การควบคุมการทำงานของเครื่องปรับอากาศด้วยรีโมทคอนโทรล

Air Conditioner Remote Control

โดย

นาย ปฏิพัทธ์ โพธิ์นิ่มแดง

อาจารย์ที่ปรึกษา

อาจารย์ เกียรติวรรณ ทรงสัตย์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรคณะวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมระบบควบคุม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2539

ปริญญานิพนธ์ ปีการศึกษา 2539

ภาควิชา ระบบควบคุม

คณะ วิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้า
เจ้าคุณทหารลาดกระบัง

เรื่อง การควบคุมการทำงานของเครื่อง
ปรับอากาศด้วยรีโมทคอนโทรล
Air - Conditioner remote control

ผู้จัดทำ นาย ปฏิพัทธ์ โพธิ์นิ่มแดง 35104242

อาจารย์ เกียรติวรรณ ทรงสัตย์ อาจารย์ที่ปรึกษา

(.....)

การควบคุมการทำงานของเครื่องปรับอากาศด้วยรีโมทคอนโทรล

AIR-CONDITIONER REMOTE CONTROL

โดย

นาย ปฏิพัทธ์ โพธิ์นิ่มแดง 35104242

อาจารย์ที่ปรึกษา

อาจารย์ เกียรติวรรณ ทรงสัตย์

บทคัดย่อ

โครงงานนี้ เป็นการศึกษาเกี่ยวกับการนำรีโมทคอนโทรลมาใช้ส่งคำสั่งควบคุมการทำงานของเครื่องปรับอากาศที่ใช้ไมโครโปรเซสเซอร์ซึ่งเป็นการควบคุมแบบ เปิด - ปิด โดยจะเน้นที่การนำสัญญาณจากรีโมทมาเปลี่ยนเป็นสัญญาณที่สามารถส่งให้ซีพียู 89C51 ประมวลผลและควบคุมระบบของวงจรเครื่องปรับอากาศที่ใช้ไมโครโปรเซสเซอร์ 89C51 เป็นตัวควบคุมตามโปรแกรมการทำงานที่ได้บรรจุไว้ในร็อมภายใน 89C51 เอง

Abstract

This project is study to using remote control to send instruction to control an on-off control Air-Conditioner control circuit by microprocessor, in particular, the way it decode the signal from remote to the signal that CPU 89C51 can use for operation and control the system of Air-Conditioner control circuit that uses a microprocessor 89C51 to operate as the main part to receive instruction ,send control signals and react as is programmed itself.

กิตติกรรมประกาศ

ขอขอบคุณอาจารย์ เกียรติวรรณ ทรงสัตย์ ที่ได้กรุณาให้คำปรึกษาแนะนำตลอดจนให้ความอนุเคราะห์ทางด้านเครื่องมือ วัสดุอุปกรณ์

ขอขอบคุณ พี่ ๆ น้อง ๆ ทุกคนที่คอยช่วยเหลือให้ความช่วยเหลือทุก ๆ เรื่อง

ขอขอบคุณ คุณแม่ที่ช่วยสนับสนุนทุก ๆ อย่างมาโดยตลอด

สุดท้ายนี้ขอขอบคุณ ทุกคนที่รัก ตลอดจนกำลังใจและความห่วงใยที่ส่งมาให้เสมอ

สารบัญ

สารบัญรูป

หน้าที่

หน้าที่

สารบัญตาราง

หน้าที่

บทที่ 1
บทนำ

เครื่องปรับอากาศเป็นเครื่องใช้ไฟฟ้าที่อำนวยความสะดวกสบาย ซึ่งมีความจำเป็นมากในปัจจุบัน เนื่องจากประเทศไทยเป็นประเทศร้อน เครื่องปรับอากาศจะช่วยปรับสภาพอากาศภายในที่ทำงานหรือที่อยู่อาศัยให้เย็นสบาย ไม่ร้อนอบอ้าว ในบางสถานที่ที่มีความจำเป็นอย่างยิ่งที่จะต้องใช้เครื่องปรับอากาศ เช่น ในศูนย์คอมพิวเตอร์ซึ่งมีคอมพิวเตอร์เป็นจำนวนมาก ถ้าขาดเครื่องปรับอากาศแล้ว เครื่องคอมพิวเตอร์เหล่านี้อาจทำงานได้ไม่ดีเท่าที่ควร เครื่องปรับอากาศโดยทั่วๆ ไป จะประกอบด้วยส่วนสำคัญ 4 ส่วน คือ

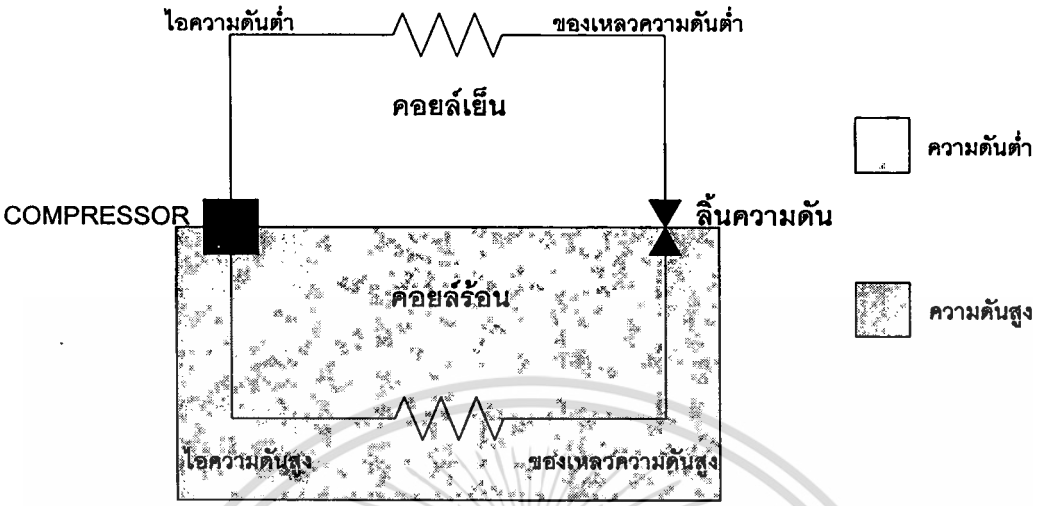
1. คอยล์เย็น (EVAPORATOR) มีลักษณะเป็นท่อขดอยู่ภายนอกห้อง ทำหน้าที่เป็นตัวสร้างความเย็น

2. คอยล์ร้อน (CONDENSOR) มีลักษณะเป็นท่อขดอยู่ภายนอกห้อง ทำหน้าที่เป็นตัวระบายความร้อน

3. ลิ้นลดความดัน (EXPANSION VALVE) ทำหน้าที่ในการปรับระดับความดันของเหลวภายในท่อให้ลดลง

4. คอมเพรสเซอร์ (COMPRESSOR) ทำหน้าที่เป็นตัวดูดและอัดของเหลวภายในท่อให้ได้ความดันตามต้องการ และเป็นตัวทำให้ของเหลวภายในท่อไหลวนในปริมาณที่ต้องการ

ในส่วนประกอบที่สำคัญของเครื่องปรับอากาศทั้ง 4 ส่วน ตัวคอมเพรสเซอร์ถือเป็นส่วนที่สำคัญที่สุด เนื่องจากคุณภาพการทำงาน และอายุการใช้งานของเครื่องปรับอากาศนั้น ขึ้นอยู่กับคอมเพรสเซอร์นี้เป็นส่วนใหญ่ ในขณะที่คอมเพรสเซอร์ทำงาน บางครั้งอาจจะมีปัจจัยภายนอกที่มีผลกระทบทำให้ตัวคอมเพรสเซอร์เสื่อมสภาพ เนื่องจากลักษณะการทำงานที่ผิดปกติ เช่น ลักษณะที่เกิดจากไฟตกไฟเกิน หรือลักษณะของไฟที่ดับและติดขึ้นมาใหม่อย่างกระทันหันในช่วงเวลาสั้นๆ สิ่งเหล่านี้ทำให้ตัวคอมเพรสเซอร์เกิดความเสียหาย หรือเสื่อมสภาพได้ทั้งสิ้น



รูปที่ 1.1 ส่วนประกอบที่สำคัญของเครื่องปรับอากาศ

เพื่อป้องกัน และรักษาตัวคอมเพรสเซอร์ไว้ให้มีอายุการใช้งานที่ยืนนาน จึงมีความจำเป็นที่เครื่องปรับอากาศ ควรจะมีวงจรเพื่อควบคุมการทำงานของคอมเพรสเซอร์ นอกจากนี้ วงจรนี้ยังสามารถที่จะควบคุมระดับความเร็ว (SPEED) ของพัดลม เพื่อให้ความเย็นกระจายไปทั่วห้องอย่างเหมาะสมตามความต้องการ ดังนั้น จึงสรุปประโยชน์ของวงจรควบคุมเครื่องปรับอากาศที่สำคัญดังนี้

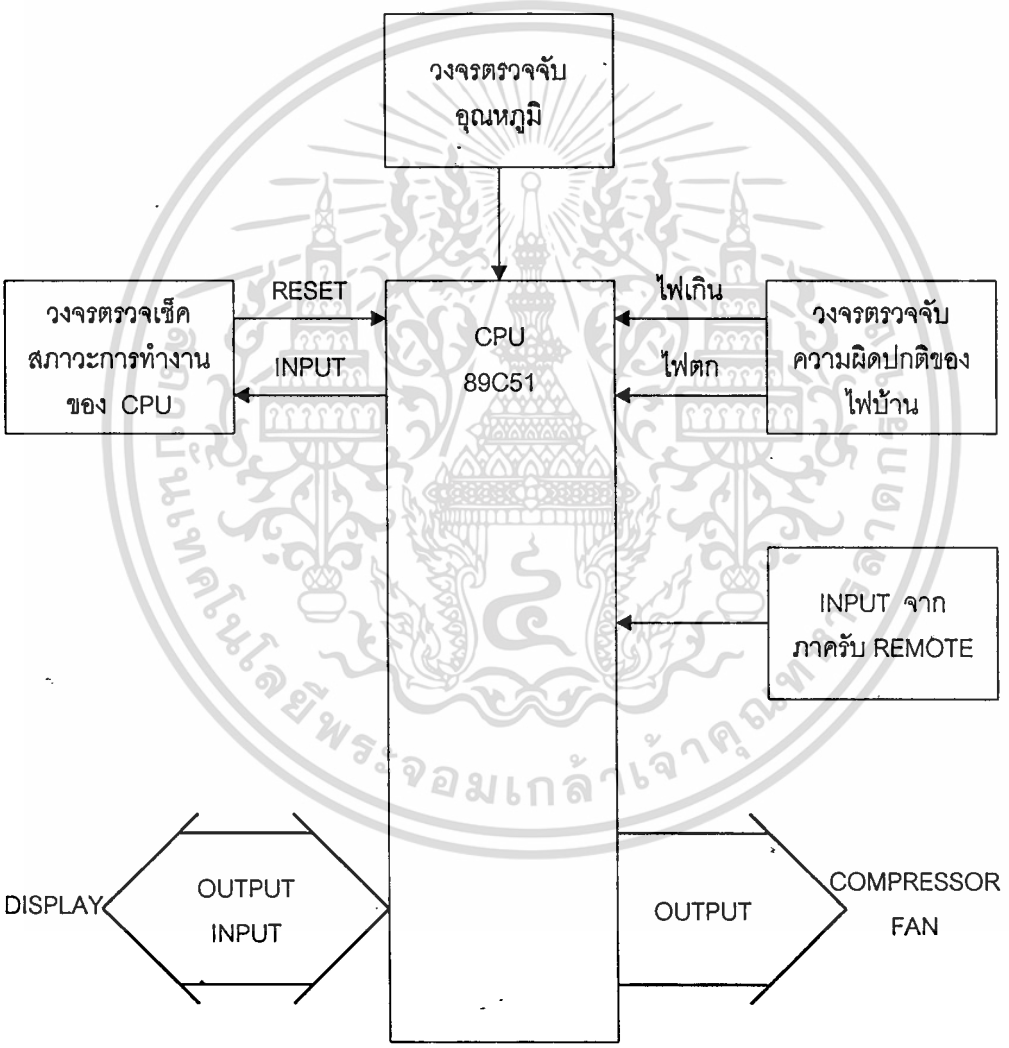
1.เพื่อป้องกันและรักษาสภาพของตัวคอมเพรสเซอร์ให้มีอายุการใช้งานที่ยาวนาน

2. เพื่อควบคุมระดับของอุณหภูมิภายในห้องให้เป็นไปตามที่เราต้องการ

โดยวงจรควบคุมในโครงงานนี้ จึงออกแบบมาเพื่อใช้งานกับเครื่องปรับอากาศที่ทำงานควบคุมแบบ ปิด-เปิด (ON-OFF CONTROL) ซึ่งเราจะได้ศึกษาตัววงจร และทำความเข้าใจหลักการทำงานของตัววงจรต่างๆในบทต่อๆไป

บทที่ 2
วงจรควบคุมการทำงานของเครื่องปรับอากาศ

ส่วนประกอบของวงจรควบคุมการทำงานของเครื่องปรับอากาศ ประกอบไปด้วยวงจรย่อยต่างๆ โดยมีศูนย์กลางการควบคุมการทำงานอยู่ที่ซีพียู 89C51 (CPU,Central Processing Unit) บล็อกไดอะแกรมของระบบทั้งหมด แสดงได้ด้วยรูปที่ 2.1



รูปที่ 2.1 แสดงระบบของวงจรควบคุมเครื่องปรับอากาศ

จากภาพจะเห็นได้ว่าระบบประกอบไปด้วย

1. ซีพียู 89C51 ทำหน้าที่เป็นศูนย์กลางการควบคุมการทำงานของระบบทั้งหมด

2. วงจรตรวจเช็คสภาวะการทำงานของซีพียู (วงจร WATCH DOG) ซึ่งเป็นวงจรที่คอยตรวจสอบดูว่า การทำงานของซีพียูเป็นปกติดีหรือเปล่า โยถ้าเกิดการผิดปกติขึ้นเมื่อใด วงจรก็จะทำการรีเซทให้ซีพียูเริ่มต้นทำงานใหม่ได้ทันที ทั้งนี้เพื่อเป็นการป้องกันการเสียหายของอุปกรณ์ทั้งหมด

3. วงจรตรวจจับอุณหภูมิ (TEMPERATURE SENSOR) จะทำหน้าที่ตรวจวัดอุณหภูมิของห้อง แล้วทำการส่งข้อมูลให้ซีพียูประมวลผล

4.วงจรตรวจจับความผิดปกติของไฟบ้าน ซึ่งจะคอยตรวจสอบดูว่าระดับแรงดันของไฟบ้านที่จะป้อนให้แก่คอมเพรสเซอร์เป็นปกติดีหรือเปล่า ทั้งนี้เพื่อป้องกันความเสียหายที่จะเกิดขึ้นกับคอมเพรสเซอร์ในกรณีที่ระดับแรงดันสูงหรือต่ำกว่าปกติ

5. ส่วนแสดงผล (DISPLAY) เป็นส่วนที่แสดงสถานะต่างๆของระบบให้ผู้ใช้ได้รับรู้และเป็นส่วนที่รับคำสั่งจากผู้ใช้

6.ส่วนภาครับคำสั่งจากเครื่องควบคุมระยะไกล (REMOTE INFRARED RECIEVER) เป็นส่วนที่จะรับสัญญาณจากภาคส่งระยะไกล (REMOTE CONTROL) แล้วนำมาแปลงรูปสัญญาณเพื่อส่งให้กับ ซีพียู 89C51 เพื่อทำการประมวลผลสัญญาณที่ได้มา

บทที่ 3
ไมโครคอนโทรลเลอร์แบบชิพเดี่ยวตระกูล 51

ไมโครคอนโทรเลอร์แบบชิพเดี่ยว (SINGLE CHIP MICROCONTROLLER) คือ ไมโครคอมพิวเตอร์ที่มีขนาดเล็กโดยบรรจุไว้ในแผงวงจรรวม (INTREGRATED CIRCUIT) เพียงชิพเดี่ยว เหมาะสำหรับงานควบคุมอุปกรณ์อื่นๆแบบอัตโนมัติ เพราะสามารถเขียนโปรแกรมควบคุมการทำงานได้ตามความต้องการ ไมโครคอนโทรลเลอร์แบบชิพเดี่ยวตระกูล 51 หรือ MCS-51 แต่ละเบอร์มีโครงสร้างและชุดคำสั่งต่างๆแสดงดังตาราง 3.1

Devices	ROMless version	EPROM version	ROM Bytes	RAM Bytes	8-bit I/O Ports	16-bit Timer/ Counters	Programmable Counter Array (PCA)	UART	Serial Expanstion port (SEP)	Global Serial Channel (GSC)	DMA Channels	A/D Channels	Interrupt Sources/ Vectors	Power down and Idle Modes
8051	8031	-	4K	128	4	2		/					6/5	
8051AH	8031AH	87C51H 87C51BH	4K	128	4	2		/					6/5	
8052AH	8032AH	87C52BH	8K	256	4	3		/					8/6	
80C51BH	80C31BH	87C51	4K	128	4	2		/					6/5	/
80C52	80C32	-	8K	256	4	3		/					8/6	/
83C51FA	80C51FA	87C51FA	8K	256	4	3	/	/					14/7	/
83C51FB	80C51FA	87C51FB	16K	256	4	3	/	/					14/7	/
83C152JA	80C152JA	-	3K	256	5	2		/		/	2		19/11	/
-	80C152JB	-	-	256	7	2		/		/	2		19/11	/
83C152JC	80C152JC	-	8K	256	5	2		/		/	2		19/11	/
-	80C152JD	-	-	256	7	2		/		/	2		19/11	/
83C452	80C452	87C452P	8K	256	5	2		/					9/8	/

ตารางที่ 3.1 ไมโครคอนโทรลเลอร์แบบชิพเดี่ยวตระกูล 51

MCS-51 ผลิตโดยบริษัท Intel มีการทำงานเป็นแบบ 8 บิท หรือ คำนวณทีละ 8 บิทนั่นเอง MCS-51 มีข้อดีดังนี้

- สามารถนำข้อมูลมา แอนด์ (AND), ออร์ (OR)หรือ คอมพลีเมนท์ (Complement) ได้ทั้งแบบทีละ 8 บิทและแบบทีละ 1 บิท

- สามารถใช้กับหน่วยความจำสำหรับโปรแกรม (Program Memory) หรือหน่วยความจำที่ใช้เก็บชุดคำสั่ง ได้สูงสุด 64 กิโลไบท์

- สามารถต่อหน่วยความจำสำหรับข้อมูล (Data Memory) ได้สูงสุด 64 กิโลไบท์

- ใน 8051 และ 89C51 มีหน่วยความจำสำหรับโปรแกรม 4 กิโลไบท์ และใน 8052 และ 8752 มีหน่วยความจำสำหรับโปรแกรม 8 กิโลไบท์ อยู่ภายในทำให้ไม่ต้องต่อหน่วยความจำสำหรับโปรแกรมอยู่ภายนอก ทำให้วงจรของระบบมีขนาดเล็ก และเป็นการลดสัญญาณรบกวนจากภายนอก

- มีพอร์ทแบบขนานสำหรับข้อมูลเข้าและออก จำนวน 32 บิท ที่ข้อมูลแต่ละบิทเป็นอิสระต่อกัน

- มีวงจรเวลาและวงจรนับ (Timer/Counter) ขนาด 16 บิท 2 ชุด (ใน 8052 มี 3 ชุด) ที่สามารถทำงานในโหมดต่างๆได้ 4 โหมด

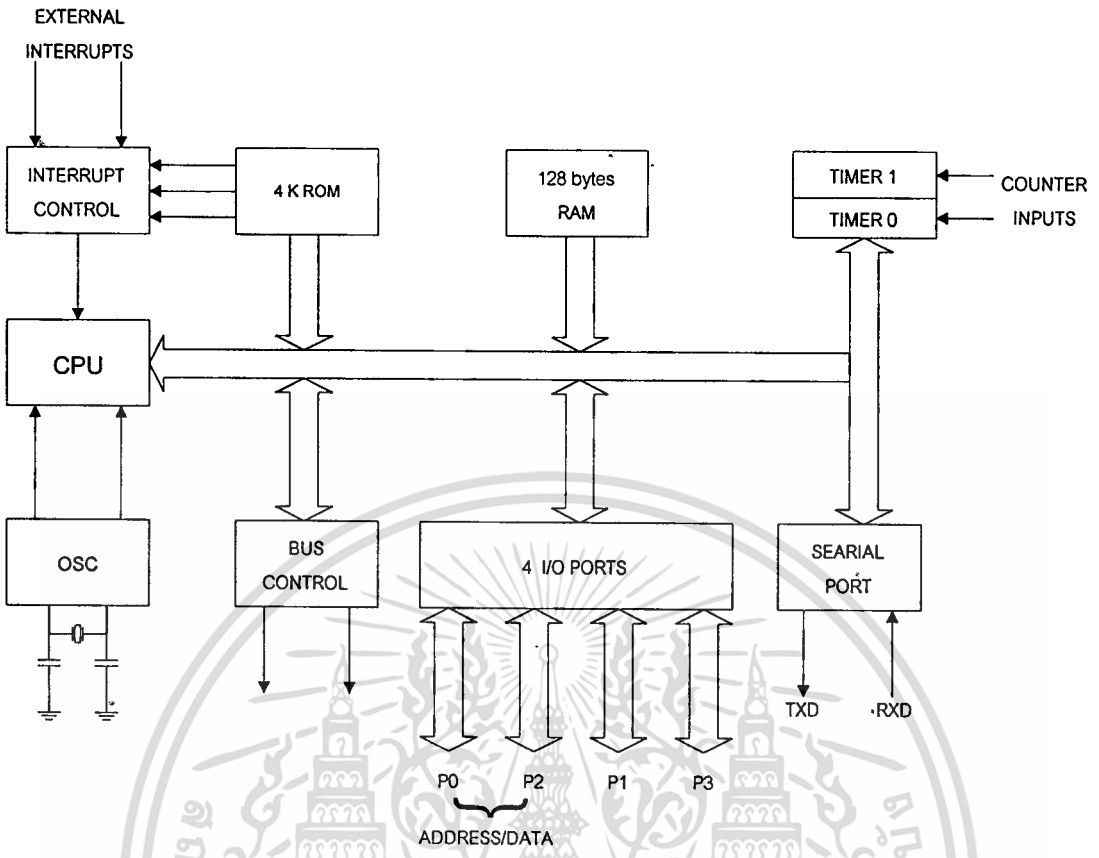
- และมี พอร์ทอนุกรมความเร็วสูง (Universal Asynchonous Receiver Tranmitter หรือ UART) สำหรับรับส่งข้อมูลอนุกรมแบบฟูลดูเพลกซ์ (Full Duplex) ที่เลือกรูปแบบการรับส่งได้ 4 แบบ

- มีแหล่งกำเนิดสัญญาณขัดจังหวะการทำงานของโปรแกรม(Interrupt Request Signal) 6 แหล่งซึ่งสามารถกระโดดไปทำงานตอบสนองการขัดจังหวะ (Interrupt Service Routine) ได้ต่างๆกัน 5 ตำแหน่ง

- สามารถเลือกการทำงานให้อยู่ในโหมดของไอเดียล (Idle) และประหยัดพลังงาน (Power down) ซึ่งประหยัดกำลังในการทำงาน

จากข้อดีดังกล่าว ทำให้ MCS-51 เป็นที่นิยมนำมาใช้ในการควบคุมระบบอัตโนมัติมาก คุณสมบัติดังกล่าวบรรจุในวงจรชิพเดี่ยว ขนาด 40 ขา จึงสามารถออกแบบให้ระบบทั้งหมดมีขนาดเล็ก การตรวจสอบหาข้อผิดพลาดในระบบทำได้ง่าย และยังลดปัญหาสัญญาณรบกวนในระบบ

โครงสร้างของ MCS-51 ภายใน จะประกอบไปด้วยส่วนย่อยๆดังไดอะแกรมในรูปที่ 3.1



รูปที่ 3.1 บล็อคไดอะแกรมของ MCS-51

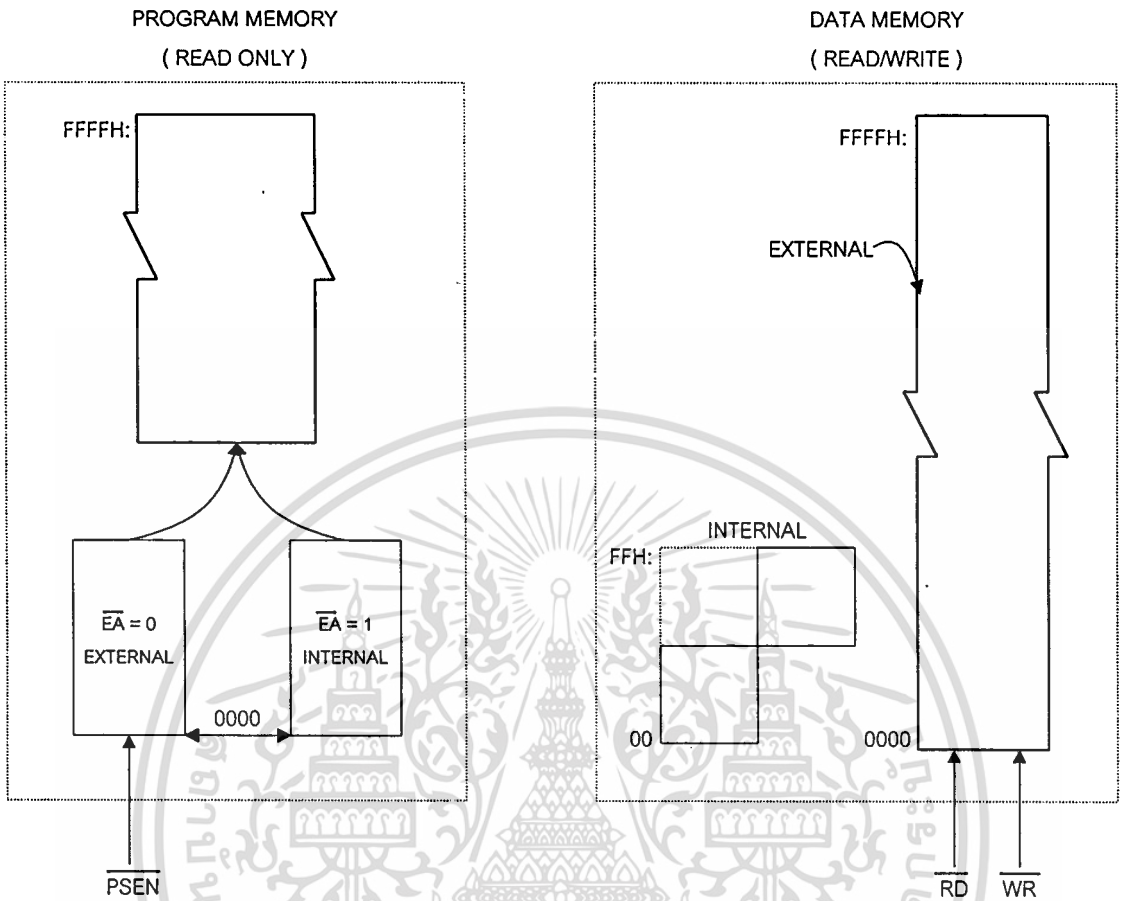
การจัดการหน่วยความจำของ MCS-51 แสดงในรูปที่ 3.2

แบ่งตามลักษณะการใช้งานได้ 2 แบบคือ

1. โปรแกรมเมมโมรี (Program Memory) เป็นหน่วยความจำที่ใช้เก็บคำสั่งที่ต้องการให้ทำงานซึ่งหน่วยความจำนี้ จะต้องเป็นร็อม (ROM) จึงต้องใช้เครื่องมือพิเศษในการเขียนข้อมูล ดังนั้นระหว่างที่ทำงาน จึงไม่สามารถเขียนข้อมูลลงในหน่วยความจำชนิดนี้ได้ ซึ่งใน 89C51 จะมีร็อมชนิดอีอีพร็อม (EEPROM หรือ Eletrical Erasable Program Read Only Memory) อยู่ภายในจำนวน 4K ไบท์

2. ดาต้าเมมโมรี (Data Memory) เป็นหน่วยความจำที่ใช้สำหรับพักข้อมูล เก็บข้อมูลแล้วเรียกมาใช้ใหม่ได้ในระหว่างการทำงาน ดังนั้น หน่วยความจำนี้ จะต้องเป็น RAM ซึ่งใน 89C51มีแรม (RAM) อยู่ภายใน จำนวน 128 ไบท์ (ตั้งแต่ 00H ถึง 7FH)

รายละเอียดอื่นๆดูได้ในภาคผนวก

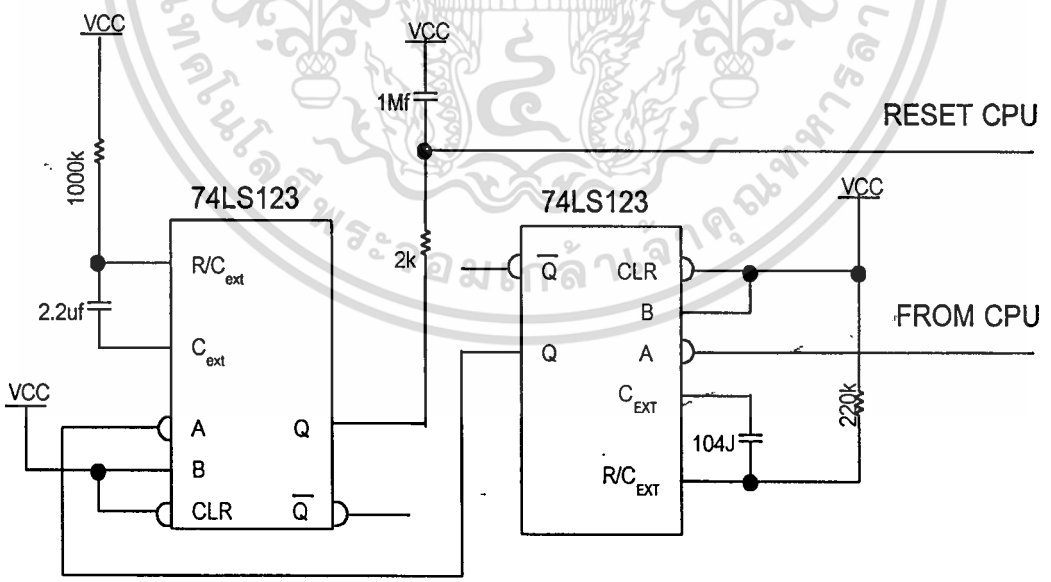


รูปที่ 3.2 โครงสร้างหน่วยความจำของ MCS-51

บทที่ 4
วงจรตรวจเช็คสถานะของซีพียู

เป็นวงจรที่ใช้ตรวจสอบสถานะการทำงานของซีพียูเนื่องจากขณะที่วงจรทำงานอยู่อาจเกิดความผิดพลาดในการทำงาน หรืออาจเกิดอาการแฮงค์ (hang)ของซีพียูทำให้ซีพียู ไม่สามารถควบคุมการทำงานได้อย่างถูกต้อง ซึ่งทำให้คอมเพรสเซอร์ทำงานผิดปกติ อาจจะก่อให้เกิดความเสียหายได้ ดังนั้น วงจรควบคุมการทำงานของเครื่องปรับอากาศ จึงมีความจำเป็นที่จะต้องมี วงจรตรวจเช็คสถานะซีพียูเพื่อทำหน้าที่ในการตรวจสอบเมื่อเกิดข้อผิดพลาด หรือเกิดการแฮงค์จากโปรแกรม วงจรตรวจเช็คสถานะ ซีพียู จะรับสัญญาณผิดปกติจากซีพียูได้ และจะสร้างสัญญาณรีเซท ส่งไปที่ซีพียู เพื่อให้ซีพียูเริ่มทำงาน ในสถานะเริ่มต้นใหม่ (INITIAL CONDITION) จึงทำให้โปรแกรมเริ่มต้นที่จะทำงานต่อไปอีกทีหนึ่ง

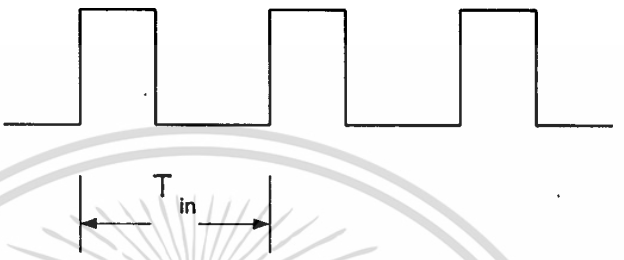
วงจรตรวจเช็คสถานะซีพียูนี้ใช้ไอซี (IC หรือ Intregral circuit)เบอร์ 74LS123 เป็นอุปกรณ์หลัก ดังรูปที่ 4.1



รูปที่ 4.1 วงจรตรวจเช็คสถานะซีพียู

หลักการทำงานของวงจรตรวจเช็คสถานะซีพียู

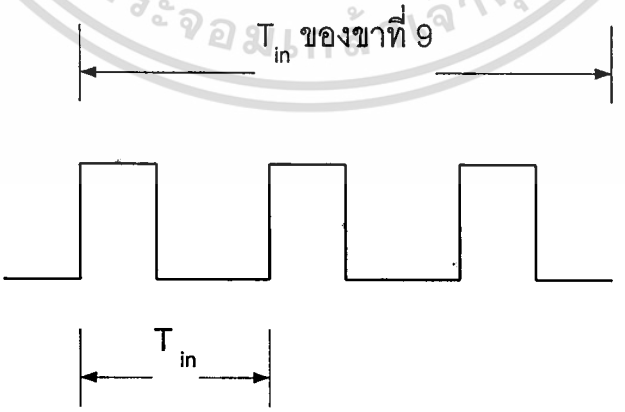
ภายในไอซีเบอร์ 74LS123**ประกอบด้วยวงจร** Retriggerable Monostable 2 **วงจร หลักการทำงานคือวงจรจะรับสัญญาณจากซีพียูเข้าที่ขา** 9 **ซึ่งเป็นอินพุทของโมโนสเตเบิล ตัวที่** 1 **สัญญาณแสดงดังรูปที่** 4.2



รูปที่ 4.2 **แสดงลักษณะสัญญาณที่ส่งมาจากซีพียู**

จากวงจร ค่าเวลาคงที่ (Tc หรือ Time Constant) หาจาก $Tc = 0.37R_1C_1$ ซึ่งมีค่าเท่ากับ 0.814 วินาที

เมื่อวงจรทำงานปกติ สัญญาณที่ส่งมาจากซีพียู จะมีคาบเวลา T_{in} น้อยกว่าค่าค่าเวลาคงที่ (Tc) เพื่อที่จะรับสัญญาณทริก (Trig) ขอบขาลงของสัญญาณที่ส่งมาจากซีพียู ดังรูปที่ 4.3



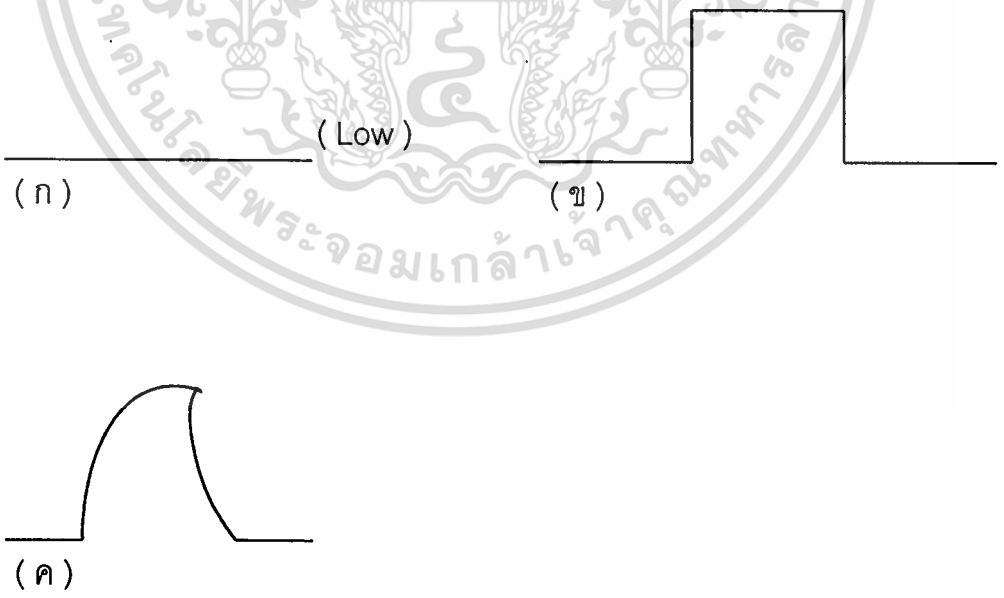
รูปที่ 4.3 **แสดงค่าเวลาคงที่** (Time Constant) **ของขาที่** 9

เมื่อรับสัญญาณทริกขอบขาขึ้นจากซีพียูแล้ว โมโนสเตเบิล ตัวแรกจะส่งเอาท์พุทออกที่ขา 5 มีสถานะสูง (High) ไปยัง โมโนสเตเบิล ตัวที่ 2 ที่ขา 1 โดยมีค่าค่าเวลาคงที่คำนวณจาก $Tc = 0.37R_2C_2$ ซึ่งมีค่าเท่ากับ 0.00814 วินาที

เมื่อเกิดสภาวะผิดปกติขึ้นกับซีพียูทำให้สัญญาณที่ส่งมาจากซีพียูขาดหายไปนานมากกว่าค่าของค่าเวลาคงที่ที่กำหนด โมโนสเตเบิล ตัวแรกจะส่งสถานะต่ำ (Low) ไปยังโมโนสเตเบิล ตัวที่ 2 ที่ขา 1 โดยมีลักษณะสัญญาณดังรูป

รูปที่ 4.4 ลักษณะสัญญาณที่ออกมาทริก (Trig) ขา 1

เนื่องจากขา 1 ต้องการสัญญาณทริกขอบขาลง เพื่อที่จะสร้างสัญญาณไปรีเซทซีพียู ทางขา 13 ผ่าน R_3C_3 เพื่อที่จะได้สัญญาณสูง (High) ทุกครั้งเมื่อเปิดเครื่อง หรืออีกนัยหนึ่ง คือมีการรีเซทก่อนทุกครั้งเมื่อมีการเปิดเครื่อง (Power On) ลักษณะของสัญญาณจากขา 13 ที่ส่งไปซีพียูแสดงดังรูป

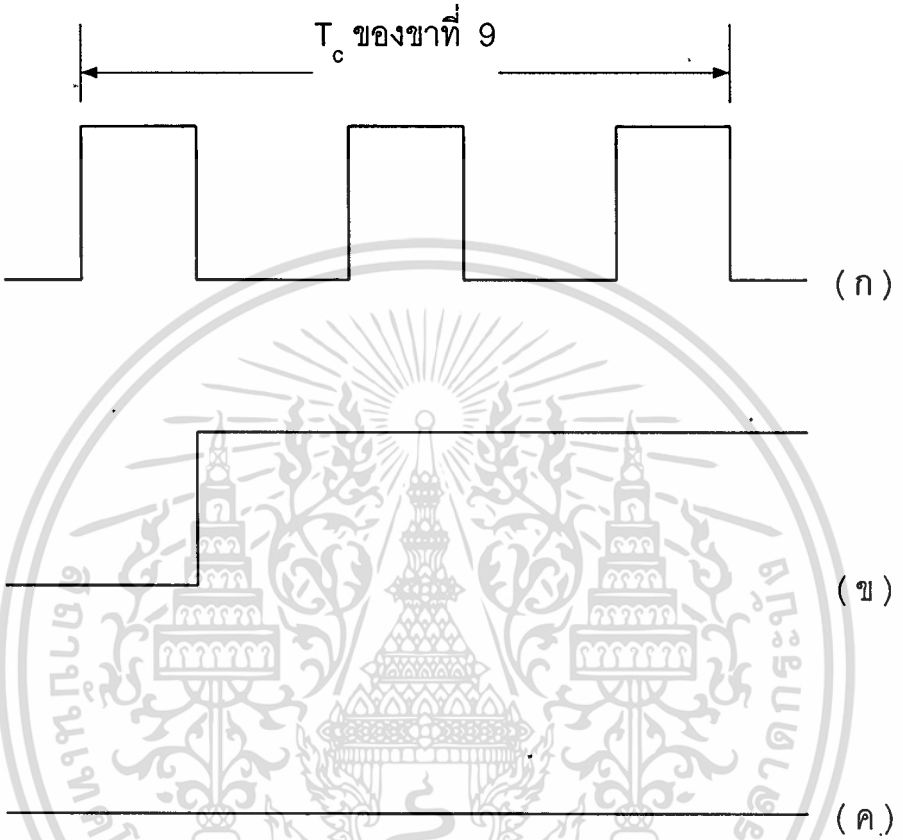


รูปที่ 4.5 ลักษณะของสัญญาณที่ส่งไปรีเซตซีพียู จากขา 13

(ก) สัญญาณขณะซีพียูทำงานปกติ (ข) สัญญาณขณะซีพียูทำงานผิดปกติ

(ค) สัญญาณจากขา 13 ที่ผ่าน RC แล้ว

สรุปลักษณะของสัญญาณที่สำคัญเมื่อซีพียูทำงานปกติ และเมื่อซีพียูทำงานผิดปกติได้ดังรูปที่ 4.6 และ รูปที่ 4.7

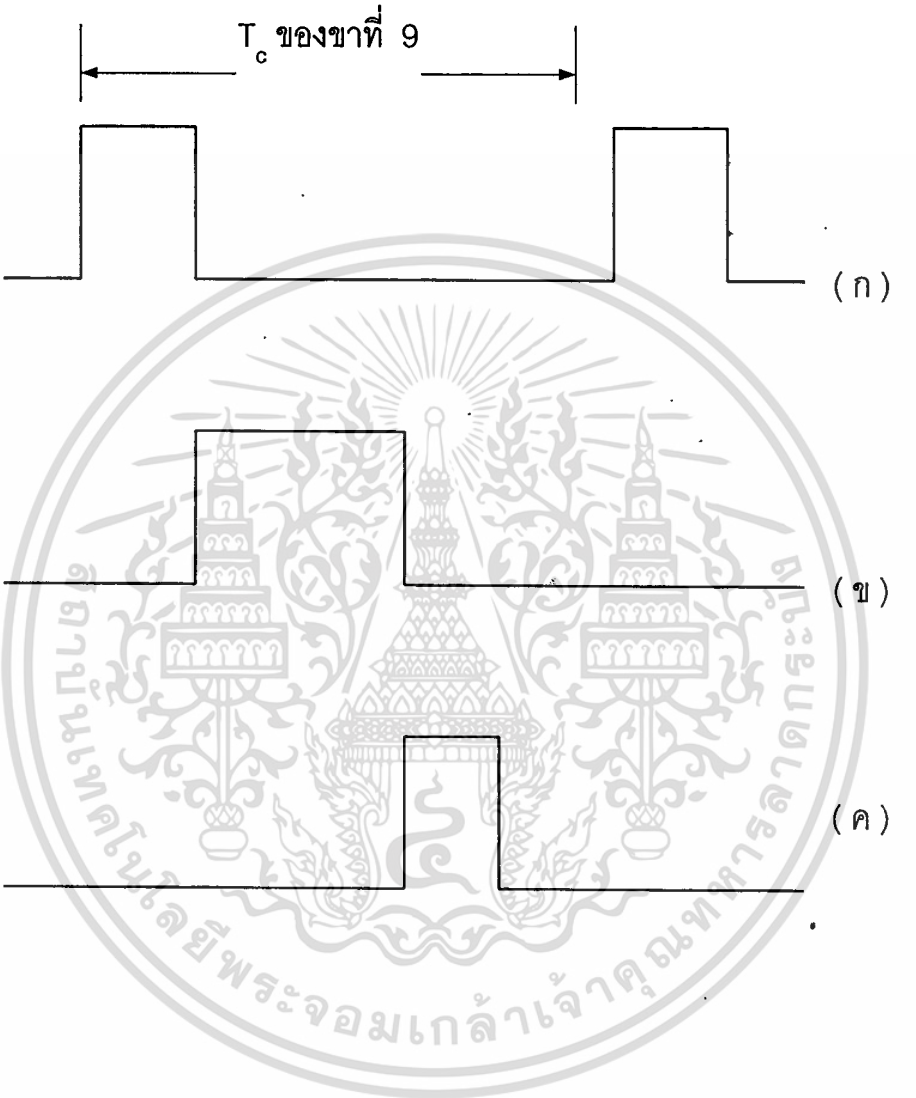


รูปที่ 4.6 ลักษณะของสัญญาณที่สำคัญเมื่อซีพียูทำงานปกติ

(ก) สัญญาณจากซีพียูเข้าขา 9

(ข) สัญญาณจากขา 5 ไปขา 1

(ค) สัญญาณเอาท์พุทจากขา 13 ไปรีเซตซีพียู



รูปที่ 4.7 ลักษณะของสัญญาณที่สำคัญเมื่อซีพียูทำงานผิดปกติ

(ก) สัญญาณจากซีพียูเข้าขา 9

(ข) สัญญาณจากขา 5 ไปยังขา 1

(ค) สัญญาณเอาท์พุทจากขา 13 ไปรีเซตซีพียู

บทที่ 5
วงจรตรวจจับค่าอุณหภูมิ (TEMPERATURE SENSOR)

เป็นวงจรสำหรับตรวจจับค่าอุณหภูมิของห้องเพื่อส่งเป็นข้อมูลไปให้ซีพียูทำการประมวลผลตามที่โปรแกรมไว้ ในการที่จะควบคุมการทำงานของคอมเพรสเซอร์และพัดลม เพื่อให้อุณหภูมิของห้องอยู่ที่ค่าที่ผู้ใช้ตั้งไว้

องค์ประกอบหลักของวงจรคือ ตัวเทอร์มิสเตอร์ ซึ่งเป็นประเภท NEGATIVE COEFICIENT คือเมื่ออุณหภูมิสูงความต้านทานจะต่ำ และเมื่ออุณหภูมิต่ำความต้านทานจะสูงเราใช้เทอร์มิสเตอร์ เป็นตัวตรวจจับอุณหภูมิ โดยใช้ ไอซี LM555 ประกอบเป็นวงจรโมโนสเตเบิลซึ่งค่าเวลาคงที่ของวงจรจะขึ้นอยู่กับค่าความต้านทานของเทอร์มิสเตอร์ และค่า R , C ที่ตั้งเอาไว้ ซีพียูจะส่งสัญญาณทริกให้กับวงจร และจะรับเอาสัญญาณพัลซ์ ซึ่งเป็นเอาท์พุทของวงจรกลับไปเพื่อทำการประมวลผล โดยจะทำการเปลี่ยนสัญญาณพัลซ์ ดังกล่าวให้เป็นค่าของอุณหภูมิที่สอดคล้องกันเสียก่อน

หลักการทำงานในการวัดค่าอุณหภูมิ

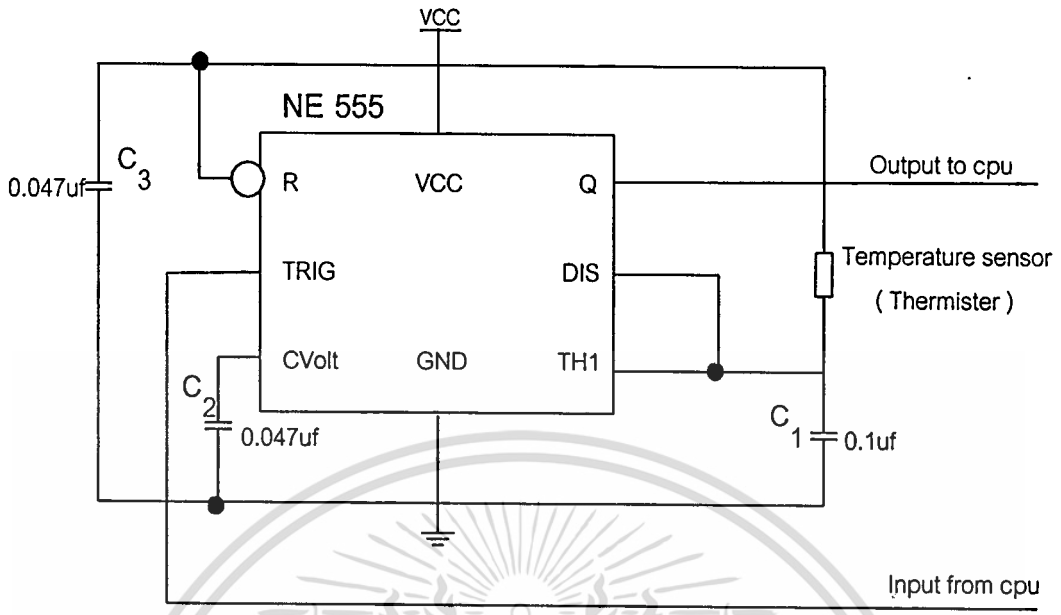
การวัดค่าอุณหภูมิสามารถแบ่งได้เป็น 2 ส่วน คือ ส่วนิของวงจรและส่วนของโปรแกรม

1.ส่วนวงจร จะใช้วงจรโมโนสเตเบิลของ ไอซี 555 ซึ่งมีตัวเทอร์มิสเตอร์ ตัวต้านทาน และตัวเก็บประจุ เป็นส่วนประกอบดังรูปที่ 5.1

ไอซี 555 จะให้สัญญาณพัลซ์เป็นเอาท์พุท ออกมาทุกครั้งที่ได้รับสัญญาณทริก จากซีพียู โดยที่ความกว้างของสัญญาณจะหาได้จาก

$$T = 1.1RC$$

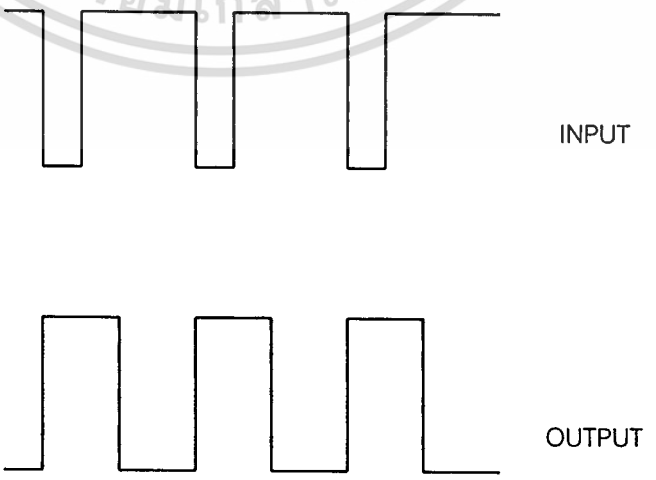
โดยที่ T คือความกว้างของพัลซ์ , C มีค่าเทากับ 0.1 uF , R เป็นผลรวมแบบขนานของความต้านทานที่ปรับค่าได้ และความต้านทานของเทอร์มิสเตอร์



รูปที่ 5.1 วงจรตรวจวัดค่าอุณหภูมิ

ความต้านทานของเทอร์มิสเตอร์จะเปลี่ยนไปตามอุณหภูมิ ดังได้กล่าวมาแล้ว จึงทำให้เราได้เอาท์พุทจากวงจรที่มีความกว้างของสัญญาเปลี่ยนแปลงไปตามอุณหภูมิด้วย

วงจร ไอซี 555 จะให้เอาท์พุท ออกมาเมื่อสัญญาณที่มาทริกจากซีพียูเปลี่ยนจาก 1 เป็น 0 ดังรูปที่ 5.2

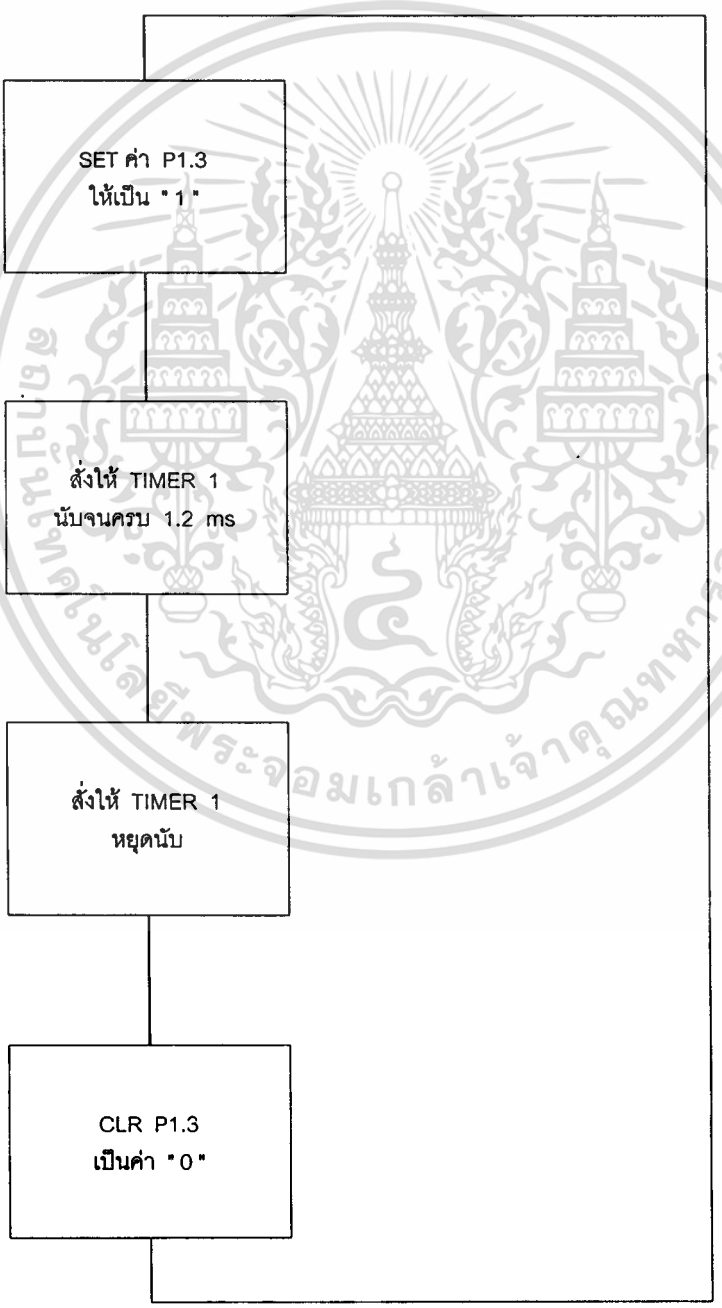


รูปที่ 5.2 ลักษณะของสัญญาณพัลซ์

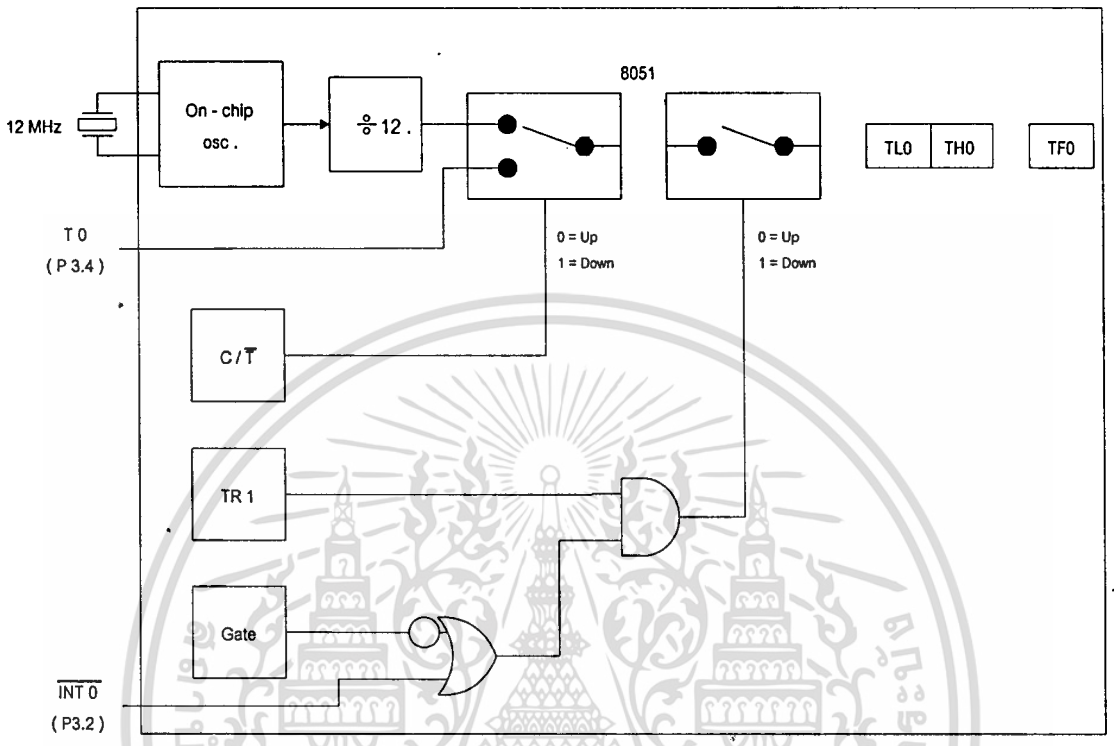
2.ส่วนของโปรแกรมมี 2 ส่วนคือ

2.1 โปรแกรมการสร้างสัญญาณทริก เราเลือกที่จะส่งสัญญาณทริกออกมาที่ขา P1.3 ของซีพียู และเขียนโปรแกรมให้ไทเมอร์ 1 (TIMER 1) เป็นตัวนับเวลา 1.2 มิลลิวินาที เมื่อครบแล้วก็ให้ส่งสัญญาณทริก คือเปลี่ยนสถานะของขา P1.3 จาก 1 เป็น 0 และจาก 0 เป็น 1 แล้วทำการรับสัญญาณ พัลซ์ เข้าทางขา INT0 ของซีพียูเพื่อทำการนับความกว้างของสัญญาณต่อไป

2.2 โปรแกรมนับความกว้างของสัญญาณพัลซ์ จะใช้ไทเมอร์ 0 (TIMER 0) เป็นตัวนับ



รูปที่ 5.3 โฟลชาร์ต แสดงการสร้างสัญญาณทริก



รูปที่ 5.4 วงจรแสดงการทำงาน Timer Counter ในการนับความกว้างของสัญญาณ พัลซ์

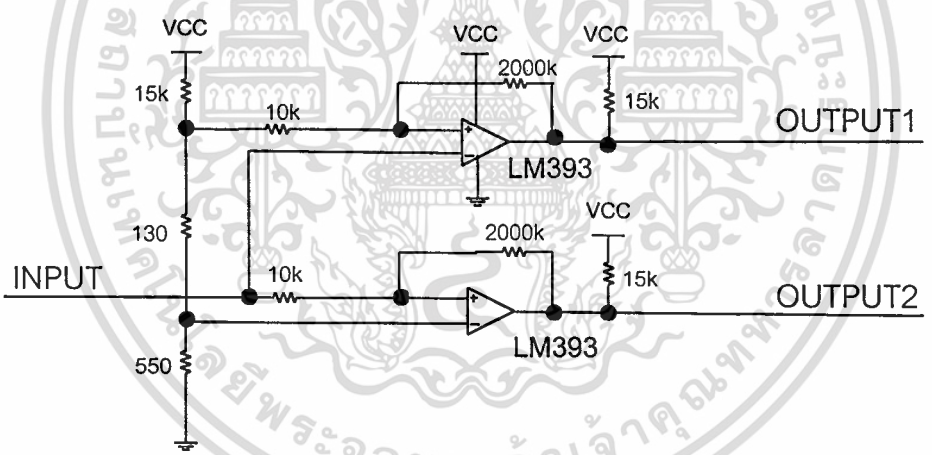
จากรูป ถ้าเราตั้งค่าให้ TR0 เป็น 1, ตั้ง GATE เป็น 1 แล้ว เมื่อ ซีพียูได้รับสัญญาณพัลซ์เข้าทางขา INT0 ตัวไทเมอร์ 0 ก็จะเริ่มนับสัญญาณนาฬิกา เมื่อสัญญาณพัลซ์ที่เข้ามามีสถานะเป็น 1 และจะหยุดการนับเมื่อสัญญาณนั้นเปลี่ยนจาก 1เป็น 0 ดังแสดงในรูปที่ 5.5

รูปที่ 5.5 โฟลชาร์ต แสดงการนับความกว้างของสัญญาณ

บทที่ 6

วงจรตรวจจับระดับแรงดันไฟฟ้า

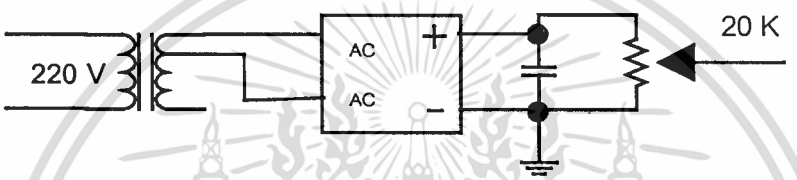
เนื่องจากแรงดันไฟฟ้ากระแสสลับ 220 โวลท์ (V หรือ Volt AC)ในแต่ละย่านที่อยู่อาศัย อาจจะมีระดับแรงดันไม่คงที่ โดยขึ้นอยู่กับปัจจัยหลายอย่าง เช่น ความหนาแน่นของการใช้ไฟฟ้าในแต่ละช่วงเวลา ซึ่งคอมเพรสเซอร์จะทำงานได้ในช่วง 189-250 โวลท์ AC เพื่อทำให้คอมเพรสเซอร์มีอายุการใช้งานมากขึ้น หรือ ลดความเสียหายที่อาจเกิดขึ้น จึงจำเป็นที่จะต้องมีวงจรตรวจจับระดับแรงดันไฟฟ้าว่าอยู่ในขอบเขตของแรงดันที่ทำให้คอมเพรสเซอร์ทำงานได้ปกติหรือเป็นอันตรายต่อคอมเพรสเซอร์ โดยวงจรนี้ใช้คอมพาราเตอร์ เบอร์ LM393 ประกอบเป็นวงจรวินโดว์คอมพาราเตอร์ (Window Comparator) ซึ่งมีลักษณะวงจรดังรูป



รูปที่ 6.1 ลักษณะของวงจรตรวจจับระดับของแรงดันไฟฟ้า

หลักการทำงานของวงจร

เราจะกำหนดให้ค่าแรงดันไฟฟ้าที่คอมเพรสเซอร์สามารถทำงานได้ปกติ ให้อยู่ในช่วง 189-250 โวลท์ ถ้าแรงดันไฟบ้านอยู่นอกช่วงนี้ ให้ตัดการทำงานของคอมเพรสเซอร์ทันที ในการตรวจเช็คระดับแรงดัน ทำโดยการผ่านไฟบ้านเข้าสู่หม้อแปลงแบบแปลงไฟลงแล้วทำการเร็กติไฟ (Rectify) ให้เป็นไฟตรง (DC) เพื่อนำไปเป็นอินพุทของวงจรตรวจจับระดับแรงดันไฟฟ้า ซึ่งวงจรแปลงสัญญาณไฟสลับ (AC)เป็นไฟตรง (DC) แสดงดังรูปที่ 6.2



รูปที่ 6.2 วงจรแปลงสัญญาณ AC เป็น DC

จากรูปที่ 6.1 วงจรตรวจจับระดับแรงดันไฟฟ้าเมื่อ R_1=220, R_2=130, R_3=560 โอห์ม (OHM) วงจรจะให้เอาท์พุทเป็นสถานะสูงทั้งคู่ (สภาวะปกติ) เมื่ออินพุทอยู่ในช่วง 3.1 ถึง 3.8 โวลท์ โดยมีช่วง hysteresis 0.2 โวลท์ กำหนดโดยค่า R 2.2 เมกะโอห์ม ถ้าต้องการช่วง hysteresis มากกว่านี้ให้ลดค่าความต้านทาน (R)

จากรูป 6.2 วงจรแปลงสัญญาณไฟสลับเป็นไฟตรงเพื่อเป็นอินพุท โดยรับไฟสลับ 220 โวลท์ ที่ใช้กันตามปกติ มาผ่านหม้อแปลงลงเป็น 12 โวลท์ แล้วทำการเร็กติไฟให้เป็นไฟตรงให้ได้แรงดันไฟฟ้าประมาณ 3.4 โวลท์

การทำงานของวงจร อธิบายได้ดังนี้

<u>เมื่ออินพุทมีค่าเพิ่มขึ้น</u>

เริ่มที่อินพุทมีระดับต่ำกว่า 3.2 โวลท์ จะให้เอาท์พุท 1 เป็นสถานะสูง (High) และเอาท์พุท 2เป็นสถานะต่ำ (low หรือ สภาวะ under)

เมื่ออินพุทเพิ่มขึ้นเท่ากับ 3.3 โวลท์ จะให้เอาท์พุท 1 เป็นสถานะสูงและเอาท์พุท 2 เป็นสถานะสูง (สภาวะ normal)

เมื่ออินพุทเพิ่มขึ้นเท่ากับหรือมากกว่า 3.9 โวลท์จะให้เอาท์พุท 1 เป็นสถานะต่ำ และเอาท์พุท 2 เป็นสถานะสูง (สภาวะ over)

<u>เมื่ออินพุทมีค่าลดลง</u>

เริ่มที่อินพุทมีระดับสูงกว่า 3.8 โวลท์ จะให้เอาท์พุท 1 เป็นสถานะต่ำ และเอาท์พุท 2เป็นสถานะสูง (สภาวะ over)

เมื่ออินพุทลดลงเท่ากับ 3.7 โวลท์ จะให้เอาท์พุท 1 เป็นสถานะสูง และเอาท์พุท 2 เป็นสถานะสูง (สภาวะ normal)

เมื่ออินพุทลดลงเท่ากับ3.1 โวลท์ จะให้เอาท์พุท 1 เป็นสถานะสูง และเอาท์พุท 2 เป็นสถานะต่ำ (สภาวะ under)

สรุปสภาวะแสดงดังตารางดังนี้

	UNDER	ปกติ	OVER	ขา
OUTPUT1	1	1	0	1
OUTPUT2	0	1	1	7

ตารางที่ 6.1 แสดงสัญญาณในสภาวะแรงดันต่างๆ

บทที่ 7
ส่วนรับคำสั่งและแสดงผล (INPUT BOTTON AND DISPLAY)

ส่วนรับคำสั่งและแสดงผล จะเป็นส่วนที่ช่วยในการสื่อสารระหว่างผู้ใช้ กับระบบโดยระบบจะรับคำสั่งจากผู้ใช้ผ่านทางปุ่มรับคำสั่ง (INPUT BOTTON) และจะแสดงดถานะต่างๆให้ผู้ใช้ทราบทางส่วนแสดงผล ซึ่งมีทั้ง LED และเซเวนเซกเมนท์ (SEVEN SEGMENT) ส่วนรับคำสั่งและแสดงผลของระบบดังต่อไปนี้

1. POWER ON/OFF

เครื่องจะเริ่มทำงานเมื่อกดปุ่ม POWER และจะแสดงไฟ LED ที่ตำแหน่ง ON ในกรณีที่แรงดันไฟที่ป้อนให้แก่คอมเพรสเซอร์อยู่ในระดับผิดปกติ และแสดงที่ตำแหน่ง " OV/UN " ในกรณีที่แรดันสูงหรือต่ำกว่าปกติ โดยจะแสดงไปเรื่อยๆจนกว่าแรงดนจะกลับเข้าสู่สภาวะปกติ สวนไฟที่ตำแหน่ง "COMP" นั้น แสดงว่าขณะนั้นคอมเพรสเซอร์กำลังทำงานอยู่หรือไม่ โดยถ้าไฟติด ก็แสดงว่ากำลังทำงานอยู่ แต่ถ้าไฟดับ ก็แสดงว่าไม่ได้ทำงาน

2. FAN

ผู้ใช้สามารถเลือกการเปิดพัดลมได้ 2 แบบคือ แบบเปิดโดยที่ผู้ใช้ต้องเลือกระดับความเร็วของพัดลมเอง และแบบเปิดโดยที่ระดับความเร็วของพัดลมจะถูกเปลี่ยนเองโดยอัตโนมัติ ความเร็วจะมี 3 ระดับคือ เบา ,ปานกลาง และแรง โดยจะแสดงทางไฟ LED 3 ดวง คือ "LOW" "MEDIUM" "HI" ตามลำดับ ในกรณีที่ผู้ใช้เลือกแบบอัตโนมัติ จะมีไฟแสดงที่ตำแหน่ง "AUTO" และไฟแสดงระดับความเร็วของพัดลมที่ถูกปรับโดยอัตโนมัติ ให้สอดคล้องกับความแตกต่างของอุณหภูมิห้องและอุณหภูมิที่ตั้งไว้ ดังนี้

ถ้าอุณหภูมิห้องแตกต่างกับที่ตั้งไว้มาก 2 องศา ก็จะปรับให้พัดลมหมุนเร็ว เพื่อเร่งกระจายความเย็นให้ทั่วถึงทั้งห้องอย่างรวดเร็ว

ถ้าอุณหภูมิห้องอยู่ในช่วงค่าที่ตั้งไว้จนถึง ค่าที่ตั้งไว้ +2 องศา ก็จะปรับให้ความเร็วปานกลาง

ถ้าอุณหภูมิต่ำกว่าที่ตั้งไว้ก็จะปรับให้มีความเร็วต่ำสุด

3. MODE

เป็นปุ่มสำหรับให้ผู้ใช้เลือกการทำงานดังต่อไปนี้

- เมื่อครั้งแรกไฟ LED จะแสดงที่ "TEMP" ซึ่งหมายความว่าเซเวนเซกเมนท์จะแสดงค่าของอุณหภูมิที่วัดได้ในขณะนั้น

- เมื่อกดครั้งแรกไฟ LED จะแสดงที่ "SET" และเซเวนเซกเมนท์จะแสดงค่าอุณหภูมิที่ตั้งไว้ ถ้าผู้ใช้ต้องการที่จะตั้งที่ค่าใหม่ก็สามารถทำได้โดยกดปุ่ม เพิ่ม หรือ ลดค่า

- เมื่อกดครั้งที่ 2 ไฟ LED จะแสดงที่ " TIMER 0" และเซเวนเซกเมนท์จะแสดงค่าเวลาชั่วโมงที่ตั้ง ซึ่งจะทำให้ผู้ใช้ตั้งเวลาในการปิดเครื่องเองโดยอัตโนมัติ โดยผู้ใช้จะต้องใส่ค่าเป็นชั่วโมงที่ต้องการให้เครื่องปิดหลังจากการตั้งโดยการกดปุ่มเพิ่มหรือลดค่า

- เมื่อกดครั้งที่ 3 ไฟ LED "TIMER 1" และ เซเวนเซกเมนท์จะแสดงค่าชั่วโมงที่จะตั้ง ซึ่งจะทำให้ผู้ใช้ตั้งเวลาในการเปิดเครื่องเองโดยอัตโนมัติ ในกรณีที่เครื่องกำลังปิดอยู่โดยการกดปุ่มเพิ่มหรือลดค่า

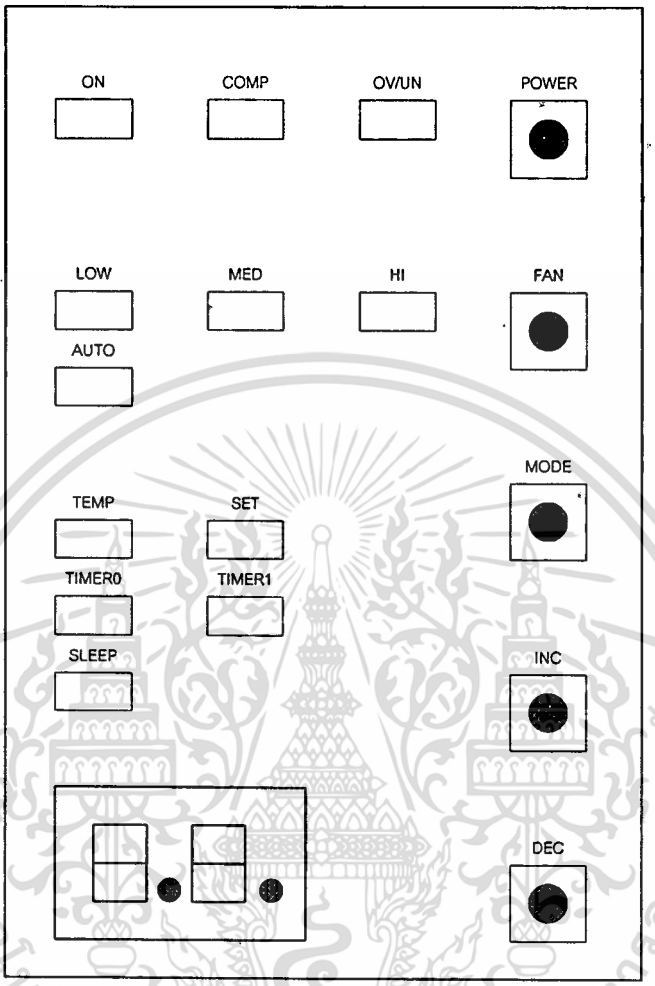
ผู้ใช้สามารถที่จะตั้งเวลาให้ปิด และตั้งเวลาให้เปิดต่ออีกได้ เพียงแต่ป้อนค่าเป็นชั่วโมงใน mode timer 0 และตามด้วยการป้อนค่าใน mode timer 1 และในทางตรงกันข้าม ในกรณีที่เครื่องกำลังปิดอยู่ก็สามารถตั้งให้เปิด และตั้งให้ปิดกลับคืนได้ ด้วยการป้อนค่าใน mode timer 1 ก่อน แล้วจึงย้อนกลับไปตั้งค่า ใน mode timer 0 ได้

- เมื่อกดครั้งที่ 4 ไฟจะแสดงที่ " SLEEP" ซึ่งเป็น mode พิเศษ สำหรับใช้ควบคุมเครื่อง ในขณะที่ผู้ใช้ต้องการใช้เครื่องในยามหลับนอน เนื่องจากในขณะที่คนเรากำลังหลับอยู่การใช้พลังงานจะน้อยลง จึงอาจทำให้รู้สึกว่าอากาศเย็นเกินไปสำหรับค่าอุณหภูมิที่ตั้งเอาไว้ในตอนก่อนนอน ดังนั้นถ้าผู้ใช้เลือก mode นี้ (โดยการกดปุ่มเพิ่ม "^") ค่าเซ็ทพอย์ตก็จะเพิ่มขึ้นเองอัตโนมัติ 1 องศา ทุกๆ1 ชั่วโมง แต่เพิ่มขึ้นเพียง 2 องศา แล้วก็จะคงที่ตลอดไป แล้วระบบก็จะออกจาก sleep mode โดยอัตโนมัติ แต่ถ้าจะต้องการออกจาก sleep mode ก็เพียงเข้า mode นี้แล้วกดปุ่มลดค่า

- คุณสมบัติพิเศษ หน่วงเวลา 3 นาที

บางครั้งกระแสไฟฟ้าอาจดับในช่วงระยะเวลาสั้นๆ แล้วกลับติดขึ้นมาใหม่ ซึ่งอาจทำความเสียหายต่อ คอมเพรสเซอร์ได้ โปรแกรมการทำงานจะมีการหน่วงเวลาการเริ่มทำงานของคอมเพรสเซอร์ออกไป 3 นาที เพื่อป้องกันความเสียหายที่อาจเกิดขึ้น แต่ในการเปิดเครื่องตามปกติจะไม่มีการหน่วงเวลา

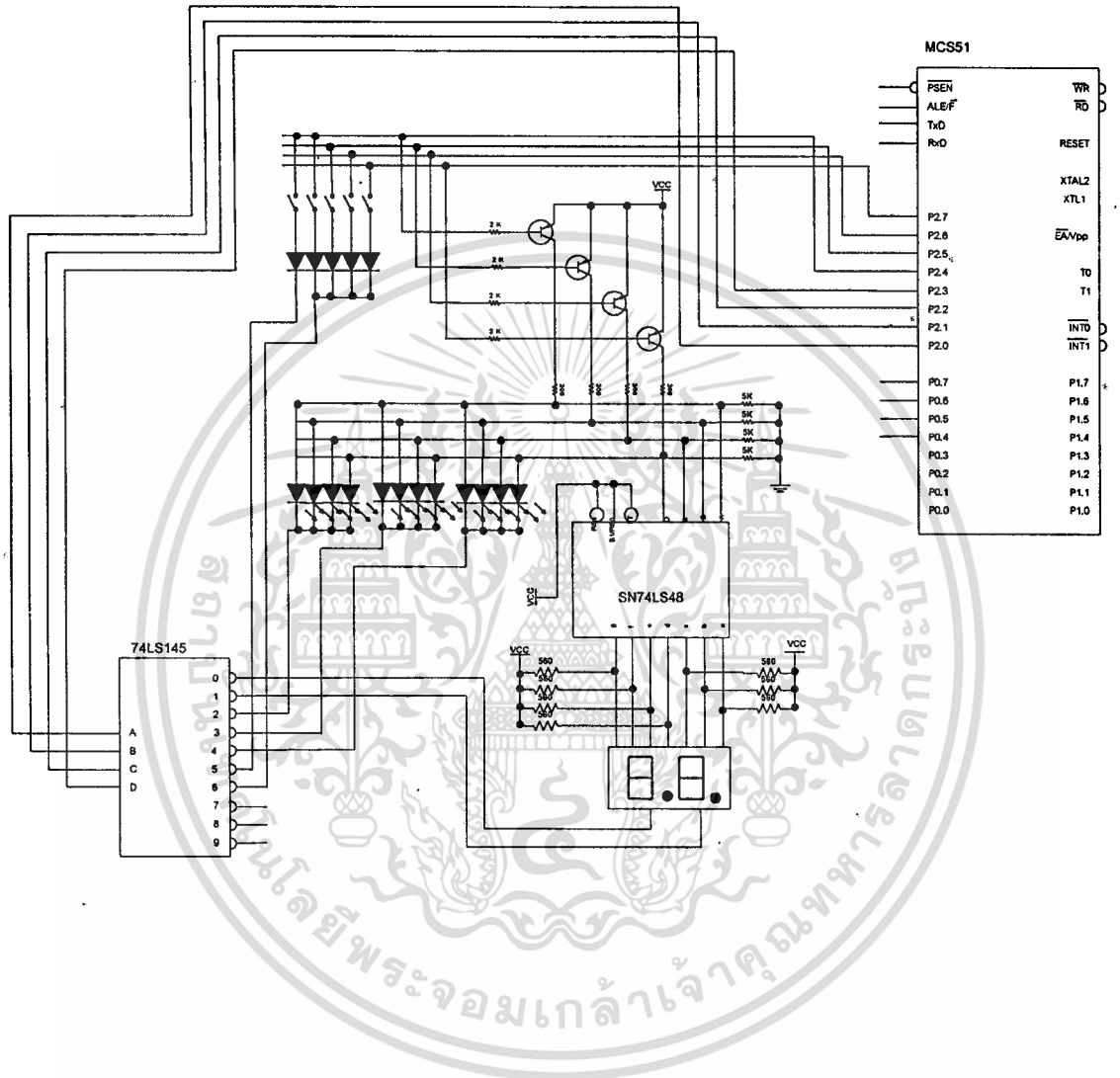
ส่วนรับคำสั่งและแสดงผล ได้ทำการออกแบบดังรูปที่ 7.1



รูปที่ 7.1 ส่วนรับคำสั่งและแสดงผล

หลักการทำงานของวงจรในการรับคำสั่งและการแสดงผล

วงจรประกอบไปด้วย LED 12 ดวง เซเวนเซกเมนท์ 2 หลักสำหรบแสดงผลและใช้ ปุ่ม 5ปุ่ม สำหรับรับคำสั่ง การต่อวงจรแสดงดังรูปที่ 7.2

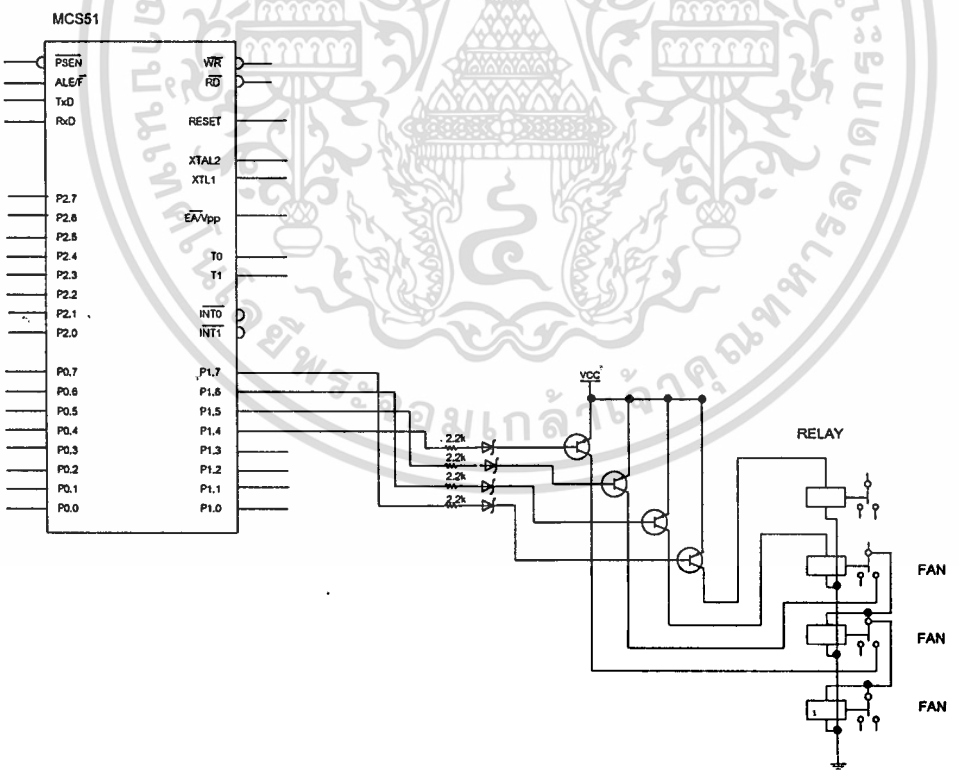


รูปที่ 7.2 วงจรส่วนรับคำสั่งและแสดงผล (input button and display)

วงจรแสดงผลประกอบไปด้วย LED 3 ชุด ชุดละ 4 ดวง และเซเวนเซกเมนท์ 2 หลัก วงจรใช้ ไอซี 74145 เป็นตัวถอดรหัส เพื่อสร้างสัญญาณสแกน (scan) แสดงค่าออกทาง LED แต่ละชุดและเซเวนเซกเมนท์ ในส่วนของเซเวนเซกเมนท์ แต่ละหลักจะต่อคอมมอน (common) กับสัญญาณสแกนโดยมีไอซี 7448 เป็นตัวถอดรหัส BCD เพื่อส่งค่าไปยังแต่ละ segment โดยจะใช้ไอซี 7448 ร่วมกันทั้ง 2 หลัก

ส่วนปุ่มรับคำสั่ง แบ่งออกเป็น 2 ชุด ชุดละ 4 อันและ ชุดละ 5 อัน ตามลำดับ โดยที่สัญญาณสแกน จะได้มาจาก ไอซี 74145 เช่นเดียวกับที่ส่วนแสดงผล (display)

ในการส่วนสัญญาณของซีพียูเพื่อทำการขับคอมเพรสเซอร์และพัดลม โดยส่งสัญญาณออกทางขา P1.4 สำหรับคอมเพรสเซอร์และ P1.5, P1.6, P1.7 สำหรับพัดลมความเร็วต่ำ (low) กลาง (medium) สูง (high) ตามลำดับ โดยสั่งให้ทรานซิสเตอร์ไป เปิด - ปิด รีเลย์ ซึ่งรีเลย์ของพัดลม 3 ตัว จะไม่สามารถเปิดพร้อมกันได้ เพื่อป้องกันความเสียหายในขดลวดพัดลม โดยนำรีเลย์พัดลม 2 ตัว ที่เหลือ ต่ออนุกรมกับขั้ว นอร์มอนรีโคลส (normally closed) ของรีเลย์ตัวก่อนหน้าของมัน โดยที่วงจรแสดงดังรูปที่ 7.3



รูปที่ 7.3 วงจรขับคอมเพรสเซอร์และพัดลม

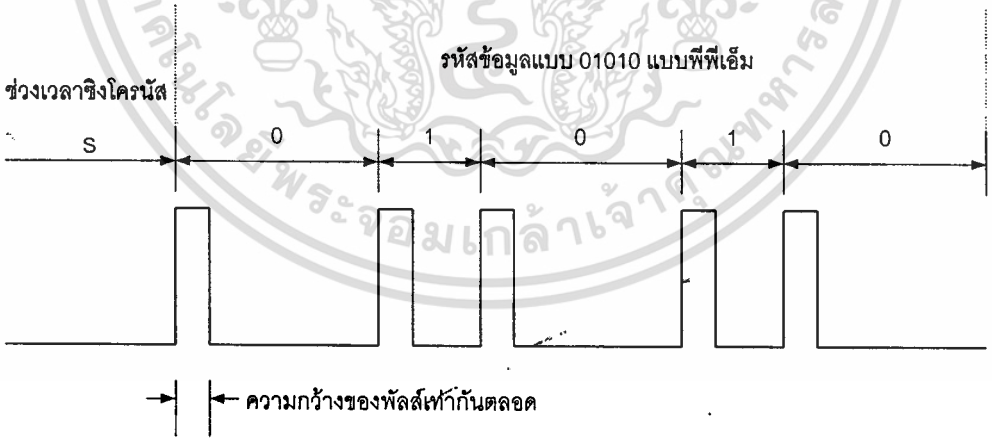
บทที่ 8
วงจรรับคำสั่งจากรีโมทและวงจรรีโมท (REMOTE CONTROL)

รีโมท (Remote) เป็นส่วนที่ช่วยอำนวยความสะดวกแก่ผู้ใช้ในการควบคุมระบบจากระยะไกล หลักการทำงานและออกแบบรีโมทคอนโทรลแทบทุกชนิด ล้วนมีพื้นฐานในทำนองเดียวกันทั้งสิ้น ไม่ว่าจะเป็นการควบคุมโดยตรงด้วยสายไฟ การควบคุมแบบไร้สายโดยใช้วิทยุ หรือคลื่นแสง ตลอดจนการควบคุมผ่านทางสายไฟฟ้าภายในบ้าน เพียงแต่ใช้ชุดวงจรรับส่ง สัญญาณควบคุม หรือคลื่นพาห์ที่แตกต่างกันเท่านั้น

ในโครงงานนี้ใช้รูปแบบของการส่งรหัสควบคุมแบบดิจิตอลที่ค่อนข้างเป็นที่นิยมกันมาก คือระบบพีพีเอ็ม หรือ พัลซ์โพสิชั่นมอดูเลชั่น (PPM : Pulse Position Modulation)

พีพีเอ็ม

สัญญาณชนิดพีพีเอ็ม (PPM :Pulse Position Modulation)เกิดจากการมอดูเลทสัญญาณในลักษณะของตำแหน่งพัลซ์ คือขนาดความกว้างของพัลซ์จะมีค่าเท่ากันตลอด และไม่มีความสำคัญในการบ่งบอกชนิดของข้อมูลเลย แต่จะใช้คาบเวลาหรือพีเรียด (period) ของพัลซ์แต่ละลูกเป็นตัวกำหนดชนิดของข้อมูล เช่น ข้อมูลที่เป็น "1" แทนด้วยพัลซ์ที่มีคาบเวลาคงที่ค่าหนึ่ง ซึ่งแตกต่างจากคาบเวลาของพัลซ์ที่แสดงข้อมูลที่เป็น "0" ดังแสดงในรูปที่ 8.1



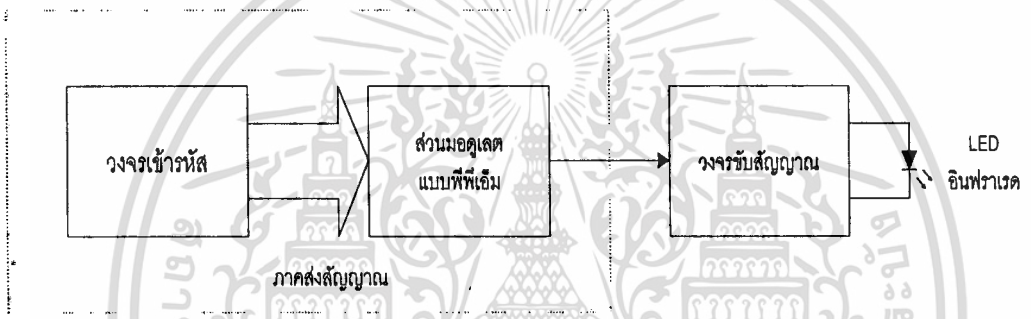
รูปที่ 8.1 แสดงรูปแบบของสัญญาณพีพีเอ็ม

โดยหลักการแล้ว การมอดูเลทสัญญาณแบบพีพีเอ็ม จะใช้การแบ่งช่วงสัญญาณด้วยคาบเวลาที่เท่ากัน แต่จุดเวลาที่แสดงสัญญาณพัลซ์ต่างกัน เช่น หากสัญญาณเป็น 0 สัญญาณพัลซ์จะปรากฏ ณ ตำแหน่งกึ่งกลางของคาบเวลาที่กำหนด ถ้าหากสัญญาณมีแอมปลิจูดเป็นบวก สัญญาณพัลซ์จะปรากฏในตำแหน่งที่ล้ำออกไปทางขวา โดยมีระยะห่างขึ้นกับค่าของแอมปลิจูดในลักษณะเป็นสัดส่วนกัน

ในทำนองกลับกัน หากสัญญาณมีแอมปลิจูดเป็นลบ สัญญาณพัลซ์จะปรากฏในช่วงแรกของคาบเวลา ดังนั้นการมอดูเลทแบบพีพีเอ็มจึงสามารถใช้ได้ทั้งสัญญาณที่เป็นอนาลอกและดิจิตอล เพียงแต่ในสัญญาณแบบดิจิตอลเราจะเห็นระยะห่างของพัลซ์ได้แน่นอนกว่า เพราะมีขนาดสัญญาณเพียง 2 ระดับ จึงดูเสมือนว่าคาบเวลาของพัลซ์เป็นตัวกำหนดชนิดของข้อมูล

ในส่วนของวงจรจะมีส่วนประกอบอยู่ 2 ส่วน คือ ภาคส่งสัญญาณ และภาครับสัญญาณ ซึ่งซีพียูจะรับคำสั่งที่ได้รับการแปลงรูปสัญญาณจากภาครับของรีโมตเพื่อนำไปใช้ในการควบคุมระบบ ซึ่งมีดังต่อไปนี้

1. ภาคส่งสัญญาณ



รูปที่ 8.2แสดงบล็อกไดอะแกรมของวงจรในภาคส่งสัญญาณควบคุมที่ใช้แสง

รูปที่ 8.2 แสดงบล็อกไดอะแกรมของวงจรในภาคส่งสัญญาณควบคุมที่ใช้แสง ซึ่งประกอบไปด้วยวงจรเข้ารหัส ทำหน้าที่ จัดรูปแบบของรหัสควบคุมตามที่กำหนด จากนั้นทำการแปลงรูปสัญญาณให้เป็นสัญญาณพีพีเอ็มก่อนส่งไปยังวงจรขับสัญญาณ เพื่อแปลงให้เป็นแสงสำหรับส่งออกไป โดยใช้ไอซี SL490 ซึ่งวงจรจะมีขนาดเล็ก และวงจรไม่ยุ่งยากซับซ้อน

ไอซี SL490 เป็นไอซีที่ ถูกออกแบบมาสำหรับรีโมทคอนโทรลโดยเฉพาะ โดยจะกินกระแสไฟเพียง 6 ไมโครแอมป์ ในขณะที่ไม่มีการส่งสัญญาณลักษณะของสัญญาณพีพีเอ็มที่ได้สร้างใช่ส่งออกไปเลย หรืออาจรวมกับคลื่นพาห์ในแบบของสัญญาณโทนเบิรสต์ก็ได้

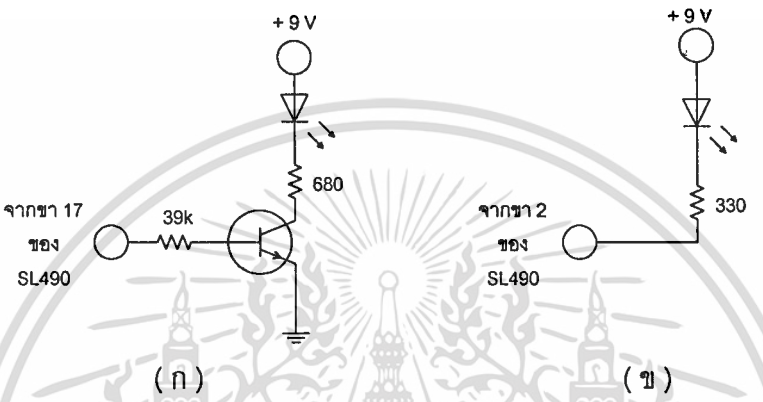
ไอซี SL490 ขณะทำการกดสวิทช์ส่งสัญญาณ ชุดข้อมูลจะถูกสร้างขึ้นตามรหัสไบนารี ABCD แล้วส่งออก หากส่งหมดไปแล้วแต่สวิทช์ยังคงถูกกดค้างอยู่ข้อมูลชุดเดิมจะถูกสร้างขึ้นมาใหม่แล้วส่งออกไปเรื่อยๆ โดยมีช่วงเวลาที่ซิงโครนัส (S) เป็นตัวแยกชุดข้อมูลเอาไว้ และหากมีการปล่อยสวิทช์ ข้อมูลในบิทที่เหลือ ก็ยังคงถูกส่งออกไปจนหมดแล้วจึงหยุด สัญญาณเอาท์พุทที่เป็นสัญญาณ พีพีเอ็มจะถูกส่งออกมาที่ขา 2

ส่วนขาที่ 3 เป็นเอาท์พุทของไอซี ที่มีสถานภาพทางเฟสตรงกันข้ามกับสัญญาณพีพีเอ็มที่ได้จากขาสอง สัญญาณเอาท์พุทอีกอันหนึ่งเป็นสัญญาณที่ได้จากขา 17 โดยปกติขณะไม่มีการส่งสัญญาณ เอาท์พุท์ที่ขา 17 จะอยู่ในสภาวะต่ำ และเมื่อมีการกดสวิทช์ส่งสัญญาณ เอาท์พุทนี้จะ

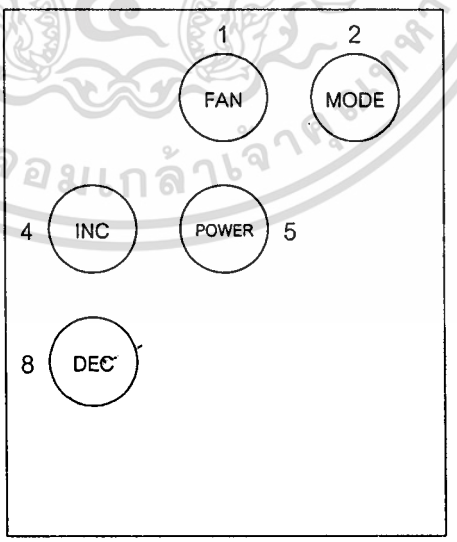
อยู่ในสภาวะสูง ทั้งสัญญาณที่ได้จากขา 2 และขา 17 สามารถนำมาใช้เป็นตัวบอกสภาวะชั่วขณะที่ทำการกดสวิทช์ส่งข้อมูลได้ โดยการต่อวงจรเพิ่มเติมดังรูปที่ 8.3

รูปที่ 8.4 แสดงรูปเครื่องส่งรีโมทที่ใช้ในการส่งคำสั่งควบคุมเครื่องปรับอากาศ

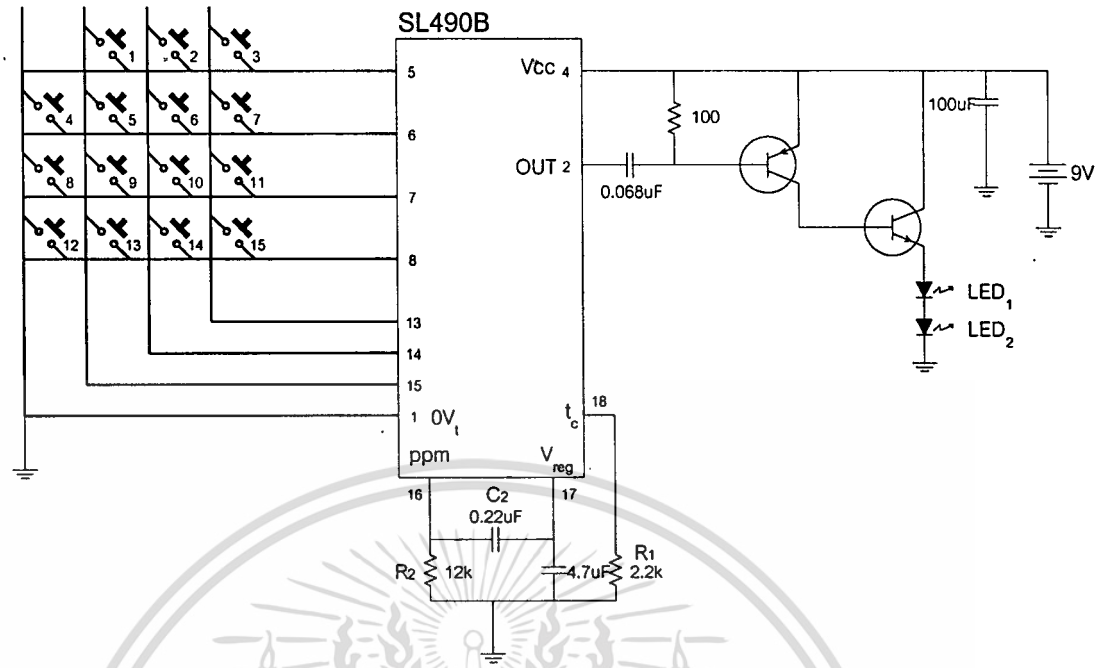
รูปที่ 8.5 แสดงวงจรของภาคส่งรีโมทคอนโทรลโดยที่แต่ละปุ่มจะกำหนดฟังก์ชันไว้ให้ตรงกับปุ่มที่ใช้ในส่วนของแผงคีย์บอร์ดที่กำหนดไว้เพื่อไม่ให้เป็นการสับสนเมื่อนำไปใช้งานจริง



รูปที่ 8.3 การต่อ LED เพื่อแสดงสภาวะการส่งสัญญาณควบคุม
(ก) ผ่านการขับจากทรานซิสเตอร์
(ข) ใช้ SL490 ขับโดยตรง



รูปที่ 8.4 เครื่องส่งสัญญาณระยะไกล



รูปที่ 8.5 วงจรภาคส่งของรีโมทคอนโทรล

การส่งสัญญาณควบคุมแบบโทนเบิรสต์ ใช้วงจรในรูปที่ 8.5 โดยการกำหนดค่าของ R_1 C_1 จากสมการ

$$f = 1 / C_1R_1$$

เมื่อ f เป็นความถี่ในหน่วยของเฮิรตซ์ C_1 R_1 มีหน่วยเป็นฟารัดและโอห์มตามลำดับ ในกรณีนี้ต้องการส่งสัญญาณควบคุมโดดๆ ไม่มีคลื่นพาห์ร่วมไปด้วยจึงใช้ตัวต้านทานขนาด 2.2 กิโลโอห์ม ต่อแทน R_1 แล้วถอด C_1 ออกดังปรากฏในรูปที่ 8.5

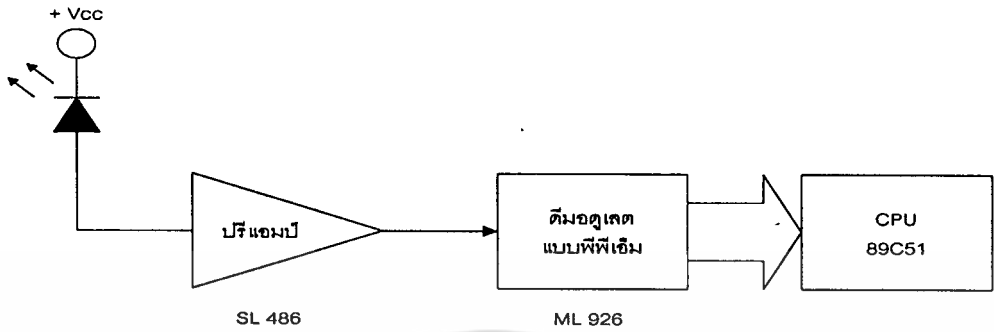
คาบเวลาที่ใช้สำหรับส่งข้อมูลที่เป็น "0" หรือ t_0 กำหนดด้วยค่าของ C_2 และ R_2 จากสมการ

$$t_0 = 1.4C_2R_2$$

เมื่อ t_0 มีหน่วยเป็นวินาที C_2 และ R_2 มีหน่วยเป็นฟารัดและโอหม์ตามลำดับ

สำหรับคาบเวลาของข้อมูลที่เป็น "1" จะถูกกำหนดอย่างอัตโนมัติ ให้มีค่าประมาณ 2 ใน 3 ของเวลา t_0 และในกรณีที่ชุดข้อมูลถูกส่งออกมาติดๆ กัน (ซึ่งอาจเกิดจากการกดสวิทช์ค้างนานเกินไป) จะมีการสร้างช่วงเวลาซิงโครนัส (S) กั้นไว้ระหว่างชุดรหัสข้อมูลโดยคาบเวลาซิงโครนัสนี้จะมีค่าเป็น 2 เท่า ของ t_0

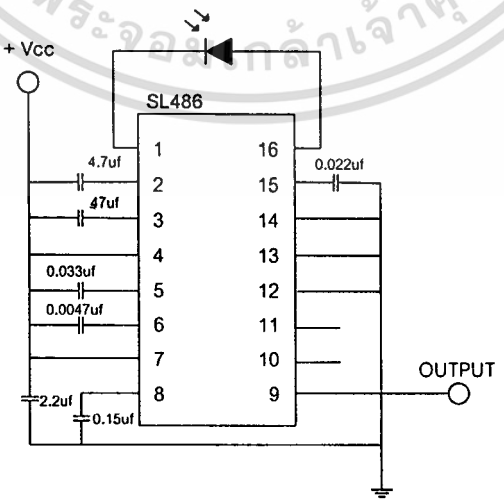
2. ภาครับสัญญาณ



รูปที่ 8.6 บล็อกไดอะแกรมภาครับ

รูปที่ 8.6 เป็นบล็อกไดอะแกรมของภาครับสัญญาณ เริ่มจากโฟโตไดโอด D1 ทำหน้าที่เปลี่ยนสัญญาณแสงเป็นสัญญาณไฟฟ้า แล้วทำการขยายสัญญาณด้วยวงจรปรีแอมป์ ก่อนถูกมอดูเลตออกเป็นรหัสทางไบนารี จากนั้นทำการถอดรหัสสำหรับส่งไปควบคุมระบบต่อไป

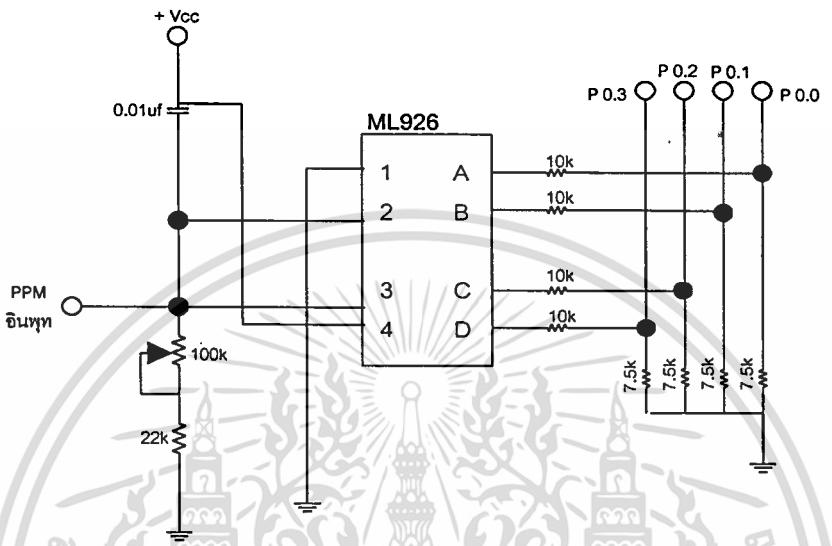
ส่วนของวงจรปรีแอมป์ใช้ไอซี SL486 ซึ่งถูกออกแบบมาใช้กับสัญญาณอินฟาเรด ที่เป็นการส่งข้อมูลแบบดิจิตอล โดยเฉพาะสัญญาณไฟฟ้าจากโฟโตไดโอด ที่มีค่าในช่วงของนาโนแอมป์และไมโครแอมป์ จะถูกขยายและส่งออกทางขา 9 ของไอซี SL486 ด้วยค่าของกระแสที่อาจมากถึง 5 มิลลิแอมป์ ข้อดีของการใช้ไอซี SL486 คือ สามารถตัดปัญหาสัญญาณรบกวนจากภายนอกที่ไม่เกี่ยวข้องได้เป็นอย่างดี วงจรใช้งานของไอซี SL486 แสดงดังรูปที่ 8.7



รูปที่ 8.7 ภาครับสัญญาณอินฟราเรด

วงจรดีมอดูเลต

การดีมอดูเลตสัญญาณพีพีเอ็ม ที่ได้จากขาเอาท์พุทของไอซี SL486 จะใช้ไอซี ML926 ซึ่งไอซีจะดีมอดูเลตสัญญาณออกมาได้เพียง 16 ช่องสัญญาณ



รูปที่ 8.8 ส่วนดีมอดูเลต

การทำงานของไอซี ML926 จากวงจรรูปที่ 8.8 ถูกกำหนดด้วยสัญญาณนาฬิกาภายในที่มีความถี่ที่หาได้จากสมการต่อไปนี้

$$f = 1 / (0.15 * C1 * (VR_1 + R_1))$$

เมื่อค่าของตัวต้านทานและตัวเก็บประจุ มีหน่วยเป็นโอห์มและฟารัด ตามลำดับ และค่าความต้านทานของ VR_1 รวมกับ R_1 ควรมีค่าอยู่ระหว่าง 2-200 กิโลโอห์ม

ค่าความถี่ f ของวงจรรับนี้ ต้องมีความสัมพันธ์กับความถี่ในวงจรส่ง โดยกำหนดจาก

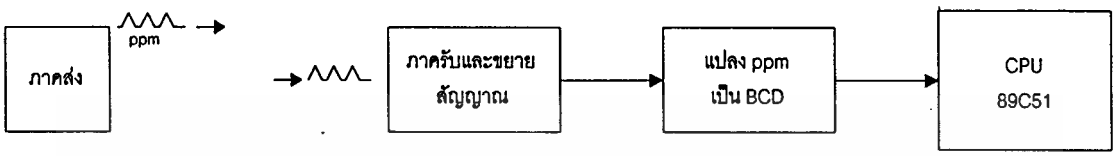
$$f = 40 / t_o$$

เมื่อ t_o เป็นคาบเวลาของข้อมูลบิทที่เป็น "0" ในสัญญาณพีพีเอ็ม

จากวงจรสำหรับรหัสควบคุม 4 บิทสัญญาณไบนารีที่ถูกดีมอดูเลตแล้วจะถูกส่งออกมาที่ขา 5, 6, 7, 8, เป็นบิท A, B, C, D ตามลำดับ

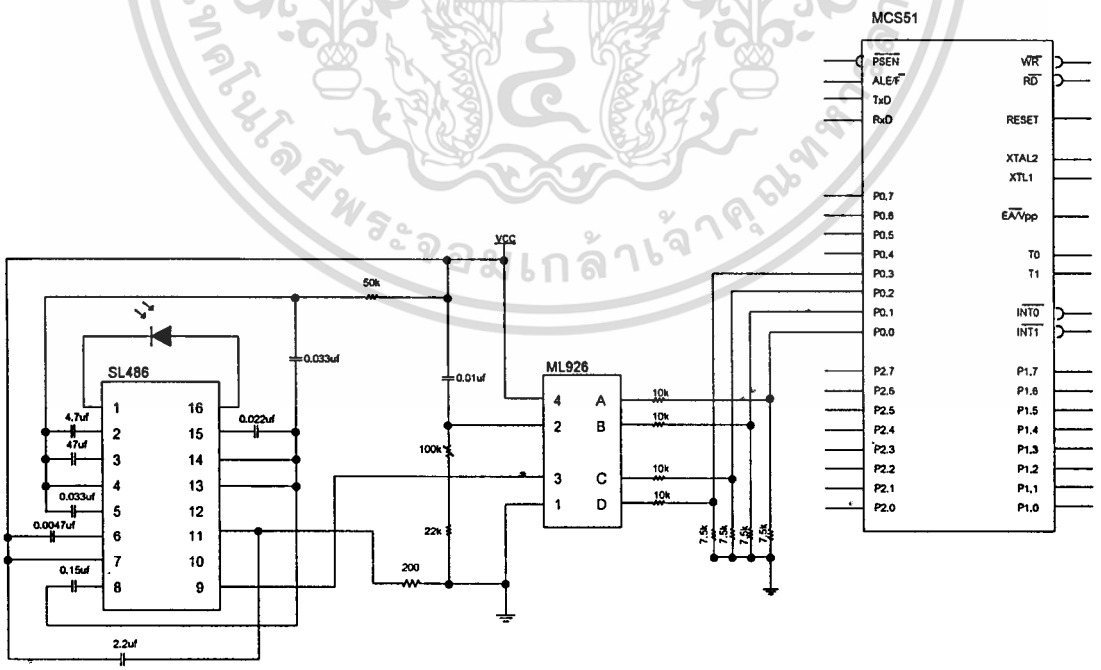
การส่งสัญญาณจากภาครับรีโมทเข้าซีพียู 89C51

เมื่อนำรีโมทมาช่วยในการควบคุมการทำงานของเครื่องปรับอากาศจากระยะไกลจะช่วยอำนวยความสะดวกแก่ผู้ใช้ โดยมีหลักการทำงานเบื้องต้นของรีโมทดังรูปที่ 8.9



รูปที่ 8.9 แสดงหลักการทำงานเบื้องต้นของรีโมท

เมื่อผู้ใช้กดคำสั่งที่รีโมท SL490จะสร้างสัญญาณเพื่อส่งมาที่ภาครับมาเข้าไอซี SL486 เพื่อสร้างสัญญาณไบนารีส่งให้ ML926 ดีมอดูเลตและจะถูกส่งออกทางขา 5, 6, 7, 8 แล้วจะส่งไปยัง 89C51 เข้าทางพอร์ท 0 เพื่อนำมาใช้กำหนดคำสั่งในการควบคุมระบบของเครื่องปรับอากาศ ดังรูปที่ 8.10 แสดงวงจรรับสัญญาณจากเครื่องควบคุมระยะไกลที่ต่อเข้าทางพอร์ท 0 ของ 89C51



รูปที่ 8.10 วงจรรับสัญญาณจากเครื่องควบคุมระยะไกล

บทที่ 9
โปรแกรมการทำงานของระบบ

ฟังก์ชันการทำงานทั้งหมด ของการควบคุมการทำงาน ของเครื่องปรับอากาศ มีดังต่อไปนี้

1.การป้องกันความเสียหาย ที่จะเกิดต่อคอมเพรสเซอร์ ในสภาวะที่ไฟตกหรือไฟเกิน

2.การรับคำสั่ง, เลือก mode การทำงาน, ตั้งเซตพอย์ต (set point) ที่ต้องการ

3.การควบคุมการปิด-เปิด คอมเพรสเซอร์เพื่อให้ได้อุณหภูมิตามที่ตั้งไว้

4.การเลือกระดับความเร็วของพัดลม เพื่อการระบายความเย็นที่เหมาะสม โดยให้มีทั้งการเลือกโดยผู้ใช้ หรือเลือกโดยอัตโนมัติ

5.การตั้งเวลาปิด-เปิด การทำงานของระบบโดยอัตโนมัติ

6.การเพิ่มค่าเซตพอย์ต ขึ้นเองโดยอัตโนมัติ ในขณะนอนหลับ (sleep mode)

รูปที่ 9.1เป็น โฟลชาร์ตแสดงลูป (Loop) การทำงานของระบบโดยรวม ซึ่งมีการสแกน รับคำสั่งทางคีย์บอร์ด (keyboard) สแกนรับคำสั่งจากรีโมท (remote control) และสแกนส่งค่าต่างๆออกทางส่วนแสดงผล (display) และเข้าสู่การทำงานในฟังชันต่างๆ

การทำงานแต่ละฟังก์ชัน อธิบายได้ดังนี้

<u>การป้องกันความเสียหาย อันเนื่องมาจากไฟตก ไฟเกิน</u>

โปรแกรมจะรับเอาค่าที่แสดงสภาวะของระดับแรงดันไฟฟ้าของไฟบ้านมาจากวงจรเปรียบเทียบ (Comparator Circuit) โดยผ่านทางขา p1.0 และ p1.1 และทำการตรวจสอบว่า อยู่ในระดับปกติหรือเปล่า โดยถ้าปกติ คือ p1.0 และ p1.1 เป็นสถานะสูง ทั้งคู่ก็จะผ่านการตรวจสอบออกไป แต่ถ้าไม่ปกติ ก็จะส่งสัญญาณเตือน และทำการปิดคอมเพรสเซอร์ โฟลชาร์ตแสดงการทำงานแสดงในรูปที่ 9.2

<u>การควบคุมการปิด-เปิดคอมเพรสเซอร์</u>

โปรแกรมจะรับเอาค่าอุณหภูมิที่วัดได้จริง มาเปรียบเทียบกับค่าเซ็ทพอย์ตและถ้าอุณหภูมิสูงเกินค่าเซ็ทพอย์ตก็จะทำการเปิดคอมเพรสเซอร์แต่การที่จะเปิดได้นั้นต้องตรวจสอบดูก่อน ว่าคอมเพรสเซอร์ได้ผ่านการปิดครั้งสุดท้ายมาแล้ว อย่างน้อย 3 นาที หรือยัง ทั้งนี้เพื่อให้ความดันของของเหลวภายในท่อแอร์ปรับตัวอยู่ในสภาวะสมดุลย์ และเพื่อเป็นการป้องกันความเสียหาย ที่จะเกิดต่อคอมเพรสเซอร์ในการที่จะต้องเปิดปิด บ่อยๆด้วย

ถ้าอุณหภูมิต่ำกว่าเซ็ทพอย์ตก็ให้ปิดคอมเพรสเซอร์ และเริ่มทำการนับเวลา 3 นาที เพื่อการตรวจสอบ เมื่อจะเปิดคอมเพรสเซอร์ในครั้งต่อไป

รูปที่ 9.3 เป็น โฟลชาร์ต แสดงการทำงาน

การเลือกระดับความเร็วของพัดลม

การเลือกมีทั้งแบบเลือกโดยตรงจากผู้ใช้ และเลือกเองโดยอัตโนมัติ ซึ่งถ้าเป็นการเลือกแบบอัตโนมัติ (auto mode) จะมีวิธีกาสรเลือกดังนี้คือ นำค่าอุณหภูมิมาเปรียบเทียบกับเซ็ทพอย์ต ถ้าอุณหภูมิสูงกว่าเซ็ทพอย์ตตั้งแต่ 2 องศาขึ้นไป ก็เลือกความเร็วระดับสูง เพื่อเร่งระบายความเย็นให้กระจายไปทั่วห้องอย่างรวดเร็ว แต่ถ้าอุณหภูมิต่ำกว่าเซ็ทพอย์ต ก็ให้เลือกความเร็วระดับต่ำ และถ้าอุณหภูมิอยู่ระหว่างสองเงื่อนไขข้างต้น ก็ให้เลือกความเร็วระดับกลาง

โฟลชาร์ต แสดงการทำงานของโปรแกรมดังรูปที่ 9.4

การตั้งเวลาเปิดปิดเองโดยอัตโนมัติ

ผู้ใช้สามารถตั้งเวลาปิดเองได้โดยอัตโนมัติ โดยเข้าสู่ mode timer 0 แล้วใส่ค่าเวลาเป็นชั่วโมงที่ต้องเริ่มปิดการทำงาน และผู้ใช้สามารถตั้งเวลาให้เปิดต่อจากนั้นได้อีกโดยวิธีเดียวกัน คือ เข้าสู่ mode timer 1 แล้วใส่ค่าชั่วโมง ที่ต้องการเริ่มต้นการทำงาน ใหม่อีกครั้งหนึ่ง เช่น ต้องการปิดการทำงานในอีก 2 ชั่วโมง ข้างหน้า แล้วอีก 4 ชั่วโมงหลังจากนั้นให้เริ่มทำงานใหม่อีกครั้งหนึ่ง ก็ทำได้โดยการเข้าสู่ mode timer 0 แล้วใส่ค่า 2 และเข้าสู่ mode timer 1 แล้วใส่ค่า 4 เป็นต้น

โฟลชาร์ต แสดงการทำงานใน mode การตั้งเวลาเปิดปิดอัตโนมัติแสดงดังรูปที่ 9.5

sleep mode

ในขณะนอนหลับร่างกายมีการใช้พลังงานน้อย ดังนั้นอุณหภูมิที่เย็นสบายในตอนตื่น อาจทำให้รู้สึกเย็นเกินไปในขณะนอนหลับได้ ดังนั้นใน sleep mode นี้ จะทำการเพิ่มค่าเซ็ทพอย์ต ขึ้นเองโดยอัตโนมัติในทุกๆ 1 ชั่วโมง แต่จะเพิ่มให้เพียง 2 ครั้งเท่านั้น แล้วก็ออกจาก sleep mode

ผู้ใช้สามารถเลือกให้ทำงานใน sleep mode ได้โดยการเข้าสู่ mode "sleep" แล้วกด key "UP" และการยกเลิก sleep mode ก็เข้าสู่ mode "sleep" แล้วตามด้วย key "DOWN"

การทำงานของโปรแกรมเริ่มต้นด้วยการตั้งค่า และสั่งเริ่มนับเวลา 1 ชั่วโมง ทุกรอบของการเข้าสู่โหมดนี้ก็จะมีการตรวจสอบว่าครบเวลา 1 ชั่วโมงหรือยัง ถ้ายังก็จะออกไปก่อนเพื่อกลับมาตรวจสอบอีกในรอบต่อไป แต่ถ้าครบ 1 ชั่วโมงแล้วก็จะทำการลดเซ็ทพอย์ต ลง 1 องศา แล้วจะตรวจสอบว่า ลดค่ามา 2 ครั้งหรือยัง ถ้าครบแล้ว ก็เป็นอันสิ้นสุดการทำงานใน sleep mode และถ้ายังไม่ครบ ก็จะเริ่มตั้งค่านับอีกครั้งหนึ่ง ลำดับการทำงานแสดงดังโฟลชาร์ต รูปที่ 9.6

การรับค่าอุณหภูมิ

การทำงานของโปรแกรมที่สำคัญอีกส่วนหนึ่งก็คือ การรับอุณหภูมิเข้ามา เพื่อทำการประมวลผล ซึ่งมีหลักการทำงานดังนี้

เราจะใช้ timer 1 ของ ซีพียู เป็นตัวนับเวลาเพื่อให้กำเนิดสัญญาณพัลซ์ (pulse) ที่มีคงที่ออกมา โดยในที่นี้ เรากำหนดให้ความกว้างของ pulse ที่สร้างขึ้นดังกล่าว มีค่าเป็น 1.2 มิลลิวินาที

สัญญาณพัลซ์ แต่ละลูกจะถูกส่งเป็นทริกเกอร์ ให้กับวงจรเซนเซอร์ เพื่อรับเอาสัญญาณพัลซ์ ซึ่งมีความกว้าง เปลี่ยนไปตามอุณหภูมิ เข้ามา และเราสามารถนับความกว้างของสัญญาณพัลซ์นี้โดยการรับสัญญาณพัลซ์ เข้าทางขา external interrupt 0 และตั้งให้ timer 0 ทำงานเป็น timer ตลอดช่วงที่สัญญาณพัลซ์ เป็นสภาวะสูงแล้วจึงนำค่านี้ไปทำให้เป็นค่าของอุณหภูมิ โดยใช้ ตารางที่สร้างมาจากการทดลอง

เราใช้เทคนิคของการอินเทอร์รับ (interrupt) ในการเขียนโปรแกรมโดยตั้งค่า timer1 เป็น timer mode 1 สำหรับนับเวลา 1.2 มิลลิวินาที และเมื่อนับครบ ก็จะกระโดดไปยัง routine ซึ่งมีลำดับการทำงานดังโฟลชาร์ต ในรูปที่ 9.7 ซึ่งจะทำการส่งทริกเกอร์และสั่งให้ timer0 รอนับความกว้างของพัลซ์ และเรายังใช้พัลซ์ที่มีความถี่คงที่ใช้นับเวลาต่างๆในโปรแกรมอีกด้วย

เมื่อสัญญาณพัลซ์ที่เข้ามาทางขา interrupt0 เปลี่ยนสถานะจากสภาวะสูงเป็นสภาวะต่ำ ก็จะกระโดดเข้าสู่ routine ที่แสดงโดยโฟลชาร์ต ในรูปที่ 9.8 ซึ่งจะทำการหยุดนับของ timer 0 แล้วนำค่าที่ได้ไปค้นหา และเปลี่ยนเป็นค่า อุณหภูมิ และเก็บค่าไว้ประมวลผลต่อไป

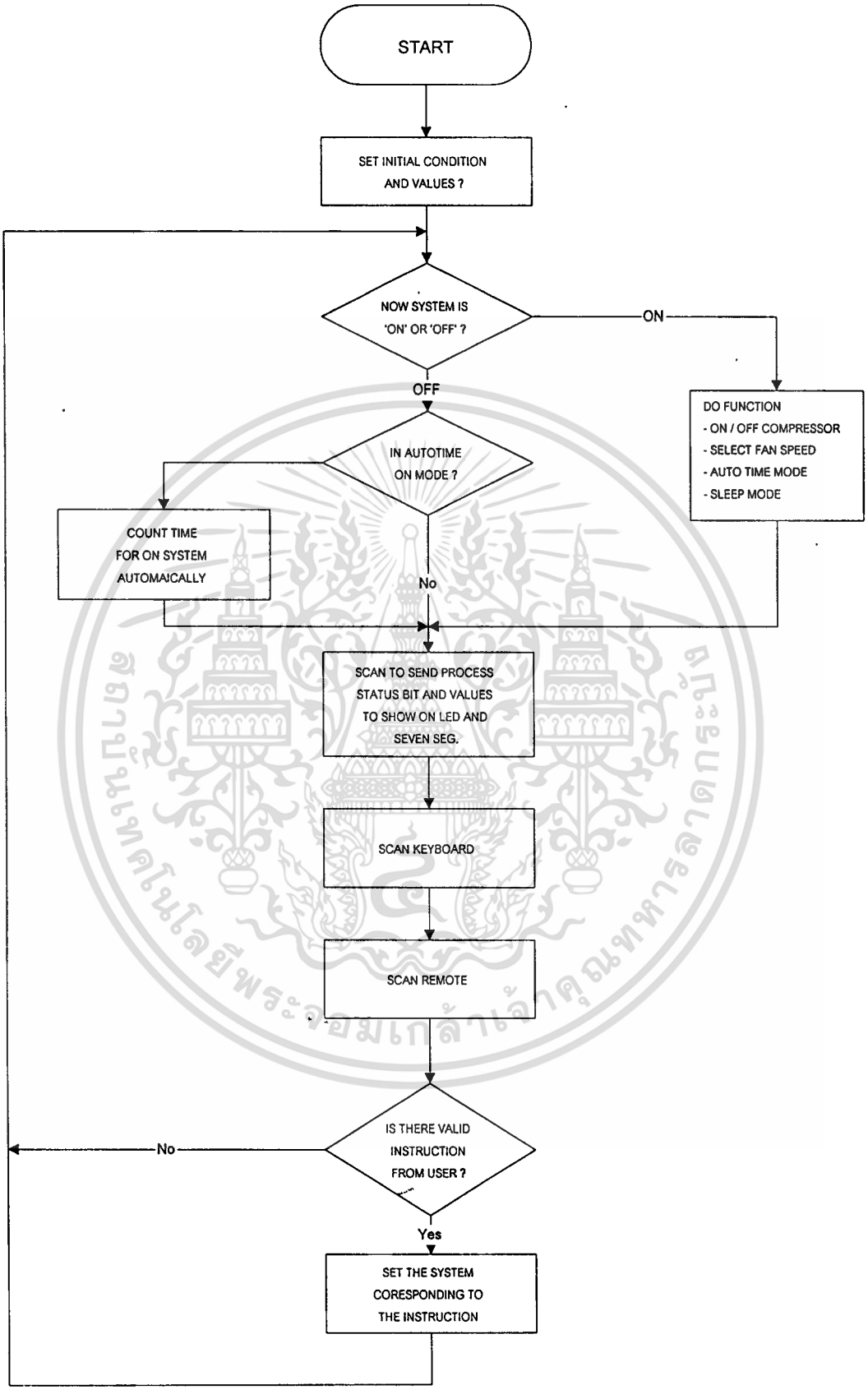
การสร้างตารางเพื่อใช้สำหรับการเปลี่ยนค่าความกว้างของพัลซ์เป็นค่าอุณหภูมิ มีวิธีการดังนี้

- เริ่มจาก ทำการทดลองโดยวัดค่าอุณหภูมิจริง กับค่าที่ timer0 นับได้ และบันทึกค่าเอาไว้ ผลการทดลองที่ได้แสดงดังตารางที่ 9.1

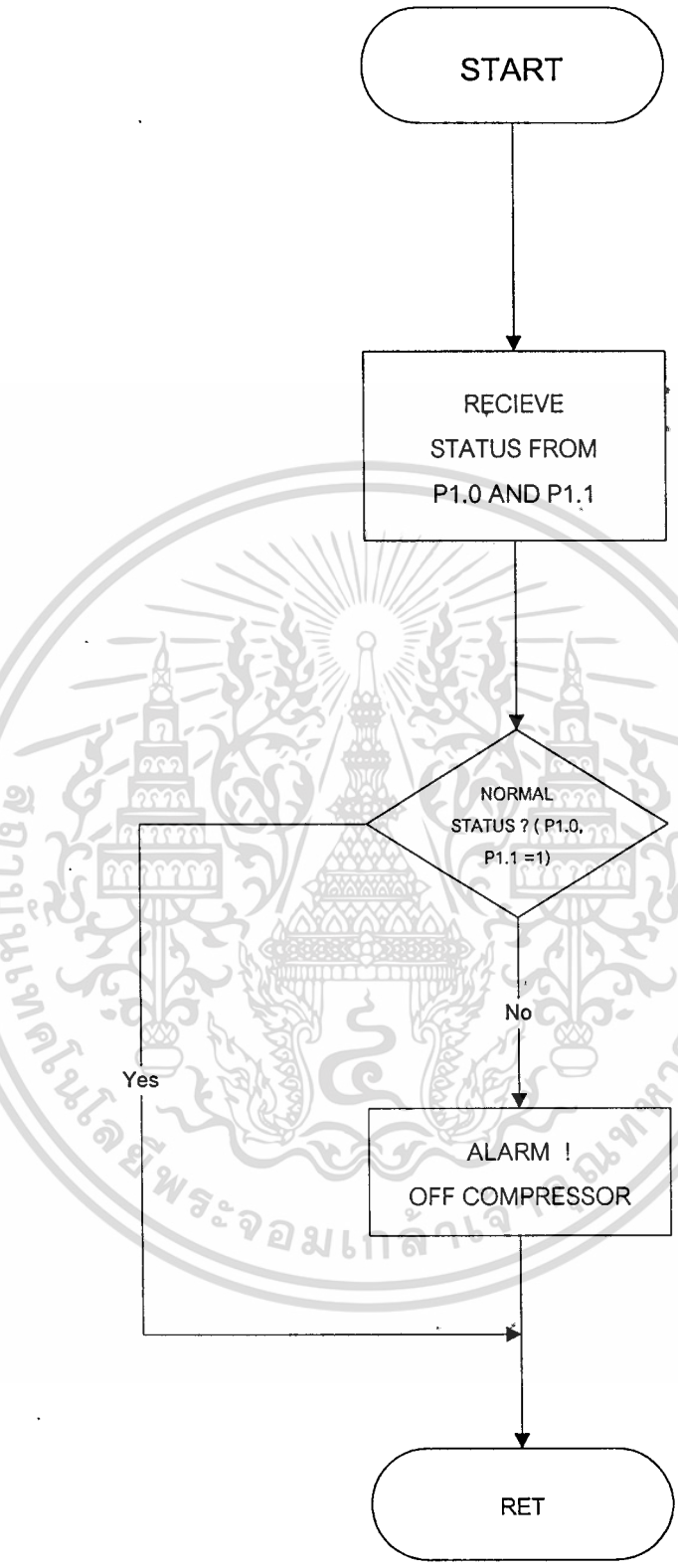
-นำค่าที่ได้มาทำการหาค่าเฉลี่ย และทำการประมาณค่าโดยสมมุติว่า ความกว้างของพัลซ์ แปรผันเป็นเชิงเส้นกับอุณหภูมิ ทั้งนี้เนื่องจากอุณหภูมิสำหรับการทำงานอยู่ในช่วงไม่กว้างคือ 21-28 องศา ประกอบกับโหลด (load) ที่แท้จริงของระบบก็คือผู้ใช้ ซึ่งจะมีประสาทสัมผัสที่ไม่จำเป็นต้องใช้ค่าที่เที่ยงตรงมากนักก็ได้ เราจึงใช้การประมาณค่าดังกล่าวก็ได้

-จัดช่วงของค่าความกว้างของพัลซ์สำหรับประมาณเป็นค่าอุณหภูมิที่เป็นจำนวนเต็มหนึ่งๆดังแสดงในตาราง เอ ของตารางที่ 9.2 ซึ่งจะทำการสร้างตารางดังกล่าว ต้องกำหนดให้มี ช่วงของ hysteresis ด้วยทั้งนี้เพื่อไม่ให้ค่าของอุณหภูมิมีการออสซิลเลท ซึ่งจะทำให้เกิดผลเสีย ต่อพัดลมและคอมเพรสเซอร์ ได้ ค่าอุณหภูมิโดยประมาณ ที่สอดคล้องกันแสดงในตาราง บี

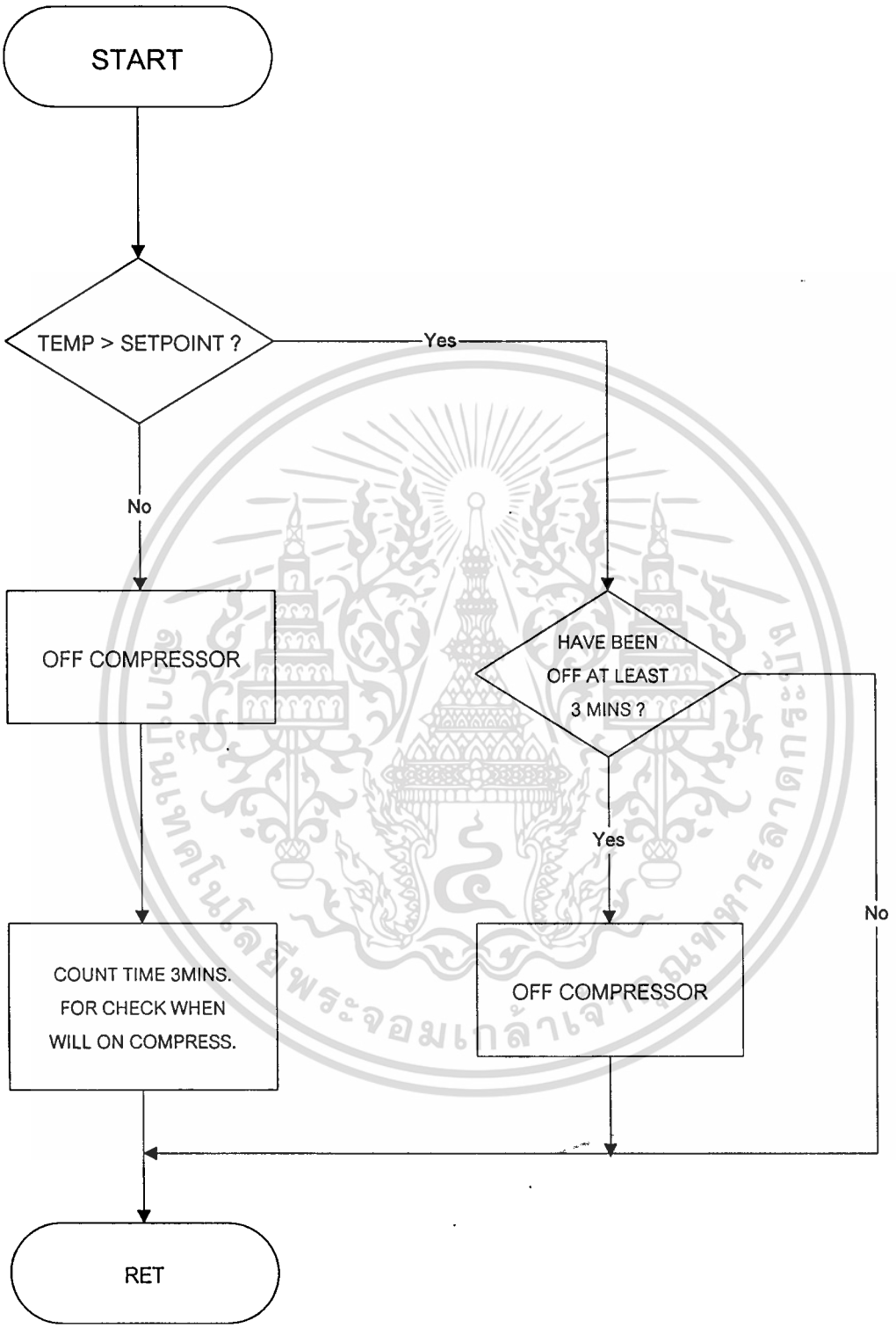
ในการค้นหา ก็จะนำค่าที่นับได้มาเปรียบเทียบเพื่อหาช่วงจาก ตาราง เอ ก่อน แล้วจึงนำไปชี้เป็นค่าของอุณหภูมิจาก ตาราง บี



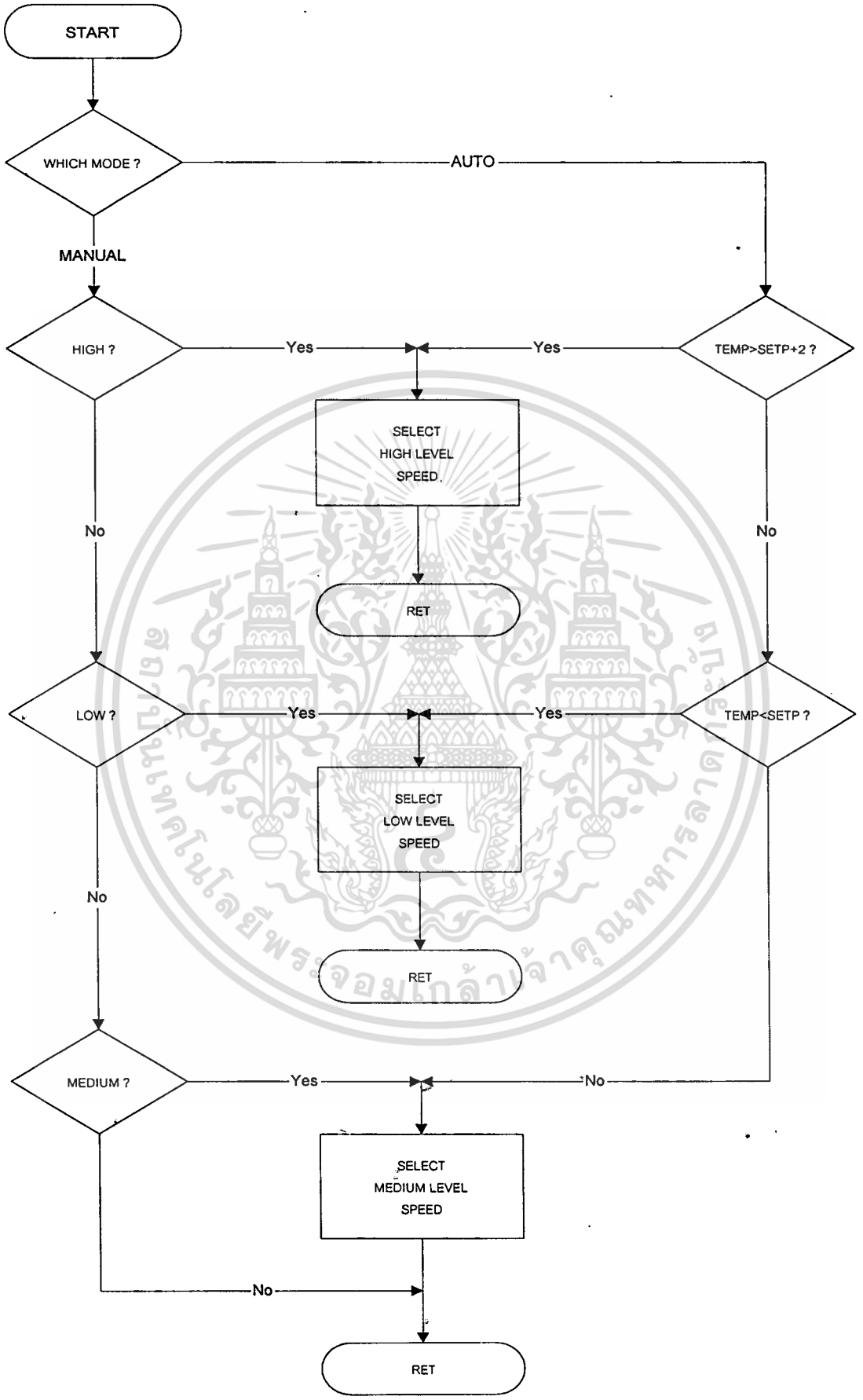
รูปที่ 9.1 โฟลชาร์ตแสดงการทำงานของระบบโดยรวม



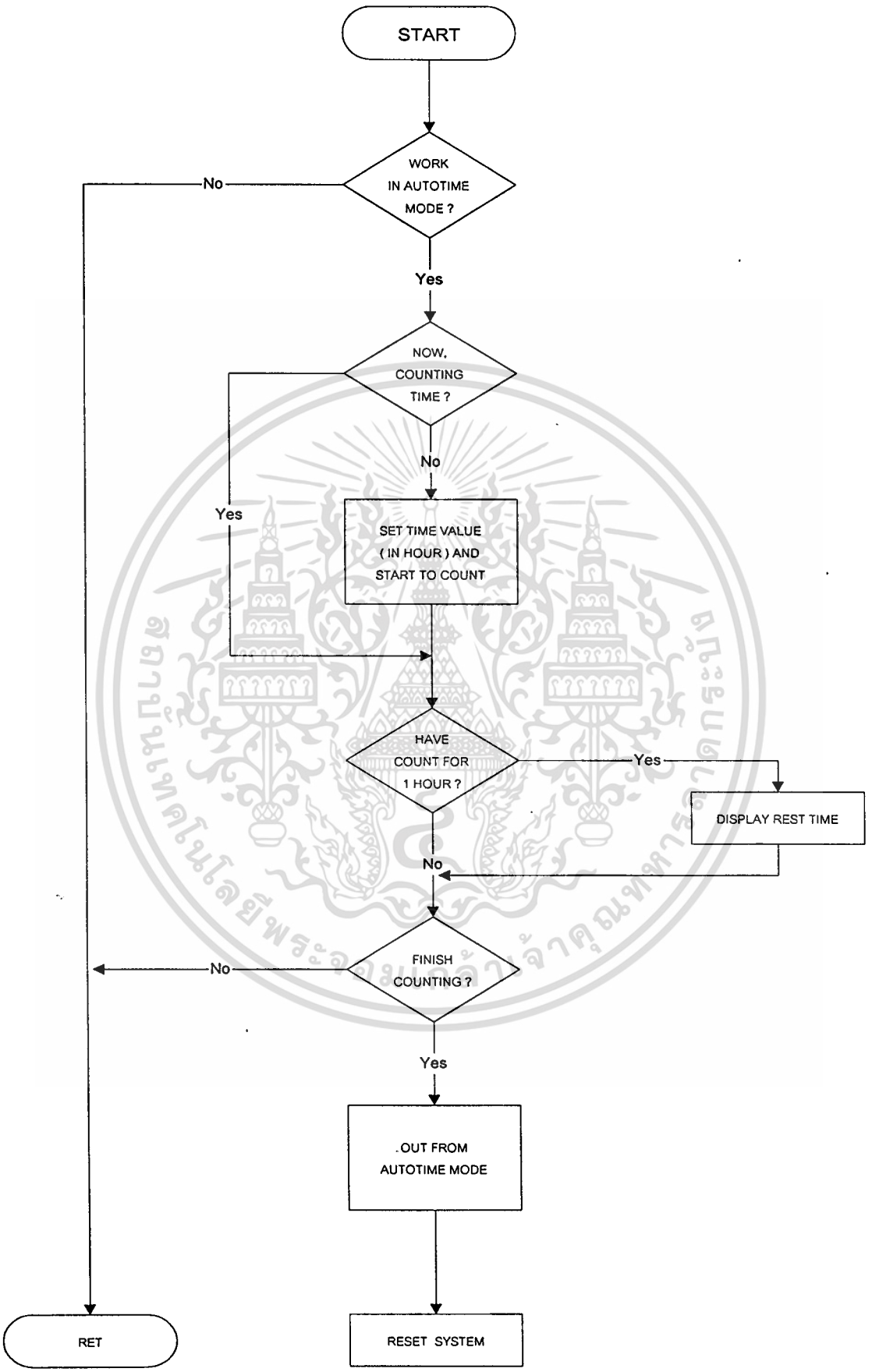
รูปที่ 9.2 โฟลชาร์ตส่วนการป้องกันความเสียหายจากสภาวะไฟตกหรือไฟเกิน



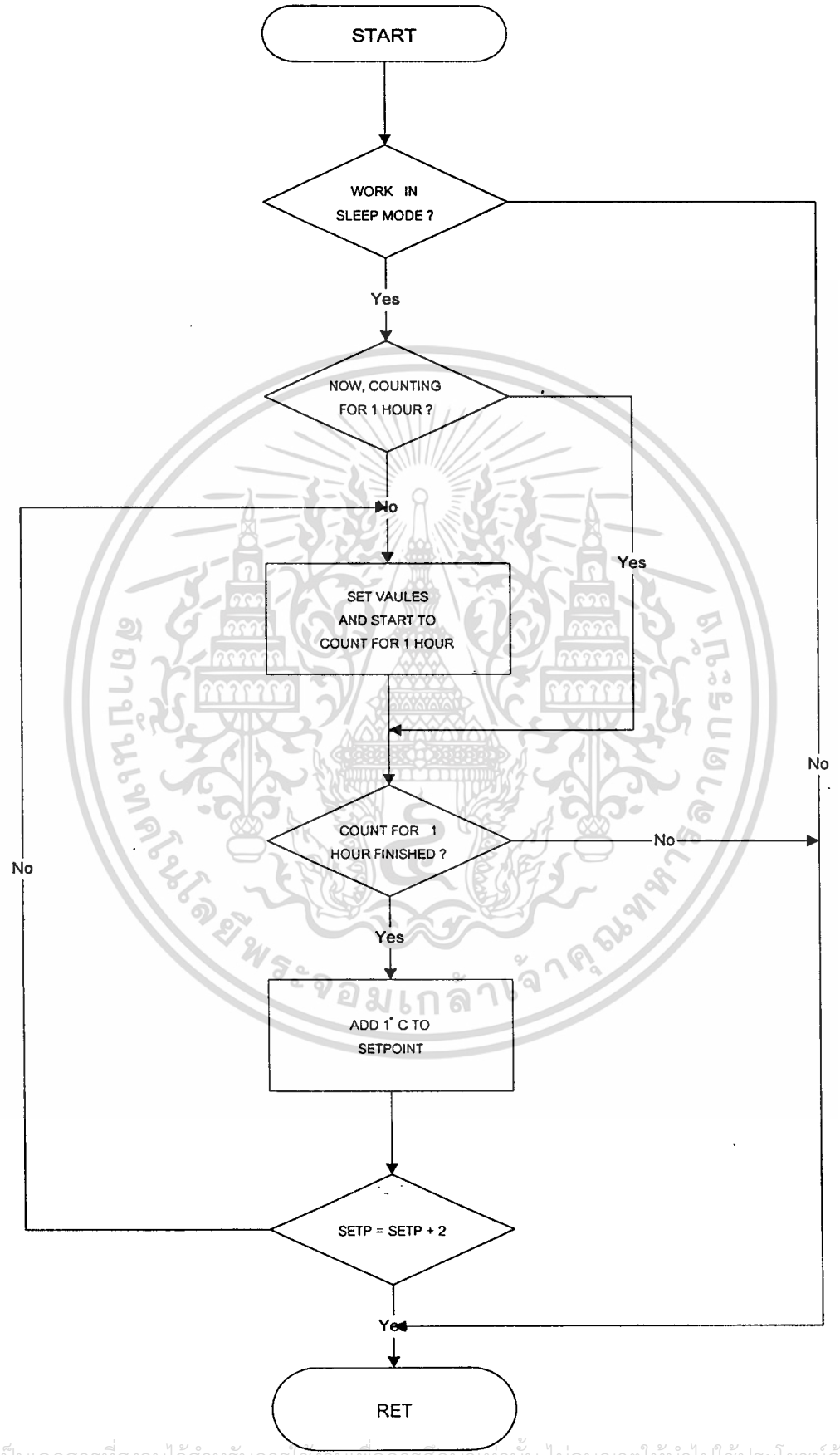
รูปที่ 9.3 โฟลชาร์ตการควบคุมการปิด - เปิด compressor



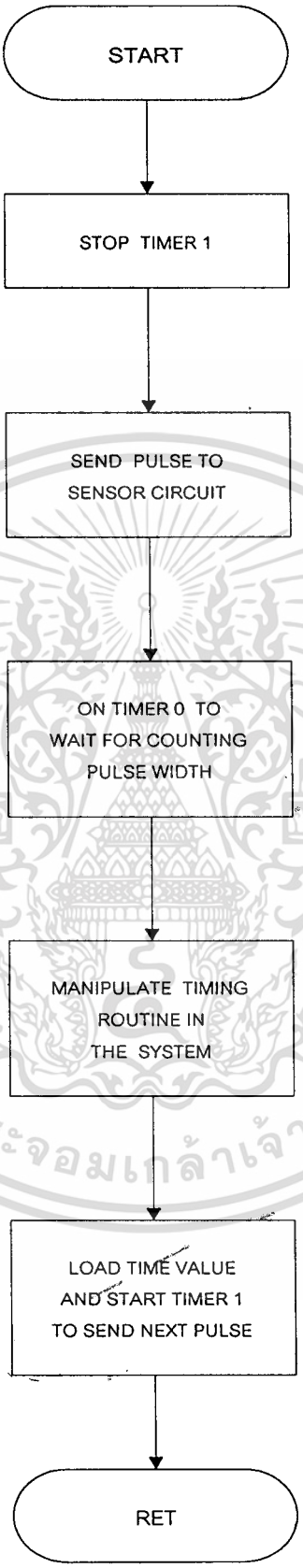
รูป 9.4 โฟลชาร์ตการเลือกระดับความเร็วของพัดลม



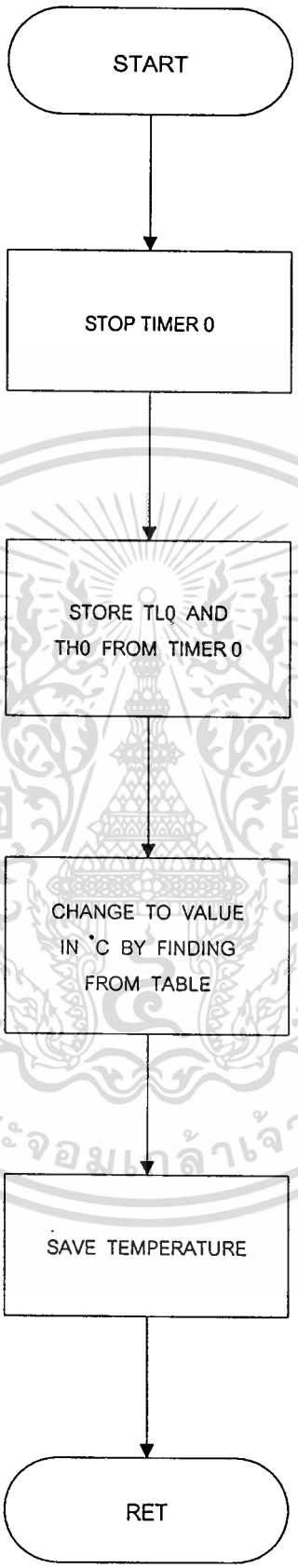
รูปที่ 9.5 โฟลชาร์ตการตั้งเวลาเปิด - ปิดอัตโนมัติ



รูปที่ 9.6 โฟลชาร์ตโหมด sleep



รูปที่ 9.7 โฟลชาร์ตการรับค่าอุณหภูมิส่วนสร้างสัญญาณ pulse



รูปที่ 9.8 โฟลชาร์ตการรับค่าอุณหภูมิส่วนการนับความกว้างของสัญญาณภายนอก

อุณหภูมิ (°C)	ค่าความกว้างของพัลซ์ (uS)		ค่าเฉลี่ย (uS) (average)	ค่าเชิงเส้น (uS) (linearize)
	ครั้งที่ 1	ครั้งที่ 2		
21	1019	1019	1019	1019
22	960	976	968	965
23	912	912	912	911
24	848	864	856	857
25	800	800	800	803
26	768	752	760	749
27	688	704	696	695
28	624	640	632	641
29	576	592	584	587
30	544	544	544	533

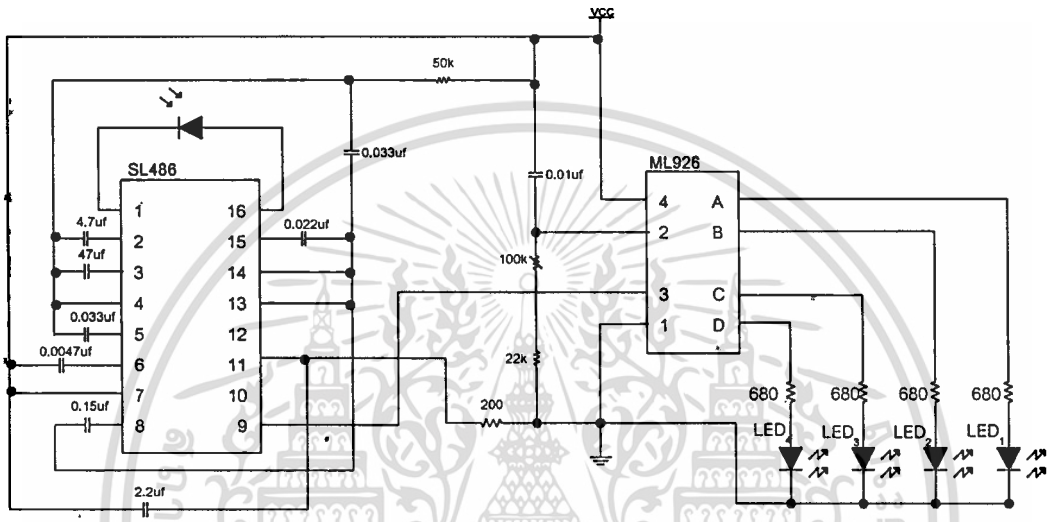
ตารางที่ 9.1 ตารางผลการทดลองวัดค่าอุณหภูมิจริงที่แทนด้วยความกว้างของพัลซ์จากวงจรรูปที่ 5.1

ตาราง A	ตาราง B
ช่วง (range)	อุณหภูมิ (° C)
552 ลงมา	30
551 - 570	29
605 - 624	28
658 - 698	27
713 - 732	26
767 - 786	25
821 - 840	24
875 - 894	23
929 - 948	22
983 - 1003	21
1037 ขึ้นไป	20

ตารางที่ 9.2 การประมาณค่าอุณหภูมิ

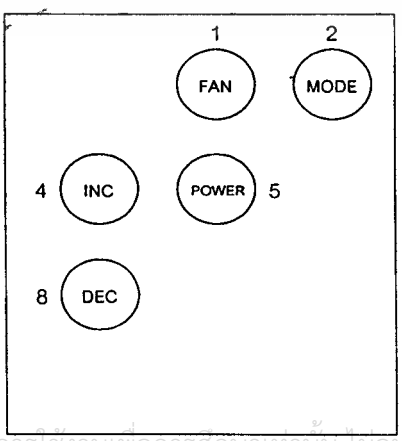
บทที่ 10
ผลการทดลอง

เมื่อนำชุดรีโมทมาต่อใช้ในการควบคุมการทำงานของเครื่องปรับอากาศ ได้ทำการออกแบบวงจรเพื่อทดสอบการรับ-ส่ง สัญญาณของชุดรีโมทว่าตรงตามที่ออกแบบไว้หรือไม่ดังแสดงใน รูปที่ 10.1



รูปที่ 10.1 แสดงการต่อ LED เพื่อทดสอบการรับ-ส่งสัญญาณรีโมท

ทำการทดลองโดยการนำ LED มาต่อเพื่อตรวจสอบเอาท์พุทที่ออกจากภาครับของรีโมทว่าตรงตามที่ออกแบบไว้ตามที่กำหนดในโปรแกรมควบคุมหรือไม่ โดยมี LED อยู่ทั้งหมด 4 ชุดเพื่อทำการอ่านเอาท์พุทในลักษณะของเลขฐาน 2 จำนวน 4 หลักตามที่ออกแบบเอาไว้ตั้งแต่ต้น โดยถ้า LED สว่างให้หลักนั้นมีค่าเป็น "1" และไม่สว่างให้หลักนั้นมีค่าเป็น "0"



รูปที่ 10.2 แสดงรีโมทที่ออกแบบไว้

จากรูปที่ 10.2 จะเห็นว่าได้กำหนดฟังก์ชันการทำงานของรีโมทไว้ 5 ปุ่มด้วยกันคือ FAN, MODE, POWER, INC, DEC โดยได้กำหนดเอาท์พุทที่ออกจากแต่ละปุ่มให้อยู่ในรูปของเลขฐาน 2 ดังตารางที่ 10.1

ปุ่มที่กำหนดไว้	เลขที่กำหนดไว้	เลขฐาน 2 ที่กำหนดไว้
FAN	1	0001
MODE	2	0010
POWER	5	0101
INC	4	0100
DEC	8	1000

ตารางที่ 10.1 แสดงค่าของฟังก์ชันที่ออกแบบเอาไว้

เมื่อทำการทดลองกดปุ่มต่าง ๆ ดูปรากฏว่าได้ผลการทดลองดังตารางที่ 10.2

เมื่อกดปุ่มที่กำหนด	เลขฐาน 2 ที่ได้จากการทดลอง	ได้เลขฐานสิบคือ
FAN	0001	1
MODE	0010	2
POWER	0101	5
INC	0100	4
DEC	1000	8

ตารางที่ 10.2 แสดงค่าที่ตรวจสอบได้จากชุดทดลอง

จะเห็นว่าค่าของเลขฐาน 2 ที่ได้จากวงจรชุดทดลองนั้นได้ค่าตรงกับที่ได้ออกแบบเอาไว้แต่แรกดังที่แสดงในตารางที่ 10.3 ดังนั้นจึงสรุปได้ว่าสามารถนำรีโมทมาช่วยควบคุมวงจรเครื่องปรับอากาศได้ตามที่ได้ออกแบบไว้

เมื่อกดปุ่มที่กำหนด	เลขฐาน 2 ที่กำหนดไว้	เลขฐาน 2 ที่ได้จากการทดลอง
FAN	0001	0001
MODE	0010	0010
POWER	0101	0101
INC	0100	0100
DEC	1000	1000

ตารางที่ 10.3 แสดงค่าที่ออกแบบไว้เปรียบเทียบกับผลการทดลอง

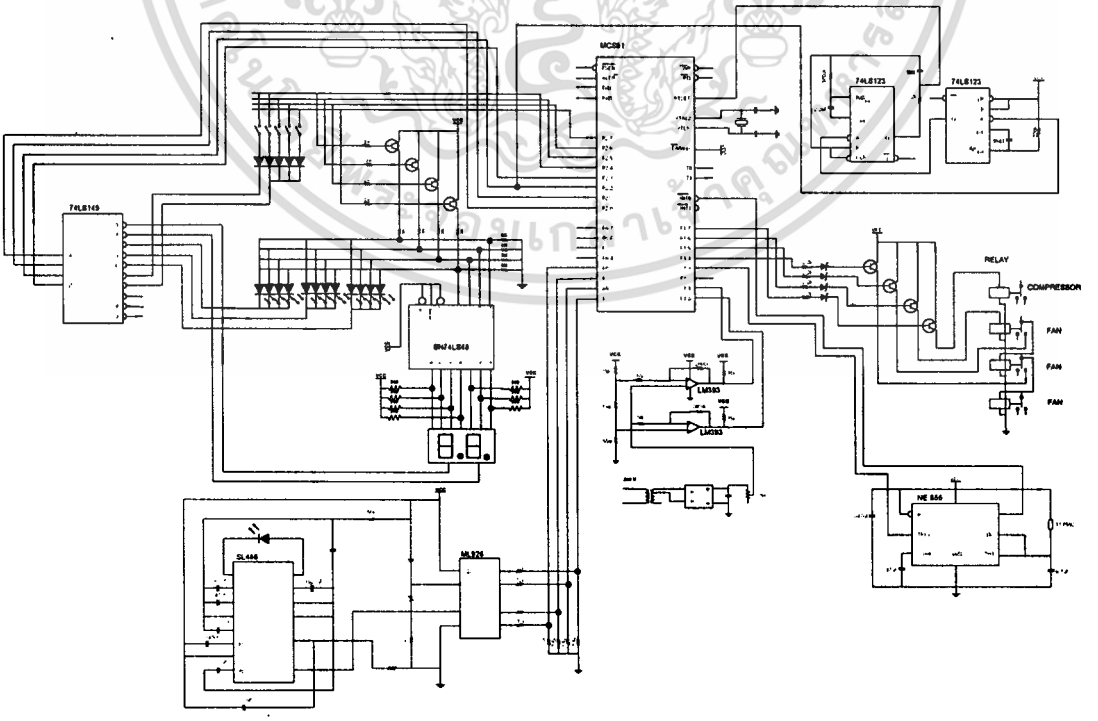
บทที่ 11
สรุปผล และ วิจารณ์

วงจรควบคุมการทำงานของเครื่องปรับอากาศด้วยรีโมทคอนโทรล โดยใช้ไมโครโปรเซสเซอร์ (Microprocessor) ในการควบคุมระบบทั้งหมด นี้สามารถมานำไปใช้งานได้จริงตามต้องการ อีกทั้งต้นทุนในการสร้างก็ไม่สูงนัก

แต่อย่างไรก็ตามวงจรที่ออกแบบลงแผ่นปริ้นท์ ยังคงมีขนาดไม่เล็กเท่าที่ควร และวงจรรีโมทก็ยังสามารถใช้ส่งคำสั่งต่าง ๆ ได้อีกหลายฟังก์ชัน ซึ่งสามารถนำไปพัฒนาให้วงจรมีขนาดเล็กลงและมีฟังก์ชันการทำงานมากกว่านี้ตามความต้องการของผู้พัฒนาต่อไป

วงจรรวมของระบบเครื่องปรับอากาศด้วยรีโมทโดยสรุป แสดงดังรูปที่ 11.1

การควบคุมอุณหภูมิภายในห้องให้ได้ระดับความเย็นตามที่ต้องการ โดยการใช้ไมโครโปรเซสเซอร์ควบคุมการทำงานของคอมเพรสเซอร์และพัดลมนี้ เป็นเพียงตัวอย่างหนึ่งของการประยุกต์ใช้ไมโครโปรเซสเซอร์เท่านั้น ในงานควบคุมและงานทางด้านวิศวกรรมอื่นๆก็สามารถที่จะนำไมโครโปรเซสเซอร์ ไปประยุกต์ใช้ได้อย่างกว้างขวาง ซึ่งจะทำให้การทำงานเป็นไปอย่างถูกต้องแม่นยำ และยังอำนวยความสะดวกสบายในการทำงานอีกด้วย



รูปที่ 11.1 แสดงวงจรโดยรวมของระบบ

หนังสืออ้างอิง

Kenneth J.Ayala , The 8051 Microcontroller Architecture, programming, and Application, West Publishing Company, 1991.

Microprocessor Data Book MCS-51 Microcontroller, Intel.

Signetics Microcontroller Users ' Guide ' , Signetics Company, 1989.

IC TTL Data Book, National Semiconductor Inc.

CMOS Data Book, Motorola Inc.

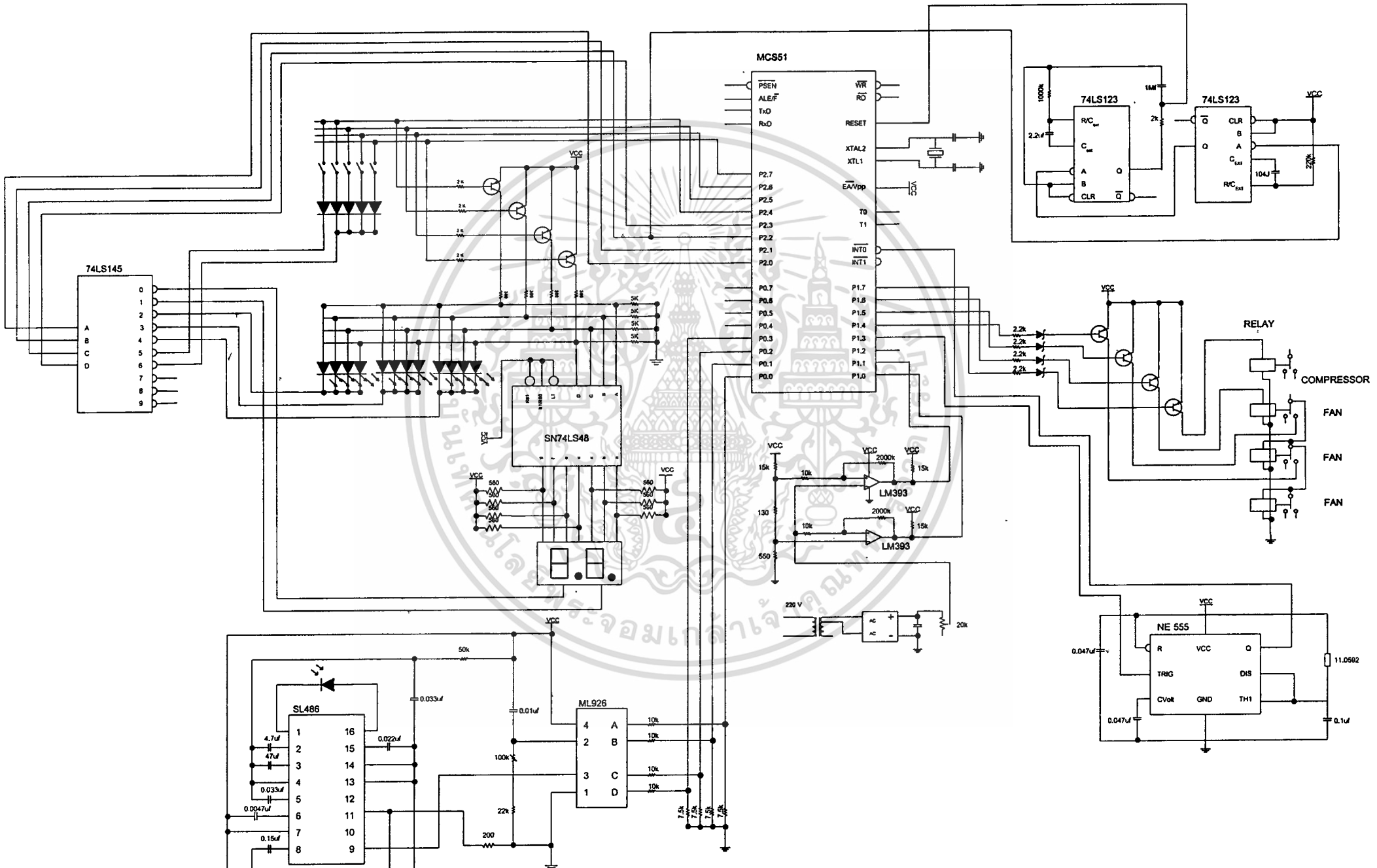
สุเจตน์ จันทรังษ์, ไมโครคอนโทรลเลอร์ซีพเดี่ยว 8051, วิทยาลัยมหานคร, 2535.

สุนทร วิทูสุรพจน์, การใช้งานไมโครคอนโทรลเลอร์ตระกูล 8051, บริษัท ซีเอ็ดยูเคชัน จำกัด, 2537.

รีโมท เครื่องควบคุมไร้สาย, บริษัท ซีเอ็ดยูเคชัน จำกัด, 2538.

มานพ หว่านนา, วรพิชญ์ ใจสะอาด, การควบคุมการทำงานของเครื่องปรับอากาศโดยใช้ไมโครโปรเซสเซอร์, 2536.

ภาคผนวก


MCS51
74LS145
74LS123
74LS123
SN74LS48
LM393
LM393
RELAY
COMPRESSOR
FAN
FAN
FAN
NE 555
SL486
ML926
220 V
VCC
GND

Features

- Compatible with MCS-51™ Products
- 4 Kbytes of In-System Reprogrammable Flash Memory
  - Endurance: 1,000 Write/Erase Cycles
- Fully Static Operation: 0 Hz to 24 MHz
- Three-Level Program Memory Lock
- 128 x 8-Bit Internal RAM
- 32 Programmable I/O Lines
- Two 16-Bit Timer/Counters
- Six Interrupt Sources
- Programmable Serial Channel
- Low Power Idle and Power Down Modes

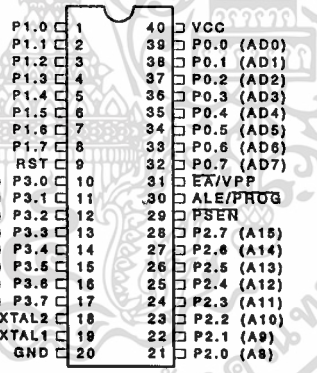
Description

The AT89C51 is a low-power, high-performance CMOS 8-bit microcomputer with 4 Kbytes of Flash Programmable and Erasable Read Only Memory (PEROM). The device is manufactured using Atmel's high density nonvolatile memory technology and is compatible with the industry standard MCS-51™ instruction set and pinout. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional nonvolatile memory programmer. By combining a versatile 8-bit CPU with Flash on a monolithic chip, the Atmel AT89C51 is a powerful microcomputer which provides a highly flexible and cost effective solution to many embedded control applications.

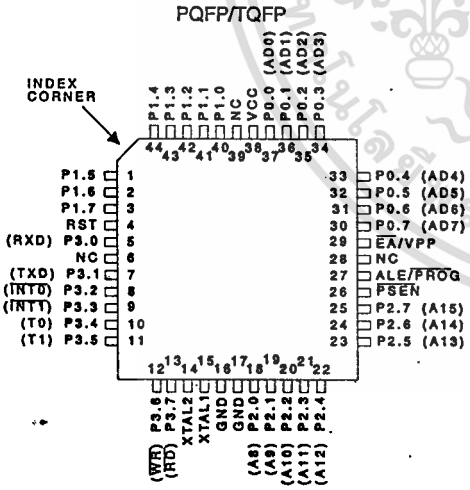
(continued)

Pin Configurations

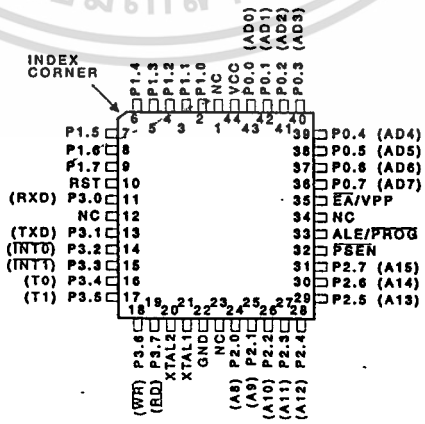
PDIP/Cerdip



PQFP/TQFP



PLCC/LCC

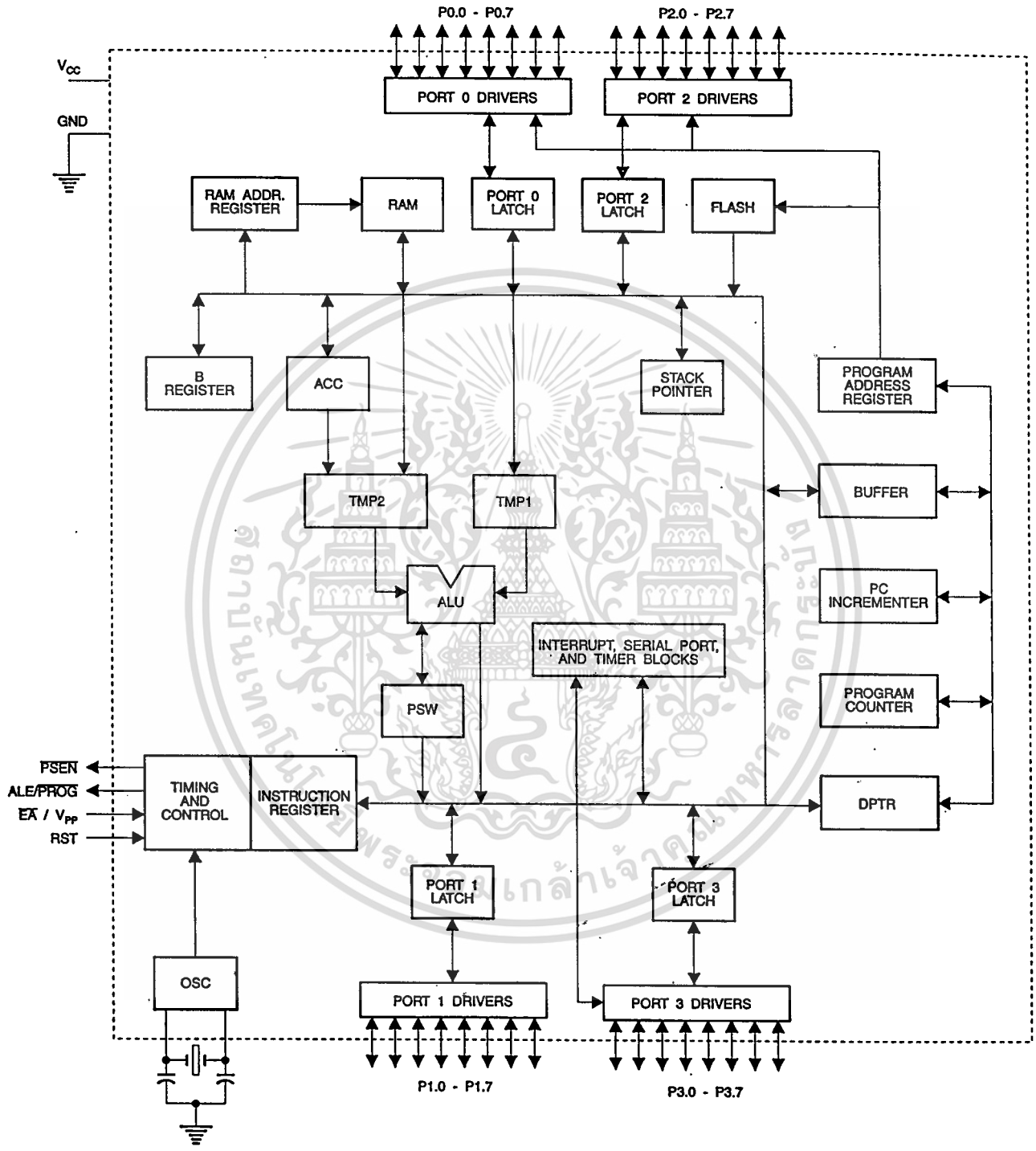


8-Bit Microcontroller with 4 Kbytes Flash

AT89C51

0265E

Block Diagram



Description (Continued)

The AT89C51 provides the following standard features: 4 Kbytes of Flash, 128 bytes of RAM, 32 I/O lines, two 16-bit timer/counters, a five vector two-level interrupt architecture, a full duplex serial port, on-chip oscillator and clock circuitry. In addition, the AT89C51 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port and interrupt system to continue functioning. The Power Down Mode saves the RAM contents but freezes the oscillator disabling all other chip functions until the next hardware reset.

Pin Description

V_{CC}

Supply voltage.

GND

Ground.

Port 0

Port 0 is an 8-bit open drain bidirectional I/O port. As an output port each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high-impedance inputs.

Port 0 may also be configured to be the multiplexed low-order address/data bus during accesses to external program and data memory. In this mode P0 has internal pullups.

Port 0 also receives the code bytes during Flash programming, and outputs the code bytes during program verification. External pullups are required during program verification.

Port 1

Port 1 is an 8-bit bidirectional I/O port with internal pullups. The Port 1 output buffers can sink/source four TTL inputs. When 1s are written to Port 1 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

Port 1 also receives the low-order address bytes during Flash programming and program verification.

Port 2

Port 2 is an 8-bit bidirectional I/O port with internal pullups. The Port 2 output buffers can sink/source four TTL inputs. When 1s are written to Port 2 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.

Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16-bit addresses (MOVX @ DPTR). In this application it uses strong internal pullups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOVX @ RI), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits and some control signals during Flash programming and verification.

Port 3

Port 3 is an 8-bit bidirectional I/O port with internal pullups. The Port 3 output buffers can sink/source four TTL inputs. When 1s are written to Port 3 pins they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (I_{IL}) because of the pullups.

Port 3 also serves the functions of various special features of the AT89C51 as listed below:

Port Pin	Alternate Functions
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (extenal interrupt 0)
P3.3	$\overline{INT1}$ (extenal interrupt 1)
P3.4	T0 (timer 0 extenal input)
P3.5	T1 (timer 1 external input)
P3.6	$\overline{WR}$ (extenal data memory write strobe)
P3.7	$\overline{RD}$ (external data memory read strobe)

Port 3 also receives some control signals for Flash programming and programming verification.

RST

Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/$\overline{PROG}$

Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input ($\overline{PROG}$) during Flash programming.

In normal operation ALE is emitted at a constant rate of 1/6 the oscillator frequency, and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external Data Memory.

If desired, ALE operation can be disabled by setting bit 0 of SFR location 8EH. With the bit set, ALE is active only during a MOVX or MOVC instruction. Otherwise, the pin is weakly pulled high. Setting the ALE-disable bit has no effect if the microcrontroller is in external execution mode.

$\overline{PSEN}$

Program Store Enable is the read strobe to external program memory.

(continued)

Pin Description (Continued)

When the AT89C51 is executing code from external program memory, PSEN is activated twice each machine cycle, except that two PSEN activations are skipped during each access to external data memory.

EA/VPP

External Access Enable. EA must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FFFFH. Note, however, that if lock bit 1 is programmed, EA will be internally latched on reset.

EA should be strapped to VCC for internal program executions.

This pin also receives the 12-volt programming enable voltage (VPP) during Flash programming, for parts that require 12-volt VPP.

XTAL1

Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

XTAL2

Output from the inverting oscillator amplifier.

Oscillator Characteristics

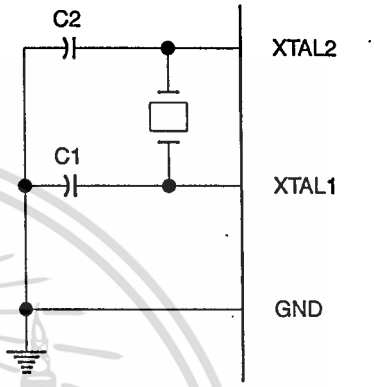
XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier which can be configured for use as an on-chip oscillator, as shown in Figure 1. Either a quartz crystal or ceramic resonator may be used. To drive the device from an external clock source, XTAL2 should be left unconnected while XTAL1 is driven as shown in Figure 2. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clocking circuitry is through a divide-by-two flip-flop, but minimum and maximum voltage high and low time specifications must be observed.

Idle Mode

In idle mode, the CPU puts itself to sleep while all the on-chip peripherals remain active. The mode is invoked by software. The content of the on-chip RAM and all the special functions registers remain unchanged during this mode. The idle mode can be terminated by any enabled interrupt or by a hardware reset.

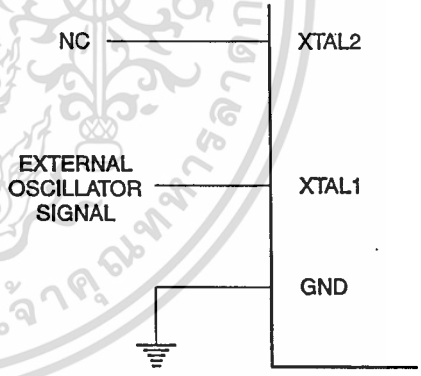
It should be noted that when idle is terminated by a hardware reset, the device normally resumes program execution, from where it left off, up to two machine cycles before the internal reset algorithm takes control. On-chip hard-

Figure 1. Oscillator Connections



Notes: C1, C2 = 30 pF ± 10 pF for Crystals
= 40 pF ± 10 pF for Ceramic Resonators

Figure 2. External Clock Drive Configuration



Status of External Pins During Idle and Power Down

Mode	Program Memory	ALE	PSEN	PORT0	PORT1	PORT2	PORT3
Idle	Internal	1	1	Data	Data	Data	Data
Idle	External	1	1	Float	Data	Address	Data
Power Down	Internal	0	0	Data	Data	Data	Data
Power Down	External	0	0	Float	Data	Data	Data

ware inhibits access to internal RAM in this event, but access to the port pins is not inhibited. To eliminate the possibility of an unexpected write to a port pin when Idle is terminated by reset, the instruction following the one that invokes Idle should not be one that writes to a port pin or to external memory.

Power Down Mode

In the power down mode the oscillator is stopped, and the instruction that invokes power down is the last instruction executed. The on-chip RAM and Special Function Registers retain their values until the power down mode is terminated. The only exit from power down is a hardware reset. Reset redefines the SFRs but does not change the on-chip RAM. The reset should not be activated before V_{CC} is restored to its normal operating level and must be held active long enough to allow the oscillator to restart and stabilize.

Program Memory Lock Bits

On the chip are three lock bits which can be left unprogrammed (U) or can be programmed (P) to obtain the additional features listed in the table below:

When lock bit 1 is programmed, the logic level at the $\overline{EA}$ pin is sampled and latched during reset. If the device is powered up without a reset, the latch initializes to a random value, and holds that value until reset is activated. It is necessary that the latched value of $\overline{EA}$ be in agreement with the current logic level at that pin in order for the device to function properly.

Lock Bit Protection Modes

Program Lock Bits				
	LB1	LB2	LB3	Protection Type
1	U	U	U	No program lock features.
2	P	U	U	MOVC instructions executed from external program memory are disabled from fetching code bytes from internal memory, $\overline{EA}$ is sampled and latched on reset, and further programming of the Flash is disabled.
3	P	P	U	Same as mode 2, also verify is disabled.
4	P	P	P	Same as mode 3, also external execution is disabled.

Programming the Flash

The AT89C51 is normally shipped with the on-chip Flash memory array in the erased state (that is, contents = FFH) and ready to be programmed. The programming interface accepts either a high-voltage (12-volt) or a low-voltage (V_{CC}) program enable signal. The low voltage programming mode provides a convenient way to program the AT89C51 inside the user's system, while the high-voltage programming mode is compatible with conventional third party Flash or EPROM programmers.

The AT89C51 is shipped with either the high-voltage or low-voltage programming mode enabled. The respective top-side marking and device signature codes are listed in the following table.

	V_{PP} = 12 V	V_{PP} = 5 V
Top-Side Mark	AT89C51 xxxx yyww	AT89C51 xxxx-5 yyww
Signature	(030H)=1EH (031H)=51H (032H)=FFH	(030H)=1EH (031H)=51H (032H)=05H

The AT89C51 code memory array is programmed byte-by-byte in either programming mode. *To program any non-blank byte in the on-chip Flash Memory, the entire memory must be erased using the Chip Erase Mode.*

Programming Algorithm: Before programming the AT89C51, the address, data and control signals should be set up according to the Flash programming mode table and Figures 3 and 4. To program the AT89C51, take the following steps.

1. Input the desired memory location on the address lines.
2. Input the appropriate data byte on the data lines.
3. Activate the correct combination of control signals.
4. Raise $\overline{EA}/V_{PP}$ to 12 V for the high-voltage programming mode.
5. Pulse ALE/$\overline{PROG}$ once to program a byte in the Flash array or the lock bits. The byte-write cycle is self-timed and typically takes no more than 1.5 ms. Repeat steps 1 through 5, changing the address and data for the entire array or until the end of the object file is reached.

$\overline{Data}$ Polling: The AT89C51 features $\overline{Data}$ Polling to indicate the end of a write cycle. During a write cycle, an at-

(continued)

Programming the Flash (Continued)

tempted read of the last byte written will result in the complement of the written datum on PO.7. Once the write cycle has been completed, true data are valid on all outputs, and the next cycle may begin. Data Polling may begin any time after a write cycle has been initiated.

Ready/Busy: The progress of byte programming can also be monitored by the RDY/BSY output signal. P3.4 is pulled low after ALE goes high during programming to indicate BUSY. P3.4 is pulled high again when programming is done to indicate READY.

Program Verify: If lock bits LB1 and LB2 have not been programmed, the programmed code data can be read back via the address and data lines for verification. The lock bits cannot be verified directly. Verification of the lock bits is achieved by observing that their features are enabled.

Chip Erase: The entire Flash array is erased electrically by using the proper combination of control signals and by holding ALE/PROG low for 10 ms. The code array is written with all "1"s. The chip erase operation must be executed before the code memory can be re-programmed.

Reading the Signature Bytes: The signature bytes are read by the same procedure as a normal verification of locations 030H,

031H, and 032H, except that P3.6 and P3.7 must be pulled to a logic low. The values returned are as follows.

(030H) = 1EH indicates manufactured by Atmel
(031H) = 51H indicates 89C51
(032H) = FFH indicates 12 V programming
(032H) = 05H indicates 5 V programming

Programming Interface

Every code byte in the Flash array can be written and the entire array can be erased by using the appropriate combination of control signals. The write operation cycle is self-timed and once initiated, will automatically time itself to completion.

All major programming vendors offer worldwide support for the Atmel microcontroller series. Please contact your local programming vendor for the appropriate software revision.

Flash Programming Modes

Mode		RST	PSEN	ALE/PROG	EA/VPP	P2.6	P2.7	P3.6	P3.7
Write Code Data		H	L	⎍	H/12V(1)	L	H	H	H
Read Code Data		H	L	H	H	L	L	H	H
Write Lock	Bit - 1	H	L	⎍	H/12V	H	H	H	H
	Bit - 2	H	L	⎍ (2)	H/12V	H	H	L	L
	Bit - 3	H	L	⎍	H/12V	H	L	H	L
Chip Erase		H	L	⎍	H/12V	H	L	L	L
Read Signature Byte		H	L	H	H	L	L	L	L

Notes: 1. The signature byte at location 032H designates whether V_{PP} = 12 V or V_{PP} = 5 V should be used to enable programming.

2. Chip Erase requires a 10 ms $\overline{PROG}$ pulse.

Figure 3. Programming the Flash

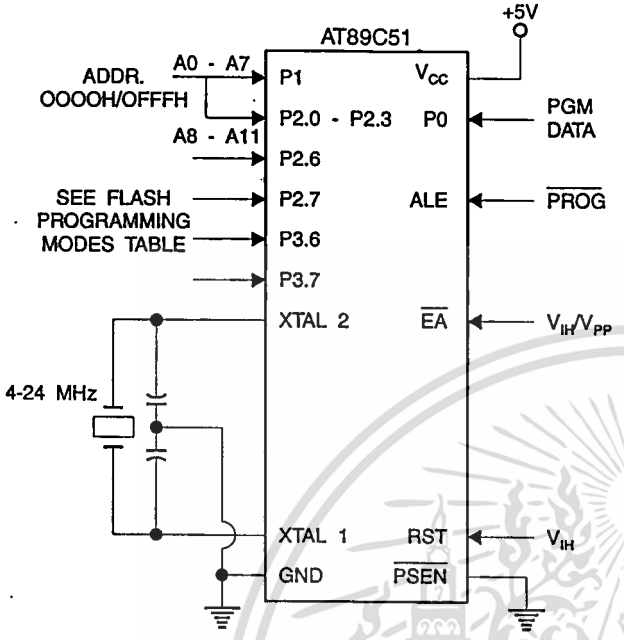
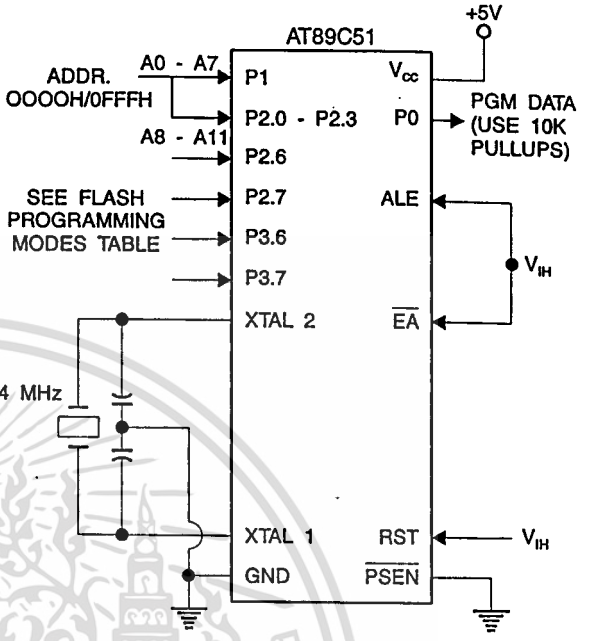


Figure 4. Verifying the Flash



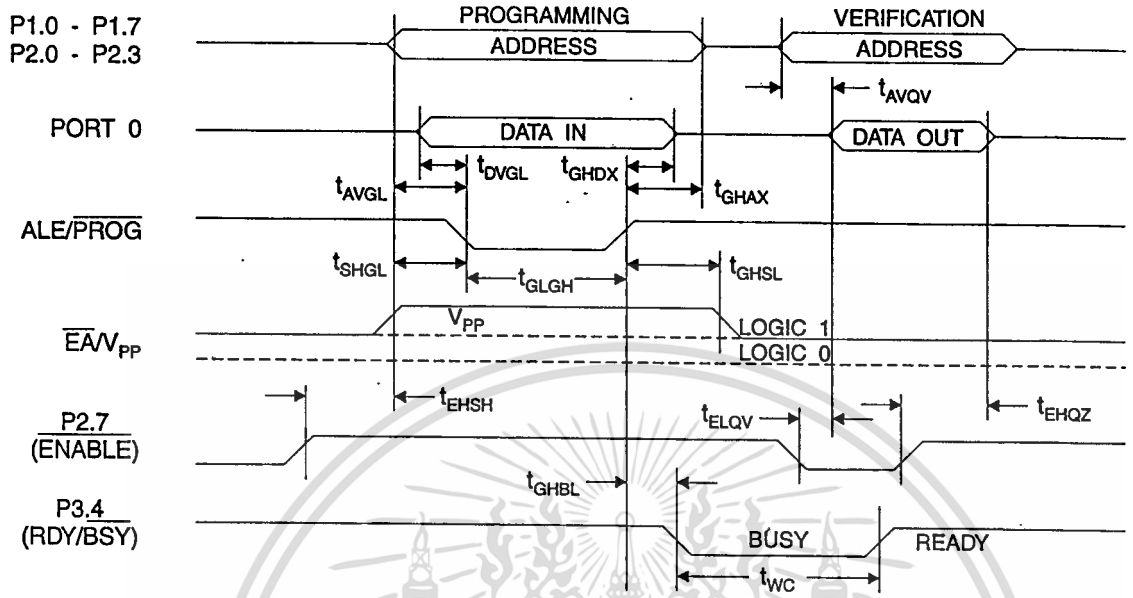
Flash Programming and Verification Characteristics

T_A = 21°C to 27°C, V_{CC} = 5.0 ± 10%

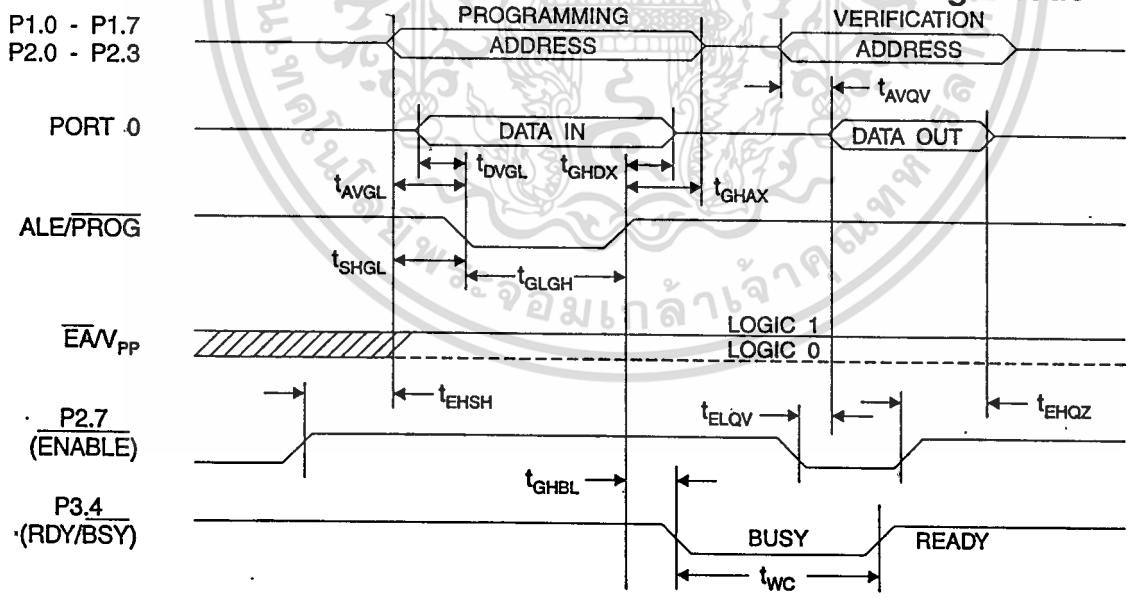
Symbol	Parameter	Min	Max	Units
V_{PP}(1)	Programming Enable Voltage	11.5	12.5	V
I_{PP}(1)	Programming Enable Current		1.0	mA
$1/t_{CLCL}$	Oscillator Frequency	4	24	MHz
t_{AVGL}	Address Setup to $\overline{PROG}$ Low	$48t_{CLCL}$		
t_{GHAX}	Address Hold After $\overline{PROG}$	$48t_{CLCL}$		
t_{DVGL}	Data Setup to $\overline{PROG}$ Low	$48t_{CLCL}$		
t_{GHDX}	Data Hold After $\overline{PROG}$	$48t_{CLCL}$		
t_{EHSH}	P2.7 ($\overline{ENABLE}$) High to V_{PP}	48tCLCL		
t_{SHGL}	V_{PP} Setup to $\overline{PROG}$ Low	10		µs
t_{GHSL}(1)	V_{PP} Hold After $\overline{PROG}$	10		µs
t_{GLGH}	$\overline{PROG}$ Width	1	110	µs
t_{AVQV}	Address to Data Valid		48tCLCL	
t_{ELQV}	$\overline{ENABLE}$ Low to Data Valid		$48t_{CLCL}$	
t_{EHQV}	Data Float After $\overline{ENABLE}$	0	$48t_{CLCL}$	
t_{GHBL}	$\overline{PROG}$ High to $\overline{BUSY}$ Low		1.0	µs
t_{WC}	Byte Write Cycle Time		2.0	ms

Note: 1. Only used in 12-volt programming mode.

Flash Programming and Verification Waveforms - High Voltage Mode



Flash Programming and Verification Waveforms - Low Voltage Mode



Absolute Maximum Ratings*

Operating Temperature	-55°C to +125°C
Storage Temperature	-65°C to +150°C
Voltage on Any Pin with Respect to Ground	-1.0 V to +7.0 V
Maximum Operating Voltage	6.6 V
DC Output Current	15.0 mA

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

D.C. Characteristics

T_A = -40°C to 85°C, V_{CC} = 5.0 V ± 20% (unless otherwise noted)

Symbol	Parameter	Condition	Min	Max	Units
V_{IL}	Input Low Voltage	(Except $\overline{EA}$)	-0.5	0.2 V_{CC}-0.1	V
V_{IL1}	Input Low Voltage ($\overline{EA}$)		-0.5	0.2 V_{CC}-0.3	V
V_{IH}	Input High Voltage	(Except XTAL1, RST)	0.2 V_{CC}+0.9	V_{CC}+0.5	V
V_{IH1}	Input High Voltage	(XTAL1, RST)	0.7 V_{CC}	V_{CC}+0.5	V
V_{OL}	Output Low Voltage[1] (Ports 1,2,3)	I_{OL} = 1.6 mA		0.45	V
V_{OL1}	Output Low Voltage[1] (Port 0, ALE, $\overline{PSEN}$)	I_{OL} = 3.2 mA		0.45	V
V_{OH}	Output High Voltage (Ports 1,2,3, ALE, $\overline{PSEN}$)	I_{OH} = -60 µA, V_{CC} = 5 V ± 10%	2.4		V
		I_{OH} = -25 µA	0.75 V_{CC}		V
		I_{OH} = -10 µA	0.9 V_{CC}		V
V_{OH1}	Output High Voltage (Port 0 in External Bus Mode)	I_{OH} = -800 µA, V_{CC} = 5 V ± 10%	2.4		V
		I_{OH} = -300 µA	0.75 V_{CC}		V
		I_{OH} = -80 µA	0.9 V_{CC}		V
I_{IL}	Logical 0 Input Current (Ports 1,2,3)	V_{IN} = 0.45 V		-50	µA
I_{TL}	Logical 1 to 0 Transition Current (Ports 1,2,3)	V_{IN} = 2 V		-650	µA
I_{LI}	Input Leakage Current (Port 0, $\overline{EA}$)	0.45 < V_{IN} < V_{CC}		±10	µA
RRST	Reset Pulldown Resistor		50	300	KΩ
C_{IO}	Pin Capacitance	Test Freq. = 1 MHz, T_A = 25°C		10	pF
I_{CC}	Power Supply Current	Active Mode, 12 MHz		20	mA
		Idle Mode, 12 MHz		5	mA
	Power Down Mode[2]	V_{CC} = 6 V		100	µA
		V_{CC} = 3 V		40	µA

Notes: 1. Under steady state (non-transient) conditions, I_{OL} must be externally limited as follows:
Maximum I_{OL} per port pin:10 mA
Maximum I_{OL} per 8-bit port:
Port 0:26 mA
Ports 1,2, 3:15 mA
Maximum total IOL for all output pins:71 mA
If IOL exceeds the test condition, VOL may exceed the related specification. Pins are not guaranteed to sink current greater than the listed test conditions.
2. Minimum VCC for Power Down is 2 V.

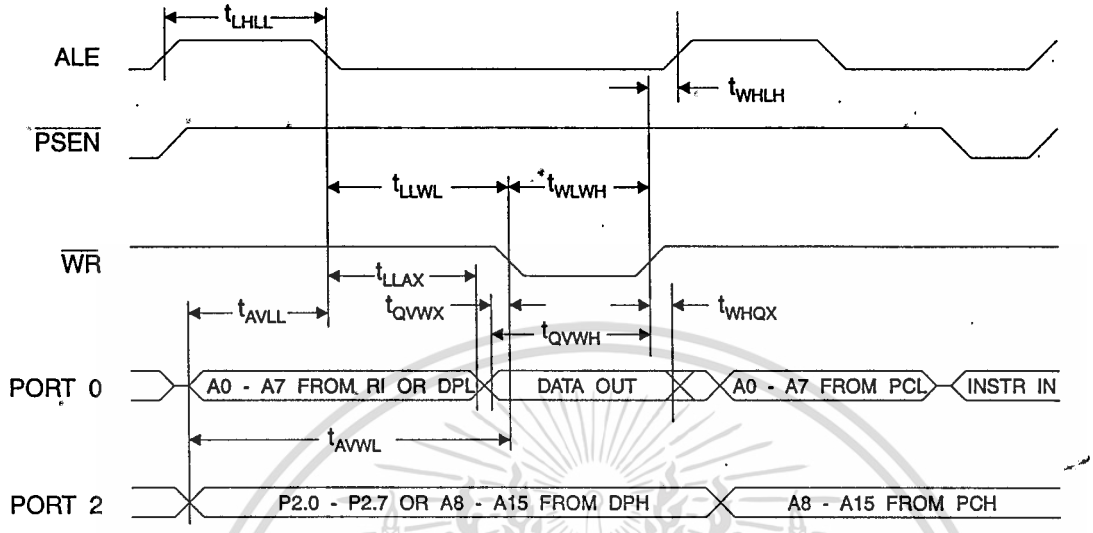
A.C. Characteristics

(Under Operating Conditions; Load Capacitance for Port 0, ALE/$\overline{PROG}$, and $\overline{PSEN}$ = 100 pF; Load Capacitance for all other outputs = 80 pF)

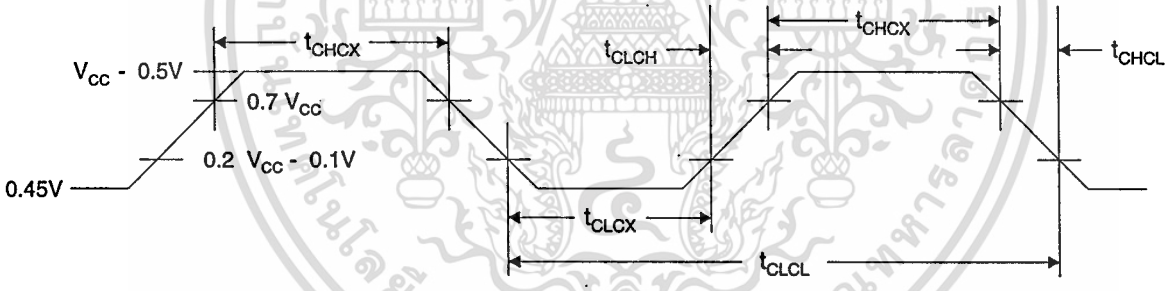
External Program and Data Memory Characteristics

Symbol	Parameter	12 MHz Oscillator		16 to 24 MHz Oscillator		Units
		Min	Max	Min	Max	
$1/t_{CLCL}$	Oscillator Frequency			0	24	MHz
t_{LHLL}	ALE Pulse Width	127		$2t_{CLCL}$-40		ns
t_{AVLL}	Address Valid to ALE Low	28		tCLCL-13		ns
t_{LLAX}	Address Hold After ALE Low	48		tCLCL-20		ns
t_{LLIV}	ALE Low to Valid Instruction In		233		4tCLCL-65	ns
t_{LLPL}	ALE Low to $\overline{PSEN}$ Low	43		t_{CLCL}-13		ns
t_{PLPH}	$\overline{PSEN}$ Pulse Width	205		$3t_{CLCL}$-20		ns
t_{PLIV}	$\overline{PSEN}$ Low to Valid Instruction In		145		$3t_{CLCL}$-45	ns
t_{PXIX}	Input Instruction Hold After $\overline{PSEN}$	0		0		ns
t_{PXIZ}	Input Instruction Float After $\overline{PSEN}$		59		t_{CLCL}-10	ns
t_{PXAV}	$\overline{PSEN}$ to Address Valid	75		t_{CLCL}-8		ns
t_{AVIV}	Address to Valid Instruction In		312		$5t_{CLCL}$-55	ns
t_{PLAZ}	$\overline{PSEN}$ Low to Address Float		10		10	ns
t_{RLRH}	$\overline{RD}$ Pulse Width	400		$6t_{CLCL}$-100		ns
t_{WLWH}	$\overline{WR}$ Pulse Width	400		$6t_{CLCL}$-100		ns
t_{RLDV}	$\overline{RD}$ Low to Valid Data In		252		$5t_{CLCL}$-90	ns
t_{RHDX}	Data Hold After $\overline{RD}$	0		0		ns
t_{RHDZ}	Data Float After $\overline{RD}$		97		$2t_{CLCL}$-28	ns
t_{LLDV}	ALE Low to Valid Data In		517		8tCLCL-150	ns
t_{AVDV}	Address to Valid Data In		585		9tCLCL-165	ns
t_{LLWL}	ALE Low to $\overline{RD}$ or $\overline{WR}$ Low	200	300	$3t_{CLCL}$-50	$3t_{CLCL}$+50	ns
t_{AVWL}	Address to $\overline{RD}$ or $\overline{WR}$ Low	203		$4t_{CLCL}$-75		ns
t_{QVWX}	Data Valid to $\overline{WR}$ Transition	23		t_{CLCL}-20		ns
t_{QVWH}	Data Valid to $\overline{WR}$ High	433		$7t_{CLCL}$-120		ns
t_{WHQX}	Data Hold After $\overline{WR}$	33		t_{CLCL}-20		ns
t_{RLAZ}	$\overline{RD}$ Low to Address Float		0		0	ns
t_{WHLH}	$\overline{RD}$ or $\overline{WR}$ High to ALE High	43	123	t_{CLCL}-20	t_{CLCL}+25	ns

External Data Memory Cycle



External Clock Drive Waveforms



External Clock Drive

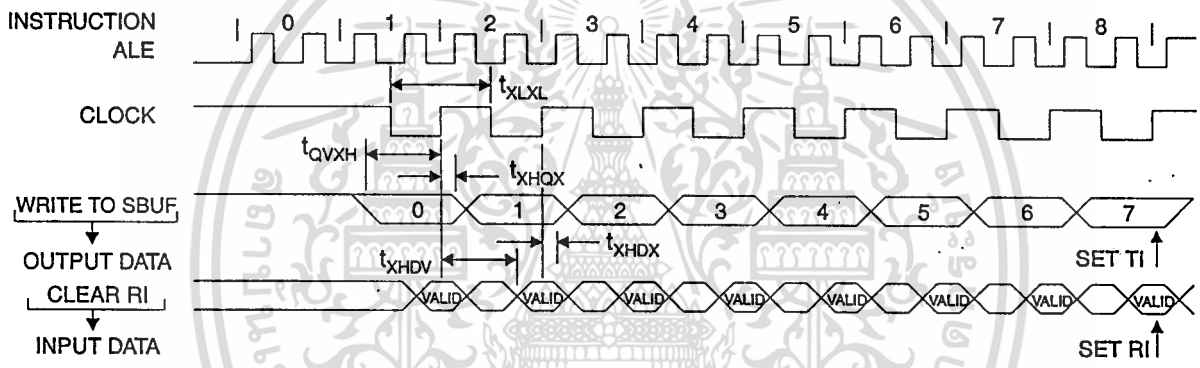
Symbol	Parameter	Min	Max	Units
$1/t_{CLCL}$	Oscillator Frequency	0	24	MHz
t_{CLCL}	Clock Period	41.6		ns
t_{CHCX}	High Time	15		ns
t_{CLCX}	Low Time	15		ns
t_{CLCH}	Rise Time		20	ns
t_{CHCL}	Fall Time		20	ns

Serial Port Timing: Shift Register Mode Test Conditions

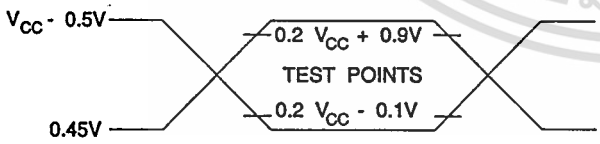
(V_{CC} = 5.0 V ± 20%; Load Capacitance = 80 pF)

Symbol	Parameter	12 MHz Osc		Variable Oscillator		Units
		Min	Max	Min	Max	
t_{XLXL}	Serial Port Clock Cycle Time	1.0		$12t_{CLCL}$		µs
t_{QVXH}	Output Data Setup to Clock Rising Edge	700		$10t_{CLCL}$-133		ns
t_{XHQX}	Output Data Hold After Clock Rising Edge	50		$2t_{CLCL}$-33		ns
t_{XHDX}	Input Data Hold After Clock Rising Edge	0		0		ns
t_{XHDV}	Clock Rising Edge to Input Data Valid		700		10tCLCL-133	ns

Shift Register Mode Timing Waveforms

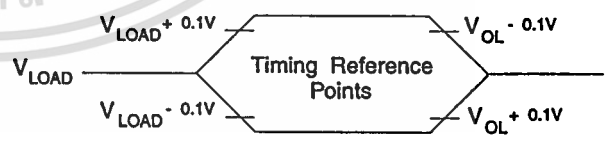


AC Testing Input/Output Waveforms[1]



Note: 1. AC Inputs during testing are driven at V_{CC} - 0.5 V for a logic 1 and 0.45 V for a logic 0. Timing measurements are made at V_{IH} min. for a logic 1 and V_{IL} max. for a logic 0.

Float Waveforms[1]



Note: 1. For timing purposes, a port pin is no longer floating when a 100 mV change from load voltage occurs. A port pin begins to float when a 100 mV change from the loaded V_{OH}/V_{OL} level occurs.

Ordering Information

Speed (MHz)	Power Supply	Ordering Code	Package	Operation Range
12	5 V ± 20%	AT89C51-12AC AT89C51-12JC AT89C51-12PC AT89C51-12QC	44A 44J 40P6 44Q	Commercial (0°C to 70°C)
		AT89C51-12AI AT89C51-12JI AT89C51-12PI AT89C51-12QI	44A 44J 40P6 44Q	Industrial (-40°C to 85°C)
		AT89C51-12AA AT89C51-12JA AT89C51-12PA AT89C51-12QA	44A 44J 40P6 44Q	Automotive (-40°C to 125°C)
	5 V ± 10%	AT89C51-12DM AT89C51-12LM	40D6 44L	Military (-55°C to 125°C)
		AT89C51-12DM/883 AT89C51-12LM/883	40D6 44L	Military/883C Class B, Fully Compliant (-55°C to 125°C)
16	5 V ± 20%	AT89C51-16AC AT89C51-16JC AT89C51-16PC AT89C51-16QC	44A 44J 40P6 44Q	Commercial (0°C to 70°C)
		AT89C51-16AI AT89C51-16JI AT89C51-16PI AT89C51-16QI	44A 44J 40P6 44Q	Industrial (-40°C to 85°C)
		AT89C51-16AA AT89C51-16JA AT89C51-16PA AT89C51-16QA	44A 44J 40P6 44Q	Automotive (-40°C to 125°C)
20	5 V ± 20%	AT89C51-20AC AT89C51-20JC AT89C51-20PC AT89C51-20QC	44A 44J 40P6 44Q	Commercial (0°C to 70°C)
		AT89C51-20AI AT89C51-20JI AT89C51-20PI AT89C51-20QI	44A 44J 40P6 44Q	Industrial (-40°C to 85°C)

Ordering Information

Speed (MHz)	Power Supply	Ordering Code	Package	Operation Range
24	5 V ± 20%	AT89C51-24AC AT89C51-24JC AT89C51-24PC AT89C51-24QC	44A 44J 44P6 44Q	Commercial (0°C to 70°C)
		AT89C51-24AI AT89C51-24JI AT89C51-24PI AT89C51-24QI	44A 44J 44P6 44Q	Industrial (-40°C to 85°C)

Package Type	
44A	44 Lead, Thin Plastic Gull Wing Quad Flatpack (TQFP)
40D6	40 Lead, 0.600" Wide, Non-Windowed, Ceramic Dual Inline Package (Cerdip)
44J	44 Lead, Plastic J-Leaded Chip Carrier (PLCC)
44L	44 Pad, Non-Windowed, Ceramic Leadless Chip Carrier (LCC)
40P6	40 Lead, 0.600" Wide, Plastic Dual Inline Package (PDIP)
44Q	44 Lead, Plastic Gull Wing Quad Flatpack (PQFP)

```
mode   equ 30h
temp   equ 31h
setp   equ 32h
fan    equ 33h
time0  equ 34h
time1  equ 35h
stime  equ 36h
newkey equ 37h
count  equ 38h
xcount equ 39h
ch0    equ 20h
ch1    equ 21h
ch2    equ 22h
ch3    equ 23h
ch4    equ 24h
power  equ 17h
auto   equ 16h
tempb  equ 15h
low    equ 14h
comp   equ 1fh
timeb1 equ 25h
bset   equ 1dh
med    equ 1ch
ovb    equ 27h
high   equ 26h
timeb0 equ 1eh
sleep  equ 24h

counth0 equ 3ah
counth1 equ 3bh
counth2 equ 3ch
counth3 equ 3dh
counttt0 equ 3eh
countt1 equ 3fh
countt2 equ 40h
countt3 equ 41h
countm0 equ 42h
countm1 equ 43h
countm2 equ 44h
figh3   equ 2bh
figt3   equ 2fh
```

```
        flgm2   equ 32h
        nowcounm  equ 33h
        nowcounh  equ 34h
        nowcount  equ 35h
        timef0  equ 36h
        timef1  equ 37h
        upbit   equ 38h
        counttwo  equ 39h
        sbit    equ 3ah
        timeb0x equ 3eh
        sleepx  equ 3fh
        flg     equ 3bh
        err_bitl equ 3ch
        err_bith equ 3dh
        delreg   equ 45h
        delreg2  equ 46h
        tempx    equ 47h
        tempy    equ 48h
        tempxx   equ 49h
        tempyy   equ 4ah

        ;-----------------

;here is the program origin
        org 0000h
        sjmp begins
        org 0003h
        ljmp caltemp
        org 001bh
        ljmp trigg

begins: mov  delreg,#00h
        acall delay

;initial condition and values

begins1:
        mov mode,#00h
        mov temp,#1bh ;27 c
        mov setp,#17h ;23 c
        mov fan,#03h
```

```
mov time0,#00h
mov stime,#00h
mov time1,#00h
mov ch0,#00h
mov ch1,#00h
mov ch2,#0dh
mov ch3,#0ch
mov ch4,#0bh
mov counth0,#0c8h
mov counth1,#0c8h
mov counth2,#0fh
mov countt0,#0c8h
mov countt1,#0c8h
mov countm0,#0c8h
mov countm1,#32h
mov counth3,#05h
mov countt2,#0fh
mov countm2,#0fh
mov xcount,#0ah
clr flgh3
clr flgt3
setb flgm2
setb nowcounm
clr  nowcounh
clr  nowcount
clr  timef0
clr  timef1
clr  upbit
setb counttwo
clr  flg
clr  err_bitl
clr  err_bith
clr  sleepx
clr  timeb0x   ;ins
clr  timeb0x
clr  sbit
setb p1.0
setb p1.1
setb p1.4   ;comp
setb p1.5   ;high
setb p1.6   ;med
```

```
        setb p1.7   ;low
        mov  tcon,#01h
        mov  tmod,#11h
        mov  ie,#89h
        mov  ip,#08h
        mov  tl0,#00h
        mov  th0,#00h
        mov  tl1,#18h
        mov  th1,#0fch
        mov  sp,#07h

; start to send pluses
        setb p1.3
        setb tcon.6

;system loop

 scan:

        jb    power,regular
        sjmp  scan1
 regular:
         acall  process
   scan1: mov   r7,#07h
   loop1: dec   r7
         cjne  r7,#04h,jp

        acall  remote      ;scan input from remote
        jz    lp1
        mov   r6,a
        acall  disp
        acall  remote
        xrl   a,r6
        jnz   loop1
        mov   a,r6

        cjne  r6,#05h,rem_cov
     lp3: acall  remote
        jnz   lp3
        sjmp  c6_key
```

```
rem_cov: acall  convt
        jb    flg,loop1
        acall  delay
        mov    newkey,a
  lp2:  acall  remote
        jnz    lp2
        sjmp   inst_ch
  lp1:
        ajmp   scan
jp:
        acall keyscan
        jz    loop1
        mov   r6,a
        acall disp
        acall keyscan
        xrl   a,r6
        jnz   loop1

        cjne  r7,#06h,c6_key
        acall convt
        jb    flg,loop1
        mov   newkey,a
        sjmp  inst_ch

 c6_key: mov   newkey,#04h
inst_ch: jb    power,inst
        mov   a,newkey
        cjne  a,#04h,scan
 inst:   acall instruct
loop2:
        mov   a,time0
        acall disp
        acall keyscan
        jnz   loop2
        ajmp  loop1
```

```
;subroutine

keyscan:
;scan keyboard
        mov  a,r7
        orl  a,#0f0h
        mov  p2,a
        mov  c,p2.4
        mov  acc.0,c
        mov  c,p2.5
        mov  acc.1,c
        mov  c,p2.6
        mov  acc.2,c
        mov  c,p2.7
        mov  acc.3,c
        cpl  a
        anl  a,#0fh
        ret

remote:
;scan remote from port 0
        clr  a
        mov  p0,#0fh
        mov  c,p0.0
        mov  acc.0,c
        mov  c,p0.1
        mov  acc.1,c
        mov  c,p0.2
        mov  acc.2,c
        mov  c,p0.3
        mov  acc.3,c
        anl  a,#0fh
        ret

disp:
;scan display

        jnb  bset,m_find
        jb   err_bitl,no_fnd
        jb   err_bith,no_fnd
m_find:  acall fnddata
```

```
        acall brink
        sjmp dis_p
no_fnd:  mov  ch0,#0ffh
        mov  ch1,#0eeh
        acall brink
dis_p:  mov  r0,#ch0  ;point to chx
        mov  r1,#05h  ;number of column
        mov  r2,#00h  ;value of column scaning
lpdp:   mov  a,@r0
        acall show
        inc  r0
        inc  r2
        djnz r1,lpdp
        ret

show:
        cpl  a
        mov  p2,a
        mov  delreg,#04h
        acall delay
        ret

delay:
        mov  delreg2,#00h
    wts: djnz delreg2,wts
        djnz delreg,delay
        ret

fnddata:
;find data for show in seven segment

        jnb  tempb,disps
        mov  a,temp
        sjmp conv
   disps: jnb  bset,dispt0
        mov  a,setp
        sjmp conv
   dispt0: jnb  timeb0x,dispt1
        mov  a,stime
        sjmp conv
   dispt1: jnb  timeb0x,sl
```

```
        mov  a,time1
    conv: mov  r4,a    ;store value for disp to seven segment
        mov  dptr,#lowbyte
        movc a,@a+dptr
        mov  ch0,a
        mov  a,r4
        mov  dptr,#higbyte
        movc a,@a+dptr
        mov  ch1,a
        ret
     sl: jnb  sleepx,emp
        mov  ch0,#5fh
        mov  ch1,#0eeh
    emp:  ret

convt:
;give value to key pressed
        clr  flg
        clr  a
        cjne  r6,#01h,one
        ret
    one:  inc  a
        cjne  r6,#02h,two
        ret
    two:  inc  a
        cjne  r6,#04h,three
        ret
    three: inc  a
        cjne  r6,#08h,bad
        ret
     bad:  setb  flg
        ret


   brink:
        jnb   timef1,tibrink
        cpl   timeb1
   tibrink: jnb   timef0,slbrink
        cpl   timeb0
   slbrink: jnb   upbit,nobrink
        cpl   sleep
   nobrink: ret
```

```
;instruct:scan keyboard to recicve instruction from user
instruct:
;'power'key
      mov   a,newkey
      cjne  a,#04h,modkey
      jnb   power,onn
      clr   power
      ljmp  begins
  onn: setb  power
      setb  auto
      setb  high
      setb  tempb
      acall disp
      ret

  modkey:
      mov   a,newkey
      cjne  a,#01h,upkey
      mov   a,mode
      cjne  a,#04h,adds
      mov   mode,#00h
      sjmp  tempm
adds: inc   mode
tempm: mov  a,mode
      cjne  a,#00h,setmod
      clr   sleep
      clr   sleepx
      setb  tempb
      ret
setmod: mov  a,mode
      cjne  a,#1h,timer_0
      clr   tempb
      setb  bset
      ret
timer_0: mov  a,mode
      cjne  a,#2h,timer_1
      clr   bset
      setb  timeb0
      setb  timeb0x
      ret
timer_1: mov  a,mode
```

```
        cjne  a,#03h,ssleep
        clr   timeb0
        clr   timeb0x
        setb  timeb1
        setb  timeb0x
        ret
ssleep: mov   a,mode
        cjne  a,#04,outb1
        clr   timeb1
        clr   timeb0x
        setb  sleep
        setb  sleepx
outb1: ret

upkey:
        mov   a,newkey
        cjne  a,#02h,go_downkey
        jnb   sleepx,increas
        setb  upbit
        setb  counttwo
        ret

go_downkey:ljmp  downkey
increas:jnb   bset,tim0
        jnb   err_bitl,not_err
        clr   err_bitl
        ret
not_err:inc   setp
        mov   a,#1ch
        subb  a,setp
        jbc   0d7h,err
        ret
err:    mov   setp,#1ch
        mov   ch0,#0ffh
        mov   ch1,#0eeh
        setb  err_bith
        ret
tim0:   jnb   timeb0x,tim1
        jnb   timef0,set_f0
        clr   nowcount    ;setting new time
        inc   time0
```

```
        inc   stime
        mov   a,time0
        cjne  a,#19h,n_over
        mov   time0,#01h
        mov   stime,#01h
n_over: ret
set_f0: jb    timef1,downkey
        setb  timef0
        mov   time0,#01h
        mov   stime,#01h
        ret
tim1:   jnb   timeb0x,outbx
        jnb   timef1,set_f1
        inc   time1
        mov   a,time1
        cjne  a,#19h,n_overs
        mov   time1,#01h
n_overs: ret
set_f1: jb    timef0,outbx
        setb  timef1
        mov   time1,#01h
        setb  p1.4
        clr   comp
        setb  p1.5
        clr   high
        setb  p1.6
        clr   med
        setb  p1.7
        clr   low
        clr   auto
        clr   count
        clr   sleep
        clr   countt1
outbx:  ret

downkey:mov   a,newkey
        cjne  a,#03h,fankey
        jnb   bset,ti0
        jnb   err_bith,not_er
        clr   err_bith
        ret
```

```
not_er: dec   setp
        mov   a,setp
        subb  a,#15h
        jbc   0d7h,er
        ret
er:     mov   setp,#15h
        mov   ch0,#0ffh
        mov   ch1,#0eeh
        setb  err_bitl
        ret
ti0:    jnb   timeb0x,ti1
        mov   a,time0
        cjne  a,#00h,do
        ret
do:     dec   time0
        dec   stime
        clr   nowcount      ;seting new time
        ret
ti1:    jnb   timeb0x,slep
        mov   a,time1
        cjne  a,#00h,does
        ret
does:   dec   time1
        ret
slep:   jnb   sleepx,outb
        clr   upbit
        setb  sleep
        setb  sleepx
        ret

fankey:
        mov   a,newkey
        cjne  a,#00h,outb
        mov   a,fan
        cjne  a,#03h,ad
        mov   fan,#00h
        sjmp  lo
ad:     inc   fan
lo:     mov   a,fan
        cjne  a,#00h,me
        clr   auto
```

```
        setb  low
        ret
me:     mov   a,fan
        cjne  a,#01h,hi
        clr   low
        setb  med
        ret
hi:     mov   a,fan
        cjne  a,#02h,aut
        clr   med
        setb  high
        ret
aut:    mov   a,fan
        cjne  a,#03h,outb
        mov   a,setp
        add   a,#02h
        clr   psw.7
        subb  a,temp
        jbc   psw.7,nclr_hi
        clr   high
nclr_hi: setb  auto
outb:   ret

process:
;all functions in the system

        acall  conv_tmp
        acall  safty
        jb     timef1,go_time_on
        acall  compr
        acall  fans
        acall  disp
        acall  sleepmod
        acall  disp
        acall  autotime
go_time_on:acall  time_on
        ret
```

```
;safty:check if power apply to the compressor is normal level
  safty:
        acall   recieve
        jz      pass
        mov     delreg,#01h
        acall   delay
        acall   recieve
        jz      pass
        setb    p1.4
        clr     comp
        setb    ovb
        mov     countm0,#0c8h ;count s minuts
        mov     countm1,#032h
        mov     countm2,#0fh
        clr     flgm2
        setb    nowcounm
        ret
  pass: clr     ovb
        ret

  recieve:
  ;recieve status bits
        mov     c,p1.0
        mov     acc.0,c
        mov     c,p1.1
        mov     acc.1,c
        orl     a,#11111100b
        cpl     a
        ret

  ;compr:make decision to operate with compressor,open or close
  compr:
        jb      ovb,oute
        clr     psw.7
        mov     a,setp
        subb    a,temp
        jbc     psw.7,oncomp   ;temp > setp
        setb    p1.4
        clr     comp
        jb      nowcounm,not_sets
        mov     countm0,#0c8h   ;count for 3 minuts
```

```
        mov    countm1,#032h
        mov    countm2,#0fh
        clr    flgm2
        setb   nowcounm
not_sets: ret
 oncomp: jnb    flgm2,oute
        clr    flgm2
        clr    nowcounm
        clr    p1.4
        setb   comp
  oute: ret

;fan:make decision on select speed of fan
fans:
        mov    a,fan
        cjne   a,#03h,lows
  au_to: clr    psw.7
        mov    a,setp
        add    a,#02h
        subb   a,temp
        jbc    psw.7,auhi    ;temp > setp+2
        mov    a,temp
        subb   a,setp
        jbc    psw.7,aulo    ;temp < setp
        mov    a,setp
        inc    a
        subb   a,temp
        jz     aume
        ret
   lows: mov    a,fan
        cjne   a,#00h,meds
   aulo: setb   p1.6
        clr    med
        setb   p1.5
        clr    high
        clr    p1.7
        setb   low
        ret
   meds: mov    a,fan
        cjne   a,#01h,higs
   aume: setb   p1.7
```

```
        clr    low
        setb   p1.5
        clr    high
        clr    p1.6
        setb   med
        ret
  higs:
        mov    a,fan
        cjne   a,#02h,outf
  auhi: setb   p1.6
        clr    med
        setb   p1.7
        clr    low
        clr    p1.5
        setb   high
  outf: ret

;sleepmod:if this mode is choosen ,set point will be increase for 1 degree
;every 1 hour and will not increas exeed 2 degrees

  sleepmod:
        jnb    upbit,outc
        jb     nowcounh,cros
        mov    counth0,#0c8h  ;set values for 1 hour counting
        mov    counth1,#0c8h
        mov    counth2,#0fh
        mov    counth3,#05h
        clr    flgh3
        setb   nowcounh
  cros: jnb    flgh3,outc
        clr    nowcounh
        clr    flgh3
        inc    setp
        cpl    counttwo
        jnb    counttwo,outc   ;if counttwo=1 means 2 rounds,out sleepmod
        clr    upbit
  outc: ret
```

```
;autotime:stop or start work automatically by setting time in hours

autotime:
    jnb    timef0,outd
    jb     nowcount,across
    mov    a,time0
    jz     n_counts
    mov    countt0,#0c8h  ;set values
    mov    countt1,#0c8h
    mov    countt2,#0fh
    mov    b,#05h
    mov    a,time0
    mul    ab
    mov    countt3,a
    clr    flgt3
    setb   nowcount
across: jnb    sbit,across1  ;check to dec stime every 1 hr
    dec    stime
    clr    sbit
    mov    xcount,#0ah
across1: jnb   flgt3,outd
    clr    nowcount
    clr    flgt3
    clr    timef0
    ljmp   begins1
n_counts:
    clr    nowcount
    clr    flgt3
    clr    timef0
    setb   timeb0
    setb   timeb0x
outd:  ret


time_on:
;count time to start work automatically
    jnb    timef1,timing
    mov    a,time1
    jz     go_out
    jb     nowcount,cross
    mov    b,#05h
```

```
        mul    ab
        mov    countt3,a
        mov    countt0,#0c8h
        mov    countt1,#0c8h
        mov    countt2,#0fh
        clr    flgt3
        setb   nowcount
cross:  jnb    flgt3,go_out
        clr    nowcount
        clr    flgt3
        clr    timef1
        ret
go_out: clr    nowcount
        clr    flgt3
        clr    timef1
        setb   timeb1
        setb   timeb0x
        ret

timing:
;count   time has been set in the program

;count 1 hour
        push acc
        jnb  nowcounh,settime
 dec_0: dec  counth0
        mov  a,counth0
        jz   dec_1
        sjmp settime
 dec_1: dec  counth1
        mov  a,counth1
        jz   dec_2
        mov  counth0,#0c8h
        sjmp settime
 dec_2: dec  counth2
        mov  a,counth2
        jz   dec_3
        mov  counth0,#0c8h
        mov  counth1,#0c8h
        sjmp settime
 dec_3: dec  counth3
```

```
        mov  a,counth3
        jz   h_ok
        mov  counth0,#0c8h
        mov  counth1,#0c8h
        mov  counth2,#0fh
        sjmp settime
 h_ok:  setb flgh3

;count in autotime mode
settime:
        jnb  nowcount,threemin
 dec_0t: dec  countt0
        mov  a,countt0
        jz   dec_1t
        sjmp threemin
 dec_1t: dec  countt1
        mov  a,countt1
        jz   dec_2t
        mov  countt0,#0c8h
        sjmp threemin
 dec_2t: dec  countt2
        mov  a,countt2
        jz   dec_3t
        mov  countt0,#0c8h
        mov  countt1,#0c8h
        sjmp threemin
 dec_3t: dec  countt3
        dec  xcount     ;initial xcount =10
        mov  a,xcount
        cjne a,#00h,yet  ;means count finished for 1 hour
        setb sbit
    yet: mov  a,countt3
        jz   t_ok
        mov  countt0,#0c8h
        mov  countt1,#0c8h
        mov  countt2,#0fh
        sjmp threemin
  t_ok:  setb flgt3
```

```
;count 3 minuts
threemin:
      jnb  nowcounm,brnk
dec_0m: dec  countm0
      mov  a,countm0
      jz   dec_1m
      sjmp brnk
dec_1m: dec  countm1
      mov  a,countm1
      jz   dec_2m
      mov  countm0,#0c8h
      sjmp brnk
dec_2m: dec  countm2
      mov  a,countm2
      jz   m_ok
      mov  countm0,#0c8h
      mov  countm1,#032h
      sjmp brnk
 m_ok:  setb flgm2

 brnk:  pop acc
      ret

;interupt service routine for external interupt 0

caltemp:
      push 00h
      push 01h

      clr  tcon.4
      mov  r0,tl0
      mov  tempxx,r0  ;save value
      mov  r1,th0
      mov  tempyy,r1
      mov  tl0,#00h
      mov  th0,#00h

      pop  01h
      pop  00h
      reti
```

```
;interupt service routine of timer1

trigg:
        clr  tcon.6
        setb tcon.4
        clr  p1.3
        acall timing
        acall genpulse
        reti

genpulse:
;generate new pulse
        mov  th1,#0fbh
        mov  tl1,#50h
        setb p1.3
        setb tcon.6
        ret

conv_tmp:
;change from pulse width to temperature value

        clr  psw.7
        mov  dptr,#llimit
        mov  r2,#00h
        mov  count,#00h
  loop_ff: mov  a,r2
        movc a,@a+dptr
        mov  r0,tempxx
        mov  tempx,r0
        xch  a,tempx
        subb a,tempx
    nextbyt:inc  r2
        mov  a,r2
        movc a,@a+dptr
        mov  r1,tempyy
        mov  tempy,r1
        xch  a,tempy
        subb a,tempy
        jb   psw.7,find
        inc  r2
        inc  count
```

```
    cjne r2,#28h,loop_ff
    mov  count,#12h

find: mov  a,count
    subb a,#01h
    jz   hyst
    mov  a,count
    subb a,#03h
    jz   hyst
    mov  a,count
    subb a,#05h
    jz   hyst
    mov  a,count
    subb a,#07h
    jz   hyst
    mov  a,count
    subb a,#09h
    jz   hyst
    mov  a,count
    subb a,#0bh
    jz   hyst
    mov  a,count
    subb a,#0dh
    jz   hyst
    mov  a,count
    subb a,#0fh
    jz   hyst
    mov  a,count
    subb a,#11h
    jz   hyst

    mov  a,count
    add  a,#02h
    movc a,@a+pc
    sjmp get
    db  1eh      ;30
    db  1bh;
    db  1dh
    db  1bh;
    db  1ch
    db  1bh;
```

```
    db  1bh
    db  1bh;
    db  1ah
    db  1bh;
    db  19h
    db  1bh;
    db  18h
    db  1bh;
    db  17h
    db  1bh;
    db  16h
    db  1bh;
    db  15h   ;21
 get: add  a,#03h
    mov  temp,a
 hyst:
    ret

;table used in program

;seven segment patterns
lowbyte:db 0eh
     db 1eh
     db 2eh
     db 3eh
     db 4eh
     db 5eh
     db 6eh
     db 7eh
     db 8eh
     db 9eh
     db 0eh
     db 1eh
     db 2eh
     db 3eh
     db 4eh
     db 5eh
     db 6eh
     db 7eh
     db 8eh
     db 9eh
```

```
        db 0eh
        db 1eh
        db 2eh
        db 3eh
        db 4eh
        db 5eh
        db 6eh
        db 7eh
        db 8eh
        db 9eh
        db 0eh

higbyte:db 0fh
        db 0fh
        db 0fh
        db 0fh
        db 0fh
        db 0fh
        db 0fh
        db 0fh
        db 0fh
        db 0fh
        db 1fh
        db 1fh
        db 1fh
        db 1fh
        db 1fh
        db 1fh
        db 1fh
        db 1fh
        db 1fh
        db 1fh
        db 2fh
        db 2fh
        db 2fh
        db 2fh
        db 2fh
        db 2fh
        db 2fh
        db 2fh
        db 2fh
        db 2fh
```

```
        db 2fh
        db 3fh

;table for finding temperature
limit:
        db 27h
        db 02h
        db 3bh
        db 02h
        db 5dh
        db 02h
        db 71h
        db 02h
        db 92h
        db 02h
        db 0bbh
        db 02h
        db 0c9h
        db 02h
        db 0ddh
        db 02h
        db 0ffh
        db 02h
        db 13h
        db 03h
        db 35h
        db 03h
        db 49h
        db 03h
        db 6bh
        db 03h
        db 7fh
        db 03h
        db 0a1h
        db 03h
        db 0b5h
        db 03h
        db 0d7h
        db 03h
        db 0ech
        db 03h
```

```
        db 0dh
        db 04h
        db 21h
        db 04h

flimit: nop
        end
```