ล็อคประตูแบบรหัส

THE DOOR LOCK BY ENCODER

โดย

นาย สมชาย เกียรติก้องคีรี

นาย อนุพงษ์ หาชานนท์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมสารสนเทศ

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

หัวข้อปริญญานิพนธ์	ล็อคประตูแบบรหัส
นักศึกษา	นายสมชาย เกียรติก้องคีรี รหัสประจำตัว 43015750
	นายอนุพงษ์ หาชานนท์ รหัสประจำตัว 43015757
อาจารย์ผู้ควบคุมปริญญานิพนธ์	อ.มนชนก ศรีเสือขาม
ระดับการศึกษา	ปริญญาอุตสาหกรรมศาสตรบัณฑิต
	สาขาวิชาเทคโนโลยีอิเล็กทรอนิกส์
ภาควิชา	วิศวกรรมสารสนเทศ
ปีการศึกษา	2544

บทคัดย่อ

ปริญญานิพนธ์นี้เป็นการศึกษา ล็อคประตูแบบรหัสมีจุดมุ่งหมายเพื่อศึกษาการทำงานของไมโครคอนโทรลเลอร์ ในการนำไปประยุกต์เพื่อควบคุมอุปกรณ์เปิด-ปิดประตู และแสดงผลทางไดโอดเปล่งแสง 7 ส่วน เมื่อเรากดรหัสที่ถูกต้อง โครงงานนี้ประกอบด้วยส่วนสำคัญ 2 ส่วนใหญ่ๆคือ ส่วนที่เป็น ฮาร์ดแวร์ และส่วนที่เป็น ซอร์ฟแวร์ ส่วนที่เป็น ฮาร์ดแวร์ ประกอบด้วยส่วนของการควบคุมการเปิด-ปิดประตูโดยมีกลไกในการเปิด-ปิดประตู ส่วนของการรับข้อมูลจากคีย์และแสดงผลทางไดโอดเปล่งแสง 7 ส่วน รวมไปถึงส่วนที่เป็น แหล่งจ่ายไฟสำหรับจ่ายไฟให้กับวงจรต่างๆตามความต้องการของวงจรซึ่งโครงงานนี้ใช้ 5 และ 12 โวลท์ส่วนทางด้าน ซอร์ฟแวร์ นั้นจะประกอบด้วยโปรแกรมที่ใช้ควบคุมการทำงานของไมโครคอนโทรลเลอร์ ผลของโครงงานนี้จะให้ประโยชน์โดยผู้ใช้งานเพียงแค่กดรหัสที่ถูกต้องตามรหัสที่โปรแกรมไว้ก็จะสามารถเปิดประตูได้โดยไม่ต้องใช้ลูกกุญแจจึงทำให้ไม่ต้องพกลูกกุญแจไปไหนมาไหนและมีเสียงรบกวนตลอดเวลาอีกทั้งยังไม่ต้องคอยระวังการสูญหายของลูกกุญแจรวมไปถึงการโจรกรรมของขโมยด้วยการใช้กุญแจผีอีกด้วย

PROJECT TITLE	**THE DOOR LOCK BY ENCODER**		
STUDENT	**Mr. Chomchai**	**Kaitkongkeeree**	**No. 43015750**
	Mr. Anupong	**Hachanont**	**No. 43015757**
ADVISOR	**Ms. Monchanok**	**Srisuakam**	
COURSE	**Bachelor of Industrial Technology in Electronics**		
DEPARTMENT	**Information Engineer**		
YEAR	**2001**		

ABSTRACT

This thesis presents encoder the Door Lock is used to control the opening and closing the door lock and the presentation on the LED 7 Segment screen when pressing the right code. The equipment is composed of two important parts, the hardware and the software. The hardware is composed of the opening and closing instrument and the receives. The information will be seen on the LED 7 Segment screen and it will send the 5,12 volts current to other parts of the microcontroller. The software is composed of the Controlling program which will control all the work of the microcontroller. Encoder the Door Lock will help the users open the door automatically without using the keys (Only press the right code). There will be no problem about losing the keys and the noises of many keys striking with one another will disappear. Besides, robbers can't open the door with false keys.

หัวข้อปริญญานิพนธ์	ล็อคประตูแบบรหัส
TITLE	**THE DOOR LOCK BY ENCODER**
โดย	นายสมชาย เกียรติก้องคีรี รหัสประจำตัว 43015750
	นายอนุพงษ์ หาชานนท์ รหัสประจำตัว 43015757
อาจารย์ผู้ควบคุมปริญญานิพนธ์	อ.มนชนก ศรีเสือขาม
ภาควิชา	วิศวกรรมสารสนเทศ
ปีการศึกษา	2544

ปริญญานิพนธ์ฉบับนี้ได้รับการอนุมัติให้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรอุตสาหกรรมศาสตรบัณฑิต คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

.....

(อ.มนชนก ศรีเสือขาม)

อาจารย์ผู้ควบคุมปริญญานิพนธ์

กิตติกรรมประกาศ

ในการจัดทำโครงงานนี้ ทางผู้จัดทำได้รับคำปรึกษาและได้รับการแนะแนวทางในการดำเนินงานจาก อ.มนชนก ศรีเสือขาม จนกระทั่งจัดทำโครงงานสำเร็จได้ด้วยดี ทีมงานผู้จัดทำโครงงานขอกราบขอบพระคุณท่านอาจารย์ที่ให้ความกรุณามา ณ ที่นี้

ท้ายนี้ทีมงานผู้จัดทำขอกราบขอบพระคุณ บิดา-มารดา และพี่ๆ ซึ่งให้การสนับสนุนทางด้านการเงินและเป็นกำลังใจแก่ผู้จัดทำเสมอมาจนสำเร็จการศึกษา

สมชาย เกียรติก้องคีรี

อนุพงษ์ หาชานนท์

สารบัญ

สารบัญ (ต่อ)

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

เมื่อท่านประสบปัญหาเสียงรบกวนของลูกกุญแจเมื่อพกพาไปไหนมาไหน อีกทั้งยังต้องคอยระวังการสูญหายของลูกกุญแจ รวมไปถึงการโจรกรรมของขโมยที่ใช้กุญแจผี คุณไม่ต้องพะวงกับปัญหานี้อีกต่อไป เมื่อโครงงานล็อคประตูแบบรหัส (The Door Lock by Encoder) เกิดขึ้นมาเพื่อแก้ปัญหาที่ท่านประสบอยู่

เนื่องจากในบ้านเรือนทั่วไปจะมีการใช้ประตูแบบเป็นระบบลูกบิดหรือแบบอื่นๆซึ่งก็ต้องใช้ลูกกุญแจเมื่อท่านเดินทางไปไหนก็ต้องนำลูกกุญแจไปด้วยเสมอซึ่งบางครั้งลูกกุญแจก็ก่อความรำคาญให้แก่ท่านไม่ว่าจะเป็นเสียงดัง หรือบางครั้งบ้านใครมีการล็อคประตูหลายๆชั้นก็ต้องมีลูกกุญแจมากขึ้นตามมาด้วยทำให้มีน้ำหนักมาก และต้องคอยสำรวจดูลูกกุญแจอยู่ตลอดเวลา เพราะกลัวการสูญหาย ถึงแม้ท่านจะล็อคประตูอย่างแน่นหนาก็ไม่ได้หมายความว่าทรัพย์สินที่อยู่ในบ้านจะไม่ได้ถูกโจรกรรมจากขโมยด้วยกุญแจผี ซึ่งปัญหาเหล่านี้จะถูกแก้ไข หรือมีทางออกเพราะโครงงานล็อคประตูแบบรหัสจะเป็นการเปิด-ปิดประตูด้วยการกดรหัสแทนการใช้ลูกกุญแจ

1.1 วัตถุประสงค์

เพื่อศึกษาการทำงานของ ไมโครคอนโทรลเลอร์ในการนำไปประยุกต์ใช้งานเพื่อ ควบคุมอุปกรณ์เปิด-ปิดประตูและแสดงผลทางไดโอดเปล่งแสง 7 ส่วน เมื่อเรากดรหัสที่ถูกต้อง

1.2 ขอบเขต

สามารถนำไมโครคอนโทรลเลอร์มาประยุกต์ใช้งานโดยการเขียนโปรแกรมในการรับคีย์สวิตช์รหัสโดยถ้ารหัสถูกต้องก็จะสั่งให้ไมโครคอนโทรลเลอร์ควบคุมกลไกเปิดประตู ซึ่งสามารถที่จะเปลี่ยนรหัสเปิดประตูได้ด้วย และก็จะแสดงผลโดยไดโอดเปล่งแสง 7 ส่วน

1.3 ขั้นตอนการดำเนินโครงงาน

- เตรียมข้อมูล
- ออกแบบวงจร
- ทดสอบวงจร
- ออกแบบ ทำ PCB

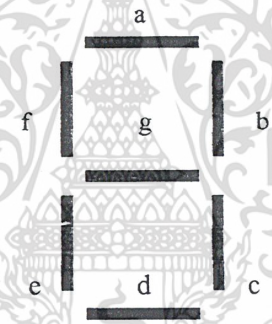
- ออกแบบส่วนซอร์ฟแวร์
- ประกอบและทดสอบฮาร์ดแวร์
- ทดสอบการทำงานของซอร์ฟแวร์
- ทดสอบการทำงานร่วมกันของซอร์ฟแวร์กับฮาร์ดแวร์
- ทดสอบการใช้งานจริง
- เขียนปริญญานิพนธ์

บทที่ 2
ทฤษฎีที่ใช้ในโครงงาน

2.1 การเชื่อมต่อกับหน่วยแสดงผล

2.1.1 การเชื่อมต่อกับไดโอดเปล่งแสง (LED) 7 ส่วน

การเชื่อมต่อกับไดโอดเปล่งแสงที่แสดงผลแบบตัวเลขได้ที่เรียกว่า ไดโอดเปล่งแสง 7 ส่วน (7 Segment Display) หรือ 7 เซกเมนต์ ซึ่งมีทั้งแบบคาโทดร่วม (Common-cathode) และแบบอาโนดร่วม (Common-anode) ไดโอดเปล่งแสง 7 ส่วนนี้จะเป็นการรวมไดโอดเปล่งแสง 7 หลอด ประกอบกันให้สามารถแสดงเป็นตัวเลขได้ถ้าเป็นชนิดที่นำขาคาโทดของหลอดไดโอดเปล่งแสง ทุกตัวมารวมกันเรียกว่าแบบคาโทดร่วม แบบอาโนดร่วมก็ทำนองเดียวกัน แต่ละเซกเมนต์จะมีชื่อขาดังรูปที่ 2.1



รูปที่ 2.1 แสดงสัญลักษณ์ของแต่ละเซกเมนต์

ถ้าเราเชื่อมต่อแต่ละขากับบัสข้อมูลของไมโครคอนโทรลเลอร์ โดยแต่ละบิตจะต่อกับขาของหลอดไดโอดเปล่งแสง 7 ส่วน ดังตารางที่ 2.1 ถ้าหากต้องการให้ไดโอดเปล่งแสง 7 ส่วนแสดงตัวเลขและตัวอักษรต่างๆเราจะต้องส่งข้อมูลให้แต่ละเซกเมนต์สว่างหรือดับให้ประกอบเป็นอักษรต่างๆ ดังนั้นข้อมูลที่ส่งไปที่พอร์ทจะเป็นตัวกำหนดตัวอักษรที่จะแสดงบนไดโอดเปล่งแสง 7 ส่วน ตัวอักษรและค่าต่างๆที่ส่งออกมามีความสัมพันธ์กันดังแสดงในตารางที่ 2.2

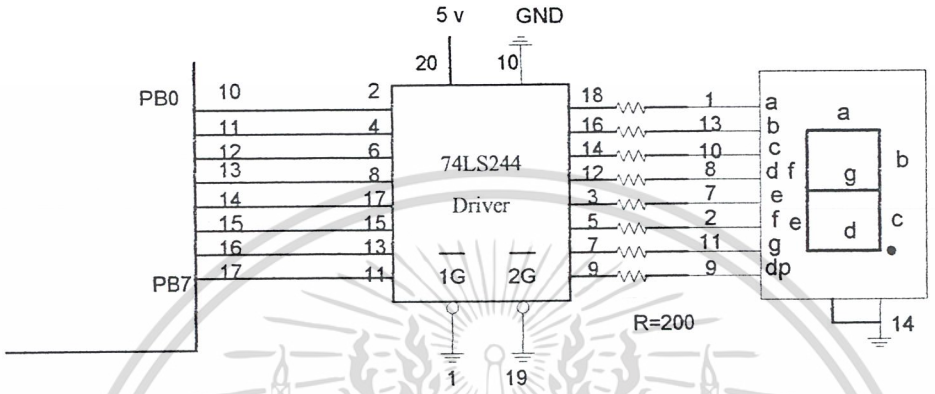
ตำแหน่งบิต	ตำแหน่งเซกเมนต์
7	-
6	g
5	f
4	e
3	d
2	c
1	b
0	a

ตารางที่ 2.1 แสดงตำแหน่งขาที่จะต่อกับพอร์ท

แสดงผล	อาโนดร่วม	คาโทดร่วม	แสดงผล	อาโนดร่วม	คาโทดร่วม
0	C0	3F	J	E1	1E
1	F9	06	L	C7	38
2	A4	5B	O	C0	3F
3	B0	4F	P	8C	73
4	99	66	U	C1	3E
5	92	6D	Y	99	66
6	82	7D	b	83	7C
7	F8	07	c	A7	58
8	80	7F	d	A1	5E
9	98	67	h	8B	74
A	88	77	n	AB	54
C	C6	39	o	A3	5C
E	86	79	r	AF	50
F	8E	71	u	E3	1C
G	82	70	-	BF	40
H	89	76	?	AC	53
I	F9	06	BLANK	FF	00

ตารางที่ 2.2 แสดงข้อมูลที่ส่งให้ไดโอดเปล่งแสง 7 ส่วนแสดงเป็นเลขต่างๆ

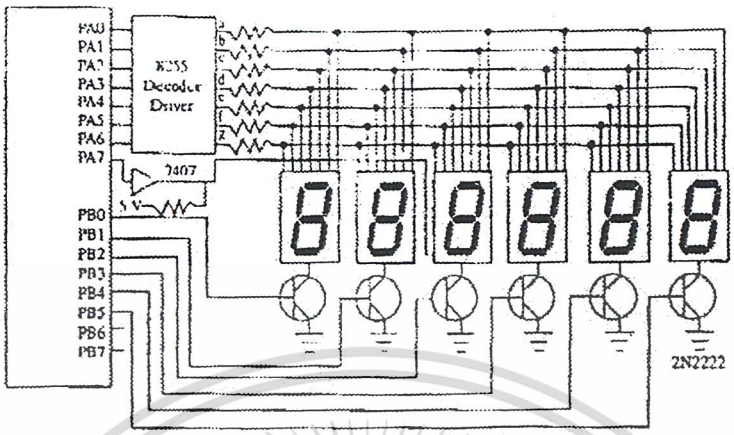
พิจารณาวงจรรูปที่ 2.2 ถ้าหากต่อไดโอดเปล่งแสง 7 ส่วนกับพอร์ท A ของ 8255 ซึ่งถอดรหัสไว้ที่หมายเลข 0FC00H-0FC03H โดยจะส่งข้อมูลให้ไดโอดเปล่งแสง 7 ส่วนทางพอ์ท A ส่วนขาคาโทดร่วมจะต่อกับบิต PC0 ซึ่งมีเกตบัฟเฟอร์ที่เอาต์พุตเป็นแบบ OC ต่ออยู่ด้วย ดังแสดงในรูปที่ 2.2



รูปที่ 2.2 แสดงการต่อไดโอดเปล่งแสง 7 ส่วนเข้ากับ 8255

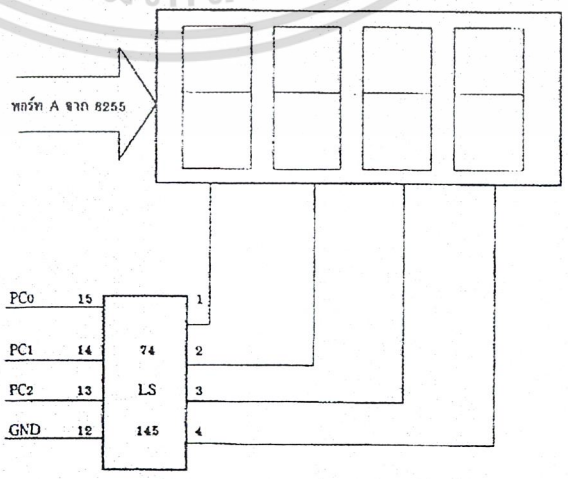
2.1.2 การแสดงผลกับไดโอดเปล่งแสง 7 ส่วนหลายๆตัว

ระบบไมโครคอนโทรลเลอร์หากต้องการแสดงผลเป็นตัวเลขหลายๆหลัก จำเป็นต้องใช้ไดโอดเปล่งแสง 7 ส่วนหลายๆตัว ไดโอดเปล่งแสง 7 ส่วนหนึ่งตัวจะต้องใช้พอร์ทขนาด 8 บิต หนึ่งพอร์ท หากต้องการแสดงตัวเลข 3 หลักจะต้องใช้พอร์ทถึง 3 พอร์ท ซึ่งจะเห็นว่าถ้าต้องการแสดงหลายหลักจะต้องใช้พอร์ทหลายพอร์ท มีวิธีหนึ่งที่จะแสดงผลหลายหลักได้ โดยจะประหยัดจำนวนพอร์ทและเป็นวิธีที่ใช้กัน โดยทั่วๆไปเรียกว่า มัลติเพลกดีสเพลย์ (Multiplexed Displays) โดยต่อขาแต่ละเซกเมนต์เข้าด้วยกัน จากนั้นจะใช้วิธีสแกนให้ไดโอดเปล่งแสง 7 ส่วนติดทีละหลัก โดยการสแกนแต่ละหลักจะต้องเร็วจนตาไม่สามารถมองการดับของ ไดโอดเปล่งแสง 7 ส่วนได้ทันจนดูเหมือนว่าไดโอดเปล่งแสง 7 ส่วนทุกตัวติดพร้อมกัน พิจารณาวงจรตามรูปที่ 2.3



รูปที่ 2.3 แสดงการต่อไดโอดเปล่งแสง 7 ส่วนหลายๆตัวกับ 8255

วงจรตามรูปที่ 2. 3 เรียกว่า ซิก-ดิจิท มัลติเพลกซิง ดีสเพลย์ (Six-digit Multiplexing Displays) ข้อมูลที่ส่งให้ ไดโอดเปล่งแสงแต่ละหลักจะใช้พอร์ท A ของ 8255 ถ้าไดโอดเปล่งแสง 7 ส่วน ทุกเซกเมนต์สว่างพร้อมกันในแต่ละหลัก (แสดงเลข 8) จะมีกระแสไหลที่ขาคาโทดร่วมประมาณ 60 mA ดังนั้นจะใช้ ทรานซิสเตอร์ช่วยในการขับกระแสลงกราวด์ โดยต่อขา B กับพอร์ท B ของ 8255 ตรงนี้จะเรียกว่า ดิจิทไดรเวอร์ (Digit Driver) เมื่อ 8255 ส่งข้อมูลให้พอร์ท A ไดโอดเปล่งแสง 7 ส่วนทุกหลักจะพร้อมที่จะ แสดงผล ขึ้นกับว่าในพอร์ท B นั้น บิตใดเป็น “1” หลักนั้นก็จะแสดงผล ถ้าต้องการให้แสดงผลเป็นตัวเลข 123456 จะทำได้โดยขั้นแรกส่งข้อมูลที่แสดงเลข 1 ออกไป จากนั้นทำให้ PB1 เป็น “1” ทำไปจนถึงหลักที่ 6 แล้วเขียนโปรแกรมวนลูปโดยหน่วงเวลาให้เหมาะสมก็จะเห็นตัวเลข 123456 พิจารณาวงจรตามรูปที่ 2.4

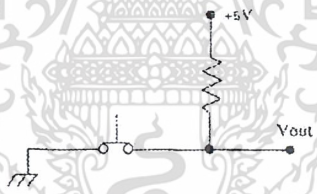


รูปที่ 2.4 แสดงการต่อไดโอดเปล่งแสง 7 ส่วน 4 ตัว

จากรูปที่ 2.4 เป็นการต่อไดโอดเปล่งแสง 7 ส่วน 4 ตัวเข้ากับพอร์ทเพื่อที่จะแสดงผลแบบมัลติเพลก (Multiplexed) ในการต่อวงจรจากพอร์ท A ของ 8255 เข้ากับทุกเซกเมนต์ของไดโอดเปล่งแสง 7 ส่วน จากนั้นจะใช้พอร์ท C เป็นตัวสแกนทางหลักโดยใช้ไอซีเบอร์ 74LS145 เป็นตัวถอดรหัส ซึ่งไอซีเบอร์นี้สามารถต่อกับไดโอดเปล่งแสง 7 ส่วนได้ถึง 10 ตัว (รายละเอียดศึกษาได้จากคู่มือไอซี TTL โดยตรง) และเอาต์พุตของไอซีเบอร์นี้จะเป็นแบบ โอเพนคอลเลคเตอร์ (Open Collector) ซึ่งสามารถรับกระแสเข้าได้มากเมื่อไดโอดเปล่งแสง 7 ส่วนทุกเซกเมนต์สว่างพร้อมกัน

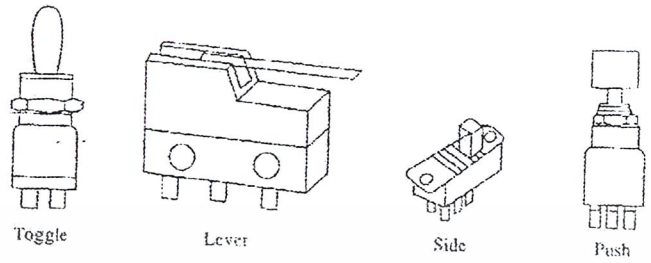
2.2 การเชื่อมต่อสวิตช์เข้ากับระบบไมโครคอนโทรลเลอร์

สวิตช์หรือคีบอร์ด (Keyboard) เป็นอุปกรณ์อินพุตพื้นฐานที่ระบบไมโครคอนโทรลเลอร์สามารถรับข้อมูลได้ การสร้างสวิตช์ให้กับระบบไมโครคอนโทรลเลอร์นั้นอาจทำได้ดังรูปที่ 2.5 โดยการต่อเข้ากับแต่ละบิตของพอร์ทอินพุต ถ้าสวิตช์ ON จะให้ลอจิก “0” แต่ถ้าสวิตช์ OFF จะให้ลอจิก “1”

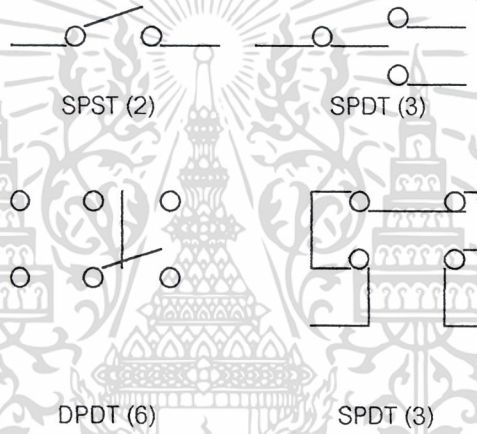


รูปที่ 2.5 แสดงการสร้างลอจิกจากสวิตช์

สวิตช์ที่นิยมใช้กันมีหลายชนิดดังรูปที่ 2.6 แต่ละแบบอาจแบ่งได้จากจำนวนหน้าสัมผัสและจำนวนขั้วของมัน ได้แก่แบบ ซิงเกิล-โพล/ซิงเกิล-โทล (Single-pole/Single – throw , SPST) , ซิงเกิล-โพล / ดับเบิล-โทล (Single - pole / Double - throw , SPDT) , ดับเบิล-โพล / ดับเบิล-โทล (Double – pole / Double – throw , DPDT) เป็นต้น สัญลักษณ์แสดงได้ดังรูปที่ 2.7 นอกจากนี้ยังมีสวิตช์ที่มีโครงสร้างภายในเป็น แบบหมุน ซึ่งจะให้สัญญาณออกมาเป็นรหัส BCD ได้เลยเรียกว่า ทัมวีลสวิตช์ (Thumbwheel Switch) การใช้สวิตช์แบบนี้จะต้องป้อนสัญญาณเข้าที่ขาคอมมอน (Common)

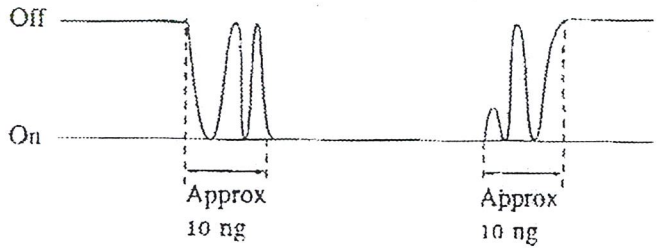


รูปที่ 2.6 แสดงตัวอย่างสวิตช์ที่นิยมใช้



รูปที่ 2.7 แสดงสัญลักษณ์ของสวิตช์แบบต่างๆ

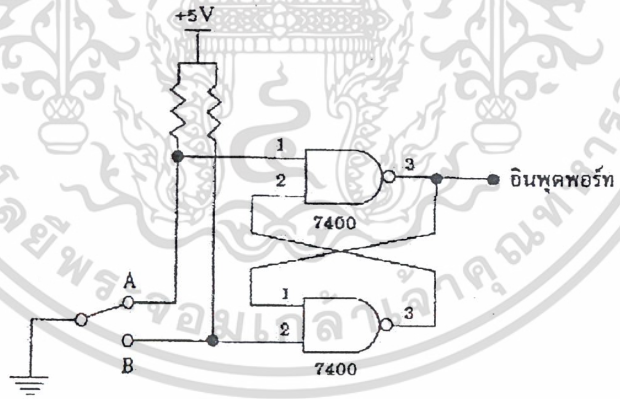
การต่อสวิตช์เข้ากับระบบไมโครคอนโทรลเลอร์จะต้องให้หน้าสัมผัสของสวิตช์สะอาดอยู่เสมอ พิจารณาตามรูปที่ 2.5 ถ้าสวิตช์ไม่มีการกดสัญญาณที่ได้จะมีค่าเป็น “1” ถ้าสวิตช์มีการกดข้อมูลที่ได้จะมีค่าเป็น “0” แต่โดยทั่วไปแล้วการเปลี่ยนระดับสัญญาณจาก “1” เป็น “0” จะมีสัญญาณที่ไม่ต้องการเกิดขึ้น ดังรูปที่ 2.8 ซึ่งเกิดจากการสั่นของหน้าสัมผัสของสวิตช์ ทำให้เกิดการแกว่งของสัญญาณซึ่งเรียกว่าการ บาวนซ์ (Bounce) อยู่ชั่วระยะหนึ่ง โดยปกติจะมีเวลาประมาณ 5 ถึง 50 มิลลิวินาที ดังนั้นสิ่งที่ MCS-51 ได้รับจากการกดสวิตช์จะไม่ใช่สัญญาณหนึ่งลูก แต่เป็นสัญญาณหลายๆลูก ซึ่ง MCS-51 อาจได้รับข้อมูลผิดพลาด การแก้ปัญหานี้เรียกว่า การทำ ดีบาวนซ์ (Debounce) ซึ่งอาจทำได้ 2 วิธีคือ ฮาร์ดแวร์ดีบาวนซ์ (Hardware Debounce) และ ซอร์ฟแวร์ดีบาวนซ์ (Software Debounce)



รูปที่ 2.8 แสดงสัญญาณที่เกิดจากการสั่นของหน้าสัมผัสของสวิตช์

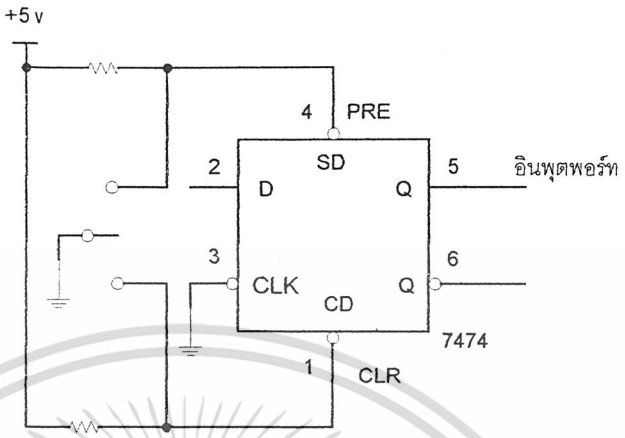
2.2.1 ฮาร์ดแวร์ดีบาวนซ์

การแก้โดยวิธีทางฮาร์ดแวร์วิธีง่ายๆวิธีหนึ่งคือการใช้ แนนเกต ต่อเป็น R-S ฟลิปฟลอป หรืออาจใช้ไอซีเบอร์ 74LS279 ก็ได้ พิจารณารูปที่ 2.9 เมื่อสวิตช์ถูกโยกมาที่ตำแหน่ง A ขาอินพุตของแนนเกตตัวบนจะได้รับลอจิก "0" ซึ่งจะทำให้เอาต์พุตของเกตตัวบนมีลอจิกเป็น "1" เอาต์พุตนี้จะถูกนำไปใช้งาน ขณะเดียวกันก็จะป้อนกลับมาให้กับอินพุตของแนนเกตตัวล่าง ทำให้เอาต์พุตของแนนเกตตัวล่างมีค่าเป็น "0" เมื่อมีการแกว่งของสัญญาณที่จุด A แนนเกตตัวบนจะได้รับลอจิกอินพุตที่เปลี่ยนแปลง แต่ไม่มีผลทำให้เอาต์พุตเปลี่ยนแปลง



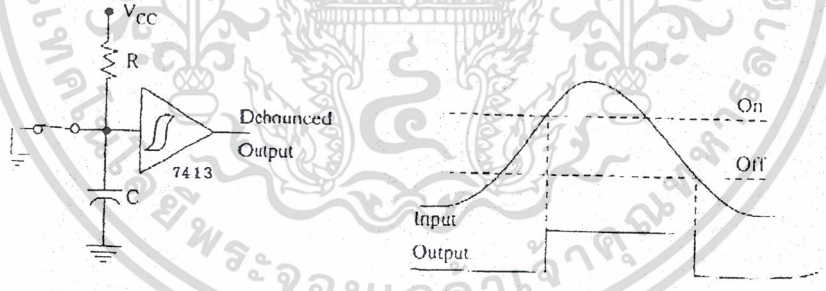
รูปที่ 2.9 แสดงการนำแนนเกตมาต่อเป็น R-S ฟลิปฟลอป เพื่อแก้การสั่นของสวิตช์

การแก้ปัญหาอีกวิธีหนึ่งอาจนำเอา D ฟลิปฟลอป มาช่วยดังแสดงในรูปที่ 2.10 โดยใช้ไอซีเบอร์ 7447 โดยให้อินพุตจากสวิตช์ต่อเข้าที่ขาพรีเซท (Preset) และ เคลียร์ (Clear) ถ้าขาพลีเซทเป็น "0" เอาต์พุต จะเป็น "1" ถ้ามีการแกว่งของสัญญาณเกิดขึ้นจะไม่มีผลต่อเอาต์พุต และเมื่อสวิตช์ถูกทำให้ เคลียร์เป็น "0" เอาต์พุตที่ได้จะเป็นลอจิก "0"



รูปที่ 2.10 การใช้ฟลิปฟลอปมาแก้ปัญหาสวิตช์

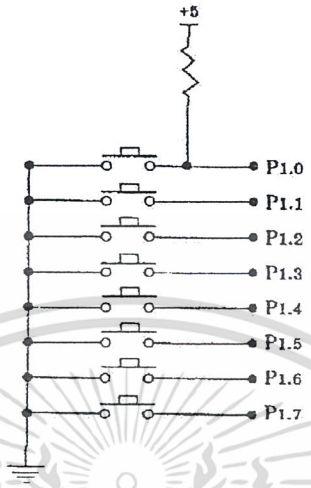
ถ้าใช้สวิตช์แบบ SPST อาจแก้ได้โดยใช้ D ฟลิปฟลอป โดยจะได้สัญญาณอินพุตเมื่อมี คล็อก (Clock) เข้าไปหรืออาจใช้ สมิททริกเกอร์ (Schmitt Trigger) ถ้าสัญญาณอินพุตจากสวิตช์มีลักษณะไม่เหมาะกับระบบดิจิตอล ดังรูปที่ 2.11



รูปที่ 2.11 การแก้ปัญหาสวิตช์โดยใช้สชมิททริกเกอร์

2.2.2 การต่อสวิตช์เข้ากับพอร์ทของ MCS-51 โดยตรง

ตามที่ทราบมาแล้วว่า MCS-51 มีพอร์ทให้ใช้งานอยู่หลายพอร์ท ในที่นี้จะยกตัวอย่างการต่อสวิตช์ 8 ตัวเข้ากับพอร์ท 1 ดังแสดงในรูปที่ 2.12 ระบบนี้เป็นการต่อสวิตช์แบบง่ายที่สุดเหมาะสำหรับระบบที่ไม่ต้องการสวิตช์มากนัก สำหรับการเขียนโปรแกรมจะต้องอ่านค่าจากสวิตช์คืออ่านค่าจากพอร์ท 1 จากนั้นต้องเขียนโปรแกรมตรวจสอบการสั่นของสวิตช์ เมื่อได้ค่าจากการกดสวิตช์แน่นอนแล้วก็ทำการตรวจสอบว่าค่าที่อ่านได้นั้นเป็นค่าจากการกดสวิตช์ใด



รูปที่ 2.12 แสดงการต่อสวิตช์เข้ากับพอร์ท P1

ต่อไปจะยกตัวอย่างการเชื่อมต่อสวิตช์เข้ากับพอร์ทของ 8255 โดยใช้สวิตช์แบบดิพสวิตช์เชื่อมต่อกับพอร์ท A ของ 8255 การเขียนโปรแกรมอย่างง่ายๆอาจทำได้โดยอ่านค่าจากพอร์ท A แล้วทำการตรวจสอบว่าสวิตช์ใด ON โดยนำค่าที่อ่านได้ส่งให้โปรแกรมประมวลผล ซึ่งการเขียนโปรแกรมอาจทำได้หลายวิธีในที่นี้จะยกตัวอย่างง่ายๆ โดยอ่านค่าเข้ามาแล้วตรวจสอบว่า สวิตช์แรก ON หรือไม่ ถ้าไม่ใช้ตรวจสวิตช์ที่สอง ทำไปเรื่อยๆจนครบ สมมติว่าถ้าเรามีโปรแกรมต่างๆหลายๆ โปรแกรมและเก็บโปรแกรมเหล่านั้นเป็นโปรแกรมย่อย จากนั้นจะเลือกทำโปรแกรมต่างๆ โดยการกดสวิตช์ เราอาจเชื่อมต่อสวิตช์เข้ากับพอร์ท A ของ 8255 จากนั้นเขียนโปรแกรมให้อ่านค่าจากสวิตช์เพื่อเลือกทำโปรแกรมย่อยต่างๆอาจเขียนได้ดังนี้

```
PORT_A      EQU         0FC00H
MAIN :      ACALL       APORT
            CJNE   A,#01,KEY1
            ACALL       PRO1
            SJMP        MAIN
KEY1 :      CJNE   A,#02,KEY2
ACALL       PRO2
SJMP        MAIN
```

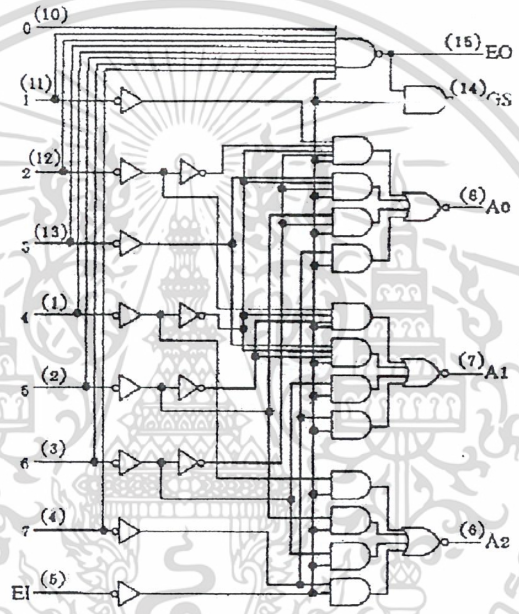
```
KEY2 :      CJNE   A,#03,KEY3
ACALL       PRO3
SJMP        MAIN

.....
.....
KEY7 :      CJNE   A,#128,MAIN
            ACALL       PRO7
            SJMP        MAIN
;*************************************
APORT :     MOV    DPTR,#PORT_A
            MOVX        A,@DPTR
            RET
;*************************************
```

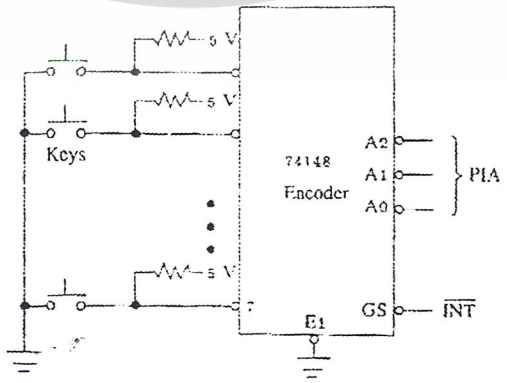
2.2.3 การต่อสวิตช์เป็นจำนวนมาก

จะเห็นว่าสวิตช์หนึ่งตัวจะใช้พอร์ท 1 บิต ดังนั้นพอร์ทหนึ่งพอร์ทจะต่อสวิตช์ได้ 8 ตัว ถ้าหากระบบของเราต้องการสวิตช์มากกว่านี้อาจออกแบบได้หลายวิธี ในที่นี้จะยกตัวอย่างการเข้ารหัสสวิตช์และการต่อสวิตช์แบบเมทริกซ์ การเข้ารหัสสวิตช์ (Encoding Switches) ในที่นี้จะยกตัวอย่างการใช้ไอซีเข้ารหัสเบอร์ 74LS148 ที่ทำงานเป็นแบบ 8 To 3 ซึ่งเป็นไอซีเข้ารหัสเป็นเลขไบนารี 3 บิตถ้าเราให้อินพุตขาใดขาหนึ่งเป็นลอจิก "0" จะให้เอาต์พุตเป็นไบนารีที่มีค่าสัมพันธ์กับขานั้น การทำงานของไอซีเบอร์นี้ดูได้ ดังรูปที่2.13 การใช้ไอซีเบอร์นี้มาสร้างเป็นวงจรเข้ารหัสสวิตช์เราสามารถนำเอาต์พุตไบนารีต่อ โดยตรงเข้ากับพอร์ทได้และอินพุตได้เลยในขณะเดียวกันถ้ามีการกดสวิตช์ใดสวิตช์หนึ่งสัญญาณที่ขา GS จะเป็น ลอจิก "0" ทันทีซึ่งเราสามารถนำสัญญาณนี้ไปอินเทอร์รัพท์ MCS-51 ได้ ดังนั้นการใช้ไอซีเบอร์นี้สามารถเขียนโปรแกรมได้ทั้งแบบ โพลลิ่ง (Polling) และแบบ อินเทอร์รัพท์ (Interrupt) การต่อคีย์สวิตช์เข้ากับไอซีเบอร์นี้แสดงได้ ดังรูปที่ 2.14

Input									Outputs				
EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	X	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H



รูปที่ 2.13 แสดงลักษณะ ไอซีเบอร์ 74LS148



รูปที่ 2.14 การเชื่อมต่อสวิตช์เข้ากับ 74LS148

2.2.4 การต่อสวิตช์แบบเมทริกซ์

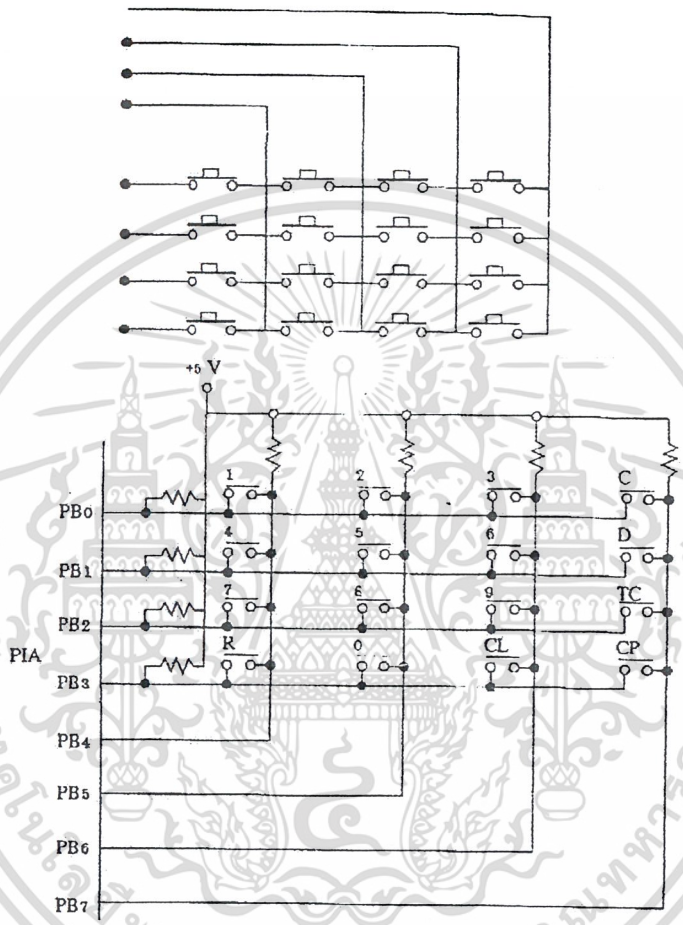
หลายคนคงเคยเห็นคีย์สวิตช์แบบเมทริกซ์ ที่เรียกว่า คีย์แพด (Keypad) การต่อสวิตช์แบบนี้จะใช้พอร์ท 2 พอร์ทคือ พอร์ทเอาต์พุตและพอร์ทอินพุต ในที่นี้จะยกตัวอย่างการต่อคีย์แพด ขนาด 4 × 4 คีย์เข้ากับพอร์ท C ของ 8255 ซึ่งโปรแกรมให้พอร์ท C ล่างเป็นพอร์ทอินพุตและพอร์ท C บนเป็นพอร์ทเอาต์พุต การต่อสวิตช์ลักษณะนี้จะต้องโปรแกรมสแกนคีย์สวิตช์เพื่อที่จะเช็คว่าสวิตช์ใดถูกกด ซึ่งทำได้โดยส่งค่าออกไปที่พอร์ทเอาต์พุตให้เป็นลอจิก "0" ทีละบิตแล้วอ่านค่าเข้าทางพอร์ทอินพุต จากนั้นตรวจเช็คดูว่าคีย์สวิตช์ใดถูกกดหรือไม่ จากนั้นให้บิตต่อไปของพอร์ทเอาต์พุตเป็น "0" แล้วอ่านค่าเข้ามาใหม่ การส่งค่าสแกนไปที่พอร์ทเอาต์พุตนี้จะต้องทำอย่างเร็วเพื่อที่จะตรวจสอบการกดสวิตช์ได้ทัน จากตัวอย่างวงจรในรูปที่ 2.15 จะทำการสแกนคีย์สวิตช์ได้โดย ขั้นแรกจะส่งข้อมูลไปที่พอร์ท C บนโดยส่งค่าบิต PC4 เป็น "0" ก่อนคือส่งค่า 1110 จากนั้นอ่านค่าเข้าทางพอร์ท C ล่าง ถ้าสวิตช์ 1 ถูกกดค่าที่อ่านได้จะเป็น 1110 ถ้าสวิตช์ 4 ถูกกด ค่าอ่านได้จะเป็น 1101 ถ้าไม่มีสวิตช์ใดในหลักที่ 1 ถูกกดเลยค่าที่อ่านเข้ามาได้จะเป็น 1111 จังหวะต่อไปให้ PB5 เป็น "0" โดยส่งค่าออกไปเป็น 1101 แล้วอ่านค่าเข้ามา ถ้าอ่านมาได้เป็น 1111 แสดงว่าหลักที่ 2 ไม่มีการกด ถ้าอ่านเข้ามาได้เป็น 1101 แสดงว่าสวิตช์ 5 ถูกกด แล้วทำการสแกนแบบนี้ไปเรื่อยๆค่าของพอร์ท C ของ 8255 เมื่อมีการกดคีย์สวิตช์ใดๆจะเป็นไปตามตารางที่ 2.3

KEY	PC7 - PC0	KEY	PC7 - PC0
1	1110 1110	7	1110 1011
2	1101 1110	8	1101 1011
3	1011 1110	9	1011 1011
C	0111 1110	TC	0111 1011
4	1110 1101	R	1110 0111
5	1101 1101	O	1101 0111
6	1011 1101	CL	1011 0111
D	0111 1101	CP	0111 0111

ตารางที่ 2.3 แสดงค่าจากพอร์ท C เมื่อกดสวิตช์

การเขียนโปรแกรมสแกนสวิตช์ระบบไมโครคอนโทรลเลอร์ต้องสแกนตลอดเวลาถ้าไม่สแกนแล้วหากมีการกดสวิตช์ระบบจะไม่รับรู้ การต่อสวิตช์แบบเมทริกซ์นี้สามารถใช้วิธีอินเตอร์รัพท์การต่อแบบนี้ MCS-51 ไม่ต้องเสียเวลาสแกนคีย์สวิตช์ เมื่อมีการกดสวิตช์จะมีสัญญาณไป

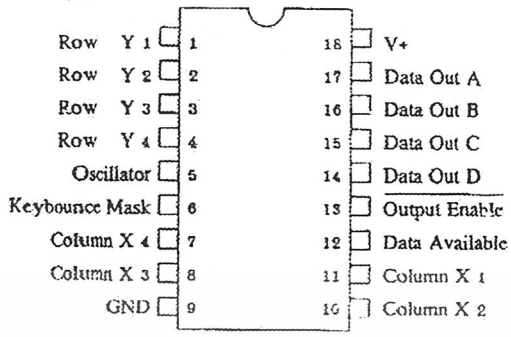
อินเตอร์รัพท์ MCS-51 จากนั้นจะทำโปรแกรมตอบสนองการอินเตอร์รัพท์คือโปรแกรมสแกนคีย์สวิตช์แต่อย่าลืมว่าจะต้องมีการตรวจสอบการแกว่งของสัญญาณสวิตช์ด้วย

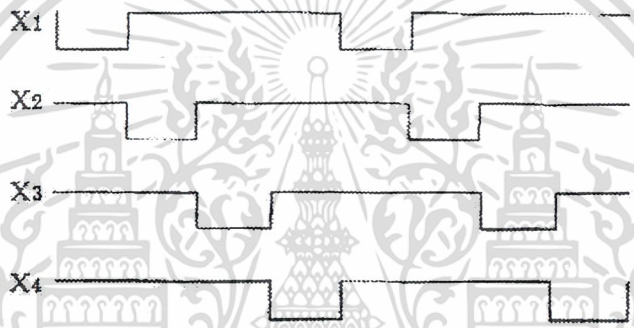


รูปที่ 2.15 แสดงการต่อพอร์ทแบบเมทริกซ์เข้ากับพอร์ท C ของ 8255

2.2.5 การใช้ไอซีสแกนคีย์สวิตช์

ในปัจจุบันได้มีการออกแบบไอซีสแกนคีย์สวิตช์แบบเมทริกซ์ โดยเอาต์พุตที่ออกจากไอซีจะเป็นเลขไบนารี จากนั้นนำไปต่อกับพอร์ทของระบบไมโครคอนโทรลเลอร์ทำให้ระบบไมโครคอนโทรลเลอร์ไม่ต้องเสียเวลาในการทำโปรแกรมสแกนคีย์สวิตช์ ในที่นี้จะใช้ไอซีเบอร์ 74C922





รูปที่ 2.16 แสดงไอซีสแกนคีย์สวิตช์

	Data Output			
Switch	D	C	B	A
Y1,X1	0	0	0	0
Y1,X2	0	0	0	1
Y1,X3	0	0	1	0
Y1,X4	0	0	1	1
Y2,X1	0	1	0	0
Y2,X2	0	1	0	1
Y2,X3	0	1	1	0
Y2,X4	0	1	1	1
Y3,X1	1	0	0	0
Y3,X2	1	0	0	1
Y3,X3	1	0	1	0
Y3,X4	1	0	1	1
Y4,X1	1	1	0	0
Y4,X2	1	1	0	1
Y4,X3	1	1	1	0
Y4,X4	1	1	1	1

ตารางที่ 2.4 แสดงไอซีสแกนคีย์สวิตช์

ตัวอย่างต่อไปเป็นการอ่านค่าจากพอร์ท P1 ของ MCS-51 ซึ่งต่อจากไอซีสแกนคีย์สวิตช์ เมื่อเรากดสวิตช์ใดๆเข้าไปจะแสดงค่าทางไดโอดเปล่งแสง 7 ส่วน จากโปรแกรมจะเห็นว่าจะอ่านค่าสวิตช์มาเก็บไว้ที่หน่วยความจำ DATA_1 จากนั้นจะเรียกโปรแกรมย่อยแสดงผลให้แสดงทางไดโอดเปล่งแสง 7 ส่วน

```
PORT_A      EQU     0FC00H
PORT_B      EQU     0FC01H
PORT_C      EQU     0FC02H
PORT_CON    EQU     0FC03H
;************************************
MAIN :      MOV     A,#88H
            MOV     DPTR,#PORT_CON
            MOVX    @DPTR,A
;************************************
DATA_1      EQU     20H             ; หน่วยความจำเก็บค่าจากสวิตช์
;************************************
RUN :
            MOV     A,P1            ; อ่านค่าจากพอร์ท 1 เข้ามา
            ANL     A,#00001111B    ; ตัดสี่บิตบนทิ้งเนื่องจากสวิตช์ต่อกับสี่บิตล่าง
            MOV     DATA_1,A        ; นำค่าที่ได้ไปเก็บในหน่วยความจำ DATA_1
            CALL    DISPLAY         ; เรียกโปรแกรมแสดงผล
            SJMP    RUN
;************************************
DISPLAY :   MOV     R3,#09H
SHOW1 :     MOV     R2,DATA_1
            MOV     DPTR,#SEG_CODE
            MOV     A,R2
            MOVC    A,@A+DPTR
            MOV     DPTR,#PORT_A
            MOVX    @DPTR,A
            MOV     A,#00H
```

```
        MOV     DPTR,#PORT_C
        MOVX    @DPTR,A
        RET
;*************** END DISPLAY ***************
SEG_CODE :  DB   3FH,06H,5BH,4FH,66H
            DB   6DH,7DH,07H,7FH,0FH
```

2.3 การเชื่อมต่อพอร์ทอนุกรมกับระบบบัส I^2C

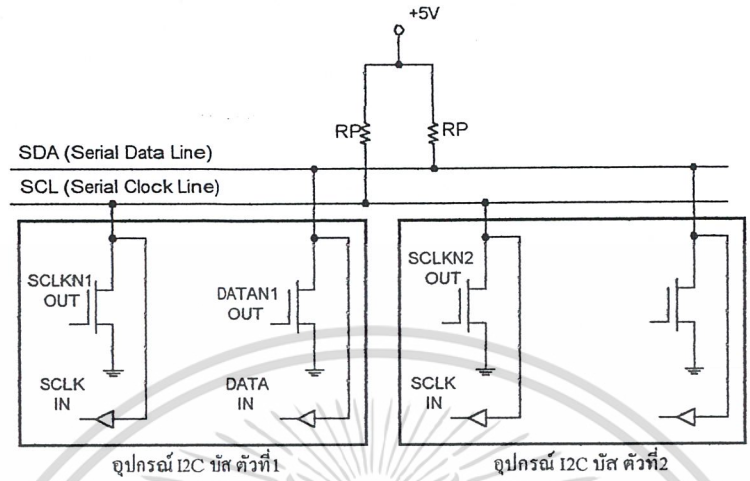
I^2C ย่อมาจาก Inter-IC Communication หมายถึง การติดต่อสื่อสารระหว่างไอซี โดยบัส I^2C ได้รับการพัฒนาขึ้นโดยฟิลิปส์ (Philips) ด้วยจุดมุ่งหมายหลักคือ ต้องการให้ไอซีหรือโมดูลสามารถติดต่อสั่งงาน และควบคุมภายใต้สายสัญญาณเพียง 2 เส้น เส้นหนึ่งคือ สายข้อมูล อีกเส้นหนึ่งคือ สายสัญญาณนาฬิกาที่ใช้ในการกำหนดจังหวะการทำงาน การต่อร่วมกันของอุปกรณ์บนบัส I^2C ทำได้ง่ายมากเพียงต่อสายข้อมูลและสายสัญญาณนาฬิกาของอุปกรณ์แต่ละตัวขนานหรือพ่วงกันไปส่วนการกำหนดแอดเดรสหรือตำแหน่งสำหรับติดต่ออุปกรณ์แต่ละตัว จะใช้รหัสข้อมูลและการกำหนดสภาวะลอจิกที่ขาแอดเดรสของอุปกรณ์แต่ละตัว

สายข้อมูลบนบัส I^2C มีชื่อเรียกอย่างเป็นทางการว่า สายข้อมูลอนุกรม หรือ SDA (Serial Data line) ส่วนสายสัญญาณนาฬิกามีชื่อเรียกว่า สายสัญญาณนาฬิกาอนุกรม หรือ SCL (Serial Clock line) ในการอธิบายต่อไปนี้จะเรียกสายสัญญาณทั้งสองว่า สาย SDA หรือ SCL

อุปกรณ์ที่ใช้การเชื่อมต่อด้วยบัส I^2C มีหลากหลาย ไม่ว่าจะเป็นไอซีขยายพอร์ทอินพุตเอาต์พุต (I/O Expander) , ไอซีแปลงสัญญาณอะนาลอกเป็นดิจิตอล (ADC) และแปลงสัญญาณดิจิตอลเป็นอะนาลอก (DAC) , ไอซีรีลไทม์คล็อก (RTC) , ไอซีขับโมดูล LCD , หน่วยความจำอีอีพรอมและไมโครคอนโทรลเลอร์

2.3.1 คุณสมบัติโดยทั่วไปของบัส I^2C

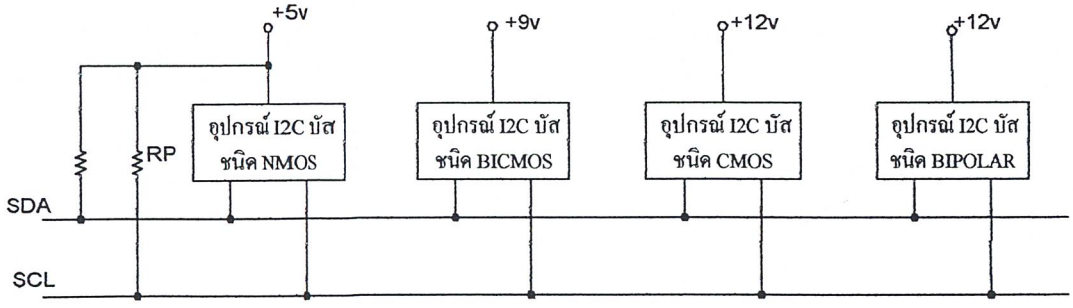
สาย SDA และ SCL เป็นสายสัญญาณ 2 ทิศทาง (bi – direction line) ต้องมีการต่อตัวต้านทานพูลอัปกับแรงดัน +5V ไว้ตลอดเวลา เพื่อให้สายมีสถานะลอจิกสูงในขณะที่ไม่มีการติดต่อใช้งาน ทั้งยังช่วยในการป้องกันสัญญาณรบกวนที่อาจมีเข้ามาในสายสัญญาณทั้งสองวงจรเอาต์พุตของอุปกรณ์ที่ต่ออยู่บนบัส I^2C ต้องมีลักษณะเป็นวงจรเดรนเปิด (open – drain) หรือคอลเล็กเตอร์เปิด (open collector) ดังแสดงรายละเอียดในรูปที่ 2.17



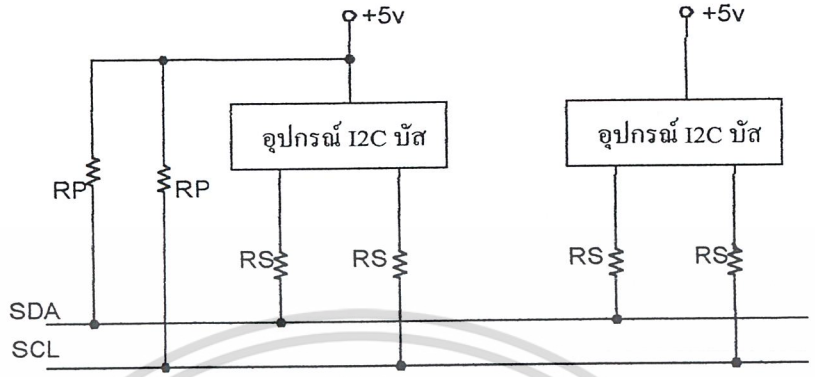
รูปที่ 2.17 โครงสร้างวงจรเอาต์พุตของอุปกรณ์ที่ใช้การเชื่อมต่อบนระบบบัส I^2C

อัตราการถ่ายทอดข้อมูลบนบัส I^2C สูงถึง 100 กิโลบิตต่อวินาทีในโหมดปกติ (standard mode) และสูงถึง 400 กิโลบิตต่อวินาทีในโหมดความเร็วสูง (fast mode) อุปกรณ์ที่ต่อร่วมอยู่บนบัส I^2C จะต้องมีค่าความจุไฟฟ้ารวมที่เกิดขึ้นระหว่างสาย SDA และ SCL ไม่เกิน 400pF การเข้าถึงอุปกรณ์บนบัส I^2C ใช้ข้อมูลสำหรับการเข้าถึง 2 ค่าคือ 7 บิต (7-bit addressing) หรือ 10 บิต (10 – bit addressing)

ข้อเด่นอีกประการหนึ่งของบัส I^2C คือสามารถเชื่อมต่ออุปกรณ์ที่ใช้ไฟเลี้ยงไม่เท่ากันให้สามารถติดต่อสื่อสารกันได้ โดยอุปกรณ์บนบัส I^2C ตัวหนึ่งอาจใช้ไฟเลี้ยง +5V ในขณะที่อีกตัวหนึ่งใช้ไฟเลี้ยง +12V การต่อร่วมกันบนบัส I^2C สามารถกระทำได้ในลักษณะเดียวกับกรณีที่อุปกรณ์ทั้งสองใช้ไฟเลี้ยงเท่ากัน กล่าวคือให้ต่อสาย SDA และ SCL ของอุปกรณ์แต่ละตัวเข้าด้วยกัน และต้องต่อตัวต้านทานพูลอัป (Rp) เข้ากับแรงดัน +5V ไว้ด้วยเสมอ ดังแสดงในรูปที่ 2.18



รูปที่ 2.18 แสดงการเชื่อมต่ออุปกรณ์บนระบบบัส I^2C ที่ใช้ไฟเลี้ยงไม่เท่ากัน



รูปที่ 2.19 การต่อตัวต้านทานเพื่อป้องกันแรงดันกระชากที่อาจปะปนเข้ามาในไฟเลี้ยงของอุปกรณ์ในระบบบัส I^2C

ในกรณีที่อาจมีแรงดันไฟกระชากขนาดใหญ่ปะปนเข้ามาในบัส I^2C ที่ขา SDA และ SCL ของอุปกรณ์แต่ละตัวต้องต่อตัวต้านทานอนุกรมกับขา SDA และ SCL เรียกว่า Rs ก่อนต่อเข้าสู่บัส I^2C ดังแสดงในรูปที่ 2.19

2.3.2 หลักการของบัส I^2C

บัส I^2C ประกอบด้วยสายสัญญาณ 2 เส้น คือ SDA และ SCL อุปกรณ์ที่ต่อพ่วงบนบัสสามารถมีได้มากมาย ดังนั้นจึงต้องมีการกำหนดรูปแบบของการติดต่อบนบัส หรือเรียกว่าโปรโตคอล (protocol) เพื่อให้ผู้ใช้งานทราบว่า ขณะนี้อุปกรณ์ใดติดต่อกันอยู่ และอุปกรณ์ตัวใดเป็นตัวรับหรือตัวส่ง ต่อไปนี้จะขออธิบายลักษณะ หน้าที่และนิยามของอุปกรณ์ที่ต่ออยู่บนบัส I^2C เพื่อเป็นข้อตกลงพื้นฐานก่อนที่จะอธิบายการทำงานของบัส I^2C ต่อไป

อุปกรณ์ที่เป็นผู้สร้างข้อมูลหรือส่งข้อมูล เรียกว่า ตัวส่ง (transmitter)

อุปกรณ์ที่เป็นผู้รับข้อมูล เรียกว่า ตัวรับ (receiver)

ในอุปกรณ์บนบัส I^2C สามารถเป็นได้ทั้งตัวรับและตัวส่ง บางอุปกรณ์ทำหน้าที่เป็นตัวรับเพียงอย่างเดียว จะไม่มีอุปกรณ์ใดบนบัส I^2C ที่ทำหน้าที่เป็นตัวส่งเพียงอย่างเดียว

อุปกรณ์ที่ทำหน้าที่ควบคุมจังหวะการทำงานหรือการติดต่อบนบัส I^2C เรียกว่ามาสเตอร์ (master)

อุปกรณ์ที่ถูกควบคุมหรืออุปกรณ์ที่ต่อพ่วงเข้าไปบนบัส I^2C เรียกว่า สเลฟ (slave)

ข้อกำหนด 2 ประการสำคัญของการติดต่อบนบัส I^2C คือ

(1) การถ่ายทอดข้อมูลจะเกิดขึ้นได้เมื่อบัสว่างเท่านั้น

(2) ในระหว่างการถ่ายทอดข้อมูล เมื่อใดก็ตามที่สาย SCL มีสถานะเป็นลอจิกสูง สายข้อมูลต้องรักษาข้อมูลไว้ อย่าให้เกิดการเปลี่ยนแปลงขึ้นเด็ดขาด มิฉะนั้น สัญญาณที่เกิดขึ้นจะได้รับการแปลความหมายเป็นสัญญาณควบคุมแทน

2.3.3 สภาวะที่เกิดขึ้นบนบัส I^2C

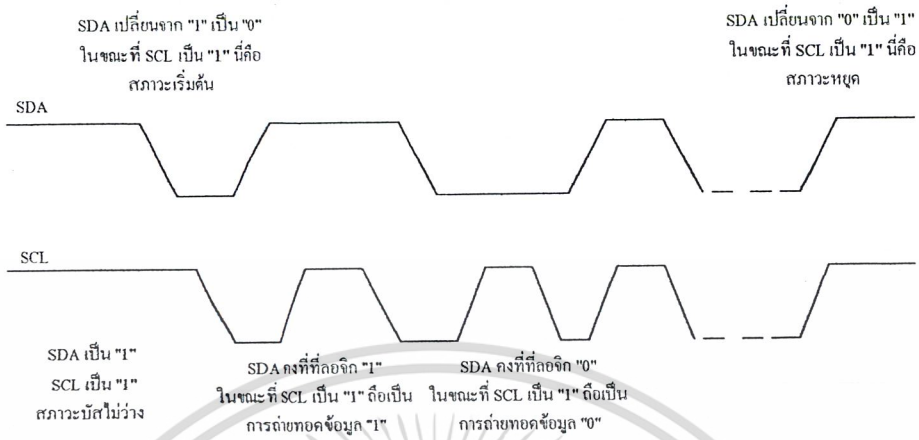
มีด้วยกัน 5 สภาวะดังนี้

(1) บัสว่าง (Bus not busy) สภาวะนี้เกิดขึ้นเมื่อสถานะลอจิกบนสาย SDA และ SCL เป็นลอจิกสูงทั้งคู่ นั่นหมายความว่า การถ่ายทอดข้อมูลสามารถเริ่มต้นขึ้นได้

(2) เริ่มต้นการถ่ายทอดข้อมูล (start data transfer) เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับลอจิกจากสูงไปต่ำ ในขณะที่สาย SCL มีสถานะลอจิกสูง เรียกสภาวะที่เกิดขึ้นนี้ว่า สภาวะเริ่มต้น (START)

(3) ข้อมูลดำรงอยู่บนบัส (data valid) สภาวะนี้เกิดขึ้นถัดจากสภาวะเริ่มต้น โดยสถานะลอจิกที่เกิดขึ้นบนสาย SDA ก็คือข้อมูลที่ทำการถ่ายทอด เมื่อสาย SCL เป็นลอจิกสูง สถานะที่สาย SDA ต้องคงที่ เพื่อให้อุปกรณ์รับรู้ข้อมูลในจังหวะนั้นว่า เป็น “0” หรือ “1” ข้อมูลอาจเกิดการเปลี่ยนแปลงได้ในขณะที่สาย SCL เป็นลอจิกต่ำ แต่เมื่อใดก็ตามที่ต้องการให้เกิดการถ่ายทอดข้อมูลอย่างสมบูรณ์ สถานะลอจิกที่ขา SDA ต้องคงที่ตลอดช่วงเวลาที่สาย SCL มีสถานะลอจิกสูง หากเกิดการเปลี่ยนแปลงสถานะลอจิกในขณะที่สาย SCL มีลอจิกสูงอยู่นั้น อุปกรณ์มาสเตอร์ที่ทำการควบคุมการถ่ายทอดข้อมูลจะแปลความหมายเป็นสภาวะหยุดหรือสภาวะเริ่มต้นก็ได้ ทำให้ข้อมูลที่ทำการถ่ายทอดนั้นเกิดความผิดพลาดขึ้น

(4) รับรู้ข้อมูล (acknowledge) เกิดขึ้นหลังจากที่การถ่ายทอดข้อมูลจากตัวส่งมายังตัวรับเกิดขึ้นอย่างสมบูรณ์ โดยตัวส่งจะทำการส่งข้อมูลมา 1 บิตเรียกว่า บิตรับรู้ (acknowledge bit) มีสถานะเป็นลอจิกสูง หลังจากที่ส่งข้อมูลมาครบถ้วน ส่วนอุปกรณ์มาสเตอร์จะทำการส่งสัญญาณรับรู้พิเศษซึ่งสัมพันธ์กับสัญญาณนาฬิกา อุปกรณ์สเลฟที่ถูกอ้างถึงในการติดต่อหรือกำลังติดต่ออยู่ในขณะนั้นก็จะกำเนิดบิตรับรู้ที่มีสถานะลอจิกต่ำเพื่อตอบสนองให้ทราบว่าได้รับข้อมูลเรียบร้อยแล้ว



รูปที่ 2.20 ไดอะแกรมเวลาแสดงสภาวะต่างๆที่เกิดขึ้นบนระบบบัส I^2C

(5) หยุดการถ่ายทอดข้อมูล (stop data transfer) เกิดขึ้นเมื่อสาย SDA มีการเปลี่ยนแปลงระดับลอจิกจากต่ำไปสูง ในขณะที่สาย SCL มีสถานะลอจิกสูง เรียกสภาวะที่เกิดขึ้นนี้ว่า สภาวะหยุด (STOP)

ในรูปที่ 2.20 เป็นไดอะแกรมเวลาที่แสดงถึงการเกิดสภาวะต่างๆบนบัส I^2C ไม่ว่าจะเป็นสภาวะบัสว่าง , เริ่มต้น , ถ่ายทอดข้อมูล , รับรู้และหยุดการถ่ายทอดข้อมูล

2.3.4 การทำงานบนบัส I^2C

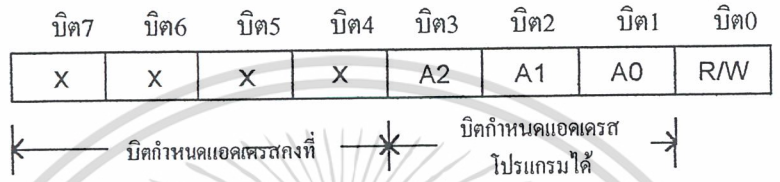
ก่อนที่จะเริ่มต้นการถ่ายทอดข้อมูลระหว่างอุปกรณ์ต่างๆที่ต่ออยู่บนบัส ต้องมีการอ้างถึงอุปกรณ์เสียก่อน โดยการอ้างถึงอุปกรณ์บนบัส I^2C นั้นจะใช้การอ้างถึงแบบ 7 บิตหรือ 10 บิต ในกรณีที่มีอุปกรณ์ต่ออยู่บนบัสไม่มาก ใช้การอ้างถึงแบบ 7 บิตก็เพียงพอ แต่ถ้ามีอุปกรณ์ต่ออยู่บนบัสมากกว่า 127 แอดเดรส จำเป็นต้องใช้การอ้างถึงแบบ 10 บิต หลังจากที่ติดต่ออุปกรณ์แต่ละตัวไปเรียบร้อยแล้ว ก็จะเริ่มต้นการถ่ายทอดข้อมูลกันต่อไป

ดังนั้นหัวใจสำคัญในอันดับแรกของการทำงานบนบัส I^2C คือการอ้างถึงอุปกรณ์แต่ละตัว ซึ่งในที่นี้จะอธิบายรายละเอียดของการอ้างถึงทั้ง 2 รูปแบบ

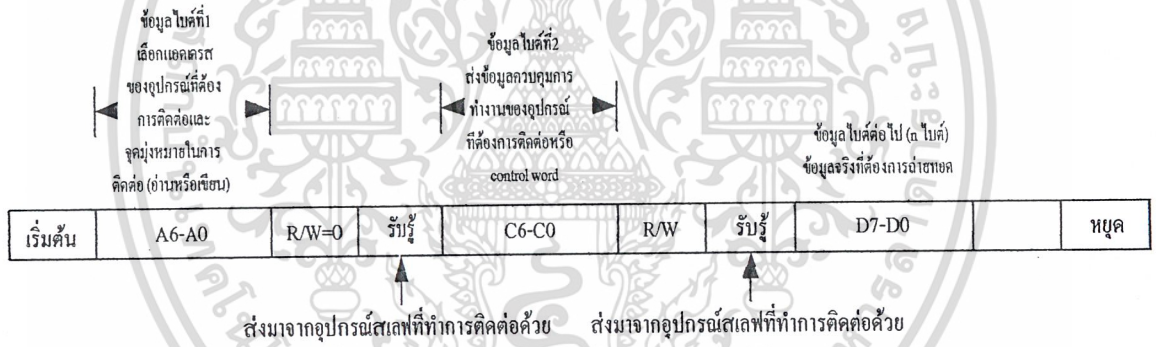
2.3.4.1 การอ้างถึงแบบ 7 บิต (7-bit addressing)

ข้อมูลไบต์แรกที่เกิดขึ้นหลังจากสภาวะเริ่มต้นคือ ข้อมูลที่ใช้ในการอ้างถึงอุปกรณ์ที่ต้องการติดต่อ โดยมีรูปแบบแสดงในรูปที่ 2.21 ใน 7 บิตบนรวมทั้งบิต MSB ด้วยจะเป็นข้อมูลแอดเดรสของอุปกรณ์สเลฟที่ต้องการติดต่อ โดยแบ่งเป็นบิตกำหนดแอดเดรสคงที่ (fixed address bit) จำนวน 4 บิต ซึ่งข้อมูลนี้อุปกรณ์แต่ละตัวจะถูกกำหนดมาจากผู้ผลิตไม่สามารถเปลี่ยนแปลงแก้ไขได้ ถัดมาอีก 3 บิต

เป็นบิตกำหนดแอดเดรสที่สามารถโปรแกรมได้ (programmable address bit) โดยผู้ใช้งานต้องกำหนดสถานะลอจิกให้แก่ขา A0-A2 ของอุปกรณ์ที่มีการเชื่อมต่อแบบบัส I^2C ส่วนในบิต LSB เป็นบิตที่ใช้กำหนดการอ่านหรือเขียนข้อมูลกับอุปกรณ์สเลฟตัวนั้นๆหากบิต LSB เป็น “0” หมายถึงต้องการเขียนข้อมูลไปยังอุปกรณ์นั้น ถ้าเป็น “1” จะเป็นการอ่านข้อมูลจากอุปกรณ์สเลฟ



รูปที่ 2.21 รูปแบบของข้อมูลที่ใช้ในการอ้างถึงอุปกรณ์บนระบบบัส I^2C



รูปที่ 2.22 รูปแบบข้อมูลอนุกรมที่ใช้ในการติดต่อกับอุปกรณ์บนระบบบัส I^2C แบบ 7 บิต

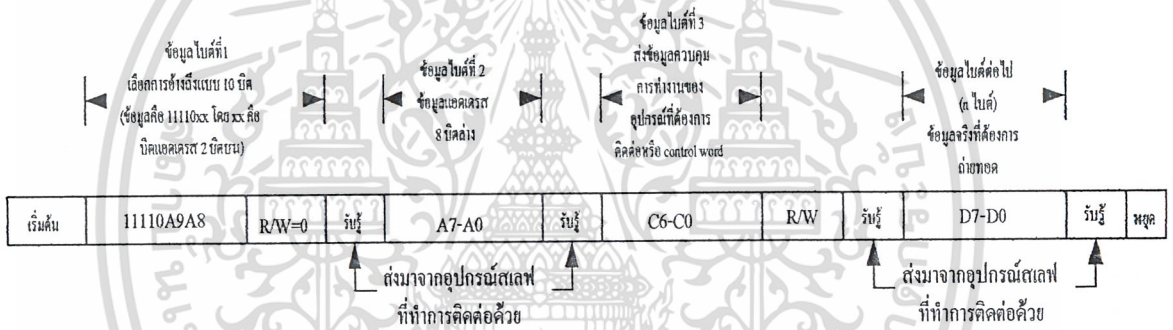
ข้อมูลในไบต์ต่อมาคือ ข้อมูลควบคุม (control byte) ในอุปกรณ์แต่ละตัวมีการกำหนดข้อมูลควบคุมที่แตกต่างกันไป ยกตัวอย่าง ไอซีขยายพอร์ทมีข้อมูลควบคุมที่ใช้กำหนดว่า บิตใดเป็นอินพุต บิตใดเป็นเอาต์พุตในขณะที่ไอซี ADC/DAC ต้องการข้อมูลควบคุมเพื่อกำหนดให้ทำงานเป็นวงจร ADC หรือ DAC เป็นต้น

ข้อมูลในไบต์ต่อมา คือ ข้อมูลที่ทำการถ่ายทอดจริง

หลังจากที่มีการถ่ายทอดข้อมูลในแต่ละไบต์ อุปกรณ์สเลฟที่ได้รับการติดต่อต้องส่งสัญญาณรับรู้ตอบกลับมาด้วยทุกครั้ง เพื่อให้กระบวนการถ่ายทอดข้อมูลสามารถดำเนินต่อไปได้ ในรูปที่ 2.22 แสดงรูปแบบข้อมูลอนุกรมที่เกิดขึ้นในการติดต่อบนบัส I^2C ของการอ้างถึงแบบ 7 บิต

2.3.4.2 การอ้างถึงแบบ 10 บิต

ในการอ้างถึงแบบนี้ยังคงใช้รูปแบบข้อมูลอนุกรมที่เหมือนกับแบบ 7 บิต หากแต่จะมีข้อมูลเพิ่มเติมขึ้นมาเล็กน้อย โดยในข้อมูลไบต์แรกหลังจากเกิดสภาวะเริ่มต้น ต้องกำหนดให้ 5 บิตบนมีข้อมูลเป็น 11110 ส่วนอีก 2 บิตถัดมาเป็นบิตแอดเดรสของอุปกรณ์ที่ต้องการติดต่อ ในบิต LSB ของข้อมูลไบต์แรกยังคงเป็นการกำหนดว่า ต้องการอ่านหรือเขียนข้อมูลกับอุปกรณ์สเลฟตัวที่ต้องการติดต่อด้วย ข้อมูลไบต์ถัดไปจึงเป็นข้อมูลควบคุม ข้อมูลหลังจากนั้นก็จะเป็นข้อมูลจริงที่ใช้ในการติดต่อ เช่นเดียวกับการอ้างถึงแบบ 7 บิต หลังจากถ่ายทอดข้อมูลครบทุกไบต์ ต้องมีสภาวะรับรู้เกิดขึ้นเพื่อให้กระบวนการถ่ายทอดข้อมูลสามารถดำเนินต่อไปได้ ในรูปที่ 2.23 แสดงรูปแบบข้อมูลอนุกรมของการอ้างถึงแบบ 10 บิต



รูปที่ 2.23 รูปแบบข้อมูลอนุกรมที่ใช้ในการติดต่อกับอุปกรณ์บนระบบบัส I^2C แบบ 10 บิต

2.4 การเขียนโปรแกรมภาษาแอสแซมบลี

การที่จะให้ MCS-51 ทำงานได้จะต้องให้คำสั่งแก่มัน การนำรหัสคำสั่งหลายๆรหัสคำสั่งมาเรียงกันให้ MCS-51 ทำงานเรียกว่าโปรแกรมภาษาแอสแซมบลี ซึ่งภาษานี้จะเป็นภาษาที่อยู่ระหว่างภาษาเครื่องกับภาษาระดับสูง โดยภาษาระดับสูงเช่น Pascal หรือ C จะใช้ชุดคำสั่งที่ใกล้เคียงกับภาษาที่มนุษย์เข้าใจได้ง่าย ส่วนภาษาเครื่องจะเป็นภาษาที่ใช้เลขฐานสองซึ่งคอมพิวเตอร์จะเข้าใจสำหรับ CPU แต่ละเบอร์ โปรแกรมภาษาเครื่องจะเป็นชุดจำนวนไบต์ของเลขฐานสองซึ่งขึ้นกับชุดคำสั่งที่จะให้คอมพิวเตอร์ทำงาน

ภาษาแอสแซมบลีจะแทนรหัสภาษาเครื่องเลขฐานสองด้วยรหัสที่เราเข้าใจได้ง่ายคือ นีโมนิก "Mnemonics" ถ้าหากให้ CPU บวกเลขจะต้องป้อนรหัสคำสั่งเป็นภาษาเครื่องว่า "10110011" ซึ่งถ้าเขียนเป็นภาษาแอสแซมบลี อาจแทนด้วย "ADD" การเขียนโปรแกรมภาษาแอสแซมบลีจะเขียนเป็น

รหัสนีโมนิก ทั้งหมดเพื่อให้เข้าใจได้ง่ายแต่ถ้าจะให้ CPU เข้าใจจะต้องแปลงรหัสนีโมนิกเหล่านี้ให้เป็นเลขฐานสอง ขั้นตอนนี้เรียกว่า Assembler ซึ่งอาจแปลงได้โดยการเปิดตารางรหัสคำสั่งของ CPU ที่กำลังศึกษาอยู่ หรือ แปลงโดยใช้โปรแกรมแอสแซมเบอร์ ในปัจจุบันการเขียนโปรแกรมภาษาแอสแซมบลีจะเขียนบนเครื่องคอมพิวเตอร์แล้วใช้โปรแกรมแอสแซมเบอร์ ทั้งสิ้น นอกจากนี้การเขียนโปรแกรมบนเครื่องคอมพิวเตอร์ยังมีคำสั่งพิเศษต่างๆที่ช่วยในการเขียนโปรแกรมอีกด้วย

2.4.1 รูปแบบของภาษาแอสแซมบลี

โปรแกรมภาษาแอสแซมบลีประกอบด้วยส่วนต่างๆดังต่อไปนี้

1. Machine Instructions
2. Assembler Directives
3. Assembler Controls
4. Comments

Machine Instructions คือชุดคำสั่งภาษาเครื่องโดยแทนด้วยรหัส Mnemonics เช่น MOV , ANL เป็นต้น Assembler Directives หรือคำสั่งเทียมเป็นคำสั่งที่กำหนดขึ้นในภาษาแอสแซมบลีโดยจะไม่ถูกเปลี่ยนแปลงเป็นรหัสภาษาเครื่อง มีไว้สำหรับกำหนดโครงสร้างโปรแกรม , สัญลักษณ์ , ข้อมูลต่างๆ เช่น ORG , EQU เป็นต้น Assembler Controls เป็นชุดคำสั่งในการควบคุมของตัวแอสแซมเบอร์ Comments เป็นการเขียนคำอธิบายเข้าไปในโปรแกรม

ในการเขียนโปรแกรมในภาษาแอสแซมบลีในแต่ละบรรทัดมีส่วนประกอบที่สำคัญ 4 ส่วนคือ Label , Mnemonics , Operand , Comment โดยรูปแบบทั่วไปจะเป็นดังนี้

[Label :] Mnemonic [Operand] [, Operand] [....] [; Comment]

ในโปรแกรมมักจะเริ่ม Label ที่คอลัมน์ที่ 1

2.4.1.1 Label Field

ในภาษาแอสแซมบลี Label จะเป็นตัวแทนตำแหน่งของชุดคำสั่งหรือข้อมูลที่ตามหลัง Label อยู่ในคำสั่งกระโดดหรือคำสั่งต่างๆที่ต้องมีการอ้างตำแหน่ง สามารถแทนค่าตำแหน่งด้วย Label ได้เลย เช่น SJMP SKIP เป็นการกระโดดไปที่ตำแหน่งที่แทนด้วย Label SKIP

โดยทั่วไปแล้ว Label Field แทนได้ สองความหมาย คือ เป็น “Label” แทนค่าตำแหน่งและเป็น “Symbol” ซึ่งจะแทนค่าสัญลักษณ์ ถ้าหากจะใช้เป็น Label จะตามด้วยเครื่องหมาย Colon (:) แต่ถ้าจะเป็น Symbol จะตามด้วยคำสั่งเทียม เช่น EQU , SEGMENT , BIT , DATA เป็นต้น พิจารณาโปรแกรมต่อไปนี้จะใช้ PAR แทน Symbol และ START แทน Label

PAR EQU 500 ; "PAR" เป็นสัญลักษณ์แทนด้วยค่า 500

START : MOV , #0FFH ; "START" เป็นลาเบลจะมีความหมายแทนตำแหน่งที่เก็บคำสั่ง MOV

การตั้งชื่อให้ Label Field โดยมากจะเริ่มด้วยตัวอักษร สำหรับรายละเอียดต่างๆสามารถศึกษาได้จากคู่มือของตัวแอสแซมเบอร์โดยตรง

2.4.1.2 Mnemonic Field

ส่วน Mnemonic Field จะเป็นชุดคำสั่งหรือคำสั่งเทียมก็ได้ ซึ่งจะเขียนตามหลัง Label ตัวอย่างของชุดคำสั่งได้แก่ ADD , MOV , DIV , หรือ INC ตัวอย่างของคำสั่งเทียมได้แก่ ORG , EQU , หรือ DB

2.4.1.3 Operand Field

ในการเขียน Mnemonic เป็นตัวบอกว่าจะให้ Mnemonic ทำอะไร เช่น MOV จะเป็นตัวบอกว่าให้ย้ายข้อมูล แต่ยังไม่มีตำแหน่งที่จะย้ายว่าจะย้ายจากไหนไปไหน ส่วนที่จะบอกว่าจะให้ทำกับอะไรเรียกว่า Operand ซึ่งจะเขียนตามหลังรหัส Mnemonic ใน Operand Field นี้ประกอบด้วย Address หรือ Data ที่จะใช้กับชุดคำสั่ง ซึ่งอาจจะเป็น Label ที่แทนตำแหน่งของข้อมูลหรือ Symbol ที่แทนค่าข้อมูลด้วยก็ได้คำสั่งส่วนใหญ่จะมี Operand Field ตามหลังแต่บางคำสั่งจะไม่มี เช่น คำสั่ง RET

2.4.1.4 Comment Field

ส่วนนี้เป็นส่วนที่เขียนขึ้นในภาษาแอสแซมบลี เพื่อเป็นคำอธิบายส่วนต่างๆในโปรแกรม โดยการเขียนจะใช้เครื่องหมาย Semicolon (;) นำหน้าส่วน Comment นี้ตัวแอสแซมเบอร์จะไม่แปลเป็นภาษาเครื่อง

2.4.1.5 Special Assembler Symbols

ในภาษาแอสแซมบลีของ MCS-51 การอ้างตำแหน่งของรีจีสเตอร์ต่างๆสามารถเขียนเป็นตัวอักษรได้โดยตรง เช่น A , R0 ถึง R7 , DPTR , PC , C และ AB ส่วนสัญลักษณ์ $ (Dollar Sign) จะแทนตำแหน่งที่รีจีสเตอร์ PC ชี้อยู่ โดยมากมักจะใช้กับคำสั่งกระโดด ตัวอย่างเช่น

```
SETB  C      ; เซตบิต C
INC   DPTR   ; เพิ่มค่า DPTR
JNB   T1 , $ ; กระโดดไปตำแหน่งเดิมโดยการตรวจบิต T1
```

คำสั่งในบรรทัดสุดท้ายจะใช้เครื่องหมาย $ ซึ่งจะมีค่าเท่ากับคำสั่งต่อไปนี้

HERE : JNB T1 , HERE

2.4.1.6 Indirect Address

การกำหนดตำแหน่งของข้อมูลทางอ้อมจะใช้เครื่องหมาย "At" (@) หน้ารีจีสเตอร์ โดยรีจีสเตอร์ที่ใช้ได้แก่ R0 , R1 , DPTR , PC ตัวอย่างเช่น

ADD A , @R0

MOV A , @A+PC

2.4.1.7 Immediate Data

การกำหนดค่าโดยตรงในส่วนของ Operand Field ในภาษาแอสแซมบลีจะใช้เครื่องหมาย Pound Sign (#) พิจารณาตัวอย่างต่อไปนี้

```
CONSTANT    EQU    100
            MOV    A , #0FEH
            ORL    40H , #CONSTANT
```

การกำหนดค่าโดยตรงถ้าหากกำหนดค่าขนาด 16 บิตให้กับรีจีสเตอร์ขนาด 8 บิต ผลที่ได้จะนำค่า Low – byte ไปเก็บในรีจีสเตอร์ โดย High – byte จะกำหนดได้เพียง 2 ค่าเท่านั้นคือ 00H และ FFH ถ้าเป็นค่าอื่นจะเกิดข้อผิดพลาดขึ้น เช่น

MOV A , #0FF00H

MOV A , #00FFH

แต่ถ้าเขียนในลักษณะสองบรรทัดต่อไปนี้ จะเกิดข้อผิดพลาดขึ้น เนื่องจากใน High – byte กำหนดค่าผิด

MOV A , #0FE00H

MOV A , #01FFH

ถ้าหากใช้สัญลักษณ์เป็นเลขฐานสิบ ซึ่งมักมีค่าระหว่าง –256 ถึง +256 ถ้าหากเขียนในลักษณะสองบรรทัดต่อไปนี้ค่าที่ได้จะมีผลเท่ากัน

MOV A , #-256

MOV A , #0FF00H

2.4.1.8 Data Address

คำสั่งในการเข้าถึงข้อมูลในหน่วยความจำ จะใช้วิธีเข้าถึงโดยตรงได้กับหน่วยความจำภายในชิพตำแหน่ง 00H ถึง 7FH สำหรับรีจีสเตอร์ฟังก์ชันพิเศษจะอยู่ในตำแหน่ง 80H ถึง 0FFH ในการเขียนโปรแกรมสามารถอ้างตำแหน่งโดยใช้ชื่อรีจีสเตอร์โดยตรงได้

```
MOV    A , 45H
MOV    A , SBUF        ; มีผลเท่ากับการอ่านจากตำแหน่ง 99H
```

2.4.1.9 Bit Address

การเข้าถึงข้อมูลระดับบิตใน MCS-51 จะอยู่ในไบต์ที่ 20H ถึง 2FH โดยตำแหน่งของบิต คือ ตำแหน่งของบิตที่ 00H ถึง 7FH และในรีจีสเตอร์ในฟังก์ชันพิเศษบางตัวก็สามารถจะเข้าถึงข้อมูลระดับบิตได้ ในการเขียนโปรแกรมภาษาแอสแซมบลี ถ้าเป็นคำสั่งที่กระทำกับบิตสามารถใส่ตำแหน่งของบิตได้ทันที และถ้าเป็นการอ้างตำแหน่งบิตในรีจีสเตอร์แบบต่างๆสามารถใช้เครื่องหมายจุดระหว่างรีจีสเตอร์กับตำแหน่งของบิตหรือใช้ชื่อบิตของรีจีสเตอร์นั้นเลย ตัวอย่างเช่น

```
SETB   0E7H   ; เซตบิตตำแหน่งที่ 0E7H
SETB   ACC.7  ; เซตบิตที่ 7 ในรีจีสเตอร์ A
JNB    TI , $ ; ใช้เครื่องหมาย TI แทนตำแหน่งบิต
JNB    99H , $
```

2.4.1.10 Code Address

เราสามารถใช้รหัสตำแหน่ง (Code Address) ซึ่งสร้างจาก Label แทนในคำสั่ง JUMP และคำสั่ง CALLS ต่างๆได้

2.4.1.11 Generic Jumps and Calls

การเขียนโปรแกรมภาษาแอสแซมบลีแล้วใช้โปรแกรมแอสแซมเบอร์แปลงเป็นรหัสคำสั่งนั้นมีข้อดีอีกอย่างหนึ่ง คือการใช้คำสั่งกระโดดและคำสั่งเรียกโปรแกรมย่อยสามารถใช้ได้โดยง่ายโดยรูปแบบคำสั่งที่ใช้คือ JMP และ CALL โดย "JMP" สามารถแทนคำสั่ง SJMP , AJMP หรือ LJMP ได้ สำหรับคำสั่ง "CALL" สามารถใช้แทน ACALL หรือ LCALL ได้ โดยตัวแอสแซมเบอร์จะแปลงคำสั่ง JMP และ CALL ให้เป็นคำสั่งที่เหมาะสมเอง โดยพิจารณาจากระยะทางที่ PC กระโดดไป

พิจารณาโปรแกรมดังรูปที่ 2.24 จากโปรแกรมจะเริ่มที่ตำแหน่ง 1234H คำสั่งกระโดดคำสั่งแรกจะเริ่มที่บรรทัดที่ 3 ซึ่งระยะทางที่จะกระโดดไปจะเห็นว่าอยู่ระหว่าง –128 ถึง +127 ดังนั้นตัวแอสแซมเบอร์จะแปลง JMP เป็น SJMP ซึ่งดูได้จากรหัส OBJ (รหัส 80 เป็นรหัสของ SJMP) ในบรรทัดที่ 4 จะกำหนด ORG เพิ่มอีก 200 ตำแหน่ง ห่างจากตำแหน่งที่อ้างด้วย Label START ในบรรทัดที่ 5 จะมีคำสั่ง JMP ซึ่งตำแหน่งที่จะกระโดดไปมีค่าเกิน –128 ถึง +127 แต่ไม่เกิน 2 k ตำแหน่ง ตัวแอสแซมเบอร์จะแปลงให้เป็น AJMP

LOC	OBJ	LINE	SOURCE		
1234		1		ORG	1234
1234	04	2	START :	INC	A
1235	80FD	3		JMP	START
12FC		4		ORG	START + 200
12FC	4134	5		JMP	START
12FE	021301	6		JMP	FINISH
1301	04	7	FINISH :	INC	A
		8		END	

รูปที่ 2.24 ตัวอย่างโปรแกรมภาษาแอสแซมบลี

2.4.2 Assemble – time Expression Evalution

ในที่นี้จะกล่าวถึงการกำหนดค่าคงที่และตัวแปรต่างๆ

2.4.2.1 Number Bases

การกำหนดค่าตัวเลขหรือค่าคงที่ต่างๆให้กับตัวแปรในการเขียนภาษาแอสแซมบลีนั้นถ้าเป็นเลขไบนารีจะลงท้ายด้วยตัว "B" ถ้าเป็นเลขฐานสิบหกจะใส่ตัวเลข 0 นำหน้าและลงท้ายด้วยตัว "H" ถ้าเป็นเลขฐานสิบจะลงท้ายด้วย "D" ถ้าเป็นเลขฐานแปดจะลงท้ายด้วย O ตัวอย่างเช่น

```
MOV   A , #1111B
MOV   A , #0FH
MOV   A , #15D
```

2.4.2.2 Character Strings

การกำหนดตัวอักษรจะใช้ 1 หรือ 2 ตัว โดยการกำหนดจะใช้เครื่องหมาย (') ปิดตัวอักษร ตัวแปรภาษาแอสแซมเบอร์จะแปลงตัวอักษรเป็นรหัส ASCII ดังตัวอย่างต่อไปนี้

```
CJNE   A , # 'Q' , AGAIN
SUBB   A , # 'Q'
MOV    DPTR , # 'AB'
MOV    DPTR , #4124H
```

2.4.2.3 Arithmetic Operators

ตัวดำเนินการทางคณิตศาสตร์ที่ใช้ในภาษาแอสแซมบลีมีดังนี้

ตารางที่ 2.5 ตัวดำเนินการทางคณิตศาสตร์

ตัวดำเนินการ	ความหมาย
+	บวก
-	ลบ
*	คูณ
/	หาร
MOD	หารแบบ module (เอาเศษที่ได้จากการหาร)

การเขียนโปรแกรมทางคณิตศาสตร์ของสองบรรทัดต่อไปนี้ให้ผลเหมือนกัน โดยบรรทัดแรกจะบวกค่า 10 ในฐานสิบหกมาเก็บไว้ในรีจิสเตอร์ A ซึ่งมีค่าเท่ากับการเก็บค่า 1AH ลงในรีจิสเตอร์ A โดยตรง

```
MOV   A , #10+10H
MOV   A , #1AH
```

การเขียนโปรแกรมดังสองบรรทัดต่อไปนี้ให้ผลเช่นเดียวกันโดยบรรทัดแรกจะเอาเศษที่ได้จากการหารคือ 4 ใส่ในรีจิสเตอร์ A

```
MOV   A , #25 MOD 7
MOV   A , #4
```

2.4.2.4 Logical Operators

ตัวดำเนินการทางลอจิกมีดังนี้

ตารางที่ 2.6 ตัวดำเนินการทางลอจิก

ตัวดำเนินการ	ความหมาย
OR	การกระทำ OR
AND	การกระทำ AND
XOR	การกระทำ Exclusive OR
NOT	การกระทำ Complement

การเขียนโปรแกรมเราสามารถใส่ตัวดำเนินการเข้าไปในคำสั่งได้เลยดังตัวอย่างต่อไปนี้

```
MOV    A , '#9' AND 0FH
```

ตัวอย่างต่อไปนี้จะแสดงให้เห็นว่าตัวดำเนินการ NOT สามารถใส่เข้าไปในตัว Operand ได้ทันที ซึ่งคำสั่ง MOV ทั้งสามคำสั่งผลที่ได้จะเหมือนกัน

```
        THREE       EUQ   3
MINUS_THREE         EQU   -3
                    MOV   A , #(NOT THREE)+1
                    MOV   A , #MINUS_THREE
                    MOV   A , #11111101B
```

2.4.2.5 Special Operators

ตัวดำเนินการพิเศษมีดังนี้

ตัวดำเนินการ	ความหมาย
SHR	เลื่อนไปทางขวา
SHL	เลื่อนไปทางซ้าย
HIGH	ค่าไบต์สูง
LOW	ค่าไบต์ต่ำ
()	ให้ทำค่านี้ก่อน

ตารางที่ 2.7 ตัวดำเนินการพิเศษ

ตัวอย่างเช่น

```
        MOV    A , #8 SHL 1
        MOV    A , #10H
```

จากตัวอย่างจะเป็นคำสั่งเลื่อนค่า 8 บิตไปหนึ่งครั้งแล้วเก็บไว้ในรีจีสเตอร์ A ค่า 8 มีเลขไบนารีเป็น 00001000 เมื่อถูกเลื่อนไปทางซ้ายหนึ่งครั้งจะมีค่าเป็น 00010000 ซึ่งผลลัพธ์จะเท่ากับ 10H ซึ่งจะทำให้คำสั่งทั้งสองมีผลเท่ากัน ตัวอย่างเช่น

```
MOV    A , #HIGH 1234H
MOV    A , #12H
```

จากตัวอย่างคำสั่งแรกจะเป็นการนำค่าไบต์สูงของ 1234H มาใส่ในรีจีสเตอร์ A ซึ่งค่าไบต์สูงคือ 12 H ทำให้คำสั่งทั้งสองมีผลเท่ากัน

2.4.2.6 Relational Operators

ตัวดำเนินการแสดงความสัมพันธ์จะใช้กับ Operand สองตัวมาเปรียบเทียบกัน ซึ่งมีดังนี้

ตัวดำเนินการ		ความหมาย
=	EQ	เท่ากับ
<>	N	ไม่เท่ากับ
<	LT	น้อยกว่า
<=	LE	น้อยกว่าหรือเท่ากับ
>	GT	มากกว่า
>=	GE	มากกว่าหรือเท่ากับ

ตารางที่ 2.8 ตัวดำเนินการเปรียบเทียบ

การใช้ตัวดำเนินการให้เลือกใช้อย่างใดอย่างหนึ่ง ("=" หรือ "EQ") โดยผลลัพธ์ที่ได้จะเป็นจริงหรือเท็จถ้าเป็นจริงค่าในรีจีสเตอร์ A จะเป็น FFH ถ้าเป็นเท็จจะเป็น 00H พิจารณาตัวอย่างต่อไปนี้

```
MOV   A , #5=5
MOV   A , 5 NE 4
MOV   A , # 'X' LT 'Z'
MOV   A , # 'X' >= 'X'
MOV   A , #$ > 0
MOV   A , #100 GE 50
```

ผลลัพธ์ที่ได้ทุกคำสั่งจะมีค่าเท่ากับ

```
MOV   A , #0FFH
```

ตัวอย่างตัวดำเนินการต่างๆและผลลัพธ์ที่ได้จะเป็นดังนี้

การกระทำ	ผลที่ได้
'B' - 'A'	0001H
8/3	0002H
155 MOD 2	0001H
4 * 4	0010H
8 AND 7	0000H
NOT 1	FFFEH

การกระทำ	ผลที่ได้
'A' SHL 8	4100H
LOW 65535	00FFH
(8+1) * 2	0012H
5 EQ 4	0000H
'A' LT 'B'	FFFFH
3 <= 3	FFFFH

จากตัวอย่างในเรื่อง Timer ที่เก็บค่า –500 ลงในรีจีสเตอร์ TH1 และ TL1 จะเห็นว่ามีการใช้คำสั่ง HIGH และ LOW ดังนี้

```
COUNT    EQU   -500
         MOV   TH1 , #HIGH VALUE
         MOV   TL1 , #LOW VALUE
```

ตัวแอสแซมเบอร์จะแปลงค่า –500 เป็นเลขฐานสิบหก คือ FE0CH โดยค่า High คือ FEH และค่า Low คือ 0CH

ถ้าหากมีการใช้ตัวดำเนินการหลายๆตัวในนิพจน์เดียวกัน จะมีลำดับการทำงานก่อนหลังดังนี้

ลำดับ	ตัวดำเนินการ
1	( )
2	HIGH LOW
3	* / MOD SHL SHR
4	+ -
5	EQ NE LT LE GT GE
6	= <> < <= > >=
7	NOT
8	OR XOR

ตารางที่ 2.9 การใช้ตัวดำเนินการเปรียบเทียบ

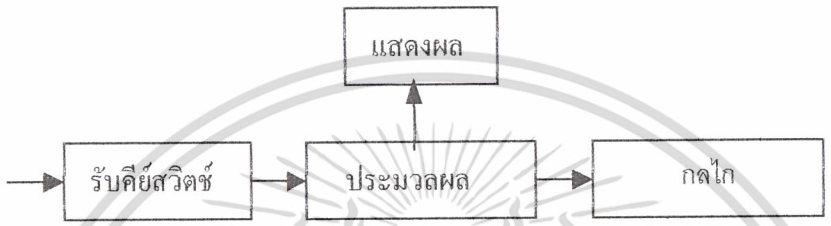
ถ้าหากมีการใช้ตัวดำเนินการหลายตัวในบรรทัดเดียวกัน ผลลัพธ์ที่ได้จะแสดงได้ดังตัวอย่างต่อไปนี้

Expression	ผลลัพธ์ที่ได้
HIGH ('A" SHL 8)	0041H
HIGH 'A' SHL 8	0000H
NOT 'A' - 1	FFBFH
'A' OR 'A' SHL 8	4141H

บทที่ 3
ทฤษฎีการทำงานของโครงงาน

3.1 บล็อกไดอะแกรม (BLOCK DIAGRAM)

บล็อกไดอะแกรมของโครงงานประกอบด้วยส่วนต่างๆดังนี้



รูปที่ 3.1 บล็อกไดอะแกรม

เมื่อมีการคีย์สวิตช์จะนำค่าที่ได้ทำการประมวลผล เมื่อประมวลผลแล้วก็จะนำค่าที่ได้ไปทำตามคำสั่งตามที่ผู้ใช้งานต้องการ ซึ่งจะมีการแสดงผลทางไดโอดเปล่งแสง 7 ส่วน เมื่อผู้ใช้งานกดคีย์สวิตช์ต้องการที่จะเปิดประตูจะนำค่าในการคีย์ไปทำการประมวลซึ่งจะตรวจสอบรหัสว่าถูกต้องหรือไม่ถ้าถูกต้องก็จะไปสั่งให้ส่วนของกลไกทำการคลายล็อกประตูทำให้สามารถเปิดประตูได้ ถ้าตรวจสอบว่ารหัสไม่ถูกต้องก็จะให้ผู้ใช้งานใส่รหัสจนกว่าจะถูกต้อง

โครงงานประกอบด้วยส่วนสำคัญ 2 ส่วนใหญ่ๆ คือ

- โครงสร้างทางฮาร์ดแวร์
- การออกแบบซอร์ฟแวร์

3.2 โครงสร้างทางฮาร์ดแวร์ ประกอบด้วย

3.2.1 แหล่งจ่ายไฟ (Power Supply)

3.2.2 ส่วนคีย์สวิตช์รับข้อมูล (Keypad)

3.2.3 ส่วนแสดงผล (Display)

3.2.4 ไมโครคอนโทรลเลอร์ (MCS-51)

3.2.5 กลไก

3.2.1 แหล่งจ่ายไฟ

เนื่องจากอุปกรณ์อิเล็กทรอนิกส์แต่ละตัวมีความต้องการใช้ไฟไม่เท่ากัน ดังนั้นในการออกแบบแหล่งจ่ายไฟตรงจะต้องพิจารณาในเรื่องต่อไปนี้

1. ต้องทราบว่าอุปกรณ์ตัวนั้นต้องการแรงดันไฟตรงขนาดเท่าใดหรือดูที่ความต้องการไฟของโหลด
2. แหล่งจ่ายไฟที่ดีต้องสามารถปรับค่าได้เมื่อโหลดต้องการกระแสสูงสุด
3. โวลท์เตจเร็กกูเรชัน (Voltage Regulation) ของแหล่งจ่ายไฟ เมื่อมีการเปลี่ยนแปลงแรงดันไฟตรงทางด้านเอาต์พุต เทียบกับการเปลี่ยนแปลงกระแสของโหลดจากค่ากระแสต่ำสุด (no load) ถึงค่ากระแสสูงสุด (full load) สามารถคำนวณในรูปของเปอร์เซ็นได้ดังนี้

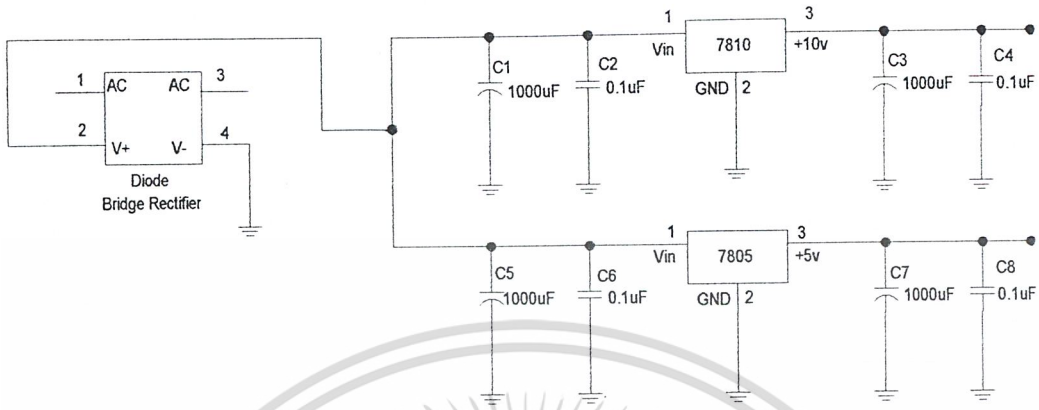
$$\% \text{ Regulation} = \frac{V_{NL} - V_{FL}}{V_{FL}} \times 100$$

4. การเปลี่ยนแปลงแรงดันเอาต์พุตอย่างรวดเร็ว เนื่องจากการกรองกระแสที่ไม่สมบูรณ์ การเปลี่ยนแปลงแรงดันนี้มีความถี่มูลฐานสัมพันธ์กับความถี่ของไฟกระแสสลับ ซึ่งเรียกว่าแรงดันกระเพื่อม (Ripple Voltage) หรือเรียกง่ายๆว่า ริปเปิล (Ripple) สามารถคำนวณหาได้จาก อัตราส่วนระหว่างค่า rms ของแรงดันกระเพื่อมต่อแรงดันไฟตรงขณะฟูลโหลด (full load dc-voltage) สามารถคำนวณในรูปของเปอร์เซ็น

$$\% \text{ Ripple} = \frac{V_{rms}\,(\text{ripple})}{V_{FL}\,(\text{Vdc})}$$

จากคุณลักษณะต่างๆของแหล่งจ่ายไฟตรง เรายังต้องพิจารณาถึงคุณสมบัติข้ออื่นๆด้วยขึ้นอยู่กับความต้องการใช้งานของผู้ใช้

สำหรับโครงงานนี้เราใช้แหล่งจ่ายไฟ 5 โวลท์และ 10โวลท์ วงจรที่ใช้แหล่งจ่ายไฟ 5โวลท์ ได้แก่ ส่วนไมโครคอนโทรลเลอร์ , ไดโอดเปล่งแสง 7 ส่วน , คีย์สวิตช์ ส่วนวงจรที่ใช้แหล่งจ่ายไฟ 10โวลท์ ได้แก่ ส่วนที่เป็นกลไก วงจรที่ใช้ในโครงงานนี้แสดงได้ดังรูปที่ 3.2



รูปที่ 3.2 วงจรจ่ายไฟ 5โวลท์และ 10โวลท์

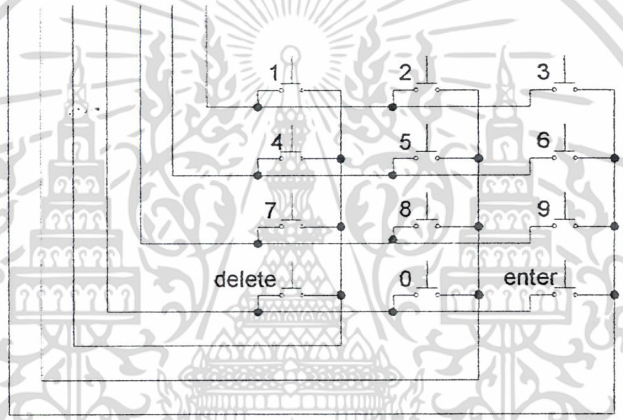
วงจรจะเป็นการเรียงกระแสเต็มคลื่นแบบบริดจ์ (Full-Wave Bridge Rectifier) ซึ่งวงจรใช้ไอซีบริดจ์ และจะมีตัวเก็บประจุ (capacitor) เป็นตัวกรองกระแสให้เรียบขึ้น และจะมีไอซีเรกกูเรเตอร์ (IC Regulator) เบอร์ 7805 ซึ่งจะให้แรงดันทางเอาต์พุตมีค่า 5 โวลท์ และไอซีเรกกูเรเตอร์ 7810 ซึ่งจะให้แรงดันทางเอาต์พุตมีค่า 10โวลท์

3.2.2 ส่วนคีย์สวิตช์รับข้อมูล

โครงงานจะใช้สวิตช์แบบเมทริกซ์ หรือที่เรียกว่าคีย์แพด ใช้พอร์ทของ AT89S8252 จำนวนหนึ่งพอร์ทในที่นี้จะใช้คีย์แพดขนาด 4 × 3 ต่อเข้ากับ AT89S8252 ที่พอร์ท P1 การต่อสวิตช์ลักษณะนี้จะต้องโปรแกรมสแกนคีย์สวิตช์เพื่อที่จะเช็คว่าพอร์ทใดถูกกดซึ่งทำได้โดยส่งค่า “0” ออกไปที่พอร์ทซึ่งจะต้องเริ่มส่งจากหลักที่ 1 ก่อน (ซึ่งหลักที่ 1 ต่ออยู่ที่ P1.0) แล้วจึงทำการตรวจสอบค่าที่พอร์ทว่ามีการกดคีย์หรือไม่ซึ่งถ้ามีการกดคีย์ที่พอร์ทนั้นจะมีค่าเป็น “1” ถ้าพบว่ามีค่าเป็น “1” ซึ่งก็คือมีการคีย์สวิตช์ แล้วจะนำค่าที่ได้ไปเก็บไว้ในบัฟเฟอร์ซึ่งค่าประจำหลักที่ 1 นี้มีค่าเป็น 1 ตามตารางที่ 3.1 เมื่อนำค่าไปเก็บไว้แล้วก็จะทำการ ตรวจสอบที่แถว ว่าค่าที่อยู่ในหลักที่ 1 ที่ตรวจสอบได้ค่าเป็น “1” นั้นมีการคีย์สวิตช์ที่ตำแหน่งแถวใด เมื่อ ตรวจสอบเจอแล้วก็จะนำค่าที่ได้ประจำแถวนั้นซึ่งดูได้จากตารางที่ 3.1 ไปบวกกับค่าที่เก็บไว้ใน บัฟเฟอร์ ก่อนนั้นซึ่งก็จะเป็นรหัสที่เราคีย์ลงไป ซึ่งการตรวจสอบคีย์จะเริ่มที่หลักก่อนถ้าเจอแล้วจึงตรวจสอบที่แถว ตารางที่ 3.1 เป็นการแสดงคีย์แพดที่ใช้ในโครงงานพร้อมค่าประจำแถวและหลักที่นำไปใช้ในการคิดออกแบบโปรแกรมในการตรวจสอบคีย์สวิตช์รหัสว่าที่คีย์ลงไปเป็นค่าอะไร

	หลักที่1 ค่าประจำหลัก 1	หลักที่2 ค่าประจำหลัก 2	หลักที่3 ค่าประจำหลัก 3
แถวที่1 ค่าประจำแถว 0	**1**	**2**	**3**
แถวที่2 ค่าประจำแถว 3	**4**	**5**	**6**
แถวที่3 ค่าประจำแถว 6	**7**	**8**	**9**
แถวที่4 ค่าประจำแถว 9	#	**0**	*

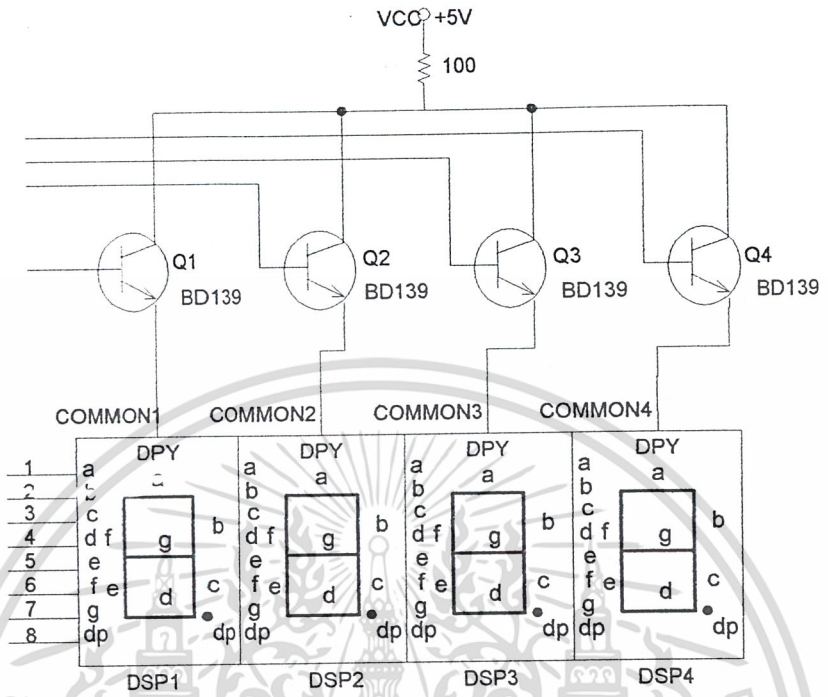
ตารางที่ 3.1 คีย์แพดขนาด 4 × 3 พร้อมค่าประจำหลักและแถว



รูปที่ 3.3 แสดงการต่อพอร์ทแบบเมทริกซ์เข้ากับพอร์ท AT89S8252

3.2.3 ส่วนแสดงผล

ในการแสดงผลตัวเลขหลายๆหลัก เราต้องใช้ไดโอดเปล่งแสง 7 ส่วนหลายๆตัว ซึ่งจะเห็นว่าไดโอดเปล่งแสง 7 ส่วน ตัวหนึ่งต้องใช้พอร์ทขนาด 8 บิตหนึ่งพอร์ท ดังนั้นเมื่อต้องการแสดงตัวเลขหลายหลักจึงต้องใช้หลายๆพอร์ท ซึ่งในโครงงานต้องใช้ไดโอดเปล่งแสง 7 ส่วน ในการแสดงผลถึง 4 หลัก ซึ่งเมื่อคิดดูแล้วต้องใช้ถึง 4 พอร์ท ซึ่งจะเป็นการสิ้นเปลืองอย่างมาก ในการแก้ไขปัญหานี้ทำได้โดยใช้วิธีต่อแบบ มัลติเพล็กดีสเพลย์ (Multiplexed Displays) ซึ่งจะทำให้ประหยัดพอร์ทได้อย่างมาก วิธีนี้ทำได้โดยการต่อขาแต่ละเซกเมนต์เข้าด้วยกัน จากนั้นจะใช้วิธีสแกนให้ไดโอดเปล่งแสง 7 ส่วนติดทีละหลักโดยการสแกนแต่ละหลักจะต้องทำอย่างเร็ว จึงทำให้เราไม่สามารถมองการดับของไดโอดเปล่งแสง 7 ส่วน ได้ทันจึงดูเหมือนว่าไดโอดเปล่งแสง 7 ส่วนทุกตัวติดพร้อมกัน

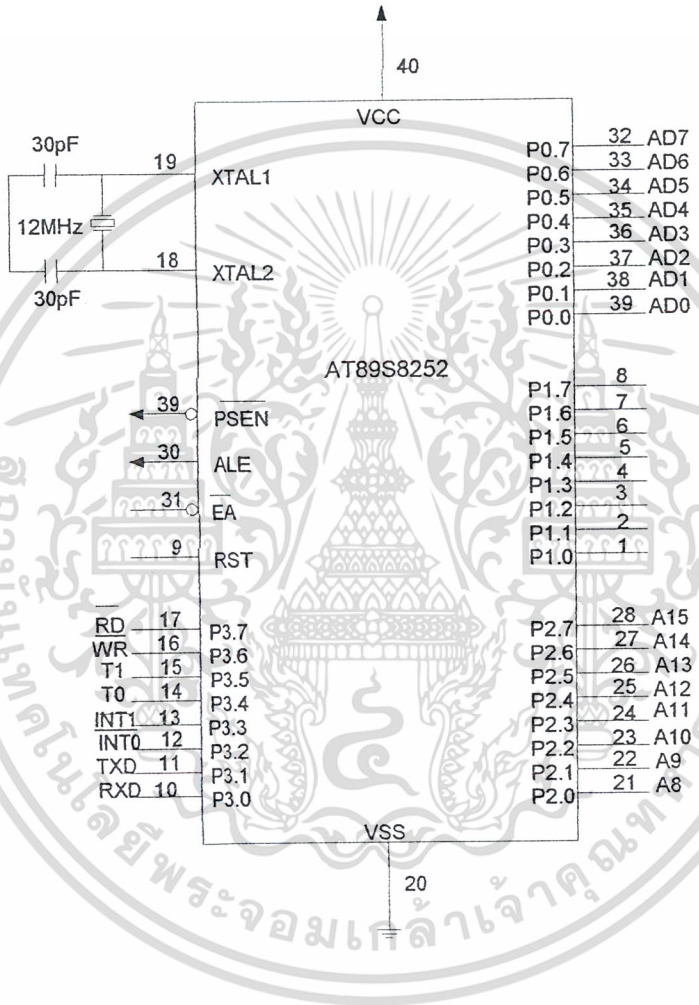


รูปที่ 3.4 แสดงวงจรของส่วนแสดงผล

ข้อมูลที่ส่งให้ไดโอดเปล่งแสง 7 ส่วนแต่ละหลักจะใช้พอร์ท 0 คือ P0.0-P0.7 ของ AT89S8252 ซึ่งการต่อวงจรแสดงได้ดังรูปที่ 3.4 คือ P0.0 ต่อกับขา a ของไดโอดเปล่งแสง 7 ส่วน ซึ่งใช้ไดโอดเปล่งแสง 7 ส่วนแบบ อาโนดร่วม ส่วน P0.1 ก็จะต่อกับขา b ของไดโอดเปล่งแสง 7 ส่วน และที่บิตอื่นๆดูได้จากรูปที่ 3.4 ในส่วนของทรานซิสเตอร์ที่ต่ออยู่กับขา คอมมอนของไดโอดเปล่งแสง 7 ส่วน เพื่อที่จะช่วยในการขับกระแส ลงกราวจ์ โดยขาเบสของ ทรานซิสเตอร์จะต่ออยู่ที่พอร์ท 1 ของ AT89S8252 เมื่อ AT89S8252 ส่งข้อมูลให้พอร์ท 0 ไดโอดเปล่งแสง 7 ส่วนทุกตัวพร้อมที่จะแสดงผลขึ้นอยู่กับว่าพอร์ท 1 ของ AT89S8252 นั้นบิตใดเป็น “1” ซึ่งหลักนั้นก็จะแสดงผล เช่น ถ้าต้องการแสดงตัวเลข 1 3 5 7 ก็ทำได้โดยขั้นแรกส่งข้อมูลที่แสดงเลข 1 ออกไปจากนั้นทำให้ P1.0 เป็น “1” ต่อมาส่งข้อมูลที่ทำให้แสดงตัวเลข 3 ออกไป จากนั้นทำให้ P1.1 เป็น “1” ทำอย่างนี้ไปเรื่อยจนครบทุกตัวเลขที่ต้องการให้แสดงผล และก็มีการเขียนโปรแกรมวนลูปหน่วงเวลาให้เหมาะสมตัวเลขที่แสดงที่ไดโอดเปล่งแสง 7 ส่วนก็จะเป็น 1 3 5 7 ดังที่ได้ป้อนข้อมูลจากคีย์แพด

3.2.4 ไมโครคอนโทรลเลอร์

ไอซีไมโครคอนโทรลเลอร์โครงสร้างไอซีเป็นแบบ DIP มีขาทั้งหมด 40 ขาโดยขาต่างๆจะใช้เป็นขาพอร์ทอินพุต , เอาต์พุต , ขาสัญญาณควบคุม , ขาตำแหน่งหน่วยความจำ และขาข้อมูลดัง รูปที่ 3.5



รูปที่ 3.5 แสดงขาต่างๆของ AT89S8252

ความหมายของขาต่างๆมีดังนี้

3.2.4.1 พอร์ท 0 (port 0)

พอร์ท 0 ได้แก่ขาที่ 32-39 ของ AT89S8252 สามารถใช้เป็นอินพุตเอาต์พุตได้นอกจากนี้ในการติดต่อกับหน่วยความจำภายนอกยังใช้เป็น ขาแอดเดรสบัส (Addess Bus) และ ดาต้าบัส (Data Bus) อีกด้วย

3.2.4.2 พอร์ท 1 (port 1)

พอร์ท 1 ได้แก่ขาที่ 1-8 เป็นพอร์ท 8 บิต สามารถอ้างทีละบิตได้ คือ p1.0,p1.1,.....etc

3.2.4.3 พอร์ท 2 (port 2)

พอร์ท 2 ได้แก่ขาที่ 21-28 จะใช้งาน 2 หน้าที่ คือใช้เป็นพอร์ท 8 บิตกับใช้เป็นขาแอดเดรส 8 บิต ในการอ้างหน่วยความจำภายนอก

3.2.4.4 พอร์ท 3 (port 3)

พอร์ท 3 ได้แก่ขาที่ 10-17 จะใช้งานสองหน้าที่คือ เป็นพอร์ทอินพุตและพอร์ทเอาต์พุต และใช้เป็นขาควบคุมต่างๆดังตารางที่ 3.2

บิต	ชื่อ	หน้าที่พิเศษ
P3.0	RXD	ใช้รับข้อมูลทางพอร์ทอนุกรม
P3.1	TXD	ใช้ส่งข้อมูลทางพอร์ทอนุกรม
$\overline{\text{P3.2}}$	INT0	อินเทอร์รัพท์ภายนอกหมายเลข 0
$\overline{\text{P3.3}}$	INT1	อินเทอร์รัพท์ภายนอกหมายเลข 1
P3.4	T0	ตัวจับเวลา / ตัวนับ ตัวที่ 0
P3.5	T1	ตัวจับเวลา / ตัวนับ ตัวที่ 1
$\overline{\text{P3.6}}$	WR	สัญญาณเขียนข้อมูลหน่วยความจำภายนอก
$\overline{\text{P3.7}}$	RD	สัญญาณอ่านข้อมูลหน่วยความจำภายนอก

ตารางที่ 3.2 แสดงบิตและหน้าที่ต่างๆของพอร์ท 3

3.2.4.5 $\overline{\text{PSEN}}$ (Program Store Enable)

ขา $\overline{\text{PSEN}}$ เป็นขาที่ส่งสัญญาณออกคือขา 29 ขานี้จะแอคทีฟเมื่อ AT89S8252 ต้องการอ่านโคด โปรแกรมภายนอก โดยปกติถ้าหน่วยความจำภายนอกเป็น EPROM ขา $\overline{\text{PSEN}}$ จะต่อกับขาเอาต์พุต อีเนเบิล (Output Enable) ($\overline{\text{OE}}$) ของ EPROM

3.2.4.6 ALE (Addess Latch Enable)

เนื่องจากพอร์ท 0 สามารถใช้เป็นขาอ้างตำแหน่ง และขาข้อมูล AT89S8252 จะมีขา ALE ได้แก่ขา 30 ขานี้จะใช้ มัลติเพล็ก สัญญาณแอดเดรสบัส ของพอร์ท 0 ในการใช้งานระบบ AT89S8252 นั้นจะต้องมีอุปกรณ์มาต่อกับพอร์ท 0 ที่ทำหน้าที่ แลท (Latch) สัญญาณแอดเดรสบัสเมื่อ AT89S8252 ต้องการติดต่อกับหน่วยความจำ ภายนอก AT89S8252 จะส่งสัญญาณแอดเดรสบัสออกมาก่อนทางพอร์ท 0 จากนั้นจะส่งสัญญาณ ALE มา แลท อุปกรณ์ภายนอก ให้เก็บค่าแอดเดรสบัสของพอร์ท 0 ไว้เพื่อใช้พอร์ท 0 เป็น ดาต้าบัส ต่อไป

3.2.4.7 $\overline{EA}$ (External Access)

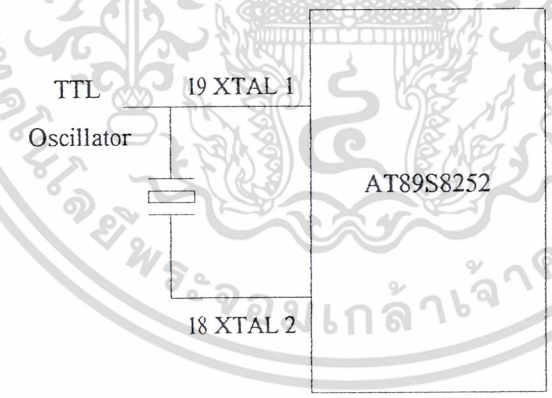
ขา $\overline{EA}$ ได้แก่ขาที่ 31 ถ้าขานี้เป็นลอจิก " 1 " จะใช้กับเบอร์ 8051 / 8052 เพื่อบอกว่าให้อ่านโปรแกรมจากหน่วยความจำโปรแกรมภายใน แต่ถ้าเป็นลอจิก " 0 " จะบอกว่าให้ AT89S8252 ทำโปรแกรมโดย อ่านจากหน่วยความจำภายนอก (ถ้าขา $\overline{EA}$ เป็น " 0 " ขา $\overline{PSEN}$ จะแอคทีฟ) ถ้าหากเป็นเบอร์ 8031 หรือ 8032 ขา $\overline{EA}$ จะเป็น "0" เสมอ เพราะว่าไม่มีโปรแกรมหน่วยความจำภายใน แต่ถ้าใช้เบอร์ 8051 / 8052 ซึ่งมีหน่วยความจำโปรแกรมภายในและให้ขา $\overline{EA}$ เป็น "0" ซึ่งจะดีสเอเบิลรอม (Disabled ROM) ภายในและจะอ่านโปรแกรมจาก EPROM ภายนอกแทน

3.2.4.8 RST (Reset)

ขา RST ได้แก่ขา 9 จะใช้ในการรีเซต AT89S8252 โดยจะให้ขานี้เป็นลอจิก "1" อย่างน้อย 2 แมชชีนไซเคิล (Machine Cycles) จึงจะรีเซตระบบได้

3.2.4.9 ความถี่สัญญาณนาฬิกาบนชิพ (On-chip Oscillator Inputs)

เป็นวงจรออสซิลเลเตอร์ (Oscillator) บนชิพ ได้แก่ขา 18-19 โดยต่อ คริสตอล (Crytal) เข้ากับขานี้ โดยปกติมักจะใช้ คริสตอล ความถี่ 12 MHz กับตัวเก็บประจุหรืออาจใช้สัญญาณนาฬิกาจาก TTL Clock Source ต่อกับ XTAL1 และ XTAL2 ดังรูปที่ 3.6



รูปที่ 3.6 ขาของ AT89S8252 ที่ใช้ต่อกับ XTAL

ใน AT89S8252 จะใช้แหล่งจ่ายไฟ 5v ต่อเข้ากับขา vcc (40) ส่วนขา vss (20) จะต่อลง กราวจ์

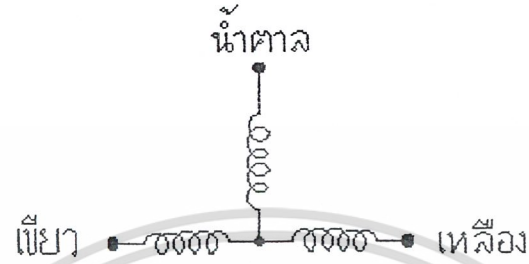
3.2.5 กลไก

กลไกเป็นส่วนประกอบที่สำคัญอีกอย่างหนึ่งของโครงงานเพราะจะใช้เป็นตัวล็อคประตู โดยมีความคิดอยู่ว่าทำยังไงถึงจะล็อคตัวเดือยของลูกบิดได้ จึงคิดว่าต้องทำอะไรมาขัดตัวเดือยของลูกบิดไว้ซึ่งตัวกลไกก็ต้องสามารถหมุนได้กรณีเราต้องเปิดประตู และเราจะควบคุมตัวกลไกกรณีเปิดและปิดอย่างไร ซึ่งเมื่อจะเปิดประตูกลไกต้องหมุนได้ส่วนจะปิดประตูกลไกต้องไม่หมุนจึงได้นำตัวโซลินอยมาเป็นตัวควบคุม ดังแสดงได้ดังรูปที่ 3.7

รูปที่ 3.7 แสดงส่วนของกลไก

โซลินอย จะทำหน้าที่โดยเดือยจะเลื่อนออกจากตัวโซลินอยเองกรณีสั่งล็อคประตูเมื่อได้รับคำสั่งจากไมโครคอนโทรลเลอร์และจะเลื่อนเข้าไปขัดยังรูข้างหลังตัวกลไกซึ่งปกติกลไกจะหมุนไปมาได้เมื่อเดือยของโซลินอยเข้าไปขัดตัวกลไกก็ไม่สามารถหมุนได้ทำให้เดือยของลูกบิดไปขัดกับเดือยของกลไกจึงทำให้ไม่สามารถเปิดประตูได้ และในทางตรงกันข้ามกรณีเราต้องการเปิดประตูสามารถทำได้โดยเราต้องใส่รหัสที่ถูกต้องตรงตามรหัสที่เราโปรแกรมไมโครคอนโทรลเลอร์ก็จะไปควบคุมให้โซลินอยเลื่อนเดือยที่ขัดกับกลไกออกทำให้กลไกหมุนไปมาได้ดังนั้นเดือยของลูกบิด

จึงสามารถออกจากเดือยของกลไกได้ทำให้สามารถเปิดประตูได้ สปริงจะเป็นตัวดึงให้กลไกหมุนมายังตำแหน่งที่เดือยของโซลินอยตรงกับรูข้างหลังของตัวกลไกพอดี



รูปที่ 3.8 แสดงโครงสร้างของโซลินอย

จากโซลินอยที่นำมาใช้ในโครงงานนี้จะเป็นโซลินอยแบบ 2 ขดลวดคือจะมีขดลวดที่ใช้ในการผลักและดูดซึ่งขดลวดน้ำตาล , เขียว จะทำหน้าที่ผลักและขดลวดน้ำตาล , เหลืองจะทำหน้าที่ดูด วิธีการนำขดลวดทั้ง 2 มาใช้งานก็คือจะต้องจ่ายไฟบวกเข้าทางขดลวดสีน้ำตาลและไฟลบเข้าขดลวดสีเขียวในกรณีที่ต้องการให้โซลินอยผลักและต้องจ่ายไฟบวกเข้าทางขดลวดสีน้ำตาลและไฟลบเข้าขดลวดสีเหลืองในกรณีที่ต้องการให้โซลินอยดูด

3.2 การออกแบบซอร์ฟแวร์

แนวความคิดในการเขียน โปรแกรม คือ เมื่อรับคีย์สวิตช์ก็จะตรวจสอบว่าการคีย์ที่เข้ามานั้นเป็นคีย์ของการเปลี่ยนรหัส หรือคีย์ของรหัสเปิดประตู เมื่อตรวจสอบว่าเป็นคีย์ในการเปลี่ยนรหัส ก็จะมีการเรียกให้ผู้ใช้คีย์รหัสเก่า เมื่อคีย์รหัสเก่าถูกต้อง ก็จะให้คีย์รหัสใหม่ได้ ถ้าคีย์รหัสเก่าไม่ถูกต้องต้องทำการคีย์ใหม่ ส่วนถ้าตรวจสอบว่าเป็นการคีย์รหัสเปิดประตูไม่ใช่เป็นการเปลี่ยนรหัส ก็จะนำค่าที่คีย์เข้าไปมาเปรียบเทียบกับรหัสที่ถูกต้อง ถ้ารหัสถูกต้อง ก็จะสั่งให้ไมโครคอนโทรลเลอร์ควบคุมการเปิดประตู ถ้ารหัสที่คีย์ไม่ถูกต้องก็จะทำการรับรหัสใหม่อีกครั้ง ซึ่งสามารถเขียนออกมาเป็นโฟชาร์ทได้ดังรูปที่ 3.9

ในการสั่งให้ไมโครคอนโทรลเลอร์ควบคุมกลไกเพื่อให้คลายล็อคแล้วหลังจากนั้นต้องมีการหน่วงเวลาประมาณ 3 วินาที ในการที่จะสั่งให้ไมโครคอนโทรลเลอร์ควบคุมกลไกให้ล็อคอีกครั้ง เพราะว่าเป็นช่วงเวลาที่มีการทำการคลายล็อค เพื่อให้ผู้ใช้เปิดประตูเข้าไปได้หลังจาก 3 วินาทีแล้วประตูจะล็อคอีกครั้ง โปรแกรมบางส่วนในโครงงานสามารถอธิบายได้ดังนี้

โปรแกรมที่สั่งให้ไดโอดเปล่งแสง 7 ส่วน แสดงตัวเลข 1 (หลักเดียว) ทำได้โดย

```
MOV   P0,#F9  (ส่งเลข 1 ออกไปที่พอร์ท 0 ซึ่งมีไดโอดเปล่งแสง 7 ส่วนต่ออยู่ )
MOV   P1,#01  (ส่งเลข 1 ออกไปที่พอร์ท 1 ซึ่งเป็นการขับคอมมอนของ LED)
```

โปรแกรมแสดงผลไดโอดเปล่งแสง 7 ส่วน แบบ มัลติเพล็ก 4 หลัก ซึ่งแสดงผลจากหลักที่ 1 ไปยังหลักที่ 2 หลักที่ 3 และหลักที่ 4 ตามลำดับ

```
        MOV   A,#01
        MOV   P0,#F9          (ส่งเลข 1 ออกที่พอร์ท 0)
LOOP:   MOV   P1,A            (ส่งค่า 1 ออกไปที่พอร์ท 1 ให้ LED หลักแรกติด)
        ACALL DELAY-15        (หน่วงเวลาให้มองเห็นการติดของ LED)
        RL    A               (เลื่อนค่าของ A มาทางซ้าย 1 บิต)
        CJNE  A,#10,LOOP      (เปรียบเทียบค่า A = 10 หรือไม่ถ้าไม่กลับไปที่ LOOP)
        END
```

โปรแกรมหน่วงเวลา

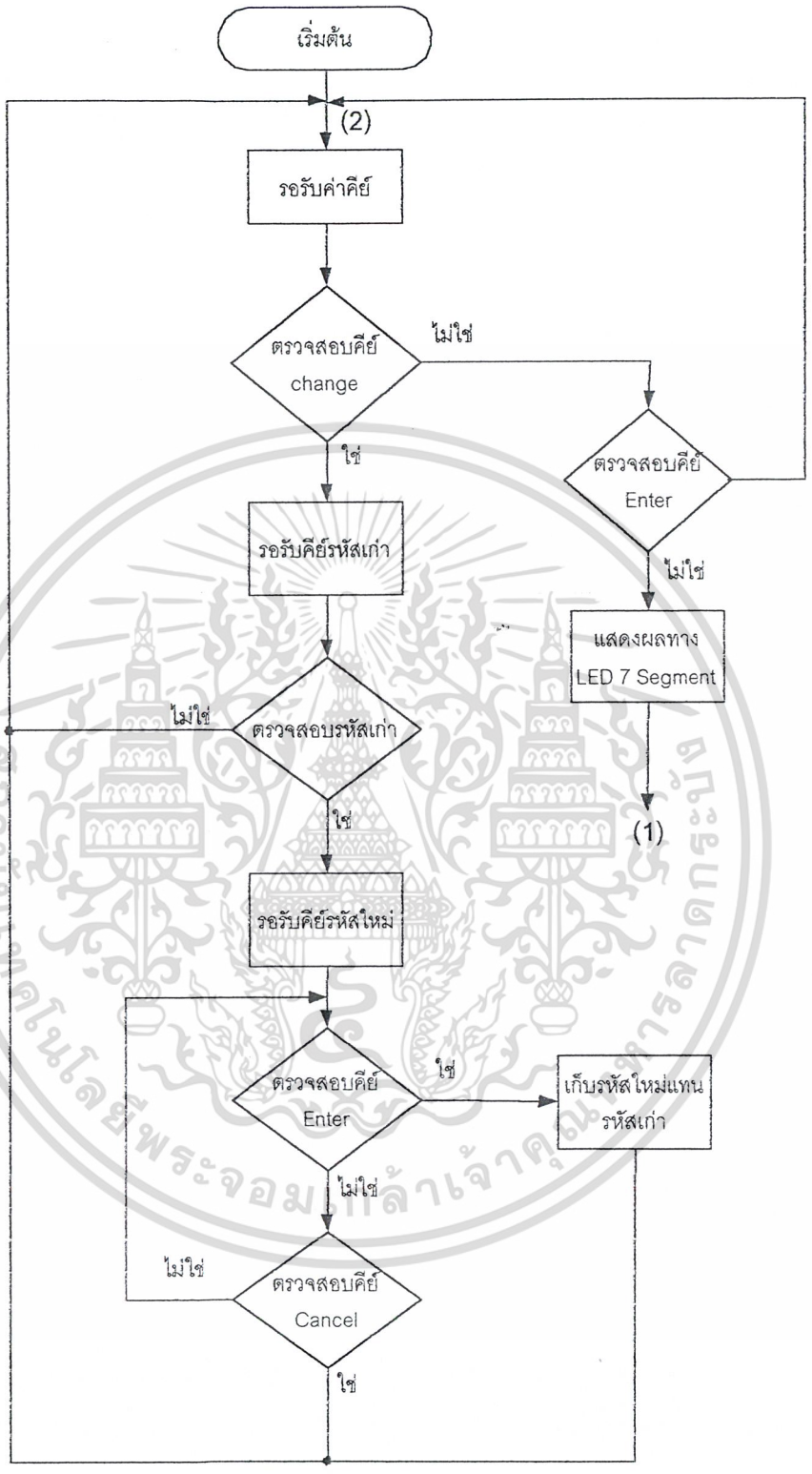
```
DELAY_10MS :      MOV   R7,#010
DELAY_10MS_1 :    MOV   R,0E6H
DELAY_10MS_2 :    NOP
                  NOP
                  DJNZ  R6, DELAY_10MS_2
                  DJNZ  R7,DELAY_10MS_1
                  RET
DELAY_1S :        MOV   R5,#100
DELAY_1S_1 :      ACALL DELAY_10MS
                  DJNZ  R5,DELAY_1S_1
                  RET
```

โปรแกรมรับค่าคีย์ เริ่มจากการกำหนดให้ P1,#0FFH ก่อนเพื่อเป็นการกำหนดค่าเริ่มต้น แล้วจึงทำการเคลียร์บิต COL0 และ AND A,#0FH เพื่อทำการทดลองว่ามีการกดคีย์ที่ COL0 หรือไม่ ถ้าไม่มีจึงไปที่ COL1 และตามด้วย COL2 ถ้าไม่มีการเปลี่ยนแปลงเลยแสดงว่าไม่มีการกดคีย์ใดๆ แต่ถ้าที่ COL0 , COL1 และ COL2 มีการเปลี่ยนแปลง โดยถ้าที่ COL0 มีการกดคีย์ก็ให้ทำการเก็บค่า 1 ไว้ในบัฟเฟอร์ก่อน แต่ถ้าเป็น COL1 และ COL2 ก็ให้เก็บ 2 และ 3 ตามลำดับ แล้วจึงไปทำการ

ตรวจสอบที่ ROW0 , ROW1 , ROW2 และ ROW3 ต่อไป โดยถ้าที่ ROW0 มีการกดก็ให้ออกจากโปรแกรม แต่ถ้าที่ ROW1 ก็ให้ทำการบวกค่าในบัฟเฟอร์ด้วย 3 และที่ ROW2 และ ROW3 ก็ให้บวกด้วย 6 และ 9 ตามลำดับและจึงออกจากโปรแกรม ดังโปรแกรมข้างล่าง

```
GET_KPAD :      MOV     P2,#0FFH
                MOV     DATA,#0
CHK_COL0 :      CLR     COL0
                MOV     A,P2
                ANL     A,#00FH
                CJNE    A,#00FH,COL0_DETECT
                AJMP    CHK_COL1
COL0_DETECT :   MOV     DATA,#01
                AJMP    GET_ROW
CHK_COL1 :      SETB    COL0
                CLR     COL1
                MOV     A,P2
                ANL     A,#00FH
                CJNE    A,#00FH,COL1_DETECT
                AJMP    CHK_COL2
COL1_DETECT :   MOV     DATA,#02
                AJMP    GET_ROW
CHK_COL2 :      SETB    COL1
                CLR     COL2
                MOV     A,P2
                ANL     A,#00FH
                CJNE    A,#00FH,COL2_DETECT
                RET
COL2_DETECT :   MOV     DATA,#03
GET_ROW :       CLR     COL0
                CLR     COL1
                CLR     COL2
```

```
                JB      ROW0,CHK_ROW1
                RET
CHK_ROW1 :      JB      ROW1,CHK_ROW2
                MOV     A,DATA
                ADD     A,#3
                MOV     DATA,A
                RET
CHK_ROW2 :      JB      ROW2,CHK_ROW3
                MOV     A,DATA
                ADD     A,#6
                MOV     DATA,A
                RET
CHK_ROW3 :      MOV     A,DATA
                ADD     A,#9
                MOV     DATA,A
                RET
```



รูปที่ 3.9 แสดงโฟชาร์ทการทำงานของโครงงาน

รูปที่ 3.9 (ต่อ) แสดงโฟชาร์ทการทำงานของโครงงาน

บทที่ 4

การทดลองและผลการทดลอง

ลำดับขั้นในการทดลอง

4.1 ทำการตรวจสอบอุปกรณ์ที่สำคัญก่อนการนำไปต่อใช้งาน

4.1.1 การตรวจสอบไอซีไมโครคอนโทรลเลอร์ (AT89S8252)

ทำการตรวจสอบว่าไอซีไมโครคอนโทรลเลอร์ เสียหรือใช้งานได้หรือไม่ ทำการตรวจสอบดังนี้

- จ่ายไฟให้แก่วงจร
- ขณะที่ยังไม่มีการกดคีย์ใดๆส่งค่า FF ออกไปที่พอร์ท 0
- นำมิเตอร์มาวัดที่พอร์ท 0 คือขา P0.0 ถึง P0.7 วัดค่าออกมาได้ 3 โวลท์
- นำมิเตอร์มาวัดที่พอร์ท 1 วัดค่าได้ 4.8 โวลท์
- ทำการกดคีย์ และทำการส่งค่า F9 ออกไปที่พอร์ท 0
- นำมิเตอร์มาวัดพอร์ทที่มีคำสั่งจากโปรแกรมที่ส่งค่า 1 วัดได้ 0.6 โวลท์ ส่วนวัดพอร์ทที่มีคำสั่งจากโปรแกรมให้ส่งค่า 0 วัดค่าได้ 0.22 โวลท์
- นำมิเตอร์วัดที่พอร์ท 1 วัดได้ 2.8 โวลท์ ขณะโปรแกรมส่งค่า 1 และ 0 ออกทางพอร์ท เมื่อทำการตรวจสอบจากการวัดค่าต่างๆก็สรุปได้ว่าไมโครคอนโทรลเลอร์ตัวนี้สามารถใช้งานได้

4.1.2 การทดลองทดสอบการแสดงผลโดยไดโอดเปล่งแสง 7 ส่วน

- ทำการส่งข้อมูลออกทางพอร์ท 0
- สั่งให้พอร์ท 2 มีค่าเป็น "1" ทีละบิตโดยเริ่มจากหลักที่ 1 ปรากฏว่าไดโอดเปล่งแสง 7 ส่วน หลักที่ 1 ติดสว่าง
- สั่งให้พอร์ท 2 มีค่าเป็น "1" ในหลักที่ 2 , 3 และ 4 ตามลำดับปรากฏว่าไดโอดเปล่งแสง 7 ส่วน ติดสว่างตามหลักที่ 2 , 3 และ 4 ตามลำดับ
- ทดลองให้หลักที่ 1 แสดงผล และตามด้วยหลักที่ 1 และหลักที่ 2 แสดงผลพร้อมกัน , ให้หลักที่ 1 , 2 และ 3 แสดงผลพร้อมกัน , ให้หลักที่ 1 , 2 , 3 และ 4 แสดงผลพร้อมกันปรากฏว่าแสดงผลได้ถูกต้อง

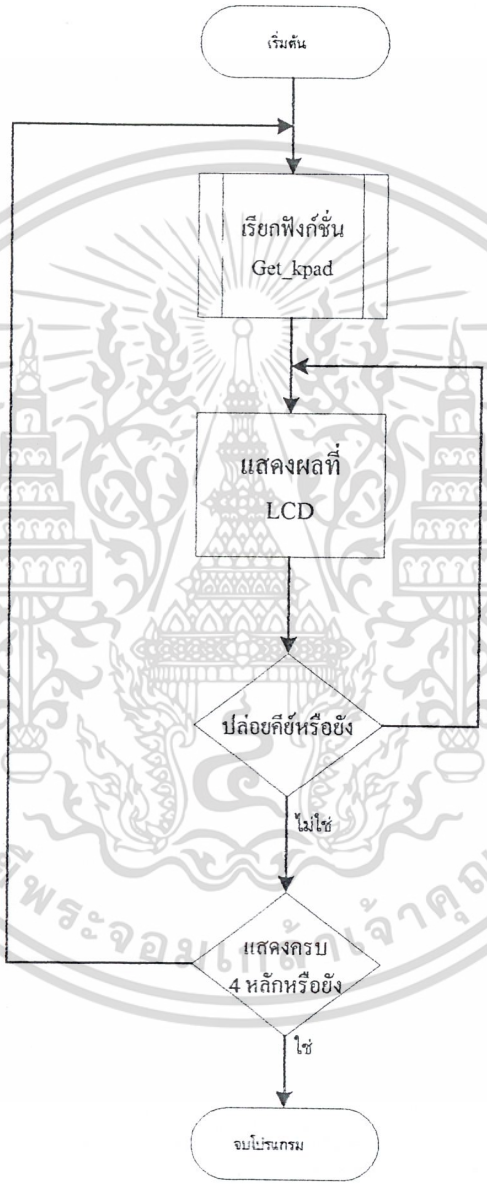
4.1.3 การทดลองทดสอบคีย์สวิตช์

การทดสอบคีย์สวิตช์ทำได้โดยทำการต่อเข้ากับส่วนไมโครคอนโทรลเลอร์และส่วนแสดงผลรวมกันแล้วทำการเขียนโปรแกรมรับค่าคีย์ที่เข้ามา เมื่อมีการทดสอบกดคีย์ตัวเลขต่างๆปรากฏว่ามีการแสดงผลตามที่กดคีย์แสดงว่าคีย์สวิตช์ใช้งานได้

4.2 ทำการทดลองเมื่อประกอบโครงงานเสร็จ

การทดลองกดค่าจากคีย์แพดแล้วแสดงผลที่ไดโอดเปล่งแสง 7 ส่วน

- เขียนโฟชาร์ทของโปรแกรม แสดงได้ดังรูปที่ 4.1



รูปที่ 4.1 แสดงโฟชาร์ทการกดค่าจากคีย์แพดแล้วแสดงผลที่ไดโอดเปล่งแสง 7 ส่วน

รูปที่ 4.2 แสดงโฟชาร์ทของฟังก์ชั่น Get_kpad

- เขียนโปรแกรมจากโฟชาร์ท
- ทำการโปรแกรมลงในไอซีไมโครคอนโทรลเลอร์ แล้วนำไปประกอบลงในวงจร
- ป้อนไฟให้แก่วงจร
- ทดลองกดคีย์แพดเริ่มจากกด 0 0 0 0 กดศูนย์ 4 ครั้งปรากฏว่าที่ไดโอดเปล่งแสง 7 ส่วนแสดงหมายเลข 0 ทั้ง 4 หลัก
- ทดลองกดคีย์แพดเลข 1 1 1 1 กดหนึ่ง 4 ครั้ง ปรากฏว่าที่ไดโอดเปล่งแสง 7 ส่วนแสดงหมายเลข 1 ทั้งหลักที่ 1 , 2 , 3 และ 4

- ทำเหมือนเดิมไล่จนครบทุกตัวปรากฏว่าที่ไดโอดเปล่งแสง 7 ส่วนแสดงหมายเลขได้ถูกต้อง

การทดลองการทำงานร่วมกันทั้งหมดของโครงงาน

- เขียนโฟชาร์ทการทำงานทั้งหมดของโครงงาน ซึ่งดูได้จากรูปที่ 3.9
- เขียนโปรแกรมจากโฟชาร์ทดูได้จากภาคผนวก
- ทำการโปรแกรมลงในไอซีไมโครคอนโทรลเลอร์ แล้วนำไปประกอบลงในวงจร
- ป้อนไฟให้แก่วงจร
- ทำรหัสการเปิดประตู ในตอนแรกเราจะได้รหัสจากการเขียนโปรแกรม ซึ่งให้เป็นรหัสลับ ซึ่งรหัสลับก็ยังคงอยู่จะเป็นรหัสที่ไม่สามารถแก้ไขได้ และเราสามารถที่จะทำรหัสใหม่ได้โดยต้องใส่รหัสลับให้ถูกต้องก่อน ซึ่งรหัสลับคือ 1 3 5 7 ดังนั้นเราจะมีรหัส 2 รหัส คือ รหัสลับ กับ รหัสที่เราเปลี่ยนแปลงได้จากการกดคีย์แพด เราทำการทดลองโดยกด 1 3 5 7 จะเกิดไฟสว่างขึ้น 4 จุด ต่อจากนั้นทำการกดรหัสใหม่ โดยต้องกดให้ครบ 4 หลัก ซึ่งจะได้เป็นรหัสในการเปิดประตู
- ลองใส่รหัสที่ผิด สังเกตว่ามีผลอะไรเกิดขึ้น ปรากฏว่าโซลินอยไม่มีการเปลี่ยนแปลง ประตูก็ยังไม่สามารถเปิดได้ ต่อจากนั้นทำการวัดค่าที่จุดต่างๆดังนี้

จุดที่วัด	ค่าที่วัดได้ (v)	จุดที่วัด	ค่าที่วัดได้ (v)
P0.0	-0.045	P2.0	5
P0.1	-0.045	P2.1	5
P0.2	-0.045	P2.2	5
P0.3	-0.045	P2.3	5
P0.4	-0.045	P2.4	4.09
P0.5	-0.045	P2.5	3.91
P0.6	-0.045	P2.6	3.92
P0.7	-0.045	P2.7	ไม่ใช้
P1.0	-0.012	TR1 VCE	2.64
P1.1	0.592	TR2 VCE	2.64
P1.2	ไม่ใช้	TR3 VCE	2.64
P1.3	-0.035	TR4 VCE	2.64
P1.4	-0.034	ULN2003 ขา 3	3.6

จุดที่วัด	ค่าที่วัดได้ (v)	จุดที่วัด	ค่าที่วัดได้ (v)
P1.5	-0.034	ULN2003 ขา 4	0.03
P1.6	-0.034	ULN2003 ขา 13	9.83
P1.7	ไม่ใช้	ULN2003 ขา 14	1.147

ตารางที่ 4.1 แสดงการวัดค่าที่จุดต่างๆเมื่อใส่รหัสที่ผิด

- ลองใส่รหัสที่ถูกต้องสังเกตว่ามีอะไรเกิดขึ้น ปรากฏว่าโซลินอยดึงเดือยกลับเข้าหาตัวเองทำให้กลไกหมุนไปมาได้ และสามารถที่จะเปิดประตูได้ ต่อจากนั้นทำการวัดค่าที่จุดต่างๆดังนี้

จุดที่วัด	ค่าที่วัดได้ (v)	จุดที่วัด	ค่าที่วัดได้ (v)
P0.0	2.096	P2.0	4.97
P0.1	2.906	P2.1	4.97
P0.2	2.894	P2.2	4.97
P0.3	2.901	P2.3	4.97
P0.4	2.902	P2.4	-0.028
P0.5	2.902	P2.5	-0.019
P0.6	0.438	P2.6	-0.013
P0.7	2.891	P2.7	ไม่ใช้
P1.0	5.82	TR1 VCE	-0.624
P1.1	-0.023	TR2 VCE	-0.624
P1.2	ไม่ใช้	TR3 VCE	-0.624
P1.3	2.904	TR4 VCE	-0.624
P1.4	2.906	ULN2003 ขา 3	0.045
P1.5	2.906	ULN2003 ขา 4	3.68
P1.6	2.906	ULN2003 ขา 13	1.06
P1.7	ไม่ใช้	ULN2003 ขา 14	9.84

ตารางที่ 4.2 แสดงการวัดค่าที่จุดต่างๆเมื่อใส่รหัสที่ถูกต้อง

บทที่ 5
บทสรุป

ในส่วนของการแสดงผลเราจะใช้ไดโอดเปล่งแสง 7 ส่วน แสดงตัวเลขรหัสที่ผู้ใช้คีย์สวิตช์ลงไป ซึ่งในโครงงาน จะใช้ไดโอดเปล่งแสง 7 ส่วนจำนวน 4 ตัวก็จะหมายถึง การแสดงตัวเลขเป็นจำนวน 4 หลัก โดยต่อขา a , b ไปจนถึง g รวมทั้งขา dp ของไดโอดเปล่งแสง 7 ส่วนต่อเข้ากับพอร์ท 0 ของ AT89S8252 โดยมีตัวต้านทานเข้ามาคั่นก่อนก็เพื่อที่จะป้องกันไม่ให้เกิดความเสียหายกับไดโอดเปล่งแสง 7 ส่วน เพราะอาจมีกระแสเกินค่าที่ไดโอดเปล่งแสงรับได้ ส่วนขาคอมมอนจะต่อกับพอร์ท 2 ของ AT89S8252 โดยผ่านตัวทรานซิสเตอร์ก่อนเพราะตัวทรานซิสเตอร์ที่ต่อเข้าไปทำหน้าที่ในการขับกระแสลงกราวจ์และขาเบสของทรานซิสเตอร์ก็จะต่อเข้าทางพอร์ท 2 ของ AT89S8252 ซึ่งถ้าจะทำการแสดงผลทางไดโอดเปล่งแสง 7 ส่วนก็จะสามารถทำได้ดังที่ได้ทดลองมา คือทำการเขียนโปรแกรมควบคุมให้แสดงผลออกทางไดโอดเปล่งแสง 7 ส่วนซึ่งจะแสดงผลตามรหัสที่คีย์

โครงงานจะใช้สวิตช์แบบเมทริกซ์ หรือที่เรียกว่าคีย์แพด ใช้พอร์ทของ AT89S8252 จำนวนหนึ่งพอร์ทซึ่งในโครงงานนี้ใช้นาด 4 × 3 ต่อเข้ากับ AT89S8252 ที่พอร์ท P2 การต่อสวิตช์ลักษณะนี้จะต้องโปรแกรมสแกนคีย์สวิตช์เพื่อที่จะเช็คการกดคีย์รหัส ส่วนรอรับคีย์สวิตช์จากผู้ใช้ซึ่งเป็นคีย์แพด เมื่อเริ่มใช้งานผู้ใช้ต้องกดคีย์รหัสเพื่อที่ให้รหัสนี้เป็นรหัสที่ใช้ในการเปิดประตู โปรแกรมจะใช้รหัสนี้ในการนำไปเปรียบเทียบกับข้อมูลที่กดคีย์ในครั้งต่อไปเพื่อเปิดประตู ถ้ารหัสที่คีย์เข้าไปเปรียบเทียบว่าถูกต้องโปรแกรมจะสั่งให้ไมโครคอนโทรลเลอร์ควบคุมในการเปิดประตู ถ้ารหัสที่คีย์เข้าไปเปรียบเทียบแล้วว่าไม่ถูกต้องโปรแกรมจะสั่งให้รับคีย์ข้อมูลใหม่ ส่วนกลไก เมื่อมีการป้อนรหัสที่ถูกต้องโปรแกรมจะสั่งให้ ไมโครคอนโทรลเลอร์ควบคุมให้โซลินอยเลื่อนเดือยล็อคออกจากกลไก ทำให้สามารถเปิดประตูได้ ซึ่งจากการทดลองเมื่อเราทำการใส่รหัสที่ไม่ถูกต้องปรากฏว่าไมโครคอนโทรลเลอร์ไม่ควบคุมให้โซลินอยเลื่อนเดือยออกจากตัวกลไก ทำให้กลไกยังล็อคประตูอยู่ไม่สามารถทำการเปิดประตูได้ และเมื่อเราทดลองใส่รหัสที่ถูกต้องตรงตามรหัสที่โปรแกรมไว้ปรากฏว่าไมโครคอนโทรลเลอร์จะควบคุมให้โซลินอยเลื่อนเดือยออกจากตัวกลไกทำให้กลไกสามารถหมุนได้ ดังนั้นจึงทำให้สามารถเปิดประตูได้

จากการทดลองวัดค่าต่างๆ เมื่อมีการใส่รหัสผิดหลังจากนั้นเราไปทำการวัดค่าที่พอร์ท 0 ปรากฏว่าค่าที่ได้ออกมาจะมีค่า -0.045 โวลท์ เพราะหลังจากการกดคีย์รหัสผิดจะไม่มีการแสดงผลที่ไดโอดเปล่งแสง 7 ส่วน และที่ทรานซิสเตอร์ตัวที่ 1 ถึง 4 มีแรงดันตกคร่อม 2.64 โวลท์แสดงว่าเกิดการย้อนกลับของกระแสจึงทำการขับลงกราวจ์

เมื่อมีการใส่รหัสที่ถูกต้องหลังจากนั้นเราไปทำการวัดค่าต่างๆปรากฏว่าที่พอร์ท 0 มีการส่งข้อมูลมายังไดโอดเปล่งแสง 7 ส่วนดูได้จากค่าที่วัดได้ซึ่งพร้อมที่จะแสดงผล ซึ่งขึ้นอยู่กับ P1.3 , P1.4 , P1.5 และ P1.6ในการเลือกตำแหน่งหลักที่จะแสดงผล ซึ่ง P1.3 คือหลักที่ 4 แสดงผล , P1.4 คือหลักที่ 3 แสดงผล P1.5 และ P1.6 คือหลักที่ 2 และ 1 แสดงผลตามลำดับ ซึ่งจากการทดลองผลที่วัดได้ ในพอร์ทที่ P1.3 , P1.4 , P1.5 และ P1.6 มีค่าประมาณ 2.9 โวลท์ ก็หมายถึงเป็นการให้แสดงผลทางไดโอดเปล่งแสง 7 ส่วนทั้ง 4 หลัก ส่วนพอร์ท P1.0 และ P1.1 จะเป็นการส่งสัญญาณเพื่อไปควบคุมโซลินอย ซึ่งก่อนที่จะส่งไปให้โซลินอย จะต้องผ่านตัวบัฟเฟอร์ก่อนซึ่งใช้เบอร์ ULN2003

ปัญหาในการทำโครงงานและในการทดลอง

- ในการหาซื้ออุปกรณ์บางตัวหาซื้อได้ยาก
- ราคาอุปกรณ์บางตัวมีราคาแพงต้องทำการหาอุปกรณ์ที่มีราคาถูกและสามารถทำงานได้เหมือนเดิมมาแทน เช่น LCD มีราคาแพงต้องเปลี่ยนไปใช้ไดโอดเปล่งแสง 7 ส่วนแทน
- ในการออกแบบวงจร โดยใช้ Protel 99 se หาอุปกรณ์บางตัวยาก และเมื่อต่อวงจรเสร็จทำการ update PCB แล้วมีข้อผิดพลาดในวงจรแสดงให้เห็น แล้วกลับไปทำการแก้ไขข้อบกพร่องบางจุดต้องใช้เวลานาน เมื่อ update PCB ผ่านแล้วไปทำ PCB เมื่อจัดวางอุปกรณ์แล้วทำการ route เมื่อเสร็จมีการชอร์ทของวงจร ต้องใช้เวลาในการแก้ไขนาน
- ทำการประกอบวงจรเสร็จทดลองการทำงาน ปรากฏว่าการทำงานไม่เป็นดังที่คิด คือเมื่อใส่รหัสที่ถูกต้องแล้ว ปรากฏว่าไม่มีอะไรเกิดขึ้น ตามปกติแล้วไมโครคอนโทรลเลอร์ต้องควบคุมให้โซลินอยเลื่อนเดือยออกจากการล็อคตัวกลไก
- ไฟที่จ่ายให้โซลินอยไม่พอต้องทำการแก้ไขวงจร

บรรณานุกรม

1. ธีรวัฒน์ ประกอบผล. การประยุกต์ใช้งานไมโครคอนโทรลเลอร์. กรุงเทพ : ดวงกมลสมัย จำกัด 2542 , 235 หน้า

2. วรพจน์ กรแก้ววัฒนกุล และ ชัยวัฒน์ ลิ้มพรจิตรวิไล. เรียนรู้และปฏิบัติการไมโครคอนโทรลเลอร์ MCS-51. แบบแฟลช. กรุงเทพ : บริษัทอินโนเวตีฟเอ็กเพอริเมนต์ จำกัด , 399 หน้า

3. Ayala,K.J.,"The 8051 Microcontroller Architecture,Programming and Application,"West Publishing Company,1991.

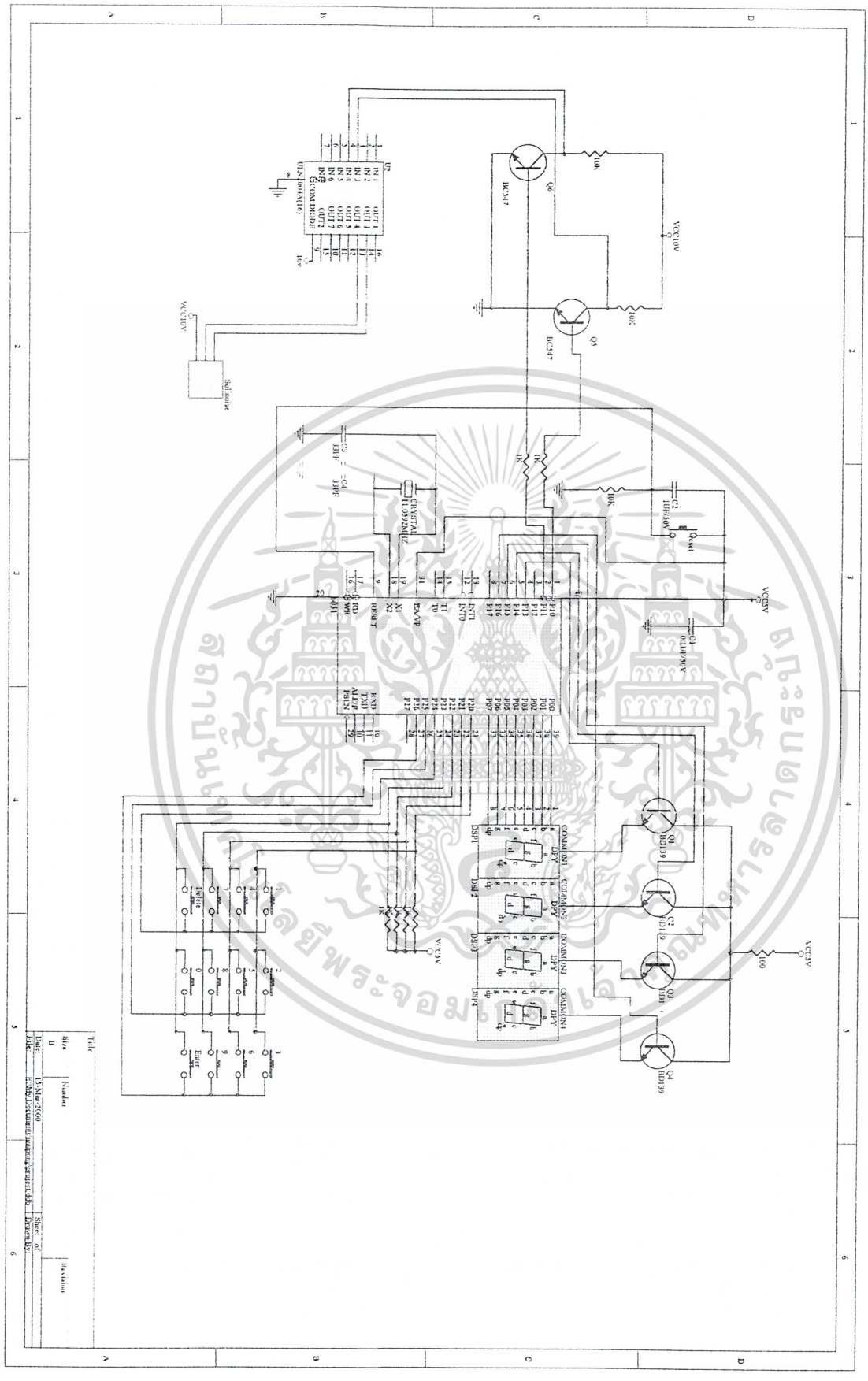
4. Intel Corporation,"Microcontroller Handbook,"Intel Corporation,1992.

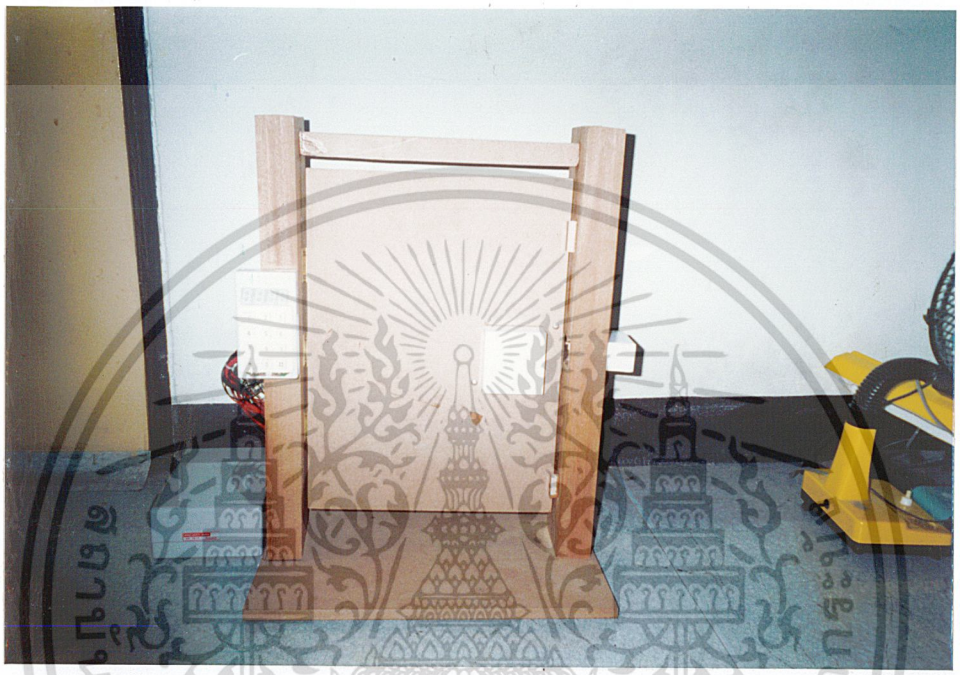
5. MacKenzie,1.Scott.,"Microcontroller,"Macmillan Publishing Company,1992.

ภาคผนวก

วิธีใช้เครื่องล็อคประตูแบบรหัส

1. ก่อนใช้ต้องมีการตั้งรหัสก่อน โดยกดปุ่ม change password หลังจากนั้นกดรหัสที่ถูกต้องโดยครั้งแรกที่เริ่มใช้รหัสที่ถูกต้องคือ 1 3 5 7 ซึ่งเป็นรหัสลับหลังจากนั้นจะมีจุดจำนวน 4 จุดแสดงที่ไดโอดเปล่งแสง 7 ส่วน ต่อจากนั้นให้กดรหัสที่เราต้องการจำนวน 4 หลัก
2. ต่อจากนั้นใช้รหัสที่ตั้งไว้สำหรับเปิดประตูโดยกดให้ถูกต้อง แล้วตามด้วยปุ่ม ENTER ตัวกลไกก็จะทำการคลายล็อคทำให้สามารถเปิดประตูได้ ถ้าใส่รหัสผิดต้องใส่รหัสใหม่จนถูกต้อง
3. เมื่อจะทำการเปลี่ยนรหัสอีก ต้องกดปุ่ม change password ต่อจากนั้นกดรหัสที่ถูกต้องที่ได้ตั้งไว้ในคราวที่แล้วเมื่อกดครบแล้ว จะมีจุดจำนวน 4 จุดแสดงที่ไดโอดเปล่งแสง 7 ส่วน ต่อจากนั้นให้กดรหัสที่เราต้องการจำนวน 4 หลัก เราก็จะได้รหัสใหม่สำหรับใช้เปิดประตู
4. กรณีที่ลืมรหัสเราก็จะสามารถใช้รหัสลับได้คือ 1 3 5 7 ซึ่งสามารถใช้ในการเปิดประตูได้ หรือจะใช้ในการเปลี่ยนรหัสดังขั้นตอนที่ 1





รูปโครงงาน

กรณีมี Safety code "1357"

```
; ********** VARIABLE SET **********
WMCON     EQU     96H          ; watchdog and memory control register
EEMEN     EQU     00001000b    ; EEPROM access enable bit
EEMWE     EQU     00010000b    ; EEPROM write enable bit
WDTRST    EQU     00000010b    ; EEPROM RDY/BSY bit
DSP1      EQU     P1.3
DSP2      EQU     P1.4
DSP3      EQU     P1.5
DSP4      EQU     P1.6
ROW0      EQU     P2.0
ROW1      EQU     P2.1
ROW2      EQU     P2.2
ROW3      EQU     P2.3
COL2      EQU     P2.4
COL1      EQU     P2.5
COL0      EQU     P2.6
BUF1      EQU     30H
BUF2      EQU     31H
BUF3      EQU     32H
BUF4      EQU     33H
DATA      EQU     34H
PASS      EQU     35H
BUFP1     EQU     36H
BUFP2     EQU     37H
BUFP3     EQU     38H
BUFP4     EQU     39H
SB1       EQU     40H
SB2       EQU     41H
SB3       EQU     42H
SB4       EQU     43H
```

```
SECODE1     EQU     44H
SECODE2     EQU     45H
SECODE3     EQU     46H
SECODE4     EQU     47H

            ORG     0000H
MAIN:       MOV     R0,#0
            MOV     R1,#0
            MOV     R2,#4
            MOV     R3,#100
            MOV     P0,#0
            MOV     P1,#0
            MOV     P2,#0FFH
            MOV     BUF1,#0
            MOV     BUF2,#0
            MOV     BUF3,#0
            MOV     BUF4,#0
            CLR     P1.0
            SETB    P1.1 ;THE DOOR CLOSE
            MOV     SECODE1,#0F9H       ; DATA=1
            MOV     SECODE2,#0B0H       ; DATA=3
            MOV     SECODE3,#92H        ; DATA=5
            MOV     SECODE4,#0F8H       ; DATA=7
GET_A:      ACALL   GET_KPAD
            MOV     A,DATA
            JZ      GET_A
            CJNE    A,#0AH,LABEL1
            CALL    CH_PASS
            JMP     GET_A
LABEL1:     CJNE    A,#0CH,LABEL2
            JMP     GET_A
```

```
LABEL2:   MOV     DPTR,#DSP_BLANK
          MOVC    A,@A+DPTR
          MOV     BUF1,A
JJ:       CJNE    R0,#0,NEXT1
          MOV     P0,BUF1
          SETB    DSP1
          CLR     DSP2
          CLR     DSP3
          CLR     DSP4
          ACALL   DELAY
          MOV     R1,P2
          MOV     B,R1
          ANL     B,#00FH
          MOV     R1,B
          CJNE    R1,#00FH,JJ
          ACALL   GET_KPAD
          MOV     A,DATA
          JZ      JJ
          CJNE    A,#0AH,LABEL9
          JMP     JJ
LABEL9:   CJNE    A,#0CH,LABEL10
          JMP     JJ
LABEL10:  INC     R0
NEXT1:    MOV     DPTR,#DSP_BLANK
          MOVC    A,@A+DPTR
          CJNE    R0,#1,NEXT2
          MOV     BUF2,A
ZZZ:      MOV     P0,BUF1
          SETB    DSP1
          CLR     DSP2
          ACALL   DELAY
```

```
                CLR     DSP1
                MOV     P0,BUF2
                SETB    DSP2
                ACALL   DELAY
                MOV     R1,P2
                MOV     B,R1
                ANL     B,#00FH
                MOV     R1,B
                CJNE    R1,#00FH,ZZZ
                ACALL   GET_KPAD
                MOV     A,DATA
                JZ      ZZZ
                CJNE    A,#0AH,LABEL3
                JMP     ZZZ
LABEL3:         CJNE    A,#0CH,LABEL4
                JMP     ZZZ
LABEL4:         INC     R0
NEXT2:          CJNE    R0,#2,NEXT3
                MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR
                MOV     BUF3,A
ZZZ2:           MOV     P0,BUF1
                SETB    DSP1
                CLR     DSP2
                CLR     DSP3
                ACALL   DELAY
                CLR     DSP1
                MOV     P0,BUF2
                SETB    DSP2
                ACALL   DELAY
                CLR     DSP1
```

```
            CLR     DSP2
            MOV     P0,BUF3
            SETB    DSP3
            ACALL   DELAY
            MOV     R1,P2
            MOV     B,R1
            ANL     B,#00FH
            MOV     R1,B
            CJNE    R1,#00FH,ZZZ2
            ACALL   GET_KPAD
            MOV     A,DATA
            JZ      ZZZ2
            CJNE    A,#0AH,LABEL5
            JMP     ZZZ2
LABEL5:     CJNE    A,#0CH,LABEL6
            JMP     ZZZ2
LABEL6:     INC     R0
NEXT3:      CJNE    R0,#3,UUU
            MOV     DPTR,#DSP_BLANK
            MOVC    A,@A+DPTR
            MOV     BUF4,A
ZZZ3:       MOV     P0,BUF1
            SETB    DSP1
            CLR     DSP2
            CLR     DSP3
            CLR     DSP4
            ACALL   DELAY
            CLR     DSP1
            MOV     P0,BUF2
            SETB    DSP2
            ACALL   DELAY
```

```
        CLR     DSP2
        MOV     P0,BUF3
        SETB    DSP3
        ACALL   DELAY
        CLR     DSP3
        MOV     P0,BUF4
        SETB    DSP4
        ACALL   DELAY
        MOV     R1,P2
        MOV     B,R1
        ANL     B,#00FH
        MOV     R1,B
        CJNE    R1,#00FH,ZZZ3
        ACALL   GET_KPAD
        MOV     A,DATA
        JZ      ZZZ3
        CJNE    A,#0AH,LABEL7
        JMP     ZZZ3
LABEL7: CJNE    A,#0CH,ZZZ3
        CALL    ENTER
        MOV     R0,#0
        MOV     P1,#00000010b
UUU:    LJMP    GET_A
ENTER:  MOV     DPTR,#0000H
        CALL    ReadEE
        MOV     BUFP1,A
        MOV     DPTR,#0001H
        CALL    ReadEE
        MOV     BUFP2,A

        MOV     DPTR,#0002H
```

```
                CALL        ReadEE
                MOV         BUFP3,A
                MOV         DPTR,#0003H
                CALL        ReadEE
                MOV         BUFP4,A
CODESECREST:MOV             SB1,SECODE1
                MOV         SB2,SECODE2
                MOV         SB3,SECODE3
                MOV         SB4,SECODE4
                MOV         A,SB1
                CJNE        A,BUF1,CP
                MOV         A,SB2
                CJNE        A,BUF2,CP
                MOV         A,SB3
                CJNE        A,BUF3,CP
                MOV         A,SB4
                CJNE        A,BUF4,CP
                JMP         SETNEWPASSWORD
CP:             JMP         CODEPASS
SETNEWPASSWORD:
                MOV         P0,#7FH                 ;PRINT ....
                MOV         P1,#01111010B
SNP1:           ACALL       GET_KPAD
                MOV         A,DATA
                JZ          SNP1
                CJNE        A,#0AH,LABEL11
                JMP         SNP1
LABEL11:        CJNE        A,#0CH,LABEL12
                JMP         SNP1

LABEL12:        MOV         DPTR,#DSP_BLANK
```

```
                MOVC    A,@A+DPTR
                MOV     P1,#01110010B
                MOV     PASS,A          ;
                MOV     DPTR,#0000H     ; WRITE DATA TO ADDRESS 0000H
                CALL    WriteEE
                CALL    CH_KEY
SNP2:           ACALL   GET_KPAD
                MOV     A,DATA
                JZ      SNP2
                CJNE    A,#0AH,LABEL13
                JMP     SNP2
LABEL13:        CJNE    A,#0CH,LABEL14
                JMP     SNP2
LABEL14:        MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR
                MOV     P1,#01100010B
                MOV     PASS,A          ;
                MOV     DPTR,#0001H     ; WRITE DATA TO ADDRESS 0001H
                CALL    WriteEE
                CALL    CH_KEY
SNP3:           ACALL   GET_KPAD
                MOV     A,DATA
                JZ      SNP3
                CJNE    A,#0AH,LABEL15
                JMP     SNP3
LABEL15:        CJNE    A,#0CH,LABEL16
                JMP     SNP3
LABEL16:        MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR

                MOV     P1,#01000010B
```

```
                MOV     PASS,A          ;
                MOV     DPTR,#0002H     ; WRITE DATA TO ADDRESS 0002H
                CALL    WriteEE
                CALL    CH_KEY
SNP4:           ACALL   GET_KPAD
                MOV     A,DATA
                JZ      SNP4
                CJNE    A,#0AH,LABEL17
                JMP     SNP4
LABEL17:        CJNE    A,#0CH,LABEL18
                JMP     SNP4
LABEL18:        MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR
                MOV     P1,#00000010B
                MOV     PASS,A          ;
                MOV     DPTR,#0003H     ; WRITE DATA TO ADDRESS 0003H
                CALL    WriteEE
                CALL    CH_KEY
                MOV     P0,#0BFH                ;PRINT ----
                MOV     P1,#01111010B
                CALL    DELAY_1S
                JMP     FL_PASS
CODEPASS:       MOV     A,BUFP1
                CJNE    A,BUF1,FL_PASS
                MOV     A,BUFP2
                CJNE    A,BUF2,FL_PASS
                MOV     A,BUFP3
                CJNE    A,BUF3,FL_PASS
                MOV     A,BUFP4
                CJNE    A,BUF4,FL_PASS
TR_PASS:        MOV     BUF1,#0
```

```
                MOV     BUF2,#0
                MOV     BUF3,#0
                MOV     BUF4,#0
                MOV     P0,#0BFH
                MOV     P1,#01111010B
                SETB    P1.0                    ;THE DOOR OPEN
                CLR     P1.1
                CALL    DELAY_1S
                CALL    DELAY_1S
                CLR     P1.0
                SETB    P1.1                    ;THE DOOR CLOSE
FL_PASS:        MOV     P0,#0FFH
                RET

WR_PASS:        MOV     P1,#72H
                CALL    CH_KEY
          ww:   CALL    GET_KEY
                CJNE    A,#0AH,LABEL36
                JMP     ww
LABEL36:        CJNE    A,#0CH,LABEL31
                JMP     ww
LABEL31:        MOV     P1,#62H
                CALL    CH_KEY
         ww1:   CALL    GET_KEY
                CJNE    A,#0AH,LABEL32
                JMP     ww1
LABEL32:        CJNE    A,#0CH,LABEL33
                JMP     ww1
LABEL33:        MOV     P1,#42H
                CALL    CH_KEY
ww2:            CALL    GET_KEY
```

```
                CJNE    A,#0AH,LABEL34
                JMP     ww2
LABEL34:        CJNE    A,#0CH,LABEL35
                JMP     ww2
LABEL35:        MOV     P1,#02H
                CALL    CH_KEY
                LJMP    GET_A
CH_PASS:        CALL    CH_KEY
                MOV     P0,#01111111B
                MOV     P1,#01111010B
                MOV     DPTR,#0000H
                CALL    ReadEE
                MOV     BUF1,A
                MOV     DPTR,#0001H
                CALL    ReadEE
                MOV     BUF2,A
                MOV     DPTR,#0002H
                CALL    ReadEE
                MOV     BUF3,A
                MOV     DPTR,#0003H
                CALL    ReadEE
                MOV     BUF4,A
pp:             CALL    GET_KEY         ;GET KEY AND CHECK OLD PASSWORD
                CJNE    A,#0AH,LABEL40
                JMP     pp
LABEL40:        CJNE    A,#0CH,LABEL41
                JMP     pp
label41:        MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR
tra:            CJNE    A,BUF1,WR_PASS
                MOV     P1,#70H
```

```
                CALL    CH_KEY
pp1:            CALL    GET_KEY
                CJNE    A,#0AH,LABEL42
                JMP     pp1
LABEL42:        CJNE    A,#0CH,LABEL43
                JMP     pp1
LABEL43:        MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR
                CJNE    A,BUF2,tra
                MOV     P1,#60H
                CALL    CH_KEY
pp2:            CALL    GET_KEY
                CJNE    A,#0AH,LABEL44
                JMP     pp2
LABEL44:        CJNE    A,#0CH,LABEL45
                JMP     pp2
LABEL45:        MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR
                CJNE    A,BUF3,tra
                MOV     P1,#40H
                CALL    CH_KEY
pp3:            CALL    GET_KEY
                CJNE    A,#0AH,LABEL21
                JMP     pp3
LABEL21:        CJNE    A,#0CH,LABEL22
                JMP     pp3
LABEL22:        MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR
                CJNE    A,BUF4,tra
                MOV     P1,#00H
                CALL    CH_KEY
```

```
                MOV     P0,#7FH
                MOV     P1,#78H                 ;END
pp4:            CALL    GET_KEY                 ;GET NEW PASSWORD
                CJNE    A,#0AH,LABEL23
                JMP     pp4
LABEL23:        CJNE    A,#0CH,LABEL24
                JMP     pp4
LABEL24:        MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR
                MOV     DPTR,#0000H
                MOV     PASS,A
                CALL    WriteEE
                MOV     P1,#70H
                CALL    CH_KEY
pp5:            CALL    GET_KEY
                CJNE    A,#0AH,LABEL25
                JMP     pp5
LABEL25:        CJNE    A,#0CH,LABEL26
                JMP     pp5
LABEL26:        MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR
                MOV     DPTR,#0001H
                MOV     PASS,A
                CALL    WriteEE
                MOV     P1,#60H
                CALL    CH_KEY
pp6:            CALL    GET_KEY
                CJNE    A,#0AH,LABEL27
                JMP     pp6
LABEL27:        CJNE    A,#0CH,LABEL28
                JMP     pp6
```

```
LABEL28:    MOV     DPTR,#DSP_BLANK
            MOVC    A,@A+DPTR
            MOV     DPTR,#0002H
            MOV     PASS,A
            CALL    WriteEE
            MOV     P1,#40H
            CALL    CH_KEY
pp7:        CALL    GET_KEY
            CJNE    A,#0AH,LABEL29
            JMP     pp7
LABEL29:    CJNE    A,#0CH,LABEL30
            JMP     pp7
label30:    MOV     DPTR,#DSP_BLANK
            MOVC    A,@A+DPTR
            MOV     DPTR,#0003H
            MOV     PASS,A
            CALL    WriteEE
            MOV     P1,#00H
            CALL    CH_KEY              ;END
            MOV     P0,#0BFH
            MOV     P1,#01111010B
            RET

;JUMP_WR:   JMP     WR_PASS
GET_KEY:
GET:        CALL    GET_KPAD
            MOV     A,DATA
            Z       GET
            RET
```

```
CH_KEY:
CH_K:     MOV     R1,P2
          MOV     B,R1
          ANL     B,#00FH
          MOV     R1,B
          CJNE    R1,#00FH,CH_K
          RET

WriteEE:
          ORL     WMCON,#EEMEN      ; enable EEPROM accesses
          ORL     WMCON,#EEMWE      ; enable EEPROM writes
          MOV     A,PASS            ; data to write
          MOVX    @DPTR,A           ; write EEPROM
Loop:
          MOV     A,WMCON           ; get EEPROM write status
          ANL     A,#WDTRST         ; check RDY/BSY
          JZ      Loop              ;jump if busy
          MOVX    A,@DPTR           ; read EEPROM
          CJNE    A,PASS,WriteEE    ; jump if data compare fails
          XRL     WMCON,#EEMWE      ; disable EEPROM writes
          XRL     WMCON,#EEMEN      ; disable EEPROM accesses
          RET
ReadEE:
          ORL     WMCON,#EEMEN          ; enable EEPROM accesses
          MOVX    A,@DPTR               ; read EEPROM
          XRL     WMCON,#EEMEN          ; disable EEPROM accesses
          RET

GET_KPAD: MOV     P2,#0FFH
```

```
                MOV     DATA,#0
CHK_COL0:       CLR     COL0
                MOV     A,P2
                ANL     A,#00FH
                CJNE    A,#00FH,COL0_DETECT
                AJMP    CHK_COL1
COL0_DETECT:    MOV     DATA,#01
                AJMP    GET_ROW
CHK_COL1:       SETB    COL0
                CLR     COL1
                MOV     A,P2
                ANL     A,#00FH
                CJNE    A,#00FH,COL1_DETECT
                AJMP    CHK_COL2
COL1_DETECT:    MOV     DATA,#02
                AJMP    GET_ROW
CHK_COL2:       SETB    COL1
                CLR     COL2
                MOV     A,P2
                ANL     A,#00FH
                CJNE    A,#00FH,COL2_DETECT
                MOV     P2,#0FFH
                RET

COL2_DETECT:    MOV     DATA,#03
GET_ROW:        CLR     COL0
                CLR     COL1
                CLR     COL2
                JB      ROW0,CHK_ROW1
                RET
```

```
CHK_ROW1:       JB      ROW1,CHK_ROW2
                MOV     A,DATA
                ADD     A,#3
                MOV     DATA,A
                RET
CHK_ROW2:       JB      ROW2,CHK_ROW3
                MOV     A,DATA
                ADD     A,#6
                MOV     DATA,A
                RET
CHK_ROW3:       MOV     A,DATA
                ADD     A,#9
                MOV     DATA,A
                RET
DELAY:          MOV     R7,#40H
D1:             MOV     R6,#10H
D2:             DJNZ    R6,D2
                DJNZ    R7,D1
                RET
DELAY_1MS:      MOV     R6,#0E6H
DELAY_1MS_1:    NOP
                NOP
                DJNZ    R6,DELAY_1MS_1
                RET
DELAY_10MS:     MOV     R7,#010
DELAY_10MS_1:   MOV     R6,#0E6H
DELAY_10MS_2:   NOP
                NOP
                DJNZ    R6,DELAY_10MS_2
                DJNZ    R7,DELAY_10MS_1
                RET
```

```
DELAY_1S:       MOV     R5,#400
DELAY_1S_1:     ACALL   DELAY_10MS
                DJNZ    R5,DELAY_1S_1
                RET

DSP_BLANK:      DB      01111111B
DSP_NUM1:       DB      11111001B
DSP_NUM2:       DB      10100100B
DSP_NUM3:       DB      10110000B
DSP_NUM4:       DB      10011001B
DSP_NUM5:       DB      10010010B
DSP_NUM6:       DB      10000010B
DSP_NUM7:       DB      11111000B
DSP_NUM8:       DB      10000000B
DSP_NUM9:       DB      10010000B
DSP_STAR:       DB      10001001B
DSP_NUM0:       DB      11000000B
DSP_HASH:       DB      10011100B
                END
                END
```

กรณีไม่มี Safety code "1357" และผู้ใช้สามารถตั้งรหัสได้เองตอนเปิดเครื่องครั้งแรก

```
; ********** VARIABLE SET **********
WMCON     EQU     96H          ; watchdog and memory control register
EEMEN     EQU     00001000b    ; EEPROM access enable bit
EEMWE     EQU     00010000b    ; EEPROM write enable bit
WDTRST    EQU     00000010b    ; EEPROM RDY/BSY bit
DSP1      EQU     P1.3
DSP2      EQU     P1.4
DSP3      EQU     P1.5
DSP4      EQU     P1.6
ROW0      EQU     P2.0
ROW1      EQU     P2.1
ROW2      EQU     P2.2
ROW3      EQU     P2.3
COL2      EQU     P2.4
COL1      EQU     P2.5
COL0      EQU     P2.6
BUF1      EQU     30H
BUF2      EQU     31H
BUF3      EQU     32H
BUF4      EQU     33H
DATA      EQU     34H
PASS      EQU     35H
BUFP1     EQU     36H
BUFP2     EQU     37H
BUFP3     EQU     38H
BUFP4     EQU     39H
SB1       EQU     40H
SB2       EQU     41H
SB3       EQU     42H
SB4       EQU     43H
```

```
SB5         EQU     44H
SB6         EQU     45H
SB7         EQU     46H
SB8         EQU     47H

            ORG     0000H
MAIN:       MOV      R0,#0
            MOV     R1,#0
            MOV     R2,#4
            MOV     R3,#100
            MOV     P0,#0
            MOV     P1,#0
            MOV     P2,#0FFH
            MOV     BUF1,#0
            MOV     BUF2,#0
            MOV     BUF3,#0
            MOV     BUF4,#0
            CLR     P1.0
            SETB    P1.1                ;THE DOOR CLOSE
GET_A:      ACALL   GET_KPAD
            MOV     A,DATA
            JZ      GET_A
            CJNE    A,#0AH,LABEL1
            CALL    CH_PASS
            JMP     GET_A
LABEL1:     CJNE    A,#0CH,LABEL2
            JMP     GET_A
LABEL2:     MOV     DPTR,#DSP_BLANK
            MOVC    A,@A+DPTR
            MOV     BUF1,A
JJ:         CJNE    R0,#0,NEXT1
```

```
                MOV     P0,BUF1
                SETB    DSP1
                CLR     DSP2
                CLR     DSP3
                CLR     DSP4
                ACALL   DELAY
                MOV     R1,P2
                MOV     B,R1
                ANL     B,#00FH
                MOV     R1,B
                CJNE    R1,#00FH,JJ
                ACALL   GET_KPAD
                MOV     A,DATA
                JZ      JJ
                CJNE    A,#0AH,LABEL9
                JMP     JJ
LABEL9:         CJNE    A,#0CH,LABEL10
                JMP     JJ
LABEL10:        INC     R0
NEXT1:          MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR
                CJNE    R0,#1,NEXT2
                MOV     BUF2,A
ZZZ:            MOV     P0,BUF1
                SETB    DSP1
                CLR     DSP2
                ACALL   DELAY
                CLR     DSP1
                MOV     P0,BUF2
                SETB    DSP2
                ACALL   DELAY
```

```
                MOV     R1,P2
                MOV     B,R1
                ANL     B,#00FH
                MOV     R1,B
                CJNE    R1,#00FH,ZZZ
                ACALL   GET_KPAD
                MOV     A,DATA
                JZ      ZZZ
                CJNE    A,#0AH,LABEL3
                JMP     ZZZ
LABEL3:         CJNE    A,#0CH,LABEL4
                JMP     ZZZ
LABEL4:         INC     R0
NEXT2:          CJNE    R0,#2,NEXT3
                MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR
                MOV     BUF3,A
ZZZ2:           MOV     P0,BUF1
                SETB    DSP1
                CLR     DSP2
                CLR     DSP3
                ACALL   DELAY
                CLR     DSP1
                MOV     P0,BUF2
                SETB    DSP2
                ACALL   DELAY
                CLR     DSP1
                CLR     DSP2
                MOV     P0,BUF3
                SETB    DSP3
                ACALL   DELAY
```

```
                MOV     R1,P2
                MOV     B,R1
                ANL     B,#00FH
                MOV     R1,B
                CJNE    R1,#00FH,ZZZ2
                ACALL   GET_KPAD
                MOV     A,DATA
                JZ      ZZZ2
                CJNE    A,#0AH,LABEL5
                JMP     ZZZ2
LABEL5:         CJNE    A,#0CH,LABEL6
                JMP     ZZZ2
LABEL6:         INC     R0
NEXT3:          CJNE    R0,#3,UUU
                MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR
                MOV     BUF4,A
ZZZ3:           MOV     P0,BUF1
                SETB    DSP1
                CLR     DSP2
                CLR     DSP3
                CLR     DSP4
                ACALL   DELAY
                CLR     DSP1
                MOV     P0,BUF2
                SETB    DSP2
                ACALL   DELAY
                CLR     DSP2
                MOV     P0,BUF3
                SETB    DSP3
                ACALL   DELAY
```

```
                CLR     DSP3
                MOV     P0,BUF4
                SETB    DSP4
                ACALL   DELAY
                MOV     R1,P2
                MOV     B,R1
                ANL     B,#00FH
                MOV     R1,B
                CJNE    R1,#00FH,ZZZ3
                ACALL   GET_KPAD
                MOV     A,DATA
                JZ      ZZZ3
                CJNE    A,#0AH,LABEL7
                JMP     ZZZ3
LABEL7:         CJNE    A,#0CH,ZZZ3
                CALL    ENTER
                MOV     R0,#0
                MOV     P1,#00000010b
UUU:            LJMP    GET_A
ENTER:          MOV     DPTR,#0000H
                CALL    ReadEE
                MOV     BUFP1,A
                MOV     DPTR,#0001H
                CALL    ReadEE
                MOV     BUFP2,A
                MOV     DPTR,#0002H
                CALL    ReadEE
                MOV     BUFP3,A
                MOV     DPTR,#0003H
                CALL    ReadEE
                MOV     BUFP4,A
```

```
                JMP         CODEPASS
SETNEWPASSWORD:
                MOV         P0,#7FH                 ;PRINT ....
                MOV         P1,#01111010B
SNP1:           ACALL       GET_KPAD
                MOV         A,DATA
                JZ          SNP1
                CJNE        A,#0AH,LABEL11
                JMP         SNP1
LABEL11:        CJNE        A,#0CH,LABEL12
                JMP         SNP1
LABEL12:        MOV         DPTR,#DSP_BLANK
                MOVC        A,@A+DPTR
                MOV         P1,#01110010B
                MOV         PASS,A          ;
                MOV         DPTR,#0000H   ; WRITE DATA TO ADDRESS 0000H
                CALL        WriteEE
                CALL        CH_KEY
SNP2:           ACALL       GET_KPAD
                MOV         A,DATA
                JZ          SNP2
                CJNE        A,#0AH,LABEL13
                JMP         SNP2
LABEL13:        CJNE        A,#0CH,LABEL14
                JMP         SNP2
LABEL14:        MOV         DPTR,#DSP_BLANK
                MOVC        A,@A+DPTR
                MOV         P1,#01100010B
                MOV         PASS,A          ;
                MOV         DPTR,#0001H   ; WRITE DATA TO ADDRESS 0001H
```

```
                CALL    WriteEE
                CALL    CH_KEY
SNP3:           ACALL   GET_KPAD
                MOV     A,DATA
                JZ      SNP3
                CJNE    A,#0AH,LABEL15
                JMP     SNP3
LABEL15         CJNE    A,#0CH,LABEL16
                JMP     SNP3
LABEL16:        MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR
                MOV     P1,#01000010B
                MOV     PASS,A          ;
                MOV     DPTR,#0002H   ; WRITE DATA TO ADDRESS 0002H
                CALL    WriteEE
                CALL    CH_KEY
SNP4:           ACALL   GET_KPAD
                MOV     A,DATA
                JZ      SNP4
                CJNE    A,#0AH,LABEL17
                JMP     SNP4
LABEL17:        CJNE    A,#0CH,LABEL18
                JMP     SNP4
LABEL18:        MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR
                MOV     P1,#00000010B
                MOV     PASS,A          ;
                MOV     DPTR,#0003H   ; WRITE DATA TO ADDRESS 0003H
                CALL    WriteEE
                CALL    CH_KEY
```

```
                MOV     P0,#0BFH            ;PRINT ----
                MOV     P1,#01111010B
                CALL    DELAY_1S
                JMP     FL_PASS

CODEPASS:       MOV     A,BUFP1
                CJNE    A,BUF1,FL_PASS
                MOV     A,BUFP2
                CJNE    A,BUF2,FL_PASS
                MOV     A,BUFP3
                CJNE    A,BUF3,FL_PASS
                MOV     A,BUFP4
                CJNE    A,BUF4,FL_PASS
TR_PASS:        MOV     BUF1,#0
                MOV     BUF2,#0
                MOV     BUF3,#0
                MOV     BUF4,#0
                MOV     P0,#0BFH
                MOV     P1,#01111010B
                SETB    P1.0                ;THE DOOR OPEN
                CLR     P1.1
                CALL    DELAY_1S
                CALL    DELAY_1S
                CLR     P1.0
                SETB    P1.1                ;THE DOOR CLOSE
FL_PASS:        MOV     P0,#0FFH
                RET

WR_PASS:
                MOV     P1,#72H
                CALL    CH_KEY
```

```
ww:         CALL    GET_KEY
            CJNE    A,#0AH,LABEL36
            JMP     ww
LABEL36:    CJNE    A,#0CH,LABEL31
            JMP     ww
LABEL31:    MOV     P1,#62H
            CALL    CH_KEY
ww1:        CALL    GET_KEY
            CJNE    A,#0AH,LABEL32
            JMP     ww1
LABEL32:    CJNE    A,#0CH,LABEL33
            JMP     ww1
LABEL33:    MOV     P1,#42H
            CALL    CH_KEY
ww2:        CALL    GET_KEY
            CJNE    A,#0AH,LABEL34
            JMP     ww2
LABEL34:    CJNE    A,#0CH,LABEL35
            JMP     ww2
LABEL35:    MOV     P1,#02H
            CALL    CH_KEY
            LJMP    GET_A
CH_PASS:    CALL    CH_KEY
            MOV     P0,#01111111B
            MOV     P1,#01111010B
            MOV     DPTR,#0005H
            CALL    ReadEE
            MOV     SB1,A
            MOV     DPTR,#0006H
            CALL    ReadEE
            MOV     SB2,A
```

```
                MOV     DPTR,#0007H
                CALL    ReadEE
                MOV     SB3,A
                MOV     DPTR,#0008H
                CALL    ReadEE
                MOV     SB4,A
                CJNE    SB1,#00H,KK
                CJNE    SB2,#00H,KK
                CJNE    SB3,#00H,KK
                CJNE    SB4,#00H,KK
                JMP     PP4
KK:             MOV     DPTR,#0000H
                CALL    ReadEE
                MOV     BUF1,A
                MOV     DPTR,#0001H
                CALL    ReadEE
                MOV     BUF2,A
                MOV     DPTR,#0002H
                CALL    ReadEE
                MOV     BUF3,A
                MOV     DPTR,#0003H
                CALL    ReadEE
                MOV     BUF4,A
pp:             CALL    GET_KEY     ;GET KEY AND CHECK OLD PASSWORD
                CJNE    A,#0AH,LABEL40
                JMP     pp
LABEL40:        CJNE    A,#0CH,LABEL41
                JMP     pp
label41:        MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR
```

```
tra:        CJNE    A,BUF1,WR_PASS
            MOV     P1,#70H
            CALL    CH_KEY
pp1:        CALL    GET_KEY
            CJNE    A,#0AH,LABEL42
            JMP     pp1
LABEL42:    CJNE    A,#0CH,LABEL43
            JMP     pp1
LABEL43:    MOV     DPTR,#DSP_BLANK
            MOVC    A,@A+DPTR
            CJNE    A,BUF2,tra
            MOV     P1,#60H
            CALL    CH_KEY
pp2:        CALL    GET_KEY
            CJNE    A,#0AH,LABEL44
            JMP     pp2
LABEL44:    CJNE    A,#0CH,LABEL45
            JMP     pp2
LABEL45:    MOV     DPTR,#DSP_BLANK
            MOVC    A,@A+DPTR
            CJNE    A,BUF3,tra
            MOV     P1,#40H
            CALL    CH_KEY
pp3:        CALL    GET_KEY
            CJNE    A,#0AH,LABEL21
            JMP     pp3
LABEL21:    CJNE    A,#0CH,LABEL22
            JMP     pp3
LABEL22:    MOV     DPTR,#DSP_BLANK
            MOVC    A,@A+DPTR
            CJNE    A,BUF4,tra
```

```
                MOV     P1,#00H
                CALL    CH_KEY
                MOV     P0,#7FH
                MOV     P1,#78H                 ;END
pp4:            CALL    GET_KEY                 ;GET NEW PASSWORD
                CJNE    A,#0AH,LABEL23
                JMP     pp4
LABEL23:        CJNE    A,#0CH,LABEL24
                JMP     pp4
LABEL24:        MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR
                MOV     DPTR,#0000H
                MOV     SB5,A
                MOV     PASS,A
                CALL    WriteEE
                MOV     P1,#70H
                CALL    CH_KEY
pp5:            CALL    GET_KEY
                CJNE    A,#0AH,LABEL25
                JMP     pp5
LABEL25:        CJNE    A,#0CH,LABEL26
                JMP     pp5
LABEL26:        MOV     DPTR,#DSP_BLANK
                MOVC    A,@A+DPTR
                MOV     DPTR,#0001H
                MOV     SB6,A
                MOV     PASS,A
                CALL    WriteEE
                MOV     P1,#60H
                CALL    CH_KEY
```

```
pp6:        CALL    GET_KEY
            CJNE    A,#0AH,LABEL27
            JMP     pp6
LABEL27:    CJNE    A,#0CH,LABEL28
            JMP     pp6
LABEL28:    MOV     DPTR,#DSP_BLANK
            MOVC    A,@A+DPTR
            MOV     DPTR,#0002H
            MOV     SB7,A
            MOV     PASS,A
            CALL    WriteEE
            MOV     P1,#40H
            CALL    CH_KEY
pp7:        CALL    GET_KEY
            CJNE    A,#0AH,LABEL29
            JMP     pp7
LABEL29:    CJNE    A,#0CH,LABEL30
            JMP     pp7
label30:    MOV     DPTR,#DSP_BLANK
            MOVC    A,@A+DPTR
            MOV     DPTR,#0003H
            MOV     SB8,A
            MOV     PASS,A
            CALL    WriteEE
            MOV     P1,#00H
            CALL    CH_KEY                  ;END
            CJNE    SB1,#00H,HHH
            CJNE    SB2,#00H,HHH
            CJNE    SB3,#00H,HHH
            CJNE    SB4,#00H,HHH
            MOV     DPTR,#0005H
```

```
        MOV     PASS,SB5
        CALL    WriteEE
        MOV     DPTR,#0006H
        MOV     PASS,SB6
        CALL    WriteEE
        MOV     DPTR,#0007H
        MOV     PASS,SB7
        CALL    WriteEE
        MOV     DPTR,#0008H
        MOV     PASS,SB8
        CALL    WriteEE
HHH:    MOV     P0,#0BFH
        MOV     P1,#01111010B
        RET
GET_KEY:
GET:    CALL    GET_KPAD
        MOV     A,DATA
        JZ      GET
        RET

CH_KEY:
CH_K:   MOV     R1,P2
        MOV     B,R1
        ANL     B,#00FH
        MOV     R1,B
        CJNE    R1,#00FH,CH_K
        RET

WriteEE:
        ORL     WMCON,#EEMEN    ; enable EEPROM accesses
        ORL     WMCON,#EEMWE    ; enable EEPROM writes
```

```
                MOV     A,PASS          ; data to write
                MOVX    @DPTR,A         ; write EEPROM
Loop:           MOV     A,WMCON         ; get EEPROM write status
                ANL     A,#WDTRST       ; check RDY/BSY
                JZ      Loop            ;jump if busy
                MOVX    A,@DPTR         ; read EEPROM
                CJNE    A,PASS,WriteEE  ; jump if data compare fails
                XRL     WMCON,#EEMWE    ; disable EEPROM writes
                XRL     WMCON,#EEMEN    ; disable EEPROM accesses
                RET

ReadEE:         ORL     WMCON,#EEMEN        ; enable EEPROM accesses
                MOVX    A,@DPTR             ; read EEPROM
                XRL     WMCON,#EEMEN        ; disable EEPROM accesses
                RET

GET_KPAD:       MOV     P2,#0FFH
                MOV     DATA,#0
CHK_COL0:       CLR     COL0
                MOV     A,P2
                ANL     A,#00FH
                CJNE    A,#00FH,COL0_DETECT
                AJMP    CHK_COL1
COL0_DETECT:
                MOV     DATA,#01
                AJMP    GET_ROW
CHK_COL1:       SETB    COL0
                CLR     COL1
                MOV     A,P2
                ANL     A,#00FH
                CJNE    A,#00FH,COL1_DETECT
```

```
                AJMP        CHK_COL2
COL1_DETECT:
                MOV         DATA,#02
                AJMP        GET_ROW
CHK_COL2:       SETB        COL1
                CLR         COL2
                MOV         A,P2
                ANL         A,#00FH
                CJNE        A,#00FH,COL2_DETECT
                MOV         P2,#0FFH
                RET

COL2_DETECT:
                MOV         DATA,#03
GET_ROW:        CLR         COL0
                CLR         COL1
                CLR         COL2
                JB          ROW0,CHK_ROW1
                RET

CHK_ROW1:       JB          ROW1,CHK_ROW2
                MOV         A,DATA
                ADD         A,#3
                MOV         DATA,A
                RET

CHK_ROW2:       JB          ROW2,CHK_ROW3
                MOV         A,DATA
                ADD         A,#6
                MOV         DATA,A
                RET
```

```
CHK_ROW3:   MOV     A,DATA
            ADD     A,#9
            MOV     DATA,A
            RET

DELAY:      MOV     R7,#40H
D1:         MOV     R6,#10H
D2:         DJNZ    R6,D2
            DJNZ    R7,D1
            RET

DELAY_1MS:  MOV     R6,#0E6H
DELAY_1MS_1:
            NOP
            NOP
            DJNZ    R6,DELAY_1MS_1
            RET

DELAY_10MS:
            MOV     R7,#010
DELAY_10MS_1:
            MOV     R6,#0E6H
DELAY_10MS_2:
            NOP
            NOP
            DJNZ    R6,DELAY_10MS_2
        DJNZ        R7,DELAY_10MS_1
        RET

DELAY_1S:   MOV     R5,#400
DELAY_1S_1: ACALL   DELAY_10MS
```

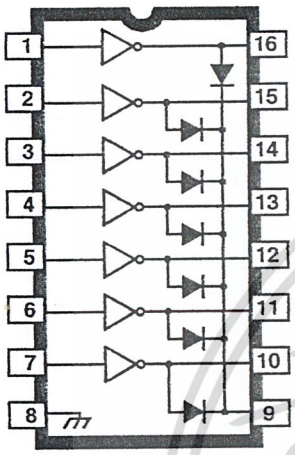
```
        DJNZ        R5,DELAY_1S_1
        RET

DSP_BLANK:  DB      01111111B
DSP_NUM1:   DB      11111001B
DSP_NUM2:   DB      10100100B
DSP_NUM3:   DB      10110000B
DSP_NUM4:   DB      10011001B
DSP_NUM5:   DB      10010010B
DSP_NUM6:   DB      10000010B
DSP_NUM7:   DB      11111000B
DSP_NUM8:   DB      10000000B
DSP_NUM9:   DB      10010000B
DSP_STAR:   DB      10001001B
DSP_NUM0:   DB      11000000B
DSP_HASH:   DB      10011100B
        END
        END
```

2003 THRU 2024

Data Sheet 29304F

HIGH-VOLTAGE, HIGH-CURRENT DARLINGTON ARRAYS



Dwg. No. A-9594

Note that the ULN20xxA series (dual in-line package) and ULN20xxL series (small-outline IC package) are electrically identical and share a common terminal number assignment.

ABSOLUTE MAXIMUM RATINGS

Output Voltage, V_{CE}
(ULN200xA and ULN200xL) ..... 50 V
(ULN202xA and ULN202xL) ..... 95 V
Input Voltage, V_{IN} ..... 30 V
Continuous Output Current, I_C ..... 500 mA
Continuous Input Current, I_{IN} ..... 25 mA
Power Dissipation, P_D
(one Darlington pair) ..... 1.0 W
(total package) ..... See Graph
Operating Temperature Range, T_A ..... -20°C to +85°C
Storage Temperature Range, T_S ..... -55°C to +150°C

Ideally suited for interfacing between low-level logic circuitry and multiple peripheral power loads, the Series ULN20xxA/L high-voltage, high-current Darlington arrays feature continuous load current ratings to 500 mA for each of the seven drivers. At an appropriate duty cycle depending on ambient temperature and number of drivers turned ON simultaneously, typical power loads totaling over 230 W (350 mA x 7, 95 V) can be controlled. Typical loads include relays, solenoids, stepping motors, magnetic print hammers, multiplexed LED and incandescent displays, and heaters. All devices feature open-collector outputs with integral clamp diodes.

The ULN2003A/L and ULN2023A/L have series input resistors selected for operation directly with 5 V TTL or CMOS. These devices will handle numerous interface needs — particularly those beyond the capabilities of standard logic buffers.

The ULN2004A/L and ULN2024A/L have series input resistors for operation directly from 6 to 15 V CMOS or PMOS logic outputs.

The ULN2003A/L and ULN2004A/L are the standard Darlington arrays. The outputs are capable of sinking 500 mA and will withstand at least 50 V in the OFF state. Outputs may be paralleled for higher load current capability. The ULN2023A/L and ULN2024A/L will withstand 95 V in the OFF state.

These Darlington arrays are furnished in 16-pin dual in-line plastic packages (suffix "A") and 16-lead surface-mountable SOICs (suffix "L"). All devices are pinned with outputs opposite inputs to facilitate ease of circuit board layout. All devices are rated for operation over the temperature range of -20°C to +85°C. Most (see matrix, next page) are also available for operation to -40°C; to order, change the prefix from "ULN" to "ULQ".

FEATURES

- TTL, DTL, PMOS, or CMOS-Compatible Inputs
- Output Current to 500 mA
- Output Voltage to 95 V
- Transient-Protected Outputs
- Dual In-Line Plastic Package or Small-Outline IC Package

x = digit to identify specific device. Characteristic shown applies to family of devices with remaining digits as shown. See matrix on next page.

2003 THRU 2024
HIGH-VOLTAGE, HIGH-CURRENT DARLINGTON ARRAYS

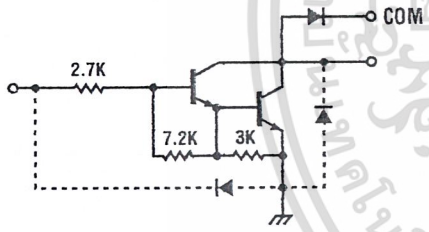
DEVICE PART NUMBER DESIGNATION

$V_{CE(MAX)}$	50 V	95 V
$I_{C(MAX)}$	500 mA	500 mA
Logic	**Part Number**	
5V TTL, CMOS	ULN2003A* ULN2003L*	ULN2023A* ULN2023L
6-15 V CMOS, PMOS	ULN2004A* ULN2004L*	ULN2024A ULN2024L

*Also available for operation between -40°C and +85°C. To order, change prefix from "ULN" to "ULQ".

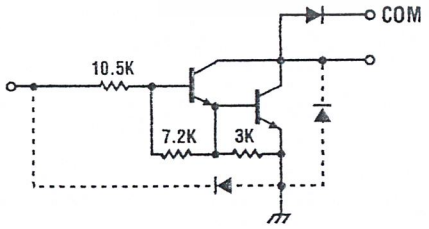
PARTIAL SCHEMATICS

ULN20x3A/L (Each Driver)

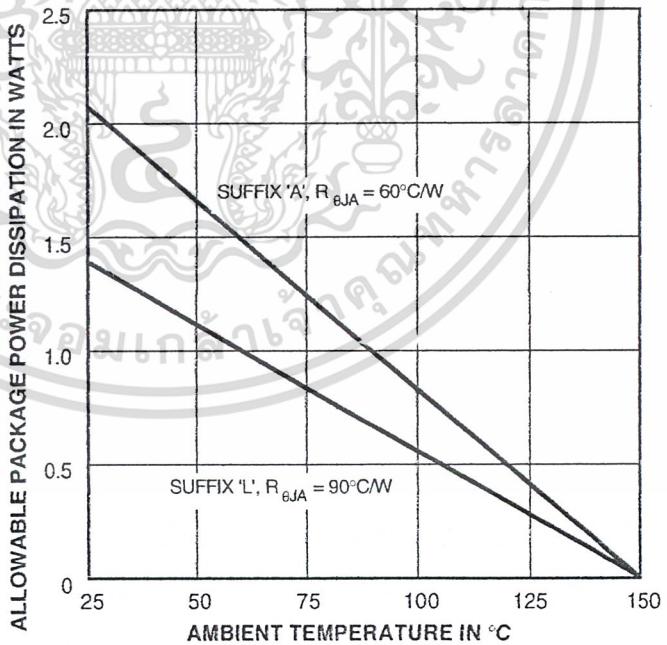


Dwg. No. A-9651

ULN20x4A/L (Each Driver)



Dwg. No. A-9898A



Dwg. GP-006A

X = Digit to identify specific device. Specification shown applies to family of devices with remaining digits as shown. See matrix above.

Allegro® MicroSystems, inc.
115 Northeast Cutoff, Box 15036
Worcester, Massachusetts 01615-0036 (508) 853-5000

Types ULN2003A, ULN2003L, ULN2004A, and ULN2004L ELECTRICAL CHARACTERISTICS at +25°C (unless otherwise noted).

Characteristic	Symbol	Test Fig.	Applicable Devices	Test Conditions	Limits			
					Min.	Typ.	Max.	Units
Output Leakage Current	I_{CEX}	1A	All	V_{CE} = 50 V, T_A = 25°C	—	< 1	50	µA
				V_{CE} = 50 V, T_A = 70°C	—	< 1	100	µA
		1B	ULN2004A/L	V_{CE} = 50 V, T_A = 70°C, V_{IN} = 1.0 V	—	< 5	500	µA
Collector-Emitter Saturation Voltage	$V_{CE(SAT)}$	2	All	I_C = 100 mA, I_B = 250 µA	—	0.9	1.1	V
				I_C = 200 mA, I_B = 350 µA	—	1.1	1.3	V
				I_C = 350 mA, I_B = 500 µA	—	1.3	1.6	V
Input Current	$I_{IN(ON)}$	3	ULN2003A/L	V_{IN} = 3.85 V	—	0.93	1.35	mA
			ULN2004A/L	V_{IN} = 5.0 V	—	0.35	0.5	mA
				V_{IN} = 12 V	—	1.0	1.45	mA
	$I_{IN(OFF)}$	4	All	I_C = 500 µA, T_A = 70°C	50	65	—	µA
Input Voltage	$V_{IN(ON)}$	5	ULN2003A/L	V_{CE} = 2.0 V, I_C = 200 mA	—	—	2.4	V
				V_{CE} = 2.0 V, I_C = 250 mA	—	—	2.7	V
				V_{CE} = 2.0 V, I_C = 300 mA	—	—	3.0	V
			ULN2004A/L	V_{CE} = 2.0 V, I_C = 125 mA	—	—	5.0	V
				V_{CE} = 2.0 V, I_C = 200 mA	—	—	6.0	V
				V_{CE} = 2.0 V, I_C = 275 mA	—	—	7.0	V
				V_{CE} = 2.0 V, I_C = 350 mA	—	—	8.0	V
Input Capacitance	C_{IN}	—	All		—	15	25	pF
Turn-On Delay	t_{PLH}	8	All	0.5 E_{IN} to 0.5 E_{OUT}	—	0.25	1.0	µs
Turn-Off Delay	t_{PHL}	8	All	0.5 E_{IN} to 0.5 E_{OUT}	—	0.25	1.0	µs
Clamp Diode Leakage Current	I_R	6	All	V_R = 50 V, T_A = 25°C	—	—	50	µA
				V_R = 50 V, T_A = 70°C	—	—	100	µA
Clamp Diode Forward Voltage	V_F	7	All	I_F = 350 mA	—	1.7	2.0	V

Complete part number includes suffix to identify package style: A = DIP, L = SOIC.

2003 THRU 2024
HIGH-VOLTAGE, HIGH-CURRENT DARLINGTON ARRAYS

Types ULN2023A, ULN2023L, ULN2024A, and ULN2024L
ELECTRICAL CHARACTERISTICS at +25°C (unless otherwise noted).

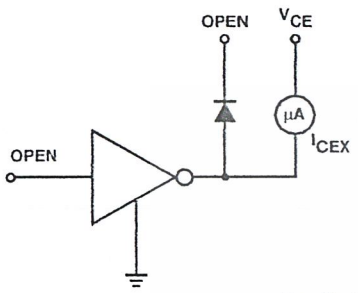
Characteristic	Symbol	Test Fig.	Applicable Devices	Test Conditions	Limits			
					Min.	Typ.	Max.	Units
Output Leakage Current	I_{CEX}	1A	All	V_{CE} = 95 V, T_A = 25°C	—	< 1	50	µA
				V_{CE} = 95 V, T_A = 70°C	—	< 1	100	µA
		1B	ULN2024A/L	V_{CE} = 95 V, T_A = 70°C, V_{IN} = 1.0 V	—	< 5	500	µA
Collector-Emitter Saturation Voltage	$V_{CE(SAT)}$	2	All	I_C = 100 mA, I_B = 250 µA	—	0.9	1.1	V
				I_C = 200 mA, I_B = 350 µA	—	1.1	1.3	V
				I_C = 350 mA, I_B = 500 µA	—	1.3	1.6	V
Input Current	$I_{IN(ON)}$	3	ULN2023A/L	V_{IN} = 3.85 V	—	0.93	1.35	mA
			ULN2024A/L	V_{IN} = 5.0 V	—	0.35	0.5	mA
				V_{IN} = 12 V	—	1.0	1.45	mA
	$I_{IN(OFF)}$	4	All	I_C = 500 µA, T_A = 70°C	50	65	—	µA
Input Voltage	$V_{IN(ON)}$	5	ULN2023A/L	V_{CE} = 2.0 V, I_C = 200 mA	—	—	2.4	V
				V_{CE} = 2.0 V, I_C = 250 mA	—	—	2.7	V
				V_{CE} = 2.0 V, I_C = 300 mA	—	—	3.0	V
			ULN2024A/L	V_{CE} = 2.0 V, I_C = 125 mA	—	—	5.0	V
				V_{CE} = 2.0 V, I_C = 200 mA	—	—	6.0	V
				V_{CE} = 2.0 V, I_C = 275 mA	—	—	7.0	V
				V_{CE} = 2.0 V, I_C = 350 mA	—	—	8.0	V
Input Capacitance	C_{IN}	—	All		—	15	25	pF
Turn-On Delay	t_{PLH}	8	All	0.5 E_{IN} to 0.5 E_{OUT}	—	0.25	1.0	µs
Turn-Off Delay	t_{PHL}	8	All	0.5 E_{IN} to 0.5 E_{OUT}	—	0.25	1.0	µs
Clamp Diode Leakage Current	I_R	6	All	V_R = 95 V, T_A = 25°C	—	—	50	µA
				V_R = 95 V, T_A = 70°C	—	—	100	µA
Clamp Diode Forward Voltage	V_F	7	All	I_F = 350 mA	—	1.7	2.0	V

Complete part number includes suffix to identify package style: A = DIP, L = SOIC.

Allegro® MicroSystems, Inc.
115 Northeast Cutoff, Box 15036
Worcester, Massachusetts 01615-0036 (508) 853-5000

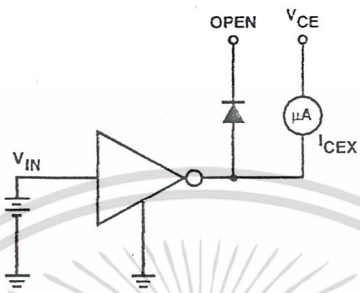
TEST FIGURES

FIGURE 1A



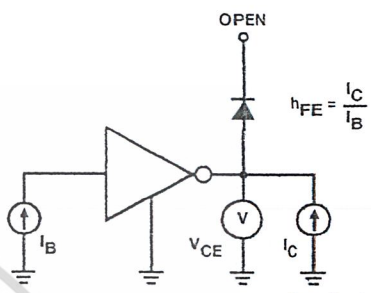
Dwg. No. A-9729A

FIGURE 1B



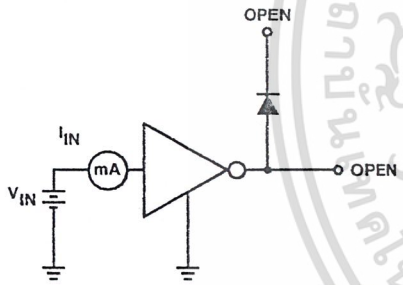
Dwg. No. A-9730A

FIGURE 2



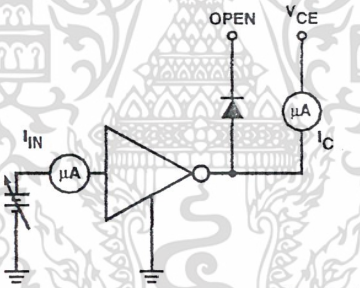
Dwg. No. A-9731A

FIGURE 3



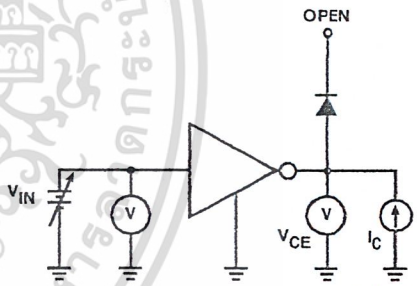
Dwg. No. A-9732A

FIGURE 4



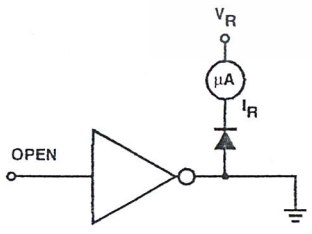
Dwg. No. A-9733A

FIGURE 5



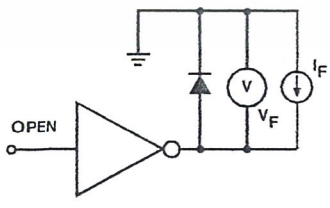
Dwg. No. A-9734A

FIGURE 6



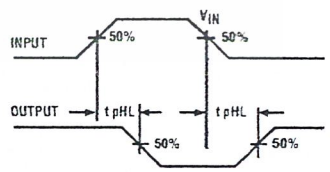
Dwg. No. A-9735A

FIGURE 7

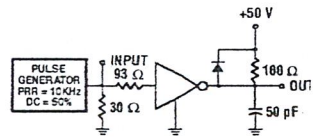


Dwg. No. A-9736A

FIGURE 8



	V_{in}
ULN20X3*	3.5 V
ULN20X4*	12 V



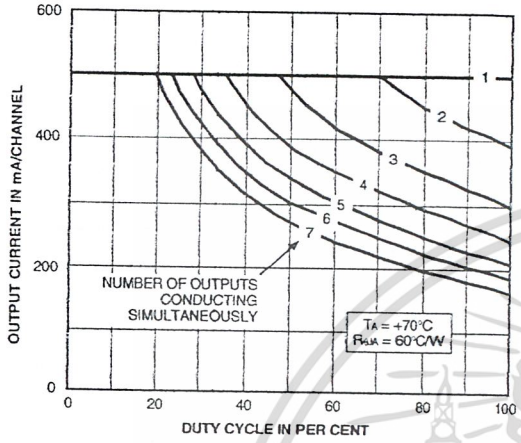
* Complete part number includes a final letter to indicate package.

X = Digit to identify specific device. Specification shown applies to family of devices with remaining digits as shown.

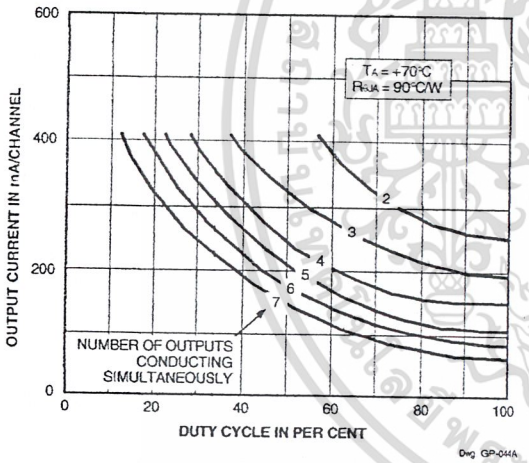
2003 THRU 2024
HIGH-VOLTAGE, HIGH-CURRENT DARLINGTON ARRAYS

ALLOWABLE COLLECTOR CURRENT AS A FUNCTION OF DUTY CYCLE

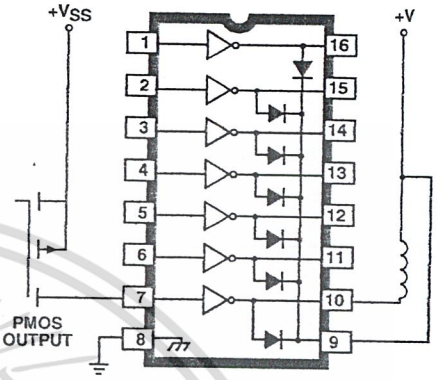
(Dual In-line-Packaged Devices, Suffix 'A')



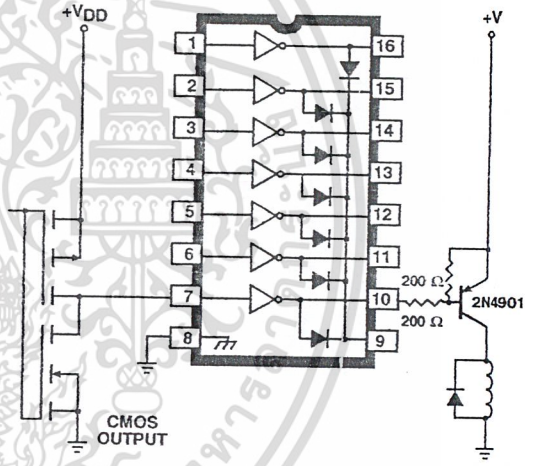
(Small-Outline-Packaged Devices, Suffix 'L')



TYPICAL APPLICATIONS

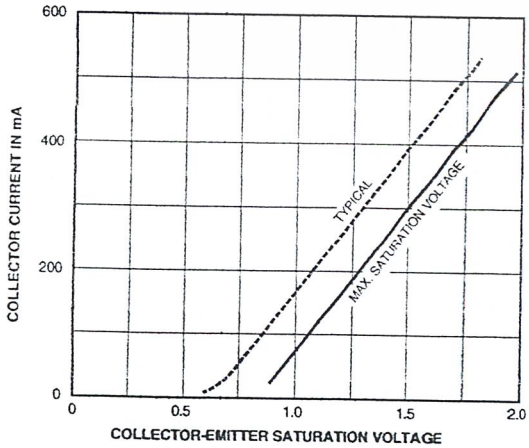


Dwg. No. A-9652

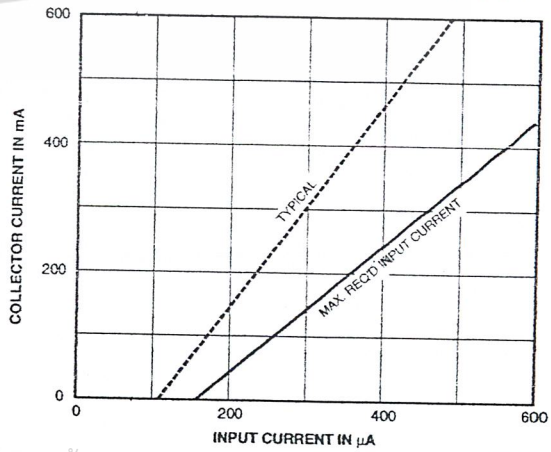


Dwg. No. A-9654A

SATURATION VOLTAGE AS A FUNCTION OF COLLECTOR CURRENT

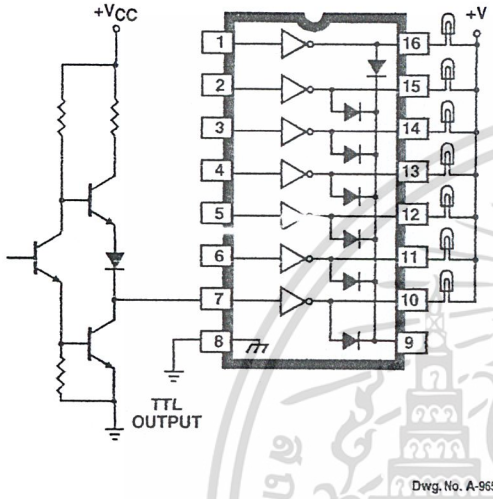


COLLECTOR CURRENT AS A FUNCTION OF INPUT CURRENT

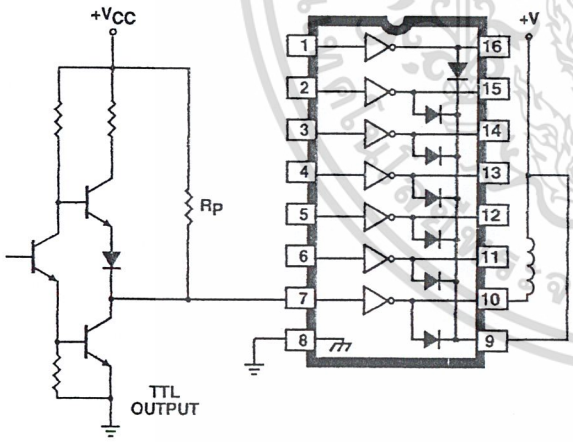


Allegro MicroSystems, Inc.
115 Northeast Cutoff, Box 15036
Worcester, Massachusetts 01615-0036 (508) 853-5000

TYPICAL APPLICATIONS



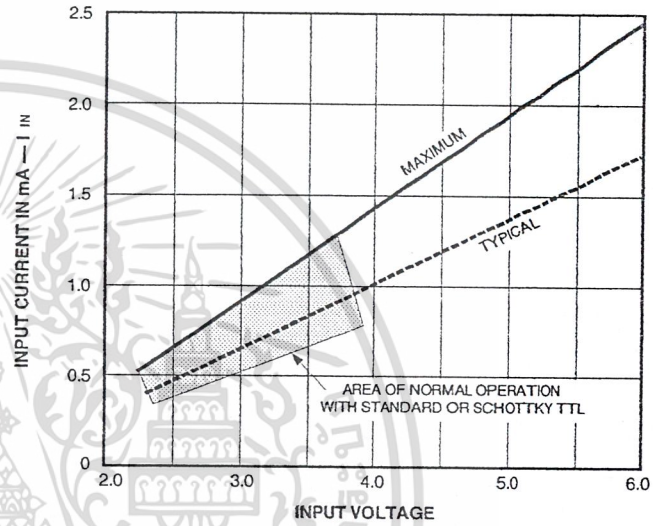
Dwg. No. A-9653A



Dwg. No. A-10,175

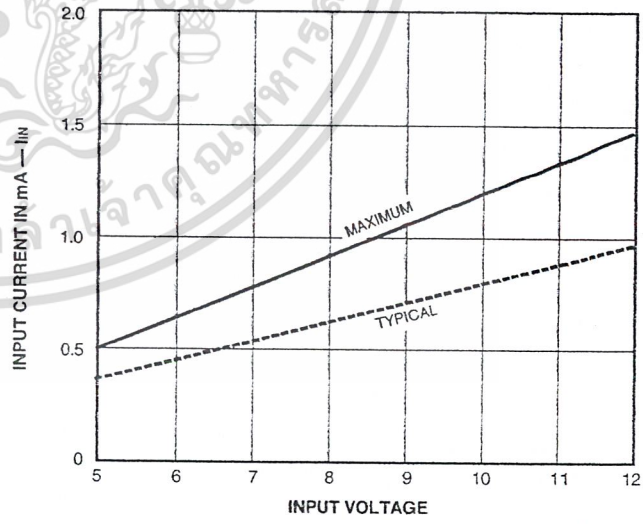
INPUT CURRENT AS A FUNCTION OF INPUT VOLTAGE

Types ULN2003A, ULN2003L, ULN2023A, and ULN2023L



Dwg. GP-069

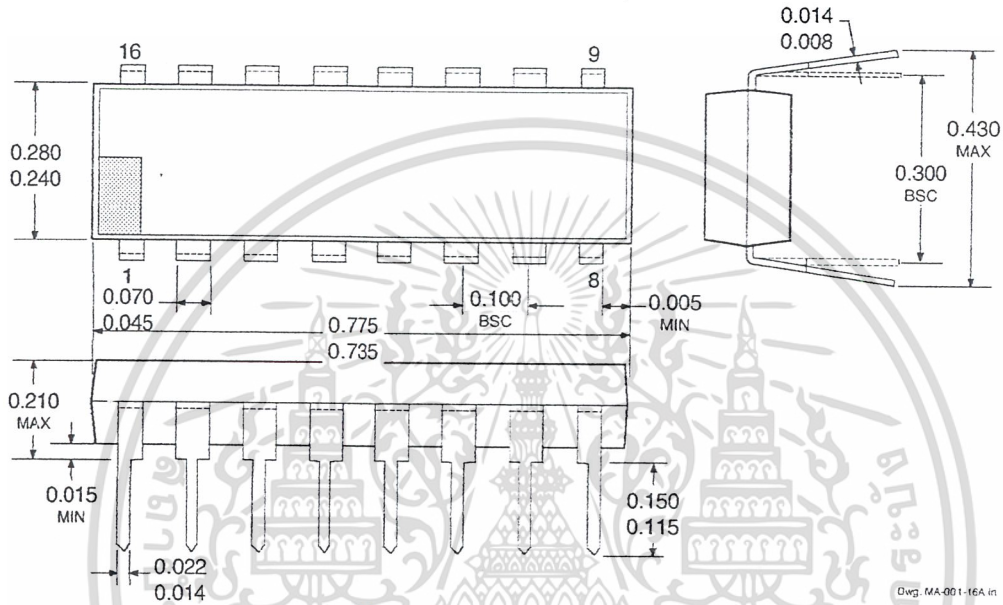
Types ULN2004A, ULN2004L, ULN2024A, and ULN2024L



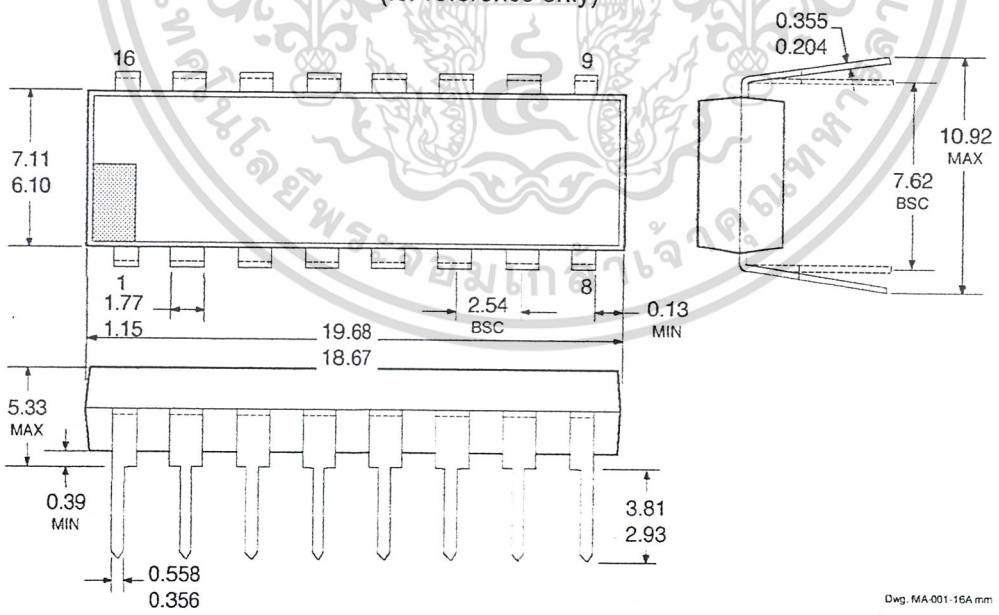
Dwg. GP-069-1

PACKAGE DESIGNATOR "A"

Dimensions in Inches
(controlling dimensions)



Dimension in Millimeters
(for reference only)

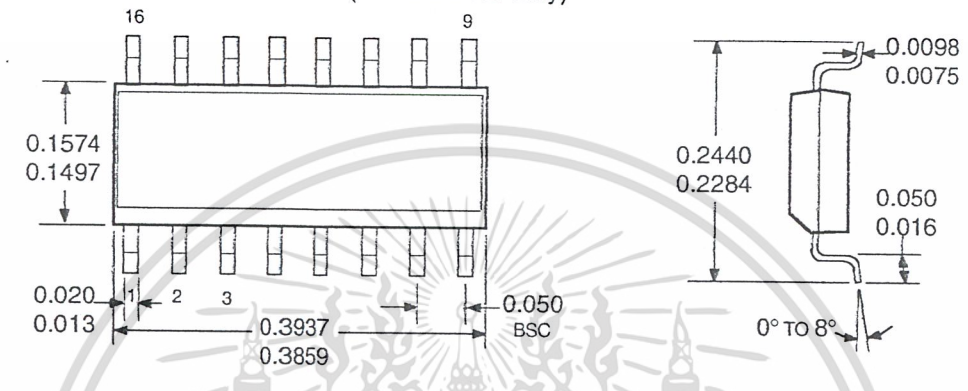


NOTES: 1. Leads 1, 8, 9, and 16 may be half leads at vendor's option.
2. Lead thickness is measured at seating plane or below.
3. Lead spacing tolerance is non-cumulative.
4. Exact body and lead configuration at vendor's option within limits shown.

Allegro MicroSystems, Inc.
115 Northeast Cutoff, Box 15036
Worcester, Massachusetts 01615-0036 (508) 853-5000

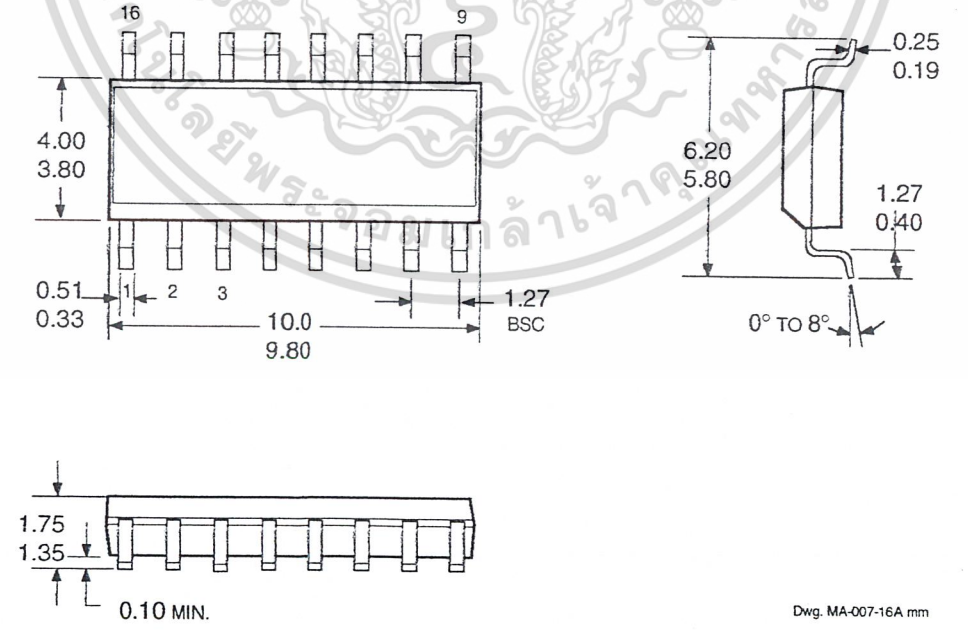
PACKAGE DESIGNATOR "L"

Dimensions in Inches
(for reference only)



Dwg. MA-007-16 in

Dimension in Millimeters
(controlling dimensions)



Dwg. MA-007-16A mm

NOTES: 1. Lead spacing tolerance is non-cumulative.
2. Exact body and lead configuration at vendor's option within limits shown.

AT89S8252 Primer

Introduction

The Atmel AT89S8252 microcontroller is a low-power, high-performance device featuring 8K bytes of Flash memory (CMOS PEROM), 2K bytes of EEPROM, and a Serial Peripheral Interface (SPI). The Flash and EEPROM memories may be reprogrammed in-system via the SPI. The EEPROM provides applications with re-writable, nonvolatile data storage. These features, and others, are described in the text which follows. Code samples are provided. Additional information on the AT89S8252 microcontroller may be found in the data sheet and relevant sections in the Atmel AT89-series Microcontroller Databook.

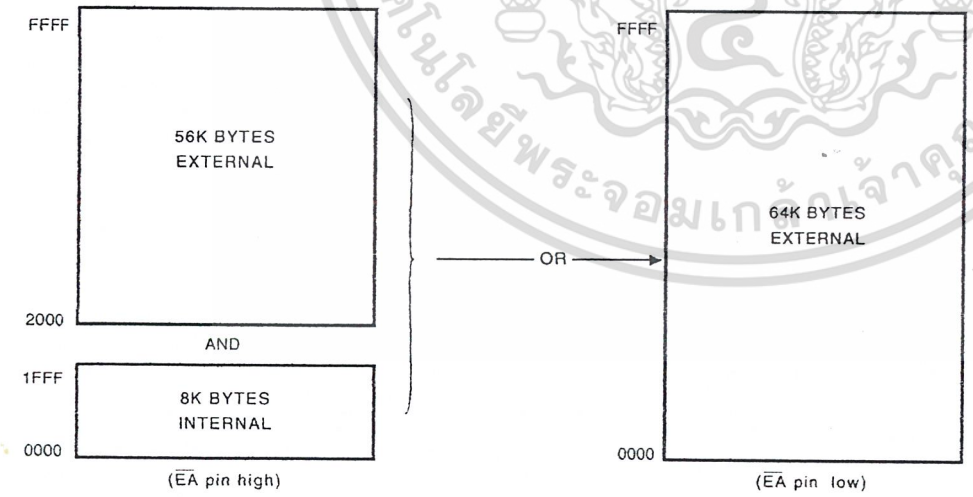
AT89S8252 Memory Organization

Program Memory

The AT89S8252 has separate address spaces for program memory and data memory. Figure 1 shows two alternate maps of program memory.

Program memory is read-only: the microcontroller generates no write signals for program memory. Depending on the state of the $\overline{EA}$ pin, program memory may consist of 8K bytes of internal Flash memory supplemented by up to 56K bytes of external memory, or may consist entirely of up to 64K bytes of external memory. The 8K bytes of internal Flash memory are accessed at addresses 0000H-1FFFH. Program memory accesses at addresses 2000H-FFFFH always access external memory.

Figure 1. The AT89S8252 Program Memory



AT89S8252 Primer

Application Note

Rev. 1018A–03/98

Data Memory

Figure 2 shows a map of AT89S8252 data memory, which consists of 256 bytes of internal RAM, the Special Function Registers (SFRs), 2K bytes of on-chip EEPROM and, optionally, up to 64K bytes of external memory.

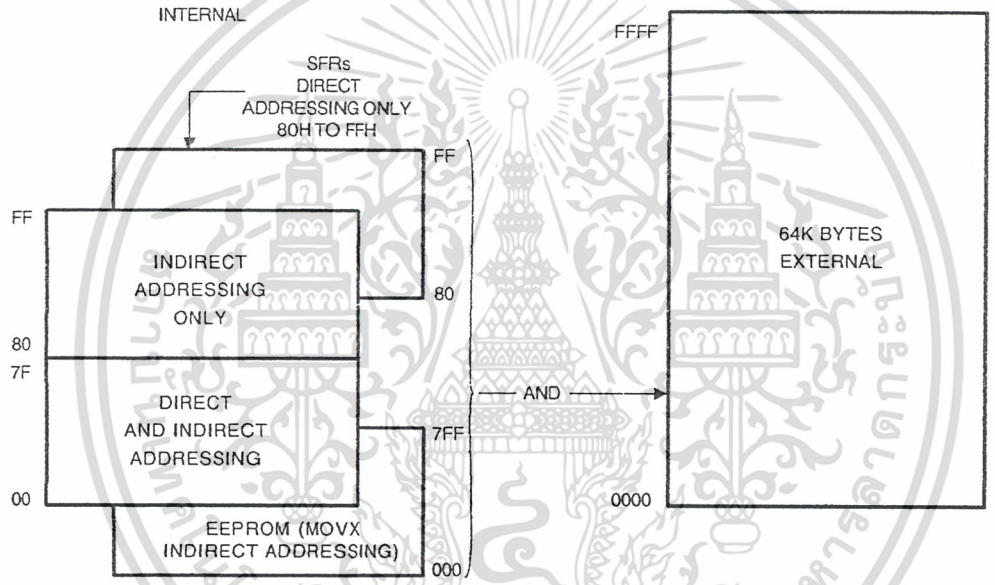
To the left in Figure 2 are shown the 256 bytes of internal RAM and the SFRs, which shadow the upper 128 bytes of internal RAM. The lower 128 bytes (00H-7FH) of internal RAM are accessible by both direct and indirect addressing, while the upper 128 bytes (80H-FFH) are accessible by indirect addressing only. The SFRs (80H-FFH) are accessible by direct addressing only. The addressing mode of an instruction distinguishes accesses to the upper 128 bytes of internal RAM from accesses to the overlapping SFRs.

For an explanation of addressing modes, consult the Architectural Overview section in the Atmel AT89-series Microcontroller Databook.

The stack, which grows upward, may reside anywhere in the 256 bytes of internal RAM.

To the right in Figure 2 are shown the 2K bytes of on-chip EEPROM and the optional 64K bytes of external data memory. Although the EEPROM is internal, it is shown in the diagram shadowing the lower 2K bytes of external data memory because some of the same instructions are used to access EEPROM as are used to access external data memory.

Figure 2. The AT89S8252 Data Memory



EEPROM

EEPROM and external data memory are accessible by indirect addressing only, utilizing the MOVX instructions, which come in two flavors: 8-bit and 16-bit. Only the 16-bit MOVX instructions (those utilizing DPTR) may be used to access internal EEPROM. The 2K bytes of EEPROM are accessed at addresses 000H-7FFH.

Accesses to EEPROM are distinguished from accesses to external data memory by the state of the EEMEN bit in SFR WMCON (96H). To access EEPROM, EEMEN is set; to access external data memory, EEMEN is cleared. Reset clears EEMEN.

To enable write accesses to EEPROM, bit EEMWE in SFR WMCON must also be set. Reset clears this bit, disabling EEPROM writes. It is not necessary to explicitly erase any portion of EEPROM before writing new data.

A write to a location in EEPROM triggers an internal programming cycle, which is guaranteed to last no longer than 10 milliseconds. The completion of an EEPROM programming cycle may be determined by monitoring the RDY/$\overline{\text{BSY}}$ bit in SFR WMCON. RDY/$\overline{\text{BSY}}$ low indicates that programming is in progress; RDY/$\overline{\text{BSY}}$ high indicates that programming is complete. When programming is complete, the contents of the written location may be read back and verified.

The end of an EEPROM programming cycle may also be determined utilizing the $\overline{\text{DATA}}$ Polling method, in which the location written is read repeatedly. During programming, the most significant bit of the data read is the complement of the data bit written. When programming is complete, true data is returned. The return of true data also serves as verification of the write operation.

Sample code showing EEPROM reads and writes is presented in Listing 1.

Listing 1: EEPROM Read/Write Examples.

```
; The WMCON register is not bit-addressable, so Boolean operations are used
; to control functions and test bits.

WMCON     DATA     96h            ; watchdog and memory control register
EEMEN     EQU      00001000b      ; EEPROM access enable bit
EEMWE     EQU      00010000b      ; EEPROM write enable bit
WDTRST    EQU      00000010b      ; EEPROM RDY/BSY bit

; EEPROM read example.

          orl      WMCON, #EEMEN  ; enable EEPROM accesses
          mov      dptr, #ADDRESS ; address to read
          movx     a, @dptr       ; read EEPROM
          xrl      WMCON, #EEMEN  ; disable EEPROM accesses

; EEPROM write example, utilizing fixed delay for write cycle.
; Delay is worst case (10 ms). Code for delay is not shown.
; Write is followed by verify (read and compare), but code to handle
; verification failure is not shown.

          orl      WMCON, #EEMEN  ; enable EEPROM accesses
          orl      WMCON, #EEMWE  ; enable EEPROM writes

          mov      dptr, #ADDRESS ; address to write
          mov      a, #DATA       ; data to write
          movx     @dptr, a       ; write EEPROM

          call     DELAY_10_MS    ; wait 10 ms

          movx     a, @dptr       ; read EEPROM
          cjne     a, #DATA, ERROR; jump if data compare fails

          xrl      WMCON, #EEMWE  ; disable EEPROM writes
          xrl      WMCON, #EEMEN  ; disable EEPROM accesses

; EEPROM write example, utilizing RDY/BSY to determine the end of
; the write cycle. Write is followed by verify (read and compare),
; but code to handle verification failure is not shown.
; Needs timeout to prevent write error from causing an infinite loop.

          orl      WMCON, #EEMEN  ; enable EEPROM accesses
          orl      WMCON, #EEMWE  ; enable EEPROM writes

          mov      dptr, #ADDRESS ; address to write
          mov      a, #DATA       ; data to write
          movx     @dptr, a       ; write EEPROM
    loop:
          mov      a, WMCON       ; get EEPROM write status
          anl      a, #WDTRST     ; check RDY/BSY
          jz       loop           ;jump if busy
```

```
        movx    a, @dptr        ; read EEPROM
        cjne    a, #DATA, ERROR; jump if data compare fails

        xrl     WMCON, #EEMWE   ; disable EEPROM writes
        xrl     WMCON, #EEMEN   ; disable EEPROM accesses

; EEPROM write example, utilizing DATA Polling to determine the end of
; the write cycle. After data is loaded, the code loops on read until
; data is returned true. Write verification is implicit in this method.
; Needs timeout to prevent write error from causing an infinite loop.

        orl     WMCON, #EEMEN   ; enable EEPROM accesses
        orl     WMCON, #EEMWE   ; enable EEPROM writes

        mov     dptr, #ADDRESS  ; address to write
        mov     a, #DATA        ; data to write
        movx    @dptr, a        ; write EEPROM
    loop:
        movx    a, @dptr        ; read EEPROM
        cjne    a, #DATA, loop  ; jump if data compare fails (busy)

        xrl     WMCON, #EEMWE   ; disable EEPROM writes
        xrl     WMCON, #EEMEN   ; disable EEPROM accesses
```

Dual Data Pointers

The AT89S8252 features two 16-bit data pointers (DP0 and DP1) for accessing data in program memory, external data memory, and on-chip EEPROM. The low and high bytes of DP0 are stored in SFRs DP0L (82H) and DP0H (83H), respectively. The low and high bytes of DP1 are stored in SFRs DP1L (84H) and DP1H (85H), respectively. Note that DP0 occupies the same SFRs as the single data pointer in conventional 8051 microcontrollers.

In the AT89S8252, the DPS bit in SFR WMCON (96H) selects the active data pointer (DP0 or DP1). All instructions which reference DPTR utilize the data pointer which is currently selected. To select DP0, DPS is cleared; to select DP1, DPS is set. Reset clears DPS.

The two data pointers may be used to expedite the transfer of data between program memory, external data memory, and on-chip EEPROM, as shown in Listing 2.

Listing 2: Dual Data Pointer Examples.

```
; The WMCON register is not bit-addressable, so Boolean operations are used.
WMCON     DATA     96h            ; watchdog and memory control register
EEMEN     EQU      00001000b      ; EEPROM access enable bit
EEMWE     EQU      00010000b      ; EEPROM write enable bit
WDTRST    EQU      00000010b      ; EEPROM RDY/BSY bit
DPS       EQU      00000100b      ; data pointer select bit

; Copy block from program memory to external data memory.

          mov      r7, #COUNT     ; block byte count
          mov      dptr, #PGM_ADDR; pointer to program memory
          xrl      WMCON, #DPS    ; switch data pointers
          mov      dptr, #XD_ADDR ; pointer to external data memory
    loop:
          xrl      WMCON, #DPS    ; switch data pointers
          clr      a              ; read program memory
          movc     a, @a+dptr     ;
          inc      dptr           ; advance program memory pointer
          xrl      WMCON, #DPS    ; switch data pointers
          movx     @dptr, a       ; write external data memory
          inc      dptr           ; advance external data memory pointer
          djnz     r7, loop       ; continue until done

; Copy block from external data memory to on-chip EEPROM.
; Utilizes RDY/BSY to determine the end of the EEPROM write cycle.
; Needs timeout to prevent write error from causing an infinite loop.

          orl      WMCON, #EEMEN ; enable EEPROM accesses
          orl      WMCON, #EEMWE ; enable EEPROM writes

          mov      r7, #COUNT     ; block byte count
          mov      dptr, #EE_ADDR ; pointer to EEPROM
          xrl      WMCON, #DPS    ; switch data pointers
          mov      dptr, #XD_ADDR ; pointer to external data memory
    copy:
          movx     a, @dptr       ; read external data memory
          inc      dptr           ; advance external data memory pointer
          xrl      WMCON, #DPS    ; switch data pointers
          movx     @dptr, a       ; write EEPROM
          inc      dptr           ; advance EEPROM pointer
          xrl      WMCON, #DPS    ; switch data pointers
    wait:
          mov      a, WMCON       ; get EEPROM write status
          anl      a, #WDTRST     ; check RDY/BSY
          jz       wait           ; jump if busy
          djnz     r7, copy       ; continue until done

          xrl      WMCON, #EEMWE; disable EEPROM writes
          xrl      WMCON, #EEMEN; disable EEPROM accesses
```

IMPORTANT: The state of DPS affects ALL accesses to the data pointer SFRs (82H, 83H, 84H, 85H). Any machine instruction whose operand is one of the data pointer SFRs may produce unexpected results, as shown below.

Examples:

```
; Define the new data pointer SFRs for a generic 8051 assembler.

DP0L    DATA    82h             ; data pointer 0
DP0H    DATA    83h             ;
DP1L    DATA    84h             ; data pointer 1
DP1H    DATA    85h             ;
          .
        orl     WMCON, #DPS     ; set DPS

        push    DP0L            ; PUSHES DP1L!!!
        mov     83h, a          ; COPIES ACCUMULATOR TO 85H!!!

        xrl     WMCON, #DPS     ; clear DPS

        push    DP1H            ; PUSHES DP0H!!!
        mov     84h, a          ; COPIES ACCUMULATOR TO 82H!!!
```

The user must exercise caution to avoid accessing the wrong SFRs. The solution to the problem demonstrated above is to clear DPS before any accesses to SFRs 82H and 83H and to set DPS before any accesses to SFRs 84H and 85H.

Watchdog Timer

The AT89S8252 features a watchdog timer which allows control of the microcontroller to be regained, should it be lost. When enabled, the timer will reset the microcontroller after a specified period has elapsed, unless prevented from doing so by the intervention of the firmware.

To enable the watchdog timer, the WDTEN bit in SFR WMCON (96H) must be set; to disable the timer, WDTEN should be cleared. Once the timer is enabled, the firmware must set the WDRST bit in SFR WMCON (or disable the timer) before the reset period elapses to prevent the timer from resetting the microcontroller. Each time WDRST is set, a new reset period begins, requiring another response from the firmware. The firmware does not need to clear WDRST after setting it; WDRST is automatically cleared by the microcontroller.

The watchdog timer reset period varies from 16 to 2048 milliseconds, as specified by bits PS0, PS1 and PS2 in SFR WMCON. Refer to the AT89S8252 data sheet for the nominal reset periods corresponding to the bit settings. The timer reset period is independent of the frequency of the clock source driving the microcontroller and may deviate from the documented nominal value by a huge percentage.

The watchdog timer continues to operate even when the microcontroller is in Idle mode, but is disabled during Power Down mode. The elapsed time between a watchdog timer reset and the execution of the first instruction is approximately 16 ms. Reset (including reset generated by the watchdog timer) clears WDTEN, WDRST, SP0, SP1 and SP2, disabling the watchdog timer.

A typical application of the watchdog timer is outlined in Listing 3.

Listing 3: Watchdog Timer Example.

```
; Use the watchdog timer to regain control of the microcontroller if an
; operation takes longer than expected. The details of the operation are not
; shown. The operation is expected to take less than 20 ms to complete and
; the reset period chosen is 32 ms. Adequate margin must be allowed between
; the desired reset period and the selected period to allow for the slop
; present in the timer.
; The WMCON register is not bit-addressable, so Boolean operations are used.

WMCON     DATA     96h            ; watchdog and memory control register
WDTEN     EQU      00000001b      ; watchdog timer enable bit
WDTRST    EQU      00000010b      ; watchdog timer reset bit
PS0       EQU      00100000b      ; watchdog timer period select bits
PS1       EQU      01000000b      ;
PS2       EQU      10000000b      ;

          orl      WMCON, #PS0    ; select 32-ms period
          orl      WMCON, #WDTEN  ; enable watchdog
      loop:
          ; Do something which normally takes less than 20 ms.
          .
          .
          .

          orl      WMCON, #WDTRST ; keep watchdog at bay
          jmp      loop
```

Power Off Flag

The Power Off Flag (POF) indicates that power has been removed from the AT89S8252. This allows the firmware to differentiate between reset due to the application of power and reset due to the watchdog timer, or a logic high on the RST pin. POF is set when power is applied to the microcontroller and is not affected by the watchdog timer or by activity on RST. POF is located at bit four in SFR PCON (87H), and may be read, set, or cleared by firmware. Note that PCON is not bit-addressable.

A typical application of the Power Off Flag is outlined in Listing 4.

Listing 4. Power Off Flag Example.

```
; After reset, the microcontroller begins executing code at program memory
; address 0000H. POF is tested to determine if the controller was reset
; by the application of power (cold start) or by the watchdog timer or a
; high on RST (warm start).
; Code for the cold start and warm start routines is not shown.

POF     EQU     00010000b       ; Power Off Flag bit

        CSEG                    ; code segment

        ORG     0000h           ; location of reset vector
        jmp     xreset          ; vector
        .
        .
        .
xreset:                         ; code for responding to reset
        .
        .
        .
        mov     a, PCON         ; get Power Control register
        anl     a, #POF         ; test Power Off Flag
        jz      WARM_START      ; POF=0 indicates reset from
                                ;   watchdog timer or RST
        xrl     PCON, #POF      ; clear POF for next time
        jmp     COLD_START      ; POF=1 indicates reset from power
```

Serial Peripheral Interface

The Serial Peripheral Interface (SPI) permits compatible devices to communicate serially over a high-speed, synchronous bus. Devices resident on the bus act as masters or slaves, with only one master and one slave active at any one time. Data transfers are always initiated by a master, and are actually data exchanges, with data flowing from the master to the slave and from the slave to the master simultaneously.

SPI-compatible devices have four pins in common: SCK, MOSI, MISO, and $\overline{SS}$. All devices in a system have their SCK, MOSI, and MISO pins tied together. Data flows from master to slave via MOSI (Master Out Slave In) and from slave to master via MISO (Master In Slave Out). Data transfers are synchronized to a clock generated by the master and output on its SCK pin. SCK is an input for devices configured as slaves. Inactive masters must be reconfigured as slaves to prevent them from driving their SCK and MOSI pins.

The $\overline{SS}$ (Slave Select) pins on the devices in the system are not bussed. Each slave is connected to its master by a select line from its $\overline{SS}$ input to a general purpose output on the master. If a slave has multiple masters, the multiple select lines must be gated to its $\overline{SS}$ input. Masters do not utilize their $\overline{SS}$ pins during SPI data transfers, freeing them for use as general-purpose outputs.

To initiate an SPI data transfer, the active master selects a slave by applying a logic low to the slave's $\overline{SS}$ input. The master starts the serial clock, which it outputs on its SCK pin, and shifts out a byte on its MOSI pin, synchronized to the clock. Simultaneously, the slave shifts out a byte on its MISO pin, synchronized to the clock. When the master and slave have exchanged data, the transfer is complete. The master stops the serial clock and may deselect the slave. Slaves which are not selected ignore their SCK inputs and float their MISO outputs to avoid contention with the active output of the selected slave.

In the AT89S8252, the SPI is configured via SFR SPCR (D5H), the SPI Control Register. The frequency of the serial clock, the ordering of the serial data, and the relationship between the clock and the shifting and sampling of data are all programmable, as described below.

To enable the SPI feature, the SPE bit in SFR SPCR must be set; to disable the SPI, SPE is cleared. When the SPI is enabled, microcontroller pins P1.4, P1.5, P1.6 and P1.7 become $\overline{SS}$, MOSI, MISO, and SCK, respectively. The SPI may not operate correctly unless pins P1.4-P1.7 are first

programmed high. Reset sets pins P1.4-P1.7 high and clears SPE, disabling the SPI.

The MSTR bit in SFR SPCR configures the microcontroller as a SPI master when set, and as a slave when cleared. Reset clears MSTR. When the microcontroller is configured as a SPI master, $\overline{SS}$ (P1.4) is not utilized and may be used as a general-purpose, programmable output.

When the microcontroller is configured as a SPI master, the frequency of the serial clock is determined by bits SPR0 and SPR1 in SFR SPCR. The frequency of the serial clock is the frequency of the microcontroller's clock source divided by the selected divisor. The divisor must be selected to produce a serial clock frequency which is compatible with the master's slaves. Refer to the AT89S8252 data sheet for the divisors corresponding to the settings of bits SPR0 and SPR1.

The DORD bit in SFR SPCR determines the order in which the bits in the serial data are transferred. Data is transferred least-significant bit (LSB) first when DORD is set; most-significant bit (MSB) first when DORD is cleared. Reset clears DORD. Note that only MSB-first data transfers are shown in the diagrams in the AT89S8252 data sheet.

The polarity of the SPI serial clock is determined by the CPOL bit in SFR SPCR. Setting CPOL specifies serial clock high when idle; clearing CPOL specifies serial clock low when idle. Reset clears CPOL.

The CPHA bit in SFR SPCR controls the phase of the SPI serial clock, which defines the relationship between the clock and the shifting and sampling of serial data. Setting CPHA specifies that data is to be shifted on the leading edge of the clock and sampled on the trailing edge. Clearing CPHA specifies that data is to be sampled on the leading edge of the clock and shifted on the trailing edge. Reset sets CPHA. The state of bit CPHA also affects the slave selects. If CPHA is set, the slave may remain selected between consecutive byte transfers, or may be permanently selected ($\overline{SS}$ tied low). If CPHA is clear, the slave must be deselected ($\overline{SS}$ returned high) after each byte transferred. Examples of SPI serial clock phase and polarity are shown in the diagrams in the AT89S8252 data sheet.

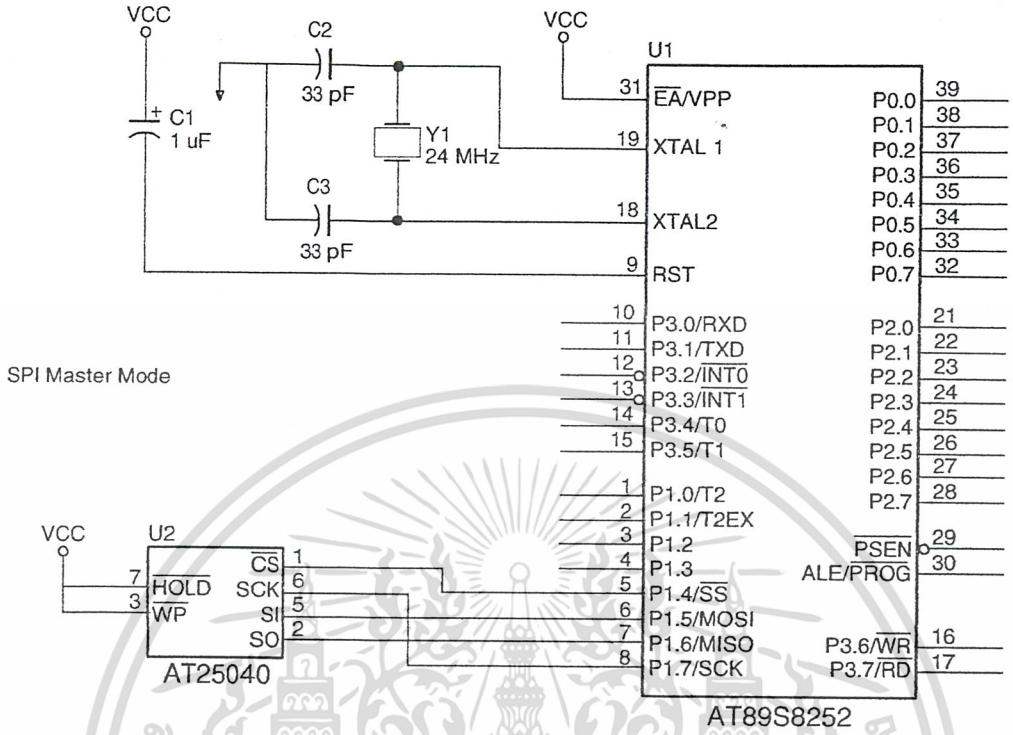
Only an AT89S8252 configured as an SPI master may initiate a data transfer. A data transfer is triggered by a byte written to SFR SPDR (86H), the SPI Data Register. As data is shifted out of the master, data from the selected slave is simultaneously shifted in, replacing the data in SPDR. When a data transfer is complete, the SPIF bit is set in SFR SPSR (AAH), the SPI Status Register. The data received from the slave may then be read from SPDR. Writing SPDR during a data transfer sets the Write Collision bit (WCOL) in SPSR. The progress of the data transfer is not affected by a collision. To clear bits SPIF and WCOL, read SPSR and read or write SPDR.

An interrupt may be generated as an alternative to polling SPIF to determine the end of a SPI data transfer. To enable the SPI interrupt, three bits must be set. The first is the SPIE bit in SPCR, which causes an interrupt to be generated when SPIF is set. The second and third bits are ES and EA in SFR IE (A8H). ES is the UART interrupt enable bit, which must be set because the SPI shares an interrupt vector with the UART. EA is the global interrupt enable bit. When an SPI interrupt occurs, the SPI/UART interrupt service routine must determine the source of the interrupt. An SPI interrupt is indicated when the SPIF bit in SPSR is set. Bits SPIF and ES must be cleared by software.

In the application shown below, the AT89S8252 is configured as an SPI master and interfaces to an Atmel AT25040 SPI-compatible EEPROM. The EEPROM provides 512 bytes of re-writable, non-volatile storage while requiring only a four-pin interface to the microcontroller. The microcontroller and EEPROM are wired as shown in Figure 3. Note that the microcontroller's $\overline{SS}$ pin is used as a slave select, since it is unused when the microcontroller is configured as a SPI master. Additional EEPROMs may be connected to the microcontroller's SCK, MISO and MOSI pins, but each device must have its own select line.

Sample code for the application is shown in Listing 5. A SPI master must be configured to meet the requirements of its slaves. The AT25040 data sheet states that the maximum clock rate for the device is 2 MHz. The microcontroller's clock source is a 24-MHz crystal (Figure 3), so a SPI serial clock divisor of 16 was chosen to produce a serial clock of 1.5 MHz. As shown in the AT25040 data sheet, the device's chip select ($\overline{CS}$) input must remain active (low) for the duration of an operation, which may include multiple data transfers. Also, the serial clock must be low when idle and data is transferred most-significant bit first. Therefore, CPHA=1, CPOL=0 and DORD=0. In the example, SPI interrupts are not used.

Figure 3. AT89S8252 as an SPI Master



Listing 5: SPI Example.

```
; Write/Read AT25C040 EEPROM via the Serial Peripheral Interface (SPI).
; Completion of AT25C040 programming is determined by polling the device.
; SPI interrupt is not used. ; Works with a microcontroller clock of 24 MHz (or slower).
;
; The AT25040 routines ("read_status", "enable_write", "read_byte",
; "write_byte") are excerpted from code previously made available by Atmel
; for use with the AT89Cx051 microcontrollers. In that code, access to the
; AT25040 was via "bit banging". The two routines which shifted the serial
; data in/out have been replaced by the single SPI routine "masterIO".

; Microcontroller registers and bit definitions.

SPCR    DATA    0d5h            ; SPI control register
SPSR    DATA    0aah            ; SPI status register
SPIF    EQU     10000000b       ; interrupt flag
SPDR    DATA    86h             ; SPI data register

; Microcontroller connections to AT25040.

CS_     BIT     p1.4            ; AT25040 slave select
MOSI    BIT     p1.5            ; SPI
MISO    BIT     p1.6            ; SPI
SCK     BIT     p1.7            ; SPI

; AT25040 device command and bit definitions.
```

```
RDSR    EQU     05h             ; Read Status Register
WRSR    EQU     01h             ; Write Status Register
READ    EQU     03h             ; Read Data from Memory
WRITE   EQU     02h             ; Write Data to Memory
WREN    EQU     06h             ; Write Enable
WRDI    EQU     04h             ; Write Disable

A8      BIT     acc.3           ; MSB of address
NRDY    BIT     acc.0           ; high = write cycle in progress

main:
      ; SPI master mode initialization code.

        setb    CS_             ; deselect AT25040

        setb    MOSI            ; initialize SPI pins
        setb    MISO            ;
        setb    SCK             ;

        mov     SPCR, #01010101b        ; initialize SPI master
                                ; interrupt disable, pin enable,
                                ; MSB first, polarity 0, phase 1,
                                ; clock rate /16
; Write one byte to AT25040 and verify (read and compare).
; Code to handle verification failure is not shown.
; Needs timeout to prevent write error from causing an infinite loop.

        call    enable_write    ; must precede each byte write
        mov     a, #DATA        ; data
        mov     dptr, #ADDRESS  ; address
        call    write_byte      ; write
    wchk:
        call    read_status     ; check write status
        jb      NRDY, wchk      ; loop until done

        mov     dptr, #ADDRESS  ; address
        call    read_byte       ; read
        cjne    a, #DATA, ERROR ; jump if data compare fails
        .
        .
        .
read_status:

    ; Read device status.
    ; Returns status byte in A.

        clr     CS_             ; select device
        mov     a, #RDSR        ; get command
        call    masterIO        ; send command
        call    masterIO        ; get status
```

```
        setb    CS_             ; deselect device
        ret

enable_write:

    ; Enable write.
    ; Does not check for device ready before sending command.
    ; Returns nothing. Destroys A.

        clr     CS_             ; select device
        mov     a, #WREN        ; get command
        call    masterIO        ; send command
        setb    CS_             ; deselect device
        ret

read_byte:

    ; Read one byte of data from specified address.
    ; Does not check for device ready before sending command.
    ; Called with address in DPTR.
    ; Returns data in A.

        clr     CS_             ; select device
        mov     a, dph          ; get high byte of address
        rrc     a               ; move LSB into carry bit
        mov     a, #READ        ; get command
        mov     A8, c           ; combine command and high bit of addr
        call    masterIO        ; send command and high bit of address
        mov     a, dpl          ; get low byte of address
        call    masterIO        ; send low byte of address
        call    masterIO        ; get data
        setb    CS_             ; deselect device
        ret

write_byte:

    ; Write one byte of data to specified address.
    ; Does not check for device ready or write enabled before sending
    ; command. Does not wait for write cycle to complete before returning.
    ; Called with address in DPTR, data in A.
    ; Returns nothing.

        clr     CS_             ; select device
        push    acc             ; save data
        mov     a, dph          ; get high byte of address
        rrc     a               ; move LSB into carry bit
        mov     a, #WRITE       ; get command
        mov     A8, c           ; combine command and high bit of address
        call    masterIO        ; send command and high bit of address
        mov     a, dpl          ; get low byte of address
```

```
        call    masterIO        ; send low byte of address
        pop     acc             ; restore data
        call    masterIO        ; send data
        setb    CS_             ; deselect device
        ret

masterIO:
    ; Send/receive data through the SPI port.
    ; A byte is shifted in as a byte is shifted out,
    ; receiving and sending simultaneously.
    ; Waits for shift out/in complete before returning.
    ; Expects slave already selected.
    ; Called with data to send in A. Returns data received in A.

        mov     SPDR, a         ; write output data
    bbb:
        mov     a, SPSR         ; get status
        anl     a, #SPIF        ; check for done
        jz      bbb             ; loop until done

        move    a, SPDR         ; read input data
        ret
```