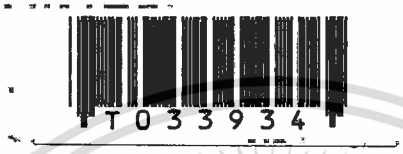
Programmable Stepping Motor Controller



โดย

นายชนพ แซ่ตัง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมการวัดคุม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2541

ปริญญานิพนธ์ปีการศึกษา 2541

ภาควิชา เทคโนโลยีการวัดคุมทางอุตสาหกรรม

คณะ วิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง

เรื่อง Programmable Stepping Motor Controller

ผู้จัดทำ

นายชนพ แซ่ตัง 38014098

.....อาจารย์ที่ปรึกษา

(ผศ. ประภาษ อุดคกิมาพันธุ์)

Programmable Stepping Motor Controller

นายชนพ แซ่ตัง

ผศ. ประภาษ อุคคกิมาพันธ์ อาจารย์ที่ปรึกษา

ปีการศึกษา 2541

บทคัดย่อ

โครงงานนี้เป็นการพัฒนาการออกแบบระบบเชิงกล ให้ดีขึ้น โดยใช้สเตปปิงมอเตอร์ ซึ่งเป็นมอเตอร์ชนิดหนึ่งที่มีความสามารถในความแม่นยำในการเคลื่อนที่สูงแต่ยุ่งยากในการใช้งานมาประยุกต์ให้ง่ายขึ้นโดยใช้ไมโครคอนโทรลเลอร์มาโปรแกรมช่วยให้การใช้งานทำได้ง่ายขึ้น โดยสามารถ ควบคุมความเร็ว ทิศทาง อัตราเร่ง และจำนวนสเต็ปที่ต้องการเคลื่อนที่ได้ โดยใช้การโปรแกรมไว้ล่วงหน้า และยังมีอินพุตและเอาต์พุตสำรอง สำหรับ ใช้โปรแกรมได้อีกด้วย

Programmable Stepping Motor Controller

Chanop Saetang

Prapart Ukakimapan Advisor

1999

ABSTRACT

This Project Is Development The Mechanical Design By Using Stepping Motor .That Stepping Motor Has High Precision To Motion But Dificult To Set In To System. This Project Will Use Microcontroller To Program For Easily Use Stepping Motor. So This Project can control Motion Factor in Velocity Direction Step and Acceleration . And This Controller has I/O for Programming .

สารบัญ

	หน้า
บทที่ 1 บทนำ	
1.1 รายละเอียดของวิทยานิพนธ์	1
1.2 วัตถุประสงค์ของปริญญานิพนธ์	2
1.3 ประโยชน์ที่ได้รับ	2
1.4 ขอบเขตของโครงงาน	3
บทที่ 2 ทฤษฎีและหลักการที่ใช้	
2.1 หลักการทำงานโดยทั่วไปของสเตปปิงมอเตอร์	4
2.2 ประเภทของสเตปปิงมอเตอร์	6
2.3 สถาปัตยกรรมของไมโครคอนโทรลเลอร์	12
บทที่ 3 การออกแบบและการสร้าง	
3.1 การออกแบบ Controller	40
3.2 การออกแบบ Driver	42
3.3 การออกแบบส่วนของโปรแกรม	44
บทที่ 4 การนำไปใช้งาน	
4.1 ชุดคำสั่งของตัว Controller และตัวอย่างการใช้งาน	45
4.2 การนำไปใช้ในส่วน Stand Alone Mode	46
4.3 การนำไปใช้ในส่วน Remote Mode	46
บทที่ 5 ข้อเสนอแนะและวิจารณ์	
5.1 ปัญหาและจุดบกพร่องของระบบ	**48**
5.2 แนวทางการพัฒนาต่อ	**48**
5.3 สรุป	**48**
ภาคผนวก	

สารบัญ (ต่อ)

กิตติกรรมประกาศ

หนังสืออ้างอิง

สารบัญรูปภาพ

	หน้า
รูปที่ 2.1 แสดงบล็อกไดอะแกรมของตัวควบคุมสำหรับควบคุมการทำงานของสเตปปิงมอเตอร์	4
รูปที่ 2.2 แสดงสนามแม่เหล็กที่เกิดขึ้นในลักษณะต่าง ๆ	4
รูปที่ 2.3 แสดงแรงดึงดูดซึ่งทำให้เกิดแรงบิด (Torque) หมุนโรเตอร์ที่อยู่ในตำแหน่งสมดุลย์	5
รูปที่ 2.4 แสดงโครงสร้างของสเตปปิงมอเตอร์แบบแม่เหล็กถาวรสี่เฟส	6
รูปที่ 2.5 แสดงโครงสร้างของสเตปปิงมอเตอร์แบบแปรค่าความต้านทานของสนามแม่เหล็กได้	7
รูปที่ 2.6 แสดงเส้นแรงแม่เหล็กที่เกิดขึ้น	8
รูปที่ 2.7 แสดงตำแหน่งของสถานะที่สมดุลย์เมื่อเกิดการกระตุ้นที่เฟสใดเฟสหนึ่ง	9
รูปที่ 2.8 แสดงโรเตอร์และสเตเตอร์ของมอเตอร์สามเฟสและสี่เฟส	9
รูปที่ 2.9 แสดงภาพหน้าตัดของมอเตอร์สี่เฟส ที่มีฟันโรเตอร์ห้าสิบซี่ มุมสเตป 1.8 องศา	10
รูปที่ 2.10 แสดงโครงสร้างของสเตปปิงมอเตอร์แบบไฮบริดจ์	10
รูปที่ 2.11 แสดงภาพการวางโรเตอร์ตามยาวเพื่อใช้สำหรับสร้างสนามแม่เหล็กขั้วเดียวกัน	11
รูปที่ 2.12 แสดงหลักการทำงานของสเตปปิงมอเตอร์แบบไฮบริดจ์	11
รูปที่ 2.13 ไดอะแกรมโครงสร้างของ 8051	14
รูปที่ 2.14 แผนภูมิหน่วยความจำของ 8051	17
รูปที่ 2.15 สถาปัตยกรรมภายในของ 8051	19
รูปที่ 2.16 ไดอะแกรมขาของ 8051 แบบ DIP	19
รูปที่ 2.17 โครงสร้างของพอร์ท 0	20
รูปที่ 2.18 โครงสร้างของพอร์ท 1	22
รูปที่ 2.19 โครงสร้างของพอร์ท 2	22
รูปที่ 2.20 โครงสร้างของพอร์ท 3	23
รูปที่ 2.21 ค่าของรีจิสเตอร์เมื่อเกิดการรีเซ็ต 8051	24

สารบัญรูปภาพ (ต่อ)

	หน้า
รูปที่ 2.22 วงจรออสซิเลเตอร์ภายใน 8051	26
รูปที่ 2.23 8051 ที่ทำงานโดยสัญญาณที่มาจากภายนอก	27
รูปที่ 2.24 ลำดับสถานะการทำงานใน MCS-51	28
รูปที่ 3.1 แสดงวงจรส่วนของ คอนโทรลเลอร์ที่ได้ออกแบบไว้	41
รูปที่ 3.2 แสดงวงจรไดร์ฟที่ออกแบบไว้	42
รูปที่ 3.3 แสดงลายแผ่นวงจรพิมพ์ของคอนโทรลเลอร์ที่ออกแบบไว้	43
รูปที่ 3.4 แสดงลายแผ่นวงจรพิมพ์ของวงจรไดร์ฟที่ออกแบบไว้	43
รูปที่ 3.5 แสดงโฟลว์ชาร์ตของโปรแกรมหลัก	44
รูปที่ 4.1 แสดงการใช้งานในส่วน Stand Alone Mode	46

บทที่ 1
บทนำ

ในอุปกรณ์เชิงกลต่าง ๆ นั้น ส่วนใหญ่ประกอบด้วยการเคลื่อนที่ 2 แบบหลัก ๆ คือการเคลื่อนที่ เชิงมุม (การหมุน) และการเคลื่อนที่เชิงเส้น (การเลื่อนที่) แล้ว นำการเคลื่อนที่สองส่วนนี้มาทำงานสัมพันธ์กัน โดยมีการควบคุม ตัวแปรของการเคลื่อนที่ ไม่ว่าจะเป็น ความเร็ว ระยะทาง องศาการหมุน ซึ่งการออกแบบอุปกรณ์เหล่านี้ มักจะประสบปัญหาเกี่ยวกับการควบคุมตัวแปรที่ว่ามา ให้สัมพันธ์กัน และหากมีการออกแบบระบบที่ไม่ดี ยังส่งผลให้เมื่อมีการเปลี่ยนแปลงกระบวนการในการทำงาน ผู้ออกแบบต้องมาเปลี่ยนแปลงระบบทางด้าน Hardware และ Software อีกด้วย หากเราสามารถควบคุมให้ ตัวแปรเหล่านี้ทำงานร่วมกันอย่างง่าย และรวดเร็ว มีการเปลี่ยนแปลงแก้ไขตัวแปรที่เราสามารถตั้งค่าไว้ได้โดยไม่ยุ่งยาก ก็จะทำให้การพัฒนาสร้างอุปกรณ์เชิงกลต่าง ๆ ทำได้อย่างรวดเร็ว ลดขั้นตอนในการออกแบบลงได้มาก

สเตปปิ้งมอเตอร์เป็นอุปกรณ์ที่มีความเป็นไปได้มากที่สุด ที่จะทำให้แนวคิดข้างต้นสามารถนำไปออกแบบสร้างได้จริง เพราะ สเตปปิ้งมอเตอร์เป็นอุปกรณ์ที่เปลี่ยนแปลงอินพุทซึ่งเป็นพัลซ์ ไปเป็นการเคลื่อนที่ เป็น Step ที่เป็นการหมุนที่ เพลาของมอเตอร์ ทำให้เราสามารถควบคุมความเร็วหรือจำนวนรอบการหมุนได้ดี เพราะเป็นการควบคุมแบบ Open Loop ทำให้ไม่มี Error และยังมีองศาต่อ step ที่ต่ำเหมาะกับงานที่ต้องการความละเอียดสูง

ในการใช้งานสเตปปิ้งมอเตอร์โดยทั่วไป มักใช้การควบคุมด้วยไมโครโปรเซสเซอร์หรือไมโครคอนโทรลเลอร์ ที่มีลักษณะซับซ้อนและไม่ค่อยมีความยืดหยุ่นในการใช้งานเช่น ใช้คอมพิวเตอร์เป็นตัวคอนโทรลเลอร์ หรือการที่ออกแบบให้ส่วนของคอนโทรลเลอร์ปะปนอยู่กับส่วนของไดรเวอร์ เป็นต้น การที่ออกแบบเช่นดังที่กล่าวมานี้ทำให้การใช้งานมีความยืดหยุ่นน้อย อีกทั้งไม่เอื้ออำนวยในการใช้งาน เช่น เมื่อต้องเข้าไปแก้ระยะทางหรือความเร็วในการเคลื่อนที่โดยใช้วิธีแก้ทางอีพรอม(EPROM) ด้วยภาษาเครื่องหรือภาษาแอสเซมบลี (ASSEMBLY) ทำให้ผู้ที่จะนำสเตปมอเตอร์ไปใช้งานได้จะต้องมีความรู้ในโครงสร้างและภาษาของไมโครโปรเซสเซอร์หรือไมโครคอนโทรลเลอร์ และนอกจากนี้แล้ว ยังต้องรู้ฮาร์ดแวร์ของเครื่องเป็นอย่างดี จึงทำให้ผู้ที่ไม่มีความรู้หรือผู้ที่ไม่มีส่วนในการสร้าง มีความยากที่จะนำสเตปปิ้งมอเตอร์ไปใช้งาน,ทำการแก้ไขข้อผิดพลาดหรือ ปรับปรุงเปลี่ยนแปลงการทำงานได้

1.1 รายละเอียดของวิทยานิพนธ์

ส่วนของปริญญานิพนธ์นี้ได้ ศึกษา ออกแบบ และสร้างระบบ การควบคุมสเตปปิ้งมอเตอร์ที่มีความง่าย (USER FRIENDLY) และยืดหยุ่นในการใช้งาน (FLEXIBLE) เพื่อให้ใช้งานสเตปปิ้งมอเตอร์ให้ได้สะดวก สบายยิ่งขึ้น

โดยระบบที่ออกแบบไว้นั้น มี 2 Mode คือ ในส่วน Stand Alone Mode และ ส่วน Remote Mode โดยที่ในส่วน Stand Alone นั้น ผู้ใช้สามารถนำไปใช้ได้โดยการป้อนคำสั่งให้ตัว Controller ผ่านทาง พอร์ตอนุกรม RS-232 C ครั้งแรก แล้ว หลังจากนั้น ตัว Controller ก็จะเก็บข้อมูลไว้ใน Ram ที่มีการ Back up ไว้ เมื่อสั่งให้ Run แล้ว ตัว Controller ก็จะทำตาม Sequencer ที่ได้โปรแกรมเก็บไว้ โดยไม่ต้องใช้ พอร์ตอนุกรมทำการ Program อีกต่อไป ส่วนใน Remote Mode นั้น ผู้ใช้จะต้องต่อ Port อนุกรม RS-232C แล้วทำการส่ง คำสั่งทีละคำสั่งไปยังที่ตัว Controller เมื่อปฏิบัติคำสั่งเสร็จ Controller ก็จะรอรับคำสั่งต่อไปจาก พอร์ตอนุกรม ใน ส่วนของ Remote Mode นั้น สามารถที่จะต่อ Controller ร่วมกัน ได้ถึง 16 ตัว ต่อ 1 พอร์ต RS-232C

นอกจากนี้ Controller ที่ออกแบบไว้ยังได้ออกแบบให้มี Port อเนกประสงค์ ขนาด 8 บิท เพื่อช่วยให้มีการทำงานเป็น Sequence ได้ง่ายขึ้น และยังออกแบบให้มีปุ่มหยุดฉุกเฉิน เพื่อใช้ในกรณีที่เกิดอุบัติเหตุ

วัตถุประสงค์ของปริญญานิพนธ์

1. เพื่อศึกษาแนวทางการพัฒนาการออกแบบเครื่องมือเชิงกลให้ทำได้ง่ายและสะดวกขึ้น
2. เพื่อศึกษาหลักการทำงานของ สเตปปิ้งมอเตอร์อย่างละเอียด
3. เพื่อศึกษาแนวทางในการนำสเตปปิ้งมอเตอร์ไปใช้ให้ง่ายขึ้น
4. เพื่อศึกษาทฤษฎีและหลักการทำงาน ของ Microcontroller
5. เพื่อศึกษาหลักการทำงานของอุปกรณ์เพาเวอร์อิเล็กทรอนิคส์

ประโยชน์ที่ได้รับ

1. ทำให้การออกแบบเครื่องมือเชิงกลได้ง่ายและรวดเร็ว
2. ทำให้นำสเตปปิ้งมอเตอร์มาใช้ได้โดยที่ไม่ต้องศึกษาหลักการทำงาน หรือวิธีการนำไปใช้ ให้ยุ่งยาก
3. ได้รับความรู้เพิ่มเติมในส่วนที่เกี่ยวข้อง

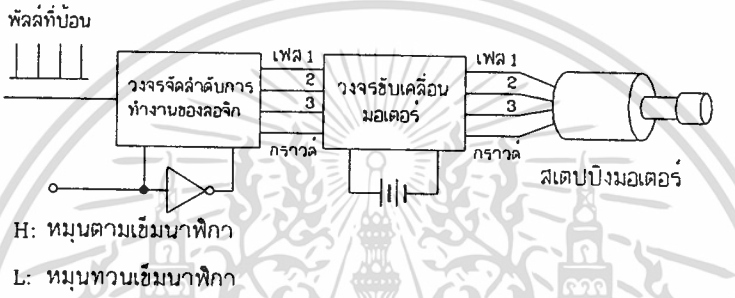
ขอบเขตของโครงงาน

1. ออกแบบและสร้างระบบควบคุมสเตปปิ้งมอเตอร์ โดยเลือกระบบที่สามารถนำไปประยุกต์ใช้งานได้
2. นำระบบที่ออกแบบขึ้นมาไปสร้างเป็น Application เพื่อให้เห็นประโยชน์ของอุปกรณ์ที่สร้างขึ้น

บทที่ 2
ทฤษฎีและหลักการที่ใช้

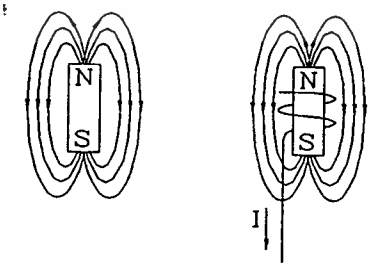
2.1 หลักการทำงานโดยทั่วไปของสเตปปิงมอเตอร์ (Stepping Motors)

สเตปปิงมอเตอร์เป็นมอเตอร์ไฟฟ้ากระแสสลับชนิดหนึ่ง ซึ่งการทำงานของสเตปปิงมอเตอร์นั้น จะขึ้นอยู่กับพัลส์ (Pulse) ที่ป้อนให้กับขดลวดเฟสของมอเตอร์ในลำดับที่ถูกต้องด้วยวงจรลอจิกสำหรับจัดลำดับและกระแสที่พอเพียงโดยวงจรขับ



ภาพที่ 2.1 แสดงบล็อกไดอะแกรม (Block Diagram) ของตัวควบคุมสำหรับควบคุมการทำงานของสเตปปิงมอเตอร์

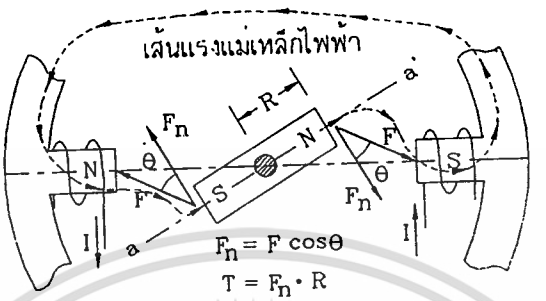
จากบล็อกไดอะแกรมภาพที่ 2.1 แสดงถึงหลักการทำงานพื้นฐานของสเตปปิงมอเตอร์ ซึ่งตัวควบคุมประกอบด้วยวงจรลอจิกสำหรับจัดลำดับ วงจรขับสัญญาณความถี่อินพุต และสัญญาณควบคุมทิศทางการหมุน



(ก) (ข)

ภาพที่ 2.2 แสดงสนามแม่เหล็กที่เกิดขึ้นในลักษณะต่าง ๆ

ในภาพที่ 2.2 แสดงหลักการพื้นฐานของเส้นแรงแม่เหล็ก โดยภาพ (ก) แสดงสนามแม่เหล็ก โดยภาพ (ก) แสดงสนามแม่เหล็ก (Magnetic Field) ที่เกิดจากแม่เหล็กถาวร ส่วนภาพ(ข) แสดงสนามแม่เหล็กไฟฟ้าที่เกิดจากกระแส(I)



ภาพที่ 2.3 แสดงแรงดึงดูดซึ่งทำให้เกิดแรงบิด(Torque) หมุนโรเตอร์ให้อยู่ในตำแหน่งที่สมดุลย์

จากภาพที่ 2.3 จะเห็นว่าแท่งแม่เหล็กที่ติดอยู่กับเพลา(Shaft) จะสามารถหมุนได้โดยอิสระ แม้ว่าจะไม่มีขั้วไฟฟ้ามากระตุ้นที่สเตเตอร์ ซึ่งถ้าก่อนที่จะมีกระแสไฟฟ้ามากระตุ้นที่สเตเตอร์ ตำแหน่งของโรเตอร์แม่เหล็กจะอยู่ตามแกน a-a' ซึ่งทำมุม θ กับแกนของขั้วแม่เหล็กไฟฟ้า ดังนั้น จะได้เส้นแรงแม่เหล็กที่เกิดจากการดึงดูดของขั้วแม่เหล็กที่ต่างกัน ทำให้เกิดส่วนของแรงบิดปกติ

$$F_n = F\cos\theta \text{ (แรงนี้ตั้งฉากกับแกน a-a')} \qquad (2\text{-}1)$$

เมื่อ F คือแรงแม่เหล็กที่เกิดจากการดึงดูดของขั้วแม่เหล็กที่ต่างกัน

F_n คือแรงปกติที่เป็นผลมาจากแรงแม่เหล็ก

θ คือมุมที่เกิดจากตำแหน่งแกนของอาร์มาเจอร์แม่เหล็ก กับตำแหน่งแกนขั้วของแม่เหล็กไฟฟ้า

จะได้ผลรวมของแรงบิดดังนี้

$$T = F_n R \qquad (2\text{-}2)$$

ทำให้โรเตอร์หมุนไปในทิศทางตามเข็มนาฬิกา จนกว่าแกนของโรเตอร์ a-a' จะอยู่ในแนวเดียวกับแกนของสเตเตอร์ ถ้าหากมีคู่ของขั้วแม่เหล็กไฟฟ้าหลาย ๆ ขั้ววางอยู่ตำแหน่งรอบ ๆ สเตเตอร์ และถ้าหากขั้วเหล่านั้นถูกกระตุ้นด้วยกระแสพัลส์ ในลักษณะที่เรียงลำดับและ

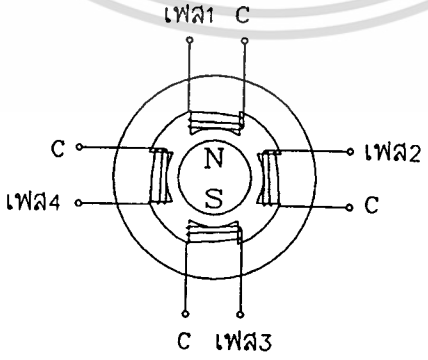
ต่อเนื่องก็จะทำให้โรเตอร์หมุนไปตามการหมุนของสนามแม่เหล็กที่เกิดจากการเปิดปิดวงจร (Switching) ที่เรียงลำดับของขดลวดขั้วแม่เหล็กไฟฟ้าของสเตเตอร์

2.2 ประเภทของสเตปปิงมอเตอร์

2.2.1 สเตปปิงมอเตอร์แบบแม่เหล็กถาวร (Permanent Magnet Stepping Motor)

สเตปปิงมอเตอร์ชนิดนี้จะใช้แม่เหล็กถาวรเป็นโรเตอร์ และมีขั้วฟันของสเตเตอร์ล้อมรอบ ภาพที่ 2.4 เป็นตัวอย่างของสเตปปิงมอเตอร์แบบแม่เหล็กถาวรสี่เฟส โดยที่ โรเตอร์เป็นทรงกระบอก สเตเตอร์มีขั้วฟันสี่ขั้วโดยที่แต่ละขั้วจะมีขดลวดพันโดยรอบ เพื่อทำให้เกิดการสร้างสนามแม่เหล็ก เมื่อต้องการให้สเตปปิงมอเตอร์แบบแม่เหล็กถาวรมีขนาดมุมสเตป (Step) เล็กลง จะต้องเพิ่มจำนวนขั้วแม่เหล็กของโรเตอร์เนื่องจากการสร้างแม่เหล็กถาวรให้มีโครงสร้างแบบมีขั้วแม่เหล็กหลาย ๆ ขั้วนั้นทำได้ยาก

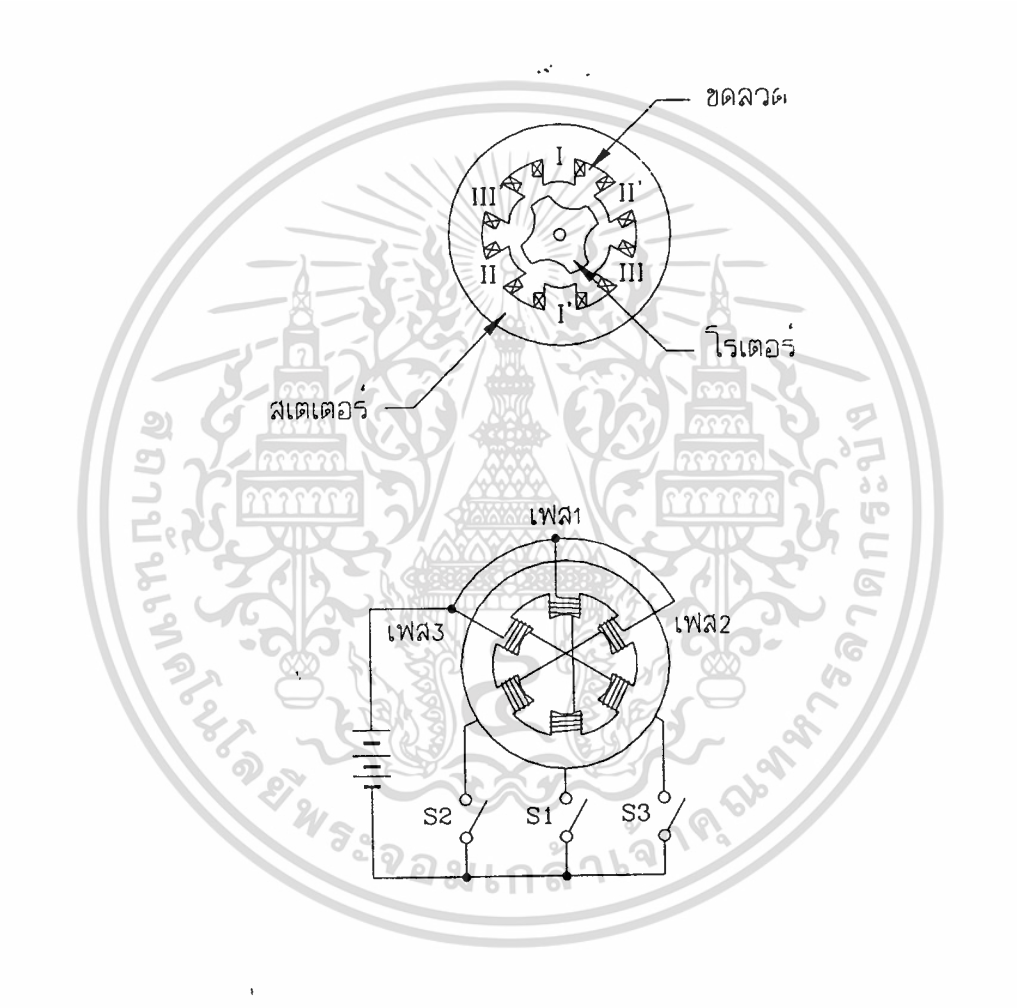
ลักษณะทั่วไปของมอเตอร์แบบนี้ก็คือ โรเตอร์จะถูกยึดอยู่กับที่แม้ว่าจะไม่มีการกระตุ้นเฟส ลักษณะเช่นนี้เรียกว่า ดีเทนท์แมคคานิคซั่ม (Detent Mechanism) ข้อเสียของสเตปปิงมอเตอร์แบบแม่เหล็กถาวรคือมีขนาดมุมสเตปใหญ่ ทำให้มีความละเอียดของสเตปต่อรอบน้อยมากเนื่องจากว่าโครงสร้างของโรเตอร์เป็นแม่เหล็กถาวร การสร้างแม่เหล็กถาวรให้มีหลายขั้วทำได้ยากดังที่กล่าวมาแล้ว ทำให้ไม่สามารถสร้างสเตปให้มีขนาดเล็กลงได้ สเตปปิงมอเตอร์แบบแม่เหล็กถาวรส่วนใหญ่จะมีโครงสร้างขนาดเล็ก ทำให้ค่าของแรงบิดที่ได้ต่อหน่วยปริมาตรมีค่าต่ำถ้าต้องการปรับปรุงประสิทธิภาพในเรื่องของแรงบิด แม่เหล็กถาวรที่ใช้จะต้องทำมาจากสารแม่เหล็กที่มีสภาพความเป็นแม่เหล็กสูง



ภาพที่ 2.4 แสดงโครงสร้างของสเตปปิงมอเตอร์แบบแม่เหล็กถาวรสี่เฟส

2.2.2 สเตปปิงมอเตอร์แบบแปรค่าความต้านทานของสนามแม่เหล็กได้(Variable Reluctance Stepping Motor)

สเตปปิงมอเตอร์ชนิดนี้เรียกอีกอย่างหนึ่งว่า วีอาร์สเตปปิงมอเตอร์(VR Stepping Motor) โดยโครงสร้างมีโรเตอร์และสเตเตอร์ทำจากโลหะซิลิคอน(Silicon) ซึ่งเป็นสารแม่เหล็กที่มีสภาพซึมซาบทางแม่เหล็ก (Permeability) สูง และสามารถให้เส้นแรงแม่เหล็กไหลผ่านได้มากโดยโรเตอร์จะติดอยู่กับแกนของมอเตอร์ และสเตเตอร์จะติดอยู่กับโครงของตัวมอเตอร์ ดังภาพที่ 2.5



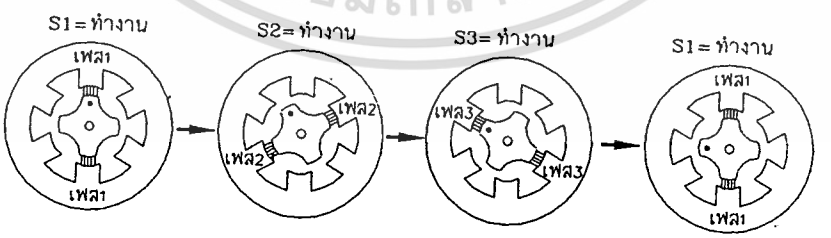
ภาพที่ 2.5 แสดงโครงสร้างของสเตปปิงมอเตอร์แบบแปรค่าความต้านทานของสนามแม่เหล็กได้

ในภาพเป็นภาพตัดขวางของสเตปปิงมอเตอร์ซึ่งเป็นมอเตอร์สามเฟส โดยที่โรเตอร์มีซี่ฟันสี่ซี่ และสเตเตอร์จะมีซี่ฟันอยู่หกซี่อยู่ในตำแหน่งตรงข้ามและทำมุม 180 องศา ต่อกันเป็นอนุกรมหรือขนานก็ได้แต่ในภาพต่อในลักษณะอนุกรม จะเห็นว่าฟันของสเตเตอร์สองซี่ที่มีเฟสเดียวกันจะมีขั้วแม่เหล็กตรงข้ามซึ่งกันและกัน สมมุติว่าฟัน I,II และ III มีขั้วเป็นขั้วเหนือ (N) และ I',II',III' เป็นขั้วใต้ (S) เมื่อมีการกระตุ้น กระแสแต่ละเฟสจะทำให้เกิดเส้นแรงแม่เหล็กขึ้น ดังภาพที่ 2.6

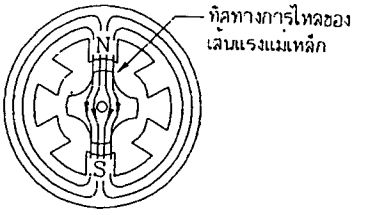
นอกจากนี้ฟันของโรเตอร์จะมีตำแหน่งในแนวเดียวกันดับฟันของสเตเตอร์ มีผลทำให้ความต้านทางทางแม่เหล็กมีค่าน้อยที่สุด โดยจะถือว่าสถานะนี้เป็นสถานะที่สมดุลย์ และเป็นตำแหน่งที่มีเส้นแรงแม่เหล็กสั้นที่สุด ดังภาพที่ 2.7

โครงสร้างพื้นฐานของสเตปปิงมอเตอร์แบบแปรค่าความต้านทางของสนามแม่เหล็กได้จะมีลักษณะดังนี้

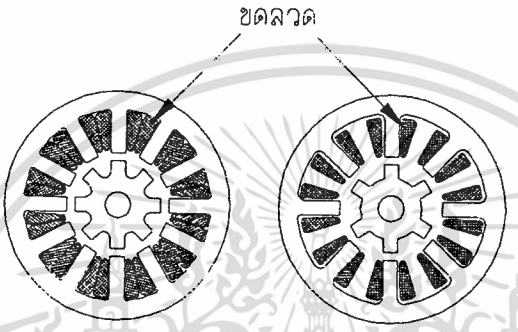
1) ช่องว่างอากาศ (Airgap) ควรจะมีขนาดเล็กที่สุดเท่าที่จะทำได้ ซึ่งหมายถึงว่าช่องว่างอากาศระหว่างฟันของโรเตอร์กับสเตเตอร์ควรจะมีค่าความห่างกันน้อยมาก เพื่อให้เกิดค่าของแรงบิดสูงสุด และมีตำแหน่งที่แน่นอนมากขึ้น เนื่องจากว่าระดับของแรงเคลื่อนแม่เหล็ก (Magnetomotive Force) มีระดับเดียวกัน ช่องอากาศขนาดเล็ก จะทำให้เส้นแรงแม่เหล็กสูงกว่า ซึ่งมีผลทำให้ค่าของแรงบิดสูงมากขึ้น
2) สำหรับมุมสเตปเล็ก ๆ จากภาพที่ 2.8 (ก) แสดงมอเตอร์สามเฟสที่สเตเตอร์มีฟันสิบสองซี่ และมีฟันที่โรเตอร์แปดซี่ ส่วนภาพที่ 2.8 (ข) เป็นภาพมอเตอร์สี่เฟส ที่สเตเตอร์มีฟันแปดซี่และโรเตอร์มีฟันหกซี่ โดยทั้งสองภาพนี้มีมุมสเตปเท่ากับ 15 องศา
3) ในการสร้างสเตปปิงมอเตอร์ให้มีโครงสร้างแบบหลายชุดวางซ้อนกัน (Stack) เพื่อเพิ่มประสิทธิภาพในเรื่องของแรงบิดนั้น ชุดแต่ละชุดที่วางซ้อนจะถูกกำหนดเป็นหนึ่งเฟส โดยโรเตอร์และสเตเตอร์จะมีซี่ฟันเหมือนกัน ทำให้เกิดการเพิ่มแรงบิดต่อปริมาตรของโรเตอร์ เป็นการเพิ่มประสิทธิภาพของเครื่องจักร



ภาพที่ 2.6 แสดงเส้นแรงแม่เหล็กที่เกิดขึ้น



ภาพที่ 2.7 แสดงตำแหน่งของสถานะที่สมดุลย์เมื่อเกิดการกระตุ้นที่เฟสใดเฟสหนึ่ง



ภาพที่ 2.8 แสดงโรเตอร์และสเตเตอร์ของมอเตอร์สามเฟสและสี่เฟส ตามลำดับ

โดยที่จำนวนสเตปต่อรอบ (S) หาได้จากสมการ

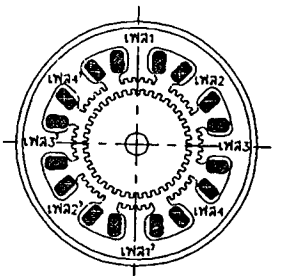
$$S = 360/\phi_m - mN_r \qquad (2\text{-}3)$$

เมื่อ ϕ_m **คือความสัมพันธ์ระหว่างมุมสเตป**

m **คือมุมเฟส**

N_r **คือจำนวนฟันของโรเตอร์**

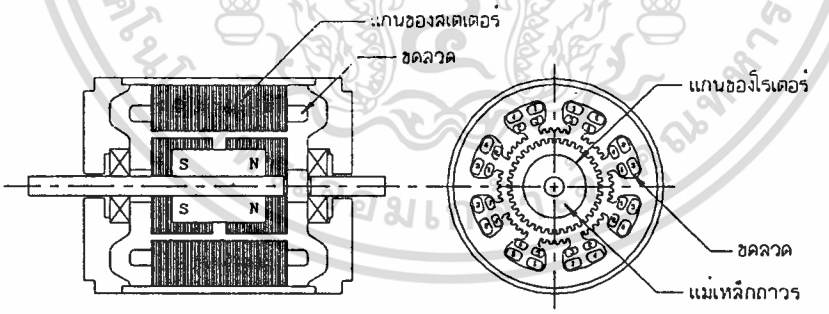
ในการลดขนาดมุมสเตปลง จำนวนซี่ฟันของโรเตอร์จะต้องเพิ่มขึ้น ดังภาพที่ 2.9



ภาพที่ 2.9 แสดงภาพหน้าตัดของมอเตอร์สี่เฟส ที่มีฟันโรเตอร์ห้าสิบซี่ มุมสเตป 1.8 องศา

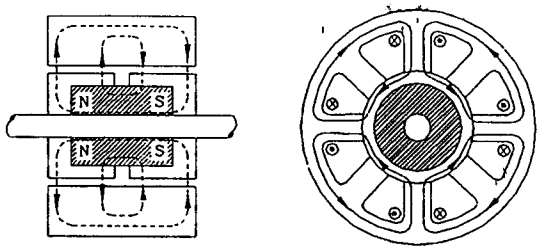
2.2.3 สเตปปิงมอเตอร์แบบไฮบริดจ์ (Hybrid Stepping Motor)

เป็นสเตปปิงมอเตอร์ทที่มีโรเตอร์เป็นแม่เหล็กถาวร การใช้ชื่อว่าไฮบริดจ์ได้มาจากการรวมหลักสำคัญของมอเตอร์แบบแม่เหล็กถาวร และแบบแปรค่าความต้านทานของสนามแม่เหล็กได้เข้าด้วยกัน สเตปปิงมอเตอร์แบบไฮบริดจ์จะมีโครงสร้างของสเตเตอร์คล้ายกับโครงสร้างของสเตปปิงมอเตอร์แบบแปรค่าความต้านทานแม่เหล็กได้ แต่การพันและการต่อขดลวดจะต่างกัน โดยในหนึ่งเฟสของสเตปปิงมอเตอร์แบบแปรค่าความต้านทานแม่เหล็กได้ จะมีขดลวดอยู่สองขด และขดลวดทั้งสองจะพันอยู่ที่ขั้วเดียวกันในลักษณะของสองภาวะขั้ว (Bipolar) ซึ่งจะทำให้ขั้วแม่เหล็กมีความต่างกันขณะที่มีการกระตุ้นแต่ละครั้ง แสดงได้ดังภาพที่ 2.10



ภาพที่ 2.10 แสดงโครงสร้างของสเตปปิงมอเตอร์แบบไฮบริดจ์

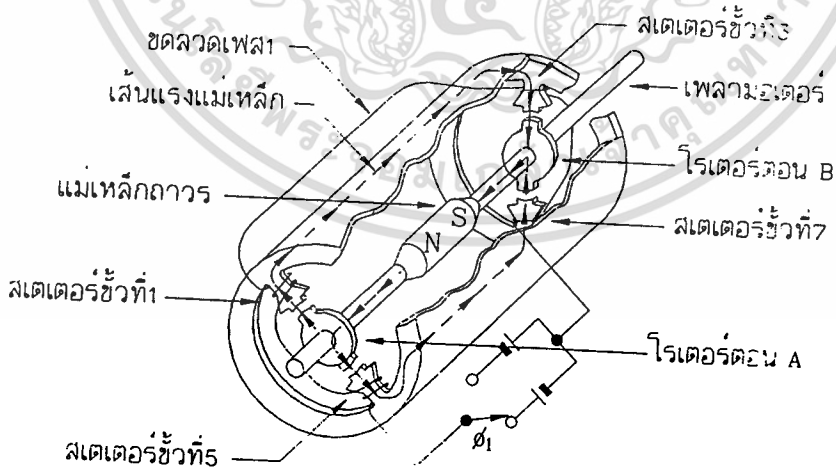
ลักษณะที่สำคัญอีกประการหนึ่งของสเตปปิงมอเตอร์แบบไฮบริดจ์ก็คือ ตัวโรเตอร์จะเป็นแม่เหล็กถาวร รูปร่างทรงกระบอกตามยาวอยู่ในแกนเหล็กของโรเตอร์ เพื่อใช้สร้างสนามแม่เหล็กขั้วเดียว ดังแสดงในภาพที่ 2.11 โดยที่แต่ละขั้วของแม่เหล็กจะถูกล้อมรอบด้วยฟันเหล็กอ่อน ฟันของโรเตอร์กับสเตเตอร์จะอยู่ในตำแหน่งที่เหลื่อมกันอยู่ 90 องศา



(ก) (ข)

ภาพที่ 2.11 แสดงภาพการวางโรเตอร์ตามยาวเพื่อใช้สำหรับสร้างสนามแม่เหล็กขั้วเดียวกัน

หลักการทำงานของสเตปปิงมอเตอร์แบบไฮบริดจ์ ที่แตกต่างจากสเตปปิงมอเตอร์แบบแปรค่าความต้านทานของสนามแม่เหล็กได้ก็คือ แรงบิดที่เกิดขึ้นจากสนามแม่เหล็กจะไม่ขึ้นอยู่กับกระแสที่ไหลผ่านขดลวดเพียงอย่างเดียว แต่จะขึ้นอยู่กับโครงสร้างของซี่ฟันด้วย ซึ่งสามารถแสดงให้เห็นได้ดังภาพที่ 2.12 ข้อดีของสเตปปิงมอเตอร์แบบไฮบริดจ์คือมีขนาดของสเตปเล็ก ทำให้มีความละเอียดค่อนข้างสูง นอกจากนี้ยังมีค่าของแรงบิดสูงกว่าสเตปปิงมอเตอร์แบบแปรค่าความต้านทานของสนามแม่เหล็กได้ แต่สเตปปิงมอเตอร์แบบแปรค่าความต้านทานของสนามแม่เหล็กได้ จะมีแรงเฉื่อยทางกลไกน้อยกว่า



ภาพที่ 2.12 แสดงหลักการทำงานของสเตปปิงมอเตอร์แบบไฮบริดจ์

2.2 สถาปัตยกรรมของไมโครคอนโทรลเลอร์แบบชิพเดี่ยวตระกูล 51

ไมโครคอนโทรลเลอร์แบบชิพเดี่ยว (Single Chip Microcontroller) คือไมโครคอมพิวเตอร์แบบที่มีขนาดเล็กโดยบรรจุไว้ในแผงวงจรรวม (Integrated Circuit) เพียงชิพเดี่ยว เหมาะสำหรับงานควบคุมอุปกรณ์อื่น ๆ แบบอัตโนมัติ เพราะผู้ใช้สามารถเขียนโปรแกรมควบคุมได้ตามต้องการ

MCS-51 ผลิตโดยบริษัท Intel มีการทำงานเป็นแบบ 8 บิท หมายความว่าส่วนที่ทำหน้าที่ในการคำนวณ (ALU) จะทำงานสูงสุดทีละ 8 บิท

MCS-51 มีข้อดีดังต่อไปนี้

1. สามารถนำข้อมูลมา AND,OR หรือทำ COMPLEMENT ทั้งทีละแบบ 8 บิทและ 1 บิท

2. สามารถใช้กับหน่วยความจำสำหรับโปรแกรม (Program Memory) ซึ่งเป็นหน่วยความจำที่ใช้สำหรับเก็บชุดคำสั่งที่จะให้ MCS-51 ทำงาน ได้สูงสุด 64 กิโลไบท์ ทำให้เขียนโปรแกรมควบคุมการทำงานได้มาก

3. สามารถติดต่อกับหน่วยความจำสำหรับข้อมูล (Data Memory) ซึ่งเป็นหน่วยความจำสำหรับเก็บข้อมูลในระหว่างการทำงานของโปรแกรมได้สูงสุด 64 กิโลไบท์

4. ใน 8051 และ 8751 มีหน่วยความจำสำหรับโปรแกรมจำนวน 4 กิโลไบท์ (ใน 8052 และ 8752 มีหน่วยความจำสำหรับโปรแกรมจำนวน 8 กิโลไบท์) อยู่ภายในวงจรรวมทำให้ไม่ต้องต่อหน่วยความจำสำหรับโปรแกรมอยู่ภายนอก ระบบรวมทั้งหมดจึงมีขนาดเล็กและสัญญาณรบกวนจากภายนอกจะทำให้ MCS-51ทำงานผิดพลาดได้ยาก

5. มีพอร์ทแบบขนาน (parallel Port) สำหรับข้อมูลเข้าและออกจำนวน 32 บิท ที่ข้อมูลแต่ละบิทเป็นอิสระต่อกัน

6. มีวงจร Timer/Counter ขนาด 16 บิท 2 ชุด (8052 มี 3 ชุด) ที่ทำงานโหมดต่างๆได้ถึง 4 โหมด

7. มี Universal Asynchronous Receiver Transmitter (UART) สำหรับรับ ส่ง ข้อมูลอนุกรมแบบ Full Duplex ที่สามารถเลือกรูปแบบการรับ-ส่งข้อมูลได้ 4 แบบ

8. มีแหล่งกำเนิดสัญญาณขอขัดจังหวะการทำงานของโปรแกรม (Interrupt Request Signal) 6 แหล่งซึ่งสามารถทำกระโดดไปทำงานตอบสนองการขัดจังหวะ (Interrupt Service Routine) ได้ต่างๆ กัน 5 ตำแหน่ง

9. สามารถเลือกการทำงานให้อยู่ในโหมดของ Idle และ Power Down ซึ่งจะประหยัดการใช้กำลังไฟในการทำงาน

ซึ่งจากข้อดีดังกล่าวทำให้ MCS-51 เป็นที่นิยมนำมาใช้ในการควบคุมระบบอัตโนมัติมาก คุณสมบัติดังกล่าวบรรจุไว้ในวงจรรวมเดียว (Single Chip) ขนาด 40 ขา ดังนั้น จึงสามารถออกแบบให้ระบบทั้งหมดมีขนาดเล็ก และการที่ทั้งหมดบรรจุอยู่ภายในวงจรรวมเดียวจึงทำให้การตรวจสอบหาข้อผิดพลาดในระบบง่ายไม่สลับซับซ้อน รวมทั้งลดปัญหาเรื่องการที่มีสัญญาณรบกวนในระบบจนทำให้การทำงานผิดพลาดไป แต่การที่จะนำเอา MCS-51 มาใช้งานได้จำเป็นต้องศึกษาและทำความเข้าใจถึงโครงสร้างและองค์ประกอบของ MCS-51 เสียก่อนแล้วจึงจะเขียนโปรแกรมเพื่อควบคุมการทำงานของ MCS-51 ให้เป็นไปตามต้องการ

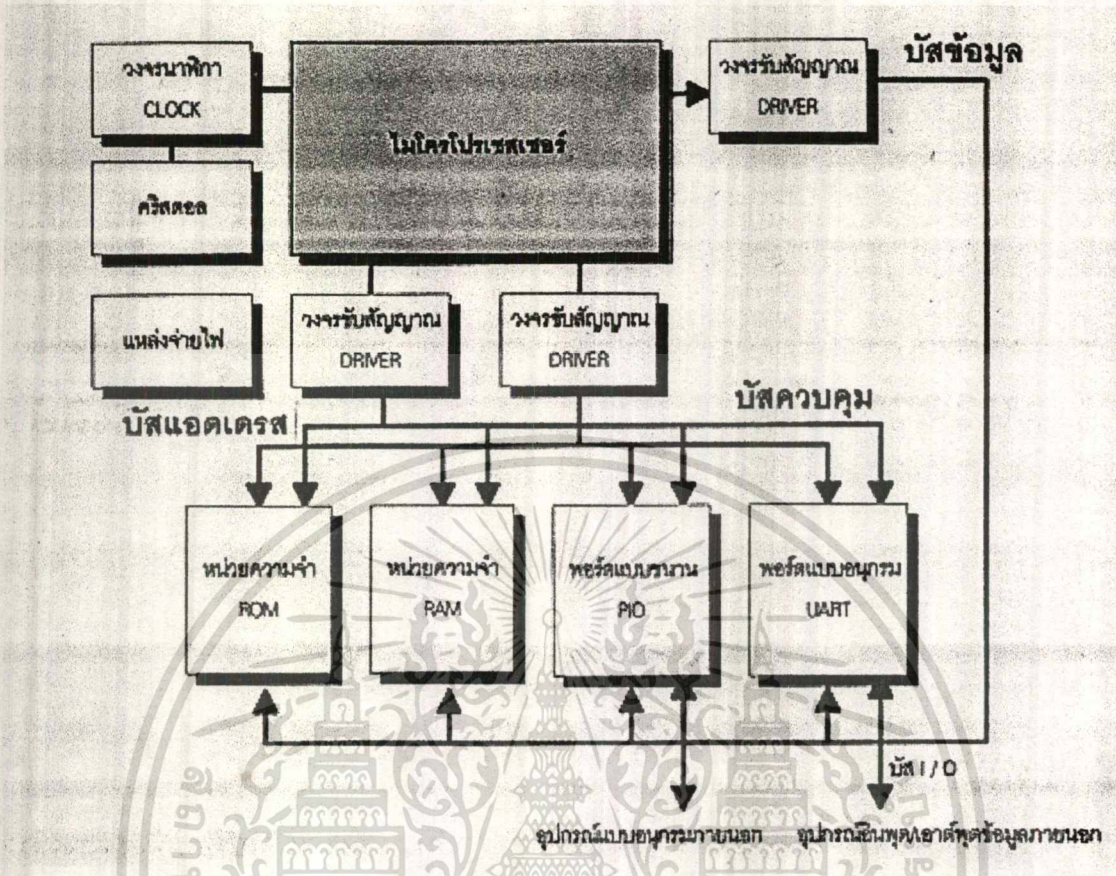
และไมโครคอนโทรเลอร์ตระกูลนี้ ยังมีบางบริษัท ออกแบบและสร้าง ให้มีระบบหน่วยความจำแบบ Flash ทำให้สามารถ โปรแกรมใหม่ได้หลายพันครั้ง และยังสามารถลบด้วยไฟฟ้าได้อีกด้วย ทำให้การออกแบบงาน ทำได้ง่าย ไม่ยุ่งยาก พร้อมทั้งเทคโนโลยีแบบใหม่ ๆ ที่ช่วยให้สามารถลดพลังงานที่ต้องใช้ลงไปมาก ทำให้ระบบมีความร้อนต่ำ มีสเถียรภาพในวงจรสูง

ไมโครคอนโทรเลอร์ที่ออกแบบมาใหม่ ยังมีรุ่นที่ใช้แรงดันในการทำงานต่ำ ทำให้สามารถ ใช้งานได้ด้วยแรงดันเพียง 2.7 V เท่านั้น ส่วนความถี่ในการทำงานก็สามารถทำงานในความถี่ที่สูงขึ้น โดยทำได้ถึง 24 MHz เลยทีเดียว

แต่ไมโครคอนโทรเลอร์ตระกูลนี้ก็มีจุดด้อยอยู่อย่างหนึ่งก็คือเมื่อทำการต่อหน่วยความจำจากภายนอกแล้ว จะต้องเสียพอร์ตไปถึง 3 พอร์ตเลยทีเดียว ทำให้การใช้งานทำได้ยากขึ้นเพราะจำนวน พอร์ตลดลง

โครงสร้างของ 8051

ภายใน 8051 จะประกอบขึ้นด้วย GATE ต่างๆ เช่น AND,OR,NOT ซึ่ง GATE เหล่านี้จะถูกนำเอามาออกแบบให้มีหน้าที่การทำงานต่างๆ เช่นวงจรถอดรหัสคำสั่ง วงจรสร้างสัญญาณนาฬิกา โครงสร้างภายในของ 8051 จะประกอบด้วยส่วนย่อยๆ ดังไดอะแกรมในรูปที่ 2.13

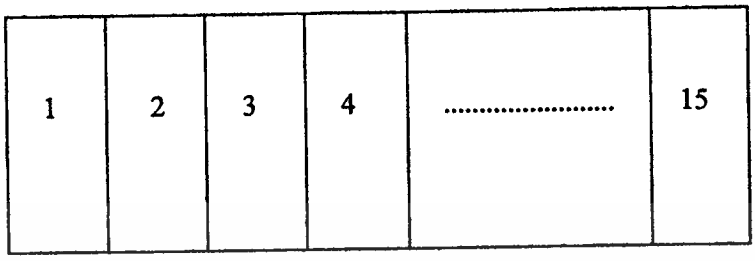


รูปที่ 2.13 ไดอะแกรมโครงสร้างของ 8051

ไดอะแกรมในรูปที่ 2.13 นี้เป็นโครงสร้างใหญ่ๆ ของ 8051 เป็นคอมพิวเตอร์จึงประกอบด้วยส่วนหลักๆ 3 ส่วนคือ

ส่วนที่ 1 คือ CPU (Central Processing Unit) หรือตัวประมวลผล ส่วนนี้จะมีวงจรที่ทำหน้าที่สร้างสัญญาณควบคุมในการติดต่อกับส่วนอื่นๆ เรียกว่าวงจรควบคุม (Control Unit) สัญญาณที่สร้างจากวงจรควบคุมได้แก่สัญญาณสำหรับการติดต่อกับหน่วยความจำ, อุปกรณ์ รับข้อมูลหรือส่งข้อมูลออกจากตัว 8051 ซึ่งส่วนควบคุมการขัดจังหวะ (Interrupt Control) และส่วนควบคุมบัส (Bus Control) ก็เป็นส่วนหนึ่งของวงจรควบคุมด้วย การสร้างสัญญาณควบคุมจากส่วน CPU นี้จะทำการสร้างสัญญาณโดยการถอดรหัสจากคำสั่งตามที่มีการกำหนดไว้ และสัญญาณที่สร้างขึ้นมาจะอ้างอิงกับสัญญาณนาฬิกาที่สร้างจากวงจรออสซิลเลเตอร์เพื่อให้ทุกๆ ส่วนในวงจรทำงานประสานกัน (Synchronize) อย่างถูกต้อง ใน CPU นี้ยังประกอบด้วยส่วนย่อยอีกส่วนที่เรียกว่าส่วนประมวลผล (ALU) ส่วนนี้จะทำหน้าที่ประมวลผลข้อมูลเช่น การบวก,ลบ,คูณ หรือหารข้อมูลแล้วนำผลลัพธ์ไปเก็บไว้ใน Register หรือหน่วยความจำที่ต้องการ

ส่วนที่ 2 คือหน่วยความจำ (Memory) มีไว้สำหรับจดจำข้อมูล ถ้าจะให้เห็นภาพพจน์ของหน่วยความจำได้ดีก็คือ หน่วยความจำเปรียบเหมือนกล่องเก็บเอกสารจำนวนมากที่นำมาต่อเรียงกันไว้ แต่ละกล่องมีเอกสาร 1 แผ่น ดังในรูปข้างล่าง มีกล่องเอกสารทั้งหมด 15 กล่อง



ถ้าต้องการเอกสารจากกล่องใดหรือเอาเอกสารไปเก็บที่กล่องใด จะต้องรู้ หมายเลขของกล่องข้อมูลเสียก่อนซึ่งถ้าเป็นหน่วยความจำแล้วหมายเลขกล่องก็คือตำแหน่งของหน่วยความจำหรือแอดเดรสนั่นเอง การเอาข้อมูลไปเก็บในหน่วยความจำเรียกว่าการเขียนข้อมูล และการเอาข้อมูลออกจากหน่วยความจำจะเรียกว่าการอ่านข้อมูล ซึ่งแต่ละตำแหน่งของหน่วยความจำจะเก็บข้อมูลได้เพียงค่าเดียวเท่านั้น ในไมโครโปรเซสเซอร์ทั่วไปรวมทั้ง 8051 นั้นข้อมูลในแต่ละตำแหน่งของหน่วยความจำจะมีค่าได้เพียง 8 หลักของเลขฐาน 2 ดังนั้นแต่ละตำแหน่งของหน่วยความจำจะเก็บข้อมูลมีค่าได้ระหว่าง 0 ถึง 255 (00000000 ถึง 11111111 ในเลขฐาน 2) แต่จำนวนตำแหน่งที่จะเก็บข้อมูลได้ขึ้นกับไมโครโปรเซสเซอร์แต่ละเบอร์ การติดต่อกับหน่วยความจำจะต้องมีสัญญาณ 3 กลุ่มคือ

1. แอดเดรสหรือค่าตำแหน่งที่ต้องการติดต่อกับหน่วยความจำ ใน 8051 จะติดต่อกับหน่วยความจำประเภท Program Memory หรือ Data Memory ได้สูงสุดชนิดละ 65536 ตำแหน่ง ดังนั้น การอ้างอิงแต่ละตำแหน่งของหน่วยความจำจะต้องใช้เส้นแสดงตำแหน่งในเลขฐาน 2 ทั้งหมด 16 เส้น
2. ข้อมูลที่จะอ่านหรือเขียนกับหน่วยความจำที่ตำแหน่งในข้อ 1
3. สัญญาณควบคุมที่จะส่งไปยังหน่วยความจำ เพื่อบอกกับหน่วยความจำว่าต้องการอ่านหรือเขียนข้อมูล สัญญาณเหล่านี้จะถูกวงจรควบคุมภายใน 8051 สร้างมาจากวงจรถอดรหัสของคำสั่งที่ 8051 อ่านจากหน่วยความจำ Program Memory เข้าไปทำงานนั่นเอง

ส่วนที่ 3 อุปกรณ์อินพุทและเอาท์พุท เป็นส่วนที่จะใช้ส่งข้อมูลเข้าหรือออกจาก 8051 ทำให้ 8051 ติดต่อกับภายนอกได้ ดังในรูปที่ 1 อุปกรณ์อินพุทเอาท์พุทได้แก่ 4 I/O Port,Timer0,Timer1,Serial Port การทำงานของแต่ละส่วนมีดังนี้

1. 4 I/O Port คำว่าพอร์ทหมายถึงจุดที่จะติดต่อกับส่วนที่อยู่ภายนอก 4 I/O Port ของ 8051 เป็นที่ใช้สำหรับรับ - ส่งข้อมูลซึ่งเป็นสัญญาณดิจิตอลเข้าหรือออกจากตัว MCS-51 พอร์ทมีทั้งหมด

4 พอร์ท โดยแต่ละพอร์ทจะรับส่งข้อมูลได้ 8 บิท มีพอร์ท P0,P1,P2 และ P3 บางพอร์ทจะทำงานมากกว่า 1 อย่างก็ได้ เช่นพอร์ท P0 และ P2 จะใช้สำหรับการส่งค่าตำแหน่ง (Address) ของหน่วยความจำที่ต้องการติดต่อและพอร์ท P0 จะใช้รับส่งข้อมูลเมื่อติดต่อกับหน่วยความจำได้ด้วยแต่สิ่งเหล่านี้ไม่ได้เกิดขึ้นในเวลาเดียวกัน แต่จะทำงานตามลำดับโดยควบคุมจากสัญญาณควบคุมที่ถอดรหัสมาจากแต่ละคำสั่งที่ให้คอมพิวเตอร์ทำงานนั่นเอง และสัญญาณทั้งหมดจะอ้างอิงกับ สัญญาณนาฬิกา

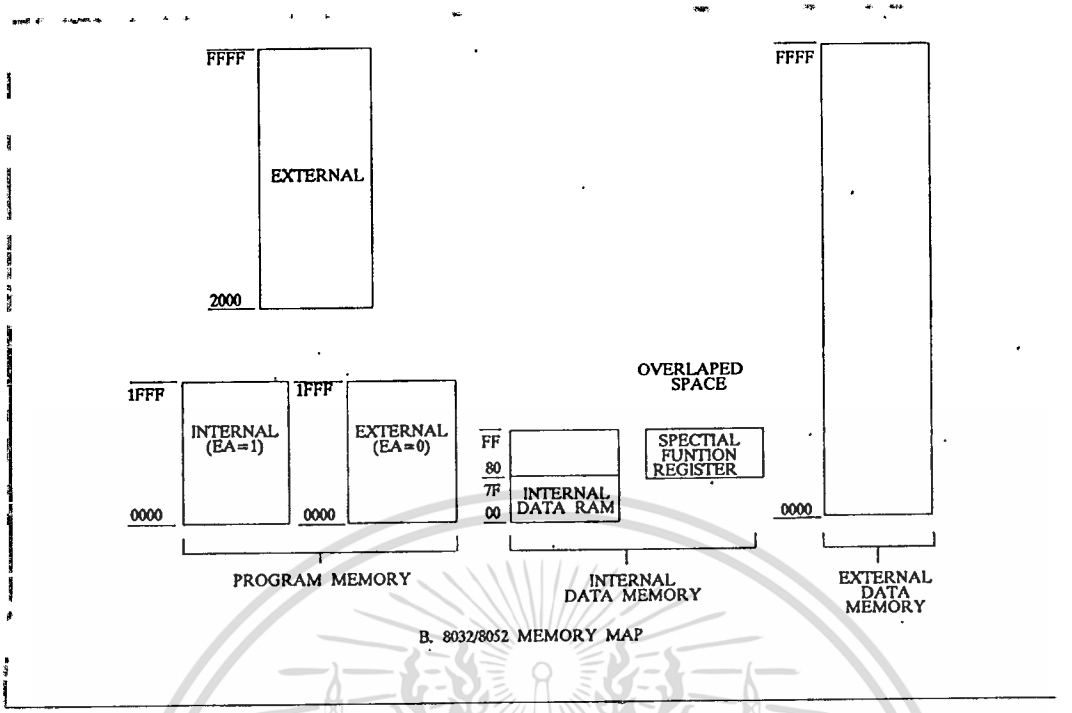
2. Timer0 และ Timer1 เป็นวงจรนับที่สามารถกำหนดให้ทำการนับจำนวนไซเคิลของสัญญาณที่ต่อจากภายนอก 8051 หรือจำนวนไซเคิลของสัญญาณนาฬิกาภายใน 8051 ก็ได้ค่าจากการนับจะถูกอ่านหรือตั้งค่าเริ่มต้นของการนับได้โดย CPU

3. Serial Port หรือพอร์ทอนุกรม CPU จะอ่านและเขียนข้อมูลกับ Serial Port เป็นแบบ 8 บิท แต่ข้อมูลจะถูกส่งออกจาก 8051 เรียงไปทีละบิทออกจากขา TXD และในการรับข้อมูลเข้าก็จะรับเข้ามาทีละบิททางขา RXD แล้วจัดเรียงใหม่เป็น 8 บิทเพื่อให้ CPU อ่านไปใช้งานต่อไป

การจัดการหน่วยความจำของ 8051

หน่วยความจำของ 8051 แบ่งออกไว้เป็น 2 แบบตามลักษณะการใช้งานคือ

1. **Program Memory** เป็นหน่วยความจำที่ใช้เก็บคำสั่งในรูปรหัสภาษาเครื่อง (Machine Language) ซึ่งต้องการให้ 8051 ทำงาน เมื่อ 8051 ทำงานก็จะอ่านข้อมูลที่เก็บในหน่วยความจำประเภทนี้เข้าไปถอดรหัสแล้วสร้างสัญญาณควบคุมส่วนอื่นๆ ตามการทำงานของแต่ละคำสั่งนั้น หน่วยความจำแบบนี้ต้องเป็นแบบ Read Only Memory และผู้ใช้ต้องเขียนข้อมูลในแต่ละตำแหน่งของหน่วยความจำเป็นรหัสภาษาเครื่องของ 8051 ตามลำดับการทำงานที่ต้องการ (หน่วยความจำแบบ ROM เป็นแบบ Non volatile ซึ่งเมื่อปิดไฟแล้วข้อมูลก็ไม่มีการสูญหาย) การเขียนข้อมูลลงไปบน ROM จะต้องใช้เครื่องมือพิเศษ ในระหว่างการทำงานของ 8051 ผู้ใช้ไม่สามารถใช้คำสั่งทำการเขียนข้อมูลลงในหน่วยความจำแบบนี้ได้ จำนวนตำแหน่งสูงสุดของหน่วยความจำแบบนี้ที่ 8051 จะใช้งานได้คือ 65536 ตำแหน่ง ค่าของตำแหน่ง (Address) จะเขียนเป็นเลขฐาน 16 ได้ตั้งแต่ 0000H ถึง FFFFH หน่วยความจำตำแหน่ง 0000H ถึง 0FFFH จำนวน 4 กิโลไบท์นั้น ผู้ใช้จะเลือกได้ว่าเป็นตำแหน่งของ ROM ที่อยู่ภายในหรือภายนอก8051 (ไมโครคอนโทรลเลอร์เบอร์อื่นๆ เช่น 8052 จะมีขนาดของ ROM ส่วนนี้ได้ถึง 8 กิโลไบท์ ตำแหน่ง 0000H ถึง1FFFH) ถ้าต้องการให้ 8051 ทำงานตามคำสั่งที่เก็บไว้ใน ROM ภายใน 8051 ก็ให้ป้อนสัญญาณสภาวะลอจิก High(1) เข้าที่ขาของ EA ของ 8051 แต่ถ้าต้องการให้ทำงานในโปรแกรมที่เก็บไว้ใน ROM ภายนอก 8051 ก็ให้ต่อลอจิก Low(0) เข้าที่ขาของ EA ของ 8051 ส่วนหน่วยความจำที่ตำแหน่ง 1FFFH ถึง FFFFH จะต้องอยู่ภายนอก 8051 เสมอดังแสดงในแผนภูมิหน่วยความจำ(Memory Map) ในรูปที่ 2.14



รูปที่ 2.14 แผนภูมิหน่วยความจำของ 8051

Internal Memory หมายถึงหน่วยความจำนั้นอยู่ภายใน 8051 ส่วน External Memory หมายถึงหน่วยความจำนั้นอยู่ภายนอก 8051

ไมโครคอนโทรลเลอร์เบอร์ 8031, 8051 และ 8751 นั้นโดยโครงสร้างและรหัสคำสั่งจะเหมือนกันทุกประการแตกต่างกันที่

1. 8031 ไม่มี ROM ขนาด 4 กิโลไบท์อยู่ภายใน ผู้ใช้จะต้องเลือกการใช้งาน Program Memory อยู่ภายนอกวงจรรวมทั้งหมด 64 กิโลไบท์

2. 8051 จะมี ROM ขนาด 4 กิโลไบท์อยู่ภายใน ถ้าต้องการเก็บคำสั่งควบคุมการทำงานไว้ในหน่วยความจำส่วนนี้ จะต้องส่งโปรแกรมคำสั่งไปให้โรงงานผู้ผลิตทำการเขียนใส่ ROM ให้ ตั้งแต่ในขั้นตอนการผลิตวงจรรวม ผู้ใช้ไม่สามารถแก้ไขโปรแกรมได้เอง ถ้าจะนำมาใช้งานโดยเก็บโปรแกรมไว้ในหน่วยความจำช่วง 4 กิโลไบท์แรกอยู่ภายนอกก็สามารถทำได้ โดยการต่อ ROM ไว้ภายนอก แล้วต่อขา EA ของ 8051 ไว้กับสัญญาณที่มีสภาวะลอจิกเป็น 0

3. 8751 จะมีหน่วยความจำขนาด 4 กิโลไบท์เป็นแบบ EPROM (Erasable Program Read Only Memory) อยู่ภายในวงจรรวมเอาไว้ ใช้เก็บโปรแกรมคำสั่งที่จะให้ 8751 ทำงาน ผู้ใช้สามารถเขียนคำสั่งลงใน EPROM ได้เองโดยใช้เครื่องมือที่เรียกว่าเครื่องโปรแกรม EPROM (Eprom Programmer) และผู้ใช้สามารถแก้ไขโปรแกรมที่อยู่ใน EPROM ได้โดยการล้างข้อมูลในทุกตำแหน่งของ EPROM ออกด้วยการฉายแสงอุลตร้าไวโอเลตผ่านกระจกใสบนวงจรรวมเข้าไปยังวง

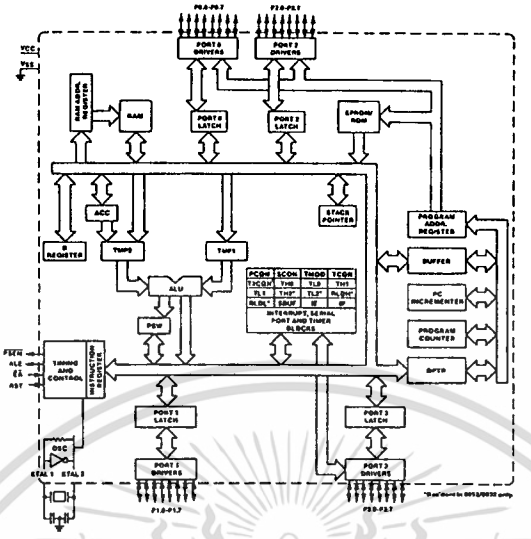
จรภายใน ตามเวลาที่กำหนดในคู่มือเฉพาะ (Data Sheet) ของ 8751 จากนั้นก็ใช้เครื่องโปรแกรม EPROM เขียนโปรแกรมลงไปใหม่ 8751 นี้สะดวกมากสำหรับการพัฒนาโปรแกรม

2. Data Memory เป็นหน่วยความจำที่ 8051 จะใช้สำหรับพัก เก็บข้อมูล แล้วเรียกมาใช้ใหม่ในระหว่างการทำงานของ 8051 การอ่านหรือเขียนข้อมูลจากหน่วยความจำจะกระทำโดยคำสั่งที่เก็บไว้ใน Program Memory หน่วยความจำแบบนี้เป็นประเภท Random Access Memory (RAM) ถ้ามีไฟเลี้ยงอยู่ข้อมูลที่เก็บไว้จะไม่สูญหาย แต่ถ้าปิดเครื่องหรือไม่จ่ายไฟให้แก่ RAM แล้ว ข้อมูลใน RAM ก็จะสูญหายไป การสูญหายของข้อมูลไม่ได้หมายความว่าไม่มีอะไรอยู่เลยแต่เป็นการที่มีข้อมูลใหม่ซึ่งไม่ใช่ข้อมูลที่เก็บไว้เดิมเข้ามาอยู่แทนที่ เช่นเดิมเก็บข้อมูล 18H ไว้ที่ตำแหน่ง 1900H เมื่อปิดไฟแล้วเปิดใหม่ ข้อมูลที่ตำแหน่ง 1900H จะไม่ใช่ 18H อาจเป็นค่าอะไรก็ได้ ซึ่งเรียกการเกิดลักษณะแบบนี้ว่าข้อมูลสูญหายไป หน่วยความจำแบบ Data Memory ของ 8051 จะมีอยู่ 2 ชุด ชุดหนึ่งอยู่ภายใน 8051 จำนวน 128 ไบท์ที่ตำแหน่ง 00H ถึง 7FH (เบอร์ 8052 จะมี 256 ไบท์อยู่ที่ตำแหน่ง 00H ถึง FFH) และอีกชุดหนึ่งจะต้องต่ออยู่กับภายนอกของวงจรรวม 8051 มีได้สูงสุด 65536 ไบท์ (64 กิโลไบท์) อยู่ที่ตำแหน่ง 0000H ถึง FFFFH ดังแสดงในรูปที่ 3 หน่วยความจำแบบ Data Memory ภายใน 8051 ที่ตำแหน่ง 80H ถึง FFH.นั้นไม่ได้มีอยู่ทุกตำแหน่ง จะมีเฉพาะในบางตำแหน่งซึ่งเรียกหน่วยความจำบางตำแหน่งนี้ว่า Special Function Register (SFR) เพราะจะใช้หน่วยความจำเหล่านี้สำหรับงานพิเศษเท่านั้น แต่ละตำแหน่งของหน่วยความจำแบบ SFR นี้อาจเป็น RAM หรือวงจรนับ (Counter) วงจรตั้งเวลา (Timer) ก็ได้เช่นเป็น Timer0,Timer1 ดังนั้นใน 8051 จึงไม่ถือว่า SFR เป็น Data Memory ถ้าเป็น 8052 ซึ่งมี Data Memory ขนาด 256 ไบท์ จะใช้บางตำแหน่งของหน่วยความจำช่วงตำแหน่ง 80H ถึง FFH เป็น SFR ส่วนตำแหน่งอื่นที่เหลือก็เป็น RAM เหมือนกับหน่วยความจำ ช่วง 00H ถึง 7FH นั่นเอง

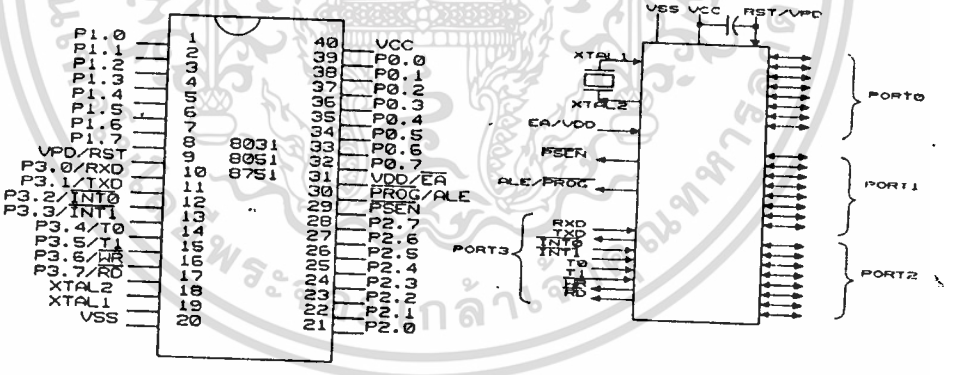
สถาปัตยกรรมของ 8051

ในส่วนที่ผ่านมาได้กล่าวถึงไดอะแกรมภายในของ 8051 อย่างกว้างๆ ซึ่งพอจะบอกได้โดยสังเขปว่า ประกอบด้วยส่วนใหญ่ๆ อะไรบ้าง ในรูปที่ 2.15 เป็นสถาปัตยกรรมภายในของ 8051

ซึ่งจะอธิบายถึงส่วนย่อยๆ ของภายใน 8051 เพียงชิพเดียว และสัญญาณจากภายใน จะต่อออกสู่ภายนอกทางขา (Pin) ของ 8051 ที่มีอยู่ 40 ขา ดังรูปที่ 2.16



รูปที่ 2.15 สถาปัตยกรรมภายในของ 8051



รูปที่ 2.16 ไดอะแกรมขาของ 8051 แบบ DIP

8051 ไมโครคอนโทรลเลอร์ที่บรรจุอยู่ในวงจรรวมแบบ Dual Inline Package (DIP) ซึ่งแต่ละข้างของ 80581 มีขาอยู่ข้างละ 20 ขารวมทั้งหมด 40 ขานั้นจะใช้งานต่างๆ กันดังนี้

Vcc

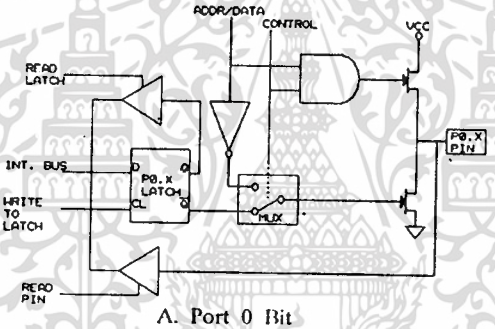
ขา 40 เป็นขาที่ต้องป้อนไฟเลี้ยง +5 โวลท์เข้าไปเพื่อให้วงจรรวมทำงานได้ ระดับโวลเตจของลอจิก 0 และ 1 ของ 8051 จึงต่อเข้ากับอุปกรณ์ลอจิกแบบ TTL โดยตรง

Vss

ขา 20 เป็นขาที่ต้องต่อกับกราวด์(Ground) ของแหล่งจ่ายไฟ การต่ออุปกรณ์ทั้งหมดจะต้องมีกราวด์ของอุปกรณ์ต่อเข้าด้วนกัน

Port 0

เป็นพอร์ทขนานขนาด 8 บิท อยู่ที่ขา 39 ถึง 32 เริ่มจากบิท 0 ถึงบิท 7 ตามลำดับดังในรูปที่ 5 แต่ละขาจะเขียนว่า P0.0,P0.1,...,P0.7 นั้น P0.7 หมายถึงบิท 7 ของพอร์ท 0 ซึ่งเป็นบิทที่มีนัยสำคัญสูงสุด (Most Significant) และ P0.0 คือบิท 0 ของพอร์ท 0 เป็นบิทที่มีนัยสำคัญต่ำสุด (Least Significant) พอร์ท 0 นี้ใช้ได้ทั้งการรับ-ส่งตำแหน่งและข้อมูลกับหน่วยความจำหรือใช้เป็นพอร์ทรับ-ส่งข้อมูลก็ได้ ข้อมูลที่ส่งออกทางพอร์ท 0 จะถูก Latch ไว้ที่ขาของพอร์ท โครงสร้างของแต่ละบิทของพอร์ท 0 เป็นแบบ Open Drain Bidirectional ดังรูปที่ 2.17



A. Port 0 Bit

รูปที่ 2.17 โครงสร้างของพอร์ท 0

ในรูปที่ 2.17 เมื่อเปรียบเทียบกับรูปที่ 2.15 ส่วนที่ 1 ของรูปที่ 2.17 ก็คือ Port 0 Driver ของรูปที่ 2.15 นั่นเอง จากโครงสร้างในรูปที่ 2.17 เมื่อมีคำสั่งการเขียนข้อมูลมายังพอร์ท 0 ข้อมูลจาก Internal Data Bus จะถูก Latch ไว้ที่ D-FF โดยสัญญาณ "Write to latch" ที่ถูกสร้างมาจากส่วน Timing and Control และในการอ่านข้อมูลจากพอร์ท 0 จะอ่านได้ 2 แบบคือการอ่านข้อมูลที่ส่งไปเก็บไว้ที่พอร์ทก็จะมีสัญญาณ Read Latch มาเพื่ออ่านข้อมูลจาก D-FF กลับเขาไปยัง Internal Data Bus การอ่านข้อมูลอีกแบบก็คือการอ่านสภาวะของสัญญาณที่เข้ามาทางพอร์ท 0 ก็จะมีสัญญาณ Read Pin มาควบคุมการอ่าน พอร์ท 0 จะใช้งานหลายอย่างดังนี้

1.ใช้สำหรับส่งค่าตำแหน่งหน่วยความจำภายนอกที่ต้องการติดต่อด้วย ตำแหน่งหน่วยความจำสูงสุดที่จะติดต่อได้ก็คือ 64 กิโลไบท์ จึงมีค่าตำแหน่งหน่วยความจำ 16 บิทของเลขฐาน 2

ค่าตำแหน่งหน่วยความจำ 8 บิทล่างจะถูกส่งออกทางพอร์ท 0 และ 8บิทบนจะส่งออกไปทางพอร์ท 2

2. ใช้รับ-ส่งข้อมูลกับ Data Memory หรือใช้รับข้อมูลจาก Program Memory

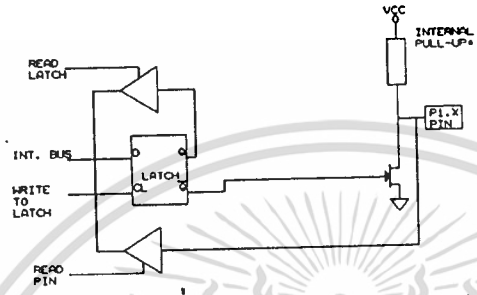
3.ใช้รับ-ส่งข้อมูลผ่านทางพอร์ทโดยตรง ในกรณีที่ไม่มีการใช้หน่วยความจำของ Program Memory หรือData Memory ภายนอก

วงจรภายในส่วน Timing and Control จะเป็นตัวสร้างสัญญาณมาควบคุมวงจรในรูปที่ 6 เพื่อให้การทำงานแต่ละอย่างข้างต้น เมื่อแต่ละบิทของพอร์ท 0 ทำงานตามข้อ 1 และ 2 ข้างต้น วงจร Timing and Control จะทำให้สภาวะลอจิกของขา Control เป็น 1 ซึ่งทำให้สวิทช์ MUX อยู่ในตำแหน่งข้างบน เมื่อพอร์ท 0 จะส่งข้อมูลซึ่งเป็นค่าตำแหน่งหน่วยความจำหรือข้อมูลที่จะเขียนออกไปยังหน่วยความจำภายนอกก็จะส่งค่าดังกล่าวมายัง ADDR/DATA ถ้าข้อมูลที่ส่งมาเป็น 1 จะทำให้สัญญาณออกจาก AND GATE เป็น 1 และสัญญาณที่ออกจาก Inverter เป็น 0 ดังนั้น FET ตัวบน ON (สภาวะ ON ของ FET คือความต้านทานระหว่างขา D กับ S ต่ำมากเสมือนเป็นวงจรปิด) ส่วน FET ตัวล่าง OFF (สภาวะ OFF ของ FET คือความต้านทานระหว่างขา D กับ S มีค่าสูงมากเสมือนเป็นวงจรเปิด) สภาวะลอจิกที่ขา P0.X PIN จะเป็น 1 แต่ถ้าข้อมูลที่ส่งออกมายัง ADDR/DATA เป็น 0 ก็จะทำให้สัญญาณจาก AND GATE เป็น 0 และสัญญาณที่ออกจาก Inverter เป็น 1 ดังนั้น FET ตัวบนจะ OFF ส่วน FET ตัวล่างจะ ON ทำให้สภาวะลอจิกที่ขา P0.X PIN เป็น 0 เมื่อ 8051 ต้องการใช้พอร์ท 0 สำหรับการอ่านข้อมูลจากหน่วยความจำภายนอก หรือใช้ทำงานใน 3 ข้อข้างบน ก็จะทำได้โดยวงจร Timing and Control ทำให้สภาวะลอจิกของสัญญาณ Control ในรูปเป็น 0 ทำให้เอาท์พุทจาก AND GATE เป็น 0 FET ตัวบนจะ OFF และสวิทช์ MUX จะอยู่ในตำแหน่งข้างล่าง ดังนั้น FET ตัวข้างล่างจะ ON หรือ OFF ก็แล้วแต่ข้อมูที่ขา Q ของ D-FF เมื่อมีการเขียนข้อมูลจาก Internal Data Bus มายัง D-FF ก็จะมีสัญญาณ Write to Latch มายัง D-FF ด้วย ถ้าข้อมูลที่เขียนมาเป็น 1 ก็จะทำให้ขา Q มีสภาวะลอจิกเป็น 0 ทำให้ FET ตัวล่าง OFF ดังนั้นขา P0.X จะอยู่ในสภาวะอิมพิแดนซ์สูง (High Impedance) เพราะ FET ทั้ง 2 ตัว OFF แต่ถ้าข้อมูลที่เขียนมายัง D-FF เป็น 0 จะทำให้ FET ตัวล่าง ON แต่ตัวบน OFF ทำให้สภาวะลอจิกที่ขา P0.X เป็น 1 ดังนั้น PORT 0 เมื่อให้ทำงานเป็นพอร์ทส่งข้อมูล (ไม่ใช่ส่งตำแหน่งหน่วยความจำ) จะไม่สามารถแสดงสภาวะลอจิก 1 ได้จึงต้องต่อตัวต้านทาน Pull Up ไว้ภายนอก ระหว่างขา P0.X กับไฟเลี้ยงวงจร ถ้าใช้พอร์ท 0 สำหรับรับข้อมูลเข้าจะต้องเขียน 1 มาเก็บไว้ยัง D-FF เสียก่อนเพื่อให้ขา P0.X อยู่ในสภาวะ High Impedance แล้วจึงใช้คำสั่งอ่านสภาวะลอจิกเข้าไปยัง Internal Data Bus ต่อไป โดยคำสั่งอ่านสภาวะลอจิกทางพอร์ท 0 ก็จะทำให้วงจร Timing and Control สร้างสัญญาณ Read Pin สำหรับการอ่านสภาวะลอจิกข้างต้น ถ้าไม่เขียน 1 มาเก็บไว้ยัง D-FF ก่อนที่จะอ่านข้อมูลแล้วอาจมีข้อมูลค้างอยู่ที่ D-FF ทำให้ Q เป็น 0 และ Q เป็น 1 ซึ่งทำให้ FET ตัวล่าง ON สัญญาณที่ต่อเข้ามาที่ขา P0.X ไม่ว่าจะมีสภาวะลอจิกใดจะถูกดึงลงกราวด์ ดังนั้นเมื่ออ่านข้อมูลเข้าไปก็พบว่าเป็น 0 เสมอ ในการอ่าน

ข้อมูลจากหน่วยความจำภายนอกนั้นวงจร Timing and Control ก็จะเขียนข้อมูลมายัง D-FF ให้เป็น 1 และสร้างสัญญาณ Control ให้มีลอจิกเป็น 0 ก่อนจะอ่านข้อมูลเข้าไปด้วย

Port 1

เป็นพอร์ทขนานขนาด 8 บิท ในรูปที่ 2.16 คือขา P1.0 ถึง P1.7 (ขา 1-8) P1.0 หมายถึงบิท 0 ของพอร์ท 1 ซึ่งเป็นบิท Least Significant Bit และบิท P1.7 หมายถึงบิทที่ 7 ของพอร์ท ๆ ซึ่งเป็นบิท Most Significant Bit โครงสร้างของพอร์ท 1 แต่ละบิทมีดังรูปที่ 2.18

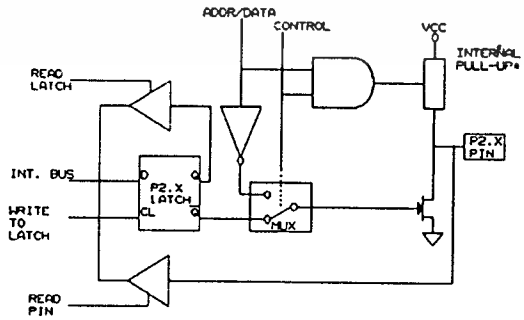


รูปที่ 2.18 โครงสร้างของพอร์ท 1

ส่วนที่ 1 คือ Port 1 Latch ในรูปที่ 2.15 ซึ่งจะมีการทำงานเหมือนส่วนที่ 1 ของพอร์ท 0 ในรูปที่ 6 ส่วนที่ 2 คือ Port 1 Driver ในรูปที่ 2.15 Port 1 Driver นี้จะมีตัวต้านทานต่ออยู่เป็น Internal Pull Up พอร์ท 1 นี้จะใช้ทำหน้าที่เป็นตัวรับ-ส่งข้อมูลเท่านั้นข้อมูลที่ส่งออกทางพอร์ท 1 จะถูก Latch ไว้ก่อนแล้วจึงส่งออกไปทางแต่ละขา ก่อนที่จะอ่านข้อมูลเข้าไปทางพอร์ท 1 จะต้องเขียน 1 ไปยังทุกบิทของพอร์ท 1 เสียก่อนเพื่อให้ FET อยู่ในสภาวะ OFF ก่อน มิฉะนั้นแล้วถ้ามีข้อมูล 0 ส่งออกมาค้างอยู่ที่ D-FF จะทำให้ FET อยู่ในสภาวะ ON ดังนั้นถ้าสัญญาณภายนอกส่งเข้ามาที่ขานี้ก็จะถูกลัดวงจรลงกราวด์ โดยไม่สนใจว่าสภาวะลอจิกของสัญญาณที่เข้ามาจะเป็นอะไร ข้อมูลที่อ่านเข้าไปจึงเป็น 0 เสมอ

Port 2

พอร์ทขนานขนาด 8 บิท คือขา P2.0 ถึง P2.7 (บิท 0 ถึงบิท 7 ของ พอร์ท 2) ในรูปที่ 2.16 โครงสร้างของพอร์ท 2 แต่ละบิทจะมีดังรูปที่ 2.19



รูปที่ 2.19 โครงสร้างของพอร์ท 2

ลักษณะโครงสร้างจะเหมือนกับ Port 0 แตกต่างกันที่ใน Port 2 นั้นภาค Driver จะใช้งานเพียง 2 ลักษณะคือ

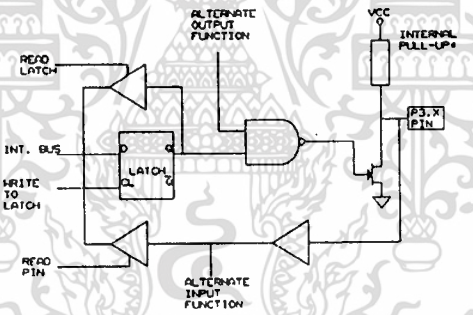
1. ใช้ส่งค่าตำแหน่งหน่วยความจำภายนอก ที่ต้องการติดต่อ ค่าตำแหน่งนี้เป็น 8 บิทบนของค่าตำแหน่ง

2. ใช้เป็นพอร์ทรับและส่งข้อมูลจากภายนอก

ดังนั้นภาค Driver ของพอร์ท 2 จึงแตกต่างจาก Driver ของพอร์ท 0 โดยที่ในพอร์ท 2 นั้นจะมีเฉพาะ ADDR (ตำแหน่งหน่วยความจำ) เข้ามาที่ MUX (Multiplexer) เท่านั้น นอกนั้นแล้วการทำงานจะเหมือนกันและที่เอาท์พุทของพอร์ท 2 จะมี Internal pull up ซึ่งเป็นตัวต้านทานและจะทำให้เอาท์พุทของพอร์ท 2 แสดงสถานะเป็นลอจิก 1 ได้ ถ้า FET อยู่ในสภาวะ OFF บางครั้งเรียกว่า " Quasi-bidirectional" เมื่อใช้เป็นพอร์ทอินพุทก็สามารถทำได้โดยการต่อสัญญาณภายนอกเข้ามาโดยตรง ถ้าสัญญาณภายนอกเป็น 0 ก็จะมีกระแสไหลออกจากพอร์ท (Source Current) ในการที่จะใช้พอร์ทนี้เป็นพอร์ทรับข้อมูลเข้า จะต้องเขียน 1 ไปยังแต่ละบิทของพอร์ทเสียก่อน ดังได้อธิบายในเรื่อง Port 1 และ Port 0

Port 3

คือขา P3.0 ถึง P3.7 หรือขา 10-17 ตามลำดับในรูปที่ 5 พอร์ทนี้มีโครงสร้างดังรูปที่ 2.20



รูปที่ 2.20 โครงสร้างของพอร์ท 3

ส่วนที่ 1 ในรูปที่ 2.20 เป็นส่วน Latch ข้อมูลที่เขียนมายัง Port 3 ทางInternal Bus เหมือนกับ พอร์ทอื่นๆ และพอร์ท 3 จะมี Internal Pull Up อยู่ทุกบิท แต่พอร์ท 3 นี้แต่ละบิทจะใช้ในการทำงานอื่นได้โดยใช้คำสั่งควบคุมการทำงาน ในส่วนที่ 2 จะมีสัญญาณ Alternative Output Function ที่สร้างมาจากส่วน Timing and Control สัญญาณ Alternative Output Function เป็นสัญญาณที่ส่งออกในกรณีที่ใช้พอร์ท 3 ทำงานในฟังก์ชั่นอื่น และจุด Alternative Input Function เป็นจุดที่จะเอาสัญญาณไปเข้ากับส่วนอื่นตามการทำงานของบิทนั้น แต่ละบิทของพอร์ท 3 จะมีฟังก์ชั่นอื่นดังนี้

P3.0/RXD (Serial Input Port) เป็นขาที่ใช้รับข้อมูลแบบอนุกรม

P3.1/TXD (Serial Output Port) เป็นขาที่ใช้ส่งข้อมูลแบบอนุกรม

P3.2/INT0 (External Interrupt) ใช้รับสัญญาณขัดจังหวะจากภายนอก

P3.3/INT1 (External Interrupt) ใช้รับสัญญาณขัดจังหวะจากภายนอก

P3.4/T0 (Timer / Counter 0 External Input) ขารับสัญญาณเข้าไปยังวงจร Timer/Counter0 ที่ทำหน้าที่นับจำนวนไซเคิลของสัญญาณ T0 นี้หรือสัญญาณนาฬิกาก็ได้

P3.5/T1 (Timer/Counter 1 External Input) ขารับสัญญาณเข้าไปยัง Timer/Counter 1 ซึ่งมีการทำงานเหมือนกับ T0

P3.6/WR (External Data Memory Write Strobe) ขาสัญญาณควบคุมการเขียนข้อมูลไปยังหน่วยความจำสำหรับข้อมูลภายนอก 8051

P3.7/RD (External Data Memory Read Strobe) ขาสัญญาณควบคุมการอ่านข้อมูลจากหน่วยความจำสำหรับข้อมูลภายนอก

RST

ขา Reset ขานี้จะใช้ทำการ Reset การทำงานของ 8051 ที่ขา RST ภายใน 8051 จะมีตัวต้านทานต่อระหว่างขานี้กับกราวด์ ถ้าป้อนสัญญาณที่มีสภาวะลอจิก 1 เข้าไปที่ขานี้จะเป็นการ Reset การทำงานของ 8051 ดังนั้นจึงสามารถต่อตัวเก็บประจุ (Capacitor) ภายนอกระหว่างขา RST กับไฟเลี้ยง +5 โวลท์ เพื่อให้เกิดการ Reset เมื่อเริ่มป้อนไฟเลี้ยงให้กับ 8051 ซึ่งเรียกว่า Power On Reset การ Reset จะทำให้ค่าใน Register ต่างๆ เปลี่ยนไปเป็นค่าหนึ่งดังตารางรูปที่ 2.21

เรจิสเตอร์	มีค่าข้อมูลเป็น	เรจิสเตอร์	มีค่าข้อมูลเป็น
PC	0000H	T2CON	00H
ACC	00H	TH0	00H
B	00H	TL0	00H
PSW	00H	TH1	00H
SP	07H	TL1	00H
DPTR	0000H	TH2	00H
P0-P3	0FFH	TL2	00H
IP(8051)	XXX00000B	RLDH	00H
IP(8052)	XX000000B	RLDH	00H
IE(8051)	0XX00000B	SCAN	00H
IE(8052)	0X000000B	SBUF	Indeterminate
TMOD	00H	PCON(HMOS)	0XXXXXXXB
TCON	00H	PCON(CMOS)	0XXX0000B

รูปที่ 2.21 ค่าของรีจิสเตอร์เมื่อเกิดการรีเซ็ต 8051

ในตารางรูปที่ 2.21 ช่องทางขวาเป็นค่าของ Register ที่อยู่ทางซ้ายเมื่อสิ้นสุดการ Reset ใน Register SBUF เมื่อสิ้นสุดการ Reset จะมีค่าที่ไม่แน่นอน และพอร์ทจะอยู่ในสภาวะลอจิก 1 ทุกบิท ตลอดเวลาที่สัญญาณของขา RST เป็น HIGH อยู่

เมื่อสัญญาณที่ขา RST กลับเป็น 0 ก็จะออกจากการ Reset 8051 จะเริ่มทำงานจากคำสั่งที่อยู่ใน Program memory ตำแหน่ง 0000H เพราะค่าของ Register PC (Program Counter) ซึ่งใช้ชี้ตำแหน่งโปรแกรมที่จะทำงานถูกเปลี่ยนให้เป็น 0000H ในเครื่องคอมพิวเตอร์แบบบอร์ดเดี่ยว (Single Board Microcomputer) จะมีโปรแกรมที่เขียนเก็บไว้เริ่มจากตำแหน่ง 0000H นี้เรียกว่า Monitor Program ที่คอยรับการกดจาก Keyboard และแสดงผลทางตัวแสดงผลแบบ 7 Segment

ALE

Address Latch Enable ขานี้ จะส่งสัญญาณที่มีความถี่ 1/6 เท่า ของสัญญาณนาฬิกา จากออสซิลเลเตอร์ สัญญาณนี้จะส่งออกมาตลอดเวลายกเว้นบางครั้งของการติดต่อกับหน่วยความจำสำหรับข้อมูลภายนอก 8051 สัญญาณนี้จะบอกกับอุปกรณ์ภายนอก 8051 ว่าขณะนี้สัญญาณนี้ Active (ลอจิก เป็น 1) จะมีการส่งข้อมูลที่เป็น 8 บิทล่างของตำแหน่งหน่วยความจำภายนอก 8051 ที่ต้องการติดต่อออกไปทางพอร์ท 0 อุปกรณ์ภายนอกจะใช้สัญญาณนี้ในการ Latch ข้อมูลไว้เพราะพอร์ท 0 จะส่งค่าตำแหน่งหน่วยความจำออกมาเพียงชั่วขณะเท่านั้น ซึ่งในเวลาต่อมาพอร์ท 0 จะใช้รับ-ส่งข้อมูลกับหน่วยความจำภายนอก สัญญาณ ALE จะสามารถต่อเข้ากับอุปกรณ์ TTL ชนิด LS ได้ถึง 8 อินพุท

PSEN

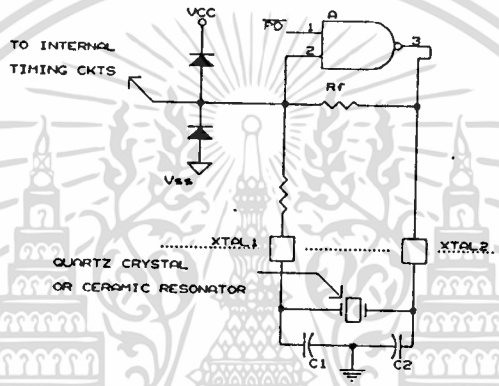
Program Store Enable เป็นขาที่ 29 ในรูปที่ 2.16 ขานี้ปกติจะให้ลอจิก 1 แต่จะส่งลอจิก 0 เมื่อต้องการอ่านคำสั่ง (Fetch Instruction) ที่จะนำไปทำงานมาจากหน่วยความจำสำหรับโปรแกรมภายนอก 8051 ในกรณีที่อ่านคำสั่งซึ่งเก็บอยู่ในหน่วยความจำสำหรับโปรแกรมภายใน 8051 แล้ว สัญญาณนี้จะไม่เปลี่ยนลอจิกเป็น 0 ขา PSEN นี้สามารถต่อไปยังขาอินพุทของ TTL ชนิด LS ได้ถึง 8 อินพุท

EA

External Access ขา 31 ของรูปที่ 2.16 ขานี้เป็นขาอินพุทที่ต่อเข้าไปยังวงจร Timing and Control ในรูปที่ 2.15 เพื่อควบคุมการสร้างสัญญาณ PSEN ถ้าป้อนสัญญาณลอจิก 0 เข้าไปที่ขา EA นี้แสดงว่าโปรแกรมในตำแหน่ง 0000H ถึง 0FFFH ที่ต้องการให้ทำงานถูกเก็บไว้ภายนอก 8051 จะต้องสร้างสัญญาณ PSEN ออกไปยังภายนอก เพื่อทำการ FETCH คำสั่งเข้ามาทำงาน แต่ถ้าสัญญาณที่ป้อนให้ขา EA เป็น 1 หมายความว่าโปรแกรมในตำแหน่ง 0000H ถึง 0FFFH ถูกเก็บไว้ภายใน 8051 การทำงานในตำแหน่งหน่วยความจำช่วงนี้จะอ่านคำสั่งต่างๆ จาก ROM ภายใน 8051

XTAL 1

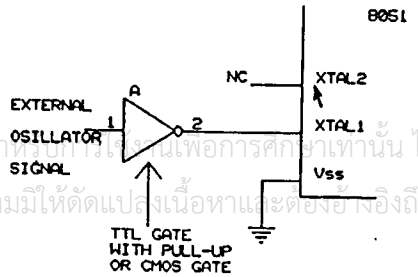
ขาที่ 19 ของรูปที่ 2.16 ขานี้จะต่อเข้ากับขาของ Inverting Amplifier (วงจรขยายแบบป้อนกลับเฟสสัญญาณ) ที่ประกอบเป็นวงจรออสซิลเลเตอร์ ในรูปที่ 2.22 จะเห็นวงจรภายในของออสซิลเลเตอร์ NAND Gate จะทำหน้าที่เป็นวงจรขยายแบบกลับเฟสของสัญญาณที่ควบคุมให้มีการออสซิลเลตหรือไม่ก็ขึ้นกับสัญญาณ PD ซึ่งต่อมาจากบิท PD ของ Register PCON ถ้าต้องการใช้สัญญาณนาฬิกา (Clock Signal) จากภายนอกมาเป็นสัญญาณนาฬิกา ควบคุมการทำงานของ 8051 ก็ให้ป้อนสัญญาณเข้ามาที่จุดนี้แต่ถ้าต้องการใช้วงจรออสซิลเลเตอร์ภายในก็ให้ต่อ Crystal หรือเซรามิคโซเนเตอร์ดังรูปที่ 2.22 Capacitor ในวงจรควรมีค่าประมาณ 20 PF



รูปที่ 2.22 วงจรออสซิลเลเตอร์ภายใน 8051

XTAL 2

ขาที่ 18 ของรูปที่ 2.16 ขานี้เป็นจุดเอาท์พุทของวงจรขยายแบบกลับเฟสสัญญาณที่ประกอบเป็นวงจรออสซิลเลเตอร์ (อินพุทคือขา XTAL 1) ถ้าจะใช้สัญญาณนาฬิกาที่สร้างมาจากภายนอกมาเป็นสัญญาณนาฬิกาของ 8051 แล้ว ให้ปล่อยขานี้ลอยไว้แล้วป้อนสัญญาณนาฬิกาจากภายนอกเข้ามาที่ขา XTAL 1 ดังรูปที่ 2.23



รูปที่ 2.23 8051 ที่ทำงานโดยสัญญาณที่มาจากภายนอก

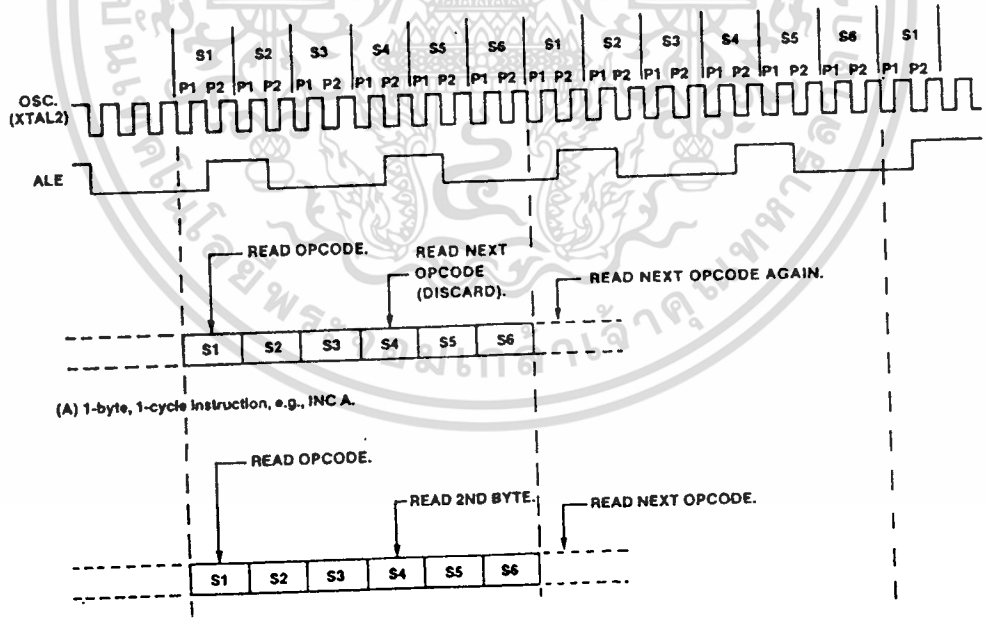
การทำงานของ 8051

คอมพิวเตอร์จะทำงานด้วยวงจรที่เรียกว่าฮาร์ดแวร์ (Hardware) **ประกอบขึ้นมาเพียงอย่างเดียวไม่ได้จะต้องมีโปรแกรมหรือคำสั่งที่จัดเรียงกันไว้ให้คอมพิวเตอร์ทำงานตามลำดับใน 8051 ก็เช่นกัน ผู้ใช้จะต้องเขียนโปรแกรมเป็นภาษาเครื่อง ซึ่งอยู่ในรูปของเลขฐาน 2 เก็บไว้ในหน่วยความจำประเภท** Program Memory **แต่ละคำสั่งของ 8051 อาจประกอบด้วย 1,2 หรือ 3 ไบท์ แล้วแต่ว่าจะเป็นคำสั่งให้ทำงานอะไร คอมพิวเตอร์ก็จะเหมือนกับคนที่จะต้องทำงานตามคำสั่ง เมื่อรับคำสั่งแล้วก็จะไปทำตามคำสั่งนั้นเสร็จสิ้นแล้วก็จะกลับมารับคำสั่งต่อไป**

จากรูปที่ 2.15 เมื่อเริ่มป้อนไฟเลี้ยงให้กับ 8051 ซึ่งมีวงจร Power on Reset **ต่ออยู่จะมีการ** Reset **เกิดขึ้น การทำงานภายใน 8051 จะเริ่มจากบล็อก** Program Counter **ซึ่งเป็นวงจรนับ** (Counter Circuit) **ชนิดหนึ่งส่งค่าตำแหน่งหน่วยความจำสำหรับโปรแกรมลงไปบนบัส** (bus) **หมายเลข 1 บัสนี้มีขนาด 16 บิท ค่าตำแหน่งหน่วยความจำนี้จะถูกส่งไปเก็บไว้ที่** Program ADDR Register **ที่เป็นวงจร** Latch **ข้อมูลซึ่งเป็นค่าตำแหน่งหน่วยความจำ จะปรากฏที่บนบัส 16 บิทหมายเลข 2 ถ้าเป็นค่าตำแหน่งหน่วยความจำแรกหลังจาก** Reset **ค่าตำแหน่งหน่วยความจำจะเป็น 0000H หน่วยความจำสำหรับโปรแกรมจะเลือกได้ว่าเป็น** ROM **ภายในหรือภายนอก 8051 โดยการป้อนสภาวะลอจิกเข้าไปที่ 8051 ทางขา** EA **ซึ่งต่ออยู่กับส่วน** Timing and Control **ทำหน้าที่เป็นวงจรถอดรหัส** (Decoder) **แล้วสร้างสัญญาณควบคุมต่อไปถ้าป้อนสัญญาณลอจิก 0 เข้าไปที่ขา** EA **จะเป็นการเลือกใช้** ROM **ภายใน 8051 โดยที่วงจร** Timing and Control **จะสร้างสัญญาณไปยัง** ROM **ภายในให้ส่งข้อมูลที่เป็นคำสั่งจากตำแหน่งที่ถูกชี้ด้วยค่าตำแหน่งที่ส่งมาทางบัสหมายเลข 2 ข้อมูลจาก** ROM **จะถูกส่งลงไปยังบัส หมายเลข 3 ที่เรียกว่า** Internal Data Bus **แล้วนำไปเก็บไว้ที่** Instruction Register (**เป็นวงจร** Latch) **เพื่อส่งต่อไปให้กับวงจร** Timing and Control **ทำการถอดรหัสแล้วควบคุมการทำงานส่วนอื่นๆ ต่อไปแล้วแต่ว่าเป็นคำสั่งให้ทำงานอะไร ในกรณีที่เลือก** ROM **ภายนอก 8051 โดยป้อนสัญญาณลอจิก 1 เข้าไปที่ขา** EA **จะทำให้วงจร** Timing and Control **ส่งสัญญาณไปยังพอร์ท 0 และพอร์ท 2 เพื่อส่งค่าตำแหน่งหน่วยความจำบนบัสหมายเลข 2 ออกไปชี้หน่วยความจำภายนอก จากนั้นจะอ่านข้อมูลที่เป็นคำสั่งกลับเข้ามาทางพอร์ท 0 ไปยัง** Internal Data Bus **แล้วไปเก็บที่** Instruction Register **เพื่อทำงานต่อไปเหมือนกับตอนอ่านคำสั่งจาก** ROM **ภายในการทำงานในช่วงส่งค่าตำแหน่งหน่วยความจำไปยังหน่วยความจำแล้วอ่านข้อมูลที่เป็นคำสั่งกลับเข้ามาเก็บไว้ใน** Instruction Register **เรียกว่าเป็นช่วงของการ** FETCH (FETCH CYCLE) **ช่วงต่อไปจะเป็นช่วง**

ของการทำงานตามคำสั่งเรียกว่า Execute Cycle เช่นถ้าเป็นคำสั่งให้บวกข้อมูลใน Register Accumulator กับข้อมูลจากหน่วยความจำ Data Memory ภายใน RAM ตำแหน่ง 23 H วงจร Timing and Control ก็จะส่งสัญญาณให้ Instruction Register ส่งค่าตำแหน่งหน่วยความจำ 23H ลงไปยัง Internal Data Bus แล้วนำข้อมูลไปเก็บไว้ที่ RAM ADDR Register เพื่อใช้ชี้ตำแหน่งหน่วยความจำ RAM จากนั้น Timing and Control จะสั่งให้ RAM ส่งข้อมูลที่เก็บอยู่ในหน่วยความจำตำแหน่ง 23H ลงมายัง Internal Data Bus แล้วนำข้อมูลไปเก็บไว้ที่ TMP1 (วงจร Latch) ขณะเดียวกันวงจร Timing and Control ก็จะส่งสัญญาณไปยัง ACC ให้ส่งข้อมูลมายัง TMP 2 (วงจร Latch) วงจร ALU ซึ่งโครงสร้างเป็นวงจรทำการคำนวณทางคณิตศาสตร์ (บวก,ลบ,คูณ,หาร) และยังสามารถทำงานทางลอจิก (AND , OR ,NOT , XOR) จะทำการบวกเลขจาก TMP1 และ TMP2 เข้าด้วยกันผลลัพธ์ที่ได้จะส่งผ่าน Internal Data Bus กลับไปเก็บยัง ACC PSW (Program Status Word) ซึ่งจะทำหน้าที่เก็บสถานะผลลัพธ์ของการทำงานใน ALU เช่นผลลัพธ์การบวกมีค่าเกิน 8 บิท ก็จะทำให้บิทหนึ่งใน PSW ถูก SET เป็น 1

การทำงานที่กล่าวมาข้างต้นจะขึ้นกับสัญญาณควบคุมที่สร้างมาจากวงจร Timing and Control และสัญญาณที่สร้างขึ้นนี้จะอ้างอิงกับสัญญาณนาฬิกาที่สร้างมาจากวงจร Oscillator ทำให้การทำงานต่างๆ เป็นไปตามลำดับที่ผู้ผลิตได้ออกแบบไว้ ดังในรูปที่ 2.24



รูปที่ 2.24 ลำดับสถานะการทำงานใน MCS-51

คำสั่งแต่ละคำสั่งของ 8051 จะใช้เวลาทำงาน 1,2 หรือ 3 ไซเคิลของเครื่อง (Machine Cycle) แล้วแต่ว่าเป็นคำสั่งประเภทใด 1 ไซเคิลของเครื่องจะใช้เวลา 12 ไซเคิลของสัญญาณนาฬิกา ดังนั้น

ในแต่ละคำสั่งของ 8051 จะใช้เวลาทำงาน 12,24 หรือ 36 ไซเคิลของสัญญาณนาฬิกานั่นเอง แต่ละไซเคิลของเครื่องจะถูกแบ่งออกเป็น 6 State คือ S1,S2,S3,S4,S5 และ S6 แต่ละ State จะประกอบด้วย 2 ไซเคิลของสัญญาณนาฬิกา ในไซเคิลแรกจะเรียกว่า เฟส 1(P1) และไซเคิลที่ 2 เรียกเฟส 2 (P2) ในแต่ละเฟสจะนับตั้งแต่ขอบขาลงของสัญญาณนาฬิกาจนถึงขอบขาลงของสัญญาณนาฬิกาที่อยู่ถัดไปดังที่เห็นในรูปที่ 2.24 เมื่อ8051 ทำงานเสร็จ 1ไซเคิลของเครื่องก็จะเริ่มทำงาน State1 Phase1 (S1P1) ของไซเคิลต่อไป ใน 1 ไซเคิลของเครื่องวงจร Timing and Control จะสร้างสัญญาณ ALE ออกมา 2 ไซเคิลเพื่อ Fetch คำสั่งเข้าไป 2 ครั้งเสมอ ที่บริเวณขอบขาขึ้นของสัญญาณ ALE คำสั่งใดจะมีกี่ไบท์หรือใช้เวลาทำงานกี่ไซเคิลจะสามารถดูได้จากตารางชุดคำสั่ง 8051

คำสั่งประเภท 1 ไบท์ 1 ไซเคิลของเครื่องได้แก่คำสั่ง INCA จะมีการอ่านคำสั่งจากหน่วยความจำสำหรับโปรแกรม 2 ครั้ง ที่เวลาประมาณขอบขาขึ้นของสัญญาณ ALE เมื่อคำสั่งแรกถูกอ่านเข้าไปที่เวลาที่เวลาขอบขาขึ้นของสัญญาณ ALE แรก แล้วนำไปเก็บที่ Instruction Register เพื่อให้วงจร Timing and Control ถอดรหัสและเข้าสู่การ Execute ขณะเดียวกันก็จะเริ่มต้นการ Fetch คำสั่งที่อยู่ในหน่วยความจำตำแหน่งถัดไปเข้ามาและคำสั่งที่ 2 จะถูกอ่านเข้ามาที่เวลาขอบขาขึ้นของสัญญาณ ALE ถัดไป วงจร Timing and Control เมื่อถอดรหัสคำสั่งแรกก็จะทราบว่าการทำงานคำสั่งนี้ให้สิ้นสุดจะใช้คำสั่งเพียง 1 ไบท์ ดังนั้นคำสั่งที่ถูกอ่านมาในไบท์ที่ 2 จะไม่ถูกนำมาทำงานเพียงแต่อ่านเข้ามาแล้วทิ้งไป (Discard) ดังในรูปที่ 2.24a

คำสั่งประเภท 2 ไบท์ และใช้เวลา 1 ไซเคิลของเครื่องได้แก่คำสั่ง ADD A ,#data ใน 1 ไซเคิลของเครื่องนี้ จะมีการอ่านคำสั่งเข้ามา 2 ไบท์ เหมือนกับคำสั่งประเภท 1 ไบท์ 1 ไซเคิลของเครื่อง แตกต่างกันที่ไบท์ที่ 2 จะถูกนำมาใช้งานด้วยไม่ได้ถูกทิ้งไปดังที่เห็นในรูป 13 b ตัวอย่างของคำสั่ง ADD A,#33H จะเขียนเป็นภาษาเครื่องได้ 2 ไบท์คือ 24 33 เมื่ออ่านคำสั่งไบท์แรกคือ 24 เข้าไปไว้ที่ Instruction Register แล้ว Timing and Control จะถอดรหัสพบว่าเป็นคำสั่งบวกเลข ก็จะส่งสัญญาณไปที่ Accumulator ให้เอาข้อมูลไปไว้ที่ TMP1 เมื่อคำสั่งที่ 2 ถูกอ่านเข้ามาที่ Instruction Register แล้ว Timing and Control จะสั่งให้เอาข้อมูลไบท์ที่ 2 ส่งลงไปยัง Internal Data Bus ไปเก็บยัง TMP1 จากนั้นวงจร ALU จะนำเอาข้อมูล TMP1 และ TMP2 มาบวกกัน ผลลัพธ์ที่ได้จะส่งออกจาก ALU ไปยัง Internal Data Bus แล้วไปเก็บไว้ที่ Accumulator

คำสั่งประเภท 1 , 2 และ 3 ไบท์ที่ใช้เวลาทำงาน 2 ไซเคิลของเครื่องเช่นคำสั่ง INC DPTR จะมีการอ่านคำสั่งเข้าไป 4 ครั้งทุกๆ ขอบขาขึ้นของสัญญาณ ALE ที่มี 2 ครั้งต่อ 1 ไซเคิลของเครื่อง ถ้าเป็นคำสั่งประเภท 1,2 หรือ 3 ไบท์ วงจร Timing and Control จะเอาคำสั่ง 1,2 หรือ 3 ไบท์แรกเท่านั้นไปทำงานส่วนคำสั่งที่เหลือทิ้งไปดังในรูปที่ 2.24C คำสั่ง 1 ไบท์ที่ใช้เวลาทำงาน 2 ไซเคิลของเครื่องที่กล่าวมาแล้วจะไม่รวมถึงคำสั่ง MOVX ซึ่งใช้ในการอ่านหรือเขียนข้อมูลกับหน่วยความจำ Data Memory ภายนอก การทำงานของคำสั่งนี้จะมีการ Fetch คำสั่งเข้าไปแต่จะเป็นช่วง

สั่งนั้นให้เสร็จสิ้นเสียก่อนจึงจะเริ่มการ Reset ได้โดย 8051 จะดูสภาวะของสัญญาณที่ขา RST ของ S5P2 ในไซเคิลของเครื่องสุดท้ายเท่านั้น ดังนั้นใน S5P2 ของไซเคิลเครื่องแรกๆ ในคำสั่งอาจมีสภาวะลอจิกที่ขา RST เป็น 1 แต่ที่ S5P2 ของไซเคิลเครื่องสุดท้าย มีสภาวะลอจิกที่ขา RST เป็น 0 ก็จะไม่เกิดการ Reset

ขึ้นที่เวลา S5P2 เมื่อตรวจสอบสภาวะสัญญาณที่ขา RST แล้วพบว่าเป็น 1 จะต้องรอไปจนถึงเวลา S4P2 ที่ตรวจพบสัญญาณ RST มีลอจิกเป็น 1 จนถึง S4P2 ของไซเคิลเครื่องถัดไปจะยังคงมีการ Fetch คำสั่งเข้าไปทำงานอีก 2 คำสั่ง เมื่อสัญญาณ Reset ภายในเปลี่ยนเป็น 1 ก็จะเริ่มการ Reset โดยการเขียนข้อมูล 0 ไปยัง Special Function Register ทุกตัว ยกเว้นพอร์ท 0 ถึงพอร์ท 3 Stack Pointer และ Register SBUF ดังตารางในรูปที่ 2.21 ระหว่างนี้ข้อมูลใน RAM ภายใน 8051 จะไม่เปลี่ยนแปลงข้อมูลในระหว่างการเขียนข้อมูลลงไปยัง SFR จะมีการ Fetch คำสั่งเข้ามาทำงานอีก 1 คำสั่งจนกว่าจะถึง S3P1 ของไซเคิลที่ 2 (นับตั้งแต่ไซเคิลของเครื่องที่ตรวจพบลอจิก 1 ที่ขา RST) ก็จะทำให้สภาวะลอจิกที่ขา ALE และ PSEN ค้างอยู่ที่สภาวะลอจิก 1 และจะเป็นอย่างนี้ไปจนกว่าสภาวะลอจิกที่ขา RST เป็น 0 เวลานับตั้งแต่พบสัญญาณลอจิก 1 ที่ขา RST ที่เวลา S5P2 จนถึงเวลาที่ ALE และ PSEN ค้างอยู่ที่ 1 จะเท่ากับ 19 ไซเคิลของ Oscillator เมื่อสัญญาณที่ขา RST ถูกเปลี่ยนกลับเป็นลอจิก 0 8051 จะรออีก 1 ถึง 2 ไซเคิลของเครื่องสัญญาณ ALE และ PSEN จะเริ่มเปลี่ยนแปลงเพื่อเริ่มกระบวนการ Fetch คำสั่งเข้าไปทำงานเริ่มจากคำสั่งในหน่วยความจำสำหรับโปรแกรมตำแหน่ง 0000H

ชุดคำสั่งของ 8051 โดยสังเขป

สัญลักษณ์ที่ใช้

Rn	Register ภายใน R0-R7
direct	Register SFR และหน่วยความจำข้อมูลภายใน
@Ri	ค่าแอดเดรสหน่วยความจำภายใน อ้างแอดเดรสโดยอ้อม โดยผ่าน Register R0 หรือ R1
#data	ค่าคงที่ขนาด 8 บิท (ค่าจาก 00-FF)
#data16	ค่าคงที่ขนาด 16 บิท (ค่าจาก 00-FFFF)

addr11 ค่าตำแหน่งแอดเดรสจำนวน 11 บิท สำหรับคำสั่ง ACALL หรือ AJMP

addr16 ค่าตำแหน่งแอดเดรสจำนวน 16 บิท สำหรับคำสั่ง LCALL หรือ LJMP

rel ค่าออฟเซ็ท (offset) หรือค่าบอกความสัมพัทธ์ (ค่าจาก -17 ถึง 18 ไบท์)

bit ตำแหน่งบิทของหน่วยความจำภายในที่อ้างถึงได้แบบบิทหรือ SFR

@DPTR แอดเดรสของหน่วยความจำภายนอก หรือพอยเตอร์โดยอ้อมผ่าน Register DPTR

คำสั่งประมวลผลทางคณิตศาสตร์ (Arithematic Operations)

ADD A,Rn ให้ทำการบวกข้อมูลภายใน Register R0-R7 กับข้อมูลภายในรีจิสเตอร์ A และ นำผลที่ได้ไปเก็บไว้ยังรีจิสเตอร์ A

ADD A,#data8 ให้ทำการบวกค่าข้อมูลตัวเลข data8 กับข้อมูลภายในรีจิสเตอร์ A และนำผลที่ได้ ไปเก็บไว้ยังรีจิสเตอร์ A

ADD A,addr8 ให้ทำการบวกค่าข้อมูลที่เก็บอยู่ภายในหน่วยความจำตำแหน่ง addr8 กับข้อมูลภายในรีจิสเตอร์ A และนำผลที่ได้ไปเก็บไว้ยังรีจิสเตอร์ A

ADD A,@Ri ให้ทำการบวกค่าข้อมูลที่เก็บอยู่ภายในหน่วยความจำ ตำแหน่งที่มีค่าเก็บอยู่ภายใน Register R0 หรือ R1 กับข้อมูลภายในรีจิสเตอร์ A และนำผลที่ได้ไปเก็บไว้ยัง รีจิสเตอร์ A

ADDC A,#data8 ให้ทำการบวกค่าข้อมูลตัวเลข data8 กับ ข้อมูลภายในรีจิสเตอร์ A และข้อมูลภายในแฟลกทด โดยนำผลที่ได้ไปเก็บไว้ยังรีจิสเตอร์ A

ADDC A,Rn ให้ทำการบวกข้อมูลภายในรีจิสเตอร์ R0-R7 กับข้อมูลภายในรีจิสเตอร์ A และ ข้อมูลภายในแฟลกทด โดยนำผลที่ได้ไปเก็บไว้ยังรีจิสเตอร์ A

ADDC A,addr8 ให้ทำการบวกค่าข้อมูล ที่เก็บอยู่ภายในหน่วยความจำตำแหน่ง addr8 กับข้อมูลภายในรีจิสเตอร์ A และข้อมูลภายในแฟลกทด โดยนำผลที่ได้ ไปเก็บไว้ ยังรีจิสเตอร์ A

ADDC A,@Ri	ให้ทำการบวกค่าข้อมูลที่เก็บอยู่ภายในหน่วยความจำตำแหน่ง ที่มีค่าเก็บอยู่ภายในรีจิสเตอร์ R0 หรือ R1 กับข้อมูลภายในรีจิสเตอร์ A และข้อมูลภายในแฟลก ทด โดยนำผลที่ได้ไปเก็บไว้ยังรีจิสเตอร์ A
SUBB A,#data8	ให้ทำการลบค่าข้อมูลตัวเลข data8 กับข้อมูลภายในรีจิสเตอร์ A และข้อมูลภาย ในแฟลกทด โดยนำผลที่ได้ไปเก็บไว้ยังรีจิสเตอร์ A
SUBB A,Rn	ให้ทำการลบข้อมูลภายในรีจิสเตอร์ R0-R7 กับข้อมูลภายในรีจิสเตอร์ A และ ข้อมูลภายในแฟลกทด โดยนำผลที่ได้ไปเก็บไว้ยังรีจิสเตอร์ A
SUBB A,addr8	ให้ทำการลบค่าข้อมูลที่เก็บอยู่ภายในหน่วยความจำตำแห่ง addr8 กับข้อมูลภายในรีจิสเตอร์ A และข้อมูลภายในแฟลกทด โดยนำผลที่ได้ไปเก็บไว้ยังรีจิสเตอร์ A
SUBB A,@Ri	ให้ทำการลบค่าข้อมูลที่เก็บอยู่ภายในหน่วยความจำ ตำแหน่งที่มีค่าเก็บอยู่ภายในรีจิสเตอร์ R0 หรือ R1 กับข้อมูลภายในรีจิสเตอร์ A และข้อมูลภายในแฟลกทด โดยนำผลที่ได้ไปเก็บไว้ยังรีจิสเตอร์ A
MUL AB	ให้ทำการคูณค่าข้อมูลภายในรีจิสเตอร์ A และรีจิสเตอร์ B โดยนำผลลัพธ์ที่ได้ในไบท์ต่ำไปเก็บไว้ ยังรีจิสเตอร์ A และ ข้อมูลในไบท์หลักสูงไปเก็บไว้ที่รีจิสเตอร์ B
DIV AB	ให้ทำการหารค่าข้อมูลภายในรีจิสเตอร์ A และรีจิสเตอร์ B โดยนำผลลัพธ์ที่ได้ในไบท์หลักต่ำ ไปเก็บยังรีจิสเตอร์ A และข้อมูลไบท์หลักสูงไปเก็บไว้ที่รีจิสเตอร์ B
DA A	ให้ทำการปรับค่าผลบวกค่าข้อมูลภายในรีจิสเตอร์ A เป็นเลขแบบบีซีดี และนำผลที่ได้ไปเก็บไว้ยังรีจิสเตอร์ A
INC A	เป็นการเพิ่มค่าข้อมูลที่อยู่ภายในรีจิสเตอร์ A
INC Rn	เป็นการเพิ่มค่าข้อมูลที่อยู่ภายในรีจิสเตอร์ R0-R7
INC direct	เป็นการเพิ่มค่าข้อมูลที่อยู่ภายในหน่วยความจำข้อมูลภายในที่แอดเดรส direct
INC @Ri	เป็นการเพิ่มค่าให้กับค่าในหน่วยความจำที่เก็บอยู่ใน R0 หรือ R1
DEC A	เป็นการลดค่าข้อมูลที่อยู่ภายในรีจิสเตอร์ A
DEC Rn	เป็นการลดค่าข้อมูลที่อยู่ภายในรีจิสเตอร์ A

DEC	direct	เป็นการลดค่าข้อมูลที่อยู่ภายในหน่วยความจำข้อมูลภายใน ที่แอดเดรสdirect
DEC	@Ri	เป็นการลดค่าข้อมูลที่อยู่ภายในตำแหน่งของหน่วยความจำที่เก็บในรีจิสเตอร์ R0 หรือ R1

คำสั่งทางตรรก (Logical Operations)

ANL	A,#data	ให้ทำการ AND แต่ละบิทข้อมูลของรีจิสเตอร์ A กับแต่ละบิทภายใน ตัวเลข data ที่มีตำแหน่งตรงกัน และเก็บค่าของผลลัพธ์ไว้ยังรีจิสเตอร์ A
ANL	A,direct	ให้ทำการ AND แต่ละบิทข้อมูล ของรีจิสเตอร์ A กับแต่ละบิทภายในข้อมูลที่อยู่ในแอดเดรส direct ที่มีตำแหน่งตรงกัน และเก็บค่าของผลลัพธ์ ไว้ยัง รีจิสเตอร์ A
ANL	A,Rn	ให้ทำการ AND แต่ละบิทข้อมูลของรีจิสเตอร์ A กับแต่ละบิทภายในรีจิสเตอร์ R0-R7ที่มีตำแหน่งตรงกัน และเก็บค่าผลลัพธ์ไว้ยังรีจิสเตอร์ ให้ทำการANDแต่ละบิทข้อมูลของรีจิสเตอร์ A
ANL	A,@Ri	ให้ทำการ AND แต่ละบิทภายในข้อมูลของรีจิสเตอร์ A กับแต่ละบิทภายในของข้อมูล ของแอดเดรสที่มีค่าเก็บไว้ภายในรีจิสเตอร์ R0 หรือ R1 ที่มีตำแหน่งตรงกัน และเก็บค่าของผลลัพธ์ไว้ยังรีจิสเตอร์ A
ANL	direct,A	ให้ทำการ AND แต่ละบิทข้อมูลของหน่วยความจำแอดเดรส direct กับแต่ละบิทภายในข้อมูลของรีจิสเตอร์ A ที่มีตำแหน่งตรงกัน และเก็บค่าของผลลัพธ์ไว้ยังหน่วยความจำ direct
ANL	direct,#data	ให้ทำการ AND แต่ละบิทข้อมูลของหน่วยความจำแอดเดรส direct กับแต่ละบิทภายในตัวเลข data ที่มีตำแหน่งตรงกัน และเก็บค่าของผลลัพธ์ไว้ยังหน่วยความจำ direct
ORL	A,#data	ให้ทำการ OR แต่ละบิทข้อมูลของรีจิสเตอร์ A กับแต่ละบิทภายในตัวเลข data ที่มีตำแหน่งตรงกัน และเก็บค่าของผลลัพธ์ไว้ยังรีจิสเตอร์ A
ORL	A,direct	ให้ทำการ OR แต่ละบิทของข้อมูลของรีจิสเตอร์ A กับแต่ละบิทภายในข้อมูลที่อยู่ในแอดเดรส direct ที่มีตำแหน่งตรงกัน และเก็บค่าผลลัพธ์ไว้ยังรีจิสเตอร์ A

ORL	A,Rn	ให้ทำการ OR แต่ละบิทข้อมูลของรีจิสเตอร์ A กับแต่ละบิทภายในรีจิสเตอร์ R0-R7 ที่มีตำแหน่งตรงกัน และเก็บค่าผลลัพธ์ไว้ยังรีจิสเตอร์ A
ORL	A,@Ri	ให้ทำการ OR แต่ละบิทข้อมูลของรีจิสเตอร์ A กับแต่ละบิทภายในข้อมูลของแอดเดรสที่มีค่าเก็บไว้ภายในรีจิสเตอร์ R0 หรือ R1 ที่มีตำแหน่งตรงกัน และเก็บค่าของผลลัพธ์ไว้ยังรีจิสเตอร์ A
ORL	direct,A	ให้ทำการ OR แต่ละบิทข้อมูลของหน่วยความจำแอดเดรส direct กับแต่ละบิทภายในข้อมูลของรีจิสเตอร์ A ที่มีตำแหน่งตรงกัน และเก็บค่าของผลลัพธ์ไว้ยังหน่วยความจำ direct
ORL	direct,#data	ให้ทำการ OR แต่ละบิทข้อมูลของหน่วยความจำแอดเดรส direct กับแต่ละบิทภายในตัวเลข data ที่มีตำแหน่งตรงกัน และเก็บค่าของผลลัพธ์ไว้ยังหน่วยความจำ direct
XRL	A,#data	ให้ทำการ XOR แต่ละบิทข้อมูลของรีจิสเตอร์ A กับแต่ละบิทภายในตัวเลข data ที่มีตำแหน่งตรงกัน และเก็บค่าของผลลัพธ์ไว้ยังรีจิสเตอร์ A ให้ทำการ XOR แต่ละบิทข้อมูลของรีจิสเตอร์ A กับแต่ละบิทภายในข้อมูล ที่อยู่ในแอดเดรส direct ที่มีตำแหน่งตรงกัน และเก็บค่าของผลลัพธ์ไว้ยังรีจิสเตอร์ A
XRL	A,Rn	ให้ทำการ XOR แต่ละบิทของรีจิสเตอร์ A กับแต่ละบิทภายในรีจิสเตอร์ R0-R7 ที่มีตำแหน่งตรงกัน และเก็บค่าของผลลัพธ์ไว้ยังรีจิสเตอร์ A
XRL	A,@Ri	ให้ทำการ XOR แต่ละบิทข้อมูลของรีจิสเตอร์ A กับแต่ละบิทภายในข้อมูลของแอดเดรส ที่มีค่าเก็บไว้ภายในรีจิสเตอร์ R0 หรือ R1 ที่มีตำแหน่งตรงกัน และเก็บค่าของผลลัพธ์ไว้ยังรีจิสเตอร์ A
XRL	direct,A	ให้ทำการ XOR แต่ละบิทข้อมูลของหน่วยความจำแอดเดรส direct กับแต่ละบิทภายในข้อมูลของรีจิสเตอร์ A ที่มีตำแหน่งตรงกัน และเก็บค่าของผลลัพธ์ไว้ยังหน่วยความจำ direct
XRL	direct,#data	ให้ทำการ XOR แต่ละบิทข้อมูลของหน่วยความจำแอดเดรส direct กับแต่ละบิทภายในตัวเลข data ที่มีตำแหน่งตรงกัน และเก็บค่าของผลลัพธ์ไว้ยังหน่วยความจำ direct
CLR	A	ให้กำหนดค่าของแต่ละบิทภายในรีจิสเตอร์ A เป็น 0

CPL	A	ให้เปลี่ยนค่าของแต่ละบิทภายในรีจิสเตอร์ A เป็นค่าตรงกันข้าม (จาก 0 เป็น 1 หรือ จาก 1 เป็น 0)
RL	A	ให้ทำการเลื่อนบิทข้อมูลภายในรีจิสเตอร์ A ไปทางด้านซ้ายหนึ่งตำแหน่ง โดย บิทนัยสำคัญสูงสุด (บิท 7) ให้เลื่อนมาเก็บยังตำแหน่งบิท 0
RR	A	ให้ทำการเลื่อนบิทข้อมูลภายในรีจิสเตอร์ A ไปทางด้านขวาหนึ่งตำแหน่ง โดย บิทนัยสำคัญต่ำสุด (บิท 0) ให้เลื่อนมาเก็บยังตำแหน่งบิท 7
RLC	A	ให้ทำการเลื่อนบิทข้อมูลภายในรีจิสเตอร์ A ไปทางด้านซ้ายหนึ่งตำแหน่ง โดย บิทนัยสำคัญสูงสุด (บิท 7) ให้เลื่นมาเก็บยังตำแหน่งของแฟลกทดและค่าในแฟลกทดเดิมให้เลื่อนเข้าไปเก็บยังบิท 0
RRC	A	ให้ทำการเลื่อนบิทข้อมูลภายในรีจิสเตอร์ A ไปทางด้านขวาหนึ่งตำแหน่ง โดย บิทนัยสำคัญต่ำสุด (บิท 0) ให้เลื่อนมาเก็บยังตำแหน่งของแฟลกทดและค่าในแฟลกทดเดิมให้เลื่อนเข้าไปเก็บยังบิท 7
SWAP	A	ให้ทำการสลับค่าบิทข้อมูลภายในรีจิสเตอร์ A โดยย้ายตำแหน่งข้อมูล 4 บิทบน มายังตำแหน่ง 4 บิทล่าง และย้ายข้อมูลที่เดิมอยู่ในตำแหน่ง 4 บิทล่างไปอยู่ในตำแหน่ง 4 บิทบน

คำสั่งเคลื่อนย้ายข้อมูล (Data Tranfer)

MOV	A,Rn	ให้นำข้อมูลจากรีจิสเตอร์ R0 -R7 มาเก็บไว้ยังรีจิสเตอร์ A
MOV	A,direct	ให้นำค่าจากหน่วยความจำข้อมูลภายใน มาเก็บไว้ที่รีจิสเตอร์ A
MOV	A,data	ให้นำค่าข้อมูลตัวเลขมาเก็บไว้ที่รีจิสเตอร์ A
MOV	Rn,#data	ให้นำข้อมูลจากรีจิสเตอร์ A มาเก็บไว้ยังรีจิสเตอร์ Rn
MOV	Rn,direct	ให้นำค่าจากหน่วยความจำข้อมูลภายใน มาเก็บไว้ที่รีจิสเตอร์ Rn
MOV	Rn,#data	ให้นำค่าข้อมูลตัวเลขมาเก็บไว้ที่รีจิสเตอร์ Rn
MOV	direct,#data	ให้นำค่าข้อมูลตัวเลขมาเก็บไว้ในหน่วยความจำข้อมูลภายใน
MOV	A,@Rn	ให้นำข้อมูล จากหน่วยความจำที่มีค่าเก็บอยู่ภายในรีจิสเตอร์ Rn มาเก็บไว้ ยังรีจิสเตอร์ A

MOV	@Rn,A	ให้นำข้อมูลจากรีจิสเตอร์ A มาเก็บไว้ยัง หน่วยความจำ ที่มีค่าเก็บอยู่ ภายในรีจิสเตอร์Rn
MOV	C,bit	ให้นำข้อมูลแบบบิตจากหน่วยความจำมาเก็บไว้ยังแฟล็กทด (รีจิสเตอร์ C)
MOV	bit,C	ให้นำข้อมูลจากแฟล็กทด มาเก็บไว้ยังหน่วยความจำบริเวณที่เก็บข้อมูลแบบบิตได้
MOVX	A,@Rn	ให้นำข้อมูลจากหน่วยความจำภายนอก ที่มีค่าเก็บอยู่ภายในรีจิสเตอร์ Rn มาเก็บไว้ยังรีจิสเตอร์ A
MOVX	A,DPTR	ให้นำข้อมูลจากหน่วยความจำภายนอก ที่มีค่าเก็บอยู่ภายใน รีจิสเตอร์ DPTR มาเก็บไว้ยังรีจิสเตอร์ A
MOVX	@Rn,A	ให้นำข้อมูลจากรีจิสเตอร์ A มาเก็บไว้ยังหน่วยความจำภายนอกที่มีค่าเก็บ อยู่ภายในรีจิสเตอร์ Rn
MOVX	DPTR,A	ให้นำข้อมูลจากรีจิสเตอร์ A มาเก็บไว้ยังหน่วยความจำภายนอกที่มีค่าเก็บ อยู่ภายในรีจิสเตอร์ DPTR
MOVC	A,@A+DPTR	ให้นำข้อมูลจากหน่วยความจำโปรแกรม โดยพิจารณาตำแหน่งจากการบวกค่ารีจิสเตอร์ A กับค่าของรีจิสเตอร์ DPTR
MOVC	A,@A+PC	ให้นำข้อมูลจากหน่วยความจำโปรแกรม โดยพิจารณาตำแหน่งจากการบวกค่ารีจิสเตอร์ A กับค่าของรีจิสเตอร์
XCH	A,Rn	ให้แลกเปลี่ยนข้อมูลภายในรีจิสเตอร์ R0-R7 กับรีจิสเตอร์ A
XCH	A,direct	ให้แลกเปลี่ยนข้อมูลภายในหน่วยความจำตำแหน่ง direct กับ รีจิสเตอร์ A
XCH	A,Rn	ให้แลกเปลี่ยน ข้อมูลภายในหน่วยความจำตำแหน่งที่มีค่าเก็บไว้ภายในรีจิสเตอร์ R0 หรือ R1 กับรีจิสเตอร์ A
XCHD	A,direct	ให้แลกเปลี่ยนข้อมูลเฉพาะ 4 บิตล่าง ระหว่างหน่วยความจำตำแหน่งที่มีค่าภายในรีจิสเตอร์ R0 หรือ R1 กับรีจิสเตอร์ A
PUSH	direct	ให้ทำการเพิ่มค่าภายในรีจิสเตอร์ SP จากนั้นให้โอนย้ายข้อมูลจากหน่วยความ จำ แอดเดรส direct ไปเก็บยังหน่วยความจำข้อมูลภายในที่ตำแหน่งซึ่งเก็บค่าไว้ในรีจิสเตอร์ SP
POP	direct	ให้ทำการโอนย้ายข้อมูลจากหน่วยความจำข้อมูลภายใน ที่ตำแหน่งซึ่งเก็บค่าไว้ในรีจิสเตอร์ SPไปยังหน่วยความจำข้อมูลภายในที่ตำแหน่ง direct และค่าภายในตำแหน่ง direct และลดค่าภายในรีจิสเตอร์ SP

คำสั่งจัดการข้อมูลแบบบิต (Boolean Variable Manipulation)

CPL	C	ให้กลับค่าบิทของแฟลกทดเป็นค่าตรงกันข้าม
CPL	bit	ให้กลับค่าบิทของข้อมูลในตำแหน่งบิทแอดเดรส bit เป็นค่าตรงกันข้าม
CLR	C	ให้ทำการเคลียร์ค่าภายในแฟลกทดให้เป็นค่า 0
CLR	bit	ให้ทำการเคลียร์ค่าบิทของข้อมูลในตำแหน่งบิทแอดเดรส bit ให้เป็นค่า 0
SETB	C	ให้ทำการเซตค่าภายในแฟลกทดให้เป็น 1
SETB	bit	ให้ทำการเซตค่าบิทของข้อมูลในตำแหน่งบิทแอดเดรส bit เป็น 1
ANL	C,bit	ให้ทำการ AND ข้อมูลของแฟลกทดกับบิทแอดเดรส bit และเก็บค่าของผลลัพธ์ไว้ยังแฟลกทด
ANL	C,/bit	ให้ทำการ AND ข้อมูลของแฟลกทดกับค่าตรงข้ามของบิทแอดเดรส bit และเก็บค่าของผลลัพธ์ไว้ยังแฟลกทด
ORL	C,bit	ให้ทำการ OR ข้อมูลของแฟลกทดกับบิทแอดเดรส bit และเก็บค่าของผลลัพธ์ ไว้ยังแฟลกทด
ORL	C,/bit	ให้ทำการ OR ข้อมูลของแฟลกทดกับค่าตรงข้ามของบิทแอดเดรส bit และเก็บค่าของผลลัพธ์ไว้ยังแฟลกทด
MOV	C,bit	ให้โอนย้ายบิตข้อมูลจากตำแหน่งบิตแอดเดรส bit มาเก็บไว้ยังแฟล็กทด
MOV	bit,C	ให้โอนย้ายบิตข้อมูลจากแฟล็กทด มาเก็บไว้ยังตำแหน่งบิตแอดเดรส bit
JC	rel	กระโดด ถ้าค่าแฟล็ก carry เป็น 1
JNC	rel	กระโดด ถ้าค่าแฟล็ก carry เป็น 0
JB	bit,rel	กระโดด ถ้าค่า bit เป็น 1
JNB	bit,rel	กระโดด ถ้าค่า bit เป็น 0
JBC	bit,rel	กระโดด ถ้าค่า bit เป็น 1 และเปลี่ยนค่า bit เป็น 0

คำสั่งควบคุมการทำงานโปรแกรม (Program and Machine Control)

CLR	C	ให้ทำการเคลียร์ค่าภายในแฟลกทดให้เป็นค่า 0

ACALL	addr11	ไปทำโปรแกรมย่อยจากแอดเดรส 11 บิต
LCALL	addr16	ไปทำโปรแกรมย่อยจากแอดเดรส 16 บิต
RET		คำสั่งสิ้นสุดการทำงานโปรแกรมย่อย
RETI		คำสั่งสิ้นสุดการทำงานโปรแกรมย่อยอินเทอร์รัปต์
AJMP	addr11	ให้กระโดดไปยังตำแหน่ง addr 11 โดยใช้ค่าของตัวเลขสัมบรูณ์เพียง 11 บิตและไม่มีเงื่อนไขการกระโดด
LJMP	addr11	ให้กระโดดไปยังตำแหน่ง addr 16 โดยใช้ค่าของตัวเลขสัมบรูณ์เพียง 16 บิต และไม่มีเงื่อนไขการกระโดด
SJMP	rel	ให้กระโดดไปยังตำแหน่งสัมพันธ์ rel โดยไม่มีเงื่อนไข
JMP	@A+DPTR	ให้กระโดดไปยังหน่วยความจำโปรแกรม โดยพิจารณาตำแหน่งจากการบวกค่ารีจิสเตอร์ A กับค่าของรีจิสเตอร์
JZ	rel	ให้กระโดดไปยังหน่วยความจำโปรแกรมตำแหน่งสัมพันธ์ rel เมื่อค่าของรีจิสเตอร์ A เป็น 0
JNZ	rel	ให้กระโดดไปยังหน่วยความจำโปรแกรมตำแหน่งสัมพันธ์ rel เมื่อค่าของรีจิสเตอร์ A ไม่เป็น 0
CJNE	A,data,rel	ให้ทำการเปรียบเทียบ รีจิสเตอร์ A กับค่าของ ตัวเลขdata เมื่อมีค่าไม่เท่ากัน จึงกระโดดไปยังหน่วยความจำโปรแกรมตำแหน่งสัมพันธ์ rel พร้อมทั้งพิจารณาค่าของตัวเลขทั้งสองด้วยว่า ถ้าค่าในรีจิสเตอร์ A น้อยกว่า ค่าตัวเลขของ data ก็ให้กำหนดค่าของแฟ็ลกทดเป็น 1 หากกรณีตรงข้าม ให้กำหนดค่าของแฟ็ลกทดเป็น 0
CJNE	Rn,#data,rel	ให้ทำการเปรียบเทียบ รีจิสเตอร์ R0-R7 กับค่าของเลข data เมื่อมีค่าไม่เท่ากัน จึงกระโดดไปยังหน่วยความจำโปรแกรมตำแหน่งสัมพันธ์ rel พร้อมทั้งพิจารณาค่าของตัวเลขทั้งสองด้วยว่า ถ้าค่าในรีจิสเตอร์ R0-R7 น้อยกว่า ค่าตัวเลขของ data ก็ให้กำหนดค่าของแฟ็ลกทดเป็น 1 หากกรณีตรงข้าม ให้กำหนดค่าของแฟ็ลกทดเป็น 0
CJNE	@Ri,#data,rel	ให้ทำการเปรียบเทียบค่าของหน่วยความจำ กับค่าของตัวเลข data ที่มีเก็บอยู่ในรีจิสเตอร์ R0 หรือ R1 กับค่าของเลข data เมื่อมีค่าไม่เท่ากันจึงกระโดดไปยังหน่วยความจำโปรแกรมตำแหน่งสัมพันธ์ rel พร้อมทั้งพิจารณาค่าของตัวเลขทั้งสองด้วยว่า ถ้าค่าในรีจิสเตอร์ R0-R7 น้อยกว่า ค่าตัวเลขของ

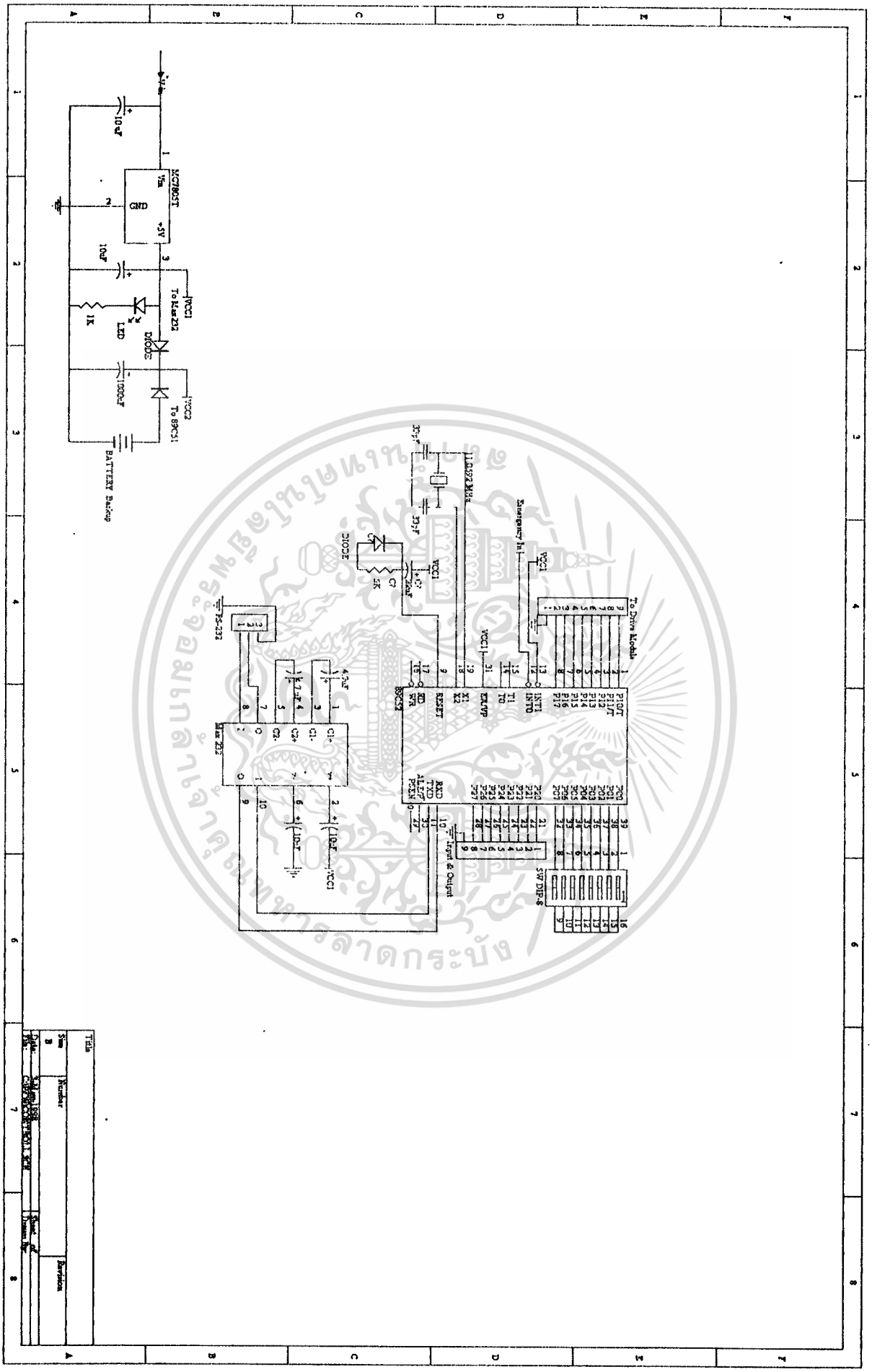
		data ก็ให้ กำหนดค่าของแฟ็ลก เป็น 1 หากกรณีตรงข้ามให้ กำหนดค่า ของแฟ็ลก ทดเป็น 0
DJNZ	Rn,rel	ลดค่าใน Rn และกระโดดไปยังตำแหน่งที่สัมพัทธ์กับตแหน่ง ปัจจุบัน ถ้าค่าไม่เป็น 0
DJNZ	direct,rel	ลดค่าในหน่วยความจำ direct และกระโดดไปยังตำแหน่ง ที่สัมพัทธ์กับตำแหน่งปัจจุบัน ถ้าค่าไม่เป็น 0
NOP		ไม่มีการทำงานใดๆทั้งสิ้น

บทที่ 3
การออกแบบและการสร้าง

3.1 การออกแบบ Controller

วิทยานิพนธ์ฉบับนี้ได้เลือกใช้ระบบคอนโทรลเลอร์ตระกูล MCS-51 เพราะผู้ออกแบบมีความรู้ความเข้าใจในการใช้งานเป็นอย่างดี และเลือกใช้ Microcontroller ของบริษัท Atmel เบอร์ 89C52 เพราะสามารถหาซื้อได้ง่ายในท้องตลาด มีความสามารถสูง สามารถโปรแกรมใหม่ได้ถึง 1000 ครั้ง ผ่านหน่วยความจำแบบ Flash ทำให้ง่ายในการพัฒนาและแก้โปรแกรม มีโหมดประหยัดพลังงาน ทำให้สามารถคงข้อมูลไว้ในหน่วยความจำข้อมูลภายใน chip ได้ โดยเพิ่มวงจรส่วน Backup เพียงเล็กน้อยเท่านั้น ดังแสดงในรูป 3.1 และมี IC เบอร์ MAX 232 ช่วยแปลงระดับสัญญาณ TTL ให้เป็นระดับสัญญาณ ตามมาตรฐาน RS-232 ส่วนที่ขา INT1 ของ CPU นั้น หากมีการปิดเครื่อง หรือระบบไฟขัดข้องอันเป็นเหตุให้ระดับแรงดันลดต่ำลงก็จะทำให้ มีสัญญาณไป Interupp ตัว CPU ให้ Set ตัวเองให้อยู่ในโหมดประหยัดพลังงาน และเมื่อมีการเปิดเครื่องใหม่ ก็จะเกิดการ Reset CPU ทำให้ CPU กลับมาทำงานอีกครั้ง

Port 0 นั้น จะต่อกับ Dip Switch เพื่อเป็นตัว Set ว่า จะให้ Controller ทำงานในโหมดใด เป็นตัวที่เท่าใดของระบบ ส่วนของ Port 1 นั้นก็จะต่อไปยังวงจรไดร์ฟเพื่อขับสเตปปิงมอเตอร์โดยจะอธิบายวงจรไดร์ฟในภายหลัง และ Port 2 ก็จะเป็นส่วนของ I/O อเนกประสงค์ซึ่งผู้ใช้สามารถกำหนดได้ในโปรแกรม



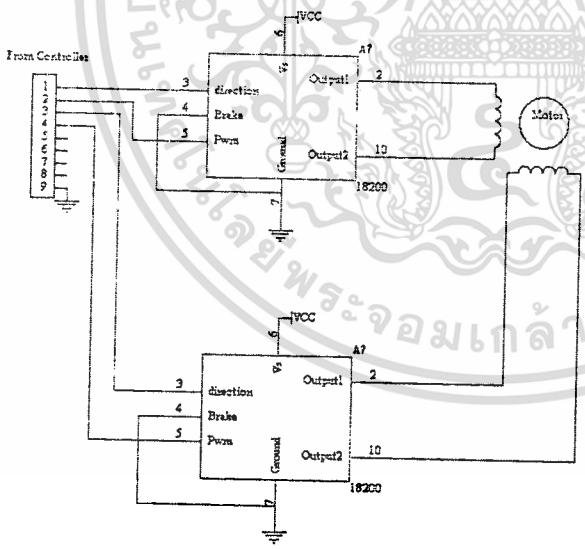
รูปที่ 3.1 แสดงวงจรส่วนของ คอนโทรลเลอร์ที่ได้ออกแบบไว้

3.2 การออกแบบไดรเวอร์

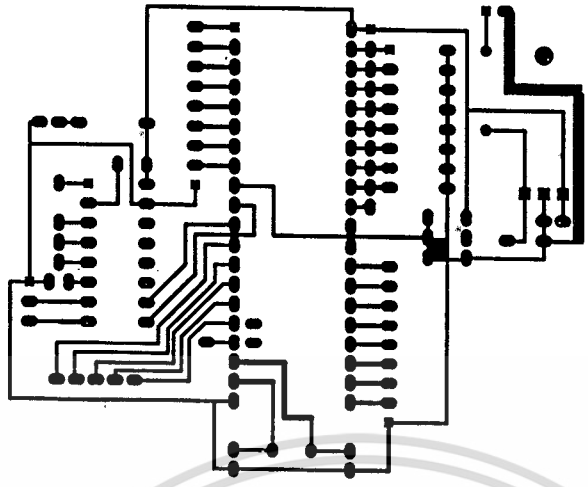
ในส่วนของไดรเวอร์นั้นได้ออกแบบโดยกำหนดคุณสมบัติดังนี้

1. เป็นวงจรไดรฟ์แบบ บริดจ์ ใช้แหล่งจ่ายเพียงแหล่งเดียว
2. ใช้วิธีไดร์ฟแบบ Half Step เพื่อเพิ่มความละเอียดได้อีก 1 เท่า
3. ใช้กับสเตปปิงมอเตอร์แบบ 2 เฟส 200 สเตป/รอบ
4. โวลท์เตจที่ป้อนให้กับวงจรอยู่ในช่วง 12-50 Volts
5. สามารถขับกระแสได้ 3 แอมป์ และกระแสสูงสุดที่ 6 แอมป์ (Peak)
6. สัญญาณอินพุตที่ป้อนให้เป็นสัญญาณ Logic CMOS

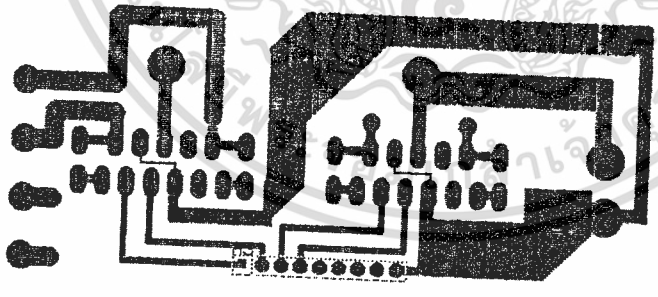
จากข้อกำหนดดังกล่าว จึงได้ออกแบบวงจรไดร์ฟ ได้ดังรูป 3.2



รูป 3.2 แสดงวงจรไดร์ฟที่ออกแบบไว้



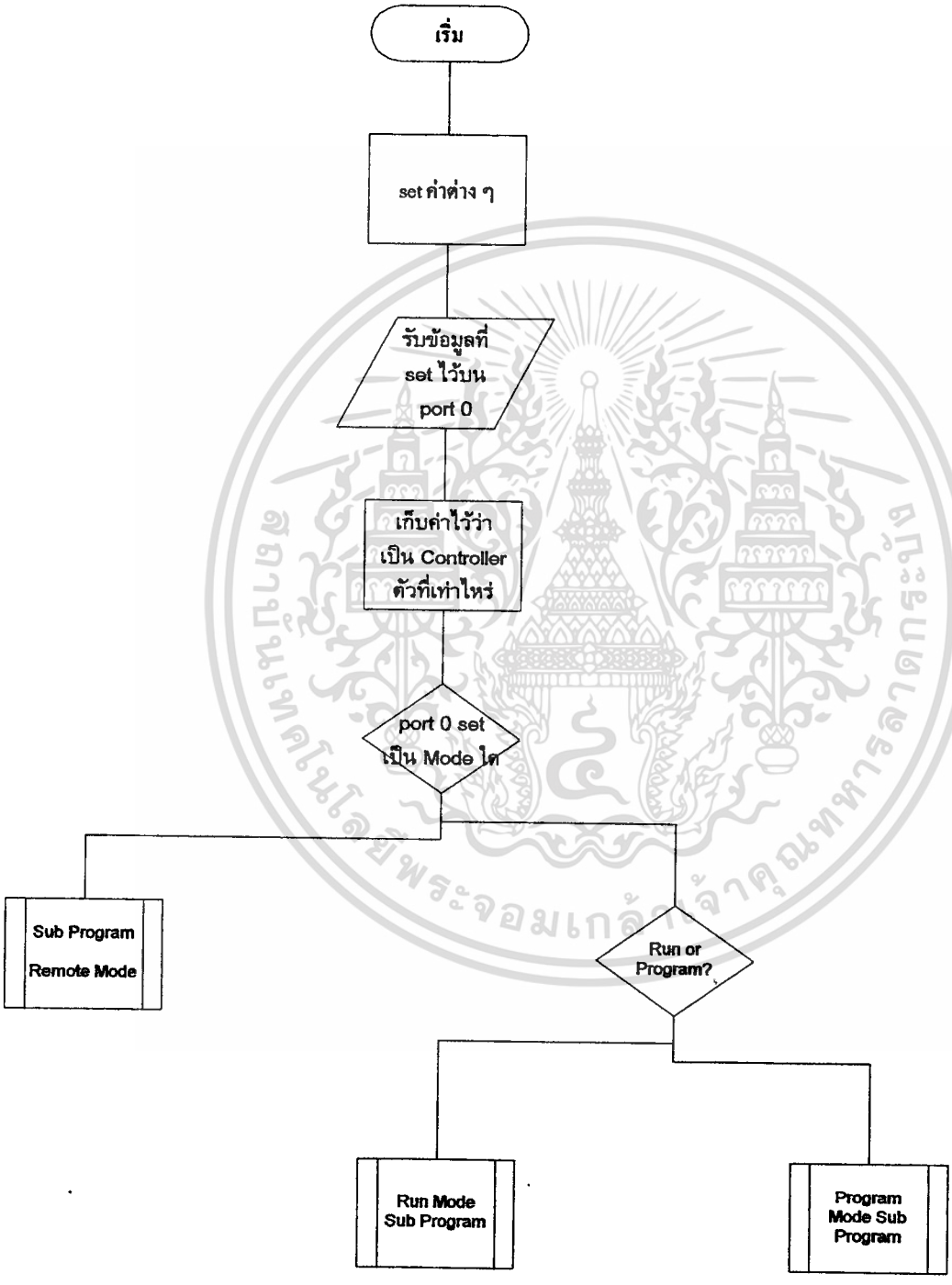
รูปที่ 3.3 แสดงลายแผนวงจรพิมพ์ของคอนโทรลเลอร์ที่ออกแบบไว้



รูปที่ 3.4 แสดงลายแผ่นวงจรพิมพ์ของวงจรไดร์ฟที่ออกแบบไว้

3.3 การออกแบบส่วนของโปรแกรม

ในส่วนของตัวโปรแกรมนั้น ได้ใช้ภาษา Assembly ของ MCS-51 เขียนขึ้น โดยมี Flow Chart ดังนี้



รูปที่ 3.5 แสดงโฟลว์ชาร์ตของโปรแกรมหลัก

บทที่ 4
การนำไปใช้งาน

4.1 ชุดคำสั่งของตัว Controller และตัวอย่างการใช้งาน

ชุดคำสั่งของ Controller ที่ได้ออกแบบไว้ มีรูปแบบหลัก ๆ 3 รูปแบบคือ

4.1.1 Input Command เป็นชุดที่ใช้ควบคุมการอ่านค่าของข้อมูลโดยจะประกอบด้วยบิตของพอร์ตและ Operator อันได้แก่ '&' (AND) , '*' (OR) , 'I' (INVERT) โดยรูปแบบจะยกตัวอย่างเช่น " 1&2*3I* " ก็หมายถึง ค่าของ Bit 1 AND กับ Bit 2 OR กับ Invert ของ 3 ใช้ในการควบคุมการทำงานของฟังก์ชันส่วนของ Step Motion Command

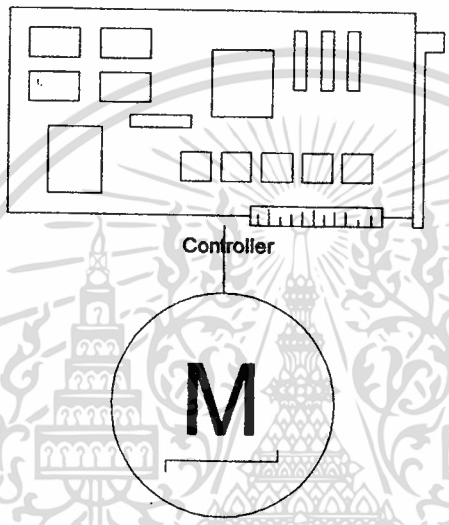
4.1.2 Step Motion Command เป็นชุดที่ใช้ควบคุมการหมุนของตัวสเตปปิ้งมอเตอร์จะประกอบไปด้วยฟังก์ชั่นของการหมุนของตัวมอเตอร์ ซึ่งประกอบไปด้วย ทิศทาง จำนวน Step ความเร็ว อัตราการเร่งจากเริ่มต้น โดยรูปแบบจะเป็น " SxxxxxVxxxAxxDxTxxxOxxx " โดยจะอธิบายได้ดังนี้ หลังตัว S จะหมายถึงจำนวน Step ที่จะหมุนซึ่งจะมีค่าอยู่ระหว่าง 1-65535 Step หลังตัว V หมายถึงความเร็วรอบของมอเตอร์ ซึ่งอยู่ระหว่าง 1-255 rpm หลังตัว A ซึ่งหมายถึงความเร่ง ซึ่งมีค่าตั้งแต่ 0-99 โดยที่ 0 หมายถึงไม่ต้องมีความเร่ง คือเริ่มออกตัวที่ความเร็ว V ทันที หลังตัว D หากเป็น 1 จะหมายถึงการหมุนตามเข็มนาฬิกา ส่วน 0 จะหมายถึงหมุนทวนเข็มนาฬิกา (ขึ้นกับการเดินสายที่ตัวมอเตอร์กับวงจรไดร์ฟด้วย) ส่วนหลังตัว T หมายถึง Timer ที่จะ Delay ค่าไว้ก่อนที่จะเริ่มทำงาน

4.1.3 Output Command เป็นการส่ง Output ออกมาหลังจากที่มอเตอร์หมุนเสร็จหมดแล้ว โดยรูปแบบจะเป็น "Ox" หมายถึง แสดงสถานะ High ให้บิตที่ x หรือ "OxI" หมายถึง แสดงสถานะ Low ที่บิตที่ x โดยค่าเหล่านี้ จะ Latch ไว้จนกว่าจะมีการเปลี่ยนแปลงต่อไป

ตัวอย่างของรูปแบบคำสั่งยกตัวอย่างเช่น 1&3I*5D1S6000V50A0O4 ตามด้วยรหัสของปุ่ม Enter หรือ 0Dh 0Ah ก็จะหมายความว่า มอเตอร์จะหมุนด้วยความเร็ว 50 รอบต่อนาที ด้วยระยะทาง 6000 สเตป หมุนตามเข็มนาฬิกา ได้ก็ต่อเมื่อ มีอินพุตที่ บิทที่ 1 ของ I/O AND กับ บิทที่ 3 หรือ บิทที่ 5 เมื่อหมุนเสร็จแล้ว ก็จะแสดงสถานะสูงที่ บิทที่ 4

4.2 การนำไปใช้ในส่วน Stand Alone Mode

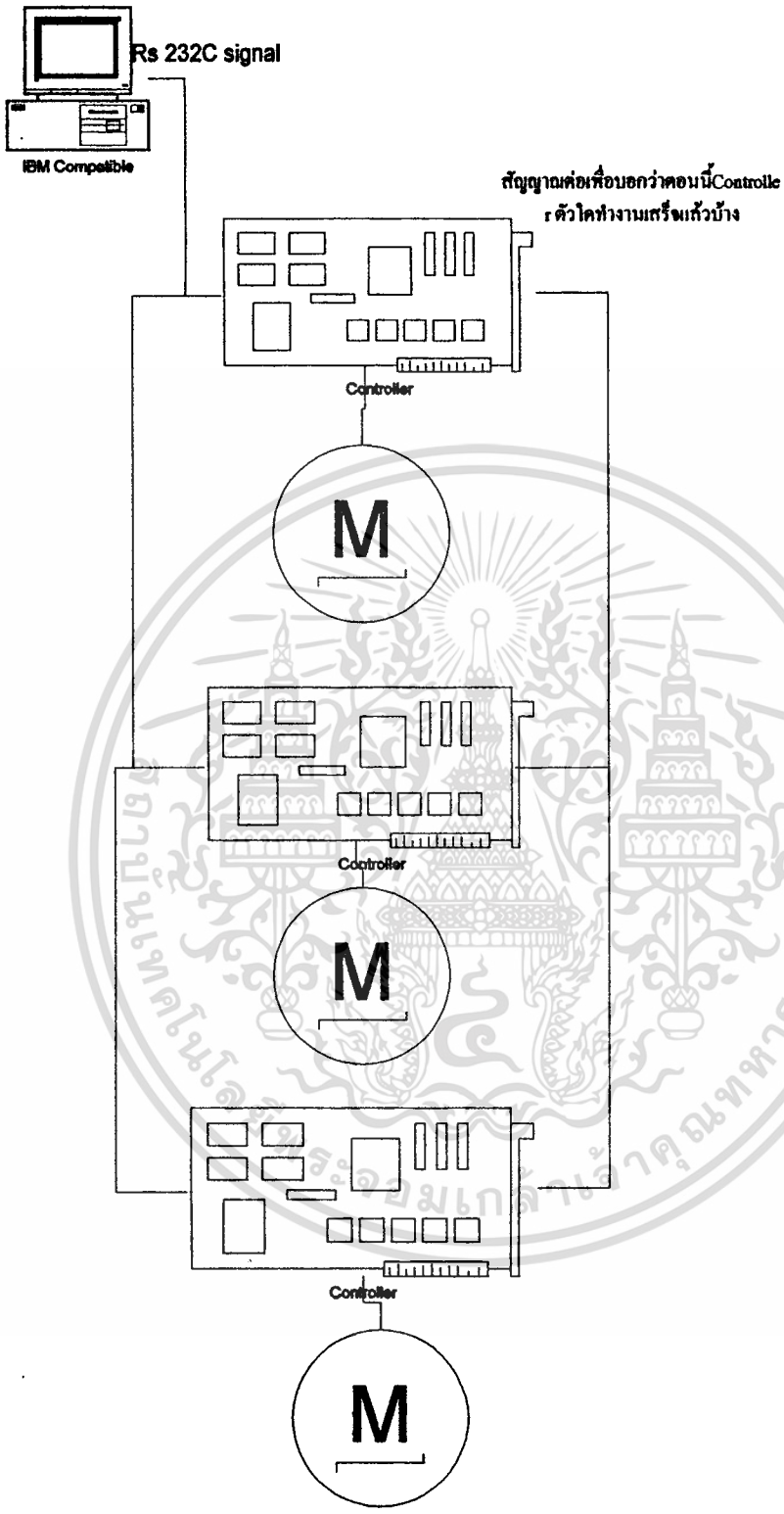
ในส่วนนี้จะต้องทำการ Set Dip Switch มาที่ Receive Mode ก่อนเพื่อรับโปรแกรมคำสั่งเริ่มต้น โดยที่ ตัวส่งโปรแกรมนั้นจะใช้ คอมพิวเตอร์ส่ง File Text Format ผ่านทาง พอร์ตอนุกรม RS-232C โดยทำการเซ็ตค่า ความเร็วไว้ที่ 9600 บิตต่อวินาที ไม่มี Parity และ มี 1 Stop Bit เมื่อส่งโปรแกรมมาแล้ว ก็ทำการ Set Dip Switch เป็น Run Mode เท่านี้ ตัว Controller ก็จะทำงานตาม Sequence ที่เราสั่งไว้



ภาพ 4.1 แสดงการใช้งานส่วน Stand Alone Mode

4.3 การนำไปใช้ในส่วน Remote Mode

ในส่วนนี้จะใช้ Dip Switch Set มาที่ Remote Mode โดยจะต้อง Set ว่า Controller ที่เราจะต่อนั้นต่อเป็นตัวที่เท่าไรอยู่เพื่อประโยชน์ในการต่อร่วมกันหลาย ๆ ตัวของ Controller และเมื่อเริ่มการทำงาน ตัว Controller จะรอรับสัญญาณจากคอมพิวเตอร์ หากไม่ใช่สัญญาณของตนก็จะไม่เก็บไว้ เมื่อได้รับสัญญาณครบแล้ว ก็จะรอสัญญาณเริ่มต้นเพื่อเริ่มทำงานพร้อม ๆ กัน จากนั้น คอนโทรลเลอร์ตัวที่ 1 ก็จะตรวจสอบ ตัวที่ 2 ตัวที่ 2 ก็จะตรวจสอบตัว ต่อ ๆ ไป เพื่อเช็คดูว่าทำงานเสร็จหรือยัง เมื่อทำงานเสร็จหมดแล้ว คอนโทรลเลอร์ตัวที่ 1 ก็จะส่งสัญญาณอนุกรมกลับไปเพื่อบอกว่าทำงานเสร็จแล้ว ทำให้ Computer เริ่มส่งสัญญาณชุดใหม่ต่อไป



การนำไปต่อใช้งานกรณีใช้งานใน Remote Mode

บทที่ 5
ข้อเสนอแนะและวิจารณ์

5.1 ปัญหาและจุดบกพร่องของระบบ

ในการทำปริญญานิพนธ์นี้มุ่งเน้นจะศึกษาในส่วนการควบคุมโดยมอเตอร์แบบสเตปปิง โดยใช้คำสั่งที่เข้าใจได้ง่าย และสามารถโปรแกรมใหม่ได้ โดยใช้วิธีขับสเตปปิงที่มีศักยภาพดีในระดับที่ใช้งานกันทั่วไป ไม่ได้คำนึงถึงวิธีการขับสเตปปิงมอเตอร์โดยใช้เทคนิคที่ซับซ้อน ที่จะทำให้มีการประหยัดพลังงานมากกว่านี้ หรือ มีความละเอียดสูงกว่านี้ จึงนับเป็นจุดบกพร่องอย่างหนึ่งของระบบ แต่ก็เป็นจุดที่นอกเหนือจากวัตถุประสงค์ที่วางไว้

5.2 แนวทางการพัฒนาต่อ

จากข้อ 5.1 ดังที่กล่าวมาแล้วนั้น จะเห็นได้ว่าแนวทางการพัฒนาต่อก็คือ การเพิ่มศักยภาพของระบบให้ดียิ่งขึ้น เนื่องจากระบบยังไม่ได้นำเทคนิคบางอย่าง ที่มีการพัฒนาไว้แล้วมาใช้ยกตัวอย่างเช่น การใช้เทคนิค MicroStep หรือการใช้เทคนิค การ Modulation กับสัญญาณ Sine เพื่อให้การหมุนต่อเนื่องและราบเรียบขึ้น หรือการพัฒนาให้สามารถขับภาระเชิงกลได้มากขึ้น และยังมีแนวทางทางด้านโปรแกรมอีก คล่าวคือ พัฒนาโปรแกรมให้เป็นไปในแนว Visual มากขึ้น ซึ่งถ้าหากทำสำเร็จไม่ว่าใครก็สามารถสั่งให้มอเตอร์หมุนได้ตามใจโดยไม่ต้องศึกษา Command ก่อนเลย เพราะ การติดต่อด้วย ICON ทำให้ง่ายต่อการเข้าใจ

5.3 สรุป

อุปกรณ์ที่เราสร้างขึ้นมานั้นสามารถควบคุมสเตปปิ้งมอเตอร์ได้โดยใช้ภาษาที่ง่ายแก่ความเข้าใจมากขึ้น และสามารถเปลี่ยนแปลง Sequence ได้ง่าย ทำให้สะดวกต่อการนำไปประยุกต์ใช้งานสร้างเครื่องมือกล ต่าง ๆ ได้เป็นอย่างดี

ภาคผนวก

รายละเอียดของไอซีที่ใช้ในโครงงาน

```
; FILENAME     teststep.ASM
; DESCRIPTION   stepping motor
;controller
; HARDWARE     chanop step
; ASSEMBLER    SXA51
; START-DATE   12/2/42
; SOFTWARE ENG. CHANOP S.
; COMPANY      Kmit'l
;** set constant for my project
ubeep        equ 00a5h
step1        equ 00000010b
step2        equ 00001010b
step3        equ 00001000b
step4        equ 00001011b
step5        equ 00000011b
step6        equ 00001111b
step7        equ 00001100b
step8        equ 00001110b
drivebuff    equ 18h
m            equ 1ah  ; acceration in hex
n            equ 19h  ; speed in hex
direction    equ 31h  ;
halfsteph    equ 1bh
halfstepl    equ 1ch
decrement    equ 1dh
fixsteph     equ 1eh
fixstepl     equ 1fh
summary      equ 20h  ; total in speed
vmax         equ 21h  ; maximum speed
sum_acc      equ 22h  ; sum in velocity
ansl         equ 23h
ansh         equ 24h
flath        equ 25h
flatl        equ 26h
steph        equ 27h
stepl        equ 28h
wait_time    equ 29h
out_value    equ 2ah
first_locate equ 2bh
iden_nip     equ 2ch
;********************** set value at boot
up ******************************
             org 0000h
             jmp 0040h
             org 000bh ;timer 0
             jmp int_timer0
             org 0040h ;
initial:     mov scon,#50h
             mov tmod,#21h
             mov th1,#0fdh
             setb tr1
             mov th0,#0feh
             mov tl0,#00h
             setb ea
             setb et0
             mov r0,#80h
             mov first_locate,#80h
             mov out_value,#00h
;***************** end of setup
*********************

;******************* main menu
**********************
             mov c,p0.3
             jc not_remote
             jmp remote_mode
not_remote:  mov c,p0.2
             jc start_recv
```

```
jmp single_mode
;*** Big procedure *** to recieve from
;RS-232 to run mode*
start_recv:    mov r1,#7fh
               mov r0,#80h
               mov a,#00h

clearoldvalue: mov @r0,a
               inc r0
               djnz r1,clearoldvalue
               mov r0,#80h
               mov first_locate,#80h
not_65d:       call m_reciv
               cjne a,#65d,not_65d

new:           call m_reciv   ; decode call
zero2nine
               mov r1,b
               cjne r1,#00h,not_number
               mov r2,a
               mov a,@r0
               orl a,r2
               mov @r0,a
               jmp new
not_number:    cjne a,#83d,not_step
;Check Character 'S'
               jmp cache_step

not_step:      cjne a,#38d,not_and_oper
; Check &
               mov r0,first_locate
               jmp new
not_and_oper:  cjne
a,#42d,not_or_oper  ; Check *
               mov r0,first_locate
               inc r0
               jmp new

not_or_oper:   cjne a,#73d,cache_step
; check i
               call m_reciv
               cjne a,#38d,not_andl_oper  ;
check l&
               mov r0,first_locate
               inc r0
               inc r0
               jmp new
not_andl_oper: cjne
a,#42d,cache_step     ; check l*
               mov r0,first_locate
               inc r0
               inc r0
               inc r0
               jmp new
cache_step:

               call m_reciv
               mov r5,a
               call m_reciv
               mov r4,a
               call m_reciv
               mov r3,a
               call m_reciv
               mov r2,a
               call m_reciv
```

```
        mov r1,a
        call dec2hex
        mov r0,first_locate
        inc r0
        inc r0
        inc r0
        inc r0
        mov a,r7
        mov @r0,a
        mov a,r6
        inc r0
        mov @r0,a

cache_V:    call m_reciv
        call m_reciv
        mov r3,a
        call m_reciv
        mov r2,a
        call m_reciv
        mov r1,a
        call dec2hex8
        inc r0
        mov @r0,a

cache_A:    call m_reciv
        call m_reciv
        mov r3,a
        call m_reciv
        mov r2,a
        call m_reciv
        mov r1,a
        call dec2hex8
        inc r0
        mov @r0,a

cache_direction:call m_reciv
        call m_reciv
        cjne a,#30h,cache_delay
        mov a,@r0
        setb acc.7
        mov @r0,a

cache_delay:  call m_reciv
        call m_reciv
        mov r3,a
        call m_reciv
        mov r2,a
        call m_reciv
        mov r1,a
        call dec2hex8
        inc r0
        mov @r0,a
cache_out:    call m_reciv
        call m_reciv
        mov r3,a
        call m_reciv
        mov r2,a
        call m_reciv
        mov r1,a
        call dec2hex8
        inc r0
        mov @r0,a

        inc r0
        mov first_locate,r0
        cjne r0,#0f8h,not_full
        sjmp $
not_full:   jmp not_65d
```

```
single_mode:
        mov r0,first_locate
        mov a,@r0
        cjne a,#00h,have_algo
        jmp next_loop
have_algo:  cpl a
        mov b,a
        ;mov dptr,#0b000h
        ;movx a,@dptr   ; such as p2
        mov a,p2
        orl a,b
        mov b,a
        mov r1,a

        inc r0
        mov a,@r0
        mov b,a
        ; mov dptr,#0b000h
        ; movx a,@dptr
        mov a,p2
        anl a,b
        jnz starter

;       mov a,r1
;       cjne a,#0ffh,fail_run
;       jmp and_not
and_not:
        inc r0
        mov a,@r0
        cpl a
        mov b,a
        ;mov dptr,#0b000h
        ;movx a,@dptr
        mov a,p2
        cpl a
        orl a,b
        anl a,r1
        mov r1,a
        cjne a,#0ffh,or_not
starter:    jmp prepare_drive
or_not:     inc r0
        mov a,@r0
        mov b,a
        ;mov dptr,#0b000h
        ;movx a,@dptr
        mov a,p2
        cpl a
        anl a,b
        cjne a,#00h,starter
        jmp next_loop

prepare_drive:
        mov r0,first_locate
        inc r0
        inc r0
        inc r0
        inc r0      ; point steph
        mov a,@r0
        mov steph,a
        dec a
        mov fixsteph,a
        inc r0      ; point stepl
        mov a,@r0
        mov stepl,a
        dec a
        mov fixstepl,a
        inc r0      ; point velocity
        mov a,@r0
```

```
            mov vmax,a
            inc r0
            mov a,@r0
            mov c,acc.7
            jnc neo_direct
            mov direction,#00h
            jmp preat_A
neo_direct:  mov direction,#30h
preat_a:     clr acc.7
            mov m,a
            inc r0
            mov a,@r0
            mov wait_time,a
            inc r0
            mov a,@r0
            anl a,#00001111b
            swap a
            mov out_value,a
            call drive_system

wait_for_drv: mov r1,b
            cjne r1,#00h,wait_for_drv

next_loop:   mov r0,first_locate
            cjne r0,#0f8h,repeat_again
            mov first_locate,#80h
            jmp single_mode
repeat_again: inc r0
            inc r0
            inc r0
            inc r0
            inc r0
            inc r0
            inc r0
            inc r0
            inc r0
            inc r0
            mov first_locate,r0
            jmp single_mode

remote_mode:
initial_rem:
            mov a,#00h
            anl a,#11110000b
            swap a
            add a,#65d
            mov iden_nip,a
begin_wait:  call m_reciv
            cjne a,iden_nip,begin_wait
            call m_reciv
            cjne a,#83d,begin_wait
            call m_reciv
            mov r5,a
            call m_reciv
            mov r4,a
            call m_reciv
            mov r3,a
            call m_reciv
            mov r2,a
            call m_reciv
            mov r1,a
            call dec2hex
            mov steph,r7
            mov a,r7
            dec a
            mov fixsteph,a
            mov stepl,r6
            mov a,r6
```

```
        dec a
        mov fixstepl,a
        call m_reciv ;Velocity
        call m_reciv
        mov r3,a
        call m_reciv
        mov r2,a
        call m_reciv
        mov r1,a
        call dec2hex8
        mov Vmax,a

        call m_reciv
        call m_reciv
        mov r3,a
        call m_reciv
        mov r2,a
        call m_reciv
        mov r1,a
        call dec2hex8
        mov m,a
        call m_reciv
        call m_reciv
        cjne a,#30h,invertum
        mov direction,#30h
        jmp turnonly
invertum:   mov direction,#00h

turnonly:   call m_reciv
        call m_reciv
        mov r3,a
        call m_reciv
        mov r2,a
        call m_reciv
        mov r1,a
        call dec2hex8
        mov wait_time,a
        call m_reciv
        call m_reciv
        mov r3,a
        call m_reciv
        mov r2,a
        call m_reciv
        mov r1,a
        call dec2hex8

        anl a,#00001111b
        swap a
        mov out_value,a
check_Z:    call m_reciv
        cjne a,#90d,check_Z
        call drive_System
        jmp remote_mode
drive_system:  mov r0,wait_time
        inc r0
        mov b,#0ffh
        cjne r0,#01h,waittime
        jmp zerowait
waittime:   call delay_01
        djnz r0,waittime
zerowait:   mov decrement,#00h
        mov sum_acc,#00h
        mov n,#01h
        mov dptr,#0000h
        clr c
        mov a,steph
        dec a
        rrc a
```

```
            inc a
            mov halfsteph,a
            mov a,stepl
            dec a
            rrc a
            inc a
            mov halfstepl,a
            setb tr0
            ret

int_timer0:  clr tr0
            mov a,b
            cjne a,#0ffh,not_0step
            mov a,steph
            cjne a,#01,not_0step
            mov a,stepl
            cjne a,#00h,not_0step

;           mov a,out_value
;           orl a,p2
;           mov p2,a
;           call delay_01
            mov b,#0h
            reti
not_0step:  mov b,#0fh
            clr c
            mov a,summary
            add a,n
            mov summary,a
            jnc no_carry
            call drive_action
            inc dptr
            call count_down
            call chk_acc
            mov r6,steph
            mov r7,stepl
            mov a,r6
            cjne a,halfsteph,fail_check
            mov a,r7
            cjne a,halfstepl,fail_check
            mov a,decrement
            cjne a,#00h,fail_check
            mov decrement,#0ffh     ;
                                    begin decrement Velocity
fail_check:

fail_check2:  djnz r7,notyet
            djnz r6,notyet
            mov b,#0h
            reti

notyet:     mov steph,r6
            mov stepl,r7
no_carry:   mov th0,#(0feh)
            mov tl0,#(00h)
            call chk_acc
            setb tr0
            reti

count_down:
            mov r3,ansl
            mov r2,ansh
            mov a,r3
            orl a,r2
            jz not_oper
            djnz r3,not_overflow
```

```
        djnz r2,not_overflow
        mov decrement,#0ffh
not_overflow:  mov ansl,r3
        mov ansh,r2
not_oper:    ret
chk_acc:     mov a,m    ; When
acceleration is 0 Drive will max speed
        jnz cont1
        mov n,vmax

cont1:       mov a,vmax
        cjne a,n,cont
        mov a,decrement
        inc a
        jz cont

        mov a,decrement
        cjne a,#00h,not_save
        mov a,#0fh
        mov decrement,a
        mov r5,dph  ; Store value that
finish Vmax
        mov r4,dpl
        mov a,r4
        clr c
        rlc a
        mov flatl,a
        mov a,r5
        rlc a
        mov flath,a
        mov a,fixstepl
        clr c
        subb a,flatl
        inc a
        mov ansl,a
        mov a,fixsteph
        subb a,flath
        inc a
        mov ansh,a
not_save:    ret

cont:        clr c
        mov a,sum_acc
        add a,m
        mov sum_acc,a
        jc inc_veacc
        ret
inc_veacc:     mov a,decrement
        cjne a,#0ffh,inc_v
        mov a,n
        dec a
        mov n,a
        ret
inc_v:       mov a,n
        inc a
        mov n,a
        ret

;Xxl        equ r0
;xxh        equ r1
;xx3        equ r2
;yyl        equ r3
;ansl       equ r4
;ansh       equ r5
;ans3       equ r6
div24:       mov r7,#8h
        mov a,r2
        mov b,r3
```

```
            div ab
            mov r6,a
di_loop1:   mov a,r1
            mov c,acc.7
            rl a
            mov r1,a
            mov a,b
            rlc a
            mov b,r3
            div ab
            jz notinc
            mov a,r5
            setb acc.0
            rl a
            mov r5,a
            jmp div_ok
notinc:     mov a,r5
            clr acc.0
            rl a
            mov r5,a
div_ok:     djnz r7,di_loop1
            rr a
            mov r5,a
            mov r7,#8h
di_loop2:   mov a,r0
            mov c,acc.7
            rl a
            mov r0,a
            mov a,b
            rlc a
            mov b,r3
            div ab
            jz notinc2
            mov a,r4
            setb acc.0
            rl a
            mov r4,a
            jmp div_ok2
notinc2:    mov a,r4
            clr acc.0
            rl a
            mov r4,a
div_ok2:    djnz r7,di_loop2
            rr a
            mov r4,a
            ret
dec2hex:    mov dptr,#0000h
            mov r0,#01h
            mov b,#07h
condec:     mov a,@r0
            call conasci
            mov @r0,a
            inc r0
            djnz b,condec
            mov a,r2
            mov b,#10d
            mul ab
            add a,r1
            mov dpl,a
            mov a,r3
            mov b,#100d
            mul ab
            clr c
            add a,dpl
            mov dpl,a
            jnc next_r4
            inc b
next_r4:    mov dph,b
```

```
        mov a,r4
        mov b,#0e8h
        mul ab
        clr c
        add a,dpl
        mov dpl,a
        mov a,b
        addc a,dph
        mov dph,a
        mov a,r4
        mov b,#03h
        mul ab
        add a,dph
        mov dph,a
        mov a,r5
        mov b,#10h
        mul ab
        clr c
        add a,dpl
        mov dpl,a
        mov a,b
        addc a,dph
        mov dph,a
        mov a,r5
        mov b,#27h
        mul ab
        add a,dph
        mov dph,a
        inc dph
        mov r6,dpl
        mov r7,dph
        ret
conasci:  clr c
        subb a,#30h
        ret
;r1,r2,r3  is ascii code 0-255
;out put is A
dec2hex8:   mov a,r3
        clr c
        subb a,#30h
        mov r3,a
        mov a,r2
        clr c
        subb a,#30h
        mov r2,a
        mov a,r1
        clr c
        subb a,#30h
        mov r1,a
        mov a,r3
        mov b,#100d
        mul ab
        mov r3,a
        mov a,r2
        mov b,#10d
        mul ab
        add a,r3
        add a,r1
        ret
drive_action:  mov a,direction
        cjne a,#30h,drv_cw      ;!!!!
ASCII
        call drive_ccw
        jmp success
drv_cw:     call drive_cw
success:    ret
drive_cw:   mov a,drivebuff
```

```
        cjne a,#step1,nostep1
        mov drivebuff,#step2
        mov a,#step2
        orl a,out_value
        mov p1,a
        ret
nostep1:    cjne a,#step2,nostep2
        mov drivebuff,#step3
        mov a,#step3
        orl a,out_value
        mov p1,a
        ret
nostep2:    cjne a,#step3,nostep3
        mov drivebuff,#step4
        mov a,#step4
        orl a,out_value
        mov p1,a
        ret

nostep3:    cjne a,#step4,nostep4
        mov drivebuff,#step5
        mov a,#step5
        orl a,out_value
        mov p1,a
        ret
nostep4:    cjne a,#step5,nostep5
        mov drivebuff,#step6
        mov a,#step6
        orl a,out_value
        mov p1,a
        ret
nostep5:    cjne a,#step6,nostep6
        mov drivebuff,#step7
        mov a,#step7
        orl a,out_value
        mov p1,a
        ret
nostep6:    cjne a,#step7,nostep7
        mov drivebuff,#step8
        mov a,#step8
        orl a,out_value
        mov p1,a
        ret
nostep7:    cjne a,#step8,nostep8
        mov drivebuff,#step1
        mov a,#step1
        orl a,out_value
        mov p1,a
        ret
nostep8:    mov drivebuff,#step1
        mov a,#step1
        orl a,out_value
        mov p1,a
        ret
drive_ccw:  mov a,drivebuff
        cjne a,#step1,notstep1
        mov drivebuff,#step8
        mov a,#step8
        orl a,out_value
        mov p1,a
        ret
notstep1:   cjne a,#step2,notstep2
        mov drivebuff,#step1
        mov a,#step1
        orl a,out_value
        mov p1,a
        ret
notstep2:   cjne a,#step3,notstep3
```

```
        mov drivebuff,#step2
        mov a,#step2
        orl a,out_value
        mov p1,a
        ret
notstep3:   cjne a,#step4,notstep4
        mov drivebuff,#step3
        mov a,#step3
        orl a,out_value
        mov p1,a
        ret
notstep4:   cjne a,#step5,notstep5
        mov drivebuff,#step4
        mov a,#step4
        orl a,out_value
        mov p1,a
        ret
notstep5:   cjne a,#step6,notstep6
        mov drivebuff,#step5
        mov a,#step5
        orl a,out_value
        mov p1,a
        ret
notstep6:   cjne a,#step7,notstep7
        mov drivebuff,#step6
        mov a,#step6
        orl a,out_value
        mov p1,a
        ret
notstep7:   cjne a,#step8,notstep8
        mov drivebuff,#step7
        mov a,#step7
        orl a,out_value
        mov p1,a
        ret
notstep8:   mov drivebuff,#step8
        mov a,#step8
        orl a,out_value
        mov p1,a
        ret
;****************************** mini reciv
**************************
m_reciv:    jbc ri,m_can_reciv
        sjmp m_reciv
m_can_reciv:  mov a,sbuf
        ret
transmit:   mov sbuf,a
        jnb ti,$
        clr ti
        ret
delay:      mov r6,#10h
        mov r5,#00h
del1:       djnz r5,del1
        djnz r6,del1
        ret
delay_01:   mov r5,#01h
        mov r6,#0b5h
delay_01_01:  djnz r5,delay_01_01
        djnz r6,delay_01_01
        ret
zero2nine:  mov b,#00h
        cjne a,#48d,zeronotzero
        mov a,#00d
        ret
zeronotzero:  cjne a,#49d,zeronotone
        mov a,#01h
        ret
zeronotone:   cjne a,#50d,zeronottwo
```

```
            mov a,#02h
            ret
zeronottwo:   cjne a,#51d,zeronotthree
            mov a,#04h
            ret
zeronotthree:  cjne a,#52d,zeronotfour
            mov a,#08h
            ret
zeronotfour:   cjne a,#53d,zeronotfive
            mov a,#10h
            ret
zeronotfive:   cjne a,#54d,zeronotsix
            mov a,#20h
            ret
zeronotsix:   cjne a,#55d,zeronotseven
            mov a,#40h
            ret
zeronotseven:  cjne
a,#56d,zeronoteight
            mov a,#80h
            ret
zeronoteight:  cjne a,#57d,zeronotnine
            mov a,#00d
            ret
zeronotnine:   mov b,#0ffh
            ret

           end
```

National Semiconductor

April 1998

LMD18201 3A, 55V H-Bridge

General Description

The LMD18201 is a 3A H-Bridge designed for motion control applications. The device is built using a multi-technology process which combines bipolar and CMOS control circuitry with DMOS power devices on the same monolithic structure. The H-Bridge configuration is ideal for driving DC and stepper motors. The LMD18201 accommodates peak output currents up to 6A. Current sensing can be achieved via a small sense resistor connected in series with the power ground lead. For current sensing without disturbing the path of current to the load, the LMD18200 is recommended.

Features

- Delivers up to 3A continuous output
- Operates at supply voltages up to 55V
- Low $R_{DS(ON)}$ typically 0.33Ω per switch
- TTL and CMOS compatible inputs
- No "shoot-through" current
- Thermal warning flag output at 145°C
- Thermal shutdown (outputs off) at 170°C
- Internal clamp diodes
- Shorted load protection
- Internal charge pump with external bootstrap capability

Applications

- DC and stepper motor drives
- Position and velocity servomechanisms
- Factory automation robots
- Numerically controlled machinery
- Computer printers and plotters

Functional Diagram

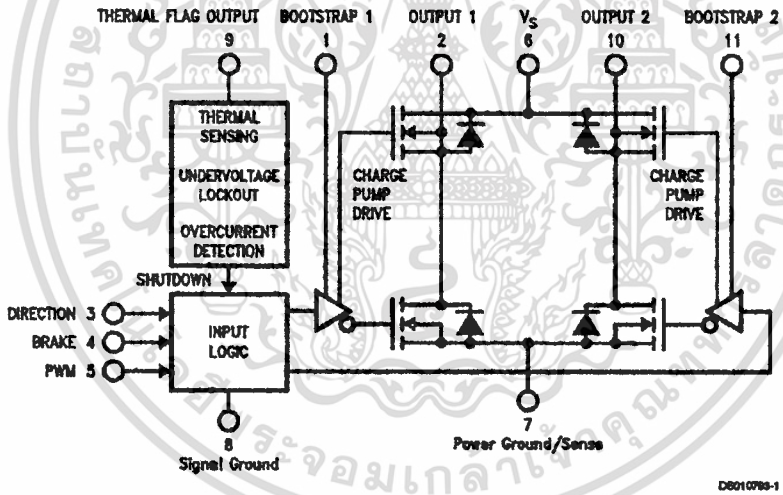
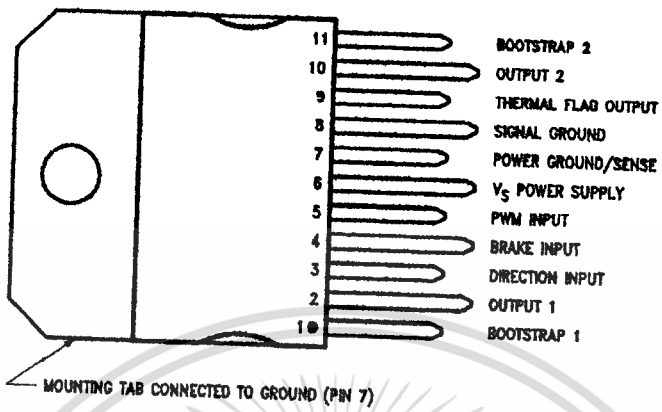


FIGURE 1. Functional Block Diagram of LMD18201

Connection Diagram and Ordering Information



Top View
Order Number LMD18201T
See NS Package Number TA11B

Absolute Maximum Ratings (Note 1)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/ Distributors for availability and specifications.

Total Supply Voltage (V_S, Pin 6)	60V
Voltage at Pins 3, 4, 5 and 9	12V
Voltage at Bootstrap Pins (Pins 1 and 11)	V_{OUT} + 16V
Peak Output Current (200 ms)	6A
Continuous Output Current (Note 2)	3A
Power Dissipation (Note 3)	25W
Sense Voltage (Pin 7 to Pin 8)	+0.5V to −1.0V
Power Dissipation (T_A = 25°C, Free Air)	3W
Junction Temperature, $T_{J(max)}$	150°C
ESD Susceptibility (Note 4)	1500V
Storage Temperature, T_{STG}	−40°C to +150°C
Lead Temperature (Soldering, 10 sec.)	300°C

Operating Ratings (Note 1)

Junction Temperature, T_J	−40°C to +125°C
V_S Supply Voltage	+12V to +55V

Electrical Characteristics (Note 5)

The following specifications apply for V_S = 42V, unless otherwise specified. **Boldface** limits apply over the entire operating temperature range, −40°C ≤ T_J ≤ +125°C, all other limits are for T_A = T_J = 25°C.

Symbol	Parameter	Conditions	Typ	Limit	Units
$R_{DS(ON)}$	Switch ON Resistance	Output Current = 3A (Note 6)	0.33	0.4/0.6	Ω (max)
$R_{DS(ON)}$	Switch ON Resistance	Output Current = 6A (Note 6)	0.33	0.4/0.6	Ω (max)
V_{CLAMP}	Clamp Diode Forward Drop	Clamp Current = 3A (Note 6)	1.2	1.5	V (max)
V_{IL}	Logic Low Input Voltage	Pins 3, 4, 5		−0.1	V (min)
				0.8	V (max)
I_{IL}	Logic Low Input Current	V_{IN} = −0.1V, Pins = 3, 4, 5		−10	µA (max)
V_{IH}	Logic High Input Voltage	Pins 3, 4, 5		2	V (min)
				12	V (max)
I_{IL}	Logic High Input Current	V_{IN} = 12V, Pins = 3, 4, 5		10	µA (max)
	Undervoltage Lockout	Outputs Turn OFF		9	V (min)
				11	V (max)
T_{JW}	Warning Flag Temperature	Pin 9 ≤ 0.8V, I_L = 2 mA	145		°C
$V_{F(ON)}$	Flag Output Saturation Voltage	T_J = T_{JW}, I_L = 2 mA	0.15		V
$I_{F(OFF)}$	Flag Output Leakage	V_F = 12V	0.2	10	µA (max)
T_{JSD}	Shutdown Temperature	Outputs Turn OFF	170		°C
I_S	Quiescent Supply Current	All Logic Inputs Low	13	25	mA (max)
$t_{D(ON)}$	Output Turn-On Delay Time	Sourcing Outputs, I_{OUT} = 3A	300		ns
		Sinking Outputs, I_{OUT} = 3A	300		ns
t_{ON}	Output Turn-On Switching Time	Bootstrap Capacitor = 10 nF			
		Sourcing Outputs, I_{OUT} = 3A	100		ns
		Sinking Outputs, I_{OUT} = 3A	80		ns
$t_{D(OFF)}$	Output Turn-Off Delay Times	Sourcing Outputs, I_{OUT} = 3A	200		ns
		Sinking Outputs, I_{OUT} = 3A	200		ns
t_{OFF}	Output Turn-Off Switching Times	Bootstrap Capacitor = 10 nF			
		Sourcing Outputs, I_{OUT} = 3A	75		ns
		Sinking Outputs, I_{OUT} = 3A	70		ns
t_{PW}	Minimum Input Pulse Width	Pins 3, 4 and 5	1		µs
t_{CPR}	Charge Pump Rise Time	No Bootstrap Capacitor	20		µs

Note 1: Absolute Maximum Ratings indicate limits beyond which damage to the device may occur. DC and AC electrical specifications do not apply when operating the device beyond its rated operating conditions.

Note 2: See Application Information for details regarding current limiting.

Note 3: The maximum power dissipation must be derated at elevated temperatures and is a function of $T_{J(max)}$, θ_{JA}, and T_A. The maximum allowable power dissipation at any temperature is $P_{D(max)} = (T_{J(max)} - T_A)/\theta_{JA}$, or the number given in the Absolute Ratings, whichever is lower. The typical thermal resistance from junction to case (θ_{JC}) is 1.0°C/W and from junction to ambient (θ_{JA}) is 30°C/W. For guaranteed operation $T_{J(max)}$ = 125°C.

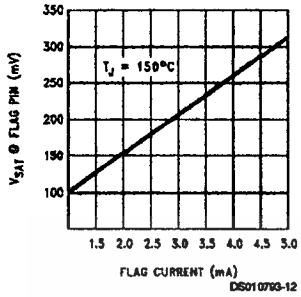
Note 4: Human-body model, 100 pF discharged through a 1.5 kΩ resistor. Except Bootstrap pins (pins 1 and 11) which are protected to 1000V of ESD.

Note 5: All limits are 100% production tested at 25°C. Temperature extreme limits are guaranteed via correlation using accepted SQC (Statistical Quality Control) methods. All limits are used to calculate AOQL, (Average Outgoing Quality Level).

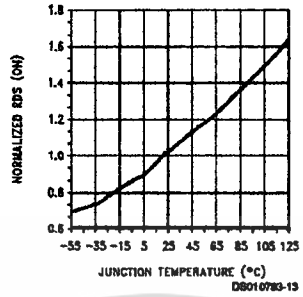
Note 6: Output currents are pulsed (t_W < 2 ms, Duty Cycle < 5%).

Typical Performance Characteristics

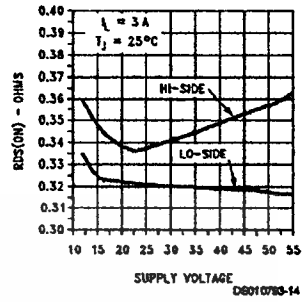
V_{SAT} vs Flag Current



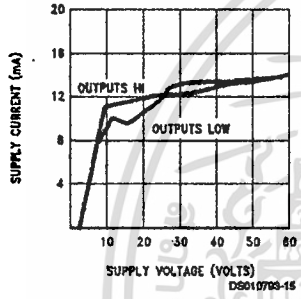
$R_{DS(ON)}$ vs Temperature



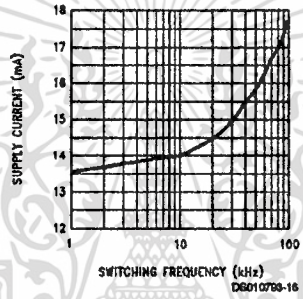
$R_{DS(ON)}$ vs Supply Voltage



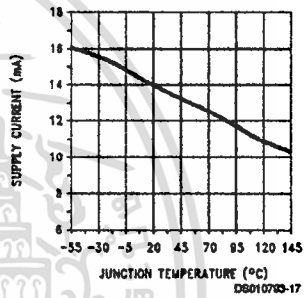
Supply Current vs Supply Voltage



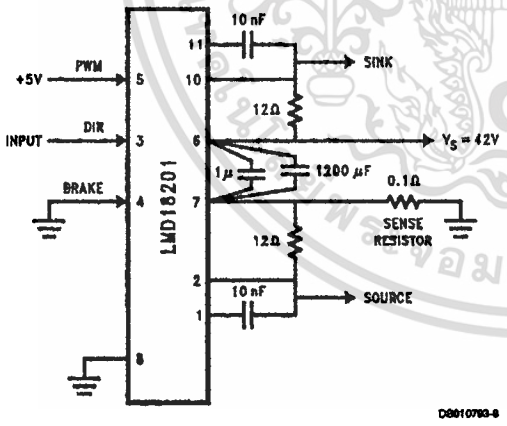
Supply Current vs Frequency (V_S = 42V)



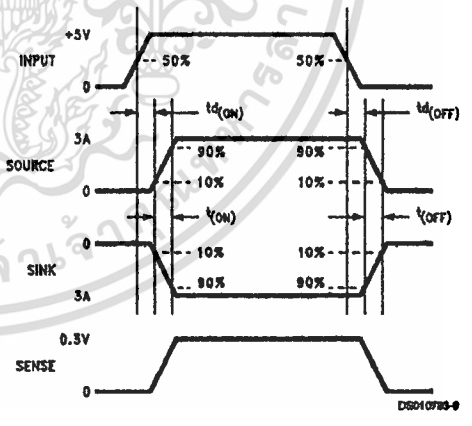
Supply Current vs Temperature (V_S = 42V)



Test Circuit



Switching Time Definitions



19-4323; Rev 7b; 11/97

MAXIM

+5V-Powered, Multichannel RS-232 Drivers/Receivers

General Description

The MAX220–MAX249 family of line drivers/receivers is intended for all EIA/TIA-232E and V.28/V.24 communications interfaces, particularly applications where ±12V is not available.

These parts are especially useful in battery-powered systems, since their low-power shutdown mode reduces power dissipation to less than 5µW. The MAX225, MAX233, MAX235, and MAX245/MAX246/MAX247 use no external components and are recommended for applications where printed circuit board space is critical.

Applications

Portable Computers

Low-Power Modems

Interface Translation

Battery-Powered RS-232 Systems

Multi-Drop RS-232 Networks

Features

Superior to Bipolar

- ♦ **Operate from Single +5V Power Supply (+5V and +12V—MAX231/MAX239)**
- ♦ **Low-Power Receive Mode in Shutdown (MAX223/MAX242)**
- ♦ **Meet All EIA/TIA-232E and V.28 Specifications**
- ♦ **Multiple Drivers and Receivers**
- ♦ **3-State Driver and Receiver Outputs**
- ♦ **Open-Line Detection (MAX243)**

Ordering Information

PART	TEMP. RANGE	PIN-PACKAGE
MAX220CPE	0°C to +70°C	16 Plastic DIP
MAX220CSE	0°C to +70°C	16 Narrow SO
MAX220CWE	0°C to +70°C	16 Wide SO
MAX220C/D	0°C to +70°C	Dice*
MAX220EPE	-40°C to +85°C	16 Plastic DIP
MAX220ESE	-40°C to +85°C	16 Narrow SO
MAX220EWE	-40°C to +85°C	16 Wide SO
MAX220EJE	-40°C to +85°C	16 CERDIP
MAX220MJE	-55°C to +125°C	16 CERDIP

Ordering Information continued at end of data sheet.

**Contact factory for dice specifications.*

Selection Table

Part Number	Power Supply (V)	No. of RS-232 Drivers/Rx	No. of Ext. Caps	Nominal Cap. Value (µF)	SHDN & Three-State	Rx Active in SHDN	Data Rate (kbps)	Features
MAX220	+5	2/2	4	4.7/10	No	—	120	Ultra-low-power, industry-standard pinout
MAX222	+5	2/2	4	0.1	Yes	—	200	Low-power shutdown
MAX223 (MAX213)	+5	4/5	4	1.0 (0.1)	Yes	✔	120	MAX241 and receivers active in shutdown
MAX225	+5	5/5	0	—	Yes	✔	120	Available in SO
MAX230 (MAX200)	+5	5/0	4	1.0 (0.1)	Yes	—	120	5 drivers with shutdown
MAX231 (MAX201)	+5 and +7.5 to +13.2	2/2	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; same functions as MAX232
MAX232 (MAX202)	+5	2/2	4	1.0 (0.1)	No	—	120 (64)	Industry standard
MAX232A	+5	2/2	4	0.1	No	—	200	Higher slew rate, small caps
MAX233 (MAX203)	+5	2/2	0	—	No	—	120	No external caps
MAX233A	+5	2/2	0	—	No	—	200	No external caps, high slew rate
MAX234 (MAX204)	+5	4/0	4	1.0 (0.1)	No	—	120	Replaces 1488
MAX235 (MAX205)	+5	5/5	0	—	Yes	—	120	No external caps
MAX236 (MAX206)	+5	4/3	4	1.0 (0.1)	Yes	—	120	Shutdown, three state
MAX237 (MAX207)	+5	5/3	4	1.0 (0.1)	No	—	120	Complements IBM PC serial port
MAX238 (MAX208)	+5	4/4	4	1.0 (0.1)	No	—	120	Replaces 1488 and 1489
MAX239 (MAX209)	+5 and +7.5 to +13.2	3/5	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; single-package solution for IBM PC serial port
MAX240	+5	5/5	4	1.0	Yes	—	120	DIP or flatpack package
MAX241 (MAX211)	+5	4/5	4	1.0 (0.1)	Yes	—	120	Complete IBM PC serial port
MAX242	+5	2/2	4	0.1	Yes	✔	200	Separate shutdown and enable
MAX243	+5	2/2	4	0.1	No	—	200	Open-line detection simplifies cabling
MAX244	+5	8/10	4	1.0	No	—	120	High slew rate
MAX245	+5	8/10	0	—	Yes	✔	120	High slew rate, int. caps, two shutdown modes
MAX246	+5	8/10	0	—	Yes	✔	120	High slew rate, int. caps, three shutdown modes
MAX247	+5	8/9	0	—	Yes	✔	120	High slew rate, int. caps, nine operating modes
MAX248	+5	8/8	4	1.0	Yes	✔	120	High slew rate, selective half-chip enables
MAX249	+5	6/10	4	1.0	Yes	✔	120	Available in quad flatpack package

ABSOLUTE MAXIMUM RATINGS—MAX220/222/232A/233A/242/243

Supply Voltage (V_{CC}) ..... -0.3V to +6V
Input Voltages
T_{IN} ..... -0.3V to (V_{CC} - 0.3V)
R_{IN} ..... ±30V
T_{OUT} (Note 1) ..... ±15V
Output Voltages
T_{OUT} ..... ±15V
R_{OUT} ..... -0.3V to (V_{CC} + 0.3V)
Driver/Receiver Output Short Circuited to GND ..... Continuous
Continuous Power Dissipation (T_A = +70°C)
16-Pin Plastic DIP (derate 10.53mW/°C above +70°C) 842mW
18-Pin Plastic DIP (derate 11.11mW/°C above +70°C) 889mW
20-Pin Plastic DIP (derate 8.00mW/°C above +70°C) 440mW
16-Pin Narrow SO (derate 8.70mW/°C above +70°C) 696mW
16-Pin Wide SO (derate 9.52mW/°C above +70°C) 762mW
18-Pin Wide SO (derate 9.52mW/°C above +70°C) 762mW
20-Pin Wide SO (derate 10.00mW/°C above +70°C) 800mW
20-Pin SSOP (derate 8.00mW/°C above +70°C) 640mW
16-Pin CERDIP (derate 10.00mW/°C above +70°C) 800mW
18-Pin CERDIP (derate 10.53mW/°C above +70°C) 842mW
Operating Temperature Ranges
MAX2_ _AC_ _, MAX2_ _C_ _ ..... 0°C to +70°C
MAX2_ _AE_ _, MAX2_ _E_ _ ..... -40°C to +85°C
MAX2_ _AM_ _, MAX2_ _M_ _ ..... -55°C to +125°C
Storage Temperature Range ..... -65°C to +160°C
Lead Temperature (soldering, 10sec) ..... +300°C

Note 1: Input voltage measured with T_{OUT} in high-impedance state, $\overline{SHDN}$ or V_{CC} = 0V.

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

ELECTRICAL CHARACTERISTICS—MAX220/222/232A/233A/242/243

(V_{CC} = +5V ±10%, C1–C4 = 0.1µF, T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
RS-232 TRANSMITTERS						
Output Voltage Swing	All transmitter outputs loaded with 3kΩ to GND		±5	±8		V
Input Logic Threshold Low				1.4	0.8	V
Input Logic Threshold High			2	1.4		V
Logic Pull-Up/Input Current	Normal operation			5	40	µA
	$\overline{SHDN}$ = 0V, MAX222/242, shutdown			±0.01	±1	
Output Leakage Current	V_{CC} = 5.5V, $\overline{SHDN}$ = 0V, V_{OUT} = ±15V, MAX222/242			±0.01	±10	µA
	V_{CC} = $\overline{SHDN}$ = 0V, V_{OUT} = ±15V			±0.01	±10	
Data Rate	All except MAX220, normal operation			200	116	kbits/ sec
	MAX220			22	20	
Transmitter Output Resistance	V_{CC} = V+ = V- = 0V, V_{OUT} = ±2V		300	10M		Ω
Output Short-Circuit Current	V_{OUT} = 0V		±7	±22		mA
RS-232 RECEIVERS						
RS-232 Input Voltage Operating Range					±30	V
RS-232 Input Threshold Low	V_{CC} = 5V	All except MAX243 $R2_{IN}$	0.8	1.3		V
		MAX243 $R2_{IN}$ (Note 2)	-3			
RS-232 Input Threshold High	V_{CC} = 5V	All except MAX243 $R2_{IN}$		1.8	2.4	V
		MAX243 $R2_{IN}$ (Note 2)		-0.5	-0.1	
RS-232 Input Hysteresis	All except MAX243, V_{CC} = 5V, no hysteresis in shdn.		0.2	0.5	1	V
	MAX243			1		
RS-232 Input Resistance			3	5	7	kΩ
TTL/CMOS Output Voltage Low	I_{OUT} = 3.2mA			0.2	0.4	V
TTL/CMOS Output Voltage High	I_{OUT} = -1.0mA		3.5	V_{CC} - 0.2		V
TTL/CMOS Output Short-Circuit Current	Sourcing V_{OUT} = GND		-2	-10		mA
	Shrinking V_{OUT} = V_{CC}		10	30		
TTL/CMOS Output Leakage Current	$\overline{SHDN}$ = V_{CC} or $\overline{EN}$ = V_{CC} ($\overline{SHDN}$ = 0V for MAX222), 0V ≤ V_{OUT} ≤ V_{CC}			±0.05	±10	µA

ELECTRICAL CHARACTERISTICS—MAX220/222/232A/233A/242/243 (continued)

(V_{CC} = +5V ±10%, C1–C4 = 0.1µF, T_A = T_{MIN} to T_{MAX}, unless otherwise noted.)

PARAMETER	CONDITIONS		MIN	TYP	MAX	UNITS
$\overline{EN}$ Input Threshold Low	MAX242			1.4	0.8	V
$\overline{EN}$ Input Threshold High	MAX242		2.0	1.4		V
Operating Supply Voltage			4.5		5.5	V
V_{CC} Supply Current ($\overline{SHDN}$ = V_{CC}), Figures 5, 6, 11, 19	No load	MAX220		0.5	2	mA
		MAX222/232A/233A/242/243		4	10	
	3kΩ load both inputs	MAX220		12		
		MAX222/232A/233A/242/243		15		
Shutdown Supply Current	MAX222/242	T_A = +25°C		0.1	10	µA
		T_A = 0°C to +70°C		2	50	
		T_A = -40°C to +85°C		2	50	
		T_A = -55°C to +125°C		35	100	
$\overline{SHDN}$ Input Leakage Current	MAX222/242				±1	µA
$\overline{SHDN}$ Threshold Low	MAX222/242			1.4	0.8	V
$\overline{SHDN}$ Threshold High	MAX222/242		2.0	1.4		V
Transition Slew Rate	C_L = 50pF to 2500pF, R_L = 3kΩ to 7kΩ, V_{CC} = 5V, T_A = +25°C, measured from +3V to -3V or -3V to +3V	MAX222/232A/233A/242/243	6	12	30	V/µs
		MAX220	1.5	3	30	
Transmitter Propagation Delay TTL to RS-232 (normal operation), Figure 1	t_{PHLT}	MAX222/232A/233A/242/243		1.3	3.5	µs
		MAX220		4	10	
	t_{PLHT}	MAX222/232A/233A/242/243		1.5	3.5	
		MAX220		5	10	
Receiver Propagation Delay RS-232 to TTL (normal operation), Figure 2	t_{PHLR}	MAX222/232A/233A/242/243		0.5	1	µs
		MAX220		0.6	3	
	t_{PLHR}	MAX222/232A/233A/242/243		0.6	1	
		MAX220		0.8	3	
Receiver Propagation Delay RS-232 to TTL (shutdown), Figure 2	t_{PHLS}	MAX242		0.5	10	µs
	t_{PLHS}	MAX242		2.5	10	
Receiver-Output Enable Time, Figure 3	t_{ER}	MAX242		125	500	ns
Receiver-Output Disable Time, Figure 3	t_{DR}	MAX242		160	500	ns
Transmitter-Output Enable Time ($\overline{SHDN}$ goes high), Figure 4	t_{ET}	MAX222/242, 0.1µF caps (includes charge-pump start-up)		250		µs
Transmitter-Output Disable Time ($\overline{SHDN}$ goes low), Figure 4	t_{DT}	MAX222/242, 0.1µF caps		600		ns
Transmitter + to - Propagation Delay Difference (normal operation)	t_{PHLT} - t_{PLHT}	MAX222/232A/233A/242/243		300		ns
		MAX220		2000		
Receiver + to - Propagation Delay Difference (normal operation)	t_{PHLR} - t_{PLHR}	MAX222/232A/233A/242/243		100		ns
		MAX220		225		

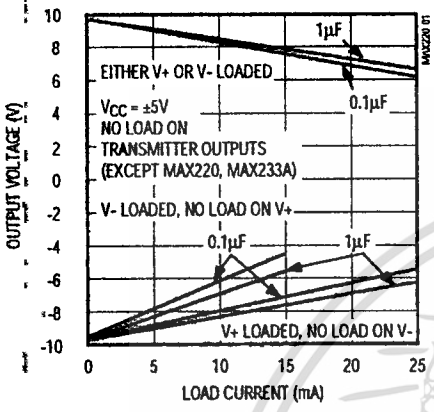
Note 2: MAX243 $R2_{OUT}$ is guaranteed to be low when $R2_{IN}$ ≥ 0V or is floating.

+5V-Powered, Multichannel RS-232 Drivers/Receivers

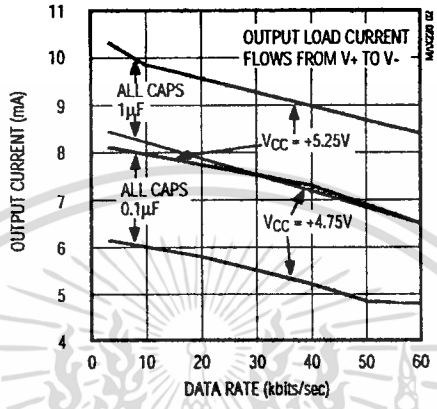
Typical Operating Characteristics

MAX220/MAX222/MAX232A/MAX233A/MAX242/MAX243

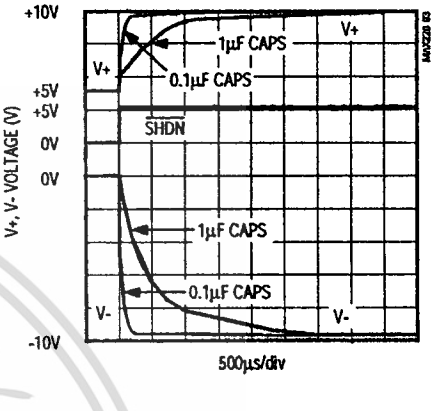
OUTPUT VOLTAGE vs. LOAD CURRENT

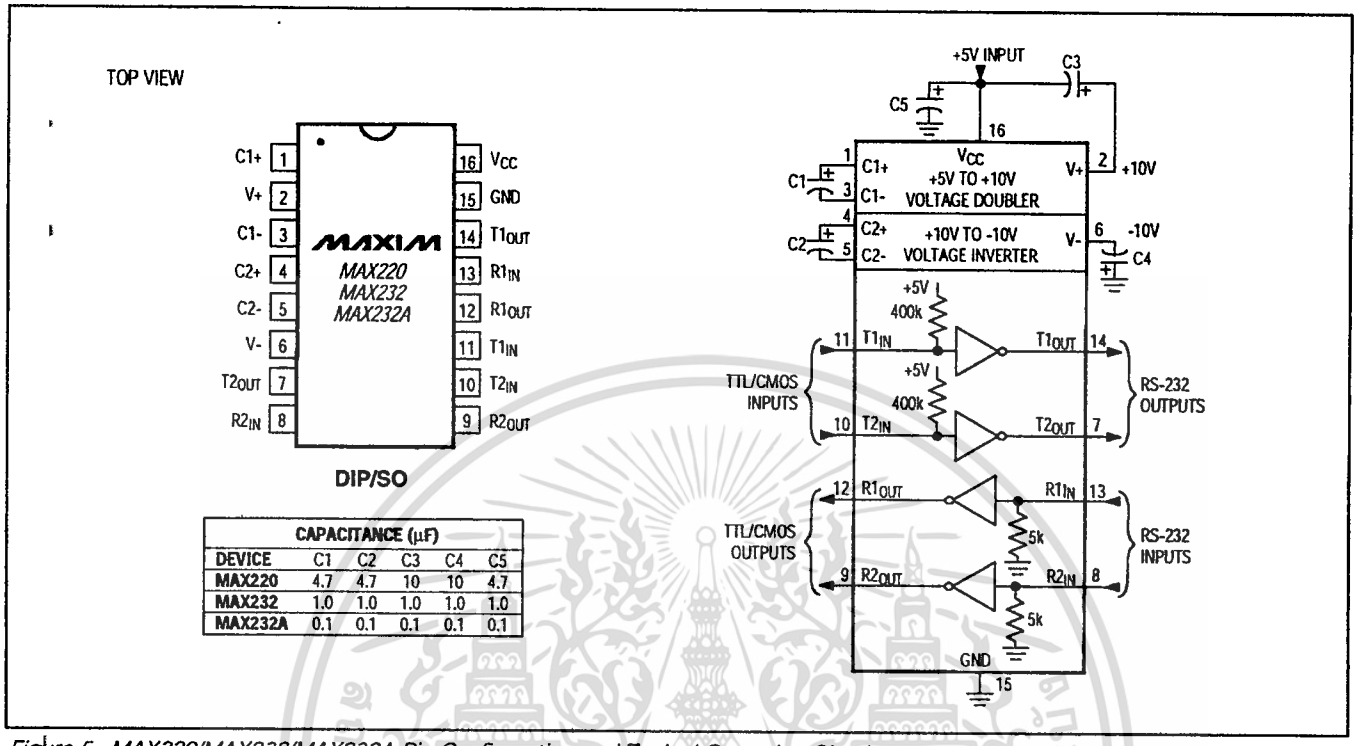


AVAILABLE OUTPUT CURRENT vs. DATA RATE



MAX222/MAX242 ON-TIME EXITING SHUTDOWN





CAPACITANCE (µF)					
DEVICE	C1	C2	C3	C4	C5
MAX220	4.7	4.7	10	10	4.7
MAX232	1.0	1.0	1.0	1.0	1.0
MAX232A	0.1	0.1	0.1	0.1	0.1

Figure 5. MAX220/MAX232/MAX232A Pin Configuration and Typical Operating Circuit

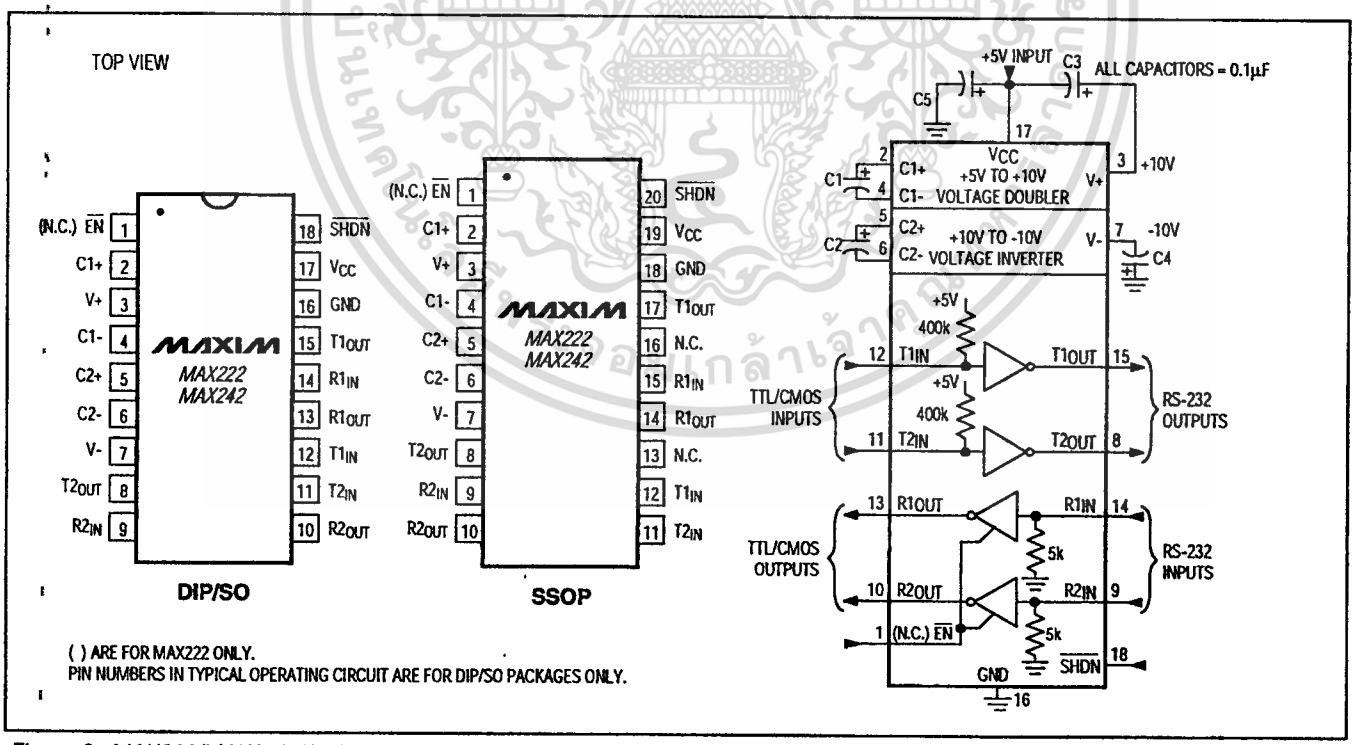


Figure 6. MAX222/MAX242 Pin Configurations and Typical Operating Circuit

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้คงไม่สามารถที่จะสำเร็จลุล่วงลงได้ถ้าไม่ได้รับความช่วยเหลือจาก อาจารย์ประภาษ อุดมกิมาพันธุ์ ที่คอยให้คำปรึกษาและช่วยแก้ไขปัญหาต่าง ๆ ด้วยดีตลอดมา ตลอดจนถึงเครื่องมือต่าง ๆ ที่ได้หยิบยืมอาจารย์มาใช้ในการทำโปรเจค ขอบคุณอาจารย์วิศรุต ศรีรัตนะ ที่ให้ภาพสวย ๆ เกี่ยวกับสเตปปิงมอเตอร์ ขอบคุณเพื่อน ๆ ห้อง 2 ที่ช่วยให้กำลังใจ ในการทำงานตลอดจนให้คำปรึกษาในเรื่องเกี่ยวกับ ภาษา Pascal ขอบคุณเพื่อน ๆ ภาคเทคโนโลยีการวัดคุมทางอุตสาหกรรมที่คอยส่งข้าวส่งน้ำ ขอบคุณบริษัท Sila และ บริษัท อิเลคทรอนิคส์ซอร์ส ที่นำเข้าและพัฒนาด้านไมโครคอนโทรเลอร์ ขอบคุณชุมนุม Robot ที่ทำให้ผู้จัดทำมีความรู้ได้ถึงเพียงนี้ และสุดท้ายต้องขอบคุณ ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรมสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังที่ได้เอื้อเฟื้อในเรื่องเครื่องมือและอุปกรณ์ต่าง ๆ ในการทำโครงงานที่ให้ประสบผลสำเร็จลุล่วงไปด้วยดี

หนังสืออ้างอิง

1. Atmel Coporation, "Microcontroller Product with Flash Technology" , www.atmel.com, 1999

2. Intel Coporation, "Introduction to MCS-51" ,www.intel.com,1998

3. โยธิน เปรมปราณี , "ระบบเซอร์โวมอเตอร์และอิเลคทรอนิคส์คอนโทรลมอเตอร์" , สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง,2533

4. สุนทร วิทูสุรพจน์," การโปรแกรมภาษาแอสเซมบลีของไมโครคอนโทรลเลอร์ตระกูล 8051", บริษัทซีเอ็ดยูเคชั่น,2537

5. ทวีชัย ภูรีทิพย์, "ไขปัญหา RS-232" ,บริษัทซีเอ็ดยูเคชั่น ,2538

6. พิพัฒน์ เลาหสงคราม, "ไมโครคอนโทรลเลอร์ MCS-48 MCS-51",ภาควิชาเทคโนโลยีการวัดคุมทางอุตสาหกรรม คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง,2537