ซอร์ฟแวร์สำหรับควบคุมการแสดงผล

Display Board Controller Software

นายนุกูล เอี่ยมองอาจ
นายพัชรัตน์ กอร่ม
นายมนตรี ทับทองหลาง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาอุตสาหกรรมศาสตร์บัณฑิต
สาขาเทคโนโลยีโทรคมนาคม ภาควิชาเทคนิคอุตสาหกรรม
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2543

Display Board Controller Software

Mr. Nukool Aiemongart

Mr. Patcharat Korrom

Mr. Montree Tubthonghlang

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MACHELOR OF THE TECHNOLOGY COMMUNICATION

FACULTY OF ENGINEERING

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2000

หัวข้อปริญญานิพนธ์	ซอร์ฟแวร์สำหรับควบคุมการแสดงผล			
	Display Board Controller Software			
นักศึกษา	นายนุกูล	เอี่ยมองอาจ	เลขประจำตัว	41013370
	นายพัชรัตน์	กอร่ม	เลขประจำตัว	41013373
	นายมนตรี	ทับทองหลาง	เลขประจำตัว	41013378
อาจารย์ที่ปรึกษา	อาจารย์ภุชงค์	หงษ์สุวรรณ		
	อาจารย์สุธีรา	พันธุ์ธีรานุรักษ์		
ภาควิชา	เทคนิคอุตสาหกรรม			
ปีการศึกษา	2543			

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรอุตสาหกรรมศาสตร์บัณฑิต

.....อาจารย์ที่ปรึกษา
(อาจารย์ภุชงค์ หงษ์สุวรรณ)

..... อาจารย์ที่ปรึกษา
(อาจารย์สุธีรา พันธุ์ธีรานุรักษ์)

..... กรรมการ
( )

..... กรรมการ
( )

..... กรรมการ
( )

ลิขสิทธิ์ของคณะวิศวกรรมศาสตร์สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

หัวข้อปริญญานิพนธ์	ซอร์ฟแวร์สำหรับควบคุมการแสดงผล			
	Display Board Controller Software			
นักศึกษา	นายนุกูล	เอี่ยมองอาจ	เลขประจำตัว	41013370
	นายพัชรัตน์	กอร่ม	เลขประจำตัว	41013373
	นายมนตรี	ทับทองหลาง	เลขประจำตัว	41013378
อาจารย์ที่ปรึกษา	อาจารย์ภุชงค์	หงษ์สุวรรณ		
	อาจารย์สุธีรา	พันธุ์ธีรานุรักษ์		
ภาควิชา	เทคนิคอุตสาหกรรม			
ปีการศึกษา	2543			

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ได้นำเสนอการประยุกต์การนำคอมพิวเตอร์ส่วนบุคคล (Personal Computer) มาใช้ควบคุมการแสดงผลเป็นตัวอักษรภาษาไทยหรือตัวอักษรภาษาอังกฤษ โดยตัวอักษรจะมีรูปแบบการวิ่งจากซ้ายไปขวา ขวาไปซ้าย บนลงล่าง และล่างขึ้นบน และยังสามารถแสดงผลเป็นสีแดงหรือเขียวได้ สำหรับการเปลี่ยนข้อความและรูปแบบนั้นจะทำการเปลี่ยนแปลงที่คอมพิวเตอร์เพียงอย่างเดียว

Thesis Title	Display Board Controller Software			
Student	Mr.Nukool	Aiemongart	ID	41013370
	Mr.Patcharat	Korrom	ID	41013373
	Mr.Montree	Tubthonghlang	ID	41013378
Advisor	Mr.Puchong	Hongsuwan		
	Miss.Suteera	Panteranurak		
Academic Year	2000			

ABSTRACT

This thesis presents the application of controlling LED-matrix display by personal computer. The Led-matrix display can be shown to Thai or English letter which they move from left to right, right to left, up to down and down to up. It can selected color to shown, red or green. Exchanced this form or letter on personal computer.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้อย่างดี ด้วยคำแนะนำและคำปรึกษาเกี่ยวกับการเขียนปริญญานิพนธ์ จาก อาจารย์ภุชงค์ หงษ์สุวรรณ และ อาจารย์สุธีรา พันธุ์ธีรานุรักษ์ ซึ่งเป็นอาจารย์ที่ปรึกษา คณะผู้จัดทำรู้สึกซาบซึ้งในความอนุเคราะห์จากท่าน และขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณ อาจารย์วสันต์ อาจารย์โรงเรียนช่างฝีมือทหารที่ช่วยให้คำแนะนำ และเพื่อน วิเชษฐ์ เจือตี๋ วิศวกรรมคอมพิวเตอร์ ที่ช่วยค้นคว้าหาข้อมูล ซึ่งมีส่วนทำให้คู่มือฉบับนี้สำเร็จลงได้

คุณค่าและประโยชน์อันพึงมีจากคู่มือฉบับนี้ ผู้จัดทำขอมอบแด่ผู้มีพระคุณทุกท่าน

นายนุกูล เอี่ยมองอาจ
นายพัชรัตน์ กอร่ม
นายมนตรี ทับทองหลาง

สารบัญ

	หน้า
บทคัดย่อภาษาไทย	ก
บทคัดย่อภาษาอังกฤษ	ข
กิตติกรรมประกาศ	ค
สารบัญ	ง
สารบัญรูปภาพ	ฉ
สารบัญตาราง	ซ
บทที่ 1 บทนำ	1
1.1 บทนำ	1
1.2 วัตถุประสงค์ของโครงงาน	1
1.3 ขอบเขตของโครงงาน	1
1.4 ส่วนประกอบของโครงงาน	2
บทที่ 2 ทฤษฎี MCS-51	3
2.1 บทนำ	3
2.2 โครงสร้างภายในของ 8051	4
2.3 พอร์ทของ 8051	6
2.4 ผังเวลาของซีพียู	10
2.5 การต่อหน่วยความจำ Program Memory และ Data Memory	12
2.6 การแบ่งประเภทของหน่วยความจำ	12
บทที่ 3 8255 Programmable Peripheral Intergface	28
3.1 บทนำ	28
3.2 คุณสมบัติขาสัญญาณต่างๆของ 8255	29
3.3 ลักษณะการจัด Control Word โหมด 0	31
บทที่ 4 การอินเตอร์เฟส MCS-51 กับ 8255 PPI	33
4.1 กล่าวนำ	33
4.2 การอินเตอร์เฟส MCS-51 กับ 8255 โดยใช้ EPROM ภายนอก	33
4.3 การอินเตอร์เฟส MCS-51 กับ 8255 โดยใช้ EPROM ภายใน	38
บทที่ 5 มาตรฐาน RS-232	42
5.1 กล่าวนำ	42
5.2 RS-232	42
5.3 รายละเอียดของขา DB-9	43

5.3 รายละเอียดของขา DB-9 43
บทที่ 6 การแสดงผลของ Matrix LED 44
6.1 โครงสร้างของ Matrix LED 44
6.2 การแสดงผล 45
6.3 รูปแบบการแสดงผล 50
บทที่ 7 ชุดคำสั่งของ MCS-51 52
7.1 รหัสคำสั่งของ MCS-51 52
7.2 การแบ่งกลุ่มคำสั่งของ MCS-51 53
บทที่ 8 การทดลองและข้อกำหนดการใช้งาน 61
8.1 ลักษณะรูปร่างของชิ้นงาน 61
8.2 ส่วนของ Supply 61
8.3 ชุดไมโครคอนโทรลเลอร์ MCS-51 63
8.4 แผงแสดงผล 63
8.5 หน้าที่และการทำงานของส่วนประกอบต่างๆ 64
8.6 การใช้งาน 64
8.7 Flow Chart 67
บทที่ 9 สรุปผลการทดลอง ปัญหาและข้อเสนอแนะ 72
บรรณานุกรม 73
ภาคผนวก 74
ภาคผนวก ก. รูปวงจร 75
ภาคผนวก ข. โปรแกรม 81
ภาคผนวก ค. แสดงขาอุปกรณ์และชุดคำสั่งพื้นฐาน 125

สารบัญรูป

รูปที่	หน้า
รูปที่ 1.1 Block Diagram	2
รูปที่ 2.1 (a) บล็อกไดอะแกรมของ MCS-51	4
รูปที่ 2.1 (b) ตำแหน่งของรีจีสเตอร์ต่างๆและหน่วยความจำภายใน	5
รูปที่ 2.2 การจัดวางขาของ 8051	5
รูปที่ 2.3 แสดงโครงสร้างพอร์ท 0 (บิต)	6
รูปที่ 2.4 โครงสร้างของพอร์ท 1 (บิต)	7
รูปที่ 2.5 โครงสร้างของพอร์ท 2 (บิต)	7
รูปที่ 2.6 โครงสร้างของพอร์ท 3 (บิต)	8
รูปที่ 2.7 การต่อขารีเซ็ทให้กับ 8051	9
รูปที่ 2.8 ผังเวลาการทำงานของแต่ละคำสั่ง	10
รูปที่ 2.9 แสดงผังเวลาการติดต่อกับหน่วยความจำภายนอก	11
รูปที่ 2.10 การต่อหน่วยความจำโปรแกรม และหน่วยความจำข้อมูลภายนอกชิป	12
รูปที่ 2.11 ผังหน่วยความจำสำหรับเก็บโปรแกรมสำหรับเบอร์ 8051	13
รูปที่ 2.12 ผังหน่วยความจำสำหรับเก็บโปรแกรมสำหรับเบอร์ 8052	13
รูปที่ 2.13 ผังหน่วยความจำสำหรับ Data Memory เบอร์ 8051	13
รูปที่ 2.14 ผังหน่วยความจำสำหรับ Program Memory ของ 8052	14
รูปที่ 2.15 ผังการทำงานของไทม์เมอร์/เคาน์เตอร์1(โหมด0)13-bit และรีจีสเตอร์ควบคุม	16
รูปที่ 2.16 รายละเอียดของIP	25
รูปที่ 3.1 แสดงลักษณะการจัดขา 8255 PPI	28
รูปที่ 3.2 แสดงลักษณะการจัด Control Word	30
รูปที่ 3.3 แสดงลักษณะการจัด Control Word Mode 0	31
รูปที่ 4.1 แสดงการอินเตอร์เฟส MCU,8255PPI, EPROM ภายนอก	34
รูปที่ 4.2 แสดงการประยุกต์ใช้ P2.5, P2.6 ควบคุม A0, A1 ของ 8255 โดยตรง	35
รูปที 4.3 แสดงการอินเตอร์เฟส MCU, 8255, EPROM ภายในรูปแบบที่ 1	38
รูปที่ 4.4 แสดงการอินเตอร์เฟส MCU, 8255, EPROM ภายในรูปแบบที่ 2	39
รูปที่ 4.5 แสดงการอินเตอร์เฟส MCU, 8255, EPROM ภายในรูปแบบที่ 3	40
รูปที่ 5.1 แสดงตำแหน่งของตัวเชื่อมแบบ DB-9	42
รูปที่ 6.1 แสดงการต่อภายใน Matrix LED 1 ตัว	44
รูปที่ 6.2 แสดง Matrix LED 8x8	45

	หน้า
รูปที่ 6.3	46
รูปที่ 6.4	46
รูปที่ 6.5	47
รูปที่ 6.6	48
รูปที่ 6.7	48
รูปที่ 6.8	49
รูปที่ 6.9	49
รูปที่ 6.10	50
รูปที่ 7.1 Program status Word (PSW)	54
รูปที่ 8.1 แสดงส่วนประกอบของชิ้นงานก่อนนำไปประกอบกับคอมพิวเตอร์	61
รูปที่ 8.2 แสดงการตอ่วงจรของ LED ของ Matrix LED ที่ใช้สร้างแผง Display	61
รูปที่ 8.3 Power Supply	62
รูปที่ 8.4 แสดงชุดไมโครคอนโทรลเลอร์ MCS-51 และการต่อ RAM เพิ่ม	63
รูปที่ 8.5 แสดงแผงแสดงผล (Display Board)	63
รูปที่ 8.6 แสดงหน้าต่างสำหรับการเปลี่ยนแปลงการแสดงผล	64
รูปที่ 8.7 แสดงหน้าต่างที่ยังไม่พร้อมที่จะส่งข้อมูล	65
รูปที่ 8.8 แสดงหน้าต่างในการเลือกสีของตัวอักษร	65
รูปที่ 8.9 แสดงหน้าต่างในการเลือกรูปแบบการแสดงผล	65
รูปที่ 8.10 แสดงหน้าต่างการเลือกรูปภาพ	66
รูปที่ 8.11 flow chart แสดงการทำงานของโปรแกรม VISUAL BASIC	67
รูปที่ 8.12 flow chart แสดงการทำงานของโปรแกรมหลักของ MCS-51	68
รูปที่ 8.13 flow chart แสดงการทำงานของโปรแกรมย่อยการแสดงตัวอักษร	69
รูปที่ 8.14 flow chart แสดงโปรแกรมย่อยการแสดงผลรูปภาพ	70
รูปที่ 8.15 flow chart แสดงโปรแกรมย่อยการแสดงรูปแบบการแสดงผล	71

สารบัญตาราง

ตารางที่	หน้า
2.1 MCS-51 family	3
2.2 โหมดการทำงานของไทม์เมอร์ 2	18
2.3 แสดงการทำงานของ SM1 และ SM0 ในโหมดต่างๆ	21
2.4 ตารางการใช้ไทม์เมอร์ 1 กำหนดบอดเรท	22
2.5 อินเตอร์รัปต์เวคเตอร์ของ MCS-51 และลำดับความสำคัญของการอินเตอร์รัปต์	23
2.6 แสดงรายละเอียดของรีจิสเตอร์ IE	24
2.7 Interrupt Vector	26
3.1 ตารางความจริงของ 8255	29
3.2 สรุปการทำงาน 8255 PPI โหมด 0	32
4.1 แสดงการจัด Memory Map ของวงจรที่ 4.1	33
4.2 แสดงการจัด Memory Map ของวงจรที่ 4.2	36
4.3 แสดงการจัด Memory Map ของวงจรที่ 4.3	38
4.4 แสดงการจัด Memory Map ของวงจรที่ 4.4	39
4.5 แสดงการจัด Memory Map ของวงจรที่ 4.5	40
5.1 แสดงรายละเอียดของ ขา DB-9	43
7.1 คำสั่งทางคณิตศาสตร์ของ MCS-51	53
7.2 การกระทำทางลอจิก	56
7.3 การเคลื่อนย้ายข้อมูลภายใน (Internal Data Memory Moves)	57
7.4 การเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำภายนอกชิป	58
7.5 กลุ่มคำสั่งประมวลผลแบบบลูลีน(Boolean Instruction)	58
7.6 กลุ่มคำสั่งควบคุมลำดับการทำงานแบบไม่มีเงื่อนไข (Uncondition Jump)	59
7.7 กลุ่มคำสั่งควบคุมการทำงานแบบมีเงื่อนไข (Condition Jump)	60
7.8 แสดงการทดลองหาค่า Rc ของส่วน Display	62

บทที่ 1
บทนำ

1.1 บทนำ

โครงงานนี้เป็นการประยุกต์นำเอาคอมพิวเตอร์ส่วนบุคคล (Personal Computer) มาควบคุมการแสดงผลสำหรับแผงแสดงผล ในรูปแบบต่างๆ ตามที่กำหนดให้ เนื่องจากในปัจจุบันซอร์ฟแวร์ที่ใช้ควบคุมแผงแสดงผลเป็นแบบที่ไม่สามารถเปลี่ยนแปลงได้โดยตรง ต้องทำการแก้ไขข้อมูลในซอร์ฟแวร์เพื่อเปลี่ยนแปลงการแสดงผล ซอร์ฟแวร์ที่พัฒนาขึ้นจะช่วยในการแก้ไขข้อมูลและการควบคุมการแสดงผลได้ง่ายขึ้น เพื่อให้บุคคลทั่วไปสามารถนำซอร์ฟแวร์สำเร็จรูปไปใช้งานได้ โดยผู้ใช้สามารถป้อนข้อมูลที่จะแสดงผลและรูปแบบการแสดงผลได้จากโปรแกรมโดยตรงจากเครื่องคอมพิวเตอร์

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อศึกษาการทำงานการเชื่อมต่อคอมพิวเตอร์ส่วนบุคคลกับไมโครคอนโทรลเลอร์ (Microcontroller) ได้
2. เพื่อศึกษาโปรแกรมที่สามารถควบคุมอินพุทและเอาท์พุทแล้วส่งข้อมูลไปยังไมโครคอนโทรลเลอร์ได้
3. เพื่อศึกษาการควบคุมตัวอักษรของแผงแสดงผลด้วยไมโครคอนโทรลเลอร์
4. เพื่อศึกษาการออกแบบของแผงแสดงผล

1.3 ขอบเขตของโครงงาน

1. เขียนโปรแกรมการแสดงผลเป็นตัวอักษรภาษาไทยและภาษาอังกฤษ โดยให้มีรูปแบบการแสดงผลดังนี้

- ตัวอักษรวิ่งไปทางซ้าย
- ตัวอักษรวิ่งขึ้น
- ตัวอักษรวิ่งไปทางขวา
- ตัวอักษรวิ่งลง

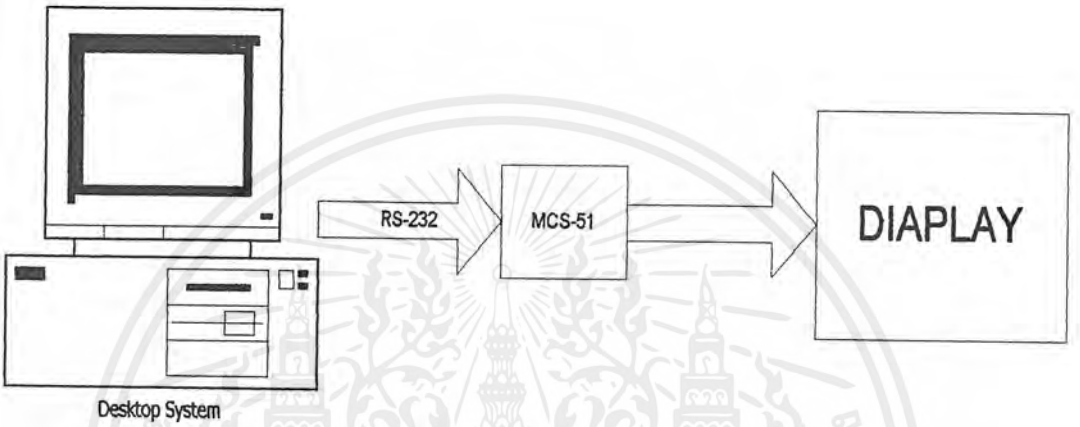
2. สร้างชุดวงจรส่วนของการเชื่อมโยงระบบ (Interface)

มาตรฐาน RS-232 เพื่อทำหน้าที่เป็นส่วนรับ-ส่งข้อมูลและสัญญาณการควบคุมต่างๆ ระหว่างส่วนรับข้อมูลและส่วนควบคุม (Control) กับคอมพิวเตอร์ส่วนบุคคล

3. เขียนโปรแกรมแอสเซมบลี (Assemby) ให้กับไมโครคอนโทรลเลอร์ เบอร์ 8051 เพื่อทำการรับส่งข้อมูลและควบคุม (Control) แผงแสดงผล

4. สร้างชุดแผงแสดงผลโดยใช้ LED แบบ Matrix ขนาด 8x8 ดวง มี 2 สี คือ สีแดงและสีเขียว โดยมีขนาดของแผงแสดงผล 24x64 ดวง

1.4 ส่วนประกอบของโครงงาน



รูปที่ 1.1 Block Diagram

โครงงานนี้จะประกอบไปด้วย 3 ส่วนสำคัญ คือ

1. ชุดควบคุมและแสดงผลด้วยคอมพิวเตอร์(Control and Display)
2. ชุดวงจรไมโครคอนโทรลเลอร์ (Microcontroller)
3. ชุดแผงแสดงผล LED ขนาด 24 x 64 ดวง

บทที่ 2
ทฤษฎี MCS-51

2.1 บทนำ

ไมโครคอนโทรลเลอร์แบบชิพเดี่ยวตระกูล MCS-51 นี้ผลิตโดยบริษัทอินเทลมีอยู่ด้วยกันหลายเบอร์ซึ่งมีรายละเอียดดังตาราง

ตารางที่ 2.1 MCS-51 family

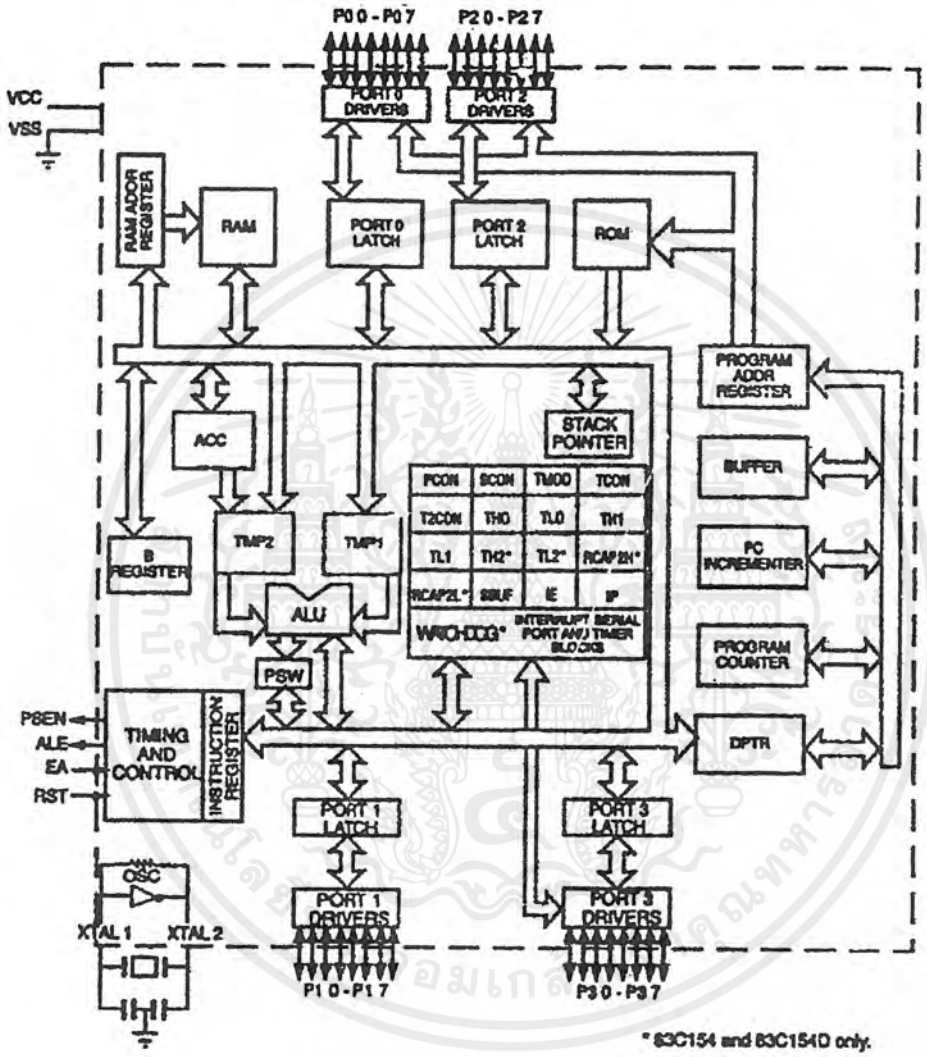
Device	ROM less Version	EPROM Version	ROM	RAM	8-Bit I/O Ports	16-Bit Timer/ Counters	PCA	UART	SEP	GSC	DMA CH	A/D CH	Interrupt Sources/ Vectors	Power Downand Idle Modes
8051	8031	-	4K	128	4	2		/					6/5	
8051AH	8031AH	8751H 8751BH	4K	128	4	2		/					6/5	
8052AH	8032AH	875ZBH	8K	256	4	3		/					8/6	
80C51BH	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	80C51FB	87C51FB	16K	256	4	3	/	/					14/7	/
83C152JA	80C152JA	-	8K	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	/

คุณสมบัติของไมโครคอนโทรลเลอร์ตระกูล MCS-51

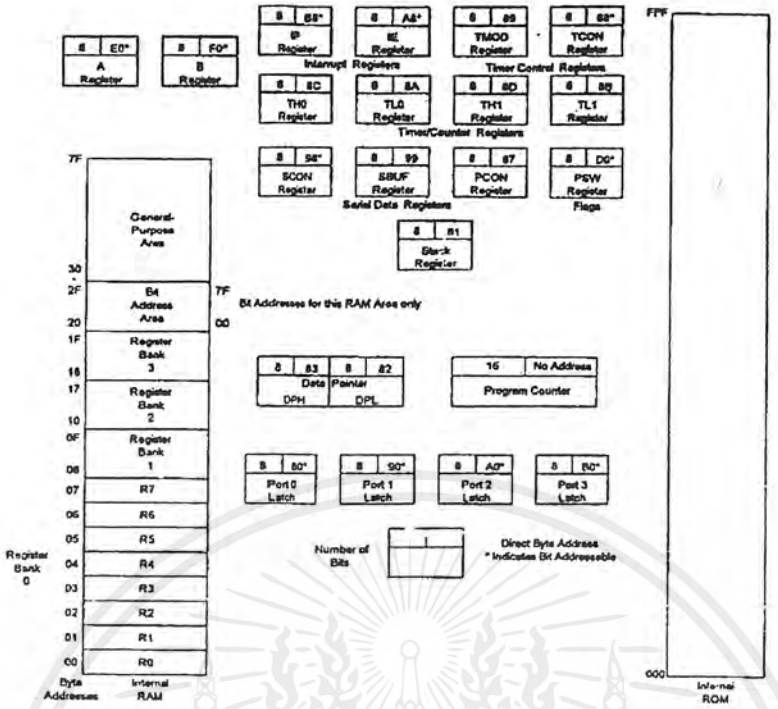
- ต้องการแหล่งจ่ายไฟ + 5V ชุดเดียว
- มีหน่วยความจำโปรแกรม (Program Memory) ขนาด 4 กิโลไบต์สำหรับเบอร์ 8051 และ 8031 สำหรับเบอร์ 8052
- มีหน่วยความจำสำหรับเก็บข้อมูล (Data Memory) ขนาด 128 ไบต์ สำหรับเบอร์ 8052 ขึ้นไปมีถึง 256 ไบต์
- หน่วยความจำสำหรับเก็บโปรแกรมและข้อมูลแยกจากกันอย่างละ 64 กิโลไบต์
- มีไทม์เมอร์เคาน์เตอร์ ขนาด 16 บิต 2 ชุด (สำหรับ 8052 มี 3 ชุด) ทำงานได้ 4 โหมด
- รับอินเตอร์รัพท์ได้ 6 แหล่ง 5 เวคเตอร์ สำหรับเบอร์ 8052 ขึ้นไปมี 8 แหล่ง 6 เวคเตอร์

2.2 โครงสร้างภายในของ 8051

MCS-51 ใช้เทคโนโลยีในการผลิตเป็นแบบ NMOS และ CMOS เบอร์ 8032 และ 8052 จะมี ROM BASIC อยู่ภายในจึงสะดวกสำหรับโปรแกรมเมอร์ที่จะเขียนโปรแกรมด้วยภาษาเบสิก โครงสร้างภายในสำหรับเบอร์ 8051 ดังแสดงในรูป 2.1 (a) และ 2.1 (b)



รูปที่ 2.1(a) 8051 บล็อกไดอะแกรมของ MCS-51



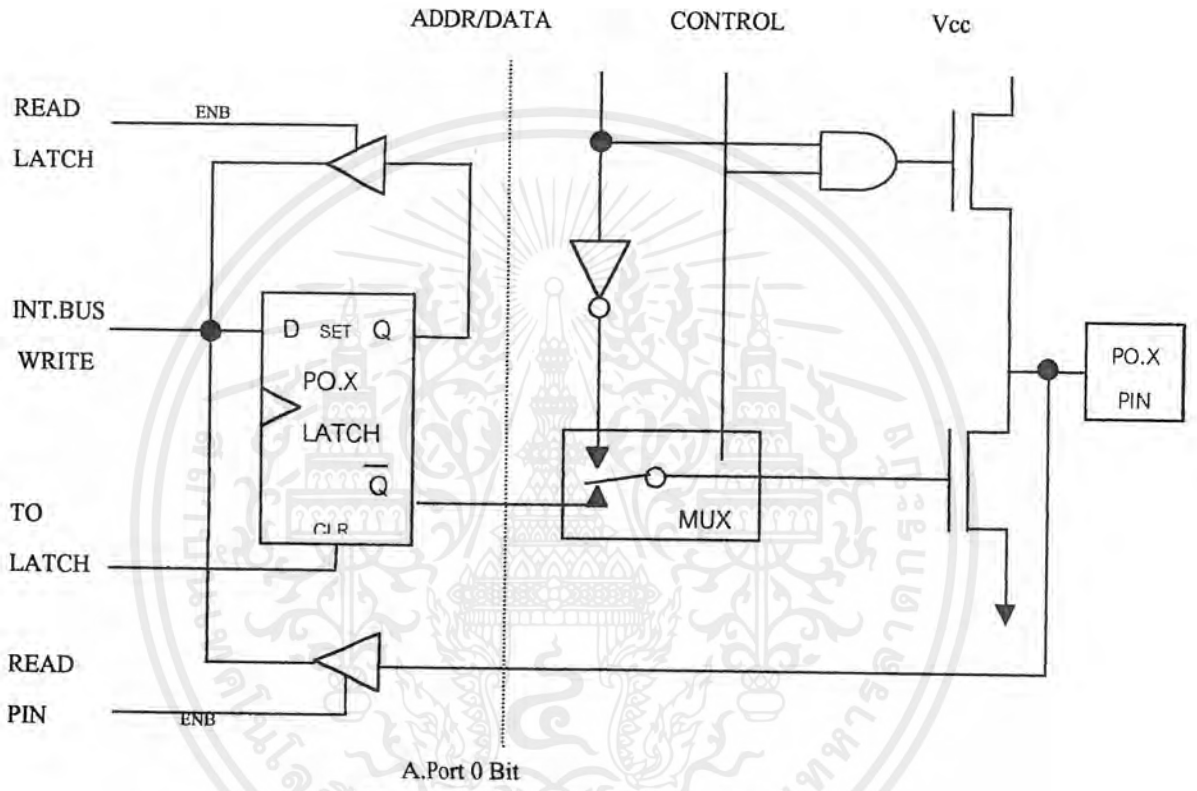
รูปที่ 2.1 (b) ตำแหน่งของรีจีสเตอร์ต่างๆ และหน่วยความจำภายใน

รูปที่ 2.2 การจัดวางขาของ 8051

2.3 พอร์ทของ 8051

8051 เป็นไมโครคอนโทรลเลอร์ขนาด 40 ขา ซึ่งมีขาต่างๆดังนี้

- Vcc (ขา 40) ต่อกับ +5V
- Vss (ขา 20) เป็นขา GND
- พอร์ท 0 (ขา 32-39) มีทั้งหมด 8 บิต คือ (P0.7-P0.0) มีโครงสร้างแบบ Open-Drain Bi-directional ดังแสดงในรูป 2.3



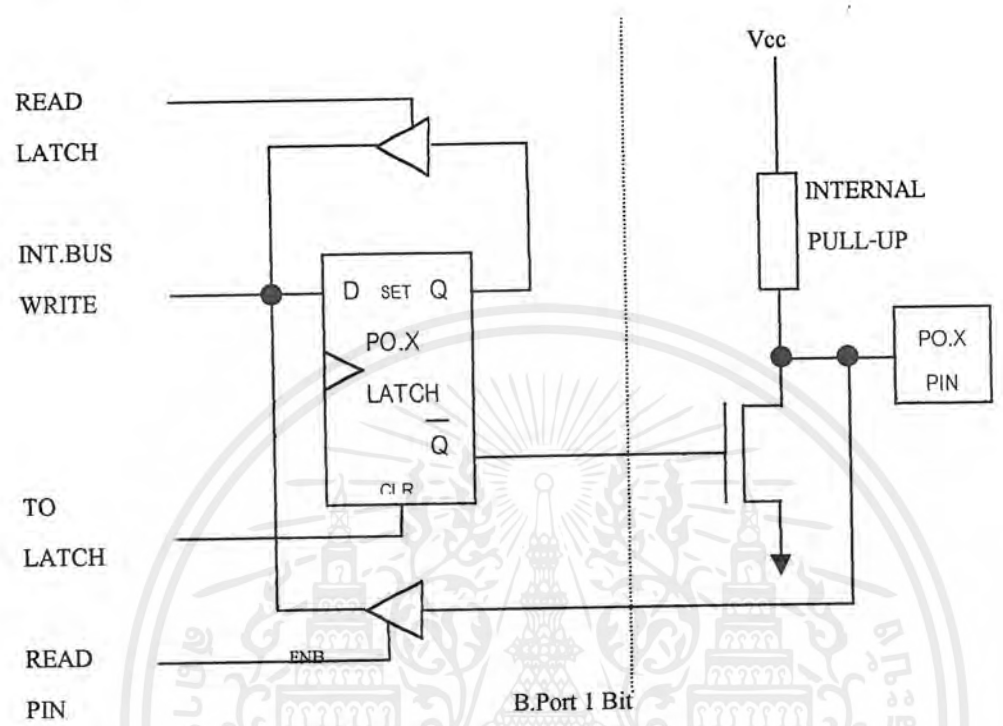
รูปที่ 2.3 แสดงโครงสร้าง พอร์ท 0 (บิต)

- **พอร์ท 0 (ขา 32-39)** มีทั้งหมด 8 บิต คือ (PO.7-PO.0) ใช้งานได้ 2 หน้าที่ คือแอดเดรสบัสและดาต้าบัสเมื่อต้องการติดต่อกับหน่วยความจำภายนอกหรือเป็นไอโอพอร์ท และถ้าต้องการให้ทำงานเป็นอินพุทพอร์ทต้องส่งลอจิก "1" ไปยังพอร์ทนี้ จะมีผลให้ Q ของ D-FF เป็น "0" ทำให้ FET ตัวล่างมีสถานะ OFF สัญญาณที่ใช้อ่านอินพุทพอร์ทแลทช์โดยสัญญาณ READ LATCH ไปกระตุ้นที่ Tri-State Buffer ตัวบนและการ อ่าน Port (pin) จะใช้สัญญาณ Read (pin)

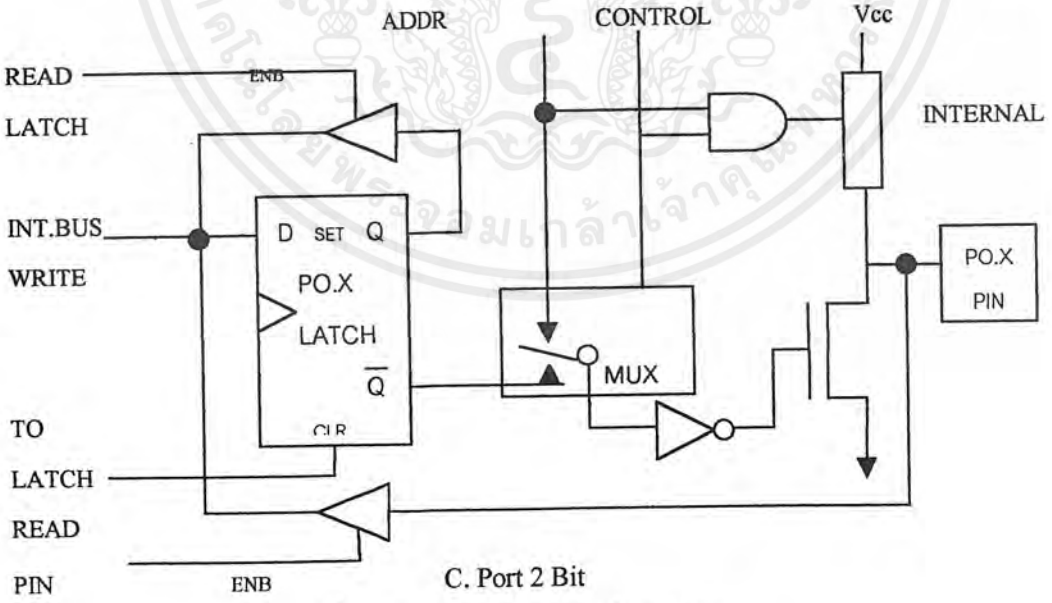
- **พอร์ท 1 (ขา 1-8)** มีทั้งหมด 8 บิต คือ (P1.0-P1.7) มีโครงสร้างคล้าย พอร์ท 0 แต่จะใช้ความต้านทานภายในพลูอัพแทน Internal Pull up Register มีโครงสร้างดังรูปที่ 2.4

- **พอร์ท 2 (ขา 21-28)** มีทั้งหมด 8 บิต คือ ขา (P2.7-P2.0) มีโครงสร้างคล้าย พอร์ท 0 โดยมี FET ตัวล่างตัวเดียวส่วนด้านบนใช้ความต้านทานพูลอัพแทน (Internal pull up) พอร์ทนี้ทำงาน 2

หน้าที่ คือสามารถใช้เป็นแอดเดรสบัสขนาด 8 บิต (A15-A8) และเป็นไอโอพอร์ทใช้งานทั่วไปเมื่อจะใช้งานเป็นอินพุทพอร์ทต้องส่งลอจิก "1" มาที่พอร์ทนี้ก่อนเพื่อบังคับให้ FET อยู่ในสภาวะ OFF ดังแสงในรูปที่ 2.5



รูปที่2.4 โครงสร้างของพอร์ท 1 (บิต)



รูปที่ 2.5 โครงสร้างของ พอร์ท 2 (บิต)

- **พอร์ท 3 (ขา 10-17)** มีทั้งหมด 8 บิต คือ ขา (P3.7-3.0) มีโครงสร้างคล้ายพอร์ท 1 ทำงานได้ 2 หน้าที่คือเป็นไอโอพอร์ทถ้าจะโปรแกรมให้เป็นอินพุตพอร์ทต้องส่งลอจิก "1"มาที่พอร์ทนี้ก่อนและอีกหน้าที่หนึ่งก็คือใช้ส่งสัญญาณควบคุมออกมาและรับสัญญาณเข้าไป สัญญาณต่างๆมีดังนี้

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

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

P3.2/$\overline{\text{INT0}}$ (External Interrupt 0) ใช้รับสัญญาณการขัดจังหวะจากภายนอกเบอร์ 0

P3.3/$\overline{\text{INT1}}$ (External Interrupt 1) ใช้รับสัญญาณการขัดจังหวะจากภายนอกเบอร์ 1

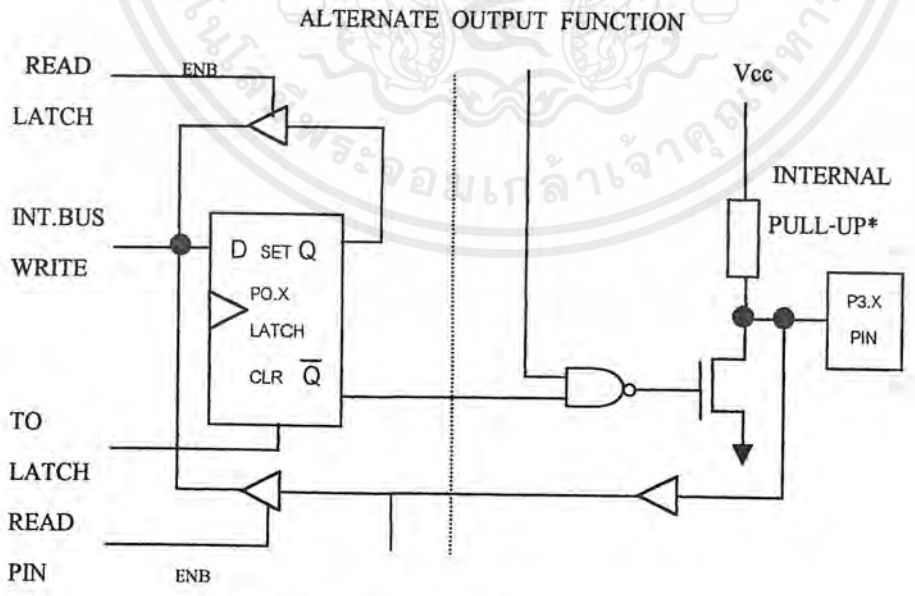
P3.4/T0 (Counter 0 Extrenal Input) ขารับสัญญาณพัลซ์อินพุทเข้าไปยังวงจร Counter 0 (เป็นอินพุทโหมดเคาน์เตอร์)

P3.5/T1 (Counter 1 External Input) ขารับสัญญาณพัลซ์อินพุทเข้าไปยังวงจร Counter 1 (เป็นอินพุทโหมดเคาน์เตอร์)

P3.6/$\overline{\text{WR}}$ (External Data Memory Write Strobe) ขาสัญญาณควบคุมการเขียนข้อมูลข้อมูลลงหน่วยความจำข้อมูลภายนอก

P3.7/$\overline{\text{RD}}$ (External Data Memory Read Strobe) ขาสัญญาณควบคุมการอ่านข้อมูลจากหน่วยความจำข้อมูลภายนอก

โครงสร้างของ (พอร์ท 3) ดังแสดงในรูป 2.6

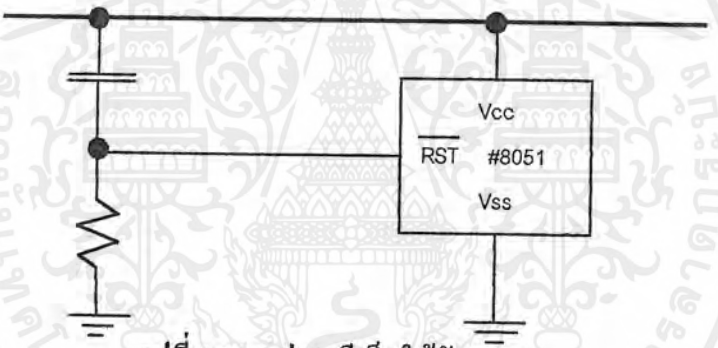


รูปที่ 2.6 โครงสร้างของพอร์ท 3 (บิต)

- **ALE (ขา 30)** เป็นขาส่งสโตรบสำหรับใช้ในการแลตซ์แอดเดรสไบต์ต่ำ (A7-A0) ที่ส่งออกมาจาก(พอร์ท 0)สัญญาณนี้จะแอตทีฟทุกๆ 2 ครั้งใน 1 แมชชีนไซเคิล
- **$\overline{\text{PSEN}}$ (ขา 29)** เป็นขาสโตรปที่ใช้สำหรับอ่านข้อมูลจาก Program Memory ภายนอก สัญญาณนี้จะส่งออกมา 2 ครั้งในแต่ละแมชชีนไซเคิลแต่ถ้าเป็นการอ่าน Internal Program Memory จะไม่มีสัญญาณออกที่ขานี้
- **$\overline{\text{EA}}$ (ขา 30)** ใช้เลือกหน่วยความจำโปรแกรมภายนอก

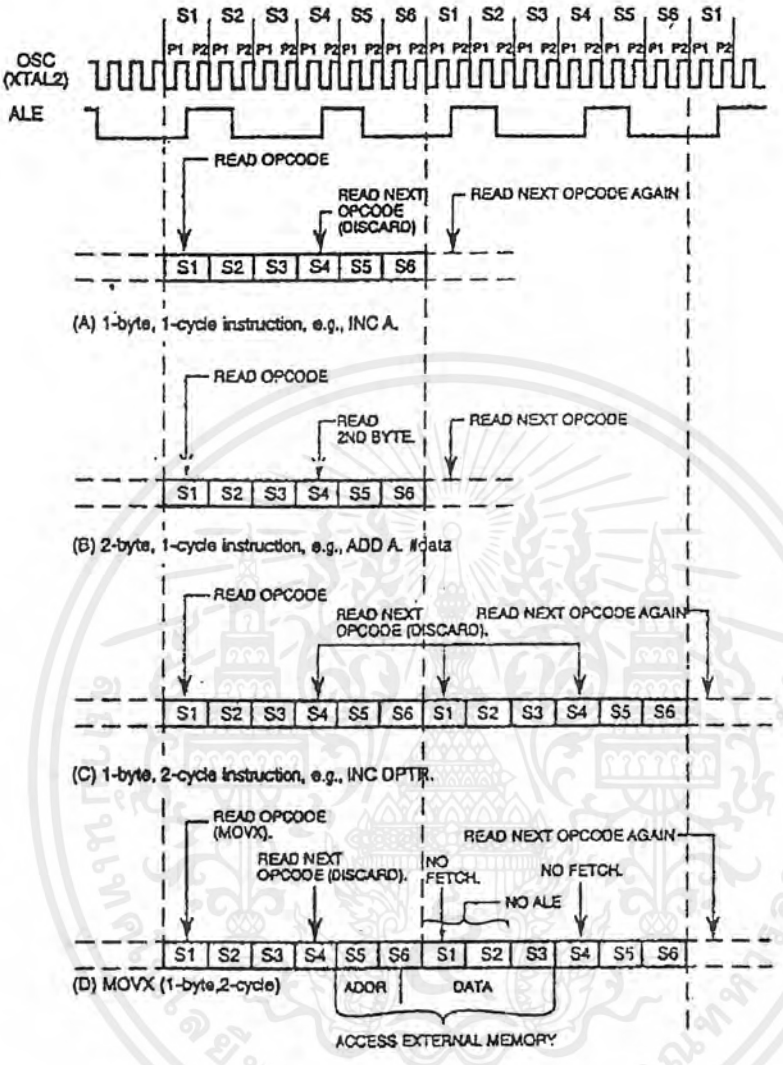
  ป้อน "0" จะอ่านโปรแกรมจากภายนอกชิพ

  ป้อน "1" จะอ่านโปรแกรมจากภายในชิพ
- **RST (ขา 9)** ขารีเซ็ท จะรีเซ็ทได้ก็ต่อเมื่อป้อนลอจิก "1" เข้าที่ขานี้นานอย่างน้อย 2 แมชชีนไซเคิล
- **XTAL1 (ขา 19)** ใช้ต่อคริสตอลภายนอกโดยเป็นอินพุทเข้าสู่วงจรออสซิสเลเตอร์ภายใน
- **XTAL2 (ขา 18)** ใช้ต่อคริสตอลภายนอกโดยเป็นเอาท์พุทของวงจรออสซิสเลเตอร์ภายใน



รูปที่ 2.7 การต่อขารีเซ็ทให้กับ 8051

2.4 ผังเวลาของซีพียู (CPU Timing)



รูปที่ 2.8 ผังเวลาการทำงานของแต่ละคำสั่ง

การทำงานใน 1 คำสั่งต่ำสุดจะกินเวลาเพียง 1 uS เช่น คำสั่ง INC A ซึ่งเป็นคำสั่ง 1 ไบต์ 1 แมชชีนไซเคิล ซึ่งจะใช้คล็อคไปเท่ากับ 12 ลูก โดยคล็อคลูกที่ 1 และ 2 จะอยู่ในช่วง S1 P1 และ S1 P2 และคล็อคลูกที่ 12 ก็จะอยู่ในช่วง S6P2 นั่นเอง (ปรกติแล้ว ซีพียูจะ RUN ด้วยความเร็วเท่ากับ 12 MHz ดังนั้น คล็อค 12 ลูกจะกินเวลาเท่ากับ 1 uS

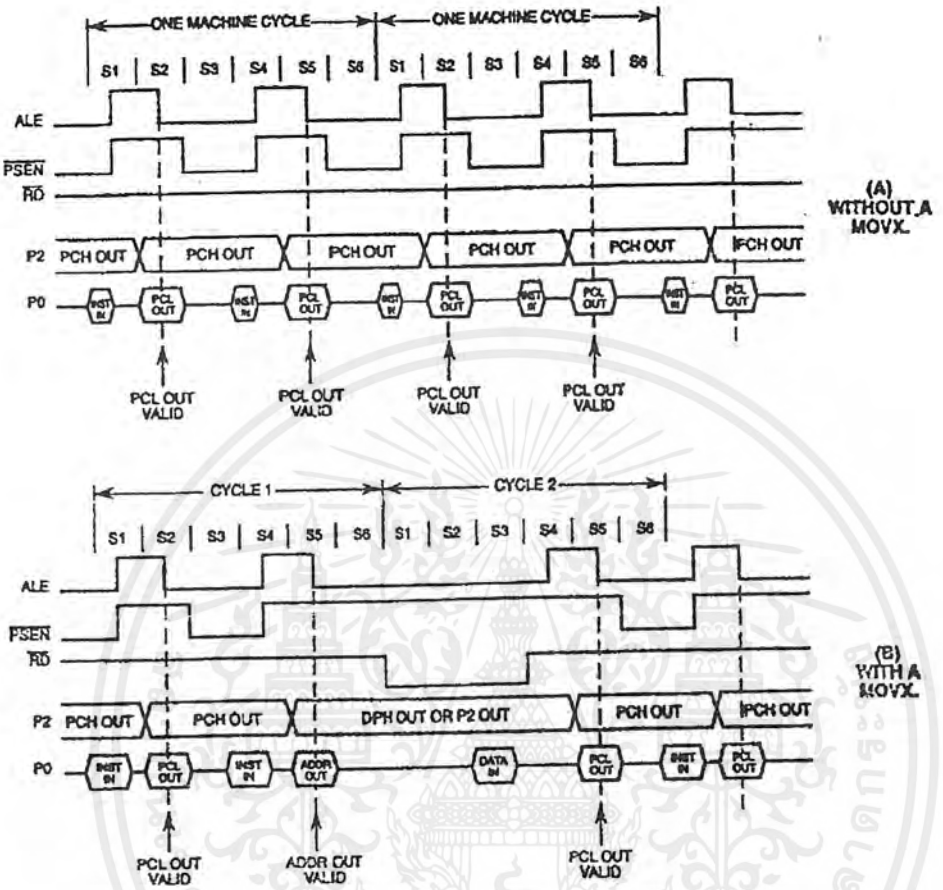
*** คำว่า 1 แมชชีนไซเคิล คือช่วงการทำงานตั้งแต่ S1 จนถึง S6 ***

รูป 2.8(a) การทำงานของคำสั่ง INC A ซึ่งเป็นคำสั่ง 1 ไบต์ทำงานเสร็จภายใน 1 แมชชีนไซเคิล

รูป 2.8(b) การทำงานของคำสั่ง ADD A.#Data ซึ่งเป็นคำสั่ง 2 ไบต์ทำงานเสร็จใน1แมชชีนไซเคิล

รูป 2.8(c) การทำงานของคำสั่ง INC DPTR ซึ่งเป็นคำสั่ง 1 ไบต์ แต่ทำงานเสร็จใน2แมชชีนไซเคิล

รูป 2.8 (d) การทำงานของคำสั่ง MOVX ซึ่งเป็นคำสั่ง 1 ไบต์ แต่ทำงานเสร็จใน 2 แมชชีนไซเคิล



รูปที่ 2.9 แสดงผังเวลาการติดต่อกับหน่วยความจำภายนอก

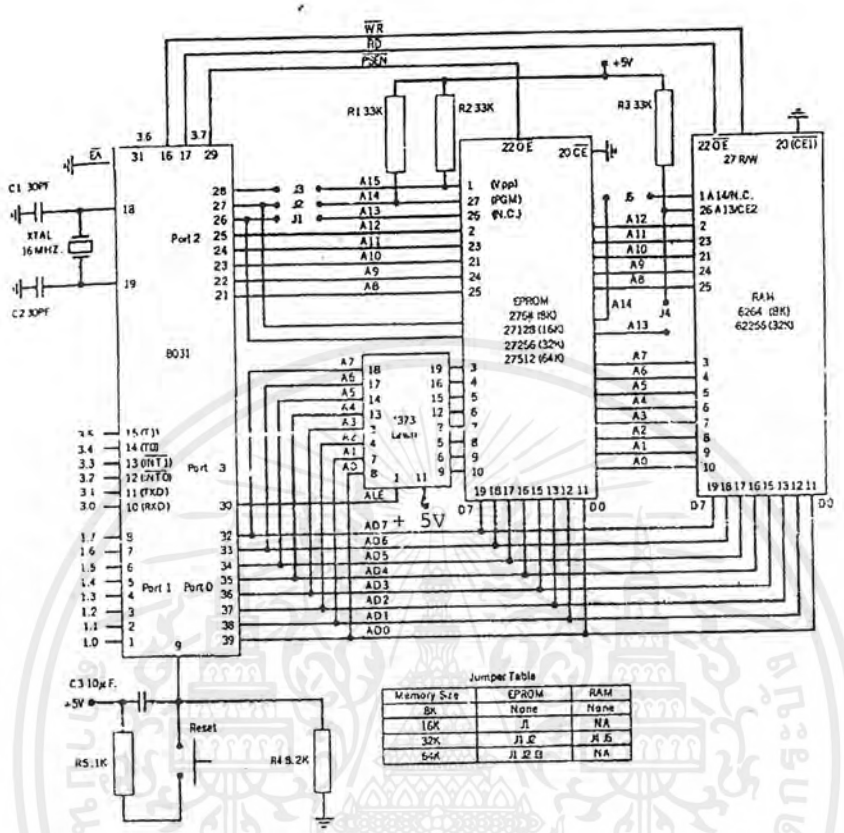
รูป 2.9 (a) เป็นผังเวลาของสัญญาณซึ่งเกี่ยวข้องกับเฟิทซ์เมื่อส่วนของ Program Memory อยู่ภายนอก ดังนั้น สัญญาณที่จะนำไปใช้อ่านออปโค้ด จาก Program Memory ก็คือ $\overline{\text{PSEN}}$ ซึ่งจะแอคทีฟ 2 ครั้งใน 1 แมชชีนไซเคิล ดังนั้น สัญญาณที่ใช้อ่านข้อมูลจาก Program Memory จะใช้สัญญาณ $\overline{\text{PSEN}}$

รูป 2.9 (b) เป็นผังเวลาของสัญญาณที่ใช้การอ่านข้อมูลจาก Data Memory สัญญาณ $\overline{\text{PSEN}}$ จะมีเพียง 1 ลูก เพราะช่วงเวลาถัดมาจะเป็นช่วงเวลาในการอ่านข้อมูลจาก Data Memory โดยใช้สัญญาณ $\overline{\text{RD}}$

(การอ่านข้อมูลจาก Program Memory จะใช้สัญญาณ $\overline{\text{PSEN}}$ และการอ่านข้อมูลจาก Data Memory จะใช้สัญญาณ $\overline{\text{RD}}$ ส่วนสัญญาณ ALE คือ สัญญาณที่ใช้ในการ Latch Address A7-A0 นั่นเอง)

2.5 การต่อหน่วยความจำ Program Memory และ Data Memory

การต่อหน่วยความนำดังแสดงในรูป 2.10

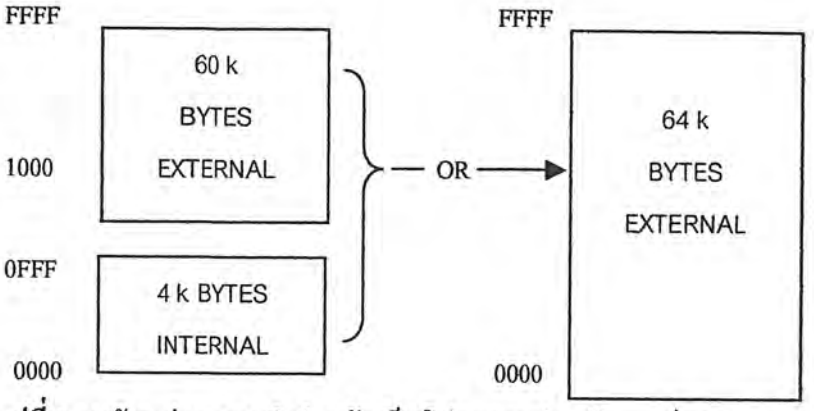


รูปที่ 2.10 การต่อหน่วยความจำโปรแกรม และหน่วยความจำข้อมูลภายนอกชิพ

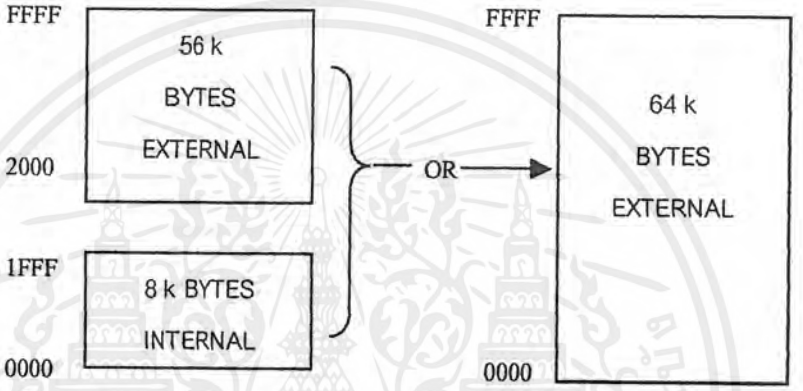
2.6 การแบ่งประเภทของหน่วยความจำ

2.6.1 หน่วยความจำที่ใช้กับ MCS-51 มีอยู่ด้วยกัน 2 ชนิด คือ

2.6.1.1 หน่วยความจำสำหรับเก็บโปรแกรม (Program Memory) เป็นหน่วยความจำที่ใช้เก็บโปรแกรมสั่งงานบรรจุอยู่ในชิพ 8051 ส่วนที่เป็น Program Memory ก็คือ ROM ขนาด 4 กิโลไบต์นั่นเอง แต่ถ้าเป็นเบอร์ 8052 จะมี ROM ขนาด 8 กิโลไบต์ ดังแสดงในรูป 2.11 และ 2.12

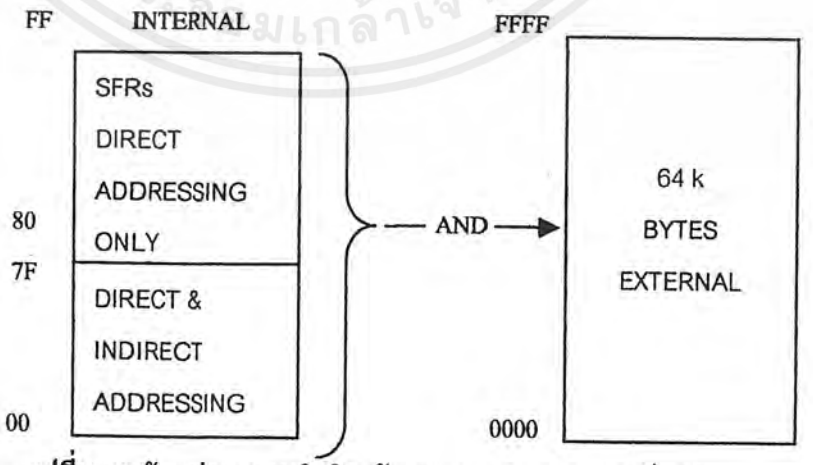


รูปที่ 2.11 ผังหน่วยความจำสำหรับเก็บโปรแกรมสำหรับเบอร์ 8051

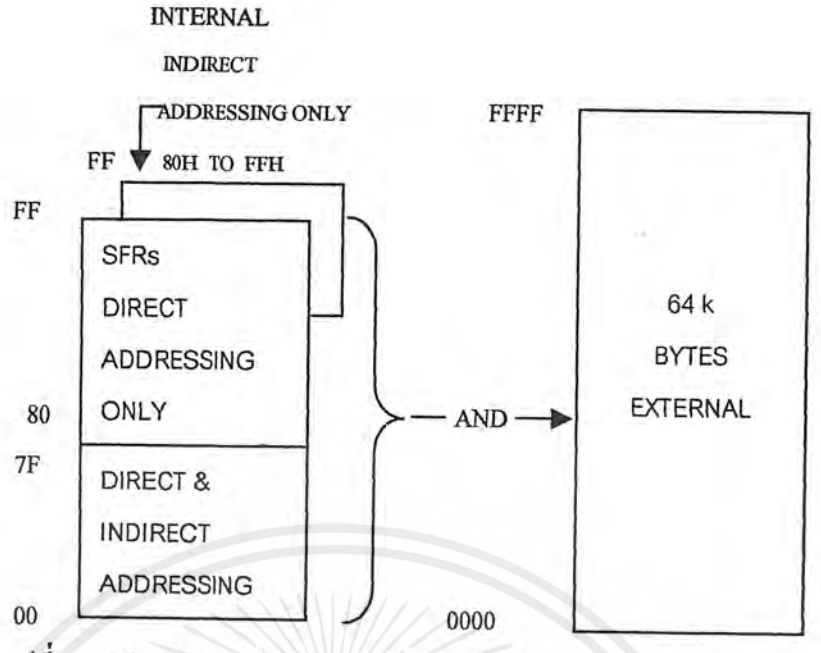


รูปที่ 2.12 ผังหน่วยความจำสำหรับเก็บโปรแกรมสำหรับเบอร์ 8052

2.6.1.2 Data Memory (RAM) แบ่งเป็น 2 ส่วนคือ หน่วยความจำข้อมูลภายในชิพมีเพียง 128 ไบต์ สำหรับเบอร์ 8051 มีจำนวน 256 ไบต์ สำหรับเบอร์ 8052 ขึ้นไปและหน่วยความจำข้อมูลภายนอกชิพมีความจุ 64 กิโลไบต์ ดังแสดงในรูปที่ 2.14 และ 2.15



รูปที่ 2.13 ผังหน่วยความจำสำหรับ Data Memory เบอร์ 8051



รูปที่ 2.14 ผังหน่วยความจำสำหรับ Program Memory ของ 8052

บางครั้งอาจจะสงสัยว่าตำแหน่งของหน่วยความจำสำหรับโปรแกรมและดาต้ามีตำแหน่งที่ซ้อนกันซีพียูจะรู้ได้อย่างไรว่าติดต่อกับหน่วยความจำโปรแกรมและหน่วยความจำข้อมูล บริษัทอินเทลได้ออกแบบแยกคำสั่งออกเป็น 3 ส่วน คือ

MOV ใช้ติดต่อกับ RAM ภายใน

MOVC ใช้ติดต่อ Program Memory

MOVX ใช้ติดต่อ Data Memory ภายนอกชิพ โดยระบุตำแหน่งผ่าน DPTR และ PC

* ชิพเบอร์ 8052 จะมีพื้นที่บริเวณ 80h-FFh ซึ่งถ้าจะเขียนอ่านข้อมูล ณ บริเวณนี้จะเข้าถึงข้อมูลโดยอ้อมเท่านั้น ดังผังหน่วยความจำดังรูป 2.15 *

2.6.2 พื้นที่หน่วยความจำที่เข้าถึงข้อมูลโดยทางอ้อมเท่านั้น (Indirect Address Area)

พื้นที่หน่วยความจำบริเวณ (80h - FFh) ตามรูป 2.15 เป็นพื้นที่ซ้อนกันอยู่อย่างละ 128 ไบต์ โดยส่วนแรกจะเป็น SFR แอดเดรสและ Indirect Address Area ดังนั้นผู้เขียนโปรแกรมถ้าจะติดต่อกับ SFR จะต้องใช้คำสั่งแบบเข้าถึงข้อมูลโดยตรงเท่านั้น (Direct Address Area) ส่วนพื้นที่อีกส่วนหนึ่งจะเข้าถึงข้อมูลแบบทางอ้อมเท่านั้น (Indirect Address Area) ส่วนตำแหน่ง (00h – 7Fh) จะเข้าถึงข้อมูลได้ทั้ง 2 แบบ

2.6.3 พื้นที่หน่วยความจำที่เข้าถึงข้อมูลโดยตรงและทางอ้อม (Direct and Indirect Address Area)

2.6.3.1 รีจิสเตอร์ แบงค์ (Register Banks 0-3)

ตั้งแต่ตำแหน่ง (00h-1Fh) จะเป็นส่วนของรีจิสเตอร์แบงค์ (0-3) โดยแบ่งเป็นแบงค์ละ 8 ไบต์รวมแล้วได้ 32 ไบต์ (แต่ละแบงค์จะมีรีจิสเตอร์ R0, R1, R2, R3, R4, R5, R6, R7) ถ้าซีพียูทำงานอยู่ที่แบงค์ 3 เมื่อถูกรีเซ็ทก็จะกลับมาทำงานที่แบงค์ 0 เสมอ และ SP จะมาเริ่มต้นที่ตำแหน่ง 07h ทันที

2.6.3.2 บริเวณหน่วยความจำที่ใช้คำสั่งอ่านเขียนทีละบิตได้ (Bit Addressable Area)

พื้นที่ตั้งแต่แอดเดรส (20h-7Fh) จำนวน 16 ไบต์หรือแบ่งเป็นบิตจะได้เท่ากับ 128 บิต ซึ่งตำแหน่งบิตมีดังนี้ 00, 01, 02, 03, 04, 05, 06, 07 จนถึง7FH

เช่น บิต 00 ก็คือ D0 ของหน่วยความจำตำแหน่งที่ 20h

บิต 01 ก็คือ DI ของหน่วยความจำตำแหน่งที่ 20h

ดูรูปที่ 2.15 ประกอบ เช่นต้องการเซ็ทบิต 00 ต้องเขียนคำสั่งว่า SET 00h

2.6.5 ไทม์เมอร์/เคาน์เตอร์

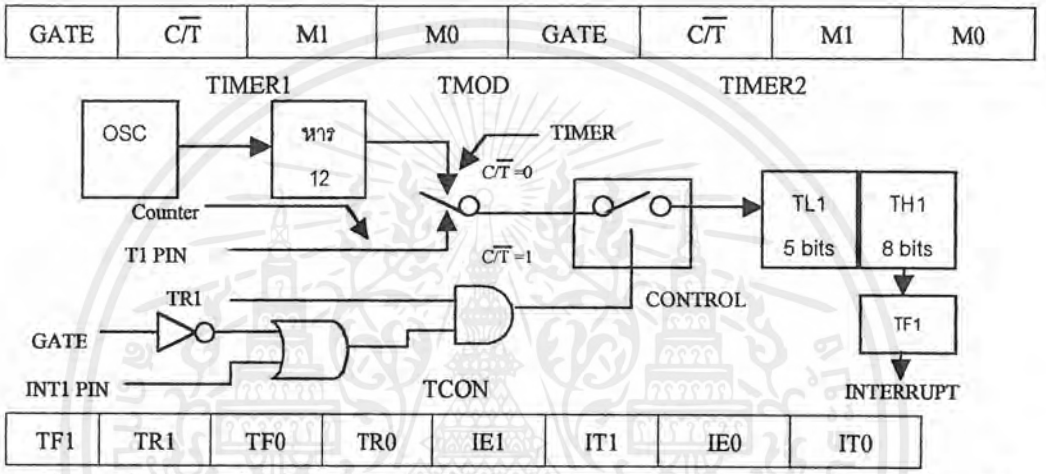
2.6.5.1 ไทม์เมอร์/เคาน์เตอร์ สามารถเลือกให้มีการทำงานเป็นไทม์เมอร์ หรือเคาน์เตอร์อย่างใดอย่างหนึ่ง โดยเลือกที่บิต $\overline{C/T}$ ในรีจิสเตอร์ใช้งานเฉพาะ TMOD โหมดไทม์เมอร์และเคาน์เตอร์จะใช้ up Counter Register (TH_x, TL_x) ตัวเดียวกันซึ่งเป็นแบบนับขึ้น

2.6.5.2 โหมดไทม์เมอร์ up Counter Register (TH_x, TL_x) จะถูกเพิ่มค่าทุกๆ 1 แมชชีนไซเคิล (12 คาบเวลาของ CPU osc) โหมดนี้ไม่ต้องป้อนสัญญาณจากภายนอกเข้ามาแต่จะใช้สัญญาณ CPU osc

2.6.5.3 โหมดเคาน์เตอร์ up Counter Register (TH_x, TL_x) จะถูกเพิ่มค่าทีละหนึ่งเมื่อสัญญาณนาฬิกาจากภายนอกเข้ามา 1 ลูกเข้ามาทางขา T0 (pin) หรือ T1 (pin) อยู่ที่ขา 14 และ 15 ตามลำดับโดยไม่สนใจคาบเวลาของพัลซ์แต่ละลูกการตรวจสอบสัญญาณที่เข้ามาทางขานี้โดยจะตรวจสอบทุกๆ S5P2 ของแต่ละแมชชีนไซเคิล ดังนั้นการตรวจสอบสัญญาณนาฬิกา 1 ลูกจะต้องใช้ถึง 2 แมชชีนไซเคิล (1/24 คาบเวลาของ CPU osc)

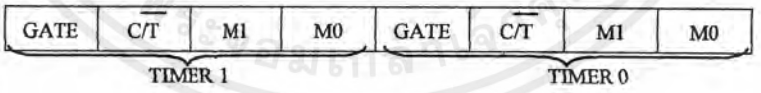
2.6.5.4 โครงสร้างของไทม์เมอร์และเคาน์เตอร์ มีโครงสร้างดังนี้

- เคาน์เตอร์ แบบนับขึ้น up Counter Register (TH_x, TL_x)
- ส่วนเลือกโหมดไทม์เมอร์ และ เคาน์เตอร์ เลือกที่บิต $\overline{C/T}$
- ส่วนควบคุมการนับ และหยุดนับ (Start Counter) ควบคุมที่บิต TR_x, GATE และสัญญาณจากภายนอกที่ขา $\overline{INTx}$ (pin)
- ในโหมดเคาน์เตอร์จะรับอินพุตพับซ์จากภายนอกที่ขา Tx (pin)
- ในโหมดไทม์เมอร์จะรับอินพุตพัลซ์จากคล็อคซีพียูที่หารด้วย 12 แล้ว
- ในโหมดเคาน์เตอร์และไทม์เมอร์ใช้เคาน์เตอร์ตัวเดียวกันเป็นแบบนับขึ้น



รูปที่ 2.15 ผังการทำงานของไทม์เมอร์/เคาน์เตอร์1(โหมด 0) 13-bit Counter และรีจิสเตอร์ควบคุม

2.6.5.5 Timer/Counter Mode Control Register (TMOD) อยู่ใน SFR ตำแหน่งที่ (89H)



GATEx - เป็นบิตเลือกการสตาร์ทไทม์เมอร์เคาน์เตอร์ x

0 ควบคุมโดย Software (Internal Control)

1 ควบคุมโดย Hardware (External Control โดยการทริกจากภายนอก)

$\overline{C/T}$ - บิตเลือกการทำงานเป็นไทม์เมอร์หรือเคาน์เตอร์โดยเลือกดังนี้

ถ้า $\overline{C/T} = 0$ เป็นการเลือกโหมดไทม์เมอร์ (นับจำนวนแมชชีนไซเคิล)

ถ้า $\overline{C/T} = 1$ เป็นการเลือกโหมดเคาน์เตอร์ (นับจำนวนพัลซ์จากภายนอก)

M1,M2 - เลือกโหมดการทำงานได้ 4 โหมด

M1	M2	โหมด	การทำงาน
0	0	0	13 บิต ไทม์เมอร์หรือเคาน์เตอร์
0	1	1	16 บิต ไทม์เมอร์หรือเคาน์เตอร์
1	0	2	8 บิต ไทม์เมอร์หรือเคาน์เตอร์แบบโหลดซ้ำอัตโนมัติ
1	1	3	8 บิต ไทม์เมอร์หรือเคาน์เตอร์ โดยใช้ TL0
1	1	3	8 บิต ไทม์เมอร์โดยใช้ TH0

2.6.5.6 Timer/Counter Control Register (TCON) อยู่ใน SFR ตำแหน่งที่ (088H)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

บิต TF1 - แฟลกซ์แสดงการเกิดโอเวอร์โฟลว์ของไทม์เมอร์ 1 จะเซ็ทเมื่อไทม์เมอร์ 1 โอเวอร์โฟลว์และจะถูกเคลียร์เองเมื่อซีพียูย้ายไปที่โปรแกรมบริการอินเตอร์รัปต์หรือใช้คำสั่ง CLR TF1

บิต TR1 - บิตควบคุมการนับของไทม์เมอร์ เคาเตอร์ 1 ควบคุมจากโปรแกรม

1 ไทม์เมอร์ หรือ เคาน์เตอร์ 1 เริ่มทำงาน

0 ไทม์เมอร์ หรือ เคาน์เตอร์ 1 หยุดทำงาน

(กรณีนี้ต้องเซ็ทหรือรีเซ็ทบิต GATE1 ใน TMOD ก่อน)

บิต TF0 - แฟลกซ์แสดงการเกิด โอเวอร์โฟลว์ ของไทม์เมอร์ 0 ถูกเซ็ทเมื่อไทม์เมอร์ 0 เกิดโอเวอร์โฟลว์ เช่นเดียวกับ TF1

บิต TR0 - เช่นเดียวกับ TR1 แต่ใช้ควบคุมไทม์เมอร์ 0

บิต IE1 - แฟลกซ์แสดงการเกิดสัญญาณอินเตอร์รัปต์ภายนอกหมายเลข 1 เมื่อมีสัญญาณอินเตอร์รัปต์เข้ามาที่ขา $\overline{\text{INT 0}}$ และถูกเคลียร์เองโดยคำสั่ง RETI ในโปรแกรมส่วนบริการอินเตอร์รัปต์

บิต IT1 - แฟลกซ์เลือกประเภทการตรวจสอบสัญญาณอินเตอร์รัปต์ที่เกิดขึ้นที่ขา $\overline{\text{INT 1}}$ โดย

1 จะตรวจสอบการเปลี่ยนระดับแบบขอบขาลงที่ขา $\overline{\text{INT 1}}$

0 จะตรวจสอบระดับศูนย์ของสัญญาณที่ขา $\overline{\text{INT 1}}$

บิต IE0 - แฟลกซ์แสดงการเกิดสัญญาณอินเตอร์รัปต์ภายนอกหมายเลข 0 เมื่อมีสัญญาณอินเตอร์รัปต์เข้ามาที่ขา $\overline{\text{INT 1}}$ และถูกเคลียร์เอง โดยคำสั่ง RETI ที่อยู่ในโปรแกรม ส่วนบริการอินเตอร์รัปต์

บิต IT0 - เช่นเดียวกับ IT1

ตารางที่ 2.2 โหมดการทำงานของ ไทม์เมอร์ 2

RCLK+TCLK	$\overline{CP/RL2}$	TR2	MODE
0	0	1	16-bit auto-reload
0	1	1	16-bit capture
1	X	1	Baud rate generator
X	X	0	(off)

บิต TF2 - แฟลกซ์แสดงการเกิดโอเวอร์โฟลว์ของ ไทม์เมอร์ 2 จะเซ็ทเมื่อไทม์เมอร์ 2 เกิดโอเวอร์โฟลว์และจะถูกเคลียร์โดยใช้ซอร์ฟแวร์เท่านั้นโดยใช้คำสั่ง CLR TF2 เท่านั้น บิต TF2 จะไม่ถูกเซ็ทถ้า บิต RCLK=1 และ TCLK=1

บิต EXF2 - แฟลกซ์แสดงการเกิดอินเตอร์รัพต์จากภายนอกเบอร์ 2 เมื่อมีสัญญาณทริกจากภายนอกแบบขอบขาลงที่ขา T2EX pin และ บิต EXEN2 ต้อง =1 การเคลียร์จะใช้ซอร์ฟแวร์เท่านั้นเช่น CLR EXF2

บิต RCLK - ใช้เลือกแหล่งกำเนิดสัญญาณคล้อคสำหรับกำหนดบอดเรทสำหรับพอร์ทสื่อสาร **ทางด้านรับ**

1 = เลือกจาก ไทม์เมอร์ 2

0 = เลือกจาก ไทม์เมอร์ 1

บิต TCLK - ใช้เลือกแหล่งกำเนิดสัญญาณคล้อคสำหรับกำหนดบอดเรทสำหรับพอร์ทสื่อสาร **ทางด้านส่ง**

1 = เลือกจาก ไทม์เมอร์ 2

0 = เลือกจาก ไทม์เมอร์ 1

บิต EXEN2 - บิตเลือก อนุญาตหรือไม่อนุญาตสัญญาณอินเตอร์รัปต์ภายนอกที่ขา T2EX pin

1 = อนุญาต

0 = ไม่อนุญาต

บิต TR2 - บิตควบคุมการสตาร์ทของไทม์เมอร์ 2 ควบคุมจากโปรแกรม

1 ไทม์เมอร์ หรือ เคาน์เตอร์ 2 **เริ่มทำงาน**

0 ไทม์เมอร์ หรือ เคาน์เตอร์ 2 **หยุดทำงาน**

บิต $C/\overline{T2}$ - บิตเลือกโหมดการทำงาน ไทม์เมอร์ หรือ เคาน์เตอร์ 2

1 ไทม์เมอร์ หรือ เคาน์เตอร์ 2 **ทำงาน** โหมดเคาน์เตอร์โดยทริกแบบขอบขาลงที่ที่ขา T2EX pin และต้องเซ็ทบิตEXEN2ไว้ล่วงหน้า

0 ไทม์เมอร์ หรือ เคาน์เตอร์ 2 **เลือกการทำงาน**โหมดเคาน์เตอร์

บิต CP/RL2 - บิตเลือกโหมดการทำงาน Capture หรือ Reload

1 ไทม์เมอร์ หรือ เคาน์เตอร์ 2 **ทำงาน** Capture mode โดยทริกแบบขอบขาลงที่ขา T2EX pin และต้องเซ็ทบิตEXEN2ไว้ล่วงหน้า

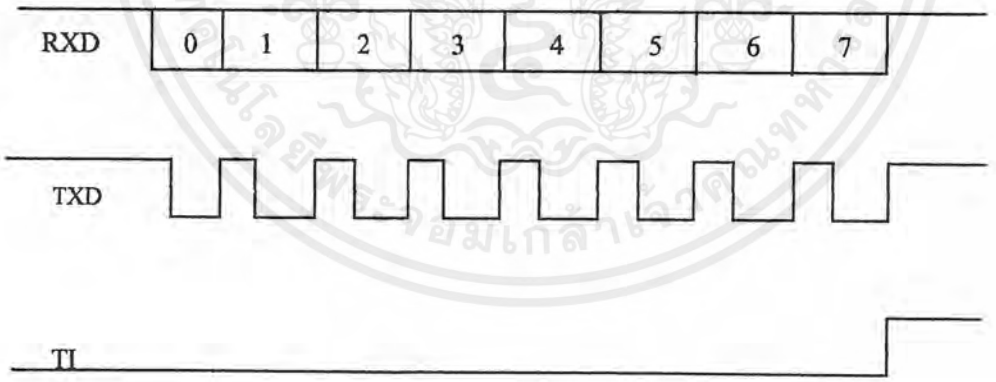
0 ไทม์เมอร์ หรือ เคาน์เตอร์ 2 **เลือกการทำงานแบบ** Reload mode โหลดค่าซ้ำอัตโนมัติเมื่อ TF2= 1 หรือ โดยการทริกแบบขอบขาลงที่ขา T2EX pin และต้องเซ็ทบิต EXEN2 ไว้ก่อน

2.6.6 การรับส่งข้อมูลอนุกรม (UART) ของ 8051 และ 8052

พอร์ทสื่อสารอนุกรมของ 8051, 8052 มีโครงสร้างการทำงานในแบบที่เรียกว่า ฟูลดูเพล็กซ์ (Full Duplex) ในการรับและส่งข้อมูลอนุกรมได้ในเวลาเดียวกัน โดยทางด้านส่งใช้ขา TxD (พอร์ท 3.1) ทางด้านรับใช้ขา RxD (พอร์ท 3.0) SBUF ใช้เป็นบัฟเฟอร์สำหรับรับและส่งข้อมูลอนุกรม

พอร์ทสื่อสารอนุกรมของ 8051 สามารถโปรแกรมการทำงานได้หลายโหมดด้วยกันโดยเลือกที่บิต SM1 และ SM0 ซึ่งอยู่ในรีจิสเตอร์ควบคุม SCON การทำงานทั้ง 4 โหมด ของพอร์ทสื่อสารอนุกรม มีดังนี้

โหมด 0 : พอร์ทสื่อสารอนุกรม 8 บิต โดยการส่งจะเลื่อนออกทีละบิตโดยส่งบิต D0 ออกไปก่อนทาง RxD และไม่มีการส่ง start bit แต่จะส่ง shift clock ทางขา TxD (ความเร็ว 1/12 เท่าของ CPU Clock)



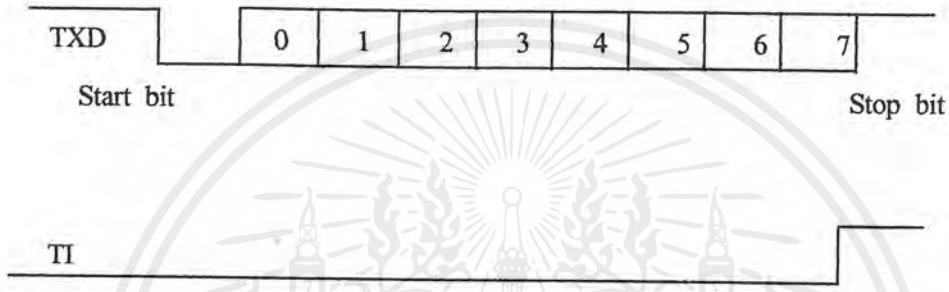
โหมด 1 : พอร์ทสื่อสารอนุกรม 10 บิต ข้อมูล 8 บิต 1 start bit และ 1 stop bit และสามารถเปลี่ยนแปลงความเร็วในการส่งข้อมูลได้ โดยขึ้นกับบิต SMOD ใน PCON และอัตราโอเวอร์โฟลว์ของ Timer 1)

$$\text{Baud Rate Mode 1} = \frac{K \times \text{Oscillator Freq.}}{32 \times 12 \times [256 - (TH1)]}$$

สำหรับ 8032,8052 โดยใช้ Timer 1

$$\text{Baud Rate Mode 1} = \frac{\text{Oscillator Freq.}}{32 \times [65536 - (RCAP2H,RCAP2L)]}$$

สำหรับ 8032,8052 โดยใช้ Timer 2

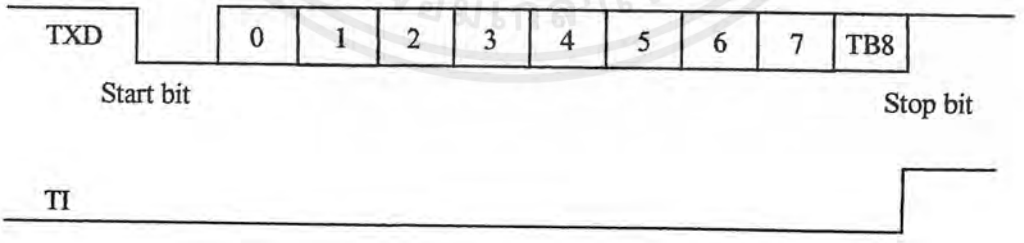


โหมด 2 : พอร์ทสื่อสารอนุกรม 11 บิต ใช้ข้อมูล 9 บิต 1 start bit และ 1 stop bit (TB8 นิยมนำมาใช้ส่ง Parity bit

(ความเร็วในการรับส่งข้อมูลเท่ากับ 1/32 และ 1/64 ของ CPU clock โดยขึ้นกับบิต SMOD ใน PCON)

Baud Rate Mode2 = 1/(32 Osc Freq) เมื่อ SMOD = 1

Buad Rate Mode2 = 1/(64 Osc Freq) เมื่อ SMOD = 0



โหมด 3 : พอร์ทสื่อสารอนุกรมแบบ 11 bit UART โดยส่งข้อมูล 9 บิต 1 start bit และ 1stop bit เหมือนโหมด 2 ยกเว้นอัตราความเร็วจะขึ้นกับบิต SMOD ใน PCON และอัตราโอเวอร์โฟลว์ของ Timer 1 สำหรับ 8051 หรือขึ้นกับ อัตราโอเวอร์โฟลว์ของ Timer 2 สำหรับ 80C154D

Buad Rate Mode3 = $\frac{\text{K x Oscillator Freq.}}{32 \times 12 \times [256 - (TH1)]}$ สำหรับ 8031,8051 โดยใช้ Timer 1

Buad Rate Mode3 = $\frac{\text{Oscillator}}{32 \times [65536 - (RCAP2H,RCAP2L)]}$ สำหรับ 8032,8052 80154,80154D โดยใช้ Timer

2.6.7 การเชื่อมต่อไมโครโปรเซสเซอร์เพื่อรับส่งข้อมูลอนุกรม (UART)

มีอยู่ 2 โหมดด้วยกันคือ

Single Processor Mode

Multiprocessors Mode

Single Processor Mode : ในโหมดนี้เราจะใช้ไมโครคอนโทรลเลอร์ 2 ตัวเชื่อมเข้าหากัน

Multiprocessors Mode : ในโหมดนี้เราจะใช้ไมโครคอนโทรลเลอร์ 1 ตัวเป็นตัวแม่ (Master) และอีก 256 ตัวลูก (Slave) รีจิสเตอร์ที่ใช้ควบคุมการรับส่งข้อมูลอนุกรม

ตารางที่ 2.3 แสดงการทำงานของ SM1 และ SM0 ในโหมดต่างๆ

SM1	SM0	โหมด	การทำงาน
0	0	0	Shift register อัตราเร็วในการรับหรือส่งข้อมูลเท่ากับ ½ ของความถี่ออสซิลเลเตอร์ของ CPU
0	1	1	8 bit UART อัตราเร็วในการรับหรือส่งข้อมูลขึ้นกับ Timer 1, 2 และ บิต SMOD
1	0	2	9 bit UART อัตราเร็วในการรับหรือส่งข้อมูล=1/32 หรือ 1/64 ของความถี่ออสซิลเลเตอร์ ขึ้นกับบิต SMOD ใน PCON
1	1	3	9 bit UART อัตราเร็วในการรับหรือส่งข้อมูลกำหนดที่ Timer 1, 2 และ บิต SMOD

SM2 บิตเลือกการทำงานแบบ Single Processor Mode หรือ Multiprocessor Mode

1 : เลือก Multiprocessor Mode ใช้ได้กับโหมด 2,3

0 : เลือก Single Processor Mode ใช้ได้กับทุกโหมด

เมื่อเลือกการทำงานรับข้อมูลแบบ Multiprocessor Mode แล้ว

ถ้าข้อมูลบิตที่ 9 ที่รับได้มีค่าเป็น **1** RI จะเซ็ท

ถ้ารับข้อมูลบิตที่ 9 ที่รับได้มีค่าเป็น **0** RI จะเคลียร์

REN (Receive Enable) บิตควบคุมให้รับหรือไม่รับข้อมูล

1 : ให้รับข้อมูลได้

0 : ห้ามรับข้อมูล

TB8 (Transmit bit D8) ข้อมูลบิตที่ 9 ที่จะส่งออกไปในโหมด 2, 3 ให้ใส่ในบิตนี้

RB8 (Receive bit D8) ข้อมูลบิตที่ 9 ที่รับเข้ามาจะเก็บในบิตนี้

(ข้อมูลบิตที่ 9ก็คือค่าใน TB8 ทางด้านส่งนั่นเอง)

TI แฟลกซ์ TI จะเป็น 1 เมื่อสิ้นสุดการส่งข้อมูลบิตสุดท้าย

RI แฟลกซ์ RI จะเป็น 1 เมื่อรับข้อมูลเสร็จบิตสุดท้ายเข้ามา (บิต RI,TI ผู้เขียนโปรแกรมจะต้องเคลียร์เอง)

ตารางที่ 2.4 ตารางการใช้ ไทม์เมอร์ 1 กำหนดบอดเรท

Baud Rate	Fosc	SMOD	TIMER 1		
			$C/\overline{T}$	MODE	Reload Value
(MODE 0) Max : 1 MHz	12 MHz	X	X	X	X
(MODE 2) Max : 375 KHz	12 MHz	1	X	X	X
(MODE 2) Min : 187.5 KHz	12 MHz	0	X	X	X
MODE 1, 3 : 62.5 KHz	12 MHz	1	0	2	FFH
19.2 KHz	11.059 MHz	1	0	2	FDH
9.6 KHz	11.059 MHz	0	0	2	FDH
4.8 KHz	11.059 MHz	0	0	2	FAH
2.4 KHz	11.059 MHz	0	0	2	F4H
1.2 KHz	11.059 MHz	0	0	2	E8H
137.5	11.059 MHz	0	0	2	1DH
110	6 MHz	0	0	2	72H
110	12 MHz	0	0	1	FEEBH

2.6.8 การอินเตอร์รัปต์

คือ การขัดจังหวะโปรแกรมชั่วขณะ แล้วมาทำโปรแกรมบริการอินเตอร์รัปต์ (Interrupt Service Routine; ISR) การตรวจสอบสัญญาณการต้องขออินเตอร์รัปต์จะตรวจสอบที่ตำแหน่ง S5P2 ของทุกๆ แมชชีนไซเคิลเมื่อพบแล้วในช่วงแมชชีนไซเคิลที่ 2 จะเป็นการตรวจสอบว่าเป็นของอุปกรณ์ใดและแมชชีนไซเคิลที่ 3 จะกระโดดไปทำโปรแกรมบริการอินเตอร์รัปต์

ตารางที่ 2.5 อินเตอร์รัปต์เวคเตอร์ของ MCS-51 และลำดับความสำคัญของการอินเตอร์รัปต์

ลำดับ	ชื่อสัญญาณอินเตอร์รัปต์	Vector Address	Priority
1	$\overline{\text{INT0}}$	0003H	Highest
2	TF0	000BH	
3	$\overline{\text{INT1}}$	0013H	
4	TF1	001BH	
5	TI+RI	0023H	
6	TF2+EXF2	002B	Lowest

* ถ้ามีอินเตอร์รัปต์เข้ามาพร้อมกัน $\overline{\text{INT0}}$ จะถือว่ามี Priority สูงสุด *

2.6.8.1 Interrupt Enable Register (IE) อยู่ใน SFR ตำแหน่งที่ (0A8H)

ใช้ควบคุมอินเตอร์รัปต์ได้ 8 แหล่ง ดูตารางที่ 2.6 ประกอบเราสามารถสั่งห้ามหรือไม่ห้ามการอินเตอร์รัปต์ได้จากรีจิสเตอร์ชุดนี้ดังมีรายละเอียดดังรูปที่ 2.26 เราสามารถสั่งห้ามไม่ให้เกิดการขัดจังหวะทั้งหมดก็ได้เพียงแต่ไปรีเซ็ทบิต $\overline{\text{EA}}$ ใน Interrupt Enable Register (IE) ถ้าต้องการ Enable อินเตอร์รัปต์จากอุปกรณ์ตัวไหนก็เพียงแต่ไปเซ็ทบิตของอุปกรณ์ตัวนั้นไว้ แต่อย่าลืมเซ็ทบิต $\overline{\text{EA}}$

ตารางที่ 2.6 แสดงรายละเอียดของรีจีสเตอร์ IE

บิต	ชื่อบิต	การทำงาน
IE.7	$\overline{EA}$	=1 หมายถึงยอมให้เลือกการทำอินเตอร์รัปต์จากอินเตอร์รัปต์จากแหล่งได้ =0 หมายถึง ไม่ยอมให้ทำอินเตอร์รัปต์จากแหล่งใดๆ ทั้งสิ้น
IE.6	X	ไม่ได้ใช้งาน
IE.5 IE.2		=1 หมายถึง ยอมให้ทำอินเตอร์รัปต์เมื่อ TF2 เกิดโอเวอร์โฟลว์ =0 หมายถึง ไม่ยอมให้ทำอินเตอร์รัปต์เมื่อ TF2 เกิดโอเวอร์โฟลว์
IE.4	ES	=1 หมายถึง ยอมให้ทำอินเตอร์รัปต์จากพอร์ทสื่อสารอนุกรมได้ =0 หมายถึง ไม่ยอมให้ทำอินเตอร์รัปต์จากพอร์ทสื่อสารอนุกรมได้
IE.3 ET1		=1 หมายถึง ยอมให้ทำอินเตอร์รัปต์เมื่อ TF1 เกิดโอเวอร์โฟลว์ =0 หมายถึง ไม่ยอมให้ทำอินเตอร์รัปต์เมื่อ TF1 เกิดโอเวอร์โฟลว์
IE.2 EX1		=1 หมายถึง ยอมให้ทำอินเตอร์รัปต์จากสัญญาณภายนอกหมายเลข 1 =0 หมายถึง ไม่ยอมให้ทำอินเตอร์รัปต์จากสัญญาณภายนอกหมายเลข 1
IE.1 ET0		=1 หมายถึง ยอมให้ทำอินเตอร์รัปต์เมื่อ TF0 เกิดโอเวอร์โฟลว์ =0 หมายถึง ไม่ยอมให้ทำอินเตอร์รัปต์เมื่อ TF0 เกิดโอเวอร์โฟลว์
IE.0	EX0	=1 หมายถึง ยอมให้ทำอินเตอร์รัปต์จากสัญญาณภายนอกได้ (ขา $\overline{INT0}$) =0 หมายถึง ไม่ยอมให้ทำอินเตอร์รัปต์จากสัญญาณภายนอก (ขา $\overline{INT0}$)

ความหมายของสัญญลักษณ์

E = Enable หรือ External

T = Timer

0,1,2 = Channel 0,Channel 1, Channel 2

- อินเตอร์รัปต์ภายใน MCS-51 ได้จาก Timer 0, Timer 1 โดยตรวจสอบที่ TF0 และ TF1 และอินเตอร์รัปต์จาก Serial Port โดยตรวจสอบที่ TI และ RI
- ขา INT และ INT ใช้งาน 2 หน้าที่ โดยที่

  เมื่อทำอินเตอร์รัปต์จะเป็นขา External Interrupt Input

  เมื่อไม่ทำอินเตอร์รัปต์ จะใช้ Start Counter หรือที่เราเรียกว่า Hardware Start

  การขัดจังหวะของการอินเตอร์รัปต์เราสามารถกำหนดลำดับความสำคัญได้จาก Interrupt Priority Register (IP) ดังรายละเอียดดังรูปที่ 2.16

2.6.8.2 Interrupt Priority Register (IP) อยู่ใน SFR ตำแหน่งที่ (0B8H)

ใช้กำหนดลำดับความสำคัญของการอินเตอร์รัปต์

PCT	-	PT2	PS	PT1	PX1	PT0	PX0

PCT IP.7 Defines the priority level for all the source interrupt (83C154 and 83C154D only).

IP.6 Not implemented, reseved for future use*.

PT2 IP.5 Defines the Timer2 interrupt priority level (80C52 and 83C154D only).

PS IP.4 Defines the Serial port interrupt priority level.

PT1 IP.3 Defines the Timer 1 interrupt priority level.

PX1 IP.2 Defines External Interrupt priority level.

PT0 IP.1 Defines the Timer 0 interrupt priority level.

PX0 IP.0 Defines the External interrupt 0 pririty level.

* User software should not write 1s to reserved bits. These bits may be used in future MHS C51 products to invoke new features. In that case, the reset or inactive value of the now bit will be 0, and its active value will be 1.

รูปที่ 2.16 รายละเอียดของ IP

คำอธิบายความหมายในแต่ละบิตใน (IP)

PT2 : 0 Timer 2 มีความสำคัญต่ำสุด

1 Timer 2 มีความสำคัญสูงสุด

PS : 0 พอร์ทสื่อสารอนุกรม UART มีลำดับความสำคัญต่ำสุด

1 พอร์ทสื่อสารอนุกรม มีลำดับความสำคัญสูงสุด

PT1 : 0 Timer 1 มีลำดับความสำคัญต่ำสุด

1 Timer 1 มีลำดับความสำคัญสูงสุด

PT0 : 0 Timer 0 มีลำดับความสำคัญต่ำสุด

1 Timer 0 มีลำดับความสำคัญสูงสุด

PX0 : 0 อินเตอร์รัปต์ภายนอกชนิด 0 มีลำดับความสำคัญต่ำสุด

1 อินเตอร์รัปต์ภายนอกชนิด 0 มีลำดับความสำคัญสูงสุด

PCT : 0 ยอมให้มีการจัดลำดับความสำคัญของการอินเตอร์รัปต์ (priority)

Interrupt Vector ของ MCS-51

มีอยู่ 8 แหล่งดังแสดงในตารางที่ 2.7 ดังนี้

ตารางที่ 2.7 Interrupt Vector

Source	Vector Address
IE0	0003H
TF0	000BH
IE1	0013H
TF1	001BH
RI+TI	0023H
*TF2+EXF2	002BH

* มีเฉพาะในเบอร์ 8052 ขึ้นไป

2.6.8.3 รีจีสเตอร์ที่ใช้ในการประหยัดพลังงาน Power Control Register (PCON) อยู่ใน SFR ตำแหน่งที่ (87H)

SMOD	HPD	RPD	-	GF1	GF0	PD	IDL

บิต SMOD - บิตกำหนดอัตราความเร็วการรับส่งข้อมูลอนุกรม UART

0 = อัตราความเร็ว 1 เท่า

1 = อัตราความเร็ว 2 เท่า

บิต HPD - Hard Power Down bit (มีใน 83C154, 83C154D เท่านั้น) หมายถึงการใช้สัญญาณจากภายนอกมากระตุ้นให้หยุดหรือเริ่ม Power Down mode เมื่อถูกรีเซ็ทจะหยุดการทำงานในโหมดนี้

บิต RPD - Recover Power Down bit (มีใน 83C154, 83C154D เท่านั้น) มันได้ถูกนำไปใช้สำหรับยกเลิก Power-Down/Idle mode

1 = ถ้าการร้องขออินเตอร์รัปต์ถูก Enable ไว้จะกระโดดไปทำโปรแกรมบริการอินเตอร์รัปต์

0 = ถ้าการร้องขออินเตอร์รัปต์ถูก Disable ไว้โปรแกรมจะทำงานต่อหลังจาก Power-Down/Idle Instruction

บิต GF1 - แฟลกซ์ใช้งานทั่วไป

บิต GF2 - แฟลกซ์ใช้งานทั่วไป

บิต PD - Power Down bit

1 = หยุดออสซิเลเตอร์ของซีพียูสัญญาณรีเซ็ทหรืออินเตอร์รัปต์ (83C154, 83C154D เท่านั้น) ที่จะยกเลิกโหมดนี้

บิต IDL - Idle Mode bit

1 = หยุดการทำงานของซีพียู สัญญาณรีเซ็ทหรืออินเตอร์รัปต์เท่านั้นที่จะยกเลิกโหมดนี้ได้

บทที่ 3

8255 Programmable Peripheral Interface

2.6 บทนำ

IC อีกตัวหนึ่งที่เป็นที่รู้จักกันดีและนิยมนำมาใช้งานร่วมกับ ไมโครคอนโทรลเลอร์ MCS-51 ในลักษณะงานควบคุมระบบที่พอร์ต I/O ของMCU ไม่เพียงพอต่อการใช้งานนั้น IC 8255 PPI ตัวนี้ เป็น IC ที่ผลิตมาโดยมีคุณสมบัติในการควบคุมการทำงานที่ง่ายและมีจำนวนพอร์ต I/O ขนาด 8 บิต จำนวน 3 พอร์ต ภายในตัวจึงเป็น IC ที่น่าสนใจที่จะนำมาใช้งานในการเพิ่มขยายพอร์ต I/O ให้กับ MCU โดยรูปแบบโครงสร้างคุณสมบัติต่างๆมีดังนี้

- เป็นชิพขนาด 40 ขา
- มีพอร์ตใช้งาน 3 พอร์ต แบ่งออกเป็นพอร์ต A,B และ C
- พอร์ตใช้งานมีลักษณะเป็นพอร์ตแล็ช(Latch)คงสภาวะ

รูปที่ 3.1 แสดงลักษณะการจัดขา 8255 PPI

- ควบคุมการทำงานด้วยขา AO และ A1
- สามารถโปรแกรมเลือกลักษณะการใช้งานพอร์ตได้ ทั้งในลักษณะ Input, Output หรือ Bi-Directional พอร์ต

3.2 คุณสมบัติขาสัญญาณต่างๆของ8255

Vcc เป็นขารับไฟเลี้ยงของระบบ +5 V

GND เป็นขาต่อกราวด์ของระบบ

D0-D7 เป็นขา Bi-Directional Bus โดยเป็น Data Bus ของระบบ

RESET เมื่อขานี้มีสภาวะ high จะทำให้ 8255 ถูกรีเซ็ตมีผลทำให้ทุกพอร์ตเปลี่ยนเป็นอินพุต พอร์ตทันที หรือในลักษณะอินพุตโหมด

CS (Chip Select) เป็นขาที่ใช้เป็นตัวควบคุมเลือกจังหวะการทำงานของ 8255จะทำงานเมื่อขานี้อยู่ใน สภาวะ low

RD (Read) เมื่อขานี้มีสภาวะ low MCU จะทำการอ่านข้อมูลผ่านทาง Data Busของ 8255 ได้โดยพร้อมกับขา CS ต้องมีสภาวะ low

WR (Write) เมื่อขานี้มีสภาวะ low 8255 จะทำการรับข้อมูลหรือ control word ที่ส่งมาจาก MCU ได้โดยพร้อมกับขา CS ต้องมีสภาวะ low

A0-A1 ใช้ในการรับสัญญาณควบคุมหรือ เลือกการใช้งานพอร์ต A, B และCของ8255

PA0-PA7 เป็นพอร์ตI/O ขนาด 8 บิต

PB0-PB7 เป็นพอร์ต I/O ขนาด 8 บิต

PC0-PC7 เป็นพอร์ตขนาด I/O ขนาด 8 บิต และมีคุณสมบัติแตกต่างจากพอร์ต A และ B คือ สามารถแบ่งการใช้งานเป็น 4 บิต(C4-C7) และ 4 บิตล่าง(C0-C3)ได้

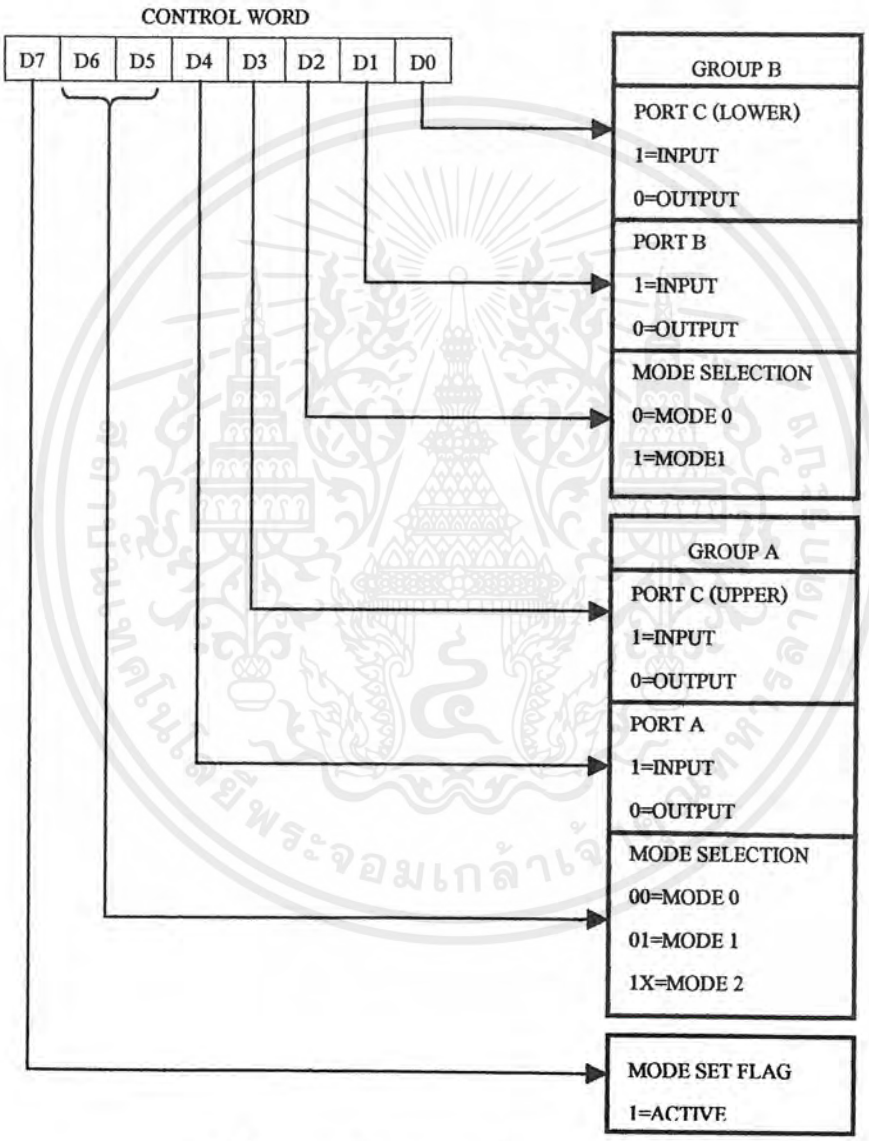
8255 PPI นั้น สามารถที่จะเลือกโหมด การใช้งานได้จากคำสั่งทาง Software จาก MCU ผ่าน Data Bus D0และ D7 ของ 8255โดยโหมดการใช้งาน 8255 มีด้วยกัน 3 โหมด คือ โหมด 0 (Basic I/O) เป็นลักษณะการใช้งาน พอร์ต A, B และC ของ 8255 เป็น I/O ปกติ โหมด 1 (Strobed I/O) จะมีลักษณะการตรวจสอบสัญญาณ Strobe ก่อนที่จะทำงานโหมด 2 (Bi-directional bus) จะใช้งานพอร์ตในลักษณะI/O แบบ 2 ทิศทาง(รับส่งข้อมูล)

ตารางที่ 3.1 ตารางความจริงของ 8255

A1	A0	RD	WR	CS	INPUT OPERATION (READ)
0	0	0	1	0	Port A → Data Bus
0	1	0	1	0	Port B → Data Bus
1	0	0	1	0	Port C → Data Bus
1	1	0	1	0	Control Word → Data Bus
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	Data Bus → Port A
0	1	1	0	0	Data Bus → Port B
1	0	1	0	0	Data Bus → Port C
1	1	1	0	0	Data Bus → Control

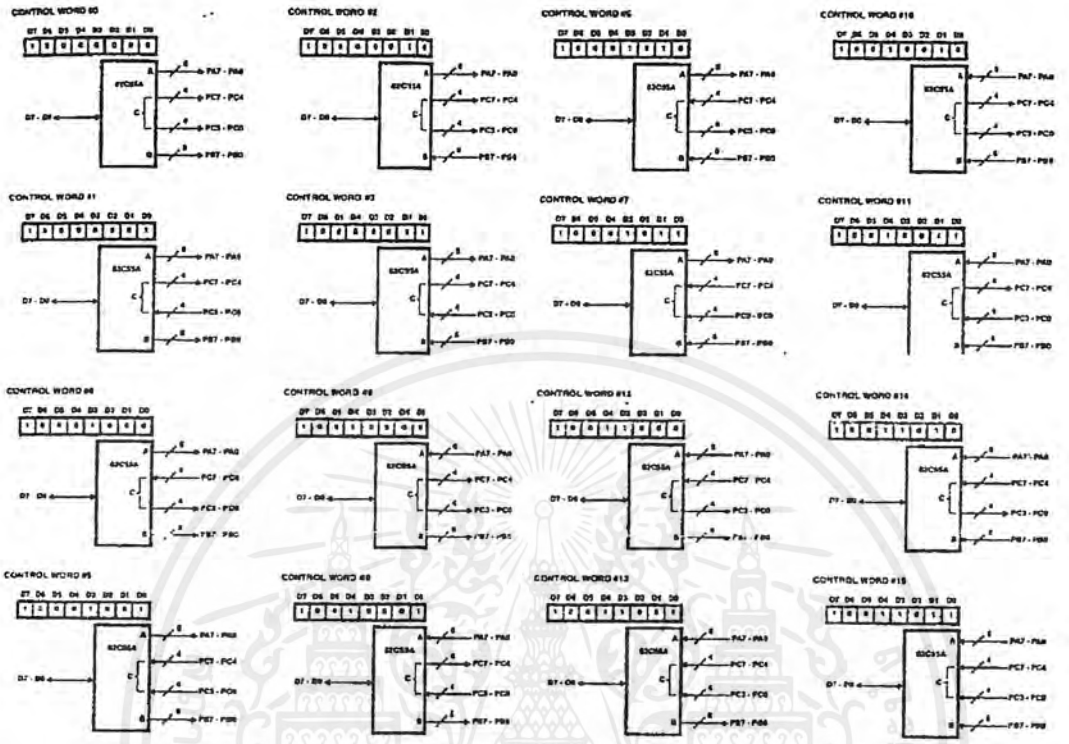
DISABLE FUNCTION					
X	X	X	X	1	Data Bus → Three-State
X	X	1	1	0	Data Bus → Three-State

โดยข้อมูลที่ถูกส่งไปควบคุมการทำงานของ 8255 ผ่านทาง Data bus D0-D7 นั้นจะถูกเรียกว่า Control Word หรือ Control Code โดยมีรูปแบบหรือคำสั่งอยู่ในแต่ละบิตของชุด Control Word ที่ส่งไปควบคุม 8255 ดังรูปที่ 3.2



รูปที่ 3.2 แสดงลักษณะการจัด control word

3.3 ลักษณะการจัด Control Word Mode 0



รูปที่ 3.3 แสดงลักษณะการจัด Control Word Mode 0

เมื่อเริ่มต้นก่อนที่จะใช้งาน 8255 นั้นจำเป็นจะต้องกำหนดโหมดการใช้งาน พอร์ตต่างๆ ของ 8255 ในลักษณะใดด้วย ซึ่งคำสั่งในการกำหนดโหมดการใช้งานรวมถึงลักษณะการใช้งาน พอร์ตต่างๆของ 8255 ทั้งหมดที่กล่าวมาจะถูกเรียกว่า Control Word ดังในรูปที่ 3.2 จะแสดงลักษณะการจัด Control Word ในการควบคุม 8255 โดยจะเห็นว่าการใช้งาน 8255 ในโหมด 0 นั้น ชุดคำสั่งของ Control Word ในบิตที่ 2, 5 และ 6 จะต้องเป็น 0 ส่วนบิตที่เหลือจะเป็นบิตที่กำหนดการใช้งาน พอร์ตของ 8255 ในลักษณะ I/O พอร์ต สรุปแล้วก็คือ ชุดคำสั่งหรือ Control Word ที่กล่าวมาได้ถูกสรุปไว้ในตารางที่ 3.2 หรือถ้าจะดูรายละเอียดค่าแต่ละบิตของ Control Word สามารถดูได้จากหัวข้อ 3.3

ตาราง 3.1 เป็นตารางความจริงของ 8255 โดยจะพบว่า A0 และ A1 เป็นตัวกำหนดการใช้งานพอร์ตของ 8255 ว่าจะใช้งานพอร์ตใด ที่นี้ในการที่จะส่ง Control Word ไปยัง 8255 เพื่อให้ 8255 รับรู้ว่าเป็นชุดข้อมูลซึ่งเป็น Control word ไม่ใช่ชุด Data ธรรมดา จะต้องเซ็ตค่าให้ A0 และ

A1 เป็น 1 ทั้งคู่ หลังจากนั้นขา A0 และ A1 ก็จะถูกใช้งานเป็นขาเลือก (Select) ความต้องการใช้พอร์ตของ 8255 ว่าต้องการรับส่งข้อมูลกับพอร์ตใดของ 8255 ต่อไป

ตาราง ที่ 3.2 สรุปการทำงาน 8255 PPI โหมด 0

PORT A (PA0-PA7)	PORT B (PB0-PB7)	PORT C (PC4-PC7)	PORT C (PC0-PC3)	CONTROL WORD(HEX)
OUTPUT	OUTPUT	OUTPUT	OUTPUT	80H
OUTPUT	OUTPUT	OUTPUT	INPUT	81H
OUTPUT	INPUT	OUTPUT	OUTPUT	82H
OUTPUT	INPUT	OUTPUT	INPUT	83H
OUTPUT	OUTPUT	INPUT	OUTPUT	88H
OUTPUT	OUTPUT	INPUT	INPUT	89H
OUTPUT	INPUT	INPUT	OUTPUT	8AH
OUTPUT	INPUT	INPUT	INPUT	8BH
INPUT	OUTPUT	OUTPUT	OUTPUT	90H
INPUT	OUTPUT	OUTPUT	INPUT	91H
INPUT	INPUT	OUTPUT	OUTPUT	92H
INPUT	INPUT	OUTPUT	INPUT	93H
INPUT	OUTPUT	INPUT	OUTPUT	98H
INPUT	OUTPUT	INPUT	INPUT	99H
INPUT	INPUT	INPUT	OUTPUT	9AH
INPUT	INPUT	INPUT	INPUT	9BH

สรุปการใช้งาน 8255 ในโหมด 0 อย่างง่ายๆ ออกเป็น 2 ขั้นตอนคือ

1. ทำการเซ็ต A0 และ A1 ของ 8255 ให้เป็น 1 ทั้งคู่ แล้วทำการส่ง Control Word 8 บิต ไปยัง 8255 เพื่อเริ่มการใช้งาน

2. เลือกใช้งานพอร์ตของ 8255 ได้อย่างอิสระ โดยการควบคุมขา A0 และ A1 ของ 8255 ในตารางที่ 3.2

ที่ผ่านมานั้นเป็นการกล่าวถึงคุณสมบัติการใช้งาน โดยเบื้องต้นของ 8255 เพียงเท่านั้น ซึ่งการนำมาประยุกต์ใช้งาน ก็จะกล่าวเน้นในการใช้งาน 8255 ใน Mode 0 ซึ่งก็คือการใช้งาน 8255 เป็น Basic I/O พอร์ต เพื่อนำไปใช้งานร่วมกับ MCU ให้ใช้งานได้มากยิ่งขึ้น

บทที่ 4
การอินเตอร์เฟส MCS-51 กับ 8255 PPI

4.1 กล่าวนำ

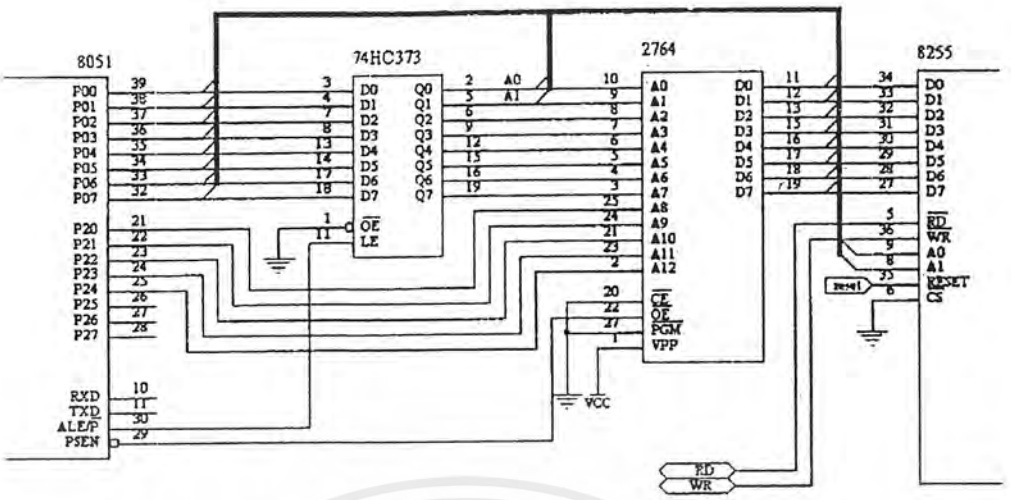
การประยุกต์ใช้งานการอินเตอร์เฟสไมโครคอนโทรลเลอร์ MCS-51 ร่วมกับอุปกรณ์ภายนอก ที่นิยมใช้งานเพื่อเพิ่มประสิทธิภาพในการใช้งานกับ MCU การอินเตอร์เฟส MCS-51 กับ 8255 โดยมีการใช้งานกับ EPROM ภายนอก / ภายในของ MCU ซึ่งจากวงจรดังรูปที่ 4.1 เป็นการต่อวงจรใช้งาน MCU กับ EPROM ภายนอกและมีการขยายพอร์ต I/O เพิ่มขึ้นโดยการต่อร่วมกับ IC 8255 PPI จากวงจรจะเห็นว่าจะมีการใช้ IC 74HC373 ในการ Latch แอดเดรสไบต์ต่ำ A0-A7ในการรับส่งข้อมูลกับ EPROM และ 8255 โดย IC 74HC373 ยังถูกใช้งานใน Latch Control Word ,A0 และ A1 ให้กับ 8255 ได้ง่ายโดยมีการจัด Memory Map เพื่อการใช้งาน 8255 จากวงจรในรูปที่ 4.1 ดังต่อไปนี้

4.2 การอินเตอร์เฟส MCS-51 กับ 8255 โดยใช้ EPROM ภายนอก

จากตารางที่ 4.1 จะเป็นการแสดงการจัด Memory Map ของวงจรในรูปที่ 4.1 จะเห็นว่าเมื่อต้องการใช้งาน PORT A ,B และ C หรือส่ง Control Word เพื่อทำการรีเซ็ต 8255 ให้ทำงานตามความต้องการนั้นๆ ก็เพียงแต่อ้างแอดเดรสของหน่วยความจำเท่านั้น เช่น ต้องการใช้งาน PORT A ก็เพียงแต่อ้างแอดเดรสของหน่วยความจำ 00H เท่านั้น ก็จะทำให้ขา A0 และ A1 ของ 8255 มีค่าเป็น ลอจิก 0 หรือ สภาวะ LOW ก็จะเป็นการควบคุม 8255 ในการใช้งาน PORT A นั้นเอง ต่อไปจะขอยกตัวอย่างการใช้งาน 8255 PORT A โดยมีวงจรการใช้งานในรูปที่ 4.1 และมีการจัด Memory Map ตามตารางที่ 4.1

ตารางที่ 4.1 แสดงการจัด Memory Map ของวงจรที่ 4.1

Memory Map	Address	A1	A0
PORT_A	EQU 00H	0	0
PORT_B	EQU 01H	1	1
PORT_C	EQU 02H	0	1
Control Word	EQU 03H	0	1



รูปที่ 4.1 แสดงการอินเตอร์เฟส MCU, 8255 PPI, EPROM ภายนอก

ตัวอย่างที่ 4-1 โปรแกรมตัวอย่างการใช้งาน 8255 PORT_A ของวงจรในรูปที่ 4.1 โดยการทำงานของโปรแกรม จะทำการส่งค่า 00001111B หรือ 0FH ออกไปยัง 8255 PORT _A

```
; *** Exsemple For circuit fig 4.1 ***
PORT_A          EQU   00H
CTRL_WORD       EQU   03H
ORG             0000H
ACALL           DELAY
ACALL           CTRL_8255
MOV             A,#00001111B
MOV             R1,#PORT_A
MOVX            @R1,A
END
```

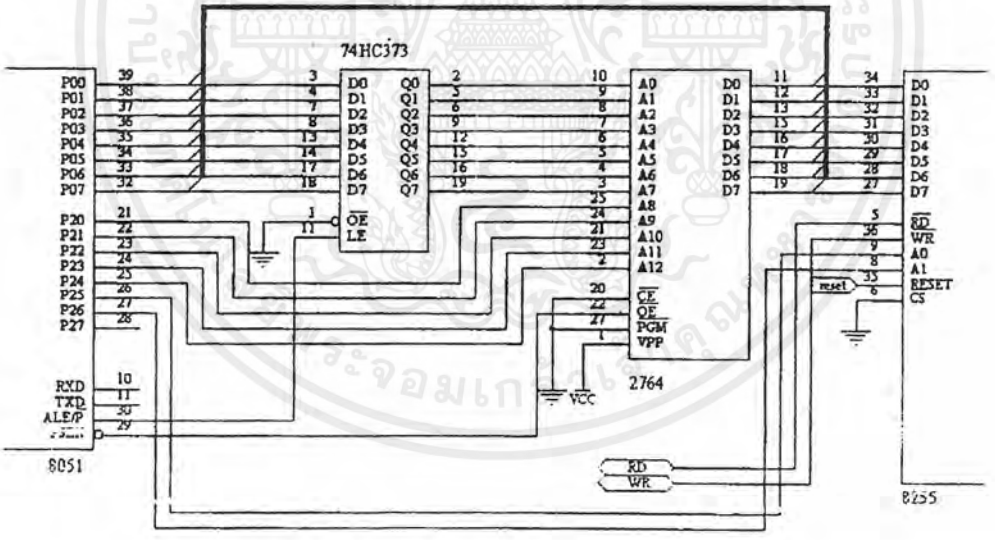
อธิบายการทำงานโปรแกรมตัวอย่างที่ 4-1

จากโปรแกรมที่ 4-1 เริ่มต้นโปรแกรมจะมีการกำหนดตัวแปรค่าคงที่ต่างๆที่จะนำมาใช้ในโปรแกรมโดยจะเห็นว่าพอร์ต A แทน 00Hและ CTRL_WORD แทน 03H จากนั้นจะเข้าสู่ Main Program โดยเริ่มต้นที่ตำแหน่ง 0000H ของหน่วยความจำโปรแกรม จะเห็นว่าเริ่มต้นจะมีการเรียกใช้ Sub Function Delay เพื่อหน่วงเวลาให้กับอุปกรณ์ภายนอกที่ต่อร่วมกับ MCU ให้ทำการเซ็ตตัวเพื่อพร้อมที่จะรับข้อมูลหรือสัญญาณต่างๆ ที่ส่งมาจาก MCU เนื่องจากกรณีการทำงานภายใน

ตัว MCU นั้น มีความเร็วในการทำงานสูงกว่าอุปกรณ์ภายนอกนั้นเองจึงอาจทำให้อุปกรณ์ที่นำมาต่อร่วมภายนอกไม่สามารถเริ่มต้นการทำงานได้ทัน โดยความเร็วของ CPU ในตัว MCU นั้นจะขึ้นอยู่กับ X-TAL ที่นำมาต่อใช้งาน ซึ่งเมื่อ X-TAL ยิ่งมีความถี่สูงมาก (ในหน่วย MHz) ก็อาจจำเป็นที่จะต้องใช้การหน่วงเวลาการทำงานให้กับ CPU ในตัว MCU ให้มากขึ้น

หลังจากการเรียกใช้ Sub Function Delay แล้ว ต่อมาจะมีการเรียกใช้ Sub Function CTRL_8255 เพื่อทำการส่ง CTRL_WORD ไปยัง 8255 เพื่อทำการเซ็ตโหมดและลักษณะพอร์ตใช้งานของ 8255 ถัดมาเป็นคำสั่งในการย้ายค่า 00001111B หรือ 0FH เก็บไว้ในแอกคิวมูเลเตอร์ จากนั้นก็จะเป็นคำสั่งในการย้ายค่า PORT_A (โดยเรากำหนดไว้ในตอนต้นโปรแกรม PORT_A ก็คือ 00H) เก็บไว้ในรีจิสเตอร์ R1(Indirect Addressing) จากนั้นก็ถึงคำสั่งสุดท้ายคือ คำสั่งเคลื่อนย้ายข้อมูลกับอุปกรณ์ภายนอกโดยใช้คำสั่ง MOVX จะเป็นการนำค่าในแอกคิวมูเลเตอร์ออกไปยังตำแหน่งที่รีจิสเตอร์ R1 ชี้อยู่ (00H) หรือ คือ การใช้งาน PORT_A ของ8255 นั้นเอง

โดยผลที่ได้นั้น ที่ PORT_A ของ 8255 จะมีค่าเป็น 00001111B หรือ 0FH ซึ่งเราสามารถใช้ LED ต่อเพื่อตรวจสอบก็ได้หรือจะตรวจสอบด้วยวิธีใช้ Multimeter ตรวจวัดก็ได้เช่นกัน โดย 0 ก็คือ 0 โวลท์ และ 1 ก็คือ 5 โวลท์นั่นเอง



รูปที่ 4.2 แสดงการประยุกต์ใช้ P2.5, P2.6 ควบคุม A0, A1 ของ 8255 โดยตรง

จากวงจรในรูปที่ 4.2 นั้นแสดงการประยุกต์โดยการนำพอร์ต 2 ที่เหลือจากการอ้างแอดเดรสไบต์สูง หรือจากการติดต่อกับ EPROM เบอร์ 2764 ทำให้เหลือขา P2.5, P2.6 และ P2.7 ของพอร์ต 2 โดยขาใช้งานที่เหลือนี้สามารถนำมาประยุกต์ใช้เป็นตัวควบคุม 8255 ที่ขา A0 และA1

โดยการต่อวงจรดังรูปที่ 4.2 ซึ่งการต่อวงจรในลักษณะนี้ในการจัด Memory Map ก็จะแตกต่างไปจาก Memory Map ของวงจรในรูปที่ 4.1 โดยจะมีการจัด Memory Map ดังแสดงในตารางที่ 4.2

ตารางที่ 4.2 แสดงการจัด Memory Map ของวงจรที่ 4.2

Memory	Address	A1(P2,6)	A0(P2,5)
PORT_A	EQU 1FFFH	0	0
PORT_B	EQU 3FFFH	0	1
PORT_C	EQU 5FFFH	1	0
Control Word	EQU 7FFFH	1	1

จาก Memory Map ในตารางที่ 4.2 จะเห็นว่าหลักการต่างๆ ในการอ้างแอดเดรสของของหน่วยความจำก็เพียงเพื่อควบคุมขา A0 และA1 ของ 8255 เท่านั้น ซึ่ง A0 และ A1 ยังคงมีหน้าที่เหมือนเดิมคือ A0 = 0 และ A1= 0 ก็คือใช้งาน PORT_A ของ 8255โดยพอร์ตอื่นๆ ก็ยังคงมีการควบคุมด้วย A0 และ A1 หมือนเดิมเปลี่ยนแต่เพียงแอดเดรสของหน่วยความจำในการอ้างถึงหรือใช้งาน

เมื่อมีการจัด Memory Map ในตารางที่ 4.2 จะเห็นว่ามีการกำหนดให้ PORT A เป็นตัวแปรที่ใช้แทนค่า 1FFFH ฉะนั้นในโปรแกรมการใช้งานการใช้คำสั่ง MOV DPTR,#PORT_A จะเป็นการให้ DTPR เก็บค่าหรือชี้ค่า 1FFFH เอาไว้ (โดยในตอนนี้ MCU จะมองค่า 1FFFH เป็นค่าคงที่ธรรมดา) จากนั้นเมื่อใช้คำสั่ง MOVX@DPTR,A จะเห็นว่าคำสั่ง MOVX นั้นเป็นคำสั่งในการติดต่อกับอุปกรณ์ภายนอกโดยผ่ายทางพอร์ต Address ของ MCU (พอร์ต 0 และ พอร์ต 2) ซึ่งจากคำสั่งข้างต้นจะเป็นการนำค่าในแอกคิวมูเลเตอร์ไปเก็บไว้ในหน่วยความจำภายนอกที่ชี้โดย DPTR (ตอนนี้ MCU จะเห็นว่าค่า 1FFFH ก็คือตำแหน่งของหน่วยความจำภายนอกนั่นเอง) ฉะนั้นก่อนที่จะทำการนำค่าในแอกคิวมูเลเตอร์ออกไปยังตำแหน่งที่ DPTR ชี้อยู่นั้น ที่พอร์ต Address (พอร์ต 0 และพอร์ต 2) ของ MCU ก็จะมีค่า 1FFFH (PORT_A) หรือ 0001 1111 1111 B ซึ่งเป็นการเซ็ตการใช้งาน 8255 PORT_A นั้นเอง จากนั้นจึงจะนำค่าในแอกคิวมูเลเตอร์ส่งออกไป ซึ่งก็เท่ากับว่าเมื่อต้องการใช้ 8255 PORT_A ก็เพียงแต่อ้างตำแหน่งของหน่วยความจำภายนอกโดยทำการกำหนดเป็นตัวแปรที่เข้าใจง่าย นั่นก็คือ PORT_A, B และ C แทนการใช้งาน 8255 PORT_A, B และ C ตามลำดับซึ่งที่อธิบายมาทั้งหมดก็คือประโยชน์ของการจัด Memory Map

เพื่อให้เข้าใจได้ง่ายขึ้น จากแอดเดรส 1FFFH จะเห็นว่าขา P2.5 , P2.6 และ P2.7 จะมีสภาวะเป็น LOW หรือ 0 ทั้งหมดเมื่อมีการอ้างแอดเดรสนี้ ซึ่งจากวงจรที่ 4.2 จะนำ P2.5 ต่อกับขา A0 และ P2.6 ต่อกับขา A1 ซึ่งเมื่ออ้างแอดเดรสข้างต้น ก็จะทำให้ขา A0 และ A1 ของ 8255 เป็น 0

ทั้งคู่ ซึ่งก็คือการเลือกใช้งาน PORT_A นั่นเอง การอ้างแอดเดรสอื่นๆก็จะมีหลักการดังที่กล่าวมาแล้วทั้งสิ้น

จากวงจรในรูป 4.2 เป็นวงจรตัวอย่างในการประยุกต์ การอ้างแอดเดรสเพื่อให้ผู้อ่านมีความเข้าใจในการจัดตำแหน่งเพื่อสร้าง Memory Map ในการใช้งานต่อวงจรในรูปแบบอื่นๆได้

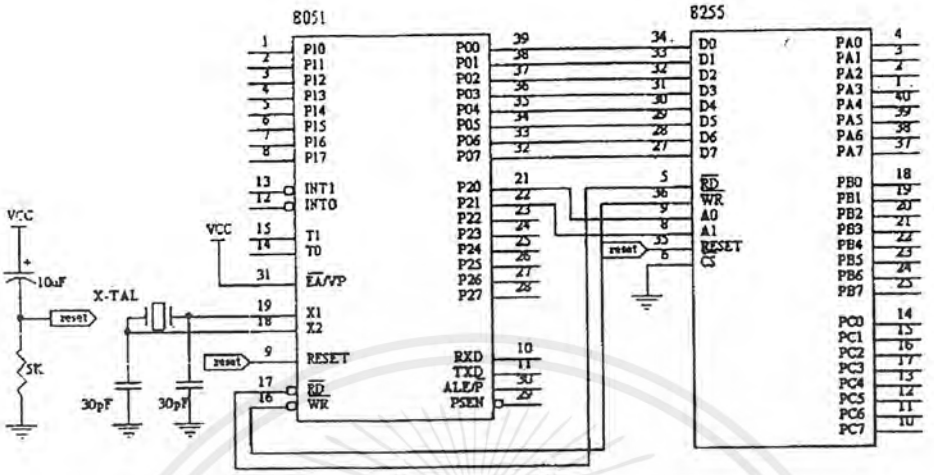
ตัวอย่างที่ 4-2 โปรแกรมตัวอย่างการใช้งาน 8255 PORT_A โดยการทำงานของโปรแกรมจะทำการส่งค่า 00001111B หรือ 0FH ออกไปยัง 8255 PORT_A เช่นเดียวกันกับโปรแกรมตัวอย่างที่ 4-1 แต่เปลี่ยนวงจรใช้งานดังวงจรใน รูปที่ 4.2 และ การจัด Memory Map ตามตารางที่ 4.2

```
; *** Exsemple For circuit fig 4.2 ***
PORT_A          EQU   1FFFH
CTRL_WORD       EQU   7FFFH
ORG             0000H
ACALL           DELAY
ACALL           CTRL_8255
MOV             A,#00001111B
MOV             DPTR,#PORT_A
MOVX            @DPTR,A
END
```

อธิบายการทำงานโปรแกรมตัวอย่างที่ 4-2

จากตัวอย่างโปรแกรมสำหรับวงจรดังรูปที่ 4.2 นั้น จะมีการทำงานเหมือนกับโปรแกรมตัวอย่างที่ 4-1 จะแตกต่างกันเพียงการจัด Memory Map โดยPORT_A จะมีการจัดที่เปลี่ยนค่าจาก 00H ในโปแกรมตัวอย่างที่ 4-1 ไปเป็น 1FFFHเท่านั้น รวมถึง CTRL_WORD

4.3 การอินเตอร์เฟส MCS-51 กับ 8255 โดยใช้ EPROM ภายใน

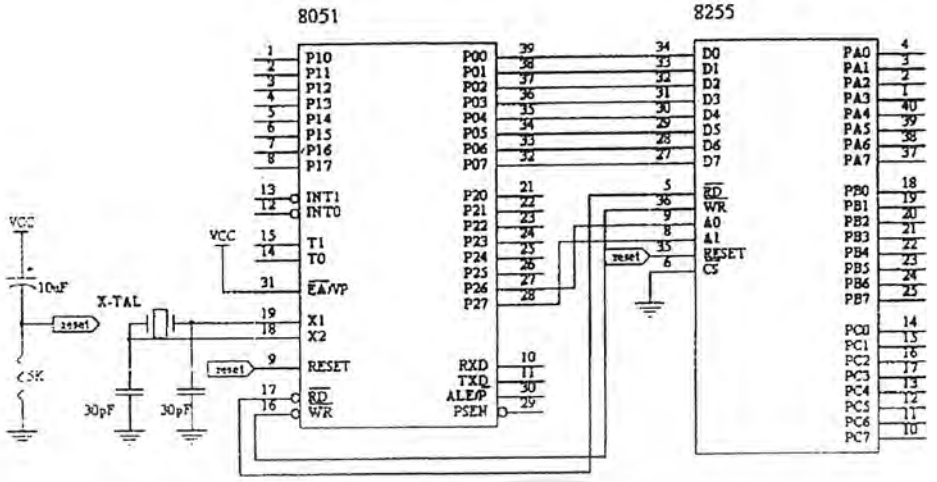


รูปที่ 4.3 แสดงการอินเตอร์เฟส MCU, 8255, EPROM ภายในรูปแบบที่ 1

ตารางที่ 4.3 แสดงการจัด Memory Map

Memory Map	Address	A1(P2,1	A0(P2,0)
PORT_A	EQU 0FFFH	0	0
PORT_B	EQU 1FFFH	0	1
PORT_C	EQU 2FFFH	1	0
Control Word	EQU 3FFFH	1	1

จากวงจรในรูปที่ 4.3 จะเห็นว่า จะใช้ PORT_0 เป็น Data Bus และใช้ PORT_2 เป็นพอร์ตในการควบคุม A0 และ A1 ของ 8255 ซึ่งจะมีการจัด Memory Map ดังตารางที่ 4.3 ซึ่งจะเห็นว่า เมื่อต่อวงจรในลักษณะนี้แล้ว PORT_2 ก็จะถูกใช้งานในการอ้างแอดเดรส ในลักษณะของ Memory Map ให้กับขา A0,A1 ของ 8255 โดยการอ้างแอดเดรสนั้นจะใช้งานทั้งพอร์ตของ PORT_2 คือ P2.0, P2.7 นั่นเองทำให้เกิดข้อจำกัดในการใช้งานพอร์ตลดลง



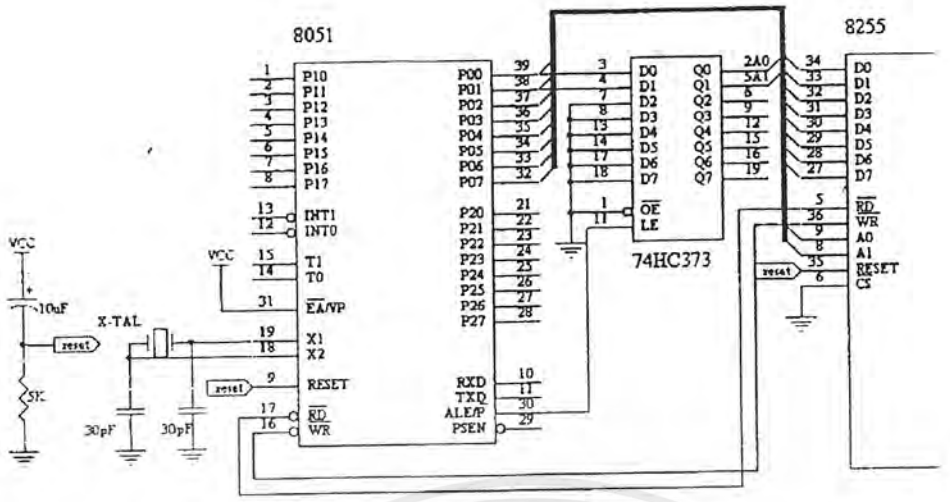
รูปที่ 4.4 แสดงการอินเตอร์เฟส MCU, 8255, EPROM ภายในรูปแบบที่ 2

จากวงจรจะไม่สามารถใช้งาน PORT_0 และPORT_2 ในลักษณะอื่นได้อีก แต่จะมีพอร์ตให้ใช้งานได้ 5 พอร์ต คือ PORT_1,PORT_3,8255 PORT_A, B และ C ซึ่งถ้าเพียงพอต่อการใช้งานแล้วก็ไม่ต้องคำนึงถึง PORT_0 และ PORT_2 อีก สามารถต่อวงจรการใช้งานตามวงจรข้างต้นได้เลย

ตารางที่ 4.4 แสดงการจัด Memory Map ของวงจรที่ 4.4

Memory Map	Address	A1(P2,7)	A0(P2,6)
PORT_A	EQU 0FFFH	0	0
PORT_B	EQU 4FFFH	0	1
PORT_C	EQU 8FFFH	1	0
Control Word	EQU CFFFH	1	1

จากวงจรในรูปที่ 4.4 จะต่างจากวงจรในรูปที่ 4.3 โดยการใช้งานของ P2.6 และ P2.7 แทน P2.0 และ P2.1 ในการควบคุมขา A0 และ A1 ของ 8255 ซึ่งถ้าใช้ P2.7 เป็น A1 และ P2.6 เป็น A0 ก็จะสามารถจัด Memory Map ได้ใหม่ดังตารางที่ 4.4 โดยเปลี่ยนแต่ Memory Map เท่านั้น แต่การทำงานในรูป 4.3 และรูปที่ 4.4 ก็ยังคงเหมือนกันซึ่งในวงจรที่ 4.4 สร้างขึ้นเพื่อให้เข้าใจการใช้ Memory Map ได้ดียิ่งขึ้น



รูปที่ 4.5 แสดงการอินเตอร์เฟส MCU, 8255, EPROM ภายในรูปแบบที่ 3

ตารางที่ 4.5 แสดงการจัด Memory Map ของวงจรที่ 4.5

Memory Map	Address	A1(P2,7)	A0(2,6)
PORT_A	EQU 00H	0	0
PORT_B	EQU 01H	0	1
PORT_C	EQU 02H	1	0
Control Word	EQU 03H	1	1

จากวงจรในรูปที่ 4.5 จะเห็นได้ว่าการนำ IC 74HC373 เข้ามาใช้ในวงจรก็เพื่อใช้ในการ LATCH แอดเดรสไบต์ต่ำ แต่จะ LATCH เพียง 2 บิตเท่านั้น คือ P0.0 และ P0.1 เพื่อใช้เป็นบิตควบคุม A0และ A1ให้กับ 8255 โดยจะทำให้มี Memory Map ในการควบคุม 8255 แสดงดังตารางที่ 4.5 ซึ่งจะต่อวงจรในรูปที่ 4.5 จะทำให้มีพอร์ตใช้งานถึง 6 พอร์ต คือ PORT 1, 2, 3, 8255 PORT A, B และ C ซึ่งการใช้งานแต่ละพอร์ต สามารถใช้งานตามคุณสมบัติของมันได้เต็มที่

จากรูปวงจรหลักๆที่กล่าวมาแล้วทั้งหมดนั้นสามารถนำไปต่อใช้งานได้จริงทุกวงจร แต่ในรูปที่ 4.1และรูปที่ 4.2 นั้น จะใช้ในการทดลองมากกว่าในการใช้งานจริงเนื่องจากสามารถใช้ EPROM Emulator ในการทดลอง (Simulator) การทำงานต่างๆได้ เมื่อทำการ Simulate ได้ผลตามความต้องการหรือบรรลุเป้าหมายแล้ว สามารถนำโปรแกรมนั้นไปทำการเปลี่ยน Memory Map ให้ตรงกับวงจรทางด้าน Hardware ที่ออกแบบ จากนั้นทำการ Down load โปรแกรมที่ถูกต้องสมบูรณ์ลง EPROM ภายในชิป MCS-51ที่มี EPROM ในตัวก็สามารถทำงานได้เหมือนกับ

โปรแกรมแม่แบบทุกประการ อย่าลืมว่าจากที่กล่าวมาทั้งหมดจะนิยมใช้วงจรในรูปที่ 4.1 และ 4.2 เป็นวงจรทดลองเพราะจำนวน Hardware ที่มาก ทั้งยังมีความซับซ้อนในการออกแบบวงจรพิมพ์ จึงไม่นิยมที่จะนำไปออกแบบในการใช้งานจริง ซึ่งเป็นข้อเสียที่สำคัญในวงจร รูปที่4.1 และ 4.2 โดยจะเห็นว่าวงจรในรูปที่ 4.3 ถึง 4.5 ก็สามารถใช้งานได้เช่นเดียวกันกับวงจรในรูปที่ 4.1 และ 4.2 ทุกประการเช่นกัน เพียงแต่ไม่สามารถใช้ EPROM Emulator ในการ Simulate เท่านั้น แต่เมื่อนำวงจรในรูปที่ 4.3 ถึง 4.5 ไปใช้งานจริงจะทำให้ประหยัดในด้านอุปกรณ์ HARDWARE ทั้งยังออกแบบลายวงจรพิมพ์ได้ง่าย สามารถนำไปใช้งานจริงร่วมกับวงจรอื่นๆได้ง่ายกว่ามาก

บทที่ 5

มาตรฐาน RS-232

5.1 กล่าวนำ

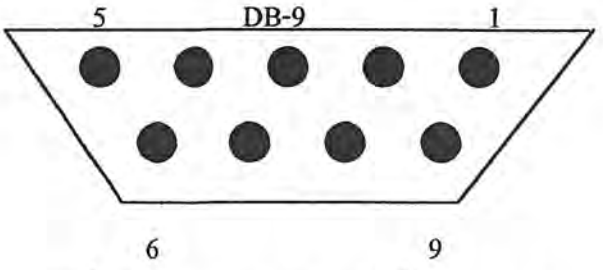
โดยทั่วไปแล้วการเชื่อมต่อพอร์ตอนุกรมจะมีมาตรฐานการเชื่อมต่ออยู่ 3 แบบ มาตราฐานที่เราพบมากที่สุดคือ มาตราฐานการเชื่อมต่อแบบ EIA เราเรียกว่า RS-232 สายส่ง RS-232 จะใช้ในหน่วยแสดงผล (Terminal), เครื่องพิมพ์, โมเด็ม และอุปกรณ์อื่นๆ ที่มีความยาวไม่เกิน 50 ฟุต

5.2 RS-232

RS-232 มีการสร้างระดับแรงดันออกมาเป็นมาตราฐานดังนี้ ถ้าระดับแรงดันมีค่าตั้งแต่ +3V ขึ้นไปเรียกว่า ลอจิก (Logic) 1 ถ้าหากระดับแรงดันมีค่าต่ำกว่า –3V เรียกว่า ลอจิก 0 ไอซีที่ใช้ส่งข้อมูล RS-232 จะใช้แหล่งจ่ายไฟที่มีแรงดัน +/- 12V และใช้สาย 1 เส้นสำหรับส่งข้อมูล ในแต่ละสายจะอ้างอิงเทียบกราวด์ (Ground) นอกจากนี้ยังมีการกำหนดสัญญาณตอบรับสำหรับควบคุมการรับส่งข้อมูลอีกด้วย

สัญญาณ RS-232 จะใช้ตัวเชื่อม (Connector) แบบ DB ชนิด 9 ขา และ 25 ขา ในการเชื่อมต่อดังรูป (ในที่นี้จะใช้ DB ชนิด 9 ขา) แม้ว่า RS-232 จะมีตัวเชื่อมต่อแต่ก็ยังไม่เป็นมาตรฐานเพราะว่าตัวเชื่อมแบบ 25 ขา จะต้องใช้พื้นที่ในการติดตั้งมาก ในคอมพิวเตอร์ส่วนบุคคล (PC) เราจึงพบตัวเชื่อมแบบ 9 ขา มากกว่า

มาตรฐาน RS-232 สามารถใช้รับส่งข้อมูลได้ไกลที่สุด 50 ฟุต ด้วยอัตราบอด (Baud Rate) 9600 ถ้าหากระยะไกลกว่านี้จะต้องส่งด้วยความเร็วที่ต่ำลง แต่ระยะใกล้เราสามารถส่งด้วยความเร็วที่สูงกว่านี้ได้ ทุกวันนี้เราอาจจะพบการรับส่งข้อมูลด้วย RS-232 ที่มีตัวขับกระแสเพิ่มทำให้ระยะในการสื่อสารมีความยาวเกินกว่า 50 ฟุตได้



รูปที่ 5.1 แสดงตำแหน่งของตัวเชื่อมแบบ DB-9

5.3 รายละเอียดของขา DB-9

ตารางที่ 5.1 แสดงรายละเอียดของขา DB-9

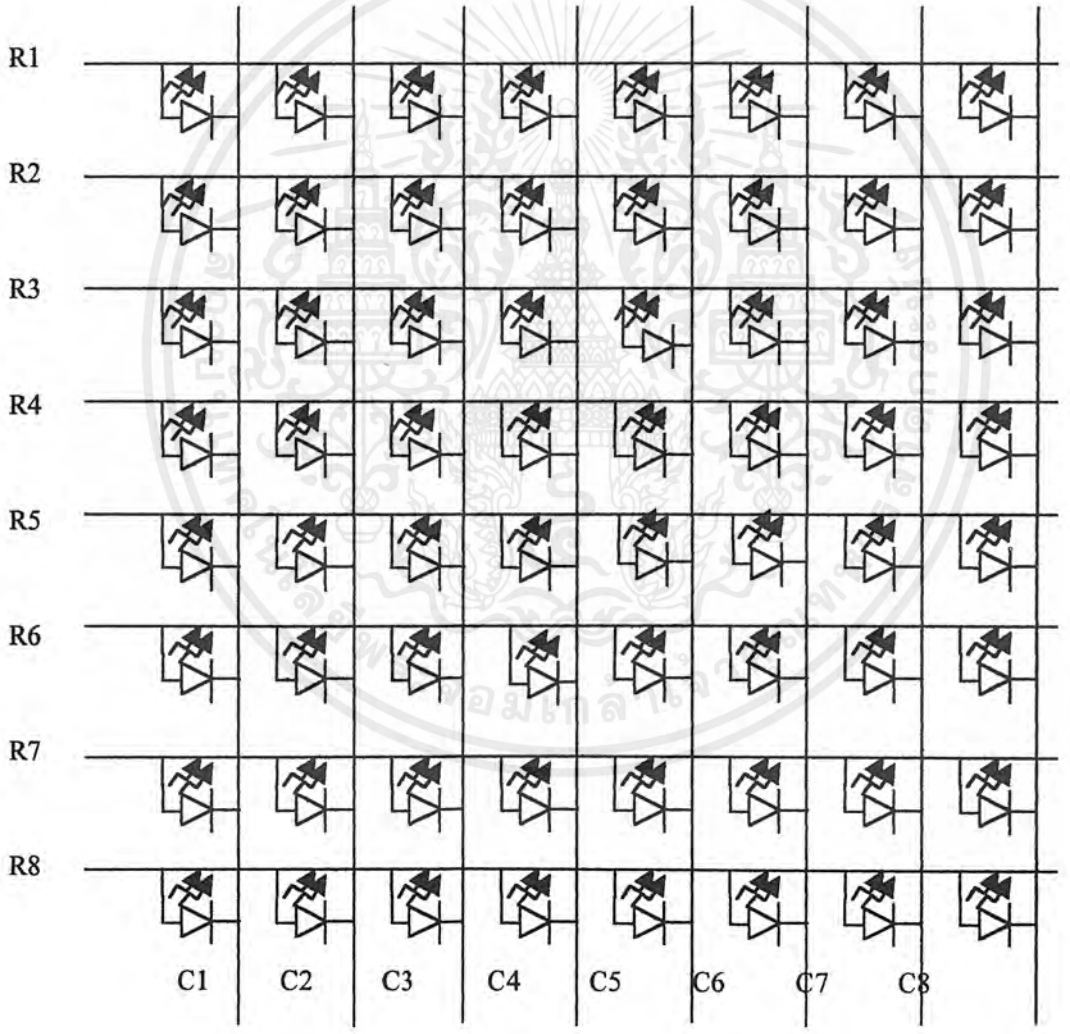
9 PIN	EIA RS-232 Circuit	RS-232 Description
5	Gnd	กราวน์ของสัญญาณ
2	RD	รับข้อมูล
3	TD	ส่งข้อมูล
1	CD	ตอบความพร้อมจากอุปกรณ์
4	DTR	แจ้งความพร้อมที่จะรับ-ส่งข้อมูล
6	DSR	พร้อมที่จะรับข้อมูลเป็นชุด
7	RTS	ต้องการจะส่งข้อมูล
8	CTS	Clear ก่อนส่งข้อมูล
9	RI	แสดงว่าอุปกรณ์ได้รับสัญญาณจากแหล่งอื่น

บทที่ 6

การแสดงผลของ MATRIX LED

6.1 โครงสร้างของ Matrix LED

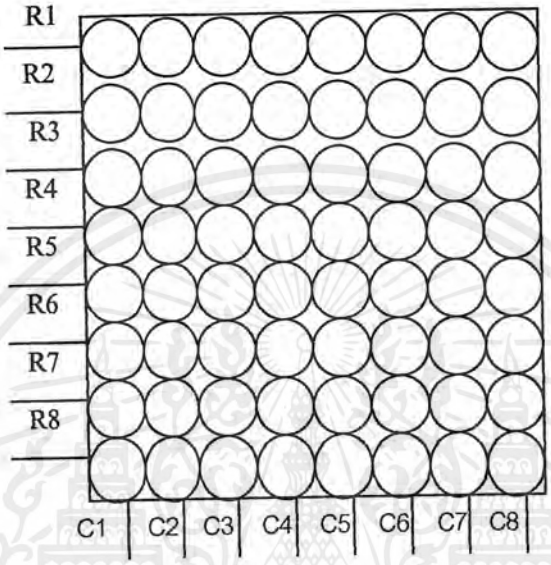
LED Matrix 1 ตัวประกอบด้วย LED 8x8 ดวง กล่าวคือ ลักษณะของ LED Matrix เป็นสี่เหลี่ยมจตุรัส มี LED ด้าน Column จำนวน 8 ดวง และมี LED ด้าน Row จำนวน 8 ดวง รวมทั้งสิ้น ใน 1 ตัว LED Matrix มี LED รวม คือ 8x8=64 ดวง



รูปที่ 6.1 แสดงการต่อภายใน Matrix LED 1 ตัว

6.2 การแสดงผล

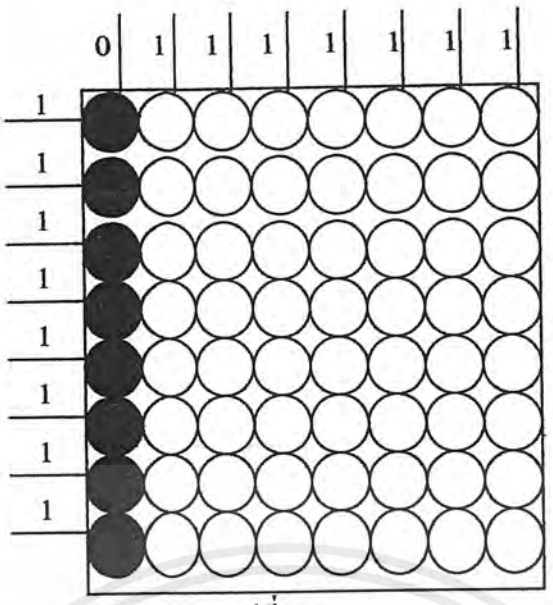
ส่วนแสดงผลของโครงงานนั้นได้นำ Matrix LED มาต่อเป็น Board ขนาดความยาวใช้ 8 ตัว Matrix LED ซึ่งเป็นด้าน Column ใน 1 ตัวมี 8 Column ดังนั้น Board แสดงผลทางด้านความยาวมีทั้งหมด 64 Column ส่วนทางด้านความกว้างใช้ 3 ตัว Matrix LED ซึ่งเป็นด้าน Row มีขนาดทั้งหมด 24 Row



รูปที่ 6.2 แสดง Matrix LED 8x8

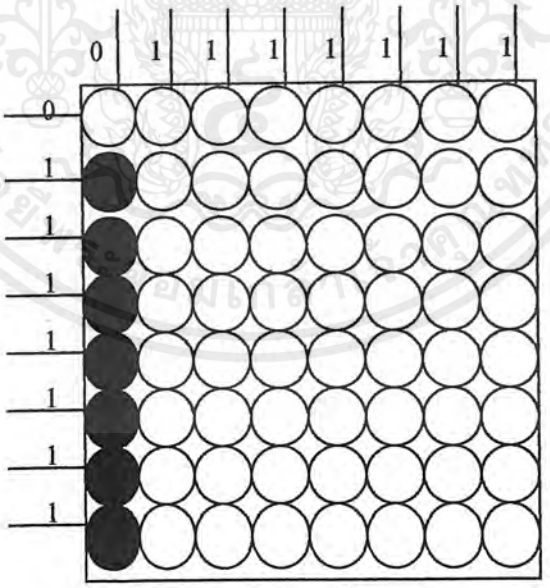
เมื่อต้องการให้ LED Matrix ติดก็ป้อนไฟบวกให้กับขา Row และป้อน ground ให้กับขา Column ซึ่งก็เหมือนกับการป้อน Logic "1" ให้ขา Row และป้อน Logic "0" ให้ขา Column จึงนำหลักการนี้มาทำการแสดงผลของ SCORE BOARD ซึ่งให้ Logic ทาง Row และ Column ควบคุมโดย CPU

ตัวอย่างที่ 6-1 เมื่อต้องการให้ LED Matrix Column ที่ 1 ติด ทั้ง Column ก็จะป้อน Logic ทาง Row เป็น Logic "1" ทุก Row และป้อน Logic "0" ให้ Column ที่ 1 ส่วนขา Column อื่นป้อน Logic "1" ก็จะทำให้ Matrix LED Column ที่ 1 ติดทั้ง Column ดังแสดงในรูปที่ 6.3



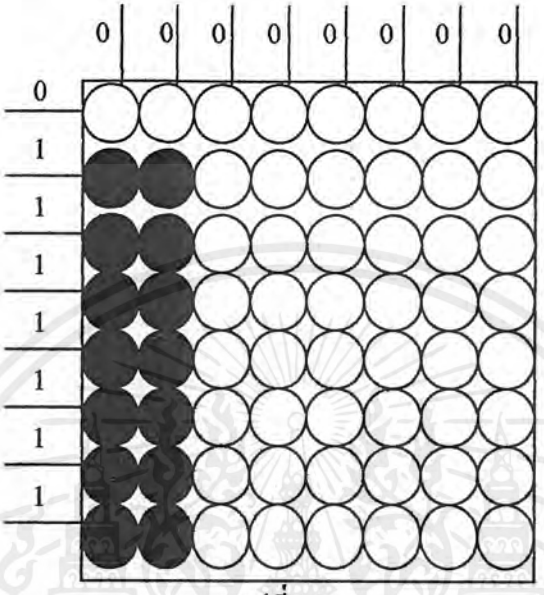
รูปที่ 6.3

จากรูปที่ 6.3 Logic "1" หมดที่ป้อนทาง Row นั้นถ้านำมาเขียนเป็นเลขฐาน 16 ก็จะได้เป็น "FFH" ซึ่งเลขฐาน 16 อันนี้เรียกว่า "Character Generator" ฉะนั้นอยากจะให้ Matrix LED Row ไหนติดหรือดับก็กำหนดได้โดย Character Generator ที่ป้อนทาง Row นี้เอง จากตัวอย่างนี้ถ้าอยากให้ Row ที่ 1 ดับ เราป้อน Character gen เป็น "7FH" ก็จะได้ Matrix LED Row ที่ 1 ไม่ติด ดังรูปที่ 6.4



รูปที่ 6.4

ส่วนทางด้าน Column นั้นเมื่อต้องการให้ Column ไหนติดก็ให้ Logic "0" ที่ Column นั้น จากตัวอย่างที่ 6-1 หากต้องการให้ Column ติดอีก Column ก็ให้ Logic "0" ที่ Column ที่ 2 อีก Column หนึ่ง Column ที่ 2 ก็จะติดดังแสดงในรูปที่ 6.5



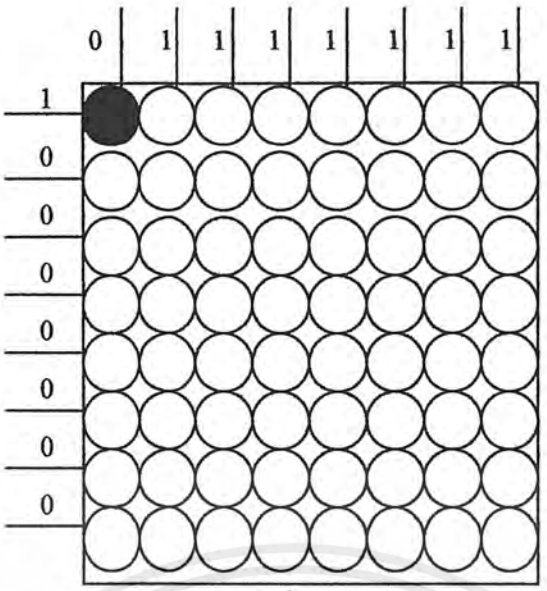
รูปที่ 6.5

จากรูป หากว่าจะให้ Column ที่ 1 และ 2 แสดงผลไม่เหมือนกันจะทำไม่ได้ เนื่องจากเราป้อน Character Gen ทาง Row เป็น "7FH" เหมือนกันทั้ง 2 Column

ฉะนั้นจะเห็นว่าเราสามารถ Control การติดของ Matrix LED ได้เพียงทีละ Column เท่านั้น หากจะให้ Column ที่ 1 กับ Column แสดงผลไม่เหมือนกัน ก็ทำได้โดยการป้อน Character Gen ทีละตัวทาง Row แล้วให้ Logic"0" ทางต้น Column ทีละ Column ดังตัวอย่าง

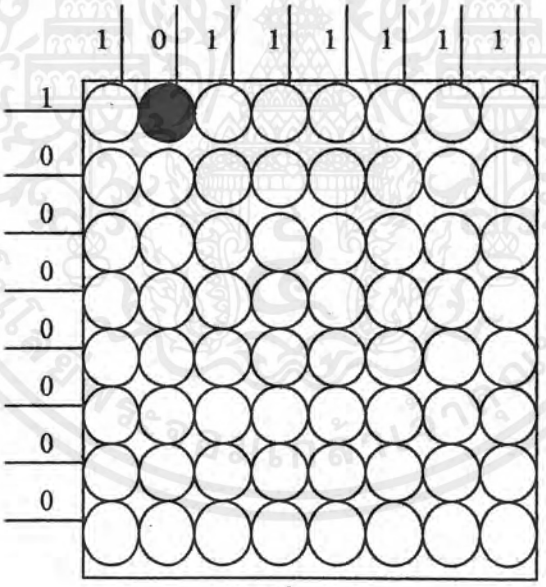
ตัวอย่างที่ 6-2 ให้ Matrix LED เป็นตัวอักษร "T" ก็จะต้องป้อน Character Gen ทาง Row และ Logic "0" ทางด้าน Column เป็นลำดับดังนี้

1. ป้อน Character Gen "01" ทาง Row และให้ Logic "0" Column ที่ 1 Matrix LED ก็จะแสดงผลดังแสดงในรูปที่ 6.6 ซึ่งจะติดกระพริบ แล้วดับไปเลย



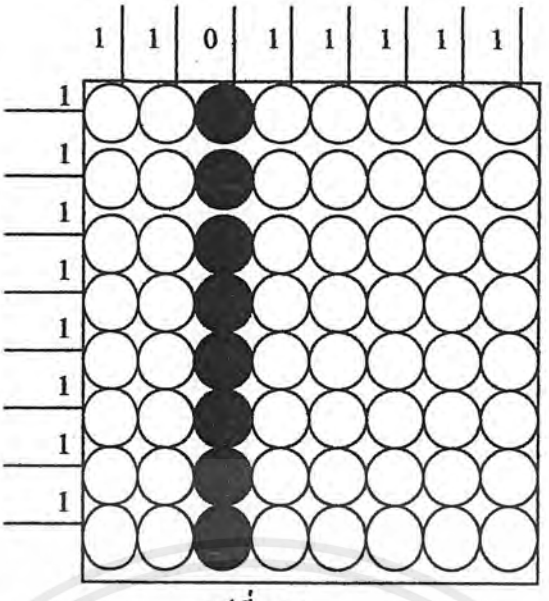
รูปที่ 6.6

2. ป้อน Character Gen "01" ทาง Row และป้อน Logic "0" ที่ Column ที่ 2 Matrix LED ก็จะแสดงผลดังรูปที่ 6.7



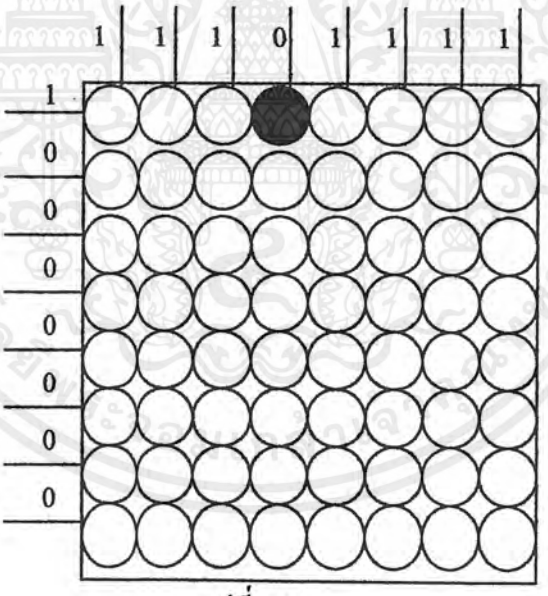
รูปที่ 6.7

3. ป้อน Character Gen "FFH" ทาง Row และป้อน Logic "0" ที่ Column ที่ 3 Matrix LED ก็จะแสดงผลดังรูปที่ 6.8



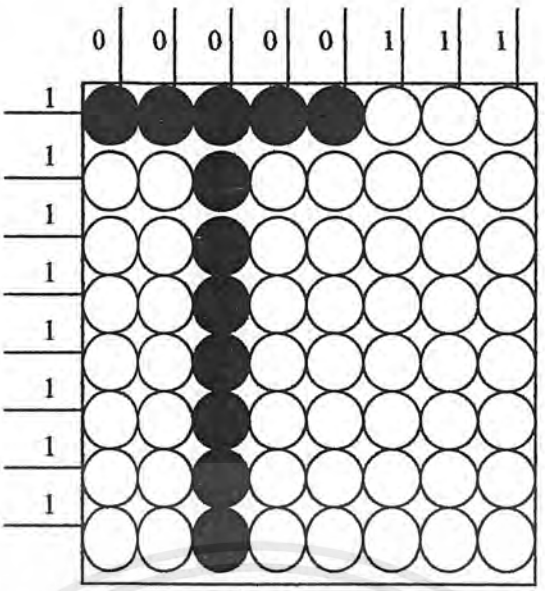
รูปที่ 6.8

4. ป้อน Character Gen "01" ทาง Row และป้อน Logic "0" Column ที่ 4 และ 5 ตามลำดับทีละ Column ก็จะได้ Matrix LED แสดงผลดังรูปที่ 6.9



รูปที่ 6.9

จะเห็นว่า Matrix LED จะติดที่ละ Column เท่านั้น แล้วจะเป็นตัว "T" ได้อย่างไร แต่จะเป็นไปได้เมื่อเราทำขั้นตอนทั้ง 4 ติดต่อกันอย่างพอเหมาะแน่นอน Matrix LED จะยังคงติดทีละ Column เหมือนเดิม ความเร็วในการแสดงผลโดยให้ CPU ควบคุมนั้นเร็วมาก จนตามนุษย์ไม่สามารถจับได้ว่า Matrix ติดทีละ Column แต่ตามนุษย์จะมองเห็นเป็นรูปตัว "T" ดังแสดงในรูปที่ 6.10



รูปที่ 6.10

แต่ก็จะติดอยู่ครู่เดียวก็ดับอีก เพราะเราป้อน Character Gen ตามขั้นตอน 1-4 เพียงเที่ยวเดียว ดังนั้นหากจะให้ติดเป็นรูปตัว "T" อยู่นานจนพอใจก็เพียงแต่ป้อน Character Gen และ Logic ทาง Row และ Column ตามลำดับ 1-4 วนหลายๆ รอบ ก็จะได้รูปตัว "T" ติดนานจนพอใจ

เพราะฉะนั้นจะเก็บ Character Gen ตัวอักษรต่างๆ ไว้ใน Memory เพื่อจะให้แสดงผลได้หลายรอบ ถ้าเป็น Matrix LED ขนาด 24 x 64 ก็จำเป็นต้องมี Character Gen 1ชุด ชุดละ 16 Row ส่วนทางด้าน Column ก็จะต้อง Rotate Logic "0" ไปทีละ Column เรื่อยๆ ทั้ง 64 Column

6.3 รูปแบบการแสดงผล

6.3.1 ตัวอักษรวิ่งไปหน้าหรือถอยหลัง

เมื่อเราสามารถทำให้ Matrix LED แสดงผลเป็นตัวอักษรอยู่กับที่แล้ว และเก็บ Character Gen ไว้ในหน่วยความจำแล้ว การที่จำทำให้ Matrix Led แสดงผลเป็นตัวอักษรวิ่งไปหน้าหรือถอยหลังได้โดยไม่ยาก ในที่นี้จะกล่าวถึงเพียงการวิ่งไปหน้าเพราะการวิ่งถอยหลังก็อาศัยหลักการเดียวกัน

การให้ตัวอักษรวิ่งไปหน้าทำได้จากการที่ Matrix LED แสดงผลเป็นตัวอักษรอยู่กับที่ ได้นานก็เพราะว่าเราป้อน Character Gen วนขบวนการเดิมหลายครั้ง หากว่าจะให้มันวิ่งเราก็เพียงแต่ให้มันวนอยู่ n ครั้งแล้วเลื่อน Character Gen ตัวที่ 2 มาแทนชุดที่ 1 และชุดที่ 3 มาแทน ชุดที่ 2 shift ขึ้นมาเรื่อยๆ

6.3.2 ตัวอักษรเลื่อนขึ้นหรือเลื่อนลง

ในตอนการให้ตัวอักษรวิ่งไปหน้าหรือถอยหลัง เราป้อน Character Gen ทาง Row และป้อน Logic "0" ทาง Column เป็น ground ในการที่จะให้ Matrix LED เลื่อนขึ้น หรือลง นี้เราเปลี่ยน Character Gen ไปป้อนทาง Column และป้อน Logic "1" เป็นไฟ ทาง Row โดยที่ Character Gen เดิมนั้น ต้องทำให้เป็น 1's Complement เสียก่อน เนื่องจากทาง Column เป็นขาลบ (Cathode) ของ LED หากให้ดวงไหนติดก็ต้องป้อน Logic "0"

ต่อจากนั้นก็ใช้วิธีการป้อน Character Gen เหมือนกันกับในกรณีการให้ตัวอักษรเลื่อนไปหน้าหรือถอยหลัง ก็จะได้รูปตัวอักษรวิ่งขึ้นหรือลงได้ตามต้องการ

บทที่ 7
ชุดรหัสคำสั่งของ MCS-51

7.1 รหัสคำสั่งของ MCS-51

ชุดรหัสคำสั่งของ 8051 มีทั้งหมด 256 คำสั่ง วิธีการแปลงภาษาแอสเซมบลีให้เป็นภาษาเครื่องสามารถดำเนินการได้ 2 แบบ คือ การทำด้วยมือและใช้โปรแกรมแอสเซมเบลอร์ เช่น SXA-51, Cross-16, Cross-32 ฯลฯ โดยจะต้องเขียนโปรแกรมแล้วพิมพ์เก็บไว้ใน File โดยตั้งชื่อให้มีนามสกุลเป็นจุด ASM โปรแกรมที่ใช้พิมพ์อาจจะใช้ SK หรือ Wordstar ก็ได้ (โปรแกรม Editor นั่นเอง)

การเขียนโปรแกรมภาษาแอสเซมบลีจะใช้ รหัสช่วยจำ (Mnemonic Code) ซึ่งประกอบด้วย Operation Code, Operand และ Comment

1. Operation Code (ตัวดำเนินการ) ประกอบด้วย 2 ถึง 4 อักษร เช่น MOV, MOVX, MOVC, ADD ฯลฯ
2. Operand (ตัวถูกดำเนินการ) เป็นชุดอักษรที่บอกการดำเนินการ อาจจะเป็นชุดเดียวหรือ 2 ชุดจะคั่นด้วยเครื่องหมาย คอมมา(,) เช่น A,#20H เมื่อเขียนให้เต็มรูปแบบของภาษาแอสเซมบลีจะได้ดังนี้

   MOV A,#20H หมายว่าให้นำ 20H เข้ารีจีสเตอร์ A

   โดยที่ A หมายถึง Destination

   #20H หมายถึง Source

   โดยที่ Source จะต้องอยู่ทางขวามือและ Destination (ข้อมูลปลายทาง)จะอยู่ทางซ้ายมือ
3. Comment เป็นส่วนขยายความหมายของชุดรหัสคำสั่งจะนำหน้าด้วยเครื่องหมาย ";" ในส่วนของ Comment จะมีหรือไม่มีก็ได้
4. นิยาม

   - คำว่าไบต์ หมายถึง จำนวนไบต์ของรหัสคำสั่ง
   - คำว่าไซเคิล หมายถึง จำนวน Machine Cycle ของรหัสคำสั่ง
   - คำว่า Source หมายถึง แหล่งข้อมูลต้นทาง
   - คำว่า Destination หมายถึง แหล่งข้อมูลปลายทาง

7.2 การแบ่งกลุ่มคำสั่งของ MCS-51

เราสามารถแบ่งได้ 5 กลุ่มใหญ่ๆ ดังนี้

- กลุ่มคำสั่งทางคณิตศาสตร์
- กลุ่มคำสั่งทางลอจิก
- กลุ่มคำสั่งในการเคลื่อนย้ายข้อมูล
- กลุ่มคำสั่งการประมวลผลแบบบลูลีน
- กลุ่มคำสั่งควบคุมลำดับการทำงาน

7.2.1 Arithmetic Instructions

คือคำสั่งที่กระทำทางคณิตศาสตร์ บวก ลบ คูณ หาร ปรับค่าเป็นเลข BCD ดังมีรายละเอียดในตารางที่ 7.1

ตารางที่ 7.1 คำสั่งทางคณิตศาสตร์ของ MCS-51

รหัสนีโมนิก	การทำงาน	แอดเดรสโหมด				แฟลกซ์		
		Dir	Ind	Reg	Imm	C	0V	AC
ADD A,<byte>	A=A+<byte>	/	/	/	/	X	X	X
ADDC A,<byte>	A=A+<byte>+C	/	/	/	/	X	X	X
SUBB A,<byte>	A=A-<byte>-C	/	/	/	/	X	X	X
INC A	A=A+1	Accumulator only						
INC <byte>	<byte>=<byte>+1	/	/	/				
INC DPTR	DPTR=DPTR=1	Data Pointer only						
DEC A	A=A-1	Accumulator only						
DEC <byte>	<byte>=<byte>-1	/	/	/				
MUL AB	B:A=BxA	ACC and B only				0	X	
DIV AB	A=Int [A/B] B=Mod [A/B]	ACC and B only				0	0	
DA A	Decimal adjust	Accumulator only						

Program Status Word

มีชื่อย่อว่า PSW ซึ่งภายในรีจีสเตอร์ตัวนี้ประกอบด้วย 2 ส่วนคือ แฟลกซ์บิตและคอนโทรลบิต

CY	AC	F0	RS1	RS0	0V	F1	P

รูปที่ 7.1 Program Status Word (PSW)

แฟลกซ์บิต	- Carry Bit (CY) - Auxiliary Carry ใช้สำหรับบวกลบเลข BCD (AC) - Overflow bit (0V) - Parity bit (P)
C Flag	จะเซ็ท เมื่อเกิด Carry จากบิต D7 ออกไป
0V Flag	จะเซ็ทเมื่อมีการทดจากบิต D7 ออกไป แต่ไม่มีการทดจากบิต D6 ไป D7 หรือ มีการทดจากบิต D6 ไปบิต D7 แต่ไม่มีการทดออกจากบิต D7 ออกไป
AC Flag	จะเซ็ทเมื่อมีการทดหรือยืมระหว่างบิต D3 และ บิต D4
P Flag	จะเซ็ทเมื่อจำนวนเลข 1 ใน A เป็นเลขคู่ เช่น 0000 0000b หรือ 0000 0011b หรือ 0000 1111b และในทางกลับกันจะเป็นศูนย์
F0 Flag	แฟลกซ์สถานะที่ใช้งานทั่วไปเบอร์ 0
F1 Flag	แฟลกซ์สถานะที่ใช้งานทั่วไปเบอร์ 1

คอนโทรลเลอร์บิต	- Register Bank Select 0 (RS_0) - Register Bank Select 1 (RS_1)
(RS_1), (RS_0)	ใช้เลือกรีจีสเตอร์แบงค์จะเลือกได้ทั้งหมด 4 แบงค์

(RS_1)	(RS_0)	แบงค์
0	0	0
0	1	1
1	0	2
1	1	3

ในแต่ละแบงค์จะมีรีจีสเตอร์ R_0 จนถึง R_7 จำนวน 8 ตัว

MUL AB หมายถึงนำค่า 8 บิตในรีจีสเตอร์ A คูณกับค่า 8 บิต ในรีจีสเตอร์ B ผลลัพธ์ที่ได้คือ 16 บิต โดย 8 บิตล่างเข้าเก็บใน A และ 8 บิตบนเข้าเก็บใน B

เช่น MUL AB
ถ้า A = 02H B = 04H จะได้ (02 x 04 = 00 08H) A = 08H B = 00H CY = 0 OV = 0
ถ้า A = 0FFH B = 02H จะได้ (255 x 02 = 01 00H) A = 00H B = 01H CY = 0 OV = 1
OV = 0 เพราะคูณแล้วค่าที่ได้ไม่เกิน 8 บิต ถ้าเกินจะเป็น 1

DIV AB หมายถึง เอาค่า 8 บิตในรีจีสเตอร์ A ตั้งหารด้วยค่า 8 บิต ในรีจีสเตอร์ B จะได้ผลลัพธ์จำนวนเต็มเก็บใน A และ เศษเก็บในรีจีสเตอร์ B ส่วน CY = 0 และ OV = 0

DA A คือคำสั่งปรับค่า Hex ให้เป็น BCD ส่วนมากจะใช้หลังจากทำคำสั่ง ADD หรือADDC

7.2.2 Logical Instruction

ในตารางที่ 7.2 แสดงชุดคำสั่งทางลอจิกของ MCS-51 (MCS-51 Logical Instructions) คำสั่งชุดนี้แสดงการประมวลผลทางบลูลีน (Boolean Operation) เช่น คำสั่ง AND, OR, XOR, NOT, Complement, Rotate และ SWAP โดยกระทำบิตต่อบิต ส่วนมากจะกระทำกับรีจิสเตอร์ A แต่มีบางคำสั่งกระทำกับค่าในหน่วยความจำ

รูปแบบ ANL A,<byte> ดังแสดงได้ดังต่อไปนี้

ANL	A,7FH	(direct addressing)
ANL	A,@RI	(indirect addressing)
ANL	A,R6	(register Instruction)
ANL	A,#20H	(immediate constant)

เวลาที่สูญเสียในการ Execute ในแต่ละคำสั่งถ้าทำกับรีจีสเตอร์ A จะใช้ 1 uS แต่ถ้าไม่ใช่จะใช้ 2 uS

ข้อสังเกตุ การประมวลผลทางบลูลีน จะกระทำกับค่าในหน่วยความจำข้อมูล 128 ไบต์ล่างสุด (Lower 128 bytes Internal RAM) และ SFR เท่านั้น (ใช้ direct addressing เท่านั้น) คำสั่งที่สามารถเปลี่ยนค่าใน Port เป็นตรงข้ามจะใช้คำสั่ง เช่น XRL P1,#0FFH

ตารางที่ 7.2 การกระทำทางลอจิก

รหัสนีโมนิก	การทำงาน	แอดเดรสโหมด				ไซเคิล
		Dir	Ind	Reg	Imm	
ANL A,<byte>	A=A AND <byte>	/	/	/	/	1
ANL <byte>,A	<byte>=<byte> AND A	/				1
ANL <byte>,#data	<byte>=<byte> AND #data	/				2
ORL A,<byte>	A = A OR <byte>	/	/	/	/	1
ORL <byte>,A	<byte>=<byte> OR A	/				1
ORL <byte>,#data	<byte>=<byte> OR #data	/				2
XRL A,<byet>	A = A XOR <byte>	/	/	/	/	1
XRL <byte>,A	<byte> =<byte> XOR A	/				1
XRL <byte>,#data	<byte>=<byte>XOR #data	/				2
CRL A	A = 00H	Accumulator only				1
CPL A	A = NOT A	Accumulator only				1
RL A	เลื่อนค่าใน A ไปทางซ้าย 1 บิต (Rotate ACC left 1 bit)	Accumulator only				1
RLC A	เลื่อนค่าใน A ไปทางซ้าย 1 บิตผ่าน CY (Rotate left througt Carry)	Accumulator only				1
RR A	เลื่อนค่าใน A ไปทางขวา 1 บิต (Rotate ACC right 1 bit)	Accumulator only				1
RRC A	เลื่อนค่าใน A ไปทางขวา 1 บิตผ่าน CY (Rotate Right througt Carry)	Accumulator only				1
SWAP A	สลับค่าใน A ระหว่าง 4 บิตล่างและ 4 บิตบน (Swap nibbles in A)	Accumulator only				1

Rotate คือการหมุนเลื่อนข้อมูลในรีจีสเตอร์ A มีทั้งไปทางซ้ายและขวาโดยผ่านCYและไม่ผ่านCY

เช่น RR A หมายถึง Rotate ค่าใน A ไปทางขวา โดยนำค่าใน D0 ย้อนมาเข้า D7

RL A หมายถึง Rotate ค่าใน A ไปทางซ้าย โดยนำ D7 มาแทน D0

RRC หมายถึง Rotate ค่าใน A ไปทางขวา โดยนำค่าใน D0 เข้า CY นำค่าใน CY เข้า D7

RLC A หมายถึง Rotate ค่าใน A ไปทางซ้าย โดยนำค่าใน D7 เข้า CY นำคาใน CY เข้า D0

SWAP A หมายถึง สลับค่าในรีจีสเตอร์ A ระหว่าง 4 บิตล่างกับ 4 บิตบน

CPL A หมายถึง สลับค่าในรีจีสเตอร์ A เป็นตรงข้าม

CLR A หมายถึง เคลียร์ค่าในรีจีสเตอร์ A (A=00H)

7.2.3 การเคลื่อนย้ายข้อมูล

ใช้ในการเคลื่อนย้ายข้อมูลโดยแบ่งเป็น 2 กลุ่มใหญ่ดังนี้

- การเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำข้อมูล (Data Memory)
- การเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำโปรแกรม (Program Memory)

7.2.3.1 การเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำข้อมูล

แบ่งได้เป็น 2 กลุ่มคือการเคลื่อนย้ายข้อมูลระหว่าง Internal RAM และ External RAM ดังตารางที่ 7.3 และตารางที่ 7.4

ตารางที่ 7.3 การเคลื่อนย้ายข้อมูลภายใน (Internal Data Memory Moves)

รหัสนีโมนิก	การทำงาน	แอดเดรสโหมด				ไซเคิล
		Dir	Ind	Reg	Imm	
MOV A,<src>	A = <src>	/	/	/	/	1
MOV <dest>,A	<dest> = A	/	/	/		1
MOV <dest>,<src>	<dest> = <src>	/	/	/	/	2
MOV DPTR,#data 16	DPTR = 16 bit immediate Constant				/	2
PUSH <src>	PUSH <src> เข้า stack ที่ (SP+1)	/				2
POP <dest>	POP ค่าในstackเข้า<dest>แล้วลดSP	/				2
XCH A,<byte>	แลกเปลี่ยนค่าระหว่าง A และ <byte>	/	/	/		1
XCHD A,@Ri	แลกเปลี่ยนค่า 4 บิตล่างระหว่าง A และค่าในหน่วยความจำที่ระบุโดย Ri		/			1

7.2.3.2 การเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำข้อมูลภายใน

ตารางที่ 7.3 แสดงคำสั่งทั้งหมดที่สามารถเคลื่อนย้ายข้อมูลใน Internal RAM ถ้ากระทำผ่านรีจีสเตอร์ A จะใช้เวลาในการ Execute เพียง 1 uS แต่ถ้าไม่ใช่จะใช้ 2 uS (คิดที่ CPU Clock เท่ากับ 12 MHz)

คำสั่ง MOV <dest>,<src> โดยที่ <dest> หมายถึงแหล่งข้อมูลปลายทาง <destination> และ <src> หมายถึงแหล่งข้อมูลต้นทาง (source) คำสั่งนี้มีผลให้ข้อมูลใน <src> ถูก Copy เข้าเก็บใน <dest> คำสั่งที่กล่าวมานี้เป็นการเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำ 2 ตำแหน่งสามารถใช้กับ Lower 128 bytes ใน Internal RAM (ใช้ได้ทั้ง Direct และ Indirect addressing Mode) ส่วนพื้นที่ที่เกิน 128 ไบต์แรกต้องใช้ Indirect addressing Mode เท่านั้น ส่วน SFR ซึ่งอยู่เหนือ 128 ไบต์แรกต้องใช้ Direct Addressing Mode

7.2.3.3 การเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำข้อมูลภายนอกชิป

การเคลื่อนย้ายข้อมูลจะระบุผ่านรีจีสเตอร์ 8 บิต และ 16 บิต ดังตารางที่ 7.4

ตารางที่ 7.4 การเคลื่อนย้ายข้อมูลระหว่างหน่วยความจำข้อมูลภายนอกชิป

แอดเดรส(บิต)	รหัสนีโมนิก	การทำงาน	ไซเคิล
8	MOVX A,@Ri	นำค่าในหน่วยความจำสำหรับเก็บข้อมูลภายนอกชิป ที่ระบุโดยตำแหน่ง Ri เข้า A	2
8	MOVX @Ri,A	นำค่าใน A ออกไปยังหน่วยความจำสำหรับเก็บข้อมูลภายนอกชิปที่ระบุโดยตำแหน่งโดย Ri	2
16	MOVX A,@DPTR	นำค่าในหน่วยความจำสำหรับเก็บข้อมูลภายนอกชิป ที่ระบุโดยตำแหน่งโดย DPTR เข้า A	2
16	MOVX @DPTR,A	นำค่าใน A ออกไปยังหน่วยความจำสำหรับเก็บข้อมูลภายนอกชิปที่ระบุโดยตำแหน่งโดย DPTR	2

คำสั่งเคลื่อนย้ายข้อมูล External Data Memory Moves จะใช้แบบ Indirect เท่านั้น ถ้าระบุแอดเดรส 8บิตระบุผ่านรีจีสเตอร์R_iซึ่งก็คือR_0,R_1ถ้าระบุแอดเดรส16บิตจะระบุผ่านรีจีสเตอร์ DPTR

7.2.4 Boolean Instructions

ตารางที่ 7.5 กลุ่มคำสั่งประมวลผลแบบบลูบีน (Boolean Instruction)

รหัสนีโมนิก	ความหมาย	การทำงาน	ไบต์	ไซเคิล
SETB C	Set Carry Flag	CY = 1	1	1
SETB bit	Set direct bit	Bit = 1	2	1
CLR C	Clear Carry Flag	CY = 0	1	1
CLR bit	Clear direct bit	Bit = 0	2	1
CPL C	Complement Carry Flag	CY = CY\	1	1
CPL bit	Complement direct bit	Bit = bit\	2	1
MOV C,bit	Move direct bit to Carry Flag	ย้ายข้อมูลในตำแหน่ง bit เข้า CY	2	1
MOV bit,C	Move Carry Flag to direct bit	ย้ายข้อมูลใน CY เข้าตำแหน่ง bit	2	1
ANL C,bit	AND Carry Flag to direct bit	ข้อมูลในตำแหน่ง CY AND กับ bit	2	2
ANL C,/bit	AND Carry Flag to not direct bit	ข้อมูลในตำแหน่ง CY AND กับ not bit	2	2
ORL C,bit	OR Carry Flag to direct bit	ข้อมูลใน CY OR กับข้อมูลในตำแหน่ง bit	2	2
ORL bit,C	OR Carry Flag to not direct bit	ข้อมูลใน CY OR กับข้อมูลในตำแหน่ง not bit	2	2
JC rel	Jump if Carry Flag is set	จะกระโดดไปถ้า CY=1	2	2
JNC rel	Jump if No Carry Flag	จะกระโดดไปถ้า CY=0	2	2
JB bit,rel	**Jump if direct bit is Set**	จะกระโดดไปถ้าค่าในตำแหน่ง bit =1	3	2
JNB bit,rel	**Jump if direct bit is Not Set**	จะกระโดดไปถ้าค่าในตำแหน่ง bit =0	3	2
JBC bit,rel	**Jump if direct bit is Set and Clear direct bit**	จะกระโดดไปถ้าค่าในตำแหน่ง bit =1 และจะเคลียร์ค่าใน bit นั้นทิ้ง	3	2

ตารางที่ 7.5 แสดงให้เห็นถึงชุดคำสั่งประมวลผลทางบลูลีน (Boolean Instruction) สามารถทำงานในระดับบิตได้ (bit level operation) พื้นที่ใน Internal RAM ที่สามารถเข้าถึงข้อมูลทีละบิตได้ คือตั้งแต่แอดเดรส (20-2FH) ซึ่งแบ่งเป็นจำนวนบิตได้ 128 บิต ($16x8=128_{10}$) คือตั้งแต่บิต (00 จนถึง0FFH) และอีกพื้นที่หนึ่งสามารถเข้าถึงข้อมูลได้คือ รีจีสเตอร์ใน SFR

7.2.5 การควบคุมลำดับการทำงานของโปรแกรม (Program branching)

7.2.5.1 Unondition Jump

จะ Jumps ไปยังตำแหน่งต่างๆ โดยไม่ต้องตรวจสอบก่อน

ตารางที่ 7.6 กลุ่มคำสั่งควบคุมลำดับการทำงานแบบไม่มีเงื่อนไข (Uncondition Jump)

คำสั่งควบคุมลำดับการทำงานของโปรแกรมแบบไม่มีเงื่อนไข (Uncondition Jump)		
นิวโมนิค	การทำงาน	ไซเคิล
JUMP addr เช่น AJMP addr 11 เช่น LJMP addr 16 เช่น SJMP rel	กระโดดไปยังแอดเดรส addr มีอยู่ 3 รูปแบบ กระโดดไปยังแอดเดรสในช่วง 2^{11} ตำแหน่ง กระโดดไปยังแอดเดรสในช่วง 2^{16} ตำแหน่ง กระโดดไปยังแอดเดรสในช่วง (-128 ถึง +126)	2
JMP @A+DPTR	กระโดดไปยังแอดเดรสที่ระบุโดยค่าใน (A + DPTR)	2
CALL addr เช่น ACALL addr 11 เช่น LCALL addr 16	เรียกโปรแกรมย่อยที่แอดเดรส addr มีอยู่ 2 รูปแบบ เรียกโปรแกรมย่อยในช่วง 2 ตำแหน่ง เรียกโปรแกรมย่อยในช่วง 2 ตำแหน่ง	2
RET	กระโดดกลับจากโปรแกรมย่อย	2
RETI	กระโดดกลับจากโปรแกรมย่อยของอินเตอร์รัพต์	2
NOP	ไม่มีการทำงานใดๆ (No Operation)	1

7.2.5.2 คำสั่งควบคุมลำดับการทำงานแบบมีเงื่อนไข (Condition Jump)

ตารางที่ 7.7 กลุ่มคำสั่งควบคุมลำดับการทำงานแบบมีเงื่อนไข (Condition Jump)

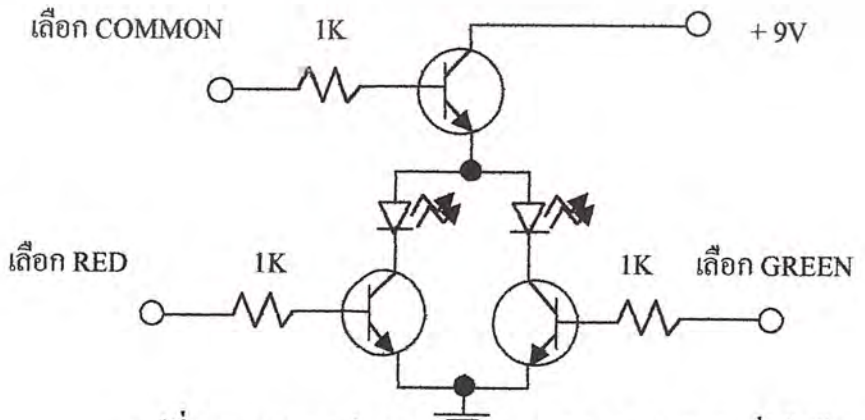
นิวโมนิค	การทำงาน	แอดเดรสโหมด				CY	ไซเคิล
		Dir	Ind	Reg	Imm		
JZ relเช่น JZ loop	Jump ถ้า A = 0	Accumulator only					2
JNZ rel เช่น JNZ loop	Jump ถ้า A≠ 0	Accumulator only					2
DJNZ <byte>,rel เช่น DJNZ 30H,loop เช่น DJNZ R7,loop	ลดค่าใน <byte> ลงหนึ่งถ้า ≠ 0 Jumps ไปยัง (PC) + 2 + rel	/		/			2
CJNE A,<byte>,rel เช่น CJNE A,30H,loop เช่น CJNE A,#30H,loop	ลดค่าใน <byte> ลงหนึ่งถ้า≠0 Jumps ไปยัง (PC)+3+rel ถ้า A น้อยกว่า<byte>ได้CY=0	/			/	X	2
CJNE<byte>,#data,rel เช่น CJNE Rn,#22H,loop เช่นCJNE@Ri,#22H,loop	เปรียบเทียบค่าใน<byte>กับ #data ถ้า ≠0 Jumpไปยัง (PC+rel) ถ้า<byte>น้อยกว่าdataได้CY=0		/	/		X	2

บทที่ 8
การทดลองและข้อกำหนดการใช้งานของวงจร

8.1 ลักษณะและรูปร่างของชิ้นงาน

รูปที่ 8.1 แสดงส่วนประกอบของชิ้นงานก่อนนำไปประกอบกับคอมพิวเตอร์

8.2 ส่วนของ Supply



รูปที่ 8.2 แสดงการต่อวงจรของ LED ของ Matrix LED ที่ใช้สร้างแผง Display

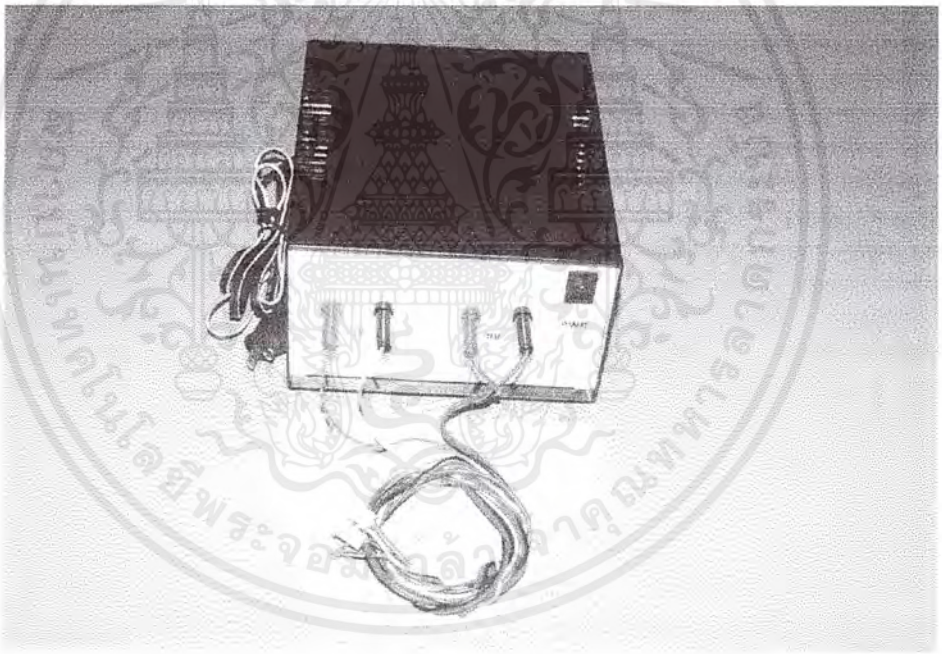
ตารางที่ 8.1 แสดงการทดลองหาค่า Rc ของส่วน Display

แรงดัน	Rc=1KΩ	Rc=2KΩ	Rc=3KΩ	Rc=4KΩ	Rc=5KΩ
9V	7mA	3.5mA	2.5mA	1.7mA	1.4mA
12V	9.5mA	5mA	3.3mA	2.4mA	2mA

จากการทดลองจะได้ว่าภายใน Matrix LED 1ตัวจะมี LED =8x8=64 ดวง แต่ละดวงกินกระแส = 1.4 mA Display นี้ใช้ Matrix LED ขนาด 3x8 จะมีจำนวน LED = 24x64=1536 ดวง ถ้า LED ทั้งหมดติดพร้อมกันจะกินกระแสทั้งหมด = 1536x2mA = 2.15 A

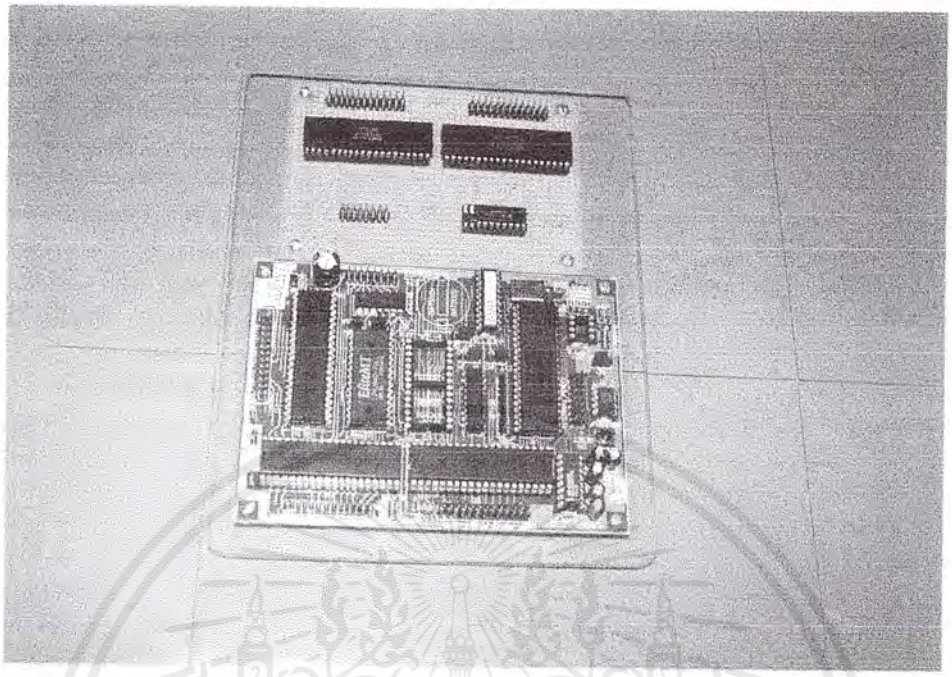
Power Supply จะมี 2 ขนาดคือ 9V 2.15 A สำหรับDisplay และ 5V 0.5A สำหรับเลี้ยงบอร์ดไมโครคอนโทรลเลอร์ MCS-51

ดังนั้นต้องสร้าง Power Supply จ่ายไฟ 9V และ 5V กระแส = 3A



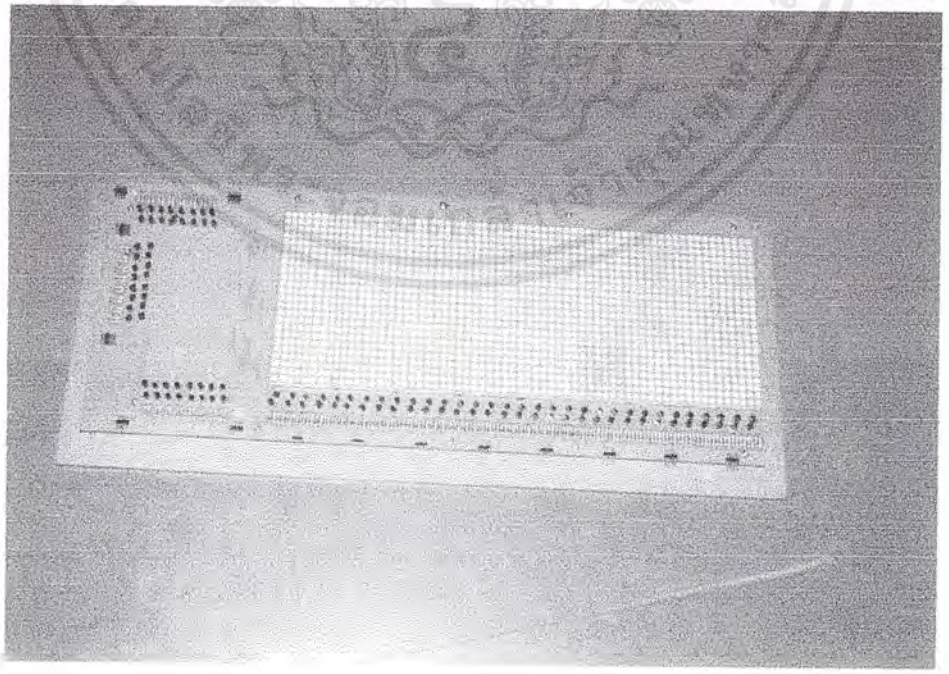
รูปที่ 8.3 Power Supply

8.3 ชุดไมโครคอนโทรลเลอร์ MCS-51



รูปที่ 8.4 แสดงชุดไมโครคอนโทรลเลอร์ MCS-51 และการต่อ RAM เพิ่ม

8.4 แผงแสดงผล



รูปที่ 8.5 แสดงแผงแสดงผล (Display Board)

8.5 หน้าที่และการทำงานของส่วนประกอบต่างๆ

มีด้วยกัน 3 ส่วนประกอบหลักๆ ดังนี้

8.5.1 คอมพิวเตอร์

คอมพิวเตอร์ทำหน้าที่รับคำสั่งจาก Key Borad ผ่านทางทางจอของ Program visual basic แล้วส่ง Data คำสั่ง ไปควบคุม ชุดไมโครคอนโทรลเลอร์ MCS-51 ผ่านทาง Serial Port โดยใช้มาตรฐานการส่ง RS-232

8.5.2 ชุดไมโครคอนโทรลเลอร์

รับคำสั่งจากคอมพิวเตอร์ แล้วส่ง Data ไปแสดงผลทางจอ Display ตามคำสั่งที่รับเข้ามาจากผู้ใช้ คำสั่งที่รับเข้ามาชุดไมโครคอนโทรลเลอร์จะรักษาสภาพการทำงานตามคำสั่งนี้ไปเรื่อยๆ จนกว่าจะมีการแก้ไขคำสั่งการแสดงผลผ่านทางคอมพิวเตอร์ใหม่

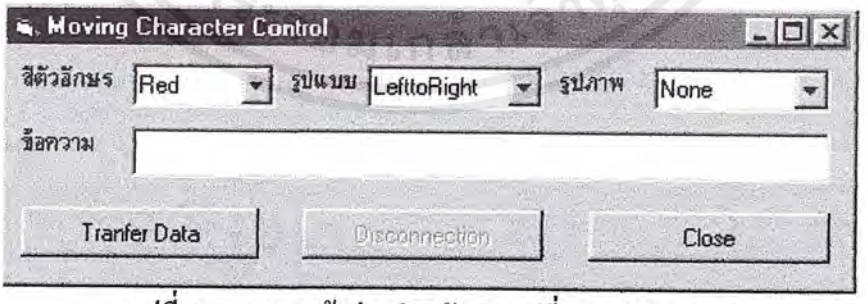
8.5.3 ชุดแผงแสดง LED

แผงแสดงผลขนาด 24 x 64 ดวง ทำหน้าที่ แสดงผลตาม Data ที่ชุดไมโครคอนโทรลเลอร์สั่งงานมา

8.6 การใช้งาน

คู่มือการใช้งาน

1. สังเกตุหน้าจอว่าอยู่ในสภาพที่พร้อมที่จะ Tranfer Data หรือไม่ ดังรูปที่ 8.6 ถ้าไม่พร้อม ดังรูปที่ 8.7 ก็ให้ทำการ Disconnection หน้าจอก็กลับมาพร้อมที่จะ Tranfer Data



รูปที่ 8.6 แสดงหน้าต่างสำหรับการเปลี่ยนแปลงการแสดงผล

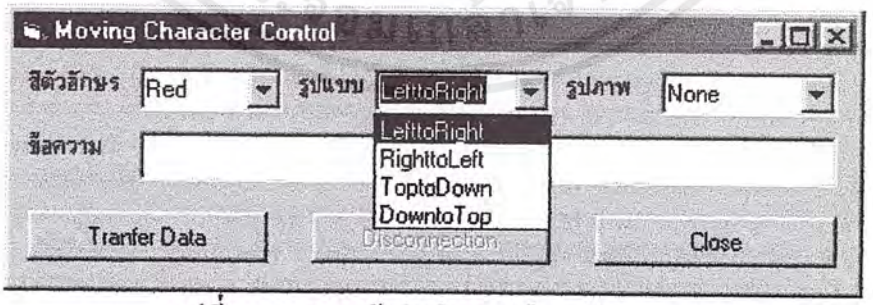
รูปที่ 8.7 แสดงหน้าต่างที่ยังไม่พร้อมที่จะส่งข้อมูล

2. ป้อนข้อความเป็นภาษาไทยหรือภาษาอังกฤษ โดยเลือกภาษาได้จาก Keyboard

3. เลือกสีข้อความที่จะแสดงผล เป็นสีแดงหรือสีเขียว โดยคลิกเลือกที่สีตัวอักษร ดังรูปที่ 8.8

รูปที่ 8.8 แสดงหน้าต่างในการเลือกสีของตัวอักษร

4. เลือกรูปแบบของข้อความที่จะแสดงผลว่าจะให้มีรูปแบบใด ขวาไปซ้าย ซ้ายไปขวา บนลงล่าง หรือล่างขึ้นบน ดังรูปที่ 8.9



รูปที่ 8.9 แสดงหน้าต่างในการเลือกรูปแบบการแสดงผล

5. เลือกรูปที่จะแสดงผลก่อนข้อความ มีอยู่ด้วยกัน 5 รูป หรือไม่ต้องมีรูปแสดงผลก็ได้ ดังรูปที่ 8.10

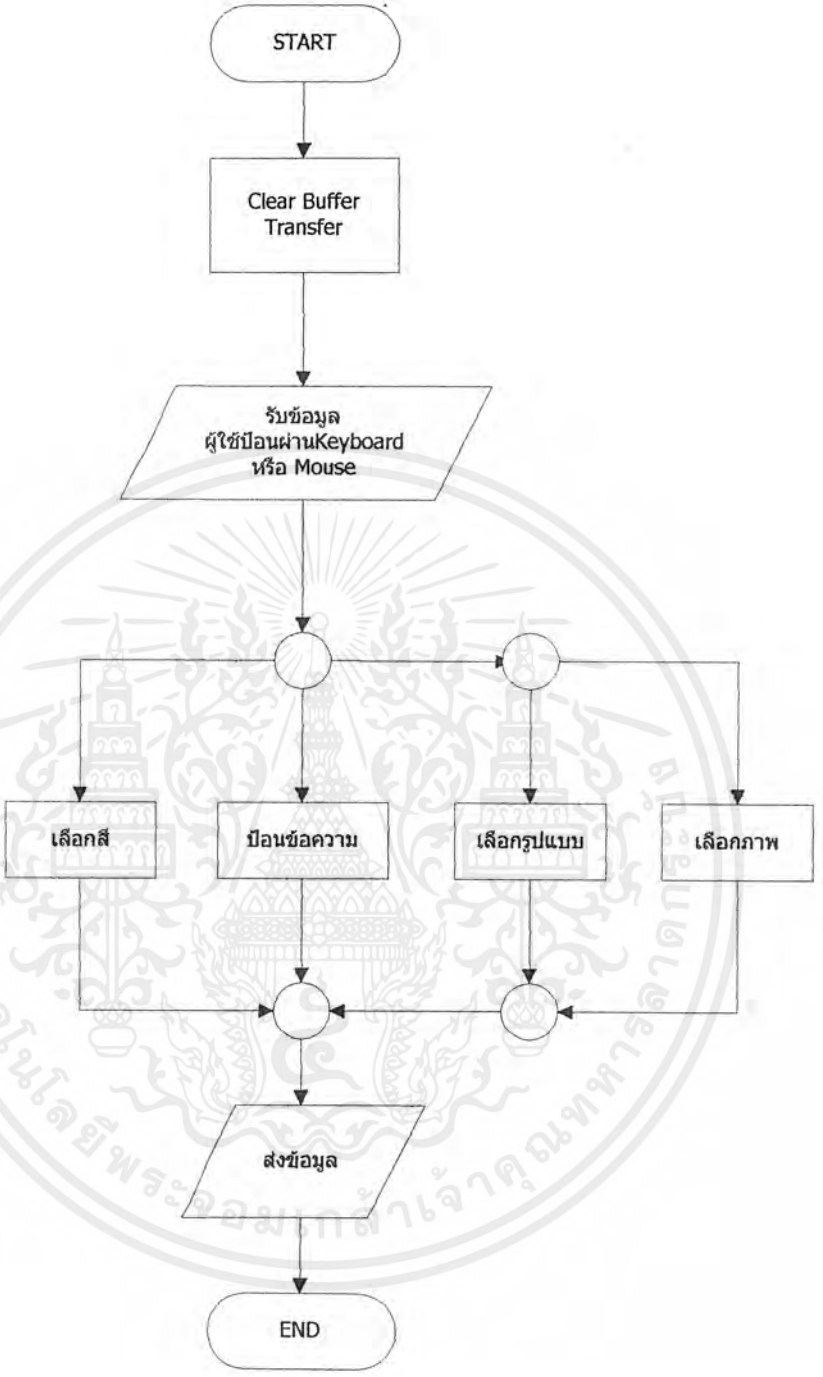
รูปที่ 8.10 แสดงหน้าต่างการเลือกรูปภาพ

6. กด Tranfer Data เพื่อส่งข้อมูลออกไป หน้าต่างการเปลี่ยนแปลงข้อมูลก็จะแสดงดังรูปที่ 8.7

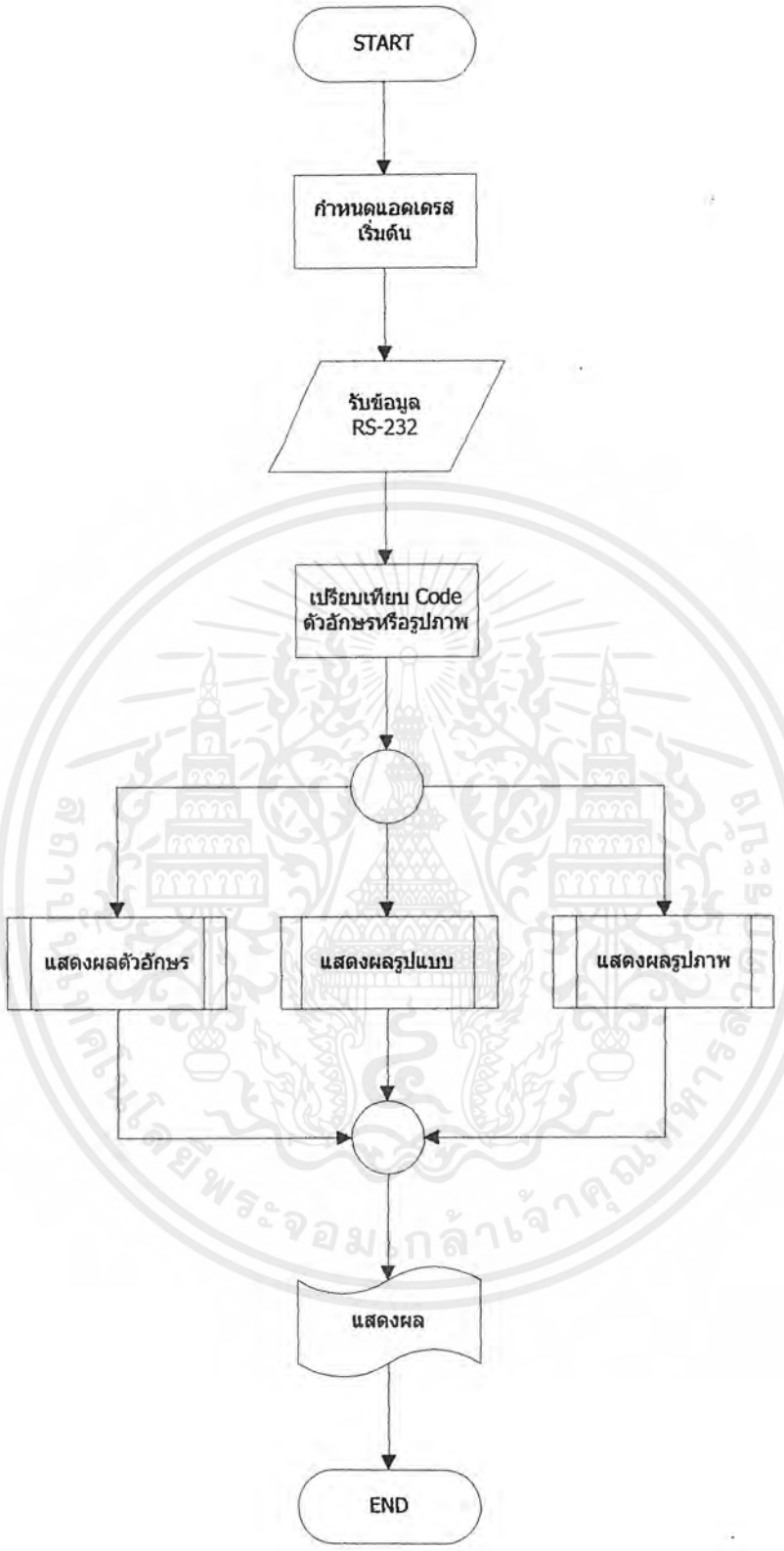
7. เมื่อต้องการเปลี่ยนข้อมูลที่จะแสดงผลให้กด Disconnection ก่อน แล้วค่อยทำตามข้อที่ 2 ไปเรื่อยๆ

8. เมื่อต้องการเลิกการเปลี่ยนแปลงข้อมูลแสดงผลให้กด Close

8.7 Flow Chart



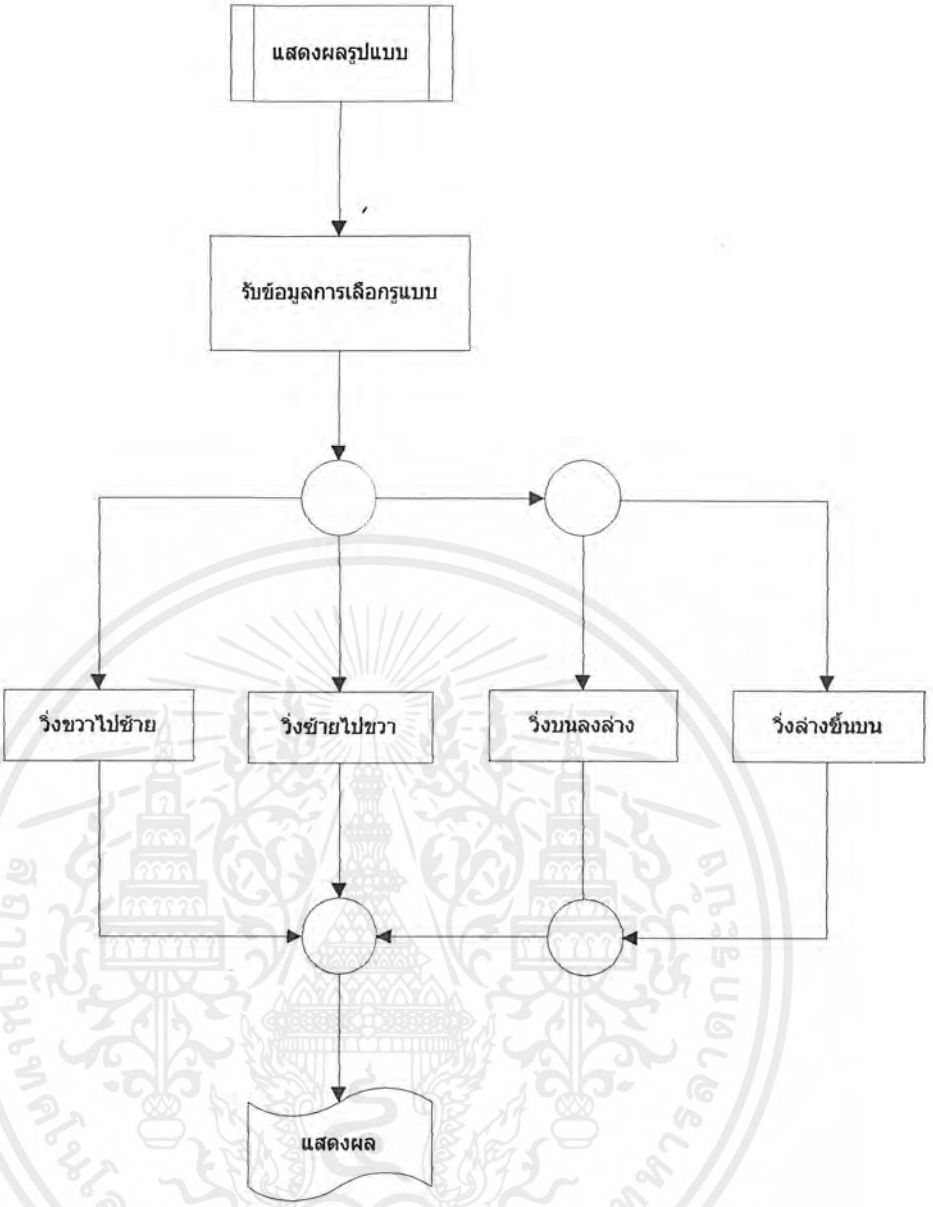
รูปที่ 8.11 flow chart แสดงการทำงานของโปรแกรม VISUAL BASIC



รูปที่ 8.12 flow chart แสดงการทำงานของโปรแกรมหลักของ MCS-51

รูปที่ 8.13 flow chart แสดงการทำงานของโปรแกรมย่อยการแสดงตัวอักษร

รูปที่ 8.14 flow chart แสดงโปรแกรมย่อยการแสดงผลรูปภาพ



รูปที่ 8.15 flow chart แสดงโปรแกรมย่อยการแสดงรูปแบบการแสดงผล

บทที่ 9
สรุปผลการทดลอง ปัญหาและข้อเสนอแนะ

โครงงานนี้ ผู้จัดทำได้ออกแบบวงจรและทดลองการทำงานตามรายละเอียดที่กล่าวมาข้างต้นและสามารถทำงานได้ตามวัตถุประสงค์ของโครงงาน จากการทดลองพบว่ายังมีปัญหาเกิดขึ้น อาทิเช่น

- ปัญหาความสว่างของแผงแสดงผลไม่สว่างเท่าตอนที่ทำการทดลอง สาเหตุเนื่องจากเมื่อประกอบส่วนประกอบต่างๆ เข้าด้วยกัน มีการ Load ของอุปกรณ์แต่ละตัวมากเกินไป จึงต้องทำการเปลี่ยนแปลง Power Supply ให้มีแรงดันและกระแสเพิ่มขึ้นอีกระดับหนึ่ง
- ปัญหาการแสดงผลของข้อความไม่ค่อยสม่ำเสมอเท่าที่ควร สาเหตุอาจเกิดตัว Matrix LED แต่ละตัว มี Characteristic ที่ผิดเพี้ยนกันไปบ้าง ทำให้ในการเขียน โปรแกรม Delay ควบคุม จึงเกิดข้อผิดพลาด
- ปัญหาการจำกัดทางด้านงบประมาณ ทำให้ได้แผงแสดงผลที่มีขนาดไม่ใหญ่มาก ทำให้ความสมบูรณ์ของการแสดงผลไม่สมบูรณ์เท่าที่ควร

โครงงานนี้แม้จะทำงานได้ผล แต่ก็ยังไม่สมบูรณ์มากนักเพราะทางผู้จัดทำนั้น ได้เน้นในเรื่องการประหยัด ทำให้ได้แผงแสดงผลที่เล็กตัวหนังสือมีความไม่สมบูรณ์เท่าที่ควร สำหรับผู้ที่สนใจถ้าต้องการให้ตัวหนังสือมีความสมบูรณ์มากกว่านี้ ก็ควรเพิ่มขนาดของแผงแสดงผลให้มีขนาดใหญ่กว่านี้ แล้วแต่ความต้องการของผู้สนใจ

บรรณานุกรม

วันสุระ ศรีใสดี. 2542. **ประยุกต์/อินเตอร์เฟสไมโครคอนโทรลเลอร์ภาคปฏิบัติ.** กรุงเทพฯ. สำนักพิมพ์ดวงกมล จำกัด.

รองศาสตราจารย์ สมยศ จุณณะปิยะ. 2543. **การประยุกต์ใช้งานไมโครคอนโทรลเลอร์ MCS-51.** พิมพ์ครั้งที่ 3. กรุงเทพฯ. สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง.

เชิดชัย บัวพันธ์, สุกิจ อุตะปะละ และอนันต์ สิรันทวิเนติ. 2534. **แผงไฟแสดงผล.** อุตสาหกรรมศาสตร์บัณฑิต สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

Kenneth J. Ayala. **The 8051 Microcontroller Architecture, Programming and Applications.** Vol.I

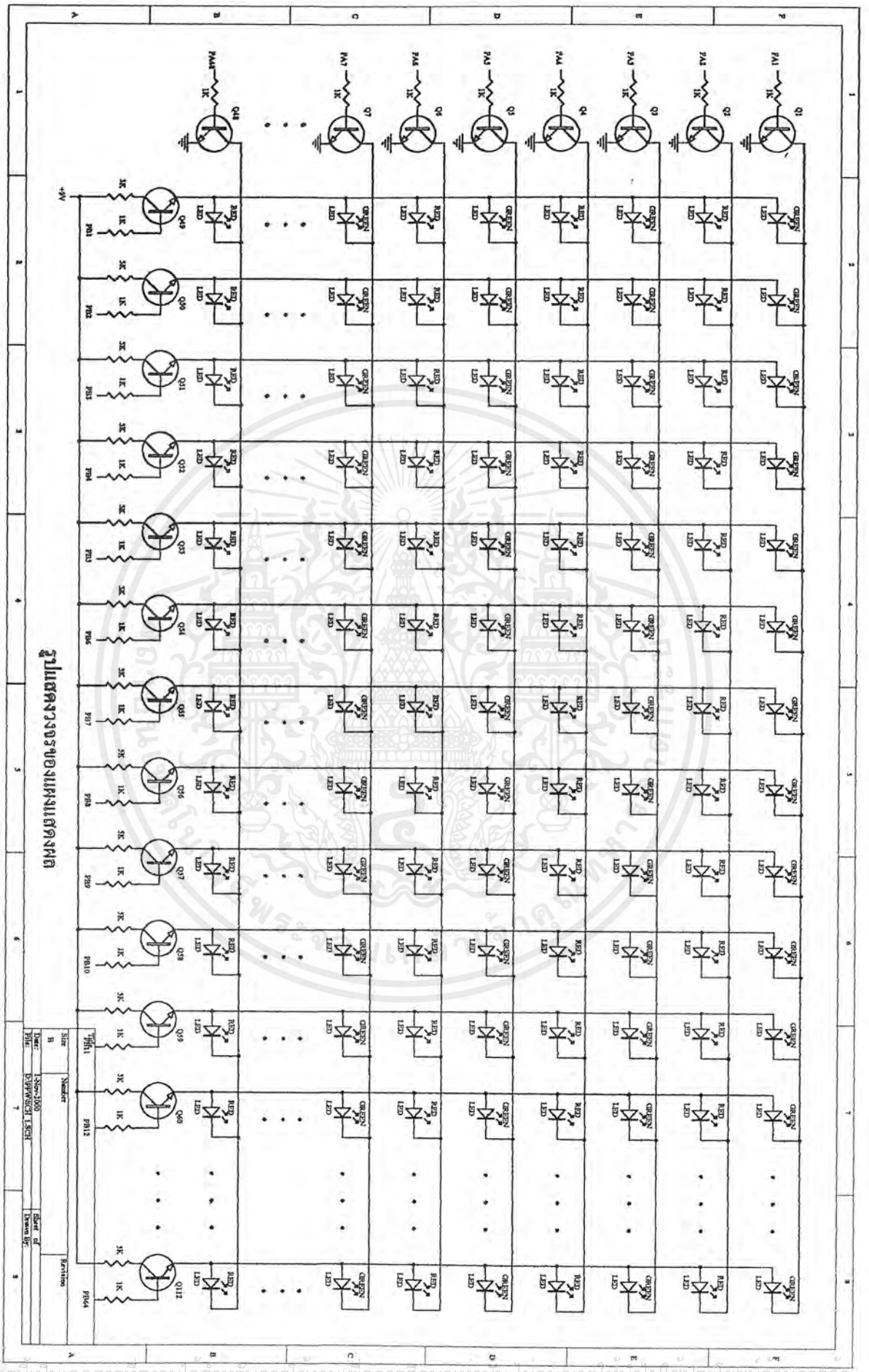
PHILIPS. **Signatics Microcontroller User's Guide.** 1989

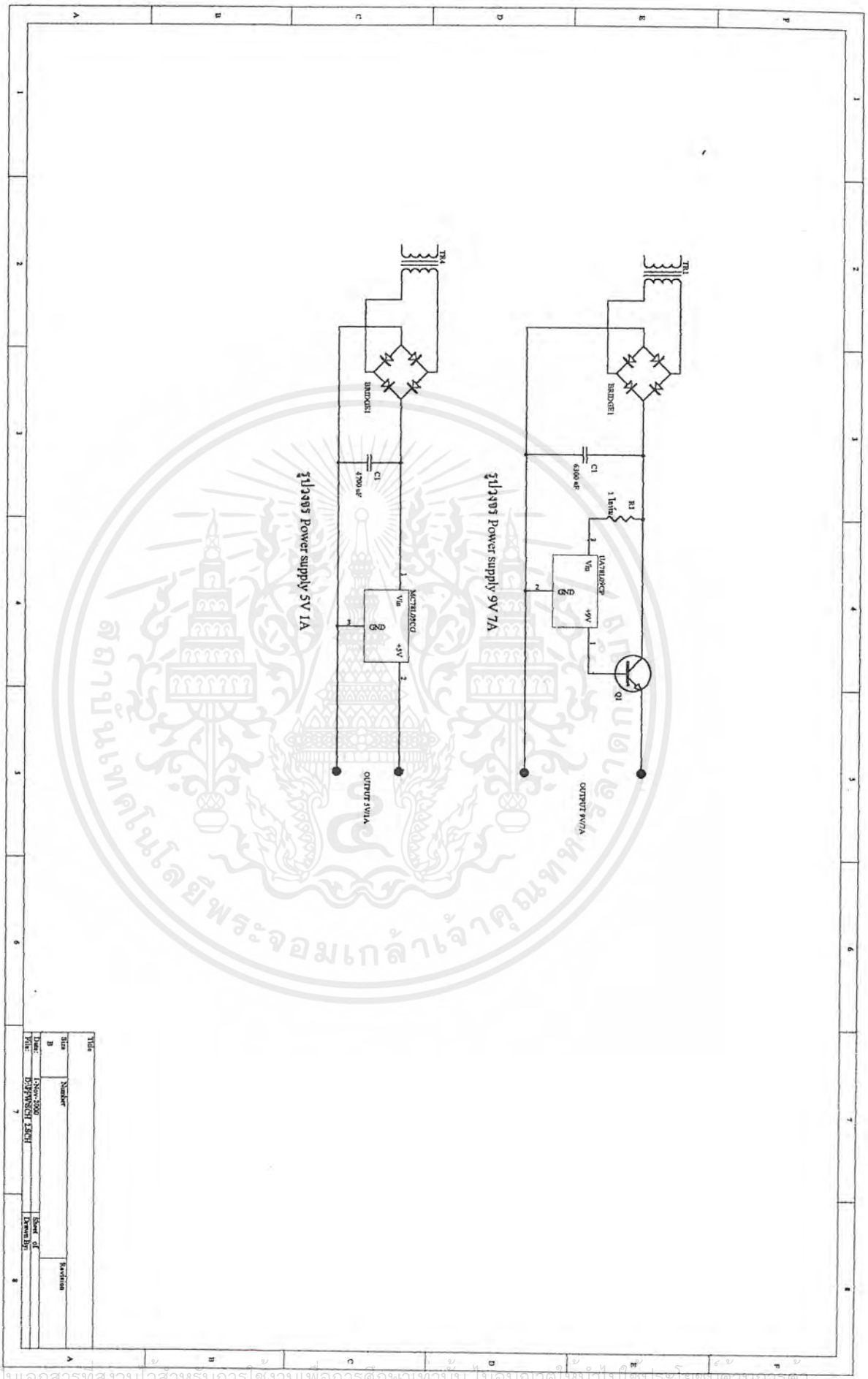
MCS-51 Microcontroller. Vol. I, Inter Coporation. 1988

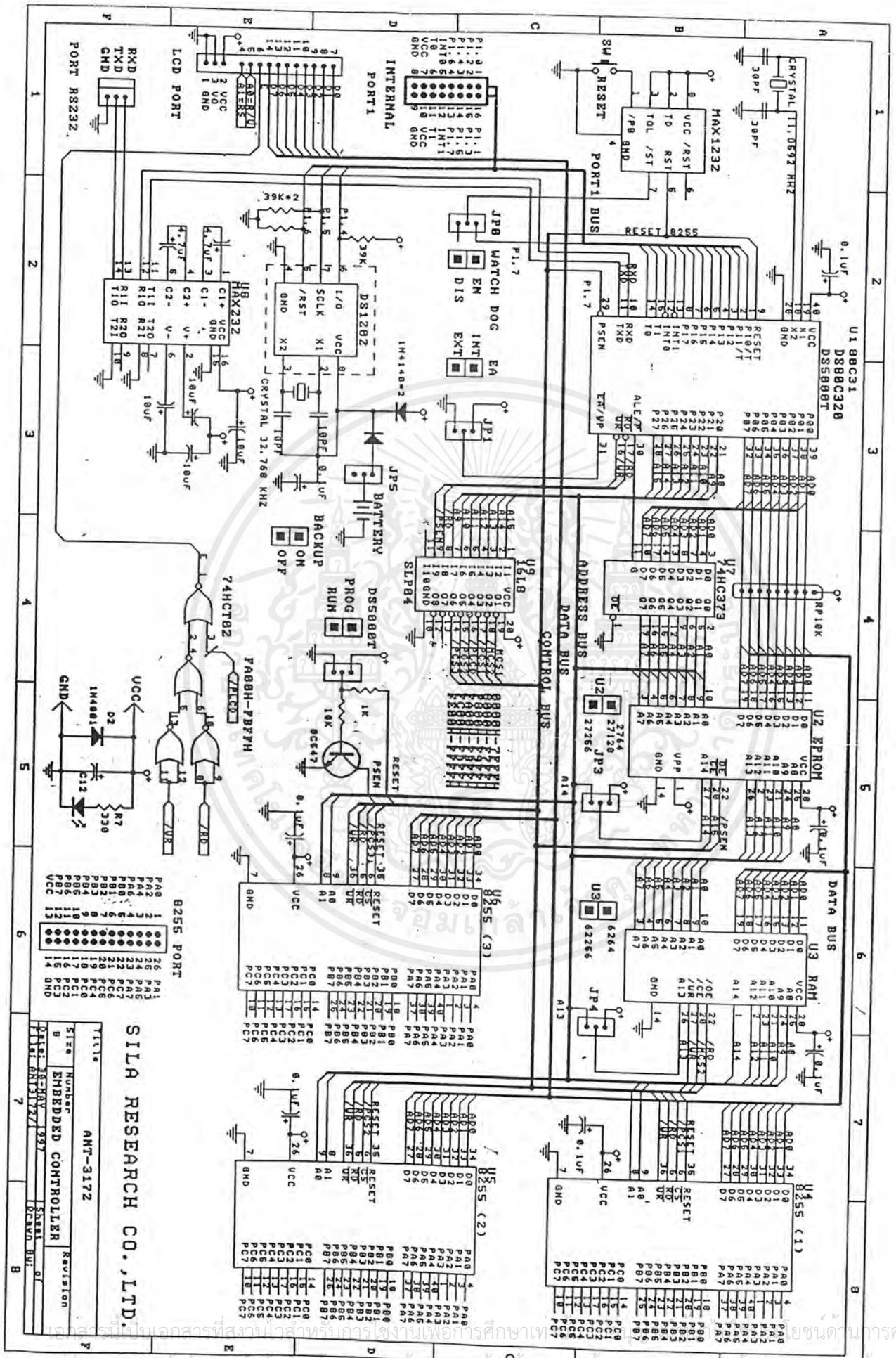
ภาคผนวก

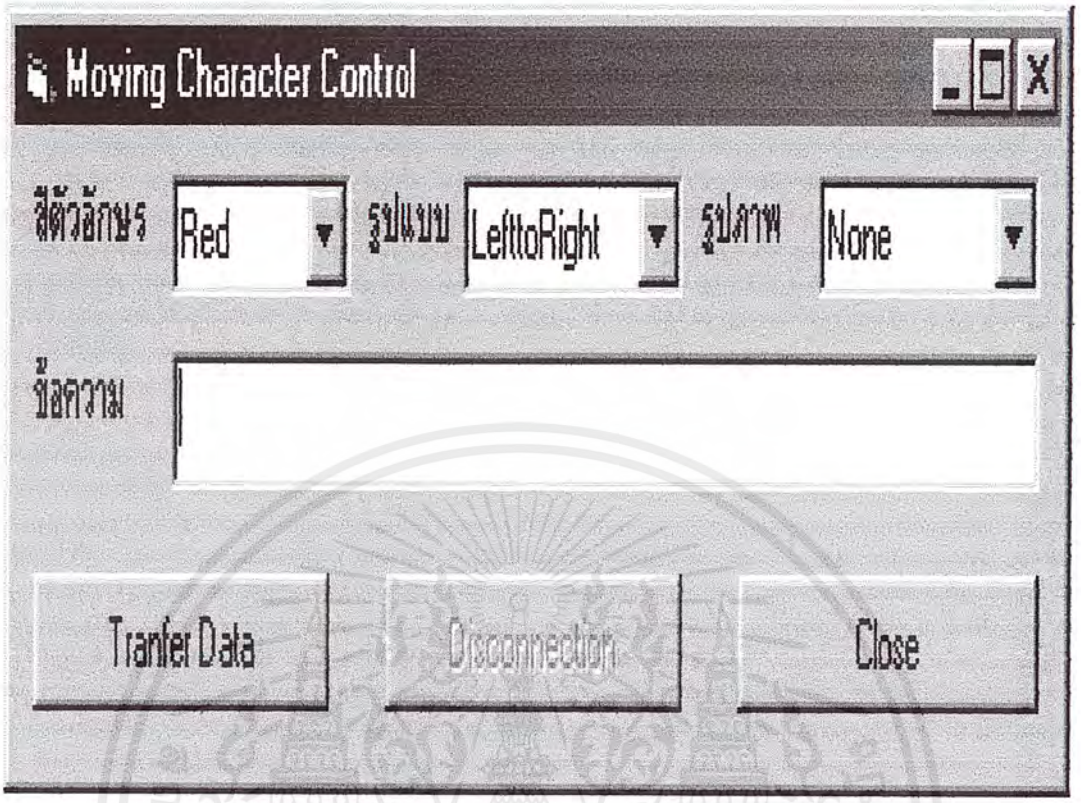
ภาคผนวก ก.

รายละเอียดของวงจร









รูปแสดงหน้าต่างสำหรับการเปลี่ยนแปลงข้อมูลสำหรับใช้งาน

ภาคผนวก ข.

```
;-------- ANT-3172 8255 PORT ---------
PORT1A      EQU   0F800H
PORT1B      EQU   0F801H
PORT1C      EQU   0F802H
PORT1CTL    EQU   0F803H

PORT2A      EQU   0FC00H
PORT2B      EQU   0FC01H
PORT2C      EQU   0FC02H
PORT2CTL    EQU   0FC03H

PORT3A      EQU   0FE00H
PORT3B      EQU   0FE01H
PORT3C      EQU   0FE02H
PORT3CTL    EQU   0FE03H

PORT4A      EQU   0FA00H
PORT4B      EQU   0FA01H
PORT4C      EQU   0FA02H
PORT4CTL    EQU   0FA03H

PORT5A      EQU   0FB00H
PORT5B      EQU   0FB01H
PORT5C      EQU   0FB02H
PORT5CTL    EQU   0FB03H

;-----------------------------------------
DISBUF1     EQU   30H;-37H
DISBUF2     EQU   38H;-3FH
DISBUF3     EQU   40H;-47H
```

```
ADDR_BUF1_H      EQU   48H
ADDR_BUF1_L      EQU   49H
ADDR_BUF2_H      EQU   4AH
ADDR_BUF2_L      EQU   4BH
ADDR_BUF3_H      EQU   4CH
ADDR_BUF3_L      EQU   4DH

ADDR_BUF1_HX     EQU   4EH
ADDR_BUF1_LX     EQU   4FH
ADDR_BUF2_HX     EQU   50H
ADDR_BUF2_LX     EQU   51H
ADDR_BUF3_HX     EQU   52H
ADDR_BUF3_LX     EQU   53H

COUNT       EQU   54H
COUNTERL    EQU   55H
COUNTERH    EQU   56H
INPUT_CHAR  EQU   57H

COLOR       EQU   58H
MODEL       EQU   59H
NUM_TX      EQU   5AH
BUF1        EQU   5BH
BUF2        EQU   5CH
PIC_NUM     EQU   5DH


ADDR_TX     EQU   9F80H
;-------- FLAG
COLOR_FLAG       EQU   05FH
FIRST_LOAD       EQU   05EH
```

```
NEW_FORM        EQU   05DH
;---------------------------------------

                ORG   0000H

;---------------------------------------
                LJMP  HOME

                ORG   0023H

                CLR   EA
                PUSH  ACC
                PUSH  DPH
                PUSH  DPL
                PUSH  02H
                PUSH  PSW
                PUSH  B

                MOV   DPTR,#ADDR_TX
                JNB   RI,$
                CLR   RI      ;SPACE
                JNB   RI,$
                MOV   A,SBUF
                MOV   COLOR,A ;COLOR
                CJNE  A,#31H,RSS1
                CLR   COLOR_FLAG
                JMP   RSS2
RSS1:           SETB  COLOR_FLAG
RSS2:           CLR   RI
                JNB   RI,$
                CLR   RI      ;-
                JNB   RI,$
```

```
        CLR   RI      ;SPACE
        JNB   RI,$
        MOV   A,SBUF
        MOV   MODEL,A ;MODEL
        CLR   RI
        JNB   RI,$
        CLR   RI      ;-
        JNB   RI,$
        MOV   A,SBUF  ;PICTURE NUMBER
        MOV   PIC_NUM,A
        SETB  FIRST_LOAD
        SETB  NEW_FORM
        CLR   RI
        JNB   RI,$
        CLR   RI      ;-
        JNB   RI,$
        MOV   A,SBUF
        MOV   R2,A
        MOV   NUM_TX,A
SR0:    CLR   RI
        JNB   RI,$
        MOV   A,SBUF
        MOVX  @DPTR,A
        INC   DPTR
        DJNZ  R2,SR0
        CALL  CLEAR_BUF
        CALL  MAKE_CODEX

        MOV   COUNTERH,#0
        MOV   COUNTERL,#60
        MOV   B,#8
        MOV   A,COUNT
```

```
        MUL   AB
        ADD   A,COUNTERL
        MOV   COUNTERL,A
        MOV   A,B
        ADDC  A,COUNTERH
        MOV   COUNTERH,A
        POP   B
        POP   PSW
        POP   02H
        POP   DPL
        POP   DPH
        POP   ACC
        CLR   RI
        SETB  EA
        RETI
;-------------------------------------

HOME:   CALL  DELAY2
        MOV   A,#80H
        MOV   DPTR,#PORT1CTL
        MOVX  @DPTR,A
        MOV   DPTR,#PORT2CTL
        MOVX  @DPTR,A
        MOV   DPTR,#PORT3CTL
        MOVX  @DPTR,A
        MOV   DPTR,#PORT4CTL
        MOVX  @DPTR,A
        MOV   DPTR,#PORT5CTL
        MOVX  @DPTR,A
        CALL  DELAY2

        MOV   TMOD,#20H
```

```
        MOV   SCON,#52H
        MOV   TH1,#0FDH
        SETB  TR1
        CLR   TI
        CLR   RI
        MOV   IE,#90H

        MOV   PIC_NUM,#1
        CLR   FIRST_LOAD
        CLR   COLOR_FLAG
        CLR   NEW_FORM

XYZ:

        CALL  CLEAR_BUF
        CALL  TRANS_CODE
        CALL  MAKE_CODE
        CALL  TRANSFER
        MOV   MODEL,#32H
X0:
RTOL:
        MOV   A,MODEL
        CJNE  A,#32H,LTOR
        MOV   ADDR_BUF1_HX,#80H
        MOV   ADDR_BUF1_LX,#00H
        MOV   ADDR_BUF2_HX,#82H
        MOV   ADDR_BUF2_LX,#00H
        MOV   ADDR_BUF3_HX,#84H
        MOV   ADDR_BUF3_LX,#00H
        MOV   COUNTERH,#0
        MOV   COUNTERL,#60
        MOV   B,#8
```

```
                MOV   A,COUNT
                MUL   AB
                ADD   A,COUNTERL
                MOV   COUNTERL,A
                MOV   A,B
                ADDC  A,COUNTERH
                MOV   COUNTERH,A
        X1:
                CALL  TRANSFER
                CALL  DISPLAY
                CALL  DELAY3
                CALL  DISPLAY
                CALL  DELAY3

                CALL  INC_ADDR

                MOV   A,COUNTERL
                CLR   C
                SUBB  A,#1
                MOV   COUNTERL,A
                MOV   A,COUNTERH
                SUBB  A,#0
                MOV   COUNTERH,A
                ORL   A,COUNTERL
                JB    NEW_FORM,RTOLZZ
                JNZ   X1
                CALL  DELAY2
                JMP   RTOL
        RTOLZZ: CLR   NEW_FORM
                JMP   RTOL
        ;-----------------------------------------------
        LTOR:   MOV   A,MODEL
```

```
        CJNE  A,#31H,TTOD
        MOV   ADDR_BUF1_HX,#80H
        MOV   ADDR_BUF1_LX,#40H
        MOV   ADDR_BUF2_HX,#82H
        MOV   ADDR_BUF2_LX,#40H
        MOV   ADDR_BUF3_HX,#84H
        MOV   ADDR_BUF3_LX,#40H
        MOV   COUNTERH,#0
        MOV   COUNTERL,#64
B1:
        CALL  TRANSFER
        CALL  DISPLAY
        CALL  DELAY3
        CALL  DISPLAY
        CALL  DELAY3

        CALL  DEC_ADDR

        MOV   A,COUNTERL
        CLR   C
        SUBB  A,#1
        MOV   COUNTERL,A
        MOV   A,COUNTERH
        SUBB  A,#0
        MOV   COUNTERH,A
        ORL   A,COUNTERL
        JB    NEW_FORM,LTORZZ
        JNZ   B1
        CALL  DELAY2
        JMP   LTOR
LTORZZ: CLR   NEW_FORM
        JMP   LTOR
```

```
;-----------------------------------------
TTOD:       MOV   A,MODEL
            CJNE  A,#33H,DTOT
            CALL  TRANSFER
            MOV   ADDR_BUF1_HX,#86H
            MOV   ADDR_BUF1_LX,#40H
            MOV   ADDR_BUF2_HX,#88H
            MOV   ADDR_BUF2_LX,#40H
            MOV   ADDR_BUF3_HX,#8AH
            MOV   ADDR_BUF3_LX,#40H
            MOV   COUNTERH,#0
            MOV   COUNTERL,#45
    TT1:
            CALL  DISPLAY
            CALL  DELAY3
            CALL  DISPLAY
            CALL  DELAY3
            CALL  DISPLAY
            CALL  DELAY3
            CALL  DISPLAY
            CALL  DELAY3
            CALL  DISPLAY
             CALL  DELAY3

            CALL  SHIFT_DATA

            MOV   A,COUNTERL
            CLR   C
            SUBB  A,#1
            MOV   COUNTERL,A
            MOV   A,COUNTERH
```

```
                SUBB   A,#0
                MOV    COUNTERH,A
                ORL    A,COUNTERL
                JB     NEW_FORM,TTODZZ
                JNZ    TT1
                CALL   DELAY2
                CALL   TRANSFER
                JMP    TTOD
RTOLX:          JMP    RTOL
TTODZZ:         CLR    NEW_FORM
                JMP    TTOD
;-----------------------------------------
DTOT:           MOV    A,MODEL
                CJNE   A,#34H,RTOLX
                CALL   TRANSFER
                MOV    ADDR_BUF1_HX,#86H
                MOV    ADDR_BUF1_LX,#40H
                MOV    ADDR_BUF2_HX,#88H
                MOV    ADDR_BUF2_LX,#40H
                MOV    ADDR_BUF3_HX,#8AH
                MOV    ADDR_BUF3_LX,#40H
                MOV    COUNTERH,#0
                MOV    COUNTERL,#45
    DD1:
                CALL   DISPLAY
                CALL   DELAY3
                CALL   DISPLAY
                CALL   DELAY3
                CALL   DISPLAY
                CALL   DELAY3
                CALL   DISPLAY
                CALL   DELAY3
```

```
                CALL   DISPLAY
                CALL   DELAY3

                CALL   SHIFT_DATAX

                MOV    A,COUNTERL
                CLR    C
                SUBB   A,#1
                MOV    COUNTERL,A
                MOV    A,COUNTERH
                SUBB   A,#0
                MOV    COUNTERH,A
                ORL    A,COUNTERL
                JB     NEW_FORM,DTOTZZ
                JNZ    DD1
                CALL   DELAY2
                CALL   TRANSFER
                JMP    DTOT
DTOTZZ:         CLR    NEW_FORM
                JMP    DTOT
;-------------------------------------
SHIFT_DATAX:
                MOV    R2,#64
                MOV    DPTR,#8440H
                MOV    BUF1,#8AH
                MOV    BUF2,#40H
SHIFT_DATAX1:
                CLR    C
                PUSH   DPH
                PUSH   DPL
                MOVX   A,@DPTR
                RLC    A
```

```
MOVX  @DPTR,A
DEC   DPH
DEC   DPH
MOVX  A,@DPTR
RLC   A
MOVX  @DPTR,A
DEC   DPH
DEC   DPH
MOVX  A,@DPTR
RLC   A
MOVX  @DPTR,A
MOV   DPH,BUF1
MOV   DPL,BUF2
MOVX  A,@DPTR
RLC   A
MOVX  @DPTR,A
DEC   DPH
DEC   DPH
MOVX  A,@DPTR
RLC   A
MOVX  @DPTR,A
DEC   DPH
DEC   DPH
MOVX  A,@DPTR
RLC   A
MOVX  @DPTR,A
MOV   DPH,BUF1
MOV   DPL,BUF2
INC   DPTR
MOV   BUF1,DPH
MOV   BUF2,DPL
POP   DPL
```

```
                POP   DPH
                INC   DPTR
                DJNZ  R2,SHIFT_DATAX1
                RET

;-------------------------------------
SHIFT_DATA:
                MOV   R2,#64
                MOV   DPTR,#8040H
SHIFT_DATA1:
                CLR   C
                PUSH  DPH
                PUSH  DPL
                MOVX  A,@DPTR
                RRC   A
                MOVX  @DPTR,A
                INC   DPH
                INC   DPH
                MOVX  A,@DPTR
                RRC   A
                MOVX  @DPTR,A
                INC   DPH
                INC   DPH
                MOVX  A,@DPTR
                RRC   A
                MOVX  @DPTR,A
                INC   DPH
                INC   DPH
                MOVX  A,@DPTR
                RRC   A
                MOVX  @DPTR,A
                INC   DPH
```

```
                INC   DPH
                MOVX  A,@DPTR
                RRC   A
                MOVX  @DPTR,A
                INC   DPH
                INC   DPH
                MOVX  A,@DPTR
                RRC   A
                MOVX  @DPTR,A
                POP   DPL
                POP   DPH
                INC   DPTR
                DJNZ  R2,SHIFT_DATA1
                RET
;-------------------------------
TRANSFER:                       ;-
                MOV   ADDR_BUF1_H,#80H;7FH     ;8000-812F |
                MOV   ADDR_BUF1_L,#38H;0F8H    ;8130-825F  > 130H
                MOV   ADDR_BUF2_H,#82H     ;8260-838F |
                MOV   ADDR_BUF2_L,#40H;30H          ;-
                MOV   ADDR_BUF3_H,#84H
                MOV   ADDR_BUF3_L,#38H
                MOV   DPTR,#8C00H      ;ADDRESS START
   X10:         MOVX  A,@DPTR
                CJNE  A,#02H,X2
                        ;MIDDLE PART
                PUSH  DPH
                PUSH  DPL
                MOV   DPH,ADDR_BUF1_H
                MOV   DPL,ADDR_BUF1_L
                CALL  ADD8DPTR
                MOV   ADDR_BUF1_H,DPH
```

```
        MOV   ADDR_BUF1_L,DPL
        MOV   DPH,ADDR_BUF3_H
        MOV   DPL,ADDR_BUF3_L
        CALL  ADD8DPTR
        MOV   ADDR_BUF3_H,DPH
        MOV   ADDR_BUF3_L,DPL
        POP   DPL
        POP   DPH
        MOV   R2,#8
X11:    INC   DPTR
        MOVX  A,@DPTR
        PUSH  DPH
        PUSH  DPL
        MOV   DPH,ADDR_BUF2_H
        MOV   DPL,ADDR_BUF2_L
        MOVX  @DPTR,A
        INC   DPTR
        MOV   ADDR_BUF2_H,DPH
        MOV   ADDR_BUF2_L,DPL
        POP   DPL
        POP   DPH
        DJNZ  R2,X11
        INC   DPTR
        JMP   X10

X2:     CJNE  A,#01H,X3
                ;UP PART
        MOV   R2,#8
X21:    INC   DPTR
        MOVX  A,@DPTR
        MOV   R4,A
        PUSH  DPH
```

```
        PUSH   DPL
        MOV    DPH,ADDR_BUF1_H
        MOV    DPL,ADDR_BUF1_L
        MOVX   A,@DPTR
        ORL    A,R4
        MOVX   @DPTR,A
        INC    DPTR
        MOV    ADDR_BUF1_H,DPH
        MOV    ADDR_BUF1_L,DPL
        POP    DPL
        POP    DPH
        DJNZ   R2,X21
        INC    DPTR
        PUSH   DPH
        PUSH   DPL
        MOV    DPH,ADDR_BUF1_H
        MOV    DPL,ADDR_BUF1_L
        CALL   SUB8DPTR
        MOV    ADDR_BUF1_H,DPH
        MOV    ADDR_BUF1_L,DPL
        POP    DPL
        POP    DPH
        JMP    X10

X22:    RET

X3:     CJNE   A,#00H,X22
                ;DOWN PART
        MOV    R2,#8
X31:    INC    DPTR
        MOVX   A,@DPTR
        MOV    R4,A
```

```
                PUSH   DPH
                PUSH   DPL
                MOV    DPH,ADDR_BUF3_H
                MOV    DPL,ADDR_BUF3_L
                MOVX   A,@DPTR
                ORL    A,R4
                MOVX   @DPTR,A
                INC    DPTR
                MOV    ADDR_BUF3_H,DPH
                MOV    ADDR_BUF3_L,DPL
                POP    DPL
                POP    DPH
                DJNZ   R2,X31
                INC    DPTR
                PUSH   DPH
                PUSH   DPL
                MOV    DPH,ADDR_BUF3_H
                MOV    DPL,ADDR_BUF3_L
                CALL   SUB8DPTR
                MOV    ADDR_BUF3_H,DPH
                MOV    ADDR_BUF3_L,DPL
                POP    DPL
                POP    DPH
                JMP    X10
;---------------------------------------
SUB8DPTR:       CLR    C
                MOV    A,DPL
                SUBB   A,#8
                MOV    DPL,A
                MOV    A,DPH
                SUBB   A,#0
                MOV    DPH,A
```

```
                RET
;---------------------------------------
INC_ADDR:
                PUSH   DPH
                PUSH   DPL
                MOV    DPH,ADDR_BUF1_HX
                MOV    DPL,ADDR_BUF1_LX
                INC    DPTR
                MOV    ADDR_BUF1_HX,DPH
                MOV    ADDR_BUF1_LX,DPL
                MOV    DPH,ADDR_BUF2_HX
                MOV    DPL,ADDR_BUF2_LX
                INC    DPTR
                MOV    ADDR_BUF2_HX,DPH
                MOV    ADDR_BUF2_LX,DPL
                MOV    DPH,ADDR_BUF3_HX
                MOV    DPL,ADDR_BUF3_LX
                INC    DPTR
                MOV    ADDR_BUF3_HX,DPH
                MOV    ADDR_BUF3_LX,DPL
                POP    DPL
                POP    DPH
                RET

;---------------------------------------
DEC_ADDR:
                PUSH   DPH
                PUSH   DPL
                MOV    DPH,ADDR_BUF1_HX
                MOV    DPL,ADDR_BUF1_LX
                CALL   SUBDPTR
                MOV    ADDR_BUF1_HX,DPH
```

```
        MOV   ADDR_BUF1_LX,DPL
        MOV   DPH,ADDR_BUF2_HX
        MOV   DPL,ADDR_BUF2_LX
        CALL  SUBDPTR
        MOV   ADDR_BUF2_HX,DPH
        MOV   ADDR_BUF2_LX,DPL
        MOV   DPH,ADDR_BUF3_HX
        MOV   DPL,ADDR_BUF3_LX
        CALL  SUBDPTR
        MOV   ADDR_BUF3_HX,DPH
        MOV   ADDR_BUF3_LX,DPL
        POP   DPL
        POP   DPH
        RET

SUBDPTR: MOV  A,DPL
        CLR   C
        SUBB  A,#1
        MOV   DPL,A
        MOV   A,DPH
        SUBB  A,#0
        MOV   DPH,A
        RET

;---------------------------------------
;------------ DISPLAY ------------------
;---------------------------------------
DISPLAY:
        ;LOAD TO CATCH MEM.

        MOV   ADDR_BUF1_H,ADDR_BUF1_HX
        MOV   ADDR_BUF1_L,ADDR_BUF1_LX
```

```
        MOV   ADDR_BUF2_H,ADDR_BUF2_HX
        MOV   ADDR_BUF2_L,ADDR_BUF2_LX
        MOV   ADDR_BUF3_H,ADDR_BUF3_HX
        MOV   ADDR_BUF3_L,ADDR_BUF3_LX

        MOV   R3,#1
        MOV   R4,#8
DISPLAY1: MOV   DPH,ADDR_BUF1_H
        MOV   DPL,ADDR_BUF1_L
        MOV   R2,#8
        MOV   R0,#DISBUF1
LOOP1:  MOVX  A,@DPTR
        MOV   @R0,A
        INC   R0
        CALL  ADD8DPTR
        DJNZ  R2,LOOP1
        MOV   DPH,ADDR_BUF1_H
        MOV   DPL,ADDR_BUF1_L
        INC   DPTR
        MOV   ADDR_BUF1_H,DPH
        MOV   ADDR_BUF1_L,DPL
        MOV   A,R3
        JB    COLOR_FLAG,ZZ1
        MOV   DPTR,#PORT4C
        JMP   ZZ2
ZZ1:    MOV   DPTR,#PORT4A
ZZ2:    MOVX  @DPTR,A
        MOV   R0,#DISBUF1
        CALL  OUT_DISP
        CALL  DELAY
        MOV   A,#0
        JB    COLOR_FLAG,ZZ11
```

```
        MOV   DPTR,#PORT4C
        JMP   ZZ21
ZZ11:   MOV   DPTR,#PORT4A
ZZ21:   MOVX  @DPTR,A

        MOV   DPH,ADDR_BUF2_H
        MOV   DPL,ADDR_BUF2_L
        MOV   R2,#8
        MOV   R0,#DISBUF2
LOOP2:  MOVX  A,@DPTR
        MOV   @R0,A
        INC   R0
        CALL  ADD8DPTR
        DJNZ  R2,LOOP2
        MOV   DPH,ADDR_BUF2_H
        MOV   DPL,ADDR_BUF2_L
        INC   DPTR
        MOV   ADDR_BUF2_H,DPH
        MOV   ADDR_BUF2_L,DPL
        MOV   A,R3
        JB    COLOR_FLAG,ZZ3
        MOV   DPTR,#PORT5A
        JMP   ZZ4
ZZ3:    MOV   DPTR,#PORT4B
ZZ4:    MOVX  @DPTR,A
        MOV   R0,#DISBUF2
        CALL  OUT_DISP
        CALL  DELAY
        MOV   A,#0
        MOV   DPTR,#PORT5A
        MOVX  @DPTR,A
        MOV   DPTR,#PORT4B
```

```
                MOVX   @DPTR,A


                MOV    DPH,ADDR_BUF3_H
                MOV    DPL,ADDR_BUF3_L
                MOV    R2,#8
                MOV    R0,#DISBUF3
LOOP3:          MOVX   A,@DPTR
                MOV    @R0,A
                INC    R0
                CALL   ADD8DPTR
                DJNZ   R2,LOOP3
                MOV    DPH,ADDR_BUF3_H
                MOV    DPL,ADDR_BUF3_L
                INC    DPTR
                MOV    ADDR_BUF3_H,DPH
                MOV    ADDR_BUF3_L,DPL
                MOV    A,R3
                JB     COLOR_FLAG,ZZ5
                MOV    DPTR,#PORT5C
                JMP    ZZ6
ZZ5:            MOV    DPTR,#PORT5B
ZZ6:            MOVX   @DPTR,A
                MOV    R0,#DISBUF3
                CALL   OUT_DISP
                CALL   DELAY
                MOV    A,#0
                MOV    DPTR,#PORT5C
                MOVX   @DPTR,A
                MOV    DPTR,#PORT5B
                MOVX   @DPTR,A
                MOV    A,R3
                RL     A
```

```
                MOV   R3,A
                DJNZ  R4,DISPLAYX
                RET
DISPLAYX:       JMP   DISPLAY1
;-----------------------------------------------
OUT_DISP:
                MOV   A,@R0
                INC   R0
                MOV   DPTR,#PORT1A
                MOVX  @DPTR,A
                MOV   A,@R0
                INC   R0
                MOV   DPTR,#PORT1B
                MOVX  @DPTR,A
                MOV   A,@R0
                INC   R0
                MOV   DPTR,#PORT1C
                MOVX  @DPTR,A
                MOV   A,@R0
                INC   R0
                MOV   DPTR,#PORT2A
                MOVX  @DPTR,A
                MOV   A,@R0
                INC   R0
                MOV   DPTR,#PORT2B
                MOVX  @DPTR,A
                MOV   A,@R0
                INC   R0
                MOV   DPTR,#PORT2C
                MOVX  @DPTR,A
                MOV   A,@R0
                INC   R0
```

```
                MOV   DPTR,#PORT3A
                MOVX  @DPTR,A
                MOV   A,@R0
                INC   R0
                MOV   DPTR,#PORT3B
                MOVX  @DPTR,A
                RET
;---------------------------------------
ADD8DPTR:
                MOV   A,#8
                ADD   A,DPL
                MOV   DPL,A
                MOV   A,#0
                ADDC  A,DPH
                MOV   DPH,A
                RET
;---------------------------------------------
;---------------------------------------------
;---------------------------------------
CLEAR_DISBUF:
                MOV   R1,#DISBUF1
                MOV   R5,#24
                MOV   A,#0
CLEAR_DIS1:     MOV   @R1,A
                INC   R1
                DJNZ  R5,CLEAR_DIS1
                RET
;---------------------------------------
CLEAR_BUF:
                MOV   COUNTERH,#1FH
                MOV   COUNTERL,#00H
                MOV   DPTR,#8000H
```

```
CLEAR1:  MOV   A,#0H
         MOVX  @DPTR,A
         INC   DPTR
         MOV   A,COUNTERL
         CLR   C
         SUBB  A,#1
         MOV   COUNTERL,A
         MOV   A,COUNTERH
         SUBB  A,#0
         MOV   COUNTERH,A
         ORL   A,COUNTERL
         JNZ   CLEAR1
         RET

;-------------------------------------
;-------------------------------------
DELAY:   MOV   R7,#01H
DELAY1:  MOV   R6,#02H
         MOV   R5,#0A9H
         DJNZ  R5,$
         DJNZ  R6,$-4
         DJNZ  R7,DELAY1
         RET
;-------------------------------------
DELAY3:  MOV   R7,#01H
DELAY31: MOV   R6,#5H
         MOV   R5,#0FFH
         DJNZ  R5,$
         DJNZ  R6,$-4
         DJNZ  R7,DELAY31
         RET
```

```
;---------------------------------------
DELAY2:     MOV   R7,#05H
DELAY21:    MOV   R6,#0FFH
            MOV   R5,#0FFH
            DJNZ  R5,$
            DJNZ  R6,$-4
            DJNZ  R7,DELAY21
            RET
;---------------------------------------
LOAD_PIC1:  MOV   DPTR,#PIC_1
            ;MOV   BUF1,#8CH
            ;MOV   BUF2,#00H
            MOV   R2,#36
LLL:        CLR   A
            MOVC  A,@A+DPTR
            PUSH  DPH
            PUSH  DPL
            MOV   DPH,ADDR_BUF1_H
            MOV   DPL,ADDR_BUF1_L
            MOVX  @DPTR,A
            INC   DPTR
            MOV   ADDR_BUF1_H,DPH
            MOV   ADDR_BUF1_L,DPL
            POP   DPL
            POP   DPH
            INC   DPTR
            DJNZ  R2,LLL

            RET

;---------------------------------------
LOAD_PIC2:  MOV   DPTR,#PIC_2
```

```
                ;MOV    BUF1,#8CH
                ;MOV    BUF2,#00H
                MOV     R2,#45
LLL2:           CLR     A
                MOVC    A,@A+DPTR
                PUSH    DPH
                PUSH    DPL
                MOV     DPH,ADDR_BUF1_H
                MOV     DPL,ADDR_BUF1_L
                MOVX    @DPTR,A
                INC     DPTR
                MOV     ADDR_BUF1_H,DPH
                MOV     ADDR_BUF1_L,DPL
                POP     DPL
                POP     DPH
                INC     DPTR
                DJNZ    R2,LLL2

                RET
;-------------------------------------
LOAD_PIC3:      MOV     DPTR,#PIC_3
                ;MOV    BUF1,#8CH
                ;MOV    BUF2,#00H
                MOV     R2,#54
LLL3:           CLR     A
                MOVC    A,@A+DPTR
                PUSH    DPH
                PUSH    DPL
                MOV     DPH,ADDR_BUF1_H
                MOV     DPL,ADDR_BUF1_L
                MOVX    @DPTR,A
                INC     DPTR
```

```
                MOV    ADDR_BUF1_H,DPH
                MOV    ADDR_BUF1_L,DPL
                POP    DPL
                POP    DPH
                INC    DPTR
                DJNZ   R2,LLL3

                RET

;-----------------------------------------
LOAD_PIC4:      MOV    DPTR,#PIC_4
                ;MOV    BUF1,#8CH
                ;MOV    BUF2,#00H
                MOV    R2,#36
LLL4:           CLR    A
                MOVC   A,@A+DPTR
                PUSH   DPH
                PUSH   DPL
                MOV    DPH,ADDR_BUF1_H
                MOV    DPL,ADDR_BUF1_L
                MOVX   @DPTR,A
                INC    DPTR
                MOV    ADDR_BUF1_H,DPH
                MOV    ADDR_BUF1_L,DPL
                POP    DPL
                POP    DPH
                INC    DPTR
                DJNZ   R2,LLL4

                RET

;-----------------------------------------
```

```
LOAD_PIC5:  MOV   DPTR,#PIC_5
            ;MOV   BUF1,#8CH
            ;MOV   BUF2,#00H
            MOV   R2,#36
LLL5:       CLR   A
            MOVC  A,@A+DPTR
            PUSH  DPH
            PUSH  DPL
            MOV   DPH,ADDR_BUF1_H
            MOV   DPL,ADDR_BUF1_L
            MOVX  @DPTR,A
            INC   DPTR
            MOV   ADDR_BUF1_H,DPH
            MOV   ADDR_BUF1_L,DPL
            POP   DPL
            POP   DPH
            INC   DPTR
            DJNZ  R2,LLL5

            RET

;-----------------------------------------
;INPUT = INPUT_CHAR(ADDRESS)
;    = ADDRESS INPUT STREAM = ADDR_BUF1_H,ADDR_BUF1_L

MAKE_CODE:   MOV   NUM_TX,#5
             CALL  MAKE_CODEX
             RET

TRANS_CODE:  MOV   DPTR,#TABLE_INPUT
             MOV   ADDR_BUF1_H,#HIGH(ADDR_TX)
```

```
                MOV   ADDR_BUF1_L,#LOW(ADDR_TX)
                MOV   R2,#5
TMAKEC1:        CLR   A
                MOVC  A,@A+DPTR
                PUSH  DPH
                PUSH  DPL
                MOV   DPH,ADDR_BUF1_H
                MOV   DPL,ADDR_BUF1_L
                MOVX  @DPTR,A
                INC   DPTR
                MOV   ADDR_BUF1_H,DPH
                MOV   ADDR_BUF1_L,DPL
                POP   DPL
                POP   DPH
                INC   DPTR
                DJNZ  R2,TMAKEC1
                RET
TABLE_INPUT:  DB    4BH,4DH,49H,54H,4CH

MAKE_CODEX:   MOV   DPTR,#ADDR_TX;TABLE_INPUT
                MOV   ADDR_BUF1_H,#8CH
                MOV   ADDR_BUF1_L,#00H
                MOV   R2,NUM_TX;#26
                MOV   COUNT,R2
MAKEC1:         CLR   A
                MOVC  A,@A+DPTR
                MOV   INPUT_CHAR,A
                PUSH  02H
                PUSH  DPH
                PUSH  DPL
                JNB   FIRST_LOAD,MAKEC2
                MOV   A,PIC_NUM
```

```
                CJNE   A,#1,MM1
                CALL   LOAD_PIC1
                JMP    MAKEC2
MM1:            CJNE   A,#2,MM2
                CALL   LOAD_PIC2
                JMP    MAKEC2
MM2:            CJNE   A,#3,MM3
                CALL   LOAD_PIC3
                JMP    MAKEC2
MM3:            CJNE   A,#4,MM4
                CALL   LOAD_PIC4
                JMP    MAKEC2
MM4:            CJNE   A,#5,MM5
                CALL   LOAD_PIC5
                JMP    MAKEC2
MM5:
MAKEC2:         CLR    FIRST_LOAD
                CALL   OPEN_CODE
                POP    DPL
                POP    DPH
                POP    02H
                INC    DPTR
                DJNZ   R2,MAKEC1
                RET

;-------------------------------------------------------
OPEN_CODE:
                CALL   FIND_ADDR
                CJNE   A,#0FFH,OPEN_CODE1
                CALL   FIND_ADDRX
                CJNE   A,#0FFH,OPEN_CODE2
                RET
```

```
OPEN_CODE1:   MOV   DPTR,#TABLE_CODE1
              MOV   B,#9
              MUL   AB
              ADD   A;DPL
              MOV   DPL,A
              MOV   A,B
              ADDC  A,DPH
              MOV   DPH,A
              MOV   R2,#9
OPENX1:       CLR   A
              MOVC  A,@A+DPTR
              PUSH  DPH
              PUSH  DPL
              MOV   DPH,ADDR_BUF1_H
              MOV   DPL,ADDR_BUF1_L
              MOVX  @DPTR,A
              INC   DPTR
              MOV   ADDR_BUF1_H,DPH
              MOV   ADDR_BUF1_L,DPL
              POP   DPL
              POP   DPH
              INC   DPTR
              DJNZ  R2,OPENX1
              MOV   A,#05H
              MOV   DPH,ADDR_BUF1_H
              MOV   DPL,ADDR_BUF1_L
              MOVX  @DPTR,A
              RET

OPEN_CODE2:   MOV   DPTR,#TABLE_CODE2
              MOV   B,#18
              MUL   AB
```

```
                ADD   A,DPL
                MOV   DPL,A
                MOV   A,B
                ADDC  A,DPH
                MOV   DPH,A
                MOV   R2,#18
OPENY1:         CLR   A
                MOVC  A,@A+DPTR
                PUSH  DPH
                PUSH  DPL
                MOV   DPH,ADDR_BUF1_H
                MOV   DPL,ADDR_BUF1_L
                MOVX  @DPTR,A
                INC   DPTR
                MOV   ADDR_BUF1_H,DPH
                MOV   ADDR_BUF1_L,DPL
                POP   DPL
                POP   DPH
                INC   DPTR
                DJNZ  R2,OPENY1
                MOV   A,#05H
                MOV   DPH,ADDR_BUF1_H
                MOV   DPL,ADDR_BUF1_L
                MOVX  @DPTR,A
                RET

FIND_ADDR:      MOV   DPTR,#TABLE_CHAR1
                MOV   R2,#132
                MOV   R3,#0
FIND_ADDR0:     CLR   A
                MOVC  A,@A+DPTR
                CJNE  A,INPUT_CHAR,FIND_ADDR1
```

```
                MOV    A,R3
                RET
FIND_ADDR1:     INC    DPTR
                INC    R3
                DJNZ   R2,FIND_ADDR0
                MOV    A,#0FFH
                RET

FIND_ADDRX:     MOV    DPTR,#TABLE_CHAR2
                MOV    R2,#25
                MOV    R3,#0
FIND_ADDRX0:    CLR    A
                MOVC   A,@A+DPTR
                CJNE   A,INPUT_CHAR,FIND_ADDRX1
                MOV    A,R3
                RET
FIND_ADDRX1:    INC    DPTR
                INC    R3
                DJNZ   R2,FIND_ADDRX0
                MOV    A,#0FFH
                RET

TABLE_CHAR1:
        DB      0A1H,0A2H,0A4H,0A6H,0A7H,0A8H,0A9H
        DB      0ACH,0B1H,0B2H
        DB      0B3H,0B4H,0B5H,0B6H,0B7H,0B8H,0B9H,0BAH
        DB      0BCH,0BEH,0C0H,0C1H,0C2H
        DB      0C3H,0C5H,0C7H,0C9H
        DB      0CBH,0CDH,0CFH,0D0H,0D1H,0D2H
        DB      0D4H,0D5H,0D6H,0D7H,0D8H,0D9H,0DAH
        DB      0DFH,0E0H,0E1H,0E7H
        DB      0E8H,0E9H,0EAH,0EBH,0ECH,0F0H,0F1H,0F2H
```

```
        DB    0F3H,0F4H,0F5H,0F6H,0F7H,0F8H,0F9H,041H
        DB    042H,043H,044H,045H,046H,047H,048H,049H
        DB    04AH,04BH,04CH,04DH,04EH,04FH,050H,051H
        DB    052H,053H,054H,055H,056H,057H,058H,059H
        DB    05AH,061H,062H,063H,064H,065H,066H
        DB    068H,069H,06BH,06CH,06DH,06EH,06FH
        DB    072H,073H,074H,075H,076H,077H
        DB    078H,07AH,021H,022H,025H,028H,029H
        DB    02AH,02BH,02CH,02DH,02EH,02FH,030H,031H
        DB    032H,033H,034H,035H,036H,037H,038H,039H
        DB    03DH,03FH,03CH,03EH,020H

TABLE_CHAR2:
        DB    0AAH,0ABH,0ADH,0AEH,0AFH,0B0H,0BBH,0BDH
        DB    0BFH,0C4H,0C6H,0C8H,0CAH,0CCH,0CEH,0D3H
        DB    0E2H,0E3H,0E4H,0E6H,067H,06AH,070H,071H,079H

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

;----------------------------------------
TABLE_CODE1:
    TH_1:  DB    02H,000H,05FH,0A0H,080H,080H,080H,080H,07FH
    TH_2:  DB    02H,000H,0C0H,0FEH,001H,0FEH,000H,000H,000H
    TH_3:  DB    02H,000H,07FH,08FH,098H,098H,080H,07FH,000H
    TH_4:  DB    02H,000H,042H,085H,05EH,0A4H,042H,001H,0FFH
    TH_5:  DB    02H,000H,010H,008H,0C4H,0FFH,000H,000H,000H
    TH_6:  DB    02H,000H,040H,098H,098H,084H,07FH,000H,000H
    TH_7:  DB    02H,000H,058H,09FH,082H,084H,07BH,003H,000H
    TH_8:  DB    02H,000H,05FH,0A3H,080H,083H,07FH,001H,0FFH
    TH_9:  DB    02H,000H,040H,080H,05FH,0A0H,040H,080H,07FH
    TH_10: DB    02H,0FFH,082H,05CH,058H,083H,0FFH,001H,0FFH
```

```
TH_11: DB   02H,05FH,0A3H,080H,080H,07FH,002H,0FFH,003H
TH_12: DB   02H,000H,07EH,081H,09AH,09CH,080H,07FH,000H
TH_13: DB   02H,000H,07EH,081H,05AH,09CH,080H,07FH,000H
TH_14: DB   02H,000H,05FH,0A3H,080H,080H,080H,07FH,000H
TH_15: DB   02H,000H,0C0H,0FFH,020H,040H,080H,080H,07FH
TH_16: DB   02H,000H,060H,0AFH,0A1H,0A1H,09FH,000H,000H
TH_17: DB   02H,000H,0C0H,0FFH,002H,004H,00BH,0FBH,000H
TH_18: DB   02H,000H,0C0H,0FFH,001H,001H,001H,0FFH,000H
TH_19: DB   02H,07CH,0C2H,0C1H,00EH,00EH,001H,002H,0FCH
TH_20: DB   02H,000H,0FFH,010H,060H,060H,010H,0FFH,000H
TH_21: DB   02H,000H,003H,05FH,0A0H,080H,080H,080H,07FH
TH_22: DB   02H,000H,0C3H,0FFH,004H,002H,001H,0FEH,000H
TH_23: DB   02H,000H,06CH,0D2H,0C1H,001H,001H,0FEH,000H
TH_24: DB   02H,000H,060H,0A0H,0A3H,0BFH,080H,000H,000H
TH_25: DB   02H,000H,04FH,093H,090H,088H,08CH,07FH,000H
TH_26: DB   02H,000H,000H,040H,080H,083H,07FH,000H,000H
TH_27: DB   02H,0C0H,0FFH,001H,001H,031H,011H,0FFH,010H
TH_28: DB   02H,000H,0C0H,0FFH,010H,020H,0C0H,0C0H,03FH
TH_29: DB   02H,000H,04EH,099H,099H,081H,081H,081H,07EH
TH_30: DB   02H,060H,090H,090H,010H,020H,040H,0FFH,000H
TH_31: DB   02H,000H,036H,012H,024H,048H,000H,000H,000H
TH_32: DB   01H,000H,000H,006H,002H,004H,008H,000H,000H
TH_33: DB   02H,000H,040H,080H,080H,080H,07FH,000H,000H
TH_34: DB   01H,002H,006H,00AH,00AH,00AH,006H,002H,000H
TH_35: DB   01H,002H,006H,00AH,00AH,00AH,006H,00EH,000H
TH_36: DB   01H,002H,006H,00AH,00AH,00AH,006H,00AH,004H
TH_37: DB   01H,002H,006H,00AH,006H,00EH,002H,00EH,000H
TH_38: DB   00H,000H,000H,000H,040H,070H,000H,000H,000H
TH_39: DB   00H,000H,000H,040H,070H,010H,070H,000H,000H
TH_40: DB   01H,000H,000H,004H,00AH,004H,000H,000H,000H
TH_41: DB   02H,000H,002H,0FFH,089H,091H,06EH,040H,000H
TH_42: DB   02H,000H,000H,000H,000H,000H,0FFH,003H,000H
```

```
TH_43: DB   02H,000H,000H,0FFH,003H,000H,0FFH,003H,000H
TH_44: DB   01H,00CH,012H,014H,012H,014H,010H,020H,000H
TH_45: DB   01H,000H,000H,000H,000H,000H,000H,0E0H,000H
TH_46: DB   01H,000H,000H,080H,0E0H,040H,080H,000H,000H
TH_47: DB   01H,000H,060H,0A0H,040H,080H,060H,040H,080H
TH_48: DB   01H,000H,000H,000H,000H,000H,020H,070H,020H
TH_49: DB   01H,000H,000H,000H,000H,000H,060H,0A0H,000H
TH_50: DB   02H,000H,009H,015H,011H,00EH,000H,000H,000H
TH_51: DB   02H,01CH,002H,006H,009H,006H,009H,006H,000H
TH_52: DB   02H,000H,007H,009H,004H,008H,007H,000H,000H
TH_53: DB   02H,006H,009H,00BH,009H,011H,020H,000H,000H
TH_54: DB   02H,006H,029H,05BH,029H,051H,000H,000H,000H
TH_55: DB   02H,000H,040H,020H,012H,009H,009H,006H,000H
TH_56: DB   02H,006H,009H,004H,008H,007H,002H,03CH,000H
TH_57: DB   02H,006H,009H,00AH,009H,00AH,010H,020H,000H
TH_58: DB   02H,006H,009H,01CH,012H,021H,040H,000H,000H
TH_59: DB   02H,000H,006H,009H,009H,009H,006H,000H,000H
TH_60: DB   02H,000H,03FH,044H,084H,084H,044H,03FH,000H
TH_61: DB   02H,000H,0FFH,091H,091H,091H,091H,06FH,000H
TH_62: DB   02H,000H,07EH,081H,081H,081H,081H,042H,000H
TH_63: DB   02H,000H,0FFH,081H,081H,081H,042H,03CH,000H
TH_64: DB   02H,000H,0FFH,091H,091H,091H,091H,081H,000H
TH_65: DB   02H,000H,0FFH,090H,090H,090H,090H,080H,000H
TH_66: DB   02H,000H,07EH,081H,081H,089H,089H,04EH,000H
TH_67: DB   02H,000H,0FFH,010H,010H,010H,010H,0FFH,000H
TH_68: DB   02H,000H,000H,081H,0FFH,081H,000H,000H,000H
TH_69: DB   02H,000H,006H,001H,001H,081H,0FEH,080H,000H
TH_70: DB   02H,000H,0FFH,018H,024H,042H,081H,000H,000H
TH_71: DB   02H,000H,0FFH,001H,001H,001H,001H,001H,000H
TH_72: DB   02H,000H,0FFH,040H,020H,010H,020H,040H,0FFH
TH_73: DB   02H,000H,0FFH,020H,010H,008H,004H,0FFH,000H
TH_74: DB   02H,000H,07EH,081H,081H,081H,081H,07EH,000H
```

```
TH_75: DB   02H,000H,0FFH,090H,090H,090H,090H,060H,000H
TH_76: DB   02H,000H,07EH,081H,081H,081H,085H,07EH,001H
TH_77: DB   02H,000H,0FFH,090H,098H,094H,092H,061H,000H
TH_78: DB   02H,000H,062H,091H,091H,091H,091H,04EH,000H
TH_79: DB   02H,000H,080H,080H,080H,0FFH,080H,080H,080H
TH_80: DB   02H,000H,0FEH,001H,001H,001H,001H,0FEH,000H
TH_81: DB   02H,000H,0F8H,004H,002H,001H,002H,004H,0F8H
TH_82: DB   02H,000H,0FFH,004H,008H,010H,008H,004H,0FFH
TH_83: DB   02H,000H,0C3H,024H,018H,018H,024H,0C3H,000H
TH_84: DB   02H,000H,0E0H,010H,008H,007H,008H,010H,0E0H
TH_85: DB   02H,000H,083H,095H,089H,095H,0A1H,0C1H,000H
TH_86: DB   02H,000H,00EH,011H,021H,022H,03FH,000H,000H
TH_87: DB   02H,000H,0FFH,021H,021H,021H,01EH,000H,000H
TH_88: DB   02H,000H,01EH,021H,021H,021H,012H,000H,000H
TH_89: DB   02H,000H,01EH,021H,021H,021H,0FFH,000H,000H
TH_90: DB   02H,000H,01EH,029H,029H,029H,01AH,000H,000H
TH_91: DB   02H,000H,010H,07FH,090H,080H,040H,000H,000H
TH_92: DB   02H,000H,0FFH,010H,020H,020H,01FH,000H,000H
TH_93: DB   02H,000H,000H,000H,000H,02FH,000H,000H,000H
TH_94: DB   02H,000H,000H,0FFH,00CH,012H,021H,000H,000H
TH_95: DB   02H,000H,000H,000H,000H,0FFH,000H,000H,000H
TH_96: DB   02H,000H,03FH,010H,020H,01FH,020H,01FH,000H
TH_97: DB   02H,000H,03FH,010H,020H,020H,01FH,000H,000H
TH_98: DB   02H,000H,01EH,021H,021H,021H,01EH,000H,000H
TH_99: DB   02H,000H,03FH,010H,020H,020H,010H,000H,000H
TH_100: DB  02H,000H,000H,01AH,029H,025H,016H,000H,000H
TH_101: DB  02H,000H,020H,020H,0FFH,020H,020H,000H,000H
TH_102: DB  02H,000H,03EH,001H,001H,002H,03FH,000H,000H
TH_103: DB  02H,000H,03CH,002H,001H,002H,03CH,000H,000H
TH_104: DB  02H,000H,03EH,001H,006H,006H,001H,03EH,000H
TH_105: DB  02H,000H,000H,033H,00CH,00CH,033H,000H,000H
TH_106: DB  02H,000H,021H,023H,025H,029H,031H,021H,000H
```

```
TH_107: DB    02H,000H,0FDH,000H,000H,000H,000H,000H,000H
TH_108: DB    02H,000H,00EH,000H,00EH,000H,000H,000H,000H
TH_109: DB    02H,001H,042H,0A4H,048H,012H,025H,042H,080H
TH_110: DB    02H,000H,000H,000H,000H,000H,03CH,042H,081H
TH_111: DB    02H,081H,042H,03CH,000H,000H,000H,000H,000H
TH_112: DB    02H,000H,054H,038H,07CH,038H,054H,000H,000H
TH_113: DB    02H,000H,008H,008H,03EH,008H,008H,000H,000H
TH_114: DB    02H,005H,006H,000H,000H,000H,000H,000H,000H
TH_115: DB    02H,000H,008H,008H,008H,008H,008H,000H,000H
TH_116: DB    02H,003H,003H,000H,000H,000H,000H,000H,000H
TH_117: DB    02H,001H,002H,004H,008H,010H,020H,040H,080H
TH_118: DB    02H,000H,07EH,081H,081H,081H,07EH,000H,000H
TH_119: DB    02H,000H,020H,041H,0FFH,001H,000H,000H,000H
TH_120: DB    02H,000H,043H,085H,089H,091H,061H,000H,000H
TH_121: DB    02H,000H,042H,091H,091H,06EH,000H,000H,000H
TH_122: DB    02H,00CH,014H,024H,044H,0FFH,004H,000H,000H
TH_123: DB    02H,000H,0FAH,091H,091H,091H,08EH,000H,000H
TH_124: DB    02H,000H,07EH,091H,091H,091H,04EH,000H,000H
TH_125: DB    02H,000H,080H,080H,080H,090H,0FFH,010H,000H
TH_126: DB    02H,000H,06EH,091H,091H,091H,06EH,000H,000H
TH_127: DB    02H,000H,072H,089H,089H,089H,07EH,000H,000H
TH_128: DB    02H,000H,028H,028H,028H,028H,028H,000H,000H
TH_129: DB    02H,000H,060H,080H,08DH,010H,060H,000H,000H
TH_130: DB    02H,000H,000H,008H,014H,022H,041H,000H,000H
TH_131: DB    02H,000H,000H,041H,022H,014H,008H,000H,000H
TH_132: DB    02H,000H,000H,000H,000H,000H,000H,000H,000H

TABLE_CODE2:
THX_1: DB     02H,000H,0C0H,0FEH,041H,0BEH,000H,000H,000H
       DB     01H,000H,000H,000H,000H,000H,001H,000H,000H
THX_2: DB     02H,000H,040H,080H,05EH,0E1H,07EH,080H,000H
       DB     01H,000H,000H,000H,000H,000H,000H,000H,001H
```

```
THX_3:  DB  02H,000H,05FH,0A3H,080H,080H,07FH,001H,0FFH
        DB  00H,000H,000H,000H,000H,000H,060H,020H,020H
THX_4:  DB  02H,000H,043H,0BFH,080H,080H,080H,07FH,000H
        DB  00H,000H,020H,050H,050H,020H,050H,0F0H,000H
THX_5:  DB  02H,000H,043H,0DFH,080H,080H,080H,07FH,000H
        DB  00H,020H,050H,020H,050H,0A0H,020H,0F0H,000H
THX_6:  DB  02H,000H,000H,020H,06CH,0AFH,0BEH,080H,000H
        DB  00H,000H,000H,020H,040H,020H,040H,060H,000H
THX_7:  DB  02H,000H,0C0H,0FFH,001H,001H,001H,0FFH,000H
        DB  01H,000H,000H,000H,000H,000H,000H,00FH,000H
THX_8:  DB  02H,07CH,0C2H,0C1H,00EH,00EH,001H,002H,0FCH
        DB  01H,000H,000H,000H,000H,000H,000H,000H,00FH
THX_9:  DB  02H,000H,0C0H,0FFH,010H,060H,060H,010H,0FFH
        DB  01H,000H,000H,000H,000H,000H,000H,000H,00FH
THX_10: DB  02H,000H,05FH,0A3H,080H,080H,080H,080H,07FH
        DB  00H,000H,000H,000H,000H,000H,000H,000H,0F0H
THX_11: DB  02H,000H,003H,05FH,0A0H,080H,080H,080H,07FH
        DB  00H,000H,000H,000H,000H,000H,000H,000H,0F0H
THX_12: DB  02H,000H,07FH,08CH,098H,098H,040H,0BFH,000H
        DB  01H,000H,000H,000H,000H,000H,000H,000H,001H
THX_13: DB  02H,000H,04FH,093H,090H,088H,0C4H,0FFH,000H
        DB  01H,000H,000H,000H,000H,000H,000H,000H,001H
THX_14: DB  02H,0C0H,0FFH,010H,060H,060H,010H,0FFH,000H
        DB  01H,000H,000H,000H,000H,006H,006H,007H,008H
THX_15: DB  02H,000H,06EH,0D9H,0D9H,0C1H,081H,081H,07EH
        DB  01H,000H,000H,000H,000H,000H,000H,001H,002H
THX_16: DB  02H,000H,000H,040H,080H,080H,080H,07FH,000H
        DB  01H,006H,006H,000H,000H,000H,000H,000H,000H
THX_17: DB  02H,000H,000H,0FFH,003H,000H,000H,000H,000H
        DB  01H,004H,00CH,013H,010H,008H,010H,000H,000H
THX_18: DB  02H,000H,000H,000H,000H,0FFH,003H,000H,000H
        DB  01H,000H,004H,012H,010H,00FH,000H,000H,000H
```

```
THX_19:  DB   02H,000H,000H,000H,000H,0FFH,003H,000H,000H
         DB   01H,010H,008H,004H,008H,01FH,000H,000H,000H
THX_20:  DB   02H,000H,00CH,010H,008H,010H,00FH,000H,000H
         DB   00H,000H,000H,000H,000H,000H,0F0H,000H,000H
THX_21:  DB   02H,000H,01FH,021H,021H,022H,01FH,000H,000H
         DB   00H,000H,020H,010H,010H,010H,0E0H,000H,000H
THX_22:  DB   02H,000H,000H,000H,000H,05FH,000H,000H,000H
         DB   00H,000H,0C0H,020H,020H,0C0H,000H,000H,000H
THX_23:  DB   02H,000H,03FH,021H,021H,021H,01EH,000H,000H
         DB   00H,000H,0F0H,000H,000H,000H,000H,000H,000H
THX_24:  DB   02H,000H,01EH,021H,021H,021H,03FH,000H,000H
         DB   00H,000H,000H,000H,000H,000H,0F0H,000H,000H
THX_25:  DB   02H,000H,03CH,002H,001H,002H,03FH,000H,000H
         DB   00H,000H,040H,020H,010H,020H,0C0H,000H,000H

PIC_1:   DB   02H,008H,018H,018H,018H,03CH,07EH,0FFH,0DBH
         DB   02H,099H,018H,038H,078H,078H,000H,000H,000H
         DB   02H,0,0,0,0,0,0,0,0
         DB   02H,0,0,0,0,0,0,0,0
PIC_2:   DB   02H,0B8H,0C0H,0E0H,0D0H,0B8H,0CEH,045H,085H
         DB   02H,005H,005H,005H,005H,005H,005H,005H,005H
         DB   02H,005H,005H,005H,00AH,014H,038H,060H,040H
         DB   02H,0,0,0,0,0,0,0,0
         DB   02H,0,0,0,0,0,0,0,0
PIC_3:   DB   02H,000H,01CH,01DH,03EH,07DH,0FCH,0FCH,0FCH
         DB   00H,040H,040H,040H,0C0H,040H,040H,040H,040H
         DB   02H,01CH,01CH,01CH,01DH,01EH,01DH,01CH,000H
         DB   00H,040H,040H,040H,040H,0C0H,040H,040H,040H
         DB   02H,0,0,0,0,0,0,0,0
         DB   02H,0,0,0,0,0,0,0,0
PIC_4:   DB   02H,018H,03CH,06EH,07EH,07EH,07EH,07EH,03CH
         DB   02H,018H,018H,03CH,07EH,066H,042H,000H,000H
```

```
        DB    02H,0,0,0,0,0,0,0
        DB    02H,0,0,0,0,0,0,0
PIC_5:  DB    02H,000H,020H,060H,0AEH,0B1H,071H,015H,02BH
        DB    02H,02BH,02BH,02DH,029H,022H,044H,0F8H,000H
        DB    02H,0,0,0,0,0,0,0
        DB    02H,0,0,0,0,0,0,0
;-------------------------------------
END
```

โปรแกรมภาษา VISUAL BASIC

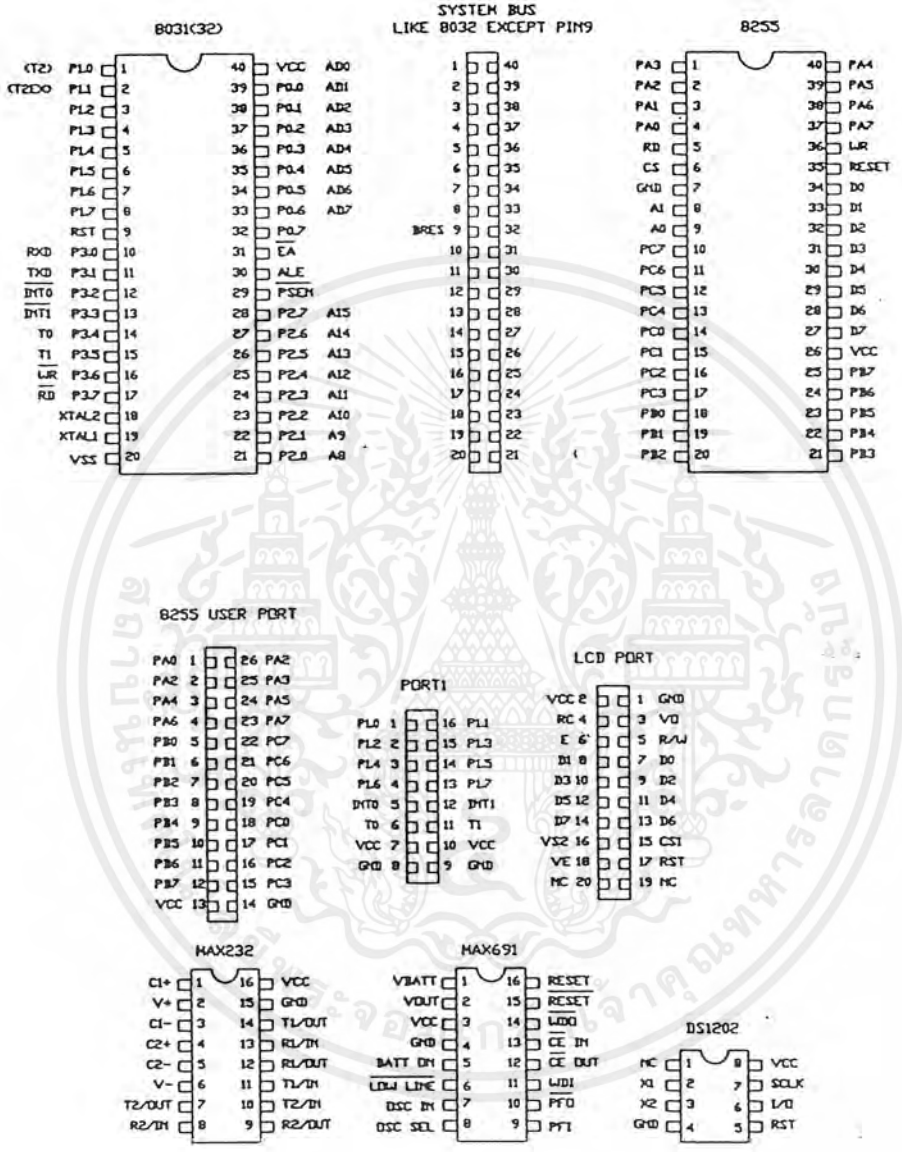
```
Public flag As Boolean


Private Sub Command1_Click()
Dim buffer As Variant
Dim model, color, pic As Integer
'=====Select Color of Character=====
flag = True
Select Case Combo1.Text
Case "Red": color = 1
Case "Green": color = 2
End Select
'==Select Type of Moving Character===
Select Case Combo2.Text
Case "LefttoRight": model = 1
Case "RighttoLeft": model = 2
Case "ToptoDown": model = 3
Case "DowntoTop": model = 4
End Select
Select Case Combo3.Text
Case "None": pic = 0
Case "Picture No.1": pic = 1
Case "Picture No.2": pic = 2
Case "Picture No.3": pic = 3
Case "Picture No.4": pic = 4
Case "Picture No.5": pic = 5
End Select
MSComm1.CommPort = 2
'========== change output buffer ==========


'MSComm1.OutBufferSize =???
```

```
'=============================================
MSComm1.PortOpen = True
buffer = Str(color) + "-" + Str(model) + "-" + Chr(Str(pic)) + "-" + Chr(Len(Text1.Text)) +
Text1.Text
MSComm1.Output = buffer
Command1.Enabled = False
Command2.Enabled = True
'for test Output buffer ==>Text1.Text = buffer
End Sub

Private Sub Command2_Click()
flag = False
MSComm1.PortOpen = False
Command1.Enabled = True
Command2.Enabled = False
End Sub

Private Sub Command4_Click()
If flag Then MSComm1.PortOpen = False
End
End Sub

Private Sub Form_Activate()
Text1.SetFocus
End Sub

Private Sub Form_Load()
Command1.Enabled = True
Command2.Enabled = False
End Sub
```

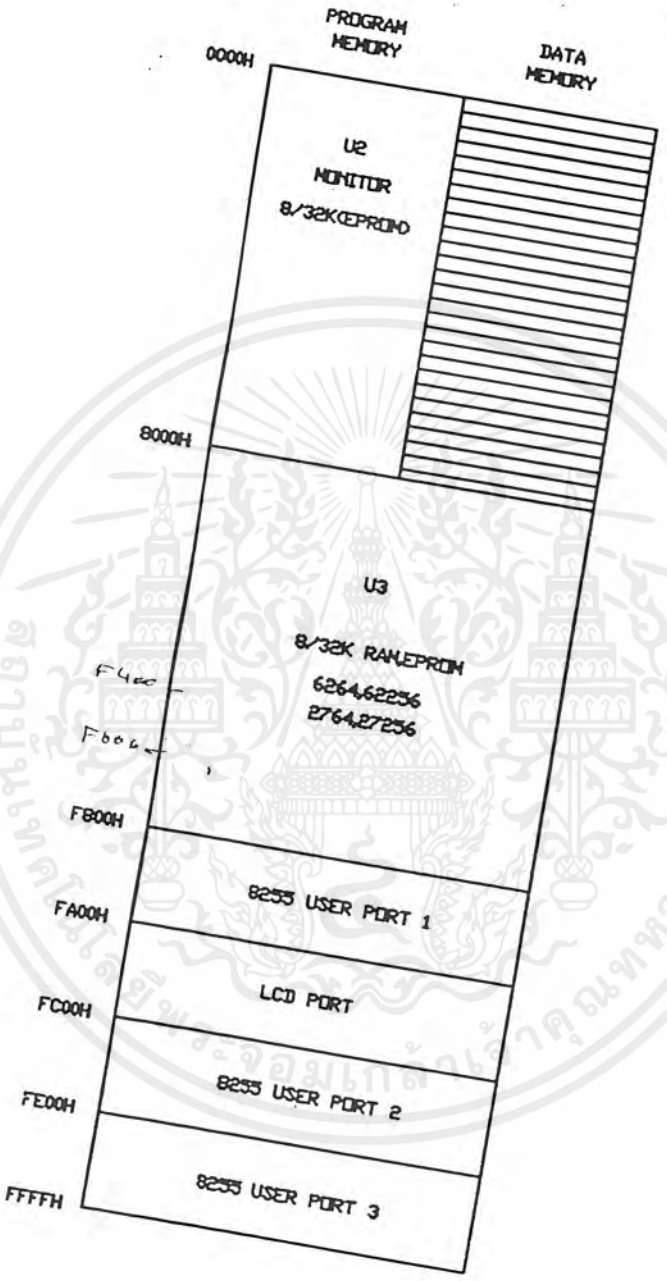
ภาคผนวก ง.

แสดงขาอุปกรณ์และกลุ่มคำสั่งพื้นฐาน

ภาพแสดง CONNECTOR , CHIP



MEMORY MAP



21

SPECIAL FUNCTION REGISTERS

Table 1 contains a list of all the SFRs and their addresses.
Comparing table 1 and figure 8 shows that all of the SFRs that are byte and bit addressable are located on the first column of the diagram in figure 8.

SYMBOL	NAME	ADDRESS
*ACC	Accumulator	0E0H
*B	B Register	0F0H
*PSW	Program Status Word	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer 2 Bytes	
DPL	Low Byte	82H
DPH	High Byte	83H
*P0	Port 0	80H
*P1	Port 1	90H
*P2	Port 2	0A0H
*P3	Port 3	0B0H
*IP	Interrupt Priority Control	0B8H
*IE	Interrupt Enable Control	0A8H
TMOD	Timer/Counter Mode Control	89H
*TCON	Timer/Counter Control	88H
*+T2CON	Timer/Counter 2 Control	0C8H
TH0	Timer/Counter 0 High Byte	8CH
TL0	Timer/Counter 0 Low Byte	8AH
TH1	Timer/Counter 1 High Byte	8DH
TL1	Timer/Counter 1 Low Byte	8BH
+TH2	Timer/Counter 1 High Byte	0CDH
+TL2	Timer/Counter 2 Low Byte	0CCH
+RCAP2H	T/C 2 Capture Reg. High Byte	0CBH
+RCAP2L	T/C 2 Capture Reg. Low Byte	0CAH
*SCON	Serial Control	98H
SBUF	Serial Data Buffer	99H
PCON	Power Control	87H
*IOCON (1)	IO Control	F8H

+ 80C52, 83C154 and 83C154D only * bit addressable
(1) 83C154 and 83C154D only

Table 1.

SFR MEMORY MAP

8 Bytes

F8	IOCON							FF
F0	B							F7
E8								EF
E0	ACC							E7
D8								DF
D0	PSW							D7
C8	T2CON		RCAP2L	RCAP2H	TL2	TH2		CF
C0								C7
B8	IP							BF
B0	P3							B7
A8	IE							AF
A0	P2							A7
98	SCON	SBUF						9F
90	P1							97
88	TCON	TMOD	TL0	TL1	TH0	TH1		8F
80	P0	SP	DPL	DPH				PCON 87

↑ bit addressable

Figure 8.

WHAT DO THE SFRs CONTAIN JUST AFTER POWER-ON OR A RESET ?

Table 2 lists the contents of each SFR after a power-on reset or a hardware reset.

REGISTER	VALUE IN BINARY
*ACC	0000 0000
*B	0000 0000
*PSW	0000 0000
SP	0000 0111
DPTR	0000 0000
*P0	1111 1111
*P1	1111 1111
*P2	1111 1111
*P3	1111 1111
*IP	XXX0 0000 80C51
	XXX0 0000 80C52
	0X00 0000 83C154/C154D
*IE	0XX0 0000 80C51
	0X000 0000 83C154/C154D and 80C52
TMOD	0000 0000

REGISTER	VALUE IN BINARY
*TCON	0000 0000
+*T2CON	0000 0000
TH0	0000 0000
TL0	0000 0000
TH1	0000 0000
TL1	0000 0000
+ TH2	0000 0000
+ TL2	0000 0000
+ RCAP2L	0000 0000
+ RCAP2H	0000 0000
*SCON	0000 0000
SBUF	Indeterminate
PCON	0XXX 0000 80C51 and 80C52
	000X 0000 83C154 and 83C154D
-*IOCON	0000 0000

*: bit addressable.
+ : 80C52, 83C154 and 83C154D only.
- : 83C154 and 83C154D only.
X : Undefined.

Table 2 : Contents of the SRFs after reset.

These SFRs that have their bits assigned for various functions are listed in this section. A brief description of each bit is provided for quick reference. For more detailed information refer to the Architecture chapter of this book.

PSW : PROGRAM STATUS WORD (BIT ADDRESSABLE)

CY	AC	F0	RS1	RS0	OV	F1	P

CY	PSW.7	Carry Flag.
AC	PSW.6	Auxiliary Carry Flag.
F0	PSW.5	Flag 0 available to the user for general purpose.
RS1	PSW.4	Register Bank selector bit 1 (SEE NOTE).
RS0	PSW.3	Register Bank selector bit 0 (SEE NOTE).
OV	PSW.2	Overflow Flag.
F1	PSW.1	Flag F1 available to the user for general purpose.
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of " 1 " bits in the accumulator.

Note :

The value presented by RS0 and RS1 selects the corresponding register bank.

RS1	RS0	REGISTER BANK	ADDRESS
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

* User software should not write 1s to reserved bits. These bits may be used in future MHS C51 products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1.

PCON : POWER CONTROL REGISTER (NOT BIT ADDRESSABLE)

SMOD	HPD	RPD	-	GF1	GF0	PD	IDL

SMOD	PCON.7	Double baud rate bit. If SMOD = 1, the baud rate is doubled when the serial part is used in mode 1, 2 and 3.
HPD	PCON.6	Hard Power Down. (83C154 and 83C154D only). The falling/rising edge of a signal connected on pin P3.5 Starts/Stops the Power-Down mode. A reset can also stop this mode.
RPD	PCON.5	Recover Power Down bit. (83C154 and 83C154D only). It's used to cancel a Power-Down/IDLE mode. If it's set, an interrupt (enable or disable) can cancel this mode. A reset can also stop this mode (see Note 1).
-	PCON.4	Not implemented, reserved for futur used*
GF1	PCON.3	General purpose bit.
GF0	PCON.2	General purpose bit.
PD	PCON.1	Power Down bit. If set, the oscillator is stopped. A reset or an interrupt (83C154 and 83C154D only) can cancel this mode (Note 1).
IDL	PCON.0	IDLE bit. If set the activity CPU is stopped. A reset or an interrupt can cancel this mode (See Note 1).

* User software should not write 1s to reserved bits. These bits may be used in future MHS C51 products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1.

Note 1 (83C154 and 83C154D only) :

- if RPD = 0 and if an interrupt cancels the mode Power-Down/IDLE, the next instruction to execute is a LCALL at the interrupt routine.
- RPD = 1 - if interrupt request is enable the next instruction to execute is a LCALL at the interrupt routine.
  - if interrupt request is disable, the program continue with the instruction immediately after the Power-Down/Idle instruction.

INTERRUPTS

In order to use any of the interrupts in the MHS C51, the following three steps must be taken.
1. Set the EA (enable all) bit in the IE register to 1.
2. Set the corresponding individual interrupt enable bit in the IE register to 1.
3. Begin the Interrupt service routine at the corresponding Vector Address of that interrupt. See Table below.

INTERRUPT SOURCE	VECTOR ADDRESS
IE0	0003H
TF0	000BH
IE1	0013H
TF1	001BH
RI & TI	0023H
TF2 & EXF2	002BH

In addition, for external interrupts, pins $\overline{INT0}$ and $\overline{INT1}$ (P3.2 and P3.3) must be set to 1, and depending on whether the interrupt is to be level or transition activated, bits IT0 or IT1 in the TCON register may need to be set to 1.

ITX = 0 level activated
ITX = 1 transition activated

IE : INTERRUPT ENABLE REGISTER (BIT ADDRESSABLE)

If the bit is 0, the corresponding interrupt is disabled. If the bit is 1, the corresponding interrupt is enabled.

EA	-	ET2	ES	ET1	EX1	ET0	EX0

EA	IE.7	Disables all interrupts. If EA = 0, no interrupt will be acknowledged. If EA = 1, interrupt source is individually enable or disabled by setting or clearing its enable bit.
-	IE.6	Not implemented, reserved for future use*.
ET2	IE.5	Enable or disable the Timer 2 overflow or capture interrupt (80C52, 83C154 and 83C154D only).
ES	IE.4	Enable or disable the Serial port interrupt.
ET1	IE.3	Enable or disable the Timer 1 overflow interrupt.
EX1	IE.2	Enable or disable External interrupt 1.
ET0	IE.1	Enable or disable the Timer 0 overflow interrupt.
EX0	IE.0	Enable or disable External Interrupt 0.

* User software should not write 1s to reserved bits. These bits may be used in future MHS C51 products to invoke new features. In that case, the reset or inactive value of the new bit will be 0, and its active value will be 1.

ASSIGNING HIGHER PRIORITY TO ONE MORE INTERRUPTS

In order to assign higher priority to an interrupt the corresponding bit in the IP register must be set to 1. Remember that while an interrupt service is in progress, it cannot be interrupted by a lower or same level interrupt.

PRIORITY WITHIN LEVEL

Priority within level is only to resolve simultaneous requests of the same priority level. From high to low, interrupt sources are listed below :

IE0
TF0
IE1
TF1
RI or TI
TF2 or EXF2

IP : INTERRUPT PRIORITY REGISTER (BIT ADDRESSABLE)

If the bit is 0, the corresponding interrupt has a lower priority and if the bit is the corresponding interrupt has a higher priority.

PCT	–	PT2	PS	PT1	PX1	PT0	PX0

PCT	IP.7	Defines the same priority level for all the source interrupt (83C154 and 83C154D only).
-	IP.6	Not implemented, reserved for future use*.
PT2	IP.5	Defines the Timer 2 interrupt priority level (80C52, 83C154 and 83C154D only).
PS	IP.4	Defines the Serial Port interrupt priority level.
PT1	IP.3	Defines the Timer 1 interrupt priority level.
PX1	IP.2	Defines External Interrupt priority level.
PT0	IP.1	Defines the Timer 0 interrupt priority level.
PX0	IP.0	Defines the External Interrupt 0 priority level.

* User software should not write 1s to reserved bits. These bits may be used in future MHS C51 products to invoke new features. In that case, the reset or inactive value of the now bit will be 0, and its active value will be 1.

IOCON : INPUT/OUTPUT CONTROL REGISTER (83C154 and 83C154D only)

WDT	T32	SERR	IZC	P3HZ	P2HZ	P1HZ	ALF

WDT	IOCON.7	Watch Dog Timer bit. Set when Timer 1 is overflow (TF = 1). The CPU is reset and the program is executed from address 0.
T32	IOCON.6	Timer 32 bits. The Timer 1 and Timer 0 are connected together to form a 32 bits Timer/Counter. If C/TO = 0, it's a Timer. If C/TO = 1, it's a counter.
SERR	IOCON.5	Serial Port Reception Error flag. Set when an overrun on frame error is received.
IZC	IOCON.4	Set/Cleared by software to select 100/10 K pull up resistance for Port 1, 2 and 3.
P3HZ	IOCON.3	When Set, Port 3 becomes a tri-state input. When cleared, the pull-up resistance value is selected by IZC.
P2HZ	IOCON.2	When Set, Port 2 becomes a tri-state input. When cleared, the pull-up resistance value is selected by IZC.
P1HZ	IOCON.1	When Set, Port 1 becomes a tri-state input. When cleared, the pull-up resistance value is selected by IZC.
ALF	IOCON.0	All Port tri-state. When Set and CPU in Power-Down mode, port 1, 2 and 3 are tri-state.

TCON : TIMER/COUNTER CONTROL REGISTER (BIT ADDRESSABLE)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1	TCON.7	Timer 1 overflow flag. Set by hardware when the Timer/Counter 1 overflows. Cleared by hardware as processor vectors to the interrupt service routine.
TR1	TCON.6	Timer 1 run control bit. Set/cleared by software to turn Timer/Counter ON/OFF.
TF0	TCON.5	Timer 0 overflow flag. Set by hardware when the Timer/Counter 0 overflows. Cleared by hardware as processor vectors to the service routine.
TR0	TCON.4	Timer 0 run control bit. Set/cleared by software to turn Timer/Counter 0 ON/OFF.
IE1	TCON.3	External Interrupt 1 edge flag. Set by hardware when External interrupt edge is detected. Cleared by hardware when interrupt is processed.
IT1	TCON.2	Interrupt 1 type control bit. Set/cleared by software to specify falling edge/flow level triggered External Interrupt.
IE0	TCON.1	External Interrupt 0 edge flag. Set by hardware when External Interrupt edge detected. Cleared by hardware when interrupt is processed.
IT0	TCON.0	Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low level triggered External Interrupt.

TMOD : TIMER/COUNTER MODE CONTROL REGISTER (NOT BIT ADDRESSABLE)

GATE	C/$\overline{T}$	M1	M0	GATE	C/$\overline{T}$	M1	M0
TIMER 1				TIMER 0			

GATE	When TRx (in TCON) is set and GATE = 1, TIMER/COUNTERx will run only while INTx pin is high (hardware control). When GATE = 0, TIMER/COUNTERx will run only while TRx = 1 (software control).
C/$\overline{T}$	Timer or Counter selector. Cleared for Timer operation (input from internal system clock). Set for Counter operation (input from Tx input pin).
M1	Mode selector bit (NOTE 1).
M0	Mode selector bit (NOTE 1).

NOTE 1 :

M1	M0		OPERATING MODE
0	0	0	13-bit Timer
0	1	1	16-bit Timer/Counter
1	0	2	8-bit Auto-Reload Timer/Counter
1	1	3	(Timer 0) TL0 is an 8-bit Timer/Counter controlled by the standard Timer 0 control bits, TH0 is an 8-bit Timer and is controlled by Timer 1 control bits.
1	1	3	(Timer 1) Timer/Counter 1 stopped.

TIMER SET-UP

Tables 3 through 6 give some values for TMOD which can be used to set up Timer 0 in different modes.

It is assumed that only one timer is being used at a time. It is desired to run Timers 0 and 1 simultaneously, in any mode, the value that in TMOD for Timer 0 must be ORed with the value shown for Timer 1 (Tables 5 and 6).

For example, if it is desired to run Timer 0 in mode 1 GATE (external control) and Timer 1 in mode 2 COUNTER, then the value must be loaded into TMOD is 69H (09H from Table 3 ORed with 60H from Table 6).

Moreover, it is assumed that the user, at this point, is not ready to turn the timers on and will do that a different point in the program by setting bit TRx (in TCON) to 1.

TIMER/COUNTER 0

MODE	TIMER 0 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13-bit Timer	00H	08H
1	16-bit Timer	01H	09H
2	8-bit Auto-Reload	02H	0AH
3	Two 8-bit Timers	03H	0BH

As a Timer : Table 3

MODE	TIMER 0 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13-bit Timer	04H	0CH
1	16-bit Timer	05H	0DH
2	8-bit Auto-Reload	06H	0EH
3	one 8-bit counter	07H	0FH

As a Counter : Table 4

Notes : 1. The Timer is turned ON/OFF by setting/clearing bit TR0 in the software.
2. The Timer is turned ON/OFF by the 1 to 0 transition on INT0 (P3.2) when TR0 = 1 (hardware control).

TIMER/COUNTER 1

MODE	TIMER 1 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13-bit Timer	00H	80H
1	16-bit Timer	10H	90H
2	8-bit Auto-Reload	20H	A0H
3	does not run	30H	B0H

As a Timer : Table 5

MODE	COUNTER 1 FUNCTION	TMOD	
		INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
0	13-bit Timer	40H	C0H
1	16-bit Timer	50H	D0H
2	8-bit Auto-Reload	60H	E0H
3	not available	–	–

As a Counter : Table 6

Notes : 1. The Timer is turned ON/OFF by setting/clearing bit TR1 in the software.
2. The Timer is turned ON/OFF by the 1 to 0 transition on INT1 (P3.2) when TR1 = 1 (hardware control).

T2CON : TIMER/COUNTER 2 CONTROL REGISTER (BIT ADDRESSABLE) (80C52, 83C154 and 83C154D only)

TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2	CP/RL2

TF2 T2CON.7 Timer 2 overflow flag set by hardware and cleared by software. TF2 cannot be set when either RCLK = 1 or CLK = 1

EXF2 T2CON.6 Timer 2 external flag set when either a capture or reload is caused by a negative transition on T2EX, and EXEN2 = 1. When Timer 2 interrupt is enabled, EXF2 = 1 will cause the CPU to vector to the Timer 2 interrupt routine. EXF2 must be cleared by software.

RCLK T2CON.5 Receive clock flag. When set, causes the Serial Port to use Timer 2 overflow pulses for its receive clock in modes 1 & 3. RCLK = 0 causes Timer 1 overflow to be used for the receive clock.

TCLK T2CON.4 Transmit clock flag. When set, causes the Serial Port use Timer 2 overflow pulses for its transmit clock in modes 1 & 3, TCLK = 0 causes Timer 1 overflows to be used for the transmit clock.

EXEN2 T2CON.3 Timer 2 external enable flag. When set, allows a capture or reload to occur as a result of negative transition on T2EX if Timer 2 is not being used to clock the Serial Port. EXEN2 = 0 causes Timer 2 to ignore events as T2EX.

TR2 T2CON.2 Software START/STOP control for Timer 2. A logic 1 starts the Timer.

$C/\overline{T2}$ T2CON.1 Timer or Counter select.

$CP/\overline{RL2}$ T2CON.0 Capture/Reload flag. When set, captures will occur on negative transitions at T2EX if EXEN2 = 1. When cleared, Auto-Reloads will occur either with Timer2 overflows or negative transitions at T2EX when EXEN2 = 1. When either RCLK = 1 or TCLK = 1, this bit is ignored and the Timer is forced to Auto-Reload on Timer 2 overflow.

TIMER/COUNTER 2 SET-UP

Except for the baud rate generator mode, the values given for T2CON do not include the setting of the TR2 bit. Therefore, bit TR2 must be set, separately, to turn the Timer on.

MODE	T2CON	
	INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
16-bit Auto-Reload	00H	08H
16-bit Capture	01H	09H
BAUD rate generator receive & transmit same baud rate	34H	36H
receive only	24H	26H
transmit only	14H	16H

As a Timer : Table 7

MODE	TMOD	
	INTERNAL CONTROL (NOTE 1)	EXTERNAL CONTROL (NOTE 2)
16-bit Auto-Reload	02H	0AH
16-bit Capture	03H	0BH

As a Counter : Table 8

Notes : 1. Capture/Reload occurs only Timer/Counter overflow.
2. Capture/Reload occurs on Timer/Counter overflow and a 1 to 0 transition on T2EX (P1.1) pin except when Timer 2 is used in the baud rate generating mode.

SCON : SERIAL PORT CONTROL REGISTER (BIT ADDRESSABLE)

SM0	SM1	SM2	REN	TN8	RB8	TI	RI

SM0	SCON.7	Serial Port mode specifier (NOTE 1).
SM1	SCON.6	Serial Port mode specifier (NOTE 1).
SM2	SCON.5	Enables the multiprocessor communication feature in mode 2 & 3. In mode 2 or 3, if SM2 is set to 1 then RI will not be activated if the received 9th data bit (RB8) is 0. In mode 1, if SM2 = 1 then RI will not be activated if a valid stop bit was not received. In mode 0, SM2 should be 0 (See table 9).
REN	SCON.4	Set/Cleared by software to Enable/Disable reception.
TB8	SCON.3	The 9th bit that will be transmitted in modes 2 & 3. Set/Cleared by software.
RB8	SCON.2	In modes 2 & 3, is the 9th data bit that was received. In mode 1, if SM2 = 0, RB8 is the stop bit that was received. In mode 0, RB8 is not used.
TI	SCON.1	Transmit interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or at the beginning of the stop bit in the other modes. Must be cleared by software.
RI	SCON.0	Receive interrupt flag. Set by hardware at the end of the 8th bit time in mode 0, or half way through the stop bit time in the other modes (except see SM2). Must be cleared by software.

NOTE 1 :

SM0	SM1	MODE	DESCRIPTION	BAUD RATE
0	0	0	SHIFT REGISTER	Fosc./12
0	1	1	8 bit UART	Variable
1	0	2	8 bit UART	Fosc./64 OR Fosc./32
1	1	3	8 bit UART	Variable

SERIAL PORT SET-UP

MODE	SCON	SM2 VARIATION
0	10H	Single Processor Environment (SM2 = 0)
1	50H	
2	90H	
3	D0H	
0	NA	Multiprocessor Environment (SM2 = 1)
1	70H	
2	B0H	
3	F0H	

Table 9

GENERATING BAUD RATES

Serial Port In Mode 0 :

Mode 0 has a fixed baud rate which is 1/12 of oscillator frequency. To run serial port in this mode none of the Timer/Counters need to be set up. Only the SCON register needs to be defined.

$$\text{Baud Rate} = \frac{\text{Osc Freq}}{12}$$

Serial Port In Mode 1 :

Mode 1 has a variable baud rate. The baud rate can be generated by either Timer 1 or Timer 2 (80C52, 83C154 and 83C154D only).

USING TIMER/COUNTER 1 TO GENERATE BAUD RATES

For this purpose, Timer 1 is used in mode 2 (Auto-Reload). Refer to Timer Setup section of this chapter.

$$\text{Baud Rate} = \frac{\text{K x Oscillator Freq.}}{32 \times 12 \times [256 - (\text{TH1})]}$$

if SMOD = 0, then K = 1.
If SMOD = 1, then K = 2. (SMOD is the PCON register).
Most of the time the user knows the baud rate and needs to know the reload value for TH1. Therefore, the equation to calculate TH1 can be written as :

$$\text{TH1} = 256 - \frac{\text{K x Osc Freq.}}{384 \times \text{baud rate}}$$

TH1 must be integer value. Rounding off TH1 to the nearest integer may not produce the desired baud rate. In this case, the user may have to choose another crystal frequency.
Since the PCON register is not bit addressable, one way to set the bit is logical ORing the PCON register (ie, ORL PCON, #80H). The address of PCON is 87H.

USING TIMER/COUNTER 2 TO GENERATE BAUD RATES

For this purpose, Timer 2 must be used in the baud rate generating mode. Refer to Timer 2 Setup Table in this chapter.
If Timer 2 is being clocked through pin T2 (P1.0) the baud rate is :

$$\text{Baud Rate} = \frac{\text{Timer 2 Overflow Rate}}{16}$$

And if it being clocked internally the baud rate is :

$$\text{Baud Rate} = \frac{\text{Osc Freq.}}{32 \times [65536 - (\text{RCAP2H, RCAP2L})]}$$

To obtain the reload value for RCAP2H and RCAP2L the above equation can be written as :

$$\text{RCAP2H, RCAP2L} = 65536 - \frac{\text{Osc Freq.}}{32 \times \text{Baud Rate}}$$

SERIAL PORT IN MODE 2

The baud rate is fixed in this mode and 1/32 or 1/64 of the oscillator frequency depending on the value of the SMOD bit in the PCON register.

In this mode none of the Timers are used and the clock comes from the internal phase 2 clock.

SMOD = 1, Baud Rate = 1/32 Osc Freq.

SMOD = 0, Baud Rate = 1/64 Osc Freq.

To set the SMOD bit : ORL PCON, #80H. The address of PCON is 87H.

SERIAL PORT IN MODE 3

The baud rate in mode 3 is variable and sets up exactly the same as in mode 1.

MHS C51 INSTRUCTION SET

Interrupt Response time : Refer to Hardware Description Chapter.

Instructions that Affect Flag Settings (1)

INSTRUC.	FLAG			INSTRUC.	FLAG		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLRC	0		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C, bit	X		
MUL	0	X		ANL C,/ bit	X		
DIV	0	X		ORL C, bit		X	
DA	X			ORL C, bit		X	
RRC	X			MOV C, bit		X	
RLC	X			CJNE	X		
SETB C	1						

(1) note that operations on SFR byte address 208 or bit addresses 209-215 (i.e., the PSW or bits in the PSW) will also affect flag settings.

Note on instruction set and addressing modes :

Rn	– Register R7-R0 of the currently selected Register Bank
direct	– 8-bit internal data location's address. This could be an Internal Data RAM location (0-127) or a SFR (i.e., I/O port, control register, status register, etc. (128-255)).
@ Ri	– 8-bit internal data RAM location (0-255) addresses indirectly through register R1 or R0.
# data	– 8-bit constant included in instruction.
# data 16	– 16-bit constant included in instruction.
addr 16	– 16-bit destination address. Used by LCALL & LJMP. Abranch can be anywhere within the 64K-byte Program memory address space
addr 11	– 11-bit destination address. Used by ACALL & AJMP. The branch will be within the same 2K-byte page of program memory as the first byte of the following instruction
rel	– Signed (two's complement) 8-bit offset byte. Used by SJMP and all conditionnal jumps. Range is–128 to + 127 bytes relative to first byte of the following instruction.
bit	– Direct Addressed bit in internal Data RAM or special Function Register.

MNEMONIC	DESCRIPTION	BYTE	OSCIL. PERIOD
ARITHMETIC OPERATIONS			
ADD A, Rn	Add register to Accumulator	1	12
ADD A, direct	Add direct byte to Accumulator	2	12
ADD A, @Ri	Add indirect RAM to Accumulator	1	12
ADD A, #data	Add immediate data to Accumulator	2	12
ADDCA, Rn	Add register to Accumulator with Carry	1	12
ADDCA, direct	Add direct byte to Accumulator with Carry	2	12
ADDCA, @Ri	Add indirect RAM to Accumulator with Carry	1	12
ADDCA, #data	Add immediate data to Acc with Carry	2	12
SUBB A, Rn	Subtract Register from Acc with borrow	1	12
SUBB A, direct	Subtract direct byte from Acc with borrow	2	12
SUBB A, @Ri	Subtract indirect RAM from ACC with borrow	1	12
SUBB A, #data	Subtract immediate data from Acc with borrow	2	12
INC A	Increment Accumulator	1	12
INC Rn	Increment register	1	12
INC direct	Increment direct byte	2	12
INC @Ri	Increment direct RAM	1	12
DEC A	Decrement Accumulator	1	12
DEC Rn	Decrement Register	1	12
DEC direct	Decrement direct byte	2	12
DEC @Ri	Decrement indirect RAM	1	12

Table 10 : 80C51 Instruction Set Summary.

MNEMONIC	DESCRIPTION	BYTE	OSCIL. PERIOD
ARITHMETIC OPERATIONS (continued)			
INC DPTR	Increment Data Pointer	1	24
MUL AB	Multiply A & B	1	48
DIV AB	Divide A by B	1	48
DA A	Decimal Adjust Accumulator	1	12
LOGICAL OPERATIONS			
ANL A, Rn	AND Register to Accumulator	1	12
ANL A, direct	AND direct byte to Accumulator	2	12
ANL A, @Ri	AND indirect RAM to Accumulator	1	12
ANL A, #data	AND immediate data to Accumulator	2	12
ANL direct, A	AND Accumulator to direct byte	2	12
ANL direct, #data	AND immediate data to direct byte	3	24
ORL A, Rn	OR register to Accumulator	1	12
ORL A, direct	OR direct byte to Accumulator	2	12
ORL A, @Ri	OR indirect RAM to Accumulator	1	12
ORL A, #data	OR immediate data to Accumulator	2	12
ORL direct, A	OR Accumulator to direct byte	2	12
ORL direct, #data	OR immediate data to direct byte	3	24
XRL A, Rn	Exclusive-OR register to Accumulator	1	12
XRL A, direct	Exclusive-OR direct byte to accumulator	2	12
XRL A, @Ri	Exclusive-OR indirect RAM to Accumulator	1	12
XRL A, #data	Exclusive-OR immediate data to Accumulator	2	12
XRL direct, A	Exclusive-OR Accumulator to direct byte	2	12
XRL direct, #data	Exclusive-OR immediate data to direct byte	3	24
CLR A	Clear Accumulator	1	12
CPL A	Complement Accumulator	1	12

MNEMONIC	DESCRIPTION	BYTE	OSCIL. PERIOD
LOGICAL OPERATIONS (continued)			
RL A	Rotate Accumulator Left	1	12
RLC A	Rotate Accumulator Left through the Carry	1	12
RR A	Rotate Accumulator Right	1	12
RRC A	Rotate Accumulator Right through the Carry	1	12
SWAP A	Swap nibbles within the Accumulator	1	12
DATA TRANSFER			
MOV A, Rn	Move Register to Accumulator	1	12
MOV A, direct	Move direct byte to Accumulator	2	12
MOV A, @Ri	Move indirect RAM to Accumulator	1	12
MOV A, #data	Move immediate data to Accumulator	2	12
MOV Rn, A	Move Accumulator to register	1	12
MOV Rn, direct	Move direct byte to register	2	24
MOV Rn, #data	Move immediate data to register	2	12
MOV direct, A	Move Accumulator to direct byte	2	12
MOV direct, Rn	Move register to direct byte	2	24
MOV direct, direct	Move direct byte to direct	3	24
MOV direct, @Ri	Move indirect RAM to direct byte	2	24
MOV direct, #data	Move immediate data to direct byte	3	24
MOV @Ri, A	Move Accumulator to indirect RAM	1	12

MNEMONIC	DESCRIPTION	BYTE	OSCIL. PERIOD
DATA TRANSFER (continued)			
MOV @Ri, direct	Move direct by to indirect RAM	2	24
MOV @Ri, #data	Move immediate data to indirect RAM	2	12
MOV DPTR, #data16	Load Data Pointer with a 16-bit constant	3	24
MOVCA, @A+DPTR	Move Code byte relative to DPTR to Acc	1	24
MOVCA, @A+PC	Move Code byte relative to PC to Acc	1	24
MOVXA, @Ri	Move External RAM (8-bit addr) to Acc	1	24
MOVX A, @DPTR	Move External RAM (16-bit addr) to Acc	1	24
MOVX@Ri, A	Move Acc to External RAM (8-bit addr)	1	24
MOVX @DPTR, A	Move Acc to External RAM (16-bit addr)	1	24
PUSH direct	Push direct byte only stack	2	24
POP direct	Pop direct byte from stack	2	24
XCH A, Rn	Exchange register with Accumulator	1	12
XCH A, direct	Exchange direct byte with Accumulator	2	12
XCH A, @Ri	Exchange indirect RAM with Accumulator	1	12
XCHD A, @Ri	Exchange loworder Digit indirect RAM with Acc	1	12

MNEMONIC	DESCRIPTION	BYTE	OSCIL. PERIOD
BOOLEAN VARIABLE MANIPULATION			
CLR C	Clear Carry	1	12
CLR bit	Clear direct bit	2	12
SETB C	Set Carry	1	12
SETB bit	Set direct bit	2	12
CPL C	Complement Carry	1	12
CPL bit	Complement direct bit	2	12
ANL C, bit	AND direct bit to Carry	2	24
ANL C, /bit	AND complement of direct bit to Carry	2	24
ORL C, bit	OR direct bit to Carry	2	24
ORL C, /bit	OR complement of direct bit to Carry	2	24
MOV C, bit	Move direct bit to Carry	2	12
MOV bit, C	Move Carry to direct bit	2	24
JC rel	Jump if Carry is set	2	24
JNC rel	Jump if Carry not set	2	24
JB bit, rel	Jump if direct Bit is set	3	24
JNB bit, rel	Jump if direct Bit is Not set	3	24
JBC bit, rel	Jump if direct Bit is set & clear bit	3	24
PROGRAM BRANCHING			
ACALLK addr11	Absolute Subroutine Call	2	24
LCALL addr16	Long Subroutine Call	3	24
RET	Return from Subroutine	1	24
RETI	Return from interrupt	1	24
AJMPaddr11	Absolute Jump	2	24
LJMPaddr16	Long Jump	3	24
SJMP rel	Short Jump (relative addr)	2	24

MNEMONIC	DESCRIPTION	BYTE	OSCIL. PERIOD
PROGRAM BRANCHING (continued)			
JMP @A+DPTR	Jump direct relative to the DPTR	1	24
JZ rel	Jump if Accumulator is zero	2	24
JNZ rel	Jump if Accumulator is not Zero	2	24
CNJE A, direct, rel	Compare direct byte to Acc and Jump if Not Equal	3	24
CJNE A, #data, rel	Compare immediate to Acc and Jump if Not Equal	3	24

MNEMONIC	DESCRIPTION	BYTE	OSCIL. PERIOD
PROGRAM BRANCHING (continued)			
CJNE Rn, #data, rel	Compare Immediate to register and Jump if Not Equal	3	24
CJNE @Ri, #data, rel	Compare immediate to indirect and Jump if Not Equal	3	24
DJNZ Rn, rel	Decrement register and Jump if Not Zero	2	24
DJNZ direct, rel	Decrement direct byte and Jump if Not Zero	3	24
NOP	No Operation	1	12

Interfacing the Serial / RS232 Port V5.0

The Serial Port is harder to interface than the Parallel Port. In most cases, any device you connect to the serial port will need the serial transmission converted back to parallel so that it can be used. This can be done using a UART. On the software side of things, there are many more registers that you have to attend to than on a Standard Parallel Port. (SPP)

So what are the advantages of using serial data transfer rather than parallel?

1. Serial Cables can be longer than Parallel cables. The serial port transmits a '1' as -3 to -25 volts and a '0' as +3 to +25 volts where as a parallel port transmits a '0' as 0v and a '1' as 5v. Therefore the serial port can have a maximum swing of 50V compared to the parallel port which has a maximum swing of 5 Volts. Therefore cable loss is not going to be as much of a problem for serial cables than they are for parallel.
2. You don't need as many wires than parallel transmission. If your device needs to be mounted a far distance away from the computer then 3 core cable (Null Modem Configuration) is going to be a lot cheaper that running 19 or 25 core cable. However you must take into account the cost of the interfacing at each end.
3. Infra Red devices have proven quite popular recently. You may of seen many electronic diaries and palmtop computers which have infra red capabilities build in. However could you imagine transmitting 8 bits of data at the one time across the room and being able to (from the devices point of view) decipher which bits are which? Therefore serial transmission is used where one bit is sent at a time. IrDA-1 (The first infra red specifications) was capable of 115.2k baud and was interfaced into a UART. The pulse length however was cut down to 3/16th of a RS232 bit length to conserve power considering these devices are mainly used on diaries, laptops and palmtops.
4. Microcontroller's have also proven to be quite popular recently. Many of these have in built SCI (Serial Communications Interfaces) which can be used to talk to the outside world. Serial Communication reduces the pin count of these MPU's. Only two pins are commonly used, Transmit Data (TXD) and Receive Data (RXD) compared with at least 8 pins if you use a 8 bit Parallel method (You may also require a Strobe).

Table of Contents

Part 1 : Hardware (PC's)
Hardware Properties
Serial Pinouts (D25 and D9 connectors)
Pin Functions
Null Modems
Loopback Plugs
DTE/DCE Speeds
Flow Control
The UART (8250's and Compatibles)
Type of UARTS (For PC's)
Part 2 : Serial Ports' Registers (PC's)
Port Addresses and IRQ's
Table of Registers
DLAB ?
Interrupt Enable Register (IER)
Interrupt Identification Register (IIR)
First In / First Out Control Register (FCR)
Line Control Register (LCR)
Modem Control Register (MCR)
Line Status Register (LSR)
Modem Status Register (MSR)
Scratch Register
Part 3 : Programming (PC's)

Polling or Interrupt Driven?
Source Code - Termpoll.c (Polling Version)
Source Code - Buff1024.c (ISR Version)
Interrupt Vectors
Interrupt Service Routine
UART Configuration
Main Routine (Loop)
Determining the type of UART via Software
Part 4 : External Hardware - Interfacing Methods
RS-232 Waveforms
RS-232 Level Converters
Making use of the Serial Format
8250 and compatable UART's
CDP6402, AY-5-1015 / D36402R-9 etc UARTs
CDP6402 UART Examples
Interfacing Example : Analog Sampling Via Serial Port
Microcontrollers
Feedback
Feedback

Part One : Hardware (PC's)

Hardware Properties

Devices which use serial cables for their communication are split into two categories. These are DCE (Data Communications Equipment) and DTE (Data Terminal Equipment.) Data Communications Equipment are devices such as your modem, TA adapter, plotter etc while Data Terminal Equipment is your Computer or Terminal.

The electrical specifications of the serial port is contained in the EIA (Electronics Industry Association) RS232C standard. It states many parameters such as -

1. A "Space" (logic 0) will be between +3 and +25 Volts.
2. A "Mark" (Logic 1) will be between -3 and -25 Volts.
3. The region between +3 and -3 volts is undefined.
4. An open circuit voltage should never exceed 25 volts. (In Reference to GND)
5. A short circuit current should not exceed 500mA. The driver should be able to handle this without damage. (Take note of this one!)

Above is no where near a complete list of the EIA standard. Line Capacitance, Maximum Baud Rates etc are also included. For more information please consult the EIA RS232-C standard. It is interesting to note however, that the RS232C standard specifies a maximum baud rate of 20,000 BPS!, which is rather slow by today's standards. A new standard, RS-232D has been recently released.

Serial Ports come in two "sizes", There are the D-Type 25 pin connector and the D-Type 9 pin connector both of which are male on the back of the PC, thus you will require a female connector on your device. Below is a table of pin connections for the 9 pin and 25 pin D-Type connectors.

Serial Pinouts (D25 and D9 Connectors)

D-Type-25 Pin No.	D-Type-9 Pin No.	Abbreviation	Full Name
Pin 2	Pin 3	TD	Transmit Data
Pin 3	Pin 2	RD	Receive Data
Pin 4	Pin 7	RTS	Request To Send
Pin 5	Pin 8	CTS	Clear To Send
Pin 6	Pin 6	DSR	Data Set Ready
Pin 7	Pin 5	SG	Signal Ground
Pin 8	Pin 1	CD	Carrier Detect
Pin 20	Pin 4	DTR	Data Terminal Ready
Pin 22	Pin 9	RI	Ring Indicator

Table 1 : D Type 9 Pin and D Type 25 Pin Connectors

Pin Functions

Abbreviation	Full Name	Function
TD	Transmit Data	Serial Data Output (TXD)
RD	Receive Data	Serial Data Input (RXD)
CTS	Clear to Send	This line indicates that the Modem is ready to exchange data.
DCD	Data Carrier Detect	When the modem detects a "Carrier" from the modem at the other end of the phone line, this Line becomes active.
DSR	Data Set Ready	This tells the UART that the modem is ready to establish a link.
DTR	Data Terminal Ready	This is the opposite to DSR. This tells the Modem that the UART is ready to link.
RTS	Request To Send	This line informs the Modem that the UART is ready to exchange data.
RI	Ring Indicator	Goes active when modem detects a ringing signal from the PSTN.

Null Modems

A Null Modem is used to connect two DTE's together. This is commonly used as a cheap way to network games or to transfer files between computers using Zmodem Protocol, Xmodem Protocol etc. This can also be used with many Microprocessor Development Systems.

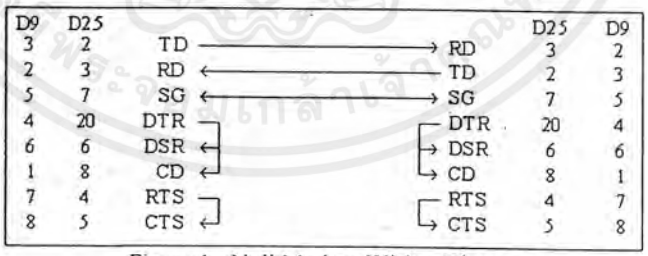


Figure 1 : Null Modem Wiring Diagram

Above is my preferred method of wiring a Null Modem. It only requires 3 wires (TD, RD & SG) to be wired straight through thus is more cost effective to use with long cable runs. The theory of operation is reasonably easy. The aim is to make to computer think it is talking to a modem rather than another computer. Any data transmitted from the first computer must be received by the second thus TD is connected to RD. The second computer must have the same set-up thus RD is connected to TD. Signal Ground (SG) must also be connected so both grounds are common to each computer.

The Data Terminal Ready is looped back to Data Set Ready and Carrier Detect on both computers. When the Data Terminal Ready is asserted active, then the Data Set Ready and Carrier Detect immediately become active. At this point the computer thinks the Virtual Modem to which it is connected is ready and has detected the carrier of the other modem.

All left to worry about now is the Request to Send and Clear To Send. As both computers communicate together at the same speed, flow control is not needed thus these two lines are also linked together on

each computer. When the computer wishes to send data, it asserts the Request to Send high and as it's hooked together with the Clear to Send, It immediately gets a reply that it is ok to send and does so.

Notice that the ring indicator is not connected to anything of each end. This line is only used to tell the computer that there is a ringing signal on the phone line. As we don't have a modem connected to the phone line this is left disconnected.

LoopBack Plug

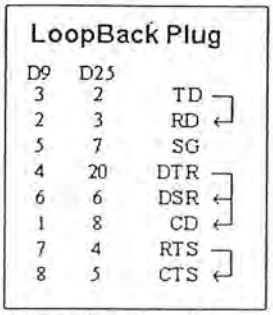


Figure 2 :.Loopback Plug Wiring Diagram

This loopback plug can come in extremely handy when writing Serial / RS232 Communications Programs. It has the receive and transmit lines connected together, so that anything transmitted out of the Serial Port is immediately received by the same port. If you connect this to a Serial Port an load a Terminal Program, anything you type will be immediately displayed on the screen. This can be used with the examples later in this tutorial.

Please note that this is not intended for use with Diagnostic Programs and thus will probably not work. For these programs you require a differently wired Loop Back plug which may vary from program to program.

DTE / DCE Speeds

We have already talked briefly about DTE & DCE. A typical Data Terminal Device is a computer and a typical Data Communications Device is a Modem. Often people will talk about DTE to DCE or DCE to DCE speeds. DTE to DCE is the speed between your modem and computer, sometimes referred to as your terminal speed. This should run at faster speeds than the DCE to DCE speed. DCE to DCE is the link between modems, sometimes called the line speed.

Most people today will have 28.8K or 33.6K modems. Therefore we should expect the DCE to DCE speed to be either 28.8K or 33.6K. Considering the high speed of the modem we should expect the DTE to DCE speed to be about 115,200 BPS.(Maximum Speed of the 16550a UART) This is where some people often fall into a trap. The communications program which they use have settings for DCE to DTE speeds. However they see 9.6 KBPS, 14.4 KBPS etc and think it is your modem speed.

Today's Modems should have Data Compression build into them. This is very much like PK-ZIP but the software in your modem compresses and decompresses the data. When set up correctly you can expect compression ratios of 1:4 or even higher. 1 to 4 compression would be typical of a text file. If we were transferring that text file at 28.8K (DCE-DCE), then when the modem compresses it you are actually transferring 115.2 KBPS between computers and thus have a DCE-DTE speed of 115.2 KBPS. Thus this is why the DCE-DTE should be much higher than your modem's connection speed.

Some modem manufacturers quote a maximum compression ratio as 1:8. Lets say for example its on a new 33.6 KBPS modem then we may get a maximum 268,800 BPS transfer between modem and UART. If you only have a 16550a which can do 115,200 BPS tops, then you would be missing out on a extra bit of performance. Buying a 16C650 should fix your problem with a maximum transfer rate of 230,400 BPS.

However don't abuse your modem if you don't get these rates. These are MAXIMUM compression ratios. In some instances if you try to send a already compressed file, your modem can spend more time trying the compress it, thus you get a transmission speed less than your modem's connection speed. If this occurs try turning off your data compression. This should be fixed on newer modems. Some files compress easier than others thus any file which compresses easier is naturally going to have a higher compression ratio.

For more information on the above topic of terminal and line speeds, try Configuring Your Modem For Optimum Performance.

Flow Control

So if our DTE to DCE speed is several times faster than our DCE to DCE speed the PC can send data to

released the 16750 UART which has 64 Byte FIFO's. Data Sheets for the TL16C750 are available from the Texas Instruments Site.

Types of UARTS (For PC's)

8250	First UART in this series. It contains no scratch register. The 8250A was an improved version of the 8250 which operates faster on the bus side.
8250A	This UART is faster than the 8250 on the bus side. Looks exactly the same to software than 16450.
8250B	Very similar to that of the 8250 UART.
16450	Used in AT's (Improved bus speed over 8250's). Operates comfortably at 38.4KBPS. Still quite common today.
16550	This was the first generation of buffered UART. It has a 16 byte buffer, however it doesn't work and is replaced with the 16550A.
16550A	Is the most common UART use for high speed communications eg 14.4K & 28.8K Modems. They made sure the FIFO buffers worked on this UART.
16650	Very recent breed of UART. Contains a 32 byte FIFO, Programmable X-On / X-Off characters and supports power management.
16750	Produced by Texas Instruments. Contains a 64 byte FIFO.

Part Two : Serial Port's Registers (PC's)

Port Addresses & IRQ's

Name	Address	IRQ
COM 1	3F8	4
COM 2	2F8	3
COM 3	3E8	4
COM 4	2E8	3

Table 3 : Standard Port Addresses

Above is the standard port addresses. These should work for most P.C's. If you just happen to be lucky enough to own a IBM P/S2 which has a micro-channel bus, then expect a different set of addresses and IRQ's. Just like the LPT ports, the base addresses for the COM ports can be read from the BIOS Data Area.

Start Address	Function
0000:0400	COM1's Base Address
0000:0402	COM2's Base Address
0000:0404	COM3's Base Address
0000:0406	COM4's Base Address

Table 4 - COM Port Addresses in the BIOS Data Area;

The above table shows the address at which we can find the Communications (COM) ports addresses in the BIOS Data Area. Each address will take up 2 bytes. The following sample program in C, shows how you can read these locations to obtain the addresses of your communications ports.

```
#include <stdio.h>
#include <dos.h>

void main(void)
{
 unsigned int far *ptraddr;  /* Pointer to location of Port Addres
 unsigned int address;       /* Address of Port */
 int a;
```

```
ptraddr=(unsigned int far *)0x00000400;

for (a = 0; a <  4; a++)
  {
   address = *ptraddr;
   if (address == 0)
                printf("No port found for COM%d \n",a+1);
   else
                printf("Address assigned to COM%d is %Xh\n",a+1,ad
   *ptraddr++;
  }
}
```

Table of Registers

Base Address	DLAB	Read/Write	Abr.	Register Name
+0	=0	Write	-	Transmitter Holding Buffer
	=0	Read	-	Receiver Buffer
	=1	Read/Write	-	Divisor Latch Low Byte
+1	=0	Read/Write	IER	Interrupt Enable Register
	=1	Read/Write	-	Divisor Latch High Byte
+2	-	Read	IIR	Interrupt Identification Register
	-	Write	FCR	FIFO Control Register
+3	-	Read/Write	LCR	Line Control Register
+4	-	Read/Write	MCR	Modem Control Register
+5	-	Read	LSR	Line Status Register
+6	-	Read	MSR	Modem Status Register
+7	-	Read/Write	-	Scratch Register

Table 5 : Table of Registers

DLAB ?

You will have noticed in the table of registers that there is a DLAB column. When DLAB is set to '0' or '1' some of the registers change. This is how the UART is able to have 12 registers (including the scratch register) through only 8 port addresses. DLAB stands for Divisor Latch Access Bit. When DLAB is set to '1' via the line control register, two registers become available from which you can set your speed of communications measured in bits per second.

The UART will have a crystal which should oscillate around 1.8432 MHZ. The UART incorporates a divide by 16 counter which simply divides the incoming clock signal by 16. Assuming we had the 1.8432 MHZ clock signal, that would leave us with a maximum, 115,200 hertz signal making the UART capable of transmitting and receiving at 115,200 Bits Per Second (BPS). That would be fine for some of the faster modems and devices which can handle that speed, but others just wouldn't communicate at all. Therefore the UART is fitted with a Programmable Baud Rate Generator which is controlled by two registers.

Lets say for example we only wanted to communicate at 2400 BPS. We worked out that we would have to divide 115,200 by 48 to get a workable 2400 Hertz Clock. The "Divisor", in this case 48, is stored in the two registers controlled by the "Divisor Latch Access Bit". This divisor can be any number which can be stored in 16 bits (ie 0 to 65535). The UART only has a 8 bit data bus, thus this is where the two registers are used. The first register (Base + 0) when DLAB = 1 stores the "Divisor latch low byte" where as the second register (base + 1 when DLAB = 1) stores the "Divisor latch high byte."

Below is a table of some more common speeds and their divisor latch high bytes & low bytes. Note that all the divisors are shown in Hexadecimal.

Speed (BPS)	Divisor (Dec)	Divisor Latch High Byte	Divisor Latch Low Byte
50	2304	09h	00h
300	384	01h	80h
600	192	00h	C0h
2400	48	00h	30h
4800	24	00h	18h
9600	12	00h	0Ch
19200	6	00h	06h
38400	3	00h	03h
57600	2	00h	02h
115200	1	00h	01h

Table 6 : Table of Commonly Used Baudrate Divisors

Interrupt Enable Register (IER)

Bit	Notes
Bit 7	Reserved
Bit 6	Reserved
Bit 5	Enables Low Power Mode (16750)
Bit 4	Enables Sleep Mode (16750)
Bit 3	Enable Modem Status Interrupt
Bit 2	Enable Receiver Line Status Interrupt
Bit 1	Enable Transmitter Holding Register Empty Interrupt
Bit 0	Enable Received Data Available Interrupt

Table 7 : Interrupt Enable Register

The Interrupt Enable Register could possibly be one of the easiest registers on a UART to understand. Setting Bit 0 high enables the Received Data Available Interrupt which generates an interrupt when the receiving register/FIFO contains data to be read by the CPU.

Bit 1 enables Transmit Holding Register Empty Interrupt. This interrupts the CPU when the transmitter buffer is empty. Bit 2 enables the receiver line status interrupt. The UART will interrupt when the receiver line status changes. Likewise for bit 3 which enables the modem status interrupt. Bits 4 to 7 are the easy ones. They are simply reserved. (If only everything was that easy!)

Interrupt Identification Register (IIR)

Bit			Notes
Bits 6 and 7	Bit 6	Bit 7	
	0	0	No FIFO
	0	1	FIFO Enabled but Unusable
	1	1	FIFO Enabled
Bit 5	64 Byte Fifo Enabled (16750 only)		
Bit 4	Reserved		
Bit 3	0		Reserved on 8250, 16450
	1		16550 Time-out Interrupt Pending
Bits 1 and 2	Bit 2	Bit 1	
	0	0	Modem Status Interrupt
	0	1	Transmitter Holding Register Empty Interrupt
	1	0	Received Data Available Interrupt
	1	1	Receiver Line Status Interrupt
Bit 0	0		Interrupt Pending
	1		No Interrupt Pending

Table 8 : Interrupt Identification Register

The interrupt identification register is a read only register. Bits 6 and 7 give status on the FIFO Buffer. When both bits are '0' no FIFO buffers are active. This should be the only result you will get from a 8250 or 16450. If bit 7 is active but bit 6 is not active then the UART has it's buffers enabled but are unusable. This occurs on the 16550 UART where a bug in the FIFO buffer made the FIFO's unusable. If both bits are '1' then the FIFO buffers are enabled and fully operational.

Bits 4 and 5 are reserved. Bit 3 shows the status of the time-out interrupt on a 16550 or higher.

Lets jump to Bit 0 which shows whether an interrupt has occurred. If an interrupt has occurred it's status will shown by bits.1 and 2. These interrupts work on a priority status. The Line Status Interrupt has the highest Priority, followed by the Data Available Interrupt, then the Transmit Register Empty Interrupt and then the Modem Status Interrupt which has the lowest priority.

First In / First Out Control Register (FCR)

Bit			Notes
Bits 6 and 7	Bit 7	Bit 6	
	0	0	Interrupt Trigger Level
	0	1	1 Byte
	1	0	4 Bytes
	1	1	8 Bytes
Bit 5	Enable 64 Byte FIFO (16750 only)		
Bit 4	Reserved		
Bit 3	DMA Mode Select. Change status of RXRDY & TXRDY pins from mode 1 to mode 2.		
Bit 2	Clear Transmit FIFO		
Bit 1	Clear Receive FIFO		
Bit 0	Enable FIFO's		

Table 9 : FIFO Control Register

FO register is a write only register. This register is used to control the FIFO (First In / First Out) which are found on 16550's and higher.

ables the operation of the receive and transmit FIFO's. Writing a '0' to this bit will disable the n of transmit and receive FIFO's, thus you will loose all data stored in these FIFO buffers.

d 2 control the clearing of the transmit or receive FIFO's. Bit 1 is responsible for the receive ile bit 2 is responsible for the transmit buffer. Setting these bits to 1 will only clear the

contents of the FIFO and will not affect the shift registers. These two bits are self resetting, thus you don't need to set the bits to '0' when finished.

Bit 3 enables the DMA mode select which is found on 16550 UARTs and higher. More on this later. Bits 4 and 5 are those easy type again, Reserved.

Bits 6 and 7 are used to set the triggering level on the Receive FIFO. For example if bit 7 was set to '1' and bit 6 was set to '0' then the trigger level is set to 8 bytes. When there is 8 bytes of data in the receive FIFO then the Received Data Available interrupt is set. See (IIR)

Line Control Register (LCR)

Bit 7	1	Divisor Latch Access Bit		
	0	Access to Receiver buffer, Transmitter buffer & Interrupt Enable Register		
Bit 6	Set Break Enable			
Bits 3, 4 And 5	Bit 5	Bit 4	Bit 3	Parity Select
	X	X	0	No Parity
	0	0	1	Odd Parity
	0	1	1	Even Parity
	1	0	1	High Parity (Sticky)
	1	1	1	Low Parity (Sticky)
Bit 2	Length of Stop Bit			
	0	One Stop Bit		
	1	2 Stop bits for words of length 6,7 or 8 bits or 1.5 Stop Bits for Word lengths of 5 bits.		
Bits 0 And 1	Bit 1	Bit 0	Word Length	
	0	0	5 Bits	
	0	1	6 Bits	
	1	0	7 Bits	
	1	1	8 Bits	

Table 10 : Line Control Register

The Line Control register sets the basic parameters for communication. Bit 7 is the Divisor Latch Access Bit or DLAB for short. We have already talked about what it does. (See DLAB?) Bit 6 Sets break enable. When active, the TD line goes into "Spacing" state which causes a break in the receiving UART. Setting this bit to '0' Disables the Break.

Bits 3,4 and 5 select parity. If you study the 3 bits, you will find that bit 3 controls parity. That is, if it is set to '0' then no parity is used, but if it is set to '1' then parity is used. Jumping to bit 5, we can see that it controls sticky parity. Sticky parity is simply when the parity bit is always transmitted and checked as a '1' or '0'. This has very little success in checking for errors as if the first 4 bits contain errors but the sticky parity bit contains the appropriately set bit, then a parity error will not result. Sticky high parity is the use of a '1' for the parity bit, while the opposite, sticky low parity is the use of a '0' for the parity bit.

If bit 5 controls sticky parity, then turning this bit off must produce normal parity provided bit 3 is still set to '1'. Odd parity is when the parity bit is transmitted as a '1' or '0' so that there is a odd number of 1's. Even parity must then be the parity bit produces and even number of 1's. This provides better error checking but still is not perfect, thus CRC-32 is often used for software error correction. If one bit happens to be inverted with even or odd parity set, then a parity error will occur, however if two bits are flipped in such a way that it produces the correct parity bit then an parity error will no occur.

Bit 2 sets the length of the stop bits. Setting this bit to '0' will produce one stop bit, however setting it to '1' will produce either 1.5 or 2 stop bits depending upon the word length. Note that the receiver only checks the first stop bit.

Bits 0 and 1 set the word length. This should be pretty straight forward. A word length of 8 bits is most commonly used today.

Modem Control Register (MCR)

Bit	Notes
Bit 7	Reserved
Bit 6	Reserved
Bit 5	Autoflow Control Enabled (16750 only)
Bit 4	LoopBack Mode
Bit 3	Aux Output 2
Bit 2	Aux Output 1
Bit 1	Force Request to Send
Bit 0	Force Data Terminal Ready

Table 11 : Modem Control Register

The Modem Control Register is a Read/Write Register. Bits 5,6 and 7 are reserved. Bit 4 activates the loopback mode. In Loopback mode the transmitter serial output is placed into marking state. The receiver serial input is disconnected. The transmitter out is looped back to the receiver in. DSR, CTS, RI & DCD are disconnected. DTR, RTS, OUT1 & OUT2 are connected to the modem control inputs. The modem control output pins are then place in an inactive state. In this mode any data which is placed in the transmitter registers for output is received by the receiver circuitry on the same chip and is available at the receiver buffer. This can be used to test the UARTs operation.

Aux Output 2 maybe connected to external circuitry which controls the UART-CPU interrupt process. Aux Output 1 is normally disconnected, but on some cards is used to switch between a 1.8432MHZ crystal to a 4MHZ crystal which is used for MIDI. Bits 0 and 1 simply control their relevant data lines. For example setting bit 1 to '1' makes the request to send line active.

Line Status Register (LSR)

Bit	Notes
Bit 7	Error in Received FIFO
Bit 6	Empty Data Holding Registers
Bit 5	Empty Transmitter Holding Register
Bit 4	Break Interrupt
Bit 3	Framing Error
Bit 2	Parity Error
Bit 1	Overrun Error
Bit 0	Data Ready

Table 12 : Line Status Register

The line status register is a read only register. Bit 7 is the error in received FIFO bit. This bit is high when at least one break, parity or framing error has occurred on a byte which is contained in the FIFO.

When bit 6 is set, both the transmitter holding register and the shift register are empty. The UART's holding register holds the next byte of data to be sent in parallel fashion. The shift register is used to convert the byte to serial, so that it can be transmitted over one line. When bit 5 is set, only the transmitter holding register is empty. So what's the difference between the two? When bit 6, the transmitter holding and shift registers are empty, no serial conversions are taking place so there should be no activity on the transmit data line. When bit 5 is set, the transmitter holding register is empty, thus another byte can be sent to the data port, but a serial conversion using the shift register may be taking place.

The break interrupt (Bit 4) occurs when the received data line is held in a logic state '0' (Space) for more than the time it takes to send a full word. That includes the time for the start bit, data bits, parity bits and stop bits.

A framing error (Bit 3) occurs when the last bit is not a stop bit. This may occur due to a timing error. You will most commonly encounter a framing error when using a null modem linking two computers or a protocol analyzer when the speed at which the data is being sent is different to that of what you have the UART set to receive it at.

A overrun error normally occurs when your program can't read from the port fast enough. If you don't get an incoming byte out of the register fast enough, and another byte just happens to be received, then the last byte will be lost and a overrun error will result.

Bit 0 shows data ready, which means that a byte has been received by the UART and is at the receiver buffer ready to be read.

Modem Status Register (MSR)

Bit	Notes
Bit 7	Carrier Detect
Bit 6	Ring Indicator
Bit 5	Data Set Ready
Bit 4	Clear To Send
Bit 3	Delta Data Carrier Detect
Bit 2	Trailing Edge Ring Indicator
Bit 1	Delta Data Set Ready
Bit 0	Delta Clear to Send

Table 13 : Modem Status Register

Bit 0 of the modem status register shows delta clear to send, delta meaning a change in, thus delta clear to send means that there was a change in the clear to send line, since the last read of this register. This is the same for bits 1 and 3. Bit 1 shows a change in the Data Set Ready line where as Bit 3 shows a change in the Data Carrier Detect line. Bit 2 is the Trailing Edge Ring Indicator which indicates that there was a transformation from low to high state on the Ring Indicator line.

Bits 4 to 7 show the current state of the data lines when read. Bit 7 shows Carrier Detect, Bit 6 shows Ring Indicator, Bit 5 shows Data Set Ready & Bit 4 shows the status of the Clear To Send line.

Scratch Register

The scratch register is not used for communications but rather used as a place to leave a byte of data. The only real use it has is to determine whether the UART is a 8250/8250B or a 8250A/16450 and even that is not very practical today as the 8250/8250B was never designed for AT's and can't hack the bus speed.

Parts 3 and 4

Interfacing the Serial / RS232 Port - Part 3 and 4

Interfacing the PC - Homepage http://www.geocities.com/SiliconValley/Bay/8302

Information on :

Parallel Ports - SPP, EPP & ECP
Serial / RS232 Ports - UARTS 16450, 16550
Interrupts and more . . . .