การแสดงภาพด้วย LED 2 สีต่างระดับ

LED ARRAY DISPLAY

โดย

นางสาวชินศรี	สุริยเดชสกุล	รหัส 33100092
นางสาวนารีนาถ	รักสุนทร	รหัส 33100165
นางสาวนิภา	พจนมงคลกิจ	รหัส 33100174

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

สาขาวิศวกรรมอิเล็คทรอนิกส์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2536

ปริญญานิพนธ์ปีการศึกษา 2536

ภาควิชา อิเล็คทรอนิคส์

คณะ วิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การแสดงภาพด้วย LED สองสีต่างระดับ

ผู้จัดทำ

1. นางสาว ชินศรี สุริยเดชสกุล รหัส 33100092
2. นางสาว นารีนาถ รักสุนทร รหัส 33100165
3. นางสาว นิภา พจนมงคลกิจ รหัส 33100174

อาจารย์ที่ปรึกษา

(อาจารย์ ประภากร สุวรรณะ)

การแสดงภาพด้วย LED 2 สีต่างระดับ

LED ARRAY DISPLAY

โดย นางสาว ชินศรี สุริยเดชสกุล รหัส 33100092

นางสาว นารีนาถ รักสุนทร รหัส 33100165

นางสาว นิภา พจนมงคลกิจ รหัส 33100174

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ เป็นการนำเสนอเนื้อหาของโครงงานการแสดงภาพด้วย LED 2 สีที่สามารถควบคุมระดับแสงสีได้ทั้งหมด 16 ระดับ โดยอาศัยไมโครคอมพิวเตอร์มาส่งข้อมูลควบคุมระบบทั้งหมดทางพอร์ทอนุกรม มีการนำไมโครคอนโทรลเลอร์ในตระกูล MCS-51 มาใช้เพื่อการรับคำสั่งและข้อมูลจากไมโครคอมพิวเตอร์ รวมไปถึงการควบคุมอาร์ดแวร์ของระบบให้สามารถแสดงภาพที่ออกแบบไว้ทางแผง LED ที่ต่อแบบดอทเมตริกซ์ขนาด 32*32 จุดในรูปแบบดังต้องการได้

ABSTRACT

This thesis is the submission of 2-color-led presentation. The 2-color-led can control color levels up to 16 levels by using microcomputer to send datas which control main system on serial port.MCS-51,one of the microcontroller's family, is used to receive instructions and datas from microcomputer in addition to control system's hardware in order to present datas that are designed on led dot matrix with 32*32 dots in the format of data presentation.

สารบัญ

บทที่ 1
บทนำ

การสื่อสารนับได้ว่ามีความสำคัญ และจำเป็นต่อการดำเนินชีวิตในสังคมของมนุษย์ในปัจจุบันนี้มาก ได้มีการนำเอาเทคโนโลยีความก้าวหน้าทางด้านอิเล็กทรอนิกส์มาประยุกต์ให้เกิดประโยชน์มากมาย ด้วยเทคโนโลยีต่าง ๆ ที่พัฒนาไปเรื่อย ๆ เหล่านี้ได้ก่อให้เกิดการสื่อสารหลากหลายรูปแบบแตกต่างกันออกไป

สำหรับการสื่อสารที่เกี่ยวข้องกับการแสดงสื่อภาพและข้อความนั้น ในรูปแบบที่เป็นการแสดงภาพและข้อความโดยใช้แผง LED เป็นสื่อชนิดหนึ่งที่สร้างและดึงดูดความสนใจจากผู้พบเห็นได้เป็นอย่างมาก เนื่องจากการแสดงภาพและข้อความด้วยวิธีนี้สามารถทำการออกแบบควบคุมและสื่อสารต่าง ๆ ได้ตามความต้องการของผู้ออกแบบ ด้วยเหตุนี้จึงทำให้เรื่องของการแสดงภาพและข้อความด้วย LED เป็นที่น่าสนใจศึกษาอย่างยิ่ง

ผู้จัดทำได้ทำการศึกษา ออกแบบ และจัดทำโครงงานเรื่อง "การแสดงภาพด้วย LED 2 สีต่างระดับ" นี้ขึ้นมา มีจุดประสงค์เพื่อให้สามารถสื่อภาพและข้อความที่ต้องการออกทางแผง LED 2 สี ขนาด 32*32 จุด โดยควบคุมการแสดงภาพนี้ได้โดยตรงจากไมโครคอมพิวเตอร์ และออกแบบให้แสงสีที่แสดงออกทาง LED นี้มีหลายระดับ (16 ระดับ) เพื่อให้การสื่อภาพและข้อความด้วยวิธีนี้น่าสนใจยิ่งขึ้น

เนื้อหาของวิทยานิพนธ์ฉบับนี้จะประกอบไปด้วยรายละเอียดต่าง ๆ อันเกี่ยวข้องกับโครงงาน ทั้งในส่วนของฮาร์ดแวร์และซอฟท์แวร์ อันได้แก่ โครงสร้างของระบบทั้งหมด หน้าที่และหลักการทำงานขององค์ประกอบส่วนต่าง ๆ ส่วนของโปรแกรมที่ใช้ควบคุมรายละเอียดวงจรแต่ละส่วนของระบบ รวมถึงข้อมูลประกอบโครงงานที่มีความสำคัญอีกด้วย

บทที่ 2

หลักการและทฤษฎีของไมโครคอนโทรลเลอร์ตระกูล 51

ไมโครคอนโทรลเลอร์แบบชิพเดี่ยว (Single Chip Microcontroller) คือ ไมโครคอมพิวเตอร์แบบที่มีขนาดเล็กโดยบรรจุไว้ในแผงวงจรรวม (Integrated Circuit) เพียงชิพเดี่ยวเหมาะสำหรับงานควบคุมอุปกรณ์อื่น ๆ แบบอัตโนมัติ เพราะผู้ใช้สามารถเขียนโปรแกรมควบคุมการทำงานได้ตามต้องการ ไมโครคอนโทรลเลอร์แบบชิพ-เดี่ยวตระกูล 51 หรือ MCS51 ซึ่งได้แก่ เบอร์ 8051 และ 8052 ซึ่งมีโครงสร้างและชุดคำสั่งแตกต่างกันเพียงเล็กน้อยดังตารางในรูปที่ 1

Device	ROMless Version	EPROM Version	ROM Bytes	RAM Bytes	8-Bit I/O Ports	16-Bit Timer/ Counters	Programmable Counter Array (PCA)	UART	Serial Expansion 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	8751H 8751BH	4K	128	4	2		✓					6/5	
8052AH	8032AH	8752BH	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		10/11	✓
—	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	✓

รูปที่ 1 ตารางของไมโครคอนโทรลเลอร์แบบชิพเดี่ยวในตระกูล 51

ลักษณะเด่นของ MCS-51 มีดังนี้

-สามารถนำเอาข้อมูลมา AND,OR หรือทำ Complement ทั้งแบบทีละ 8 บิต และ 1 บิต

-สามารถใช้กับหน่วยความจำสำหรับโปรแกรม (Program Memory) ซึ่งเป็นหน่วยความจำที่ใช้สำหรับเก็บชุดคำสั่งที่จะให้ MCS-51 ทำงานได้ถึง 64 กิโลไบท์

-สามารถต่อกับหน่วยความจำสำหรับข้อมูล (Data Memory) ซึ่งเป็นหน่วยความจำ

สำหรับเก็บข้อมูลในระหว่างการทำงานของโปรแกรมได้สูงสุด 64 กิโลไบท์

-ใน 8051 และ 8751 มีหน่วยความจำสำหรับโปรแกรมจำนวน 4 กิโลไบท์ (ใน 8052 และ 8752 มีหน่วยความจำสำหรับโปรแกรมจำนวน 8 กิโลไบท์) อยู่ภายในวงจรรวมทำให้ไม่ต้องต่อหน่วยความจำสำหรับโปรแกรมอยู่ภายนอก

-มีพอร์ทแบบขนาน (Parallel Port) สำหรับข้อมูลเข้าและออกจำนวน 32 บิต ที่ข้อมูลแต่ละบิตเป็นอิสระต่อกัน

-มีวงจร Timer/Counter ขนาด 16 บิต 2 ชุด (8052 มี 3 ชุด) ที่ทำงานในโหมดต่าง ๆ ได้ถึง 4 โมด

-มี Universal Asynchonous Receiver Transmitter (UART) สำหรับรับ-ส่งข้อมูลอนุกรม (Serial) แบบ Full duplex ที่สามารถเลือกรูปแบบการรับ-ส่งข้อมูลได้ 4 แบบ

-มีแหล่งกำเนิดสัญญาณขอขัดจังหวะการทำงานของโปรแกรม (Interrupt Request Signal) 6 แหล่ง ซึ่งสามารถกระโดดไปทำงานตอบสนองการขัดจังหวะ (Interrupt Service Routine) ได้ต่าง ๆ กัน 5 ตำแหน่ง

-สามารถเลือกการทำงานให้อยู่ในโหมดของ Idle และ Power Down ซึ่งจะประหยัดการใช้กำลังไฟฟ้าในการทำงาน

โครงสร้างของ 8051

โครงสร้างในรูปที่ 2 เป็นโครงสร้างหลักของ 8051 ประกอบด้วย 3 ส่วนใหญ่ ๆ ได้แก่

ส่วนที่ 1 คือ ตัวประมวลผล หรือ CPU (Central Processing Unit) ส่วนนี้จะมีวงจรที่ทำหน้าที่สร้างสัญญาณควบคุมในการติดต่อกับส่วนอื่น ๆ เรียกว่า วงจรควบคุม (Control Unit) สัญญาณที่สร้างจากวงจรควบคุมได้แก่ สัญญาณสำหรับการติดต่อกับหน่วยความจำ, อุปกรณ์รับข้อมูลเข้า หรือส่งข้อมูลออกจากตัว 8051 ซึ่งส่วนควบคุมการขัดจังหวะ (Interrupt Control) และส่วนควบคุมบัส (Bus Control) ก็เป็นส่วนหนึ่งของวงจรควบคุมด้วย การสร้างสัญญาณควบคุมจากส่วนตัวประมวลผลนี้จะทำการสร้างสัญญาณโดยการ

ถอดรหัสจากคำสั่ง (Instruction) ตามที่มีการกำหนดไว้ และสัญญาณที่สร้างขึ้นมาจะอ้างอิงกับสัญญาณนาฬิกาที่สร้างจากวงจรออสซิลเลเตอร์

ในตัวประเมินผลนี้ยังประกอบด้วยส่วนย่อยอีกส่วนที่เรียกว่า ส่วนประมวลผล (Arithmetic Logic Unit) ส่วนนี้จะทำหน้าที่ประมวลผลข้อมูลเช่น การบวก,ลบ,คูณ หรือหารข้อมูลแล้วนำผลลัพธ์ไปเก็บไว้ในรีจิสเตอร์หรือหน่วยความจำที่ต้องการ

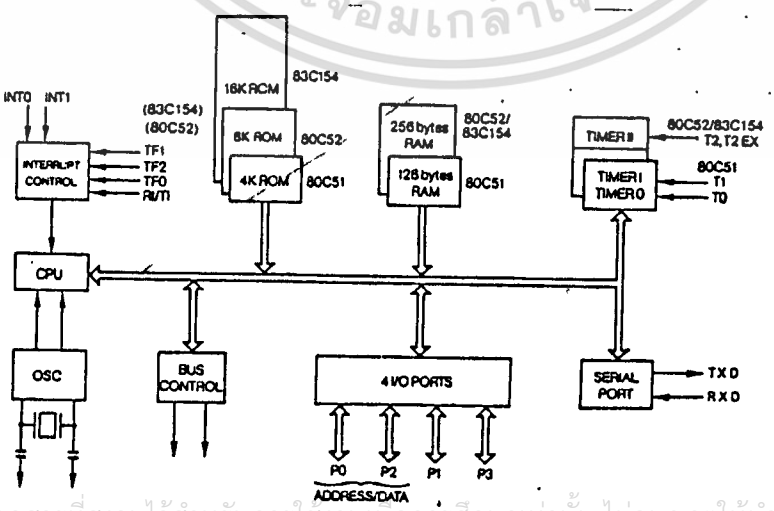
ส่วนที่ 2 คือ หน่วยความจำ (Memory) การติดต่อกับหน่วยความจำต้องมีสัญญาณ 3 กลุ่มคือ

1.แอดเดรสหรือค่าตำแหน่งที่ต้องการติดต่อกับหน่วยความจำ ใน 8051 จะติดต่อกับหน่วยความจำประเภท หน่วยความจำโปรแกรม (Program Memory) หรือหน่วยความจำข้อมูล (Data Memory) ได้สูงสุดชนิดละ 65536 ตำแหน่ง

2.ข้อมูลที่จะอ่านหรือเขียนกับหน่วยความจำที่ตำแหน่งในข้อที่ 1

3.สัญญาณควบคุมที่จะส่งไปยังหน่วยความจำ เพื่อบอกกับหน่วยความจำว่าต้องการอ่าน หรือเขียนข้อมูล

ส่วนที่ 3 อุปกรณ์อินพุทและเอาท์พุท (Input/Output Device) เป็นส่วนที่จะใช้ส่งข้อมูลเข้าหรือออกจาก 8051 ทำให้ 8051 ติดต่อกับภายนอกได้ ดังในรูปที่ 2 อุปกรณ์อินพุทและเอาท์พุทได้แก่ 4 I/O Port,Timer 0,Timer 1,Serial Port

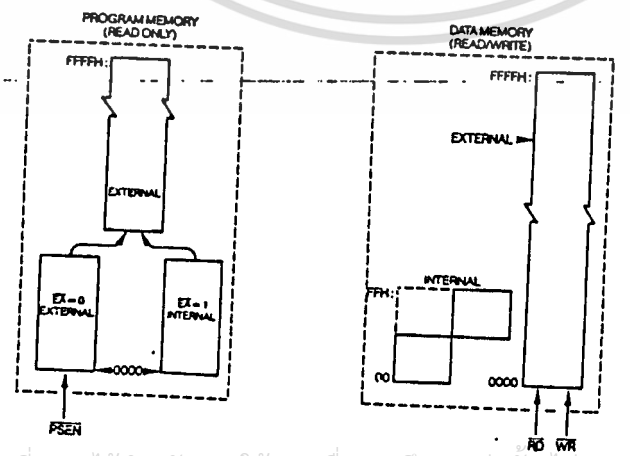


รูปที่ 2 โครงสร้างภายในของ 8051

การจัดหน่วยความจำของ 8051

หน่วยความจำของ 8051 แบ่งออกได้เป็น 2 แบบตามลักษณะของการใช้งานคือ

1. หน่วยความจำโปรแกรม เป็นหน่วยความจำที่ใช้เก็บคำสั่งในรูปรหัสภาษาเครื่อง (Machine Language) ซึ่งต้องการให้ 8051 ทำงานเมื่อ 8051 ทำงานก็จะอ่านข้อมูลที่เก็บในหน่วยความจำประเภทนี้เข้าไปถอดรหัสแล้วสร้างสัญญาณควบคุมส่วนอื่น ๆ ตามการทำงานของแต่ละคำสั่งนั้น หน่วยความจำแบบนี้จะต้องเป็น แบบ Read Only Memory (ROM) ในระหว่างการทำงานของ 8051 ผู้ใช้จะไม่สามารถใช้คำสั่งทำการเขียนข้อมูลลงในหน่วยความจำแบบนี้ได้ จำนวนตำแหน่งสูงสุดของหน่วยความจำที่จะใช้งานได้คือ 65536 ตำแหน่งมีค่าตั้งแต่ 0000H ถึง 0FFFH ซึ่งหน่วยความจำตั้งแต่ตำแหน่ง 0000H ถึง 0FFFH (4 กิโลไบท์) นั้นผู้ใช้จะเลือกได้ว่าเป็นตำแหน่งของ ROM ที่อยู่ภายในหรือภายนอก 8051 ไมโครคอนโทรลเลอร์บางรุ่น เช่น 8052 จะมีขนาดของ ROM สูงนี้ได้ถึง 8 กิโลไบท์ ตำแหน่ง 0000H ถึง 1FFFH) ถ้าต้องการให้ 8051 ทำงานตามคำสั่งที่เขียนไว้ใน ROM ภายใน 8051 ก็ให้ป้อนสัญญาณสภาวะลอจิก High(1) เข้าที่ขา EA ของ 8051 แต่ถ้าต้องการให้ทำงานในโปรแกรมที่เก็บไว้ใน ROM ภายนอก 8051 ก็ให้ต่อลอจิก "0" เข้าที่ขา EA ของ 8051 ส่วนหน่วยความจำที่ตำแหน่ง 1FFFH ถึง FFFFH จะถืออยู่ภายนอก 8051 .สมมติดังแสดงในแผนภูมิหน่วยความจำ (Memory Map) ในรูปที่ 3



รูปที่ 3 แผนภูมิหน่วยความจำของ 8051

ไมโครคอนโทรลเลอร์เบอร์ 8031, 8051 และ 8751 นั้น โดยโครงสร้าง และรหัสคำสั่งจะเหมือนกันทุกประการแตกต่างกันที่

-8031 จะไม่มี ROM ขนาด 4 กิโลไบท์อยู่ภายใน ผู้ใช้จะต้องเลือกการใช้งานหน่วยความจำภายนอก ที่อยู่ภายนอกวงจรรวมทั้งหมด 64 กิโลไบท์

-8051 จะมี ROM ขนาด 4 กิโลไบท์อยู่ภายใน ถ้าต้องการเก็บคำสั่งควบคุมการทำงานไว้ในหน่วยความจำส่วนนี้ จะต้องส่งโปรแกรมคำสั่งไปให้โรงงานผู้ผลิตทำการเขียนใส่ใน ROM ให้ตั้งแต่ในขั้นตอนของการผลิตวงจรรวม ผู้ใช้ไม่สามารถแก้ไขโปรแกรมได้เอง ถ้าจะนำมาใช้งานโดยเก็บโปรแกรมไว้ในหน่วยความจำช่วง 4 กิโลไบท์แรกอยู่ภายนอกก็สามารถทำได้โดยการต่อ ROM ไว้ภายนอก แล้วต่อขา EA ของ 8051 ไว้กับสัญญาณที่มีสภาวะลอจิกเป็น 0

-8751 จะมีหน่วยความจำขนาด 4 กิโลไบท์เป็นแบบ EPROM (Erasable Program Read Only Memory) อยู่ภายในวงจรรวมเอาไว้ ใช้เก็บโปรแกรมคำสั่งที่จะให้ 8751 ทำงาน ผู้ใช้สามารถเขียนคำสั่งลงไปใน EPROM ได้เองโดยใช้เครื่องมือที่เรียกว่า เครื่องโปรแกรม EPROM (EPROM Programmer) และผู้ใช้สามารถแก้ไขโปรแกรมที่อยู่ใน EPROM ได้โดยการล้างข้อมูลในทุกตำแหน่งของ EPROM ออกด้วยการฉายแสงอุลตร้าไวโอเลต (Ultraviolet) ผ่านกระจกใสบนวงจรรวมเข้าไปยังวงจรภายใน ตามเวลาที่กำหนดในคู่มือเฉพาะ (Data sheet) ของ 8751 จากนั้นก็ใช้เครื่องโปรแกรม EPROM เขียนโปรแกรมลงไปใหม่

2. หน่วยความจำข้อมูล แบ่งเป็นภายนอกและภายใน หน่วยความจำข้อมูลภายในจะแบ่งตามลักษณะงานดังนี้ คือ 128 ไบต์ ของบริเวณตำแหน่งล่างในเนื้อที่แรมภายใน ที่บริเวณตำแหน่งบนในเนื้อที่แรมภายในอีก 128 ไบต์ ซึ่งในเนื้อที่หน่วยความจำในส่วนนี้จะมีเฉพาะในเบอร์ 8052 เท่านั้น และอีก 128 ไบต์จะใช้เป็นที่เก็บรีจิสเตอร์ฟังก์ชันพิเศษ (Special Function Register;SFR) โดยมีการจัดแบ่งดังนี้

รีจิสเตอร์	คำอธิบาย	ตำแหน่ง
*ACC	แอคคิวมูเลเตอร์	0E0H
*B	B รีจิสเตอร์	0F0H
*PSW	รีจิสเตอร์แสดงสถานะโปรแกรม	0D0H
SP	ตัวชี้สแตก	081H
DPTR	ตัวชี้ข้อมูลไบต์สูง DPH	083H
	ตัวชี้ข้อมูลไบต์ต่ำ DPL	082H
*P1	พอร์ท 0	080H
*P1	พอร์ท 1	090H
*P2	พอร์ท 2	0A0H
*P3	พอร์ต 3	0B0H
*IP	รีจิสเตอร์ควบคุมลำดับความสำคัญของการอินเตอร์รัพต์	0B8H
*IE	รีจิสเตอร์การอินเตอร์รัพต์อีนาเบิล	0A8H
TMOD	รีจิสเตอร์ควบคุมโหมดตั้งเวลาและวงจรนับ	089H
#T2CON	รีจิสเตอร์ควบคุมวงจรตั้งเวลาและวงจรนับ 2	088H
TCON	รีจิสเตอร์ควบคุมวงจรตั้งเวลาและวงจรนับ	0C8H
TH_0	รีจิสเตอร์ตั้งเวลาและตัวนับ 0 ไบต์สูง	08CH
TL_0	รีจิสเตอร์ตั้งเวลาและตัวนับ 0 ไบต์ต่ำ	08AH
TH_1	รีจิสเตอร์ตั้งเวลาและตัวนับ 1 ไบต์สูง	08DH
TL_1	รีจิสเตอร์ตั้งเวลาและตัวนับ 1 ไบต์ต่ำ	08BH
#TH_2	รีจิสเตอร์ตั้งเวลาและตัวนับ 2 ไบต์สูง	0CDH
#TL_2	รีจิสเตอร์ตั้งเวลาและตัวนับ 2 ไบต์ต่ำ	0CCH
#RLDH	รีจิสเตอร์ตั้งเวลาและตัวนับ 2 ประจุใหม่อัตโนมัติไบต์สูง	0CBH
#RLDL	รีจิสเตอร์ตั้งเวลาและตัวนับ 2 ประจุใหม่อัตโนมัติไบต์ต่ำ	0CAH

รีจิสเตอร์	คำอธิบาย	ตำแหน่ง
*SCON	รีจิสเตอร์ควบคุมการส่งข้อมูลอนุกรม	098H
SBUF	รีจิสเตอร์ควบคุมการส่งแบบอนุกรม	099H
PCON	รีจิสเตอร์ควบคุมพลังงาน	097H

เครื่องหมาย * หน้าตัวรีจิสเตอร์ หมายถึง รีจิสเตอร์ตัวนั้นสามารที่จะเข้าถึงข้อมูลได้ทั้งระดับไบต์และบิต

หมายถึง รีจิสเตอร์ตัวนั้นจะมีเฉพาะในเบอร์ 8032 และ 8052 เท่านั้น

หน่วยความจำสำหรับข้อมูลภายใน 8051 ช่วง 00H ถึง 07FH สามารถแบ่งออกได้เป็น 3 กลุ่มคือ

1. รีจิสเตอร์ bank 0.3 อยู่ในหน่วยความจำช่วงตำแหน่งที่ 00H ถึง 1FH หน่วยความจำนี้จะแบ่งออกเป็น 4 ชุด ชุดละ 8 ไบท์ แต่ละชุดเรียกว่า bank แต่ละไบท์ใน 1 BANK จะมีชื่อรีจิสเตอร์ว่า R0, R1, R2, R3, R4, R5, R6 และ R7 รีจิสเตอร์เหล่านี้รีจิสเตอร์เหล่านี้จะเป็นชื่อซ้ำกันในทุก BANK การใช้งานจิงต้องเรียกใช้งานทีละ BANK เท่านั้น โดยการกำหนดในรีจิสเตอร์ PSW ที่จะได้กล่าวถึงต่อไป เมื่อมีการรีเซ็ทการทำงานของ 8051 จะเริ่มการใช้งานรีจิสเตอร์ R0 ถึง R7 ที่ BANK 0 ซึ่งรีจิสเตอร์ R0 ถึง R7 ในแต่ละ BANK นั้นจะอ้างอิงในหน่วยความจำสำหรับข้อมูลภายใน 8051 ดังในตารางที่ 1

ตัวอย่าง เมื่อกำลังมีการใช้งานในหน่วยความจำ BANK 1 และมีการอ้างอิงถึงรีจิสเตอร์ R7 เช่นคำสั่ง

MOV A,R7 (รหัสภาษาเครื่องคือ FF)

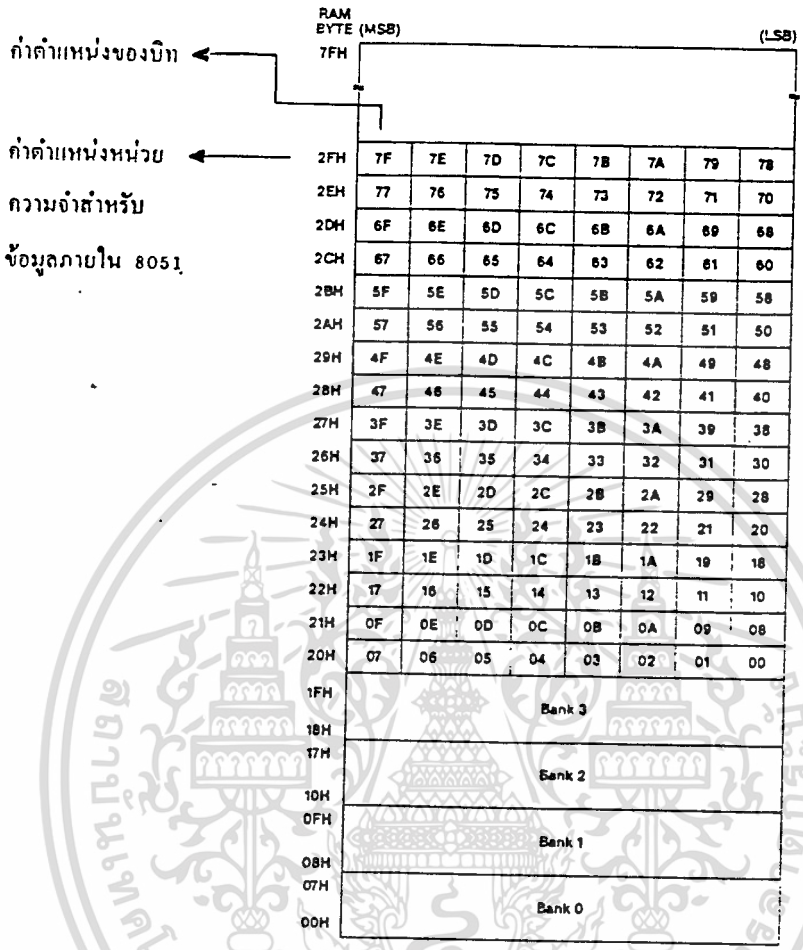
การทำงานของคำสั่งนี้คือการเอาข้อมูลในตำแหน่ง FH ของหน่วยความจำภายใน 8051 ไปไว้ยังริจิสเตอร์ A นั่นเอง

2. Bit Address Area เป็นหน่วยความจำในช่วงตำแหน่ง 20H ถึง 2FH หน่วยความจำแต่ละบิท ในช่วงของหน่วยความจำดังกล่าวจะสามารถตรวจสอบหรือตั้งค่าเป็น 1 หรือ 0 ได้ โดยการโปรแกรมภาษาเครื่อง แต่ละบิทของข้อมูลในหน่วยความจำช่วงนี้จะมีค่าของตำแหน่งดังในรูปที่ 4

3. Scratched Pod Area เป็นช่วงของหน่วยความจำตำแหน่ง 30H ถึง 7FH หน่วยความจำช่วงนี้จะใช้สำหรับเก็บข้อมูลทั่วไป ถ้าริจิสเตอร์ Stack pointer ชี้มายังหน่วยความจำช่วงนี้จะต้องระวังไม่ให้เกิดการเขียนทับของข้อมูลอันจะทำให้การทำงานของโปรแกรมผิดพลาดได้

ริจิสเตอร์	ตำแหน่งหน่วยความจำ			
	BANK 0	BANK 1	BANK 2	BANK 3
R0	0	8	10	18
R1	1	9	11	19
R2	2	A	12	1A
R3	3	B	13	1B
R4	4	C	14	1C
R5	5	D	15	1D
R6	6	E	16	1E
R7	7	F	17	1F

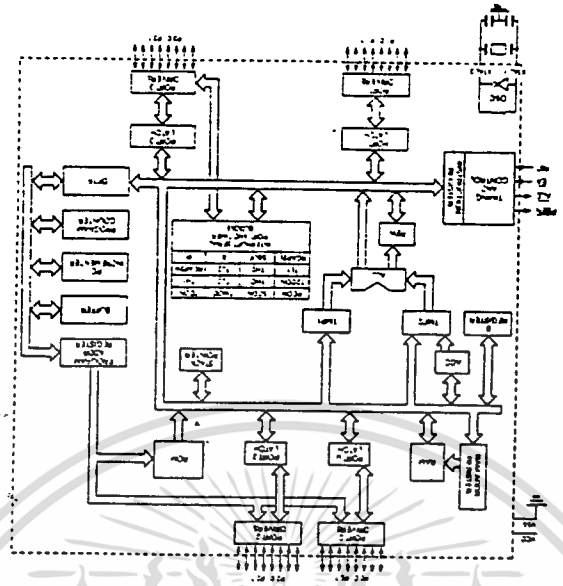
ตารางที่ 1



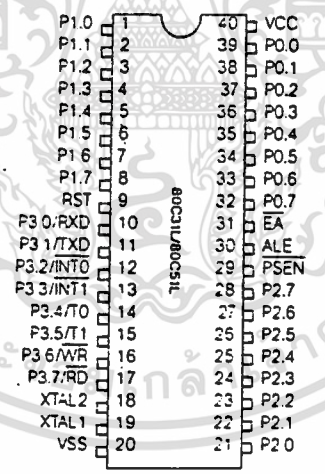
รูปที่ 4 ค่าตำแหน่งของแต่ละบิต

สถาปัตยกรรมของ 8051

ในรูปที่ 5 เป็นสถาปัตยกรรมภายในของ 8051 โดยชิงเกิลชิปแต่ละตัวของตระกูลนี้จะประกอบด้วยหน่วยศูนย์กลางประมวลผล หน่วยความจำสองชนิดนี้คือ แบบแรม กับรอมหรืออีพรอม(Eprom) พอร์ตเอาท์พุต อินพุท โหมดรีจิสเตอร์ สถานะและข้อมูล ส่วนวงจรตรรกในการแรนดอม (Random) ที่จำเป็นสำหรับตัวแปรของฟังก์ชันการต่อพ่วงส่วนต่าง ๆ ที่กล่าวมานี้จะติดต่อกันด้วยบัสข้อมูลขนาด 8 บิต และจะมีบัฟเฟอร์สำหรับการติดต่อข้อมูลภายนอกผ่านไอโอ เมื่อต้องการขยายหน่วยความจำหรือพอร์ตไอโอ



รูปที่ 5 สถาปัตยกรรมภายในของ 8051



รูปที่ 6 ไดอะแกรมขาของ 8051 แบบ DIP

V_{cc} (ขา 40) เป็นขาที่ต้องป้อนไฟเลี้ยง +5 โวลท์เข้าไปเพื่อให้วงจรรวมทำงานได้

V_{ss} (ขา 20) เป็นขาที่ต้องต่อกับกราวด์ (Ground) ของแหล่งจ่ายไฟ การต่ออุปกรณ์ทั้งหมดจะต้องมีกราวด์ของอุปกรณ์ต่อเข้าด้วยกัน

Port 0 (p0.0-p0.7/AD_0-AD_7) เป็นพอร์ทขนานขนาด 8 บิต อยู่ที่ขา 39

ถึง 32 เริ่มจากบิต 0 ถึง 7 ตามลำดับ เป็นพอร์ตอินพุท-เอาท์พุท หากเขียนค่า "0" ไปที่พอร์ตนี้จะกลายเป็นพอร์ตอินพุทพอร์ท 0 จะใช้งานหลายอย่างดังนี้

1.ใช้สำหรับส่งค่าตำแหน่งหน่วยความจำภายนอกที่ต้องการติดต่อด้วย ตำแหน่งหน่วยความจำสูงสุดที่จะติดต่อได้ก็ คือ 64 กิโลไบท์ จึงมีค่าตำแหน่งหน่วยความจำ 16 บิทของเลขฐาน 2 ค่าตำแหน่งหน่วยความจำ 8 บิตล่างจะถูกส่งออกไปทางพอร์ท 0 และ 8 บิตบนจะส่งออกไปทางพอร์ท 2

2.ใช้รับ-ส่งข้อมูลกับหน่วยความจำข้อมูลหรือใช้รับข้อมูลจากหน่วยความจำโปรแกรม

3.ใช้รับ-ส่งข้อมูลทางพอร์ทโดยตรง ในกรณีที่ไม่มีการใช้หน่วยความจำโปรแกรมหรือหน่วยความจำภายนอก

Port 1 (P1.0-P1.7 : ขา 1-8) เป็นพอร์ทอินพุทเอาท์พุทขนาด 8 บิต พร้อมด้วยการพูลอัพภายใน สามารถขับโหลดทีทีแอลตระกูลแอลเอสได้ 4 ตัว หากเขียนค่า 1 มาที่พอร์ตนี้จะกำหนดให้เป็นพอร์ตอินพุท สำหรับเบอร์ 8052 ขา p1.0 และ p1.7 จะใช้งานเป็นอินพุทผ่านเข้าวงจรตั้งเวลาชุดที่สอง

Port 2 (P2.0 ถึง P2.7) เป็นพอร์ทขนานขนาด 8 บิท ลักษณะการทำงานจะทำงานเหมือนกับพอร์ท 0 แตกต่างกันที่ใน พอร์ท 2 นั้น ภาค driver จะใช้งานเพียง 2 ลักษณะคือ

-ใช้ส่งค่าตำแหน่งหน่วยความจำภายนอกที่ต้องการติดต่อ ค่าตำแหน่งนี้เป็น 8 บิตบนของของค่าตำแหน่ง

-ใช้เป็นพอร์ทรับและส่งข้อมูลกับภายนอก

Port 3 (P3.0-P3.7 : ขา 10-17) เป็นพอร์ทอินพุท-เอาท์พุทขนาด 8 บิต นอกจากจะใช้งานกับโหลดทีทีแอลได้แล้ว ยังสามารถทำหน้าที่พิเศษได้อีกดังข้างล่าง

-P3.0/RXD (Serial Input Port) เป็นขาที่ใช้รับข้อมูลแบบอนุกรม

-P3.1/TXD (Serial Output Port) เป็นขาที่ใช้ส่งข้อมูลแบบอนุกรม

-P3.2/INT0 (External Interrupt) ใช้รับสัญญาณขัดจังหวะจากภายนอก

-P3.3/INT1 (External Interrupt) ใช้รับสัญญาณขัดจังหวะจากภายนอก

-P3.4/T0 (Timer/Counter 0 External Input) ขารับสัญญาณเข้ายังวงจร Timer/Counter 0 ที่ทำหน้าที่นับจำนวนไซเคิลของสัญญาณ T0 นี้หรือสัญญาณนาฬิกาก็ได้

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

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

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

RST (ขา 9) ขารีเซ็ตนี้จะใช้รีเซ็ตการทำงานของ 8051 ที่ขา RST ภายใน 8051 จะมีตัวต้านทานต่อระหว่างขานี้กับกราวด์ (Ground) ถ้าป้อนสัญญาณที่มีสภาวะลอจิก "1" เข้าไปที่ขานี้จะเป็นการรีเซ็ตการทำงานของ 8051 ดังนั้นจึงสามารถต่อตัวเก็บประจุ (Capacitor) ภายนอกระหว่างขา RST กับไฟเลี้ยง +5โวลท์ เพื่อให้เกิดการรีเซ็ตเมื่อเริ่มป้อนไฟเลี้ยงให้กับ 8051 ซึ่งเรียกว่า Power on reset

ขา ALE/PROG คือขาแอดเดรสแลทช์อินาเบิล เมื่อส่งพัลส์ออกมาจะใช้ในการแลทช์ค่าชี้ตำแหน่งไบต์ต่ำจาก PROTO ในระหว่างการเข้าถึงข้อมูลภายใน

ขา PSEN (ขา 29) คือขา Program Storage Enable เป็นสโตรปใช้อ่านข้อมูลจากหน่วยความจำภายนอก

ขา EA/V_{pp} (ขา 31) เมื่อขานี้ชี้สถานะ "1" ชิพก็จะทำงานตามโปรแกรมที่มีอยู่ในหน่วยความจำภายใน หากสถานะ "0" จะเป็นการควบคุมให้ทำงานตามโปรแกรมในหน่วยความจำภายนอก

ขา XTAL 1 (ขา 19) ใช้เป็นอินพุทเข้าสู่วงจรออสซิลเลเตอร์

ขา XTAL 2 (ขา 18) ใช้เป็นเอาท์พุทออกจากออสซิลเลเตอร์

ชุดคำสั่ง 8051

ชุดคำสั่ง 8051 จะแบ่งออกได้เป็น 5 กลุ่ม ได้แก่

1. ชุดคำสั่งทางคณิตศาสตร์ (Arithmetic Instruction)

8051 มีคำสั่งการกระทำทางคณิตศาสตร์ที่มีทั้งการบวก, ลบ, คูณ และหาร ซึ่งคำสั่งสำหรับการกระทำทางคณิตศาสตร์มีดังตารางที่ 2

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
ADD A,<byte>	A=A+<byte>	X	X	X	X	1
ADDC A,<byte>	A=A+<byte>+C	X	X	X	X	1
SUBB A,<byte>	A=A−<byte>−C	X	X	X	X	1
INC A	A=A+1	Accumulator only				1
INC <byte>	<byte>=<byte>+1	X	X	X		1
INC DPTR	DPTR=DPTR+1	Data Pointer only				2
DEC A	A=A−1	Accumulator only				1
DEC <byte>	<byte>=<byte>−1	X	X	X		1
MUL AB	B:A=B x A	ACC and B only				4
DIV AB	A=Int [A/B] B=Mod [A/B]	ACC and B only				4
DA A	Decimal Adjust	Accumulator only				1

ตารางที่ 2 ชุดคำสั่งทางคณิตศาสตร์

หมายเหตุ ตั้งแต่ตารางที่ 2-9 จะใช้อธิบายคำสั่งในแต่ละกลุ่มซึ่งจะประกอบด้วย 4 คอลัมน์

คอลัมน์ที่ 1 คือ Mnemonic เป็นช่องที่จะบอกถึงรหัสคำสั่งช่วยจำ

คอลัมน์ที่ 2 คือ Operation เป็นการกระทำที่เกิดขึ้นตามรหัสคำสั่งช่วยจำที่อยู่ในแถวเดียวกัน

คอลัมน์ที่ 3 คือ Addressing mode คอลัมน์นี้จะแบ่งออกเป็น 4 คอลัมน์ย่อย ๆ คือ

Dir = Direct Addressing

Ind = Indirect Addressing

Reg = Register Addressing

Imm = Immediate Addressing

เครื่องหมาย x ในช่องนี้หมายความว่า รหัสคำสั่งช่วยจำมีเครื่องหมาย <byte> อยู่ในส่วน Operand สามารถอ้างอิงตำแหน่งของหน่วยความจำได้ด้วยวิธีดังกล่าว

คอลัมน์ที่ 4 จะบอก Execution Times (us) เป็นเวลาที่ใช้ในการทำงานคำสั่งนั้น 1 คำสั่งมีหน่วยเป็นไมโครวินาที (micro-second, 1ไมโครวินาทีมีค่าเท่ากับ 10^{-6} วินาที) 8051 จะใช้กับสัญญาณนาฬิกาได้สูงถึง 12 เมกกะเฮิรตซ์ (Megahertz) ในการทำงานแต่ละคำสั่งจะใช้จำนวนรอบเครื่อง (Machine Cycle) เท่ากับ 1, 2 หรือ 3

รอบเครื่อง 1 รอบเครื่องจะใช้เวลาเท่ากับ 12 ไซเคิลของสัญญาณนาฬิกา ดังนั้นจะสามารถทราบเวลาที่ใช้ในการทำงาน 1 รอบเครื่องเท่ากับ $12/f_{clk}$ วินาที

2. คำสั่งทางตรรกศาสตร์ (logic Instruction)

กลุ่มคำสั่งในชุดนี้จะทำงานเหมือนคำสั่งทางบูลีน มีชุดคำสั่งดังในตารางที่ 3

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
ANL A,<byte>	A=A.AND.<byte>	X	X	X	X	1
ANL <byte>,A	<byte>=<byte>.AND.A	X				1
ANL <byte>,#data	<byte>=<byte>.AND.# data	X				2
ORL A,<byte>	A=A.OR.<byte>	X	X	X	X	1
ORL <byte>,A	<byte>=<byte>.OR.A	X				1
ORL <byte>,# data	<byte>=<byte>.OR.# data	X				2
XRL A,<byte>	A=A.XOR.<byte>	X	X	X	X	1
XRL <byte>,A	<byte>=<byte>.XOR.A	X				1
XRL <byte>,# data	<byte>=<byte>.XOR.# data	X				2
CLR A	A=00H	Accumulator only				1
CPL A	A=.NOT.A	Accumulator only				1
RL A	Rotate ACC Left 1 bit	Accumulator only				1
RLC A	Rotate Left through Carry	Accumulator only				1

3. คำสั่งในการเคลื่อนย้ายข้อมูล (Data Transfer)

ในคำสั่งเหล่านี้จะมีการเคลื่อนย้ายข้อมูลจากหน่วยความจำตำแหน่งหนึ่งไปยังหน่วยความจำอีกตำแหน่งหนึ่ง รูปแบบของคำสั่งกลุ่มนี้คือ

OP-CODE, Destination, Source

OP-CODE จะเป็นส่วนที่บอกการทำงานทั้งหมดของคำสั่งนี้ว่า จะเป็นการเคลื่อนย้ายข้อมูลทางเดียว หรือแลกเปลี่ยนข้อมูล

Source เป็นตำแหน่งข้อมูลที่จะถูกเคลื่อนย้าย

Destination เป็นตำแหน่งของปลายทางที่จะใช้เก็บข้อมูลในการเคลื่อนย้ายข้อมูลแบบทางเดียวนั้น เมื่อสิ้นสุดการทำงานข้อมูลที่อยู่ในตำแหน่ง Source จะเปลี่ยนแปลง แต่ถ้าเป็นการแลกเปลี่ยนข้อมูลก็จะสลับข้อมูลระหว่าง Source กับ Destination คำสั่งเคลื่อนย้ายข้อมูลจะไม่มีผลเปลี่ยนแปลงต่อแฟลกใด ๆ ใน PSW คำสั่งในกลุ่มนี้มีดังในตารางที่ 4 , ตารางที่ 5 และตารางที่ 6

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
MOV A,<src>	A=<src>	X	X	X	X	1
MOV <dest>,A	<dest>=A	X	X	X		1
MOV <dest>,<src>	<dest>=<src>	X	X	X	X	2
MOV DPTR,# data 16	DPTR=16-bit immediate constant.				X	2
PUSH <src>	INC SP : MOV "@SP",<src>	X				2
POP <dest>	MOV <dest>,"@SP" : DEC SP	X				2
XCH A,<byte>	ACC and <byte> exchange data	X	X	X		1
XCHD A,@Ri	ACC and @ Ri exchange low nibbles		X			1

ตารางที่ 4 คำสั่งเคลื่อนย้ายข้อมูลภายใน 8051

Address Width	Mnemonic	Operation	Execution Time (μs)
8 bits	MOVX A, @Ri	Read external RAM @ Ri	2
8 bits	MOVX @Ri,A	Write external RAM @ Ri	2
16 bits	MOVX A, @DPTR	Read external RAM @ DPTR	2
16 bits	MOVX @DPTR,A	Write external RAM @ DPTR	2

ตารางที่ 5 คำสั่งเคลื่อย้ายข้อมูลภายนอก 8051

Mnemonic	Operation	Execution Time (μs)
MOVC A,@A+DPTR	Read Pgm Memory at (A+DPTR)	2
MOVC A,@A+PC	Read Pgm Memory at (A+PC)	2

ตารางที่ 6 คำสั่งอ่านข้อมูลจากตารางที่อยู่ในหน่วยความจำสำหรับโปรแกรม

4. คำสั่งบูลีน

หน่วยความจำสำหรับข้อมูลภายใน 8051 สามารถติดต่อ หรือมีการกระทำต่อข้อมูลทีละบิท คือแต่ละบิทของรีจิสเตอร์ SFR และในช่วงหน่วยความจำสำหรับข้อมูลตำแหน่ง

20H ถึง 2FH จำนวน 16 ไบท์ (มีจำนวนบิตที่สามารถติดต่อได้ถึง 128 บิต หรือ 128 ตำแหน่ง) คำสั่งดังกล่าวมีดังตารางที่ 7

Mnemonic	Operation	Execution Time (μs)
ANL C,bit	C = C AND bit	2
ANL C,/bit	C = C AND NOT bit	2
ORL C,bit	C = C OR bit	2
ORL C,bit	C = C OR NOT bit	2
MOV C,bit	C = bit	1
MOV bit,C	bit = C	2
CLR C	C = 0	1
CLR bit	bit = 0	1
SETB C	C = 1	1
SETB bit	bit = 1	1
CPL C	C = NOT C	1
CPL bit	bit = NOT bit	1
JC rel	Jump if C = 1	2
JNC rel	Jump if C = 0	2
JB bit,rel	Jump if bit = 1	2
JNB bit,rel	Jump if bit = 0	2
JBC bit,rel	Jump if bit = 1 ; CLR bit	2

ตารางที่ 7 คำสั่งบูลีน

5. ชุดคำสั่งกระโดดข้าม (Jump Instruction)

คำสั่งกระโดดข้ามคือ กลุ่มของคำสั่งที่ทำให้การทำงานของโปรแกรมข้ามไปยังตำแหน่งที่กำหนดโดยไม่ต้องทำตามลำดับของโปรแกรมเดิม คำสั่งกระโดดข้ามมีดังตารางที่ 8 และในตารางที่ 9

Mnemonic	Operation	Execution Time (μs)
JMP addr	Jump to addr	2
JMP @A + DPTR	Jump to A + DPTR	2
CALL addr	Call subroutine at addr	2
RET	Return from subroutine	2
RETI	Return from interrupt	2
NOP	No operation	1

ตารางที่ 8 คำสั่งกระโดดข้ามแบบไม่มีเงื่อนไข

Mnemonic	Operation	Addressing Modes				Execution Time (μs)
		Dir	Ind	Reg	Imm	
JZ rel	Jump if A=0	Accumulator only				2
JNZ rel	Jump if A=0	Accumulator only				2
DJNZ <byte>,rel	Decrement and jump if not zero	X		X		2
CJNE A,<byte>,rel	Jump if A=<byte>	X			X	2
CJNE <byte>, #data,rel	Jump if <byte>=#data		X	X		2

ตารางที่ 9 คำสั่งกระโดดข้ามแบบมีเงื่อนไข

แอดเดรสซิ่ง (Addressing)

มี 5 วิธีดังนี้

1. Direct Addressing เป็นการกำหนดตำแหน่งที่อยู่ของหน่วยความจำที่จะติดต่อเข้าไปใน Operand ของคำสั่งโดยตรง วิธีการนี้สามารถใช้กับการติดต่อหน่วยความจำสำหรับข้อมูลในตัวของ 8051 จำนวน 256 ตำแหน่งเท่านั้น

2. Indirect Addressing เป็นการกำหนดที่อยู่ของหน่วยความจำสำหรับข้อมูลภายใน 8051 โดยอ้อม วิธีการระบุตำแหน่งของหน่วยความจำที่ต้องการติดต่อวิธีนี้ จะใช้รีจิสเตอร์ตัวหนึ่งเป็นตัวชี้ (Pointer) ไปยังหน่วยความจำที่ต้องการ รีจิสเตอร์ที่ใช้เป็นตัวชี้ได้แก่ R0, R1, DPTR เป็นต้น ใน Operand ของชุดคำสั่ง 8051 ที่ติดต่อกับหน่วยความจำโดยอ้อมจะมีสัญลักษณ์ นำหน้ารีจิสเตอร์ที่เป็นตัวชี้

3. Register Instruction เป็นคำสั่งที่ใช้ติดต่อกับรีจิสเตอร์ R0 ถึง R7 ของรีจิสเตอร์ Bank ที่กำลังใช้งานอยู่ใน Operand จะมีชื่อจริงของรีจิสเตอร์ที่ต้องการใช้อยู่ เมื่อแปลเป็นภาษาเครื่องจะพบว่า ในภาษาเครื่องของคำสั่งติดต่อกับรีจิสเตอร์เหล่านี้จะมี 3 บิตที่เป็นตัวบอกรีจิสเตอร์ R0 ถึง R7 ดังนั้นเมื่อคำสั่งนั้นถูกอ่าน (Fetch) เข้าไปประมวลผล (Execute) ก็จะแยกเอาตัวชี้รีจิสเตอร์ที่ต้องการมาจากคำสั่งภาษาเครื่อง (OP-CODE) ที่อ่านเข้าไปนั่นเอง

4. Immediate Constant เป็นคำสั่งเกี่ยวกับค่าคงที่โดยตรง คำสั่งนี้จะมีการกำหนดค่าคงที่ในส่วน Operand

5. Index Addressing การกำหนดเลขที่อยู่โดยตรรชนี การอ้างอิงหน่วย-ความจำวิธีนี้ใช้ได้เฉพาะกับการติดต่อหน่วยความจำสำหรับโปรแกรมเท่านั้น ซึ่งวิธีการอ้างอิงหน่วยความจำแบบนี้จะใช้รีจิสเตอร์ DPTR หรือ Program Counter ขนาด 16 บิต บวกด้วยรีจิสเตอร์ A ขนาด 8 บิตแล้วนำผลลัพธ์ไปชี้ตำแหน่งหน่วยความจำสำหรับโปรแกรมเพื่ออ่านข้อมูลออกมา คำสั่งนี้มีประโยชน์ในการอ่านข้อมูลซึ่งเก็บไว้เป็นตาราง (Table) ในหน่วยความจำโปรแกรม

การเชื่อมแบบอนุกรม

พอร์ทอนุกรมเป็นแบบ Full Duplex สามารถที่จะส่งและรับพร้อมกันได้ โดยทำหน้าที่เป็นพัฟเฟอร์การรับ หมายถึง พอร์ทสามารถที่จะรับไบต์ที่สอง ก่อนที่ตัวแรกจะถูกรับไปจากรีจิสเตอร์ตัวรับ อย่างไรก็ตาม ไบต์ตัวแรกจะต้องถูกอ่านไปก่อนที่ช่วงเวลาการรับไบต์ตัวที่สองจะสิ้นสุด มิฉะนั้นไบต์ตัวแรก จะถูกซ้อนและสูญหายไปได้ ในพอร์ตอนุกรม รีจิสเตอร์ตัวรับ และส่ง จะเข้าถึงติดต่อกันด้วยรีจิสเตอร์ SBUF แม้ว่าทางโครงสร้างรีจิสเตอร์ทั้งสองจะแยกกันอยู่ก็ตาม

พอร์ทอนุกรมสามารถที่จะเลือกทำงานในโหมดต่าง ๆ ได้สี่โหมด

โหมด 0 ข้อมูลจะเป็นการเข้าและออกผ่าน RDX TXD ด้วยการเลื่อนสัญญาณนาฬิกาเอาท์พุต ข้อมูลจะเป็นลักษณะแปดบิตในการรับและส่งแต่ละครั้ง โดยที่ส่งค่า LSB ก่อนอัตราบ๊อดจะคงที่ที่ 1/12 ของความถี่ออสซิสเลเตอร์

โหมด 1 จะเป็นการส่งข้อมูลขนาด 10 บิต ผ่านออก TDX หรือรับเข้ามาผ่าน RDX โดยรูปแบบบิตจะประกอบด้วย หนึ่งบิตสตาร์ท เป็น "0" แปดบิตข้อมูลโดย LSB เป็นตัวแรกที่รับและส่งข้อมูลนี้ และอีกหนึ่งบิตสต๊อบมีค่า "1" การรับบิตสต๊อบ จะนำไปเก็บที่บิต RB8 ของ SFR รีจิสเตอร์ SCON อัตราบ๊อดแปรผันได้ตามการตั้งตัวจับเวลา

โหมด 2 เป็นการส่งข้อมูลขนาด 11 บิต ผ่านออกขา TXD หรือรับเข้ามาผ่านขา RXD ประกอบด้วยหนึ่งบิตสตาร์ท มีค่า "0" แปดบิตข้อมูลโดย LSB เป็นตัวแรกที่รับ และส่งข้อมูล บิตที่เก้าของข้อมูลสามารถที่จะโปรแกรมเลือกได้ และบิตสต๊อบค่า "1" อีกหนึ่งบิตในการส่งบิตที่เก้าที่อยู่ในบิต TB8 ของรีจิสเตอร์ SCON สามารถที่จะกำหนดเลือกเป็น "1" หรือ "0" ได้

SCON เป็น SFR ที่ใช้ในการกำหนดโหมดการทำงานของพอร์ตอนุกรม เช่นการกำหนดค่า RB8 จะเป็นการใช้ตัวรับ รับบิตที่เก้าด้วยหรือไม่ เป็นต้น

การใช้ SCON ในการควบคุมการทำงานของพอร์ตอนุกรม

SMO	SM1	SM2	REN	TB8	RB8	TI	RI

โดย SMO, SM1 เป็นตัวกำหนดใช้โหมดต่าง ๆ ของพอร์ตอนุกรมดังนี้

SMO	SM1	โหมด	ลักษณะการทำงาน	อัตราบ๊อด
0	0	0	เลื่อนรีจิสเตอร์	$f_{osc}/12$
0	1	1	8-UART	แปรผันได้ตามการเลือกตัวจับเวลา
1	0	2	9-UART	$f_{osc}/32$ หรือ $f_{osc}/64$
1	1	3	9-UART	แปรผัน

*UART : Universal Asynchronous Receiver / Tranceiver

SM2 ควบคุมอินาเบิล การใช้โพรเซสเซอร์หลายตัวในการสื่อสารซึ่งกันและกัน

ในโหมด 2 และ 3 ถ้า SM2 เซตเป็น 1 ดังนั้น RI จะต้องไม่แอ๊คทีฟ ถ้ามีการรับบิตที่เก้า ทำให้บิต RB8 นี้เป็น 0

ในโหมด 1 ถ้า SM2 เซตเป็น 1 ดังนั้น RI จะไม่แอ๊คทีฟถ้าสต๊อบบิตไม่ถูกรับ

ในโหมด 0 SM2 ควรมีค่าเท่ากับ 0

REN ตัวอินาเบิลอนุกรมการรับ เซตเป็น "1" ด้วยโปรแกรมในการเลือกอินาเบิลการรับ และเป็น "0" ด้วยโปรแกรม เมื่อให้เป็นดิสเอเบิ้ล

TB8 เป็นข้อมูลที่เก้า ซึ่งจะถูกส่งในโหมด 2 และ 3 ซึ่งจะให้เป็น "1" หรือ "0" ได้ด้วยการโปรแกรม

RB8 ในโหมด 2 และ 3 ข้อมูลบิตที่เก้าจะถูกรับไป
ในโหมด 1 ถ้า SM2 = 0 RB8 จะกลายเป็นสต๊อบบิตที่ถูกรับไป
ในโหมด 0 RB8 ไม่ใช้

TI เป็นแฟลกอินเตอร์รัพต์การส่ง เซตด้วยฮาร์ดแวร์คือสัญญาณปลายช่วงเวลาของบิตแปดในโหมด 0 หรือที่จุดเริ่มต้นของบิตสต๊อบ ในโหมดอื่น ในการส่งแบบอนุกรมของทุกโหมดจะต้องเคลียร์บิตนี้ด้วยโปรแกรมหลังการส่ง

RI เป็นแฟลกอินเตอร์รัพต์การรับ เซตด้วยฮาร์ดแวร์คือสัญญาณที่ปลายช่วงเวลาของบิตที่แปดในโหมด 0 หรือที่จุดครึ่งทางของบิตสต๊อบในโหมดอื่น ในการรับแบบอนุกรมยกเว้นกรณีใช้ SM2 จะต้องเคลียร์บิตด้วยโปรแกรมหลังการรับ

โหมด 3 เป็นการส่งข้อมูลขนาด 11 บิต ผ่านออกขา TXD หรือรับเข้ามาผ่านขา RXD ประกอบด้วยบิตสตาร์ทมีค่า 0 แปดบิตข้อมูลโดย LSB เป็นบิตแรกที่รับและส่ง ข้อมูลบิตที่เก้าของข้อมูลสามารถที่จะโปรแกรมเลือกได้ และบิตสต๊อบค่า 1 อีกหนึ่งบิต ในความเป็นจริง โหมด 3 จะคล้ายกับโหมด 2 ทุกประการ ยกเว้นอัตราบ๊อด โดยอัตราบ๊อดในโหมด 3 จะแปรผันได้ไปตามการโปรแกรมการเลือกตัวจับเวลา

ทั้งสี่โหมดนี้ การส่งข้อมูลจะถูก Initiated ด้วยคำสั่งใด ๆ ที่ใช้ตัวรีจิสเตอร์ SBUF เป็นรีจิสเตอร์ตัวรับข้อมูลจากซีพียู และในโหมด 0 การรับข้อมูลจะ Initiated ด้วยการใช้สถานะ RI = 0 และ REN = 1 ส่วนในโหมดอื่น การรับข้อมูลจะถูก Initiated ด้วยการรับบิตสตาร์ท เข้ามาตรวจสอบ ถ้า REN = 1

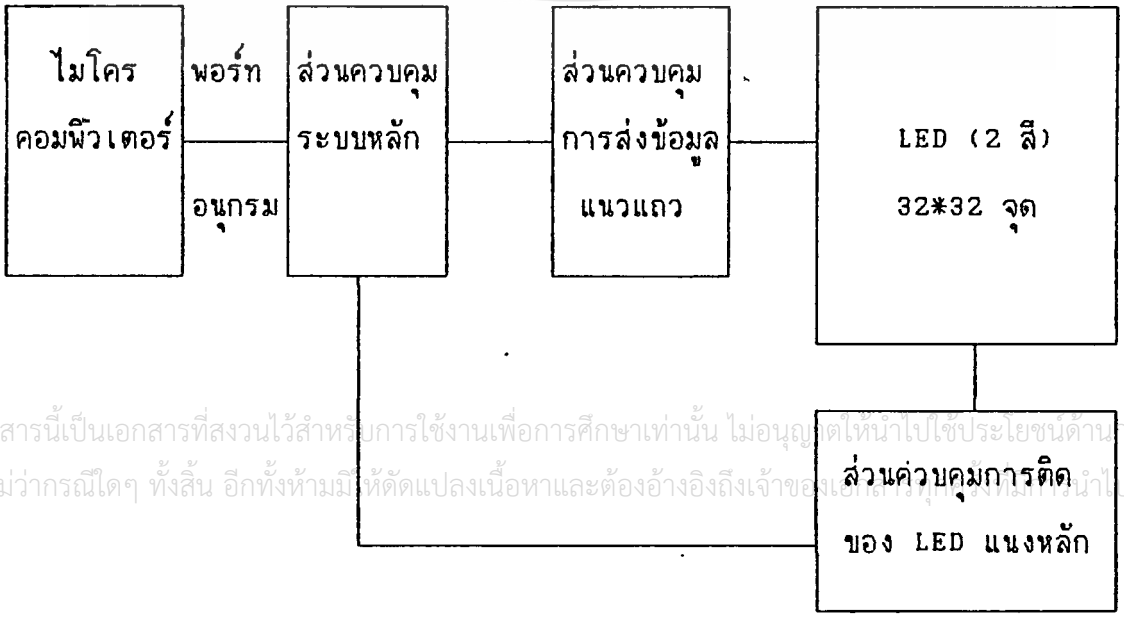
บทที่ 3

โครงสร้างของระบบ

โครงงาน "การแสดงภาพด้วย LED 2 สีต่างระดับ" นี้ อาศัยไมโครคอนโทรล-เลอร์เป็นองค์ประกอบหลักของระบบ ทำหน้าที่เป็นศูนย์กลางในการรับข้อมูลทั้งในส่วนคำสั่งและส่วนข้อมูลภาพที่ต้องการให้แสดงจากไมโครคอมพิวเตอร์แล้วทำการควบคุมองค์ประกอบย่อยของฮาร์ดแวร์ส่วนอื่น ๆ ในระบบให้ทำงานประสานสอดคล้องกัน เพื่อให้ได้ผลของการแสดงภาพเป็นไปในรูปแบบตามที่ผู้ใช้ได้ออกแบบไว้ สามารถพิจารณาและจำแนกโครงสร้างของระบบรวมที่เกี่ยวข้องสัมพันธ์กันทั้งหมด ในการควบคุมของการแสดงภาพทาง LED แบ่งออกเป็นส่วนประกอบหลักได้คือ

1. ไมโครคอมพิวเตอร์
2. ส่วนควบคุมระบบหลัก
3. ส่วนควบคุมการส่งข้อมูลแนวแถว
4. ส่วนควบคุมการติดของ LED แนวหลัก
5. แผง LED

ส่วนประกอบต่าง ๆ มีความสัมพันธ์กันดังบล็อกไดอะแกรมข้างล่างนี้



บทที่ 4

หน้าที่และหลักการทำงานของระบบ

ระบบรวมของโครงงาน "การแสดงภาพด้วย LED 2 สี ต่างระดับ" นี้ สามารถแยกพิจารณาเพื่อแสดงให้เห็นถึงหน้าที่ และ หลักการทำงานขององค์ประกอบย่อยแต่ละส่วน ซึ่งล้วนมีความสำคัญในการประกอบขึ้นมาเป็นระบบรวม แบ่งได้ดังนี้

1.ไมโครคอมพิวเตอร์ (Microcomputer)

2.ส่วนควบคุมระบบหลัก

3.ส่วนควบคุมการส่งข้อมูลแนวแถว

4.ส่วนควบคุมการติดของ LED แนวหลัก

5.แผง LED ดอทเมตริกซ์

สามารถอธิบายแสดงให้เห็นถึงหน้าที่ และ หลักการทำงานขององค์ประกอบแต่ละส่วนของระบบได้ดังนี้

1.<u>ไมโครคอมพิวเตอร์</u>

ไมโครคอมพิวเตอร์นี้ถูกนำมาใช้เพื่อให้เกิดความสะดวกสำหรับการใช้งาน ทำให้ผู้ใช้สามารถแสดงภาพ หรือ ข้อความในรูปแบบที่ต้องการทางแผง LED ได้โดยไม่ลำบากนัก เพียงทำการออกแบบภาพที่ต้องการแสดง ส่งข้อมูลของภาพที่ต้องการแสดงออกทางพอร์ทอนุกรม ไปยังส่วนควบคุมระบบ แล้วส่งข้อมูลควบคุมรูปแบบของการแสดงภาพตามไป เพื่อให้ไมโครคอนโทรลเลอร์ทำการควบคุมฮาร์ดแวร์ส่วนต่าง ๆ ให้แสดงภาพตามต้องการ

2.<u>ส่วนควบคุมระบบหลัก</u>

เป็นส่วนที่เปรียบเสมือนศูนย์กลางของระบบทั้งหมด มีไมโครคอนโทรลเลอร์เบอร์ 8031 เป็นตัวควบคุม ทำงานตามซอฟท์แวร์คอยตรวจสอบข้อมูลที่ส่งมาจากไมโคร - คอมพิวเตอร์ (ทางพอร์ทอนุกรม) ว่าข้อมูลที่ส่งมาเป็นคำสั่งให้รอรับข้อมูลภาพที่ต้องการแสดง หรือข้อมูลคำสั่งว่าต้องการให้ภาพที่ส่งมาแสดงในรูปแบบใด แล้วทำการแสดงภาพตามที่ผู้ใช้ต้องการ

พิจารณาข้อมูลทั้งที่เป็นอินพุท (Input) และ เอาท์พุท (Output) ของส่วน

ควบคุมระบบหลัก ได้ดังนี้

2.1 *สัญญาณอินพุท*

รับมาจากไมโครคอมพิวเตอร์ทางพอร์ทอนุกรม เป็นข้อมูลแบบอะซิงโครนัสในโหมด 1 (Mode 1) โดยไบท์ (Byte) ข้อมูลควบคุมที่รับมามีความหมายดังนี้

- 0 : คำสั่งให้รอรับข้อมูลภาพที่ต้องการให้แสดงที่จะส่งตามมาอีก 512 ไบท์
- 1 : คำสั่งให้หยุดการแสดงภาพ
- 2 : คำสั่งให้แสดงภาพนิ่ง
- 3 : คำสั่งให้แสดงภาพแบบกระพริบ
- 4 : คำสั่งให้แสดงภาพแบบเลื่อน

2.2 *สัญญาณเอาท์พุท*

ส่งออกจากส่วนควบคุมระบบหลัก เพื่อควบคุมฮาร์ดแวร์ของระบบให้แสดงภาพตามรูปแบบที่ต้องการ แบ่งได้ดังนี้

- พอร์ทของ 8255 ได้แก่

P_A : พอร์ท A ใช้เป็นพอร์ทเอาท์พุท ส่งข้อมูลที่ต้องการแสดงของแบงก์ (Bank) 1 และแบงก์ 3 ไปเป็นอินพุทของตัวแลทช์ข้อมูล (74LS374)

P_B : พอร์ท B ใช้เป็นพอร์ทเอาท์พุท ส่งข้อมูลที่ต้องการแสดงของแบงก์ 2 และแบงก์ 4 ไปเป็นอินพุทของตัวแลทช์ข้อมูล (74LS374)

P_C : พอร์ท C ใช้เป็นพอร์ทเอาท์พุท ส่งข้อมูลไปยังตัวดีโคดเดอร์ (Decoder) เบอร์ 74LS154 เพื่อควบคุมการพักข้อมูลแนวแถวของตัวแลทช์ข้อมูล

- พอร์ท 1 ของ 8031

ใช้ส่งข้อมูลควบคุมการติดของ LED ในแนวหลัก โดยนำสัญญาณจากพอร์ท 1 ตั้งแต่บิทที่ 0 ถึง 4 ไปเป็นอินพุทของตัวดีโคดเดอร์ (74LS154) ให้ข้อมูลที่ต้องการแสดงที่พักในแนวแถว ติดในหลักที่ถูกต้อง

3. <u>ส่วนพักข้อมูล</u>

ในส่วนควบคุมนี้จะแยกออกได้เป็น 4 ส่วน เนื่องจากมีการพักข้อมูลสำหรับการแสดงแบ่งออกเป็น 4 แบงก์ โดยวงจรส่วนควบคุมการส่งข้อมูลแนวแถวของแต่ละแบงก์จะมีโครงสร้างเหมือนกัน ซึ่งสามารถจำแนกเป็นองค์ประกอบย่อยได้ดังนี้

3.1 *ส่วนพักข้อมูล*

สำหรับแต่ละแบงก์ ส่วนพักข้อมูลจะประกอบด้วย IC เบอร์ 74LS374 จำนวน 8 ตัว เป็นที่พักข้อมูลจำนวน 8 ไบท์

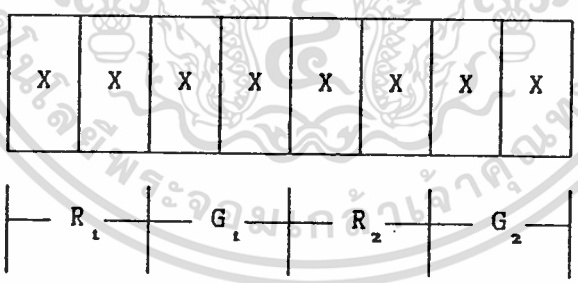
ข้อมูลอินพุทที่เข้ามาในส่วนนี้ได้จากพอร์ท A และพอร์ท B ของ 8255 จากส่วนควบคุมระบบหลัก

3.2 *ส่วนเลือกตัวพักข้อมูล*

ข้อมูลอินพุทของส่วนนี้ได้จากพอร์ท C ของ 8255 จากส่วนควบคุมระบบหลัก บิท 0 ถึง 4 ใช้ดีโคดเดอร์เบอร์ 74LS154 จำนวน 1 ตัว ซึ่งมีสัญญาณเอาท์พุท 16 บิท เพื่อควบคุมว่าต้องการให้ตัวแลทช์ตัวใดทำงาน โดยพักข้อมูลที่ต้องการแสดงในตำแหน่งแถวและแบงก์ที่ถูกต้อง

3.3 *ส่วนแปลงข้อมูลดิจิตอลเป็นอนาลอก*

ใช้แปลงข้อมูลจากข้อมูลภาพที่ต้องการแสดง ซึ่งอยู่ในรูปของข้อมูลดิจิตอลให้เป็นข้อมูลอนาลอก โดยข้อมูลแต่ละไบท์จะเป็นข้อมูลสำหรับ LED 1 ตัว ดังนี้



โดย X : ข้อมูลลอจิก (Logic) 0 หรือ 1

R_1 : ข้อมูลระดับความเข้มสีแดงของ LED ตัวที่ 1

G_1 : ข้อมูลระดับความเข้มสีเขียวของ LED ตัวที่ 1

R_2 : ข้อมูลระดับความเข้มสีแดงของ LED ตัวที่ 2

G_2 : ข้อมูลระดับความเข้มสีเขียวของ LED ตัวที่ 2

3.4 *ส่วนขับกระแส LED*

ทำหน้าที่ขับกระแส LED โดย LED แต่ละตัวจะถูกขับกระแสโดยส่วนขับ

กระแส 2 ชุด คือ ส่วนขับกระแสของแสงสีแดง และ แสงสีเขียว ส่วนนี้เป็นเอาท์พุท ของส่วนควบคุมการส่งข้อมูลแนวแถว

4.ส่วนควบคุมการติดของ LED แนวหลัก

สำหรับส่วนควบคุมนี้ มีความสำคัญในการสแกน (Scan) การติดของ LED ในแนวหลัก คือให้มีการสแกนให้ LED ติดทีละหลักเรียงวนกันไป เพื่อข้อมูลภาพที่ต้องการสามารถแสดงออกทาง LED ได้อย่างถูกต้อง

ประกอบไปด้วยส่วนต่าง ๆ ดังนี้

4.1 *ส่วนเลือกหลักที่ติด*

ใช้ตัวดีโคดเดอร์เบอร์ 74LS154 จำนวน 1 ตัว ประกอบกับอินเวอร์เตอร์ (Inverter) เบอร์ 74LS04 จำนวน 3 ตัว เป็นส่วนประกอบ

มีสัญญาณอินพุท และ เอาท์พุทดังนี้

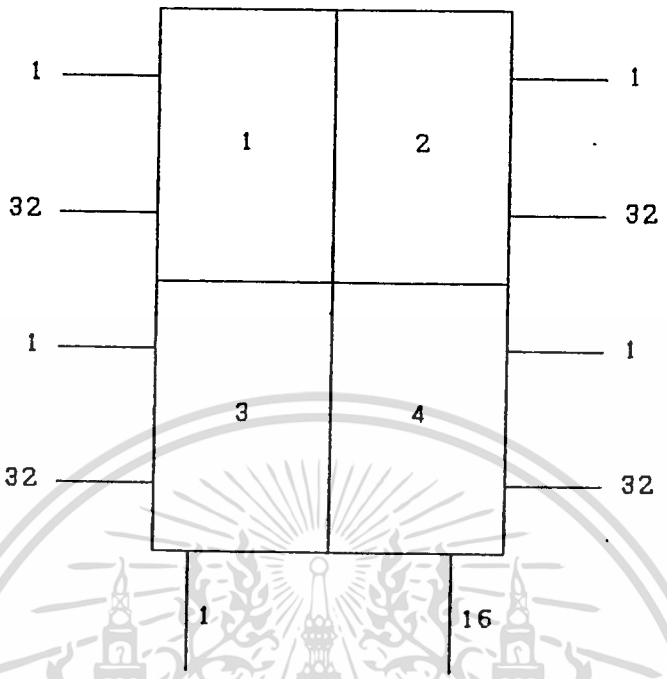
- สัญญาณอินพุท : ได้จากพอร์ท 1 ของ 8031 บิท 0 ถึง 4 เข้าตัวดีโคดเดอร์และนำเอาท์พุทของดีโคดเดอร์ไปเป็นอินพุทของตัวอินเวอร์เตอร์
- สัญญาณเอาท์พุท : ได้จากเอาท์พุทของตัวอินเวอร์เตอร์

4.2 *ส่วนขับกระแสแนวหลัก*

นำเอา MOSFET เบอร์ MIP12N10 จำนวน 16 ตัวมาใช้เป็นตัวขับกระแสของ LED ในแนวหลัก โดยทำหน้าที่เหมือนสวิทช์ (Switch) เปิดและปิดเลือกหลักของ LED ที่ติด

5.แผง LED ดอทเมตริกซ์

ประกอบด้วย LED 2 สี ต่อกันแบบเมตริกซ์ ขนาด 32*32 จุด ซึ่งทำการแบ่งเป็นแบงก์ย่อย 4 แบงก์ ดังภาพ

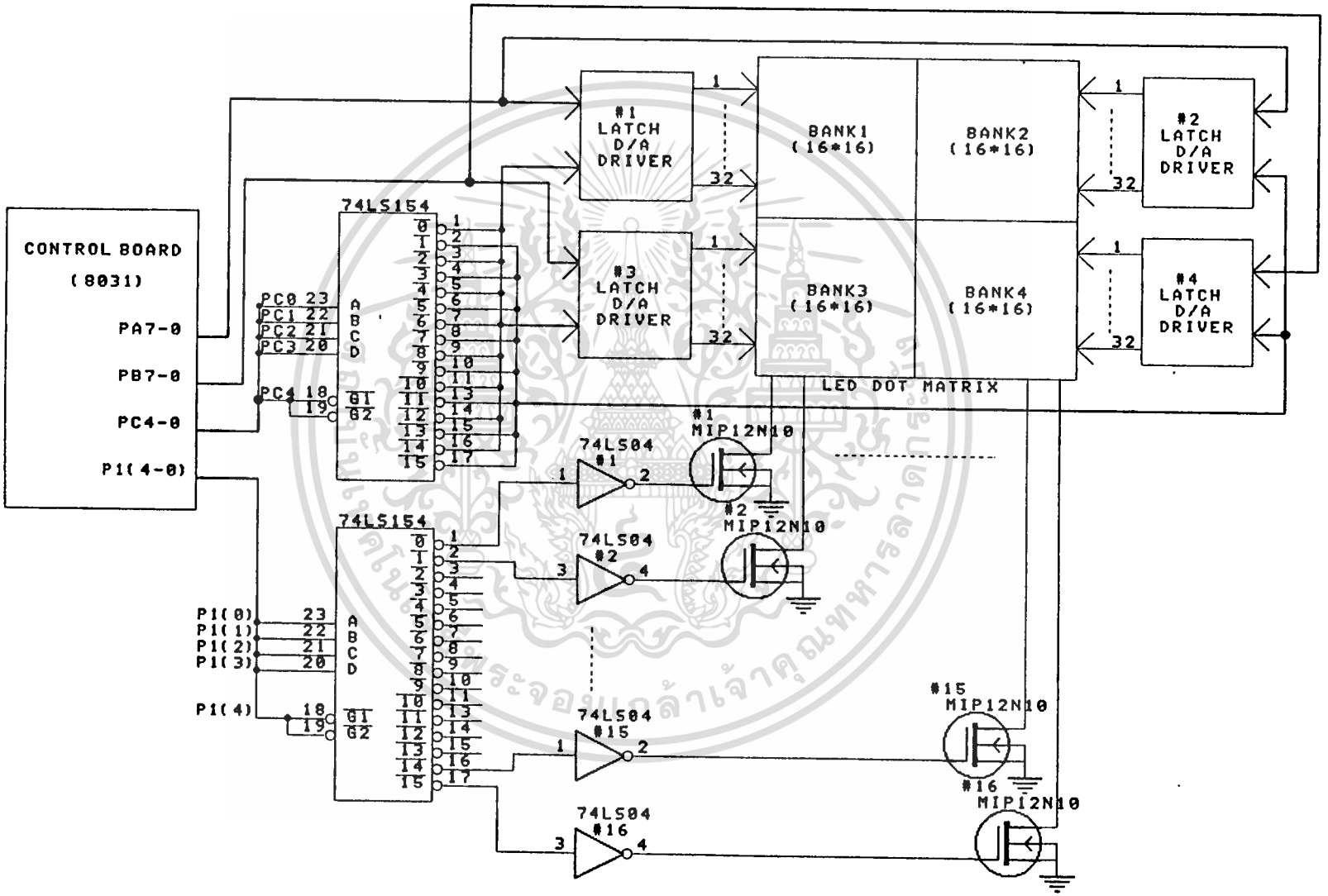


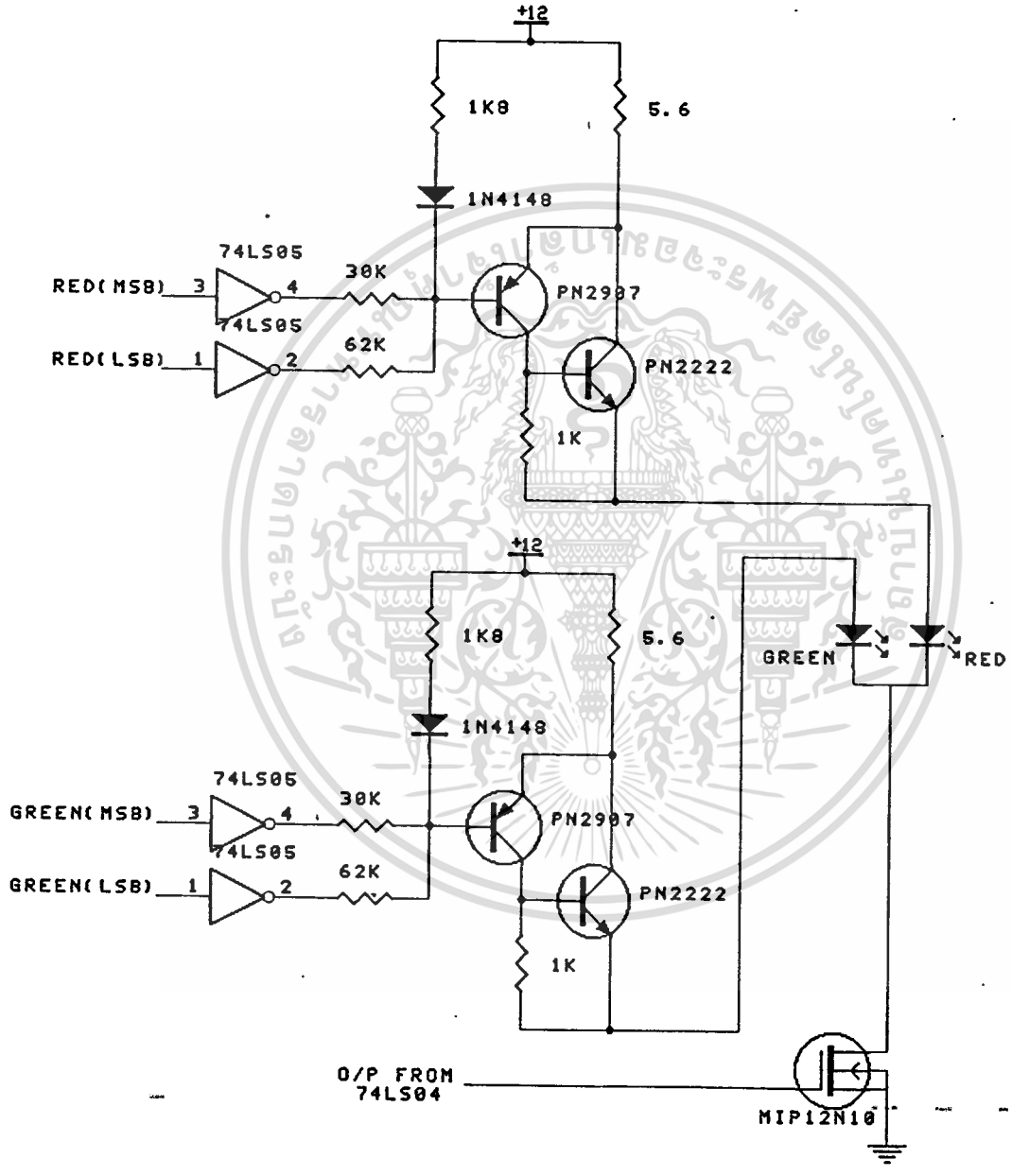
แต่ละแบงก์ประกอบด้วย LED ขนาด 16*16 จุด มีอินพุทในแนวแถว 32 จุด และทุก ๆ แบงก์ทำการเชื่อมต่อแต่ละหลักของ LED เข้าด้วยกัน จึงมีสัญญาณอินพุทในแนวหลัก 16 จุด

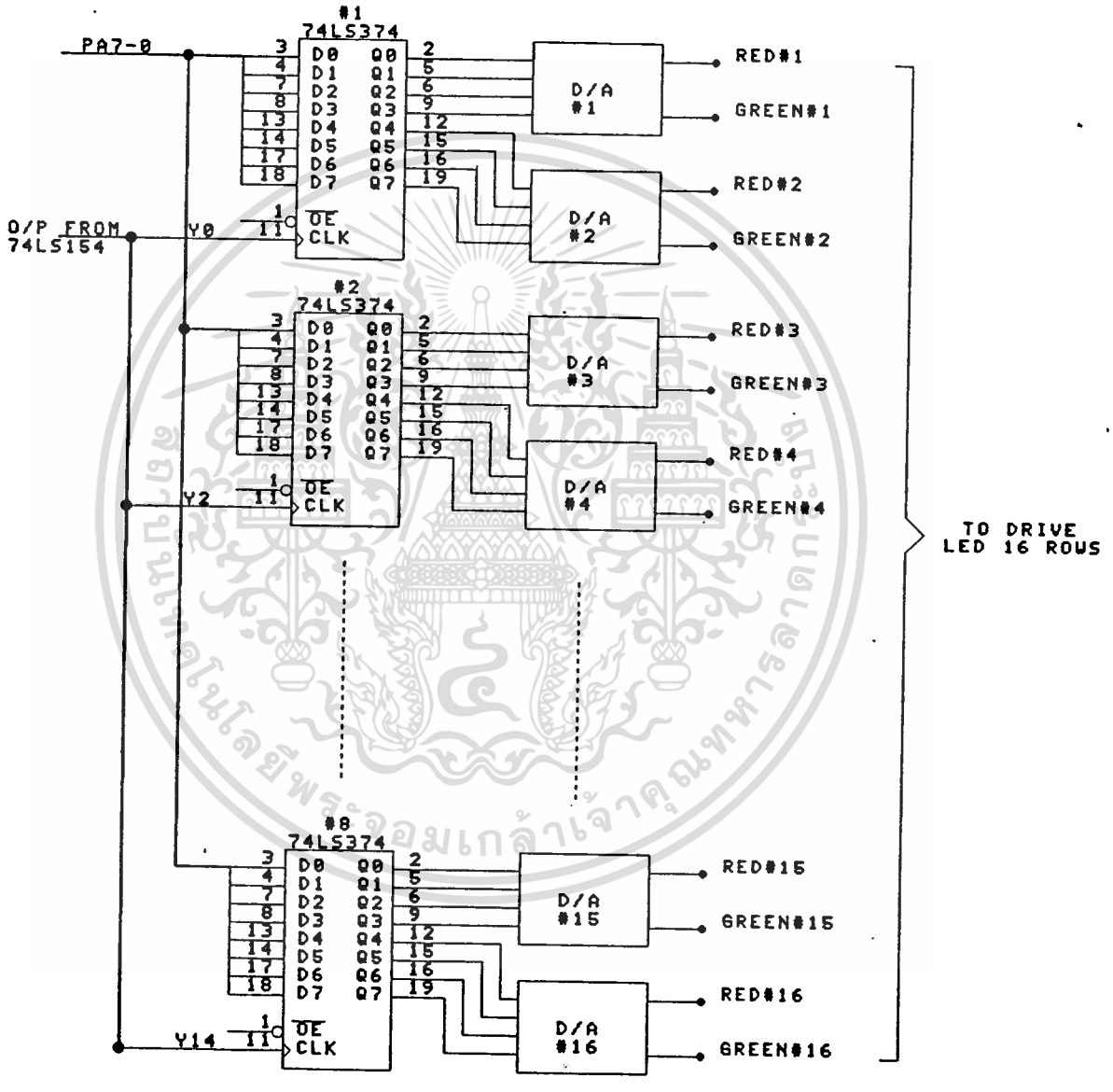
LED แต่ละตัวสามารถเปล่งแสงสีได้ 2 สี คือ แดง และ เขียว โดยอาศัยการควบคุมปริมาณกระแสที่ไหลผ่าน LED ทั้ง 2 แสงสีนี้ ทำให้ได้ระดับความเข้มของแสงสีที่ต่างกันไปถึง 16 ระดับ

บทที่ 5

วงจรของระบบ

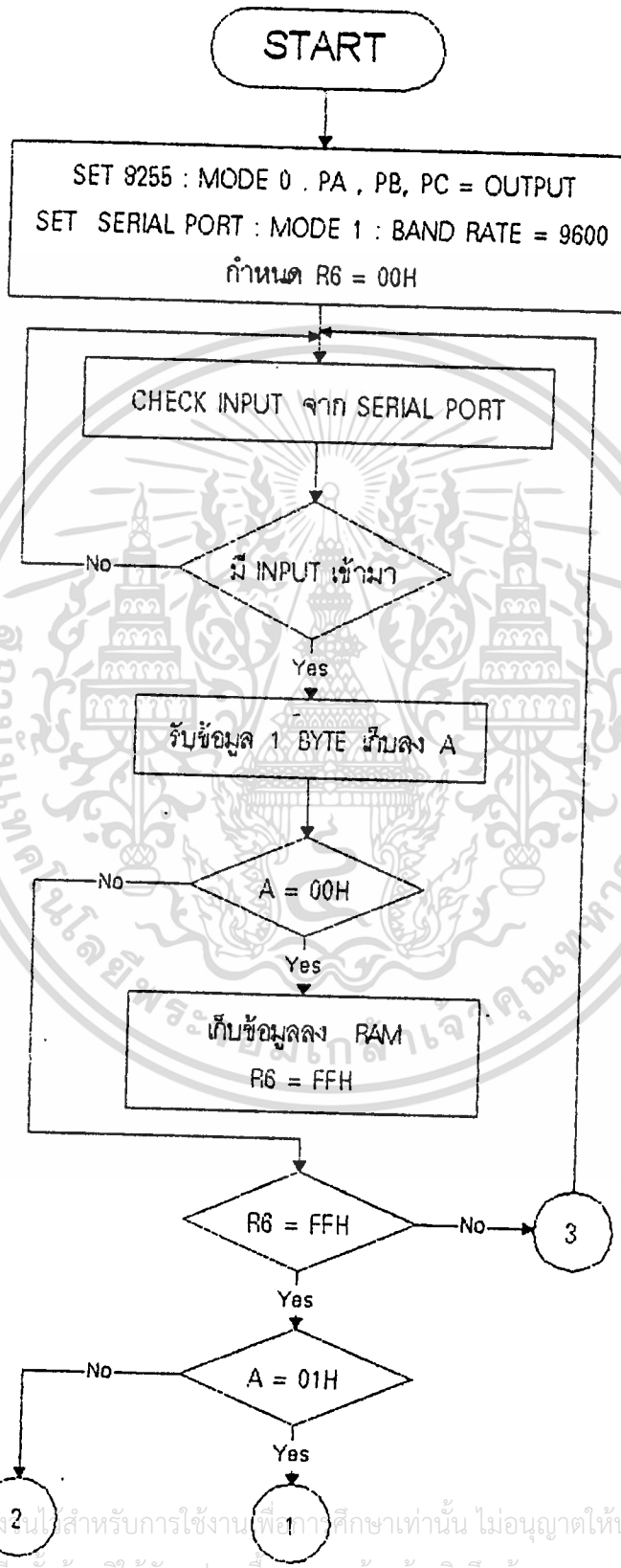

CONTROL BOARD
(8031)
PA7-0
PB7-0
PC4-0
P1(4-0)
#1 LATCH D/A DRIVER
#2 LATCH D/A DRIVER
#3 LATCH D/A DRIVER
#4 LATCH D/A DRIVER
BANK1 (16*16)
BANK2 (16*16)
BANK3 (16*16)
BANK4 (16*16)
LED DOT MATRIX
74LS154
PC0 23
PC1 22
PC2 21
PC3 20
PC4 18
19
A
B
C
D
G1
G2
74LS154
P1(0) 23
P1(1) 22
P1(2) 21
P1(3) 20
P1(4) 18
19
74LS04 #1
74LS04 #2
74LS04 #15
74LS04 #16
#1 MIP12N10
#2 MIP12N10
#15 MIP12N10
#16 MIP12N10

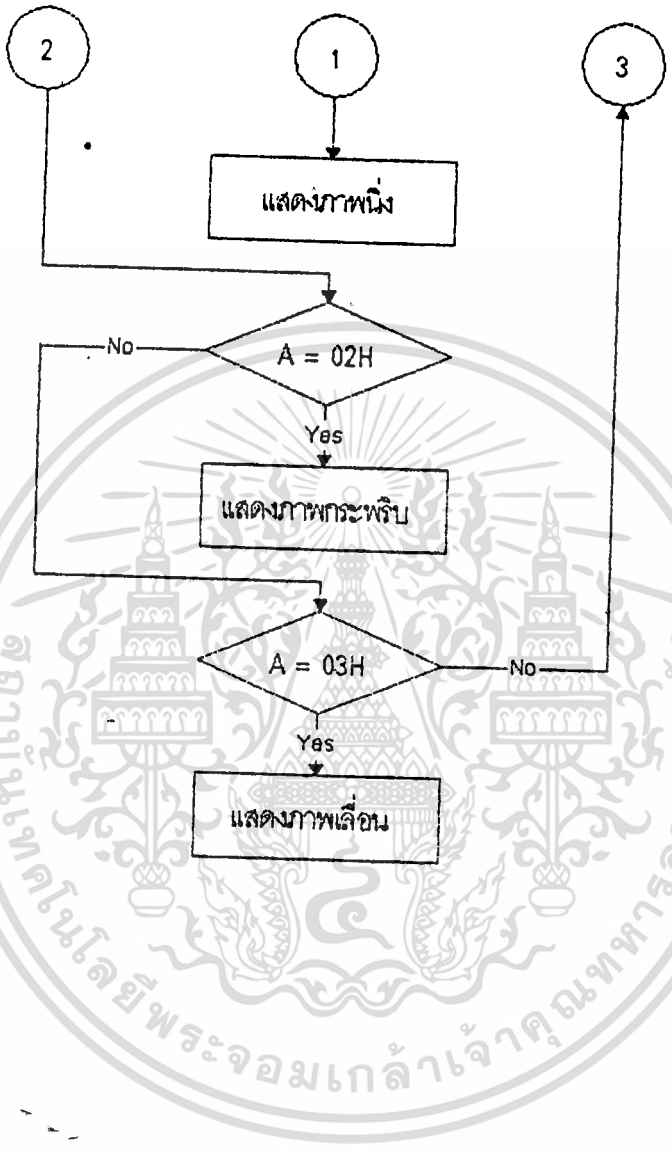

+12
1K8
5.6
1N4148
74LS05
RED(MSB)
3
4
30K
PN2907
74LS05
RED(LSB)
1
2
62K
PN2222
1K
+12
1K8
5.6
GREEN
RED
1N4148
74LS05
GREEN(MSB)
3
4
30K
PN2907
74LS05
GREEN(LSB)
1
2
62K
PN2222
1K
O/P FROM
74LS04
MIP12N10


#1
74LS374
PA7-0
D0 Q0
D1 Q1
D2 Q2
D3 Q3
D4 Q4
D5 Q5
D6 Q6
D7 Q7
OE
CLK
D/P FROM
74LS154
Y0
D/A
#1
D/A
#2
RED#1
GREEN#1
RED#2
GREEN#2
#2
74LS374
Y2
D/A
#3
D/A
#4
RED#3
GREEN#3
RED#4
GREEN#4
#8
74LS374
Y14
D/A
#15
D/A
#16
RED#15
GREEN#15
RED#16
GREEN#16
TO DRIVE
LED 16 ROWS

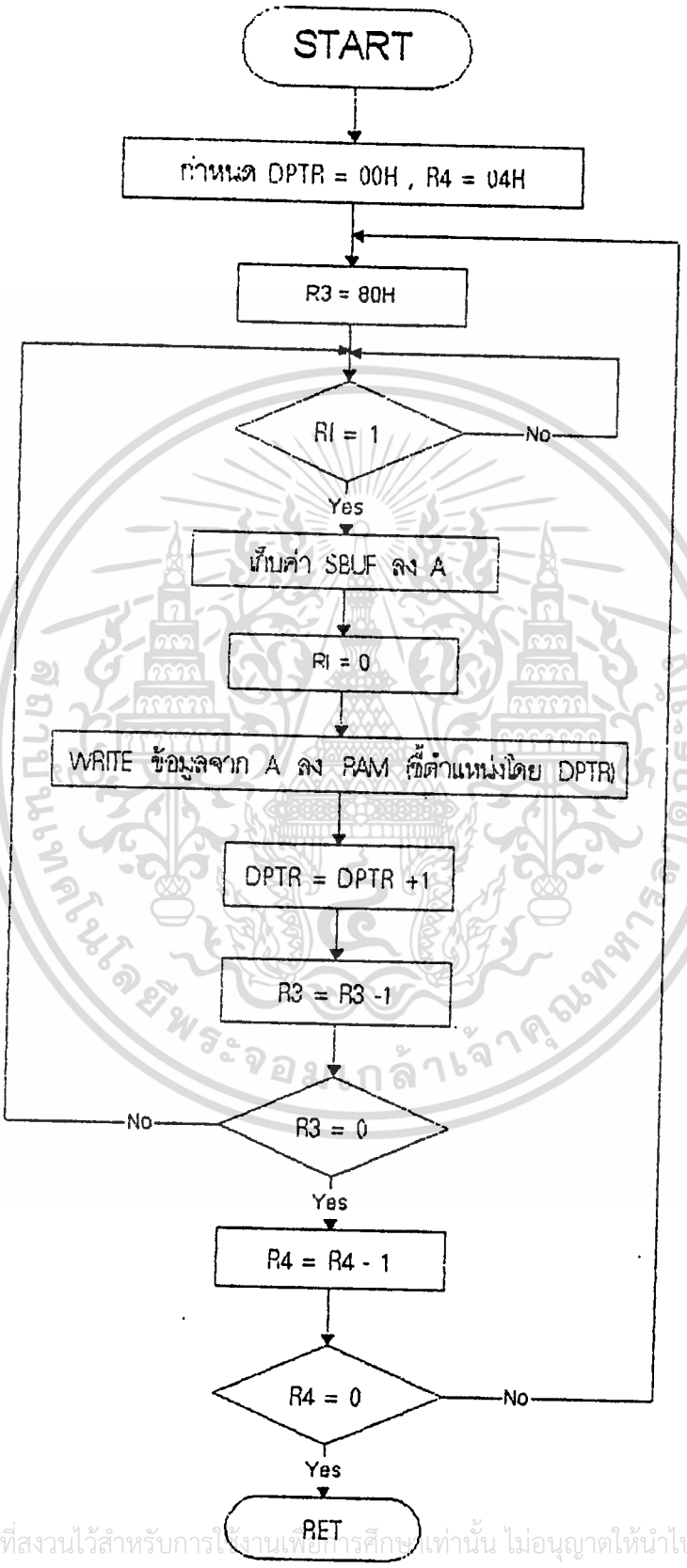
บทที่ 6
องค์ประกอบทางซอฟท์แวร์

โปรแกรมหลัก

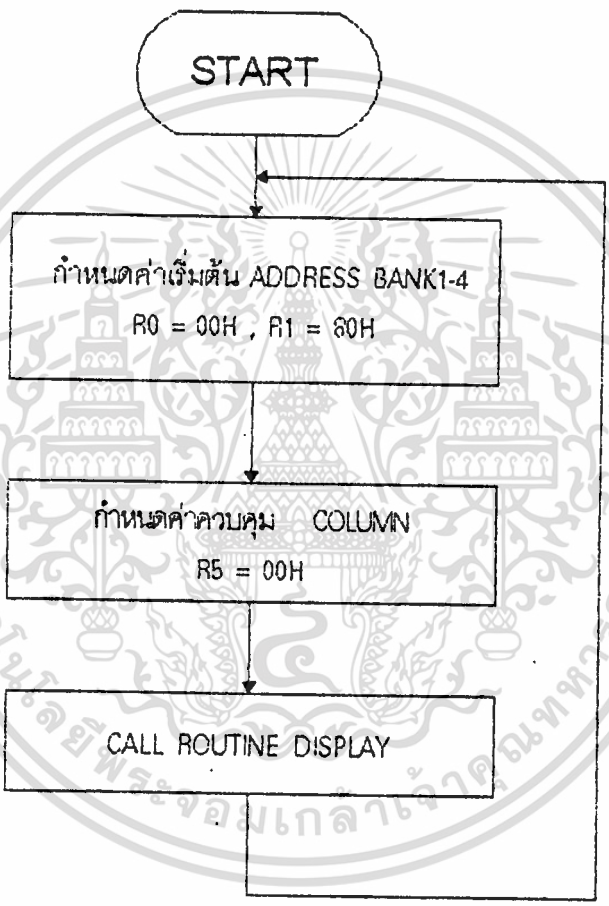




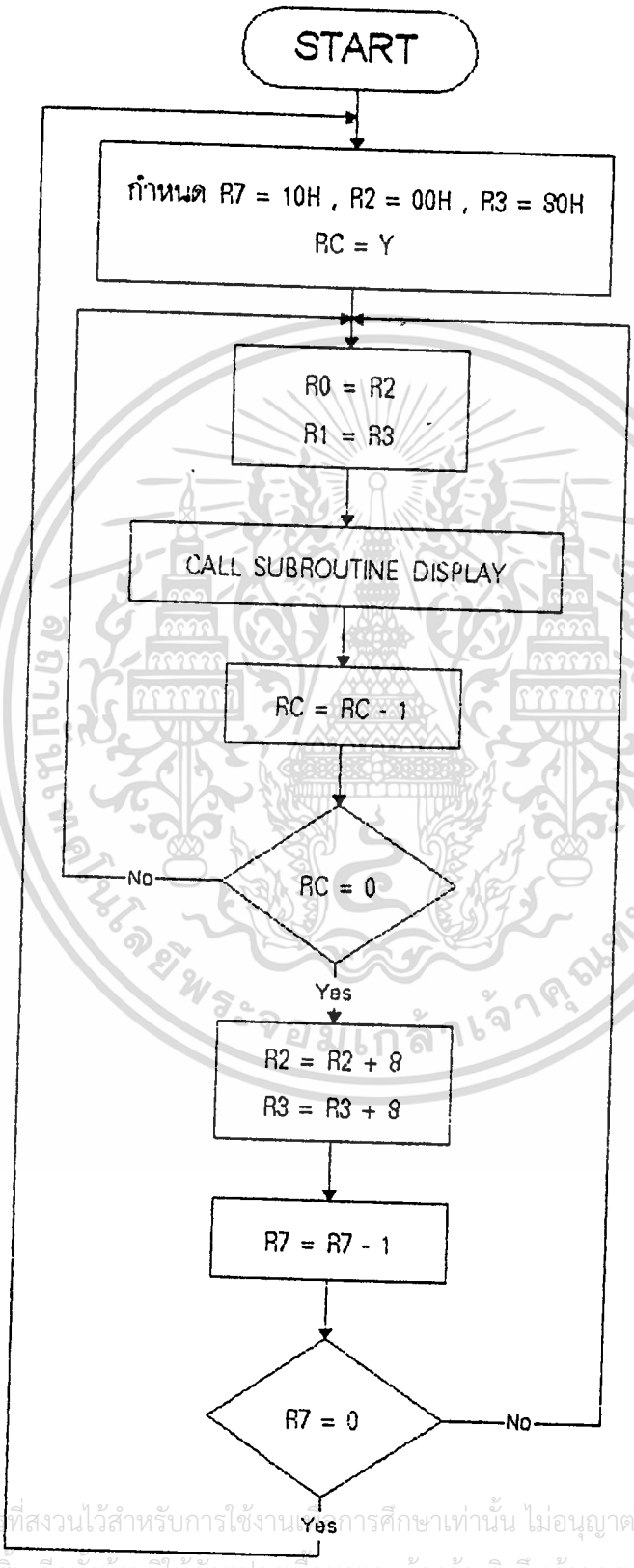
การเก็บข้อมูลภาพลงหน่วยความจำ



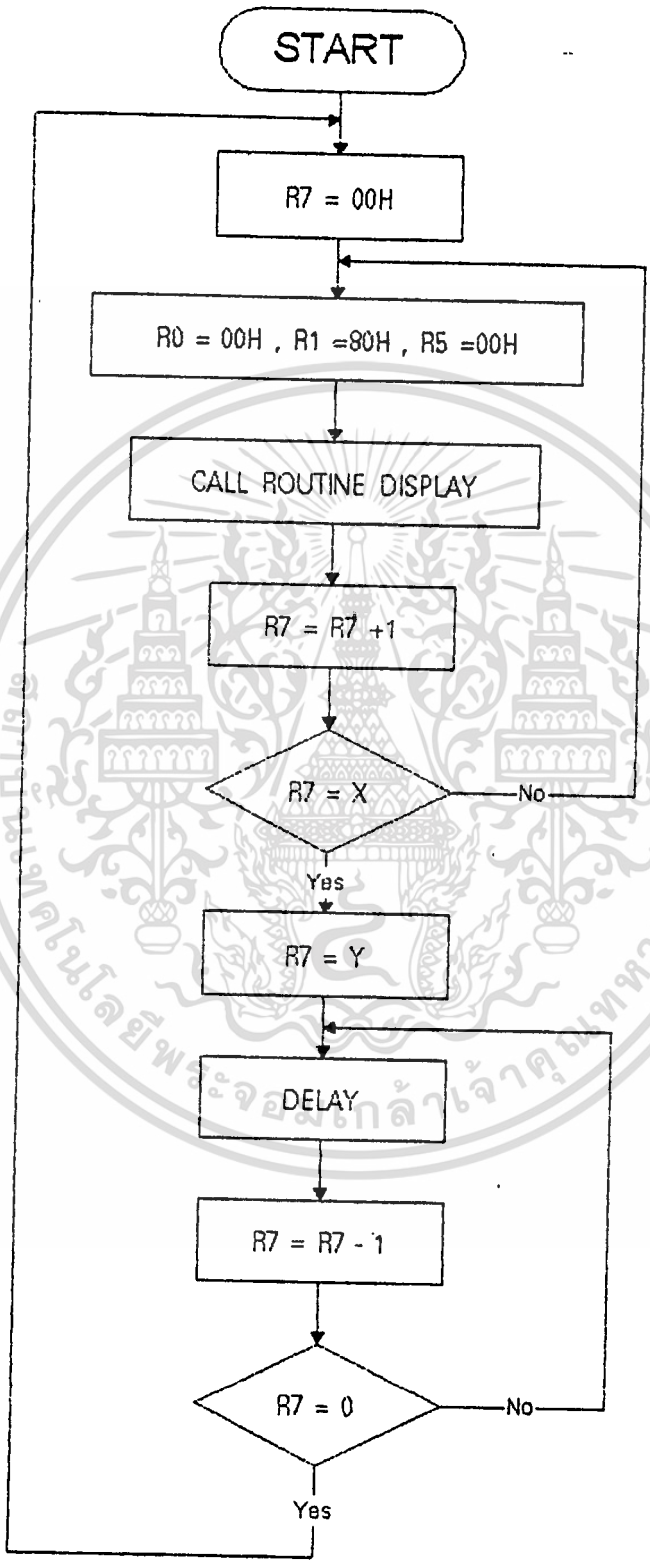
การแสดงภาพนิ่ง



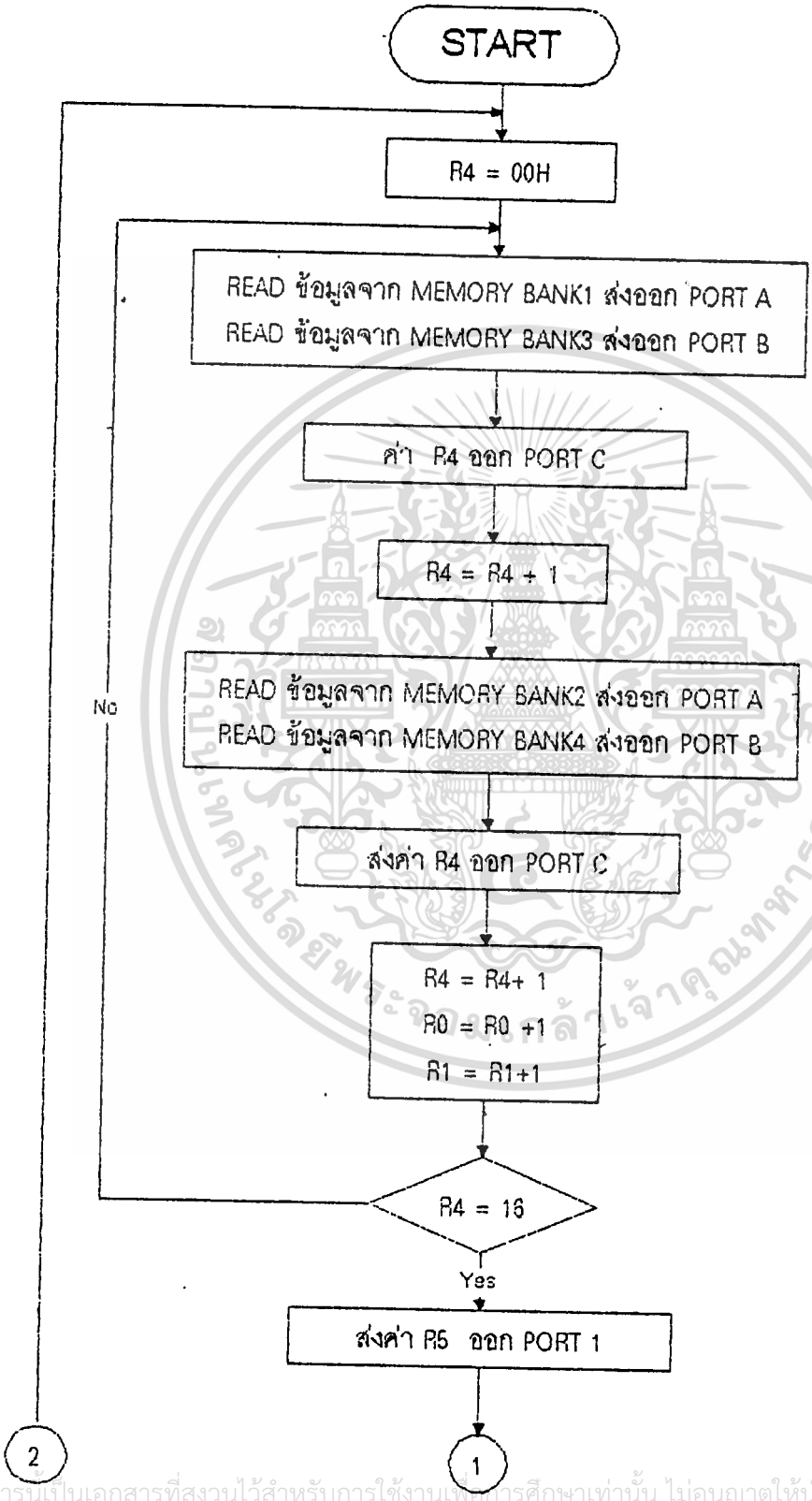
การแสดงภาพเลื่อน

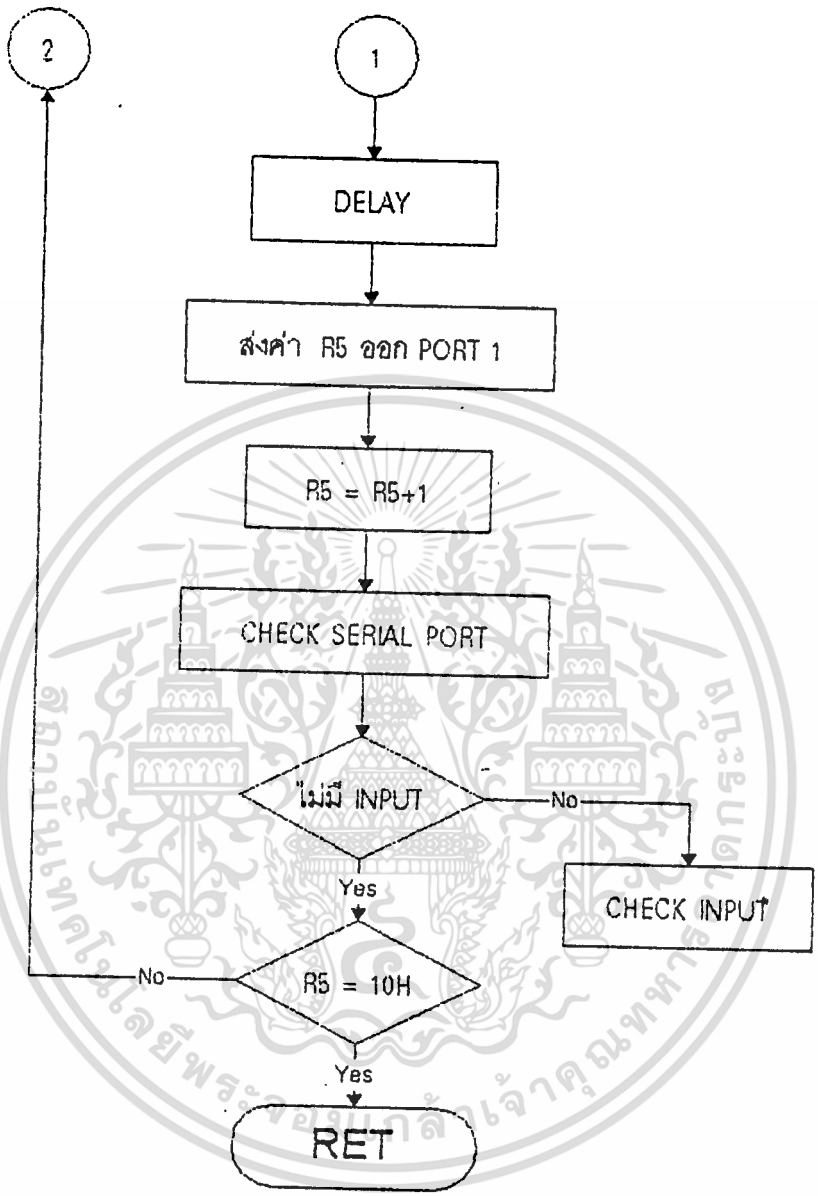


การแสดงภาพกระพริบ



SUBROUTINE DISPLAY





```
          ORG     0000H
P_A       EQU     0E0E0H
P_B       EQU     0E0E1H
P_C       EQU     0E0E2H
PCTRL     EQU     0E0E3H

MAIN:         ACALL   DL
          MOV     SP,#30H
          MOV     A,#10000000B
          MOV     DPTR,#PCTRL
          MOVX    @DPTR,A
          ACALL   DEL

          MOV     P1,#0FFH
          MOV     A,#0FFH
          MOV     DPTR,#P_C
          MOVX    @DPTR,A

       ; ### Set Serial Port ###

          MOV     TMOD,#20H
          MOV     SCON,#52H
          MOV     TH1,#0FDH       ;Baud Rate(9600)
          SETB    TR1

          MOV         R6,#00H             ;Receive Data ?

       ; ### Check Serial Port ###

DETECT:   JNB     RI,DETECT
          CLR     RI
          MOV     A,SBUF

       ; ### Check Command ###

CHECK:    CJNE    A,#00H,COMM     ;Picture
          ACALL   DATA
          MOV     R6,#0FFH
          SJMP    DETECT
COMM:     CJNE    R6,#0FFH,DETECT
NODISP:   CJNE    A,#01H,NORMAL
          SJMP    DETECT
NORMAL:   CJNE    A,#02H,BLINK      ;Normal
          SJMP    NORM
BLINK:    CJNE    A,#03H,SHIFT      ;Blink
          SJMP    BLK
SHIFT:    CJNE    A,#04H,DETECT     ;Shift
          SJMP    SHI

      ; ### Normal ###

NORM:     MOV     R0,#00H           ;Initial Address Bank1,3
          MOV     R1,#80H           ;Initial Address Bank2,4
          MOV     R5,#00H           ;Control Decoder(Coluan)
          ACALL   DISP
          SJMP    NORM
```

```
        ; ***  Blink  ***

BLK:    MOV     R7,#00H         ;Counter
BLK0:   MOV     R0,#00H
        MOV     R1,#80H
        MOV     R5,#00H
        ACALL   DISP
        INC     R7
        CJNE    R7,#20H,BLK0
        MOV     R7,#02H
BLK1:   ACALL   DL
        DJNZ    R7,BLK1
        SJMP    BLK

      ; ***  Shift Left ***

SHI:    MOV     R7,#20H
        MOV     R2,#00H
        MOV     R3,#80H
        SETB    RS0
        SETB    RS1
        MOV     R4,#05H         ;Set Delay
SSHI:   CLR     RS0
        CLR     RS1
        MOV     A,R2
        MOV     R0,A
        MOV     A,R3
        MOV     R1,A
        MOV     R5,#00H
        ACALL   DISP
        SETB    RS0
        SETB    RS1
        DJNZ    R4,SSHI
        CLR     RS0
        CLR     RS1
        MOV     A,#08H
        ADD     A,R2
        MOV     R2,A
        MOV     A,#08H
        ADD     A,R3
        MOV     R3,A
        DJNZ    R7,SSHI
        SJMP    SHI

      ; ***  Display  ***

DISP:   MOV     R4,#00H         ;Control Latch(Row)
ROW:    MOV     P2,#00H
        MOVX    A,@R0           ;Read Data Bank1
        MOV     DPTR,#P_A
        MOVX    @DPTR,A         ;Out Port_A
        MOV     P2,#01H
        MOVX    A,@R0           ;Read Data Bank3
        MOV     DPTR,#P_B
        MOVX    @DPTR,A         ;Out Port_B
        MOV     A,R4
        MOV     DPTR,#P_C
```

```
        MOVX    @DPTR,A         ;Latch Data Bank1,3
        INC     R4

        MOV     P2,#00H
        MOVX    A,@R1           ;Read Data Bank2
        MOV     DPTR,#P_A
        MOVX    @DPTR,A         ;Out Port_A
        MOV     P2,#01H
        MOVX    A,@R1           ;Read Data Bank4
        MOV     DPTR,#P_B
        MOVX    @DPTR,A         ;Out Port_B
        MOV     A,R4
        MOV     DPTR,#P_C
        MOVX    @DPTR,A         ;Latch Data Bank2,4
        INC     R4

        INC     R0
        INC     R1
        CJNE    R4,#00010000B,ROW

COL:    MOV     P1,R5
        ACALL   DELAY
        MOV     P1,#0FFH
        INC     R5

        JNB     RI,CONT         ;Check Serial Port
        MOV     A,SBUF
        CLR     RI
        AJMP    CHECK

CONT:   CJNE    R5,#00010000B,DISP
        RET

    ; ### Receive Data(RAM1) ###

DATA:   MOV     DPTR,#00H
        MOV     R4,#04H
DATA0:  MOV     R3,#80H
DATA1:  JNB     RI,DATA1
        CLR     RI
        MOV     A,SBUF
        MOVX    @DPTR,A         ;Write data
        INC     DPTR
        DJNZ        R3,DATA1
        DJNZ        R4,DATA0
        RET

    ; ### Delay ###

DL:     SETB    RS0
        SETB    RS1
        MOV     R0,#0FFH
DL0:    MOV     R1,#0FFH
DL1:    DJNZ    R1,DL1
        DJNZ    R0,DL0
        CLR     RS0
        CLR     RS1
```

```
        RET

DEL:    SETB    RS0
        SETB    RS1
        MOV     R0,#01H
DE0:    MOV     R1,#01H
DE1:    DJNZ    R1,DE1
        DJNZ    R0,DE0
        CLR     RS0
        CLR     RS1
        RET

DELAY:  SETB    RS0
        SETB    RS1
        MOV     R0,#01H
DEL0:   MOV     R1,#0FFH
DEL1:   DJNZ    R1,DEL1
        DJNZ    R0,DEL0
        CLR     RS0
        CLR     RS1
        RET

        END
```

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <bios.h>

typedef int boolean;
#define true  1
#define false 0

#define THRE       0x20
#define DR         0x01
#define TX         0
#define RX         0
#define LSR        5

void init_col(void);
void far *getmem(unsigned long size);
void outcharxy(int xpos,int ypos,char ch);
void write(char ch);
void write_text(int xpos_1,int xpos_2,int ypos,char *str,int col);
char readtext(int xpos,int ypos,char *gettext
            ,int max_char,int given_char,boolean edit);
void box(int x1,int y1,int x2,int y2,boolean type);
void frame(int x1,int y1,int x2,int y2,boolean bar_ok);
int load_save(char *file_name,int func);
char readfunc(void);

struct palettetype pal;
int x_offset=70,y_offset=10;
char current_col=15;
boolean end_edit=false;
char screen[32][32];
char font[256][5][7];
int  port=0;// COM 1 For Default

char command[20][50];
char command_temp[60];
int  parameter_count;
int command_index=-1;
extern boolean funckey=false;

void init_graph(void){
  int gdriver = VGA, gmode = VGAHI, errorcode;
  int i;

  /* initialize graphics and local variables */
  initgraph(&gdriver, &gmode, "");
  /* read result of initialization */
  errorcode=graphresult();
  if(errorcode!=grOk){  /* an error occurred */
    printf("Graphics error: %s\n", grapherrormsg(errorcode));
    printf("Press any key to halt:");
    readfunc();
    exit(1); /* terminate with an error code */
    }
```

```
    /* grab a copy of the palette */
    getpalette(&pal);
    /* create gray scale */
    for(i=0;i<pal.size;i++)
      setrgbpalette(pal.colors[i],(21*(i&0x0C)>>2),21*(i&0x03),0);
      }

  // Line Command
  #define PUTPIXEL  0
  #define LINE      1
  #define BAR       2
  #define RECTANGLE 3
  #define OUTTEXT   4
  #define LOAD      5
  #define SAVE      6
  #define CLS       7
  #define SETCOLOR  8
  #define QUIT      9
  #define DISPLAY  10
  #define SENDDATA 11

  // Display Command
  #define SEND_DATA 0
  #define NODISPLAY 1
  #define NORMAL    2
  #define BLINKING  3
  #define SHIFT     4

 void put_pixel(int x,int y,char col){
   if((x>=0)&&(x<32)&&(y>=0)&&(y<32)){
     setcolor(col);
     setfillstyle(1,col);
     pieslice(x*14+x_offset,y*14+y_offset,0,360,6);
     screen[x][y]=col;
     if(col==0) putpixel(x*14+x_offset,y*14+y_offset,15);
     }
  }

void g_line(int x1,int y1,int x2,int y2,int col){
  int i,xx1,xx2,yy1,yy2,step;

  if(x1<x2){
    xx1=x1;xx2=x2;
    }
  else{
    xx1=x2,xx2=x1;
    }
  if(y1<y2){
    yy1=y1;yy2=y2;
    }
  else{
    yy1=y2,yy2=y1;
    }
  if((x1==x2)&&(y1==y2))
    put_pixel(x1,y1,col);
  else{
    if(yy2-yy1>xx2-xx1){
```

```
        put_pixel(xx2,i,col);
      }
    for(i=xx1;i<=xx2;i++){
      put_pixel(i,yy1,col);
      put_pixel(i,yy2,col);
      }
    }

  void g_outtextxy(int x,int y,char *text,char col){
    int  i,j,k;

    for(i=0;i<strlen(text);i++){
      if(text[i]=='j')
        put_pixel(x+i*6+4,y,col);
      for(k=0;k<7;k++){
        for(j=0;j<5;j++){
          if(font[text[i]][j][k]==0xF)
            if((text[i]=='g')||(text[i]=='j')||(text[i]=='p')||
              (text[i]=='q')||(text[i]=='y'))
              put_pixel(x+i*6+j,y+k+2,col);
            else
              put_pixel(x+i*6+j,y+k,col);
          }
        }
      }
    }


  void send_char(char ch){
    bioscom(_COM_SEND,ch,port);
    }

  void init_rs232c(void){
    bioscom(_COM_INIT,_COM_9600|_COM_NOPARITY|_COM_STOP1|_COM_CHR8,port);
    }

  void load_font(void){
    FILE *fp;
    int i,j,k;
    char temp;


    if((fp=fopen("FONT.TXT","rb"))!=NULL){
      // 0 to 9
      for(i=48;i<=57;i++){
        for(k=0;k<7;k++){
          for(j=0;j<5;j++){
            temp=fgetc(fp);
            if(temp=='0')
              font[i][j][k]=0x0F;
            else
              font[i][j][k]=0x00;
            }
          fgetc(fp);
          fgetc(fp);
          }
        fgetc(fp);
```

```
        if(y2>y1)step=1;
        else step=-1;
        i=y1-step;
        do{
          i+=step;
          put_pixel(x1+((i-y1)*(x2-x1)+(y2-y1)/2)/(y2-y1),i,col);
          }while(i!=y2);
        }
      else{
        if(x2>x1)step=1;
        else step=-1;
        i=x1-step;
        do{
          i+=step;
          put_pixel(i,y1+((i-x1)*(y2-y1)+(x2-x1)/2)/(x2-x1),col);
          }while(i!=x2);
        }
      }
    }

  void g_bar(int x1,int y1,int x2,int y2,int col){
    int i,j,xx1,xx2,yy1,yy2;

    if(x1<x2){
      xx1=x1;xx2=x2;
      }
    else{
      xx1=x2,xx2=x1;
      }
    if(y1<y2){
      yy1=y1;yy2=y2;
      }
    else{
      yy1=y2,yy2=y1;
      }
    for(i=yy1;i<=yy2;i++)
      for(j=xx1;j<=xx2;j++)
        put_pixel(j,i,col);
    }

  void g_rectangle(int x1,int y1,int x2,int y2,int col){
    int i,xx1,xx2,yy1,yy2;

    if(x1<x2){
      xx1=x1;xx2=x2;
      }
    else{
      xx1=x2,xx2=x1;
      }
    if(y1<y2){
      yy1=y1;yy2=y2;
      }
    else{
      yy1=y2,yy2=y1;
      }
    for(i=yy1;i<=yy2;i++){
      put_pixel(xx1,i,col);
```

```
        fgetc(fp);
        }
      // A to Z
      for(i=65;i<=90;i++){
        for(k=0;k<7;k++){
          for(j=0;j<5;j++){
            temp=fgetc(fp);
            if(temp=='0')
              font[i][j][k]=0x0F;
            else
              font[i][j][k]=0x00;
            }
          fgetc(fp);
          fgetc(fp);
          }
        fgetc(fp);
        fgetc(fp);
        }
      // a to z
      for(i=97;i<=122;i++){
        for(k=0;k<7;k++){
          for(j=0;j<5;j++){
            temp=fgetc(fp);
            if(temp=='0')
              font[i][j][k]=0x0F;
            else
              font[i][j][k]=0x00;
            }
          fgetc(fp);
          fgetc(fp);
          }
        fgetc(fp);
        fgetc(fp);
        }
      }
    }

  void save(void){
    FILE *fp;
    char name[80];
    char ch;

    ch=load_save(name,1);
    if(!ch){
      if((fp=fopen(name,"wb"))!=NULL){
        fwrite(&screen[0][0],sizeof(screen),1,fp);
        fclose(fp);
        }
      }
    }

  void load(void){
    FILE *fp;
    int x,y;
    char name[80];
    char ch;
```

```
    ch=load_save(name,0);
    if(!ch){
      if((fp=fopen(name,"rb"))!=NULL){
        fread(&screen[0][0],sizeof(screen),1,fp);
        setfillstyle(1,0);
        bar(x_offset-6,y_offset-6,x_offset+447,y_offset+447);
        for(y=0;y<32;y++)
          for(x=0;x<32;x++)
            if(screen[x][y]==0)
              putpixel(x*14+x_offset,y*14+y_offset,15);
            else
              put_pixel(x,y,screen[x][y]);
        fclose(fp);
        }
      }
    }

  void cls(void){
    int x,y;

    setfillstyle(1,0);
    bar(x_offset-6,y_offset-6,x_offset+447,y_offset+447);
    for(y=0;y<32;y++)
    for(x=0;x<32;x++){
      screen[x][y]=0;
      putpixel(x*14+x_offset,y*14+y_offset,15);
      }
    }

  void new_color(int col){
    setcolor(0);
    rectangle(600-2,current_col*20+20-2,630+2,current_col*20+34+2);
    current_col=col;
    setcolor(15);
    rectangle(600-2,current_col*20+20-2,630+2,current_col*20+34+2);
    }

  void display(char *type){
    int  display_type=1;
    char *u_type;

    u_type=strupr(type);
    if(strcmp(u_type,"NODISPLAY")==0)
      display_type=NODISPLAY;
    else if(strcmp(u_type,"NORMAL")==0)
      display_type=NORMAL;
    else if(strcmp(u_type,"BLINK")==0)
      display_type=BLINKING;
    else if(strcmp(u_type,"SHIFT")==0)
      display_type=SHIFT;
    else
      display_type=-1;
    if(display_type!=-1)
      send_char((char)display_type);
    }

  void send_data(void){
```

```
    int i,j,k;
    char data;

    send_char(SEND_DATA);
    readfunc();
    for(i=0;i<4;i++){     // LED Diaplsy Board 1 - 4
      for(j=0;j<16;j++){  // 16  Row
        for(k=0;k<8;k++){ // 16  Colum  8 Bit For 2 Dot; 8 Byte For 16 Dot
          // x position of screen = 16*(i/2)+j
          // y position of screen = 16*(i%2)+k/2   ; Upper dot
          //                  and   16*(i%2)+k/2+1 ; Lower dot
          data=(screen[16*(i%2)+j][16*(i/2)+k*2]<<4)|
               (screen[16*(i%2)+j][16*(i/2)+k*2+1]);
          send_char(data);
          }
        }
      }
    }

  void classify_command(void){

    if(!strcmp(command[0],"PUTPIXEL"))
      command_index=PUTPIXEL;
    else if(!strcmp(command[0],"LINE"))
      command_index=LINE;
    else if(!strcmp(command[0],"BAR"))
      command_index=BAR;
    else if(!strcmp(command[0],"RECTANGLE"))
      command_index=RECTANGLE;
    else if(!strcmp(command[0],"OUTTEXT"))
      command_index=OUTTEXT;
    else if(!strcmp(command[0],"LOAD"))
      command_index=LOAD;
    else if(!strcmp(command[0],"SAVE"))
      command_index=SAVE;
    else if(!strcmp(command[0],"CLS"))
      command_index=CLS;
    else if(!strcmp(command[0],"SETCOLOR"))
      command_index=SETCOLOR;
    else if(!strcmp(command[0],"SENDDATA"))
      command_index=SENDDATA;
    else if(!strcmp(command[0],"DISPLAY"))
      command_index=DISPLAY;
    else if(!strcmp(command[0],"QUIT"))
      command_index=QUIT;
    else
      command_index=-1;
    }

  void split_command(void){
    int i,c=0;
    char *coma,temp[60];
    boolean first_coma=true;
    coma=command_temp;
    parameter_count=0;
    for(i=0;i<strlen(coma);i++){
```

```
      if(!((first_comm)&&(comm[i]==' '))){
       if((comm[i]!=',')&&(comm[i]!=' ')){
         temp[c++]=comm[i];
         first_comm=false;
         }
       else if((comm[i]==' ')&&(*(comm+(unsigned)(i-1))!=',')&&(*(comm+(unsigned)(i-1))!='
 '))){
         temp[c]=0;
         c=0;
         strcpy(command[parameter_count++],temp);
         }
       else if(comm[i]==','){
         temp[c]=0;
         c=0;
         if(*(comm+(unsigned)(i-1))!=' ')
           strcpy(command[parameter_count++],temp);
         strcpy(command[parameter_count++],",");
         }
       }
     }
     if(c!=0){
       temp[c]=0;
       strcpy(command[parameter_count++],temp);
       }
   comm=strupr(command[0]);
   strcpy(command[0],comm);
   }

 void execute_command(void){
   switch(command_index){
     case PUTPIXEL  :put_pixel(atoi(command[1]),atoi(command[3]),current_col);break;
     case LINE      :g_line(atoi(command[1]),atoi(command[3]),
                      atoi(command[5]),atoi(command[7]),current_col);
                     break;
     case BAR       :g_bar(atoi(command[1]),atoi(command[3]),
                      atoi(command[5]),atoi(command[7]),current_col);
                     break;
     case RECTANGLE :g_rectangle(atoi(command[1]),atoi(command[3]),
                       atoi(command[5]),atoi(command[7]),current_col);
                     break;
     case OUTTEXT   :g_outtextxy(atoi(command[1]),atoi(command[3]),
                       command[5],current_col);
                     break;
     case LOAD      :load();break;
     case SAVE      :save();break;
     case CLS       :cls();break;
     case SETCOLOR  :new_color(atoi(command[1]));break;
     case SENDDATA  :send_data();break;
     case DISPLAY   :display(command[1]);break;
     case QUIT      :end_edit=true;break;
     case -1        :break;
     }
 }

void editor(void){
#define EDIT    0
#define COMMAND 1
```

```
int x_pos=0,y_pos=0;
char ch;
char shift_status;
int mode = EDIT;

do{
  if(mode==EDIT){
    setcolor(15);
    circle(x_pos*14+x_offset,y_pos*14+y_offset,6);
    ch=readfunc();
    asm{
      mov ah,0x02
      int 0x16
      and al,0x03
      mov shift_status,al
      }
    put_pixel(x_pos,y_pos,screen[x_pos][y_pos]);
    if(screen[x_pos][y_pos]==0)
      putpixel(x_pos*14+x_offset,y_pos*14+y_offset,15);
    if(funckey){
      if((shift_status)&&((ch==75)||(ch==77)||(ch==72)||(ch==80))){
        put_pixel(x_pos,y_pos,current_col);
        if(screen[x_pos][y_pos]==0)
          putpixel(x_pos*14+x_offset,y_pos*14+y_offset,15);
        }
      switch(ch){
        case 75:if(x_pos>0)  x_pos--;break;
        case 77:if(x_pos<31) x_pos++;break;
        case 72:if(y_pos>0)  y_pos--;break;
        case 80:if(y_pos<31) y_pos++;break;
        case 60:save();break;
        case 61:load();break;
        case 62:cls();break;
        case 45:end_edit=true;break;
        }
      }
    else
      switch(ch){
        case 32:put_pixel(x_pos,y_pos,current_col);break;
        case 27:mode=COMMAND;break;
        case '0':new_color(0);break;
        case '1':new_color(1);break;
        case '2':new_color(2);break;
        case '3':new_color(3);break;
        case '4':new_color(4);break;
        case '5':new_color(5);break;
        case '6':new_color(6);break;
        case '7':new_color(7);break;
        case '8':new_color(8);break;
        case '9':new_color(9);break;
        case 'a':new_color(10);break;
        case 'b':new_color(11);break;
        case 'c':new_color(12);break;
        case 'd':new_color(13);break;
        case 'e':new_color(14);break;
        case 'f':new_color(15);break;
```

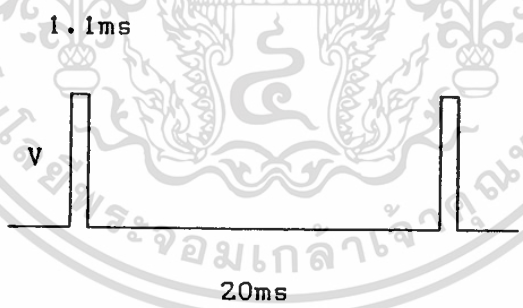
```
            }
          }
        else{
          strcpy(command_temp,"");
          ch=readtext(80,465,command_temp,40,40,false);
          if((ch==27)&&(!funckey))mode=EDIT;
          else if((ch==13)&&(!funckey)){
            split_command();
            classify_command();
            execute_command();
            }
          }
        }while(!end_edit);
      }

    void main(int param_count,char *param[]){
      int x,y;
      char temp[3];

      if(param_count>1){
        port=atoi(param[1])-1;
        }
      init_graph();
      init_col();
      load_font();
      init_rs232c();
      setcolor(2);
      for(x=0;x<16;x++){
        setfillstyle(1,x);
        bar(600,x*20+20,630,x*20+34);
        sprintf(temp,"%2d",x);
        outtextxy(570,x*20+24,temp);
        }
      setcolor(15);
      x=15;
      rectangle(600-2,x*20+20-2,630+2,x*20+34+2);
      for(x=0;x<32;x++)
      for(y=0;y<32;y++)
        screen[x][y]=0;
      for(y=0;y<32;y++)
      for(x=0;x<32;x++){
        if(screen[x][y]==0)
          putpixel(x*14+x_offset,y*14+y_offset,15);
        else
          put_pixel(x,y,screen[x][y]);
                }
      setcolor(3);
      outtextxy(0,472-7,"COMMAND : ");
      editor();
      closegraph();
      }
```

บทที่ 7

ผลการทดลอง

ในการทำโครงงานนี้ สำหรับฮาร์ดแวร์ในส่วนที่ทำหน้าที่เป็นส่วนแปลงข้อมูลดิจิตอลเป็นอนาลอก และส่วนของการขับกระแสในแนวแถว ต้องทำการทดลองเพื่อเซ็ตค่าของตัวต้านทานให้ได้ค่าความต้านทานที่เหมาะสม เพื่อให้สามารถขับแบ่งแยกระดับแสงสีของ LED ที่มีการขับกระแสในปริมาณที่ต่างกันออกให้เห็นความแตกต่างได้

ระบบนี้ได้ออกแบบให้ความเข้มของแสงสีที่แสดงออกทาง LED มี 16 ระดับ ซึ่งการควบคุมนี้อาศัยการควบคุมปริมาณกระแสที่ไหลผ่าน LED ทั้งสีแดงและสีเขียว พิจารณาวงจรส่วนแปลงข้อมูลดิจิตอลเป็นอนาลอก และขับกระแสสามารถหาค่ากระแสในรูปของ PULSE ที่ไหลผ่าน LED นี้ได้จากการวัดสัญญาณไฟฟ้าที่ตกคร่อมตัวต้านทาน 5.6 โอห์ม ซึ่งกระแสที่ผ่าน LED ต่างแถวจะมีความแตกต่างกันตามข้อมูลของภาพที่ออกแบบไว้ ถ้าให้ในหนึ่งแถวมี LED ติดสว่างเพียงดวงเดียว (แสงสีใดก็ได้) สามารถวัดลักษณะสัญญาณได้ดังนี้



ระยะเวลา 20ms นี้จะเป็นเวลาที่ใช้ในการ สแกน LED แนวหลัก ครบ 16 หลัก คือ สแกนครบ 1 ภาพ ซึ่งระยะเวลาในการสแกน 1 ภาพนี้จะสัมพันธ์กับการมองเห็นภาพ ถ้าระยะเวลายาวนานเกินไป จะทำให้เห็นภาพกระพริบ ในที่นี้ได้ออกแบบให้มีค่าความถี่ในการสแกนภาพที่เหมาะสม คือ 50 Hz ซึ่งผลที่ได้ก็เป็นไปตามต้องการ

ค่าความต่างศักย์ V ที่วัดได้นี้จะขึ้นอยู่กับข้อมูลอินพุทของส่วนแปลงข้อมูลดิจิตอลเป็นอนาลอกว่าจะกำหนดระดับความเข้มของแสงสีในระดับใด สามารถวัด และคำนวณหาค่ากระแสที่ไหลผ่าน LED โดยประมาณได้ดังนี้

ข้อมูลเข้า D/A	ความต่างศักย์คร่อม R 5.6 โอห์ม	กระแสสัญญาณ PULSE	กระแสเฉลี่ย
00	0 V	0 A	0 A
01	0.13 V	23.2 mA	1.28 mA
10	0.26 V	46.4 mA	2.55 mA
11	0.39 V	69.6 mA	3.83 mA

นอกจากนี้ เพื่อให้ได้ผลของการแสดงภาพที่ดีที่สุด คือให้เกิดการพริ้วของภาพที่แสดงทาง LED น้อยที่สุด จึงต้องมีการทดลองโดยทำการเปลี่ยนแปลงค่าเวลาหน่วงที่ใช้ในการติด หรือการทำงานของ MOSFET ในขณะที่เลือกให้ LED หลักใดหลักหนึ่งติด โดยการแก้ไขค่าเวลาหน่วงในโปรแกรมส่วนที่ควบคุม 8031 จนได้การแสดงภาพที่ดี

บทที่ 8

สรุป

ในการทำวิทยานิพนธ์เรื่อง "การแสดงภาพด้วย LED 2 สีต่างระดับ" นี้จะเห็นได้ว่าผลที่ได้เป็นไปตามวัตถุประสงค์ที่ได้ตั้งไว้ คือ สามารถที่จะแสดงภาพในรูปแบบที่ต้องการออกทางแผง LED ได้ โดยอาศัยการควบคุมต่าง ๆ ผ่านทางไมโครคอมพิวเตอร์ ซึ่งข้อมูลของภาพที่ต้องการแสดง และข้อมูลควบคุมรูปแบบการแสดงจะถูกส่งไปยังส่วนควบคุมระบบ เพื่อให้ไมโครคอนโทรเลอร์ทำการควบคุมระบบและแสดงภาพตามรูปแบบที่ต้องการ

แนวทางการพัฒนา

1. กรณีที่ต้องการให้รูปแบบของการแสดงภาพมีความหลากหลายมากขึ้น เช่น ให้มีการแสดงภาพโดยมีการเลื่อนภาพไปทางขวา เลื่อนภาพขึ้น หรือเลื่อนภาพลง ก็สามารถทำได้โดยการพัฒนาซอฟท์แวร์ที่ใช้ควบคุมระบบ
2. ในการพิจารณาว่าต้องมีการขยายฮาร์ดแวร์ส่วนควบคุมระบบหรือไม่ เมื่อต้องการที่จะขยายขนาดของ LED ให้มีขนาดใหญ่ขึ้นนั้น ขึ้นอยู่กับว่าต้องการขยายขนาดของแผง LED ให้ใหญ่มากน้องเพียงใด ถ้าหากต้องการขยายแผง LED ให้ใหญ่ขึ้นมากนั้นจำเป็นต้องมีการขยายฮาร์ดแวร์ของส่วนควบคุมระบบ เพื่อให้ได้ผลของการแสดงภาพที่ดี เป็นการลดการพริ้วของภาพลง

ภาคผนวก ก

การใช้งานส่วนซอฟท์แวร์

ส่วนซอฟท์แวร์ของโครงงานนี้ เมื่อเรียกใช้งานจะมีการแสดงผลบนจอภาพดังรูปในหน้าถัดไป

<u>ฟังก์ชั่นการใช้งาน</u>

1. โหมดการพิมพ์ (Edit Mode)

0,1,2,...,F	: เลือกสีที่ต้องการแสดงของ LED (0-15)
space	: ให้จุดที่เคอร์เซอร์อยู่นั้นเป็นสีที่เลือกไว้
←,↑,→,↓	: เคลื่อนเคอร์เซอร์ไปยังทิศทางที่ต้องการ
F_2	: เก็บข้อมูลภาพลงแฟ้ม
F_3	: เรียกข้อมูลภาพจากแฟ้ม
F_4	: ลบข้อมูลภาพที่อยู่บนจอ
ESC	: เปลี่ยนไปเป็น Common Mode

2. โหมดคำสั่ง (Command Mode)

CLS	: ลบข้อมูลภาพที่อยู่บนจอ
SETCOLOR X	: เลือกสีที่ต้องการ (X = 0-15)
PUTPIXEL X,Y	: กำหนดข้อมูลสีลงจุดที่ต้องการ
LINE X_1,Y_1,X_2,Y_2	: ลากเส้นตามตำแหน่งที่ต้องการ
BAR X_1,Y_1,X_2,Y_2	: ระบายข้อมูลภาพลงเป็นแถบสี่เหลี่ยม
RECTANGLE X_1,Y_1,X_2,Y_2	: ระบายข้อมูลภาพลงเป็นกรอบสี่เหลี่ยม
LOAD	: เรียกข้อมูลภาพจากแฟ้ม
SAVE	: เก็บข้อมูลภาพลงแฟ้ม
ESC	: เปลี่ยนไปเป็นโหมดการพิมพ์

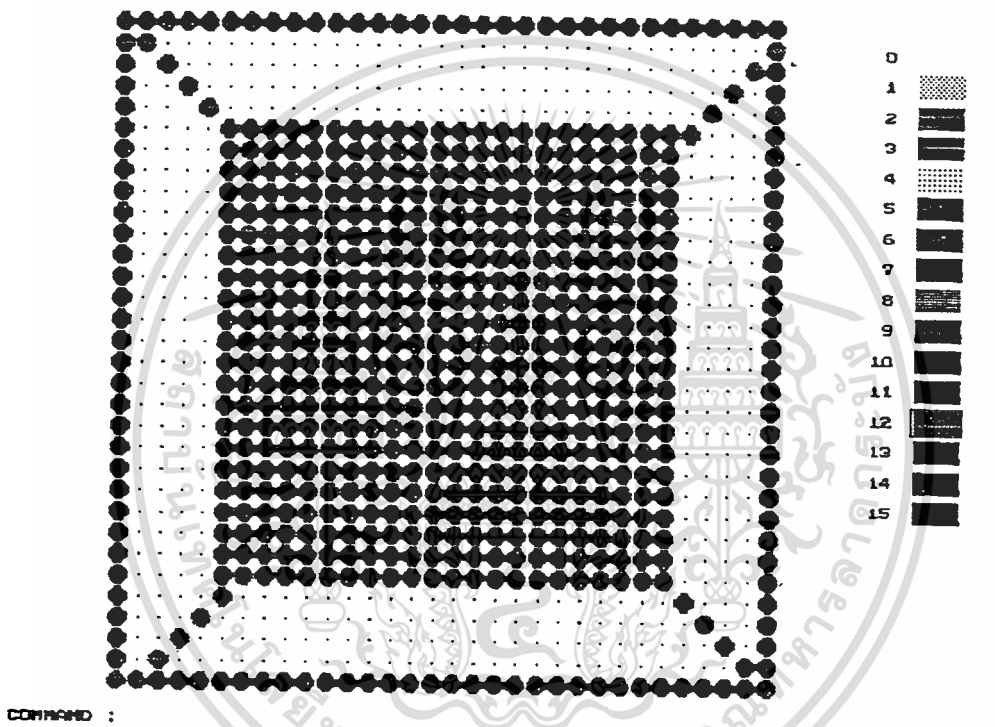
QUIT	: ออกจากโปรแกรม
SENDDATA	: ส่งข้อมูลภาพไปเก็บยังส่วนควบคุมระบบ
DISPLAY NODISPLAY	: หยุดการแสดงภาพ
DISPLAY NORMAL	: แสดงภาพนิ่ง
DISPLAY BLINK	: แสดงภาพกระพริบ
DISPLAY SHIFT	: แสดงภาพเลื่อนซ้าย

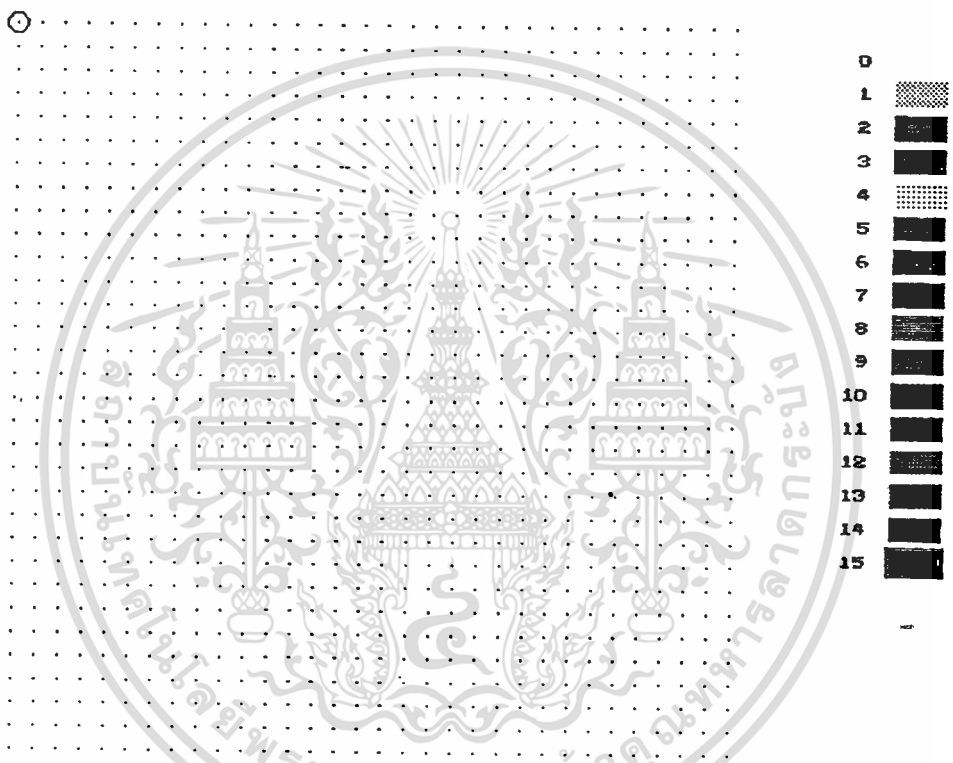
ขั้นตอนการใช้งาน

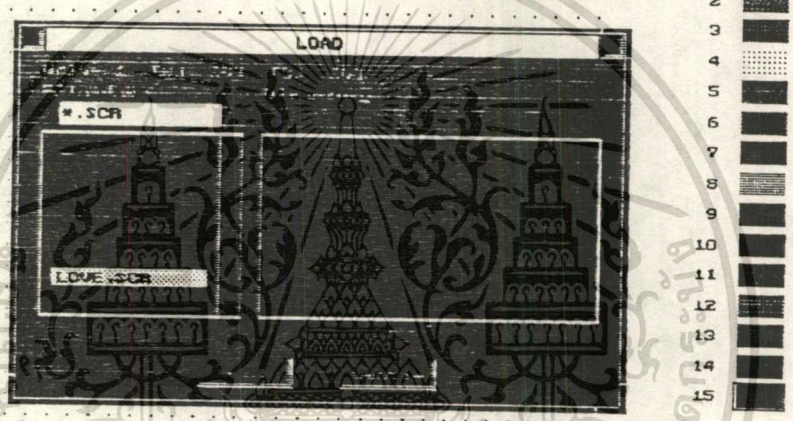
1. ทำการออกแบบภาพที่ต้องการแสดงทาง LED ให้เรียบร้อย หรือเรียกข้อมูลภาพที่ออกแบบไว้แล้วขึ้นมาจากแฟ้ม
2. ทำการส่งข้อมูลภาพที่ต้องการแสดงออกทางพอร์ทอนุกรมไปเก็บยังหน่วยความจำบนส่วนควบคุมระบบ
3. กำหนดและเลือกรูปแบบของการแสดงภาพตามต้องการคือ
   - -แสดงภาพนิ่ง
   - -แสดงภาพกระพริบ
   - -แสดงภาพเลื่อน
   - -หยุดการแสดงภาพ

หมายเหตุ

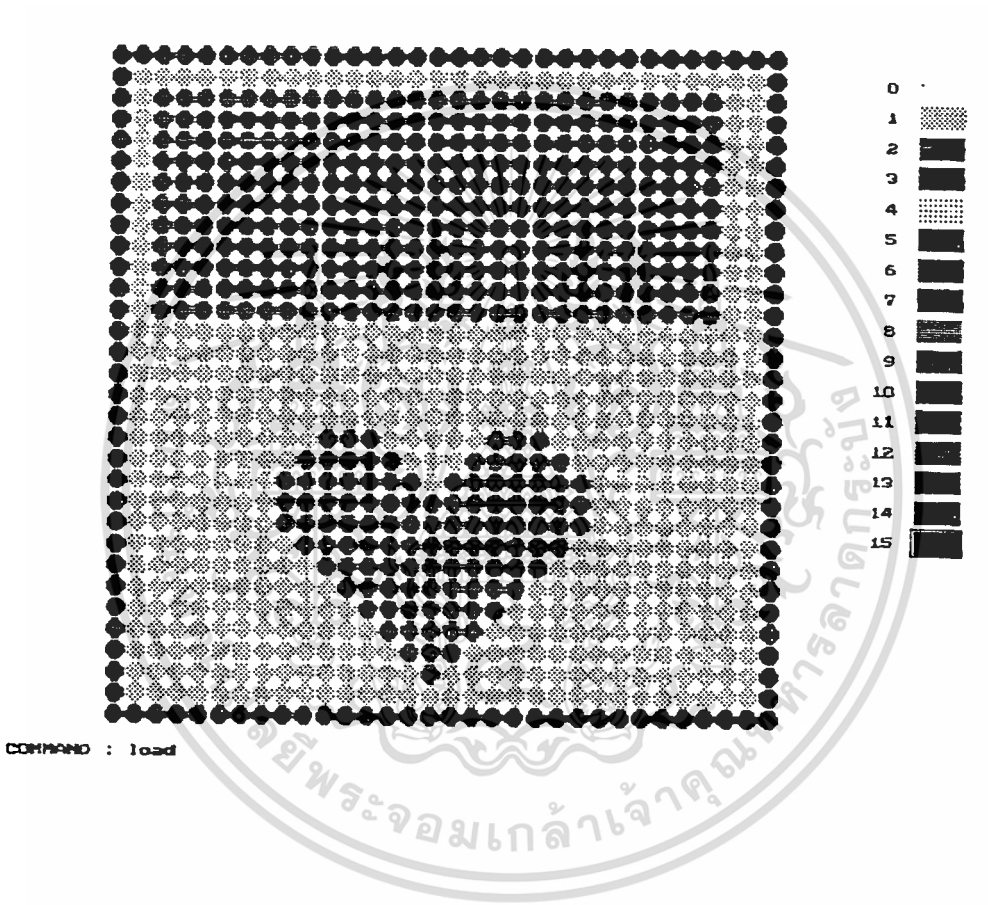
1. การแสดงภาพตามรูปแบบต่าง ๆ จะเป็นไปตามต้องการได้ก็ต่อเมื่อได้มีการส่งข้อมูลของภาพไปก่อนหลังจากรีเซ็ต (Reset)ระบบแล้ว
2. หลังจากที่มีการส่งข้อมูลภาพและสั่งให้แสดงภาพแล้ว และให้สั่งแสดงภาพตามรูปแบบที่กำหนดไว้ของคำสั่งครั้งหลัง

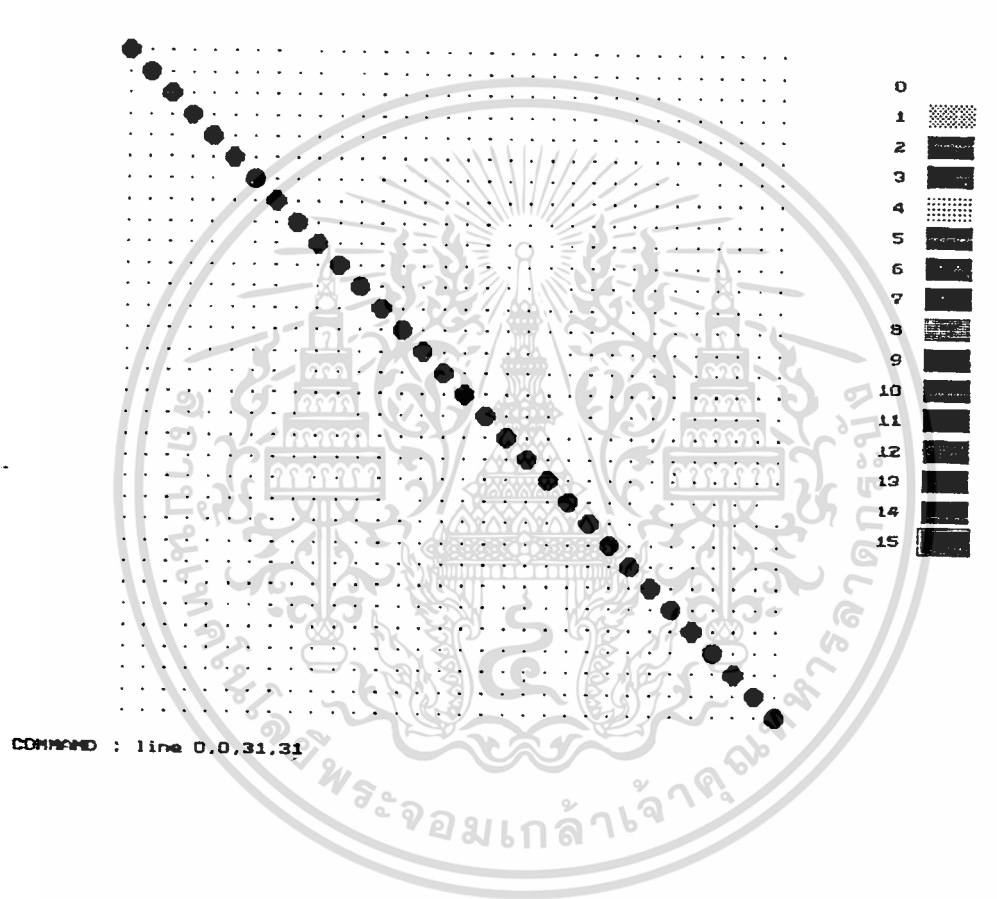


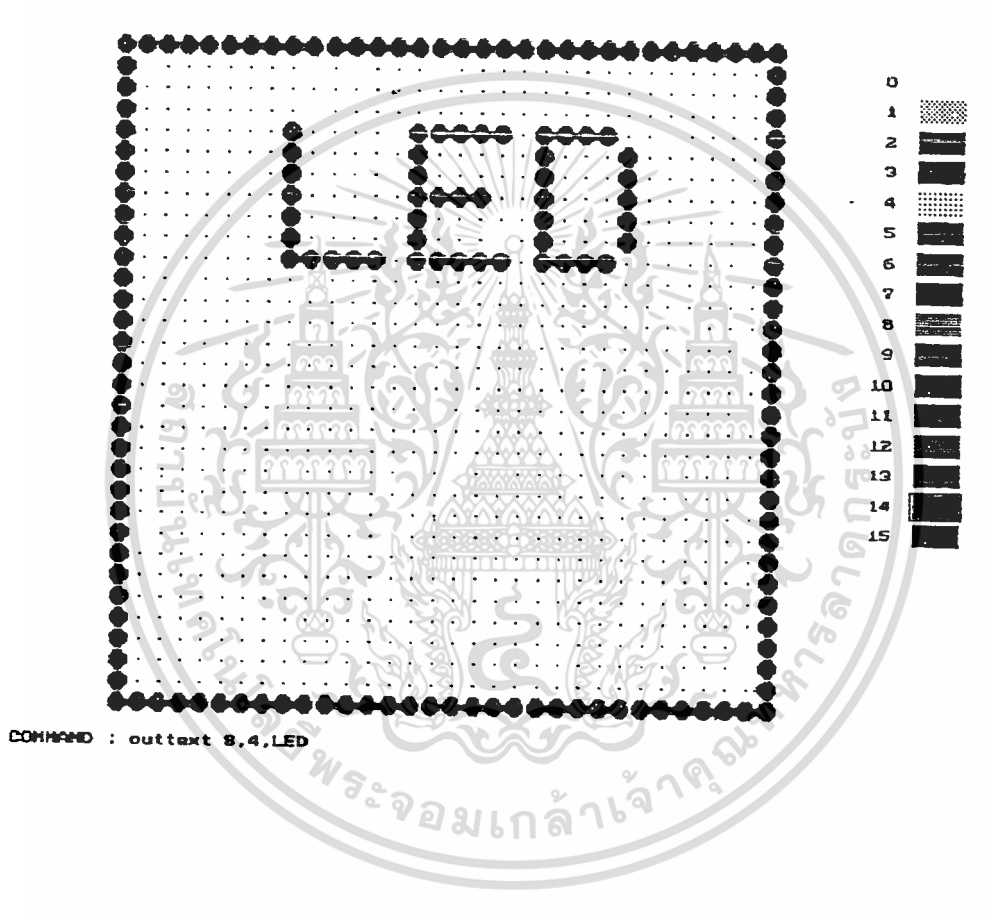




COMMAND : load







บรรณานุกรม

1. ซีเอ็ดยูเคชั่น, "คู่มือ/เทียบเบอร์ ไอซี ทีทีแอล", เอช-เอ็น การพิมพ์,396
2. โครงการตำราวิชาการวิทยาลัยมหานคร, "ไมโครคอนโทรลเลอร์ชิพเดี่ยว 8051"
3. ETT "PC-SB31 USER MANNUAL" กรุงเทพฯ
4. ETT "MCS-51 MICROCONTROLLER" กรุงเทพฯ