาสัญญาณ TxD และ RxD ของ 8051 จะต้องถูกปรับเปลี่ยนไปเป็นระดับสัญญาณ RS-232 ก่อน ที่จะทำการส่งออกไปในสายนำสัญญาณต่อไป



รูปที่ 2.15 แสดงให้เห็นถึงแนวการเปลี่ยนแปลงระดับสัญญาณ TTL จาก MCS-51 ให้เป็นระดับสัญญาณแบบ RS-232 และการเปลี่ยนระดับสัญญาณอินพุตแบบ RS-232 ไปเป็นระดับสัญญาณ TTL ก่อนที่จะได้เชื่อมต่อเข้ากับขาสัญญาณของ MCS-51

2.6 สเต็ปปิ้งมอเตอร์ (STEPPING MOTER)

สเต็ปปิ้งมอเตอร์เป็นมอเตอร์ชนิดหนึ่งที่ทำหน้าที่เปลี่ยนสัญญาณดิจิตอลทางไฟฟ้าไปเป็นการเคลื่อนที่ทางกล ดังนั้นการติดต่อกับอุปกรณ์ดิจิตอลเป็นไปโดยง่าย และวงจรขยายกำลังจากสัญญาณดิจิตอล (DIGITAL POWER AMPLIFIER) ที่ใช้ก็มีราคาถูกกว่าวงจรขยายกำลังเชิงเส้นอีกด้วย อีกทั้งการออกแบบวงจรควบคุมสเต็ปปิ้งมอเตอร์สามารถทำได้ง่ายกว่าวงจรควบคุมมอเตอร์แบบเซอร์โว และยังสามารถออกแบบวงจรสเต็ปปิ้งมอเตอร์หยุดการทำงานได้อย่างทันทีทันใดอีกด้วย

ชนิดของสเต็ปปิ้งมอเตอร์

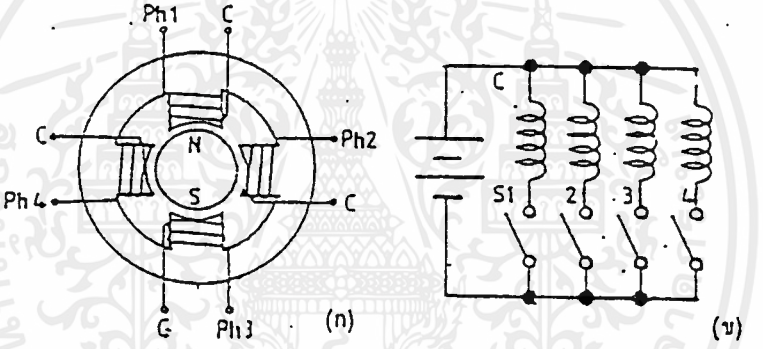
สเต็ปปิ้งมอเตอร์แบ่งตามมาตรฐานได้ ๓ แบบ คือ

1. วาริเอเบิลรีลักแตนซ์ (VARIABLE RELUCTANCE : VR)
2. เพอร์มาเนนต์แมกเน็ต (PERMANENT MAGNET : PM)
3. ไฮบริด (HYBRID)

ชนิดวาริเอเบิลรีลักแตนซ์มีโครงสร้างของโรเตอร์แบบมัลติทูธ (MULTI TOOTH) ทำจากเหล็กอ่อนเราทราบได้ว่าเป็นมอเตอร์ชนิดนี้โดยการทดสอบได้ง่ายมากคือ ใช้นิ้วหมุนเพลาของมอเตอร์และสังเกตมอเตอร์ชนิดนี้ที่โรเตอร์ไม่เกิดปรากฏการณ์ทางแม่เหล็ก MAGMATISM

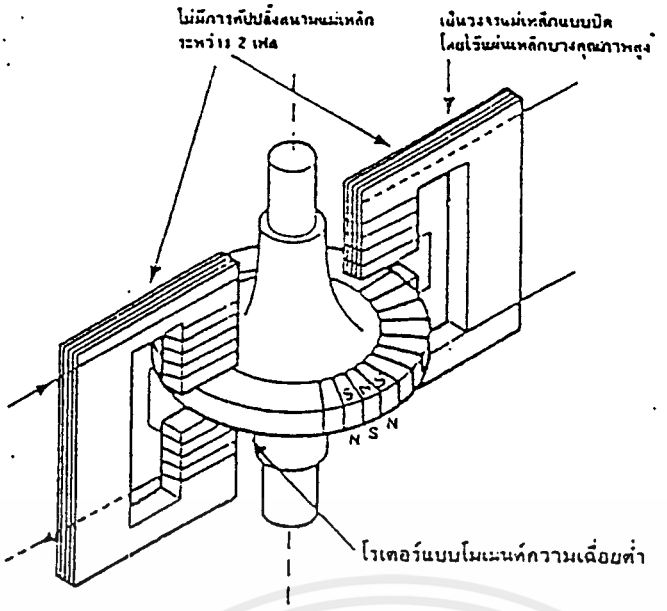
มันจึงหมุนได้ตลอดโดยไม่ติดขัดแตกต่างจากชนิด PM และไฮบริดซึ่งมีสนามแม่เหล็กที่โรเตอร์เมื่อหมุนจะรู้สึกขัด ๆ เหมือนเป็นฟันเฟือง สเต็ปในการหมุนสูง

ชนิดเพอร์มาเนนต์แมกเน็ตมีโครงสร้างแบบเรียบไม่มีขั้วแม่เหล็กและบนโรเตอร์จะเป็นแบบแม่เหล็กถาวร การควบคุมทำได้โดยการป้อนกระแสกระตุ้นที่ขดลวดบนสเตเตอร์ แบบ ๔ เฟส จะมีขั้วแม่เหล็กอยู่ ๔ ขั้ว ซึ่งมีคอยล์พันอยู่แยกจากกัน ขั้วแม่เหล็กถาวรจะถูกแรงดึงดูดจากขั้วแม่เหล็กบนสเตเตอร์เมื่อป้อนกระแสไฟฟ้าเข้าสู่ขดลวด และโรเตอร์จะอยู่คงที่ที่ขั้วแม่เหล็กบนสเตเตอร์นั้นถึงแม้ว่าจะไม่ป้อนกระแสไฟฟ้าอีกต่อไป ทำให้เกิดแรงยึดหน่วงขึ้น สเต็ปปิ้งมอเตอร์ชนิดนี้มีข้อดีในเรื่องของความถูกต้องของตำแหน่ง และความเร็วมากขึ้นเมื่อเปรียบเทียบกับชนิดอื่น



รูปที่ 2.16 ก) ภาพหน้าตัดของ PM มอเตอร์แบบ ๔ เฟส
ข) วงจรกระตุ้นเฟสพื้นฐานสำหรับ PM มอเตอร์ ๔ เฟส

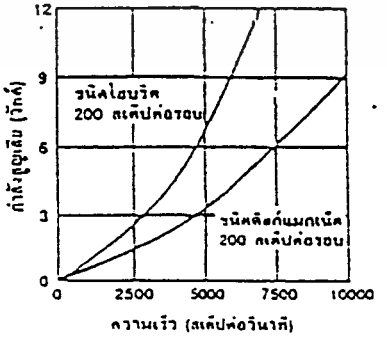
ชนิดไฮบริดเป็นชนิดที่นิยมใช้งานมากที่สุด โดยเฉพาะนำมาใช้งานอย่างมากในอุปกรณ์ที่ใช้ในเครื่องคอมพิวเตอร์ ชนิดมีโครงสร้างภายในซึ่งได้จาก การรวมเอาโครงสร้างของสเตเตอร์ชนิดวาริเอเบิลรีลักแตนซ์ และโครงสร้างของโรเตอร์จากเพอร์มาเนนต์แมกเน็ตมาประกอบเข้าด้วยกันจึงทำให้เป็นมอเตอร์ที่มีแรงยึดหน่วงสูง มีแรงบิดดึงและผลักได้ดีซึ่งมีความคงที่และทำงานได้ดี ถึงแม้ว่าจะมีสเต็ปต่อรอบในการหมุนสูง



รูปที่ 2.17 แสดงโครงสร้างพื้นฐานของชนิดแรเอิร์ธเพอร์มาเนนต์แมกเน็ต

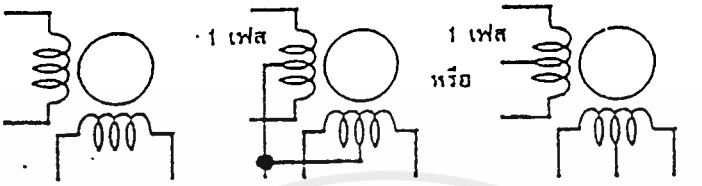
สเต็ปปิ้งมอเตอร์แบบใหม่อีกชนิดหนึ่งที่กล่าวถึงอีกเล็กน้อยคือ ชนิดที่ปรับปรุงมาจากชนิดเพอมาเนนต์แมกเน็ตนั่นคือ ชนิดแรเอิร์ธเพอร์มาเนนต์แม็กเน็ตดังแสดงโครงสร้างในรูปที่ 2.17 หรือที่เรียกกันว่าชนิดดิสค์แมกเน็ตสเต็ปปิ้งมอเตอร์

โครงสร้างของโรเตอร์ของมอเตอร์ชนิดนี้มีลักษณะเป็นแผ่นซึ่งยึดติดกับเพลาของมอเตอร์ การทำงานของมอเตอร์ยังคงเป็นเช่นเดิม แต่ด้วยโครงสร้างแบบใหม่นี้จะทำให้เกิดโมเมนต์ของความเฉื่อยต่ำมาก, มีอัตราเร่งสูง มอเตอร์ชนิดนี้จึงจัดเป็นอีกชนิดหนึ่ง และมันก็มีประสิทธิภาพสูงอีกหลายอย่างเช่น แรงบิดดึง; กำลังทางกลที่ได้ของมอเตอร์, ความถูกต้องของตำแหน่งสูงมาก และความเร็วในการเริ่มหมุนและหยุดสูง อีกทั้งยังมีความสูญเสียของกำลังต่ำ ดังแสดงในรูปที่ 2.18



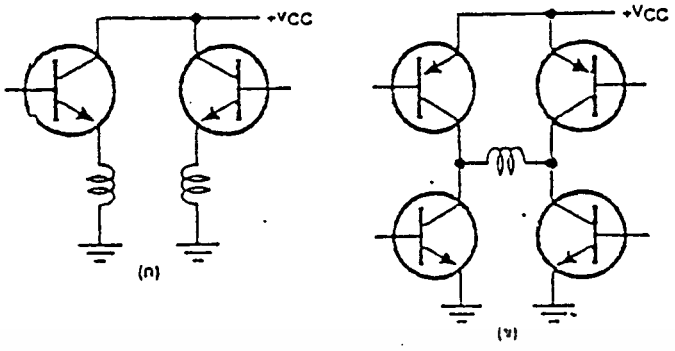
รูปที่ 2.18 แสดงกราฟข้อมูลการสูญเสียกำลังงานไฟฟ้าและความเร็วในการหมุนโดยเปรียบเทียบระหว่างสเต็ปปิ้งมอเตอร์ชนิดไฮบริดและชนิดแรเอิร์ธเพอร์มาเนนต์แมกเน็ต

การพันขดลวดหรือคอยล์บนสเต็ปปิ้งมอเตอร์มีอยู่ ๒ วิธีคือ แบบไบโพลาร์ (BIPOLAR) และแบบยูนิโพลาร์ (UNIPOLAR) ดังแสดงในรูปที่ 2.19



รูปที่ 2.19 แสดงการพันขดลวดของสเตเตอร์ด้านซ้ายเป็นแบบไบโพลาร์และที่เหลือเป็นแบบยูนิโพลาร์

สเต็ปปิ้งมอเตอร์แบบไบโพลาร์มีการพันขดลวด ๑ ขดแต่ละขั้วแม่เหล็กของสเตเตอร์ ขั้วแม่เหล็กที่เกิดขึ้นบนสเตเตอร์ถูกกำหนดโดยทิศทางของกระแสไฟฟ้า และสามารถทำให้เกิดขั้วแม่เหล็กในทิศทางตรงข้ามได้โดยการกลับทิศทางการไหลของกระแสไฟฟ้า ซึ่งการกำหนดทิศทางไหลและกลับทิศทางของกระแสไฟฟ้าทำได้โดยการใช้วงจรสวิตช์ซึ่งกลับขั้วไฟฟ้าสำหรับยูนิโพลาร์จะมีการพันขดลวด ๒ ขดบนแต่ละขั้วแม่เหล็กของสเตเตอร์ ซึ่งแต่ละขดจะทำให้เกิดขั้วแม่เหล็กในทิศทางตรงข้ามกัน การกลับขั้วแม่เหล็กเปลี่ยนไปมาทำได้โดยการสวิตช์กระแสไฟฟ้าจากขดลวดขดหนึ่งไปยังอีกขดลวดหนึ่งแทนเท่านั้น โดยปกติขดลวดทั้งสองจะมีการเชื่อมต่อกันหรือมีจุดร่วมเพื่อลดจำนวนของสายไฟที่ต่อกับมอเตอร์ วงจรกำลังไฟฟ้าของมอเตอร์แบบยูนิโพลาร์ทำได้ง่ายกว่าไบโพลาร์ เพราะมันต้องการเพียงสวิตช์ธรรมดาในการเปิดปิดกำลังไฟฟ้าขดลวดบนสเตเตอร์ในทิศทางที่ต้องการให้หมุนได้ทันที รูปที่ 2.21 แสดงวงจรการจ่ายไฟฟ้าซึ่งทรานซิสเตอร์ทำหน้าที่เป็นสวิตช์ให้กับสเต็ปปิ้งมอเตอร์ที่มีการพันขดลวดทั้งสองแบบ จะเห็นว่าแบบยูนิโพลาร์เป็นวงจรที่ง่ายและไม่ซับซ้อน

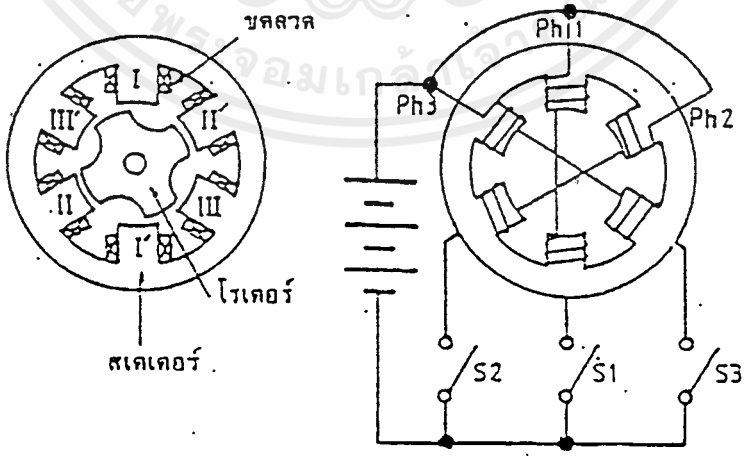


รูปที่ 2.20 แสดงการจ่ายไฟฟ้าให้กับสเต็ปปิ้งมอเตอร์

ก) แบบยูนิโพลาร์ซึ่งใช้ทรานซิสเตอร์เพียงตัวเดียวต่อ ๑ คอยล์

ข) แบบไบโพลาร์ ซึ่งต้องใช้ทรานซิสเตอร์ ๔ ตัวต่อ ๑ คอยล์

การทำงานของสเต็ปปิ้งมอเตอร์ แบบ VR จะเป็นพื้นฐานสำคัญในการทำงานของสเต็ปปิ้งมอเตอร์ ซึ่งจะช่วยให้เข้าใจการทำงานของสเต็ปปิ้งมอเตอร์ชนิดอื่น ๆ ได้ง่ายยิ่งขึ้น ดังในรูปจะเป็นภาพแสดงถึงการพันขดลวดแบบ VR มอเตอร์แบบ ๓ เฟส มีขั้วเหนือและขั้วใต้อยู่ตรงข้ามกัน ๓ คู่ โดยจะพันขดลวดแบบอนุกรมกันในแต่ละชุด ถ้ามีการกระตุ้นเฟสเกิดขึ้นขั้ว I' ,II, III' จะเป็นขั้วใต้ และ I, II, III จะเป็นขั้วเหนือ ทั้งโรเตอร์และสเตเตอร์จะทำจากเหล็กซิลิกอน ซึ่งเป็นวัสดุที่มีความซึมซับสูง สามารถให้เส้นแรงแม่เหล็กไหลผ่านได้มาก



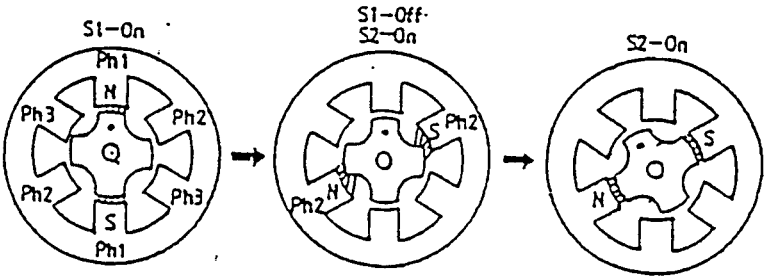
รูปที่ 2.21 ภาพหน้าตัดและการพันขดลวดของวาริเอเบิลรีลักแตนซ์สเต็ปมอเตอร์ ๓ เฟส

การทำงานจะมีการกระตุ้นที่เฟส I ก่อน (S "ON") ซึ่งจะทำให้เส้นแรงแม่เหล็กเกิดขึ้นดังรูป ตัวโรเตอร์จะพยายามวางตำแหน่งของตัวเองให้อยู่ในทิศทางที่ทำให้เกิดค่าความต้านทานแม่เหล็กน้อยที่สุด ในขณะที่เริ่มต้นที่กระตุ้นที่เฟส II (S1 "OFF" , S2 "ON") เส้นแรงจะไม่อยู่ในแนวทางเดินที่สะดวก จึงทำให้ค่าความต้านทานแม่เหล็กมีค่าสูง ตัวโรเตอร์ก็พยายามปรับตัวเองให้มีค่าความต้านทานแม่เหล็กน้อยที่สุดด้วยการหมุนในทิศทางทวนเข็มนาฬิกาซึ่งแรงบิดที่ใช้หมุนเกิดจากแรงของเส้นแรงแม่เหล็ก แล้วจะไปหยุดในตำแหน่งที่ความต้านทานแม่เหล็กน้อยที่สุด นั่นคือจะหมุนไป ๑ สเต็ป หรือ ๓๐ องศานั่นเอง ความสัมพันธ์ระหว่างจำนวนสเต็ปของการหมุนของโรเตอร์ไป ๑ รอบ (S) มุมที่เปลี่ยนไป ๑ สเต็ป จำนวนเฟสของสเตเตอร์ (m) และจำนวนฟันของโรเตอร์ (Nr) ดังแสดงไว้ในสมการที่ ๑

$$S = 360/\theta s = mNr$$

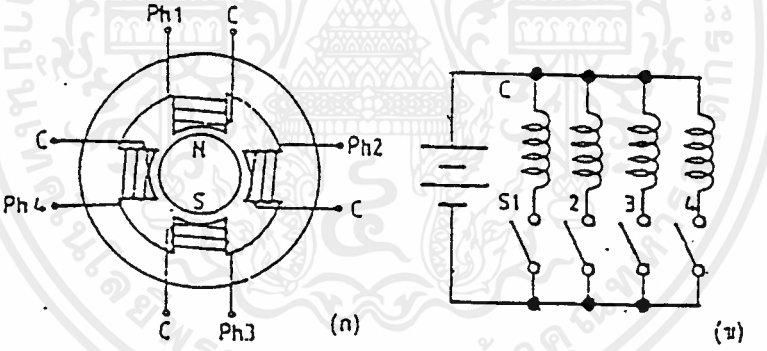
ตัวอย่างเช่น สเต็ปปิ้งมอเตอร์ตัวหนึ่งมี m = 3 , Nr = 4 ก็จะได้ S= 3 * 4 = 12 สเต็ป และมุมในการหมุน θs = 360. / 12 = 30 องศา ซึ่งจากสมการที่ ๑ ทำให้เราทราบอีกว่าถ้าจะลดค่าของ θs ให้น้อยลง อาจทำได้โดยการเพิ่มค่า m หรือค่า Nr ให้สูงขึ้น และลดช่องว่างระหว่างโรเตอร์กับสเตเตอร์ให้มีค่าน้อย ๆ เพื่อให้เกิดแรงบิดสูงสุดและยังมีผลต่อความเที่ยงตรงของตำแหน่งมากยิ่งขึ้นด้วย

รูปที่ 2.22 แสดงเส้นแรงแม่เหล็กและกระตุ้นเฟส ๑

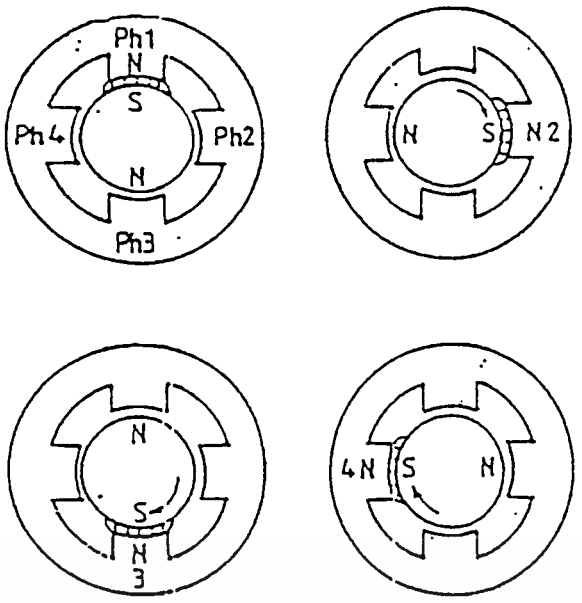


รูปที่ 2.23 แสดงขั้นตอนการหมุนของ VR สเต็ปปิ้งมอเตอร์เมื่อมีการกระตุ้นเฟส ๑ ไปยังเฟส ๒

สำหรับสเต็ปปิ้งมอเตอร์ชนิดเพอร์มาเนนต์แนกเน็ต หรือ PM จะมีข้อแตกต่างสำคัญจาก VR สเต็ปปิ้งมอเตอร์ คือ โรเตอร์จะเป็นแบบแม่เหล็กถาวรการพันขดลวดจึงจะต้องมีความแตกต่างกันออกไปซึ่งแสดงดังรูป



รูปที่ 2.24 ก) ภาพหน้าตัดของ PM มอเตอร์แบบ ๔ เฟส
ข) วงจรกระตุ้นเฟสพื้นฐานของ PM มอเตอร์ ๔ เฟส

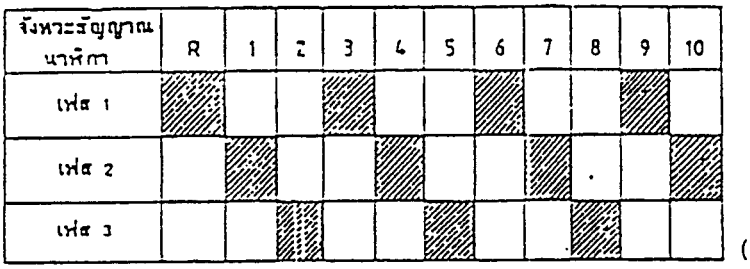


รูปที่ 2.25 ลำดับขั้นตอนการหมุน ๔ เฟส

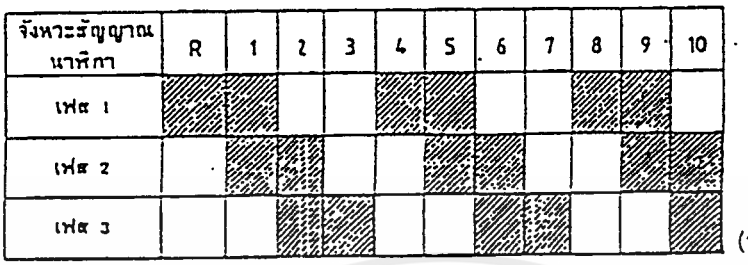
จากรูป ก) จะเห็นว่าสเตเตอร์ในแต่ละขั้วจะมีขดลวดพันอยู่ ซึ่งถือว่าแต่ละขั้วคือหนึ่ง ดังนั้นจากรูปจึงมีทั้งหมด ๔ เฟส ด้วยกัน สำหรับการต่อวงจรกระตุ้นเฟสอย่างง่ายแสดงไว้ในรูป ข) จะเห็นว่าปลายขดลวด (c) จะต่อร่วมกันถึงขั้วบวกของแหล่งจ่ายไฟดังนั้นเมื่อเกิดการกระตุ้นที่เฟสใดแล้วขั้วสเตเตอร์ที่เฟสนั้นจะกลายเป็นขั้วเหนือ และจากรูปจะเห็นว่าเป็นการแสดงตำแหน่งของโรเตอร์แต่ละสเต็ป หลังจากถูกกระตุ้นที่เฟส ๑ , ๒ , ๓ และ ๔ ตามลำดับ และจะหมุนไปในทิศทางตามเข็มนาฬิกา ทุก ๙๐ องศา ต่อสเต็ป ถ้าต้องการให้หมุนองศาต่อสเต็ปมีค่าลดลงหรือมีความละเอียดในตำแหน่งมากขึ้น จะต้องเพิ่มจำนวนเฟสของสเตเตอร์และจำนวนขั้วแม่เหล็กของโรเตอร์ให้มากขึ้น ข้อเสียของ PM มอเตอร์ คือ ราคาแพง และความหนาแน่นของเส้นแรงแม่เหล็กจะถูกจำกัดที่เส้นแรงแม่เหล็กภายใน ทำให้ไม่สามารถผลิตแรงบิดได้มาก

วิธีการกระตุ้นเฟส

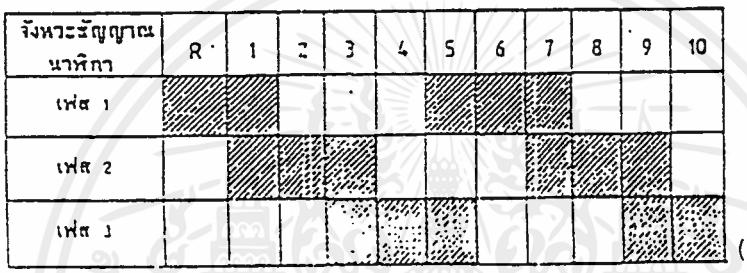
การที่จะทำให้สเต็ปปิ้งมอเตอร์หมุนได้อย่างต่อเนื่องเหมือนการหมุนของมอเตอร์ไฟฟ้ากระแสตรงนั้นต้องมีการจ่ายกระแสพัลซ์เป็นลำดับอย่างต่อเนื่อง วิธีการที่จะกระตุ้นแบบเฟสมีด้วยกันหลายวิธีแต่จะอธิบายวิธีที่ใช้กันมากเริ่มจากแบบแรก คือ การกระตุ้นแบบเฟสเดียว Single Phase Excitation เป็นการกระตุ้นเฟสเพียงเฟสเดียวเท่านั้นที่สัญญาณนาฬิกาหนึ่ง ๆ แบบที่ ๒ เป็นการกระตุ้น แบบเฟสคู่ Two Phase Excitation ก็จะมีการกระตุ้นเฟส ๒ เฟส พร้อมกันในจังหวะสัญญาณนาฬิกาหนึ่ง ๆ สำหรับแบบสุดท้ายเป็นการกระตุ้นแบบกึ่งสเต็ป Half Step Excitation จะเป็นการรวมเอาทั้งสองแบบเข้าด้วยกันโดยจะทำการกระตุ้นเฟสทั้งสองแบบสลับกันไป ซึ่งลักษณะของการกระตุ้นเฟสทั้งสามแบบสามารถดูได้จากรูป



(ก)



(ข)



(ค)

ในการกระตุ้นเฟสแบบเฟสคู่จะมีทิศทางของเส้นและแรงแม่เหล็กไม่เป็นเส้นตรงเหมือนกับการกระตุ้นแบบเฟสเดียว ดังแสดงในรูป ก) แต่ถึงกระนั้นค่าของมุมที่เปลี่ยนไปในหนึ่งสเต็ปก็ยังคงมีค่าเท่าเดิมเหมือนกับการกระตุ้นแบบเฟสเดียว สำหรับการกระตุ้นแบบกึ่งสเต็ปค่ามุมที่เปลี่ยนไปในหนึ่งสเต็ปจะมีค่าลดลงครึ่งหนึ่ง การกระตุ้นแบบเฟสคู่จะมีแรงบิดที่มากกว่าการกระตุ้นแบบเฟสเดียว ขณะเดียวกันยังสามารถเข้าถึงตำแหน่งได้รวดเร็วกว่าดังแสดงไว้ในรูป ข)

ข้อดีอีกอย่างหนึ่งของการกระตุ้นเฟสแบบกึ่งสเต็ปก็คือสามารถลดผลกระทบจากความถี่ในย่านรีโซแนนซ์ได้ แต่ที่ความถี่ต่ำ ๆ จะมีแรงบิดลดลง ดังแสดงไว้ในรูป ค) และสำหรับทิศทางการหมุนของมอเตอร์ขึ้นอยู่กับทิศทางของลำดับเฟสที่ถูกกระตุ้นด้วย

บทที่ 3
การออกแบบและกระบวนการสร้าง

ในส่วนของการออกแบบและกระบวนการสร้าง ชิ้นงานในการปฏิบัติงานครั้งนี้ จะแบ่งส่วนหลักๆได้เพียงสองส่วนใหญ่ๆคือ ทางด้าน

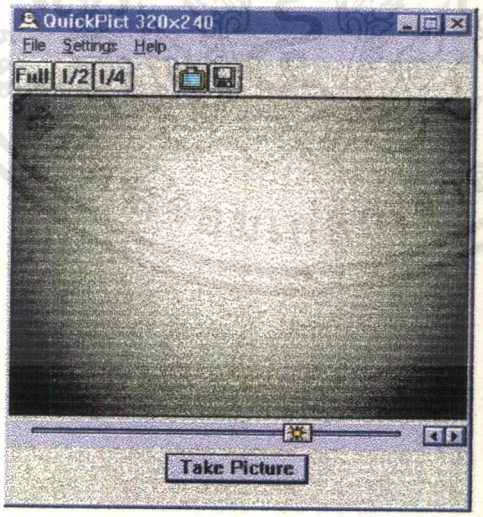
-Software (Delphi และ MCS51)

-Hardware

3.1 Software (Program Delphi)

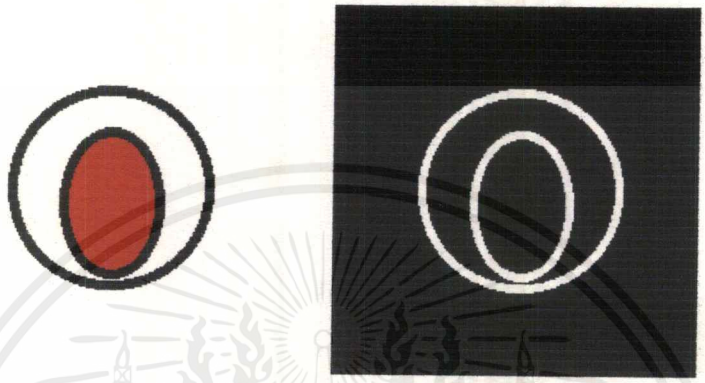
ในการทำงานชิ้นนี้เราได้เลือกเอา โปรแกรม เดลไฟล์ เวอร์ชั่น 3 มาใช้งานในการสร้างโปรแกรมควบคุมส่วนต่างๆ รายละเอียดของส่วนซอฟแวร์ สามารถแบ่งรายละเอียดได้ดังนี้

3.1.1 การรับภาพ (Load picture) การกระทำทางภาพนั้นเราจะจับภาพวัตถุที่ต้องการด้วยกล้อง และโปรแกรมสำเร็จรูป (Connectix VideoPhone ดังรูป) ที่จะใช้กับกล้องตัวนี้ โดยจะมีส่วนของการบันทึกภาพได้ และภาพที่ได้นี้จะได้เป็น Bitmap (*.bmp) และบันทึกไว้ในส่วนที่จะติดต่อได้



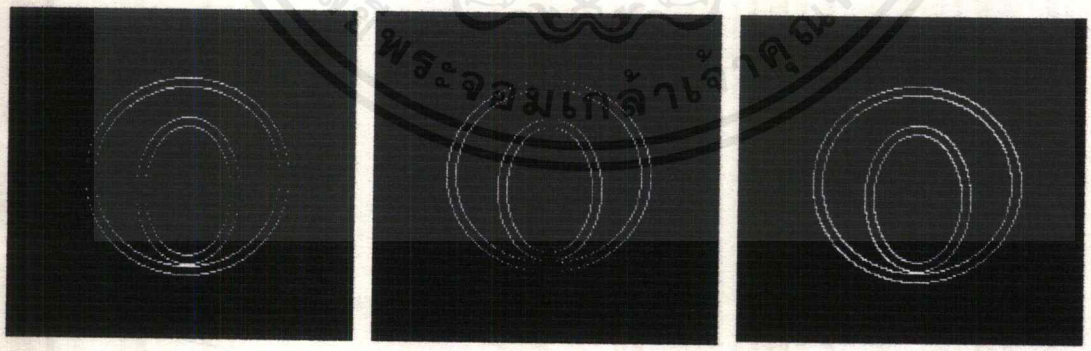
รูปที่ 3.1 โปรแกรมสำเร็จรูปในการรับภาพ

3.1.2 Color Threshold เป็นขั้นตอนของการคัดเลือกเอาสีที่ต้องการ ซึ่งอยู่ในภาพที่ได้บันทึกมาแล้วนั้น ให้มีความแตกต่างของสีกันเพียง 2 ระดับเท่านั้น คือ สีดำและขาว โดยโปรแกรมจะกระทำกับภาพไปทีละพิกเซล (pixels) หากพิกเซลใดมีสีดำ ก็จะทำการเปลี่ยนให้เป็นสีขาว และหากพิกเซลใดมีสีที่ไม่ใช่สีดำ ก็จะเปลี่ยนให้เป็นสีดำทั้งหมด ทั้งนี้เพื่อความสะดวกในขั้นตอนต่อไป



รูปที่ 3.2 แสดงขั้นตอน Color Threshold

3.1.3 Edge detection เป็นขั้นตอนๆหนึ่งที่สำคัญ โดยจะใช้การสแกน (scan) ไปตามพิกเซลทีละจุด ทีละบรรทัด โดยพิจารณาทีละบรรทัด หากพบขอบสีขาวในพิกเซลแรกของบรรทัดนั้น ก็จะทำการจำจุดนั้นไว้ เพื่อเปลี่ยนพิกเซลจุดต่อไปให้เป็นสีดำ และจะเปลี่ยนไปเรื่อยๆ จนกระทั่งเจอสีดำอีกครั้งหนึ่ง ตามด้วยการเปลี่ยนจุดสุดท้ายนั้นเป็นสีขาว แล้วจดจำไว้ เพื่อเป็นขอบต่อไป

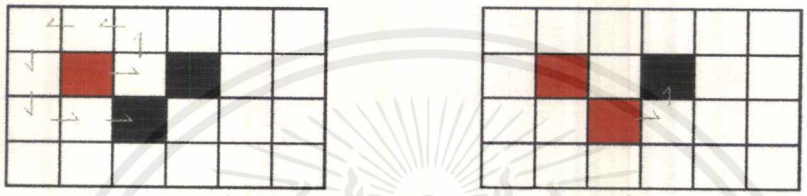


รูปที่ 3.3 แสดงการรวมกันทั้งด้านแนวตั้งและแนวนอน

แต่อย่างไรก็ตามขั้นตอนนี้จะต้องเกิดปัญหาขึ้นเพราะ หากขอบที่โดนกระทำการเป็นขอบบนสุด (ดูรูป) จะทำให้ Edge detection ไม่สมบูรณ์คือ ขอบของภาพจะไม่ใช่วงปิด เพราะฉะนั้น

จะต้องมีการแก้ปัญหาโดยการ เพิ่มการหาขอบโดยไล่สแกนมาทางแนวตั้งบ้าง และจำจุดต่างๆที่เปลี่ยนไปเอาไว้ แล้วจึงนำภาพที่ได้จำเอาไว้ทั้งทาง สแกนแนวตั้งและแนวนอน มาพล็อตทับกันอีกครั้ง ก็จะได้ภาพ Edge detection ที่สมบูรณ์แบบ คือเป็นลักษณะวงปิดนั่นเอง

3.1.4 Edge Contour เริ่มจากการสแกนตามแนวนอนเช่นเดิม จนกว่าจะเจอพิกเซลที่มีสีขาวจุดแรก จากนั้นจึงเข้าสู่ส่วนต่อไป



รูปที่ 3.4 แสดงขั้นตอนการหาขอบ

ส่วนต่อมา (ดูรูปที่ 3.4) จะทำการวิเคราะห์พิกเซลรอบข้าง ว่าจุดใดเป็นขอบ และทำการจำแต่ละจุดไว้จนครบรอบ (ไปเจอจุดแรกที่จำไว้) ส่วนที่สำคัญขั้นตอนนี้คือ การจำจุดที่เป็นขอบแล้วเก็บค่าไว้ในตัวแปรอะเรย์ (array)

3.1.5 การติดต่อไมโครคอนโทรลเลอร์ จะทำการติดต่อแบบอนุกรม (serial) โดยจะเอาจุดต่างๆที่จำไว้ในขั้นตอนที่แล้ว มาพิจารณาว่า ตำแหน่งเดิมของมอเตอร์ กับค่าที่จำมานั้นเท่ากันหรือไม่ ถ้า

-เท่ากัน ส่งรหัส "0" ให้ไมโครคอนโทรลเลอร์

-มากกว่า ส่งรหัส "1" ให้ไมโครคอนโทรลเลอร์

-น้อยกว่า ส่งรหัส "2" ให้ไมโครคอนโทรลเลอร์

ซึ่งการพิจารณาทุกๆตัวแปร จะกระทำห่างกันทุกๆ 0.04 วินาที เพื่อรอให้มอเตอร์ทำงานเสร็จก่อน และจะส่งรหัส "3" ให้ไมโครคอนโทรลเลอร์ เพื่อเปลี่ยนแกนการทำงานของมอเตอร์ (ซึ่งจะอธิบายต่อไป)

3.2 Software (ไมโครคอนโทรลเลอร์)

หลักการนั้นเราจะรับ ข้อมูลจาก PC โดยวิเคราะห์รหัสที่ส่งมา หากเป็น

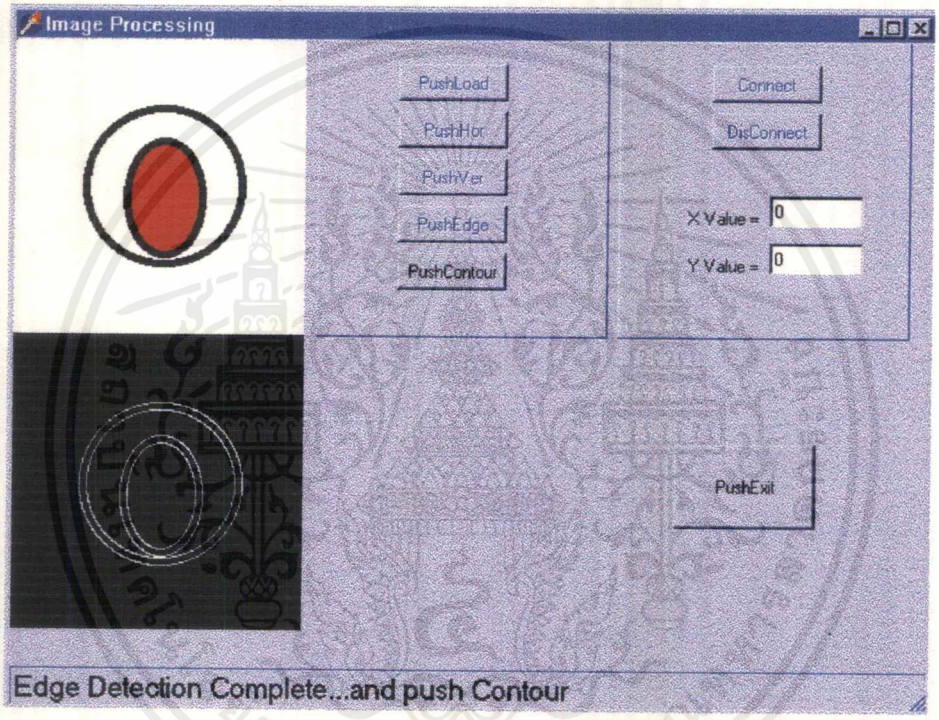
"1" จะส่งพัลซ์ ให้กับสเต็ปปิ้งมอเตอร์ ตัวขณะนั้นหมุนขวา

"2" จะส่งพัลซ์ ให้กับสเต็ปปิ้งมอเตอร์ ตัวขณะนั้นหมุนซ้าย

"0" จะไม่ส่งพัลซ์ ให้กับสเต็ปปิ้งมอเตอร์

"3" จะทำการเปลี่ยนการควบคุมสเต็ปปิ้งมอเตอร์ ไปเป็นอีกตัวหนึ่ง

เมื่อรับรหัสที่ส่งมาเสร็จเรียบร้อยหนึ่งรหัส ก็จะทำการรอรับรหัสตัวต่อไปมาวิเคราะห์ทันที



รูปที่ 3.5 แสดงหน้าจอของโปรแกรมสั่งงานจากคอมพิวเตอร์

3.3 Hardware

ส่วนของฮาดร์แวร์สามารถดูได้จาก แบบที่ได้ออกแบบไว้ในภาคผนวก

บทที่ 4
ผลการทดลอง

จากโครงงานนี้เป็นการนำไมโครครอนโทรลเลอร์มาใช้ควบคุมการเคลื่อนที่ในแนวแกน x และแนวแกน Y ซึ่งในภาคนี้จะเป็นการสร้างชิ้นงานของแกน x และแกน y ขึ้นมาแล้วนำไปควบคุมด้วยไมโครคอนโทรเลอร์ ให้มีการทำการเลื่อนของแกน x และ y ได้ ซึ่งการทดลองนั้นก็จะต้องประกอบกันหลายส่วน ตั้งแต่ส่วนโครงสร้างของชิ้นงาน กำลังขับของ Stepping Motor โดยการใช้พัลส์จากไมโครคอนโทรลเลอร์มาควบคุมการทำงานซึ่งการทดลองการทำงานก็จะประกอบด้วยส่วนต่างๆดังนี้

4.1 โครงสร้างของชิ้นงาน

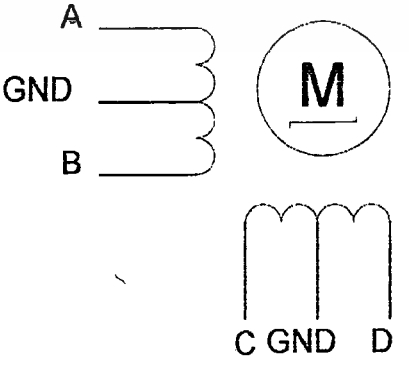
ชิ้นงานที่สร้างขึ้นนั้นมีการสร้างใน 2 แบบด้วยกันคือ

4.1.1 เป็นการใช้ skew ในการเคลื่อนที่ ซึ่งการใช้ skew ในการเคลื่อนที่นี้พบว่า สามารถเคลื่อนที่เป็นระยะทางที่ละเอียด แต่ก็มีข้อเสียคือเมื่อใช้การเคลื่อที่แบบนี้พบว่า ในการส่งกำลังงานจาก motor ไปยัง skew จะไม่สามารถดึงแกน x และ y ไปด้วยกันได้ และทำให้เกิดการติดขัดจนทำให้ไม่สามารถเคลื่อนที่ได้

4.1.2 ใช้เฟืองรางในการเคลื่อนที่ มีความละเอียดน้อยกว่าแบบ skew ทำให้การเคลื่อนที่สามารถเคลื่อนที่ได้รวดเร็วกว่าและกำลังส่งในการเคลื่อนที่ก็จะดีกว่า

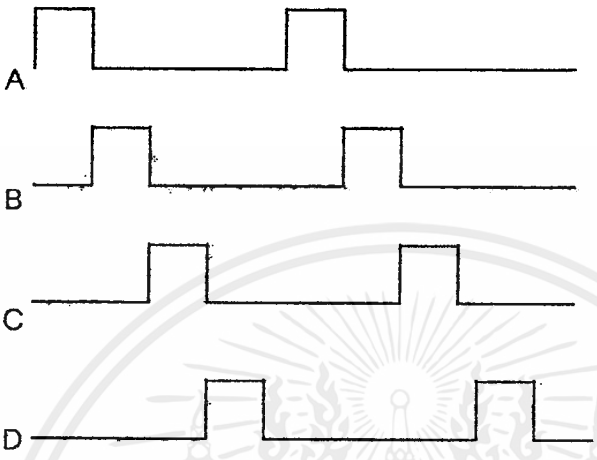
4.2 การทดลองป้อนพัลส์ให้แก่ stepping motor

stepping motor ที่ใช้ในโครงการนี้เป็นแบบ 6 สาย 4 เฟส ซึ่งมีลักษณะดังรูป



รูปที่ 4.1 แสดงลักษณะโครงสร้างของ stepping motor

ซึ่งจะมีเฟสทั้งหมด 4 เฟส โดย 2 เฟส จะอยู่ในขดลวดเดียวกัน ดังนั้นในการต่อจึงต้องต่อขดลวด gnd เข้าด้วยกัน และทำการกระตุ้นโดยการเรียงตามลำดับเฟส A-B-C-D ซึ่งในโครงงานนี้จะทำการกระตุ้นแบบ single phase ซึ่งจะมีลักษณะดังรูป



รูปที่ 4.2 แสดงการกระตุ้นแบบ single phase

ซึ่งจากการกระตุ้นแบบนี้จะให้ค่าแรงดันพัลส์ 1 จังหวะ สลับการทำงานไปในแต่ละ phase และพบว่าในการใช้แรงดันพลัส์จากไมโครคอนโทรลเลอร์นั้นค่าของแรงดันพลัส์จะมีกำลังงานไม่เพียงพอในการจ่ายให้แก่ stepping motor จึงต้องมีวงจร driver สำหรับการจ่ายกำลังงานให้แก่ stepping motor

4.3 การทดสอบการทำงานของโปรแกรม

เมื่อวงจรในการทำงานส่วนต่างๆเสร็จสิ้นลง ก็จะเป็นการทดลองในส่วนของโปรมแกรมที่ใช้ควบคุมการทำงาน ซึ่งโปรแกรมที่ใช้ควบคุมการทำงานนี้ได้แสดงไว้ในส่วนของภาคผนวกใช้ภาษาแอสแซมบลี้ในการเขียนโปรแกรม

4.3.1 ค่าผิดพลาดจากการเคลื่อนที่

จำนวนพัลซ์ที่ป้อน (pulse)	ระยะทาง (มิลลิเมตร)		%ผิดพลาด
	คำนวณ	วัดระยะจริง	
5	3.75	4	6.6%
10	7.5	8	6.6%
15	11.25	12	6.6%
20	15	16	6.6%
30	22.5	23.8	5.7%
50	37.5	39.8	6.13%

ตารางที่ 4.1 แสดงผลการวัดระยะทางจริงกับค่าที่คำนวณ

4.3.2 การตั้งระยะกล้องที่เหมาะสม

ใช้รูปอ้างอิง : เส้นตรงสีดำยาว 7 cm.

รูปที่ 4.3 แสดงรูปที่ใช้อ้างอิงในการทดลอง

ระยะกล้องกับภาพ(cm)	ระยะที่พล็อตได้	%ผิดพลาด
12	7.5	7.14
13	7.1	1.42
14	6.3	10
15	6.1	12.8

ตารางที่ 4.2 แสดงผลการวางระยะกล้องที่เหมาะสม

เพราะฉะนั้นจะได้ตำแหน่งกล้องที่เหมาะสมที่สุดคือ ห่างจากภาพ 13 cm.

และจะทำการติดตั้งกล้องให้มีระยะห่างจากภาพเท่านี้โดยตลอดการทดลอง

4.3.3 การทดลองเทียบระยะการพล็อตกับระยะจริง

ภาพ	ระยะของภาพ (cm.)	ระยะที่พล็อตได้ (cm.)	%ผิดพลาด
เส้นตรง	2	2.2	10
เส้นตรง	5	5.1	2
เส้นตรง	7	7.15	2.14
เส้นตรง	10	10.2	2
วงกลม	เส้ยผ่าศ.ก.= 2	1.85	7.5
วงกลม	เส้ยผ่าศ.ก.= 4	3.9	2.5
วงกลม	เส้ยผ่าศ.ก.= 6	6.1	1.66

เส้นผ่าศ.ก. = เส้นผ่าศูนย์กลาง

บทที่ 5
สรุปและวิจารณ์ผลการทดลอง

5.1 สรุปผลการทดลอง

จากการทดลองในบทที่ 4 ซึ่งได้ทำการออกแบบการวัดขนาดของภาพจริง กับขนาดที่สามารถทำการวัดได้ ซึ่งจากการทดลองสามารถที่จะสรุปผลการทดลองได้ดังต่อไปนี้คือ

5.1.1การวัดขนาดของภาพที่พล็อตได้เทียบกับขนาดภาพจริงที่ระยะตั้งกล้องต่างๆกัน จากผลการทดลองสามารถที่จะสรุปผลการทดลองได้ดังต่อไปนี้คือ

ระยะกล้อง(cm)	เปอร์เซนต์ผิดพลาด
12	7.14
13	1.42
14	10
15	12.8

ตารางที่ 5.1 สรุปผลการวัดระยะตั้งกล้องที่เหมาะสม

ซึ่ งจากค่าของเปอร์เซนต์ผิดพลาดที่ได้นี้พบว่าระยะของกล้องที่ห่างจากภาพที่จะจับเป็นระยะทางประมาณ 13 cm. จะได้ขนาดของภาพที่จับได้มีขนาดที่ใกล้เคียงขนาดของภาพจริงมากที่สุด

1.5.2การวัดขนาดของภาพที่ได้จากการพล็อตกับขนาดของภาพจริง

จากการทดลองในส่วนนี้จะเป็นการวัดรูปร่างต่างๆ ของขนาดจริงกับขนาดที่ได้จากการวัดโดยการส่งค่าตำแหน่ง x-y ให้แก่ x-y position พล็อตซึ่งในส่วนนี้จะใช้ค่าที่ดีที่สุดของระยะกล้องคือ 13 cm.สรุปผลการทดลองได้ดังต่อไปนี้

ภาพ	เปอร์เซนต์ผิดพลาด
เส้นตรง	10
เส้นตรง	2
เส้นตรง	2.14
วงกลม	2
วงกลม	7.5
วงกลม	2.5

ตารางที่ 5.2 สรุปผลการวัดรูปร่างต่างๆ

5.2 วิจารณ์ผลการทดลอง

จากผลการทดลองที่ได้กับโครงงานชิ้นนี้พบว่ามีข้อผิดพลาดต่างๆดังนี้

-การออกแบบโปรแกรมที่สร้างขึ้นได้ทำการกำหนดให้ระยะของภาพที่ห่างกันเป็นระยะ 1pixels ทำการส่งค่าให้แก่ steppingmotor หมุนไป 1 step ซึ่งจะได้ระยะทางที่เท่ากับ ซึ่งจุดนี้ก็คือความละเอียดของโครงงานนี้ที่สามารถที่จะทำได้ ไม่สามารถที่จะมีความละเอียดในการพล็อตค่าได้มากกว่านี้จะเห็นได้จากการพล็อตรูปร่างที่เป็นลักษณะของวงกลม ซึ่งรูปร่างที่ได้จะไม่เป็นวงกลมโดยสมบูรณ์จะมีลักษณะของส่วนที่เป็นเส้นหยักอยู่

-ตัวของHARDWARE การเคลื่อนที่ของ x-y position บนเฟืองร่างที่เป็นตัวนำพาการเคลื่อนที่มีการเคลื่อนที่ไม่สม่ำเสมอ จะเห็นได้จากที่จุดเริ่มต้นและจุดสุดท้ายจะไม่ทับกันสนิทพอดี

5.3ขอบเขตของโครงงาน

จากการที่ได้ทำการสร้างโครงงานชิ้นนี้ขึ้นมานั้นและก็ได้ทำการทดลองและปรับปรุงก็พบว่ามีข้อจำกัดของโครงงานบางประการ

5.3.1 SOFTWARE

ความสามารถของ software ที่ได้สร้างขึ้นมีขอบเขตจำกัดไม่สามารถจะทำการตรวจสอบภาพที่มีลักษณะดังต่อไปนี้

-ภาพที่มีลักษณะของเส้นขอบที่บาง ซึ่งเมื่อนำไปทำการ threshold เพื่อจะเปลี่ยนสีของภาพให้มีเพียง 2ระดับสี ซึ่งถ้าเป็นเส้นขอบที่บางจะทำให้เส้นนั้นหายไปไม่สามารถที่จะนำไปทำขั้นตอนต่อไปได้

-ภาพที่ใช้จะเป็นภาพที่เป็นวงปิด มีเพียงรูปเดียวภายในกรอบของภาพ ซึ่งตัว software จะทำการตรวจสอบรูปที่มีจุด x และ y เจอก่อนและทำรูปนั้นไปจนเสร็จโดยจะไม่ไปทำรูปอื่น

-รูปนั้นจะต้องไม่มีทางแยกของเส้นซึ่งยังไม่สามารถที่จะตัดสินใจเกี่ยวกับทางแยกได้

5.3.2 HARDWARE

ความสามารถของ hardware ในโครงงานนี้มีความสามารถดังต่อไปนี้

-สามารถเคลื่อนที่ได้ละเอียดที่สุด ทำให้การพล็อตวงกลมหรือรูปที่มีความละเอียดสูงมีขอบเขตที่จำกัด

5.4 การปรับปรุงและการพัฒนา

สำหรับการปรับปรุงและการพัฒนาเพื่อให้โครงงานชิ้นนี้ สามารถที่จะก่อให้เกิดประโยชน์สูงสุดและนำไปสู่การพัฒนาเพื่อให้โครงงานนี้มีขีดความสามารถที่สูงขึ้น ซึ่งในการพัฒนานั้นสามารถที่จะพัฒนานำไปใช้ประโยชน์ได้มากมาย ขึ้นอยู่กับความคิดของผู้ที่ต้องการจะพัฒนา แต่ในที่นี้ก็ขอเสนอความคิดเล็กๆน้อยๆบางประการซึ่งก็คิดว่าคงจะมีประโยชน์บ้าง

5.4.1 การนำไปใช้ประโยชน์

-สามารถที่จะนำไปประยุกต์ใช้เป็น เครื่องตัดโพม,ตัดสติกเกอร์

5.4.2 การพัฒนาขีดความสามารถ

-เพื่อความสามารถให้สามารถตรวจสอบรูปหลายๆรูปได้

-เพื่อความสามารถตัดสินใจทางแยกของรูปภาพได้

กิตติกรรมประกาศ

การทำโครงงานชิ้ นนี้จะไม่สามารถสำเร็จลุล่วงไปได้เลย ถ้าหากขาดบุคคลเหล่านี้ที่คอยให้ความช่วยเหลือมาตลอด ข้าพเจ้ากลุ่มผู้ทำโครงงานนี้สำนึกในพระคุณของท่านทุกๆคน

ผศ.ดร.สุรพันธ์ เอื้อไพบูลย์ อาจารย์ที่ปรึกษา แนะนำว่าควรจะทำงานไปในทิศทางใด สนับสนุนในการทำโครงงานจนกระทั่งสำเร็จลุล่วงไปได้ด้วยดี

คุณ ธีรยุทธ์ ตุณภรณ์ ที่ให้ยืมบอร์ดทดลองไมโครคอนโทรลเลอร์ Mcs-51

ท้ายที่สุดขอขอบพระคุณ พ่อ แม่ ที่ทำให้มีวันนี้ ทางกลุ่มผู้จัดทำขอขอบพระคุณท่านด้วยความจริงใจตลอดไป

บรรณานุกรม

1.รศ.สมยศ จุณณะปิยะ, 'การประยุกต์ใช้งานไมโครคอนโทรเลอร์ ตระกูล mcs51 ', คณะวิศวกรรมศาสตร์, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง,พิมพ์ครั้งที่ 2 พ.ศ. 2541

2.รศ.ดร.โยธิน เปรมปราณีรัตต์ ,'steppingmotor',วารสารวิศวสารลาดกระบัง,ปีที่9 ฉบับที่ 4 เมษายน พ.ศ. 2532,หน้า 46-56

3.ผศ. พิพัฒน์ เลาหสงคราม ,'ไมโครคอนโทรเลอร์ mcs51',วารสารวิศวสารลาดกระบัง, ปีที่ 9 ฉบับที่ 4 เดือน เมษายน พ.ศ. 2532,หน้า 60-68

4.Neil J. Rubenking ,'DELPHI PROGRAMMING FOR DUMMIES',พิมพ์ครั้งที่ 2 ปี1997

ภาคผนวก ก.

```
unit PiRGB;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialog
s,
  StdCtrls, ExtCtrls, RulerBox, ImageEnView, DBImageEn, ImageEnProc,
  ImageEnIO, ComCtrls, ComDrv32;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    PushExit: TButton;
    Panel2: TPanel;
    Image1: TImage;
    PushLoad: TButton;
    PushHor: TButton;
    PushVer: TButton;
    PushEdge: TButton;
    PushContour: TButton;
    Panel1: TPanel;
    ConnectBtn: TButton;
    DisConnectBtn: TButton;
    Timer1: TTimer;
    StatusBar1: TStatusBar;
    CommPortDriver1: TCommPortDriver;
    Edit1: TEdit;
    Edit2: TEdit;
    Label3: TLabel;
    Label4: TLabel;
    procedure PushHorClick(Sender: TObject);
    procedure PushEdgeClick(Sender: TObject);
    procedure PushExitClick(Sender: TObject);
    procedure PushLoadClick(Sender: TObject);
    procedure PushVerClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure PushContourClick(Sender: TObject);
    procedure ConnectBtnClick(Sender: TObject);
    procedure DisConnectBtnClick(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);

  private
    { Private declarations }
    procedure ApplyCommSettings;
  public
```

```
var
  Form1: TForm1;
  a:integer=0;
  b:integer=0;
  M:integer=0;
  N:integer=0;
  Ver:array[0..200,0..200]of longint;
  Hor:array[0..200,0..200]of longint;
  Start:array[0..200,0..200]of longint;
  Find:Tcolor;
  after:longint;
  before:longint;
  Keep:longint;
  axisX:array[0..1000]of integer;
  axisY:array[0..1000]of integer;
  axisXOld:integer=0;
  axisXNew:integer=0;
  axisYOld:integer=0;
  axisYNew:integer=0;
  changsend:boolean=true;
  drawing:boolean;
  ck:integer=0;
  EndX:integer;
  EndY:integer;

implementation
label 99;

{$R *.DFM}


procedure TForm1.PushLoadClick(Sender: TObject);
begin
    a:=0;
    Image1.Picture.LoadFromFile('c:\pi.bmp');
    StatusBar1.SimpleText := 'Please push "Hor  button"';
    PushLoad.Enabled := false;
    PushVer.Enabled := false;
    PushHor.Enabled := true;
    PushEdge.Enabled := false;
    PushExit.Enabled := true;
    repeat
    axisX[a]:=0;
    axisy[a]:=0;
    a:=a+1;
    until a=1001;
    a:=0;
```

```
end;

procedure TForm1.PushHorClick(Sender: TObject);
var L:Longint;
begin
    PushLoad.Enabled := false;
    PushHor.Enabled := false;
    PushVer.Enabled := false;
    PushExit.Enabled := false;

//*********************************************************************
*
//**************************** Threshold ****************************
*
    a:=0;
    b:=0;
    Repeat
        StatusBar1.SimpleText := 'Please Wait...';
        Repeat
            Label1.font.color := Canvas.Pixels[a,b];
            L := colorToRGB(Label1.font.color);
            if L<200 then
              begin
                Canvas.Pixels[a,b] := clwhite;
                a:=a+1;
              end
            else
              begin
                Canvas.Pixels[a,b] := clblack;
                a:=a+1;
              end;
        Until a=image1.Width;
        a:=0;
        b:=b+1;
    Until b=image1.Height;
    b:=0;
//*********************************************************************
**
//************************** Keep Value Hor ***********************
**
    a:=0;
    b:=0;
    Repeat
    Label1.font.color := Canvas.Pixels[a,b];
    before := colorToRGB(Label1.font.color);
    Repeat
```

```
        Label1.font.color := Canvas.Pixels[a,b];
        after := colorToRGB(Label1.font.color);
        if before=after then
          begin
            before := after;
            if Canvas.Pixels[a,b]=clwhite then
              begin
                Canvas.Pixels[a,b]:=clblack;
              end
            else
              begin
                Canvas.Pixels[a,b] := clblack;
              end;
          end
        else    //before <> after
          begin
            before := after;
            Canvas.Pixels[a,b] := clwhite;
          end;
        Hor[a,b] := Canvas.Pixels[a,b] ;
        a:=a+1;
    Until a=image1.Width;
    a:=0;
    b:=b+1;
  Until  b=image1.Height;
  b:=0;

    PushLoad.Enabled := false;
    PushExit.Enabled := true;
    PushVer.Enabled := true;
    StatusBar1.SimpleText := 'Please push "PushVer button"';
    Image1.Picture.LoadFromFile('c:\pi.bmp');
end;



procedure TForm1.PushVerClick(Sender: TObject);
var L:longint;
begin
    PushLoad.Enabled := false;
    PushVer.Enabled := false;
    PushHor.Enabled := false;
    PushEdge.Enabled := false;
    PushExit.Enabled := false;

//**********************************************************************
```

```
*
//*************************** Threshold ****************************
*
    a:=0;
    b:=0;
     Repeat
       StatusBar1.SimpleText := 'Please Wait...';
       Repeat
           Label1.font.color := Canvas.Pixels[a,b];
           L := colorToRGB(Label1.font.color);
           if L<200 then
             begin
                Canvas.Pixels[a,b] := clwhite;
                a:=a+1;
             end
           else
             begin
                Canvas.Pixels[a,b] := clblack;
                a:=a+1;
             end;
       Until a=image1.Width;
       a:=0;
       b:=b+1;
    Until b=image1.Height;
    b:=0;

//*********************************************************************
**
//***************************** Keep Value Ver **********************
**
    a:=0;
    b:=0;
    Repeat
    Label1.font.color := Canvas.Pixels[a,b];
    before := colorToRGB(Label1.font.color);
    Repeat
        Ver[a,b] := before;
        Label1.font.color := Canvas.Pixels[a,b];
        after := colorToRGB(Label1.font.color);
        if before=after then
          begin
             before := after;
             if Canvas.Pixels[a,b]=clwhite then
               begin
                  Canvas.Pixels[a,b]:=clblack;
               end
             else
               begin
```

```
                    Canvas.Pixels[a,b] := clblack;
                end;
            end
          else    //before <> after
            begin
                before := after;
                Canvas.Pixels[a,b] := clwhite;
            end;
          Ver[a,b] := Canvas.Pixels[a,b] ;
          b:=b+1;
      Until b=image1.Height;
      b:=0;
      a:=a+1;
    Until  a=image1.Width;
    a:=0;

      PushEdge.Enabled := true;
      PushExit.Enabled := true;
      StatusBar1.SimpleText := 'Please push "Edge button"';

end;

procedure TForm1.PushEdgeClick(Sender: TObject);
begin
      PushLoad.Enabled := false;
      PushVer.Enabled := false;
      PushHor.Enabled := false;
      PushEdge.Enabled := false;
      PushExit.Enabled := false;

//*********************************************************************
**
//****************************** Edge ********************************
**
      a:=0;
      b:=200;
      Repeat
        StatusBar1.SimpleText := 'Please Wait';
       Repeat
       Canvas.Pixels[a,b] := Hor[a,b];
       a:=a+1;
       Until a=200;
      a:=0;
      b:=b+1;
      Until b=400;
      a:=0;
      b:=200;
      Repeat
```

```
      Repeat
          if Ver[a,b]=Canvas.Pixels[a,b] then
            begin
                if Canvas.Pixels[a,b]=clWhite then
                  begin
                      Canvas.Pixels[a,b] := clWhite;
                  end
                else
                  begin
                      Canvas.Pixels[a,b] := clblack;
                  end;
            end
          else    // not equl
            begin
                Canvas.Pixels[a,b] := clWhite;
            end;
       a:=a+1;
       Until a=200;
     a:=0;
     b:=b+1;
     Until b=400;
     b:=200;
     Image1.Picture.LoadFromFile('c:\pi.bmp');

     PushContour.Enabled := true;
     PushExit.Enabled := true;
     StatusBar1.SimpleText := 'Edge Detection Complete...and push Con
tour';

end;

procedure TForm1.PushContourClick(Sender: TObject);
label 99;
var Check:boolean;
begin
     Timer1.Enabled:= false;
     PushLoad.Enabled := false;
     PushVer.Enabled := false;
     PushHor.Enabled := false;
     PushEdge.Enabled := false;
     PushExit.Enabled := false;
     PushContour.Enabled := false;
     Check := false;
//*************************************************************************
****
//************************** Start Picture at 0,0 ********************
*****
     a:=0;
```

```
    b:=200;
    M:=0;
    N:=0;
    Repeat
      Repeat
          start[M,N] :=  Canvas.Pixels[a,b] ;
          a:=a+1;
          m:=M+1;
      Until  a=200 ;
     a:=0;
     M:=0;
     b:=b+1;
     N:=N+1;
    Until  b=400;
    b:=200;
    N:=0;

    a:=0;
    b:=0;
    Repeat
     Repeat
         Canvas.Pixels[a,b] := start[a,b];
         a:=a+1;
     Until  a=200 ;
     a:=0;
     b:=b+1;
    Until  b=200;
    b:=0;

//*************************************************************************
****
//************************** Find point first ***********************
****
    a:=0;
    b:=0;
    M:=0;
    N:=0;

    Repeat
     Repeat
     Find :=  Canvas.Pixels[a,b] ;
      if Find=clwhite then
        begin
        axisX[M] := a ;
        axisY[N] := b ;
        Canvas.Pixels[a,b] := clred ;
        a:=200;
        b:=199;
```

PiRGB.~pa

```
        end
      else
        begin
        a:=a+1;
        end;
     Until  a=200 ;
     a:=0;
     b:=b+1;
    Until  b=200;
    b:=0;

//*************************************************************************
***
//***************************** Contour Process **********************
***
   Repeat
     a:= axisX[M];
     b:= axisY[N];
//*********************************
      a:=a+1;        //block 1
      Find :=  Canvas.Pixels[a,b];
      if Find=clwhite then
        begin
            M:=M+1;
            N:=N+1;
            axisX[M] := a;
            axisY[N] := b;
            Canvas.Pixels[a,b] := clblue ;
            goto 99;
        end
        else if Find=clred then
        begin
            Check := true;
        end;
//*******************************
      b:=b-1;        //block 2
      Find :=  Canvas.Pixels[a,b];
      if Find=clwhite then
        begin
        M:=M+1;
        N:=N+1;
        axisX[M] := a;
        axisY[N] := b;
        Canvas.Pixels[a,b] := clblue ;
        goto 99;
        end
```

```
            Check := true;
        end;
//*********************************
    a:=a-1;        //block 3
    Find :=  Canvas.Pixels[a,b];
    if Find=clwhite then
      begin
      M:=M+1;
      N:=N+1;
      axisX[M] := a;
      axisY[N] := b;
      Canvas.Pixels[a,b] := clblue ;
      goto 99;
      end
      else if Find=clred then
      begin
          Check := true;
      end;
//*********************************
    a:=a-1;        //block 4
    Find :=  Canvas.Pixels[a,b];
    if Find=clwhite then
      begin
      M:=M+1;
      N:=N+1;
      axisX[M] := a;
      axisY[N] := b;
      Canvas.Pixels[a,b] := clblue ;
      goto 99;
      end
      else if Find=clred then
      begin
          Check := true;
      end;
//*********************************
    b:=b+1;        //block 5
    Find :=  Canvas.Pixels[a,b];
    if Find=clwhite then
      begin
      M:=M+1;
      N:=N+1;
      axisX[M] := a;
      axisY[N] := b;
      Canvas.Pixels[a,b] := clblue ;
      goto 99;
      end
      else if Find=clred then
      begin
```

```
            Check := true;
        end;
//*******************************
    b:=b+1;        //block 6
    Find :=  Canvas.Pixels[a,b];
    if Find=clwhite then
        begin
        M:=M+1;
        N:=N+1;
        axisX[M] := a;
        axisY[N] := b;
        Canvas.Pixels[a,b] := clblue ;
        goto 99;
        end
        else if Find=clred then
        begin
            Check := true;
        end;
//*******************************
    a:=a+1;        //block 7
    Find :=  Canvas.Pixels[a,b];
    if Find=clwhite then
        begin
        M:=M+1;
        N:=N+1;
        axisX[M] := a;
        axisY[N] := b;
        Canvas.Pixels[a,b] := clblue ;
        goto 99;
        end
        else if Find=clred then
        begin
            Check := true;
        end;
//*******************************
    a:=a+1;        //block 8
    Find :=  Canvas.Pixels[a,b];
    if Find=clwhite then
        begin
        M:=M+1;
        N:=N+1;
        axisX[M] := a;
        axisY[N] := b;
        Canvas.Pixels[a,b] := clblue ;
        goto 99;
        end
        else if Find=clred then
        begin
```

```
            Check := true;
        end;
//*****************************
99:

   Until Check=true  ;

//*************************************************************************
***

    PushLoad.Enabled := true;
    PushExit.Enabled := true;
    ConnectBtn.Enabled := true;
    DisConnectBtn.Enabled := true;
    StatusBar1.SimpleText := 'Connect ready!!!"';
end;

procedure TForm1.PushExitClick(Sender: TObject);
begin
    CommPortDriver1.SendString('0');
    Timer1.Enabled := false;
    Close;
end;


procedure TForm1.FormCreate(Sender: TObject);
begin
    StatusBar1.SimpleText := 'Please push "Load button"';
    PushLoad.Enabled := true;
    PushVer.Enabled := false;
    PushHor.Enabled := false;
    PushEdge.Enabled := false;
    PushExit.Enabled := true;
    PushContour.Enabled := false;
    ConnectBtn.Enabled := false;
    DisConnectBtn.Enabled := false;

end;

//*************************************************************************
****
//*************************************************************************
****
//*********************  Commport Driver ***************************
****
//*************************************************************************
****
```

```
//********************** setting *********************************

procedure TForm1.ApplyCommSettings;
var wasConnected: boolean;
begin
  wasConnected := CommPortDriver1.Connected;
  // This change needs CommPortDriver not connected
  if wasConnected then
    DisconnectBtnClick( nil );

    // Reconnect
  if wasConnected then
    ConnectBtnClick( nil );
end;

procedure TForm1.ConnectBtnClick(Sender: TObject);
begin
     ck:=0;
     M:=0;
     N:=0;
     Timer1.Enabled := true;
  ApplyCommSettings;
  if CommPortDriver1.Connect then
  begin
    ConnectBtn.Enabled := false;
    DisconnectBtn.Enabled := true;
    StatusBar1.SimpleText := 'Connected';
  end
  else // Error !
    begin
      StatusBar1.SimpleText := 'Error: could not connect. Check COM p
ort settings and try again.';
      MessageBeep( 0 );
    end;
    axisXOLD:=0;
    axisYOLD:=0;
end;

procedure TForm1.DisConnectBtnClick(Sender: TObject);
begin
     CommPortDriver1.SendString('0');
     StatusBar1.SimpleText := 'Disconnected';
     CommPortDriver1.Disconnect;
     DisconnectBtn.Enabled := false;
     ConnectBtn.Enabled := true;
     Timer1.Enabled := false;
end;
//*******************************************************************
```

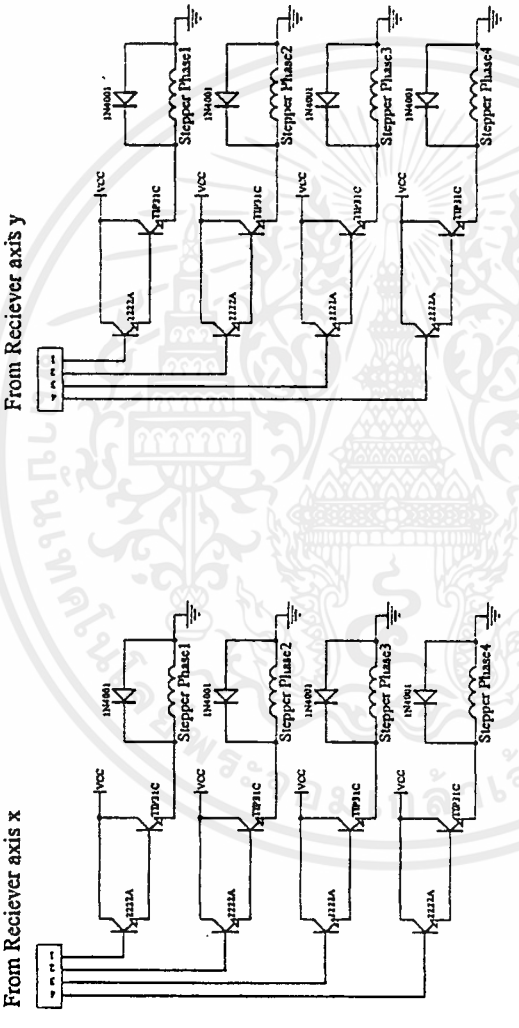
```
*****
//***************************** Send To MCS51 ************************
*****


procedure TForm1.Timer1Timer(Sender: TObject);
begin
    ck:=ck+1;
    axisXNew := axisX[M] ;
    edit1.Text := inttostr(axisX[M]);
    axisYNew := axisY[N] ;
    edit2.Text := inttostr(axisY[N]);
//*****************
    if changsend=true then
      begin
//*****************
    if axisXNew>axisXOld then
      begin
          CommPortDriver1.SendString('1');
          axisXOld := axisXOld+1;
          canvas.Pixels[axisXOLD,axisyOLD]:=clred;
      end
     else if axisXNew<axisXOLD then
      begin
          CommPortDriver1.SendString('2');
          axisXOLD := axisXOLD-1;
          canvas.Pixels[axisXOLD,axisyOLD]:=clred;
      end
     else if axisXNew=axisXOLD then
      begin
          CommPortDriver1.SendString('0');
      end;


    if axisX[M]=axisXold  then
      begin
      M:=M+1;
      changsend := false;
      canvas.Pixels[axisXOLD,axisyOLD]:=clred;
      CommPortDriver1.SendString('3');
      end;
 //******************
      end
    else //chang is axisY
        begin
//******************
      if axisYNew>axisYOld then
      begin
```

```
            CommPortDriver1.SendString('1');
            axisYOld := axisYOld+1;
            canvas.Pixels[axisXOLD,axisyOLD]:=clred;
        end
       else if axisYNew<axisYOLD then
        begin
            CommPortDriver1.SendString('2');
            axisYOLD := axisYOLD-1;
            canvas.Pixels[axisXOLD,axisyOLD]:=clred;
        end
       else if axisYNew=axisYOLD then
        begin
            CommPortDriver1.SendString('0');
        end;


     if axisY[N]=axisYold  then
       begin
           N:=N+1;
           changsend := true;
           canvas.Pixels[axisXOLD,axisyOLD]:=clred;
           CommPortDriver1.SendString('3');
       end;
 //*********************
     end;
//**************************

end;




end.
```

```
;*****************************************************************
;*********************  PROGRAM FOR  MCS51 ***********************
;*****************************************************************

        ORG 8100H

                MOV R7,#11H
                MOV 21H,#11H
                MOV 22H,#11H

                MOV DPTR,#0FC03H
                MOV A,#80H
                MOVX @DPTR,A

                MOV DPTR,#0FC01H
                MOV A,#11H
                MOVX @DPTR,A

                MOV DPTR,#0FC00H
                MOV A,#11H
                MOVX @DPTR,A

                MOV PCON,#00H
                MOV SCON,#50H
                MOV TMOD,#20H
                MOV TH1,#0E8H
                SETB TR1

      WAIT:     JNB RI,WAIT
                CLR RI
                MOV R6,SBUF

                CJNE R6,#30H,A31
                SJMP WAIT

      A31:      CJNE R6,#31H,A32
                LCALL UP
                SJMP WAIT
```

```
A32:    CJNE R6,#32H,A33
        LCALL DOWN
        SJMP WAIT

A33:    CJNE R6,#33H,BACK
        LCALL CHG
BACK:   SJMP WAIT

        UP:     MOV A,R7
                RR A
                MOV R7,A
                MOVX @DPTR,A
                RET

        DOWN:   MOV A,R7
                RL A
                MOV R7,A
                MOVX @DPTR,A
                RET

        CHG:    MOV 20H,DPL
                JNB 00H,PORTB       ;jump when FC00H

                MOV 21H,R7
                DEC DPL
                MOV R7,22H
                RET
         PORTB: MOV 22H,R7
                INC DPL
                MOV R7,21H
                RET


        END
```

ภาคผนวก ข.



รูป แสดงวงจรขับมอเตอร์ปีนบอเตอร์